ESCOLA TÈCNICA SUPERIOR ENGINYERIA
UNIVERSITAT ROVIRA I VIRGILI
Departament d'Enginyeria Electrònica Elèctrica i Automàtica

Lector de Etiquetas Pasivas de RFID

TITULACIÓN: Ingeniería Técnica Industrial en Electrónica Industrial

AUTOR: Óscar Aragón Andreu
DIRECTOR: José Luís Ramírez Falo

FECHA: diciembre del 2009

Índice General

Índice de Figuras

Índice de Tablas

Acrónimos:

AC/DC, Alternating Current/Direct Current
EAS, Electronic Article Surveillance
EPC, Electronic Product Code
FIFO, First Input First Output
FW, Firmware
HF, High Frequency
HW, Hardware
IC, Integrated Circuit
I/O, Input/Output
ISO, International Organization for Standardization
LF, Low Frequency
MISO, Master Input Slave Output
MOSI, Master Output Slave Input
PDIP, Plastic Dual In-line Package
PWR, Power (Periférico de Alimentación)
QFN, Quad Flat No leads
RF, Radio Frequency
RFID, Radio Frequency Identification
SO, Small Out-line Package
SPI, Serial Peripheral Interface
SS, Slave Select
SW, Software
UHF,Ultra High Frequency
USB, Universal Serial Bus
WORM, Write Once Read Many

1. Memoria Descriptiva

Lector de Etiquetas Pasivas de RFID

TITULACIÓN: Ingeniería Técnica Industrial en Electrónica Industrial

AUTOR: Óscar Aragón Andreu
DIRECTOR: José Luís Ramírez Falo

FECHA: diciembre / 2009

1.1. Objetivo

El objetivo principal del Proyecto consiste en diseñar y montar el prototipo de un lector de etiquetas pasivas de RFID.

Para alcanzar el objetivo se seguirán cuatro etapas:

- **Estudio sobre la Identificación por Radiofrecuencia (RFID)**, permitirá conocer el funcionamiento, características, aplicaciones y normativas de los sistemas de RFID para implementarlos posteriormente.

- **Diseño del Hardware**, se realizará un estudio para seleccionar y elegir los componentes adecuados. Se definirán los periféricos necesarios, se realizarán los esquemas y se montará el circuito.

- **Diseño del Firmware**, se programará la lógica de control del Lector. Esta lógica gestionará la secuencia de lectura de las etiquetas y las comunicaciones entre el Lector y un PC. Por requisitos de diseño estas comunicaciones se realizarán a través del bus de datos estándar USB.

- **Diseño del Software**, se creará una aplicación software que permita al usuario el control del sistema y la gestión de la información. Para ello se creará una interfaz de usuario visual que funcionará bajo un entono de Windows.

1.2. Estudio sobre la Identificación por Radiofrecuencia (RFID)

1.2.1. Introducción

En la actualidad, la tecnología más extendida para la identificación de objetos es la de códigos de barras. Sin embargo, éstos presentan algunas desventajas como son la escasa cantidad de datos que pueden almacenar, la imposibilidad de ser modificados una vez imprimidos y la necesidad de crear una vía de visión directa entre el lector y el código de barras.

La mejora que se ideó consistía en usar chips de silicio que pudieran transferir los datos almacenados al lector mediante ondas de radio, sin precisar así de la vía de visión directa. Ésta idea dio paso al nacimiento de la Identificación por Radio Frecuencia. Esta tecnología no sólo es aplicable al etiquetado de productos, ya se está empleando en muchas otras situaciones como los chips de identificación que llevan las mascotas bajo la piel, en sistemas de acceso, en la facturación de equipajes, para evitar la falsificación de moneda, etc. Incluso existen empresas como *Applied Digital Systems*, que defienden la implantación de estos chips bajo la piel de todos los ciudadanos, como un método de identificación personal infalible, imposible de robar o de perder.

Desafortunadamente, implementar un sistema con mayor complejidad y potencia resulta más caro y es precisamente su precio el obstáculo que está frenando su implantación. Los expertos calculan que habrá que esperar aún entre cuatro y seis años para lograrla, ya que una etiqueta RFID cuesta hoy entre 20 céntimos y 50 euros, eso sin contar el precio de los lectores. Otro inconveniente para su popularización es que hay demasiados sistemas RFID distintos, siendo necesario encontrar un estándar lo antes posible. Estos motivos son los causantes de que se esté llevando a cabo una transición lenta, en la que los códigos de barras conviven con las etiquetas de radiofrecuencia, empleadas únicamente en determinados productos o aplicaciones.

1.2.2. Breve Historia

Actualmente, la tecnología RFID ha acaparado un elevado interés por su posible implantación masiva, creando la sensación de que es nueva. Como podremos observar a continuación esta tecnología es más antigua de lo que pensamos. Antiguamente había conocimiento sobre varios campos de la radiación magnética o eléctrica, pero no es hasta el 1860 cuando se estudia la comunicación por radiofrecuencia. A continuación se destacan los años y acciones claves para la RFID:

- **En 1860:** *James Clerk Maxwell*, físico escocés, predijo la existencia de ondas radio y postulo su uso.

- **En 1886:** un científico alemán, *Heinrich Rudolf Hertz*, probó que rápidas variaciones de corriente eléctrica podían ser proyectadas en el espacio en forma de ondas de radio y que éstas eran medibles y repetibles.

- **En 1902:** *Guglielmo Marconi*, físico italiano, demostró la primera comunicación de larga distancia usando ondas de radio. Éstas atravesaron el Atlántico y la transmisión consistió en enviar un SOS en código Morse.

- **Durante la II Guerra Mundial (1942):** los británicos desarrollaron el primer sistema de etiquetado de RFID. El objetivo era discriminar rápidamente entre su propia flota de aviones y los escuadrones alemanes. Los aviones británicos incorporaban tags que contestaban a un lector con un código "I am a friend".

- **En 1960:** la necesidad de seguridad en los materiales nucleares condujo al desarrollo de una etiqueta RFID como el EAS.

- **En 1977:** la tecnología desarrollada se transfirió al sector público. En ese momento se produjo la aparición de aplicaciones más allá del simple EAS (utilizado como antirrobo), como llaves sin contacto, etc.

- **A partir de 1980:** se focaliza en la comercialización de la RFID. Nuevas aplicaciones, mejoras en el comportamiento y en el coste de lectores, etiquetas y antenas.

- **En el 2000:** *Auto-ID Center* focaliza todos sus esfuerzos en el desarrollo de la tecnología RFID. Posteriormente se convierte en *EPC global*, que gestiona y desarrolla estándares. Su intención es proporcionar un sustituto al código de barras.

1.2.3. Qué es

Como se verá a continuación, la RFID es una tecnología de identificación automática similar a la tecnología de código de barras. Su particularidad consiste en utilizar una señal de radiofrecuencia en lugar de una señal óptica.

Los sistemas de código de barras utilizan un lector y etiquetas impresas. El lector adquiere los datos, de forma unidireccional, mediante el reflejo de un rayo láser sobre el código de barras.

En cambio, la RFID utiliza un lector y una etiqueta especial. La comunicación se realiza de forma bidireccional por medio de una señal de radiofrecuencia de baja potencia. Esta señal de radio no requiere que la etiqueta esté dentro de la línea visual del lector, ya que las señales de radio pueden propagarse fácilmente a través de materiales no conductores. Por esto, la etiqueta de RFID no tiene porque estar en contacto directo con el lector.

1.2.4. Qué Componentes la Forman

En términos simples, un sistema de RFID está compuesto de los siguientes componentes:

- **Tag**: también llamado transpondedor, etiqueta inteligente, etc. Es el dispositivo contenedor de la información y está formado básicamente por un chip semiconductor y una antena. Normalmente un sistema de RFID está formado por varios tags.

- **Antena**: usada como transductor para transmitir las señales de RF entre el lector y el tag.

- **Lector**: es el dispositivo encargado de la comunicación. Envía y recibe las transmisiones RF del tag y proporciona los datos al sistema servidor para su procesado.

- **Servidor**: técnicamente llamado Middleware. Permite al usuario la gestión de la información mediante una computadora y una aplicación software.

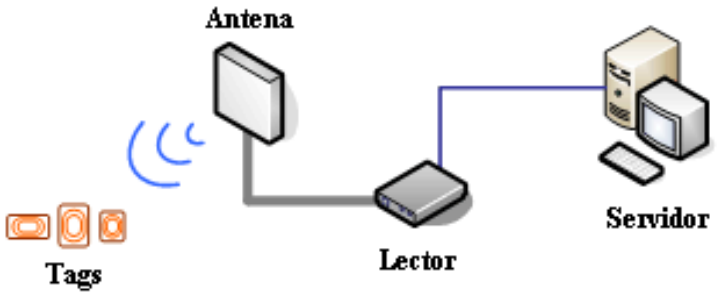


Figura 1. Componentes de un Sistema RFID

1.2.4.1. El Tag

Cuando el lector solicita información espera normalmente una respuesta de otro elemento para mantener la comunicación, en los sistemas RFID es el tag quien responde.

Generalmente podemos clasificar los tag según su fuente de energía, estructura, tipo de memoria y clase, pero la gran variedad que encontramos en el mercado nos podrían llevar a otras subdivisiones tales como los estándares que cumplen, su ciclo de vida, su tamaño, su alcance, etc… siendo este elemento el más difícil de decidir según la aplicación que vayamos a realizar.

- **Fuente de energía**: según la procedencia de la energía para poder activar el chip y enviar la información diferenciaremos tres tipos de tag: tags pasivos, tags semiactivos y tags activos.

  Los tags pasivos no requieren batería ya que toda la energía la recoge del campo electromagnético creado por la antena del lector. Son los más económicos y los de menor alcance en la comunicación, pero por su relación entre comportamiento y precio son los más utilizados. Por requisitos de diseño, éste será el tipo de tag que utilice el Lector.

  Los tags semiactivos utilizan una batería para activar la circuitería del chip, pero para la comunicación utilizan la energía que recogen de las ondas radio del lector (como en los pasivos). Debido a la utilización de batería son más grandes y caros que los pasivos, pero consiguen mejores alcances. Algunos tags llevan integrados sensores de temperatura, movimiento, etc... para proporcionar mayores funcionalidades.

  Los tags activos tienen su propia batería para el suministro de la energía. Dicha energía es utilizada para activar la circuitería del microchip y enviar la señal a la antena. Permiten una amplia cobertura de difusión, es decir, mayor alcance. Normalmente

tienen una mayor capacidad de almacenar información: contenido, origen, destino, procesos realizados, etc. También pueden llevar sensores adicionales de temperatura, de velocidad, de movimiento, etc… que permiten almacenar o controlar datos vitales en algunas aplicaciones. Uno de los ejemplos más conocidos de tags activos es el sistema *TeleTac*, para el pago en los peajes de las autopistas sin necesidad de parar.

- **Estructura**: un tag RFID pasivo esta compuesto por tres partes: el chip semiconductor, la antena y el sustrato. Destacaremos que esto sólo es cierto para el caso de los tag pasivos, debiendo añadir una batería a la estructura de los semiactivos y activos.

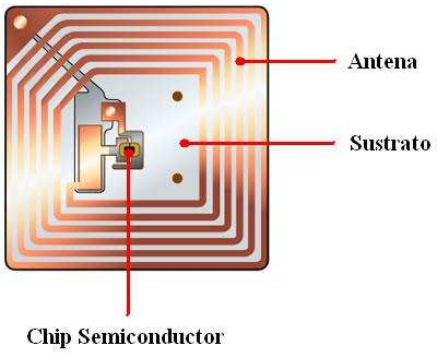


Figura 2. Estructura de un Tag RFID pasivo

El chip es un microcontrolador que almacena una serie de información y que contiene la lógica necesaria para poder responder a un lector. La mayoría de los tags pasivos deben cumplir sólo con la misión de matrícula de producto y tienen 96 bits de memoria (como el EPC), pero pueden tener una capacidad mayor.

La antena permite al chip recibir la energía procedente del lector y enviar/recibir la señal. El tamaño de la antena es crítico para el comportamiento del tag ya que normalmente determina el alcance de éste.

El sustrato es el material que mantiene el chip y la antena juntos y protegidos. En la mayoría de tag es un film plástico y tanto el chip como la antena quedan adheridos a él.

- **Tipo de memoria**: según el tipo de memoria que tiene el chip podemos diferenciar: sólo lectura, WORM y Lectura/Escritura programable.

  Sólo lectura: el identificador viene grabado de fábrica y tiene una longitud fija de caracteres.

  WORM: programable por el usuario una vez, pudiendo leer las veces que se quiera.

  Lectura/Escritura: programable una parte de la memoria, normalmente la de usuario, se puede grabar hasta 100.000 veces. Estos tags se utilizan para aplicaciones cerradas de la misma empresa en las que es necesaria la reutilización de los tags.

- **Clase:** el *EPC global*, como órgano de estandarización para la RFID en su uso con EPC, ha organizado las etiquetas en 6 clases:

  Clase 0: sólo lectura (el número EPC se codifica en la etiqueta durante el proceso de fabricación).

  Clase 1: escritura una sola vez y lecturas indefinidas (se fabrican sin número y se incorpora a la etiqueta más tarde).

  Clase 2: lectura y escritura.

  Clase 3: capacidades de la clase 2 más la fuente de alimentación que proporciona un incremento en el alcance y funcionalidades avanzadas.

  Clase 4: capacidades de la clase 3 más una comunicación activa con la posibilidad de comunicar con otras etiquetas activas.

  Clase 5: capacidades de la clase 4 más la posibilidad de poder comunicar también con etiquetas pasivas.

1.2.4.2. La Antena

Son los dispositivos que permiten radiar las señales de los lectores y recoger las ondas radio de los tags. Hay dos clases de antenas: antenas móviles y antenas fijas. La mayoría de veces también podemos catalogar los lectores con estos dos tipos.

- **Antenas móviles**: normalmente se encuentran integradas en lectores móviles o son utilizadas manualmente por un operario (tipo aspirador de tags).

- **Antenas fijas**: se conectan a los lectores mediante cables. Un único lector puede gestionar varias antenas creando, mediante en campo magnético generado, una zona de interrogación. Podemos encontrar ejemplos en las *Dock Door* (2 antenas) para puertas o en las de arco (3 antenas) para cintas transportadoras.

Figura 3. Antena Móvil

Figura 4. Antena Fija (Dock Door)

1.2.4.3. El Lector

Un lector RFID actúa como componente básico para la comunicación. Se encarga de enviar, recibir y convertir las señales que recibe del sistema.

A grandes rasgos, transmite/recibe señales de la antena que transforma en cadenas de bits de información digital. Finalmente vuelca los datos a un sistema servidor mediante varios tipos de interfaz como el RS-232, Ethernet, USB o Wireless.

Un lector RFID se compone de una caja de protección, una fuente de alimentación, la electrónica necesaria y opcionalmente de una o varias antenas. Hay multitud de tipos de lectores: simples (un sólo estándar y frecuencia), multiregionales, multifrecuencias, multiprotocolos, etc. La línea con mayor interés son los lectores ágiles y flexibles que pueden utilizar cualquier protocolo, región o frecuencia.

Figura 5. Lector de RFID

1.2.4.4. El Servidor

Normalmente los elementos de un sistema RFID no forman un sistema aislado, sino que se conectan a sistemas de producción, control, logística, etc. En esta fase entra el Servidor, dispositivo situado entre el hardware RFID y las aplicaciones software del cliente. Su función es la de gestionar todo el sistema RFID a nivel hardware, recibir las señales de los tags, filtrar la información y transmitir sólo la información útil.

1.2.5. Cómo Funciona

Debido a que el Proyecto trata sobre el diseño de un lector de etiquetas pasivas de RFID la explicación se centrará en el funcionamiento de un sistema RFID pasivo, con tags pasivos.

Una breve descripción del funcionamiento básico del sistema RFID pasivo sería:

- El lector solicita información al tag.
- El tag se activa cuando se encuentra dentro del campo de radiofrecuencia generado por la antena del lector y envía la información solicitada.
- La antena recibe la señal del tag.
- El lector convierte la señal y envía la información al sistema servidor.
- Finalmente, el sistema servidor envía la información recibida a la aplicación software para su gestión.

1.2.6. Frecuencias de Trabajo

Para empezar hay que tener en cuenta que la utilización del espectro de radio está condicionado a las normativas vigentes de cada uno de los países. Una clasificación global nos llevaría a diferenciar dos clases de bandas: las bandas licenciadas (de pago) y las bandas no licenciadas (libres). La tecnología RFID utiliza las bandas no licenciadas.

Los sistemas RFID utilizan diferentes frecuencias. La elección de éstas es un factor decisivo a la hora de implementar la aplicación, ya que tienen comportamientos diferentes. Generalmente las más utilizadas son la LF alrededor de los 125 KHz, la HF a 13,56 MHz y la UHF entre 860 y 960 MHz. Para algunas aplicaciones más específicas también se utiliza la banda de microondas a 2,45 GHz.

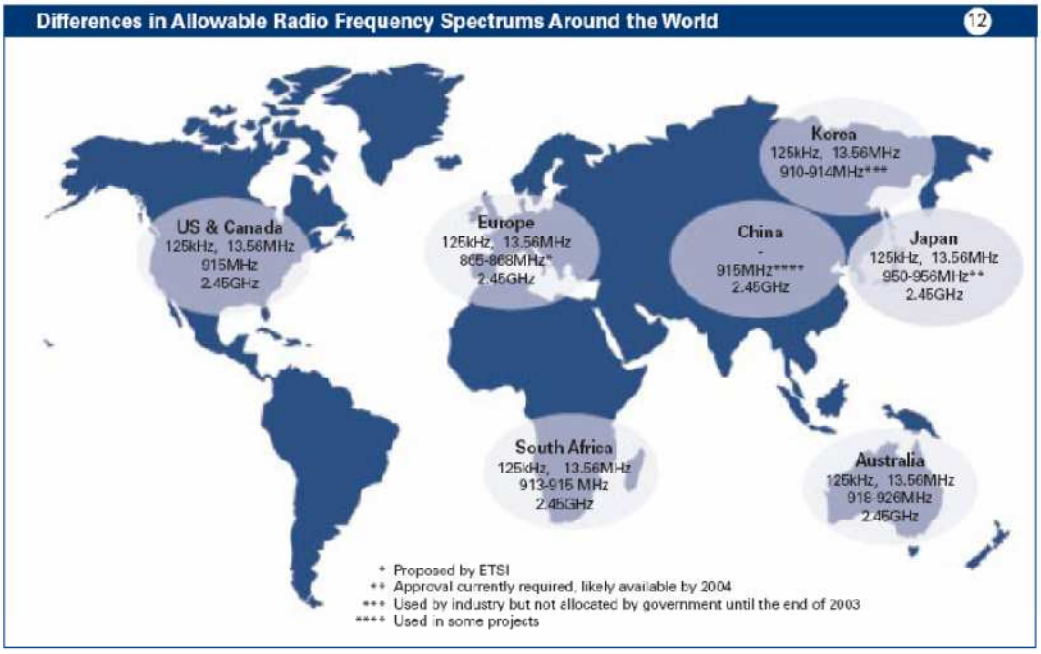


Figura 6. Frecuencias RFID en el mundo.

A continuación se muestra una comparativa entre las diferentes frecuencias utilizadas por los sistemas RFID:

Frecuencia	LF 125 KHz	HF 13,56 MHz	UHF 860 – 960 MHz	Microwave 2,45 GHz
Alcance de lectura	< 0,5 m	< 1 m	< 3 m	< 1 m
Características Generales	Caro, con baja degradación[1] pero lento	Precio medio, degradación y velocidad aceptables	Baratos a grandes cantidades, buena relación entre degradación y velocidad	Baratos a grandes cantidades, alta degradación y velocidad
Características del Tag	Gran tamaño, pasivos con acoplamiento inductivo	Tamaño medio, pasivos con acoplamiento inductivo o capacitivo	Tamaño pequeño, activos o pasivos con acoplamiento capacitivo	Tamaño muy pequeño, activos o pasivos con acoplamiento capacitivo
Aplicaciones Típicas	Control de acceso, trazabilidad de animales	Trazabilidad a nivel ítem, bibliotecas	Manejo de equipajes, trazabilidad de pallets	Logística, peajes

Tabla 1. Comparativa entre las frecuencias RFID

[1] Se entiende por la degradación que sufre la señal al pasar por materiales metálicos o líquidos.

1.2.7. Estándares

Los estándares o normalizaciones permiten disponer de soluciones comunes que pueden ser implementadas por diferentes fabricantes o integradores. Uno de los organismos internacionales más conocidos en el campo de la normalización es la ISO. Hay varios estándares según el tipo de aplicación. A continuación se hará una breve explicación de las normativas más relevantes en el entorno RFID.

Estándares desarrollados para etiquetas de identificación:

ISO/IEC 10536 Identification cards – Contactless integrated circuit cards: para etiquetas de identificación inteligentes a 13,56 MHz. Describe sus características físicas, dimensiones, localización de las áreas de interrogación, señales electrónicas, procedimientos de reset y la descripción del protocolo de transmisión.

ISO/IEC 14443 Identification cards – proximity integrated circuit cards: desarrollado para etiquetas de identificación inteligentes a 13,56 MHz con alcance superior a un metro. Describe las características físicas, el interfaz aéreo, la inicialización, anticolisión y el protocolo de transmisión.

ISO/IEC 15693 Contactless integrated circuit cards – Vicinity cards: se desarrollan las características físicas, la interfaz aérea y los protocolos de transmisión y anticolisión para etiquetas sin contacto en la banda de HF (13,56 MHz).

Estándares desarrollados para la gestión a nivel unidad:

ISO/IEC 15961 RFID for item management – Data protocol: application inteface: explica comandos funcionales comunes y características de sintaxis, tipos de tags, formatos de almacenamiento, compresión de datos, etc. Los estándares de interfaz aérea no afectan a este estándar.

ISO/IEC 15962 RFID for item management – Protocol: Data encoding rules and logical memory functions: dirigido a la gestión de la información a nivel unidad. Crea un formato de datos correcto y uniforme además de la estructura de comandos y el procesamiento de errores.

ISO/IEC 15963 for item management – Unique identification of RF tag: este estándar se dirige al sistema de numeración, el proceso de registro y el uso del tag. Se ha diseñado para el control de calidad durante el proceso de fabricación. También está dirigido a la trazabilidad de los tags durante este proceso, su ciclo de vida y el control para la anticolisión de varios tags en la zona de interrogación.

ISO/IEC 19762: Harmonized vocabulary – Part 3: radio-frequency identification: documento que proporciona términos generales y definiciones en el área de la identificación automática y las técnicas de captura de datos. La 3ª parte es la que hace referencia a la tecnología RFID.

ISO/IEC 18000 Air inteface standards: diseñada para crear una interoperabilidad global. Se define la comunicación entre los tags y los lectores a diferentes frecuencias de trabajo. El objetivo del estándar es asegurar un protocolo de interfaz aérea a nivel mundial.

ISO/IEC 18001 RFID for Item Management - Application Requirements Profiles: proporciona el resultado de tres estudios para identificar aplicaciones y los usos de la tecnología RFID en ellas. También incluye una clasificación de los tipos de tags según su aplicación y una breve explicación sobre los parámetros de distancia, número de tags dentro del campo de interrogación, etc.

EPC™ Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID: creado por *EPC global*, *EAN* (European Article Numbering) y *UCC* (Uniform Code Council). En este documento se desarrolla el estándar para el protocolo de interfaz aérea de comunicación entre el lector y el tag.

13.56 MHz ISM Band Class 1 Radio Frequency (RF) Identification Tag Interface Specification: desarrollado por *EPC global* para definir la interfaz de comunicación y el protocolo de la Clase 1 en HF (13,56MHz). Incluye los requerimientos de los tags y de los lectores para establecer comunicaciones en dicha banda de frecuencias.

Application Level Event (ALE) Specification Version 1.0: estándar desarrollado por *EPC global* que especifica un interfaz a través del cual se filtran y consolidas los códigos electrónicos EPC.

Hay otros estándares RFID, como reseña a algunos de éstos comentar que *AIAG* (Automotive Industry Action Group) ha desarrollado junto a *EPC global* estándares para la industria de la automoción. En específico el "Application Standard for RFID Devices in the Automotive Industry" que viene acompañado por otros como "AIAG B - 11", estándar para identificar neumáticos y ruedas con RFID.

Para obtener mayor información consultar la web de la ISO y/o del *EPC global*.

1.2.8. Aplicaciones RFID

1.2.8.1. Clasificación General

Una forma muy genérica de clasificar las aplicaciones RFID sería según el ciclo que sufre el tag. Se distinguirán dos tipos: las aplicaciones de ciclo cerrado y aplicaciones de ciclo abierto.

- **Aplicaciones de ciclo cerrado:** son aquellas donde el tag se recupera al final del proceso para volver a introducirlo en su inicio. En este tipo de aplicaciones el precio del tag no influye demasiado, ya que al reutilizarse su precio es casi despreciable.

- **Aplicaciones de ciclo abierto:** el tag se pierde al finalizar el proceso, bien porque no se puede aprovechar o porque el producto es traspasado a otro agente de la cadena. En este tipo de aplicaciones el precio del tag es muy importante.

1.2.8.2. Aplicaciones comunes

Hay una gran variedad de aplicaciones RFID. A continuación se mostrarán algunas de las más comunes.

- **Control de acceso:** es una de las aplicaciones más utilizadas en el mundo de la RFID. Su objetivo es permitir, o no, el acceso a una zona determinada. La banda utilizada en estas soluciones ha sido la LF, pero actualmente se está empezando a utilizar la banda de HF (13,56 MHz).

- **Cadena de suministro:** es en esta aplicación donde se centran los esfuerzos del *EPC Global*. El objetivo es identificar cada uno de los productos, mediante tecnología RFID, y sustituir al código de barras. El tag se situará en cada uno de los productos desde su fabricación hasta su venta.

- **Logística del frío:** este es un caso particular, donde entran en acción los tag semiactivos con sensores incorporados. Su aplicación básica es controlar que el transporte de la mercancía sea el correcto y que no se rompa la cadena de frío del producto.

- **Peaje automático:** es posiblemente la aplicación más conocida por los usuarios finales, como el *TeleTac* o *ViaT*. El cliente tiene un dispositivo, que realmente es un tag RFID activo, y cuando pasa por el peaje el sistema cobra y abre las puertas sin necesidad de parar. Estos sistemas también son utilizados para el control de entradas de parking y en algunas estaciones de servicio.

- **Control de producción o calidad:** muchas empresas, sobretodo en el sector de la automoción, han encontrado en la tecnología RFID una solución de control a sus complejos procesos. El sistema actúa proporcionando información sobre el estado del proceso en tiempo real.

- **Librerías y servicios de alquiler:** algunas librerías y servicios de alquiler han encontrado en la RFID la nueva codificación de sus libros, cd's, etc... gracias a su capacidad de incorporar un código único, información adicional y bit de seguridad (parecido al EAS) en un único sistema de identificación.

- **Servicios postales:** algunas empresas de servicios postales están viendo la tecnología RFID como la solución para automatizar sus procesos de gestión de envíos. Esta automatización aportaría mayor rapidez, seguridad y precisión, características que impactan directamente sobre el servicio al cliente.

- **Ticketing:** las empresas de transporte utilizan billetes con tecnología RFID para ahorrar costes, automatizar procesos y facilitar el acceso a sus clientes.

1.2.9. Beneficios

La tecnología RFID es flexible y se puede adaptar a nuestras necesidades fácilmente. A grandes rasgos nos proporciona:

- Mayor velocidad e información.
- Mejora la eficiencia y la flexibilidad.
- Permite mejorar la seguridad.
- Mejora la automatización de los procesos.

Si realizamos una breve comparación con el código de barras, otra tecnología de identificación automática, obtenemos las siguientes ventajas:

- No necesita línea de visión directa.
- Puede almacenar mayor cantidad de información, hasta 8 kBytes [1].
- Permite leer y escribir.
- Permite lecturas simultáneas.
- Combina funciones de identificación y de autentificación.
- Soporta condiciones más adversas (temperatura, suciedad, etc.)

1.2.10. Conclusiones sobre el Estudio de la RFID

El estudio realizado anteriormente permite seleccionar y elegir las características principales del sistema a diseñar. El lector de etiquetas pasivas de RFID cumplirá los siguientes requisitos:

- **Frecuencia de trabajo:** se elije la banda de HF a 13,56 MHz. Esta frecuencia permitirá tener un sistema con una relación de precio, alcance, degradación de la señal y velocidad aceptables. Además, es una frecuencia muy común en el mundo.

- **Normativa:** el comportamiento del lector se ajustará a la normativa *ISO/IEC 15693 Contactless integrated circuit cards – Vicinity cards.* Esta normativa cumple los requisitos de alcance, banda y tipo de tag que se usarán.

- **Aplicación:** el uso final se destinará a un sistema de control de acceso, por ser una de las aplicaciones más utilizadas. Además, es un una aplicación de ciclo cerrado que permitirá la reutilización de los tags, despreciando así su coste.

- **Antena:** en esta versión del Lector no se implementará la antena, ya que no entra dentro de la definición de los objetivos del Proyecto. De todas formas, se deberá tener en cuenta que el sistema RFID no estará completo sin ella y por este motivo se incluirá en los diagramas de representación del sistema.

1.3. Diseño del Hardware

1.3.1. Componentes Necesarios para el Diseño del Lector

Para el diseño del Lector se necesitará un transceiver de RFID, un microcontrolador, una fuente de alimentación y los componentes electrónicos necesarios.

- **Transceiver de RFID**: es un circuito integrado. Actualmente no hay una gran oferta y además no resulta fácil adquirirlo a nivel unidad. Es el nexo ente el microcontrolador y la antena. Su función es la de adaptar la información del tag de forma que el microcontrolador pueda descodificarla y viceversa. La elección de éste dependerá de la aplicación en que se quiera usar, pero deberá utilizar el mismo protocolo de transmisión que el tag a utilizar.

- **Microcontrolador**: también es un circuito integrado. A diferencia del transceiver es un componente muy común y fácil de adquirir. Para su elección se tendrán en cuenta las siguientes características: tamaño del bus de datos, cantidad de memoria, puertos/buses de comunicación y velocidad de procesado. También es importante conocer qué herramientas puede suministrar el proveedor, así como documentación, librerías, ejemplos, software de diseño, etc.

- **Fuente de Alimentación**: para simplificar el diseño se implementará una fuente de alimentación externa. Se usará un transformador AC/DC por ser una alternativa barata y fácil de aplicar. De todas formas, el Lector dispondrá de un regulador de tensión propio. Esta solución facilitará la elección del transformador, ya que los reguladores de tensión admiten un rango amplio en su tensión de entrada. Además, proporcionará una alimentación estable y filtrada a los dispositivos principales. Un regulador de tensión es un componente muy habitual y se puede adquirir en cualquier tienda de electrónica, por lo que su elección se deberá a su disponibilidad. Sus características deberán satisfacer las necesidades de tensión e intensidad requeridas por el sistema.

- **Componentes Electrónicos:** para que los dispositivos anteriores funcionen correctamente será necesario complementarlos con diversos componentes: resistencias, condensadores, cristales de cuarzo, etc. Actualmente hay una gran variedad de proveedores que suministran componentes de iguales características, por lo que su elección se basará en las especificaciones de los compontes principales o en los cálculos realizados.

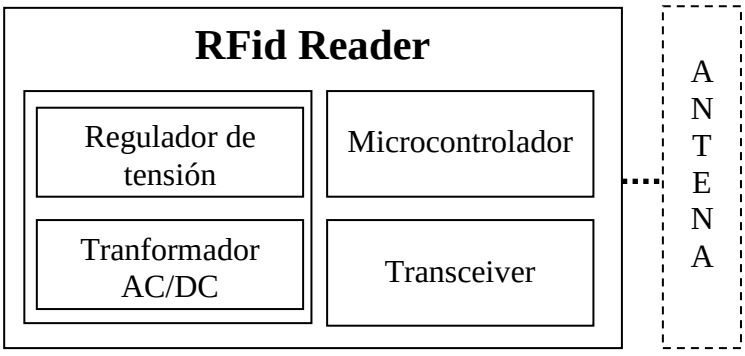


Figura 7. Diagrama General HW del Lector

1.3.2. Selección de los Componentes

Para simplificar la selección del Transceiver y del Microcontrolador se realizará un breve estudio que comparará los componentes de tres proveedores de electrónica: *Texas Instruments*, *NXP Semiconductors* y *Microchip Technology Inc*. De cada proveedor se seleccionarán uno o dos componentes, los que más se ajusten a los requisitos de diseño propuestos hasta el momento. Finalmente se elegirán los componentes más adecuados.

1.3.2.1. Selección del Transceiver de RFID

El Transceiver será el primer componente a seleccionar. Es una parte fundamental en el Lector y el resto de componentes deberán ajustarse a sus necesidades.

A continuación se muestra una tabla comparativa que relaciona al componente con sus proveedores. Se deberá tener en cuenta que todos ellos disponen de un bus de datos de 8 bits.

Texas Instruments

Dispositivo	Estándar	Frecuencia	Tensión de Alimentación (Vdc)	Consumo de Corriente (mA)	Potencia de Transmisión (mW)	Buses/Puertos de Comunicación	Encapsulado
TRF7960	ISO 14443A ISO 14443B ISO 15693 ISO 18000-3	13,56 MHz	2,7 – 5,5	10	200	EPP SPI	32QFN
TRF7961	ISO 15693 ISO 18000-3	13,56 MHz	2,7 – 5,5	10	200	EPP SPI	32QFN
Herramientas: Documentación y entrenador (199 $).							

NXP Semiconductors

Dispositivo	Estándar	Frecuencia	Tensión de Alimentación (Vdc)	Consumo de Corriente (mA)	Potencia de Transmisión (mW)	Buses/Puertos de Comunicación	Encapsulado
CL RC632	ISO 14443A ISO14443B ISO 15693 MIFARE I·CODE	13,56 MHz	3,3 - 5,5	200	750	EPP SPI	SO32
SL RC400	ISO 15693 I·CODE	13,56 MHz	3,3 - 5,5	200	750	EPP SPI	SO32
Herramientas: Documentación, código de ejemplo y librerías para MF RC5xx (o dispositivo compatible).							

Microchip Technology Inc.

Actualmente está suministrando microcontroladores con transmisores RF incorporados, pero utilizan la banda de UHF y hay que utilizar un receptor RF externo, por lo que éste proveedor queda descartado.

Tabla 2. Selección del Transceiver de RFID

1.3.2.2. Selección del Microcontrolador

Como requisito de diseño se tiene que el Lector deberá ser capaz de comunicarse con una computadora mediante el bus de datos USB. Esto simplifica en gran medida la selección del componente, ya que sólo se seleccionarán dispositivos que tengan integrado dicho periférico. Se destacará que, siendo la intención diseñar un prototipo, siempre se seleccionarán dispositivos que ofrezcan la mayor cantidad de memoria posible.

Texas Instruments

Actualmente está suministrando microcontroladores con transceivers USB integrados (basados en la serie 805x MCUs), pero sólo disponen del bus I^2C para comunicarse con otro dispositivo. Ésta característica complicaría el diseño del Lector, ya que los transceivers RFID seleccionados no disponen de dicho bus.

NXP Semiconductors

Dispositivo	Bus de Datos	Frecuencia Oscilación máx. (MHz)	Memoria	Buses/Puertos de Comunicación	Puertos I/O	Encapsulado
LPC2148	32-bit 16-bit	60	Flash 512 kB RAM 32 kB	I^2C SPI SPP UART USB 2.0	2 de32-bit 4 de 16-bit	LQFP64

Herramientas: Documentación demostrativa.

Microchip Technology Inc.

Dispositivo	Bus de Datos	Frecuencia Oscilación máx. (MHz)	RAM/Flash (kB)	Buses/Puertos de Comunicación	Puertos I/O	Encapsulado
PIC18F2550	8-bit	48	Flash 32 kB RAM 2 kB E^2PROM 256 bytes	I^2C SPI EAUSART USB 2.0	4 de 8-bit	PDIP

Herramientas: Documentación completa, Software de Diseño gratuíto, ejemplos y librerías. Se destacará la sencillez y la facilidad que ofrece su web, esto facilita la localización de dispositivos y herramientas.

Tabla 3. Selección del Microcontrolador

1.3.3. Elección de los Componentes

1.3.3.1. Elección del Transceiver de RFID

Para la elección del transceiver se hará una pequeña comparación entre las características más relevantes de los componentes seleccionados. No se tendrán en cuenta la tensión de alimentación ni el consumo, puesto que no son características críticas para el diseño. Tampoco los diferentes puertos de comunicación por ser los mismos en todos los ellos.

Dispositivo	Estándar (Nivel de multiprotocolo)	Potencia	Encapsulado (Dificultad en la soldadura)	Herramientas suministradas por el Proveedor
TRF7960	Alto	Baja	Difícil	Suficientes
TRF7961	Suficiente	Baja	Difícil	Suficientes
CL RC632	Muy Alto	Alta	Normal	Bastantes
SL RC400	Suficiente	Alta	Normal	Bastantes

Tabla 4. Elección del Transceiver de RFID

El *SL RC400* reúne unas características óptimas, sigue la normativa *ISO/IEC 15693* y su soldadura en el prototipo no será excesivamente complicada. De todas formas, se elige el *CL RC632* de *NXP Semiconductors* por los siguientes motivos:

- **Estándar:** cumple el requisito de comunicación bajo el protocolo de transmisión *ISO 15693*, pero además ofrece la posibilidad trabajar con otros protocolos. Esta característica ofrecerá la posibilidad de obtener un lector multiprotocolo mucho mejor, por si se quisieran aplicar futuras ampliaciones.

- **Potencia:** es uno de los dispositivos que ofrece mayor potencia en la transmisión, aumentando así el alcance de lectura. Por el contrario su consumo será más elevado, pero no se considera un problema, puesto que se elegirá una fuente de alimentación que proporcione la potencia necesaria.

- **Encapsulado:** el componente se implementará en un prototipo montado a mano. El encapsulado *SO32* facilitará esa tarea frente al *32QFN*.

- **Herramientas:** el proveedor ofrece más herramientas en comparación al resto.

1.3.3.2. Elección del Microcontrolador

Se elegirá el microcontrolador siguiendo las mismas pautas utilizadas para la elección del transceiver. En este caso tampoco se tendrán en cuenta los puertos de comunicación ya que ambos dispositivos pueden comunicarse con el transceiver elegido mediante el puerto SPI.

Dispositivo	Bus de Datos	Frecuencia de Oscilación	Cantidad de Memoria	Encapsulado (Dificultad en la soldadura)	Herramientas suministradas por el Proveedor
LPC2148	Suficiente	Alta	Mucha	Muy Difícil	Pobres
PIC18F2550	Suficiente	Buena	Suficiente	Muy Fácil	Excelentes

Tabla 5. Elección del Microcontrolador

En el caso del microcontrolador se necesita, como mínimo, un dispositivo que asegure y aproveche al máximo el rendimiento que pueda ofrecer el transceiver. Si se elige un dispositivo superior se complicará el diseño, se aumentará el coste y se desaprovecharán las características no útiles.

En este caso, el *PIC18F2550* de *Microchip Technology Inc.* resulta la mejor elección:

- **Bus de datos:** es suficiente ya tiene el mismo tamaño que el bus de datos del transceiver.

- **Frecuencia de oscilación:** puede llegar a ofrecer una velocidad de procesado hasta 3,5 veces superior a la del transceiver. Esto evitará tener que preocuparse de diseñar un firmware óptimo en temporización. A su vez ofrecerá la posibilidad de aplicar un control mayor y/o características adicionales.

- **Cantidad de memoria**: es una característica difícil de elegir a priori, ya que no se sabe cuánta memoria se va a necesitar. Para el diseño de un prototipo es normal elegir la mayor cantidad posible. En este caso se han consultado Proyectos similares y se ha comprobado que los 32 kB de memoria deberían ser suficientes.

- **Buses de comunicación:** dispone del bus USB, necesario por requisitos de diseño. También dispone del bus SPI, uno de los buses integrados en el transceiver de RFID elegido. Esto facilitará la implementación de las comunicaciones entre el microcontrolador y el transceiver.

- **Encapsulado:** el encapsulado PDIP es muy fácil de soldar y se puede acoplar a un zócalo fácilmente.

- **Herramientas:** el proveedor ofrece gratuitamente mucha documentación y herramientas de diseño. Además, es posible adquirir un programador compatible a un coste muy bajo.

1.3.3.3. Elección del la Fuente de Alimentación

Como se ha comentado, la fuente de alimentación del Lector estará compuesta por un transformador AC/DC y un regulador de tensión. Éste será del tipo *L7805CV*, por motivos de disponibilidad.

Según los cálculos, las características de alimentación serán:

- Tensión de entrada al regulador $7{,}5\ V < Vin < 35\ V$
- Tensión de alimentación digital $Vdd = 5\ V$
- Corriente de alimentación $Idd = 450\ mA$

A partir de ahora se denominará Periférico de Alimentación o PWR a este conjunto.

1.3.4. Caracterización de los Periféricos utilizados

En este apartado se destacarán las características de cada periférico, haciendo hincapié sólo en las utilizadas en el Lector. Para obtener información más detallada consultar las Hojas de Características de cada dispositivo u otras especificaciones.

1.3.4.1. La Memoria

Para almacenar y gestionar los datos se necesita el periférico de memoria integrado en el microcontrolador y en el transceiver.

Se diferenciarán dos tipos de memoria:

- **Memoria volátil o de datos:** almacenará datos temporalmente. Estos datos se perderán cuando el sistema deje de estar alimentado.

- **Memoria no volátil o de programa:** almacenará la información de configuración y el programa de control. Esta información se mantendrá aun sin alimentación.

Debido a las características de los componentes elegidos, el Lector dispondrá básicamente de:

- Memoria de datos, repartida entre los bloques: **SRAM** de 2 kB [2], **USBRAM** de 1kB [3] y **FIFOBuffer** de 64 Bytes [4].

- Memoria de programa **Flash** de 32 kB [5] y una **E^{2}PROM** de 32 Bytes [6].

1.3.4.2. El Sistema de Oscilación

El sistema de oscilación establece la velocidad a la que se ejecuta cada instrucción y la tasa de transferencia de datos para las comunicaciones. Es importante ajustar bien las frecuencias cuando se pretende implementar un sistema de comunicación entre varios dispositivos. Un error en los cálculos podría provocar que las comunicaciones no funcionaran.

Debido a la naturaleza específica del Lector se implementarán dos sistemas de oscilación, uno para el transceiver y otro para el microcontrolador.

El sistema de oscilación del transceiver deberá ajustarse a las características indicadas en el manual del *CL RC632* [7].

El sistema de oscilación del microcontrolador seguirá las especificaciones para que el bus USB pueda funcionar [8].

En estas condiciones, se obtendrá un Lector con las siguientes características:

- Frecuencia del oscilador del *CL RC632*, $f_{osc\_clrc632} = 13{,}56\ MHz$.
- Frecuencia del Oscilador Primario del PIC, $f_{osc\_pic} = 20\ MHz$
- Frecuencia de trabajo del reloj de la CPU, $f_{cpu} = 24\ MHz$
- Frecuencia del bus SPI, $f_{spi} = 6\ MHz$.
- Frecuencia del bus USB, $f_{usb} = 6\ MHz$. Configuración LOW-SPEED.

El resultado es un sistema dónde el microcontrolador es más rápido que el transceiver, esto permitirá poder llevar un control adecuado sobre él.

1.3.4.3. El Timer

Normalmente los dispositivos llevan integrados uno o varios periféricos Timer que hacen a su vez de contador o temporizador. Estos Timer se utilizan básicamente para crear eventos periódicos y contar tiempo.

Es posible implementar un Timer mediante firmware. Esto sería, sabiendo el tiempo de ciclo de cada instrucción y creando un bucle de espera. Es alternativa es muy configurable, pero resulta más sencillo realizar los cálculos si se utiliza el periférico integrado en el microcontrolador. Además, permite un aprovechamiento mayor de las características del mismo.

Así pues, el Lector dispondrá de un periférico Timer para contar el tiempo de la fase *Start Up* del *CL RC632*. Según las especificaciones, este tiempo deberá ser preferiblemente mayor a $t_{Start\ Up\ Phase} = 47{,}20\ \mu s$ [9].

El *PIC18F2550* dispone de cuatro periféricos Timer, siendo el Timer0 [10] el que mejor se ajusta a las necesidades de diseño actuales: permite implementar un tiempo de espera configurable y activa un flag al finalizar su cuenta, útil para comprobar el evento. A su vez, dispone de dos modos de funcionamiento: 8-bit y 16-bit. En la fase de diseño del Firmware se decidirá el modo a usar.

1.3.4.4. Los Puertos I/O

Los Puertos I/O son los periféricos de entrada y salida de los dispositivos a nivel bit. Actualmente se intenta disminuir el tamaño de los dispositivos a la vez que se aumentan sus prestaciones. Esto provoca que los pines I/O estén multiplexados y sea necesaria su configuración.

El Lector usará las siguientes señales digitales:

Standard I/O					
Señal	**Descripción**	**PIC Pin#**	**Tipo**	**CL RC632 Pin#**	**Tipo**
!MCLR	Reset General	1	I	-	-
CLRC632_RSTPD	Reset del *CL RC632*	2	O	31	I
ERROR_LED	Led de Error	4	O	-	-
TX_LED	Led de Transmisión	5	O	-	-
RX_LED	Led de Recepción	6	O	-	-
!CLR_ERROR_SW	Limpieza del error	7	I	-	-
SIM_MODE_SW	Modo Normal/!Simulado	11	I	-	-

SPI I/O					
Señal	**Descripción**	**PIC Pin#**	**Tipo**	**CL RC632 Pin#**	**Tipo**
!CLRC632_SPI_SS	SPI Slave Select	3	O	21	I
SPI_SDO	SPI MOSI	18	O	22	I
SPI_SDI	SPI MISO	21	I	13	O
SPI_SCK	SPI Clock	22	O	24	I

USB I/O					
Señal	**Descripción**	**PIC Pin#**	**Tipo**	**CL RC632 Pin#**	**Tipo**
D-	USB D-	15	D-	-	-
D+	USB D+	16	D+	-	-

Tabla 6. Pines I/O

Se destacará que el *Puerto B* del *PIC18F2550* se usa para la gestión de interrupciones externas [11]. En esta versión del Lector no se utilizarán, pero se dejaran libres para su uso en futuras ampliaciones.

1.3.4.5. El Bus SPI

El bus SPI es un estándar de comunicaciones serie usado para la transferencia de información entre circuitos integrados. Se usará el SPI por ser el bus que está disponible tanto en el PIC como en el *CL RC632*.

La conexión entre los dispositivos se realiza configurando uno de ellos como Master y el otro, u otros, como Slave/s. En el Lector, el Master será el PIC y el Slave el *CL RC632*.

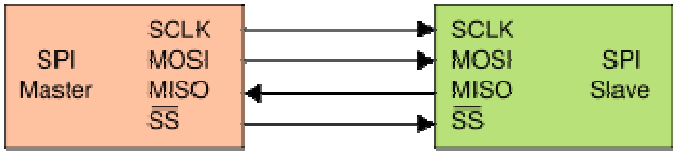


Figura 8. Comunicación SPI

El bus se compone de cuatro señales:

- **SCLK**, es la línea del reloj. La controla el Master y establece el ciclo de lectura y escritura. El límite de velocidad en la transferencia de datos viene dado por el fabricante del dispositivo y/o la configuración adoptada.

- **MOSI**, por esta línea se transmiten datos del Master al Slave.

- **MISO**, por esta línea se transmiten datos del Slave al Master.

- **!SS**, es la línea de selección, la controla el Master e indica que se va a iniciar una transmisión.

Hay cuatro estándares para la transferencia de información [12]. El Lector se deberá configurar con el MODE 1:1, por ser el que se ajusta a las especificaciones del *CL RC632* [13].

La información recibida deberá almacenarse en la memoria volátil del Microcontrolador.

1.3.4.6. El Bus USB

El bus de datos USB es un bus estándar que sirve para conectar periféricos a una computadora.

El estándar incluye la transmisión de energía eléctrica para dispositivos de bajo consumo (5V, 100 mA). La conexión entre los dispositivos USB deberá realizarse mediante un conector del tipo estándar [14]. En el caso del Lector se usará el TYPE A, por disponibilidad.

La alimentación del bus será del tipo *Bus power only* [15] por su sencillez. Esto obligará a conectar un condensador de 220 nF ± 20 % entre la patilla 14 del microcontrolador y masa.

La transmisión de datos se realiza mediante un cable de par trenzado con impedancia característica de 90 Ω ± 15 %, cuyos hilos se denominan D+ y D-. Éstos, colectivamente, utilizan señalización diferencial para combatir los efectos del ruido electromagnético en enlaces largos.

La velocidad del bus USB del Lector será de 1,5 Mbit/s, configuración LOW-SPEED. Esto requiere que se conecte una resistencia de Pull Up de 1,5 kΩ ± 5% en la línea D-. Esta resistencia está disponible en el hardware interno del microcontrolador [16].

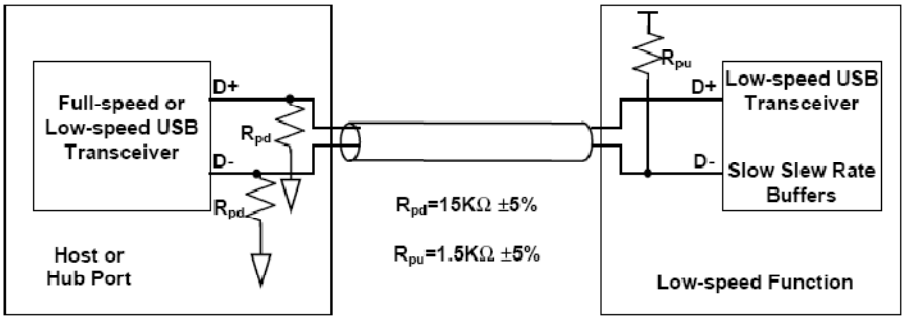


Figura 9. Comunicación USB

El *PIC18F2550* ofrece un Buffer de 1 kByte para el almacenamiento y gestión de los datos del periférico USB [17].

1.3.4.7. El Transmisor/Receptor

Para enviar y recibir datos al tag se necesita un sistema capaz de traducir los datos que el Lector envía y recibe. Estos datos se convertirán en señales eléctricas que excitarán una antena, permitiendo la comunicación entre el Lector y el tag. De este proceso se encargará el periférico Transmisor/Receptor, integrado en el *CL RC632* [18].

El *CL RC632* es un dispositivo que asegura el cumplimiento de la *ISO 15693-2*, se dará por sentado que las señales generadas se ajustan a la normativa [19].

1.3.5. Conclusiones sobre el Hardware

Los periféricos que integrarán el Lector seguirán el diagrama que se muestra a continuación:

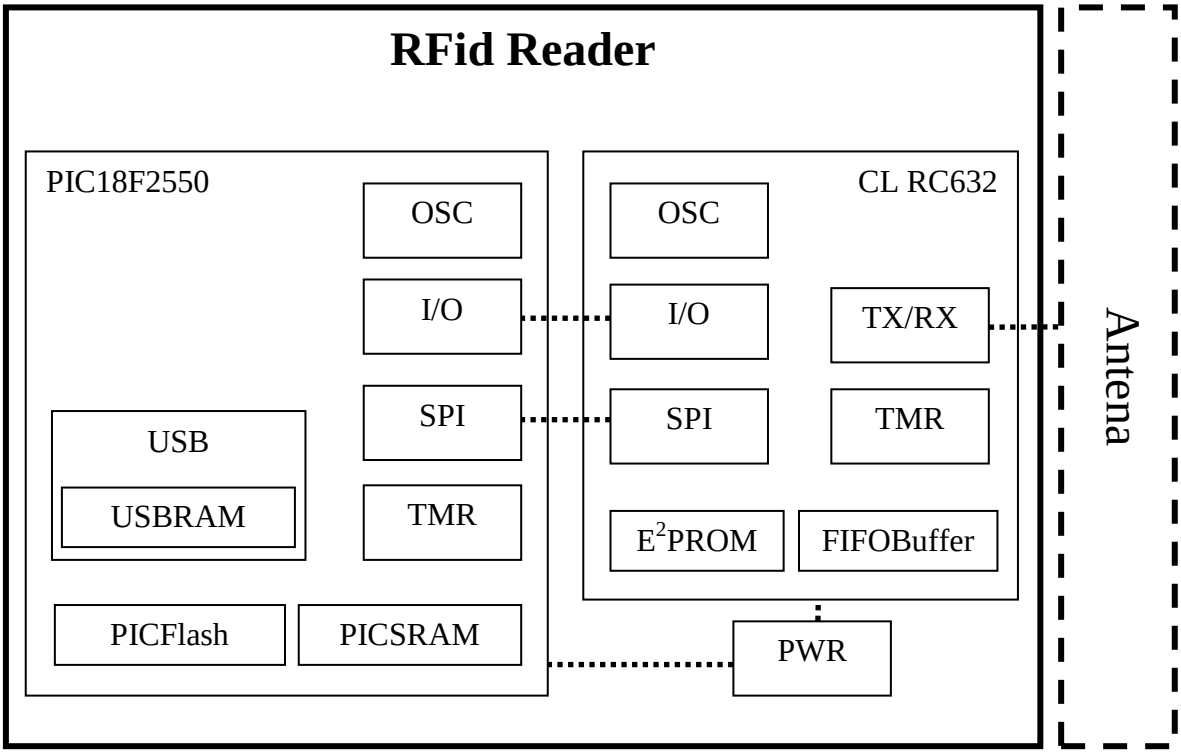


Figura 10. Diagrama Detallado HW del Lector

1.4. Diseño del Firmware

El Firmware, o programación en firme, es el programa que va grabado en una memoria no volátil. Éste establece la lógica que controla los circuitos electrónicos de un dispositivo. Se considera parte del hardware por estar integrado en la electrónica del dispositivo, pero también se considera software por estar escrito en algún tipo de lenguaje de programación.

1.4.1. Lenguaje de Programación

Un factor importante a la hora de diseñar el firmware es la elección del lenguaje de programación. Hoy en día son habituales los siguientes:

- **Ensamblador:** es un lenguaje de programación que permite realizar un código óptimo en velocidad de ejecución y en uso de memoria. Como desventaja, proporciona un nivel bajo de portabilidad y una complicación elevada en el diseño. Su uso sería justificable en aplicaciones de control sencillas en las que la velocidad de ejecución y el uso de memoria fueran un factor crítico.

- **Lenguaje C:** la facilidad en la programación y la portabilidad son sus puntos fuertes. Por el contrario, puede no cumplir con requisitos de velocidad en la ejecución de ciertas aplicaciones. De todas formas, la mejora en las características de los microcontroladores y compiladores han permitido que se convierta en el lenguaje de programación más utilizado.

Sabiendo que ni la velocidad en la ejecución ni el uso de memoria serán factores críticos, se utilizará el lenguaje C para el diseño del firmware. Esto facilitará, además, la implementación de mejoras y ampliaciones futuras.

1.4.2. Herramientas de Diseño

Para poder llevar a cabo un diseño de forma fácil y eficaz se utilizarán las herramientas que el proveedor suministra:

- **MPLAB IDE v8.30:** es un entorno de programación/depuración con varias herramientas que facilitan el diseño. Permite crear Proyectos utilizando uno o varios códigos, librerías, compiladores, etc. Una característica importante es que muestra el consumo de memoria mediante un gráfico. Debido a la elección del tipo de microcontrolador y el lenguaje de programación, se usará el compilador *MPLAB C18 Compiler*. Este compilador permite la programación en lenguaje C para los microcontroladores de la familia *PIC18 MCUs* de *Microchip Technology Inc.*

- **Programador/Depurador:** para evitar problemas se debería usar el recomendado por el proveedor, en este caso el *MPLAB ICD 2 (Programmer & Debuger)*. Desgraciadamente el programador es bastante caro, unos 116 €, por lo que se usará un clon compatible. Éste es fácil de adquirir, ofrece garantía en el funcionamiento y su coste es inferior a 50 €.

- **Librerías y Código de Ejemplo:** normalmente los proveedores suministran librerías y código de ejemplo que pueden integrarse en un proyecto. Es común en el sector de la ingeniería hacer uso de estas herramientas ya que ahorran tiempo y facilitan el diseño.

  Microchip Technology Inc. ofrece gran cantidad de código capaz de controlar las características de sus microcontroladores. Por este motivo y siempre que sea posible, se implementará el código y las librerías que suministra.

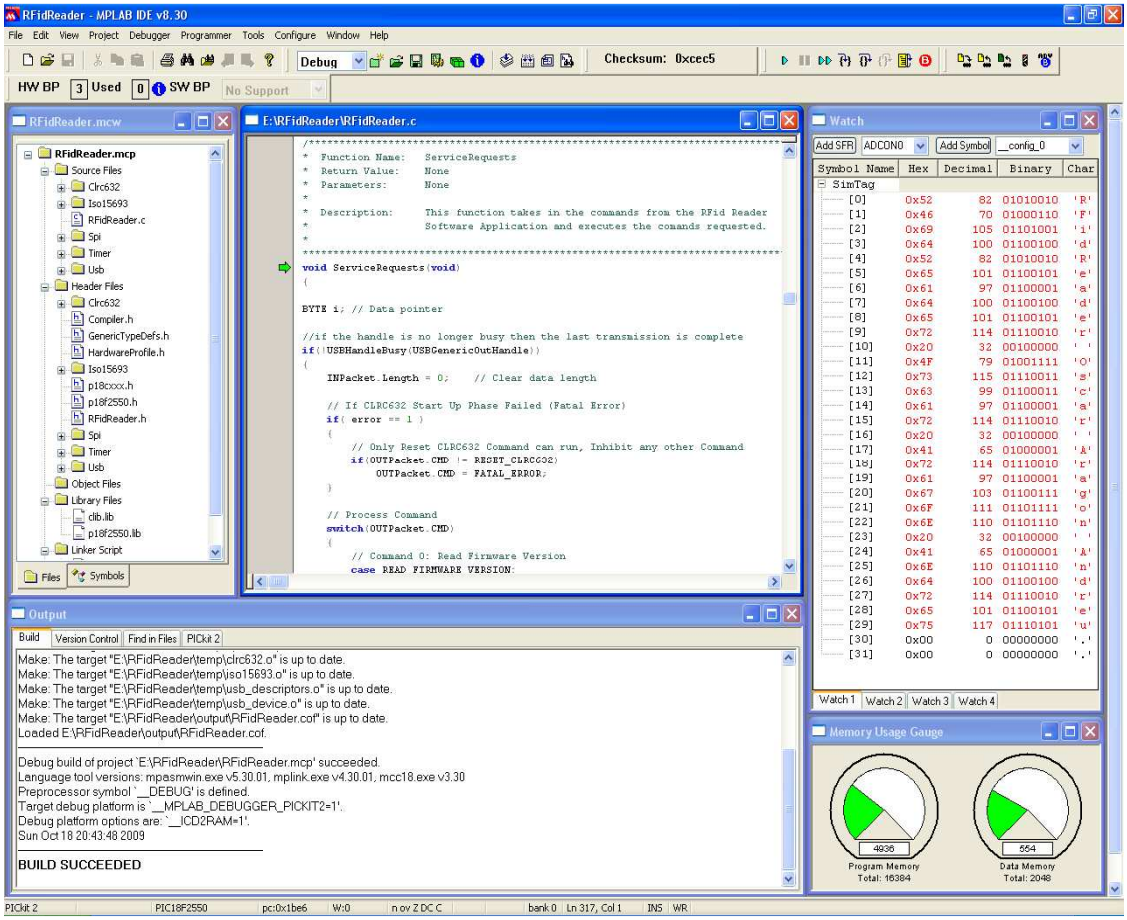
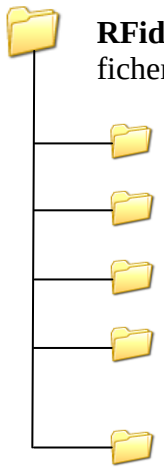


Figura 11. Entorno de Programación MPLAB IDE v8.30

1.4.3. Estructura del Firmware

El Firmware deberá ajustarse a los requisitos impuestos por el hardware. Se seguirá un tipo de programación modular, con varios códigos y librerías para configurar y controlar cada uno de los periféricos establecidos durante el diseño del hardware.

Se creará una estructura de tipo árbol:



RFidReader: es la carpeta raíz del Proyecto. En ella se encontrarán todos los ficheros necesarios para la configuración y el control de todo el Lector.

- **Timer:** contendrá los ficheros relacionados con el periférico Timer.
- **Spi:** contendrá los ficheros relacionados con el periférico SPI.
- **Usb:** contendrá los ficheros relacionados con el periférico USB.
- **Clrc632:** contendrá los ficheros necesarios para la configuración y el control del *CL RC632*.
- **Iso15693:** contendrá los ficheros necesarios para la aplicación de la normativa *ISO/IEC 15693-3*.

Figura 12. Estructura del FW

1.4.4. Desarrollo del Firmware

En los siguientes apartados se hará una explicación de las soluciones adoptadas, haciendo referencia a las funciones que se han utilizado. Para obtener información detallada sobre cada una de ellas será necesario dirigirse al código del firmware, incluido en el Anexo o en el CD del Proyecto.

1.4.4.1. La Configuración del Entorno

Se creará un Proyecto y se configurará el entorno MPLAB IDE seleccionando el compilador *C18* y el dispositivo *PIC18F2550*. Por último, se incluirán las librerías:

- **GenericTypeDefs.h:** define tipos de variables y estructuras genéricas para ayudar en la programación.
- **Compiler.h:** hace referencia a las librerías a incluir según el compilador seleccionado.
- **p18cxxx.h:** hace referencia a las librerías a incluir según el tipo de microcontrolador seleccionado.
- **p18f2550.h:** define los registros del *PIC18F2550* de una forma sencilla para facilitar la configuración y control del microcontrolador.

1.4.4.2. El Timer

Se implementará un temporizador que asegure un tiempo de espera superior a $t_{Start\ Up\ Phase} = 47{,}20\ \mu s$. Para ello se utilizará el Timer0 en modo 8-bit por su sencillez [20]. Se deberá utilizar el *Programable Prescaler* y el valor a cargar será $Cuenta_{timer} = 68\ (decimal)$.

Para configurar y controlar el módulo Timer se utilizará el código de ejemplo y las librerías suministrados por *Microchip Technology Inc* [21].

Se importará la librería original *timers.h*, sin añadir modificaciones para no perder portabilidad, y se creará el archivo *timer0.c*. Este archivo contendrá sólo las funciones que se necesiten:

```
void OpenTimer0(unsigned char config)
void CloseTimer0(void)
```

Estas funciones configurarán, habilitarán y deshabilitarán el periférico Timer.

1.4.4.3. Los Puertos I/O

El Lector usará 11 pines digitales, de los cuales 7 serán salidas y 4 entradas. También se usarán las señales *D+* i *D-* del bus USB [22].

Debido a la multiplexación en los pines hay que estudiar qué pasos se deben seguir para realizar una configuración correcta:

- **Standad I/O:** se utilizarán los *Puertos A* y *C*. Las señales se configurarán como digitales y se tipificará cada una de ellas como entrada o salida según la necesidad.

- **SPI I/O**: se configurará el periférico SPI. Se observará que el diseño del los microcontroladores PIC no tiene en cuenta la señal *"!SS"* cuando estos funcionan en modo Master [23]. Por lo tanto, esta señal se considerará como una *Standard I/O* y se le llamará *CLRC632_SPI_SS*.

- **USB I/O**: Se habilitará y configurará el periférico USB.

Sólo se llevará un control directo sobre el estado de las *Standard I/O*. El estado de las señales de los periféricos SPI y USB las controlará automáticamente el microcontrolador dependiendo de la configuración seleccionada.

1.4.4.4. El Bus SPI

Para configurar y controlar el periférico SPI se utilizará el código de ejemplo y la librería suministrados por *Microchip Technology Inc* [24].

Se importará la librería original *spi.h* y se creará el archivo *spi.c*. Este archivo contendrá las funciones:

```
void OpenSPI( unsigned char sync_mode, unsigned char bus_mode, unsigned
char smp_phase)

unsigned char ReadSPI( void )

unsigned char WriteSPI( unsigned char data_out )
```

Estas funciones se usarán para configurar el bus SPI y para escribrir/leer datos a través de él. Hay que destacar que se realizará una modificación en la función `unsigned char WriteSPI( unsigned char data_out )`, de forma que se reporte un error si se ha producido una colision durante la escritura.

```
0x02, SPI Write Collision
```

También se copiarán las funciones:

```
void CloseSPI( void )

void getsSPI( unsigned char *rdptr, unsigned char length )

void putsSPI( unsigned char *wrptr )
```

Éstas últimas sólo se tendrán en cuenta para futuras mejoras, así como perfeccionar la transmisión de datos enviándo cadenas de Bytes o para implementar un modo *Sleep* al Lector. Ésta característica ahorraría consumo, pudiendo alimentar el Lector con una batería portátil.

1.4.4.5. El Bus USB

El bus USB sigue una especificación muy extensa y resultaría muy difícil diseñar un código de configuración y control adecuados.

En este caso, especialmente, es necesario recurrir al código y librerías de *Microchip Technology Inc.*

Debido a la naturaleza tan específica del estándar, Microchip dedica toda una sección al USB. En el apartado *"Software/Tools"* puede encontrarse *MCHPFSUSB Framework* [25].

Este paquete contiene documentación, firmware y software donde pueden encontrarse varias aplicaciones de ejemplo para algunos microcontroladores PIC. Estas aplicaciones ofrecen diferentes modos de funcionamiento del periférico USB: Serial Emulator para emular *RS-232* bajo USB, HID para ratones y teclados, Mass Storage para dispositivos de almacenamiento, etc.

Sería fácil implementar el "*CDC - Serial Emulator*" en el Lector. Este firmware emula la comunicación serie *RS-232* bajo el bus USB. Su implementación sería justificable en un sistema diseñado con ese tipo de comunicación y que, por necesidades de rediseño, fuera necesaria su migración para su uso mediante el USB. De todas formas, la velocidad en la trasmisión de datos seguiría el estándar RS-232.

Debido a que el Lector se encuentra en una fase inicial de diseño, se importará el código de la aplicación "*Generic Driver Demo – Firmware"* [26]. Éste responde a las características de funcionamiento estándar del USB y permitirá velocidades de transmisión de hasta 12 Mb/s, configuración HIGH-SPEED. Un problema es que ese firmware está diseñado para funcionar en las Placas de Desarrollo que *Microchip Technology Inc.* suministra. Esto dificultará la implementación en el Lector, ya que es un código muy genérico y no está incluido el *PIC18F2550*. En cambio sí el *PIC18F4550*, un microcontrolador superior pero de características similares.

Se deberá estudiar la documentación que se ofrece con el firmware [27] y algunos puntos de la especificación del USB para poder integrar el código de la mejor forma. Este código no se modificará en cuanto a su funcionamiento y características, debiendo ajustar el firmware del Lector a éste.

El código importado realizará las tareas de:

- **Inicialización:** seguirá el diagrama de estados de los dispositivos USB [28].

- **Configuración:** podrá funcionar en modo LOW-SEED (1,5 Mb/s) y HIGH-SPEED (12 Mb/s), pero en esta versión el Lector usará el LOW-SPEED por ser suficiente. Para la transmisión de datos se habilitarán el Endpoint 0 (EP0) y el Endpoint 1 (EP1). Se reservará un espacio de 128 Bytes en la USB RAM. 64 Bytes se destinarán al EP1 OUT y 64 Bytes al EP1 IN [29].

- **Transferencia:** se utilizarán transferencias de Control (EP0) y por Interrupción (EP1) [30]. El código también ofrece la posibilidad de implementar una rutina por encuesta, pero no se hará un uso final de esta opción. De todas formas, no se eliminará del código con el fin de ayudar en tareas de depurado.

El firmware USB importado se integrará en la subcarpeta *Usb* del Lector.

Se destacarán como funciones principales:

```
void USBDeviceTasks(void)
#define USBGenWrite(ep,data,len) USBTxOnePacket(ep,data,len)
#define USBGenRead(ep,data,len) USBRxOnePacket(ep,data,len)
```

Estas funciones permiten la configuración, escritura y lectura del bus USB.

1.4.4.6. El CL RC632

El *CL RC632* es un dispositivo externo al microcontrolador, de todas formas, se tratará a éste como a un periférico más. Esto quiere decir que, al igual que al resto, habrá que configurarlo y controlarlo. Al tratarse de un circuito integrado, el microcontrolador deberá poder acceder a ciertas posiciones de memoria y a ciertos registros del dispositivo.

Estas acciones se realizarán a través del bus SPI. Será necesario diseñar un sistema capaz de enviar y recibir dados por SPI además de poder acceder a los registros a nivel bit, a modo de control.

Se distinguirán cuatro estados de funcionamiento:

- **Inicialización: s**egún las especificaciones del *CL RC632*, a partir de que éste recibe la señal *!RSTPD* debe pasar un tiempo igual o superior a $t_{Start\ Up\ Phase}$ antes de estar disponible [31].

  Para cumplir este requisito se creará la función:

  ```
  void StartUp( void )
  ```

  Esta función será capaz, además, de detectar si la fase ha finalizado correctamente. En el caso contrario se reportará un error:

  ```
  0x01, Start Up Phase Failed
  ```

- **Transferencia de Datos:** para el acceso a los registros, mediante el SPI, se deberá respetar la estructura definida en las especificaciones del *CL RC632* [32].

  Las funciones que posibilitarán la lectura y escritura de los registros serán:

  ```
  void WriteReg( const BYTE CLRC632_REG, BYTE data )
  BYTE ReadReg( const BYTE CLRC632_REG )
  ```

En el caso de intentar acceder a un registro inexistente, se reportara un error:

```
0x03, Register Address unavailable
```

Se destacará que el *CL RC632* dispone de un Buffer FIFO de 64 Bytes [33]. El acceso a éste se realizará mediante el registro FIFOData, mapeado en dirección de memoria 0x02 [34]. El transceiver gestionará automáticamente el Buffer dependiendo de si se escribe o se lee en ese registro.

- **Configuración:** una de las características más relevantes del *CL RC632* es que dispone de unos bloques en E^2PROM reservados para su configuración [35]. Esta configuración se copia cada vez que se ejecuta la fase de *Start Up* del *CL RC632*, evitando que el microcontrolador tenga que programarlo cada vez que lo inicia. Habrá que programar esos bloques con la configuración deseada. También se deberá comprobar que la operación se ha realizado con éxito. Para ello se utilizarán las funciones:

```
void WriteE2( BYTE lsb, BYTE msb, BYTE n_bytes )

void ReadE2( BYTE lsb, BYTE msb, BYTE n_bytes )
```

Un acceso incorrecto a esos bloques reportará un error:

```
0x04, E2PROM Access not alowed
```

- **Control:** además de escribir y leer en los registros a nivel Byte, será necesario el acceso a nivel bit. Esto será útil para iniciar o comprobar eventos como iniciar el Temporizador o comprobar el estado de una interrupción o flag. Para esto se utilizarán las funciones:

```
void SetBit( const BYTE CLRC632_REG, BYTE bit_nr )

BYTE CheckBit( const BYTE CLRC632_REG, BYTE bit_nr )

void ClearInterruptRq( BYTE interrupt )
```

Por otra parte, el *CL RC632* es capaz de realizar acciones preprogramadas mediante el uso de Comandos [36]. Estos realizan tareas tales como: escribir en E^2PROM, transmitir, recibir, etc. Por esta razón se crearán las funciones:

```
void ExecuteCommand( const BYTE CLRC632_COMMAND )

void StopCommand( void )
```

Estas funciones se encargaran de iniciar y detener esos comandos.

Es posible realizar tareas especificas utilizando y combinando las funciones generadas hasta el momento. En el caso del Lector se usarán las siguientes:

- **Configuración por Defecto:** se reservará un espacio en la memoria no volátil del microcontrolador para la configuración por defecto del *CL RC632*. Esto permitirá que, en el caso de tener que cambiar el dispositivo, la configuración pueda ser recuperada en todo momento. Se escribirán los *Bloques 1* y *2* de la E^2PROM *CL RC632* con la configuración y se ofrecerá la posibilidad verificar los datos.

```
void WriteDefaultConfig( void )

void ReadDefaultConfig( void )
```

- **Timeout:** con el fin de detectar errores en las tareas de transmisión y recepción se implementará un temporizador *Timeout*. El tiempo cargado será de un segundo, según los cálculos se considera tiempo más que suficiente para que cualquier operación haya finalizado. En este caso y para hacer un uso mayor del dispositivo se utilizará el Timer interno del *CL RC632* [37]. La función encargada de inicializar el temporizador será:

```
void StartTimer( void )
```

- **Reset:** será conveniente dejar definida una rutina de Reset. Esta rutina permitirá reiniciar el *CL RC632* si se provocara un error no contemplado.

```
void ResetCLRC632( void )
```

1.4.4.7. La ISO 15693

El protocolo de comunicación *ISO/IEC 15693-3* especifica cómo se debe realizar la comunicación entre el Lector y el tag. Esto obliga a utilizar una estructura de datos muy definida [38].

Para ello se deberá analizar bien de dónde provienen los datos, dónde se deberán colocar y que se deberá hacer con ellos. En el caso del Lector los datos vendrán de una aplicación software a través del bus USB, se escribirán en el buffer FIFO del *CL RC632* y se ejecutara la Transmisión mediante un Comando. Finalmente se ejecutará la Recepción y se enviarán los datos recibidos a la aplicación Software.

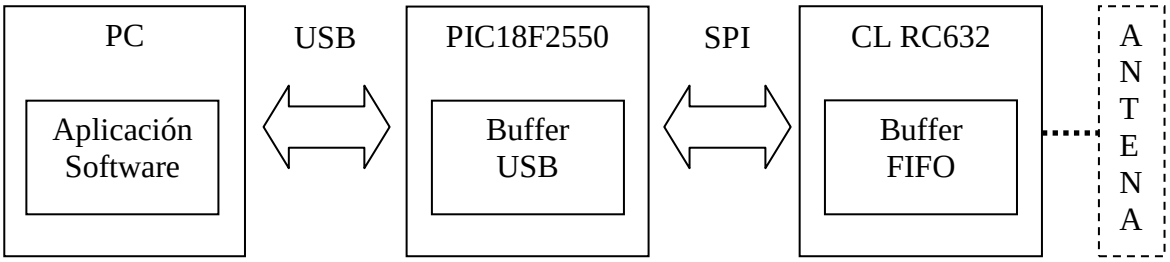


Figura 13. Flujo de Datos

Con este proceso se define un sistema de comunicación asíncrono y secuencial, donde la velocidad relativa de cada vía de comunicación no influye en el proceso como conjunto.

Para implementar esta característica se creará la función:

```
void ISOGeneralCommand( void )
```

Esta función será capaz de procesar cualquier Comando incluído en la *ISO/IEC 15693-3*. Con esta opción se obtiene una gran flexibilidad mediante una función muy sencilla, ya que ésta sólo transfiere datos de un lugar a otro, a modo Buffer FIFO. Esta alternativa facilitará la programación del firmware, pero complicará la programación del software, ya que éste deberá gestionar y procesar toda la información de la forma adecuada.

Para asegurar que el proceso de comunicación entre el tranceiver y el tag se ha realizado correctamente se implementarán dos estados de comprobación: verificar que la transmisión ha finalizado y verificar que la recepción ha finalizado. Cuando la emisión/recepción se inicien también lo hará el contador de *Timeout*, si el temporizador termina su cuenta querrá decir que ha habido un error de temporización durante la transmisión o recepción. En este caso se reportará el error correspondiente:

```
0x05, Transmit Timeout

0x06, Receive Timeout
```

En esta versión del Lector se esperará hasta que una de las condiciones anteriores se cumpla, ya no hay que procesar ningún dato más. De todas formas, hay que tener en cuenta que se podría implementar un sistema de interrupciones si se necesitara.

Una alternativa al funcionamiento de la función *ISOGeneralCommand* sería recibir sólo ciertos datos de la aplicación software y que el microcontrolador decidiera que hacer con ellos antes de enviarlos al buffer FIFO del *CL RC632*. Es una opción menos flexible, pero evita que la aplicación software tenga que organizar toda la información. Además, permitirá ver un funcionamiento diferente del Lector. Por esta razón se crearán las funciones:

```
void WriteSingleMultipleBlocks( void )

void ReadSingleMultipleBlocks( void )

extern void ClearTagData( void )
```

Son funciones específicas y cada una de ellas responde a una tarea muy concreta. Con estas funciones el Lector será capaz de escribir y leer en uno/varios bloques del tag. A estas funciones se les aplicará el mismo reconocimiento de errores usado en la función *ISOGeneralCommand.*

Como se ha mencionado en las Conclusiones sobre el Estudio de la RFID, el Lector no dispondrá de antena. En estas condiciones no tendría mucho sentido realizar una demostración del funcionamiento del sistema, ya que sólo se podría demostrar que éste es capaz de reconocer errores en la comunicación. Por este motivo, se crearán las siguientes funciones:

```
void SimWriteSingleMultipleBlocks( void )

void SimReadSingleMultipleBlocks( void )

extern void SimClearTagData( void )
```

Funcionan de forma similar a sus análogas. Reciben una petición real de la aplicación software, pero en lugar de enviar los datos al transceiver lo hacen a un tag simulado en la memoria volátil del microcontrolador. Por último, envían una respuesta simulada a la aplicación software acorde con la *ISO/IEC 15693-3*. Esta alternativa permitirá efectuar una demostración simulada.

El Tag simulado responderá a las características del *RI-I03-114A*-S1 de *Texas Instruments* [39].

1.4.4.8. El RFidReader

La carpeta principal *RFidReader* contendrá todos los archivos generados hasta el momento. Además, y para finalizar con la fase de diseño del firmware se agregarán:

- **HardwareProfile.h:** contendrá definiciones y macros referentes a las señales hardware usadas en el Lector.

- **RFidReader.h:** contendrá las definiciones y estructuras básicas para el funcionamiento del Lector: la versión del Firmware, los Comandos de Control y la organización de los datos en paquetes.

  Es muy importante analizar el volumen de datos que se pretende gestionar. La aplicación final del Lector será un Control de Acceso, así pues, será necesario reservar un espacio de memoria capaz de albergar un Número de Identificación y/o un Nombre Completo. Tanto el Buffer Genérico USB como el Buffer FIFO del *CL RC632* son de 64 Bytes. Por otra parte, se pueden adquirir tag de diferentes capacidades. Así pues, se calcula que será suficiente reservar un Buffer de Datos de 64 Bytes.

- **RFidReader.c:** contiene el código principal del Lector. Desde él se configurará el microcontrolador, se inicializarán/configurarán todos los periféricos y se atenderá/responderá a las peticiones.

  Se configurará el Sistema de Oscilación mediante el *PLL Prescaler* y *PLL Postscaller* [40], de acuerdo con los resultados obtenidos durante el diseño del hardware. Se habilitará el transceiver USB y se definirá es sistema de interrupciones, ya que el firmware USB necesita al menos una de ellas. En estas condiciones se destinará la interrupción de alta prioridad al USB y se reservará la interrupción de baja prioridad para futuras ampliaciones del Lector.

  Para la inicialización y configuración de los Periféricos *I/O, Timer, SPI, USB* y *CLRC632* se creará la función:

```
void InitializeSystem( void )
```

  La gestión de las peticiones, tanto hardware como software, se realizarán por medio de la función:

```
void ProcessIO(void)
```

  Las peticiones provenientes de la aplicación software serán atendidas por la función:

```
void ServiceRequests(void)
```

En esa función entran en juego los Comandos. Estos realizan tareas específicas programadas y cada uno de ellos tiene un número de identificación único. En el Lector se han definido diez Comandos [41], pero será posible agregar hasta un total de 256. Se destacará que, en el caso de producirse un error en la fase de *Start Up* no se responderá a ninguna petición (Comando), exceptuando la petición de Reset del *CL RC632*. Ese fallo se considera como fatal y el microcontrolador reportará ese error en todo caso.

El procesado de errores se gestionará de forma que se reporte siempre el primer error en producirse. También se habilitará un contador de errores que informe sobre cuántos errores se han producido, este podrá almacenar hasta 256 errores. Una vez alcanzada esa cifra, el contador se reiniciará automáticamente. Por último, se implementará una rutina de limpieza del error:

```
void ClearError(void)
```

Esta función tendrá la particularidad de poder ser llamada por hardware (!*CLR_ERROR_SW*) y por software, pero nunca reseteará el contador de errores con el fin de mejorar la detección de los mismos durante la fase de depurado del firmware.

Para finalizar el código, se agregarán las funciones de rellamada del firmware USB "*Generic Driver Demo – Firmware*". Únicamente se utiliza `void USBCBInitEP(void)`, pero se mantendrán todas sin modificacion con el fin de utilizarlas en futuras versiones del Lector. Son especialmente interesantes las funciones `void USBCBSuspend(void)` y `void USBCBWakeFromSuspend(void)`, que permitirían implementar un modo *Sleep* en el Lector.

1.4.5. Conclusiones sobre el Diseño del Firmware

El Firmware diseñado gestionará un sistema capaz de adquirir datos mediante el bus USB. Posteriormente, los procesará de forma que sea posible utilizar la normativa *ISO/IEC 15693* en su totalidad. Esto cumple parcialmente el objetivo del Proyecto, ya que habrá que proporcionar externamente los datos que el Lector debe procesar. De esta tarea se ocupara el Software.

Se destacará que, aunque el Sistema esta basado en un lector de etiquetas pasivas de RFID, el diseño permite utilizarlo también como un dispositivo USB cuyas caracteristicas podrán ser implementadas en un futuro.

Como puntos importantes durante el desarrollo del Firmware se distinguirán:

- **Herramientas:** se ha utilizado un entorno de programacion actual y acorde con las necesidades. Además se ha implementado/reutilizado el código y las librerías de una fuente externa.

- **Estructura:** se ha generado un código modular que facilita la portabilidad y futuras ampliaciones.

- **Periféricos:** se ha diseñado un sistema que configura y controla los puertos I/O, un Timer, el puerto SPI y el bus de datos USB del *PIC18F2550*. También un dispositivo externo al microcontrolador, el transceiver de RFID *CL RC632*, utilizado para la aplicación de la *ISO/IEC 15693*.

- **Volumen de Datos:** se podrán realizar transferencias de hasta 64 Bytes, suficiente para un Control de Acceso RFID.

- **Contol de Errores:** se ha implementado un control que identifica y reporta los errores que se hayan podido ocasionar durante el funcionamiento habitual del Sistema.

- **Memoria:** basandose en la herramienta utilizada para la programación se comprueba que el uso de memoria de programa ha sido de casi 10 kB y de 554 Bytes para la memoria de datos.

1.5. Diseño del Software

Para que el Lector pueda funcionar correctamente se diseñará una aplicación software capaz de enviar y recibir datos a través del bus USB. De nuevo, Microchip ofrece un software de ejemplo que complementa al firmware USB integrado en Lector.

1.5.1. La Aplicación Base

Se usará el *MCHPUSB PnP Demo*, incluido en el paquete "*USB Device - MCHPUSB - Generic Driver Demo*", como base para el diseño de la aplicación del Lector. Esta demostración está realizada mediante *Microsoft Visual C++ 2005* y proporciona una aplicación básica que conecta y se comunica con el microcontrolador mediante el bus de datos USB [42]. Con el fin de aprovechar el código proporcionado se estará obligado a utilizar ese lenguaje de programación.

Para la configuración y gestión de los datos a través del bus USB, la aplicación utiliza la librería *mpusbapi.dll.*

El volcado y adquisición de datos se ejecuta, en segundo plano, mediante la función:

```
Void ReadWriteThread_DoWork(System::Object^  sender, System::Component
Model::DoWorkEventArgs^  e)
```

La representación de los datos por pantalla se realiza mediante:

```
Void ShowResponse(System::Object^  sender, System::EventArgs^  e)
```

Esta función se ejecuta cuando el módulo Timer de la aplicación termina su cuenta. Este se ha configurado para ofrecer un refresco de pantalla cada 16 ms, unos 60 Hz aproximadamente.

1.5.2. El RFidReader v1.0

Aunque la aplicación final se destine a la gestión de un Sistema de Control de Acceso, se diseñara una aplicación reutilizable. Esto quiere decir que trabajará principalmente como un sistema de volcado y adquisición de datos a través del bus USB. Esta característica ofrecerá la posibilidad de reutilizar la aplicación aunque se decida hacer grandes cambios en el diseño del hardware y/o del firmware, como sustituir toda la parte referente al RFID.

Para ello se implementara un sistema de comunicación basado en dos tipos de mensaje: mensaje de pregunta y mensaje de respuesta. Cada uno de ellos estará formado por una cabecera y los datos pertinentes.

En el caso de la pregunta, la cabecera estará compuesta por la longitud del mensaje en Bytes y el comando a ejecutar. Los datos que se quieran enviar se anexarán a esa cabecera.

Longitud (1 Byte)	Comando (1 Byte)	Datos (62 Bytes)

Figura 14. Mensaje de Pregunta

La cabecera de la respuesta estará compuesta por la longitud del mensaje, el comando al que se responde y el error que se haya podido producir. Los datos que se reciban se anexarán a esa cabecera.

Longitud (1 Byte)	Comando (1 Byte)	Error (1 Byte)	Datos (61 Bytes)

Figura 15. Mensaje de Respuesta

En estas condiciones se destacará que:

- La longitud de cada mensaje será de 64 Bytes, acorde con el Buffer de datos establecido durante el diseño del Firmware.

- El uso de las cabeceras sacrificará el volumen de datos a enviar, pero no se considera crítico según los cálculos. Además, ésta es la característica principal que permitirá la reutilización del sistema.

Así pues, teniendo en cuenta esos requisitos, se diseñará una aplicación que responderá a dos modos de funcionamiento:

- **Modo Administrador:** realizará tareas de configuración, diagnóstico y ejecución. También mostrará por pantalla los datos necesarios para poder analizar el comportamiento del Lector y permitirá guardar un registro de la actividad realizada en un archivo de texto. Este modo será el que permitirá implementar futuras modificaciones en el tipo de aplicación, ya sea RFID o no.

- **Modo Usuario:** responderá específicamente al funcionamiento de un Sistema de Control de Acceso. Se podrán agregar y eliminar usuarios a una base de datos por medio de un archivo de texto. Si el usuario esta en la base datos se le permitirá el acceso, en el caso contrario no se le permitirá. Al finalizar la aplicación se guardará automáticamente un registro de la actividad en un archivo de texto.

1.5.2.1. Modo Administrador

El Modo Administrador esta diseñado para ser utilizado por un usuario con conocimientos sobre el diseño del Lector y la aplicación de la *ISO/IEC 15693-3*.

Estará compuesto por seis secciones:

- **Estatus:** en esta sección se muestra si el Lector está conectado, o no, a la aplicación software.

- **Servicios:** mostrará un listado de todos los servicios disponibles, ofreciendo la posibilidad de agregar hasta un total de 256. Los servicios integrados hasta el momento son:

  ReadFirmwareVersion, lee la versión de Firmware del Lector.

  WriteDefaultConfiguration, escribe la configuración por defecto en el *CL RC632*.

  ReadDefaultConfiguration, lee la configuración por defecto del *CL RC632*.

  ISOGeneralCommand, ejecuta cualquier comando de la *ISO/IEC15693-3*.

  WriteSingleMultipleBlocks, escribe en un bloque o varios del tag.

  ReadSingleMultipleBlocks, lee un bloque o varios del tag.

  ClearTagData, borra todos los datos del tag (escribe 0 en todos los bloques).

  ResetCLRC632, ejecuta la rutina de reset del *CL RC632*.

- **Configuración del Sistema:** permitirá configurar el comportamiento del Lector acorde con la normativa *ISO/IEC 15693-3*. Se podrán modificar los Flags "Sub-carrier", "Data_rate" y elegir la capacidad de memoria del tag a utilizar.

```
Void UpdateFlags(System::Object^ sender, System::EventArgs^ e)

Void UpdateAvailableMemory(System::Object^  sender, System::Windows::
Forms::KeyPressEventArgs^  e)
```

- **Argumentos:** en esta sección se deberán escribir los datos que se quieran enviar al Lector/Tag. Estos datos podrán ser de dos tipos: números enteros del 0 al 255 y caracteres ASCII. Se ha definido esta tipología porque permitirá enviar un único Byte, útil para la configuración de registros de 8 bits, o permitirá enviar cadenas de Bytes en codificación ASCII, para enviar caracteres alfanuméricos al tag. Se ha definido un sistema de identificación para cada tipo dato: se deberá utilizar el identificador '#' en el caso de los valores numéricos y el identificador ';' para el caso de los caracteres ASCII. Estos identificadores se deberán utilizar imprescindiblemente delante de cada tipo de dato.

Se destacará que el orden en el paso de los argumentos estará definido por el servicio que se quiera utilizar. Dependiendo del servicio seleccionado se ejecutara una de las funciones siguientes:

```
bool Get_GRL_Arguments(void)
bool Get_WSMB_Arguments(void)
bool Get_RSMB_Arguments(void)
```

- **Log:** esta ventana mostrará los datos que se envíen y reciban del Lector. Para facilitar la interpretación, los mensajes se mostrarán de tres formas distintas: cadenas de Bytes en Hexadecimal, cadenas de Bytes en ASCII e información alfanumérica. Se destacará que las cabeceras aparecerán a parte de los datos con el fin de poder identificarlas rápidamente.

- **Controles:** ejecutarán Servicios u otras tareas relacionadas con el entorno. Se han definido los siguientes:

Send, ejecuta el Servicio seleccionado.

```
Void RunRequest(System::Object^  sender, System::EventArgs^  e)
```

Clear Error, limpia el error que se haya podido ocasionar.

```
Void ClearError(System::Object^  sender, System::EventArgs^  e)
```

Save Log, guarda la actividad realizada en un fichero de texto, desde el inicio de la aplicación hasta el momento. El nombre del archivo responderá al formato “dd-MM-aa_hh.mm.ss.txt” para facilitar su posterior uso.

```
void WriteToLogFile(void)
void SaveLogFile(void)
```

User Mode, permite cambiar la aplicación al Modo Usuario.

```
Void ChangeMode(System::Object^  sender, System::EventArgs^  e)
```

Help, muestra la ayuda de la aplicación.

```
Void ShowHelp(System::Object^  sender, System::EventArgs^  e)
```

Exit, termina la aplicación.

```
Void Exit(System::Object^  sender, System::EventArgs^  e)
```

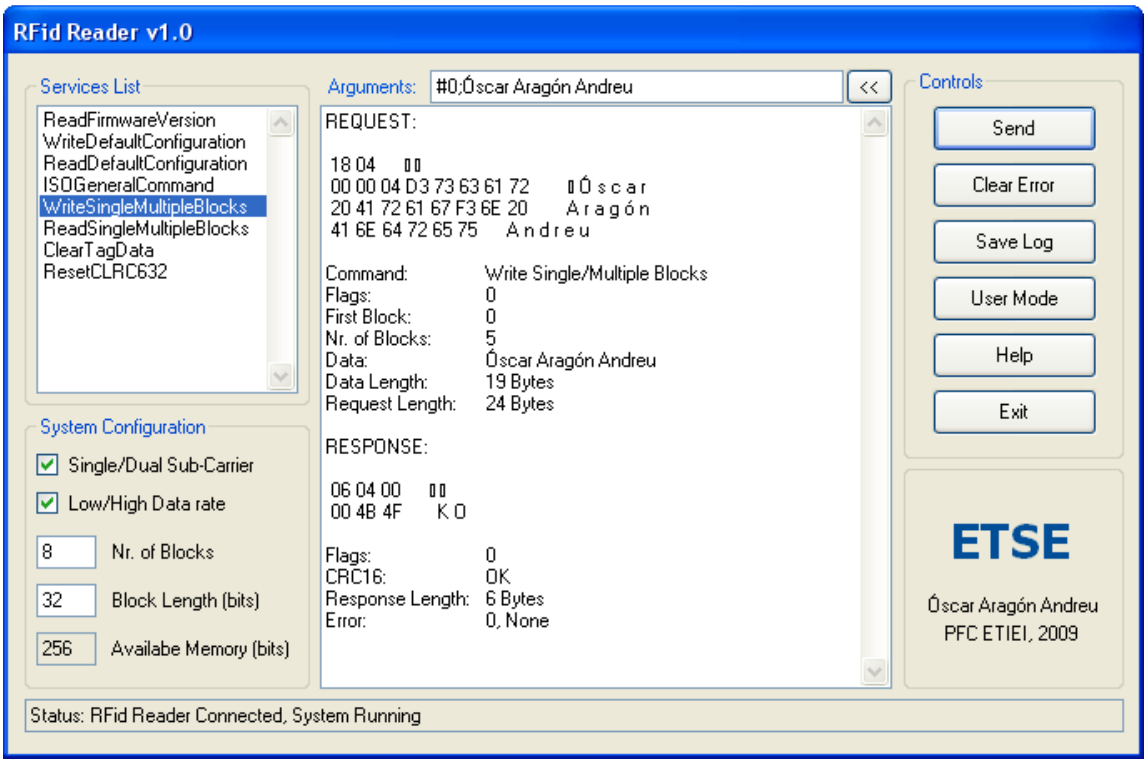


Figura 16. Modo Administrador

Con el fin de evitar ciertos errores se ha definido un conjunto de mensajes “pop-up”. Estos mensajes se mostrarán cuando se intente hacer un uso indebido de la aplicación y ofrecerán información, a modo guía, sobre como solucionar el problema.

Errores en los Argumentos	
Arguments required, click on Help	El Servicio necesita Argumentos y no se han introducido
2 Arguments required, click on Help	El Servicio necesita dos Argumentos y no se han introducido
Only 2 Arguments are required!	Se han introducido más Argumentos de los necesarios
IDs must be '#' or ';'	No se han utilizado los identificadores
Check Arguments IDs	No se han utilizado los identificadores correctamente
First Argument must be a '#' ID	La secuencia en los Argumentos no es correcta
Second Argument must be a ';' ID"	La secuencia en los Argumentos no es correcta
'#' ID must be a number	El identificador utilizado no corresponde con el tipo de dato
'#' ID requires values from 0 to 255	El dato introducido no es correcto

Errores en la Transmisión de Datos	
Nr. Of Blocks Exceeded	Se intenta acceder a un bloque inexistente del tag
Available Memory Exceeded	Los datos a enviar sobrepasan la memoria disponible del tag

Errores en la Aplicación	
Cannot find Help.txt	El fichero de ayuda no se encuentra en el directorio raíz

Tabla 7. Errores en la Aplicación SW

1.5.2.2. Modo Usuario

El Modo Usuario responderá exclusivamente a un sistema de Control de Acceso.

Estará compuesto por cinco secciones:

- **Estatus:** en esta sección se muestra si el Lector esta conectado o no a la aplicación software.
- **Lista de Usuarios:** mostrara los usuarios que actualmente están añadidos en la base de datos.
- **Usuario:** permitirá introducir usuarios a la base de datos. Se acepta cualquier valor ASCII.
- **Log:** mostrará la última acción realizada.
- **Controles:** ejecutarán tareas específicas de un Sistema de Control de Acceso. Se han definido las siguientes:

  Add User, permite añadir un usuario a la base de datos.

  ```
  Void AddUser(System::Object^  sender, System::EventArgs^  e)
  ```

  Remove User, elimina al Usuario seleccionado de la base de datos.

  ```
  Void RemoveUser(System::Object^  sender, System::EventArgs^  e)
  ```

  Access Control, comprueba el Usuario que solicita el acceso.

  ```
  Void AccessControl(System::Object^  sender, System::EventArgs^  e)
  ```

  Admin Mode, permite cambiar la aplicación al Modo Administrador.

  ```
  Void ChangeMode(System::Object^  sender, System::EventArgs^  e)
  ```

  Help, muestra la ayuda de la aplicación.

  ```
  Void ShowHelp(System::Object^  sender, System::EventArgs^  e)
  ```

  Exit, termina la aplicación.

  ```
  Void Exit(System::Object^  sender, System::EventArgs^  e)
  ```

En el Modo Usuario, a diferencia del Modo Administrador, se guardará automáticamente un registro de la actividad realizada en un fichero de texto, desde el inicio de la aplicación hasta su finalización. El nombre del archivo responderá al formato "DD-MM-AA_hh.mm.ss.txt" para facilitar su posterior uso.

Sólo los errores en la transmisión de datos y de aplicación se tendrán en cuenta para este modo de funcionamiento.

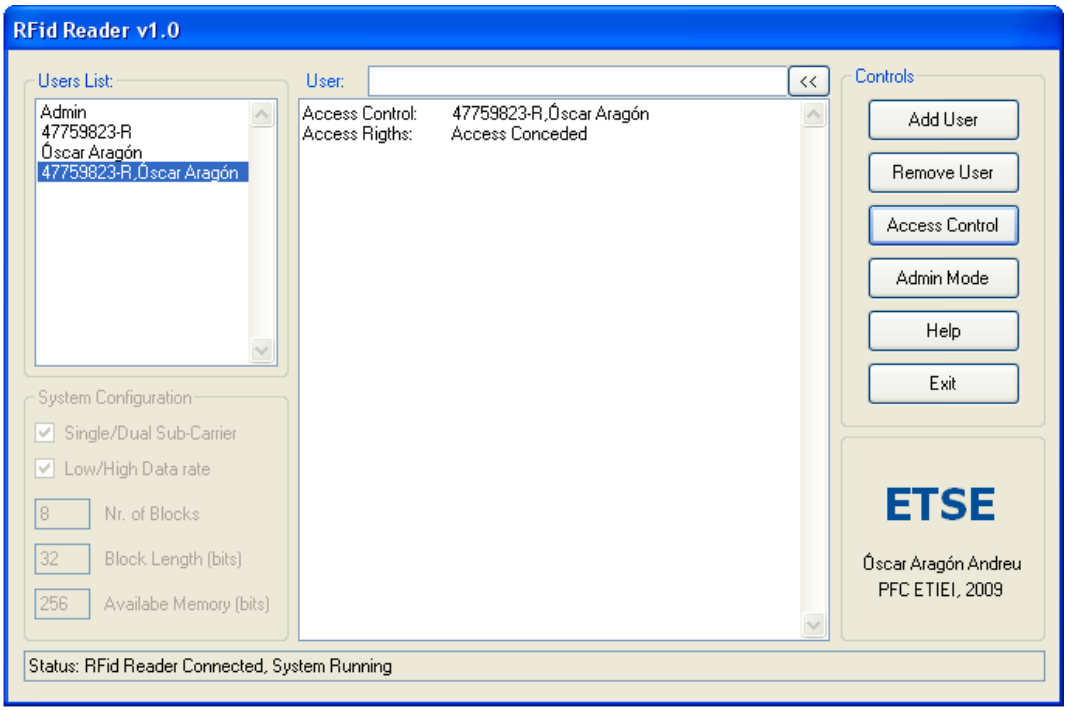


Figura 17. Modo Usuario

1.5.3. Conclusiones sobre el Diseño del Software

Se ha diseñado una aplicación software que permite al usuario el control del sistema, el volcado y la adquisición de datos. Se destacarán como puntos importantes:

- **Software de Diseño:** se ha debido aprender a utilizar un lenguaje de programación orientado a objetos. Este ha permitido crear una interfaz agradable que funciona bajo un entorno de Windows.

- **Aplicación RFID:** se ha diseñado una aplicación que responde a un control de acceso de RFID. Cubriendo así los requisitos de impuestos inicialmente.

- **Flexibilidad:** en todo momento se ha tenido en cuenta implementar una aplicación que fuera reutilizable. Esto permitirá que el sistema pueda trabajar con otro tipo de aplicaciones, ya sean RFID o no. Sólo será necesario realizar algunas modificaciones en la aplicación software actual.

2. Memoria de Cálculo

Lector de Etiquetas Pasivas de RFID

TITULACIÓN: Ingeniería Técnica Industrial en Electrónica Industrial

AUTOR: Óscar Aragón Andreu
DIRECTOR: José Luís Ramírez Falo

FECHA: diciembre / 2009

2.1. Cálculos sobre el Hardware

2.1.1. La Fuente de Alimentación

Datos:

La parte digital del Lector se podría alimentar a 3,3 V [43]. Se conseguiría un consumo reducido y un alargamiento de la vida de los dispositivos principales. Por el contrario, el ruido inyectado durante la transmisión se haría mas evidente y podría ser necesario un sistema de filtrado. Además, se debería preparar otra alimentación a 5 V para la parte analógica [44]. Por esos motivos y con el fin de simplificar el diseño, se implementará solamente una alimentación de 5 V.

Dispositivo	Tensión Digital	Tensión Analógica	Intensidad Máx.
PIC18F2550	3,3 V o 5 V	-	250 mA
CL RC632	3,3 V o 5 V	5 V	200 mA

Tabla 8. Características de Alimentación en el µC y Transceiver

Por motivos de disponibilidad, se elegirá el *L7805CV* como regulador de tensión. A éste se le conectará unos condensadores que cumplan las especificaciones [45].

Dispositivo	Tensión Entrada	Tensión Salida	Intensidad Máx.
L7805CV	hasta 35 V	5 V	1 A

Tabla 9. Características de Alimentación en el L7805CV

El transformador se deberá elegir teniendo en cuenta los cálculos.

Cálculos:

Siguiendo las especificaciones[2] de los componentes seleccionados:

(1) $7{,}5\ V < Vin < 35\ V$
(2) $Vdd = 5\ V$
(3) $Idd = 250\ mA + 200\ mA = 450\ mA$

Resultados:

$7{,}5\ V < Vin < 35\ V$

$Vdd = 5\ V$

$Idd = 450\ mA$

[2] Para el cálculo de la intensidad se tendrá en cuenta el consumo de intensidad máximo especificado.

2.1.2. El Sistema de Oscilación

Datos:

La naturaleza específica del Lector obliga a utilizar dos unidades de oscilación: una para el transceiver, con una $f_{osc\_clrc632} = 13{,}56\ MHz$ según especificaciones [46], y otra para el microcontrolador. La frecuencia de trabajo de éste deberá ser superior a la del transceiver, para asegurar un control adecuado.

El Lector debe utilizar el bus de datos USB. Por ese motivo, el sistema de oscilación del *PIC18F2550* se deberá ajustar a una de las configuraciones USB propuestas en las especificaciones del microcontrolador [47]. Por motivos de disponibilidad se implementará un oscilador con una $f_{osc\_pic} = 20\ MHz$.

Cálculos:

Las condiciones propuestas anteriormente ofrecerán un sistema con las siguientes características:

$$(4)\ f_{cpu} = \frac{96\ MHz}{PLL\ Postscaler\ value} = \frac{96\ MHz}{4} = 24\ MHz$$

$$(5)\ f_{spi} = \frac{f_{cpu}}{4} = \frac{24\ MHz}{4} = 6\ MHz$$

$$(6)\ f_{usb} = \frac{f_{cpu}}{4} = \frac{24\ MHz}{4} = 6\ MHz$$

Resultados:

$f_{osc\_clrc632} = 13{,}56\ MHz$

$f_{osc\_pic} = 20\ MHz$

$f_{cpu} = 24\ MHz$

$f_{spi} = 6\ MHz$

$f_{usb} = 6\ MHZ$

2.1.3. La Fase de Start Up del CL RC632

Datos:

Según las especificaciones, la frecuencia de trabajo del transceiver es:

$$f_{osc\_clrc632} = 13{,}56\ MHz \rightarrow {}^{1}/_{f_{osc\_clrc632}} = t_{clrc632} = \ 73{,}75\ ns$$

Se necesitan varios ciclos de reloj para completar la fase de Start Up [48]:

$$clk_{Start\ Up\ Phase} = 640$$

Cálculos:

Tiempo mínimo necesario para completar la fase de Start Up:

$$(7)\ t_{Start\ Up\ Phase} = clk_{Start\ Up\ Phase} \cdot t_{clrc362} = 640 \cdot 73{,}75\ ns = 47{,}20\ \mu s$$

Resultados:

$t_{Start\ Up\ Phase} = 47{,}20\ \mu s$

2.2. Cálculos sobre el Firmware

2.2.1. El Timer

Datos:

Tiempo necesario para completar la fase de Start Up:

$$t_{Start\ Up\ Phase} = 47{,}20\ \mu s$$

Se utilizará el Timer en modo 8-bit [49] por su sencillez. Se dispondrá de un registro de 8 bits, por lo que el valor a cargar no debe ser superior a 256:

$$Cuenta_{\ timer} \leq 256$$

Frecuencia del Timer:

$$f_{timer} = {f_{osc}}/{4} \rightarrow f_{timer} = 6\ MHz \rightarrow t_{timer} \approx 166\ ns$$

Cálculos:

Valor a cargar en el Timer:

$$(8)\ Cuenta_{timer} = \frac{t_{Start\ Up\ Phase}}{t_{timer}} = 284$$

Sobrepasa el límite permitido por el registro, por lo que se necesitará usar el Prescaler 1:2

$$t_{timer\ 1:2} \approx 332\ ns$$

$$Cuenta_{timer\ 1:2} = 142$$

El Timer es incremental:

$$(9) \quad Cuenta_{timer} = Cuenta_{timer} - Cuenta_{timer\ 1:2}$$

$$Cuenta_{timer} = 256 - 142$$

$$Cuenta_{timer} = 114$$

Resultados:

$Cuenta_{timer} = 114$

Observaciones:

Se observa que el valor de 114 no es suficiente para asegurar que la fase de Start Up se complete con éxito. Por ese motivo y sabiendo que un aumento del tiempo no será crítico para el funcionamiento del Lector, se decide ampliar $t_{Start\ Up\ Phase}$ en un 33% de su valor real.

$$t_{Start\ Up\ Phase\_1} \approx 62{,}70\ \mu s$$

$$Cuenta_{timer\_1} = 68$$

Resultados $_1$:

$Cuenta_{timer\_1} = 68$

2.2.2. El Buffer de Datos

Datos:

La aplicación final del Lector será un Control de Acceso, así pues, será necesario reservar un espacio de memoria capaz de albergar un número de identificación y/o un nombre completo. Sabemos que cada carácter ASCII ocupará un Byte.

Cálculos:

Para los cálculos se tendrá en cuenta un D.N.I. y un nombre completo separados por una coma. Esta forma de organización facilitará la importación a una base de datos como Microsoft Access, ya que usa el carácter coma para diferenciar entre campos. Se hará una estimación teniendo en cuenta un nombre completo normal y uno considerado largo:

47759823-R,Óscar Aragón Andreu = 30 caracteres

12345678-X,José Manuel Benítez de los Ríos = 42 caracteres

Resultados:

Tanto el Buffer Genérico USB como el Buffer FIFO del *CL RC632* son de 64 Bytes. Por otra parte, se pueden adquirir Tags de diferentes capacidades. Así pues, se estima como suficiente reservar un Buffer de Datos de 64 Bytes.

$$Buffer_{datos} = 64\ Bytes$$

2.2.3. El Temporizador Timeout

Datos:

Siguiendo la normativa *ISO/IEC 15693*, una transmisión de datos deberá tener la siguiente forma:

SOF	Datos	EOF

En la petición de datos [50], transferencia del Lector al tag, se tiene que:

$t_{SOF} = 75{,}52\ \mu s$

$t_{EOF} = 37{,}76\ \mu s$

Teniendo en cuenta la tasa de transmisión de datos más lenta (1 out of 256) [51], cada Byte tardará en transferirse:

$t_{Byte} = 4{,}833\ ms$

En la recepción de datos, respuesta del tag al Lector, y teniendo en cuenta la tasa de transferencia más lenta se tiene que [52]:

$t_{SOF} = 149{,}04\ \mu s$

$t_{EOF} = 151{,}04\ \mu s$

$t_{Byte} = 1{,}208\ ms$

Cálculos:

En esta versión del Lector se podrán hacer transferencias de un máximo de n_{Bytes} = 64 Bytes. En el caso que se quisieran transferir al tag todos esos datos, la petición del Lector al tag tardaría en procesarse:

$$(10) \quad t_{petición} = t_{SOF} + nbytes \cdot t_{Byte} + t_{EOF}$$

$$t_{petición} = 75{,}52\ \mu s + 64 \cdot 4{,}833\ ms + 37{,}76\ \mu s$$

$$t_{petición} \approx 309{,}42\ ms$$

La respuesta se procesaría en:

$$t_{respuesta} = t_{SOF} + nbytes \cdot t_{Byte} + t_{EOF} \quad (11)$$

$$t_{respuesta} = 149{,}04\ \mu s + 64 \cdot 1{,}208\ ms + 151{,}04\ \mu s$$

$$t_{respuesta} \approx 77{,}61\ ms$$

Resultados:

En el peor de los casos se tendría que una transferencia tarda 309,42 ms. Se dice que un humano puede reconocer eventos cada 500 ms aproximadamente, así pues, se cree oportuno implementar un temporizador *Timeout* de 1 segundo. En el caso de producirse el error, el operario podría intuir que algo no está funcionando según lo previsto. Por otra parte, este tiempo no perjudica al funcionamiento del Lector, puesto que ya se ha producido un error.

$t_{Timeout} = 1\ s$

3. Planos

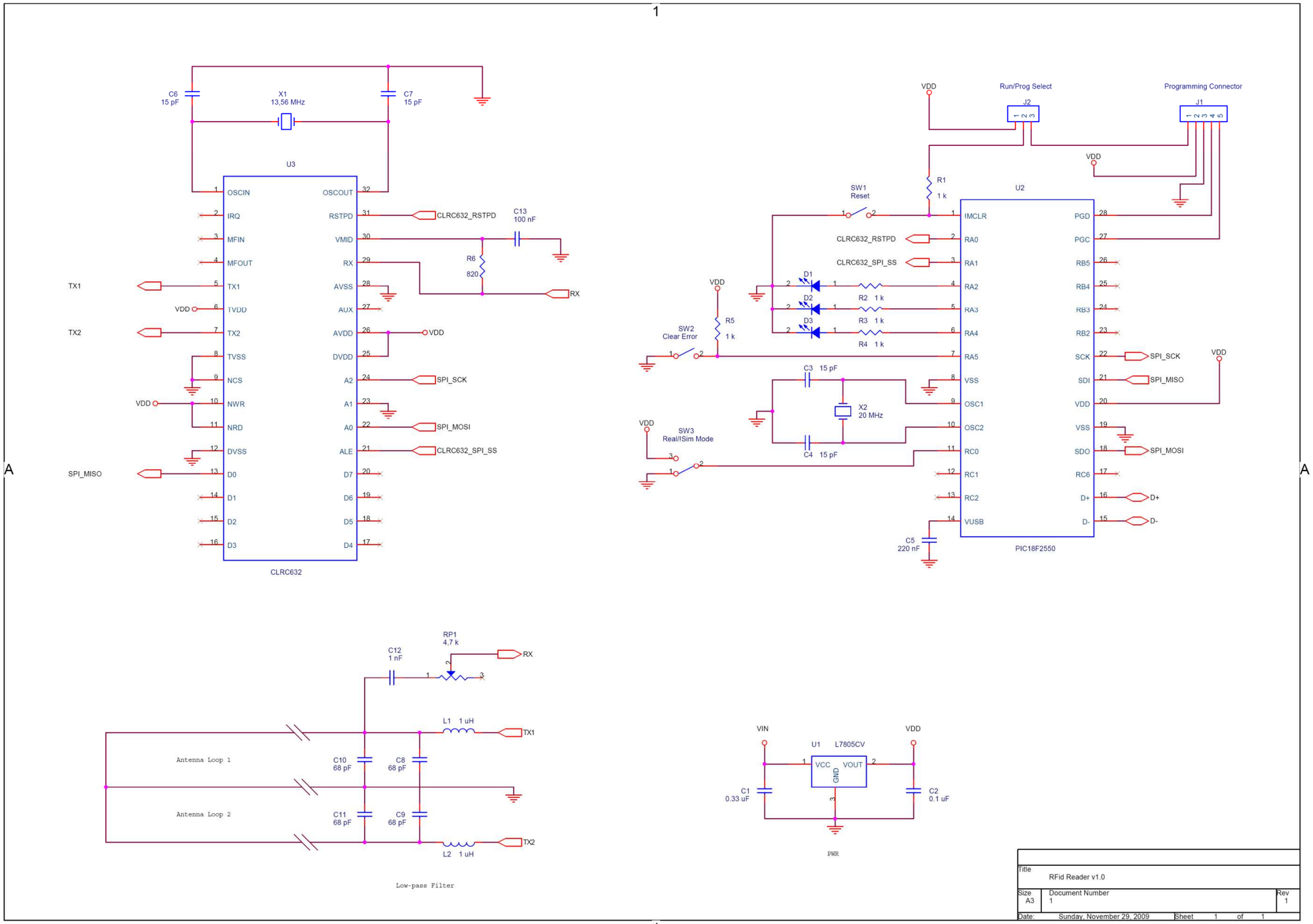
Lector de Etiquetas Pasivas de RFID

TITULACIÓN: Ingeniería Técnica Industrial en Electrónica Industrial

AUTOR: Óscar Aragón Andreu
DIRECTOR: José Luís Ramírez Falo

FECHA: diciembre / 2009

3.1. Esquema Electrónico del RFidReader



3.2. Lista de Materiales

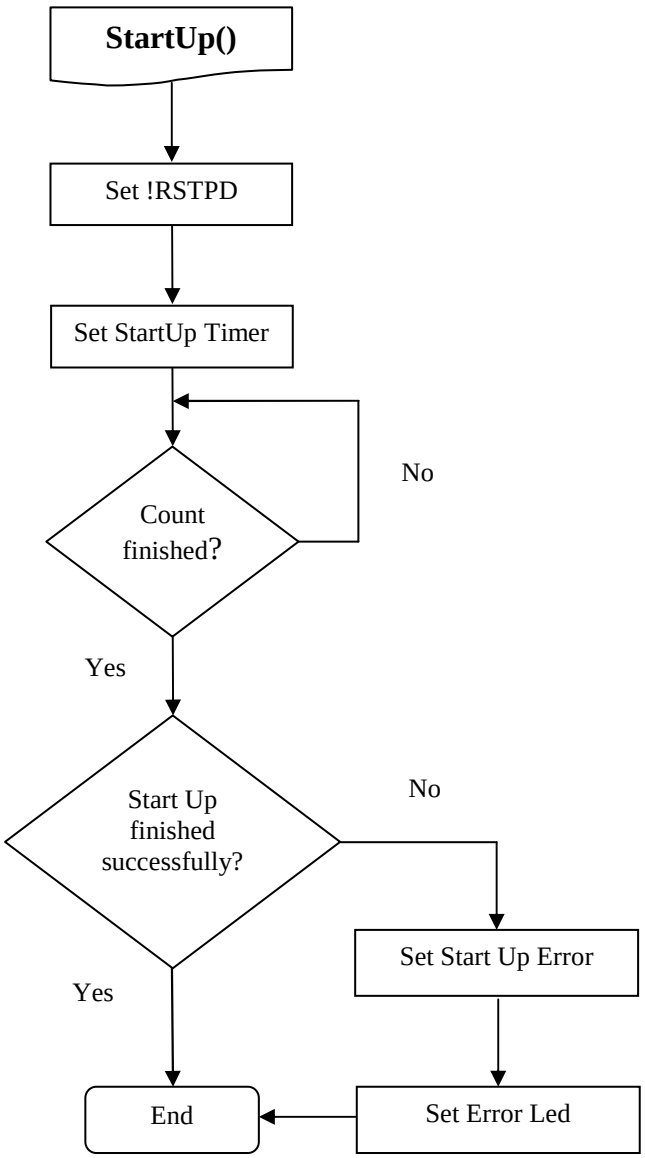
Designator	Description	Value	Qty
C1	Ceramic Capacitor, 0,33 uF 50 V	0,33 uF	1
C2	Ceramic Capacitor, 0,10 uF 50 V	0,10 uF	1
C3, C4, C6, C7	Ceramic Capacitor, 15 pF 50 V	15 pF	4
C5	Ceramic Capacitor, 220 nF 50 V	220 nF	1
C8, C9, C10, C11	Ceramic Capacitor, 68 pF 50 V	68 pF	4
C12	Ceramic Capacitor, 1 nF 50 V	1 nF	1
C13	Ceramic Capacitor, 100 nF 50 V	100 nF	1
D1	Light-Emitting Diode Green	-	1
D2	Light-Emitting Diode Red	-	1
D3	Light-Emitting Diode Yellow	-	1
J1	JTAG Programming Connector	-	1
J2	Header 4 pin, Run/Programming selector	-	1
L1, L2	Inductor, Magnetically Shielded 1uH	1 µH	2
R1, R2, R3, R4, R5	Resistor, 1 kOhm 1/4 W	1 kΩ	5
R6	Resistor, 820 Ohm 1/4 W	820 Ω	1
RP1	Potentiometer, 4,7 kΩ	4,7 kΩ	1
SW1, SW2	Tactile Microswitch	-	2
SW3	Toggle Switch	-	1
U1	L7805CV, Positive Voltage Regulator	-	1
U2	PIC18F2550, 8-bit Microcontroller	-	1
U3	CLRC632, RFID Transceiver	-	1
X1	Crystal Oscillator, 13,56 MHz	13,56 MHz	1
X2	Crystal Oscillator, 20 MHz	20 MHz	1

3.3. Diagramas del Firmware

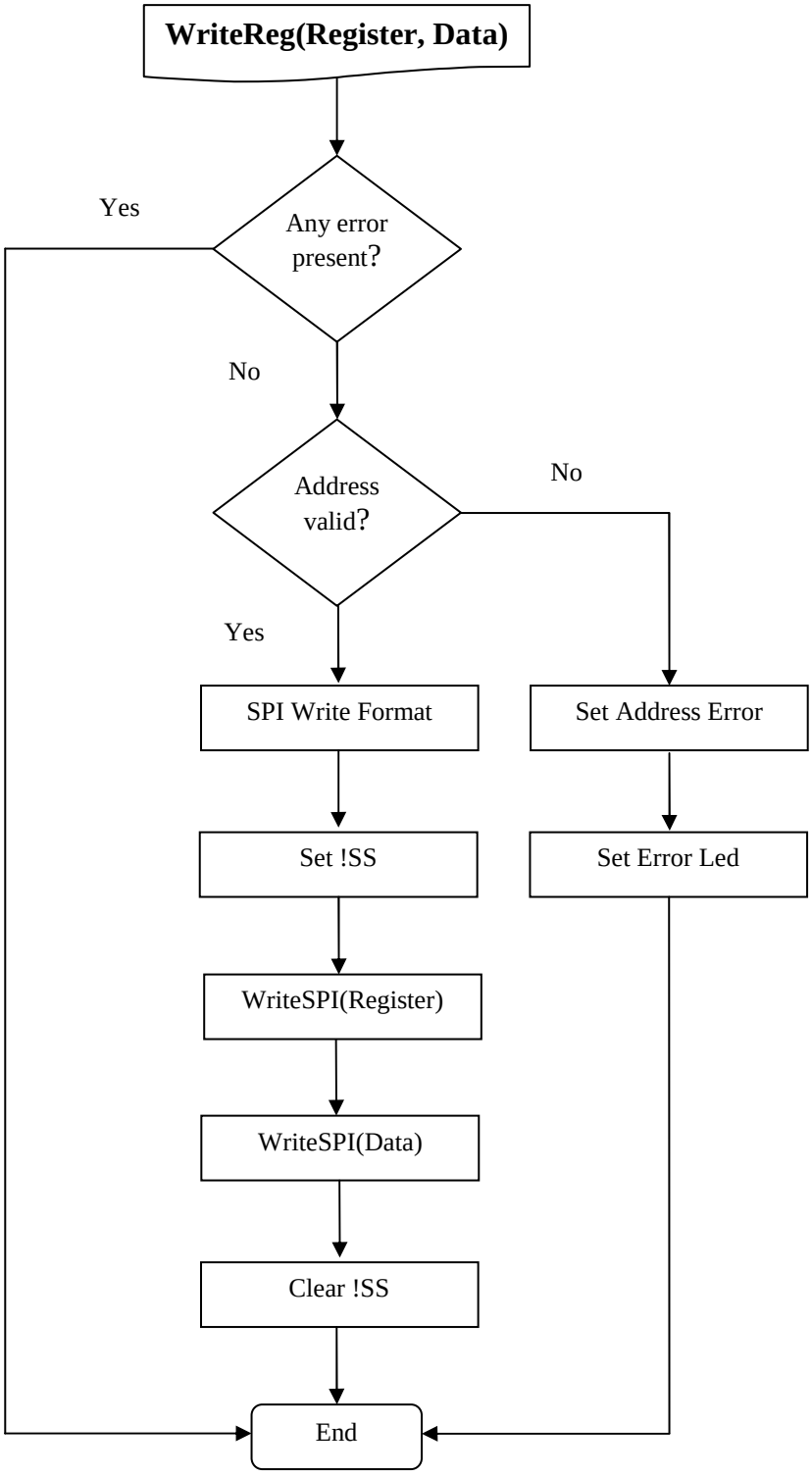
En este apartado no se representarán los diagramas referenetes al código importado. Únicamente los que se han necestitado para crear el código nuevo.

3.3.1. clrc632.c

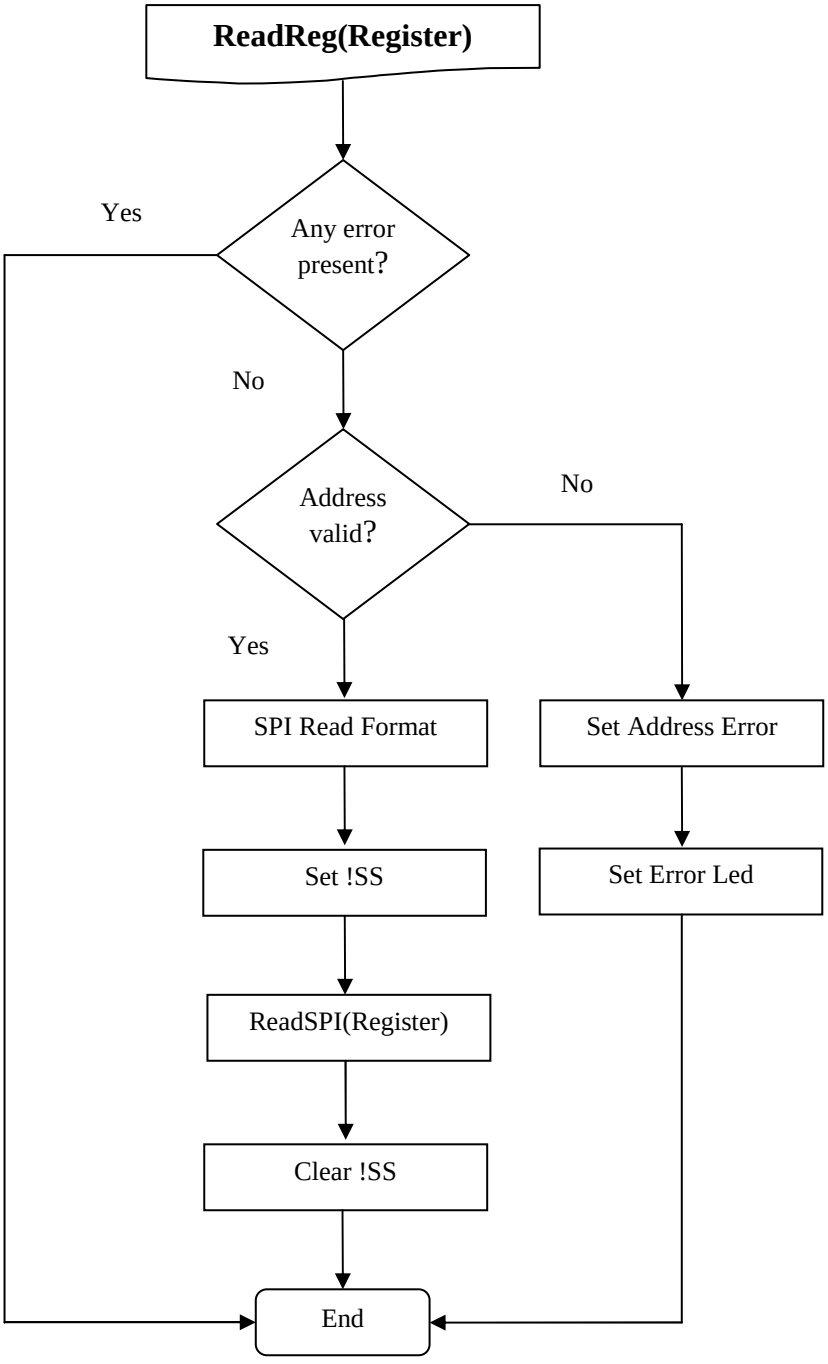
```
void StartUp( void )
```



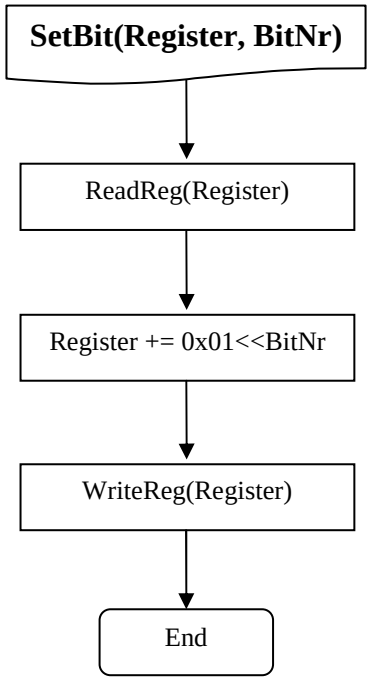
```
void WriteReg( const BYTE CLRC632_REG, BYTE data )
```



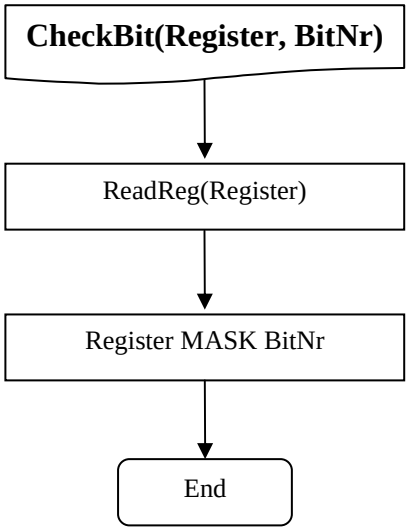
```
BYTE ReadReg( const BYTE CLRC632_REG )
```



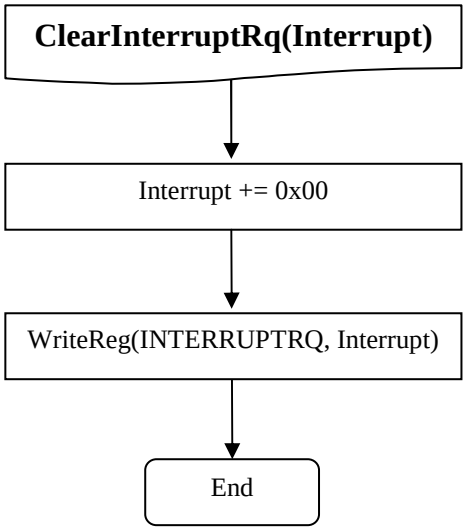
```
void SetBit( const BYTE CLRC632_REG, BYTE bit_nr )
```



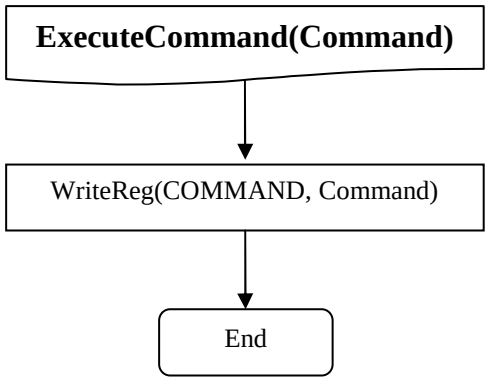

```
BYTE CheckBit( const BYTE CLRC632_REG, BYTE bit_nr )
```



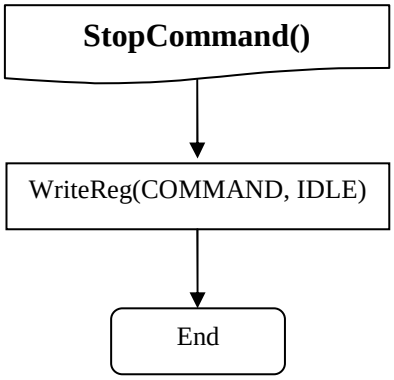
```
void ClearInterruptRq( BYTE interrupt )
```



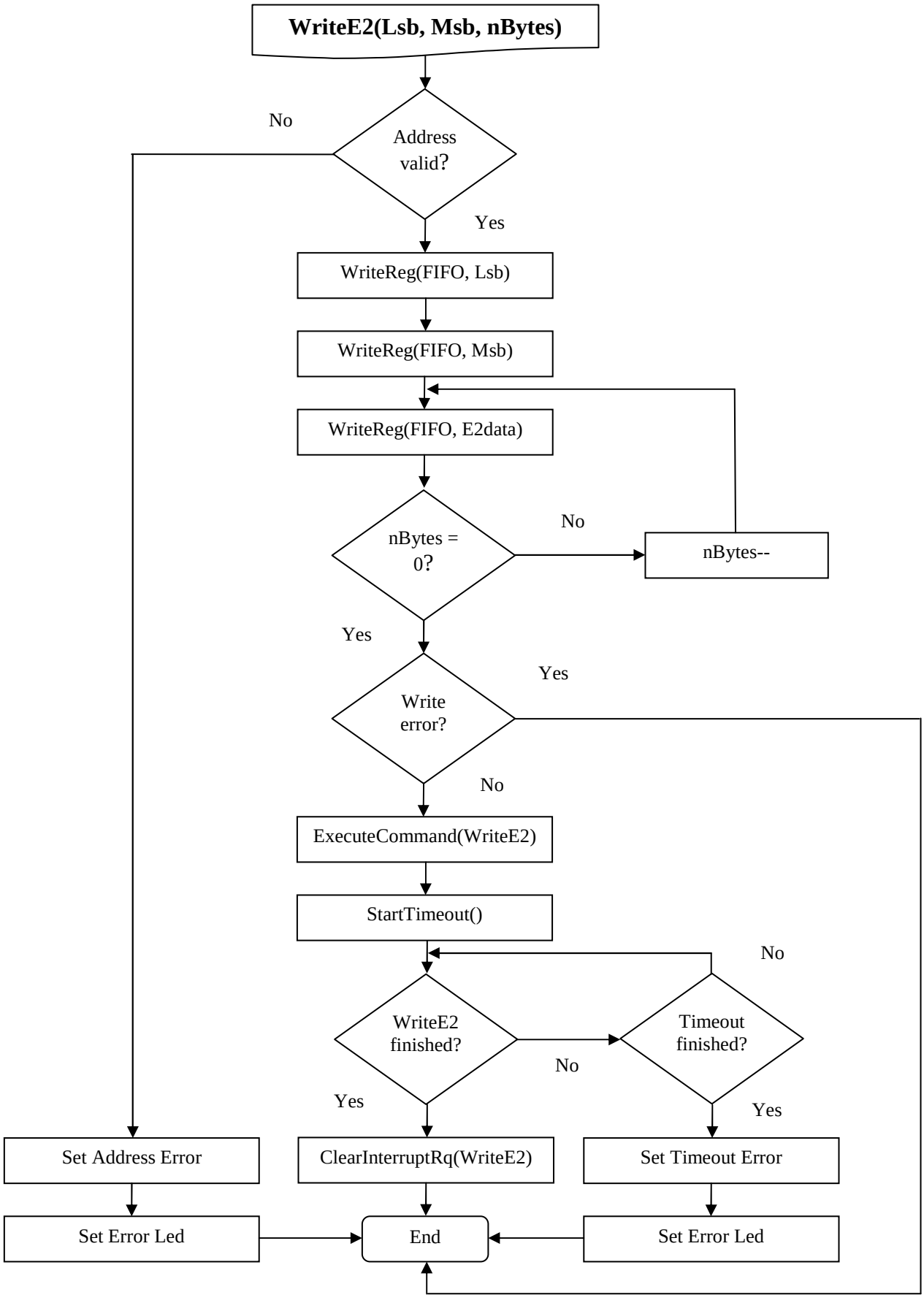

```
void ExecuteCommand( const BYTE CLRC632_COMMAND )
```



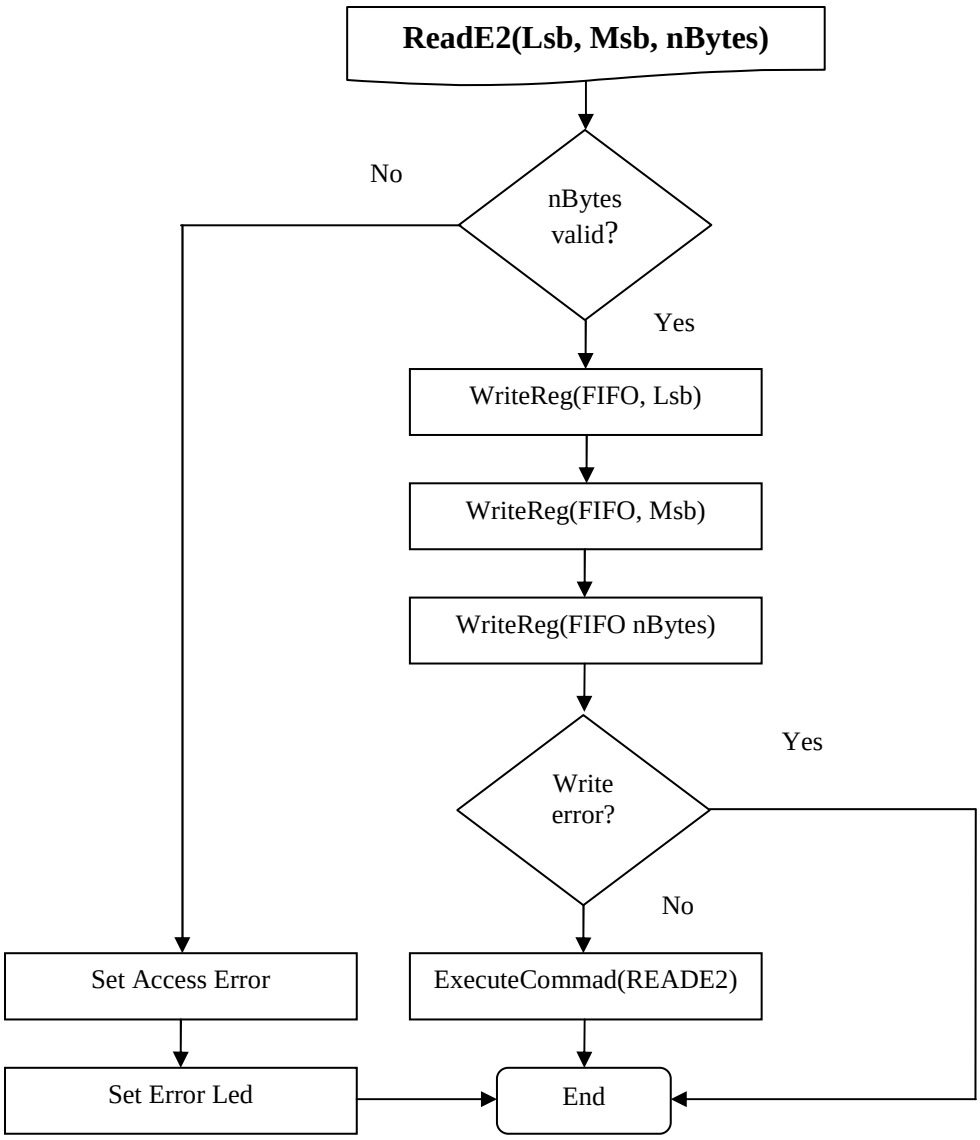

```
void StopCommand( void )
```



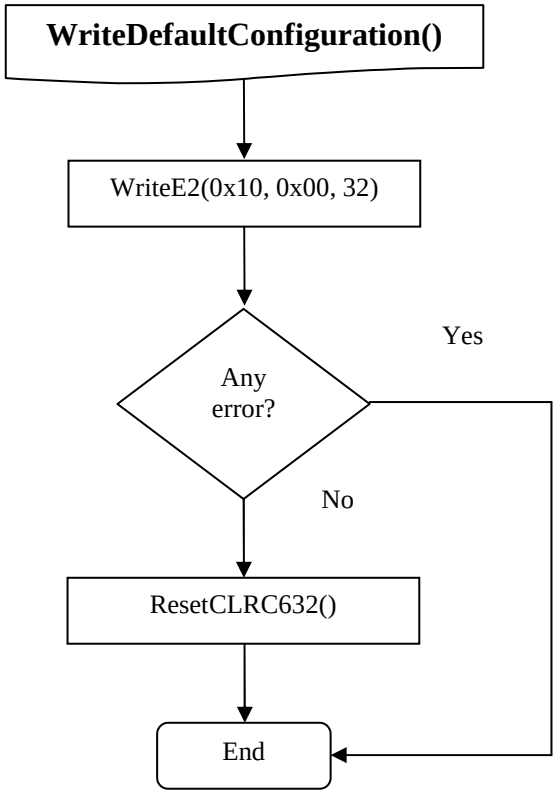
```
void WriteE2( BYTE lsb, BYTE msb, BYTE n_bytes )
```



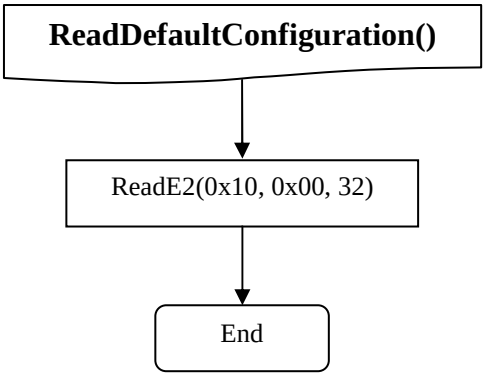
```
void ReadE2( BYTE lsb, BYTE msb, BYTE n_bytes )
```



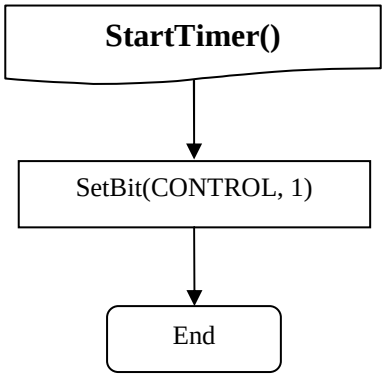
```
void WriteDefaultConfig( void )
```



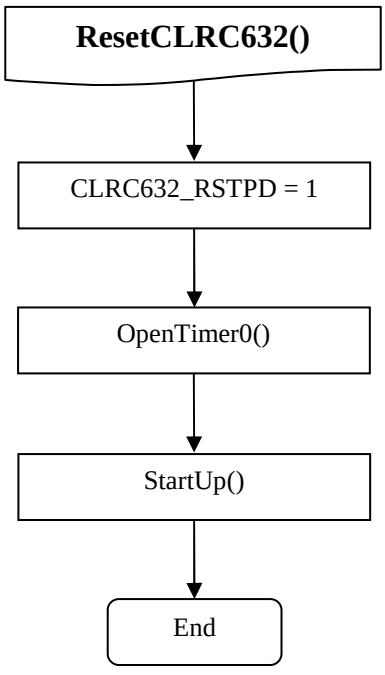

```
void ReadDefaultConfig( void )
```



```
void StartTimer( void )
```

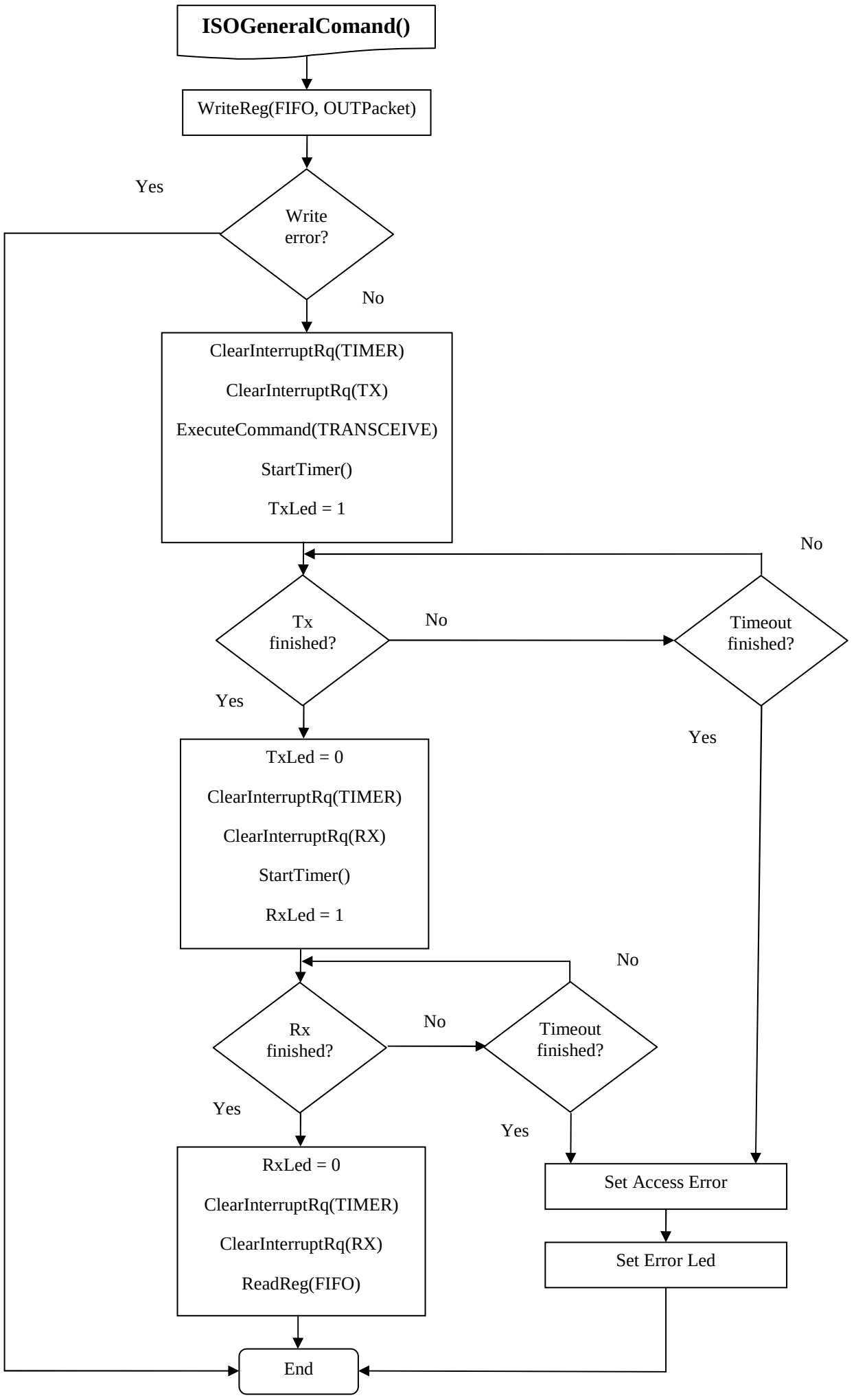



```
void ResetCLRC632( void )
```

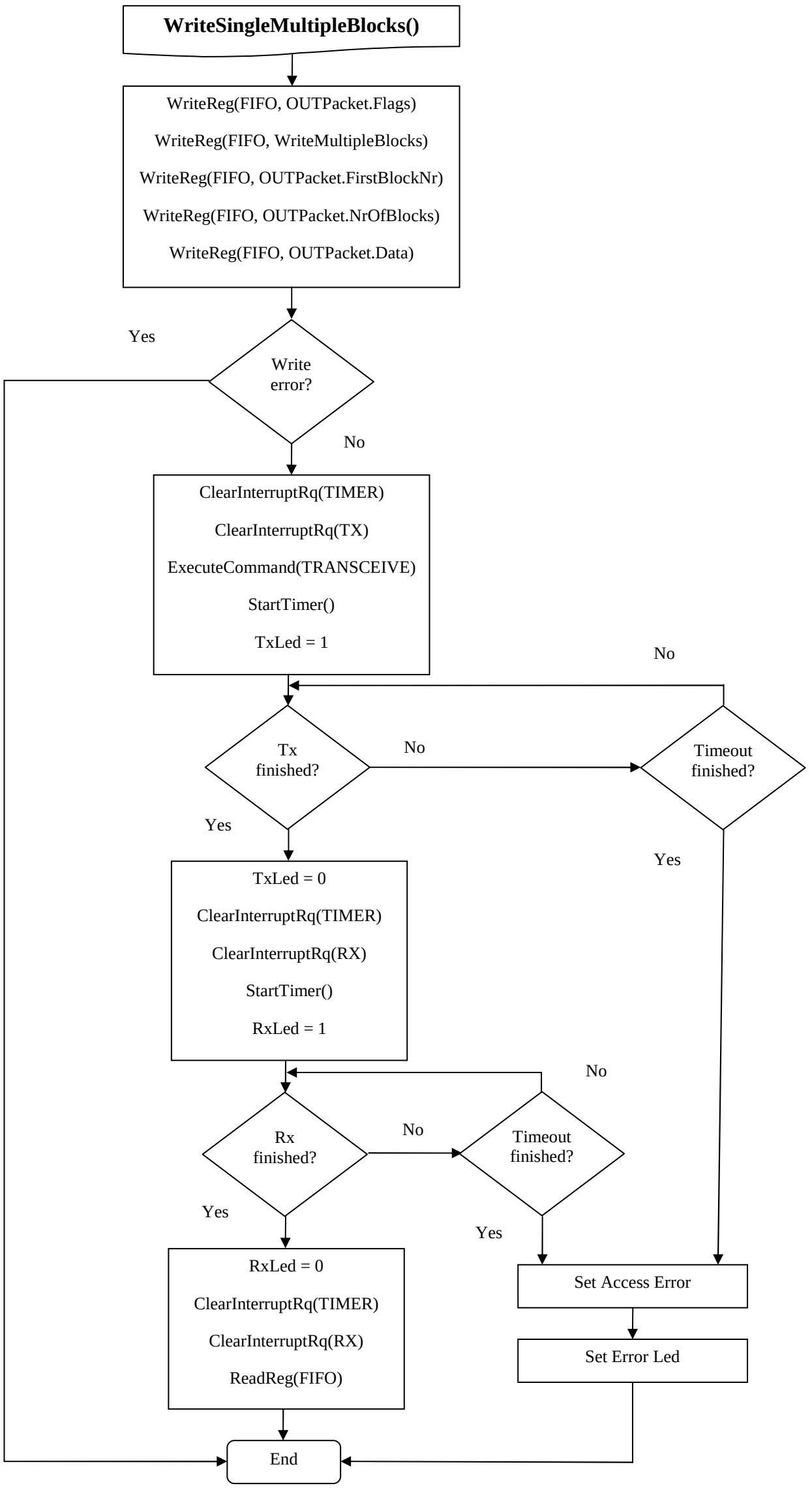


3.3.2. iso15693.c

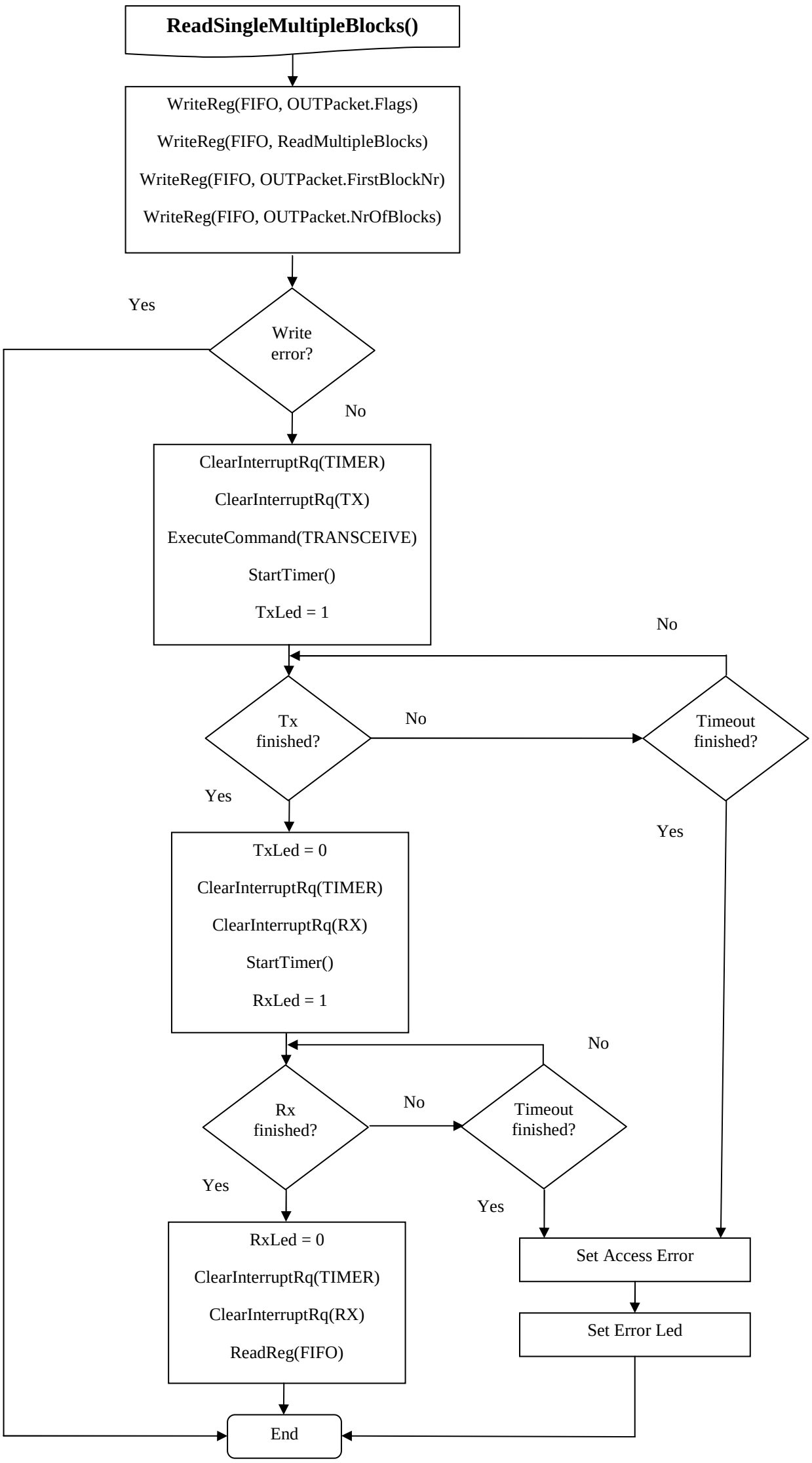
```
void ISOGeneralCommand( void )
```



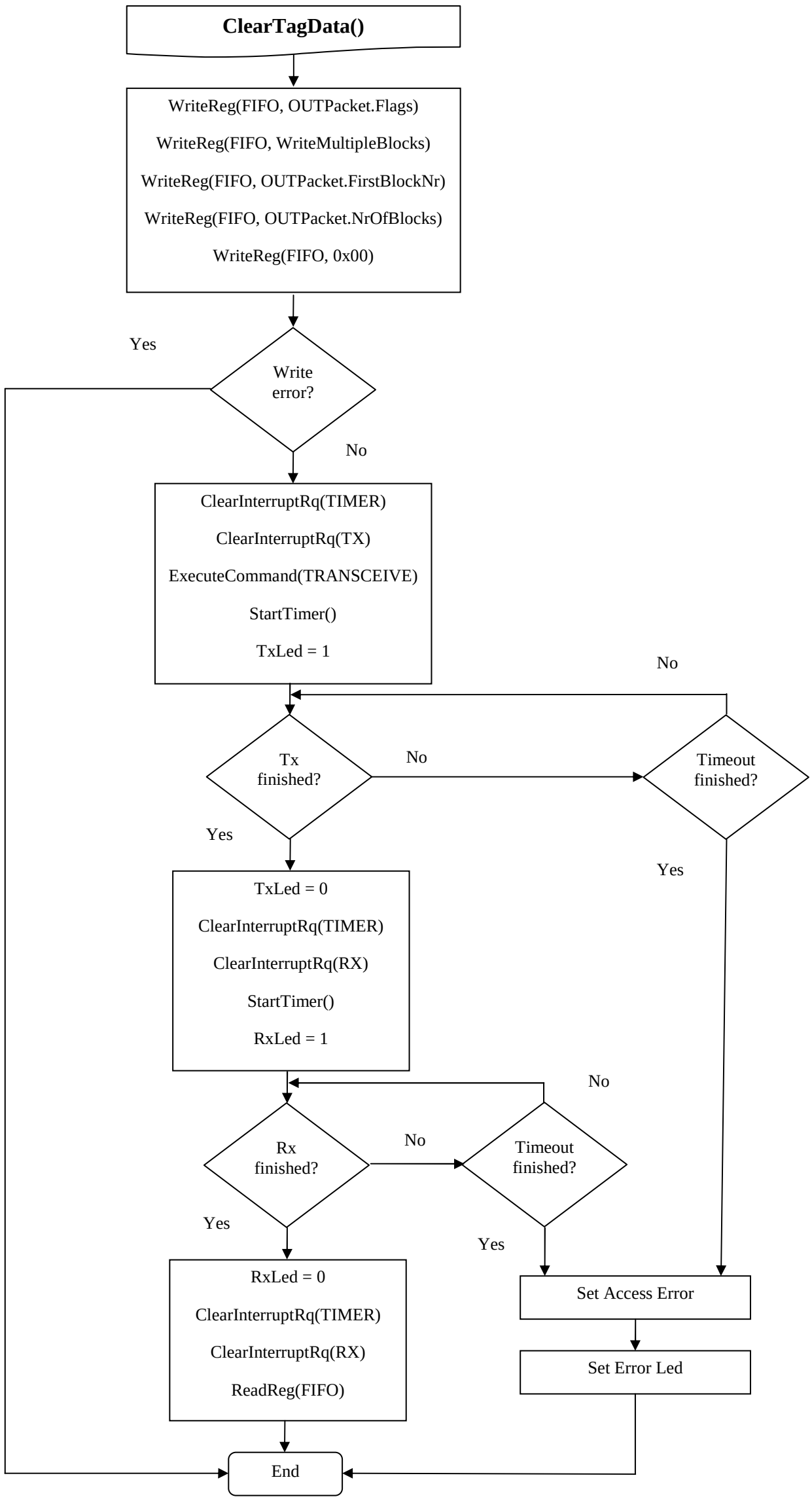
```
void WriteSingleMultipleBlocks( void )
```



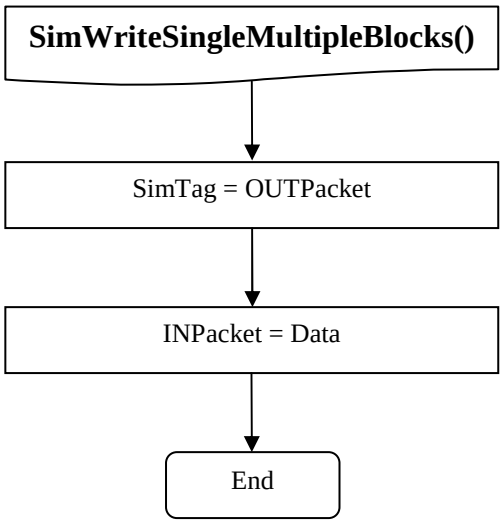
```
void ReadSingleMultipleBlocks( void )
```



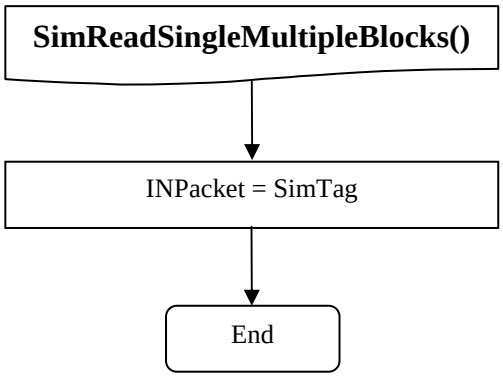
```
void ClearTagData( void )
```



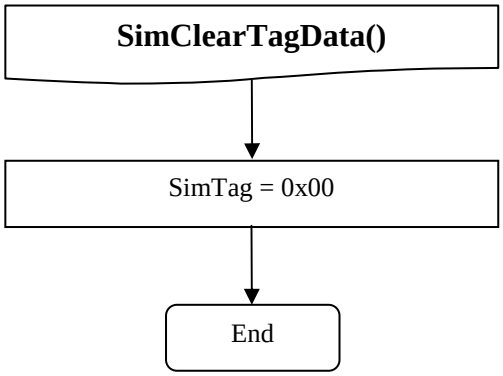
```
void SimWriteSingleMultipleBlocks( void )
```




```
void SimReadSingleMultipleBlocks( void )
```

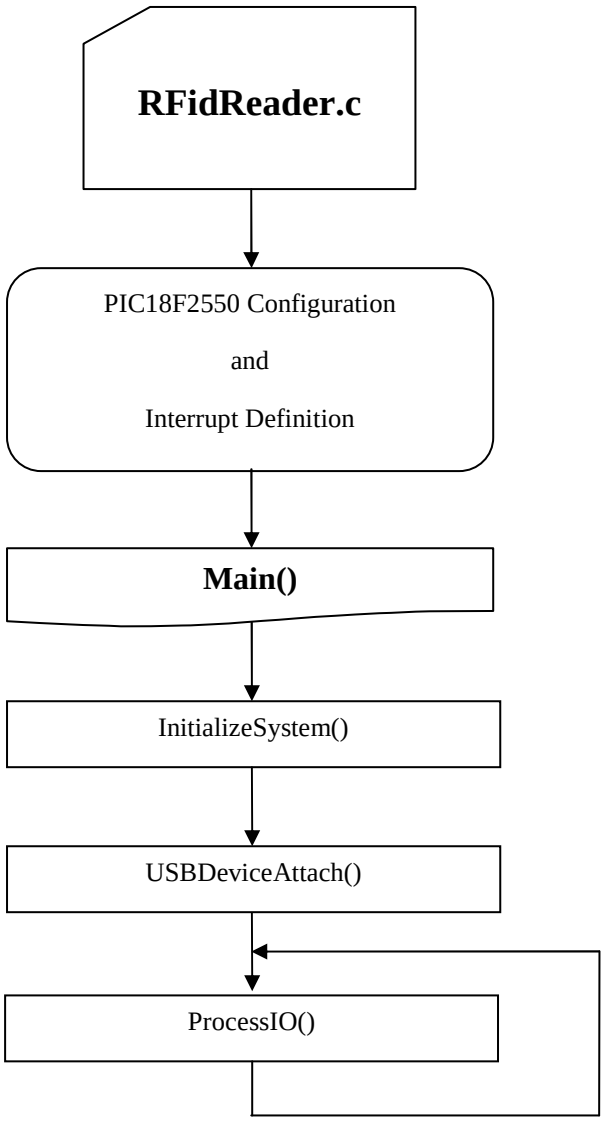



```
void SimClearTagData( void )
```

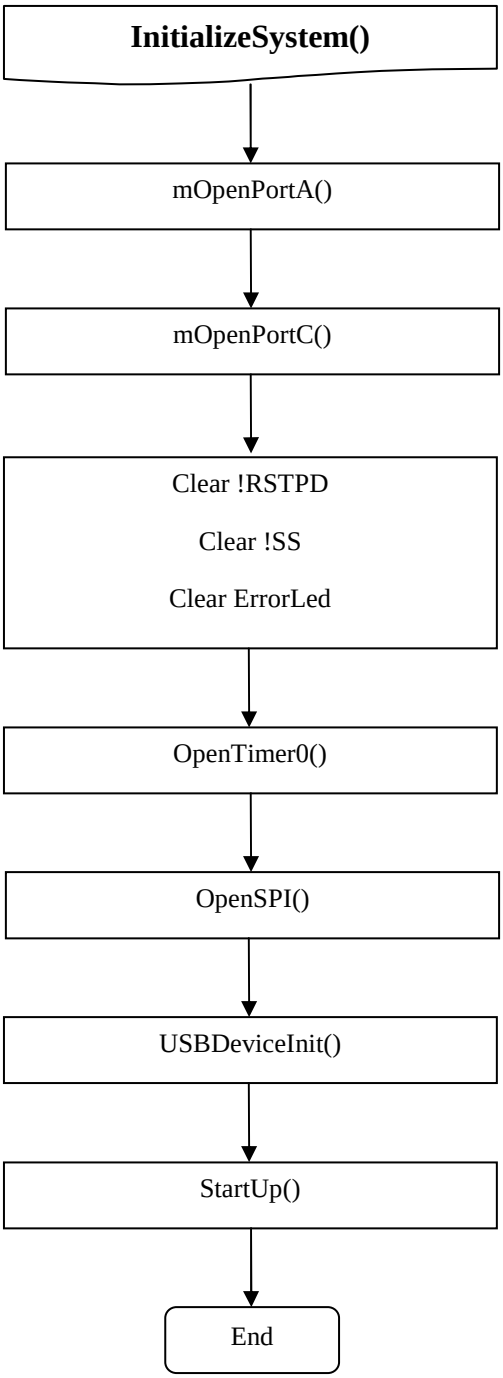


3.3.3. RFidReader.c

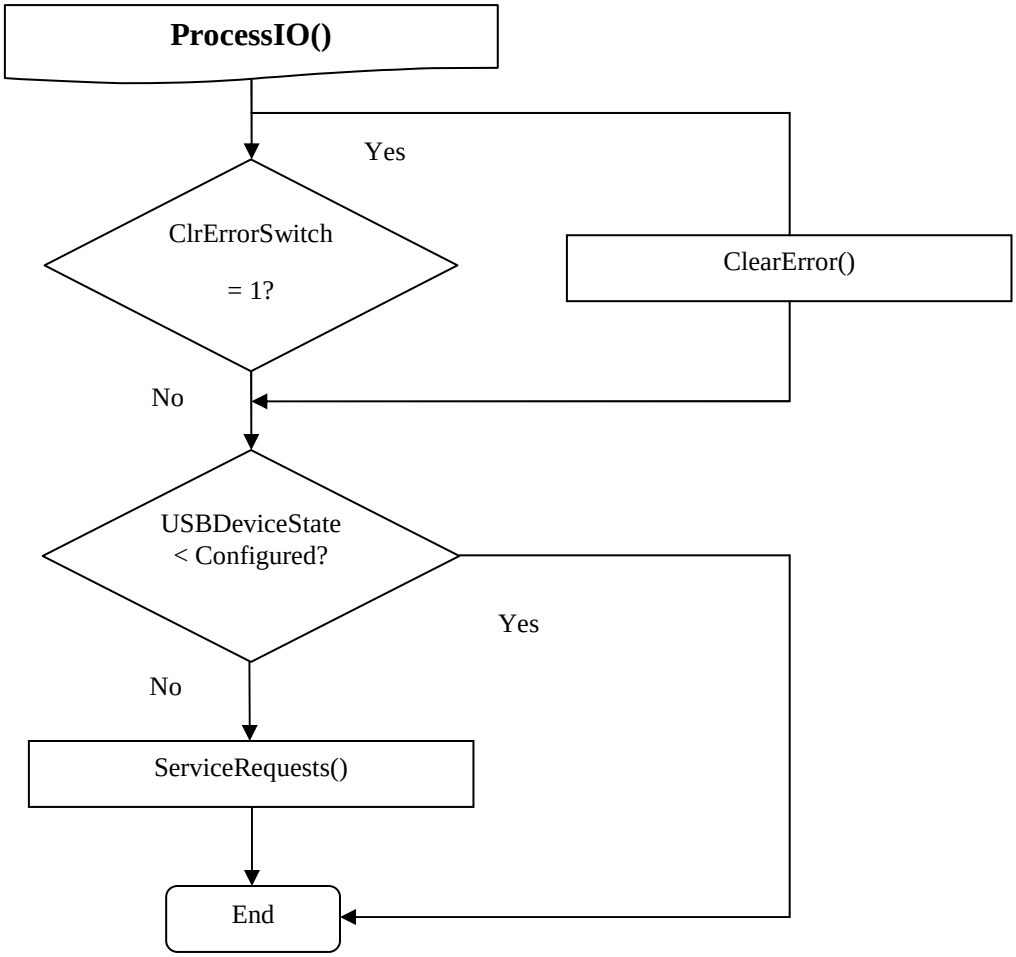
```
void main(void)
```



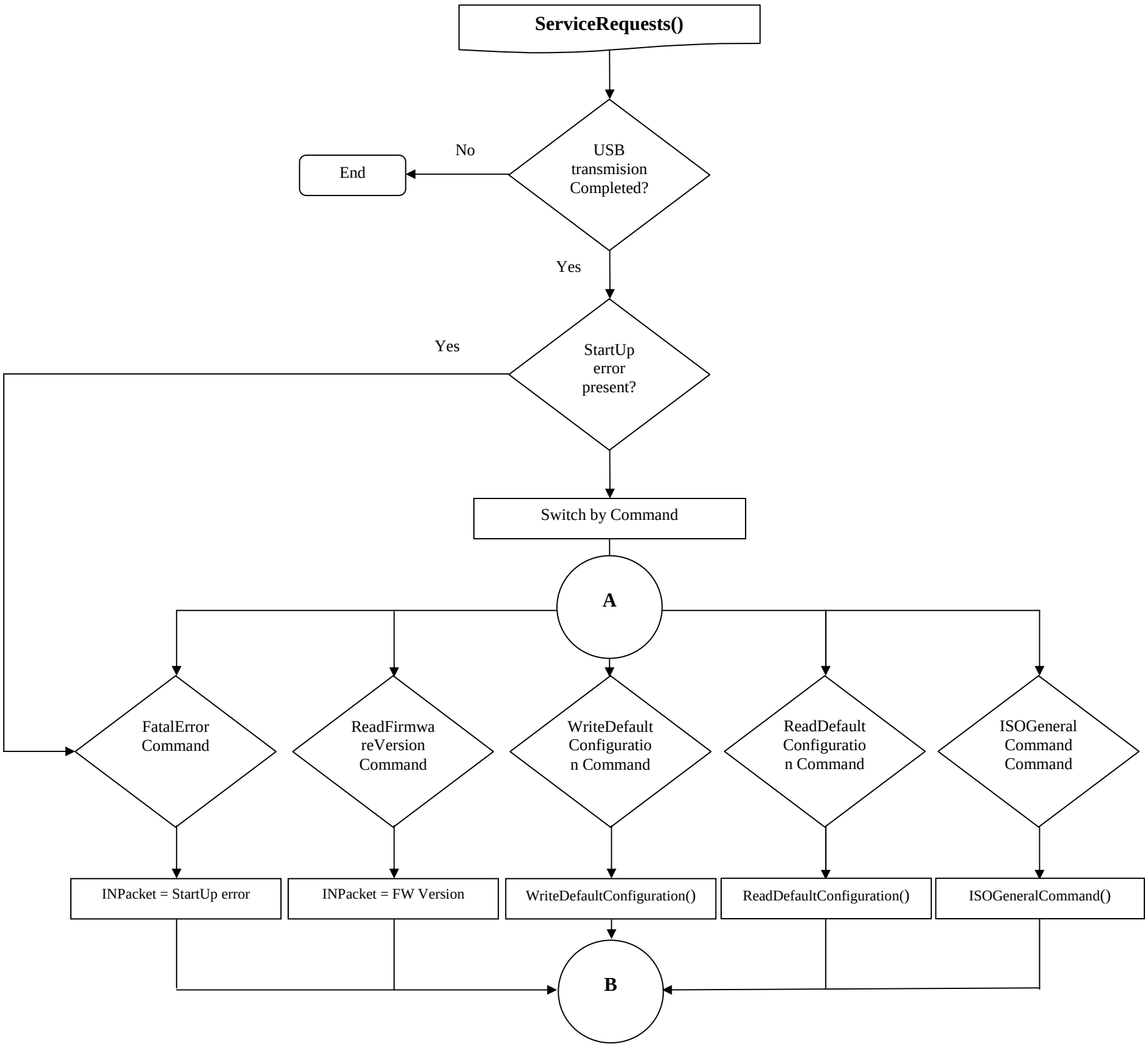
```
void InitializeSystem(void)
```

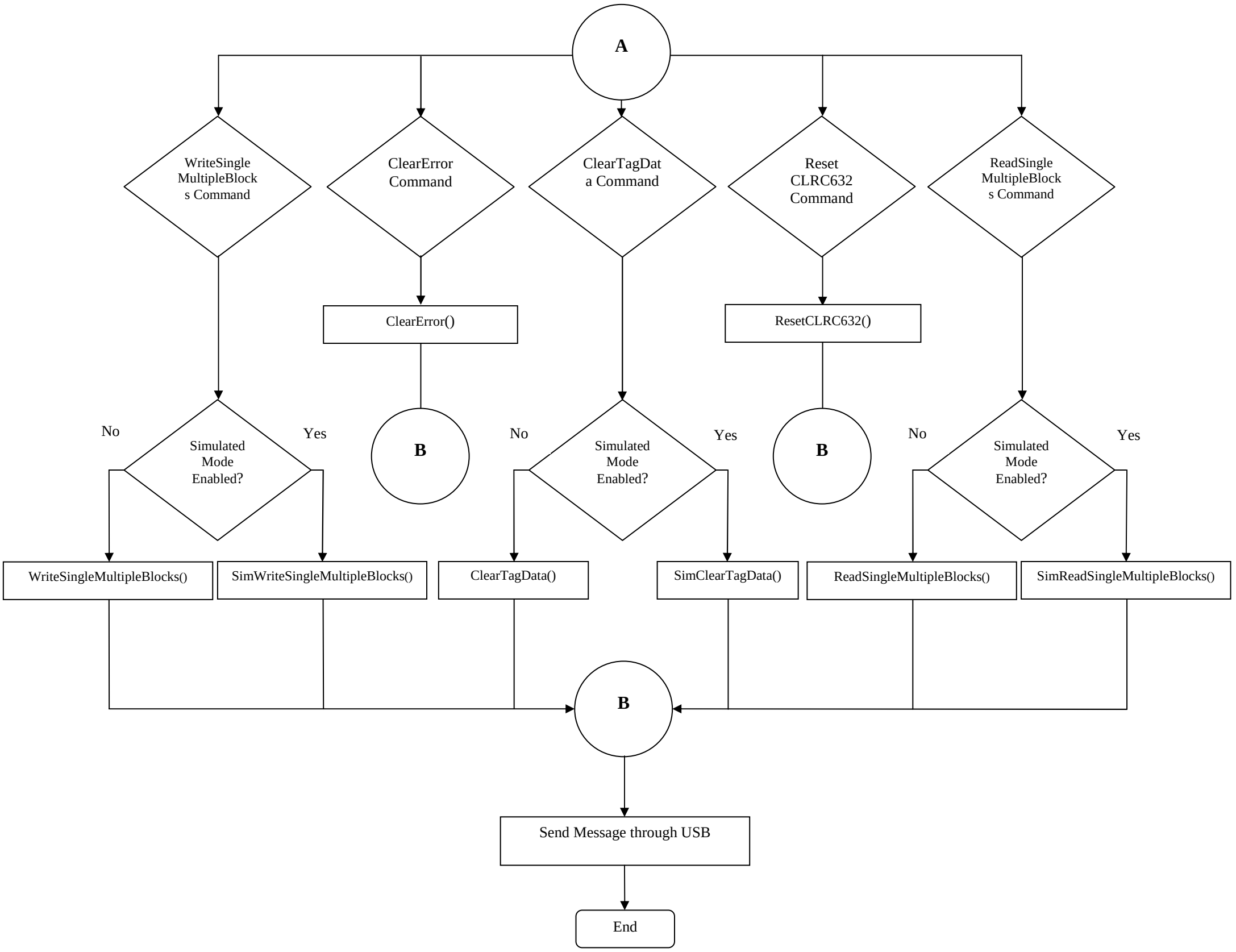


```
void ProcessIO(void)
```



```
ServiceRequests(void)
```

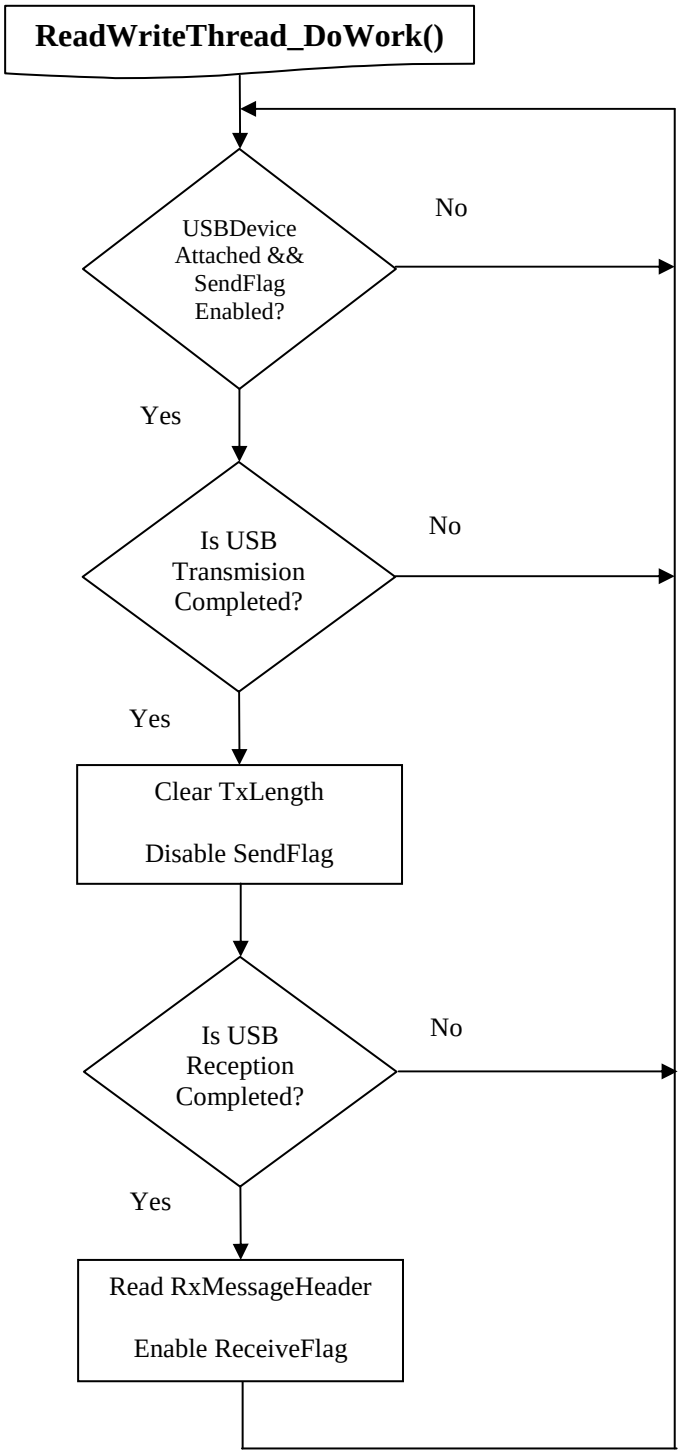



A
WriteSingle MultipleBlock s Command
ClearError Command
ClearTagDat a Command
Reset CLRC632 Command
ReadSingle MultipleBlock s Command
ClearError()
ResetCLRC632()
B
B
B
No
Yes
Simulated Mode Enabled?
No
Yes
Simulated Mode Enabled?
No
Yes
Simulated Mode Enabled?
WriteSingleMultipleBlocks()
SimWriteSingleMultipleBlocks()
ClearTagData()
SimClearTagData()
ReadSingleMultipleBlocks()
SimReadSingleMultipleBlocks()
B
Send Message through USB
End

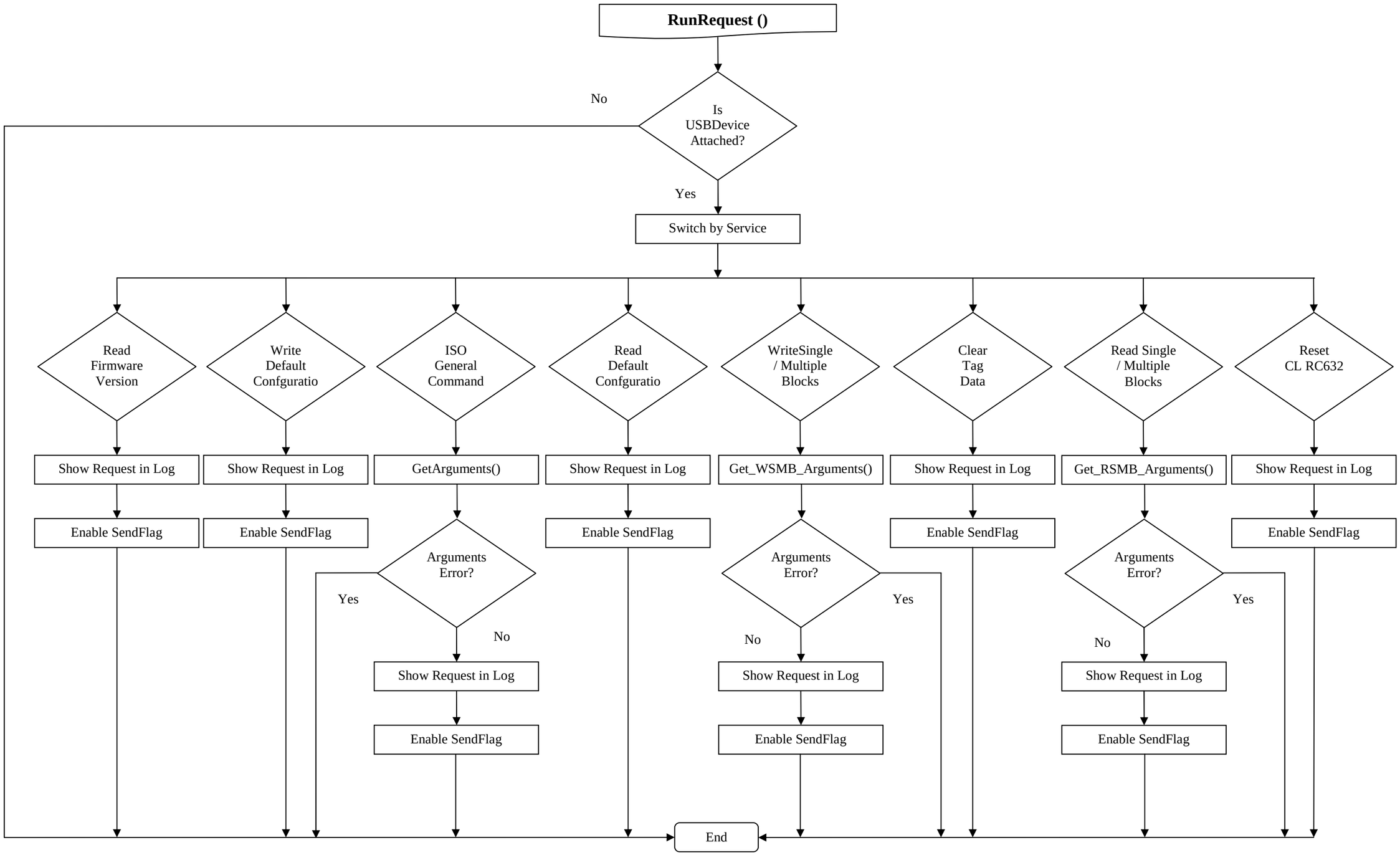
3.4. Diagramas del Software

3.4.1. Form1.h

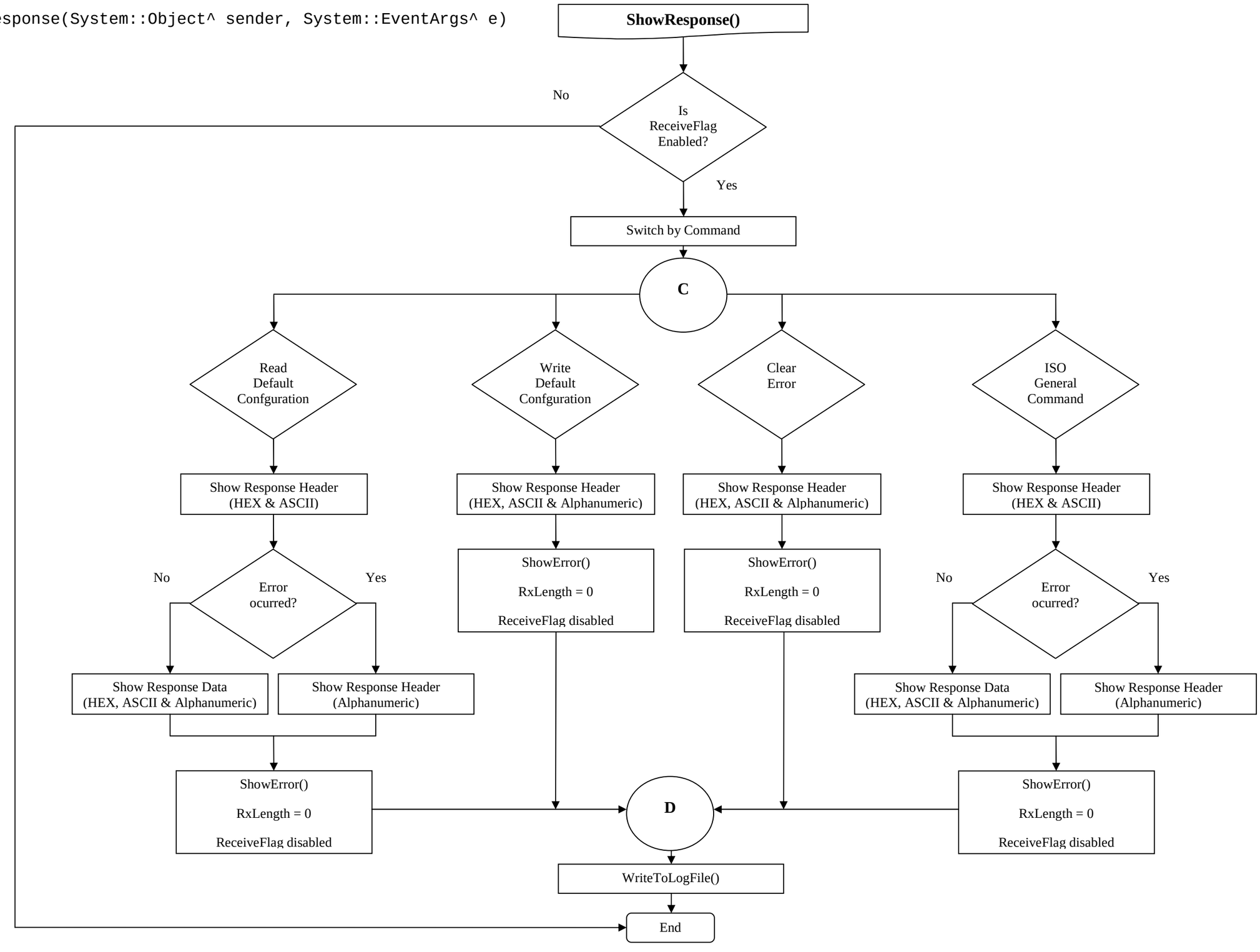
```
Void ReadWriteThread_DoWork(System::Object^ sender,System::ComponentModel
::DoWorkEventArgs^  e)
```

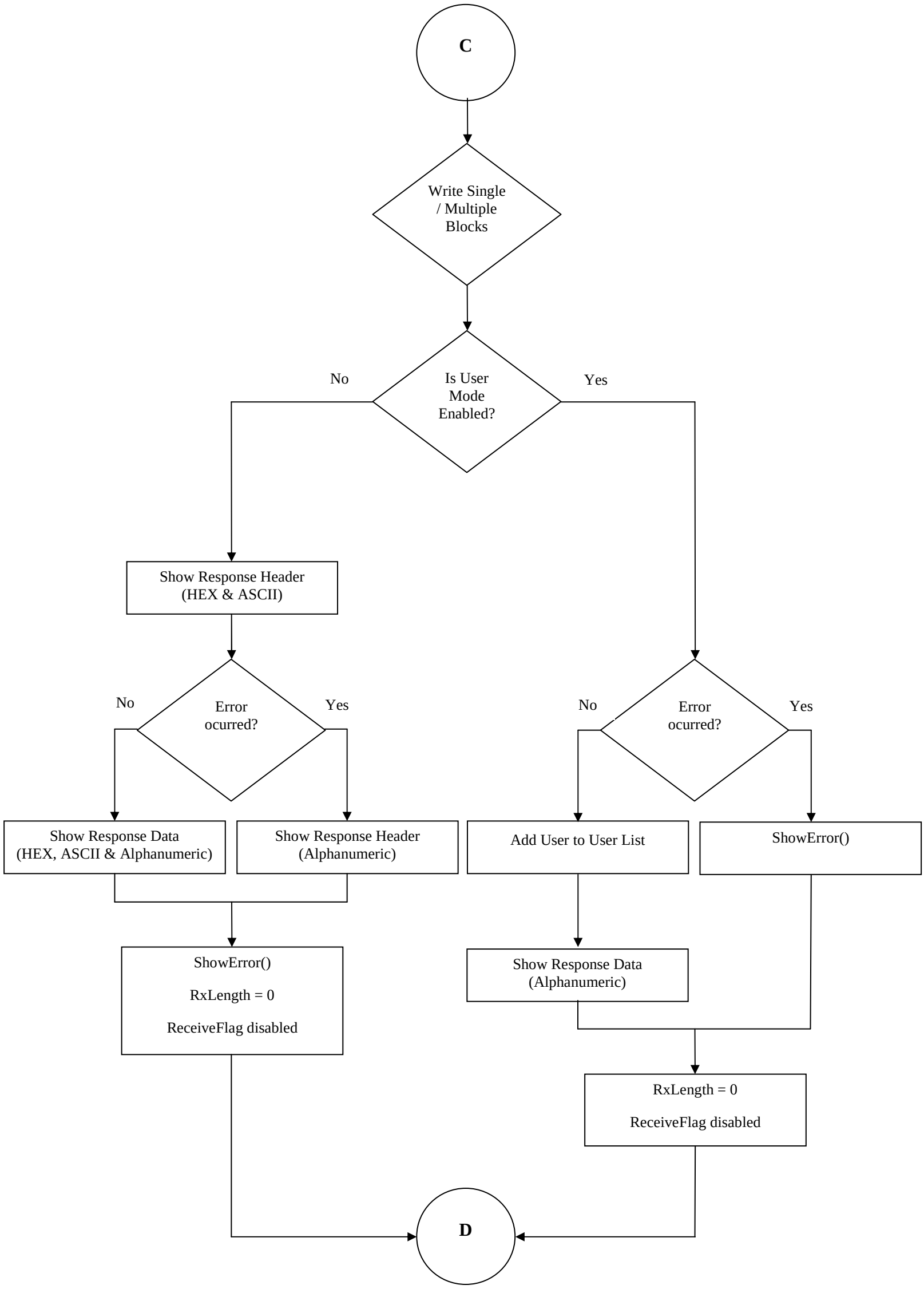


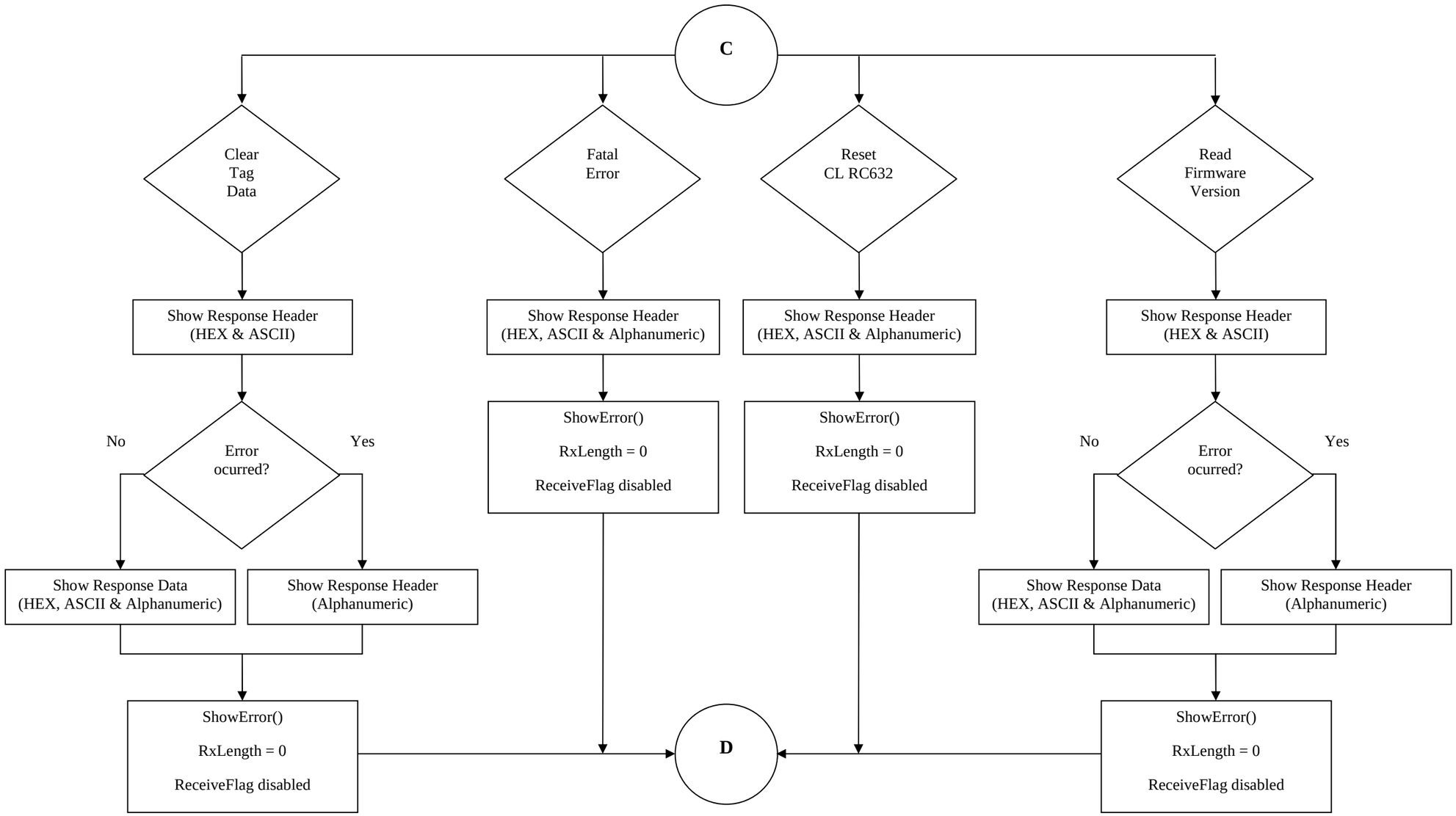
```
Void RunRequest(System::Object^ sender,System::EventArgs^ e)
```

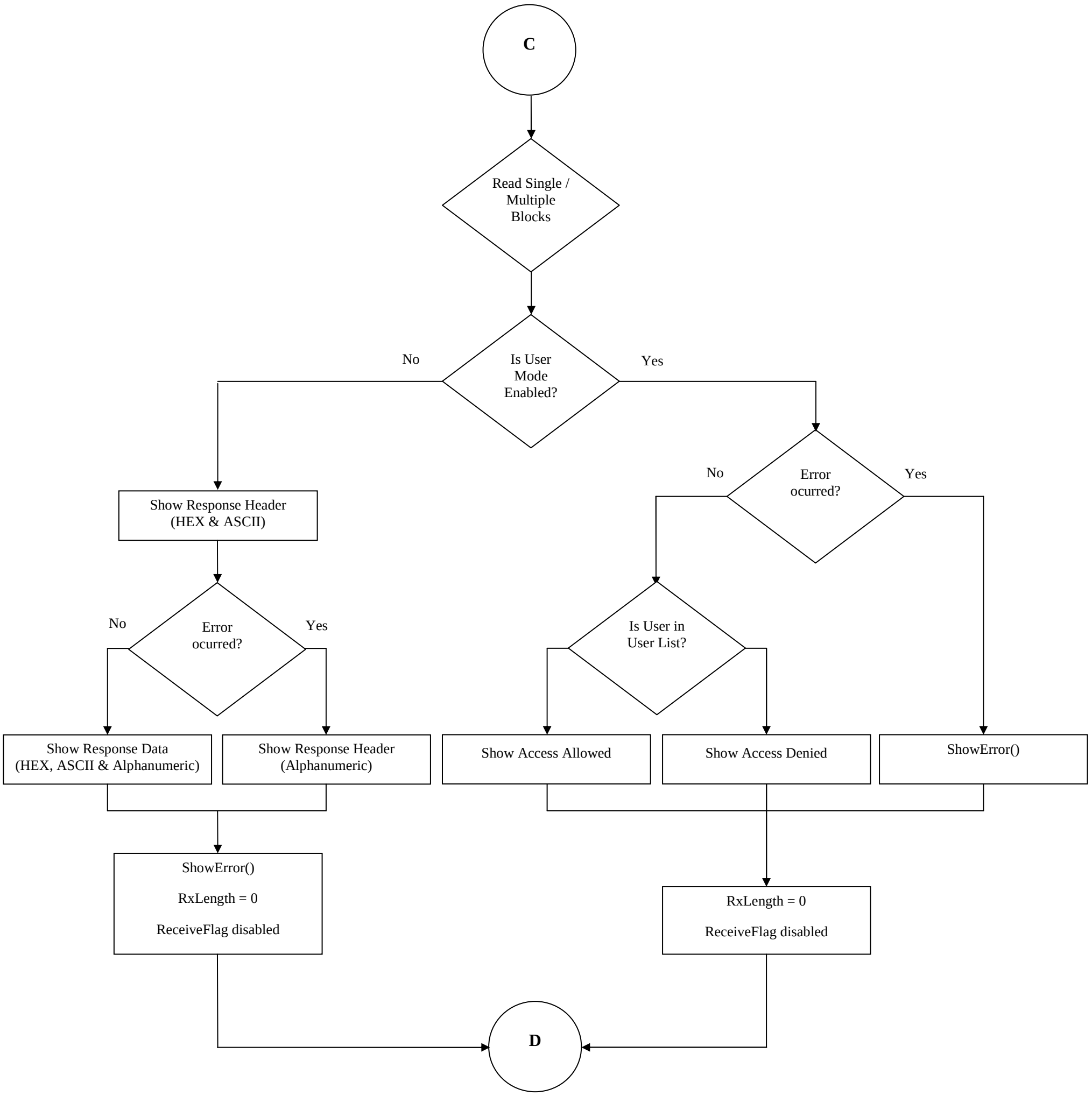


```
Void ShowResponse(System::Object^ sender, System::EventArgs^ e)
```

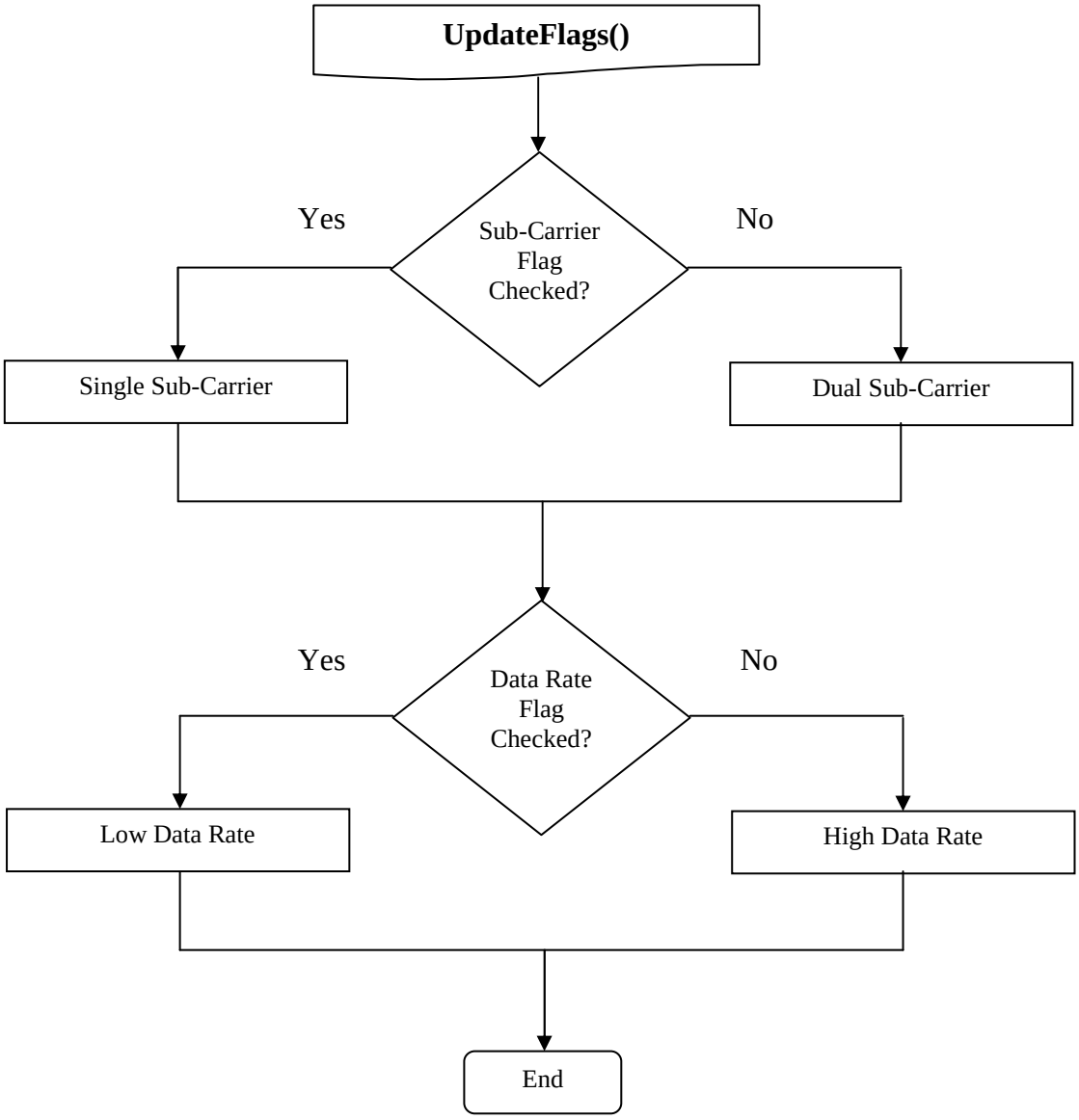



C
Write Single / Multiple Blocks
Is User Mode Enabled?
No
Yes
Show Response Header (HEX & ASCII)
Error ocurred?
No
Yes
Show Response Data (HEX, ASCII & Alphanumeric)
Show Response Header (Alphanumeric)
ShowError()
RxLength = 0
ReceiveFlag disabled
Error ocurred?
No
Yes
Add User to User List
ShowError()
Show Response Data (Alphanumeric)
RxLength = 0
ReceiveFlag disabled
D

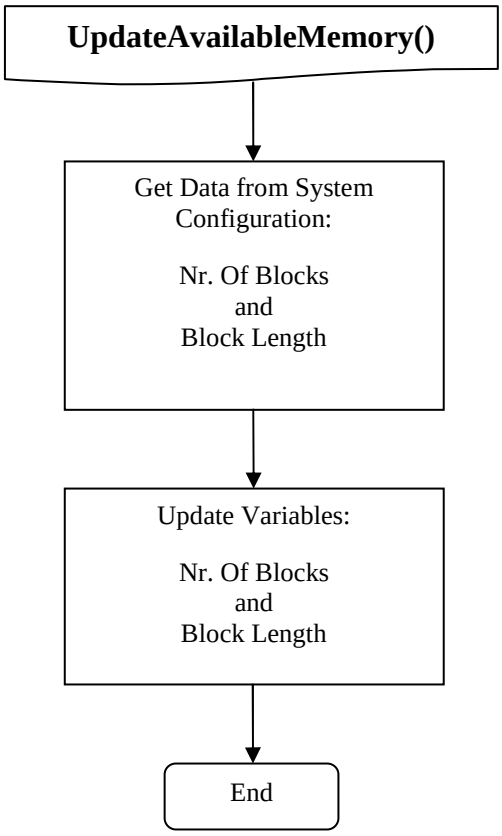

C
Clear Tag Data
Fatal Error
Reset CL RC632
Read Firmware Version
Show Response Header (HEX & ASCII)
Show Response Header (HEX, ASCII & Alphanumeric)
Show Response Header (HEX, ASCII & Alphanumeric)
Show Response Header (HEX & ASCII)
ShowError() RxLength = 0 ReceiveFlag disabled
ShowError() RxLength = 0 ReceiveFlag disabled
No
Error ocurred?
Yes
No
Error ocurred?
Yes
Show Response Data (HEX, ASCII & Alphanumeric)
Show Response Header (Alphanumeric)
Show Response Data (HEX, ASCII & Alphanumeric)
Show Response Header (Alphanumeric)
ShowError() RxLength = 0 ReceiveFlag disabled
D
ShowError() RxLength = 0 ReceiveFlag disabled


C
Read Single / Multiple Blocks
Is User Mode Enabled?
No
Yes
Show Response Header (HEX & ASCII)
Error ocurred?
No
Yes
Show Response Data (HEX, ASCII & Alphanumeric)
Show Response Header (Alphanumeric)
ShowError()
RxLength = 0
ReceiveFlag disabled
Error ocurred?
No
Yes
Is User in User List?
Show Access Allowed
Show Access Denied
ShowError()
RxLength = 0
ReceiveFlag disabled
D

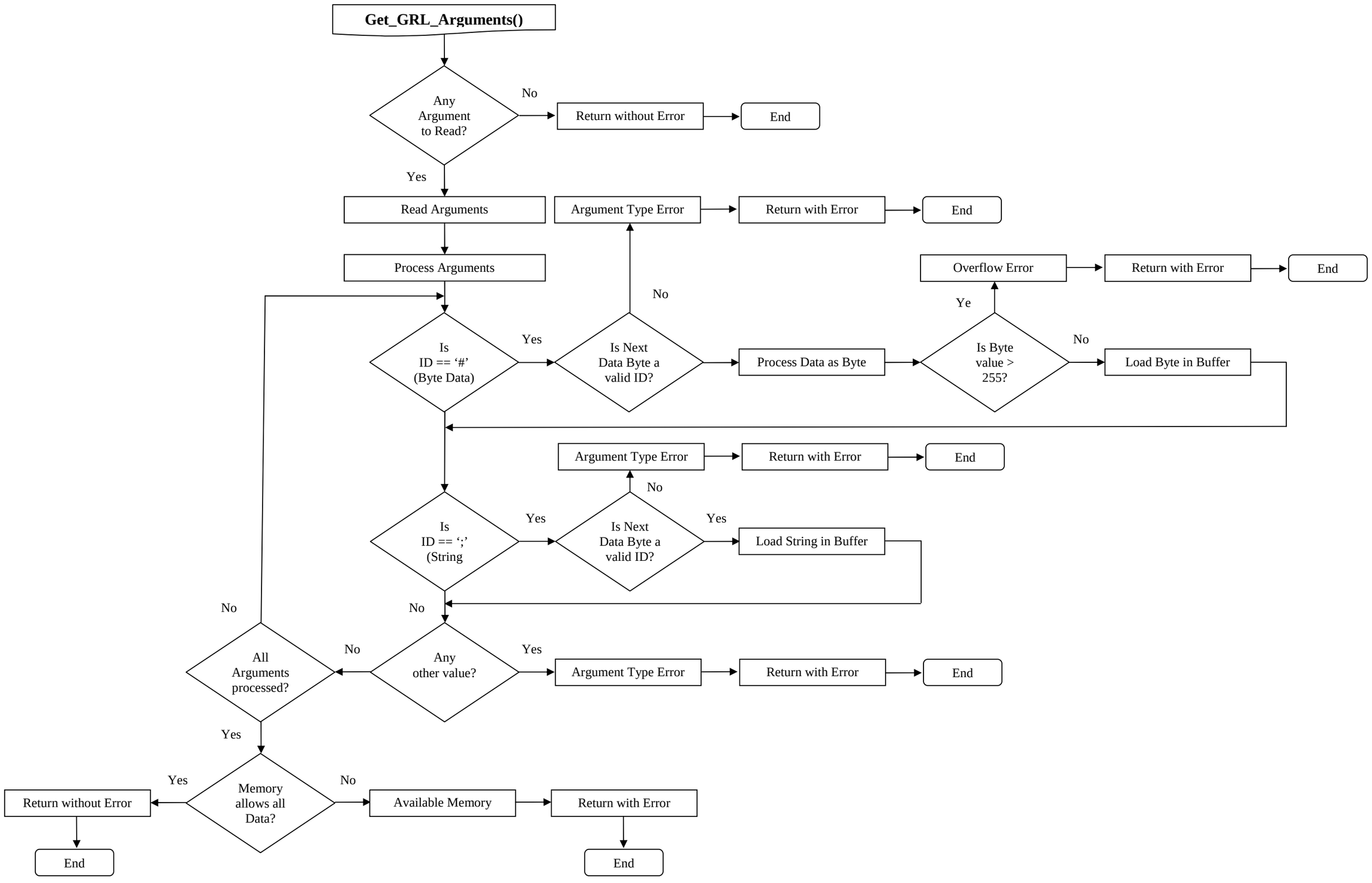
```
Void UpdateFlags(System::Object^sender, System::EventArgs^e)
```



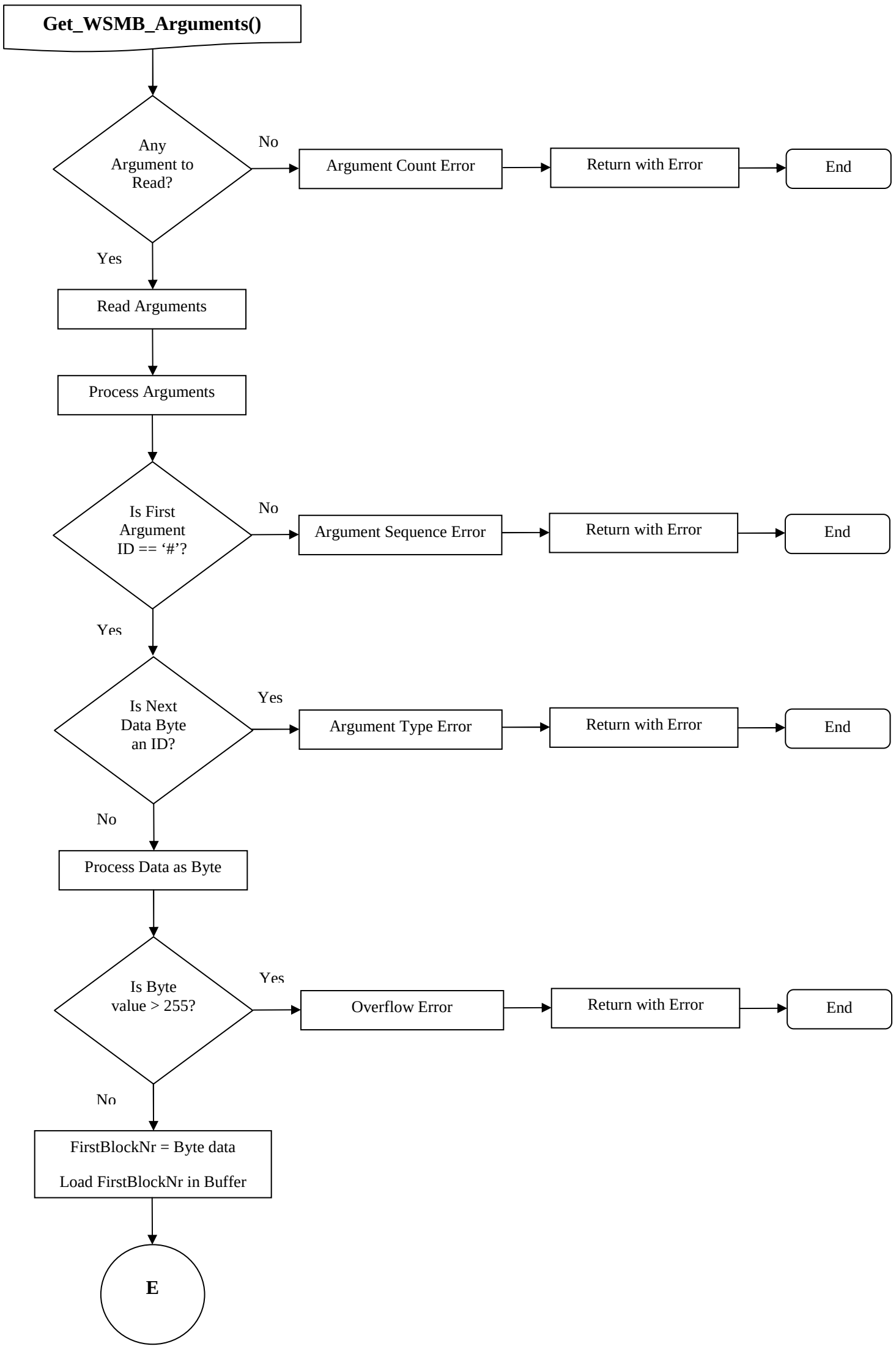
```
Void UpdateAvailableMemory(System::Object^  sender, System::Windows::
Forms::KeyPressEventArgs^  e)
```

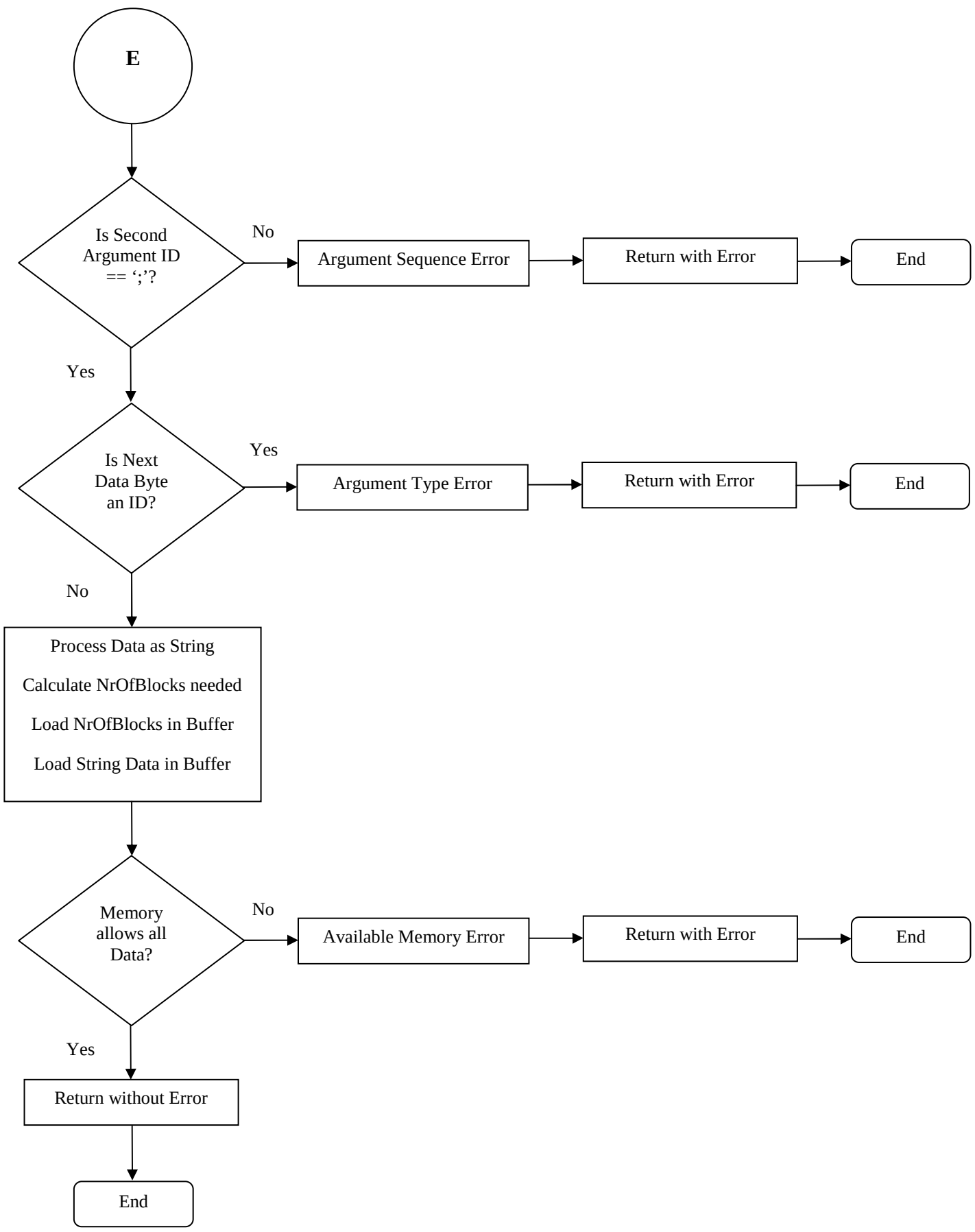


```
bool Get_GRL_Arguments(void)
```

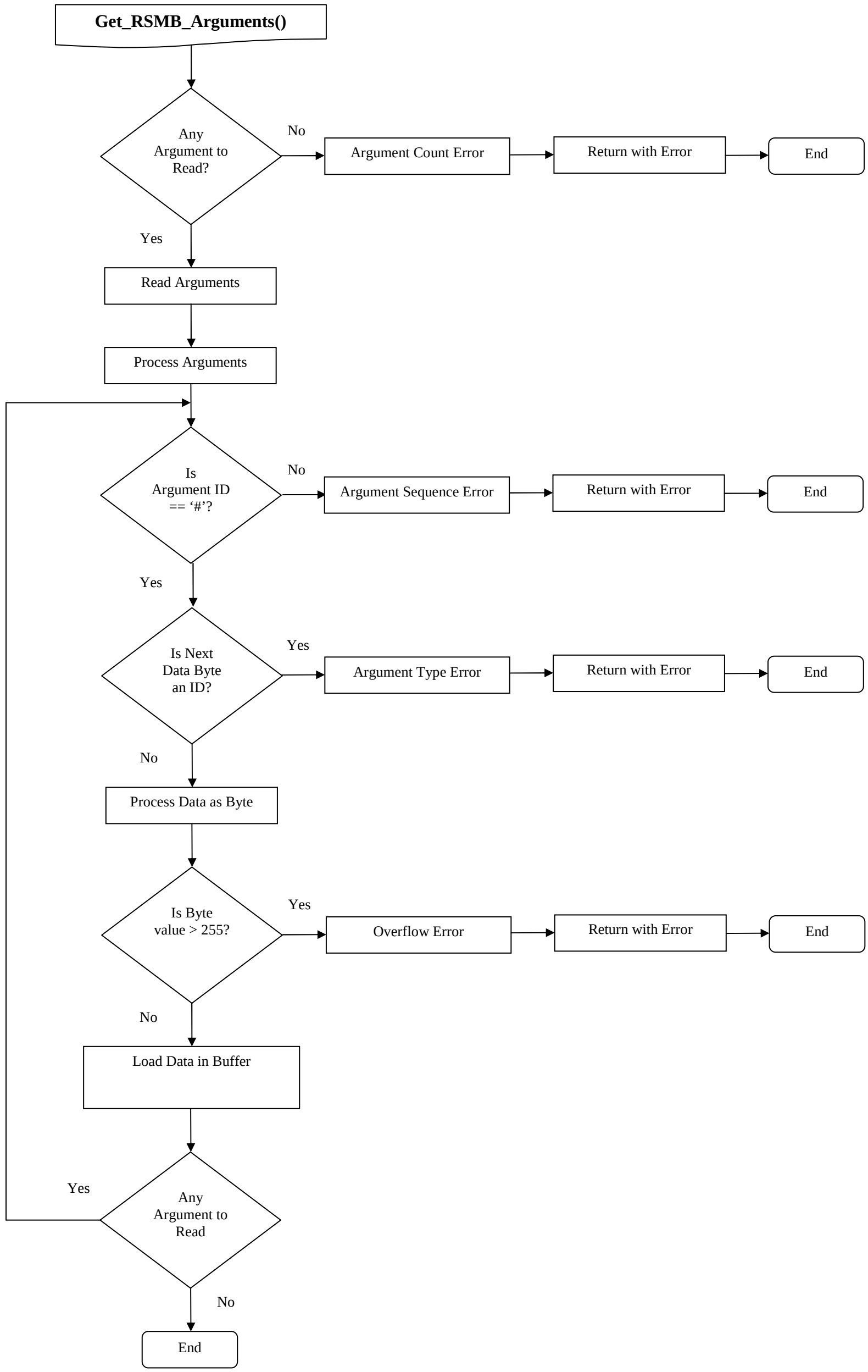


```
bool Get_WSMB_Arguments(void)
```

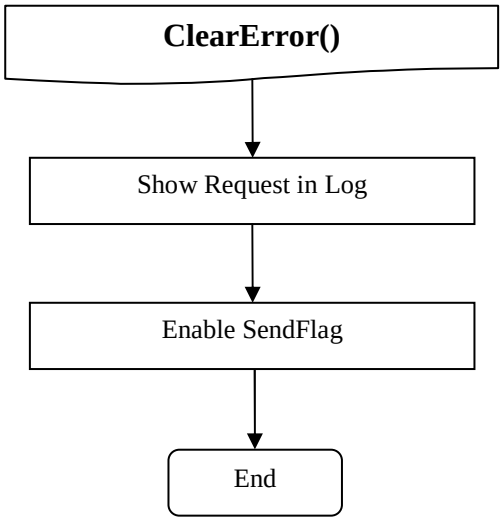



E
Is Second Argument ID == ';'?
No
Argument Sequence Error
Return with Error
End
Yes
Is Next Data Byte an ID?
Yes
Argument Type Error
Return with Error
End
No
Process Data as String
Calculate NrOfBlocks needed
Load NrOfBlocks in Buffer
Load String Data in Buffer
Memory allows all Data?
No
Available Memory Error
Return with Error
End
Yes
Return without Error
End

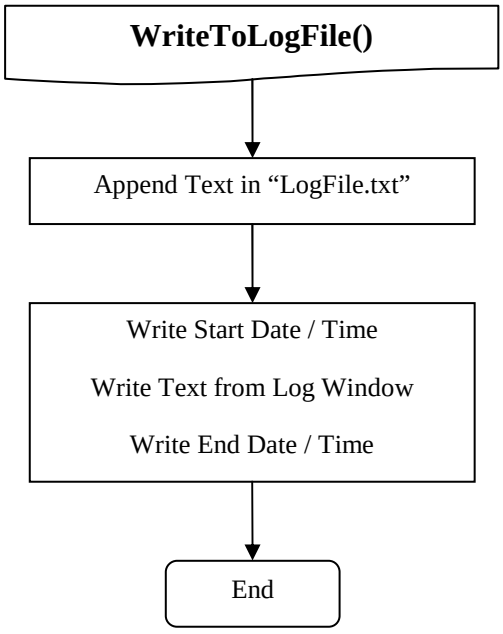
```
bool Get_RSMB_Arguments(void)
```



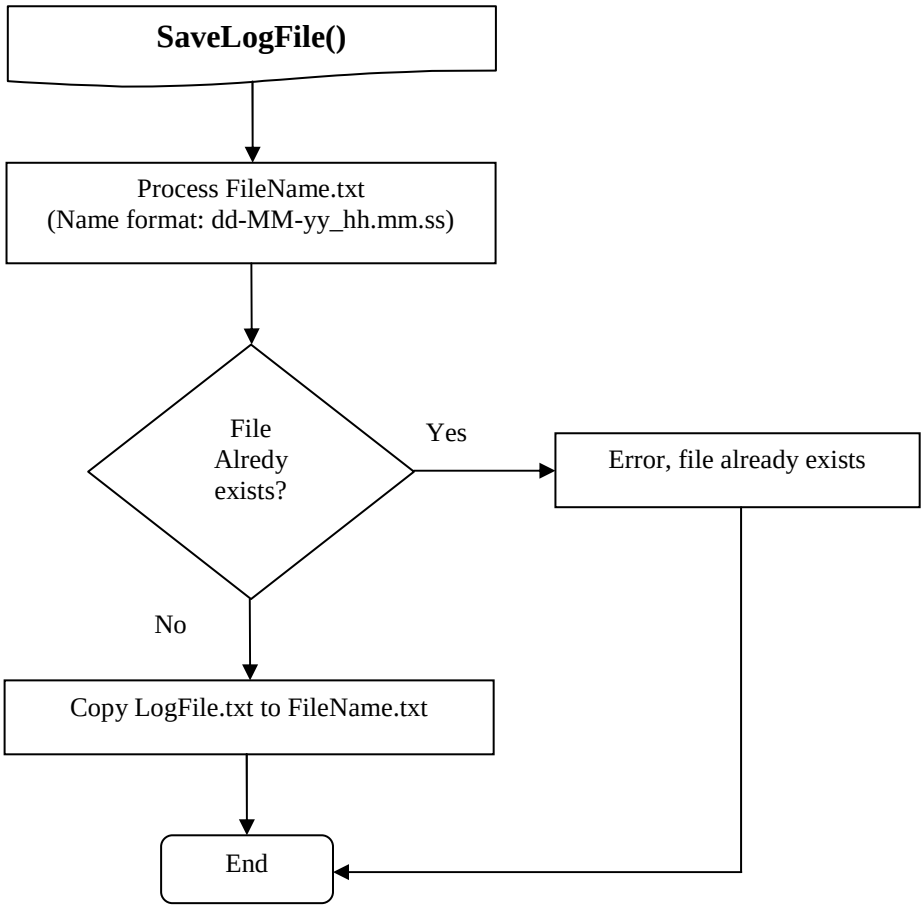
```
Void ClearError(System::Object^  sender, System::EventArgs^  e)
```



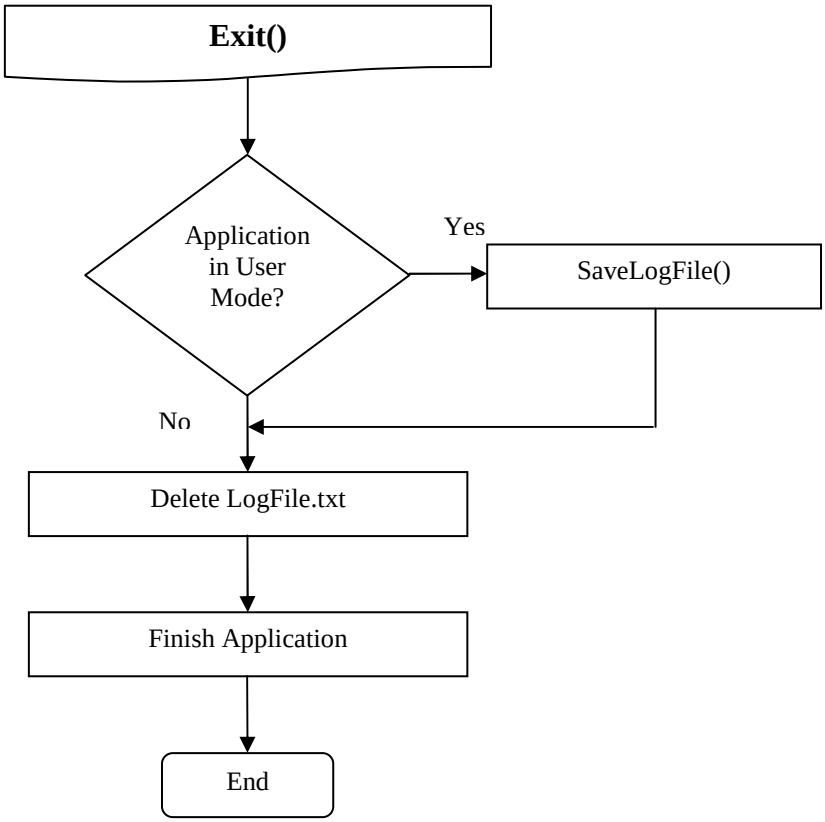

```
void WriteToLogFile(void)
```



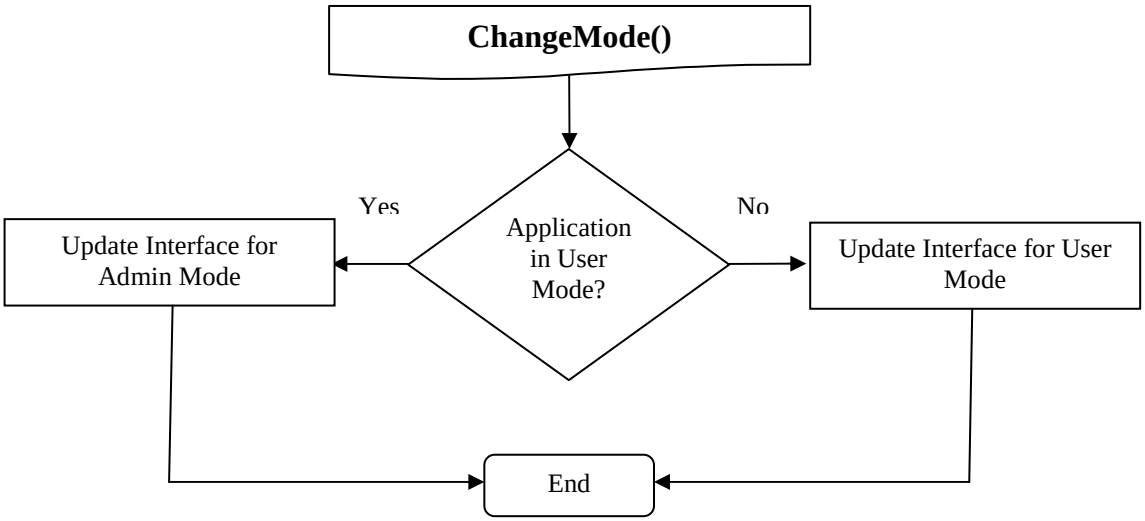
```
void SaveLogFile(void)
```



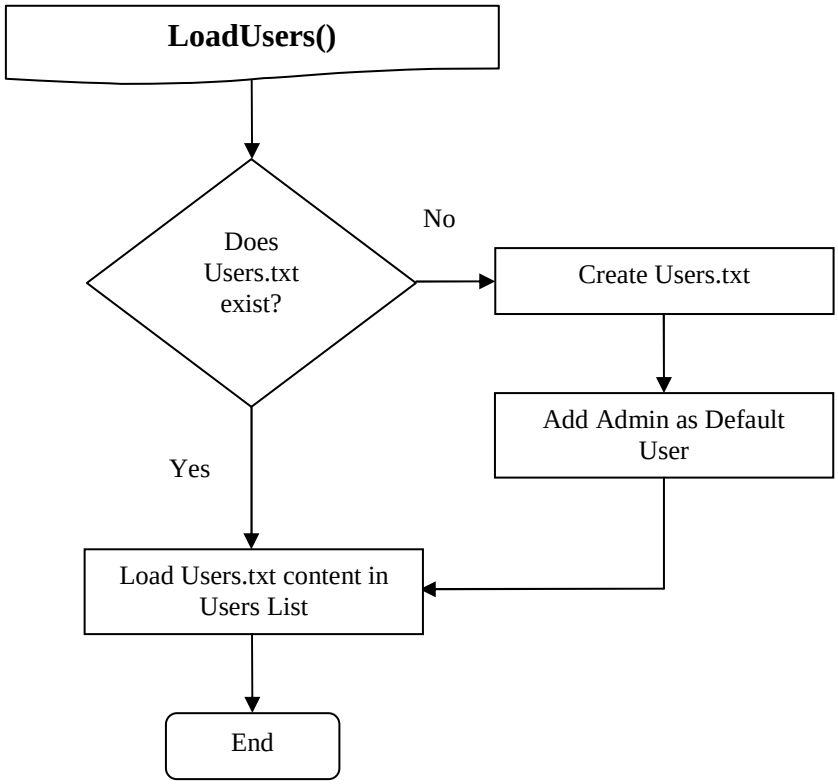
```
Void Exit(System::Object^  sender, System::EventArgs^  e)
```



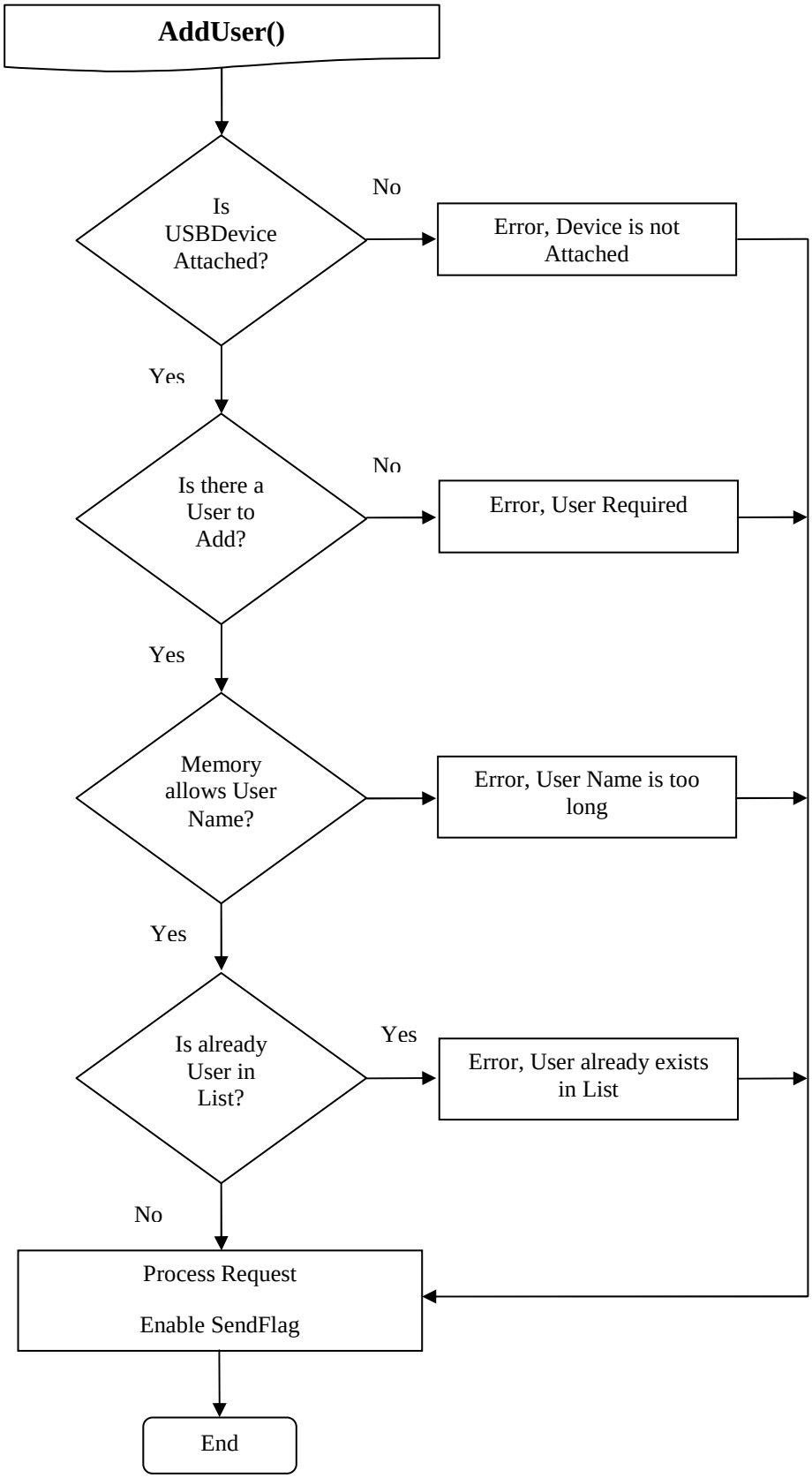

```
System::Void ChangeMode(System::Object^  sender, System::EventArgs^  e)
```



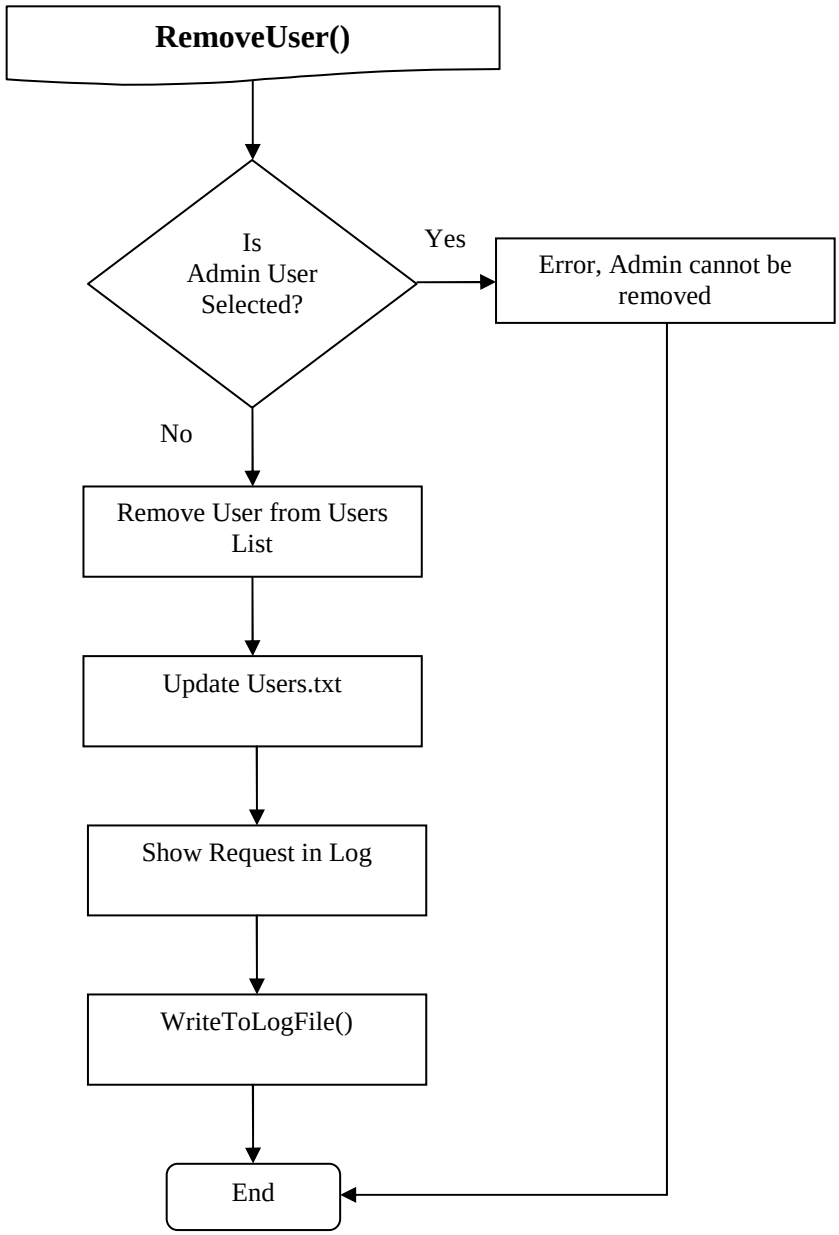
```
void LoadUsers(void)
```



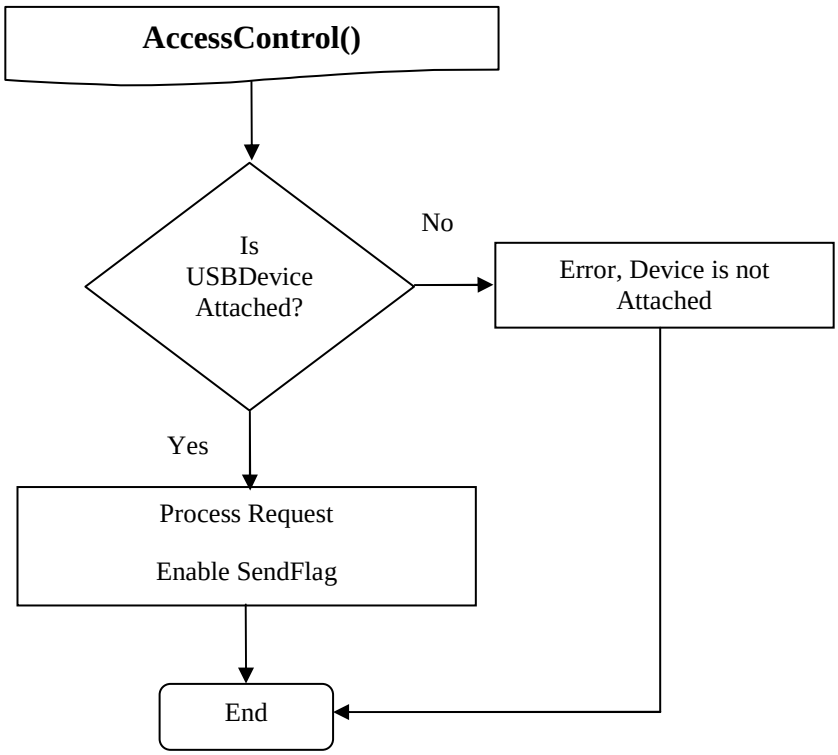
```
System::Void AddUser(System::Object^  sender, System::EventArgs^  e)
```



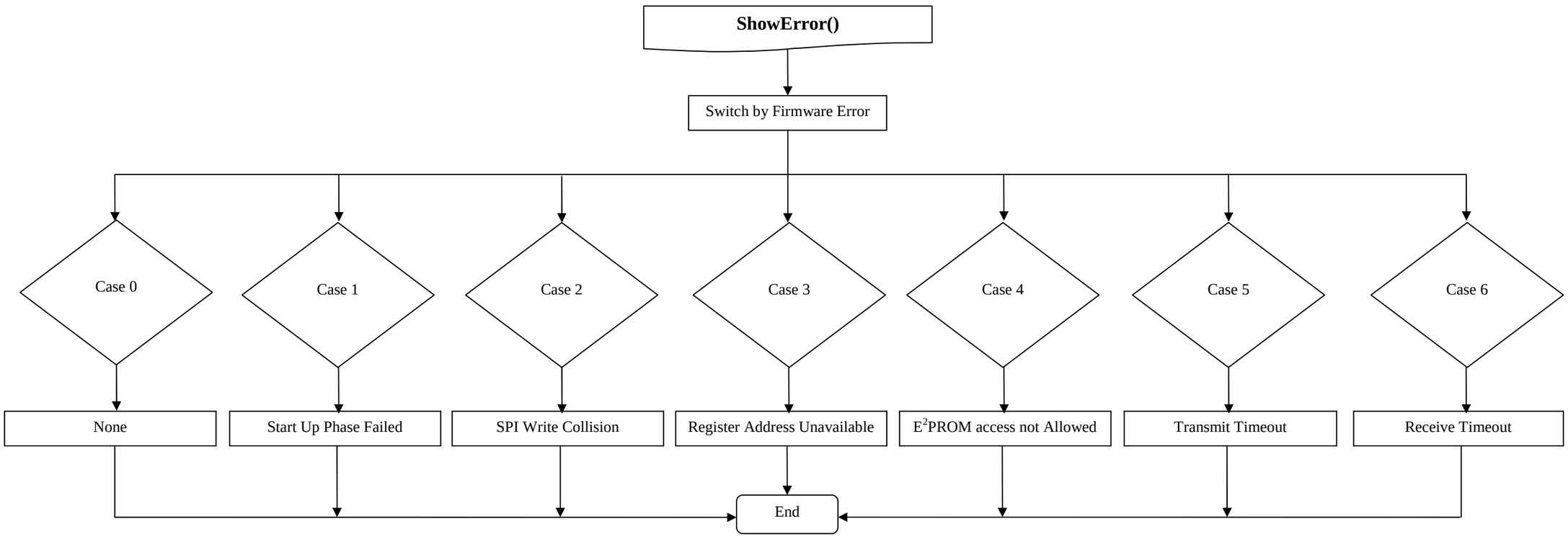
```
System::Void RemoveUser(System::Object^  sender, System::EventArgs^  e)
```



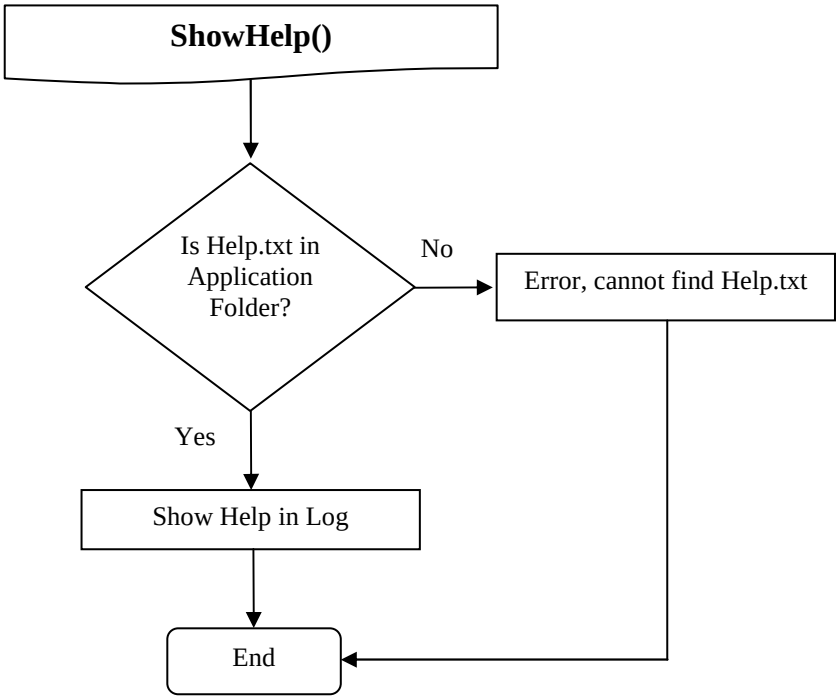
```
System::Void AccessControl(System::Object^ sender, System::EventArgs^  e)
```



```
void ShowError(void)
```



```
System::Void ShowHelp(System::Object^  sender, System::EventArgs^  e)
```



4. Presupuesto

Lector de Etiquetas Pasivas de RFID

TITULACIÓN: Ingeniería Técnica Industrial en Electrónica Industrial

AUTOR: Óscar Aragón Andreu
DIRECTOR: José Luís Ramírez Falo

FECHA: diciembre / 2009

4.1. Introducción

En el Presupuesto se pretende demostrar el coste real que implicaría el diseño y montaje del prototipo del Lector. Por este motivo se tendrán en cuenta los costes que debería afrontar un pequeño empresario, como sería la adquisición de las licencias de los programas de diseño.

En estas condiciones, también será interesante calcular cuánto costaría la elaboración de cada prototipo por separado, sin tener en cuenta el diseño. Esto será útil para realizar una estimación del coste que implicaría una pequeña producción en serie. Se deberá tener en cuenta que dependiendo del volumen de producción los costes podrían variar.

Así pues, se realizarán dos presupuestos:

- **Presupuesto A:** reflejará el coste referente al diseño y montaje del prototipo inicial del Lector.

- **Presupuesto B**: reflejará el coste que implicaría realizar cada prototipo por separado, sin tener en cuenta el proceso de diseño.

4.2. Precios Unitarios

CÓDIGO	UD	DESCRIPCIÓN	PRECIO	
IT001	h	Ingeniero Técnico Industrial	14,50	CATORCE EUROS con CINCUENTA CÉNTIMOS
TM001	h	Técnico Montador	9,00	NUEVE EUROS
DD001	u	ISO/IEC 15693-2	52,92	CINCUENTA Y DOS EUROS con NOVENTA Y DOS CÉNTIMOS
DD002	u	ISO/IEC 15693-3	93,94	NOVENTA Y TRES EUROS con NOVENTA Y CUATRO CÉNTIMOS
HD001	u	Cadence OrCAD v9.0	250,00	DOSCIENTOS CINCUENTA EUROS
HD002	u	MPLAB C Compiler for PIC18 MCUs	353,57	TRESCIENTOS CINCUENTA Y TRES EUROS CON CINCUENTA Y SIETE CÉNTIMOS
HD003	u	USB Compatible Debugger/Programmer MPLAB ICD2	43,20	CUARENTA Y TRES EUROS con VEINTE CÉNTIMOS
HD004	u	Microsoft Visual Studio 2008 Standard Edition	299,00	DOSCIENTOS NOVENTA Y NUEVE EUROS
HD005	u	Microsoft Office Small Business 2007	369,99	TRESCIENTOS SESENTA Y NUEVE EUROS con NOVENTA Y NUEVE CÉNTIMOS
HW001	u	Placa Microperforada 10 x 16 mm	2,75	DOS EUROS con SETENTA Y CINCO CÉNTIMOS
HW002	u	Transformador, 220 VAC/12 VDC 500 mA	10,20	DIEZ EUROS con VEINTE CÉNTIMOS
HW003	u	Conector Hembra para PCB, 2.5 mm 5 A 250 V	2,58	DOS EUROS con CINCUENTA Y OCHO CÉNTIMOS
HW004	u	Regulador Lineal de Tensión, L7805CV 5 V 1.5 A	0,85	OCHENTA Y CINCO CÉNTIMOS
HW005	u	Condensador Cerámico, 0,33 uF 50 V	0,15	QUINCE CÉNTIMOS
HW006	u	Condensador Cerámico, 0,10 uF 50 V	0,12	DOCE CÉNTIMOS
HW007	u	Microcontrolador, PIC18F2550-I/SP	4,81	CUATRO EUROS con OCHENTA Y UN CÉNTIMOS
HW008	u	Zócalo, DIP 28	0,75	SETENTA Y CINCO CÉNTIMOS
HW009	u	Cristal, 20 MHz HC49/4H	0,89	OCHENTA Y NUEVE CÉNTIMOS
HW010	u	Condensador Cerámico, 15 pF 50 V	0,15	QUINCE CÉNTIMOS
HW011	u	Interruptor táctil, 6x6x3,85mm	0,29	VEINTINUEVE CÉNTIMOS
HW012	u	Resistencia, 0,25 W 1 kΩ 0,5 %	0,10	DIEZ CÉNTIMOS

CÓDIGO	UD	DESCRIPCIÓN	PRECIO	DOCE CÉNTIMOS
HW014	u	Diodo Led, 3mm Rojo	0,31	TREINTA Y UN CÉNTIMOS
HW015	u	Diodo Led, 3mm Amarillo	0,23	VEINTITRES CÉNTIMOS
HW015	u	Interruptor de Palanca, 2 circuitos 250 VAC 3 A	0,87	OCHENTA Y SIETE CÉNTIMOS
HW017	u	Condensador Cerámico, 220 nF 50 V	0,15	QUINCE CÉNTIMOS
HW018	u	Conector Hembra USB para PCB, Tipo A	1,23	UN EURO con VEINTITRES CÉNTIMOS
HW019	u	Cable de interfaz USB 2.0 A-B 1m	1,73	UN EURO con SETENTA Y TRES
HW020	u	Header Vertical, 40 pines	0,35	TREINTA Y CINCO CÉNTIMOS
HW021	u	Jumper	0,15	QUINCE CÉNTIMOS
HW022	u	Transceiver de RFID, CL RC632	10,60	DIEZ EUROS con SESENTA CÉNTIMOS
HW023	u	Cristal, 13,56 MHz HC49/4H	0,93	NOVENTA Y TRES CÉNTIMOS
HW024	u	Inductor, 1 uH 1200 mA	1,17	UN EURO con DIECISIETE CÉNTIMOS
HW025	u	Condensador Cerámico, 68 pF 50 V	0,28	VEINTIOCHO CÉNTIMOS
HW026	u	Condensador Cerámico, 1 nF 50 V	0,28	VEINTIOCHO CÉNTIMOS
HW027	u	Potenciometro Multivuelta, 5 kΩ 4 mm	2,16	DOS EUROS con DIECISEIS CÉNTIMOS
HW028	u	Resistencia, 0,25 W 820 Ω 0,5 %	0,12	DOCE CÉNTIMOS
HW029	u	Condensador Cerámico, 100 nF 50 V	0,28	VEINTIOCHO CÉNTIMOS

4.3. Precios Descompuestos

4.3.1. Capítulo 1: Estudios Previos

CÓDIGO	UD	DESCRIPCIÓN	CANTIDAD	PRECIO	SUBTOTAL
IT001	h	Ingeniero Técnico Industrial	80,00	14,50	1160,00
DD001	u	ISO/IEC 15693-2	1,00	52,92	52,92
DD002	u	ISO/IEC 15693-3	1,00	93,94	93,94

Suma de la Partida	…….	1.306,86
2,00 % Costes Indirectos	…….	26,14
TOTAL PARTIDA	…….	**1.333,00 €**

4.3.2. Capítulo 2: Diseño y Montaje del Hardware

CÓDIGO	UD	DESCRIPCIÓN	CANTIDAD	PRECIO	SUBTOTAL
IT001	h	Ingeniero Técnico Industrial	80,00	14,50	1160,00
HD001	u	Cadence OrCAD v9.0	1,00	250,00	250,00
HW001	u	Placa Microperforada 10 x 16 mm	1,00	2,75	2,75
HW002	u	Transformador, 220 VAC/12 VDC 500 mA	1,00	10,20	10,20
HW003	u	Conector Hembra para PCB, 2.5 mm 5 A 250 V	1,00	2,58	2,58
HW004	u	Regulador Lineal de Tensión, L7805CV 5 V 1.5 A	1,00	0,85	0,85
HW005	u	Condensador Cerámico, 0,33 uF 50 V	1,00	0,15	0,15
HW006	u	Condensador Cerámico, 0,10 uF 50 V	1,00	0,12	0,12
HW007	u	Microcontrolador, PIC18F2550-I/SP	1,00	4,81	4,81
HW008	u	Zócalo, DIP 28	1,00	0,75	0,75
HW009	u	Cristal, 20 MHz HC49/4H	1,00	0,89	0,89
HW010	u	Condensador Cerámico, 15 pF 50 V	4,00	0,15	0,60
HW011	u	Interruptor táctil, 6x6x3,85mm	2,00	0,29	0,58
HW012	u	Resistencia, 0,25 W 1 kΩ 0,5 %	5,00	0,10	0,50
HW013	u	Diodo Led, 3mm Verde	1,00	0,12	0,12
HW014	u	Diodo Led, 3mm Rojo	1,00	0,31	0,31
HW015	u	Diodo Led, 3mm Amarillo	1,00	0,23	0,23
HW015	u	Interruptor de Palanca, 2 circuitos 250 VAC 3 A	1,00	0,87	0,87
HW017	u	Condensador Cerámico, 220 nF 50 V	1,00	0,15	0,15
HW018	u	Conector Hembra USB para PCB, Tipo A	1,00	1,23	1,23
HW019	u	Cable de interfaz USB 2.0 A-B 1m	1,00	1,73	1,73
HW020	u	Header Vertical, 40 pines	1,00	0,35	0,35
HW021	u	Jumper	1,00	0,15	0,15
HW022	u	Transceiver de RFID, CL RC632	1,00	10,60	10,60
HW023	u	Cristal, 13,56 MHz HC49/4H	1,00	0,93	0,93
HW024	u	Inductor, 1 uH 1200 mA	1,00	1,17	1,17
HW025	u	Condensador Cerámico, 68 pF 50 V	2,00	0,28	0,56
HW026	u	Condensador Cerámico, 1 nF 50 V	4,00	0,28	1,12
HW027	u	Potenciometro Multivuelta, 5 kΩ 4 mm	1,00	2,16	2,16
HW028	u	Resistencia, 0,25 W 820 Ω 0,5 %	1,00	0,12	0,12
HW029	u	Condensador Cerámico, 100 nF 50 V	1,00	0,28	0,28

Suma de la Partida	…….	1.456,86
2,00 % Costes Indirectos	…….	29,14
TOTAL PARTIDA	…….	**1.486,00 €**

4.3.3. Capítulo 3: Diseño del Firmware

CÓDIGO	UD	DESCRIPCIÓN	CANTIDAD	PRECIO	SUBTOTAL
IT001	h	Ingeniero Técnico Industrial	240,00	14,50	3480,00
HD002	u	MPLAB C Compiler for PIC18 MCUs	1,00	353,57	353,57
HD003	u	USB Compatible Debugger/Programmer MPLAB ICD2	1,00	43,20	43,20

Suma de la Partida	…….	3.876,77
2,00 % Costes Indirectos	…….	77,54
TOTAL PARTIDA	…….	**3.954,31 €**

4.3.4. Capítulo 4: Diseño del Software

CÓDIGO	UD	DESCRIPCIÓN	CANTIDAD	PRECIO	SUBTOTAL
IT001	h	Ingeniero Técnico Industrial	160,00	14,50	2320,00
HD004	u	Microsoft Visual Studio 2008 Standard Edition	1,00	299,00	299,00

Suma de la Partida	…….	2.619,00
2,00 % Costes Indirectos	…….	52,38
TOTAL PARTIDA	…….	**2.671,38 €**

4.3.5. Capítulo 5: Documentación

CÓDIGO	UD	DESCRIPCIÓN	CANTIDAD	PRECIO	SUBTOTAL
IT001	h	Ingeniero Técnico Industrial	80,00	14,50	1160,00
HD005	u	Microsoft Office Small Business 2007	1,00	369,99	369,99

Suma de la Partida	…….	1.529,99
2,00 % Costes Indirectos	…….	30,60
TOTAL PARTIDA	…….	**1.560,59 €**

4.3.6. Capítulo 6: Montaje de cada Prototipo

CÓDIGO	UD	DESCRIPCIÓN	CANTIDAD	PRECIO	SUBTOTAL
TM001	h	Técnico Montador	8,00	9,00	72,00
HW001	u	Placa Microperforada 10 x 16 mm	1,00	2,75	2,75
HW002	u	Transformador, 220 VAC/12 VDC 500 mA	1,00	10,20	10,20
HW003	u	Conector Hembra para PCB, 2.5 mm 5 A 250 V	1,00	2,58	2,58
HW004	u	Regulador Lineal de Tensión, L7805CV 5 V 1.5 A	1,00	0,85	0,85
HW005	u	Condensador Cerámico, 0,33 uF 50 V	1,00	0,15	0,15
HW006	u	Condensador Cerámico, 0,10 uF 50 V	1,00	0,12	0,12
HW007	u	Microcontrolador, PIC18F2550-I/SP	1,00	4,81	4,81
HW008	u	Zócalo, DIP 28	1,00	0,75	0,75
HW009	u	Cristal, 20 MHz HC49/4H	1,00	0,89	0,89
HW010	u	Condensador Cerámico, 15 pF 50 V	4,00	0,15	0,60
HW011	u	Interruptor táctil, 6x6x3,85mm	2,00	0,29	0,58
HW012	u	Resistencia, 0,25 W 1 kΩ 0,5 %	5,00	0,10	0,50
HW013	u	Diodo Led, 3mm Verde	1,00	0,12	0,12
HW014	u	Diodo Led, 3mm Rojo	1,00	0,31	0,31
HW015	u	Diodo Led, 3mm Amarillo	1,00	0,23	0,23
HW015	u	Interruptor de Palanca, 2 circuitos 250 VAC 3 A	1,00	0,87	0,87
HW017	u	Condensador Cerámico, 220 nF 50 V	1,00	0,15	0,15
HW018	u	Conector Hembra USB para PCB, Tipo A	1,00	1,23	1,23
HW019	u	Cable de interfaz USB 2.0 A-B 1m	1,00	1,73	1,73
HW020	u	Header Vertical, 40 pines	1,00	0,35	0,35
HW021	u	Jumper	1,00	0,15	0,15
HW022	u	Transceiver de RFID, CL RC632	1,00	10,60	10,60
HW023	u	Cristal, 13,56 MHz HC49/4H	1,00	0,93	0,93
HW024	u	Inductor, 1 uH 1200 mA	1,00	1,17	1,17
HW025	u	Condensador Cerámico, 68 pF 50 V	2,00	0,28	0,56
HW026	u	Condensador Cerámico, 1 nF 50 V	4,00	0,28	1,12
HW027	u	Potenciometro Multivuelta, 5 kΩ 4 mm	1,00	2,16	2,16
HW028	u	Resistencia, 0,25 W 820 Ω 0,5 %	1,00	0,12	0,12
HW029	u	Condensador Cerámico, 100 nF 50 V	1,00	0,28	0,28

Suma de la Partida	…….	118,86
2,00 % Costes Indirectos	…….	2,38
TOTAL PARTIDA	…….	**121,24 €**

4.4. Resumen del Presupuesto

4.4.1. Presupuesto A: Diseño y Montaje del Prototipo Inicial

CAPÍTULO	RESUMEN		IMPORTE
Capítulo 1	Estudios Previos		1333,00
Capítulo 2	Diseño y Montaje del Hardware		1486,00
Capítulo 3	Diseño del Firmware		3954,31
Capítulo 4	Diseño del Software		2671,38
Capítulo 5	Documentación		1560,59
	TOTAL EJECUCIÓN MATERIAL	……	11005,27
	13,00 % Gastos Generales	……	1430,69
	6,00 % Beneficio Industrial	……	660,32
	TOTAL EJECUCIÓN POR CONTRATO	……	13096,27
	16,00 % IVA	……	2095,40
	TOTAL PRESUPUESTO LICITACIÓN	……	**15.191,67 €**

4.4.2. Presupuesto B: Montaje de cada Prototipo

CAPÍTULO	RESUMEN		IMPORTE
Capítulo 6	Montaje de cada Prototipo		121,24
	TOTAL EJECUCIÓN MATERIAL	……	121,24
	13,00 % Gastos Generales	……	15,76
	6,00 % Beneficio Industrial	……	7,27
	TOTAL EJECUCIÓN POR CONTRATO	……	144,27
	16,00 % IVA	……	23,08
	TOTAL PRESUPUESTO LICITACIÓN	……	**167,36 €**

5.Bibliografía y Referencias

Lector de Etiquetas Pasivas de RFID

TITULACIÓN: Ingeniería Técnica Industrial en Electrónica Industrial

AUTOR: Óscar Aragón Andreu
DIRECTOR: José Luís Ramírez Falo

FECHA: diciembre / 2009

5.1. Bibliografía

- RFiD Magazine. *Tecnología RFID: Introducción.* [En línea]. Documento informativo, 12-12-2005 [Fecha de consulta: 15-09-2008] [Acceso gratuito] < http://www.rfid-magazine.com/_images/262/RFID_introduccion.pdf >

- EAN Panamá. *Introducción al RFID y aplicaciones en la Cadena de Distribución.* [En línea]. Boletín Informativo, Año 8 – Nº 03 – Marzo de 2007 [Fecha de consulta: 15-09-2008] [Acceso gratuito] < http://www.gs1pa.org/boletin/2007/marzo/rfidintro.pdf >

- Aprende Dynamics AX. *Introducción al RFID.* [En línea]. Artículo digital [Fecha de consulta: 15-09-2008] [Acceso gratuito] < http://www.aprendedynamics.com/rfid2.html >

- Aprende Dynamics AX. *Qué es y cómo funciona RFID.* [En línea]. Artículo digital [Fecha de consulta: 15-09-2008] [Acceso gratuito] < http://www.aprendedynamics.com/rfid3.html >

- Servicios Informáticos KIFER, S.L. *Introducción a los Sistemas RFID.* [En línea]. Artículo digital [Fecha de consulta: 15-09-2008] [Acceso gratuito] < http://www.kifer.es/Recursos/Pdf/RFID.pdf >

- Wikipedia. *RFID.* [En línea]. Enciclopedia digital [Fecha de consulta: 15-09-2008] [Acceso gratuito] < http://es.wikipedia.org/wiki/RFID >

- ISO/IEC 15693-2:2000 (E). *Identification cards - Contactless integrated circuit cards - Vicinity cards - Part 2: Air interface and initialization.*

- ISO/IEC 15693-3:2001 (E). *Identification cards - Contactless integrated circuit cards - Vicinity cards - Part 3: Anticollision and transmission protocol.*

- NXP Semiconductors. *CLRC63201T - Multiple protocol contactless reader IC (MIFARE/I-CODE1).* [En línea]. Rev. 3.5 – 10 November 20009, 073935 [Fecha de consulta: 08-12-2009] [Acceso gratuito] < http://www.nxp.com/documents/data_sheet/CLRC632.pdf >

- Microchip Technology Inc. *PIC18F2455/2550/4455/4550 Data sheet*. [En línea]. Nº de documento: DS39632D [Fecha de consulta: 02-11-2008] [Acceso gratuito] < http://ww1.microchip.com/downloads/en/DeviceDoc/39632d.pdf >

- STMicroelectronics. *L78xx / L78xxC Positive voltage regulators*. [En línea]. Rev19, March 2008 [Fecha de consulta: 02-11-2008] [Acceso gratuito] < http://www.st.com/stonline/products/literature/ds/2143.pdf >

- Texas Instruments. *RI-I03-114A-S1 Data Sheet.* [En línea]. SCBS823 – Dec 2005 [Fecha de consulta: 02-11-2008] [Acceso gratuito] < http://focus.ti.com/lit/ds/symlink/ri-i03-114a-s1.pdf >

- Microchip Technology Inc. *MPLAB IDE User's Guide*. [En línea]. Nº de documento: DS51519C [Fecha de consulta: 20-11-2008] [Acceso gratuito] < http://ww1.microchip.com/downloads/en/DeviceDoc/MPLAB_User_Guide_51519c.pdf >

- Microchip Technology Inc. *MPLAB C18 C Compiler Getting Started*. [En línea]. Nº de documento: DS51295F [Fecha de consulta: 20-11-2008] [Acceso gratuito] < http://ww1.microchip.com/downloads/en/DeviceDoc/MPLAB_C18_Getting_Started_51295f.pdf >

- Microchip Technology Inc. *MPLAB C18 C Compiler User's Guide*. [En línea]. Nº de documento: DS51288J [Fecha de consulta: 20-11-2008] [Acceso gratuito] < http://ww1.microchip.com/downloads/en/DeviceDoc/C18_User_Guide_51288j.pdf >

- Microchip Technology Inc. *MPLAB C18 C Compiler Libraries*. [En línea]. Nº de documento: DS51297F [Fecha de consulta: 20-11-2008] [Acceso gratuito] < http://ww1.microchip.com/downloads/en/DeviceDoc/MPLAB_C18_Libraries_51297f.pdf >

- Microchip Technology Inc. *PIC18 Configurations Setting Anddendum*. [En línea]. Nº de documento: DS51537E [Fecha de consulta: 20-11-2008] [Acceso gratuito] < http://ww1.microchip.com/downloads/en/DeviceDoc/C18_Config_Settings_51537e.pdf >

- USB.org. *USB 2.0 Specification.* [En línea]. Revision 2.0, April 27,200. [Fecha de consulta: 29-01-2009] [Acceso gratuito] < http://www.usb.org/developers/docs/ >

- Microchip Technology Inc. *USB Framework for PIC18, PIC24 & PIC32, Microchip Application Libraries*. [En línea]. [Fecha de consulta: 20-08-2009] [Acceso gratuito] < http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2680&dDocName=en537044 >

5.2. Referencias

En este apartado se pretende hacer referencia específica a las diferentes afirmaciones que se encuentran en Proyecto. Para ello se citará el tipo de documento, el nombre del documento y capítulo exacto donde encontrar la información referenciada. Los documentos se localizarán en la bibliografía indicada anteriormente o en directorio *\\RFidReader\Documents* incluido en el CD del Proyecto.

[1] Normativa: *ISO/IEC 15693-3:2001(E)*. 5 VICC memory organization

[2] Hoja de Características: *PIC18F2455/2550/4455/4550 Data Sheet Rev.D.* Features page

[3] Hoja de Características: *PIC18F2455/2550/4455/4550 Data Sheet Rev.D.* 17.3 USB RAM

[4] Hoja de Características: *CL RC632 Data sheet Rev.3.0*. 1.3 Features

[5] Hoja de Características: *PIC18F2455/2550/4455/4550 Data Sheet Rev.D*. Features page

[6] Hoja de Características: *CL RC632 Data sheet Rev.3.0*. 6.1 Diagram of the E^2PROM Memory Organization

[7] Hoja de Características: *CL RC632 Data sheet Rev.3.0*. 20.1 Circuit Diagram

[8] Hoja de Características: PIC18F2455/2550/4455/4550 Data Sheet Rev.D. Table 2-3: Oscillator Configuration Options for Usb Operation

[9] Hoja de Características: *CL RC632 Data sheet Rev.3.0*. 11 Start Up Phase

[10] Hoja de Características: *PIC18F2455/2550/4455/4550 Data Sheet Rev.D*. 11.0 Timer0 Module

[11] Hoja de Características: *PIC18F2455/2550/4455/4550 Data Sheet Rev.D*. 9.7 INTn Pin Interrupts

[12] Hoja de Características: *PIC18F2455/2550/4455/4550 Data Sheet Rev.D*. Table 19-1: SPI Bus Modes

[13] Hoja de Características: *CL RC632 Data sheet Rev.3.0*. 22.5.2.4 Timing for SPI compatible interface

[14] Especificación: USB Specification v2.0, 6.2 Keyed Connector Protocol

[15] Hoja de Características: *PIC18F2455/2550/4455/4550 Data Sheet Rev.D*. 17.6 USB Power Modes

[16] Hoja de Características: *PIC18F2455/2550/4455/4550 Data Sheet Rev.D*. 17.1 Overview of the USB Peripheral

[17] Hoja de Características: *PIC18F2455/2550/4455/4550 Data Sheet Rev.D*. 17.3 USB RAM

[18] Hoja de Características: *CL RC632 Data sheet Rev.3.0*. 14 Receiver circuitry & 15 Serial Signal Switch

[19] Normativa: *ISO/IEC 15693-2:2000(E)*. 7.1 Modulation

[20] Hoja de Características: *PIC18F2455/2550/4455/4550 Data Sheet Rev.D*. Figure 11-1: Timer0 Block Diagram (8-bit Mode)

[21] Documentación: *MPLAB C18 C Compiler Libraries Rev. D.* 2.9 Timer Functions

[22] Documentación: *Lector de Etiquetas Pasivas de RFID.* Tabla 6. Pines I/O

[23] Hoja de Características: *PIC18F2455/2550/4455/4550 Data Sheet Rev.D*. FIGURE 19-2: SPI Master/Slave connection

[24] Documentación: *MPLAB C18 C Compiler Libraries Rev. D.* 2.8 SPI Functions

[25] Página Web: http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2651&param=en534494

[26] Colección de archivos: \\RFidReader\Documentation\Microchip Solutions\USB Device - MCHPUSB - Generic Driver Demo\Generic Driver Demo – Firmware

[27] Documentación digital: \\RFidReader\Documents\Microchip Solutions\Microchip\Help\USB Device Library Help

[28] Especificación: *USB Specification v2.0*. 9.1 USB Device States

[29] Especificación: *USB Specification v2.0*. 4.2.1 Electrical & 5.3.1 Device Endpoints

[30] Especificación: *USB Specification v2.0*. 5.4 Transfer Types

[31] Hoja de Características: *CL RC632 Data sheet Rev.3.0*. 11 Start Up Phase

[32] Hoja de Características: *CL RC632 Data sheet Rev.3.0*. 4.4 SPI Compatible Interface

[33] Hoja de Características: *CL RC632 Data sheet Rev.3.0*. 1.3 Features

[34] Hoja de Características: *CL RC632 Data sheet Rev.3.0*. 5.2.1.3 FIFOData Register

[35] Hoja de Características: *CL RC632 Data sheet Rev.3.0*. 6.1 Diagram of the E2PROM Memory Organization

[36]Hoja de Características: *CL RC632 Data sheet Rev.3.0*. 18 CL RC632 Command Set

[37] Hoja de Características: *CL RC632 Data sheet Rev.3.0*. 9 Timer Unit

[38] Normativa: *ISO/IEC 15693-3:2001(E)*. 7.3 Request Format & 7.4 Response Format

[39] Hoja de Características: *RI-I03-114A-S1 Data Sheet Rev.2005*

[40] Hoja de Características: *PIC18F2455/2550/4455/4550 Data Sheet Rev.D*. Figure 2-1: Clock Diagram

[41] Código: \\RFidReader\RFidReader.h

[42] Colección de archivos: \\RFidReader\Documents\Microchip Solutions\USB Device - MCHPUSB - Generic Driver Demo\PC Software\Visual C++ 2005 Express

[43] Hoja de Características: *PIC18F2455/2550/4455/4550 Data Sheet Rev.D*. 28.0 Electrical Characteristics

[44] Hoja de Características: *CL RC632 Data sheet Rev.3.0*. 22 Electrical Characteristics

[45] Hoja de Características: *L7805CV – Positive voltaje regulator*. Application Circuits

[46] Hoja de Características: CL RC632 Data sheet Rev.3.0. 22.5.3 Clock Frequency

[47] Hoja de Características: PIC18F2455/2550/4455/4550 Data Sheet Rev.D. Table 2-3: Oscillator Configuration Options for Usb Operation

[48] Hoja de Características: CL RC632 Data sheet Rev.3.0. 11 Start Up Phase

[49] Hoja de Características: PIC18F2455/2550/4455/4550 Data Sheet Rev.D. Figure 11-1: Timer0 block Diagram (8-bit Mode)

[50] Normativa: ISO/IEC 15693-3:2001(E). 7.3 Request Format

[51] Normativa: ISO/IEC 15693-2:2000(E). 7.2.1 Data Coding Mode: 1 out 256

[52] Normativa: ISO/IEC 15693-2:2000(E). 8 Communications signal interface VICC yo VCD

6. Anexos

Lector de Etiquetas Pasivas de RFID

TITULACIÓN: Ingeniería Técnica Industrial en Electrónica Industrial

AUTOR: Óscar Aragón Andreu
DIRECTOR: José Luís Ramírez Falo

FECHA: diciembre / 2009

6.1. Código Fuente del Firmware

6.1.1. GenericTypeDefs.h

```
/********************************************************************
*
*                       Generic Type Definitions
*
********************************************************************
* FileName:        GenericTypeDefs.h
* Dependencies:    None
* Processor:       PIC18, PIC24, dsPIC, PIC32
* Compiler:        Microchip C18, C30, C32
* Company:         Microchip Technology, Inc.
*
* Software License Agreement
*
* The software supplied herewith by Microchip Technology Incorporated
* (the "Company") is intended and supplied to you, the Company's
* customer, for use solely and exclusively with products manufactured
* by the Company.
*
* The software is owned by the Company and/or its supplier, and is
* protected under applicable copyright laws. All rights are reserved.
* Any use in violation of the foregoing restrictions may subject the
* user to criminal sanctions under applicable laws, as well as to
* civil liability for the breach of the terms and conditions of this
* license.
*
* THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES,
* WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
* TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
* PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
* IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
* CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
*
********************************************************************
* File Description:
*
* Change History:
*  Rev   Date         Description
*  1.1   09/11/06     Add base signed types
*  1.2   02/28/07     Add QWORD, LONGLONG, QWORD_VAL
*  1.3   02/06/08     Add def's for PIC32
*  1.4   08/08/08     Remove LSB/MSB Macros, adopted by Peripheral lib
*  1.5   08/14/08     Simplify file header
********************************************************************/

#ifndef __GENERIC_TYPE_DEFS_H_
#define __GENERIC_TYPE_DEFS_H_

typedef enum _BOOL { FALSE = 0, TRUE } BOOL;  // Undefined size

#ifndef NULL
#define NULL    0//((void *)0)
#endif

#define PUBLIC                                  // Function attributes
#define PROTECTED
#define PRIVATE   static

typedef unsigned char     BYTE;                 // 8-bit unsigned
typedef unsigned short int WORD;                // 16-bit unsigned
typedef unsigned long            DWORD;         // 32-bit unsigned
typedef unsigned long long QWORD;               // 64-bit unsigned
typedef signed char       CHAR;                 // 8-bit signed
```

```
typedef signed short int   SHORT;              // 16-bit signed
typedef signed long        LONG;               // 32-bit signed
typedef signed long long   LONGLONG;           // 64-bit signed

/* Alternate definitions */
typedef void                 VOID;

typedef char                 CHAR8;
typedef unsigned char        UCHAR8;

/* Processor & Compiler independent, size specific definitions */
// To Do:  We need to verify the sizes on each compiler.  These
//         may be compiler specific, we should either move them
//         to "compiler.h" or #ifdef them for compiler type.
typedef signed int           INT;
typedef signed char          INT8;
typedef signed short int     INT16;
typedef signed long int      INT32;
typedef signed long long     INT64;

typedef unsigned int         UINT;
typedef unsigned char        UINT8;
typedef unsigned short int   UINT16;
typedef unsigned long int    UINT32;  // other name for 32-bit integer
typedef unsigned long long   UINT64;

typedef union _BYTE_VAL
{
    BYTE Val;
    struct
    {
        unsigned char b0:1;
        unsigned char b1:1;
        unsigned char b2:1;
        unsigned char b3:1;
        unsigned char b4:1;
        unsigned char b5:1;
        unsigned char b6:1;
        unsigned char b7:1;
    } bits;
} BYTE_VAL, BYTE_BITS;

typedef union _WORD_VAL
{
    WORD Val;
    BYTE v[2];
    struct
    {
        BYTE LB;
        BYTE HB;
    } byte;
    struct
    {
        unsigned char b0:1;
        unsigned char b1:1;
        unsigned char b2:1;
        unsigned char b3:1;
        unsigned char b4:1;
        unsigned char b5:1;
        unsigned char b6:1;
        unsigned char b7:1;
        unsigned char b8:1;
        unsigned char b9:1;
        unsigned char b10:1;
        unsigned char b11:1;
        unsigned char b12:1;
        unsigned char b13:1;
```

```
        unsigned char b14:1;
        unsigned char b15:1;
    } bits;
} WORD_VAL, WORD_BITS;

typedef union _DWORD_VAL
{
    DWORD Val;
    WORD w[2];
    BYTE v[4];
    struct
    {
        WORD LW;
        WORD HW;
    } word;
    struct
    {
        BYTE LB;
        BYTE HB;
        BYTE UB;
        BYTE MB;
    } byte;
    struct
    {
        WORD_VAL low;
        WORD_VAL high;
    }wordUnion;
    struct
    {
        unsigned char b0:1;
        unsigned char b1:1;
        unsigned char b2:1;
        unsigned char b3:1;
        unsigned char b4:1;
        unsigned char b5:1;
        unsigned char b6:1;
        unsigned char b7:1;
        unsigned char b8:1;
        unsigned char b9:1;
        unsigned char b10:1;
        unsigned char b11:1;
        unsigned char b12:1;
        unsigned char b13:1;
        unsigned char b14:1;
        unsigned char b15:1;
        unsigned char b16:1;
        unsigned char b17:1;
        unsigned char b18:1;
        unsigned char b19:1;
        unsigned char b20:1;
        unsigned char b21:1;
        unsigned char b22:1;
        unsigned char b23:1;
        unsigned char b24:1;
        unsigned char b25:1;
        unsigned char b26:1;
        unsigned char b27:1;
        unsigned char b28:1;
        unsigned char b29:1;
        unsigned char b30:1;
        unsigned char b31:1;
    } bits;
} DWORD_VAL;

typedef union _QWORD_VAL
{
    QWORD Val;
```

```
   DWORD d[2];
   WORD w[4];
BYTE v[8];
struct
{
    DWORD LD;
    DWORD HD;
} dword;
struct
{
    WORD LW;
    WORD HW;
    WORD UW;
    WORD MW;
} word;
struct
{
    unsigned char b0:1;
    unsigned char b1:1;
    unsigned char b2:1;
    unsigned char b3:1;
    unsigned char b4:1;
    unsigned char b5:1;
    unsigned char b6:1;
    unsigned char b7:1;
    unsigned char b8:1;
    unsigned char b9:1;
    unsigned char b10:1;
    unsigned char b11:1;
    unsigned char b12:1;
    unsigned char b13:1;
    unsigned char b14:1;
    unsigned char b15:1;
    unsigned char b16:1;
    unsigned char b17:1;
    unsigned char b18:1;
    unsigned char b19:1;
    unsigned char b20:1;
    unsigned char b21:1;
    unsigned char b22:1;
    unsigned char b23:1;
    unsigned char b24:1;
    unsigned char b25:1;
    unsigned char b26:1;
    unsigned char b27:1;
    unsigned char b28:1;
    unsigned char b29:1;
    unsigned char b30:1;
    unsigned char b31:1;
    unsigned char b32:1;
    unsigned char b33:1;
    unsigned char b34:1;
    unsigned char b35:1;
    unsigned char b36:1;
    unsigned char b37:1;
    unsigned char b38:1;
    unsigned char b39:1;
    unsigned char b40:1;
    unsigned char b41:1;
    unsigned char b42:1;
    unsigned char b43:1;
    unsigned char b44:1;
    unsigned char b45:1;
    unsigned char b46:1;
    unsigned char b47:1;
    unsigned char b48:1;
    unsigned char b49:1;
```

```
        unsigned char b50:1;
        unsigned char b51:1;
        unsigned char b52:1;
        unsigned char b53:1;
        unsigned char b54:1;
        unsigned char b55:1;
        unsigned char b56:1;
        unsigned char b57:1;
        unsigned char b58:1;
        unsigned char b59:1;
        unsigned char b60:1;
        unsigned char b61:1;
        unsigned char b62:1;
        unsigned char b63:1;
    } bits;
} QWORD_VAL;

#endif //__GENERIC_TYPE_DEFS_H_
```

6.1.2. Compiler.h

```
/********************************************************************
*
*               Compiler and hardware specific definitions
*
********************************************************************
* FileName:        Compiler.h
* Dependencies:    None
* Processor:       PIC18, PIC24F, PIC24H, dsPIC30F, dsPIC33F, PIC32
* Compiler:        Microchip C32 v1.00 or higher
*                             Microchip C30 v3.01 or higher
*                             Microchip C18 v3.13 or higher
*                             HI-TECH PICC-18 STD 9.50PL3 or higher
* Company:         Microchip Technology, Inc.
*
* Software License Agreement
*
* Copyright (C) 2002-2008 Microchip Technology Inc.  All rights
* reserved.
*
* Microchip licenses to you the right to use, modify, copy, and
* distribute:
* (i)  the Software when embedded on a Microchip microcontroller or
*      digital signal controller product ("Device") which is
*      integrated into Licensee's product; or
* (ii) ONLY the Software driver source files ENC28J60.c and
*      ENC28J60.h ported to a non-Microchip device used in
*      conjunction with a Microchip ethernet controller for the
*      sole purpose of interfacing with the ethernet controller.
*
* You should refer to the license agreement accompanying this
* Software for additional information regarding your rights and
* obligations.
*
* THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT
* WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT
* LIMITATION, ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL
* MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR
* CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF
* PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS
* BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE
* THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER
* SIMILAR COSTS, WHETHER ASSERTED ON THE BASIS OF CONTRACT, TORT
* (INCLUDING NEGLIGENCE), BREACH OF WARRANTY, OR OTHERWISE.
*
*
* Author               Date      Comment
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* Howard Schlunder     10/03/2006Original, copied from old Compiler.h
* Howard Schlunder     11/07/2007Reorganized and simplified
********************************************************************/
#ifndef __COMPILER_H
#define __COMPILER_H

// Include proper device header file
#if defined(__18CXX) || defined(HI_TECH_C)
       // All PIC18 processors
       #if defined(HI_TECH_C)    // HI TECH PICC-18 compiler
             #define __18CXX
             #include <htc.h>
       #else                     // Microchip C18 compiler
           #include <p18cxxx.h>
       #endif
#elif defined(__PIC24F__)        // Microchip C30 compiler
       // PIC24F processor
```

```
      #include <p24Fxxxx.h>
#elif defined(__PIC24H__)       // Microchip C30 compiler
      // PIC24H processor
      #include <p24Hxxxx.h>
#elif defined(__dsPIC33F__)     // Microchip C30 compiler
      // dsPIC33F processor
      #include <p33Fxxxx.h>
#elif defined(__dsPIC30F__)     // Microchip C30 compiler
      // dsPIC30F processor
      #include <p30fxxxx.h>
#elif defined(__PIC32MX__)      // Microchip C32 compiler
      #if !defined(__C32__)
            #define __C32__
      #endif
      #include <p32xxxx.h>
      #include <plib.h>
#else
      #error Unknown processor or compiler.  See Compiler.h
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>


// Base RAM and ROM pointer types for given architecture
#if defined(__PIC32MX__)
      #define PTR_BASE          DWORD
      #define ROM_PTR_BASE      DWORD
#elif defined(__C30__)
      #define PTR_BASE          WORD
      #define ROM_PTR_BASE      WORD
#elif defined(__18CXX)
      #define PTR_BASE          WORD
      #define ROM_PTR_BASE      unsigned short long
#endif


// Definitions that apply to all compilers, except C18
#if !defined(__18CXX) || defined(HI_TECH_C)
      #define memcmppgm2ram(a,b,c)     memcmp(a,b,c)
      #define strcmppgm2ram(a,b)       strcmp(a,b)
      #define memcpypgm2ram(a,b,c)     memcpy(a,b,c)
      #define strcpypgm2ram(a,b)       strcpy(a,b)
      #define strncpypgm2ram(a,b,c)    strncpy(a,b,c)
      #define strstrrampgm(a,b)        strstr(a,b)
      #define strlenpgm(a)             strlen(a)
      #define strchrpgm(a,b)           strchr(a,b)
      #define strcatpgm2ram(a,b)       strcat(a,b)
#endif


// Definitions that apply to all 8-bit products
// (PIC18)
#if defined(__18CXX)
      #define      __attribute__(a)

    #define FAR                         far

      // Microchip C18 specific defines
      #if !defined(HI_TECH_C)
          #define ROM                           rom
            #define strcpypgm2ram(a, b)      strcpypgm2ram(a,(far rom char*)b)
      #endif

      // HI TECH PICC-18 STD specific defines
      #if defined(HI_TECH_C)
```

```
            #define ROM                 const
               #define rom
            #define Nop()               asm("NOP");
               #define ClrWdt()         asm("CLRWDT");
            #define Reset()             asm("RESET");
        #endif

// Definitions that apply to all 16-bit and 32-bit products
// (PIC24F, PIC24H, dsPIC30F, dsPIC33F, and PIC32)
#else
        #define      ROM                const

        // 16-bit specific defines (PIC24F, PIC24H, dsPIC30F, dsPIC33F)
        #if defined(__C30__)
              #define Reset()           asm("reset")
         #define FAR                    __attribute__((far))
        #endif

        // 32-bit specific defines (PIC32)
        #if defined(__PIC32MX__)
              #define persistent
              #define far
         #define FAR
              #define Reset()           SoftReset()
              #define ClrWdt()          (WDTCONSET = _WDTCON_WDTCLR_MASK)
              #define Nop()             asm("nop")
        #endif
#endif

#endif
```

6.1.3. p18cxxx.h

```
/*********************************************************************
*
*                PIC18 microproccessor specific libraries
*
*********************************************************************
* FileName:        p18cxxx.h
* Dependencies:    See INCLUDES
* Processor:       PIC18
* Compiler:        Microchip C18
* Company:         Microchip Technology, Inc.
*********************************************************************/

#ifndef _P18CXXX_H
#define _P18CXXX_H

#if   defined(__18C658)
  #include <p18c658.h>
#elif defined(__18C858)
  #include <p18c858.h>
#elif defined(__18C601)
  #include <p18c601.h>
#elif defined(__18C801)
  #include <p18c801.h>
#elif defined(__18C242)
  #include <p18c242.h>
#elif defined(__18C252)
  #include <p18c252.h>
#elif defined(__18C442)
  #include <p18c442.h>
#elif defined(__18C452)
  #include <p18c452.h>
#elif defined(__18F242)
  #include <p18f242.h>
#elif defined(__18F248)
  #include <p18f248.h>
#elif defined(__18F252)
  #include <p18f252.h>
#elif defined(__18F258)
  #include <p18f258.h>
#elif defined(__18F442)
  #include <p18f442.h>
#elif defined(__18F448)
  #include <p18f448.h>
#elif defined(__18F452)
  #include <p18f452.h>
#elif defined(__18F458)
  #include <p18f458.h>
#elif defined(__18F1220)
  #include <p18f1220.h>
#elif defined(__18F1320)
  #include <p18f1320.h>
#elif defined(__18F2220)
  #include <p18f2220.h>
#elif defined(__18F2320)
  #include <p18f2320.h>
#elif defined(__18F2515)
  #include <p18f2515.h>
#elif defined(__18F2525)
  #include <p18f2525.h>
#elif defined(__18F2610)
  #include <p18f2610.h>
#elif defined(__18F2620)
  #include <p18f2620.h>
#elif defined(__18F4220)
  #include <p18f4220.h>
```

```
#elif defined(__18F4320)
  #include <p18f4320.h>
#elif defined(__18F4515)
  #include <p18f4515.h>
#elif defined(__18F4525)
  #include <p18f4525.h>
#elif defined(__18F4610)
  #include <p18f4610.h>
#elif defined(__18F4620)
  #include <p18f4620.h>
#elif defined(__18F6620)
  #include <p18f6620.h>
#elif defined(__18F6720)
  #include <p18f6720.h>
#elif defined(__18F8620)
  #include <p18f8620.h>
#elif defined(__18F8720)
  #include <p18f8720.h>
#elif defined(__18F2439)
  #include <p18f2439.h>
#elif defined(__18F4439)
  #include <p18f4439.h>
#elif defined(__18F2539)
  #include <p18f2539.h>
#elif defined(__18F4539)
  #include <p18f4539.h>
#elif defined(__18F8520)
  #include <p18f8520.h>
#elif defined(__18F6520)
  #include <p18f6520.h>
#elif defined(__18F8680)
  #include <p18f8680.h>
#elif defined(__18F6680)
  #include <p18f6680.h>
#elif defined(__18F8585)
  #include <p18f8585.h>
#elif defined(__18F6585)
  #include <p18f6585.h>
#elif defined(__18F8621)
  #include <p18f8621.h>
#elif defined(__18F6621)
  #include <p18f6621.h>
#elif defined(__18F8525)
  #include <p18f8525.h>
#elif defined(__18F6525)
  #include <p18f6525.h>
#elif defined(__18F2331)
  #include <p18f2331.h>
#elif defined(__18F2431)
  #include <p18f2431.h>
#elif defined(__18F4331)
  #include <p18f4331.h>
#elif defined(__18F4431)
  #include <p18f4431.h>
#elif defined(__18F6410)
  #include <p18f6410.h>
#elif defined(__18F6490)
  #include <p18f6490.h>
#elif defined(__18F8410)
  #include <p18f8410.h>
#elif defined(__18F8490)
  #include <p18f8490.h>
#elif defined(__18F2585)
  #include <p18f2585.h>
#elif defined(__18F2680)
  #include <p18f2680.h>
#elif defined(__18F4585)
```

```
  #include <p18f4585.h>
#elif defined(__18F4680)
  #include <p18f4680.h>
#elif defined(__18F2510)
  #include <p18f2510.h>
#elif defined(__18F2520)
  #include <p18f2520.h>
#elif defined(__18F4510)
  #include <p18f4510.h>
#elif defined(__18F4520)
  #include <p18f4520.h>
#elif defined(__18F2455)
  #include <p18f2455.h>
#elif defined(__18F2550)
  #include <p18f2550.h>
#elif defined(__18F4455)
  #include <p18f4455.h>
#elif defined(__18F4550)
  #include <p18f4550.h>
#elif defined(__18F6310)
  #include <p18f6310.h>
#elif defined(__18F6390)
  #include <p18f6390.h>
#elif defined(__18F8310)
  #include <p18f8310.h>
#elif defined(__18F8390)
  #include <p18f8390.h>
#elif defined(__18F6627)
  #include <p18f6627.h>
#elif defined(__18F6722)
  #include <p18f6722.h>
#elif defined(__18F8627)
  #include <p18f8627.h>
#elif defined(__18F8722)
  #include <p18f8722.h>
#elif defined(__18F64J15)
  #include <p18f64j15.h>
#elif defined(__18F65J10)
  #include <p18f65j10.h>
#elif defined(__18F65J15)
  #include <p18f65j15.h>
#elif defined(__18F66J10)
  #include <p18f66j10.h>
#elif defined(__18F66J15)
  #include <p18f66j15.h>
#elif defined(__18F67J10)
  #include <p18f67j10.h>
#elif defined(__18F84J15)
  #include <p18f84j15.h>
#elif defined(__18F85J10)
  #include <p18f85j10.h>
#elif defined(__18F85J15)
  #include <p18f85j15.h>
#elif defined(__18F86J10)
  #include <p18f86j10.h>
#elif defined(__18F86J15)
  #include <p18f86j15.h>
#elif defined(__18F87J10)
  #include <p18f87j10.h>
#elif defined(__18F2410)
  #include <p18f2410.h>
#elif defined(__18F2420)
  #include <p18f2420.h>
#elif defined(__18F4410)
  #include <p18f4410.h>
#elif defined(__18F4420)
  #include <p18f4420.h>
```

```
#elif defined(__18F2480)
  #include <p18f2480.h>
#elif defined(__18F2580)
  #include <p18f2580.h>
#elif defined(__18F4480)
  #include <p18f4480.h>
#elif defined(__18F4580)
  #include <p18f4580.h>
#elif defined(__18F6527)
  #include <p18f6527.h>
#elif defined(__18F6622)
  #include <p18f6622.h>
#elif defined(__18F8527)
  #include <p18f8527.h>
#elif defined(__18F8622)
  #include <p18f8622.h>
#elif defined(__18F24J10)
  #include <p18f24j10.h>
#elif defined(__18F25J10)
  #include <p18f25j10.h>
#elif defined(__18F44J10)
  #include <p18f44j10.h>
#elif defined(__18F45J10)
  #include <p18f45j10.h>
#elif defined(__18F4321)
  #include <p18f4321.h>
#elif defined(__18F4221)
  #include <p18f4221.h>
#elif defined(__18F2321)
  #include <p18f2321.h>
#elif defined(__18F2221)
  #include <p18f2221.h>
#elif defined(__18F97J60)
  #include <p18f97j60.h>
#else
  #error  Unknown processor!
#endif

#endif /* _P18CXXX_H */
```

6.1.4. p18f2550.h

```
/*********************************************************************
*
*                     PIC18F2550 specific library
*
*********************************************************************
* FileName:        p18f2550.h
* Dependencies:    None
* Processor:       PIC18
* Compiler:        Microchip Cxx
* Company:         Microchip Technology, Inc.
*********************************************************************/

/*-------------------------------------------------------------------
* $Id: p18f2550.h,v 1.7 2004/10/06 14:02:43 curtiss Exp $
* MPLAB-Cxx  PIC18F2550 processor header
*
* (c) Copyright 1999-2004 Microchip Technology, All rights reserved
*-------------------------------------------------------------------*/

#ifndef __18F2550_H
#define __18F2550_H

extern volatile near unsigned            UFRM;
extern volatile near unsigned char       UFRML;
extern volatile near unsigned char       UFRMH;
extern volatile near unsigned char       UIR;
extern volatile near struct {
  unsigned URSTIF:1;
  unsigned UERRIF:1;
  unsigned ACTVIF:1;
  unsigned TRNIF:1;
  unsigned IDLEIF:1;
  unsigned STALLIF:1;
  unsigned SOFIF:1;
} UIRbits;
extern volatile near unsigned char       UIE;
extern volatile near struct {
  unsigned URSTIE:1;
  unsigned UERRIE:1;
  unsigned ACTVIE:1;
  unsigned TRNIE:1;
  unsigned IDLEIE:1;
  unsigned STALLIE:1;
  unsigned SOFIE:1;
} UIEbits;
extern volatile near unsigned char       UEIR;
extern volatile near struct {
  unsigned PIDEF:1;
  unsigned CRC5EF:1;
  unsigned CRC16EF:1;
  unsigned DFN8EF:1;
  unsigned BTOEF:1;
  unsigned :2;
  unsigned BTSEF:1;
} UEIRbits;
extern volatile near unsigned char       UEIE;
extern volatile near struct {
  unsigned PIDEE:1;
  unsigned CRC5EE:1;
  unsigned CRC16EE:1;
  unsigned DFN8EE:1;
  unsigned BTOEE:1;
  unsigned :2;
  unsigned BTSEE:1;
} UEIEbits;
```

```
extern volatile near unsigned char       USTAT;
extern volatile near struct {
  unsigned :1;
  unsigned PPBI:1;
  unsigned DIR:1;
  unsigned ENDP0:1;
  unsigned ENDP1:1;
  unsigned ENDP2:1;
  unsigned ENDP3:1;
} USTATbits;
extern          near unsigned char       UCON;
extern          near struct {
  unsigned :1;
  unsigned SUSPND:1;
  unsigned RESUME:1;
  unsigned USBEN:1;
  unsigned PKTDIS:1;
  unsigned SE0:1;
  unsigned PPBRST:1;
} UCONbits;
extern          near unsigned char       UADDR;
extern          near struct {
  unsigned ADDR0:1;
  unsigned ADDR1:1;
  unsigned ADDR2:1;
  unsigned ADDR3:1;
  unsigned ADDR4:1;
  unsigned ADDR5:1;
  unsigned ADDR6:1;
} UADDRbits;
extern          near unsigned char       UCFG;
extern          near struct {
  unsigned PPB0:1;
  unsigned PPB1:1;
  unsigned FSEN:1;
  unsigned UTRDIS:1;
  unsigned UPUEN:1;
  unsigned :1;
  unsigned UOEMON:1;
  unsigned UTEYE:1;
} UCFGbits;
extern          near unsigned char       UEP0;
extern          near struct {
  unsigned EPSTALL:1;
  unsigned EPINEN:1;
  unsigned EPOUTEN:1;
  unsigned EPCONDIS:1;
  unsigned EPHSHK:1;
} UEP0bits;
extern          near unsigned char       UEP1;
extern          near struct {
  unsigned EPSTALL:1;
  unsigned EPINEN:1;
  unsigned EPOUTEN:1;
  unsigned EPCONDIS:1;
  unsigned EPHSHK:1;
} UEP1bits;
extern          near unsigned char       UEP2;
extern          near struct {
  unsigned EPSTALL:1;
  unsigned EPINEN:1;
  unsigned EPOUTEN:1;
  unsigned EPCONDIS:1;
  unsigned EPHSHK:1;
} UEP2bits;
extern          near unsigned char       UEP3;
extern          near struct {
```

```
  unsigned EPSTALL:1;
  unsigned EPINEN:1;
  unsigned EPOUTEN:1;
  unsigned EPCONDIS:1;
  unsigned EPHSHK:1;
} UEP3bits;
extern          near unsigned char       UEP4;
extern          near struct {
  unsigned EPSTALL:1;
  unsigned EPINEN:1;
  unsigned EPOUTEN:1;
  unsigned EPCONDIS:1;
  unsigned EPHSHK:1;
} UEP4bits;
extern          near unsigned char       UEP5;
extern          near struct {
  unsigned EPSTALL:1;
  unsigned EPINEN:1;
  unsigned EPOUTEN:1;
  unsigned EPCONDIS:1;
  unsigned EPHSHK:1;
} UEP5bits;
extern          near unsigned char       UEP6;
extern          near struct {
  unsigned EPSTALL:1;
  unsigned EPINEN:1;
  unsigned EPOUTEN:1;
  unsigned EPCONDIS:1;
  unsigned EPHSHK:1;
} UEP6bits;
extern          near unsigned char       UEP7;
extern          near struct {
  unsigned EPSTALL:1;
  unsigned EPINEN:1;
  unsigned EPOUTEN:1;
  unsigned EPCONDIS:1;
  unsigned EPHSHK:1;
} UEP7bits;
extern          near unsigned char       UEP8;
extern          near struct {
  unsigned EPSTALL:1;
  unsigned EPINEN:1;
  unsigned EPOUTEN:1;
  unsigned EPCONDIS:1;
  unsigned EPHSHK:1;
} UEP8bits;
extern          near unsigned char       UEP9;
extern          near struct {
  unsigned EPSTALL:1;
  unsigned EPINEN:1;
  unsigned EPOUTEN:1;
  unsigned EPCONDIS:1;
  unsigned EPHSHK:1;
} UEP9bits;
extern          near unsigned char       UEP10;
extern          near struct {
  unsigned EPSTALL:1;
  unsigned EPINEN:1;
  unsigned EPOUTEN:1;
  unsigned EPCONDIS:1;
  unsigned EPHSHK:1;
} UEP10bits;
extern          near unsigned char       UEP11;
extern          near struct {
  unsigned EPSTALL:1;
  unsigned EPINEN:1;
  unsigned EPOUTEN:1;
```

```
  unsigned EPCONDIS:1;
  unsigned EPHSHK:1;
} UEP11bits;
extern          near unsigned char       UEP12;
extern          near struct {
  unsigned EPSTALL:1;
  unsigned EPINEN:1;
  unsigned EPOUTEN:1;
  unsigned EPCONDIS:1;
  unsigned EPHSHK:1;
} UEP12bits;
extern          near unsigned char       UEP13;
extern          near struct {
  unsigned EPSTALL:1;
  unsigned EPINEN:1;
  unsigned EPOUTEN:1;
  unsigned EPCONDIS:1;
  unsigned EPHSHK:1;
} UEP13bits;
extern          near unsigned char       UEP14;
extern          near struct {
  unsigned EPSTALL:1;
  unsigned EPINEN:1;
  unsigned EPOUTEN:1;
  unsigned EPCONDIS:1;
  unsigned EPHSHK:1;
} UEP14bits;
extern          near unsigned char       UEP15;
extern          near struct {
  unsigned EPSTALL:1;
  unsigned EPINEN:1;
  unsigned EPOUTEN:1;
  unsigned EPCONDIS:1;
  unsigned EPHSHK:1;
} UEP15bits;
extern volatile near unsigned char       PORTA;
extern volatile near union {
  struct {
    unsigned RA0:1;
    unsigned RA1:1;
    unsigned RA2:1;
    unsigned RA3:1;
    unsigned RA4:1;
    unsigned RA5:1;
    unsigned RA6:1;
  };
  struct {
    unsigned AN0:1;
    unsigned AN1:1;
    unsigned AN2:1;
    unsigned AN3:1;
    unsigned T0CKI:1;
    unsigned AN4:1;
    unsigned OSC2:1;
  };
  struct {
    unsigned :2;
    unsigned VREFM:1;
    unsigned VREFP:1;
    unsigned :1;
    unsigned LVDIN:1;
  };
  struct {
    unsigned :5;
    unsigned HLVDIN:1;
  };
} PORTAbits;
```

```
extern volatile near unsigned char       PORTB;
extern volatile near union {
  struct {
    unsigned RB0:1;
    unsigned RB1:1;
    unsigned RB2:1;
    unsigned RB3:1;
    unsigned RB4:1;
    unsigned RB5:1;
    unsigned RB6:1;
    unsigned RB7:1;
  };
  struct {
    unsigned INT0:1;
    unsigned INT1:1;
    unsigned INT2:1;
  };
  struct {
    unsigned :5;
    unsigned PGM:1;
    unsigned PGC:1;
    unsigned PGD:1;
  };
} PORTBbits;
extern volatile near unsigned char       PORTC;
extern volatile near union {
  struct {
    unsigned RC0:1;
    unsigned RC1:1;
    unsigned RC2:1;
    unsigned :1;
    unsigned RC4:1;
    unsigned RC5:1;
    unsigned RC6:1;
    unsigned RC7:1;
  };
  struct {
    unsigned T1OSO:1;
    unsigned T1OSI:1;
    unsigned CCP1:1;
    unsigned :3;
    unsigned TX:1;
    unsigned RX:1;
  };
  struct {
    unsigned T13CKI:1;
    unsigned :1;
    unsigned P1A:1;
    unsigned :3;
    unsigned CK:1;
    unsigned DT:1;
  };
} PORTCbits;
extern volatile near unsigned char       PORTE;
extern volatile near struct {
  unsigned :3;
  unsigned RE3:1;
} PORTEbits;
extern volatile near unsigned char       LATA;
extern volatile near struct {
  unsigned LATA0:1;
  unsigned LATA1:1;
  unsigned LATA2:1;
  unsigned LATA3:1;
  unsigned LATA4:1;
  unsigned LATA5:1;
  unsigned LATA6:1;
```

```
} LATAbits;
extern volatile near unsigned char       LATB;
extern volatile near struct {
  unsigned LATB0:1;
  unsigned LATB1:1;
  unsigned LATB2:1;
  unsigned LATB3:1;
  unsigned LATB4:1;
  unsigned LATB5:1;
  unsigned LATB6:1;
  unsigned LATB7:1;
} LATBbits;
extern volatile near unsigned char       LATC;
extern volatile near struct {
  unsigned LATC0:1;
  unsigned LATC1:1;
  unsigned LATC2:1;
  unsigned :3;
  unsigned LATC6:1;
  unsigned LATC7:1;
} LATCbits;
extern volatile near unsigned char       DDRA;
extern volatile near struct {
  unsigned RA0:1;
  unsigned RA1:1;
  unsigned RA2:1;
  unsigned RA3:1;
  unsigned RA4:1;
  unsigned RA5:1;
  unsigned RA6:1;
} DDRAbits;
extern volatile near unsigned char       TRISA;
extern volatile near struct {
  unsigned TRISA0:1;
  unsigned TRISA1:1;
  unsigned TRISA2:1;
  unsigned TRISA3:1;
  unsigned TRISA4:1;
  unsigned TRISA5:1;
  unsigned TRISA6:1;
} TRISAbits;
extern volatile near unsigned char       DDRB;
extern volatile near struct {
  unsigned RB0:1;
  unsigned RB1:1;
  unsigned RB2:1;
  unsigned RB3:1;
  unsigned RB4:1;
  unsigned RB5:1;
  unsigned RB6:1;
  unsigned RB7:1;
} DDRBbits;
extern volatile near unsigned char       TRISB;
extern volatile near struct {
  unsigned TRISB0:1;
  unsigned TRISB1:1;
  unsigned TRISB2:1;
  unsigned TRISB3:1;
  unsigned TRISB4:1;
  unsigned TRISB5:1;
  unsigned TRISB6:1;
  unsigned TRISB7:1;
} TRISBbits;
extern volatile near unsigned char       DDRC;
extern volatile near struct {
  unsigned RC0:1;
  unsigned RC1:1;
```

```
  unsigned RC2:1;
  unsigned :3;
  unsigned RC6:1;
  unsigned RC7:1;
} DDRCbits;
extern volatile near unsigned char       TRISC;
extern volatile near struct {
  unsigned TRISC0:1;
  unsigned TRISC1:1;
  unsigned TRISC2:1;
  unsigned :3;
  unsigned TRISC6:1;
  unsigned TRISC7:1;
} TRISCbits;
extern volatile near unsigned char       OSCTUNE;
extern volatile near struct {
  unsigned TUN0:1;
  unsigned TUN1:1;
  unsigned TUN2:2;
  unsigned TUN3:1;
  unsigned TUN4:1;
  unsigned :2;
  unsigned INTSRC:1;
} OSCTUNEbits;
extern volatile near unsigned char       PIE1;
extern volatile near struct {
  unsigned TMR1IE:1;
  unsigned TMR2IE:1;
  unsigned CCP1IE:1;
  unsigned SSPIE:1;
  unsigned TXIE:1;
  unsigned RCIE:1;
  unsigned ADIE:1;
} PIE1bits;
extern volatile near unsigned char       PIR1;
extern volatile near struct {
  unsigned TMR1IF:1;
  unsigned TMR2IF:1;
  unsigned CCP1IF:1;
  unsigned SSPIF:1;
  unsigned TXIF:1;
  unsigned RCIF:1;
  unsigned ADIF:1;
} PIR1bits;
extern volatile near unsigned char       IPR1;
extern volatile near struct {
  unsigned TMR1IP:1;
  unsigned TMR2IP:1;
  unsigned CCP1IP:1;
  unsigned SSPIP:1;
  unsigned TXIP:1;
  unsigned RCIP:1;
  unsigned ADIP:1;
} IPR1bits;
extern volatile near unsigned char       PIE2;
extern volatile near union {
  struct {
    unsigned CCP2IE:1;
    unsigned TMR3IE:1;
    unsigned LVDIE:1;
    unsigned BCLIE:1;
    unsigned EEIE:1;
    unsigned USBIE:1;
    unsigned CMIE:1;
    unsigned OSCFIE:1;
  };
  struct {
```

```
    unsigned :2;
    unsigned HLVDIE:1;
  };
} PIE2bits;
extern volatile near unsigned char       PIR2;
extern volatile near union {
  struct {
    unsigned CCP2IF:1;
    unsigned TMR3IF:1;
    unsigned LVDIF:1;
    unsigned BCLIF:1;
    unsigned EEIF:1;
    unsigned USBIF:1;
    unsigned CMIF:1;
    unsigned OSCFIF:1;
  };
  struct {
    unsigned :2;
    unsigned HLVDIF:1;
  };
} PIR2bits;
extern volatile near unsigned char       IPR2;
extern volatile near union {
  struct {
    unsigned CCP2IP:1;
    unsigned TMR3IP:1;
    unsigned LVDIP:1;
    unsigned BCLIP:1;
    unsigned EEIP:1;
    unsigned USBIP:1;
    unsigned CMIP:1;
    unsigned OSCFIP:1;
  };
  struct {
    unsigned :2;
    unsigned HLVDIP:1;
  };
} IPR2bits;
extern volatile near unsigned char       EECON1;
extern volatile near struct {
  unsigned RD:1;
  unsigned WR:1;
  unsigned WREN:1;
  unsigned WRERR:1;
  unsigned FREE:1;
  unsigned :1;
  unsigned CFGS:1;
  unsigned EEPGD:1;
} EECON1bits;
extern volatile near unsigned char       EECON2;
extern volatile near unsigned char       EEDATA;
extern volatile near unsigned char       EEADR;
extern volatile near unsigned char       RCSTA;
extern volatile near union {
  struct {
    unsigned RX9D:1;
    unsigned OERR:1;
    unsigned FERR:1;
    unsigned ADDEN:1;
    unsigned CREN:1;
    unsigned SREN:1;
    unsigned RX9:1;
    unsigned SPEN:1;
  };
  struct {
    unsigned :3;
    unsigned ADEN:1;
```

```
  };
} RCSTAbits;
extern volatile near unsigned char       TXSTA;
extern volatile near struct {
  unsigned TX9D:1;
  unsigned TRMT:1;
  unsigned BRGH:1;
  unsigned SENDB:1;
  unsigned SYNC:1;
  unsigned TXEN:1;
  unsigned TX9:1;
  unsigned CSRC:1;
} TXSTAbits;
extern volatile near unsigned char       TXREG;
extern volatile near unsigned char       RCREG;
extern volatile near unsigned char       SPBRG;
extern volatile near unsigned char       SPBRGH;
extern volatile near unsigned char       T3CON;
extern volatile near union {
  struct {
    unsigned TMR3ON:1;
    unsigned TMR3CS:1;
    unsigned T3SYNC:1;
    unsigned T3CCP1:1;
    unsigned T3CKPS0:1;
    unsigned T3CKPS1:1;
    unsigned T3CCP2:1;
    unsigned RD16:1;
  };
  struct {
    unsigned :2;
    unsigned T3NSYNC:1;
  };
  struct {
    unsigned :2;
    unsigned NOT_T3SYNC:1;
  };
} T3CONbits;
extern volatile near unsigned char       TMR3L;
extern volatile near unsigned char       TMR3H;
extern volatile near unsigned char       CMCON;
extern volatile near struct {
  unsigned CM0:1;
  unsigned CM1:1;
  unsigned CM2:1;
  unsigned CIS:1;
  unsigned C1INV:1;
  unsigned C2INV:1;
  unsigned C1OUT:1;
  unsigned C2OUT:1;
} CMCONbits;
extern volatile near unsigned char       CVRCON;
extern volatile near union {
  struct {
    unsigned CVR0:1;
    unsigned CVR1:1;
    unsigned CVR2:1;
    unsigned CVR3:1;
    unsigned CVREF:1;
    unsigned CVRR:1;
    unsigned CVROE:1;
    unsigned CVREN:1;
  };
  struct {
    unsigned :4;
    unsigned CVRSS:1;
  };
```

```
} CVRCONbits;
extern volatile near unsigned char       CCP1AS;
extern volatile near struct {
  unsigned :2;
  unsigned PSSAC0:1;
  unsigned PSSAC1:1;
  unsigned ECCPAS0:1;
  unsigned ECCPAS1:1;
  unsigned ECCPAS2:1;
  unsigned ECCPASE:1;
} CCP1ASbits;
extern volatile near unsigned char       CCP1DEL;
extern volatile near struct {
  unsigned filler0:7;
  unsigned PRSEN:1;
} CCP1DELbits;
extern volatile near unsigned char       BAUDCON;
extern volatile near union {
  struct {
    unsigned ABDEN:1;
    unsigned WUE:1;
    unsigned :1;
    unsigned BRG16:1;
    unsigned SCKP:1;
    unsigned :1;
    unsigned RCIDL:1;
    unsigned ABDOVF:1;
  };
  struct {
    unsigned :6;
    unsigned RCMT:1;
  };
} BAUDCONbits;
extern volatile near unsigned char       CCP2CON;
extern volatile near struct {
  unsigned CCP2M0:1;
  unsigned CCP2M1:1;
  unsigned CCP2M2:1;
  unsigned CCP2M3:1;
  unsigned DC2B0:1;
  unsigned DC2B1:1;
} CCP2CONbits;
extern volatile near unsigned            CCPR2;
extern volatile near unsigned char       CCPR2L;
extern volatile near unsigned char       CCPR2H;
extern volatile near unsigned char       CCP1CON;
extern volatile near struct {
  unsigned CCP1M0:1;
  unsigned CCP1M1:1;
  unsigned CCP1M2:1;
  unsigned CCP1M3:1;
  unsigned DC1B0:1;
  unsigned DC1B1:1;
} CCP1CONbits;
extern volatile near unsigned            CCPR1;
extern volatile near unsigned char       CCPR1L;
extern volatile near unsigned char       CCPR1H;
extern volatile near unsigned char       ADCON2;
extern volatile near struct {
  unsigned ADCS0:1;
  unsigned ADCS1:1;
  unsigned ADCS2:1;
  unsigned ACQT0:1;
  unsigned ACQT1:1;
  unsigned ACQT2:1;
  unsigned :1;
  unsigned ADFM:1;
```

```
} ADCON2bits;
extern volatile near unsigned char       ADCON1;
extern volatile near struct {
  unsigned PCFG0:1;
  unsigned PCFG1:1;
  unsigned PCFG2:1;
  unsigned PCFG3:1;
  unsigned VCFG0:1;
  unsigned VCFG1:1;
} ADCON1bits;
extern volatile near unsigned char       ADCON0;
extern volatile near union {
  struct {
    unsigned ADON:1;
    unsigned GO_DONE:1;
    unsigned CHS0:1;
    unsigned CHS1:1;
    unsigned CHS2:1;
    unsigned CHS3:1;
  };
  struct {
    unsigned :1;
    unsigned DONE:1;
  };
  struct {
    unsigned :1;
    unsigned GO:1;
  };
  struct {
    unsigned :1;
    unsigned NOT_DONE:1;
  };
} ADCON0bits;
extern volatile near unsigned            ADRES;
extern volatile near unsigned char       ADRESL;
extern volatile near unsigned char       ADRESH;
extern volatile near unsigned char       SSPCON2;
extern volatile near struct {
  unsigned SEN:1;
  unsigned RSEN:1;
  unsigned PEN:1;
  unsigned RCEN:1;
  unsigned ACKEN:1;
  unsigned ACKDT:1;
  unsigned ACKSTAT:1;
  unsigned GCEN:1;
} SSPCON2bits;
extern volatile near unsigned char       SSPCON1;
extern volatile near struct {
  unsigned SSPM0:1;
  unsigned SSPM1:1;
  unsigned SSPM2:1;
  unsigned SSPM3:1;
  unsigned CKP:1;
  unsigned SSPEN:1;
  unsigned SSPOV:1;
  unsigned WCOL:1;
} SSPCON1bits;
extern volatile near unsigned char       SSPSTAT;
extern volatile near union {
  struct {
    unsigned BF:1;
    unsigned UA:1;
    unsigned R_W:1;
    unsigned S:1;
    unsigned P:1;
    unsigned D_A:1;
```

```
    unsigned CKE:1;
    unsigned SMP:1;
  };
  struct {
    unsigned :2;
    unsigned I2C_READ:1;
    unsigned I2C_START:1;
    unsigned I2C_STOP:1;
    unsigned I2C_DAT:1;
  };
  struct {
    unsigned :2;
    unsigned NOT_W:1;
    unsigned :2;
    unsigned NOT_A:1;
  };
  struct {
    unsigned :2;
    unsigned NOT_WRITE:1;
    unsigned :2;
    unsigned NOT_ADDRESS:1;
  };
  struct {
    unsigned :2;
    unsigned READ_WRITE:1;
    unsigned :2;
    unsigned DATA_ADDRESS:1;
  };
  struct {
    unsigned :2;
    unsigned R:1;
    unsigned :2;
    unsigned D:1;
  };
} SSPSTATbits;
extern volatile near unsigned char       SSPADD;
extern volatile near unsigned char       SSPBUF;
extern volatile near unsigned char       T2CON;
extern volatile near struct {
  unsigned T2CKPS0:1;
  unsigned T2CKPS1:1;
  unsigned TMR2ON:1;
  unsigned T2OUTPS0:1;
  unsigned T2OUTPS1:1;
  unsigned T2OUTPS2:1;
  unsigned T2OUTPS3:1;
} T2CONbits;
extern volatile near unsigned char       PR2;
extern volatile near unsigned char       TMR2;
extern volatile near unsigned char       T1CON;
extern volatile near union {
  struct {
    unsigned TMR1ON:1;
    unsigned TMR1CS:1;
    unsigned T1SYNC:1;
    unsigned T1OSCEN:1;
    unsigned T1CKPS0:1;
    unsigned T1CKPS1:1;
    unsigned T1RUN:1;
    unsigned RD16:1;
  };
  struct {
    unsigned :2;
    unsigned NOT_T1SYNC:1;
  };
} T1CONbits;
extern volatile near unsigned char       TMR1L;
```

```
extern volatile near unsigned char       TMR1H;
extern volatile near unsigned char       RCON;
extern volatile near union {
  struct {
    unsigned NOT_BOR:1;
    unsigned NOT_POR:1;
    unsigned NOT_PD:1;
    unsigned NOT_TO:1;
    unsigned NOT_RI:1;
    unsigned :1;
    unsigned SBOREN:1;
    unsigned NOT_IPEN:1;
  };
  struct {
    unsigned BOR:1;
    unsigned POR:1;
    unsigned PD:1;
    unsigned TO:1;
    unsigned RI:1;
    unsigned :2;
    unsigned IPEN:1;
  };
} RCONbits;
extern volatile near unsigned char       WDTCON;
extern volatile near union {
  struct {
    unsigned SWDTEN:1;
  };
  struct {
    unsigned SWDTE:1;
  };
} WDTCONbits;
extern volatile near unsigned char       HLVDCON;
extern volatile near union {
  struct {
    unsigned LVDL0:1;
    unsigned LVDL1:1;
    unsigned LVDL2:1;
    unsigned LVDL3:1;
    unsigned LVDEN:1;
    unsigned IRVST:1;
  };
  struct {
    unsigned LVV0:1;
    unsigned LVV1:1;
    unsigned LVV2:1;
    unsigned LVV3:1;
    unsigned :1;
    unsigned BGST:1;
  };
  struct {
    unsigned HLVDL0:1;
    unsigned HLVDL1:1;
    unsigned HLVDL2:1;
    unsigned HLVDL3:1;
    unsigned HLVDEN:1;
    unsigned :2;
    unsigned VDIRMAG:1;
  };
  struct {
    unsigned :5;
    unsigned IVRST:1;
  };
} HLVDCONbits;
extern volatile near unsigned char       LVDCON;
extern volatile near union {
  struct {
```

```
    unsigned LVDL0:1;
    unsigned LVDL1:1;
    unsigned LVDL2:1;
    unsigned LVDL3:1;
    unsigned LVDEN:1;
    unsigned IRVST:1;
  };
  struct {
    unsigned LVV0:1;
    unsigned LVV1:1;
    unsigned LVV2:1;
    unsigned LVV3:1;
    unsigned :1;
    unsigned BGST:1;
  };
  struct {
    unsigned HLVDL0:1;
    unsigned HLVDL1:1;
    unsigned HLVDL2:1;
    unsigned HLVDL3:1;
    unsigned HLVDEN:1;
    unsigned :2;
    unsigned VDIRMAG:1;
  };
  struct {
    unsigned :5;
    unsigned IVRST:1;
  };
} LVDCONbits;
extern volatile near unsigned char       OSCCON;
extern volatile near union {
  struct {
    unsigned SCS0:1;
    unsigned SCS1:1;
    unsigned IOFS:1;
    unsigned OSTS:1;
    unsigned IRCF0:1;
    unsigned IRCF1:1;
    unsigned IRCF2:1;
    unsigned IDLEN:1;
  };
  struct {
    unsigned :2;
    unsigned FLTS:1;
  };
} OSCCONbits;
extern volatile near unsigned char       T0CON;
extern volatile near struct {
  unsigned T0PS0:1;
  unsigned T0PS1:1;
  unsigned T0PS2:1;
  unsigned PSA:1;
  unsigned T0SE:1;
  unsigned T0CS:1;
  unsigned T08BIT:1;
  unsigned TMR0ON:1;
} T0CONbits;
extern volatile near unsigned char       TMR0L;
extern volatile near unsigned char       TMR0H;
extern          near unsigned char       STATUS;
extern          near struct {
  unsigned C:1;
  unsigned DC:1;
  unsigned Z:1;
  unsigned OV:1;
  unsigned N:1;
} STATUSbits;
```

```
extern          near unsigned           FSR2;
extern          near unsigned char      FSR2L;
extern          near unsigned char      FSR2H;
extern volatile near unsigned char      PLUSW2;
extern volatile near unsigned char      PREINC2;
extern volatile near unsigned char      POSTDEC2;
extern volatile near unsigned char      POSTINC2;
extern          near unsigned char      INDF2;
extern          near unsigned char      BSR;
extern          near unsigned           FSR1;
extern          near unsigned char      FSR1L;
extern          near unsigned char      FSR1H;
extern volatile near unsigned char      PLUSW1;
extern volatile near unsigned char      PREINC1;
extern volatile near unsigned char      POSTDEC1;
extern volatile near unsigned char      POSTINC1;
extern          near unsigned char      INDF1;
extern          near unsigned char      WREG;
extern          near unsigned           FSR0;
extern          near unsigned char      FSR0L;
extern          near unsigned char      FSR0H;
extern volatile near unsigned char      PLUSW0;
extern volatile near unsigned char      PREINC0;
extern volatile near unsigned char      POSTDEC0;
extern volatile near unsigned char      POSTINC0;
extern          near unsigned char      INDF0;
extern volatile near unsigned char      INTCON3;
extern volatile near union {
  struct {
    unsigned INT1IF:1;
    unsigned INT2IF:1;
    unsigned :1;
    unsigned INT1IE:1;
    unsigned INT2IE:1;
    unsigned :1;
    unsigned INT1IP:1;
    unsigned INT2IP:1;
  };
  struct {
    unsigned INT1F:1;
    unsigned INT2F:1;
    unsigned :1;
    unsigned INT1E:1;
    unsigned INT2E:1;
    unsigned :1;
    unsigned INT1P:1;
    unsigned INT2P:1;
  };
} INTCON3bits;
extern volatile near unsigned char      INTCON2;
extern volatile near union {
  struct {
    unsigned RBIP:1;
    unsigned :1;
    unsigned TMR0IP:1;
    unsigned :1;
    unsigned INTEDG2:1;
    unsigned INTEDG1:1;
    unsigned INTEDG0:1;
    unsigned NOT_RBPU:1;
  };
  struct {
    unsigned :2;
    unsigned T0IP:1;
    unsigned :4;
    unsigned RBPU:1;
  };
```

```
} INTCON2bits;
extern volatile near unsigned char       INTCON;
extern volatile near union {
  struct {
    unsigned RBIF:1;
    unsigned INT0IF:1;
    unsigned TMR0IF:1;
    unsigned RBIE:1;
    unsigned INT0IE:1;
    unsigned TMR0IE:1;
    unsigned PEIE:1;
    unsigned GIE:1;
  };
  struct {
    unsigned :1;
    unsigned INT0F:1;
    unsigned T0IF:1;
    unsigned :1;
    unsigned INT0E:1;
    unsigned T0IE:1;
    unsigned GIEL:1;
    unsigned GIEH:1;
  };
} INTCONbits;
extern          near unsigned            PROD;
extern          near unsigned char       PRODL;
extern          near unsigned char       PRODH;
extern volatile near unsigned char       TABLAT;
extern volatile near unsigned short long TBLPTR;
extern volatile near unsigned char       TBLPTRL;
extern volatile near unsigned char       TBLPTRH;
extern volatile near unsigned char       TBLPTRU;
extern volatile near unsigned short long PC;
extern volatile near unsigned char       PCL;
extern volatile near unsigned char       PCLATH;
extern volatile near unsigned char       PCLATU;
extern volatile near unsigned char       STKPTR;
extern volatile near struct {
  unsigned STKPTR0:1;
  unsigned STKPTR1:1;
  unsigned STKPTR2:1;
  unsigned STKPTR3:1;
  unsigned STKPTR4:1;
  unsigned :1;
  unsigned STKUNF:1;
  unsigned STKFUL:1;
} STKPTRbits;
extern          near unsigned short long TOS;
extern          near unsigned char       TOSL;
extern          near unsigned char       TOSH;
extern          near unsigned char       TOSU;


/*--------------------------------------------------------------------------
 * Some useful defines for inline assembly stuff
 *--------------------------------------------------------------------------*/
#define ACCESS 0
#define BANKED 1

/*--------------------------------------------------------------------------
 * Some useful macros for inline assembly stuff
 *--------------------------------------------------------------------------*/
#define Nop()    {_asm nop _endasm}
#define ClrWdt() {_asm clrwdt _endasm}
#define Sleep()  {_asm sleep _endasm}
#define Reset()  {_asm reset _endasm}
```

```
#define Rlcf(f,dest,access)  {_asm movlb f rlcf f,dest,access _endasm}
#define Rlncf(f,dest,access) {_asm movlb f rlncf f,dest,access _endasm}
#define Rrcf(f,dest,access)  {_asm movlb f rrcf f,dest,access _endasm}
#define Rrncf(f,dest,access) {_asm movlb f rrncf f,dest,access _endasm}
#define Swapf(f,dest,access) {_asm movlb f swapf f,dest,access _endasm }

/*-----------------------------------------------------------------------
 * A fairly inclusive set of registers to save for interrupts.
 * These are locations which are commonly used by the compiler.
 *-----------------------------------------------------------------------*/
#define INTSAVELOCS TBLPTR, TABLAT, PROD


#endif
```

6.1.5. timers.h

```
/*********************************************************************
*
*                   PIC18 timers peripheral library
*
*********************************************************************
* FileName:        timers.h
* Dependencies:    None
* Processor:       PIC18
* Compiler:        Microchip C18
* Company:         Microchip Technology, Inc.
*********************************************************************/

#ifndef __TIMERS_H
#define __TIMERS_H

/* used to hold 16-bit timer value */
union Timers
{
  unsigned int lt;
  char bt[2];
};

/* storage class of library routine parameters; pre-built with auto;
 * do not change unless you rebuild the libraries with the new storage class */
#define PARAM_SCLASS auto

/* Interrupt bit mask to be 'anded' with the other configuration masks and
 * passed as the 'config' parameter to the 'open' routines. */

#ifdef USE_OR_MASKS
#define TIMER_INT_OFF     0b00000000  // Interrupts disabled
#define TIMER_INT_ON      0b10000000  // Interrupts enabled
#define TIMER_INT_MASK    (~TIMER_INT_ON)

/* ***** TIMER0 ***** */
/* TIMER0 configuration masks -- to be 'anded' together and passed to the
 * 'open' routine. */
#define T0_16BIT          0b00000000  // Timer 0 is in 16 Bit mode
#define T0_8BIT           0b01000000  // Timer 0 is in 8 bit mode
#define T0_BIT_MASK       (~T0_8BIT)

#define T0_SOURCE_INT     0b00000000  // Internal clock source
#define T0_SOURCE_EXT     0b00100000  // External clock source
#define T0_SOURCE_MASK    (~T0_SOURCE_EXT)

#define T0_EDGE_RISE      0b00000000  // External rising edge clocs timer
#define T0_EDGE_FALL      0b00010000  // External falling edge clocks timer
#define T0_EDGE_MASK      (~T0_EDGE_FALL)

#define T0_PS_1_1         0b00001000  // Prescaler 1:1 (NO Prescaler)
#define NO_T0_PS_MASK     (~T0_PS_1_1)

#define T0_PS_1_2         0b00000000  //           1:2
#define T0_PS_1_4         0b00000001  //           1:4
#define T0_PS_1_8         0b00000010  //           1:8
#define T0_PS_1_16        0b00000011  //           1:16
#define T0_PS_1_32        0b00000100  //           1:32
#define T0_PS_1_64        0b00000101  //           1:64
#define T0_PS_1_128       0b00000110  //           1:128
#define T0_PS_1_256       0b00000111  //           1:256
#define T0_PS_MASK        (~T0_PS_1_256)
//---------------AND MASK-------------------------------
#else //!USE_OR_MASKS
#define TIMER_INT_OFF  0b01111111  // Interrupts disabled
#define TIMER_INT_ON   0b11111111  // Interrupts enabled
```

```
/* ***** TIMER0 ***** */
/* TIMER0 configuration masks -- to be 'anded' together and passed to the
 * 'open' routine. */
#define T0_16BIT        0b10111111  // Timer 0 is in 16 Bit mode
#define T0_8BIT         0b11111111  // Timer 0 is in 8 bit mode
#define T0_SOURCE_INT   0b11011111  // Internal clock source
#define T0_SOURCE_EXT   0b11111111  // External clock source
#define T0_EDGE_RISE    0b11101111  // External rising edge clocs timer
#define T0_EDGE_FALL    0b11111111  // External falling edge clocks timer
#define T0_PS_1_1       0b11111111  // Prescaler 1:1 (NO Prescaler)
#define T0_PS_1_2       0b11110000  //           1:2
#define T0_PS_1_4       0b11110001  //           1:4
#define T0_PS_1_8       0b11110010  //           1:8
#define T0_PS_1_16      0b11110011  //           1:16
#define T0_PS_1_32      0b11110100  //           1:32
#define T0_PS_1_64      0b11110101  //           1:64
#define T0_PS_1_128     0b11110110  //           1:128
#define T0_PS_1_256     0b11110111  //           1:256

#endif//USE_OR_MASKS

void OpenTimer0 (PARAM_SCLASS unsigned char config);
void CloseTimer0 (void);
unsigned int ReadTimer0 (void);
void WriteTimer0 (PARAM_SCLASS unsigned int timer0);

/* ***** TIMER1 ***** */

/* TIMER1 configuration masks -- to be 'anded' together and passed to the
 * 'open' routine. */

#if defined (TMR_V6)

#ifdef USE_OR_MASKS

#define T1_SOURCE_PINOSC   0b00000000  // Clock source T1OSCEN = 0 Ext clock,
T1OSCEN=1 Crystal osc
#define T1_SOURCE_CAPOSC   0b00100000  // Clock source is for Capacitive Sensing
Oscillator
#define T1_SOURCE_FOSC_4   0b01000000  //Clock source is instruction clock
(FOSC/4)
#define T1_SOURCE_FOSC     0b01100000  //Clock source is system clock (FOSC)
#define       T1_SOURCE_MASK            (~T1_SOURCE_FOSC)

#define T1_PS_1_1          0b00000000  // 1:1 prescale value
#define T1_PS_1_2          0b00001000  // 1:2 prescale value
#define T1_PS_1_4          0b00010000  // 1:4 prescale value
#define T1_PS_1_8          0b00011000  // 1:8 prescale value
#define T1_PS_MASK         (~T1_PS_1_8)

#define T1_OSC1EN_OFF      0b00000000  // Timer 1 oscilator enable off
#define T1_OSC1EN_ON       0b00000100  // Timer 1 oscilator enable on
#define       T1_OSC_MASK               (~T1_OSC1EN_ON)

#define T1_SYNC_EXT_ON       0b00000000  // Synchronize external clock input
#define T1_SYNC_EXT_OFF      0b00000010  // Do not synchronize external clock
input
#define T1_SYNC_MASK         (~T1_SYNC_EXT_OFF)

#define T1_8BIT_RW           0b00000000  // 8-bit mode
#define T1_16BIT_RW          0b00000001  // 16-bit mode
#define T1_BIT_RW_MASK       (~T1_16BIT_RW)

#else
```

```
#define T1_SOURCE_PINOSC   0b10011111  // Clock source T1OSCEN = 0 Ext clock,
T1OSCEN=1 Crystal osc
#define T1_SOURCE_CAPOSC   0b10111111  // Clock source is for Capacitive Sensing
Oscillator
#define T1_SOURCE_FOSC_4   0b11011111  //Clock source is instruction clock
(FOSC/4)
#define T1_SOURCE_FOSC     0b11111111  //Closck source is system clock (FOSC)

#define T1_PS_1_1          0b11100111  // 1:1 prescale value
#define T1_PS_1_2          0b11101111  // 1:2 prescale value
#define T1_PS_1_4          0b11110111  // 1:4 prescale value
#define T1_PS_1_8          0b11111111  // 1:8 prescale value

#define T1_OSC1EN_OFF      0b11111011  // Timer 1 oscilator enable off
#define T1_OSC1EN_ON       0b11111111  // Timer 1 oscilator enable on

#define T1_SYNC_EXT_ON       0b11111101  // Synchronize external clock input
#define T1_SYNC_EXT_OFF      0b11111111  // Do not synchronize external clock
input

#define T1_8BIT_RW           0b11111110  // 8-bit mode
#define T1_16BIT_RW          0b11111111  // 16-bit mode
#define T1_BIT_RW_MASK       (~T1_16BIT_RW)

#endif

void OpenTimer1 (PARAM_SCLASS unsigned char config, PARAM_SCLASS unsigned char
config1);
void CloseTimer1 (void);
unsigned int ReadTimer1 (void);
void WriteTimer1 (PARAM_SCLASS unsigned int timer1);

#else

//--------------AND OR MASK-----------------------------
#ifdef USE_OR_MASKS
#define T1_8BIT_RW         0b00000000  // 8-bit mode
#define T1_16BIT_RW        0b01000000  // 16-bit mode
#define T1_BIT_RW_MASK     (~T1_16BIT_RW)

#define T1_PS_1_1          0b00000000  // 1:1 prescale value
#define T1_PS_1_2          0b00010000  // 1:2 prescale value
#define T1_PS_1_4          0b00100000  // 1:4 prescale value
#define T1_PS_1_8          0b00110000  // 1:8 prescale value
#define T1_PS_MASK        (~T1_PS_1_8)

#define T1_OSC1EN_OFF      0b00000000  // Timer 1 oscilator enable off
#define T1_OSC1EN_ON       0b00001000  // Timer 1 oscilator enable on
#define T1_OSC_MASK       (~T1_OSC1EN_ON)

#define T1_SYNC_EXT_ON     0b00000000  // Synchronize external clock input
#define T1_SYNC_EXT_OFF    0b00000100  // Do not synchronize external clock input
#define T1_SYNC_MASK       (~T1_SYNC_EXT_OFF)

#define T1_SOURCE_INT      0b00000000  // Internal clock source
#define T1_SOURCE_EXT      0b00000010  // External clock source
#define T1_SOURCE_MASK     (~T1_SOURCE_EXT)

//-----------------AND MASK-----------------------------
#else //!USE_OR_MASKS

#define T1_8BIT_RW         0b10111111  // 8-bit mode
#define T1_16BIT_RW        0b11111111  // 16-bit mode
#define T1_PS_1_1          0b11001111  // 1:1 prescale value
#define T1_PS_1_2          0b11011111  // 1:2 prescale value
#define T1_PS_1_4          0b11101111  // 1:4 prescale value
#define T1_PS_1_8          0b11111111  // 1:8 prescale value
```

```
#define T1_OSC1EN_OFF    0b11110111  // Timer 1 oscilator enable off
#define T1_OSC1EN_ON     0b11111111  // Timer 1 oscilator enable on
#define T1_SYNC_EXT_ON   0b11111011  // Synchronize external clock input
#define T1_SYNC_EXT_OFF  0b11111111  // Do not synchronize external clock input
#define T1_SOURCE_INT    0b11111101  // Internal clock source
#define T1_SOURCE_EXT    0b11111111  // External clock source

#endif //USE_OR_MASKS

void OpenTimer1 (PARAM_SCLASS unsigned char config);
void CloseTimer1 (void);
unsigned int ReadTimer1 (void);
void WriteTimer1 (PARAM_SCLASS unsigned int timer1);
#endif

/* ***** TIMER2 ***** */
#if defined (TMR_V2) || defined (TMR_V3) || defined (TMR_V4) ||\
    defined (TMR_V5) || defined (TMR_V6)
/* TIMER2 configuration masks -- to be 'anded' together and passed to the
 * 'open' routine. */
//-----------------AND OR MASK----------------------
#ifdef USE_OR_MASKS
#define T2_POST_1_1    0b00000000  // Postscaler 1:1
#define T2_POST_1_2    0b00001000  // Postscaler 1:2
#define T2_POST_1_3    0b00010000  // Postscaler 1:3
#define T2_POST_1_4    0b00011000  // Postscaler 1:4
#define T2_POST_1_5    0b00100000  // Postscaler 1:5
#define T2_POST_1_6    0b00101000  // Postscaler 1:6
#define T2_POST_1_7    0b00110000  // Postscaler 1:7
#define T2_POST_1_8    0b00111000  // Postscaler 1:8
#define T2_POST_1_9    0b01000000  // Postscaler 1:9
#define T2_POST_1_10   0b01001000  // Postscaler 1:10
#define T2_POST_1_11   0b01010000  // Postscaler 1:11
#define T2_POST_1_12   0b01011000  // Postscaler 1:12
#define T2_POST_1_13   0b01100000  // Postscaler 1:13
#define T2_POST_1_14   0b01101000  // Postscaler 1:14
#define T2_POST_1_15   0b01110000  // Postscaler 1:15
#define T2_POST_1_16   0b01111000  // Postscaler 1:16
#define T2_POST_MASK   (~T2_POST_1_16)

#define T2_PS_1_1      0b00000000  // Prescale 1:1
#define T2_PS_1_4      0b00000001  // Prescale 1:4
#define T2_PS_1_16     0b00000011  // Prescale 1:16
#define T2_PS_MASK     (~T2_PS_1_16)

//-----------------AND MASK-----------------------
#else //!USE_OR_MASKS
#define T2_POST_1_1    0b10000111  // Postscaler 1:1
#define T2_POST_1_2    0b10001111  // Postscaler 1:2
#define T2_POST_1_3    0b10010111  // Postscaler 1:3
#define T2_POST_1_4    0b10011111  // Postscaler 1:4
#define T2_POST_1_5    0b10100111  // Postscaler 1:5
#define T2_POST_1_6    0b10101111  // Postscaler 1:6
#define T2_POST_1_7    0b10110111  // Postscaler 1:7
#define T2_POST_1_8    0b10111111  // Postscaler 1:8
#define T2_POST_1_9    0b11000111  // Postscaler 1:9
#define T2_POST_1_10   0b11001111  // Postscaler 1:10
#define T2_POST_1_11   0b11010111  // Postscaler 1:11
#define T2_POST_1_12   0b11011111  // Postscaler 1:12
#define T2_POST_1_13   0b11100111  // Postscaler 1:13
#define T2_POST_1_14   0b11101111  // Postscaler 1:14
#define T2_POST_1_15   0b11110111  // Postscaler 1:15
#define T2_POST_1_16   0b11111111  // Postscaler 1:16
#define T2_PS_1_1      0b11111100  // Prescale 1:1
#define T2_PS_1_4      0b11111101  // Prescale 1:4
#define T2_PS_1_16     0b11111110  // Prescale 1:16
```

```
#endif //!USE_OR_MASKS

#define WriteTimer2(timer2) TMR2 = (timer2)
#define ReadTimer2()          TMR2

void OpenTimer2 (PARAM_SCLASS unsigned char config);
void CloseTimer2 (void);
//unsigned char ReadTimer2 (void);
#endif
/* ***** TIMER3 ***** */

#if defined(TMR_V1)  || defined(TMR_V2)  || defined(TMR_V4)

/* TIMER3 configuration masks -- to be 'anded' together and passed to the
 * 'open' routine. */
#ifdef USE_OR_MASKS
#define T3_8BIT_RW      0b00000000 // 8-bit mode
#define T3_16BIT_RW     0b00000001 // 16-bit mode
#define T3_BIT_MASK     (~T3_16BIT_RW)

#define T3_PS_1_1       0b00000000 // 1:1 prescale value
#define T3_PS_1_2       0b00010000 // 1:2 prescale value
#define T3_PS_1_4       0b00100000 // 1:4 prescale value
#define T3_PS_1_8       0b00110000 // 1:8 prescale value
#define T3_PS_MASK      (~T3_PS_1_8)

#define T3_SYNC_EXT_ON  0b00000000 // Synchronize external clock input
#define T3_SYNC_EXT_OFF 0b00000100 // Do not synchronize external clock input
#define T3_SYNC_MASK    (~T3_SYNC_EXT_OFF)

#define T3_SOURCE_INT   0b00000000 // Internal clock source
#define T3_SOURCE_EXT   0b00000010 // External clock source
#define T3_SOURCE_MASK  (~T3_SOURCE_EXT)

#else //!USE_OR_MASKS
#define T3_8BIT_RW      0b11111110 // 8-bit mode
#define T3_16BIT_RW     0b11111111 // 16-bit mode
#define T3_PS_1_1       0b11001111 // 1:1 prescale value
#define T3_PS_1_2       0b11011111 // 1:2 prescale value
#define T3_PS_1_4       0b11101111 // 1:4 prescale value
#define T3_PS_1_8       0b11111111 // 1:8 prescale value
#define T3_SYNC_EXT_ON  0b11111011 // Synchronize external clock input
#define T3_SYNC_EXT_OFF 0b11111111 // Do not synchronize external clock input
#define T3_SOURCE_INT   0b11111101 // Internal clock source
#define T3_SOURCE_EXT   0b11111111 // External clock source
#endif //USE_OR_MASKS

//-----------------------------------------------------------------------
/*********************************************************************
Macro        : T3_OSC1EN_ON
Overview     : sets the T1OSCEN bit in the T1CON
Parameters   : None
Remarks      : None.
*********************************************************************/
#define T3_OSC1EN_ON()   T1CONbits.T1OSCEN=1 // Timer 3 oscillator enables

/*********************************************************************
Macro        : T3_OSC1EN_OFF
Overview     : Clears the T1OSCEN bit in the T1CON
Parameters   : None
Remarks      : None.
*********************************************************************/
#define T3_OSC1EN_OFF()   T1CONbits.T1OSCEN=0 // Timer 3 oscillator Disables


void OpenTimer3 (PARAM_SCLASS unsigned char config);
void CloseTimer3 (void);
```

```
unsigned int ReadTimer3 (void);
void WriteTimer3 (PARAM_SCLASS unsigned int timer3);

#elif defined (TMR_V6)

#ifdef USE_OR_MASKS
#define T3_SOURCE_PINOSC   0b00000000  // Clock source T1OSCEN = 0 Ext clock,
T1OSCEN=1 Crystal osc
#define T3_SOURCE_CAPOSC   0b00100000  // Clock source is for Capacitive Sensing
Oscillator
#define T3_SOURCE_FOSC_4   0b01000000  //Clock source is instruction clock
(FOSC/4)
#define T3_SOURCE_FOSC     0b01100000  //Closck source is system clock (FOSC)
#define T3_SOURCE_MASK     (~T1_SOURCE_FOSC)

#define T3_PS_1_1          0b00000000  // 1:1 prescale value
#define T3_PS_1_2          0b00001000  // 1:2 prescale value
#define T3_PS_1_4          0b00010000  // 1:4 prescale value
#define T3_PS_1_8          0b00011000  // 1:8 prescale value
#define T3_PS_MASK         (~T1_PS_1_8)

#define T3_OSC1EN_OFF      0b00000000  // Timer 1 oscilator enable off
#define T3_OSC1EN_ON       0b00000100  // Timer 1 oscilator enable on
#define       T3_OSC_MASK                (~T1_OSC1EN_ON)

#define T3_SYNC_EXT_ON       0b00000000  // Synchronize external clock input
#define T3_SYNC_EXT_OFF      0b00000010  // Do not synchronize external clock
input
#define T3_SYNC_MASK         (~T1_SYNC_EXT_OFF)

#define T3_8BIT_RW           0b00000000  // 8-bit mode
#define T3_16BIT_RW          0b00000001  // 16-bit mode
#define T3_BIT_RW_MASK       (~T1_16BIT_RW)

#else

#define T3_SOURCE_PINOSC   0b10011111  // Clock source T1OSCEN = 0 Ext clock,
T1OSCEN=1 Crystal osc
#define T3_SOURCE_CAPOSC   0b10111111  // Clock source is for Capacitive Sensing
Oscillator
#define T3_SOURCE_FOSC_4   0b11011111  //Clock source is instruction clock
(FOSC/4)
#define T3_SOURCE_FOSC     0b11111111  //Closck source is system clock (FOSC)

#define T3_PS_1_1          0b11100111  // 1:1 prescale value
#define T3_PS_1_2          0b11101111  // 1:2 prescale value
#define T3_PS_1_4          0b11110111  // 1:4 prescale value
#define T3_PS_1_8          0b11111111  // 1:8 prescale value

#define T3_OSC1EN_OFF      0b11111011  // Timer 1 oscilator enable off
#define T3_OSC1EN_ON       0b11111111  // Timer 1 oscilator enable on

#define T3_SYNC_EXT_ON       0b11111101  // Synchronize external clock input
#define T3_SYNC_EXT_OFF      0b11111111  // Do not synchronize external clock
input

#define T3_8BIT_RW           0b11111110  // 8-bit mode
#define T3_16BIT_RW          0b11111111  // 16-bit mode
#define T3_BIT_RW_MASK       (~T1_16BIT_RW)
#endif

void OpenTimer3 (PARAM_SCLASS unsigned char config, PARAM_SCLASS unsigned char
config1);
void CloseTimer3 (void);
unsigned int ReadTimer3 (void);
void WriteTimer3 (PARAM_SCLASS unsigned int timer3);
#endif
```

```
/* ***** TIMER4 ***** */

#if defined(TMR_V4) || defined(TMR_V6)

/* TIMER4 configuration masks -- to be 'anded' together and passed to the
 * 'open' routine. */
//-----------------AND OR MASK
#ifdef USE_OR_MASKS
#define T4_POST_1_1    0b00000000  // Postscaler 1:1
#define T4_POST_1_2    0b00001000  // Postscaler 1:2
#define T4_POST_1_3    0b00010000  // Postscaler 1:3
#define T4_POST_1_4    0b00011000  // Postscaler 1:4
#define T4_POST_1_5    0b00100000  // Postscaler 1:5
#define T4_POST_1_6    0b00101000  // Postscaler 1:6
#define T4_POST_1_7    0b00110000  // Postscaler 1:7
#define T4_POST_1_8    0b00111000  // Postscaler 1:8
#define T4_POST_1_9    0b01000000  // Postscaler 1:9
#define T4_POST_1_10   0b01001000  // Postscaler 1:10
#define T4_POST_1_11   0b01010000  // Postscaler 1:11
#define T4_POST_1_12   0b01011000  // Postscaler 1:12
#define T4_POST_1_13   0b01100000  // Postscaler 1:13
#define T4_POST_1_14   0b01101000  // Postscaler 1:14
#define T4_POST_1_15   0b01110000  // Postscaler 1:15
#define T4_POST_1_16   0b01111000  // Postscaler 1:16
#define T4_POST_MASK   (~T4_POST_1_16)

#define T4_PS_1_1      0b00000000  // Prescale 1:1
#define T4_PS_1_4      0b00000001  // Prescale 1:4
#define T4_PS_1_16     0b00000011  // Prescale 1:16
#define T4_PS_MASK     (~T4_PS_1_16)
//----------------AND MASK--------------------------
#else //!USE_OR_MASKS
#define T4_POST_1_1    0b10000111  // Postscaler 1:1
#define T4_POST_1_2    0b10001111  // Postscaler 1:2
#define T4_POST_1_3    0b10010111  // Postscaler 1:3
#define T4_POST_1_4    0b10011111  // Postscaler 1:4
#define T4_POST_1_5    0b10100111  // Postscaler 1:5
#define T4_POST_1_6    0b10101111  // Postscaler 1:6
#define T4_POST_1_7    0b10110111  // Postscaler 1:7
#define T4_POST_1_8    0b10111111  // Postscaler 1:8
#define T4_POST_1_9    0b11000111  // Postscaler 1:9
#define T4_POST_1_10   0b11001111  // Postscaler 1:10
#define T4_POST_1_11   0b11010111  // Postscaler 1:11
#define T4_POST_1_12   0b11011111  // Postscaler 1:12
#define T4_POST_1_13   0b11100111  // Postscaler 1:13
#define T4_POST_1_14   0b11101111  // Postscaler 1:14
#define T4_POST_1_15   0b11110111  // Postscaler 1:15
#define T4_POST_1_16   0b11111111  // Postscaler 1:16
#define T4_PS_1_1      0b11111100  // Prescale 1:1
#define T4_PS_1_4      0b11111101  // Prescale 1:4
#define T4_PS_1_16     0b11111111  // Prescale 1:16

#endif //USE_OR_MASKS

#define WriteTimer4(timer4) TMR4 = (timer4)
#define ReadTimer4()          TMR4
void OpenTimer4 (PARAM_SCLASS unsigned char config);
void CloseTimer4 (void);
//unsigned char ReadTimer4 (void);

#endif

//-------- TIMER 5 DECLARATION --------------------
#if defined (TMR_V5)
//--------------AND OR MASK-----------------------------------
#ifdef USE_OR_MASKS
```

```
#define T5_SLP_EN          0b10000000   // Enable during sleep
#define T5_SLP_DIS         0b00000000   // Disable during sleep
#define T5_SLP_MASK        (~T5_SLP_EN)

#define T5_SP_EVNT_REN     0b00000000   // special event reset enable
#define T5_SP_EVNT_RDIS    0b01000000   // special event reset disable
#define T5_SP_ENNT_MASK    (~T5_SP_EVNT_RDIS)

#define T5_MD_SNGL_SHOT    0b00100000   // SINGLE SHOT MODE
#define T5_MD_CONT_COUNT   0b00000000   // CONTINOUS COUNT MODE
#define T5_MD_MASK         (~T5_MD_SNGL_SHOT)

#define T5_PS_1_1          0b00000000   // Prescale 1:1
#define T5_PS_1_2          0b00001000   // Prescale 1:2
#define T5_PS_1_4          0b00010000   // Prescale 1:4
#define T5_PS_1_8          0b00011000   // Prescale 1:8
#define T5_PS_MASK         (~T5_PS_1_8)

#define T5_EX_CLK_SYNC       0b00000000   // EXT. CLOCK SYNCHRONIZATION
#define T5_EX_CLK_NOSYNC     0b00000100   // DO NOT synchronize
#define T5_EX_CLK_SYNC_MASK (~T5_EX_CLK_NOSYNC)

#define T5_CLK_EXTRN         0b00000010   // Clock source external
#define T5_CLK_INT           0b00000000   // Clock source internal
#define T5_CLK_SOURCE_MASK   (~T5_CLK_EXTRN)
//----------------AND MASK-------------------------------------
#else //!USE_OR_MASKS
#define T5_SLP_EN          0b11111111   // Enable during sleep
#define T5_SLP_DIS         0b01111111   // Disable during sleep
#define T5_SP_EVNT_REN     0b10111111   // special event reset enable
#define T5_SP_EVNT_RDIS    0b11111111   // special event reset disable
#define T5_MD_SNGL_SHOT    0b11111111   // SINGLE SHOT MODE
#define T5_MD_CONT_COUNT   0b11011111   // CONTINOUS COUNT MODE
#define T5_PS_1_1          0b11100111   // Prescale 1:1
#define T5_PS_1_2          0b11101111   // Prescale 1:2
#define T5_PS_1_4          0b11110111   // Prescale 1:4
#define T5_PS_1_8          0b11111111   // Prescale 1:8
#define T5_EX_CLK_SYNC     0b11111011   // EXT. CLOCK SYNCHRONIZATION
#define T5_EX_CLK_NOSYNC   0b11111111   // DO NOT synchronize
#define T5_CLK_EXTRN       0b11111111   // Clock source external
#define T5_CLK_INT         0b11111101   // Clock source internal
#endif //USE_OR_MASKS

void CloseTimer5(void);
void OpenTimer5(PARAM_SCLASS unsigned char, PARAM_SCLASS unsigned int);
unsigned int ReadTimer5(void);
void WriteTimer5(PARAM_SCLASS unsigned int);

#endif

#if defined (TMR_V6)
#ifdef USE_OR_MASKS
#define TIMER_GATE_ON           0b10000000  // Timer counting is controlled by
Timer1 gate function
#define TIMER_GATE_OFF          0b00000000  // Timer is always counting
#define TIMER_GATE_MASK         (~T1_GATE_ON)

#define TIMER_GATE_POL_HI        0b01000000  //Gate  is active-high
#define TIMER_GATE_POL_LO        0b00000000  //Gate is active-low
#define TIMER_GATE_POL_MASK      (~TIMER_GATE_POL_HI)

#define TIMER_GATE_TOGGLE_ON     0b00100000  // Gate Toggle mode is enabled
#define TIMER_GATE_TOGGLE_OFF    0b00000000  // Gate Toggle mode is disabled
#define TIMER_GATE_TOGGLE_MASK   (~TIMER_GATE_TOGGLE_ON)

#define TIMER_GATE_1SHOT_ON     0b00010000  //Gate one shot is enabled
#define TIMER_GATE_1SHOT_OFF    0b00000000  //Gate one shot is disabled
```

```
#define TIMER_GATE_1SHOT_MASK   (~TIMER_GATE_1SHOT_MASK)

#define TIMER_GATE_SRC_T1GPIN   0b00000000  //Timer1 gate pin
#define TIMER_GATE_SRC_T0       0b00000001  //Timer0 overflow output
#define TIMER_GATE_SRC_T2       0b00000010  //Timer2 match PR2  output
#define TIMER_GATE_SRC_MASK     (~TIMER_GATE_SRC_T2)

#define TIMER_GATE_INT_OFF      0b00000000  // Interrupts disabled
#define TIMER_GATE_INT_ON       0b00000100  // Interrupts enabled
//#define TIMER_INT_MASK            (~TIMER_GATE_INT_ON)
#else
#define TIMER_GATE_ON           0b11111111  // Timer counting is controlled by
Timer1 gate function
#define TIMER_GATE_OFF          0b01111111  // Timer is always counting
#define TIMER_GATE_POL_HI       0b11111111  //Gate  is active-high
#define TIMER_GATE_POL_LO       0b10111111  //Gate is active-low
#define TIMER_GATE_TOGGLE_ON    0b11111111  // Gate Toggle mode is enabled
#define TIMER_GATE_TOGGLE_OFF   0b11011111  // Gate Toggle mode is disabled
#define TIMER_GATE_1SHOT_ON     0b11111111  //Gate one shot is enabled
#define TIMER_GATE_1SHOT_OFF    0b11101111  //Gate one shot is disabled
#define TIMER_GATE_SRC_T1GPIN   0b11111100  //Timer1 gate pin
#define TIMER_GATE_SRC_T0       0b11111101  //Timer0 overflow output
#define TIMER_GATE_SRC_T2       0b11111110  //Timer2 match PR2  output
#define TIMER_GATE_INT_OFF      0b11111011  // Interrupts disabled
#define TIMER_GATE_INT_ON       0b11111111  // Interrupts enabled
#endif
#endif

#if defined (TMR_V4)
#ifdef USE_OR_MASKS
#define T34_SOURCE_CCP          0b01001000          // T3 and T4 are sources for
CCP1 thru CCP5
#define T12_CCP12_T34_CCP345    0b01000000          // T1 and T2 are sources for
CCP1 and CCP2 and T3 and T4 are sources for CCP3 thru CCP5
#define T12_CCP1_T34_CCP2345    0b00001000          // T1 and T2 are sources for
CCP1 and T3 and T4 are sources for CCP2 thru CCP5
#define T12_SOURCE_CCP          0b00000000          // T1 and T2 are sources for
CCP1 thru CCP5
#define TMR_SOURCE_CCP_MASK     (~T34_SOURCE_CCP)
#else  //!USE_OR_MASKS
#define T34_SOURCE_CCP          0b11111111          // T3 and T4 are sources for
CCP1 thru CCP5
#define T12_CCP12_T34_CCP345    0b11110111          // T1 and T2 are sources for
CCP1 and CCP2 and T3 and T4 are sources for CCP3 thru CCP5
#define T12_CCP1_T34_CCP2345    0b10111111          // T1 and T2 are sources for
CCP1 and T3 and T4 are sources for CCP2 thru CCP5
#define T12_SOURCE_CCP          0b10110111          // T1 and T2 are sources for
CCP1 thru CCP5
#endif //USE_OR_MASKS

#elif defined (TMR_V2)//USE_OR_MASKS
//---------------AND OR MASK-----------------------------------------------
#ifdef USE_OR_MASKS
#define T3_SOURCE_CCP           0b01001000              // T3 is source for
CCP
#define T1_CCP1_T3_CCP2         0b00001000              // T1 is source for
CCP1 and T3 is source for CCP2
#define T1_SOURCE_CCP           0b00000000              // T1 is source for
CCP
#define TMR_SOURCE_CCP_MASK     (~T3_SOURCE_CCP)
//---------------AND MASK-------------------------------------------------
#else //!USE_OR_MASKS
#define T3_SOURCE_CCP           0b11111111              // T3 is source for
CCP
#define T1_CCP1_T3_CCP2         0b10111111              // T1 is source for
CCP1 and T3 is source for CCP2
```

```
#define T1_SOURCE_CCP           0b10110111                // T1 is source for
CCP
#endif //USE_OR_MASKS

#elif defined (TMR_V6)
#ifdef USE_OR_MASKS
#define T34_SOURCE_CCP12        0b00000011          // T3 and T4 are sources for
CCP1 and CCP2
#define T12_CCP1_T34_CCP2       0b00000001          // T1 and T2 are sources for
CCP1 and T3 and T4 are sources for CCP2
#define T12_SOURCE_CCP           0b00000000         // T1  and T2 is source for
CCP1 and CCP2
#define TMR_SOURCE_CCP_MASK      (T34_CCP12)
#else
#define T34_SOURCE_CCP12         0b11111111         // T3 and T4 are sources for
CCP1 and CCP2
#define T12_CCP1_T34_CCP2        0b11111101         // T1 and T2 are sources for
CCP1 and T3 and T4 are sources for CCP2
#define T12_SOURCE_CCP           0b11111100         // T1  and T2 is source for
CCP1 and CCP2
#endif
#endif

#if defined (TMR_V4) || defined (TMR_V2) || defined (TMR_V6)
void SetTmrCCPSrc( PARAM_SCLASS unsigned char );
#endif

#endif//__TIMERS_H
```

6.1.6. timer0.c

```
/*********************************************************************************
 FileName:      Timer0.c
 Dependencies:  See INCLUDES section
 Processor:     PIC18 Microcontroller
 Compiler:      Microchip C18 (for PIC18)
 Overview:      This file contains source code and function prototypes to enable
                , disable and configure Timer0 on PIC18 MCUs

*********************************************************************************
 File Description:

 Change History:
  Rev   Date         Author                   Description
  1.0   15/10/2009   Óscar Aragón Andreu      Initial release

*********************************************************************************/

/** INCLUDES *******************************************************************/

#include "Compiler.h"
#include "timer/timers.h"

/** PROTOTYPES *****************************************************************/

void OpenTimer0(unsigned char config);
void CloseTimer0(void);

/*********************************************************************************
*    Function Name:  OpenTimer0                                                  *
*    Return Value:   void                                                        *
*    Parameters:     config: bit definitions to configure Timer0                 *
*    Description:    This routine resets the Timer0 regs to the POR state and    *
*                    configures the interrupt, clock source, edge and prescaler.*
*    Notes:          The bit definitions for config can be found  in the         *
*                    timers.h file.                                              *
*                                                                                *
*********************************************************************************/
void OpenTimer0(unsigned char config)
{
  T0CON = (0x7f & config);// Configure timer, but don't start it yet
  TMR0H = 0;              // Reset Timer0 to 0x0000
  TMR0L = 0;
  INTCONbits.T0IF = 0;    // Clear Timer0 overflow flag

  if(config&0x80)         // If interrupts enabled
    INTCONbits.T0IE = 1;  // Enable Timer0 overflow interrupt
  else
    INTCONbits.T0IE = 0;

  T0CONbits.TMR0ON = 1;   // Start Timer
}


/*********************************************************************************
*    Function Name:  CloseTimer0                                                 *
*    Return Value:   void                                                        *
*    Parameters:     void                                                        *
*    Description:    This routine disables the Timer0 interrupt.                 *
*                                                                                *
*********************************************************************************/
void CloseTimer0(void)
{
  T0CONbits.TMR0ON = 0;   // Disable Timer1
  INTCONbits.TMR0IE = 0;  // Disable Timer1 overflow interrupts
}
```

6.1.7. spi.h

```
/*********************************************************************
*
*                 PIC18 SPI peripheral library header
*
*********************************************************************
* FileName:        spi.h
* Dependencies:    None
* Processor:       PIC18
* Compiler:        Microchip C18
* Company:         Microchip Technology, Inc.
*********************************************************************/

#ifndef __SPI_H
#define __SPI_H

/* SSPSTAT REGISTER */

// Master SPI mode only

#define   SMPEND        0x80            // Input data sample at end of data out
#define   SMPMID        0x00            // Input data sample at middle of data out

#define   MODE_00       0               // Setting for SPI bus Mode 0,0
//CKE           0x40                    // SSPSTAT register
//CKP           0x00                    // SSPCON1 register

#define   MODE_01       1               // Setting for SPI bus Mode 0,1
//CKE           0x00                    // SSPSTAT register
//CKP           0x00                    // SSPCON1 register

#define   MODE_10       2               // Setting for SPI bus Mode 1,0
//CKE           0x40                    // SSPSTAT register
//CKP           0x10                    // SSPCON1 register

#define   MODE_11       3               // Setting for SPI bus Mode 1,1
//CKE           0x00                    // SSPSTAT register
//CKP           0x10                    // SSPCON1 register

/* SSPCON1 REGISTER */
#define   SSPENB        0x20            // Enable serial port and configures SCK,
SDO, SDI

#define   SPI_FOSC_4    0               // SPI Master mode, clock = Fosc/4
#define   SPI_FOSC_16   1               // SPI Master mode, clock = Fosc/16
#define   SPI_FOSC_64   2               // SPI Master mode, clock = Fosc/64
#define   SPI_FOSC_TMR2 3               // SPI Master mode, clock = TMR2 output/2
#define   SLV_SSON      4               // SPI Slave mode, /SS pin control enabled
#define   SLV_SSOFF     5               // SPI Slave mode, /SS pin control
disabled


/*  25Cxxx EEPROM instruction set */
#define   SPI_WREN      6               // write enable latch
#define   SPI_WRDI      4               // reset the write enable latch
#define   SPI_RDSR      5               // read status register
#define   SPI_WRSR      1               // write status register
#define   SPI_READ      3               // read data from memory
#define   SPI_WRITE     2               // write data to memory

/*  Bits within status register of 25Cxxx */
#define   WIP           0               // write in progress status
#define   WEL           1               // write enable latch status
#define   BP0           2               // block protection bit status
#define   BP1           3               // block protection bit status
```

```
/* FUNCTION PROTOTYPES */

#define PARAM_SCLASS auto


/* These devices have two SPI modules */
#if defined(__18F64J15) || defined(__18F65J10) || defined(__18F65J15) || \
    defined(__18F66J10) || defined(__18F66J15) || defined(__18F67J10) || \
    defined(__18F84J15) || defined(__18F85J10) || defined(__18F85J15) || \
    defined(__18F86J10) || defined(__18F86J15) || defined(__18F87J10) || \
    defined(__18F6527)  || defined(__18F6622)  || \
    defined(__18F6627)  || defined(__18F6722)  || \
    defined(__18F8527)  || defined(__18F8622)  || \
    defined(__18F8627)  || defined(__18F8722)  || \
    defined(__18F44J10) || defined(__18F45J10)

/* ***** SPI1 ***** */

/* CloseSPI1
 * Disable SPI1 module
 */
#define  CloseSPI1()      (SSP1CON1 &=0xDF)
#define  CloseSPI CloseSPI1

/* DataRdySPI1
 * Test if SSP1BUF register is full
 */
#define  DataRdySPI1()    (SSP1STATbits.BF)
#define DataRdySPI DataRdySPI1

/* ReadSPI1
 * Read byte from SSP1BUF register
 */
unsigned char ReadSPI1( void );
#define ReadSPI ReadSPI1

/* getcSPI1
 * Read byte from SSP1BUF register
 */
#define  getcSPI1  ReadSPI1
#define getcSPI getcSPI1

/* OpenSPI1
 */
void OpenSPI1( PARAM_SCLASS unsigned char sync_mode,
               PARAM_SCLASS unsigned char bus_mode,
               PARAM_SCLASS unsigned char smp_phase );
#define OpenSPI OpenSPI1

/* WriteSPI1
 * Write byte to SSP1BUF register
 */
unsigned char WriteSPI1( PARAM_SCLASS unsigned char data_out );
#define WriteSPI WriteSPI1

/* putcSPI1
 * Write byte to SSP1BUF register
 */
#define  putcSPI1  WriteSPI1
#define  putcSPI putcSPI1

/* getsSPI1
 * Write string to SSP1BUF
 */
void getsSPI1( PARAM_SCLASS unsigned char *rdptr, PARAM_SCLASS unsigned char
length );
```

```
#define getsSPI getsSPI1

/* putsSPI1
 * Read string from SSP1BUF
 */
void putsSPI1( PARAM_SCLASS unsigned char *wrptr );
#define putsSPI putsSPI1

/* ***** SPI2 ***** */

/* CloseSPI2
 * Disable SPI2 module
 */
#define  CloseSPI2()      (SSP2CON1 &=0xDF)


/* DataRdySPI2
 * Test if SSP2BUF register is full
 */
#define  DataRdySPI2()    (SSP2STATbits.BF)

/* ReadSPI2
 * Read byte from SSP2BUF register
 */
unsigned char ReadSPI2( void );

/* getcSPI2
 * Read byte from SSP2BUF register
 */
#define  getcSPI2  ReadSPI2

/* OpenSPI2
 */
void OpenSPI2( PARAM_SCLASS unsigned char sync_mode,
               PARAM_SCLASS unsigned char bus_mode,
               PARAM_SCLASS unsigned char smp_phase );

/* WriteSPI2
 * Write byte to SSP2BUF register
 */
unsigned char WriteSPI2( PARAM_SCLASS unsigned char data_out );

/* putcSPI2
 * Write byte to SSP2BUF register
 */
#define  putcSPI2  WriteSPI2

/* getsSPI2
 * Write string to SSP2BUF
 */
void getsSPI2( PARAM_SCLASS unsigned char *rdptr, PARAM_SCLASS unsigned char
length );

/* putsSPI2
 * Read string from SSP2BUF
 */
void putsSPI2( PARAM_SCLASS unsigned char *wrptr );

#else

/* ***** SPI ***** */

/* CloseSPI
 * Disable SPI module
 */
#define  CloseSPI()      (SSPCON1 &=0xDF)
```

```
/* DataRdySPI
 * Test if SSPBUF register is full
 */
#define  DataRdySPI()    (SSPSTATbits.BF)

/* ReadSPI
 * Read byte from SSPBUF register
 */
unsigned char ReadSPI( void );

/* getcSPI
 * Read byte from SSPBUF register
 */
#define  getcSPI  ReadSPI

/* OpenSPI
 */
void OpenSPI( PARAM_SCLASS unsigned char sync_mode,
              PARAM_SCLASS unsigned char bus_mode,
              PARAM_SCLASS unsigned char smp_phase );

/* WriteSPI
 * Write byte to SSPBUF register
 */
unsigned char WriteSPI( PARAM_SCLASS unsigned char data_out );

/* putcSPI
 * Write byte to SSPBUF register
 */
#define  putcSPI  WriteSPI

/* getsSPI
 * Write string to SSPBUF
 */
void getsSPI( PARAM_SCLASS unsigned char *rdptr, PARAM_SCLASS unsigned char
length );


/* putsSPI
 * Read string from SSPBUF
 */
void putsSPI( PARAM_SCLASS unsigned char *wrptr );

#endif

#endif  /* __SPI_H */
```

6.1.8. spi.c

```
/*********************************************************************************
 FileName:      spi.c
 Dependencies:  See INCLUDES section
 Processor:     PIC18 USB Microcontroller
 Compiler:      Microchip C18 (for PIC18)
 Overview:      This file contains function prototypes to manage the behaviour
                of SPI Peripheral

*********************************************************************************
 File Description:

 Change History:
  Rev   Date         Author                  Description
  1.0   15/10/2009   Óscar Aragón Andreu      Initial release

*********************************************************************************/

/** INCLUDES *******************************************************************/

#include "GenericTypeDefs.h"
#include "Compiler.h"
#include "HardwareProfile.h"
#include "RFidReader.h"
#include "/spi/spi.h"

/** VARIABLES ******************************************************************/

extern BYTE error;
extern BYTE error_counter;

/** PROTOTYPES *****************************************************************/

/*********************************************************************
*   Function Name:  OpenSPI                                          *
*   Return Value:   void                                             *
*   Parameters:     SSP peripheral setup bytes                       *
*   Description:    This function sets up the SSP module on a        *
*                   PIC18Cxxx device for use with a Microchip SPI    *
*                   EEPROM device or SPI bus device.                 *
*********************************************************************/
void OpenSPI( unsigned char sync_mode, unsigned char bus_mode, unsigned char
smp_phase)
{
  SSPSTAT &= 0x3F;                // power on state
  SSPCON1 = 0x00;                 // power on state
  SSPCON1 |= sync_mode;           // select serial mode
  SSPSTAT |= smp_phase;           // select data input sample phase

  switch( bus_mode )
  {
    case 0:                       // SPI bus mode 0,0
      SSPSTATbits.CKE = 1;        // data transmitted on rising edge
      break;
    case 2:                       // SPI bus mode 1,0
      SSPSTATbits.CKE = 1;        // data transmitted on falling edge
      SSPCON1bits.CKP = 1;        // clock idle state high
      break;
    case 3:                       // SPI bus mode 1,1
      SSPCON1bits.CKP = 1;        // clock idle state high
      break;
    default:                      // default SPI bus mode 0,1
      break;
  }

  switch( sync_mode )
```

```
  {
    case 4:                       // slave mode w /SS enable
#if defined(__18F6310)  || defined(__18F6390)  || \
    defined(__18F6410)  || defined(__18F6490)  || \
    defined(__18F8310)  || defined(__18F8390)  || \
    defined(__18F8410)  || defined(__18F8490)  || \
    defined(__18F6527)  || defined(__18F6622)  || \
    defined(__18F6627)  || defined(__18F6722)  || \
    defined(__18F8527)  || defined(__18F8622)  || \
    defined(__18F8627)  || defined(__18F8722)  || \
    defined(__18F65J10) || defined(__18F65J15) || \
    defined(__18F66J10) || defined(__18F66J15) || \
    defined(__18F67J10) || defined(__18F85J10) || \
    defined(__18F85J15) || defined(__18F86J10) || \
    defined(__18F86J15) || defined(__18F87J10)
      TRISFbits.TRISF7 = 1;       // define /SS pin as input
#else
      TRISAbits.TRISA5 = 1;       // define /SS pin as input
#endif
    case 5:                       // slave mode w/o /SS enable
#if defined(__18F2455) || defined(__18F2550) || \
    defined(__18F4455) || defined(__18F4550)
      TRISBbits.TRISB1 = 1;       // define clock pin as input
#else
      TRISCbits.TRISC3 = 1;       // define clock pin as input
#endif
      SSPSTATbits.SMP = 0;        // must be cleared in slave SPI mode
      break;
    default:                      // master mode, define clock pin as output
#if defined(__18F2455) || defined(__18F2550) || \
    defined(__18F4455) || defined(__18F4550)
      TRISBbits.TRISB1 = 0;       // define clock pin as output
#else
      TRISCbits.TRISC3 = 0;       // define clock pin as output
#endif
      break;
  }

#if defined(__18F2455) || defined(__18F2550) || \
    defined(__18F4455) || defined(__18F4550)
  TRISC &= 0x7F;                  // define SDO as output (master or slave)
  TRISB |= 0x01;                  // define SDI as input (master or slave)
#else
  TRISC &= 0xDF;                  // define SDO as output (master or slave)
  TRISC |= 0x10;                  // define SDI as input (master or slave)
#endif
  SSPCON1 |= SSPENB;              // enable synchronous serial port
}


/*********************************************************************
*     Function Name:    CloseSPI                                     *
*     Return Value:     void                                         *
*     Parameters:       void                                         *
*     Description:      This function disables the SSP module. Pin   *
*                       I/O returns under the control of the port    *
*                       registers.                                   *
*********************************************************************/
#undef CloseSPI
void CloseSPI( void )
{
  SSPCON1 &= 0xDF;                // disable synchronous serial port
}


/*********************************************************************
*     Function Name:    ReadSPI                                      *
```

```
*     Return Value:     contents of SSPBUF register                 *
*     Parameters:       void                                        *
*     Description:      Read single byte from SPI bus.              *
*****************************************************************/
unsigned char ReadSPI( void )
{
  SSPBUF = 0x00;                  // initiate bus cycle
  while ( !SSPSTATbits.BF );      // wait until cycle complete
  return ( SSPBUF );              // return with byte read
}


/*****************************************************************
*     Function Name:    WriteSPI                                    *
*     Return Value:     Status byte for WCOL detection.             *
*     Parameters:       Single data byte for SPI bus.               *
*     Description:      This routine writes a single byte to the    *
*                       SPI bus.                                    *
*****************************************************************/
unsigned char WriteSPI( unsigned char data_out )
{
  SSPBUF = data_out;              // write byte to SSPBUF register
  if ( SSPCON1 & 0x80 )           // test if write collision occurred
       {
        error = 0x02;             // Error occurred: 0x02, SPI Write Collision
        ERROR_LED = 1;            // Set Error Led
        error_counter++;          // Count error (To improve in future version)
        return(1);                // If error occurred exit immediately
       }
  else
  {
    while( !SSPSTATbits.BF );     // wait until bus cycle complete
  }
  return(0);                      // Return without error
}


/*****************************************************************
*     Function Name:    getsSPI                                     *
*     Return Value:     void                                        *
*     Parameters:       address of read string storage location and *
*                       length of string bytes to read              *
*     Description:      This routine reads a string from the SPI    *
*                       bus.  The number of bytes to read is deter- *
*                       mined by parameter 'length'.                *
*****************************************************************/
void getsSPI( unsigned char *rdptr, unsigned char length )
{
  while ( length )                // stay in loop until length = 0
  {
    *rdptr++ = getcSPI();         // read a single byte
    length--;                     // reduce string length count by 1
  }
}


/*****************************************************************
*     Function Name:    putsSPI                                     *
*     Return Value:     void                                        *
*     Parameters:       address of write string storage location    *
*     Description:      This routine writes a string to the SPI bus.*
*****************************************************************/
void putsSPI( unsigned char *wrptr )
{
  while ( *wrptr )                // test for string null character
  {
     SSPBUF = *wrptr++;           // initiate SPI bus cycle
```

```
      while( !SSPSTATbits.BF );    // wait until 'BF' bit is set
  }
}
```

6.1.9. usb.h

```
/********************************************************************************
 FileName:     usb.h
 Dependencies: See INCLUDES section
 Processor:    PIC18, PIC24, & PIC32 USB Microcontrollers
 Hardware:
 Complier:     Microchip C18 (for PIC18), C30 (for PIC24), or C32 (for PIC32)
 Company:      Microchip Technology, Inc.

 Software License Agreement:

 The software supplied herewith by Microchip Technology Incorporated
 (the “Company”) for its PICmicro® Microcontroller is intended and
 supplied to you, the Company’s customer, for use solely and
 exclusively on Microchip PICmicro Microcontroller products. The
 software is owned by the Company and/or its supplier, and is
 protected under applicable copyright laws. All rights are reserved.
 Any use in violation of the foregoing restrictions may subject the
 user to criminal sanctions under applicable laws, as well as to
 civil liability for the breach of the terms and conditions of this
 license.

 THIS SOFTWARE IS PROVIDED IN AN “AS IS” CONDITION. NO WARRANTIES,
 WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
 TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
 IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
 CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

********************************************************************

 Change History:
 Date         Description
 02/20/2008   Initial version
*******************************************************************/

//DOM-IGNORE-BEGIN
/*******************************************************************
 * The following lines are used by VDI.
 * GUID=E537A0C0-6FEE-4afd-89B9-0C35BF72A80B
 * GUIInterfaceVersion=1.00
 * LibraryVersion=2.4
 *******************************************************************/
//DOM-IGNORE-END
/********************************************************************************

    USB Header File

Summary:
    This file aggregates all necessary header files for the Microchip USB Host,
    Device, and OTG libraries.  It provides a single-file can be included in
    application code.  The USB libraries simplify the implementation of USB
    applications by providing an abstraction of the USB module and its registers
    and bits such that the source code for the can be the same across various
    hardware platforms.

Description:
    This file aggregates all necessary header files for the Microchip USB Host,
    Device, and OTG libraries.  It provides a single-file can be included in
    application code.  The USB libraries simplify the implementation of USB
    applications by providing an abstraction of the USB module and its registers
    and bits such that the source code for the can be the same across various
    hardware platforms.

    Note that this file does not include the header files for any client or
    function drivers.
```

```
    This file is located in the "\<Install Directory\>\\Microchip\\Include\\USB"
    directory.

    When including this file in a new project, this file can either be
    referenced from the directory in which it was installed or copied
    directly into the user application folder. If the first method is
    chosen to keep the file located in the folder in which it is installed
    then include paths need to be added so that the library and the
    application both know where to reference each others files. If the
    application folder is located in the same folder as the Microchip
    folder (like the current demo folders), then the following include
    paths need to be added to the application's project:

    ..\\Include

    ..\\..\\Include

    ..\\..\\MicrochipInclude

    ..\\..\\\<Application Folder\>

    ..\\..\\..\\\<Application Folder\>

    If a different directory structure is used, modify the paths as
    required. An example using absolute paths instead of relative paths
    would be the following:

    C:\\Microchip Solutions\\Microchip\\Include

    C:\\Microchip Solutions\\My Demo Application

*******************************************************************************/

//DOM-IGNORE-BEGIN
#ifndef _USB_H_
#define _USB_H_
//DOM-IGNORE-END


// *****************************************************************************
// *****************************************************************************
// Section: All necessary USB Library headers
// *****************************************************************************
// *****************************************************************************


#include "usb/usb_config.h"             // Must be defined by the application
#include "usb/usb_common.h"             // Common USB library definitions
#include "usb/usb_ch9.h"                // USB device framework definitions
#include "usb/usb_hal.h"                // Hardware Abstraction Layer interface
#include "usb/usb_function_generic.h"   // Generic Write and Read Functions


#if defined( USB_SUPPORT_DEVICE )
    #include "usb/usb_device.h"     // USB Device abstraction layer interface
#endif

#if defined( USB_SUPPORT_HOST )
    #include "usb/usb_host.h"       // USB Host abstraction layer interface
#endif

#if defined ( USB_SUPPORT_OTG )
    #include "usb/usb_otg.h"
#endif
```

```
// ****************************************************************************
// ****************************************************************************
// Section: Host Firmware Version
// ****************************************************************************
// ****************************************************************************

#define USB_MAJOR_VER   1       // Firmware version, major release number.
#define USB_MINOR_VER   0       // Firmware version, minor release number.
#define USB_DOT_VER     0       // Firmware version, dot release number.


#endif // _USB_H_
/*******************************************************************************
 * EOF
 */
```

6.1.10. usb_config.h

```
/********************************************************************
 FileName:          usb_config.h
 Dependencies:      Always: GenericTypeDefs.h, usb_device.h
                    Situational: usb_function_hid.h, usb_function_cdc.h,
usb_function_msd.h, etc.
 Processor:         PIC18 or PIC24 USB Microcontrollers
 Hardware:          The code is natively intended to be used on the following
                          hardware platforms: PICDEM™ FS USB Demo Board,
                          PIC18F87J50 FS USB Plug-In Module, or
                          Explorer 16 + PIC24 USB PIM.  The firmware may be
                          modified for use on other USB platforms by editing the
                          HardwareProfile.h file.
 Complier:   Microchip C18 (for PIC18) or C30 (for PIC24)
 Company:           Microchip Technology, Inc.

 Software License Agreement:

 The software supplied herewith by Microchip Technology Incorporated
 (the “Company”) for its PIC® Microcontroller is intended and
 supplied to you, the Company’s customer, for use solely and
 exclusively on Microchip PIC Microcontroller products. The
 software is owned by the Company and/or its supplier, and is
 protected under applicable copyright laws. All rights are reserved.
 Any use in violation of the foregoing restrictions may subject the
 user to criminal sanctions under applicable laws, as well as to
 civil liability for the breach of the terms and conditions of this
 license.

 THIS SOFTWARE IS PROVIDED IN AN “AS IS” CONDITION. NO WARRANTIES,
 WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
 TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
 IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
 CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

********************************************************************
 File Description:

 Change History:
  Rev   Date         Description
  1.0   11/19/2004   Initial release
  2.1   02/26/2007   Updated for simplicity and to use common
                     coding style
 ********************************************************************/

/********************************************************************
 * Descriptor specific type definitions are defined in: usbd.h
 *******************************************************************/

#ifndef USBCFG_H
#define USBCFG_H

/** DEFINITIONS ****************************************************/
#define USB_EP0_BUFF_SIZE 8  // Valid Options: 8, 16, 32, or 64 bytes.
                             // Using larger options take more SRAM, but
                             // does not provide much advantage in most types
                             // of applications.  Exceptions to this, are
                             // applications that use EP0 IN or OUT for sending
                             // large amounts of  application related data.

#define USB_MAX_NUM_INT   1  // For tracking Alternate Setting
#define USB_MAX_EP_NUMBER 1

//Device descriptor - if these two definitions are not defined then
//  a ROM USB_DEVICE_DESCRIPTOR variable by the exact name of device_dsc
```

```
//  must exist.
#define USB_USER_DEVICE_DESCRIPTOR &device_dsc
#define USB_USER_DEVICE_DESCRIPTOR_INCLUDE extern ROM USB_DEVICE_DESCRIPTOR
device_dsc

//Configuration descriptors - if these two definitions do not exist then
//  a ROM BYTE *ROM variable named exactly USB_CD_Ptr[] must exist.
#define USB_USER_CONFIG_DESCRIPTOR USB_CD_Ptr
#define USB_USER_CONFIG_DESCRIPTOR_INCLUDE extern ROM BYTE *ROM USB_CD_Ptr[]

//Make sure only one of the below "#define USB_PING_PONG_MODE"
//is uncommented.
#define USB_PING_PONG_MODE USB_PING_PONG__NO_PING_PONG
//#define USB_PING_PONG_MODE USB_PING_PONG__FULL_PING_PONG
//#define USB_PING_PONG_MODE USB_PING_PONG__EP0_OUT_ONLY
//#define USB_PING_PONG_MODE USB_PING_PONG__ALL_BUT_EP0          //NOTE: This
mode is not supported in PIC18F4550 family rev A3 devices


//#define USB_POLLING
#define USB_INTERRUPT

/* Parameter definitions are defined in usb_device.h */
#define USB_PULLUP_OPTION USB_PULLUP_ENABLE
//#define USB_PULLUP_OPTION USB_PULLUP_DISABLED

#define USB_TRANSCEIVER_OPTION USB_INTERNAL_TRANSCEIVER
//External Transceiver support is not available on all product families.  Please
//  refer to the product family datasheet for more information if this feature
//  is available on the target processor.
//#define USB_TRANSCEIVER_OPTION USB_EXTERNAL_TRANSCEIVER

//#define USB_SPEED_OPTION USB_FULL_SPEED
#define USB_SPEED_OPTION USB_LOW_SPEED //(not valid option for PIC24F devices)

#define USB_SUPPORT_DEVICE

#define USB_NUM_STRING_DESCRIPTORS 3

//#define USB_INTERRUPT_LEGACY_CALLBACKS
#define USB_ENABLE_ALL_HANDLERS
//#define USB_ENABLE_SUSPEND_HANDLER
//#define USB_ENABLE_WAKEUP_FROM_SUSPEND_HANDLER
//#define USB_ENABLE_SOF_HANDLER
//#define USB_ENABLE_ERROR_HANDLER
//#define USB_ENABLE_OTHER_REQUEST_HANDLER
//#define USB_ENABLE_SET_DESCRIPTOR_HANDLER
//#define USB_ENABLE_INIT_EP_HANDLER
//#define USB_ENABLE_EP0_DATA_HANDLER
//#define USB_ENABLE_TRANSFER_COMPLETE_HANDLER

/** DEVICE CLASS USAGE *********************************************/
#define USB_USE_GEN

/** ENDPOINTS ALLOCATION ***************************************/

/* Generic */
#define USBGEN_EP_SIZE          64
#define USBGEN_EP_NUM            1

/** DEFINITIONS ****************************************************/

#endif //USBCFG_H
```

6.1.11. usb_common.h

```
/********************************************************************************

* FileName:        usb_common.h
* Dependencies:    See included files, below.
* Processor:       PIC18/PIC24/PIC32MX microcontrollers with USB module
* Compiler:        C18 v3.13+/C30 v2.01+/C32 v0.00.18+
* Company:         Microchip Technology, Inc.

Software License Agreement

The software supplied herewith by Microchip Technology Incorporated
(the “Company”) for its PICmicro® Microcontroller is intended and
supplied to you, the Company’s customer, for use solely and
exclusively on Microchip PICmicro Microcontroller products. The
software is owned by the Company and/or its supplier, and is
protected under applicable copyright laws. All rights are reserved.
Any use in violation of the foregoing restrictions may subject the
user to criminal sanctions under applicable laws, as well as to
civil liability for the breach of the terms and conditions of this
license.

THIS SOFTWARE IS PROVIDED IN AN “AS IS” CONDITION. NO WARRANTIES,
WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

Author          Date       Comments
--------------------------------------------------------------------------------
BC/KO          15-Oct-2007 First release

Change History:

********************************************************************************/
//DOM-IGNORE-BEGIN
/********************************************************************************

  Common USB Library Definitions (Header File)

Summary:
    This file defines data types, constants, and macros that are common to
    multiple layers of the Microchip USB Firmware Stack.

Description:
    This file defines data types, constants, and macros that are common to
    multiple layers of the Microchip USB Firmware Stack.

    This file is located in the "\<Install Directory\>\\Microchip\\Include\\USB"
    directory.

    When including this file in a new project, this file can either be
    referenced from the directory in which it was installed or copied
    directly into the user application folder. If the first method is
    chosen to keep the file located in the folder in which it is installed
    then include paths need to be added so that the library and the
    application both know where to reference each others files. If the
    application folder is located in the same folder as the Microchip
    folder (like the current demo folders), then the following include
    paths need to be added to the application's project:

    ..\\Include

    ..\\..\\Include
```

```
    ..\\..\\MicrochipInclude

    ..\\..\\\<Application Folder\>

    ..\\..\\..\\\<Application Folder\>

    If a different directory structure is used, modify the paths as
    required. An example using absolute paths instead of relative paths
    would be the following:

    C:\\Microchip Solutions\\Microchip\\Include

    C:\\Microchip Solutions\\My Demo Application
*******************************************************************************/
//DOM-IGNORE-END

//DOM-IGNORE-BEGIN
#ifndef _USB_COMMON_H_
#define _USB_COMMON_H_
//DOM-IGNORE-END

#include <limits.h>
#include "GenericTypeDefs.h"
#include "usb/usb_config.h"


// *****************************************************************************
// *****************************************************************************
// Section: USB Constants
// *****************************************************************************
// *****************************************************************************

// Section: Error Code Values

#define USB_SUCCESS                             0x00    // USB operation
successful.
#define USB_INVALID_STATE                       0x01    // Operation cannot be
performed in current state.
#define USB_BUSY                                0x02    // A transaction is
already in progress.
#define USB_ILLEGAL_REQUEST                     0x03    // Cannot perform
requested operation.
#define USB_INVALID_CONFIGURATION               0x04    // Configuration
descriptor not found.
#define USB_MEMORY_ALLOCATION_ERROR             0x05    // Out of dynamic memory.
#define USB_UNKNOWN_DEVICE                      0x06    // Device with specified
address is not attached.
#define USB_CANNOT_ENUMERATE                    0x07    // Cannot enumerate the
attached device.
#define USB_EVENT_QUEUE_FULL                    0x08    // Event queue was full
when an event occured.
#define USB_ENDPOINT_BUSY                       0x10    // Endpoint is currently
processing a transaction.
#define USB_ENDPOINT_STALLED                    0x11    // Endpoint is currently
stalled. User must clear the condition.
#define USB_ENDPOINT_ERROR                      0x12    // Will need more than
this eventually
#define USB_ENDPOINT_ERROR_ILLEGAL_PID          0x13    // Illegal PID received.
#define USB_ENDPOINT_NOT_FOUND                  0x14    // Requested endpoint
does not exist on device.
#define USB_ENDPOINT_ILLEGAL_DIRECTION          0x15    // Reads must be performe
on IN endpoints, writes on OUT endpoints.
//#define USB_ENDPOINT_TRANSACTION_IN_PROGRESS    0x16
#define USB_ENDPOINT_NAK_TIMEOUT                0x17    // Too many NAK's
occurred while waiting for the current transaction.
#define USB_ENDPOINT_ILLEGAL_TYPE               0x18    // Transfer type must
match endpoint description.
```

```
#define USB_ENDPOINT_UNRESOLVED_STATE           0x19    // Endpoint is in an
unknown state after completing a transaction.
#define USB_ENDPOINT_ERROR_BIT_STUFF            0x20    // USB Module - Bit stuff
error.
#define USB_ENDPOINT_ERROR_DMA                  0x21    // USB Module - DMA
error.
#define USB_ENDPOINT_ERROR_TIMEOUT              0x22    // USB Module - Bus
timeout.
#define USB_ENDPOINT_ERROR_DATA_FIELD           0x23    // USB Module - Data
field size error.
#define USB_ENDPOINT_ERROR_CRC16                0x24    // USB Module - CRC16
failure.
#define USB_ENDPOINT_ERROR_END_OF_FRAME         0x25    // USB Module - End of
Frame error.
#define USB_ENDPOINT_ERROR_PID_CHECK            0x26    // USB Module - Illegal
PID received.
#define USB_ENDPOINT_ERROR_BMX                  0x27    // USB Module - Bus
Matrix error.
#define USB_ERROR_INSUFFICIENT_POWER            0x28    // Too much power was
requested

// Section: Return values for USBHostDeviceStatus()

#define USB_DEVICE_STATUS                       0x30                        //
Offset for USBHostDeviceStatus() return codes
#define USB_DEVICE_ATTACHED                     (USB_DEVICE_STATUS | 0x30)  //
Device is attached and running
#define USB_DEVICE_DETACHED                     (USB_DEVICE_STATUS | 0x01)  // No
device is attached
#define USB_DEVICE_ENUMERATING                  (USB_DEVICE_STATUS | 0x02)  //
Device is enumerating
#define USB_HOLDING_OUT_OF_MEMORY               (USB_DEVICE_STATUS | 0x03)  //
Not enough heap space available
#define USB_HOLDING_UNSUPPORTED_DEVICE          (USB_DEVICE_STATUS | 0x04)  //
Invalid configuration or unsupported class
#define USB_HOLDING_UNSUPPORTED_HUB             (USB_DEVICE_STATUS | 0x05)  //
Hubs are not supported
#define USB_HOLDING_INVALID_CONFIGURATION       (USB_DEVICE_STATUS | 0x06)  //
Invalid configuration requested
#define USB_HOLDING_PROCESSING_CAPACITY         (USB_DEVICE_STATUS | 0x07)  //
Processing requirement excessive
#define USB_HOLDING_POWER_REQUIREMENT           (USB_DEVICE_STATUS | 0x08)  //
Power requirement excessive
#define USB_HOLDING_CLIENT_INIT_ERROR           (USB_DEVICE_STATUS | 0x09)  //
Client driver failed to initialize
#define USB_DEVICE_SUSPENDED                    (USB_DEVICE_STATUS | 0x0A)  //
Device is suspended

#define USB_ERROR_CLASS_DEFINED                 0x50    // Offset for application
defined errors

#define USB_SINGLE_DEVICE_ADDRESS               0x01    // Default USB device
address (single device support)


// *****************************************************************************
// *****************************************************************************
// Section: USB Data Types
// *****************************************************************************
// *****************************************************************************

// *****************************************************************************
/* Data Transfer Flags

The following flags are used in the flags parameter of the "USBDEVTransferData"
and "USBHALTransferData" routines.  They can be accessed by the bitfield
definitions or the macros can be OR'd together to identify the endpoint number
```

```
and properties of the data transfer.

<code>
 7 6 5 4 3 2 1 0 - Field name
 | | | | \_____/
 | | | |    +----- ep_num    - Endpoint number
 | | | +---------- zero_pkt  - End transfer with short or zero-sized packet
 | | +------------ dts       - 0=DATA0 packet, 1=DATA1 packet
 | +-------------- force_dts - Force data toggle sync to match dts field
 +---------------- direction - Transfer direction: 0=Receive, 1=Transmit
</code>
*/

typedef union
{
    BYTE    bitmap;
    struct
    {
        BYTE ep_num:    4;
        BYTE zero_pkt:  1;
        BYTE dts:       1;
        BYTE force_dts: 1;
        BYTE direction: 1;
    }field;

} TRANSFER_FLAGS;


// ****************************************************************************
/* Data Transfer Flags, Endpoint Number Constants

These macros can be used as values for the "ep_num" field of the TRANSFER_FLAGS
data type.
*/
#define USB_EP0         0       //
#define USB_EP1         1       //
#define USB_EP2         2       //
#define USB_EP3         3       //
#define USB_EP4         4       //
#define USB_EP5         5       //
#define USB_EP6         6       //
#define USB_EP7         7       //
#define USB_EP8         8       //
#define USB_EP9         9       //
#define USB_EP10        10      //
#define USB_EP11        11      //
#define USB_EP12        12      //
#define USB_EP13        13      //
#define USB_EP14        14      //
#define USB_EP15        15      //

// ****************************************************************************
/* Data Transfer Flags, Bitmap Constants

These macros can be used as values for the "bitmap" field of the TRANSFER_FLAGS
data type.
*/
#define USB_TRANSMIT        0x80                        // Data will be
transmitted to the USB
#define USB_RECEIVE         0x00                        // Data will be
received from the USB
#define USB_FORCE_DTS       0x40                        // Forces data toggle
sync as below:
#define USB_DTS_MASK        0x20                        // Mask for DTS bit
(below)
#define USB_ZERO_PKT        0x10                        // End transfer w/a
short or zero-length packet
```

```
#define USB_DATA0           0x00|USB_FORCE_DTS              // Force DATA0
#define USB_DATA1           0x20|USB_FORCE_DTS              // Force DATA1
#define USB_SETUP_PKT       USB_RECEIVE|USB_DATA0|USB_EP0   // Setup Packet
#define USB_SETUP_DATA      USB_DATA1|USB_ZERO_PKT|USB_EP0  // Setup-transfer
Data Packet
#define USB_SETUP_STATUS    USB_DATA1|USB_EP0               // Setup-transfer
Status Packet
#define USB_EP_NUM_MASK     0x0F                            // Endpoint number
(ep_num) mask

// ***************************************************************************
/* Data Transfer Flags, Initialization Macro

This macro can be used with the above bitmap constants to initialize a
TRANSFER_FLAGS value.  It provides the correct data type to avoid compiler
warnings.
*/
#define XFLAGS(f) ((TRANSFER_FLAGS)((BYTE)(f)))             // Initialization
Macro


// ***************************************************************************
/* USB Events

This enumeration identifies USB events that occur.  It is used to
inform USB drivers and applications of events on the bus.  It is passed
as a parameter to the event-handling routine, which must match the
prototype of the USB_CLIENT_EVENT_HANDLER data type, when an event occurs.
*/

typedef enum
{
    // No event occured (NULL event)
    EVENT_NONE = 0,

    // A USB transfer has completed.  The data associated with this event is of
    // the data type HOST_TRANSFER_DATA if the event is generated from the host
    // stack.
    EVENT_TRANSFER,

    // A USB Start of Frame token has been received.  This event is not
    // used by the Host stack.
    EVENT_SOF,

    // Device-mode resume received.  This event is not used by the Host stack.
    EVENT_RESUME,

    // Device-mode suspend/idle event received.  This event is not used by the
    // Host stack.
    EVENT_SUSPEND,

    // Device-mode bus reset received.  This event is not used by the Host
    // stack.
    EVENT_RESET,

    // Device-mode USB cable has been attached.  This event is not used by the
    // Host stack.  The client driver may provide an application event when a
    // device attaches.
    EVENT_ATTACH,

    // USB cable has been detached.  The data associated with this event is the
    // address of detached device, a single BYTE.
    EVENT_DETACH,

    // A USB hub has been attached.  Hub support is not currently available.
    EVENT_HUB_ATTACH,
```

```
    // A stall has occured.  This event is not used by the Host stack.
    EVENT_STALL,

    // Device Mode: A setup packet received (data: SETUP_PKT).  This event is
    // not used by the Host stack.
    EVENT_SETUP,

    // VBus SRP Pulse, (VBus > 2.0v),  Data: BYTE Port Number (For future
support)
    EVENT_VBUS_SES_REQUEST,

    // The voltage on Vbus has dropped below 4.4V/4.7V.  The application is
    // responsible for monitoring Vbus and calling USBHostVbusEvent() with this
    // event.  This event is not generated by the stack.
    EVENT_VBUS_OVERCURRENT,

    // An enumerating device is requesting power.  The data associated with this
    // event is of the data type USB_VBUS_POWER_EVENT_DATA.  Note that
    // the requested current is specified in 2mA units, identical to the power
    // specification in a device's Configuration Descriptor.
    EVENT_VBUS_REQUEST_POWER,

    // Release power from a detaching device. The data associated with this
    // event is of the data type USB_VBUS_POWER_EVENT_DATA.  The current value
    // specified in the data can be ignored.
    EVENT_VBUS_RELEASE_POWER,

    // The voltage on Vbus is good, and the USB OTG module can be powered on.
    // The application is responsible for monitoring Vbus and calling
    // USBHostVbusEvent() with this event.  This event is not generated by the
    // stack.  If the application issues an EVENT_VBUS_OVERCURRENT, then no
    // power will be applied to that port, and no device can attach to that
    // port, until the application issues the EVENT_VBUS_POWER_AVAILABLE for
    // the port.
    EVENT_VBUS_POWER_AVAILABLE,

    // The attached device is not supported by the application.  The attached
    // device is not allowed to enumerate.
    EVENT_UNSUPPORTED_DEVICE,

    // Cannot enumerate the attached device.  This is generated if communication
    // errors prevent the device from enumerating.
    EVENT_CANNOT_ENUMERATE,

    // The client driver cannot initialize the the attached device.  The
    // attached is not allowed to enumerate.
    EVENT_CLIENT_INIT_ERROR,

    // The Host stack does not have enough heap space to enumerate the device.
    // Check the amount of heap space allocated to the application.  In MPLAB,
    // select Project> Build Options...> Project.  Select the appropriate
    // linker tab, and inspect the "Heap size" entry.
    EVENT_OUT_OF_MEMORY,

    // Unspecified host error. (This error should not occur).
    EVENT_UNSPECIFIED_ERROR,

    // Notification that a SET_CONFIGURATION() command was received (device)
    EVENT_CONFIGURED,

    // A SET_DESCRIPTOR request was received (device)
    EVENT_SET_DESCRIPTOR,

    // An endpoint 0 request was received that the stack did not know how to
    // handle.  This is most often a request for one of the class drivers.
    // Please refer to the class driver documenation for information related
    // to what to do if this request is received. (device)
```

```
    EVENT_EP0_REQUEST,

    // Class-defined event offsets start here:
    EVENT_GENERIC_BASE  = 100,      // Offset for Generic class events

    EVENT_MSD_BASE      = 200,      // Offset for Mass Storage Device class
events

    EVENT_HID_BASE      = 300,      // Offset for Human Interface Device class
events

    EVENT_PRINTER_BASE  = 400,      // Offset for Printer class events

    EVENT_CDC_BASE      = 500,      // Offset for CDC class events

    EVENT_CHARGER_BASE  = 600,      // Offset for Charger client driver events.

       EVENT_USER_BASE     = 10000,    // Add integral values to this event
number
                                    // to create user-defined events.

    // There was a transfer error on the USB.  The data associated with this
    // event is of data type HOST_TRANSFER_DATA.
    EVENT_BUS_ERROR     = UINT_MAX

} USB_EVENT;


// ****************************************************************************
/* EVENT_TRANSFER Data

This data structure is passed to the appropriate layer's
USB_EVENT_HANDLER when an EVT_XFER event has occured, indicating
that a transfer has completed on the USB.  It provides the endpoint,
direction, and actual size of the transfer.
 */

typedef struct _transfer_event_data
{
    TRANSFER_FLAGS  flags;          // Transfer flags (see above)
    UINT32          size;           // Actual number of bytes transferred
    BYTE            pid;            // Packet ID

} USB_TRANSFER_EVENT_DATA;


// ****************************************************************************
/* EVENT_VBUS_REQUEST_POWER and EVENT_VBUS_RELEASE_POWER Data

This data structure is passed to the appropriate layer's
USB_EVENT_HANDLER when an EVENT_VBUS_REQUEST_POWER or EVENT_VBUS_RELEASE_POWER
event has occured, indicating that a change in Vbus power is being requested.
*/

typedef struct _vbus_power_data
{
    BYTE            port;           // Physical port number
    BYTE            current;        // Current in 2mA units
} USB_VBUS_POWER_EVENT_DATA;


// ****************************************************************************
/* EVT_STALL Data

The EVT_STALL event has a 16-bit data value associated with it where
a bit is set in the position for each endpoint that is currently
stalled (ie. bit 0 = EP0, bit 1 = EP1, etc.)
```

```
*/


// ****************************************************************************
// ****************************************************************************
// Section: Event  Handling Routines
// ****************************************************************************
// ****************************************************************************

/*****************************************************************************
    Function:
        BOOL <Event-handling Function Name> ( USB_EVENT event,
              void *data, unsigned int size )

    Description:
        This routine is a "call out" routine that must be implemented by
        any layer of the USB SW Stack (except the HAL which is at the
        root of the event-call tree that needs to receive events.  When
        an event occurs, the HAL calls the next higher layer in the
        stack to handle the event.  Each layer either handles the event
        or calls the layer above it to handle the event.  Events are
        identified by the "event" parameter and may have associated
        data.  If the higher layer was able to handle the event, it
        should return TRUE.  If not, it should return FALSE.

    Preconditions:
        USBInitialize must have been called to initialize the USB SW
        Stack.

    Paramters:
        USB_EVENT event    - Identifies the bus event that occured
        void *data         - Pointer to event-specific data
        unsigned int size - Size of the event-specific data

    Return Values:
        None

    Remarks:
        The function is name is defined by the layer that implements
        it.  A pointer to the function will be placed by into a table
        that the lower-layer will use to call it.  This requires the
        function to use a specific call "signature" (return data type
        and values and data parameter types and values).

*****************************************************************************/

typedef BOOL (*USB_EVENT_HANDLER) ( USB_EVENT event, void *data, unsigned int
size );


// ****************************************************************************
// ****************************************************************************
// Section: USB Application Program Interface (API) Routines
// ****************************************************************************
// ****************************************************************************

/*****************************************************************************
    Function:
        BOOL USBInitialize ( unsigned long flags )

    Summary:
        This interface initializes the variables of the USB host stack.

    Description:
        This interface initializes the USB stack.

    Precondition:
```

```
        None

    Parameters:
        flags - reserved

    Return Values:
        TRUE  - Initialization successful
        FALSE - Initialization failure

    Remarks:
        This interface is implemented as a macro that can be defined by the
        application or by default is defined correctly for the stack mode.

  ***************************************************************************/

#ifndef USBInitialize
    #if defined( USB_SUPPORT_DEVICE )
        #if defined( USB_SUPPORT_HOST )
            #if defined( USB_SUPPORT_OTG )
                #error "USB OTG is not yet supported."
            #else
                #define USBInitialize(f) \
                        (USBDEVInitialize(f) && USBHostInit(f)) ? \
                        TRUE : FALSE
            #endif
        #else
            #define USBInitialize(f) USBDEVInitialize(f)
        #endif
    #else
        #if defined( USB_SUPPORT_HOST )
            #define USBInitialize(f) USBHostInit(f)
        #else
            #error "Application must define support mode in usb_config.h"
        #endif
    #endif
#endif


/*********************************************************************
    Function:
        void USBTasks( void )

    Summary:
        This function executes the tasks for USB operation.

    Description:
        This function executes the tasks for USB host operation.  It must be
        executed on a regular basis to keep everything functioning.

    Precondition:
        USBInitialize() has been called.

    Parameters:
        None

    Returns:
        None

    Remarks:
        This interface is implemented as a macro that can be defined by the
        application or by default is defined correctly for the stack mode.

  ********************************************************************/

#ifndef USBTasks    // Implemented as a macro that can be overridden.
    #if defined( USB_SUPPORT_DEVICE )
        #if defined( USB_SUPPORT_HOST )
```

```
            #if defined( USB_SUPPORT_OTG )
                #error "USB OTG is not yet supported."
            #else
                #define USBTasks() {USBHostTasks(); USBHALHandleBusEvent();}
            #endif
        #else
            #define USBTasks() USBHALHandleBusEvent()
        #endif
    #else
        #if defined( USB_SUPPORT_HOST )
            #define USBTasks() USBHostTasks()
        #else
            #error "Application must define support mode in usb_config.h"
        #endif
    #endif
#endif


#endif // _USB_COMMON_H_
/*******************************************************************************
 * EOF
 */
```

6.1.12. usb_ch9.h

```
/********************************************************************************

* FileName:        usb_ch9.h
* Dependencies:    None
* Processor:       PIC18/PIC24/PIC32MX microcontrollers with USB module
* Compiler:        C18 v3.13+/C30 v2.01+/C32 v0.00.18+
* Company:         Microchip Technology, Inc.
* File Description:
* This file contains the definitions and prototypes used for
* specification chapter 9 compliance.

Software License Agreement

The software supplied herewith by Microchip Technology Incorporated
(the “Company”) for its PICmicro® Microcontroller is intended and
supplied to you, the Company’s customer, for use solely and
exclusively on Microchip PICmicro Microcontroller products. The
software is owned by the Company and/or its supplier, and is
protected under applicable copyright laws. All rights are reserved.
Any use in violation of the foregoing restrictions may subject the
user to criminal sanctions under applicable laws, as well as to
civil liability for the breach of the terms and conditions of this
license.

THIS SOFTWARE IS PROVIDED IN AN “AS IS” CONDITION. NO WARRANTIES,
WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

********************************************************************************/
//DOM-IGNORE-BEGIN
/********************************************************************************

    USB Chapter 9 Protocol (Header File)

Summary:
    This file defines data structures, constants, and macros that are used to
    to support the USB Device Framework protocol described in Chapter 9 of the
    USB 2.0 specification.

Description:
    This file defines data structures, constants, and macros that are used to
    to support the USB Device Framework protocol described in Chapter 9 of the
    USB 2.0 specification.

    This file is located in the "\<Install Directory\>\\Microchip\\Include\\USB"
    directory.

    When including this file in a new project, this file can either be
    referenced from the directory in which it was installed or copied
    directly into the user application folder. If the first method is
    chosen to keep the file located in the folder in which it is installed
    then include paths need to be added so that the library and the
    application both know where to reference each others files. If the
    application folder is located in the same folder as the Microchip
    folder (like the current demo folders), then the following include
    paths need to be added to the application's project:

    ..\\Include

    ..\\..\\Include

    ..\\..\\MicrochipInclude
```

```
    ..\\..\\\<Application Folder\>

    ..\\..\\..\\\<Application Folder\>

    If a different directory structure is used, modify the paths as
    required. An example using absolute paths instead of relative paths
    would be the following:

    C:\\Microchip Solutions\\Microchip\\Include

    C:\\Microchip Solutions\\My Demo Application
*********************************************************************/
//DOM-IGNORE-END

//DOM-IGNORE-BEGIN
#ifndef _USB_CH9_H_
#define _USB_CH9_H_
//DOM-IGNORE-END


// *****************************************************************************
// *****************************************************************************
// Section: USB Descriptors
// *****************************************************************************
// *****************************************************************************

#define USB_DESCRIPTOR_DEVICE           0x01    // bDescriptorType for a Device
Descriptor.
#define USB_DESCRIPTOR_CONFIGURATION    0x02    // bDescriptorType for a
Configuration Descriptor.
#define USB_DESCRIPTOR_STRING           0x03    // bDescriptorType for a String
Descriptor.
#define USB_DESCRIPTOR_INTERFACE        0x04    // bDescriptorType for an
Interface Descriptor.
#define USB_DESCRIPTOR_ENDPOINT         0x05    // bDescriptorType for an
Endpoint Descriptor.
#define USB_DESCRIPTOR_DEVICE_QUALIFIER 0x06    // bDescriptorType for a Device
Qualifier.
#define USB_DESCRIPTOR_OTHER_SPEED      0x07    // bDescriptorType for a Other
Speed Configuration.
#define USB_DESCRIPTOR_INTERFACE_POWER  0x08    // bDescriptorType for Interface
Power.
#define USB_DESCRIPTOR_OTG              0x09    // bDescriptorType for an OTG
Descriptor.

// *****************************************************************************
/* USB Device Descriptor Structure

This struct defines the structure of a USB Device Descriptor.  Note that this
structure may need to be packed, or even accessed as bytes, to properly access
the correct fields when used on some device architectures.
*/
typedef struct __attribute__ ((packed)) _USB_DEVICE_DESCRIPTOR
{
    BYTE bLength;               // Length of this descriptor.
    BYTE bDescriptorType;       // DEVICE descriptor type
(USB_DESCRIPTOR_DEVICE).
    WORD bcdUSB;                // USB Spec Release Number (BCD).
    BYTE bDeviceClass;          // Class code (assigned by the USB-IF). 0xFF-
Vendor specific.
    BYTE bDeviceSubClass;       // Subclass code (assigned by the USB-IF).
    BYTE bDeviceProtocol;       // Protocol code (assigned by the USB-IF). 0xFF-
Vendor specific.
    BYTE bMaxPacketSize0;       // Maximum packet size for endpoint 0.
    WORD idVendor;              // Vendor ID (assigned by the USB-IF).
    WORD idProduct;             // Product ID (assigned by the manufacturer).
```

```
    WORD bcdDevice;             // Device release number (BCD).
    BYTE iManufacturer;         // Index of String Descriptor describing the
manufacturer.
    BYTE iProduct;              // Index of String Descriptor describing the
product.
    BYTE iSerialNumber;         // Index of String Descriptor with the device's
serial number.
    BYTE bNumConfigurations;    // Number of possible configurations.
} USB_DEVICE_DESCRIPTOR;


// ************************************************************************
/* USB Configuration Descriptor Structure

This struct defines the structure of a USB Configuration Descriptor.  Note that
this
structure may need to be packed, or even accessed as bytes, to properly access
the correct fields when used on some device architectures.
*/
typedef struct __attribute__ ((packed)) _USB_CONFIGURATION_DESCRIPTOR
{
    BYTE bLength;               // Length of this descriptor.
    BYTE bDescriptorType;       // CONFIGURATION descriptor type
(USB_DESCRIPTOR_CONFIGURATION).
    WORD wTotalLength;          // Total length of all descriptors for this
configuration.
    BYTE bNumInterfaces;        // Number of interfaces in this configuration.
    BYTE bConfigurationValue;   // Value of this configuration (1 based).
    BYTE iConfiguration;        // Index of String Descriptor describing the
configuration.
    BYTE bmAttributes;          // Configuration characteristics.
    BYTE bMaxPower;             // Maximum power consumed by this configuration.
} USB_CONFIGURATION_DESCRIPTOR;

// Attributes bits
#define USB_CFG_DSC_REQUIRED     0x80                       // Required attribute
#define USB_CFG_DSC_SELF_PWR    (0x40|USB_CFG_DSC_REQUIRED) // Device is self
powered.
#define USB_CFG_DSC_REM_WAKE    (0x20|USB_CFG_DSC_REQUIRED) // Device can request
remote wakup


// ************************************************************************
/* USB Interface Descriptor Structure

This struct defines the structure of a USB Interface Descriptor.  Note that this
structure may need to be packed, or even accessed as bytes, to properly access
the correct fields when used on some device architectures.
*/
typedef struct __attribute__ ((packed)) _USB_INTERFACE_DESCRIPTOR
{
    BYTE bLength;               // Length of this descriptor.
    BYTE bDescriptorType;       // INTERFACE descriptor type
(USB_DESCRIPTOR_INTERFACE).
    BYTE bInterfaceNumber;      // Number of this interface (0 based).
    BYTE bAlternateSetting;     // Value of this alternate interface setting.
    BYTE bNumEndpoints;         // Number of endpoints in this interface.
    BYTE bInterfaceClass;       // Class code (assigned by the USB-IF).  0xFF-
Vendor specific.
    BYTE bInterfaceSubClass;    // Subclass code (assigned by the USB-IF).
    BYTE bInterfaceProtocol;    // Protocol code (assigned by the USB-IF).  0xFF-
Vendor specific.
    BYTE iInterface;            // Index of String Descriptor describing the
interface.
} USB_INTERFACE_DESCRIPTOR;
```

```
// ***************************************************************************
/* USB Endpoint Descriptor Structure

This struct defines the structure of a USB Endpoint Descriptor.  Note that this
structure may need to be packed, or even accessed as bytes, to properly access
the correct fields when used on some device architectures.
*/
typedef struct __attribute__ ((packed)) _USB_ENDPOINT_DESCRIPTOR
{
    BYTE bLength;               // Length of this descriptor.
    BYTE bDescriptorType;       // ENDPOINT descriptor type
(USB_DESCRIPTOR_ENDPOINT).
    BYTE bEndpointAddress;      // Endpoint address. Bit 7 indicates direction
(0=OUT, 1=IN).
    BYTE bmAttributes;          // Endpoint transfer type.
    WORD wMaxPacketSize;        // Maximum packet size.
    BYTE bInterval;             // Polling interval in frames.
} USB_ENDPOINT_DESCRIPTOR;


// Endpoint Direction
#define EP_DIR_IN           0x80    // Data flows from device to host
#define EP_DIR_OUT          0x00    // Data flows from host to device


// ************************************************************
// USB Endpoint Attributes
// ************************************************************

// Section: Transfer Types
#define EP_ATTR_CONTROL     (0<<0)  // Endoint used for control transfers
#define EP_ATTR_ISOCH       (1<<0)  // Endpoint used for isochronous transfers
#define EP_ATTR_BULK        (2<<0)  // Endpoint used for bulk transfers
#define EP_ATTR_INTR        (3<<0)  // Endpoint used for interrupt transfers

// Section: Synchronization Types (for isochronous enpoints)
#define EP_ATTR_NO_SYNC     (0<<2)  // No Synchronization
#define EP_ATTR_ASYNC       (1<<2)  // Asynchronous
#define EP_ATTR_ADAPT       (2<<2)  // Adaptive synchronization
#define EP_ATTR_SYNC        (3<<2)  // Synchronous

// Section: Usage Types (for isochronous endpoints)
#define EP_ATTR_DATA        (0<<4)  // Data Endpoint
#define EP_ATTR_FEEDBACK    (1<<4)  // Feedback endpoint
#define EP_ATTR_IMP_FB      (2<<4)  // Implicit Feedback data EP

// Section: Max Packet Sizes
#define EP_MAX_PKT_INTR_LS  8       // Max low-speed interrupt packet
#define EP_MAX_PKT_INTR_FS  64      // Max full-speed interrupt packet
#define EP_MAX_PKT_ISOCH_FS 1023    // Max full-speed isochronous packet
#define EP_MAX_PKT_BULK_FS  64      // Max full-speed bulk packet
#define EP_LG_PKT_BULK_FS   32      // Large full-speed bulk packet
#define EP_MED_PKT_BULK_FS  16      // Medium full-speed bulk packet
#define EP_SM_PKT_BULK_FS   8       // Small full-speed bulk packet


// ***************************************************************************
/* USB OTG Descriptor Structure

This struct defines the structure of a USB OTG Descriptor.  Note that this
structure may need to be packed, or even accessed as bytes, to properly access
the correct fields when used on some device architectures.
*/
typedef struct __attribute__ ((packed)) _USB_OTG_DESCRIPTOR
{
    BYTE bLength;               // Length of this descriptor.
    BYTE bDescriptorType;       // OTG descriptor type (USB_DESCRIPTOR_OTG).
```

```
    BYTE bmAttributes;          // OTG attributes.
} USB_OTG_DESCRIPTOR;


// ************************************************************************
// Section: USB String Descriptor Structure
// ************************************************************************
// This structure describes the USB string descriptor.  The string
// descriptor provides user-readable information about various aspects of
// the device.  The first string desriptor (string descriptor zero (0)),
// provides a list of the number of languages supported by the set of
// string descriptors for this device instead of an actual string.
//
// Note: The strings are in 2-byte-per-character unicode, not ASCII.
//
// Note: This structure only describes the "header" of the string
// descriptor.  The actual data (either the language ID array or the
// array of unicode characters making up the string, must be allocated
// immediately following this header with no padding between them.

typedef struct __attribute__ ((packed)) _USB_STRING_DSC
{
    BYTE   bLength;             // Size of this descriptor
    BYTE   bDescriptorType;     // Type, USB_DSC_STRING

} USB_STRING_DESCRIPTOR;


// ************************************************************************
// Section: USB Device Qualifier Descriptor Structure
// ************************************************************************
// This structure describes the device qualifier descriptor.  The device
// qualifier descriptor provides overall device information if the device
// supports "other" speeds.
//
// Note: A high-speed device may support "other" speeds (ie. full or low).
// If so, it may need to implement the the device qualifier and other
// speed descriptors.

typedef struct __attribute__ ((packed)) _USB_DEVICE_QUALIFIER_DESCRIPTOR
{
    BYTE bLength;               // Size of this descriptor
    BYTE bType;                 // Type, always USB_DESCRIPTOR_DEVICE_QUALIFIER
    WORD bcdUSB;                // USB spec version, in BCD
    BYTE bDeviceClass;          // Device class code
    BYTE bDeviceSubClass;       // Device sub-class code
    BYTE bDeviceProtocol;       // Device protocol
    BYTE bMaxPacketSize0;       // EP0, max packet size
    BYTE bNumConfigurations;    // Number of "other-speed" configurations
    BYTE bReserved;             // Always zero (0)

} USB_DEVICE_QUALIFIER_DESCRIPTOR;


// ************************************************************************
// Section: USB Setup Packet Structure
// ************************************************************************
// This structure describes the data contained in a USB standard device
// request's setup packet.  It is the data packet sent from the host to
// the device to control and configure the device.
//
// Note: Refer to the USB 2.0 specification for additional details on the
// usage of the setup packet and standard device requests.

typedef struct __attribute__ ((packed)) SetupPkt
{
    union                           // offset   description
```

```
    {                                   // ------   ------------------------
        BYTE bmRequestType;             //   0      Bit-map of request type
        struct
        {
            BYTE    recipient:  5;  //          Recipient of the request
            BYTE    type:       2;  //          Type of request
            BYTE    direction:  1;  //          Direction of data X-fer
        };
    }requestInfo;

    BYTE    bRequest;               //   1      Request type
    UINT16  wValue;                 //   2      Depends on bRequest
    UINT16  wIndex;                 //   4      Depends on bRequest
    UINT16  wLength;                //   6      Depends on bRequest

} SETUP_PKT, *PSETUP_PKT;


// *****************************************************************
// *****************************************************************
// Section: USB Specification Constants
// *****************************************************************
// *****************************************************************

// Section: Valid PID Values
//DOM-IGNORE-BEGIN
#define PID_OUT                                 0x1     // PID for an OUT token
#define PID_ACK                                 0x2     // PID for an ACK
handshake
#define PID_DATA0                               0x3     // PID for DATA0 data
#define PID_PING                                0x4     // Special PID PING
#define PID_SOF                                 0x5     // PID for a SOF token
#define PID_NYET                                0x6     // PID for a NYET
handshake
#define PID_DATA2                               0x7     // PID for DATA2 data
#define PID_SPLIT                               0x8     // Special PID SPLIT
#define PID_IN                                  0x9     // PID for a IN token
#define PID_NAK                                 0xA     // PID for a NAK
handshake
#define PID_DATA1                               0xB     // PID for DATA1 data
#define PID_PRE                                 0xC     // Special PID PRE (Same
as PID_ERR)
#define PID_ERR                                 0xC     // Special PID ERR (Same
as PID_PRE)
#define PID_SETUP                               0xD     // PID for a SETUP token
#define PID_STALL                               0xE     // PID for a STALL
handshake
#define PID_MDATA                               0xF     // PID for MDATA data

#define PID_MASK_DATA                           0x03    // Data PID mask
#define PID_MASK_DATA_SHIFTED                  (PID_MASK_DATA << 2) // Data PID
shift to proper position
//DOM-IGNORE-END

// Section: USB Token Types
//DOM-IGNORE-BEGIN
#define USB_TOKEN_OUT                           0x01    // U1TOK - OUT token
#define USB_TOKEN_IN                            0x09    // U1TOK - IN token
#define USB_TOKEN_SETUP                         0x0D    // U1TOK - SETUP token
//DOM-IGNORE-END

// Section: OTG Descriptor Constants

#define OTG_HNP_SUPPORT                         0x02    // OTG Descriptor
bmAttributes - HNP support flag
#define OTG_SRP_SUPPORT                         0x01    // OTG Descriptor
bmAttributes - SRP support flag
```

```
// Section: Endpoint Directions

#define USB_IN_EP                               0x80    // IN endpoint mask
#define USB_OUT_EP                              0x00    // OUT endpoint mask

// Section: Standard Device Requests

#define USB_REQUEST_GET_STATUS                  0       // Standard Device
Request - GET STATUS
#define USB_REQUEST_CLEAR_FEATURE               1       // Standard Device
Request - CLEAR FEATURE
#define USB_REQUEST_SET_FEATURE                 3       // Standard Device
Request - SET FEATURE
#define USB_REQUEST_SET_ADDRESS                 5       // Standard Device
Request - SET ADDRESS
#define USB_REQUEST_GET_DESCRIPTOR              6       // Standard Device
Request - GET DESCRIPTOR
#define USB_REQUEST_SET_DESCRIPTOR              7       // Standard Device
Request - SET DESCRIPTOR
#define USB_REQUEST_GET_CONFIGURATION           8       // Standard Device
Request - GET CONFIGURATION
#define USB_REQUEST_SET_CONFIGURATION           9       // Standard Device
Request - SET CONFIGURATION
#define USB_REQUEST_GET_INTERFACE               10      // Standard Device
Request - GET INTERFACE
#define USB_REQUEST_SET_INTERFACE               11      // Standard Device
Request - SET INTERFACE
#define USB_REQUEST_SYNCH_FRAME                 12      // Standard Device
Request - SYNCH FRAME

#define USB_FEATURE_ENDPOINT_HALT               0       // CLEAR/SET FEATURE -
Endpoint Halt
#define USB_FEATURE_DEVICE_REMOTE_WAKEUP        1       // CLEAR/SET FEATURE -
Device remote wake-up
#define USB_FEATURE_TEST_MODE                   2       // CLEAR/SET FEATURE -
Test mode

// Section: Setup Data Constants

#define USB_SETUP_HOST_TO_DEVICE                0x00    // Device Request
bmRequestType transfer direction - host to device transfer
#define USB_SETUP_DEVICE_TO_HOST                0x80    // Device Request
bmRequestType transfer direction - device to host transfer
#define USB_SETUP_TYPE_STANDARD                 0x00    // Device Request
bmRequestType type - standard
#define USB_SETUP_TYPE_CLASS                    0x20    // Device Request
bmRequestType type - class
#define USB_SETUP_TYPE_VENDOR                   0x40    // Device Request
bmRequestType type - vendor
#define USB_SETUP_RECIPIENT_DEVICE              0x00    // Device Request
bmRequestType recipient - device
#define USB_SETUP_RECIPIENT_INTERFACE           0x01    // Device Request
bmRequestType recipient - interface
#define USB_SETUP_RECIPIENT_ENDPOINT            0x02    // Device Request
bmRequestType recipient - endpoint
#define USB_SETUP_RECIPIENT_OTHER               0x03    // Device Request
bmRequestType recipient - other

// Section: OTG SET FEATURE Constants

#define OTG_FEATURE_B_HNP_ENABLE                3       // SET FEATURE OTG -
Enable B device to perform HNP
#define OTG_FEATURE_A_HNP_SUPPORT               4       // SET FEATURE OTG - A
device supports HNP
#define OTG_FEATURE_A_ALT_HNP_SUPPORT           5       // SET FEATURE OTG -
Another port on the A device supports HNP
```

```
// Section: USB Endpoint Transfer Types

#define USB_TRANSFER_TYPE_CONTROL               0x00    // Endpoint is a control
endpoint.
#define USB_TRANSFER_TYPE_ISOCHRONOUS           0x01    // Endpoint is an
isochronous endpoint.
#define USB_TRANSFER_TYPE_BULK                  0x02    // Endpoint is a bulk
endpoint.
#define USB_TRANSFER_TYPE_INTERRUPT             0x03    // Endpoint is an
interrupt endpoint.

// Section: Standard Feature Selectors for CLEAR_FEATURE Requests
#define USB_FEATURE_ENDPOINT_STALL              0       // Endpoint recipient
#define USB_FEATURE_DEVICE_REMOTE_WAKEUP        1       // Device recipient
#define USB_FEATURE_TEST_MODE                   2       // Device recipient


// Section: USB Class Code Definitions
#define USB_HUB_CLASSCODE                       0x09    //  Class code for a hub.


#endif  // _USB_CH9_H_
/*****************************************************************************
 * EOF
 */
```

6.1.13. usb_hal.h

```
/********************************************************************

 File Description:

 This file defines the interface to the USB hardware abstraction layer.

 Filename:        usb_hal.h
 Dependancies:    none
 Processor:       PIC18, PIC24, or PIC32 USB Microcontrollers
 Hardware:        The code is natively intended to be used on the following
                  hardware platforms: PICDEM™ FS USB Demo Board,
                  PIC18F87J50 FS USB Plug-In Module, or
                  Explorer 16 + PIC24 USB PIM.  The firmware may be
                  modified for use on other USB platforms by editing the
                  HardwareProfile.h file.
 Compiler:        Microchip C18 (for PIC18) or C30 (for PIC24)
 Company:         Microchip Technology, Inc.

 Software License Agreement:

 The software supplied herewith by Microchip Technology Incorporated
 (the “Company”) for its PICmicro® Microcontroller is intended and
 supplied to you, the Company’s customer, for use solely and
 exclusively on Microchip PICmicro Microcontroller products. The
 software is owned by the Company and/or its supplier, and is
 protected under applicable copyright laws. All rights are reserved.
 Any use in violation of the foregoing restrictions may subject the
 user to criminal sanctions under applicable laws, as well as to
 civil liability for the breach of the terms and conditions of this
 license.

 THIS SOFTWARE IS PROVIDED IN AN “AS IS” CONDITION. NO WARRANTIES,
 WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
 TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
 IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
 CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

 Change History:

 ********************************************************************/
//DOM-IGNORE-BEGIN
/********************************************************************

    USB Hardware Abstraction Layer (HAL)  (Header File)

Summary:
    This file abstracts the hardware interface.

Description:
    This file abstracts the hardware interface.

    This file is located in the "\<Install Directory\>\\Microchip\\Include\\USB"
    directory.

    When including this file in a new project, this file can either be
    referenced from the directory in which it was installed or copied
    directly into the user application folder. If the first method is
    chosen to keep the file located in the folder in which it is installed
    then include paths need to be added so that the library and the
    application both know where to reference each others files. If the
    application folder is located in the same folder as the Microchip
    folder (like the current demo folders), then the following include
    paths need to be added to the application's project:
```

```
    ..\\Include

    ..\\..\\Include

    ..\\..\\MicrochipInclude

    ..\\..\\\<Application Folder\>

    ..\\..\\..\\\<Application Folder\>

    If a different directory structure is used, modify the paths as
    required. An example using absolute paths instead of relative paths
    would be the following:

    C:\\Microchip Solutions\\Microchip\\Include

    C:\\Microchip Solutions\\My Demo Application
*******************************************************************************/

#ifndef _USB_HAL_H_
#define _USB_HAL_H_
//DOM-IGNORE-END

#include ".\USB\usb_common.h"

#if defined(USB_SUPPORT_HOST) || defined(USB_SUPPORT_OTG)

#if defined(__C30__)
    #include "USB\usb_hal_pic24.h"

#elif defined(__PIC32MX__)
    #include "USB\usb_hal_pic32.h"
#endif

/*********************
 Interface Routines
 *********************/

/*******************************************************************
    Function:
        void USBHALSetBusAddress( BYTE addr )

    Description:
        This routine sets the address of the system on the USB
        when acting as a peripheral device.

    Preconditions:
        1. USBHALInitialize must have been called to
           initialize the USB HAL.
        2. Endpoint zero (0) must be configured as appropriate
           by calls to USBHALSetEpConfiguration.
        3. The system must have been enumerated on the USB (as
           a device).

    Parameters:
        addr    Desired address of this device on the USB.

    Return Values:
        None

    Side Effect:
        The bus address has been set.

    Remmarks:
        The address is assigned by the host and is received in
        a SET_ADDRESS setup request.
```

```
 ********************************************************************/
/*
 This routine is implemented as a macro to a lower-level level routine.
 */

#define USBHALSetBusAddress OTGCORE_SetDeviceAddr

void USBHALSetBusAddress( BYTE addr );


/*********************************************************************
    Function:
        void USBHALControlUsbResistors( BYTE flags );

    Description:
        This routine enables or disables the USB pull-up or
        pull-down resistors as requested.

    Precondition:
        USBInitialize must have been called to initialize the
        USB SW stack.

    Parameters:
        flags - This is a bit-mapped flags value indicating
        which resistors to enable or disable (see below).

    Return Values:
        TRUE if successful, FALSE if not.

    Side Effects:
        The resistors are enabled as requested.

    Remarks:
        Used for USB peripheral control to connect to or
        disconnect from the bus.  Otherwise, used for OTG
        SRP/HNP and host support.

 ********************************************************************/

/*
 This routine is implemented as a macro to a lower-level level routine.
 */
 #if defined(__18CXX)
    void USBHALControlUsbResistors( BYTE flags );
 #else
    #define USBHALControlUsbResistors OTGCORE_ControlUsbResistors
    void USBHALControlUsbResistors( BYTE flags );
#endif

/* USBHALControlUsbResistors flags */
#define USB_HAL_PULL_UP_D_PLUS  0x80    // Pull D+ line high
#define USB_HAL_PULL_UP_D_MINUS 0x40    // Pull D- line high
#define USB_HAL_PULL_DN_D_PLUS  0x20    // Pull D+ line low
#define USB_HAL_PULL_DN_D_MINUS 0x10    // Pull D- line low
/*
 The following are defined for convenience:
 */
#define USB_HAL_DEV_CONN_FULL_SPD       USB_HAL_PULL_UP_D_PLUS
#define USB_HAL_DEV_CONN_LOW_SPD        USB_HAL_PULL_UP_D_MINUS
#define USB_HAL_DEV_DISCONNECT          0


/*
 To Do: Define a method to check for SE0 & a way to send a reset (SE0).
 */
```

```
/*********************************************************************
    Function:
       BOOL USBHALSessionIsValid( void )

    Description:
        This routine determines if there is currently a valid
        USB session or not.

    Precondition:
        USBInitialize must have been called to initialize the
        USB SW stack.

    Parameters:
        None

    Return Values:
        TRUE if the session is currently valid, FALSE if not.

    Remarks:
        Only used for host and OTG support.

 ********************************************************************/
BOOL USBHALSessionIsValid( void );


/*********************************************************************
    Function:
        USBHALControlBusPower

    Description:
        This routine provides a bitmap of the most recent
        error conditions to occur.

    Precondition:
        USBInitialize must have been called to initialize the
        USB SW stack.

    Parameters:
        cmd - Identifies desired command (see below).

    Return Values:
        TRUE if successful, FALSE if not.

    Remarks:
        Only used for host and OTG support.

 ********************************************************************/
BOOL USBHALControlBusPower( BYTE cmd );

/* USBHALControlBusPower Commands */
#define USB_VBUS_DISCHARGE  0       // Dicharge Vbus via resistor
#define USB_VBUS_CHARGE     1       // Charge Vbus via resistor
#define USB_VBUS_POWER_ON   3       // Supply power to Vbus
#define USB_VBUS_POWER_OFF  4       // Do not supply power to Vbus
/*
 Note: All commands except USB_VBUS_POWER_ON imply that this device
 does not actively supply power to Vbus.
 */


/*********************************************************************
    Function:
        unsigned long USBHALGetLastError( void )

    Description:
```

```
        This routine provides a bitmap of the most recent
        error conditions to occur.

    Precondition:
        USBInitialize must have been called to initialize the
        USB SW stack.

    Parameters:
        None

    Return Values:
        Bitmap indicating the most recent error condition(s).

    Side Effect:
        Error record is cleared.

    Remarks:
        Although record of the error state is cleared, nothing
        is done to fix the condition or recover from the
        error.  The client must take appropriate steps.

 *****************************************************************************/

unsigned long USBHALGetLastError( void );

/*
 USBHALGetLastError Error Bits.
 */
#define USBHAL_PID_ERR  0x00000001  // Packet ID Error
#define USBHAL_CRC5     0x00000002  // (Host) Token CRC5 check failed
#define USBHAL_HOST_EOF 0x00000002  // (Host) EOF not reached before next SOF
#define USBHAL_CRC16    0x00000004  // Data packet CRC error
#define USBHAL_DFN8     0x00000008  // Data field size not n*8 bits
#define USBHAL_BTO_ERR  0x00000010  // Bus turn-around timeout
#define USBHAL_DMA_ERR  0x00000020  // DMA error, unable to read/write memory
#define USBHAL_BTS_ERR  0x00000080  // Bit-stuffing error
#define USBHAL_XFER_ID  0x00000100  // Unable to identify transfer EP
#define USBHAL_NO_EP    0x00000200  // Invalid endpoint number
#define USBHAL_DMA_ERR2 0x00000400  // Error starting DMA transaction


/*****************************************************************************
    Function:
        void USBHALHandleBusEvent ( void )

    Description:
        This routine checks the USB for any events that may
        have occured and handles them appropriately.  It may
        be called directly to poll the USB and handle events
        or it may be called in response to an interrupt.

    Precondition:
        USBInitialize must have been called to initialize the
        USB SW stack.

    Parameters:
        None

    Return Values:
        None

    Side Effects:
        Depend on the event that may have occured.

    Remarks:
        None
```

```
 ********************************************************************/

void USBHALHandleBusEvent ( void );


/*********************************************************************
    Function:
        BOOL USBHALStallPipe( TRANSFER_FLAGS pipe )

    Description:
        This routine stalls the given endpoint.

    Preconditions:
        USBHALInitialize must have been called to initialize
        the USB HAL.

    Parameters:
        pipe -  Uses the TRANSFER_FLAGS (see USBCommon.h) format to
                identify the endpoint and direction making up the
                pipe to stall.

        Note: Only ep_num and direction fields are required.

    Return Values:
        TRUE if able to stall endpoint, FALSE if not.

    Side Effects:
        The endpoint will stall if additional data transfer is
        attempted.
        Given endpoint has been stalled.

    Remarks:
        Starting another data transfer automatically
        "un-stalls" the endpoint.

 ********************************************************************/
/*
 Note: This function is implemented as a macro, calling directly into
 an internal HAL routine.
 */

#define USBHALStallPipe OTGCORE_StallPipe

BOOL USBHALStallPipe( TRANSFER_FLAGS pipe );


/*********************************************************************
    Function:
        BOOL USBHALUnstallPipe( TRANSFER_FLAGS pipe )

    Description:
        This routine clears the stall condition for the given pipe.

    PreCondition:
        Assumes OTGCORE_DeviceEnable has been called and
        OTGCORE_StallPipe has been called on the given pipe.

    Parameters:
        pipe -  Uses the TRANSFER_FLAGS (see USBCommon.h) format to
                identify the endpoint and direction making up the
                pipe to unstall.

    Return Values:
        TRUE if able to stall the pipe, FALSE if not.

    Side Effects:
        The BSTALL and UOWN bits (and all other control bits) in
```

```
        the BDT for the given pipe will be cleared.

    Remarks:
        None

 ***************************************************************************/
/*
 Note: This function is implemented as a macro, calling directly into
 an internal HAL routine.
 */

#define USBHALUnstallPipe OTGCORE_UnstallPipe

BOOL USBHALUnstallPipe( TRANSFER_FLAGS pipe );


/*****************************************************************************
    Function:
        USBHALGetStalledEndpoints

    Description:
        This function returns a 16-bit bitmapped value with a
        bit set in the position of any endpoint that is stalled
        (i.e. if endpoint 0 is stalled then bit 0 is set, if
        endpoint 1 is stalled then bit 1 is set, etc.).

    Preconditions:
        USBHALInitialize must have been called to initialize
        the USB HAL.

    Parameters:
        None

    Return Values:
        Bitmap of the currently stalled endpoints (see overview).

    Remarks:
        None
 *****************************************************************************/

/*
 Note: This function is implemented as a macro, calling directly into
 a HAL routine.
 */

#define USBHALGetStalledEndpoints OTGCORE_GetStalledEndpoints

UINT16 USBHALGetStalledEndpoints ( void );


/*****************************************************************************
    Function:
        BOOL USBHALFlushPipe( TRANSFER_FLAGS pipe )

    Description:
        This routine clears any pending transfers on the given
        pipe.

    Preconditions:
        USBHALInitialize must have been called to initialize the
        USB HAL.

        The caller must ensure that there is no possible way for
        hardware to be currently accessing the pipe (see notes).

    Parameters:
        pipe -  Uses the TRANSFER_FLAGS (see USBCommon.h) format to
```

```
                identify the endpoint and direction making up the
                pipe to flush.

    Return Values:
        TRUE if successful, FALSE if not.

    Side Effects:
        Transfer data for this pipe has been zero'd out.

    Remarks:
        This routine ignores the normal HW protocol for ownership
        of the pipe data and flushes the pipe, even if it is in
        process.  Thus, the caller must ensure that data transfer
        cannot be in process.  This situation occurs when a
        transfer has been terminated early by the host.
 *****************************************************************************/

BOOL USBHALFlushPipe( TRANSFER_FLAGS pipe );


/*****************************************************************************
    Function:
        USBHALTransferData

    Description:
        This routine prepares to transfer data on the USB.
        If the system is in device mode, the actual transfer
        will not occur until the host peforms an OUT request
        to the given endpoint.  If the system is in host mode,
        the transfer will not start until the token has been
        sent on the bus.

    Preconditions:
        1. USBHALInitialize must have been called to
           initialize the USB HAL.
        2. The endpoint through which the data will be
           transferred must be configured as appropriate by a
           call to USBHALSetEpConfiguration.
        3. The bus must have been enumerated (either as a host
           or device).  Except for EP0 transfers.

    Parameters:
        flags - Flags consists of the endpoint number OR'd
                with one or more flags indicating transfer
                direction and such (see "Data Transfer
                Macros" in USBCommon.h):

                  7 6 5 4 3 2 1 0 - Description
                  | | | | \_____/
                  | | | |    +----- Endpoint Number
                  | | | +---------- Short or zero-size pkt
                  | | +------------ Data Toggle 0/1
                  | +-------------- Force Data Toggle
                  +---------------- 1=Transmit/0=Receive

        buffer      Address of the buffer to receive data.

        size        Number of bytes of data to transfer.

    Return Values:
        TRUE if the HAL was able to successfully start the
        data transfer, FALSE if not.

    Side Effects:
        The HAL has prepared to transfer the data on the USB.

    Ramarks:
```

```
        The HAL will continue the data transfer, keeping track
        of the buffer address, data remaining, and ping-pong
        buffer details internally when USBHALHandleBusEvent is
        called (either polled or in response to an interrupt).
        The caller will receive notification that the transfer
        has completed when the EVT_XFER event is passed into
        the USBHALBusEventCallout call-out function.

 *****************************************************************************/

BOOL USBHALTransferData ( TRANSFER_FLAGS    flags,
                          void             *buffer,
                          unsigned int      size      );


/*****************************************************************************
    Function:
        USBHALSetEpConfiguration

    Description:
        This routine allows the caller to configure various
        options (see "Flags for USBHALSetEpConfiguration",
        below) and set the behavior for the given endpoint.

    Precondition:
        USBHALInitialize has been called.

    Parameters:
        ep_num - Number of endpoint to configur, Must be
                 (ep_num >=0) && (ep_num <= USB_DEV_HIGHEST_EP_NUMBER)
                  max_pkt_size Size of largest packet this enpoint can
                  transfer.

        flags - Configuration flags (see below)

    Return Values:
        TRUE if successful, FALSE if not.

    Side Effects:
        The endpoint has been configured as desired.

    Remarks:
        The base address and size of the buffer is not set by
        this routine.  Those features of an endpoint are
        dynamically managed by the USBHALTransferData routine.
        An endpoint can be "de-configured" by setting its max
        packet size to 0.  When doing this, you should also
        set all flags to 0.
 *****************************************************************************/

BOOL USBHALSetEpConfiguration ( BYTE ep_num, UINT16 max_pkt_size, UINT16 flags );

/* Flags for USBHALSetEpConfiguration */
#if defined(__18CXX)
    #define USB_HAL_TRANSMIT    0x0400  // Enable EP for transmitting data
    #define USB_HAL_RECEIVE     0x0200  // Enable EP for receiving data
    #define USB_HAL_HANDSHAKE   0x1000  // Enable EP to give ACK/NACK (non isoch)

    #define USB_HAL_NO_INC      0x0010  // Use for DMA to another device FIFO
    #define USB_HAL_HW_KEEPS    0x0020  // Cause HW to keep EP
#else
    #define USB_HAL_TRANSMIT    0x0400  // Enable EP for transmitting data
    #define USB_HAL_RECEIVE     0x0800  // Enable EP for receiving data
    #define USB_HAL_HANDSHAKE   0x0100  // Enable EP to give ACK/NACK (non isoch)

    /* Does not work, Fix this if needed. 3/1/07 - Bud
    #define USB_HAL_NO_INC      0x0010  // Use for DMA to another device FIFO
```

```
    #define USB_HAL_HW_KEEPS    0x0020  // Cause HW to keep EP
    */
    #define USB_HAL_ALLOW_HUB   0x8000  // (host only) Enable low-spd hub support
    #define USB_HAL_NO_RETRY    0x4000  // (host only) disable auto-retry on NACK
#endif


/*********************************************************************
    Function:
        USBHALInitialize

    Description:
        This call performs the basic initialization of the USB
        HAL.  This routine must be called before any of the
        other HAL interface routines are called.

    Precondition:
        The system has been initialized.

    Paramters:
        flags -  Initialization flags

    Return Values:
        TRUE if successful, FALSE if not.

    Side Effects:
        The USB HAL SW stack was initialized.

    Remarks:
        This routine can be called to reset the controller.

 ********************************************************************/

BOOL USBHALInitialize ( unsigned long flags );

#else   // defined(USB_SUPPORT_HOST) || defined(USB_SUPPORT_OTG)
    #if defined(__18CXX)
        #include "USB\usb_hal_pic18.h"
    #elif defined(__C30__)
        #include "USB\usb_hal_pic24.h"
    #elif defined(__PIC32MX__)
        #include "USB\usb_hal_pic32.h"
    #endif
#endif  // defined(USB_SUPPORT_HOST) || defined(USB_SUPPORT_OTG)
#endif  // _USB_HAL_H_
/*********************************************************************
 * EOF
 */
```

6.1.14. usb_hal_pic18.h

```
/********************************************************************************

 File Description:

 This file defines the interface to the USB hardware abstraction layer.

 * Filename:    usb_hal_pic18.h
 Dependencies:  See INCLUDES section
 Processor:     Use this header file when using this firmware with PIC18 USB
                microcontrollers
 Hardware:
 Complier:      Microchip C18 (for PIC18)
 Company:       Microchip Technology, Inc.

 Software License Agreement:

 The software supplied herewith by Microchip Technology Incorporated
 (the “Company”) for its PIC® Microcontroller is intended and
 supplied to you, the Company’s customer, for use solely and
 exclusively on Microchip PIC Microcontroller products. The
 software is owned by the Company and/or its supplier, and is
 protected under applicable copyright laws. All rights are reserved.
 Any use in violation of the foregoing restrictions may subject the
 user to criminal sanctions under applicable laws, as well as to
 civil liability for the breach of the terms and conditions of this
 license.

 THIS SOFTWARE IS PROVIDED IN AN “AS IS” CONDITION. NO WARRANTIES,
 WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
 TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
 IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
 CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

********************************************************************

 Change History:


 ********************************************************************************/
//DOM-IGNORE-BEGIN
/********************************************************************************

    USB Hardware Abstraction Layer (HAL)  (Header File)

Summary:
    This file abstracts the hardware interface.  The USB stack firmware can be
    compiled to work on different USB microcontrollers, such as PIC18 and PIC24.
    The USB related special function registers and bit names are generally very
    similar between the device families, but small differences in naming exist.

Description:
    This file abstracts the hardware interface.  The USB stack firmware can be
    compiled to work on different USB microcontrollers, such as PIC18 and PIC24.
    The USB related special function registers and bit names are generally very
    similar between the device families, but small differences in naming exist.

    In order to make the same set of firmware work accross the device families,
    when modifying SFR contents, a slightly abstracted name is used, which is
    then "mapped" to the appropriate real name in the usb_hal_picxx.h header.

    Make sure to include the correct version of the usb_hal_picxx.h file for
    the microcontroller family which will be used.

    This file is located in the "\<Install Directory\>\\Microchip\\Include\\USB"
```

```
    directory.

    When including this file in a new project, this file can either be
    referenced from the directory in which it was installed or copied
    directly into the user application folder. If the first method is
    chosen to keep the file located in the folder in which it is installed
    then include paths need to be added so that the library and the
    application both know where to reference each others files. If the
    application folder is located in the same folder as the Microchip
    folder (like the current demo folders), then the following include
    paths need to be added to the application's project:

    ..\\Include

    ..\\..\\Include

    ..\\..\\MicrochipInclude

    ..\\..\\\<Application Folder\>

    ..\\..\\..\\\<Application Folder\>

    If a different directory structure is used, modify the paths as
    required. An example using absolute paths instead of relative paths
    would be the following:

    C:\\Microchip Solutions\\Microchip\\Include

    C:\\Microchip Solutions\\My Demo Application

*****************************************************************************/
//DOM-IGNORE-END
/*****************************************************************
    Function:
        void USBPowerModule(void)

    Description:
        This macro is used to power up the USB module if required<br>
        PIC18: defines as nothing<br>
        PIC24: defines as U1PWRCbits.USBPWR = 1;<br>

    Parameters:
        None

    Return Values:
        None

    Remarks:
        None

  *****************************************************************/
#define USBPowerModule()

/*****************************************************************
    Function:
        void USBModuleDisable(void)

    Description:
        This macro is used to disable the USB module

    Parameters:
        None

    Return Values:
        None

    Remarks:
```

```
        None

  *****************************************************************/
#define USBModuleDisable() {\
    UCON = 0;\
    UIE = 0;\
    USBDeviceState = DETACHED_STATE;\
}

/*********************************************************************
    Function:
        USBSetBDTAddress(addr)

    Description:
        This macro is used to power up the USB module if required

    Parameters:
        None

    Return Values:
        None

    Remarks:
        None

  *****************************************************************/
#define USBSetBDTAddress(addr)

#if defined(USB_ENABLE_SOF_HANDLER)
    #define USB_SOF_INTERRUPT 0x40
#else
    #define USB_SOF_INTERRUPT 0x00
#endif
#if defined(USB_ENABLE_ERROR_HANDLER)
    #define USB_ERROR_INTERRUPT 0x02
#else
    #define USB_ERROR_INTERRUPT 0x02
#endif

//STALLIE, IDLEIE, TRNIE, and URSTIE are all enabled by default and are required
#define USBEnableInterrupts() {RCONbits.IPEN = 1;UIE = 0x39 | USB_SOF_INTERRUPT |
USB_ERROR_INTERRUPT; IPR2bits.USBIP = 1;PIE2bits.USBIE = 1;INTCONbits.GIEH = 1;}
#if defined(USB_INTERRUPT)
    #define USBMaskInterrupts() {PIE2bits.USBIE = 0;}
    #define USBUnmaskInterrupts() {PIE2bits.USBIE = 1;}
#else
    #define USBMaskInterrupts()
    #define USBUnmaskInterrupts()
#endif

#define ENDPOINT_MASK 0b01111000


#define USBPingPongBufferReset UCONbits.PPBRST

#define USBTransactionCompleteIE UIEbits.TRNIE
#define USBTransactionCompleteIF UIRbits.TRNIF
#define USBTransactionCompleteIFReg (BYTE*)&UIR
#define USBTransactionCompleteIFBitNum 3

#define USBResetIE  UIEbits.URSTIE
#define USBResetIF  UIRbits.URSTIF
#define USBResetIFReg (BYTE*)&UIR
#define USBResetIFBitNum 0

#define USBIdleIE UIEbits.IDLEIE
#define USBIdleIF UIRbits.IDLEIF
```

```
#define USBIdleIFReg (BYTE*)&UIR
#define USBIdleIFBitNum 4

#define USBActivityIE UIEbits.ACTVIE
#define USBActivityIF UIRbits.ACTVIF
#define USBActivityIFReg (BYTE*)&UIR
#define USBActivityIFBitNum 2

#define USBSOFIE UIEbits.SOFIE
#define USBSOFIF UIRbits.SOFIF
#define USBSOFIFReg (BYTE*)&UIR
#define USBSOFIFBitNum 6

#define USBStallIE UIEbits.STALLIE
#define USBStallIF UIRbits.STALLIF
#define USBStallIFReg (BYTE*)&UIR
#define USBStallIFBitNum 5

#define USBErrorIE UIEbits.UERRIE
#define USBErrorIF UIRbits.UERRIF
#define USBErrorIFReg (BYTE*)&UIR
#define USBErrorIFBitNum 1

#define USBSE0Event UCONbits.SE0
#define USBSuspendControl UCONbits.SUSPND
#define USBPacketDisable UCONbits.PKTDIS
#define USBResumeControl UCONbits.RESUME

#define U1ADDR UADDR
#define U1IE UIE
#define U1IR UIR
#define U1EIR UEIR
#define U1EIE UEIE
#define U1CON UCON
#define U1EP0 UEP0
#define U1CONbits UCONbits
#define U1EP1 UEP1
#define U1CNFG1 UCFG
#define U1STAT USTAT
#define U1EP0bits UEP0bits

/* Buffer Descriptor Status Register Initialization Parameters */
#define _BSTALL     0x04        //Buffer Stall enable
#define _DTSEN      0x08        //Data Toggle Synch enable
#define _INCDIS     0x10        //Address increment disable
#define _KEN        0x20        //SIE keeps buff descriptors enable
#define _DAT0       0x00        //DATA0 packet expected next
#define _DAT1       0x40        //DATA1 packet expected next
#define _DTSMASK    0x40        //DTS Mask
#define _USIE       0x80        //SIE owns buffer
#define _UCPU       0x00        //CPU owns buffer

#define _STAT_MASK  0xFF

/* BDT entry structure definition */
typedef union _BD_STAT
{
    BYTE Val;
    struct{
        //If the CPU owns the buffer then these are the values
        unsigned BC8:1;         //bit 8 of the byte count
        unsigned BC9:1;         //bit 9 of the byte count
        unsigned BSTALL:1;      //Buffer Stall Enable
        unsigned DTSEN:1;       //Data Toggle Synch Enable
        unsigned INCDIS:1;      //Address Increment Disable
        unsigned KEN:1;         //BD Keep Enable
        unsigned DTS:1;         //Data Toggle Synch Value
```

```
        unsigned UOWN:1;        //USB Ownership
    };
    struct{
        //if the USB module owns the buffer then these are
        // the values
        unsigned BC8:1;         //bit 8 of the byte count
        unsigned BC9:1;         //bit 9 of the byte count
        unsigned PID0:1;        //Packet Identifier
        unsigned PID1:1;
        unsigned PID2:1;
        unsigned PID3:1;
        unsigned :1;
        unsigned UOWN:1;        //USB Ownership
    };
    struct{
        unsigned :2;
        unsigned PID:4;         //Packet Identifier
        unsigned :2;
    };
} BD_STAT;                      //Buffer Descriptor Status Register

// BDT Entry Layout
typedef union __BDT
{
    struct
    {
        BD_STAT STAT;
        BYTE CNT;
        BYTE ADRL;                      //Buffer Address Low
        BYTE ADRH;                      //Buffer Address High
    };
    struct
    {
        unsigned :8;
        unsigned :8;
        BYTE* ADR;                      //Buffer Address
    };
    DWORD Val;
    BYTE v[4];
} BDT_ENTRY;

#define USTAT_EP0_PP_MASK   ~0x02
#define USTAT_EP_MASK       0x7E
#define USTAT_EP0_OUT       0x00
#define USTAT_EP0_OUT_EVEN  0x00
#define USTAT_EP0_OUT_ODD   0x02

#define USTAT_EP0_IN        0x04
#define USTAT_EP0_IN_EVEN   0x04
#define USTAT_EP0_IN_ODD    0x06

typedef union
{
    WORD UEP[16];
} _UEP;

#define UEP_STALL 0x0001

#define USB_BDT_ADDRESS 0x400
#if defined(USB_POLLING)
    #define USB_VOLATILE
#else
    #define USB_VOLATILE volatile
#endif

typedef union _POINTER
{
```

```
    struct
    {
        BYTE bLow;
        BYTE bHigh;
        //byte bUpper;
    };
    WORD _word;                         // bLow & bHigh

    //pFunc _pFunc;                       // Usage: ptr.pFunc(); Init: ptr.pFunc
= &<Function>;

    BYTE* bRam;                         // Ram byte pointer: 2 bytes pointer
pointing
                                        // to 1 byte of data
    WORD* wRam;                         // Ram word poitner: 2 bytes poitner
pointing
                                        // to 2 bytes of data

    ROM BYTE* bRom;                     // Size depends on compiler setting
    ROM WORD* wRom;
    //rom near byte* nbRom;               // Near = 2 bytes pointer
    //rom near word* nwRom;
    //rom far byte* fbRom;                // Far = 3 bytes pointer
    //rom far word* fwRom;
} POINTER;

#define USB_PING_PONG__NO_PING_PONG         0x00    //0b00
#define USB_PING_PONG__EP0_OUT_ONLY         0x01    //0b01
#define USB_PING_PONG__FULL_PING_PONG       0x02    //0b10
#define USB_PING_PONG__ALL_BUT_EP0          0x03    //0b11

    //******** Depricated: v2.2 - will be removed at some point of time ***
    #define _LS         0x00            // Use Low-Speed USB Mode
    #define _FS         0x04            // Use Full-Speed USB Mode
    #define _TRINT      0x00            // Use internal transceiver
    #define _TREXT      0x08            // Use external transceiver
    #define _PUEN       0x10            // Use internal pull-up resistor
    #define _OEMON      0x40            // Use SIE output indicator
    //*************************************************************************

    #define USB_PULLUP_ENABLE 0x10
    #define USB_PULLUP_DISABLED 0x00

    #define USB_INTERNAL_TRANSCEIVER 0x00
    #define USB_EXTERNAL_TRANSCEIVER 0x08

    #define USB_FULL_SPEED 0x04
    #define USB_LOW_SPEED  0x00

#define ConvertToPhysicalAddress(a) a
#define USBClearUSBInterrupt() PIR2bits.USBIF = 0;
```

6.1.15. usb_function_generic.h

```
/********************************************************************
  File Information:
      FileName:       usb_function_generic.h
      Dependencies:   See INCLUDES section
      Processor:      PIC18 or PIC24 USB Microcontrollers
      Hardware:       The code is natively intended to be used on the following
                      hardware platforms: PICDEM™ FS USB Demo Board,
                      PIC18F87J50 FS USB Plug-In Module, or
                      Explorer 16 + PIC24 USB PIM.  The firmware may be
                      modified for use on other USB platforms by editing the
                      HardwareProfile.h file.
      Complier:       Microchip C18 (for PIC18) or C30 (for PIC24)
      Company:        Microchip Technology, Inc.

      Software License Agreement:

      The software supplied herewith by Microchip Technology Incorporated
      (the “Company”) for its PIC® Microcontroller is intended and
      supplied to you, the Company’s customer, for use solely and
      exclusively on Microchip PIC Microcontroller products. The
      software is owned by the Company and/or its supplier, and is
      protected under applicable copyright laws. All rights are reserved.
      Any use in violation of the foregoing restrictions may subject the
      user to criminal sanctions under applicable laws, as well as to
      civil liability for the breach of the terms and conditions of this
      license.

      THIS SOFTWARE IS PROVIDED IN AN “AS IS” CONDITION. NO WARRANTIES,
      WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
      TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
      PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
      IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
      CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

       Change History:
       Rev         Description

  Summary:
    This file contains all of functions, macros, definitions, variables,
    datatypes, etc. that are required for usage with vendor class function
    drivers. This file should be included in projects that use vendor class
    \function drivers. This file should also be included into the
    usb_descriptors.c file and any other user file that requires access to
    vendor class interfaces.



    This file is located in the "\<Install
    Directory\>\\Microchip\\Include\\USB" directory.
  Description:
    USB Vender Class Custom Driver File

    This file contains all of functions, macros, definitions, variables,
    datatypes, etc. that are required for usage with vendor class function
    drivers. This file should be included in projects that use vendor class
    \function drivers. This file should also be included into the
    usb_descriptors.c file and any other user file that requires access to
    vendor class interfaces.

    This file is located in the "\<Install
    Directory\>\\Microchip\\Include\\USB" directory.

    When including this file in a new project, this file can either be
    referenced from the directory in which it was installed or copied
    directly into the user application folder. If the first method is
```

```
    chosen to keep the file located in the folder in which it is installed
    then include paths need to be added so that the library and the
    application both know where to reference each others files. If the
    application folder is located in the same folder as the Microchip
    folder (like the current demo folders), then the following include
    paths need to be added to the application's project:

    ..\\Include

    ..\\..\\Include

    ..\\..\\Microchip\\Include

    ..\\..\\\<Application Folder\>

    ..\\..\\..\\\<Application Folder\>

    If a different directory structure is used, modify the paths as
    required. An example using absolute paths instead of relative paths
    would be the following:

    C:\\Microchip Solutions\\Microchip\\Include

    C:\\Microchip Solutions\\My Demo Application
  ********************************************************************************/
#ifndef USBGEN_H
#define USBGEN_H

/** I N C L U D E S **************************************************************/

#include "GenericTypeDefs.h"
#include "usb/usb_config.h"

/** D E F I N I T I O N S ********************************************************/

/*********************************************************************************
    Macro:
        (bit) mUSBGenRxIsBusy(void)

    Description:
        This macro is used to check if the OUT endpoint is
        busy (owned by SIE) or not.
        Typical Usage: if(mUSBGenRxIsBusy())

    PreCondition:
        None

    Parameters:
        None

    Return Values:
        None

    Remarks:
        None

 *********************************************************************************/

/*********************************************************************************
    Macro:
        (bit) mUSBGenTxIsBusy(void)

    Description:
        This macro is used to check if the IN endpoint is
        busy (owned by SIE) or not.
        Typical Usage: if(mUSBGenTxIsBusy())
```

```
    PreCondition:
        None

    Parameters:
        None

    Return Values:
        None

    Remarks:
        None

 ********************************************************************************/

/********************************************************************************
    Macro:
        byte mUSBGenGetRxLength(void)

    Description:
        mUSBGenGetRxLength is used to retrieve the number of bytes
        copied to user's buffer by the most recent call to
        USBGenRead function.

    PreCondition:
        None

    Parameters:
        None

    Return Values:
        mUSBGenGetRxLength returns usbgen_rx_len

    Remarks:
        None

 ********************************************************************************/

/** S T R U C T U R E S ********************************************************/

/** E X T E R N S **************************************************************/

/** P U B L I C  P R O T O T Y P E S ******************************************/

/*****************************************************************
    Function:
        USB_HANDLE USBGenWrite(BYTE ep, BYTE* data, WORD len)

    Summary:
        Sends the specified data out the specified endpoint

    Description:
        This function sends the specified data out the specified
        endpoint and returns a handle to the transfer information.

        Typical Usage:
        <code>
        //make sure that the last transfer isn't busy by checking the handle
        if(!USBHandleBusy(USBGenericInHandle))
        {
            //Send the data contained in the INPacket[] array out on
            //  endpoint USBGEN_EP_NUM
            USBGenericInHandle =
USBGenWrite(USBGEN_EP_NUM,(BYTE*)&INPacket[0],sizeof(INPacket));
        }
        </code>

    PreCondition:
```

```
        None

    Parameters:
        ep - the endpoint you want to send the data out of
        data - pointer to the data that you wish to send
        len - the length of the data that you wish to send

    Return Values:
        USB_HANDLE - a handle for the transfer.  This information
        should be kept to track the status of the transfer

    Remarks:
        None

 ***************************************************************/
#define USBGenWrite(ep,data,len) USBTxOnePacket(ep,data,len)

/***************************************************************
    Function:
        USB_HANDLE USBGenRead(BYTE ep, BYTE* data, WORD len)

    Summary:
        Receives the specified data out the specified endpoint

    Description:
        Receives the specified data out the specified endpoint.

        Typical Usage:
        <code>
        //Read 64-bytes from endpoint USBGEN_EP_NUM, into the OUTPacket array.
        //  Make sure to save the return handle so that we can check it later
        //  to determine when the transfer is complete.
        if(!USBHandleBusy(USBOutHandle))
        {
            USBOutHandle = USBGenRead(USBGEN_EP_NUM,(BYTE*)&OUTPacket,64);
        }
        </code>

    PreCondition:
        None

    Parameters:
        ep - the endpoint you want to receive the data into
        data - pointer to where the data will go when it arrives
        len - the length of the data that you wish to receive

    Return Values:
        USB_HANDLE - a handle for the transfer.  This information
        should be kept to track the status of the transfer

    Remarks:
        None

 ***************************************************************/
#define USBGenRead(ep,data,len) USBRxOnePacket(ep,data,len)

#endif //USBGEN_H
```

6.1.16. usb_device.h

```
/********************************************************************
 FileName:          usb_device.h
 Dependencies:      See INCLUDES section
 Processor:         PIC18 or PIC24 USB Microcontrollers
 Hardware:          The code is natively intended to be used on the following
                        hardware platforms: PICDEM™ FS USB Demo Board,
                        PIC18F87J50 FS USB Plug-In Module, or
                        Explorer 16 + PIC24 USB PIM.  The firmware may be
                        modified for use on other USB platforms by editing the
                        HardwareProfile.h file.
 Complier:   Microchip C18 (for PIC18) or C30 (for PIC24)
 Company:           Microchip Technology, Inc.

 Software License Agreement:

 The software supplied herewith by Microchip Technology Incorporated
 (the “Company”) for its PIC® Microcontroller is intended and
 supplied to you, the Company’s customer, for use solely and
 exclusively on Microchip PIC Microcontroller products. The
 software is owned by the Company and/or its supplier, and is
 protected under applicable copyright laws. All rights are reserved.
 Any use in violation of the foregoing restrictions may subject the
 user to criminal sanctions under applicable laws, as well as to
 civil liability for the breach of the terms and conditions of this
 license.

 THIS SOFTWARE IS PROVIDED IN AN “AS IS” CONDITION. NO WARRANTIES,
 WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
 TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
 IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
 CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

********************************************************************
 File Description:

 Change History:
  Rev   Date         Description
  1.0   11/19/2004   Initial release
  2.1   02/26/2007   Added "(" & ")" to EP definitions
                     Updated for simplicity and to use common
                     coding style
********************************************************************/
//DOM-IGNORE-BEGIN
/*********************************************************************

    USB Device header file

Summary:
    This file, with its associated C source file, provides the main substance of
    the USB device side stack.  These files will receive, transmit, and process
    various USB commands as well as take action when required for various events
    that occur on the bus.

Description:
    This file, with its associated C source file, provides the main substance of
    the USB device side stack.  These files will receive, transmit, and process
    various USB commands as well as take action when required for various events
    that occur on the bus.

    This file is located in the "\<Install Directory\>\\Microchip\\Include\\USB"
    directory.

    When including this file in a new project, this file can either be
    referenced from the directory in which it was installed or copied
```

```
    directly into the user application folder. If the first method is
    chosen to keep the file located in the folder in which it is installed
    then include paths need to be added so that the library and the
    application both know where to reference each others files. If the
    application folder is located in the same folder as the Microchip
    folder (like the current demo folders), then the following include
    paths need to be added to the application's project:

    ..\\Include

    ..\\..\\Include

    ..\\..\\MicrochipInclude

    ..\\..\\\<Application Folder\>

    ..\\..\\..\\\<Application Folder\>

    If a different directory structure is used, modify the paths as
    required. An example using absolute paths instead of relative paths
    would be the following:

    C:\\Microchip Solutions\\Microchip\\Include

    C:\\Microchip Solutions\\My Demo Application

********************************************************************************/

#ifndef USBDEVICE_H
#define USBDEVICE_H
//DOM-IGNORE-END

//#include "HardwareProfile.h"
#include "Compiler.h"
#include "USB/usb_ch9.h"
#include "USB/usb_hal.h"
#include "usb/usb_config.h" //This file needs to be included after the
                            //  usb_hal.h file to insure that the user
                            //  options are selected after the defines
                            //  are created.
/** DEFINITIONS ****************************************************/

/*********************************************************************
USB Endpoint Definitions
USB Standard EP Address Format: DIR:X:X:X:EP3:EP2:EP1:EP0
This is used in the descriptors. See usbcfg.c

NOTE: Do not use these values for checking against USTAT.
To check against USTAT, use values defined in usbd.h.
*********************************************************************/
#define _EP_IN      0x80
#define _EP_OUT     0x00
#define _EP01_OUT   0x01
#define _EP01_IN    0x81
#define _EP02_OUT   0x02
#define _EP02_IN    0x82
#define _EP03_OUT   0x03
#define _EP03_IN    0x83
#define _EP04_OUT   0x04
#define _EP04_IN    0x84
#define _EP05_OUT   0x05
#define _EP05_IN    0x85
#define _EP06_OUT   0x06
#define _EP06_IN    0x86
#define _EP07_OUT   0x07
#define _EP07_IN    0x87
#define _EP08_OUT   0x08
```

```
#define _EP08_IN    0x88
#define _EP09_OUT   0x09
#define _EP09_IN    0x89
#define _EP10_OUT   0x0A
#define _EP10_IN    0x8A
#define _EP11_OUT   0x0B
#define _EP11_IN    0x8B
#define _EP12_OUT   0x0C
#define _EP12_IN    0x8C
#define _EP13_OUT   0x0D
#define _EP13_IN    0x8D
#define _EP14_OUT   0x0E
#define _EP14_IN    0x8E
#define _EP15_OUT   0x0F
#define _EP15_IN    0x8F

/* Configuration Attributes */
#define _DEFAULT    (0x01<<7)       //Default Value (Bit 7 is set)
#define _SELF       (0x01<<6)       //Self-powered (Supports if set)
#define _RWU        (0x01<<5)       //Remote Wakeup (Supports if set)
#define _HNP         (0x01 << 1)    //HNP (Supports if set)
#define _SRP         (0x01)         //SRP (Supports if set)

/* Endpoint Transfer Type */
#define _CTRL       0x00            //Control Transfer
#define _ISO        0x01            //Isochronous Transfer
#define _BULK       0x02            //Bulk Transfer

#define _INTERRUPT        0x03            //Interrupt Transfer
#if defined(__18CXX) || defined(__C30__)
    #define _INT        0x03            //Interrupt Transfer
#endif

/* Isochronous Endpoint Synchronization Type */
#define _NS         (0x00<<2)       //No Synchronization
#define _AS         (0x01<<2)       //Asynchronous
#define _AD         (0x02<<2)       //Adaptive
#define _SY         (0x03<<2)       //Synchronous

/* Isochronous Endpoint Usage Type */
#define _DE         (0x00<<4)       //Data endpoint
#define _FE         (0x01<<4)       //Feedback endpoint
#define _IE         (0x02<<4)       //Implicit feedback Data endpoint

#define _ROM        USB_INPIPES_ROM
#define _RAM        USB_INPIPES_RAM

//These are the directional indicators used for the USBTransferOnePacket()
//  function.
#define OUT_FROM_HOST 0
#define IN_TO_HOST 1

/*****************************************************************
 * CTRL_TRF_SETUP: Every setup packet has 8 bytes.  This structure
 * allows direct access to the various members of the control
 * transfer.
 *****************************************************************/
typedef union __attribute__ ((packed)) _CTRL_TRF_SETUP
{
    /** Standard Device Requests ********************************/
    struct __attribute__ ((packed))
    {
        BYTE bmRequestType; //from table 9-2 of USB2.0 spec
        BYTE bRequest; //from table 9-2 of USB2.0 spec
        WORD wValue; //from table 9-2 of USB2.0 spec
        WORD wIndex; //from table 9-2 of USB2.0 spec
        WORD wLength; //from table 9-2 of USB2.0 spec
```

```
    };
    struct __attribute__ ((packed))
    {
        unsigned :8;
        unsigned :8;
        WORD_VAL W_Value; //from table 9-2 of USB2.0 spec, allows byte/bitwise
access
        WORD_VAL W_Index; //from table 9-2 of USB2.0 spec, allows byte/bitwise
access
        WORD_VAL W_Length; //from table 9-2 of USB2.0 spec, allows byte/bitwise
access
    };
    struct __attribute__ ((packed))
    {
        unsigned Recipient:5;   //Device,Interface,Endpoint,Other
        unsigned RequestType:2; //Standard,Class,Vendor,Reserved
        unsigned DataDir:1;     //Host-to-device,Device-to-host
        unsigned :8;
        BYTE bFeature;          //DEVICE_REMOTE_WAKEUP,ENDPOINT_HALT
        unsigned :8;
        unsigned :8;
        unsigned :8;
        unsigned :8;
        unsigned :8;
    };
    struct __attribute__ ((packed))
    {
        unsigned :8;
        unsigned :8;
        BYTE bDscIndex;         //For Configuration and String DSC Only
        BYTE bDescriptorType;   //Device,Configuration,String
        WORD wLangID;           //Language ID
        unsigned :8;
        unsigned :8;
    };
    struct __attribute__ ((packed))
    {
        unsigned :8;
        unsigned :8;
        BYTE_VAL bDevADR;       //Device Address 0-127
        BYTE bDevADRH;          //Must equal zero
        unsigned :8;
        unsigned :8;
        unsigned :8;
        unsigned :8;
    };
    struct __attribute__ ((packed))
    {
        unsigned :8;
        unsigned :8;
        BYTE bConfigurationValue; //Configuration Value 0-255
        BYTE bCfgRSD;             //Must equal zero (Reserved)
        unsigned :8;
        unsigned :8;
        unsigned :8;
        unsigned :8;
    };
    struct __attribute__ ((packed))
    {
        unsigned :8;
        unsigned :8;
        BYTE bAltID;            //Alternate Setting Value 0-255
        BYTE bAltID_H;          //Must equal zero
        BYTE bIntfID;           //Interface Number Value 0-255
        BYTE bIntfID_H;         //Must equal zero
        unsigned :8;
        unsigned :8;
```

```
    };
    struct __attribute__ ((packed))
    {
        unsigned :8;
        unsigned :8;
        unsigned :8;
        unsigned :8;
        BYTE bEPID;             //Endpoint ID (Number & Direction)
        BYTE bEPID_H;           //Must equal zero
        unsigned :8;
        unsigned :8;
    };
    struct __attribute__ ((packed))
    {
        unsigned :8;
        unsigned :8;
        unsigned :8;
        unsigned :8;
        unsigned EPNum:4;       //Endpoint Number 0-15
        unsigned :3;
        unsigned EPDir:1;       //Endpoint Direction: 0-OUT, 1-IN
        unsigned :8;
        unsigned :8;
        unsigned :8;
    };

    /** End: Standard Device Requests *****************************/

} CTRL_TRF_SETUP;

// Defintion of the PIPE structure
//  This structure is used to keep track of data that is sent out
//  of the stack automatically.
typedef struct __attribute__ ((packed))
{
    union __attribute__ ((packed))
    {
        //Various options of pointers that are available to
        // get the data from
        BYTE *bRam;
        ROM BYTE *bRom;
        WORD *wRam;
        ROM WORD *wRom;
    }pSrc;
    union __attribute__ ((packed))
    {
        struct __attribute__ ((packed))
        {
            //is this transfer from RAM or ROM?
            BYTE ctrl_trf_mem          :1;
            BYTE reserved              :5;
            //include a zero length packet after
            //data is done if data_size%ep_size = 0?
            BYTE includeZero           :1;
            //is this PIPE currently in use
            BYTE busy                  :1;
        }bits;
        BYTE Val;
    }info;
    WORD_VAL wCount;
}IN_PIPE;

#define CTRL_TRF_RETURN void
#define CTRL_TRF_PARAMS void
typedef struct __attribute__ ((packed))
{
    union __attribute__ ((packed))
```

```
    {
        //Various options of pointers that are available to
        // get the data from
        BYTE *bRam;
        WORD *wRam;
    }pDst;
    union __attribute__ ((packed))
    {
        struct __attribute__ ((packed))
        {
            BYTE reserved              :7;
            //is this PIPE currently in use
            BYTE busy                  :1;
        }bits;
        BYTE Val;
    }info;
    WORD_VAL wCount;
    CTRL_TRF_RETURN (*pFunc)(CTRL_TRF_PARAMS);
}OUT_PIPE;

//Various options for setting the PIPES
#define USB_INPIPES_ROM            0x00     //Data comes from RAM
#define USB_INPIPES_RAM            0x01     //Data comes from ROM
#define USB_INPIPES_BUSY           0x80     //The PIPE is busy
#define USB_INPIPES_INCLUDE_ZERO   0x40     //include a trailing zero packet
#define USB_INPIPES_NO_DATA        0x00     //no data to send
#define USB_INPIPES_NO_OPTIONS     0x00     //no options set

#define USB_EP0_ROM            USB_INPIPES_ROM
#define USB_EP0_RAM            USB_INPIPES_RAM
#define USB_EP0_BUSY           USB_INPIPES_BUSY
#define USB_EP0_INCLUDE_ZERO   USB_INPIPES_INCLUDE_ZERO
#define USB_EP0_NO_DATA        USB_INPIPES_NO_DATA
#define USB_EP0_NO_OPTIONS     USB_INPIPES_NO_OPTIONS

/*********************************************************************
 * Standard Request Codes
 * USB 2.0 Spec Ref Table 9-4
 ********************************************************************/
#define GET_STATUS  0
#define CLR_FEATURE 1
#define SET_FEATURE 3
#define SET_ADR     5
#define GET_DSC     6
#define SET_DSC     7
#define GET_CFG     8
#define SET_CFG     9
#define GET_INTF    10
#define SET_INTF    11
#define SYNCH_FRAME 12

/* Section: Standard Feature Selectors */
#define DEVICE_REMOTE_WAKEUP    0x01
#define ENDPOINT_HALT           0x00

/* Section: USB Device States - To be used with [BYTE usb_device_state] */

/* Detached is the state in which the device is not attached to the bus.  When
in the detached state a device should not have any pull-ups attached to either
the D+ or D- line.  This defintions is a return value of the
function USBGetDeviceState() */
#define DETACHED_STATE          0x00
/* Attached is the state in which the device is attached ot the bus but the
hub/port that it is attached to is not yet configured. This defintions is a
return value of the function USBGetDeviceState() */
#define ATTACHED_STATE          0x01
/* Powered is the state in which the device is attached to the bus and the
```

```
hub/port that it is attached to is configured. This defintions is a return
value of the function USBGetDeviceState() */
#define POWERED_STATE           0x02
/* Default state is the state after the device receives a RESET command from
the host. This defintions is a return value of the function USBGetDeviceState()
 */
#define DEFAULT_STATE           0x04
/* Address pending state is not an official state of the USB defined states.
This state is internally used to indicate that the device has received a
SET_ADDRESS command but has not received the STATUS stage of the transfer yet.
The device is should not switch addresses until after the STATUS stage is
complete.  This defintions is a return value of the function
USBGetDeviceState() */
#define ADR_PENDING_STATE       0x08
/* Address is the state in which the device has its own specific address on the
bus. This defintions is a return value of the function USBGetDeviceState().*/
#define ADDRESS_STATE           0x10
/* Configured is the state where the device has been fully enumerated and is
operating on the bus.  The device is now allowed to excute its application
specific tasks.  It is also allowed to increase its current consumption to the
value specified in the configuration descriptor of the current configuration.
This defintions is a return value of the function USBGetDeviceState(). */
#define CONFIGURED_STATE        0x20

/* UCFG Initialization Parameters */
#if defined(__18CXX)
    #if defined(UCFG_VAL)
        //This has been depricated in v2.2 - it will be removed in future
releases
        #define SetConfigurationOptions() {U1CNFG1 = UCFG_VAL;}   //UCFG_VAL
defined in usb_config.h
    #else
        #define SetConfigurationOptions()   {\
                                                U1CNFG1 = USB_PULLUP_OPTION |
USB_TRANSCEIVER_OPTION | USB_SPEED_OPTION | USB_PING_PONG_MODE;\
                                            }
    #endif

#elif defined(__C30__)
   #if defined(UCFG_VAL)
        //This has been depricated in v2.2 - it will be removed in future
releases
        #define SetConfigurationOptions() {U1CNFG1 = UCFG_VAL;}   //UCFG_VAL
defined in usb_config.h
    #else
        #define SetConfigurationOptions()   {\
                                                U1CNFG1 = USB_PING_PONG_MODE;\
                                                U1CNFG2 = USB_TRANSCEIVER_OPTION
| USB_SPEED_OPTION | USB_PULLUP_OPTION;\
                                            }
    #endif

    #if defined(USB_SPEED_OPTION) && (USB_SPEED_OPTION != USB_FULL_SPEED)
        #error "Low speed operation in device mode is not currently supported in
the PIC24F family devices."
    #endif
#elif defined(__C32__)
    #define SetConfigurationOptions()   {U1CNFG1 = 0;}
#endif
//#define _UTEYE      0x80            // Use Eye-Pattern test

/* UEPn Initialization Parameters */
#if defined (__18CXX)
    #define EP_CTRL     0x06            // Cfg Control pipe for this ep
    #define EP_OUT      0x0C            // Cfg OUT only pipe for this ep
    #define EP_IN       0x0A            // Cfg IN only pipe for this ep
    #define EP_OUT_IN   0x0E            // Cfg both OUT & IN pipes for this ep
```

```
                                        // Handshake should be disable for isoch

    #define USB_HANDSHAKE_ENABLED   0x10
    #define USB_HANDSHAKE_DISABLED  0x00

    #define USB_OUT_ENABLED         0x04
    #define USB_OUT_DISABLED        0x00

    #define USB_IN_ENABLED          0x02
    #define USB_IN_DISABLED         0x00

    #define USB_ALLOW_SETUP         0x00
    #define USB_DISALLOW_SETUP      0x08

    #define USB_STALL_ENDPOINT      0x01
#elif defined(__C30__) || defined(__C32__)
    #define EP_CTRL     0x0C            // Cfg Control pipe for this ep
    #define EP_OUT      0x18            // Cfg OUT only pipe for this ep
    #define EP_IN       0x14            // Cfg IN only pipe for this ep
    #define EP_OUT_IN   0x1C            // Cfg both OUT & IN pipes for this ep
    #define HSHK_EN     0x01            // Enable handshake packet
                                        // Handshake should be disable for isoch

    #define USB_HANDSHAKE_ENABLED   0x01
    #define USB_HANDSHAKE_DISABLED  0x00

    #define USB_OUT_ENABLED         0x08
    #define USB_OUT_DISABLED        0x00

    #define USB_IN_ENABLED          0x04
    #define USB_IN_DISABLED         0x00

    #define USB_ALLOW_SETUP         0x00
    #define USB_DISALLOW_SETUP      0x10

    #define USB_STALL_ENDPOINT      0x02
#endif

//USB_HANDLE is a pointer to an entry in the BDT.  This pointer can be used
//  to read the length of the last transfer, the status of the last transfer,
//  and various other information.  Insure to initialize USB_HANDLE objects
//  to NULL so that they are in a known state during their first usage.
#define USB_HANDLE volatile BDT_ENTRY*

#if !defined(USBDEVICE_C)
/** EXTERNS ***************************************************/
    //Definitions for the BDT
    #if (USB_PING_PONG_MODE == USB_PING_PONG__NO_PING_PONG)
        extern volatile BDT_ENTRY BDT[(USB_MAX_EP_NUMBER + 1) * 2];
    #elif (USB_PING_PONG_MODE == USB_PING_PONG__EP0_OUT_ONLY)
        extern volatile BDT_ENTRY BDT[((USB_MAX_EP_NUMBER+1) * 2)+1];
    #elif (USB_PING_PONG_MODE == USB_PING_PONG__FULL_PING_PONG)
        extern volatile BDT_ENTRY BDT[(USB_MAX_EP_NUMBER + 1) * 4];
    #elif (USB_PING_PONG_MODE == USB_PING_PONG__ALL_BUT_EP0)
        extern volatile BDT_ENTRY BDT[((USB_MAX_EP_NUMBER + 1) * 4)-2];
    #else
        #error "No ping pong mode defined."
    #endif

    //Depricated in v2.2 - will be removed in a future revision
    #if !defined(USB_USER_DEVICE_DESCRIPTOR)
        //Device descriptor
        extern ROM USB_DEVICE_DESCRIPTOR device_dsc;
    #else
        USB_USER_DEVICE_DESCRIPTOR_INCLUDE;
    #endif
```

```
    //Configuration descriptor
    extern ROM BYTE configDescriptor1[];

    #if !defined(USB_USER_CONFIG_DESCRIPTOR)
        //Array of configuration descriptors
        extern ROM BYTE *ROM USB_CD_Ptr[];
    #else
        USB_USER_CONFIG_DESCRIPTOR_INCLUDE;
    #endif

    //Array of string descriptors
    extern ROM BYTE *ROM USB_SD_Ptr[];

    #if defined(USB_USE_HID)
    //Class specific - HID report descriptor
    #if !defined(__USB_DESCRIPTORS_C)
        extern ROM struct{BYTE report[HID_RPT01_SIZE];} hid_rpt01;
    #endif
    #endif

    //Buffer for control transfers
    extern volatile CTRL_TRF_SETUP SetupPkt;           // 8-byte only
    //Buffer for control transfer data
    extern volatile BYTE CtrlTrfData[USB_EP0_BUFF_SIZE];

    #if defined(USB_USE_HID)
    //class specific data buffers
    extern volatile unsigned char hid_report_out[HID_INT_OUT_EP_SIZE];
    extern volatile unsigned char hid_report_in[HID_INT_IN_EP_SIZE];
    #endif


#endif

/* Control Transfer States */
#define WAIT_SETUP          0
#define CTRL_TRF_TX         1
#define CTRL_TRF_RX         2

/* v2.1 fix - Short Packet States - Used by Control Transfer Read  - CTRL_TRF_TX
*/
#define SHORT_PKT_NOT_USED  0
#define SHORT_PKT_PENDING   1
#define SHORT_PKT_SENT      2

/* USB PID: Token Types - See chapter 8 in the USB specification */
#define SETUP_TOKEN         0x0D    // 0b00001101
#define OUT_TOKEN           0x01    // 0b00000001
#define IN_TOKEN            0x09    // 0b00001001

/* bmRequestType Definitions */
#define HOST_TO_DEV         0
#define DEV_TO_HOST         1

#define STANDARD            0x00
#define CLASS               0x01
#define VENDOR              0x02

#define RCPT_DEV            0
#define RCPT_INTF           1
#define RCPT_EP             2
#define RCPT_OTH            3

/** EXTERNS ***************************************************/
#if !defined(USBDEVICE_C)
    extern USB_VOLATILE BYTE USBDeviceState;
    extern USB_VOLATILE BYTE USBActiveConfiguration;
```

```
    extern USB_VOLATILE IN_PIPE inPipes[1];
    extern USB_VOLATILE OUT_PIPE outPipes[1];
    extern volatile BDT_ENTRY *pBDTEntryIn[USB_MAX_EP_NUMBER+1];
#endif

/** PUBLIC PROTOTYPES *********************************************/

/*********************************************************************
  Function:
        void USBDeviceTasks(void)

  Summary:
    This function is the main state machine of the USB device side stack.
    This function should be called periodically to receive and transmit
    packets through the stack. This function should be called preferably
    once every 100us during the enumeration process. After the enumeration
    process this function still needs to be called periodically to respond
    to various situations on the bus but is more relaxed in its time
    requirements. This function should also be called at least as fast as
    the OUT data expected from the PC.

  Description:
    This function is the main state machine of the USB device side stack.
    This function should be called periodically to receive and transmit
    packets through the stack. This function should be called preferably
    once every 100us during the enumeration process. After the enumeration
    process this function still needs to be called periodically to respond
    to various situations on the bus but is more relaxed in its time
    requirements. This function should also be called at least as fast as
    the OUT data expected from the PC.

    Typical usage:
    <code>
    void main(void)
    {
        USBDeviceInit()
        while(1)
        {
            USBDeviceTasks();
            if((USBGetDeviceState() \< CONFIGURED_STATE) ||
               (USBIsDeviceSuspended() == TRUE))
            {
                //Either the device is not configured or we are suspended
                //  so we don't want to do execute any application code
                continue;   //go back to the top of the while loop
            }
            else
            {
                //Otherwise we are free to run user application code.
                UserApplication();
            }
        }
    }
    </code>

  Conditions:
    None
  Remarks:
    This function should be called preferably once every 100us during the
    enumeration process. After the enumeration process this function still
    needs to be called periodically to respond to various situations on the
    bus but is more relaxed in its time requirements.
  ********************************************************************/
void USBDeviceTasks(void);

/*********************************************************************
    Function:
```

```
        void USBDeviceInit(void)

    Description:
        This function initializes the device stack it in the default state. The
        USB module will be completely reset including all of the internal
        variables, registers, and interrupt flags.

    Precondition:
        This function must be called before any of the other USB Device
        functions can be called, including USBDeviceTasks().

    Parameters:
        None

    Return Values:
        None

    Remarks:
        None

  *****************************************************************************/
void USBDeviceInit(void);

/********************************************************************
  Function:
        BOOL USBGetRemoteWakeupStatus(void)

  Summary:
    This function indicates if remote wakeup has been enabled by the host.
    Devices that support remote wakeup should use this function to
    determine if it should send a remote wakeup.

  Description:
    This function indicates if remote wakeup has been enabled by the host.
    Devices that support remote wakeup should use this function to
    determine if it should send a remote wakeup.

    If a device does not support remote wakeup (the Remote wakeup bit, bit
    5, of the bmAttributes field of the Configuration descriptor is set to
    1), then it should not send a remote wakeup command to the PC and this
    function is not of any use to the device. If a device does support
    remote wakeup then it should use this function as described below.

    If this function returns FALSE and the device is suspended, it should
    not issue a remote wakeup (resume).

    If this function returns TRUE and the device is suspended, it should
    issue a remote wakeup (resume).

    A device can add remote wakeup support by having the _RWU symbol added
    in the configuration descriptor (located in the usb_descriptors.c file
    in the project). This done in the 8th byte of the configuration
    descriptor. For example:

    <code lang="c">
    ROM BYTE configDescriptor1[]={
        0x09,                           // Size
        USB_DESCRIPTOR_CONFIGURATION,   // descriptor type
        DESC_CONFIG_WORD(0x0022),       // Total length
        1,                              // Number of interfaces
        1,                              // Index value of this cfg
        0,                              // Configuration string index
        _DEFAULT | _SELF | _RWU,        // Attributes, see usb_device.h
        50,                             // Max power consumption in 2X mA(100mA)

        //The rest of the configuration descriptor should follow
    </code>
```

```
    For more information about remote wakeup, see the following section of
    the USB v2.0 specification available at www.usb.org:
        * Section 9.2.5.2
        * Table 9-10
        * Section 7.1.7.7
        * Section 9.4.5

  Conditions:
    None

  Return Values:
    TRUE -   Remote Wakeup has been enabled by the host
    FALSE -  Remote Wakeup is not currently enabled

  Remarks:
    None

  ***************************************************************/
#define USBGetRemoteWakeupStatus() RemoteWakeup

/*********************************************************************
  Function:
        BYTE USBGetDeviceState(void)

  Summary:
    This function will return the current state of the device on the USB.
    This function should return CONFIGURED_STATE before an application
    tries to send information on the bus.
  Description:
    This function returns the current state of the device on the USB. This
    \function is used to determine when the device is ready to communicate
    on the bus. Applications should not try to send or receive data until
    this function returns CONFIGURED_STATE.

    It is also important that applications yield as much time as possible
    to the USBDeviceTasks() function as possible while the this function
    \returns any value between ATTACHED_STATE through CONFIGURED_STATE.

    For more information about the various device states, please refer to
    the USB specification section 9.1 available from www.usb.org.

    Typical usage:
    <code>
    void main(void)
    {
        USBDeviceInit()
        while(1)
        {
            USBDeviceTasks();
            if((USBGetDeviceState() \< CONFIGURED_STATE) ||
               (USBIsDeviceSuspended() == TRUE))
            {
                //Either the device is not configured or we are suspended
                //  so we don't want to do execute any application code
                continue;   //go back to the top of the while loop
            }
            else
            {
                //Otherwise we are free to run user application code.
                UserApplication();
            }
        }
    }
    </code>
  Conditions:
    None
```

```
  Return Values:
    DETACHED_STATE -     The device is not attached to the bus
    ATTACHED_STATE -     The device is attached to the bus but
    POWERED_STATE -      The device is not officially in the powered state
    DEFAULT_STATE -      The device has received a RESET from the host
    ADR_PENDING_STATE -  The device has received the SET_ADDRESS command but
                         hasn't received the STATUS stage of the command so
                         it is still operating on address 0.
    ADDRESS_STATE -      The device has an address assigned but has not
                         received a SET_CONFIGURATION command yet or has
                         received a SET_CONFIGURATION with a configuration
                         number of 0 (deconfigured)
    CONFIGURED_STATE -   the device has received a non\-zero
                         SET_CONFIGURATION command is now ready for
                         communication on the bus.
  Remarks:
    None
  *************************************************************************/
#define USBGetDeviceState() USBDeviceState

/************************************************************************
  Function:
        BOOL USBGetSuspendState(void)

  Summary:
    This function indicates if this device is currently suspended. When a
    device is suspended it will not be able to transfer data over the bus.
  Description:
    This function indicates if this device is currently suspended. When a
    device is suspended it will not be able to transfer data over the bus.
    This function can be used by the application to skip over section of
    code that do not need to exectute if the device is unable to send data
    over the bus.

    Typical usage:
    <code>
       void main(void)
       {
           USBDeviceInit()
           while(1)
           {
               USBDeviceTasks();
               if((USBGetDeviceState() \< CONFIGURED_STATE) ||
                  (USBIsDeviceSuspended() == TRUE))
               {
                   //Either the device is not configured or we are suspended
                   //  so we don't want to do execute any application code
                   continue;   //go back to the top of the while loop
               }
               else
               {
                   //Otherwise we are free to run user application code.
                   UserApplication();
               }
           }
       }
    </code>
  Conditions:
    None
  Return Values:
    TRUE -   this device is suspended.
    FALSE -  this device is not suspended.
  Remarks:
    None
  *************************************************************************/
#define USBIsDeviceSuspended() USBSuspendControl
```

```
void USBSoftDetach(void);
void USBCtrlEPService(void);
void USBCtrlTrfSetupHandler(void);
void USBCtrlTrfInHandler(void);
void USBCheckStdRequest(void);
void USBStdGetDscHandler(void);
void USBCtrlEPServiceComplete(void);
void USBCtrlTrfTxService(void);
void USBPrepareForNextSetupTrf(void);
void USBCtrlTrfRxService(void);
void USBStdSetCfgHandler(void);
void USBStdGetStatusHandler(void);
void USBStdFeatureReqHandler(void);
void USBCtrlTrfOutHandler(void);
BOOL USBIsTxBusy(BYTE EPNumber);
void USBPut(BYTE EPNum, BYTE Data);
void USBEPService(void);
void USBConfigureEndpoint(BYTE EPNum, BYTE direction);

void USBProtocolResetHandler(void);
void USBWakeFromSuspend(void);
void USBSuspend(void);
void USBStallHandler(void);
volatile USB_HANDLE USBTransferOnePacket(BYTE ep, BYTE dir, BYTE* data, BYTE
len);
void USBEnableEndpoint(BYTE ep,BYTE options);

#if defined(USB_DYNAMIC_EP_CONFIG)
    void USBInitEP(BYTE ROM* pConfig);
#else
    #define USBInitEP(a)
#endif

#if defined(ENABLE_EP0_DATA_RECEIVED_CALLBACK)
    void USBCBEP0DataReceived(void);
    #define USBCB_EP0_DATA_RECEIVED() USBCBEP0DataReceived()
#else
    #define USBCB_EP0_DATA_RECEIVED()
#endif

/** Section: CALLBACKS ********************************************************/

/******************************************************************************
  Function:
      void USBCBSuspend(void)

  Summary:
    Call back that is invoked when a USB suspend is detected.
  Description:
    Call back that is invoked when a USB suspend is detected.

    \Example power saving code. Insert appropriate code here for the
    desired application behavior. If the microcontroller will be put to
    sleep, a process similar to that shown below may be used:

    \Example Psuedo Code:
    <code>
    ConfigureIOPinsForLowPower();
    SaveStateOfAllInterruptEnableBits();
    DisableAllInterruptEnableBits();

    //should enable at least USBActivityIF as a wake source
    EnableOnlyTheInterruptsWhichWillBeUsedToWakeTheMicro();

    Sleep();
```

```
    //Preferrably, this should be done in the
    //  USBCBWakeFromSuspend() function instead.
    RestoreStateOfAllPreviouslySavedInterruptEnableBits();

    //Preferrably, this should be done in the
    //  USBCBWakeFromSuspend() function instead.
    RestoreIOPinsToNormal();
    </code>

    IMPORTANT NOTE: Do not clear the USBActivityIF (ACTVIF) bit here. This
    bit is cleared inside the usb_device.c file. Clearing USBActivityIF
    here will cause things to not work as intended.
  Conditions:
    None

    Paramters: None
  Side Effects:
    None

    Remark: None
  *****************************************************************************/
void USBCBSuspend(void);

/******************************************************************************
 Function:
   void USBCBWakeFromSuspend(void)

 Summary:
   This call back is invoked when a wakeup from USB suspend
   is detected.

 Description:
   The host may put USB peripheral devices in low power
   suspend mode (by "sending" 3+ms of idle).  Once in suspend
   mode, the host may wake the device back up by sending non-
   idle state signalling.

   This call back is invoked when a wakeup from USB suspend
   is detected.

   If clock switching or other power savings measures were taken when
   executing the USBCBSuspend() function, now would be a good time to
   switch back to normal full power run mode conditions.  The host allows
   a few milliseconds of wakeup time, after which the device must be
   fully back to normal, and capable of receiving and processing USB
   packets.  In order to do this, the USB module must receive proper
   clocking (IE: 48MHz clock must be available to SIE for full speed USB
   operation).

 PreCondition:  None

 Parameters:    None

 Return Values: None

 Remarks:       None
 *****************************************************************************/
void USBCBWakeFromSuspend(void);

/********************************************************************
  Function:
    void USBCB_SOF_Handler(void)

  Summary:
    This callback is called when a SOF packet is received by the host.
    (optional)
```

```
  Description:
    This callback is called when a SOF packet is received by the host.
    (optional)

    The USB host sends out a SOF packet to full-speed
    devices every 1 ms. This interrupt may be useful
    for isochronous pipes. End designers should
    implement callback routine as necessary.

  PreCondition:
    None

  Parameters:
    None

  Return Values:
    None

  Remarks:
    None
 *****************************************************************/
void USBCB_SOF_Handler(void);

/*****************************************************************
  Function:
    void USBCBErrorHandler(void)

  Summary:
    This callback is called whenever a USB error occurs. (optional)

  Description:
    This callback is called whenever a USB error occurs. (optional)

    The purpose of this callback is mainly for
    debugging during development. Check UEIR to see
    which error causes the interrupt.

  PreCondition:
    None

  Parameters:
    None

  Return Values:
    None

  Remarks:
    No need to clear UEIR to 0 here.
    Callback caller is already doing that.

    Typically, user firmware does not need to do anything special
    if a USB error occurs.  For example, if the host sends an OUT
    packet to your device, but the packet gets corrupted (ex:
    because of a bad connection, or the user unplugs the
    USB cable during the transmission) this will typically set
    one or more USB error interrupt flags.  Nothing specific
    needs to be done however, since the SIE will automatically
    send a "NAK" packet to the host.  In response to this, the
    host will normally retry to send the packet again, and no
    data loss occurs.  The system will typically recover
    automatically, without the need for application firmware
    intervention.

    Nevertheless, this callback function is provided, such as
    for debugging purposes.
 *****************************************************************/
void USBCBErrorHandler(void);
```

```
/*******************************************************************
  Function:
      void USBCBCheckOtherReq(void)

  Summary:
    This function is called whenever a request comes over endpoint 0 (the
    control endpoint) that the stack does not know how to handle.
  Description:
    When SETUP packets arrive from the host, some firmware must process the
    request and respond appropriately to fulfill the request. Some of the
    SETUP packets will be for standard USB "chapter 9" (as in, fulfilling
    chapter 9 of the official USB specifications) requests, while others
    may be specific to the USB device class that is being implemented. For
    \example, a HID class device needs to be able to respond to "GET
    REPORT" type of requests. This is not a standard USB chapter 9 request,
    and therefore not handled by usb_device.c. Instead this request should
    be handled by class specific firmware, such as that contained in
    usb_function_hid.c.

    Typical Usage:
    <code>
    void USBCBCheckOtherReq(void)
    {
        //Since the stack didn't handle the request I need to check
        //  my class drivers to see if it is for them
        USBCheckMSDRequest();
    }
    </code>
  Conditions:
    None
  Remarks:
    None
  *****************************************************************/
void USBCBCheckOtherReq(void);

/*******************************************************************
  Function:
    void USBCBStdSetDscHandler(void)

  Summary:
    This callback is called when a SET_DESCRIPTOR request is received (optional)

  Description:
    The USBCBStdSetDscHandler() callback function is
    called when a SETUP, bRequest: SET_DESCRIPTOR request
    arrives.  Typically SET_DESCRIPTOR requests are
    not used in most applications, and it is
    optional to support this type of request.

  PreCondition:
    None

  Parameters:
    None

  Return Values:
    None

  Remark:             None
 *****************************************************************/
void USBCBStdSetDscHandler(void);

/*******************************************************************************
*******************************
  Function:
      void USBCBInitEP(void)
```

```
  Summary:
    This function is called whenever the device receives a
    SET_CONFIGURATION request.
  Description:
    This function is called when the device becomes initialized, which
    occurs after the host sends a SET_CONFIGURATION (wValue not = 0)
    request. This callback function should initialize the endpoints for the
    device's usage according to the current configuration.

    Typical Usage:
    <code>
    void USBCBInitEP(void)
    {

USBEnableEndpoint(MSD_DATA_IN_EP,USB_IN_ENABLED|USB_OUT_ENABLED|USB_HANDSHAKE_ENA
BLED|USB_DISALLOW_SETUP);
        USBMSDInit();
    }
    </code>
  Conditions:
    None
  Remarks:
    None

*******************************************************************************
*******************************/
void USBCBInitEP(void);

/******************************************************************************
  Function:
        void USBCBSendResume(void)

  Summary:
    This function should be called to initiate a remote wakeup. (optional)
  Description:
    The USB specifications allow some types of USB peripheral devices to
    wake up a host PC (such as if it is in a low power suspend to RAM
    state). This can be a very useful feature in some USB applications,
    such as an Infrared remote control receiver. If a user presses the
    "power" button on a remote control, it is nice that the IR receiver can
    detect this signalling, and then send a USB "command" to the PC to wake
    up.

    The USBCBSendResume() "callback" function is used to send this special
    USB signalling which wakes up the PC. This function may be called by
    application firmware to wake up the PC. This function should only be
    called when:

      1. The USB driver used on the host PC supports the remote wakeup
         capability.
      2. The USB configuration descriptor indicates the device is remote
         wakeup capable in the bmAttributes field. (see usb_descriptors.c and
         _RWU)
      3. The USB host PC is currently sleeping, and has previously sent
         your device a SET FEATURE setup packet which "armed" the remote wakeup
         capability. (see USBGetRemoteWakeupStatus())

    This callback should send a RESUME signal that has the period of
    1-15ms.

    Typical Usage:
    <code>
    if((USBDeviceState == CONFIGURED_STATE)
        &amp;&amp; (USBIsDeviceSuspended() == TRUE)
        &amp;&amp; (USBGetRemoteWakeupStatus() == TRUE))
    {
```

```
        if(ButtonPressed)
        {
            //Wake up the USB module from suspend
            USBWakeFromSuspend();

            //Issue a remote wakeup command on the bus
            USBCBSendResume();
        }
    }
    </code>
  Conditions:
    None
  Remarks:
    A user can switch to primary first by calling USBCBWakeFromSuspend() if
    required/desired.

    The modifiable section in this routine should be changed to meet the
    application needs. Current implementation temporary blocks other
    functions from executing for a period of 1-13 ms depending on the core
    frequency.

    According to USB 2.0 specification section 7.1.7.7, "The remote wakeup
    device must hold the resume signaling for at lest 1 ms but for no more
    than 15 ms." The idea here is to use a delay counter loop, using a
    common value that would work over a wide range of core frequencies.
    That value selected is 1800. See table below:
    <table>
    Core Freq(MHz)   MIP (for PIC18)   RESUME Signal Period (ms)
    --------------   ---------------   -------------------------
    48               12                1.05
    4                1                 12.6
    </table>
      * These timing could be incorrect when using code optimization or
        extended instruction mode, or when having other interrupts enabled.
        Make sure to verify using the MPLAB SIM's Stopwatch and verify the
        actual signal on an oscilloscope.
      * These timing numbers should be recalculated when using PIC24 or
        PIC32 as they have different clocking structures.
      * A timer can be used in place of the blocking loop if desired.

*******************************************************************************/
void USBCBSendResume(void);

/********************************************************************
  Function:
    void USBCBEP0DataReceived(void)

  Summary:
    This function is called whenever a EP0 data
    packet is received. (optional)

  Description:
    This function is called whenever a EP0 data
    packet is received.  This gives the user (and
    thus the various class examples a way to get
    data that is received via the control endpoint.
    This function needs to be used in conjunction
    with the USBCBCheckOtherReq() function since
    the USBCBCheckOtherReq() function is the apps
    method for getting the initial control transfer
    before the data arrives.

  PreCondition:
    ENABLE_EP0_DATA_RECEIVED_CALLBACK must be
    defined already (in usb_config.h)

  Parameters:
```

```
    None

  Return Values:
    None

  Remarks:
    None
*****************************************************************/
void USBCBEP0DataReceived(void);



/** Section: MACROS *************************************************/

#define DESC_CONFIG_BYTE(a) (a)
#define DESC_CONFIG_WORD(a) (a&0xFF),((a>>8)&0xFF)
#define DESC_CONFIG_DWORD(a) (a&0xFF),((a>>8)&0xFF),((a>>16)&0xFF),((a>>24)&0xFF)

/********************************************************************
  Function:
    BOOL USBHandleBusy(USB_HANDLE handle)

  Summary:
    Checks to see if the input handle is busy

  Description:
    Checks to see if the input handle is busy

    Typical Usage
    <code>
    //make sure that the last transfer isn't busy by checking the handle
    if(!USBHandleBusy(USBGenericInHandle))
    {
        //Send the data contained in the INPacket[] array out on
        //  endpoint USBGEN_EP_NUM
        USBGenericInHandle =
USBGenWrite(USBGEN_EP_NUM,(BYTE*)&INPacket[0],sizeof(INPacket));
    }
    </code>

  Conditions:
    None
  Input:
    USB_HANDLE handle -  handle of the transfer that you want to check the
                         status of
  Return Values:
    TRUE -   The specified handle is busy
    FALSE -  The specified handle is free and available for a transfer
  Remarks:
    None
  *******************************************************************/
#define USBHandleBusy(handle) (handle==0?0:handle->STAT.UOWN)

/*********************************************************************
    Function:
        WORD USBHandleGetLength(USB_HANDLE handle)

    Summary:
        Retrieves the length of the destination buffer of the input
        handle

    Description:
        Retrieves the length of the destination buffer of the input
        handle

    PreCondition:
```

```
        None

    Parameters:
        USB_HANDLE handle - the handle to the transfer you want the
        address for.

    Return Values:
        WORD - length of the current buffer that the input handle
        points to.  If the transfer is complete then this is the
        length of the data transmitted or the length of data
        actually received.

    Remarks:
        None

 ***************************************************************************/
#define USBHandleGetLength(handle) (handle->CNT)

/****************************************************************************
    Function:
        WORD USBHandleGetAddr(USB_HANDLE)

    Summary:
        Retrieves the address of the destination buffer of the input
        handle

    Description:
        Retrieves the address of the destination buffer of the input
        handle

    PreCondition:
        None

    Parameters:
        USB_HANDLE handle - the handle to the transfer you want the
        address for.

    Return Values:
        WORD - address of the current buffer that the input handle
        points to.

    Remarks:
        None

 ***************************************************************************/
#define USBHandleGetAddr(handle) (handle->ADR)

/****************************************************************************
    Function:
        void USBEP0SetSourceRAM(BYTE* src)

    Summary:
        Sets the address of the data to send over the
        control endpoint

    PreCondition:
        None

    Paramters:
        src - address of the data to send

    Return Values:
        None

    Remarks:
        None
```

```
 ***************************************************************/
#define USBEP0SetSourceRAM(src) inPipes[0].pSrc.bRam = src

/*********************************************************************
    Function:
        void USBEP0SetSourceROM(BYTE* src)

    Summary:
        Sets the address of the data to send over the
        control endpoint

    PreCondition:
        None

    Parameters:
        src - address of the data to send

    Return Values:
        None

    Remarks:
        None

 ***************************************************************/
#define USBEP0SetSourceROM(src) inPipes[0].pSrc.bRom = src

/*********************************************************************
    Function:
        void USBEP0Transmit(BYTE options)

    Summary:
        Sets the address of the data to send over the
        control endpoint

    PreCondition:
        None

    Paramters:
        options - the various options that you want
                  when sending the control data. Options are:
                       USB_INPIPES_ROM
                       USB_INPIPES_RAM
                       USB_INPIPES_BUSY
                       USB_INPIPES_INCLUDE_ZERO
                       USB_INPIPES_NO_DATA
                       USB_INPIPES_NO_OPTIONS

    Return Values:
        None

    Remarks:
        None

 ***************************************************************/
#define USBEP0Transmit(options) inPipes[0].info.Val = options | USB_INPIPES_BUSY

/*********************************************************************
    Function:
        void USBEP0SetSize(WORD size)

    Summary:
        Sets the size of the data to send over the
        control endpoint

    PreCondition:
        None
```

```
    Parameters:
        size - the size of the data needing to be transmitted

    Return Values:
        None

    Remarks:
        None

 ********************************************************************/
#define USBEP0SetSize(size) inPipes[0].wCount.Val = size

/********************************************************************
    Function:
        void USBEP0SendRAMPtr(BYTE* src, WORD size, BYTE Options)

    Summary:
        Sets the source, size, and options of the data
        you wish to send from a RAM source

    PreCondition:
        None

    Parameters:
        src - address of the data to send
        size - the size of the data needing to be transmitted
        options - the various options that you want
        when sending the control data. Options are:
        USB_INPIPES_ROM
        USB_INPIPES_RAM
        USB_INPIPES_BUSY
        USB_INPIPES_INCLUDE_ZERO
        USB_INPIPES_NO_DATA
        USB_INPIPES_NO_OPTIONS

    Return Values:
        None

    Remarks:
        None

 ********************************************************************/
#define USBEP0SendRAMPtr(src,size,options)
{USBEP0SetSourceRAM(src);USBEP0SetSize(size);USBEP0Transmit(options |
USB_EP0_RAM);}

/********************************************************************
    Function:
        void USBEP0SendROMPtr(BYTE* src, WORD size, BYTE Options)

    Summary:
        Sets the source, size, and options of the data
        you wish to send from a ROM source

    PreCondition:
        None

    Parameters:
        src - address of the data to send
        size - the size of the data needing to be transmitted
        options - the various options that you want
        when sending the control data. Options are:
        USB_INPIPES_ROM
        USB_INPIPES_RAM
        USB_INPIPES_BUSY
        USB_INPIPES_INCLUDE_ZERO
        USB_INPIPES_NO_DATA
```

```
        USB_INPIPES_NO_OPTIONS

    Return Values:
        None

    Remarks:
        None

 ********************************************************************/
#define USBEP0SendROMPtr(src,size,options)
{USBEP0SetSourceROM(src);USBEP0SetSize(size);USBEP0Transmit(options |
USB_EP0_ROM);}

/********************************************************************
    Function:
        USB_HANDLE USBTxOnePacket(BYTE ep, BYTE* data, WORD len)

    Summary:
        Sends the specified data out the specified endpoint

    PreCondition:
        None

    Parameters:
        ep - the endpoint you want to send the data out of
        data - the data that you wish to send
        len - the length of the data that you wish to send

    Return Values:
        USB_HANDLE - a handle for the transfer.  This information
        should be kept to track the status of the transfer

    Remarks:
        None

 ********************************************************************/
#define USBTxOnePacket(ep,data,len)
USBTransferOnePacket(ep,IN_TO_HOST,data,len)

/********************************************************************
    Function:
        USB_HANDLE USBRxOnePacket(BYTE ep, BYTE* data, WORD len)

    Summary:
        Receives the specified data out the specified endpoint

    PreCondition:
        None

    Parameters:
        ep - the endpoint you want to receive the data into
        data - where the data will go when it arrives
        len - the length of the data that you wish to receive

    Return Values:
        None

    Remarks:
        None

 ********************************************************************/
#define USBRxOnePacket(ep,data,len)
USBTransferOnePacket(ep,OUT_FROM_HOST,data,len)

/********************************************************************
    Function:
        void USBClearInterruptFlag(WORD reg, BYTE flag)
```

```
    Summary:
        Clears the specified interrupt flag

    PreCondition:
        None

    Parameters:
        WORD reg - the register name holding the interrupt flag
        BYTE flag - the bit number needing to be cleared

    Return Values:
        None

    Remarks:
        None

 *****************************************************************/
void USBClearInterruptFlag(BYTE* reg, BYTE flag);

/*****************************************************************
    Function:
        void USBClearInterruptRegister(WORD reg)

    Summary:
        Clears the specified interrupt register

    PreCondition:
        None

    Parameters:
        WORD reg - the register name that needs to be cleared

    Return Values:
        None

    Remarks:
        None

 *****************************************************************/
#if defined(__18CXX)
    #define USBClearInterruptRegister(reg) reg = 0;
#elif  defined(__C30__) || defined(__C32__)
    #define USBClearInterruptRegister(reg) reg = 0xFF;
#endif

/*****************************************************************
    Function:
        void USBStallEndpoint(BYTE ep, BYTE dir)

    Summary:
         STALLs the specified endpoint

    PreCondition:
        None

    Parameters:
        BYTE ep - the endpoint the data will be transmitted on
        BYTE dir - the direction of the transfer

    Return Values:
        None

    Remarks:
        None

 *****************************************************************/
```

```
void USBStallEndpoint(BYTE ep, BYTE dir);

#if defined USB_ENABLE_ALL_HANDLERS
    #define USB_ENABLE_SUSPEND_HANDLER
    #define USB_ENABLE_WAKEUP_FROM_SUSPEND_HANDLER
    #define USB_ENABLE_SOF_HANDLER
    #define USB_ENABLE_ERROR_HANDLER
    #define USB_ENABLE_OTHER_REQUEST_HANDLER
    #define USB_ENABLE_SET_DESCRIPTOR_HANDLER
    #define USB_ENABLE_INIT_EP_HANDLER
    #define USB_ENABLE_EP0_DATA_HANDLER
    #define USB_ENABLE_TRANSFER_COMPLETE_HANDLER
#endif

#if defined USB_ENABLE_SUSPEND_HANDLER
    #define USB_SUSPEND_HANDLER                 USB_SUSPEND_HANDLER_FUNC
#else
    #define USB_SUSPEND_HANDLER
#endif

#if defined USB_ENABLE_WAKEUP_FROM_SUSPEND_HANDLER
    #define USB_WAKEUP_FROM_SUSPEND_HANDLER
USB_WAKEUP_FROM_SUSPEND_HANDLER_FUNC
#else
    #define USB_WAKEUP_FROM_SUSPEND_HANDLER
#endif

#if defined USB_ENABLE_SOF_HANDLER
    #define USB_SOF_HANDLER                 USB_SOF_HANDLER_FUNC
#else
    #define USB_SOF_HANDLER
#endif

#if defined USB_ENABLE_ERROR_HANDLER
    #define USB_ERROR_HANDLER                 USB_ERROR_HANDLER_FUNC
#else
    #define USB_ERROR_HANDLER
#endif

#if defined USB_ENABLE_OTHER_REQUEST_HANDLER
    #define USB_OTHER_REQUEST_HANDLER
USB_OTHER_REQUEST_HANDLER_FUNC
#else
    #define USB_OTHER_REQUEST_HANDLER
#endif

#if defined USB_ENABLE_SET_DESCRIPTOR_HANDLER
    #define USB_SET_DESCRIPTOR_HANDLER
USB_SET_DESCRIPTOR_HANDLER_FUNC
#else
    #define USB_SET_DESCRIPTOR_HANDLER
#endif

#if defined USB_ENABLE_INIT_EP_HANDLER
    #define USB_INIT_EP_HANDLER                 USB_INIT_EP_HANDLER_FUNC
#else
    #define USB_INIT_EP_HANDLER
#endif

#if defined USB_ENABLE_EP0_DATA_HANDLER
    #define USB_EP0_DATA_HANDLER                USB_EP0_DATA_HANDLER_FUNC
#else
    #define USB_EP0_DATA_HANDLER
#endif

#if defined USB_ENABLE_TRANSFER_COMPLETE_HANDLER
```

```
    #define USB_TRASFER_COMPLETE_HANDLER
USB_TRASFER_COMPLETE_HANDLER_FUNC
#else
    #define USB_TRASFER_COMPLETE_HANDLER
#endif


#if defined(USB_INTERRUPT_LEGACY_CALLBACKS)
    #define USB_SUSPEND_HANDLER_FUNC(event,pointer,size)
USBCBSuspend()
    #define USB_WAKEUP_FROM_SUSPEND_HANDLER_FUNC(event,pointer,size)
USBCBWakeFromSuspend()
    #define USB_SOF_HANDLER_FUNC(event,pointer,size)
USBCB_SOF_Handler()
    #define USB_ERROR_HANDLER_FUNC(event,pointer,size)
USBCBErrorHandler()
    #define USB_OTHER_REQUEST_HANDLER_FUNC(event,pointer,size)
USBCBCheckOtherReq()
    #define USB_SET_DESCRIPTOR_HANDLER_FUNC(event,pointer,size)
USBCBStdSetDscHandler()
    #define USB_INIT_EP_HANDLER_FUNC(event,pointer,size)
USBCBInitEP()
    #define USB_EP0_DATA_HANDLER_FUNC(event,pointer,size)
USBCBEP0DataReceived()
    #define USB_TRASFER_COMPLETE_HANDLER_FUNC(event,pointer,size)
#else
    #define USB_SUSPEND_HANDLER_FUNC(event,pointer,size)
USER_USB_CALLBACK_EVENT_HANDLER(event,pointer,size)
    #define USB_WAKEUP_FROM_SUSPEND_HANDLER_FUNC(event,pointer,size)
USER_USB_CALLBACK_EVENT_HANDLER(event,pointer,size)
    #define USB_SOF_HANDLER_FUNC(event,pointer,size)
USER_USB_CALLBACK_EVENT_HANDLER(event,pointer,size)
    #define USB_ERROR_HANDLER_FUNC(event,pointer,size)
USER_USB_CALLBACK_EVENT_HANDLER(event,pointer,size)
    #define USB_OTHER_REQUEST_HANDLER_FUNC(event,pointer,size)
USER_USB_CALLBACK_EVENT_HANDLER(event,pointer,size)
    #define USB_SET_DESCRIPTOR_HANDLER_FUNC(event,pointer,size)
USER_USB_CALLBACK_EVENT_HANDLER(event,pointer,size)
    #define USB_INIT_EP_HANDLER_FUNC(event,pointer,size)
USER_USB_CALLBACK_EVENT_HANDLER(event,pointer,size)
    #define USB_EP0_DATA_HANDLER_FUNC(event,pointer,size)
USER_USB_CALLBACK_EVENT_HANDLER(event,pointer,size)
    #define USB_TRASFER_COMPLETE_HANDLER_FUNC(event,pointer,size)
USER_USB_CALLBACK_EVENT_HANDLER(event,pointer,size)
#endif

#if defined(USB_INTERRUPT)
void USBDeviceDetach(void);
void USBDeviceAttach(void);
#endif

#if (USB_PING_PONG_MODE == USB_PING_PONG__NO_PING_PONG)
    #define USB_NEXT_EP0_OUT_PING_PONG 0x0000   // Used in USB Device Mode only
    #define USB_NEXT_EP0_IN_PING_PONG 0x0000    // Used in USB Device Mode only
    #define USB_NEXT_PING_PONG 0x0000           // Used in USB Device Mode only
    #define EP0_OUT_EVEN    0                   // Used in USB Device Mode only
    #define EP0_OUT_ODD     0                   // Used in USB Device Mode only
    #define EP0_IN_EVEN     1                   // Used in USB Device Mode only
    #define EP0_IN_ODD      1                   // Used in USB Device Mode only
    #define EP1_OUT_EVEN    2                   // Used in USB Device Mode only
    #define EP1_OUT_ODD     2                   // Used in USB Device Mode only
    #define EP1_IN_EVEN     3                   // Used in USB Device Mode only
    #define EP1_IN_ODD      3                   // Used in USB Device Mode only
    #define EP2_OUT_EVEN    4                   // Used in USB Device Mode only
    #define EP2_OUT_ODD     4                   // Used in USB Device Mode only
    #define EP2_IN_EVEN     5                   // Used in USB Device Mode only
    #define EP2_IN_ODD      5                   // Used in USB Device Mode only
```

```
    #define EP3_OUT_EVEN    6                   // Used in USB Device Mode only
    #define EP3_OUT_ODD     6                   // Used in USB Device Mode only
    #define EP3_IN_EVEN     7                   // Used in USB Device Mode only
    #define EP3_IN_ODD      7                   // Used in USB Device Mode only
    #define EP4_OUT_EVEN    8                   // Used in USB Device Mode only
    #define EP4_OUT_ODD     8                   // Used in USB Device Mode only
    #define EP4_IN_EVEN     9                   // Used in USB Device Mode only
    #define EP4_IN_ODD      9                   // Used in USB Device Mode only
    #define EP5_OUT_EVEN    10                  // Used in USB Device Mode only
    #define EP5_OUT_ODD     10                  // Used in USB Device Mode only
    #define EP5_IN_EVEN     11                  // Used in USB Device Mode only
    #define EP5_IN_ODD      11                  // Used in USB Device Mode only
    #define EP6_OUT_EVEN    12                  // Used in USB Device Mode only
    #define EP6_OUT_ODD     12                  // Used in USB Device Mode only
    #define EP6_IN_EVEN     13                  // Used in USB Device Mode only
    #define EP6_IN_ODD      13                  // Used in USB Device Mode only
    #define EP7_OUT_EVEN    14                  // Used in USB Device Mode only
    #define EP7_OUT_ODD     14                  // Used in USB Device Mode only
    #define EP7_IN_EVEN     15                  // Used in USB Device Mode only
    #define EP7_IN_ODD      15                  // Used in USB Device Mode only
    #define EP8_OUT_EVEN    16                  // Used in USB Device Mode only
    #define EP8_OUT_ODD     16                  // Used in USB Device Mode only
    #define EP8_IN_EVEN     17                  // Used in USB Device Mode only
    #define EP8_IN_ODD      17                  // Used in USB Device Mode only
    #define EP9_OUT_EVEN    18                  // Used in USB Device Mode only
    #define EP9_OUT_ODD     18                  // Used in USB Device Mode only
    #define EP9_IN_EVEN     19                  // Used in USB Device Mode only
    #define EP9_IN_ODD      19                  // Used in USB Device Mode only
    #define EP10_OUT_EVEN   20                  // Used in USB Device Mode only
    #define EP10_OUT_ODD    20                  // Used in USB Device Mode only
    #define EP10_IN_EVEN    21                  // Used in USB Device Mode only
    #define EP10_IN_ODD     21                  // Used in USB Device Mode only
    #define EP11_OUT_EVEN   22                  // Used in USB Device Mode only
    #define EP11_OUT_ODD    22                  // Used in USB Device Mode only
    #define EP11_IN_EVEN    23                  // Used in USB Device Mode only
    #define EP11_IN_ODD     23                  // Used in USB Device Mode only
    #define EP12_OUT_EVEN   24                  // Used in USB Device Mode only
    #define EP12_OUT_ODD    24                  // Used in USB Device Mode only
    #define EP12_IN_EVEN    25                  // Used in USB Device Mode only
    #define EP12_IN_ODD     25                  // Used in USB Device Mode only
    #define EP13_OUT_EVEN   26                  // Used in USB Device Mode only
    #define EP13_OUT_ODD    26                  // Used in USB Device Mode only
    #define EP13_IN_EVEN    27                  // Used in USB Device Mode only
    #define EP13_IN_ODD     27                  // Used in USB Device Mode only
    #define EP14_OUT_EVEN   28                  // Used in USB Device Mode only
    #define EP14_OUT_ODD    28                  // Used in USB Device Mode only
    #define EP14_IN_EVEN    29                  // Used in USB Device Mode only
    #define EP14_IN_ODD     29                  // Used in USB Device Mode only
    #define EP15_OUT_EVEN   30                  // Used in USB Device Mode only
    #define EP15_OUT_ODD    30                  // Used in USB Device Mode only
    #define EP15_IN_EVEN    31                  // Used in USB Device Mode only
    #define EP15_IN_ODD     31                  // Used in USB Device Mode only

    #define EP(ep,dir,pp) (2*ep+dir)            // Used in USB Device Mode only

    #define BD(ep,dir,pp)   ((8 * ep) + (4 * dir))      // Used in USB Device
Mode only

#elif (USB_PING_PONG_MODE == USB_PING_PONG__EP0_OUT_ONLY)
    #define USB_NEXT_EP0_OUT_PING_PONG 0x0004
    #define USB_NEXT_EP0_IN_PING_PONG 0x0000
    #define USB_NEXT_PING_PONG 0x0000
    #define EP0_OUT_EVEN    0
    #define EP0_OUT_ODD     1
    #define EP0_IN_EVEN     2
    #define EP0_IN_ODD      2
    #define EP1_OUT_EVEN    3
```

```
    #define EP1_OUT_ODD     3
    #define EP1_IN_EVEN     4
    #define EP1_IN_ODD      4
    #define EP2_OUT_EVEN    5
    #define EP2_OUT_ODD     5
    #define EP2_IN_EVEN     6
    #define EP2_IN_ODD      6
    #define EP3_OUT_EVEN    7
    #define EP3_OUT_ODD     7
    #define EP3_IN_EVEN     8
    #define EP3_IN_ODD      8
    #define EP4_OUT_EVEN    9
    #define EP4_OUT_ODD     9
    #define EP4_IN_EVEN     10
    #define EP4_IN_ODD      10
    #define EP5_OUT_EVEN    11
    #define EP5_OUT_ODD     11
    #define EP5_IN_EVEN     12
    #define EP5_IN_ODD      12
    #define EP6_OUT_EVEN    13
    #define EP6_OUT_ODD     13
    #define EP6_IN_EVEN     14
    #define EP6_IN_ODD      14
    #define EP7_OUT_EVEN    15
    #define EP7_OUT_ODD     15
    #define EP7_IN_EVEN     16
    #define EP7_IN_ODD      16
    #define EP8_OUT_EVEN    17
    #define EP8_OUT_ODD     17
    #define EP8_IN_EVEN     18
    #define EP8_IN_ODD      18
    #define EP9_OUT_EVEN    19
    #define EP9_OUT_ODD     19
    #define EP9_IN_EVEN     20
    #define EP9_IN_ODD      20
    #define EP10_OUT_EVEN   21
    #define EP10_OUT_ODD    21
    #define EP10_IN_EVEN    22
    #define EP10_IN_ODD     22
    #define EP11_OUT_EVEN   23
    #define EP11_OUT_ODD    23
    #define EP11_IN_EVEN    24
    #define EP11_IN_ODD     24
    #define EP12_OUT_EVEN   25
    #define EP12_OUT_ODD    25
    #define EP12_IN_EVEN    26
    #define EP12_IN_ODD     26
    #define EP13_OUT_EVEN   27
    #define EP13_OUT_ODD    27
    #define EP13_IN_EVEN    28
    #define EP13_IN_ODD     28
    #define EP14_OUT_EVEN   29
    #define EP14_OUT_ODD    29
    #define EP14_IN_EVEN    30
    #define EP14_IN_ODD     30
    #define EP15_OUT_EVEN   31
    #define EP15_OUT_ODD    31
    #define EP15_IN_EVEN    32
    #define EP15_IN_ODD     32

    #define EP(ep,dir,pp) (2*ep+dir+(((ep==0)&&(dir==0))?pp:2))
    #define BD(ep,dir,pp) (4*(ep+dir+(((ep==0)&&(dir==0))?pp:2)))

#elif (USB_PING_PONG_MODE == USB_PING_PONG__FULL_PING_PONG)
    #if defined (__18CXX) || defined(__C30__)
        #define USB_NEXT_EP0_OUT_PING_PONG 0x0004
        #define USB_NEXT_EP0_IN_PING_PONG 0x0004
```

```
    #define USB_NEXT_PING_PONG 0x0004
#elif defined(__C32__)
    #define USB_NEXT_EP0_OUT_PING_PONG 0x0008
    #define USB_NEXT_EP0_IN_PING_PONG 0x0008
    #define USB_NEXT_PING_PONG 0x0008
#else
    #error "Not defined for this compiler"
#endif
#define EP0_OUT_EVEN    0
#define EP0_OUT_ODD     1
#define EP0_IN_EVEN     2
#define EP0_IN_ODD      3
#define EP1_OUT_EVEN    4
#define EP1_OUT_ODD     5
#define EP1_IN_EVEN     6
#define EP1_IN_ODD      7
#define EP2_OUT_EVEN    8
#define EP2_OUT_ODD     9
#define EP2_IN_EVEN     10
#define EP2_IN_ODD      11
#define EP3_OUT_EVEN    12
#define EP3_OUT_ODD     13
#define EP3_IN_EVEN     14
#define EP3_IN_ODD      15
#define EP4_OUT_EVEN    16
#define EP4_OUT_ODD     17
#define EP4_IN_EVEN     18
#define EP4_IN_ODD      19
#define EP5_OUT_EVEN    20
#define EP5_OUT_ODD     21
#define EP5_IN_EVEN     22
#define EP5_IN_ODD      23
#define EP6_OUT_EVEN    24
#define EP6_OUT_ODD     25
#define EP6_IN_EVEN     26
#define EP6_IN_ODD      27
#define EP7_OUT_EVEN    28
#define EP7_OUT_ODD     29
#define EP7_IN_EVEN     30
#define EP7_IN_ODD      31
#define EP8_OUT_EVEN    32
#define EP8_OUT_ODD     33
#define EP8_IN_EVEN     34
#define EP8_IN_ODD      35
#define EP9_OUT_EVEN    36
#define EP9_OUT_ODD     37
#define EP9_IN_EVEN     38
#define EP9_IN_ODD      39
#define EP10_OUT_EVEN   40
#define EP10_OUT_ODD    41
#define EP10_IN_EVEN    42
#define EP10_IN_ODD     43
#define EP11_OUT_EVEN   44
#define EP11_OUT_ODD    45
#define EP11_IN_EVEN    46
#define EP11_IN_ODD     47
#define EP12_OUT_EVEN   48
#define EP12_OUT_ODD    49
#define EP12_IN_EVEN    50
#define EP12_IN_ODD     51
#define EP13_OUT_EVEN   52
#define EP13_OUT_ODD    53
#define EP13_IN_EVEN    54
#define EP13_IN_ODD     55
#define EP14_OUT_EVEN   56
#define EP14_OUT_ODD    57
#define EP14_IN_EVEN    58
```

```
    #define EP14_IN_ODD     59
    #define EP15_OUT_EVEN   60
    #define EP15_OUT_ODD    61
    #define EP15_IN_EVEN    62
    #define EP15_IN_ODD     63

    #define EP(ep,dir,pp) (4*ep+2*dir+pp)

    #if defined (__18CXX) || defined(__C30__)
        #define BD(ep,dir,pp) (4*(4*ep+2*dir+pp))
    #elif defined(__C32__)
        #define BD(ep,dir,pp) (8*(4*ep+2*dir+pp))
    #else
        #error "Not defined for this compiler"
    #endif

#elif (USB_PING_PONG_MODE == USB_PING_PONG__ALL_BUT_EP0)
    #define USB_NEXT_EP0_OUT_PING_PONG 0x0000
    #define USB_NEXT_EP0_IN_PING_PONG 0x0000
    #define USB_NEXT_PING_PONG 0x0004
    #define EP0_OUT_EVEN    0
    #define EP0_OUT_ODD     0
    #define EP0_IN_EVEN     1
    #define EP0_IN_ODD      1
    #define EP1_OUT_EVEN    2
    #define EP1_OUT_ODD     3
    #define EP1_IN_EVEN     4
    #define EP1_IN_ODD      5
    #define EP2_OUT_EVEN    6
    #define EP2_OUT_ODD     7
    #define EP2_IN_EVEN     8
    #define EP2_IN_ODD      9
    #define EP3_OUT_EVEN    10
    #define EP3_OUT_ODD     11
    #define EP3_IN_EVEN     12
    #define EP3_IN_ODD      13
    #define EP4_OUT_EVEN    14
    #define EP4_OUT_ODD     15
    #define EP4_IN_EVEN     16
    #define EP4_IN_ODD      17
    #define EP5_OUT_EVEN    18
    #define EP5_OUT_ODD     19
    #define EP5_IN_EVEN     20
    #define EP5_IN_ODD      21
    #define EP6_OUT_EVEN    22
    #define EP6_OUT_ODD     23
    #define EP6_IN_EVEN     24
    #define EP6_IN_ODD      25
    #define EP7_OUT_EVEN    26
    #define EP7_OUT_ODD     27
    #define EP7_IN_EVEN     28
    #define EP7_IN_ODD      29
    #define EP8_OUT_EVEN    30
    #define EP8_OUT_ODD     31
    #define EP8_IN_EVEN     32
    #define EP8_IN_ODD      33
    #define EP9_OUT_EVEN    34
    #define EP9_OUT_ODD     35
    #define EP9_IN_EVEN     36
    #define EP9_IN_ODD      37
    #define EP10_OUT_EVEN   38
    #define EP10_OUT_ODD    39
    #define EP10_IN_EVEN    40
    #define EP10_IN_ODD     41
    #define EP11_OUT_EVEN   42
    #define EP11_OUT_ODD    43
    #define EP11_IN_EVEN    44
```

```
    #define EP11_IN_ODD     45
    #define EP12_OUT_EVEN   46
    #define EP12_OUT_ODD    47
    #define EP12_IN_EVEN    48
    #define EP12_IN_ODD     49
    #define EP13_OUT_EVEN   50
    #define EP13_OUT_ODD    51
    #define EP13_IN_EVEN    52
    #define EP13_IN_ODD     53
    #define EP14_OUT_EVEN   54
    #define EP14_OUT_ODD    55
    #define EP14_IN_EVEN    56
    #define EP14_IN_ODD     57
    #define EP15_OUT_EVEN   58
    #define EP15_OUT_ODD    59
    #define EP15_IN_EVEN    60
    #define EP15_IN_ODD     61

    #define EP(ep,dir,pp) (4*ep+2*dir+((ep==0)?0:(pp-2)))
    #define BD(ep,dir,pp) (4*(4*ep+2*dir+((ep==0)?0:(pp-2))))

#else
    #error "No ping pong mode defined."
#endif

extern USB_VOLATILE BOOL RemoteWakeup;

#endif //USBD_H
```

6.1.17. usb_descriptors.h

```
/********************************************************************
 FileName:          usb_descriptors.c
 Dependencies:      See INCLUDES section
 Processor:         PIC18 or PIC24 USB Microcontrollers
 Hardware:          The code is natively intended to be used on the following
                    hardware platforms: PICDEM™ FS USB Demo Board,
                    PIC18F87J50 FS USB Plug-In Module, or
                    Explorer 16 + PIC24 USB PIM.  The firmware may be
                    modified for use on other USB platforms by editing the
                    HardwareProfile.h file.
 Complier:   Microchip C18 (for PIC18) or C30 (for PIC24)
 Company:           Microchip Technology, Inc.

 Software License Agreement:

 The software supplied herewith by Microchip Technology Incorporated
 (the “Company”) for its PIC® Microcontroller is intended and
 supplied to you, the Company’s customer, for use solely and
 exclusively on Microchip PIC Microcontroller products. The
 software is owned by the Company and/or its supplier, and is
 protected under applicable copyright laws. All rights are reserved.
 Any use in violation of the foregoing restrictions may subject the
 user to criminal sanctions under applicable laws, as well as to
 civil liability for the breach of the terms and conditions of this
 license.

 THIS SOFTWARE IS PROVIDED IN AN “AS IS” CONDITION. NO WARRANTIES,
 WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
 TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
 IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
 CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

********************************************************************
-usb_descriptors.c-
-------------------------------------------------------------------
Filling in the descriptor values in the usb_descriptors.c file:
-------------------------------------------------------------------

[Device Descriptors]
The device descriptor is defined as a USB_DEVICE_DESCRIPTOR type.
This type is defined in usb_ch9.h  Each entry into this structure
needs to be the correct length for the data type of the entry.

[Configuration Descriptors]
The configuration descriptor was changed in v2.x from a structure
to a BYTE array.  Given that the configuration is now a byte array
each byte of multi-byte fields must be listed individually.  This
means that for fields like the total size of the configuration where
the field is a 16-bit value "64,0," is the correct entry for a
configuration that is only 64 bytes long and not "64," which is one
too few bytes.

The configuration attribute must always have the _DEFAULT
definition at the minimum. Additional options can be ORed
to the _DEFAULT attribute. Available options are _SELF and _RWU.
These definitions are defined in the usb_device.h file. The
_SELF tells the USB host that this device is self-powered. The
_RWU tells the USB host that this device supports Remote Wakeup.

[Endpoint Descriptors]
Like the configuration descriptor, the endpoint descriptors were
changed in v2.x of the stack from a structure to a BYTE array.  As
endpoint descriptors also has a field that are multi-byte entities,
please be sure to specify both bytes of the field.  For example, for
```

```
the endpoint size an endpoint that is 64 bytes needs to have the size
defined as "64,0," instead of "64,"

Take the following example:
    // Endpoint Descriptor //
    0x07,                       //the size of this descriptor //
    USB_DESCRIPTOR_ENDPOINT,    //Endpoint Descriptor
    _EP02_IN,                   //EndpointAddress
    _INT,                       //Attributes
    0x08,0x00,                  //size (note: 2 bytes)
    0x02,                       //Interval

The first two parameters are self-explanatory. They specify the
length of this endpoint descriptor (7) and the descriptor type.
The next parameter identifies the endpoint, the definitions are
defined in usb_device.h and has the following naming
convention:
_EP<##>_<dir>
where ## is the endpoint number and dir is the direction of
transfer. The dir has the value of either 'OUT' or 'IN'.
The next parameter identifies the type of the endpoint. Available
options are _BULK, _INT, _ISO, and _CTRL. The _CTRL is not
typically used because the default control transfer endpoint is
not defined in the USB descriptors. When _ISO option is used,
addition options can be ORed to _ISO. Example:
_ISO|_AD|_FE
This describes the endpoint as an isochronous pipe with adaptive
and feedback attributes. See usb_device.h and the USB
specification for details. The next parameter defines the size of
the endpoint. The last parameter in the polling interval.

-----------------------------------------------------------------------
Adding a USB String
-----------------------------------------------------------------------
A string descriptor array should have the following format:

rom struct{byte bLength;byte bDscType;word string[size];}sdxxx={
sizeof(sdxxx),DSC_STR,<text>};

The above structure provides a means for the C compiler to
calculate the length of string descriptor sdxxx, where xxx is the
index number. The first two bytes of the descriptor are descriptor
length and type. The rest <text> are string texts which must be
in the unicode format. The unicode format is achieved by declaring
each character as a word type. The whole text string is declared
as a word array with the number of characters equals to <size>.
<size> has to be manually counted and entered into the array
declaration. Let's study this through an example:
if the string is "USB" , then the string descriptor should be:
(Using index 02)
rom struct{byte bLength;byte bDscType;word string[3];}sd002={
sizeof(sd002),DSC_STR,'U','S','B'};

A USB project may have multiple strings and the firmware supports
the management of multiple strings through a look-up table.
The look-up table is defined as:
rom const unsigned char *rom USB_SD_Ptr[]={&sd000,&sd001,&sd002};

The above declaration has 3 strings, sd000, sd001, and sd002.
Strings can be removed or added. sd000 is a specialized string
descriptor. It defines the language code, usually this is
US English (0x0409). The index of the string must match the index
position of the USB_SD_Ptr array, &sd000 must be in position
USB_SD_Ptr[0], &sd001 must be in position USB_SD_Ptr[1] and so on.
The look-up table USB_SD_Ptr is used by the get string handler
function.
```

```
-------------------------------------------------------------------

The look-up table scheme also applies to the configuration
descriptor. A USB device may have multiple configuration
descriptors, i.e. CFG01, CFG02, etc. To add a configuration
descriptor, user must implement a structure similar to CFG01.
The next step is to add the configuration descriptor name, i.e.
cfg01, cfg02,.., to the look-up table USB_CD_Ptr. USB_CD_Ptr[0]
is a dummy place holder since configuration 0 is the un-configured
state according to the definition in the USB specification.

*******************************************************************/

/*******************************************************************
 * Descriptor specific type definitions are defined in:
 * usb_device.h
 *
 * Configuration options are defined in:
 * usb_config.h
 *******************************************************************/
#ifndef __USB_DESCRIPTORS_C
#define __USB_DESCRIPTORS_C

/** INCLUDES *******************************************************/
#include "GenericTypeDefs.h"
#include "Compiler.h"
#include "/usb/usb.h"


/** CONSTANTS ******************************************************/
#if defined(__18CXX)
#pragma romdata
#endif

/* Device Descriptor */
ROM USB_DEVICE_DESCRIPTOR device_dsc=
{
    0x12,                   // Size of this descriptor in bytes
    USB_DESCRIPTOR_DEVICE,  // DEVICE descriptor type
    0x0200,                 // USB Spec Release Number in BCD format
    0x00,                   // Class Code
    0x00,                   // Subclass code
    0x00,                   // Protocol code
    USB_EP0_BUFF_SIZE,          // Max packet size for EP0, see usb_config.h
    0x04D8,                 // Vendor ID
    0x000C,                 // Product ID: PICDEM FS USB (DEMO Mode)
    0x0000,                 // Device release number in BCD format
    0x01,                   // Manufacturer string index
    0x02,                   // Product string index
    0x00,                   // Device serial number string index
    0x01                    // Number of possible configurations
};

/* Configuration 1 Descriptor */
ROM BYTE configDescriptor1[]={
    /* Configuration Descriptor */
    0x09,//sizeof(USB_CFG_DSC),    // Size of this descriptor in bytes
    USB_DESCRIPTOR_CONFIGURATION,               // CONFIGURATION descriptor type
    0x20,0x00,            // Total length of data for this cfg
    1,                      // Number of interfaces in this cfg
    1,                      // Index value of this configuration
    0,                      // Configuration string index
    _DEFAULT | _SELF,               // Attributes, see usb_device.h
    50,                     // Max power consumption (2X mA)

    /* Interface Descriptor */
    0x09,//sizeof(USB_INTF_DSC),   // Size of this descriptor in bytes
```

```
    USB_DESCRIPTOR_INTERFACE,              // INTERFACE descriptor type
    0,                      // Interface Number
    0,                      // Alternate Setting Number
    2,                      // Number of endpoints in this intf
    0x00,                   // Class code
    0x00,                   // Subclass code
    0x00,                   // Protocol code
    0,                      // Interface string index

    /* Endpoint Descriptor */
    0x07,                        /*sizeof(USB_EP_DSC)*/
    USB_DESCRIPTOR_ENDPOINT,    //Endpoint Descriptor
    _EP01_OUT,                  //EndpointAddress
    _BULK,                       //Attributes
    USBGEN_EP_SIZE,0x00,        //size
    1,                          //Interval

    0x07,                        /*sizeof(USB_EP_DSC)*/
    USB_DESCRIPTOR_ENDPOINT,    //Endpoint Descriptor
    _EP01_IN,                   //EndpointAddress
    _BULK,                       //Attributes
    USBGEN_EP_SIZE,0x00,        //size
    1                           //Interval
};


//Language code string descriptor
ROM struct{BYTE bLength;BYTE bDscType;WORD string[1];}sd000={
sizeof(sd000),USB_DESCRIPTOR_STRING,{0x0409}};

//Manufacturer string descriptor
ROM struct{BYTE bLength;BYTE bDscType;WORD string[25];}sd001={
sizeof(sd001),USB_DESCRIPTOR_STRING,
{'M','i','c','r','o','c','h','i','p',' ',
'T','e','c','h','n','o','l','o','g','y',' ','I','n','c','.'
}};

//Product string descriptor
ROM struct{BYTE bLength;BYTE bDscType;WORD string[27];}sd002={
sizeof(sd002),USB_DESCRIPTOR_STRING,
{'M','i','c','r','o','c','h','i','p',' ','C','u','s','t','o','m',
' ','U','S','B',' ','D','e','v','i','c','e'}};

//Array of configuration descriptors
ROM BYTE *ROM USB_CD_Ptr[]=
{
    (ROM BYTE *ROM)&configDescriptor1
};
//Array of string descriptors
ROM BYTE *ROM USB_SD_Ptr[]=
{
    (ROM BYTE *ROM)&sd000,
    (ROM BYTE *ROM)&sd001,
    (ROM BYTE *ROM)&sd002
};

/** EOF usb_descriptors.c ****************************************************/

#endif
```

6.1.18. usb_device.c

```
/********************************************************************
  File Information:
    FileName:       usb_device.c
    Dependencies:   See INCLUDES section
    Processor:            PIC18 or PIC24 USB Microcontrollers
    Hardware:       The code is natively intended to be used on the following
                          hardware platforms: PICDEM™ FS USB Demo Board,
                          PIC18F87J50 FS USB Plug-In Module, or
                          Explorer 16 + PIC24 USB PIM.  The firmware may be
                          modified for use on other USB platforms by editing the
                          HardwareProfile.h file.
    Complier:       Microchip C18 (for PIC18) or C30 (for PIC24)
    Company:        Microchip Technology, Inc.

    Software License Agreement:

    The software supplied herewith by Microchip Technology Incorporated
    (the “Company”) for its PIC® Microcontroller is intended and
    supplied to you, the Company’s customer, for use solely and
    exclusively on Microchip PIC Microcontroller products. The
    software is owned by the Company and/or its supplier, and is
    protected under applicable copyright laws. All rights are reserved.
    Any use in violation of the foregoing restrictions may subject the
    user to criminal sanctions under applicable laws, as well as to
    civil liability for the breach of the terms and conditions of this
    license.

    THIS SOFTWARE IS PROVIDED IN AN “AS IS” CONDITION. NO WARRANTIES,
    WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
    TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
    PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
    IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
    CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

  Summary:
    This file contains functions, macros, definitions, variables,
    datatypes, etc. that are required for usage with the MCHPFSUSB device
    stack. This file should be included in projects that use the device stack.

    This file is located in the "\<Install Directory\>\\Microchip\\USB"
    directory.

  Description:
    USB Device Stack File

    This file contains functions, macros, definitions, variables,
    datatypes, etc. that are required for usage with the MCHPFSUSB device
    stack. This file should be included in projects that use the device stack.

    This file is located in the "\<Install Directory\>\\Microchip\\USB"
    directory.

    When including this file in a new project, this file can either be
    referenced from the directory in which it was installed or copied
    directly into the user application folder. If the first method is
    chosen to keep the file located in the folder in which it is installed
    then include paths need to be added so that the library and the
    application both know where to reference each others files. If the
    application folder is located in the same folder as the Microchip
    folder (like the current demo folders), then the following include
    paths need to be added to the application's project:

    ..\\Include

    ..\\..\\Include
```

```
    ..\\..\\MicrochipInclude

    ..\\..\\\<Application Folder\>

    ..\\..\\..\\\<Application Folder\>

    If a different directory structure is used, modify the paths as
    required. An example using absolute paths instead of relative paths
    would be the following:

    C:\\Microchip Solutions\\Microchip\\Include

    C:\\Microchip Solutions\\My Demo Application

********************************************************************/
#ifndef USB_DEVICE_C
#define USB_DEVICE_C
#endif

/** INCLUDES *******************************************************/
#include "GenericTypeDefs.h"
#include "Compiler.h"
#include "/usb/usb.h"
#include "HardwareProfile.h"


#if defined(USB_USE_MSD)
    #include "./USB/usb_function_msd.h"
#endif


BOOL USER_USB_CALLBACK_EVENT_HANDLER(USB_EVENT event, void *pdata, WORD size);

#if defined(__C32__)
    #if (USB_PING_PONG_MODE != USB_PING_PONG__FULL_PING_PONG)
        #error "PIC32 only supports full ping pong mode.  A different mode other
than full ping pong is selected in the usb_config.h file."
    #endif
#endif

//#define DEBUG_MODE

#ifdef DEBUG_MODE
    #include "uart2.h"
#endif

/** VARIABLES ******************************************************/
#pragma udata

USB_VOLATILE BYTE USBDeviceState;
USB_VOLATILE BYTE USBActiveConfiguration;
USB_VOLATILE BYTE USBAlternateInterface[USB_MAX_NUM_INT];
volatile BDT_ENTRY *pBDTEntryEP0OutCurrent;
volatile BDT_ENTRY *pBDTEntryEP0OutNext;
volatile BDT_ENTRY *pBDTEntryOut[USB_MAX_EP_NUMBER+1];
volatile BDT_ENTRY *pBDTEntryIn[USB_MAX_EP_NUMBER+1];
USB_VOLATILE BYTE shortPacketStatus;
USB_VOLATILE BYTE controlTransferState;
USB_VOLATILE IN_PIPE inPipes[1];
USB_VOLATILE OUT_PIPE outPipes[1];
USB_VOLATILE BYTE *pDst;
USB_VOLATILE BOOL RemoteWakeup;
USB_VOLATILE BYTE USTATcopy;
USB_VOLATILE WORD USBInMaxPacketSize[USB_MAX_EP_NUMBER];
USB_VOLATILE BYTE *USBInData[USB_MAX_EP_NUMBER];
```

```
/** USB FIXED LOCATION VARIABLES ********************************/
#if defined(__18CXX)
    #if defined(__18F14K50) || defined(__18F13K50) || defined(__18LF14K50) ||
defined(__18LF13K50)
        #pragma udata USB_BDT=0x200     //See Linker Script,usb2:0x200-0x2FF(256-
byte)
    #else
        #pragma udata USB_BDT=0x400     //See Linker Script,usb4:0x400-0x4FF(256-
byte)
    #endif
#endif

/*****************************************************************
 * Section A: Buffer Descriptor Table
 * - 0x400 - 0x4FF(max)
 * - USB_MAX_EP_NUMBER is defined in usb_config.h
 *****************************************************************/
#if (USB_PING_PONG_MODE == USB_PING_PONG__NO_PING_PONG)
    volatile BDT_ENTRY BDT[(USB_MAX_EP_NUMBER + 1) * 2] __attribute__ ((aligned
(512)));
#elif (USB_PING_PONG_MODE == USB_PING_PONG__EP0_OUT_ONLY)
    volatile BDT_ENTRY BDT[((USB_MAX_EP_NUMBER + 1) * 2)+1] __attribute__
((aligned (512)));
#elif (USB_PING_PONG_MODE == USB_PING_PONG__FULL_PING_PONG)
    volatile BDT_ENTRY BDT[(USB_MAX_EP_NUMBER + 1) * 4] __attribute__ ((aligned
(512)));
#elif (USB_PING_PONG_MODE == USB_PING_PONG__ALL_BUT_EP0)
    volatile BDT_ENTRY BDT[((USB_MAX_EP_NUMBER + 1) * 4)-2] __attribute__
((aligned (512)));
#else
    #error "No ping pong mode defined."
#endif

//#if defined(__18CXX)
//#pragma udata usbram5=0x400     //See Linker Script,usb5:0x500-0x5FF(256-byte)
//#endif

/*****************************************************************
 * Section B: EP0 Buffer Space
 *****************************************************************/
volatile CTRL_TRF_SETUP SetupPkt;           // 8-byte only
volatile BYTE CtrlTrfData[USB_EP0_BUFF_SIZE];

/*****************************************************************
 * Section C: non-EP0 Buffer Space
 *****************************************************************/
// Can provide compile time option to do software pingpong
#if defined(USB_USE_HID)
    volatile unsigned char hid_report_out[HID_INT_OUT_EP_SIZE];
    volatile unsigned char hid_report_in[HID_INT_IN_EP_SIZE];
#endif

#if defined(USB_USE_MSD)
      //volatile far USB_MSD_CBW_CSW msd_cbw_csw;
      volatile USB_MSD_CBW msd_cbw;
      volatile USB_MSD_CSW msd_csw;
      //#pragma udata

      #if defined(__18CXX)
            #pragma udata myMSD=MSD_BUFFER_ADDRESS
      #endif
      volatile char msd_buffer[512];
#endif

#if defined(__18CXX)
#pragma udata
#endif
```

```
/** DECLARATIONS ***************************************************/
#pragma code

//DOM-IGNORE-BEGIN
/*********************************************************************
  Function:
    void USBDeviceInit(void)

  Description:
    This function initializes the device stack
    it in the default state

  Precondition:
    None

  Parameters:
    None

  Return Values:
    None

  Remarks:
    The USB module will be completely reset including
    all of the internal variables, registers, and
    interrupt flags.
  *********************************************************************/
//DOM-IGNORE-END
void USBDeviceInit(void)
{
    BYTE i;

    // Clear all USB error flags
    USBClearInterruptRegister(U1EIR);

    // Clears all USB interrupts
    USBClearInterruptRegister(U1IR);
    #if defined(USB_POLLING)

    U1EIE = 0x9F;                   // Unmask all USB error interrupts
    U1IE = 0xFB;                    // Enable all interrupts except ACTVIE
    #endif

    //power up the module
    USBPowerModule();

    //set the address of the BDT (if applicable)
    USBSetBDTAddress(BDT);

    // Reset all of the Ping Pong buffers
    USBPingPongBufferReset = 1;
    USBPingPongBufferReset = 0;

    // Reset to default address
    U1ADDR = 0x00;

    //Clear all of the endpoint control registers
    memset((void*)&U1EP1,0x00,(USB_MAX_EP_NUMBER-1));

    //Clear all of the BDT entries
    for(i=0;i<(sizeof(BDT)/sizeof(BDT_ENTRY));i++)
    {
        BDT[i].Val = 0x00;
    }

    // Initialize EP0 as a Ctrl EP
    U1EP0 = EP_CTRL|USB_HANDSHAKE_ENABLED;
```

```
    // Flush any pending transactions
    while(USBTransactionCompleteIF == 1)
    {

USBClearInterruptFlag(USBTransactionCompleteIFReg,USBTransactionCompleteIFBitNum)
;
    }

    //clear all of the internal pipe information
    inPipes[0].info.Val = 0;
    outPipes[0].info.Val = 0;
    outPipes[0].wCount.Val = 0;

    // Make sure packet processing is enabled
    USBPacketDisable = 0;

    //Get ready for the first packet
    pBDTEntryIn[0] = (volatile BDT_ENTRY*)&BDT[EP0_IN_EVEN];

    // Clear active configuration
    USBActiveConfiguration = 0;

    //Indicate that we are now in the detached state
    USBDeviceState = DETACHED_STATE;
}

//DOM-IGNORE-BEGIN
/*******************************************************************************
  Function:
    void USBDeviceTasks(void)

  Description:
    This function is the main state machine of the
    USB device side stack.  This function should be
    called periodically to receive and transmit
    packets through the stack.  This function should
    be called  preferably once every 100us
    during the enumeration process.  After the
    enumeration process this function still needs to
    be called periodically to respond to various
    situations on the bus but is more relaxed in its
    time requirements.  This function should also
    be called at least as fast as the OUT data
    expected from the PC.

  Precondition:
    None

  Parameters:
    None

  Return Values:
    None

  Remarks:
    None
  *****************************************************************************/
//DOM-IGNORE-END
#if defined(USB_INTERRUPT)
  #if defined(__18CXX)
    void USBDeviceTasks(void)
  #elif defined(__C30__)
    //void __attribute__((interrupt,auto_psv,address(0xA800))) _USB1Interrupt()
    void __attribute__((interrupt,auto_psv)) _USB1Interrupt()
  #elif defined(__PIC32MX__)
    #pragma interrupt _USB1Interrupt ipl4 vector 45
```

```
    void _USB1Interrupt( void )
  #endif
#else
void USBDeviceTasks(void)
#endif
{
    BYTE i;

    USBClearUSBInterrupt();

#ifdef USB_SUPPORT_OTG
    //SRP Time Out Check
    if (USBOTGSRPIsReady())
    {
        if (USBT1MSECIF && USBT1MSECIE)
        {
            if (USBOTGGetSRPTimeOutFlag())
            {
                if (USBOTGIsSRPTimeOutExpired())
                {
                    USB_OTGEventHandler(0,OTG_EVENT_SRP_FAILED,0,0);
                }
            }

            //Clear Interrupt Flag
            USBClearInterruptFlag(USBT1MSECIFReg,USBT1MSECIFBitNum);
        }
    }
#endif

    #if defined(USB_POLLING)
    //If the interrupt option is selected then the customer is required
    //  to notify the stack when the device is attached or removed from the
    //  bus by calling the USBDeviceAttach() and USBDeviceDetach() functions.
    if (USB_BUS_SENSE != 1)
    {
         // Disable module & detach from bus
         U1CON = 0;

         // Mask all USB interrupts
         U1IE = 0;

         //Move to the detached state
         USBDeviceState = DETACHED_STATE;

         #ifdef  USB_SUPPORT_OTG
             //Disable D+ Pullup
             U1OTGCONbits.DPPULUP = 0;

             //Disable HNP
             USBOTGDisableHnp();

             //Deactivate HNP
             USBOTGDeactivateHnp();

             //If ID Pin Changed State
             if (USBIDIF && USBIDIE)
             {
                 //Re-detect & Initialize
                  USBOTGInitialize();

                  //Clear ID Interrupt Flag
                  USBClearInterruptFlag(USBIDIFReg,USBIDIFBitNum);
             }
         #endif

         #ifdef __C30__
```

```
            //USBClearInterruptFlag(U1OTGIR, 3);
        #endif
           //return so that we don't go through the rest of
           //the state machine
         return;
    }

#ifdef USB_SUPPORT_OTG
    //If Session Is Started Then
   else
   {
        //If SRP Is Ready
        if (USBOTGSRPIsReady())
        {
            //Clear SRPReady
            USBOTGClearSRPReady();

            //Clear SRP Timeout Flag
            USBOTGClearSRPTimeOutFlag();

            //Indicate Session Started
            UART2PrintString( "\r\n***** USB OTG B Event - Session Started
*****\r\n" );
        }
    }
#endif

    //if we are in the detached state
    if(USBDeviceState == DETACHED_STATE)
    {
        //#if defined(__18CXX)
            U1CON = 0;                          // Disable module & detach from
bus
        //#else
        //#endif

        // Mask all USB interrupts
        U1IE = 0;

        // Enable module & attach to bus
        while(!U1CONbits.USBEN){U1CONbits.USBEN = 1;}

        //moved to the attached state
        USBDeviceState = ATTACHED_STATE;

        //Enable/set things like: pull ups, full/low-speed mode,
        //set the ping pong mode, and set internal transceiver
        SetConfigurationOptions();

        #ifdef  USB_SUPPORT_OTG
            U1OTGCON |= USB_OTG_DPLUS_ENABLE | USB_OTG_ENABLE;
        #endif
    }
#endif  //#if defined(USB_POLLING)

    if(USBDeviceState == ATTACHED_STATE)
    {
        /*
         * After enabling the USB module, it takes some time for the
         * voltage on the D+ or D- line to rise high enough to get out
         * of the SE0 condition. The USB Reset interrupt should not be
         * unmasked until the SE0 condition is cleared. This helps
         * prevent the firmware from misinterpreting this unique event
         * as a USB bus reset from the USB host.
         */

        if(!USBSE0Event)
```

```
        {
            USBClearInterruptRegister(U1IR);// Clear all USB interrupts
            #if defined(USB_POLLING)
                U1IE=0;                         // Mask all USB interrupts
            #endif
            USBResetIE = 1;             // Unmask RESET interrupt
            USBIdleIE = 1;             // Unmask IDLE interrupt
            USBDeviceState = POWERED_STATE;
        }
    }

    #ifdef  USB_SUPPORT_OTG
        //If ID Pin Changed State
        if (USBIDIF && USBIDIE)
        {
            //Re-detect & Initialize
            USBOTGInitialize();

            USBClearInterruptFlag(USBIDIFReg,USBIDIFBitNum);
        }
    #endif

    /*
     * Task A: Service USB Activity Interrupt
     */
    if(USBActivityIF && USBActivityIE)
    {
        USBClearInterruptFlag(USBActivityIFReg,USBActivityIFBitNum);
        #if defined(USB_SUPPORT_OTG)
            U1OTGIR = 0x10;
        #else
            USBWakeFromSuspend();
        #endif
    }

    /*
     * Pointless to continue servicing if the device is in suspend mode.
     */
    if(USBSuspendControl==1)
    {
        return;
    }

    /*
     * Task B: Service USB Bus Reset Interrupt.
     * When bus reset is received during suspend, ACTVIF will be set first,
     * once the UCONbits.SUSPND is clear, then the URSTIF bit will be asserted.
     * This is why URSTIF is checked after ACTVIF.
     *
     * The USB reset flag is masked when the USB state is in
     * DETACHED_STATE or ATTACHED_STATE, and therefore cannot
     * cause a USB reset event during these two states.
     */
    if(USBResetIF && USBResetIE)
    {
        USBDeviceInit();
        USBDeviceState = DEFAULT_STATE;

        /*******************************************************************
        Bug Fix: Feb 26, 2007 v2.1 (#F1)
        *******************************************************************
        In the original firmware, if an OUT token is sent by the host
        before a SETUP token is sent, the firmware would respond with an ACK.
        This is not a correct response, the firmware should have sent a STALL.
        This is a minor non-compliance since a compliant host should not
        send an OUT before sending a SETUP token. The fix allows a SETUP
        transaction to be accepted while stalling OUT transactions.
```

```
        ********************************************************************/
        BDT[EP0_OUT_EVEN].ADR = (BYTE*)ConvertToPhysicalAddress(&SetupPkt);
        BDT[EP0_OUT_EVEN].CNT = USB_EP0_BUFF_SIZE;
        BDT[EP0_OUT_EVEN].STAT.Val &= ~_STAT_MASK;
        BDT[EP0_OUT_EVEN].STAT.Val |= _USIE|_DAT0|_DTSEN|_BSTALL;

        #ifdef USB_SUPPORT_OTG
             //Disable HNP
             USBOTGDisableHnp();

             //Deactivate HNP
             USBOTGDeactivateHnp();
        #endif

        USBClearInterruptFlag(USBResetIFReg,USBResetIFBitNum);
    }

    /*
     * Task C: Service other USB interrupts
     */
    if(USBIdleIF && USBIdleIE)
    {
        #ifdef  USB_SUPPORT_OTG
            //If Suspended, Try to switch to Host
            USBOTGSelectRole(ROLE_HOST);
        #else
            USBSuspend();
        #endif

        USBClearInterruptFlag(USBIdleIFReg,USBIdleIFBitNum);
    }

    if(USBSOFIF && USBSOFIE)
    {
        USB_SOF_HANDLER(EVENT_SOF,0,1);
        USBClearInterruptFlag(USBSOFIFReg,USBSOFIFBitNum);
    }

    if(USBStallIF && USBStallIE)
    {
        USBStallHandler();
    }

    if(USBErrorIF && USBErrorIE)
    {
        USB_ERROR_HANDLER(EVENT_BUS_ERROR,0,1);
        USBClearInterruptRegister(U1EIR);               // This clears UERRIF
    }

    /*
     * Pointless to continue servicing if the host has not sent a bus reset.
     * Once bus reset is received, the device transitions into the DEFAULT
     * state and is ready for communication.
     */
    if(USBDeviceState < DEFAULT_STATE) return;

    /*
     * Task D: Servicing USB Transaction Complete Interrupt
     */
    if(USBTransactionCompleteIE)
    {
          for(i = 0; i < 4; i++)//Drain or deplete the USAT FIFO entries.  If
the USB FIFO ever gets full, USB bandwidth
            {                                       //utilization can be
compromised, and the device won't be able to receive SETUP packets.
                if(USBTransactionCompleteIF)
                {
```

```
                    USTATcopy = U1STAT;


USBClearInterruptFlag(USBTransactionCompleteIFReg,USBTransactionCompleteIFBitNum)
;

                    /*
                     * USBCtrlEPService only services transactions over EP0.
                     * It ignores all other EP transactions.
                     */

                if((USTATcopy & ENDPOINT_MASK) == 0)
                {
                        USBCtrlEPService();
                }
                else
                {
                    USB_TRASFER_COMPLETE_HANDLER(
                        EVENT_TRANSFER,
                        (BYTE*)&USTATcopy,
                        0);
                }
                 }//end if(USBTransactionCompleteIF)
                 else
                    break; //USTAT FIFO must be empty.
             }//end for()
       }//end if(USBTransactionCompleteIE)
}//end of USBDeviceTasks()

/********************************************************************
 * Function:        void USBStallHandler(void)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:
 *
 * Overview:        This function handles the event of a STALL
 *                  occuring on the bus
 *
 * Note:            None
 *******************************************************************/
void USBStallHandler(void)
{
    /*
     * Does not really have to do anything here,
     * even for the control endpoint.
     * All BDs of Endpoint 0 are owned by SIE right now,
     * but once a Setup Transaction is received, the ownership
     * for EP0_OUT will be returned to CPU.
     * When the Setup Transaction is serviced, the ownership
     * for EP0_IN will then be forced back to CPU by firmware.
     */

    /* v2b fix */
    if(U1EP0bits.EPSTALL == 1)
    {
        // UOWN - if 0, owned by CPU, if 1, owned by SIE
        if((pBDTEntryEP0OutCurrent->STAT.Val == _USIE) && (pBDTEntryIn[0]-
>STAT.Val == (_USIE|_BSTALL)))
        {
            // Set ep0Bo to stall also
            pBDTEntryEP0OutCurrent->STAT.Val = _USIE|_DAT0|_DTSEN|_BSTALL;
        }//end if
```

```
        U1EP0bits.EPSTALL = 0;               // Clear stall status
    }//end if

    USBClearInterruptFlag(USBStallIFReg,USBStallIFBitNum);
}

/*********************************************************************
 * Function:        void USBSuspend(void)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:
 *
 * Overview:        This function handles if the host tries to
 *                  suspend the device
 *
 * Note:            None
 ********************************************************************/
void USBSuspend(void)
{
    /*
     * NOTE: Do not clear UIRbits.ACTVIF here!
     * Reason:
     * ACTVIF is only generated once an IDLEIF has been generated.
     * This is a 1:1 ratio interrupt generation.
     * For every IDLEIF, there will be only one ACTVIF regardless of
     * the number of subsequent bus transitions.
     *
     * If the ACTIF is cleared here, a problem could occur when:
     * [       IDLE       ][bus activity ->
     * <--- 3 ms ----->     ^
     *                ^     ACTVIF=1
     *                IDLEIF=1
     *  #           #           #           #   (#=Program polling flags)
     *                          ^
     *                          This polling loop will see both
     *                          IDLEIF=1 and ACTVIF=1.
     *                          However, the program services IDLEIF first
     *                          because ACTIVIE=0.
     *                          If this routine clears the only ACTIVIF,
     *                          then it can never get out of the suspend
     *                          mode.
     */
    USBActivityIE = 1;                      // Enable bus activity interrupt
    USBClearInterruptFlag(USBIdleIFReg,USBIdleIFBitNum);

#if defined(__18CXX)
    U1CONbits.SUSPND = 1;                    // Put USB module in power conserve
                                             // mode, SIE clock inactive
#endif


    /*
     * At this point the PIC can go into sleep,idle, or
     * switch to a slower clock, etc.  This should be done in the
     * USBCBSuspend() if necessary.
     */
    USB_SUSPEND_HANDLER(EVENT_SUSPEND,0,0);
}

/*********************************************************************
 * Function:        void USBWakeFromSuspend(void)
 *
```

```
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:
 *
 * Note:            None
 *******************************************************************/
void USBWakeFromSuspend(void)
{
    #if defined(__18CXX)
    U1CONbits.SUSPND = 0;                   // Bring USB module out of power
conserve
                                            // mode.
    #endif

    /*
     * If using clock switching, the place to restore the original
     * microcontroller core clock frequency is in the USBCBWakeFromSuspend()
callback
     */
    USB_WAKEUP_FROM_SUSPEND_HANDLER(EVENT_RESUME,0,0);

    USBActivityIE = 0;

    /********************************************************************
    Bug Fix: Feb 26, 2007 v2.1
    ********************************************************************
    The ACTVIF bit cannot be cleared immediately after the USB module wakes
    up from Suspend or while the USB module is suspended. A few clock cycles
    are required to synchronize the internal hardware state machine before
    the ACTIVIF bit can be cleared by firmware. Clearing the ACTVIF bit
    before the internal hardware is synchronized may not have an effect on
    the value of ACTVIF. Additonally, if the USB module uses the clock from
    the 96 MHz PLL source, then after clearing the SUSPND bit, the USB
    module may not be immediately operational while waiting for the 96 MHz
    PLL to lock.
    ********************************************************************/

    // UIRbits.ACTVIF = 0;                      // Removed
    #if defined(__18CXX)
    while(USBActivityIF)
    #endif
    {
        USBClearInterruptFlag(USBActivityIFReg,USBActivityIFBitNum);
    }  // Added

}//end USBWakeFromSuspend

/********************************************************************
 * Function:        void USBCtrlEPService(void)
 *
 * PreCondition:    USTAT is loaded with a valid endpoint address.
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        USBCtrlEPService checks for three transaction
 *                  types that it knows how to service and services
 *                  them:
```

```
 *                  1. EP0 SETUP
 *                  2. EP0 OUT
 *                  3. EP0 IN
 *                  It ignores all other types (i.e. EP1, EP2, etc.)
 *
 * Note:            None
 *******************************************************************/
void USBCtrlEPService(void)
{
       //If the last packet was a EP0 OUT packet
    if((USTATcopy & USTAT_EP0_PP_MASK) == USTAT_EP0_OUT_EVEN)
    {
             //Point to the EP0 OUT buffer of the buffer that arrived
        #if defined(__18CXX)
            pBDTEntryEP0OutCurrent = (volatile BDT_ENTRY*)&BDT[(USTATcopy &
USTAT_EP_MASK)>>1];
        #elif defined(__C30__) || defined(__C32__)
            pBDTEntryEP0OutCurrent = (volatile BDT_ENTRY*)&BDT[(USTATcopy &
USTAT_EP_MASK)>>2];
        #else
            #error "unimplemented"
        #endif

             //Set the next out to the current out packet
        pBDTEntryEP0OutNext = pBDTEntryEP0OutCurrent;
             //Toggle it to the next ping pong buffer (if applicable)
        ((BYTE_VAL*)&pBDTEntryEP0OutNext)->Val ^= USB_NEXT_EP0_OUT_PING_PONG;

             //If the current EP0 OUT buffer has a SETUP token
        if(pBDTEntryEP0OutCurrent->STAT.PID == SETUP_TOKEN)
        {
                    //Handle the control transfer
            USBCtrlTrfSetupHandler();
        }
        else
        {
                    //Handle the DATA transfer
            USBCtrlTrfOutHandler();
        }
    }
    else if((USTATcopy & USTAT_EP0_PP_MASK) == USTAT_EP0_IN)
    {
             //Otherwise the transmission was and EP0 IN
             //  so take care of the IN transfer
        USBCtrlTrfInHandler();
    }

}//end USBCtrlEPService

/*******************************************************************
 * Function:        void USBCtrlTrfSetupHandler(void)
 *
 * PreCondition:    SetupPkt buffer is loaded with valid USB Setup Data
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        This routine is a task dispatcher and has 3 stages.
 *                  1. It initializes the control transfer state machine.
 *                  2. It calls on each of the module that may know how to
 *                     service the Setup Request from the host.
 *                     Module Example: USBD, HID, CDC, MSD, ...
 *                     A callback function, USBCBCheckOtherReq(),
 *                     is required to call other module handlers.
```

```
 *                  3. Once each of the modules has had a chance to check if
 *                     it is responsible for servicing the request, stage 3
 *                     then checks direction of the transfer to determine how
 *                     to prepare EP0 for the control transfer.
 *                     Refer to USBCtrlEPServiceComplete() for more details.
 *
 * Note:            Microchip USB Firmware has three different states for
 *                  the control transfer state machine:
 *                  1. WAIT_SETUP
 *                  2. CTRL_TRF_TX
 *                  3. CTRL_TRF_RX
 *                  Refer to firmware manual to find out how one state
 *                  is transitioned to another.
 *
 *                  A Control Transfer is composed of many USB transactions.
 *                  When transferring data over multiple transactions,
 *                  it is important to keep track of data source, data
 *                  destination, and data count. These three parameters are
 *                  stored in pSrc,pDst, and wCount. A flag is used to
 *                  note if the data source is from ROM or RAM.
 *
 *****************************************************************/
void USBCtrlTrfSetupHandler(void)
{
       //if the SIE currently owns the buffer
    if(pBDTEntryIn[0]->STAT.UOWN != 0)
    {
              //give control back to the CPU
              //  Compensate for after a STALL
        pBDTEntryIn[0]->STAT.Val = _UCPU;
    }

      //Keep track of if a short packet has been sent yet or not
    shortPacketStatus = SHORT_PKT_NOT_USED;

    /* Stage 1 */
    controlTransferState = WAIT_SETUP;

    inPipes[0].wCount.Val = 0;
    inPipes[0].info.Val = 0;

    /* Stage 2 */
    USBCheckStdRequest();
    USB_OTHER_REQUEST_HANDLER(EVENT_EP0_REQUEST,0,0);

    /* Stage 3 */
    USBCtrlEPServiceComplete();

}//end USBCtrlTrfSetupHandler
/*****************************************************************************
 * Function:        void USBCtrlTrfOutHandler(void)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        This routine handles an OUT transaction according to
 *                  which control transfer state is currently active.
 *
 * Note:            Note that if the the control transfer was from
 *                  host to device, the session owner should be notified
 *                  at the end of each OUT transaction to service the
 *                  received data.
```

```
 *
 *******************************************************************************/
void USBCtrlTrfOutHandler(void)
{
    if(controlTransferState == CTRL_TRF_RX)
    {
        USBCtrlTrfRxService();
    }
    else    // CTRL_TRF_TX
    {
        USBPrepareForNextSetupTrf();
    }
}

/*******************************************************************************
 * Function:        void USBCtrlTrfInHandler(void)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        This routine handles an IN transaction according to
 *                  which control transfer state is currently active.
 *
 *
 * Note:            A Set Address Request must not change the acutal address
 *                  of the device until the completion of the control
 *                  transfer. The end of the control transfer for Set Address
 *                  Request is an IN transaction. Therefore it is necessary
 *                  to service this unique situation when the condition is
 *                  right. Macro mUSBCheckAdrPendingState is defined in
 *                  usb9.h and its function is to specifically service this
 *                  event.
 *******************************************************************************/
void USBCtrlTrfInHandler(void)
{
    BYTE lastDTS;

    lastDTS = pBDTEntryIn[0]->STAT.DTS;

    //switch to the next ping pong buffer
    ((BYTE_VAL*)&pBDTEntryIn[0])->Val ^= USB_NEXT_EP0_IN_PING_PONG;

    //mUSBCheckAdrPendingState();       // Must check if in ADR_PENDING_STATE
    if(USBDeviceState == ADR_PENDING_STATE)
    {
        U1ADDR = SetupPkt.bDevADR.Val;
        if(U1ADDR > 0)
        {
            USBDeviceState=ADDRESS_STATE;
        }
        else
        {
            USBDeviceState=DEFAULT_STATE;
        }
    }//end if


    if(controlTransferState == CTRL_TRF_TX)
    {
        pBDTEntryIn[0]->ADR = (BYTE *)ConvertToPhysicalAddress(CtrlTrfData);
        USBCtrlTrfTxService();
```

```
        /* v2b fix */
        if(shortPacketStatus == SHORT_PKT_SENT)
        {
            // If a short packet has been sent, don't want to send any more,
            // stall next time if host is still trying to read.
            pBDTEntryIn[0]->STAT.Val = _USIE|_BSTALL;
        }
        else
        {
            if(lastDTS == 0)
            {
                pBDTEntryIn[0]->STAT.Val = _USIE|_DAT1|_DTSEN;
            }
            else
            {
                pBDTEntryIn[0]->STAT.Val = _USIE|_DAT0|_DTSEN;
            }
        }//end if(...)else
    }
    else // CTRL_TRF_RX
    {
        USBPrepareForNextSetupTrf();
    }
}

/*********************************************************************
 * Function:        void USBPrepareForNextSetupTrf(void)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        The routine forces EP0 OUT to be ready for a new
 *                  Setup transaction, and forces EP0 IN to be owned
 *                  by CPU.
 *
 * Note:            None
 ********************************************************************/
void USBPrepareForNextSetupTrf(void)
{
    /*******************************************************************
    Bug Fix: Feb 26, 2007 v2.1
    *******************************************************************
    Facts:
    A Setup Packet should never be stalled. (USB 2.0 Section 8.5.3)
    If a Setup PID is detected by the SIE, the DTSEN setting is ignored.
    This causes a problem at the end of a control write transaction.
    In USBCtrlEPServiceComplete(), during a control write (Host to Device),
    the EP0_OUT is setup to write any data to the CtrlTrfData buffer.
    If <SETUP[0]><IN[1]> is completed and USBCtrlTrfInHandler() is not
    called before the next <SETUP[0]> is received, then the latest Setup
    data will be written to the CtrlTrfData buffer instead of the SetupPkt
    buffer.

    If USBCtrlTrfInHandler() was called before the latest <SETUP[0]> is
    received, then there would be no problem,
    because USBPrepareForNextSetupTrf() would have been called and updated
    ep0Bo.ADR to point to the SetupPkt buffer.

    Work around:
    Check for the problem as described above and copy the Setup data from
    CtrlTrfData to SetupPkt.
    *******************************************************************/
```

```
    if((controlTransferState == CTRL_TRF_RX) &&
       (USBPacketDisable == 1) &&
       (pBDTEntryEP0OutCurrent->CNT == sizeof(CTRL_TRF_SETUP)) &&
       (pBDTEntryEP0OutCurrent->STAT.PID == SETUP_TOKEN) &&
       (pBDTEntryEP0OutNext->STAT.UOWN == 0))
    {
        unsigned char setup_cnt;

        pBDTEntryEP0OutNext->ADR = (BYTE*)ConvertToPhysicalAddress(&SetupPkt);

        // The Setup data was written to the CtrlTrfData buffer, must copy
        // it back to the SetupPkt buffer so that it can be processed correctly
        // by USBCtrlTrfSetupHandler().
        for(setup_cnt = 0; setup_cnt < sizeof(CTRL_TRF_SETUP); setup_cnt++)
        {
            *(((BYTE*)&SetupPkt)+setup_cnt) = *(((BYTE*)&CtrlTrfData)+setup_cnt);
        }//end for
    }
    /* End v3b fix */
    else
    {
        controlTransferState = WAIT_SETUP;
        pBDTEntryEP0OutNext->CNT = USB_EP0_BUFF_SIZE;      // Defined in
usb_config.h
        pBDTEntryEP0OutNext->ADR = (BYTE*)ConvertToPhysicalAddress(&SetupPkt);

        /*******************************************************************
        Bug Fix: Feb 26, 2007 v2.1 (#F1)
        *******************************************************************
        In the original firmware, if an OUT token is sent by the host
        before a SETUP token is sent, the firmware would respond with an ACK.
        This is not a correct response, the firmware should have sent a STALL.
        This is a minor non-compliance since a compliant host should not
        send an OUT before sending a SETUP token. The fix allows a SETUP
        transaction to be accepted while stalling OUT transactions.
        *******************************************************************/
        //ep0Bo.Stat.Val = _USIE|_DAT0|_DTSEN;        // Removed
        pBDTEntryEP0OutNext->STAT.Val = _USIE|_DAT0|_DTSEN|_BSTALL;  //Added #F1

        /*******************************************************************
        Bug Fix: Feb 26, 2007 v2.1 (#F3)
        *******************************************************************
        In the original firmware, if an IN token is sent by the host
        before a SETUP token is sent, the firmware would respond with an ACK.
        This is not a correct response, the firmware should have sent a STALL.
        This is a minor non-compliance since a compliant host should not
        send an IN before sending a SETUP token.

        Comment why this fix (#F3) is interfering with fix (#AF1).
        *******************************************************************/
        pBDTEntryIn[0]->STAT.Val = _UCPU;             // Should be removed

        {
            BDT_ENTRY* p;

            p = (BDT_ENTRY*)(((unsigned
int)pBDTEntryIn[0])^USB_NEXT_EP0_IN_PING_PONG);
            p->STAT.Val = _UCPU;
        }

        //ep0Bi.Stat.Val = _USIE|_BSTALL;   // Should be added #F3
    }

    //if someone is still expecting data from the control transfer
    //  then make sure to terminate that request and let them know that
    //  they are done
    if(outPipes[0].info.bits.busy == 1)
```

```
    {
        if(outPipes[0].pFunc != NULL)
        {
            outPipes[0].pFunc();
        }
        outPipes[0].info.bits.busy = 0;
    }

}//end USBPrepareForNextSetupTrf

/********************************************************************
 * Function:        void USBCheckStdRequest(void)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        This routine checks the setup data packet to see
 *                  if it knows how to handle it
 *
 * Note:            None
 *******************************************************************/
void USBCheckStdRequest(void)
{
    if(SetupPkt.RequestType != STANDARD) return;

    switch(SetupPkt.bRequest)
    {
        case SET_ADR:
            inPipes[0].info.bits.busy = 1;            // This will generate a
zero length packet
            USBDeviceState = ADR_PENDING_STATE;       // Update state only
            /* See USBCtrlTrfInHandler() for the next step */
            break;
        case GET_DSC:
            USBStdGetDscHandler();
            break;
        case SET_CFG:
            USBStdSetCfgHandler();
            break;
        case GET_CFG:
            inPipes[0].pSrc.bRam = (BYTE*)&USBActiveConfiguration;         // Set
Source
            inPipes[0].info.bits.ctrl_trf_mem = _RAM;               // Set memory
type
            inPipes[0].wCount.v[0] = 1;                          // Set data count
            inPipes[0].info.bits.busy = 1;
            break;
        case GET_STATUS:
            USBStdGetStatusHandler();
            break;
        case CLR_FEATURE:
        case SET_FEATURE:
            USBStdFeatureReqHandler();
            break;
        case GET_INTF:
            inPipes[0].pSrc.bRam =
(BYTE*)&USBAlternateInterface+SetupPkt.bIntfID;  // Set source
            inPipes[0].info.bits.ctrl_trf_mem = _RAM;               // Set memory
type
            inPipes[0].wCount.v[0] = 1;                          // Set data count
            inPipes[0].info.bits.busy = 1;
            break;
```

```
        case SET_INTF:
            inPipes[0].info.bits.busy = 1;
            USBAlternateInterface[SetupPkt.bIntfID] = SetupPkt.bAltID;
            break;
        case SET_DSC:
            USB_SET_DESCRIPTOR_HANDLER(EVENT_SET_DESCRIPTOR,0,0);
            break;
        case SYNCH_FRAME:
        default:
            break;
    }//end switch
}//end USBCheckStdRequest

/********************************************************************
 * Function:        void USBStdFeatureReqHandler(void)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        This routine handles the standard SET & CLEAR
 *                  FEATURES requests
 *
 * Note:            None
 *******************************************************************/
void USBStdFeatureReqHandler(void)
{
    BDT_ENTRY *p;
    #if defined(__C32__)
        DWORD* pUEP;
    #else
        unsigned int* pUEP;
    #endif
#ifdef USB_SUPPORT_OTG
    if ((SetupPkt.bFeature == OTG_FEATURE_B_HNP_ENABLE)&&
        (SetupPkt.Recipient == RCPT_DEV))
    {
        inPipes[0].info.bits.busy = 1;
        if(SetupPkt.bRequest == SET_FEATURE)
            USBOTGEnableHnp();
        else
            USBOTGDisableHnp();
    }

    if ((SetupPkt.bFeature == OTG_FEATURE_A_HNP_SUPPORT)&&
        (SetupPkt.Recipient == RCPT_DEV))
    {
        inPipes[0].info.bits.busy = 1;
        if(SetupPkt.bRequest == SET_FEATURE)
            USBOTGEnableSupportHnp();
        else
            USBOTGDisableSupportHnp();
    }


    if ((SetupPkt.bFeature == OTG_FEATURE_A_ALT_HNP_SUPPORT)&&
        (SetupPkt.Recipient == RCPT_DEV))
    {
        inPipes[0].info.bits.busy = 1;
        if(SetupPkt.bRequest == SET_FEATURE)
            USBOTGEnableAltHnp();
        else
            USBOTGDisableAltHnp();
```

```
    }
#endif
    if((SetupPkt.bFeature == DEVICE_REMOTE_WAKEUP)&&
       (SetupPkt.Recipient == RCPT_DEV))
    {
        inPipes[0].info.bits.busy = 1;
        if(SetupPkt.bRequest == SET_FEATURE)
            RemoteWakeup = TRUE;
        else
            RemoteWakeup = FALSE;
    }//end if

    if((SetupPkt.bFeature == ENDPOINT_HALT)&&
       (SetupPkt.Recipient == RCPT_EP)&&
       (SetupPkt.EPNum != 0))
    {
        inPipes[0].info.bits.busy = 1;
        /* Must do address calculation here */

        if(SetupPkt.EPDir == 0)
        {
            p = (BDT_ENTRY*)pBDTEntryOut[SetupPkt.EPNum];
        }
        else
        {
            p = (BDT_ENTRY*)pBDTEntryIn[SetupPkt.EPNum];
        }

             //if it was a SET_FEATURE request
        if(SetupPkt.bRequest == SET_FEATURE)
        {
                    //Then STALL the endpoint
            p->STAT.Val = _USIE|_BSTALL;
        }
        else
        {
                    //If it was not a SET_FEATURE
                    //point to the appropriate UEP register
            #if defined(__C32__)
                pUEP = (DWORD*)(&U1EP0);
                pUEP += (SetupPkt.EPNum*4);
            #else
                pUEP = (unsigned int*)(&U1EP0+SetupPkt.EPNum);
            #endif

                    //Clear the STALL bit in the UEP register
            *pUEP &= ~UEP_STALL;

            if(SetupPkt.EPDir == 1) // IN
            {
                        //If the endpoint is an IN endpoint then we
                        //  need to return it to the CPU and reset the
                        //  DTS bit so that the next transfer is correct
                #if (USB_PING_PONG_MODE == USB_PING_PONG__ALL_BUT_EP0) ||
(USB_PING_PONG_MODE == USB_PING_PONG__FULL_PING_PONG)
                    p->STAT.Val = _UCPU|_DAT0;
                    //toggle over the to the next buffer
                    ((BYTE_VAL*)&p)->Val ^= USB_NEXT_PING_PONG;
                    p->STAT.Val = _UCPU|_DAT1;
                #else
                    p->STAT.Val = _UCPU|_DAT1;
                #endif
            }
            else
            {
                        //If the endpoint was an OUT endpoint then we
                        //  need to give control of the endpoint back to
```

```
                        //  the SIE so that the function driver can
                        //  receive the data as they expected.  Also need
                        //  to set the DTS bit so the next packet will be
                        //  correct
                #if (USB_PING_PONG_MODE == USB_PING_PONG__ALL_BUT_EP0) ||
(USB_PING_PONG_MODE == USB_PING_PONG__FULL_PING_PONG)
                    p->STAT.Val = _USIE|_DAT0|_DTSEN;
                    //toggle over the to the next buffer
                    ((BYTE_VAL*)&p)->Val ^= USB_NEXT_PING_PONG;
                    p->STAT.Val = _USIE|_DAT1|_DTSEN;
                #else
                    p->STAT.Val = _USIE|_DAT1|_DTSEN;
                #endif

            }
        }//end if

    }//end if
}//end USBStdFeatureReqHandler

/*****************************************************************
 * Function:        void USBStdGetDscHandler(void)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        This routine handles the standard GET_DESCRIPTOR
 *                  request.
 *
 * Note:            None
 *****************************************************************/
void USBStdGetDscHandler(void)
{
    if(SetupPkt.bmRequestType == 0x80)
    {
        inPipes[0].info.Val = USB_INPIPES_ROM | USB_INPIPES_BUSY |
USB_INPIPES_INCLUDE_ZERO;

        switch(SetupPkt.bDescriptorType)
        {
            case USB_DESCRIPTOR_DEVICE:
                #if !defined(USB_USER_DEVICE_DESCRIPTOR)
                    inPipes[0].pSrc.bRom = (ROM BYTE*)&device_dsc;
                #else
                    inPipes[0].pSrc.bRom = (ROM BYTE*)USB_USER_DEVICE_DESCRIPTOR;
                #endif
                inPipes[0].wCount.Val = sizeof(device_dsc);
                break;
            case USB_DESCRIPTOR_CONFIGURATION:
                #if !defined(USB_USER_CONFIG_DESCRIPTOR)
                    inPipes[0].pSrc.bRom = *(USB_CD_Ptr+SetupPkt.bDscIndex);
                #else
                    inPipes[0].pSrc.bRom =
*(USB_USER_CONFIG_DESCRIPTOR+SetupPkt.bDscIndex);
                #endif
                inPipes[0].wCount.Val = *(inPipes[0].pSrc.wRom+1);
// Set data count
                break;
            case USB_DESCRIPTOR_STRING:
                //USB_NUM_STRING_DESCRIPTORS was introduced as optional in
release v2.3.  In v2.4 and
```

```
                //  later it is now manditory.  This should be defined in
usb_config.h and should
                //  indicate the number of string descriptors.
                if(SetupPkt.bDscIndex<USB_NUM_STRING_DESCRIPTORS)
                {
                    //Get a pointer to the String descriptor requested
                    inPipes[0].pSrc.bRom = *(USB_SD_Ptr+SetupPkt.bDscIndex);
                    // Set data count
                    inPipes[0].wCount.Val = *inPipes[0].pSrc.bRom;
                }
                else
                {
                    inPipes[0].info.Val = 0;
                }
                break;
            default:
                inPipes[0].info.Val = 0;
                break;
        }//end switch
    }//end if
}//end USBStdGetDscHandler

/*****************************************************************
 * Function:        void USBStdGetStatusHandler(void)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        This routine handles the standard GET_STATUS request
 *
 * Note:            None
 *****************************************************************/
void USBStdGetStatusHandler(void)
{
    CtrlTrfData[0] = 0;                // Initialize content
    CtrlTrfData[1] = 0;

    switch(SetupPkt.Recipient)
    {
        case RCPT_DEV:
            inPipes[0].info.bits.busy = 1;
            /*
             * [0]: bit0: Self-Powered Status [0] Bus-Powered [1] Self-Powered
             *      bit1: RemoteWakeup        [0] Disabled    [1] Enabled
             */
            if(self_power == 1) // self_power is defined in HardwareProfile.h
            {
                CtrlTrfData[0]|=0x01;
            }

            if(RemoteWakeup == TRUE)
            {
                CtrlTrfData[0]|=0x02;
            }
            break;
        case RCPT_INTF:
            inPipes[0].info.bits.busy = 1;     // No data to update
            break;
        case RCPT_EP:
            inPipes[0].info.bits.busy = 1;
            /*
             * [0]: bit0: Halt Status [0] Not Halted [1] Halted
```

```
             */
            {
                BDT_ENTRY *p;

                if(SetupPkt.EPDir == 0)
                {
                    p = (BDT_ENTRY*)pBDTEntryOut[SetupPkt.EPNum];
                }
                else
                {
                    p = (BDT_ENTRY*)pBDTEntryIn[SetupPkt.EPNum];
                }

                if(p->STAT.Val & _BSTALL)    // Use _BSTALL as a bit mask
                    CtrlTrfData[0]=0x01;    // Set bit0
                break;
            }
    }//end switch

    if(inPipes[0].info.bits.busy == 1)
    {
        inPipes[0].pSrc.bRam = (BYTE*)&CtrlTrfData;            // Set Source
        inPipes[0].info.bits.ctrl_trf_mem = _RAM;               // Set memory
type
        inPipes[0].wCount.v[0] = 2;                          // Set data count
    }//end if(...)
}//end USBStdGetStatusHandler

/********************************************************************************
 * Function:        void USBCtrlEPServiceComplete(void)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        This routine wrap up the ramaining tasks in servicing
 *                  a Setup Request. Its main task is to set the endpoint
 *                  controls appropriately for a given situation. See code
 *                  below.
 *                  There are three main scenarios:
 *                  a) There was no handler for the Request, in this case
 *                     a STALL should be sent out.
 *                  b) The host has requested a read control transfer,
 *                     endpoints are required to be setup in a specific way.
 *                  c) The host has requested a write control transfer, or
 *                     a control data stage is not required, endpoints are
 *                     required to be setup in a specific way.
 *
 *                  Packet processing is resumed by clearing PKTDIS bit.
 *
 * Note:            None
 *******************************************************************************/
void USBCtrlEPServiceComplete(void)
{
    /*
     * PKTDIS bit is set when a Setup Transaction is received.
     * Clear to resume packet processing.
     */
    USBPacketDisable = 0;

    if(inPipes[0].info.bits.busy == 0)
    {
        if(outPipes[0].info.bits.busy == 1)
```

```
        {
            controlTransferState = CTRL_TRF_RX;
            /*
             * Control Write:
             * <SETUP[0]><OUT[1]><OUT[0]>...<IN[1]> | <SETUP[0]>
             *
             * 1. Prepare IN EP to respond to early termination
             *
             *    This is the same as a Zero Length Packet Response
             *    for control transfer without a data stage
             */
            pBDTEntryIn[0]->CNT = 0;
            pBDTEntryIn[0]->STAT.Val = _USIE|_DAT1|_DTSEN;

            /*
             * 2. Prepare OUT EP to receive data.
             */
            pBDTEntryEP0OutNext->CNT = USB_EP0_BUFF_SIZE;
            pBDTEntryEP0OutNext->ADR =
(BYTE*)ConvertToPhysicalAddress(&CtrlTrfData);
            pBDTEntryEP0OutNext->STAT.Val = _USIE|_DAT1|_DTSEN;
        }
        else
        {
            /*
             * If no one knows how to service this request then stall.
             * Must also prepare EP0 to receive the next SETUP transaction.
             */
            pBDTEntryEP0OutNext->CNT = USB_EP0_BUFF_SIZE;
            pBDTEntryEP0OutNext->ADR =
(BYTE*)ConvertToPhysicalAddress(&SetupPkt);

            /* v2b fix */
            pBDTEntryEP0OutNext->STAT.Val = _USIE|_DAT0|_DTSEN|_BSTALL;
            pBDTEntryIn[0]->STAT.Val = _USIE|_BSTALL;
        }
    }
    else    // A module has claimed ownership of the control transfer session.
    {
        if(outPipes[0].info.bits.busy == 0)
        {
                    if(SetupPkt.DataDir == DEV_TO_HOST)
                    {
                          if(SetupPkt.wLength < inPipes[0].wCount.Val)
                          {
                                inPipes[0].wCount.Val = SetupPkt.wLength;
                          }
                          USBCtrlTrfTxService();
                          controlTransferState = CTRL_TRF_TX;
                          /*
                           * Control Read:
                           * <SETUP[0]><IN[1]><IN[0]>...<OUT[1]> | <SETUP[0]>
                           * 1. Prepare OUT EP to respond to early termination
                           *
                           * NOTE:
                           * If something went wrong during the control transfer,
                           * the last status stage may not be sent by the host.
                           * When this happens, two different things could happen
                           * depending on the host.
                           * a) The host could send out a RESET.
                           * b) The host could send out a new SETUP transaction
                           *    without sending a RESET first.
                           * To properly handle case (b), the OUT EP must be
setup
                           * to receive either a zero length OUT transaction, or
a
                           * new SETUP transaction.
```

```
                            *
                            * Furthermore, the Cnt byte should be set to prepare
for
                            * the SETUP data (8-byte or more), and the buffer
address
                            * should be pointed to SetupPkt.
                            */
                           pBDTEntryEP0OutNext->CNT = USB_EP0_BUFF_SIZE;
                           pBDTEntryEP0OutNext->ADR =
(BYTE*)ConvertToPhysicalAddress(&SetupPkt);
                           pBDTEntryEP0OutNext->STAT.Val = _USIE;           //
Note: DTSEN is 0!

                           pBDTEntryEP0OutCurrent->CNT = USB_EP0_BUFF_SIZE;
                           pBDTEntryEP0OutCurrent->ADR = (BYTE*)&SetupPkt;
                           pBDTEntryEP0OutCurrent->STAT.Val = _USIE;           //
Note: DTSEN is 0!

                           /*
                            * 2. Prepare IN EP to transfer data, Cnt should have
                            *    been initialized by responsible request owner.
                            */
                           pBDTEntryIn[0]->ADR =
(BYTE*)ConvertToPhysicalAddress(&CtrlTrfData);
                           pBDTEntryIn[0]->STAT.Val = _USIE|_DAT1|_DTSEN;
                    }
                    else   // (SetupPkt.DataDir == HOST_TO_DEVICE)
                    {
                           controlTransferState = CTRL_TRF_RX;
                           /*
                            * Control Write:
                            * <SETUP[0]><OUT[1]><OUT[0]>...<IN[1]> | <SETUP[0]>
                            *
                            * 1. Prepare IN EP to respond to early termination
                            *
                            *    This is the same as a Zero Length Packet Response
                            *    for control transfer without a data stage
                            */
                           pBDTEntryIn[0]->CNT = 0;
                           pBDTEntryIn[0]->STAT.Val = _USIE|_DAT1|_DTSEN;

                           /*
                            * 2. Prepare OUT EP to receive data.
                            */
                           pBDTEntryEP0OutNext->CNT = USB_EP0_BUFF_SIZE;
                           pBDTEntryEP0OutNext->ADR =
(BYTE*)ConvertToPhysicalAddress(&CtrlTrfData);
                           pBDTEntryEP0OutNext->STAT.Val = _USIE|_DAT1|_DTSEN;
                    }
        }
    }//end if(ctrl_trf_session_owner == MUID_NULL)

}//end USBCtrlEPServiceComplete


/*******************************************************************************
 * Function:        void USBCtrlTrfTxService(void)
 *
 * PreCondition:    pSrc, wCount, and usb_stat.ctrl_trf_mem are setup properly.
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        This routine should be called from only two places.
```

```
 *                  One from USBCtrlEPServiceComplete() and one from
 *                  USBCtrlTrfInHandler(). It takes care of managing a
 *                  transfer over multiple USB transactions.
 *
 * Note:            This routine works with isochronous endpoint larger than
 *                  256 bytes and is shown here as an example of how to deal
 *                  with BC9 and BC8. In reality, a control endpoint can never
 *                  be larger than 64 bytes.
 *****************************************************************************/
void USBCtrlTrfTxService(void)
{
    WORD_VAL byteToSend;

    /*
     * First, have to figure out how many byte of data to send.
     */
    if(inPipes[0].wCount.Val < USB_EP0_BUFF_SIZE)
    {
        byteToSend.Val = inPipes[0].wCount.Val;

        /* v2b fix */
        if(shortPacketStatus == SHORT_PKT_NOT_USED)
        {
            shortPacketStatus = SHORT_PKT_PENDING;
        }
        else if(shortPacketStatus == SHORT_PKT_PENDING)
        {
            shortPacketStatus = SHORT_PKT_SENT;
        }//end if
        /* end v2b fix for this section */
    }
    else
    {
        byteToSend.Val = USB_EP0_BUFF_SIZE;
    }

    /*
     * Next, load the number of bytes to send to BC9..0 in buffer descriptor
     */
    #if defined(__18CXX)
        pBDTEntryIn[0]->STAT.BC9 = 0;
        pBDTEntryIn[0]->STAT.BC8 = 0;
    #endif

    #if defined(__18CXX) || defined(__C30__)
        pBDTEntryIn[0]->STAT.Val |= byteToSend.byte.HB;
        pBDTEntryIn[0]->CNT = byteToSend.byte.LB;
    #elif defined(__C32__)
        pBDTEntryIn[0]->CNT = byteToSend.Val;
    #else
        #error "Not defined for this compiler"
    #endif

    /*
     * Subtract the number of bytes just about to be sent from the total.
     */
    inPipes[0].wCount.Val = inPipes[0].wCount.Val - byteToSend.Val;

    pDst = (USB_VOLATILE BYTE*)CtrlTrfData;        // Set destination pointer

    if(inPipes[0].info.bits.ctrl_trf_mem == USB_INPIPES_ROM)       // Determine
type of memory source
    {
        while(byteToSend.Val)
        {
            *pDst++ = *inPipes[0].pSrc.bRom++;
            byteToSend.Val--;
```

```
        }//end while(byte_to_send.Val)
    }
    else  // RAM
    {
        while(byteToSend.Val)
        {
            *pDst++ = *inPipes[0].pSrc.bRam++;
            byteToSend.Val--;
        }//end while(byte_to_send.Val)
    }//end if(usb_stat.ctrl_trf_mem == _ROM)

}//end USBCtrlTrfTxService

/*******************************************************************************
 * Function:        void USBCtrlTrfRxService(void)
 *
 * PreCondition:    pDst and wCount are setup properly.
 *                  pSrc is always &CtrlTrfData
 *                  usb_stat.ctrl_trf_mem is always _RAM.
 *                  wCount should be set to 0 at the start of each control
 *                  transfer.
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        *** This routine is only partially complete. Check for
 *                  new version of the firmware.
 *
 * Note:            None
 *******************************************************************************/
void USBCtrlTrfRxService(void)
{
    BYTE byteToRead;
    BYTE i;

    byteToRead = pBDTEntryEP0OutCurrent->CNT;

    /*
     * Accumulate total number of bytes read
     */
    if(byteToRead > outPipes[0].wCount.Val)
    {
        byteToRead = outPipes[0].wCount.Val;
    }
    else
    {
        outPipes[0].wCount.Val = outPipes[0].wCount.Val - byteToRead;
    }

    for(i=0;i<byteToRead;i++)
    {
        *outPipes[0].pDst.bRam++ = CtrlTrfData[i];
    }//end while(byteToRead.Val)

    //If there is more data to read
    if(outPipes[0].wCount.Val > 0)
    {
        /*
         * Don't have to worry about overwriting _KEEP bit
         * because if _KEEP was set, TRNIF would not have been
         * generated in the first place.
         */
        pBDTEntryEP0OutNext->CNT = USB_EP0_BUFF_SIZE;
        pBDTEntryEP0OutNext->ADR = (BYTE*)ConvertToPhysicalAddress(&CtrlTrfData);
```

```
        if(pBDTEntryEP0OutCurrent->STAT.DTS == 0)
        {
            pBDTEntryEP0OutNext->STAT.Val = _USIE|_DAT1|_DTSEN;
        }
        else
        {
            pBDTEntryEP0OutNext->STAT.Val = _USIE|_DAT0|_DTSEN;
        }
    }
    else
    {
        pBDTEntryEP0OutNext->ADR = (BYTE*)ConvertToPhysicalAddress(&SetupPkt);
        if(outPipes[0].pFunc != NULL)
        {
            outPipes[0].pFunc();
        }
        outPipes[0].info.bits.busy = 0;
    }


    // reset ep0Bo.Cnt to USB_EP0_BUFF_SIZE

}//end USBCtrlTrfRxService

/********************************************************************
 * Function:        void USBStdSetCfgHandler(void)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        This routine first disables all endpoints by
 *                  clearing UEP registers. It then configures
 *                  (initializes) endpoints by calling the callback
 *                  function USBCBInitEP().
 *
 * Note:            None
 *******************************************************************/
void USBStdSetCfgHandler(void)
{
    // This will generate a zero length packet
    inPipes[0].info.bits.busy = 1;

    //disable all endpoints except endpoint 0
    memset((void*)&U1EP1,0x00,(USB_MAX_EP_NUMBER-1));

    //clear the alternate interface settings
    memset((void*)&USBAlternateInterface,0x00,USB_MAX_NUM_INT);

    //set the current configuration
    USBActiveConfiguration = SetupPkt.bConfigurationValue;

    //if the configuration value == 0
    if(SetupPkt.bConfigurationValue == 0)
    {
        //Go back to the addressed state
        USBDeviceState = ADDRESS_STATE;
    }
    else
    {
        //Otherwise go to the configured state
        USBDeviceState = CONFIGURED_STATE;
        //initialize the required endpoints
```

```
        USBInitEP((BYTE ROM*)(USB_CD_Ptr[USBActiveConfiguration-1]));
        USB_INIT_EP_HANDLER(EVENT_CONFIGURED,0,0);

    }//end if(SetupPkt.bConfigurationValue == 0)
}//end USBStdSetCfgHandler

/*********************************************************************
 * Function:        void USBConfigureEndpoint(BYTE EPNum, BYTE direction)
 *
 * PreCondition:    None
 *
 * Input:           BYTE EPNum - the endpoint to be configured
 *                  BYTE direction - the direction to be configured
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        This function will configure the specified
 *                  endpoint
 *
 * Note:            None
 ********************************************************************/
void USBConfigureEndpoint(BYTE EPNum, BYTE direction)
{
    volatile BDT_ENTRY* handle;

    handle = (volatile BDT_ENTRY*)&BDT[EP0_OUT_EVEN];
    handle += BD(EPNum,direction,0)/sizeof(BDT_ENTRY);

    handle->STAT.UOWN = 0;

    if(direction == 0)
    {
        pBDTEntryOut[EPNum] = handle;
    }
    else
    {
        pBDTEntryIn[EPNum] = handle;
    }

    #if (USB_PING_PONG_MODE == USB_PING_PONG__FULL_PING_PONG)
        handle->STAT.DTS = 0;
        (handle+1)->STAT.DTS = 1;
    #elif (USB_PING_PONG_MODE == USB_PING_PONG__NO_PING_PONG)
        //Set DTS to one because the first thing we will do
        //when transmitting is toggle the bit
        handle->STAT.DTS = 1;
    #elif (USB_PING_PONG_MODE == USB_PING_PONG__EP0_OUT_ONLY)
        if(EPNum != 0)
        {
            handle->STAT.DTS = 1;
        }
    #elif (USB_PING_PONG_MODE == USB_PING_PONG__ALL_BUT_EP0)
        if(EPNum != 0)
        {
            handle->STAT.DTS = 0;
            (handle+1)->STAT.DTS = 1;
        }
    #endif
}

/*********************************************************************************
*******************************
  Function:
        void USBEnableEndpoint(BYTE ep, BYTE options)
```

```
  Summary:
    This function will enable the specified endpoint with the specified
    options
  Description:
    This function will enable the specified endpoint with the specified
    options.

    Typical Usage:
    <code>
    void USBCBInitEP(void)
    {

USBEnableEndpoint(MSD_DATA_IN_EP,USB_IN_ENABLED|USB_OUT_ENABLED|USB_HANDSHAKE_ENA
BLED|USB_DISALLOW_SETUP);
        USBMSDInit();
    }
    </code>

    In the above example endpoint number MSD_DATA_IN_EP is being configured
    for both IN and OUT traffic with handshaking enabled. Also since
    MSD_DATA_IN_EP is not endpoint 0 (MSD does not allow this), then we can
    explicitly disable SETUP packets on this endpoint.
  Conditions:
    None
  Input:
    BYTE ep -       the endpoint to be configured
    BYTE options -  optional settings for the endpoint. The options should
                    be ORed together to form a single options string. The
                    available optional settings for the endpoint. The
                    options should be ORed together to form a single options
                    string. The available options are the following\:
                    * USB_HANDSHAKE_ENABLED enables USB handshaking (ACK,
                      NAK)
                    * USB_HANDSHAKE_DISABLED disables USB handshaking (ACK,
                      NAK)
                    * USB_OUT_ENABLED enables the out direction
                    * USB_OUT_DISABLED disables the out direction
                    * USB_IN_ENABLED enables the in direction
                    * USB_IN_DISABLED disables the in direction
                    * USB_ALLOW_SETUP enables control transfers
                    * USB_DISALLOW_SETUP disables control transfers
                    * USB_STALL_ENDPOINT STALLs this endpoint
  Return:
    None
  Remarks:
    None

*******************************************************************************
*******************************/
void USBEnableEndpoint(BYTE ep, BYTE options)
{
    //Set the options to the appropriate endpoint control register
    //*((unsigned char*)(&U1EP0+ep)) = options;
    {
        unsigned char* p;

        #if defined(__C32__)
            p = (unsigned char*)(&U1EP0+(4*ep));
        #else
            p = (unsigned char*)(&U1EP0+ep);
        #endif
        *p = options;
    }

    if(options & USB_OUT_ENABLED)
    {
        USBConfigureEndpoint(ep,0);
```

```
    }
    if(options & USB_IN_ENABLED)
    {
        USBConfigureEndpoint(ep,1);
    }
}

/********************************************************************
 * Function:        void USBStallEndpoint(BYTE ep, BYTE dir)
 *
 * PreCondition:    None
 *
 * Input:
 *   BYTE ep - the endpoint the data will be transmitted on
 *   BYTE dir - the direction of the transfer
 *
 * Output:          None
 *
 * Side Effects:    Endpoint is STALLed
 *
 * Overview:        STALLs the specified endpoint
 *
 * Note:            None
 *******************************************************************/
void USBStallEndpoint(BYTE ep, BYTE dir)
{
    BDT_ENTRY *p;

    if(ep == 0)
    {
        /*
         * If no one knows how to service this request then stall.
         * Must also prepare EP0 to receive the next SETUP transaction.
         */
        pBDTEntryEP0OutNext->CNT = USB_EP0_BUFF_SIZE;
        pBDTEntryEP0OutNext->ADR = (BYTE*)ConvertToPhysicalAddress(&SetupPkt);

        /* v2b fix */
        pBDTEntryEP0OutNext->STAT.Val = _USIE|_DAT0|_DTSEN|_BSTALL;
        pBDTEntryIn[0]->STAT.Val = _USIE|_BSTALL;
    }
    else
    {
        p = (BDT_ENTRY*)(&BDT[EP(ep,dir,0)]);
        p->STAT.Val |= _BSTALL | _USIE;

        //If the device is in FULL or ALL_BUT_EP0 ping pong modes
        //then stall that entry as well
        #if (USB_PING_PONG_MODE == USB_PING_PONG__FULL_PING_PONG) || \
            (USB_PING_PONG_MODE == USB_PING_PONG__ALL_BUT_EP0)

        p = (BDT_ENTRY*)(&BDT[EP(ep,dir,1)]);
        p->STAT.Val |= _BSTALL | _USIE;
        #endif
    }
}

/********************************************************************
 * Function:        USB_HANDLE USBTransferOnePacket(
 *                      BYTE ep,
 *                      BYTE dir,
 *                      BYTE* data,
 *                      BYTE len)
 *
 * PreCondition:    None
 *
 * Input:
```

```
 *   BYTE ep - the endpoint the data will be transmitted on
 *   BYTE dir - the direction of the transfer
                This value is either OUT_FROM_HOST or IN_TO_HOST
 *   BYTE* data - pointer to the data to be sent
 *   BYTE len - length of the data needing to be sent
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        Transfers one packet over the USB
 *
 * Note:            None
 *******************************************************************/
USB_HANDLE USBTransferOnePacket(BYTE ep,BYTE dir,BYTE* data,BYTE len)
{
    USB_HANDLE handle;

    //If the direction is IN
    if(dir != 0)
    {
        //point to the IN BDT of the specified endpoint
        handle = pBDTEntryIn[ep];
    }
    else
    {
        //else point to the OUT BDT of the specified endpoint
        handle = pBDTEntryOut[ep];
    }

    //Toggle the DTS bit if required
    #if (USB_PING_PONG_MODE == USB_PING_PONG__NO_PING_PONG)
        handle->STAT.Val ^= _DTSMASK;
    #elif (USB_PING_PONG_MODE == USB_PING_PONG__EP0_OUT_ONLY)
        if(ep != 0)
        {
            handle->STAT.Val ^= _DTSMASK;
        }
    #endif

    //Set the data pointer, data length, and enable the endpoint
    handle->ADR = (BYTE*)ConvertToPhysicalAddress(data);
    handle->CNT = len;
    handle->STAT.Val &= _DTSMASK;
    handle->STAT.Val |= _USIE | _DTSEN;

    //Point to the next buffer for ping pong purposes.
    if(dir != 0)
    {
        //toggle over the to the next buffer for an IN endpoint
        ((BYTE_VAL*)&pBDTEntryIn[ep])->Val ^= USB_NEXT_PING_PONG;
    }
    else
    {
        //toggle over the to the next buffer for an OUT endpoint
        ((BYTE_VAL*)&pBDTEntryOut[ep])->Val ^= USB_NEXT_PING_PONG;
    }
    return handle;
}

/*******************************************************************
 * Function:        void USBClearInterruptFlag(BYTE* reg, BYTE flag)
 *
 * PreCondition:    None
 *
 * Input:
 *   BYTE* reg - the register address holding the interrupt flag
```

```
 *   BYTE flag - the bit number needing to be cleared
 *
 * Output:           None
 *
 * Side Effects:     None
 *
 * Overview:         clears the specified interrupt flag.
 *
 * Note:
 *******************************************************************/
void USBClearInterruptFlag(BYTE* reg, BYTE flag)
{
    #if defined(__18CXX)
        *reg &= ~(0x01<<flag);
    #elif defined(__C30__) || defined(__C32__)
        *reg = (0x01<<flag);
    #else
        #error "Function not defined for this compiler
    #endif
}

/*********************************************************************
    Function:
        void USBDeviceDetach(void)

    Description:

    Precondition:

    Parameters:
        None

    Return Values:
        None

    Remarks:
        None

  ********************************************************************/
#if defined(USB_INTERRUPT)
void USBDeviceDetach(void)
{
    //If the interrupt option is selected then the customer is required
    //  to notify the stack when the device is attached or removed from the
    //  bus by calling the USBDeviceAttach() and USBDeviceDetach() functions.
    if (USB_BUS_SENSE != 1)
    {
         // Disable module & detach from bus
         U1CON = 0;

         // Mask all USB interrupts
         U1IE = 0;

         //Move to the detached state
         USBDeviceState = DETACHED_STATE;

         #ifdef  USB_SUPPORT_OTG
             //Disable D+ Pullup
             U1OTGCONbits.DPPULUP = 0;

             //Disable HNP
             USBOTGDisableHnp();

             //Deactivate HNP
             USBOTGDeactivateHnp();

             //If ID Pin Changed State
```

```
            if (USBIDIF && USBIDIE)
            {
                //Re-detect & Initialize
                 USBOTGInitialize();

                 //Clear ID Interrupt Flag
                 USBClearInterruptFlag(USBIDIFReg,USBIDIFBitNum);
            }
        #endif

        #ifdef __C30__
            //USBClearInterruptFlag(U1OTGIR, 3);
        #endif
           //return so that we don't go through the rest of
           //the state machine
         return;
    }

#ifdef USB_SUPPORT_OTG
    //If Session Is Started Then
   else
   {
        //If SRP Is Ready
        if (USBOTGSRPIsReady())
        {
            //Clear SRPReady
            USBOTGClearSRPReady();

            //Clear SRP Timeout Flag
            USBOTGClearSRPTimeOutFlag();

            //Indicate Session Started
            UART2PrintString( "\r\n***** USB OTG B Event - Session Started
*****\r\n" );
        }
    }
#endif
}
/*********************************************************************
    Function:
        void USBDeviceAttach(void)

    Description:

    Precondition:

    Parameters:
        None

    Return Values:
        None

    Remarks:
        None

  ********************************************************************/
void USBDeviceAttach(void)
{
    //if we are in the detached state
    if(USBDeviceState == DETACHED_STATE)
    {
        U1CON = 0;                          // Disable module & detach from bus

        // Mask all USB interrupts
        U1IE = 0;

        // Enable module & attach to bus
```

```
        while(!U1CONbits.USBEN){U1CONbits.USBEN = 1;}

        //moved to the attached state
        USBDeviceState = ATTACHED_STATE;

        //Enable/set things like: pull ups, full/low-speed mode,
        //set the ping pong mode, and set internal transceiver
        SetConfigurationOptions();

        USBEnableInterrupts();

        #ifdef  USB_SUPPORT_OTG
            U1OTGCON = USB_OTG_DPLUS_ENABLE | USB_OTG_ENABLE;
        #endif
    }
}
#endif  //#if defined(USB_INTERRUPT)
/** EOF USBDevice.c ***************************************************/
```

6.1.19. clrc632.h

```
/*********************************************************************************
 FileName:      clrc632.h
 Dependencies:  See INCLUDES section
 Processor:     PIC18 USB Microcontroller
 Compiler:      Microchip C18 (for PIC18)
 Overview:      This file contains function prototypes and definitions to manage
                the behaviour of CLRC632 RFid Transceiver

*********************************************************************************
 File Description:

 Change History:
  Rev   Date         Author                    Description
  1.0   15/10/2009   Óscar Aragón Andreu       Initial release

*********************************************************************************/

/** INCLUDES *******************************************************************/

#include "GenericTypeDefs.h"

/** PROTOTYPES *****************************************************************/

void StartUp( void );
void WriteReg( const BYTE CLRC632_REG, BYTE data );
BYTE ReadReg( const BYTE CLRC632_REG );
void SetBit( const BYTE CLRC632_REG, BYTE bit_nr );
BYTE CheckBit( const BYTE CLRC632_REG, BYTE bit_nr );
void ClearInterruptRq( BYTE interrupt );
void ExecuteCommand( const BYTE CLRC632_COMMAND );
void StopCommand( void );
void WriteE2( BYTE lsb, BYTE msb, BYTE n_bytes );
void ReadE2( BYTE lsb, BYTE msb, BYTE n_bytes );
void WriteDefaultConfig( void );
void ReadDefaultConfig( void );
void StartTimer( void );
void ResetCLRC632( void );

/** DEFINITIONS ****************************************************************/

//CLRC632 REGISTER SET

// Page 0: Command and Status
#define PAGE0                   0x00   // Selects the page register
#define COMMAND                 0x01   // Stars and stops the command execution
#define FIFODATA                0x02   // IN and OUT of 64 byte FIFO buffer
#define PRIMARYSTATUS           0x03   // Primary Status flags
#define FIFOLENGTH              0x04   // Number of bytes buffered in the FIFO
#define SECONDARYSTATUS         0X05   // Diverse estatus flags
#define INTERRUPTEN             0x06   // Control to enable and disable
interrupts
#define INTERRUPTRQ             0x07   // Interrupt request flags

// Page 1: Control and Status
#define PAGE1                   0x08   // Selects the register page
#define CONTROL                 0x09   // Diverse control flags
#define ERRORFLAG               0x0A   // Error flags showing the error status
#define COLLPOS                 0x0B   // Bit position of the RF collision
#define TIMERVALUE              0x0C   // Actual value of the timer
#define CRCRESULTLSB            0x0D   // LSB of the CRC-Coprocessor register
#define CRCRESULTMSB            0x0E   // MSB of the CRC-Coprocessor register
#define BITFRAMING              0x0F   // Adjustments for bit oriented frames

// Page 2: Transmitter and Coder Control
#define PAGE2                   0x10   // Selects the register page
```

```
#define TXCONTROL               0x11   // Controls of the antenna driver
#define CWCONDUCTANCE           0x12   // Conductance of the antenna driver
#define MODCONDUCTANCE          0x13   // Defines the driver output conductance
#define CODERCONTROL            0x14   // Sets the clock rate and the coding mode
#define MODWIDTH                0x15   // Selects the width of the modulation
pulse
#define MODWIDTHSOF             0x16   // Width of the modulation pulse for SOF
#define TYPEBFRAMING            0x17   // Framing for ISO14443-B communication

// Page 3: Receiver and Decoder Control
#define PAGE3                   0x18   // Selects the register page
#define RXCONTROL1              0x19   // Controls receiver behaviour
#define DECODERCONTROL          0x1A   // Controls decoder behaviour
#define BITPHASE                0x1B   // Selects the bit-phase
#define RXTHRESHOLD             0x1C   // Selects thresholds for the bit decoder
#define BPSKDEMCONTROL          0x1D   // Control BPSK receiver behaviour
#define RXCONTROL2              0x1E   // Controls decoder behaviour
#define CLOCKQCONTROL           0x1F   // Controls clock generation

// Page 4: RF-Timing and Channel Redundancy
#define PAGE4                   0x20   // Selects the register page
#define RXWAIT                  0x21   // Time interval after transmission
#define CHANNELREDUNDANCY       0x22   // Selects the kind and mode of checking
#define CRCPRESETLSB            0x23   // LSB for the CRC register
#define CRCPRESETMSB            0x24   // MSB or the CRC register
#define TIMESLOTPERIOD          0x25   // Time between automatically mitted
Frames
#define MFOUTSELECT             0x26   // signal applied to pin MFOUT
#define PRESET27                0x27   // These values shall not be changed

// Page 5: FIFO, Timer and IRQ-Pin Configuration
#define PAGE5                   0x28   // Selects the register page
#define FIFOLEVEL               0X29   // FIFO over- and underflow warning
#define TIMERCLOCK              0x2A   // Selects the divider for the timer clock
#define TIMERCONTROL            0x2B   // Start and stop conditions for the timer
#define TIMERRELOAD             0x2C   // Defines the pre-set value for the timer
#define IRQPINCONFIG            0x2D   // Configures the output stage of pin IRQ
#define PRESET2E                0x2E   // These values shall not be changed
#define PRESET2F                0x2F   // These values shall not be changed

// Page 6: RFU
#define PAGE6                   0x30   // Selects the register page
#define RFU0                    0x31   // Reserved for future use
#define RFU1                    0x32   // Reserved for future use
#define RFU2                    0x33   // Reserved for future use
#define RFU3                    0x34   // Reserved for future use
#define RFU4                    0x35   // Reserved for future use
#define RFU5                    0x36   // Reserved for future use
#define RFU6                    0x37   // Reserved for future use

// Page 7: Test Control
#define PAGE7                   0x38   // Selects the register page
#define RFU7                    0x39   // Reserved for future use
#define TESTANASELECT           0x3A   // Selects analog test mode
#define RFU8                    0x3B   // Reserved for future use
#define RFU9                    0x3C   // Reserved for future use
#define TESTDIGISELECT          0x3D   // Selects digital test mode
#define RFU10                   0x3E   // Reserved for future use
#define RFU11                   0x3F   // Reserved for future use

//CLRC632 COMMAND SET

#define IDLE        0x00   // No action, cancels current command execution
#define TRANSMIT    0x1A   // Transmit data from the FIFO buffer to the card
#define RECEIVE     0x16   // Activates receiver circuitry
#define TRANSCEIVE  0x1E   // Transmit data and activates the receiver
#define WRITEE2     0x01   // Gets data from FIFO buffer and writes it to E2PROM
```

```
#define READE2      0x03   // Reads data from E2PROM and puts it in  FIFO buffer
#define LOADKEYE2   0x0B   // Copies a key from the E2PROM into the key buffer
#define LOADKEY     0x19   // Reads a key from FIFO and put it in the key buffer
#define AUTHENT1    0x0C   // Performs the first part of Crypto1 card auth.
#define AUTHENT2    0x14   // Performs the second part of the card auth.
#define LOADCONFIG  0x07   // Reads data from E2PROM
#define CALCCRC     0x12   // Activates the CEC-Coprocessor

//MASKS (for SPI read/write operations)

#define WRITE_MASK  0x00   // Mask to addapt address to write to CLRC632
#define READ_MASK   0x80   // Mask to addapt address to read from CLRC632

//INTERRUPT SOURCES

#define TIMER_IRQ   0x20   // Timer interrupt
#define TX_IRQ      0x10   // Transmiter interrupt
#define RX_IRQ      0x08   // Receiver interrupt
#define IDLE_IRQ    0x04   // IDLE interrupt
#define HIALERT_IRQ 0x02   // FIFO Buffer High alert interrupt
#define LOALERT_IRQ 0x01   // FIFO Buffer Low alert interrupt
```

6.1.20. clrc632.c

```
/*********************************************************************************

 FileName:      clrc632.c
 Dependencies:  See INCLUDES section
 Processor:     PIC18 USB Microcontroller
 Compiler:      Microchip C18 (for PIC18)
 Overview:      This file contains function prototypes to manage the behaviour
                of CLRC632 RFid Transceiver

*********************************************************************************
 File Description:

 Change History:
  Rev   Date         Author                   Description
  1.0   15/10/2009   Óscar Aragón Andreu      Initial release

*********************************************************************************/

/** INCLUDES *******************************************************************/

#include "GenericTypeDefs.h"
#include "Compiler.h"
#include "HardwareProfile.h"
#include "/timer/timers.h"
#include "/spi/spi.h"
#include "/clrc632/clrc632.h"

/** VARIABLES ******************************************************************/

#pragma udata

// Default CLRC632 E2PROM Configuration for RFidReader and ISO15693
BYTE e2prom_config[32] = { 0x00, 0x51, 0x3F, 0x05,
                           0x2F, 0x3F, 0x3F, 0x00,
                           0x00, 0x8B, 0x14, 0x54,
                           0x68, 0x00, 0x41, 0x07,
                           0x00, 0x08, 0x2C, 0xFF,
                           0xFF, 0x00, 0x02, 0x00,
                           0x00, 0x00, 0x15, 0x00,
                           0x08, 0x03, 0x00, 0x00 };

// Error Handling
extern BYTE error;        // Error Type
extern BYTE error_counter;// Error Counter

/** DECLARATIONS ***************************************************************/

#pragma code

/*********************************************************************************
*  Function Name:    StartUp                                                     *
*  Return Value:     None                                                        *
*  Arguments:        None                                                        *
*                                                                                *
*  Description:      Starts the Start Up Phase and wait 47,20 us regarding       *
*                    CLRC632 specifications. If start Up Phase finishes and      *
*                    CLRC632 is not ready an error is set and error counter is   *
*                    incremented.                                                *
*                                                                                *
*********************************************************************************/
extern void StartUp( void )
{
 // Run Start Up Phase
 CLRC632_RSTPD = 0;          // Set !RSTPD pin
 TMR0L = TIMER0L_VAL;        // Init timer value;
```

```
 while(!INTCONbits.TMR0IF); // wait for t = StartUp Phase (Specs. Chapter 11)
 INTCONbits.TMR0IF = 0;     // Clear Timer0 overflow flag
 CloseTimer0();             // Close Timer0

 // Check if Start Up Phase finished successfully
 if ( CheckBit( PAGE0, 7 )) // CL RC632 Ready
      return;               // Return without error
 else
 {
    error = 0x01;           // Error occurred: 0x01, Start Up Phase Failed
      ERROR_LED = 1;        // Set Error Led
      error_counter++;      // Count error (To improve in future version)
      return;               // If error occurred exit immediately
 }
}// end void StartUp( void )

/*****************************************************************************
*  Function Name:    WriteReg                                                 *
*  Return Value:     None                                                     *
*  Arguments:        CLRC632_REG: CLRC632 Register Name or Address            *
*                    data: Data to be writen (1 byte)                         *
*                                                                             *
*  Description:      Assures Write structure and writes data to   specified   *
*                    CLRC632 Register. If the address of the register is not  *
*                    valid an error is set and counted and data will not be   *
*                    sent.                                                    *
*                                                                             *
*****************************************************************************/
extern void WriteReg( const BYTE CLRC632_REG, BYTE data )
{
 BYTE regaddr;            // CLRC632 Register address
 BYTE tx_structure;       // Write strucure (Register address is included)

 // Check Error, if occured exit immediately
 if(error)
      return;

 // Check if Register/Address is valid
 regaddr = CLRC632_REG;
 if( regaddr > 0x3F )     // Address cannot be upper than 0x3F
      {
       error = 0x03;      // Error occurred: 0x03, Register Address unavaible
       ERROR_LED = 1;     // Set Error Led
       error_counter++;   // Count error (To improve in future version)
       return;            // If error occurred exit inmmediately
      }

 // Write through SPI
 else
      {
       tx_structure = (WRITE_MASK |(regaddr<<1));   // Write operation structure
       CLRC632_SPI_SS = 0;                          // Start Write operation
       WriteSPI( tx_structure );                    // Send Write structure
       WriteSPI( data );                            // Send data (1 byte)
       CLRC632_SPI_SS = 1;                          // Finish Write operation
      }
}// end void WriteReg( const BYTE CLRC632_REG, BYTE data )

/*****************************************************************************
*  Function Name:    ReadReg                                                  *
*  Return Value:     CLRC632 Register data    or invalid value                *
*  Arguments:        CLRC632_REG: CL RC632 Register Name or address           *
*                                                                             *
*  Description:      Assures Read structure and writes data to    specified   *
*                    CLRC632 Register. If the address of the register is not  *
*                    valid an error is set and counted and data sent will be  *
*                    invalid 0xFF.                                            *
```

```
*                                                                              *
*******************************************************************************/
extern BYTE ReadReg( const BYTE CLRC632_REG )
{
 BYTE regaddr;             // CLRC632 Register address
 BYTE rx_structure;        // Read strucure (Register address is included)
 BYTE data;                // Data from CLRC632 Register

 // Check Error, if occured exit immediately
 if(error)
       return(error);

 // Check if Register/Address is valid
 regaddr = CLRC632_REG;    // Get Register Address
 if( regaddr > 0x3F )      // Addres cannot be upper than 0x3F
       {
        error = 0x03;      // Error cocurred: 0x03, Register Address unavaible
        ERROR_LED = 1;     // Set Error Led
        error_counter++;   // Count error (To improve in future version)
        return(0xFF);      // If error occurred set invalid value and exit
       }

 // Read through SPI
 else
       {
       rx_structure = (READ_MASK |(regaddr<<1)); // Asure Read operation
structure
       CLRC632_SPI_SS = 0;        // Set CLRC632 Slave Select
       WriteSPI( rx_structure ); // Send register adapted address to be read
       data  = ReadSPI();         // Receive data from specified register
       WriteSPI( 0x00 );          // Finish Read cycle
       CLRC632_SPI_SS = 1;        // Clear CLRC632 Slave Select
       }

 // Return SPI Data
 return ( data );
}// end BYTE ReadReg( const BYTE CLRC632_REG )

/*******************************************************************************
*  Function Name:   SetBit                                                     *
*  Return Value:    None                                                       *
*  Arguments:       CLRC632_REG: CL RC632 Register Name or address             *
*                   bit_nr:      bit to set, from b7 to b0                     *
*                                                                              *
*  Description:     Set to '1' specified bit of specified Register.            *
*                                                                              *
*******************************************************************************/
extern void SetBit( const BYTE CLRC632_REG, BYTE bit_nr )
{
 BYTE last_register_value = 0;   // Previous register value

 BYTE new_register_value = 0;    // New Register value

 // Get Register value, change bit value and write new register value
 last_register_value = ReadReg( CLRC632_REG );
 new_register_value = last_register_value | ( 0x01<<bit_nr );
 WriteReg( CLRC632_REG, new_register_value );
}// end void SetBit( const BYTE CLRC632_REG, BYTE bit_nr )

/*******************************************************************************
*  Function Name:   CheckBit                                                   *
*  Return Value:    Bit value (bool: 0 or 1)                                   *
*  Arguments:       CLRC632_REG: CL RC632 Register Name                        *
*                   bit_nr: Bit number o be read (b7 to b0)                    *
*                                                                              *
*  Description:     Returns the value of the selected bit from specified       *
*                   CLRC632 Register.                                          *
```

```
*                                                                              *
*******************************************************************************/
extern BYTE CheckBit( const BYTE CLRC632_REG, BYTE bit_nr )
{
 BYTE bit_mask = 1;       // Bit Mask
 BYTE bit_value = 0;      // Init Bit value
 BYTE register_value = 0; // Init Register value

 // Check and Return Bit value
 register_value = ReadReg( CLRC632_REG ); // Read Register value
 bit_mask = bit_mask << bit_nr;           // Update mask according to Bit Nr.
 bit_value = register_value & bit_mask;   // Get Bit value
 if ( !bit_value )
       {
        return( 0 );// Bit value is 0
       }
 else
       {
        return( 1 );// Bit value is 1
       }
}// end BYTE CheckBit( const BYTE CLRC632_REG, BYTE bit_nr )

/*******************************************************************************
*  Function Name:   ClearInterruptRq                                           *
*  Return Value:    None                                                       *
*  Arguments:       interrupt: interrupt name                                  *
*                                                                              *
*  Description:     Clear specified interrupt flag.                            *
*                                                                              *
*******************************************************************************/
extern void ClearInterruptRq( BYTE interrupt )
{
 interrupt = 0x00 | interrupt;      // Enables clear opetation
 WriteReg( INTERRUPTRQ, interrupt );// Clear interrupt in Interrupt Rq Register
}// end extern void ClearInterruptRq( BYTE interrupt )

/*******************************************************************************
*  Function Name:    ExecuteCommand                                            *
*  Return Value:     None                                                      *
*  Arguments:        CLRC632_REG: CL RC632 Register Name or address            *
*                                                                              *
*  Description:      Execute sfecified CLRC632 Command and stop current Command *
*                    execution.                                                *
*                                                                              *
*******************************************************************************/
extern void ExecuteCommand( const BYTE CLRC632_COMMAND )
{
 WriteReg( COMMAND, CLRC632_COMMAND ); // Execute Command and stop ongoin command
}// end void ExecuteCommand( const BYTE CLRC632_COMMAND )

/*******************************************************************************
*  Function Name:    StopCommand                                               *
*  Return Value:     None                                                      *
*  Arguments:        None                                                      *
*                                                                              *
*  Description:      Stop current comand execution inmediately                 *
*                                                                              *
*******************************************************************************/
extern void StopCommand( void )
{
 WriteReg( COMMAND, IDLE );            // Stop Command inmediately
}// end void StopCommand( void )

/*******************************************************************************
*  Function Name:    WriteE2                                                   *
*  Return Value:     None                                                      *
*  Arguments:        lsb: E2PPOM LSB Address                                   *
```

```
*                   msb: E2PPOM MSB Address                                  *
*                   n_bytes: Nr. of bytes to write                           *
*                                                                            *
*  Description:      Write data into CLRC632 E2PROM. If E2PROM address is not  *
*                    valid or Write Timeout finishes an error is set and the  *
*                    operation is cancelled.                                  *
*                                                                            *
******************************************************************************/
extern void WriteE2( BYTE lsb, BYTE msb, BYTE n_bytes )
{
 BYTE i;

 // Check Address (Specs. Chapter 6)
 if ( lsb < 0x10 || msb > 0x01 || n_bytes > 62 ) // Available blocks: 1 to 31
       {
// Maximun of 62 bytes
        error = 0x04;      // Error cocurred: 0x04, E2PROM Access not alowed
        ERROR_LED = 1;     // Set Error Led
        error_counter++;   // Count error (To improve in future version)
        return;            // If error occurred exit immediately
       }

 // Write Data in E2PROM
 else
       {
        WriteReg( FIFODATA, lsb );      // Write Address (LSB)
        WriteReg( FIFODATA, msb );      // Write Address (MSB)
        for ( i = 0; i < n_bytes; i++)  // Write Data
              {
               WriteReg( FIFODATA, e2prom_config[i] );
              }

        // Check Error, if occured exit immediately
        if(error)
              return;

        // Execute WriteE2 Command
        ExecuteCommand( WRITEE2 );

        // Init Timeout
        StartTimer();
        while( !CheckBit( INTERRUPTRQ, 4 ))  // Wait until programming finishes
              {
               if( CheckBit( INTERRUPTRQ, 5 ))  // Timeout
                     {
                      error = 0x06;      // Error cocurred: 0x06, E2PROM timeout
                      ERROR_LED = 1;     // Set Error Led
                      error_counter++;   // Count error
                      break;
                     }
              }
       }

 // Finish operation
 TX_LED = 0;
 StopCommand();                  // Stop WRITEE2 command
 ClearInterruptRq( TX_IRQ );     // Clear E2PROM interrupt
}// end void WriteE2( BYTE lsb, BYTE msb, BYTE n_bytes )

/******************************************************************************
*  Function Name:   ReadE2                                                   *
*  Return Value:    None                                                     *
*  Arguments:       lsb: E2PPOM LSB Address                                  *
*                   msb: E2PPOM MSB Address                                  *
*                   n_bytes: Nr. of bytes to read                            *
*                                                                            *
*  Description:     Read data from CLRC632 E2PROM. If Read operation requests  *
```

```
*                   and invalid number of bytes an error is set and operation   *
*                   is cancelled.                                               *
*******************************************************************************/
extern void ReadE2( BYTE lsb, BYTE msb, BYTE n_bytes )
{
 BYTE i;

 // Check Nr. of Bytes
 if ( n_bytes > 128 )     // Only up to 128 bytes can be read, from block 1 to 7
       {
        error = 0x04;     // Error cocurred: 0x04, E2PROM Access not alowed
        ERROR_LED = 1;    // Set Error Led
        error_counter++;  // Count error (To improve in future version)
        return;           // If error occurred exit immediately
       }
 else
       {
        WriteReg( FIFODATA, lsb );      // Write Address (LSB)
        WriteReg( FIFODATA, msb );      // Write Address (MSB)
        WriteReg( FIFODATA, n_bytes );  // Write Nr. of Bytes to read

        // Check Error, if occured exit immediately
        if(error)
              return;

        // Execute ReadE2 Command
        ExecuteCommand( READE2 );
        for ( i = 0; i < n_bytes; i++) // Read Data
              {
               e2prom_config[i] = ReadReg( FIFODATA );
              }
       }
}// end void ReadE2( BYTE lsb, BYTE msb, BYTE n_bytes )

/*******************************************************************************
*  Function Name:    WriteDefaultConfig                                        *
*  Return Value:     None                                                      *
*  Arguments:        None                                                      *
*                                                                              *
*  Description:      Write Defaut Configuration in E2PROM Blocks 1 and 2, based *
*                    on RFidReader and ISO15693 Specifications. After that run *
*                    Reset Phase in order to load new configuration. If any    *
*                    error is set the operarion finishes.                      *
*                                                                              *
*******************************************************************************/
extern void WriteDefaultConfig( void )
{
 // Write Configuration to E2PROM (blocks 1 and 2)
 WriteE2( 0x10, 0x00, 32 );

 // Check Error, if occured exit immediately
 if(error)
       return;

 // Reset
 ResetCLRC632();    // Reset CLRC632
}// end void WriteDefaultConfig( void )

/*******************************************************************************
*  Function Name:   ReadDefaultConfig                                          *
*  Return Value:    None                                                       *
*  Arguments:       None                                                       *
*                                                                              *
*  Description:     Read Default E2PROM Configuration from Blocks 1 and 2      *
*                                                                              *
*******************************************************************************/
extern void ReadDefaultConfig( void )
```

```
{
 ReadE2( 0x10, 0x00, 32 );// Read Configuration from E2PROM (blocks 1 and 2)
}// end extern void ReadDefaultConfig( void )

/*******************************************************************************
*  Function Name:   StartTimer                                                 *
*  Return Value:    None                                                       *
*  Arguments:       None                                                       *
*                                                                              *
*  Description:     Start CLRC632 Timer count.                                 *
*                                                                              *
*******************************************************************************/

extern void StartTimer( void )
{
 SetBit( CONTROL, 1 );    // Start Timer
}// end void StartTimer( void )

/*******************************************************************************
*  Function Name:    ResetCLRC632                                              *
*  Return Value:     None                                                      *
*  Arguments:        None                                                      *
*                                                                              *
*  Description:      Reset CLRC632. Clear !RSTPD pin, init StartUp timer and   *
*                    run Start Up Phase.                                       *
*                                                                              *
*******************************************************************************/
extern void ResetCLRC632( void )
{
 CLRC632_RSTPD = 1;                    // Reset CLRC632
 OpenTimer0( TIMER_INT_OFF &           // StartUp Timer enabled
                    T0_8BIT &
                    T0_SOURCE_INT &
                    T0_PS_1_2 );
 StartUp();                            // Init Start Up phase
}// end void ResetCLRC632( void )
```

6.1.21. iso15693.h

```
/********************************************************************************
 FileName:      iso15693.h
 Dependencies:  See INCLUDES section
 Processor:     PIC18 USB Microcontroller
 Compiler:      Microchip C18 (for PIC18)
 Overview:      This file contains definitions to manage the behaviour of
                ISO15693-3

********************************************************************************
 File Description:

 Change History:
  Rev   Date         Author                  Description
  1.0   15/10/2009   Óscar Aragón Andreu      Initial release

********************************************************************************/

#ifndef ISO15693_H
#define ISO15693_H

/** DEFINITIONS ****************************************************************/

//ISO15693 COMMAND CODES

#define      ISO15693_INVENTORY                     0x01
#define      ISO15693_STAY_QUIET                    0x02
#define      ISO15693_READ_SINGLE_BLOCK             0x20
#define      ISO15693_WRITE_SINGLE_BLOCK            0x21
#define      ISO15693_LOCK_BLOCK                    0x22
#define      ISO15693_READ_MULTIPLE_BLOCKS          0x23
#define      ISO15693_WRITE_MULTIPLE_BLOCKS         0x24
#define      ISO15693_SELECT                        0x25
#define      ISO15693_RESET_TO_READY                0x26
#define      ISO15693_WRITE_AFI                     0x27
#define      ISO15693_LOCK_AFI                      0x28
#define      ISO15693_WRITE_DSFID                   0x29
#define      ISO15693_LOCK_DSFID                    0x2A
#define      ISO15693_GET_SYSTEM_INFO               0x2B
#define      ISO15693_GET_MULTIPLE_BLOCK_SECURITY   0x2C

#endif //ISO15693_H
```

6.1.22. iso15693.c

```
/*********************************************************************************
 FileName:      iso1569.c
 Dependencies:  See INCLUDES section
 Processor:     PIC18 USB Microcontroller
 Compiler:      Microchip C18 (for PIC18)
 Overview:      This file contains function prototypes to manage the behaviour
                of ISO15693 based on CLRC632 RFid Transceiver

*********************************************************************************
 File Description:

 Change History:
  Rev   Date         Author                   Description
  1.0   15/10/2009   Óscar Aragón Andreu      Initial release

*********************************************************************************/

/** INCLUDES *******************************************************************/

#include "GenericTypeDefs.h"
#include "Compiler.h"
#include "HardwareProfile.h"
#include "RFidReader.h"
#include "/clrc632/clrc632.h"
#include "/iso15693/iso15693.h"

/** VARIABLES ******************************************************************/

#pragma udata

// Data
extern DATA_PACKET INPacket;
extern DATA_PACKET OUTPacket;
extern unsigned char fifo_pointer;

// Error Handling
extern BYTE error;        // Error Type
extern BYTE error_counter;// Error Counter

// Tx & Rx Headers
extern BYTE TX_HEADER;
extern BYTE RX_HEADER;

// Simulated Tag
BYTE SimTag[32] = { 0 };  // Simulated Tag:RI-I03-114A-S1 ( Texas Instruments )
                          // 256-Bit User Memory in 8-Bit x 32-Bit Blocks
BYTE SimNrOfBlocks = 8;   // Blocks = 8
BYTE SimBlockLength = 4;  // Block Length = 4 bytes

/** PROTOTYPES *****************************************************************/

void ISOGeneralCommand( void );
void WriteSingleMultipleBlocks( void );
void ReadSingleMultipleBlocks( void );
void ClearTagData( void );

/** SIMULATION PROTOTYPES ******************************************************/

void SimWriteSingleMultipleBlocks( void );
void SimReadSingleMultipleBlocks( void );
void SimClearTagData( void );

/** DECLARATIONS ***************************************************************/

#pragma code
```

```
/*******************************************************************************
*  Function Name:   ISOGeneralCommand                                          *
*  Return Value:    None                                                       *
*  Parameters:      None                                                       *
*                                                                              *
*  Description:     Executes any ISO15693 Command, so a high knowledge of      *
*                   ISO15693 is needed. Gets Data from RFid Reader Software    *
*                   Application and writes it in CLR632 FIFO, then it starts   *
*                   Tx/Rx/Timeout sequences. If no Error occurred it gets Data *
*                   from CLRC632 FIFO and puts it into USB RAM Buffer back to  *
*                   Software Application.                                      *
*                                                                              *
*******************************************************************************/
void ISOGeneralCommand( void )
{
 int i;      // Pointer to Tx Data

 // Write Data in CLRC632 FIFO Buffer
 for ( i = TX_HEADER; i < OUTPacket.Length; i++)
 {
       WriteReg(FIFODATA, OUTPacket._byte[i]);
 }

 // Check Error, if occured exit immediately
 if(error)
       return;

 // Wait until Tx or Timeout end
 ClearInterruptRq( TIMER_IRQ );          // Clear Timeout interrupt
 ClearInterruptRq( TX_IRQ );             // Clear Tx interrupt
 ExecuteCommand( TRANSCEIVE );           // Start Transmit command
 StartTimer();                           // Init Timeout counter
 while ( !CheckBit( INTERRUPTRQ, 4 ))    // Wait until Tx or Timeout end
       {
        TX_LED = 1;                      // Tx Led ON
        if( CheckBit( INTERRUPTRQ, 5 ))  // If Timeout ends
              {
               error = 0x05;             // Error occurred: 0x05, Transmit Timeout
               ERROR_LED = 1;            // Set Error Led
               error_counter++;          // Count error
               break;
              }
       }
 ClearInterruptRq( TIMER_IRQ );          // Clear Timeout interrupt
 ClearInterruptRq( TX_IRQ );             // Clear Tx interrupt
 TX_LED = 0;                             // Tx Led OFF
 if(error)                               // If error ocurred
        return;                          // Exit immediately

 // Wait until Rx or Timeout end
 ClearInterruptRq( TIMER_IRQ );          // Clear Timeout interrupt
 ClearInterruptRq( RX_IRQ );             // Clear Rx interrupt
 StartTimer();                           // Start Timeout counter
 while ( !CheckBit( INTERRUPTRQ, 3 ))    // Wait until Rx or Timeout end
       {
        RX_LED = 1;                      // Rx Led ON
        if( CheckBit( INTERRUPTRQ, 5 ))  // If Timeout ends
              {
               error = 0x06;             // Error occurred: 0x06, Receive Timeout
               ERROR_LED = 1;            // Set Error Led
               error_counter++;          // Count error
               break;
              }
       }
 ClearInterruptRq( TIMER_IRQ );          // Clear Timeout interrupt
 ClearInterruptRq( RX_IRQ );             // Clear Rx interrupt
 RX_LED = 0;                             // Rx Led OFF
```

```
 if(error)                                  // If error ocurred
       return;                              // Exit immediately

 // Get Data from CLRC632 Buffer
 INPacket.Length = ReadReg( FIFOLENGTH ); // Packet length
 for ( i = RX_HEADER; i < INPacket.Length + RX_HEADER; i++)
 {
      INPacket._byte[i] = ReadReg( FIFODATA ); // Read Data from CLRC632 Buffer
 }
}// end void ISOGeneralCommand( void )

/*****************************************************************************
*  Function Name:   WriteSingleMultipleBlocks                                *
*  Return Value:    None                                                     *
*  Parameters:      None                                                     *
*                                                                            *
*  Description:     Runs Write Single or Multiple Blocks ISO Commands. To do it *
*                   gets Data from RFid Reader Software Application and writes  *
*                   it in CLR632 FIFO, then it starts Tx/Rx/Timeout sequences.  *
*                   If no Error occurred it gets Data from CLRC632 FIFO and     *
*                   puts it into USB RAM Buffer back to Software Application.   *
*                                                                            *
*****************************************************************************/
void WriteSingleMultipleBlocks( void )
{
 int i = 0;          // Data Offset
 //int j = 0;        // Tag Data Pointer

 BYTE FirstBlockNr = OUTPacket._byte[3];      // Get First Block Number
 BYTE NrOfBlocks = OUTPacket._byte[4];        // Get Nr. of Blocks

 // Write Data to CLRC632 FIFO Buffer, data offset is taken into account
 WriteReg( FIFODATA, OUTPacket.Flags);                      // Write Flags
 WriteReg( FIFODATA, ISO15693_WRITE_MULTIPLE_BLOCKS );      // Write Command
 WriteReg( FIFODATA, FirstBlockNr );                        // Write FirstBlockNr
 WriteReg( FIFODATA, NrOfBlocks );                          // Write NrOfBlocks
 for(i = TX_HEADER + 4; i < OUTPacket.Length; i++)
 {
      WriteReg( FIFODATA, OUTPacket._byte[i]);
 }

 // Check Error, if occured exit immediately
 if(error)
       return;

 // Wait until Tx or Timeout end
 ClearInterruptRq( TIMER_IRQ );          // Clear Timeout interrupt
 ClearInterruptRq( TX_IRQ );             // Clear Tx interrupt
 ExecuteCommand( TRANSCEIVE );           // Start Transmit command
 StartTimer();                           // Init Timeout counter
 while ( !CheckBit( INTERRUPTRQ, 4 ))    // Wait until Tx or Timeout end
       {
        TX_LED = 1;                      // Tx Led ON
        if( CheckBit( INTERRUPTRQ, 5 ))  // If Timeout ends
              {
               error = 0x05;             // Error occurred: 0x05, Transmit Timeout
               ERROR_LED = 1;            // Set Error Led
               error_counter++;          // Count error
               break;
              }
       }
 ClearInterruptRq( TIMER_IRQ );           // Clear Timeout interrupt
 ClearInterruptRq( TX_IRQ );              // Clear Tx interrupt
 TX_LED = 0;                              // Tx Led OFF
 if(error)                                // If error ocurred
       return;                            // Exit immediately
```

```
 // Wait until Rx or Timeout end
 ClearInterruptRq( TIMER_IRQ );          // Clear Timeout interrupt
 ClearInterruptRq( RX_IRQ );            // Clear Rx interrupt
 StartTimer();                          // Start Timeout counter
 while ( !CheckBit( INTERRUPTRQ, 3 ))   // Wait until Rx or Timeout end
       {
        RX_LED = 1;                     // Rx Led ON
        if( CheckBit( INTERRUPTRQ, 5 ))  // If Timeout ends
              {
               error = 0x06;             // Error occurred: 0x06, Receive Timeout
               ERROR_LED = 1;            // Set Error Led
               error_counter++;          // Count error
               break;
              }
       }
 ClearInterruptRq( TIMER_IRQ );          // Clear Timeout interrupt
 ClearInterruptRq( RX_IRQ );             // Clear Rx interrupt
 RX_LED = 0;                             // Rx Led OFF
 if(error)                               // If error ocurred
        return;                          // Exit immediately

 // Get Data from CLRC632 Buffer
 INPacket.Length = ReadReg( FIFOLENGTH );     // INPacket length
 for ( i = RX_HEADER; i < INPacket.Length + RX_HEADER; i++)
 {
       INPacket._byte[i] = ReadReg( FIFODATA ); // Read Data from CLRC632 Buffer
 }
}// end void WriteSingleMultipleBlocks( void )

/*****************************************************************************
*  Function Name:   ReadSingleMultipleBlocks                                 *
*  Return Value:    None                                                     *
*  Parameters:      None                                                     *
*                                                                            *
*  Description:     Runs Read Single or Multiple Blocks ISO Commands. To do it  *
*                   gets Data from RFid Reader Software Application and writes  *
*                   it in CLR632 FIFO, then it starts Tx/Rx/Timeout sequences.  *
*                   If no Error occurred it gets Data from CLRC632 FIFO and     *
*                   puts it into USB RAM Buffer back to Software Application.   *
*                                                                            *
*****************************************************************************/
void ReadSingleMultipleBlocks( void )
{
 int i = 0;         // Data Offset

 // Get Data from RFid Reader Software Application
 BYTE FirstBlockNr = OUTPacket._byte[3];      // Get First Block Number
 BYTE NrOfBlocks = OUTPacket._byte[4];        // Get Nr. of Blocks

 // Write Data to CLRC632 FIFO Buffer, data offset is taken into account
 WriteReg( FIFODATA, OUTPacket.Flags);              // Write Flags
 WriteReg( FIFODATA, ISO15693_READ_MULTIPLE_BLOCKS );// Write Command
 WriteReg( FIFODATA, FirstBlockNr );                // Write FirstBlockNr
 WriteReg( FIFODATA, NrOfBlocks );                  // Write NrOfBlocks

 // Check Error, if occured exit immediately
 if(error)
       return;

 // Wait until Tx or Timeout end
 ClearInterruptRq( TIMER_IRQ );          // Clear Timeout interrupt
 ClearInterruptRq( TX_IRQ );             // Clear Tx interrupt
 ExecuteCommand( TRANSCEIVE );           // Start Transmit command
 StartTimer();                           // Init Timeout counter
 while ( !CheckBit( INTERRUPTRQ, 4 ))    // Wait until Tx or Timeout end
       {
        TX_LED = 1;                             // Tx Led ON
```

```
        if( CheckBit( INTERRUPTRQ, 5 ))  // If Timeout ends
              {
               error = 0x05;             // Error occurred: 0x05, Transmit Timeout
               ERROR_LED = 1;           // Set Error Led
               error_counter++;         // Count error
               break;
              }
       }
 ClearInterruptRq( TIMER_IRQ );         // Clear Timeout interrupt
 ClearInterruptRq( TX_IRQ );            // Clear Tx interrupt
 TX_LED = 0;                            // Tx Led OFF
 if(error)                              // If error ocurred
        return;                         // Exit immediately

 // Wait until Rx or Timeout end
 ClearInterruptRq( TIMER_IRQ );         // Clear Timeout interrupt
 ClearInterruptRq( RX_IRQ );            // Clear Rx interrupt
 StartTimer();                          // Start Timeout counter
 while ( !CheckBit( INTERRUPTRQ, 3 ))  // Wait until Rx or Timeout end
       {
        RX_LED = 1;                     // Rx Led ON
        if( CheckBit( INTERRUPTRQ, 5 )) // If Timeout ends
              {
               error = 0x06;            // Error occurred: 0x06, Receive Timeout
               ERROR_LED = 1;           // Set Error Led
               error_counter++;         // Count error
               break;
              }
       }
 ClearInterruptRq( TIMER_IRQ );         // Clear Timeout interrupt
 ClearInterruptRq( RX_IRQ );            // Clear Rx interrupt
 RX_LED = 0;                            // Rx Led OFF
 if(error)                              // If error ocurred
        return;                         // Exit immediately

 // Get Data from CLRC632 Buffer
 INPacket.Length = ReadReg( FIFOLENGTH );     // INPacket length
 for ( i = RX_HEADER; i < INPacket.Length + RX_HEADER; i++)
 {
       INPacket._byte[i] = ReadReg( FIFODATA ); // Read Data from CLRC632 Buffer
 }
}// end void ReadSingleMultipleBlocks( void )

/*****************************************************************************
*  Function Name:   ClearTagData                                             *
*  Return Value:    None                                                     *
*  Parameters:      None                                                     *
*                                                                            *
*  Description:     Clears all Tag Data. All Blocks will be set to 0x00.     *
*                                                                            *
*****************************************************************************/
void ClearTagData( void )
{
 int i = 0;               // Data Offset
 int BytesToErase = 0;    // Bytes to Erase

 // Get Data from RFid Reader Software Application
 BYTE FirstBlockNr = OUTPacket._byte[3];      // Get First Block Number
 BYTE NrOfBlocks = OUTPacket._byte[4];        // Get Nr. of Blocks
 BYTE BlockLength = OUTPacket._byte[5];       // Get block Length in Bytes

 // Bytes to Erase
 BytesToErase = (NrOfBlocks + 1) * BlockLength;

 // Write Data to CLRC632 FIFO Buffer, data offset is taken into account
 WriteReg( FIFODATA, OUTPacket.Flags);                    // Write Flags
 WriteReg( FIFODATA, ISO15693_WRITE_MULTIPLE_BLOCKS );    // Write Command
```

```
 WriteReg( FIFODATA, FirstBlockNr );                    // Write FirstBlockNr
 WriteReg( FIFODATA, NrOfBlocks );                      // Write NrOfBlocks
 for(i = FirstBlockNr; i < BytesToErase; i++)
 {
      WriteReg( FIFODATA, 0x00 );      // Clear Data = 0x00
 }

 // Check Error, if occured exit immediately
 if(error)
      return;

 // Wait until Tx or Timeout end
 ClearInterruptRq( TIMER_IRQ );          // Clear Timeout interrupt
 ClearInterruptRq( TX_IRQ );             // Clear Tx interrupt
 ExecuteCommand( TRANSCEIVE );           // Start Transmit command
 StartTimer();                           // Init Timeout counter
 while ( !CheckBit( INTERRUPTRQ, 4 ))   // Wait until Tx or Timeout end
      {
       TX_LED = 1;                       // Tx Led ON
       if( CheckBit( INTERRUPTRQ, 5 ))  // If Timeout ends
            {
             error = 0x05;               // Error occurred: 0x05, Transmit Timeout
             ERROR_LED = 1;              // Set Error Led
             error_counter++;            // Count error
             break;
            }
      }
 ClearInterruptRq( TIMER_IRQ );           // Clear Timeout interrupt
 ClearInterruptRq( TX_IRQ );              // Clear Tx interrupt
 TX_LED = 0;                              // Tx Led OFF
 if(error)                                // If error ocurred
       return;                            // Exit immediately

 // Wait until Rx or Timeout end
 ClearInterruptRq( TIMER_IRQ );           // Clear Timeout interrupt
 ClearInterruptRq( RX_IRQ );              // Clear Rx interrupt
 StartTimer();                            // Start Timeout counter
 while ( !CheckBit( INTERRUPTRQ, 3 ))    // Wait until Rx or Timeout end
      {
       RX_LED = 1;                        // Rx Led ON
       if( CheckBit( INTERRUPTRQ, 5 ))   // If Timeout ends
            {
             error = 0x06;               // Error occurred: 0x06, Receive Timeout
             ERROR_LED = 1;              // Set Error Led
             error_counter++;            // Count error
             break;
            }
      }
 ClearInterruptRq( TIMER_IRQ );           // Clear Timeout interrupt
 ClearInterruptRq( RX_IRQ );              // Clear Rx interrupt
 RX_LED = 0;                              // Rx Led OFF
 if(error)                                // If error ocurred
       return;                            // Exit immediately

 // Get Data from CLRC632 Buffer
 INPacket.Length = ReadReg( FIFOLENGTH );// INPacket length
 for ( i = RX_HEADER; i < INPacket.Length + RX_HEADER; i++)
 {
      INPacket._byte[i] = ReadReg( FIFODATA ); // Read Data from CLRC632 Buffer
 }
}// end void ClearTagData( void )

/** SIMULATION PROTOTYPES *****************************************************/

/*******************************************************************************
*  Function Name:   SimWriteSingleMultipleBlocks                               *
*  Return Value:    None                                                       *
```

```
*  Parameters:      None                                                    *
*                                                                           *
*  Description:     Gets Data from RFid Reader Software Application and writes  *
*                   a Single or Multiples Blocks in Simulated Tag. Finally, it  *
*                   sends a simulated response to Software Application. This    *
*                   prototype can be used as demo to show how System will work. *
*                                                                           *
*****************************************************************************/
void SimWriteSingleMultipleBlocks( void )
{
 int i = 0;         // Data Offset
 int j = 0;         // Tag Data Pointer

 // Get Data from RFid Reader Software Application
 BYTE FirstBlockNr = OUTPacket._byte[3];      // Get First Block Number
 BYTE NrOfBlocks = OUTPacket._byte[4];        // Get Nr. of Blocks
 j = FirstBlockNr*SimBlockLength;             // Pointer to SimTag Block

 // Write Data to Simulated Tag, data offset is taken into account
 for(i = 0; i < OUTPacket.Length - 5; i++)
 {
       SimTag[j + i] = OUTPacket._byte[TX_HEADER + 3 + i];
 }

 // Send Simulated Response
 INPacket._byte[RX_HEADER] = 0x00;      // Flags
 INPacket._byte[RX_HEADER + 1] = 75;    // CRC16 LSB = K in ASCII
 INPacket._byte[RX_HEADER + 2] = 79;    // CRC16 LSB = O in ASCII
 INPacket.Length = 3;                                 // 3 Data Bytes
}// end void SimWriteSingleMultipleBlocks( void )

/*****************************************************************************
*  Function Name:   SimReadSingleMultipleBlocks                               *
*  Return Value:    None                                                      *
*  Parameters:      None                                                      *
*                                                                             *
*  Description:     Gets Data from RFid Reader Software Application and it     *
*                   requests from Simulated Tag its response. Finally, it sends *
*                   a simulated response to Software Application. Prototype   *
*                   can be used as demo to show how System will work.         *
*                                                                             *
*****************************************************************************/
void SimReadSingleMultipleBlocks( void )
{
 int i;      // Data Offset

 // Get Data from RFid Reader Software Application
 BYTE FirstBlockNr = OUTPacket._byte[3];      // Get First Block Number
 BYTE NrOfBlocks = OUTPacket._byte[4];        // Get Nr. of Blocks

 // Get Data from Simulated Tag and Send Simulated Response
 INPacket._byte[RX_HEADER] = 0x00;            // Flags
 for(i = (FirstBlockNr*SimBlockLength); i < ((NrOfBlocks+1)*(SimBlockLength));
i++)
 {
       INPacket._byte[RX_HEADER + 1 + i] = SimTag[i];      // Send Data from
SimTag
 }
 INPacket._byte[RX_HEADER + 1 + i] = 75;      // CRC16 LSB = K in ASCII
 INPacket._byte[RX_HEADER + 2 + i] = 79;      // CRC16 LSB = O in ASCII
 INPacket.Length = 3 + i;                     // 3 Data Bytes
}// end void SimReadSingleMultipleBlocks( void )
```

```
/*******************************************************************************
*  Function Name:   SimClearTagData                                            *
*  Return Value:    None                                                       *
*  Parameters:      None                                                       *
*                                                                              *
*  Description:     Clear all Data of Simulated Tag. All Blocks will be set    *
*                   to 0x00. Finally, it sends a simulated response to Software *
*                   Application. Prototype can be used as demo to show how     *
*                   System will work.                                          *
*                                                                              *
*******************************************************************************/
void SimClearTagData( void )
{
 BYTE i = 0;

 // Clear all Tag Data
 for( i = 0; i < SimNrOfBlocks*SimBlockLength; i++)
 {
       SimTag[i] =  0;                // Clear Tag Data
 }

 // Send Simulated Response
 INPacket._byte[RX_HEADER] = 0x00;      // Flags
 INPacket._byte[RX_HEADER + 1] = 75;    // CRC16 LSB = K in ASCII
 INPacket._byte[RX_HEADER + 2] = 79;    // CRC16 LSB = O in ASCII
 INPacket.Length = 3;                   // 3 Data Bytes
}// end void SimClearTagData( void )
```

6.1.23. HardwareProfile.h

```
/********************************************************************************
 FileName:      HardwareProfile.h
 Dependencies:  See INCLUDES section
 Processor:     PIC18 USB Microcontroller
 Compiler:      Microchip C18 (for PIC18)
 Overview:      This file contains definitions to manage the behaviour of
                RFid Reader Hardware Signals

********************************************************************************
 File Description:

 Change History:
  Rev   Date         Author                                  Description
  1.0   15/10/2009   Óscar Aragón Andreu      Initial release

********************************************************************************/

#ifndef HARDWARE_PROFILE_H
#define HARDWARE_PROFILE_H

/** INCLUDES *******************************************************************/

#include "Compiler.h"

/** DEFINITIONS ****************************************************************/

// Start Up Timer value (aprox. 62,7 us)
#define TIMER0L_VAL       68

// Stardard I/O
// PortA
#define mOpenPortA()      {ADCON1 |= 0x0F; LATA &= 0x00; TRISA &= 0x20;}
                                    // All pins to digital
                                    // Clear PORTA data latches
                                    // Configure PORTA as OUTPUTS

#define CLRC632_RSTPD     LATAbits.LATA0  // Output: CLR632 Reset pin
#define CLRC632_SPI_SS    LATAbits.LATA1  // Output: CLRC632 SPI Slave Select
#define ERROR_LED         LATAbits.LATA2  // Output: Error Led
#define TX_LED            LATAbits.LATA3  // Output: Transmit Led
#define RX_LED            LATAbits.LATA4  // Output: Error Led
#define CLR_ERROR_SW      PORTAbits.RA5   // Input: Clear Error Switch

// PortC
#define mOpenPortC()      {LATC &= 0x00; TRISC |= 0x01;}
                                    // Clear PORTC latches
                                    // RC0 as Input

#define SIM_MODE_SW       PORTCbits.RC0// Input: Sim Mode Switch

// USB Special I/O, required by MCHPFSUSB framework
#define self_power          1
#define USB_BUS_SENSE       1

#endif  //HARDWARE_PROFILE_H
```

6.1.24. RFidReader.h

```
/********************************************************************************
 FileName:      RfidReader.h
 Dependencies:  See INCLUDES section
 Processor:     PIC18 USB Microcontroller
 Compiler:      Microchip C18 (for PIC18)
 Overview:      This file contains definitions to manage the behaviour of
                RFid Reader as platform

********************************************************************************
 File Description:

 Change History:
  Rev   Date         Author                              Description
  1.0   15/10/2009   Óscar Aragón Andreu      Initial release

********************************************************************************/

#ifndef RFID_READER_H
#define RFID_READER_H

/** INCLUDES *******************************************************************/

#include "GenericTypeDefs.h"
#include "usb/usb_config.h"

/** DEFINITIONS ****************************************************************/

// RFid Reader Firmware Version
#define FIRMWARE_MINOR_VERSION   0x30; // ASCII = 0
#define FIRMWARE_MAJOR_VERSION   0x31; // ASCII = 1

// RFid Reader Commands
typedef enum
{
      READ_FIRMWARE_VERSION        = 0, // Command 0: Read Firmware Version
      WRITE_DEFAULT_CONFIG         = 1, // Command 1: Write Default Configuration
      READ_DEFAULT_CONFIG          = 2, // Command 2: Read Default Configuration
      ISO_GENERAL_COMMAND          = 3, // Command 3: ISO General Command
      WRITE_SINGLE_MULTIPLE_BLOCKS = 4,// Command 4: Write Single/Multiple Blocks
      READ_SINGLE_MULTIPLE_BLOCKS  = 5, // Command 5: Read Single/Multiple Blocks
      CLEAR_TAG_DAT                = 6, // Command 6: Clear Tag Data
      RESET_CLRC632                = 7, // Command 7: Reset CLRC632
      CLEAR_ERROR                  = 8, // Command 8: Clear Error
      FATAL_ERROR                  = 255 // Command 255: Fatal Error Start Up
Failed
}TYPE_CMD;

/** STRUCTURES *****************************************************************/

// RFid Reader Data Packet
typedef union DATA_PACKET
{
    BYTE _byte[USBGEN_EP_SIZE];  //For byte access

      // Transmit Header, TX_HEADER
      struct
      {
            BYTE Length; // Tx Length
            BYTE CMD;    // Tx Command
      };

      // Receive Header, RX_HEADER
      struct
      {
            BYTE Length; // Rx Length
```

```
            BYTE CMD;    // Rx Command
            BYTE Error;  // Rx Error
      };

      // ISO Transmit Header, TX_HEADER + Flags
      struct
      {
            BYTE Length; // Tx Length
            BYTE CMD;    // Tx Command
            BYTE Flags;  // Tx Flags
      };

      // ISO Receive Header, RX_HEADER + Flags
      struct
      {
            BYTE Length;   // Rx Length
            BYTE CMD;      // Rx Command
            BYTE Error;    // Rx Error
            BYTE Flags;    // Rx Flags
            BYTE ISOError; // Rx ISOError
   };
} DATA_PACKET;

#endif //RFID_READER_H
```

6.1.25. RFidReader.c

```
/********************************************************************************
 FileName:      RFidReader.c
 Dependencies:  See INCLUDES section
 Processor:     PIC18 USB Microcontroller
 Compiler:      Microchip C18 (for PIC18)
 Overview:      This file contains Main source code and function prototypes to
                manage the behaviour of RFidReader

********************************************************************************
 File Description:

 Change History:
  Rev   Date         Author                    Description
  1.0   15/10/2009   Óscar Aragón Andreu       Initial release

********************************************************************************/

/** INCLUDES *******************************************************************/

#include "GenericTypeDefs.h"
#include "Compiler.h"
#include "HardwareProfile.h"
#include "RFidReader.h"
#include "/timer/timers.h"
#include "/spi/spi.h"
#include "/usb/usb.h"
#include "/clrc632/clrc632.h"

/** PIC18F2550 CONFIGURATION ***************************************************/

#if defined(__18F2550)
    #pragma config PLLDIV   = 5         // 20 MHz crystal on System
    #pragma config CPUDIV   = OSC3_PLL4// [96MHzPLL Src: /4]
    #pragma config USBDIV   = 1         // Clock source from OSC1/OSC2
    #pragma config FOSC     = HSPLL_HS // HS oscillator,PLL enabled,HS used by
USB
    #pragma config FCMEN    = OFF       // Fail Safe Clock Monitor disabled
    #pragma config IESO     = OFF       // Internal/External Switch Over disabled
    #pragma config PWRT     = OFF       // Power Up Timer disabled
    #pragma config VREGEN   = ON        // USB Voltage Regulator enabled
    #pragma config WDT      = OFF       // Watchdog Timer HW Disabled - SW
Controlled
    #pragma config WDTPS    = 32768     // Watchdog Postscaler1:32768
    #pragma config MCLRE    = ON        // Master Clear Reset enabled
    #pragma config LPT1OSC  = OFF       // Low Power Timer1 Oscillator disabled
    #pragma config PBADEN   = OFF       // A/D disabled, PortB<4:0> are configured
as digital I/O on RESET
    #pragma config CCP2MX   = ON        // CCP2 input/output is multiplexed with
RC1
    #pragma config STVREN   = ON        // Stack Overflow Reset enabled
    #pragma config LVP      = OFF       // Low Voltage ICSP disabled
    #pragma config XINST    = OFF       // Extended Instruction Set disabled
    #pragma config DEBUG    = ON        // Background Debugger enabled
    #pragma config CP0      = OFF       // Code Protection Block 0 disabled
    #pragma config CP1      = OFF       // Code Protection Block 1 disabled
    #pragma config CP2      = OFF       // Code Protection Block 2 disabled
    #pragma config CP3      = OFF       // Code Protection Block 3 disabled
    #pragma config CPB      = OFF       // Boot Block Code Protection disabled
    #pragma config CPD      = OFF       // Data EEPROM Code Protection disabled
    #pragma config WRT0     = OFF       // Write Protection Block 0 disabled
    #pragma config WRT1     = OFF       // Write Protection Block 1 disabled
    #pragma config WRT2     = OFF       // Write Protection Block 2 disabled
    #pragma config WRT3     = OFF       // Write Protection Block 3 disabled
    #pragma config WRTB     = OFF       // Boot Block Write Protection disabled
```

```
    #pragma config WRTC     = OFF       // Configuration Register Write Protection
disabled
    #pragma config WRTD     = OFF       // Data EEPROM Write Protection disabled
    #pragma config EBTR0    = OFF       // Table Read Protection Block 0 disabled
    #pragma config EBTR1    = OFF       // Table Read Protection Block 1 disabled
    #pragma config EBTR2    = OFF       // Table Read Protection Block 2 disabled
    #pragma config EBTR3    = OFF       // Table Read Protection Block 3 disabled
    #pragma config EBTRB    = OFF       // Boot Block Table Read Protection
disabled

#else
    #error Microcontroller not defined, see "HardwareProfile.h" and __FILE__
#endif

/** VARIABLES ******************************************************************/

// USB variables in USB RAM
#pragma udata USB_VARIABLES = 0x500     // USB RAM Address
DATA_PACKET INPacket;                   // IN Data Packet
DATA_PACKET OUTPacket;                  // OUT Data Packet

// USB variables in Flash
#pragma udata
USB_HANDLE USBGenericInHandle = 0;      // USB IN Data Handler
USB_HANDLE USBGenericOutHandle = 0;     // USB OUT Data Handler

// Error Handling
BYTE error = 0;             // Error Code
BYTE error_counter = 0;     // Error Counter

// Headers Length
BYTE TX_HEADER = 2;         // Tx Header, 2 bytes length
BYTE RX_HEADER = 3;         // Rx Header, 3 bytes length

// CLRC632 Default Configuration
extern BYTE e2prom_config[32];

// Simulated Tag
extern BYTE SimTag[32];           // SimTag Available Memory = 32 bytes
extern BYTE SimNrOfBlocks;        // SimTag Blocks = 8
extern BYTE SimBlockLength;       // SimTag Block Length = 4 bytes

/** PROTOTYPES *****************************************************************/

void InitializeSystem(void);
void USBDeviceTasks(void);
void HighPriorityISRCode(void);
void LowPriorityISRCode(void);
void ProcessIO(void);
void ServiceRequests(void);
void ClearError(void);
extern void OpenTimer0 (PARAM_SCLASS unsigned char config);
extern void ISOGeneralCommand( void );
extern void WriteSingleMultipleBlocks(void);
extern void ReadSingleMultipleBlocks(void);
extern void ClearTagData( void );
extern void SimWriteSingleMultipleBlocks(void);
extern void SimReadSingleMultipleBlocks(void);
extern void SimClearTagData( void );

/** VECTOR MAPING **************************************************************/

//On PIC18 devices, addresses 0x00, 0x08, and 0x18 are used for
//the reset, high priority interrupt, and low priority interrupt
//vectors.

#define MAPPED_RESET_VECTOR_ADDRESS           0x00
```

```
#define MAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS  0x08
#define MAPPED_LOW_INTERRUPT_VECTOR_ADDRESS   0x18

#pragma code REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS =
MAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS
void Remapped_High_ISR (void)
{
       _asm goto HighPriorityISRCode _endasm
}
#pragma code REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS =
MAPPED_LOW_INTERRUPT_VECTOR_ADDRESS
void Remapped_Low_ISR (void)
{
       _asm goto LowPriorityISRCode _endasm
}

/** INTERRUPTS ****************************************************************/

#pragma code

// High Interrupt Handling Routine
#pragma interrupt HighPriorityISRCode
void HighPriorityISRCode()
{
      #if defined(USB_INTERRUPT)
            USBDeviceTasks();
      #endif
}

// Low Interrupt Handling Routine
#pragma interruptlow LowPriorityISRCode
void LowPriorityISRCode()
{
      //CLRC632 Interrupts:
      //Check which interrupt flag caused the interrupt.
      //Service the interrupt
      //Clear the interrupt flag
}

/** DECLARATIONS **************************************************************/

#pragma code

/********************************************************************************
*  Function Name:   main                                                        *
*  Return Value:    None                                                        *
*  Parameters:      None                                                        *
*                                                                               *
*  Description:     Initializes System and wait for any request from RFid       *
*                   Reader Software Application or Hardware Signal.             *
*                                                                               *
********************************************************************************/
void main(void)
{
 InitializeSystem();      // Initilize System

 #if defined(USB_INTERRUPT)
    USBDeviceAttach();    // Attach to USB
 #endif

 while(1)
 {
    #if defined(USB_POLLING)
    // Check bus status and service USB interrupts.
    USBDeviceTasks(); // Interrupt or polling method.  If using polling, must
                      // call this function periodically.  This function will
                      // take care of processing and responding to SETUP
```

```
                    // transactions (such as during the enumeration process
                    // when you first plug in).  USB hosts require that USB
                    // devices should accept and process SETUP packets in a
                    // timely fashion.  Therefore,when using polling, this
                    // function should be called
                    // frequently (such as once about every 100 microseconds)
                    // at any time that a SETUP packet might reasonably be
                    // expected to be sent by the host to your device.
                    // In most cases, the USBDeviceTasks() function does not
                    // take very long to execute (~50 instruction cycles)
                    // before it returns.
    #endif

    // Application-specific tasks.
    // Application related code may be added here, or in the ProcessIO()
    // function.
    ProcessIO();
 }//end while
}//end void main(void)

/********************************************************************************
*  Function Name:   InitializeSystem                                            *
*  Return Value:    None                                                        *
*  Parameters:      None                                                        *
*                                                                               *
*  Description:     Initializes all Nodes: Ports, StartUp Timer, SPI, CLRC632   *
*                   and USB.                                                    *
*                                                                               *
********************************************************************************/
void InitializeSystem(void)
{
 // Init PORTS
 mOpenPortA();      // Configure PORTA RA1:RA4 as OUTPUTS and RA5 as INPUT

 mOpenPortC();      // Configure PORTC RC0 as INPUT

 // Init Signals
 CLRC632_RSTPD = 1;       // CLRC632 not selected
 CLRC632_SPI_SS = 1;      // Set CLRC632_SPI_SS (Slave Select)
 ERROR_LED = 0;           // Clear Error Led


 // Init Start Up Timer
 OpenTimer0( TIMER_INT_OFF &           // Timer Interrupt Disabled
                    T0_8BIT &          // Timer as 8 bit counter
                    T0_SOURCE_INT &    // Timer Souce
                    T0_PS_1_2 );       // Prescaler 1:2

 // Init SPI
 OpenSPI( SPI_FOSC_4,     // SPI Source Fosc/4
          MODE_11,        // SPI Mode 11
          SMPMID );       // SPI Input Sample

 // Init USB
 USBDeviceInit();

 // Init CLRC632
 StartUp();

}//end static void InitializeSystem(void)

/********************************************************************************
*  Function Name:   ProcessIO                                                   *
*  Return Value:    None                                                        *
*  Parameters:      None                                                        *
*                                                                               *
*  Description:     Services Requests from RFid Reader Software Application,    *
```

```
*                   but only if Device is attached to USB. Also check if any    *
*                   Hardware signal change its state.                           *
*                                                                               *
********************************************************************************/
void ProcessIO(void)
{
 // IO purposes
 if( CLR_ERROR_SW == 0 )  // Clear Error (Hardware Signal)
       ClearError();

 // RFid Reader USB tasks
 if((USBDeviceState < CONFIGURED_STATE)||(USBSuspendControl==1))
        return;
 else
       ServiceRequests();

}//end ProcessIO

/********************************************************************************
*  Function Name:   ServiceRequests                                             *
*  Return Value:    None                                                        *
*  Parameters:      None                                                        *
*                                                                               *
*  Description:     This function takes in the commands from the RFid Reader    *
*                   Software Application and executes the comands requested.    *
*                                                                               *
********************************************************************************/
void ServiceRequests(void)
{

BYTE i;       // Data pointer

//if the handle is no longer busy then the last transmission is complete
if(!USBHandleBusy(USBGenericOutHandle))
{
       INPacket.Length = 0;       // Clear data length

       // If CLRC632 Start Up Phase Failed (Fatal Error)
       if( error == 1 )
       {
              // Only Reset CLRC632 Command can run, Inhibit any other Command
              if(OUTPacket.CMD != RESET_CLRC632)
                     OUTPacket.CMD = FATAL_ERROR;
       }

       // Process Command
    switch(OUTPacket.CMD)
    {
              // Command 0: Read Firmware Version
              case READ_FIRMWARE_VERSION:
                     ClearError();                    // Clear Previous Errors
                     INPacket.CMD = READ_FIRMWARE_VERSION;   // Send Command
                     INPacket.Error = error;                 // Send Error
                     INPacket._byte[3] = FIRMWARE_MINOR_VERSION;// Send FW Minor
V.
                     INPacket._byte[4] = FIRMWARE_MAJOR_VERSION;   // Send FW
Major V.
                     INPacket.Length = RX_HEADER + 2;        // Send Rx Message
             break;

              // Command 1: Write Defaut configuration
              case WRITE_DEFAULT_CONFIG:
                     ClearError();                    // Clear Previous Errors
                     WriteDefaultConfig();            // Execute Command
                     INPacket.CMD = WRITE_DEFAULT_CONFIG;    // Send Command
                     INPacket.Error = error;                 // Send Error
                     INPacket.Length = RX_HEADER;            // Send RX_HEADER
```

```
                  break;

            // Command 2: Read Default Configuration
            case READ_DEFAULT_CONFIG:
                  ClearError();                    // Clear Previous Errors
                  ReadDefaultConfig();             // Execute Command
                  INPacket.CMD = READ_DEFAULT_CONFIG; // Send Command
                  INPacket.Error = error;          // Send Error
                  if(!error)
                  {
                        for ( i = 0; i < 32; i++)              // Send Data
                        {
                              INPacket._byte[RX_HEADER + i] =
e2prom_config[i];
                        }
                        INPacket.Length = RX_HEADER + i; // Send Rx Message
                  }
                  else
                  {
                        INPacket.Length = RX_HEADER; // Send RX_HEADER
                  }
                  break;

            // Command 3: ISOGeneralCommand
            case ISO_GENERAL_COMMAND:
                  ClearError();                    // Clear Previous Errors
                  ISOGeneralCommand();             // Run Command
                  INPacket.CMD = ISO_GENERAL_COMMAND; // Send Command
                  INPacket.Error = error;          // Send Error
                  if(!error)
                  {

                        INPacket.Length += RX_HEADER; // Send Rx Message
                  }
                  else
                  {
                        INPacket.Length = RX_HEADER; // Send RX_HEADER
                  }
                  break;

            // Command 4: WriteSingleMultipleBlocks
            case WRITE_SINGLE_MULTIPLE_BLOCKS:
                  ClearError();                    // Clear Previous Errors
                  if( SIM_MODE_SW == 1 )           // If SimMode Selected
                        SimWriteSingleMultipleBlocks();  // Run SimCommand
                  else                                   // Else
                        WriteSingleMultipleBlocks();     // Run Command
                  INPacket.CMD = WRITE_SINGLE_MULTIPLE_BLOCKS; // Send Command
                  INPacket.Error = error;                // Send Error
                  INPacket.Length += RX_HEADER;          // Send Rx Message
                  break;

            // Command 5: ReadSingleMultipleBlocks
            case READ_SINGLE_MULTIPLE_BLOCKS:
                  ClearError();                    // Clear Previous Errors
                  if( SIM_MODE_SW == 1)            // If SimMode Selected
                        SimReadSingleMultipleBlocks();   // Run SimCommand
                  else                                   // Else
                        ReadSingleMultipleBlocks();      // Run Command
                  INPacket.CMD = READ_SINGLE_MULTIPLE_BLOCKS;   // Send Command
                  INPacket.Error = error;                // Send Error
                  INPacket.Length += RX_HEADER;          // Send Rx Message
                  break;

            // Command 6: Clear Tag Data
            case CLEAR_TAG_DATA:
                  ClearError();              // Clear Previous Errors
```

```
                    if( SIM_MODE_SW == 1)      // If SimMode Selected
                          SimClearTagData();   // Run SimCommand
                    else                       // Else
                          ClearTagData();      // Run Command
                    INPacket.CMD = CLEAR_TAG_DATA;   // Send Command
                    INPacket.Error = error;          // Send Error
                    INPacket.Length += RX_HEADER;    // Send Rx Message
                    break;

              // Command 7: Reset CLRC362
              case RESET_CLRC632:
                    ClearError();              // Clear Previous Errors
                    ResetCLRC632();            // Run Command
                    INPacket.CMD = RESET_CLRC632; // Send Command
                    INPacket.Error = error;       // Send Error
                    INPacket.Length = RX_HEADER;  // Send RX_HEADER
                    break;

              // Command 8: Clear Error
              case CLEAR_ERROR:
                    ClearError();                    // Run Command
                    INPacket.CMD = CLEAR_ERROR;      // Send Command
                    INPacket.Error = error;          // Send Error
                    INPacket.Length = RX_HEADER;     // Send RX_HEADER
                    break;

              // Command 255: Fatal Error
              case FATAL_ERROR:
                    INPacket.CMD = FATAL_ERROR;      // Send Command
                    INPacket.Error = error;          // Send Error
                    INPacket.Length = RX_HEADER;     // Send RX_HEADER
                    break;

          default:
              Nop();
              break;
      }//end switch()

        // Send Data to RFid Reader Software Application
      if(INPacket.Length != 0)
      {
          if(!USBHandleBusy(USBGenericInHandle))
          {
              USBGenericInHandle =
USBGenWrite(USBGEN_EP_NUM,(BYTE*)&INPacket,INPacket.Length);
          }
      }//end if

      //Re-arm the OUT endpoint for the next packet
      USBGenericOutHandle =
USBGenRead(USBGEN_EP_NUM,(BYTE*)&OUTPacket,USBGEN_EP_SIZE);
   }//end if
}//end ServiceRequests



/********************************************************************************
*  Function Name:   ClearError                                                  *
*  Return Value:    None                                                        *
*  Parameters:      None                                                        *
*                                                                               *
*  Description:     Clear Error and switch off Error Led, but do not modify     *
*                   Error Counter in order to save Error statistics             *
*                                                                               *
********************************************************************************/
void ClearError(void)
{
```

```
 error = 0;         // Clear Error
 ERROR_LED = 0;     // Switch OFF Error Led
}// end void ClearError(void)

//******************************************************************************
//************************ USB Callback Functions ******************************
//******************************************************************************
// The USB firmware stack will call the callback functions USBCBxxx() in response
// to certain USB related events.  For example, if the host PC is powering down,
// it will stop sending out Start of Frame (SOF)packets to your device. In
// response to this, all USB devices are supposed to decrease their power
// consumption from the USB Vbus to <2.5mA each.  The USB module detects this
// condition (which according to the USB specifications is 3+ms of no bus
// activity/SOF packets) and then calls the USBCBSuspend() function.  You should
// modify these callback functions to take appropriate actions for each of these
// conditions.  For example, in the USBCBSuspend(), you may wish to add code that
// will decrease power consumption from Vbus to <2.5mA (such as by clock
// switching, turning off LEDs, putting the microcontroller to sleep, etc.).
// Then, in the USBCBWakeFromSuspend() function, you may then wish to add code
// that undoes the power saving things done in the USBCBSuspend() function.

// The USBCBSendResume() function is special, in that the USB stack will not
// automatically call this function.  This function is meant to be called from
// the application firmware instead.  See the additional comments near the
// function.

/******************************************************************************
 * Function:        void USBCBSuspend(void)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        Call back that is invoked when a USB suspend is detected
 *
 * Note:            None
 *****************************************************************************/
void USBCBSuspend(void)
{
      //Example power saving code.  Insert appropriate code here for the desired
      //application behavior.  If the microcontroller will be put to sleep, a
      //process similar to that shown below may be used:

      //ConfigureIOPinsForLowPower();
      //SaveStateOfAllInterruptEnableBits();
      //DisableAllInterruptEnableBits();
      //EnableOnlyTheInterruptsWhichWillBeUsedToWakeTheMicro();
      //Sleep();
      //RestoreStateOfAllPreviouslySavedInterruptEnableBits();
      //RestoreIOPinsToNormal();

      //IMPORTANT NOTE: Do not clear the USBActivityIF (ACTVIF) bit here. This
      // bit is cleared inside the usb_device.c file.  Clearing USBActivityIF
      // here will cause
      //things to not work as intended.
}


/******************************************************************************
 * Function:        void _USB1Interrupt(void)
 *
 * PreCondition:    None
 *
```

```
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        This function is called when the USB interrupt bit is set
 *                  In this example the interrupt is only used when the device
 *                  goes to sleep when it receives a USB suspend command
 *
 * Note:            None
 *******************************************************************************/
#if 0
void __attribute__ ((interrupt)) _USB1Interrupt(void)
{
    #if !defined(self_powered)
        if(U1OTGIRbits.ACTVIF)
        {
            IEC5bits.USB1IE = 0;
            U1OTGIEbits.ACTVIE = 0;
            IFS5bits.USB1IF = 0;

            //USBClearInterruptFlag(USBActivityIFReg,USBActivityIFBitNum);
            USBClearInterruptFlag(USBIdleIFReg,USBIdleIFBitNum);
            //USBSuspendControl = 0;
        }
    #endif
}
#endif

/*******************************************************************************
 * Function:        void USBCBWakeFromSuspend(void)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        The host may put USB peripheral devices in low power
 *                  suspend mode (by "sending" 3+ms of idle).  Once in suspend
 *                  mode, the host may wake the device back up by sending non-
 *                  idle state signalling.
 *
 *                  This call back is invoked when a wakeup from USB suspend
 *                  is detected.
 *
 * Note:            None
 *******************************************************************************/
void USBCBWakeFromSuspend(void)
{
      // If clock switching or other power savings measures were taken when
      // executing the USBCBSuspend() function, now would be a good time to
      // switch back to normal full power run mode conditions.  The host allows
      // a few milliseconds of wakeup time, after which the device must be
      // fully back to normal, and capable of receiving and processing USB
      // packets.  In order to do this, the USB module must receive proper
      // clocking (IE: 48MHz clock must be available to SIE for full speed USB
      // operation).
}

/********************************************************************
 * Function:        void USBCB_SOF_Handler(void)
 *
 * PreCondition:    None
```

```
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        The USB host sends out a SOF packet to full-speed
 *                  devices every 1 ms. This interrupt may be useful
 *                  for isochronous pipes. End designers should
 *                  implement callback routine as necessary.
 *
 * Note:            None
 *******************************************************************/
void USBCB_SOF_Handler(void)
{
    // No need to clear UIRbits.SOFIF to 0 here.
    // Callback caller is already doing that.
}

/*******************************************************************
 * Function:        void USBCBErrorHandler(void)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        The purpose of this callback is mainly for
 *                  debugging during development. Check UEIR to see
 *                  which error causes the interrupt.
 *
 * Note:            None
 *******************************************************************/
void USBCBErrorHandler(void)
{
    // No need to clear UEIR to 0 here.
    // Callback caller is already doing that.

      // Typically, user firmware does not need to do anything special
      // if a USB error occurs.  For example, if the host sends an OUT
      // packet to your device, but the packet gets corrupted (ex:
      // because of a bad connection, or the user unplugs the
      // USB cable during the transmission) this will typically set
      // one or more USB error interrupt flags.  Nothing specific
      // needs to be done however, since the SIE will automatically
      // send a "NAK" packet to the host.  In response to this, the
      // host will normally retry to send the packet again, and no
      // data loss occurs.  The system will typically recover
      // automatically, without the need for application firmware
      // intervention.

      // Nevertheless, this callback function is provided, such as
      // for debugging purposes.
}


/*******************************************************************
 * Function:        void USBCBCheckOtherReq(void)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
```

```
 * Output:           None
 *
 * Side Effects:    None
 *
 * Overview:        When SETUP packets arrive from the host, some
 *                              firmware must process the request and respond
 *                              appropriately to fulfill the request.  Some of
 *                              the SETUP packets will be for standard
 *                              USB "chapter 9" (as in, fulfilling chapter 9 of
 *                              the official USB specifications) requests, while
 *                              others may be specific to the USB device class
 *                              that is being implemented.  For example, a HID
 *                              class device needs to be able to respond to
 *                              "GET REPORT" type of requests.  This
 *                              is not a standard USB chapter 9 request, and
 *                              therefore not handled by usb_device.c.  Instead
 *                              this request should be handled by class specific
 *                              firmware, such as that contained in
usb_function_hid.c.
 *
 * Note:             None
 *******************************************************************************/
void USBCBCheckOtherReq(void)
{
}//end


/********************************************************************
 * Function:        void USBCBStdSetDscHandler(void)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        The USBCBStdSetDscHandler() callback function is
 *                              called when a SETUP, bRequest: SET_DESCRIPTOR
request
 *                              arrives.  Typically SET_DESCRIPTOR requests are
 *                              not used in most applications, and it is
 *                              optional to support this type of request.
 *
 * Note:             None
 *******************************************************************************/
void USBCBStdSetDscHandler(void)
{
    // Must claim session ownership if supporting this request
}//end


/******************************************************************************
 * Function:        void USBCBInitEP(void)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        This function is called when the device becomes
 *                  initialized, which occurs after the host sends a
```

```
 *                               SET_CONFIGURATION (wValue not = 0) request.
This
 *                               callback function should initialize the
endpoints
 *                               for the device's usage according to the current
 *                               configuration.
 *
 * Note:             None
 *******************************************************************************/
void USBCBInitEP(void)
{

USBEnableEndpoint(USBGEN_EP_NUM,USB_OUT_ENABLED|USB_IN_ENABLED|USB_HANDSHAKE_ENAB
LED|USB_DISALLOW_SETUP);
    USBGenericOutHandle =
USBGenRead(USBGEN_EP_NUM,(BYTE*)&OUTPacket,USBGEN_EP_SIZE);
}

/******************************************************************
 * Function:        void USBCBSendResume(void)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        The USB specifications allow some types of USB
 *                  peripheral devices to wake up a host PC (such
 *                  as if it is in a low power suspend to RAM state).
 *                  This can be a very useful feature in some
 *                  USB applications, such as an Infrared remote
 *                  control      receiver.  If a user presses the "power"
 *                  button on a remote control, it is nice that the
 *                  IR receiver can detect this signalling, and then
 *                  send a USB "command" to the PC to wake up.
 *
 *                  The USBCBSendResume() "callback" function is used
 *                  to send this special USB signalling which wakes
 *                  up the PC.  This function may be called by
 *                  application firmware to wake up the PC.  This
 *                  function should only be called when:
 *
 *                  1.  The USB driver used on the host PC supports
 *                  the remote wakeup capability.
 *                  2.  The USB configuration descriptor indicates
 *                  the device is remote wakeup capable in the
 *                  bmAttributes field.
 *                  3.  The USB host PC is currently sleeping,
 *                  and has previously sent your device a SET
 *                  FEATURE setup packet which "armed" the
 *                  remote wakeup capability.
 *
 *                  This callback should send a RESUME signal that
 *                  has the period of 1-15ms.
 *
 * Note:            Interrupt vs. Polling
 *                  -Primary clock
 *                  -Secondary clock ***** MAKE NOTES ABOUT THIS *******
 *                   > Can switch to primary first by calling
USBCBWakeFromSuspend()

 *                  The modifiable section in this routine should be changed
 *                  to meet the application needs. Current implementation
 *                  temporary blocks other functions from executing for a
```

```
 *                  period of 1-13 ms depending on the core frequency.
 *
 *                  According to USB 2.0 specification section 7.1.7.7,
 *                  "The remote wakeup device must hold the resume signaling
 *                  for at lest 1 ms but for no more than 15 ms."
 *                  The idea here is to use a delay counter loop, using a
 *                  common value that would work over a wide range of core
 *                  frequencies.
 *                  That value selected is 1800. See table below:
 *                  ==========================================================
 *                  Core Freq(MHz)      MIP         RESUME Signal Period (ms)
 *                  ==========================================================
 *                      48              12          1.05
 *                       4              1           12.6
 *                  ==========================================================
 *                  * These timing could be incorrect when using code
 *                    optimization or extended instruction mode,
 *                    or when having other interrupts enabled.
 *                    Make sure to verify using the MPLAB SIM's Stopwatch
 *                    and verify the actual signal on an oscilloscope.
 *******************************************************************/
void USBCBSendResume(void)
{
    static WORD delay_count;

    USBResumeControl = 1;                // Start RESUME signaling

    delay_count = 1800U;                // Set RESUME line for 1-13 ms
    do
    {
        delay_count--;
    }while(delay_count);
    USBResumeControl = 0;
}


/********************************************************************
 * Function:        BOOL USER_USB_CALLBACK_EVENT_HANDLER(
 *                        USB_EVENT event, void *pdata, WORD size)
 *
 * PreCondition:    None
 *
 * Input:           USB_EVENT event - the type of event
 *                  void *pdata - pointer to the event data
 *                  WORD size - size of the event data
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        This function is called from the USB stack to
 *                  notify a user application that a USB event
 *                  occured.  This callback is in interrupt context
 *                  when the USB_INTERRUPT option is selected.
 *
 * Note:            None
 *******************************************************************/
BOOL USER_USB_CALLBACK_EVENT_HANDLER(USB_EVENT event, void *pdata, WORD size)
{
    switch(event)
    {
        case EVENT_CONFIGURED:
            USBCBInitEP();
            break;
        case EVENT_SET_DESCRIPTOR:
            USBCBStdSetDscHandler();
            break;
```

```
        case EVENT_EP0_REQUEST:
            USBCBCheckOtherReq();
            break;
        case EVENT_SOF:
            USBCB_SOF_Handler();
            break;
        case EVENT_SUSPEND:
            USBCBSuspend();
            break;
        case EVENT_RESUME:
            USBCBWakeFromSuspend();
            break;
        case EVENT_BUS_ERROR:
            USBCBErrorHandler();
            break;
        case EVENT_TRANSFER:
            Nop();
            break;
        default:
            break;
    }
    return TRUE;
}
```

6.2. Código Fuente del Software

En este apartado no se tendrá el código reutilizado, únicamente el que se ha debido de generar para que la aplicación funcione como RFidReader v1.0.

6.2.1. RFidReader.h

```
/*****************************************************************
 FileName:   RFidReader.h
 Desciption: Definitions for RFidReader SW Application

 *****************************************************************
 File Description:

 Change History:
  Rev   Date         Description
  1.0  15/10/2009    Initial Release
*****************************************************************/

#defineREAD_FIRMWARE_VERSION                0
#define WRITE_DEFAULT_CONFIG                1
#define READ_DEFAULT_CONFIG                 2
#define ISO_GENERAL_COMMAND                 3
#define WRITE_SINGLE_MULTIPLE_BLOCKS        4
#define READ_SINGLE_MULTIPLE_BLOCKS         5
#define CLEAR_TAG_DATA                      6
#define RESET_CLRC632                       7
#define CLEAR_ERROR                         8
```

6.2.2. Form1.h

```
#pragma once
//-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
---
//-----------------------------------------------------BEGIN CUT AND PASTE
BLOCK-----------------------------------------------------------------------------
-------
/*****************************************************************
 FileName:          Form1.h
 Dependencies:      Windows Server 2003 R2 Platform SDK during development.
                        When compiled, needs .NET framework 2.0 redistributable
                        to run (under Microsoft Windows XP, Vista comes with
                        .NET 2.0 redistributable package already installed)
                        May also need Microsoft Visual C++ redistributable to
run.
 Hardware:          Need a free USB port to connect USB peripheral device
                        programmed with appropriate MCHPUSB demo firmware.  VID
and
                        PID in firmware must match the VID and PID in this
                        program.
 Compiler:   Microsoft Visual C++ 2005 Express Edition (or better)
                        (Microsoft Visual C++ 2008 Express Edition is believed
                        to work, but is currently untested)
 Company:           Microchip Technology, Inc.

 Software License Agreement:

 The software supplied herewith by Microchip Technology Incorporated
 (the “Company”) for its PIC® Microcontroller is intended and
 supplied to you, the Company’s customer, for use solely and
 exclusively with Microchip PIC Microcontroller products. The
 software is owned by the Company and/or its supplier, and is
 protected under applicable copyright laws. All rights are reserved.
 Any use in violation of the foregoing restrictions may subject the
 user to criminal sanctions under applicable laws, as well as to
 civil liability for the breach of the terms and conditions of this
 license.

 THIS SOFTWARE IS PROVIDED IN AN “AS IS” CONDITION. NO WARRANTIES,
 WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
 TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
 IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
 CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

*****************************************************************
 File Description:

 Change History:
  Rev   Date         Description
  1.0   06/15/2008   Initial release
  2.0  15/10/2009    Release adapted to work as a RFid Reader based
                                 on CL RC632 and ISO/IEC 15693-3
*****************************************************************
NOTE:  All user made code contained in this project is in the Form1.h file.
             All other code and files were generated automatically by either the
             new project wizard, or by the development environment (ex: code is
             automatically generated if you create a new button on the form, and
             then double click on it, which creates a click event handler
             function).  All user made code is contained in clearly marked cut
and
             paste blocks.

NOTE2:  Need Windows Server 2003 R2 Platform SDK.  Untested with Windows SDK.
```

```
            If getting build errors when trying to compile this project, make
sure the
            platform SDK is correctly installed, and that it has been integrated
into the VC++ development
            environment.  In other words, make sure the include paths have been
setup correctly in this IDE.
            Microsoft created some small tutorial videos showing how to do this.
            These may be located at:
            http://msdn.microsoft.com/en-us/visualc/aa336415.aspx
            The link is functional as of 21 May 2008, but could move.  If the
above link does not work,
            try searching for "Video1_PSDK.wmv", which may be associated with:
            "Using Visual C++ Express Edition and the Platform SDK"
*****************************************************************/


//Includes
#include <windows.h>      //Definitions for various common and not so common
types like DWORD, PCHAR, HANDLE, etc.
#include <Dbt.h>          //Need this for definitions of WM_DEVICECHANGE messages
#include "mpusbapi.h"     //Make sure this header file is located in your project
directory.
#include "RFidReader.h"   //Command Codes for Rfid Reader Services

//When modifying the firmware and changing the Vendor and Device ID's, make
//sure to update the PC application as well.
//Use the formatting: "Vid_xxxx&Pid_xxxx" where xxxx is a 16-bit hexadecimal
number.
//This is the USB device that this program will look for:
#define DeviceVID_PID "vid_04d8&pid_000c"
//-------------------------------------------------------END CUT AND PASTE BLOCK-
-----------------------------------------------------------------------------
---
//-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
---

namespace RFidReader {

      using namespace System;
      using namespace System::ComponentModel;
      using namespace System::Collections;
      using namespace System::Windows::Forms;
      using namespace System::Data;
      using namespace System::Drawing;
//-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
---
      using namespace System::IO;
      using namespace System::Globalization;
//-------------------------------------------------------BEGIN CUT AND PASTE
BLOCK-----------------------------------------------------------------------
-------
      using namespace System::Threading;
      using namespace System::Runtime::InteropServices;  //Need this to support
"unmanaged" code.

      /*
      In order to use these unmanaged functions from within the managed .NET
environment, we need
      to explicitly import the functions which we will be using from other .DLL
file(s).  Simply
      including the appropriate header files is not enough.

      Note: In order to avoid potential name conflicts in the header files
(which we still use),
```

```
       I have renamed the functions by adding "UM" (unmanaged) onto the end of
them.  To find
       documentation for the functions in MSDN, search for the function name
without the extra
       "UM" attached.
       Note2: In the header files, normally the function names are
       remapped, depending upon if UNICODE is defined or not.  For example, two
versions of the
       function SetupDiGetDeviceInterfaceDetail() exist.  One for UNICODE, and
one for ANSI.
       If the wrong version of the function is called, things won't work
correctly.  Therefore,
       in order to make sure the correct one gets called (based on your compiler
settings, which
       may or may not define "UNICODE"), it is useful to explicity specify the
CharSet when doing
       the DLL import.
       */

       #ifdef UNICODE
       #define      Seeifdef     Unicode
       #else
       #define Seeifdef    Ansi
       #endif

       //See the mpusbapi.dll source code (_mpusbapi.cpp) for API related
documentation for these functions.
       //The source code is in the MCHPFSUSB vX.X distributions.
       [DllImport("MPUSBAPI.dll" , EntryPoint="_MPUSBGetDLLVersion")]
       extern "C" DWORD MPUSBGetDLLVersion(void);
       [DllImport("MPUSBAPI.dll" , EntryPoint="_MPUSBGetDeviceCount")]
       extern "C" DWORD MPUSBGetDeviceCount(PCHAR pVID_PID);
       [DllImport("MPUSBAPI.dll" , EntryPoint="_MPUSBOpen")]
       extern "C" HANDLE MPUSBOpen(DWORD instance,   //  Input
                                                                  PCHAR pVID_PID,
       // Input
                                                                  PCHAR pEP,
       // Input
                                                                  DWORD dwDir, //
Input
                                                                  DWORD
dwReserved);// Input

       [DllImport("MPUSBAPI.dll" , EntryPoint="_MPUSBClose")]
       extern "C" BOOL MPUSBClose(HANDLE handle);    //Input
       [DllImport("MPUSBAPI.dll" , EntryPoint="_MPUSBRead")]
       extern "C" DWORD MPUSBRead(HANDLE handle,     // Input
                                                                  PVOID pData, //
Output
                                                                  DWORD dwLen, //
Input
                                                                  PDWORD pLength,
       // Output
                                                                  DWORD
dwMilliseconds);// Input

       [DllImport("MPUSBAPI.dll" , EntryPoint="_MPUSBWrite")]
       extern "C" DWORD MPUSBWrite(HANDLE handle,    // Input
                                                                  PVOID pData, //
Output
                                                                  DWORD dwLen, //
Input
                                                                  PDWORD pLength,
       // Output
                                                                  DWORD
dwMilliseconds);// Input
       [DllImport("MPUSBAPI.dll" , EntryPoint="_MPUSBReadInt")]
```

```
	extern "C" DWORD MPUSBReadInt(HANDLE handle,  // Input
                                                                  PVOID pData, //
Output
                                                                  DWORD dwLen, //
Input
                                                                  PDWORD pLength,
	// Output
                                                                  DWORD
dwMilliseconds);// Input

	//Need this function for receiving all of the WM_DEVICECHANGE messages.
See MSDN documentation for
	//description of what this function does/how to use it. Note: name is
remapped "RegisterDeviceNotificationUM" to
	//avoid possible build error conflicts.
	[DllImport("user32.dll" , CharSet = CharSet::Seeifdef,
EntryPoint="RegisterDeviceNotification")]
	extern "C" HDEVNOTIFY WINAPI RegisterDeviceNotificationUM(
		HANDLE hRecipient,
		LPVOID NotificationFilter,
		DWORD Flags);

	//----------------Global variables used in this application---------------
----------------
	HANDLE  EP1INHandle = INVALID_HANDLE_VALUE;
	HANDLE  EP1OUTHandle = INVALID_HANDLE_VALUE;
	BOOL AttachedState = FALSE;		//Need to keep track of the USB
device attachment status for proper plug and play operation.
//-----------------------------------------------------------END CUT AND PASTE BLOCK-
-------------------------------------------------------------------------------
---

/* RFidReader v1.0 - Óscar Aragón Andreu
*****************************************************************************
**********************/
	unsigned char Buffer[64] = {0};		// Data Buffer
	unsigned char TxLength = 0;			// Length of Transmited Data
in bytes
	unsigned char RxLength = 0;			// Length of Received Data
in bytes
	unsigned char Command = 0;		// Service Command
	unsigned char Error = 0;		// Error
	unsigned char TX_HEADER = 2;		// Transmit Header, 2 bytes length:
Length & CMD
	unsigned char RX_HEADER = 3;		// Receive Header, 3 bytes length:
Length, CMD & Error
	BOOL SendFlag = FALSE;			// Send Flag
	BOOL ReceiveFlag = FALSE;		// Receive Flag
	unsigned char Flags = 0;		// ISO Flags
	int ArgumentsLength = 0;		// Arguments Length in bytes
	int ArgumentCounter = 0;		// Argument Counter
	int MaxAvailableMemory = 256;		// Default Max. Available Mamory
	int NrOfBlocks = 8;			// Default Nr. of Blocks
	int BlockLength = 32;			// Default Block Length
	int DataCounter = 0;			// Tag Data Counter
	BOOL UserMode = FALSE;			// User Mode Flag
/*****************************************************************************
********************** RFidReader v1.0 - Óscar Aragón Andreu */

//-----------------------------------------------------------------------------
-------------------------------------------------------------------------------
---

	public ref class Form1 : public System::Windows::Forms::Form
	{
	public:
		Form1(void)
```

```
            {
                  InitializeComponent();

//--------------------------------------------------------------------------
------------------------------------------------------------------------------
---
//-----------------------------------------------------BEGIN CUT AND PASTE
BLOCK-------------------------------------------------------------------------
-------
                  //Additional constructor code

                  //Globally Unique Identifier (GUID). Windows uses GUIDs to
identify things.
                  GUID InterfaceClassGuid = {0xa5dcbf10, 0x6530, 0x11d2, 0x90,
0x1F, 0x00, 0xC0, 0x4F, 0xB9, 0x51, 0xED}; //Globally Unique Identifier (GUID)
for USB peripheral devices

                  //Register for WM_DEVICECHANGE notifications:
                  DEV_BROADCAST_DEVICEINTERFACE MyDeviceBroadcastHeader;// =
new DEV_BROADCAST_HDR;
                  MyDeviceBroadcastHeader.dbcc_devicetype =
DBT_DEVTYP_DEVICEINTERFACE;
                  MyDeviceBroadcastHeader.dbcc_size =
sizeof(DEV_BROADCAST_DEVICEINTERFACE);
                  MyDeviceBroadcastHeader.dbcc_reserved = 0;    //Reserved says
not to use...
                  MyDeviceBroadcastHeader.dbcc_classguid = InterfaceClassGuid;
                  RegisterDeviceNotificationUM((HANDLE)this->Handle,
&MyDeviceBroadcastHeader, DEVICE_NOTIFY_WINDOW_HANDLE);

                  //Now perform an initial start up check of the device state
(attached or not attached), since we would not have
                  //received a WM_DEVICECHANGE notification.
                  if(MPUSBGetDeviceCount(DeviceVID_PID)) //Check and make sure
at least one device with matching VID/PID is attached
                  {
                        EP1OUTHandle = MPUSBOpen(0, DeviceVID_PID,
"\\MCHP_EP1", MP_WRITE, 0);
                        EP1INHandle = MPUSBOpen(0, DeviceVID_PID, "\\MCHP_EP1",
MP_READ, 0);

                        AttachedState = TRUE;
                        textBoxStatus->Text = "Status: RFid Reader Connected,
System Running";
                        //label2->Enabled = true; //Make the label no longer
greyed out
                  }
                  else  //Device must not be connected (or not programmed with
correct firmware)
                  {
                        textBoxStatus->Text = "Status: Device Not Detected,
Verify Connection/Correct Firmware";
                        AttachedState = FALSE;
                        //label2->Enabled = false;//Make the label greyed out
                  }
                  ReadWriteThread->RunWorkerAsync();     //Recommend performing
USB read/write operations in a separate thread.  Otherwise,

      //the Read/Write operations are effectively blocking functions and can
lock up the

      //user interface if the I/O operations take a long time to complete.
//-----------------------------------------------------END CUT AND PASTE BLOCK-
------------------------------------------------------------------------------
---
```

```
//-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
---
            }

      protected:
            /// <summary>
            /// Clean up any resources being used.
            /// </summary>
            ~Form1()
            {
//-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
---
//-----------------------------------------------------BEGIN CUT AND PASTE
BLOCK-------------------------------------------------------------------------
-------
                  //Make sure to close any open handles, before exiting the
application
                  if (EP1OUTHandle != INVALID_HANDLE_VALUE)
                        MPUSBClose (EP1OUTHandle);
                  if (EP1INHandle != INVALID_HANDLE_VALUE)
                        MPUSBClose (EP1INHandle);
//-------------------------------------------------------END CUT AND PASTE BLOCK-
-------------------------------------------------------------------------------
---
//-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
---

                  if (components)
                  {
                        delete components;
                  }
            }

      protected:
      private: System::ComponentModel::BackgroundWorker^  ReadWriteThread;
      private: System::Windows::Forms::Timer^  timer1;
      private: System::Windows::Forms::TextBox^  textBoxStatus;
      private: System::Windows::Forms::ListBox^  listBoxServices;
      private: System::Windows::Forms::Label^  labelArguments;
      private: System::Windows::Forms::TextBox^  textBoxArguments;
      private: System::Windows::Forms::TextBox^  textBoxLog;
      private: System::Windows::Forms::Button^  buttonSend;
      private: System::Windows::Forms::GroupBox^  groupBoxSystemConfiguration;
      private: System::Windows::Forms::CheckBox^  checkBoxDataRate;
      private: System::Windows::Forms::CheckBox^  checkBoxSubCarrier;
      private: System::Windows::Forms::Label^  labelBlockLength;
      private: System::Windows::Forms::Label^  labelNrOfBlocks;
      private: System::Windows::Forms::TextBox^  textBoxBlockLength;
      private: System::Windows::Forms::TextBox^  textBoxNrOfBlocks;
      private: System::Windows::Forms::GroupBox^  groupBoxServices;
      private: System::Windows::Forms::Label^  labelAvailableMemory;
      private: System::Windows::Forms::TextBox^  textBoxAvailableMemory;
      private: System::Windows::Forms::GroupBox^  groupBoxControls;
      private: System::Windows::Forms::Label^  labelAuthor;
      private: System::Windows::Forms::Button^  buttonClearError;
      private: System::Windows::Forms::Button^  buttonSaveLog;
      private: System::Windows::Forms::Button^  buttonExit;
      private: System::Windows::Forms::Button^  buttonHelp;
      private: System::Windows::Forms::Button^  buttonChangeMode;
      private: System::Windows::Forms::ListBox^  listBoxUsers;
      private: System::Windows::Forms::Button^  buttonRemoveUser;
      private: System::Windows::Forms::Button^  buttonAddUser;
      private: System::Windows::Forms::Button^  buttonAccessControl;
      private: System::Windows::Forms::Button^  buttonClearArguments;
```

```
        private: System::Windows::Forms::Label^  label1;
        private: System::Windows::Forms::GroupBox^  groupBox1;
        private: System::Windows::Forms::Label^  label2;
        private: System::ComponentModel::IContainer^  components;

        private:
                /// <summary>
                /// Required designer variable.
                /// </summary>


#pragma region Windows Form Designer generated code
                /// <summary>
                /// Required method for Designer support - do not modify
                /// the contents of this method with the code editor.
                /// </summary>
                void InitializeComponent(void)
                {
                        this->components = (gcnew
System::ComponentModel::Container());
                        this->ReadWriteThread = (gcnew
System::ComponentModel::BackgroundWorker());
                        this->timer1 = (gcnew System::Windows::Forms::Timer(this-
>components));
                        this->textBoxStatus = (gcnew
System::Windows::Forms::TextBox());
                        this->listBoxServices = (gcnew
System::Windows::Forms::ListBox());
                        this->labelArguments = (gcnew
System::Windows::Forms::Label());
                        this->textBoxArguments = (gcnew
System::Windows::Forms::TextBox());
                        this->textBoxLog = (gcnew System::Windows::Forms::TextBox());
                        this->buttonSend = (gcnew System::Windows::Forms::Button());
                        this->groupBoxSystemConfiguration = (gcnew
System::Windows::Forms::GroupBox());
                        this->labelAvailableMemory = (gcnew
System::Windows::Forms::Label());
                        this->textBoxAvailableMemory = (gcnew
System::Windows::Forms::TextBox());
                        this->labelBlockLength = (gcnew
System::Windows::Forms::Label());
                        this->labelNrOfBlocks = (gcnew
System::Windows::Forms::Label());
                        this->textBoxBlockLength = (gcnew
System::Windows::Forms::TextBox());
                        this->textBoxNrOfBlocks = (gcnew
System::Windows::Forms::TextBox());
                        this->checkBoxDataRate = (gcnew
System::Windows::Forms::CheckBox());
                        this->checkBoxSubCarrier = (gcnew
System::Windows::Forms::CheckBox());
                        this->groupBoxServices = (gcnew
System::Windows::Forms::GroupBox());
                        this->listBoxUsers = (gcnew
System::Windows::Forms::ListBox());
                        this->groupBoxControls = (gcnew
System::Windows::Forms::GroupBox());
                        this->buttonExit = (gcnew System::Windows::Forms::Button());
                        this->buttonHelp = (gcnew System::Windows::Forms::Button());
                        this->buttonClearError = (gcnew
System::Windows::Forms::Button());
                        this->buttonSaveLog = (gcnew
System::Windows::Forms::Button());
                        this->buttonChangeMode = (gcnew
System::Windows::Forms::Button());
```

```
                    this->buttonAddUser = (gcnew
System::Windows::Forms::Button());
                    this->buttonRemoveUser = (gcnew
System::Windows::Forms::Button());
                    this->buttonAccessControl = (gcnew
System::Windows::Forms::Button());
                    this->labelAuthor = (gcnew System::Windows::Forms::Label());
                    this->buttonClearArguments = (gcnew
System::Windows::Forms::Button());
                    this->label1 = (gcnew System::Windows::Forms::Label());
                    this->groupBox1 = (gcnew System::Windows::Forms::GroupBox());
                    this->label2 = (gcnew System::Windows::Forms::Label());
                    this->groupBoxSystemConfiguration->SuspendLayout();
                    this->groupBoxServices->SuspendLayout();
                    this->groupBoxControls->SuspendLayout();
                    this->groupBox1->SuspendLayout();
                    this->SuspendLayout();
                    //
                    // ReadWriteThread
                    //
                    this->ReadWriteThread->WorkerReportsProgress = true;
                    this->ReadWriteThread->DoWork += gcnew
System::ComponentModel::DoWorkEventHandler(this, &Form1::ReadWriteThread_DoWork);
                    //
                    // timer1
                    //
                    this->timer1->Enabled = true;
                    this->timer1->Interval = 16;
                    this->timer1->Tick += gcnew System::EventHandler(this,
&Form1::ShowResponse);
                    //
                    // textBoxStatus
                    //
                    this->textBoxStatus->Location = System::Drawing::Point(10,
387);
                    this->textBoxStatus->Name = L"textBoxStatus";
                    this->textBoxStatus->ReadOnly = true;
                    this->textBoxStatus->Size = System::Drawing::Size(659, 20);
                    this->textBoxStatus->TabIndex = 11;
                    //
                    // listBoxServices
                    //
                    this->listBoxServices->DisplayMember = L"(none)";
                    this->listBoxServices->FormattingEnabled = true;
                    this->listBoxServices->Items->AddRange(gcnew cli::array<
System::Object^  >(8) {L"ReadFirmwareVersion", L"WriteDefaultConfiguration",
                            L"ReadDefaultConfiguration", L"ISOGeneralCommand",
L"WriteSingleMultipleBlocks", L"ReadSingleMultipleBlocks", L"ClearTagData",
                            L"ResetCLRC632"});
                    this->listBoxServices->Location = System::Drawing::Point(7,
18);
                    this->listBoxServices->Name = L"listBoxServices";
                    this->listBoxServices->ScrollAlwaysVisible = true;
                    this->listBoxServices->Size = System::Drawing::Size(157,
173);
                    this->listBoxServices->TabIndex = 13;
                    //
                    // labelArguments
                    //
                    this->labelArguments->AutoSize = true;
                    this->labelArguments->ForeColor =
System::Drawing::SystemColors::ActiveCaption;
                    this->labelArguments->Location = System::Drawing::Point(189,
13);
                    this->labelArguments->Name = L"labelArguments";
                    this->labelArguments->Size = System::Drawing::Size(60, 13);
                    this->labelArguments->TabIndex = 15;
```

```
                    this->labelArguments->Text = L"Arguments:";
                    //
                    // textBoxArguments
                    //
                    this->textBoxArguments->Location =
System::Drawing::Point(253, 10);
                    this->textBoxArguments->MaxLength = 128;
                    this->textBoxArguments->Name = L"textBoxArguments";
                    this->textBoxArguments->Size = System::Drawing::Size(248,
20);
                    this->textBoxArguments->TabIndex = 16;
                    //
                    // textBoxLog
                    //
                    this->textBoxLog->BackColor =
System::Drawing::SystemColors::ActiveCaptionText;
                    this->textBoxLog->Location = System::Drawing::Point(187, 31);
                    this->textBoxLog->Multiline = true;
                    this->textBoxLog->Name = L"textBoxLog";
                    this->textBoxLog->ReadOnly = true;
                    this->textBoxLog->ScrollBars =
System::Windows::Forms::ScrollBars::Vertical;
                    this->textBoxLog->Size = System::Drawing::Size(343, 350);
                    this->textBoxLog->TabIndex = 17;
                    //
                    // buttonSend
                    //
                    this->buttonSend->Location = System::Drawing::Point(17, 21);
                    this->buttonSend->Name = L"buttonSend";
                    this->buttonSend->Size = System::Drawing::Size(99, 28);
                    this->buttonSend->TabIndex = 18;
                    this->buttonSend->Text = L"Send";
                    this->buttonSend->UseVisualStyleBackColor = true;
                    this->buttonSend->Click += gcnew System::EventHandler(this,
&Form1::RunRequest);
                    //
                    // groupBoxSystemConfiguration
                    //
                    this->groupBoxSystemConfiguration->Controls->Add(this-
>labelAvailableMemory);
                    this->groupBoxSystemConfiguration->Controls->Add(this-
>textBoxAvailableMemory);
                    this->groupBoxSystemConfiguration->Controls->Add(this-
>labelBlockLength);
                    this->groupBoxSystemConfiguration->Controls->Add(this-
>labelNrOfBlocks);
                    this->groupBoxSystemConfiguration->Controls->Add(this-
>textBoxBlockLength);
                    this->groupBoxSystemConfiguration->Controls->Add(this-
>textBoxNrOfBlocks);
                    this->groupBoxSystemConfiguration->Controls->Add(this-
>checkBoxDataRate);
                    this->groupBoxSystemConfiguration->Controls->Add(this-
>checkBoxSubCarrier);
                    this->groupBoxSystemConfiguration->Location =
System::Drawing::Point(10, 217);
                    this->groupBoxSystemConfiguration->Name =
L"groupBoxSystemConfiguration";
                    this->groupBoxSystemConfiguration->Size =
System::Drawing::Size(171, 164);
                    this->groupBoxSystemConfiguration->TabIndex = 20;
                    this->groupBoxSystemConfiguration->TabStop = false;
                    this->groupBoxSystemConfiguration->Text = L"System
Configuration";
                    //
                    // labelAvailableMemory
                    //
```

```
                    this->labelAvailableMemory->AutoSize = true;
                    this->labelAvailableMemory->Location =
System::Drawing::Point(49, 133);
                    this->labelAvailableMemory->Name = L"labelAvailableMemory";
                    this->labelAvailableMemory->Size = System::Drawing::Size(113,
13);
                    this->labelAvailableMemory->TabIndex = 7;
                    this->labelAvailableMemory->Text = L"Availabe Memory (bits)";
                    //
                    // textBoxAvailableMemory
                    //
                    this->textBoxAvailableMemory->Location =
System::Drawing::Point(7, 130);
                    this->textBoxAvailableMemory->MaxLength = 4;
                    this->textBoxAvailableMemory->Name =
L"textBoxAvailableMemory";
                    this->textBoxAvailableMemory->ReadOnly = true;
                    this->textBoxAvailableMemory->Size =
System::Drawing::Size(36, 20);
                    this->textBoxAvailableMemory->TabIndex = 6;
                    this->textBoxAvailableMemory->Text = L"256";
                    //
                    // labelBlockLength
                    //
                    this->labelBlockLength->AutoSize = true;
                    this->labelBlockLength->Location = System::Drawing::Point(49,
104);
                    this->labelBlockLength->Name = L"labelBlockLength";
                    this->labelBlockLength->Size = System::Drawing::Size(95, 13);
                    this->labelBlockLength->TabIndex = 5;
                    this->labelBlockLength->Text = L"Block Length (bits)";
                    //
                    // labelNrOfBlocks
                    //
                    this->labelNrOfBlocks->AutoSize = true;
                    this->labelNrOfBlocks->Location = System::Drawing::Point(49,
75);
                    this->labelNrOfBlocks->Name = L"labelNrOfBlocks";
                    this->labelNrOfBlocks->Size = System::Drawing::Size(68, 13);
                    this->labelNrOfBlocks->TabIndex = 4;
                    this->labelNrOfBlocks->Text = L"Nr. of Blocks";
                    //
                    // textBoxBlockLength
                    //
                    this->textBoxBlockLength->Location =
System::Drawing::Point(7, 101);
                    this->textBoxBlockLength->MaxLength = 2;
                    this->textBoxBlockLength->Name = L"textBoxBlockLength";
                    this->textBoxBlockLength->Size = System::Drawing::Size(36,
20);
                    this->textBoxBlockLength->TabIndex = 3;
                    this->textBoxBlockLength->Text = L"32";
                    this->textBoxBlockLength->KeyPress += gcnew
System::Windows::Forms::KeyPressEventHandler(this,
&Form1::UpdateAvailableMemory);
                    //
                    // textBoxNrOfBlocks
                    //
                    this->textBoxNrOfBlocks->Location = System::Drawing::Point(7,
72);
                    this->textBoxNrOfBlocks->MaxLength = 3;
                    this->textBoxNrOfBlocks->Name = L"textBoxNrOfBlocks";
                    this->textBoxNrOfBlocks->Size = System::Drawing::Size(36,
20);
                    this->textBoxNrOfBlocks->TabIndex = 2;
                    this->textBoxNrOfBlocks->Text = L"8";
```

```
                    this->textBoxNrOfBlocks->KeyPress += gcnew
System::Windows::Forms::KeyPressEventHandler(this,
&Form1::UpdateAvailableMemory);
                    //
                    // checkBoxDataRate
                    //
                    this->checkBoxDataRate->AutoSize = true;
                    this->checkBoxDataRate->Checked = true;
                    this->checkBoxDataRate->CheckState =
System::Windows::Forms::CheckState::Checked;
                    this->checkBoxDataRate->Location = System::Drawing::Point(7,
45);
                    this->checkBoxDataRate->Name = L"checkBoxDataRate";
                    this->checkBoxDataRate->Size = System::Drawing::Size(120,
17);
                    this->checkBoxDataRate->TabIndex = 1;
                    this->checkBoxDataRate->Text = L"Low/High Data rate";
                    this->checkBoxDataRate->UseVisualStyleBackColor = true;
                    this->checkBoxDataRate->CheckStateChanged += gcnew
System::EventHandler(this, &Form1::UpdateFlags);
                    //
                    // checkBoxSubCarrier
                    //
                    this->checkBoxSubCarrier->AutoSize = true;
                    this->checkBoxSubCarrier->Checked = true;
                    this->checkBoxSubCarrier->CheckState =
System::Windows::Forms::CheckState::Checked;
                    this->checkBoxSubCarrier->Location =
System::Drawing::Point(7, 22);
                    this->checkBoxSubCarrier->Name = L"checkBoxSubCarrier";
                    this->checkBoxSubCarrier->Size = System::Drawing::Size(137,
17);
                    this->checkBoxSubCarrier->TabIndex = 0;
                    this->checkBoxSubCarrier->Text = L"Single/Dual Sub-Carrier";
                    this->checkBoxSubCarrier->UseVisualStyleBackColor = true;
                    this->checkBoxSubCarrier->CheckStateChanged += gcnew
System::EventHandler(this, &Form1::UpdateFlags);
                    //
                    // groupBoxServices
                    //
                    this->groupBoxServices->Controls->Add(this->listBoxServices);
                    this->groupBoxServices->Controls->Add(this->listBoxUsers);
                    this->groupBoxServices->Location = System::Drawing::Point(10,
13);
                    this->groupBoxServices->Name = L"groupBoxServices";
                    this->groupBoxServices->Size = System::Drawing::Size(171,
198);
                    this->groupBoxServices->TabIndex = 21;
                    this->groupBoxServices->TabStop = false;
                    this->groupBoxServices->Text = L"Services List";
                    //
                    // listBoxUsers
                    //
                    this->listBoxUsers->DisplayMember = L"True";
                    this->listBoxUsers->FormattingEnabled = true;
                    this->listBoxUsers->Location = System::Drawing::Point(7, 18);
                    this->listBoxUsers->Name = L"listBoxUsers";
                    this->listBoxUsers->ScrollAlwaysVisible = true;
                    this->listBoxUsers->Size = System::Drawing::Size(157, 173);
                    this->listBoxUsers->TabIndex = 14;
                    this->listBoxUsers->Visible = false;
                    //
                    // groupBoxControls
                    //
                    this->groupBoxControls->Controls->Add(this->buttonExit);
                    this->groupBoxControls->Controls->Add(this->buttonHelp);
                    this->groupBoxControls->Controls->Add(this->buttonSend);
```

```
                    this->groupBoxControls->Controls->Add(this-
>buttonClearError);
                    this->groupBoxControls->Controls->Add(this->buttonSaveLog);
                    this->groupBoxControls->Controls->Add(this-
>buttonChangeMode);
                    this->groupBoxControls->Controls->Add(this->buttonAddUser);
                    this->groupBoxControls->Controls->Add(this-
>buttonRemoveUser);
                    this->groupBoxControls->Controls->Add(this-
>buttonAccessControl);
                    this->groupBoxControls->Location =
System::Drawing::Point(537, 10);
                    this->groupBoxControls->Name = L"groupBoxControls";
                    this->groupBoxControls->Size = System::Drawing::Size(132,
233);
                    this->groupBoxControls->TabIndex = 22;
                    this->groupBoxControls->TabStop = false;
                    this->groupBoxControls->Text = L"Controls";
                    //
                    // buttonExit
                    //
                    this->buttonExit->Location = System::Drawing::Point(17, 191);
                    this->buttonExit->Name = L"buttonExit";
                    this->buttonExit->Size = System::Drawing::Size(99, 28);
                    this->buttonExit->TabIndex = 22;
                    this->buttonExit->Text = L"Exit";
                    this->buttonExit->UseVisualStyleBackColor = true;
                    this->buttonExit->Click += gcnew System::EventHandler(this,
&Form1::Exit);
                    //
                    // buttonHelp
                    //
                    this->buttonHelp->Location = System::Drawing::Point(17, 157);
                    this->buttonHelp->Name = L"buttonHelp";
                    this->buttonHelp->Size = System::Drawing::Size(99, 28);
                    this->buttonHelp->TabIndex = 21;
                    this->buttonHelp->Text = L"Help";
                    this->buttonHelp->UseVisualStyleBackColor = true;
                    this->buttonHelp->Click += gcnew System::EventHandler(this,
&Form1::ShowHelp);
                    //
                    // buttonClearError
                    //
                    this->buttonClearError->Location = System::Drawing::Point(17,
55);
                    this->buttonClearError->Name = L"buttonClearError";
                    this->buttonClearError->Size = System::Drawing::Size(99, 28);
                    this->buttonClearError->TabIndex = 19;
                    this->buttonClearError->Text = L"Clear Error";
                    this->buttonClearError->UseVisualStyleBackColor = true;
                    this->buttonClearError->Click += gcnew
System::EventHandler(this, &Form1::ClearError);
                    //
                    // buttonSaveLog
                    //
                    this->buttonSaveLog->Location = System::Drawing::Point(17,
89);
                    this->buttonSaveLog->Name = L"buttonSaveLog";
                    this->buttonSaveLog->Size = System::Drawing::Size(99, 28);
                    this->buttonSaveLog->TabIndex = 20;
                    this->buttonSaveLog->Text = L"Save Log";
                    this->buttonSaveLog->UseVisualStyleBackColor = true;
                    this->buttonSaveLog->Click += gcnew
System::EventHandler(this, &Form1::SaveLog);
                    //
                    // buttonChangeMode
                    //
```

```
                    this->buttonChangeMode->Location = System::Drawing::Point(17,
123);
                    this->buttonChangeMode->Name = L"buttonChangeMode";
                    this->buttonChangeMode->Size = System::Drawing::Size(99, 28);
                    this->buttonChangeMode->TabIndex = 23;
                    this->buttonChangeMode->Text = L"User Mode";
                    this->buttonChangeMode->UseVisualStyleBackColor = true;
                    this->buttonChangeMode->Click += gcnew
System::EventHandler(this, &Form1::ChangeMode);
                    //
                    // buttonAddUser
                    //
                    this->buttonAddUser->Location = System::Drawing::Point(17,
21);
                    this->buttonAddUser->Name = L"buttonAddUser";
                    this->buttonAddUser->Size = System::Drawing::Size(99, 28);
                    this->buttonAddUser->TabIndex = 24;
                    this->buttonAddUser->Text = L"Add User";
                    this->buttonAddUser->UseVisualStyleBackColor = true;
                    this->buttonAddUser->Visible = false;
                    this->buttonAddUser->Click += gcnew
System::EventHandler(this, &Form1::AddUser);
                    //
                    // buttonRemoveUser
                    //
                    this->buttonRemoveUser->Location = System::Drawing::Point(17,
55);
                    this->buttonRemoveUser->Name = L"buttonRemoveUser";
                    this->buttonRemoveUser->Size = System::Drawing::Size(99, 28);
                    this->buttonRemoveUser->TabIndex = 25;
                    this->buttonRemoveUser->Text = L"Remove User";
                    this->buttonRemoveUser->UseVisualStyleBackColor = true;
                    this->buttonRemoveUser->Visible = false;
                    this->buttonRemoveUser->Click += gcnew
System::EventHandler(this, &Form1::RemoveUser);
                    //
                    // buttonAccessControl
                    //
                    this->buttonAccessControl->Location =
System::Drawing::Point(17, 89);
                    this->buttonAccessControl->Name = L"buttonAccessControl";
                    this->buttonAccessControl->Size = System::Drawing::Size(99,
28);
                    this->buttonAccessControl->TabIndex = 26;
                    this->buttonAccessControl->Text = L"Access Control";
                    this->buttonAccessControl->UseVisualStyleBackColor = true;
                    this->buttonAccessControl->Visible = false;
                    this->buttonAccessControl->Click += gcnew
System::EventHandler(this, &Form1::AccessControl);
                    //
                    // labelAuthor
                    //
                    this->labelAuthor->AutoSize = true;
                    this->labelAuthor->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 8,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                        static_cast<System::Byte>(0)));
                    this->labelAuthor->Location = System::Drawing::Point(11, 81);
                    this->labelAuthor->Name = L"labelAuthor";
                    this->labelAuthor->Size = System::Drawing::Size(109, 13);
                    this->labelAuthor->TabIndex = 23;
                    this->labelAuthor->Text = L"Óscar Aragón Andreu";
                    //
                    // buttonClearArguments
                    //
                    this->buttonClearArguments->Location =
System::Drawing::Point(502, 9);
```

```
                    this->buttonClearArguments->Name = L"buttonClearArguments";
                    this->buttonClearArguments->Size = System::Drawing::Size(29,
22);
                    this->buttonClearArguments->TabIndex = 24;
                    this->buttonClearArguments->Text = L"<<";
                    this->buttonClearArguments->UseVisualStyleBackColor = true;
                    this->buttonClearArguments->Click += gcnew
System::EventHandler(this, &Form1::ClearArguments);
                    //
                    // label1
                    //
                    this->label1->AutoSize = true;
                    this->label1->BackColor =
System::Drawing::SystemColors::Control;
                    this->label1->Font = (gcnew System::Drawing::Font(L"Tahoma",
21.75F, System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
                            static_cast<System::Byte>(0)));
                    this->label1->ForeColor =
System::Drawing::SystemColors::Desktop;
                    this->label1->Location = System::Drawing::Point(22, 31);
                    this->label1->Name = L"label1";
                    this->label1->Size = System::Drawing::Size(87, 35);
                    this->label1->TabIndex = 25;
                    this->label1->Text = L"ETSE";
                    //
                    // groupBox1
                    //
                    this->groupBox1->BackgroundImageLayout =
System::Windows::Forms::ImageLayout::None;
                    this->groupBox1->Controls->Add(this->label2);
                    this->groupBox1->Controls->Add(this->label1);
                    this->groupBox1->Controls->Add(this->labelAuthor);
                    this->groupBox1->ForeColor =
System::Drawing::SystemColors::ControlText;
                    this->groupBox1->Location = System::Drawing::Point(537, 243);
                    this->groupBox1->Name = L"groupBox1";
                    this->groupBox1->Size = System::Drawing::Size(132, 138);
                    this->groupBox1->TabIndex = 26;
                    this->groupBox1->TabStop = false;
                    //
                    // label2
                    //
                    this->label2->AutoSize = true;
                    this->label2->Font = (gcnew System::Drawing::Font(L"Microsoft
Sans Serif", 8, System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
                            static_cast<System::Byte>(0)));
                    this->label2->Location = System::Drawing::Point(21, 98);
                    this->label2->Name = L"label2";
                    this->label2->Size = System::Drawing::Size(87, 13);
                    this->label2->TabIndex = 26;
                    this->label2->Text = L"PFC ETIEI, 2009";
                    //
                    // Form1
                    //
                    this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
                    this->AutoScaleMode =
System::Windows::Forms::AutoScaleMode::Font;
                    this->ClientSize = System::Drawing::Size(678, 420);
                    this->ControlBox = false;
                    this->Controls->Add(this->groupBox1);
                    this->Controls->Add(this->buttonClearArguments);
                    this->Controls->Add(this->groupBoxControls);
                    this->Controls->Add(this->groupBoxServices);
                    this->Controls->Add(this->groupBoxSystemConfiguration);
                    this->Controls->Add(this->textBoxLog);
                    this->Controls->Add(this->textBoxArguments);
```

```
                    this->Controls->Add(this->labelArguments);
                    this->Controls->Add(this->textBoxStatus);
                    this->FormBorderStyle =
System::Windows::Forms::FormBorderStyle::FixedDialog;
                    this->Name = L"Form1";
                    this->StartPosition =
System::Windows::Forms::FormStartPosition::CenterScreen;
                    this->Text = L"RFid Reader v1.0";
                    this->groupBoxSystemConfiguration->ResumeLayout(false);
                    this->groupBoxSystemConfiguration->PerformLayout();
                    this->groupBoxServices->ResumeLayout(false);
                    this->groupBoxControls->ResumeLayout(false);
                    this->groupBox1->ResumeLayout(false);
                    this->groupBox1->PerformLayout();
                    this->ResumeLayout(false);
                    this->PerformLayout();

             }
#pragma endregion



//-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
---
//-----------------------------------------------------BEGIN CUT AND PASTE
BLOCK-------------------------------------------------------------------------
-------

/* RFidReader v1.0 - Óscar Aragón Andreu
*****************************************************************************
************************/
private: System::Void ReadWriteThread_DoWork(System::Object^  sender,
System::ComponentModel::DoWorkEventArgs^  e)
              {
                    //This thread does the actual USB read/write operations (but
only when AttachedState == TRUE).
                    //Since this is a separate thread, this code below executes
asynchronously from the reset of the
                    //code in this application.

                    DWORD ActualLength;

                    while(true)
                    {
                          if(AttachedState == TRUE && SendFlag == TRUE) //Do not
try and send USB traffic unless the device is actually attached,
enumerated/configured and Send Operation is requested
                          {
                                if(MPUSBWrite(EP1OUTHandle, Buffer, TxLength,
&ActualLength, 1000))     //Send the command now over USB
                                {
                                      TxLength = 0;       // Clear Tx Length
                                      SendFlag = FALSE;   // Clear Send Flag

                                      if(MPUSBRead(EP1INHandle, Buffer, 64,
&ActualLength, 60000))    //Receive the answer from the device firmware through
USB
                                      {
                                            RxLength = Buffer[0];     //
RX_HEADER.Length
                                            Command = Buffer[1];      //
RX_HEADER.Command
                                            Error = Buffer[2];        //
RX_HEADER.Error
```

```
                                                ReceiveFlag = TRUE;       // Set
Receive Flag
                                        }
                                }
                        }
                        Sleep(4);    //Add a small delay.  Otherwise, this
while(true) loop executes at warp speed and can cause
                                                //tangible CPU utilization
increase, with no particular benefit to the application.
                }
        }
/*************************************************************************
************************ RFidReader v1.0 - Óscar Aragón Andreu */

protected: virtual void WndProc( Message% m ) override{
        //This is a callback function that gets called when a Windows
message is received by the form.

        // Listen for Windows messages.  We will receive various different
types of messages, but the ones we really want to use are the WM_DEVICECHANGE
messages.
        if(m.Msg == WM_DEVICECHANGE)
        {
                if(((int)m.WParam == DBT_DEVICEARRIVAL) || ((int)m.WParam ==
DBT_DEVICEREMOVEPENDING) || ((int)m.WParam == DBT_DEVICEREMOVECOMPLETE) ||
((int)m.WParam == DBT_CONFIGCHANGED) )
                {

                        //WM_DEVICECHANGE messages by themselves are quite
generic, and can be caused by a number of different
                        //sources, not just your USB hardware device.
Therefore, must check to find out if any changes relavant
                        //to your device (with known VID/PID) took place before
doing any kind of opening or closing of endpoints.
                        //(the message could have been totally unrelated to
your application/USB device)
                        if(MPUSBGetDeviceCount(DeviceVID_PID)) //Check and
make sure at least one device with matching VID/PID is attached
                        {
                                if(AttachedState == FALSE)
                                {
                                        EP1OUTHandle = MPUSBOpen(0, DeviceVID_PID,
"\\MCHP_EP1", MP_WRITE, 0);
                                        EP1INHandle = MPUSBOpen(0, DeviceVID_PID,
"\\MCHP_EP1", MP_READ, 0);

                                        AttachedState = TRUE;
                                        textBoxStatus->Text = "Status: RFid Reader
Connected, System Running";
                                        //label2->Enabled = true; //Make the
label no longer greyed out
                                }
                                //else AttachedState == TRUE, in this case, do
not try to re-open already open and functioning endpoints.  Doing
                                //so will break them and make them no longer
work.
                        }
                        else   //No devices attached, WM_DEVICECHANGE message
must have been caused by your USB device falling off the bus (IE: user
unplugged/powered down)
                        {
                                if(MPUSBClose(EP1OUTHandle))
                                        EP1OUTHandle = INVALID_HANDLE_VALUE;
                                if(MPUSBClose(EP1INHandle))
                                        EP1INHandle = INVALID_HANDLE_VALUE;

                                AttachedState = FALSE;
```

```
                                    textBoxStatus->Text = "Status: Device Not
Detected, Verify Connection/Correct Firmware";
                              }
                        }
                  }

                  Form::WndProc( m );
            }
//-----------------------------------------------------END CUT AND PASTE BLOCK-
-------------------------------------------------------------------------------
---
//-----------------------------------------------------------------------------
-------------------------------------------------------------------------------
---

/* RFidReader v1.0 - Óscar Aragón Andreu
*****************************************************************************
**********************/
private: System::Void RunRequest(System::Object^  sender, System::EventArgs^  e)
                  {

                        if( textBoxStatus->Text == "Status: Device Not Detected,
Verify Connection/Correct Firmware" ) // If Device is not connected
                        {
                              MessageBox::Show("Device Not Detected, Verify
Connection/Correct Firmware");   // Show Error
                              return;      // Exit
                        }

                        String ^req = "";  // Request String
                        String ^cmd = "";  // Service Command
                        unsigned char HEXCounter = 0; // Counter for HEX Data
                        unsigned char ASCIICounter = 0; // Counter for ASCII Data

                        String ^arg = textBoxArguments->Text; // Get Arguments
                        array<Char>^arg_array = arg->ToCharArray(); // Arguments to
Char Array

                        if(textBoxArguments->Text == "" ) // If no arguments
                              arg = "None";

                        textBoxLog->BackColor =
System::Drawing::SystemColors::ActiveCaptionText;    // Back Color to Active
Caption

                        switch(listBoxServices->SelectedIndex) // Switch by command
                        {

      //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++//

                              case 0: // Command 0: Read Firmware Version
                                    cmd = "Read Firmware Version";
                                    arg = "None";

                                    // Load OUT DATA to Buffer
                                    TxLength = TX_HEADER;     // n BYTES to send
                                    Buffer[0] = TxLength;     // Tx is n Bytes
length
                                    Buffer[1] = READ_FIRMWARE_VERSION;     //
Command is Read Firmware Version

                                    // Show Request in LOG Window
                                    textBoxLog->Text = "REQUEST:\r\n" + "\r\n";
```

```
                                for( unsigned char i = 0; i < TxLength; i++)  //
Show Request in HEX format
                                {
                                       textBoxLog->Text += " " + req-
>Format("{0:X2}", Buffer[i]);
                                }// end for( unsigned char i = 0; i < TxLength;
i++)

                                textBoxLog->Text += "     ";

                                for( unsigned char i = 0; i < TxLength; i++)  //
Show Request in ASCII format
                                {
                                       textBoxLog->Text += " " +
Char::ToString(Buffer[i]);
                                }// end for( unsigned char i = 0; i < TxLength;
i++)

                                // Show Alphanumeric Data
                                textBoxLog->Text += "\r\n" + "\r\nCommand:\t" +
cmd;   // Show Request Command
                                textBoxLog->Text += "\r\nArguments:     " + arg;
       // Show Request Arguments
                                textBoxLog->Text += "\r\nRequest Length:\t" +
req->Format("{0:D}", TxLength) + " Bytes";    // Show Request Length in bytes

                                // Enable Transmission
                                SendFlag = TRUE;    // Set Send Flag
                                break;                    // Finish


       //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++//

                          case 1: // Command 1: Write Default Configuration
                                cmd = "Write Default Configuration";
                                arg = "None";

                                // Load OUT DATA to Buffer
                                TxLength = TX_HEADER;     // n BYTES to send
                                Buffer[0] = TxLength;     // Tx is n Bytes
length
                                Buffer[1] = WRITE_DEFAULT_CONFIG;

                                // Show Request in LOG Window
                                textBoxLog->Text = "REQUEST:\r\n" + "\r\n";


                                for( unsigned char i = 0; i < TxLength; i++)  //
Show Request in HEX format
                                {
                                       textBoxLog->Text += " " + req-
>Format("{0:X2}", Buffer[i]);
                                }// end for( unsigned char i = 0; i < TxLength;
i++)

                                textBoxLog->Text += "     ";

                                for( unsigned char i = 0; i < TxLength; i++)  //
Show Request in ASCII format
                                {
                                       textBoxLog->Text += " " +
Char::ToString(Buffer[i]);
                                }// end for( unsigned char i = 0; i < TxLength;
i++)

                                // Show Alphanumeric Data
```

```
                                textBoxLog->Text += "\r\n" + "\r\nCommand:\t" +
cmd;   // Show Request Command
                                textBoxLog->Text += "\r\nArguments:     " + arg;
      // Show Request Arguments
                                textBoxLog->Text += "\r\nRequest Length:\t" +
req->Format("{0:D}", TxLength) + " Bytes";    // Show Request Length in bytes

                                // Enable Transmission
                                SendFlag = TRUE;    // Set Send Flag
                                break;                    // Finish


      //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++//

                          case 2: // Command 2: Read Default Configuration
                                cmd = "Read Default Configuration";
                                arg = "None";

                                // Load OUT Data into Buffer
                                TxLength = TX_HEADER;                    // n
BYTES to send
                                Buffer[0] = TxLength;                           //
OUTPacket.Length
                                Buffer[1] = READ_DEFAULT_CONFIG; //
OUTPacket.CMD

                                // Show Request in LOG Window
                                textBoxLog->Text = "REQUEST:\r\n" + "\r\n";


                                for( unsigned char i = 0; i < TxLength; i++)  //
Show Request in HEX format
                                {
                                      textBoxLog->Text += " " + req-
>Format("{0:X2}", Buffer[i]);
                                }// end for( unsigned char i = 0; i < TxLength;
i++)

                                textBoxLog->Text += "     ";

                                for( unsigned char i = 0; i < TxLength; i++)  //
Show Request in ASCII format
                                {
                                      textBoxLog->Text += " " +
Char::ToString(Buffer[i]);
                                }// end for( unsigned char i = 0; i < TxLength;
i++)


                                // Show Alphanumeric Data
                                textBoxLog->Text += "\r\n" + "\r\nCommand:\t" +
cmd;   // Show Request Command
                                textBoxLog->Text += "\r\nArguments:\t" + arg; //
Show Request Arguments
                                textBoxLog->Text += "\r\nRequest Length:\t" +
req->Format("{0:D}", Buffer[0]) + " Bytes";   // Show Request Length in bytes

                                // Enable Transmission
                                SendFlag = TRUE;    // Set Send Flag
                                break;                    // Finish


      //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++//

                          case 3: // Command 3: ISO General Command
                                cmd = "ISO General Command";
```

```
                                // Get Arguments
                                if( GetArguments() == 1 ) // Process Arguments
and Check if error occurred
                                        return;      // If error occured Exit

                                if( ArgumentCounter == 0 )// ISO General Command
requieres Arguments
                                {
                                        MessageBox::Show("ERROR: Arguments
required, click on Help");// Show Error
                                        SendFlag = FALSE;   // Disable Transmit
                                        return;                         // Exit
immediately
                                }// end if(arg == "None")

                                // Get Arguments
                                if( GetArguments() == 1 ) // Process Arguments
and Check if error occurred
                                        return;      // If error occured Exit

                                // Load OUT Data into Buffer
                                TxLength = TX_HEADER + ArgumentsLength;// n
BYTES to send
                                Buffer[0] = TxLength;
        // OUTPacket.Length
                                Buffer[1] = ISO_GENERAL_COMMAND;        //
OUTPacket.CMD

                                // Show Request in LOG Window
                                textBoxLog->Text = "REQUEST:\r\n" + "\r\n";

                                for( HEXCounter = 0; HEXCounter < TX_HEADER;
HEXCounter++)// Show Request TX_HEADER in HEX format
                                {
                                        textBoxLog->Text += " " + req-
>Format("{0:X2}", Buffer[HEXCounter]);
                                }// end for( HEXCounter = 0; HEXCounter <
TX_HEADER; HEXCounter++)

                                textBoxLog->Text += "     ";

                                for( ASCIICounter = 0; ASCIICounter < TX_HEADER;
ASCIICounter++)     // Show Request TX_HEADER in ASCII format, error will not
shown
                                {
                                        textBoxLog->Text += " " +
Char::ToString(Buffer[ASCIICounter]);
                                }// end for( ASCIICounter = 0; ASCIICounter <
TX_HEADER; ASCIICounter++)

                                while( HEXCounter && ASCIICounter < TxLength )
        // Show Arguments in HEX and ASCII format
                                {
                                        textBoxLog->Text += "\r\n";
                                        for( unsigned char i = 0; i < 8; i++)   //
Show HEX Arguments in 8 Bytes blocks
                                        {
                                         textBoxLog->Text += " " + req-
>Format("{0:X2}", Buffer[HEXCounter]);
                                         HEXCounter++;
                                         if(HEXCounter == TxLength)
                                                break;
                                        }// end for( unsigned char i = 0; i < 8;
i++)

                                        textBoxLog->Text += "     ";
```

```
                                        for( unsigned char i = 0; i < 8; i++)   //
Show ASCII Arguments in 8 Bytes blocks
                                        {
                                         textBoxLog->Text += " " +
Char::ToString(Buffer[ASCIICounter]);
                                         ASCIICounter++;
                                         if(ASCIICounter == TxLength)
                                                break;
                                        }// end for( unsigned char i = 0; i < 8;
i++)
                                }// end while( HEXCounter && ASCIICounter <
RxLength )

                                // Show Alphanumeric Data
                                textBoxLog->Text += "\r\n" + "\r\nCommand:\t" +
cmd;   // Show Request Command
                                textBoxLog->Text += "\r\nArguments:\t" + arg; //
Show Request Arguments
                                textBoxLog->Text += "\r\nRequest Length:\t" +
req->Format("{0:D}", Buffer[0]) + " Bytes";   // Show Request Length in bytes

                                // Enable Transmission
                                SendFlag = TRUE;    // Set Send Flag
                                break;                  // Finish


      //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++//

                        case 4: // Command 4: Write Single/Multiple Blocks
                                cmd = "Write Single/Multiple Blocks";

                                // Get Write Single/Multiple Blocks Arguments
and check if error occurred
                                if( Get_WSMB_Arguments() == 1 )
                                        return;      // If error occured Exit

                                if(ArgumentCounter != 2 ) // Write Multiple
blocks requieres 2 Arguments
                                {
                                        MessageBox::Show("ERROR: 2 Arguments
required, click on Help");
                                        SendFlag = FALSE;
                                        return;
                                }// end if(ArgumentCounter != 2 )


                                // Load OUT Data into Buffer
                                TxLength = TX_HEADER + 1 + ArgumentsLength;   //
n BYTES to send
                                Buffer[0] = TxLength;
             // OUTPacket.Length
                                Buffer[1] = WRITE_SINGLE_MULTIPLE_BLOCKS;     //
OUTPacket.CMD
                                Buffer[2] = Flags;
             // OUTPacket.Flags

                                // Show Request in LOG Window
                                textBoxLog->Text = "REQUEST:\r\n" + "\r\n";

                                for( HEXCounter = 0; HEXCounter < TX_HEADER;
HEXCounter++)// Show Request TX_HEADER in HEX format
                                {
                                        textBoxLog->Text += " " + req-
>Format("{0:X2}", Buffer[HEXCounter]);
```

```
                                }// end for( HEXCounter = 0; HEXCounter <
TX_HEADER; HEXCounter++)

                                textBoxLog->Text += "     ";

                                for( ASCIICounter = 0; ASCIICounter < TX_HEADER;
ASCIICounter++)     // Show Request TX_HEADER in ASCII format, error will not
shown
                                {
                                        textBoxLog->Text += " " +
Char::ToString(Buffer[ASCIICounter]);
                                }// end for( ASCIICounter = 0; ASCIICounter <
TX_HEADER; ASCIICounter++)

                                while( HEXCounter && ASCIICounter < TxLength )
      // Show Arguments in HEX and ASCII format
                                {
                                        textBoxLog->Text += "\r\n";
                                        for( unsigned char i = 0; i < 8; i++)  //
Show HEX Arguments in 8 Bytes blocks
                                        {
                                         textBoxLog->Text += " " + req-
>Format("{0:X2}", Buffer[HEXCounter]);
                                         HEXCounter++;
                                         if(HEXCounter == TxLength)
                                                break;
                                        }// end for( unsigned char i = 0; i < 8;
i++)

                                        textBoxLog->Text += "     ";

                                        for( unsigned char i = 0; i < 8; i++)  //
Show ASCII Arguments in 8 Bytes blocks
                                        {
                                         textBoxLog->Text += " " +
Char::ToString(Buffer[ASCIICounter]);
                                         ASCIICounter++;
                                         if(ASCIICounter == TxLength)
                                                break;
                                        }// end for( unsigned char i = 0; i < 8;
i++)
                                }// end while( HEXCounter && ASCIICounter <
RxLength )

                                // Show Alphanumeric Data
                                textBoxLog->Text += "\r\n" + "\r\nCommand:\t" +
cmd;   // Show Request Command
                                textBoxLog->Text += "\r\nFlags:\t\t" + + req-
>Format("{0:D}", Buffer[2]);     // Show Request Flags
                                textBoxLog->Text += "\r\nFirst Block:\t" + req-
>Format("{0:D}", Buffer[3]);     // Show First Block
                                textBoxLog->Text += "\r\nNr. of Blocks:\t" +
req->Format("{0:D}", (Buffer[4] + 1)); // Show Nr. of Blocks, 1 added
                                textBoxLog->Text += "\r\nData:\t\t";
                                for( unsigned char j = 5; j < TxLength; j++ ) //
Data Offset = 5
                                {
                                        textBoxLog->Text +=
Char::ToString(Buffer[j]);// Show Request ASCII Data
                                }
                                textBoxLog->Text += "\r\nData Length:\t" + req-
>Format("{0:D}", DataCounter) + " Bytes";     // Show Data Length in bytes
                                textBoxLog->Text += "\r\nRequest Length:\t" +
req->Format("{0:D}", Buffer[0]) + " Bytes";   // Show Request Length in bytes

                                // Enable Transmission
                                SendFlag = TRUE;    // Set Send Flag
```

```
                                    break;                    // Finish


      //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++++++++//

                              case 5: // Command 5: Read Single/Multiple Blocks
                                    cmd = "Read Single/Multiple Blocks";

                                    // Get Read Single/Multiple Blocks Arguments and
check if error occurred
                                    if( Get_RSMB_Arguments() == 1 )
                                          return;      // If error occured Exit

                                    if(ArgumentCounter != 2 ) // Write Multiple
blocks requieres Arguments
                                    {
                                          MessageBox::Show("ERROR: 2 Arguments
required, click on Help");
                                          SendFlag = FALSE;
                                          return;
                                    }// end if(ArgumentCounter != 2 )

                                    // Load OUT Data into Buffer
                                    TxLength = TX_HEADER + 1 + ArgumentsLength;   //
n BYTES to send
                                    Buffer[0] = TxLength;
             // OUTPacket.Length
                                    Buffer[1] = READ_SINGLE_MULTIPLE_BLOCKS;      //
OUTPacket.CMD
                                    Buffer[2] = Flags;
             // OUTPacket.Flags

                                    // Show Request in LOG Window
                                    textBoxLog->Text = "REQUEST:\r\n" + "\r\n";

                                    for( HEXCounter = 0; HEXCounter < TX_HEADER;
HEXCounter++)// Show Request TX_HEADER in HEX format
                                    {
                                          textBoxLog->Text += " " + req-
>Format("{0:X2}", Buffer[HEXCounter]);
                                    }// end for( HEXCounter = 0; HEXCounter <
TX_HEADER; HEXCounter++)

                                    textBoxLog->Text += "     ";

                                    for( ASCIICounter = 0; ASCIICounter < TX_HEADER;
ASCIICounter++)     // Show Request TX_HEADER in ASCII format
                                    {
                                          textBoxLog->Text += " " +
Char::ToString(Buffer[ASCIICounter]);
                                    }// end for( ASCIICounter = 0; ASCIICounter <
TX_HEADER; ASCIICounter++)

                                    while( HEXCounter && ASCIICounter < TxLength )
      // Show Arguments in HEX and ASCII format
                                    {
                                          textBoxLog->Text += "\r\n";
                                          for( unsigned char i = 0; i < 8; i++)   //
Show HEX Arguments in 8 Bytes blocks
                                          {
                                           textBoxLog->Text += " " + req-
>Format("{0:X2}", Buffer[HEXCounter]);
                                           HEXCounter++;
                                           if(HEXCounter == TxLength)
                                                 break;
```

```
                                        }// end for( unsigned char i = 0; i < 8;
i++)

                                        textBoxLog->Text += "     ";

                                        for( unsigned char i = 0; i < 8; i++)  //
Show ASCII Arguments in 8 Bytes blocks
                                        {
                                         textBoxLog->Text += " " +
Char::ToString(Buffer[ASCIICounter]);
                                         ASCIICounter++;
                                         if(ASCIICounter == TxLength)
                                                break;
                                        }// end for( unsigned char i = 0; i < 8;
i++)
                                }// end while( HEXCounter && ASCIICounter <
TxLength )

                                // Show Alphanumeric Data
                                textBoxLog->Text += "\r\n" + "\r\nCommand:\t" +
cmd;   // Show Request Command
                                textBoxLog->Text += "\r\nFlags:\t\t" + + req-
>Format("{0:D}", Buffer[2]);     // Show Request Flags
                                textBoxLog->Text += "\r\nFirst Block:\t" + req-
>Format("{0:D}", Buffer[3]);     // Show First Block
                                textBoxLog->Text += "\r\nNr. of Blocks:\t" +
req->Format("{0:D}", (Buffer[4] + 1)); // Show Nr. of Blocks, 1 added

                                // Enable Transmission
                                SendFlag = TRUE;    // Set Send Flag
                                break;                    // Finish


      //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++//

                        case 6: // Command 6: Clear Tag Data
                                cmd = "Clear Tag Data";
                                arg = "None";

                                // Load OUT Data into Buffer
                                TxLength = TX_HEADER + 4;        // n BYTES to
send
                                Buffer[0] = TxLength;                    //
OUTPacket.Length
                                Buffer[1] = CLEAR_TAG_DATA;              //
OUTPacket.CMD
                                Buffer[2] = Flags;                       //
OUTPacket.Flags
                                Buffer[3] = 0;                                  //
First Block Number
                                Buffer[4] = NrOfBlocks - 1;              // System
Number of blocks (-1, as ISO15693 Spec)
                                Buffer[5] = (BlockLength/8);     // System Block
Length

                                // Show Request in LOG Window
                                textBoxLog->Text = "REQUEST:\r\n" + "\r\n";

                                for( HEXCounter = 0; HEXCounter < TX_HEADER;
HEXCounter++)// Show Request TX_HEADER in HEX format
                                {
                                        textBoxLog->Text += " " + req-
>Format("{0:X2}", Buffer[HEXCounter]);
                                }// end for( HEXCounter = 0; HEXCounter <
TX_HEADER; HEXCounter++)
```

```
                                    textBoxLog->Text += "     ";

                                    for( ASCIICounter = 0; ASCIICounter < TX_HEADER;
ASCIICounter++)     // Show Request TX_HEADER in ASCII format, error will not
shown
                                    {
                                          textBoxLog->Text += " " +
Char::ToString(Buffer[ASCIICounter]);
                                    }// end for( ASCIICounter = 0; ASCIICounter <
TX_HEADER; ASCIICounter++)

                                    while( HEXCounter && ASCIICounter < TxLength )
      // Show Arguments in HEX and ASCII format
                                    {
                                          textBoxLog->Text += "\r\n";
                                          for( unsigned char i = 0; i < 8; i++)  //
Show HEX Arguments in 8 Bytes blocks
                                          {
                                           textBoxLog->Text += " " + req-
>Format("{0:X2}", Buffer[HEXCounter]);
                                           HEXCounter++;
                                           if(HEXCounter == TxLength)
                                                 break;
                                          }// end for( unsigned char i = 0; i < 8;
i++)

                                          textBoxLog->Text += "     ";

                                          for( unsigned char i = 0; i < 8; i++)  //
Show ASCII Arguments in 8 Bytes blocks
                                          {
                                           textBoxLog->Text += " " +
Char::ToString(Buffer[ASCIICounter]);
                                           ASCIICounter++;
                                           if(ASCIICounter == TxLength)
                                                 break;
                                          }// end for( unsigned char i = 0; i < 8;
i++)
                                    }// end while( HEXCounter && ASCIICounter <
RxLength )

                                    // Show Alphanumeric Data
                                    textBoxLog->Text += "\r\n" + "\r\nCommand:\t" +
cmd;   // Show Request Command
                                    textBoxLog->Text += "\r\nArguments:     " + arg;
      // Show Request Arguments
                                    textBoxLog->Text += "\r\nRequest Length:\t" +
req->Format("{0:D}", TxLength) + " Bytes";    // Show Request Length in bytes

                                    // Enable Transmission
                                    SendFlag = TRUE;    // Set Send Flag
                                    break;                    // Finish


      //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++//

                              case 7: // Command 7: Reset CLRC632
                                    cmd = "Reset CLRC632";
                                    arg = "None";

                                    // Load OUT DATA to Buffer
                                    TxLength = TX_HEADER;     // n BYTES to send
                                    Buffer[0] = TxLength;           // Tx is n
Bytes length
                                    Buffer[1] = RESET_CLRC632;// OUTPacket.CMD
```

```
                                    // Show Request in LOG Window
                                    textBoxLog->Text = "REQUEST:\r\n" + "\r\n";


                                    for( unsigned char i = 0; i < TxLength; i++)  //
Show Request in HEX format
                                    {
                                          textBoxLog->Text += " " + req-
>Format("{0:X2}", Buffer[i]);
                                    }// end for( unsigned char i = 0; i < TxLength;
i++)

                                    textBoxLog->Text += "     ";

                                    for( unsigned char i = 0; i < TxLength; i++)  //
Show Request in ASCII format
                                    {
                                          textBoxLog->Text += " " +
Char::ToString(Buffer[i]);
                                    }// end for( unsigned char i = 0; i < TxLength;
i++)

                                    // Show Alphanumeric Data
                                    textBoxLog->Text += "\r\n" + "\r\nCommand:\t" +
cmd;   // Show Request Command
                                    textBoxLog->Text += "\r\nArguments:     " + arg;
      // Show Request Arguments
                                    textBoxLog->Text += "\r\nRequest Length:\t" +
req->Format("{0:D}", TxLength) + " Bytes";    // Show Request Length in bytes

                                    // Enable Transmission
                                    SendFlag = TRUE;    // Set Send Flag
                                    break;                    // Finish


      //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++//

                              default:
                                    break;


      //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++/

                        }
                  }//end System::Void buttonSend_Click


private: System::Void ShowResponse(System::Object^  sender, System::EventArgs^
e)
                  {
                        //When the timer goes off on the main form, update the user
inferface with the new data obtained from the thread.

                        String ^res = "";  // Response string

                        unsigned char HEXCounter = 0; // Data HEX Counter
                        unsigned char ASCIICounter = 0; // Data ASCII Counter

                        if(ReceiveFlag == TRUE)
                        {
                              textBoxLog->Text += "\r\n" + "\r\nRESPONSE:\r\n" +
"\r\n";

                              switch(Command) // Switch by Command
```

```
                        {


      //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++//

                        case 0: // Command 0: Read Firmware Version

                              // Show Response Header in HEX format
                              for( HEXCounter = 0; HEXCounter < RX_HEADER;
HEXCounter++)
                              {
                                    textBoxLog->Text += " " + res-
>Format("{0:X2}", Buffer[HEXCounter]);
                              }// end for( HEXCounter = 0; HEXCounter <
RX_HEADER; HEXCounter++)

                              textBoxLog->Text += "     ";

                              // Show Response Header in ASCII format
                              for( ASCIICounter = 0; ASCIICounter < RX_HEADER;
ASCIICounter++)
                              {
                                    textBoxLog->Text += " " +
Char::ToString(Buffer[ASCIICounter]);
                              }// end for( ASCIICounter = 0; ASCIICounter <
RX_HEADER; ASCIICounter++)

                              if(!Error)    // If no error
                              {
                                    // Show Response Data
                                    while( HEXCounter && ASCIICounter <
RxLength )
                                    {
                                          textBoxLog->Text += "\r\n";

                                          // Show Response Data in HEX Format
                                          for( unsigned char i = 0; i < 8;
i++)
                                          {
                                           textBoxLog->Text += " " + res-
>Format("{0:X2}", Buffer[HEXCounter]);
                                           HEXCounter++;
                                           if(HEXCounter == RxLength)
                                                 break;
                                          }// end for( unsigned char i = 0; i
< 8; i++)


                                          textBoxLog->Text += "     ";

                                          // Show Response Data in ASCII
Format
                                          for( unsigned char i = 0; i < 8;
i++)
                                          {
                                           textBoxLog->Text += " " +
Char::ToString(Buffer[ASCIICounter]);
                                           ASCIICounter++;
                                           if(ASCIICounter == RxLength)
                                                 break;
                                          }// end for( unsigned char i = 0; i
< 8; i++)
                                    }// end while( HEXCounter && ASCIICounter
< RxLength )

                                    // Show Alphanumeric Data
```

```
                                        textBoxLog->Text += "\r\n" + "\r\nFirmware
Version:\t" + Char::ToString(Buffer[4]) + "." + Char::ToString(Buffer[3]);
                                        textBoxLog->Text += "\r\nResponse
Length:\t" + res->Format("{0:D}", RxLength) + " Bytes";
                                        textBoxLog->Text += "\r\n" + "Error:\t\t"
+ res->Format("{0:D}", Error);
                                  }//end if(!Error)

                                  else
                                  {
                                        // Show Alphanumeric Data
                                        textBoxLog->Text += "\r\nResponse
Length:\t" + res->Format("{0:D}", RxLength) + " Bytes";
                                        textBoxLog->Text += "\r\n" + "Error:\t\t"
+ res->Format("{0:D}", Error);
                                  }// end else

                                  // Show Error
                                  ShowError();

                                  // End Command
                                  RxLength = 0; // Clear RxLength
                                  ReceiveFlag = FALSE; // Clear Receive Flag
                                  break;


      //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++//

                            case 1: // Command 1: Write Default Configuration

                                  // Show Response Header in HEX format
                                  for( HEXCounter = 0; HEXCounter < RX_HEADER;
HEXCounter++)
                                  {
                                        textBoxLog->Text += " " + res-
>Format("{0:X2}", Buffer[HEXCounter]);
                                  }// end for( HEXCounter = 0; HEXCounter <
RX_HEADER; HEXCounter++)

                                  textBoxLog->Text += "     ";

                                  // Show Response Header in ASCII format
                                  for( ASCIICounter = 0; ASCIICounter < RX_HEADER;
ASCIICounter++)
                                  {
                                        textBoxLog->Text += " " +
Char::ToString(Buffer[ASCIICounter]);
                                  }// end for( ASCIICounter = 0; ASCIICounter <
RX_HEADER; ASCIICounter++)

                                  // Show Alphanumeric Data
                                  textBoxLog->Text += "\r\n" + "\r\nResponse
Length:\t" + res->Format("{0:D}", RxLength) + " Bytes";
                                  textBoxLog->Text += "\r\n" + "Error:\t\t" + res-
>Format("{0:D}", Error);

                                  // Show Error
                                  ShowError();

                                  // End Command
                                  RxLength = 0; // Clear RxLength
                                  ReceiveFlag = FALSE; // Clear Receive Flag
                                  break;
```

```
      //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++/

                        case 2: // Read Default Configuration

                                // Show Response Header in HEX format
                                for( HEXCounter = 0; HEXCounter < RX_HEADER;
HEXCounter++)
                                {
                                        textBoxLog->Text += " " + res-
>Format("{0:X2}", Buffer[HEXCounter]);
                                }// end for( HEXCounter = 0; HEXCounter <
RX_HEADER; HEXCounter++)

                                textBoxLog->Text += "     ";

                                // Show Response Header in ASCII format
                                for( ASCIICounter = 0; ASCIICounter < RX_HEADER;
ASCIICounter++)
                                {
                                        textBoxLog->Text += " " +
Char::ToString(Buffer[ASCIICounter]);
                                }// end for( ASCIICounter = 0; ASCIICounter <
RX_HEADER; ASCIICounter++)

                                if(!Error)   // If no error
                                {
                                        // Show Response Data
                                        while( HEXCounter && ASCIICounter <
RxLength )
                                        {
                                                textBoxLog->Text += "\r\n";

                                                // Show Response Data in HEX Format
                                                for( unsigned char i = 0; i < 8;
i++)
                                                {
                                                 textBoxLog->Text += " " + res-
>Format("{0:X2}", Buffer[HEXCounter]);
                                                 HEXCounter++;
                                                 if(HEXCounter == RxLength)
                                                        break;
                                                }// end for( unsigned char i = 0; i
< 8; i++)

                                                textBoxLog->Text += "     ";

                                                // Show Response Data in ASCII
Format
                                                for( unsigned char i = 0; i < 8;
i++)
                                                {
                                                 textBoxLog->Text += " " +
Char::ToString(Buffer[ASCIICounter]);
                                                 ASCIICounter++;
                                                 if(ASCIICounter == RxLength)
                                                        break;
                                                }// end for( unsigned char i = 0; i
< 8; i++)
                                        }// end while( HEXCounter && ASCIICounter
< RxLength )

                                        // Show Alphanumeric Data
                                        textBoxLog->Text += "\r\n" + "\r\nResponse
Length:\t" + res->Format("{0:D}", RxLength) + " Bytes";
```

```
                                        textBoxLog->Text += "\r\n" + "Error:\t\t"
+ res->Format("{0:D}", Error);
                                }// end if(!Error)
                                else
                                {
                                        // Show Alphanumeric Data
                                        textBoxLog->Text += "\r\n" + "\r\nResponse
Length:\t" + res->Format("{0:D}", RxLength) + " Bytes";
                                        textBoxLog->Text += "\r\n" + "Error:\t\t"
+ res->Format("{0:D}", Error);
                                }// end else

                                // Show Error
                                ShowError();

                                // End Command
                                RxLength = 0; // Clear RxLength
                                ReceiveFlag = FALSE; // Clear Receive Flag
                                break;


        //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++//

                        case 3: // Command 3: ISO General Command

                                // Show Response Header in HEX format
                                for( HEXCounter = 0; HEXCounter < RX_HEADER;
HEXCounter++)
                                {
                                        textBoxLog->Text += " " + res-
>Format("{0:X2}", Buffer[HEXCounter]);
                                }// end for( HEXCounter = 0; HEXCounter <
RX_HEADER; HEXCounter++)

                                textBoxLog->Text += "     ";

                                // Show Response Header in ASCII format
                                for( ASCIICounter = 0; ASCIICounter < RX_HEADER;
ASCIICounter++)
                                {
                                        textBoxLog->Text += " " +
Char::ToString(Buffer[ASCIICounter]);
                                }// end for( ASCIICounter = 0; ASCIICounter <
RX_HEADER; ASCIICounter++)

                                if(!Error)    // If no error
                                {
                                        // Show Response Data
                                        while( HEXCounter && ASCIICounter <
RxLength )
                                        {
                                                textBoxLog->Text += "\r\n";

                                                // Show Response Data in HEX Format
                                                for( unsigned char i = 0; i < 8;
i++)
                                                {
                                                 textBoxLog->Text += " " + res-
>Format("{0:X2}", Buffer[HEXCounter]);

                                                 HEXCounter++;
                                                 if(HEXCounter == RxLength)
                                                        break;
                                                }// end for( unsigned char i = 0; i
< 8; i++)

                                                textBoxLog->Text += "     ";
```

```
                                                // Show Response Data in ASCII
Format
                                                for( unsigned char i = 0; i < 8;
i++)
                                                {
                                                 textBoxLog->Text += " " +
Char::ToString(Buffer[ASCIICounter]);
                                                 ASCIICounter++;
                                                 if(ASCIICounter == RxLength)
                                                        break;
                                                }// end for( unsigned char i = 0; i
< 8; i++)
                                          }// end while( HEXCounter && ASCIICounter
< RxLength )

                                          // Show Alphanumeric Data
                                          textBoxLog->Text += "\r\n" + "\r\nResponse
Length:\t" + res->Format("{0:D}", RxLength) + " Bytes";
                                          textBoxLog->Text += "\r\n" + "Error:\t\t"
+ res->Format("{0:D}", Error);
                                    }// end if(!Error)
                                    else
                                    {
                                          // Show Alphanumeric Data
                                          textBoxLog->Text += "\r\n" + "\r\nResponse
Length:\t" + res->Format("{0:D}", RxLength) + " Bytes";
                                          textBoxLog->Text += "\r\n" + "Error:\t\t"
+ res->Format("{0:D}", Error);
                                    }// end else

                                    // Show Error
                                    ShowError();

                                    // End Command
                                    RxLength = 0; // Clear RxLength
                                    ReceiveFlag = FALSE; // Clear Receive Flag
                                    break;


      //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++//

                              case 4: // Command 4: Write Multiple Blocks

                                    // Task in Admin Mode
                                    if( UserMode == FALSE )
                                    {
                                          // Show Response Header in HEX format
                                          for( HEXCounter = 0; HEXCounter <
RX_HEADER; HEXCounter++)
                                          {
                                                textBoxLog->Text += " " + res-
>Format("{0:X2}", Buffer[HEXCounter]);
                                          }// end for( HEXCounter = 0; HEXCounter <
RX_HEADER; HEXCounter++)

                                          textBoxLog->Text += "     ";

                                          // Show Response Header in ASCII format
                                          for( ASCIICounter = 0; ASCIICounter <
RX_HEADER; ASCIICounter++)
                                          {
                                                textBoxLog->Text += " " +
Char::ToString(Buffer[ASCIICounter]);
                                          }// end for( ASCIICounter = 0;
ASCIICounter < RX_HEADER; ASCIICounter++)
```

```
                                        if(!Error)    // If no error
                                        {
                                               // Show Response Data
                                               while( HEXCounter && ASCIICounter <
RxLength )
                                               {
                                                      textBoxLog->Text += "\r\n";

                                                      // Show Response Data in HEX
Format
                                                      for( unsigned char i = 0; i
< 8; i++)
                                                      {
                                                       textBoxLog->Text += " " +
res->Format("{0:X2}", Buffer[HEXCounter]);
                                                       HEXCounter++;
                                                       if(HEXCounter == RxLength)
                                                              break;
                                                      }// end for( unsigned char i
= 0; i < 8; i++)

                                                      textBoxLog->Text += "     ";

                                                      // Show Response Data in
ASCII Format
                                                      for( unsigned char i = 0; i
< 8; i++)
                                                      {
                                                       textBoxLog->Text += " " +
Char::ToString(Buffer[ASCIICounter]);
                                                       ASCIICounter++;
                                                       if(ASCIICounter ==
RxLength)
                                                              break;
                                                      }// end for( unsigned char i
= 0; i < 8; i++)
                                               }// end while( HEXCounter &&
ASCIICounter < RxLength )

                                               // Show Alphanumeric Data
                                               textBoxLog->Text += "\r\n" +
"\r\nFlags:\t\t" + res->Format("{0:D}", Buffer[3]);
                                               textBoxLog->Text +=
"\r\nCRC16:\t\t" + Char::ToString(Buffer[5]) + Char::ToString(Buffer[4]);
                                               textBoxLog->Text += "\r\nResponse
Length:\t" + res->Format("{0:D}", RxLength) + " Bytes";
                                               textBoxLog->Text +=
"\r\nError:\t\t" + res->Format("{0:D}", Error);
                                        }// end if(!Error)

                                        else
                                        {
                                               // Show Alphanumeric Data
                                               textBoxLog->Text += "\r\n" +
"\r\nResponse Length:\t" + res->Format("{0:D}", RxLength) + " Bytes";
                                               textBoxLog->Text +=
"\r\nError:\t\t" + res->Format("{0:D}", Error);
                                        }// end else

                                        // Show Error
                                        ShowError();

                                 }// end if( UserMode == FALSE )

                                 // Task in User Mode
                                 else
```

```
                                {
                                        if( !Error ) // If No Error
                                        {
                                                // Update User List
                                                listBoxUsers->BeginUpdate();
                                                listBoxUsers->Items-
>Add(String::Format(textBoxArguments->Text)); // Add User to User List
                                                listBoxUsers->EndUpdate();

                                                // Update Users.txt
                                                StreamWriter^ sw =
File::AppendText( "Users.txt" );
                                                sw->WriteLine( textBoxArguments-
>Text );
                                                if ( sw )
                                                        delete (IDisposable^)(sw);

                                                // Show Alphanumeric Response
                                                textBoxLog->Text = "User Added:\t"
+ textBoxArguments->Text;
                                                textBoxArguments->Clear();
                                        }
                                        else
                                        {
                                                textBoxLog->Text = "Communication
Error: " + res->Format("{0:D}", Error);
                                                ShowError();
                                                textBoxLog->Text += "\r\nPlease,
call Technical Service";
                                        }
                                }

                                // End Command
                                RxLength = 0; // Clear RxLength
                                ReceiveFlag = FALSE; // Clear Receive Flag
                                break;


        //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++/

                        case 5: // Command 5: Read Multiple Blocks

                                // Task in Admin Mode
                                if( UserMode == FALSE )
                                {
                                        // Show Response Header in HEX format
                                        for( HEXCounter = 0; HEXCounter <
RX_HEADER; HEXCounter++)
                                        {
                                                textBoxLog->Text += " " + res-
>Format("{0:X2}", Buffer[HEXCounter]);
                                        }// end for( HEXCounter = 0; HEXCounter <
RX_HEADER; HEXCounter++)

                                        textBoxLog->Text += "     ";

                                        // Show Response Header in ASCII format
                                        for( ASCIICounter = 0; ASCIICounter <
RX_HEADER; ASCIICounter++)
                                        {
                                                textBoxLog->Text += " " +
Char::ToString(Buffer[ASCIICounter]);
                                        }// end for( ASCIICounter = 0;
ASCIICounter < RX_HEADER; ASCIICounter++)
```

```
                                        if(!Error)   // If no error
                                        {
                                               // Show Response Data
                                               while( HEXCounter && ASCIICounter <
RxLength )
                                               {
                                                      textBoxLog->Text += "\r\n";
                                                      for( unsigned char i = 0; i
< 8; i++)
                                                      {
                                                       textBoxLog->Text += " " +
res->Format("{0:X2}", Buffer[HEXCounter]);
                                                       HEXCounter++;
                                                       if(HEXCounter == RxLength)
                                                              break;
                                                      }// end for( unsigned char i
= 0; i < 8; i++)

                                                      textBoxLog->Text += "     ";

                                                      for( unsigned char i = 0; i
< 8; i++)
                                                      {
                                                       textBoxLog->Text += " " +
Char::ToString(Buffer[ASCIICounter]);
                                                       ASCIICounter++;
                                                       if(ASCIICounter ==
RxLength)
                                                              break;
                                                      }// end for( unsigned char i
= 0; i < 8; i++)
                                               }// end while( HEXCounter &&
ASCIICounter < RxLength )

                                               // Show Alphanumeric Data
                                               unsigned char j = 4;
                                               textBoxLog->Text += "\r\n" +
"\r\nFlags:\t\t" + res->Format("{0:D}", Buffer[3]);
                                               textBoxLog->Text +=
"\r\nData:\t\t";
                                               for( j; j < RxLength - 2; j++ )
                                               {
                                                      textBoxLog->Text +=
Char::ToString(Buffer[j]);// Show Response ASCII Data
                                               }
                                               textBoxLog->Text += "\r\nData
Length:\t" + res->Format("{0:D}", (RxLength - 6)) + " Bytes";     // Show Data
Length in bytes
                                               textBoxLog->Text +=
"\r\nCRC16:\t\t" + Char::ToString(Buffer[j+1]) + Char::ToString(Buffer[j]);
                                               textBoxLog->Text += "\r\nResponse
Length:\t" + res->Format("{0:D}", RxLength) + " Bytes";
                                               textBoxLog->Text +=
"\r\nError:\t\t" + res->Format("{0:D}", Error);
                                        }// end if(!Error)
                                        else
                                        {
                                               // Show Alphanumeric Data
                                               textBoxLog->Text += "\r\n" +
"\r\nResponse Length:\t" + res->Format("{0:D}", RxLength) + " Bytes";
                                               textBoxLog->Text +=
"\r\nError:\t\t" + res->Format("{0:D}", Error);
                                        }// end else

                                        // Show Error
                                        ShowError();
```

```
                                }//if( UserMode == FALSE )

                                // Task in User Mode
                                else
                                {
                                        if( !Error ) // If No Error
                                        {
                                                // User
                                                String ^User = "";

                                                // Show Alphanumeric Response
                                                unsigned char k = 4;
                                                textBoxLog->Text = "Access
Control:\t";

                                                for( k; k < RxLength - 2; k++ )
                                                {
                                                        textBoxLog->Text +=
Char::ToString(Buffer[k]);// Show User

                                                        User +=
Char::ToString(Buffer[k]);// Get User
                                                }


                                                // Access Rigths
                                                int index = listBoxUsers-
>FindStringExact( User ); // Find User

                                                if ( index == ListBox::NoMatches )
        // If user NOT exists
                                                        textBoxLog->Text +=
"\r\nAccess Rigths:\t" + "Access Denied";
                                                else
                                                        textBoxLog->Text +=
"\r\nAccess Rigths:\t" + "Access Conceded";

                                        }
                                        else
                                        {
                                                textBoxLog->Text = "Communication
Error: " + res->Format("{0:D}", Error);
                                                ShowError();
                                                textBoxLog->Text += "\r\nPlease,
call Technical Service";
                                        }
                                }

                                // End Command
                                RxLength = 0; // Clear RxLength
                                ReceiveFlag = FALSE; // Clear Receive Flag
                                break;


        //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++//

                        case 6: // Command 6: Clear Tag Data

                                // Show Response Header in HEX format
                                for( HEXCounter = 0; HEXCounter < RX_HEADER;
HEXCounter++)
                                {
                                        textBoxLog->Text += " " + res-
>Format("{0:X2}", Buffer[HEXCounter]);
                                }// end for( HEXCounter = 0; HEXCounter <
RX_HEADER; HEXCounter++)

                                textBoxLog->Text += "     ";

                                // Show Response Header in ASCII format
```

```
                                for( ASCIICounter = 0; ASCIICounter < RX_HEADER;
ASCIICounter++)
                                {
                                        textBoxLog->Text += " " +
Char::ToString(Buffer[ASCIICounter]);
                                }// end for( ASCIICounter = 0; ASCIICounter <
RX_HEADER; ASCIICounter++)

                                if(!Error)    // If no error
                                {
                                       // Show Response Data
                                       while( HEXCounter && ASCIICounter <
RxLength )
                                       {
                                              textBoxLog->Text += "\r\n";

                                              // Show Response Data in HEX Format
                                              for( unsigned char i = 0; i < 8;
i++)
                                              {
                                               textBoxLog->Text += " " + res-
>Format("{0:X2}", Buffer[HEXCounter]);
                                               HEXCounter++;
                                               if(HEXCounter == RxLength)
                                                      break;
                                              }// end for( unsigned char i = 0; i
< 8; i++)

                                              textBoxLog->Text += "     ";

                                              // Show Response Data in ASCII
Format
                                              for( unsigned char i = 0; i < 8;
i++)
                                              {
                                               textBoxLog->Text += " " +
Char::ToString(Buffer[ASCIICounter]);
                                               ASCIICounter++;
                                               if(ASCIICounter == RxLength)
                                                      break;
                                              }// end for( unsigned char i = 0; i
< 8; i++)
                                       }// end while( HEXCounter && ASCIICounter
< RxLength )

                                       // Show Alphanumeric Data
                                       textBoxLog->Text += "\r\n" +
"\r\nFlags:\t\t" + res->Format("{0:D}", Buffer[3]);
                                       textBoxLog->Text += "\r\nCRC16:\t\t" +
Char::ToString(Buffer[5]) + Char::ToString(Buffer[4]);
                                       textBoxLog->Text += "\r\nResponse
Length:\t" + res->Format("{0:D}", RxLength) + " Bytes";
                                       textBoxLog->Text += "\r\nError:\t\t" +
res->Format("{0:D}", Error);
                                }// end if(!Error)

                                else
                                {
                                       // Show Alphanumeric Data
                                       textBoxLog->Text += "\r\n" + "\r\nResponse
Length:\t" + res->Format("{0:D}", RxLength) + " Bytes";
                                       textBoxLog->Text += "\r\nError:\t\t" +
res->Format("{0:D}", Error);
                                }// end else

                                       // Show Error
                                       ShowError();
```

```
                                    // End Command
                                    RxLength = 0; // Clear RxLength
                                    ReceiveFlag = FALSE; // Clear Receive Flag
                                    break;


      //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++//


                              case 7: // Command 7: Reset CLRC632

                                    for( HEXCounter = 0; HEXCounter < RX_HEADER;
HEXCounter++)
                                    {
                                          textBoxLog->Text += " " + res-
>Format("{0:X2}", Buffer[HEXCounter]);
                                    }// end for( HEXCounter = 0; HEXCounter <
RX_HEADER; HEXCounter++)

                                    textBoxLog->Text += "     ";

                                    for( ASCIICounter = 0; ASCIICounter < RX_HEADER;
ASCIICounter++)
                                    {
                                          textBoxLog->Text += " " +
Char::ToString(Buffer[ASCIICounter]);
                                    }// end for( ASCIICounter = 0; ASCIICounter <
RX_HEADER; ASCIICounter++)

                                    textBoxLog->Text += "\r\n" + "\r\nResponse
Length:\t" + res->Format("{0:D}", RxLength) + " Bytes";
                                    textBoxLog->Text += "\r\n" + "Error:\t\t" + res-
>Format("{0:D}", Error);

                                    // Show Error
                                    ShowError();

                                    // End Command
                                    RxLength = 0; // Clear RxLength
                                    ReceiveFlag = FALSE; // Clear Receive Flag
                                    break;


      //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++//


                              case 8: // Command 8: Clear Error

                                    for( HEXCounter = 0; HEXCounter < RX_HEADER;
HEXCounter++)
                                    {
                                          textBoxLog->Text += " " + res-
>Format("{0:X2}", Buffer[HEXCounter]);
                                    }// end for( HEXCounter = 0; HEXCounter <
RX_HEADER; HEXCounter++)

                                    textBoxLog->Text += "     ";

                                    for( ASCIICounter = 0; ASCIICounter < RX_HEADER;
ASCIICounter++)
                                    {
                                          textBoxLog->Text += " " +
Char::ToString(Buffer[ASCIICounter]);
                                    }// end for( ASCIICounter = 0; ASCIICounter <
RX_HEADER; ASCIICounter++)
```

```
                                    textBoxLog->Text += "\r\n" + "\r\nResponse
Length:\t" + res->Format("{0:D}", RxLength) + " Bytes";
                                    textBoxLog->Text += "\r\n" + "Error:\t\t" + res-
>Format("{0:D}", Error);

                                    // Show Error
                                    ShowError();

                                    // End Command
                                    RxLength = 0; // Clear RxLength
                                    ReceiveFlag = FALSE; // Clear Receive Flag
                                    break;


      //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++/

                              case 255: // Command 255: Fatal Error

                                    for( HEXCounter = 0; HEXCounter < RX_HEADER;
HEXCounter++)
                                    {
                                          textBoxLog->Text += " " + res-
>Format("{0:X2}", Buffer[HEXCounter]);
                                    }// end for( HEXCounter = 0; HEXCounter <
RX_HEADER; HEXCounter++)

                                    textBoxLog->Text += "     ";

                                    for( ASCIICounter = 0; ASCIICounter < RX_HEADER;
ASCIICounter++)
                                    {
                                          textBoxLog->Text += " " +
Char::ToString(Buffer[ASCIICounter]);
                                    }// end for( ASCIICounter = 0; ASCIICounter <
RX_HEADER; ASCIICounter++)

                                    textBoxLog->Text += "\r\n" + "\r\nResponse
Length:\t" + res->Format("{0:D}", RxLength) + " Bytes";
                                    textBoxLog->Text += "\r\n" + "Fatal Error:\t" +
res->Format("{0:D}", Error);

                                    // Show Error
                                    ShowError();

                                    // End Command
                                    RxLength = 0; // Clear RxLength
                                    ReceiveFlag = FALSE; // Clear Receive Flag
                                    break;


      //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++/

                              }// end switch(Command)

                              // Write LOG Window into LogFile.txt
                              WriteToLogFile();

                        }// end if(ReceiveFlag == TRUE)
                  } //end System::Void timer1_Tick

private: System::Void UpdateFlags(System::Object^  sender, System::EventArgs^  e)
                  {
                        // Sub-Carrier Flag
```

```
                    if( checkBoxSubCarrier->Checked == true )     // Single sub-
Carrier selected
                          Flags = Flags & 0xFE;
                    else
                          Flags = Flags | 0x01;                              //
Dual-subcarrier selected

                    // Data Rate flag
                    if( checkBoxDataRate->Checked == true )       // Low data
Rate selected
                          Flags = Flags & 0xFD;
                    else
                          Flags = Flags | 0x02;                              //
High data Rate selected
              }


private: System::Void UpdateAvailableMemory(System::Object^  sender,
System::Windows::Forms::KeyPressEventArgs^  e)
              {
                    if(e->KeyChar::get() == 13) // Update when Enter is pressed
                    {
                    // Get System Memory Configuration
                    String ^SysNrOfBlocks = textBoxNrOfBlocks->Text;    // Get
Nr. of Blocks
                    String ^SysBlockLength = textBoxBlockLength->Text;  // Get
Block Length

                    // Convert managed String to Char Array
                    char* NrOfBlocksArray = (char*)
Marshal::StringToHGlobalAnsi(SysNrOfBlocks).ToPointer();
                    char* BlockLengthArray = (char*)
Marshal::StringToHGlobalAnsi(SysBlockLength).ToPointer();

                    // Update NrOfBlocks and BlockLength
                    NrOfBlocks = atoi(NrOfBlocksArray);
                    BlockLength = atoi(BlockLengthArray);

                    // Calculate MaxAvailableMemory
                    MaxAvailableMemory = NrOfBlocks * BlockLength;

                    // Free Char Array
                    Marshal::FreeHGlobal(IntPtr(NrOfBlocksArray));
                    Marshal::FreeHGlobal(IntPtr(BlockLengthArray));

                    // Show Max. Available Memory
                    textBoxAvailableMemory->Text =
Convert::ToString(MaxAvailableMemory);
                    }

                    else
                          return;
              }// end private: System::Void UpdateAvailableMemory(System::Object^
sender, System::Windows::Forms::KeyPressEventArgs^  e)


public: bool GetArguments(void)
              {
                    int i = 0;                                          // Pinter
for Text Arguments
                    unsigned char BufferPointer = 0;      // Pointer to Buffer[]
                    int ByteArgument = 0;                       // Argument
Value (Byte Type), int type to check Overflow
                    String ^StringArgument = "";          // Argument Value
(String Type)
                    String ^Arguments = textBoxArguments->Text;  // Get
Arguments in ASCII
```

```
                    int TxtBoxArgumentsLength = textBoxArguments->TextLength; //
Get Text Arguments Length from textBoxArguments

                    ArgumentCounter = 0;     // Init Argument counter
                    TxLength = 0;                  // Init Tx Length
                    ArgumentsLength = 0;     // Init Arguments Length
                    textBoxLog->Clear();     // Clear LOG Window

                    if( TxtBoxArgumentsLength == 0 )      // If no arguments
                          return(0);  // Exit

                    else
                    {
                          char* ArgumentArray = (char*)
Marshal::StringToHGlobalAnsi(Arguments).ToPointer(); // Unmanage String
^Arguments

                          // Check Arguments ID
                          if( ArgumentArray[0] != 35 && ArgumentArray[0] != 59)
      // If 1st argument does not start with required IDs "#" or ";"
                          {
                                MessageBox::Show("ERROR: Argument Type Mismatch,
IDs must be '#' or ';'"); // Error
                                Marshal::FreeHGlobal(IntPtr(ArgumentArray));  //
Free unmanaged char* ArgumentArray
                                return(1);   // Exit with error
                          }// end if( ArgumentArray[0] != 35 && ArgumentArray[0]
!= 59)

                          // Process Arguments
                          while( i < TxtBoxArgumentsLength )
                          {
                                // Process Argument ID == '#'
                                if( ArgumentArray[i] == 35 )  // If Argument ID
== '#', Argument will be a Byte
                                {
                                      StringArgument = "";      // Clear
Argument Value
                                      ArgumentCounter++;        // Count
Argument
                                      i++;                            //
Pointer to next value

                                      // Next value cannot be an ID
                                      if( ArgumentArray[i] == 35 ||
ArgumentArray[i] == 59 )
                                      {
                                            MessageBox::Show("ERROR: Argument
Type Mismatch, Check Arguments IDs"); // Error
                                            return(1);   // Exit with error

                                      }// end if( ArgumentArray[i] == 35 ||
ArgumentArray[i] > 59 )

                                      else
                                      {
                                            // Process Argument as Byte
                                            while( ArgumentArray[i] != 35 &&
ArgumentArray[i] != 59 && i < TxtBoxArgumentsLength )
                                            {
                                                  if( ArgumentArray[i] < 48 ||
ArgumentArray[i] > 57 )   // If Argument is not a Number
                                                  {

      MessageBox::Show("ERROR: Argument Type Mismatch, '#' ID must be a
number"); // Error
```

```
                                                        return(1);    // Exit
with error
                                                }// end if( ArgumentArray[i]
< 48 && ArgumentArray[i] > 57 )
                                                else
                                                {
                                                        StringArgument +=
Char::ToString(ArgumentArray[i]); // Process Argument to Byte
                                                        i++;   // Pointer to
next value
                                                }
                                        }// end while( ArgumentArray[i] !=
35 && ArgumentArray[i] != 59 && i < TxtBoxArgumentsLength )

                                        // Convert String Argument to Byte
Argument
                                        char* StringArgumentArray = (char*)
Marshal::StringToHGlobalAnsi(StringArgument).ToPointer();
                                        ByteArgument =
atoi(StringArgumentArray);// String Argument to Byte Argument

      Marshal::FreeHGlobal(IntPtr(StringArgumentArray)); // Free the unmanaged
string

                                        // Check Byte Argument Value
                                        if( ByteArgument > 255 )  // If
Byte Argument Value > 255
                                        {
                                                MessageBox::Show("ERROR:
Argument Type Overflow, '#' ID requires values from 0 to 255"); // Error
                                                return(1);    // Exit with
error
                                        }// end if( ByteArgument > 255 )
                                        else
                                        {
                                                Buffer[TX_HEADER +
BufferPointer] = ByteArgument;   // Load Argument in Buffer
                                                ArgumentsLength++;  // Count
Argument Length
                                                BufferPointer++;    //
Pointer to next Buffer position
                                        }// end else
                                }// end else
                        }// end if( ArgumentArray[i] == 35 )

                        // Process Argument ID == ';'
                        else if( ArgumentArray[i] == 59 )      // If
Argument ID == ';', Argument will be a String
                        {
                                ArgumentCounter++;  // Count Argument
                                i++;   // Pointer to next value

                                // Next value cannot be an ID
                                if( ArgumentArray[i] == 35 ||
ArgumentArray[i] == 59 )
                                {
                                        MessageBox::Show("ERROR: Argument
Type Mismatch, Check Arguments IDs"); // Error
                                        return(1);    // Exit with error

                                }// end if( ArgumentArray[i] == 35 ||
ArgumentArray[i] > 59 )

                                // Process Argument as String
                                while( ArgumentArray[i] != 35 &&
ArgumentArray[i] != 59 && i < TxtBoxArgumentsLength )
                                {
```

```
                                                Buffer[TX_HEADER + BufferPointer] =
ArgumentArray[i];   // Load Argument in Buffer
                                                i++;   // Pointer to next value
                                                ArgumentsLength++;  // Count
Argument Length
                                                BufferPointer++;    // Pointer to
next Buffer position
                                          }// end while( ArgumentArray[i] != 35 &&
ArgumentArray[i] != 59 && i < TxtBoxArgumentsLength )
                                    }// end else if( ArgumentArray[i] == 59 )

                                    else   // If ID is not "#" or ";"
                                    {
                                          MessageBox::Show("ERROR: Argument Type
Mismatch, IDs must be '#' or ';'"); // Error
                                          return(1);    // Exit with error
                                    }// end else

                              }// end while( i < TxtBoxArgumentsLength)

                              Marshal::FreeHGlobal(IntPtr(ArgumentArray)); // Free
the unmanaged string

                              // Check Data Overflow according to ISO15693 and System
Configuration
                              // Worst Case: Write Multiple Blocks [12 bytes] + Data
[MaxAvailableMemory]
                              if( ArgumentsLength > (12 + (MaxAvailableMemory/8)) )
                              {
                                    MessageBox::Show("ERROR: Available Memory
Exceeded, Check System Configuration"); // Error
                                    return(1);    // Exit with Error
                              }
                              else
                                    return(0);    // Exit without Error

                        }// end else
                  }// end public: void GetArguments(void)


public: bool Get_WSMB_Arguments(void)
                  {
                        // initializations
                        int TextPointer = 0;                          // Pointer for
Text Arguments
                        unsigned char BufferPointer = 1;      // Pointer to Buffer,
in this case value is 1 in order to allow Flags in Buffer[3]
                        int FirstBlockNr = 0;                         // Argument
Value (Byte Type), int type to check Overflow
                        String ^StringArgument = "";          // Argument Value
(String Type)
                        String ^Arguments = textBoxArguments->Text;  // Get
Arguments in ASCII
                        int TxtBoxArgumentsLength = textBoxArguments->TextLength; //
Get Text Arguments Length from textBoxArguments
                        int NrOfBlocksCounter = 0;                    // Nr. Of
Blocks Counter, Value depends on ASCII Data to be sent
                        ArgumentCounter = 0;                          // Init
Argument counter
                        TxLength = 0;                                       // Init
Tx Length
                        ArgumentsLength = 0;                          // Init
Arguments Length
                        DataCounter = 0;                              // Init Tag
data counter
                        textBoxLog->Clear();                          // Clear LOG
Window
```

```
                    // Arguments Check
                    if( TxtBoxArgumentsLength == 0 )      // If no arguments
                    {
                          MessageBox::Show("ERROR: 2 Arguments required, click on
Help");      // Error
                          return(1);                                    // Exit
with Error
                    }
                    else
                    {
                          char* ArgumentArray = (char*)
Marshal::StringToHGlobalAnsi(Arguments).ToPointer(); // Unmanage String
^Arguments

                          // Check Arguments ID
                          if( ArgumentArray[0] != 35)           // If 1st
argument does not start with "#" ID
                          {
                                MessageBox::Show("ERROR: Argument Type Mismatch,
Firts Argument must be '#' ID"); // Error
                                Marshal::FreeHGlobal(IntPtr(ArgumentArray));  //
Free unmanaged char* ArgumentArray
                                return(1);                              // Exit
with error
                          }

                          // Process Arguments
                          while( TextPointer < TxtBoxArgumentsLength )
                          {
                                // Process Argument ID == '#'
                                if( ArgumentArray[TextPointer] == 35 )  // If
Argument ID == '#', Argument will be a Byte
                                {
                                      StringArgument = "";      // Clear
Argument Value
                                      ArgumentCounter++;        // Count
Argument
                                      TextPointer++;                    //
Pointer to next value

                                      // Next value cannot be an ID
                                      if( ArgumentArray[TextPointer] == 35 ||
ArgumentArray[TextPointer] == 59 )
                                      {
                                            MessageBox::Show("ERROR: Argument
Type Mismatch, Check Arguments IDs"); // Error
                                            return(1);   // Exit with error

                                      }// end if( ArgumentArray[i] == 35 ||
ArgumentArray[i] > 59 )

                                      // Process Argument as Byte
                                      while( ArgumentArray[TextPointer] != 59 &&
TextPointer < TxtBoxArgumentsLength )
                                      {
                                            if( ArgumentArray[TextPointer] ==
35 )
                                            {
                                                  MessageBox::Show("ERROR:
Argument Type Mismatch, Second Argument must be a ';' ID"); // Error
                                                  return(1);          // Exit
with error
                                            }
                                            else if( ArgumentArray[TextPointer]
< 48 || ArgumentArray[TextPointer] > 57 )     // If Argument is not a Number
                                            {
```

```
                                                MessageBox::Show("ERROR:
Argument Type Mismatch, '#' ID must be a number"); // Error
                                                return(1);          // Exit
with error
                                        }
                                        else
                                        {
                                                StringArgument +=
Char::ToString(ArgumentArray[TextPointer]); // Process Argument to Byte
                                                TextPointer++;      //
Pointer to next value
                                        }
                                    }// end while( ArgumentArray[TextPointer]
!= 59 && TextPointer < TxtBoxArgumentsLength )

                                    // Convert String Argument to Byte
Argument
                                    char* StringArgumentArray = (char*)
Marshal::StringToHGlobalAnsi(StringArgument).ToPointer();
                                    FirstBlockNr = atoi(StringArgumentArray);
                    // String Argument to Byte Argument

       Marshal::FreeHGlobal(IntPtr(StringArgumentArray));  // Free the unmanaged
string

                                    // Check Byte Argument Value Overflow
                                    if( FirstBlockNr > 255 )  // If Byte
Argument Value > 255
                                    {
                                        MessageBox::Show("ERROR: Argument
Type Overflow, '#' ID requires values from 0 to 255"); // Error
                                        return(1);              // Exit
with error
                                    }
                                    else
                                    {
                                        Buffer[TX_HEADER + BufferPointer] =
FirstBlockNr; // Load Argument in Buffer
                                        ArgumentsLength++;  // Count
Argument Length
                                        BufferPointer++;    // Pointer to
next Buffer position
                                    }
                                }// end if( ArgumentArray[TextPointer] == 35 )

                                // Process ASCII Data, Argument ID == ';'
                                else if( ArgumentArray[TextPointer] == 59 )   //
If Argument ID == ';', Argument will be a String
                                {
                                    ArgumentCounter++;        // Count
Argument
                                    TextPointer++;                  //
Pointer to next value

                                    // Process WSMB ASCII Data  to adapt it
into System Blocks
                                    NrOfBlocksCounter =
((TxtBoxArgumentsLength - 1 - TextPointer)/(BlockLength / NrOfBlocks)); // Get
WSMB Data Length, According to ISO Spec
                                    Buffer[TX_HEADER + BufferPointer] =
NrOfBlocksCounter; // Load Nr. of Blocks in Buffer
                                    ArgumentsLength++;        // Count
Argument Length
                                    BufferPointer++;          // Pointer to
next Buffer position
```

```
                                        while( TextPointer <
TxtBoxArgumentsLength)
                                        {
                                                if( ArgumentArray[TextPointer] ==
35 || ArgumentArray[TextPointer] == 59 )
                                                {
                                                        MessageBox::Show("ERROR:
Only 2 Arguments are required!"); // Error
                                                        return(1);          // Exit
with error
                                                }
                                                else
                                                {
                                                        Buffer[TX_HEADER +
BufferPointer] = ArgumentArray[TextPointer];  // Load Argument in Buffer
                                                        TextPointer++;             //
Pointer to next value
                                                        ArgumentsLength++;  // Count
Argument Length
                                                        BufferPointer++;    //
Pointer to next Buffer position
                                                        DataCounter++;             //
Count Data byte
                                                }
                                        }// end while( TextPointer <
TxtBoxArgumentsLength)
                                }// else if( ArgumentArray[TextPointer] == 59 )

                                // Process another kind of Argument
                                else   // If ID is not "#" or ";"
                                {
                                        MessageBox::Show("ERROR: Argument Type
Mismatch, values ID must be '#' or ';'"); // Error
                                        return(1);   // Exit with error
                                }// end else

                        }// end while( i < TxtBoxArgumentsLength)

                        Marshal::FreeHGlobal(IntPtr(ArgumentArray)); // Free
the unmanaged tring

                        // Check Data Overflow according to System
Configuration
                        if( FirstBlockNr > NrOfBlocks - 1)
                        {
                                MessageBox::Show("ERROR: Nr. Of Blocks Excedded,
Check System Configuration"); // Error
                                return(1);   // Exit with Error
                        }
                        else if( DataCounter > (NrOfBlocks -
FirstBlockNr)*(BlockLength/NrOfBlocks))
                        {
                                MessageBox::Show("ERROR: Available Memory
Excedded, Check System Configuration"); // Error
                                return(1);   // Exit with Error
                        }
                        else
                                return(0);   // Exit

                }// end else
        }// end public: bool Get_WSMB_Arguments(void)


public: bool Get_RSMB_Arguments(void)
        {
                // initializations
```

```
                    int TextPointer = 0;                          // Pointer for
Text Arguments
                    unsigned char BufferPointer = 1;      // Pointer to Buffer,
in this case value is 1 in order to allow Flags in Buffer[3]
                    int FirstBlockNr = 0;                         // Argument
Value (Byte Type), int type to check Overflow
                    String ^StringArgument = "";          // Argument Value
(String Type)
                    String ^Arguments = textBoxArguments->Text;  // Get
Arguments in ASCII
                    int TxtBoxArgumentsLength = textBoxArguments->TextLength; //
Get Text Arguments Length from textBoxArguments
                    int NrOfBlocksCounter = 0;                    // Nr. Of
Blocks Counter, Value depends on ASCII Data to be sent
                    ArgumentCounter = 0;                          // Init
Argument counter
                    TxLength = 0;                                       // Init
Tx Length
                    ArgumentsLength = 0;                          // Init
Arguments Length
                    textBoxLog->Clear();                          // Clear LOG
Window

                    // Arguments Check
                    if( TxtBoxArgumentsLength == 0 )      // If no arguments
                    {
                          MessageBox::Show("ERROR: 2 Arguments required, click on
Help");      // Error
                          return(1);                                    // Exit
with Error
                    }

                    else
                    {
                          char* ArgumentArray = (char*)
Marshal::StringToHGlobalAnsi(Arguments).ToPointer(); // Unmanage String
^Arguments

                          // Check Arguments ID
                          if( ArgumentArray[0] != 35)      // If 1st argument
does not start with "#" ID
                          {
                                MessageBox::Show("ERROR: Argument Type Mismatch,
Firts Argument must be a '#' ID");      // Error
                                Marshal::FreeHGlobal(IntPtr(ArgumentArray));  //
Free unmanaged char* ArgumentArray
                                return(1);   // Exit with error
                          }// end if( ArgumentArray[0] != 35 )

                          // Process Arguments
                          while( TextPointer < TxtBoxArgumentsLength )
                          {
                                // Process Argument ID == '#'
                                if( ArgumentArray[TextPointer] == 35 )  // If
Argument ID == '#', Argument will be a Byte
                                {
                                      StringArgument = "";      // Clear
Argument Value
                                      ArgumentCounter++;        // Count
Argument
                                      TextPointer++;
      // Pointer to next value

                                      // Next value cannot be an ID
                                      if( ArgumentArray[TextPointer] == 35 ||
ArgumentArray[TextPointer] == 59 )
                                      {
```

```
                                                MessageBox::Show("ERROR: Argument
Type Mismatch, Check Arguments IDs"); // Error
                                                return(1);   // Exit with error

                                          }// end if( ArgumentArray[i] == 35 ||
ArgumentArray[i] > 59 )

                                          // Process Argument as Byte
                                          //while( ArgumentArray[i] != 35 &&
ArgumentArray[i] != 59 && i < TxtBoxArgumentsLength )
                                          while( ArgumentArray[TextPointer] != 35 &&
TextPointer < TxtBoxArgumentsLength )
                                          {
                                                if( ArgumentArray[TextPointer] ==
59 )
                                                {
                                                      if(ArgumentCounter == 2)

      MessageBox::Show("ERROR: Only 2 Arguments are required!"); // Error
                                                      else

      MessageBox::Show("ERROR: Argument Type Mismatch, Second Argument must be a
'#' ID"); // Error
                                                      return(1);   // Exit with
error
                                                }

                                                else if( ArgumentArray[TextPointer]
< 48 || ArgumentArray[TextPointer] > 57 )     // If Argument is not a Number
                                                {
                                                      MessageBox::Show("ERROR:
Argument Type Mismatch, '#' ID must be a number"); // Error
                                                      return(1);   // Exit with
error
                                                }// end if( ArgumentArray[i] < 48
&& ArgumentArray[i] > 57 )

                                                else
                                                {
                                                      StringArgument +=
Char::ToString(ArgumentArray[TextPointer]); // Process Argument to Byte
                                                      TextPointer++;      //
Pointer to next value
                                                }
                                          }// end while( ArgumentArray[i] != 35 &&
ArgumentArray[i] != 59 && i < TxtBoxArgumentsLength )

                                          // Convert String Argument to Byte
Argument
                                          char* StringArgumentArray = (char*)
Marshal::StringToHGlobalAnsi(StringArgument).ToPointer();
                                          FirstBlockNr = atoi(StringArgumentArray);
      // String Argument to Byte Argument

      Marshal::FreeHGlobal(IntPtr(StringArgumentArray)); // Free the unmanaged
string

                                          // Check Byte Argument Value
                                          if( FirstBlockNr > 255 )  // If Byte
Argument Value > 255
                                          {
                                                MessageBox::Show("ERROR: Argument
Type Overflow, '#' ID requires values from 0 to 255"); // Error
                                                return(1);   // Exit with error
                                          }// end if( FirstBlockNr > 255 )
                                          else
                                          {
```

```
                                                Buffer[TX_HEADER + BufferPointer] =
FirstBlockNr;// Load Argument in Buffer
                                                ArgumentsLength++;  // Count
Argument Length
                                                BufferPointer++;    // Pointer to
next Buffer position
                                        }// end else
                                    }// end if( ArgumentArray[i] == 35 )

                                    else   // If ID is not "#" or ";"
                                    {
                                        MessageBox::Show("ERROR: Argument Type
Mismatch, IDs must be '#' or ';'"); // Error
                                        return(1);    // Exit with error
                                    }// end else

                            }// end while( i < TxtBoxArgumentsLength)

                            // Check Data Overflow according to System
Configuration
                            if( FirstBlockNr > NrOfBlocks - 1)
                            {
                                MessageBox::Show("ERROR: Nr. Of Blocks Excedded,
Check System Configuration"); // Error
                                return(1);    // Exit with Error
                            }

                            Marshal::FreeHGlobal(IntPtr(ArgumentArray)); // Free
the unmanaged string
                            return(0);    // Exit

                        }// end else
                }// end public: void GetArguments(void)


private: System::Void ClearError(System::Object^  sender, System::EventArgs^  e)
                {
                    if( textBoxStatus->Text == "Status: Device Not Detected,
Verify Connection/Correct Firmware" ) // If Device is not connected
                    {
                        MessageBox::Show("Device Not Detected, Verify
Connection/Correct Firmware");   // Show Error
                        return;      // Exit
                    }

                    String ^req = "";                       // Request String
                    String ^cmd = "Clear Error";     // Service Command
                    String ^arg = "None";                   // no Arguments
requiered
                    unsigned char HEXCounter = 0;    // Counter for HEX Data
                    unsigned char ASCIICounter = 0; // Counter for ASCII Data

                    // Load OUT DATA to Buffer
                    TxLength = TX_HEADER;      // n BYTES to send
                    Buffer[0] = TxLength;      // Tx is n Bytes length
                    Buffer[1] = CLEAR_ERROR;

                    // Show Request in LOG Window
                    textBoxLog->Text = "REQUEST:\r\n" + "\r\n";


                    for( unsigned char i = 0; i < TxLength; i++)  // Show Request
in HEX format
                        {
                            textBoxLog->Text += " " + req->Format("{0:X2}",
Buffer[i]);
                        }// end for( unsigned char i = 0; i < TxLength; i++)
```

```
                        textBoxLog->Text += "     ";

                        for( unsigned char i = 0; i < TxLength; i++)  // Show
Request in ASCII format
                        {
                              textBoxLog->Text += " " +
Char::ToString(Buffer[i]);
                        }// end for( unsigned char i = 0; i < TxLength; i++)

                        textBoxLog->Text += "\r\n" + "\r\nCommand:\t" + cmd; //
Show Request Command
                        textBoxLog->Text += "\r\nArguments:     " + arg;     //
Show Request Arguments
                        textBoxLog->Text += "\r\nRequest Length:\t" + req-
>Format("{0:D}", TxLength) + " Bytes"; // Show Request Length in bytes

                        // Enable Transmission
                        SendFlag = TRUE;    // Set Send Flag
             }

public: void WriteToLogFile(void)
            {
                  String ^LogFile = "LogFile.txt";
                  String ^SOF_LOGHeader = "
*****************************************/\r\n";
                  String ^EOF_LOGHeader =
"\r\n/*****************************************  ";

                  StreamWriter^ sw = File::AppendText( LogFile );

                  sw->WriteLine( DateTime::Now + SOF_LOGHeader );
                  sw->WriteLine( textBoxLog->Text );
                  sw->WriteLine( EOF_LOGHeader + DateTime::Now );

                  if ( sw )
                 delete (IDisposable^)(sw);

            }// end public: void WriteToLogFile(void)


public: void SaveLogFile(void)
            {
                  // Process FileName.txt
                  DateTime dt = DateTime::Now;
                  String ^Date = dt.ToString( "dd-MM-yy",
DateTimeFormatInfo::InvariantInfo );
                  String ^Time = Time->Format("{0:D2}", dt.Hour) + "h " + Time-
>Format("{0:D2}", dt.Minute) + "m " + Time->Format("{0:D2}", dt.Second) + "s";
                  String ^FileDateTime = Date + "_" + Time + ".txt";

                  if( File::Exists(FileDateTime))  // If FileName.txt exists
                        MessageBox::Show("ERROR: File already exists"); //
Error
                  else
                  {
                        File::Copy( "LogFile.txt", FileDateTime );    // Copy
LogFile.txt to FileName.txt
                        MessageBox::Show("LOG File Saved"); // Error
                  }
            }


private: System::Void SaveLog(System::Object^  sender, System::EventArgs^  e)
            {
                  SaveLogFile();      // Run Save Log File
```

```
            }// end private: System::Void SaveLogFile(System::Object^  sender,
System::EventArgs^  e)


private: System::Void Exit(System::Object^  sender, System::EventArgs^  e)
            {
                  // In Admin Mode
                  if( UserMode == FALSE )
                  {
                        File::Delete("LogFile.txt");     // Delete LogFile.txt
                        Close();                                    // Exit
Application
                  }

                  // In User Mode
                  else
                  {
                        SaveLogFile();                              // Always
Save Log File
                        File::Delete("LogFile.txt");     // Delete LogFile.txt
                        Close();                                    // Exit
Application
                  }

            }// end private: System::Void Exit(System::Object^  sender,
System::EventArgs^  e)


private: System::Void ChangeMode(System::Object^  sender, System::EventArgs^  e)
            {
                  // Admin Mode selected
                  if ( buttonChangeMode->Text == "Admin Mode" )
                  {
                        UserMode = FALSE;
                        groupBoxServices->Text = "Services List";
                        listBoxServices->Visible = true;
                        listBoxUsers->Visible = false;
                        groupBoxSystemConfiguration->Enabled = true;
                        labelArguments->Text = "Arguments:";
                        textBoxArguments->Clear();
                        textBoxArguments->Location =
System::Drawing::Point(255, 10);
                        textBoxArguments->Size = System::Drawing::Size(248,
20);
                        textBoxLog->Clear();
                        textBoxLog->BackColor =
System::Drawing::SystemColors::ActiveCaptionText;
                        buttonAddUser->Visible = false;
                        buttonSend->Visible = true;
                        buttonRemoveUser->Visible = false;
                        buttonClearError->Visible = true;
                        buttonAccessControl->Visible = false;
                        buttonSaveLog->Visible = true;
                        buttonChangeMode->Text = "User Mode";
                  }

                  // User Mode selected
                  else if( buttonChangeMode->Text == "User Mode" )
                  {
                        UserMode = TRUE;
                        groupBoxServices->Text = "Users List:";
                        LoadUsers(); // Load Users in Users List
                        listBoxServices->Visible = false;
                        listBoxUsers->Visible = true;
                        groupBoxSystemConfiguration->Enabled = false;
                        labelArguments->Text = "User:";
```

```
                        textBoxArguments->Clear();
                        textBoxArguments->Location =
System::Drawing::Point(232, 10);
                        textBoxArguments->Size = System::Drawing::Size(271,
20);
                        textBoxLog->Clear();
                        textBoxLog->BackColor =
System::Drawing::SystemColors::ActiveCaptionText;
                        buttonSend->Visible = false;
                        buttonAddUser->Visible = true;
                        buttonClearError->Visible = false;
                        buttonRemoveUser->Visible = true;
                        buttonSaveLog->Visible = false;
                        buttonAccessControl->Visible = true;
                        buttonChangeMode->Text = "Admin Mode";
                }

        }// end private: System::Void ChangeMode(System::Object^  sender,
System::EventArgs^  e)


public: void LoadUsers(void)
        {
                String^ User;

                // Make sure Users List is clear
                listBoxUsers->Items->Clear();

                // Check Users.txt file
                if(!File::Exists( "Users.txt" )) // If Users.txt does not
exist
                {
                        StreamWriter^ sw = File::AppendText( "Users.txt" );  //
Create Users.txt
                        sw->WriteLine( "Admin" );  // Add Admin as default user
in Users.txt
                        listBoxUsers->Items->Add(String::Format("Admin"));  //
Add Admin as default user in Users List
                        if ( sw )
                  delete (IDisposable^)(sw); // Free File
                  }
                  else
                  {
                        StreamReader^ sr = gcnew StreamReader( "Users.txt" );
// Enable File Reader

                        // Load Users in User List
                        listBoxUsers->BeginUpdate();
                        while ( User = sr->ReadLine() )  // Update Users List
                        {
                              listBoxUsers->Items->Add(String::Format(User));
                        }
                        listBoxUsers->EndUpdate();
                        if ( sr )
                              delete (IDisposable^)(sr);// Free file
                }

        }// end public: void LoadUsers(void)


private: System::Void AddUser(System::Object^  sender, System::EventArgs^  e)
        {
                // Check System Running
                if( textBoxStatus->Text == "Status: Device Not Detected,
Verify Connection/Correct Firmware" ) // If Device is not connected
                {
```

```
                        MessageBox::Show("Device Not Detected, Verify
Connection/Correct Firmware");   // Show Error
                        return;      // Exit
                }

                // Check Required Data
                if(textBoxArguments->Text == "" ) // If no User to add
                {
                        MessageBox::Show( "ERROR: Insert a User" );   // error
                        return;      // Exit
                }

                // Check Data Overflow
                if(textBoxArguments->TextLength > (MaxAvailableMemory/8))
                {
                        MessageBox::Show( "ERROR: User Name is too Long" );  //
error
                        return;      // Exit
                }

                // Get User
                String ^User = textBoxArguments->Text; // Get Arguments
                array<Char>^UserArray = User->ToCharArray(); // Arguments to
Char Array

                // Check if User is alredy in User List
                int index = listBoxUsers->FindStringExact( User ); // Find
User
                if ( index != ListBox::NoMatches )     // If user exists
                {
                        MessageBox::Show( "ERROR: User already exists" ); //
Error
                        return;     // Exit
                 }

                // Process User
                for(unsigned char i = 0; i < textBoxArguments->TextLength;
i++)
                {
                        Buffer[i+5] = UserArray[i];      // take into account
Data Offset
                }

                // Process Request
                TxLength = TX_HEADER + 3 + textBoxArguments->TextLength;   //
n BYTES to send
                Buffer[0] = TxLength;     // Tx is n Bytes length
                Buffer[1] = WRITE_SINGLE_MULTIPLE_BLOCKS;     // Command is
Read Firmware Version
                Buffer[2] = 0;      // Flags
                Buffer[3] = 0;      // First Block
                Buffer[4] = ((textBoxArguments->TextLength - 1)/(BlockLength
/ NrOfBlocks));     // One block

                // Enable Transmission
                SendFlag = TRUE;    // Set Send Flag

            }// end private: System::Void AddUser(System::Object^  sender,
System::EventArgs^  e)


private: System::Void RemoveUser(System::Object^  sender, System::EventArgs^  e)
            {
                    // Select User to remove
                    int index = listBoxUsers->SelectedIndex;

                    // If selected user is Admin
```

```
                    if ( index == 0 )
                    {
                          MessageBox::Show( "ERROR: User Admin cannot be
removed" ); // Error
                          return;     // Exit
                    }

                    // If selected user is NOT Admin
                    else
                    {
                          String ^RemovedUser = listBoxUsers-
>GetItemText(listBoxUsers->SelectedItem);
                          // Update Users List
                          listBoxUsers->BeginUpdate();
                          listBoxUsers->Items->Remove(listBoxUsers-
>SelectedItem);     // Remove selected User
                          listBoxUsers->EndUpdate();

                          // Update Users.txt
                          index = listBoxUsers->Items->Count;     // Count
present Users
                          File::Delete( "Users.txt" );     // Delete User.txt
                          StreamWriter^ sw = File::AppendText( "Users.txt" );  //
Create User.txt again

                          for( unsigned char i = 0; i < index; i++ )    // Update
Users
                          {
                                listBoxUsers->SetSelected( i, true );  // Select
User from Users List
                                sw->WriteLine( listBoxUsers->SelectedItem );  //
Write User to Users.txt
                          }

                          if ( sw )
                delete (IDisposable^)(sw); // Free File

                          // Show Alphanumeric Response
                          textBoxLog->Text = "User Removed:\t" + RemovedUser;

                          // Write LOG Window into LogFile.txt
                          WriteToLogFile();
                    }

             }// end private: System::Void RemoveUser(System::Object^  sender,
System::EventArgs^  e)


private: System::Void AccessControl(System::Object^  sender, System::EventArgs^
e)
             {
                    // Check System Running
                    if( textBoxStatus->Text == "Status: Device Not Detected,
Verify Connection/Correct Firmware" ) // If Device is not connected
                    {
                          MessageBox::Show("Device Not Detected, Verify
Connection/Correct Firmware");   // Show Error
                          return;      // Exit
                    }

                    // Process Request
                    TxLength = TX_HEADER + 3; // n BYTES to send
                    Buffer[0] = TxLength;            // Tx is n Bytes length

                    Buffer[1] = READ_SINGLE_MULTIPLE_BLOCKS; // Command
                    Buffer[2] = 0;                           // Flags
```

```
                    Buffer[3] = 0;                          // Start in First
Block
                    Buffer[4] = NrOfBlocks-1; // Read all Blocks

                    // Enable Transmission
                    SendFlag = TRUE;    // Set Send Flag
             }


private: System::Void ClearArguments(System::Object^  sender, System::EventArgs^
e)
             {
                    textBoxArguments->Clear();// Clear Arguments
                    textBoxArguments->Focus();// Get Focus
             }

public: void ShowError(void)
             {
                    switch( Error ) // Switch by Error
                    {
                           case 0:
                                  textBoxLog->Text += ", None";
                                  break;

                           case 1:
                                  textBoxLog->Text += ", Start Up Phase Failed";
                                  break;

                           case 2:
                                  textBoxLog->Text += ", SPI Write Collision";
                                  break;

                           case 3:
                                  textBoxLog->Text += ", Register Address
Unavailable";
                                  break;

                           case 4:
                                  textBoxLog->Text += ", E2PROM Access not
Allowed";
                                  break;

                           case 5:
                                  textBoxLog->Text += ", Transmit Timeout";
                                  break;

                           case 6:
                                  textBoxLog->Text += ", Receive Timeout";
                                  break;

                    }// end switch( Error )
             }// end public: void ShowError(void)

private: System::Void ShowHelp(System::Object^  sender, System::EventArgs^  e)
             {
                    // Check if File Help Exists
                    if( !File::Exists("Help.txt") )
                           MessageBox::Show("ERROR: Cannot find Help.txt");
                    else
                    {
                           StreamReader^ sr = gcnew StreamReader( "Help.txt" );
                           String^ HelpLine;
                           textBoxLog->Clear();
                           textBoxLog->BackColor =
System::Drawing::SystemColors::ControlLight;
                           while ( HelpLine = sr->ReadLine() )
```

```
                        {
                                textBoxLog->Text += HelpLine + "\r\n";
                        }
                }
        }

/**************************************************************************
************************ RFidReader v1.0 - Óscar Aragón Andreu */

};
}
```