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

El PC como generador de funciones usando la tarjeta PCI-DAS6025

TITULACIÓN: Ingeniería Técnica Industrial en Electrónica Industrial

AUTOR: Héctor Zazo Jiménez

DIRECTOR: Esteban del Castillo Pérez

FECHA:Septiembre del 2009

Índice

1.- Introducción

Este proyecto desarrolla una aplicación con la que podremos generar diferentes tipos de formas de onda. Se puede decir que se convertirá el ordenador en un generador de funciones.

Un generador de señal es todo aquello que permita generar una señal estable y calibrada. Su aplicación habitual es en el campo de la instrumentación y el test, o para el funcionamiento de dispositivos que convierten un tipo de señal eléctrica en otra.

Un generador de señal tiene cuatro elementos básicos necesarios para su funcionamiento: la frecuencia, la amplitud, el offset y el tipo de onda.

Por el rango de frecuencias los generadores se pueden clasificar en generadores de frecuencia baja, generadores de radiofrecuencia (RF), desde 1kHz a 1GHz, y los generadores de señales de microondas, desde 1GHz a 45GHz.

Otra manera de clasificar los generadores sería mediante el tipo de onda que produce. Por ese motivo se puede hablar de generador de funciones (onda cuadrada, triangular, sinusoidal, etc.), generador de señales (sinusoide con modulación), osciladores (sinusoide con frecuencia y/o amplitud fija), generadores de barrido (modulación de frecuencia lenta y cíclica), generadores de tren de pulsos (pulsos y ondas cuadradas), generadores de ondas arbitrarias, entre otras utilidades.

El tipo más común de generadores de señal es el llamado generador de funciones, el cual recibe este nombre ya que produce señales que se pueden expresar con una función matemática. Las formas de señal obtenidas son: la triangular, empleada para medidas de nivel de disparo, estudios de linealidad, etc.; la cuadrada, empleada para el análisis de la respuesta transitoria, entre otras utilidades, y la sinusoidal, usada para la obtención de la respuesta temporal. Otro tipo de ondas derivadas de estas son el diente de sierra, la onda trapezoidal, la onda rectangular y la onda cosinusoidal.

El generador realizado en este proyecto será una mezcla de generador de funciones, que permita generar ondas cuadradas, triangulares, sinusoidales, trapezoidales; y un generador de ondas arbitrarias, es decir, una onda que es un tren de pulsos de valores de amplitud aleatorios.

Este proyecto se realiza para la tarjeta PCI DAS6025 de Measurament Computing®.

2.- Objetivo

El objetivo de este proyecto es realizar un generador de funciones de baja frecuencia mediante un PC y la tarjeta PCI DAS6025 de Measurament Computing®. Para ello se generará una aplicación que permita que la tarjeta PCI DAS6025 funcione como tal de manera fácil para el usuario. Por una salida de la tarjeta se obtendrá la señal deseada que podrá ser comprobada con un osciloscopio. La señal generada estará disponible en un pin de salida de la tarjeta.

Otro objetivo fundamental es realizar el objetivo mencionado anteriormente en una aplicación de fácil manejo, atractiva visualmente y cómoda. Debe de tener la aplicación una fácil instalación y desinstalación, así como una fácil accesibilidad a los diferentes tipos de onda de manera rápida.

3.- Especificaciones Previas

Para la realización de este proyecto se marcan las siguientes especificaciones previas:

- El software debe desarrollarse en lenguaje de programación C++.
- La interfaz de usuario debe de ser intuitiva y de fácil asimilación.
- Se deben utilizar librerías públicas de código abierto.
- En caso de que sea necesario aportar un software externo, este debe ser de código abierto y libre distribución (“freeware”).
- Para la realización del generador debe utilizarse la tarjeta PCI DAS6025.

El código para este proyecto es el C++ que es un lenguaje universal para cualquier máquina, de fácil interpretación y capaz de realizar desde programación a muy alto nivel a programación de bajo nivel (bits). Además la programación C++ está orientada a objetos con lo que se pueden realizar interfaz de usuarios de manera sencilla. Para realizar una interfaz de usuario más intuitiva se utilizará la plataforma de Microsoft Framework .NET, plataforma *freeware*, que se usará para realizar una interfaz más atractiva y sencill. Para su funcionamiento será necesario ordenadores con Windows y dicha plataforma (incluida en el instalador de la aplicación de este proyecto).

El generador de funciones creado en este proyecto es para la tarjeta PCI DAS6025 de Measurament Computing®. Es necesaria la instalación de dicha tarjeta en la ranura PCI y la comunicación con un cable C100HD50-x. El software creado para el generador es para el funcionamiento correcto de la tarjeta PCI DAS6025, el uso de otras tarjetass debe de comportar que esta tarjeta y la tarjeta PCI DAS6025 contengan las mismas especificaciones técnicas o sino sería necesario una pequeña modificación del código como: cambiar el rango de frecuencia o cambiar el rango de voltaje.Igualmente, el programa está configurado para que la tarjeta esté instalada en la posición 0 y se usará el convertidor D/A número 0, si se desea usar otra posición de tarjeta será necesario la modificación del código.

Otra especificación es que el proyecto debe de realizarse y funcionar en el ordenador del laboratorio 113 de la Universitat Rovira i Virgili ya que es donde disponemos de la tarjeta PCI-DAS6025 y mostrar mediante un osciloscopio convencional que el resultado del proyecto es el correcto.

Figura 3.1: Imágenes del laboratorio 113 de la Universitat Rovira i Virgili.

4.- Antecedentes

4.1.- Generadores de funciones

La mayoría de los generadores de funciones estan creados para el fin de testear y alimentar tarjetas electrónicas, y por ello la evolución de los generadores ha crecido rápidamente. Inicialmente los generadores de funciones eran completamente analógicos con la posibilidad de sólo generar tres tipos de onda: cuadrada, sinusoidal y triangular. El gran inconveniente de estos generadores era que tenían muy bajo rango de frecuencia (de 1 Hz a 1 MHz) y, sobre todo, la inexactitud al configurar la amplitud y la frecuencia, ya que, al ser un sistema analógico, la poca precisión del usuario al utilizar el generador mediante los selectores de rueda impedía controlar la exactitud de las características de la señal.

Figura 4.1: Generador de funciones analógico GF1000 de ProMax.

Los generadores de funciones analógicos han ido evolucionando, mejorando la precisión y aumentando el rango de frecuencias y nuevas utilidades como invertir la onda inicial. Al no disponer de un indicador en el generador de funciones se desconoce el valor exacto de nuestra salida y si el generador estaba bien calibrado. Por esa razón empezaron a salir los generadores de funciones digitales que permitían visualizar el valor de tensión o frecuencia mediante un *display*; así solventaban el problema de dudar de la salida del generador y tener mucha más exactitud, ya que el valor estaba reflejado por pantalla y se podía precisar a la décima, centésima y, más adelante, a la milésima parte.

Figura 4.2: Generador de funciones digital 4017B de BK Precision.

Pero las aplicaciones de los generadores de funciones han aumentado y cada vez es más necesario generar formas de onda diferentes o modificar parámetros de este tipo de ondas (modificar el tiempo de subida de uno onda triangular, etc.). Así que salieron nuevos generadores de funciones programables mediante programas informáticos como MatLab o LabView que se encontraban en un ordenador conectado al generador de funciones mediante el puerto paralelo, USB o buses específicos de instrumentación. Este sistema permite generar nuevos tipos de onda como el diente de sierra, la onda trapezoidal, etc. y, además, modificar parámetros de los tipo de ondas como tiempos, ciclos de trabajo, offset, etc. a una frecuencia muy elevada (llegando a los GHz).

Actualmente se están utilizando generadores de funciones digitales si se necesitan tipos de ondas conocidas (sinusoidal, cuadrada, triangular) con los parámetros habituales (mismo tiempo de alto que de bajo, mismo tiempo de subida que de bajada…) y generadores de funciones programables por ordenador para otras aplicaciones. Coexisten los dos tipos de generadores ya que el generador de funciones programado tiene 2 grandes inconvenientes: hay que saber programarlo y el precio es elevado (se puede hablar de 2100€ sin software). En este proyecto se pretende crear un software de uso sencillo que podrá armar funciones cuadradas, triangulares, sinusoidales, de amplitud constante, trapezoidales e incluso dibujar una onda propia, todos con parámetros variables. Esta onda saldrá por el PC mediante una tarjeta PCI que será mucho más barata, pero, por contra, con un rango de frecuencias muy bajo. A continuación se mencionan los diferentes tipos de tarjetas PCI que existen en el mercado.

Hay que mencionar que la calidad de los generadores de funciones analógicos es mejor que los generadores de funciones digitales ya que estos últimos hacen un muestreo de la onda analógica y hacen una aproximación entre valores. Si la cantidad de puntos muestreadas es muy pequeña la salida será una onda con mucho error. Si la cantidad de valores es mayor el error se irá reduciendo pero nunca llegará a la calidad del generador analógico.

Figura 4.3: Errores al digitalizar una señal

4.2.- Tarjetas PCI

Hay un total de 73 tarjetas PCI de la compañía *Measurament Computing*. *Measurament Computing* es una de las principales empresas que nos distribuye hardware específico para computadores como son las tarjetas PCI. Buscar la tarjeta apropiada para este proyecto en una cantidad tan elevada es bastante complicado. Pero si se utiliza un filtro con los parámetros necesarios para este proyecto, el número baja considerablemente. La tarjeta que se necesita debe tener como mínimo un convertidor D/A para que pueda funcionar el proyecto. Con ello el número de tarjetas PCI que cumple esta condición se reduce a 27.

A continuación, se presenta una tabla con las características de estas tarjetas. Se centrará la atención en las características de salida analógica, ya que son las necesarias para el programa porque sólo son imprescindibles los convertidores D/A. Se han eliminado de todas las tarjetas PCI que nos ofrece *Measurament Computing* todas las tarjetas que no son de la familia PCI-DAC, PCI-DAS y PCI-DDA, ya que éstas carecen de buenas características para la conversión D/A. Aquellas tarjetas que no permitan jugar con la frecuencia también serán eliminadas de la lista, ya que se busca un generador de funciones de frecuencia variable y que el usuario pueda escoger dentro de un rango. Finalmente, las veintidós tarjetas que cumplen con estos requisitos son:

NOMBRE	RESOLUCIÓN	Nº CANALES	FRECUENCIA	RANGO	PRECIO
PCI-DAC6072	16 bits	8	1111 Hz	-10V a 10V	659$
PCI-DAC6073	16 bits	16	1111 Hz	-10V a 10V	799$
PCI-DAS1612	12 bits	2	250 KHz	4 rangos distintos	749$
PCI-DAS1616	16 bits	2	100 KHz	4 rangos distintos	959$
PCI-DAS4020	12 bits	2	500 Hz	4 rangos distintos	1299$
PCI-DAS6014	16 bits	2	10KHz	-10V a 10V	459$
PCI-DAS6025	12 bits	2	10KHz	-10V a 10V	519$
PCI-DAS6030	16 bits	2	100KHz	2 rangos distintos	1185$
PCI-DAS6031	16 bits	2	100KHz	2 rangos distintos	1225$
PCI-DAS6035	12 bits	2	10KHz	-10V a 10V	719$
PCI-DAS6036	16 bits	2	10KHz	-10V a 10V	679$
PCI-DAS6040	12 bits	2	1MHz	2 rangos distintos	719$
PCI-DAS6052	16 bits	2	333KHz	2 rangos distintos	1150$
PCI-DAS6070	12 bits	2	1MHz	2 rangos distintos	1050$
PCI-DAS6071	12 bits	2	1MHz	2 rangos distintos	1195$

PCI-DAS6402	16 bits	2	100KHz	4 rangos distintos	1199$
PCI-DDA2/12	12 bits	2	10KHz	6 rangos distintos	449$
PCI-DDA2/16	16 bits	2	10KHz	6 rangos distintos	699$
PCI-DDA4/12	12 bits	4	10KHz	6 rangos distintos	699$
PCI-DDA4/16	16 bits	4	10KHz	6 rangos distintos	899$
PCI-DDA8/12	12 bits	8	10KHz	6 rangos distintos	999$
PCI-DDA8/16	16 bits	8	10KHz	6 rangos distintos	1339$

Figura 4.4: Tabla de diferentes PCI de *Measurament Computing*

Aunque se dispone de numerosas tarjetas PCI, se pueden dividir en tres familias que son las tres familias de tarjetas PCI donde los convertidores D/A presentan unas características óptimas para este proyecto. Estas tres familias son PCI-DAC, PCI-DAS y PIC-DDA.

La primera, la PCI-DAC, son tarjetas con muchos canales pero con una frecuencia muy baja; además de un precio bastante elevado, por lo que estas tarjetas han sido descartadas.

La familia DDA son tarjetas económicas y con buenas características de salida analógica, pero estas tarjetas no cumplen ninguna otra función. Si se quiere una tarjeta que pueda servir para otras aplicaciones, estas tarjetas no son las idóneas.

La única familia de tarjetas que queda por analizar son las PCI-DAS. El mercado cuenta con una gran variedad, pero para este proyecto se ha decidido utilizar una tarjeta barata porque, para comprar una tarjeta cara, hubiese sido preferible la idea de comprar un generador de funciones programable. Las 2 tarjetas que tienen mejor características son la PCI DAS6014 y la PCI DAS6025.

En ambas tarjetas las características D/A son muy parecidas, incluso la PCI DAS6014 tiene mejor resolución y es más económica; eso es debido a que tiene características peores en lo que respecta a la entrada analógica. La PCI DAS6014 es la tarjeta perfecta para esta aplicación, pero, no obstante, se trabajará con la tarjeta PCI DAS6025 que, aunque tenga peores características de salida analógica y sea más cara, puede tener más utilidades porque tiene mejores parámetros de contadores, timers, E/S digitales, entrada analógica, etc.

Nuestra tarjeta es posible conectarse con otras tarjetas PCI mencionadas con anterioridad como la tarjeta PCI DAS6030, PCI DAS6031, PCI DAS6040, PCI DAS6070, PCI DAS6071 o PCI DAS6402.

NOMBRE	RESOLUCIÓN	Nº CANALES	FRECUENCIA	RANGO	PRECIO
PCI-DAS6025	12 bits	2	10KHz	-10V a 10V	519$
PCI-DAS6030	16 bits	2	100KHz	2 rangos distintos	1185$
PCI-DAS6031	16 bits	2	100KHz	2 rangos distintos	1225$
PCI-DAS6040	12 bits	2	1MHz	2 rangos distintos	719$
PCI-DAS6070	12 bits	2	1MHz	2 rangos distintos	1050$
PCI-DAS6071	12 bits	2	1MHz	2 rangos distintos	1195$
PCI-DAS6402	16 bits	2	100KHz	4 rangos distintos	1199$

Figura 4.5: Tabla de tarjetas compatibles con la PCI DAS6025

Aquí tenemos diferentes placas disponibles para la conexión y ampliación de nuestra tarjeta PCI. Cada una de ellas tiene unos aspectos mejores que otros, por ejemplo, la tarjeta PCI-DAS6031 dispone de unas características de entrada analógica mejor que otras tarjetas o la tarjeta PCI-DAS6402 trabaja con mejores resoluciones que las demás. La mejor para nuestra aplicación es la tarjeta PCI-DAS6040 ya que dispone de una frecuencia de conversión D/A muy elevada y es más barata que las demás tarjetas mencionadas.

Todas las placas listadas son de precio más elevado y especificaciones mejores que la tarjeta utilizada PCI DAS6025, así que si se desea podemos adquirir una de ellas para mejorar las utilidades y especificaciones.

5.- Análisis Previo

Antes de empezar a “crear” el Generador de funciones, se deben conocer qué características gustaría que tuviese dicho generador de funciones. Desde siempre los generadores de funciones han tenido básicamente 3 tipos de ondas: cuadrada, triangular y sinusoidal. El generador de funciones que se creará en este proyecto también debe de tener esas formas de ondas, pero además se les podrá aplicar un offset, cambiar el ciclo de trabajo (en el caso de la onda cuadrada), cambiar el tiempo de subida y bajada de la onda (en el caso de la onda triangular) y cambiar el desfasaje (en el caso de la onda sinusoidal), además del rango de amplitud y la frecuencia como en los antiguos generadores.

Pero además de todo esto se van a crear otras formas de ondas. Una modificación de la onda triangular sería la onda diente de sierra o la onda trapezoidal que es una onda que mezcla el tiempo ON/OFF de la onda cuadrada y los tiempos de subida y bajada de la onda triangular. Esta onda da mucho juego para crear ondas que pueden ser útiles. Aunque sea una fusión de las ondas cuadrada y triangular no se eliminarán éstas, ya que su uso es muy frecuente y es interesante poderlas configurar de manera rápida.

Otra onda que presenta este generador de funciones es la señal de corriente continua. Puede que se necesite una señal constante de 5V, de -10V o cualquier otro valor dentro del rango, este generador de funciones puede crearlas. Anteriormente, se ha mencionado que los generadores de señal se utilizan en el campo de la instrumentación y el test. Hay muchos aparatos electrónicos que deben de testearse mediante diferentes voltajes dentro de un rango al azar para ver si estos cambios no provocan una alteración indebida al sistema. Para ello se ha creado un tipo de onda que produce un tren de pulsos, al rango que se quiera, de valor aleatorio y a la frecuencia que se escoja.

Finalmente, la última forma de onda es una onda que se ha denominado “Personalizada”. Con este tipo de onda podrá el usuario “dibujar” su propia onda mediante el posicionamiento de 10 puntos. Estos puntos se podrán configurar para que sean pulsos de una décima parte del periodo o unir los puntos mediante rampas progresivas. Este tipo de onda permite crear ondas al gusto del usuario y es una onda que con generadores de funciones convencionales no se podría realizar.

Nuestro generador de funciones trabajará en frecuencias bajas. Este rango de frecuencias son utilizadas en aparatos electrónicos que requiere el control del ojo humano ya que son frecuencia tan bajas que el ojo humano puede detectar el movimiento. Un ejemplo de ello son las televisiones de 60 Hz o 200 Hz donde el ojo humano puede presenciar la diferencia. Este rango de frecuencia bajo se le conoce como rango de consumo o visual. Las ventajas de este tipo de generadores es su gran utilidad en sistemas cotidianos.

Todas las partes funcionales de este proyecto se le darán al usuario al usuario en una aplicación cómoda, fácil y atractiva para que fácilmente y sin conocimientos de informática o electrónica, pueda generar las funciones mencionadas anteriormente de manera sencilla.

6.- Tarjeta PCI DAS6025

Como se ha mencionado anteriormente, hay muchasde posibilidades para la elección de la tarjeta que se utilizará, pero en esta ocasión la elegida es la PCI DAS6025 ya que así lo exigían las especificaciones previas.

6.1.- Hardware

La tarjeta PCI DAS6025 dispone de 16 canales de 12 bits de entrada analógica a una velocidad máxima de 200k transferencias por segundo, de 2 canales duales de 12 bits de salida analógica (2 convertidores D/A de 12 bits que pueden trabajar simultáneamente) a una velocidad máxima de 10000 conversiones por segundo cada canal, 32 líneas digitales de E/S y 2 contadores de tiempo de 12 bits. Los canales de entrada analógica tienen un rango de voltaje seleccionable por software entre: ± 10 V, ± 5 V, ± 500 mV, y ± 50 mV. En cambio, los canales de salida analógica tienen un rango de voltaje único que es ± 10 V. Dispone de un chip 82C54 que contiene 3 contadores de 16 bits cada uno a 20MHz. Además esta tarjeta puede generar eventos e interrupciones. Esta tarjeta, como todas las de la serie PCI DAS6000, se puede interconexionar con otras tarjetas de la misma serie (hasta un máximo de 5).

Figura 6.1: PCI DAS6025

6.1.1.- Conexionado de la tarjeta

Uno de los aspectos importantes es la conexión de la tarjeta en el PC y ésta en el mundo exterior para sacar la salida de los convertidores, que será la salida de este generador de funciones. El diagrama de este generador de funciones es el siguiente:

La tarjeta PCI DAS6025 está conectada en el ordenador, que será donde reside el software que programará la tarjeta. La tarjeta se conexionará con una tarjetas de apoyo

que servirán para tratar los diferentes *pins* y, finalmente, los *pins* que se necesitan saldrán al exterior mediante un cable y una clavija del tipo banana.

La conexión entre la tarjeta y el PC se hace mediante una conexión PCI (Peripheral Component Interconnect). La conexión PCI consiste en un bus estándar para conectar dispositivos periféricos a la tarjeta base del ordenador. Las tarjetas bases de los PCs disponen de una buena cantidad de ranuras PCI desplazando al bus ISA como estándar, ya que, a diferencia del bus ISA, el bus PCI permite la interacción con la BIOS para negociar la asignación de las IRQs de los dispositivos colgados en ese bus.

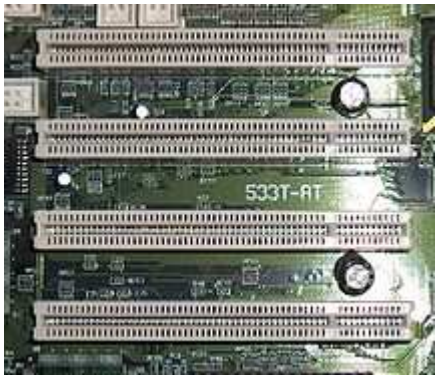


Figura 6.2: Buses PCI de una tarjeta base Pentium I

Una vez se ha conexionado la tarjeta al ordenador mediante el bus PCI, se debe saber donde están las entradas y salidas de la tarjeta PCI DAS6025. Las entradas y salidas de la tarjeta están en una conexión de 100 *pins* ubicada en la parte exterior de la misma. Así que es necesaria alguna manera de separar esos 100 *pins* para dedicar la atención a los *pins* que se necesitan. Como una separación física dentro de la tarjeta es imposible, se conectará un cable C100HD50-3. Es un cable, en este caso de 3 pies (91,44 cm), que permite separar las 100 conexiones en 2 ramas de 50 conexiones. Cada rama de 50 *pins* se conectará a una tarjeta CIO-MINI50, que es una tarjeta que separa los 50 *pins* en 50 clavijas de fácil manipulación.

Figura 6.3: Conexión del cable C100HD50-3 a la tarjeta PCI DAS6025 en el ordenador de trabajo del proyecto.

Figura 6.4: Ejemplo de la conexión de la tarjeta PCI DAS6025 con las 2 CIO-MINI50 mediante un cable C100HD50-X

Ahora que se tienen los 100 *pins* separados, se comprobará qué es cada pin y cuál de ellos interesa para este proyecto. Según el mapa que da el *datasheet* de la tarjeta se observa lo siguiente:

Signal Name	Pin		Pin	Signal Name
LLGND	1		51	FIRSTPORTA Bit 0 *
CH0 IN HI	2		52	FIRSTPORTA Bit 1 *
CH0 IN LO	3		53	FIRSTPORTA Bit 2 *
CH1 IN HI	4		54	FIRSTPORTA Bit 3 *
CH1 IN LO	5		55	FIRSTPORTA Bit 4 *
CH2 IN HI	6		56	FIRSTPORTA Bit 5 *
CH2 IN LO	7		57	FIRSTPORTA Bit 6 *
CH3 IN HI	8		58	FIRSTPORTA Bit 7 *
CH3 IN LO	9		59	FIRSTPORTB Bit 0 *
CH4 IN HI	10		60	FIRSTPORTB Bit 1 *
CH4 IN LO	11		61	FIRSTPORTB Bit 2 *
CH5 IN HI	12		62	FIRSTPORTB Bit 3 *
CH5 IN LO	13		63	FIRSTPORTB Bit 4 *
CH6 IN HI	14		64	FIRSTPORTB Bit 5 *
CH6 IN LO	15		65	FIRSTPORTB Bit 6 *
CH7 IN HI	16		66	FIRSTPORTB Bit 7 *
CH7 IN LO	17		67	FIRSTPORTC Bit 0 *
LLGND	18		68	FIRSTPORTC Bit 1 *
n/c	19		69	FIRSTPORTC Bit 2 *
n/c	20		70	FIRSTPORTC Bit 3 *
n/c	21		71	FIRSTPORTC Bit 4 *
n/c	22		72	FIRSTPORTC Bit 5 *
n/c	23		73	FIRSTPORTC Bit 6 *
n/c	24		74	FIRSTPORTC Bit 7 *
n/c	25		75	n/c
n/c	26		76	n/c
n/c	27		77	n/c
n/c	28		78	n/c
n/c	29		79	n/c
n/c	30		80	n/c
n/c	31		81	n/c
n/c	32		82	n/c
n/c	33		83	n/c
n/c	34		84	n/c
AISENSE	35		85	DIO0
D/A OUT 0*	36		86	DIO1
D/A GND*	37		87	DIO2
D/A OUT1*	38		88	DIO3
PC +5 V	39		89	DIO4
AUXOUT0 / D/A PACER	40		90	DIO5
AUXOUT1 / A/D PACER	41		91	DIO6
AUXOUT2 / SCANCLK	42		92	DIO7
AUXIN0 / A/D CONVERT	43		93	CTR1 CLK
n/c	44		94	CTR1 GATE
AUXIN1 / A/D START	45		95	CTR1 OUT
AUXIN2 / A/D STOP	46		96	GND
AUXIN3 / D/A UPDATE	47		97	CTR2 CLK
AUXIN4 / D/A START	48		98	CTR2 GATE
AUXIN5 / A/D PACER	49		99	CTR2 OUT
GND	50		100	GND

PCI slot ↓

* Not connected on the PCI-DAS6023

Figura 6.5: Tabla de E/S de la tarjeta PCI DAS6023 y PCI DAS6025. Coloreada en amarillo los *pins* interesantes para el proyecto.

Como se puede observar, para este proyecto, sólo son interesantes los *pines* 36, 37 y 38 que son los relacionados con los convertidores D/A. Entre el pin 36 y el pin 38 tan solo se necesita uno de ellos, el que se quiera (en el código de este proyecto se utiliza el convertidor D/A 0). El *pin* 37 corresponde a la tierra del convertido. También es posible usar otra tierra, ya que todas las de la tarjeta están referidas al mismo sitio. Teniendo la diferencia de potencial entre la salida del convertidor D/A escogido y la línea de referencia ya está listo el generador de funciones.

Como los *pines* que se necesitan están en la misma CIO-MINI50, para reducir costes (véase en el apartado de presupuesto), sólo se utilizará una y se dejará la otra salida del cable C100HD50-X sin conectar.

Finalmente para acabar la conexión de este generador de funciones, se conectarán 2 cables en las clavijas de las 2 salidas que se necesitan (en este caso el *pin* 36 y 37) como se muestra en la siguiente figura:

Figura 6.6: Conexión de dos cables en las salidas del convertidor D/A0 y tierra en una tarjeta CIO-MINI50

Al final de ambos cables puede haber una clavija tipo banana para facilitar la conexión a otros dispositivos como los generadores de funciones comerciales. Ahora ya se puede conectar el generador de funciones con cualquier sistema que necesite de esta alimentación o con cualquier dispositivo de testeo, como puede ser un osciloscopio.

6.1.2.- Arquitectura de la tarjeta

A continuación, se presenta un esquema de la arquitectura de la tarjeta para poder explicarla componente por componente:

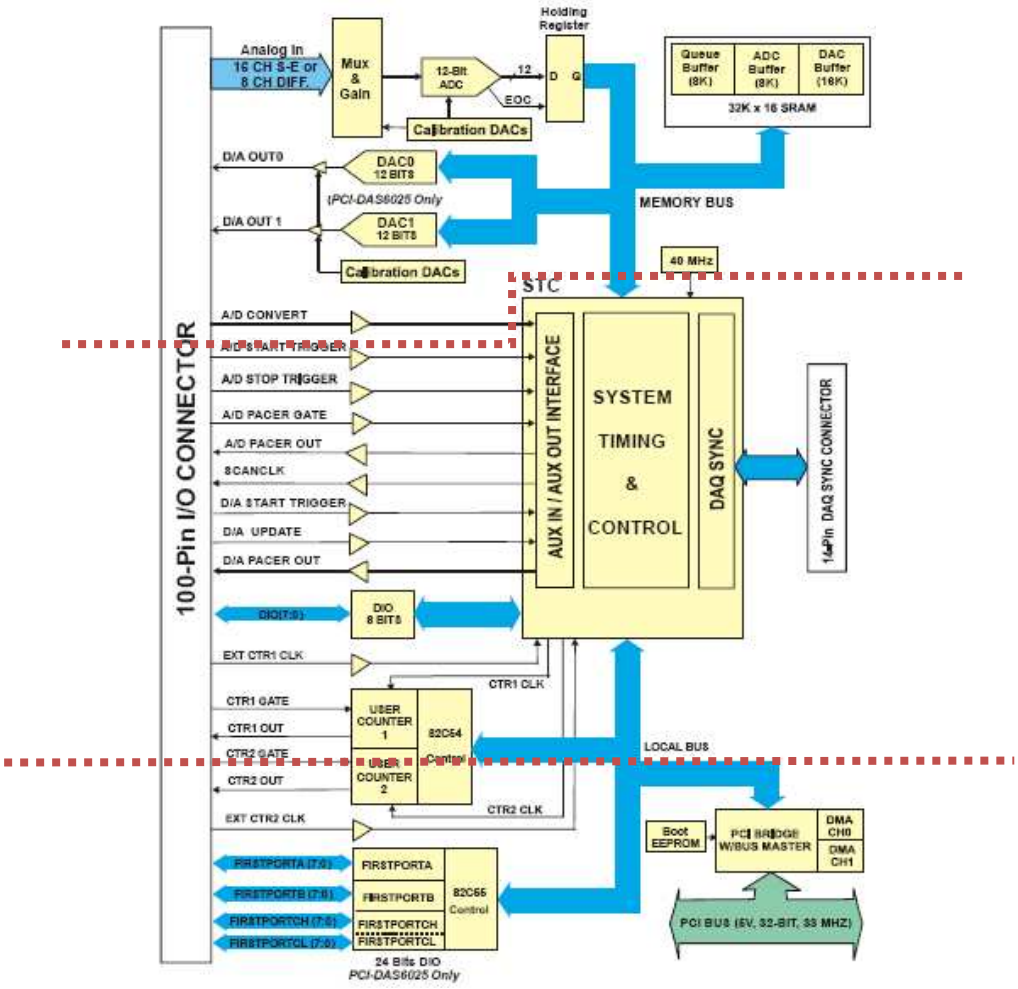


Figura 6.7: Esquema de la arquitectura de una tarjeta PCI DAS6023 y una tarjeta PCI DAS6025

Se ha dividido la figura en tres partes para poder explicarlas mejor. A la izquierda de la figura se puede ver el conector de 100 *pins* de entrada y salida. En el tercio superior de la imagen se encuentran los convertidores y dieciséis canales de entrada analógica, que proceden del conector de 100 pins y que van a parar a un multiplexor con ganancia. La salida del multiplexor va al convertidor A/D de 12 bits que convertirá la señal de entrada en un número binario del 0 al 4095 y lo guardará en un biestable (hasta acumular los 12 bits) y luego lo enviará a la memoria por el bus de memoria. Los dos convertidores D/A funcionan de modo inverso: recogen la información de memoria en forma digital y gracias a los convertidores lo traducen a valores analógicos y salen por los pins del conector como se menciona en el apartado anterior (véase el funcionamiento de los convertidores más detenidamente en el apartado siguiente).

Se ha estado hablando de la memoria, pero no se sabe a qué memoria se hacía referencia: ¿A la memoria del ordenador? No. La tarjeta dispone de una memoria propia

de 32 Kbytes. Estos 32 Kbytes están divididos en grupos: 8Kbytes para la configuración de rango de voltaje, de offset, etc. de la tarjeta; 8Kbytes están para el buffer del convertidor A/D, y los 16Kbytes restantes para el buffer de los convertidores D/A. Por ello, al tener poco espacio en memoria de la tarjeta, es interesante usarla sólo para el buffer D/A y no para guardar otros vectores como los de aspectos gráficos (éste fue uno de los fallos que se cometió al principio de este proyecto y se tuvo que arreglar, ya que el ordenador se quedaba sin memoria al utilizar el programa). Esos otros vectores hay que guardarlos en la RAM del ordenador. Estos 32 Kbytes de memoria que se encuentran en la tarjeta son memoria tipo SRAM (Memoria Estática de Acceso Aleatorio); una memoria compuesta por 6 transistores, cada bit del cual permite el acceso aleatorio en memoria, es decir, que las posiciones de memoria pueden ser escritas o leídas en cualquier orden, y, al contrario que las DRAM, permite almacenar datos mientras está alimentada haciendo innecesario refrescar los datos que tienen sus celdas. Es la mejor memoria que se podría tener, ya que se puede actualizar el buffer de memoria que se sacará por el convertidor D/A de manera rápida y sencilla.

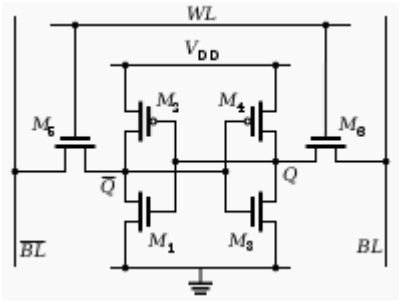


Figura 6.8: Esquema eléctrico de una celda de memoria del tipo SRAM

En el tercio del medio del esquema de la figura 6.7 está el sistema de tiempo y control y los *pins* relacionados para ello, es decir, entradas para controlar el inicio de las conversiones o finalizarlas de manera manual mediante un impulso en uno de los *pines* del conector. Además disponemos de un reloj interno de 40MHz para hacer funcionar la unidad de control.

En el tercio inferior de la tarjeta principalmente se encuentran dos chips: el 82C54 y el 82C55. El 82C54 es un chip que contiene 3 contadores de 16 bits (puede contar de 0 a 65535) a una velocidad de 20 MHz. Este chip se configura mediante diferentes *pins* del conector. El otro chip es el 82C55 que es un chip capaz de gestionar una serie de E/S de manera que actúe como entradas o salidas en función de las necesidades del usuario. En este tercio también encontramos esquematizado el bus PCI, de 32 bits, que entrega la velocidad del PC y la alimentación de la tarjeta. Los datos del bus los trata la tarjeta mediante un chip puente entre el bus ISA y la unidad de proceso y control de la tarjeta. También se dipone de dos canales DMA (acceso directo a memoria).

Después de esta visión general, es necesario enfocar esta memoria hacia la parte concreta que interesa para el proyecto: los convertidores D/A que se encuentran en el primer tercio del esquema.

6.1.3.- Funcionamiento de los convertidores D/A de la tarjeta

En este proyecto la parte de la tarjeta que se utilizará son los convertidores D/A y la memoria de la tarjeta para guardar el buffer. A continuación, una tarjeta con las características de los convertidores D/A.

D/A converter type	Double-buffered, multiplying
Resolution	12-bits, 1-in-4096
Number of channels	2 voltage output
Voltage range	±10 V
Monotonicity	*12-bits, guaranteed monotonic*
DNL	*±1.0 LSB max*
Slew rate	10V/µs min
Settling time	20V step to 0.012% (0.5 LSB): 10 µs max
Noise	200 µVrms, DC to 1 MHz BW
Glitch energy	*±24 mV @ 2 µs duration measured at mid-scale transition.*
Current drive	±5 mA
Output short-circuit duration	*Indefinite @25 mA*
Output coupling	*DC*
Output impedance	0.1 ohms max
Power up and reset	DACs cleared to 0 volts ±200 mV max

Figura 6.13: Tabla de características de los convertidores D/A de la PCI DAS6025 extraído del datasheet.

Se han enumerado los parámetros fundamentales que se deben de tener en cuenta para hacer correctamente las conversiones.

La resolución de entrada digital es de 12 bits, es decir, la entrada del convertidor necesita un número del 0 al 4095. El número 0 corresponde al nivel inferior del rango (en este caso -10 v), el 4095 el nivel superior (en este caso 10 v) y el número 2048 corresponde al valor medio, es decir, 0 v.

En esta tarjeta se puede observar que el único rango de voltaje del que se dispone es de ±10 V. Pero antes se tiene que comprobar que este valor es correcto, para ello, se van a crear diferentes valores de tensión constante mediante la función *cbAOut(),* enviando una serie de valores (del 0 al 4095 ya que esa es la resolución) y mirando el valor de salida con un voltímetro. Dados diferentes valores, se obtiene lo siguiente:

Resolución	Voltímetro	Rango ideal
0	-9,93	-10
1	-9,92	-9,995
205	-8,93	-9
1024	-4,96	-5
2252	0,99	1
2253	1	1,004
3071	4,96	5
2049	0,01	0,007
2048	0	0
2047	-0,01	-0,002
4095	9,92	10
4094	9,92	9,995

Figura 6.9: Tabla de rangos según el valor digital añadido.

Se observa que el único valor que está en rango es el 2048 que da 0v, los demás valores están desfasados siguiendo estas gráficas:

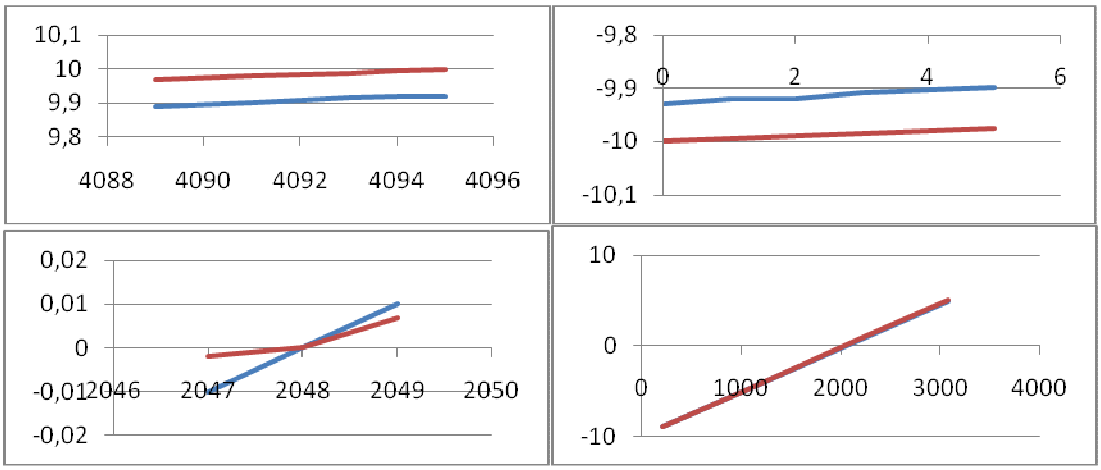


Figura 6.10: De izquierda a derecha y de arriba abajo: diferencia entre valores altos, diferencia entre valores bajos, diferencia entre valores medios y diferencia general del rango teórico (en rojo) y de los valores dados (azul).

Es obvio que, tanto por arriba como por abajo, el rango no llega a ±10 V, sino que se queda a ±9,92 V. Este es el motivo por el que se trabajará en rango de ±9,9 V y no ±10 V.

Ahora con una conversión se conocerá por cada valor digital qué salida analógica se obtendrá. La conversión se hará mediante la siguiente fórmula:

$$salida\ analógica = \frac{número\ digital \cdot 19{,}8}{4095} - 9{,}9 \quad (V)$$

Figura 6.11: Fórmula de conversión digital a analógica

A continuación, se estudiará la velocidad de conversión de los convertidores. Mirando las especificaciones técnicas, se observa que la velocidad de conversión está entre 200 y

10000 conversiones por segundo (uno de los grandes problemas que se tuvo al crear este proyecto es que se creyó que la velocidad era de 10000 conversiones del tamaño del buffer de datos entero cada segundo, pero la velocidad es de 10000 conversiones de cada dato del buffer por segundo).

Ya se conoce la velocidad de conversión y qué datos hay que convertir para conseguir la salida que se quiere. Ahora el problema es el tamaño del buffer que se ha de hacer para convertir, es decir, el vector de datos que hay que enviarle al convertidor, ya que tiene un mínimo de 1024 datos (que han quedado redondeados a 2000) y un máximo de 200000. Con la velocidad de conversión y los valores del buffer se puede obtener la frecuencia de la señal que estará entre 1mHz y 5Hz:

$$frecuencia\ mínima = \frac{Velocidad\ mínima}{Contador\ máximo} = \frac{200}{200000} = 0{,}001\ Hz$$

$$frecuencia\ máxima = \frac{Velocidad\ máxima}{Contador\ mínimo} = \frac{10000}{2000} = 5\ Hz$$

Figura 6.12: Cálculo de la frecuencia mínima y máxima del convertidor D/A.

El generador de funciones debería tener la misma frecuencia que otorga el convertidor, pero se puede mejorar la frecuencia máxima y aumentarla mediante software. Para hacerlo se debe disponer en frecuencia de 5 Hz de una velocidad de conversión de 10000 y un tamaño de buffer de 2000. Se puede dividir por software este tamaño en las partes que se quieran y repetir el tipo de onda para mejorar la frecuencia, es decir, en vez de haber un periodo por vector de buffer que hayan varios. En este caso, se aumentará la frecuencia del generador de funciones a 500 Hz, por lo tanto, en un mismo buffer de memoria estará repetido el periodo 100 veces y, por consiguiente, en cada periodo solo habrá 20 puntos para dibujar la función.

6.2.- Software

Antes de hablar sobre la tarjeta en sí misma, se verá qué software se entrega con la tarjeta. Todo este software será entregado en un CD en el embalaje junto a la tarjeta al comprarla. El CD es el siguiente que se muestra en la figura:

Figura 6.13: CD de instalación de software de Measurament Computing

Principalmente en este CD hay 3 cosas que incumben al usuario:

6.2.1.- InstalCal32

La primera aplicación interesante es el programa InstalCal32 que permite instalar la tarjeta en el PC del usuario (explicado más adelante en el apartado *Manual de instalación y uso*) y probar mediante ondas de prueba si la instalación ha sido correcta y se obtiene en la salida las señales de muestra. Si todo es correcto se podrá seguir avanzando, sino es que la tarjeta está mal instalada o dañada. Las señales que aparecen de muestra son una señal cuadrada, diente de sierra, sinusoidal y sinusoidal amortiguada. Es un software de calibración e instalación de la tarjeta.

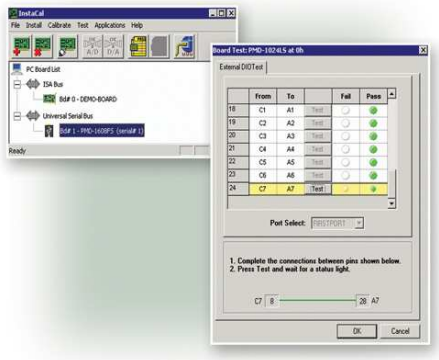


Figura 6.14: Programa InstalCal32. Imagen extraída de www.mccdaq.com

6.2.2.- TracerDAQ

Esta aplicación también es importante, ya que hace una simulación de la tarjeta y permite obtener la salida de los convertidores D/A y A/D por pantalla y saber si el procedimiento del usuario es correcto. Es un software de adquisición de datos que permite al usuario utilizar el software de osciloscopio y dar la señal de las entradas y salidas analógicas. Para su funcionamiento es necesaria tener instalada la tarjeta.

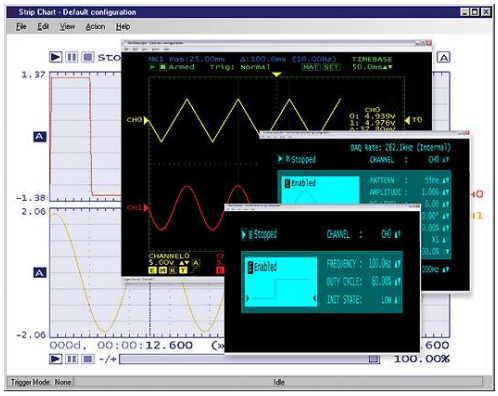


Figura 6.15: Programa TracerDAQ. Imagen extraída de www.mccdaq.com

6.2.3.- Manuales y librerías

La tercera cosa importante dentro del CD son los manuales de la tarjeta: *Datasheet*. Y también manuales de uso y de funciones que se deben utilizar para el funcionamiento de la tarjeta, ya que este CD incluye una librería para que, mediante funciones, permita al usuario programar desde los convertidores hasta los contadores si fuese necesario. Este proyecto, sin embargo, se basa principalmente en dos funciones que son las dos que se utilizan para el funcionamiento de los convertidores D/A, los convertidores que interesan para este proyecto.

Estos manuales y librerías se encuentran en un CD diferente:

Figura 6.16: CD de instalación de las librerías de Measurament Computing

De esta librería y usando la tarjeta PCI DAS6025 solo interesan las funciones relacionadas con la salida analógica, y esas funciones son dos:

- `int cbAOut(int BoardNum, int Channel, int Range, unsigned shortDataValue)`

Siendo *BoardNum* el número de tarjeta asignado (véase apartado 6.1.1), *Channel* el número de convertidor de la tarjeta (en nuestro caso el 0), *Range* el rango de nuestra tarjeta (en este caso hay una constante creada por la librería para asignarlo que es BIP10VOLTS, es decir, de -10V a 10V), y *shortDataValue* es el valor dentro de la resolución de la tarjeta (de 0 a 4095) que aparecerá en la salida y que se convertirá en un voltaje dentro del rango anterior. Esta función se utilizará para realizar una función de

amplitud constante y para, al salir, dejar la salida a 0V (valor de shortDataValue de 2048).

- `int cbAOutScan(int BoardNum, int LowChan, int HighChan, long NumPoints, long *Rate, int Range, int MemHandle, int Options)`

Esta función permite lanzar a la salida unos valores guardados en un buffer (el buffer está identificado mediante el manejador *MemHandle*). Este buffer será de un tamaño indicado en *NumPoints* y se mostrará a una velocidad de *Rate*. Estos datos serán enviados de manera interrumpida, cíclica y hasta que se le indique que pare; para ello, el valor de *Options* será: CONTINUOUS|BACKGROUND.

Toda esta información se puede encontrar en el manual de funciones de la librería.

7.- Plataforma .NET Framework

La plataforma .NET Framework es una infraestructura que simplifica el desarrollo de aplicaciones. Creada básicamente para la programación para internet y uso de base de datos, la plataforma .NET Framework ha permitido en esta aplicación hacer una interfaz de usuario cómoda y de aspecto atractivo para el usuario y de manera fácil y lógica para el programador. La plataforma .NET Framework está distribuida por la compañía Microsoft.

Los principales componentes de este entorno son:

- Lenguajes de compilación
- Biblioteca de clases de .Net
- CLR (Common Language Runtime)

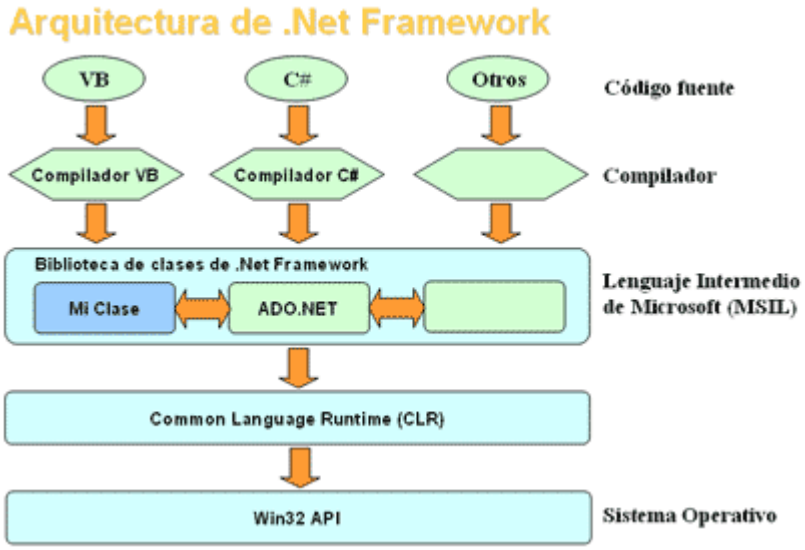


Figura 7.1: Arquitectura de la plataforma .NET Framework. Imagen extraída de desarrolloweb.com

Hoy por hoy la plataforma .NET Framework no la incluye ningún sistema operativo, ni siquiera Windows de Microsoft. Para usar cualquier aplicación que haya utilizado la plataforma .NET Framework es necesaria la instalación de dicha plataforma para poder ejecutarla. La plataforma .NET Framework se puede descargar gratuitamente de la página de Microsoft. Esta aplicación utiliza la plataforma .NET Framework, pero no es necesario instalarla porque lo hace automáticamente al iniciar la aplicación.

La plataforma .NET Framework soporta 30 tipos de lenguaje diferentes, como pueden ser C# (C Sharp), Visual Basic, C++, Perl, Cobol, etc.; aunque cada lenguaje tiene sus características propias. En este caso se ha utilizado el lenguaje C++, ya que es el lenguaje más semejante al C que es el utilizado en las prácticas de la carrera. La principal diferencia entre el C++ y el C es que el primero es una evolución del segundo y permite la programación orientada a objetos. Además el C++ permite utilizar código C y funciona perfectamente.

7.1.- Common Language Runtime (CLR)

El Common Language Runtime (CLR) es el núcleo de la plataforma .NET Framework, ya que es el entorno de ejecución en el que se cargan las aplicaciones escritas en los diferentes lenguajes ampliando los servicios que ofrece el sistema operativo estándar Win32.

La herramienta de desarrollo compila el código fuente de cualquiera de los lenguajes soportados por .NET en un mismo código, denominado código intermedio (MSIL, Microsoft Intermediate Lenguaje). Para generar dicho código el compilador se basa en el Common Language Specification (CLS) que determina las reglas necesarias para crear código MSIL compatible con el CLR. De esta forma, cualquier lenguaje elegido se traduce en un lenguaje único, el MSIL, y el único lenguaje que es compatible con CLR y, además, este código es trasparente a la aplicación porque el generador lo genera automáticamente.

Pero el lenguaje MSIL es un lenguaje intermedio y no un lenguaje máquina y, por lo tanto, no puede ejecutarse directamente. Así que es necesaria una herramienta llamada compilador JIT (Just-In-Time) que genera el código máquina real que se pueda ejecutar desde la plataforma de cualquier ordenador. De esta manera se consigue con .NET Framework cierta independencia de la plataforma, ya que éste crea el código intermedio MSIL y cada plataforma genera a partir de él y del JIT que tenga su propio código máquina.

La compilación JIT la realiza el CLR a medida que se invocan los métodos en el programa y, el código ejecutable obtenido, se almacena en la memoria caché de la computadora, siendo recompilado sólo cuando se produce algún cambio en el código fuente.

7.2.- Biblioteca de clases de .NET

El Framework organiza toda la funcionalidad del sistema operativo en un espacio de nombres jerárquico. Para ello, el Framework posee un sistema denominado Common Type System (CTS). Este sistema permite que el programador pueda interactuar los tipos que se incluyen en el propio Framework (biblioteca de clases de .NET) con los creados por él mismo (clases). De esta forma se aprovechan las ventajas propias de la programación orientada a objetos, como la herencia de clases predefinidas para crear nuevas clases, o se pueden modificar o ampliar funcionalidades de clases ya existentes.

La biblioteca de clases de .NET Framework incluye, entre otros, tres componentes clave:

- ASP.NET para construir aplicaciones y servicios Web.
- Windows Forms para desarrollar interfaces de usuario.
- ADO.NET para conectar las aplicaciones a bases de datos.

En esta aplicación se utilizará el Windows Forms ya que sólo se requiere la plataforma .NET Framework para hacer una interfaz de usuario atractiva y sencilla.

La forma de organizar la biblioteca de clases de .NET dentro del código es a través de los espacios de nombres (namespaces), donde cada clase está organizada en espacios de nombres según su funcionalidad. Por ejemplo, para manejar ficheros se utiliza el espacio de nombres System.IO y si lo que se quiere es hacer unos dibujos mediante líneas y círculos se utilizará el espacio de nombres System.Drawing. En la siguiente figura se pueden ver los diferentes "namespaces":

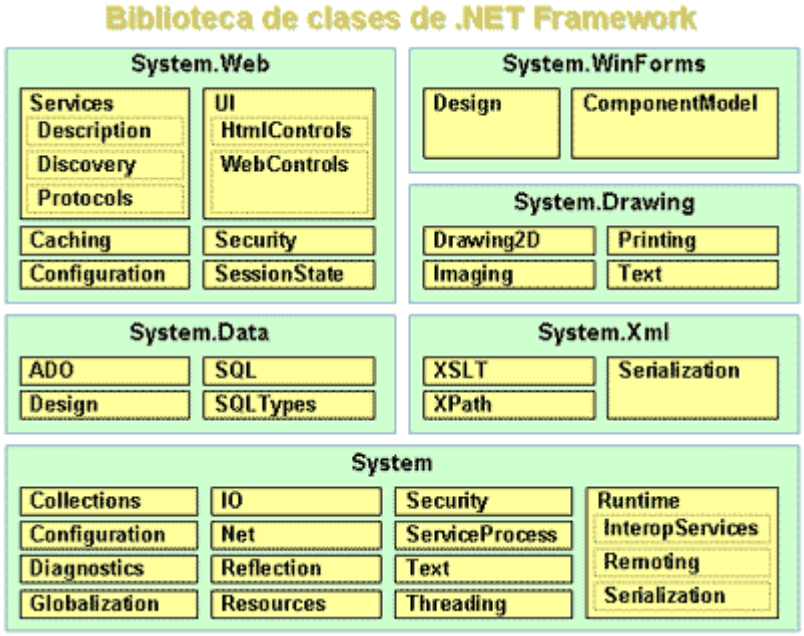


Figura 7.2: "Namespaces" de la biblioteca .NET Framework. Imagen extraída de desarrolloweb.com

La principal ventaja de los "namespace" es que se tiene todas las librerías centralizadas en un mismo nombre, System, además todos los lenguajes tiene el mismo sistema de denominación.

7.3.- Ensamblados

Uno de los mayores problemas de las aplicaciones actuales es que en muchos casos tienen que tratar con diferentes archivos binarios (DLL´s), elementos de registro, conectividad abierta a bases de datos (ODBC), etc. Para solucionarlo el Framework de .Net maneja un nuevo concepto denominado 'ensamblado'. Los ensamblados son ficheros con forma de EXE o DLL que contienen toda la funcionalidad de la aplicación de forma encapsulada. Por tanto, la solución al problema puede ser tan fácil como

copiar todos los ensamblados en el directorio de la aplicación. En el caso que nos atañe, en el archivo EXE ya están encapsulados todas las librerías, clases y archivos.

Con los ensamblados ya no es necesario registrar los componentes de la aplicación ni declarar las funciones antes de usarlas, ya que los ensamblados almacenan dentro de sí mismos toda la información necesaria en lo que se denomina el manifiesto del ensamblado. El manifiesto recoge todos los métodos y propiedades junto con otra información descriptiva, como permisos, dependencias, etc.

Para gestionar el uso que hacen la aplicaciones de los ensamblados .NET, se utiliza la llamada caché global de ensamblados (GAC, Global Assembly Cache). Así, .NET Framework puede albergar en el GAC los ensamblados que puedan ser usados por varias aplicaciones e incluso distintas versiones de un mismo ensamblado.

8.- Desarrollo del software

Una vez visto todo el hardware y la plataforma .NET Framework que se utilizará en este proyecto, se explicará el software para el funcionamiento del generador de funciones.

El programa dispone de dos archivos: form1.h y prueba3.cpp. El primero contiene la declaración de variables globales y funciones que se utilizaran en el programa prueba3.cpp. Además el archivo cabecera form1.h dispone de la creación, inicialización y parametrización de los componentes visuales de la aplicación y la destrucción de estos, así como la creación de los eventos de dichos componentes (no la gestión de los eventos que se hará en el archivo prueba3.cpp). El segundo, prueba3.cpp, es el motor de la aplicación. Contiene la función main() para arrancar la aplicación y dispones de todas las funciones que nos permitirá el funcionamiento de nuestro generador de funciones y de la aplicación.

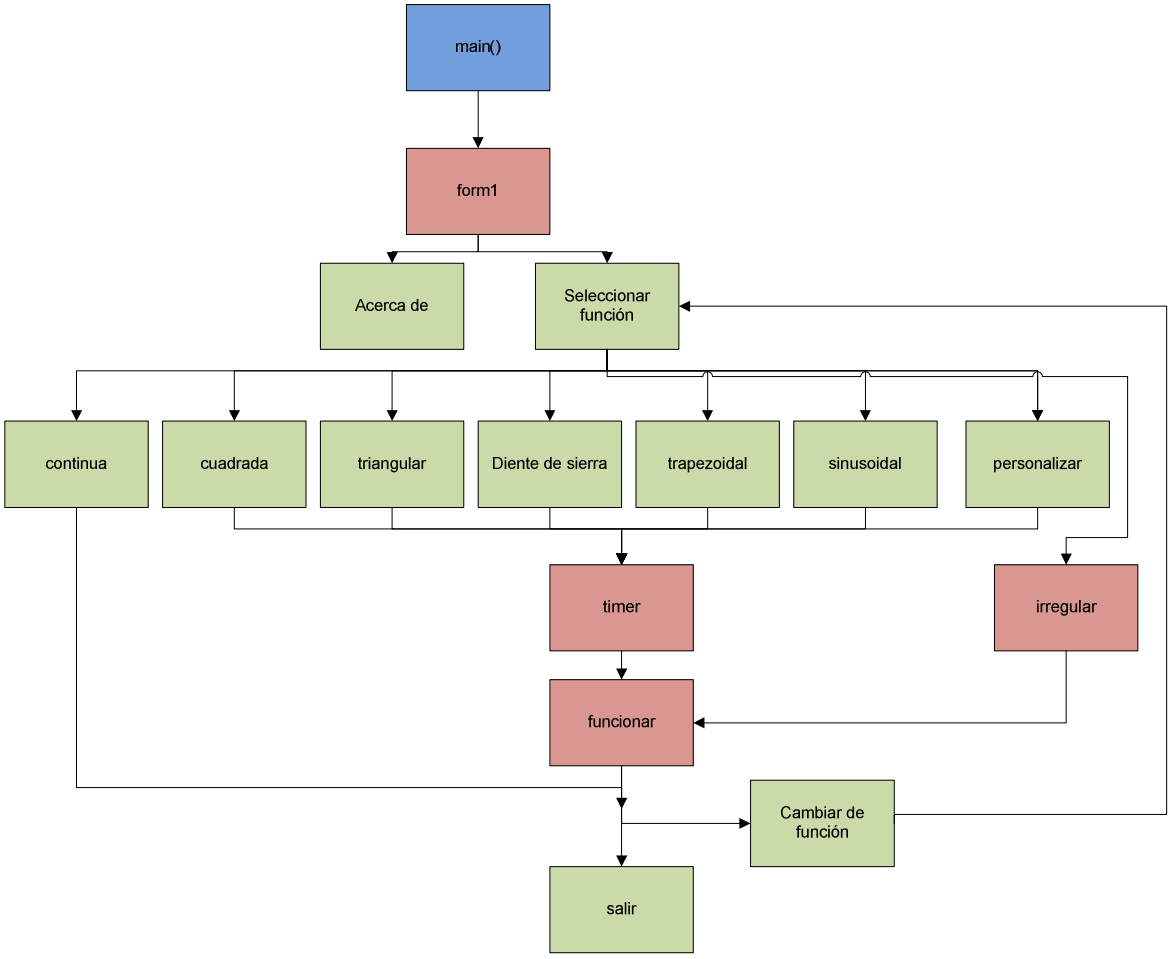


Figura 8.1: Diagrama de flujo general del programa. En azul el inicio, en verde lo que podemos seleccionar y en rojo las funciones llamadas automáticamente.

Para describir estos archivos se pueden diferenciar dos partes muy claras: la parte funcional y la parte gráfica. En la parte funcional se encuentra el código necesario para el funcionamiento perfecto del generador de funciones; en la parte gráfica se encuentra el código necesario para dar al usuario una manera cómoda y fácil de controlar el generador de funciones.

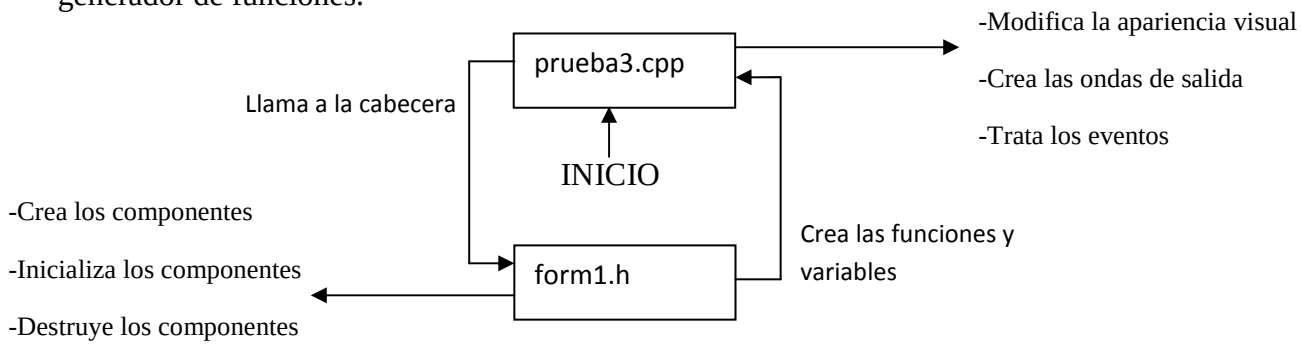


Figura 8.2: Diagrama de flujo del programa

8.1.-Parte gráfica

En primer lugar, se creará un selector para escoger el tipo de onda y un botón para confirmar que el valor del selector es el que se desea (comboBox1 y button2). Este selector contiene los tipos de onda de los que se dispone: Continua, Cuadrada, Triangular, Diente de sierra, Trapezoidal, Sinusoidal, Irregular y Personalizada (cuadrado verde de la figura 8.1).

En segundo lugar, se creará el aspecto visual para conseguir los parámetros para crear la función que se ha seleccionado. Para ello se crearán varios componentes gráficos como: Label, NumericUpDown, RadioButton, Button o PictureBox (figura 15.36 dentro del apartado anexos). Se han de inicializar estos componentes (figura 15.37). La creación de componentes y la parametrización inicial se hacen en el archivo de cabecera form1.h.

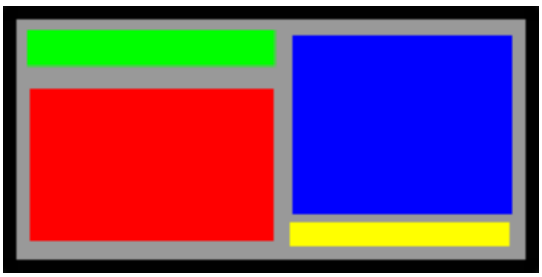


Figura 8.3: Esquema visual de nuestra aplicación.

Una vez creados los componentes la parametrización de estos dependerá del tipo de función escogida. Por ello, al seleccionar una función o cambiar de función escogida es necesario cambiar parámetros de los componentes creados (visibilidad, texto…) todo para llegar a conseguir las variables necesarias para la función escogida. En la figura 15.24 dentro del apartado anexos se puede observar que según el tipo de función seleccionada disponemos de unos componentes con unos determinados parámetros u

otros. La función continua solo necesita la amplitud; la cuadrada necesita la amplitud, el offset, la frecuencia y el dutycicle; para la triangular son necesarios la amplitud, el offset, el tiempo de subida y el tiempo de bajada; para la diente de sierra se necesita la amplitud, el offset, el tiempo de bajada o el de subida; para la trapezoidal se necesita la amplitud, el offset, el tiempo ON, el tiempo de subida, el tiempo de baja y el tiempo OFF; la función sinusoidal necesita la amplitud, el offset, la frecuencia y el desfase; la función irregular, la amplitud, el offset y la velocidad de mostreo; en el caso de la función personalizada no existen parámetros sino que hay que dibujarla y se tratará en un apartado aparte (esta serie de componentes estarán ubicados en el cuadrado rojo de la figura 8.1). Principalmente los componentes que encontramos son Labels que nos da información de que es cada cosa, NumericUpDown que es un selector de números dentro de un rango, Button que son botones de confirmación, y, también, en menor medida, encontramos selectores RadioButton e imágenes PictureBox.

Para conseguir los parámetros mencionados y no crear muchos componentes lo que se hace es utilizar estos para diferentes cosas, por ello, se usará los NumericUpDown de la manera mostrada en la siguiente figura:

	Continua	Cuadrada	Triangular	Diente de sierra	Trapezoidal	Sinusoidal	Irregular
NumericUpDown 1	Amplitud	Amplitud	Amplitud	Amplitud	Amplitud	Amplitud	Amplitud
NumericUpDown 2		Offset	Offset	Offset	Offset	Offset	Offset
NumericUpDown 3		Frecuencia				Frecuencia	velocidad
NumericUpDown 4		Dutycicle				Desfase	
NumericUpDown 5			Tiempo sub		Tiempo sub		
NumericUpDown 6			Tiempo baj	Tiempo sub/baj	Tiempo baj		
NumericUpDown 7					Tiempo ON		
NumericUpDown 8					Tiempo OFF		

Figura 8.4: Tabla con la correspondencia de cada componente NumericUpDown para los diferentes tipos de función

Y, para ayudar, los diferentes Label explicarán qué poner en cada NumericUpDown. Además los NumericUpDown tienen regulados su mínimo y máximo según la función escogida. Si se cambia con el selector la función, desaparecen estos componentes visuales y reaparecen los necesarios para la próxima función. Estos cambios son provocados por los eventos del programa (el evento del botón aceptar y del cambio de función), por ello, el tratamiento de la configuración de los componentes se hace en el archivo prueba3.cpp.

Se puede observar que la función diente de sierra tiene el NumericUpDown6 que es en tiempo de subida o de bajada. Para seleccionarlo existen dos RadioButton con los que escoger qué tiempo, si el de subida o el de bajada, será nulo.

Para la función personalizada se creará una cuadrícula 21x10 de pictureBox. La fila superior corresponderá a 10v y por cada fila se restará 1v hasta llegar a -10v. También habrá un NumericUpDown (NumericUpDown9) para seleccionar el peridodo y un selector (comboBox2) para poder escoger si los cambios son progresivos o pulsantes, es decir, si se hacen por saltos los cambios o no. La cuadrícula será inhabilitada y de color rojo menos la 1º columna que estará habilitada y verde. Al presionar una casilla de la primera columna se pondrá en azul y las demás en rojo y se habilitará la columna de su derecha y se pondrá de color verde y, así, en todas las columnas. El código que posibilita esto se encuentra en la figura 15.4 y 15.31 de los anexos. Para crear la cuadrícula se ha utilizado un vector de *pictureBox*, ya que era más sencillo que hacerlo uno a uno. Para crear los vectores es necesario, anteriormente, guardar un espacio de memoria del ordenador (figura 15.2 del anexo). Este espacio no se puede guardar en la memoria de la tarjeta ya que ésta no es suficientemente grande para hacerlo.

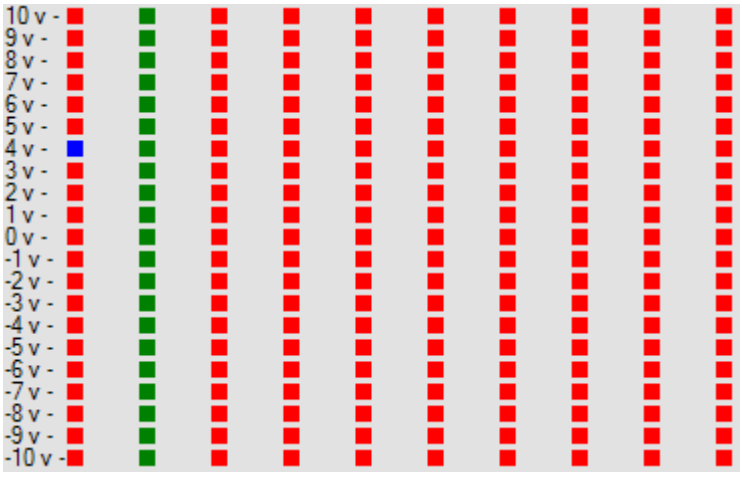


Figura 8.5: Cuadrícula 21x10 de pictureBox para “dibujar” la onda.

La última parte del aspecto gráfico es realizar una gráfica con la función de salida (ubicado en el cuadrado azul de la figura 8.1). En primer lugar hay que crear un *pictureBox* para añadir todos los elementos dentro de él. Luego se creará un eje de coordenadas como el siguiente:

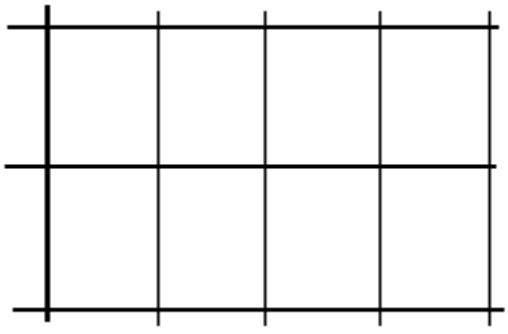


Figura 8.6: Esquema del eje de coordenadas creado para dibujar la onda en la aplicación

Para ello se debe crear unas funciones que nos permita crear las líneas y configurarlas con el grosor o el color que queramos como se hace en las figuras 15.22 del anexo. La creación del eje personalizado de coordenadas se encuentra en la figura 15.27 del anexo. En este eje de coordenadas dibujaremos tres características de la onda diferentes. La primera, de color azul (figura 15.19), se dibujará la onda que hemos configurado; de color verde (figura 15.20), se dibujará el valor eficaz del periodo de la onda, y, por último, de color rojo (figura 15.21) se dibujara el valor medio. Para dibujar la función lo que se ha hecho es dividir el espacio para dibujarlo en 4096 partes el eje "y" y "Count" partes el eje "x". Así en cada parte se inserta el valor que aparece en el buffer de la tarjeta (en el próximo subapartado se explicará que es el buffer y "Count").

En el próximo apartado (apartado número 9), podemos observar el aspecto gráfico de nuestra aplicación de cada una de las funciones y cómo los componentes quedan situados en nuestra aplicación.

8.2.- Parte funcional

Este es el apartado más importante del proyecto. Aquí se crearán los tipos de ondas y se enviarán a la salida. Principalmente dispondremos de un buffer en memoria de valores entre 0 y 4095 que correspondrá a los valores de amplitud necesarios para dibujar la onda deseada. Los datos del buffer correspondrá a *n* veces el periodo de la señal (*n* podrá ser 1, 10 o 100). El tamaño del buffer dependerá de la frecuencia que deseamos para poder tener siempre la misma resolución temporal que será de 100µs. A este tamaño del buffer le llamaremos "Count".

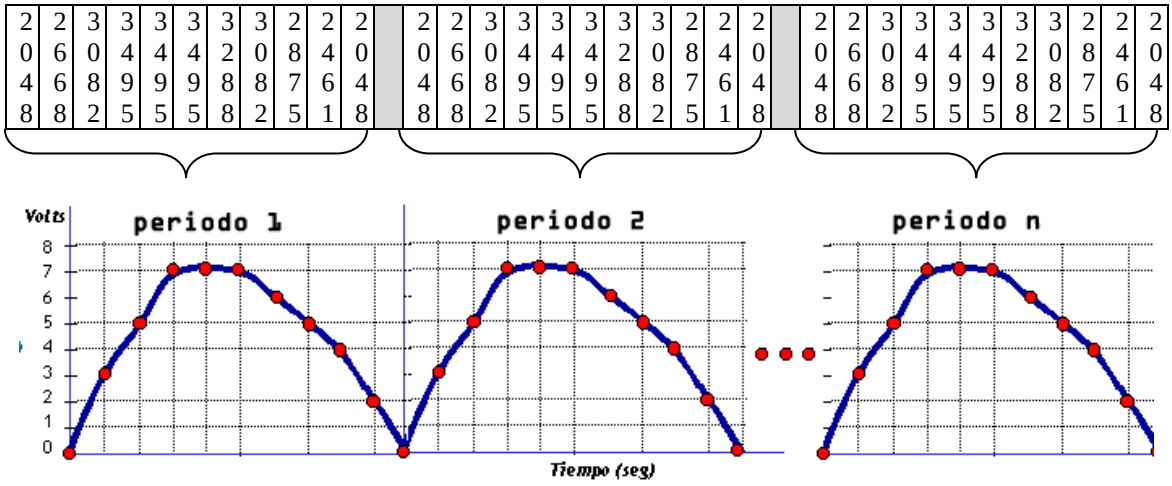


Figura 8.7: Esquema de un buffer y su correspondente salida analógica. Las casillas del buffer en gris son casillas que realmente no existirían en el buffer, por lo tanto, este ejemplo el buffer tiene un tamaño (Count) de 30.

A continuación, la explicación de cada tipo de onda:

8.2.1.- Función de amplitud constante

Para la función de amplitud constante (o llamada en el código como función continua) tan solo hay que lanzar el voltaje introducido en un *numericUpDown* a la salida. En este caso es el único caso que lo explicado con anterioridad no es necesario ya que solo e dispone de un valor y no hay porque crear un buffer de datos. Como solo se dispone de un dato se enviará a la tarjeta mediante la función *cbAOut(),* explicada en el apartado 6.1.3.

El valor obtenido mediante el *NumericUpDown* será el valor en volteos que se desea en la salida y estará comprendido entre -9,9 y 9,9 V. Para conseguir ese voltaje en la salida hay que enviarle un valor al convertidor D/A adecuado que estará entre 0 y 4095. Para calcularlo usaremos la siguiente fórmula:

$$número\ digital = \frac{(voltaje\ deseado + 9{,}9) \cdot 4095}{19{,}8}$$

Figura 8.8: Fórmula de conversión analógica a digital

Después solo hay que enviar ese valor al convertidor mediante la función *cbAOut()*. En la figura 15.7 de los anexos se puede ver cómo se ha hecho para elaborar una salida de amplitud constante.

8.2.2.- Función cuadrada

La función cuadrada es una función que pasará del valor offset al valor amplitud+offset cada cierto tiempo. Un tiempo determinado por el ciclo de trabajo y por la frecuencia de la señal. Como no es una señal de amplitud constante se necesitará enviar la función mediante la función *cbAOutScan()* explicada en el apartado 6.1.3. En ese apartado se explica que para crear una señal alterna es necesario un buffer de memoria con los valores que se desean enviar y hacer los envíos configurando la velocidad del convertidor D/A. Para ello se ha creado la función *timer()* (figura 15.6 de los anexos) que según la frecuencia que se desea de la señal se establece un número de valores que tendrá el buffer (tamaño del buffer, *Count*) y se calcula el valor de la velocidad de conversión (una variable es constante según la frecuencia, el tamaño del buffer; y la otra es variable para dar la frecuencia exacta, la velocidad de conversión). Además habrá otra variable (divisor) que indicará cuántos periodos de la onda (*n* veces) hay en el buffer de memoria y esto permitirá llegar a frecuencias más elevadas (500Hz).

Ahora que ya se ha hablado de la velocidad del convertidor y el tamaño del buffer de datos, falta rellenar este buffer con los valores que permitan crear la forma cuadrada.

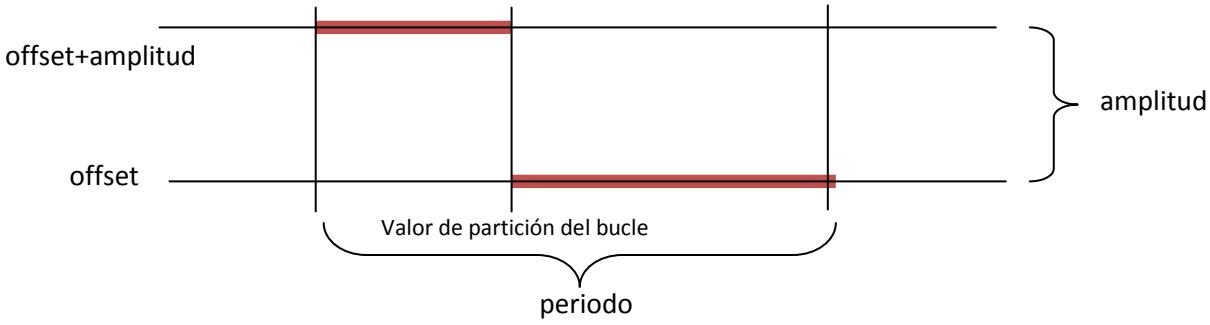


Figura 8.9: Aspecto de la onda de la función cuadrada

Para ello se crearán tres bucles: dos para crear un periodo y el otro por si es necesario repetirlo varias veces dentro del buffer (en caso de que la variable *divisor* sea diferente a 1).

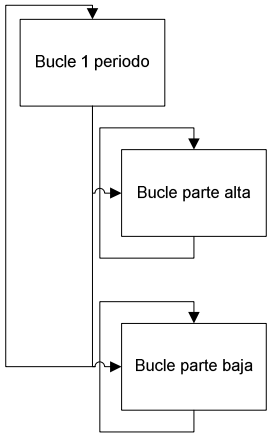


Figura 8.10: Diagrama funcional de la función cuadrada

Con el ciclo de trabajo se sacará a partir de qué valor del tamaño del buffer se ha de conmutar de estado alto a estado bajo. De los dos bucles que se tienen para rellenar el periodo de valores, el primero irá del inicio al valor anteriormente indicado y se incluirán dentro de esas posiciones del buffer el valor *amplitud+offset* traducido dentro de su resolución como se explicó en el apartado anterior. El segundo bucle irá del valor que ha dado el ciclo de trabajo hasta el final del periodo, y se rellenará con valores iguales al offset. Por lo tanto el buffer podría estar dividido de la siguiente manera:

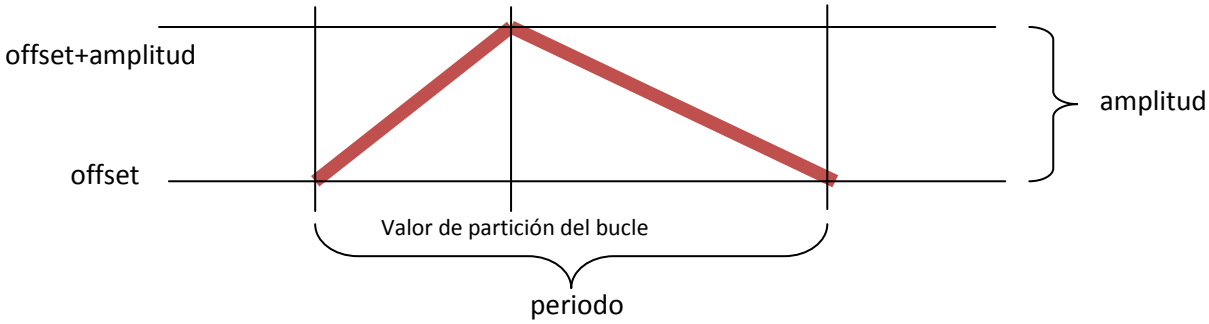
Periodo1 Offset+ amplitud	Periodo 1 Offset	Periodo 1 Offset	Periodo 2 Offset+ amplitud	Periodo 2 Offset	Periodo 2 Offset	Periodo 3 Offset+ amplitud	Periodo 3 Offset	Periodo 3 Offset

Figura 8.11: Ejemplo de un posible buffer

En la figura 15.8 se muestra cómo se ha hecho.

8.2.3.- Función triangular

El sistema es como en la función cuadrada, pero ahora cambiando la manera de rellenar el buffer. Antes se dividió el periodo en dos tramos, el alto y el bajo, y cada tramo se rellenaba mediante un bucle con los valores deseados. En esta función también se dividirá el periodo en dos tramos, el de subida y el de bajada. Para hacer la división se dividirá el tiempo de subida entre el periodo. Ese será el parámetro de cambio entre los dos bucles como se hizo en la onda anterior. Esta vez para rellenar el buffer en el primer bucle se hará sumando el offset con un incremento que se calculo siguiendo la siguiente fórmula:



$$\Delta = \frac{Amplitud}{Valor\ partición\ del\ bucle}$$

Figura 8.12: Dibujo de la onda triangular y fórmula de cálculo del incremento

Para rellenar la segunda parte del periodo lo que se hará es, desde el valor *offset+amplitud*, ir restando un decremento calculado de la siguiente manera:

$$\Delta = \frac{Amplitud}{Valor\ del\ periodo - valor\ partición\ del\ bucle}$$

Figura 8.13: fórmula de cálculo del decremento

Como en el caso anterior también se usará tres bucles como en el esquema de la figura 8.9. Se puede ver esta función *triangular()* en la figura 15.9 del anexo.

8.2.4.- Función diente de sierra

Este caso es como el triangular, pero en vez de haber dos bucles para rellenar el buffer sólo hay uno y, según como se quiera la forma diente de sierra lo que se hará es sumar el offset más un incremento (amplitud/periodo) o restar del valor amplitud+offset un decremento (amplitud/periodo), así solo se conseguirá la subida o bajada de la onda triangular que será el diente de sierra deseado. En la figura 15.10 se puede ver dicha función.

8.2.5.- Función trapezoidal

Este caso es como las funciones anteriores, pero el buffer se rellenará mediante 4 bucles. Para dividir el periodo en 4 partes lo que se hará es, en la primera parte, dividir el tiempo de subida entre el periodo; la segunda parte será el valor de la primera parte más la división del tiempo ON entre el periodo; la tercera parte será el valor de la segunda parte más la división del tiempo de bajada entre el periodo; y la cuarta parte será el periodo restante.

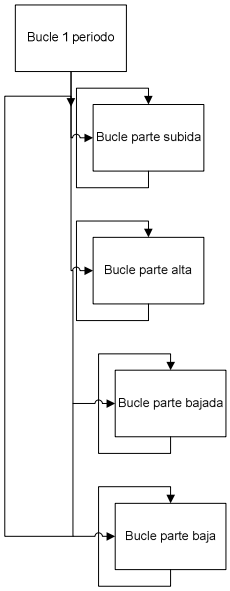


Figura 8.14: Diagrama funcional de la función trapezoidal

Una vez se tienen los cuatro bucles definidos se rellena el buffer como se hizo en la función cuadrada y triangular. La primera parte será el tiempo de subida y se rellenará desde offset hasta offset+amplitud con un incremento calculado como el de la figura 8.11; la segunda será el tiempo en alto, es decir, constantemente el valor offset+amplitud; la tercera parte será el tiempo de bajada, se rellenará el buffer con valores de offset+amplitud menos el decremento que se calcula mediante la fórmula 8.12, y la cuarta parte será el tiempo bajo que se rellenará con el valor de offset. En el caso que unos de los tiempos sea nulo implicará que dos valores de partición de bucle coincidan y por lo tanto, en uno de los bucles no entrará y no formará esa parte. En la figura 15.11 de los anexos se puede encontrar la función trapezoidal().

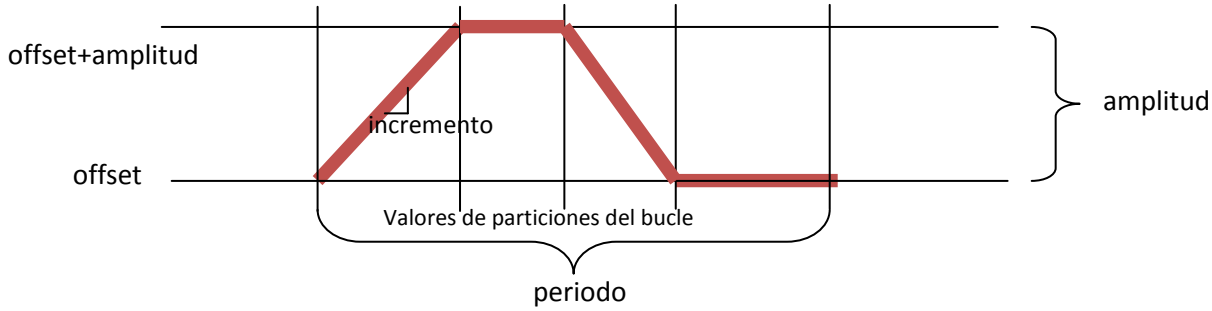


Figura 8.15: Dibujo de la onda trapezoidal

8.2.6.- Función sinusoidal

Para la función sinusoidal sólo se usarán dos bucles: uno para rellenar el periodo y el otro para rellenar el buffer con *n* periodos (siendo *n* el valor de la variable divisor). Este último bucle se ha utilizado con las otras funciones. Para rellenar el buffer se usará la función *sin()* que se encuentra en la librería *math.h*.

Esta función dibuja un seno de valor 1 a -1. Este valor se multiplica por la amplitud para sacar un valor de -9,9 a 9,9. Se le suma el offset y el resultado obtenido será la función deseada. La función *sin()* necesita el valor en radiantes del ángulo que se necesita. Como se rellenará el valor del buffer con un periodo, es decir, 360º de la función *sin()* lo que se hará es pasar el tamaño del buffer a incrementos de grados mediante una regla de tres. A este incremento de grados se le sumará el desfasaje y se pasarán a radiantes. Se puede ver esta función en la figura 15.15 del anexo.

$$radiantes = \frac{desfase + \sum_{i=0}^{tamaño\ buffer\ un\ periodo} i \cdot (incremento)}{180} \cdot \pi \qquad siendo$$

$$incremento = \frac{tamaño\ del\ buffer\ en\ un\ periodo}{360}$$

Figura 8.16: Fórmula del calculo del ángulo que debe de introducirse en la función *sin()* de math.h

8.2.7.- Función personalizada

En la función personalizada lo que se hará es dividir los tramos del periodo en 10 tramos (los 10 tramos que se pueden “dibujar” en la onda personalizada). Cada tramo será una décima parte del periodo y se rellenará el buffer mediante un bucle en cada tramo. Si se ha escogido “cambios pulsantes”, los valores obtenidos del dibujo del usuario (véase apartado 8.1 y figuras 15.4 y 15.31) serán los valores del buffer en cada tramo, convertidos en un valor del 0 al 4095 para que el convertidor D/A pueda traducirlo. En cambio, si se ha escogido “cambios progresivos”, se sumará el valor escogido del usuario más un incremento. Este incremento se calcula de la siguiente manera:

$$\Delta = \frac{valor\ escogido\ siguiente - valor\ escogido\ actual}{\frac{periodo}{10}}$$

Figura 8.17: fórmula de cálculo del incremento

Si el valor actual es más grande que el siguiente el valor incremental será negativo y por lo tanto se hará un decremento (bajada). Se puede ver la función *personalizada()* en la figura 15.13 de los anexos.

8.2.8.- Función irregular

Esta función es completamente diferente a las demás, ya que no se usa la función *timer().* En esta ocasión el tamaño del buffer siempre será el mismo y la velocidad del convertidor dependerá del periodo con la que se quieran mostrar los datos.

Se tiene que asignar el manejador de memoria dentro de esta función, ya que se hacía en la función *timer()* y ahora esta función no es llamada. Es necesario asignar memoria de la tarjeta al buffer para su funcionamiento. Para rellenar el buffer lo que se hará es crear un número al azar (*random*) que estará entre 0 y el valor que se le ha dado a la amplitud. Ese valor aleatorio se le sumará al offset. Ya se cuenta con una forma de función aleatoria que se puede ver en la figura 15.17.

8.2.9.- Cálculo del valor medio y el valor eficaz

Además de crear la función deseada, se procederá a calcular su valor medio y su valor eficaz.

Se llama valor medio de una tensión alterna a la media aritmética de todos los valores instantáneos de tensión medidos en un cierto intervalo de tiempo. En este caso, como se cuenta con un buffer con los valores, sólo se ha de sumarlos y dividirlos por el tamaño del buffer.

$$media = \frac{\sum v(i)}{i}$$

Figura 8.18: fórmula de cálculo del valor medio

Este valor será mostrado por pantalla y dibujado. Podemos ver como hemos hecho la media en la figura 15.18 de los anexos.

Se llama valor eficaz de una corriente alterna al valor que tendría una corriente continua que produjera la misma potencia que dicha corriente alterna, al aplicarla sobre una misma resistencia. Teniendo los valores del buffer se puede calcular el valor eficaz mediante la siguiente fórmula:

$$valor\ eficaz = \sqrt{\frac{\sum v(i)^2}{i}}$$

Figura 8.19: fórmula de cálculo del valor eficaz

Este valor será mostrado por pantalla y dibujado. Se puede ver cómo se ha hecho la media en la figura 15.19 de los anexos.

8.2.10.- Inicialización, cambio entre funciones y cierre de programa

Al abrir el programa lo primero que se hará es llamar a la función *inicializacion(),* figura 15.3, que permite poner las salidas del convertidor D/A a 0 por si se había utilizado anteriormente y se había dejado con algún valor que podría dañar los dispositivos que se conecten en la salida.

Cuando se cambia de función, se deja de emitir una función o se cierra el programa; se hará como en *inicializacion()* y se pondrá la salida del convertidor D/A a 0V. Además, también se vaciará la memoria de la tarjeta y del ordenador y se indicará mediante la función *cbStopBackGround()* que debe parar de lanzar a la salida lo que hay en el buffer, ya que el lanzamiento del buffer es cíclico y debe interrumpirse para su detención. Con todo esto se conseguirá que la parada del programa sea segura y que el cambio entre funciones no provoque errores.

Por eso mismo, al cambiar el componente *comboBox1* de valor, figura 15.29, indicará que se quiere cambiar de tipo de función y, por ello, se realizará lo anteriormente mencionado. Igualmente sucederá, cuando se apriete el botón de Stop (button4), figura 15.28. En el caso de presionar el botón de *Salir* (button3), figura 15.27, sucederá lo mismo pero, además, se cerrará la aplicación.

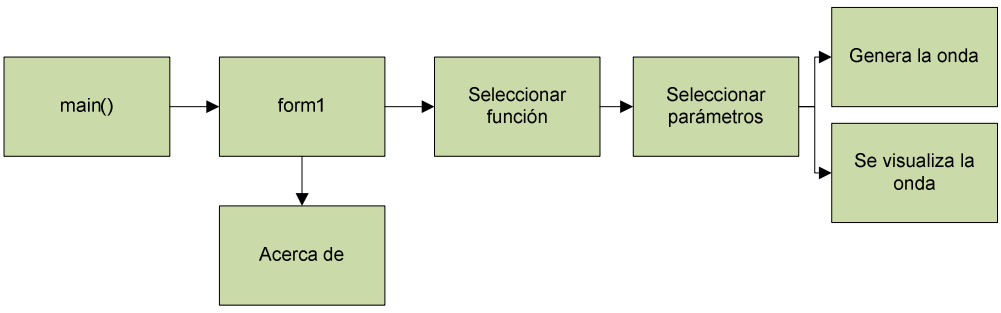


Figura 8.20: diagrama de flujo de los pasos del programa

8.3.- Pasos para la realización del código

En los apartados 8.1 y 8.2 se ha explicado cómo ha quedado el código que hace funcionar el programa. Pero para realizar el código han sucedido varios problemas. En este apartado se comentarán ligeramente los pasos que se han realizado para llegar al código final explicado con anterioridad y los problemas que han surgido.

Lo primero que hice, como autor del proyecto, para plantearlo fue leerme los manuales de la tarjeta PCI DAS6025 y las funciones de las librerías e intentar entender cómo funcionaba. Este fue uno de los puntos que más tiempo me llevó, ya que era necesario tener una base para poder empezar, una base que inicialmente me era completamente desconocida.

El segundo paso fue crear un programa para MS-DOS que me permitiera poner un valor de voltaje y sacarlo en la salida de manera continua. En este paso me di cuenta que la tarjeta no podía enviar 10V, ya que solo llegaba a 9,92. Con diferentes pruebas de diferentes voltajes y valores pude llegar a deducir que nuestro rango lo podía limitar de -9,9V a 9,9V.

El tercer paso fue crear fórmulas que nos permitiesen crear la función cuadrada, triangular, sinusoidal, diente de sierra y trapezoidal e implementar estas fórmulas a programación C. Aquí era importante mirar como poder crear un contador de tiempo real. Se podía hacer mediante tres métodos diferentes. Dos de ellos son mediante eventos o interrupciones, pero ambos están condicionados por el sistema operativo que puede hacer perder interrupciones o eventos que no permitirían un control perfecto del tiempo. Otro método sería controlar el hardware, es decir, controlar el tiempo de disparo del convertidor D/A. Con este método se puede controlar perfectamente el tiempo entre disparos sin necesidad de otros elementos como interrupciones o contadores.

Después creé un programa para MS-DOS con las funciones mencionadas anteriormente para probar que funcionaba correctamente. Pero no funcionó. En la salida no sacaba ningún valor. Estuve mucho tiempo arreglando el error (es de los errores que más me costó encontrar el porqué fallaba). El error era una mal interpretación del manual de la tarjeta PCI DAS6025. Creía que la velocidad máxima de conversión del buffer (de 10000 transferencias por segundo) era por cada buffer y no por cada valor del buffer, así que puse más velocidad de la que el convertidor podía soportar. Al arreglar este error, seguía sin dar valor en la salida. Había otro error. El tamaño del buffer tenía un máximo (el de la memoria de la tarjeta destinada para la conversión D/A), pero tenía también un mínimo que no había leído. El mínimo del tamaño del buffer era de 1024 valores. Arreglando estos parámetros conseguí, después de bastante tiempo, que funcionase.

Funcionando ya el proyecto en MS-DOS ahora faltaba hacer una interfaz de usuario sencilla, cómoda y atractiva. Encontré varias maneras para hacerlo. La primera era mediante las librerías propias de C++, pero era una manera de programar bastante difícil ya que se usaban muchos objetos y clases para realizar poca cosa y, además, el aspecto gráfico era poco atractivo. Otra opción era usar un software gratuito ajeno llamado

FormBuild. Éste presentaba un gran inconveniente que era la dificultad de insertar código ajeno (el programa creado para MS-DOS de nuestro generador de funciones) al código generado para el aspecto gráfico. La tercera opción era usar la plataforma .NET Framework que nos permitía hacer una interfaz de usuario atractiva y, además, con un gran manual de ayuda que nos proporcionaba la página web de Micrososft (véase el apartado de Bibliografía y Referencias) y era una plataforma gratuita. Así que escogí esta tercera opción y empecé a mirar el tutorial y crear pequeños aspectos gráficos que poco a poco irían evolucionando hasta crear el aspecto gráfico de nuestro programa.

Ahora era cuestión de fusionar el aspecto gráfico creado con el código del generador de funciones creado en MS-DOS y comprobar que todo funcionaba correctamente, además añadí la función irregular y funcionaba correctamente.

Las funciones alternas funcionaban como máximo a 5 Hz. Era una frecuencia muy baja pero era el máximo que podían darme los convertidores D/A de la tarjeta PCI DAS6025. La idea era subir la frecuencia repitiendo en un mismo buffer varias veces el periodo de una onda. Así pude subir la frecuencia a 500Hz, pero el gran inconveniente, aparte de que perdía resolución, era que tenía que reprogramar el valor máximo y mínimo de los parámetros de cada función y, también, reprogramar la manera de crear las ondas, ya que ahora podía tener varios períodos en el buffer. Después de varios intentos se consiguió.

El próximo paso era aprovechar las ventajas del aspecto gráfico del programa y poder dibujar por pantalla un periodo de la onda que había configurado. Mediante líneas podía crear un eje de coordenadas en pantalla y un dibujo de los valores de salida que tenía, así, si no tenía un osciloscopio podía observar la salida que obtenía. Además podía rizar el rizo y calcular el valor medio y el valor eficaz y dibujarlo también por pantalla.

Ya que tenía un aspecto gráfico, una manera de aprovecharlo era crear un tipo de función con la que el usuario pudiese “dibujar” su propia onda y se extrajera a la salida. Por ello creé la función “personalizada” que el principal problema que me ha dado ha sido la creación y configuración del vector de *pictureBox*() ya que por internet no había nada sobre cómo hacerlo. Además, cuando me funcionó, si el programa llevaba mucho tiempo encendido salía un error de falta de memoria y se cerraba. Eso era debido a que guardé el vector en la memoria interna de la tarjeta y era demasiado pequeña, ahora está guardado en la memoria del ordenador que es más grande y funciona perfectamente.

Por último he creado el instalador del programa mediante un software gratuito de apoyo.

9.- Resultados finales

El resultado final de todo el código explicado anteriormente presenta la siguiente forma visual de inicio:

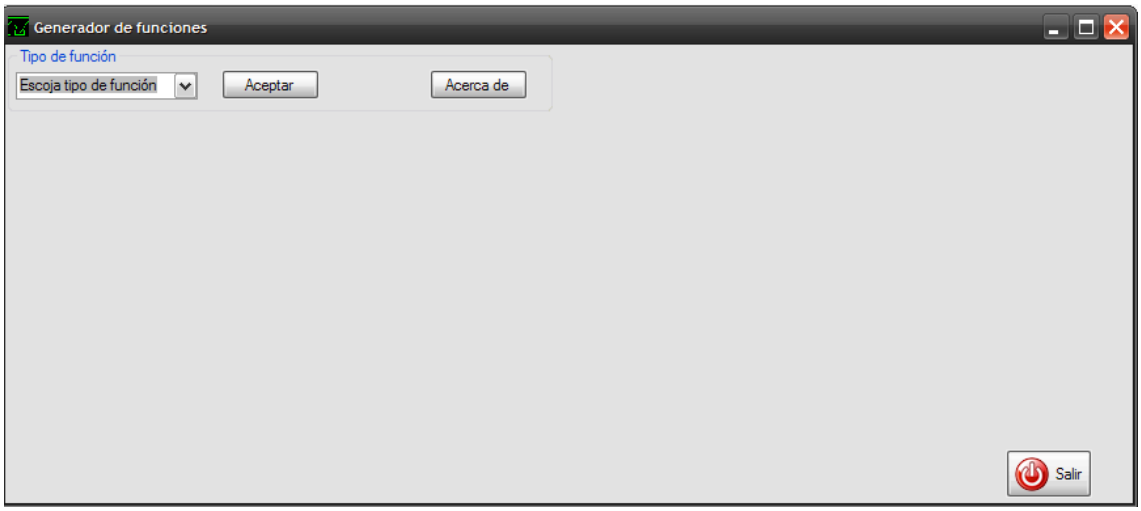


Figura 9.1: Aspecto inicial del programa

Ahora se procederá a ver diferentes ejemplos de resultado final de varias formas de onda. Para cada ejemplo se enseñará el aspecto visual del programa para ver los parámetros y como el software ha dibujado la onda, y también se observará la forma de onda que se obtiene en la salida de los *pins* de la tarjeta (que es lo que realmente es importante). Para observar esta salida se utilizará un osciloscopio convencional. Las ondas visualizadas en el osciloscopio y en la pantalla deben de ser idénticas y deben de tener las características que se han configurado.

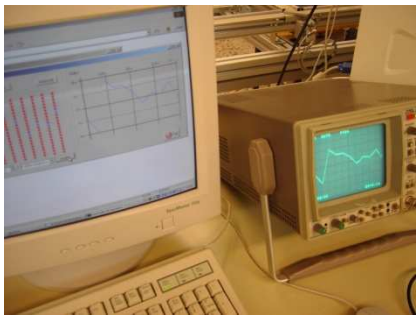


Figura 9.2: Fotografía del osciloscopio y la pantalla del ordenador. Se observa que ambas disponen de la misma solución

Estos son algunos ejemplos concretos para enseñar una muestra de lo que nuestro generador de funciones puede llegar ha hacer. Estos son los ejemplos más variados para demostrar las ondas que puede crear nuestro generador.

9.1.- Ejemplos de función de amplitud constante

La función continua solo permite escoger la amplitud. Esta función puede ser útil para alimentar un aparato que funcione con alimentación constante.

9.1.1.- Ejemplo: Función de amplitud constante de 9,9v

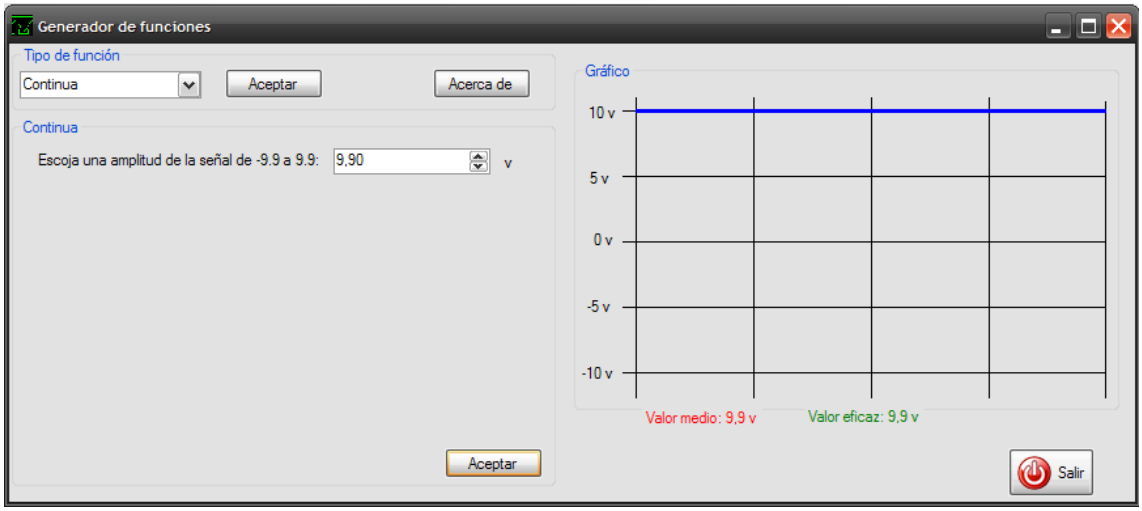


Figura 9.3: Función de amplitud constante de 9,9v. Aspecto gráfico del programa

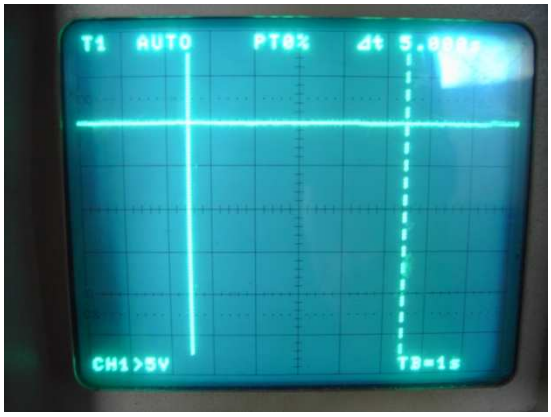


Figura 9.4: Función de amplitud constante de 9,9v. Osciloscopio en la salida.

9.1.2.- Ejemplo: Función de amplitud constante de -5v

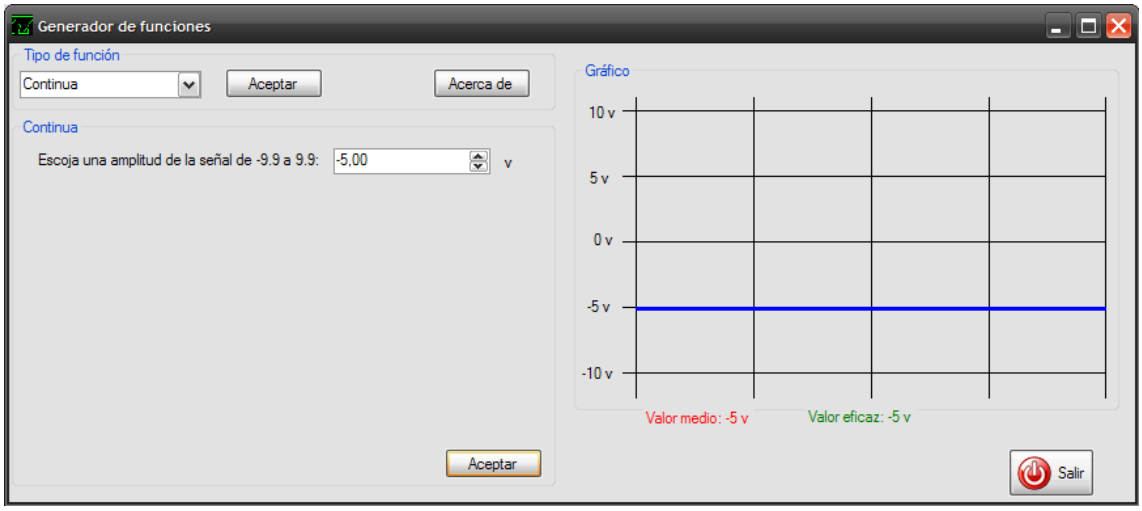


Figura 9.5: Función de amplitud constante de -5v. Aspecto gráfico del programa.

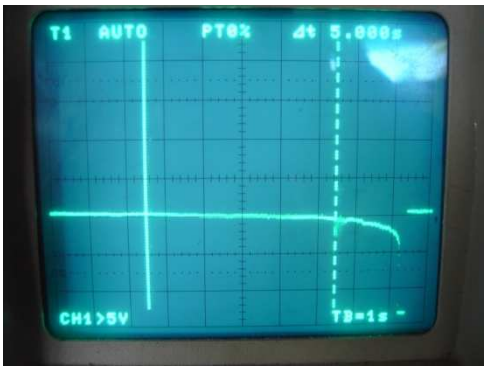


Figura 9.6: Función de amplitud constante de -5v. Osciloscopio en la salida

9.2.-Ejemplos de función cuadrada

La función cuadrada es útil para el análisis de respuestas transitorias o para accionamientos de motores PWM que según el ciclo de trabajo de dicha onda enviará más o menos potencia al motor. La función cuadrada puede variar la amplitud, el offset, la frecuencia y el ciclo de trabajo (dutycicle). En el caso de las funciones de ondas cuadradas lo que se hará es dibujar por pantalla dos periodos para poder observar el cambio entre un periodo y el siguiente.

9.2.1.- Ejemplo: Función cuadrada de 19,8 v de amplitud, -9,9 v de offset, 1Hz de frecuencia y un ciclo de trabajo del 50%

En este ejemplo hay que destacar que la onda estará la mitad en 9,9v (ON) y la otra mitad en 0,5 segundos en -9,9v (OFF). Esto provoca que la media sea cero (línea roja) y el valor eficaz 9,9v.

Figura 9.7: Función cuadrada de 19,8v de amplitud, -9,9v de offset, 1Hz de frecuencia y un ciclo de trabajo del 50%. Aspecto gráfico del programa.

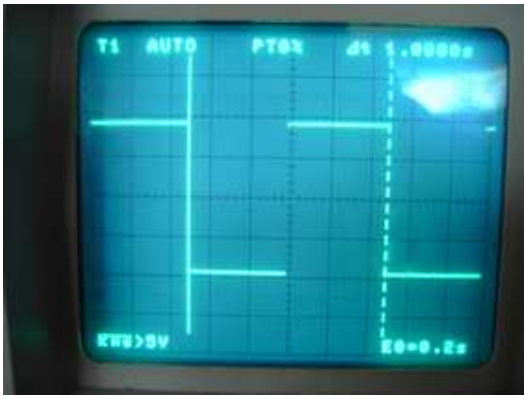


Figura 9.8: Función cuadrada de 19,8v de amplitud, -9,9v de offset, 1Hz de frecuencia y un ciclo de trabajo del 50%. Osciloscopio en la salida.

9.2.2.- Ejemplo: Función cuadrada de 10 v de amplitud, -4 v de offset, 50Hz de frecuencia y un ciclo de trabajo del 70%

En este caso se está un 70% en alto y la amplitud va de -4v a 6v. La frecuencia es de 50Hz, una frecuencia tan alta que para realizarla hay que bajar resolución. Esta resolución bajada se nota en el aspecto gráfico porque aparece una imagen verde debajo del gráfico.

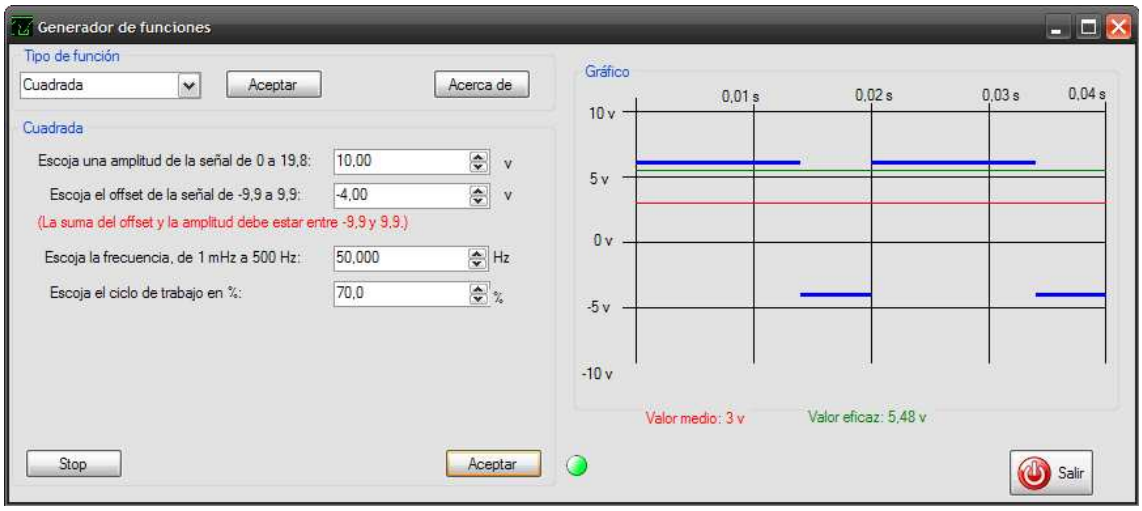


Figura 9.9: Función cuadrada de 10v de amplitud, -4v de offset, 50Hz de frecuencia y un ciclo de trabajo del 70%. Aspecto gráfico del programa.

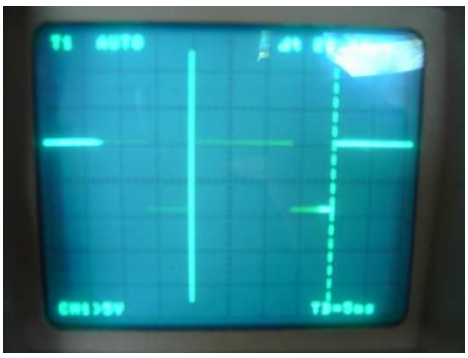


Figura 9.10: Función cuadrada de 10v de amplitud, -4v de offset, 50Hz de frecuencia y un ciclo de trabajo del 70%. Osciloscopio en la salida.

9.2.3.-Ejemplo: Función cuadrada de 5 v de amplitud, sin offset, 0,05Hz de frecuencia y un ciclo de trabajo del 10%

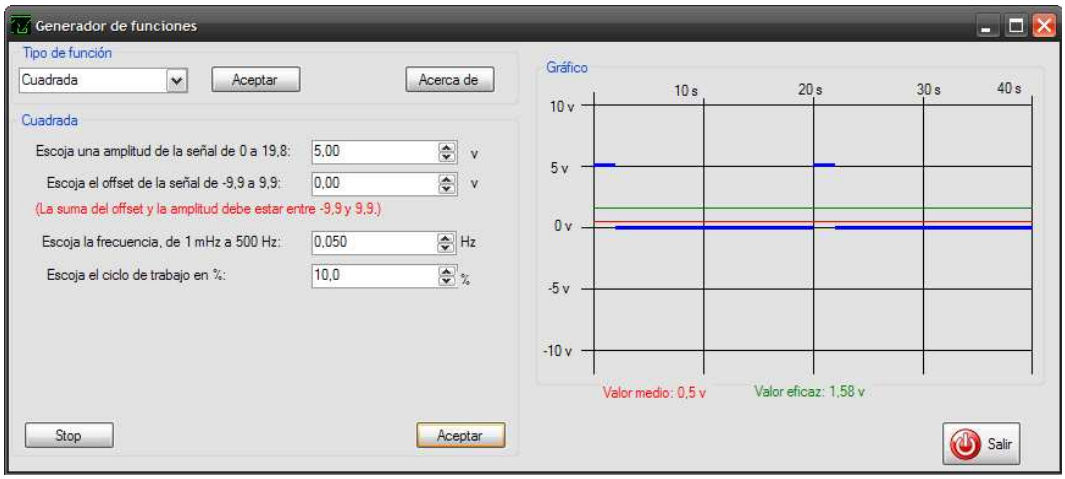


Figura 9.11: Función cuadrada de 5v de amplitud, sin offset, 0,05Hz de frecuencia y un ciclo de trabajo del 10%. Aspecto gráfico del programa.

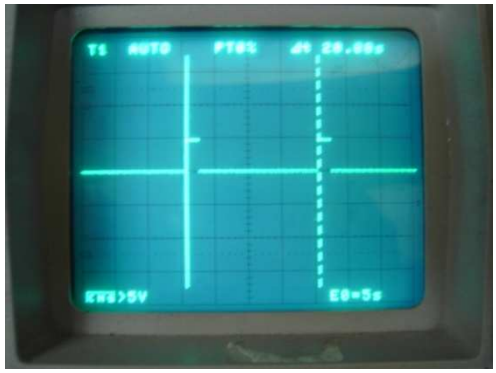


Figura 9.12: Función cuadrada de 5v de amplitud, sin offset, 0,05Hz de frecuencia y un ciclo de trabajo del 10%. Osciloscopio en la salida.

9.3.-Ejemplos de función triangular

La función triangular básicamente es empleada para medidas de nivel de disparo o estudios de linealidad, para ello es necesario disponer de la amplitud, el offset, el tiempo de subida y el tiempo de bajada.

9.3.1.-Ejemplo: Función triangular de 9,9 v de amplitud, sin offset y tiempo de subida y de bajada de 1s.

En este ejemplo, como en todos, es posible apreciar que la escala de tiempos del aspecto gráfico coincide con la dada en el osciloscopio. En este caso se observa que en ambas

figuras la señal triangular tiene un periodo (tiempo de subida + tiempo de bajada) de 2 segundos.

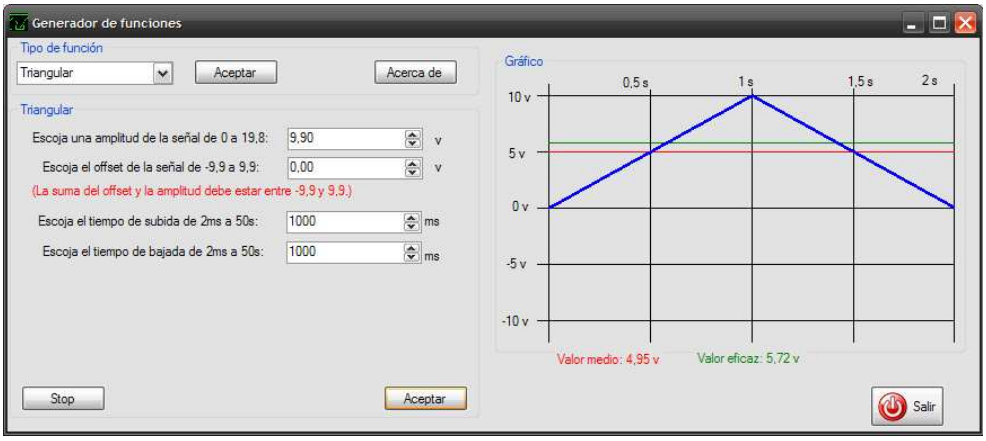


Figura 9.13: Función triangular de 9,9 v de amplitud, sin offset y tiempo de subida y de bajada de 1s. Aspecto gráfico del programa.

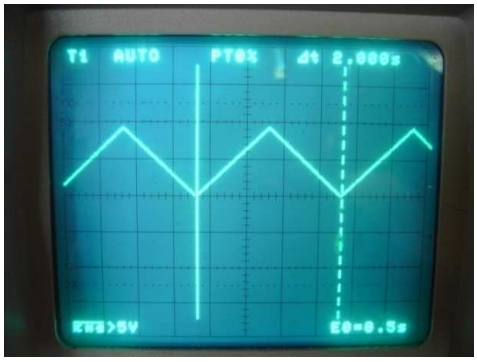


Figura 9.14: Función triangular de 9,9 v de amplitud, sin offset y tiempo de subida y de bajada de 1s. Osciloscopio en la salida.

9.3.2.-Ejemplo: Función triangular de 5 v de amplitud, -5v de offset, tiempo de subida de 100ms y tiempo de bajada de 1s.

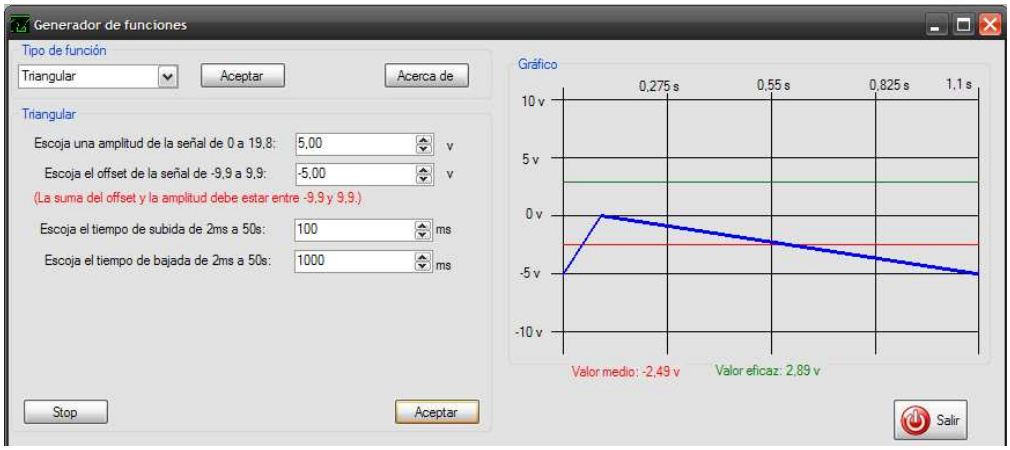


Figura 9.15: Función triangular de 5 v de amplitud, -5 de offset, tiempo de subida de 100ms y tiempo de bajada de 1s. Aspecto gráfico del programa.

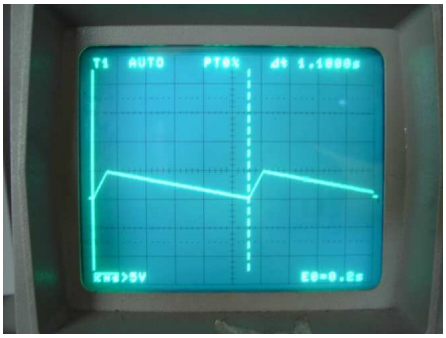


Figura 9.16: Función triangular de 5 v de amplitud, -5 de offset, tiempo de subida de 100ms y tiempo de bajada de 1s. Osciloscopio en la salida.

9.3.3.-Ejemplo: Función triangular de 4 v de amplitud, -2v de offset, tiempo de subida de 100ms y tiempo de bajada de 50ms.

Se puede observar que el valor medio es cero, ya que el rango es de -2 a 2v. Al ser unos tiempos de subida y de bajada muy bajos saldrá en el aspecto gráfico del programa la imagen verde como que se está perdiendo resolución.

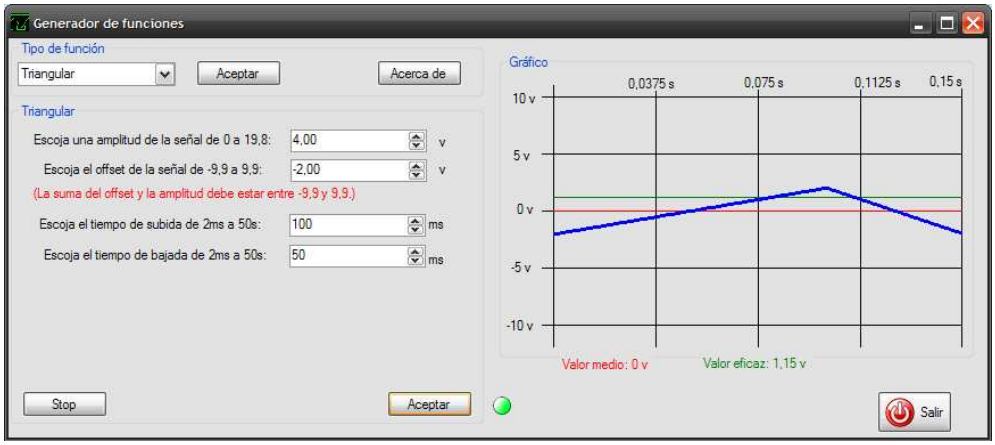


Figura 9.17: Función triangular de 4 v de amplitud, -2 de offset, tiempo de subida de 100ms y tiempo de bajada de 50ms. Aspecto gráfico del programa.

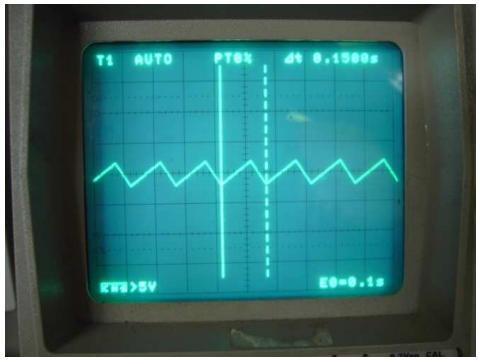


Figura 9.18: Función triangular de 4 v de amplitud, -2 de offset, tiempo de subida de 100ms y tiempo de bajada de 50ms. Osciloscopio en la salida

9.4.-Ejemplos de función diente de sierra

Esta onda es como la onda triangular, pero siendo uno de los tiempos (el de subida o el de bajada) igual a 0. La onda diente de sierra puede ser útil para el disparo de dispositivos de conmutación. En el caso de las funciones de ondas dientes de sierra lo que se hará es dibujar por pantalla dos periodos para poder observar el cambio entre un periodo y el siguiente.

9.4.1.-Ejemplo: Función diente de sierra de 4,5 v de amplitud, 0,5 v de offset, tiempo de subida de 500ms y tiempo de bajada nulo

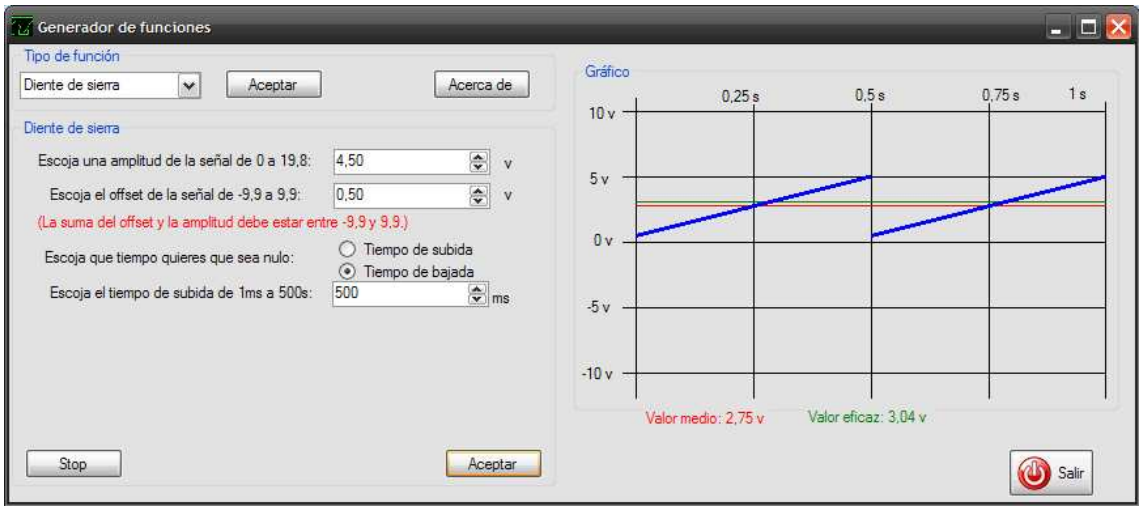


Figura 9.19: Función diente de sierra de 4,5 v de amplitud, 0,5 v de offset, tiempo de subida de 500ms y tiempo de bajada nulo. Aspecto gráfico del programa.

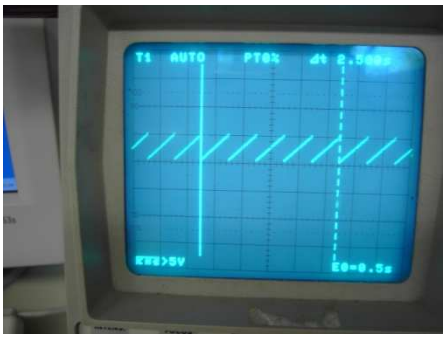


Figura 9.20: Función diente de sierra de 4,5 v de amplitud, 0,5 v de offset, tiempo de subida de 500ms y tiempo de bajada nulo. Osciloscopio en la salida.

9.4.2.-Ejemplo: Función diente de sierra de 1,5 v de amplitud, -5 v de offset, tiempo de bajada de 3s y tiempo de subida nulo

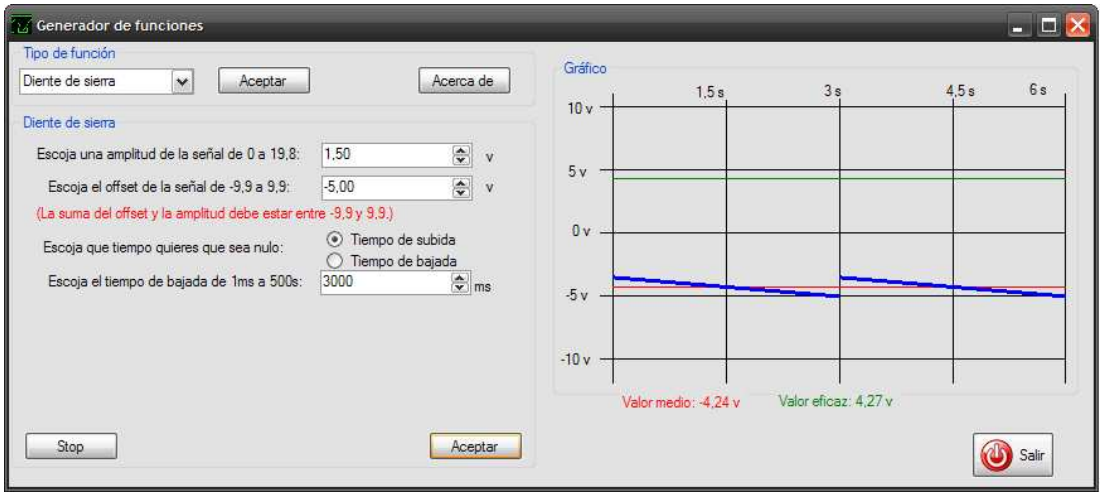


Figura 9.20: Función diente de sierra de 1,5 v de amplitud, -5 v de offset, tiempo de bajada de 3s y tiempo de subida nulo. Aspecto gráfico del programa.

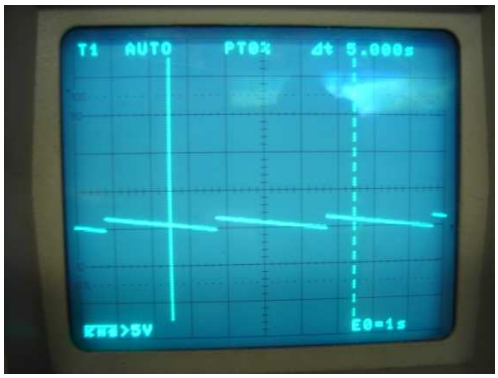


Figura 9.21: Función diente de sierra de 1,5 v de amplitud, -5 v de offset, tiempo de bajada de 3s y tiempo de subida nulo. Osciloscopio en la salida.

9.5.-Ejemplos de función trapezoidal

La función trapezoidal es una mezcla de funciones ya descritas. La función trapezoidal tiene como parámetros la amplitud, el offset, el tiempo de subida, el tiempo de bajada, el tiempo en alto (tiempo ON) y el tiempo en bajo (tiempo OFF). Si se juega con esos parámetros, se pueden obtener los diferentes tipos de ondas antes mencionadas. Por ejemplo, la onda cuadrada es posible hacerla con una onda trapezoidal de tiempo de subida y de bajada nulo. Una onda triangular es una onda trapezoidal de tiempo ON y tiempo OFF nulo, o la onda diente de sierra que es como una onda triangular pero sin el tiempo de subida o el de bajada. Pero son necesarias las funciones anteriores porque son muy usadas y es necesario un acceso rápido, fácil y cómodo para su uso. La onda trapezoidal además puede hacer otras formas diferentes como las que hay a continuación:

9.5.1.-Ejemplo: Función trapezoidal de 10 v de amplitud, -5v de offset, tiempo de subida, de bajada, en alto y en bajo de 1s

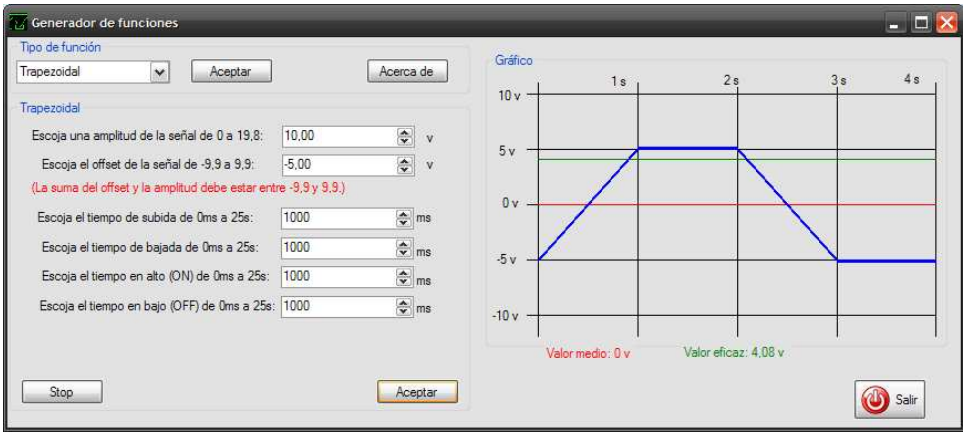


Figura 9.22: Función trapezoidal de 10 v de amplitud, -5v de offset, tiempo de subida, de bajada, en alto y en bajo de 1s. Aspecto gráfico del programa.

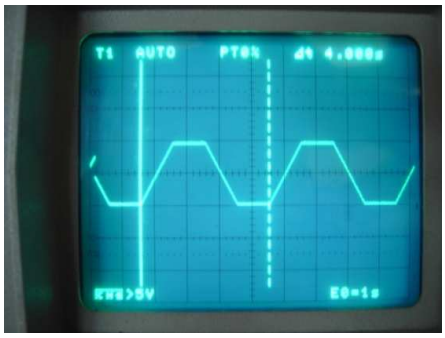


Figura 9.23: Función trapezoidal de 10 v de amplitud, -5v de offset, tiempo de subida, de bajada, en alto y en bajo de 1s. Osciloscopio en la salida.

9.5.2.-Ejemplo: Función trapezoidal de 5 v de amplitud, sin offset, tiempo de subida de 100ms, tiempo de bajada de 1s, tiempo en alto de 500ms y tiempo en bajo de 100ms

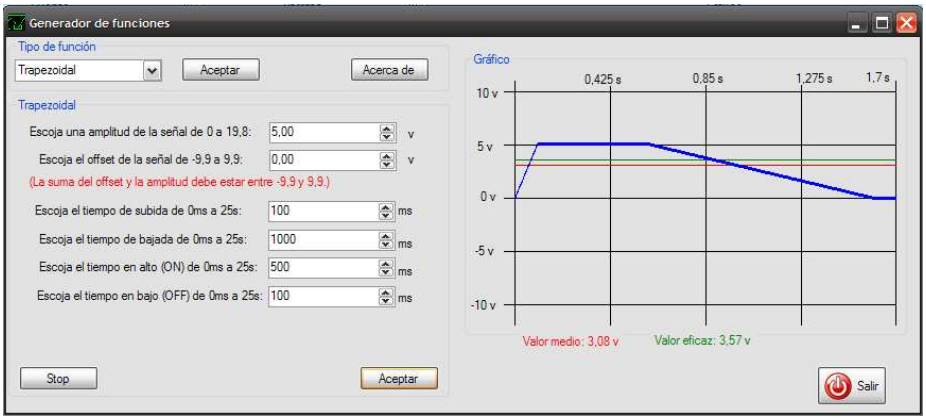


Figura 9.24: Función trapezoidal de 5 v de amplitud, sin offset, tiempo de subida de 100ms, tiempo de bajada de 1s, tiempo en alto de 500ms y tiempo en bajo de 100ms. Aspecto visual del programa.

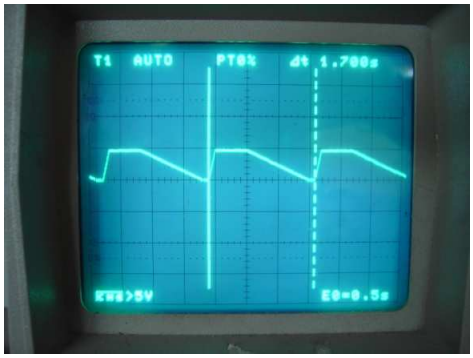


Figura 9.25: Función trapezoidal de 5 v de amplitud, sin offset, tiempo de subida de 100ms, tiempo de bajada de 1s, tiempo en alto de 500ms y tiempo en bajo de 100ms. Osciloscopio en la salida.

9.5.3.-Ejemplo: Función trapezoidal de 5 v de amplitud, -5v de offset, tiempo de subida y tiempo bajo nulo y tiempo de bajada y en alto de 1s

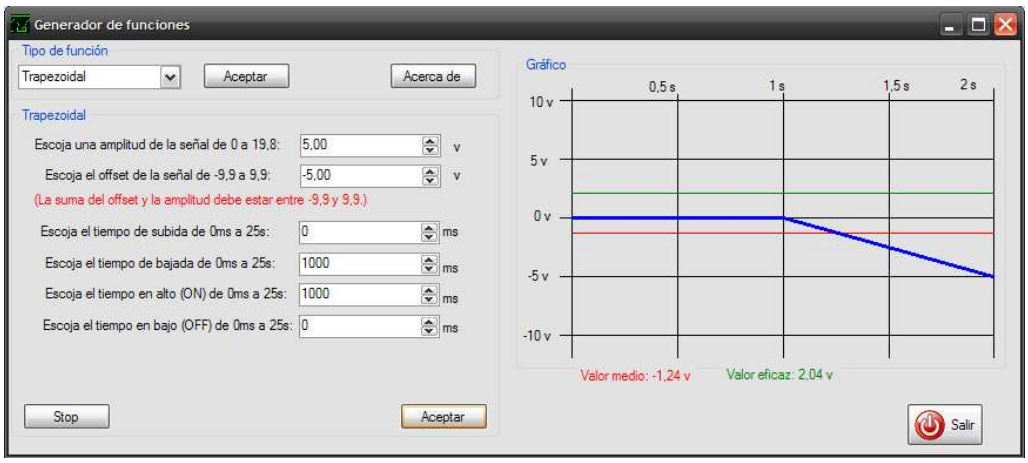


Figura 9.26: Función trapezoidal de 5 v de amplitud, -5v de offset, tiempo de subida y tiempo bajo nulo y tiempo de bajada y en alto de 1s. Aspecto visual del programa.

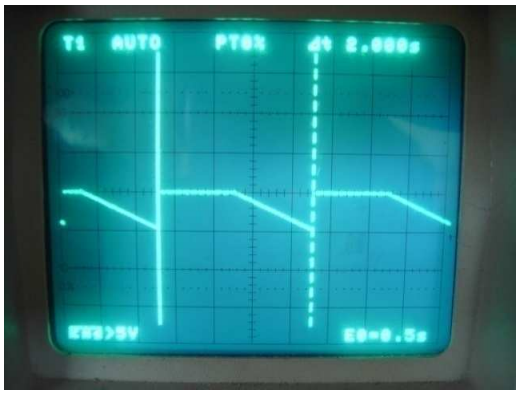


Figura 9.27: Función trapezoidal de 5 v de amplitud, -5v de offset, tiempo de subida y tiempo bajo nulo y tiempo de bajada y en alto de 1s. Osciloscopio en la salida.

9.6.- Ejemplos de función sinusoidal

La función sinusoidal se usa para la obtención de respuesta temporal. También puede usarse por pequeños aparatos de corriente alterna de voltaje y frecuencia bajos. La función sinusoidal contiene las siguientes variables para crearse: la amplitud, el offset, la frecuencia y el desfase. El desfase no tiene importancia en un sistema de alimentación continua, pero sí en el momento de encendido. Si el desfase es de 90º (función coseno), el encendido es instantáneo y no progresivo como una sinusoidal de 0º. Véase algún ejemplo:

9.6.1.-Ejemplo: Función sinusoidal de 5 v de amplitud, sin offset, frecuencia de 2Hz y desfase de 0º

En esta función sinusoidal se puede ver que la media es de 0v (ya que el rango es de -5 a 5v) y el valor eficaz es de 3,54v, ya que la amplitud es 5v y, como es una función sinusoidal, el valor eficaz es:

$$V_{ef} = \frac{A}{\sqrt{2}} = \frac{5}{\sqrt{2}} = 3{,}54\ V$$

Figura 9.28: Fórmula del valor eficaz de una onda sinusoidal

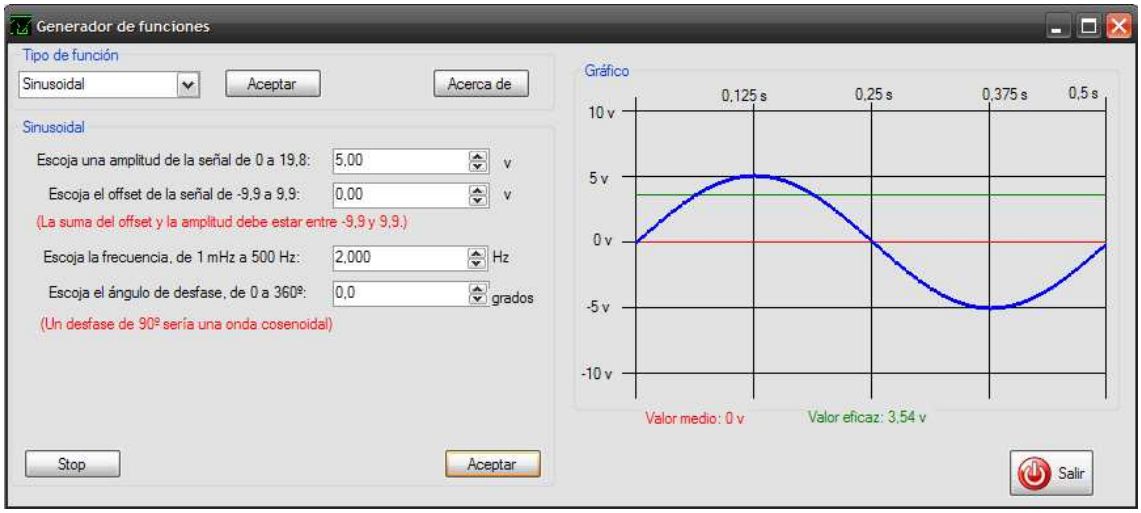


Figura 9.29: Función sinusoidal de 5 v de amplitud, sin offset, frecuencia de 2Hz y desfase de 0º. Aspecto visual del programa.

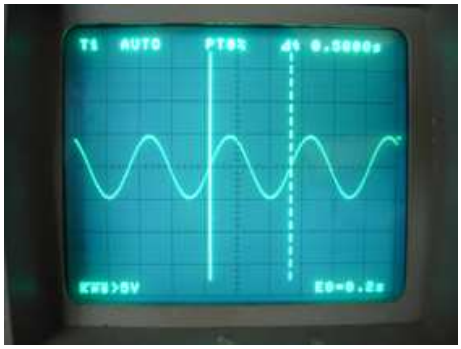


Figura 9.30: Función sinusoidal de 5 v de amplitud, sin offset, frecuencia de 2Hz y desfase de 0º. Osciloscopio en la salida.

9.6.2.-Ejemplo: Función sinusoidal de 2 v de amplitud, 3v de offset, frecuencia de 0,2 Hz y desfase de 90º

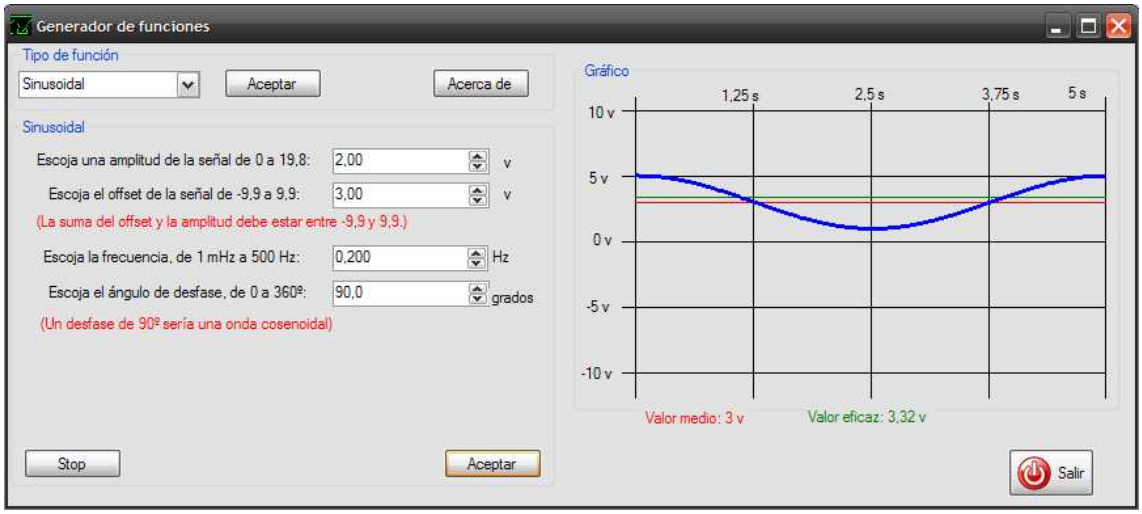


Figura 9.31: Función sinusoidal de 2 v de amplitud, 3v de offset, frecuencia de 0,2 Hz y desfase de 90º. Aspecto visual del programa.

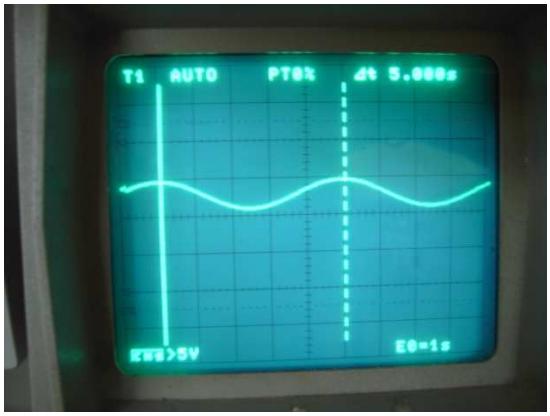


Figura 9.32: Función sinusoidal de 2 v de amplitud, 3v de offset, frecuencia de 0,2 Hz y desfase de 90º. Osciloscopio en la salida.

9.7.-Ejemplos de función irregular

La función irregular da valores al azar dentro del rango marcado; por ello los valores nunca coincidirán. Los valores del aspecto gráfico y del osciloscopio no son coincidentes porque las fotos del osciloscopio se hicieron después de un rato de ejecución del programa; en cambio, los valores del programa son los primeros de la ejecución. Los valores al azar pueden servir para testear una máquina de entrada analógica. El periodo de la onda es de 20000 puntos, es decir, el punto 20001º coincidirá siempre con el 1º. La media y valor eficaz calculados en el programa corresponde a la media y valor eficaz de los 20000 puntos de cada periodo.

9.7.1.-Ejemplo: Función irregular de 5 v de amplitud, sin offset y velocidad de 10 muestras por segundo

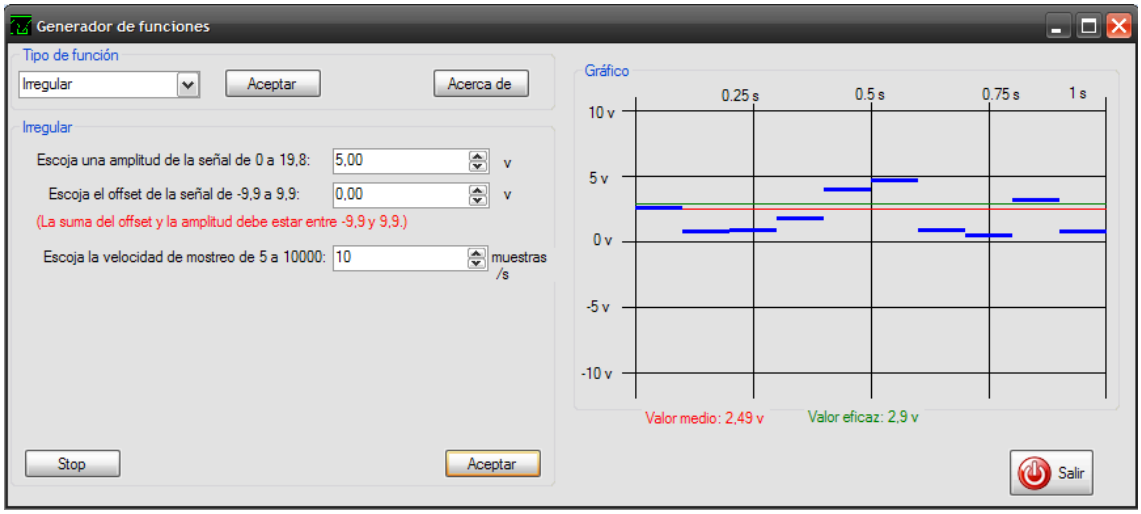


Figura 9.33: *Función irregular de 5 v de amplitud, sin offset y velocidad de 10 muestras por segundo. Aspecto gráfico del programa.*

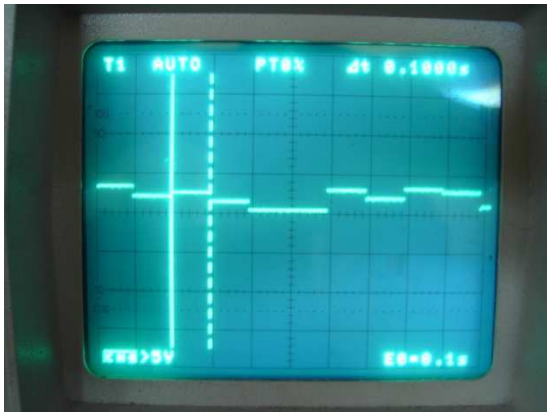


Figura 9.34: *Función irregular de 5 v de amplitud, sin offset y velocidad de 10 muestras por segundo. Osciloscopio en la salida.*

9.7.2.-Ejemplo: Función irregular de 10 v de amplitud, -5v de offset y velocidad de 30 muestras por segundo

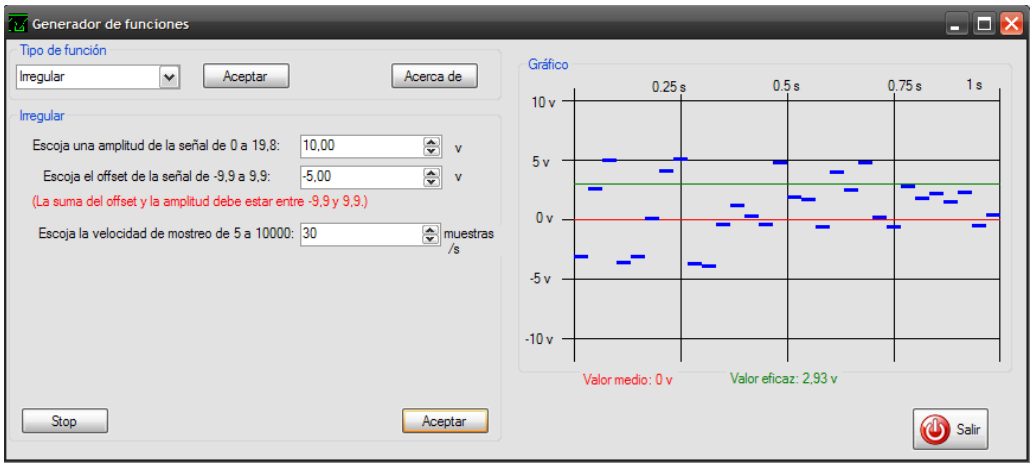


Figura 9.35: Función irregular de 10 v de amplitud, -5 de offset y velocidad de 30 muestras por segundo. Aspecto gráfico del programa.

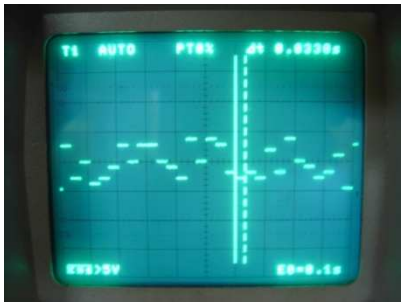


Figura 9.36: Función irregular de 10 v de amplitud, -5 de offset y velocidad de 30 muestras por segundo. Osciloscopio en la salida.

9.7.3.- Función irregular de 2 v de amplitud, -7 de offset y velocidad de 10000 muestras por segundo

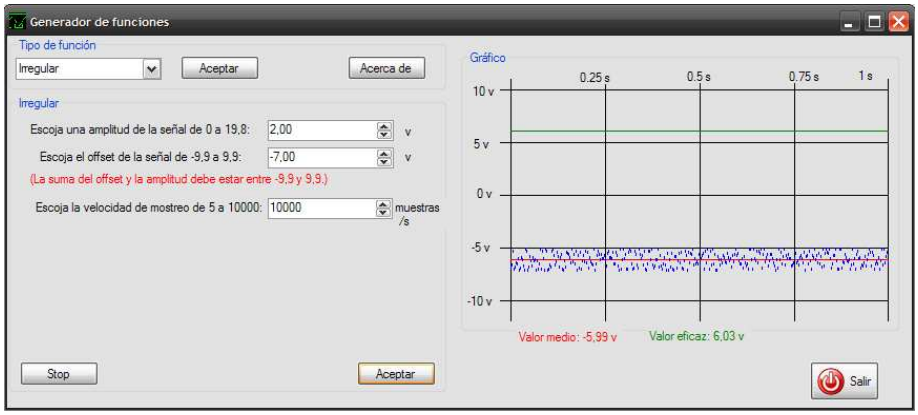


Figura 9.37: Función irregular de 2 v de amplitud, -7 de offset y velocidad de 10000 muestras por segundo. Aspecto gráfico del programa.

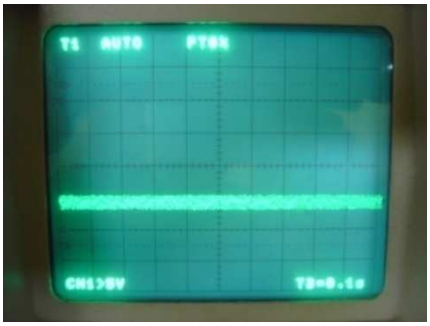


Figura 9.38: Función irregular de 2 v de amplitud, -7 de offset y velocidad de 10000 muestras por segundo. Osciloscopio en la salida.

9.7.4.-Ejemplo: Función irregular de 19,8 v de amplitud, -9,9v de offset y velocidad de 10000 muestras por segundo

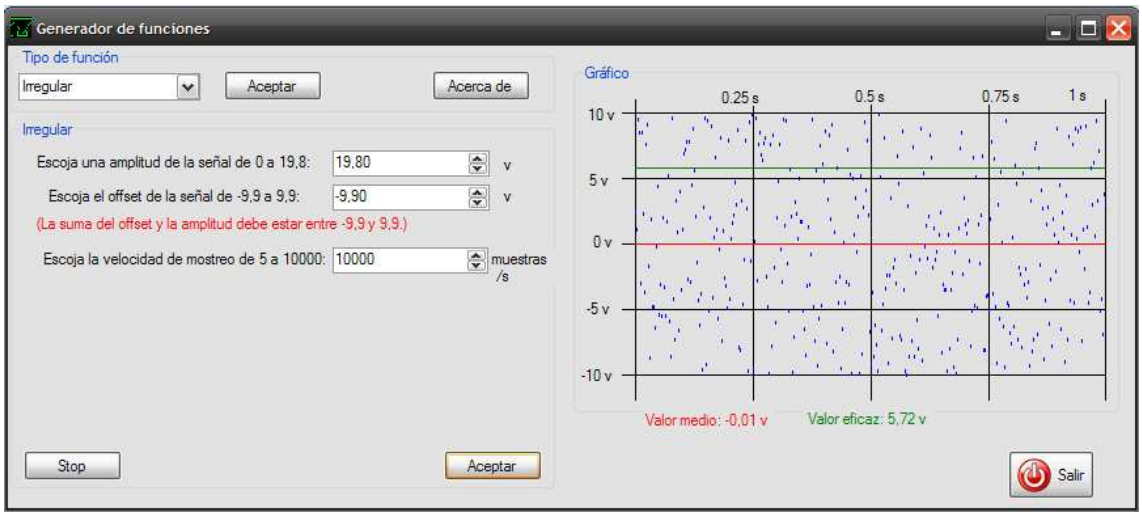


Figura 9.39: Función irregular de 19,8 v de amplitud, -9,9 de offset y velocidad de 10000 muestras por segundo. Aspecto gráfico del programa.

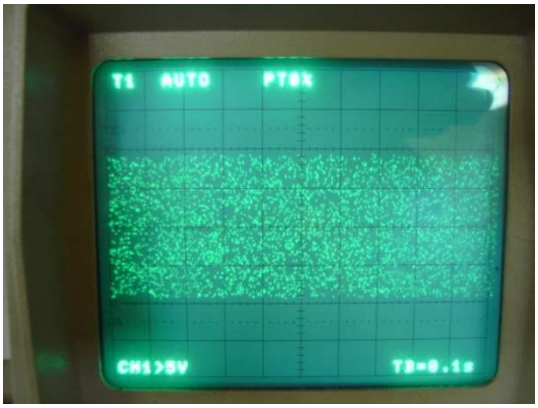


Figura 9.40: Función irregular de 19,8 v de amplitud, -9,9 de offset y velocidad de 10000 muestras por segundo. Osciloscopio en la salida.

9.8.-Ejemplo de función personalizada

La función personalizada es un tipo de onda donde el usuario puede escoger su forma mediante el "dibujo" de la función. Así el usuario puede utilizarlo para el uso que sea necesario y conseguir forma de ondas que otros dispositivos convencionales no pueden crear, ya que no siguen una fórmula matemática específica. En este tipo de función es necesario "dibujar" la onda, dar la frecuencia y controlar si los cambios son pulsantes o progresivos. Véase un ejemplo con los mismos valores, el mismo periodo; pero de tipo de cambio pulsante primeramente y después de tipo de cambio progresivo.

9.8.1.-Ejemplo: Función personalizada de periodo 1s y tipo de cambio pulsante

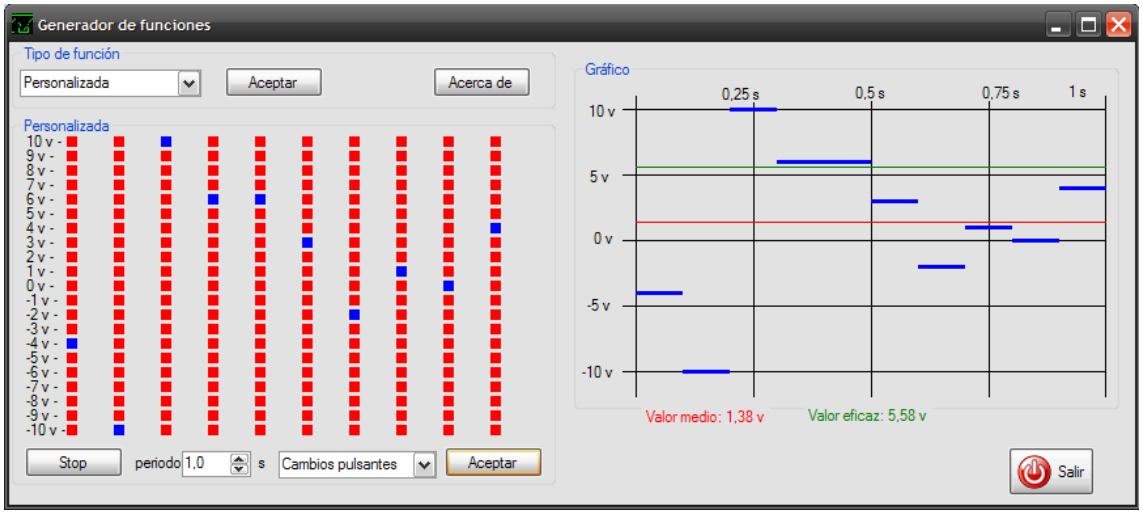


Figura 9.41: Función personalizada de periodo 1s y tipo de cambio pulsante. Aspecto gráfico del programa.

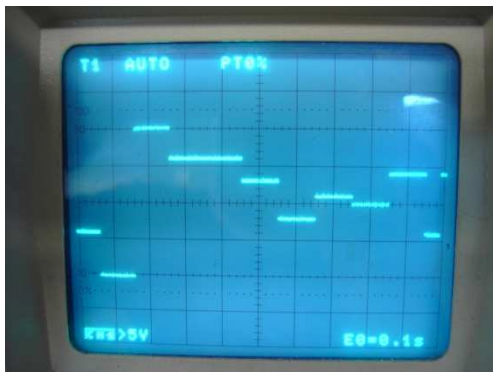


Figura 9.42: Función personalizada de periodo 1s y tipo de cambio pulsante. Osciloscopio en la salida.

9.8.2.-Ejemplo: Función personalizada de periodo 1s y tipo de cambio progresivo

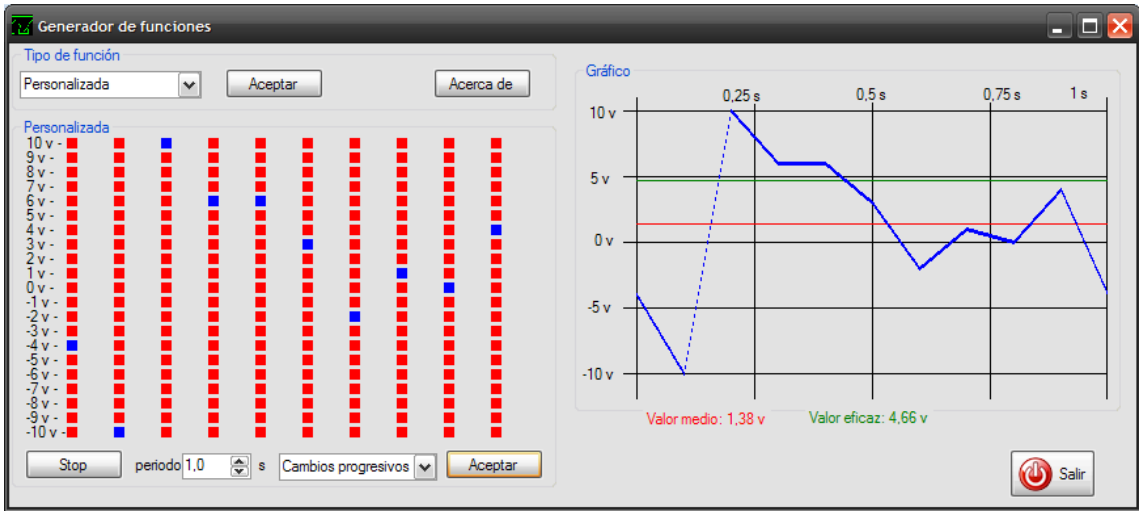


Figura 9.43: Función personalizada de periodo 1s y tipo de cambio pulsante. Aspecto gráfico del programa.

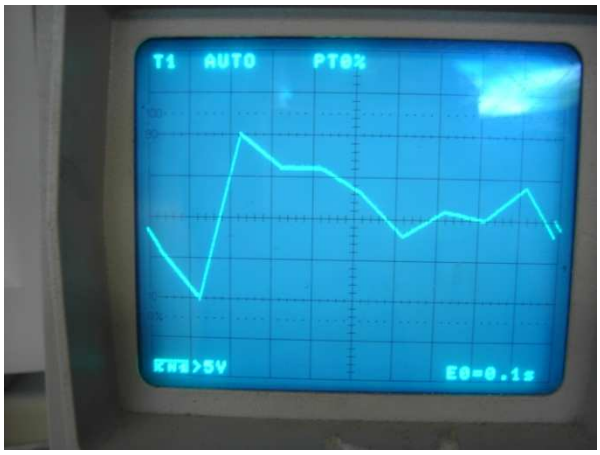


Figura 9.44: Función personalizada de periodo 1s y tipo de cambio pulsante. Osciloscopio en la salida.

10.- Manual de instalación y uso

10.1.- Instalación de la tarjeta PCI DAS6025 (InstalCal32)

Para el funcionamiento de este generador de funciones es necesaria la instalación de la tarjeta PCI DAS6025 correctamente. Para ello se utilizará el software que se encuentra en el CD de la tarjeta llamado InstalCal32. A continuación, se abrirá la aplicación y saldrá una lista de las tarjetas instaladas. Inicialmente la tarjeta de pruebas (una tarjeta virtual) será la única tarjeta instalada.

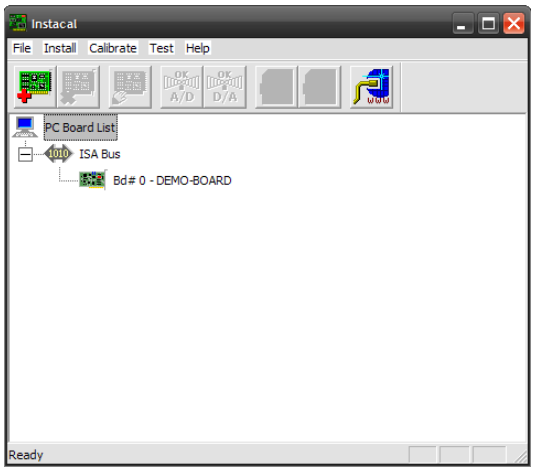


Figura 10.1: Instalación de la tarjeta PCI DAS6025 paso 1 de 2.

En primer lugar, hay que cambiar la tarjeta virtual de número, por ejemplo, el número 2. Hacer click derecho al icono de la tarjeta y coger la opción *Change Board#...* y seleccionar el 2. El siguiente paso es añadir una nueva tarjeta apretando el botón siguiente:

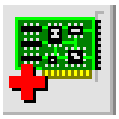


Figura 10.2: Icono para añadir e instalar una nueva tarjeta.

En la lista de tarjetas aparecerá la tarjeta que se utilizará si ya ha sido puesta en la ranura PCI. Automáticamente se añadirá como tarjeta número 0 (necesario para que funcione esta aplicación). Ya se encuentra instalada la tarjeta. En el programa InstalCal32 de programación de tarjetas se encuentra ahora la tarjeta PCI DAS6025 como tarjeta número 0 y la tarjeta virtual como tarjeta número 2.

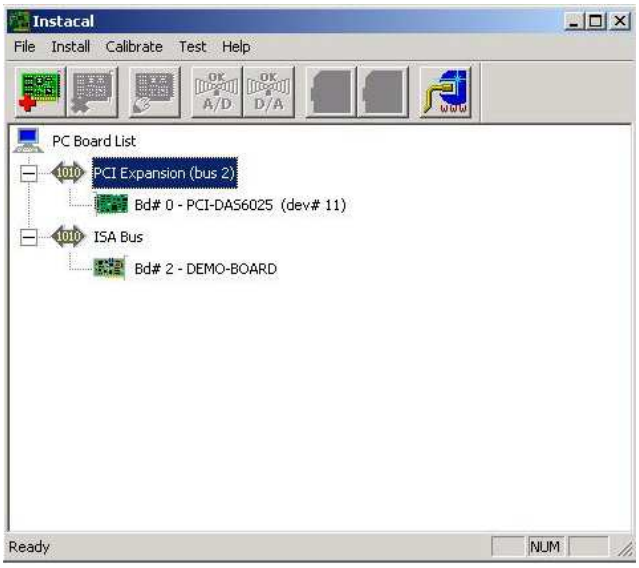


Figura 10.3: Instalación de la tarjeta PCI DAS6025 paso 2 de 2.

Ahora se debe configurar la tarjeta PCI-DAS6025 con el siguiente botón:

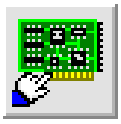


Figura 10.4: Icono para configurar una tarjeta.

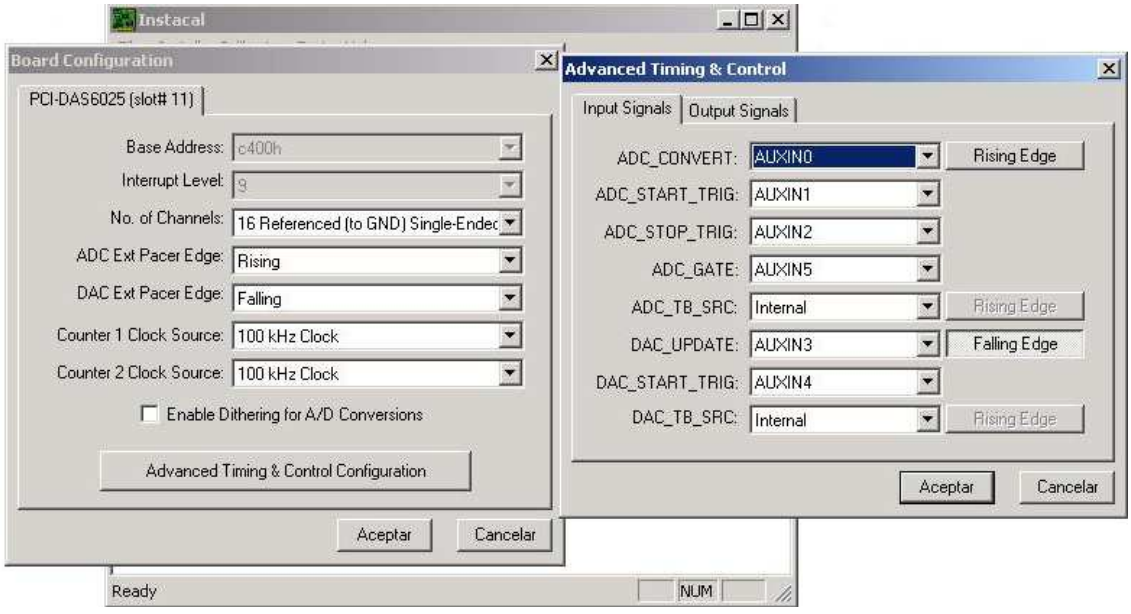


Figura 10.5: InstalCal32. Configuración de la tarjeta PCI DAS6025.

Esto son los parámetros con los que se configurará la tarjeta. Así se consigue que la tarjeta ya esté instalada y configurada para el funcionamiento correcto del programa.

10.2.- Instalación del software *Generador de funciones*

Para instalar el software solo es necesario ejecutar el instalador, y para ello se buscará el archivo *setup.exe* dentro del CD.

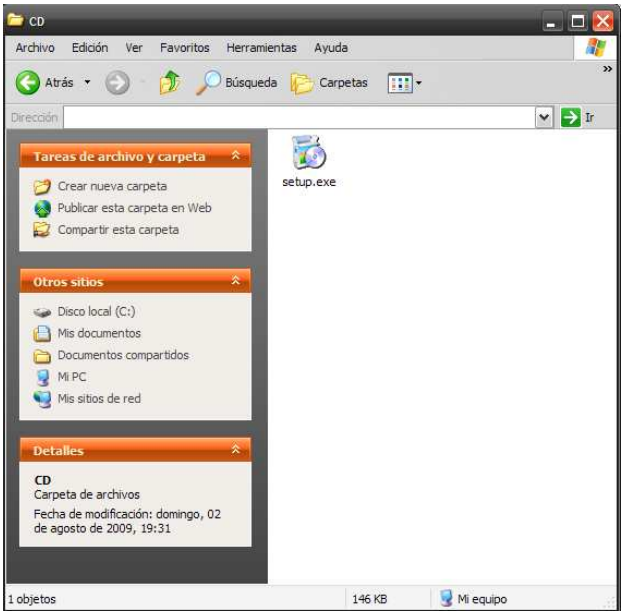


Figura 10.6: Archivo que hay que ejecutar, *setup.exe, para iniciar la instalación de nuestra aplicación*.

Primero aparecerá una pantalla de bienvenida a la instalación del Generador de funciones y la lectura de la ley de derechos de autor y cómo respetar dicha ley. Para continuar hay que apretar el botón de 'siguiente'.

Figura 10.7: Instalación Generador de funciones paso 1 de 4.

La siguiente ventana corresponde a datos del Generador de funciones como cuál es el autor, que se trata de un proyecto final de carrera de la Universitat Rovira i Virgili (URV), etc., además de una pequeña explicación de la funcionalidad de esta aplicación

y también las características que debe cumplir el computador para su correcto funcionamiento.

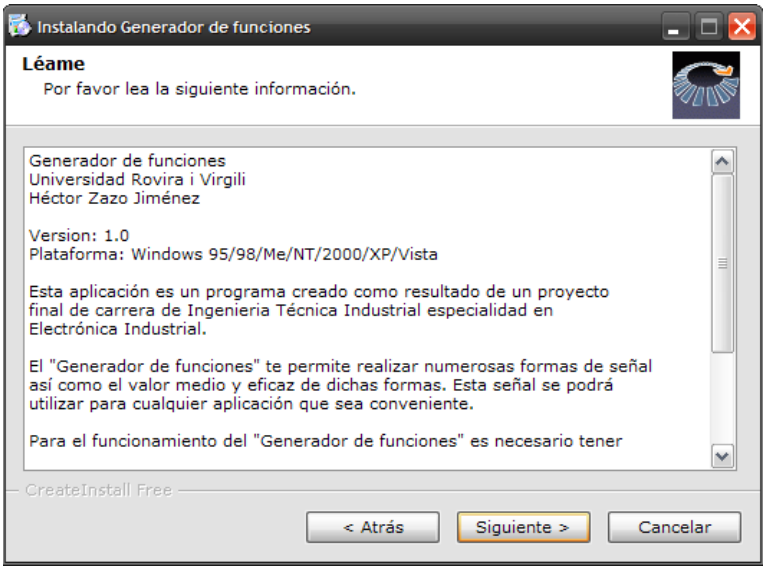


Figura 10.8: Instalación Generador de funciones paso 2 de 4.

El siguiente paso consiste en guardar la aplicación en el directorio que se quiera. Por defecto, el directorio será *C:\Archivos de programa\Generador de funciones*. También se puede observar que la instalación del software tan sólo utiliza 22,51 MB de disco duro, 79,70KB del programa y lo demás de la plataforma .NET Framework. El programa ocupa muy poco, es decir, si ya tiene la plataforma instalada, la instalación del Generador de funciones ocupará menos de un cuarto de canción.

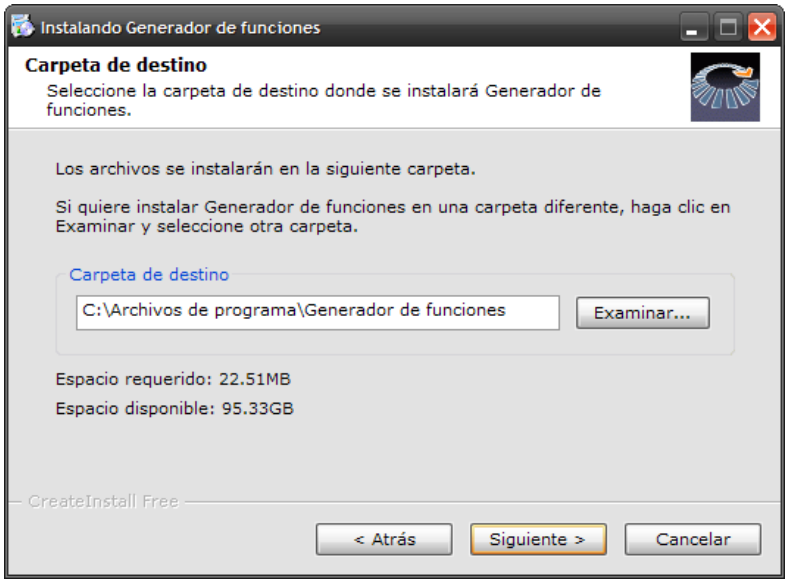


Figura 10.9: Instalación Generador de funciones paso 3 de 4.

Después de darle a ‘siguiente’ empezará la instalación, que durará pocos segundos. Una vez acabada la instalación aparecerá una ventana como la siguiente para asegurar al usuario que la instalación ha sido un éxito.

Figura 10.10: Instalación Generador de funciones paso 4 de 4.

La instalación del software ya se ha llevado a cabo. Ahora empezará la extracción para instalar la plataforma .NET Framework y seguidamente la instalación de ella. Si ya se dispone de la plataforma, se puede cancelar la instalación.

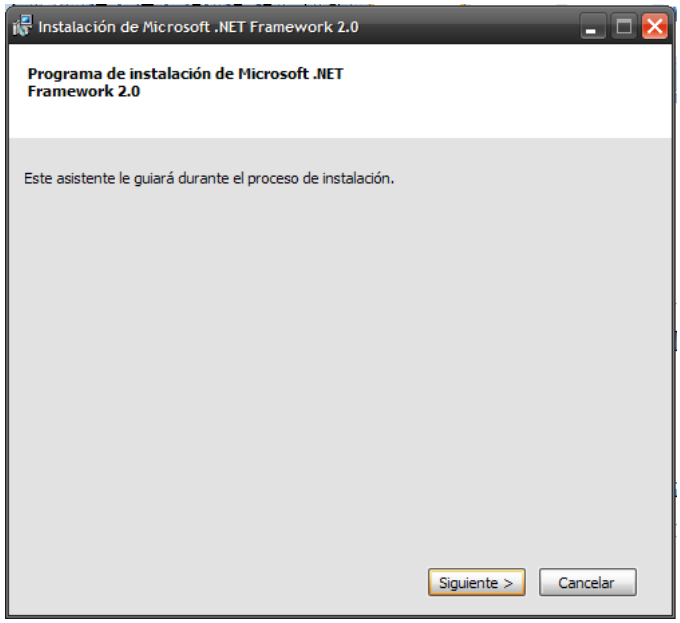


Figura 10.11: Instalación de la plataforma .NET Framework paso 1 de 3.

Presionar ‘siguiente’ para continuar la instalación de la plataforma o ‘cancelar’ para no hacerlo. Si se presiona ‘siguiente’ se irá a la siguiente ventana, que son los términos de la licencia del producto. Hay que leerlos, imprimirlos si cree conveniente y aceptarlos haciendo *click* en la casilla creada para ese fin. Una vez hecho se proseguirá dándole al botón de ‘instalar’.

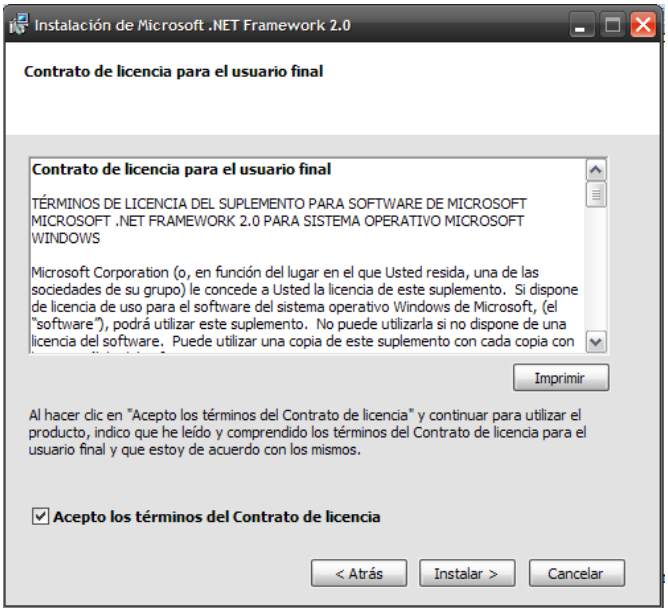


Figura 10.12: Instalación de la plataforma .NET Framework paso 2 de 3.

Al aceptar los términos del contrato de licencia y darle a instalar, empezará la instalación de la plataforma .NET Framework. Después de la instalación se confirmará que la instalación ha sido correcta. Si ya se tiene instalada una versión anterior de la plataforma .NET Framework se permitirá instalar esta versión, pero se podrá trabajar con la versión anterior.

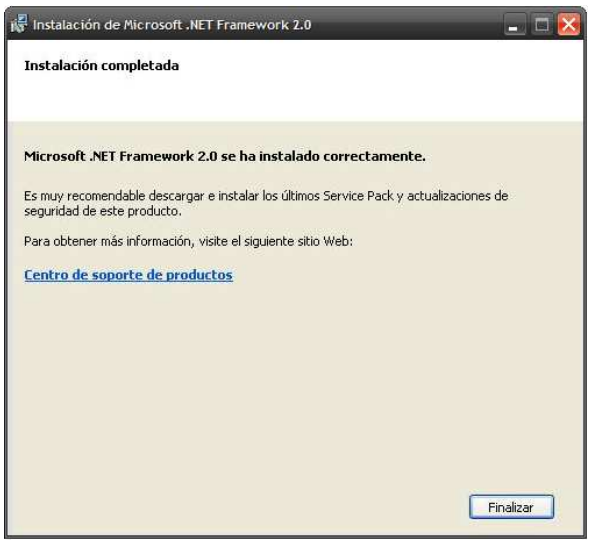


Figura 10.13: Instalación de la plataforma .NET Framework paso 3 de 3.

Ya se tiene la instalación acabada.

10.3.- Uso de la aplicación *Generador de funciones*

Una vez instalado el programa y la plataforma .NET Framework se creará un acceso directo de la aplicación en el Escritorio:

Figura 10.14: Acceso directo de nuestro programa que encontraremos en el Escritorio.

Al ejecutarlo, se abrirá una ventana como la siguiente:

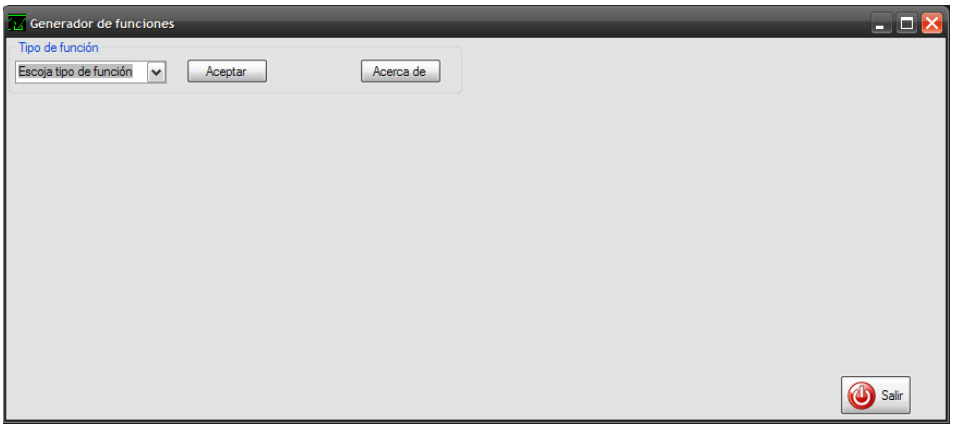


Figura 10.15: Vista inicial del programa.

El programa dispone en la ventana el botón de cerrarse, maximizarse o minimizarse. Inicialmente el programa ya dispone de las medidas para que aparezca todo el programa por pantalla. Inicialmente se dispone de un selector y tres botones. El botón 'salir' sirve para cerrar el programa, es interesante hacerlo por el botón ese y no por la cruz ya que se asegura borrar el espacio reservado en memoria. Se dispone de un botón 'Aceptar' para seleccionar la onda que se ha clickado en el selector. Si no se ha seleccionado ninguna o se ha inventado alguna, aparecerá una ventana de error. También se dispone del botón 'Acerca de' que dará información del programa como se ve en la siguiente imagen:

Figura 10.16: Ventana Acerca de.

Si se ha seleccionado algún tipo de onda de la lista y se ha apretado al botón ‘Aceptar’ aparecerá un cuadro con unos parámetros que se deben rellenar para crear la onda. Es importante no dejar el parámetro de amplitud a 0, ya que, si no, la señal de salida será siempre nula. Junto a los parámetros aparecerá un nuevo botón de ‘Aceptar’ para confirmar los parámetros seleccionados. Véase la siguiente imagen que es el ejemplo visual de lo explicado:

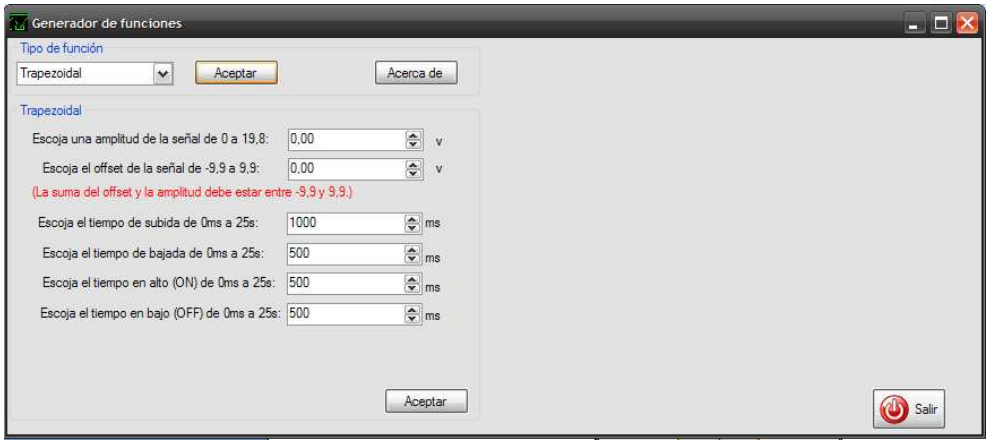


Figura 10.17: Parámetros para la onda trapezoidal.

En el caso de seleccionar la función personalizada, a parte de seleccionar un par de parámetros, hay que dibujar una onda mediante unos cuadraditos. El funcionamiento es apretar el cuadradito verde que le interesa para formar la onda a su gusto (en la figura 8.2 se puede ver un ejemplo). Una vez “dibujada” la onda y seleccionados los parámetros, como en los otros tipos de función, hay que apretar el botón ‘Aceptar’ para que inicie la forma de onda configurada con los parámetros. Esta onda será extraída del ordenador por los *pins* 36 y 37 de la tarjeta. También, justo al apretar el botón ‘Aceptar’, se dibujará en la parte derecha del programa el periodo de la onda resultante. Además en el mismo dibujo se dirá el valor medio y eficaz de la onda.

Figura 10.18: Ejemplo del programa con la onda sinusoidal.

Una vez funcionando se puede presionar el botón 'Stop' que ha aparecido abajo a la izquierda para parar de generar la función. Después se puede seleccionar otro tipo de onda o cambiar parámetros y seguir los pasos anteriores. Cuando se haya acabado, se puede cerrar el programa con el botón 'Salir'.

10.4.- Desinstalación del software

Una vez acabado de usar el Generador de funciones se puede dejar el programa instalado, ya que ocupa muy poco espacio en el disco duro, o desinstalarlo. Para desinstalarlo hay que buscar en el directorio del programa el archivo *unistall.exe*.

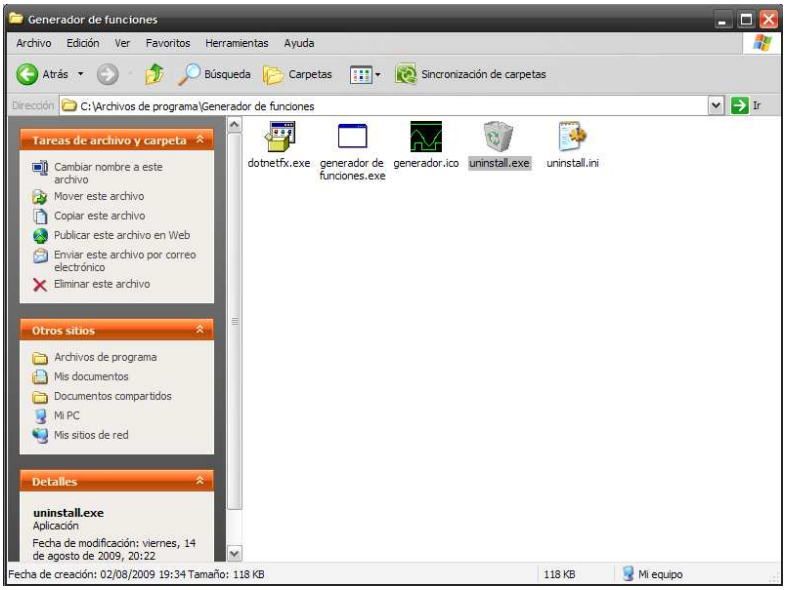


Figura 10.19: Archivo que hay que ejecutar, uninstall.*exe, para iniciar la desinstalación de nuestra aplicación*.

Al apretar el icono aparecerá la pantalla de desinstalación. Apretando ‘Siguiente’ acabará de desinstalarse.

Figura 10.20: Desinstalación paso 1 de 2.

Figura 10.21: Desinstalación paso 2 de 2.

11.- Presupuesto

El presupuesto necesario para el desarrollo de este proyecto es:

Descripción	Unidades	Precio unitario	Precio total
Tarjeta PCI DAS6025	1 u.	519$ ó 365,19€	365,19€
Cable C100HD50-3	1 u.	57,50£ ó 67,12€	67,12€
Tarjeta CIO mini-50	1 u.	60£ ó 70,04€	70,04€
Cable de conexión de 1 polo	0,6 m.	0,75€	0,45€
Clavija HRL Definition HRL-BANANA	2 u.	3,70€	7,40€
Horas de desarrollo del software	240 h	35€/h	8400,00€
TOTAL			8910,20€

Para amortizar el coste del desarrollo del proyecto y previniendo 1000 ventas de este generador de funciones + tarjeta PCI DAS6025 su precio de venta sería de:

Descripción	
Coste de la Tarjeta PCI DAS6025 por unidad	365,19€
Coste del material por unidad	145,01€
Coste de amortización de creación del software si hacemos 1000 ventas	8,40€
SUBTOTAL	518,60€
Si queremos obtener un beneficio del 10% por unidad	51,86€
TOTAL	570,46€

En el caso de que se disponga ya de la tarjeta y sólo se quieran el software y las conexiones:

Descripción	
Coste del material por unidad	145,01€
Coste de amortización de creación del software si hacemos 1000 ventas	8,40€
Si queremos obtener un beneficio del 10% por unidad	15,34€
TOTAL	168,75€

12.- Hoja de características

Las características del generador de funciones son las siguientes:

Tipo de ondas	Constante, cuadrada, triangular, diente de sierra, trapezoidal, sinusoidal, irregular y personalizada
Rango	-10V a 10V
Frecuencia	De 1 mHz a 500 Hz
Resolución	4,89 mV
Características	Parámetros de ondas variables
Impedancia de salida	500 mΩ como máximo
Corriente de conducción	±5 mA

Constante	
Parámetros variables	Amplitud
Resolución	4,89 mV
Error	Máximo de 2,5 mV
Cuadrada	
Parámetros variables	Amplitud, offset, frecuencia, ciclo de trabajo
Resolución	4,89 mV y 100µs
Error	Máximo de 2,5 mV
Triangular	
Parámetros variables	Amplitud, offset, tiempo de bajada, tiempo de subida
Resolución	4,89 mV y 100µs
Error	Máximo de 2,5 mV
Diente de sierra	
Parámetros variables	Amplitud, offset, tiempo de bajada o tiempo de subida
Resolución	4,89 mV y 100µs
Error	Máximo de 2,5 mV
Trapezoidal	
Parámetros variables	Amplitud, offset, tiempo de bajada, tiempo en alto, tiempo en bajo, tiempo de subida
Resolución	4,89 mV y 100µs
Error	Máximo de 2,5 mV
Sinusoidal	
Parámetros variables	Amplitud, offset, frecuencia, desfase
Resolución	4,89 mV y 100µs
Error	Máximo de 2,5 mV
Personalizada	
Parámetros variables	Dibujo de la onda, tipo de cambio, periodo
Resolución	4,89 mV y 100µs
Error	Máximo de 2,5 mV
Irregular	
Parámetros variables	Amplitud, offset, valores por segundo
Resolución	4,89 mV y 100µs
Error	Sin error ya que el valor es aleatorio

El software necesita los siguientes requisitos del sistema:

Sistema operativo	Microsoft Windows 95, 98, 2000, NT, XP, Vista.
Procesador	Pentium III a 1,5GHz, se recomienda Pentium IV a 2GHz
Disco duro	Mínimo de 22 MBytes
Memoria RAM	256 MB. Se recomienda mínimo 512MB para el correcto funcionamiento
Tarjeta gráfica	Indiferente
Tarjetas de apoyo	PCI DAS6025
Otro software necesario	Plataforma .NET Framework, disponible junto con el instalador del generador de funciones

13.- Conclusiones

Aunque hay muchos generadores de señales en el mercado, todos tienen algún inconveniente. Muchos generadores carecen de diferentes tipos de funciones y otros tienen un precio muy elevado. Con este software se ha pretendido conseguir un generador de funciones que permita crear varios tipos de funciones a un precio muy bajo.

El generador de funciones permite realizar señales cuadradas, continuas, triangulares, sinusoidales, dientes de sierra, trapezoidales, aleatorias e, incluso, señales dibujadas por el propio usuario, ya que la aplicación tiene un aspecto gráfico sencillo y atractivo que sólo ocupa pocos MB de memoria. Esta solución es barata, ya que sólo se necesita una tarjeta PCI para su funcionamiento, en el caso que nos atañe la tarjeta PCI DAS6025. El software está diseñado para el funcionamiento de la tarjeta PCI DAS6025. Otro tipo de tarjeta no funcionaría, pero con pequeños cambios en el código se solventaría perfectamente.

Uno de los principales inconvenientes surgidos creando este proyecto ha sido en la manera de trabajar. Al solo disponer de una tarjeta PCI DAS6025 en los laboratorios de la universidad tuve que trabajar desde casa a ciegas creando un código que más tarde tenía que probar en los laboratorios. Esto era un gran hándicap ya que tenía que esperar para probar las cosas y arreglarlas en casa y volverlo a probar, cosa que me hacía perder mucho tiempo. Además las características de ambos ordenadores son muy diferentes, ya que disponemos de dos ordenadores con diferente sistema operativo (pero ambos con Windows) y diferentes características técnicas como la memoria RAM (256MB el ordenador del laboratorio frente a 4GB del ordenador de casa).

Este programa nos puede ser útil para infinidad de aplicaciones. Por ejemplo, para el control de la velocidad de un motor de continua, para el arranque progresivo de un motor monofásico o para el testeo de instrumentos de medida y calibración.

14.- Bibliografía y referencias

14.1.- Bibliografía

STANFIELD, Scott: *Cómo se hace con visual C++ 4*; Inforbook's, Barcelona 1996.

14.2.- Referencias

Instrumentación y generador de señales:

- http://electroac.com.ar/intrumentacion.htm
- http://pe.kalipedia.com/tecnologia/tema/generador-senales-funciones.html?x=20070822klpingtcn_131.Kes&ap=1
- http://www.redestelecom.es/Productos/200907240009/Generador-de-senales-de-microondas-de-R-S.aspx

Manuales:

- http://www.mccdaq.com/PDFs/Manuals/PCI-DAS6025-23.pdf
- http://www.mccdaq.com/pci-data-acquisition/PCI-DAS6025.aspx
- http://www.mccdaq.com/PDFs/Manuals/sm-ul-functions.pdf
- http://www.mccdaq.com/PDFs/Manuals/sm-ul-user-guide.pdf

Hardware:

- https://directory.adeptscience.co.uk/productid/C100HD50-3/1/C100HD50-3.html
- http://www.mccdaq.com/products/pci_products.aspx?ao=1

Software:

- http://www.desarrolloweb.com/articulos/1328.php
- http://www.vacationinnicaragua.com/microsoft/visual-basic/visual-basic-2008.pdf
- http://www.cplusplus.com/doc/tutorial/dynamic/
- http://msdn.microsoft.com/es-es/default.aspx

15.- Anexos

Se empezará a nombrar el archivo prueba3.cpp. Este ha sido desglosado en sus distintas funciones para una mayor comprensión. La unión de las diferentes funciones crea el código del archivo principal. En color verde se dispone de los comentarios realizados en el código que se explica las partes fundamentales. Si se desea saber la explicación exhausta de cada función es recomendable ver el apartado octavo de este proyecto.

NOTA: Aquellas líneas que se encuentra con una separación inferior a lo normal significa que es una misma línea de código.

```
// prueba3.cpp : main project file.

#include "stdafx.h"

#include "Form1.h"

#include "stdio.h"

#include "C:/MCC/CWIN/cbw.h"

#include "conio.h"

#include <cstdlib>

#include <ctime>

#include <iostream>

#include "math.h"


using namespace prueba3;

[STAThreadAttribute]


int main(array<System::String ^> ^args)

{

                              // Enabling Windows XP visual effects
before any controls are created

      Application::EnableVisualStyles();

      Application::SetCompatibleTextRenderingDefault(false);

                              // Create the main window and run it

      Application::Run(gcnew Form1());


      return 0;

}
```

Figura 15.1: Archivo prueba3.cpp

```
System::Void vector(void)

{

      Personalizar = new unsigned short [12];

      pictureBox = gcnew array<System::Windows::Forms::PictureBox^>
(211);                          //Guarda 211 espacios en memoria

            for (int i = 0; i < 211; i++)

                  pictureBox[i] = gcnew System::Windows::Forms:
:PictureBox();                //En cada espacio pone un PictureBox

            Texto = gcnew array<System::Windows::Forms: :Label^>(21);
                              //Guarda 21 puestos en memoria

            for (int i = 0; i < 21; i++)

                  Texto[i] = gcnew System::Windows::Forms::Label();
                              //En cada puesto pone un Label

}
```

Figura 15.2: Función vector() del archivo prueba3.cpp

```
System::Void funcionar(void)

{

            pin_ptr<long> Ratepointer = &Rate;      // Combierte la @
de una variable en una variable long

      cbAOutScan(0,0,0,Count,Ratepointer,BIP10VOLTS,MemHandle,CONTINUO
US|BACKGROUND);               //Lanza los datos al D/A

            button4->Visible=true;

}
```

Figura 15.3: Función funcionar() del archivo prueba3.cpp

```
//PERSONALIZADO

System::Void graf(short vari)

{

      if(vari<21)

      {                             // Si click 1º columna

            for(int i=0; i<21; i++)
```

```
                  pictureBox[i]->BackColor = System::Drawing::Color:
:Red;                                   //1º columna en rojo
            for(int i=21; i<42; i++){

                  pictureBox[i]->BackColor = System::Drawing::Color:
:Green;                                 //2º columna en verde

                  pictureBox[i]->Enabled = true;      //2º clickable

             }

            pictureBox[vari]->BackColor= System::Drawing::Color::Blue;
                                        // Dato clickado en azul

            Personalizar[0] = short(4095*(19.8-(vari*0.99))/19.8);
            // y guardado

       }

      else if(vari<42)                  // Si click 2º columna

      {

            for(int i=21; i<42; i++)

                  pictureBox[i]->BackColor = System::Drawing::Color:
:Red;

            for(int i=42; i<63; i++){

                  pictureBox[i]->BackColor = System::Drawing::Color:
:Green;

                   pictureBox[i]->Enabled = true;


            }

            pictureBox[vari]->BackColor= System::Drawing::Color::Blue;

            Personalizar[1] = short(4095*(19.8-0.99*(vari-21))/19.8);

      }

      else if(vari<63)                  // Si click 3º columna

      {

            for(int i=42; i<63; i++)

                  pictureBox[i]->BackColor = System::Drawing::Color:
:Red;

            for(int i=63; i<84; i++){

                  pictureBox[i]->BackColor = System::Drawing::Color:
:Green;

                  pictureBox[i]->Enabled = true;

            }

            pictureBox[vari]->BackColor= System::Drawing::Color::Blue;
```

```
            Personalizar[2] = short(4095*(19.8-0.99*(vari-42))/19.8);
      }
      else if(vari<84)                    // Si click 4º columna
      {
            for(int i=63; i<84; i++)
                  pictureBox[i]->BackColor = System::Drawing::Color:
:Red;
            for(int i=84; i<105; i++){
                  pictureBox[i]->BackColor = System::Drawing::Color:
:Green;
            pictureBox[i]->Enabled = true;
            }
            pictureBox[vari]->BackColor= System::Drawing::Color::Blue;
            Personalizar[3] = short(4095*(19.8-0.99*(vari-63))/19.8);
            }
      else if(vari<105)             // Si click 5º columna
      {
            for(int i=84; i<105; i++)
                  pictureBox[i]->BackColor = System::Drawing::Color:
:Red;
            for(int i=105; i<126; i++){
                  pictureBox[i]->BackColor = System::Drawing::Color:
:Green;
                  pictureBox[i]->Enabled = true;
            }
            pictureBox[vari]->BackColor= System::Drawing::Color::Blue;
            Personalizar[4] = short(4095*(19.8-0.99*(vari-84))/19.8);
            }
      else if(vari<126)                   // Si click 6º columna
      {
            for(int i=105; i<126; i++)
                  pictureBox[i]->BackColor = System::Drawing::Color:
:Red;
            for(int i=126; i<147; i++)
            {
```

```
                  pictureBox[i]->BackColor = System::Drawing::Color:
:Green;
                  pictureBox[i]->Enabled = true;
            }
            pictureBox[vari]->BackColor= System::Drawing::Color::Blue;
            Personalizar[5] = short(4095*(19.8-0.99*(vari-105))/19.8);
      }
      else if(vari<147)                  // Si click 7º columna
      {
            for(int i=126; i<147; i++)
                  pictureBox[i]->BackColor = System::Drawing::Color:
:Red;
            for(int i=147; i<168; i++){
                  pictureBox[i]->BackColor = System::Drawing::Color:
:Green;
                  pictureBox[i]->Enabled = true;
            }
            pictureBox[vari]->BackColor= System::Drawing::Color::Blue;
            Personalizar[6] = short(4095*(19.8-0.99*(vari-126))/19.8);
      }
      else if(vari<168)                  // Si click 8º columna
      {
            for(int i=147; i<168; i++)
                  pictureBox[i]->BackColor = System::Drawing::Color:
:Red;
            for(int i=168; i<189; i++){
                  pictureBox[i]->BackColor = System::Drawing::Color:
:Green;
                  pictureBox[i]->Enabled = true;
            }
            pictureBox[vari]->BackColor= System::Drawing::Color::Blue;
            Personalizar[7] = short(4095*(19.8-0.99*(vari-147))/19.8);
      }
      else if(vari<189)                  // Si click 9º columna
      {
```

```
            for(int i=168; i<189; i++)
                  pictureBox[i]->BackColor = System::Drawing::Color:
:Red;
            for(int i=189; i<210; i++){
                  pictureBox[i]->BackColor = System::Drawing::Color:
:Green;
                  pictureBox[i]->Enabled = true;
            }
            pictureBox[vari]->BackColor= System::Drawing::Color::Blue;
            Personalizar[8] = short(4095*(19.8-0.99*(vari-168))/19.8);
      }
      else if(vari<210)                  // Si click 10º columna
      {
            for(int i=189; i<210; i++)
                  pictureBox[i]->BackColor = System::Drawing::Color:
:Red;
            pictureBox[vari]->BackColor= System::Drawing::Color::Blue;
            Personalizar[9] = short(4095*(19.8-0.99*(vari-189))/19.8);
            button2->Enabled= true;
      }
}
```

Figura 15.4: Función graf() del archivo prueba3.cpp

```
System::Void ver_person(bool visible)
{                                          //Pone visibles o
invisibles la parte de "Personalizada"
      for(int i=0;i<210;i++)
            pictureBox[i]->Visible=visible;  //Aquí los cuadrados
      for(int i=0;i<21;i++)
            Texto[i]->Visible=visible;      //Aquí el texto
             }
```

Figura 15.5: Función ver_person() del archivo prueba3.cpp

```
//Contador y rate del convertidor D/A
System::Void timer(float frec)
```

```
{
      if (frec<0.5)                        //Baja frecuencia
      {
            Count=20000;                   //Muchos puntos de referencia
            divisor=1;
            Rate=(long)(Count*frec/divisor + .5);
      }
      else if (frec<=5)                    //Aumentando la frecuencia
      {
            Count=2000;                    //Menos cantidad de puntos
            divisor=1;
            Rate=(long)(Count*frec/divisor + .5);
      }
      else if (frec<=50)                   //Si aumentamos la frecuencia
      {
            Count=2000;                    //Y no podemos disminuir los
puntos porque estamos al mínimo
            divisor=10;                    //Los disminuimos por
software mediante divisor
            Rate=(long)(Count*frec/divisor + .5);
      }
      else if (frec<=500)                  //Si aumentamos la frecuencia
      {
            Count=2000;                    //Y no podemos bajar Count
            divisor=100;                   //Aumenta división bajandole
la resolución
            Rate=(long)(Count*frec/divisor + .5);
      }
      MemHandle = cbWinBufAlloc(Count);   //Creamos un buffer en la
memoria de la tarjeta
      if(!MemHandle) printf("Error en buffer\n");          //Miramos
si existe

      DataArray = (unsigned short*)GlobalLock(MemHandle);  //Asignamos
un vector al espacio reservado
}
```

Figura 15.6: Función timer() del archivo prueba3.cpp

```
//FUNCIONES

System::Void continua(System::Decimal amplitud)

{

      short i_amplitud;

      double decimal;

      decimal= float(amplitud);   //Pasamos el valor de Decimal a
float para poder trabajar con él

      decimal += 9.93;              //Ahora que lo tenemos pasado lo
convertimos a valor entre 0 y 19.8

      i_amplitud=short((4095.0*decimal)/19.86 + .501);  /*Convertimos
el valor en V a un valor entre 0 y 4095 mediante una regla de 3. Si
19.8->4095, decimal->i_amplitud. Además redondeamos.*/

      cbAOut(0,0,BIP10VOLTS,i_amplitud);  /*Sacamos el valor por la
salida. Al ser un único valor, no necesitamos guardarlo en un buffer y
darle una frecuencia de mostreo*/

}
```

Figura 15.7: Función continua() del archivo prueba3.cpp

```
System::Void cuadrada(System::Decimal amplitud, System::Decimal offset,
System::Decimal frecuencia, System::Decimal dutycicle)

{

      short cuenta=0;                     //Variable que determina en que
valor de Count hay que cambiar de ON a OFF

      timer(float(frecuencia));        //Llama a timer() para saber el valor
de Count y Rate

      cuenta= short(Count * float(dutycicle)/(100*divisor));     //Segun
Count y dutycicle se determina cuenta

      short i;

      short j;

      for(j=0; j<divisor; j++)           //Crea "divisor" numero de periodos
dentro del buffer

      {

            for(i=short(Count/divisor) * j; i<(cuenta + short(Count/divisor)
* j);i++)

            {                                   //Zona ON
```

```
                  DataArray[i] = short(.5+(4095*(float(amplitud) +
float(offset) + 9.9)/19.8));                    //valor ON

            }

            for(i=cuenta + short(Count/divisor) * j; i<short(Count/divisor)
*(j+1) ; i++ )

            {                                   //Zona OFF

                  DataArray[i] = short(.5+(4095*(float(offset) + 9.9)/19.8));
                  //Valor OFF

            }

      }

      funcionar();        //Llama a funcionar() para que empiece a convertir
el buffer preparado

}
```

Figura 15.8: Función cuadrada() del archivo prueba3.cpp

```
System::Void triangular(System::Decimal amplitud, System::Decimal offset,
System::Decimal tiempo_s, System::Decimal tiempo_b)

{

     float A_tiempo=0.0;                        //Incremento de tiempo

      short cuenta=0;                   /*Variable que determina en que
      valor de Count hay que cambiar de subida a bajada*/

      timer(1000/(float(tiempo_s) + float(tiempo_b))); /*la frecuencia es la
      inversa del tiempo de subida más el tiempo de bajada*/

      A_tiempo=(float)(1000.0/Rate);     //Tiempo entre conversion, periodo

      cuenta= short (float(tiempo_s)/A_tiempo);   //Cuenta es el tiempo de
      subida entre el periodo

      short i;

      short j;

      for(j=0; j<divisor; j++)

      {

            for(i=short(Count/divisor) * j; i<(cuenta + short(Count/divisor)
      * j);i++)

            {

                  DataArray[i] = short(.5+(4095*(float(offset) + 9.9 +((i-
      (Count/divisor)*j) * float(amplitud)/cuenta))/19.8));      /*El buffer
      se carga con valores que empiezan desde el offset(punto más bajo)
      hasta el offset+amplitud (punto más alto) mediante pequeños
      incrementos*/
```

```
            }

            for(i=cuenta + short(Count/divisor) * j; i<short(Count/divisor)*
      (j+1);i++)

            {

                  DataArray[i] = short(.5+(4095*(float(offset) + 9.9 +
      float(amplitud) -((i-(cuenta + (Count/divisor) * j)) *
      float(amplitud)/((Count/divisor)-cuenta)))/19.8)); /*El buffer se
      carga desde offset+amplitud hasta offset con pequeños decrementos
      (bajada)*/

            }

      }

      funcionar();      //Llama a funcionar() para que empiece a convertir
      el buffer preparado

}
```

Figura 15.9: Función triangular() del archivo prueba3.cpp

```
System::Void diente(System::Decimal amplitud, System::Decimal offset,
System::Decimal tiempo, System::Boolean option)

{

      if (option==true)            // Si queremos el diente: /|

      {

            timer(1000/float(tiempo));

            for(int j=0; j<divisor; j++)

            {

                  for(int i=short(Count/divisor) * j; i<(short(Count/divisor)
      * (j+1));i++)

                        DataArray[i] = short(.5+(4095*(float(offset) + 9.9
      +((i- (Count/divisor)*j) * float(amplitud)/(Count/divisor)))/19.8));

            }

            funcionar();            //Llama a funcionar() para que empiece a
      convertir el buffer preparado

      }

      else                        //Si queremos el diente: |\

      {

            timer(1000/float(tiempo));
```

```
            for(int j=0; j<divisor; j++)

            {

                  for(int i=short(Count/divisor) * j; i<(short(Count/divisor)
      * (j+1));i++)

                        DataArray[i] = short(.5+(4095*(float(offset) + 9.9 +
      float(amplitud) - ((i-((Count/divisor) * j)) * float(amplitud)/
      (Count/divisor)))/19.8));

            }

            funcionar();                  //Llama a funcionar() para que
      empiece a convertir el buffer preparado

      }

}
```

Figura 15.10: Función diente() del archivo prueba3.cpp

```
System::Void trapezoidal(System::Decimal amplitud, System::Decimal offset,
System::Decimal tiempo_s, System::Decimal tiempo_b, System::Decimal
tiempo_ON, System::Decimal tiempo_OFF)

{

      float A_tiempo=0.0;                 //periodo

      short cuenta1= 0;                   //Paso de OFF a subida

      short cuenta2= 0;                   //Paso de subida a ON

      short cuenta3 = 0;                  //Paso de ON a bajada

      timer(1000/(float(tiempo_s) + float(tiempo_b) + float(tiempo_ON) +
      float(tiempo_OFF)));

      A_tiempo=(float)(1000.0/Rate);

      cuenta1= short (float(tiempo_s)/A_tiempo);

      cuenta2= cuenta1 + short (float(tiempo_ON)/A_tiempo);

      cuenta3= cuenta2 + short (float(tiempo_b)/A_tiempo);

      short i;                        //Es la fusión de cuadrada y triangular

      short j;

      for(j=0; j<divisor; j++)        //Preparación del buffer

      {
```

```
            for(i=short(Count/divisor) * (j); i< cuenta1 +
      short(Count/divisor) * j ;i++ )

                  DataArray[i] = short(.5+(4095*(float(offset) + 9.9 +((i-
      (Count/divisor)*j) * float(amplitud)/cuenta1))/19.8));

            for(i=cuenta1 + short(Count/divisor) * j; i< cuenta2 +
      short(Count/divisor) * j;i++)

                  DataArray[i] = short(.5+(4095*(float(amplitud) +
      float(offset) + 9.9)/19.8));

            for(i=cuenta2 + short(Count/divisor) * j ; i<cuenta3 +
      short(Count/divisor) * j; i++)

                  DataArray[i]= short(.5+(4095*(float(offset) + 9.9 +
      float(amplitud) -((i-cuenta2-(Count/divisor)*j)*float(amplitud)
      /(cuenta3-cuenta2)))/19.8));

            for(i=cuenta3 + short(Count/divisor) * j; i< short(Count
      /divisor)*(j+1); i++)

                  DataArray[i]= short(.5+(4095*(float(offset) + 9.9)/19.8));

      }

      funcionar();           //Llama a funcionar() para que empiece a
      convertir el buffer preparado

}
```

Figura 15.11: Función trapezoidal() del archivo prueba3.cpp

```
System::Void sinusoidal(System::Decimal amplitud, System::Decimal offset,
System::Decimal frecuencia, System::Decimal desfase)

{

      timer(float(frecuencia));  //Prepara Count y Rate según la frecuencia

      short j;

      short i;

      for(j=0; j<divisor; j++)

      {

            for(i=short(Count/divisor) * j;i<short(Count/divisor)*(j+1);i++)

            DataArray[i] = short(.5+(4095*(float(amplitud) * sin(((i-
      (Count/divisor)*j)*360/(Count/divisor) + float(desfase))* 3.14 /180.0)
      + float(offset) + 9.9)/19.8));    //Crea un seno con amplitud, offset
      y desfase

      }

      funcionar();                  //Llama a funcionar() para que empiece a
      convertir el buffer preparado

}
```

Figura 15.12: Función sinusoidal() del archivo prueba3.cpp

```
System::Void personal(System::Decimal tiempo, int opcion)
{
	short cuenta=0;
	short i;
	if(opcion==1){                          //Si es pulsante
		timer(1/float(tiempo));    //Ponemos valores de Count y Rate
		for(i=0; i<int(Count/10);i++) //Y cada decima parte de Count
			DataArray[i] = short(Personalizar[0]);  //Le ponemos el
	valor escogido por dibujo
		for(i=int(Count/10); i<2*int(Count/10);i++)
			DataArray[i] = short(Personalizar[1]);
		for(i=2*int(Count/10); i<3*int(Count/10);i++)
			DataArray[i] = short(Personalizar[2]);
		for(i=3*int(Count/10); i<4*int(Count/10);i++)
			DataArray[i] = short(Personalizar[3]);
		for(i=4*int(Count/10); i<5*int(Count/10);i++)
			DataArray[i] = short(Personalizar[4]);
		for(i=5*int(Count/10); i<6*int(Count/10);i++)
			DataArray[i] = short(Personalizar[5]);
		for(i=6*int(Count/10); i<7*int(Count/10);i++)
			DataArray[i] = short(Personalizar[6]);
		for(i=7*int(Count/10); i<8*int(Count/10);i++)
			DataArray[i] = short(Personalizar[7]);
		for(i=8*int(Count/10); i<9*int(Count/10);i++)
			DataArray[i] = short(Personalizar[8]);
		for(i=9*int(Count/10); i<10*int(Count/10);i++)
			DataArray[i] = short(Personalizar[9]);
	}
	else
```

```
        {                                   //Si es progresivo

              timer(1/float(tiempo));   //Ponemos valores de Count y Rate

              for(i=0; i<int(Count/10);i++)    //Y cada decima parte de Count

                    DataArray[i] = short(Personalizar[0]+(Personalizar[1]-
        Personalizar[0])/ (Count/10.0)*i);        /* Le ponemos el valor y
        pequeños incrementos o decrementos hasta llegar al siguiente valor.
        Todos los valores se han guardado en la función graf()*/

              for(i=int(Count/10); i<2*int(Count/10);i++)

                    DataArray[i] = short(Personalizar[1]+(Personalizar[2]-
        Personalizar[1])/(Count/10.0)*(i-int(Count/10)));

              for(i=2*int(Count/10); i<3*int(Count/10);i++)

                    DataArray[i] = short(Personalizar[2]+(Personalizar[3]-
        Personalizar[2])/(Count/10.0)*(i-2*int(Count/10)));

              for(i=3*int(Count/10); i<4*int(Count/10);i++)

                    DataArray[i] = short(Personalizar[3]+(Personalizar[4]-
        Personalizar[3])/(Count/10.0)*(i-3*int(Count/10)));

              for(i=4*int(Count/10); i<5*int(Count/10);i++)

                    DataArray[i] = short(Personalizar[4]+(Personalizar[5]-
        Personalizar[4])/(Count/10.0)*(i-4*int(Count/10)));

              for(i=5*int(Count/10); i<6*int(Count/10);i++)

                    DataArray[i] = short(Personalizar[5]+(Personalizar[6]-
        Personalizar[5])/(Count/10.0)*(i-5*int(Count/10)));

              for(i=6*int(Count/10); i<7*int(Count/10);i++)

                    DataArray[i] = short(Personalizar[6]+(Personalizar[7]-
        Personalizar[6])/(Count/10.0)*(i-6*int(Count/10)));

              for(i=7*int(Count/10); i<8*int(Count/10);i++)

                    DataArray[i] = short(Personalizar[7]+(Personalizar[8]-
        Personalizar[7])/(Count/10.0)*(i-7*int(Count/10)));

              for(i=8*int(Count/10); i<9*int(Count/10);i++)

                    DataArray[i] = short(Personalizar[8]+(Personalizar[9]-
        Personalizar[8])/(Count/10.0)*(i-8*int(Count/10)));

              for(i=9*int(Count/10); i<10*int(Count/10);i++)




                    DataArray[i] = short(Personalizar[9]+(Personalizar[0]-
              Personalizar[9])/(Count/10.0)*(i-9*int(Count/10)));

        }

}
```

Figura 15.13: Función personal() del archivo prueba3.cpp

```
System::Void irregular(System::Decimal amplitud, System::Decimal offset,
System::Decimal velocidad)

{

	short amplitut=0;

	double random_float=0.0;

	float random=0.0;

	amplitut= short(amplitud*10);   //se multiplica la amplitud por 10
	para crear decimales después y no solo int

	srand((unsigned)time(0));       //Crea la semilla del random()

	Rate=long(float(velocidad));    //El Rate será la velocidad indicada

	Count=20000;                    //El valor de Count ya está determinado

	MemHandle = cbWinBufAlloc(Count);  //Al no llamar a timer() reservamos
	la memoria aquí

	if(!MemHandle) printf("Error en buffer\n");

	DataArray = (unsigned short*)GlobalLock(MemHandle);

	for(short i=0; i<Count;i++){

		random= float((rand()%(amplitut+1))/10.0); //Creamos un número
	aleatorio q estará entre 0 y la "amplitut"

		random_float = random + float(offset);   //Hemos tratado para
	que sea entre 0 y amlitud y sumamos el offset

		random_float+=9.9;               //Sumamos 9.9 para poder
	pasarlo a numero entre 0 y 19.8

		DataArray[i] = short(.5+(4095.0 * random_float) /19.8);
	//Pasamos mediante regla de 3 a numero entre 0 y 4095

	}

	funcionar();                        //Llama a funcionar() para que
	empiece a convertir el buffer preparado

}
```

Figura 15.14: Función irregular() del archivo prueba3.cpp

```
double media(void)

{

	int media=0;
```

```
	int i;

	for(i=0;i<Count;i++)

	media+=DataArray[i];		//Sumamos todos los valores del buffer

	return(((19.8*(media/i)/4095.0)-9.9)); //Lo dividimos por el número de
	muestras y lo pasamos entre 0 y 4095

}
```

Figura 15.15: Función media() del archivo prueba3.cpp

```
double valor_eficaz(void)

{

	double media=0;

	int i;

	for(i=0;i<Count;i++)  //Sumatorio del valor de buffer, ya pasado entre
	0 y 4095, al cuadrado

		media+=(((19.8*DataArray[i]/4095.0)-9.9)*((19.8*DataArray[i]
	/4095.0)-9.9));

	media=media/i;		//La suma de cuadrados se divide entre el número
	de muestras

	media=sqrt(media);	//Y se hace la raíz

	return(media);

}
```

Figura 15.16: Función valor_eficaz() del archivo prueba3.cpp

```
System::Void inicializacion(void)

{

	cbAOut(0,0,BIP10VOLTS,2048);	//Pone el valor 2048 (0 V) a la salida
```

```
}
```

Figura 15.17: Función inicialización() del archivo prueba3.cpp

```
System::Void dibujar_grafico(void)
{                                          //Dibuja lineas negras
      linea_black(40,10,40,240);
      //linea_black(x_start,y_start,x_final,y_final)
      linea_black(30,120,400,120);
      linea_black(30,20,400,20);
      linea_black(30,220,400,220);
      linea_black(130,10,130,240
      linea_black(220,10,220,240);
      linea_black(310,10,310,240);
      linea_black(399,10,399,240);
      linea_black(30,170,400,170);
      linea_black(30,70,400,70);
}
```

Figura 15.18: Función dibujar_grafico() del archivo prueba3.cpp

```
System::Void linea_blue(float x, float y, float x1, float y1 )
{
      Pen^ lin = gcnew Pen( Color::Blue ,3.0F );     //Línea azul de 3.0F de
      grosor
      lin->LineJoin = System::Drawing::Drawing2D::LineJoin::Bevel;
      Graphics^ graf = pictureBox1->CreateGraphics (); //La línea se creará
      en pictureBox1
      graf->DrawLine (lin,x,y,x1,y1);          //Crea la línea
      delete lin;
      delete graf;
}
```

Figura 15.19: Función linea_blue() del archivo prueba3.cpp

```
System::Void linea_green(float x, float y, float x1, float y1 )
{
	Pen^ lin = gcnew Pen( Color::Green ,1.0F );	//Línea verde de 1.0F
	de grosor
	lin->LineJoin = System::Drawing::Drawing2D::LineJoin::Bevel;
	Graphics^ graf = pictureBox1->CreateGraphics ();
	graf->DrawLine (lin,x,y,x1,y1);
	delete lin;
	delete graf;
}
```

Figura 15.20: Función linea_green() del archivo prueba3.cpp

```
System::Void linea_red(float x, float y, float x1, float y1 )
{
	Pen^ lin = gcnew Pen( Color::Red ,1.0F );	//Línea roja de 1.0F de
	grosor
	lin->LineJoin = System::Drawing::Drawing2D::LineJoin::Bevel;
	Graphics^ graf = pictureBox1->CreateGraphics ();
	graf->DrawLine (lin,x,y,x1,y1);
	delete lin;
	delete graf;
}
```

Figura 15.21: Función linea_red() del archivo prueba3.cpp

```
System::Void linea_black(int x, int y, int x1, int y1 )
{
	Pen^ lin = gcnew Pen( Color::Black,1.0F );	//Línea negra de 1.0F
	de grosor
	lin->LineJoin = System::Drawing::Drawing2D::LineJoin::Bevel;
	Graphics^ graf = pictureBox1->CreateGraphics ();
```

```
	graf->DrawLine (lin,x,y,x1,y1);

	delete lin;

	delete graf;

}
```

Figura 15.22: Función linea_black() del archivo prueba3.cpp

```
System::Void button1_Click_1(System::Object^  sender, System::EventArgs^  e)
{                                       //Al presionar button1

	inicializacion();	//Llamamos a inicialización para poner todo a 0

	groupBox2->Visible=true;	//Y hacemos visible el groupBox2

	if(comboBox1->Text=="Continua")   //Si el comboBox pone continua

	{

		groupBox2->Text = comboBox1->Text; //Al texto de gruopBox le
	ponemos el mismo que a comboBox

		label1->Text="Escoja una amplitud de la señal de -9.9 a 9.9:";
	/*Y activamos/desactivamos los componentes que nos haga falta para
	hacer el aspecto visual que queremos y la variables que necesitemos*/

		numericUpDown1->Maximum = System::Decimal(gcnew cli::array<
	System::Int32 >(4) {99, 0, 0, 65536});

		numericUpDown1->Minimum = System::Decimal(gcnew cli::array<
	System::Int32 >(4) {99, 0, 0, -2147418112});

		numericUpDown1->Visible=true;

		label2->Text="";

		label3->Visible=false;

		numericUpDown2->Visible=false;

		label4->Text="";

		numericUpDown3->Visible=false;

		label5->Text="";

		label20->Text="";

		label21->Text="";

		label22->Text="";

		label23->Text="";

		numericUpDown4->Visible=false;
```

```
        numericUpDown5->Visible=false;
        numericUpDown6->Visible=false;
        radioButton1->Visible= false;
        radioButton2->Visible= false;
        label6->Text="";
        numericUpDown7->Visible=false;
        label7->Text="";
        numericUpDown8->Visible=false;
        label8->Visible=false;
        label18->Visible=false;
        label17->Visible=true;
        ver_person(false);
        label26->Visible=false;
        label27->Visible=false;
        numericUpDown9->Visible=false;
        comboBox2->Visible=false;
    }
    else if(comboBox1->Text=="Cuadrada")
    {
        groupBox2->Text = comboBox1->Text;
        label1->Text="Escoja una amplitud de la señal de 0 a 19,8:";
        numericUpDown1->Maximum = System::Decimal(gcnew cli::array<
    System::Int32 >(4) {198, 0, 0, 65536});
        numericUpDown1->Minimum = System::Decimal(gcnew cli::array<
    System::Int32 >(4) {0, 0, 0, 0});
        numericUpDown1->Visible=true;
        label2->Text="Escoja el offset de la señal de -9,9 a 9,9:";
        label3->Visible=true;
        numericUpDown2->Visible=true;
        label4->Text="Escoja la frecuencia, de 1 mHz a 500 Hz:";
        numericUpDown3->Visible=true;
        label20->Text="Hz";
        numericUpDown3->DecimalPlaces = 3;
```

```
      numericUpDown3->Increment = System::Decimal(gcnew cli::array<
System::Int32 >(4) {1, 0, 0, 65536});

      numericUpDown3->Maximum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {500, 0, 0, 0});

      numericUpDown3->Minimum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {1, 0, 0, 196608});

      label5->Text="Escoja el ciclo de trabajo en %:";

      label21->Text="%";

      label22->Text="";

      label23->Text="";

      numericUpDown4->Visible=true;

      numericUpDown4->Maximum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {100, 0, 0, 0});

      numericUpDown4->Value = System::Decimal(gcnew cli::array<
System::Int32 >(4) {50, 0, 0, 0});

      numericUpDown5->Visible=false;

      numericUpDown6->Visible=false;

      radioButton1->Visible= false;

      radioButton2->Visible= false;

      label6->Text="";

      numericUpDown7->Visible=false;

      label7->Text="";

      numericUpDown8->Visible=false;

      label8->Visible=false;

      label18->Visible=true;

      label17->Visible=true;

      ver_person(false);

      label26->Visible=false;

      label27->Visible=false;

      numericUpDown9->Visible=false;

      comboBox2->Visible=false;

}

else if (comboBox1->Text=="Triangular")

{

      groupBox2->Text = comboBox1->Text;
```

```
      label1->Text="Escoja una amplitud de la señal de 0 a 19,8:";

      numericUpDown1->Maximum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {198, 0, 0, 65536});

      numericUpDown1->Minimum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {0, 0, 0, 0});

      numericUpDown1->Visible=true;

      label2->Text="Escoja el offset de la señal de -9,9 a 9,9:";

      label3->Visible=true;

      numericUpDown2->Visible=true;

      label4->Text="Escoja el tiempo de subida de 2ms a 50s:";

      label20->Text="ms";

      numericUpDown3->Visible=false;

      label5->Text="Escoja el tiempo de bajada de 2ms a 50s:";

      label21->Text="ms";

      label22->Text="";

      label23->Text="";

      numericUpDown4->Visible=false;

      numericUpDown5->Visible=true;

      numericUpDown5->Maximum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {50000, 0, 0, 0});

      numericUpDown5->Minimum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {2, 0, 0, 0});

      numericUpDown6->Visible=true;

      numericUpDown6->Maximum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {50000, 0, 0, 0});

      numericUpDown6->Minimum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {2, 0, 0, 0});

      radioButton1->Visible= false;

      radioButton2->Visible= false;

      label6->Text="";

      numericUpDown7->Visible=false;

      label7->Text="";

      numericUpDown8->Visible=false;

      label8->Visible=false;

      label18->Visible=true;

      label17->Visible=true;
```

```
	ver_person(false);

	label26->Visible=false;

	label27->Visible=false;

	numericUpDown9->Visible=false;

	comboBox2->Visible=false;

}

else if (comboBox1->Text=="Diente de sierra")

{

	groupBox2->Text = comboBox1->Text;

	label1->Text="Escoja una amplitud de la señal de 0 a 19,8:";

	numericUpDown1->Maximum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {198, 0, 0, 65536});

	numericUpDown1->Minimum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {0, 0, 0, 0});

	numericUpDown1->Visible=true;

	label2->Text="Escoja el offset de la señal de -9,9 a 9,9:";

	label3->Visible=true;

	numericUpDown2->Visible=true;

	label4->Text="Escoja que tiempo quieres que sea nulo:";

	label20->Text="";

	numericUpDown3->Visible=false;

	radioButton1->Visible= true;

	radioButton2->Visible= true;

	label5->Text="Escoja el tiempo de bajada de 2ms a 50s:";

	label21->Text="ms";

	label22->Text="";

	label23->Text="";

	numericUpDown4->Visible=false;

	numericUpDown5->Visible=false;

	numericUpDown6->Visible=true;

	numericUpDown6->Maximum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {50000, 0, 0, 0});

	numericUpDown6->Minimum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {2, 0, 0, 0});

	label6->Text="";
```

```
	numericUpDown7->Visible=false;
	label7->Text="";
	numericUpDown8->Visible=false;
	label8->Visible=false;
	label18->Visible=true;
	label17->Visible=true;
	ver_person(false);
	label26->Visible=false;
	label27->Visible=false;
	numericUpDown9->Visible=false;
	comboBox2->Visible=false;
}
else if (comboBox1->Text=="Trapezoidal")
{
	groupBox2->Text = comboBox1->Text;
	label1->Text="Escoja una amplitud de la señal de 0 a 19,8:";
	numericUpDown1->Maximum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {198, 0, 0, 65536});
	numericUpDown1->Minimum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {0, 0, 0, 0});
	numericUpDown1->Visible=true;
	label2->Text="Escoja el offset de la señal de -9,9 a 9,9:";
	label3->Visible=true;
	numericUpDown2->Visible=true;
	label4->Text="Escoja el tiempo de subida de 0ms a 25s:";
	label20->Text="ms";
	numericUpDown3->Visible=false;
	label5->Text="Escoja el tiempo de bajada de 0ms a 25s:";
	label21->Text="ms";
	numericUpDown4->Visible=false;
	numericUpDown5->Visible=true;
	numericUpDown5->Maximum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {25000, 0, 0, 0});
	numericUpDown5->Minimum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {0, 0, 0, 0});
```

```
      numericUpDown6->Visible=true;

      numericUpDown6->Maximum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {25000, 0, 0, 0});

      numericUpDown6->Minimum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {0, 0, 0, 0});

      radioButton1->Visible= false;

      radioButton2->Visible= false;

      label6->Text="Escoja el tiempo en alto (ON) de 0ms a 25s:";

      label22->Text="ms";

      numericUpDown7->Visible=true;

      label7->Text="Escoja el tiempo en bajo (OFF) de 0ms a 25s:";

      label23->Text="ms";

      numericUpDown8->Visible=true;

      label8->Visible=false;

      label18->Visible=true;

      label17->Visible=true;

      ver_person(false);

      label26->Visible=false;

      label27->Visible=false;

      numericUpDown9->Visible=false;

      comboBox2->Visible=false;

}

else if (comboBox1->Text=="Sinusoidal")

{

      groupBox2->Text = comboBox1->Text;

      label1->Text="Escoja una amplitud de la señal de 0 a 19,8:";

      numericUpDown1->Maximum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {198, 0, 0, 65536});

      numericUpDown1->Minimum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {0, 0, 0, 0});

      numericUpDown1->Visible=true;

      label2->Text="Escoja el offset de la señal de -9,9 a 9,9:";

      label3->Visible=true;

      numericUpDown2->Visible=true;

      label4->Text="Escoja la frecuencia, de 1 mHz a 500 Hz:";
```

```
      label20->Text="Hz";

      label21->Text="grados";

      label22->Text="";

      label23->Text="";

      numericUpDown3->Visible=true;

      numericUpDown3->DecimalPlaces = 3;

      numericUpDown3->Increment = System::Decimal(gcnew cli::array<
System::Int32 >(4) {1, 0, 0, 65536});

      numericUpDown3->Maximum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {500, 0, 0, 0});

      numericUpDown3->Minimum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {1, 0, 0, 196608});

      label5->Text="Escoja el ángulo de desfase, de 0 a 360º:";

      numericUpDown4->Visible=true;

      numericUpDown4->Maximum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {360, 0, 0, 0});

      this->numericUpDown4->Value = System::Decimal(gcnew cli::array<
System::Int32 >(4) {0, 0, 0, 0});

      numericUpDown5->Visible=false;

      numericUpDown6->Visible=false;

      radioButton1->Visible= false;

      radioButton2->Visible= false;

      label6->Text="";

      numericUpDown7->Visible=false;

      label7->Text="";

      numericUpDown8->Visible=false;

      label8->Visible=true;

      label18->Visible=true;

      label17->Visible=true;

      ver_person(false);

      label26->Visible=false;

      label27->Visible=false;

      numericUpDown9->Visible=false;

      comboBox2->Visible=false;

}
```

```
else if (comboBox1->Text=="Irregular")

{

      groupBox2->Text = comboBox1->Text;

      label1->Text="Escoja una amplitud de la señal de 0 a 19,8:";

      numericUpDown1->Maximum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {198, 0, 0, 65536});

      numericUpDown1->Minimum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {0, 0, 0, 0});

      numericUpDown1->Visible=true;

      label2->Text="Escoja el offset de la señal de -9,9 a 9,9:";

      label3->Visible=true;

      numericUpDown2->Visible=true;

      label4->Text="Escoja la velocidad de mostreo de 5 a 10000:";

      label20->Text="muestras \n /s";

      label21->Text="";

      label22->Text="";

      label23->Text="";

      numericUpDown3->Visible=true;

      numericUpDown3->DecimalPlaces = 0;

      numericUpDown3->Increment = System::Decimal(gcnew cli::array<
System::Int32 >(4) {1, 0, 0, 0});

      numericUpDown3->Maximum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {10000, 0, 0, 0});

      numericUpDown3->Minimum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {5, 0, 0, 0});

      label5->Text="";

      numericUpDown4->Visible=false;

      numericUpDown5->Visible=false;

      numericUpDown6->Visible=false;

      radioButton1->Visible= false;

      radioButton2->Visible= false;

      label6->Text="";

      numericUpDown7->Visible=false;

      label7->Text="";

      numericUpDown8->Visible=false;
```

```
        label8->Visible=false;
        label18->Visible=true;
        label17->Visible=true;
        ver_person(false);
        label26->Visible=false;
        label27->Visible=false;
        numericUpDown9->Visible=false;
        comboBox2->Visible=false;
    }
    else if (comboBox1->Text=="Personalizada")
    {
        groupBox2->Text = comboBox1->Text;
        label1->Text="";
        numericUpDown1->Visible=false;
        label2->Text="";
        label3->Visible=false;
        numericUpDown2->Visible=false;
        label4->Text="";
        label20->Text="";
        label21->Text="";
        label22->Text="";
        label23->Text="";
        numericUpDown3->Visible=false;
        label5->Text="";
        numericUpDown4->Visible=false;
        numericUpDown5->Visible=false;
        numericUpDown6->Visible=false;
        radioButton1->Visible= false;
        radioButton2->Visible= false;
        label6->Text="";
        numericUpDown7->Visible=false;
        label7->Text="";
        numericUpDown8->Visible=false;
```

```
            label8->Visible=false;
            label18->Visible=false;
            label17->Visible=false;
            ver_person(true);
            button2->Enabled= false;
            label26->Visible=true;
            label27->Visible=true;
            numericUpDown9->Visible=true;
            comboBox2->Visible=true;
      }
      else                          //Si es otra opción de las NO dadas
      {
            groupBox2->Visible=false;    //El gruopBox se hace invisible


            MessageBox::Show("Debe escoger una función de las dadas en la
      lista.", "Error: esa función no existe", MessageBoxButtons::OK,
      MessageBoxIcon::Error);            //Y da una ventana de error
      }
}
```

Figura 15.23: Evento al clickar el componente button1, dentro del archivo prueba3.cpp

```
// Al cambiar el valor de radioButton1
System::Void radioButton1_CheckedChanged(System::Object^  sender,
System::EventArgs^  e)
{
      if(radioButton1->Checked == true) //Si el 1r radioButton es verdadero
            label5->Text="Escoja el tiempo de subida de 1ms a 500s:";
      else                          //Si el 1r radioButton es falso
            label5->Text="Escoja el tiempo de bajada de 1ms a 500s:";
}
```

Figura 15.24: Evento al cambiar el valor del componente radioButton1, dentro del archivo prueba3.cpp

```
System::Void button2_Click(System::Object^  sender, System::EventArgs^  e)
{                                          //Al presionar el button2
      cbStopBackground(0, AOFUNCTION);    //Paramos converiones anteriores
      inicializacion();                    //Ponemos el valor a 0
      double mediana;                      //Creamos variables locales para
calcular la media y el valor eficaz
      short media_aritmetica;
      double valor;
      short eficaz;
      groupBox3->Visible=true;             //Hacemos visible el groupBox3
      label24->Visible=true;
      label25->Visible=true;
      if(comboBox1->Text=="Continua")
      {
            pictureBox1->Refresh();         //Refrescamos la pictureBox1
            dibujar_grafico();             //Y llamamos dibujar_grafico() para
que dibuje las líneas
            label14->Visible=false;
            label15->Visible=false;
            label16->Visible=false;
            label19->Visible=false;
            label24->Text= "Media: " + (float(numericUpDown1->Value)).
ToString() + " v";
            label25->Text= "Valor eficaz: " + (float(numericUpDown1-
>Value)).ToString() + " v";
            continua(numericUpDown1->Value);
            linea_blue(40.0,float(220-(200.0*(9.9+float(numericUpDown1->
Value))/19.8)),400.0,
            float(220-(200.0*(9.9+float(numericUpDown1->Value))/19.8)));
//dibuja la gráfica
            pictureBox2->Visible=false;
      }
      else if(comboBox1->Text=="Cuadrada")
      {
```

```
            if(float(numericUpDown1->Value)+float(numericUpDown2->
Value)>9.9)

            {     //offset+amplitud no debe superar los 9.9 V ya que nos
saldríamos de rango

                  groupBox3->Visible=false;

                  MessageBox::Show("La suma de offset y amplitud debe estar
entre -9,9 a 9,9 v.", "Error: parámetros erróneos", MessageBoxButtons::OK,
MessageBoxIcon::Error);

            }

            else

            {

                  pictureBox1->Refresh();

                  dibujar_grafico();

                  label19->Text= (2/float(numericUpDown3->Value)).ToString()
+ " s";

                  label14->Text= ((2/float(numericUpDown3->Value))/4).
ToString() + " s";

                  label15->Text= ((2/float(numericUpDown3->Value))/2).
ToString() + " s";

                  label16->Text= (3*(2/float(numericUpDown3->Value))/4).
ToString() + " s";

                  label14->Visible=true;

                  label15->Visible=true;

                  label16->Visible=true;

                  label19->Visible=true;

                  cuadrada(numericUpDown1->Value,numericUpDown2->Value,
numericUpDown3->Value,numericUpDown4->Value);

                  mediana = media();

                  media_aritmetica= int(mediana*100 + .5);

                  mediana= media_aritmetica/100.0;    //pone 2 decimales

                  label24->Text= "Media: " + mediana.ToString() + " v";

                  linea_red(40.0,float(220-(200.0*(9.9 + mediana)/
19.8)),400.0, float(220-(200.0*(9.9 + mediana)/19.8)));  //Dibuja de rojo la
media

                  valor = valor_eficaz();

                  eficaz= int(valor*100 + .5);

                  valor= eficaz/100.0;                //pone 2 decimales

                  label25->Text= "Valor eficaz: " + valor.ToString() + " v";
```

```
                  //Dibuja de verde el valor eficaz

                  linea_green(40.0,float(220-(200.0*(9.9 + valor)/19.8)),
400.0, float(220-(200.0*(9.9 + valor)/19.8)));

                  for(int i=0;i<short(Count/divisor); i++)

                  {

                        //dibuja de azul la gráfica, 2 periodos

                        linea_blue(float(40.0 + 180.0/(Count/divisor)*(i)),
float(220.0-(200.0*DataArray[i]/4095.0)),float(40.0 + 180.0/(Count/divisor)
*(i+1)), float(220.0-(200.0*DataArray[i]/4095.0)));

                        linea_blue(float(220.0 + 180.0/(Count/divisor)*(i)),
float(220.0-(200.0*DataArray[i]/4095.0)),float(220.0 + 180.0/(Count/divisor)
*(i+1)), float(220.0-(200.0*DataArray[i]/4095.0)));

                  }

                  if(divisor==1)

                        pictureBox2->Visible= false; //Se apaga un indicador
como que la resolución es baja

                  else

                        pictureBox2->Visible= true;  //Se hace visible el
indicador

            }

      }

      else if(comboBox1->Text=="Triangular")

      {

            if(float(numericUpDown1->Value)+float(numericUpDown2->
Value)>9.9)

            {

                  groupBox3->Visible=false;

                  MessageBox::Show("La suma de offset y amplitud debe estar
entre -9,9 a 9,9 v.", "Error: parámetros erróneos", MessageBoxButtons::OK,
MessageBoxIcon::Error);

            }

            else

            {

                  pictureBox1->Refresh();

                   dibujar_grafico();

                  label19->Text= ((float(numericUpDown5->Value) +
float(numericUpDown6->Value))/1000.0).ToString() + " s";
```

```
                  label14->Text= ((float(numericUpDown5->Value) +
float(numericUpDown6->Value))/4000.0).ToString() + " s";

                  label15->Text= ((float(numericUpDown5->Value) +
float(numericUpDown6->Value))/2000.0).ToString() + " s";

                  label16->Text= (3*(float(numericUpDown5->Value) +
float(numericUpDown6->Value))/4000.0).ToString() + " s";

                  label14->Visible=true;

                  label15->Visible=true;

                  label16->Visible=true;

                  label19->Visible=true;

                  triangular(numericUpDown1->Value,numericUpDown2->
Value,numericUpDown5->Value,numericUpDown6->Value);

                  mediana = media();

                  media_aritmetica= int(mediana*100 + .5);

                  mediana= media_aritmetica/100.0;    //pone 2 decimales

                  label24->Text= "Media: " + mediana.ToString() + " v";

                  linea_red(40.0,float(220-(200.0*(9.9 + mediana)/19.8)),
400.0, float(220-(200.0*(9.9 + mediana)/19.8)));

                  valor = valor_eficaz();

                  eficaz= int(valor*100 + .5);

                  valor= eficaz/100.0;                //pone 2 decimales

                  label25->Text= "Valor eficaz: " + valor.ToString() + " v";

                  linea_green(40.0,float(220-(200.0*(9.9 + valor)/19.8)),
400.0, float(220-(200.0*(9.9 + valor)/19.8)));

                  for(int i=0;i< short(Count/divisor); i++)

                        linea_blue(float(40.0 + (i)*360.0/(Count/divisor)),
float(220.0-(200.0*DataArray[i]/4095.0)), float(40.0 + 360.0/(Count/divisor)
*(i+1)),float(220.0-(200.0*DataArray[i]/4095.0)));

                  if(divisor==1)

                        pictureBox2->Visible= false;

                  else

                        pictureBox2->Visible= true;

            }

      }

      else if(comboBox1->Text=="Diente de sierra")

      {
```

```
            if(float(numericUpDown1->Value)+float(numericUpDown2->
Value)>9.9)

            {

                  groupBox3->Visible=false;

                  MessageBox::Show("La suma de offset y amplitud debe estar
entre -9,9 a 9,9 v.", "Error: parámetros erróneos", MessageBoxButtons::OK,
MessageBoxIcon::Error);

            }

            else

            {

                  pictureBox1->Refresh();

                  dibujar_grafico();

                  label19->Text= (2*(float(numericUpDown6->Value))/1000.0).
ToString() + " s";

                  label14->Text= (2*(float(numericUpDown6->Value))/4000.0).
ToString() + " s";

                  label15->Text= (2*(float(numericUpDown6->Value))/2000.0).
ToString() + " s";

                  label16->Text= (6*(float(numericUpDown6->Value))/4000.0).
ToString() + " s";

                  label14->Visible=true;

                  label15->Visible=true;

                  label16->Visible=true;

                  label19->Visible=true;

                  diente(numericUpDown1->Value,numericUpDown2->Value,
numericUpDown6->Value,radioButton1->Checked);

                  mediana = media();

                  media_aritmetica= int(mediana*100 + .5);

                  mediana= media_aritmetica/100.0;

                  label24->Text= "Media: " + mediana.ToString() + " v";

                  linea_red(40.0,float(220-(200.0*(9.9 + mediana)/19.8)),
400.0, float(220-(200.0*(9.9 + mediana)/19.8)));

                  valor = valor_eficaz();

                  eficaz= int(valor*100 + .5);

                  valor= eficaz/100.0;

                  label25->Text= "Valor eficaz: " + valor.ToString() + " v";
```

```
                  linea_green(40.0,float(220-(200.0*(9.9 + valor)/19.8)),
400.0, float(220-(200.0*(9.9 + valor)/19.8)));

                  for(int i=0;i< short(Count/divisor); i++)

                  {

                        //dibuja de azul la gráfica, 2 periodos

                        linea_blue(float(40.0 + 180.0/(Count/divisor)*(i)),
float(220.0-(200.0*DataArray[i]/4095.0)),float(40.0 + 180.0/(Count/divisor)
*(i+1)), float(220.0-(200.0*DataArray[i]/4095.0)));

                        linea_blue(float(220.0 + 180.0/(Count/divisor)*(i)),
float(220.0-(200.0*DataArray[i]/4095.0)),float(220.0 + 180.0/(Count/divisor)
*(i+1)), float(220.0-(200.0*DataArray[i]/4095.0)));

                  }

                  if(divisor==1)

                        pictureBox2->Visible= false;

                  else

                        pictureBox2->Visible= true;

            }

      }

      else if(comboBox1->Text=="Trapezoidal")

      {

            if(float(numericUpDown1->Value)+float(numericUpDown2->
Value)>9.9)

            {

                  groupBox3->Visible=false;

                  MessageBox::Show("La suma de offset y amplitud debe estar
entre -9,9 a 9,9 v.", "Error: parámetros erróneos", MessageBoxButtons::OK,
MessageBoxIcon::Error);

            }

            else

            {

                  pictureBox1->Refresh();

                  dibujar_grafico();

                  label19->Text= ((float(numericUpDown5->Value) + float
(numericUpDown6->Value) + float(numericUpDown7->Value)+ float(numericUpDown8
->Value))/1000.0).ToString() + " s";

                  label14->Text= ((float(numericUpDown5->Value) + float
(numericUpDown6->Value)+ float(numericUpDown7->Value)+ float(numericUpDown8
->Value))/4000.0).ToString() + " s";
```

```
                  label15->Text= ((float(numericUpDown5->Value) + float
(numericUpDown6->Value)+ float(numericUpDown7->Value)+ float(numericUpDown8
->Value))/2000.0).ToString() + " s";

                  label16->Text= (3*(float(numericUpDown5->Value) + float
(numericUpDown6->Value)+ float(numericUpDown7->Value)+ float(numericUpDown8
->Value))/4000.0).ToString() + " s";

                  label14->Visible=true;

                  label15->Visible=true;

                  label16->Visible=true;

                  label19->Visible=true;

                  trapezoidal(numericUpDown1->Value,numericUpDown2->
Value,numericUpDown5->Value,numericUpDown6->Value,numericUpDown7->
Value,numericUpDown8->Value);

                  mediana = media();

                  media_aritmetica= int(mediana*100 + .5);

                  mediana= media_aritmetica/100.0;    //pone 2 decimales

                  label24->Text= "Media: " + mediana.ToString() + " v";

                  linea_red(40.0,float(220-(200.0*(9.9 + mediana)/19.8)),
400.0, float(220-(200.0*(9.9 + mediana)/19.8)));

                  valor = valor_eficaz();

                  eficaz= int(valor*100 + .5);

                  valor= eficaz/100.0;          //pone 2 decimales

                  label25->Text= "Valor eficaz: " + valor.ToString() + " v";

                  linea_green(40.0,float(220-(200.0*(9.9 + valor)/19.8)),
400.0, float(220-(200.0*(9.9 + valor)/19.8)));

                  for(int i=0;i< short(Count/divisor); i++)

                        linea_blue(float(40.0 + (i)*360.0/(Count/divisor)),
float(220.0-(200.0*DataArray[i]/4095.0)), float(40.0 + 360.0/(Count/divisor)
*(i+1)),float(220.0-(200.0*DataArray[i]/4095.0)));

                  if(divisor==1)

                        pictureBox2->Visible= false;

                  else

                        pictureBox2->Visible= true;

            }

      }

      else if(comboBox1->Text=="Sinusoidal")

      {
```

```
            if(float(numericUpDown1->Value)+float(numericUpDown2->Value)>9.9
|| -(float(numericUpDown1->Value))+float(numericUpDown2->Value)<-9.9)
//debe estar entre -9.9 a 9.9 V

            {

                  groupBox3->Visible=false;

                  MessageBox::Show("La suma de offset y amplitud debe estar
entre -9,9 a 9,9 v.","Error: parámetros erróneos", MessageBoxButtons::OK,
MessageBoxIcon::Error);

            }

            else

            {

                  pictureBox1->Refresh();

                  dibujar_grafico();

                  label19->Text= (1/float(numericUpDown3->Value)).ToString()
+ " s";

                  label14->Text= ((1/float(numericUpDown3->Value))/4).
ToString() + " s";

                  label15->Text= ((1/float(numericUpDown3->Value))/2).
ToString() + " s";

                  label16->Text= (3*(1/float(numericUpDown3->Value))/4).
ToString() + " s";

                  label14->Visible=true;

                  label15->Visible=true;

                  label16->Visible=true;

                  label19->Visible=true;

                  sinusoidal(numericUpDown1->Value,numericUpDown2->
Value,numericUpDown3->Value,numericUpDown4->Value);

                  mediana = media();

                  media_aritmetica= int(mediana*100 + .5);

                  mediana= media_aritmetica/100.0;

                  label24->Text= "Media: " + mediana.ToString() + " v";

                  linea_red(40.0,float(220-(200.0*(9.9 + mediana)/19.8)),
400.0, float(220-(200.0*(9.9 + mediana)/19.8)));

                  valor = valor_eficaz();

                  eficaz= int(valor*100 + .5);

                  valor= eficaz/100.0;

                  label25->Text= "Valor eficaz: " + valor.ToString() + " v";
```

```
                  linea_green(40.0,float(220-(200.0*(9.9 + valor)/19.8)),
400.0, float(220-(200.0*(9.9 + valor)/19.8)));

                  for(int i=0;i< short(Count/divisor); i++)

                        linea_blue(float(40.0 + (i)*360.0/(Count/divisor)),
float(220.0-(200.0*DataArray[i]/4095.0)), float(40.0 + 360.0/(Count/divisor)
*(i+1)),float(220.0-(200.0*DataArray[i]/4095.0)));

                  if(divisor==1)

                        pictureBox2->Visible= false;

                  else

                        pictureBox2->Visible= true;

            }

      }

      else if(comboBox1->Text=="Irregular")

      {

            if(float(numericUpDown1->Value)+float(numericUpDown2->
Value)>9.9)

            {

                  groupBox3->Visible=false;

                  MessageBox::Show("La suma de offset y amplitud debe estar
entre -9,9 a 9,9 v.","Error: parámetros erróneos", MessageBoxButtons::OK,
MessageBoxIcon::Error);

            }

            else

            {

                  pictureBox1->Refresh();

                  dibujar_grafico();

                  label19->Text=  "1 s";

                  label14->Text=  "0.25 s";

                  label15->Text= "0.5 s";

                  label16->Text=  "0.75 s";

                  label14->Visible=true;

                  label15->Visible=true;

                  label16->Visible=true;

                  label19->Visible=true;

                  irregular(numericUpDown1->Value,numericUpDown2->
Value,numericUpDown3->Value);
```

```
                  mediana = media();

                  media_aritmetica= int(mediana*100 + .5);

                  mediana= media_aritmetica/100.0;    //pone 2 decimales

                  label24->Text= "Media: " + mediana.ToString() + " v";

                  linea_red(40.0,float(220-(200.0*(9.9 + mediana)/19.8)),
400.0, float(220-(200.0*(9.9 + mediana)/19.8)));

                  valor = valor_eficaz();

                  eficaz= int(valor*100 + .5);

                  valor= eficaz/100.0;                //pone 2 decimales

                  label25->Text= "Valor eficaz: " + valor.ToString() + " v";

                  linea_green(40.0,float(220-(200.0*(9.9 + valor)/19.8)),
400.0, float(220-(200.0*(9.9 + valor)/19.8)));

                  for(int i=0;i<int(numericUpDown3->Value); i++)

                        linea_blue(float(40.0 + (i)*360.0/int(numericUpDown3-
>Value)),float(220.0-(200.0*DataArray[i]/4095.0)), float(40.0 + 360.0/
int(numericUpDown3->Value)*(i+1)),float(220.0-(200.0*DataArray[i]/4095.0)));

                  pictureBox2->Visible= false;

            }

      }

      else if(comboBox1->Text=="Personalizada")

      {

            if(comboBox2->Text!="Cambios pulsantes" && comboBox2-
>Text!="Cambios progresivos")

            {     //Si no es ninguna de las 2 opciones dadas nos sale una
ventana de error

                  groupBox3->Visible=false;

                  MessageBox::Show("Debes de indicar como quieres la onda si
cambio progresivo o pulsante.","Error: parámetros erróneos",
MessageBoxButtons::OK, MessageBoxIcon::Error);

            }

            else

            {

                  label24->Text="";

                  label25->Text="";

                  pictureBox1->Refresh();

                  dibujar_grafico();
```

```
                  label19->Text= (float(numericUpDown9->Value)).ToString() +
" s";

                  label14->Text= (float(numericUpDown9->Value)/4).ToString()
+ " s";

                  label15->Text= (float(numericUpDown9->Value)/2).ToString()
+ " s";

                  label16->Text= (3*float(numericUpDown9->Value)/4).
ToString() + " s";

                  label14->Visible=true;

                  label15->Visible=true;

                  label16->Visible=true;

                  label19->Visible=true;

                  if(comboBox2->Text=="Cambios pulsantes")

                        personal(numericUpDown9->Value,1);

                  if(comboBox2->Text=="Cambios progresivos")

                        personal(numericUpDown9->Value,2);

                  mediana = media();

                  media_aritmetica= int(mediana*100 + .5);

                  mediana= media_aritmetica/100.0;    //pone 2 decimales

                  label24->Text= "Media: " + mediana.ToString() + " v";

                  linea_red(40.0,float(220-(200.0*(9.9 + mediana)/19.8)),
400.0, float(220-(200.0*(9.9 + mediana)/19.8)));

                  valor = valor_eficaz();

                  eficaz= int(valor*100 + .5);

                  valor= eficaz/100.0;                //pone 2 decimales

                  label25->Text= "Valor eficaz: " + valor.ToString() + " v";

                  linea_green(40.0,float(220-(200.0*(9.9 + valor)/19.8)),
400.0, float(220-(200.0*(9.9 + valor)/19.8)));

                  for(int i=0;i< short(Count/divisor); i++)

                        linea_blue(float(40.0 + (i)*360.0/(Count/divisor)),
float(220.0-(200.0*DataArray[i]/4095.0)), float(40.0 + 360.0/(Count/divisor)
*(i+1)),float(220.0-(200.0*DataArray[i]/4095.0)));

                  pictureBox2->Visible= false;

                  funcionar();

            }

      }
```

```
      else   //Si no es una de esas opciones que nos de una ventana de error
            MessageBox::Show("Debe escoger una función de las dadas en la
lista.", "Error: esa función no existe", MessageBoxButtons::OK,
MessageBoxIcon::Error);
}
```

Figura 15.25: Evento al presionar el componente button2, dentro del archivo prueba3.cpp

```
System::Void button3_Click(System::Object^  sender, System::EventArgs^  e)
{                       //Al apretar el button3, el botón de salir
      cbStopBackground(0, AOFUNCTION);   //Paramos las conversiones
      inicializacion();                  //Ponemos a 0 la salida
      delete [] Personalizar;      //Elimina el espacio guardado en memoria
      delete [] Texto;
      Form1::Close();                    //Cierra form1
}
```

Figura 15.26: Evento al presionar el componente button3, dentro del archivo prueba3.cpp

```
System::Void button4_Click(System::Object^  sender, System::EventArgs^  e)
{                           //Al apretar el button4, el botón de stop
      cbStopBackground(0, AOFUNCTION);   //Paramos las converiones
      button4->Visible=false;
      inicializacion();                       //Ponemos a 0 la salida
}
```

Figura 15.27: Evento al presionar el componente button4, dentro del archivo prueba3.cpp

```
//Al cambiar el valor del comboBox1, el selector de función
System::Void comboBox1_SelectedIndexChanged_1(System::Object^  sender,
System::EventArgs^  e)
{
      cbStopBackground(0, AOFUNCTION);    //Paramos la conversion
      inicializacion();                   //Ponemos a 0 la salida
```

```
        button2->Enabled= true;            /*Hacemos invisible los
        componenetes que no interesa para que quede todo como inicialmente */

        groupBox2->Visible=false;

        groupBox3->Visible=false;

        label24->Visible=false;

        label25->Visible=false;

        pictureBox2->Visible=false;

}
```

Figura 15.28: Evento al cambiar el valor del componente comboBox1, dentro del archivo prueba3.cpp

```
System::Void button5_Click(System::Object^  sender, System: :EventArgs^  e)
{
        //Sale un mensaje explicativo de la aplicación

        MessageBox::Show("Generador de funciones \nUniversitat Rovira i
        Virgili \n 2009 \n \n Proyecto Final de carrera de: \n Héctor Zazo
        Jiménez", "Generador de funciones", MessageBoxButtons::OK,
        MessageBoxIcon::Information);

}
```

Figura 15.29: Evento al presionar el componente button5, dentro del archivo prueba3.cpp

```
//Al apretar cualquier imagen de "función personalizada"

System::Void picture_Click(System::Object^  sender, System: :EventArgs^  e)
{
        for(int i=0;i<210;i++){

                if(pictureBox[i]->Capture)    //Detecta que botón fue apretado

                        pulsar=i;              //Lo guardamos

        }

        graf(pulsar);                    //Y llamamos a graf() con ese valor

}
```

Figura 15.30: Evento al presionar picture, dentro del archivo prueba3.cpp

El archivo siguiente, form1.h, está dividido por partes para que sea más fácil su comprensión. Las partes están ordenadas según el código original, es decir, la suma de las partes quedaría el archivo form1.h completo y ordenado según la versión final.

```
#pragma once
#include "stdafx.h"
#include "C:/MCC/CWIN/cbw.h"
#include <cstdlib>
#include <ctime>
#include <iostream>
#include "math.h"

namespace prueba3
{
	using namespace System;
	using namespace System::ComponentModel;
	using namespace System::Collections;
	using namespace System::Windows::Forms;
	using namespace System::Data;
	using namespace System::Drawing;
	public ref class Form1 : public System::Windows::Forms::Form
	{
```

Figura 15.31: Librerías e inicio de la clase form1 del archivo form1.h

```
	//-----------VARIABLES----------//
	long Count;
	unsigned short *DataArray;
	long Rate;
	HGLOBAL MemHandle;
	short Status;
	long CurCount;
	long CurIndex;
```

```
        short pulsar;
        short divisor;
        int valor;
        unsigned short *Personalizar;

        private: System::Windows::Forms::Button^  button3;
        private: System::Windows::Forms::Label^  label9;
        private: System::Windows::Forms::Label^  label10;
        private: System::Windows::Forms::Label^  label13;
        private: System::Windows::Forms::Label^  label12;
        private: System::Windows::Forms::Label^  label11;
        private: System::Windows::Forms::Label^  label16;
        private: System::Windows::Forms::Label^  label15;
        private: System::Windows::Forms::Label^  label14;
        private: System::Windows::Forms::Label^  label17;
        private: System::Windows::Forms::Label^  label18;
        private: System::Windows::Forms::Label^  label19;
        private: System::Windows::Forms::Label^  label20;
        private: System::Windows::Forms::Label^  label21;
        private: System::Windows::Forms::Label^  label22;
        private: System::Windows::Forms::Label^  label23;
        private: System::Windows::Forms::Button^  button5;
        private: System::Windows::Forms::Label^  label24;
        private: System::Windows::Forms::Label^  label25;
        private: cli::array<System::Windows::Forms::PictureBox^>
^pictureBox;
        private: cli::array<System::Windows::Forms::Label^>  ^Texto;
        private: System::Windows::Forms::Label^  label26;
        private: System::Windows::Forms::NumericUpDown^  numericUpDown9;
        private: System::Windows::Forms::ComboBox^  comboBox2;
        private: System::Windows::Forms::Label^  label27;
        private: System::Windows::Forms::Button^  button4;
        protected:
```

```
private: System::Windows::Forms::ComboBox^  comboBox1;
private: System::Windows::Forms::Button^  button1;
private: System::Windows::Forms::GroupBox^  groupBox1;
private: System::Windows::Forms::GroupBox^  groupBox2;
private: System::Windows::Forms::NumericUpDown^  numericUpDown1;
private: System::Windows::Forms::Label^  label1;
private: System::Windows::Forms::NumericUpDown^  numericUpDown2;
private: System::Windows::Forms::Label^  label2;
private: System::Windows::Forms::Label^  label3;
private: System::Windows::Forms::NumericUpDown^  numericUpDown3;
private: System::Windows::Forms::Label^  label4;
private: System::Windows::Forms::NumericUpDown^  numericUpDown4;
private: System::Windows::Forms::Label^  label5;
private: System::Windows::Forms::Button^  button2;
private: System::Windows::Forms::NumericUpDown^  numericUpDown5;
private: System::Windows::Forms::NumericUpDown^  numericUpDown6;
private: System::Windows::Forms::RadioButton^  radioButton2;
private: System::Windows::Forms::RadioButton^  radioButton1;
private: System::Windows::Forms::NumericUpDown^  numericUpDown7;
private: System::Windows::Forms::Label^  label6;
private: System::Windows::Forms::NumericUpDown^  numericUpDown8;
private: System::Windows::Forms::Label^  label7;
private: System::Windows::Forms::Label^  label8;
private: System::Windows::Forms::GroupBox^  groupBox3;
private: System::Windows::Forms::PictureBox^  pictureBox1;
private: System::Windows::Forms::PictureBox^  pictureBox2;
private: System::ComponentModel::IContainer^  components;
```

Figura 15.32: Declaración de variables globales y de componentes que se usaran en la clase form1 del archivo form1.h

```
System::Void vector(void);

System::Void funcionar(void);

System::Void graf(short vari);

System::Void ver_person(bool visible);

System::Void timer(float frec);

System::Void continua(System::Decimal amplitud);

System::Void cuadrada(System::Decimal amplitud, System::Decimal offset,
System::Decimal frecuencia, System::Decimal dutycicle);

System::Void triangular(System::Decimal amplitud, System::Decimal offset,
System::Decimal tiempo_s, System::Decimal tiempo_b);

System::Void diente(System::Decimal amplitud, System::Decimal offset,
System::Decimal tiempo, System::Boolean option);

System::Void trapezoidal(System::Decimal amplitud, System::Decimal offset,
System::Decimal tiempo_s, System::Decimal tiempo_b, System::Decimal
tiempo_ON, System::Decimal tiempo_OFF);

System::Void sinusoidal(System::Decimal amplitud, System::Decimal offset,
System::Decimal frecuencia, System::Decimal desfase);

System::Void personal(System::Decimal tiempo, int opcion);

System::Void irregular(System::Decimal amplitud, System::Decimal offset,
System::Decimal velocidad);

double media(void);

double valor_eficaz(void);

System::Void inicializacion(void);

System::Void dibujar_grafico(void);

System::Void linea_blue(float x, float y, float x1, float y1 );

System::Void linea_green(float x, float y, float x1, float y1 );

System::Void linea_red(float x, float y, float x1, float y1 );

System::Void linea_black(int x, int y, int x1, int y1 );

System::Void linea_black_(int x, int y, int x1, int y1 );

System::Void button1_Click_1(System::Object^  sender, System::EventArgs^
e);

System::Void radioButton1_CheckedChanged(System::Object^  sender,
System::EventArgs^  e);

System::Void button2_Click(System::Object^  sender, System::EventArgs^  e);

System::Void button3_Click(System::Object^ sender, System::EventArgs^  e);

System::Void button4_Click(System::Object^  sender, System::EventArgs^  e);
```

```
System::Void comboBox1_SelectedIndexChanged_1(System::Object^  sender,
System::EventArgs^  e);

System::Void button5_Click(System::Object^  sender, System: :EventArgs^  e);

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

Figura 15.33: Declaración de las funciones que se utilizaran en el programa principal (prueba3.cpp). Estas declaraciones se encuentran en form1.h

```
public:
            Form1(void)                              //Al abrirse
            {
                  vector();
                  InitializeComponent();
            }
      protected:
            ~Form1()                                    //Al cerrarse
            {
                  if (components)
                  {
                        delete components;
                  }
            }
```

Figura 15.34: Funciones cuando se abre form1 y cuando se cierra dentro del archivo form1.h

```
private:
#pragma region Windows Form Designer generated code
void InitializeComponent(void)           //Creación de todos los componentes
{
      System::ComponentModel::ComponentResourceManager^  resources = (gcnew
      System::ComponentModel::ComponentResourceManager(Form1::typeid));
      this->comboBox1 = (gcnew System::Windows::Forms::ComboBox());
      this->button1 = (gcnew System::Windows::Forms::Button());
      this->groupBox1 = (gcnew System::Windows::Forms::GroupBox());
```

```
this->button5 = (gcnew System::Windows::Forms::Button());
this->groupBox2 = (gcnew System::Windows::Forms::GroupBox());
this->label26 = (gcnew System::Windows::Forms::Label());
this->label23 = (gcnew System::Windows::Forms::Label());
this->label22 = (gcnew System::Windows::Forms::Label());
this->label21 = (gcnew System::Windows::Forms::Label());
this->label20 = (gcnew System::Windows::Forms::Label());
this->label18 = (gcnew System::Windows::Forms::Label());
this->label17 = (gcnew System::Windows::Forms::Label());
this->button4 = (gcnew System::Windows::Forms::Button());
this->label8 = (gcnew System::Windows::Forms::Label());
this->numericUpDown8=(gcnew System::Windows::Forms::NumericUpDown());
this->label7 = (gcnew System::Windows::Forms::Label());
this->numericUpDown7=(gcnew System::Windows::Forms::NumericUpDown());
this->numericUpDown5=(gcnew System::Windows::Forms::NumericUpDown());
this->label6 = (gcnew System::Windows::Forms::Label());
this->radioButton2 = (gcnew System::Windows::Forms::RadioButton());
this->numericUpDown6=(gcnew System::Windows::Forms::NumericUpDown());
this->radioButton1 = (gcnew System::Windows::Forms::RadioButton());
this->button2 = (gcnew System::Windows::Forms::Button());
this->numericUpDown4=(gcnew System::Windows::Forms::NumericUpDown());
this->label5 = (gcnew System::Windows::Forms::Label());
this->numericUpDown3=(gcnew System::Windows::Forms::NumericUpDown());
this->label4 = (gcnew System::Windows::Forms::Label());
this->label3 = (gcnew System::Windows::Forms::Label());
this->label2 = (gcnew System::Windows::Forms::Label());
this->numericUpDown2=(gcnew System::Windows::Forms::NumericUpDown());
this->numericUpDown1=(gcnew System::Windows::Forms::NumericUpDown());
this->label1 = (gcnew System::Windows::Forms::Label());
this->groupBox3 = (gcnew System::Windows::Forms::GroupBox());
this->label19 = (gcnew System::Windows::Forms::Label());
this->label16 = (gcnew System::Windows::Forms::Label());
this->label15 = (gcnew System::Windows::Forms::Label());
```

```
this->label14 = (gcnew System::Windows::Forms::Label());

this->label13 = (gcnew System::Windows::Forms::Label());

this->label12 = (gcnew System::Windows::Forms::Label());

this->label11 = (gcnew System::Windows::Forms::Label());

this->label10 = (gcnew System::Windows::Forms::Label());

this->label9 = (gcnew System::Windows::Forms::Label());

this->pictureBox1 = (gcnew System::Windows::Forms::PictureBox());

this->pictureBox2 = (gcnew System::Windows::Forms::PictureBox());

this->button3 = (gcnew System::Windows::Forms::Button());

this->label24 = (gcnew System::Windows::Forms::Label());

this->label25 = (gcnew System::Windows::Forms::Label());

this->numericUpDown9=(gcnew System::Windows::Forms::NumericUpDown());

this->label27 = (gcnew System::Windows::Forms::Label());

this->comboBox2 = (gcnew System::Windows::Forms::ComboBox());

this->groupBox1->SuspendLayout();

this->groupBox2->SuspendLayout();

//Los componentes numericUpDown y pictureBox necesitan crearse e
inicializarse

(cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->
numericUpDown8))->BeginInit();

(cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->
numericUpDown7))->BeginInit();

(cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->
numericUpDown5))->BeginInit();

(cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->
numericUpDown6))->BeginInit();

(cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->
numericUpDown4))->BeginInit();

(cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->
numericUpDown3))->BeginInit();

(cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->
numericUpDown2))->BeginInit();

(cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->
numericUpDown1))->BeginInit();

for(int i=0;i<210;i++)

      (cli::safe_cast<System::ComponentModel::ISupportInitialize^>
(this->pictureBox[i]))->BeginInit();
```

```
this->groupBox3->SuspendLayout();

(cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->
pictureBox1))->BeginInit();

(cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this-
>npictureBox2))->BeginInit();

this->SuspendLayout();
```

Figura 15.35: Función InitializeComponent() parte de creación e inicialización, del archivo form1.h

```
// comboBox1

this->comboBox1->FormattingEnabled = true;

this->comboBox1->Items->AddRange(gcnew cli::array< System::Object^
>(8) {L"Continua", L"Cuadrada", L"Triangular", L"Diente de sierra",
L"Trapezoidal", L"Sinusoidal", L"Irregular", L"Personalizada"});
//Items seleccionables

this->comboBox1->Location = System::Drawing::Point(6, 19);  //Posición
donde crearse

this->comboBox1->Name = L"comboBox1";      //Nombre del componente

this->comboBox1->Size = System::Drawing::Size(139, 21);     //Tamaño

this->comboBox1->TabIndex = 1;           //Índice de prioridad por si
hay 2 en el mismo lugar

this->comboBox1->Text = L"Escoja tipo de función";    //Texto

this->comboBox1->SelectedIndexChanged += gcnew System::EventHandler
(this, &Form1::comboBox1_SelectedIndexChanged_1);     //Crear evento
al cambiarse

// button1

this->button1->Location = System::Drawing::Point(163, 17);

this->button1->Name = L"button1";

this->button1->Size = System::Drawing::Size(75, 23);

this->button1->TabIndex = 2;

this->button1->Text = L"Aceptar";

this->button1->UseVisualStyleBackColor = true;

this->button1->Click += gcnew System::EventHandler(this, &Form1:
:button1_Click_1);  //Crea evento

// groupBox1

this->groupBox1->Controls->Add(this->button5);  //Componentes añadidos
dentro del groupBox
```

```
this->groupBox1->Controls->Add(this->comboBox1);

this->groupBox1->Controls->Add(this->button1);

this->groupBox1->Location = System::Drawing::Point(2, 2);

this->groupBox1->Name = L"groupBox1";

this->groupBox1->Size = System::Drawing::Size(416, 49);

this->groupBox1->TabIndex = 3;

this->groupBox1->TabStop = false;

this->groupBox1->Text = L"Tipo de función";

// pictureBox

for(int i=0;i<210;i++){    //creamos 210 pictureBox en memoria ya
guardada en vector()

      if(i<21)     //Primeros 21 cuadrados de un color

      {

            this->pictureBox[i]->BackColor=System::Drawing::Color:
:Green;

            this->pictureBox[i]->Enabled = true;

      }

      else        //Los siguientes de otro

      {

            this->pictureBox[i]->BackColor= System::Drawing::Color:
:Red;

            this->pictureBox[i]->Enabled = false;

      }

      this->pictureBox[i]->Location = System::Drawing::Point(42+36*
(i/21), 15 +(i%21)*11);  //La localización es en forma de cuadricula
21x10

      this->pictureBox[i]->Size = System::Drawing::Size(8, 8);

      this->pictureBox[i]->Name="pictureBox["+i+"]";

      this->pictureBox[i]->TabIndex= 600 + i;

      this->pictureBox[i]->Click += gcnew System::EventHandler(this,
&Form1::picture_Click);

}

// Texto

for(int i=0;i<21;i++)

{                                          //creamos 21 Label
```

```
      this->Texto[i]->Location = System::Drawing::Point(8, 12 +(i%21)
*11);

      this->Texto[i]->AutoSize = true;

      this->Texto[i]->Name="Texto["+i+"]";

      this->Texto[i]->TabIndex = 1000 + i;

      this->Texto[i]->Text = (10-i)+ " v -";

      this->Texto[i]->Visible= true;

}

// button5

this->button5->Location = System::Drawing::Point(322, 17);

this->button5->Name = L"button5";

this->button5->Size = System::Drawing::Size(75, 23);

this->button5->TabIndex = 3;

this->button5->Text = L"Acerca de";

this->button5->UseVisualStyleBackColor = true;

this->button5->Click += gcnew System::EventHandler(this, &Form1:
:button5_Click);

// groupBox2

this->groupBox2->Controls->Add(this->comboBox2); //componentes
añadidos en groupBox2

this->groupBox2->Controls->Add(this->label27);

this->groupBox2->Controls->Add(this->numericUpDown9);

this->groupBox2->Controls->Add(this->label26);

this->groupBox2->Controls->Add(this->label23);

this->groupBox2->Controls->Add(this->label22);

this->groupBox2->Controls->Add(this->label21);

this->groupBox2->Controls->Add(this->label20);

this->groupBox2->Controls->Add(this->label18);

this->groupBox2->Controls->Add(this->label17);

this->groupBox2->Controls->Add(this->button4);

this->groupBox2->Controls->Add(this->label8);

this->groupBox2->Controls->Add(this->numericUpDown8);

this->groupBox2->Controls->Add(this->label7);

this->groupBox2->Controls->Add(this->numericUpDown7);
```

```
this->groupBox2->Controls->Add(this->numericUpDown5);
this->groupBox2->Controls->Add(this->label6);
this->groupBox2->Controls->Add(this->radioButton2);
this->groupBox2->Controls->Add(this->numericUpDown6);
this->groupBox2->Controls->Add(this->radioButton1);
this->groupBox2->Controls->Add(this->button2);
this->groupBox2->Controls->Add(this->numericUpDown4);
this->groupBox2->Controls->Add(this->label5);
this->groupBox2->Controls->Add(this->numericUpDown3);
this->groupBox2->Controls->Add(this->label4);
for(int i=0; i<210; i++)
      this->groupBox2->Controls->Add(this->pictureBox[i]);
for(int i=0; i<21; i++)
      this->groupBox2->Controls->Add(this->Texto[i]);
this->groupBox2->Controls->Add(this->label3);
this->groupBox2->Controls->Add(this->label2);
this->groupBox2->Controls->Add(this->numericUpDown2);
this->groupBox2->Controls->Add(this->numericUpDown1);
this->groupBox2->Controls->Add(this->label1);
this->groupBox2->Location = System::Drawing::Point(2, 57);
this->groupBox2->Name = L"groupBox2";
this->groupBox2->Size = System::Drawing::Size(416, 281);
this->groupBox2->TabIndex = 4;
this->groupBox2->TabStop = false;
this->groupBox2->Visible = false;
// label26
this->label26->AutoSize = true;
this->label26->Location = System::Drawing::Point(92, 257);
this->label26->Name = L"label26";
this->label26->Size = System::Drawing::Size(45, 13);
this->label26->TabIndex = 25;
this->label26->Text = L"periodo:";
// label23
```

```
this->label23->AutoSize = true;
this->label23->Location = System::Drawing::Point(365, 181);
this->label23->Name = L"label23";
this->label23->Size = System::Drawing::Size(0, 13);
this->label23->TabIndex = 23;
// label22
this->label22->AutoSize = true;
this->label22->Location = System::Drawing::Point(365, 156);
this->label22->Name = L"label22";
this->label22->Size = System::Drawing::Size(0, 13);
this->label22->TabIndex = 22;
// label21
this->label21->AutoSize = true;
this->label21->Location = System::Drawing::Point(365, 130);
this->label21->Name = L"label21";
this->label21->Size = System::Drawing::Size(0, 13);
this->label21->TabIndex = 21;
// label20
this->label20->AutoSize = true;
this->label20->Location = System::Drawing::Point(365, 99);
this->label20->Name = L"label20";
this->label20->Size = System::Drawing::Size(0, 13);
this->label20->TabIndex = 20;
// label18
this->label18->AutoSize = true;
this->label18->Location = System::Drawing::Point(374, 51);
this->label18->Name = L"label18";
this->label18->Size = System::Drawing::Size(13, 13);
this->label18->TabIndex = 19;
this->label18->Text = L"v";
// label17
this->label17->AutoSize = true;
this->label17->Location = System::Drawing::Point(374, 27);
```

```
this->label17->Name = L"label17";

this->label17->Size = System::Drawing::Size(13, 13);

this->label17->TabIndex = 18;

this->label17->Text = L"v";

// button4

this->button4->Location = System::Drawing::Point(10, 252);

this->button4->Name = L"button4";

this->button4->Size = System::Drawing::Size(75, 23);

this->button4->TabIndex = 17;

this->button4->Text = L"Stop";

this->button4->UseVisualStyleBackColor = true;

this->button4->Visible = false;

this->button4->Click += gcnew System::EventHandler(this, &Form1:
:button4_Click);

// label8

this->label8->AutoSize = true;

this->label8->ForeColor = System::Drawing::Color::Red;

this->label8->Location = System::Drawing::Point(20, 150);

this->label8->Name = L"label8";

this->label8->Size = System::Drawing::Size(230, 13);

this->label8->TabIndex = 16;

this->label8->Text = L"(Un desfase de 90º sería una onda cosenoidal)";

// numericUpDown8

this->numericUpDown8->Increment = System::Decimal(gcnew cli::array<
System::Int32 >(4) {5, 0, 0, 0});     //Incrementos de 5 en 5

this->numericUpDown8->Location = System::Drawing::Point(245, 177);

this->numericUpDown8->Maximum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {25000, 0, 0, 0});   //máximo de 25000

this->numericUpDown8->Name = L"numericUpDown8";

this->numericUpDown8->Size = System::Drawing::Size(120, 20);

this->numericUpDown8->TabIndex = 15;

this->numericUpDown8->Value = System::Decimal(gcnew cli::array<
System::Int32 >(4) {500, 0, 0, 0});      //valor inicial de 500

this->numericUpDown8->Visible = false;     //que al inicio NO sea
visible
```

```
// label7

this->label7->AutoSize = true;

this->label7->Location = System::Drawing::Point(24, 179);

this->label7->Name = L"label7";

this->label7->Size = System::Drawing::Size(0, 13);

this->label7->TabIndex = 14;

// numericUpDown7

this->numericUpDown7->Increment = System::Decimal(gcnew cli::array<
System::Int32 >(4) {5, 0, 0, 0});

this->numericUpDown7->Location = System::Drawing::Point(245, 150);

this->numericUpDown7->Maximum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {25000, 0, 0, 0});

this->numericUpDown7->Name = L"numericUpDown7";

this->numericUpDown7->Size = System::Drawing::Size(120, 20);

this->numericUpDown7->TabIndex = 13;

this->numericUpDown7->Value = System::Decimal(gcnew cli::array<
System::Int32 >(4) {500, 0, 0, 0});

this->numericUpDown7->Visible = false;

// numericUpDown5

this->numericUpDown5->Increment = System::Decimal(gcnew cli::array<
System::Int32 >(4) {50, 0, 0, 0});

this->numericUpDown5->Location = System::Drawing::Point(245, 97);

this->numericUpDown5->Maximum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {50000, 0, 0, 0});

this->numericUpDown5->Minimum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {2, 0, 0, 0});

this->numericUpDown5->Name = L"numericUpDown5";

this->numericUpDown5->Size = System::Drawing::Size(120, 20);

this->numericUpDown5->TabIndex = 10;

this->numericUpDown5->Value = System::Decimal(gcnew cli::array<
System::Int32 >(4) {1000, 0, 0, 0});

this->numericUpDown5->Visible = false;

// label6

this->label6->AutoSize = true;

this->label6->Location = System::Drawing::Point(26, 152);

this->label6->Name = L"label6";
```

```
this->label6->Size = System::Drawing::Size(0, 13);

this->label6->TabIndex = 12;

// radioButton2

this->radioButton2->AutoSize = true; //El tamaño dependerá de lo que
ocupe el componente y no del Size puesto

this->radioButton2->Checked = true;

this->radioButton2->Location = System::Drawing::Point(250, 91);

this->radioButton2->Name = L"radioButton2";

this->radioButton2->Size = System::Drawing::Size(109, 17);

this->radioButton2->TabIndex = 6;

this->radioButton2->TabStop = true;

this->radioButton2->Text = L"Tiempo de subida";

this->radioButton2->UseVisualStyleBackColor = true;

// numericUpDown6

this->numericUpDown6->Increment = System::Decimal(gcnew cli::array<
System::Int32 >(4) {5, 0, 0, 0});

this->numericUpDown6->Location = System::Drawing::Point(245, 124);

this->numericUpDown6->Maximum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {50000, 0, 0, 0});

this->numericUpDown6->Minimum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {2, 0, 0, 0});

this->numericUpDown6->Name = L"numericUpDown6";

this->numericUpDown6->Size = System::Drawing::Size(120, 20);

this->numericUpDown6->TabIndex = 11;

this->numericUpDown6->Value = System::Decimal(gcnew cli::array<
System::Int32 >(4) {500, 0, 0, 0});

this->numericUpDown6->Visible = false;

// radioButton1

this->radioButton1->AutoSize = true;

this->radioButton1->Location = System::Drawing::Point(250, 108);

this->radioButton1->Name = L"radioButton1";

this->radioButton1->Size = System::Drawing::Size(110, 17);

this->radioButton1->TabIndex = 5;

this->radioButton1->TabStop = true;

this->radioButton1->Text = L"Tiempo de bajada";
```

```
this->radioButton1->UseVisualStyleBackColor = true;

this->radioButton1->CheckedChanged += gcnew System::EventHandler(this,
&Form1::radioButton1_CheckedChanged);   //Al cambiarse de TRUE a FALSE
o viceversa

// button2

this->button2->Location = System::Drawing::Point(331, 252);

this->button2->Name = L"button2";

this->button2->Size = System::Drawing::Size(75, 23);

this->button2->TabIndex = 9;

this->button2->Text = L"Aceptar";

this->button2->UseVisualStyleBackColor = true;

this->button2->Click += gcnew System::EventHandler(this, &Form1:
:button2_Click);

// numericUpDown4

this->numericUpDown4->DecimalPlaces = 1;

this->numericUpDown4->Location = System::Drawing::Point(246, 124);

this->numericUpDown4->Name = L"numericUpDown4";

this->numericUpDown4->Size = System::Drawing::Size(120, 20);

this->numericUpDown4->TabIndex = 8;

this->numericUpDown4->Visible = false;

// label5

this->label5->AutoSize = true;

this->label5->Location = System::Drawing::Point(26, 126);

this->label5->Name = L"label5";

this->label5->Size = System::Drawing::Size(0, 13);

this->label5->TabIndex = 7;

// numericUpDown3

this->numericUpDown3->DecimalPlaces = 3;

this->numericUpDown3->Increment = System::Decimal(gcnew cli::array<
System::Int32 >(4) {1, 0, 0, 65536});

this->numericUpDown3->Location = System::Drawing::Point(246, 97);

this->numericUpDown3->Maximum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {10000, 0, 0, 0});

this->numericUpDown3->Minimum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {1, 0, 0, 65536});

this->numericUpDown3->Name = L"numericUpDown3";
```

```
this->numericUpDown3->Size = System::Drawing::Size(119, 20);

this->numericUpDown3->TabIndex = 6;

this->numericUpDown3->Value = System::Decimal(gcnew cli::array<
System::Int32 >(4) {1, 0, 0, 0});

this->numericUpDown3->Visible = false;

// label4

this->label4->AutoSize = true;

this->label4->Location = System::Drawing::Point(22, 99);

this->label4->Name = L"label4";

this->label4->Size = System::Drawing::Size(0, 13);

this->label4->TabIndex = 5;

// label3

this->label3->AutoSize = true;

this->label3->ForeColor = System::Drawing::Color::Red;

this->label3->Location = System::Drawing::Point(17, 72);

this->label3->Name = L"label3";

this->label3->Size = System::Drawing::Size(290, 13);

this->label3->TabIndex = 4;

this->label3->Text = L"(La suma del offset y la amplitud debe estar
entre -9,9 y 9,9.)";

// label2

this->label2->AutoSize = true;

this->label2->Location = System::Drawing::Point(26, 51);

this->label2->Name = L"label2";

this->label2->Size = System::Drawing::Size(0, 13);

this->label2->TabIndex = 3;

// numericUpDown2

this->numericUpDown2->DecimalPlaces = 2;

this->numericUpDown2->Increment = System::Decimal(gcnew cli::array<
System::Int32 >(4) {5, 0, 0, 65536});

this->numericUpDown2->Location = System::Drawing::Point(246, 49);

this->numericUpDown2->Maximum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {99, 0, 0, 65536});

this->numericUpDown2->Minimum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {99, 0, 0, -2147418112});
```

```
this->numericUpDown2->Name = L"numericUpDown2";

this->numericUpDown2->Size = System::Drawing::Size(120, 20);

this->numericUpDown2->TabIndex = 2;

this->numericUpDown2->Visible = false;

// numericUpDown1

this->numericUpDown1->DecimalPlaces = 2;

this->numericUpDown1->Increment = System::Decimal(gcnew cli::array<
System::Int32 >(4) {5, 0, 0, 65536});

this->numericUpDown1->Location = System::Drawing::Point(246, 23);

this->numericUpDown1->Maximum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {99, 0, 0, 65536});

this->numericUpDown1->Minimum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {99, 0, 0, -2147418112});

this->numericUpDown1->Name = L"numericUpDown1";

this->numericUpDown1->Size = System::Drawing::Size(120, 20);

this->numericUpDown1->TabIndex = 1;

this->numericUpDown1->Visible = false;

// label1

this->label1->AutoSize = true;

this->label1->Location = System::Drawing::Point(17, 25);

this->label1->Name = L"label1";

this->label1->Size = System::Drawing::Size(0, 13);

this->label1->TabIndex = 0;

// groupBox3

this->groupBox3->Controls->Add(this->label19);

this->groupBox3->Controls->Add(this->label16);

this->groupBox3->Controls->Add(this->label15);

this->groupBox3->Controls->Add(this->label14);

this->groupBox3->Controls->Add(this->label13);

this->groupBox3->Controls->Add(this->label12);

this->groupBox3->Controls->Add(this->label11);

this->groupBox3->Controls->Add(this->label10);

this->groupBox3->Controls->Add(this->label9);

this->groupBox3->Controls->Add(this->pictureBox1);
```

```
this->groupBox3->Location = System::Drawing::Point(432, 14);
this->groupBox3->Name = L"groupBox3";
this->groupBox3->Size = System::Drawing::Size(417, 266);
this->groupBox3->TabIndex = 5;
this->groupBox3->TabStop = false;
this->groupBox3->Text = L"Gráfico";
this->groupBox3->Visible = false;
// label19
this->label19->AutoSize = true;
this->label19->Location = System::Drawing::Point(375, 17);
this->label19->Name = L"label19";
this->label19->Size = System::Drawing::Size(41, 13);
this->label19->TabIndex = 9;
this->label19->Text = L"label19";
// label16
this->label16->AutoSize = true;
this->label16->Location = System::Drawing::Point(309, 18);
this->label16->Name = L"label16";
this->label16->Size = System::Drawing::Size(33, 13);
this->label16->TabIndex = 8;
this->label16->Text = L"100 s";
this->label16->Visible = false;
// label15
this->label15->AutoSize = true;
this->label15->Location = System::Drawing::Point(212, 18);
this->label15->Name = L"label15";
this->label15->Size = System::Drawing::Size(27, 13);
this->label15->TabIndex = 7;
this->label15->Text = L"12 s";
this->label15->Visible = false;
// label14
this->label14->AutoSize = true;
this->label14->Location = System::Drawing::Point(110, 19);
```

```
this->label14->Name = L"label14";
this->label14->Size = System::Drawing::Size(41, 13);
this->label14->TabIndex = 6;
this->label14->Text = L"100 ms";
this->label14->Visible = false;
// label13
this->label13->AutoSize = true;
this->label13->ForeColor = System::Drawing::SystemColors::ControlText;
this->label13->Location = System::Drawing::Point(7, 180);
this->label13->Name = L"label13";
this->label13->Size = System::Drawing::Size(25, 13);
this->label13->TabIndex = 5;
this->label13->Text = L"-5 v";
// label12
this->label12->AutoSize = true;
this->label12->ForeColor = System::Drawing::SystemColors::ControlText;
this->label12->Location = System::Drawing::Point(9, 82);
this->label12->Name = L"label12";
this->label12->Size = System::Drawing::Size(22, 13);
this->label12->TabIndex = 4;
this->label12->Text = L"5 v";
// label11
this->label11->AutoSize = true;
this->label11->Location = System::Drawing::Point(3, 231);
this->label11->Name = L"label11";
this->label11->Size = System::Drawing::Size(31, 13);
this->label11->TabIndex = 3;
this->label11->Text = L"-10 v";
// label10
this->label10->AutoSize = true;
this->label10->Location = System::Drawing::Point(12, 129);
this->label10->Name = L"label10";
this->label10->Size = System::Drawing::Size(22, 13);
```

```
this->label10->TabIndex = 2;

this->label10->Text = L"0 v";

// label9

this->label9->AutoSize = true;

this->label9->ForeColor = System::Drawing::Color::Black;

this->label9->Location = System::Drawing::Point(8, 32);

this->label9->Name = L"label9";

this->label9->Size = System::Drawing::Size(28, 13);

this->label9->TabIndex = 1;

this->label9->Text = L"10 v";

// pictureBox1

this->pictureBox1->Location = System::Drawing::Point(7, 17);

this->pictureBox1->Name = L"pictureBox1";

this->pictureBox1->Size = System::Drawing::Size(400, 240);

this->pictureBox1->TabIndex = 0;

this->pictureBox1->TabStop = false;

// pictureBox2

this->pictureBox2->ErrorImage=(cli::safe_cast<System::Drawing::Image^
>(resources->GetObject(L"pictureBox2.ErrorImage")));    //Imagen si
falla al cargar

this->pictureBox2->ImageLocation = L"/luz.png";        //Imagen que
queremos poner

this->pictureBox2->InitialImage=(cli::safe_cast<System::Drawing:
:Image^ > (resources->GetObject(L"pictureBox2.InitialImage")));
//Imagen inicial

this->pictureBox2->Location = System::Drawing::Point(424, 312);

this->pictureBox2->Name = L"pictureBox2";

this->pictureBox2->Size = System::Drawing::Size(20, 20);

this->pictureBox2->TabIndex = 17;

this->pictureBox2->TabStop = false;

this->pictureBox2->Visible = false;

// button3

this->button3->Image=(cli::safe_cast<System::Drawing::Image^
>(resources->GetObject(L"button3.Image")));

this->button3->ImageAlign=System::Drawing::ContentAlignment:
:MiddleLeft;
```

```
this->button3->Location = System::Drawing::Point(764, 309);

this->button3->Name = L"button3";

this->button3->RightToLeft = System::Windows::Forms::RightToLeft::No;

this->button3->Size = System::Drawing::Size(66, 37);

this->button3->TabIndex = 19;

this->button3->Text = L"Salir";

this->button3->TextAlign=System::Drawing::ContentAlignment:
:MiddleRight;

this->button3->UseVisualStyleBackColor = true;

this->button3->Click += gcnew System::EventHandler(this, &Form1:
:button3_Click);

// label24

this->label24->AutoSize = true;

this->label24->ForeColor = System::Drawing::Color::Red;

this->label24->Location = System::Drawing::Point(484, 279);

this->label24->Name = L"label24";

this->label24->Size = System::Drawing::Size(0, 13);

this->label24->TabIndex = 20;

// label25

this->label25->AutoSize = true;

this->label25->ForeColor = System::Drawing::Color::Green;

this->label25->Location = System::Drawing::Point(608, 278);

this->label25->Name = L"label25";

this->label25->Size = System::Drawing::Size(0, 13);

this->label25->TabIndex = 21;

// numericUpDown9

this->numericUpDown9->DecimalPlaces = 1;

this->numericUpDown9->Location = System::Drawing::Point(130, 255);

this->numericUpDown9->Maximum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {50, 0, 0, 0});

this->numericUpDown9->Minimum = System::Decimal(gcnew cli::array<
System::Int32 >(4) {2, 0, 0, 65536});

this->numericUpDown9->Name = L"numericUpDown9";

this->numericUpDown9->Size = System::Drawing::Size(53, 20);

this->numericUpDown9->TabIndex = 26;
```

```
this->numericUpDown9->Value = System::Decimal(gcnew cli::array<
System::Int32 >(4) {1, 0, 0, 0});

// label27

this->label27->AutoSize = true;

this->label27->Location = System::Drawing::Point(186, 257);

this->label27->Name = L"label27";

this->label27->Size = System::Drawing::Size(12, 13);

this->label27->TabIndex = 27;

this->label27->Text = L"s";

// comboBox2

this->comboBox2->DisplayMember = L"Cambios pulsantes";

this->comboBox2->FormattingEnabled = true;

this->comboBox2->Items->AddRange(gcnew cli::array< System::Object^
>(2) {L"Cambios pulsantes", L"Cambios progresivos"});

this->comboBox2->Location = System::Drawing::Point(204, 255);

this->comboBox2->Name = L"comboBox2";

this->comboBox2->Size = System::Drawing::Size(121, 21);

this->comboBox2->TabIndex = 28;

// Form1

this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->BackColor = System::Drawing::SystemColors::Control;

this->ClientSize = System::Drawing::Size(861, 350);

this->Controls->Add(this->label25);   //Componentes que no van en
ningun groupBox

this->Controls->Add(this->label24);

this->Controls->Add(this->button3);

this->Controls->Add(this->groupBox3);

this->Controls->Add(this->pictureBox2);

this->Controls->Add(this->groupBox1);

this->Controls->Add(this->groupBox2);

this->Cursor = System::Windows::Forms::Cursors::Arrow;

this->Icon = (cli::safe_cast<System::Drawing::Icon^  >(resources-
>GetObject(L"$this.Icon")));

this->Name = L"Form1";
```

```
        this->Text = L"Generador de funciones";

        this->groupBox1->ResumeLayout(false);

        this->groupBox2->ResumeLayout(false);

        this->groupBox2->PerformLayout();

        //Finalizamos las inicializaciones

        (cli::safe_cast<System::ComponentModel::ISupportInitialize^  >(this->
        numericUpDown8))->EndInit();

        cli::safe_cast<System::ComponentModel::ISupportInitialize^  >(this->
        numericUpDown7))->EndInit();

        (cli::safe_cast<System::ComponentModel::ISupportInitialize^  >(this->
        numericUpDown5))->EndInit();

        (cli::safe_cast<System::ComponentModel::ISupportInitialize^  >(this->
        numericUpDown6))->EndInit();

        (cli::safe_cast<System::ComponentModel::ISupportInitialize^  >(this->
        numericUpDown4))->EndInit();

        (cli::safe_cast<System::ComponentModel::ISupportInitialize^  >(this->
        numericUpDown3))->EndInit();

        (cli::safe_cast<System::ComponentModel::ISupportInitialize^  >(this->
        numericUpDown2))->EndInit();

        (cli::safe_cast<System::ComponentModel::ISupportInitialize^  >(this->
        numericUpDown1))->EndInit();

        this->groupBox3->ResumeLayout(false);

        this->groupBox3->PerformLayout();

        (cli::safe_cast<System::ComponentModel::ISupportInitialize^  >(this->
        pictureBox1))->EndInit();

        (cli::safe_cast<System::ComponentModel::ISupportInitialize^  >(this->
        pictureBox2))->EndInit();

        this->ResumeLayout(false);

        this->PerformLayout();

}

#pragma endregion

};

}
```

Figura 15.36: Función InitializeComponent() parte de parametrización de componentes y finalización del archivo form1.h