**Laia Altès Figueras**

# PRÀCTIQUES AMB EL MICROCONTROLADOR ATMEGA328

**TREBALL DE FI DE GRAU**

**dirigit per Nicolau Cañellas Alberich**

**Grau d'Enginyeria Electrònica i Automàtica**

UNIVERSITAT ROVIRA I VIRGILI

**Tarragona**

**2021**

# Índex

ÍNDEX DE FIGURES ..... 3
ÍNDEX DE TAULES ..... 4
1. Introducció ..... 5
1.1. Objectius del Treball ..... 5
1.2. Abast ..... 5
1.3. Antecedents ..... 5
2. Introducció a l'Electrònica Digital i als Microcontroladors ..... 7
2.1. Electrònica Digital ..... 7
2.2. Microprocessadors ..... 8
2.2.1. Unitat de Control (UC) ..... 8
2.2.2. Unitat de Procés ..... 9
2.3. Microcontroladors ..... 10
2.3.1. Memòria ..... 11
2.3.2. Unitat d'Entrada i Sortida ..... 12
2.3.3. Busos del Sistema ..... 12
2.3.4. Tipus d'Arquitectures ..... 14
*2.4.* Sistemes *Embedded* ..... 16
2.4.1. Perspectiva general ..... 16
2.4.2. Estructura dels Sistemes Embedded ..... 18
2.4.3. Components Hardware ..... 18
2.4.4. Components Software ..... 19
2.4.5. ARM ..... 19
2.5. Màquina Senzilla ..... 20
3. ATmega328 ..... 22
3.1. Introducció ..... 22
3.2. Microchip ..... 22
3.3. Família AVR ..... 24
3.4. El Processador ..... 26
3.4.1. CPU ..... 29
3.4.2. Registres de la CPU ..... 30
3.5. La Memòria ..... 31
3.5.1. Memòria de Programa ..... 32
3.5.2. Memòria de Dades SRAM ..... 34
3.5.3. Memòria de Dades EEPROM ..... 35
3.5.4. Modes d'Adreçament ..... 36

3.6. Joc d'Instruccions de la Família AVR ..... 37
3.7. Unitat d'Entrada i Sortida ..... 41
3.8. Gestió de les Interrupcions ..... 44
3.9. ATmega328P Xplained Mini ..... 45
3.10. Shield Board ..... 46
3.11. LCD ..... 49
4. Entorn de Programació ..... 50
4.1. Microchip Studio ..... 50
4.2. Compilador ..... 51
4.3. Programació en Assemblador i C ..... 53
5. Pràctiques ..... 54
5.1. Introducció ..... 54
5.2. Software utilitzat ..... 55
5.3. Pràctica 0: Introducció a les Pràctiques ..... 56
5.4. Pràctica 1: Funció Delay ..... 57
5.5. Pràctica 2: Interfícies d'Usuari de la Xplained mini ..... 58
5.6. Pràctica 3: Ports d'Entrada i Sortida ..... 59
5.7. Pràctica 4: Introducció a la Programació i Depuració en C ..... 60
5.8. Pràctica 5: Control LCD ..... 61
6. Conclusions ..... 62
7. Bibliografia ..... 63
8. Annexos ..... 65
8.1. Esquemàtic Mòdul Extern ..... 66
8.2. Codi Desenvolupat ..... 67
8.2.1. Pràctica 1 ..... 67
8.2.2. Pràctica 2 ..... 68
8.2.3. Pràctica 3 ..... 72
8.2.4. Pràctica 4 ..... 76
8.2.5. Pràctica 5 ..... 81

# ÍNDEX DE FIGURES

**Figura 1.** Símbol esquemàtic d'una Unitat Aritmètico-Lògica [1]. ..... 9
**Figura 2**. Esquema d'un Microprocessador i d'un Microcontrolador [2]. ..... 11
**Figura 3.** Bus del Sistema [3]. ..... 13
**Figura 4.** Arquitectura General d'un Microcontrolador [4]. ..... 13
**Figura 5**. Diagrama d'Execució Pipeline. ..... 16
**Figura 6.** Esquema general d'un sistema encastat [4]. ..... 18
**Figura 7**. Cera i Execució d'Instruccions paral·leles en microcontroladors AVR [5]. ..... 24
**Figura 8.** Diagrama de blocs del processador ATmega328P [6]. ..... 28
**Figura 9.** Diagrama de Blocs de l'Arquitectura AVR [7]. ..... 29
**Figura 10.** Estructura dels Registres de Propòsit General [6]. ..... 30
**Figura 11**. Detall dels Bits del Registre d'Estat [6]. ..... 31
**Figura 12.** Estructura de la Memòria d'un Microcontrolador. ..... 32
**Figura 13.** Mapa de la Memòria de Programa de l'ATmega328P [6]. ..... 33
**Figura 14.** Mapa de la Memòria de Dades de l'ATmega328P [6]. ..... 34
**Figura 15.** Exemple del Format de les Instruccions en Assemblador. ..... 38
**Figura 16.** Instruccions d'Operacions aritmètiques o lògiques [8]. ..... 39
**Figura 17.** Instruccions de Canvi de Flux de Programa [8]. ..... 40
**Figura 18.** Instruccions per a la Gestió del Stack Pointer [6]. ..... 41
**Figura 19.** Pinout de l'ATmega328P. ..... 42
**Figura 20.** Esquemàtic d'un Pin d'Entrada i Sortida [6]. ..... 43
**Figura 21.** Components Principals del Xplained mini. ..... 46
**Figura 22.** Vista Top i Bottom de la Shield Board utilitzada [9]. ..... 47
**Figura 23.** Simulació Gràfica del Mòdul Extern [10]. ..... 48
**Figura 24.** Mòdul Extern configurat. ..... 48
**Figura 25.** Mòdul Extern amb la Pantalla LCD connectada. ..... 49
**Figura 26.** Diagrama de Flux bàsic del Compilador. ..... 52
**Figura 27.** Seqüència del Procés de compilació [11]. ..... 52

# ÍNDEX DE TAULES

**Taula 1.** Famílies de Microcontroladors de Microchip. ..... 23
**Taula 2.** Funcionalitat dels Bits del Registre d'Estat. ..... 31
**Taula 3.** Instruccions de Càrrega i Descàrrega de Dades en l'Espai de Memòria. ..... 37
**Taula 4.** Nomenclatura del Joc d'Instruccions. ..... 37

# 1. Introducció

## 1.1. Objectius del Treball

L'objectiu principal d'aquest treball de fi de grau és l'ampliació de les pràctiques que conformen el ventall d'opcions de què disposa l'assignatura de Microcontroladors i Sistemes *Embedded* del grau d'Enginyeria de Sistemes i Serveis de Telecomunicacions. Amb la finalitat que el professorat disposi d'un repertori més ampli per a la tria i selecció dels continguts que donen eix i forma a l'assignatura, en funció dels coneixements i habilitats de cada grup.

En conseqüència, l'objectiu principal de les pràctiques és que l'alumne assoleixi progressivament els conceptes clau del món dels sistemes *embedded* i, en concret, obtingui les competències necessàries en la programació de microcontroladors.

Així doncs, per a realitzar aquestes pràctiques s'utilitza el kit d'avaluació ATmega328P Xplained Mini, el qual porta integrat el microcontrolador de 8 bits ATmega328P. A més, a fi de treballar altres conceptes tècnics i funcions del microcontrolador, també s'utilitza una *shield bord*, la qual permet la utilització de perifèrics.

## 1.2. Abast

L'abast del projecte inclou la realització del nombre de pràctiques pertinents per tal que l'alumne adquireixi progressivament els coneixements impartits en les classes teòriques de l'assignatura. Particularment, han de permetre l'assoliment de les següents competències i resultats d'aprenentatge que s'indiquen a la guia docent:

- Utilització de microprocessadors i circuits integrats.
- Capacitat per analitzar, codificar, processar i transmetre informació multimèdia emprant tècniques de processat analògic i digital de senyal.
- Conèixer l'arquitectura, funcionament i programació de microcontroladors.

Si bé, per tal d'assolir els coneixements necessaris per a poder dissenyar correctament les pràctiques, primer es presenta una introducció teòrica sobre l'electrònica digital i els microcontroladors, sempre centrada en el món dels sistemes encastats.

Seguidament, s'aprofundeix en el microcontrolador emprat en el present projecte. Estudiant la seva arquitectura, joc d'instruccions i el kit d'avaluació utilitzat.

Finalment, amb l'objectiu de disposar del millor entorn de treball, se solda una placa de prototip amb tots els elements electrònics i interfícies necessàries.

## 1.3.Antecedents

Fins al curs 2019/20, als laboratoris de l'assignatura de Microcontroladors i Sistemes Embedded, s'ha utilitzat la placa de desenvolupament PICDEM2 Plus la qual incorpora el microcontrolador PIC16F877 de 8 bits.

Ara bé, a partir del curs 2020/21, aquestes pràctiques es realitzen amb el microcontrolador ATmega328P, el qual pertany a la família AVR. Aquest canvi ha estat motivat per les següents causes.

En primer lloc, les plaques d'Arduino Uno, les quals es fan servir en assignatures posteriors, utilitzen el mateix microcontrolador. D'aquesta manera, els estudiants ja estaran familiaritzats amb l'arquitectura AVR i el seu joc d'instruccions.

En segon lloc, s'utilitza el potent entorn de desenvolupament Atmel Studio, el qual ha sigut recentment redenominat Microchip Studio. Tenint en compte que aquest software és comú tant pels dispositius de la família AVR, com per les plaques d'Arduino i és àmpliament utilitzat en els sistemes *embedded*, són raons de pes per a justificar aquesta evolució.

En tercer lloc, la utilització d'aquest tipus de microcontrolador facilita la transició a l'arquitectura ARM, atès que les dues estan basades en l'arquitectura RISC (*Reduced Instrucction Set Computer*).

Fet molt important, ja que cal ressaltar que ARM és el principal proveïdor de microprocessadors a la indústria. Vist que ofereix una àmplia gamma de productes amb els quals poder complir els requisits de funcionament, potència i cost econòmic exigits per gairebé totes les aplicacions actuals. Segons la web oficial[1], més del 70% de la població mundial està utilitzant la tecnologia d'ARM, essent el conjunt d'instruccions més àmpliament emprat.

Un clar exemple és que algunes de les versions d'Arduino utilitzen microcontroladors ARM. Obtenint com a resultat la unió d'una de les arquitectures principals amb la plataforma de desenvolupament de hardware i software lliure més estesa mundialment.

Finalment, gràcies a les seves característiques i que cada vegada ofereixen més rendiment i potència, cal destacar l'ús massiu de processadors d'arquitectura AVR en qualsevol àmbit. Particularment, aquests han esdevingut la peça clau en l'evolució i creixement de la internet de les coses i els sistemes *embedded*.

En conclusió, a raó de la creixent demanda del mercat, és imprescindible que els alumnes tinguin els coneixements necessaris per a poder treballar de forma òptima amb aquest tipus de processadors, ja que es preveu que liderin els avenços en els camps tan prometedors anteriorment citats.

De manera paral·lela, convé ressaltar que el cost del kit de desenvolupament utilitzat és molt més baix que el dispositiu emprat anteriorment. Consegüentment, tots els estudiants podran disposar d'un kit.

D'altra banda, pel que fa a l'evolució dels coneixements dels estudiants d'aquest grau, la primera presa de contacte amb els microcontroladors té lloc durant l'assignatura d'Electrònica Digital, en la qual s'explica la màquina senzilla. Seguidament, cursaran l'assignatura en la qual se centra aquest treball de fi de grau, en la qual s'aprofundeix més en aquest tema.

[1] https://www.arm.com/company/news/2019/03/embedded-world-2019 Consultat: 09/04/2019

# 2. Introducció a l'Electrònica Digital i als Microcontroladors

## 2.1. Electrònica Digital

L'ampli camp de l'electrònica es pot dividir, de manera general, en dues grans branques: electrònica analògica i electrònica digital. La diferència més significativa entre elles és el tipus de variables que utilitzen per a representar la informació amb la qual treballen.

Així doncs, els sistemes analògics utilitzen valors elèctrics que varien de forma contínua i poden prendre infinits valors. En canvi, els sistemes digitals treballen amb variables digitals, les quals prenen valors discrets, és a dir, un nombre finit de valors separats entre ells per uns increments fixos. Més concretament, en la majoria de sistemes digitals, s'utilitzen variables digitals binàries, les quals només poden prendre dos valors, comunament anomenats: 0/1 o cert/fals.

En aquest sentit, els circuits digitals presenten uns avantatges respecte als analògics:

- Més immunitat al soroll, ja que la informació emmagatzemada és menys sensible a les alteracions produïdes per fluctuacions no desitjades.
- Són més fàcils de dissenyar, ja que el nombre d'operacions bàsiques és reduït.
- Poden emmagatzemar informació més fàcilment.
- Aquesta informació es pot processar a una velocitat molt elevada.
- Tenen una gran capacitat d'integració.

Ara bé, el principal inconvenient que presenten és que la majoria de vegades han de treballar amb magnituds analògiques, tant les d'entrada com les de sortida. Atès que els sistemes del món real són principalment analògics. Com a conseqüència, s'han d'utilitzar convertidors d'analògic a digital i de digital a analògic, fet que pot comportar errors de quantificació.

No obstant això, l'electrònica digital ha tingut un gran impacte en les darreres dècades, molts sistemes tradicionalment analògics han esdevingut digitals. Actualment, està en constant evolució i cada cop està més integrada en el dia a dia, ja que ofereix una infinitat de possibilitats en camps molt diversos.

Cal destacar, que aquest projecte se centra en l'electrònica digital, ja que el dispositiu utilitzat per a realitzar les pràctiques, un microcontrolador, utilitza la lògica digital pel processat i tractament de dades.

En concret, tota l'electrònica digital es basa en la lògica binària, és a dir, en la utilització de circuits electrònics formats per portes lògiques que implementen les funcions lògiques booleanes.

Tanmateix, l'aparició dels circuits integrats va marcar el desenvolupament de l'electrònica digital. Aquests components, també anomenats xips o microxips, estan formats per una base de petites dimensions de material semiconductor, normalment silici, sobre la qual s'interconnecten els components electrònics, generalment condensadors, díodes, resistències i transistors. A més, per tal de protegir-los, es presenten dins d'un encapsulat de plàstic o ceràmica. El qual també permet, mitjançant uns conductors metàl·lics, connectar el circuit integrat amb el circuit imprès.

Des de la seva aparició, els circuits integrats no han parat d'evolucionar i millorar les seves característiques. Actualment, es podria dir que tots els dispositius electrònics utilitzen circuits integrats gràcies al seu baix cost i elevat rendiment.

Més concretament, la capacitat d'integració de grans quantitats de petits transistors en les reduïdes dimensions dels circuits integrats ha permès un gran avanç en el món de l'electrònica.

## 2.2. Microprocessadors

Cal destacar, que els circuits integrats més avançats i complexos es troben en els microprocessadors.

Així doncs, en primer lloc, cal definir què és un microprocessador. També anomenat CPU (Unitat Central de Processament) és el circuit integrat central d'un sistema informàtic. És l'encarregat d'executar els programes, en altres paraules, rep dades binàries d'entrada, les processa segons les instruccions emmagatzemades a la memòria, és a dir, duu a terme les operacions lògiques i aritmètiques necessàries, i proporciona els resultats obtinguts com a sortida.

En concret, les instruccions han d'estar programades en llenguatge de baix nivell i els números i símbols utilitzats han de pertànyer al sistema binari, ja que l'arquitectura del microprocessador està basada en aquest sistema.

D'altra banda, la integració de tota una CPU en un sol circuit integrat, va permetre reduir considerablement el cost de la potència de processament i va comportar un augment de la fiabilitat, ja que menys connexions elèctriques són susceptibles a fallar. A més, com que es fabriquen en grans quantitats mitjançant processos altament automatitzats, el preu final unitari no és elevat.

Tot seguit, es detallen els components essencials d'un microprocessador.

### *2.2.1. Unitat de Control (UC)*

La unitat de control dirigeix el funcionament del processador. La seva funció és interpretar i executar les instruccions emmagatzemades a la memòria. Per a fer-ho, primer ha d'anar a buscar les instruccions a la memòria principal, descodificar-les i executar-les. Finalment, ha de generar els senyals de control necessaris per activar o desactivar els diversos components del microprocessador en funció de la instrucció que s'està executant per tal de generar la resposta adequada.

Existeixen dos tipus d'unitats de control: les cablejades i les microprogramades. Les primeres s'implementen mitjançant l'ús d'unitats lògiques combinacionals, amb un nombre finit de portes que generen resultats específics. El seu disseny utilitza una arquitectura fixa, és a dir, requereix canvis en el cablejat si el conjunt d'instruccions es modifica. Per aquest motiu, aquest tipus d'unitats de control són preferibles en dissenys RISC, ja que utilitzen un conjunt d'instruccions més senzill i és més fàcil d'implementar.

Així doncs, aquestes UC presenten l'avantatge que poden funcionar a grans velocitats, generalment més ràpid que les de l'altre tipus. No obstant això, presenten poca flexibilitat i la complexitat del joc d'instruccions que poden implementar és limitada. És per això, que amb l'evolució dels microprocessadors, la tecnologia cablejada ha esdevingut menys popular.

Les segones estan organitzades en una seqüència de microinstruccions emmagatzemades en una memòria de control especial. A més, normalment, el seu algorisme s'especifica

mitjançant un diagrama de flux. El principal avantatge d'aquest tipus d'UC és la senzillesa de la seva estructura i que es pot modificar fàcilment.

### *2.2.2. Unitat de Procés*

La unitat de procés és l'encarregada d'executar els programes emmagatzemats a la memòria i realitzar les operacions sobre les dades. A fi de poder complir amb les seves funcions, està integrada pels següents components hardware:

*a) Unitat Aritmètico-Lògica (ALU)*

És un circuit digital combinacional, és a dir, no té memòria, les sortides únicament depenen de les entrades actuals. Realitza operacions aritmètiques, com per exemple sumes o restes, i operacions lògiques. Tant a escala de bit, com ara AND o OR, com de registre. Les operacions de desplaçament, generalment conegudes com a *shift operations*, en són un bon exemple.

A continuació, es mostra la representació gràfica més habitual d'aquest component, mitjançant la qual es pot entendre més fàcilment el seu funcionament.

Tal com es pot observar a la Figura 1, les entrades d'una ALU són els operands A i B, és a dir, són les dades amb les quals ha de realitzar l'operació indicada a través del codi que conté l'entrada F.

D'altra banda, pel que fa a les sortides, la variable R conté el resultat d'aquesta operació i la D és una sortida d'estat, la qual conté informació addicional sobre el resultat de l'operació actual. Normalment, les ALU de propòsit general tenen el senyal d'estat de zero, el qual indica que tots els bits de la sortida R són zeros lògics.

Convé destacar que l'UC governa aquesta unitat, proporcionant les variables d'entrada i definint la destinació de les sortides.

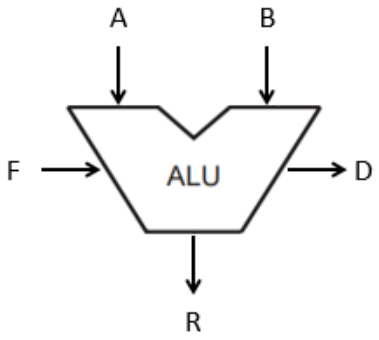



***Figura 1.*** *Símbol esquemàtic d'una Unitat Aritmètico-Lògica [1].*

Més concretament, al final de cada operació de la unitat aritmètico-lògica, els senyals de sortida d'estat s'emmagatzemen en registres externs, a fi d'estar disponibles per a futures operacions. Els diferents registres de bits que emmagatzemen les sortides d'estat sovint es tracten com un únic registre de diversos bits anomenat registre d'estat.

*b) Registres:*

Els registres de la CPU proporcionen memòria d'alta velocitat, essent la manera més ràpida que té el sistema d'emmagatzemar dades. Ara bé, ofereixen una capacitat molt baixa i de tipus volàtil. Permeten guardar dades transitòriament i accedir a valors molt utilitzats, fet que beneficia la realització d'operacions matemàtiques.

En particular, es poden dividir entre registres generals i específics. Els primers s'utilitzen per a emmagatzemar resultats intermedis.

En canvi, els segons són registres amb una funcionalitat específica. Els utilitzats més freqüentment pels programadors són: el comptador de programa (PC), el qual conté l'adreça de memòria de la paraula on es troba la següent instrucció a executar i el comunament anomenat *Stack Pointer* (SP), el qual permet gestionar la pila del sistema.

La característica que comparteixen aquests dos registres és que, normalment, el programador ni els manipula ni hi accedeix, ja que la CPU és l'encarregada d'actualitzar-los a mesura que el programa va corrent.

Al contrari que el registre d'estat, generalment conegut com a SR o SREG, de la sigla en anglès *Status Register*, que és utilitzat àmpliament per a prendre decisions. Un altre tret diferenciador és que està format per, normalment, 8 bits, anomenats *flags*, cadascun d'ells indica l'esdeveniment d'una condició particular. Com ara, si una operació realitzada per l'ALU té com a resultat zero o si el resultat és negatiu, entre altres.

Tanmateix, perquè un microprocessador pugui funcionar, ha d'interactuar amb dos elements externs, la memòria i la unitat d'entrada i sortida. Per a fer-ho, utilitza el bus del sistema. Aquests tres elements s'expliquen en detall al següent apartat.

### 2.3. Microcontroladors

Com es pot observar a la Figura 2, la principal diferència entre un microcontrolador i un microprocessador és que el primer inclou en un mateix circuit integrat la CPU, la memòria i els perifèrics d'entrada i sortida, els quals ja vénen fixats de fàbrica. En canvi, per un microprocessador aquests són elements externs, és a dir, el dissenyador els ha d'integrar, en funció de les característiques de l'aplicació, perquè pugui funcionar correctament. En altres paraules, un microprocessador no es pot utilitzar de forma aïllada, sempre necessita perifèrics, fet que farà augmentar el cost del sistema final.

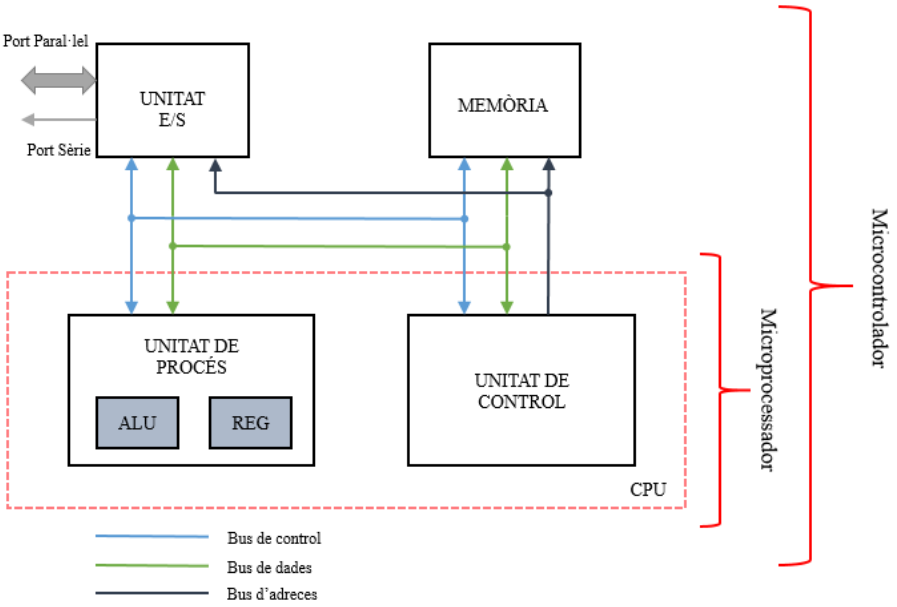



***Figura 2**. Esquema d'un Microprocessador i d'un Microcontrolador [2].*

A més, a diferència dels microprocessadors, que són utilitzats en ordinadors personals i sistemes de propòsit general, els microcontroladors s'utilitzen en tasques específiques, és a dir, la relació entre les entrades i les sortides està definida prèviament. És per això, que necessiten menys recursos, com memòria i ports d'E/S, i poden ser integrats en un sol xip. En conseqüència, la mida, el consum d'energia i el cost es redueixen.

Una altra propietat clau és que la majoria de microcontroladors disposen de modes de repòs en diferents nivells, és a dir, poden suspendre diverses funcions o blocs, per separat o a la vegada. Fins que succeeix un esdeveniment o una interrupció, els quals el faran despertar i el faran tornar al mode d'operació normal. Aquesta estratègia d'estalvi d'energia els fa idonis per a aplicacions que funcionen amb bateria.

Actualment, l'oferta de microcontroladors és molt variada, amb una àmplia gamma de funcions disponibles en diferents versions i, majoritàriament, amb un preu molt assequible.

Gràcies al gran ventall de possibilitats i a les característiques anteriorment esmentades, es poden trobar en múltiples aplicacions i de cada vegada s'utilitzen per a controlar més dispositius i processos. Convé destacar que són àmpliament utilitzats en sistemes *embedded* i en el prometedor camp de la Internet de les coses.

D'altra banda, els tres elements que integren un microcontrolador que no s'havien descrit prèviament són:

### *2.3.1. Memòria*

És el bloc del computador encarregat d'emmagatzemar la informació necessària. En funció de la naturalesa d'aquesta, s'utilitza un tipus de memòria o un altre. Concretament, a la memòria RAM, s'emmagatzemen les dades del programa i d'usuari, és a dir, les dades generades durant l'execució del programa. En canvi, les dades pròpies del microcontrolador i les instruccions que formen el programa s'emmagatzemen a la memòria ROM.

D'una banda, la memòria RAM, anomenada així per l'acrònim en angles de memòria d'accés aleatori. És una memòria volàtil, si s'interromp l'alimentació, la informació emmagatzemada es perd. No obstant això, presenta l'avantatge que l'accés a les dades és ràpid i és més econòmica que la del següent tipus.

D'altra banda, la memòria ROM, només permet la lectura i és no volàtil. Fet que la fa ideal per emmagatzemar les instruccions del programa, ja que és imprescindible que no s'esborrin encara que es talli l'alimentació. En principi, aquestes dades no s'han de modificar durant l'execució normal del programa.

### *2.3.2. Unitat d'Entrada i Sortida*

És el bloc encarregat d'establir les connexions amb l'exterior.

Així doncs, els microcontroladors tenen diversos pins d'entrada/sortida d'ús general, comunament anomenats GPIO, del seu acrònim en anglès de *General Purpose Input/Output pins*. S'anomenen d'ús general perquè mitjançant el software es poden configurar com pins d'entrada, per a llegir sensors o senyals externs, o de sortida, per a controlar dispositius externs, des de LED a components de més potència com ara motors. És a dir, aquests elements permeten connectar el µcontrolador amb els perifèrics disponibles.

Consegüentment, sovint, aquesta lectura implica haver de tractar amb senyals analògics. Ara bé, el microcontrolador funciona amb lògica digital, és per això, que s'han d'utilitzar convertidors analògics – digitals freqüentment.

### *2.3.3. Busos del Sistema*

La funcionalitat dels busos del sistema és interconnectar els tres components bàsics d'un microcomputador, la CPU, la memòria i la interfície d'entrada i sortida. Aquest sistema de comunicació està format per components hardware, com ara cables i pistes del circuit imprès, i software, inclosos els protocols de comunicació.

Actualment, es poden diferenciar dos tipus de bus en funció del mètode que utilitzen per a enviar la informació. Per una banda, hi ha el bus sèrie, el qual només és capaç de transferir les dades bit a bit. I per l'altra, hi ha el bus paral·lel, el qual permet transferir diversos bits simultàniament.

Pel cas dels microprocessadors i microcontroladors, es poden distingir tres categories de busos en funció de la seva funció:

*a) Bus d'adreces:*

A través d'ell, la CPU envia les adreces de memòria físiques on es troben les instruccions que s'han de llegir o les ubicacions on ha de llegir o escriure dades.

Cal destacar, que aquest bus és de tipus unidireccional, ja que la CPU és l'única responsable de determinar l'adreça amb la qual ha de treballar.

L'amplada d'aquest bus determina la quantitat de memòria que pot adreçar el sistema.

*b) Bus de dades:*

Permet l'intercanvi de dades i d'instruccions entre la CPU i els altres blocs. Cal destacar, que aquest bus és bidireccional, és a dir, permet la lectura i l'escriptura.

*c) Bus de control:*

Governa l'ús i l'accés al bus de dades i al d'adreces. A través d'ell, es transmeten senyals de control, com ara de lectura, escriptura o rellotge. Més concretament, transmet comandes de la CPU i retorna senyals d'estat dels altres blocs.

Així, tal com es pot observar en la Figura 3, aquest bus és de tipus bidireccional. A més, convé destacar que en aquesta figura, es representa l'arquitectura von Neumann, la qual es detalla a continuació.

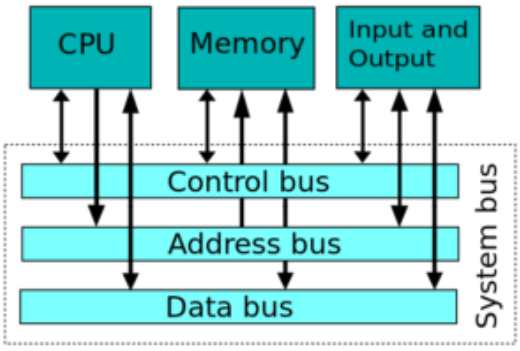



***Figura 3.*** *Bus del Sistema [3].*

Finalment, per tal que el microcontrolador pugui operar correctament, cal proporcionar-li l'alimentació adequada i un element de sincronització.

En la següent figura, es pot observar la integració dels components mínims d'un microcontrolador i com interactuen entre ells.

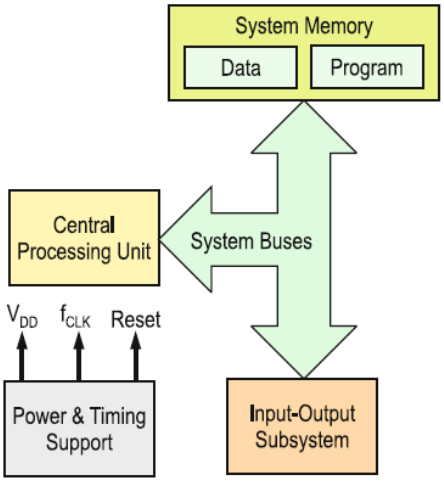



***Figura 4.*** *Arquitectura General d'un Microcontrolador [4].*

### *2.3.4. Tipus d'Arquitectures*

A fi de proveir de funcionalitat al conjunt d'elements hardware anteriorment esmentats, tot microcontrolador ha d'integrar la seva part software, atès que és l'encarregada d'establir la seqüència en què els elements hardware actuen. En efecte, aquests dos elements estan estretament vinculats per a proporcionar el correcte funcionament d'un microcontrolador.

Tal com s'ha detallat anteriorment, tot controlador està format per tres elements bàsics, la unitat central de procés, la memòria i la unitat d'entrada i sortida. A més, és necessari un sistema d'interconnexió que permeti el traspàs d'informació entre els tres components elementals, els busos.

Així doncs, l'estructura, l'organització i la manera com interactuen les diverses unitats es descriu en l'arquitectura del controlador. És a dir, en ella s'especifiquen els requisits funcionals i el model de treball intern. Com ara, el mapa de memòria i d'entrades i sortides, el joc d'instruccions, els modes de direccionament, els sistemes d'interconnexió i control, entre altres.

En primer lloc, cal fer referència al fet que els microcontroladors es poden agrupar en dues arquitectures principals en funció del joc d'instruccions que suporten. Per un costat, l'arquitectura CISC, de la sigla en anglès de *Complex Instruction Set Computer*, la qual ofereix un extens conjunt d'instruccions complexes, executades en diversos cicles i amb accés a memòria. Permet generar codis curts i senzills, donat que cada instrucció engloba un conjunt de microinstruccions.

Per l'altre, la RISC, de la sigla en anglès de *Reduced Instruction Set Computer*. La qual es caracteritza per estar formada per un conjunt més reduït d'instruccions simples, la majoria del mateix format, amb constants accessos a memòria a través de registres. Si bé aquest disseny simplifica l'estructura hardware, en resulten codis més llargs, ja que per a executar una mateixa operació, es necessiten més instruccions. Però, tal com es detalla a continuació, s'assoleix una execució més ràpida i eficient gràcies a la segmentació.

Així doncs, els principals beneficis que s'obtenen de la utilització de l'estructura RISC és que s'aconsegueixen unitats de control més simples que permeten una segmentació més senzilla.

De manera paral·lela, els microcontroladors es poden classificar en dos tipus més d'arquitectures. D'una banda, hi ha l'anomenada Von Neumann, amb la principal característica que només contempla un únic espai de memòria de lectura i escriptura, on s'emmagatzemen les dades i les instruccions del programa.

L'objectiu principal del seu creador era construir un sistema flexible que permetés resoldre diferents problemes utilitzant el mateix joc d'instruccions. L'estructura d'aquesta es pot observar en les dues figures anteriors.

De l'altra, l'arquitectura Harvard, la qual també es va crear durant la dècada dels quaranta. Ara bé, existeix una gran diferència entre elles, en aquest model, hi ha dues memòries diferents, una per a les dades i l'altra pel programa, connectades a la CPU mitjançant dos busos diferents. Aquest fet facilita el procés de segmentació, mentre una etapa accedeix a la memòria de programa, l'altra pot accedir a la de dades, ja que s'utilitza un hardware diferent.

Així doncs, en funció de les necessitats de l'aplicació, se selecciona un tipus d'arquitectura o altre. Essent un dels factors clau per a aconseguir la màxima eficiència del procés en qüestió.

Arribats en aquest punt, és necessari explicar en què consisteix aquest mètode. La segmentació d'instruccions o *pipeline* és una tècnica mitjançant la qual s'aconsegueix augmentar el rendiment del microcontrolador.

Primerament, cal recordar com un microcontrolador executa una instrucció. La unitat de control governa la CPU com una màquina d'estats finita amb el següent cicle, generalment conegut com a *instruction cycle* o *CPU cycle*, format dels següents estats:

- ***Fetch* o cerca:** Durant aquest estat, es va a buscar a la memòria la instrucció a executar i es guarda al registre d'instrucció (IR).

- **Descodificació:** Seguidament, cal descodificar el codi de la instrucció per a saber quines accions cal dur a terme.

- **Execució:** En aquest estat, a fi de dur a terme les accions especificades per la instrucció, la UC envia els senyals de control necessaris a les unitats funcionals corresponents. Com ara, llegir valors dels registres, efectuar operacions a l'ALU i, finalment, escriure el resultat de l'operació en el registre adequat o enviar el senyal generat a un dispositiu de sortida o actuador.
  Al final de l'etapa d'execució, el comptador de programa s'incrementa per a apuntar a l'adreça de la següent instrucció de memòria.

Ara bé, en molts casos, quan es requereix accedir més cops a la memòria del sistema, són necessàries etapes intermèdies. A més, en tractar-se d'una màquina d'estats finita, es necessita un senyal de Reset per a inicialitzar el cicle per primera vegada, és a dir, per anar a buscar la primera instrucció.

En aquest cas, es considera un processador segmentat en 5 etapes. Cada una de les quals, utilitza uns determinats recursos de la CPU, és a dir, es pot considerar independent de les altres.

- Lectura de la instrucció, generalment coneguda com a *instruction fetch* (IF).
- Descodificació de la instrucció i lectura dels operands (ID).
- Càlcul o execució de la instrucció (EX).
- Accés a la memòria de dades (MEM).
- Escriptura del resultat (WRT).

A més, com és evident, en qualsevol aplicació hi ha més d'una instrucció esperant a ser executada.

Gràcies a aquestes característiques, es poden superposar diverses instruccions durant l'execució del programa. Atès que en cada cicle de rellotge, es duu a terme una etapa diferent de cada instrucció. Així, s'executen simultàniament diferents etapes de diferents instruccions.

En realitat, en alguns casos, les etapes o les instruccions sí que són dependents entre elles. Per exemple, quan el resultat d'una instrucció és utilitzat com a operand per una altra o en instruccions de tipus salt, condicionals o interrupcions, en les quals es trenca la seqüència d'execució normal definida en el programa. Consegüentment, calen registres per a emmagatzemar els resultats intermedis entre etapes, comunament anomenats bancs de registres.

En definitiva, mitjançant aquesta tècnica, s'aconsegueix augmentar la velocitat d'execució, és a dir, s'incrementa l'eficiència del sistema. Si bé, l'avantatge més conegut que resulta de

la utilització d'aquesta solució d'arquitectura és que sembla que les instruccions s'executen en un sol cicle de rellotge, encara que cada instrucció en necessita cinc.

Tal com es pot observar en el següent diagrama, mentre es descodifica la primera instrucció, es va a buscar a la memòria la segona instrucció, i així simultàniament amb les altres etapes.

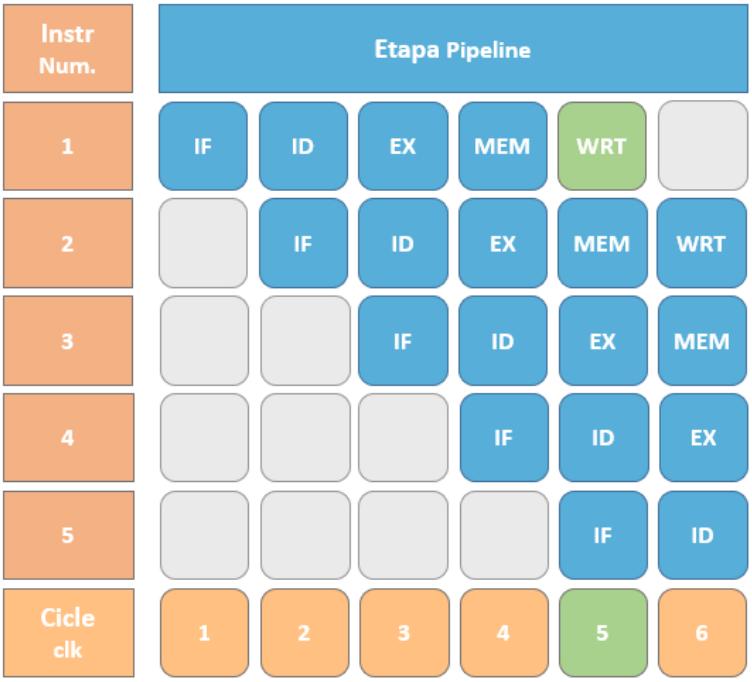



***Figura 5**. Diagrama d'Execució Pipeline.*

Per acabar, convé destacar, que el microcontrolador objecte d'estudi està dissenyat seguint una arquitectura Harvard amb un joc d'instruccions RISC, fet molt important a tenir en compte a l'hora de programar-lo. A més, com és evident, a fi d'obtenir un rendiment més elevat, utilitza el recurs de la segmentació.

## *2.4.* Sistemes *Embedded*

### *2.4.1. Perspectiva general*

Tal com s'ha mencionat anteriorment, la principal aplicació dels microcontroladors és en sistemes *embedded*, és a dir, sistemes encastats.

Així doncs, cal conèixer què és un sistema encastat. En termes generals, un sistema *embedded* té una funcionalitat molt delimitada, ja que el nombre de tasques i les seves especificacions són conegudes, i forma part d'un sistema més ampli.

Gràcies a l'elevada integració dels seus components, els microcontroladors presenten les següents característiques, les quals els fan idonis per a ser utilitzats en aquest tipus d'aplicacions.

- Són més fiables: com que gran part dels components estan ubicats en un mateix xip, es redueixen les interconnexions, és a dir, es disminueix la probabilitat de fallada.
- Permeten un major rendiment: ja que cada un dels elements del sistema ha estat minuciosament escollit per a optimitzar i assolir el millor funcionament.

El tret més significatiu d'aquests microcontroladors és que són programats per a realitzar només una o, com a molt, un nombre reduït de tasques específiques. Fins i tot, el fet d'haver de canviar o modificar la tasca, molts cops implica haver d'alterar tot el sistema.

No obstant això, un determinat sistema encastat és compatible amb una àmplia gamma de processadors, ja que, actualment, el nombre i la varietat de microprocessadors disponibles al mercat és molt elevat. Aquest fet, permet que l'usuari pugui seleccionar el dispositiu que més s'ajusti a les necessitats de cada aplicació en concret i li permet solucionar el problema d'una manera òptima.

Un altre tret a destacar, és que tenen restriccions de temps real. És a dir, el correcte funcionament d'aquest sistema té lloc quan el resultat és l'esperat i, a més, aquest resultat ha estat generat en l'instant precís i dins d'un termini de temps prèviament definit. En altres paraules, aquests sistemes interactuen activament amb el seu entorn i han de donar resposta als esdeveniments externs en el moment adequat i a una velocitat controlada.

D'altra banda, aquests sistemes també presenten limitacions pel que fa al preu i al consum d'energia. Pel que fa a la segona restricció, cal tenir en compte que la majoria d'aquests dispositius han de treballar de manera fiable durant un període llarg de temps alimentant-se de les seves bateries. A més, cal tenir present que les limitacions d'energia tenen un gran impacte en el disseny general del sistema, ja que limiten el nombre de components electrònics actius, com ara els ports d'entrada i sortida, o el llenguatge que s'utilitzarà per a escriure el codi. Pel cas que el consum hagi de ser el menor possible, s'utilitzarà el llenguatge assemblador en lloc de C o C++, ja que aquest permet un millor rendiment.

Més concretament, el consum d'energia està directament relacionat amb la velocitat del rellotge de la CPU. Aquest fet és a causa que la majoria de les CPU utilitzen la tecnologia dels transistors CMOS, els quals es comporten com interruptors perfectes, quan no canvien d'estat, la dissipació d'energia és gairebé zero. Ara bé, quan el circuit commuta entre els dos nivells lògics, sí que es produeix un consum considerable.

Així doncs, com més elevada és la velocitat del rellotge, més commutacions per segon es produeixen. En conseqüència, es dissipa més energia, fet que implica més dissipació de calor, la qual també s'ha de tindre en compte, ja que genera una altra restricció. De fet, pels processadors actuals, la relació entre la velocitat de la CPU i la dissipació de potència és gairebé lineal.

També hi ha altres recursos per a estalviar energia, un dels més utilitzats és dissenyar el software al voltant d'un sistema en el qual el processador estigui la major part del temps en mode repòs i només es desperti quan es produeixi un determinat esdeveniment. És a dir, el sistema funciona per interrupcions.

A més, atès que els sistemes encastats s'utilitzen per a solucionar incomptables problemes, molts d'ells han de fer front a condicions climàtiques adverses. No només temperatures extremes, siguin màximes o mínimes, sinó altres inclemències del clima. Sovint, aquestes condicions impliquen restriccions més severes pel que fa al consum d'energia o a la dissipació de calor.

Finalment, cal tenir present que els errors o les avaries són molt menys tolerables en els sistemes incrustats que en la majoria d'ordinadors, ja que les conseqüències d'una fallada són molt més severes. Vist que moltes de les aplicacions són crítiques, com per exemple, sistemes de seguretat en automòbils o avions.

Ara bé, això no significa que els sistemes *embedded* no fallin mai, sinó que la majoria incorporen mecanismes per tal que el sistema no perdi el control totalment i es recuperi després d'un error, com ara el *watchdog timer*.

### *2.4.2. Estructura dels Sistemes Embedded*

Com s'ha comentat anteriorment, l'element clau dels sistemes encastats és el microcontrolador. Per això, fins a la seva consolidació, a la dècada dels setanta[2], com a elements electrònics amb un gran potencial per a revolucionar els sistemes de control, no es va produir l'auge dels sistemes encastats, evidenciant els nombrosos avantatges de la implementació d'aquests.

Encara que en un primer moment la seva arquitectura estava basada en 4 bits, ràpidament van evolucionar a 8 o 16 bits i van dominar el mercat dels sistemes encastats.

Actualment, per a poder donar solució als problemes cada vegada més complexos, aquests s'han de dividir i s'ha de dissenyar un conjunt de sistemes *embedded* que interactuïn entre ells. Encara que la majoria d'aquests sistemes requereixen la mínima interacció humana per tal de dur a terme la seva tasca, aquests sistemes es poden trobar en pràcticament tots els aspectes del dia a dia. Fins i tot, molts cops passen gairebé desapercebuts per l'usuari.

Així doncs, tal com es pot observar en la Figura 6, els sistemes encastats estan formats per dos components estretament relacionats, la part hardware i la software. En conseqüència, els programadors especialitzats en aquests dispositius han de considerar els dos components per a poder obtenir una bona aplicació.

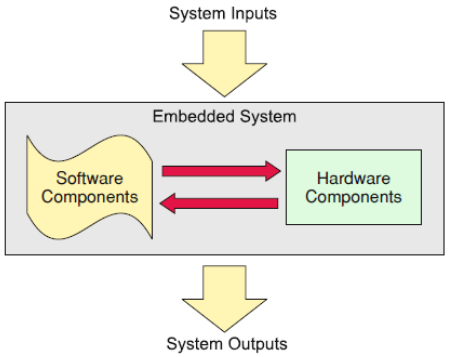



***Figura 6.*** *Esquema general d'un sistema encastat [4].*

### *2.4.3. Components Hardware*

Per una banda, hi ha els components hardware, essent el microcontrolador l'element principal. No obstant això, també en forma part tota l'electrònica necessària per a realitzar satisfactòriament la tasca desitjada. Per aquest motiu, en funció de l'aplicació, tindrà uns components o uns altres per tal de complir les necessitats en qüestió.

Tanmateix, tal com s'ha comentat anteriorment en l'apartat 2.3. Microcontroladors, hi ha tres elements imprescindibles en qualsevol sistema *embedded*: la CPU, la memòria i els ports d'entrada i sortida.

A més, des d'aquest punt de vista, també es gestionen les interaccions amb el món extern. Tant els inputs del sistema, els quals són majoritàriament dades proporcionades pels sensors i els ports d'entrada. Com els outputs, essent les sortides generades pel sistema, generalment en forma d'accions de control sobre els actuadors, com motors o relés. Així com informació per l'usuari o per altres subsistemes o sistemes interconnectats. En alguns casos, l'intercanvi de dades amb l'usuari es produeix mitjançant interfícies, com ara polsadors, LED, pantalles LCD o fins i tot pantalles HMI.

A fi d'intercanviar dades amb l'entorn i d'aportar la funcionalitat requerida al sistema, els elements perifèrics són essencials. En són un bon exemple, els convertidors d'analògic a digital, o a la inversa, les interfícies d'usuari o els ports de comunicació. Com també els dispositius de suport, els quals proporcionen serveis essencials per a treballar. Per exemple, fonts d'alimentació, generadors de la freqüència de rellotge, *timers*, entre altres.

### *2.4.4. Components Software*

I per l'altra, els components de software, és a dir, els programes que donen funcionalitat al hardware i marquen quines accions s'han de realitzar, seguint la seqüència d'instruccions adequada.

Normalment, els programes són emmagatzemats en memòria, és a dir, en un component que pertany al primer grup. Fet que ressalta l'evidència de l'estreta relació entre aquests dos elements. A més, dins d'aquest conjunt es pot fer la següent subdivisió:

- *Tasques del sistema*: En els sistemes encastats, l'aplicació software està dividida en un conjunt de programes més petits anomenats tasques. Cada tasca realitza una funció en concret. Per a fer-ho, ha d'utilitzar els recursos del sistema que li siguin necessaris per a cada situació en concret. Així, les tasques envien sol·licituds de servei, mitjançant registres o interrupcions al *kernel*.

- *Kernel, nucli del sistema:* És el component software que controla, programa i gestiona els recursos del sistema. És a dir, quan li arriba una sol·licitud d'una tasca, li dóna servei en funció de la seva prioritat. A més, també proporciona un entorn adequat i segur per tal que les tasques que ho precisin intercanviïn informació entre elles.

- *Rutines de Servei*: Les tasques són ateses mitjançant trossos de codi que atorguen funcionalitat a un recurs. En funció del tipus d'arquitectura, aquestes poden ser activades a través d'enquestes o rutines de servei a la interrupció, comunament conegudes com a ISR, de la sigla en anglès.

### *2.4.5. ARM*

L'arquitectura ARM, de la sigla en anglès d'*Advanced* RISC *Machine,* va ser desenvolupada i comercialitzada per primera vegada a la dècada dels vuitanta per la companyia ARM Holdings. La qual té la seu a Cambridge i, actualment, és considerada una de les empreses més dominants en la indústria de semiconductors i, especialment, en el mercat dels xips.

Així doncs, en primer lloc, cal conèixer els trets principals d'aquesta arquitectura, gràcies als quals ha esdevingut l'element clau en la majoria dels dispositius d'electrònica mòbil i integrada, convertint-se en el conjunt d'instruccions més àmpliament utilitzat.

La principal característica és que és de tipus RISC, és a dir, utilitza un conjunt d'instruccions simples implementat per hardware directament a la CPU, el qual ofereix la possibilitat d'executar processos paral·lels i reduir els accessos a memòria, permetent executar les instruccions més ràpidament i, per tant, amb un estalvi d'energia.

A més, aquesta arquitectura és més simple i necessita menys transistors que l'arquitectura CISC. Fet que provoca que aquests processadors siguin més econòmics, més ràpids i que dissipin menys calor, de manera que no necessiten un sistema de refrigeració potent. Si bé, cal tenir present que aquest ha d'estar dimensionat segons la freqüència de rellotge del sistema, ja que com més commutacions per segon, més energia es dissipa.

A raó d'aquestes qualitats, els processadors ARM s'han convertit en els dominants en el camp dels dispositius que funcionen amb bateries, és a dir, en l'electrònica mòbil i sistemes *embedded*, on l'eficiència energètica és primordial. Amb tot plegat, s'obtenen microcontroladors amb capacitats de disseny més flexibles i senzills, però amb un elevat rendiment.

No obstant això, el que fa realment especial aquesta empresa, és que no només dissenya i ven eines de programació, sinó que també ven llicències de fabricació. Concretament, és el principal proveïdor mundial de llicències, oferint la seva tecnologia a una extensa xarxa de companyies, les quals comercialitzen des dels dispositius més simples d'internet de les coses fins a solucions molt més complexes i sofisticades.

De fet, ARM no fabrica els xips, més aviat, el seu principal negoci és la venda d'IP, de la sigla en anglès d'*Intellectual Property*. És a dir, desenvolupa, millora i ven l'arquitectura ARM i el set d'instruccions basat en aquesta. En altres paraules, ven el coneixement sobre la interacció i la gestió entre el software i el hardware, com treballa el hardware quan una instrucció en particular és executada.

Així, en funció de l'aplicació, ARM ofereix una àmplia gamma de productes, els quals varien en cost i prestacions. Això sí, a tots els clients els hi proporciona una descripció del maquinari i el conjunt d'eines de desenvolupament de programari, com ara un compilador, un depurador i un kit de desenvolupament.

El comprador utilitza aquestes llicències per a dissenyar el seu producte final. Combinant el nucli ARM amb altres components, com ara memòria o perifèrics, per a crear una CPU completa que satisfaci les seves necessitats.

### 2.5. Màquina Senzilla

La màquina senzilla és un processador amb un propòsit únicament didàctic. El seu principal objectiu és introduir als alumnes els conceptes bàsics sobre l'estructura i l'arquitectura dels computadors. Així, permet realitzar una transició natural entre l'estudi dels circuits digitals i la iniciació en el llenguatge màquina.

En el cas del grau d'Enginyeria de Sistemes i Serveis de Telecomunicacions, s'explica en l'assignatura de Microcontroladors i Sistemes Embedded. Essent aquesta la matèria sobre la qual es realitza la proposta de pràctiques del present projecte.

Així doncs, aquesta potent eina pedagògica no és més que un simple processador basat en l'arquitectura tipus von Neumann. En conseqüència, només té dos blocs, la memòria i la unitat central de procés. Pel que fa al primer bloc, és de tipus RAM amb 128 posicions de 16 bits cada una.

Pel que fa al segon, es caracteritza pel repertori d'instruccions que és capaç d'executar, en aquest cas 4 operacions: suma, moviment, comparació i salt.

En definitiva, mitjançant la màquina senzilla l'estudiant és capaç de comprendre el funcionament d'un microcontrolador, coneixent les característiques dels blocs funcionals i el comportament de les instruccions, tot familiaritzant-se amb el llenguatge assemblador.

# 3. ATmega328

## 3.1. Introducció

En primer lloc, convé destacar que el professorat a càrrec de l'assignatura de Microcontroladors i Sistemes Embedded ha seleccionat el microcontrolador ATmega328P com a dispositiu principal per a realitzar les pràctiques corresponents al laboratori.

Aquesta decisió s'ha pres d'acord amb el propòsit principal de facilitar un aprenentatge didàctic i útil per als estudiants, proporcionant-los les eines idònies i adequades per al futur laboral. Atès que aquest microcontrolador té un ús molt estès, ja que es tracta del nucli de l'Arduino UNO.

A més, en els darrers anys, el nombre d'usuaris de microcontroladors de la família AVR ha augmentat considerablement, fins al punt de competir amb els famosos microcontroladors PIC. Gràcies al fet que ofereixen un elevat rendiment a baix cost i, el més important en aplicacions pedagògiques, un entorn de programació intuïtiu i fàcil d'utilitzar.

Així doncs, a fi de complir l'objectiu principal del present treball de final de grau, s'utilitzen els següents dispositius per a realitzar les pràctiques.

Tot i que abans d'entrar en detall amb les especificacions de cada component, cal tindre una visió general de la gran quantitat de possibilitats de control que hi ha avui en dia en el camp dels sistemes *embedded*.

També convé destacar que al llarg d'aquest punt, es consulten i es mencionen freqüentment els manuals del microcontrolador en qüestió, ja que són un recurs imprescindible a l'hora de treballar amb aquests dispositius. Tenint en compte que contenen tota la informació necessària per a poder-los programar adequadament.

En conseqüència, és primordial que els estudiants estiguin familiaritzats amb aquest tipus de documentació i sàpiguen consultar-la adientment. Per aquesta raó, en les pràctiques desenvolupades en el present projecte, es fa especial referència a aquest tema.

## 3.2.Microchip

Microchip Technology va ser creat l'any 1986 a Chandler, Arizona, Estats Units, formant part de la indústria dels semiconductors i la microelectrònica. Atès que era proveïdor de memòries programables no volàtils, processadors de senyal digital, circuits integrats i microcontroladors. Essent el desenvolupador i fabricant de la famosa família de microcontroladors PIC, els quals estan basats en arquitectura Harvard, emprant un joc d'instruccions de tipus RISC.

A més, a fi de facilitar i donar suport en la programació dels seus dispositius, ofereix eines de desenvolupament i programació. Com ara, entorns integrats de desenvolupament, compiladors, emuladors i depuradors.

Actualment, és una de les empreses líders en el mercat dels microcontroladors. L'any 2016, amb la compra de l'empresa Atmel, es va consolidar com a proveïdor líder en el mercat de solucions de control de sistemes *embedded*. Vist que ofereix una àmplia gamma de microcontroladors, els quals es detallen a continuació.

Pel que fa a Atmel, era una empresa de semiconductors, fundada l'any 1984, enfocada a proporcionar solucions per a sistemes encastats. És per això, que encara que el seu principal

producte eren els microcontroladors, també oferien alguns dels elements perifèrics necessaris, com memòries EEPROM i Flash, dispositius de radiofreqüència i sistemes tàctils.

L'any 1995, va ser de les primeres empreses a utilitzar les llicències de fabricació d'ARM per a crear i comercialitzar dispositius basats en la seva IP.

Si bé, des de l'any 1997, també distribuïa la seva pròpia família de microcontroladors, basada en l'arquitectura desenvolupada per ells, AVR, de 8 o 32 bits. Concretament, els AVR es caracteritzen per a estar dissenyats a partir d'una arquitectura Harvard, utilitzant instruccions de tipus RISC. Un altre tret a destacar, és que van ser ideats amb l'objectiu d'executar eficientment llenguatges d'alt nivell, proporcionant un disseny simple i facilitant la programació. És més, també oferia un entorn de desenvolupament integrat, l'Atmel Studio 7, el qual era gratuït.

Gràcies a aquestes característiques, actualment, els microcontroladors AVR, especialment els de 8 bits, compten amb una gran comunitat de seguidors en el món dels sistemes encastats, tan professionals, aficionats, com en aplicacions didàctiques.

Un altre fet que els ha ajudat a popularitzar-se, ha sigut que la plataforma de maquinari de codi obert, Arduino, utilitzi microcontroladors AVR en la gran majoria de les seves plaques.

Així doncs, amb motiu de l'adquisició d'Atmel, Microchip és capaç d'oferir un gran ventall de possibilitats, en funció de les necessitats de cada aplicació. A continuació, amb la finalitat d'estructurar les idees i característiques prèviament esmentades, es mostra una taula resum de les diferents famílies de microcontroladors amb les diferències clau.

| | **PIC** | **AVR** | **ARM** |
|---|---|---|---|
| *Ample de bus* | 8/16/32 bit | 8 (majoria)/32 bit | 32 (majoria)/64 bit |
| *Joc d'instruccions* | RISC | RISC | RISC |
| *Arquitectura de memòria* | Harvard | Harvard modificada | Harvard modificada |
| *Velocitat de la majoria d'instruccions* | 1 (cicle de rellotge/instrucció) | 1 (cicle de rellotge/instrucció) | 1 (cicle de rellotge/instrucció) |
| *IDE* | Semi-Gratuït | Gratuït | Gratuït |
| *Creador* | Microchip | Atmel | ARM |

***Taula 1.*** *Famílies de Microcontroladors de Microchip.*

En primer lloc, convé prestar especial atenció a la fila on s'especifica la velocitat de la majoria d'instruccions, ja que pot causar confusió. Així, a continuació, es detalla el temps d'execució d'una instrucció pel cas dels microcontroladors AVR.

Primerament, cal recordar els conceptes introduïts anteriorment sobre el mètode *pipelining*, el qual és possible en una arquitectura RISC amb una organització de memòria Harvard. Com es pot constatar en la Taula 1, aquestes dues premisses es compleixen pels xips AVR.

Concretament, tal com es detalla en el *datasheet*[3] del microcontrolador emprat en el present projecte:

[3] http://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061B.pdf Consultat 15/03/2021

*"Instructions in the program memory are executed with a single level pipelining. While one instruction is being executed, the next instruction is pre-fetched from the program memory. This concept enables instructions to be executed in every clock cycle".*

En la següent figura, es pot observar l'execució d'instruccions amb un únic nivell de *pipeline*. On la CPU duu a terme paral·lelament l'execució de la primera instrucció amb la cerca de la segona, governada pel rellotge $clk_{CPU}$.

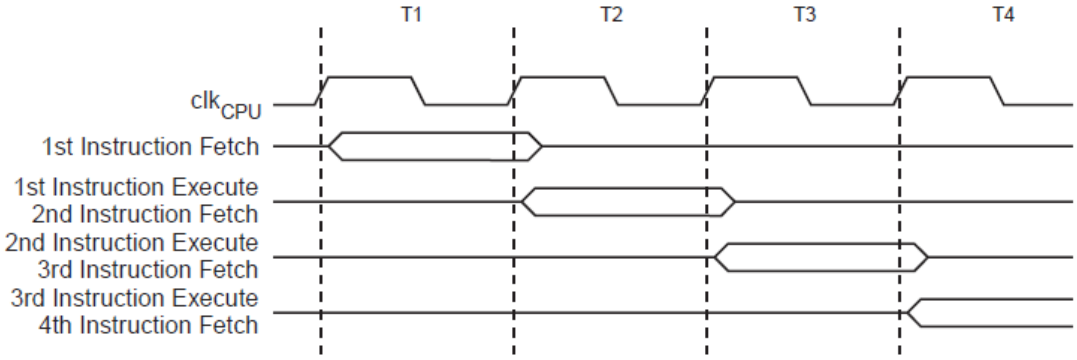



***Figura 7**. Cera i Execució d'Instruccions paral·leles en microcontroladors AVR [5].*

En conseqüència, com que les tres famílies representades en la taula anterior utilitzen aquest recurs, aconsegueixen executar la majoria d'instruccions en un sol cicle de rellotge.

En segon lloc, convé recordar que mentre PIC i AVR són marques comercials, sota les quals s'inclouen les arquitectures de microcontrolador desenvolupades per cada creador, ARM, en realitat, són microprocessadors dels quals se'n pot comprar la seva llicència de nucli. En aquest cas, Microchip ven dispositius basats en aquesta arquitectura, als quals hi ha afegit els elements necessaris, memòria i altres perifèrics, per a obtenir una altra família de microcontroladors.

Com es pot observar en la Taula 1, a simple vista, sembla que les tres famílies de microcontroladors siguin iguals. No obstant això, són ben diferents, ja que cada una té la seva arquitectura. La qual implica diferències en els registres, la configuració de la memòria, els mètodes d'adreçament, consum d'energia, com interactua la CPU amb els perifèrics, joc d'instruccions, entre altres trets a tenir en compte.

Aquesta diversitat resulta en el fet que cada família té la seva pròpia estructura pel desenvolupament: llenguatge assemblador, IDE i interfícies de programació. Convé ressaltar que els de tipus AVR i ARM sovint són més *compiler-friendly* i flexibles.

En conclusió, en funció de l'aplicació i les necessitats d'aquesta, se selecciona un dispositiu o altre.

### 3.3. Família AVR

El microcontrolador emprat en el present projecte de fi de grau forma part de la família AVR. Consegüentment, a continuació, s'exposen les principals característiques i avantatges que presenta aquesta família. I, més endavant, es detallen les particularitats del xip en qüestió, essent aquestes necessàries per a poder assolir la solució adequada a cada problema, amb una programació òptima.

En primer lloc, cal conèixer què s'entén per família de microcontroladors, una família es compon de dispositius que han estat desenvolupats seguint les mateixes línies de disseny, és a dir, tenen les mateixes característiques bàsiques per a formar el que s'anomena una *microcontroller unit* (MCU) o *core*. En concret, aquestes particularitats comunes poden ser l'arquitectura, el format de les instruccions, el mapejat de memòria o funcionalitats que comparteixen, com ara protocols de comunicació o interfícies d'usuari, entre altres.

Així doncs, cada família es constitueix per múltiples microcontroladors, els quals comparteixen el mateix disseny del nucli, però, evidentment, es diferencien per variacions en el nombre de pins, quantitat de memòria, tipus de perifèrics, entre altres paràmetres i funcions.

A continuació, es proporciona una perspectiva general sobre la línia de productes AVR, destacant-ne les principals prestacions del nucli.

Primerament, cal destacar que és la família de microcontroladors dominant en el mercat de 8 bits, oferint la millor solució per a gairebé qualsevol aplicació de sistema encastat. Atès que aquests dispositius ofereixen una combinació ideal entre rendiment, eficiència energètica i flexibilitat de disseny. Punts de gran rellevància pels dissenyadors de sistemes *embedded*.

En concret, al llarg dels anys, aquesta família s'ha consolidat com a referent en aquest exigent món, gràcies als següents avantatges, els quals han permès l'optimització d'aquestes MCU de 8 bits. D'aquesta afirmació, ja se'n pot extreure un dels principals trets distintius dels dispositius AVR, tots ells treballen amb un ample de bus de 8 bits, és a dir, la CPU només pot treballar amb aquest nombre de bits de dades a la vegada. En el cas d'haver de processar dades més grans, aquestes s'han de dividir en fragments de 8 bits.

D'una banda, amb l'objectiu de reduir el temps de desenvolupament, són els primers i únics nuclis de 8 bits dissenyats per a l'execució eficient de codi C.

De l'altra, presenten un elevat rendiment, el qual és imprescindible per a satisfer les necessitats fonamentals de moltes aplicacions, que s'assoleix a conseqüència de les següents característiques. Com és ben conegut, estan dissenyats sobre una arquitectura Harvard, amb memòria Flash per al codi i memòria SRAM per les dades, i utilitzen un joc d'instruccions de tipus RISC, reduint la complexitat del sistema. La combinació d'aquests dos trets, permet executar la majoria d'instruccions en un sol cicle.

A més, la simplicitat de la CPU també es pot observar en el joc d'instruccions disponible, ja que aquest és més regular que en els microcontroladors PIC de 8 bits. No obstant això, com es detalla més endavant, no és ortogonal, és a dir, no es pot utilitzar qualsevol mode d'adreçament en qualsevol instrucció. Fet que eleva lleugerament el grau de complexitat de la programació.

Per acabar, un altre paràmetre que influeix positivament en l'eficiència dels microcontroladors és el baix consum energètic. En aquest cas, s'assoleix mitjançant la utilització de la *picoPower Technology*, una tecnologia pròpia de Microchip, desenvolupada específicament per als components de les famílies AVR i AVR32. Els aspectes claus que permeten aquest estalvi són l'ús de sis *sleep modes*, conjuntament amb la capacitat de tornar ràpidament a l'execució normal, el que es coneix com a *fast wake-up feature*.

En un altre ordre de coses, la família AVR és molt nombrosa, ja que la conformen més de 70 dispositius diferents, els quals es poden agrupar en les quatre categories següents, ordenades de menys a més complexitat:

- Classic AVR: Són els xips originals de la família AVR. Malgrat que en els nous dissenys ja no s'utilitzen, perquè han sigut substituïts pels actuals.

- Tiny AVR: Encara que són els dispositius més petits, són perfectes per aplicacions amb limitacions considerables que, així i tot, necessiten un processador potent.
  Si bé és cert que disposen de poca memòria i pocs pins, tenen un elevat nivell d'integració en espais compactes.

- Mega AVR: Aquesta categoria ofereix l'oferta més àmplia de microcontroladors amb múltiples opcions de perifèrics incorporats, amb una bona capacitat de memòria i un nombre considerable de pins d'entrada i sortida, de 32 a 100. És per això, que són els més populars, essent els adequats per una àmplia gamma d'aplicacions.

- AVR XMega: Aquests dispositius ofereixen un resultat extraordinari en aplicacions complexes, on es requereix una elevada quantitat de memòria de programa i velocitat.

Finalment, un altre avantatge que presenten tots els integrants de la família AVR és que són *user-friendly*, és a dir, presenten un funcionament senzill i permeten una experiència còmoda. Gràcies al fet que presenten un suport excel·lent per a la generació de codi C, oferint un conjunt d'eines de desenvolupament, generalment conegut com a *toolchain,* de gran qualitat amb avançades prestacions.

Encara més, aquests recursos de suport al disseny, estan integrats en un mateix ecosistema compatible per a tots els nuclis AVR. Com a resultat, l'usuari només s'ha de familiaritzar amb una sola plataforma.

Esdevenint punts a tenir en compte a l'hora de maximitzar la funcionalitat i disminuir el temps de desenvolupament de qualsevol aplicació. En aquest cas, en tractar-se d'una aplicació amb finalitats didàctiques, és especialment remarcable.

A més, ofereixen un elevat nivell de portabilitat de software entre els diferents components de la família.

### 3.4. El Processador

Al llarg d'aquest apartat, s'exposen detalladament les principals característiques i una visió general de l'arquitectura del microcontrolador objecte d'estudi. Vist que són coneixements imprescindibles per a poder treballar amb aquest dispositiu.

Com és ben conegut, el processador ATmega328P forma part de la família AVR, més concretament de la categoria Mega AVR. En conseqüència, és un xip amb un bus de dades de 8 bits, basat en l'arquitectura RISC. La qual ha sigut millorada per AVR, per tal d'obtenir un baix consum energètic i un elevat rendiment.

Actualment, és un dels microcontroladors més utilitzats del mercat, gràcies al fet de disposar de diverses característiques que el fan idoni per a moltes aplicacions, no només en entorns industrials sinó també per a projectes d'aficionats. Atès que li permeten oferir una flexibilitat i una execució superior en àrees de vital importància per a les modernes aplicacions dels sistemes encastats.

Tot seguit, es detallen breument les qualitats més destacables i, més endavant, s'exposen en detall. Moltes d'aquestes funcionalitats es poden observar en el següent diagrama de blocs del dispositiu, en la Figura 8.

Pel que fa a les característiques elèctriques, admet un rang de tensió de funcionament més ampli que la majoria de xips, d'1,8 V a 5,5 V. Si bé, normalment s'utilitza la tensió de 5 V com a estàndard. Quant al consum, l'usuari pot seleccionar d'entre sis *sleep modes* diferents, els quals permeten assolir un gran estalvi energètic en funció de les necessitats de funcionament a cada instant.

Les funcions perifèriques a tenir en compte des del punt de vista del programador del dispositiu són:

- 6 canals PWM.
- 3 *timers* flexibles incorporats, dos d'ells de 8 bits i l'altre de 16 bits.
- Un rellotge de temps real amb oscil·lador independent, generalment conegut per la sigla en anglès RTC de *real-time clock*.
- Un convertidor analògic-digital.
- Suporta els següents protocols de comunicació: USART, SPI i una interfície sèrie de dos fils orientada a bytes.
- Com a element de seguretat, disposa d'un *Watchdog timer*.

També disposa d'una unitat específica per a gestionar les interrupcions, tant internes com externes, i els recursos necessaris per a atendre-les correctament.

A més, uns altres trets a considerar són les principals propietats del MCU. D'una banda, amb motiu d'esser desenvolupat a partir d'un joc d'instruccions RISC, l'usuari té a la seva disposició 131 potents instruccions, la majoria de les quals, gràcies a la tècnica de *pipelinig*, s'executen en un sol cicle de rellotge. Així com també pot utilitzar fins a 32 registres de propòsit general i 23 línies d'entrada i sortida d'ús general.

De l'altra, la gestió de la memòria. Com que tots els membres de la família AVR estan dissenyats a partir d'una arquitectura Harvard, tenen dos espais de memòria principal, amb els seus respectius busos independents. La memòria de dades, de tipus SRAM, i la de programa, de tipus Flash. En aquest cas, l'ATmega328P també té un espai de memòria EEPROM per a emmagatzemar informació. Si bé, tal com s'indica en el següent apartat, els tres espais són lineals i regulars.

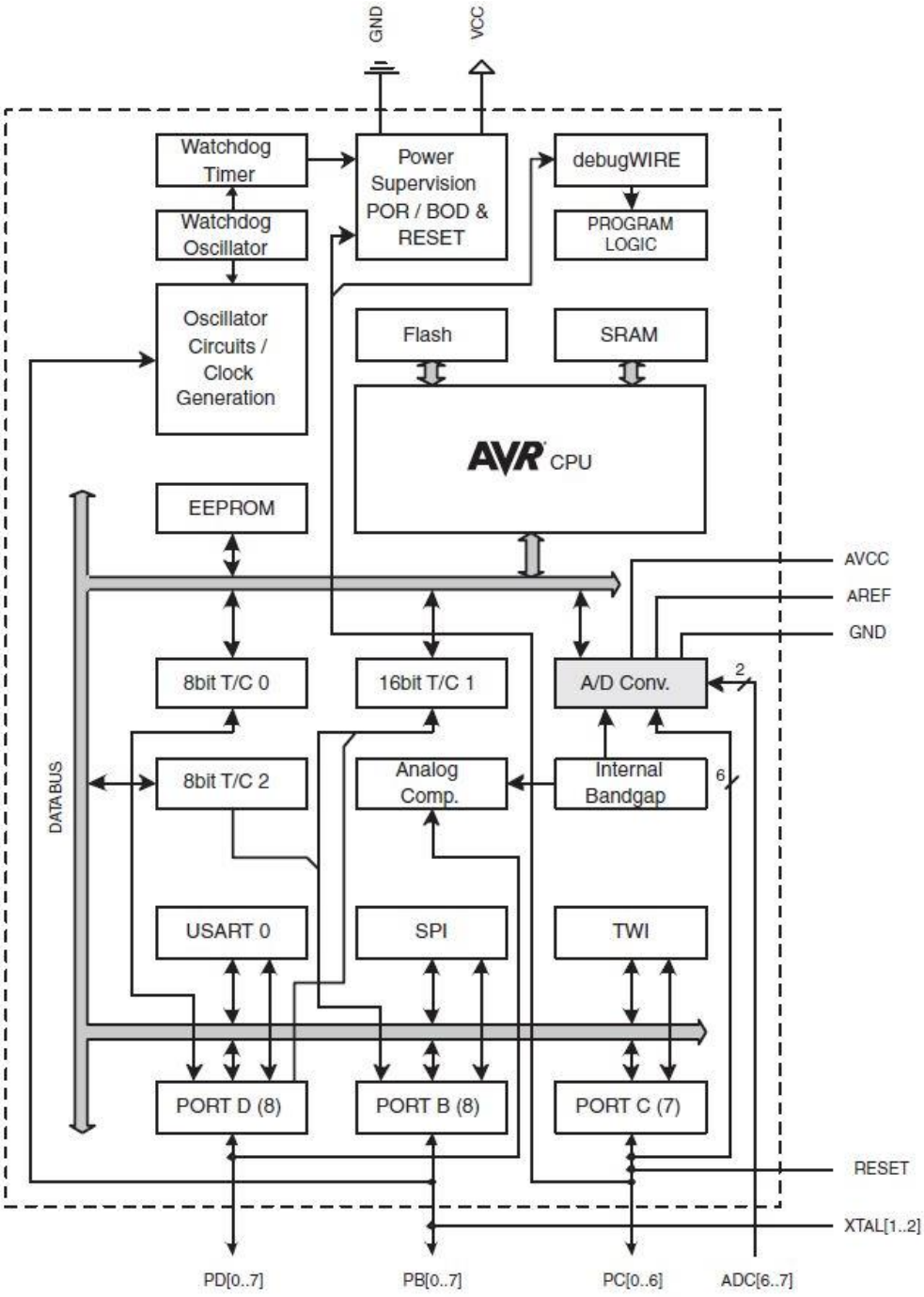



***Figura 8.*** *Diagrama de blocs del processador ATmega328P [6].*

En últim terme, cal tenir present que està disponible en diferents encapsulats. L'elecció d'un o altre determina, evidentment, les dimensions. Però també el nombre de pins, 28 o 32, el *pinout* i el nombre de canals de l'ADC.

Com es detalla més endavant, a fi de poder dur a terme la part pràctica del present projecte de fi de grau s'utilitza el kit d'avaluació ATmega328P Xplained Mini, el qual porta integrat el microcontrolador ATmega328P-MUR en l'encapsulat 32M1-A. En conseqüència, l'usuari disposa de 32 pins i un convertidor analògic-digital de 8 canals.

### *3.4.1. CPU*

A fi de poder entendre el funcionament de la CPU i així poder programar adequadament el dispositiu, cal conèixer les següents característiques del nucli.

A continuació, es mostra el diagrama de blocs funcional de l'arquitectura AVR. En el qual es poden observar els principals components que garanteixen la correcta execució del programa.

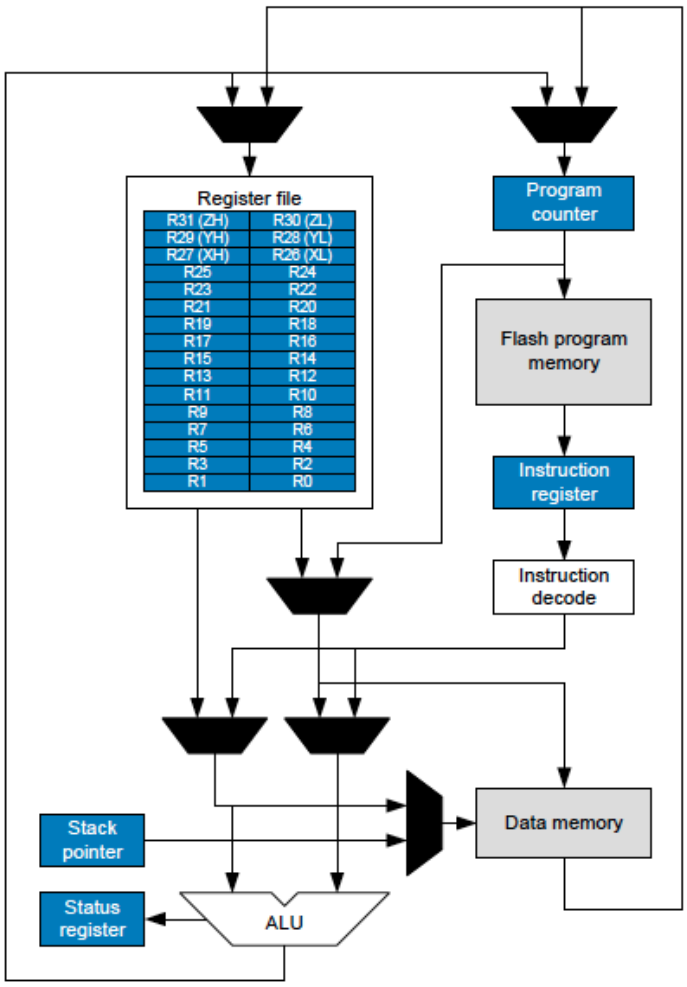



***Figura 9.*** *Diagrama de Blocs de l'Arquitectura AVR [7].*

D'una banda, l'ALU duu a terme operacions lògiques i aritmètiques entre registres o entre un registre i una constant, amb només un cicle de rellotge. Així, opera amb connexió directa amb els 32 registres de propòsit general. Pel cas de l'execució d'una operació aritmètica, el registre d'estat és actualitzat per a indicar el resultat.

De l'altra, els ja mencionats 32 registres de propòsit general de 8 bits, els quals s'emmagatzemen en un arxiu de ràpid accés, anomenat *Register File* i que permet l'accés en un sol cicle de rellotge. Consegüentment, s'aconsegueix una *Single-cycle ALU operation*, és a dir, es realitza la lectura dels dos operands, s'executa l'operació, es guarda el resultat obtingut en el banc de registres i s'actualitza el registre d'estat en un sol cicle de rellotge.

Convé ressaltar que el flux del programa es proporciona mitjançant salts condicionals i incondicionals i crides a funcions, amb accés a tot l'espai de memòria. La majoria d'instruccions AVR tenen un únic format de paraula de 16 bits. Així, com es detalla més

endavant, cada adreça de memòria de programa conté una instrucció de 16 o 32 bits, la qual ha de ser descodificada per a poder-la executar.

### *3.4.2. Registres de la CPU*

Els registres disponibles en l'arquitectura AVR permeten al programador realitzar càlculs, accedir a la memòria i a les dades d'entrada i sortida i prendre decisions. Evidentment, com més registres, la programació resulta més fàcil. Si bé, repercuteix en un augment de la CPU, que implica més consum energètic i un cost econòmic major. És per això, que a l'hora de seleccionar el microcontrolador per a cada projecte, cal analitzar quines seran les necessitats reals per a no obtenir un producte sobredimensionat.

Al llarg d'aquest apartat, s'aprofundeix en els registres que l'usuari ha de manipular per a dissenyar la solució. Per això, no es detalla ni el *Program Counter* ni l'*Stack Pointer*, ja que aquests són actualitzats per la CPU automàticament.

Tal com es pot observar en la Figura 9, el banc de registres generals conté 32 registres de propòsit general de 8 bits, els quals són assignats als primers 32 bytes de la memòria de dades.

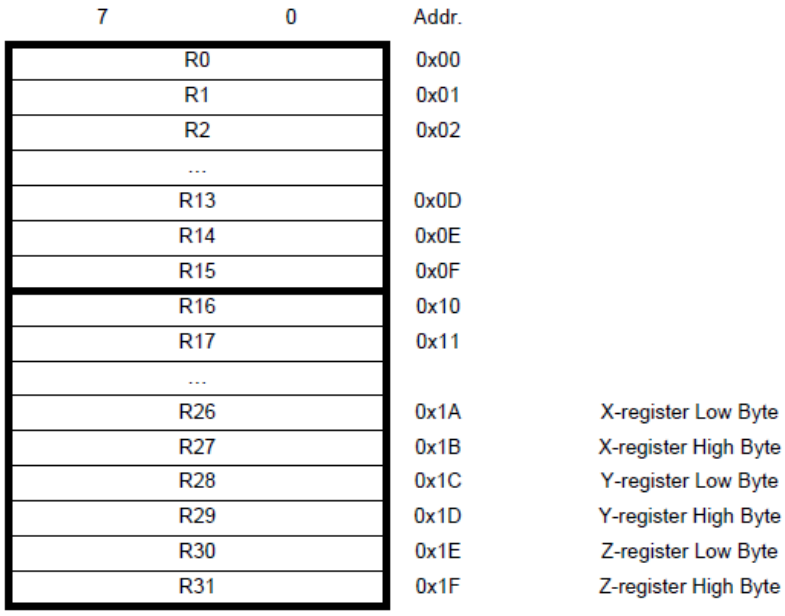



***Figura 10.*** *Estructura dels Registres de Propòsit General [6].*

Com s'indica en la Figura 10, els sis últims registres, del r26 al r31, tenen funcions de valor afegit. Atès que es poden utilitzar com a tres apuntadors de 16 bits amb adreçament indirecte a l'espai de dades o a la memòria de programa. Es coneixen com a registres d'índex i es denominen registre X, Y i Z.

També hi ha el registre d'estat, generalment anomenat SREG, de l'acrònim en anglès. Gràcies al fet que conté la informació sobre l'última instrucció aritmètica executada, en les operacions condicionals, es poden prendre decisions per a determinar el flux del programa i les sortides del sistema.

Així doncs, l'SREG està format per 8 *flags*, és a dir, per vuit bits, de les quals se'n detalla la funcionalitat en la següent taula. No obstant això, pel cas del bit 7, s'explica amb més detall en l'apartat sobre interrupcions.

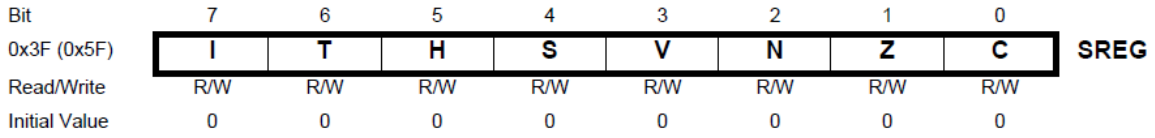



***Figura 11**. Detall dels Bits del Registre d'Estat [6].*

| Flag | Nom | Funcionalitat |
|---|---|---|
| I | Global Interrupt Enable | Si I = 1; S'habiliten totes les interrupcions.<br>Si I = 0; Cap interrupció està permesa, independentment de la configuració individual. |
| T | Bit Copy Storage | És utilitzat en les instruccions BLD (*Bit Load*) i BST (*Bit Store*), com a font o destí del bit sobre el qual es treballa. |
| H | Half Carry | És útil en operacions BCD. |
| S | Sign Bit | S = N ⊕ V; Operació OR exclusiva entre el bit 2 i 3. |
| V | Overflow | Bandera de desbordament en operacions en complement A 2. |
| N | Negative | Si N = 1; Indica un resultat negatiu en una operació aritmètica o lògica. |
| Z | Zero | Si Z = 1; Indica que el resultat en una operació aritmètica o lògica és zero. |
| C | Carry | Si C = 1; Indica que hi ha hagut un desbordament en el resultat d'una operació aritmètica o lògica. |

***Taula 2.** Funcionalitat dels Bits del Registre d'Estat.*

A fi d'utilitzar correctament el SREG, cal tenir en compte que aquest s'actualitza després de cada operació de l'ALU. És per això, que en molts casos, s'elimina la necessitat d'utilitzar les instruccions específiques de comparació. Fet que permet un codi més ràpid i compacte.

A més, quan es treballa amb interrupcions, cal tenir present que el registre d'estat no es guarda automàticament quan s'entra en una interrupció, ni es recupera en sortir-ne, sinó que cal fer-ho mitjançant el software.

### 3.5. La Memòria

Des del punt de vista del microprocessador, la memòria és un dels dos perifèrics, juntament amb la unitat d'entrada i sortida, essencial per a poder dur a terme les tasques definides de manera fiable i eficient.

No obstant això, com que és un element hardware, és un recurs limitat. En termes generals, el principal motiu perquè els microcontroladors no disposin d'una elevada quantitat de memòria, és que aquesta ocupa molta àrea del silici. En conseqüència, el cost del producte final s'incrementa i, el que és més important, la probabilitat de fallada augmenta.

Si bé, la majoria de sistemes encastats no necessiten grans quantitats de memòria, ja que donen solució a tasques ben definides i acotades, amb un elevat nivell de qualitat.

A fi d'utilitzar aquest recurs adequadament, cal conèixer la gestió i les diferents tipologies que existeixen.

Com és ben conegut, els sistemes digitals estan basats en el sistema binari. És per això que l'espai de memòria s'organitza en bytes. Cada byte constitueix una posició de memòria, la qual té una adreça única, normalment expressada en format hexadecimal.

Aquest tipus d'organització es coneix com a *byte addressable*. Concretament, tal com es detalla en el llibre *Introduction to Embedded Systems*[4]:

*"These bits are organized in n-bit words, working as a register, usually referred to as cell or location. The contents of cells is a basic unit of information called memory word. In addition, each memory location is identified by a unique identifier, its memory address, which is used by the CPU to either read or write over the memory word stored at the location."*

En aquest cas, com s'ha comentat anteriorment, l'ATmega328P ha estat desenvolupat a partir d'una arquitectura Harvard, consegüentment, integrats en el mateix xip, presenta dos espais de memòria independents, un per a les dades i l'altre per a les instruccions, amb busos de dades de 8 bits.

A més, la seva arquitectura determina que cada cel·la de memòria conté la informació de 8 bits, és a dir, d'un byte.

Així doncs, la memòria es classifica segons dos criteris principals: la capacitat d'escriptura i la capacitat de conservar les dades escrites. En altres paraules, la memòria és de tipus volàtil si en eliminar l'alimentació, les dades es perden. En canvi, és no volàtil si pot conservar les dades encara que no tingui alimentació.

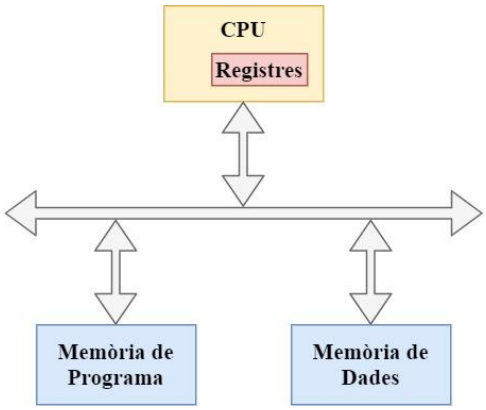



***Figura 12.*** *Estructura de la Memòria d'un Microcontrolador.*

En la Figura 12, es pot observar l'estructura de la memòria d'un microcontrolador amb les tres varietats principals. D'una banda, en la part inferior, apareixen els dos tipus de memòria que es detallen a continuació. De l'altra, els registres exposats anteriorment, els quals són de tipus volàtil.

### *3.5.1. Memòria de Programa*

Les instruccions que formen el programa es guarden en la memòria de tipus flash, és a dir, memòria no volàtil amb capacitat d'escriptura i, evidentment, de lectura. La tècnica d'escriptura es va desenvolupar a partir de la utilitzada en les EEPROM, les quals permeten

[4] M. Jiménez, R. Palomera i I. Couvertier, Introduction to Embedded Systems: Using Microcontrollers and the MSP430, EEUU: Springer, 2014.

un esborrat i una programació elèctrica. Però en aquest cas, proporciona una major flexibilitat i funcionalitat. Si bé, continua tenint una vida útil limitada, és a dir, en funció de la seva naturalesa, permet un nombre finit de cicles d'esborrat i escriptura. Pel cas de l'ATmega328P, pot suportar, com a mínim, 1000 cicles.

Com que conté el programa i, generalment, el codi és més extens que les dades del programa, el microcontrolador disposa de més quantitat de memòria flash que de la destinada a les dades. A més, un cop programat el dispositiu, durant l'execució del programa, aquest espai només té permís de lectura. Consegüentment, només s'hi pot sobreescriure quan el dispositiu és reprogramat.

El present microcontrolador té 32 kbytes *d'In-system reprogrammable Flash Program Memory*, organitzada en 16 k x 16, ja que totes les instruccions d'AVR tenen una amplada de 16 o 32 bits.

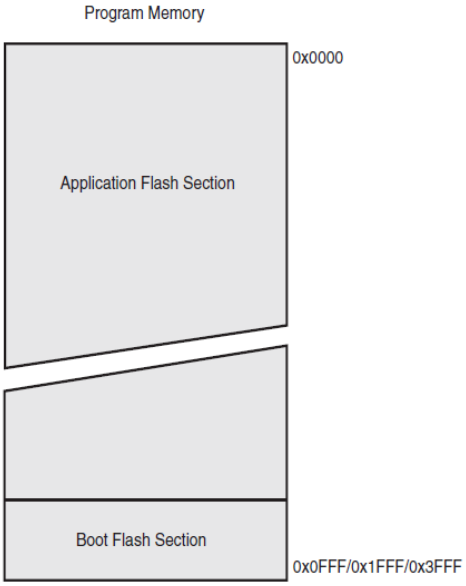



***Figura 13.*** *Mapa de la Memòria de Programa de l'ATmega328P [6].*

Com es pot observar en la Figura 13, on es mostra el mapa de la memòria de programa del processador, per la seguretat del software, l'espai de memòria flash està dividit en dues seccions: l'*Application Program Section*, on s'emmagatzemen les instruccions, i el *Boot Loader*.

El segon espai està reservat per a emmagatzemar-hi el *Bootloader*, el qual és un petit programa especial destinat a llegir dades d'una font externa per a reescriure el programa. Aquest només s'executa durant els segons després d'haver encès el microcontrolador o quan s'hi ha realitzat una inicialització, enviant el comptador de programa a l'adreça 0x0000.

Concretament, quan la CPU comença a executar el programa, ha de saber quina ha de ser la primera adreça a llegir. Així, hi ha dos mètodes per a determinar la primera instrucció a descodificar. Per un costat, sempre pot anar a llegir a la mateixa adreça. Per l'altre, pot utilitzar una taula de vectors que li permet començar en adreces diverses.

Encara que s'utilitzi el primer mètode, també és possible inicialitzar el programa en adreces diferents, simplement cal escriure una instrucció de salt en la primera posició.

Així mateix, existeix una instrucció per a l'assignació de taules dins de l'espai de memòria de programa, la LPM, de la sigla en anglès de *Load Program Memory*.

### *3.5.2. Memòria de Dades SRAM*

Atès que la memòria de dades del present microcontrolador és de tipus SRAM, és volàtil, és a dir, quan el dispositiu perd l'alimentació, les dades s'esborren. A més, a l'hora de programar, cal tindre en compte que és un recurs limitat. Per tant, cal gestionar-lo correctament. En aquest cas, l'usuari disposa de 2 kB de memòria interna SRAM.

Com s'ha comentat anteriorment, la quantitat de memòria en els sistemes *embedded* és força reduïda. Concretament, pel cas de la memòria SRAM, com que aquesta s'ha de refrescar constantment perquè no perdi la informació, suposa un consum d'energia que moltes vegades l'aplicació no pot suportar.

No obstant això, la quantitat de memòria també està restringida per l'amplada del bus de dades. Particularment, l'arquitectura AVR només permet direccionar 256 bytes directament, ja que l'amplada és de 8 bits.

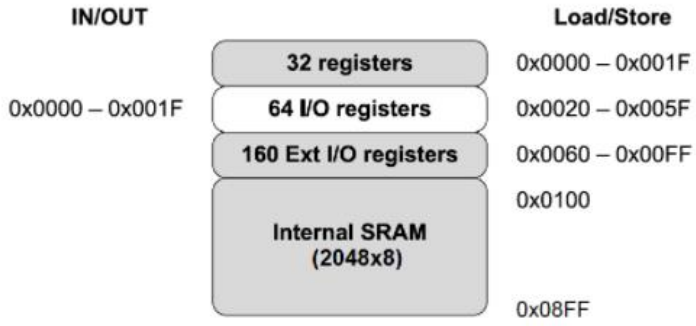



***Figura 14.*** *Mapa de la Memòria de Dades de l'ATmega328P [6].*

En la figura, es pot observar el mapa de la memòria de dades, el qual consta de diversos segments que es detallen a continuació.

En les primeres posicions de memòria, hi ha el banc de registres d'ús general detallat en l'apartat 3.4.2. Registres de la CPU.

Les següents 64 posicions allotgen els registres d'entrada i sortida, seguides per les 160 posicions d'*Extended I/O memory*. Les primeres estan reservades per funcions perifèriques, com ara registres de control i altres funcions d'entrada i sortida. Però com que aquest microcontrolador té més perifèrics, necessita més espai per a poder gestionar-los. És per això que existeixen aquests registres addicionals. Més endavant, s'aprofundeix sobre aquest tema.

Finalment, les últimes 2048 adreces permeten accedir a la memòria SRAM interna, la qual permet emmagatzemar temporalment resultats i variables intermèdies durant l'execució del programa.

Així, aquest espai de memòria és emprat per a diferents propòsits durant l'evolució de la seqüència. En primer lloc, hi ha un bloc d'espai reservat per a totes les variables globals i estàtiques del programa. Pel cas de les variables amb valor inicial, quan el programa comença, aquestes són inicialitzades des de la memòria flash, mitjançant una rutina generada automàticament pel compilador.

Seguidament, hi ha l'espai anomenat *heap* o zona d'emmagatzematge lliure, ja que guarda les dades dinàmiques de manera aleatòria. Gràcies a ell, es pot reservar memòria dinàmicament.

En tercer lloc, hi ha l'*stack*, és a dir, la pila del sistema. La qual es comporta com un buffer LIFO, de la sigla en anglès de *Last In, First Out*, on es guarden les variables locals i les involucrades en les interrupcions i les crides a funció. En són un bon exemple els registres PC i SP.

Finalment, hi ha l'espai entre els dos blocs anteriors, la memòria disponible lliure. La mida de la qual es modifica en funció de l'evolució de la pila i el *heap*. Cal prestar especial atenció en què l'aplicació no es quedi sense aquest recurs, sinó el programa es pot comportar de forma no controlada i provocar problemes.

Amb relació a la memòria d'entrada i sortida, cal destacar els següents paràgrafs del manual.

*"The ATmega328P I/Os and peripherals are placed in the I/O space. All I/O locations may be accessed by the LD/LDS/LDD and ST/STS/STD instructions, transferring data between the 32 general purpose working registers and the I/O space. I/O Registers within the address range 0x00 - 0x1F are directly bit accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions.*

*When using the I/O specific commands IN and OUT, the I/O addresses 0x00 - 0x3F must be used. When addressing I/O Registers as data space using LD and ST instructions, 0x20 must be added to these addresses. The ATmega328P is a complex microcontroller with more peripheral units than can be supported within the 64 location reserved for the IN and OUT instructions. For the Extended I/O space from 0x60 - 0xFF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used."*

En altres paraules, totes les entrades i sortides i els senyals relatius als perifèrics estan mapejats en l'espai de memòria I/O i l'ampliat, als quals s'hi pot accedir a través de les instruccions LD/LDS/LDD i ST/STS/STD, transferint les dades entre els registres d'ús general i l'espai d'E/S, mitjançant el bus de dades estàndard.

Si bé, les adreces entre la 0x00 i 0x1F, és a dir, el banc de registres, s'hi pot accedir directament a nivell de bit mitjançant les funcions SBI i CBI, i llegir el valor amb les instruccions SBIS i SBIC.

A més, per tal d'accedir a la secció d'*I/O Memory*, es pot realitzar de dues maneres. D'una banda, utilitzant l'anomenat *In/Out Data Bus*, el qual hi té accés directe utilitzant el rang d'adreces de la 0x00 a la 0x1F. De l'altra, utilitzant el bus estàndard però amb un offset de direcció de 0x20 en la instrucció d'accés. En conseqüència, cal tindre en compte que en funció del tipus d'instrucció, els 64 registres I/O tenen una adreça o altra.

Com es pot constatar, els aspectes relatius a les entrades i sortides i als perifèrics del microcontrolador són força complexos i, a l'hora, importants per a assolir el coneixement adequat sobre el funcionament. És per això, que en el següent punt es detallen més minuciosament.

Finalment, convé destacar que es pot accedir a tot l'espai de memòria de dades SRAM a través dels cinc modes d'adreçament, els quals també s'expliquen més endavant.

### *3.5.3. Memòria de Dades EEPROM*

Per acabar, la majoria de microcontroladors també tenen una quantitat limitada d'EEPROM, de la sigla en anglès d'*Electrically Erasable Programmable Read Only Memory*, de tipus no

volàtil. Així, s'utilitza per a emmagatzemar dades permanentment, fins i tot si es perd l'alimentació, com ara paràmetres de configuració o constants del programa.

No obstant això, en comparació amb la memòria SRAM, l'accés a aquest espai de memòria és bastant més lent. A més, es requereixen instruccions especials, ja que només es pot llegir byte a byte, i les dades es poden corrompre si el voltatge d'alimentació cau per sota del mínim establert per a funcionar correctament. En definitiva, la seva utilització és més complexa, resultant ser fins i tot una mica incòmode.

L'ATmega328P conté 1024 bytes de memòria EEPROM amb una vida útil de 100000 cicles d'esborrat i escriptura. Ara bé, no té cap limitació en la lectura. L'accés entre la CPU i aquest espai de memòria es realitza mitjançant 4 registres: EEPROM *Address Register Low and High Byte*, EEPROM *Data Register* i EEPROM *Control Register*. Anomenats respectivament: EEARL, EEARH, EEDR i EECR.

### *3.5.4. Modes d'Adreçament*

Els dispositius que formen part de la família AVR poden accedir a l'espai de la memòria de dades SRAM mitjançant els cinc modes d'adreçament que es detallen a continuació.

No obstant això, primer cal tenir present que per a seleccionar el mode d'adreçament adequat, cal tenir en compte on està ubicat l'operand en el mapa de memòria i quan es coneix la seva adreça, en temps d'execució o quan s'assembla el projecte.

- **Mode d'Adreçament Directe**: mitjançant el qual es pot accedir a tot l'espai de memòria, utilitzant les instruccions sts, per a emmagatzemar el valor d'un registre en una determinada adreça, i lds, per a descarregar el valor contingut en una adreça al registre indicat.

- **Mode d'Adreçament Indirecte**: Aquest mode i els tres següents s'utilitzen quan no es coneix l'adreça de memòria fins que no s'executa el programa. Atès que és una manera eficaç d'accedir o manipular una llista d'ubicacions de memòria dins d'un bucle.
  Si bé, aquest mode només està disponible per als tres registres d'índex, del r26 al r31, ja que aquests actuen com a apuntadors a l'adreça desitjada.

- **Indirecte amb desplaçament**: Mitjançant aquest mètode, es té accés a 63 posicions de memòria a partir de l'adreça base proporcionada pel registre Y o Z.

- **Indirecte amb pre-decrement:** A fi de poder avançar a través de la sèrie d'ubicacions seqüencials, cal incrementar o disminuir els registres x, y i z. En aquest cas, els registres d'índex es disminueixen.

- **Indirecte amb post-decrement:** En aquest mètode, els registres d'índex s'incrementen.

D'altra banda, per a accedir a la memòria de dades Flash dels dispositius AVR, cal utilitzar el mètode d'adreçament immediat. En concret, només els registres d'índex poden utilitzar aquest mètode, en el qual les dades es codifiquen amb la instrucció.

A continuació, es mostra una taula resum amb els modes d'adreçament disponibles amb les respectives instruccions de càrrega i descàrrega de dades.

| | Espai de memòria | | |
|---|---|---|---|
| **Mètode d'adreçament** | SRAM | I/O | Flash |
| Directe | lds, sts | in, out | |
| Indirecte (-/pre/post decrement) | ld, st | | lpm, spm |
| Indirecte amb desplaçament | ldd, std | | |
| Immediat | | | ldi |

***Taula 3.*** *Instruccions de Càrrega i Descàrrega de Dades en l'Espai de Memòria.*

### 3.6. Joc d'Instruccions de la Família AVR

Com és ben conegut, la família del microcontrolador objecte d'estudi és del tipus RISC, és a dir, té un joc d'instruccions reduït. Així i tot, ofereix 131 potents instruccions, les quals resulta impossible conèixer-les de memòria durant les primeres preses de contacte dels dispositius AVR.

No obstant això, a fi de poder escriure correctament els programes, l'usuari hi ha d'estar familiaritzat. Per aquest motiu, és imprescindible estar habituat a consultar el manual del joc d'instruccions[5], per a poder obtenir la informació necessària de forma ràpida i eficaç.

Gràcies a aquest document, es poden consultar totes les instruccions disponibles. A més, tal com es detalla en el document citat anteriorment: *"Each instruction has its own section containing functional description, it's opcode, and syntax, the end state of the status register, and cycle times."*

A continuació, es mostra una taula resum amb la nomenclatura utilitzada en les instruccions, la qual és essencial per a poder comprendre correctament cada instrucció.

| **Registres i Operands** | **Definició** |
|---|---|
| Rd | Registre font i destí del banc de registres |
| Rr | Registre font del banc de registres |
| R | Resultat després de l'execució de la instrucció |
| K | Constant de dada |
| k | Constant d'adreça |
| b | Posició de bit (0...7) en el banc de registres o en els registres I/O |
| s | Posició de bit (0...7) en el registre d'estat |
| X, Y, Z | Registre de direccions indirecte |
| A | Direcció de memòria dels I/O |
| q | Desplaçament per l'adreçament indirecte (6-bit) |

***Taula 4.*** *Nomenclatura del Joc d'Instruccions.*

Concretament, el format de les instruccions és el següent. D'una banda, ressaltat de color taronja, hi ha el mnemònic, és a dir, el conjunt de caràcters que simbolitzen l'operació a realitzar. Evidentment, mitjançant aquesta tècnica de representació, es facilita el procés de

[5] https://ww1.microchip.com/downloads/en/DeviceDoc/AVR-Instruction-Set-Manual-DS40002198A.pdf Consultat 15/04/2021

programació, ja que els mnemònics són més fàcils de recordar que els codis d'operació, generalment anomenats Opcode. Tal com es pot constatar en el següent exemple.

De l'altra, tota operació està composta pels operands sobre els quals s'executa l'operació, els quals estan marcats de color blau. Pel cas del següent exemple, suma el valor dels dos registres i l'emmagatzema en el primer, Rd.

| ADD | Rd, Rr |
|---|---|

16-bit Opcode:

| 0000 | 11rd | dddd | rrrr |
|---|---|---|---|

***Figura 15.*** *Exemple del Format de les Instruccions en Assemblador.*

Cal tindre en compte que, en funció del mode d'adreçament utilitzat, l'adreça de memòria de l'operand es pot especificar d'una manera o altra. Pel cas del present microcontrolador, admet els següents modes d'adreçament principals:

- **Mode directe amb un registre:** L'operand es troba en el registre destí.

- **Mode directe amb dos registres:** Els operands estan inclosos en els registres font i destí, el resultat es guarda en Rd.

- **Mode directe I/O:** Com que el microcontrolador té més unitats perifèriques de les que es poden gestionar amb les 64 posicions de memòria reservades per aquest fi, existeix aquest mètode. En el qual, la instrucció conté la direcció de memòria del *extended* I/O que es vol accedir.

- **Mode directe:** L'operand conté la direcció de memòria de l'operand.

- **Mode indirecte:** L'adreça de l'operand és el contingut de l'apuntador X, Y o Z.

- **Mode indirecte amb offset**: En aquest cas, es realitza un offset sobre l'apuntador X, Y o Z.

De manera paral·lela, les instruccions es poden classificar segons la seva funcionalitat. Seguidament, es mostra el joc d'instruccions que suporta el microcontrolador utilitzat en el present projecte, organitzat en les quatre funcionalitats principals.

En primer lloc, hi ha les instruccions que realitzen operacions aritmètiques o lògiques:

| Mnemonic | Operands | Description | Operation | | | Flags | #Clocks AVRe | #Clocks AVRxm | #Clocks AVRxt | #Clocks AVRrc |
|---|---|---|---|---|---|---|---|---|---|---|
| ADD | Rd, Rr | Add without Carry | Rd | ← | Rd + Rr | Z,C,N,V,S,H | 1 | 1 | 1 | 1 |
| ADC | Rd, Rr | Add with Carry | Rd | ← | Rd + Rr + C | Z,C,N,V,S,H | 1 | 1 | 1 | 1 |
| ADIW | Rd, K | Add Immediate to Word | R[d + 1]:Rd | ← | R[d + 1]:Rd + K | Z,C,N,V,S | 2 | 2 | 2 | N/A |
| SUB | Rd, Rr | Subtract without Carry | Rd | ← | Rd - Rr | Z,C,N,V,S,H | 1 | 1 | 1 | 1 |
| SUBI | Rd, K | Subtract Immediate | Rd | ← | Rd - K | Z,C,N,V,S,H | 1 | 1 | 1 | 1 |
| SBC | Rd, Rr | Subtract with Carry | Rd | ← | Rd - Rr - C | Z,C,N,V,S,H | 1 | 1 | 1 | 1 |
| SBCI | Rd, K | Subtract Immediate with Carry | Rd | ← | Rd - K - C | Z,C,N,V,S,H | 1 | 1 | 1 | 1 |
| SBIW | Rd, K | Subtract Immediate from Word | R[d + 1]:Rd | ← | R[d + 1]:Rd - K | Z,C,N,V,S | 2 | 2 | 2 | N/A |
| AND | Rd, Rr | Logical AND | Rd | ← | Rd ∧ Rr | Z,N,V,S | 1 | 1 | 1 | 1 |
| ANDI | Rd, K | Logical AND with Immediate | Rd | ← | Rd ∧ K | Z,N,V,S | 1 | 1 | 1 | 1 |
| OR | Rd, Rr | Logical OR | Rd | ← | Rd v Rr | Z,N,V,S | 1 | 1 | 1 | 1 |
| ORI | Rd, K | Logical OR with Immediate | Rd | ← | Rd v K | Z,N,V,S | 1 | 1 | 1 | 1 |
| EOR | Rd, Rr | Exclusive OR | Rd | ← | Rd ⊕ Rr | Z,N,V,S | 1 | 1 | 1 | 1 |
| COM | Rd | One's Complement | Rd | ← | 0xFF - Rd | Z,C,N,V,S | 1 | 1 | 1 | 1 |
| NEG | Rd | Two's Complement | Rd | ← | 0x00 - Rd | Z,C,N,V,S,H | 1 | 1 | 1 | 1 |
| SBR | Rd,K | Set Bit(s) in Register | Rd | ← | Rd v K | Z,N,V,S | 1 | 1 | 1 | 1 |
| **Mnemonic** | **Operands** | **Description** | **Operation** | | | **Flags** | **#Clocks AVRe** | **#Clocks AVRxm** | **#Clocks AVRxt** | **#Clocks AVRrc** |
| CBR | Rd,K | Clear Bit(s) in Register | Rd | ← | Rd ∧ (0xFFh - K) | Z,N,V,S | 1 | 1 | 1 | 1 |
| INC | Rd | Increment | Rd | ← | Rd + 1 | Z,N,V,S | 1 | 1 | 1 | 1 |
| DEC | Rd | Decrement | Rd | ← | Rd - 1 | Z,N,V,S | 1 | 1 | 1 | 1 |
| TST | Rd | Test for Zero or Minus | Rd | ← | Rd ∧ Rd | Z,N,V,S | 1 | 1 | 1 | 1 |
| CLR | Rd | Clear Register | Rd | ← | Rd ⊕ Rd | Z,N,V,S | 1 | 1 | 1 | 1 |
| SER | Rd | Set Register | Rd | ← | 0xFF | None | 1 | 1 | 1 | 1 |
| MUL | Rd,Rr | Multiply Unsigned | R1:R0 | ← | Rd x Rr (UU) | Z,C | 2 | 2 | 2 | N/A |
| MULS | Rd,Rr | Multiply Signed | R1:R0 | ← | Rd x Rr (SS) | Z,C | 2 | 2 | 2 | N/A |
| MULSU | Rd,Rr | Multiply Signed with Unsigned | R1:R0 | ← | Rd x Rr (SU) | Z,C | 2 | 2 | 2 | N/A |
| FMUL | Rd,Rr | Fractional Multiply Unsigned | R1:R0 | ← | Rd x Rr<<1 (UU) | Z,C | 2 | 2 | 2 | N/A |
| FMULS | Rd,Rr | Fractional Multiply Signed | R1:R0 | ← | Rd x Rr<<1 (SS) | Z,C | 2 | 2 | 2 | N/A |
| FMULSU | Rd,Rr | Fractional Multiply Signed with Unsigned | R1:R0 | ← | Rd x Rr<<1 (SU) | Z,C | 2 | 2 | 2 | N/A |
| DES | K | Data Encryption | if (H = 0) then R15:R0<br>else if (H = 1) then R15:R0 | ←<br>← | Encrypt(R15:R0, K)<br>Decrypt(R15:R0, K) | | N/A | 1 / 2 | N/A | N/A |

***Figura 16.*** *Instruccions d'Operacions aritmètiques o lògiques [8].*

En segon lloc, les instruccions de canvi de flux de programa, entre les quals hi ha les instruccions de salt. En conseqüència, la majoria d'aquestes manipulen el registre *Program Counter* (PC), ja que hi carreguen l'adreça de memòria on s'haurà de continuar l'execució si es compleix la condició de canvi de flux.

| Mnemonic | Operands | Description | Operation | Flags | #Clocks AVRe | #Clocks AVRxm | #Clocks AVRxt | #Clocks AVRrc |
|---|---|---|---|---|---|---|---|---|
| RJMP | k | Relative Jump | PC ← PC + k + 1 | None | 2 | 2 | 2 | 2 |
| IJMP | | Indirect Jump to (Z) | PC(15:0) ← Z<br>PC(21:16) ← 0 | None | 2 | 2 | 2 | 2 |
| EIJMP | | Extended Indirect Jump to (Z) | PC(15:0) ← Z<br>PC(21:16) ← EIND | None | 2 | 2 | 2 | N/A |
| JMP | k | Jump | PC ← k | None | 3 | 3 | 3 | N/A |
| RCALL | k | Relative Call Subroutine | PC ← PC + k + 1 | None | 3 / 4(1) | 2 / 3(1) | 2 / 3 | 3 |
| ICALL | | Indirect Call to (Z) | PC(15:0) ← Z<br>PC(21:16) ← 0 | None | 3 / 4(1) | 2 / 3(1) | 2 / 3 | 3 |
| EICALL | | Extended Indirect Call to (Z) | PC(15:0) ← Z<br>PC(21:16) ← EIND | None | 4(1) | 3(1) | 3 | N/A |
| CALL | k | Call Subroutine | PC ← k | None | 4 / 5(1) | 3/ 4(1) | 3 /4 | N/A |
| RET | | Subroutine Return | PC ← STACK | None | 4 / 5(1) | 4 / 5(1) | 4 / 5 | 6 |
| RETI | | Interrupt Return | PC ← STACK | I | 4 / 5(1) | 4 / 5(1) | 4 / 5 | 6 |
| CPSE | Rd,Rr | Compare, skip if Equal | if (Rd == Rr) PC ← PC + 2 or 3 | None | 1 / 2 / 3 | 1 / 2 / 3 | 1 / 2 / 3 | 1 / 2 |
| CP | Rd,Rr | Compare | Rd - Rr | Z,C,N,V,S,H | 1 | 1 | 1 | 1 |
| CPC | Rd,Rr | Compare with Carry | Rd - Rr - C | Z,C,N,V,S,H | 1 | 1 | 1 | 1 |
| CPI | Rd,K | Compare with Immediate | Rd - K | Z,C,N,V,S,H | 1 | 1 | 1 | 1 |
| SBRC | Rr, b | Skip if Bit in Register Cleared | if (Rr(b) == 0) PC ← PC + 2 or 3 | None | 1 / 2 / 3 | 1 / 2 / 3 | 1 / 2 / 3 | 1 / 2 |
| SBRS | Rr, b | Skip if Bit in Register Set | if (Rr(b) == 1) PC ← PC + 2 or 3 | None | 1 / 2 / 3 | 1 / 2 / 3 | 1 / 2 / 3 | 1 / 2 |
| SBIC | A, b | Skip if Bit in I/O Register Cleared | if (I/O(A,b) == 0) PC ← PC + 2 or 3 | None | 1 / 2 / 3 | 2 / 3 / 4 | 1 / 2 / 3 | 1 / 2 |
| **Mnemonic** | **Operands** | **Description** | **Operation** | **Flags** | **#Clocks AVRe** | **#Clocks AVRxm** | **#Clocks AVRxt** | **#Clocks AVRrc** |
| SBIS | A, b | Skip if Bit in I/O Register Set | If (I/O(A,b) == 1) PC ← PC + 2 or 3 | None | 1 / 2 / 3 | 2 / 3 / 4 | 1 / 2 / 3 | 1 / 2 |
| BRBS | s, k | Branch if Status Flag Set | if (SREG(s) == 1) then PC ← PC + k + 1 | None | 1 / 2 | 1 / 2 | 1 / 2 | 1 / 2 |
| BRBC | s, k | Branch if Status Flag Cleared | if (SREG(s) == 0) then PC ← PC + k + 1 | None | 1 / 2 | 1 / 2 | 1 / 2 | 1 / 2 |
| BREQ | k | Branch if Equal | if (Z == 1) then PC ← PC + k + 1 | None | 1 / 2 | 1 / 2 | 1 / 2 | 1 / 2 |
| BRNE | k | Branch if Not Equal | if (Z == 0) then PC ← PC + k + 1 | None | 1 / 2 | 1 / 2 | 1 / 2 | 1 / 2 |
| BRCS | k | Branch if Carry Set | if (C == 1) then PC ← PC + k + 1 | None | 1 / 2 | 1 / 2 | 1 / 2 | 1 / 2 |
| BRCC | k | Branch if Carry Cleared | if (C == 0) then PC ← PC + k + 1 | None | 1 / 2 | 1 / 2 | 1 / 2 | 1 / 2 |
| BRSH | k | Branch if Same or Higher | if (C == 0) then PC ← PC + k + 1 | None | 1 / 2 | 1 / 2 | 1 / 2 | 1 / 2 |
| BRLO | k | Branch if Lower | if (C == 1) then PC ← PC + k + 1 | None | 1 / 2 | 1 / 2 | 1 / 2 | 1 / 2 |
| BRMI | k | Branch if Minus | if (N == 1) then PC ← PC + k + 1 | None | 1 / 2 | 1 / 2 | 1 / 2 | 1 / 2 |
| BRPL | k | Branch if Plus | if (N == 0) then PC ← PC + k + 1 | None | 1 / 2 | 1 / 2 | 1 / 2 | 1 / 2 |
| BRGE | k | Branch if Greater or Equal, Signed | if (S == 0) then PC ← PC + k + 1 | None | 1 / 2 | 1 / 2 | 1 / 2 | 1 /2 |
| BRLT | k | Branch if Less Than, Signed | if (S == 1) then PC ← PC + k + 1 | None | 1 / 2 | 1 / 2 | 1 / 2 | 1 / 2 |
| BRHS | k | Branch if Half Carry Flag Set | if (H == 1) then PC ← PC + k + 1 | None | 1 / 2 | 1 /2 | 1 / 2 | 1 / 2 |
| BRHC | k | Branch if Half Carry Flag Cleared | if (H == 0) then PC ← PC + k + 1 | None | 1 / 2 | 1 / 2 | 1 / 2 | 1 / 2 |
| BRTS | k | Branch if T Bit Set | if (T == 1) then PC ← PC + k + 1 | None | 1 / 2 | 1 / 2 | 1 / 2 | 1 / 2 |
| BRTC | k | Branch if T Bit Cleared | if (T == 0) then PC ← PC + k + 1 | None | 1 / 2 | 1 / 2 | 1 / 2 | 1 / 2 |
| BRVS | k | Branch if Overflow Flag is Set | if (V == 1) then PC ← PC + k + 1 | None | 1 / 2 | 1 / 2 | 1 / 2 | 1 / 2 |
| BRVC | k | Branch if Overflow Flag is Cleared | if (V == 0) then PC ← PC + k + 1 | None | 1 / 2 | 1 / 2 | 1 / 2 | 1 / 2 |
| BRIE | k | Branch if Interrupt Enabled | if (I == 1) then PC ← PC + k + 1 | None | 1 / 2 | 1 / 2 | 1 / 2 | 1 / 2 |
| BRID | k | Branch if Interrupt Disabled | if (I == 0) then PC ← PC + k + 1 | None | 1 / 2 | 1 / 2 | 1 / 2 | 1 / 2 |

***Figura 17.*** *Instruccions de Canvi de Flux de Programa [8].*

En tercer lloc, hi ha les instruccions de transferència de dades, les quals permeten el moviment, la càrrega i descàrrega dels registres.

En quart lloc, es troben les instruccions de bit i bit-test, gràcies a les quals, el programador pot treballar a nivell de bit. Com ara les instruccions SEI i CLI, les quals serveixen per a gestionar les interrupcions, habilitant-les o deshabilitant-les respectivament.

Finalment, hi ha les instruccions de control de la MCU. Mitjançant les quals, per exemple, l'usuari pot veure l'estat del *Watchdog* o la gestió de l'energia.

Si bé, es pot fer una classificació més precisa, establint funcionalitats més acotades. Per exemple, hi ha un conjunt d'instruccions destinat específicament a gestionar la pila, a través del control del Stack Pointer.

| Instruction | Stack pointer | Description |
|---|---|---|
| PUSH | Decremented by 1 | Data is pushed onto the stack |
| CALL<br>ICALL<br>RCALL | Decremented by 2 | Return address is pushed onto the stack with a subroutine call or interrupt |
| POP | Incremented by 1 | Data is popped from the stack |
| RET<br>RETI | Incremented by 2 | Return address is popped from the stack with return from subroutine or return from interrupt |

***Figura 18.*** *Instruccions per a la Gestió del Stack Pointer [6].*

### 3.7. Unitat d'Entrada i Sortida

Com s'ha comentat anteriorment, un dels tres elements bàsics d'un controlador és la unitat d'entrada i sortida, ja que és el bloc encarregat d'establir les connexions amb l'exterior.

Qualsevol tipus d'aplicació de sistema encastat no té sentit si no pot interaccionar amb el món exterior, sigui per a processar dades externes o per enviar dades a l'exterior, és a dir, sense unes entrades i sortides amb què treballar.

Com s'ha exposat anteriorment, totes les E/S i els perifèrics estan mapejats a l'espai de memòria I/O. Si bé, presenten les següents diferències amb l'espai de memòria de dades o de programa.

D'una banda, la memòria s'ajusta a la velocitat del microprocessador. En canvi, els perifèrics són més lents i poden arribar a paralitzar el procés. De l'altra, convé destacar que els dispositius d'entrada i sortida poden alterar l'execució normal del codi, ja que poden demanar accés als busos i la majoria d'ells treballen per interrupcions. Al contrari que la memòria, la qual serveix per a llegir o escriure, però quan el procés ho requereix.

Així doncs, al llarg d'aquest apartat, s'amplien tots els coneixements necessaris per a poder treballar i controlar els perifèrics.

En primer lloc, cal conèixer el *pinout* del dispositiu, ja que a través d'ell es poden veure tots els pins i se'n pot deduir la seva funcionalitat. Com s'ha mencionat anteriorment, a causa del tipus d'encapsulat emprat en el microcontrolador utilitzat, aquest disposa de 32 pins, la designació i disposició dels quals es pot observar en la següent figura. A més, seguidament, es realitza una descripció dels pins.

Evidentment, es necessita subministrar la tensió d'alimentació adequada al dispositiu, els pins que duen a terme aquesta funció estan marcats de color vermell i negre. Si bé, convé destacar que els pins AREF i AVCC estan relacionats amb el convertidor A/D. Com els pins

19 i 22 que també estan específicament destinats a aquest perifèric. Concretament, són entrades analògiques del convertidor. En conseqüència, aquest microcontrolador pot treballar amb *inputs* analògics.

La resta de pins, constitueixen els ports, és a dir, la interfície mitjançant la qual el xip es comunica amb el món exterior. Aquests estan *memory mapped*, el que significa que cada port té la seva adreça de memòria específica.

Tot i que convé destacar que la majoria dels pins comparteixen funcionalitats, el que es coneix com a *pin-sharing*. Atès que generalment els microcontroladors AVR ofereixen més perifèrics que el nombre de pins físics que tenen. Consegüentment, l'usuari ha de seleccionar quina funció li interessa més a cada moment. Per exemple, hi ha el pin de reset i els dos pins de l'oscil·lador de cristall, els quals proporcionen el pols de rellotge necessari per a la sincronització.

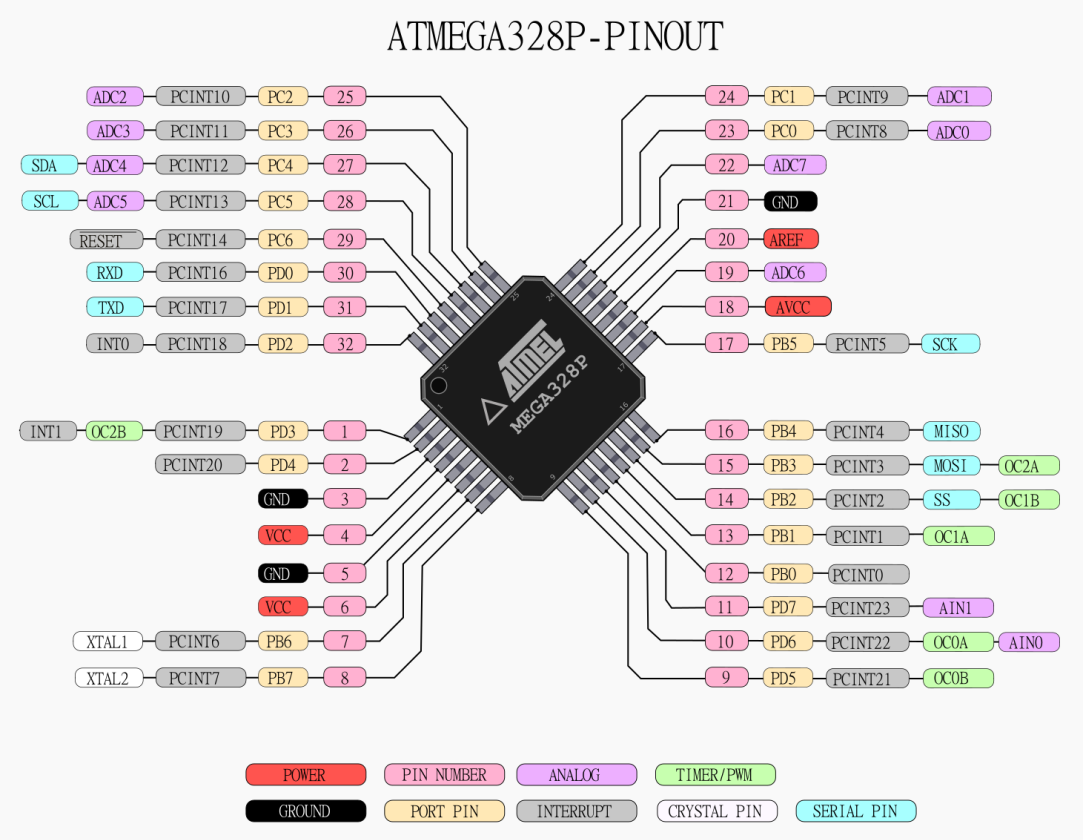



***Figura 19.*** *Pinout de l'ATmega328P.*

A fi de poder monitoritzar i controlar els perifèrics, aquests s'han de connectar als ports. Pel cas del present dispositiu, aquest disposa de tres ports digitals, anomenats: PORTB, PORTC i PORTD. Els quals s'escriuen de forma general com PORTxn, essent respectivament la lletra del port i el número del pin. En aquest sentit, els ports b i d tenen 8 bits i el c 7.

En aquest cas, cada bit s'anomena pin i aquest es pot comportar com una entrada o una sortida del sistema. Tal com s'indica en el manual del microcontrolador: *"All AVR ports have true Read-Modify-Write functionality when used as general digital I/O ports."* És a dir, es pot modificar l'estat d'un bit determinat, sense causar alteracions en els altres bits del port, utilitzant les instruccions SBI i CBI. A continuació, es detalla què implica el nivell dels pins.

Cada port consisteix de tres registres ubicats en l'espai I/O:

- **DDRx** – Registre de direcció de dades: Com el seu nom indica, serveix per a determinar la direcció del pin. És a dir, per a configurar si un pin es comporta com una entrada o com una sortida, quan s'hi escriu un 0 o un 1 respectivament.

- **PORTx** – Registre de dades: Com l'anterior, és un registre de lectura i escriptura. Si el pin està configurat com un *output*, aquest s'utilitza per a activar els bits desitjats.
  Encara que cada pin té prou energia per a accionar un LED, evidentment, no és capaç de poder controlar un motor directament. Cal fer-ho a través d'un altre component, com ara un transistor activat per una sortida. Tenint en compte que, com s'indica en la taula 30-1 del manual, a condicions normals, el corrent de sortida és de 20 mA.

- **PINx** – *Port Input Pins*: Aquest registre només és de lectura, ja que s'utilitza per a llegir l'estat dels pins, és a dir, per a emmagatzemar el valor del pin.

Com es pot observar en la següent figura, cada pin té una resistència pull-up interna i díodes de protecció, tant a terra com a $V_{cc}$.

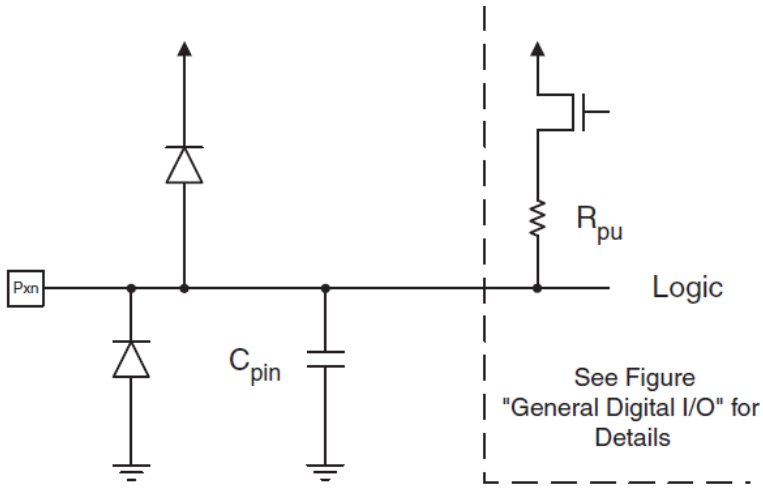



***Figura 20.*** *Esquemàtic d'un Pin d'Entrada i Sortida [6].*

Atès que hi ha diversos modes d'operació, cal veure les diferents opcions de configuració dels pins. D'una banda, per a activar la resistència $R_{pu}$ cal configurar el pin com a input i cal escriure un 1 al PORTxn.

De l'altra, per a desactivar aquesta resistència, es pot fer de dues maneres, configurant el pin com una sortida o posant a 0 el PORTxn.

A més, com s'ha comentat, el registre PINx només és de lectura. No obstant això, tal com s'indica en el *datasheet*: "*Writing a logic one to PINxn toggles the value of PORTxn, independent on the value of DDRxn. Note that the SBI instruction can be used to toggle one single bit in a port.*"

Finalment, hi ha un altre registre relacionat amb la configuració dels ports, el registre *MCU Control Register*, concretament el bit 4. Mitjançant la seva activació, es desactiva la funció de pull-up per a tots els pins de tots els ports. Encara que s'utilitzi la configuració anteriorment esmentada.

### 3.8. Gestió de les Interrupcions

Les interrupcions són un recurs essencial en els sistemes encastats, ja que sovint és necessari interrompre la seqüència habitual del programa per tal d'atendre una prioritat, normalment generada per un esdeveniment extern a la CPU.

Les principals causes de l'activació d'aquest mecanisme són la realització d'un reset, l'activació dels pins d'entrada o la transferència d'informació dels perifèrics i/o de la memòria, com per exemple, l'*overflow* d'un *timer* o l'estat de l'ADC. En definitiva, les interrupcions són senyals aleatoris que poden tenir lloc en qualsevol moment i alteren el flux normal d'execució del programa.

Concretament, les tasques més habituals que duen a terme són la monitorització d'entrades i la lectura de perifèrics amb especificacions crítiques de temps. En conseqüència, no serveix el mètode *poll*, és a dir, enquesta, ja que aquest es basa en consultar periòdicament si el valor de l'entrada ha canviat. Si el pols és molt curt, pot ser que el microcontrolador perdi el senyal perquè en aquell instant està dedicat a una altra tasca. A més, no garanteix que una acció que presenta un elevat grau de criticitat, sigui atesa immediatament. Fet que pot provocar seriosos problemes de seguretat.

Consegüentment, cal emprar les interrupcions, les quals són gestionades per la CPU de la següent manera:

- Quan la CPU rep el senyal de petició d'interrupció, IRQ, de la sigla en anglès d'*Interrupt Request*, acaba d'executar la instrucció en curs i detecta que hi ha una petició.

- Seguidament, comença el cicle de reconeixement de la interrupció, és a dir, s'autoritza o no l'atenció a través del senyal *interrupt_acknowledge*. Si bé, abans de cridar a la ISR (*Interrupt Service Routine*), cal salvar el context, l'estat del comptador de programa i del registre d'estat, a la pila.

- Un cop s'ha obtingut l'adreça d'inici de la ISR a través del vector d'interrupcions, s'executa la rutina de servei associada. Si bé, primer, per tal d'assegurar el correcte funcionament, cal inhabilitar la resta d'interrupcions i guardar a la pila els registres que es modifiquen durant aquesta rutina.

- Finalment, es retorna al programa principal, recuperant les dades anteriorment guardades i tornant a habilitar les interrupcions. Així, es retorna el control al *main* i se segueix amb l'execució normal.

En l'arquitectura dels dispositius AVR, hi ha un mòdul destinat específicament a la gestió de les interrupcions, anomenat unitat d'interrupcions. El qual té els seus registres de control en l'espai I/O i un bit en el Registre d'estat, anomenat *Global Interrupt Enable*.

A més, cada font d'interrupció té un vector d'interrupció en la taula de vectors d'interrupció, on estan ordenats de més a menys prioritat, ja que com més baixa és l'adreça del vector, més alta és la prioritat. En aquest cas, hi ha 26 vectors ubicats en la memòria de programa, els quals es poden consultar en la taula 12.1 del manual del microcontrolador. Concretament, aquesta taula conté l'adreça de la primera instrucció de cada rutina de servei. Com per exemple la del reset o les relacionades amb els *timers*.

Així doncs, els dispositius AVR duen a terme la gestió de les interrupcions de la següent manera. En primer lloc, cada interrupció té assignats bits d'habilitació individuals anomenats

EIMSK, els quals s'han d'escriure a 1, juntament amb el *Global Interrupt Enable* bit, per a poder habilitar la interrupció.

Seguidament, quan es produeix una interrupció, es baixa la bandera I del SREG, provocant que totes les interrupcions quedin deshabilitades. Quan es retorna de la ISR, aquest bit es torna a activar automàticament.

Si bé, si s'utilitza llenguatge assemblador, l'aplicació pot controlar l'estat del bit I mitjançant les instruccions CLI i SEI, respectivament.

Pel que fa a les interrupcions externes, és a dir, les activades pels pins INT0 i INT1 o per qualsevol dels pins PCINT [0:23], si estan habilitades, s'activaran encara que els pins estiguin configurats com a sortides, aquesta característica permet generar interrupcions per software.

Finalment, encara hi ha dos registres més associats a la gestió de les interrupcions. Per un costat, el registre EIFR compost de dos bits, els quals actuen com a *flag* de cada font d'interrupció. De l'altre, com que les interrupcions es poden activar per un flanc ascendent o descendent o per un nivell baix, cal indicar quin dels mètodes s'utilitza mitjançant el registre EICRA.

### 3.9. ATmega328P Xplained Mini

Com s'ha comentat anteriorment, durant la realització del present projecte, s'utilitza el kit ATmega328P Xplained Mini, ja que aquest proporciona una plataforma de desenvolupament que permet l'avaluació i la posterior programació del microcontrolador ATmega328P.

En primer lloc, és important recordar que per a poder treballar correctament amb aquest tipus de dispositius, cal consultar sovint el seu manual[6]. És per això, que al llarg d'aquest apartat i durant les pràctiques, se'n farà especial referència.

Així doncs, la principal característica del kit és que inclou un depurador, anomenat *Mini Embedded Debugger* (mEDBG), que proporciona una integració perfecta amb el software Microchip Studio, el qual es detalla més endavant. Aquesta combinació, permet depurar i programar el microcontrolador sense la necessitat de hardware extern. En concret, el mEDBG està incorporat en l'altre xip de la placa, el qual també forma part de la família AVR.

Si bé, per a poder programar el dispositiu a través de la interfície *debugWIRE*, cal que els dos xips del kit estiguin sincronitzats. Així, el mEDBG proporciona un senyal de rellotge a l'ATmega328P. Concretament, com que es treballa amb un voltatge de 5 V, la freqüència de rellotge és de 16 MHz.

A fi d'interactuar amb el món exterior i rebre l'alimentació necessària, 5 V, aquesta plataforma està equipada amb un connector USB, compost per un port virtual COM, el qual permet establir una comunicació sèrie amb el PC.

A més, el Xplained Mini disposa dels següents elements hardware per a establir una comunicació amb l'usuari. D'una banda, té dos LED, un de color verd, que mostra l'estat del mEDBG, i un de color groc anomenat *User LED*, el qual està connectat al portb5 i es pot

[6] https://ww1.microchip.com/downloads/en/DeviceDoc/ATmega328P-Xplained-Mini-UG-DS50002659B.pdf Consultat 15/04/2021

utilitzar segons les necessitats de l'aplicació. De l'altra, també hi ha un polsador connectat al portb7.

Un altre tret a destacar, és que aquesta placa d'avaluació permet, mitjançant els connectors soldats sobre aquesta, un accés fàcil i còmode a tots els pins del microcontrolador. En conseqüència, facilita un disseny personalitzat, ja que s'hi poden connectar sensors o actuadors directament o utilitzar *shield borads* personalitzades segons els requeriments de cada solució.

En definitiva, és una plataforma molt fàcil d'utilitzar, ja que només cal connectar el kit al PC a través d'un cable mini USB i, automàticament, el software de desenvolupament ja reconeix quin model de placa s'ha connectat i el LED d'estat parpelleja per tal d'indicar que està alimentat correctament. I encara presenta un altre avantatge, es pot adquirir a un cost considerablement baix.

A continuació, es mostra una imatge del Xplained mini amb els principals components senyalitzats. De color taronja els dos microcontroladors, el mEDBG i l'ATmega328P, i de color verd els elements perifèrics que permeten la interconnexió entre la placa i l'exterior.

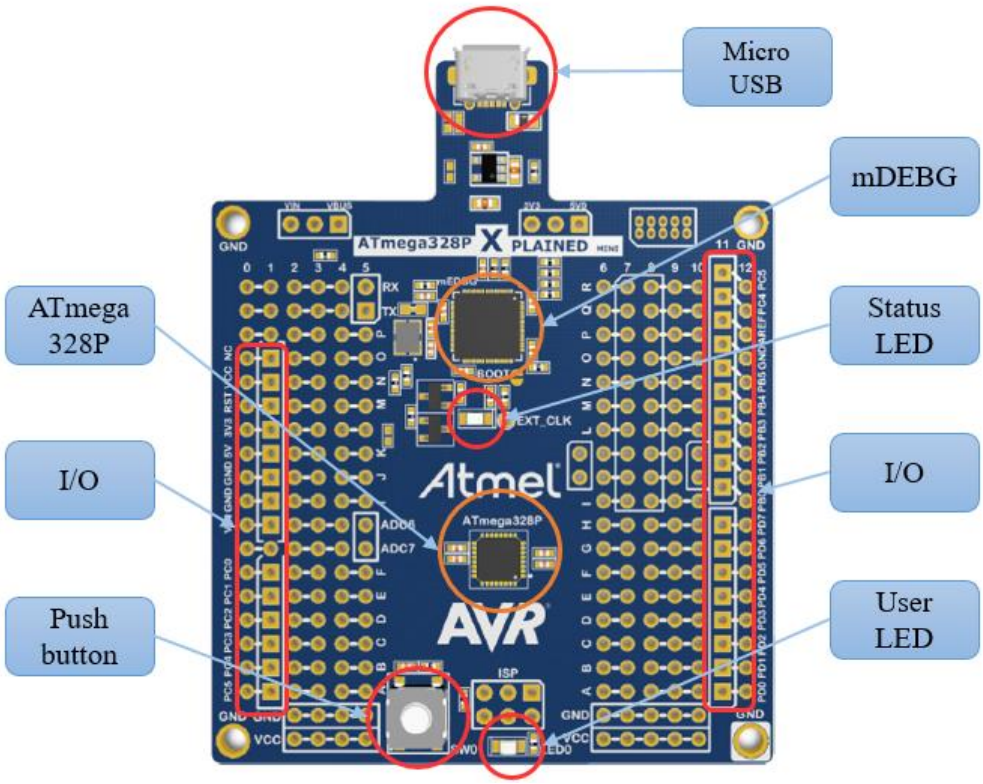



***Figura 21.*** *Components Principals del Xplained mini.*

### 3.10. Shield Board

Les plaques d'expansió, més conegudes pel seu nom en anglès, *shield boards*, són plaques que es connecten a sobre de la PCB principal, a través dels ports d'entrada i sortida, per a ampliar les seves funcionalitats. El seu principal objectiu és facilitar la creació de prototips, especialment en entorns acadèmics o en aplicacions dissenyades per aficionats.

En aquest cas, s'utilitza el mòdul Arduino ProtoShield – Bare PCB[7] del fabricant Sparkfun. El qual proporciona una àrea de soldadura, en la qual es poden soldar tots els components necessaris per a cada projecte, i fàcil accés als pins del microcontrolador ATmega328P, a través dels connectors de tipus *header* soldats en els laterals de la placa.

A més, també inclou alguns *footprints* ja definits, com ara dos LED amb les seves respectives resistències, i algunes pistes. Gràcies a les quals es facilita el procés de muntatge i connexió dels diversos components, ja que proporcionen més pins de GND i Vcc i algunes connexions entre els *footprints* esmentats.

Si bé, aquest mòdul està basat en el *pinout* de l'Arduino R3. Per aquest motiu, algunes de les designacions dels components indicades en la capa *silkscreen* no es corresponen amb la nomenclatura emprada en l'ATmega328P.

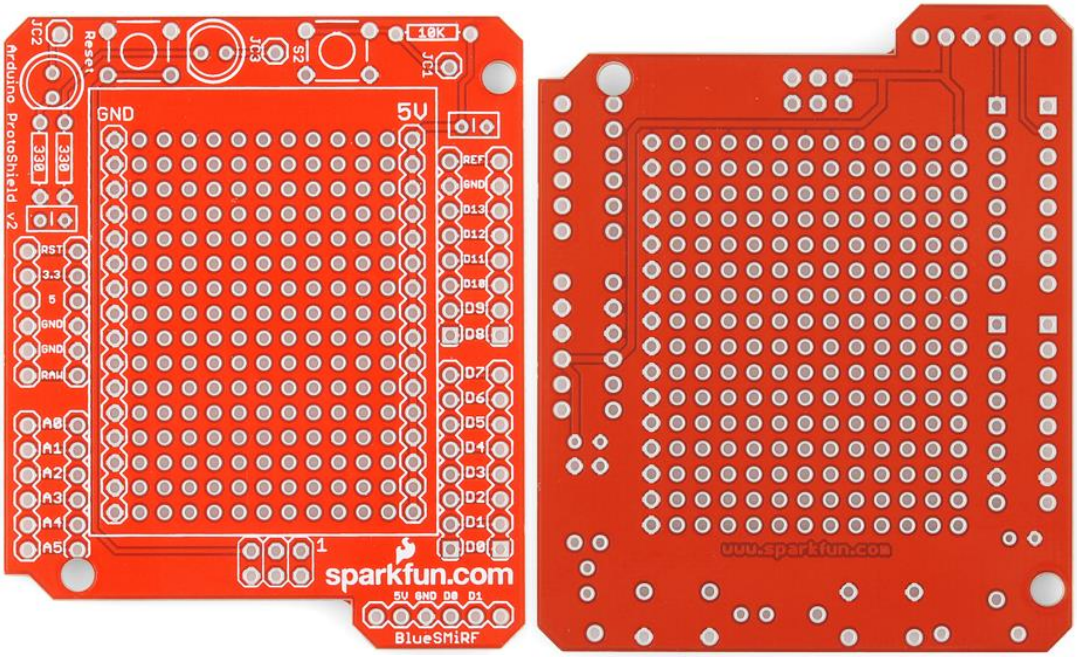



***Figura 22.*** *Vista Top i Bottom de la Shield Board utilitzada [9].*

Així doncs, el primer pas per a dissenyar el prototip és consultar la documentació de la placa[8]. En aquest document esquemàtic, es poden observar les connexions entre els *pads* de la PCB.

En segon lloc, cal definir els components que cal soldar-hi. En aquest cas, cal instal·lar-hi quatre LED amb les seves respectives resistències de 150 Ω. En concret, tres díodes simulen els colors d'un semàfor i el quart s'utilitza per a indicar, de manera visual, si la placa està correctament alimentada, és a dir, si està en estat on; un polsador amb la seva resistència *pull-up* i un condensador de 100 nF entre l'alimentació i terra i la interfície que permet connectar els senyals necessaris de la pantalla LCD amb el microcontrolador. En concret, s'ha utilitzat un connector IDC-16 mascle, que permet la connexió del cable pla del *display*.

[7] https://www.sparkfun.com/products/retired/11665?_ga=2.89517661.798630531.1629966232-1311065752.1627207735 Consultat 20/06/2021
[8] https://www.sparkfun.com/datasheets/DevTools/Arduino/ProtoShield-v25.pdf Consultat 20/06/2021

Seguidament, cal realitzar el circuit esquemàtic de les noves connexions del mòdul. A més, amb la finalitat d'assegurar la correcta ubicació i facilitar la tasca de soldadura, es realitza el següent disseny gràfic amb els components que cal soldar sobre la placa de prototip.

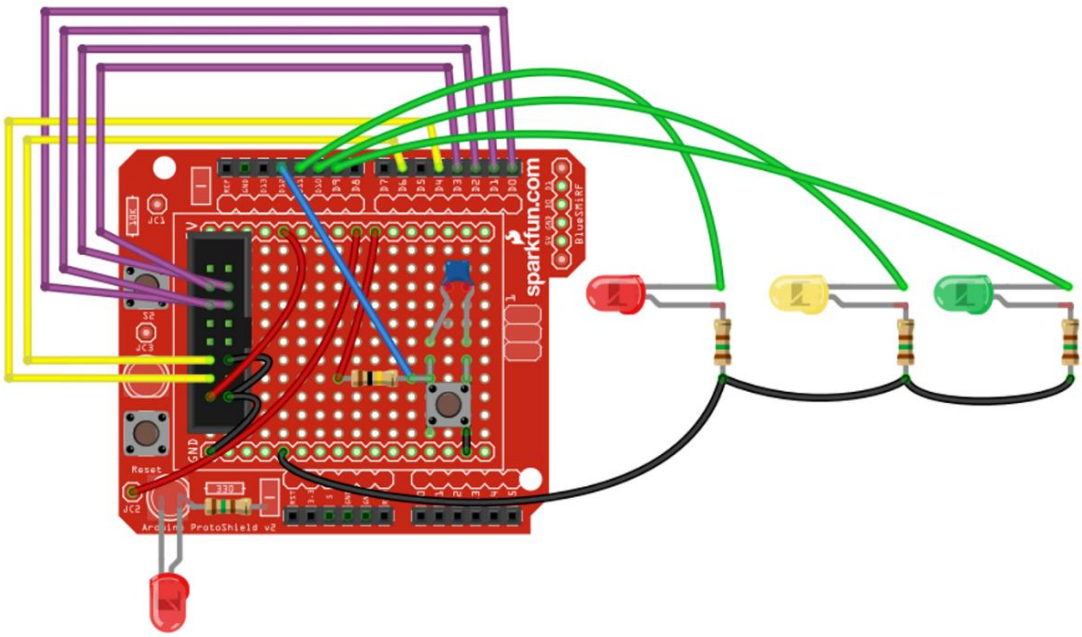



***Figura 23.*** *Simulació Gràfica del Mòdul Extern [10].*

En definitiva, tota la documentació relativa al mòdul extern configurat expressament per a les presents pràctiques, es pot trobar en l'Annex 8.1 Esquemàtic Mòdul Extern.

A continuació, s'adjunta una imatge del mòdul extern completament acabat, amb tots els components soldats.

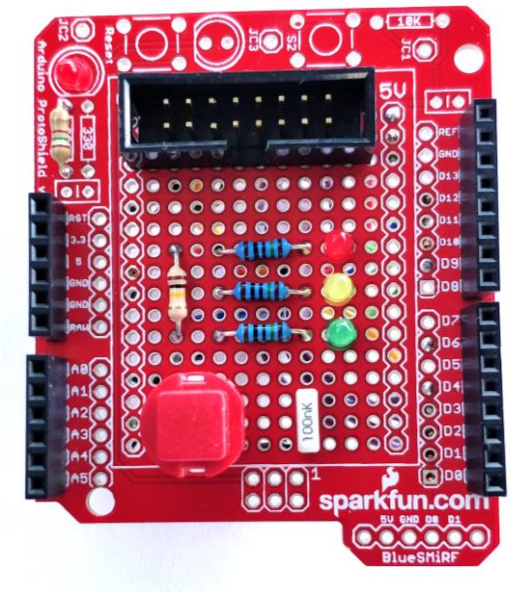



***Figura 24.*** *Mòdul Extern configurat.*

En la següent imatge, es pot observar la pantalla LCD connectada a la placa de prototip mitjançant el cable pla. En conseqüència, es poden observar tots els perifèrics connectats al microcontrolador.

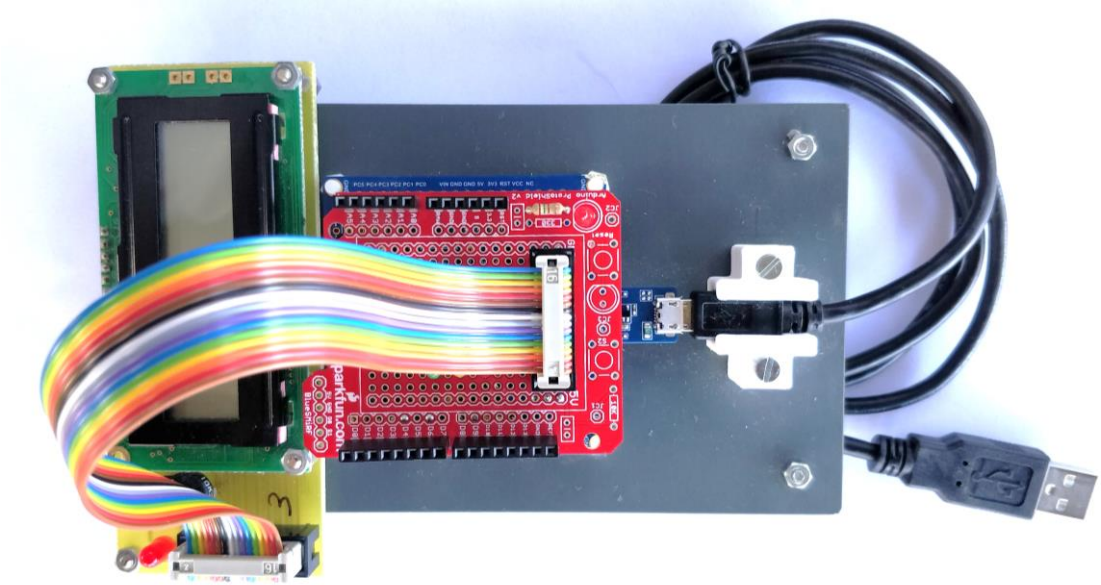


***Figura 25.*** *Mòdul Extern amb la Pantalla LCD connectada.*

### 3.11. LCD

La pantalla LCD Hitachi 44780 s'utilitza com a perifèric del microcontrolador ATmega328P. Com s'ha comentat anteriorment, la connexió entre aquests dos elements es realitza a través d'un cable pla, connectat a un connector IDC-16 mascle de la *shield board* utilitzada.

Les principals característiques, els detalls més importants sobre el seu funcionament i les funcions específicament desenvolupades per a controlar-la, es detallen a la pràctica 5 del present projecte.

# 4. Entorn de Programació

Per tal de poder dur a terme les tasques de desenvolupament del programari desitjat, el primer que cal fer és instal·lar a l'ordinador un entorn de desenvolupament integrat, generalment conegut com a IDE, de les seves sigles en anglès. S'anomena així, ja que proporciona un entorn integrat únic per a generar i depurar les aplicacions de microcontroladors, la majoria de les quals formen part d'un sistema encastat.

En aquest cas, com que s'utilitza un microcontrolador AVR, l'usuari pot escollir entre utilitzar el software Microchip Studio o el MPLAB X. Aquest últim és recomanable pel cas que anteriorment hagi treballat amb microcontroladors de la família PIC i ja estigui familiaritzat amb aquest entorn. Ara bé, cal afegir-hi una capa de versatilitat al conjunt d'eines per a poder treballar amb dispositius AVR. No obstant això, durant la realització d'aquest projecte i les pràctiques que posteriorment realitzaran els alumnes, s'utilitza el primer, el qual es detalla a continuació.

## 4.1. Microchip Studio

Primerament, convé destacar que el software Microchip Studio és en realitat el conegut Atmel Studio. A causa de la compra d'Atmel per part de Microchip i amb la voluntat d'unificar els noms dels productes, s'ha realitzat aquest canvi de nom.

El software Microchip Studio és un entorn de desenvolupament integrat senzill, intuïtiu i fàcil d'utilitzar per a escriure, desenvolupar, depurar i compilar les aplicacions creades pels microcontroladors d'AVR, escrites tant en C, C++, com en codi assemblador. A més, cal destacar que proporciona una connexió perfecta amb els kits de desenvolupament i avaluació dels dispositius d'aquesta família. Fet remarcable, ja que en aquest cas, s'empra un kit d'avaluació. En conseqüència, Microchip proporciona totes les eines necessàries per a crear una aplicació de microcontrolador des de zero.

Particularment, tal com s'indica en la guia d'usuari, les característiques a ressaltar d'aquest software són:

- És compatible amb més de 500 dispositius de la família AVR i SAM, suportant totes les eines d'aquests.
- Gràcies al compilador integrat, l'usuari pot escriure i depurar codi en llenguatge C, C++ i assemblador.
- Disposa d'un simulador, el qual permet obtenir un model precís de la CPU, interrupcions, perifèrics i estímuls externs.
  Pel cas del present projecte, és una eina molt útil. Atès que durant la realització de les pràctiques, l'alumne pot simular el codi generat abans d'anar al laboratori i així optimitzar el temps i poder detectar els possibles errors comesos. És per això, que en la part pràctica es detalla el funcionament d'aquesta funció.
- Disposa d'un editor de codi integrat, anomenat Visual Assist, el qual es detalla més endavant.
- Permet *In-system programming and debugging*, és a dir, presenta l'habilitat de poder programar el microcontrolador quan està encastat dins d'un sistema complet. En aquest cas, integrat en el kit de desenvolupament Xplained mini.
- Segons les necessitats de cada usuari, aquest pot ampliar l'entorn de desenvolupament gràcies a la *Microchip Gallery*, la botiga d'aplicacions i extensions en línia.

Algunes de les extensions més útils i populars són l'*Atmel Start* i el *Data Visualizer*. La primera és una eina de configuració que ajuda en la selecció i configuració dels components més adequats per a cada aplicació. Un cop seleccionats, proporciona exemples de codi per a facilitar l'inici del projecte.
En canvi, la segona permet processar i visualitzar les dades i analitzar el consum d'energia de l'aplicació.

Aquests trets faciliten la tasca del programador, ja que permeten desenvolupar i depurar el codi de manera més senzilla, ràpida i eficient, en una interfície gràfica intuïtiva i fàcil d'utilitzar.

Pel que fa al Visual Assist, no només és un editor de text. Sinó que també permet redissenyar, llegir, navegar i, evidentment, escriure codi en C o C++. És a dir, és una eina que permet augmentar la productivitat i l'eficiència a l'hora de treballar amb el codi de l'aplicació. És per això, que és recomanable utilitzar al màxim les seves funcions, ja que faciliten notablement la tasca d'escriure el codi i la seva posterior modificació o consulta. A més, també permeten que el codi quedi més net i entenedor.

Les funcions més remarcables són:

- *Funció de suggeriment i acrònims:* Molt útil, ja que permet estalviar temps i evitar possibles errors en l'escriptura.
- *Comandes ràpides:* També permeten estalviar temps. Es pot trobar una taula amb la funció de cadascuna d'elles a la guia d'usuari anteriorment citada. Per exemple, si se selecciona una funció del *main* i es premen les tecles "ALT + G" a la vegada, l'assistent et porta a la declaració de la funció corresponent.
- *Navegar a través del codi:* La comanda anterior és un exemple de navegació pel codi.

Així doncs, aquest software es pot obtenir de manera gratuïta accedint a la pàgina web del fabricant Microchip[9] i descarregant l'executable adequat. Si bé, cal tenir en compte que només és compatible en entorns Windows.

### 4.2. Compilador

Juntament amb l'entorn integrat de desenvolupament, es disposa del compilador MPLAB XC. En concret, com que pel present projecte es treballa amb el microcontrolador ATmega328P de 8 bits, s'instal·la la versió de compilador XC8, la qual admet tots els MCU de 8 bits de les famílies AVR i PIC. Consegüentment, es disposa de tot l'entorn necessari integrat per a poder dissenyar qualsevol solució.

El Microchip Studio també inclou l'AVR GNU Toolchain. Consegüentment, quan es crea un projecte, l'usuari pot seleccionar el compilador desitjat. O bé el MPLAB XC o el compilador AVR GCC, el qual era l'emprat en l'entorn de desenvolupament Atmel Studio. Essent aquesta la principal diferència entre els dos softwares.

Si bé, en termes generals, els dos compiladors tenen el mateix objectiu principal, el qual es detalla a continuació.

L'objectiu principal del compilador és la generació de codi executable, de tipus .hex, a partir dels arxius de codi font, escrits en llenguatge d'alt nivell, en aquest cas de tipus .c. Durant

[9] https://www.microchip.com/en-us/development-tools-tools-and-software/microchip-studio-for-avr-and-sam-devices#Downloads Consultat 20/04/2021

el procés, el compilador realitza una optimització del codi, augmentant-ne l'eficiència. A més, també presenta una altra funcionalitat coneguda com a *linker*, la qual combina tots els mòduls necessaris per a generar el programa executable. En la següent figura, es pot observar un diagrama de flux del procés principal del compilador.

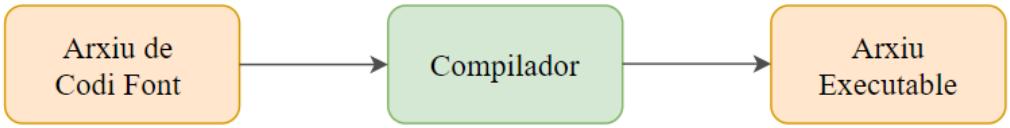



***Figura 26.*** *Diagrama de Flux bàsic del Compilador.*

Si bé, la seqüència de compilació és més extensa, ja que inclou diverses aplicacions internes i arxius temporals relacionats entre ells i, generalment, més d'un arxiu font.

En la Figura 27, es descriu el procés detallat de compilació. El requadre de color gris representa el compilador i els requadres petits del seu interior les diverses aplicacions necessàries. També es mostren els fitxers temporals, els quals es representen en el moment de la seqüència que es generen.

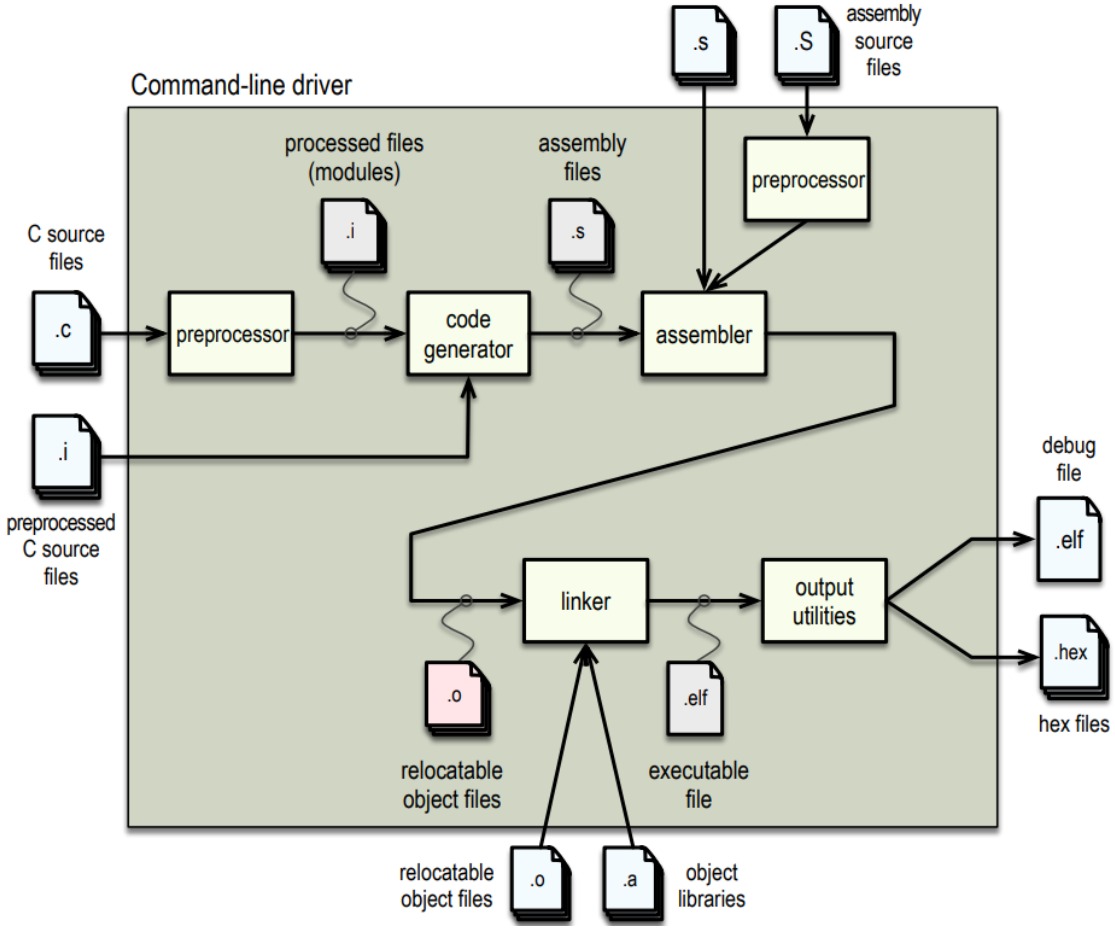



***Figura 27.*** *Seqüència del Procés de compilació [11].*

Així doncs, un cop l'usuari ha escrit els fitxers de codi font, el compilador és l'encarregat de generar els fitxers de tipus .o. Concretament, els arxius de sortida del preprocessador, els arxius de tipus .i, anomenats mòduls, són el resultat de la combinació dels *source files* i les llibreries, de tipus .h, especificades per la directiva #include en el codi. Com es pot observar

en la cantonada superior dreta, aquesta descripció també es pot aplicar pels arxius de codi font escrits en assemblador (.S). Als quals també se'ls pot incloure llibreries mitjançant el preprocessador.

Arribats en aquest punt, és el torn del *linker*, el qual crea el fitxer executable en codi màquina .elf i el mapa de memòria.

Finalment, com a resultat s'obté l'arxiu .hex, el qual es carrega en la memòria flash del microcontrolador per a poder ser executat.

### 4.3. Programació en Assemblador i C

Actualment, la majoria de solucions de sistemes encastats estan programades en codi C, és a dir, en llenguatge d'alt nivell. Si bé, al principi del desenvolupament de les solucions *embedded*, tot el codi s'escrivia en llenguatge assemblador, ja que no hi havia cap més alternativa.

Així doncs, avui en dia, quan s'inicia el desenvolupament de projectes amb microcontroladors, cal decidir quin llenguatge s'utilitza, si assemblador o C.

En un primer moment, pot semblar evident la resposta: llenguatge d'alt nivell, gràcies al fet que és més *user friendly*, senzill i còmode.

A més, com que els processadors actuals són més potents i disposen de més memòria, generalment, no hi ha limitacions crítiques de memòria. En conseqüència, no cal exercir un control molt estricte ni precís sobre la gestió d'aquests recursos. Un altre factor a considerar, és l'augment de la complexitat de les aplicacions, fet que resulta, gairebé sempre, en la inviabilitat de programar-les en llenguatge assemblador.

No obstant això, pels dissenyadors de sistemes *embedded*, és necessari tenir coneixements i competències en la generació de codi en assemblador. A continuació, s'esmenten els principals motius d'aquesta afirmació.

D'una banda, és una eina didàctica, ja que per a poder escriure en assemblador, l'usuari ha de conèixer amb més detall l'arquitectura i el joc d'instruccions del dispositiu emprat. Així, pot comprendre com el codi interactua amb el hardware.

De l'altra, durant l'estudi de la viabilitat del projecte, l'eficiència del codi generat és un dels punts claus a considerar. Malgrat que els compiladors actuals solen fer una gran tasca d'optimització de codi i són força fiables, existeix la possibilitat que els arxius generats no siguin del tot correctes o continguin errors. És per això, que és recomanable que el programador revisi el resultat obtingut.

Com també, en alguns casos concrets, resulta impossible implementar determinades funcionalitats en C, com ara *drivers* de perifèrics, rutines altament optimitzades o quan és necessari treballar amb els registres de la CPU.

En definitiva, les aplicacions actuals de sistemes encastats contenen gran part del codi escrit en C i integren parts desenvolupades en assemblador. Consegüentment, les pràctiques proposades en el present treball de fi de grau estan dissenyades seguint aquest criteri. A fi que l'alumne primer es familiaritzi amb el microcontrolador i, posteriorment, dissenyi solucions més semblants a les professionals.

Finalment, independentment del llenguatge seleccionat, cal tenir present que cal seguir una metodologia per a poder obtenir un bon programa. En primer lloc, és recomanable seguir un disseny *top-down* i realitzar un esquema o un diagrama de flux de la solució que es vol implementar.

En segon lloc, cal tenir present que un programa està format per l'algorisme i per l'estructura de dades. Pel cas d'utilitzar llenguatge assemblador, aquest no proporciona una estructura ni un control de tipus de dades, és el programador qui ha de satisfer correctament aquestes necessitats.

# 5. Pràctiques

## 5.1. Introducció

En aquest apartat s'han agrupat les pràctiques dissenyades en el present projecte de fi de grau, essent l'objectiu principal d'aquest. Com s'ha comentat anteriorment, cal utilitzar el microcontrolador ATmega328P com a dispositiu principal.

Les pràctiques s'han dissenyat amb la finalitat que l'alumne vagi adquirint progressivament els coneixements impartits en les classes teòriques de l'assignatura. No obstant això, també s'ha inclòs la informació teòrica i tècnica complementària per tal de poder seguir els laboratoris satisfactòriament.

Així doncs, el format de les pràctiques és el següent. D'una banda, primer, hi ha els objectius i els enunciats de l'estudi previ i del treball al laboratori. Essent la part del document que caldrà entregar a l'alumne. De l'altra, hi ha la solució esperada, en la qual s'ha detallat la resposta que s'espera obtenir dels enunciats previs.

Cal destacar també la importància que els estudiants es familiaritzin en la consulta de manuals i documents tècnics, ja que són un recurs imprescindible per a poder treballar correctament amb el microcontrolador i els seus perifèrics. Atès que contenen tota la informació i detalls sobre la seva arquitectura i com cal programar-los. En finalitzar les pràctiques, els estudiants tindran l'habilitat de consultar de manera eficaç els *datasheets*.

A fi d'obtenir una corba d'aprenentatge suau i proporcionar les eines per a consolidar els coneixements clau en el món dels sistemes *embedded*, s'han organitzat les pràctiques en 3 blocs. En primer lloc, durant la pràctica 0, es realitza una introducció al treball al laboratori. Presentant els dispositius i l'entorn de desenvolupament emprat.

En segon lloc, cal realitzar un seguit de pràctiques en llenguatge assemblador. D'aquesta manera, l'alumne posa en pràctica els coneixements sobre l'arquitectura i el joc d'instruccions del microcontrolador, ja que la programació a baix nivell requereix una major comprensió del dispositiu.

En tercer lloc, es realitza la transició al desenvolupament d'aplicacions en llenguatge C, essent el més emprat en solucions de sistemes encastats.

Finalment, com que sempre és recomanable realitzar un diagrama de flux per tal de plasmar la solució que compleixi amb tots els requisits, es demana a l'alumne comprendre i, posteriorment, desenvolupar aquesta eina de disseny.

### 5.2. Software utilitzat

Inicialment, es va plantejar el desenvolupament de les pràctiques utilitzant l'entorn de desenvolupament Microchip Studio. Atès que, actualment, és el software que proporciona el fabricant del microcontrolador, Microchip.

Si bé, durant les primeres preses de contacte, es va detectar que el compilador integrat no funcionava correctament, impedint una correcta simulació i emulació dels projectes. Per exemple, no es podien fixar *breakpoints* per tal de comprovar l'execució del codi o l'execució de bucles de retard tardava un temps excessiu.

En conseqüència, s'ha decidit dissenyar les pràctiques utilitzant el software Atmel Studio, ja que és el que s'ha utilitzat fins al moment per a treballar amb els microcontroladors de la família AVR.

A més, gràcies a la gran quantitat d'aficionats i professionals que han utilitzat anys aquesta eina informàtica al llarg dels anys, s'han corregit una gran quantitat d'errors, aconseguint un entorn de desenvolupament molt fiable i potent.

### 5.3.Pràctica 0: Introducció a les Pràctiques

UNIVERSITAT ROVIRA i VIRGILI


# PRÀCTICA 0:
# INTRODUCCIÓ A LES PRÀCTIQUES

**Assignatura:** Microcontroladors i Sistemes *Embedded*

**Ensenyament:** Enginyeria de Sistemes i Serveis de Telecomunicacions

**Curs:** 2020/21

## 1. Introducció i Objectius

Al llarg dels laboratoris de l'assignatura de Microcontroladors i Sistemes *Embedded* es treballa amb el kit d'avaluació ATmega328P Xplained Mini, el qual porta integrat el microcontrolador de 8 bits ATmega328P. Ambdós dispositius pertanyen al fabricant Microchip. A continuació, es realitza una breu introducció de les seves especificacions i característiques.

A més, per a poder realitzar les pràctiques amb aquest kit, s'ha d'utilitzar l'entorn de desenvolupament Atmel Studio, el qual es pot obtenir de forma gratuïta a la web de Microchip.

Així doncs, els objectius d'aquesta pràctica introductòria són:

- Breu explicació sobre el funcionament de les pràctiques i punts a tenir en compte.
- Breu descripció dels dispositius utilitzats al llarg dels laboratoris.
- Instal·lació del software Atmel Studio.
- Familiarització amb l'entorn de desenvolupament.
- Creació d'un projecte nou.

## 2. Metodologia i Informació rellevant

El treball al laboratori de l'assignatura de Microcontroladors i Sistemes *Embedded* es fonamenta a partir de dos objectius principals. El primer és que l'alumne ha d'adquirir progressivament els coneixements que es treballen en les sessions teòriques de l'assignatura. En conseqüència, durant les pràctiques, es van introduint, gradualment, aquests conceptes i els diferents elements que conformen el kit de desenvolupament.

La segona finalitat és que l'alumne assoleixi les competències bàsiques per a poder treballar amb qualsevol microcontrolador, sigui en les assignatures posteriors o en el món laboral. Per aquest motiu, és essencial que es familiaritzi amb la cerca dels manuals adequats i sàpiga buscar la informació necessària en cadascun d'ells, ja que, normalment, aquest tipus de documentació sol ser bastant extensa.

D'altra banda, a fi de desenvolupar les solucions necessàries per a poder assolir les tasques del laboratori, s'utilitzen els següents elements hardware i software.

- El microcontrolador ATmega328P.
- El qual està integrat en la placa de desenvolupament Xplained mini.
- Finalment, es necessita un entorn de desenvolupament integrat (IDE), en aquest cas s'utilitza l'Atmel Studio.

Així doncs, els documents que sempre cal tenir a mà i que s'hauran de consultar recurrentment per tal d'obtenir la informació necessària per a realitzar les pràctiques són:

- El manual del microcontrolador: ATmega48A/PA/88A/PA/168A/PA/328/P *Data Sheet.*
- El manual de la Xplained mini: ATmega328P Xplained Mini
- El manual del joc d'instruccions: AVR *Instruction Set Manual*
- El manual de l'IDE: Atmel Studio *User Guide*
- El manual del simulador de l'IDE: AVR *Simulator*

Finalment, convé ressaltar que s'utilitzen dos llenguatges de programació. En les primeres pràctiques, l'alumne ha d'escriure el codi en llenguatge assemblador. Posteriorment, el llenguatge utilitzat és el C.

Aquesta transició és deguda al fet que actualment la major part dels projectes de sistemes *embedded* es realitzen en C. Si bé, és cert que per a poder implementar determinades funcionalitats o quan es requereix un elevat grau de control i precisió, s'utilitza el llenguatge assemblador.

A més, la programació en baix nivell requereix uns coneixements més elevats de l'arquitectura i del joc d'instruccions del microcontrolador emprat. En conseqüència, amb la realització de les primeres pràctiques, l'alumne assolirà les competències bàsiques de l'ATmega328P i aprendrà com interactuar correctament amb aquest. Per a posteriorment, desenvolupar aplicacions més complexes utilitzant les habilitats adquirides en llenguatge C.

## 3. Dispositius utilitzats

A continuació, es realitza una breu introducció dels dispositius utilitzats. No obstant això, l'alumne necessitarà més informació per a poder superar les pràctiques. És en aquest moment, quan haurà de consultar de forma adequada els manuals corresponents.

### 3.1. ATmega328P

El microcontrolador utilitzat és l'ATmega328P, el qual forma part de la família Mega AVR. En conseqüència, és un xip amb un bus de dades de 8 bits, basat en l'arquitectura Harvard i amb un joc d'instruccions de tipus RISC.

Actualment, és un dels microcontroladors més utilitzats en el mercat de sistemes *embedded* gràcies al fet que presenta unes característiques que el fan idoni per aquest tipus d'aplicacions:

- Com que té un joc d'instruccions RISC, l'usuari té a la seva disposició 131 potents instruccions en llenguatge assemblador.
- Pel que fa a la gestió de la memòria, la qual es detalla més endavant, pel fet que és de tipus Harvard, té dos espais de memòria principal. La memòria de dades SRAM i la de programa de tipus Flash.
- L'usuari també té a la seva disposició 32 registres de propòsit general i 23 línies d'entrada i sortida d'ús general.
- Admet un rang de tensió de funcionament més ampli que la majoria de xips, d'1,8 V a 5,5 V. Si bé, normalment s'utilitza la tensió de 5 V com a estàndard.
- Disposa de sis *sleep modes* diferents.
- Disposa de diverses funcions perifèriques, com ara *timers*, una unitat de gestió d'interrupcions, canals de PWM, un ADC, entre altres. Algunes de les quals es treballen en les següents pràctiques i l'alumne s'haurà de familiaritzar amb el seu funcionament i propietats.

Al llarg de la realització de les pràctiques, s'aprofundeix en els components necessaris per a poder superar amb èxit els enunciats. Així, al final de l'assignatura, l'alumne estarà familiaritzat amb els principals components i funcionalitats del microcontrolador i haurà adquirit els coneixements necessaris per a treballar amb qualsevol MCU.

### 3.2. Xplained mini

A fi de poder treballar amb el microcontrolador i dissenyar els projectes necessaris, s'utilitza el kit d'avaluació ATmega328P Xplained Mini, ja que aquest proporciona una plataforma de desenvolupament que permet l'avaluació i la posterior programació del microcontrolador ATmega328P.

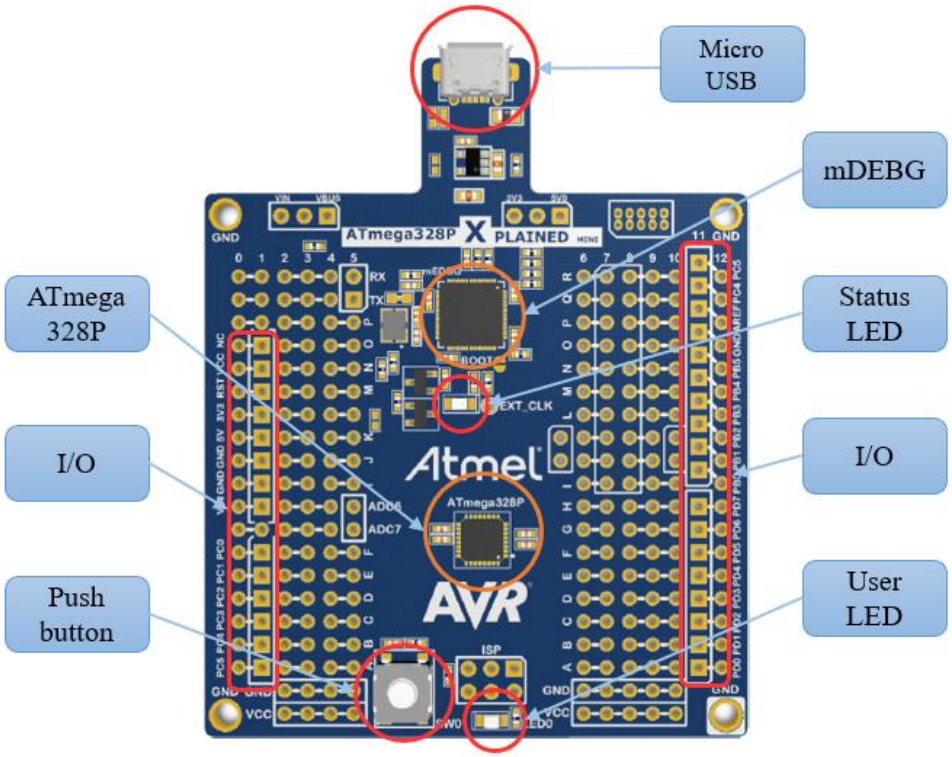



***Figura 1***. *Components Principals del Xplained mini.*

En la figura anterior, es poden observar els principals components del Xplained mini. De color taronja l'ATmega328P i un segon microcontrolador, l'ATmega32U4, el qual s'encarrega de la gestió de les tasques de depuració en placa. De color verd, els elements perifèrics que permeten la interconnexió entre la placa i l'exterior.

La principal característica del kit és que inclou un depurador, anomenat *Mini Embedded Debugger* (mEDBG) en el microcontrolador ATmega32U4, també de la família AVR. El qual proporciona una integració perfecta amb el software Microchip Studio. Aquesta combinació, permet depurar i programar el microcontrolador sense la necessitat de hardware extern.

Atès que per a poder depurar *on-chip*, és a dir, en la placa, és necessari aquest segon microcontrolador, ja que és l'encarregat d'interrompre l'execució de l'ATmega32P i proporcionar a l'usuari la interfície necessària, juntament amb el software de desenvolupament, per a consultar els valors i l'estat dels registres. Per exemple, a través de la finestra de visualització dels pins d'entrada i sortida

Si bé, per a poder programar el dispositiu a través de la interfície *debugWIRE*, cal que els dos xips del kit estiguin sincronitzats. Així, el mEDBG proporciona un senyal de rellotge a l'ATmega328P. Concretament, com que es treballa amb un voltatge de 5 V, la freqüència de rellotge és de 16 MHz.

A fi d'interactuar amb el món exterior i rebre l'alimentació necessària, 5 V, aquesta plataforma està equipada amb un connector USB, compost per un port virtual COM, el qual permet establir una comunicació sèrie amb el PC.

Un altre tret a destacar, és que aquesta placa d'avaluació permet, mitjançant els connectors soldats sobre aquesta, un accés fàcil i còmode a tots els pins del microcontrolador. En conseqüència, facilita un disseny personalitzat, ja que s'hi poden connectar sensors o actuadors directament o utilitzar *shield borads* personalitzades segons els requeriments de cada solució.

En definitiva, és una plataforma molt fàcil d'utilitzar, ja que només cal connectar el kit al PC a través d'un cable mini USB i, automàticament, el software de desenvolupament ja reconeix quin model de placa s'ha connectat i el LED d'estat parpelleja per tal d'indicar que està correctament alimentat.

### 3.3. Hardware extern

A cada pràctica, es detallen els components necessaris, ja formin part de la mateixa placa de desenvolupament o siguin hardware extern, com ara un display de 7 segments.

### 3.4. Software

A fi de generar i depurar les aplicacions necessàries a cada pràctica, es necessita un entorn integrat de desenvolupament. En aquest cas, s'utilitza el software Atmel Studio, amb el qual l'alumne s'hi ha de familiaritzar progressivament.

Així, a continuació, es detalla el procés d'instal·lació i de creació d'un projecte nou. També es comença a introduir la potent eina del simulador.

Encara que convé destacar que, actualment, a causa de l'adquisició d'Atmel per part de Microchip, existeix una nova versió d'aquest entorn de desenvolupament. El qual s'ha reanomenat com a Microchip Studio, però, en essència, és el mateix software.

Si bé, en les presents pràctiques s'utilitza la versió de l'Atmel Studio 7, ja que gràcies a la seva àmplia i prolongada utilització, s'han corregit tots els possibles *bugs* i garanteix una correcta execució de totes les seves funcionalitats.

## 4. Instal·lació de l'Atmel Studio

Tot i que al laboratori disposareu d'ordinadors amb tot el software necessari, és recomanable que us l'instal·leu en el vostre ordinador personal, ja que d'aquesta manera us podreu preparar els programes a casa i simular-los abans d'anar al laboratori. Fet que us permetrà poder resoldre els dubtes que us vagin sorgint i no quedar-vos estancats durant la sessió de laboratori, estalviar temps i ser més eficients.

El software és totalment gratuït, però només és compatible en sistemes operatius Windows i, evidentment, necessita una certa quantitat de memòria RAM lliure.

Així doncs, el primer que cal fer és accedir a la pàgina web del fabricant www.microchip.com. Un cop allí, navegar per la web: *Tools and Software –AVR and SAM Downloads Archive*. D'aquesta manera, us podreu començar a familiaritzar amb l'entorn

web del fabricant, fet que us serà molt útil, ja que haureu de cercar informació per a poder realitzar les pràctiques satisfactòriament.

Un cop arribats a la secció desitjada[1], cal descarregar l'executable segons la situació en aquell moment. En aquest cas, s'utilitza l'opció de realitzar la instal·lació amb connexió a internet, ja que permet escollir què es vol instal·lar. Seguidament, cal obrir l'executable i seguir els passos indicats en l'assistent d'instal·lació:

- Seleccionar l'arquitectura desitjada. Pel cas del microcontrolador que s'utilitza, cal seleccionar AVR.
- A continuació, apareix una opció d'instal·lar una extensió de Software Framework (ASF). Com que no la necessitarem, no la seleccionarem.
- Seguidament, apareix una finestra amb la validació dels requisits del sistema. Si està tot correcte, apareixerà en verd i es podrà continuar.

D'altra banda, quan hi hagi actualitzacions disponibles, a la part superior dreta de la pantalla, la icona de la bandera s'il·luminarà de color taronja:

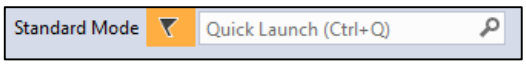



***Figura 2**. Icona indicant actualitzacions disponibles.*

Si es desitja actualitzar, només cal fer clic a la bandera i apareixerà una finestra amb totes les actualitzacions pendents. Es recomana revisar periòdicament que el software estigui actualitzat a últim nivell.

Finalment, cal destacar que aquest entorn de desenvolupament és una plataforma modular. És per això que la majoria de funcions es presenten en forma d'extensions i l'usuari pot decidir si instal·lar-les o no. No obstant això, les que s'utilitzen durant la realització d'aquestes pràctiques ja venen per defecte. Per exemple:

- Visual Assist: És l'editor de codi, permet redissenyar, llegir i navegar a través del codi escrit en C o C++. Resulta ser una eina molt útil, ja que facilita la tasca al programador.

Les altres extensions s'utilitzen en nivells més avançats o quan es requereixen funcions més específiques. Com ara la popular extensió Arduino IDE for Atmel Studio 7. La qual permet el desenvolupament de plaques i la utilització de llibreries Arduino, dins de l'entorn de Microchip.

## 5. Entorn de Desenvolupament Atmel Studio

Així doncs, un cop s'ha instal·lat satisfactòriament l'entorn, cal conèixer les seves característiques i familiaritzar-se amb ell.

[1] https://www.microchip.com/en-us/development-tools-tools-and-software/avr-and-sam-downloads-archive

### 5.1. Generació d'un nou Projecte

Per a crear un nou projecte des de zero, cal accedir a: **File** – **New** – **Project** i apareix la següent pantalla de configuració:

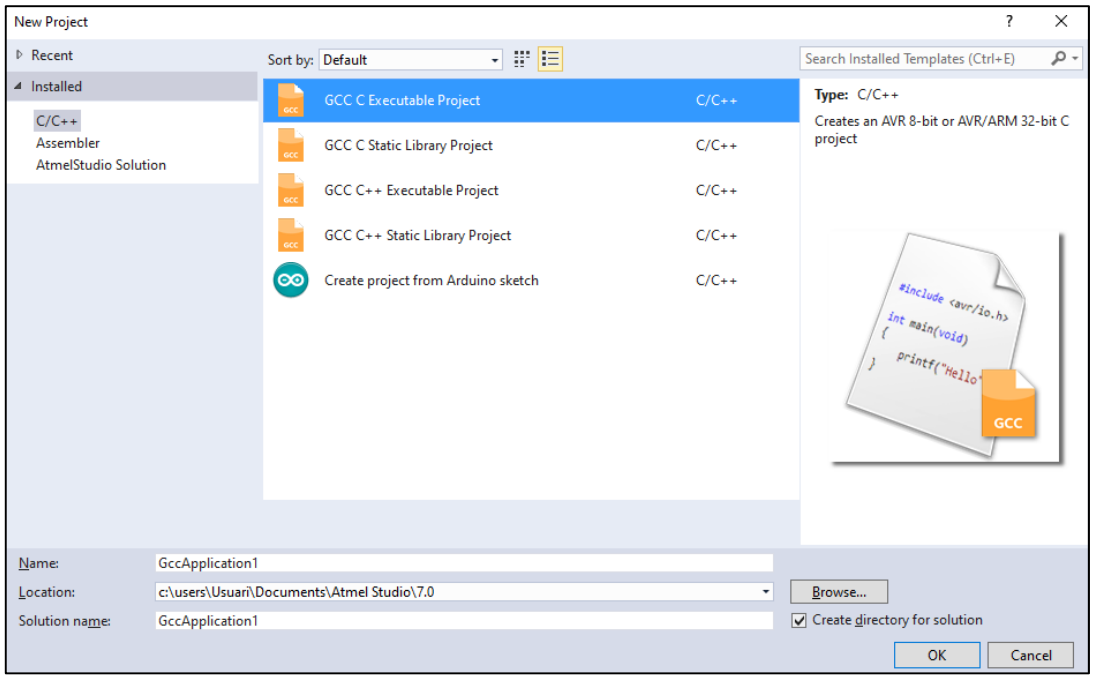



***Figura 3**. Generació d'un nou projecte amb llenguatge C.*

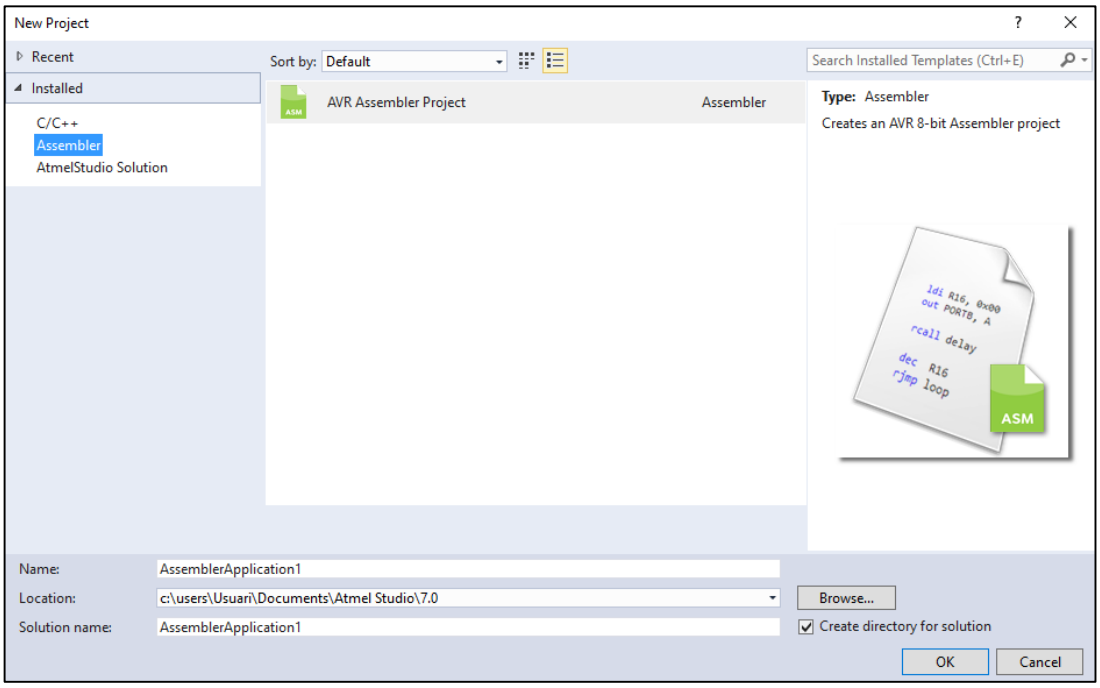



***Figura 4**.Generació d'un nou projecte amb assemblador.*

Mitjançant la qual es pot seleccionar:

- El llenguatge que s'utilitzarà: A la cantonada superior esquerra, es pot seleccionar entre C/C++ o assemblador. Després, al mig de la pantalla, apareixen les següents opcions:

  - Si se selecciona C/C++, es pot escollir entre un projecte executable, el qual és el tipus més comú, o un projecte de llibreria estàtica (amb extensió .a). El qual permet crear llibreries pre-compilades per a posteriorment enllaçar-les a altres projectes per tal de reutilitzar codi de diferents aplicacions que necessiten la mateixa funció.
  - Pel cas d'escriure el codi en assemblador, només hi ha una opció disponible.
- Name: Cal escriure-hi el nom del projecte.
- Location: Per tal de definir la ubicació on es vol guardar el projecte en qüestió. Cal destacar que per a cada projecte generat, es crearà automàticament una carpeta amb la solució, designada amb el nom prèviament introduït.

Un cop configurat tot correctament, s'ha de fer clic en el botó **OK** i apareixerà la següent pantalla, on s'ha de seleccionar el nom del dispositiu utilitzat en el projecte. En aquest cas, ATmega328P.

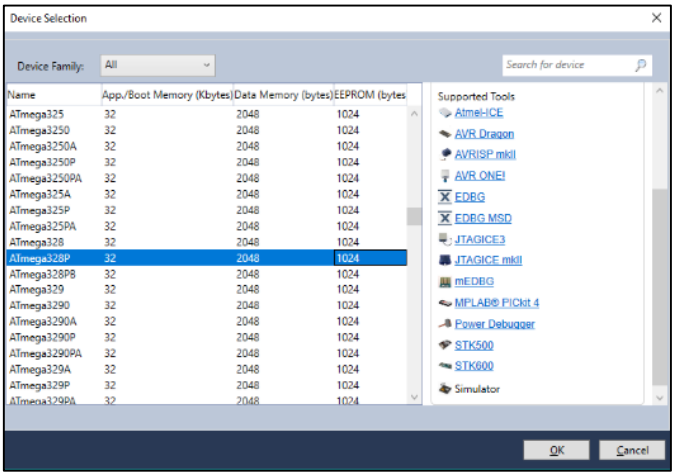



***Figura 5**. Selecció del dispositiu desitjat.*

Finalment, ja s'ha creat el projecte desitjat i apareix la pantalla a través de la qual l'usuari pot començar a treballar. A més, a través de **View** – **Solution Explorer**, l'usuari pot veure detalladament el contingut del projecte.

Per a afegir un nou fitxer o llibreria cal seleccionar: **File** – **New** – **File**.

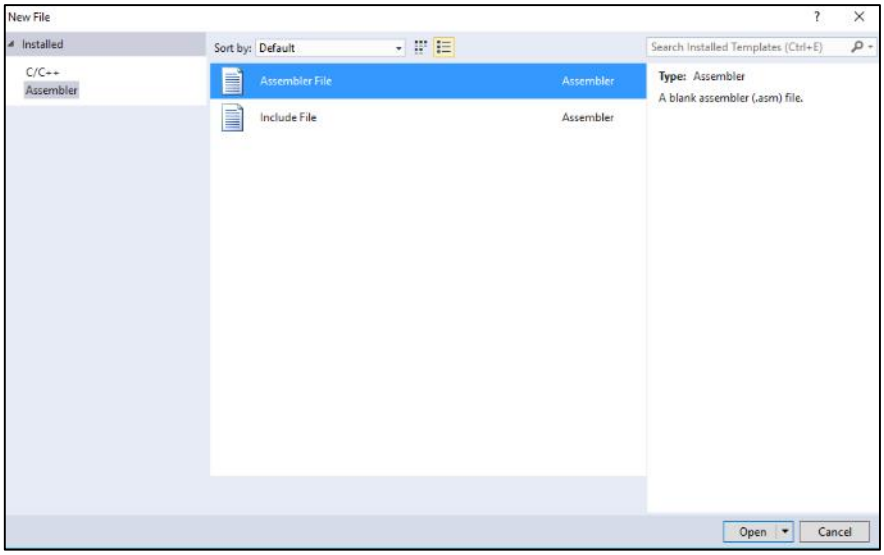



***Figura 6**. Afegir un nou fitxer o llibreria al projecte.*

Novament, es pot seleccionar el llenguatge que s'utilitza i el tipus de fitxer. En aquest cas, com que se selecciona el tipus assemblador, només apareixen dues opcions: Assembler File o Include File, per si es vol crear una llibreria, és a dir, un fitxer referenciat al projecte actual. A més, per a tenir correctament direccionada la llibreria, s'ha de guardar en el directori del sistema. Així, es podrà realitzar qualsevol crida a la llibreria sense problemes.

### 5.2. Funcionalitats del Programa

A fi de treure-li el màxim rendiment a la interfície de programació, a continuació es mostra una taula resum amb les icones de les eines principals i més útils.

| Símbol | Eina | Descripció |
|---|---|---|
| | New Project | Crea un nou projecte |
| | Open File | Obre un projecte existent |
| | Save Project | Guarda el projecte obert |
| | Start Without Debugging | Executa sense depuració |
| | Start Debugging and Break | Executa amb depuració |
| | Decrease Indent | Disminueix sangria |
| | Increase Indent | Augmenta sangria |
| | Comment out selected lines | Posa com a comentari les línies de codi |
| | Uncomment selected lines | Posa com a codi les línies de comentari |
| | Stop Debbuging | Atura l'execució amb depuració |
| | Restart | Torna a començar l'execució amb depuració |
| | Break All | Pausa l'execució amb depuració |
| | Start Debugging | Comença l'execució amb depuració |
| | QuickWatch | Estableix variables per visualitzar-ne el contingut |
| | Step Into | Para a la següent línia de codi (entra en funcions) |
| | Step Over | Igual a l'anterior (no entra en funcions) |
| | Step Out | Para a la següent línia de codi després d'una funció |
| | Run To Cursor | Executa el codi i para on es trobi el cursor |
| | Reset | Torna a començar l'execució del codi des de l'inici |
| Hex | Hexadecimal Display | Variables observades amb hexadecimal |
| | Breakpoints | Activa i desactiva punts de parada |
| | Disassembly | Observa les línies de codi en llenguatge ASM |
| | Registers | Observa el valor dels registres |
| | Memory 1 | Observa el contingut de les posicions de memòria |
| | Processor Status | Especificacions de SREG, f_clk i d'altres |
| | I/O | Observa l'estat dels registres dels pins |
| | Build Solution | Compila el codi i construeix la solució |
| | Device | Especifica el dispositiu a programar |
| | Tool | Especifica l'execució i la comunicació del codi |

***Taula 1**. Funcionalitats de les eines d'Atmel Studio.*

### 5.3. Depuració del Programa generat

Gràcies al fet d'utilitzar el kit d'avaluació, en funció de l'eina de depuració seleccionada, l'usuari pot seleccionar entre dos mètodes de depuració i posterior programació del dispositiu. D'una banda, si es vol simular el programa, cal seleccionar l'opció de simulador. De l'altra, si es vol realitzar una emulació, és a dir, que el codi s'executi sobre el hardware, cal seleccionar el depurador mEDBG.

Així doncs, només cal connectar el kit al PC a través d'un cable mini USB i automàticament, el software ja detecta quin model de placa s'està utilitzant.

Primer, cal seleccionar l'eina de depuració fent clic sobre el botó ressaltat en vermell i apareix la pantalla de la Figura 8, on cal seleccionar el depurador desitjat.

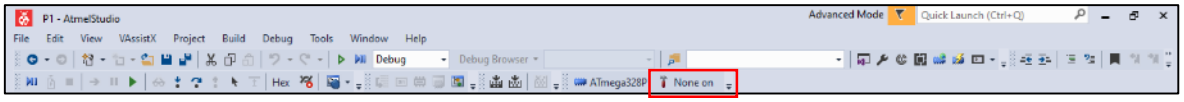



***Figura 7**. Configuració de la depuració.*

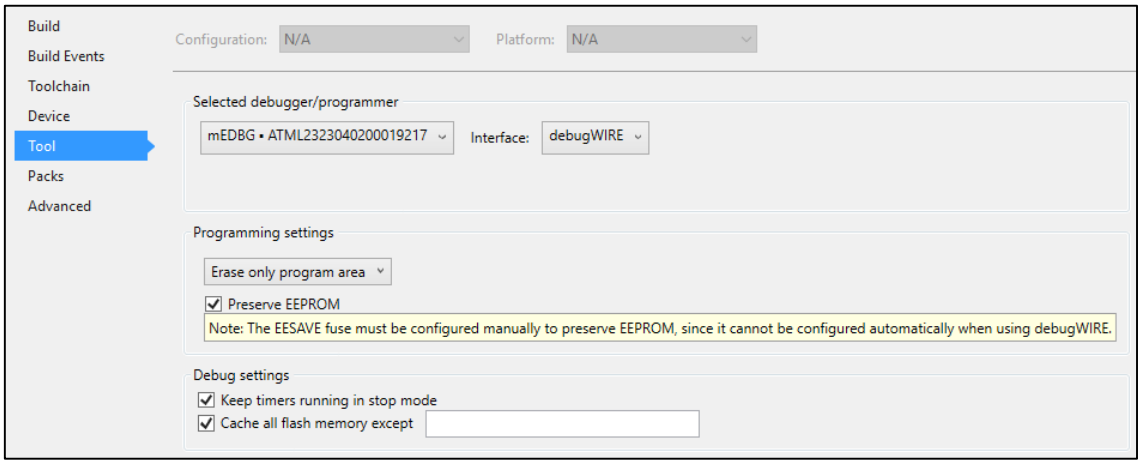



***Figura 8**. Configuració de Depuració sobre el Dispositiu (mEDBG).*

Pel cas de dur a terme l'emulació del programa, cal seleccionar com a interfície de comunicació **debugWire** i com a paràmetres de programació: ***Erase only program area***.

### 5.4. Simulador

Com s'ha mencionat anteriorment, el software Atmel Studio porta incorporat un simulador, el qual és molt útil per a identificar i corregir errors en el codi sense la necessitat de tenir un microcontrolador físic, ja que aquest simula la CPU, juntament amb totes les instruccions, les interrupcions i la majoria de mòduls d'entrada i sortida del xip.

En conseqüència, durant la realització de les pràctiques, especialment en els estudis previs, es recomana utilitzar aquesta potent eina. Atès que permet utilitzar les funcions de depuració normal, com ara **Run**, **Break**, **Reset** i fixar **Breakpoints**. Així com visualitzar els canvis en les variables, registres, memòria i I/O.

| **TIP** | Simulador – *AVR Simulator User Guide* |
|---|---|
|  | *"The performance of the simulator model is slow compared to a real device, but because it is made of software it gives the user some extra debugging possibilities that are not available with a real device."*<br>Com per exemple, es poden fixar un nombre il·limitat de *breakpoints*, els quals es poden editar mentre el programa s'executa. |

### 5.5. Llibreries en l'Atmel Studio

Independentment del llenguatge emprat, la utilització de llibreries en la programació de dispositius és una pràctica molt estesa, ja que faciliten l'escriptura de codi i el desenvolupament de la solució.

Concretament, les llibreries són trossos de codi que duen a terme una funcionalitat ben definida i permeten generar un codi més modular i clar.

Es poden diferenciar dos tipus de llibreries, les creades específicament per l'usuari o les ja creades, aquestes últimes ofereixen l'avantatge addicional que són més fiables, ja que han estat utilitzades i testejades en moltes més aplicacions. Per a utilitzar-les en projectes escrits en C, només cal incloure la seva capçalera en l'inici del programa principal:
`#include <nom_arxiu.h>` i, si s'utilitzen funcions, les línies de crida corresponents.

En aquest cas, el software Atmel Studio proporciona diverses llibreries integrades en codi C per als dispositius de 8 bits de la família AVR. Les quals es poden trobar en la següent adreça:

"C:\Program Files (x86)\ Atmel\Studio\7.0\toolchain\avr8\avr8-gnu-toolchain\avr\include"

Finalment, cal tindre en compte que durant la realització de les pràctiques, segons el tipus de llenguatge emprat, cal incloure les següents línies de codi a l'inici del programa principal.

D'una banda, per les primeres pràctiques escrites en assemblador, cal incloure les directives:

```
.include "m328pdef.inc";        Inclou el fitxer m328pdef
.device atmega328p;             Especifica el mmicrocontrolador que s'utilitza
```

De l'altra, per a les aplicacions desenvolupades en C, cal escriure:

```
#include <avr/io.h>;            Inclou la llibreria avr/io
```

En definitiva, gràcies als fitxers inclosos, es declaren les definicions apropiades dels ports I/O del dispositiu. Així, per exemple, es pot utilitzar la nomenclatura PORTB0 per a referir-se al bit 0 del portb.

### 5.4. Pràctica 1: Funció Delay

UNIVERSITAT ROVIRA i VIRGILI


# PRÀCTICA 1:
# FUNCIÓ DELAY

**Assignatura:** Microcontroladors i Sistemes Embedded

**Ensenyament:** Enginyeria de Sistemes i Serveis de Telecomunicacions

**Curs:** 2020/21

## 1. Introducció i Objectius

En primer lloc, cal tindre en compte que per a poder realitzar satisfactòriament aquesta pràctica, l'alumne primer ha de llegir i comprendre la pràctica introductòria. En la qual s'exposen els principals components utilitzats durant la realització de les pràctiques i les principals característiques del software Atmel Studio. Concretament, es detalla com instal·lar el software, com generar un nou projecte i les principals funcionalitats del programa.

Els objectius d'aquesta pràctica són:

- Iniciació en la consulta dels *datasheets* dels dispositius utilitzats. En aquest cas, l'alumne cal que es familiaritzi amb els registres i el mapa de memòria del microcontrolador.
- Familiarització amb el joc d'instruccions dels dispositius AVR.
- Generació i simulació del primer projecte.

## 2. Estudi Previ

1. En primer lloc, cal llegir amb atenció la Pràctica 0 per a poder instal·lar el software de desenvolupament Atmel Studio i generar un nou projecte, en aquest cas en assemblador.

2. Per a poder dur a terme la present pràctica, cal cercar la següent informació al manual de l'ATmega328P:
    a. Banc de registres d'ús generals.
    b. Registre d'estat SREG.
    c. Mapa de memòria del microcontrolador.

3. A fi de començar-se a familiaritzar amb el joc d'instruccions dels dispositius AVR, cal obrir el fitxer adjunt P1_Delay.asm, comentar cada línia del codi i explicar l'objectiu principal del programa.

| **ATENCIÓ** | Freqüència de treball de l'ATmega328P |
|---|---|
|  | Tal com s'indica en el manual de l'ATmega328P: "*The device is shipped with internal RC oscillator at 8.0 MHz and with the fuse CKDIV8 programmed, resulting in 1.0 MHz system clock".* És a dir, el microcontrolador té un rellotge intern de 8 MHz, però la freqüència de treball per defecte és d'1 MHz, ja que el divisor de freqüència per 8 està activat.<br><br>A més, tal com s'indica en el codi del fitxer, per a desenvolupar el bucle de retard, s'ha considerat una freqüència de treball d'1 MHz. Atès que a major freqüència de treball, major és el consum energètic. Per aquest motiu, com que la majoria d'aplicacions de sistemes encastats presenten restriccions energètiques, sempre que no hi hagi limitacions de velocitat, s'acostuma a treballar a freqüències baixes. |

| **TIP** | Llenguatge assemblador |
|---|---|

La directiva `.org` seguida d'un número determina a quines posicions de memòria es guarden les següents instruccions.
Normalment, els programes es comencen a la posició 0. És per aquest motiu, que a l'inici de qualsevol programa en assemblador cal escriure:

```
.org 0x0000
```

| **TIP** | Joc d'Instruccions AVR |
|---|---|

- Per a moure dades entre els registres de propòsit general (r0 al r31), cal utilitzar la instrucció `mov`.
- Per a moure informació entre els registres I/O, cal utilitzar les instruccions `in` o `out`.
- Només els registres del r16 al r31 accepten instruccions amb mode d'adreçament immediat, com ara `ldi`.

4. Dibuixar el diagrama de flux de l'anterior algorisme.

| **ATENCIÓ** | Ús de Registres |
|---|---|

En els programes escrits en llenguatge assemblador, cal prestar especial atenció als registres que s'utilitzen, ja que són un recurs limitat. En conseqüència, el programador ha de ser conscient de quins registres s'utilitzen per a no sobreescriure-hi i perdre les dades.
En concret, pel cas del bucle que genera el retard, s'utilitza el registre 16.

5. Comptar el nombre de cicles que tarda a executar-se l'anterior algorisme.

## 3. Treball al Laboratori

1. Iniciar el software Atmel Studio i crear un nou projecte de llenguatge assemblador.

2. Copiar l'algorisme subministrat P1_Delay i compilar el projecte per a comprovar que tot estigui correcte.

| **TIP** | Simulador |
|---|---|
|  | Per a afegir el número de les línies en l'editor de codi, eina molt útil durant la programació i la posterior depuració, cal navegar: **Tools – Options – Text Editor – All Lenguages** i seleccionar **Line Numbers**. |

3. Simular el codi i comprovar quant de temps es tarda a executar el bucle delay.

En primer lloc, cal seleccionar l'eina de depuració utilitzada, en aquest cas, el simulador.

Existeixen diverses formes de depurar el programa, cal seleccionar la més adequada per a cada situació. A més, es poden posar punts de ruptura en les instruccions que més convingui. En aquest cas, per a poder visualitzar quant tarda a executar la rutina delay, cal posar un breakpoint al començament de la rutina i l'altre al final.

Un cop iniciada la simulació, prement la icona de ***Start Debugging and Break***,  , per a mostrar els resultats, cal prémer **Debug** – **Windows** i seleccionar **Processor Status** i **I/O**. Gràcies a la informació que mostren aquestes finestres, es poden comprovar els canvis i resultats del programa. A més, en la finestra **Processor Status,** es pot comprovar que la freqüència sigui la correcta, en aquest cas 1 MHz, i es pot reiniciar el rellotge.

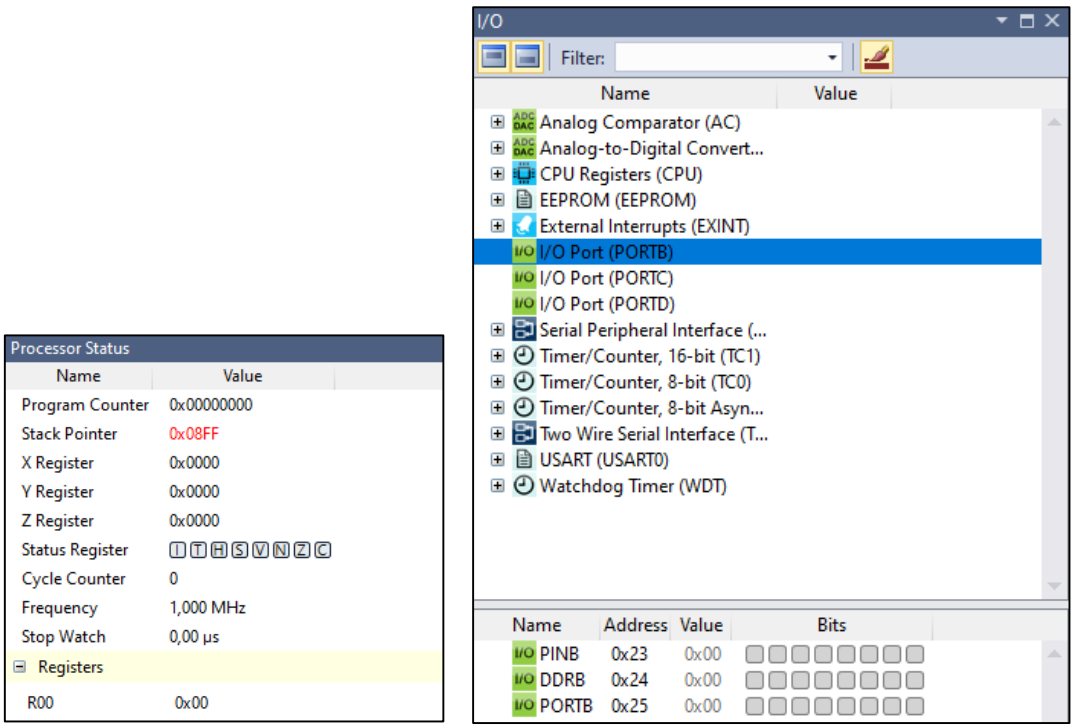



***Figura 1**. Finestres del Simulador: Processor Status i I/O.*

# 4. Solució Esperada

## 4.1. Estudi Previ

2.

Tal com es pot observar en la Figura 2, el banc de registres generals conté 32 registres de propòsit general de 8 bits, els quals són assignats als primers 32 bytes de la memòria de dades.

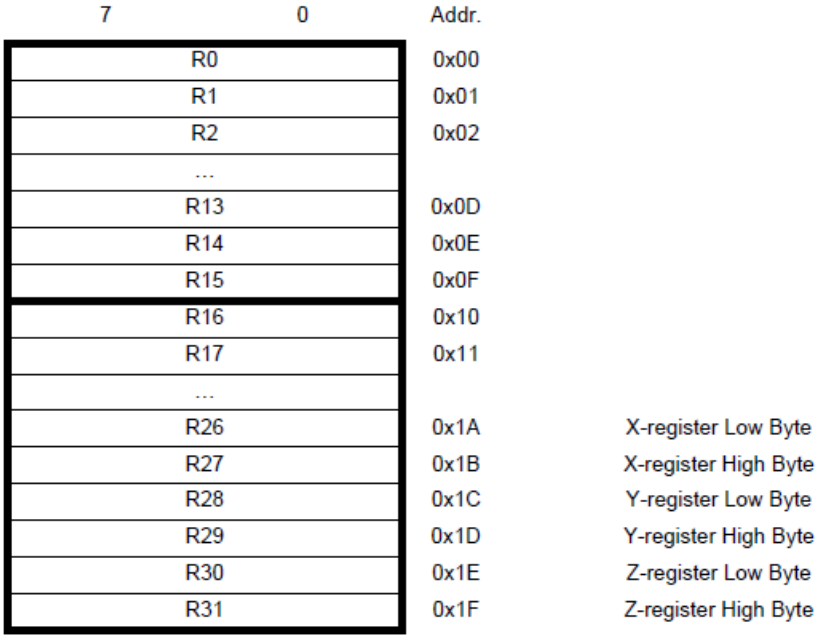



***Figura 2**. Estructura dels Registres de Propòsit General.*

També hi ha el registre d'estat, generalment anomenat SREG, el qual està format per 8 *flags*, és a dir, per vuit bits, de les quals se'n detalla la funcionalitat en les següents taules.

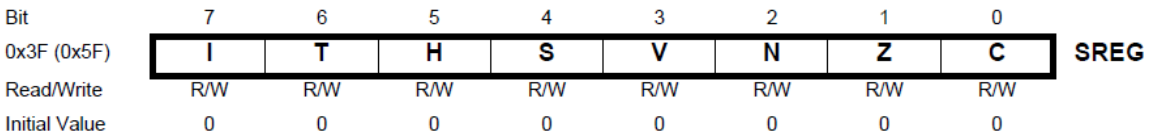



***Figura 3**. Detall dels Bits del Registre d'Estat.*

| Flag | Nom |
|---|---|
| I | Global Interrupt Enable |
| T | Bit Copy Storage |
| H | Half Carry |
| S | Sign Bit |
| V | Overflow |
| N | Negative |
| Z | Zero |
| C | Carry |

***Taula 1**. Funcionalitat dels Bits del Registre d'Estat.*

Finalment, pel que fa al mapa de memòria, com que els dispositius de la família AVR s'han desenvolupat a partir d'una arquitectura Harvard, tenen dos espais de memòria independents. D'una banda, hi ha la memòria de programa amb 32 kbytes de memòria Flash. De l'altra, la memòria de dades amb 2 kB de memòria de dades disponible.

A més, la seva arquitectura és de 8 bits, és a dir, treballa amb una ample de bus de 8 bits. Per tant, la CPU només pot treballar amb aquest nombre de bits de dades a la vegada. En el cas d'haver de processar dades més grans, aquestes s'han de dividir en fragments de 8 bits.

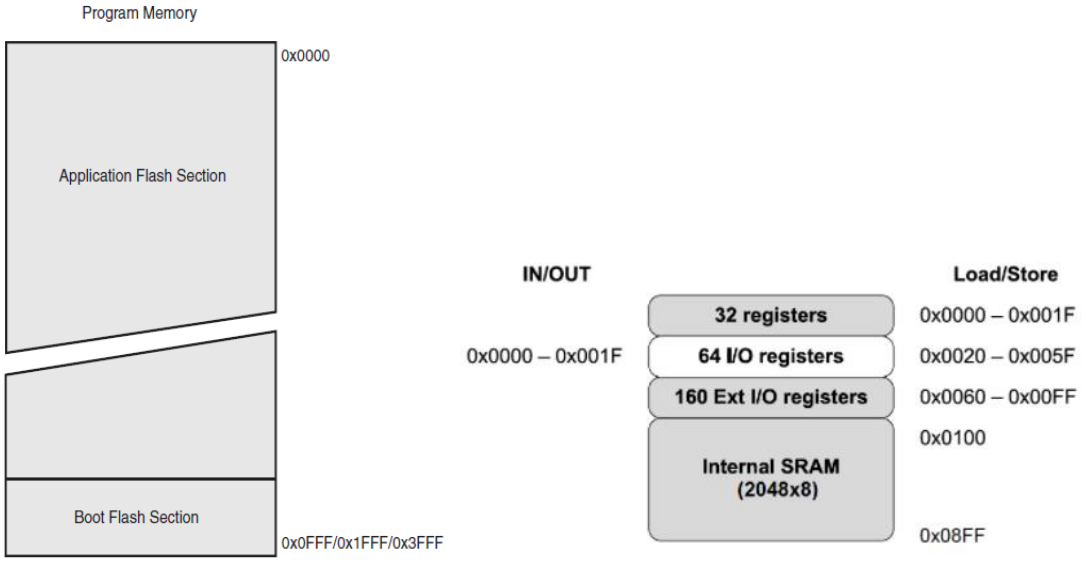



***Figura 4**. Mapa de la Memòria de l'ATmega328P.*

En la figura anterior, es pot observar com estan organitzats els dos espais de memòria.

3.

Gràcies a consultar el manual del joc d'instruccions dels dispositius AVR, es pot deduir la funcionalitat de cada línia de codi. A més, l'objectiu final del present algorisme és generar un bucle de retard.

```
.org 0x0000                    ;El programa comença a la posició 0x0000

ldi r16, 99                    ;Carrega el valor 99 al registre r16
loop:                          ;Etiqueta
nop                            ;Bucle amb instruccions que no fan res
      nop                      ;Serveixen per a esperar
      nop
      nop
      nop
      nop
      nop
      dec r16                  ;Resta 1 del valor que hi ha en r16
      brne loop                ;Salt condicional, salta si no és igual a 0
```

4.

Com es pot observar en la següent figura, on es mostra el diagrama de flux de l'anterior algorisme, per a generar el bucle de retard, primer cal carregar el valor del comptador. En aquest cas, es carrega el valor 99 al registre 16, essent aquest l'únic registre que s'utilitza en aquest tros de codi. Així doncs, el programador només ha de tindre en compte aquest registre.

Dins del bucle, s'executen les instruccions que no fan res, ja que la seva finalitat només és perdre temps, i es decrementa el registre r16.

Aquest bucle s'executa fins que el registre assoleix el valor 0. Arribats en aquest punt, s'acaba aquest tros de codi.

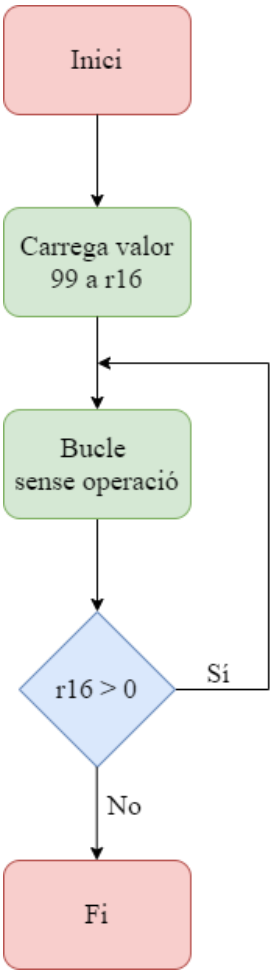



***Figura 5**. Diagrama de Flux del Bucle Delay.*

5.

Novament, consultant el manual del joc d'instruccions, es pot saber quants cicles tarda a executar-se cada instrucció.

```
ldi r16, 99                    ;1 cicle

loop:                          ;1 cicle
      nop                      ;1 cicle
      nop                      ;1 cicle
      nop                      ;1 cicle
      nop                      ;1 cicle
      nop                      ;1 cicle
      nop                      ;1 cicle
      nop                      ;1 cicle
      nop                      ;1 cicle
      dec r16                  ;1 cicle
      brne loop                ;1(No salta)/2(Salta) cicles
```

Com que en aquest cas hi ha un bucle, en total es tarden els següents cicles a executar l'algorisme anterior:

```
delay_1ms:
            ldi r16, 99        ;1 cicle
loop:
            nop                ;1 cicle
            nop                ;1 cicle
            nop                ;1 cicle
            nop                ;1 cicle
            nop                ;1 cicle
            nop                ;1 cicle
            nop                ;1 cicle
            dec r16            ;1 cicle
            brne loop          ;1(No salta)/2(Salta) cicles
```

Així doncs, tenim que:

$$(1) + \big((7 + 1 + 2) \cdot 98\big) + (7 + 1 + 1) = 990\ cicles$$

Pel cas del *loop*, en l'última volta, com que ja no salta, només tarda 1 cicle. A més, segons el manual de l'ATmega328P:

***Default Clock Source:*** *The device is shipped with internal RC oscillator at 8.0 MHz and with the fuse CKDIV8 programmed, resulting in 1.0 MHz system clock.*

Per a calcular el temps d'execució, es pot deduir la següent equació:

$$t_{delay} = \frac{n_{cicles}}{f_{clock}} \cdot = \frac{990}{1 \cdot 10^6\ Hz} = 9{,}90 \cdot 10^{-4}\ s$$

En definitiva, s'aconsegueix un retard de gairebé 1 ms.

### 4.2. Treball al Laboratori

2.

Un cop copiat el tros de codi subministrat, cal compilar el projecte prement el botó  i comprovar que en la finestra output no hi ha cap error.

3.

Seguint els passos indicats en l'enunciat, en la finestra Processor Status, es poden observar els següents resultats, els quals coincideixen amb els obtinguts a través de l'anàlisi del codi.

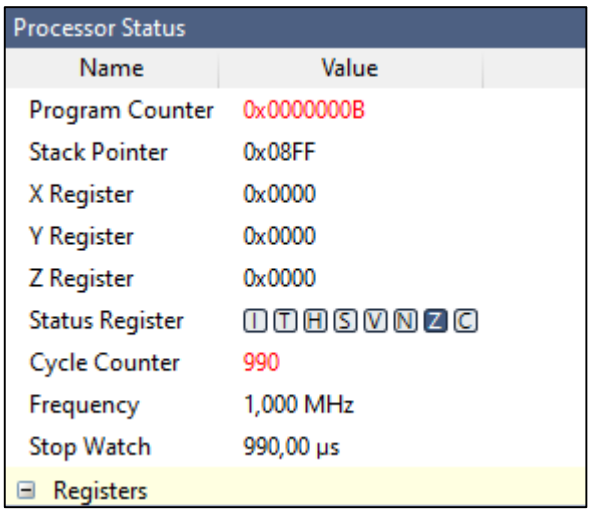



***Figura 6**. Resultats del Simulador del Bucle Delay.*

A més, depurant pas a pas i observant el paràmetre Cycle Counter, es pot comprovar quants cicles tarda cada instrucció a executar-se.

En conclusió, es pot afirmar que utilitzant el bucle delay, s'aconsegueix un retard de gairebé 1 ms, amb una freqüència de rellotge d'1 MHz.

### 5.5. Pràctica 2: Interfícies d'Usuari de la Xplained mini

UNIVERSITAT ROVIRA i VIRGILI


# PRÀCTICA 2:
# Interfícies d'Usuari de la Xplained mini

**Assignatura:** Microcontroladors i Sistemes Embedded

**Ensenyament:** Enginyeria de Sistemes i Serveis de Telecomunicacions

**Curs:** 2020/21

# 1. Introducció i Objectius

Els objectius d'aquesta pràctica són:

- Aprofundir en la cerca d'informació en el manual de l'ATmega328P. En aquest cas, cal que l'alumne es familiaritzi amb el *pinout* del dispositiu i amb els registres necessaris per a treballar amb els ports.
- Entendre el funcionament d'un polsador mecànic i conèixer com tractar-lo per a obtenir la lectura correcta.
- Emular el primer programa sobre el kit de desenvolupament. Per a fer-ho, cal que l'alumne cerqui la informació sobre la interfície d'usuari de la placa en el manual corresponent.

# 2. Estudi Previ

1. Per a poder dur a terme la present pràctica, cal cercar el *pinout* del dispositiu en el manual de l'ATmega328P.
   Aquesta descripció és necessària per a poder conèixer quina funcionalitat té cada pin i, així, poder interactuar amb l'exterior. Atès que, a fi de monitoritzar i controlar els perifèrics, aquests s'han de connectar als pins del xip.
   En aquest cas, a causa del tipus d'encapsulat emprat pel kit de desenvolupament Xplained mini, el microcontrolador té 32 pins d'entrada i sortida.

2. Pel cas del present dispositiu, aquest disposa de tres ports digitals, anomenats: PORTB, PORTC i PORTD. Els quals s'escriuen de forma general com PORTxn, essent respectivament la lletra del port i el número del pin. En aquest sentit, els ports b i d tenen 8 bits i el c 7. En aquest cas, cada bit s'anomena pin i aquest es pot comportar com una entrada o una sortida del sistema.
   A més, cada port consisteix de tres registres ubicats en l'espai I/O. Cal cercar informació al respecte i detallar quan cal utilitzar un o altre.

| **TIP** | Ports d'Entrada i Sortida |
|---|---|
|  | Tal com s'indica en el manual del microcontrolador: *"All AVR ports have true Read-Modify-Write functionality when used as general digital I/O ports."* És a dir, es pot modificar l'estat d'un bit determinat, sense causar alteracions en els altres bits del port, utilitzant les instruccions SBI i CBI. Cada nivell de pin, implica una funcionalitat o acció diferent. |

3. A fi de començar a treballar amb la placa de desenvolupament, cal cercar a quins pins estan connectats l'*User LED* i el polsador d'aquesta.

4. Cal obrir el fitxer adjunt P2_1, comentar cada línia del codi i explicar l'objectiu principal del programa. És recomanable utilitzar l'eina del simulador.

| **TIP** | Inicialització d'un algorisme en assemblador |
|---|---|

A partir d'ara, tot algorisme desenvolupat, ha de començar amb les següents línies de codi:

```
reset:
ldi r16, 0x80           ;Carrega el valor 0x80 al registre r16
sts CLKPR, r16          ;Carrega el valor de r16 a l'espai de
                        ;memòria
ldi r16, 0x04           ;Carrega el valor 0x04 al registre r16
sts CLKPR, r16          ;Carrega el valor de r16 a l'espai de
                        ;memòria
```

Com s'ha comentat en la pràctica anterior, conèixer la freqüència de treball del dispositiu és essencial per a poder obtenir la solució esperada.
Si bé, a partir d'ara, com que l'objectiu final és emular el projecte sobre el kit, cal tindre en compte la freqüència de la placa, i no la del microcontrolador.
Tal com s'indica en el respectiu manual, el Xplained mini té una freqüència per defecte de 16 MHz.

No obstant això, aquesta es pot configurar mitjançant el *prescaler*. Així doncs, les quatre primeres línies de codi tenen la finalitat d'establir la freqüència de treball de la placa Xplained mini a 1 MHz. Per a fer-ho, cal utilitzar el *prescaler* per a disminuir la freqüència del rellotge del sistema.
A fi d'evitar canvis involuntaris, cal seguir el procediment especial indicat en el manual i plasmat en el present codi.
Convé destacar que els detalls teòrics es detallen més endavant. Ara, només cal conèixer la funcionalitat final d'aquest tros de codi.

5. Dibuixar el diagrama de flux de l'anterior algorisme.

6. Ara, es treballarà conjuntament amb el LED i el polsador de la placa. Cal escriure un programa que mentre es polsi el polsador del kit, el LED estigui encès. Quan es deixi de prémer, el LED s'ha d'apagar.
   A fi de facilitar el procés de programació, a continuació, es mostra el diagrama de flux de l'algorisme.

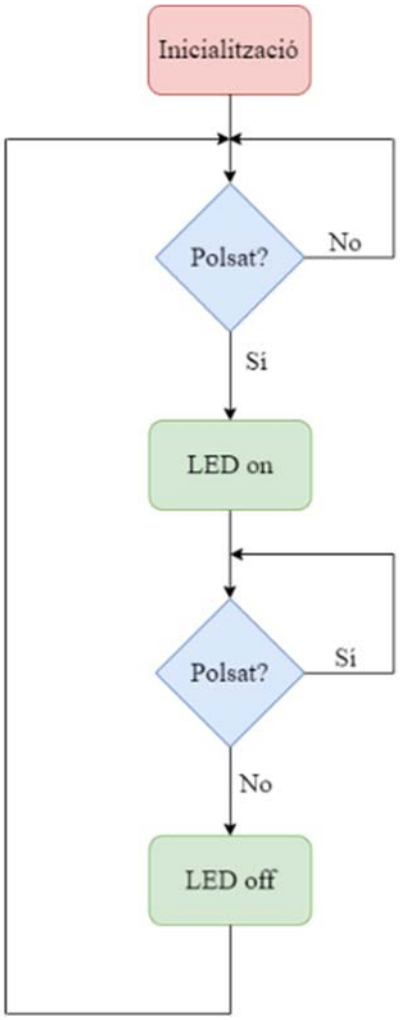
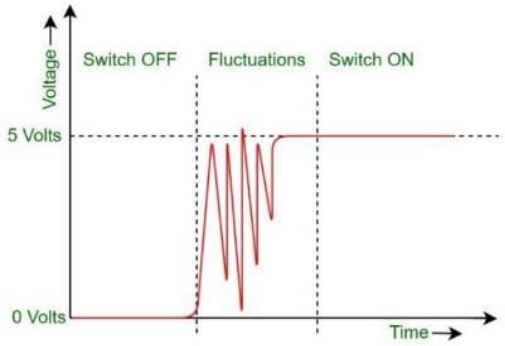



***Figura 1**. Diagrama de Flux de l'Algorisme: Si Polsador premut, LED on.*

7. Ara, es treballarà conjuntament amb el LED i el polsador de la placa. Abans, però, cal entendre el funcionament de lectura del senyal d'un polsador.
   En aquest cas, es considera que el temps de rebot mai serà superior a 30 ms.
   Així doncs, primer, cal crear un bucle de retard de 30 ms.
   Cal obrir el fitxer P2_3 i estudiar i comentar el codi que genera un retard.

| **TIP** | Lectura d'un Polsador |
|---|---|

Cal tindre en compte que tota activació d'un polsador, provoca un efecte rebot. Concretament, a causa de la seva construcció mecànica, durant el canvi d'estat, on/off o off/on, el polsador genera una transició oscil·lant fins que no s'estabilitza. En conseqüència, s'introdueixen errors en la lectura, ja que es registren més commutacions de les que realment ha generat l'usuari.




A fi de solucionar aquest problema, s'afegeix un retard per a assegurar que realment s'ha realitzat l'activació del polsador i el canvi d'estat no ha estat ocasionat per soroll o per les fluctuacions abans d'estabilitzar-se.

8. A continuació, es mostra un diagrama de flux per a llegir un polsador, considerant l'efecte rebot tant en prémer com en deixar de prémer el polsador.
   Cal comentar la següent figura.

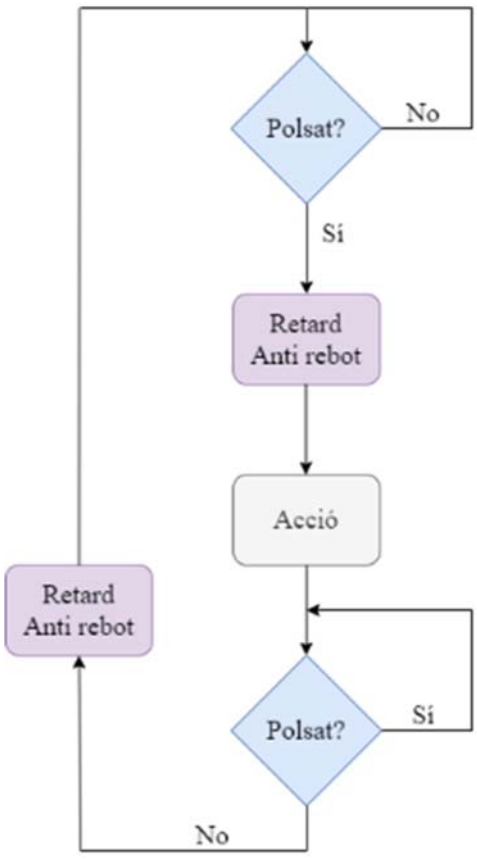



***Figura 2**. Diagrama de flux de la Lectura d'un Polsador evitant l'Efecte Anti-rebot.*

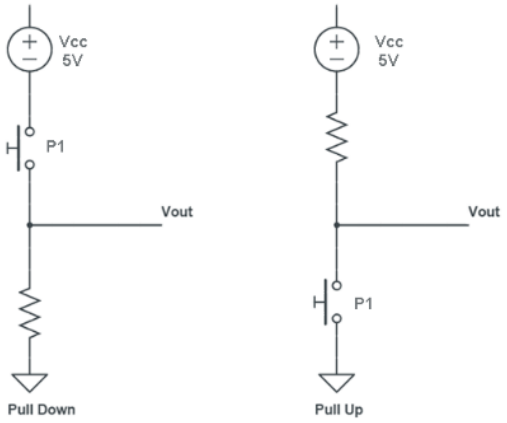
## 3. Treball al Laboratori

1. Cal generar un nou projecte i emular sobre la placa de desenvolupament el programa que genera pampallugues en l'User LED.
   Per a fer-ho, cal canviar l'eina de depuració, seleccionant mEDBG i la interfície debugWIRE.

2. Utilitzant el bucle de retard de 30 ms desenvolupat en l'estudi previ i el diagrama de flux de la lectura d'un polsador evitant l'efecte rebot, cal escriure una rutina que, partint de l'estat inicial del LED apagat, l'encengui i l'apagui cada vegada que es premi el polsador.
   A fi de facilitar la programació i obtenir una solució més òptima, primer, es recomana realitzar un diagrama de flux de l'aplicació.

| **TIP** | Tipus de Polsadors |
|---|---|

Existeixen dos mètodes per a evitar que el polsador es quedi en estat d'alta impedància quan està obert. Atès que aquest estat pot provocar falsos estats, els quals es produeixen a causa del soroll dels circuits electrònics, que poden derivar en errors, sobre consums i averies.
Així doncs, a fi d'assegurar un estat lògic segur en un determinat pin d'entrada, existeixen les següents configuracions.
D'una banda, la resistència *pull-down*, la qual està connectada entre la sortida digital i GND, establint un estat baix.
De l'altra, la resistència pull-up, la qual està connectada entre la sortida i l'alimentació, establint un estat alt.



# 4. Solució Esperada

## 4.1. Estudi Previ

1.

En la Figura 1, extreta del manual del microcontrolador ATmega328P, es mostren els 32 pins d'entrada i sortida, juntament amb la seva designació i disposició física. Seguidament, es realitza una descripció dels pins.

Evidentment, es necessita subministrar la tensió d'alimentació adequada al dispositiu, els pins que duen a terme aquesta funció són del 3 al 6 i el 18 i 21. Si bé, convé destacar que els pins AREF i AVCC estan relacionats amb el convertidor A/D. Com els pins 19 i 22 que també estan específicament destinats a aquest perifèric.

La resta de pins, constitueixen els ports, és a dir, la interfície mitjançant la qual el xip es comunica amb el món exterior. Aquests estan *memory mapped*, el que significa que cada port té la seva adreça de memòria específica.

Cal destacar que la majoria dels pins comparteixen funcionalitats; el que es coneix com a *pin-sharing*, que permet als microcontroladors oferir més perifèrics que el nombre de pins físics que tenen. Consegüentment, l'usuari ha de seleccionar quina funció li interessa més a cada moment. Per exemple, hi ha el pin de reset i els dos pins de l'oscil·lador de cristall, els quals proporcionen el pols de rellotge necessari per a la sincronització.

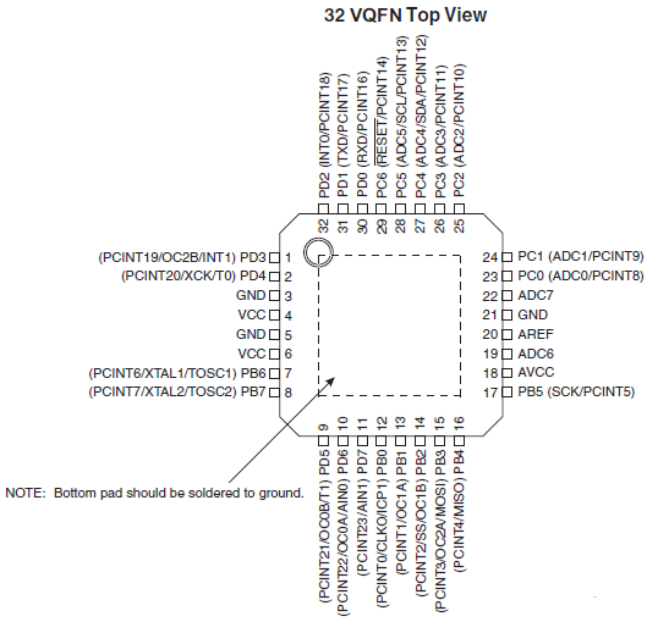



***Figura 3**. Pinout de l'ATmega328P.*

2.

Tal com s'indica en el manual, cada port es gestiona amb tres registres ubicats en l'espai I/O:

- **DDRx** – Registre de direcció de dades: Com el seu nom indica, serveix per a determinar la direcció del pin. És a dir, per a configurar si un pin es comporta com una entrada o com una sortida, quan s'hi escriu un 0 o un 1 respectivament.

- **PORTx** – Registre de dades: Com l'anterior, és un registre de lectura i escriptura. Si el pin està configurat com un *output*, aquest s'utilitza per a activar els bits desitjats.

- **PINx** – *Port Input Pins*: Aquest registre només és de lectura, ja que s'utilitza per a llegir l'estat dels pins, és a dir, per a emmagatzemar el valor del pin.

Així doncs, pel cas del portb:

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| 0x05 (0x25) | PORTB7 | PORTB6 | PORTB5 | PORTB4 | PORTB3 | PORTB2 | PORTB1 | PORTB0 | PORTB |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| 0x04 (0x24) | DDB7 | DDB6 | DDB5 | DDB4 | DDB3 | DDB2 | DDB1 | DDB0 | DDRB |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| 0x03 (0x23) | PINB7 | PINB6 | PINB5 | PINB4 | PINB3 | PINB2 | PINB1 | PINB0 | PINB |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | |

***Figura 4**. Registres del PORTB.*

3.

En aquest cas, cal consultar el manual del kit de desenvolupament Xplained mini per a poder conèixer a quin pin dels 32 disponibles es connecten el polsador i el LED de la placa.

Concretament, com es mostra en els fragments de l'esquemàtic de la placa, el LED està connectat al portb5 i el polsador al portb7.

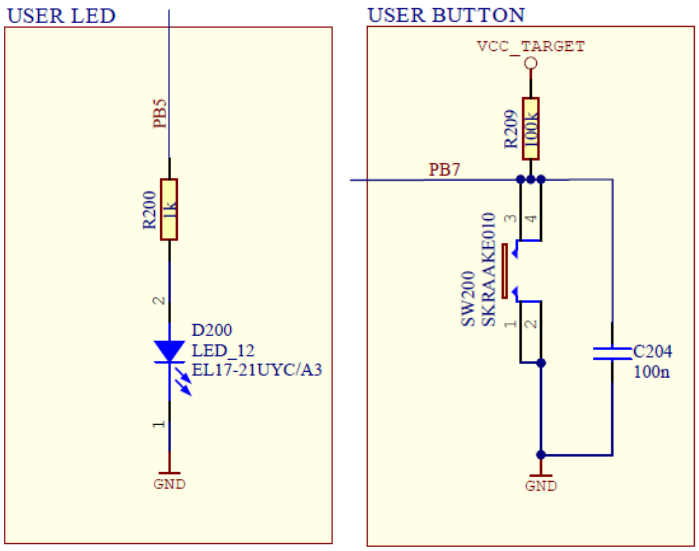



***Figura 5**. Detall de l'User LED i del Polsador de la Placa.*

4.

Així doncs, un cop estudiat el codi, es poden realitzar els següents comentaris:

```
.include "m328pdef.inc"
.device atmega328p                  ;Microcontrolador que s'utilitza

.org 0x0000                         ;Programa comença a l'adreça 0x0000
rjmp reset


reset:                              ;Freqüència de treball a 1 MHz
      ldi r16, 0x80                 ;Carrega el valor 0x80 al r16
      sts CLKPR, r16                ;Carrega valor a l'espai de memòria
      ldi r16, 0x04                 ;Carrega el valor 0x04 al r16
      sts CLKPR, r16                ;Carrega valor a l'espai de memòria

start:
      ldi r17, 0xff                 ;Carrega 0xff a r17
      out DDRB, r17                 ;Assigna PORTB com a sortida

blink:
      ldi r17, 0b00100000
      out PORTB, r17                ;LED on, posa a 1 pin1 del PORTB

;Delay
      ldi r17, 0xff                 ;Carrega valor més gran possible r17
loop1:
      ldi r16, 99                   ;Carrega el valor 99 al registre r16
loop2:
      nop                           ;Bucle amb instruccions que no fan
      nop                           ;res
      nop                           ;Serveixen per esperar
      nop
      nop
      nop
      nop
      dec r16                       ;Resta 1 del valor que hi ha en r16
      brne loop2                    ;Salt condicional, salta si no és
                                    ;igual a 0
      dec r17                       ;Resta 1 del valor que hi ha en r17
      brne loop1                    ;Salt condicional, salta si no és
                                    ;igual a 0


      ldi r17, 0b00000000
      out PORTB, r17                ;LED off

;Delay
      ldi r17, 0xff                 ;Carrega valor més gran possible r17
loop3:
      ldi r16, 99                   ;Carrega el valor 99 al registre r16
loop4:
      nop                           ;Bucle amb instruccions que no fan
      nop                           ;res
      nop                           ;Serveixen per esperar
      nop
      nop
      nop
      nop
      dec r16                       ;Resta 1 del valor que hi ha en r16
      brne loop4                    ;Salt condicional, salta si no és
```

```
                                ;igual a 0
dec r17                         ;Resta 1 del valor que hi ha en r17
brne loop3                      ;Salt condicional, salta si no és
                                ;igual a 0

rjmp blink
```

A més, per a obtenir el retard s'utilitza el bucle estudiat en la pràctica anterior. Però perquè aquest retard sigui més llarg i el canvi sigui visible per l'ull humà, es posa dins d'un altre bucle. El comptador del qual se li assigna el valor més gran que un nombre de 8 bits pot suportar, 0b11111111.
En conseqüència, s'obté un programa que fa fer pampallugues a l'User LED de la placa de desenvolupament.

Cal destacar, que aquest programa utilitza els registres r16 i r17.

Finalment, si se simula el programa, després d'executar la instrucció `out PORTB, r17,` s'obté el següent:

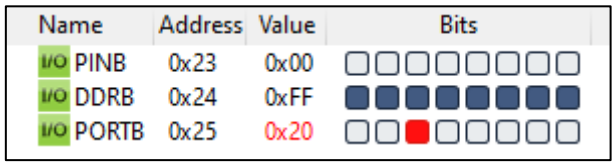



***Figura 6**. Detall del Simulador amb l'User LED activat.*

Com es pot veure, la segona línia indica que s'han configurat tots els pins del portb com a sortides. En la tercera, es pot veure com el pin 5 està a nivell alt, és a dir, quan s'emuli el programa sobre el hardware, aquest pin rebrà un corrent d'entrada que encendrà el LED.

5.

El diagrama de flux de l'anterior algorisme és el següent:

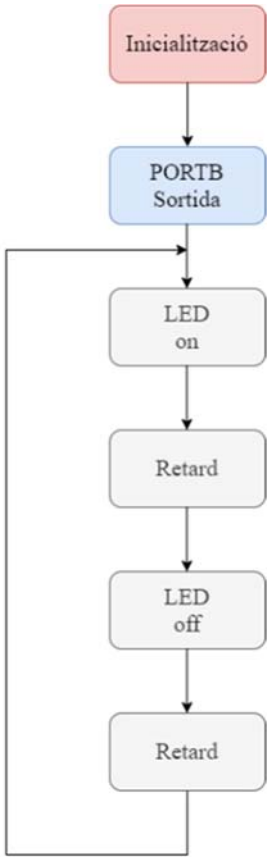



***Figura 7**. Diagrama de Flux de les Pampallugues de l'User LED.*

Com es pot observar en l'anterior figura, primer cal realitzar la inicialització necessària, canviar el valor del *prescaler* per a treballar a una freqüència d'1 MHz i inicialitzar el SP a la primera adreça de memòria. Seguidament, es duen a terme els estats necessaris perquè el LED faci pampallugues, amb un interval de temps, el qual es calcula en el següent exercici.

6.

A partir del diagrama de flux mostrat en l'enunciat, s'ha desenvolupat el següent codi:

```
.include "m328pdef.inc"
.device atmega328p                  ;Microcontrolador que s'utilitza

.org 0x0000                         ;Programa comença a l'adreça 0x0000
rjmp reset


reset:                              ;Freqüència de treball a 1 MHz
      ldi r16, 0x80                 ;Carrega el valor 0x80 a r16
      sts CLKPR, r16                ;Carrega valor a l'espai de memòria
      ldi r16, 0x04                 ;Carrega el valor 0x04 a r16
      sts CLKPR, r16                ;Carrega valor a l'espai de memòria

start:
      ldi r17, 0b00100000
      out DDRB, r17                 ;Assigna PORTB5 com a sortida i
                                    ;PORTB7 com a entrada

No_polsat:
```

```
      sbic PINB, 7                ;Polsador està al PORTB 7
      rjmp No_polsat

      sbi PORTB, 5                ;LED on

still_pushed:
      sbis PINB, 7
      rjmp still_pushed
      cbi PORTB, 5                ;LED off
      rjmp No_polsat
```

Convé destacar que s'ha simulat el codi i s'ha obtingut el resultat esperat. Com es pot observar en la següent imatge, el pin 5 del port b només està a 1 quan el pin 7 està a zero.

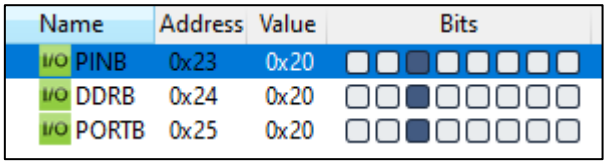



***Figura 8**. Detall del Simulador amb l'User LED activat, quan el PINB7 està a zero.*

7.

En el fitxer P2_3 hi ha el següent tros de codi.

Com s'ha comentat anteriorment, quan es dissenya codi per a ser reutilitzat vàries vegades, cal prestar especial atenció als registres que s'utilitzen. En aquest cas, el següent bucle utilitza els registres 16 i 17. Per tant, sempre que el programador utilitzi aquest bucle per a generar un retard, ha de tindre en compte que aquests registres es manipulen.

```
      ldi r17, 30                 ;Carrega el valor 30 a r17
loop1:
      ldi r16, 99                 ;Carrega el valor 99 al registre r16
loop2:
      nop                         ;Bucle amb instruccions que no fan
      nop                         ;res
      nop                         ;Serveixen per esperar
      nop
      nop
      nop
      nop
      dec r16                     ;Resta 1 del valor que hi ha en r16
      brne loop2                  ;Salt condicional, salta si no és
                                  ;igual a 0
      dec r17                     ;Resta 1 del valor que hi ha en r17
      brne loop1                  ;Salt condicional, salta si no és igual
                                  ;a 0
```

Com es pot observar, s'ha agafat el bucle de retard d'1 ms estudiat en la pràctica anterior i s'ha posat dins d'un altre bucle amb menys iteracions per tal d'aconseguir un retard de 30 ms.

En concret, si es compten els cicles que es tarda a executar el codi, s'obté el següent resultat:

```
delay:
        ldi r17, 30         ;1 cicle
loop_1:
        ldi r16, 99         ;1 cicle
loop_2:
        nop                 ;1 cicle
        nop                 ;1 cicle
        nop                 ;1 cicle
        nop                 ;1 cicle
        nop                 ;1 cicle
        nop                 ;1 cicle
        nop                 ;1 cicle
        dec r16             ;1 cicle
        brne loop_2         ;1(No salta)/2(Salta) cicles
        dec r17             ;1 cicle
        brne loop_1         ;1(No salta)/2(Salta) cicles
```

Així doncs, sabem que el bucle marcat de color blau, tarda a executar-se:

$$\big((7+1+2)\cdot 98\big)+(7+1+1)=989$$

Tenim que:

$$(1)+\big((1+989+1+2)\cdot 29\big)+(1+989+1+1)=29790\ cicles$$

A més, segons el manual de l'ATmega328P:

***Default Clock Source:*** *The device is shipped with internal RC oscillator at 8.0 MHz and with the fuse CKDIV8 programmed, resulting in 1.0MHz system clock.*

Per a calcular el temps d'execució, es pot deduir la següent equació:

$$t_{delay}=\frac{n_{cicles}}{f_{clock}}\cdot=\frac{29790}{1\cdot 10^{6}\ Hz}=0{,}029\ s$$

En definitiva, s'aconsegueix un retard de gairebé 30 ms.

8.

En primer lloc, convé destacar que gràcies als dos retards anti-rebots s'evita l'efecte rebot tant en prémer com en deixar de prémer el polsador, respectivament.

En concret, el programa comença fent una enquesta al pin del port on està connectat el polsador. Segons la resposta obtinguda, es queda en el mateix estat o passa a executar el primer retard. En segon lloc, cal dur a terme les accions necessàries per a satisfer les necessitats de l'aplicació.

Finalment, cal executar novament el retard per a evitar l'efecte rebot quan ja no es prem el polsador.

### 4.2. Treball al Laboratori

1.

En primer lloc, cal generar un nou projecte d'assemblador i copiar-hi la rutina de l'arxiu P2_1.

Seguidament, cal seleccionar l'eina de depuració adequada i començar a depurar pas a pas per a comprovar que el programa s'executa de la manera adequada.

Finalment, s'emula el projecte sense depuració i es comprova que, efectivament, el LED del kit de desenvolupament s'encén i s'apaga amb un interval d'un segon.

2.

En primer lloc, cal determinar els estats del polsador. Segons la Figura 3, la qual s'ha extret del manual del kit de desenvolupament, el pin 7 del portb té una resistència pull-up. Conseqüentment, quan el polsador està obert, l'entrada rep un nivell alt. Per contra, quan està tancat, l'entrada rep un nivell baix.

En segon lloc, cal realitzar el diagrama de flux del programa indicat en l'enunciat. És a dir, cal desenvolupar l'estat designat com a Acció en el diagrama de flux de la Figura 6. El qual s'indica de color verd en la següent figura.

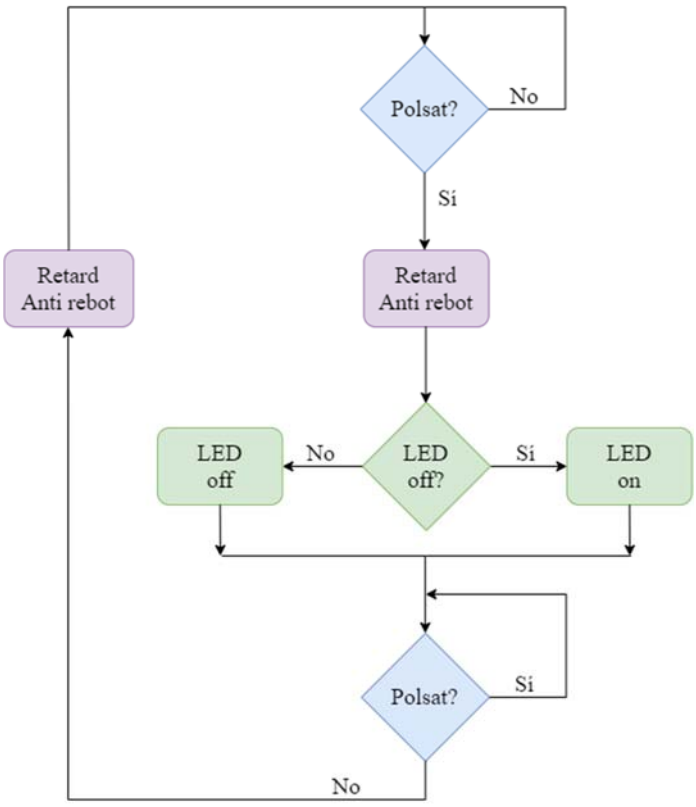



***Figura 9**. Diagrama de Flux per a commutar el LED quan es prem el Polsador.*

Finalment, cal escriure el codi necessari, considerant els dos diagrames de flux de les figures 7 i 8, que el polsador de la placa té connectada una resistència pull-up i la el bucle de retard de 30 ms desenvolupat prèviament.

Així doncs, un cop simulat, emulat sobre la placa i comprovat que té la funcionalitat esperada, es pot afirmar que el codi generat és el següent:

```
.include "m328pdef.inc"
.device atmega328p                  ;Microcontrolador que s'utilitza

.org 0x0000                         ;Programa comença a l'adreça 0x0000
rjmp reset

reset:                              ;Freqüència de treball a 1 MHz
      ldi r16, 0x80                 ;Carrega el valor 0x80 a r16
      sts CLKPR, r16                ;Carrega valor a l'espai de memòria
      ldi r16, 0x04                 ;Carrega el valor 0x04 a r16
      sts CLKPR, r16                ;Carrega valor a l'espai de memòria

start:
;Control pulsador
      ldi r17, 0b00100000
      out DDRB, r17                 ;Assigna PORTB5 com a sortida i
                                    ;PORTB7 com a entrada
      ldi r20, 0x00                 ;r20 = off
      ldi r19, 0x00                 ;r19 = estat - comença a off

No_polsat:
      sbic PINB, 7                  ;Polsador està al PORTB 7
      rjmp No_polsat

;Delay
      ldi r17, 30                   ;Carrega el valor 30 a r17
loop1:
      ldi r16, 99                   ;Carrega el valor 99 al registre r16
loop2:
      nop                           ;Bucle amb instruccions que no fan
      nop                           ;res
      nop                           ;Serveixen per esperar
      nop
      nop
      nop
      nop
      dec r16                       ;Resta 1 del valor que hi ha en r16
      brne loop2                    ;Salt condicional, salta si no és
                                    ;igual a 0
      dec r17                       ;Resta 1 del valor que hi ha en r17
      brne loop1                    ;Salt condicional, salta si no és
                                    ;igual a 0


      cp r20, r19
      brne apagar

      ldi r19, 0xff                 ;estat on
      sbi PORTB, 5                  ;LED on
      rjmp still_pushed

apagar:
      ldi r19, 0x00                 ;estat off
      cbi PORTB, 5                  ;LED off

still_pushed:
      sbis PINB, 7
      rjmp still_pushed
```

```
;Delay
      ldi r17, 30                ;Carrega el valor 30 a r17
loop3:
      ldi r16, 99                ;Carrega el valor 99 al registre r16
loop4:
      nop                        ;Bucle amb instruccions que no fan
      nop                        ;res
      nop                        ;Serveixen per esperar
      nop
      nop
      nop
      nop
      dec r16                    ;Resta 1 del valor que hi ha en r16
      brne loop4                 ;Salt condicional, salta si no és
                                 ;igual a 0
      dec r17                    ;Resta 1 del valor que hi ha en r17
      brne loop3                 ;Salt condicional, salta si no és
                                 ;igual a 0


      rjmp No_polsat
```

### 5.6. Pràctica 3: Ports d'Entrada i Sortida

UNIVERSITAT ROVIRA i VIRGILI


# PRÀCTICA 3:

# Ports d'Entrada i Sortida

**Assignatura:** Microcontroladors i Sistemes Embedded

**Ensenyament:** Enginyeria de Sistemes i Serveis de Telecomunicacions

**Curs:** 2020/21

## 1. Introducció i Objectius

Els objectius d'aquesta pràctica són:

- Creació i utilització de la primera rutina inclosa en una llibreria, ja que és una pràctica molt estesa per a facilitar l'escriptura de codi i obtenir una millor solució.
- A fi que l'alumne treballi amb diferents components i funcionalitats del microcontrolador, l'objectiu de la present pràctica i les següents és simular el funcionament d'un semàfor. En conseqüència, cal utilitzar un mòdul extern equipat amb tots els components necessaris.
- Coneixement del funcionament i les característiques del mòdul, a través del document Especificacions Mòdul Extern.
- Concretament, en aquesta pràctica, cal desenvolupar l'algorisme que executi la seqüència dels tres LEDs.
- Millora de la rutina que gestiona l'efecte rebot dels polsadors.

## 2. Estudi Previ

1. Com s'ha constatat durant la realització de les dues pràctiques anteriors, sovint, diverses aplicacions requereixen utilitzar el mateix tros de codi. Així doncs, a fi de reutilitzar el bucle de retard d'1 ms utilitzat prèviament, es crea una rutina anomenada delay_1ms.
   Cal estudiar i comentar el codi del fitxer P3_1.

| **ATENCIÓ** | Ús de Registres |
|---|---|
|  | El programador ha de prestar especial atenció al fet que aquesta rutina de retard utilitza el registre 16. |

2. El següent pas en l'optimització de reutilització de codi és la creació de llibreries pròpies. Així doncs, cal crear una llibreria que contingui la rutina delay_1ms i afegir-la a un nou projecte, anomenat P3_2.
   Cal fer-ho pels dos mètodes que es detallen a continuació.

| **TIP** | Llibreries modulars |
|---|---|
|  | A fi d'obtenir una llibreria encara més reutilitzable, es recomana desenvolupar més d'una rutina a partir de crides a la rutina base de delay_1ms. En conseqüència, l'usuari disposarà de diferents duracions de retard dins de la mateixa llibreria.<br><br>Per exemple, per a obtenir el retard de 30 ms necessari per a implementar l'efecte anti-rebot en la lectura d'un polsador, es pot realitzar de la següent manera:<br><br>`delay_30ms:`<br>`  rcall delay_10ms  ;Crida de la subrutina delay_10ms`<br>`  rcall delay_10ms`<br>`  rcall delay_10ms`<br>`  ret  ;Retorn de la subrutina` |

**ATENCIÓ** Tipus de Funció segons el Tipus de Codi

En primer lloc, convé destacar que es poden diferenciar dos tipus de llibreries segons el tipus de codi utilitzat.
D'una banda, la principal característica del codi reubicable és que es pot carregar a qualsevol posició de memòria disponible. A fi d'executar correctament les instruccions, les direccions que depenen de la ubicació física del programa es calculen en temps d'execució.

En canvi, el codi no reubicable sempre s'emmagatzema en les mateixes posicions de memòria. La primera d'aquestes és indicada pel programador amb la directiva `.org`. Conseqüentment, si es vol afegir codi, primer cal realitzar un procés d'integració per tal d'evitar conflictes generats per la repetició d'etiquetes, solapament d'adreces de memòria o per un ús múltiple no previst de la mateixa adreça. Així doncs, aquest tipus de codi és difícilment reutilitzable.

Des del punt de vista dels sistemes *embedded*, el primer tipus és millor, ja que permet distribuir, modificar i afegir codi en diferents mòduls de manera molt més simple perquè no cal tenir en compte la localització. Alguns d'aquests mòduls poden ser llibreries.

**TIP** Creació d'una Llibreria No reubicable en Assemblador

En primer lloc, cal crear la llibreria. Com que la rutina *delay* està escrita en llenguatge assemblador, cal seleccionar el següent tipus d'arxiu:

Es recomana crear una carpeta on s'hi guardaran totes les llibreries noves creades per l'alumne. Així, és més fàcil direccionar les llibreries i assegurar un funcionament correcte. Seguidament, cal escriure el codi que integra la llibreria, en aquest cas la rutina *delay*. Si bé, cal prestar especial atenció a quina posició de memòria comença el codi. Per tal d'evitar problemes, com que el programa principal s'emmagatzema a partir de l'adreça 0x00, es recomana començar aquesta rutina amb la directiva `.org 0x300`.

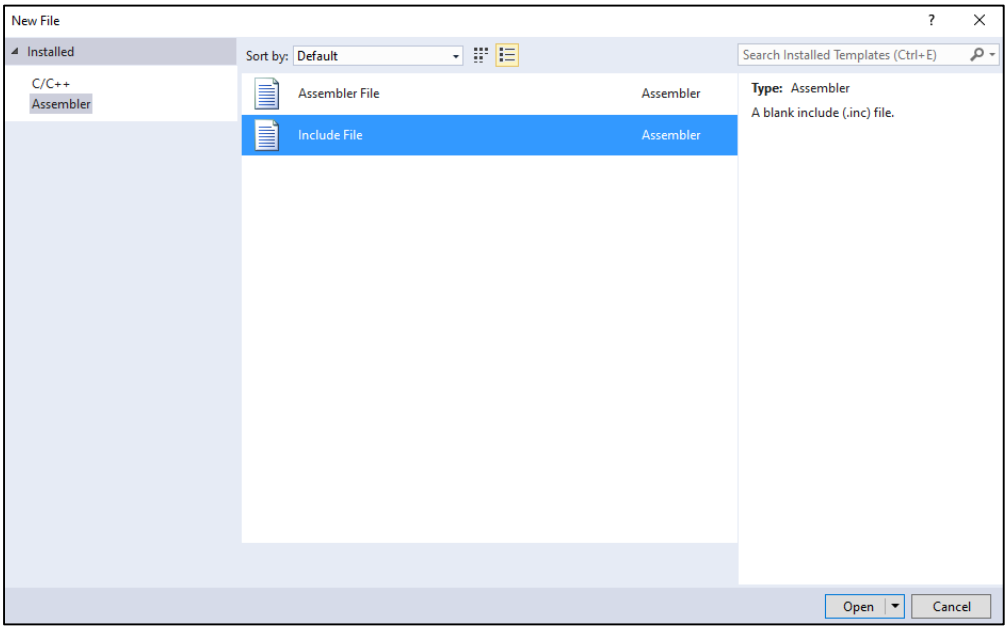



***Figura 1**. Creació d'un nou Fitxer de tipus include d'Assemblador.*

| **TIP** | Adreçament d'una Llibreria no reubicable en un Projecte Assemblador |
|---|---|

A fi de direccionar correctament la llibreria en un projecte, per tal de realitzar qualsevol crida. Cal seguir els següents passos:

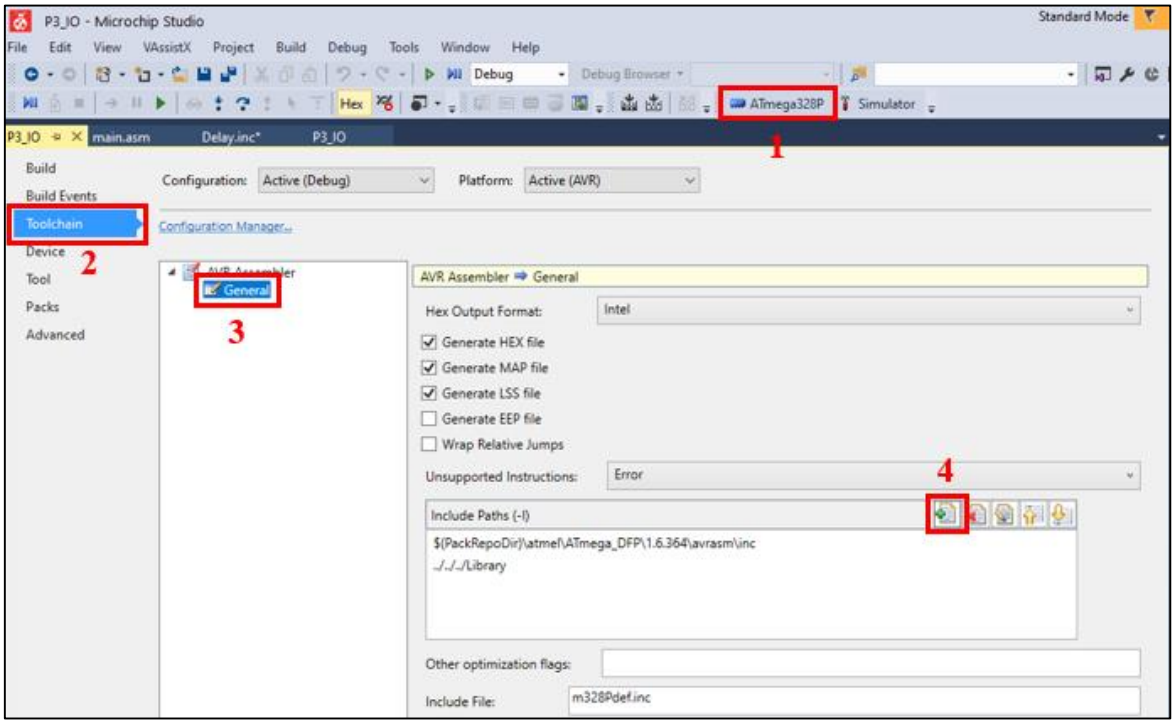



***Figura 2**. Direccionament d'una Llibreria en un Proejecte en Assemblador.*

Al polsar la icona número 4, s'obre una altra finestra en la qual cal introduir el *path* de la carpeta amb les llibreries, anomenada per exemple, *Library*.
Arribats en aquest punt, ja es poden incloure les llibreries necessàries a l'inici del programa principal amb la directiva:

```
.include "Nom_Llibreria.inc"
```

Finalment, cal compilar el projecte per a assegurar que no hi ha cap error.

---

| **TIP** | Creació d'una Llibreria reubicable en Assemblador |
|---|---|

En aquest cas, com que es genera una llibreria reubicable, primer, s'ha de crear un fitxer de tipus .asm, el qual contindrà el codi de la llibreria.

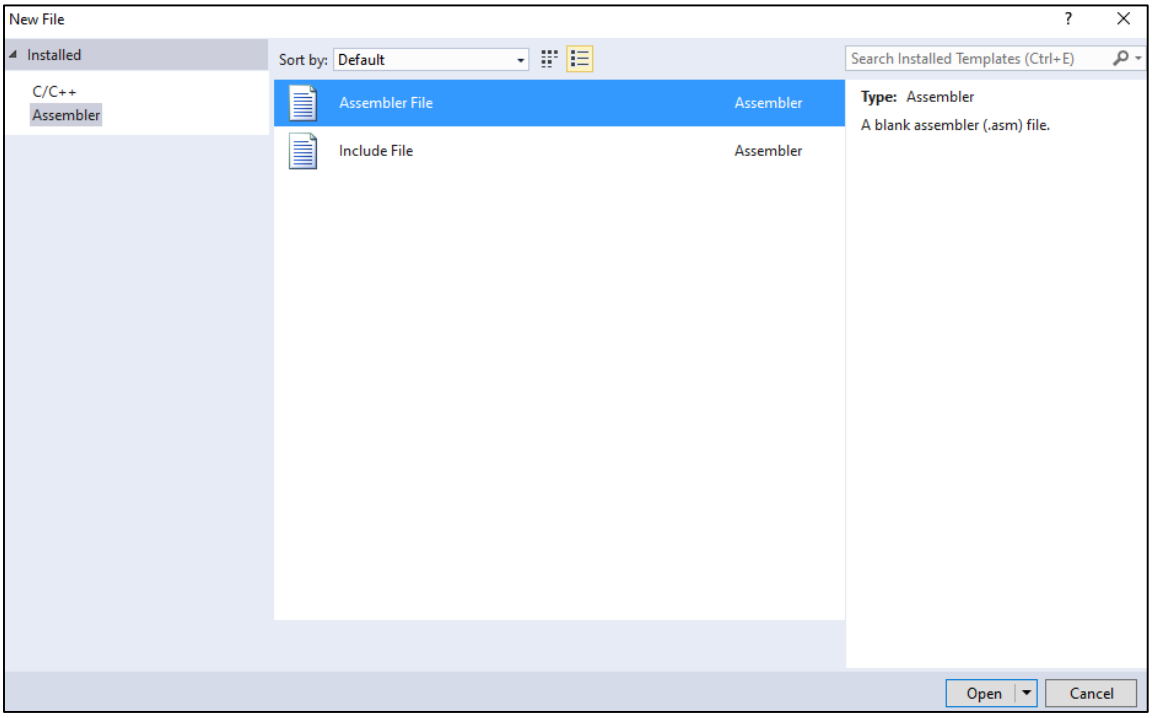



***Figura 3**. Creació d'un nou Fitxer de tipus Assembler.*

Evidentment, com que es tracta de codi reubicable, aquest no ha de contenir la directiva `.org`.

---

| **TIP** | Afegir una Llibreria reubicable en un Projecte Assemblador |
|---|---|

Un cop creat el fitxer .asm, en aquest cas el fitxer delay.asm, s'ha d'incloure manualment dins de la carpeta del projecte on es desitja utilitzar.

Seguidament, cal seguir la següent seqüència per a incloure l'arxiu en el projecte: ***Show all files*** (1) – clic dret sobre el nom del fitxer (2) – ***Include in Project*** (3).

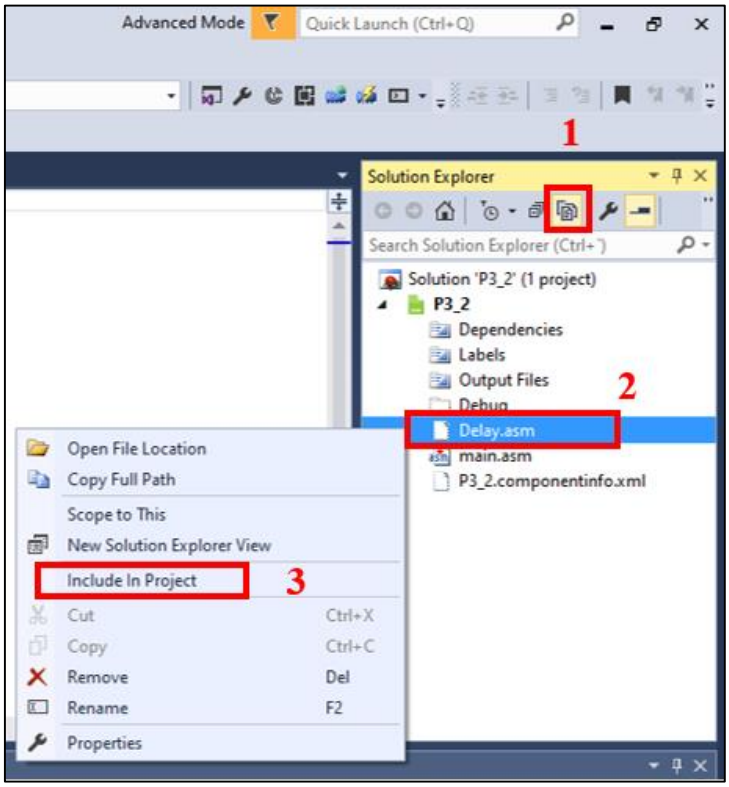



***Figura 4**. Afegir un Fitxer en un Projecte Assemblador.*

Atès que el compilador utilitzat és bastant senzill, cal tenir en compte els següents punts:

- En el programa principal, cal escriure la següent línia de codi: `.include "delay.asm"`. Si bé, només es genera la solució correctament si s'escriu després de la directiva `.org` i la instrucció `rjmp reset`.
- És a dir, l'inici del *main* ha de quedar de la següent manera:

```
.include "m328pdef.inc"
.device atmega328p

.org 0x0000
rjmp reset
.include "delay.asm"
```

- Finalment, cal comprovar que la llibreria s'ha inclòs correctament, verificant que apareix sota del desplegable *Dependencies*. Si és així, ja es poden realitzar les crides a les funcions necessàries.

---

3. Al llarg dels següents exercicis, es treballa amb les rutines necessàries per a poder simular el funcionament d'un semàfor. Els components del qual són perifèrics de la placa de desenvolupament Xplained mini.
   Així doncs, en primer lloc, cal conèixer a quins pins de la placa es connecten. Per a obtenir la informació necessària, cal consultar el document Especificacions Mòdul Extern. En concret, al llarg d'aquesta pràctica només s'utilitzen els tres LEDs i el polsador.

4. En el següent diagrama de flux, es mostra la seqüència dels tres LEDs del semàfor. Cal desenvolupar el codi corresponent, incloent-hi les crides necessàries a la llibreria referenciada al projecte.
   Es recomana simular el projecte per a detectar possibles errors o millores.

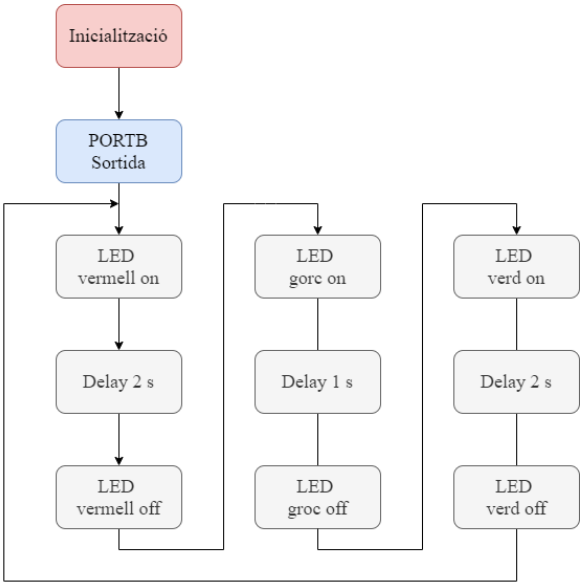



***Figura 5**. Diagrama de Flux de la Seqüència dels LED d'un Semàfor.*

## 3. Treball al Laboratori

1. Cal emular sobre la placa de desenvolupament el programa de la seqüència dels tres LEDs del semàfor.

2. Cal realitzar una millora en la gestió de la pulsació d'un polsador. Atès que la utilitzada fins al moment, sí que té en compte els rebots ocasionats en prémer i deixar de prémer el polsador. Però no té en compte una possible activació errònia a causa del soroll generat pels circuits electrònics.
   Així doncs, a partir del següent diagrama de flux, cal desenvolupar l'algorisme necessari.

| **TIP** | Tractament de l'activació d'un polsador |
|---|---|
|  | Per a realitzar correctament el control de l'activació d'un polsador, cal tindre en compte la seva construcció mecànica. D'una banda, quina és la seva posició de repòs, si és normalment obert o normalment tancat. De l'altra, quin tipus de configuració s'utilitza, és a dir, si té connectada una resistència pull-down o pull-up en la seva sortida digital.<br><br>Com es pot observar en el diagrama de flux següent, els estats marcats de color lila realitzen el retard per tal d'evitar els rebots provocats tant en prémer com en deixar de prémer el polsador. |

A més, cal afegir el tractament per tal d'evitar activacions indesitjades a causa del soroll. Consegüentment, passat el primer retard, cal tornar a comprovar si el polsador encara està activat. Si ja no ho està, significa que l'activació ha sigut provocada per soroll, per això cal tornar a l'inici de la rutina.
Si encarà està polsat, cal avançar al següent estat. El qual permet que fins que el polsador no es deixi de polsar, no es duen a terme les accions que desencadena l'activació d'aquest.

No obstant això, hi ha dispositius que requereixen que les accions es duguin a terme quan es prem el polsador, és a dir, abans de comprovar si encara està polsat.

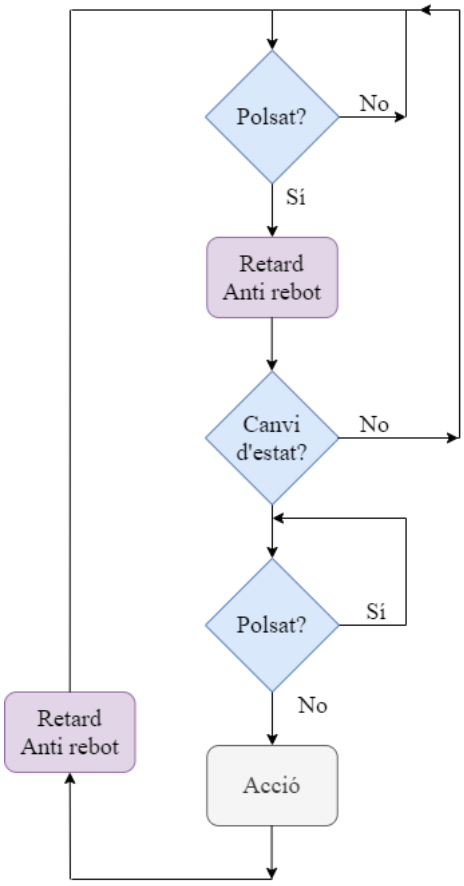



***Figura 6**. Diagrama de Flux del Control de l'Activació d'un Polsador.*

3. Finalment, per a comprovar la correcta gestió de l'activació d'un polsador, cal incloure en l'algorisme de la seqüència dels tres LEDs l'acció d'un polsador. El qual únicament marca l'inici de la seqüència lluminosa.

# 4. Solució Esperada

## 4.1. Estudi Previ

1.

El codi que forma la rutina delay_1ms és el següent:

```
delay_1ms:
      ldi r16, 99             ;Carrega el valor 99 al registre r16
loop:                         ;Etiqueta
      nop                     ;Bucle amb instruccions que no fan res
      nop                     ;Serveixen per esperar
      nop
      nop
      nop
      nop
      nop
      dec r16                 ;Resta 1 del valor que hi ha en r16
      brne loop               ;Salt condicional, salta si no és igual
      ret                     ;Retorn de la subrutina
```

Com es pot observar, el codi que forma el bucle és igual que l'utilitzat en les anteriors pràctiques.

Si bé, com que ara es crea una rutina, aquesta ha de començar amb una etiqueta, la qual indica l'inici de la subrutina, i ha d'acabar amb la instrucció `ret`. Mitjançant la qual, el compilador sap que ha de retornar de la subrutina al programa principal.

Convé destacar, que en afegir la instrucció `ret`, segons el manual del joc d'instruccions, el nombre de cicles s'ha vist augmentat en 4. En conseqüència, s'obté un retard més proper a 1 ms.

Concretament, si se simula la rutina delay_1ms, s'obtenen els següents resultats. Cal fixar-se en el comptador de cicles i el Stop Watch.

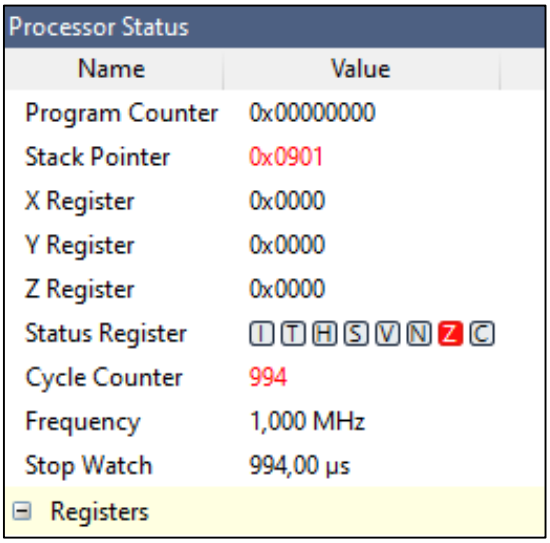



***Figura 7**. Resultats del Simulador de la Rutina Delay_1ms*

2.

Seguint els passos de l'enunciat, d'una banda, es crea una llibreria no reubicable amb la rutina *delay* utilitzada prèviament i es referencia la carpeta que contindrà totes les llibreries generades per l'alumne.

Així doncs, s'aconsegueix el següent projecte:

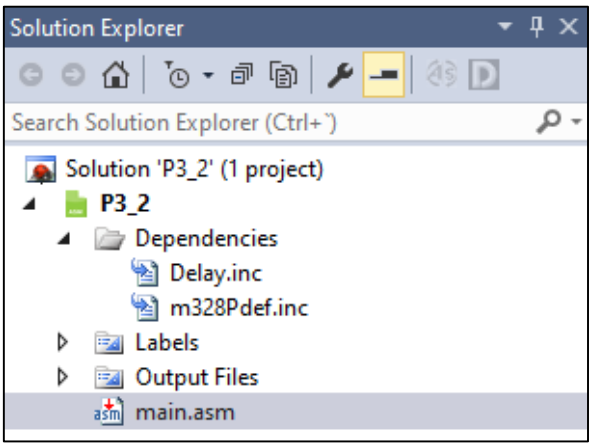



***Figura 8**. Solució generada amb la Llibreria no reubicable referenciada.*

Com es pot observar en l'anterior figura, dins del projecte P3_2, hi ha referenciades la llibreria de creació pròpia, *delay*, i el fitxer, m328Pdef, on es declaren les definicions apropiades dels ports I/O del dispositiu.

De l'altra, si s'afegeix una llibreria reubicable en el projecte, s'obté la següent solució.

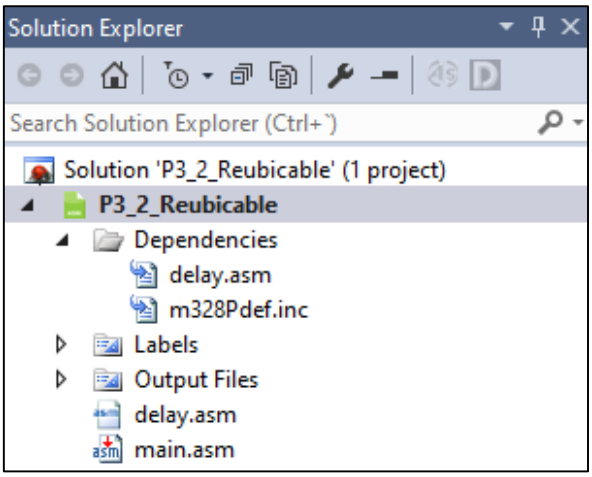



***Figura 9**. Solució generada amb la Llibreria reubicable.*

3.

Segons el document Especificacions Mòdul Extern, la informació necessària per a treballar amb els tres LEDs, vermell, groc i verd, i el polsador del sistema és la següent:

| Component | Port |
|---|---|
| LED verd | PORTB, pin 1 |
| LED groc | PORTB, pin 2 |
| LED vermell | PORTB, pin 3 |
| Polsador | PORTB, pin 4 |

4.

Així doncs, el codi necessari per a realitzar la seqüència dels tres LEDs indicada en l'anterior diagrama de flux és el següent:

```
.include "m328pdef.inc"
.include "Delay.inc"
.device atmega328p

.org 0x0000
rjmp reset


reset:                              ;Freqüència de treball a 1 MHz
      ldi r16, 0x80                 ;Carrega el valor 0x80 a r16
      sts CLKPR, r16                ;Carrega valor a l'espai de memòria
      ldi r16, 0x04                 ;Carrega el valor 0x04 a r16
      sts CLKPR, r16                ;Carrega valor a l'espai de memòria

start:
      ldi r17, 0xff
      out DDRB, r17                 ;Assigna PORTB com a sortida

vermell:
      sbi PORTB, 3                  ;LED vermell on
      call delay_1s
      call delay_1s
      cbi PORTB, 3                  ;LED vermell off

groc:
      sbi PORTB, 2                  ;LED groc on
      call delay 1s
      cbi PORTB, 2                  ;LED groc off

verd:
      sbi PORTB, 1                  ;LED verd on
      call delay_1s
      call delay_1s
      cbi PORTB, 1                  ;LED verd off
      rjmp vermell
```

Durant la simulació d'aquest codi, s'ha pogut comprovar que la crida a la rutina delay_1s, la qual està inclosa en la llibreria Delay, funciona correctament. En la següent imatge, es pot observar el valor del Stop Wacth abans d'executar-se la instrucció que apaga el LED vermell, és a dir, un cop acabat el primer retard de 2 segons indicat en el diagrama.

A més, en el detall dels registres del PORTB, es pot observar com el LED connectat al pin 3, estaria encès.

Per acabar, es realitza el mateix exercici però utilitzant una llibreria reubicable. És a dir, primer, cal crear el fitxer delay.asm i copiar-lo dins de la carpeta del projecte. Evidentment, s'obté el mateix resultat.

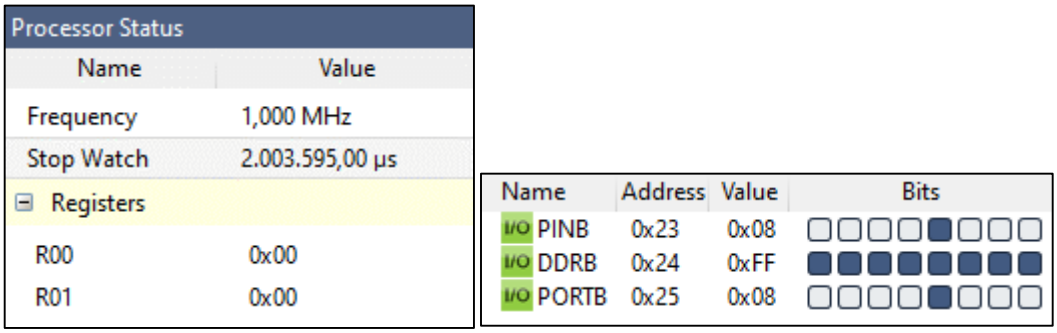



***Figura 10**. Detall del Simulador de la Rutina dels tres LED.*

En definitiva, es pot afirmar que la rutina compleix amb les especificacions indicades en l'enunciat.

### 4.2. Treball al Laboratori

1.

S'emula el projecte sense depuració i es comprova que, efectivament, els tres LEDs segueixen la seqüència marcada pel diagrama de flux de la Figura 4.

2.

Així doncs, tenint en compte tota la informació de l'enunciat i les característiques mecàniques del polsador utilitzat, s'ha desenvolupat el següent codi:

```
No_polsat:
      sbic PINB, 7            ;Polsador està al PORTB 7
      rjmp No_polsat

      call delay_10ms
      call delay_10ms
      call delay_10ms

      sbic PINB, 7            ;Evitar soroll
      rjmp No_polsat

;*************************************************
;Incloure les instruccions relatives a l'acció
;*************************************************

still_pushed:
      sbis PINB, 7
      rjmp still_pushed
      call delay_10ms
      call delay_10ms
      call delay_10ms
      rjmp No_polsat
```

3.

El codi desenvolupat per a complir amb les indicacions de l'enunciat és el següent:

```
.include "m328pdef.inc"
.include "Delay.asm"
.device atmega328p

.org 0x0000
rjmp reset


reset:                              ;Freqüència de treball a 1 MHz
      ldi r16, 0x80                 ;Carrega el valor 0x80 a r16
      sts CLKPR, r16                ;Carrega valor a l'espai de memòria
      ldi r16, 0x04                 ;Carrega el valor 0x04 a r16
      sts CLKPR, r16                ;Carrega valor a l'espai de memòria

start:
      ldi r17, 0b00001111
      out DDRB, r17                 ;Assigna PORTB com entrada i sortida

No_polsat:
      sbic PINB, 4                  ;Polsador està al PORTB 4
      rjmp No_polsat

      call delay_10ms
      call delay_10ms
      call delay_10ms

      sbic PINB, 4
      rjmp No_polsat                ;Evitar soroll

vermell:
      sbi PORTB, 3                  ;LED vermell on
      call delay_1s
      call delay_1s
      cbi PORTB, 3                  ;LED vermell off

groc:
      sbi PORTB, 2                  ;LED groc on
      call delay_1s
      cbi PORTB, 2                  ;LED groc off

verd:
      sbi PORTB, 1                  ;LED verd on
      call delay_1s
      call delay_1s
      cbi PORTB, 1                  ;LED verd off
      rjmp vermell
```

### 5.7. Pràctica 4: Introducció a la Programació i Depuració en C

UNIVERSITAT ROVIRA i VIRGILI


# PRÀCTICA 4:

# Introducció a la Programació i Depuració en C

**Assignatura:** Microcontroladors i Sistemes Embedded

**Ensenyament:** Enginyeria de Sistemes i Serveis de Telecomunicacions

**Curs:** 2020/21

## 1. Introducció i Objectius

Els objectius d'aquesta pràctica són:

- Familiarització en la programació i depuració en llenguatge C en l'entorn de desenvolupament Atmel Studio.
- Creació, depuració i simulació d'un projecte en C.
- Familiarització en la manipulació de bits.
- Emulació del primer programa escrit en C sobre la placa de desenvolupament.

## 2. Estudi Previ

1. Tal com es detalla en la Pràctica 0, l'entorn de desenvolupament Atmel Studio, permet la creació de projectes en assemblador o en C. A continuació, es pot observar un breu resum dels passos a seguir per a obtenir un projecte en llenguatge d'alt nivell.

| **TIP** | Creació d'un nou Projecte en C |
|---|---|

En primer lloc, cal seleccionar el tipus de fitxer. En aquest cas, un fitxer C executable:

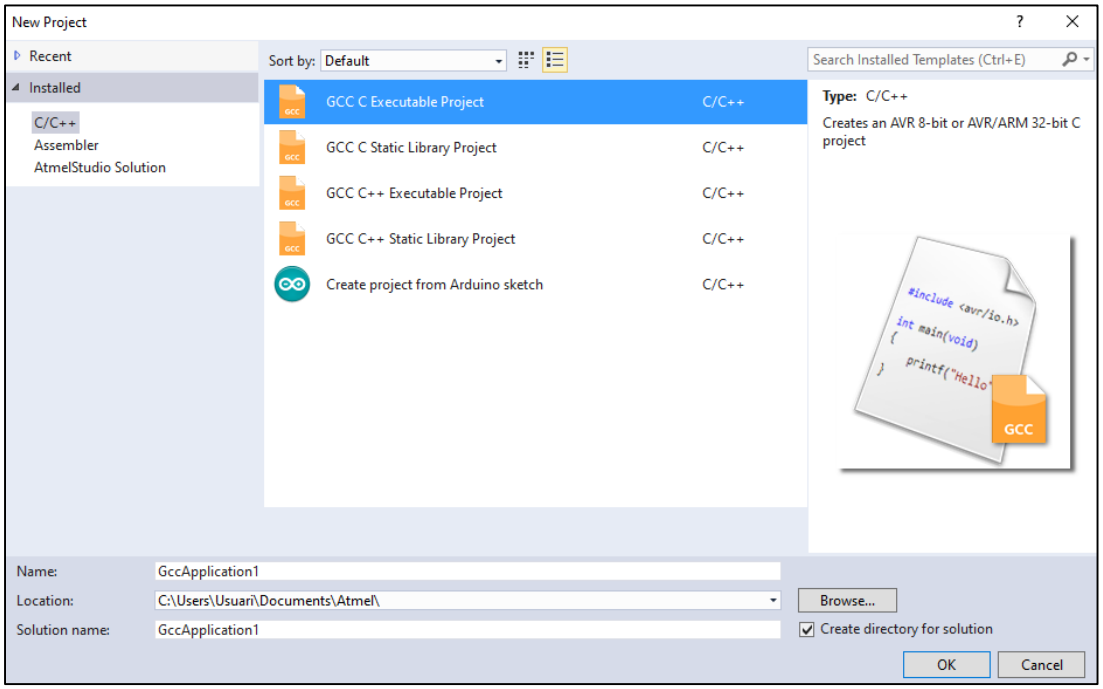



***Figura 1**. Generació d'un nou Projecte amb Llenguatge C.*

Seguidament, cal indicar el nom del projecte i on es vol guardar. Finalment, cal seleccionar el dispositiu utilitzat, l'ATmega328P.

2. Pel que fa a la compilació i depuració, s'executen de la mateixa manera que pel cas d'un projecte assemblador. Així doncs, l'alumne ja està familiaritzat amb les principals característiques. Com per exemple, les funcionalitats que cal fer servir per a generar una solució i depurar-la o establir *breakpoints* en els punts on es vol suspendre l'execució.
   Si bé, durant la depuració d'un projecte de tipus C, hi ha disponibles més opcions de visualització.

| **TIP** | Depuració d'un Projecte en Llenguatge C |
|---|---|

Un cop s'ha generat la solució correctament i es comença a depurar, el programador té a la seva disposició diverses eines per a comprovar l'execució del programa.

D'una banda, l'usuari pot visualitzar variables afegint-les a la finestra Watch. A la que s'hi pot accedir a través de la següent seqüència en la barra d'eines. O bé, posicionant el ratolí sobre la variable, fent clic amb el botó dret i seleccionant Add Watch.

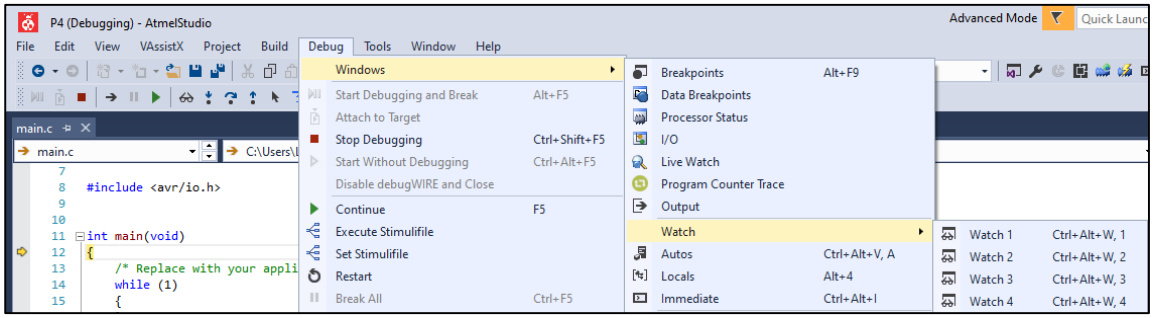



***Figura 2**. Visualització d'una Variable determinada (Watch).*

De l'altra, en mode simulació, també hi ha la possibilitat d'observar el valor de les variables en temps real, és a dir, sense haver d'aturar l'execució. Per a fer-ho, cal fer clic en: Debug – Windows – Live Watch i afegir la variable desitjada.

| **ATENCIÓ** | Projecte en Llenguatge C |
|---|---|

A fi de declarar les definicions apropiades dels ports I/O del dispositiu, a l'inici de qualsevol programa principal, cal incloure la següent línia de codi. La qual inclou la llibreria avr/io dins del projecte.

```
#include <avr/io.h>
```

3. Una altra eina molt útil per a comprovar si es duu a terme l'execució esperada del codi és la visualització de ***Disassembly view*.** Gràcies a aquesta opció, l'usuari pot comprovar com el compilador ha traduït el codi a llenguatge assemblador i verificar que no s'ha produït cap interpretació errònia del llenguatge d'alt nivell.

   Per a accedir-hi, només cal prémer la icona  .

| **TIP** | Configuració del Nivell d'Optimització |
|---|---|

Per a poder estudiar millor la solució generada pel compilador o assegurar una correcta execució pas a pas, cal modificar la configuració d'optimització de codi, desactivant aquesta opció.

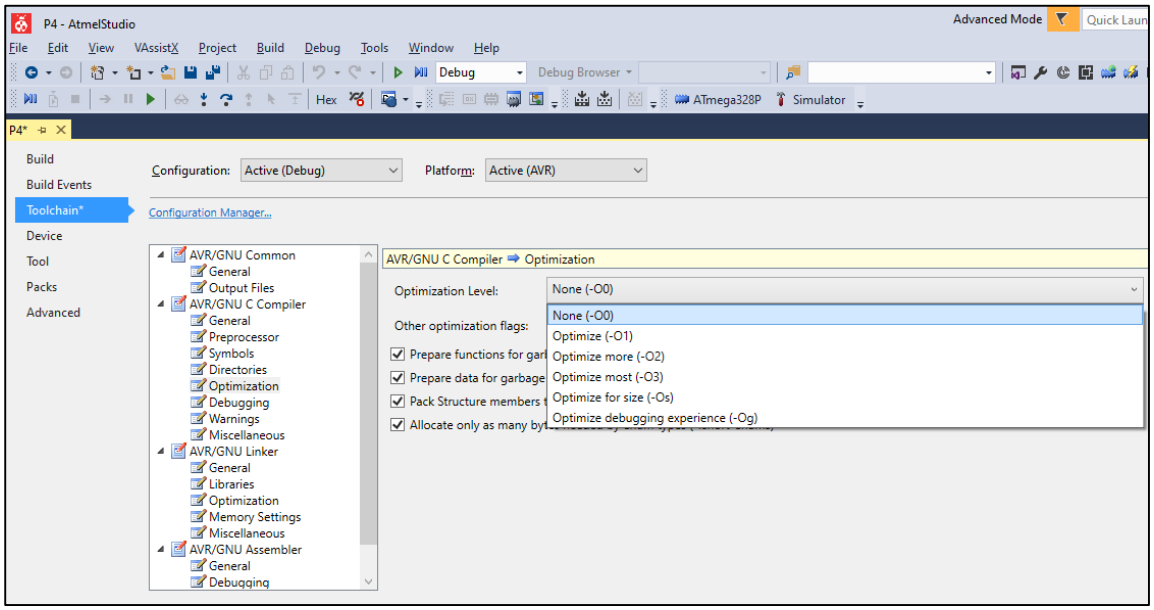



***Figura 3**. Configuració del Nivell d'Optimització.*

Si bé, un cop estudiat el codi i comprovat que tot funciona com s'esperava, es recomana tornar a activar l'opció d'optimització. Per tal d'obtenir un codi més curt i més ràpid.

4. Pel que fa a la programació de microcontroladors, la manipulació de bits individuals és un dels conceptes més importants a entendre. Atès que és necessària per a llegir l'estat dels components, configurar paràmetres i canviar l'estat dels pins de sortida.

| **TIP** | *Bitwise Operations* |
|---|---|

Les operacions bit a bit, generalment més conegudes pel seu nom en anglès, *bitwise operations*, s'utilitzen per a manipular els bits individuals d'un byte sense alterar els altres bits. A continuació, es mostren els diferents tipus d'operacions a nivell de bit, conjuntament amb l'operand corresponent.

| **Funció** | **Operació** | **Operand** | **Comentari** |
|---|---|---|---|
| Activar un bit (1) | OR | \| | Es força a 1 quan un dels operands és un 1. |
| Desactivar un bit (0) | AND | & | Es força a 0 quan un dels operands és un 0. |
| Desplaçament a l'esquerra | | << | Desplaça tots els bits el nombre de posicions indicades i omple els espais buits amb 0.<br>Molt útil en les màscares de bit. |
| Desplaçament a la dreta | | >> | |

***Taula 1**. Operacions bit a bit.*

**ATENCIÓ** Operands de negació – *Bitwise vs Logic*

Convé destacar que en el llenguatge C, es poden diferenciar dos tipus d'operands de negació. En funció de si es tracta d'una operació lògica o a nivell de bit.

D'una banda, la negació lògica avalua el byte sencer i pren el seu operand com un booleà (FALS si és zero o CERT per a qualsevol valor diferent de zero) i retorna 1 o 0 respectivament. Es realitza amb l'operand !.

De l'altra, existeix l'operand bitwise NOT, el qual es representa amb el símbol ~. També s'anomena *bitwise complement*, ja que realitza el complement a 1. És a dir, inverteix, bit a bit, els bits d'un byte i retorna un valor enter. Com es pot observar en el següent exemple:

```
//Bitwise NOT
              res = 0b11110000;
              res = ~res;                 //res = 0b00001111

              res = 0b01110001;
              res = ~res;                 //res = 0b10001110

//Negació lògica
              res = 0b11110000;
              res = !res;                 //res = 0x00

              res = 0b01110001;
              res = !res;                 //res = 0x00

              res = 0xff;
              res = !res;                 //res = 0x00. Atenció pot portar a confusió
```

Com es detalla a continuació, el segon operador és molt útil per a realitzar operacions a nivell de bit. Especialment, per les operacions de desplaçament.

**TIP** Màscares de Bits

S'anomenen màscares de bit les constants que s'utilitzen per a treballar amb les operacions bit a bit. La màscara indica quin és el bit que es vol modificar sense alterar els altres.

Les màscares s'utilitzen conjuntament amb les operacions OR i AND.
Per exemple, en el següent tros de codi les màscares són els nombres hexadecimals 0x01 i 0xFE.

```
PORTB = PORTB | 0x01; //Força a 1 el bit 0 del PORTB
PORTB = PORTB & 0xFE; //Força a 0 el bit 0 del PORTB
```

**ATENCIÓ** *Bitwise Operations* – Operacions de Desplaçament

Cal tindre en compte que existeixen dos tipus diferents d'operacions de desplaçament.

En primer lloc, convé destacar que pel cas del desplaçament a l'esquerra, no existeix diferència entre els dos tipus d'operacions. Atès que els bits que surten per l'esquerra es perden i per l'extrem dret es reomplen amb zeros.

En canvi, pel cas de la rotació a la dreta, sí que hi ha diferència, segons si l'operand té signe o no. Si és del tipus `unsigned`, l'extrem esquerre es reomple amb zeros. Ara bé, si és del tipus `signed`, l'operació és *sign extended*. És a dir, es manté el signe del valor i la correcta codificació en complement a 2.

A fi de comprendre millor cada operació i quan s'utilitza una o altra, cal analitzar les instruccions de llenguatge assemblador que duen a terme aquesta funcionalitat.

D'una banda, hi ha el tipus *Logical Shift*, el qual desplaça tots els bits una posició a la dreta o a l'esquerra. Els bits que surten del registre per l'extrem indicat es perden. Per l'altre extrem, es reomplen amb zeros els bits que han quedat buits a causa del desplaçament.

- LSL (Logical Shift Left): Desplaça tots els bits del registre una posició a l'esquerra. El bit 0 es posa a zero i el bit 7 es guarda a la bandera de Carry del registre d'estat SREG.
  Aquesta operació multiplica per dos valors amb signe o sense.
- LSR (Logical Shift Right): Desplaça tots els bits del registre una posició a la dreta. El bit 7 es posa a zero i el bit 0 es guarda a la C Flag del SREG.
  Aquesta operació divideix una variable sense signe per dos.

De l'altra, hi ha el tipus *Arithmetic Shift*, el qual manté constant el signe de l'operand.

- ASR (Arithmetic Shift Right): Desplaça tots els bits del registre una posició a la dreta. El bit 7, el qual fixa el signe del valor, es manté constant i el bit 0 es guarda a la C Flag.
  Aquesta operació divideix un valor amb signe sense canviar-li el signe.

Si bé, com s'observa en la taula anterior, el llenguatge C només té un operand de desplaçament a la dreta (>>). En conseqüència, l'elecció del tipus d'operació depèn del compilador. La majoria de compiladors implementen el desplaçament a la dreta de nombres enters amb signe amb l'operació aritmètica, la qual manté el signe.

Així doncs, si l'operand és `unsigned`, cal realitzar un desplaçament lògic. El qual reomple els bits buits amb zeros.

Però si és `signed`, es duu a terme un desplaçament aritmètic. En el qual, els bits que queden buits a causa del moviment a la dreta es reomplen amb el valor que tenia el MSB abans de realitzar el desplaçament. D'aquesta manera, es manté el signe de la variable.

| **TIP** | Desplaçament a l'Esquerra |
|---|---|

D'altra banda, l'operació de desplaçament a l'esquerra és molt útil quan es vol treballar directament amb la posició del bit o quan el bit té un nom definit.

A més a més, per a facilitar l'escriptura del codi, es poden fer servir les declaracions (`#define …`) del fitxer de capçalera associat al microcontrolador, on cada port i pin té un nom associat; per exemple, en posar `PORTB |= (1<<PORTB2)` ens estalviem de saber la posició de memòria associada al PORTB i no cal escriure en decimal o hexadecimal la posició del pin 2 del port.

Convé destacar que per a conèixer el nom dels bits i registres del microcontrolador, cal consultar el *datasheet* o el fitxer de capçalera per a cada família.

En conseqüència, el resultat final és més llegible i més fàcil de mantenir, ja que amb un simple cop d'ull, es pot veure a quin bit del registre es fa referència.

Finalment, es mostren dos procediments per a activar el bit 2 del PORTB mitjançant aquesta tècnica. Per un costat, primer cal generar la màscara necessària i després utilitzar-la conjuntament amb una operació OR. En canvi, pel segon cas, directament s'utilitza l'etiqueta del pin del port que es vol manipular.

```
Opció 1:
mask = (0x01 << 2);       //mask = 0x04 o mask = 0b00000100
PORTB = PORTB | mask;    //Bit 2 del PORTB a 1

Opció 2:
PORTB |= (1<<PORTB2);      //El número 1 es desplaça a l'esquerra fins que
arriba al bit anomenat PORTB2.
```

5. Un cop s'ha comprès tota la informació detallada fins al moment, cal analitzar i comentar l'execució esperada del codi del fitxer P4_1.c
   Seguidament, cal simular el projecte i corregir els errors per a obtenir el resultat esperat. Es recomana utilitzar la visualització de variables i la ***Disassembly view***.

**ATENCIÓ** Visualitzar variables

A fi de visualitzar correctament les variables durant la depuració, és recomanable convertir-les en tipus volàtils (`volatile`).

Atès que l'Atmel Studio utilitza un compilador amb opció d'optimització, l'objectiu principal d'aquest és obtenir el codi més eficient possible, és a dir, genera la solució més ràpida i curta que pot. Si bé, aquest avantatge sovint pot generar resultats no esperats. Els més comuns quan se simula o es depura el codi són:

- Si una variable no té un propòsit real, el compilador és prou intel·ligent per no utilitzar-la. En canvi, si se li atribueix alguna tasca concreta, per exemple assignar el seu valor al PORTB, sí que la utilitzarà.
- Amb el mode execució pas a pas, la fletxa groga que indica el punt d'execució salta i no segueix l'ordre seqüencial d'execució normal.
- No compila el retard creat a partir d'un bucle `for()` buit.
- En intentar visualitzar una variable amb la finestra ***Watch***, es mostra un missatge d'error i no es pot monitoritzar el valor d'aquesta.

En efecte, a fi d'obtenir el resultat més ràpid, el compilador descarta les línies innecessàries. Per exemple, les declaracions que no involucren entrades o sortides no són convertides a instruccions d'assemblador.

Una altra funció de l'optimitzador és reconèixer quan simplement pot mantenir una còpia local de la variable en un registre i no perdre temps en llegir i escriure en una adreça de memòria SRAM. En conseqüència, quan la finestra ***Watch*** va a llegir a la memòria SRAM el valor de la variable declarada, no troba la informació que busca i es mostra el missatge d'error.

La solució a aquests problemes és declarar les variables de tipus volàtil. D'aquesta manera, s'indica al compilador que les variables no s'han d'ignorar perquè possiblement estan subjectes a ser utilitzades en altres llocs. Per aquest motiu, sempre que s'indiqui, s'han de llegir o escriure a memòria.

En definitiva, si s'utilitza un compilador amb optimització, és molt important conèixer el tipus volàtil. De tota manera, un cop s'ha depurat el codi i s'ha comprovat que tot funciona correctament, cal eliminar el tipus volàtil, recompilar i comprovar que tot funciona correctament.

6. A partir d'ara, per tal de generar retards, es recomana utilitzar funcions específicament dissenyades per aquest propòsit. Així doncs, cal obrir el fitxer P4_2.c en un nou projecte, comentar el codi i simular-lo.

| **TIP** | Llibreries pels Microcontroladors AVR. |
|---|---|

Com s'ha comentat prèviament en la pràctica 0, l'ús de llibreries és una pràctica molt estesa en la programació de microcontroladors. Si bé, cal saber com treuen el màxim profit.

En primer lloc, convé destacar que gràcies a la plataforma de software lliure GNU, estan a l'abast de tots els usuaris.
En concret, la llibreria més coneguda és l'AVR Libc, la qual proporciona les llibreries estàndard més utilitzades en codi C.

Hi ha dues maneres de consultar quines són les llibreries disponibles. D'una banda, es pot accedir als arxius de capçalera a través de la següent adreça:
"C:\Program Files (x86)\ Atmel\Studio\7.0\toolchain\avr8\avr8-gnu-toolchain\avr\include"

O bé, a través del següent enllaç web. En el qual, es detallen les funcionalitats de les funcions i punts a tenir en compte quan s'utilitzen.
https://www.nongnu.org/avr-libc/user-manual/modules.html

En aquest cas, per a poder generar bucles d'espera mitjançant una llibreria, cal utilitzar la llibreria delay.h de la següent manera. Primer, a l'inici del codi, cal incloure les línies:

```
#define F_CPU 16000000    //Establir la freqüència de treball a 16 MHz
#include <util/delay.h>   //Incloure la llibreria delay.h
```

Seguidament, quan es necessita introduir un retard, cal escriure una de les dues crides a funció:

```
_delay_ms(500);   //Retard en ms
_delay_us();      //Retard en us
```

7. Cal canviar la tècnica de manipulació de bits, ja que la utilitzada en el codi font proporcionat en l'exercici anterior (P4_2.c) pot induir a errors més fàcilment. Cal comprovar que amb les modificacions s'obté el mateix resultat final.

8. A fi de posar en pràctica tots els coneixements apresos fins ara, cal adaptar la pràctica de la gestió de l'efecte rebot d'un polsador, realitzada anteriorment en assemblador, a llenguatge C.

   En aquest cas, durant l'estat d'acció, cal controlar el LED del kit de la següent manera: començant amb el LED apagat, cada vegada que es premi el polsador, cal canviar l'estat del LED.

| **TIP** | Resistència Pull-up dels Pins |
|---|---|

Com s'ha comentat anteriorment, cada port es gestiona amb tres registres ubicats en l'espai de memòria I/O.
A més, com es pot observar en la següent figura, cada pin té una resistència pull-up interna i díodes de protecció, tant a terra com a $V_{cc}$.

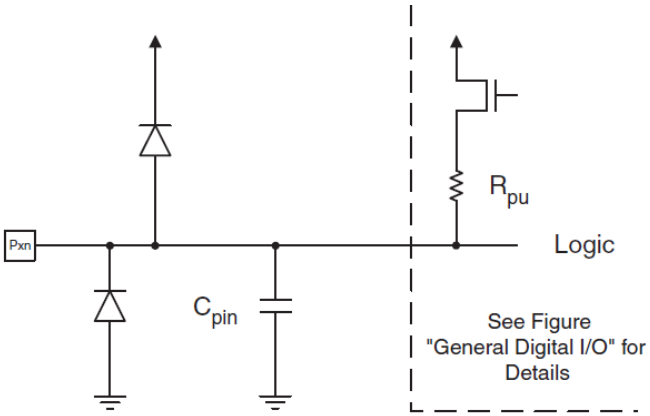



***Figura 4**. Esquemàtic d'un Pin d'Entrada i Sortida*

Atès que hi ha diversos modes d'operació, cal veure les diferents opcions de configuració dels pins. D'una banda, per a activar la resistència $R_{pu}$ cal configurar el pin com a input i cal escriure un 1 al PORTxn.
De l'altra, per a desactivar aquesta resistència, es pot fer de dues maneres, configurant el pin com una sortida o posant a 0 el PORTxn.

Tal com s'indica en el *datasheet* del microcontrolador, pel cas dels pins configurats com a entrada que no s'utilitzin, és recomanable que tinguin un nivell definit. De manera que cal habilitar la resistència $R_{pu}$.

## 3. Treball al Laboratori

1. Cal emular sobre la placa de desenvolupament els dos algorismes de control del LED realitzats en l'estudi previ. S'obté el mateix resultat tot i utilitzar tècniques de manipulació de bits diferents?

2. Cal emular l'algorisme de la gestió de l'efecte rebot sobre la placa de desenvolupament.

3. Cal dissenyar una solució que mostri en els LED del mòdul extern la representació binària del nombre de cops que s'ha premut el polsador. Quan se supera el valor màxim que els tres LED poden mostrar, cal que el bit de menys pes utilitzat faci pampallugues tres cops.

4. Finalment, cal emular el codi del comptador binari sobre la placa per tal de comprovar la correcta implementació de la solució.

# 4. Solució Esperada

## 4.1. Estudi Previ

5.

A continuació, s'indica l'execució esperada per a cada línia de codi.

```
#include <avr/io.h>

int main(void)
{
            char mask = 0b00000001; //Es declara la variable mask
            char res = 0x00;        //Es declara la variable res

            DDRB = 0xff;       //PORTB com a sortida
            PORTB = 0x00;      //Inicialitza el PORTB a 0

      while (1)
      {

            PORTB = PORTB | 0x08;    //Força a 1 el bit 3 del PORTB
            PORTB = PORTB & 0xF7;    //Força a 0 el bit 3 del PORTB

            res = res | mask;  //Força a 1 el bit 0 de la variable res
            mask = 0xfe;       //mask = 11111110
            res = res & mask;  //Força a 0 el bit 0 de la variable res

            //Utilitzant la tècnica de desplaçament a l'esquerra
            PORTB |= (1<<PORTB0);    //Força a 1 el bit 0 del PORTB
            PORTB |= (1<<PORTB1);    //Força a 1 el bit 1 del PORTB
            PORTB &= ~(1<<PORTB0); //Força a 0 el bit 0 del PORTB.
                                     Deixant a 1 el bit 1
            mask = (0x01 << 2);      // Màscara amb el bit 2 activat;
                                     mask = 0x04
            PORTB = PORTB | mask;    //Força a 1 el bit 3 del PORTB

      }

   }
```

No obstant això, quan se simula el codi, no s'obtenen els resultats esperats.

D'una banda, les instruccions que treballen amb el registre PORTB del microcontrolador sí que generen el resultat esperat. Per exemple, després d'executar-se la primera línia del bucle, s'observa que efectivament s'ha activat el bit 3 del PORTB:

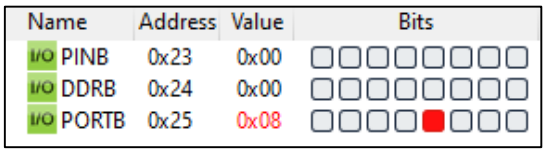



***Figura 5**. Detall del Resultat del Programa.*

De l'altra, les línies de codi que contenen una de les dues variables declarades a l'inici del programa no s'executen. És a dir, la simulació se les salta com si no hi fossin.

Per tal d'esbrinar què està passant, cal obrir la finestra ***Disassembly view***:

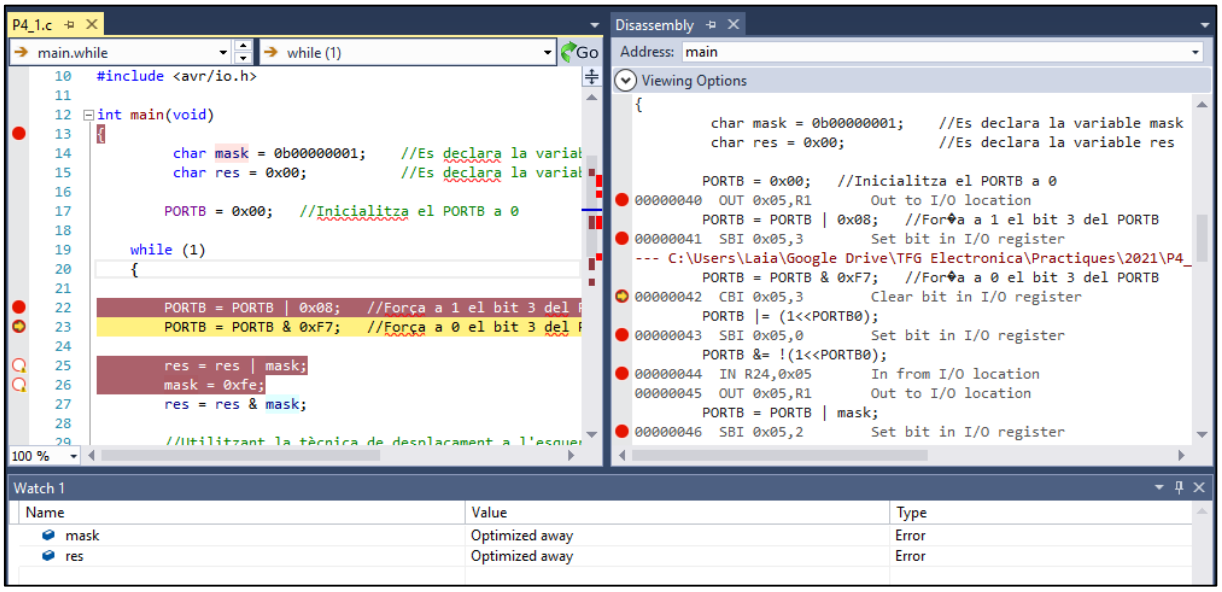



***Figura 6**. Detall Disassembly View.*

En la Figura 6, es pot observar com les dues línies de codi on es declaren i s'inicialitzen les variables `mask` i `res`, no es tradueixen a cap instrucció d'assemblador. Per aquest motiu, en la part inferior de la imatge, es pot veure com la visualització de variables indica un error en aquestes dues.

A més, per les dues declaracions de les línies 25 i 26 tampoc s'ha generat cap codi en assemblador. Atès que si s'observa la finestra Disassembly, es veuen les línies de codi font sense cap instrucció que les correspongui.

Com se sap, aquest fet és a causa que pel compilador aquestes instruccions no tenen cap finalitat útil pel projecte, ja que no alteren l'estat de cap sortida ni llegeixen cap entrada.

Com es pot veure en la següent imatge, a fi de solucionar aquest problema, només cal declarar aquestes dues variables de tipus volàtil. Les instruccions ressaltades de color gris, duen a terme la correcta declaració i inicialització de les dues variables, per a, posteriorment, ser utilitzades segons convingui.

Per exemple, en les dues últimes línies de codi, primer s'activa el bit 2 de la variable mask. Seguidament, s'utilitza aquest valor per a fer una operació bit a bit de tipus OR per a activar el bit 2 del PORTB.

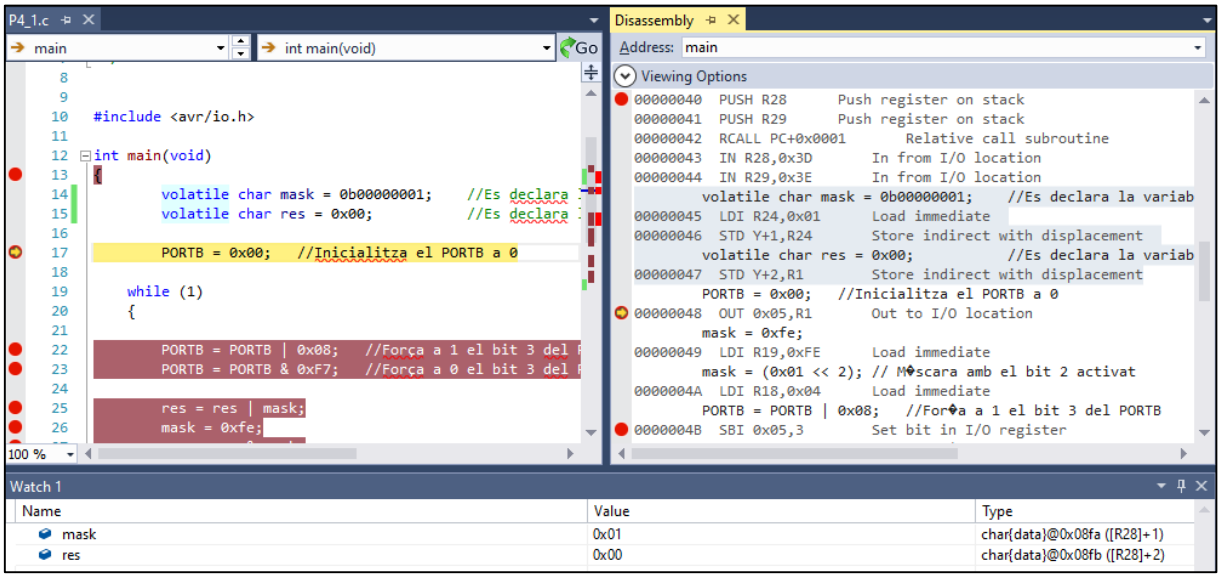



***Figura 7**. Detall del Disassembly View i de la Finestra Watch.*

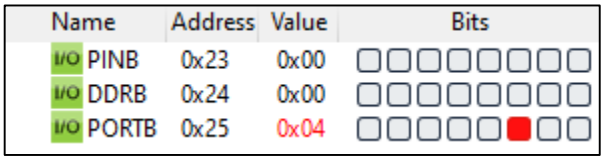



***Figura 8**. Resultat final del Codi.*

Finalment, pel que fa a la comprovació de la implementació del desplaçament a la dreta, cal observar la finestra Disassembly per a determinar com s'executen les següents línies:

```
//COMPROVACIÓ DE LA IMPLEMENTACIÓ DEL DESPLAÇAMENT A LA DRETA

//Sense especificació del signe de l'enter
            volatile int output = 0;
            volatile int valor = 0x2f;
            output = valor >>2;

            output = 0;
            valor = -10;
            output = valor >> 1;
            output = valor >> 3;

            //Variable unsigned
            volatile unsigned int test = 10;
            test = test >>1;

            test = -10;
            test = test >>1;

            //Variable signed
            volatile signed int resultat = -20;
            resultat = resultat >>1;

            resultat = 20;
            resultat = resultat >> 2;
```

En primer lloc, les línies de codi que treballen amb les variables que no s'especifica si són amb signe o no, es tradueixen a llenguatge assemblador de la següent manera:

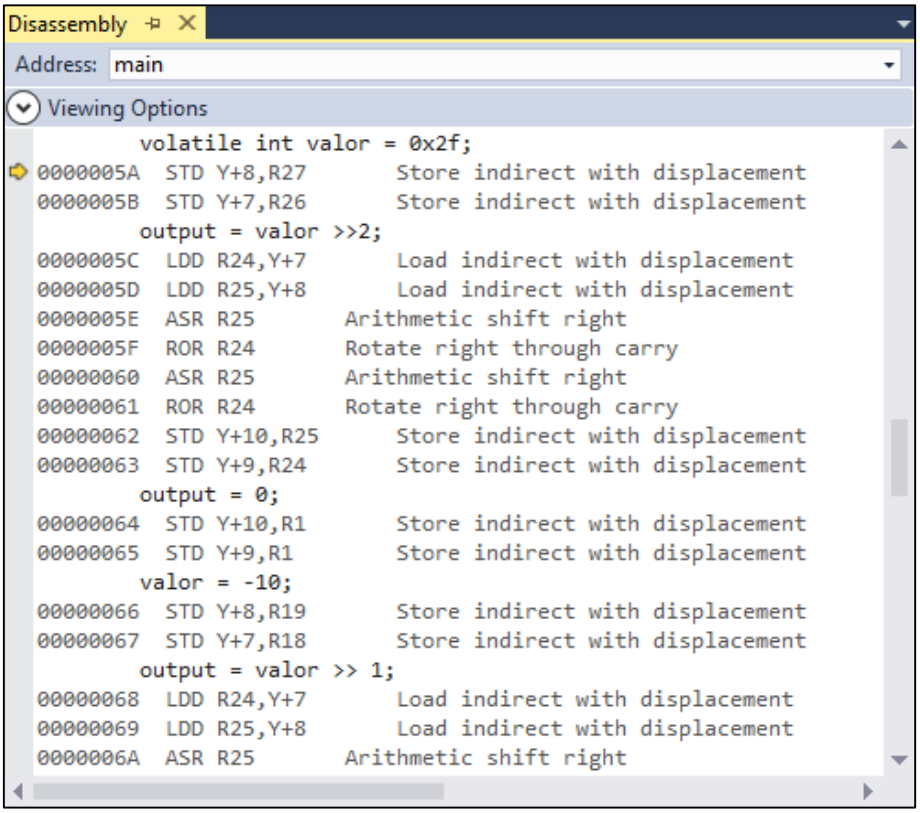



***Figura 9**. Detall del Disassembly de la primera Comprovació del Desplaçament a la Dreta.*

En la figura anterior, es pot observar que els desplaçaments es realitzen amb el desplaçament aritmètic, és a dir, es manté el signe del valor.

En concret, després d'executar la línia: `output = valor >> 1`, el valor de les variables és el següent. Efectivament, s'ha mantingut el signe de la variable output.

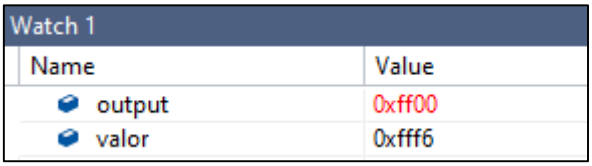



***Figura 10**. Finestra Watch de la primera Comprovació del Desplaçament a la Dreta.*

En segon lloc, si s'estudien les declaracions que utilitzen variables sense signe, aquestes són traduïdes a llenguatge assemblador com a desplaçaments lògics. Evidentment, el resultat obtingut en desplaçar a la dreta una posició el valor -10 és erroni. Atès que el compilador espera rebre un valor sense signe i rep un valor negatiu.

Com es pot observar en la següent figura, no s'ha mantingut el signe de la variable test.

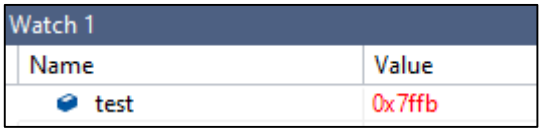



***Figura 11**. Finestra Watch de la segona Comprovació del Desplaçament a la Dreta.*

En tercer lloc, pel cas de la variable resultat definida explícitament amb signe, el compilador tradueix a llenguatge assemblador els dos desplaçaments a la dreta amb la instrucció ASR.

En definitiva, quan es desplaça a la dreta un valor amb signe, es manté el signe. Tant utilitzant variables explícitament declarades amb signe, com sense indicar-ho.

6.

```
#include <avr/io.h>
#define F_CPU 16000000      //Establir la freqüència de treball a 16 MHz
#include <util/delay.h>     //Incloure la llibreria delay.h

int main(void)
{
      DDRB = 0xff;      //PORTB com a sortida

    while (1)
    {
            PORTB |= 0x20;    //LED on

            _delay_ms(500);   //Retard en ms

            PORTB &= 0xdf;    //LED off

            _delay_ms(500);   //Retard en ms
    }
}
```

Com es pot veure a l'inici del codi, per tal d'utilitzar la llibreria delay.h, prèviament declarada, cal indicar la freqüència de treball de la CPU. Com se sap, tenint en compte el kit de desenvolupament utilitzat en les presents pràctiques, la freqüència és de 16 MHz.

L'objectiu d'aquest codi és ben senzill, encendre i apagar el LED de la placa, ja que les operacions bit a bit manipulen el bit 5 del PORTB, el qual es correspon amb el LED.

7.

En aquest cas, a fi de controlar les pampallugues del LED de la placa, s'utilitza la tècnica de manipulació de bits mitjançant l'operació de desplaçament a l'esquerra.

Així doncs el codi resultant és el següent:

```
#include <avr/io.h>
#define F_CPU 16000000      //Establir la freqüència de treball a 16 MHz
#include <util/delay.h>     //Incloure la llibreria delay.h

int main(void)
{
      DDRB = 0xff;      //PORTB com a sortida

    while (1)
    {
            PORTB |= (1<<PORTB5);   //LED on

            _delay_ms(500);         //Retard en ms

            PORTB &= ~(1<<PORTB5);  //LED off
```

```
            _delay_ms(500);          //Retard en ms
    }
}
```

Evidentment, si s'observa la finestra ***I/O***, s'obté el mateix resultat per a les dues solucions plantejades.

8.

En primer lloc, per tal de dur a terme el procés de programació correctament, cal recuperar el diagrama de flux dissenyat per aquesta aplicació.

Si bé, en aquest cas, per tal de comprovar la correcta implementació de l'efecte rebot, cal implementar les següents etapes en l'estat d'acció.

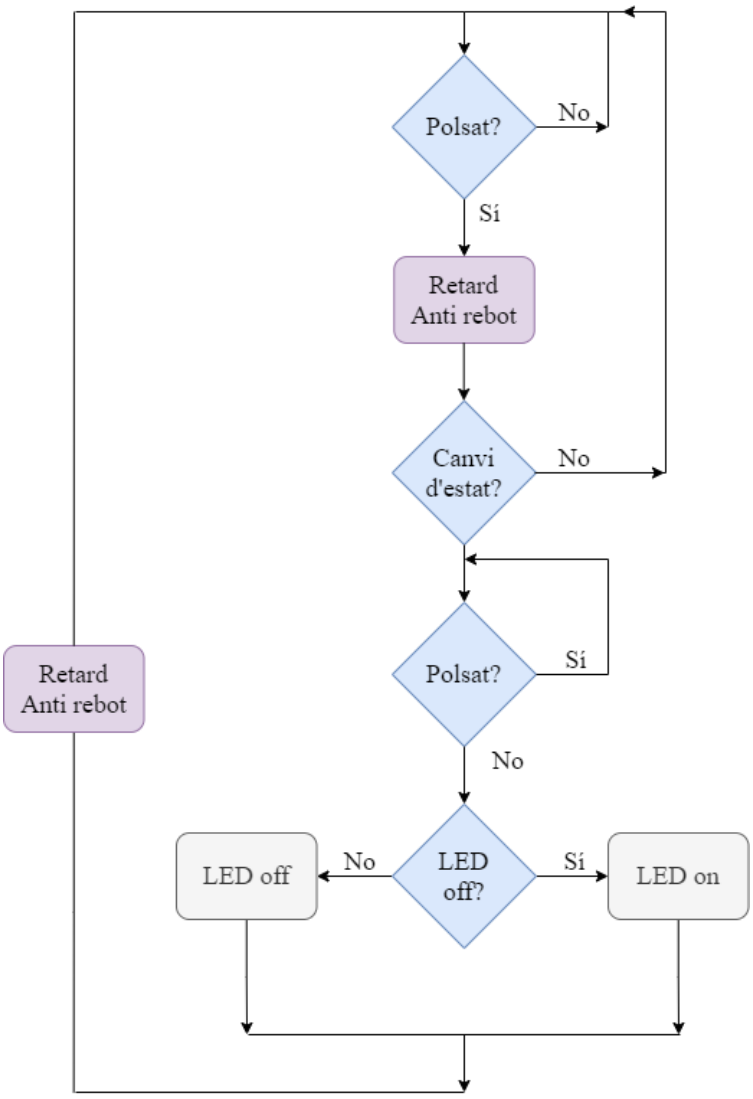



***Figura 12***. *Diagrama de Flux del Control de l'Activació d'un Polsador.*

Així doncs, seguint el diagrama de flux de la figura anterior i utilitzant el polsador de la placa de desenvolupament, el qual es monitoritza a través del pin 7 del PORTB, s'obté el següent algorisme:

```
#include <avr/io.h>
#define F_CPU 16000000      //Establir la freqüència de treball a 16 MHz
#include <util/delay.h>     //Incloure la llibreria delay.h

volatile char LED_on = 0;

int main(void)
{
      DDRB &= 0x7f;           //PORTB7 com a entrada
      DDRB |= 0x20;           //PORTB5 com a sortida LED
      PORTB = PORTB & 0x7F;   //Polsador amb Pull-up deshabilitat

      PORTB &= ~(1<<PORTB5);  //LED off

    while (1)
    {
            if ((PINB & 0b10000000) == 0b00000000)    //Si polsador tancat
            {
                  _delay_ms(30);    //Retard antirebot
                  if ((PINB & 0b10000000) == 0b00000000)
                  {
                        while ((PINB & 0b10000000) == 0b00000000);
      //Espera fins que no polsat
                        if (LED_on)
                        {
                              PORTB &= ~(1<<PORTB5);  //LED off
                              LED_on = 0;
                        }
                        else
                        {
                              PORTB |= (1<<PORTB5);   //LED on
                              LED_on = 1;
                        }

                  }
                  _delay_ms(30);    //Retard antirebot
            }
    }
}
```

### 4.2. Treball al Laboratori

1.

Efectivament, en l'emular les dues opcions sobre el kit de desenvolupament, el resultat obtingut és el mateix. El LED de la placa fa pampallugues a la mateixa freqüència.

2.

Altre cop, en emular el codi simulat en el software Atmel Studio, s'ha obtingut el resultat esperat. Començant amb el LED apagat, cada vegada que es prem el polsador, el LED canvia d'estat.

A més, per tal de comprovar la correcta gestió de l'efecte rebot, es prem el polsador en intervals molt curts i la resposta del LED és l'esperada.

3.

En primer lloc, a fi de dur a terme correctament el desenvolupament de la solució, cal dibuixar el diagrama de flux.

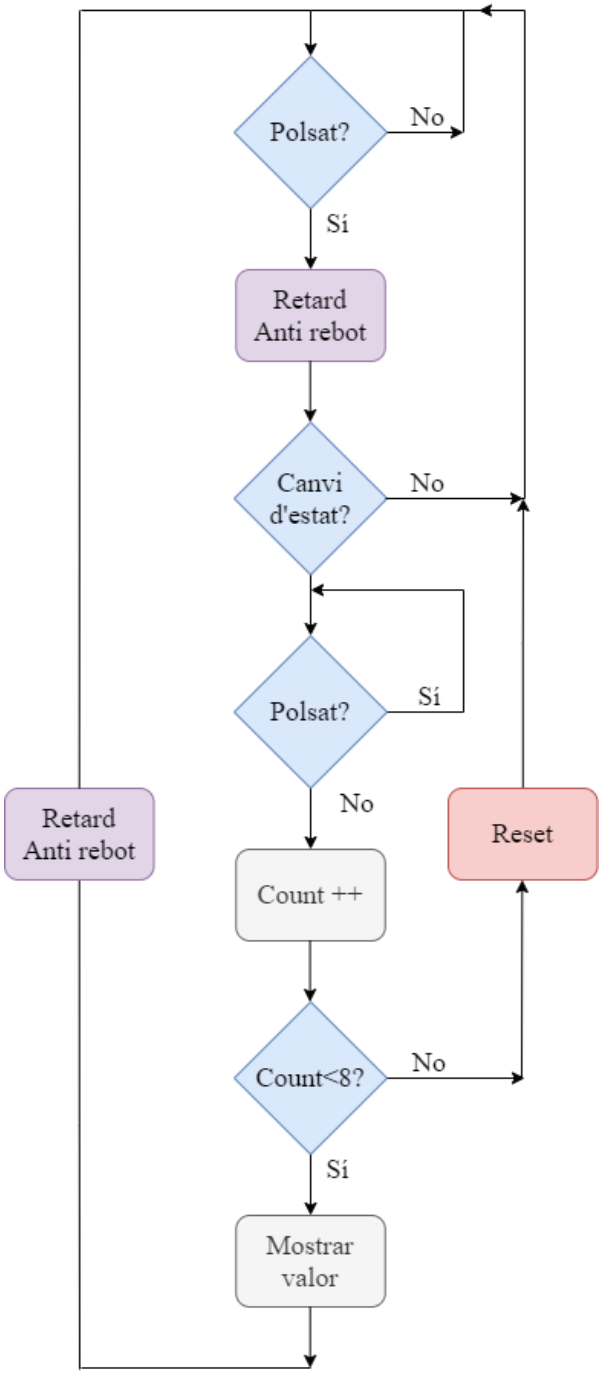



***Figura 13**. Diagrama de Flux del Comptador Binari.*

Seguidament, cal escriure el codi que compleixi les tasques i condicions especificades en el diagrama.

```
#include <avr/io.h>
#define F_CPU 16000000      //Establir la freqüència de treball a 16 MHz
#include <util/delay.h>     //Incloure la llibreria delay.h


int main(void)
{
      volatile char count = 9;

      DDRB &= 0xef;           //PORTB4 com a entrada
      DDRB |= 0x0e;           //Pins 1, 2 i 3 del PORTB com a sortida LED
      PORTB = PORTB & 0xef;   //Polsador amb Pull-up deshabilitat

      PORTB &= ~((1<<PORTB1)|(1<<PORTB2)|(1<<PORTB3));      //LEDs off

      while (1)
      {
            if ((PINB & 0b00010000) == 0b00000000)    //Polsador tancat
            {
                  _delay_ms(30);    //Retard antirebot
                  if ((PINB & 0b00010000) == 0b00000000)
      //Evita falses activacions per soroll
                  {
                        while ((PINB & 0b00010000) == 0b00000000);
      //Espera fins que no polsat
                        count++;

                        if (count<8)
                        {
                              PORTB = (count<<1);
            //Mostra el valor pels LEDs
                        }
                        else
      //Reset
                        {
                              count = 0;
                              PORTB &=
~((1<<PORTB1)|(1<<PORTB2)|(1<<PORTB3));   //LEDs off
                                    for (int i =0; i<3; i++)
                                    {
                                    _delay_ms(500);
                                    PORTB |= (1<<PORTB1);   //LED on
                                    _delay_ms(500);
                                    PORTB &= ~(1<<PORTB1);  //LED off
                                    }
                        }
                  }
                  _delay_ms(30);    //Retard antirebot
            }
      }
}
```

4.

Primer, se simula el codi i es comprova que tot funciona correctament.

Finalment, s'emula el projecte sobre la placa de desenvolupament i es constata que l'aplicació compleix amb tots els requeriments.

### 5.8. Pràctica 5: Control LCD

UNIVERSITAT ROVIRA i VIRGILI


# PRÀCTICA 5:

# Control LCD

**Assignatura:** Microcontroladors i Sistemes Embedded

**Ensenyament:** Enginyeria de Sistemes i Serveis de Telecomunicacions

**Curs:** 2020/21

## 1. Introducció i Objectius

Els objectius d'aquesta pràctica són:

- L'objectiu principal d'aquesta pràctica és que l'alumne adquireixi l'habilitat d'utilitzar una pantalla de cristall líquid, generalment coneguda com a LCD, de la sigla en anglès de *Liquid Crystal Display*.
- Familiarització en la utilització de funcions en C.
- Familiarització en la consulta de manuals de dispositius electrònics.

## 2. Estudi Previ

1. En primer lloc, primer cal conèixer què és una pantalla LCD. Així, a continuació, es realitza una breu introducció teòrica sobre aquest dispositiu.

**Informació** Pantalla LCD

Tal com el seu nom indica, una LCD és una pantalla plana de cristall formada per un nombre de píxels, en color o en blanc i negre, ubicats davant d'una font de llum o reflectora.

Aquest dispositiu té múltiples aplicacions: des de rellotges digitals o pantalles en petits electrodomèstics, a interfícies d'usuari de màquines industrials o pantalles de televisió. En conseqüència, en els darrers anys, s'ha perfeccionat aquesta tecnologia fins a obtenir productes de gran qualitat i definició.

En concret, gràcies a les seves principals característiques, el seu ús està molt estès en els sistemes encastats. Atès que es poden construir en petites dimensions i consumeixen molt poca energia. Essent aquest un factor important a considerar en les aplicacions que funcionen amb bateries.

Els cristalls líquids alfanumèrics es classifiquen segons el nombre de caràcters i línies que poden representar. Les configuracions més habituals són: 8x1 o 16x2. A més, la mida dels caràcters també pot variar: 5x7 o 5x10 píxels.

En efecte, permeten una configuració i un control relativament senzills a través de 14 pins. O, pel cas de les pantalles retroil·luminades, 16 pins. Com es detalla a continuació, generalment, presenten dos modes d'interfície disponibles, 4 o 8 bits.

Concretament, durant les pràctiques al laboratori, s'utilitza el controlador HITACHI 44780, el qual és el més utilitzat del mercat. Convé destacar que el seu manual està penjat al Moodle de l'assignatura i es recomana la seva consulta.

2. Durant la present pràctica de l'assignatura, es treballa amb la pantalla LCD HITACHI 44780 com a perifèric. Si bé, primer cal conèixer el seu funcionament i la configuració necessària.

**Informació** Pantalla LCD HITACHI 44780 – Característiques Principals

Les principals característiques de la pantalla LCD HITACHI 44780 són:

- La pantalla utilitzada porta integrat un controlador, el qual es pot configurar per a realitzar el control amb un bus de dades de 4 o 8 bits. Durant les presents pràctiques es treballa amb una interfície de 4 bits. De manera que s'utilitzen menys pins del microcontrolador. Si bé, la configuració i comunicació resulta més complexa.
- Funcionament a un baix consum d'energia. De 2,7 a 5,5 V.
- Disposa d'un joc d'instruccions definit i complet.

**Informació** Pantalla LCD HITACHI 44780 – Pins

La comunicació entre el microcontrolador principal i el xip del controlador de la pantalla es realitza a través de 14 pins, els quals es detallen a continuació.

D'esquerra a dreta, la funcionalitat dels pins és:

| Pin | Senyal | Funció |
|---|---|---|
| 1 | GND | Terra |
| 2 | Vcc | Alimentació a 2,7 V – 5,5 V |
| 3 | Vo | Ajust del contrast de la pantalla. Entrada analògica generalment connectada a un potenciòmetre. |
| 4 | RS | *Register Select*. Permet seleccionar el tipus d'informació que s'envia. Si RS = 1, s'envien dades. Si RS = 0, instruccions. |
| 5 | RW | Si RW = 1, lectura. Si RW = 0, escriptura. Generalment, la lectura de la pantalla no té sentit. Per això, es pot connectar aquest pin permanentment a terra i no cal controlar-lo. |
| 6 | E | Senyal d'*Enable* disparat pel flanc de baixada. |
| 7 | Bit 0 | Bus de dades. No utilitzat en l'operació a 4 bits. |
| 8 | Bit 1 | Bus de dades. No utilitzat en l'operació a 4 bits. |
| 9 | Bit 2 | Bus de dades. No utilitzat en l'operació a 4 bits. |
| 10 | Bit 3 | Bus de dades. No utilitzat en l'operació a 4 bits. |
| 11 | Bit 4 | Bus de dades. |
| 12 | Bit 5 | Bus de dades. |
| 13 | Bit 6 | Bus de dades. |
| 14 | Bit 7 | Bus de dades. |

***Taula 1**. Funcionalitat dels Pins del LCD Hitachi 44780.*

**Informació** Pantalla LCD HITACHI 44780 – Pinout

En la següent imatge es poden observar els pins utilitzats en les presents pràctiques:

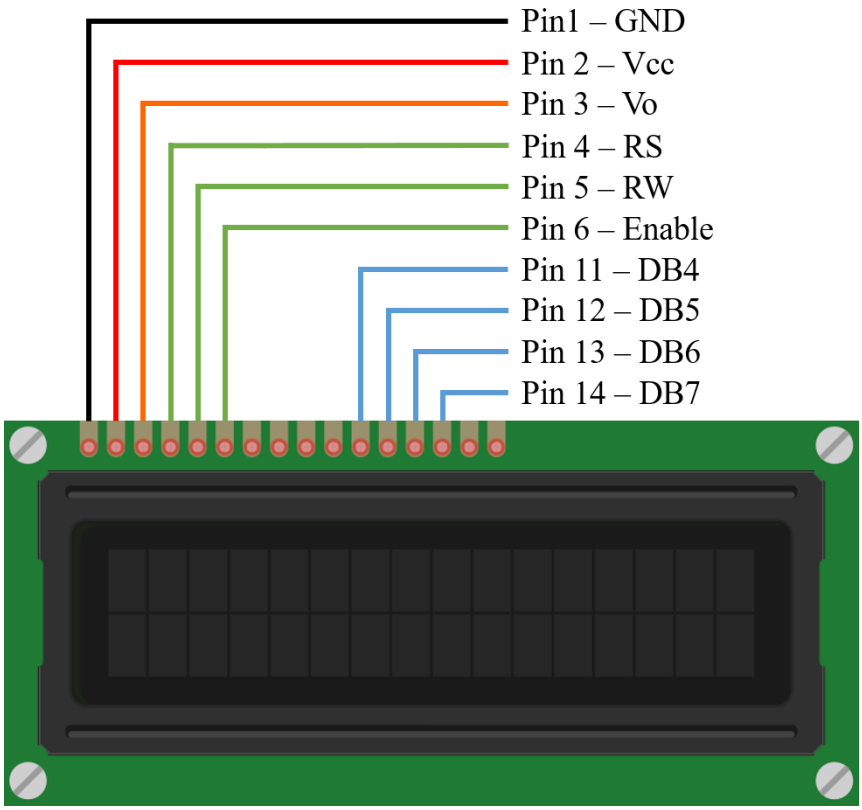



***Figura 1**. Pinout LCD HITACHI 44780.*

**TIP** Pantalla LCD HITACHI 44780 – Interfície amb el controlador

Com s'ha mencionat anteriorment, en les presents pràctiques només es treballa amb la interfície de 4 bits.

Així doncs, només cal utilitzar 4 bits del bus de comunicació, del DB4 al DB7. Si bé, com que les dades que s'envien ocupen 8 bits, cal realitzar dues operacions de 4 bits per a realitzar la transmissió correctament.

A més, cal tindre en compte l'ordre d'enviament. Primer, cal enviar els 4 bits de més pes i després els 4 bits de menys pes. És a dir, la comunicació es realitza amb parells de 4 bits, anomenats *Nibbles*, i es necessiten dos cicles.

| **TIP** | Pantalla LCD HITACHI 44780 – Registres |
|---|---|

Mitjançant el senyal RS, controlat a través del pin 4, l'usuari pot seleccionar el tipus d'informació que s'envia per la interfície de comunicació.

Concretament, activant o desactivant aquest pin, se selecciona, respectivament, un d'aquests dos registres de 8 bits:

D'una banda, el DR, de la sigla en anglès de *Data Register*. El qual emmagatzema temporalment les dades a escriure en la pantalla.

De l'altra, el IR, de la sigla en anglès de *Instruction Register*. El qual emmagatzema els codis d'instruccions.

En conseqüència, el controlador del LCD sap com ha de tractar les dades que rep en funció del senyal RS: Dades a mostrar per la pantalla, o bé comandes a executar.

| **TIP** | Pantalla LCD HITACHI 44780 – Joc d'Instruccions |
|---|---|

A continuació, es presenta una taula resum (Taula 2) amb les principals instruccions disponibles per a controlar la pantalla.
Convé destacar, que abans d'escriure les instruccions, cal posar a 0 el senyal RS. D'aquesta manera, el controlador sap que les dades rebudes són comandes a executar. Evidentment, cal tindre habilitat el mode d'escriptura.

A més, pel cas del codi proporcionat a l'alumne, s'ha creat la següent funció per a facilitar l'escriptura de funcions:

```
void LCDWrite_Instruccio (char data);
```

Així, només cal passar per paràmetre el codi de la instrucció que es vol executar.

Si bé, cal tindre en compte que, tal com es detalla més endavant, fins a l'execució de la primera instrucció del bloc 3, el controlador del LCD està configurat per a treballar amb una interfície de comunicació de 8 bits. Consegüentment, cal utilitzar la funció:

```
LCDWriteNibbleI(char data);
```

L'objectiu principal d'aquesta és l'escriptura d'una instrucció de 4 bits. Més endavant s'especifica amb més detall.

| **TIP** | Pantalla LCD HITACHI 44780 – Inicialització |
|---|---|

Convé destacar que cada vegada que s'alimenta la pantalla, cal realitzar una inicialització per tal de configurar-la segons les necessitats de l'usuari.

Concretament, tal com s'indica en el manual, cal seguir la seqüència d'instruccions que es mostra en la Figura 2.

En primer lloc, després d'alimentar el dispositiu, cal esperar més de 40 ms per començar a enviar-li instruccions.

En el bloc 2, s'envien les instruccions de *reset*. Si bé, primer, cal tindre en compte que cada vegada que s'activa el LCD està configurat per defecte per a treballar amb una interfície de 8 bits. En conseqüència, les primeres escriptures es realitzen amb l'ample de bus de 8 bits. És a dir, cada vegada que el senyal *Enable* té un flac de baixada, el controlador llegeix els 8 pins de dades (DB0 – DB7). No obstant això, la configuració inicial del LCD estableix que els 4 bits de menys pes són ignorats durant la fase inicial del procés d'inicialització. D'aquesta manera, cal escriure tres vegades la instrucció *Function Set*, configurada a 8 bits. En concret, cal enviar el següent codi d'instrucció: 0011****. Essent en format hexadecimal: 0x30.

Seguidament, en el bloc 3, s'envien les instruccions de configuració. Fixant primer el tipus d'interfície utilitzada, en aquest cas 4 bits.
A partir d'aquí, el controlador realitza la lectura del bus de dades en parells de *nibbles*. És a dir, per cada senyal d'*Enable*, llegeix els quatre bits de més pes.
A continuació, cal enviar quatre instruccions de configuració mes.

Per acabar, un cop finalitzada la inicialització, és recomanable deixar el display LCD encès, amb el mode del cursor i pampallugues segons els requeriments de la solució. Utilitzant novament la instrucció Display on/off.

Tanmateix, durant qualsevol procés d'enviament d'informació, siguin instruccions o dades a mostrar, cal tindre en compte els retards necessaris per a assegurar que el controlador ha finalitzat l'execució de la instrucció anterior.
A més, com es pot observar en la taula 2, el fabricant del LCD ha estipulat el temps d'execució de cada una de les instruccions principals.

| Instrucció | Codi | | | | | | | | | | Descripció | $t_{execució}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | RS | RW | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | | |
| Clear display | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | Esborra la pantalla i retorna el cursor a la posició de home, és a dir, a la cantonada superior esquerra. A més, estableix a 1 el paràmetre I/D. | 1,52 ms |
| Cursor home | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | * | Retorna el cursor a la posició de home. A més, si el desplaçament (*shift*) està activat, torna la pantalla a la posició original. | 1,52 ms |
| Entry Mode Set | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | I/D | S | Estableix la direcció de moviment del cursor (I/D). I especifica si hi ha desplaçament o no (S). | 37 µs |
| Display on/off | 0 | 0 | 0 | 0 | 0 | 0 | 1 | D | C | B | Controla l'activació o desactivació del display (D), del cursor (C) i les pampallugues del caràcter o cursor (B). | 37 µs |
| Cursor/display shift | 0 | 0 | 0 | 0 | 0 | 1 | S/C | R/L | * | * | Estableix el moviment del cursor o del desplaçament del display (S/C). Fixa la direcció del desplaçament (R/L). Sense llegir o escriure dades a la pantalla. | 37 µs |
| Function Set | 0 | 0 | 0 | 0 | 1 | DL | N | F | * | * | Estableix la mida de la interfície de dades(DL), el nombre de línies del LCD(N) i el tipus de caràcters (F). | 37 µs |

- I/D = 0: Decrement de la posició del cursor // =1: Increment de la posició del cursor
- S = 0: No hi ha desplaçament de la pantalla LCD // =1: Desplaçament de la pantalla LCD
- D = 0: Pantalla LCD off // = 1: Pantalla LCD on
- C = 0: Cursor off // =1: Cursor on
- B = 0: Pampallugues off // = 1: Pampallugues on
- S/C = 0: Moviment del cursor // =1: Desplaçament del display
- R/L = 0: Desplaçament esquerra // = 1: Desplaçament dreta
- DL = 0: Interfície 4-bits // = 1: Interfície 8-bits
- N = 0: 1 línia // = 1: 2 línies
- F = 0: Mida caràcter 5x8 píxels // =1: Mida caràcter 5x10 píxels

***Taula 2**. Resum Joc d'Instruccions LCD.*

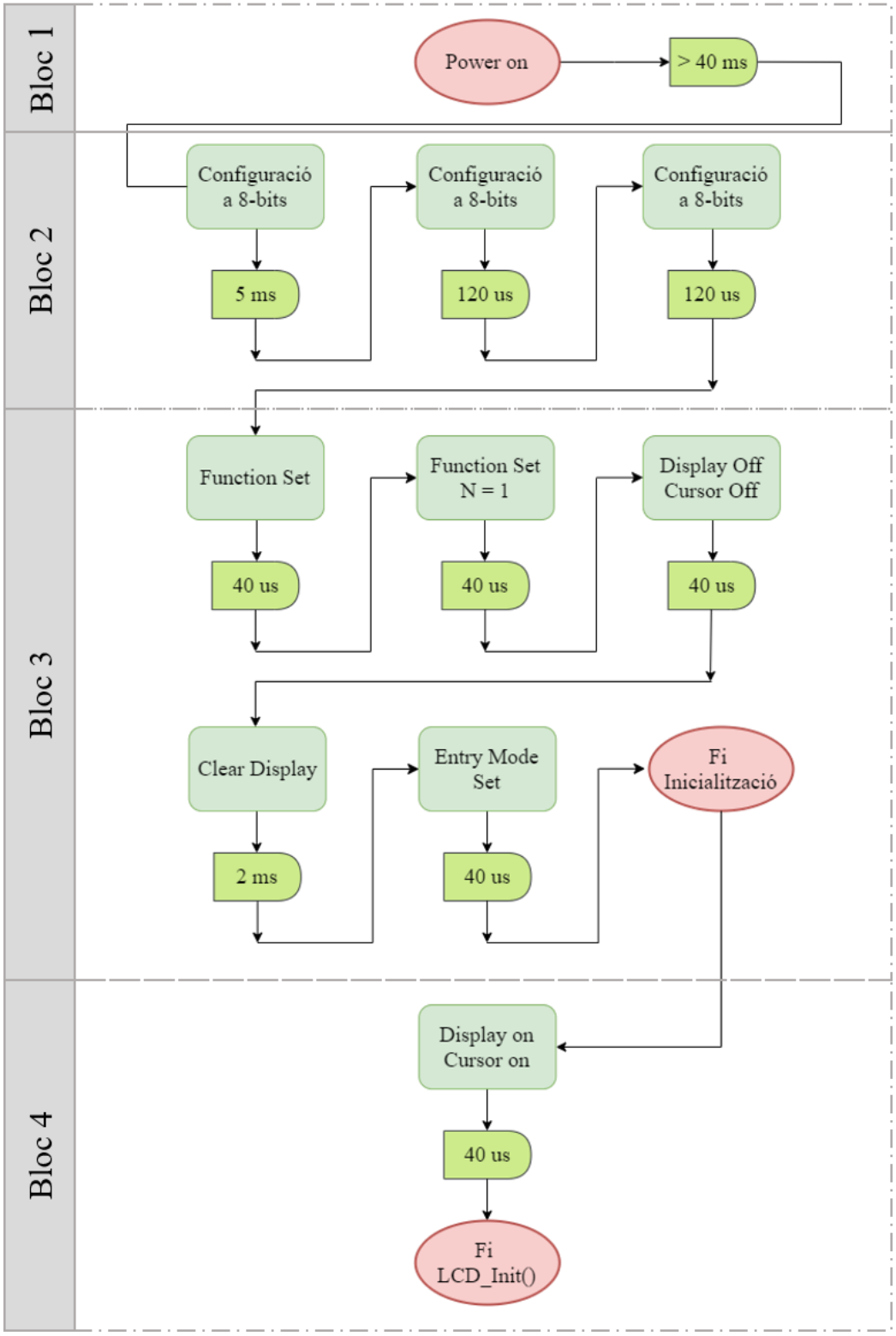



***Figura 2**. Diagrama de flux de la Inicialització del LCD Hitachi HD44780.*

3. A fi de posar en pràctica els coneixements teòrics exposats anteriorment, cal crear un nou projecte d'Atmel Studio amb el fitxer P5_LCD.c i afegir-hi el fitxer de capçalera Include.h.
   En primer lloc, cal explicar a quins pins dels ports del microcontrolador Atmega328P es connecten els pins del display LCD.
   En segon lloc, cal comentar el codi de la funció `int LCD_Init()`.

4. Per tal de simular la funció LCD_Init, cal passar-li per paràmetre el tipus d'interfície desitjada. Com és conegut, en les presents pràctiques, s'utilitza un bus de dades de 4 bits. Així, cal escriure el següent *main* en el projecte.

```
int main(void)
{
	int error = FALSE;

	while (1)
	{
		error = LCD_Init(4);
		//Escriptura en el LCD
	}
}
```

   Cal comentar el resultat obtingut en la simulació i justificar si es correspon amb el resultat esperat explicat en la teoria.

5. Com s'ha constatat durant la simulació, la funció `int LCD_Init()` crida a altres funcions. La funció fonamental d'aquesta aplicació és la `LCDWriteNibbleI(char data)`. Atès que és l'encarregada de la gestió de l'escriptura d'instruccions al controlador.
   Cal simular aquesta funció pas a pas i comentar-ne el funcionament.

**TIP** Funció `LCDWriteNibbleI (char data)`.

L'objectiu principal d'aquesta funció és realitzar l'escriptura de la instrucció indicada a través del paràmetre.

Si bé, convé destacar que aquesta funció és cridada directament en la fase inicial de configuració o per la funció `LCDWrite_Instruccio (char data)`. La qual és l'encarregada de descompondre el codi d'una instrucció en parells de Nibbles. Primer enviant els 4 *Most Significant Bits* i després els 4 bits de menys pes.
L'usuari utilitza la funció `LCDWrite_Instruccio()` per a enviar les instruccions necessàries durant el funcionament de la pantalla.

A continuació, es mostra el diagrama de flux d'aquesta funció. El qual s'ha extret del diagrama de temps de l'operació d'escriptura, en el manual del dispositiu.

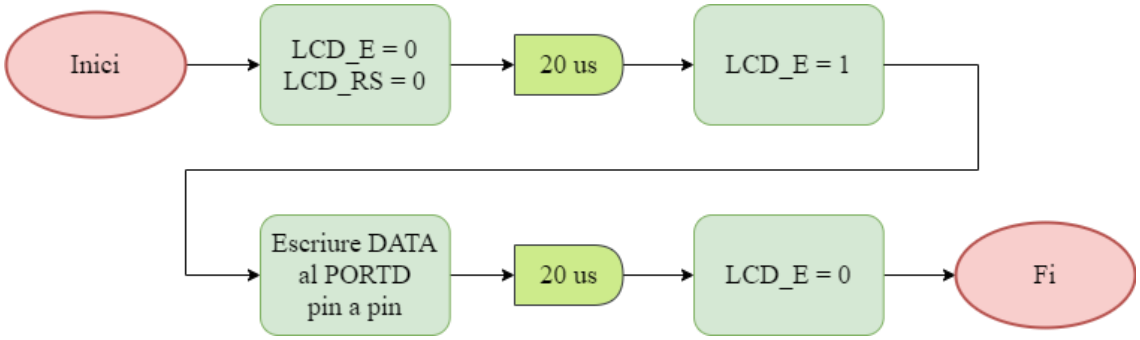



***Figura 3**. Diagrama de Flux de la Funció LCDWriteNibbleI (char data).*

6. A continuació, cal modificar la funció estudiada anteriorment per a desenvolupar la funció `LCDWriteNibbleD(char lletra)`. L'objectiu de la qual és escriure el caràcter que es vol mostrar per pantalla.

## 3. Treball al Laboratori

1. Cal emular la funció que inicialitza la pantalla LCD i comprovar que l'estat final del display és amb el cursor a la primera posició, és a dir, a la cantonada superior esquerra, fent pampallugues.

2. Cal escriure el codi necessari, utilitzant les funcions estudiades, per a poder mostrar per la pantalla el missatge "HELLO".

3. Com s'ha comentat anteriorment, l'ús de funcions és una tècnica molt estesa, ja que permet dividir un programa extens en petits segments que executen tasques concretes. D'aquesta manera, es millora la llegibilitat del programa i el seu manteniment.
   Així, cal crear una llibreria amb les funcions d'usuari estudiades durant la present pràctica per a controlar la pantalla LCD.

| **TIP** | Funcions en Llenguatge C |
|---|---|

El disseny de codi C es basa en la utilització de funcions. Per aquesta raó, l'usuari ha de conèixer com emprar-les correctament.

Primer, cal declarar la funció, és a dir, cal escriure el prototip de la funció especificant el nom, els paràmetres de la funció i el tipus de dada que retorna. Normalment, es declaren en la capçalera del programa i tenen la següent forma:

```
tipo_de_retorn nom_de_la_funció (llista_de_paràmetres);
```

En segon lloc, cal implementar la funció, és a dir, cal escriure la seva definició especificant les instruccions necessàries per a assolir l'objectiu d'aquesta.

Amb la finalitat de crear llibreries amb les funcions d'usuari desenvolupades, se separen les declaracions de les definicions en dos fitxers diferents.

D'una banda, en el fitxer de capçalera .h, es declaren els prototips de les funcions. De l'altra, les descripcions de les funcions s'escriuen en un fitxer amb el mateix nom que el *header*, però amb extensió .c.

Així, quan l'usuari vol utilitzar alguna de les funcions de la llibreria creada, ha d'especificar el fitxer de capçalera que conté el prototip d'aquesta, a l'inici del programa principal.

---

4. Cal recuperar el diagrama de flux del comptador binari utilitzat anteriorment, el qual es mostra en la Figura 13 de la pràctica anterior. Tot i que en aquest cas, cal mostrar a la vegada el valor en binari pels tres LEDs de la placa i, també, el valor en decimal per la pantalla LCD.

   En la fase de Reset, cal apagar els LEDs i que l'últim LED faci pampallugues 3 cops. A més, cal esborrar la pantalla, mostrar la paraula "END" i, finalment, tornar a deixar la pantalla només amb el cursor.

# 4. Solució Esperada

## 4.1. Estudi Previ

3.

Per tal de conèixer a quins pins es connecten els senyals necessaris de la pantalla LCD HITACHI 44780 cal consultar el fitxer Include.h o la documentació del mòdul extern.

Per un costat, si es consulta el fitxer de capçalera Include.h hi ha les següents declaracions. Gràcies a les quals, l'usuari pot deduir a quin pin del microcontrolador està connectat cada senyal de la pantalla a través de l'IDC 16.

```
#define LCD_D4    PORTD0
#define LCD_D5    PORTD1
#define LCD_D6    PORTD2
#define LCD_D7    PORTD3

#define LCD_RS    PORTD4
#define LCD_E     PORTD6
```

Per l'altre, si s'observa conjuntament la Figura 1 del present document, la qual mostra el *pinout* del LCD, i l'esquemàtic del mòdul d'expansió, s'arriba a la conclusió que:

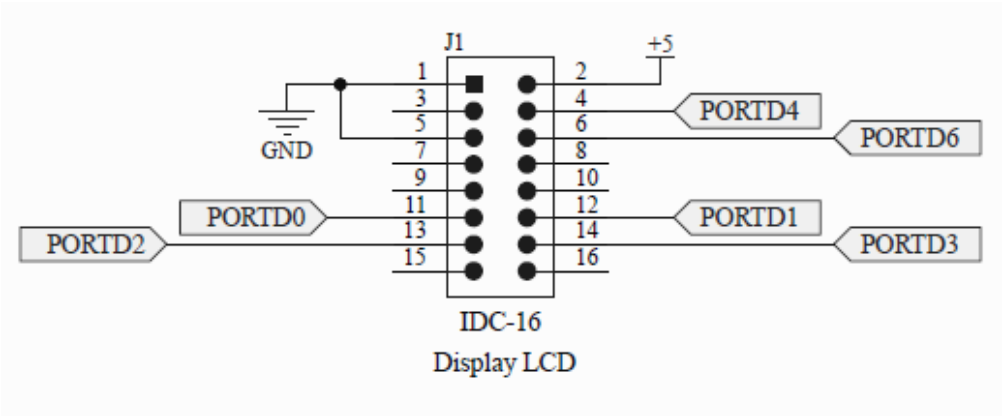



*Figura 4. Detall de l'Esquemàtic del Mòdul Extern.*

| Pin LCD | Senyal | Port Microcontrolador |
|---|---|---|
| Pin 1 | GND | GND |
| Pin 2 | $V_{cc}$ | $V_{cc}$ = 5 V |
| Pin 3 | $V_O$ | Not Connected |
| Pin 4 | RS | PORTD, pin 4 |
| Pin 5 | RW | GND |
| Pin 6 | Enable | PORTD, pin 6 |
| Pin 7 – Pin 10 | Not used | Not Connected |
| Pin 11 | DB4 | PORTD, pin 0 |
| Pin 12 | DB5 | PORTD, pin 1 |
| Pin 13 | DB6 | PORTD, pin 2 |
| Pin 14 | DB7 | PORTD, pin 3 |

***Figura 5**. Correspondència dels Pins del LCD amb els Pins del Microcontrolador.*

D'altra banda, la funció encarregada d'inicialitzar la pantalla LCD és:

```
int LCD_Init(int DataLenght)
{
	if (DataLenght == 4)    //Segons especificacions en l'enunciat,
només es treballa amb la interfície de 4 bits
	{
		DDRD = 0xff;			//PORTD com a sortida
		PORTD = 0xff;			//Per defecte, PORTD a 1

		_delay_ms(50);			//Delay després de Power-on. Wait for
more than 40 ms after VCC rises to 2.7 V

		//*****************
		//Inicialització LCD

		//Function Set: Establir la interfície de bits, 4 o 8

		LCDWriteNibbleI(0x30);		//Cal enviar 3 cops la
configuració a 8-bit
		_delay_ms(5);
		LCDWriteNibbleI(0x30);
		_delay_us(120);
		LCDWriteNibbleI(0x30);
		_delay_us(120);                                  //Fi del Bloc
2
```

```
            LCDWriteNibbleI(FUNCTION_SET);            //Interfície 4 bits,
1 fila, Format 5x8
            _delay_us(40);                                    //Fins aquí
encara anem a 8bit

            LCDWrite_Instruccio(LINES_2);       //Function Set: 2 linies,
N = 1
            _delay_us(40);

            LCDWrite_Instruccio(DISPLAY_OFF_CURSOR_OFF);
            _delay_us(40);

            LCDWrite_Instruccio(CLEAR_DISPLAY);
            _delay_ms(2);

            LCDWrite_Instruccio(ENTRY_MODE_SET);
            _delay_us(40);

            //Inicialització completa. Fi del bloc 3
            //Display on

            LCDWrite_Instruccio(DISPLAY_ON_CURSOR_ON);
            _delay_us(40);

            return FALSE;           //Per indicar que el procés
d'inicialització s'ha realitzat amb èxit
      }

      else              //ERROR
      {
            return TRUE;
      }
}
```

4.

Gràcies a la simulació de la funció `LCDWrite_Instruccio()`, es pot constatar que es realitza la seqüència d'instruccions indicada en el diagrama de flux de la Figura 2.

Per exemple, quan la simulació arriba a l'etapa de fi d'inicialització, s'ha acabat de cridar la funció `LCDWrite_Instruccio(char data);` amb el paràmetre `ENTRY_MODE_SET.` El qual, segons la Taula 2, es correspon amb el codi d'instrucció:

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | I/D | S | (I/D = 1) i (S = 0) |

Amb la direcció del cursor cap a la dreta ( I/D = 1) i sense desplaçament (S = 0). Per tant, l'últim valor enviat pel bus de dades ha de ser 0x06.

Abans d'analitzar el resultat, convé recordar que es treballa amb una interfície de 4 bits. En conseqüència, les dades s'envien en parells de Nibbles. Així, quan s'estudia el bus de dades, només cal fixar-se en els bits DB4 – DB7. Essent en aquest cas, els pins del 0 al 3 del PORTD.

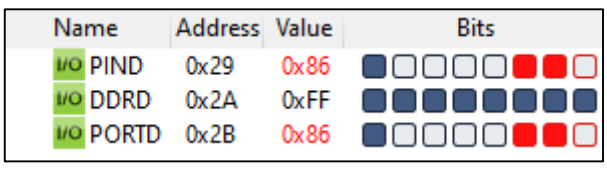



***Figura 6.*** *Resultat de la Simulació del Fi de la Inicialització del LCD.*

Com es pot observar en la Figura 6, els bits 1 i 2 estan activats. Obtenint un valor 0b0110, és a dir, 6.

En definitiva, s'obté el resultat esperat segons la teoria estudiada prèviament.

5.

Durant la simulació de la funció `LCDWriteNibbleI()`, s'ha observat que, efectivament, es realitza la seqüència indicada en el diagrama de flux de la Figura 3. Per a manipular els bits necessaris, s'utilitza l'operació de desplaçament a l'esquerra conjuntament amb l'etiqueta del bit en qüestió.

En concret, si s'analitza la fase d'escriure la variable DATA al PORTD, s'observa que cal manipular bit a bit els bits del bus de dades, és a dir, del pin 0 al 3 del PORTD.

6.

A fi d'obtenir una funció que enviï dades en lloc d'instruccions, tal com s'indica en la Taula 1 sobre la funcionalitat dels pins del LCD, cal forçar el senyal RS a 1. D'aquesta manera, el controlador del perifèric sap que la informació rebuda a través del bus de dades l'ha de tractar com a dades a mostrar per la pantalla.

En conseqüència, només cal modificar la segona i tercera línia de la funció:

```
	PORTD &= ~((1<<LCD_E)); //Clear E
	PORTD |= (1<<LCD_RS);   //Set RS. Per a enviar dades i no
instruccions
```

A continuació, es mostra el codi de la funció `LCDWriteNibbleD()` obtingut:

```
void LCDWriteNibbleD(char lletra)
{
	int volatile pin;
	PORTD &= ~((1<<LCD_E));			//Clear E
	PORTD |= (1<<LCD_RS);			//Set RS. Per a enviar dades i
no instruccions
	delay_us(20);				//Small delay
	PORTD |= (1<<LCD_E);			//Set E

	//Escriu lletra al PORTD
	//LCD_D7
	pin = 0b10000000;
	if (lletra & pin)
	{
		PORTD |= (1<<LCD_D7);
	}
	else
	{
		PORTD &= ~(1<<LCD_D7);
	}

	//LCD_D6
	pin = 0b01000000;
	if (lletra & pin)
	{
		PORTD |= (1<<LCD_D6);
	}
```

```
        else
        {
                PORTD &= ~(1<<LCD_D6);
        }

        //LCD_D5

        pin = 0b00100000;
        if (lletra & pin)
        {
                PORTD |= (1<<LCD_D5);
        }
        else
        {
                PORTD &= ~(1<<LCD_D5);
        }

        //LCD_D4
        pin = 0b00010000;
        if (lletra & pin)
        {
                PORTD |= (1<<LCD_D4);
        }
        else
        {
                PORTD &= ~(1<<LCD_D4);
        }

        _delay_us(20);                  //Small delay
        PORTD &= ~(1<<LCD_E);           //Clear E
}
```

Com es pot observar, l'activació del senyal d'*Enable*, marca l'inici de la transferència de les dades.

De manera paral·lela, cal tindre en compte que la funció `void LCDWrite_Dada (char cadena[])` crida a la funció `void Escriu_Lletra (char lletra)`. La qual, en el moment necessari de la seva execució, crida a la funció `LCDWriteNibbleD()`.

Així doncs, en primer lloc, cal recordar que a causa d'utilitzar una interfície de 4 bits, cal enviar la informació pel bus de dades amb parells de *nibbles*. Per aquest motiu, la funció `Escriu_Lletra()` primer passa per paràmetre a la funció `LCDWriteNibbleD()` els 4 MSB i després els LSB.

En segon lloc, el paràmetre de la funció `LCDWrite_Dada()` és el que conté la informació a mostrar per pantalla. Concretament, cal passar-li una cadena de tipus `char` amb les lletres en codi ASCII.

A més, com es pot deduir a partir de la condició `if` de fins del bucle `for`, l'usuari ha d'indicar el fi de la paraula a través del caràcter de final de cadena: '\0'. Aquest caràcter simbolitza el fi de la cadena de caràcters. D'aquesta manera, el codi sap quan ha de parar d'escriure.

### 4.2. Treball al Laboratori

1.

Efectivament, quan s'emula el projecte al kit de desenvolupament amb el display connectat a través de l'IDC, el resultat obtingut és que el cursor fa pampallugues a la primera posició de la pantalla.

2.

Per a mostrar per pantalla la paraula "HELLO", cal escriure el següent *main*, utilitzant les funcions estudiades durant la present pràctica.

A més, amb l'objectiu de fer saber a l'aplicació el final de la paraula, cal escriure el caràcter '\0'.

```
int main(void)
{
	int error = FALSE;
	char paraula[16];

	while (1)
	{
		error = LCD_Init(4);
		if (!error)
		{
			//Escriptura en el LCD
			paraula[0] = 0x48;
			paraula[1] = 0x45;
			paraula[2] = 0x4C;
			paraula[3] = 0x4C;
			paraula[4] = 0x4F;
			paraula[5] = '\0';

			LCDWrite_Dada(paraula);
			_delay_ms(2000);
			LCDWrite_Instruccio(CLEAR_DISPLAY);
			_delay_ms(2000);
		}
	}
}
```

A fi d'utilitzar més coneixements, amb les tres últimes línies de codi, s'aconsegueix que la paraula HELLO faci pampallugues amb intervals de 2 segons.

3.

Seguint el consell sobre la utilització de funcions en el llenguatge C, s'han creat els fitxers LCD.h i LCD.c. Els quals, respectivament, contenen les declaracions i definicions de totes les funcions estudiades necessàries per a gestionar correctament el display LCD.

4.

Així doncs, utilitzant el diagrama de flux del comptador binari de la pràctica anterior i la llibreria creada anteriorment amb les funcions per a gestionar el display LCD, la solució per a satisfer les especificacions plantejades en l'enunciat és:

```
#include <avr/io.h>
#define F_CPU 16000000      //Establir la freqüència de treball a 16 MHz
#include <util/delay.h>     //Incloure la llibreria delay.h
#include "Include.h"
#include "LCD.h"


int main(void)
{
      volatile char count = 0;
      int error = FALSE;
      volatile char caracter = 0;
      char paraula [16];

      DDRB &= 0xef;           //PORTB4 com a entrada
      DDRB |= 0x0e;           //Pins 1, 2 i 3 del PORTB com a sortida LED
      PORTB = PORTB & 0xef;   //Polsador amb Pull-up deshabilitat

      PORTB &= ~((1<<PORTB1)|(1<<PORTB2)|(1<<PORTB3));      //LEDs off
      error = LCD_Init(4);
      LCDWrite_Instruccio(BLINK_OFF);


      while (!error)
      {
            if ((PINB & 0b00010000) == 0b00000000)    //Polsador tancat
            {
                  _delay_ms(30);    //Retard antirebot
                  if ((PINB & 0b00010000) == 0b00000000)
      //Evita falses activacions per soroll
                  {
                        while ((PINB & 0b00010000) == 0b00000000);
      //Espera fins que no polsat
                        count++;

                        if (count<8)
                        {
                              PORTB = (count<<1);
            //Mostra el valor pels LEDs

                              caracter = Convertir_Caracter(count);
                              paraula[0] = caracter;
                              paraula[1] = '\0';

                              LCDWrite_Dada(paraula);

                              LCDWrite_Instruccio(CURSOR_HOME);

                              _delay_ms(2);
                        }
                        else
      //Reset
                        {
                              count = 0;
                              PORTB &=
~((1<<PORTB1)|(1<<PORTB2)|(1<<PORTB3));  //LEDs off
                              LCDWrite_Instruccio(CLEAR_DISPLAY);
                              _delay_ms(2);
                                          //Mostra per pantalla END
                              paraula[0] = 0x45;
                              paraula[1] = 0x4e;
```

```
                                    paraula[2] = 0x44;
                                    paraula[3] = '\0';

                                    LCDWrite_Dada(paraula);

                                    for (int i =0; i<3; i++)
                                    {
                                          _delay_ms(500);
                                          PORTB |= (1<<PORTB1);   //LED on
                                          _delay_ms(500);
                                          PORTB &= ~(1<<PORTB1);  //LED off
                                    }


                                    _delay_ms(3000);
                                    LCDWrite_Instruccio(CLEAR_DISPLAY);
                                    _delay_ms(2);
                              }
                        }
                        _delay_ms(30);    //Retard antirebot
                  }
            }
}
```

# 6. Conclusions

Finalment, en el darrer punt del treball de fi de grau s'exposen les conclusions a què s'ha arribat i que formen part d'aquest.

Així doncs, d'acord amb l'objectiu principal del present projecte, es conclou afirmant que amb la realització de les pràctiques dissenyades, l'alumne assolirà les competències enumerades en la guia docent.

Amb aquesta finalitat, s'han creat un total de sis pràctiques que permeten a l'estudiant adquirir, progressivament, les habilitats necessàries per a treballar amb els sistemes *embedded*. Així, les primeres pràctiques estan formulades per a utilitzar el llenguatge assemblador. El qual, en aquest cas, té un objectiu didàctic, ja que gràcies a treballar a baix nivell, el programador ha de comprendre millor l'arquitectura del processador per a poder dissenyar la solució.

Un cop adquirits els coneixements sobre les funcionalitats principals, s'evoluciona a treballar amb llenguatge C. El qual, actualment, és el més emprat en aplicacions de sistemes encastats.

A més, per tal de poder treballar de manera òptima amb els perifèrics necessaris, s'ha creat una *shield board* que compleix amb tots els requeriments fixats. D'aquesta manera, també s'ha treballat el procés de muntatge i soldadura d'una placa de circuit imprès.

Per acabar, personalment considero que gràcies a la realització del present projecte, he consolidat els coneixements que ja tenia sobre els microcontroladors. Atès que amb la voluntat de formular uns enunciats els més entenedors i acurats possibles, he estudiat l'arquitectura i el joc d'instruccions de l'ATmega328P i m'he familiaritzat amb l'entorn de desenvolupament pertinent. A fi de poder ordenar els aspectes clau i desenvolupar el conjunt de pràctiques que satisfan les necessitats anteriorment exposades.

# 7. Bibliografia


[1] https://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061B.pdf

[2] N. Cañelles, Tema 2: La Màquina Senzilla, Tarragona: Diapositives de l'assignatura d'Electrònica Digital del Grau en Enginyeria Electrònica Industrial i Automàtica a la URV, 2015.

[3] J.L. Ramírez, Introducción, Tarragona: Diapositives de l'assignatura de Microcontroladors del Grau en Enginyeria Electrònica Industrial i Automàtica a la URV, 2016.

[4] M. Jiménez, R. Palomera i I. Couvertier, Introduction to Embedded Systems: Using Microcontrollers and the MSP430, EEUU: Springer, 2014.

[5] http://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061B.pdf

[6] Microchip Technology Incorporated, ATmega48A/PA/88A/PA/168A/PA/328/P Data Sheet, EEUU: Microchip Technology Inc., 2020.

[7] https://microchipdeveloper.com/8avr:avrcore

[8] http://ww1.microchip.com/downloads/en/devicedoc/atmel-0856-avr-instruction-set-manual.pdf

[9] https://www.sparkfun.com/products/retired/11665?_ga=2.89517661.798630531.1629966232-1311065752.1627207735

[10] https://fritzing.org/

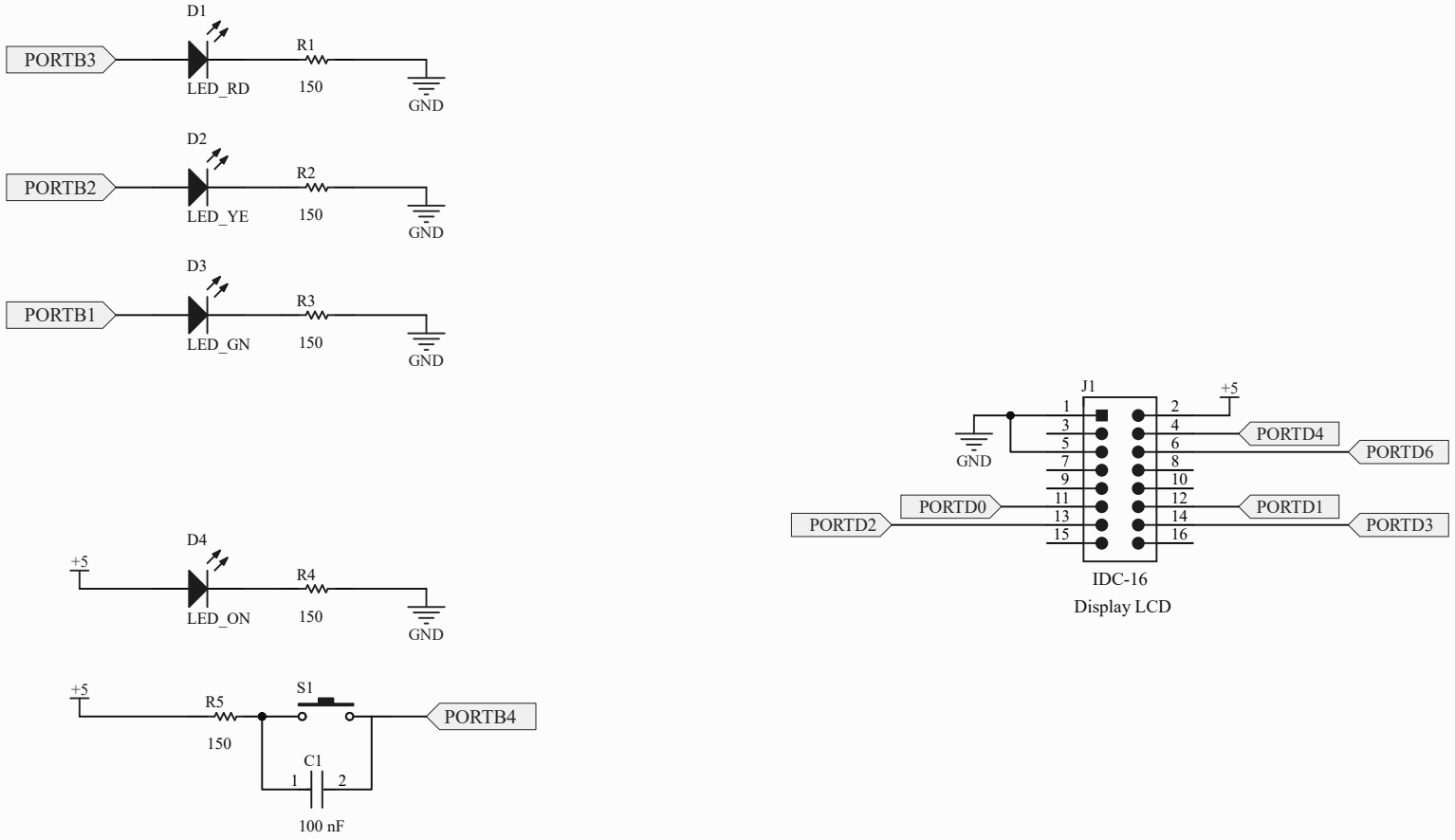
[11] http://ww1.microchip.com/downloads/en/DeviceDoc/50002750D.pdf

- https://www.microchip.com. [consulta] 15/01/2021
- https://www.learncomputerscienceonline.com/computer-bus. [consulta] 20/01/2021
- https://www.arm.com/why-arm/architecture. [consulta] 25/01/2021
- https://www.quora.com/What-is-the-difference-between-8051-PIC-AVR-and-ARM. [consulta] 25/01/2021
- https://www.docsity.com/es/maquina-sencilla-primero-de-informatica/3064054/. [consulta] 25/01/2021
- https://docencia.ac.upc.edu/eines/MR/DOCUMENTOS/Articulos/jenui97.pdf. [consulta] 25/01/2021
- https://cursoslared.com/archivos/2023/aci-maquina-sencilla-manual. [consulta] 25/01/2021
- http://cv.uoc.edu/annotation/8255a8c320f60c2bfd6c9f2ce11b2e7f/619469/PID_00218274/PID_00218274.html. [consulta] 11/02/2021
- http://www.ieec.uned.es/investigacion/Dipseil/PAC/archivos/Informacion_de_referencia_2_ISE3_4_2.pdf. [consulta] 11/02/2021
- https://www.microchip.com/content/dam/mchp/documents/OTH/ProductDocuments/Brochures/30009630M.pdf. [consulta] 16/02/2021
- https://www.digikey.es/es/ptm/a/atmel/mcu-product-line-introduction/tutorial. [consulta] 16/02/2021
- https://www.microchip.com/content/dam/mchp/documents/OTH/ProductDocuments/Brochures/30009630M.pdf. [consulta] 16/02/2021
- https://www.geeksforgeeks.org/computer-organization-and-architecture-pipelining-set-1-execution-stages-and-throughput/. [consulta] 20/02/2021
- http://web.csulb.edu/~hill/ee346/Lectures/02%20Intro%20Microcontroller.pdf. [consulta] 02/03/2021
- https://www.renesas.com/kr/en/support/engineer-school/mcu-programming-peripherals-05. [consulta] 15/03/2021
- https://learn.adafruit.com/memories-of-an-arduino/arduino-memories. [consulta] 15/03/2021
- http://ww1.microchip.com/downloads/en/devicedoc/atmel-0856-avr-instruction-set-manual.pdf. [consulta] 15/03/2021
- https://microchipdeveloper.com/. [consulta] 20/03/2021
- https://www.arxterra.com/13-2-avr-addressing-indirect/. [consulta] 20/03/2021
- https://aprendiendoarduino.wordpress.com/tag/isr/. [consulta] 24/03/2021

- http://ww1.microchip.com/downloads/Secure/en/DeviceDoc/50001894J.pdf. [consulta] 24/03/2021
- http://ww1.microchip.com/downloads/en/DeviceDoc/50002750D.pdf. [consulta] 13/04/2021
- https://ww1.microchip.com/downloads/en/DeviceDoc/AVR-Instruction-Set-Manual-DS40002198A.pdf. [consulta] 13/04/2021
- https://www.freepik.es/vector-premium/signo-exclamacion-cuidado-iconos_5322684.htm#page=1&query=attention&position=33. [consulta] 18/04/2021
- https://www.vectorstock.com/royalty-free-vector/light-bulb-icon-vector-26749598.[consulta] 18/04/2021
- https://www.allaboutcircuits.com/technical-articles/assembly-vs-c-why-learn-assembly/. [consulta] 02/05/2021
- http://www.avr-asm-tutorial.net/avr_en/timingloops/delay8.html. [consulta] 02/05/2021
- https://www.geeksforgeeks.org/addressing-modes/. [consulta] 10/05/2021
- https://www.geeksforgeeks.org/switch-debounce-in-digital-circuits/. [consulta] 12/05/2021
- https://programarfacil.com/blog/arduino-blog/resistencia-pull-up-y-pull-down/. [consulta] 12/05/2021
- http://web.alfredstate.edu/faculty/weimandn/lcd/lcd_initialization/lcd_initialization_index.html. [consulta] 15/07/2021
- https://www.softcatala.org/
- https://dlc.iec.cat/

# 8. Annexos

### 8.1. Esquemàtic Mòdul Extern




| | | |
|---|---|---|
| Title: Esquemàtic del Mòdul Extern | | |
| Size: A4 | Number: 01 | Revision: 01 |
| Date: 9/03/2021 | | Sheet1 of 1 |
| File: C:\Users\..\Placa.SchDoc | | Drawn By: Laia Altès |

## 8.2. Codi Desenvolupat

### *8.2.1. Pràctica 1*

```
;
; P1_Delay.asm
;
;***********************************************************
;FUNCIONALITAT: Bucle que genera un retard
;ATENCIÓ: S'utilitza el registre r16
;Es considera una freqüència de treball d'1 MHz
;***********************************************************

.org 0x0000                    ;El programa comença a la posició 0x0000

ldi r16, 99                    ;Carrega el valor 99 al registre r16
loop:                          ;Etiqueta
      nop                      ;Bucle amb instruccions que no fan res
      nop                      ;Serveixen per a esperar
      nop
      nop
      nop
      nop
      nop
      dec r16                  ;Resta 1 del valor que hi ha en r16
      brne loop                ;Salt condicional, salta si no és igual a 0
```

### *8.2.2. Pràctica 2*

```
;
; P2_1.asm
;
;***************************************************
;EXERCICI: Pràctica 2 - Estudi Previ, exercici 4
;FUNCIONALITAT: Pampallugues de l'User LED
;ATENCIÓ: S'utilitzen el registre r16 i r17
;***************************************************

.include "m328pdef.inc"
.device atmega328p            ;Microcontrolador que s'utilitza

.org 0x0000                   ;Programa comença a l'adreça 0x0000
rjmp reset


reset:                        ;Freqüència de treball a 1 MHz
      ldi r16, 0x80           ;Carrega el valor 0x80 a r16
      sts CLKPR, r16          ;Carrega valor a l'espai de memòria
      ldi r16, 0x04           ;Carrega el valor 0x04 al r16
      sts CLKPR, r16          ;Carrega valor a l'espai de memòria

start:
      ldi r17, 0xff           ;Carrega 0xff a r17
      out DDRB, r17           ;Assigna PORTB com a sortida

blink:
      ldi r17, 0b00100000
      out PORTB, r17          ;LED on, posa a 1 pin1 del PORTB

;Delay
      ldi r17, 0xff           ;Carrega el valor més gran possible a r17
loop1:
      ldi r16, 99             ;Carrega el valor 99 al registre r16
loop2:
      nop                     ;Bucle amb instruccions que no fan res
      nop                     ;Serveixen per esperar
      nop
      nop
      nop
      nop
      nop
      dec r16                 ;Resta 1 del valor que hi ha en r16
      brne loop2              ;Salt condicional, salta si no és igual a 0
      dec r17                 ;Resta 1 del valor que hi ha en r17
      brne loop1              ;Salt condicional, salta si no és igual a 0

      ldi r17, 0b00000000
      out PORTB, r17          ;LED off

;Delay
      ldi r17, 0xff           ;Carrega el valor més gran possible a r17
loop3:
      ldi r16, 99             ;Carrega el valor 99 al registre r16
loop4:                        ;Etiqueta
      nop                     ;Bucle amb instruccions que no fan res
      nop                     ;Serveixen per esperar
      nop
      nop
```

```
      nop
      nop
      nop
      dec r16                 ;Resta 1 del valor que hi ha en r16
      brne loop4              ;Salt condicional, salta si no és igual a 0
      dec r17                 ;Resta 1 del valor que hi ha en r17
      brne loop3              ;Salt condicional, salta si no és igual a 0
      rjmp blink



;
; P2_2.asm
;
;***********************************************
;EXERCICI: Pràctica 2 - Estudi Previ, exercici 6
;FUNCIONALITAT: Encén el LED del kit quan el
;polsador està polsat. Si no, el LED està apagat
;***********************************************

.include "m328pdef.inc"
.device atmega328p            ;Microcontrolador que s'utilitza

.org 0x0000                   ;Programa comença a l'adreça 0x0000
rjmp reset


reset:                        ;Freqüència de treball a 1 MHz
      ldi r16, 0x80           ;Carrega el valor 0x80 a r16
      sts CLKPR, r16          ;Carrega valor a l'espai de memòria
      ldi r16, 0x04           ;Carrega el valor 0x04 a r16
      sts CLKPR, r16          ;Carrega valor a l'espai de memòria

start:
      ldi r17, 0b00100000
      out DDRB, r17           ;Assigna PORTB5 com a sortida i PORTB7 com
a entrada

No_polsat:
      sbic PINB, 7            ;Polsador està al PORTB 7
      rjmp No_polsat

      sbi PORTB, 5            ;LED on

still_pushed:
      sbis PINB, 7
      rjmp still_pushed
      cbi PORTB, 5            ;LED off
      rjmp No_polsat
```

```
;
; P2_3.asm
;
;*******************************************
;EXERCICI: Pràctica 2 - Estudi Previ, exercici 7
;FUNCIONALITAT: Bucle de retard 30 ms
;ATENCIÓ: S'utilitzen els registres 16 i 17
;*******************************************

                              ;Delay
      ldi r17, 30             ;Carrega el valor 30 a r17
loop1:
      ldi r16, 99             ;Carrega el valor 99 al registre r16
loop2:
      nop                     ;Bucle amb instruccions que no fan res
      nop                     ;Serveixen per esperar
      nop
      nop
      nop
      nop
      nop
      dec r16                 ;Resta 1 del valor que hi ha en r16
      brne loop2              ;Salt condicional, salta si no és igual a 0
      dec r17                 ;Resta 1 del valor que hi ha en r17
      brne loop1              ;Salt condicional, salta si no és igual a 0



;
; P2_4.asm
;
;*****************************************************************
;EXERCICI: Pràctica 2 - Laboratori, exercici 2
;FUNCIONALITAT: On/Off LED mitjançant el polsador
;Estat inicial: LED apagat
;ATENCIÓ: S'utilitzen els registres 16 i 17 al bucle de retard
;*****************************************************************

.include "m328pdef.inc"
.device atmega328p

.org 0x0000
rjmp reset

reset:                        ;Freqüència de treball a 1 MHz
      ldi r16, 0x80           ;Carrega el valor 0x80 a r16
      sts CLKPR, r16          ;Carrega valor a l'espai de memòria
      ldi r16, 0x04           ;Carrega el valor 0x04 a r16
      sts CLKPR, r16          ;Carrega valor a l'espai de memòria


start:
;Control pulsador
      ldi r17, 0b00100000
      out DDRB, r17           ;Assigna PORTB5 com a sortida i PORTB7 com
a entrada
      ldi r20, 0x00           ;r20 = off
      ldi r19, 0x00           ;r19 = estat - comença a off

No_polsat:
      sbic PINB, 7            ;Polsador està al PORTB 7
      rjmp No_polsat
```

```
;Delay
      ldi r17, 30             ;Carrega el valor 30 a r17
loop1:
      ldi r16, 99             ;Carrega el valor 99 al registre r16
loop2:
      nop                     ;Bucle amb instruccions que no fan res
      nop                     ;Serveixen per esperar
      nop
      nop
      nop
      nop
      nop
      dec r16                 ;Resta 1 del valor que hi ha en r16
      brne loop2              ;Salt condicional, salta si no és igual a 0
      dec r17                 ;Resta 1 del valor que hi ha en r17
      brne loop1              ;Salt condicional, salta si no és igual a 0


      cp r20, r19
      brne apagar

      ldi r19, 0xff           ;estat on
      sbi PORTB, 5            ;LED on
      rjmp still_pushed

apagar:
      ldi r19, 0x00           ;estat off
      cbi PORTB, 5            ;LED off

still_pushed:
      sbis PINB, 7
      rjmp still_pushed


;Delay
      ldi r17, 30             ;Carrega el valor 30 a r17
loop3:
      ldi r16, 99             ;Carrega el valor 99 al registre r16
loop4:
      nop                     ;Bucle amb instruccions que no fan res
      nop                     ;Serveixen per esperar
      nop
      nop
      nop
      nop
      nop
      dec r16                 ;Resta 1 del valor que hi ha en r16
      brne loop4              ;Salt condicional, salta si no és igual a 0
      dec r17                 ;Resta 1 del valor que hi ha en r17
      brne loop3              ;Salt condicional, salta si no és igual a 0


      rjmp No_polsat
```

### *8.2.3. Pràctica 3*

```
;
; P3_1.asm
;
;***********************************************
;EXERCICI: Pràctica 3 - Estudi Previ, exercici 1
;FUNCIONALITAT: Rutina retard 1 ms
;ATENCIÓ: S'utilitza el registre 16
;***********************************************

ldi r17, 1
delay_1ms:
      ldi r16, 99             ;Carrega el valor 99 al registre r16
loop:                         ;Etiqueta
      nop                     ;Bucle amb instruccions que no fan res
      nop                     ;Serveixen per esperar
      nop
      nop
      nop
      nop
      nop
      dec r16                 ;Resta 1 del valor que hi ha en r16
      brne loop               ;Salt condicional, salta si no és igual
      ret                     ;Retorn de la subrutina
;-------------------------------



;
; P3_2.asm
;
;************************************************************
;EXERCICI: Pràctica 3 - Estudi Previ, exercici 4
;FUNCIONALITAT: Seqüència dels 3 LEDs (vermell, groc i verd)
;Utilitzant la llibreria no reubicable Delay.inc
;ATENCIÓ: S'utilitza el registre 16 a la rutina de retard
;************************************************************

.include "m328pdef.inc"
.include "Delay.inc"
.device atmega328p

.org 0x0000
rjmp reset

reset:                              ;Freqüència de treball a 1 MHz
      ldi r16, 0x80                 ;Carrega el valor 0x80 a r16
      sts CLKPR, r16                ;Carrega valor a l'espai de memòria
      ldi r16, 0x04                 ;Carrega el valor 0x04 a r16
      sts CLKPR, r16                ;Carrega valor a l'espai de memòria

start:
      ldi r17, 0xff
      out DDRB, r17                 ;Assigna PORTB com a sortida

vermell:
      sbi PORTB, 3                  ;LED vermell on
      call delay_1s
      call delay_1s
      cbi PORTB, 3                  ;LED vermell off
```

```
groc:
      sbi PORTB, 2                  ;LED groc on
      call delay_1s
      cbi PORTB, 2                  ;LED groc off

verd:
      sbi PORTB, 1                  ;LED verd on
      call delay 1s
      call delay_1s
      cbi PORTB, 1                  ;LED verd off
      rjmp vermell



;
; P3_2_Reubicable.asm
;
;***********************************************************
;EXERCICI: Pràctica 3 - Estudi Previ, exercici 4
;FUNCIONALITAT: Seqüència dels 3 LEDs (vermell, groc i verd)
;Utilitzant la llibreria reubicable Delay.asm
;ATENCIÓ: S'utilitza el registre 16 a la rutina de retard
;***********************************************************

.include "m328pdef.inc"
.device atmega328p

.org 0x0000
rjmp reset
.include "delay.asm"

reset:                              ;Freqüència de treball a 1 MHz
      ldi r16, 0x80                 ;Carrega el valor 0x80 a r16
      sts CLKPR, r16                ;Carrega valor a l'espai de memòria
      ldi r16, 0x04                 ;Carrega el valor 0x04 a r16
      sts CLKPR, r16                ;Carrega valor a l'espai de memòria

start:
      ldi r17, 0xff
      out DDRB, r17                 ;Assigna PORTB com a sortida

vermell:
      sbi PORTB, 3                  ;LED vermell on
      call delay_1s
      call delay_1s
      cbi PORTB, 3                  ;LED vermell off

groc:
      sbi PORTB, 2                  ;LED groc on
      call delay_1s
      cbi PORTB, 2                  ;LED groc off

verd:
      sbi PORTB, 1                  ;LED verd on
      call delay_1s
      call delay_1s
      cbi PORTB, 1                  ;LED verd off
      rjmp vermell
```

```
;
; P3_3.asm
;
;***************************************************
;EXERCICI: Pràctica 3 - Laboratori, exercici 2
;RUTINA: Polsador
;FUNCIONALITAT: Gestió de l'activació d'un polsador
;Tenint en compte l'efecte rebot i el soroll
;***************************************************

No_polsat:
      sbic PINB, 7                ;Polsador està al PORTB 7
      rjmp No_polsat

      call delay_10ms
      call delay_10ms
      call delay_10ms

      sbic PINB, 7
      rjmp No_polsat              ;Evitar soroll

;***************************************************
;Incloure les instruccions relatives a l'acció
;***************************************************

still_pushed:
      sbis PINB, 7
      rjmp still_pushed
      call delay_10ms
      call delay_10ms
      call delay_10ms
      rjmp No_polsat



;
; P3_4.asm
;
;*********************************************************
;EXERCICI: Pràctica 3 - Laboratori, exercici 3
;RUTINA: Polsador
;FUNCIONALITAT: Seqüència dels tres LEDs
;S'inicialitza al polsar el polsador
;Conté la gestió completa de l'anti-rebot
;LLIBRERIA: S'utilitza la llibreria reubicable Delay.asm
;*********************************************************

.include "m328pdef.inc"
.include "Delay.asm"
.device atmega328p

.org 0x0000
rjmp reset


reset:                            ;Freqüència de treball a 1 MHz
      ldi r16, 0x80               ;Carrega el valor 0x80 a r16
      sts CLKPR, r16              ;Carrega valor a l'espai de memòria
      ldi r16, 0x04               ;Carrega el valor 0x04 a r16
      sts CLKPR, r16              ;Carrega valor a l'espai de memòria

start:
```

```
      ldi r17, 0b00001111
      out DDRB, r17                 ;Assigna PORTB com a sortida i
entrada

No_polsat:
      sbic PINB, 4                  ;Polsador està al PORTB 4
      rjmp No_polsat

      call delay_10ms
      call delay_10ms
      call delay_10ms

      sbic PINB, 4
      rjmp No_polsat                ;Evitar soroll

vermell:
      sbi PORTB, 3                  ;LED vermell on
      call delay_1s
      call delay_1s
      cbi PORTB, 3                  ;LED vermell off

groc:
      sbi PORTB, 2                  ;LED groc on
      call delay_1s
      cbi PORTB, 2                  ;LED groc off

verd:
      sbi PORTB, 1                  ;LED verd on
      call delay_1s
      call delay_1s
      cbi PORTB, 1                  ;LED verd off
      rjmp vermell
```

### *8.2.4. Pràctica 4*

```
/*
 * P4_1.c
*************************************************
EXERCICI: Pràctica 4 - Estudi Previ, exercici 5
FUNCIONALITAT: Familiarització amb l'ús de màscares
*************************************************
 */

#include <avr/io.h>

int main(void)
{
	volatile char mask = 0b00000001;	//Es declara la variable mask
	volatile char res = 0x00;		//Es declara la variable res

	DDRB = 0xff;		//PORTB com a sortida
	PORTB = 0x00;		//Inicialitza el PORTB a 0

	while (1)
	{

		PORTB = PORTB | 0x08;	//Força a 1 el bit 3 del PORTB
		PORTB = PORTB & 0xF7;	//Força a 0 el bit 3 del PORTB

		res = res | mask;	//Força a 1 el bit 0 de la variable
res
		mask = 0xfe;
		res = res & mask;	//Força a 0 el bit 0 de la variable
res


		//Utilitzant la tècnica de desplaçament a l'esquerra

		PORTB |= (1<<PORTB0);	//Força a 1 el bit 0 del PORTB
		PORTB |= (1<<PORTB1);	//Força a 1 el bit 1 del PORTB
		PORTB &= ~(1<<PORTB0);	//Força a 0 el bit 0 del PORTB.
Deixant a 1 el bit 1

		mask = (0x01 << 2);	//Màscara amb el bit 2 activat; mask
= 0x04
		PORTB = PORTB | mask;	//Força a 1 el bit 3 del PORTB


	//*******************************************************//
		//COMPROVACIÓ DE LA IMPLEMENTACIÓ DEL DESPLAÇAMENT A LA DRETA

		//Sense especificació del signe de l'enter
		volatile int output = 0;
		volatile int valor = 0x2f;
		output = valor >>2;

		output = 0;
		valor = -10;
		output = valor >> 1;
		output = valor >> 3;

		//Variable unsigned
		volatile unsigned int test = 10;
		test = test >>1;
```

```
            test = -10;
            test = test >>1;

            //Variable signed
            volatile signed int resultat = -20;
            resultat = resultat >>1;

            resultat = 20;
            resultat = resultat >> 2;

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

      }
}



/*
 * P4_2.c
************************************************************
EXERCICI: Pràctica 4 - Estudi Previ, exercici 6
FUNCIONALITAT: On/Off LED del kit
ATENCIÓ: Els retards es generen amb la llibreria delay.h
************************************************************
 */


#include <avr/io.h>
#define F_CPU 16000000        //Establir la freqüència de treball a 16
MHz
#include <util/delay.h>       //Incloure la llibreria delay.h

int main(void)
{
      DDRB = 0xff;      //PORTB com a sortida

    while (1)
    {
            PORTB |= 0x20;    //LED on

            _delay_ms(500);   //Retard en ms

            PORTB &= 0xdf;    //LED off

            _delay_ms(500);   //Retard en ms
    }
}
```

```
/*
*************************************************************
EXERCICI: Pràctica 4 - Estudi Previ, exercici 7
FUNCIONALITAT: On/Off LED del kit amb operacions de desplaçament
ATENCIÓ: Els retards es generen amb la llibreria delay.h
*************************************************************
*/

#include <avr/io.h>
#define F_CPU 16000000        //Establir la freqüència de treball a 16
MHz
#include <util/delay.h>       //Incloure la llibreria delay.h

int main(void)
{
      DDRB = 0xff;            //PORTB com a sortida

    while (1)
    {
            PORTB |= (1<<PORTB5);         //LED on

            _delay_ms(500);               //Retard en ms

            PORTB &= ~(1<<PORTB5);        //LED off

            _delay_ms(500);               //Retard en ms
    }
}



/*
 * P4_3.c
*************************************************************
EXERCICI: Pràctica 4 - Estudi Previ, exercici 6
FUNCIONALITAT: Control Oon/Off) del LED mitjançant el polsador
ATENCIÓ: Els retards es generen amb la llibreria delay.h
*************************************************************
 */

#include <avr/io.h>
#define F_CPU 16000000      //Establir la freqüència de treball a 16 MHz
#include <util/delay.h>     //Incloure la llibreria delay.h

volatile char LED_on = 0;

int main(void)
{
      DDRB &= 0x7f;           //PORTB7 com a entrada
      DDRB |= 0x20;           //PORTB5 com a sortida LED
      PORTB = PORTB & 0x7F;   //Polsador amb Pull-up deshabilitat

      PORTB &= ~(1<<PORTB5);  //LED off

    while (1)
    {
            if ((PINB & 0b10000000) == 0b00000000)    //Polsador tancat
            {
                  _delay_ms(30);    //Retard antirebot
                  if ((PINB & 0b10000000) == 0b00000000)
                  {
```

```
                        while ((PINB & 0b10000000) == 0b00000000);
      //Espera fins que no polsat
                        if (LED_on)
                        {
                              PORTB &= ~(1<<PORTB5);  //LED off
                              LED_on = 0;
                        }
                        else
                        {
                              PORTB |= (1<<PORTB5);   //LED on
                              LED_on = 1;
                        }

                  }
                  _delay_ms(30);    //Retard antirebot
            }
    }
}



/*
 * P4_4.c
*************************************************************
EXERCICI: Pràctica 4 - Treball al Laboratori, exercici 3
FUNCIONALITAT: Comptador binari. Mostra el nombre de vegades que
s'ha premut el polsador en els tres LEDs del mòdul extern
ATENCIÓ: Els retards es generen amb la llibreria delay.h
*************************************************************
 */

#include <avr/io.h>
#define F_CPU 16000000      //Establir la freqüència de treball a 16 MHz
#include <util/delay.h>     //Incloure la llibreria delay.h


int main(void)
{
      volatile char count = 9;

      DDRB &= 0xef;           //PORTB4 com a entrada
      DDRB |= 0x0e;           //Pins 1, 2 i 3 del PORTB com a sortida LED
      PORTB = PORTB & 0xef;   //Polsador amb Pull-up deshabilitat

      PORTB &= ~((1<<PORTB1)|(1<<PORTB2)|(1<<PORTB3));      //LEDs off

      while (1)
      {
            if ((PINB & 0b00010000) == 0b00000000)    //Polsador tancat
            {
                  _delay_ms(30);    //Retard antirebot
                  if ((PINB & 0b00010000) == 0b00000000)
      //Evita falses activacions per soroll
                  {
                        while ((PINB & 0b00010000) == 0b00000000);
      //Espera fins que no polsat
                        count++;

                        if (count<8)
                        {
```

```
                                PORTB = (count<<1);
                //Mostra el valor pels LEDs
                                }
                                else
        //Reset
                                {
                                      count = 0;
                                      PORTB &=
~((1<<PORTB1)|(1<<PORTB2)|(1<<PORTB3));   //LEDs off
                                            for (int i =0; i<3; i++)
                                            {
                                                  _delay_ms(500);
                                                  PORTB |= (1<<PORTB1);   //LED on
                                                  _delay_ms(500);
                                                  PORTB &= ~(1<<PORTB1);  //LED
off
                                            }
                                }
                        }
                        _delay_ms(30);    //Retard antirebot
                }
        }
}
```

### *8.2.5. Pràctica 5*

```
/*
 * P5_LCD.c
*****************************************************************
EXERCICI: Pràctica 5 - Estudi previ, exercicis 3, 4, 5, 6
FUNCIONALITAT: Funcions de control del LCD amb el fitxer de
capçalera Include.h
- Estudiar la funció LCD_Init()
- Estudiar la funció LCDWriteNibbleI()
- Desenvolupar la fució LCDWriteNibbleD()

EXERCICI: Pràctica 5 - Treball al laboratori, exercicis 1 i 2
FUNCIONALITAT: Mostrar per pantalla la paraula "HELLO"

ATENCIÓ: Els retards es generen amb la llibreria delay.h
*****************************************************************
 */

#define F_CPU 16000000
#include <util/delay.h>
#include <avr/io.h>
#include "Include.h"

void LCDWriteNibbleI(char data)
{
	int volatile pin;
	PORTD &= ~((1<<LCD_E)|(1<<LCD_RS));

	_delay_us(20);                    //Small delay
	PORTD |= (1<<LCD_E);          //Set E

	//Escriure DATA al PORTD

	//LCD_D7
	pin = 0b10000000;
	if (data & pin)
	{
		PORTD |= (1<<LCD_D7);
	}
	else
	{
		PORTD &= ~(1<<LCD_D7);
	}

	//LCD_D6
	pin = 0b01000000;
	if (data & pin)
	{
		PORTD |= (1<<LCD_D6);
	}
	else
	{
		PORTD &= ~(1<<LCD_D6);
	}

	//LCD_D5

	pin = 0b00100000;
	if (data & pin)
	{
```

```
            PORTD |= (1<<LCD_D5);
      }
      else
      {
            PORTD &= ~(1<<LCD_D5);
      }

      //LCD D4
      pin = 0b00010000;
      if (data & pin)
      {
            PORTD |= (1<<LCD_D4);
      }
      else
      {
            PORTD &= ~(1<<LCD_D4);
      }

      _delay_us(20);                //Small delay

      PORTD &= ~(1<<LCD_E);         //Clear E
}

void LCDWriteNibbleD(char lletra)
{
      int volatile pin;
      PORTD &= ~((1<<LCD_E));       //Clear E
      PORTD |= (1<<LCD_RS);         //Set RS. Per a enviar dades i no
instruccions
      _delay_us(20);                //Small delay
      PORTD |= (1<<LCD_E);          //Set E

      //Escriu lletra al PORTD
      //LCD_D7
      pin = 0b10000000;
      if (lletra & pin)
      {
            PORTD |= (1<<LCD_D7);
      }
      else
      {
            PORTD &= ~(1<<LCD_D7);
      }

      //LCD_D6
      pin = 0b01000000;
      if (lletra & pin)
      {
            PORTD |= (1<<LCD_D6);
      }
      else
      {
            PORTD &= ~(1<<LCD_D6);
      }

      //LCD_D5

      pin = 0b00100000;
      if (lletra & pin)
      {
            PORTD |= (1<<LCD_D5);
```

```
	}
	else
	{
		PORTD &= ~(1<<LCD_D5);
	}

	//LCD_D4
	pin = 0b00010000;
	if (lletra & pin)
	{
		PORTD |= (1<<LCD_D4);
	}
	else
	{
		PORTD &= ~(1<<LCD_D4);
	}

	_delay_us(20);			//Small delay
	PORTD &= ~(1<<LCD_E);		//Clear E
}


void LCDWrite_Instruccio (char data)	//Primer ha d'enviar els 4 MSB
{					//Segon, els 4 LSB

	LCDWriteNibbleI(data);
	LCDWriteNibbleI(data<<4);
}

void Escriu_Lletra (char lletra)	//Primer ha d'enviar els 4 MSB
{					//Segon, els 4 LSB
	LCDWriteNibbleD(lletra);
	LCDWriteNibbleD(lletra<<4);
}

void LCDWrite_Dada (char cadena[])
{
	for (int i =0; i<16; i++)
	{
		if(cadena[i]!= '\0')
		{
			Escriu_Lletra(cadena[i]);
		}
		else
		{
			return;
		}
	}
}


int LCD_Init(int DataLenght)
{
	if (DataLenght == 4)	//Segons especificacions en l'enunciat,
només es treballa amb la interfície de 4 bits
	{
		DDRD = 0xff;		//PORTD com a sortida
		PORTD = 0xff;		//Per defecte, PORTD a 1

		_delay_ms(50);		//Delay després de Power-on. Wait for
more than 40 ms after VCC rises to 2.7 V
```

```
            //*****************
            //Inicialització LCD

            //Function Set: Establir la interfície de bits, 4 o 8

            LCDWriteNibbleI(0x30);        //Cal enviar 3 cops la
configuració a 8-bit
             delay ms(5);
            LCDWriteNibbleI(0x30);
            _delay_us(120);
            LCDWriteNibbleI(0x30);
            _delay_us(120);                     //Fi del Bloc 2

            LCDWriteNibbleI(FUNCTION_SET);      //Interfície 4 bits, 1
fila, Format 5x8
            _delay_us(40);                //Fins aquí encara anem a 8bit

            LCDWrite_Instruccio(LINES_2);       //Function Set: 2 linies,
N = 1
            _delay_us(40);

            LCDWrite_Instruccio(DISPLAY_OFF_CURSOR_OFF);
            _delay_us(40);

            LCDWrite_Instruccio(CLEAR_DISPLAY);
            _delay_ms(2);

            LCDWrite_Instruccio(ENTRY_MODE_SET);
            _delay_us(40);

            //Inicialització completa. Fi del bloc 3
            //Display on

            LCDWrite_Instruccio(DISPLAY_ON_CURSOR_ON);
            _delay_us(40);

            return FALSE;           //Per indicar que el procés
d'inicialització s'ha realitzat amb èxit
      }

      else              //ERROR
      {
            return TRUE;
      }
}



int main(void)
{
      int error = FALSE;
      char paraula[16];

      while (1)
      {
            error = LCD_Init(4);
            if (!error)
            {
                  //Escriptura en el LCD
                  paraula[0] = 0x48;
                  paraula[1] = 0x45;
```

```
                paraula[2] = 0x4C;
                paraula[3] = 0x4C;
                paraula[4] = 0x4F;
                paraula[5] = '\0';

                LCDWrite_Dada(paraula);
                _delay_ms(2000);
                LCDWrite Instruccio(CLEAR_DISPLAY);
                _delay_ms(2000);
            }
        }
}


/*
 * Include.h
*****************************************************************
FUNCIONALITAT: Es declaren totes les definicions necessàries
per a la gestió del LCD
*****************************************************************
 */

#ifndef INCLUDE_H_
#define INCLUDE_H_

//-------------------------------------------------------------------------
//S'utilitzaran els següents ports:
#define LCD_D4    PORTD0
#define LCD_D5    PORTD1
#define LCD_D6  PORTD2
#define LCD_D7    PORTD3

#define LCD_RS    PORTD4
#define LCD_E     PORTD6

#define FUNCTION_SET               0x20
#define DISPLAY_OFF_CURSOR_OFF     0x08
#define CLEAR_DISPLAY              0x01
#define ENTRY_MODE_SET             0x06
#define DISPLAY_ON_CURSOR_ON  0x0f
#define LINES_2                          0x2c
#define CURSOR_HOME                      0x02
#define BLINK_OFF                  0x0e


#define FALSE     0
#define TRUE      1



#endif /* INCLUDE_H_ */
```

```
/*
 * P5_1.c
*************************************************************
EXERCICI: Pràctica 5 - Treball al laboratori, exercicis 3 i 4
FUNCIONALITAT: Mostra el valor del comptador en binari pels
3 LEDs de la placa i en decimal pel LCD.
Quan arriba al màxim, realitza l'etapa de reset corresponent.

ATENCIÓ: Utilitza les funcions de la gestió del LCD declarades
en l'arxiu LCD.h

ATENCIÓ: Els retards es generen amb la llibreria delay.h
*************************************************************
 */

#include <avr/io.h>
#define F_CPU 16000000      //Establir la freqüència de treball a 16 MHz
#include <util/delay.h>     //Incloure la llibreria delay.h
#include "Include.h"
#include "LCD.h"


int main(void)
{
      volatile char count = 0;
      int error = FALSE;
      volatile char caracter = 0;
      char paraula [16];

      DDRB &= 0xef;           //PORTB4 com a entrada
      DDRB |= 0x0e;           //Pins 1, 2 i 3 del PORTB com a sortida LED
      PORTB = PORTB & 0xef;   //Polsador amb Pull-up deshabilitat

      PORTB &= ~((1<<PORTB1)|(1<<PORTB2)|(1<<PORTB3));      //LEDs off
      error = LCD_Init(4);
      LCDWrite_Instruccio(BLINK_OFF);


      while (!error)
      {
            if ((PINB & 0b00010000) == 0b00000000)    //Polsador tancat
            {
                  _delay_ms(30);    //Retard antirebot
                  if ((PINB & 0b00010000) == 0b00000000)
      //Evita falses activacions per soroll
                  {
                        while ((PINB & 0b00010000) == 0b00000000);
      //Espera fins que no polsat
                        count++;

                        if (count<8)
                        {
                              PORTB = (count<<1);
            //Mostra el valor pels LEDs

                              caracter = Convertir_Caracter(count);
                              paraula[0] = caracter;
                              paraula[1] = '\0';

                              LCDWrite_Dada(paraula);
```

```
                              LCDWrite_Instruccio(CURSOR_HOME);

                              _delay_ms(2);
                        }
                        else
      //Reset
                        {
                              count = 0;
                              PORTB &=
~((1<<PORTB1)|(1<<PORTB2)|(1<<PORTB3));   //LEDs off
                              LCDWrite_Instruccio(CLEAR_DISPLAY);
                              _delay_ms(2);

                              paraula[0] = 0x45;                  //Mostra
per pantalla END
                              paraula[1] = 0x4e;
                              paraula[2] = 0x44;
                              paraula[3] = '\0';

                              LCDWrite_Dada(paraula);

                              for (int i =0; i<3; i++)
                              {
                                    _delay_ms(500);
                                    PORTB |= (1<<PORTB1);   //LED on
                                    _delay_ms(500);
                                    PORTB &= ~(1<<PORTB1);  //LED off
                              }


                              _delay_ms(3000);
                              LCDWrite_Instruccio(CLEAR_DISPLAY);
                              _delay_ms(2);
                        }
                  }
                  _delay_ms(30);    //Retard antirebot
            }
      }
}


/*
 * LCD.c
************************************************************
FUNCIONALITAT: Conté les definicions de les funcions
necessàries per a gestionar el LCD.
************************************************************
 */

//*********************************************************************
*********************
//*********************************FUNCIONS GESTIÓ
LCD*************************************

#include <avr/io.h>
#define F_CPU 16000000      //Establir la freqüència de treball a 16 MHz
#include <util/delay.h>     //Incloure la llibreria delay.h
#include "Include.h"
#include "LCD.h"

void LCDWriteNibbleI(char data)
```

```
{
	int volatile pin;
	PORTD &= ~((1<<LCD_E)|(1<<LCD_RS));

	_delay_us(20);			//Small delay
	PORTD |= (1<<LCD_E);		//Set E

	//Escriure DATA al PORTD

	//LCD_D7
	pin = 0b10000000;
	if (data & pin)
	{
		PORTD |= (1<<LCD_D7);
	}
	else
	{
		PORTD &= ~(1<<LCD_D7);
	}

	//LCD_D6
	pin = 0b01000000;
	if (data & pin)
	{
		PORTD |= (1<<LCD_D6);
	}
	else
	{
		PORTD &= ~(1<<LCD_D6);
	}

	//LCD_D5

	pin = 0b00100000;
	if (data & pin)
	{
		PORTD |= (1<<LCD_D5);
	}
	else
	{
		PORTD &= ~(1<<LCD_D5);
	}

	//LCD_D4
	pin = 0b00010000;
	if (data & pin)
	{
		PORTD |= (1<<LCD_D4);
	}
	else
	{
		PORTD &= ~(1<<LCD_D4);
	}

	_delay_us(20);			//Small delay

	PORTD &= ~(1<<LCD_E);		//Clear E
}

void LCDWriteNibbleD(char lletra)
{
```

```
	int volatile pin;
	PORTD &= ~((1<<LCD_E));		//Clear E
	PORTD |= (1<<LCD_RS);		//Set RS. Per a enviar dades i no
instruccions
	_delay_us(20);			//Small delay
	PORTD |= (1<<LCD_E);		//Set E

	//Escriu lletra al PORTD
	//LCD_D7
	pin = 0b10000000;
	if (lletra & pin)
	{
		PORTD |= (1<<LCD_D7);
	}
	else
	{
		PORTD &= ~(1<<LCD_D7);
	}

	//LCD_D6
	pin = 0b01000000;
	if (lletra & pin)
	{
		PORTD |= (1<<LCD_D6);
	}
	else
	{
		PORTD &= ~(1<<LCD_D6);
	}

	//LCD_D5

	pin = 0b00100000;
	if (lletra & pin)
	{
		PORTD |= (1<<LCD_D5);
	}
	else
	{
		PORTD &= ~(1<<LCD_D5);
	}

	//LCD_D4
	pin = 0b00010000;
	if (lletra & pin)
	{
		PORTD |= (1<<LCD_D4);
	}
	else
	{
		PORTD &= ~(1<<LCD_D4);
	}

	_delay_us(20);			//Small delay
	PORTD &= ~(1<<LCD_E);		//Clear E
}


void LCDWrite_Instruccio (char data)	//Primer ha d'enviar els 4 MSB
{					//Segon, els 4 LSB
```

```
	LCDWriteNibbleI(data);
	LCDWriteNibbleI(data<<4);
}

void Escriu_Lletra (char lletra)		//Primer ha d'enviar els 4 MSB
{						//Segon, els 4 LSB

	LCDWriteNibbleD(lletra);
	LCDWriteNibbleD(lletra<<4);
}

void LCDWrite_Dada (char cadena[])
{
	for (int i =0; i<16; i++)
	{
		if(cadena[i]!= '\0')
		{
			Escriu_Lletra(cadena[i]);
		}
		else
		{
			return;
		}
	}
}


int LCD_Init(int DataLenght)
{
	if (DataLenght == 4)    //Segons especificacions en l'enunciat,
només es treballa amb la interfície de 4 bits
	{
		DDRD = 0xff;		//PORTD com a sortida
		PORTD = 0xff;		//Per defecte, PORTD a 1

		_delay_ms(50);		//Delay després de Power-on. Wait for
more than 40 ms after VCC rises to 2.7 V

		//*****************
		//Inicialització LCD

		//Function Set: Establir la interfície de bits, 4 o 8

		LCDWriteNibbleI(0x30);		//Cal enviar 3 cops la
configuració a 8-bit
		_delay_ms(5);
		LCDWriteNibbleI(0x30);
		 delay us(120);
		LCDWriteNibbleI(0x30);
		_delay_us(120);				//Fi del Bloc 2

		LCDWriteNibbleI(FUNCTION_SET);		//Interfície 4 bits,
1 fila, Format 5x8
		_delay_us(40);			//Fins aquí encara anem a 8bit

		LCDWrite_Instruccio(LINES_2);	//Function Set: 2 linies,
N = 1
		_delay_us(40);

		LCDWrite_Instruccio(DISPLAY_OFF_CURSOR_OFF);
		_delay_us(40);
```

```
            LCDWrite_Instruccio(CLEAR_DISPLAY);
            _delay_ms(2);

            LCDWrite_Instruccio(ENTRY_MODE_SET);
            _delay_us(40);

            //Inicialització completa. Fi del bloc 3
            //Display on

            LCDWrite_Instruccio(DISPLAY_ON_CURSOR_ON);
            _delay_us(40);

            return FALSE;           //Per indicar que el procés
d'inicialització s'ha realitzat amb èxit
      }

      else              //ERROR
      {
            return TRUE;
      }
}

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

char Convertir_Caracter(char nombre)
{
      char caracter = 0;
      switch(nombre)
      {
            case 0:
            caracter = 0x30;
            break;
            case 1:
            caracter = 0x31;
            break;

            case 2:
            caracter = 0x32;
            break;

            case 3:
            caracter = 0x33;
            break;

            case 4:
            caracter = 0x34;
            break;

            case 5:
            caracter = 0x35;
            break;

            case 6:
            caracter = 0x36;
            break;

            case 7:
```

```
            caracter = 0x37;
            break;

            case 8:
            caracter = 0x38;
            break;
      }
      return caracter;
}


/*
 * LCD.h
*************************************************************
FUNCIONALITAT: Conté les declaracions de les funcions
necessàries per a gestionar el LCD.
És a dir, conté els prototips de les funcions.
*************************************************************
 */

void LCDWriteNibbleI(char data);
void LCDWriteNibbleD(char lletra);
void LCDWrite_Instruccio (char data);
void Escriu_Lletra (char lletra);
void LCDWrite_Dada (char cadena[]);
int LCD_Init(int DataLenght);
char Convertir_Caracter(char nombre);
```