E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

DESARROLLO DE SISTEMAS BASADOS EN UN MICROCONTROLADOR PIC

Trabajo Fin de Grado

Mario Aliaga Igea

Carlos Ruiz Zamarreño

Pamplona, 6 de septiembre de 2021

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

E.T.S. de Ingeniería Industrial y de Telecomunicaciones

Resumen

En la siguiente memoria se realizará una búsqueda de una placa de desarrollo económica que incorpore con un microcontrolador PIC. La utilización de este tipo de microcontroladores se debe al gran número de aplicaciones en los que es utilizado en la fabricación de componentes electrónicos.

El objetivo que tiene esta placa es que pueda ser comprada por los alumnos de la UPNA (Universidad Pública de Navarra) y la puedan utilizar en su casa para complementar el aprendizaje de programación de microcontroladores. Para que esto resulte más sencillo esta memoria incorpora una guía en la que se explican paso a paso ejemplos prácticos la programación de microcontroladores utilizando el software MPLAB X IDE.

Por último, se explicará el desarrollo de un sistema más complejo que los vistos anteriormente en el que se utilicen varios de los módulos que incorpora el microcontrolador.

Lista de palabras clave

- Placa de desarrollo
- Microcontrolador
- Periféricos
- Entradas
- Salidas
- Sensores
- MPLAB X IDE
- *Pin Module*

Summary

In the next memory Will be performed a search with the purpose to find a development board with a microcontroller PIC for the students of UPNA. The students should buy the board and they use this for study the programming of the microcontroller with the software MPLAB X IDE.

This memory includes a guide with some exercises which use different modules, with this exercises the students can learn the fuctioning of the modules of the microcontroller.

Finally, this memory has two proyects, the first proyect simulates a home automation with a mobile APP, this APP permits the control some leds with a sensor of luminosity, a sensor of temperature and a sensor of appeareance. The second proyect is a led panel and in this panel you can see the temperature of the room.

Keywords

- Development Board
- Microcontroller
- Input
- Output
- Sensors
- MPLAB X IDE
- Pin Module

Índice

Índice de ilustraciones

1. Introducción y objetivos

En la actualidad, la mayoría de las empresas están automatizando gran parte de sus procesos de producción debido a las grandes ventajas que ofrece, como son un aumento de la producción, reducción de costes operativos, reducción de equivocaciones, aumento de la seguridad de los empleados. Además de los procesos industriales, las tareas domésticas también se están automatizando, permitiendo a las personas evitar las tareas monótonas y tener más tiempo libre.

En la gran mayoría de aplicaciones electrónicas que requieren entradas y salidas se utilizan sistemas que incorporan microcontroladores desde electrodomésticos, juguetes, despertadores hasta coches. Debido a la gran utilización de los sistemas basados en microcontroladores, resulta muy interesante que los alumnos aprendan a diseñar y manejar estos sistemas a la hora de incorporarse al mundo laboral.

En el semestre de primavera del tercer curso del grado de Ingeniería en Tecnologías Industriales impartido en la UPNA (Universidad Pública de Navarra) en la mención de Electrónica Industrial se imparte la asignatura de Sistemas Digitales, cuyo objetivo es aprender a diseñar sistemas de control basados en el microcontrolador PIC16F877A de Microchip, para ello se utiliza una placa de desarrollo en la que se incorporan diferentes elementos como son pulsadores, leds, interruptores, displays de 7 segmentos o pantalla LCD que permiten interactuar con el sistema actuando como entradas y salidas.

El desarrollo de la tecnología avanza muy rápidamente, por lo que los microcontroladores y placas de desarrollo que se utilizan en esta asignatura se quedan obsoletos en comparación con los que hay en mercado, en términos de consumo, de velocidad de ejecución de operaciones y la capacidad para incorporar diferentes módulos que permiten adaptarse a sistemas más complejos.

Este proyecto surge con el objetivo de que los alumnos puedan adquirir una placa de desarrollo y puedan complementar el trabajo realizado en las aulas desde sus casas, de esta forma su aprendizaje será mejor debido a que tendrán la capacidad de comprobar el funcionamiento directamente en la placa de desarrollo en vez de utilizar un simulador.

os objetivos de este proyecto son:

- Establecer las necesidades de aprendizaje que tienen los alumnos sobre sistemas basados en microprocesadores para poder acceder al mercado laboral con mayor facilidad.
- Estudio de las diferentes placas de desarrollo que se encuentran actualmente en el mercado para su posterior utilización en las prácticas de Sistemas Digitales.
- Elaboración de diferentes sistemas básicos que los alumnos puedan realizar en las prácticas de Sistemas Digitales con el fin de aprender a manejar los diferentes módulos que tiene un microcontrolador.

- Estudio de los diferentes sistemas de control que se pueden implantar en la placa de desarrollo escogida y posterior desarrollo de uno de esos sistemas de control.
- Comprobación del funcionamiento del sistema desarrollado y estudiar sus puntos fuertes y débiles y ver las posibles vías de solución.

2. Antecedentes

Un microcontrolador consiste en un circuito integrado programable que tiene la capacidad de recibir, interpretar y generar señales digitales internas y/o externas que se utiliza para controlar sistemas. Está compuesto básicamente de cuatro elementos de CPU, memoria, buses y puertos de entradas y salidas [1][2].

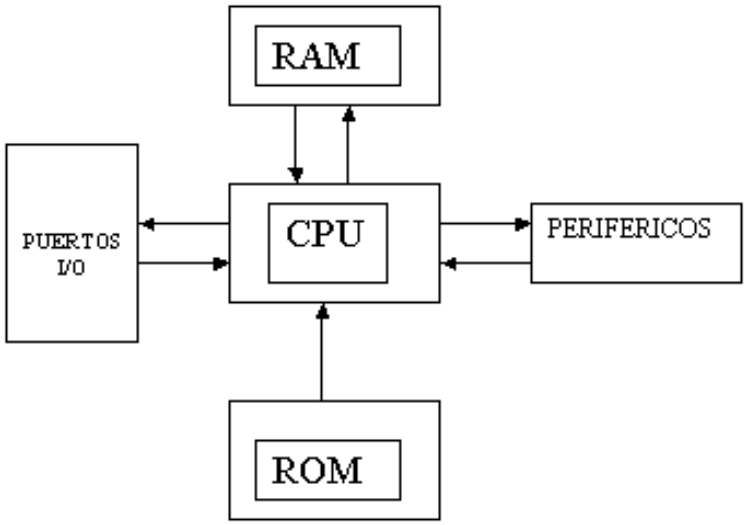


Ilustración 1: Esquema microcontrolador [1]

-La CPU (Unidad Central de Proceso): es la parte más importante del microcontrolador, se encarga de controlar a las demás e interpreta y ejecuta las instrucciones.

-Los buses: son los encargados de unir las diferentes partes del microcontrolador, pueden ser unidireccionales o bidireccionales y las señales que transmiten son de tres tipos: datos, direcciones y control.

-La memoria central se divide en dos partes:

> -Memoria de programa: contiene las instrucciones que se ejecutan, esta memoria es sólo de lectura.
>
> -Memoria de datos: utiliza la CPU para almacenar resultados parciales o finales.

-Entradas y salidas: son las encargadas de recoger la información de los diferentes sensores y actuadores y volver a transmitirla al exterior.

Además de estos cuatro elementos, en la actualidad los microcontroladores incorporan una serie de periféricos como son los Convertidores Analógico Digitales (CAD), módulos Captura/Comparación/ *Pulse Width Modulation*(CCP), interrupciones, etc.

2.1. Origen de los microcontroladores

El origen de los microcontroladores se remonta a 1971, cuando los ingenieros Gary Boone y Michael Cochran de la empresa estadounidense Texas Instruments desarrollaron el TMS1000, este microcontrolador era de 4 bits y tenía memoria RAM y ROM. Este microcontrolador era utilizado por la propia empresa en sus calculadoras y a partir de 1974 se pone a la venta para la industria [3].

Ilustración 2: Microcontrolador TMS1000 [3]

En esta época los microcontroladores se dividían en dos grupos diferentes, los que tenían memoria EPROM y PROM, la gran diferencia entre estos dos grupos es que los microcontroladores EPROM se pueden reprogramar y los PROM no, aunque para borrar los datos almacenados en la memoria EPROM era necesario exponerlos a una luz de rayos ultravioletas.

Unos años más tarde, a mitad de los años 80 se lanzaron al mercado los primeros microcontroladores con memoria EEPROM, este tipo de memoria se puede programar, borrar y reprogramar eléctricamente, lo que supone una gran ventaja respecto a la memoria EPROM.

En 1993 aparece el 16F84A, que es primer microcontrolador PIC de microchip, la gran novedad que aportaba este microcontrolador era que incorporaba un entorno de programación y simulación para Windows.

En 1996 Atmel crea el primer microcontrolador con memoria Flash para almacenar el programa, la ventaja de esta memoria es que una vez compilado el programa no se puede sobrescribir durante la ejecución del programa como si ocurre con la memoria EEPROM.

En la actualidad los microcontroladores utilizan un tipo u otro de memoria según las características del sistema en el que vayan a ser utilizados.

2.2. Tipos de microcontroladores

Los principales fabricantes de microcontroladores en la actualidad son Freescale Semiconductor, Texas Instruments (TI), ZiLOG Inc, Motorola, Intel y Microchip [5]. A pesar de que hay una gran variedad de fabricantes los microcontroladores que existen en la actualidad se dividen en dos grandes grupos dependiendo de la arquitectura utilizada: Von Neumann y Harvard.

Tradicionalmente la arquitectura usada en procesadores y microcontroladores era la arquitectura Von Neumann, pero en la actualidad debido al aumento de velocidad de procesamiento la arquitectura Harvard está siendo más utilizada para el diseño de microcontroladores.

2.2.1. Arquitectura Von Neumann

Los sistemas que utilizan este tipo de arquitectura utilizan una única memoria para la lectura y escritura, que contiene los datos necesarios, al haber solo una memoria el acceso a ella es por posición tanto para datos como para instrucciones, esto provoca que la velocidad de procesamiento sea menor que en la estructura Harvard, aunque también su diseño es más sencillo. Otra de las limitaciones que tiene este tipo de estructura es que la longitud de las instrucciones viene fijada por la longitud de los datos, por lo que para utilizar estructuras más complejas tiene que hacer diferentes accesos a la memoria.

En la actualidad, esta arquitectura se utiliza en los procesadores de los ordenadores porque al tener únicamente una memoria la cantidad de buses necesarios es menor y los procesadores pueden ser más pequeños y baratos [6][7].

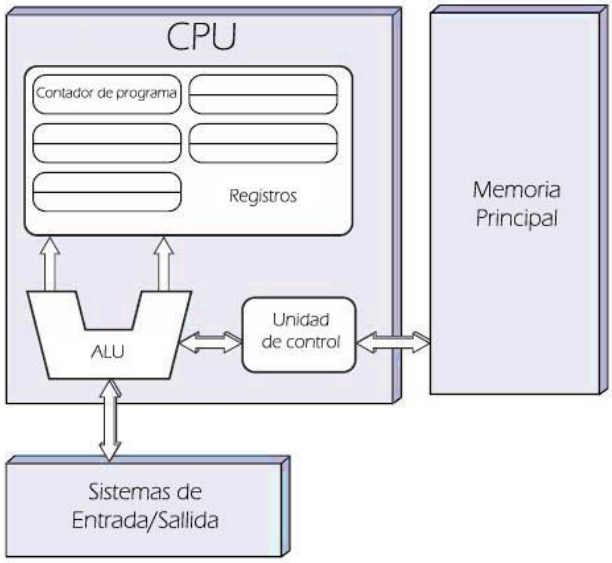


Ilustración 3: Arquitectura Von Neumann [7]

2.2.2. Arquitectura Harvard

El diseño de esta arquitectura consiste en tener dos memorias diferentes, una para instrucciones y otra para datos conectadas a la CPU por medio de dos buses diferentes por lo que la longitud de ambos puede ser diferentes. Al estar conectadas las dos memorias de forma independiente se puede acceder a ellas simultáneamente consiguiendo una mayor velocidad y menor longitud de programa [8][9].

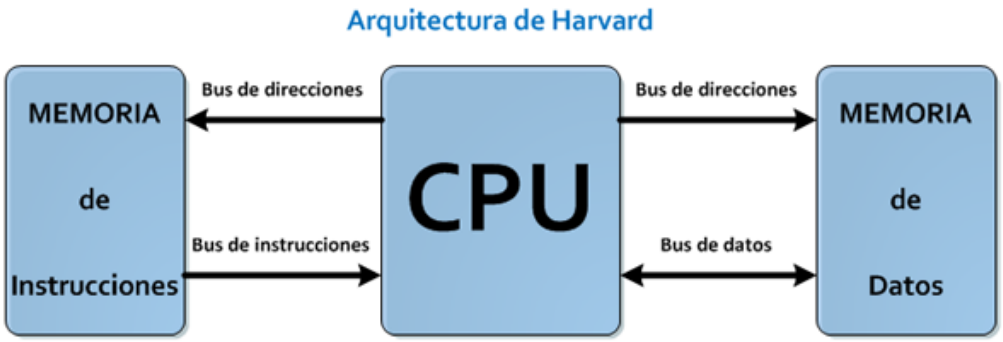


Ilustración 4: Arquitectura Harvard [9]

2.3. Microcontroladores PIC de Microchip

Los microcontroladores PIC de Microchip tienen las siguientes características [10]:

- Utilizan la Arquitectura Harvard (descrita en el apartado 2.2.2)
- Tienen una gran eficiencia del código.
- Funcionan a gran velocidad de ejecución.
- Posee una gran cantidad de herramientas de desarrollo de software y hardware y de bajo coste.
- Tiene compatibilidad de pines y código entre diferentes dispositivos de la misma familia.

Tiene una gran variedad de interrupciones que pueden ser empleadas para generar señales o para detectar cambios en las entradas.

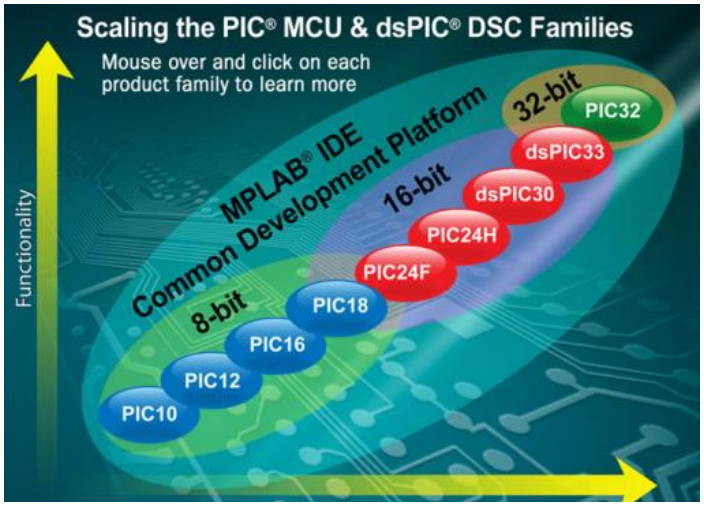


Ilustración 5: Familias de PICs [11]

Los microcontroladores se dividen en diferentes grupos según los bits utilizados para la memoria, que pueden ser 8, 16 y 32 bits. Los microcontroladores de 8 bits son los más usados debido a su facilidad de uso, bajo coste y bajo consumo.

Los microcontroladores de 16 y 32 bits se utilizan en sistemas que utilizan redes y comunicaciones porque la mayoría de las pilas de comunicaciones y protocoles de red son de 16 o 32 bits.

2.3.1. Microcontroladores de 8 bits

Dentro de la familia de microcontroladores PIC de 8 bits de Microchip, los microcontroladores se dividen según el tamaño de las instrucciones [12].

Gama baja

Esta gama se utiliza en aplicaciones en las que no se requieren grandes prestaciones, el tamaño de las instrucciones es de 12 bits, no disponen de interrupciones, la pila solo tiene dos niveles por lo que no se pueden encadenar más de dos subrutinas. Estas prestaciones no son suficientes para el sistema que se va a desarrollar, por lo que no se utilizará un PIC de esta gama.

Gama Media y Media-Mejorada

Los microcontroladores de esta gama tienen instrucciones de 14 bits, y mejora las prestaciones de la gama baja, incorporando interrupciones, convertidores analógico-digital, puertos serie y temporizadores. Los pines disponibles van desde 6 a 64. La pila tiene 8 niveles en la gama media y 16 en la media-mejorada.

Gama Alta

El tamaño de las instrucciones de esta gama es de 16 bits, incorporan desde 18 a 100 pines, tienen múltiples interrupciones, una memoria de programa de hasta 128KB, la pila tiene 32 niveles diferentes y además de incorporar las prestaciones de las anteriores gamas incluye bus CAN, Ethernet entre otras cosas.

2.3.2. Microcontroladores 16 bits

Los microcontroladores de 16 bits de memoria se dividen en cuatro grupos diferentes, según las características de cada uno. Los cuatro grupos comparten las siguientes cualidades [13]:

-Los periféricos, software y herramientas de desarrollo con comunes para los cuatro grupos.
-La memoria de programa tiene una capacidad de 4KB a 256KB.
-Encapsulados de 14 a 100 pines.
-Utilizan arquitectura Harvard modificada con un bus de instrucciones de 24 bits.
-Ejecución de instrucciones en un solo ciclo.
-Respuesta a las interrupciones determinista.

PICs 24F

Este grupo se caracteriza principalmente por su bajo consumo y bajo costo, utilizan una alimentación de 3.3V. Tienen una velocidad de 16MIPS y una memoria RAM de hasta 96KB.

PICs 24H

Los microcontroladores PIC24H tienen un alto rendimiento con una velocidad de 40MIPS y una alimentación de 3.3V.

DsPICs30

Estos microcontroladores son utilizados como controladores digitales de señal, utilizan una alimentación a 5V, una velocidad de hasta 30 MIPS.

DsPICs33

Los microcontroladores de esta gama son DSC de alto rendimiento, tienen una alimentación a 3.3V, hasta 40MIPS y se utilizan para el control de potencia y motores.

Después de ver las características de las diferentes gamas de los controladores PIC de microchip se ha decidido buscar placas de desarrollo que utilicen PICs de gama mejorada o alta de 8 bits o PIC24F de 16 bits.

2.3.3. Microcontroladores de 32 bits

Los microcontroladores de 32 bits son dispositivos con un rendimiento muy elevado, dentro de esta gama de microcontroladores hay una gran variedad de tamaños de memoria, de velocidad de procesamiento y permiten conectar periféricos analógicos y digitales avanzados, también tienen opción de conectar CAN, CAN FD, Hi-Speed/Full-Speed USB y Ethernet [14].

En la Ilustración 6 están representados los diferentes modelos de esta gama y para el uso que están diseñados.

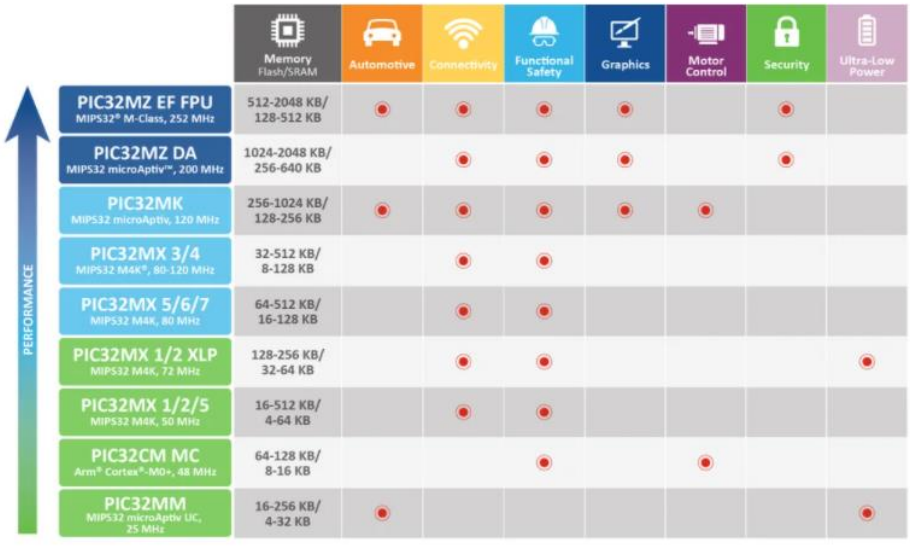


Ilustración 6: Modelos de PICs de 32 bits [14]

2.4. Microcontroladores de 8 bits

En el ámbito de la docencia se utilizan los microcontroladores de 8 bits de la gama media-mejorada y alta, el uso de este tipo de microcontroladores se debe a que son utilizados en la mayoría de las aplicaciones, son sencillos de programar y su precio es reducido [15].

Estos microcontroladores suelen incorporar una serie de módulos que permiten crear sistemas más complejos y con mayor funcionalidad. A continuación, se van a explicar los principales módulos que se suelen incorporar en los microcontroladores.

Puertos E/S

Los microcontroladores PIC pueden tener hasta 7 puertos paralelos, cada uno de estos puertos es nombrado con una letra A, B, C hasta G. Cada uno de estos puertos pueden tener hasta 8 bits, por lo que cada terminal de un puerto se numera desde 0 hasta 7.

Cada uno de los terminales de los puertos de un microcontrolador puede hacer diferentes funciones, ya sea entrada o salida digital, entrada analógica, portador de señal de entrada o salida a uno de los temporizadores…

Temporizadores

Estos módulos se utilizan en aplicaciones en las que el microcontrolador debe trabajar con la variable tiempo, como la medición de frecuencia o duración de una señal o la implementación de relojes para llevar la fecha y la hora.

Estos temporizadores se basan un contador síncrono de pulsos de una señal, que dependiendo del temporizador puede ser interna (del oscilador interno del microcontrolador) o de una señal externa.

Los temporizadores se diferencian unos de otros por la cantidad de pulsos que pueden llegar a contar, para elegir esta cantidad se asignan pre/post-divisor, también se diferencian en si tienen la capacidad para reiniciarse automáticamente o tener la opción de arranque/parada. Además, los temporizadores suelen tener asignada alguna interrupción.

Interrupciones

Las interrupciones funcionan del mismo modo que las subrutinas, sin embargo, las interrupciones se activan por medio de mecanismos hardware y se pueden producir en cualquier momento, por lo que se utilizan como conexión del sistema con los periféricos internos (Timers, CAD, PWM) y con el exterior (pulsadores).

Módulos CCP

Los módulos CCP tienen tres modos diferentes de funcionar que son modo de Captura, de Comparación y Modulación de Pulsos en Anchura(PWM).

-El modo captura guarda el valor del Timer que este configurado cuando ocurre un evento en el terminal del puerto correspondiente. Se suele utilizar para medir tiempos entre dos eventos o el periodo de una señal.

-El modo comparador genera una señal en el terminal del puerto cuando el valor del registro CCPRx es igual al del Timer utilizado para este caso.

-El modo modulador de pulsos en anchura (PWM) sirve para generar señales en las que se puede variar el periodo y el ciclo de trabajo. Este modo es de gran utilidad a la hora de controlar motores o regular la luminosidad de una bombilla.

Conversor analógico/digital

Los microcontroladores son dispositivos que funcionan con electrónica digital, por lo que si queremos utilizar señales analógicas hay que utilizar un convertidor analógico digital.

La conversión analógica/digital consta de 4 procesos:

1-Muestreo: se obtienen valores de la señal analógica cada un periodo determinado, si disminuimos el periodo de muestro se obtendrá más información de la señal, pero también el microprocesador deberá procesar mayor cantidad de información.

2-Retención: se mantiene el valor muestreado el tiempo necesario para la conversión.

3- Cuantificación: se produce la conversión de una señal continua en amplitud en una señal discreta. Con una mayor cantidad de bits de resolución se conseguirá tener una conversión de mayor calidad, ya que el error de cuantificación será menor.

4-Codificación: a cada uno de los valores de salida del cuantificador se le asigna una combinación binaria.

Conversor digital/analógico

Este conversor se utiliza cuando se quiere tener una salida analógica, ya que como se ha explicado anteriormente el microcontrolador solo trabaja con señales digitales.

Esta conversión requiere dos procesos:
-Conversión D/A: el convertidor transforma la información digital en una señal escalonada, la cantidad de niveles diferentes que puede tener la señal viene dada por el número de bits que tenga el convertidor.

-La señal escalonada pasa por un filtro para suavizar los cambios de valor.

Comparadores analógicos

Estos periféricos están basados en amplificadores operacionales, estos comparadores generan una señal digital ‘0’ o ‘1’ dependiendo de cuál de las tensiones de las dos señales introducidas es mayor.

-Comunicación serie (USART)

Este módulo permite al microcontrolador comunicarse con otros dispositivos. Dentro de la comunicación serie hay tres tipos de comunicación: *simplex, half duplex* y *full duplex* y se utiliza uno u otro dependiendo del dispositivo con el que se vaya a comunicar.

2.5. Evaluación de diferentes placas de desarrollo

Una placa de desarrollo es un dispositivo en el cual se encuentra un microcontrolador reprogramable que es capaz de ejecutar instrucciones de un sistema de control, estas placas suelen incorporar conexiones para salidas y entradas digitales y analógicas que permiten la comunicación del usuario con el sistema.

Las placas de desarrollo tienen instalados los circuitos adecuados para garantizar que los componentes reciban una alimentación adecuada y que el programa se cargue correctamente en el microcontrolador.

Además, en algunos modelos de placas de desarrollo llevan incorporados diferentes periféricos que permiten interactuar con el sistema, como pueden ser pulsadores, potenciómetros, leds entre otros, de esta forma se evita que el usuario tenga que conectarlos el mismo.

A continuación, se van a comparar diferentes placas de desarrollo para microcontroladores PIC de 8 bits que se encuentran actualmente en el mercado viendo las ventajas y desventajas que tiene cada una de ellas teniendo en cuenta que tienen que ser adecuadas para su uso en docencia y que los alumnos puedan adquirir una para poder completar su aprendizaje por su cuenta.

2.5.1. Curiosity High Pin Count 28/40 (HPC) (DM164136)

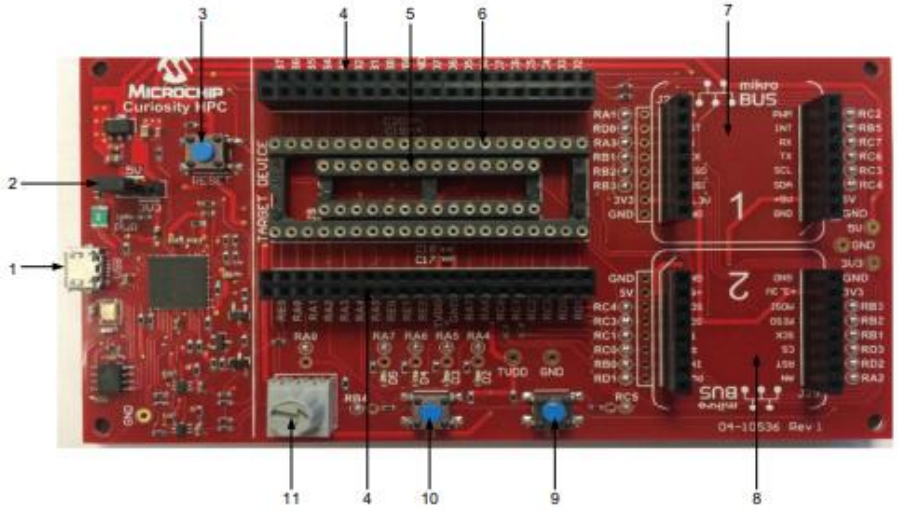


Ilustración 7: Curiosity HPC 28/40 [16]

Esta placa de desarrollo es una plataforma para microcontroladores de 8 bits con alto número de contactos, concretamente de 28 y 40 contactos. El precio de esta placa es de 33.06€. [16]

1- Conector micro USB a través del cual se carga el programa en el microcontrolador.
2- Jumper para escoger la tensión de alimentación entre 3.3V y 5V.
3- Pulsador de Reset.
4- Conectores de expansión de doble hilera.
5- Conector para microcontrolador de 28 pines
6- Conector para microcontrolador de 40 pines
7-8- Conector para mikroBUS 1 y 2
9-10- Pulsadores (RB5, RC4)
11- Potenciómetro (RA0)
12- Diodos led conectados al puerto A (RA4-RA7)

Con estos componentes que incluye esta placa de desarrollo se puede desarrollar un sistema con múltiples entradas y salidas tanto analógicas como digitales, a través del potenciómetro se puede variar el valor de una entrada analógica y los pulsadores sirven como interfaz con el usuario. Los diodos led del puerto A se pueden utilizar para que el usuario pueda observar cuando se activan diferentes salidas digitales. Y los conectores para mikroBUS permiten conectar diferentes módulos como sensores de temperatura, gas, convertidores A/D…

El microcontrolador que incluye esta placa de desarrollo es el PIC18F47Q10 de 40 pines, perteneciente a la gama alta de los microcontroladores PIC de 8 bits. Este microcontrolador tiene una arquitectura RISC que le permite compilar programas en C de forma optimizada, tiene una velocidad de operación máxima de 64MHz, por lo que es capaz de ejecutar un ciclo de instrucción en 62.5 ns. Además, se pueden programar dos niveles de prioridad de interrupción y 31 niveles de profundidad. Las diferentes memorias que tiene este microcontrolador son: una memoria de programa Flash de 128KB, 1024 bytes de memoria EEPROM y 3615 bytes de memoria de datos SRAM.

Este microcontrolador también cuenta con una serie de periféricos que permiten aprovechar mejor los ciclos de instrucción de la CPU. Estos periféricos son los siguientes:

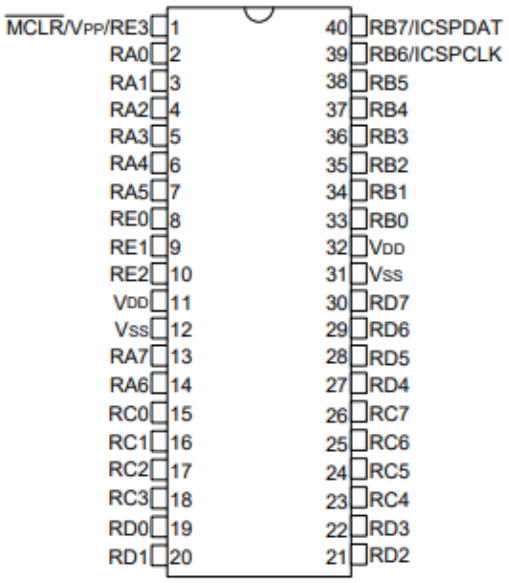


Ilustración 8:PIC18F47Q10 [16]

- 3 Timers de 8 Bits con HTL

- 4 Timers de 16 Bits

-Celdas lógicas configurables (CLC)

- Convertidor Analógico Digital de 10 Bits

- Convertidor Digital – Analógico de 5 Bits

- 2 Módulos CPPs con 16 bits de resolución para modo Captura/Comparación y 10 bits para PWM

- 2 Comparadores analógicos

- 2 conexiones USART

- 35 pines de entrada/salida.

- Módulo de selección de pines para los periféricos (PPS)

Las características del microcontrolador y de los periféricos se encuentran explicados con mayor profundidad en el datasheet del micro que está disponible en la página web de Microchip.

2.5.2. Curiosity Development Board (DM164137)

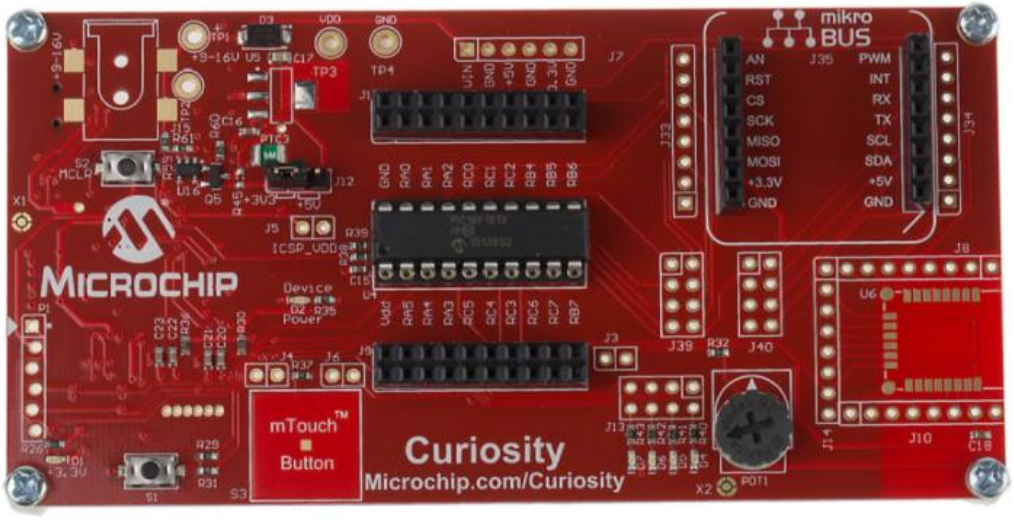


Ilustración 9:Curiosity Development Board [17]

Esta placa de desarrollo es una plataforma compatible con microcontroladores PIC de 8 bits de 8, 14 y 20 pines y tiene un precio de 22.17€. [17]

Esta placa cuenta con:

1- Conector micro USB a través del cual se carga el programa en el microcontrolador.
2- *Jumper* para escoger la tensión de alimentación entre 3.3V y 5V.
3- Pulsador de Reset.
4- Conectores de expansión de doble hilera.
5- Conector para microcontrolador de 20 pines.
6- Pulsador S1 (RC4)
7- Panel táctil mTouch Button S3 (RC1)
8- Diodos Leds D4-D7 (RA5, RA1, RA2, RC4)
9- Potenciómetro POT1 (RC0)
10- Conector mikroBUS
11- *Footprint* RN4020 Bluetooth

Esta placa al poder conectarse microcontroladores con un máximo de 20 pines, los sistemas implementados en ella deben tener un menor número de entradas y salidas. El panel táctil se puede utilizar como pulsador o como sensor de proximidad, el *footprint* del RN4020 Bluetooth permite incorporar con mayor facilidad el módulo Bluetooth, también al igual que la anterior placa incorpora el conector mikroBUS, diodos Leds y un potenciómetro.

Esta placa incluye el microcontrolador de 20 pines PIC16F18446, este microcontrolador PIC pertenece a la gama media-mejorada de la familia de 8 bits. La arquitectura que tiene este microcontrolador es del tipo RISC, que permite compilar archivos C óptimamente. Este microcontrolador incluye un oscilador interno capaz de funcionar a 32MHz, lo que significa que es capaz de ejecutar un ciclo de instrucción en 125 ns. Permite el uso de interrupciones con 16 niveles de profundidad e incorpora una memoria de programa Flash de 28KB, una memoria de datos SRAM de 2KB y una de 256B EEPROM.

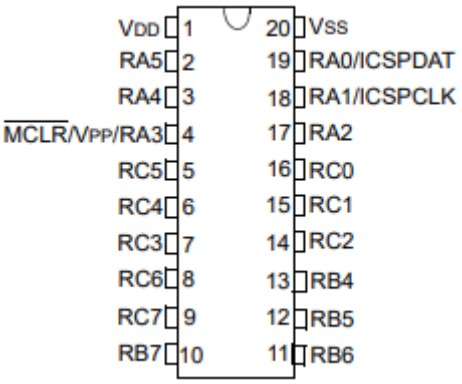


Ilustración 10: PIC16F18446 [17]

Los periféricos que vienen implementados en el microcontrolador son:

- 3 Timers de 8 bits

- 3 Timers de 16 bits

- 1 Timer configurable a 8/16 bits

- 18 pines de entrada/salida

- 4 celdas lógicas configurables (CLC)

- 1 puerto de comunicación EUSART

- 4 módulos CPPS con una resolución de 16 bits para los modos de Captura/Comparación y 10 bits para los PWM

- 2 módulos PWM

- Convertidor Analógico – Digital de 12 bits de resolución

- Convertidor Digital – Analógico de 5 bits de resolución

Al igual que el anterior microcontrolador, sus características y de los periféricos se encuentran explicados con mayor profundidad en el datasheet del microcontrolador que está disponible en la página web de Microchip.

2.5.3. PIC18F47Q10 Curiosity Nano

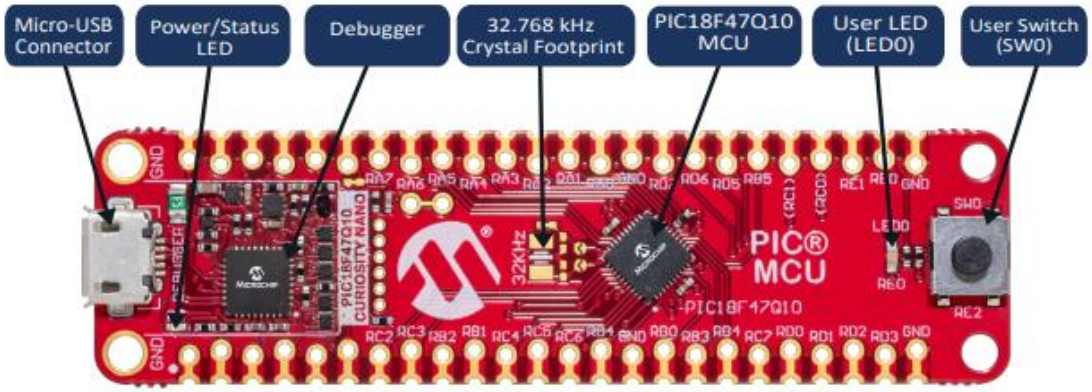


Ilustración 11:Curiosity Nano PIC18F48Q10 [18]

Este tipo de placas minimiza los componentes colocados en la placa para reducir notablemente su tamaño. Esta placa ya lleva integrado el PIC18F48Q10 perteneciente a la gama alta de los microcontroladores de 8 bits de memoria y tiene un precio de 12.31€. El microcontrolador que incorpora esta placa es el mismo que utiliza la placa de desarrollo Curiosity High Pin, por lo que las características principales se encuentran detalladas en la página 17.

Los componentes integrados que lleva la placa son los que están indicados en la Ilustración 11.

La gran ventaja de este tipo de placas respecto al resto es que al ser tan pequeñas se pueden utilizar en aplicaciones de poco tamaño, aunque si se quiere utilizar diferentes interfaces con el usuario como pueden ser pulsadores o leds, estos los tiene que colocar el usuario asegurándose de que la conexión es correcta, aunque desde otro punto de vista esto puede ser una ventaja ya que se pueden colocar en el lugar deseado según el tipo de aplicación.

2.5.4. Explorer 8 Development Kit (DM160228)

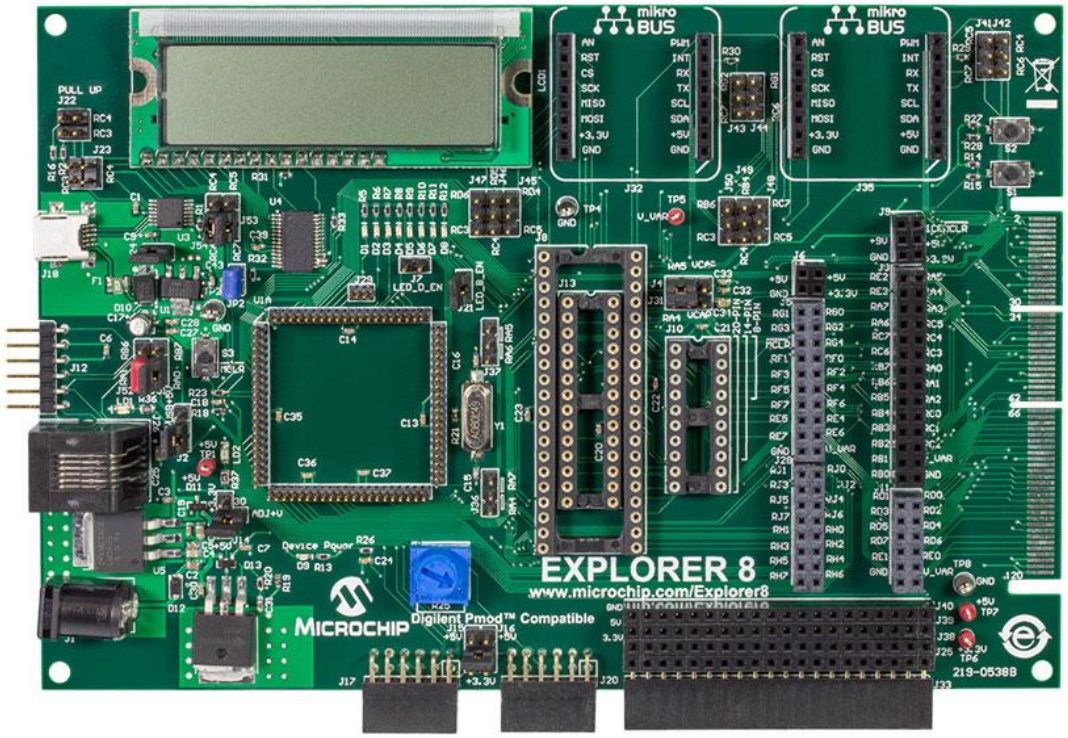


Ilustración 12: Placa Explorer 8 [19]

Esta placa de desarrollo para microcontroladores de 8 bits es compatible con modelos de 8/14/20/28/40 pines DIP y 44/64/80 pines PIM-Mouted, el precio de esta placa es de 69.43€. [19]

Esta placa incorpora los siguientes elementos:

-Pantalla LCD de 16x2
-Dos conectores mikroBUS
-Dos conectores Digilent Pmod.
-Convertidor USB-to-serial/I^2C
-Ocho leds azules D1-D8
-Dos pulsadores S1 y S2
-Puntos de Test a 5V y 3.3V
-Potenciómetro de 10K
-Botón de reset del MCLR
-Conector micro USB y USB

Esta placa de desarrollo no incluye ningún microcontrolador, por lo que en el caso de elegir esta placa de desarrollo habría que buscar uno compatible con esta placa y que se ajuste a las características deseadas.

En la siguiente tabla se van a anotar los elementos incorporados en cada una de las placas de desarrollo para posteriormente comparar dichas placas.

Placa de desarrollo	N.º Pines micro	PIC incluido	Conectores	N.º Leds	N.º Pulsadores	Potenciómetro	Conectores MikroBUS	Pantalla LCD	Precio (€)
Curiosity High Pin	28/40	PIC18F47Q10	Micro USB	4	2	1	2	No	33.06
Curiosity Development Board	14/20	PIC16F18446	Micro USB	4	2	1	1	No	22.31
Curiosity Nano	28/40	PIC18F47Q10	Micro USB	0	0	0	0	No	12.31
Explorer 8	8/14/20/28/40	-	Micro USB RJ11	8	2	1	2	Si	69.43

En esta tabla se van a comparar los dos PICs que incluyen los kits de las placas de desarrollo.

PIC	Frecuencia (MHz)	I/Os	Timers (8/16 bits)	ADC	DAC	CCP	PWM	PPS	EUSART
PIC18F47Q10	64	36	3/4	10 bits	1	2	2	SI	2
PIC16F18446	32	18	4/4	12 bits	1	4	2	SI	1

Tras observar y comparar las características de las diferentes placas de desarrollo expuestas anteriormente se ha llegado a las siguientes conclusiones:

- La placa Explorer 8 cuenta con una gran cantidad de componentes para poder comunicarse con el sistema en comparación con el resto de las placas, como son la pantalla LCD o los ocho leds que incorpora, permite conectar una gran variedad de tamaños de microcontroladores y tiene diferentes conectores (RJ11 y micro USB), a pesar de contar con todas estas prestaciones su elevado precio es un gran inconveniente para su uso en la docencia, ya que los estudiantes tendrían más difícil el acceso a su compra.
- La placa Curiosity Nano cuenta con la gran ventaja de su reducido precio y tamaño, aunque debido a estas características no tiene ningún elemento instalado, por lo que sería necesario que los alumnos instalasen los componentes necesarios en una *protoboard*.
- Las placas High Pin y Development Board de la gama Curiosity son muy parecidas, la primera está diseñada para microcontroladores de 28/40 pines y tiene dos conectores mikroBUS y la segunda permite la conexión de microcontroladores de 14/20 pines y cuenta con un conector de mikroBUS, un *footprint* del módulo *bluetooth* RN4020 y tiene un *MTouch Botom*. El precio de ambas se diferencia aproximadamente en 10€, siendo la placa High Pin más cara.

Teniendo en cuenta lo mencionado anteriormente se ha decido escoger la placa Development Board de la gama Curiosity (2.5.2), porque tiene un precio relativamente bajo (22.31€) e incorpora diferentes elementos ya conectados con los que se pueden realizar una gran variedad de sistemas. La placa High Pin permite conectar microcontroladores con un mayor número de pines, pero no incorpora ni el botón capacitivo ni la huella del conector *Bluetooth* y además su precio es mayor.

3. Entorno de programación

El entorno de programación que se va a utilizar para la realización de los programas es MPLAB X IDE, este *software* se puede descargar de forma gratuita en la página de Microchip (https://www.microchip.com/)[20].
Para descargar este *software* hay que dirigirse al menú "*Tools and Software*" y seleccionar MPLAB X IDE. En esta pestaña se encuentran diferentes versiones según el sistema operativo usado en el PC.

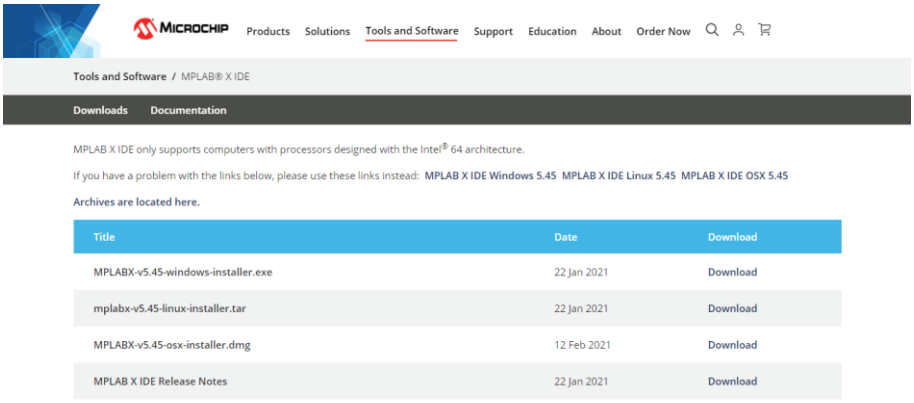


Ilustración 13:Entorno para descarga de MPLAB X

Después de descargar y ejecutar el archivo, se procederá a la instalación del programa. Para poder cargar los programas en los dispositivos es necesario instalar el compilador XC8, que se encuentra al igual que MPLAB X IDE en la página de Microchip en la pestaña de "*Tools and Software*" en la sección MPLAB XC *Compilers*.

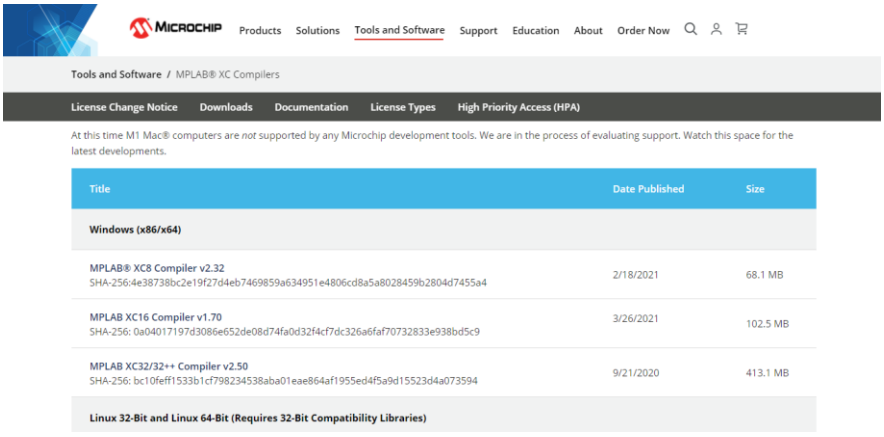


Ilustración 14:Entorno para descargar XC8

Por último, se descargará *MPLAB Code Configurator* (MCC), que se encuentra en la pestaña "*Tools and Software*" en el apartado "*Embedded Software Center*", gracias a este sistema de configuración gráfica no hace falta configurar todos los registros de forma manual, si no que configuramos a través de su entorno gráfico los diferentes módulos que tiene el microcontrolador y genera un archivo con todas las instrucciones necesarias.

Tras descargar estos archivos ya se tiene todo lo necesario para poder utilizar el entorno de programación MPLAB X IDE, al iniciar este programa nos encontraremos con la ventana que aparece en la Ilustración 15.

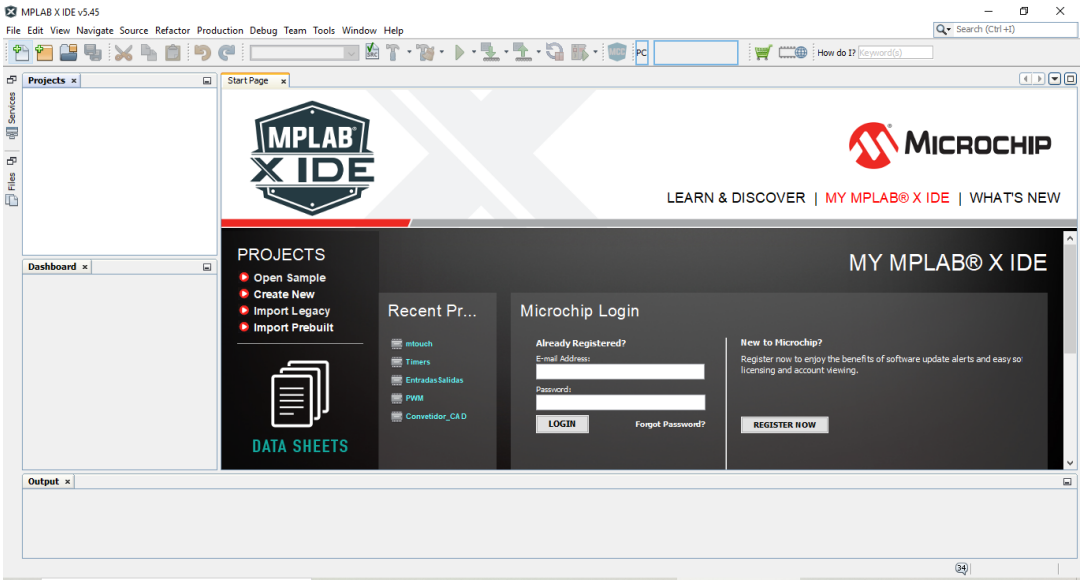


Ilustración 15: Ventana principal de MPLAB X

Esta ventana está dividida en diferentes partes que permiten trabajar de una forma más cómoda. Las diferentes partes son:

- Barra de menús y herramientas

Se encuentra situada en la parte superior y permite acceder a los diferentes menús e incorpora accesos rápidos para crear, compilar y simular proyectos y acceder al MCC.

- *Project*

Esta ventana se encuentra en la parte izquierda, en esta sección podemos acceder a todos los archivos que componen el proyecto.

- *Dashboard*

Esta sección está situada debajo de Project y permite ver las características del proyecto y las diferentes herramientas que se utilizan.

- *Output*

Se encuentra en la parte inferior y muestra las acciones que realiza el programa.

- *Start Page*

Esta ventana situada en la parte central nos permite acceder a proyector realizados anteriormente, crear nuevos proyectos, buscar datasheet, acceder a librerías y a la ayuda del programa.

3.1. Creación de proyecto

Para crear un proyecto hay que ir al menú *Files* → *New Project*, otra opción es el acceso rápido  . Tras realizar esta acción aparecerá la ventana de la Ilustración 16.

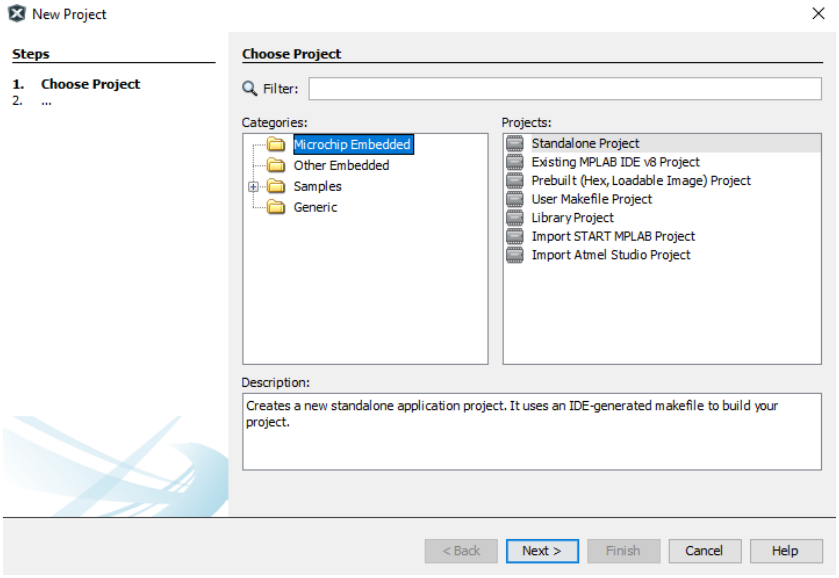


Ilustración 16: Ventana de creación de proyecto

A continuación, se elegirá la opción *Microchip Embedded* → *Standalone Project* y se hará clic en *Next >* y aparecerá la siguiente ventana de la Ilustración 17 en la cual se seleccionará el modelo de PIC, en este caso es el PIC16F18446 y en *Tool* se escogerá Curiosity Starter Kits, para que aparezca esta opción es necesario tener conectada la placa de desarrollo al ordenador a través del puerto USB.

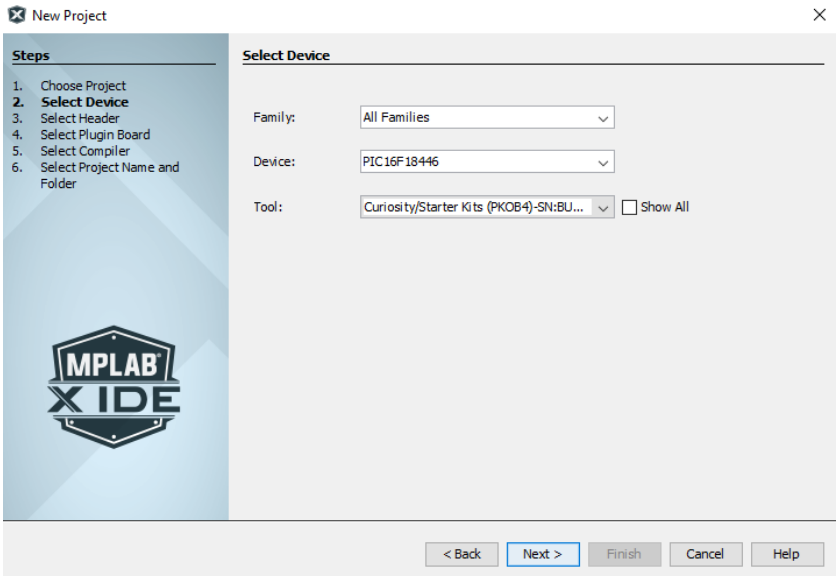


Ilustración 17: Ventana de selección de PIC

Después de seleccionar el PIC y el modo con el que vamos a cargar el programa, se escogerá el Compilador XC8 y por último se creará la carpeta en la que se quiere guardar el proyecto.

3.2. MPLAB Code Configurator

Después de crear el proyecto, se abrirá el MCC para configurar el sistema y los módulos que van a intervenir en el proyecto.

El *MPLAB Code Configurator* nos permite modificar los registros de los diferentes módulos, pero también es necesario mirar el datasheet del microcontrolador para saber su configuración y las posibilidades que ofrece cada uno de ellos.

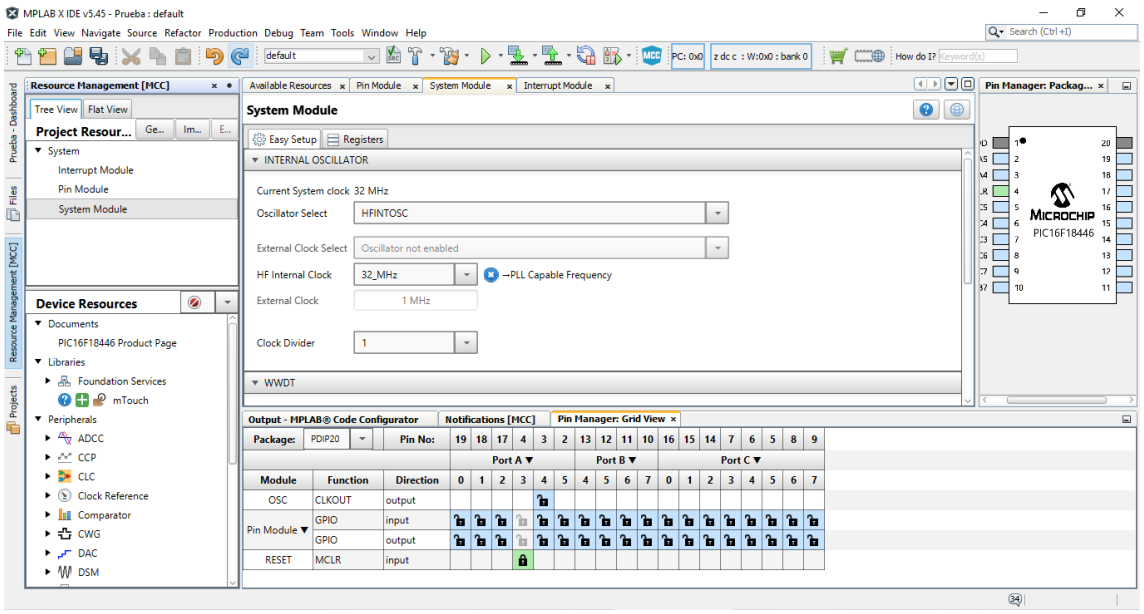


Ilustración 18: Ventana de inicio de MCC

La ventana de inicio del MCC (Ilustración 18) está compuesta a su vez por varias ventanas:

- *Project Resources*

Esta ventana está situada a la izquierda de la pantalla, desde ella podemos acceder a los recursos del sistema que se está diseñando, como son la configuración de interrupciones (*Interrup Module*), la configuración de los pines (*Pin Module*) y el módulo de funcionamiento del sistema (*System Module*).

- *Device Resources*

Se encuentra en la parte inferior izquierda, debajo de *Project Resources.* Desde esta ventana se seleccionan los diferentes periféricos que se van a utilizar en el sistema.

- Ventanas de configuración

Estas ventanas ocupan la parte central y derecha. A través del *Pin Manager* se seleccionan los pines que se van a usar en cada módulo y en la ventana *Pin Manager*: *Grid View* se observa la distribución de estos pines. En la parte central aparece la ventana seleccionada en *Project Resources*.

Después de realizar la configuración de los distintos módulos y pines hay que hacer clic en el botón *Generate* que se encuentra en *Project Resources* para que MCC cree los archivos que se implementarán en el proyecto con todas las configuraciones, además MCC genera un archivo “main.c” en el que se escribirá el programa.

3.3. Realización del programa

Para escribir las instrucciones del programa hay que abrir el archivo “main.c” que se ha generado anteriormente, este archivo se encuentra en la ventana *Projects* en la carpeta *Source Files*.

En este archivo están incluidas todas las configuraciones del MCC gracias a la instrucción *#include "mcc_generated_files/mcc.h".* En las librerías que se encuentran en carpeta “*MCC Generate Files*” que está a su vez en la carpeta “*Headers Files*” están disponibles todas las instrucciones asociadas a los periféricos configurados.

Cuando se acabe de escribir el programa habrá que darle al botón  para compilar el programa y comprobar que no hay ningún error, si hay algún error las instrucciones de programa aparecerá en la ventana *Output*.

En el momento que no haya ningún error y el programa compile correctamente se puede volcar a la placa de desarrollo a través del botón. 

4. Programas básicos

En este apartado se van a explicar paso a paso diferentes programas en los que se utilizan los periféricos del microcontrolador y los de la placa base para aprender su funcionamiento.

4.1. Entradas y salidas digitales

El primer programa va a controlar las salidas y entradas digitales del microcontrolador, en este caso la entrada digital será el pulsador S1 conectado al puerto RC4 y las salidas serán los leds LD4-LD7 conectados a los puertos RC5, RA1, RA2, RA5, cada vez que se accione el pulsador se encenderá un nuevo led hasta que estén los cuatro encendidos, cuando estén todos activados un nuevo accionamiento del pulsador apagará los leds y volverá al estado inicial. El esquema del funcionamiento está representado en la Ilustración 19.

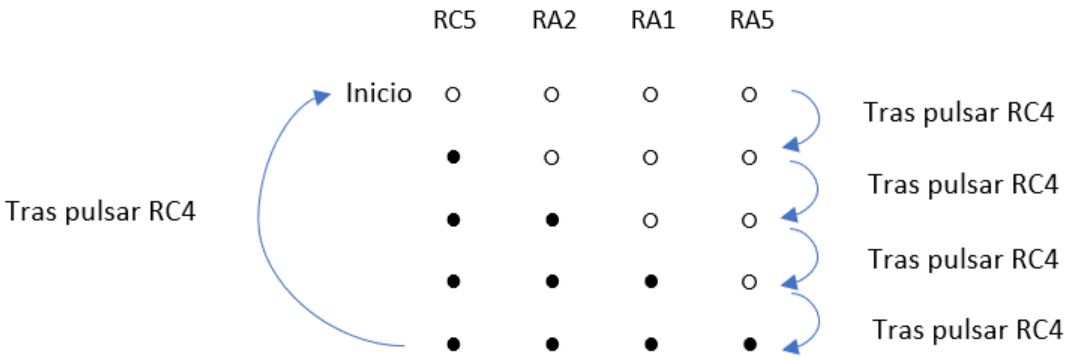


Ilustración 19: Esquema de funcionamiento del ejemplo 1

- Puertos de entrada/salida

Los puertos de entrada/salida actúan de una forma u otra según se desee. En la Ilustración 20 están representados los registros principales que intervienen en la configuración. El registro PORTx sirve para escribir/leer el valor del terminal, el registro LATx se utiliza para escribir/leer el valor en la memoria. Para configurar los terminales de los puertos se utiliza el registro TRISx, si se le asigna un '1' a un terminal con el registro TRIS, este se programará como entrada y si se le asigna un '0' el terminal se programa como salida. Después de un reset, el registro TRIS tiene todos sus bits a '1', por lo que todos los terminales están configurados como entradas por motivos de seguridad, ya que si un terminal está configurado como salida y se le conecta una entrada con diferente voltaje puede aparecer una sobre corriente que dañe el terminal. El último registro que interviene es el ANSELx, al poner este registro a '1' el terminal se configura como entrada analógica (este valor es el que tiene por defecto) y un '0' como digital.

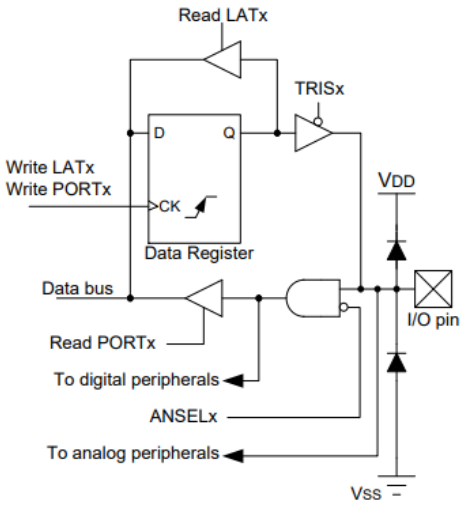


Ilustración 20: Puertos entrada/salida [21]

- Implementación del ejemplo

Tras abrir MPLAB X IDE, se creará un proyecto como se ha explicado en el apartado 3.1. Después se abrirá el MCC, con el que se ajustará la velocidad del oscilador interno a 8 MHz (Ilustración 21).

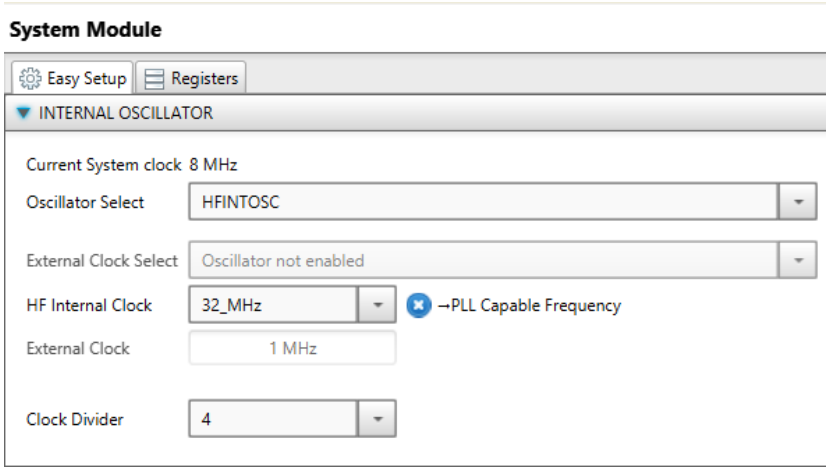


Ilustración 21: Configuración del oscilador interno

Después de ajustar el oscilador, se utilizará el Pin Module de la ventana *Project Resources* para configurar la entrada (RC4) y salidas (RC5, RA2, RA1, RA5) digitales como está indicado en la Ilustración 22.

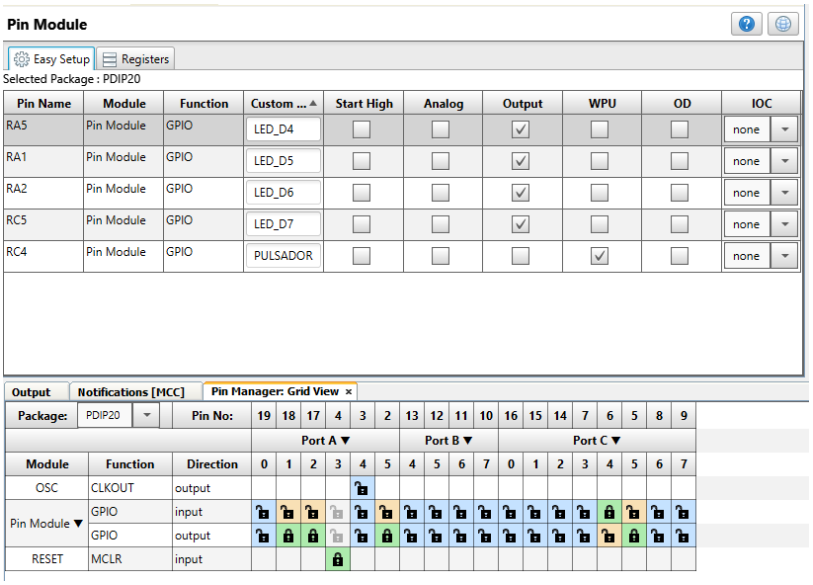


Ilustración 22: Configuración del Pin Manager

En la columna Custom Name se le puede asignar el nombre que se quiera a los pines, en este caso para facilitar la programación, a las salidas se les ha dado el nombre de los leds que están conectados a ellas y a la entrada “PULSADOR”.

En el pin RC4 se ha activado la casilla de WPU para habilitar las resistencias de pull-up.

Como se ha explicado en la página 28, se generarán los archivos necesarios para la realización del programa. A la hora de escribir el programa se utilizarán las funciones LEDs_SetHigh(), LEDs_SetLow() y PULSADOR_GetValue() que vienen recogidas en la librería “pin_manager.h”. Estas funciones sirven para modificar el estado de los leds y para saber el estado del pulsador, si está activado la entrada será ‘0’ y si no ‘1’. En la Ilustración 23 se encuentra representado el código del programa.

```
44  #include "mcc_generated_files/mcc.h"
45
46  /*...3 lines */
49  int contador=0;
50  void main(void)
51      {SYSTEM_Initialize();
52      LED_D4_SetLow();LED_D5_SetLow();LED_D6_SetLow();LED_D7_SetLow();
53      while (1)
54          {if(PULSADOR_GetValue()==0){
55              __delay_ms(200);
56              contador++;
57              switch(contador){
58                  case 1:LED_D7_SetHigh();
59                      break;
60                  case 2:LED_D6_SetHigh();
61                      break;
62                  case 3:LED_D5_SetHigh();
63                      break;
64                  case 4:LED_D4_SetHigh();
65                      break;
66                  case 5:
67                      LED_D4_SetLow();LED_D5_SetLow();LED_D6_SetLow();LED_D7_SetLow();
68                      contador=0;
69                      break;
70              }
71      }
72  }
```

Ilustración 23: Código del programa 1

Al inicio del programa se inicializa un contador que contará las veces que se activa el pulsador, más adelante se configuran todos los leds para que estén apagados al empezar el programa. La sentencia iterativa "*if"* se utiliza para comprobar el estado del pulsador, cuando este se activa hay un retardo de 200ms para evitar errores debido a los rebotes y se incrementa en uno el valor de contador y se compara con los casos que hay dentro del "*switch*".

- Comentarios sobre el programa

Aunque se ha utilizado la función "__delay_ms()" para evitar los rebotes, si se mantiene pulsado S1 los leds se encienden cada 200ms. Una solución para este problema puede ser la utilización de una función que detecte cuando se ha dejado de pulsar, y hasta que esto no se cumpla no contar una nueva pulsación. Otra solución posible es utilizar algún periférico como Timers o interrupciones al cambio de nivel como se va a ver en el siguiente en el ejemplo.

4.2. Interrupciones de cambio de nivel

En este ejemplo se va a explicar cómo configurar las interrupciones por cambio de nivel (IOC) para detectar cuando se ha pulsado el pulsador S1 conectado al terminal RC4 y los leds van a cambiar de la misma forma que en el primer ejercicio y así evitar los rebotes.

- Interrupciones de cambio de nivel

Todos los pines del microcontrolador PIC16F18446 pueden ser configurados para detectar este tipo de interrupciones, estos pines tienen que ser configurados como entradas digitales y se pueden configurar para detectar flancos de bajada, de subida y ambos.

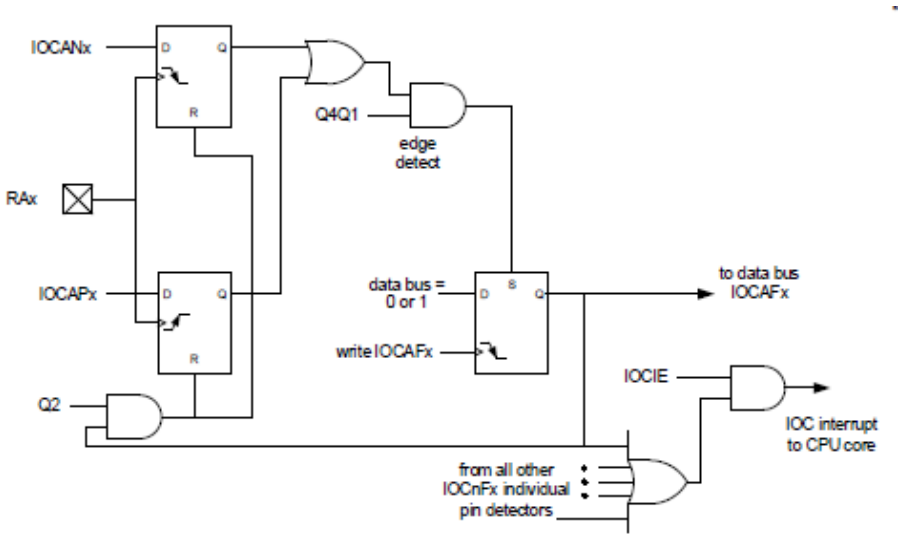


Ilustración 24: Esquema de funcionamiento de IOC [21]

Para poder utilizar este tipo de interrupciones es necesario habilitar las interrupciones globales (GIE=1) y periféricas (PEIE=1) y el IOCIE=1 del registro PIE0. Después de activar estos registros es necesario configurar el flanco que queremos que se detecte, para elegir el tipo de flanco hay dos registros por cada puerto del microcontrolador, el registro IOCxP para detectar los flancos de positivos, el IOCxB para los flancos negativos, para detectar los dos tipos de flancos es necesario habilitar los dos registros.

Al producirse un cambio de nivel ya sea positivo o negativo el registro IOCxF se pone a '1' y tras atender a esta interrupción es necesario volver a poner el registro IOCxF a '0' para cuando suceda una nueva interrupción el microcontrolador atienda la interrupción correspondiente.

- Implementación del ejemplo

Para realizar este ejemplo crearemos un nuevo proyecto y abriremos el MCC para configurar el sistema. Primero seleccionaremos una frecuencia de oscilador de 8MHz como aparece en la Ilustración 21 del primer ejercicio.

Después, en el Pin Manager configuraremos el pin RC4 como entrada digital y todos los leds de la placa como salidas digitales. Al utilizar el MCC no es necesario configurar manualmente los registros de las interrupciones, en el *Pin Module* en la columna IOC se

configura el tipo de interrupción al cambio queremos habilitar. En este caso como el pulsador tiene conectada una resistencia de *pull-up*, el nivel de señal que llega al terminal RC4 cuando no está pulsado es de 5V y cuando se encuentra pulsado es de 0V, por lo que para detectar la pulsación necesitaremos una interrupción por flanco negativo. Si se quiere detectar cuando se deja de pulsar habría que configurar una interrupción por flanco positivo y si se quiere detectar en ambos casos se seleccionará la opción "*any*". En la Ilustración 25 se encuentra la configuración necesaria para este ejemplo.

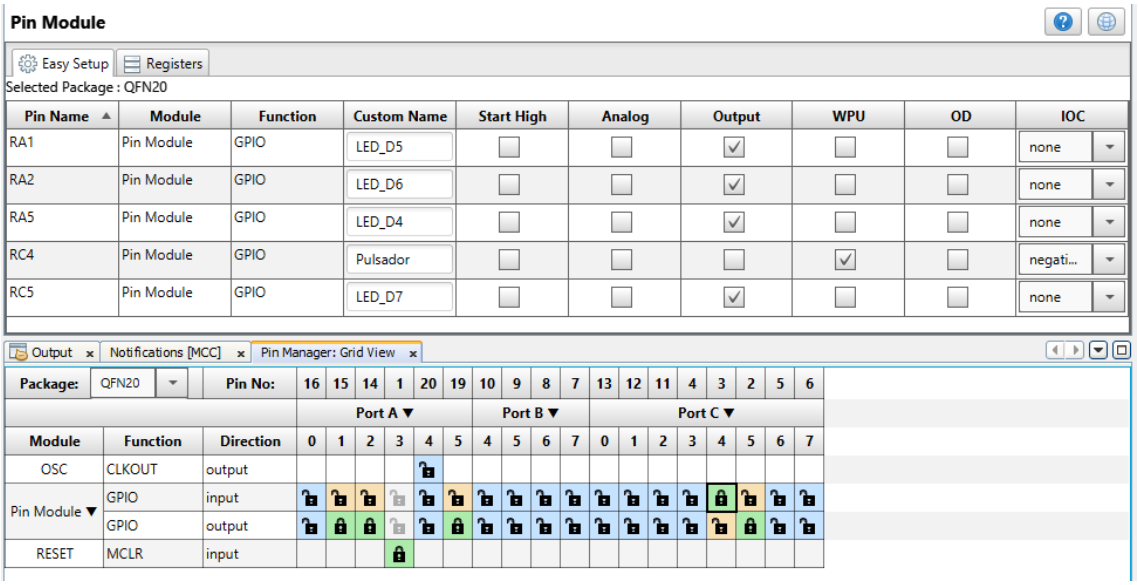


Ilustración 25: Pin Module del ejercicio 2

En la ventana de *Interrupt Module* podemos comprobar que las interrupciones por cambio de nivel están habilitadas. Tras acabar la configuración, pulsaremos en el botón *Generate* del *MPLAB Code Configurator* (MCC) para crear los archivos necesarios para la elaboración del proyecto.

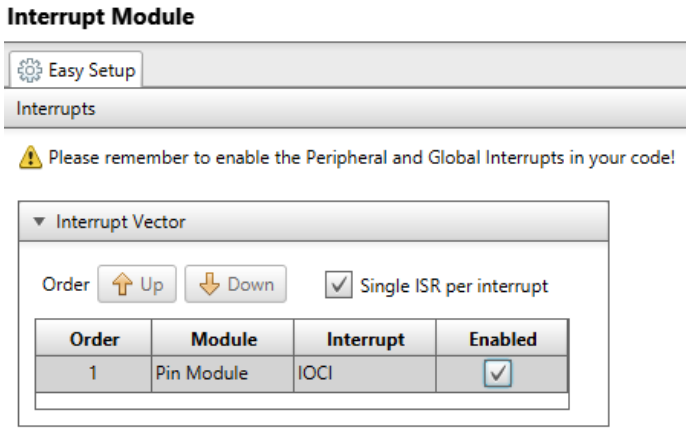


Ilustración 26: Interrup Module del ejercicio 2

Para poder modificar una variable a través de una interrupción es necesario definir esta variable como "*volatile*" y el tipo de variable que sea, (*int, char, string*…), además crearemos una función a la que se llamará cuando se produzca la interrupción para que

modificar la variable. En este caso definiremos una variable "*volatile int*" ya que la variable a modificar es un contador y la inicializaremos a cero. Y crearemos una función llamada "modo" que incremente en uno la variable cada vez que se ejecute.

Para que se ejecute la función "modo" es necesario escribir en el programa principal IOCCF4_SetInterruptHandler(modo), esta función asocia a la función "modo" un puntero (dirección de memoria) y permitir llamar a esa función al producirse la interrupción. Además, habilitaremos las interrupciones globales y periféricas. El código del programa está representado en la Ilustración 27. Tras escribir estas líneas de programa ya se puede volcar a la placa de desarrollo.

```
#include "mcc_generated_files/mcc.h"
#include "mcc_generated_files/interrupt_manager.h"
/*...3 lines */
volatile int contador=0;
void modo(void){
    contador++;
}
void main(void)
{
    SYSTEM_Initialize();
    INTERRUPT_GlobalInterruptEnable();
    INTERRUPT_PeripheralInterruptEnable();
    IOCCF4_SetInterruptHandler(modo);
    while (1)
    {
        switch(contador){
            case 1:LED_D7_SetHigh();
                break;
            case 2:LED_D6_SetHigh();
                break;
            case 3:LED_D5_SetHigh();
                break;
            case 4:LED_D4_SetHigh();
                break;
            case 5:
                LED_D4_SetLow();LED_D5_SetLow();LED_D6_SetLow();LED_D7_SetLow();
                contador=0;
                break;
        }
    }
}
```

Ilustración 27: Código del programa 2

Para poder entender mejor el funcionamiento de las interrupciones abriremos el archivo *"interrup_manager.c"* (Ilustración 29). Al abrir este archivo nos encontraremos una ventana como la de la , cuando sucede una interrupción se ejecuta este archivo en el que se comprueba el origen de la interrupción, en este caso comprueba si las interrupciones por cambio de nivel están activadas (IOCIE=1) y si el *flag* IOCIF se encuentra activado, si es el caso llama a la función PIN_MANAGER_IOC. Para poder seguir las instrucciones que ejecuta el programa le haremos CTRL+Clic en la función PIN_MANAGER_IOC y MPLAB nos llevará al código de esta función (Ilustración 30).

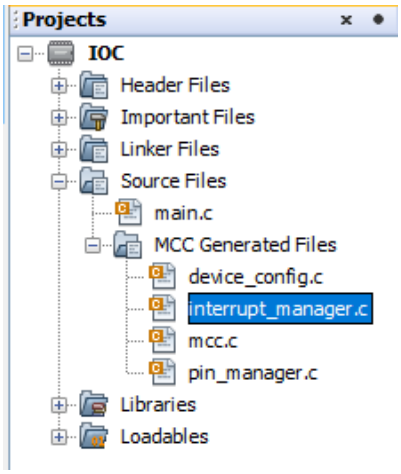


Ilustración 28: interrupt_manager.c

Ilustración 29: Ubicación interrupt_manager.c

Ilustración 30: Función PIN_MANAGER_IOC

```
129  void PIN_MANAGER_IOC(void)
130  {
131          // interrupt on change for pin IOCCF4
132      if(IOCCFbits.IOCCF4 == 1)
133      {
134          IOCCF4_ISR();
135      }
136  }
137  /**
138     IOCCF4 Interrupt Service Routine
139  */
140  void IOCCF4_ISR(void) {
141      // Add custom IOCCF4 code
142      // Call the interrupt handler for the callback registered at runtime
143      if(IOCCF4_InterruptHandler)
144      {
145          IOCCF4_InterruptHandler();
146      }
147      IOCCFbits.IOCCF4 = 0;
148  }
```

La función PIN_MANAGER_IOC comprueba que terminal ha provocado la interrupción y llama a la función de interrupción de ese terminal. En la función interrupción IOCCF4_ISR se puede escribir el código que queremos que se ejecute cuando se produce la interrupción, aunque en este llamaremos a una función del programa principal. Después comprueba si se ha asignado el puntero IOCCF4_InterruptHandler a alguna función y si es el caso ejecutará la línea de memoria de ese puntero, en nuestro caso, esa línea de memoria corresponde a la función “modo”. Al finalizar la atención a la interrupción desactiva el *flag* IOCCF4 para poder atender de nuevo a la interrupción si se vuelve a producir.

- Comentarios sobre el programa

Gracias a este ejemplo se ha podido explicar el funcionamiento de las interrupciones por cambio de nivel y como modificar una variable del programa principal cuando sucede una interrupción. Este tipo de programas se pueden aplicar en la detección de

sucesos externos como la activación de un final de carrera de una puerta o la de un sensor digital

4.3. Timers y sus interrupciones

En este ejemplo se va a explicar el funcionamiento de los Timers y sus interrupciones, para ello se va a elaborar un programa que haga parpadear un led cada segundo.

- Timer de 16 bits

El microcontrolador PIC16F18446 tiene tres *timers* de 8 bits (TMR2, TMR4 y TMR6), tres de 16 bits (TMR1, TMR3 y TMR5) y uno configurable a 8/16 bits (TMR0). En este ejemplo se va a utilizar el TMR1, cuyo esquema de funcionamiento se encuentra la Ilustración 31.

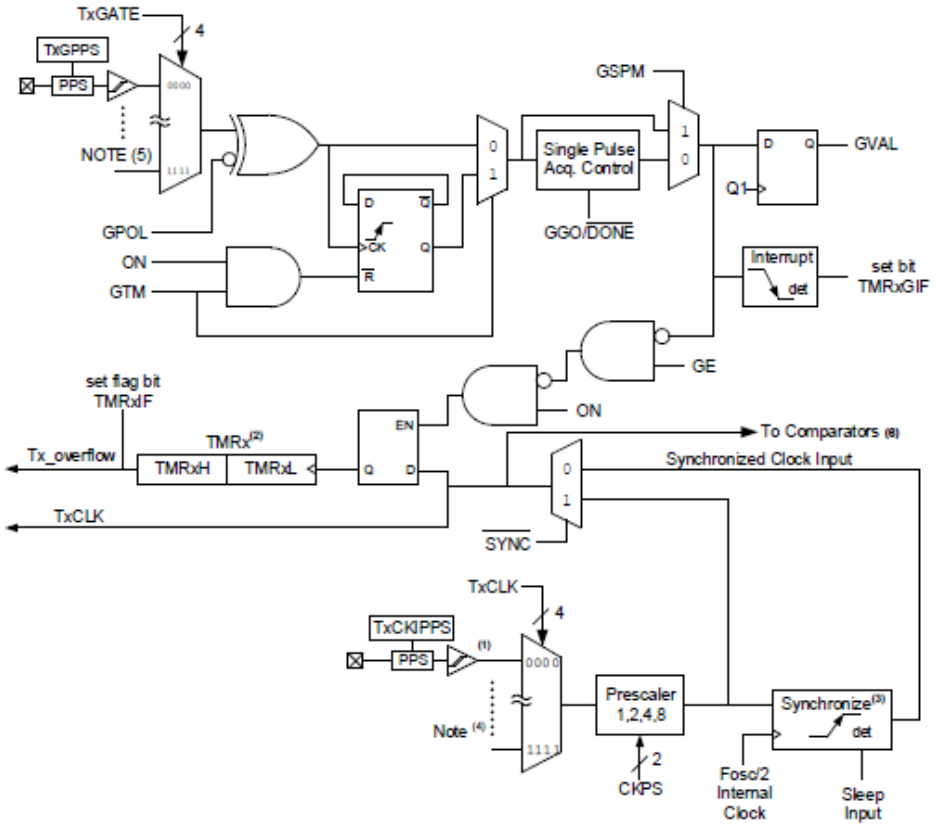


Ilustración 31:Esquema de funcionamiento de los Timers de 16 bits [21]

En este esquema de funcionamiento observan los principales registros que intervienen en la configuración del TMR1. Los registros TMRxH y TMRxL forman parte de un contador de 16 bits, el primero almacena los 8 bits más significativos y el segundo los 8 bits menos significativos, este contador cuenta con el predivisor CKPS de 3 bits, que es otro contador que utiliza una entrada de reloj configurable TxCLK, cada vez que alcanza el valor asignado se reinicia y produce un pulso que incrementa el contador del TMR1, cuando este llega al valor especificado el bit TMRxIF se activa.

Para calcular el tiempo que hay entre el comienzo del contador hasta que se activa TMRxIF se utiliza la siguiente ecuación.

$$T = T_{TxCLK} * Pred * (TMRxH + TMRxL + 1)$$

- Interrupciones

Las interrupciones funcionan del mismo modo que las subrutinas, sin embargo, las interrupciones se activan por medio de mecanismos hardware y se pueden producir en cualquier momento, por lo que se utilizan como conexión del sistema con los periféricos internos (Timers, CAD, PWM) y con el exterior (pulsadores).

Si se quiere utilizar las interrupciones en necesario habilitar el bit de interrupciones globales GIE (*Global Interrupt Enable*) y el bit PEIE (*Peripheral Interrupt Enable*) si las interrupciones están asociadas a los periféricos. A parte de estos bits, también es necesario activar los bits de los periféricos para habilitar sus interrupciones.

Cuando sucede una interrupción se activa el bit xIF del periférico que ha generado la interrupción, GIE se pone a '0' para evitar que se produzca otra interrupción durante la gestión de una anterior y el programa ejecuta la rutina de atención a la interrupción (RAI), en esta rutina es necesario comprobar que periférico ha producido la interrupción y después de responder a una interrupción hay que reiniciar los bits xIF para poder detectar una nueva interrupción.

- Implementación del ejemplo

En este ejemplo también se va a utilizar el MCC para configurar el microcontrolador y sus periféricos.

Tras crear un nuevo proyecto, se abrirá el MCC y se configurará la frecuencia del reloj a 8MHz de la misma forma que en la Ilustración 21 y se añadirá el TMR1 al proyecto desde la ventana *Device Resources* (Ilustración 32).

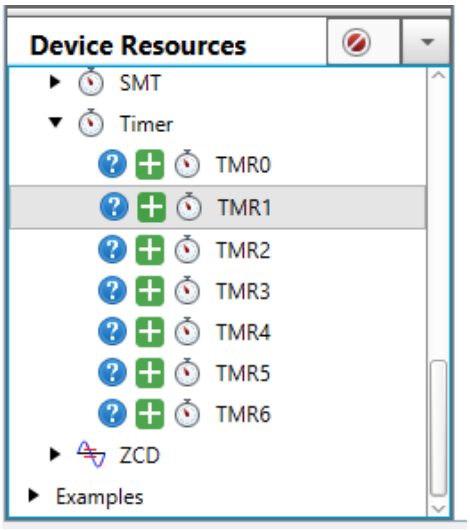


Ilustración 32: Ventana Device Resources

Tras añadir el TMR1 al proyecto se abrirá la ventana de configuración del timer, en la cual se seleccionará Fosc/4 como señal de entrada con un predivisor de 8 para tener un rango de temporización de 4us a 262.144ms, dentro de este rango se escogerá 250ms y se habilitará las interrupciones del temporizador. En la Ilustración 33 se observa la configuración final del TMR1.

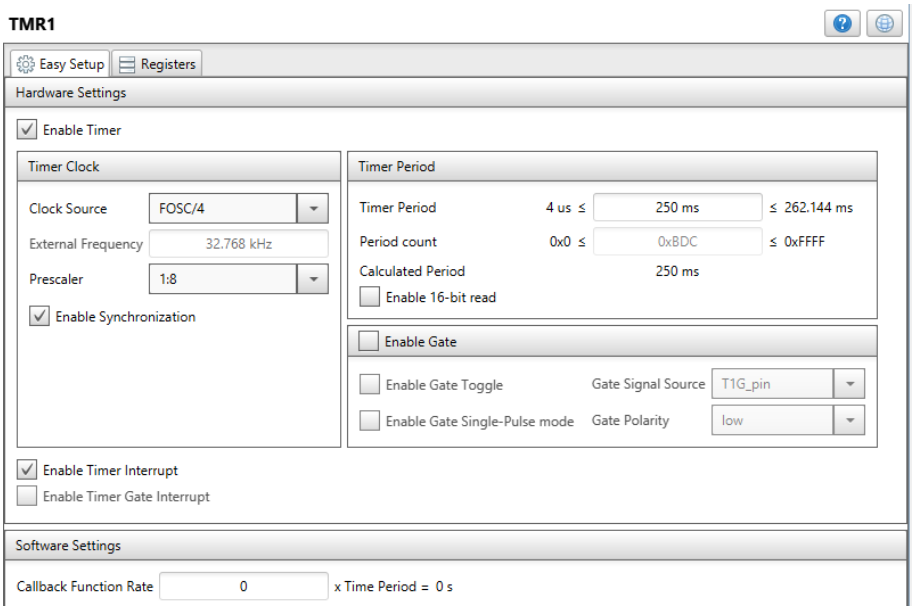


Ilustración 33: Configuración TMR1

Después de configurar el TMR1, abriremos el *Pin Module* para configurar el puerto RC5 como salida digital y le asignaremos el nombre de "LED" y le daremos al botón Generate para que MCC cree los archivos con la configuración del programa y abriremos el archivo "main.c" de la carpeta Source Files.

En este archivo se habilitarán las interrupciones globales (GIE) y las periféricas (PEIE), para ello basta con eliminar las "/" que hay delante de INTERRUPT_GlobalInterruptEnable(); y INTERRUPT_PeripheralInterruptEnable();. Además, se incluirá el archivo *"interrupt_manager.h"* que contiene las funciones de las interrupciones, si no se añade este archivo el programa no podrá encontrar esas funciones (Ilustración 34).

```
43
44  #include "mcc_generated_files/mcc.h"
45  #include "mcc_generated_files/interrupt_manager.h"
46  void main(void)
47  {
48      // initialize the device
49      SYSTEM_Initialize();
50      // When using interrupts, you need to set the Global and Peripheral Interrupt Enable bits
51      // Use the following macros to:
52      // Enable the Global Interrupts
53      INTERRUPT_GlobalInterruptEnable();
54      // Enable the Peripheral Interrupts
55      INTERRUPT_PeripheralInterruptEnable();
56      while (1)
57      {
58          // Add your application code
59      }
60  }
61  /**
62   End of File
63  */
```

Ilustración 34: Código del programa 3

En este archivo no hay que añadir ninguna instrucción para encender y apagar el led, estas instrucciones se escriben en el archivo "tmr1.c" que se encuentra en la carpeta *Sources Files* dentro de *MCC Generate Files*. En este archivo se encuentran todas las funciones asociadas al TMR1, la función en la que se escribe el código que se ejecutará

al producirse la interrupción es TMR1_DefaultInterruptHandler. Para poder utilizar la función LED_Toggle() es necesario incluir la librería “pin_manager.h”

```
51  #include <xc.h>
52  #include "tmr1.h"
53  #include "pin_manager.h"
54  /*...3 lines */
57
58  int contador=0;
59  void TMR1_DefaultInterruptHandler(void){
60      contador++;
61      if (contador==4){
62          LED_Toggle();
63          contador=0;
64      }
65  }
66  /*...3 lines */
69
```

Ilustración 35: Instrucciones de la interrupción del TMR1

En este programa se utiliza un contador porque el tiempo máximo que permite configurar el TMR1 es inferior al deseado, por ello se ha configurado el TMR1 a 250ms, el contador registra las veces que se desborda el TMR1 y cada cuatro desbordamientos (4 x 250ms = 1s) el led cambia de valor y el contador se reinicia. En el caso de querer cambiar el periodo de parpadeo basta con cambiar el contador o el tiempo del TMR1.

Ahora el programa ya está listo para compilarlo y volcarlo a la placa.

- Comentarios sobre el programa

Este programa permite comprender de una manera sencilla el funcionamiento de los temporizadores y sus interrupciones, en este caso se ha utilizado la interrupción por desbordamiento de un temporizador de 16 bits cambiar el estado de un led cada 1s. Este diseño se puede aplicar a otros casos como puede ser la comprobación del valor de una entrada cada cierto periodo de tiempo.

4.4. Conversor Analógico Digital (CAD)

A continuación, se va a explicar la configuración del conversor analógico digital a través de un programa que encienda los leds D4-D7 de tal forma que representen los cuatro bits más significativos del CAD.

- Conversor analógico digital

Los microcontroladores son dispositivos que funcionan con electrónica digital, por lo que si queremos utilizar señales analógicas hay que utilizar un convertidor analógico digital.

El conversor analógico digital que incorpora este microcontrolador convierte una entrada analógica en una digital de 12bits, este conversor permite hasta 17 entradas analógicas.

En la Ilustración 36 está representado el funcionamiento del CAD, mediante el registro PCH se elige el puerto de entrada analógico, entre todas las entradas disponibles se encuentra el sensor de temperatura que incorpora el propio microcontrolador, con el registro PREF se selecciona la señal de referencia positiva y con el NREF la negativa y en los registros ADRESH y ADRESL se guarda el resultado de la conversión. Estos son algunos de los registros más importantes, para poder conocer con más detalle los diferentes registros que intervienen en la configuración del CAD es necesario acudir al *datasheet* del microcontrolador.

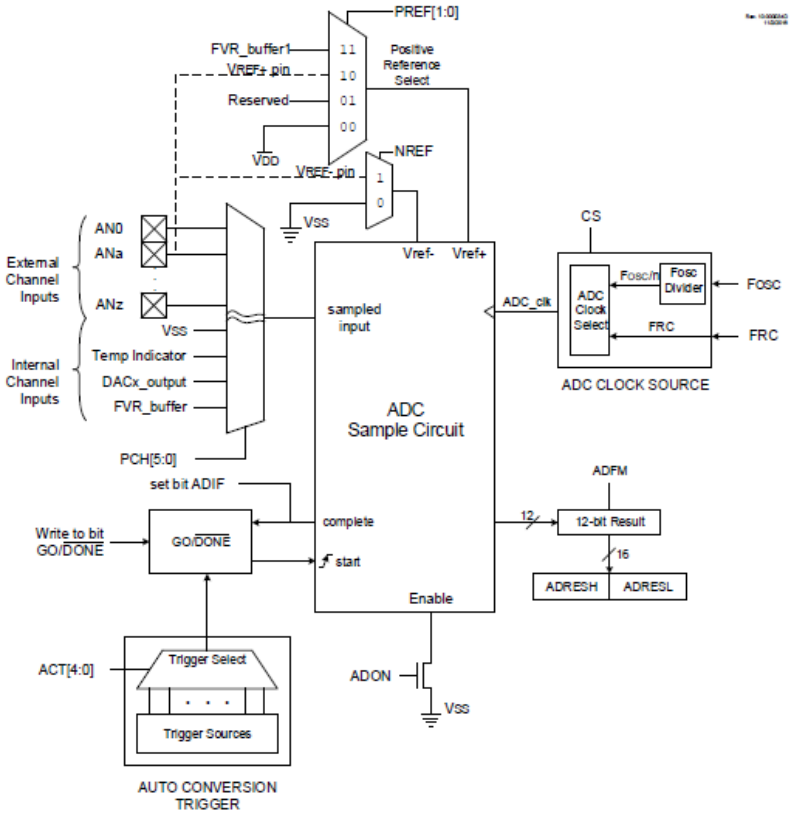


Ilustración 36: Esquema funcionamiento del CAD [21]

- Implementación del ejemplo

Para comenzar el ejemplo se creará un proyecto como se ha explicado anteriormente y abriremos el MCC para configurar el CAD y las salidas digitales.

Lo primero que realizaremos al abrir el MCC será configurar la frecuencia del reloj a 8MHz, después añadiremos al proyecto el CAD desde la ventana *Device Resources* y para su configuración.

En la ventana del CAD seleccionaremos modo básico de operación y Frc (reloj interno del conversor) como reloj del conversor. Además, se fijarán las referencias a VDD (5V) y VSS (0V) y se alineará el resultado a la izquierda para que el resultado se almacene en los registros ADRESH y ADRESL como en la Ilustración 37.

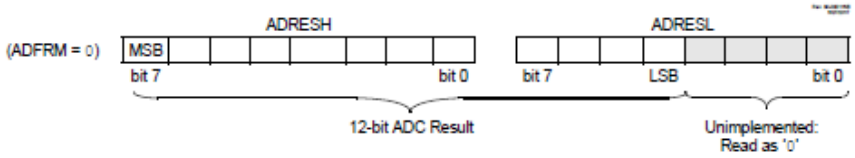


Ilustración 37: Registros ADRESH y ADRESL [21]

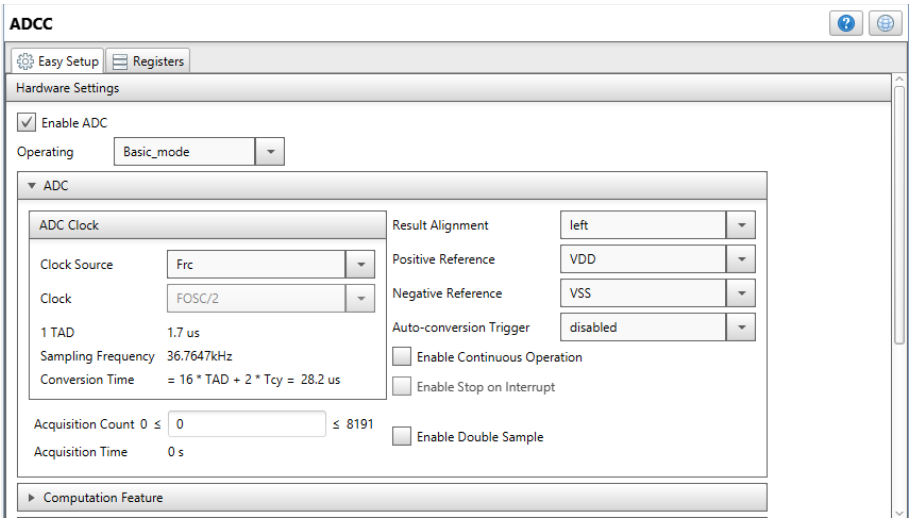


Ilustración 38: Configuración del CAD

Después de configurar el CAD, se pasará a configurar las RC5, RA2, RA1 y RA5 como salidas digitales y RC0 como entrada analógica del CAD y le daremos el nombre de "POT". En la Ilustración 39 se puede observar el *Pin Module* después de realizar la configuración. En este momento hemos terminado de configurar los diferentes módulos utilizados, por lo que le daremos a *Generate* para que MCC genere los archivos correspondientes.

Abriremos el "main.c" para escribir las instrucciones de programa, una de estas instrucciones será ADCC_GetSingleConversion() con la que obtendremos el valor la entrada analógica en ese momento. El valor obtenido con esta función tiene un tamaño de 12 bits por lo que desplazaremos los bits 12 posiciones con el operador ">>" para que nos queden los 4 bits más significativos y se guardarán en la variable de 8 bits adcResult.

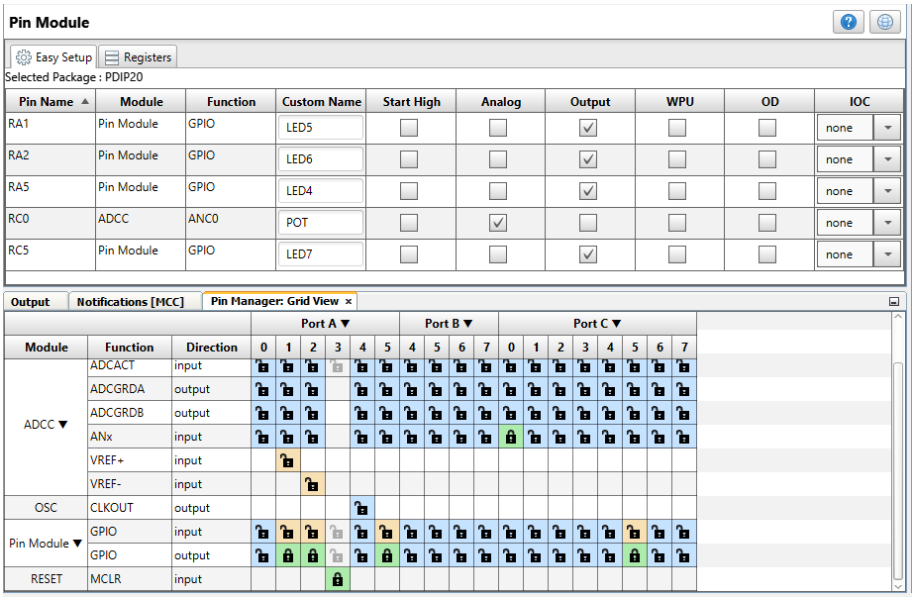


Ilustración 39: Pin manager del ejercicio 4

Mediante la operación *adcResult* & 1 se consigue darle el valor '1' al registro LAT del led4 cuando el bit menos significativo de los cuatro sea también '1', en los siguientes leds el resultado de la operación AND se desplaza varias posiciones para no tener en cuenta los bits anteriores.

```
45  #include "mcc_generated_files/mcc.h"
46
47  static uint8_t adcResult;
48
49  void main(void)
50  {
51      // initialize the device
52      SYSTEM_Initialize();
53
54      //...14 lines
68      LED4_SetLow();
69      LED5_SetLow();
70      LED6_SetLow();
71      LED7_SetLow();
72      while (1)
73      {
74          adcResult = ADCC_GetSingleConversion(POT) >> 12;
75          LED4_LAT = adcResult & 1;
76          LED5_LAT = (adcResult & 2) >> 1;
77          LED6_LAT = (adcResult & 4) >> 2;
78          LED7_LAT = (adcResult & 8) >> 3;
79              // Add your application code
```

Ilustración 40: Código del programa 4

- <u>Comentarios sobre el programa</u>

Gracias a este programa hemos visto como configurar el CAD y comprobar su funcionamiento al poder observar el valor de los cuatro bits más significativos a través de los leds. Con el siguiente programa se aprenderá a utilizar el CAD en una aplicación real.

4.5. Módulo CCP

El módulo CCP puede funcionar de tres modos diferentes, el modo captura, modo comparación y PWM, estos modos se encuentran explicados brevemente en la página 14.

En este apartado se va a explicar su funcionamiento en el modo captura y para medir la distancia con el sensor de ultrasonidos HC-SR04.

Ilustración 41: Sensor de ultrasonidos HC-SR04

En la Ilustración 42 está representado el esquema de funcionamiento del modo captura del periférico PWM, las entradas *trigger* se pueden asociar a un pin en concreto o a las salidas de otros módulos como el comparador analógico o las puertas lógicas. Los cuatro bits del registro MODE sirven para configurar el evento que queremos detectar, este evento puede ser flanco de subida, flanco de bajada, ambos flancos o el 4º o 16º flanco de bajada, cuando este evento ocurre el bit CCPxIF se activa y se almacena en los registros CCPRxH y CCPRxL el valor del Timer 1 en ese instante.

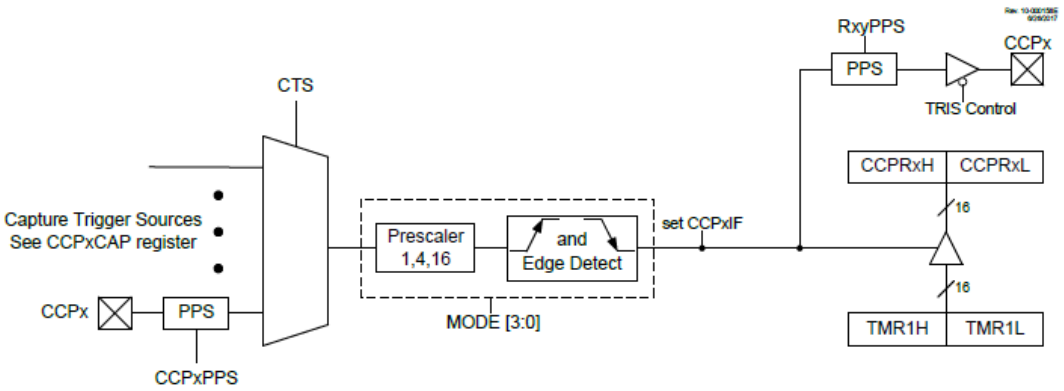


Ilustración 42: Esquema de funcionamiento modo captura [21]

-Implementación del ejemplo

Antes de explicar la programación necesaria se va a exponer el funcionamiento del sensor de ultrasonidos.

El sensor HC-SR04 tiene cuatro pines, los pines VCC y GND son para la alimentación (VCC=5V) y se pueden conectar directamente a la placa ya que el sensor tiene un consumo de apenas 15mA. Al pin *Trigger* hay que conectar una salida digital y al Echo una entrada digital.

Para obtener la distancia es necesario enviar un pulso de 10us al pin *Trigger*, entonces el sensor enviará una onda de ultrasonidos y pondrá a nivel alto la señal del pin Echo, cuando la onda de ultrasonidos rebote en un objeto volverá al sensor y este al detectar la onda pondrá la señal del pin *Echo* a nivel bajo. En el programa hay que obtener el tiempo que ha estado la señal del *Echo* a nivel alto, para convertir este dato en distancia hay que dividirlo entre dos porque la onda va hasta el objeto y vuelve y multiplicarlo por 340 que es la velocidad de propagación del sonido a través del aire [22].

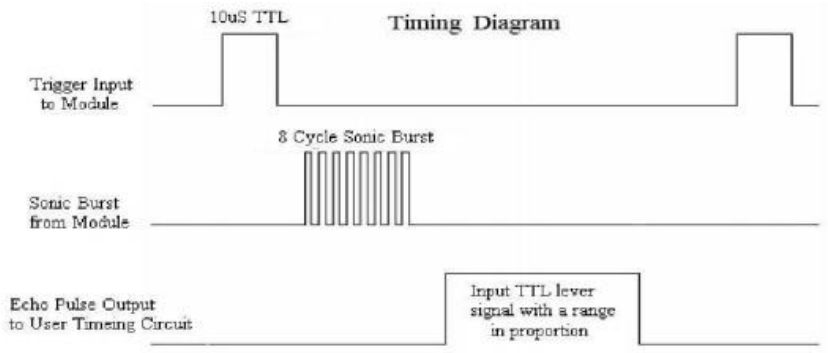


Ilustración 43: Funcionamiento del sensor HC-SR04 [23]

Una vez comprendido del funcionamiento del sensor explicaremos un programa que encienda o apague todos los leds dependiendo de la distancia detectada.

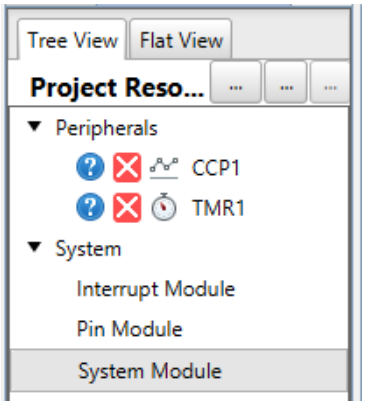


Ilustración 44: Recursos del proyecto

Lo primero será crear un nuevo proyecto, abrir el MCC para configurar el sistema, en este ejemplo utilizaremos una frecuencia de reloj de 32MHz y un divisor de reloj de 4. Después de configurar la frecuencia de reloj añadiremos el módulo CCP1 y el *Timer1* (Ilustración 44).

En la configuración del módulo CCP seleccionaremos el modo "*Capture*", el Timer1 y que detecte todos los flancos, tanto de subida como de bajada, ya que como hemos visto antes queremos obtener el tiempo que está a nivel alto la señal del Echo (**¡Error! No se encuentra el origen de la referencia.**). Además, habilitaremos la interrupción para que cuando se produzca un flanco el programa realice las instrucciones necesarias.

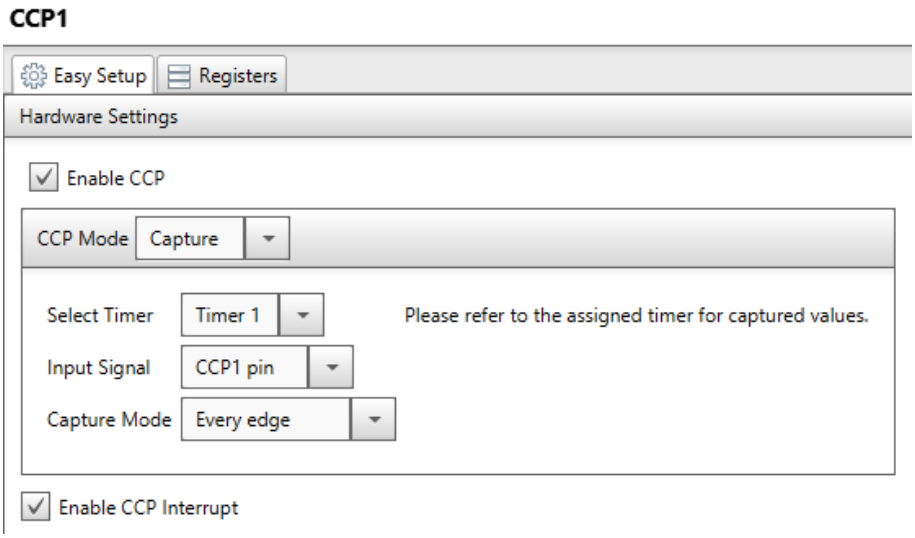


Ilustración 45: Configuración módulo CCP

Al acabar de configurar el módulo CCP pasaremos al *Timer1*. A la hora de ajustar los valores del *Timer1* hay que tener en cuenta el rango de tiempo que pueden durar los pulsos del pin *Echo*, el sensor tiene un alcance de 2cm hasta 4m, por lo que habrá que calcular cuánto tiempo tarda la onda de ultrasonidos en ir y volver en el caso de que la distancia sea igual a 4m. Para calcular este tiempo hay que tener en cuenta que la onda realiza un recorrido del doble de la distancia y que se desplaza con una velocidad de 340m/s.

$$\frac{4m * 2}{340m/s} = 0.0235s = 23.53ms$$

El tiempo máximo durante el cual está la señal a nivel alto es 23.53ms, por lo que al Timer1 lo configuraremos para que tenga un periodo de 32.786ms para tener un margen de error.

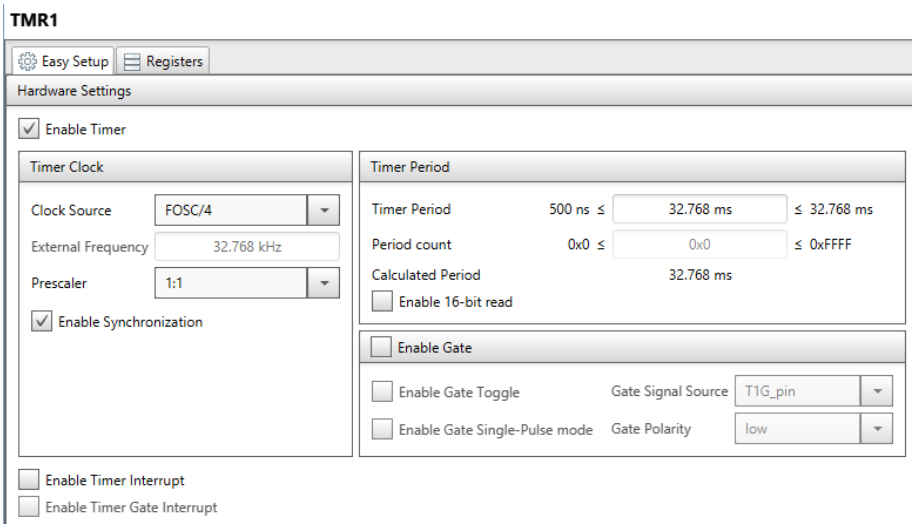


Ilustración 46: Configuración del TMR1

Tras configurar el *Timer1* asignaremos el módulo CCP1 al pin RB5 y al pin RB6 lo llamaremos *Trigger* y lo estableceremos como salida digital, además habilitaremos en

los leds de la placa (Ilustración 47) y haremos clic en el botón “Generate” para crear los archivos de configuración.

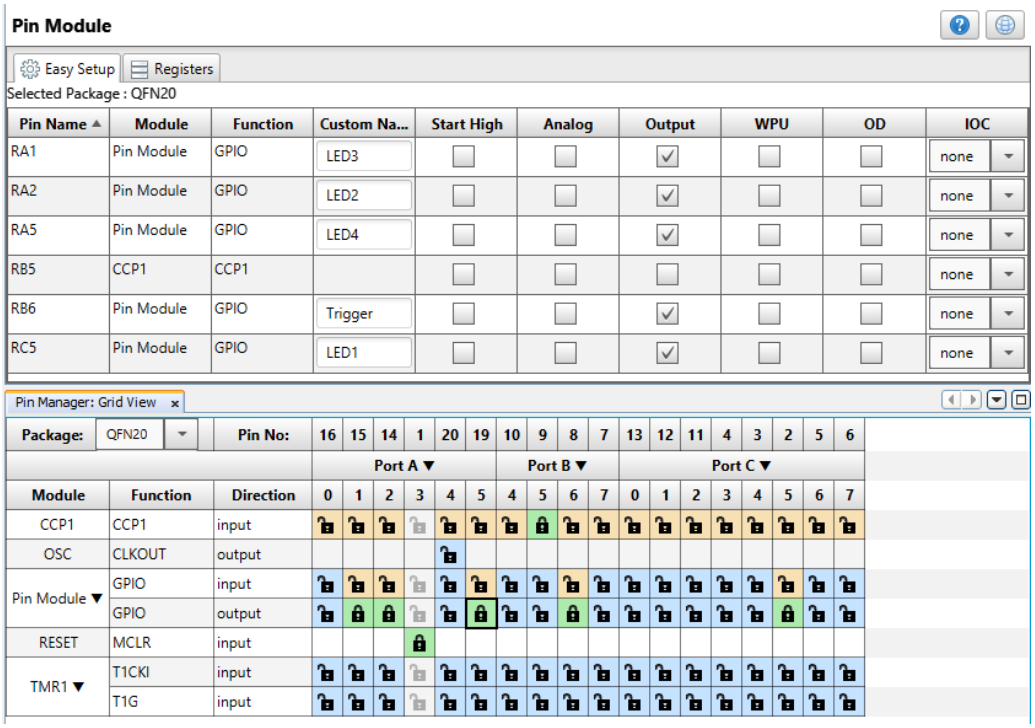


Ilustración 47: Pin Manager Ejercicio 5

Para entender el funcionamiento del programa primero debemos comprender el funcionamiento de la interrupción del módulo CCP.

Cuando sucede una interrupción el programa ejecuta la rutina de atención a la interrupción que es el archivo “interrupt_manager.c” (Ilustración 48), en este archivo comprueba que el bit PEIE ha sido activado, en caso afirmativo comprueba si esta interrupción ha sido provocada por el módulo CCP mirando primero si las interrupciones de este módulo se encuentran activadas y si el bit CCP1IF también esta activado.

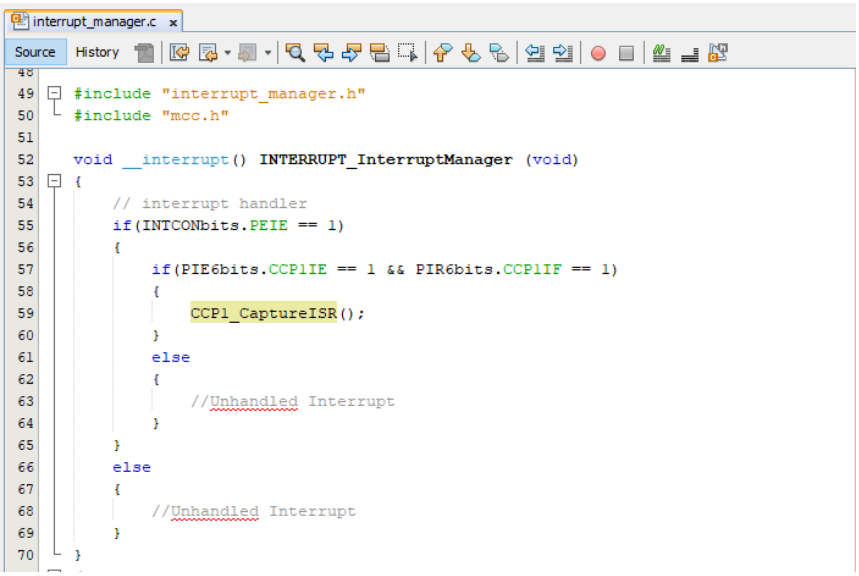


Ilustración 48: Atención a la interrupción del CCP

Si esta condición también se cumple, el programa ejecuta la función "CCP1_CaptureISR()", para trasladarnos a la función haremos Ctrl + Clic sobre ella.

Esta función inicializa de nuevo el bit CCP1IF para poder detectar una nueva interrupción y guarda en la variable module los valores capturados del Timer1 por el módulo CCP, después ejecuta la función "CCP1_CallBack" enviándole la variable module.

La función CCP1_CallBack es un puntero (dirección de memoria) en el que se almacena el valor de module para después utilizarlo en el programa.

```
86    void CCP1_CaptureISR(void)
87    {
88        CCP1_PERIOD_REG_T module;
89
90        // Clear the CCP1 interrupt flag
91        PIR6bits.CCP1IF = 0;
92
93        // Copy captured value.
94        module.ccpr1l = CCPR1L;
95        module.ccpr1h = CCPR1H;
96
97        // Return 16bit captured value
98        CCP1_CallBack(module.ccpr1_16Bit);
99    }
```

Ilustración 49: Función CCP1_CaptureISR()

Una vez entendido el funcionamiento de la interrupción del módulo CCP se explicará el programa principal.

Como en este programa se van a utilizar interrupciones es necesario añadir el archivo *"interrupt_manager.h"* utilizando la función *#include*. Luego declaremos todas las variables que vamos a necesitar a lo largo del programa, las variables uint16_t son datos de 16 bit y las variables declaradas como *volatile* son variables que pueden ser modificadas tanto desde las interrupciones como desde el programa principal.

Para simplificar el código definiremos una función llamada *Trigger* que se encargue de enviar pulsos de 10us al pin Trig del sensor de ultrasonidos.

```
44  #include "mcc_generated_files/mcc.h"
45  #include "mcc_generated_files/interrupt_manager.h"
46  /*
47                          Main application
48   */
49  int subida=0;
50  float distancia;
51  volatile uint16_t capturedValue;
52  uint16_t comienzo, resultado;
53
54  void Trigger(void){
55      Trigger_SetHigh();
56      __delay_us(10);
57      Trigger_SetLow();
58  }
```

Ilustración 50: Código programa 5 parte 1

Para calcular la distancia definiremos una función llamada ultrasonidos, esta función recibirá en la variable *capturedValue* el valor del *Timer1* en el momento que suceda la interrupción. Como siempre se va a producir el flanco de subida antes que el de bajada utilizaremos una variable (subida) para diferenciar uno de otro. Cuando se produzca el flanco de subida el sensor habrá mandado la onda de ultrasonidos y guardaremos el valor del *Timer1* en la variable comienzo y se pondrá subida=1.

Cuando se produzca el siguiente flanco en el módulo CCP será el flanco de bajada, para calcular el tiempo que ha estado el pin *Echo* a nivel alto se restará el valor guardado anteriormente del *Timer1* al último valor que se ha obtenido y el resultado se guardará en la variable "resultado". Este valor representa la diferencia de pulsos que han ocurrido durante ese periodo de tiempo, para pasar este dato a cm se realizará una conversión siguiendo la siguiente ecuación:

$$distancia = \frac{4}{8 * 10^6} * \frac{340}{2} * 100 * resultado$$

El *Timer1* utiliza una frecuencia de oscilación de Fosc/4, y en este programa el oscilador tiene una frecuencia de 8MHz, por lo que cada pulso del Timer1 son 4/(8*10^6)s. La velocidad de propagación de la onda es de 340m/s y recorre dos veces la distancia (ida y vuelta) y para obtener el resultado en cm multiplicaremos por 100. Por lo que la ecuación resultante es:

$$distancia = 0.0085 * resultado$$

```
59  void ultrasonidos (uint16_t capturedValue){
60      if(subida==0){
61          comienzo=capturedValue;
62          subida=1;
63      }
64      else{
65          resultado=capturedValue-comienzo;
66          distancia=resultado*0.0085;
67          subida=0;
68          if (distancia>25){
69              LED1_SetHigh();
70              LED2_SetHigh();
71              LED3_SetHigh();
72              LED4_SetHigh();
73          }
74          else{
75              LED1_SetLow();
76              LED2_SetLow();
77              LED3_SetLow();
78              LED4_SetLow();
79          }
80      }
81  }
```

Ilustración 51: Código programa 5 parte 2

Cuando la distancia sea mayor de 25cm se encenderán todos los leds y cuando sea mayor se apagará.

En el programa principal se utiliza la función CCP1_SetCallBack que se utiliza para que cuando ocurra una interrupción en el módulo CCP mande a la función ultrasonidos el valor obtenido del *Timer1*. Además, hay que habilitar las funciones globales y periféricas. El microcontrolador enviará pulsos por el pin *Trigger* cada 500ms mediante la función *Trigger*.

```
82      void main(void)
83      {
84          // initialize the device
85          SYSTEM_Initialize();
86          CCP1_SetCallBack(ultrasonidos);
87          // Enable the Global Interrupts
88          INTERRUPT_GlobalInterruptEnable();
89          // Enable the Peripheral Interrupts
90          INTERRUPT_PeripheralInterruptEnable();
91          while (1)
92          {
93              __delay_ms(500);
94              Trigger();
95          }
96      }
```

Ilustración 52: Código programa 5 parte 3

-Comentarios del programa

En este programa se ha utilizado el módulo CCP en modo captura en una aplicación real en la que se utiliza un sensor de distancia. Este sensor produce un flanco al enviar una onda de ultrasonidos y otra al recibirlo, por lo que el programa se utiliza para calcular el tiempo que ha transcurrido entre dos flancos y sabiendo el tiempo calcular la distancia.

4.6. Módulo PWM

Mediante este programa vamos a regular la intensidad de un led con el potenciómetro conectado al puerto RC0, para ello se utilizará el CAD para saber el valor del potenciómetro y un módulo PWM.

El funcionamiento del conversor analógico digital se explicó en el apartado 0.

- PWM (módulo generador de pulsos)

En este microcontrolador se incluyen seis módulos PWM, de los cuales cuatro forman parte de los módulos CCP que incluyen un módulo de captura y otro de comparación.

El modo modulador de pulsos en anchura (PWM) sirve para generar señales cuadradas de pulsos ON/OFF en las que se puede variar el periodo que se fija con el *Timer* 2, cuya funcionamiento se explicará más adelante, y el ciclo de trabajo que puede tomar valores entre 0 y 100% se ajusta con el registro PWMxDC.

Este modo es de gran utilidad a la hora de controlar la potencia de un motor o regular la luminosidad de una bombilla.

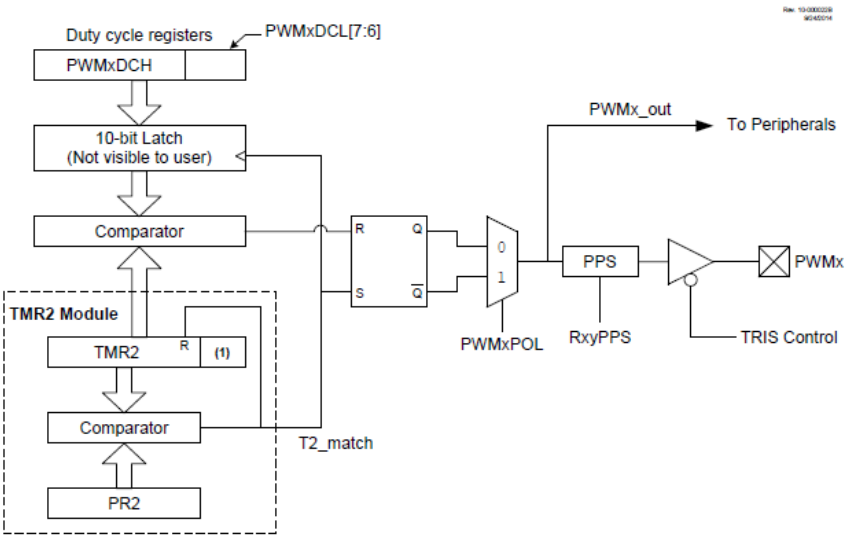


Ilustración 53: Esquema funcionamiento del módulo PWM [21]

- Timer 8 bits

Los Timers de 8 bits funcionan de forma similar a los temporizadores de 16 bits que se han explicado en el apartado 0.

En este caso se utiliza el registro de 8 bits TxTMR como contador y se dispone de un predivisor de tres bits (CKPS) y un postdivisor de cuatro bits (OUTPS), ambos se encuentran dentro del registro TxCON. La señal de reloj de entrada TMRx_clk es configurable.

El esquema de funcionamiento se representa en la Ilustración 54 y el proceso es el siguiente: la señal de reloj incrementa el valor del predivisor hasta que es igual al configurado, en ese momento se reinicia y se aumenta en uno el contador TxTMR. Este contador se va incrementado hasta que su valor es igual al valor TxPR, entonces el contador se resetea y aumenta el postdivisor, en el momento que el postdivisor alcanza el valor configurado se reinicia y activa la señal TMRxIF.

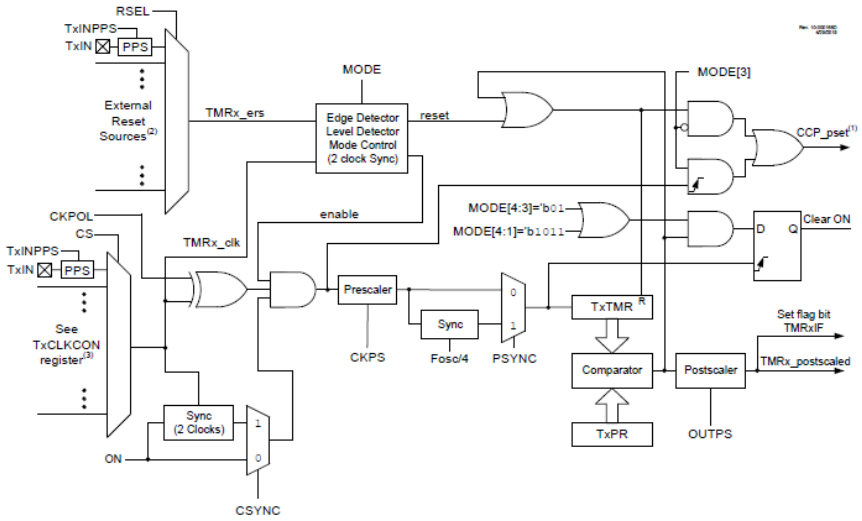


Ilustración 54: Esquema de funcionamiento Timer 8 bits [21]

El tiempo total de este proceso viene dado por la siguiente ecuación.

$$T = T_{TMRx\_clk} * Pred * (TxPR + 1) * Post$$

Los temporizadores de 8 bits tienen tres modos de funcionamiento diferentes:

- Modo libre (*Roll over Pulse*): en este modo el temporizador activa TMRxIF periódicamente cada intervalo definido en el timer.
- Modo disparo único: en este modo el postdivisor no interviene en el tiempo de configuración del timer porque cuando TxTMR es igual a TxPR se activa la señal TMRxIF. Para volver a utilizar el temporizador es necesario volver a encenderlo.
- Modo monoestable: este modo es igual al de disparo único con la diferencia de que al activar TMRxIF el temporizador no se desactiva y se vuelve a poner en funcionamiento cuando la señal enable se genera.

- Implementación del ejemplo

En este ejercicio vamos a utilizar el potenciómetro conectado a RC0 como entrada analógica a través de la cual vamos a regular la intensidad del led conectado a RA1. Para ello crearemos un nuevo proyecto y abriremos el MCC y configuraremos la frecuencia del reloj interno a 8MHz y añadiremos los periféricos TMR2, PWM6 y ADCC.

En el conversor analógico digital se utilizaremos el oscilador interno como señal y los valores VSS y VDD como referencias, el resultado de la conversión se alineará a la derecha (Ilustración 55).

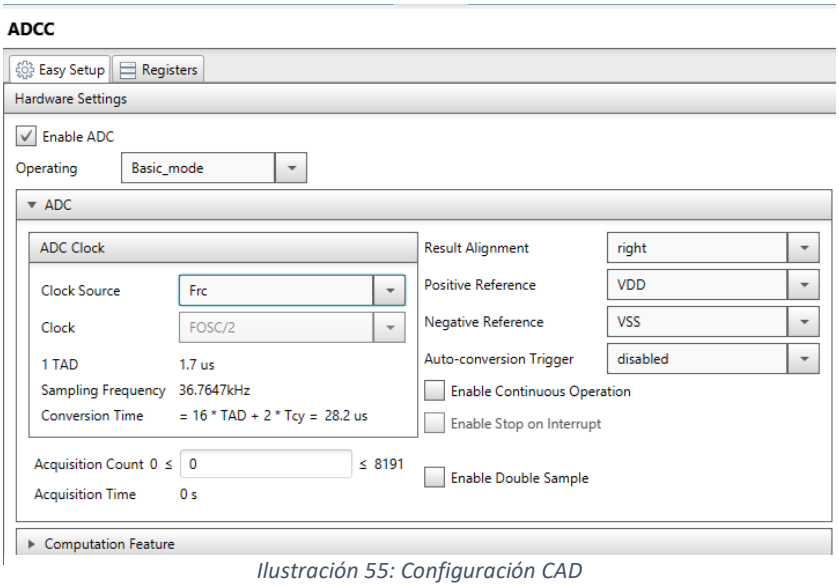


Ilustración 55: Configuración CAD

Para configurar el módulo PWM, primero ajustaremos el TMR2 a un periodo de 1.024ms con un predivisor de 8 y *Roll over Pulse* como modo de funcionamiento, en la configuración del PWM6 seleccionaremos el TMR2 y un ciclo de trabajo del 50%. En el *Pin Module* asignaremos la salida del PWM al pin RA1. (Ilustración 56)

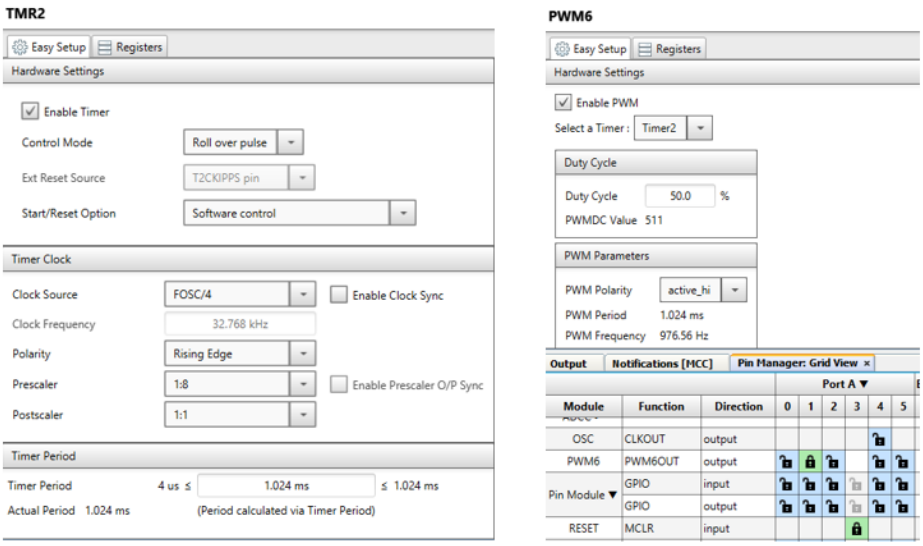


Ilustración 56: Configuración PWM

Tras acabar de configurar estos módulos, pulsaremos el botón Generate y abriremos el archivo “main.c”, en este programa utilizaremos la función ADCC_GetSingleConversion() para obtener el valor actual de la entrada analógica y este valor lo desplazaremos dos posiciones hacia la derecha porque el conversor tiene una resolución de 12 bits y la PWM de 10, de esta forma con la función PWM6_LoadDutyValue() se cargará el valor obtenido en el conversor teniendo en cuenta los dos bits más significativos.

```
44  #include "mcc_generated_files/mcc.h"
45  static uint16_t adcResult;
46  void main(void)
47  {
48      // initialize the device
49      SYSTEM_Initialize();
50      while (1)
51      {
52        adcResult=ADCC_GetSingleConversion(channel_ANC0) >> 2;
53        PWM6_LoadDutyValue(adcResult);
54      }
55  }
```

Ilustración 57: Código del programa 5

- Comentarios sobre el programa

En este ejercicio se ha aprendido a configurar una señal PWM y los Timers de 8 bits, aunque en este caso la salida de la PWM solo se haya asignado un pin se puede asignar a varios a la vez y a todos ellos les llegará la misma señal. Además, el valor del PWM también se puede cambiar mediante software.

4.7. MTouch Bottom

El siguiente ejercicio se divide en dos partes, en la primera se va a explicar la implementación del sensor *Mtouch Button* como un pulsador y en la segunda como sensor de proximidad debido a que su implementación es muy parecida. Este panel capacitivo se encuentra conectado al terminal RC1.

- Implementación del ejemplo

En este ejemplo se va a utilizar el sensor *mTouch Button* como interruptor para encender el led conectado a RC5.Como esta panel no pertenece al microcontrolador PIC16F18446 por lo que para añadir su librería tendremos que ir a la ventana *Device Resources* → *Libraries* → *Foundation Services*.

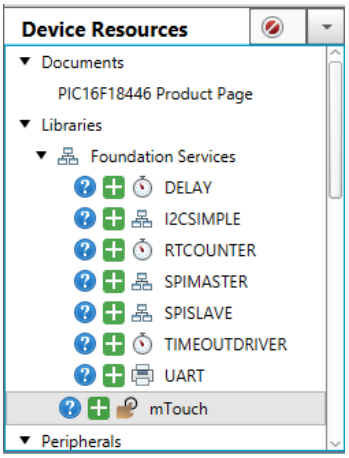


Ilustración 58: Ventana Device Resources

El panel *mTouch* necesita una frecuencia de reloj mínima de 8MHz, por lo que en este programa utilizaremos una frecuencia de 32MHz. Además, hace falta añadir el CAD, este módulo se suele añadir automáticamente al abrir la librería de *mTouch*, pero si acaso se comprobará que se ha añadido correctamente.

A continuación, se abrirá el *Pin Manager* y le asignaremos a RC1 la salida CS de *mTouch* y configuraremos RC5 como salida digital (Ilustración 59).

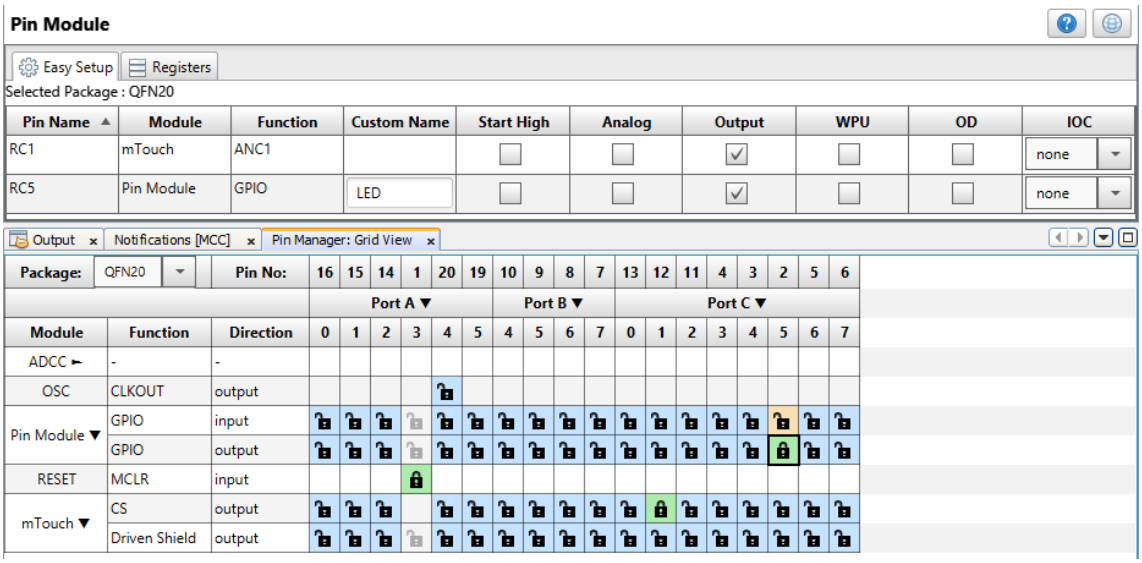


Ilustración 59: Pin Manager del ejercicio 7

- *mTouch* como botón

Para utilizar el panel capacitivo como botón hay que dirigirse al módulo *mTouch* para crear un nuevo botón como se observa en la Ilustración 60.

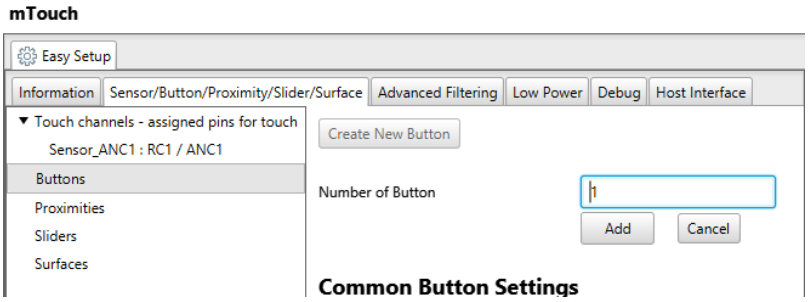


Ilustración 60: Creación botón para mTouch

Para que el microcontrolador sea capaz de registrar los cambios en el sensor es necesario que el CAD tome datos en un intervalo mayor a 1us, configurando el reloj interno (Frc) como señal para el conversor se cumple esta especificación.

Tras pulsar el botón Generate y pasaremos a escribir el programa en el archivo "main.c", en este programa es necesario habilitar las interrupciones globales y las de los periféricos, además usaremos la variable booleana MTOUCH_Service_Mainloop() para comprobar que la interrupción ha sido causada por el panel capacitivo y MTOUCH_Button_isPressed(0) para comprobar si se ha pulsado mTouch. El código del programa se encuentra reflejado en la Ilustración 61.

```
#include "mcc_generated_files/mcc.h"

/*...3 lines */

void main(void)
{
    SYSTEM_Initialize();
    INTERRUPT_GlobalInterruptEnable();
    INTERRUPT_PeripheralInterruptEnable();
    while (1)
    {
        if(MTOUCH_Service_Mainloop()){
            if (MTOUCH_Button_isPressed(0))
                LED_Toggle();
        }
    }
}
```

Ilustración 61: Código del programa 7.1

- mTouch como sensor de proximidad

En este caso configuraremos también el terminal RA5 como salida digital y en vez de crear un nuevo botón se creará un sensor de proximidad de la forma que se representa en la Ilustración 62 y le daremos a *Generate*.

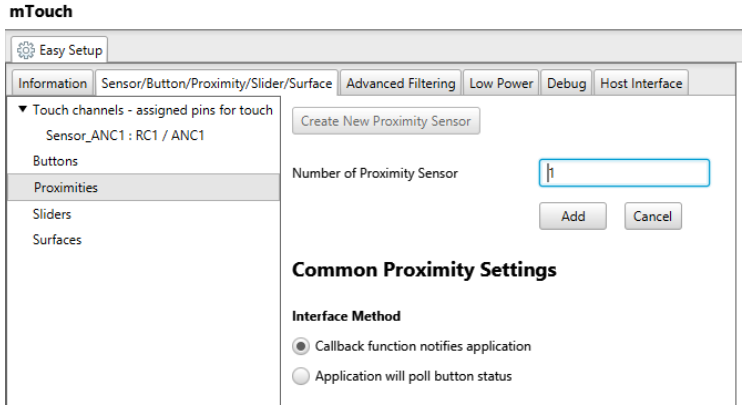


Ilustración 62: Creación del sensor de proximidad

En este programa utilizaremos dos función para atender a la interrupción, la función MTOUCH_Proximity_SetActivatedCallback() se activa al detectar un objeto y la función MTOUCH_Proximity_SetNotActivatedCallback() se activa al dejar de detectar el objeto o cuando se mantiene durante un determinado tiempo, para poder apreciar esta diferencia cada una de las funciones va a actuar sobre un led.

```
#include "mcc_generated_files/mcc.h"
void processProximityActivate(enum mtouch_proximity_names prox)
{       if(prox == Proximity0)
        LED2_Toggle();
}
void processProximityNotActivate(enum mtouch_proximity_names prox)
{       if(prox == Proximity0)
        LED_Toggle();
}
void main(void)
{
    SYSTEM_Initialize();
    INTERRUPT_GlobalInterruptEnable();
    INTERRUPT_PeripheralInterruptEnable();
    MTOUCH_Proximity_SetNotActivatedCallback(processProximityNotActivate);
    MTOUCH_Proximity_SetActivatedCallback(processProximityActivate);
    while (1)
    {
        MTOUCH_Service_Mainloop();
    }
}
```

Ilustración 63: Código del programa 7.2

- Comentarios sobre el programa

Con este programa se ha aprendido a configurar un panel táctil como pulsador, que puede ser utilizado para la activación de salidas y como un sensor de proximidad, estos sensores son utilizados para detectar de objetos sin que se produzca un contacto. El uso de estos paneles táctiles supone una gran ventaja respecto a los interruptores y pulsadores tradicionales debido a que no tiene componentes mecánicos y es más difícil que puedan sufrir una avería.

4.8. Comparador Analógico (CMP) y CDA

En este ejemplo se va a utilizar un comparador analógico para determinar si la señal del potenciómetro es mayor o menor que una señal de referencia creada por el CDA, en caso de que la señal que viene del potenciómetro sea mayor se encenderá el led D4 conectado al pin RA5 y en caso contrario el led encendido será el D7 que se encuentra conectado al terminal RC5.

- Conversor Digital Analógico

Como el microcontrolador trabaja internamente con señales digitales, este conversor se utiliza cuando se quiere tener una señal de salida analógica. En el PIC16F18446 solamente se incluye un convertidor digital analógico de 5 bits, lo que significa que la señal analógica obtenida puede tomar 32 valores diferentes. Esta señal se pasa por un filtro para suavizar los cambios de valor. En la Ilustración 64 se observa el esquema de funcionamiento del CDA.

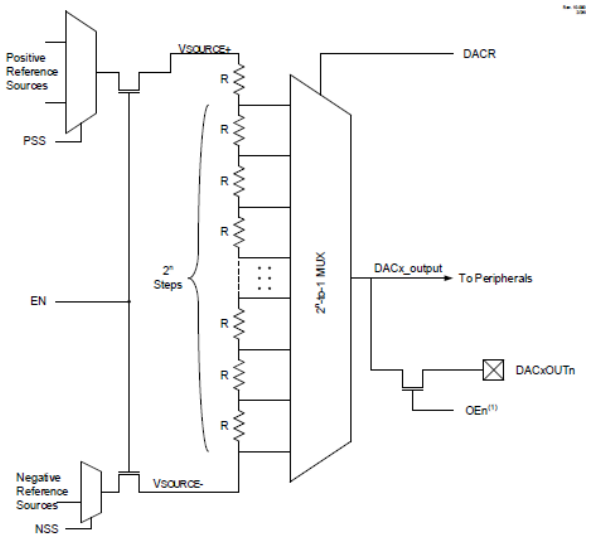


Ilustración 64: Esquema de funcionamiento del CDA [21]

$$DACx\_output = \left(\left(V_{SOURCE+} - V_{SOURCE-}\right) x \frac{DACR[4:0]}{32}\right) + V_{SOURCE-}$$

Para utilizar este conversor es necesario definir dos valores de referencia, uno a nivel alto y otro a nivel bajo. Estos valores se definen en los bits PSS y NSS que pertenecen al registro DAC1CON0.

Los valores de la señal analógica se envían a los periféricos a través de DAC1CON1.

- Comparador analógico

Los comparadores analógicos son usados para obtener una señal digital a partir de la comparación de dos señales analógicas. En este PIC se incluyen dos comparadores analógicos. En la Ilustración 65 está representado el esquema de funcionamiento de este comparador.

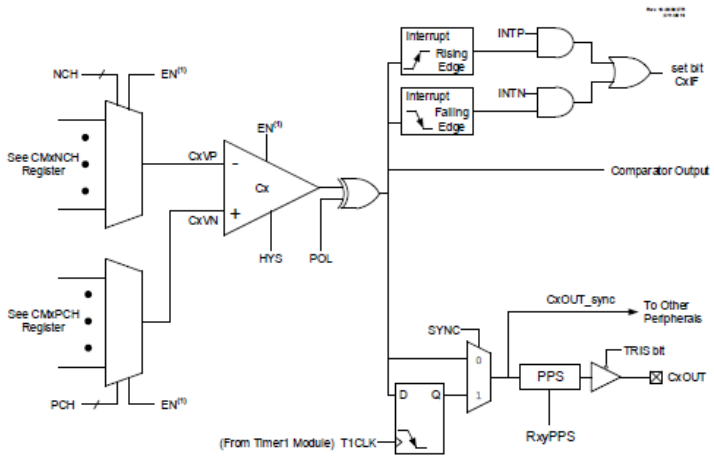


Ilustración 65: Esquema funcionamiento comparador analógico [21]

Algunos de los registros más importantes son CMxNCH y CMxPCH que permiten elegir el origen de las señales a comparar, CMxCON0 que habilita la histéresis e invierte las entradas y salidas según se desee.

- Implementación del ejemplo

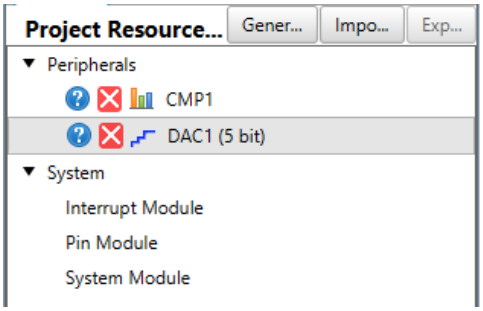


Ilustración 66: Recursos del proyecto

Para realizar este ejemplo lo primero será crear el proyecto, abrir el MCC y añadir el conversor digital analógico y el comparador analógico () y seleccionaremos 8MHz como frecuencia de reloj.

Primero configuraremos el CDA eligiendo como referencia positiva VDD (5V) y VSS (0V) como negativa. En "Required ref" debemos insertar el valor de salida del convertidor digital analógico deseado, en este caso pondremos 2V para que sea un valor intermedio.

Ahora pasaremos a configurar el comparador analógico, como entrada no inversora seleccionaremos la salida del CDA y como entrada inversora el terminal RC2, se ha seleccionado esta entrada porque el pin al que está conectado el potenciómetro (RC0) no se puede seleccionar como entrada al comparador, por lo que tendremos que cortocircuitar los terminales RC0 y RC2 con un cable. La salida del comparador se asignará a los terminales de los leds y se seleccionará invertir salida para que cuando la señal del potenciómetro sea mayor que la referencia se iluminen los leds. Además, se habilitará la opción de histéresis para evitar los rebotes, el valor típico de la histéresis es de 25mV.

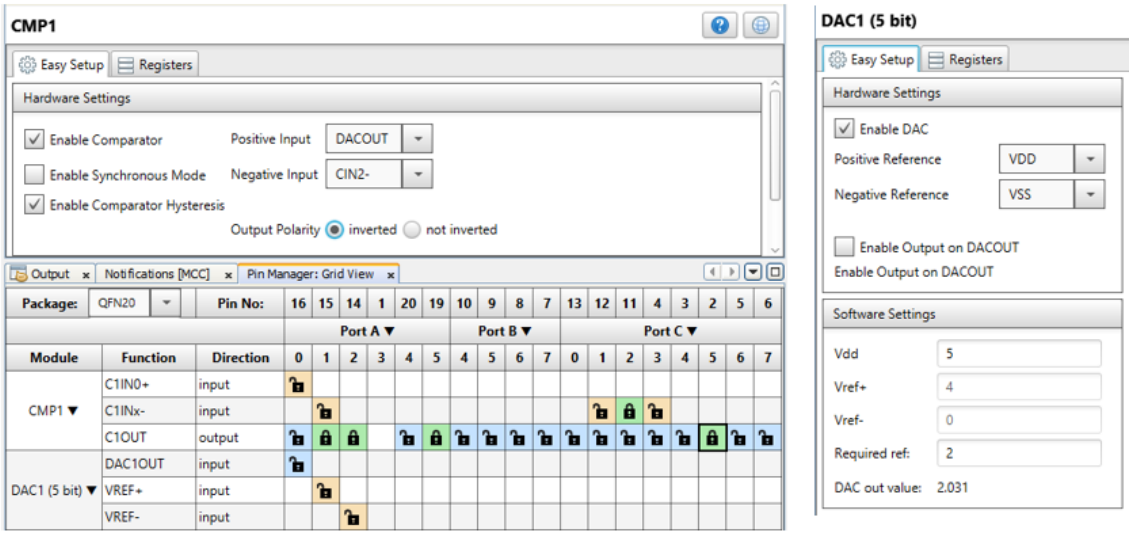


Ilustración 67: Configuración de los módulos

- Comentarios sobre el programa

A través de la implementación de este ejemplo se comprende el funcionamiento del conversor digital analógico y los comparadores analógicos, en este caso se han utilizado para activar y desactivar una salida dependiendo de la señal del potenciómetro. Este tipo de programas se pueden utilizar en aplicaciones en las que se quiera activan una salida cuando una variable alcanza un determinado valor.

4.9. Puertas lógicas y FVR

En este ejercicio se va a diseñar un programa que evite activar una salida si la señal del potenciómetro es mayor a la referencia y que desactive la salida si ya estaba activada, este ejercicio está basado en el anterior, pero utilizando como señal de referencia la producida por el módulo FVR y utilizando puertas lógicas para evitar escribir instrucciones en el programa.

- Puertas lógicas (CLC)

El microcontrolador PIC16F18446 incorpora cuatro módulos diferentes de puertas lógicas configurables, cada módulo puede implementar ocho configuraciones diferentes

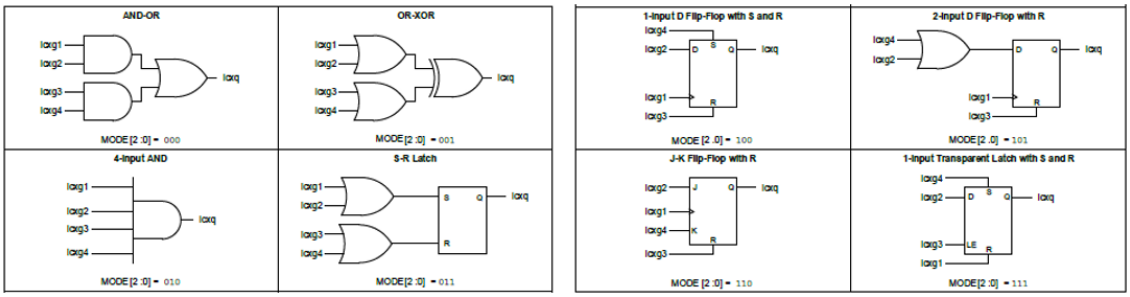


Ilustración 68: Puertas lógicas incorporadas en el microcontrolador [21]

que incluyen AND-OR, OR-XOR, AND y los biestables S-R, D y J-K. Todas estas configuraciones están representadas en la Ilustración 68.

Estas puertas lógicas operan de independientemente del microcontrolador, evitando las limitaciones de velocidad de la ejecución del software. Cada puerta puede configurarse como entrada de 64 señales y eventos diferentes, como interrupciones de los Timers, salidas de periféricos, etc.

Para configurar estos periféricos se necesita una gran cantidad de registros, pero gracias a la aplicación MCC la configuración las puertas lógicas se realiza de forma gráfica, lo que simplifica el trabajo de manera notable.

- Módulo FVR

El módulo FVR (Fixed Voltage Reference) estable una señal de referencia independiente del valor de V_{DD}, esta señal puede tomar los siguientes valores: 1.024V, 2.048V o 4.096V y se puede utilizar en el CAD como entrada y como referencia positiva, como entrada en el comparador analógico y en el CDA.

- Implementación del ejemplo

Para poder implementar este ejemplo se va a utilizar una configuración similar a la del ejemplo 0, aunque en este caso las salidas que active el comparador serán RA5 y RA1 que se encuentran conectadas a los leds D4 y D5 y la señal de referencia será producida por el módulo FVR. Si la salida del comparador analógico se mantiene invertida, el sistema permitirá encender la salida cuando la señal del potenciómetro sea mayor a la señal de referencia. Sólo cuando los leds D4 y D5 estén encendidos se podrán encender los otros dos leds (D6 y D7) mediante el pulsador S1 conectado a RC4.

Tras configurar el oscilador interno a una frecuencia de 8MHz, al igual que en el ejercicio 1 (Ilustración 21), se añadirán al proyecto el módulo CLC1, CMP1 y FVR (Ilustración 69).

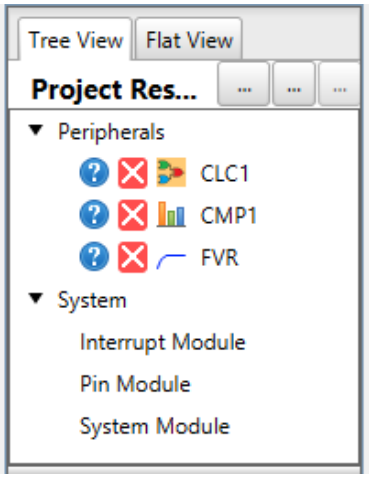


Ilustración 69: Recursos del proyecto 9

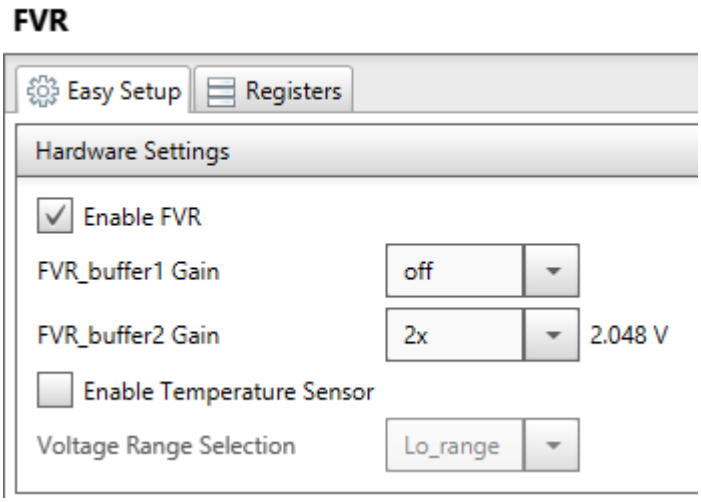


Ilustración 70:Configuración del módulo FVR

Lo primero que configuraremos será el módulo FVR, en el cual activaremos el buffer 2 con una ganancia de x2 para obtener una señal de referencia de 2.048V. Se utiliza el buffer 2 porque es el que está conectado al comparador analógico (Ilustración 70).

Después pasaremos a configurar el comparador analógico CMP1, en este comparador seleccionaremos la señal del módulo FVR como señal positiva y la como negativa la entrada CIN2- que está asociada al pin RC2, por lo que tendremos que conectar el pin RC0, en el que se encuentra conectado el potenciómetro, al pin RC2 mediante un cable. La salida del comparador analógico la asignaremos a los pines RA1 y RA5 que están conectadas a los leds D4 y D5, de esta forma se iluminarán cuando la señal del potenciómetro se encuentre en el rango adecuado para encender los otros dos leds.

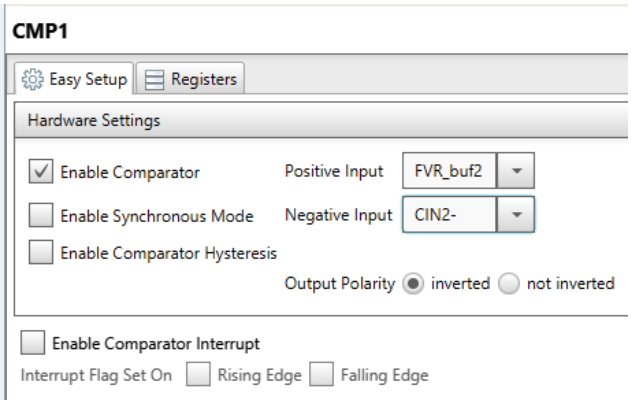


Ilustración 71: Configuración CMP1

Tras configurar el comparador y el FVR se añadirá el módulo CLC1 al proyecto. Para este sistema se va a utilizar la configuración AND-OR y las entradas de esta puerta serán CLCIN0 que la asociaremos al pulsador conectado a RC4, la salida del comparador analógico (CMP1) y la salida de este módulo (CLC1_OUT) para que cuando el pulsador

se deje de pulsar las salidas se mantengan activadas (Ilustración 72). La salida de la puerta lógica se asignará a los pines RA2 y RC5.

La entrada CLCIN0 se encuentra negada porque el pulsador utiliza resistencias de pull-up y la señal cuando no se encuentra pulsado es de nivel alto (5V).

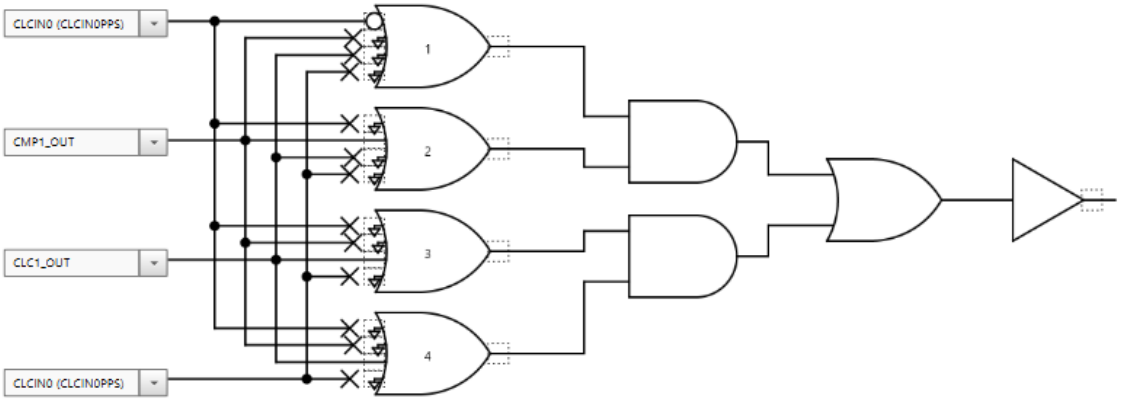


Ilustración 72: Configuración puertas lógicas

Al acabar la configuración de las entradas y salidas obteniendo el resultado de la

```
44  #include "mcc_generated_files/mcc.h"
45
46  /*
47                          Main application
48   */
49  void main(void)
50  {
51      SYSTEM_Initialize();
52      while (1)
53      {
54          SLEEP();
55      }
56  }
57  /**
58   End of File
59   */
```

Ilustración 74: Programa del ejercicio 9

Pin Module

Easy Setup | Registers

Selected Package : QFN20

Pin Name	Module	Function	Custom Name	Start High	Analog	Output	WPU	OD	IOC
RA1	CMP1	C1OUT		☐	☐	☑	☐	☐	none
RA2	CLC1	CLC1OUT		☐	☐	☑	☐	☐	none
RA5	CMP1	C1OUT		☐	☐	☑	☐	☐	none
RC2	CMP1	C1IN2-		☐	☑	☐	☐	☐	none
RC4	CLC1	CLCIN0		☐	☐	☐	☑	☐	none
RC5	CLC1	CLC1OUT		☐	☐	☑	☐	☐	none

Ilustración 73: Pin mánager del Ejercicio 9

Ilustración 73 e Ilustración 75 y darle al botón Generate ya tenemos el programa preparado para volcarlo a la placa, ya que los periféricos se encargan de gestionar las salidas y entradas y no es necesario escribir ninguna instrucción, aunque es recomendable escribir la función SLEEP(); dentro del while del programa principal para que el microcontrolador se encuentre en modo de bajo consumo (Ilustración 74).

			Port A ▼						Port B ▼				Port C ▼							
Module	Function	Direction	0	1	2	3	4	5	4	5	6	7	0	1	2	3	4	5	6	7
CLC1	CLC1OUT	output																		
CLCx ▼	CLCIN0	input																		
	CLCIN1	input																		
	CLCIN2	input																		
	CLCIN3	input																		
CMP1 ▼	C1IN0+	input																		
	C1INx-	input																		
	C1OUT	output																		

Ilustración 75: Pin Module ejercicio 9

- Comentarios sobre el programa

En este ejemplo se ha utilizado un módulo de puertas lógicas configurables para evitar tener que escribir funciones iterativas en el programa principal, así se evita que el microcontrolador tenga que estar pendiente y pueda ejecutar otras instrucciones al mismo tiempo. Este ejemplo podría aplicarse a la puesta en marcha de un sistema en el que haya una magnitud que tenga que estar por debajo o por encima de un valor para su funcionamiento óptimo, como por ejemplo la temperatura de un motor.

Al poder configurarse este módulo de ocho formas diferentes permite un gran variedad de aplicaciones diferentes a las vistas en este ejemplo.

4.10. EUSART

En este ejercicio se va a controlar el encendido y apagado de los leds desde el teclado del ordenador, para ello se utilizará el módulo EUSART que está incluido en el microcontrolador.

- Módulo EUSART

El módulo EUSART (*Enhanced Universal Synchronous Asynchronous Receiver Transmitter)* es un interfaz de comunicación serie que puede ser configurado para trabajar en modo *Full Duplex* asíncrono, utilizado para comunicarse a ordenadores personales, o *Half Duplex* síncrono que se utiliza para comunicarse con otros módulos integrados en el microcontrolador como CAD o CDA o con otros microcontroladores.

A continuación, se va a explicar de forma resumida el funcionamiento del módulo EUSART en modo *Full Duplex*: a través de los registros SPBRGH y SPBRGL se selecciona los baudios de la señal de sincronismo y por los terminales TX y RX se envían y reciben los datos, estos datos se almacenan en los registros TXxREG y RCxREG.

Para habilitar las interrupciones de los terminales TX y RX se deben activar los registros TXIE y RCxIE, y cuando se producen las interrupciones se activan los *flags* TXIF y RCxIF.

En la Ilustración 76 y en la Ilustración 77 están representados los esquemas de los bloques de transmisión y recepción del módulo EUSART.

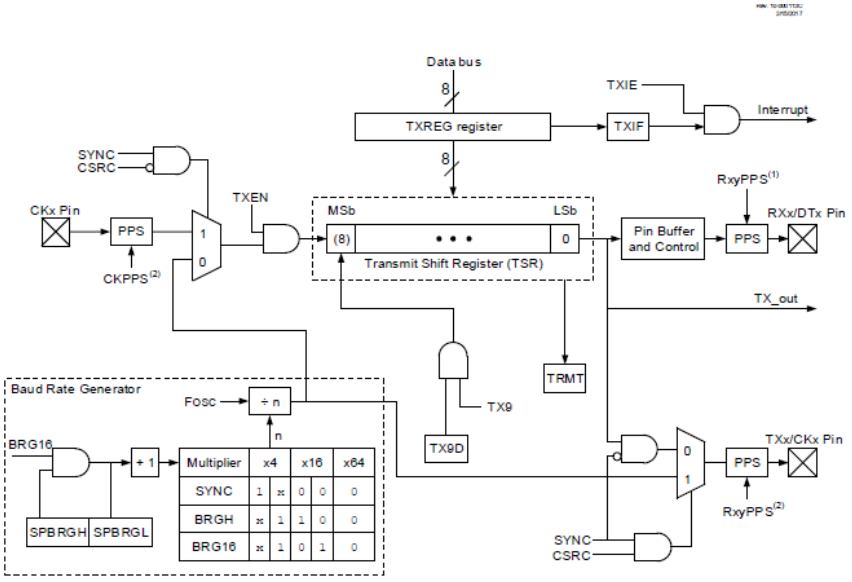


Ilustración 76: Esquema del bloque de transmisión EUSART [21]

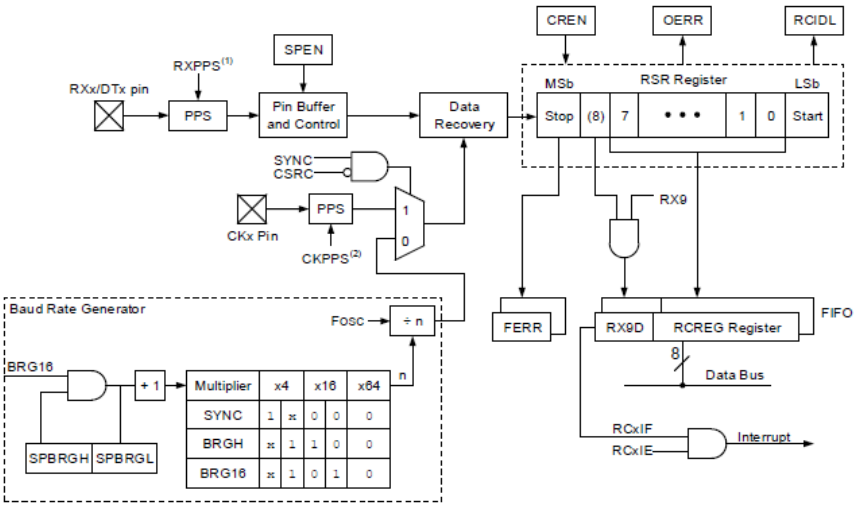


Ilustración 77: Esquema de recepción del módulo EUSART [21]

- Implementación del ejemplo

En este ejemplo vamos a controlar el estado de los leds con los números del teclado del ordenador, a cada led le asignaremos un número según su nombre, es decir, al led D4 le asignaremos el número 4, al led D5 le asignaremos el número 5 y así con el resto de los leds. Además, cada vez que se cambie el estado del led, aparecerá un mensaje que nos informará de si el led está encendido o apagado. Por último, con el número 1 el programa nos mostrará el estado actual de los cuatro leds.

Para realizar este ejercicio lo primero que debemos hacer es crear un nuevo proyecto y abrir MCC para configurar el microcontrolador.

En este ejercicio seleccionaremos una frecuencia de oscilación de 8 MHz, para ello escogeremos una frecuencia de reloj de 32MHz con un divisor de reloj de 4, al igual que en el ejemplo 1 (Ilustración 21).

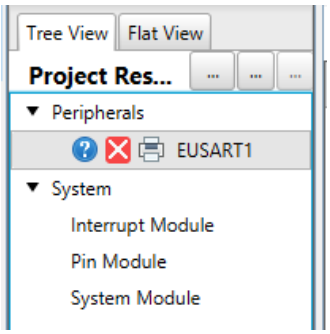


Ilustración 78: Recursos del proyecto

Tras añadir el módulo EUSART al proyecto, procederemos a su configuración. Seleccionaremos el modo asíncrono (*Full Duplex*) para que pueda enviar y recibir datos al mismo tiempo, habilitaremos la transmisión y recepción de datos (*Enable Transmit y Enable Recive*), las interrupciones del módulo EUSART y por último activaremos la opción de utilizar la librería STDIO para usar sus funciones para recibir y enviar datos.

Además, seleccionaremos una velocidad de transmisión de 9600 baudios (Ilustración 79).

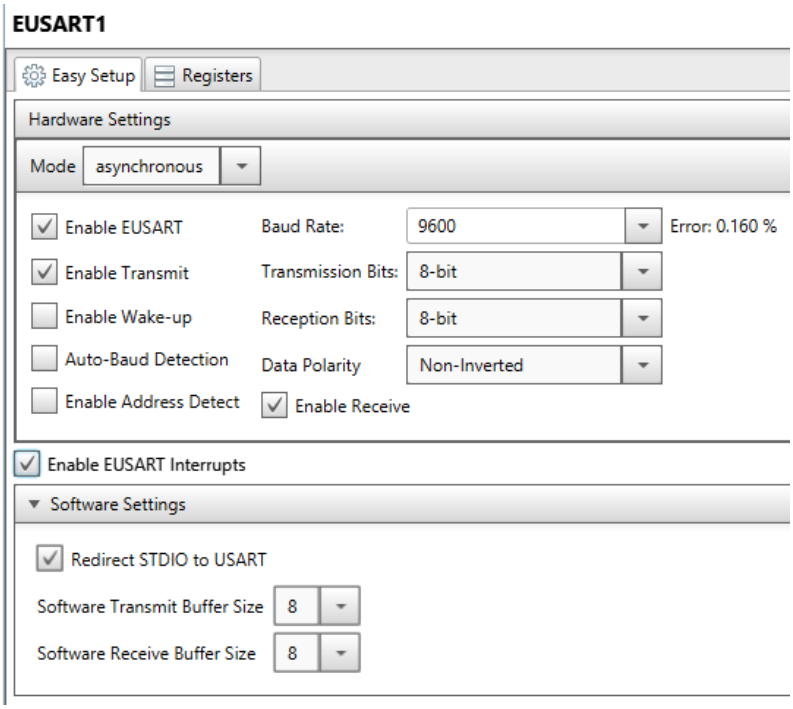


Ilustración 79: Configuración módulo EUSART

Tras acabar la configuración del módulo EUSART, comprobaremos que los pines configurados como RX y TX son los terminales RB5 y RB7 y estableceremos los terminales de los leds (RC5, RA2, RA1, RA5) como salidas digitales. (Ilustración 80)

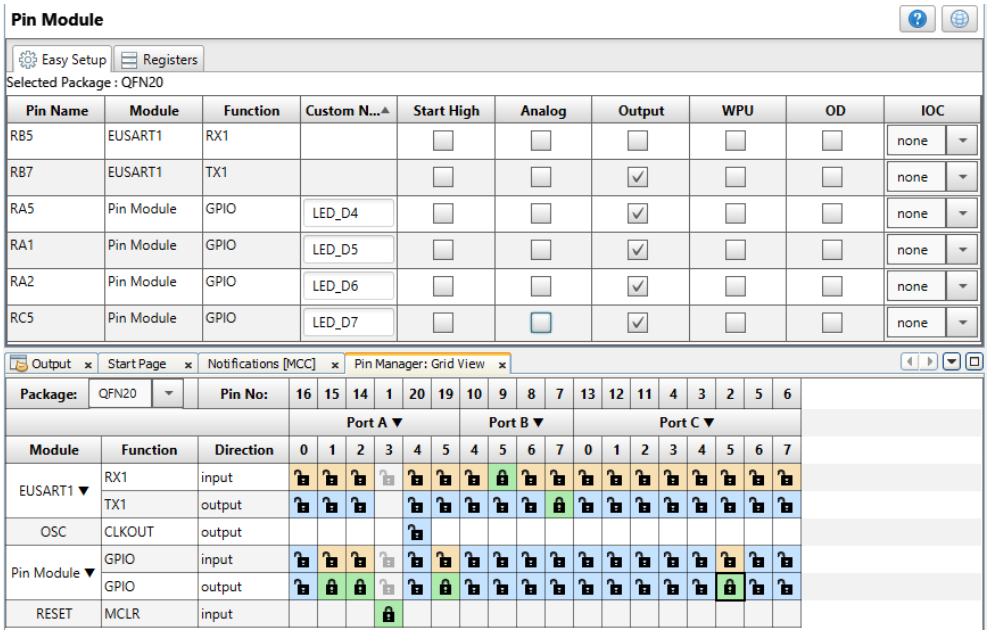


Ilustración 80: Pin Module del ejercicio 10

Después de acabar la configuración de los terminales del microcontrolador, pulsaremos el botón *Generate* para crear los archivos necesarios para la realización del proyecto.

En el archivo main.c habilitaremos las interrupciones globales y periféricas, para escribir las instrucciones crearemos un nuevo archivo de extensión ".c", de esta forma conseguiremos un código más ordenado.

Para crear un nuevo archivo de extensión ".c" haremos clic con el botón derecho sobre *Source Files* → *New* → main.c (Ilustración 81) y le daremos el nombre de "CodigoEusart.c" (Ilustración 82).

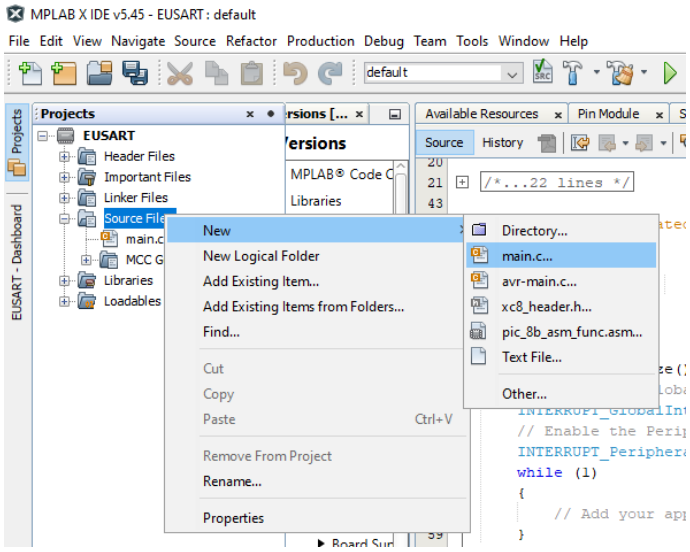


Ilustración 81: Creación de archivo .c (1)

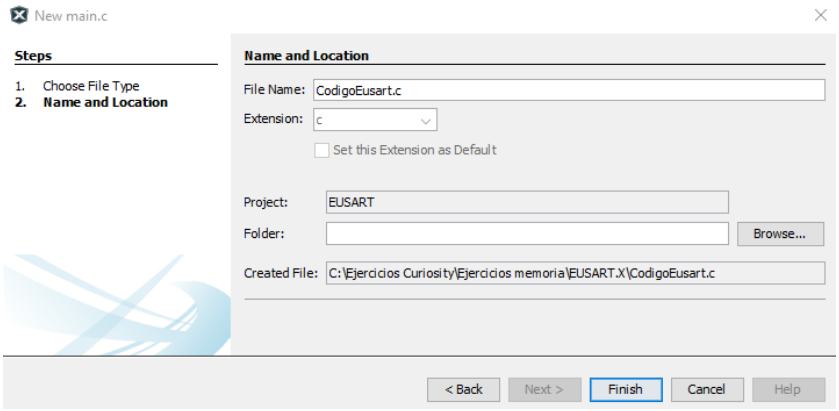


Ilustración 82: Creación de archivo .c (2)

Una vez creado el programa, añadiremos el archivo "mcc.h" con la instrucción *#include* para poder utilizar las funciones generadas por el MCC. En este archivo vamos a definir dos funciones a las que llamará el programa principal, la primera función se llamará "Inicio" e imprimirá por pantalla la frase "PROGRAMA DE CONTROL DE LEDS" a través de la función printf.

```
11  void Inicio(void)
12  {
13      printf("\n PROGRAMA  CONTROL DE LEDS\n");
14  }
```

Ilustración 83: Función Incio

La segunda función será la encargada de recoger los datos recibidos a través del módulo EUSART, está función se llamará "codigo" y en las

```
16   void codigo(void)
17   {
18       if(eusart1RxCount!=0)
19       {
20
21       recibido=EUSART1_Read();  // read a byte for RX
22
23       EUSART1_Write(recibido);  // send a byte to TX  (from
24
25       switch(recibido)     // check command
26       {case 4:
27           {
28               LED_D4_Toggle();
29               if (LED_D4_GetValue()==1){
30                   printf("\n El led D4 esta encendido\n");
31               }
32               else
33                   printf("\n El led D4 esta apagado\n");
34               break;
35           }
36        case 5:
37           {
38               LED_D5_Toggle();
39               if (LED_D5_GetValue()==1){
40                   printf("\n El led D5 esta encendido\n");
41               }
42               else
43                   printf("\n El led D5 esta apagado\n");
44               break;
45           }
```

Ilustración 85: Código de la función código 10.1

```
case 6:
   {
       LED_D6_Toggle();
      if (LED_D6_GetValue()==1){
          printf("\n El led D6 esta encendido\n");
      }
      else
          printf("\n El led D6 esta apagado\n");
      break;
   }
case 7:
   {
       LED_D7_Toggle();
      if (LED_D7_GetValue()==1){
          printf("\n El led D6 esta encendido\n");
      }
      else
          printf("\n El led D6 esta apagado\n");
      break;
   }
case 1:    //Estado de los leds
   {
      printf("\n Los leds que estan encendidos son:\n");
      if (LED_D4_GetValue()==1)
          printf(" D4 ");
      if (LED_D5_GetValue()==1)
          printf(" D5 ");
      if (LED_D6_GetValue()==1)
          printf(" D6 ");
      if (LED_D7_GetValue()==1)
          printf(" D7 ");
```

Ilustración 84 Código de la función código 10.2

- Comentarios sobre el programa

En este programa se ha utilizado el módulo EUSART, este módulo permite comunicarse a través de un puerto serie con otro aparato, ya sea un ordenador, otro microprocesador…

En este ejemplo se ha utilizado para modificar salidas digitales desde el ordenador, pero también se puede usar para enviar datos recogidos a través de las entradas al ordenador.

4.11. Módulo de comunicaciones serie síncronas (SPI/I2C)

En este ejercicio vamos a utilizar el módulo MSSP para controlar una matriz de leds 8x8 a través del driver MAX7219, a través del pulsador S1 conectado a RC4 iremos mostrando en la matriz los número del 0 al 9.

- Módulo MSSP

El módulo *Master Synchronous Serial Port* (MSSP) es un interfaz de comunicación en serie que permite comunicarnos con otros periféricos o microcontroladores. Este módulo tiene la capacidad de operar en dos modos diferentes: *Serial Peripheral Interface* (SPI) e *Inter-Intergrated Circuit* (I^2C).

El interfaz SPI puede operar tanto en modo Maestro como en modo Esclavo y utiliza la conexión *Full-Duplex*. Esta comunicación utiliza cuatro conexiones: *Serial Clock (SCK)*, *Serial Data Out (SDO), Serial Data In (SDI) y Slave Select (SS)*.

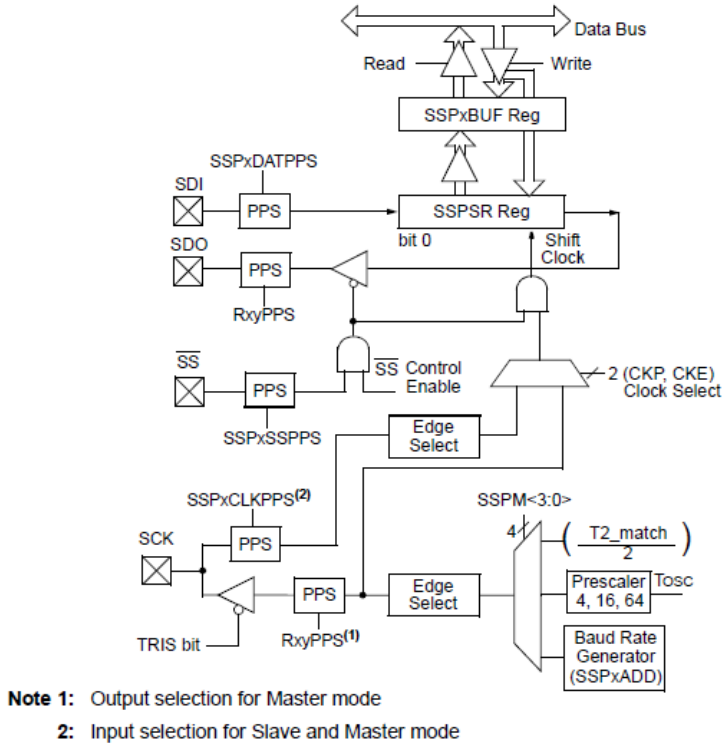


Ilustración 86: Módulo SPI [21]

En la Ilustración 86 está representado el esquema de funcionamiento del módulo SPI.

- Implementación del ejemplo

La matriz de leds que vamos a utilizar es el modelo TOP-CA-788BS (Ilustración 87**¡Error! No se encuentra el origen de la referencia.**), esta matriz incorpora 8 filas de 8 leds cada una. Estas matrices cuentan con 16 pines que permiten controlar individualmente todas las filas y columnas.

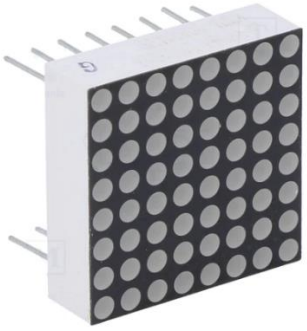


Ilustración 87:Matriz de leds

La relación de los pines con las diferentes filas y columnas está representada en la Ilustración 88 y recibe el nombre de ánodo común, para encender el led que se desee hay que aplicar 5V en la fila y 0V en la columna que correspondan, por ejemplo, para encender el led (1,1) hay que aplicar 5V en el pin 9 y 0V en el pin 13.

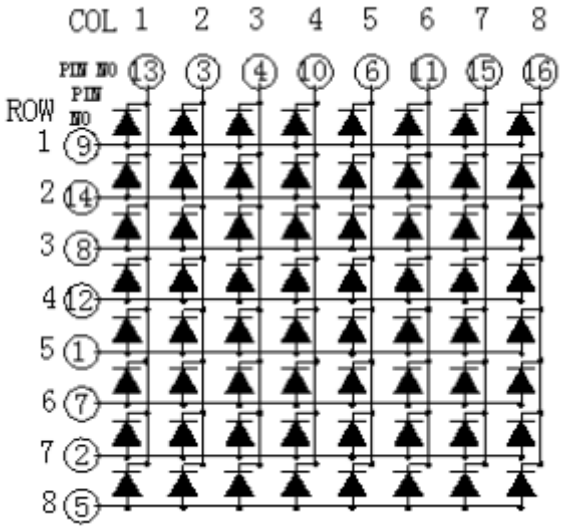


Ilustración 88:Relación de pines y leds [28]

Para evitar tener que utilizar 16 salidas digitales del microcontrolador, se va a utilizar el driver MAX7219, el cual solo utiliza tres salidas del microcontrolador, y permite emplear el resto de los pines del microcontrolador en otras funciones. En la Ilustración 89 se encuentra representada la configuración de los pines del driver MSX7219.

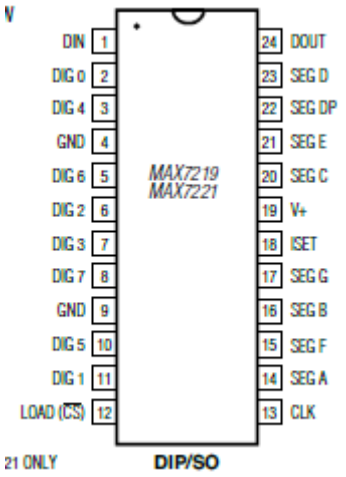


Ilustración 89: Configuración de pines del MAX7219 [29]

Los dígitos del driver están a nivel alto cuando están apagados y a nivel bajo cuando se encuentran encendidos, por lo que los conectaremos a las columnas de la matriz. El funcionamiento de los segmentos es opuesto al de los dígitos, se encuentran a nivel alto cuando están encendidos y a nivel bajo cuando están apagados, estos se conectarán a las filas de la matriz.

Para controlar la corriente que suministra el *driver* a través de sus pines hay que colocar una resistencia entre los terminales V+ (19) y ISET (18), en la siguiente tabla se muestran los valores de las resistencias que hay que conectar para obtener la corriente deseada según la tensión umbral de los leds de la matriz, al mirar el *datasheet* de la matriz veremos que la tensión umbral típica es de 2.1V por lo que conectaremos una resistencia de 27KΩ, valor comercial más próximo, para obtener una intensidad de 20mA.

I_{SEG} (mA)	V_{LED} (V)				
	1.5	2.0	2.5	3.0	3.5
40	12.2	11.8	11.0	10.6	9.69
30	17.8	17.1	15.8	15.0	14.0
20	29.8	28.0	25.9	24.5	22.6
10	66.7	63.7	59.3	55.4	51.2

Ilustración 90:Intensidades de salida del MAX7219 [29]

El montaje del circuito lo realizaremos en una *protoboard* y el resultado es el que aparece en la Ilustración 91.

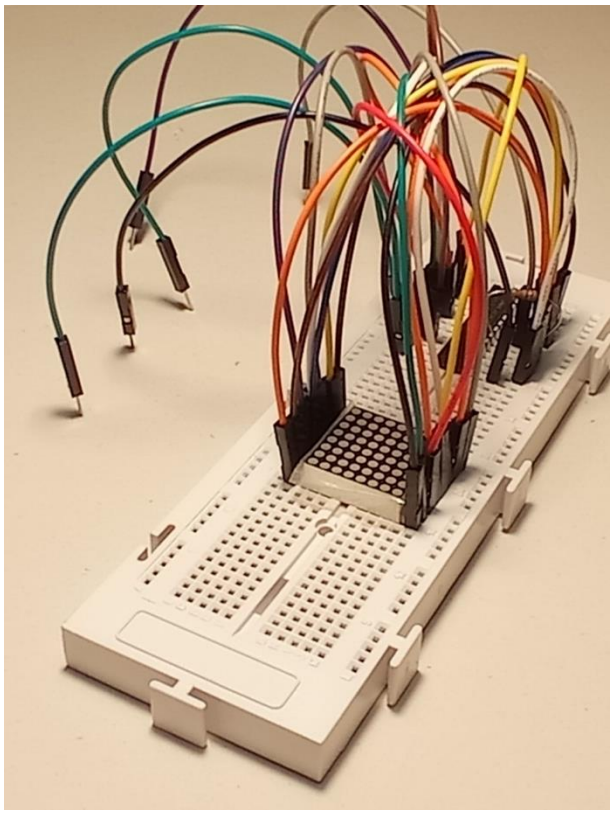


Ilustración 91: Circuito matriz de leds en protoboard

Al acabar de montar el circuito pasaremos a realizar la programación, al igual que en los anteriores ejemplos lo primero que haremos será crear un nuevo proyecto y configurar el sistema como aparece en la Ilustración 93 y añadiremos a los recursos el módulo MSSP1 (Ilustración 92).

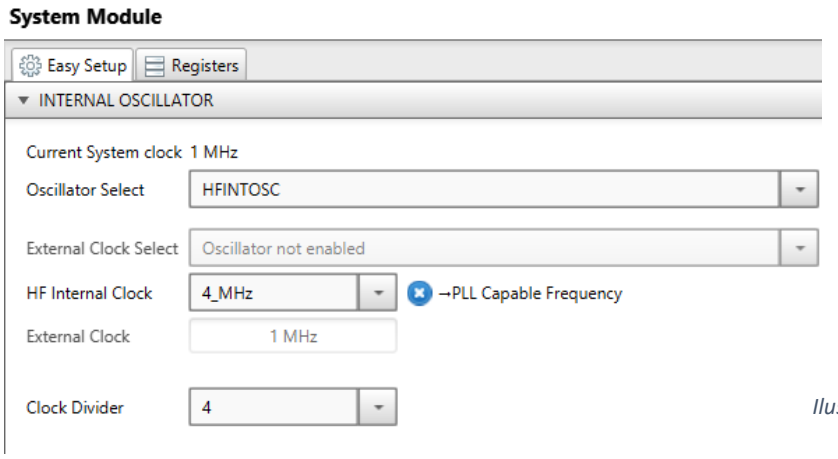


Ilustración 93: Configuración del sistema

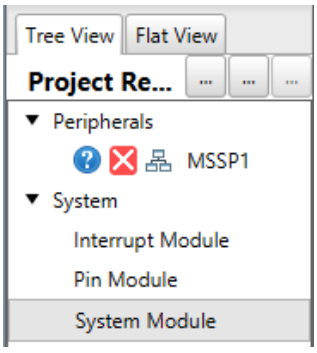


Ilustración 92: Recursos del proyecto

El módulo MSSP lo configuraremos para que trabaje en SPI en modo Maestro, que envíe los datos a mitad del ciclo de reloj y que utilice una frecuencia de 62.500Hz (Ilustración 94).

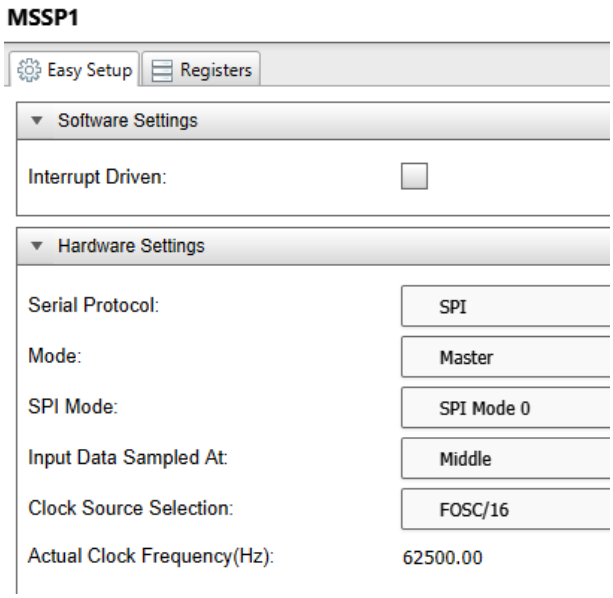


Ilustración 94: Configuración módulo MSSP

Cuando ya tengamos configurado el módulo MSSP pasaremos a asignar las salidas a los pines que queramos, como el microcontrolador va a funcionar en modo Maestro la señal SCK1 tiene que ser de salida. La salida digital asignada al pin RB7 la utilizaremos para seleccionar el esclavo. El pin RC4 lo configuraremos como entrada digital con resistencias de *pull-up* activadas e interrupciones por cambio de nivel negativas. En la **¡Error! No se encuentra el origen de la referencia.** aparece la configuración que vamos a utilizar.

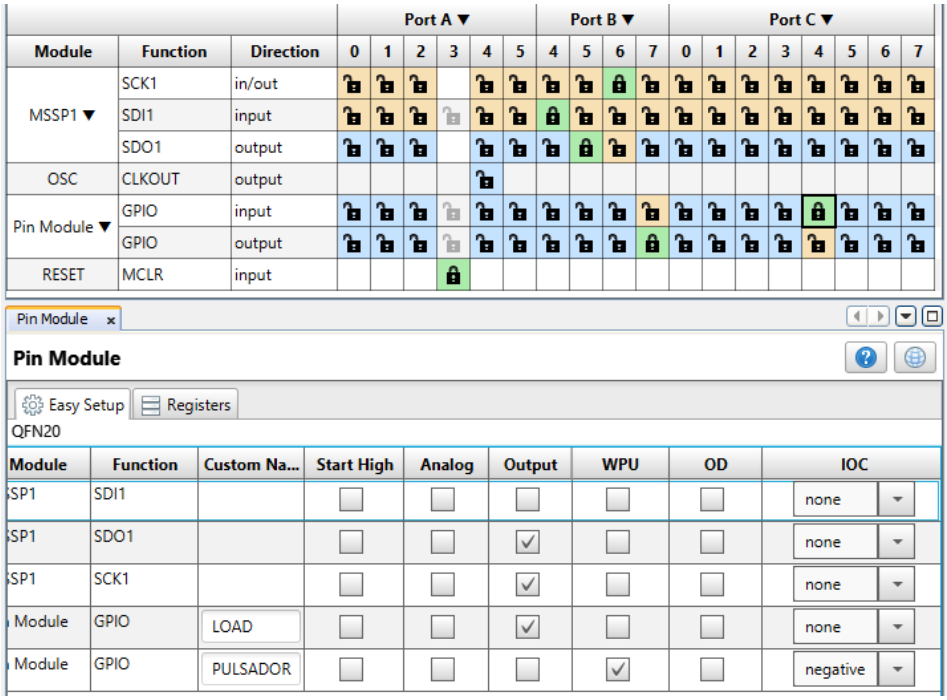


Ilustración 95: Pin Manager ejercicio 11

Al terminar la configuración de los pines haremos clic en el botón *Generate* y pasaremos a escribir el código.

Al utilizar las interrupciones por cambio de nivel será necesario añadir al programa principal el archivo de interrupciones "interrupt_manager.h" con la función "*include*".

En la Ilustración 96 está representado el formato de comunicación serie que utiliza el driver MAX7219.

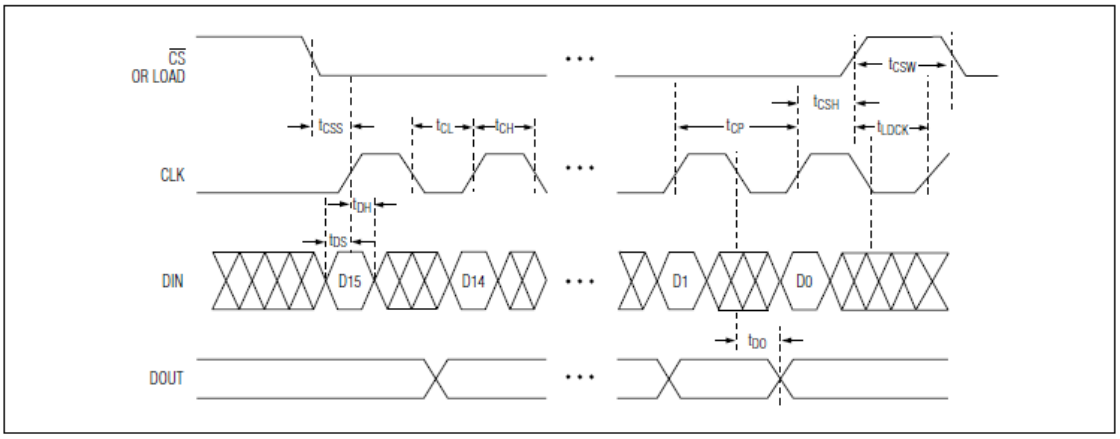


Figure 1. Timing Diagram

Table 1. Serial-Data Format (16 Bits)

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
X	X	X	X	ADDRESS				MSB	DATA						LSB

Ilustración 96: Formato comunicación MAX7219 [29]

Para simplificar el código del programa crearemos una función que utilizaremos para enviar los datos que deseemos a través del módulo SPI. Esta función está en la Ilustración 97, cada vez que envían datos es necesario utilizar la función "SPI1_Open(SPI1_DEFAULT)" para activar la señal de reloj y al finalizar la comunicación utilizar "SPI1_Close()" para desactivarla.

La función "enviarbits" recibe dos variables de tipo "uint8_t", la primera será la columna y la segunda los bits que queremos encender de esa columna. Para enviar los bits utilizaremos la función "SPI1_ExhangeByte" pero antes de enviar los datos es necesario poner a nivel bajo la salida digital LOAD y al acabar volver a ponerla a nivel alto.

```
void enviarbits(uint8_t direccion, uint8_t data){
    SPI1_Open(SPI1_DEFAULT);
    LOAD_SetLow();
    SPI1_ExchangeByte(direccion);
    SPI1_ExchangeByte(data);
    LOAD_SetHigh();
    SPI1_Close();
}
```

Ilustración 97: Función enviarbits

Para empezar a utilizar el driver MAX7219 primero debemos configurar su modo de funcionamiento, en el *datasheet* [29] aparecen todas las opciones de configuración así que aquí se va a comentar la configuración utilizada.

A través de la dirección 0b1001 seleccionaremos el modo sin codificar, con la 0b1010 escogeremos el nivel de intensidad de los leds, este valor puede tomar un valor de 0 a 15 y en este caso escogeremos el valor mínimo porque estamos alimentando el driver a través de la placa de desarrollo. Con la dirección 0b1011 habilitaremos el uso de todos los dígitos y con la 0b1100 seleccionaremos el modo de operación normal.

La función “limpiar” se utiliza para apagar todos los leds.

```
void configuracion(){
    enviarbits(0b00001001,0);
    enviarbits(0b00001010,1);
    enviarbits(0b00001011,7);
    enviarbits(0b00001100,1);
    limpiar();
}
void limpiar() {
    for (uint8_t i = 1; i <= 8; i++) {
        enviarbits(i,0);
    }
}
```

Ilustración 98: Funciones ejercicio 11

En un vector de números uint8_t guardaremos los bits necesarios para representar cada número en la matriz Ilustración 99.

```
uint8_t numeros[80]={
//0
    0b00111100,0b01000010,0b01000010,0b01000010,0b01000010,0b01000010,0b01000010,0b00111100,
//1
    0b00001000,0b00011000,0b00101000,0b01001000,0b00001000,0b00001000,0b00001000,0b00011100,
//2
    0b00111100,0b01000010,0b01000010,0b00001000,0b00010000,0b00100000,0b01000000,0b01111100,
//3
    0b01111100,0b00000110,0b00001100,0b00111000,0b00111000,0b00001100,0b00000110,0b01111100,
//4
    0b01000010,0b01000010,0b01000010,0b01111110,0b00000010,0b00000010,0b00000010,0b00000010,
//5
    0b00111100,0b01000000,0b01000000,0b01111000,0b00000100,0b00000100,0b00000100,0b00111100,
//6
    0b00111100,0b01000000,0b01000000,0b01111000,0b01000100,0b01000100,0b01000100,0b00111100,
//7
    0b01111110,0b00000110,0b00000110,0b00000110,0b00011000,0b00110000,0b01100000,0b01100000,
//8
    0b00111100,0b01000010,0b01000010,0b00111100,0b00111100,0b01000010,0b01000010,0b00111100,
//9
    0b00111100,0b01000010,0b01000010,0b01000010,0b00111110,0b00000010,0b00000010,0b01111100
};
```

Ilustración 99: Bits de representación de números

La última función que implementaremos será la de atención a la interrupción por cambio de nivel provocada por el pulsador, esta función será similar a la utilizada en el ejercicio 4.2 pero en este caso el valor máximo será 9.

```
int veces=0;
uint8_t i;
void pulsador(){
    veces++;
    if (veces==10){
        veces=0;
    }
}
```

Ilustración 100: Función de atención a la interrupción

En el código del programa debemos habilitar las interrupciones globales y periféricas, declarar la función pulsador como la de atención a la interrupción y llamar a la función "configuracion" para inicializar el driver MAX7219.

Con la función "enviarbits" enviaremos las direcciones de columna y los leds a encender en cada una de ellas.

```
void main(void)
{
    SYSTEM_Initialize();
    INTERRUPT_GlobalInterruptEnable();
    INTERRUPT_PeripheralInterruptEnable();
    IOCCF4_SetInterruptHandler(pulsador);
    configuracion();
    while (1)
    {
        for(i=0;i<8;i++){
        enviarbits(i+1,numeros[veces*8+i]);
        }
    }
}
```

Ilustración 101: Código del programa 11

- Comentarios sobre el programa

Con este programa hemos aprendido cómo funciona la comunicación SPI del microcontrolador. Además, con este ejemplo se ha visto cómo podemos controlar 64 leds con tan sólo cuatro salidas del microcontrolador. Este driver se puede conectar en serie con otros y poder crear paneles leds de mayor tamaño.

5. Proyecto final

En este apartado se van a explicar paso a paso la realización de dos proyecto más complejos en los que se van a utilizar varios de los módulos que incorpora el microcontrolador PIC16F18446.

El primer proyecto que vamos a realizar consistirá en una central domótica que controle diferentes elementos como son luces, persianas y climatización y una APP móvil que nos permita controlar todo el sistema.

El segundo proyecto iba a formar parte del primer proyecto, pero al contar con un microcontrolador de 20 pines se ha preferido separarlo para mejorar su calidad.

5.1. Proyecto 1

5.1.1. Sensores

El sistema que vamos a desarrollar va a contar con un sensor de temperatura (LM35DZ), un sensor de movimiento (HC-SR501) y un sensor de luminosidad (LDR), estos sensores tienen un bajo coste, su precio es de 1€ cada uno aproximadamente, y nos servirán para simular un sistema real.

5.1.1.1. Sensor de temperatura LM35DZ

El sensor de temperatura que vamos a utilizar es un sensor analógico con una sensibilidad de 10mV/ºC y cuando la señal es de 0V la temperatura es de -5ºC por lo que el valor de la temperatura viene dado por la siguiente función:

$$T = -5º + (\frac{V}{0.01})$$

Las especificaciones técnicas son las siguientes:
Tensión de alimentación: -0.2 – 35V
Tensión de salida: -1V – 6V
Corriente de salida: 10mA
Intervalo de medida: 0 - 100ºC

Este sensor viene soldado a una placa con un circuito de conexión con tres pines para alimentación y salida de señal, en la Ilustración 102 se puede observar dicho circuito [24].

Ilustración 102: Circuito sensor Tª

5.1.1.2. Sensor de luminosidad LDR

Este sensor consiste en una resistencia cuyo valor es dependiente de la luz dentro del rango visible [25].
La alimentación necesaria es de 5V y su señal de salida es una tensión de 0 a 5V, cuando no hay luz la señal es de 5V y cuando hay mucha luz es próxima a 0V.
Al igual que el sensor de temperatura, este sensor también viene soldado en un circuito que nos facilita su conexión. En el pin donde está la S es el de señal, el del centro es el de alimentación positiva y el tercero es el de GND.

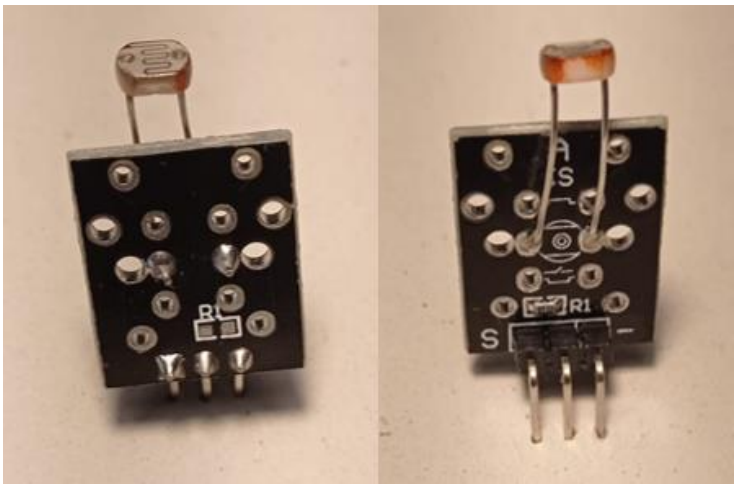


Ilustración 103: Sensor luminosidad LDR

5.1.1.3. Sensor de movimiento HC-SR501

El sensor HC-SR501 es un sensor de movimiento PIR, este tipo de sensores captan la luz infrarroja emitida por personas y animales y se pude detectar su movimiento. La cúpula blanca (lente de Fresnel) sirve para concentrar la radiación infrarroja en el sensor que se encuentra en el interior [26].

Ilustración 104: Sensor de presencia HC-SR501 [26]

En la Ilustración 105 podemos observar la utilidad de todos los elementos que incorpora el sensor para su ajuste.

Ilustración 105: Configuración Sensor HC-SR501 [26]

Para incrementar el tiempo del retraso de disparo hay que girar el potenciómetro en sentido horario y para aumentar la distancia en sentido antihorario.

En el modo un solo disparo, una vez que el sensor detecta movimiento mantiene una señal de salida de nivel alto hasta que se vuelve a reiniciar, este modo se utiliza en alarmas.

En el modo de disparos repetitivos cuando detecta movimiento mantiene una señal a nivel alto el tiempo de retraso que se ha elegido mediante el potenciómetro y al transcurrir ese periodo de tiempo puede volver a detectar movimiento.

5.1.2. Módulo Bluetooth HC-06

La placa de desarrollo lleva incorporada la *footprint* del chip de Bluetooth RN4020, lo ideal sería utilizar este módulo, pero por problemas de este dispositivo no se encuentra disponible hasta dentro de un año, por lo que hemos decidido utilizar el módulo HC-06.

Ilustración 106: Módulo Bluetooth HC-06 [27]

Este módulo sirve para trabajar únicamente en modo esclavo y tiene cuatro pines de conexión, estos pines son: [27]

-VCC: pin de alimentación entre 3.3-6V.
-GND: este pin hay que conectarlo a GND de la placa utilizada.
-TX: a través de este pin el módulo envía los datos a la placa, por lo que se conectará a RX de la placa.
-RX: a través de este pin el módulo recibe los datos de la placa y se conecta al pin TX de la placa.

5.1.3. Programación

Una vez que tenemos todos los componentes ha llegado el momento de realizar la programación.

Lo primero que haremos será exponer que es lo que queremos que haga nuestro programa, para así asignar los pines a los elementos de manera correcta, porque no todos los pines están asociados a los mismos módulos por lo que habrá que mirar el *datasheet* [21] antes de asignarlos.

-Sensor de temperatura

El sensor de temperatura es analógico por lo que su señal estará conectada al CAD y nos servirá para controlar la climatización, cuyo funcionamiento será simulado con la activación del LED1 que está conectado al terminal RC5. A través de la APP móvil se habilitará el control de temperatura y se enviará la temperatura mínima y máxima.

-Sensor de luminosidad

El sensor de luminosidad produce una señal analógica y su valor varía según la luz ambiental, en este caso no conectaremos la señal al CAD, sino que lo haremos a un comparador analógico. Según sea el valor se encenderá el LED2 que está conectado al terminal RA5, además este led se podrá encender manualmente a través de una APP cuando el control de luminosidad esté desactivado, la activación del control también se

realizará a través de la APP. Para poder habilitar y deshabilitar el control se utilizarán puertas lógicas siguiendo el siguiente esquema:

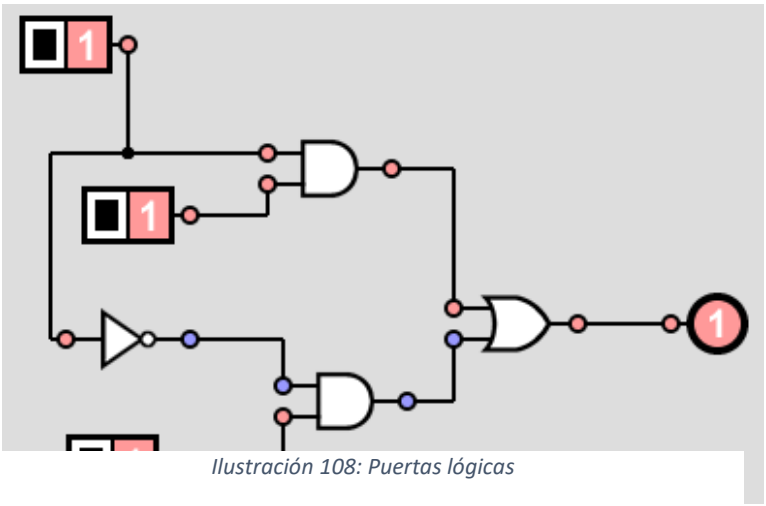


Ilustración 108: Puertas lógicas

IN	IN	IN	OUT
0	0	0	0
1	0	0	0
0	1	0	0
1	1	0	1
0	0	1	1
1	0	1	0
0	1	1	1
1	1	1	1

Ilustración 107: Tabla de verdad

La primera entrada será el terminal RC6 que estará conectado al pin RC7, este último estará a nivel alto cuando el control se encuentre habilitado.

La segunda entrada será la salida del comparador analógico.

La tercera entrada será el pin RC1 que estará conectado al RC0, este pin se encontrará a nivel alto cuando se encienda la luz manualmente.

-Sensor de presencia

La señal del sensor de presencia es digital por lo que irá conectado a una entrada digital, en este caso será el pin RA4 y encenderá el LED3 conectado al terminal RA2 cuando el sensor detecte movimiento. Al igual que los anteriores, desde la APP se habilitará el control de presencia y cuando este se encuentre desactivado se podrá controlar el led manualmente desde la aplicación.

-Módulo Bluetooth

La comunicación del módulo bluetooth con el microcontrolador se hace a través del módulo EUSART, los pines asociados a este módulo son RB5 y RB7.

-Driver MAX7219

El driver MAX7219 utiliza el protocolo SPI y necesita cuatro pines del microcontrolador uno para la señal de reloj (RB6), datos de entrada (RC3), datos de salida (RB4) y uno que habilite la comunicación (RA1).

En la siguiente tabla están representados los pines del microcontrolador con su función:

Terminales	Elemento asociado	Configuración
RA0	Sensor de Tª	Entrada analógica
RA1		
RA2	LED 3 (Sensor de presencia)	
RA3		MCLR
RA4	Sensor de presencia	Entrada digital
RA5	LED 2 (Sensor de luminosidad)	
RB4		
RB5	Bluetooth	EUSART
RB6		
RB7	Bluetooth	EUSART
RC0		Salida digital
RC1		Entrada digital
RC2	Sensor de luminosidad	
RC3		
RC4	Pulsador	Entrada digital
RC5	LED 1 (Sensor Tª)	
RC6		Entrada digital
RC7		Salida digital

Tras definir el funcionamiento del sistema y asignar todos los pines a las conexiones con los diferentes elementos ha llegado el momento de empezar a programar.

Lo primero será crear un nuevo proyecto como hemos explicado anteriormente en el apartado 3.1 y abrir el MCC. A partir de aquí se explicará la programación de cada elemento por separado para que resulte más sencillo.

-Configuración del sistema

Para configurar la frecuencia a la que funcionará el sistema iremos a la pestaña de *System Module* y seleccionaremos HFINTOSC para que utilice el oscilador interno del microcontrolador y una frecuencia de 32MHz con cuatro divisiones de reloj, lo que nos dará una frecuencia de 8MHz (Ilustración 109).

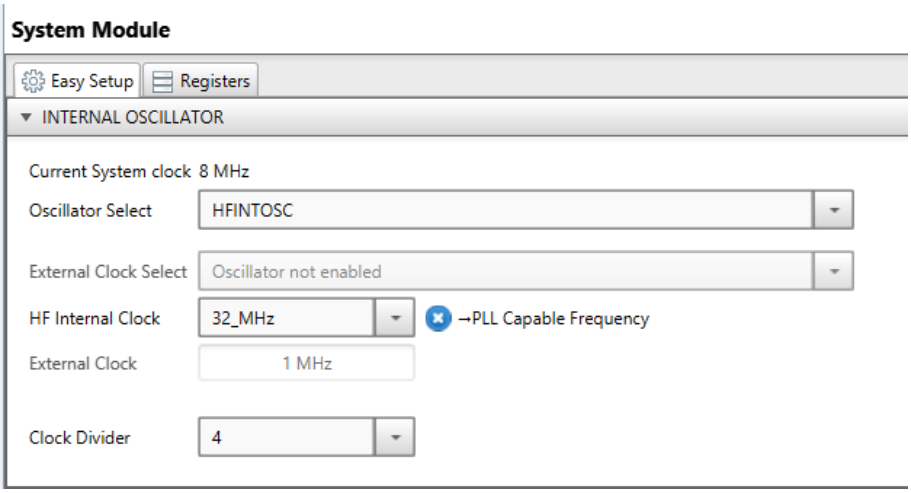


Ilustración 109: Configuración de la frecuencia

-Sensor de temperatura

Como se ha explicado anteriormente el sensor de temperatura es analógico, por lo que necesitaremos añadir el Conversor Analógico Digital (CAD) a los recursos del proyecto desde el MCC (Ilustración 110).

Una vez añadido el CAD procederemos a su configuración. Utilizaremos modo básico con el oscilador interno (FRC), el resultado de 12 bits estará alineado a la derecha y las referencias serán VDD (5V) y VSS (0V) (Ilustración 111).

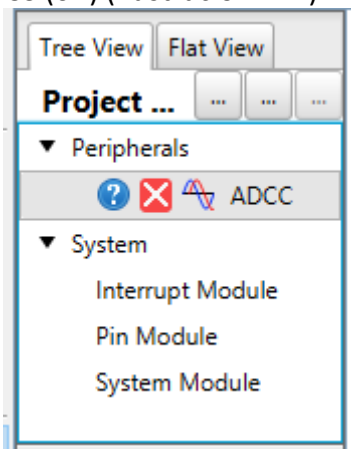


Ilustración 110: Recursos del proyecto 1

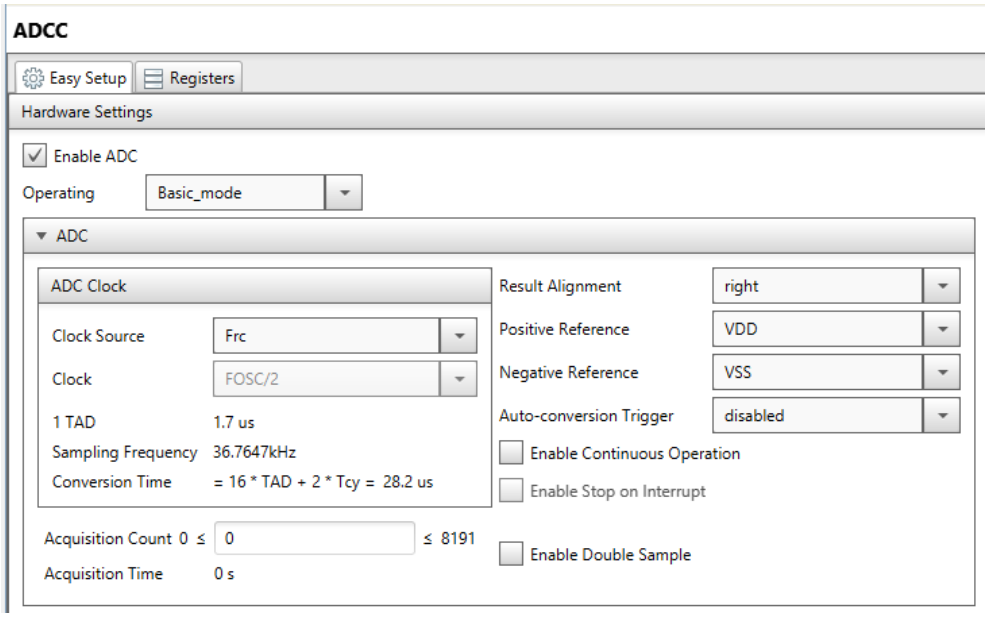


Ilustración 111: Configuración CAD

El pin en el que conectaremos la señal será RA0 y a través del *Pin Module* le asignaremos el nombre "TEMP" para facilitar la programación y configuraremos el pin RC5 como salida digital y le daremos el nombre de LED1 (Ilustración 112).

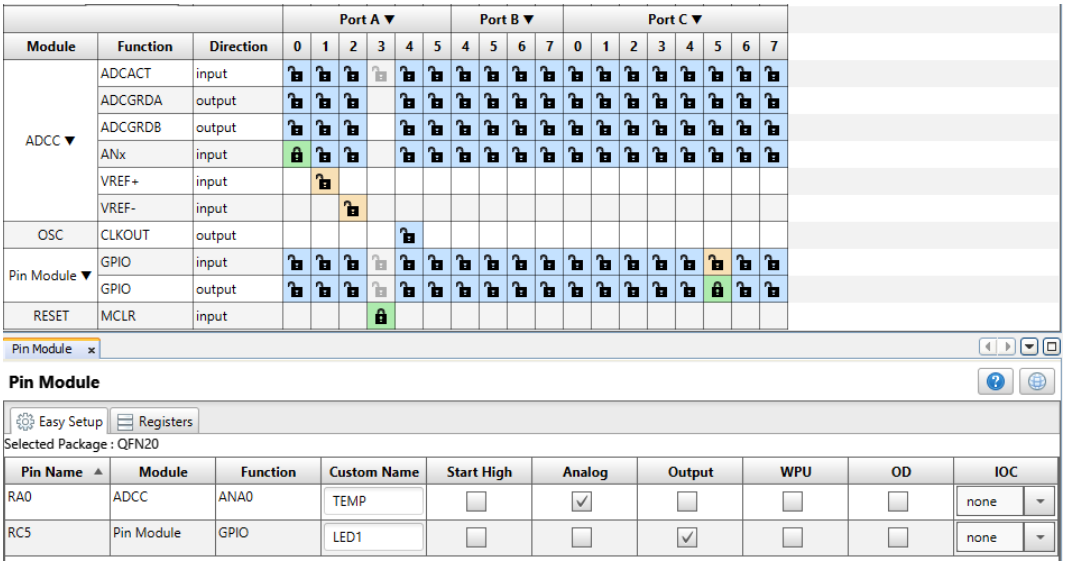


Ilustración 112: Pin Module Temperatura

Una vez acabada la configuración haremos clic en el botón *Generate* y pasaremos a escribir las líneas del programa.

Crearemos una función llamada "temperatura" para facilitar el código. En la Ilustración 113 está representado el código de la función.

```
int TEMP_HAB=0,TMIN,TMAX;
uint16_t resultado;
float resultado2;
void temperatura(void){
    resultado = ADCC_GetSingleConversion(TEMP);
        resultado2 = (resultado*(5000/4096)*0.1)+5;
        if (TEMP_HAB==1){
            if (resultado2 < TMIN){
               LED1_SetHigh();
            }
            if (resultado2 > TMAX){
                LED1_SetLow();
            }
        }
}
```

Ilustración 113: Código función temperatura

El resultado de la conversión analógica digital se almacena en la variable resultado de tipo uint16_t, que es una variable de 16 bits. Como el conversor tiene una resolución de 12 bits, el valor digital de la señal puede tomar 2^{12} = 4096 valores diferentes, entonces para saber el voltaje de la señal utilizaremos la siguiente ecuación:

$$V = \frac{V_{ref+} - V_{ref-}}{2^{nbits}}$$

Como la referencia superior es 5V y la inferior 0V la ecuación resultante es:

$$V = \frac{5 - 0}{4096}$$

Ahora para saber la temperatura captada por el sensor deberemos dividir entre la sensibilidad, que la de este sensor es de 10mV, y sumarle 5ºC al resultado de esta división. En el código del programa se han puesto los 5V de la señal de referencia en mV y la división como la multiplicación del inverso. El valor de la temperatura se almacenará en la variable tipo *float* "resultado2".

La variable "TEMP_HAB" nos servirá para saber si el control de climatización está activado o no, estará activado cuando sea igual a 1 y cuando sea igual a 0 no. Este variable al igual que "TMAX" y "TMIN" se cambiará a través de la APP móvil.

La calefacción se encenderá cuando la temperatura sea menor de TMIN y se apagará en el momento que sea superior a TMAX.

-Sensor de luminosidad

Como se ha explicado en el enunciado del programa, el control de la luminosidad se hará a través de un comparador analógico. Por lo cual añadiremos a los recursos del proyecto el CMP1, el Conversor Digital Analógico (CDA) y además las puertas lógicas CLC1 para la activación manual de la luz y activar y desactivar el control de luminosidad (Ilustración 114).

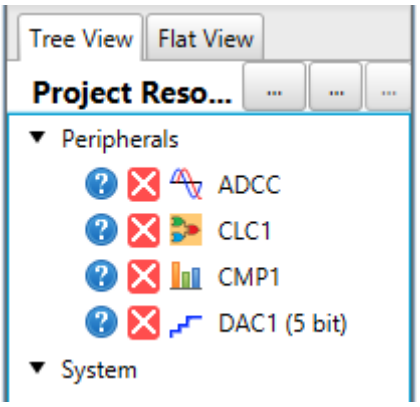


Ilustración 114: Recursos del proyecto 2

El CDA se utilizará para generar una señal de referencia e introducirla en el comparador analógico, en este caso generaremos una señal de 2V y como esta señal la utilizará internamente el microcontrolador no tenemos que habilitar la salida del DACOUT (Ilustración 115).

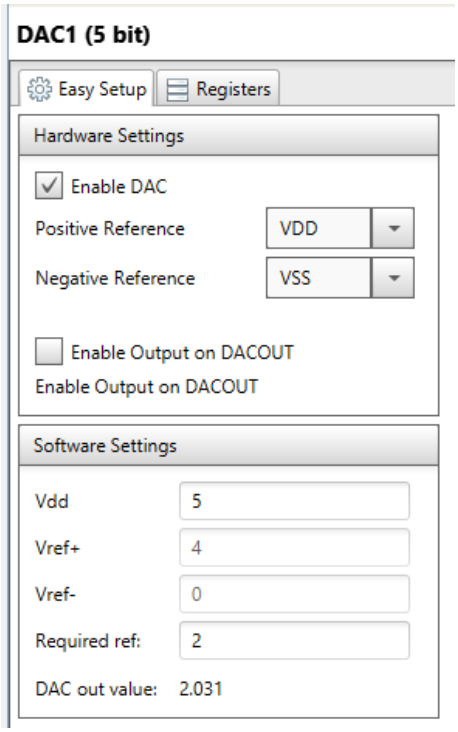


Ilustración 115: Configuración CDA

Como ya tenemos configurada la señal de referencia pasaremos a configurar el comparador. En la entrada positiva tendremos la señal de referencia y en la negativa la señal que viene del sensor de luminosidad, la salida la invertiremos para que cuando la señal del sensor sea menor que la de referencia la salida del comparador sea a nivel alto (Ilustración 116).

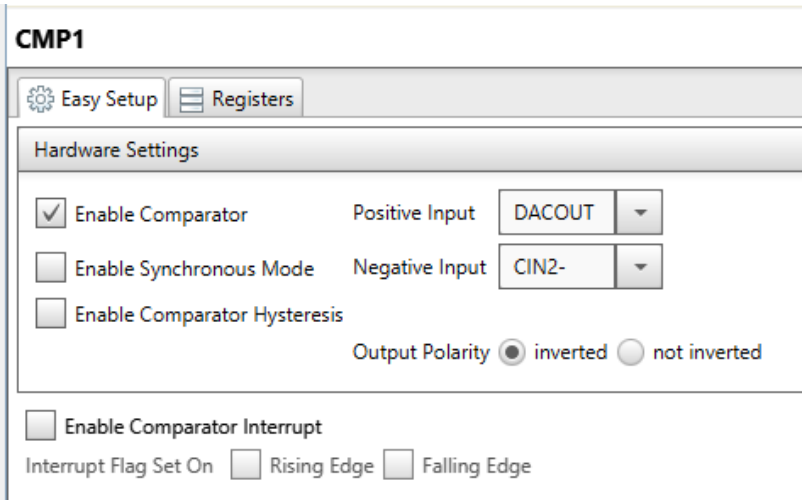


Ilustración 116: Configuración comparador analógico

Por último, debemos configurar las puertas lógicas de la misma forma que aparece en la Ilustración 117, las entradas CLCN0 y CLCN1 se asignarán a los pines RC6 y RC1 respectivamente, la salida de las puertas lógicas se asignará al pin RA5, este pin está conectado a un led.

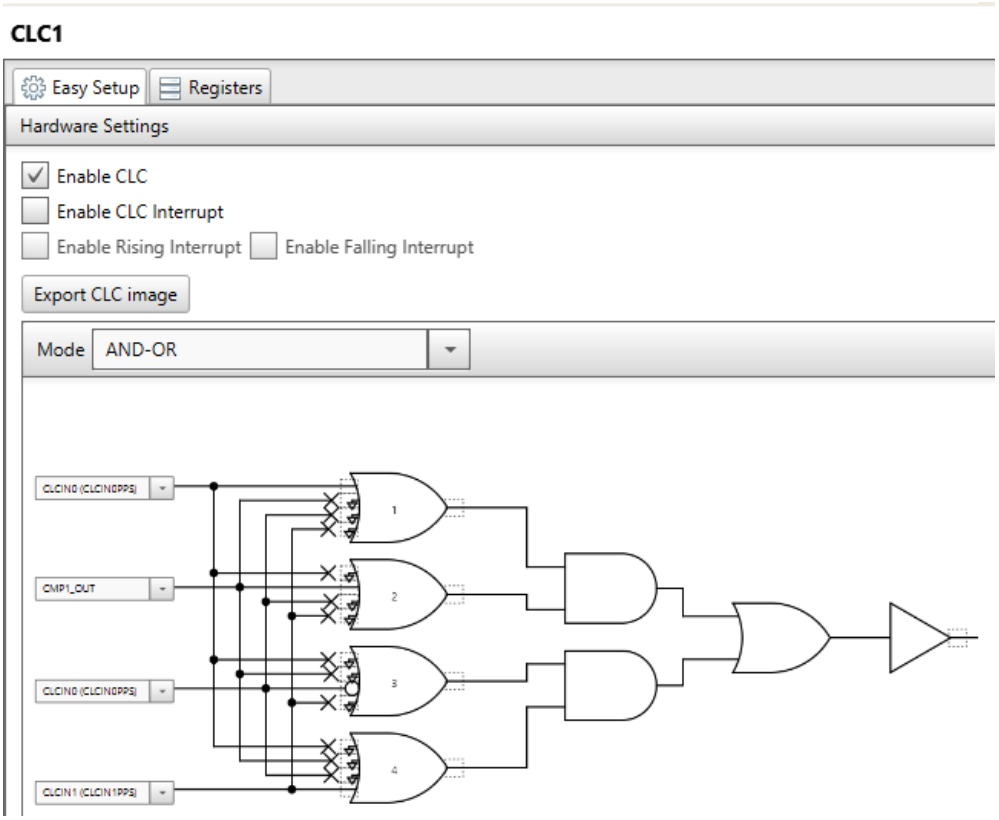
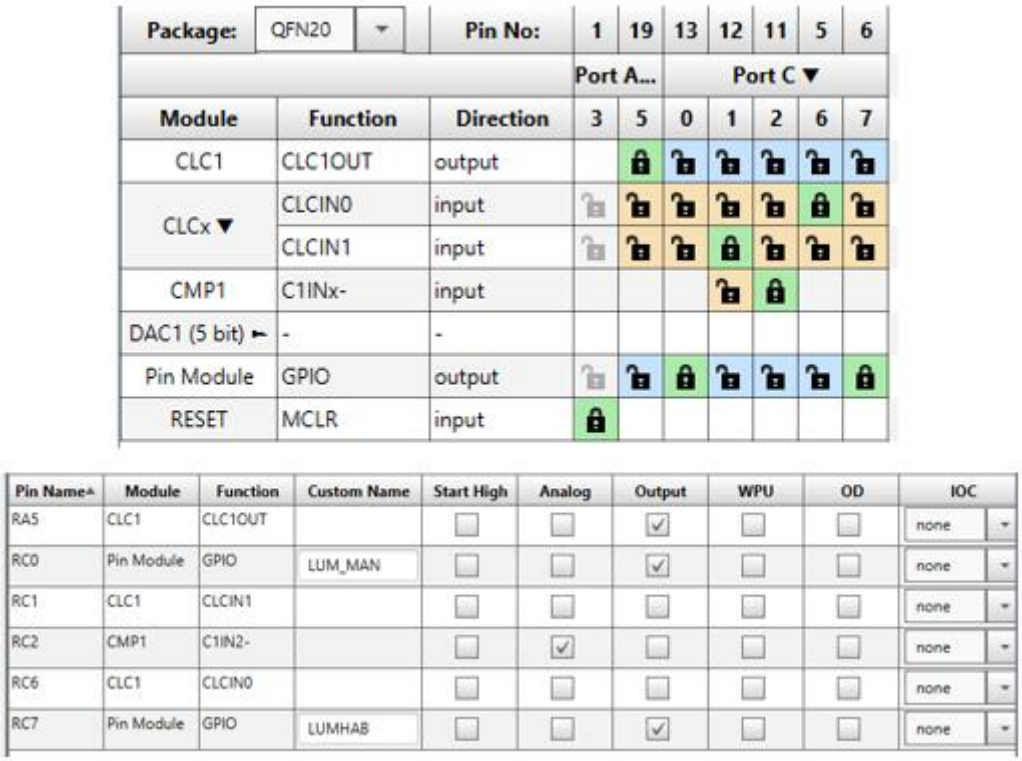


Ilustración 117: Puertas lógicas MPLAB X

En el *Pin Module* configuraremos los pines como aparece en la Ilustración 118, en esta imagen aparecen únicamente los pines utilizados para el sensor de luminosidad. Los pines RC7 y RC0 son salidas digitales, la primera estará a nivel alto cuando el control de luminosidad este activado y se conectará a RC6 y la segunda cuando manualmente se pulse el botón de encender y se conectará a RC1, ambas instrucciones se realizan a través de la APP por lo que el código necesario lo escribiremos cuando configuremos el módulo EUSART.



Package:	QFN20	Pin No:	1	19	13	12	11	5	6
			Port A...		Port C ▼				
Module	Function	Direction	3	5	0	1	2	6	7
CLC1	CLC1OUT	output							
CLCx ▼	CLCIN0	input							
	CLCIN1	input							
CMP1	C1INx-	input							
DAC1 (5 bit)	-	-							
Pin Module	GPIO	output							
RESET	MCLR	input							

Pin Name	Module	Function	Custom Name	Start High	Analog	Output	WPU	OD	IOC
RA5	CLC1	CLC1OUT		☐	☐	☑	☐	☐	none
RC0	Pin Module	GPIO	LUM_MAN	☐	☐	☑	☐	☐	none
RC1	CLC1	CLCIN1		☐	☐	☐	☐	☐	none
RC2	CMP1	C1IN2-		☐	☑	☐	☐	☐	none
RC6	CLC1	CLCIN0		☐	☐	☐	☐	☐	none
RC7	Pin Module	GPIO	LUMHAB	☐	☐	☑	☐	☐	none

Ilustración 118: Pin Manager luminosidad

-Sensor de presencia

Para este sensor no es necesario añadir ningún recurso al proyecto. En este caso para evitar escribir código utilizaremos interrupciones por cambio de nivel, cuando el sensor detecte presencia la salida cambiará de nivel bajo a nivel alto y pasado el tiempo de retardo desde que ha dejado de detectar la señal volverá a nivel bajo.

En el *Pin Manager* configuraremos RA4 como entrada digital y RA2 como salida digital, a esta última está conectado el led D6 de la placa. En la casilla IOC del *Pin Module* del terminal RA4 seleccionaremos la opción "*any*" para detectar tanto el flanco de subida como de bajada (Ilustración 119).

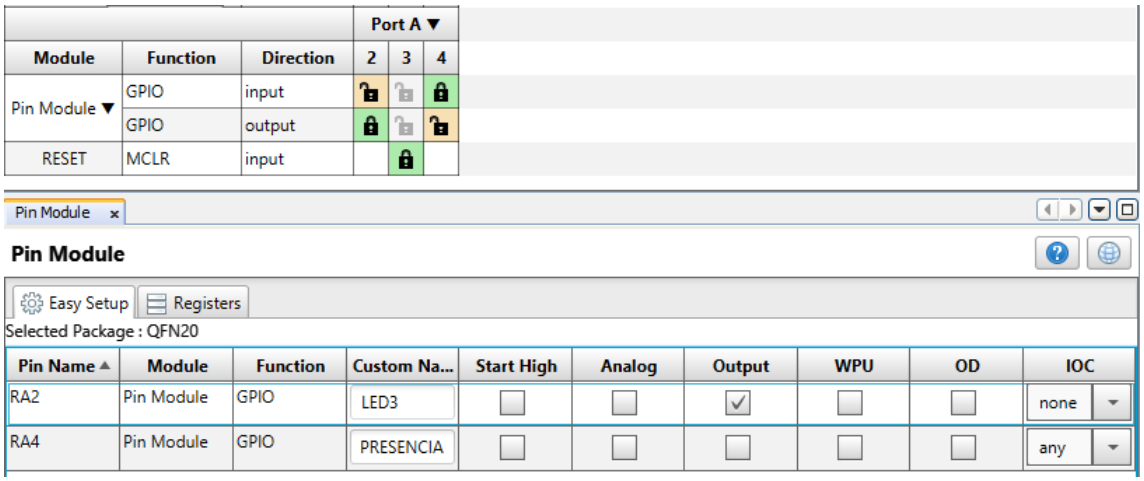


Ilustración 119: Pin Manager presencia

Después de configurar los pines, haremos clic en el botón *Generate* y pasaremos a escribir el código en el archivo "main.c".

Una vez abierto este archivo es necesario añadir el archivo "interrupt_manager.h" y habilitar las interrupciones globales y periféricas quitando las "//" de las funciones que vienen ya escritas como se explicó en el apartado 4.2.

```
#include "mcc_generated_files/mcc.h"
#include "mcc_generated_files/interrupt_manager.h"
int subida=0;
volatile int PRES_HAB=0;
void presencia (void){
    if (PRES_HAB==1){
        if(subida==0){
            LED3_SetHigh();
            subida=1;
        }
        else{
           LED3_SetLow();
           subida=0;
        }
    }
}
void main(void)
{
    SYSTEM_Initialize();
    INTERRUPT_GlobalInterruptDisable();
    INTERRUPT_PeripheralInterruptDisable();
    IOCAF4_SetInterruptHandler(presencia);
    while (1)
    {
    }
}
```

Ilustración 120: Código sensor de presencia

En la Ilustración 120 está el código utilizado para el sensor de presencia, mediante la función IOCAF4_SetInterruptHandler(presencia) ejecutaremos la función presencia cada vez que ocurra una interrupción por cambio de nivel en el terminal RA4. Como el flanco de subida y de bajada ocasionan la misma interrupción utilizaremos la variable subida para diferenciar uno de otro. Siempre va a ocurrir primero el flanco de subida que se da cuando se detecta presencia y por tanto es cuando tenemos que encender el led 3.

Al igual que en los anteriores sensores, se utiliza una variable PRES_HAB que se modifica a través de la aplicación móvil que es igual a cero cuando el control está desactivado y uno cuando si lo está. El sensor de presencia sólo activará el led 3 si el control esta activado.

-Módulo *bluetooth*

El módulo *bluetooth* HC-06 utiliza la comunicación EUSART. Lo primero que debemos hacer es abrir el MCC y añadir este módulo a los recursos del proyecto (Ilustración 121).

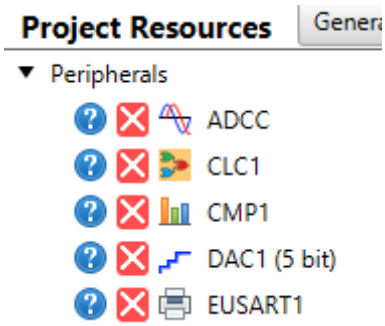


Ilustración 121: Recursos del proyecto 4

Configuraremos el módulo EUSART a una velocidad de 9600 baudios y para que envíe y reciba 8 bits sin invertir, además habilitaremos las interrupciones y la utilización de la librería STUDIO (Ilustración 122). Los pines que utilizaremos para la comunicación son RB5 como RX y RB7 como TX (Ilustración 123).

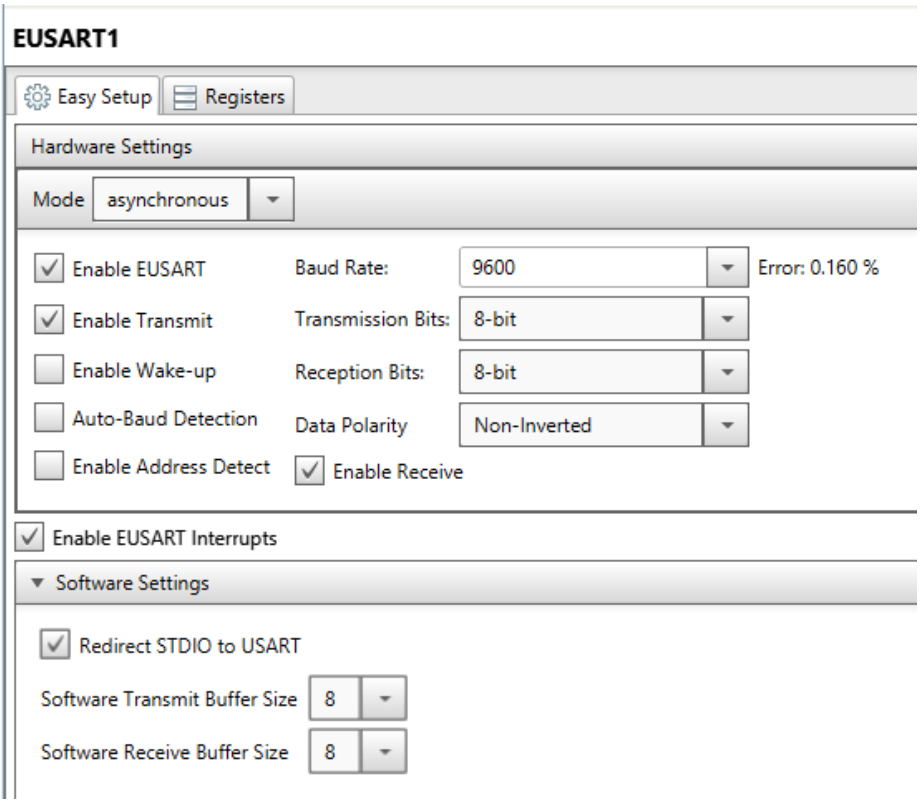


Ilustración 122: Configuración módulo EUSART

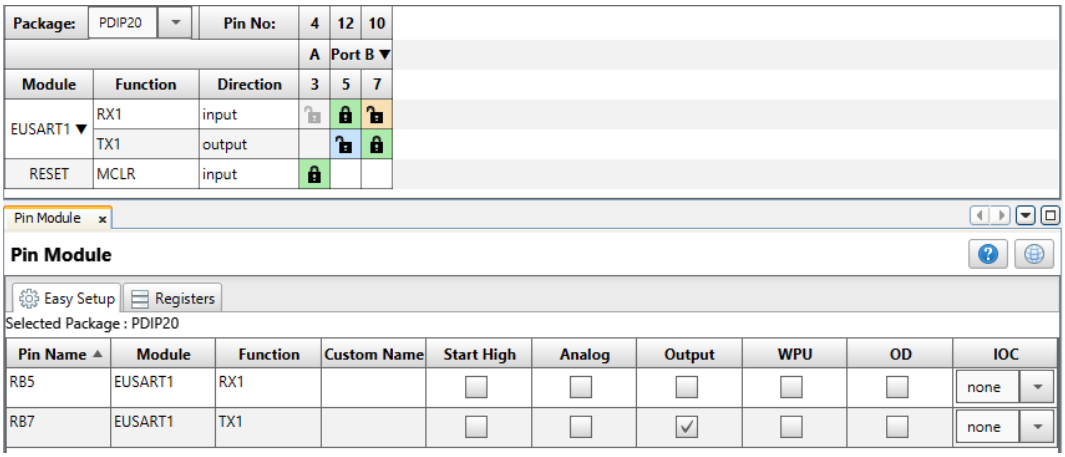


Ilustración 123: Pin Manager EUSART

Tras acabar la configuración del módulo EUSART haremos clic en el botón *Generate* y ya podremos escribir el código del programa.

Antes de escribir el código debemos tener en cuenta cómo va a funcionar la APP móvil que se comunicará con el microcontrolador a través del módulo *bluetooth*, esta aplicación la desarrollaremos a través de la plataforma APPInventor2, desarrollada por el MIT. Esta plataforma es de uso gratuito y nos permite desarrollar aplicaciones mediante la programación de bloques, por lo que se pueden crear aplicaciones sin poseer grandes conocimientos de programación.

Para crear la aplicación iremos a la plataforma de APPInventor2 a través del siguiente enlace https://appinventor.mit.edu/, haremos clic en el botón "*Create Apps!*" y nos tendremos que registrar en la web. Una vez registrados le daremos a "Comenzar un proyecto nuevo" y ya podremos empezar a diseñar nuestra aplicación. Una herramienta muy interesante que tiene esta plataforma es la aplicación móvil "MIT AI2 Companion" que a través de un código QR generado al darle al botón "Conectar" y "AI Companion" te permite ver en tu teléfono móvil como está quedando el diseño de la aplicación.

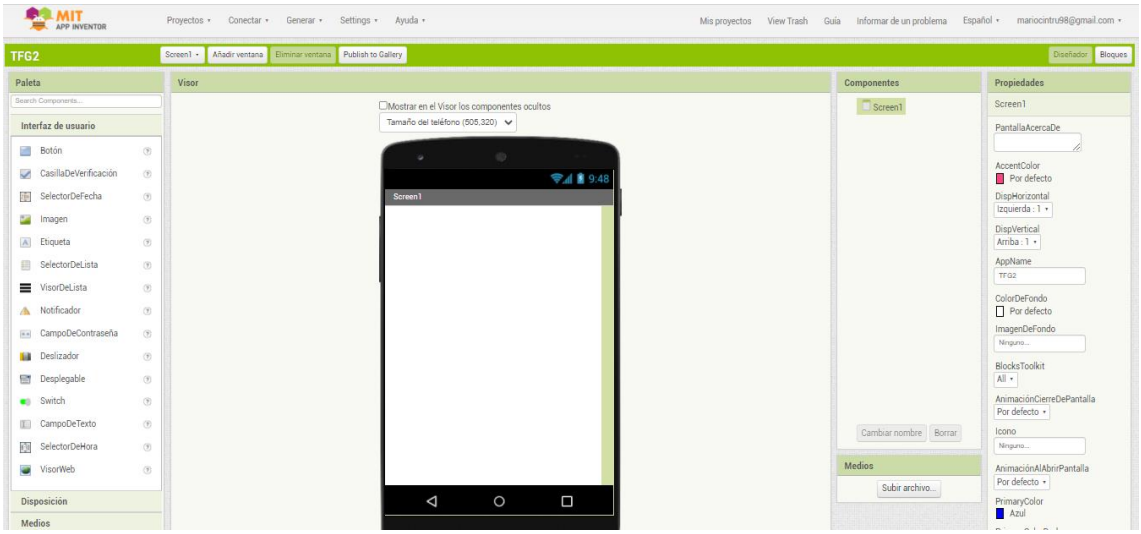


Ilustración 124: Pantalla de creación de aplicación

La aplicación que vamos a crear va a tener dos pantallas, la primera será de portada y la segunda será con la que realicemos el control, para añadir una segunda pantalla hay que hacer clic en "Añadir Ventana" de la barra superior. En la Ilustración 125 están las imágenes de las dos pantallas de la aplicación.

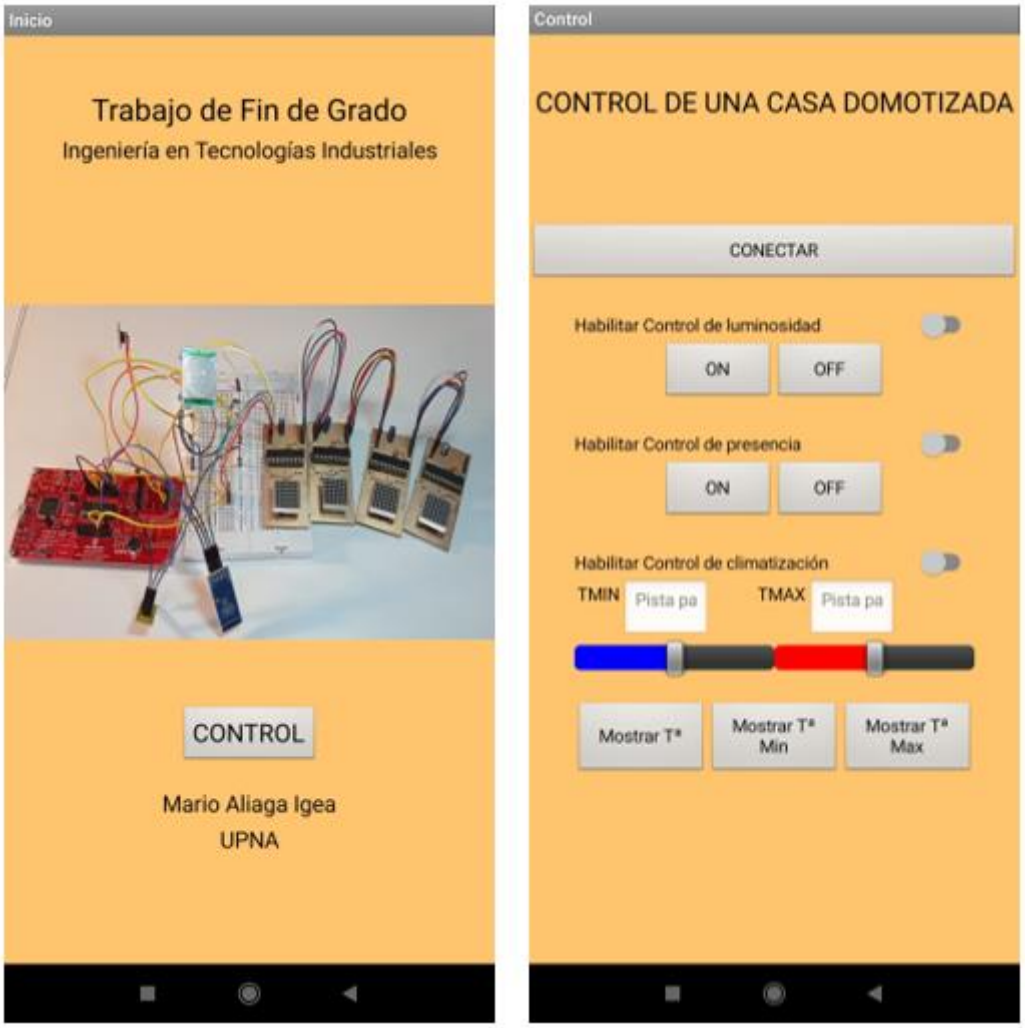


Ilustración 125: Pantallas APP móvil

Para poner el texto de la primera pantalla utilizaremos diferentes etiquetas y dentro de las propiedades de cada una elegiremos el tamaño que queramos. Para insertar una imagen añadiremos una disposición horizontal de la pestaña "Disposiciones" de la paleta, cuando la arrastremos a la pantalla iremos a las propiedades y en imagen buscaremos en el ordenador la que queramos utilizar.

El botón de control se añade desde la pestaña "interfaz con el usuario" y en sus propiedades se puede definir su tamaño, color, texto y colocación.

Para que haya separación entre los elementos que hemos añadido insertaremos disposiciones verticales y desde el menú de propiedades de cada una definiremos su tamaño según la separación que queramos.

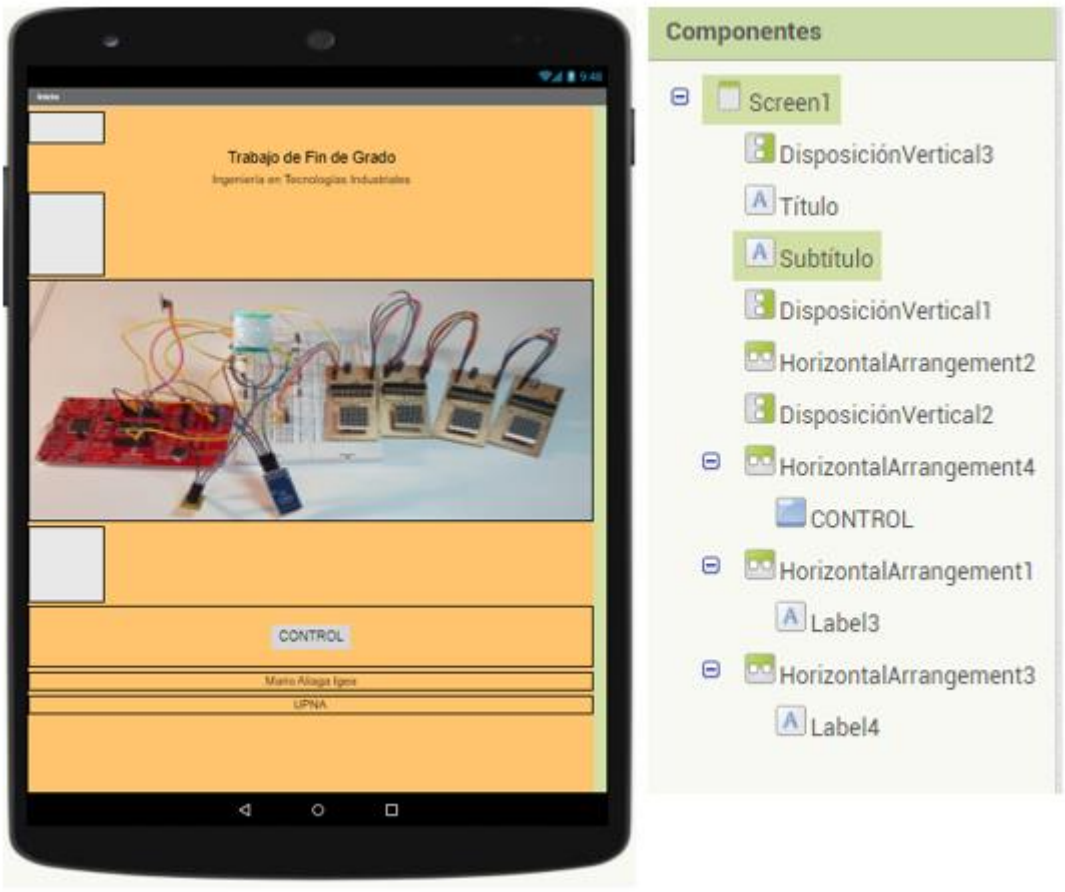


Ilustración 126: Diseño pantalla Inicio

Cuando acabemos el diseño de la primera pantalla pasaremos a la pestaña bloques para crear el código de esta pantalla, como solo tenemos un botón cuya función es pasar a la siguente pantalla la programación es muy sencilla (Ilustración 127).

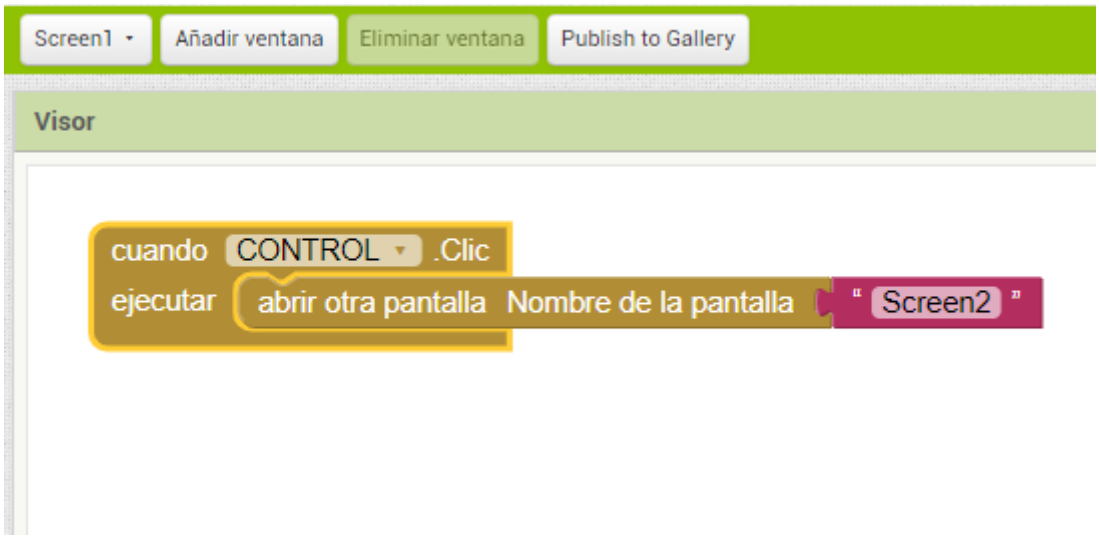


Ilustración 127: Programación Pantalla 1

A través de la segunda pantalla nos comunicaremos con el módulo *bluetooth*, lo primero que debemos hacer es añadir un selector de lista al que le asignaremos el nombre de "CONECTAR".

Para tener una mejor organización utilizaremos disposiciones verticales para agrupar los diferentes elementos que tienen que ver con el mismo sensor.

Dentro de las disposiciones verticales de luminosidad y presencia añadiremos un *switch* y una disposición horizontal con dos botones en su interior.

En la disposición vertical del control de temperatura, añadiremos un *switch* y dos disposiciones horizontales. En la primera insertaremos dos etiquetas en las que escribiremos "TMIN" y "TMAX", dos campos de texto y una disposición horizontal para tener separación entre los dos. En la segunda añadiremos dos deslizadores y en sus propiedades cambiaremos el color de la barra a azul para la TMIN, rojo para TMAX y los valores que pueden tomar los deslizadores están entre 15 y 35.

Para poder mandar datos a través de *bluetooth* es necesario añadir desde la pestaña "Conectividad" de la paleta "ClienteBluetooth".

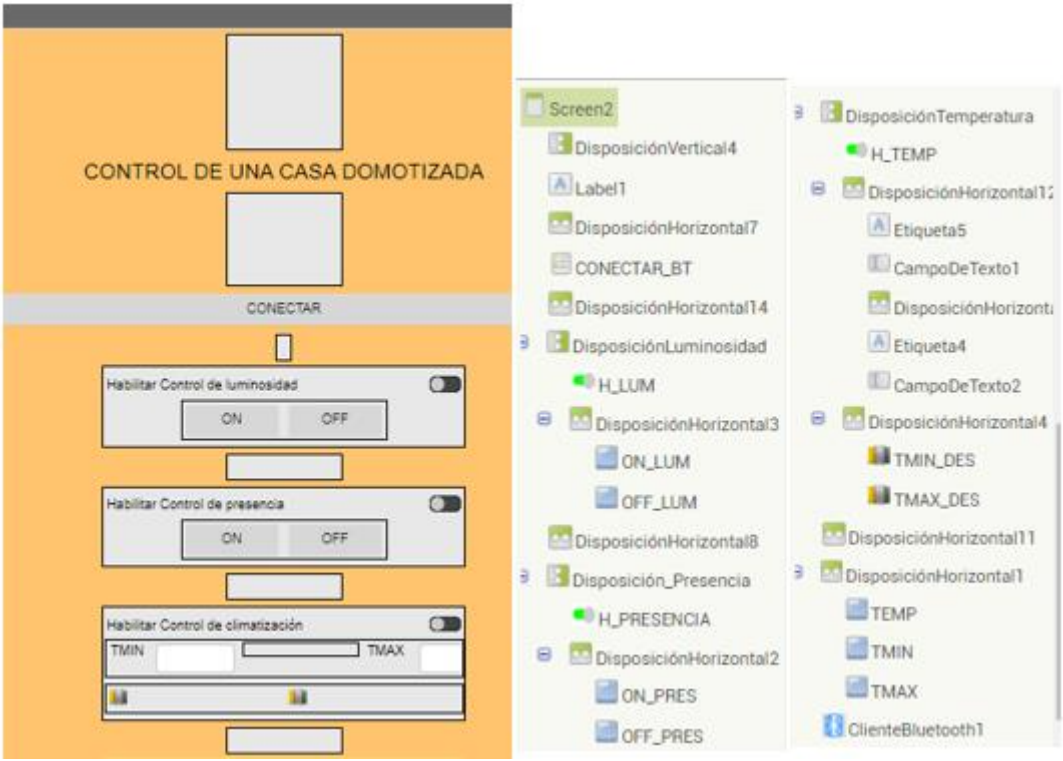


Ilustración 128: Diseño pantalla Control

Ahora procederemos a realizar la programación con bloques de la pantalla control.

Para conectar la aplicación con el módulo *bluetooth* HC-06 son necesarios los bloques de la Ilustración 129.

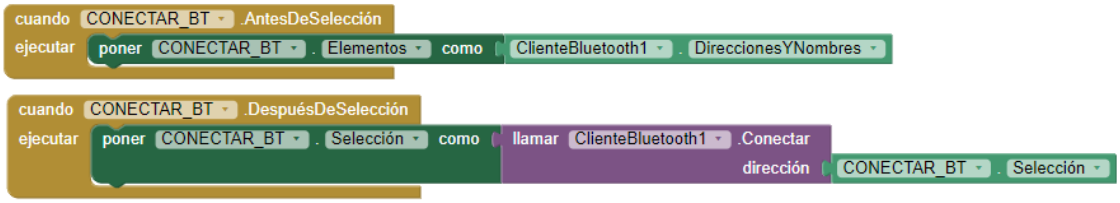


Ilustración 129: Bloques conexión Bluetooth

Para que el microcontrolador pueda diferenciar el botón que se ha pulsado asignaremos un número a cada uno y ese número será el que se envíe a través de la conexión bluetooth.

Ilustración 130: Bloques comunicación Bluetooth

Para facilitar el envío de datos se va a enviar el número redondeado de la posición de los deslizadores, además en el campo de texto aparecerá este valor para que el usuario pueda saber la temperatura que ha seleccionado. Para que el microcontrolador pueda diferenciar ambas temperaturas añadiremos 50 a TMAX y luego en el microcontrolador deberemos restarlos para tener la temperatura máxima correcta.

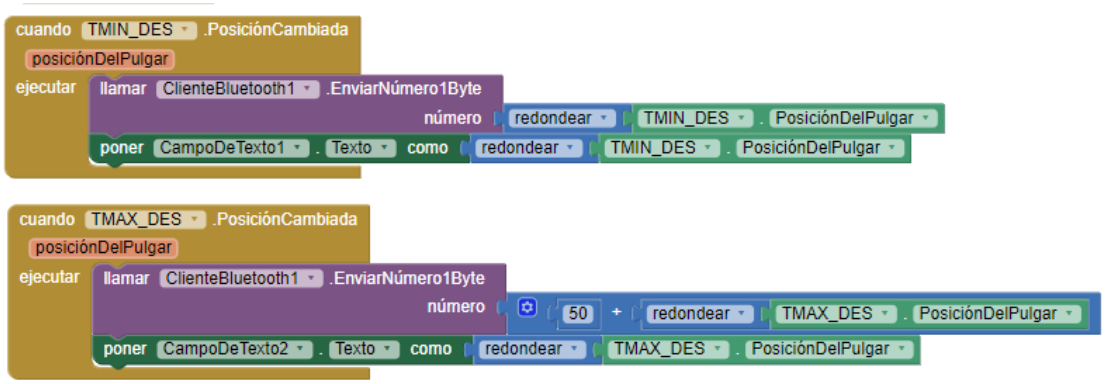


Ilustración 131: Bloques selección de temperatura

Cuando hallamos finalizado la programación de bloques ya tendremos acabada la aplicación móvil, para poder descargarla en el móvil iremos a la pestaña “Generar” del menú superior y seleccionaremos la opción “APP (generar código QR para el archivo .apk)” y a través de este código encontraremos el enlace para descargar la aplicación.

Ahora procederemos a escribir el código del programa que permita interactuar con la aplicación. Al igual que hicimos en el ejemplo del apartado 4.10 utilizaremos la función “*switch”* para ejecutar una función u otra dependiendo del número.

```
67  void bluetooth(void)
68  {
69  if(eusart1RxCount!=0)
70  {
71  recibido=EUSART1_Read();  // read
72
73  switch(recibido)     // check comm
74  {case 1:     //Switch Luminosidad
75      {
76          if(LUM_HAB==0){
77              LUM_HAB=1;
78              LUMHAB_SetHigh();
79          }
80          else{
81              LUM_HAB=0;
82              LUMHAB_SetLow();
83          }
84          break;
85      }
86   case 2: //ON luminosidad
87      {
88          if(LUM_HAB==0)
89              LUM_MAN_SetHigh();
90          break;
91      }
92   case 3:     //OFF luminosidad
93      {
94          if(LUM_HAB==0)
95              LUM_MAN_SetLow();
96          break;
97      }
98   case 4:     //Switch Presencia
99      {
100         if(PRES_HAB==0)
101            PRES_HAB=1;
102         else
103             PRES_HAB=0;
104         LED3_SetLow();
105         break;
106     }
107  case 5:     //ON presencia
108     {
109         if(PRES_HAB==0)
110              LED3_SetHigh();
111         break;
112     }
113  case 6: //OFF presencia
114     {
115         if(PRES_HAB==0)
116               LED3_SetLow();
117         break;
118     }
```

```
121         }
122     case 7: //Switch Temperatura
123         {
124             if(TEMP_HAB==0)
125                 TEMP_HAB=1;
126             else{
127                 TEMP_HAB=0;
128                 LED1_SetLow();
129             }
130             break;
131         }
132     default:
133        {
134          if(recibido<40){
135             TMIN=recibido;
136          }
137          if (recibido>60){
138             TMAX=recibido-50;
139          }
140          break;
141        }
142     }
143     }
144     }
```

5.2. Proyecto 2

El objetivo del segundo proyecto es mostrar a través de un panel de leds el valor de una variable obtenida por un sensor y a través del pulsador conectado a RC4 se podrá cambiar el modo en el que se representa la información. En este caso utilizaremos el sensor de temperatura LM35DZ utilizado en el proyecto anterior.

5.2.1. Panel de leds

El panel de leds estará compuesto por 6 matrices como la que se utilizó en el apartado 4.11. Para facilitar las conexiones entre la placa de desarrollo, el *driver* MAX7219 y la matriz de leds se diseñará una PCB a través del programa DesingSpark.

El driver MAX7219 y la matriz de leds no se encuentran disponibles en las bibliotecas del programa, por lo que el MAX7219 lo obtendremos de la web de búsqueda de componentes http://rs.componentsearchengine.com/, la matriz de leds no está disponible en esta web, su símbolo tendremos que crearlo nosotros mismos.

La primera parte del circuito esquemático es la que aparece en la Ilustración 132 y está compuesta por un conector de cinco pines que son los encargados de conectar la PCB con el microcontrolador y la alimentación. Entre los terminales de alimentación VCC y GND se encuentra el condensador de Bulk, este condensador debe tener como mínimo una capacidad 20 veces superior a la suma de todos los condensadores de desacoplo de la placa.

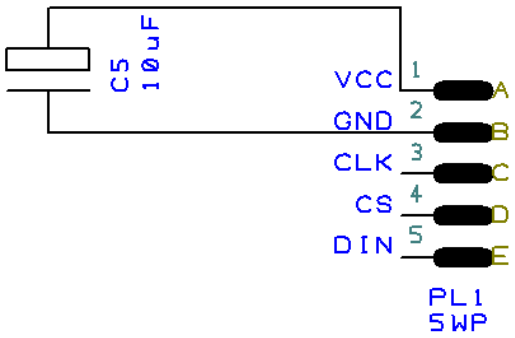


Ilustración 132: Esquemático parte 1

La segunda parte del circuito esquemático es el driver MAX7219 y sus conexiones. A través del pin DOUT está conectado a un pin del conector de 5 para enviar los datos a la siguiente matriz conectada en serie, el resto de los pines del conector sirven para tener menos cables que conectar a la siguiente matriz. Entre el pin V+ y tierra se encuentra conectado un condensador de desacoplo de 0.1uF para que la tensión que llega al driver tenga un valor constante.

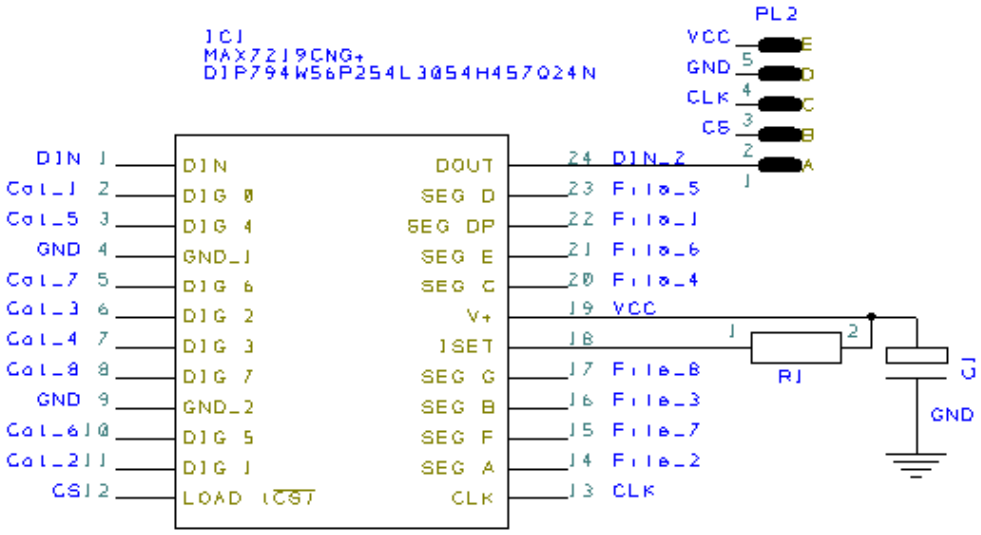


Ilustración 133: Esquemático parte 2

La última parte del esquemático consiste en las conexiones del *driver* MAX7219 con la matriz de leds.

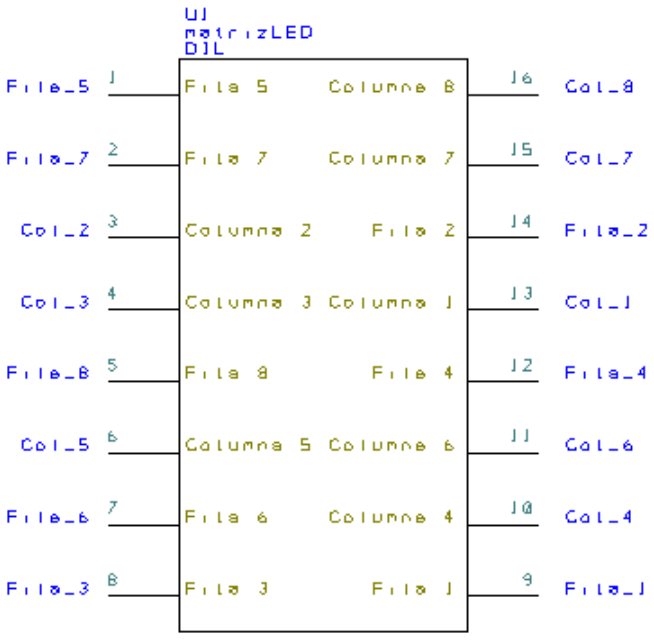


Ilustración 134: Esquemático parte 3

Para crear la huella de la matriz deberemos mirar el *datasheet* de la matriz, en la Ilustración 135 están las dimensiones del componente.

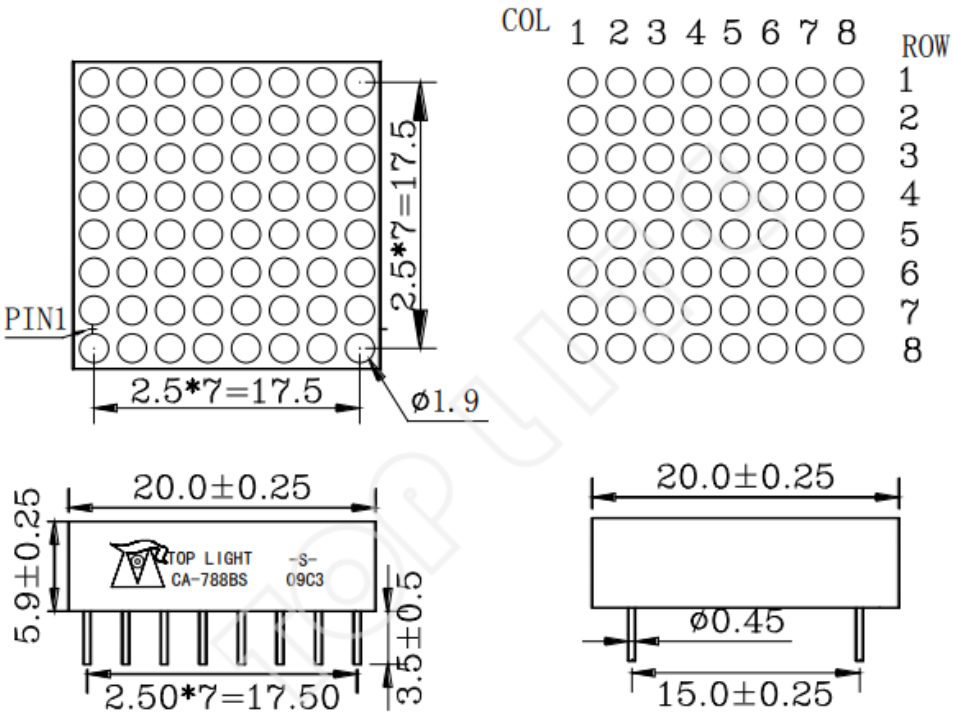


Ilustración 135: Dimensiones de la matriz de leds [28]

El diseño de la PCB tiene que cumplir las especificaciones del fabricante, en este caso se van a fabricar en la UPNA, cuyas especificaciones aparecen en la Ilustración 136. Además, la tecnología con la que se cuenta en la UPNA no es capaz de metalizar los *pads* ni las vías, por lo que las conexiones con los componentes DIL tienen que ser por la capa en la que están soldados.

	UPNA		
	mm.	mils	
Ancho de pistas	0,3	12	
Separación entre pistas o pads	0,3	12	
Diámetro mínimo de taladro	0,6	24	
Pared mínima de la corona	0,25	10	
Separación mínima entre pistas y canto (para el fresado)	0,25	10	

Ilustración 136: Especificaciones UPNA [30]

En la Ilustración 138 y en la Ilustración 137 está representado el diseño final de la PCB por ambas caras y en la Ilustración 139.

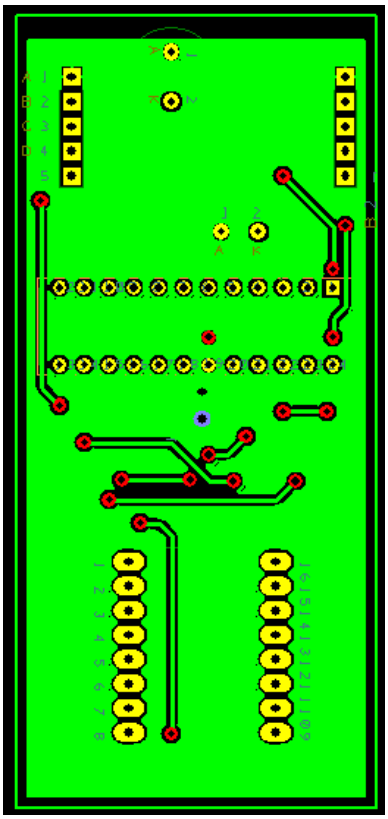


Ilustración 138: Capa superior de la PCB

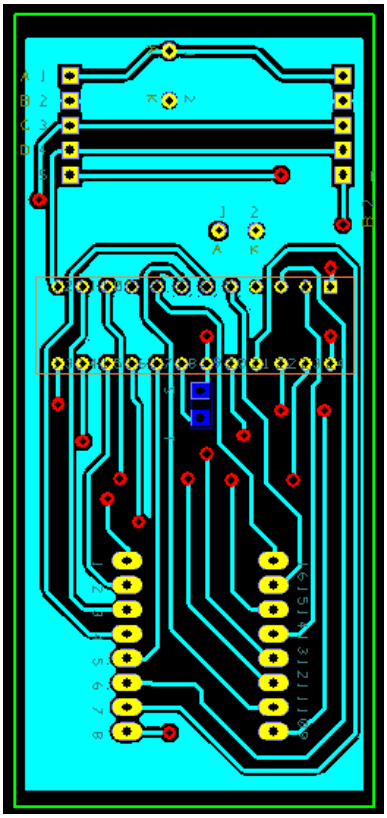


Ilustración 137: Capa inferior de la PCB

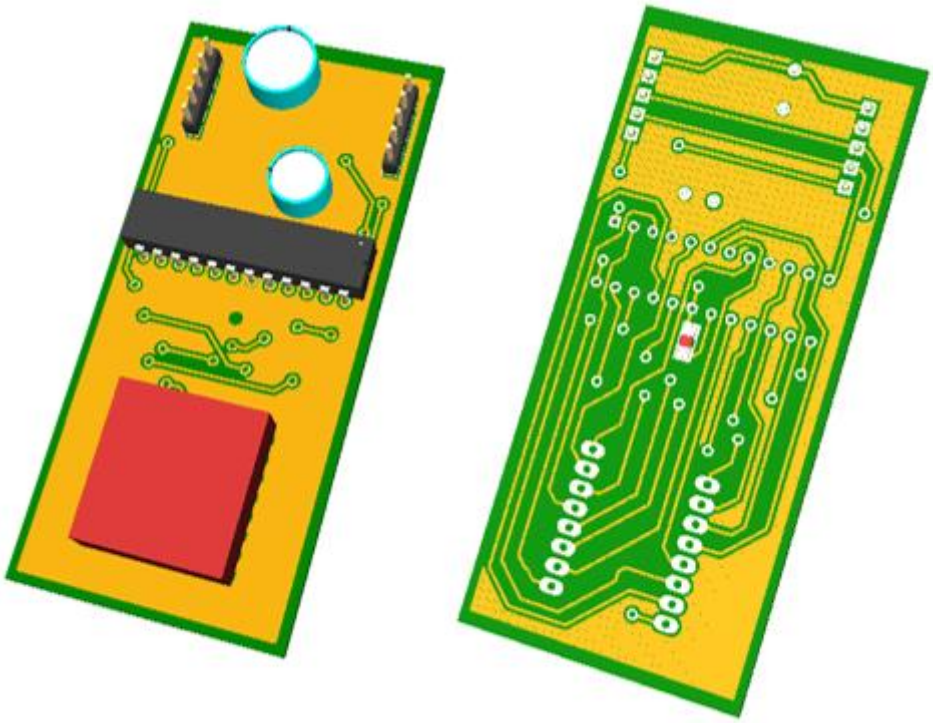
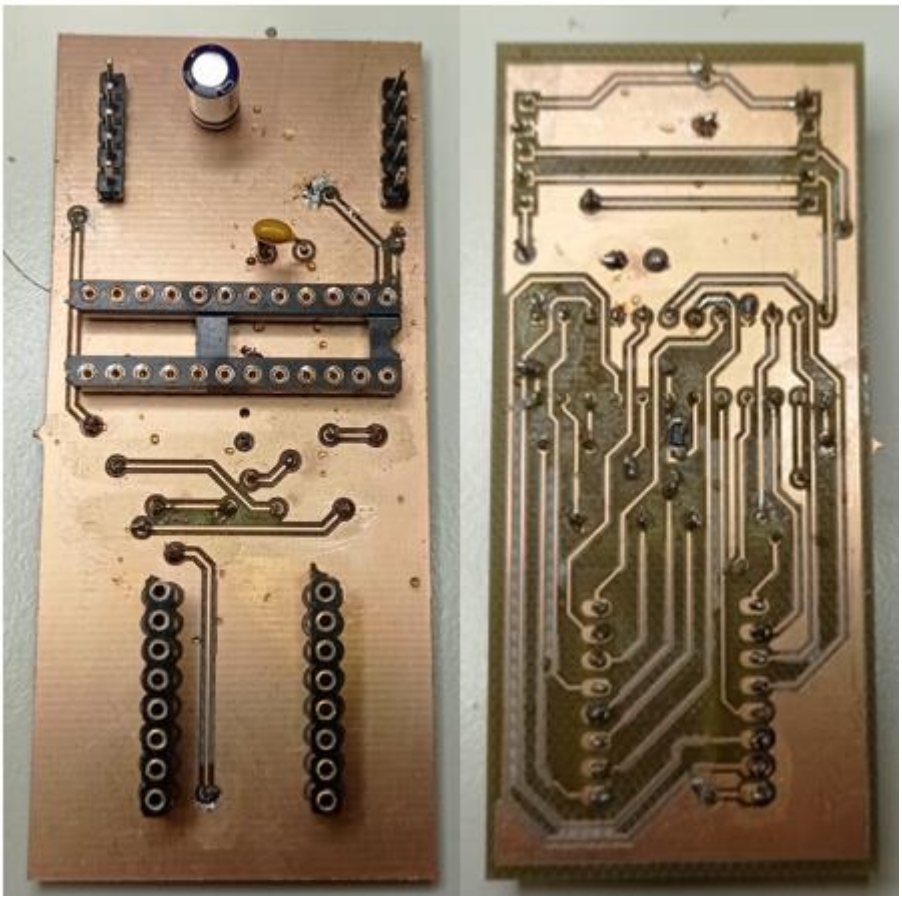


Ilustración 139: Vista 3D de la PCB

Una vez fabricada la PCB hay que soldar todos los componentes del circuito, en el caso de la matriz utilizaremos conectores hembra para poder cambiarla en caso de avería y en el driver usaremos un zócalo para evitar dañarlo al soldar. En la se puede observar la PCB ya soldada.



5.2.2. Programación

Comenzaremos creando un nuevo proyecto en MPLAB X IDE y abriremos el MCC para configurar las propiedades del sistema. En este caso utilizaremos una señal de reloj de 1 MHz (Ilustración 140).

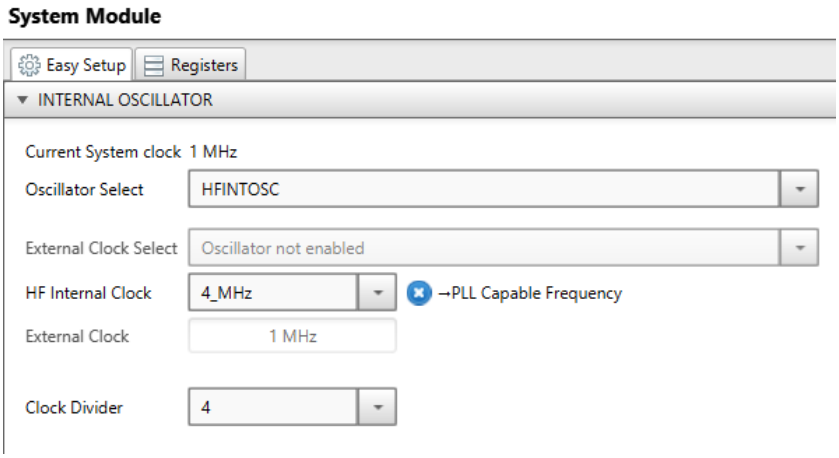


Ilustración 140: Configuración del sistema

A los recursos del sistema añadiremos el módulo MSSP1 para la comunicación con el driver MAX7219 y el CAD para obtener el valor de la señal analógica del sensor de temperatura.

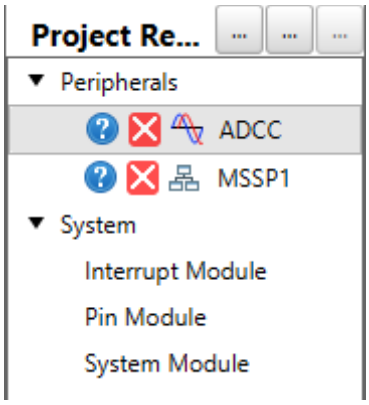


Ilustración 141: Recursos del proyecto

El módulo MSSP1 lo configuraremos para que utilice el protocolo serie SPI y una frecuencia de reloj de 250.000 Hz (Ilustración 142).

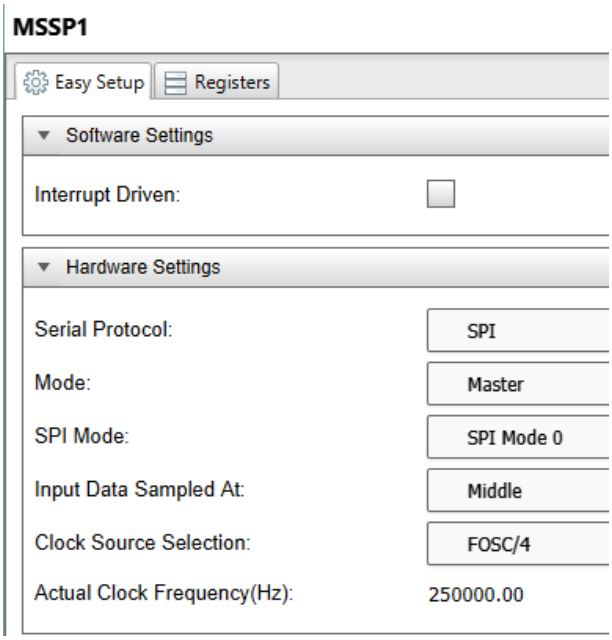


Ilustración 142: Configuración módulo MSSP1

El módulo ADCC lo configuraremos para que utilice el reloj interno (Frc), esté alineado a la derecha y utilice una señal de referencia de 5V como nivel alto y 0V como nivel bajo (Ilustración 143).

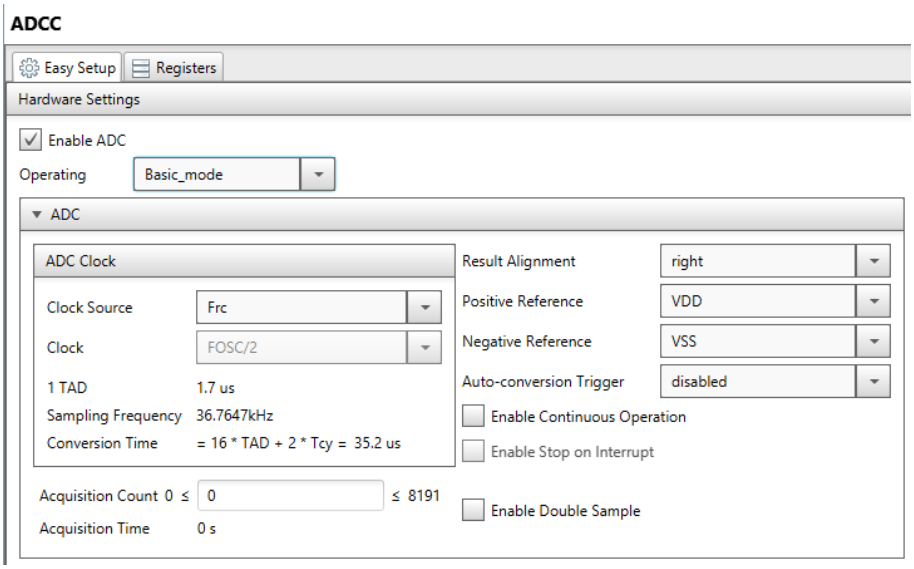


Ilustración 143: Configuración CAD

En el *pin Manager* asignaremos los pines SCK a RB6, SDO1 a RB5, SDI1 a RB4 que son los que utiliza el módulo MSSP1, el CAD al pin RA0, el pulsador a RC4 con las resistencias de *pull-up* habilitadas y interrupción por cambio de valor negativo y por último habilitaremos seis salidas digitales en los pines RC0, RC1, RC2, RC3, RC5 y RC6, estos últimos se utilizan para seleccionar el driver al que se quiere enviar los datos (Ilustración 144).

			Port A...		Port B ▼			Port C ▼						
Module	Function	Direction	0	3	4	5	6	0	1	2	3	4	6	7
ADCC	ANx	input	🔒											
MSSP1 ▼	SCK1	in/out					🔒							
	SDI1	input			🔒									
	SDO1	output				🔒								
Pin Module ▼	GPIO	input										🔒		
	GPIO	output						🔒	🔒	🔒	🔒	🔒	🔒	🔒
RESET	MCLR	input		🔒										

Pin Module

Easy Setup | Registers

Selected Package : QFN20

Pin Name	Module	Function	Custom Name	Start High	Analog	Output	WPU	OD	IOC
RA0	ADCC	ANA0	TEMP	☐	☑	☐	☐	☐	none
RB4	MSSP1	SDI1		☐	☐	☐	☐	☐	none
RB5	MSSP1	SDO1		☐	☐	☑	☐	☐	none
RB6	MSSP1	SCK1		☐	☐	☑	☐	☐	none
RC0	Pin Module	GPIO	LOAD0	☐	☐	☑	☐	☐	none
RC1	Pin Module	GPIO	LOAD1	☐	☐	☑	☐	☐	none
RC2	Pin Module	GPIO	LOAD2	☐	☐	☑	☐	☐	none
RC3	Pin Module	GPIO	LOAD3	☐	☐	☑	☐	☐	none
RC4	Pin Module	GPIO	PULSADOR	☐	☐	☐	☑	☐	negative
RC6	Pin Module	GPIO	LOAD4	☐	☐	☑	☐	☐	none
RC7	Pin Module	GPIO	LOAD5	☐	☐	☑	☐	☐	none

Ilustración 144: Pin Manager proyecto 2

Con el *Pin Manager* ya configurado ya podemos pasar a escribir las líneas del programa tras hacer clic en el botón *Generate*.

Como en este programa el módulo MSSP va a controlar seis esclavos iguales, vamos a crear seis funciones de cada, la diferencia que tienen es que la función “enviarbits” selecciona un driver diferente a través de las salidas digitales “LOAD”.

```
void enviarbits0(uint8_t direccion, uint8_t data){
SPI1_Open(SPI1_DEFAULT);
        LOAD0_SetLow();
        SPI1_ExchangeByte(direccion);
        SPI1_ExchangeByte(data);
        LOAD0_SetHigh();
        SPI1_Close();
}
     void limpiar0() {
         for (uint8_t g = 1; g <= 8; g++) {
             enviarbits0(g,0);
         }
     }
     void configuracion0(){
         enviarbits0(0b00001100,1);
         enviarbits0(0b00001001,0);
         enviarbits0(0b00001010,1);
         enviarbits0(0b00001011,0b00001111);
     }
 void enviarbits1(uint8_t direccion, uint8_t data){
     SPI1_Open(SPI1_DEFAULT);
     LOAD1_SetLow();
     SPI1_ExchangeByte(direccion);
     SPI1_ExchangeByte(data);
     LOAD1_SetHigh();
     SPI1_Close();
 }
     void limpiar1() {
         for (uint8_t g = 1; g <= 8; g++) {
             enviarbits1(g,0);
         }
     }
     void configuracion1(){
         enviarbits1(0b00001100,1);
         enviarbits1(0b00001001,0);
         enviarbits1(0b00001010,1);
         enviarbits1(0b00001011,0b00001111);
     }
```

Ilustración 145: Funciones de configuración para los drivers

```
uint8_t z,g,t;
void enviarbits0(uint8_t direccion, uint8_t data){...8 lines }
void enviarbits1(uint8_t direccion, uint8_t data){...8 lines }
void enviarbits2(uint8_t direccion, uint8_t data){...8 lines }
void enviarbits3(uint8_t direccion, uint8_t data){...8 lines }
void enviarbits4(uint8_t direccion, uint8_t data){...8 lines }
void enviarbits5(uint8_t direccion, uint8_t data){...8 lines }
void limpiar0(){...5 lines }
void limpiar1(){...5 lines }
void limpiar2(){...5 lines }
void limpiar3(){...5 lines }
void limpiar4(){...5 lines }
void limpiar5(){...5 lines }
void configuracion0(){...6 lines }
void configuracion1(){...6 lines }
void configuracion2(){...6 lines }
void configuracion3(){...6 lines }
void configuracion4(){...6 lines }
void configuracion5(){...6 lines }
```

Ilustración 146: Funciones de configuración de los drivers 2

Estas funciones las agruparemos en otras dos para evitar tener que llamar a todas las funciones.

```
void limpiartodo(){
    limpiar0();
    limpiar1();
    limpiar2();
    limpiar3();
    limpiar4();
    limpiar5();
}
void conftodo(){
    configuracion0();
    configuracion1();
    configuracion2();
    configuracion3();
    configuracion4();
    configuracion5();
}
```

Ilustración 147: Funciones limpiar y configurar

En un vector de números de 8 bits llamado “números” almacenaremos todos los números del 0 a 9 (). Con la ayuda de la siguiente página web podemos saber cómo representar cada número. https://giltesa.com/2011/10/06/conversor-de-graficos-de-8x8-pixeles-a-binario-para-matrices-de-leds

```
uint8_t numeros[88]={
    //0
    0,0,126,66,66,66,126,0,
    //1
    0,0,8,68,126,64,0,0,
    //2
    0,0,100,98,82,82,12,0,
    //3
    0,0,68,146,146,146,108,0,
    //4
    0,0,30,16,16,16,124,0,
    //5
    0,0,142,146,146,146,98,0,
    //6
    0,0,126,146,146,146,100,0,
    //7
    0,0,2,2,114,10,6,0,
    //8
    0,0,254,146,146,146,254,0,
    //9
    0,0,76,146,146,146,252,0
};
```

Ahora crearemos dos vectores con el mensaje que queremos mostrar en cada uno de los modos. En el primer modo el contenido de desplaza de derecha a izquierda permitiendo mostrar un mensaje de mayor tamaño y en el segundo modo el mensaje se mantiene fijado todo el rato.

El mensaje del primer modo será el siguiente: "TEMP AMB XX", las dos "X" representan el valor de la temperatura en ese instante. Este mensaje lo almacenaremos en un vector como aparece en la Ilustración 148.

```
uint8_t letrasTEMP_AMB[58]={
    //T
    0,2,2,126,2,2,0,0,
    //E
    0,254,146,146,146,130,0,0,
    //M
    126,2,4,8,4,2,126,0,
    //P
    0,254,18,18,18,30,0,0,
    //espacio
    0,0,
    //A
    0,126,18,18,18,18,126,0,
    //M
    126,2,4,8,4,2,126,0,
    //B
    0,0,254,146,146,146,108,0
};
```

Ilustración 148: Mensaje primer modo

El mensaje del segundo modo será el siguiente: "TAMB XX". Al igual que el mensaje del modo 1 también lo guardaremos en un vector, aunque en este caso se guardará en orden inverso porque haremos el barrido de columnas de derecha a izquierda.

```
uint8_t letrasTAMB[32]={
    //B
    0,0,108,146,146,146,254,0,
    //M
    126,2,4,8,4,2,126,0,
    //A
    0,126,18,18,18,18,126,0,
    //T
    0,2,2,126,2,2,0,0
};
```

Ilustración 149: Mensaje segundo modo

En la Ilustración 150 se representa una parte del código del modo 1, la primera sentencia "*for*" se repite tantas veces como columnas tenga el mensaje completo que vamos a mostrar y el segundo bucle "for" tantas veces como columnas tengamos, es decir, es este caso tenemos 6 matrices con 8 columnas cada una, por lo que el total de columnas es 6*8=48. Como utilizamos seis matrices utilizaremos un "if" para separar de las columnas que tienen que ir a cada matriz.

El siguiente *if* "if(t<58)" nos sirve para separar el texto del mensaje de la parte numérica, este valor tiene que ser igual a la longitud del texto, como podemos comprobar en la Ilustración 148 el mensaje del primer modo está compuesto por 58 dígitos.

El siguiente *if* "if(t>=z)" evita que envía datos del vector que ocupan posiciones negativas, en el caso de que t sea menor que z se enviará un 0.

Lo mismo sucede en la parte donde se envían los dígitos, con el *if* "if(t>=z+58)" se evitan enviar posiciones negativas y con el *if* que se encuentra en su interior dividimos las 18 columnas destinadas a los números en decenas y centenas.

```
void modo1(void){
   for (t=1;t<=76;t++){
            for(z=1;z<=48;z++){
                if(z<=8){//matriz 0
                    if(t<=58){
                                    if(t>=z)
                                        enviarbits0(z,letrasTEMP_AMB[t-z]);
                                    else
                                        enviarbits0(z,0);
                                }
                            else{
                                if(t>=z+58){
                                    if(t<=66)
                                    enviarbits0(z,numeros[decena*8+t-z-58]);
                                    else
                                    enviarbits0(z,numeros[unidad*8+t-z-58]);
                                }
                                else
                                enviarbits0(z,0);
                            }
                }
```

Ilustración 150: Código modo 1

En la __ podemos ver como es el código para el resto de las matrices, este código es muy similar al anterior y lo único en lo que se diferencia es que hemos cambiado la variable "z" por la variable "a" para que el valor de la columna esté entre los valores 1 y 8. El código del resto de matrices es igual a esta cambiando la variable "z" por otra siguiendo la formula a=z – 8 * (nº de columna).

```
else if(z<=16){//matriz 1
    a=z-8;
    if(t<=58){
                if(t>=a)
                    enviarbits1(a,letrasTEMP_AMB[t-a]);
                else
                    enviarbits1(a,0);
            }
        else{
            if(t>=a+58){
                if(t<=66)
                enviarbits1(a,numeros[decena*8+t-a-58]);
                else
                enviarbits1(a,numeros[unidad*8+t-a-58]);
            }
            else
            enviarbits1(a,0);
        }
}
```

Ilustración 151: Código modo 1 matriz 1

El código del segundo modo es más sencillo ya que en este caso el mensaje del panel siempre va a estar en el mismo lugar.

```
void modo2(){
    for (t=1;t<=48;t++){
        if(t<=8) //numeros
            enviarbits0(t,numeros[unidad*8+t-1]);
        else if(t<=16)
            enviarbits1(t,numeros[decena*8+t-1]);
        else if(t<=24)//letras
            enviarbits2(t,letrasTAMB[t-17]);
        else if(t<=32)//letras
            enviarbits3(t,letrasTAMB[t-17]);
        else if(t<=40)//letras
            enviarbits4(t,letrasTAMB[t-17]);
        else
            enviarbits5(t,letrasTAMB[t-17]);
    }
}
```

Ilustración 152: Código modo 2

Además, crearemos una función temperatura que será la encargada de obtener el valor de la temperatura, este valor lo almacenaremos en dos variables llamadas “decena” y “unidad”.

```
uint16_t resultado;
float resultado2;
int decena,unidad;
void temperatura(void){
    resultado = ADCC_GetSingleConversion(TEMP);
    resultado2 = (resultado*(5000/4096)*0.1)+5;
    decena=resultado2/10;
    unidad=resultado%10;
}
```

Ilustración 153: Código función temperatura

La función contador será a la que llamemos cuando se produce una interrupción en el pin RC4 y nos permite cambiar el modo seleccionado.

```
int modo=1;
void contador(){
    if (modo==1)
        modo=2;
    else
        modo=1;
}
```

Ilustración 154: Código función contador

En el código principal del programa solo hay que llamar a las funciones de configuración y limpiar, habilitar las interrupciones globales y periféricas y declarar la función “contador” como la de atención a la interrupción. En el bucle *while* llamaremos a la función “temperatura” para actualizar el valor del sensor y comprobaremos si se ha cambiado el modo a través del pulsador.

```
void main(void)
{
    // initialize the device
    SYSTEM_Initialize();
    conftodo();
    limpiartodo();
    // Enable the Global Interrupts
    INTERRUPT_GlobalInterruptEnable();
    // Enable the Peripheral Interrupts
    INTERRUPT_PeripheralInterruptEnable();
    IOCCF4_SetInterruptHandler(contador);
    while (1)
    {
        temperatura();
        if(modo==1)
            modo1();
        else
            modo2();
    }
}
```

Ilustración 155: Código principal del proyecto 2

6. BIBLIOGRAFÍA

[1] Electrónica Estudio "¿Qué es un microcontrolador?". URL: https://www.estudioelectronica.com/que-es-un-microcontrolador/

[2] Electrónica Digital "Introducción a los microcontroladores". URL: https://sites.google.com/site/electronicadigitalml/home/5-introduccion-a-los-micro-controladores

[3] Todo sobre microcontroladores "Historia de los microcontroladores". URL: https://sites.google.com/site/21511090proyecto/home/historia-de-los-microcontroladores

[4] *Computer History* "TMS 1000 4-Bit microcontroller, TI, 1976". URL: https://www.computerhistory.org/revolution/digital-logic/12/284/1564

[5] Microcontroladores "Empresas fabricantes de microcontroladores". URL: https://microcontroladoressesv.wordpress.com/empresas-fabricantes-de-microcontroladores/

[6] Hard Zone "Arquitectura Von Neumann". URL: https://hardzone.es/tutoriales/rendimiento/von-neumann-limitaciones/

[7] Wikipedia "Arquitectura de Von Neumann". URL: https://es.wikipedia.org/wiki/Arquitectura_de_Von_Neumann

[8] Compilando conocimiento "Arquitectura Von-Neumann vs Harvard". URL: https://compilandoconocimiento.com/2017/01/29/arquitecturasvon-newmanvsharvard/

[9] Electrotools "Diferencias entre el modelo de Von Neumann y Harvard". URL: https://www.electrontools.com/Home/WP/diferencias-entre-los-modelos-de-von-neumann-y-harvard/

[10] ECURED "Microcontroladores PIC". URL: https://www.ecured.cu/Microcontroladores_PIC.

[11] Trece dBs "PICs: los microcontroladores de Microchip". URL: https://trecedb.wordpress.com/2009/03/21/pics-los-microcontroladores-de-microchip/

[12] Microcontroladores PIC. URL: http://bibing.us.es/proyectos/abreproy/11301/fichero/Memoria%252FCap%C3%ADtulo+3.pdf+

[13] PIC 16 bits. URL: http://ele-mariamoliner.dyndns.org/~fperal/cursos-anteriores/10-11/proy/pic16bits.pdf

[14] Microchip "32-bit PIC® and SAM Microcontrollers". URL: https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors/32-bit-mcus

[15] Teoría de Sistemas Digitales UPNA

[16] Microchip "*CURIOSITY HIGH PIN COUNT (HPC) DEVELOPMENT BOARD*". URL: https://www.microchip.com/en-us/development-tool/DM164136#additional-summary

[17] Microchip "*CURIOSITY DEVELOPMENT BOARD*". URL: https://www.microchip.com/en-us/development-tool/DM164137#additional-summary

[18] Microchip “PIC18F47Q10 CURIOSITY NANO”. URL:
https://www.microchip.com/en-us/development-tool/DM182029

[19] Microchip “EXPLORER 8 DEVELOPMENT KIT”. URL:
https://www.microchip.com/en-us/development-tool/DM160228#additional-summary

[20] Microchip. URL:
https://www.microchip.com/

[21] Microchip “Datasheet PIC16F18446”. URL:
https://www.microchip.com/en-us/product/PIC16F18446

[22] Luis Llamas “MEDIR DISTANCIA CON ARDUINO Y SENSOR DE ULTRASONIDOS HC-SR04”. URL:
https://www.luisllamas.es/medir-distancia-con-arduino-y-sensor-de-ultrasonidos-hc-sr04/

[23] Datasheet “Sensor de distancia HC-SR04”. URL:
https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf

[24] Datasheet “Sensor de temperatura LM35”. URL:
https://www.ti.com/lit/ds/symlink/lm35.pdf

[25] Datasheet “Sensor de luminosidad”. URL:
https://docs.rs-online.com/7251/0900766b8156674e.pdf

[26] Datasheet “Sensor de presencia HC-SR501”.URL:
https://puntoflotante.net/MANUAL-DEL-USUARIO-SENSOR-DE-MOVIMIENTO-PIR-HC-SR501.pdf

[27] Naylamp Mechatronics “tutorial básico de uso del módulo bluetooth hc-06”. URL: https://naylampmechatronics.com/blog/12_tutorial-basico-de-uso-del-modulo-bluetooth-hc-06-y-hc-05.html

[28] Datasheet “Matriz de leds 788BS”. URL:
http://www.ledtoplite.com/uploadfile/2017/TOP-CA-788BS.pdf

[29] Datasheet “Driver MAX7219”. URL:
http://images.100y.com.tw/pdf_file/35-MAXIM-MAX7219.pdf

[30] Upna *technology*. URL:
https://miaulario.unavarra.es/access/content/group/2020_0_242814_1_G/Proyecto%202015%20-%2016/UPNA_technology.pdf