Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERÍAS INDUSTRIALES

Grado en Ingeniería en Electrónica Industrial y Automática

# Desarrollo de guiones de prácticas para un Laboratorio Colaborativo de Tecnología Electrónica basado en la placa Basys MX3


Autor:

Pardo Grajal, Javier

Tutor:

de Pablo Gómez, Santiago

Departamento de Tecnología Electrónica


Valladolid, Julio 2019.

## TÍTULO

Desarrollo de guiones de prácticas para un Laboratorio Colaborativo de Tecnología Electrónica basado en la placa Basys MX3.

## RESUMEN

Este proyecto se centra en la realización de una serie de guiones de prácticas para la placa de desarrollo Basys MX3, basada en un procesador PIC de 32 bits que incorpora múltiples periféricos.

Estos guiones servirán para introducir a otros alumnos en el aprendizaje de esta tecnología. Los guiones tendrán una descripción de los periféricos, las instrucciones necesarias para su programación y varios ejemplos para comprobar su funcionamiento.

Los guiones se depositarán en un Laboratorio Colaborativo de Tecnología Electrónica, para que el usuario que lo desee pueda disponer de ellos y le sirva de apoyo para su aprendizaje.

## PALABRAS CLAVE

- Laboratorio colaborativo
- Basys MX3
- Microcontrolador PIC32
- MPLAB X IDE

## TITLE

Development of practice scripts for an Electronic Technology Collaborative Laboratory based on the Basys MX3 board.

## ABSTRACT

This Project focuses on the realization of a series of practices scripts for the development board Basys MX3, based on the processor PIC 32 bits, that includes several peripheral appliances.

This practices scripts will serve to introduces to others students into the learning of this technology. All scripts will have a description of the peripheral appliance, the instructions that are required for their programing and a few examples to debug their working.

This practices scripts will be in a Collaborative Laboratory of Electronics Technology, so that the user who wants it can have them and support it for their learning.

## KEYWORDS

- Collaborative Laboratory
- Basys MX3
- Microcontroller PIC32
- MPLAB X IDE

# ÍNDICE

# ÍNDICE DE FIGURAS

# ÍNDICE DE TABLAS

# CAPÍTULO 1:
# INTRODUCCIÓN Y OBJETIVOS

## 1.1 INTRODUCCIÓN

A través de este trabajo se pretende redactar los primeros guiones de prácticas para la tarjeta de desarrollo Basys MX3, basada en un procesador PIC de 32 bits, que dispone de múltiples periféricos. Se ensayará el comportamiento de dichos periféricos y se propondrán varios guiones, que progresivamente introduzcan a otros alumnos en el estudio de estos componentes. Este trabajo será una aportación importante para un Laboratorio Colaborativo que se está implantando en el departamento de Tecnología Electrónica.

Un Laboratorio Colaborativo de Tecnología Electrónica es un laboratorio en el que un grupo de usuarios crean unos guiones de prácticas, basándose en su propio aprendizaje o en el de otros usuarios. Se explicará cómo usar un determinado equipo electrónico, de este modo los futuros usuarios dispondrán del trabajo de los anteriores para poder aprender a utilizar dicho equipo, que a su vez podrán realizar otras prácticas más complejas basándose en lo que otros usuarios les han enseñado con sus guiones.

Una de las finalidades es que dicho laboratorio vaya creciendo progresivamente tanto en tamaño como en dificultad, ya que lo que al primer usuario le llevo una hora aprenderlo, al siguiente le llevará diez minutos y podrá progresar en otras cuestiones. El objetivo de estos guiones es servir de apoyo para el aprendizaje, no implica que sea un documento cerrado, ya que puede sufrir modificaciones a cargo del resto de usuarios. Por este motivo, se incluye un apartado en cada práctica llamado 'Control de versiones' en el que se indicará cuándo se ha modificado un guion, junto con una breve descripción de la modificación realizada.

La placa con la que empezaremos el laboratorio colaborativo será la Basys MX3, una placa de aprendizaje de la marca Digilent, diseñada específicamente para impartir cursos introductorios a sistemas integrados. Con esta placa el usuario puede crear un gran número de aplicaciones de una manera sencilla y rápida, gracias a que dispone de una amplia gama de periféricos integrados, tales como: interruptores, botones, LEDs, etc. Además, la placa Basys MX3 tiene la posibilidad de incorporar otros periféricos a mayores de una forma simple, lo que nos permitirá aumentar el número de posibilidades a la hora de realizar diferentes ensayos.

La placa Basys MX3 incorpora un microcontrolador PIC32 de la marca Microchip. Para poder empezar a programar el microcontrolador PIC32 necesitaremos instalar, en nuestro ordenador, el software MPLAB X IDE y el compilador MPLAB XC32, desarrollados por el mismo fabricante que el microcontrolador. El lenguaje de programación con el que tendremos que programar el PIC32, en el entorno del MPLAB X IDE, es lenguaje 'C'.

Los guiones incluirán un objetivo, una descripción detallada del periférico que vamos a utilizar, una descripción de la programación usando librerías y otra sin usarlas, añadiendo un ejemplo para cada caso y, por último, un control de versiones.

Para la programación usando librerías, nos basaremos en las que nos ofrece el propio fabricante de la placa y solo nos centraremos en utilizar las funciones que se nos proporciona. Mientras que, en el modo sin librerías, realizaremos una programación a bajo nivel, en el que tendremos que definir todas las operaciones en lenguaje de programación 'C'.

## 1.2 OBJETIVOS

El principal objetivo de este proyecto es desarrollar los primeros guiones de prácticas con los que comenzar un Laboratorio Colaborativo de Tecnología Electrónica basado en la placa Basys MX3. Estos guiones suponen un trabajo de desarrollo y aprendizaje, que ayudará a futuros alumnos a empezar a programar con el software de programación, usado para trabajar con el microcontrolador PIC32 de Microchip, denominado MPLAB X IDE.

Basándose en este trabajo, los usuarios lograrán aprender, de forma autónoma y progresiva, a programar microcontroladores PIC32, y más concretamente la placa Basys MX3. Finalmente, tendrían que reflejar su propio aprendizaje en otros guiones similares, que sirvan de apoyo a futuros usuarios.

El conjunto de las seis prácticas de este trabajo cumplirá con el objetivo de explicar, desde la instalación del software necesario para programar cualquier función de la placa Basys MX3, hasta ser capaz de controlar distintos periféricos de los que dispone, pasando por una descripción detallada de los mismos.

A continuación, expondremos un resumen del objetivo individual de cada práctica:

- Práctica nº 1: El objetivo de esta práctica es descargar e instalar el software necesario para poder programar la placa Basys MX3.

- Práctica nº 2: El objetivo de esta práctica es indicar los pasos para crear un nuevo proyecto, crear los archivos de configuración para controlar nuestra placa y, por último, mostrar como programar los diodos LED.

- Práctica nº 3: El objetivo de esta práctica es aprender a programar los interruptores que incorpora nuestra placa y mostrar su configuración interna.

- Práctica nº 4: El objetivo de esta práctica es aprender a programar el LED RGB y los botones que incorpora nuestra placa y mostrar su configuración interna.

- Práctica nº 5: El objetivo de esta práctica es aprender a programar los conectores Pmod que incorpora nuestra placa, programar el Pmod 8LD que conectaremos a la placa y también mostrar la configuración interna de ambos dispositivos.

- Práctica nº 6: El objetivo de esta práctica es aprender a programar el Display de cuatro dígitos y siete segmentos que incorpora nuestra placa y mostrar su configuración interna.

El objetivo de estos guiones no trata de ser un documento que se deba seguir estrictamente paso a paso, si no que sirva de orientación y ayuda, de modo que no sea un documento cerrado. Esto quiere decir, que no es necesario seguir el guion de cada práctica de principio a fin para lograr aprender a programar los microcontroladores PIC32 y que, cualquier usuario podrá modificar estos guiones de prácticas, tratando de mejorar lo expuesto en ellos anteriormente, consiguiendo progresar y hacerlos más completos y efectivos.

# CAPÍTULO 2:
# ESTADO DEL ARTE

## 2.1 TARJETAS DE DESARROLLO

En el mercado actual hay una gran variedad de tarjetas de desarrollo, de una amplia gama de fabricantes, por lo que escoger la que más se adapta a nuestras necesidades se puede convertir en una ardua tarea.

Si para solucionar este problema optamos por buscar información en internet encontraremos una gran cantidad de páginas que nos describirán todas esas placas, por lo que tampoco nos facilitara mucho el trabajo.

*Figura 1. Amplia gama de placas de desarrollo*

Si pensamos en placas de desarrollo para aprendizaje enseguida se nos viene a la cabeza la palabra ARDUINO. Buscando un poco por internet, veremos que hay una gran cantidad de información acerca de este software de programación y gran cantidad de modelos de placas basados en este entorno de desarrollo.

El mayor inconveniente que vemos en estas placas es que, aparte de adquirir la placa, tendríamos que comprar una amplia gama de periféricos con los que poder trabajar, para después tener que montar los circuitos con los que poder ensayar nuestro trabajo. Esto nos limita el campo de búsqueda de la placa de desarrollo que usaremos en este trabajo.

En la Figura 2, podemos ver una amplia variedad de placas basadas en el entorno de desarrollo integrado, ARDUINO.


*Figura 2. Diferentes placas ARDUINO*

Dentro de las opciones disponibles de placas de desarrollo para aprendizaje, que disponen de periféricos integrados, se encuentran las placas Basys 2 y Basys 3, que están basadas en la programación en FPGA. Este tipo de programación no es el que buscamos para la realización de este trabajo, por lo que descartamos el uso de estas placas.

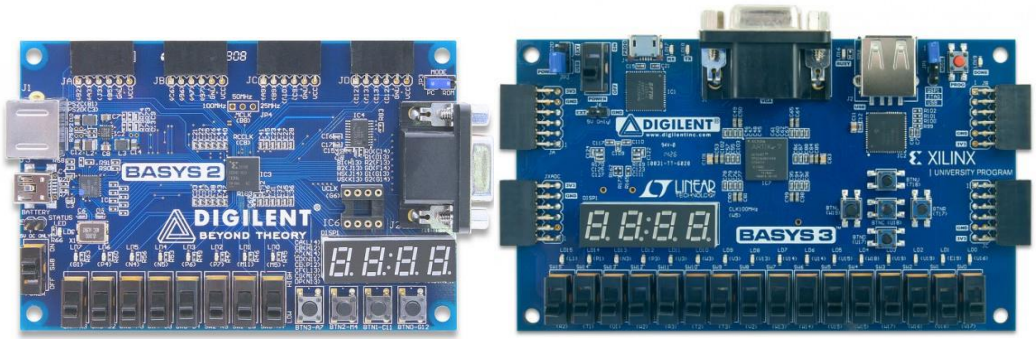



*Figura 3. Placas basadas en programación FPGA*

Dentro de las opciones que hemos encontrado, nos decantamos por la placa Basys MX3, que dispone de un procesaador PIC de 32 bits, con una amplia gama de perifericos incorporados. Gracias a ellos, podremos realizar diferentes ensayos para comprobar su funcionamiento. Ademas, otro de los motivos por el que escogemos esta placa, es que dispone de conectores Pmod con los que poder ampliar sus capacidades.

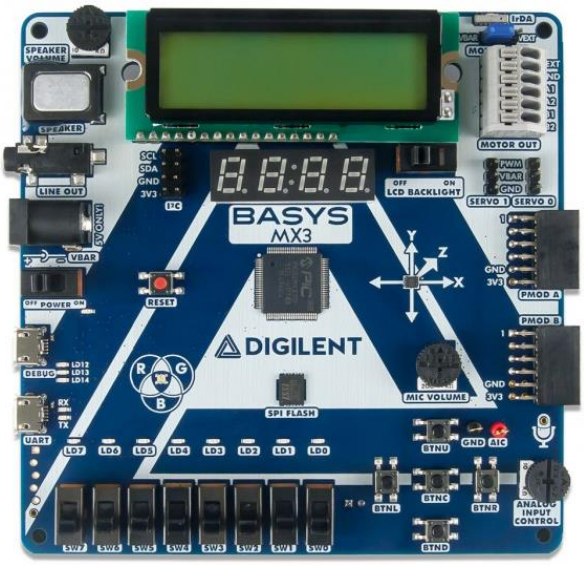



*Figura 4. Placa Basys MX3*

Podemos aplicar lo aprendido con la placa Basys MX3, para después programar otro tipo de placas más pequeñas y que incorporen el mismo procesador. En estas placas, podremos conectar el Pmod que necesitemos en ese momento y de este modo, ocupar un menor espacio. Un ejemplo de este tipo de placas podría ser la ChipKit Cmod.

*Figura 5. Placa ChipKit Cmod*

## 2.2 PLACA BASYS MX3

La placa Basys MX3 es una placa de aprendizaje de la marca Digilent Inc. diseñada especialmente para impartir cursos introductorios a sistemas integrados. El usuario puede crear gran variedad de aplicaciones, de una manera sencilla y rápida, gracias a que incorpora una amplia gama de periféricos integrados de forma compacta y la posibilidad de incorporar fácilmente otros a mayores.

La placa Basys MX3 contiene un microcontrolador PIC32 y está diseñada para ser utilizada con el entorno de desarrollo integrado, MPLAB X IDE.

La placa Basys MX3 dispone de varios dispositivos y periféricos con los que poder realizar diferentes pruebas y ensayos de programación, de una manera sencilla y rápida, sin la necesidad de incorporar otros elementos complementarios.

### 2.2.1 MICROCONTROLADOR PIC32

El procesador que controla la placa Basys MX3, es un microcontrolador PIC32MX370F512L de la marca Microchip, con un núcleo MIPS32 M4K, que puede funcionar hasta 80 MHz usando un oscilador de 8 MHz integrado.

Además, el microcontrolador dispone de:

- 512 KB de memoria flash de programa, 12 KB de memoria flash de arranque.
- 128 KB de SRAM.
- Cuatro módulos de acceso directo a la memoria (DMA).
- Incorpora dos interfaces SPI, dos $I^2C$ y cinco UART.
- Puerto maestro paralelo (PMP) para interfaces gráficas.
- Cinco temporizadores/Contadores de 16 bits.
- Cinco módulos de captura de entrada.
- Cinco módulos de comparación de salida.
- 85 pines de Entrada/Salida, de los cuales 54 pines son compatibles con Peripheral Pin Select (PPS) para la reasignación de funciones.

## 2.2.2 FUENTE DE ALIMENTACIÓN

La placa Basys MX3 requiere de una fuente de alimentación de 5 Voltios para su funcionamiento. Esta alimentación puede provenir del puerto USB-DEBUG (J12), del puerto USB-UART (J10) o de una fuente de alimentación externa de CC de 5 Voltios, conectada al conector de alimentación (J11). Tal y como se puede observar en la Figura 6.

Estas tres entradas de alimentación están conectadas entre sí mediante diodos Schottky para formar la red de alimentación de entrada primaria, 'VIN', utilizada para alimentar los reguladores y la mayoría de los periféricos integrados. La placa se encenderá automáticamente mientras el interruptor de encendido (SW8) este en la posición ON y haya alimentación en cualquiera de las entradas de alimentación.

Un LED de alta intensidad (LD11), impulsado por la salida del regulador LMR10515 de 3.3 Voltios, indicará que la placa está recibiendo alimentación.

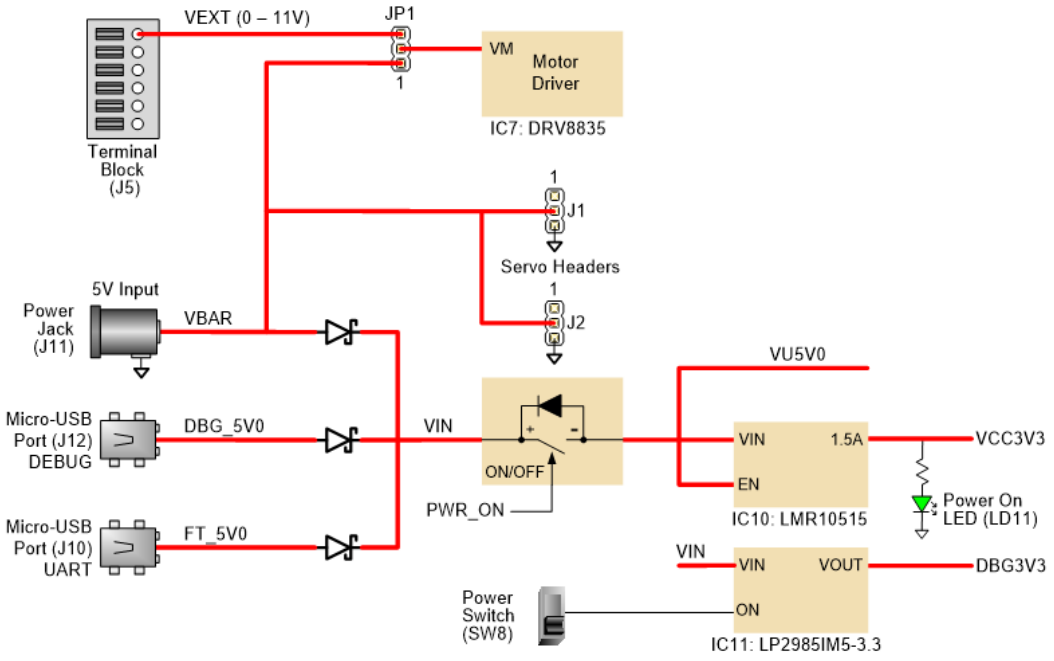



*Figura 6. Esquema del circuito de alimentación de la placa Basys MX3*

Los puertos USB pueden entregar suficiente energía para la gran mayoría de dispositivos disponibles. Sin embargo, algunos pueden requerir más energía de la que un puerto USB puede proporcionar. En estos casos, se debe utilizar una fuente de alimentación externa, como es el caso de los motores y de los servomotores, que no se pueden alimentar a través de ninguno de los puertos USB y solo pueden alimentarse a través de una fuente externa, debido a su alta demanda de corriente.

La fuente de alimentación externa se tiene que conectar en el Power Jack (J11) de la placa, mediante un enchufe coaxial de 2.0 mm de diámetro interno, y proporcionar un voltaje de 5 voltios de CC (con un mínimo de 4,75 voltios y un máximo de 5,5 voltios). Idealmente, la fuente debe ser capaz de proporcionar 20 Watios de potencia (5 Voltios de CC y 4 Amperios de intensidad).

El controlador para el motor que incorpora la placa Basys MX3 (Texas Instruments DRV8835) puede ser alimentado por una fuente de 5 voltios conectada a la toma de corriente J11 o por una fuente externa (de 0 a 11 Voltios), conectada a los pines 1 y 2 del bloque de terminales J5. El puente JP1 se usa para seleccionar qué fuente de energía usa el controlador del motor.

### 2.2.3 PERIFÉRICOS

La Placa Basys MX3 incorpora un gran número de periféricos con los que poder realizar diferentes pruebas y ensayos:

- Cinco pulsadores
- Botón de reinicio
- Ocho interruptores deslizantes
- Ocho LEDs
- Un LED RGB
- Un Display de cuatro dígitos y siete segmentos
- Pantalla LCD de 2x16 con retroiluminación
- Altavoz con conector de salida de audio y control de volumen
- Micrófono con control de volumen
- Controlador de motor de doble puente en H para hasta dos motores ce CC o un motor paso a paso
- Dos conectores para servomotores
- Módulo IrDA compatible con FIR
- Potenciómetro
- Acelerómetro de 3 ejes y 12 bits
- 4 MB de memoria flash SPI

Además, incluye varios puertos con los que poder expandir las características de la placa como son:

- Dos conectores Pmod de 2x16 pines.
- Un conector I²C.

En la Figura 7 podemos observar la localización exacta de todos los periféricos que incorpora la placa Basys MX3.

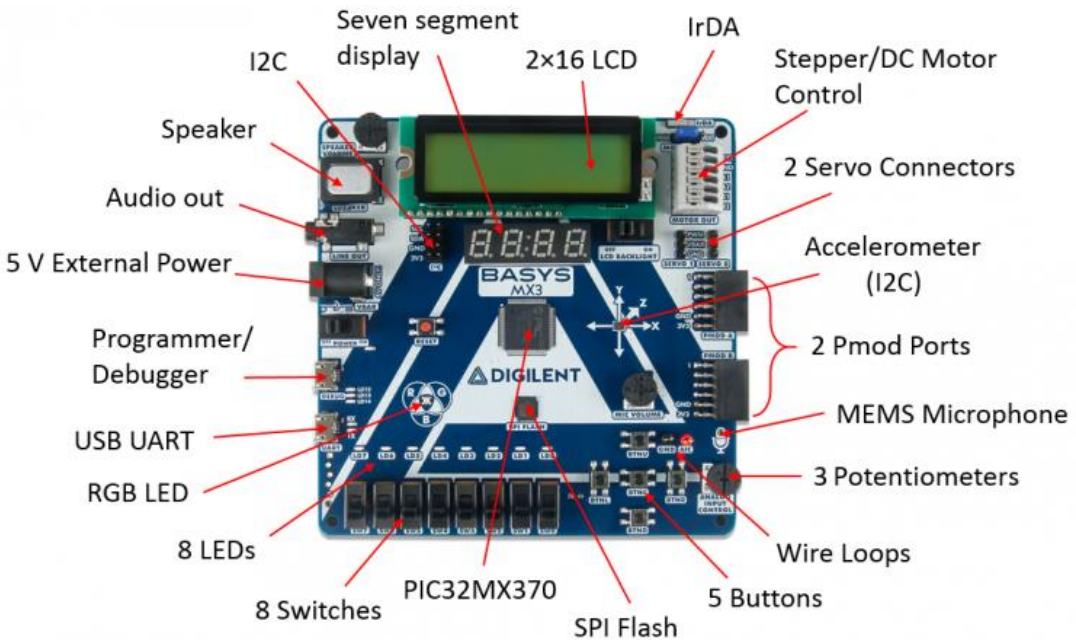



*Figura 7. Localización de los periféricos en la placa Basys MX3*

## 2.3 MPLAB X IDE

La placa Basys MX3 es totalmente compatible con el software de programación MPLAB X IDE de Microchip. Este paquete software se puede descargar de forma gratuita desde la página web del fabricante.

Esta herramienta se utiliza para escribir, compilar, programar y depurar el código que se ejecuta en la placa. La programación y la depuración de un programa usando el MPLAB X IDE es posible usando el conector USB DEBUG que incorpora la placa, ya que contiene todos los circuitos necesarios para que MPLAB X se comunique con el PIC32 incorporado. No es necesario comprar herramientas de programación adicionales como sucede con otras placas del mismo fabricante, en el que sí son necesarios la utilización de Programadores/Depuradores externos.

Para la mayoría de las placas de este fabricante necesitaríamos cuatro elementos principales para comenzar a programar:

- Un ordenador donde ejecutar MPLAB X IDE
- Un compilador
- Un Programador/Depurador interno o uno externo como lo son: el MPLAB Real ICE, el MPLAB ICD 3 y el MPLAB PICkit 3.
- Una placa de destino con un microcontrolador PIC o un software de simulación.


*Figura 8. Programadores/Depuradores externos*

En la Figura 8 podemos ver los tres Programadores/Depuradores de hardware externos más populares de la marca Microchip. Estos se conectan por un lado a la placa del microcontrolador, mediante un conector RJ45, y por el otro lado al ordenador, mediante un puerto USB. Estos dispositivos pueden trabajar de dos modos diferentes: en modo programador y en modo depurador.

## ❖ MODO PROGRAMADOR

Cuando actúan como programadores, estos dispositivos permiten que el código generado por el usuario se programe en la memoria no volátil del microcontrolador PIC conectado. Una vez reiniciado el microcontrolador PIC, el código de la aplicación comenzara a ejecutarse.


*Figura 9. Programador/Depurador trabajando como Programador*

## ❖ MODO DEPURADOR (DEBUGGER)

Cuando actúan como depuradores, permite la ejecución controlada desde el entorno MPLAB de programas en tiempo real y utilizando los recursos internos del propio microcontrolador. Además, con la posibilidad de parar, ejecutar paso a paso, ver el estado de los registros interno, establecer puntos de ruptura, etc.

Esta capacidad de control y monitoreo es lo que le da al usuario la capacidad de depurar errores de programación.

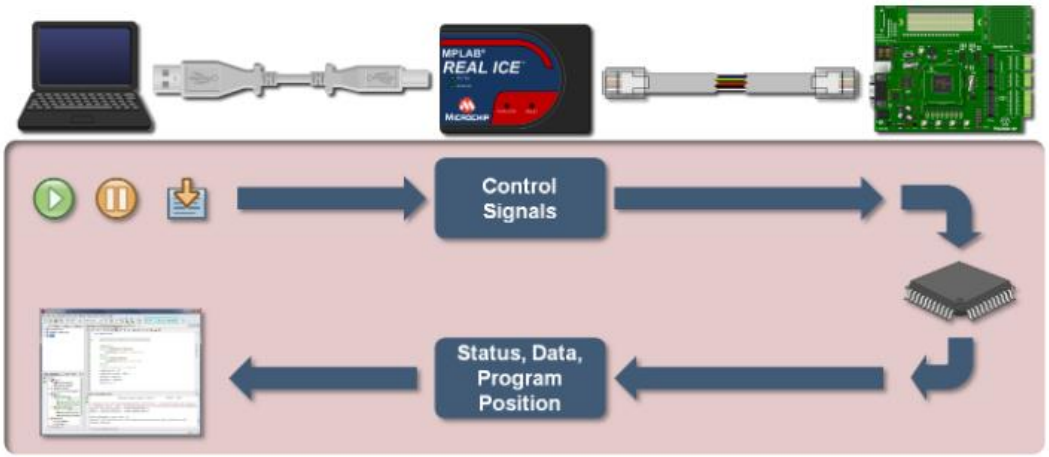



*Figura 10. Programador/Depurador trabajando como Depurador*

## 2.4 MICROCONTROLADORES PIC

Los PIC son una familia de microcontroladores tipo **RISC,** (**R**educed **I**nstrution **S**et **C**omputer. Computador con Conjunto de Instrucciones Reducidas), fabricados por Microchip Tecnology Inc. y derivados del PIC1650, originalmente desarrollados por la división de microelectrónica de General Instrument.


*Figura 11. PIC1650 de General Instrument*

El nombre actual no es un acrónimo, en realidad, el nombre completo es **PICmicro**, aunque generalmente se utiliza **P**eripheral **I**nterface **C**ontroller (**C**ontrolador de **I**nterfaz **P**eriférico).

El PIC original se diseñó para ser usado con la nueva CPU de 16 bits, CP1600. Esta tenía malas prestaciones de entradas y salidas, y en 1975, se desarrolló el PIC de 8 bits, para mejorar el rendimiento del sistema quitando peso de entradas y salidas a la CPU.

Los PIC32 de la marca Microchip son una familia de microcontroladores complejos y poderosos, que ofrecen la posibilidad de controlar muchos periféricos para fines mecatrónicos, en concreto el PIC32MX370F512L, que incorpora la placa Basys MX3, dispone de 100 pines y se encuentra empaquetado en un encapsulado TQFP.


*Figura 12. PIC32MX370F512L con encapsulado TQFP*

Las principales ventajas de los PIC de 32 bits, sobre los de 8 bits usados anteriormente, es que son más rápidos (la velocidad de reloj máxima es de 80MHz en comparación con los 40MHz de los anteriores), tienen más periféricos disponibles, ofrecen más memoria de programa (flash) y memoria de datos (RAM), y tienen significativamente más potencia computacional.

La Figura 13 muestra el diagrama de bloques de la arquitectura del PIC32. En él podemos ver, entre otras cosas, un conjunto de periféricos formados por PORTA-PORTG que son los puertos de E/S digitales, un reloj y calendario en tiempo real (RTCC) que puede mantener año, mes, día y hora de forma precisa, cinco UART para comunicación serie asíncrona, un puerto de maestro paralelo (PNP) para comunicación paralela, dos módulos de comunicación serie síncrona I²C y dos SPI.

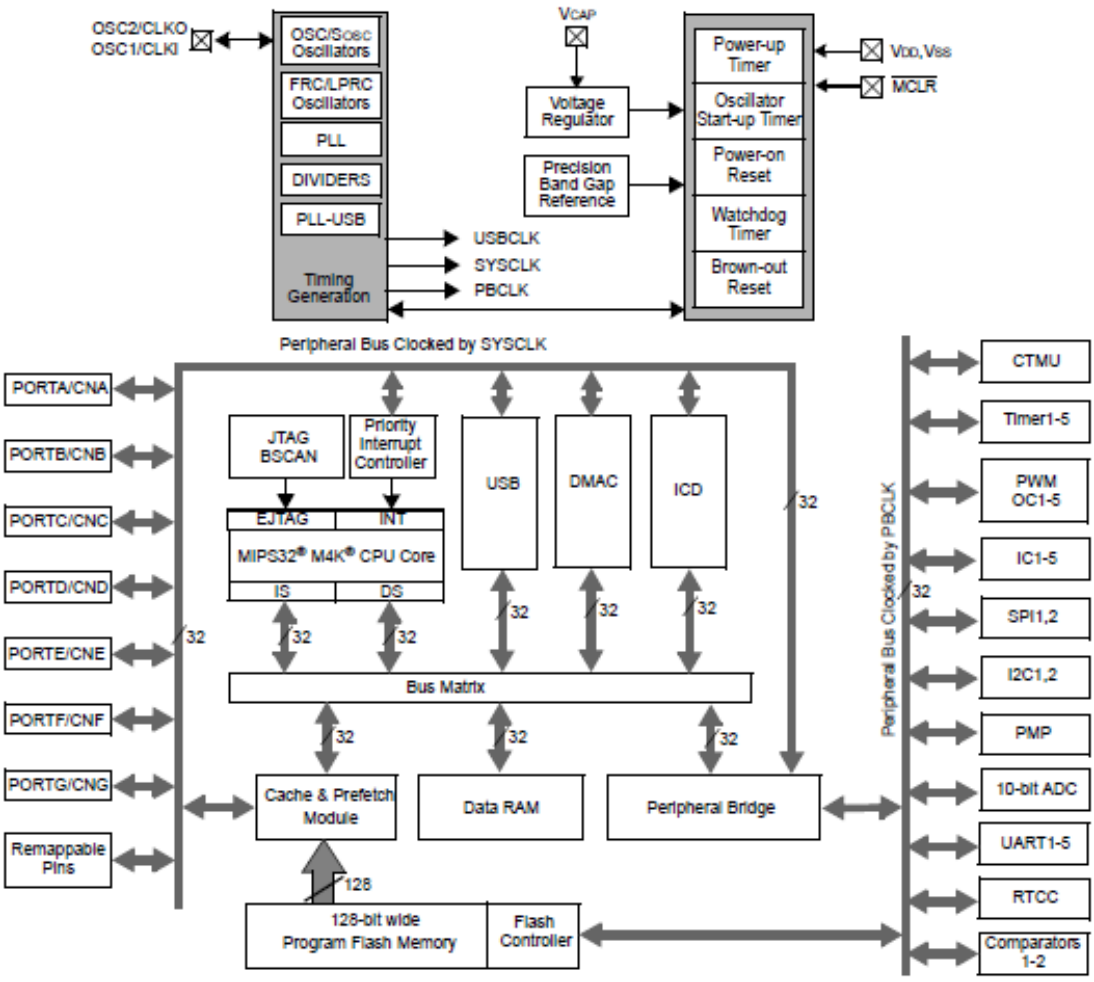



*Figura 13. Diagrama de bloques de la arquitectura del PIC32*

Los periféricos están en dos buses diferentes, uno es un bus sincronizado por el reloj del sistema 'SYSCLK' y el otro está sincronizado por el reloj del bus periférico, 'PBCLK'. Estos pueden tener la misma frecuencia, o el bus PBCLK puede tener una frecuencia más baja, dependiendo de cómo se configure los bits de configuración del reloj.

En la parte superior de la figura se encuentra la zona correspondiente a la generación de tiempo, que usa un cristal externo o un circuito de tiempo (para un tiempo más preciso), o un circuito interno, para crear SYSCLK, PBCLK y USBCLK (que se utiliza para la comunicación USB). La placa Basys MX3 usa un oscilador de cristal externo de 8 MHz para generar un SYSCLK y PBCLK de hasta 80 MHz, usando un bucle de bloqueo de fase (PLL) para multiplicar la frecuencia. Esto lo establece el software, no el diseño físico de la placa.

La CPU también interactúa con un 'módulo de precarga' (caché). Este módulo obtiene bloques de instrucciones del programa de la memoria del programa flash antes de su uso por parte de la CPU, para intentar limitar los retrasos debidos a una memoria flash relativamente lenta.

# CAPÍTULO 3:
# DESARROLLO DEL TFG

En este capítulo pretendemos presentar seis guiones de prácticas que detallarán el funcionamiento y las características de distintos periféricos que incorpora la placa Basys MX3, de modo que sirvan de ayuda a otros alumnos para empezar a programar con este tipo de placas de desarrollo.

## 3.1 INSTALACIÓN DEL SOFTWARE NECESARIO PARA CONTROLAR LA PLACA BASYS MX3 [PRÁCTICA Nº 1]

### ❖ OBJETIVO

El objetivo de la práctica nº 1 es descargar desde la web del fabricante todos los elementos software necesarios (MPLAB X IDE y MPLAB XC32), e instalarlos en nuestro ordenador personal para poder trabajar con la placa BASYS MX3.

### ❖ MATERIAL NECESARIO

#### HARDWARE

- Ordenador personal

#### SOFTWARE

- Microchip MPLAB X IDE
- Compilador MPLAB XC32

### 3.1.1 LA PLACA BASYS MX3

La placa Basys MX3 es una placa compacta de aprendizaje, que incorpora un microcontrolador PIC32MX370F512L, de la marca Microchip, y un conjunto de periféricos conectados a dicho microcontrolador.

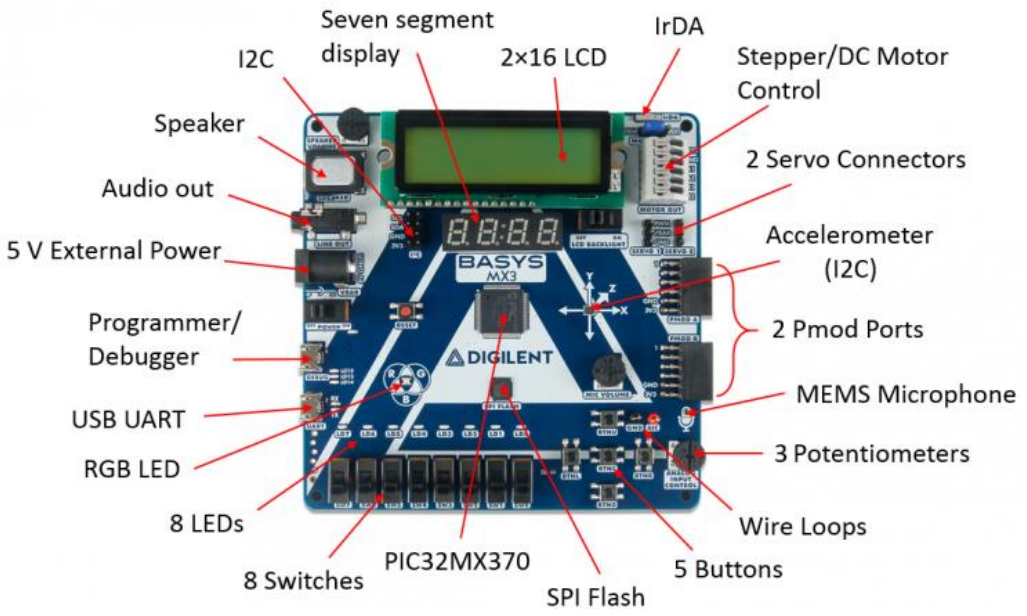



Figura 14. Descripción de la placa Basys MX3

Esta placa fue diseñada para programarse con el software MPLAB X IDE, el cual se ejecuta desde el ordenador y se usa para desarrollar aplicaciones para microcontroladores de la marca Microchip.

### 3.1.2 DESCARGAR E INSTALAR MPLAB X IDE

Este paquete de software se puede descargar de forma gratuita desde la página web del fabricante donde podemos escoger el tipo de software según sea el sistema operativo de nuestro ordenador (Windows, Linux o MAC).

- Página web del fabricante: *http://www.microchip.com/mplab/mplab-x-ide (Comprobado el 10/01/2019)*

El entorno de desarrollo integrado MPLAB X IDE es la herramienta que usaremos para escribir, compilar, programar y depurar el código que se ejecuta en la placa BASYS MX3.

La placa contiene todos los circuitos necesarios para que MPLAB X se comunique con el PIC32 integrado, mediante el conector DEBUG USB. Por lo que no es necesario adquirir herramientas de programación adicionales.

## ❖ EJECUTAR EL INSTALADOR

Vamos a la ubicación donde se guardó el instalador que acabamos de descargar y lo ejecutamos otorgándole los permisos de seguridad necesarios, si el sistema operativo nos lo solicitase.

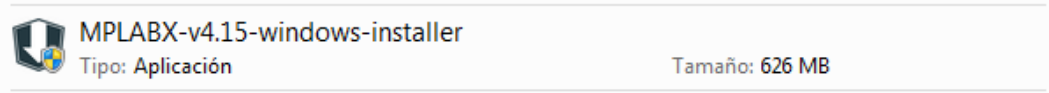



*Figura 15. Icono del archivo ejecutable del MPLAB X IDE*

## ❖ PREPARAR LA INSTALACIÓN

Para comenzar con la instalación, hacemos clic en *'Next >'*.

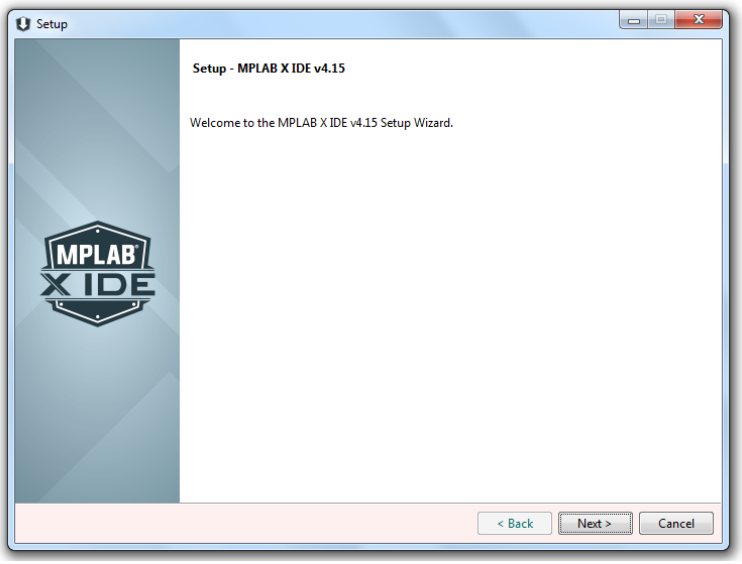



*Figura 16. Ventana inicial de la instalación*

## ❖ ACUERDO DE LICENCIA

Si estamos de acuerdo con las condiciones de uso y licencia, las aceptamos marcando dicha opción y haciendo clic en *'Next >'*.

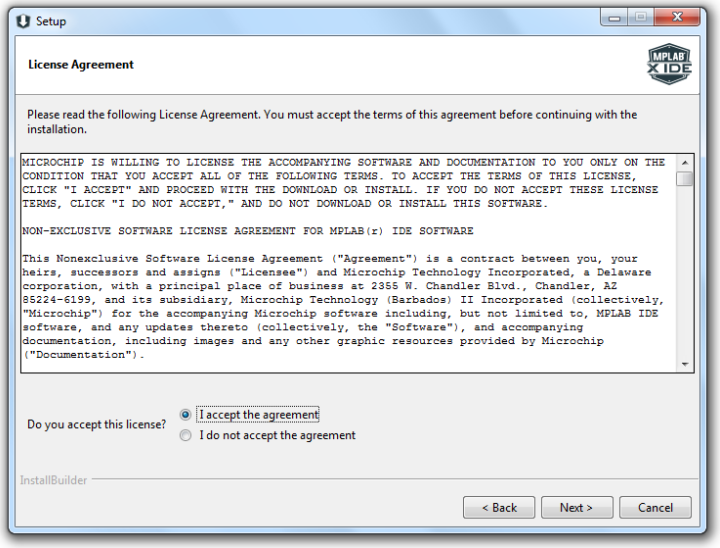



*Figura 17. Ventana de acuerdo de licencia*

## ❖ DIRECTORIO DE INSTALACIÓN

Por defecto MPLAB X IDE se instalará en un directorio predeterminado, con el resto de programas de nuestro ordenador. Si prefiriésemos un directorio diferente, tendríamos que hacer clic en el icono de la carpeta, a la derecha del cuadro de texto, y seleccionar la ubicación de instalación deseada. Una vez seleccionada la ubicación hacemos clic en *'Next >'*.

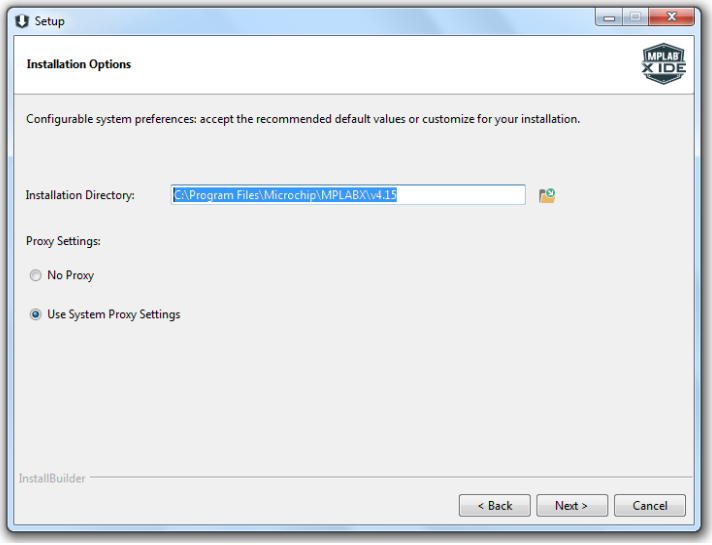



*Figura 18. Ventana de directorio de instalación*

## ❖ SELECCIÓN DE PROGRAMAS

El instalador nos da la opción de instalar MPLAB X IDE, MPLAB IPE o ambos marcando o desmarcando las casillas correspondientes.

MPLAB IPE (entorno de programación integrado) es una aplicación de software independiente de MPLAB X IDE, que proporciona una interfaz simple para acceder rápidamente a las funciones de programador para dispositivos de Microchip PIC. Como solo se trata de programación, es una aplicación pequeña, sensible y rápida.

En nuestro caso, con instalar MPLAB X IDE sería suficiente para poder trabajar con la placa Basys MX3, ya que además de las funciones de MPLAB IPE contiene las opciones de desarrollo y depuración de código. Marcamos las casillas deseadas y hacemos clic en *'Next >'*.

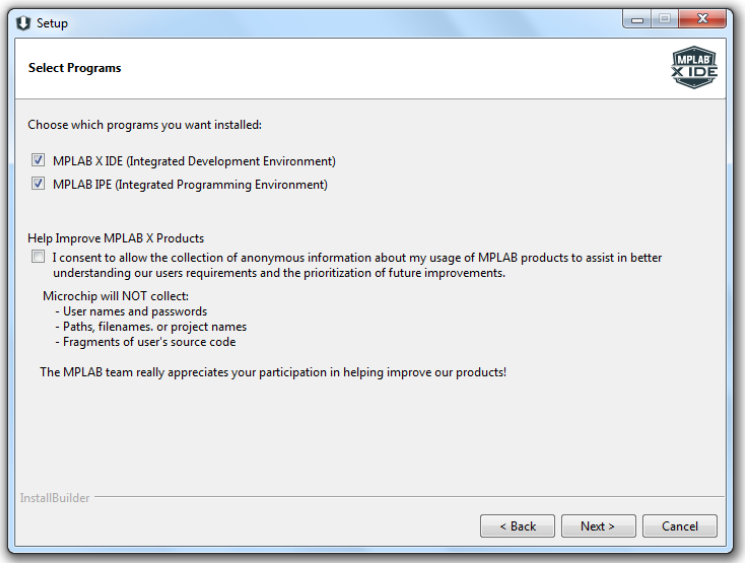



*Figura 19. Ventana de Selección de programas*

## ❖ LISTO PARA INSTALAR

Si estamos seguros de todos los pasos anteriores hacemos clic en *'Next >'* para comenzar la instalación, en caso contrario podemos volver hacia atrás haciendo clic en *'< Back'* para modificar alguna selección anterior.

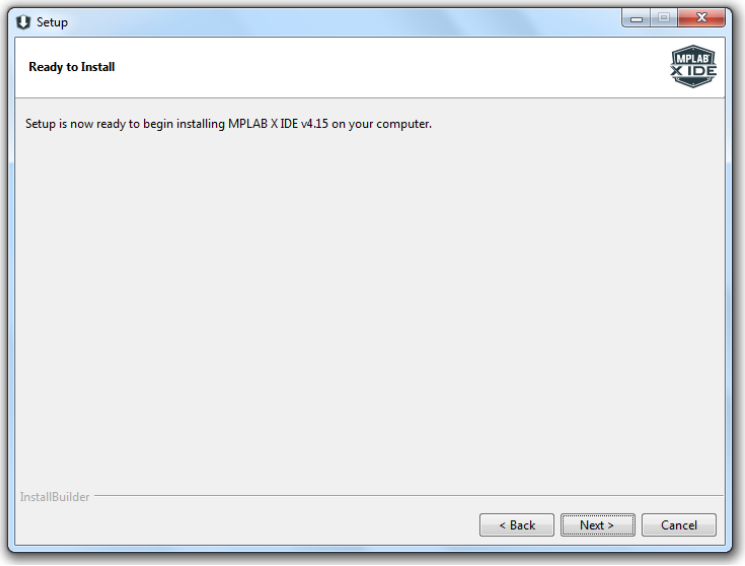



*Figura 20. Ventana instalación lista*

## ❖ INSTALACIÓN

Esperamos unos minutos hasta que el instalador haya terminado de instalar todos los complementos del entorno de desarrollo integrado (IDE).

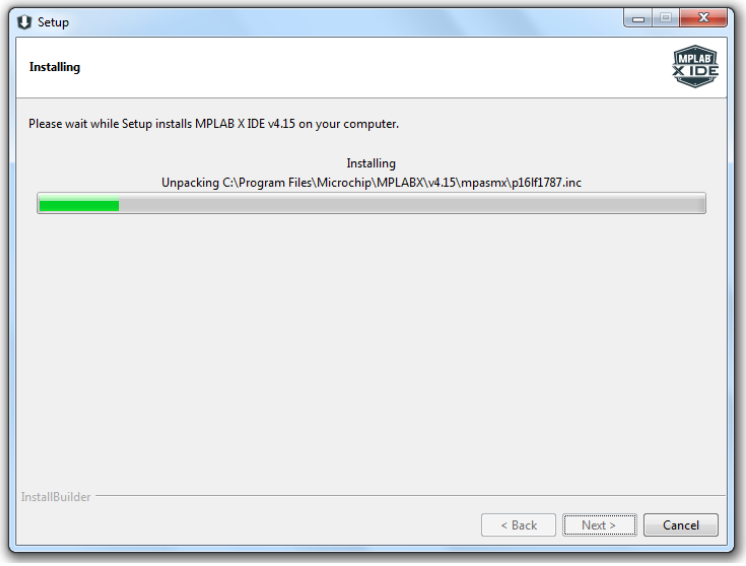



*Figura 21. Ventana de instalación*

## ❖ COMPLETAR LA INSTALACIÓN

Una vez completada la instalación de MPLAB X IDE, nos da una serie de opciones que debemos marcar o desmarcar y que nos llevarán a diferentes páginas web, desde donde podremos descargarnos diferentes complementos para nuestro programa. Estos complementos son: XC Compilers, MPLAB Harmony y MPLAB Code Configurator (MCC).

*'XC Compilers'* se trata de un conjunto de compiladores/ensambladores que consiguen una reducción del tamaño del código y mejoras de velocidad, que benefician nuestros proyectos de programación.

*'MPLAB Harmony'* es una plataforma de desarrollo de firmware flexible, abstracta y completamente integrada para microcontroladores PIC32. Incluye un conjunto de bibliotecas periféricas, controladores y servicios de sistema, que son fácilmente accesibles para el desarrollo de aplicaciones.

*'MPLAB Code Configurator (MCC)'* es un entorno de programación libre y gráfico, usa una interfaz intuitiva capaz de habilitar y configurar un amplio conjunto de periféricos y funciones específicas para los proyectos.

En nuestro caso, dejaremos marcada la primera opción, la correspondiente a XC Compilers, ya que es el único complemento que necesitamos para poder trabajar con la placa Basys MX3, aunque no será necesario puesto que, en el siguiente apartado indicamos como continuar la instalación y donde poder descargar este elemento.

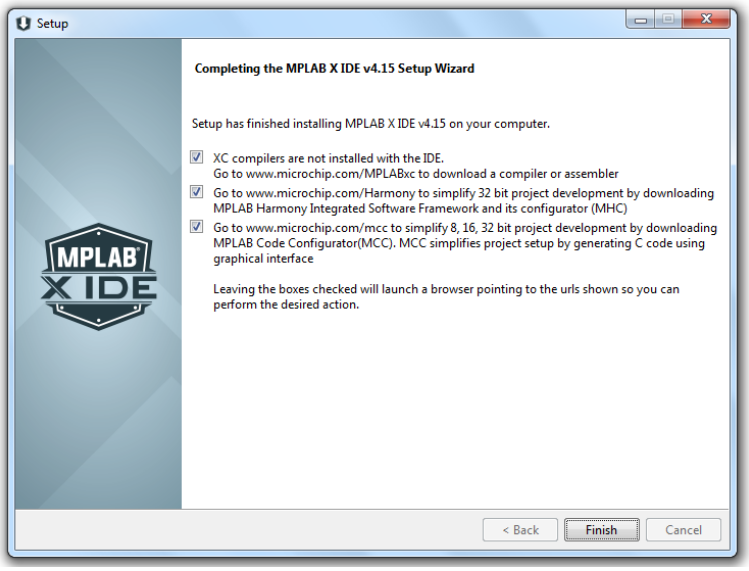



*Figura 22. Ventana instalación completa*

Para terminar con la instalación hacemos clic en *'Finish'*. La instalación está completa y deberan aparecer en nuestro escritorio tres iconos: MPLAB X IDE, MPLAB IPE (si hemos marcado la casilla correspondiente) y MPLAB driver switcher.


*Figura 23. Iconos de los 3 programas instalados*

*'MPLAB driver switcher'* es una aplicación que nos facilitaría cambiar entre los controladores de dispositivos MPLAB 8 y los controladores de dispositivos MPLAB X. En este caso no lo usaremos, ya que vamos a trabajar exclusivamente con MPLAB X y no necesitaremos la conmutación con MPLAB 8.

### 3.1.3 DESCARGAR E INSTALAR EL COMPILADOR MPLAB XC32

Escribiremos el código del programa en lenguaje 'C', usando el editor proporcionado por MPLAB X IDE. Un compilador convierte los programas en lenguaje 'C' en una representación binaria del código y que el procesador PIC puede ejecutar. El archivo generado por el compilador se carga en nuestra placa Basys MX3 mediante el puerto DEBUG USB.

La placa Basys MX3 tiene un módulo de Programador/Depurador (Programmer/Debugger) incorporado, que permite una conexión directa con nuestro ordenador. Si el Programador/Depurador no está integrado en la placa, se requiere un programador externo como el PICkit3 o un programador de ChipKit.

Desde la página web del fabricante se ofrece una amplia gama de compiladores MPLAB XC de forma gratuita, de modo que tendremos que seleccionar el adecuado para nuestro microcontrolador. En nuestro caso, escogeremos el compilador MPLAB XC32/32++ que es compatible con todos los microcontroladores PIC de 32 bits.

- Página web de fabricante: *http://www.microchip.com/mplab/compilers (Comprobado el 10/01/2019).*

## ❖ EJECUTAR EL INSTALADOR

Vamos a la ubicación donde se guardó el instalador que acabamos de descargar y lo ejecutamos otorgándole permisos de seguridad, si nuestro sistema operativo nos lo solicitara.

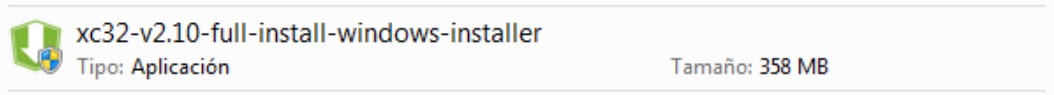



*Figura 24. Icono del archivo ejecutable del MPLAB XC32*

## ❖ PREPARAR LA INSTALACIÓN

Para comenzar la instalación hacemos clic en *'Next >'*.

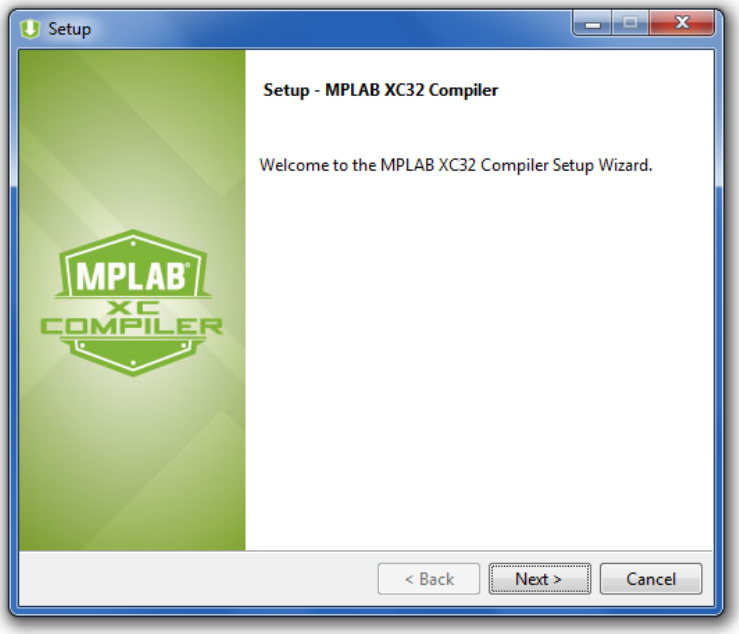



*Figura 25. Ventana inicial de la instalación*

## ❖ ACUERDO DE LICENCIA

Si estamos de acuerdo con las condiciones de uso y licencia, las aceptamos marcando dicha opción y haciendo clic en *'Next >'*.

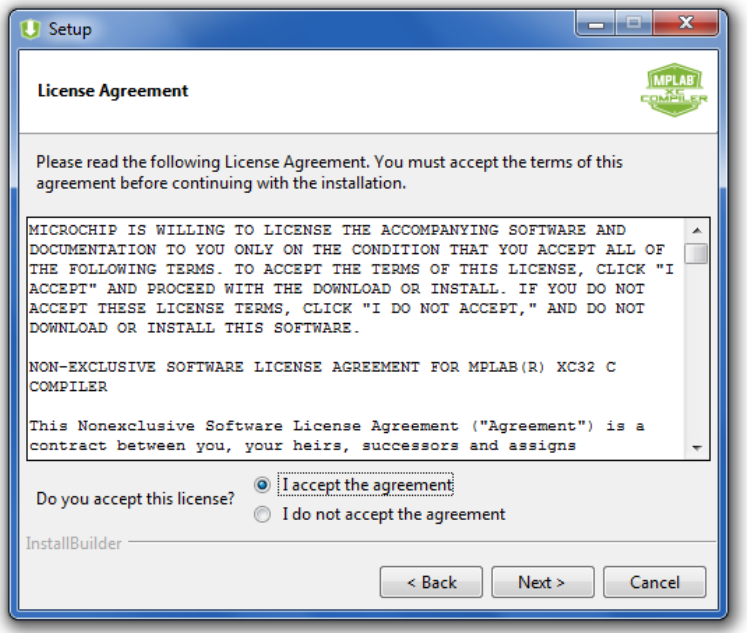



*Figura 26. Ventana de acuerdo de licencia*

## ❖ TIPO DE LICENCIA

En nuestro caso usaremos una licencia gratuita. Marcamos la opción y hacemos clic en *'Next >'*.

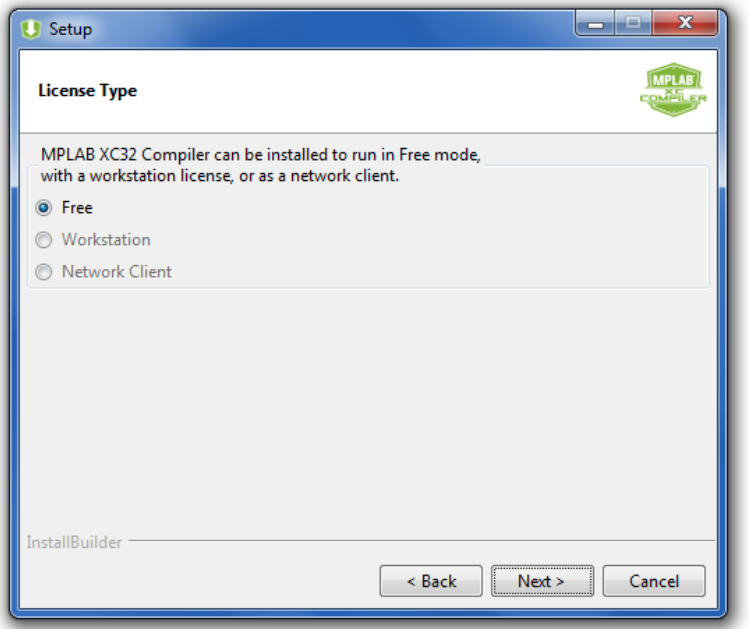



*Figura 27. Ventana de selección del tipo de licencia*

## ❖ DIRECTORIO DE INSTALACIÓN

Por defecto MPLAB XC32, se instalará en un directorio predeterminado, dentro de la carpeta llamada *'Microchip'* creada en el paso anterior. Si prefiriésemos un directorio diferente, tendríamos que hacer clic en el icono de la carpeta, a la derecha del cuadro de texto, y seleccionar la ubicación de instalación deseada. Una vez seleccionada la nueva ubicación, hacemos clic en *'Next >'.*

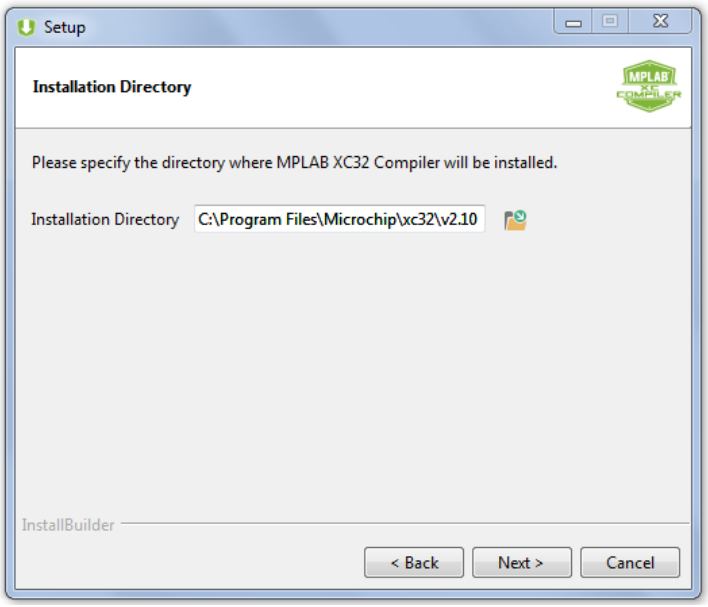



*Figura 28. Ventana de directorio de instalación*

## ❖ CONFIGURACIÓN DEL COMPILADOR

Marcamos o desmarcamos las opciones según sean nuestras necesidades. Se recomienda dejar marcada la primera, ya que así se aplica la misma configuración para todos los usuarios de este ordenador, y marcar la segunda, si planeamos usar el compilador xc32 desde la línea de comando. Una vez marcada la configuración hacemos clic en *"Next >".*

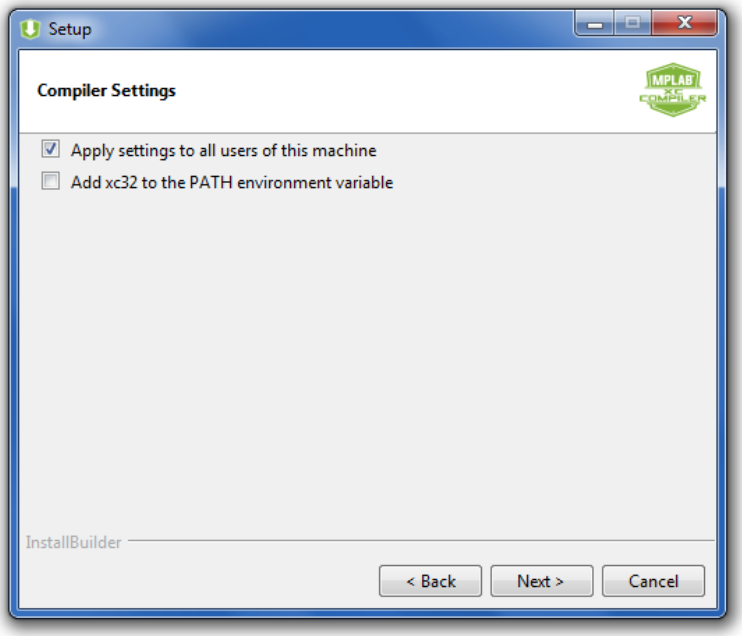



*Figura 29. Pantalla de configuración del compilador*

## ❖ LISTO PARA INSTALAR

Si estamos seguros de todos los pasos anteriores hacemos clic en *"Next >"* para comenzar el proceso de instalación, en caso contrario podemos volver hacia atrás haciendo clic en *"< Back"* para modificar alguna selección anterior.

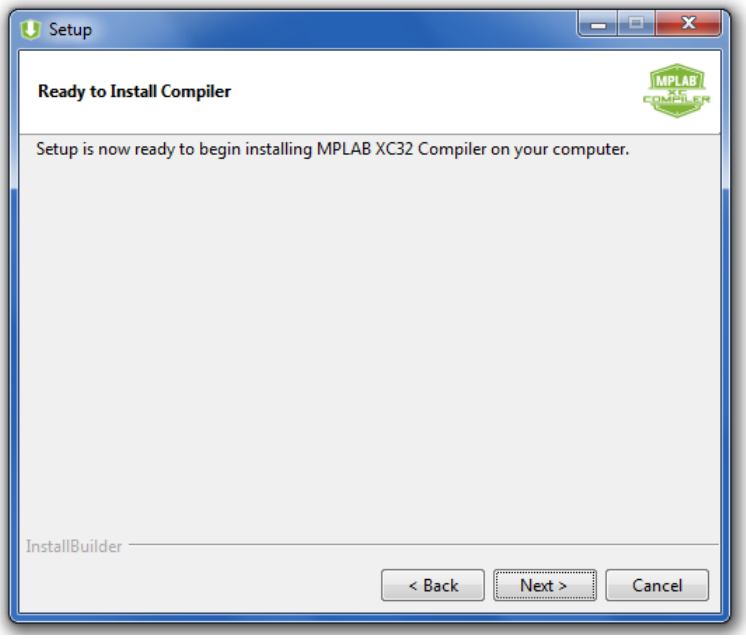



*Figura 30. Ventana de listo para instalar el compilador*

## ❖ INSTALACIÓN

Esperamos unos minutos hasta que el instalador haya terminado de instalar todos los complementos del compilador MPLAB XC32.

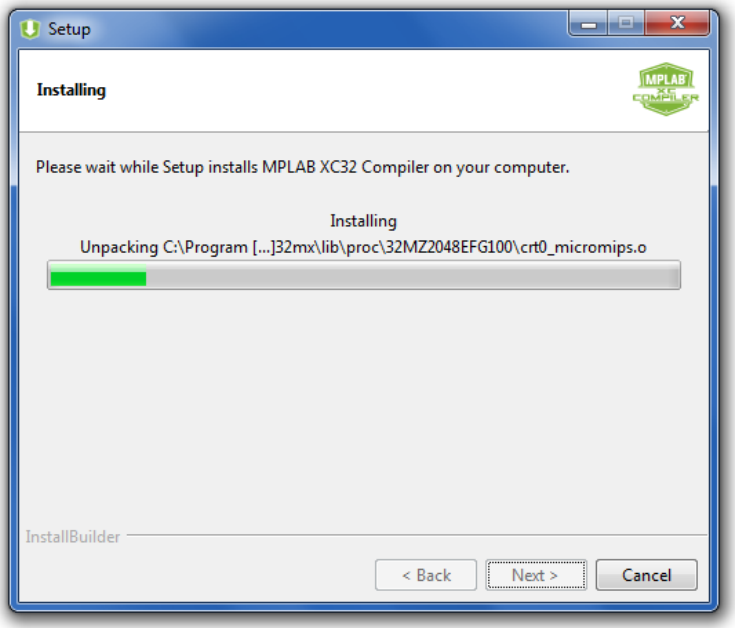



*Figura 31. Ventana del proceso de instalación*

## ❖ INFORMACIÓN DE LICENCIA

Aunque algunas funciones de optimización están deshabilitadas, para usar una versión gratuita del compilador, tenemos que hacer clic en 'Next >'.

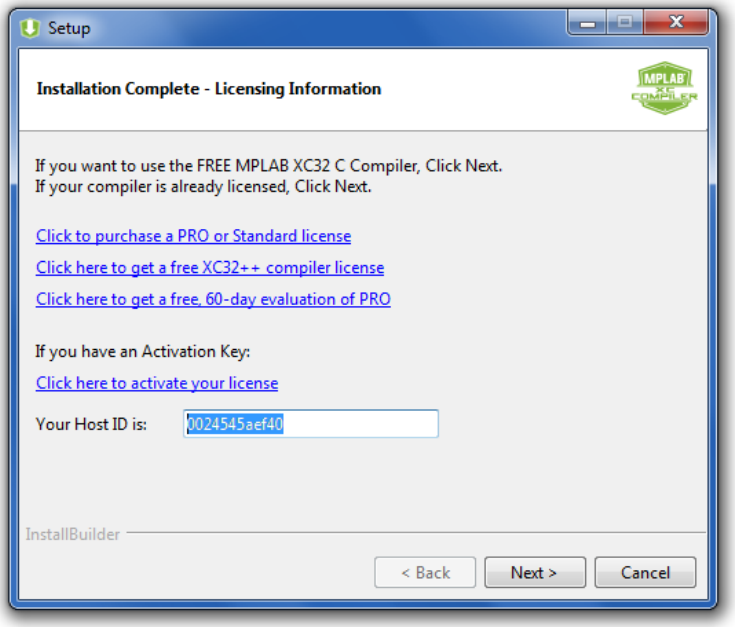



*Figura 32. Ventana de información de licencia*

### ❖ INSTALACIÓN COMPLETA

La instalación ya estaría completada. Hacemos clic en *'Finish'* para cerrar la ventana.

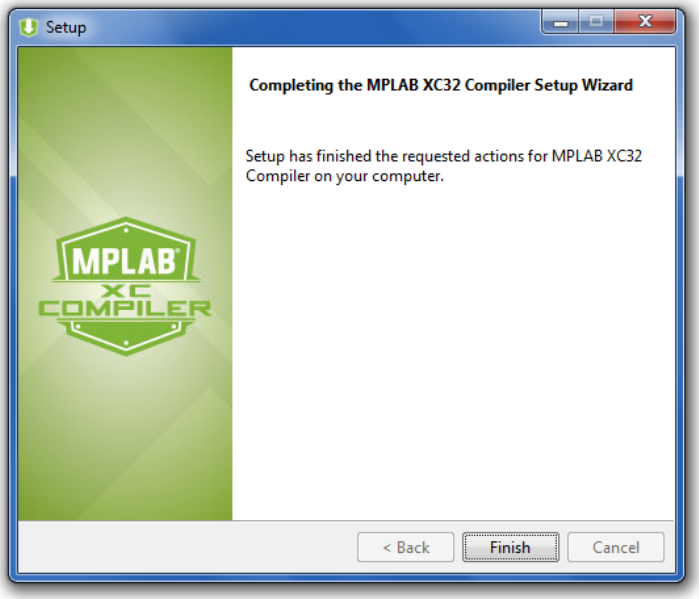



*Figura 33. Ventana de instalación completa*

Una vez instalados estos dos elementos software, ya dispondríamos de todos los elementos suficiente para empezar a programar con la placa de desarrollo Basys MX3.

## 3.1.4 CONTROL DE VERSIONES

| VERSIÓN | FECHA | COMENTARIOS |
|---|---|---|
| V 1.0 | 27/01/2019 | Versión inicial |
| | | |

*Tabla 1. Control de versiones práctica 1*

## 3.2 ESTABLECER LA VELOCIDAD DE LA CPU Y CONTROLAR EL ESTADO DE LOS LEDS DE LA PLACA BASYS MX3 [PRÁCTICA Nº 2]

### ❖ OBJETIVOS

El objetivo de la práctica nº 2 es la creación de un nuevo proyecto en MPLAB X IDE y el archivo de encabezado, donde estableceremos la velocidad de la CPU y el resto de parámetros de configuración del microcontrolador PIC32. Este archivo le usaremos en el resto de prácticas de este trabajo.

Además de todo lo anterior, también mostraremos las instrucciones necesarias para poder controlar los diodos LED, que incorpora la placa BASYS MX3.

Para tener una mejor visión de lo expuesto durante esta práctica lo probaremos mediante varios ejemplos en los que controlaremos el estado de los LEDs de la placa Basys MX3.

Esto lo realizaremos mediante el uso de las librerías, que nos proporciona el fabricante, y el uso del lenguaje “C” a bajo nivel.

### ❖ MATERIAL NECESARIO

#### HARDWARE

- Placa Basys MX3.
- Cable micro USB.
- Ordenador personal.

#### SOFTWARE

- Microchip MPLAB X IDE.
- Compilador MPLAB XC32.
- Librerías.

### 3.2.1 CREAR UN PROYECTO EN MPLAB X IDE

En este apartado vamos a explicar los pasos necesarios para crear un nuevo proyecto, desde cero, en **MPLAB X IDE**. Configuraremos el entorno y las herramientas específicas del dispositivo.

#### ❖ INICIACIÓN DE PROYECTO

Abrimos MPLAB X IDE y cerramos cualquier proyecto que pudiera estar abierto, ya que cuando el software arranca, se abre el último proyecto en el que se trabajó desde ese ordenador. Para cerrarlo podemos hacerlo de dos maneras diferentes: haciendo clic derecho sobre el nombre del proyecto y seleccionando *'Close'* o haciendo clic en '*File'* y seleccionando *'Close All Projects'.*

#### ❖ CREACIÓN DEL PROYECTO

Para crear un proyecto nuevo, hacemos clic en el icono de *'New Project'* o hacemos clic en *'File'* y seleccionamos *'New Project...'*.

*Figura 34. Icono para iniciar un proyecto nuevo*

## ❖ SELECCIÓN DE CATEGORÍA

Seleccionamos *'Microchip Embedded'* y luego *'Standalone Project'* y hacemos clic en *'Next >'*.

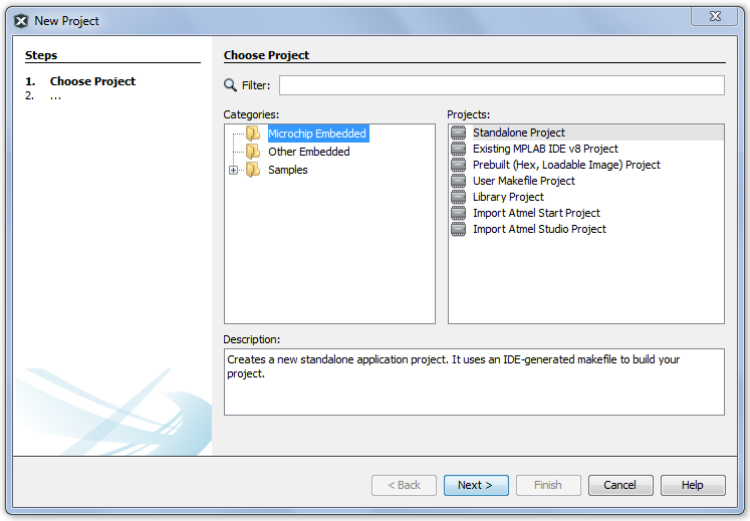



*Figura 35. Ventana de selección de categoría*

## ❖ SELECCIÓN DEL PROCESADOR

En el primer menú desplegable *'Family:'* seleccionamos la opción *'32-Bit MCUs (PIC32)'*, ya que nuestra placa Basys MX3 tiene un microcontrolador PIC32. En el segundo menú desplegable *'Device'*, seleccionamos el modelo *'PIC32MX370F512L'*, que sería el modelo de PIC32 que incorpora nuestra placa. Por último, hacemos clic en *'Next >'*.

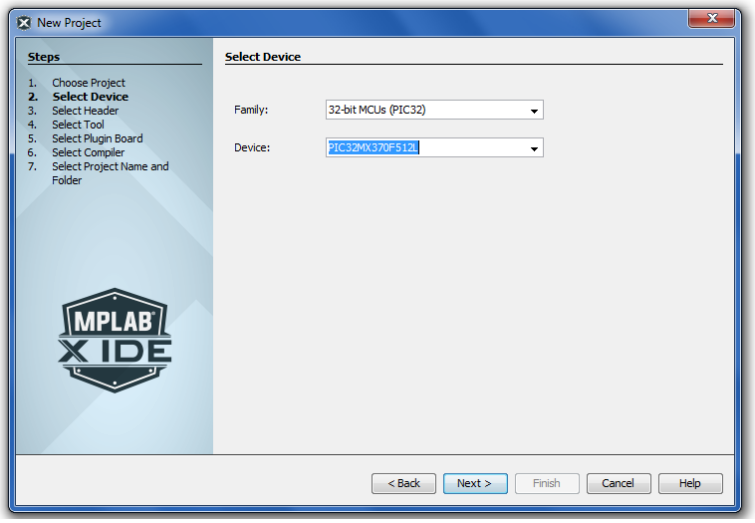



*Figura 36. Ventana de selección de dispositivo*

## ❖ SELECCIÓN DE HERRAMIENTAS

En el apartado *'Other Tools'* nos debe de aparecer una licencia de desarrollo al conectar la placa al USB del ordenador, mediante el puerto DEBUG USB.

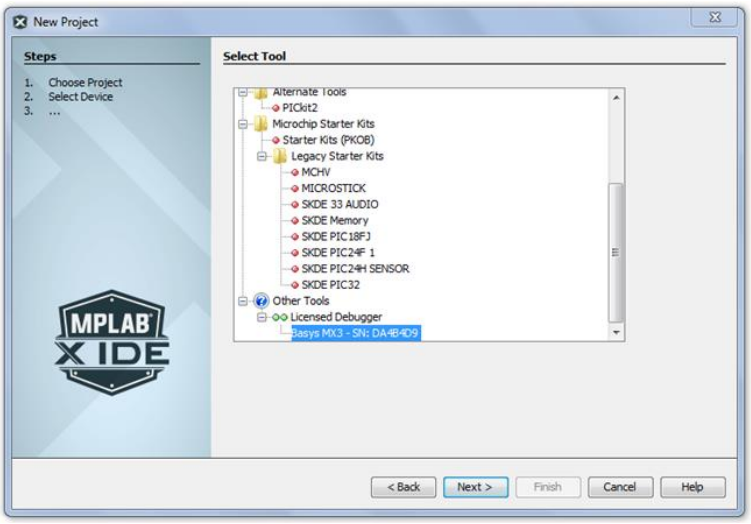



*Figura 37. Ventana de selección de herramientas*

El código de la licencia que debemos seleccionar es el correspondiente al número de serie de la placa que vamos a utilizar. Esta numeración la encontraremos escrito en el reverso de la misma, tal y como podemos observar en la imagen.


*Figura 38. Parte trasera de la placa Basys MX3*

Seleccionamos la licencia correcta de la placa que vamos a usar y hacemos clic en *'Next >'*.

**Nota:** si no conectamos la placa al ordenador y colocamos el interruptor general en la posición de encendido (ON), no nos aparecerá ninguna licencia de desarrollo. En el momento de compilar, nuestro programa nos dará error y nos solicitará esta herramienta.

## ❖ SELECCIÓN DEL COMPILADOR

Seleccionamos el compilador que instalamos en la práctica anterior *'XC32 (v2.10) …'* y hacemos clic en *'Next >'*.

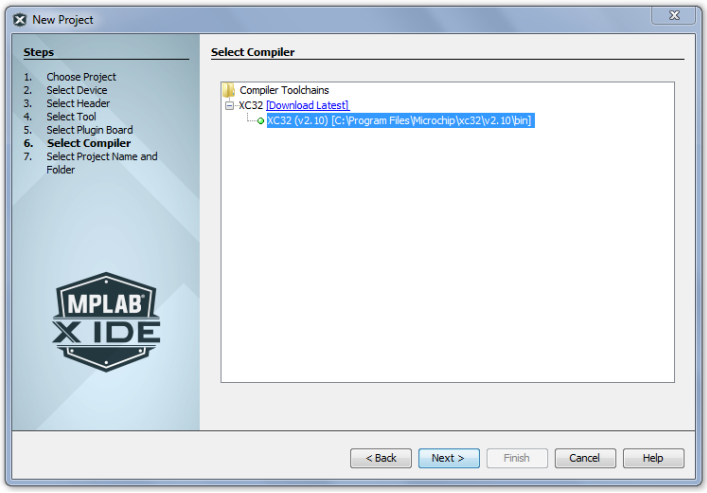



*Figura 39. Ventana de selección de compilador*

## ❖ NOMBRE DEL PROYECTO Y UBICACIÓN

Escribimos el nombre del proyecto en *'Project Name:'* y seleccionamos la localización donde deseamos guardar el proyecto en *'Project Location'*. Podemos modificar este directorio haciendo clic en *'Browse…'* y seleccionando la ubicación deseada.

Marcamos la casilla para hacer que este sea nuestro programa principal para poder seguir trabajando en él.

Por último, en el desplegable seleccionamos la opción ISO-8859-1, que es la norma que define la codificación del alfabeto latino (Alfabeto latino nº 1).

Cuando hayamos terminado, seleccionamos *'Finish'* para completar la creación del nuevo proyecto.

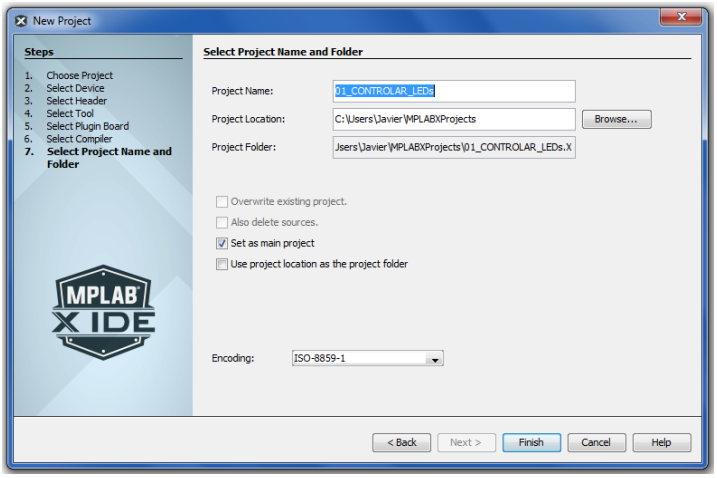



*Figura 40. Ventana de directorio de instalación y nombre*

## 3.2.2 CREAR UN ARCHIVO DE ENCABEZADO

En este apartado crearemos un archivo de encabezado con las características básicas necesarias para los proyectos que realizaremos con la placa Basys MX3, de modo que, para usarlo en los próximos proyectos que realicemos, bastará con que lo copiemos en cada nuevo proyecto.

### ❖ GENERAMOS EL ARCHIVO DE ENCABEZADO

Dentro de nuestro proyecto hacemos clic derecho en *'Header Files'*, seleccionamos *'New'* y hacemos clic en *'xc32_header.h...'*.

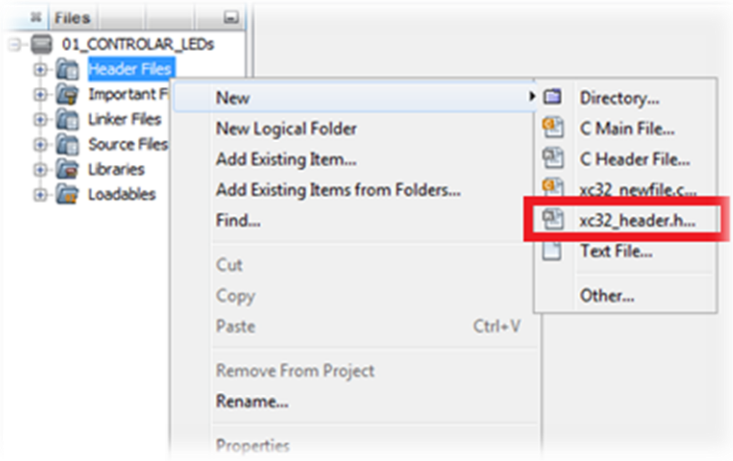



*Figura 41. Ruta para seleccionar crear nuevo archivo de encabezado*

Cambiamos el nombre en el recuadro *'File Name:'*. En nuestro caso lo llamaremos *'BasysMX3'* y en el recuadro *'Extension'* seleccionamos *'h'*. El resto de valores dejaremos los predeterminados. Para terminar, hacemos clic en *'Finish'*.

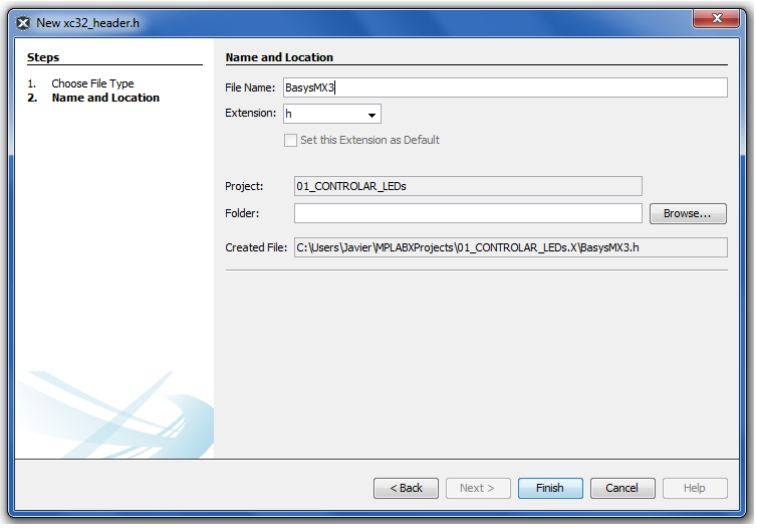



*Figura 42. Ventana selección directorio de instalación y nombre*

## ❖ MODIFICAR EL ENCABEZADO

Abrimos el archivo de encabezado *'BasysMX3.h'*, que acabamos de crear, y eliminamos desde la línea 21 hasta la 174 del texto generado automáticamente.

Sustituimos las tres veces que aparece *'_EXAMPLE_FILE_NAME_H'* por *'_BasysMX3_H'*

Algunos pines de nuestra placa se comparten, es decir, comparten funciones con otras señales utilizadas para la programación. Por lo tanto, hay que desactivar su función de programación. Además, también vamos a desactivar el temporizador watchdog. Para ello, vamos a incluir en nuestro código las siguientes instrucciones para desactivarlos respectivamente:

```
#pragma config JTAGEN = OFF
#pragma config FWDTEN = OFF
```

Después, creamos la estructura que usaremos para todos nuestros proyectos. Está formada por una función principal, en la que hay una zona de inicialización (setup) y un bucle infinito, que se repetirá constantemente (loop).

```
//ESTRUCTURA PROGRAMA PRINCIPAL
void setup();
void loop();

void main() {
    setup();
    while (1) {
        loop();
    }
}
```

Por último, realizaremos las ediciones apropiadas en los comentarios superiores del archivo de encabezado generados automáticamente, aunque esto no es realmente importante y no afecta al funcionamiento.

### 3.2.3 CONFIGURACIÓN DEL RELOJ DE LA CPU

El microcontrolador PIC32 admite numerosas opciones de configuración para el reloj del procesador principal. La placa Basys MX3 usa un cristal externo de 8 MHz para usar con la opción de oscilador XT. Las opciones del oscilador se seleccionan a través de las configuraciones especificas del microcontrolador usando: *'#pragma config'*.

Utilizando el reloj de sistema interno Phase Locked Loop (PLL) es posible seleccionar numerosos múltiplos o divisiones del oscilador de 8 MHz para producir frecuencias operativas de CPU de hasta 80 MHz.

El circuito del reloj PLL proporciona un divisor de entrada, un multiplicador y un divisor de salida. La frecuencia del reloj externo (8 MHz) primero se divide por el valor del divisor de entrada seleccionado, este se multiplica por el valor del multiplicador y luego se divide por el divisor de salida seleccionado. El resultado es la frecuencia de reloj del sistema SYSCLK.

La frecuencia SYSCLK es utilizada por la CPU, el controlador DMA, el controlador de interrupción, la memoria cache y los puertos de E/S. Además de otros muchos más elementos del microcontrolador PIC32.

Los valores que controlan la frecuencia operativa se especifican utilizando las variables de configuración PIC32MX370. Esto se establece usando la declaración de configuración *'#pragma config'.* Los comandos que debemos usar son:

- *#pragma config FPLLIDIV* para el divisor de entrada
- *#pragma config FPLLMULL* para el factor de multiplicación
- *#pragma config FPLLODIV* para el divisor de salida

Consultaremos la hoja de datos de la familia PIC32MX3xx/4xx y el manual de referencia de la familia PIC32MX, Sección de osciladores, para obtener información de cómo elegir los valores correctos, ya que no funcionará con todas las combinaciones de factores de multiplicación y división.

Además de configurar la frecuencia SYSCLK, el reloj del bus periférico, PBCLK también es configurable. Es el reloj utilizado por los temporizadores y los controladores en serie (UART, SPI, $I^2C$).

La frecuencia PBCLK es una división de la frecuencia SYSCLK seleccionada anteriormente utilizando *"#pragma config FPBDIV"*

El divisor PBCLK se puede configurar para dividir entre 1, 2, 4 u 8.

Nosotros fijaremos la frecuencia del reloj principal y la del bus periférico en 80 MHz. Esto lo hacemos mediante las siguientes instrucciones, que vamos a añadir en el archivo de encabezado *'BasysMX3.h',* justo antes de la configuración de la estructura del programa principal.

```
//FRECUENCIA RELOJ SISTEMA (SYSCLK)
#pragma config FNOSC    = FRCPLL
#pragma config FSOSCEN  = OFF
#pragma config POSCMOD  = XT
#pragma config OSCIOFNC = ON

#pragma config FPLLIDIV = DIV_2
#pragma config FPLLMUL  = MUL_20
#pragma config FPLLODIV = DIV_1

//FRECUENCIA BUS PERIFÉRICO (PBCLK)
#pragma config FPBDIV   = DIV_1
```

### 3.2.4 DESCRIPCIÓN DE PERIFÉRICOS

#### ❖ DIODOS LED

La placa Basys MX3 incorpora ocho LEDs, etiquetados como LD0 – LD7 en la placa y en los esquemas. Estos están conectados a ocho pines digitales E/S del microcontrolador PIC32. El control de los LEDs se realiza mediante el acceso básico a un pin Entrada/Salida y los posibles valores que les podemos dar es 1 o 0, dependiendo si queremos encenderlo o apagarlo respectivamente.

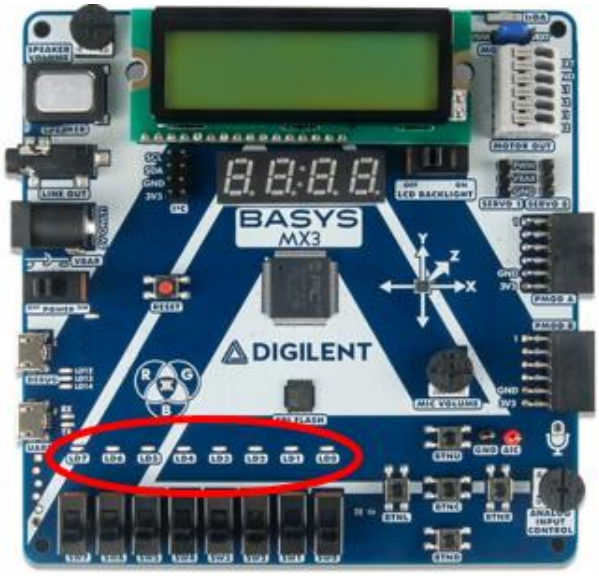



*Figura 43. Localización de los diodos LED en la placa Basys MX3*

En la Figura 44 podemos observar de forma esquemática la conexión eléctrica interna de los diodos LED dentro de la placa BASYS MX3.

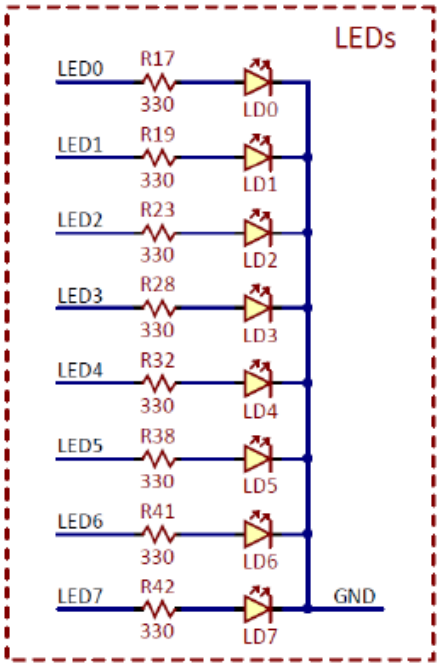



*Figura 44. Esquema eléctrico de los diodos LED*

Como podemos observar en el esquema, cada diodo LED va acompañado de una resistencia limitadora, para no sobrepasar la intensidad máxima soportada por los diodos LED.

La Tabla 2 muestra todas las señales asociadas a los diodos LED de la placa Basys MX3. En ella, podemos observar la etiqueta con la que aparecen en los esquemas y en la propia placa. Además de las posibles funciones de cada pin dentro del microcontrolador PIC32.

| LED | ETIQUETA | CARACTERÍSTICAS PIC32 |
|---|---|---|
| LED 0 | LD0 | TMS/CTED1/RA0 |
| LED 1 | LD1 | TCK/CTED2/RA1 |
| LED 2 | LD2 | SCL2/RA2 |
| LED 3 | LD3 | SDA2/RA3 |
| LED 4 | LD4 | TDI/CTED9/RA4 |
| LED 5 | LD5 | TDO/RA5 |
| LED 6 | LD6 | TRCLK/RA6 |
| LED 7 | LD7 | TRD3/CTED8/RA7 |

*Tabla 2. Señales asociadas a los pines de los diodos LED*

En lugar de utilizar los pines correspondientes a los diodos LED como pines normales de Salida, podemos asignar a cada pin del microcontrolador diferentes funciones para controlar otros periféricos. Esto se logra a través de la reasignación de cada pin.

### 3.2.5 AÑADIR LIBRERÍAS AL PROYECTO

Desde la página web del fabricante de la placa Basys MX3, nos ofrecen la posibilidad de descargarnos un conjunto de librerías, que nos servirán para facilitarnos el uso de los diferentes periféricos que incorpora.

Estas librerías ocultan los detalles de implementación de software y contienen las funciones de nivel inferior que acceden a los registros, lo que nos permite llamar a la función necesaria de una manera fácil e intuitiva.

#### ❖ DESCARGAR LIBRERÍAS

Tenemos que descargar el conjunto de librerías que nos proporciona el fabricante para poder incluirlas en nuestros proyectos.

- Página web de las librerías: *https://github.com/Digilent/Basys-MX3-library* (Comprobado el 03/02/2018)

Una vez descargado el archivo, lo descomprimimos y buscamos una carpeta que se llama *'LibPack.X'*. En ella, deberían estar todas las librerías correspondientes a los periféricos y un archivo que se llama ***'config.h'.*** Este archivo, junto con el que hemos creado anteriormente, ***'BasysMX3',*** tienen que estar siempre presentes en todos los proyectos que realicemos con la placa Basys MX3.

#### ❖ INCLUIR LAS LIBRERÍAS EN NUESTRO PROYECTO

Para incluir las librerías en nuestro proyecto lo primero que tenemos que hacer es copiar los archivos *'.c'* y *'.h'* correspondientes a los módulos que deseamos usar en cada práctica, dentro de la carpeta que se genera al crear nuestro nuevo proyecto.

En este caso, vamos a hacerlo con los archivos de librería correspondientes a los diodos LED. De esta forma, los archivos que tenemos que copiar dentro de la carpeta de nuestro proyecto, además del archivo de configuración ***'config.h'***, son los correspondientes a este módulo: ***'led.h'*** y ***'led.c'.*** Los copiamos y los pegamos en la carpeta de nuestro proyecto, que se encuentra dentro de la carpeta *'MPLABXProjects'*. Si no la encontramos basta con ver la ruta de guardado, que se sigue al crear el proyecto.

En la Figura 45 se muestra la carpeta del proyecto que estamos creando, donde podemos ver los archivos que acabamos de copiar.

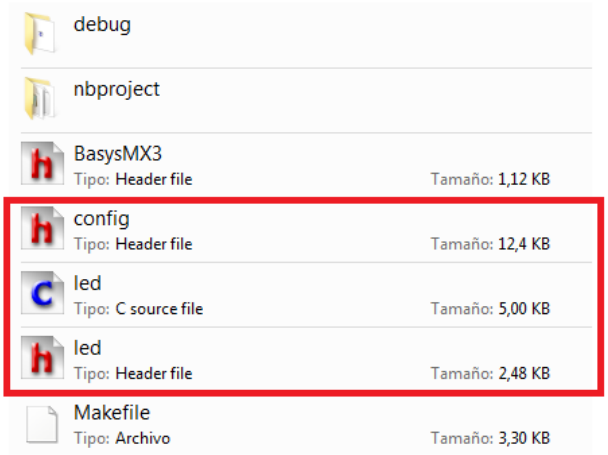



*Figura 45. Detalle de la carpeta de nuestro proyecto*

Para poder usar las librerías que acabamos de pegar en la carpeta, bastaría con añadir cada archivo a su lugar correspondiente de nuestro proyecto. Para incluir los archivos *'led.h'* y *'config.h',* tenemos que hacer clic derecho encima de *'Header Files'* dentro de nuestro proyecto y hacer clic en *'Add Existing Item...'*

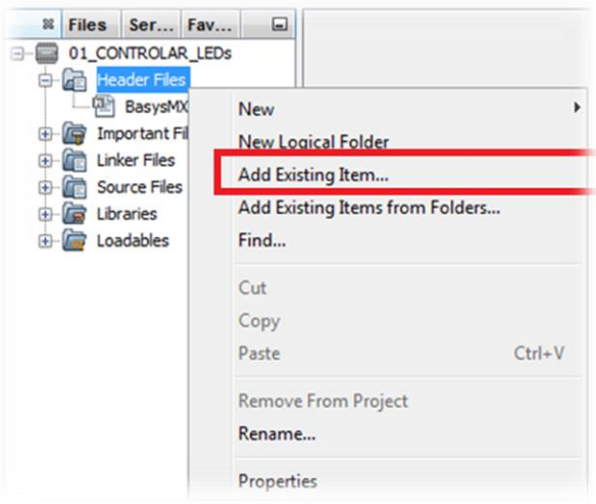



*Figura 46. Ruta para seleccionar añadir archivos de cabecera*

Seleccionamos los dos archivos dentro de la carpeta de nuestro proyecto y hacemos clic en *'Select'*. De este modo, ya estarían incluidos y listos para usarse.

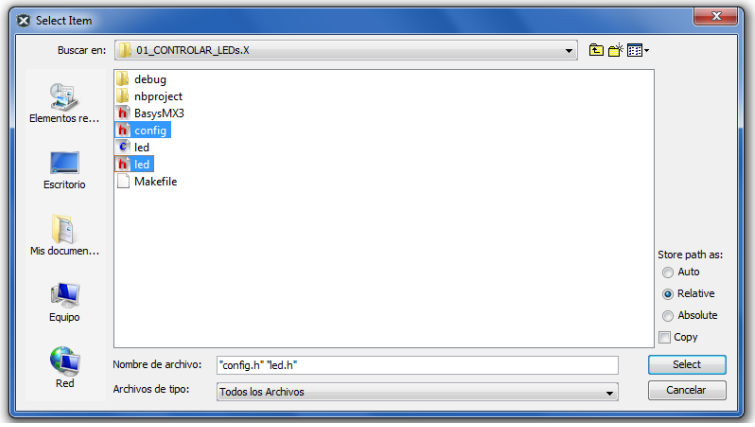



*Figura 47. Ventana para seleccionar archivos de cabecera*

Para terminar de incluir las librerías, faltaría incluir el archivo *"led.c"*, que se realizaría de forma similar a la anterior, pero en otro apartado de nuestro proyecto:

Hacemos clic derecho encima de *"Source Files"* y hacemos clic en *'Add Existing Item...'*.

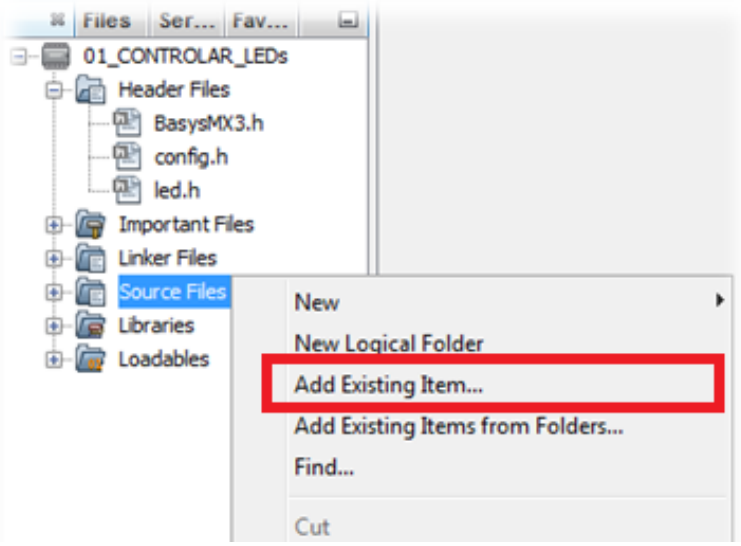



*Figura 48. Ruta para seleccionar añadir archivos fuente*

Seleccionamos el archivo *'led.c'*, que hemos copiado en la carpeta de nuestro proyecto, y hacemos clic en *'Select'*. Así ya estarían incluidos todos los archivos de librerías necesarios para empezar a programar.

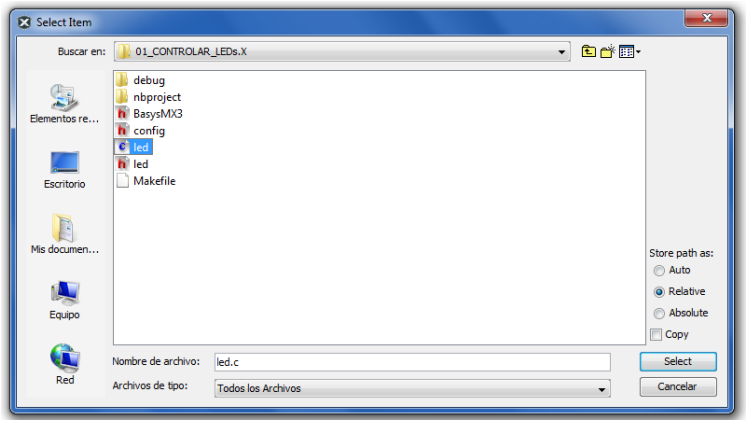



*Figura 49. Ventana para seleccionar archivos fuente*

## 3.2.6 CREAR LA FUNCIÓN PRINCIPAL

Todos los proyectos que creemos para programar la placa Basys MX3, deben contener una función principal llamada ***'main',*** donde escribiremos nuestro programa.

### ❖ CREAR LA FUNCIÓN main()

Hacemos clic derecho en el apartado *'Source Files'* de nuestro proyecto, seleccionamos *'New'* y hacemos clic en *'C Main File...'*.

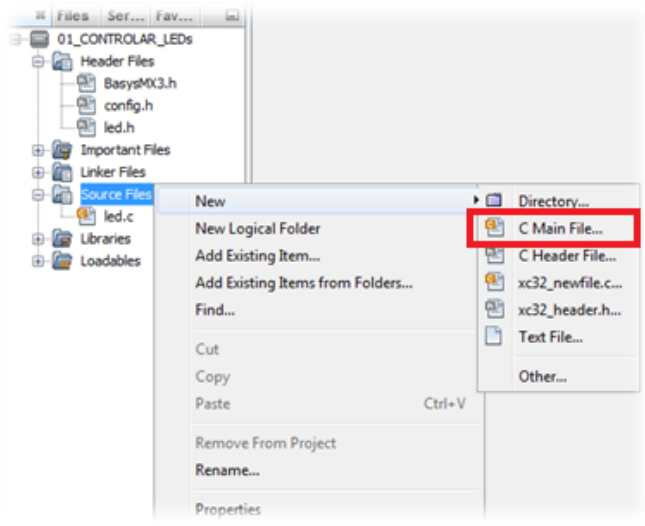



*Figura 50. Ruta para seleccionar crear un nuevo archivo fuente*

Indicamos el nombre en el apartado *'File Name:'* en nuestro caso lo llamaremos *"main"* y en el recuadro *'Extension'* seleccionamos *'c'*. El resto de valores dejaremos los predeterminados.

Importante asegurarse que lo guardamos en la carpeta de nuestro proyecto, donde hemos copiado los archivos de la librería. Para terminar, hacemos clic en *'Finish'*.

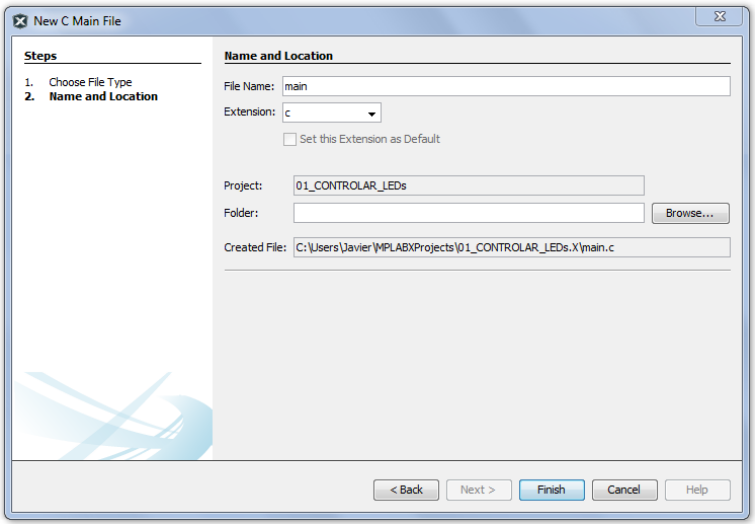



*Figura 51. Ventana para elegir directorio y nombre*

## ❖ ELEMENTOS DE LA FUNCIÓN main()

Dentro de la función main incluiremos tres apartados imprescindibles. Estos apartados son: el de inclusión de encabezados, la función setup() y la función loop().

En el primer apartado debemos incluir las librerías que vamos a usar y los archivos de encabezado que hemos creado o añadido anteriormente. Esto se realiza con el comando *'# include'*.

En el segundo apartado, función **setup()**, lo que tenemos que incluir es la parte del código que solo se repite una vez, al comienzo de nuestro programa. Instrucciones como declaración de variables, de funciones o inicializaciones de los diferentes módulos de la placa Basys MX3.

Por último, en la función **loop()**, debemos incluir la parte del código que se repite constantemente mientras el microcontrolador esté alimentado.

### 3.2.7 PROGRAMAR USANDO LIBRERÍAS

Una vez incluidos todos los archivos necesarios de los correspondientes módulos que utilizaremos en nuestro proyecto, tendremos que modificar la función principal.

#### ❖ ENCABEZADO

En este apartado lo que vamos a incluir, además de las librerías que nos aparecen por defecto al crear la función principal, son las librerías que hemos incluido anteriormente en el proyecto. Esto lo hacemos con las siguientes instrucciones:

```
#include "led.h"
#include "config.h"
#include "BasysMX3.h"
```

El archivo de encabezado de la librería para controlar los diodos LED solo será necesario introducirlo en los proyectos que se vayamos a usar dichos LEDs. Mientras que los correspondientes a las características del microcontrolador PIC32, tendremos que incluirlos en todos nuestros proyectos.

#### ❖ SETUP

Tenemos que crear la función setup(). Esta función no devuelve nada, por lo que su tipo de retorno es nulo. Además, la función setup() no tiene entrada, por lo que, dentro de los paréntesis que van después del nombre de la función, no debe ir escrito nada.

Dentro de esta función vamos a incluir la instrucción de inicialización para los LEDs de nuestra placa Basys MX3. Con una sola instrucción, gracias a las librerías que hemos incluido anteriormente, indicaremos que los pines correspondientes a los LEDs son pines de salida. Esto lo hacemos con la instrucción:

```
LED_Init();
```

### ❖ LOOP

Después de la función setup(), procedemos a escribir la función loop(). Esta se ejecutará inmediatamente después del setup() y se repetirá infinitas veces mientras el microcontrolador esté alimentado.

La instrucción que debemos introducir para controlar el estado de cada LED es: **LED_SetValue (num_LED, Estado)**. Donde *'num_LED'* corresponde a la identificación del LED, cuyo estado deseamos escribir. La identificación es mediante números, tal y como muestra la Tabla 3.

| LED | NÚMERO |
|---|---|
| LD0 | 0 |
| LD1 | 1 |
| LD2 | 2 |
| LD3 | 3 |
| LD4 | 4 |
| LD5 | 5 |
| LD6 | 6 |
| LD7 | 7 |

*Tabla 3. Identificación de los diodos LED*

Los posibles estados en los que se pueden encontrar cada LED son: apagado, cuando el estado indicado es 0, y encendido, cuando es igual a 1 o cualquier otro valor diferente de 0.

Por ejemplo, si queremos encender el LED número 3 debemos escribir:

```
LED_SetValue(3, 1);
```

### ❖ COMPILAR Y CARGAR EL PROGRAMA

Una vez escrito nuestro programa, tenemos que compilarlo para asegurarnos que no tenemos ningún error de programación y cargarlo en nuestra placa Basys MX3.

Lo que haremos será compilar el proyecto, utilizando el compilador seleccionado en las propiedades del proyecto. Desde la barra de herramientas principal, tenemos varias opciones para compilar nuestro programa. La primera opción solo compilará los archivos del proyecto que han cambiado desde la última compilación y la segunda, elimina los archivos de versiones anteriores y posteriormente crea los archivos nuevos necesarios del proyecto.

*Figura 52. Iconos para compilar el programa*

Por último, una vez que hayamos corregido los posibles errores, lo que tenemos que hacer será cargarlo en nuestra placa. Esto lo podemos realizar con cualquiera de los dos botones siguientes de la barra de herramientas principal.

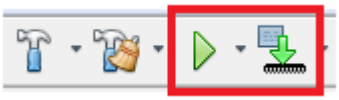


*Figura 53. Iconos para cargar el programa*

## 3.2.8 EJEMPLOS USANDO LIBRERÍAS

A continuación, vamos a comprobar lo expuesto en los apartados anteriores. Para verlo con más claridad, lo demostraremos mediante dos ejemplos en los que haremos parpadear un LED con dos velocidades de CPU diferentes.

Primero haremos parpadear un LED con una frecuencia de 80 MHz, ya fijada en el apartado 3.2.3. Después con la mitad, 40 MHz. Comprobaremos visualmente que, en el primer caso, como el periodo es menor, el parpadeo será más rápido.

Para hacer parpadear un LED necesitamos introducir en nuestro programa un tiempo de espera, de modo que el LED permanezca encendido y apagado ese tiempo. Esto lo conseguimos mediante una función que llamaremos **delay().**

La función delay() hará que el procesador entre en un bucle, que realizara un gran número de operaciones en las que no hará ninguna acción, y que introduciremos mediante la variable *"retardo"*. De esta forma, el tiempo de espera variará según el tiempo que el procesador tarde en realizar ese número de operaciones que hemos indicado en el programa.

La función delay() la copiaremos dentro de nuestro programa junto con el loop() y el setup().

```
//FUNCIÓN PARA INTRODUCIR UN DELAY
void delay(int retardo) {
    int i;
    for (i = 0; i < (retardo * 1000); i++) {
    }
}
```

### ❖ EJEMPLO 1: HACER PARPADEAR UN LED (80 MHz)

En este ejemplo vamos a mostrar cómo sería el programa que tendríamos que cargar en nuestra placa para hacer parpadear el LED nº 3.

```
#include <stdio.h>
#include <stdlib.h>
#include "config.h"
#include "led.h"
#include "BasysMX3.h"

//FUNCIÓN PARA INTRODUCIR UN DELAY
void delay(int retardo) {
    int i;
    for (i = 0; i < (retardo * 1000); i++) {
    }
}

void setup() {
    LED_Init();
}

void loop() {
    LED_SetValue(3, 1);
    delay(1000);
    LED_SetValue(3, 0);
    delay(1000);
}
```

### ❖ EJEMPLO 2: HACER PARPADEAR UN LED (40 MHz)

Para poder comprobar que efectivamente el LED parpadea a una velocidad inferior con una frecuencia menor del reloj de la CPU, tendremos que mantener el mismo programa que en el ejemplo anterior y modificar el archivo de encabezado *'BasysMX3.h'*.

Para conseguir una frecuencia de 40 MHz tenemos que modificar el divisor de salida FPLLODIV, de modo que quede de la siguiente manera:

```
#pragma config FPLLODIV = DIV_2
```

Como podremos observar en este ejemplo, la velocidad de parpadeo del LED se reduce a la mitad que, en el caso anterior. Como sería lógico pensar, ya que el periodo de actuación del microcontrolador aumenta al disminuir la frecuencia.

## 3.2.8 PROGRAMAR USANDO REGISTROS

En los microcontroladores PIC32, los pines de E/S se agrupan en puertos de E/S y se accede a ellos a través de registros del microcontrolador.

Hay siete puertos de E/S nombrados de A - G y cada uno tiene 16 bits de ancho, dependiendo del microcontrolador PIC32. Algunos de los puertos de E/S no están presentes y no todos los 16 bits son accesibles en todos los puertos de E/S.

Cada puerto de E/S tiene los siguientes **registros de control**: TRIS, LAT, PORT, ANSEL, CNPU, CNPD y ODC. Los registros para cada puerto de E/S se nombran antes de él: TRISX, LATX, PORTX, ANSELX, CNPUX, CNPDX y ODCX.

De modo que, si nos queremos referir al puerto A los registros serían: TRISA, LATA, PORTA, ANSELA, CNPUA.

Registro **TRIS**: se utiliza para establecer la dirección del pin. Establecer un bit con TRIS=0 hace que el pin sea de salida, mientras que hacer TRIS=1 hace que el pin sea de entrada.

Registro **LAT**: se utiliza para escribir en el puerto de E/S

Registro **PORT**: se usa para leer desde el puerto de E/S. La lectura desde el registro PORT devuelve el estado actual de todos los pines del puerto de E/S.

PIC 32 permiten que cualquier pin configurado como una salida se configure, como una salida normal o como una salida de drenaje abierto.

Registro **ODC**: se usa para controlar el tipo de salida, establecer un bit ODC = 0 implica que la salida es normal, mientras que ODC=1 indica que la salida es en drenaje abierto.

Los pines multifunción, que incluyen la funcionalidad de entrada analógica y que van a usarse como pines digitales, deben configurarse como tal. Estos están configurados por defecto como entrada analógica. El método para hacer esto es marcando el bit correspondiente del puerto con el registro **ANSEL** = 0. Estos pines incluirán ANX en las referencias de conexión de los pines dentro del PIC32.

Este microcontrolador tiene un Pull-Up y un Pull-Down conectados a cada pin. El registro **CNPU** (Pull-Up) =1 habilitado / =0 deshabilitado y el registro **CNPD** (Pull-Down) =1 habilitado / =0 deshabilitado. La configuración predeterminada es 0 (deshabilitados) para los dos registros.

## ❖ INSTRUCCIONES BÁSICAS EN 'C'

Cuando queremos controlar un solo bit asociado a un pin de E/S del PIC32, dentro de un determinado puerto, tenemos que hacerlo con una instrucción como la siguiente:

```
LATAbits.LATA3 = 1
```

Esta instrucción escribe en el puerto A, en el bit 3, el valor 1. Esto equivale a encender el LED nº 3.

Para indicar el número de bit dentro de un puerto lo podemos hacer de dos maneras: en binario o en hexadecimal.

En **binario** tenemos que indicarlo escribiendo *'0b',* seguido de los 16 bits correspondientes al tamaño del puerto, numerados de izquierda a derecha, desde el 15 hasta el 0. De este modo, si queremos hacer referencia a los bits que ocupan la posición 15, 11, 10, 9, 8, 3 y 0, debemos escribir *'0b1000111100001001'*

Mientas, para hacerlo en **hexadecimal** tenemos que escribir *'0x',* seguido de 4 dígitos, los correspondientes al valor en hexadecimal de los 16 bits del puerto agrupados de cuatro en cuatro. Por lo que, si queremos referirnos a los mismos bits del ejemplo anterior debemos escribir *'0x8F09'.*

A la hora de controlar un conjunto de bits dentro de un mismo puerto, nos son muy útiles las instrucciones: **INV**, **SET** y **CLR.** Estas instrucciones modifican las posiciones de los bits indicados de diferentes modos. La primera instrucción modifica los bits invirtiendo su valor lógico, la segunda los modifica poniéndolos a 1 y la última, poniéndolos a 0.

### 3.2.9 EJEMPLO USANDO REGISTROS

Para este ejemplo, en el que programaremos usando los registros, vamos a partir de lo explicado anteriormente.

Creamos un nuevo proyecto y cargamos el archivo de encabezado *'BasysMX3.h'* en el apartado *'Header Files',* pero en este caso ya no vamos a usar las librerías del fabricante.

#### ❖ ENCABEZADO

Lo que incluiremos en este apartado, además de las librerías que aparecen por defecto al crear la función principal, es el archivo de encabezado que creamos anteriormente *'BasysMX3.h',* donde está la configuración básica de nuestra placa y la estructura del programa principal.

```
#include "BasysMX3.h"
```

#### ❖ SETUP

Dentro de esta función lo que vamos a incluir son las **instrucciones de inicialización** para los LEDs. Indicaremos que los pines digitales correspondientes a los LEDs de nuestra placa Basys MX3 son de salida. Esto lo hacemos con el registro de control **TRIS** y podemos hacerlo de tres maneras diferentes: mencionando cada pin como un bit dentro del puerto correspondiente y con la instrucción SET con código binario o en hexadecimal.

Para saber qué bits de qué puertos están asociados a los ocho LEDs, tenemos que consultar la Tabla 2, donde podemos ver las características de los pines pertenecientes a los diodos LED. Podemos observar que los 8 LEDs se encuentran en el puerto A y en los bits de 0 al 7. Vamos a ver cómo sería la inicialización de los tres modos posibles ya que es el primer ejemplo y de aquí en adelante usaremos solo uno de ellos.

En el primer modo que vamos a ver, mencionaremos cada pin como un bit dentro del puerto correspondiente. damos a cada bit el valor 0 para indicar que es una salida digital.

```
TRISAbits.TRISA0 = 0; //Inicialización LED 0
TRISAbits.TRISA1 = 0; //Inicialización LED 1
TRISAbits.TRISA2 = 0; //Inicialización LED 2
TRISAbits.TRISA3 = 0; //Inicialización LED 3
TRISAbits.TRISA4 = 0; //Inicialización LED 4
TRISAbits.TRISA5 = 0; //Inicialización LED 5
TRISAbits.TRISA6 = 0; //Inicialización LED 6
TRISAbits.TRISA7 = 0; //Inicialización LED 7
```

El segundo modo será usando la instrucción **CLR** y la referencia a los bits del puerto en forma binaria. La instrucción CLR modifica el valor lógico del bit que se indica en la instrucción marcado como 1, poniéndolo a nivel lógico bajo.

```
TRISACLR = 0b0000000011111111; //Inicialización
LEDs 0-7
```

El ultimo modo es también con la instrucción CLR, pero poniendo la referencia a los bits del puerto en forma hexadecimal.

```
TRISACLR = 0x00FF; //Inicialización LEDs 0-7
```

### ❖ LOOP

Para poder escribir en los bits asociados a los LEDs e indicar si queremos encenderlos o apagarlos, tenemos que usar el registro de control **LAT**. Al igual que en el apartado anterior, podemos escribir las instrucciones de tres modos diferentes. Primero vamos a mostrar las instrucciones que corresponden a encender los LEDs y después las de apagarlos.

El primer modo que vamos a ver para encender los LEDs será mencionando cada pin como un bit dentro del puerto correspondiente. Damos a cada bit el valor 1 para indicar que queremos poner a nivel alto cada LED.

```
LATAbits.LATA0 = 1; //Encender LED 0
LATAbits.LATA1 = 1; //Encender LED 1
LATAbits.LATA2 = 1; //Encender LED 2
LATAbits.LATA3 = 1; //Encender LED 3
LATAbits.LATA4 = 1; //Encender LED 4
LATAbits.LATA5 = 1; //Encender LED 5
LATAbits.LATA6 = 1; //Encender LED 6
LATAbits.LATA7 = 1; //Encender LED 7
```

El segundo modo para encender será usando la instrucción SET y la referencia a los bits del puerto en forma binaria. La instrucción SET modifica el valor lógico del bit que se indica en la instrucción marcado como 1, poniéndolo a nivel lógico alto (encendido).

```
LATASET = 0b0000000011111111; //Encender LEDs 0-7
```

El ultimo modo para encender será también con la instrucción SET, pero poniendo la referencia a los bits del puerto en forma hexadecimal:

```
LATASET = 0x00FF; //Encender LEDs 0-7
```

A la hora de apagar los LEDs lo tenemos que hacer del mismo modo, pero ahora tenemos que hacer que los bits del puerto correspondientes a los LEDs tomen el valor lógico 0. Y al igual que antes podemos hacerlo de las mismas tres maneras.

El primer modo que vamos a ver será mencionando cada pin como un bit dentro del puerto correspondiente. Damos a cada bit el valor 0 para indicar que queremos poner a nivel bajo cada LED.

```
LATAbits.LATA0 = 0; //Apagar LED 0
LATAbits.LATA1 = 0; //Apagar LED 1
LATAbits.LATA2 = 0; //Apagar LED 2
LATAbits.LATA3 = 0; //Apagar LED 3
LATAbits.LATA4 = 0; //Apagar LED 4
LATAbits.LATA5 = 0; //Apagar LED 5
LATAbits.LATA6 = 0; //Apagar LED 6
LATAbits.LATA7 = 0; //Apagar LED 7
```

El segundo modo para apagar será usando la instrucción CLR y la referencia a los bits del puerto en forma binaria. La instrucción CLR modifica el valor lógico del bit que se indica en la instrucción marcado como 1 poniéndolo a nivel lógico bajo (apagado).

```
LATACLR = 0b0000000011111111; //Apagar LEDs 0-7
```

El ultimo modo para apagar será también con la instrucción SET, pero poniendo la referencia a los bits del puerto en forma hexadecimal:

```
LATACLR = 0x00FF; //Apagar LEDs 0-7
```

### ❖ PROGRAMA COMPLETO

Por último, tendríamos que juntar todas las instrucciones anteriores dentro de nuestro programa para conseguir que los 8 LEDs de la placa parpadeen. Entre las instrucciones de encendido y apagado tendremos que incluir un retardo para que el ojo humano pueda apreciar ese parpadeo.

Hemos explicado tres modos de escribir las mismas instrucciones, pero para nuestro ejemplo de programa completo escogeremos solo una. En este caso, escogeremos el método de introducir los datos en hexadecimal, ya que las instrucciones son más cortas:

```
#include <stdio.h>
#include <stdlib.h>
#include "BasysMX3.h"


void setup() {
    TRISACLR = 0x00FF; //Inicialización LEDs 0-7
}

void loop() {
    LATASET = 0x00FF; //Encender LEDs 0-7
    delay(1000);
    LATACLR = 0x00FF; //Apagar LEDs 0-7
    delay(1000);
}

void delay(int retardo) {
int i;
      for (i = 0; i < (retardo*1000); i++) {
      }
}
```

## 3.2.10 CONTROL DE VERSIONES

| VERSIÓN | FECHA | COMENTARIOS |
|---|---|---|
| V 1.0 | 15/02/2019 | Versión inicial |
| | | |

*Tabla 4. Control de versiones práctica 2*

## 3.3 LECTURA DE LA POSICIÓN DE LOS INTERRUPTORES DE LA PLACA BASYS MX3 [PRÁCTICA Nº 3]

### ❖ OBJETIVOS

El objetivo de la práctica nº 3 es conocer las instrucciones necesarias para poder leer la posición de los ocho interruptores, que incorpora la placa Basys MX3.

Para tener una mejor visión de lo expuesto durante esta práctica, lo comprobaremos mediante varios ejemplos en los que controlaremos el estado de los LEDs, que incorpora la placa Basys MX3, y que ya vimos cómo se utilizaban en la práctica nº 2, por medio de la posición de dichos interruptores.

Esto lo realizaremos mediante el uso de las librerías que proporciona el fabricante y el uso del lenguaje 'C' a bajo nivel.

### ❖ MATERIAL NECESARIO

#### HARDWARE

- Placa Basys MX3.
- Cable micro USB.
- Ordenador personal.

#### SOFTWARE

- Microchip MPLAB X IDE.
- Compilador MPLAB XC32.
- Librerías

### 3.3.1 DESCRIPCIÓN DE PERIFÉRICOS

#### ❖ INTERRUPTORES

La placa Basys MX3 incorpora ocho Interruptores, etiquetados como SW0 – SW7 en la placa y en los esquemas. Estos están conectados a ocho pines digitales de E/S del microcontrolador PIC32. La lectura de los Interruptores se realiza mediante el acceso básico a un pin Entrada/Salida y los posibles valores que podemos obtener son 0 o 1, dependiendo de si el interruptor está apagado o encendido respectivamente.

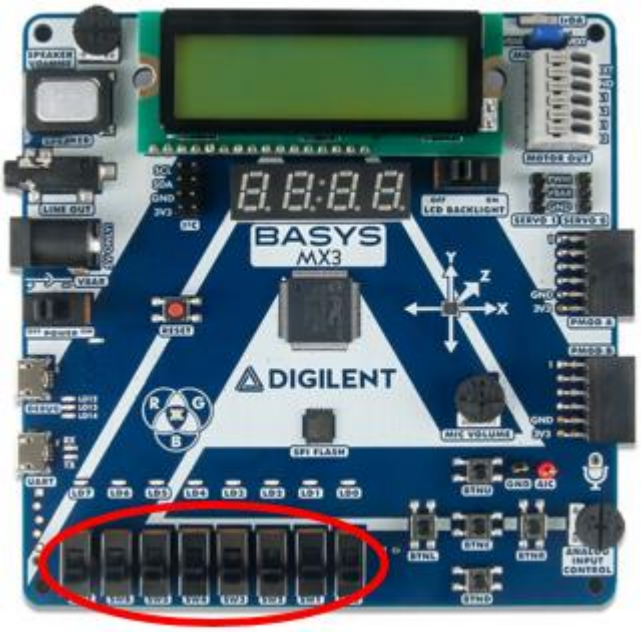



*Figura 54. Localización de los interruptores en la placa Basys MX3*

En la Figura 55 podemos observar de forma esquemática la conexión eléctrica interna de los interruptores dentro de la placa BASYS MX3.

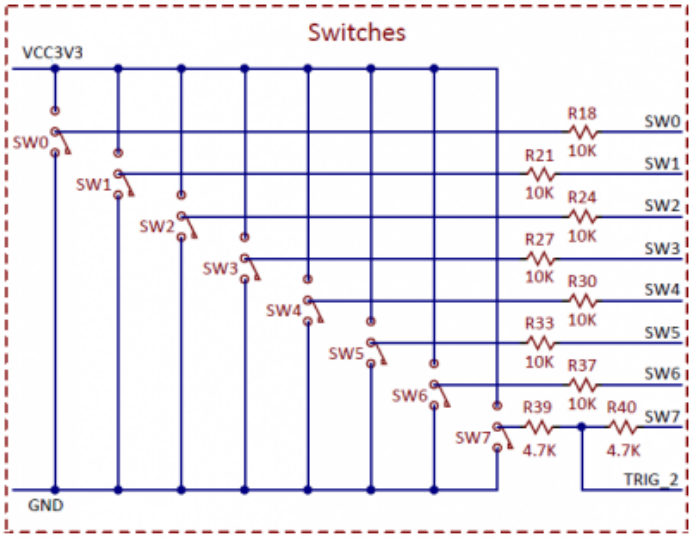



*Figura 55. Esquema eléctrico de los interruptores*

Como podemos ver en el esquema, los interruptores están conectados por un lado a una señal de 3,3 Voltios y por otra a GND. Esto hace que cuando el interruptor esté unido a GND, la señal que recibirá el pin correspondiente sea una señal lógica baja. Mientras que si lo está a la señal de 3.3 Voltios, recibirá una señal lógica alta.

Cada interruptor va acompañado de una resistencia limitadora para no sobrepasar la intensidad máxima soportada por el bloque de E/S del microcontrolador.

La Tabla 5 muestra todas las señales asociadas a los interruptores de la placa Basys MX3. En ella, podemos observar la etiqueta con la que aparecen en los esquemas y en la propia placa. Además, incluye las posibles funciones de cada pin dentro del microcontrolador PIC32.

| INTERRUPTOR | ETIQUETA | CARACTERÍSTICAS PIC32 |
|---|---|---|
| INTERRUPTOR 0 | SW0 | RPF3/RF3 |
| INTERRUPTOR 1 | SW1 | RPF5/PMA8/RF5 |
| INTERRUPTOR 2 | SW2 | RPF4/PMA9/RF4 |
| INTERRUPTOR 3 | SW3 | RPD15/RD15 |
| INTERRUPTOR 4 | SW4 | RPD14/RD14 |
| INTERRUPTOR 5 | SW5 | AN11/PMA12/RB11 |
| INTERRUPTOR 6 | SW6 | AN10/RPB10/PMA13/RB10 |
| INTERRUPTOR 7 | SW7 | AN9/RPB9/CTED4/RB9 |

*Tabla 5. Señales asociadas a los pines de los interruptores*

En lugar de utilizar los pines correspondientes a los interruptores como pines normales de Entrada, podemos asignar a cada pin del microcontrolador diferentes funciones para controlar otros periféricos. Esto se logra a través de la reasignación de cada pin.

### 3.3.2 PROGRAMAR USANDO LIBRERÍAS

Las funciones de librería, para poder usar los interruptores (SWT) de la placa Basys MX3, están contenidas en el paquete de librerías que proporciona el fabricante.

La librería SWT agrupa las funciones que acceden a las señales de los interruptores que incorpora nuestra placa. Los archivos que debemos incluir, a parte de los indicados en la práctica nº 2, en su correspondiente apartado del proyecto, para poder usar dichas funciones de librería son: **'swt.h'** y **'swt.c'**.

El archivo con extensión *'.h'* lo incluimos en el apartado *'Header Files'*, mientras que el archivo con extensión *'.c'* lo incluimos en el apartado *'Source Files'*.

Una vez incluidos todos los archivos necesarios de los correspondientes módulos que utilizaremos en nuestro proyecto, tendremos que modificar la función principal.

#### ❖ ENCABEZADO

Para usar las funciones de la librería SWT, tenemos que incluir el archivo de encabezado de la librería en la función principal, esto lo hacemos con la siguiente instrucción:

```
#include "swt.h"
```

#### ❖ SETUP

Dentro de esta función lo que tenemos que incluir es la **instrucción de inicialización** para los interruptores de nuestra placa. De modo que, con una sola instrucción y gracias a los archivos de librerías que hemos incluido anteriormente, indicaremos que los pines correspondientes a los interruptores son pines de entrada. Esto lo hacemos con la instrucción:

```
SWT_Init();
```

### ❖ LOOP

La instrucción que nos devuelve el estado del interruptor seleccionado y que, por tanto, debemos usar es: **SWT_GetValue (num_SWT)**. Donde *'num_SWT'* corresponde a la identificación del interruptor cuyo valor deseamos leer, la identificación es mediante números, tal y como muestra la Tabla 6.

| INTERRUPTOR | NÚMERO |
|---|---|
| SW0 | 0 |
| SW1 | 1 |
| SW2 | 2 |
| SW3 | 3 |
| SW4 | 4 |
| SW5 | 5 |
| SW6 | 6 |
| SW7 | 7 |

*Tabla 6. Identificación de los interruptores*

La lectura de cualquier interruptor nos devuelve el valor de 1 o 0 según sea su posición. Si la identificación que le damos no corresponde a ninguna de las mostradas en la tabla, la función nos devuelve el valor 0xFF.

Por ejemplo, si queremos asignar el valor del interruptor número 3 a la variable *'Val'* tendremos que escribir:

```
Val = SWT_GetValue (3);
```

## 3.3.3 EJEMPLO USANDO LIBRERÍAS

Para tener un ejemplo más completo de lo expuesto anteriormente, mostraremos un programa en el que controlaremos el estado de los LEDs, mediante las entradas proporcionadas por los interruptores de la placa Basys MX3.

La función de este programa es controlar el estado de cada LED, de modo que se encienda o apague, dependiendo de la posición de su respectivo interruptor. Es decir, asignaremos a cada interruptor el LED de igual numeración.

Este programa inicializa los módulos que vamos a usar de la placa y después asigna a cada LED el estado leído de su interruptor correspondiente.

```
#include <stdio.h>
#include <stdlib.h>
#include "config.h"
#include "led.h"
#include "swt.h"
#include "BasysMX3.h"

void setup() {
    //INICIALIZACIÓN LEDs COMO SALIDA
    LED_Init();
    //INICIALIZACIÓN INTERRUPTORES COMO ENTRADA
    SWT_Init();
}

void loop() {
    //ENCENDER LED SEGÚN EL ESTADO DEL INTERRUPTOR
    LED_SetValue(0, SWT_GetValue(0));
    LED_SetValue(1, SWT_GetValue(1));
    LED_SetValue(2, SWT_GetValue(2));
    LED_SetValue(3, SWT_GetValue(3));
    LED_SetValue(4, SWT_GetValue(4));
    LED_SetValue(5, SWT_GetValue(5));
    LED_SetValue(6, SWT_GetValue(6));
    LED_SetValue(7, SWT_GetValue(7));
}
```

### 3.3.4 PROGRAMAR USANDO REGISTROS

En este apartado vamos a programar usando registros a partir de lo explicado en la práctica nº 2, donde creamos un proyecto nuevo y cargamos el archivo de encabezado *'BasysMX3.h'* en el apartado *'Header Files'*, pero en este caso ya no necesitaremos las librerías del fabricante.

Ya no explicaremos todas las posibles maneras de escribir nuestro programa, como hicimos en la práctica nº 2, sino que nos centraremos en una, la hexadecimal.

### ❖ ENCABEZADO

Además de las librerías que aparecen por defecto, al crea la función principal tenemos que incluir el archivo de encabezado que hemos cargado antes, donde está la configuración básica de nuestra placa y la estructura del programa principal.

```
#include "BasysMX3.h"
```

### ❖ SETUP

Dentro de esta función lo que vamos a incluir son las **instrucciones de inicialización** para los interruptores en la forma hexadecimal. Indicaremos que los pines digitales correspondientes a los interruptores de nuestra placa Basys MX3 son de entrada.

Para saber que bits de qué puertos están asociados a los ocho interruptores tenemos que consultar la Tabla 5, donde vienen las características de los pines pertenecientes a los interruptores.

Podemos observar que los ocho interruptores se encuentran en tres puertos diferentes, por lo que tendremos que escribir tres instrucciones diferentes una por cada puerto:

```
TRISFSET = 0x0038; //Bits del puerto F
TRISDSET = 0xC000; //Bits del puerto D
TRISBSET = 0x0E00; //Bits del puerto B
```

En la práctica nº 2 vimos que los pines, que incluyen la funcionalidad de entrada analógica, y van a usarse como pines digitales, como es en este caso, deben ser configurados para ello, puesto que por defecto están configurados como entrada analógica.

Ya que es el primer ejemplo en el que nos aparecen pines analógicos, vamos a mostrar cómo sería el código, que tendríamos que escribir, de las tres formas posibles que ya hemos visto en la práctica nº 2.

Los pines que tienen asociada una entrada analógica son los que aparecen como **ANX** en las tablas de características de los pines. En este caso, consultaremos la Tabla 5 y podemos observar que los pines con entradas analógicas son los correspondientes a los bits 9, 10 y 11 del puerto B.

El primer modo que vamos a ver será mencionando cada pin como **un bit dentro del puerto** correspondiente. Damos a cada bit el valor 0 para desactivar la entrada analógica.

```
ANSELBbits.ANSB11 = 0; //Bit 11 (SW5)
ANSELBbits.ANSB10 = 0; //Bit 10 (SW6)
ANSELBbits.ANSB9  = 0; //Bit 9  (SW7)
```

El segundo modo será usando la instrucción **CLR** y la referencia a los bits del puerto en **forma binaria** La instrucción CLR modifica el valor lógico del bit, que se indica en la instrucción marcado como *'1'*, poniéndolo a nivel lógico bajo.

```
ANSELBCLR = 0b0000111000000000;//Bits 9, 10 y 11
del puerto B
```

El ultimo modo será también con la instrucción **CLR**, pero poniendo la referencia a los bits del puerto en **forma hexadecimal**:

```
ANSELBCLR = 0x0E00; //Bits 9, 10 y 11 puerto B
```

### ❖ LOOP

Para poder leer los bits asociados a los interruptores y después poder asignarlos a alguna variable, tendremos que hacerlo mediante el registro **PORT**.

La lectura de cualquier interruptor nos devuelve un valor de 0 o 1 según sea su posición.

Por ejemplo, si queremos asignar el valor del interruptor nº 3 a la variable *"Val"* tendremos que escribir:

```
Val = PORTDbits.RD15; //Leer SW3;
```

### 3.3.5 EJEMPLO USANDO REGISTROS

Para tener un ejemplo más completo de lo expuesto anteriormente, vamos a mostrar un programa en el que se encenderán los LEDs de la placa Basys MX3 según sea la posición de los interruptores asociados.

```
#include <stdio.h>
#include <stdlib.h>
#include "BasysMX3.h"

void setup() {
    //INICIALIZACIÓN LEDs COMO SALIDA
    TRISACLR = 0x00FF; //Pines del puerto A

    //INICIALIZACIÓN INTERRUPTORES COMO ENTRADA
    TRISFSET = 0x0038; //Pines del puerto F
    TRISDSET = 0xC000; //Pines del puerto D
    TRISBSET = 0x0E00; //Pines del puerto B

    //DESACTIVAR ENTRADAS ANALÓGICAS
    ANSELBCLR = 0x0E00; //Pines del puerto B
}

void loop() {
    //ENCENDER LED SEGÚN EL ESTADO DEL INTERRUPTOR
    LATAbits.LATA0 = PORTFbits.RF3; //LED0-SW0;
    LATAbits.LATA1 = PORTFbits.RF5; //LED1-SW1;
    LATAbits.LATA2 = PORTFbits.RF4; //LED2-SW2;
    LATAbits.LATA3 = PORTDbits.RD15;//LED3-SW3;
    LATAbits.LATA4 = PORTDbits.RD14;//LED4-SW4;
    LATAbits.LATA5 = PORTBbits.RB11;//LED5-SW5;
    LATAbits.LATA6 = PORTBbits.RB10;//LED6-SW6;
    LATAbits.LATA7 = PORTBbits.RB9; //LED7-SW7;
}
```

### 3.3.6 CONTROL DE VERSIONES

| VERSIÓN | FECHA | COMENTARIOS |
|---|---|---|
| V 1.0 | 1/03/2018 | Versión inicial |
| | | |

*Tabla 7. Control de versiones práctica 3*

## 3.4 VARIAR EL COLOR DEL LED RGB MEDIANTE LOS BOTONES DE LA PLACA BASYS MX3 [PRÁCTICA Nº 4]

### ❖ OBJETIVOS

El objetivo de la práctica nº 4 es conocer las instrucciones necesarias para encender y obtener cualquier color del LED RGB, y las instrucciones para detectar si pulsamos cualquiera de los cinco botones que incorpora la placa Basys MX3.

Para tener una mejor visión de lo expuesto durante esta práctica, lo probaremos mediante varios ejemplos en los que controlaremos el color del LED RGB mediante la pulsación de los botones de la placa Basys MX3.

Esto lo realizaremos a través del uso de las librerías que nos proporciona el fabricante y el uso del lenguaje 'C' a bajo nivel.

### ❖ MATERIAL NECESARIO

#### HARDWARE

- Placa Basys MX3.
- Cable micro USB.
- Ordenador personal.

#### SOFTWARE

- Microchip MPLAB X IDE.
- Compilador MPLAB XC32.
- Librerías

### 3.4.1. DESCRIPCIÓN DE PERIFÉRICOS

#### ❖ BOTONES

La placa Basys MX3 incorpora cinco botones, etiquetados tanto en la placa como en los esquemas como BTNU, BTNL, BTNC, BTNR y BTND. Estos están conectados a cinco pines digitales de E/S del PIC32. La lectura de los botones se realiza mediante el acceso básico a un pin E/S. Los posibles valores que podemos obtener de los botones son 1 o 0, dependiendo si el botón está pulsado o si no lo está.

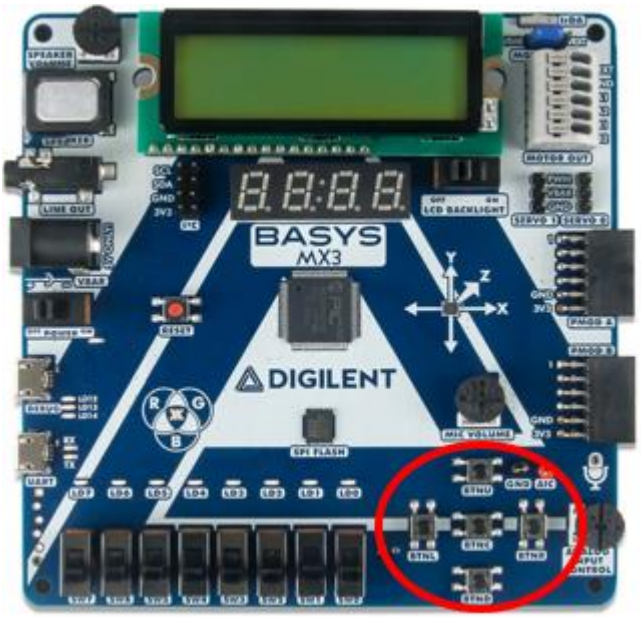



*Figura 56. Localización de los 5 botones de la placa Basys MX3*

La Figura 57 muestra de forma esquemática el modo en el que los botones están conectados eléctricamente en la placa BASYS MX3.

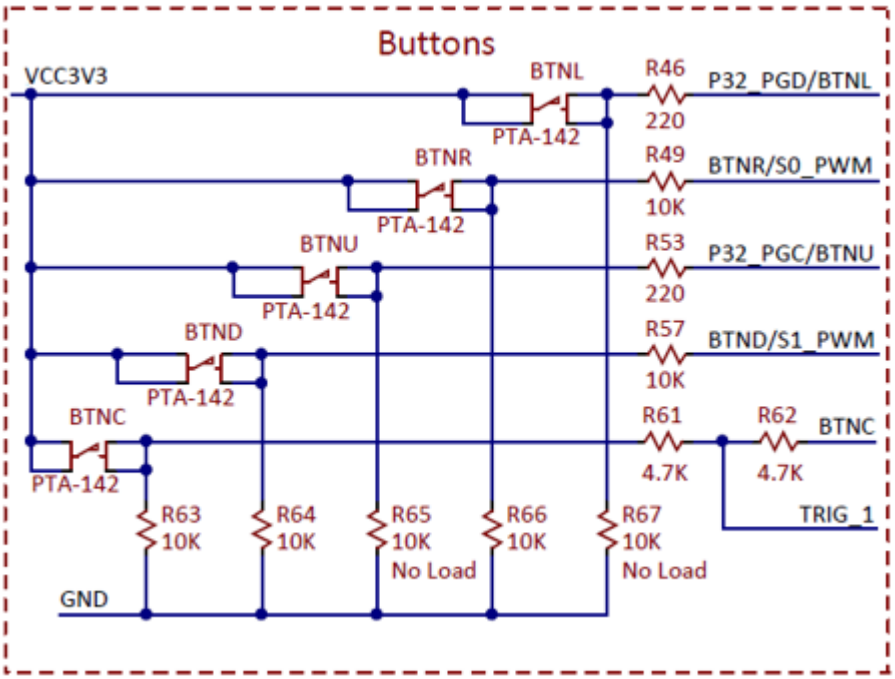



*Figura 57. Esquema electico de los botones*

Como podemos observar en el esquema eléctrico, el montaje de la alimentación de 3,3 Voltios, los botones y las resistencias es en Pull-Down. Esto provoca que, si el interruptor se encuentra cerrado, la corriente se moverá hacia el pin, dejándolo en un valor lógico alto. Mientras que si está en reposo la corriente se moverá hacia la resistencia dejando el pin con un valor lógico bajo.

La Tabla 8 muestra la etiqueta que lleva cada botón en la placa y en los esquemas. También las posibles funciones de cada pin dentro del microcontrolador PIC32.

| BOTÓN | ETIQUETA | CARACTERÍSTICAS PIC32 |
|---|---|---|
| BOTÓN ARRIBA | BTNU | AN1/RPB1/CTED12/RB1 |
| BOTÓN IZQUIERDA | BTNL | AN0/RPB0/RB0 |
| BOTÓN CENTRO | BTNC | RPF0/PMD11/RF0 |
| BOTÓN DERECHA | BTNR | AN8/RPB8/CTED10/RB8 |
| BOTÓN ABAJO | BTND | RPA15/RA15 |

*Tabla 8. Señales asociadas a los pines de los botones*

En lugar de utilizar los pines correspondientes a los botones como pines normales de Entrada, podemos asignar a cada pin del microcontrolador diferentes funciones para controlar otros periféricos. Esto se logra a través de la reasignación de cada pin.

### ❖ LED RGB

La placa Basys MX3 incorpora un diodo LED tricolor (RGB). El LED permite al usuario obtener cualquier color mediante la configuración de los diferentes componentes (rojo, verde y azul).

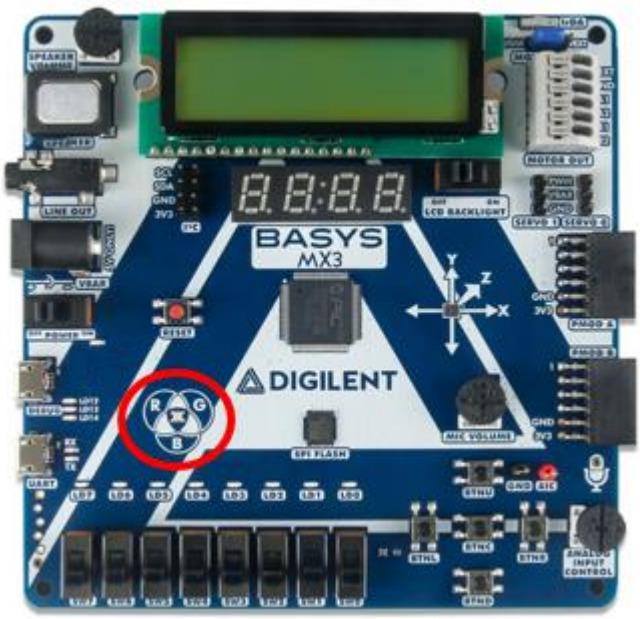



*Figura 58. Localización del LED RGB en la placa Basys MX3*

En la Figura 60 podemos observar de forma esquemática la conexión eléctrica interna del LED RGB dentro de la placa BASYS MX3.

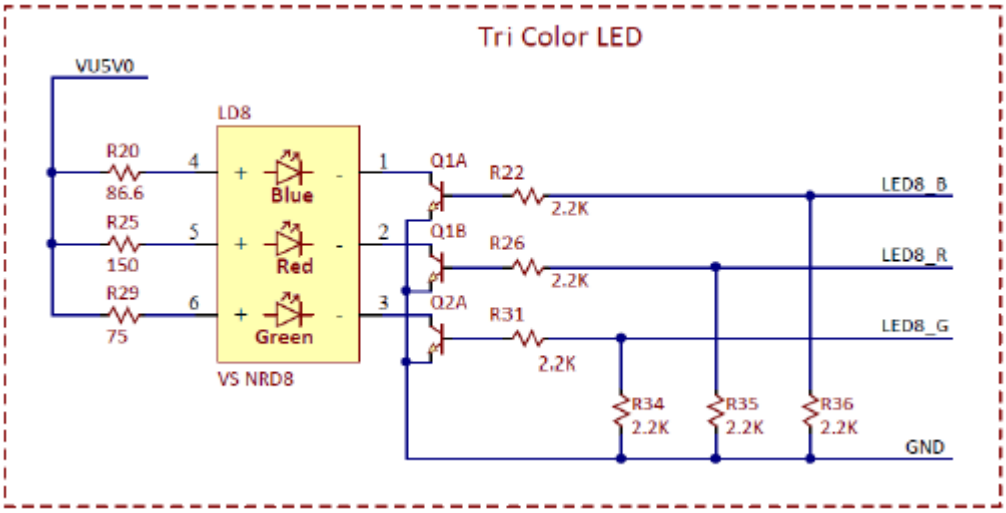



*Figura 59. Esquema eléctrico del LED RGB*

Como podemos ver en el esquema, el LED tiene una configuración de ánodo común y cada LED va acompañado de una resistencia limitadora diferente. Cada LED tiene unas características eléctricas distintas a las de los otros, así ninguno sobrepasará la intensidad máxima que soporta, ya que todos están conectados a una misma tensión de 5 Voltios.

Para controlar el LED RGB lo hacemos del mismo modo que para controlar tres LEDs separados, uno por cada color. Es decir, hay una señal digital para controlar cada componente de color. El uso de valores 0 o 1 para estas señales solo nos dará un número limitado de colores (el color del led, la mezcla con los otros dos y la combinación de los tres), por lo que la mayoría de veces esto no es suficiente. La solución es enviar una secuencia de valores 1 y 0 en estas líneas digitales con una frecuencia superior a la percepción humana, ya que el ojo humano integrará los valores de iluminación primaria en la sensación del color final.

El enfoque más utilizado para resolver este problema es el uso de señales PWM (modulación de ancho de pulso) aunque existen otros, como es el PDM (modulación de densidad de pulso). En nuestro caso, nos centraremos en el primer supuesto, solo con entradas digitales simples.

La Tabla 9 muestra todas las señales asociadas a cada uno de los colores del LED RGB. En ella, podemos observar la etiqueta con la que aparecen en los esquemas y las posibles funciones de cada pin dentro del microcontrolador PIC32.

| LED RGB | ETIQUETA | CARACTERÍSTICAS PIC32 |
|---|---|---|
| LED ROJO | LED8_R | AN25/RPD2/RD2 |
| LED VERDE | LED8_G | RPD12/PMD12/RD12 |
| LED AZUL | LED8_B | AN26/RPD3/RD3 |

*Tabla 9. Señales asociadas a los pines del LED RGB*

En lugar de utilizar los pines correspondientes al LED RGB como pines normales de Salida, podemos asignar a cada pin del microcontrolador diferentes funciones para controlar otros periféricos. Esto se logra a través de la reasignación de cada pin.

### 3.4.2 LIBRERÍAS Y FUNCIONES

Las funciones de librería para usar los botones (BTN) y el diodo LED RGB (RGBLED) están contenidas en el paquete de librerías que proporciona el fabricante.

La librería BTN agrupa las funciones que acceden a las señales de los botones, simplemente realiza una entrada digital básica, no se proporciona ninguna funcionalidad para evitar posibles rebotes. Esto provoca que leamos varias activaciones/desactivaciones del pulsador, cuando solo se ha producido una pulsación. Este problema lo trataremos más adelante.

Los archivos que debemos incluir, en su correspondiente apartado del proyecto, para poder usar las funciones de la librería son: ***'btn.h'*** y ***'btn.c'***.

La librería RGBLED agrupa las funciones que acceden a las señales del diodo LED RGB. La placa Basys MX3 proporciona tres señales digitales para controlar el LED, una para cada color básico. El uso de valores 0 o 1 en estas señales solo nos dará un número limitado de colores, la solución es mandar una secuencia de valores 1 y 0 por estas tres señales digitales para obtener un mayor rango de colores. La librería RGBLED utiliza una modulación de densidad de pulso (PDM).

Los archivos que debemos incluir, en su correspondiente apartado del proyecto, para poder usar las funciones de la librería son: ***'rgbled.h'*** y ***'rgbled.c'***.

Además de las librerías correspondientes a los módulos periféricos, en este caso vamos a incluir la **librería UTILS**. Implementa la funcionalidad para generar un retraso (delay), que se implementa utilizando un bucle, por lo que el tiempo de retraso no es exacto.

Los archivos que debemos incluir, en su correspondiente apartado del proyecto, para poder usar las funciones de esta librería son: ***'utils.h'*** y ***'utils.c'***.

Los archivos con extensión *'.h'* los incluimos en el apartado *'Header Files',* mientras que el archivo con extensión *'.c'* los incluimos en el apartado *'Source Files'*.

Una vez incluidos todos los archivos de los correspondientes módulos que utilizaremos en nuestro proyecto, tendremos que modificar la función principal.

### ❖ ENCABEZADO

Para usar las funciones de la librería BTN, de la librería RGBLED y de la librería UTILS, tenemos que incluir los archivos de encabezado de las librerías en la función principal, con las respectivas instrucciones:

```
#include "btn.h"
#include "rgbled.h"
#include "utils.h"
```

### ❖ SETUP

Dentro de esta función lo que vamos a incluir son las **instrucciones de inicialización** para los botones y el LED RGB de nuestra placa. Con una sola instrucción, gracias a los archivos de librería que hemos incluido anteriormente, indicaremos que los pines correspondientes a los botones son pines digitales de entrada y los del LED RGB corresponden a salidas digitales. Esto lo hacemos con las siguientes instrucciones respectivamente.

```
BTN_Init();
RGBLED_Init();
```

### ❖ LOOP

La instrucción que tenemos que usar para saber si un botón está presionado o no lo está es: **BTN_GetValue (Botón).** *'Botón'* corresponde a la identificación del botón cuyo valor deseamos leer. La identificación puede ser mediante números o letras, tal y como muestra la Tabla 10.

| BOTÓN | NÚMERO | LETRAS |
|---|---|---|
| BTNU | 0 | 'U', 'u' |
| BTNL | 1 | 'L', 'l' |
| BTNC | 2 | 'C', 'c' |
| BTNR | 3 | 'R', 'r' |
| BTND | 4 | 'D', 'd' |

*Tabla 10. Opciones de identificación de los botones*

La lectura de cualquier botón nos devuelve el valor 1 o 0, según este presionado o no lo esté. Si la identificación que le damos no corresponde a ninguna de las mostradas en la tabla, la función nos devuelve el valor 0xFF.

Por ejemplo, si queremos asignar el valor del botón BTNR a la variable *'Val'* tendremos que escribir:

```
Val = BTN_GetValue (3);
```

Para poder encender el diodo LED RGB tendremos que usar la instrucción: **RGBLED_SetValue (ValR, ValG, ValB).** Esta función establece el color del diodo por medio de la mezcla de los tres colores básicos, donde *'ValR'*, *'ValG'* y *'ValB'* corresponde a los valores de 8 bits de los 3 componentes: rojo, verde y azul respectivamente.

Por ejemplo, si queremos obtener el color blanco con el máximo brillo tendremos que indicar el máximo de cada uno de los tres componentes, es decir, 0b11111111 en binario, 0xFF en hexadecimal o 255 en decimal.

```
RGBLED_SetValue (0xFF, 0xFF, 0xFF);
```

Para introducir un retraso o delay en nuestro programa tenemos que hacerlo mediante la instrucción: **DelayAprox10us (microsegundos).** Esta función retrasa la ejecución del programa por el número especificado de microsegundos.

Por ejemplo, si queremos introducir un retraso de 100 microsegundos tendríamos que escribir:

```
DelayAprox10Us (100); //Delay de 100us
```

### 3.4.3 EJEMPLO USANDO LIBRERÍAS

Para tener un ejemplo más completo de lo expuesto anteriormente, veremos un programa en el que vamos a controlar el estado de los tres colores del LED RGB mediante tres pulsadores de la placa Basys MX3.

Lo que vamos a hacer es controlar el estado de cada componente del LED RGB por medio de un botón, es decir el botón BTNL encenderá y apagará el componente rojo del LED, el botón BTNC el verde y el BTNR el azul. Tendremos que añadir una serie de variables auxiliares para conseguir un efecto memoria que controle el estado del pulsador y el de cada color.

La intensidad del LED de cada componente lo pondremos bajo (5 de 255) para conseguir ver cada color sin que nos deslumbre.

Ya que la propia librería no proporciona ninguna funcionalidad, para evitar rebotes incluiremos un delay en el programa. Así lograremos que se evite cualquier posible rebote provocado al accionar el mismo botón de forma repetida.

La función del delay es introducir un pequeño retardo durante la ejecución del programa, de modo que una vez detectado el primer pulso, el controlador no detecte ese posible rebote.

Este programa lo que hace es inicializar los módulos que vamos a usar de la placa, introducir una serie de variables auxiliares que usaremos en el programa, asignar a las variables (Actual_BTNL, Actual_BTNC y Actual_BTNR) el valor actual de los botones, y mediante unos bucles, controlar el estado de los tres colores. Por ultimo encenderemos el LED RGB y actualizaremos el valor de las variables auxiliares.

```
#include <stdio.h>
#include <stdlib.h>
#include "config.h"
#include "utils.h"
#include "btn.h"
#include "rgbled.h"
#include "BasysMX3.h"

//INICIALIZAMOS LAS VARIABLES AUXILIARES
int Rojo,  Estado_Rojo   = 0;
int Verde, Estado_Verde = 0;
int Azul,  Estado_Azul   = 0
int Antes_BTNL = 0, Actual_BTNL = 0;
int Antes_BTNC = 0, Actual_BTNC = 0;
int Antes_BTNR = 0, Actual_BTNR = 0;

void setup() {
    //INICIALIZACIÓN BOTONES COMO ENTRADA DIGITAL
    BTN_Init();
    //INICIALIZACIÓN LED_RGB COMO SALIDA DIGITAL
    RGBLED_Init();
}

void loop() {
    //ASIGNAMOS VARIABLES AL ESTADO DE LOS BOTONES
    Actual_BTNL = BTN_GetValue(1);//Leer BTNL
    Actual_BTNC = BTN_GetValue(2);//Leer BTNC
    Actual_BTNR = BTN_GetValue(3);//Leer BTNR

    //BUCLE CONTROL COLOR ROJO DEL LED_RGB
    if ((Antes_BTNL == 0) && (Actual_BTNL == 1)) {
        Estado_Rojo = !Estado_Rojo;
        if (Estado_Rojo == 1) {
            Rojo = 5;
        } else if (Estado_Rojo == 0) {
            Rojo = 0;
        }
    }
```

```
        //BUCLE CONTROL COLOR VERDE DEL LED_RGB
        if ((Antes_BTNC == 0) && (Actual_BTNC == 1)) {
            Estado_Verde = !Estado_Verde;
            if (Estado_Verde == 1) {
                Verde = 5;
            } else if (Estado_Verde == 0) {
                Verde = 0;
            }
        }


        //BUCLE CONTROL COLOR AZUL DEL LED_RGB
        if ((Antes_BTNR == 0) && (Actual_BTNR == 1)) {
            Estado_Azul = !Estado_Azul;
            if (Estado_Azul == 1) {
                Azul = 5;
            } else if (Estado_Azul == 0) {
                Azul = 0;
            }
        }
        //ACTUALIZAMOS EL ESTADO DE LOS PULSADORES
        Antes_BTNL = Actual_BTNL;
        Antes_BTNC = Actual_BTNC;
        Antes_BTNR = Actual_BTNR;

        //ENCENDEMOS EL LED_RGB
        RGBLED_SetValue(Rojo, Verde, Azul);
        DelayAprox10Us(50);//Delay de 50us
    }
```

### 3.4.4 PROGRAMAR USANDO REGISTROS

En este apartado vamos a programar usando registros a partir de lo explicado en la práctica nº2, donde creamos un proyecto nuevo y cargamos el archivo de encabezado *'BasysMX3.h'* en el apartado *'Header Files',* pero en este caso ya no necesitaremos las librerías del fabricante.

Ya no explicaremos todas las posibles maneras de escribir nuestro programa, como hicimos en la práctica nº2, si no que nos centraremos en una, la hexadecimal.

### ❖ ENCABEZADO

Además de las librerías que aparecen por defecto, al crear la función principal, tenemos que incluir el archivo de encabezado que hemos cargado antes, donde está la configuración básica de nuestra placa y la estructura del programa principal.

```
#include "BasysMX3.h"
```

### ❖ SETUP

Dentro de esta función lo que vamos a incluir son las **instrucciones de inicialización** para los botones y el LED RGB en la forma hexadecimal. Indicaremos que los pines digitales correspondientes a los botones de nuestra placa Basys MX3 son de entrada, mientras que los del LED RGB son de salida.

Para saber que bits de qué puertos están asociados a los cinco botones y a los tres componentes del LED RGB tenemos que consultar la Tabla 8, donde vienen las características de los pines pertenecientes a los botones, y la Tabla 9 dónde vienen las características de los pines pertenecientes al LED RGB.

Podemos observar que los cinco botones se encuentran en tres puertos diferentes, por lo que tendremos que escribir tres instrucciones diferentes una por cada puerto.

```
TRISBSET = 0x0103; //Bits de entrada puerto B
TRISFSET = 0x0001; //Bits de entrada puerto F
TRISASET = 0x8000; //Bits de entrada puerto A
```

En la práctica 2 vimos que los pines que incluyen la funcionalidad de entrada analógica y van a usarse como pines digitales, como es este caso, deben ser configurados para ello. Ya que, por defecto están configurados como entrada analógica. En este caso, son los pines correspondientes al puerto B.

```
ANSELBCLR = 0x0103; //deshabilitar entradas
analógicas puerto B
```

Los pines del LED RGB se encuentra en el puerto D y no todos tienen asociado la funcionalidad de entrada analógica, por lo que tendremos que escribir:

```
TRISDCLR = 0x100C; //bits de entrada puerto D
ANSELDCLR = 0x000C; //deshabilitar entradas
analógicas puerto D
```

### ❖ LOOP

Para poder leer los bits asociados a los botones, y después poder asignarlos a alguna variable, tendremos que hacerlo mediante el registro **PORT**.

La lectura de cualquier botón nos devuelve un valor de 0 o 1, según esté presionado o no.

Por ejemplo, si queremos asignar el valor del botón BTNU a la variable *'BTNU'*, tendremos que escribir:

```
BTNU = PORTBbits.RB1; //Leer BTNU
```

La escritura del LED RGB se hace del mismo modo que para controlar tres LEDs separados, uno por cada color. Esto lo tendremos que hacer mediante el registro **LAT.**

Escribir un 1 o un 0, en cualquiera de los bits asociados a los tres colores del LED RGB, corresponde a encender o apagar ese color.

Por ejemplo, si queremos encender el LED rojo tenemos que poner a 1 el bit correspondiente a ese color, para ello tendríamos que escribir:

```
LATDbits.LATD2 = 1;//Encender LED rojo
```

## 3.4.5 EJEMPLO USANDO REGISTROS

Para tener un ejemplo más completo de lo expuesto anteriormente, vamos a mostrar un programa que realizara las mismas funciones, que en el ejemplo anterior usando librerías, pero ahora no las usaremos.

```
#include <stdio.h>
#include <stdlib.h>
#include "BasysMX3.h"

// INICIALIZAMOS LAS VARIABLES AUXILIARES
int Ant_BTNL = 0, Ant_BTNC = 0, Ant_BTNR = 0;
int Act_BTNL = 0, Act_BTNC = 0, Act_BTNR = 0;
int Est_Rojo = 0, Est_Verde = 0, Est_Azul = 0;
```

```
//FUNCIÓN PARA INTRODUCIR UN DELAY
void delay(int retardo) {
    int i;
    for (i = 0; i < (retardo * 1000); i++) {
    }
}


void setup() {
    //INICIALIZACIÓN BOTONES COMO ENTRADA DIGITAL
    TRISBSET = 0x0103; //Bits de entrada puerto B
    ANSELBCLR = 0x0103; //Deshabilitar analógicas
    TRISFSET = 0x0001; //Bits de entrada puerto F
    TRISASET = 0x8000; //Bits de entrada puerto A

    //INICIALIZACIÓN LED_RGB COMO SALIDA DIGITAL
    TRISDCLR = 0x100C; //Bits de salida puerto D
    ANSELDCLR = 0x000C; //Deshabilitar analógicas
}


void loop() {

    //ASIGNAMOS VARIABLES AL ESTADO DE LOS BOTONES
    Act_BTNL = PORTBbits.RB0; //Leer BTNL
    Act_BTNC = PORTFbits.RF0; //Leer BTNC
    Act_BTNR = PORTBbits.RB8; //Leer BTNR

    //BUCLE CONTROL COLOR ROJO DEL LED_RGB
    if ((Ant_BTNL == 0) && (Act_BTNL == 1)) {
        Est_Rojo = !Est_Rojo;
    }

    //BUCLE CONTROL COLOR VERDE DEL LED_RGB
    if ((Ant_BTNC == 0) && (Act_BTNC == 1)) {
        Est_Verde = !Est_Verde;
    }

    //BUCLE CONTROL COLOR AZUL DEL LED_RGB
    if ((Ant_BTNR == 0) && (Act_BTNR == 1)) {
        Est_Azul = !Est_Azul;
    }
```

```
        //ACTUALIZAMOS EL ESTADO DE LOS PULSADORES
        Ant_BTNL = Act_BTNL;
        Ant_BTNC = Act_BTNC;
        Ant_BTNR = Act_BTNR;

        //ENCENDEMOS EL LED_RGB
        LATDbits.LATD2  = Est_Rojo;
        LATDbits.LATD12 = Est_Verde;
        LATDbits.LATD3  = Est_Azul;
        delay(50);
    }
```

### 3.4.6 CONTROL DE VERSIONES

| VERSIÓN | FECHA | COMENTARIOS |
|---|---|---|
| V 1.0 | 22/03/2019 | Versión inicial |
| | | |

*Tabla 11. Control de versiones práctica 4*

# 3.5 CONTROLAR LOS LEDS DEL PMOD 8LD CONECTÁNDOLO A LA PLACA BASYS MX3 [PRÁCTICA Nº 5]

## ❖ OBJETIVOS

El objetivo de la práctica nº 5 es conocer que es un Pmod, la ubicación de los conectores Pmod en la placa Basys MX3 y conocer las instrucciones necesarias para controlar cualquier Pmod que se pueda conectar a la placa. El Pmod que conectemos podrá tener pines digitales de entrada o de salida, en nuestro caso, conectaremos el Pmod 8LD que dispone de ocho pines digitales de salida.

Para tener una mejor visión de lo expuesto durante esta práctica lo comprobaremos mediante varios ejemplos, en los que mostraremos diferentes formas de encender los diodos LEDs que incorpora el Pmod 8LD. Las señales de activación las introduciremos por medio de los interruptores que incorpora la placa Basys MX3 y que ya vimos cómo se utilizaban en la práctica nº 3.

Esto lo realizaremos mediante el uso de las librerías que nos proporciona el fabricante y el uso del lenguaje 'C' a bajo nivel.

## ❖ MATERIAL NECESARIO

### HARDWARE

- Placa Basys MX3
- Pmod 8LD rev. B
- Cable micro USB
- Ordenador personal

### SOFTWARE

- Microchip MPLAB X IDE
- Compilador MPLAB XC32
- Librerías

### 3.5.1 DESCRIPCIÓN DE PERIFÉRICOS

Los módulos periféricos Pmod son pequeñas placas auxiliares que ofrecen la posibilidad de ampliar las capacidades de nuestra placa Basys MX3. Estos pueden disponer de entradas o salidas, que pueden ser digitales o analógicas.

Los módulos Pmod se comunican con nuestra placa utilizando conectores de 6, 8 o 12 pines, que pueden transmitir múltiples señales de control digital, incluidos SPI y otros protocolos en serie. Los módulos Pmod permiten diseños más efectivos al poder disponer de señales analógicas o fuentes de alimentación solo donde se necesitan y lejos de nuestra placa.

#### ❖ CONECTORES PMOD

La placa Basys MX3 incorpora dos conectores para Pmods etiquetados en la placa como PMOD A y PMOD B, son conectores de cabezal hembra en ángulo recto y de 2x6 pines. Las señales correspondientes que van a los conectores Pmod están conectadas a pines de Entrada/Salida del microcontrolador PIC32 de la placa. Los Pmods se pueden conectar directamente a los conectores de la placa o se pueden conectar mediante cables.

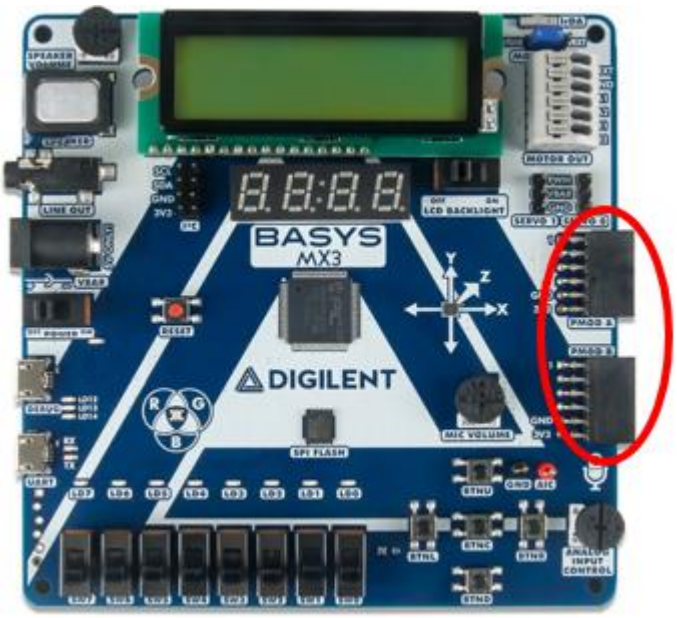



*Figura 60. Localización de los conectores Pmod en la placa Basys MX3*

En la Figura 62 podemos observar de forma esquemática la conexión eléctrica interna de cada PMOD dentro de la placa BASYS MX3.

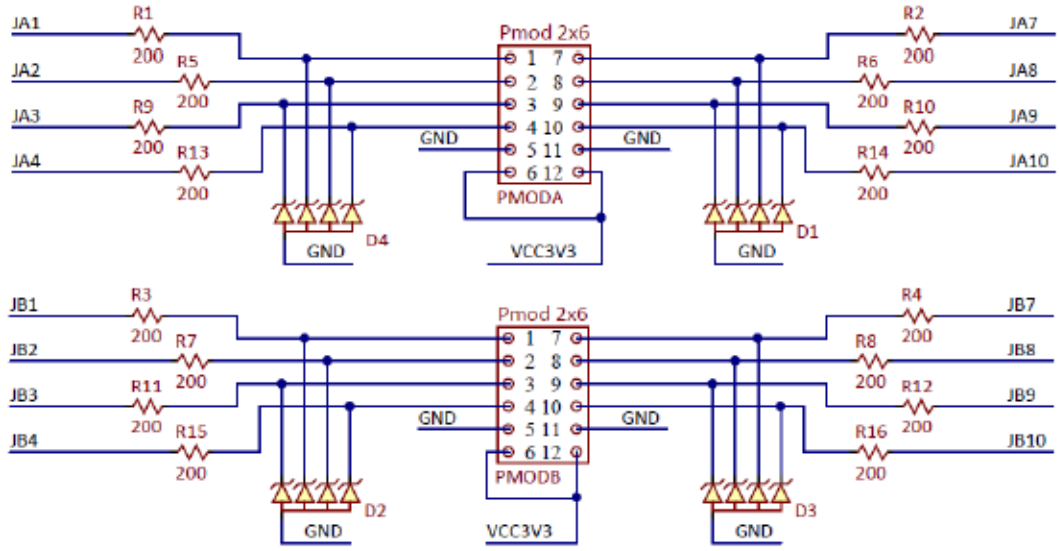



*Figura 61. Esquema eléctrico de los conectores Pmod*

Como podemos ver en el esquema, cada pin tiene una resistencia en serie de 200 ohmios y un diodo de protección contra descargas electrostáticas.

La resistencia en serie proporciona protección contra cortocircuitos, para evitar dañar el bloque de E/S del microcontrolador si el pin se cortocircuita accidentalmente a VCC o GND, o si dos salidas están cortocircuitadas entre ellas.

El diodo protege el bloque de E/S de posibles daños causados por descargas electrostáticas. Este fenómeno electrostático puede hacer que circule una corriente eléctrica repentina y momentánea entre dos objetos de distinto potencial eléctrico pudiendo causar daños en los equipos electrónicos.

Cuando se observan los conectores Pmod de la placa Basys MX3 desde arriba, el pin número uno es el que está etiquetado con el *'1'* y puede reconocerse también por su huella cuadrada.


*Figura 62. Vista detalle desde arriba de los conectores Pmod*

La numeración de los pines en los conectores Pmod de la placa Basys MX3 van del 1 al 6 de derecha a izquierda en la fila superior y del 7 al 12 en la fila inferior. Esto cumple con el diseño en el que ambas filas pueden considerarse como un conector Pmod de 6 pines cada uno individualmente.

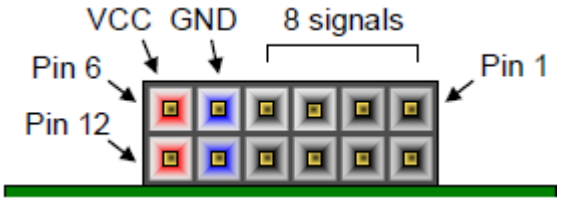



*Figura 63. Numeración de los pines en los conectores Pmod*

Los pines del 1 al 4 y del 7 al 10 de cada PMOD, son los correspondientes a las entradas o salidas, dependiendo de la configuración que se les dé. Los pines 5 y 11 son los correspondientes a GND, mientras que los pines 6 y 12 corresponden a la alimentación de 3.3 Voltios.

La Tabla 12 muestra todas las señales asociadas a los dos Pmods de la placa Basys MX3. En ella podemos observar la etiqueta con la que aparecen en los esquemas y las posibles funciones de cada pin dentro del microcontrolador PIC32.

| PMOD PIN | ETIQUETA | FUNCIONES PIC32 |
|---|---|---|
| **PMOD_A1** | **JA1** | **RPC2/RC2** |
| **PMOD_A2** | **JA2** | **RPC1/RC1** |
| **PMOD_A3** | **JA3** | **RPC4/CTED7/RC4** |
| PMOD_A4 | JA4 | AN16/C1IND/RPG6/SCK2/PMA5/RG6 |
| PMOD_A5 y A11 | JA5 y JA11 | GND |
| PMOD_A6 y A12 | JA6 y JA12 | VCC |
| **PMOD_A7** | **JA7** | **RPC3/RC3** |
| PMOD_A8 | JA8 | AN17/C1INC/RPG7/PMA4/RG7 |
| PMOD_A9 | JA9 | AN18/C2IND/RPG8/PMA3/RG8 |
| PMOD_A10 | JA10 | AN19/C2INC/RPG9/PMA2/RG9 |
| **PMOD_B1** | **JB1** | **RPD9/RD9** |
| **PMOD_B2** | **JB2** | **RPD11/PMCS1/RD11** |
| **PMOD_B3** | **JB3** | **RPD10/PMCS2/RD10** |
| **PMOD_B4** | **JB4** | **RPD8/RTCC/RD8** |
| PMOD_B5 y B11 | JB5 y JB11 | GND |
| PMOD_B6 y B12 | JB6 y JB12 | VCC |
| PMOD_B7 | JB7 | SOSCO/RPC14/T1CK/RC14 |
| PMOD_B8 | JB8 | RPD0/RD0 |
| PMOD_B9 | JB9 | AN24/RPD1/RD1 |
| PMOD_B10 | JB10 | SOSCI/RPC13/RC13 |

*Tabla 12. Señales asociadas a los pines de los Pmods*

En lugar de utilizar los pines correspondientes a los Pmod como pines normales de Entrada/Salida, podemos asignar a cada pin del microcontrolador diferentes funciones para controlar otros periféricos, esto se logra a través de la reasignación de cada pin.

Los pines resaltados en negrita toleran 5 Voltios, es decir, es totalmente seguro aplicarles señales lógicas de 5V directamente sin riesgo de dañar el microcontrolador, el resto de los pines solo toleran 3.3 Voltios.

## ❖ PMOD 8LD

El Pmod 8LD de Digilent tiene 8 LEDs verdes de alto brillo, controlados por transistores individualmente, de modo que cada LED se puede encender o apagar de forma independiente. Para iluminar uno o varios LEDs habría que dar una señal lógica alta a los pines correspondientes.

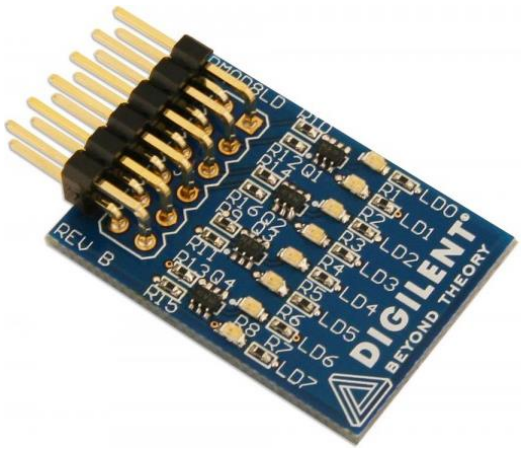



*Figura 64. Pmod 8LD*

Cuando se observa el Pmod 8LD desde arriba, el pin 1 se reconoce por su huella cuadrada, al igual que en los conectores Pmod de la placa. Es importante conectarlo bien en la placa para evitar posibles deterioros en los elementos electrónicos de ambos equipos.

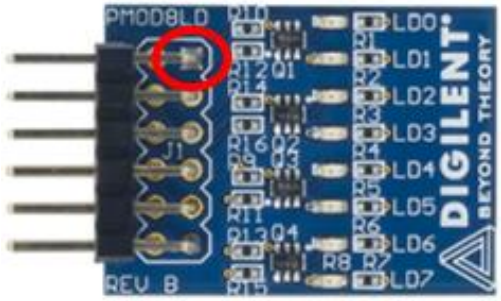



*Figura 65. Vista detalle desde arriba del Pmod 8LD*

En la Figura 67 podemos observar de forma esquemática la conexión eléctrica interna del Pmod 8LD.

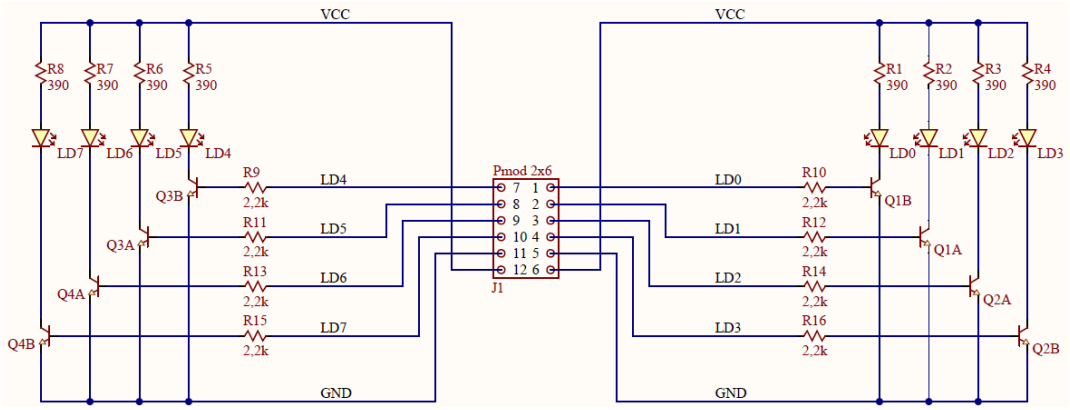



*Figura 66. Esquema de la conexión eléctrica del Pmod 8LD*

Por medio de los transistores BJT se hace un control a baja potencia de los diodos LED. Introduciendo una pequeña corriente por la base permite que una corriente mucho mayor circule entre el emisor y el colector, dicha corriente está limitada por una resistencia en serie a cada diodo LED.

La Tabla 13 muestra a que pin corresponde cada LED, la etiqueta con la que aparece en el esquema y la función de cada pin del Pmod 8LD.

| PMOD PIN | ETIQUETA | FUNCIÓN |
|---|---|---|
| PIN 1 | LD0 | LED 0 |
| PIN 2 | LD1 | LED 1 |
| PIN 3 | LD2 | LED 2 |
| PIN 4 | LD3 | LED 3 |
| PIN 5 y PIN 11 | GND | GND |
| PIN 6 y PIN 12 | VCC | VCC (3.3/5V) |
| PIN 7 | LD4 | LED 4 |
| PIN 8 | LD5 | LED 5 |
| PIN 9 | LD6 | LED 6 |
| PIN 10 | LD7 | LED 7 |

*Tabla 13. Descripción de los pines del Pmod 8LD*

### 3.5.2 LIBRERÍAS Y FUNCIONES

Las funciones de librería para controlar los posibles módulos periféricos que conectemos a la placa Basys MX3 están contenidas en el paquete de librerías que proporciona el fabricante.

La librería PMODS agrupa las funciones que acceden a las señales de los Pmods que incorpora nuestra placa. Tanto los pines del Pmod A como los del Pmod B se pueden inicializar como pines de entrada o salida digital mediante las funciones de configuración que incluye la librería.

Los archivos que debemos incluir, a parte de los indicados en la práctica número 2 en su correspondiente apartado del proyecto para poder usar las funciones de la librería son: ***'pmods.h'*** y ***'pmods.c'***.

El archivo con extensión *'.h'* lo incluiremos en el apartado *'Header Files',* mientras que el archivo con extensión *'.c'* los incluimos en el apartado *'Source Files'*.

Una vez incluidos todos los archivos necesarios de los correspondientes módulos que utilizaremos en nuestro proyecto, tendremos que modificar la función principal.

#### ❖ ENCABEZADO

Para usar las funciones de la librería PMODS tenemos que incluir el archivo de encabezado de la librería en la función principal, esto lo hacemos con la siguiente instrucción:

```
#include "pmod.h"
```

#### ❖ SETUP

Dentro de esta función lo que tenemos que incluir es la **instrucción de inicialización** para los Pmods de nuestra placa. De este modo con una sola instrucción y gracias a los archivos de librería que hemos incluido anteriormente, indicaremos que los pines correspondientes a los diferentes Pmods puedan ser usados como entradas o salidas digitales, además de otro tipo de características asociadas a cada pin que detallaremos a continuación.

La forma de inicializar cualquiera de los dos Pmods de nuestra placa es usando la instrucción: **PMODS_InitPin ('Pmod', 'Pin', 'Dir', 'Pull-Up', 'Pull-Down')**. En los siguientes puntos indicaremos a que corresponde cada variable y los posibles valores que le podemos dar para el correcto funcionamiento de la instrucción.

***'Pmod'*** hace referencia al Pmod que queremos controlar, el Pmod A o el B.

| VALOR | PMOD |
|---|---|
| 0 | PMOD A |
| 1 | PMOD B |

*Tabla 14. Opciones para dar a la variable 'Pmod'*

***'Pin'*** es el número de pin dentro del Pmod escogido con la variable anterior. Los pines 5-6 y 11-12 están reservados para la alimentación del Pmod.

| VALOR | PIN | PMOD |
|---|---|---|
| 1 | JA1 | PMOD_A = 0 |
| | JB1 | PMOD_B = 1 |
| 2 | JA2 | PMOD_A = 0 |
| | JB2 | PMOD_B = 1 |
| 3 | JA3 | PMOD_A = 0 |
| | JB3 | PMOD_B = 1 |
| 4 | JA4 | PMOD_A = 0 |
| | JB4 | PMOD_B = 1 |
| 7 | JA7 | PMOD_A = 0 |
| | JB7 | PMOD_B = 1 |
| 8 | JA8 | PMOD_A = 0 |
| | JB8 | PMOD_B = 1 |
| 9 | JA9 | PMOD_A = 0 |
| | JB9 | PMOD_B = 1 |
| 10 | JA10 | PMOD_A = 0 |
| | JB10 | PMOD_B = 1 |

*Tabla 15. Opciones para dar a la variable 'Pin'*

***'Dir'*** indica la dirección del pin señalado con las instrucciones anteriores, es decir, si el pin es de entrada o de salida.

| VALOR | DIRECCIÓN |
|---|---|
| 0 | SALIDA |
| 1 | ENTRADA |

*Tabla 16. Opciones para dar a la variable 'Dir'*

***'Pull-Up'*** y ***'Pull-Down'*** hacen referencia a si deseamos habilitar o no una resistencia interna al pin seleccionado en cualquiera de esas dos configuraciones.

| VALOR | PULL-UP | PULL-DOWN |
|---|---|---|
| 0 | NO HABILITADA | NO HABILITADA |
| 1 | SI HABILITADA | SI HABILITADA |

*Tabla 17. Opciones para dar a las variables 'Pull-Up' y 'Pull-Down'*

Por ejemplo, si queremos inicializar el pin 3 del Pmod B como una entrada digital y no queremos habilitar ni la resistencia de Pull-Up ni la Pull-Down, tendremos que escribir la siguiente instrucción:

```
PMODS_InitPin (1, 3, 1, 0, 0);
```

### ❖ LOOP

Al igual que el resto de pines de nuestra placa, los correspondientes a los Pmods pueden ser configurados como pines de entrada o salida digital, es decir, en ellos podremos realizar una operación de escritura o de lectura.

Para poder **leer el estado** de un pin dentro de un Pmod la instrucción que tenemos que usar es: **PMODS_GetValue ('Pmod', 'Pin').** Indicando el pin del cual deseamos conocer su estado y el Pmod al que pertenece. Los diferentes valores que podemos dar a las variables ***'Pmod'*** y ***'Pin'*** son los mismos que los homónimos de la instrucción de inicialización.

Las posibles respuestas que podemos obtener al leer el estado de un pin pueden ser los mostrados en la Tabla 18.

| RESPUESTA | SIGNIFICADO |
|---|---|
| 0 | PIN A NIVEL LÓGICO BAJO |
| 1 | PIN A NIVEL LÓGICO ALTO |
| 0xFF | PIN INDICADO ERRÓNEO |

*Tabla 18. Posibles respuestas al leer el estado de un pin del Pmod*

Por ejemplo, si queremos asignar a la variable *"Pin_3B"* el valor que está recibiendo el pin 3 del Pmod B tendremos que escribir la siguiente instrucción:

```
Pin_3B = PMODS_GetValue (0, 3);
```

Casi del mismo modo que antes, para poder **escribir un valor** determinado en un pin dentro de un Pmod la instrucción que tenemos que usar es: **PMODS_SetValue ('Pmod', 'Pin', 'Valor').** Indicando el valor que vamos a escribir, el pin escogido y el Pmod al que pertenece el pin. Los posibles valores que podemos dar a las variables ***'Pmod'*** y ***'Pin'*** son los mismos que los homónimos de la instrucción de inicialización.

Los valores que podemos escribir en un pin pueden ser los mostrados en la Tabla 19.

| VALOR | SIGNIFICADO |
|---|---|
| 0 | PIN A NIVEL LÓGICO BAJO |
| 1 | PIN A NIVEL LÓGICO ALTO |

*Tabla 19. Posibles valores que podemos dar a un pin del Pmod*

Por ejemplo, si queremos poner el pin 3 del Pmod B a nivel lógico alto, tendremos que escribir la siguiente instrucción:

```
PMODS_SetValue (1, 3, 1);
```

### 3.5.3 EJEMPLO USANDO LIBRERÍAS

Para tener un ejemplo más completo de lo expuesto anteriormente y verlo con más claridad, vamos a desarrollar un programa en el cual controlaremos el estado de los ocho LEDS del Pmod 8LD por medio de los interruptores que incorpora la placa Basys MX3.

El Pmod 8LD lo conectaremos en el conector PMOD A de la placa y el estado del LED lo determinará la posición del interruptor de igual numeración que el LED que queremos controlar.

Este programa lo que hace es inicializar los módulos que vamos a usar de la placa, leer el estado de los interruptores y asignarlo al LED correspondiente.

```
#include <stdio.h>
#include <stdlib.h>
#include "config.h"
#include "swt.h"
#include "pmods.h"
#include "BasysMX3.h"

void setup() {
    //INICIALIZACIÓN INTERRUPTORES COMO ENTRADA
    SWT_Init();
    //INICIALIZACIÓN PMOD-A COMO SALIDA
    PMODS_InitPin (0, 1, 0, 0, 0);
    PMODS_InitPin (0, 2, 0, 0, 0);
    PMODS_InitPin (0, 3, 0, 0, 0);
    PMODS_InitPin (0, 4, 0, 0, 0);
    PMODS_InitPin (0, 7, 0, 0, 0);
    PMODS_InitPin (0, 8, 0, 0, 0);
    PMODS_InitPin (0, 9, 0, 0, 0);
    PMODS_InitPin (0, 10, 0, 0, 0);
}

void loop() {
    //LED SEGÚN LA POSICIÓN DEL INTERRUPTOR
    PMODS_SetValue (0, 1, SWT_GetValue (0));
    PMODS_SetValue (0, 2, SWT_GetValue (1));
    PMODS_SetValue (0, 3, SWT_GetValue (2));
    PMODS_SetValue (0, 4, SWT_GetValue (3));
    PMODS_SetValue (0, 7, SWT_GetValue (4));
    PMODS_SetValue (0, 8, SWT_GetValue (5));
    PMODS_SetValue (0, 9, SWT_GetValue (6));
    PMODS_SetValue (0, 10, SWT_GetValue (7));
}
```

### 3.5.4 PROGRAMAR USANDO REGISTROS

En este apartado vamos a programar usando registros vamos a partir de lo explicado en la práctica 2, donde creamos un proyecto nuevo y cargamos el archivo de encabezado *'BasysMX3.h'* en el apartado *'Header Files',* pero en este caso ya no necesitaremos las librerías del fabricante.

No explicaremos todas las posibles maneras de escribir nuestro programa, nos centraremos en una, la hexadecimal.

#### ❖ ENCABEZADO

Además de las librerías que aparecen por defecto al crear la función principal tenemos que incluir el archivo de encabezado que hemos cargado antes, donde está la configuración básica de nuestra placa y la estructura del programa principal.

```
#include "BasysMX3.h"
```

#### ❖ SETUP

Dentro de esta función lo que vamos a incluir son las **instrucciones de inicialización** para todos los pines de Entrada/Salida del Pmod en forma hexadecimal. Indicaremos que los pines digitales correspondientes a los Pmod de nuestra placa Basys MX3 son pines de salida.

Para saber que bits y que puertos están asociados a los Pmods tenemos que consultar la Tabla 13, donde vienen las características de los pines. En este caso nos centraremos en el **Pmod A** sabiendo que para el B se haría del mismo modo.

Podemos observar que los pines correspondientes al Pmod A se encuentran en dos puertos diferentes, por lo que tendremos que escribir dos instrucciones diferentes, una por cada puerto.

```
TRISCCLR = 0x001E; //Bits de salida puerto C
TRISGCLR = 0x03C0; //Bits de salida puerto G
```

En la práctica 2 vimos que los pines que incluyen la funcionalidad de entrada analógica y van a usarse como pines digitales, como es en este caso, deben ser configurados para ello. Ya que por defecto están configurados como entrada analógica. En este caso son los pines correspondientes al puerto G.

```
ANSELGCLR = 0x03C0;//Deshabilitar entradas
analógicas puerto G
```

### ❖ LOOP

Para encender cualquier LED del Pmod 8LD se hace del mismo modo que en prácticas anteriores, mediante la escritura en el bit correspondiente de salida, esto lo hacemos por medio del registro **LAT**. La lógica para este caso es directa, es decir, para encender cualquier LED tendremos que hacerlo escribiendo un 1 en el pin correspondiente mientras que para apagarlo tendremos que escribir un 0.

Por ejemplo, si queremos encender el LED número 7, tendremos que escribir un 1 en el bit número 10 del Pmod 8LD, el cual corresponde al bit 9 del puerto G de la placa Basys MX3.

```
LATGbits.LATG9 = 1; //Encender el LED_7
```

## 3.5.5 EJEMPLO USANDO REGISTROS

Para tener un ejemplo más completo de lo expuesto anteriormente y verlo con más claridad conectaremos el Pmod 8LD al Pmod A de la placa Basys MX3 y desarrollaremos un programa con el que controlaremos los LEDs del Pmod 8LD por medio de los interruptores de la placa.

Controlaremos el estado de los diferentes LEDs mediante la activación o no de los 8 interruptores de la placa, de modo que cada interruptor controle el estado de un LED diferente.

Este programa lo que hace es inicializar los módulos que vamos a usar de la placa y encender o no encender los LEDs según sea la posición del interruptor que le corresponde, en este caso asignaremos cada LED al interruptor de igual numeración.

```
#include <stdio.h>
#include <stdlib.h>
#include "BasysMX3.h"

void setup () {
//INICIALIZACIÓN INTERRUPTORES COMO ENTRADA DIGITAL
    TRISFSET = 0x0038; //Bits de entrada puerto F
    TRISDSET = 0xC000; //Bits de entrada puerto D
    TRISBSET = 0x0E00; //Bits de entrada puerto B
    ANSELBCLR = 0x0E00;//Deshabilitar bits
analógicos puerto B

//INICIALIZACIÓN HEXADECIMAL PMOD-A COMO SALIDA
DIGITAL
    TRISCCLR = 0x001E; //Bits de salida puerto C
    TRISGCLR = 0x03C0; //Bits de salida puerto G
    ANSELBCLR = 0x03C0;//Deshabilitar bits
analógicos puerto G
}

void loop () {
//ENCENDER LEDS PMOD 8LD CON LOS INTERRUPTORES
    LATCbits.LATC2 = PORTFbits.RF3; //LED_0 con SW0
    LATCbits.LATC1 = PORTFbits.RF5; //LED_1 con SW1
    LATCbits.LATC4 = PORTFbits.RF4; //LED_2 con SW2
    LATGbits.LATG6 = PORTDbits.RD15;//LED_3 con SW3
    LATCbits.LATC3 = PORTDbits.RD14;//LED_4 con SW4
    LATGbits.LATG7 = PORTBbits.RB11;//LED_5 con SW5
    LATGbits.LATG8 = PORTBbits.RB10;//LED_6 con SW6
    LATGbits.LATG9 = PORTBbits.RB9; //LED_7 con SW7
}
```

### 3.5.6 CONTROL DE VERSIONES

| VERSIÓN | FECHA | COMENTARIOS |
|---|---|---|
| V 1.0 | 07/04/2019 | Versión inicial |
| | | |

*Tabla 20. Control de versiones práctica 5*

# 3.6 MOSTRAR CARACTERES EN EL DISPLAY DE CUATRO DÍGITOS Y SIETE SEGMENTOS DE LA PLACA BASYS MX3 [PRÁCTICA Nº 6]

## ❖ OBJETIVOS

El objetivo de la práctica nº6 es conocer que es un Display de cuatro dígitos y siete segmentos y las instrucciones necesarias para poder controlarlo, de modo que podamos obtener cualquier carácter través del Display.

Para tener una mejor visión de lo expuesto durante esta práctica lo comprobaremos mediante varios ejemplos en los que mostraremos diferentes caracteres a través del Display. La introducción de datos para variar la aparición de caracteres la haremos por medio de los interruptores que incorpora la placa Basys MX3, que ya vimos cómo se utilizaban en la práctica nº3.

Esto lo realizaremos mediante el uso de las librerías que nos proporciona el fabricante y el uso del lenguaje 'C' a bajo nivel.

## ❖ MATERIAL NECESARIO

### HARDWARE

- Placa Basys MX3.
- Cable micro USB.
- Ordenador personal.

### SOFTWARE

- Microchip MPLAB X IDE.
- Compilador MPLAB XC32.
- Librerías.

## 3.6.1 DESCRIPCIÓN DE PERIFÉRICOS

La placa Basys MX3 incorpora un Display de cuatro dígitos, con siete segmentos y un punto decimal cada uno.

Dentro del módulo podemos escoger encender un dígito o varios a la vez. Cada segmento corresponde a un LED diferente, que puede ser iluminado individualmente. Los posibles valores que podemos dar a cada segmento dentro de cada dígito es 0 o 1, dependiendo si queremos encender o no dicho segmento.

### ❖ DISPLAY DE CUATRO DÍGITOS Y SIETE SEGMENTOS

La placa Basys MX3 incorpora un display KW4-281ASB de cuatro dígitos, con siete segmentos y un punto decimal cada uno. Los dos puntos, situados entre los dígitos del medio, no se encuentran conectados y por lo tanto no pueden ser usados.

Los dígitos están etiquetados en los esquemas y tablas de características como AN0, AN1, AN2 y AN3, el punto decimal lo está como DP, mientras que los segmentos lo están como CA, CB, CC, CD, CE, CF y CG.

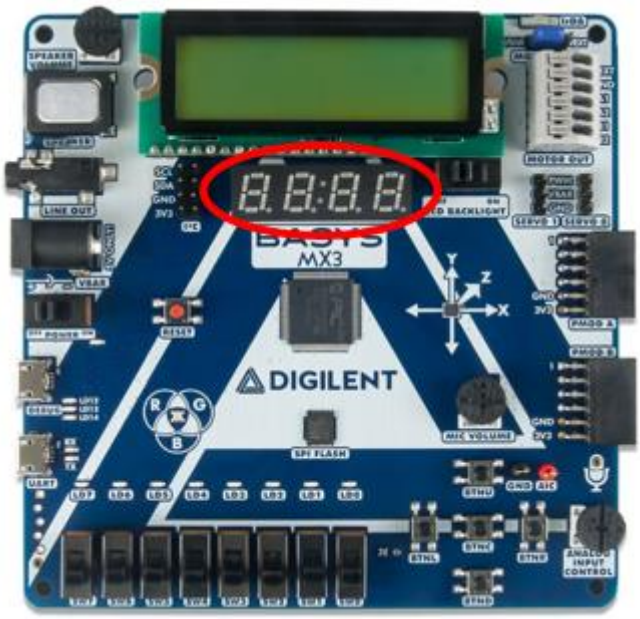



*Figura 67. Localización del display en la placa Basys MX3*

De los posibles patrones que podemos conseguir con las combinaciones encendiendo segmentos, los más útiles son los diez correspondientes a los dígitos decimales.

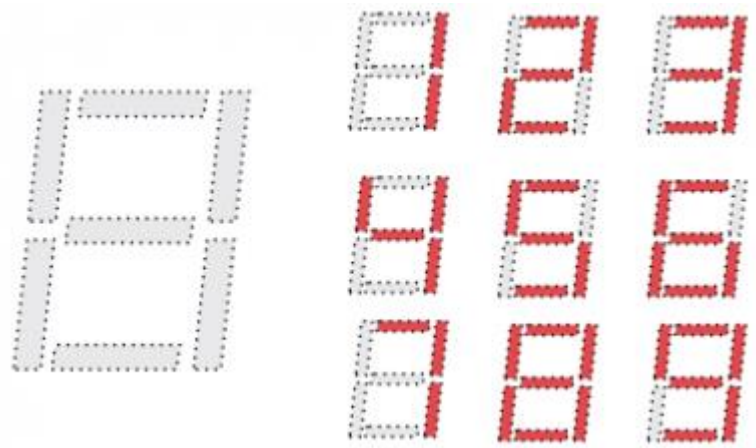


*Figura 68. Representación de los números decimales con segmentos*

En la Figura 70 podemos observar de forma esquemática la conexión eléctrica del Display de cuatro dígitos y siete segmentos dentro de la placa Basys MX3. En el esquema vemos que cada entrada a un segmento, va acompañada de una resistencia limitadora para no sobrepasar la intensidad máxima que soportan.

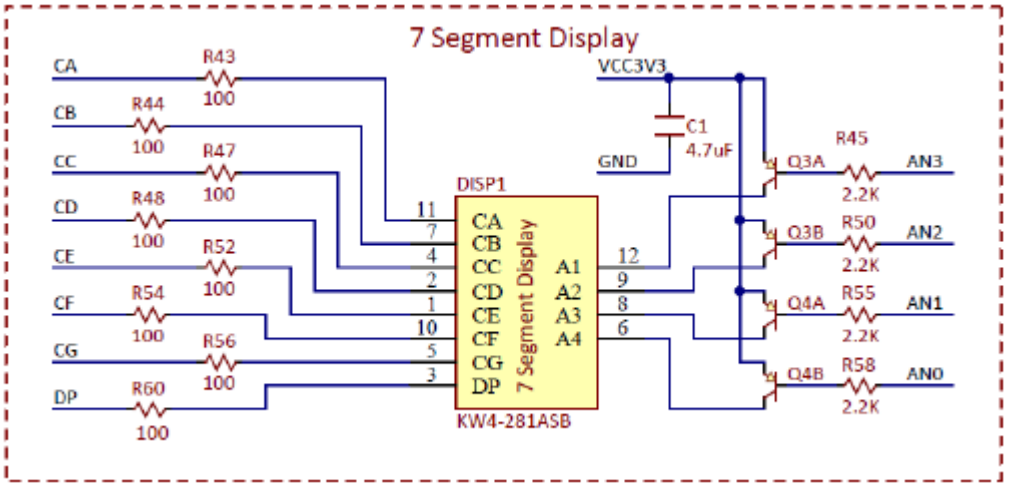



*Figura 69. Esquema eléctrico de las conexiones con el Display*

En la Figura 71 podemos observar de qué manera están conectados los diodos LED dentro de cada dígito que incorpora el Display de cuatro dígitos y siete segmentos.

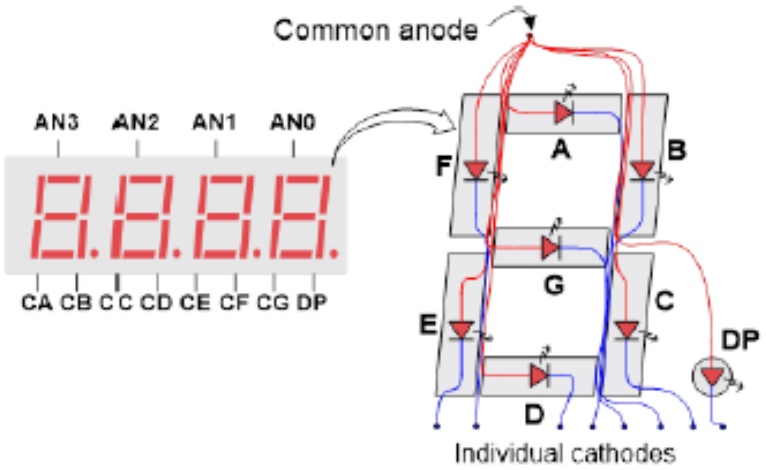



*Figura 70. Conexión de los LED dentro de los dígitos*

Los ánodos de los ocho LEDs, correspondientes a los siete segmentos y el punto decimal, que forman cada dígito están unidos entre sí en una configuración de ánodo común. Los ánodos comunes se usan como entrada para la habilitación de cada uno de los cuatro dígitos del display.

Los cátodos de los LEDs de elementos similares se conectan en ocho nodos, etiquetados como CA a CG y DP. Estas ocho entradas correspondientes a los cátodos son comunes a todos los dígitos, pero solo pueden iluminar los segmentos del dígito cuya señal de ánodo correspondiente esta activada.

Las señales de habilitación de dígito AN0...AN3 como las señales CA...CG/DP se activan cuando las ponemos a nivel bajo, es decir, cuando indicamos un 0 en la entrada.

Como solo se pude encender un dígito a la vez, para mostrar un número de más dígitos, tendremos que encender cada dígito alternamente con una velocidad de actualización mayor a la que el ojo humano puede captar, de este modo no detectaremos el oscurecimiento de un dígito antes de que se ilumine de nuevo, así el dígito parece iluminado continuamente.

Las siguientes tablas muestran todas las señales asociadas al Display, divididas en dos tablas, una para los dígitos y otra para los segmentos. En ella podemos observar la etiqueta con la que aparecen en los esquemas y las posibles funciones de cada pin dentro del microcontrolador PIC32.

| DÍGITO | ETIQUETA | CARACTERÍSTICAS PIC32 |
|---|---|---|
| DÍGITO 0 | AN0 | AN12/PMA11/RB12 |
| DÍGITO 1 | AN1 | AN13/PMA10/RB13 |
| DÍGITO 2 | AN2 | VREF-/CVREF-/PMA7/RA9 |
| DÍGITO 3 | AN3 | VREF+/CVREF+/PMA6/RA10 |

*Tabla 21. Señales asociadas a los pines de los dígitos*

| SEGMENTO | ETIQUETA | CARACTERÍSTICAS PIC32 |
|---|---|---|
| SEGMENTO A | CA | TRD1/RG12 |
| SEGMENTO B | CB | RPA14/RA14 |
| SEGMENTO C | CC | PMD14/RD6 |
| SEGMENTO D | CD | TRD0/RG13 |
| SEGMENTO E | CE | RG15 |
| SEGMENTO F | CF | PMD15/RD7 |
| SEGMENTO G | CG | PMD13/RD13 |
| PUNTO DECIMAL | DP | TRD2/RG14 |

*Tabla 22. Señales asociadas a los pines de los segmentos*

En lugar de utilizar los pines correspondientes a los dígitos o a los segmentos como pines normales de salida, podemos asignar a cada pin del microcontrolador diferentes funciones para controlar otros periféricos, esto se logra a través de la reasignación de cada pin.

### 3.6.2 LIBRERÍAS Y FUNCIONES

Las funciones de librería para usar el Display de cuatro dígitos y siete segmentos (SSD) están contenidas en el paquete de librerías que proporciona el fabricante.

La librería SSD agrupa las funciones que acceden a las señales Display de cuatro dígitos y siete segmentos. La placa Basys MX3 proporciona siete señales digitales para controlar los diferentes segmentos, otra señal digital para controlar los diferentes puntos decimales y cuatro señales más para controlar los diferentes dígitos.

La forma de actuar de la librería para el encendido de los diferentes dígitos es actualizar periódicamente cada dígito mientras que los otros están desactivados, esto lo realiza más rápido de lo que el ojo humano puede captar.

Los archivos que debemos incluir, en su correspondiente apartado del proyecto, para poder usar las funciones de la librería son: ***'ssd.h'*** y ***'ssd.c'***.

El archivo con extensión *'.h'* lo incluiremos en el apartado *'Header Files'* mientras que el archivo con extensión *'.c'* los incluimos en el apartado *'Source Files'*.

Una vez incluidos todos los archivos de los correspondientes módulos que utilizaremos en nuestro proyecto, tendremos que modificar la función principal.

### ❖ ENCABEZADO

Para usar las funciones de la librería SSD, tenemos que incluir el archivo de encabezado de la librería en la función principal, esto lo hacemos con la siguiente instrucción:

```
#include "ssd.h"
```

### ❖ SETUP

Dentro de esta función, lo que vamos a incluir es la **instrucción de inicialización** para el Display de cuatro dígitos y siete segmentos de nuestra placa. De modo que, con una sola instrucción, gracias a los archivos de librería que hemos incluido anteriormente, indiquemos que los pines correspondientes al Display, son salidas digitales y se inicialice un Timer para generar interrupciones cada tres milisegundos aproximadamente. Esto lo hacemos con la siguiente instrucción:

```
SSD_Init();
```

### ❖ LOOP

La instrucción que tenemos que usar para mostrar un carácter en cualquiera de los cuatro dígitos del Display o encender los puntos decimales que los acompañan es: **SSD_WriteDigits (D1, D2, D3, D4, Pd1, Pd2, Pd3, Pd4)**. Donde *'D1, D2, D3 y D4'* corresponden al valor que se representará en cada dígito, numerados de izquierda a derecha. Es decir, *'D1'* corresponde al dígito situado en el extremo derecho del Display, mientras que *'Pd1, Pd2, Pd3 y Pd4'* corresponden a los puntos decimales que acompañan a cada dígito respectivamente.

En la Tabla 23 observamos los diferentes valores que podemos dar a las variables *'D1, D2, D3 y D4'* y los caracteres que se mostrarán en el Display con cada una. Si se indica cualquier otro valor que no aparezca en la Tabla 23, dicho dígito no mostrara ningún carácter.

| D1 - D2 - D3 - D4 | CARÁCTER |
|---|---|
| 0 | '0' |
| 1 | '1' |
| 2 | '2' |
| 3 | '3' |
| 4 | '4' |
| 5 | '5' |
| 6 | '6' |
| 7 | '7' |
| 8 | '8' |
| 9 | '9' |
| 10 | 'A' |
| 11 | 'b' |
| 12 | 'C' |
| 13 | 'd' |
| 14 | 'E' |
| 15 | 'F' |
| 16 | 'H' |

*Tabla 23. Posibles entradas y salidas para los dígitos del Display*

A la hora de encender cualquier punto decimal tendremos que indicar cuál de ellos queremos encender, siguiendo la condición que se muestra en la Tabla 24.

| Pd1 - Pd2 - Pd3 - Pd4 | ACCIÓN |
|---|---|
| 0 | No encender el punto decimal |
| 1 | Encender el punto decimal |

*Tabla 24. Opciones para los puntos decimales del Display*

Por ejemplo, si queremos mostrar el número '2.019' en el display tendremos que escribir la siguiente instrucción:

```
SSD_WriteDigits(9, 1, 0, 2, 0, 0, 0, 1);
```

### 3.6.3 EJEMPLO USANDO LIBRERÍAS

Para tener un ejemplo más completo de lo expuesto anteriormente y verlo con más claridad, vamos a desarrollar un programa en el cual introduciremos un número por medio de los interruptores de la placa y lo mostraremos por el display, tanto en su forma decimal como en la hexadecimal.

La introducción del número la realizaremos en forma binaria, mediante la activación o no de los interruptores SW3, SW2, SW1 y SW0, siendo este último, el correspondiente al bit menos significativo de nuestro número binario.

En el dígito situado en el extremo derecho del display mostraremos las unidades y en el contiguo las decenas del número en su forma decimal. El número en forma hexadecimal lo mostraremos en el dígito situado en el extremo izquierdo, dejando apagado el dígito restante.

Este programa lo que hace es inicializar los módulos que vamos a usar de la placa, leer el estado de los interruptores y multiplicarlo por el peso de cada bit para obtener la transformación de binario a decimal. Después calculamos las unidades y las decenas y por último mostramos en el Display ambas representaciones del número.

```
#include <stdio.h>
#include <stdlib.h>
#include "config.h"
#include "swt.h"
#include "ssd.h"
#include "BasysMX3.h"

void setup() {
    //INICIALIZACIÓN INTERRUPTORES COMO ENTRADA
    SWT_Init();
    //INICIALIZACIÓN DISPLAY COMO SALIDA
    SSD_Init(0);
}

void loop() {
    int numero, unidades, decenas;

    numero = SWT_GetValue(0)*1 + SWT_GetValue(1)*2
+ SWT_GetValue(2)*4 + SWT_GetValue(3)*8;



    unidades = numero % 10;
    decenas = numero / 10;

    SSD_WriteDigits(unidades, decenas, -1, numero,
0, 0, 0, 0);

}
```

### 3.6.4 ENCENDER DIFERENTES SEGMENTOS USANDO REGISTROS

En este apartado vamos a programar usando registros a partir de lo explicado en la práctica 2. Donde creamos un proyecto nuevo y cargamos el archivo de encabezado *'BasysMX3.h'* en el apartado *'Header Files',* pero en este caso ya no necesitaremos las librerías del fabricante.

Ya no explicaremos todas las posibles maneras de escribir nuestro programa, sino que nos centraremos en una, la hexadecimal.

## ❖ ENCABEZADO

Además de las librerías que aparecen por defecto al crear la función principal, tenemos que incluir el archivo de encabezado que hemos cargado antes, donde está la configuración básica de nuestra placa y la estructura del programa principal.

```
#include "BasysMX3.h"
```

## ❖ SETUP

Dentro de esta función lo que vamos a incluir son las **instrucciones de inicialización** para el Display de cuatro dígitos y siete segmentos en forma hexadecimal. Indicaremos que los pines digitales correspondientes a los cuatro dígitos y a los ocho LEDs (siete segmentos más el punto decimal) de nuestra placa Basys MX3 son pines de salida.

Para saber que bits de que puertos están asociados a los cuatro dígitos y cuales a los ocho LEDs tenemos que consultar las Tablas 21 y 22, donde vienen las características de los pines dentro del microcontrolador PIC32.

Podemos observar que los cuatro dígitos y ocho LEDs se encuentran en cuatro puertos diferentes, por lo que tendremos que escribir cuatro instrucciones diferentes, una por cada puerto.

```
TRISACLR = 0x4600;//Bits de salida puerto A
TRISBCLR = 0x3000;//Bits de salida puerto B
TRISDCLR = 0x20C0;//Bits de salida puerto D
TRISGCLR = 0xF000;//Bits de salida puerto G
```

En la práctica 2 vimos que los pines que incluyen la funcionalidad de entrada analógica y van a usarse como pines digitales, como es en este caso, deben ser configurados como pines digitales. Ya que por defecto están configurados como entrada analógica. En este caso son los pines correspondientes al puerto B.

```
ANSELBCLR = 0x3000;//Deshabilitar entradas
analógicas puerto B
```

### ❖ LOOP

Para poder encender un LED de los ocho posibles (siete segmentos y el punto decimal) dentro de un dígito, lo primero que tenemos que hacer es indicar en que dígito lo queremos mostrar y después indicar que LEDs queremos encender.

Para encender cualquier LED del Display se hace del mismo modo que en prácticas anteriores, mediante la escritura en el bit correspondiente de salida. Esto lo hacemos por medio del registro **LAT**, pero en este caso la lógica es inversa, es decir, para encender los LEDs correspondientes a los segmentos y al punto decimal o incluso para seleccionar en que dígito queremos encenderlos tenemos que hacerlo escribiendo un 0 en los pines correspondientes y no un 1 como en el resto de casos anteriores.

Por ejemplo, si queremos mostrar el número 7 en el dígito situado en el extremo derecha del display, tendremos que indicar el dígito donde lo queremos mostrar escribiendo un 0 en el bit correspondiente y después indicar que segmentos queremos encender escribiendo un 0 en cada bit.

```
//INDICAMOS EL DÍGITO DONDE QUEREMOS ESCRIBIR
LATBbits.LATB12 = 0; //Encender    AN0
LATBbits.LATB13 = 1; //No encender AN1
LATAbits.LATA9  = 1; //No encender AN2
LATAbits.LATA10 = 1; //No encender AN3

//MOSTRAR UN 7
LATGbits.LATG12 = 0; //Encender CA
LATAbits.LATA14 = 0; //Encender CB
LATDbits.LATD6  = 0; //Encender CC
LATGbits.LATG13 = 1; //No encender CD
LATGbits.LATG15 = 1; //No encender CE
LATDbits.LATD7  = 1; //No encender CF
LATDbits.LATD13 = 1; //No encender CG
LATGbits.LATG14 = 1; //No encender DP
```

### 3.6.5 EJEMPLO USANDO REGISTROS

Para tener un ejemplo más completo de lo expuesto anteriormente y verlo con más claridad, vamos a desarrollar un programa en el cual podremos encender los distintos segmentos del display por medio de los interruptores de la placa.

Controlaremos el estado de los diferentes segmentos mediante la activación o no de los 8 interruptores de la placa, de modo que cada interruptor controle el estado de un segmento diferente, mostraremos los segmentos en todos los dígitos del display.

Este programa lo que hace es inicializar los módulos que vamos a usar de la placa, indicar en que dígito o dígitos deseamos encender los diferentes segmentos, en este caso encenderemos los segmentos en todos los dígitos del display, y por último encender o no encender el segmento según sea la posición del interruptor que le corresponde.

```
#include <stdio.h>
#include <stdlib.h>
#include "BasysMX3.h"

void setup() {
//INICIALIZACIÓN INTERRUPTORES COMO ENTRADA DIGITAL
    TRISFSET = 0x0038;//Bits de entrada puerto F
    TRISDSET = 0xC000;//Bits de entrada puerto D
    TRISBSET = 0x0E00;//Bits de entrada puerto B
    ANSELBCLR = 0x0E00;//Deshabilitar analógica B

//INICIALIZACIÓN DISPLAY COMO SALIDA DIGITAL
    TRISACLR = 0x4600;//Bits de salida puerto A
    TRISBCLR = 0x3000;//Bits de salida puerto B
    TRISDCLR = 0x20C0;//Bits de salida puerto D
    TRISGCLR = 0xF000;//Bits de salida puerto G
    ANSELBCLR = 0x3000;//Deshabilitar analógicas B
}

void loop() {
//INDICAMOS EL DÍGITO DONDE QUEREMOS ESCRIBIR
    LATBbits.LATB12 = 0; //Encender AN0
    LATBbits.LATB13 = 0; //Encender AN1
    LATAbits.LATA9  = 0; //Encender AN2
    LATAbits.LATA10 = 0; //Encender AN3
```

```
//ENCENDER SEGMENTOS CON LOS INTERRUPTORES
    LATGbits.LATG12 = PORTFbits.RF3; //CA con SW0
    LATAbits.LATA14 = PORTFbits.RF5; //CB con SW1
    LATDbits.LATD6  = PORTFbits.RF4; //CC con SW2
    LATGbits.LATG13 = PORTDbits.RD15;//CD con SW3
    LATGbits.LATG15 = PORTDbits.RD14;//CE con SW4
    LATDbits.LATD7  = PORTBbits.RB11;//CF con SW5
    LATDbits.LATD13 = PORTBbits.RB10;//CG con SW6
    LATGbits.LATG14 = PORTBbits.RB9; //DP con SW7
}
```

### 3.6.6 CONTROL DE VERSIONES

| VERSIÓN | FECHA | COMENTARIOS |
|---|---|---|
| V 1.0 | 27/04/2019 | Versión inicial |
| | | |

*Tabla 25. Control de versiones práctica 6*

# CAPÍTULO 4:
# CONCLUSIONES FINALES Y LÍNEAS FUTURAS

## 4.1 CONCLUSIONES FINALES Y LÍNEAS FUTURAS

Con la realización de este trabajo fin de grado se han conseguido varios objetivos: crear unos guiones de prácticas, que sirvan de apoyo a futuros alumnos que deseen aprender a programar de forma autónoma, y que el propio autor aprenda a programar los distintos periféricos de los que dispone la placa Basys MX3.

Para la realización de los primeros guiones hemos seguido un desarrollo más detallado y hemos llevado al usuario más de la mano durante estos guiones introductorios. El guion más importante es el correspondiente a Práctica nº 2, puesto que para la realización de los guiones siguientes nos basamos en lo que hemos desarrollado en este.

Durante el desarrollo de estos guiones se ha logrado mantener la estructura con la que se plateó al principio. Cada guion cuenta con una lista de objetivos, la descripción de los periféricos, que se han usado en cada práctica, y una descripción de la programación empleada.

Uno de los objetivos logrado fue la programación de dos modos diferentes: un modo usando las librerías, que nos proporciona el propio fabricante, y otro modo sin usarlas. Con el uso de librerías, la programación es más sencilla, ya que no tenemos que implementar las funciones de programación a bajo nivel y conseguimos hacer funcionar los distintos periféricos de una manera rápida y sencilla. Mientras, en el modo sin librerías, somos nosotros los que implementamos esas funciones a bajo nivel, por lo que la programación no será tan sencilla, pero lograremos adquirir unos conocimientos de programación que no estén tan enfocados hacia la placa Basys MX3. Este hecho facilitará a los usuarios, que dispongan de este trabajo, la programación de otros dispositivos con diferentes microcontroladores que se basen en lenguaje 'C' en los que no dispondremos de librerías.

También, tal y como se marcó en los objetivos, además de usar distintos periféricos que incorpora la placa Basys MX3, hemos incluido la utilización de un periférico Pmod. Descubrimos que son unos dispositivos de gran utilidad al ser capaces de ampliar las capacidades de nuestra placa, de una manera rápida y cómoda, ofreciendo grandes posibilidades ante futuros trabajos.

En todos estos guiones hemos incluido, al menos, un ejemplo que muestre las funciones de programación desarrolladas durante lo descrito en la práctica, debido a que a través de un caso práctico es como mejor se comprende lo explicado.

Finalmente, la aportación principal de este trabajo es introducir a otros alumnos en el estudio de estos componentes, ampliando así sus conocimientos en programación de esta familia de microcontroladores. De esta forma, servirá como base para el desarrollo de nuevas aplicaciones y proyectos futuros derivados de este mismo.

A continuación, una vez determinadas las conclusiones finales conseguidas tras la realización de este trabajo, estableceremos las diferentes líneas de actuación futuras, abriendo así las diferentes posibilidades de nuevos proyectos.

Mostraremos algunas de las posibles líneas de actuación derivadas a partir de la realización de este trabajo:

- Continuar programando y realizando guiones sobre el resto de periféricos de los que dispone la placa Basys MX3, que no se han utilizado en el desarrollo de este trabajo.

- Programar otros periféricos Pmod diferentes. Hoy en día, el mercado de dispositivos Pmod es muy amplio, y hay dispositivos para casi cualquier aplicación que deseemos implementar.

- Programar otro tipo de placas, que se basen en el microcontrolador PIC de 32 bits y ofrezca otras características diferentes.

Estas posibles líneas futuras pueden ser tomadas, tanto en el ámbito de servir simplemente al autoaprendizaje de un usuario particular, o de convertirse en una ampliación de este trabajo, y que un usuario que así lo desee realice su propio trabajo ampliando el alcance de este.

# CAPÍTULO 5:
# BIBLIOGRAFÍA

En este capítulo se hace referencia a todas las páginas web y documentos que se han utilizado como apoyo para obtener la información necesaria durante el desarrollo del este Trabajo Fin de Grado. Ordenados según aparición en este documento.

## ❖ PAGINAS WEB

- **Página web del vendedor**
  https://shop.trenz-electronic.de/en/Products/Microcontroller/

  https://shop.trenz-electronic.de/en/28278-Basys-MX3-PIC32MX-Trainer-Board-for-Embedded-Systems-Courses?c=23

- **Página web del fabricante**
  https://store.digilentinc.com/basys-mx3-pic32mx-trainer-board-for-embedded-systems-courses/

- **Página Web del fabricante del microcontrolador**
  https://www.microchip.com/wwwproducts/en/PIC32MX370F512L

- **Información microcontrolador PIC**
  https://es.wikipedia.org/wiki/Microcontrolador_PIC

- **Página web para la descarga de MPLAB X IDE**
  http://www.microchip.com/mplab/mplab-x-ide

- **Instalación de MPLAB X IDE**
  http://microchipdeveloper.com/mplabx:installation

  http://microchipdeveloper.com/tls0101:start

- **Página web para la descarga de MPLAB XC32**
  http://www.microchip.com/mplab/compilers

- **Instalación de MPLAB XC32**
  http://microchipdeveloper.com/xc32:installation

- **Cursos del fabricante**
  https://reference.digilentinc.com/learn/courses/microprocessor-io-unit-1/start

- **Manual de referencia del fabricante**

https://reference.digilentinc.com/reference/microprocessor/basys-mx3/reference-manual

- **Página web para la descarga de las librerías del fabricante**

https://reference.digilentinc.com/reference/microprocessor/basys-mx3/reference-manual

- **Página web del vendedor del Pmod 8LD**

https://shop.trenz-electronic.de/en/23530-Pmod-8LD-Eight-High-brightness-LEDs

## ❖ DOCUMENTOS

- Data Sheet PIC32MX330/350/370/430/450/470
- Data Sheet LED RGB VS NRD8
- Data Sheet PMOD 8LD
- Data Sheet Display KW4-281ASB

# CAPÍTULO 6:
# ANEXOS

En este capítulo vamos a mostrar los Data Sheet de algunos componentes de los que dispone la placa Basys MX3 y hemos usado para buscar información durante el desarrollo de este trabajo.

MICROCHIP

# PIC32MX330/350/370/430/450/470

## 32-bit Microcontrollers (up to 512 KB Flash and 128 KB SRAM) with Audio/Graphics/Touch (HMI), USB, and Advanced Analog

### Operating Conditions: 2.3V to 3.6V

- -40°C to +105°C (DC to 80 MHz)
- -40°C to +85°C (DC to 100 MHz)
- 0°C to +70°C (DC to 120 MHz)

### Core: 120 MHz/150 DMIPS MIPS32® M4K®

- MIPS16e® mode for up to 40% smaller code size
- Code-efficient (C and Assembly) architecture
- Single-cycle (MAC) 32x16 and two-cycle 32x32 multiply

### Clock Management

- 0.9% internal oscillator
- Programmable PLLs and oscillator clock sources
- Fail-Safe Clock Monitor (FSCM)
- Independent Watchdog Timer
- Fast wake-up and start-up

### Power Management

- Low-power management modes (Sleep and Idle)
- Integrated Power-on Reset, Brown-out Reset, and High Voltage Detect
- 0.5 mA/MHz dynamic current (typical)
- 50 µA IPD current (typical)

### Audio/Graphics/Touch HMI Features

- External graphics interface with up to 34 PMP pins
- Audio data communication: $I^2S$, LJ, RJ, USB
- Audio data control interface: SPI and $I^2C$
- Audio data master clock:
  - Generation of fractional clock frequencies
  - Can be synchronized with USB clock
  - Can be tuned in run-time
- Charge Time Measurement Unit (CTMU):
  - Supports mTouch™ capacitive touch sensing
  - Provides high-resolution time measurement (1 ns)

### Advanced Analog Features

- ADC Module:
  - 10-bit 1 Msps rate with one Sample and Hold (S&H)
  - Up to 28 analog inputs
  - Can operate during Sleep mode
- Flexible and independent ADC trigger sources
- On-chip temperature measurement capability
- Comparators:
  - Two dual-input Comparator modules
  - Programmable references with 32 voltage points

### Timers/Output Compare/Input Capture

- Five General Purpose Timers:
  - Five 16-bit and up to two 32-bit Timers/Counters
- Five Output Compare (OC) modules
- Five Input Capture (IC) modules
- Peripheral Pin Select (PPS) to allow function remap
- Real-Time Clock and Calendar (RTCC) module

### Communication Interfaces

- USB 2.0-compliant Full-speed OTG controller
- Up to five UART modules (20 Mbps):
  - LIN 2.1 protocols and IrDA® support
- Two 4-wire SPI modules (25 Mbps)
- Two $I^2C$ modules (up to 1 Mbaud) with SMBus support
- PPS to allow function remap
- Parallel Master Port (PMP)

### Direct Memory Access (DMA)

- Four channels of hardware DMA with automatic data size detection
- 32-bit Programmable Cyclic Redundancy Check (CRC)
- Two additional channels dedicated to USB

### Input/Output

- 15 mA or 12 mA source/sink for standard VOH/VOL and up to 22 mA for non-standard VOH1
- 5V-tolerant pins
- Selectable open drain, pull-ups, and pull-downs
- External interrupts on all I/O pins

### Qualification and Class B Support

- AEC-Q100 REVH (Grade 2 -40°C to +105°C) planned
- Class B Safety Library, IEC 60730

### Debugger Development Support

- In-circuit and in-application programming
- 4-wire MIPS® Enhanced JTAG interface
- Unlimited program and six complex data breakpoints
- IEEE 1149.2-compatible (JTAG) boundary scan

### Packages

| Type | QFN | TQFP | | | VTLA |
|---|---|---|---|---|---|
| Pin Count | 64 | 64 | 100 | 100 | 124 |
| I/O Pins (up to) | 53 | 53 | 85 | 85 | 85 |
| Contact/Lead Pitch | 0.50 | 0.50 | 0.40 | 0.50 | 0.50 |
| Dimensions | 9x9x0.9 | 10x10x1 | 12x12x1 | 14x14x1 | 9x9x0.9 |

Note: All dimensions are in millimeters (mm) unless specified.

**TABLE 1: PIC32MX330/350/370/430/450/470 CONTROLLER FAMILY FEATURES**

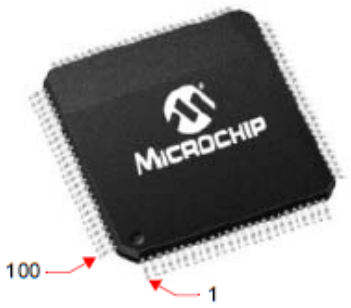
| Device | Pins | Packages | Program Memory (KB)(1) | Data Memory (KB) | Remappable Peripherals | | | | | 10-bit 1 Msps ADC (Channels) | Analog Comparators | USB On-The-Go (OTG) | CTMU | I²C | PMP | RTCC | DMA Channels (Programmable/Dedicated) | I/O Pins | JTAG | Trace |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | Remappable Pins | Timers/Capture/Compare(2) | UART | SPI/I²S | External Interrupts(3) | | | | | | | | | | | |
| PIC32MX330F064H | 64 | QFN, TQFP | 64+12 | 16 | 37 | 5/5/5 | 4 | 2/2 | 5 | 28 | 2 | N | Y | 2 | Y | Y | 4/0 | 53 | Y | N |
| PIC32MX330F064L | 100 | TQFP | 64+12 | 16 | 54 | 5/5/5 | 5 | 2/2 | 5 | 28 | 2 | N | Y | 2 | Y | Y | 4/0 | 85 | Y | Y |
| | 124 | VTLA | | | | | | | | | | | | | | | | | | |
| PIC32MX350F128H | 64 | QFN, TQFP | 128+12 | 32 | 37 | 5/5/5 | 4 | 2/2 | 5 | 28 | 2 | N | Y | 2 | Y | Y | 4/0 | 53 | Y | N |
| PIC32MX350F128L | 100 | TQFP | 128+12 | 32 | 54 | 5/5/5 | 5 | 2/2 | 5 | 28 | 2 | N | Y | 2 | Y | Y | 4/0 | 85 | Y | Y |
| | 124 | VTLA | | | | | | | | | | | | | | | | | | |
| PIC32MX350F256H | 64 | QFN, TQFP | 256+12 | 64 | 37 | 5/5/5 | 4 | 2/2 | 5 | 28 | 2 | N | Y | 2 | Y | Y | 4/0 | 53 | Y | N |
| PIC32MX350F256L | 100 | TQFP | 256+12 | 64 | 54 | 5/5/5 | 5 | 2/2 | 5 | 28 | 2 | N | Y | 2 | Y | Y | 4/0 | 85 | Y | Y |
| | 124 | VTLA | | | | | | | | | | | | | | | | | | |
| PIC32MX370F512H | 64 | QFN, TQFP | 512+12 | 128 | 37 | 5/5/5 | 4 | 2/2 | 5 | 28 | 2 | N | Y | 2 | Y | Y | 4/0 | 53 | Y | N |
| PIC32MX370F512L | 100 | TQFP | 512+12 | 128 | 54 | 5/5/5 | 5 | 2/2 | 5 | 28 | 2 | N | Y | 2 | Y | Y | 4/0 | 85 | Y | Y |
| | 124 | VTLA | | | | | | | | | | | | | | | | | | |
| PIC32MX430F064H | 64 | QFN, TQFP | 64+12 | 16 | 34 | 5/5/5 | 4 | 2/2 | 5 | 28 | 2 | Y | Y | 2 | Y | Y | 4/2 | 49 | Y | N |
| PIC32MX430F064L | 100 | TQFP | 64+12 | 16 | 51 | 5/5/5 | 5 | 2/2 | 5 | 28 | 2 | Y | Y | 2 | Y | Y | 4/2 | 81 | Y | Y |
| | 124 | VTLA | | | | | | | | | | | | | | | | | | |
| PIC32MX450F128H | 64 | QFN, TQFP | 128+12 | 32 | 34 | 5/5/5 | 4 | 2/2 | 5 | 28 | 2 | Y | Y | 2 | Y | Y | 4/2 | 49 | Y | N |
| PIC32MX450F128HB (see Note 4) | 64 | QFN, TQFP | 128+12 | 32 | 34 | 5/5/5 | 4 | 2/2 | 5 | 28 | 2 | Y | Y | 2 | Y | Y | 4/2 | 49 | Y | N |
| PIC32MX450F128L | 100 | TQFP | 128+12 | 32 | 51 | 5/5/5 | 5 | 2/2 | 5 | 28 | 2 | Y | Y | 2 | Y | Y | 4/2 | 81 | Y | Y |
| | 124 | VTLA | | | | | | | | | | | | | | | | | | |
| PIC32MX450F256H | 64 | QFN, TQFP | 256+12 | 64 | 34 | 5/5/5 | 4 | 2/2 | 5 | 28 | 2 | Y | Y | 2 | Y | Y | 4/2 | 49 | Y | N |
| PIC32MX450F256L | 100 | TQFP | 256+12 | 64 | 51 | 5/5/5 | 5 | 2/2 | 5 | 28 | 2 | Y | Y | 2 | Y | Y | 4/2 | 81 | Y | Y |
| | 124 | VTLA | | | | | | | | | | | | | | | | | | |
| PIC32MX470F512H | 64 | QFN, TQFP | 512+12 | 128 | 34 | 5/5/5 | 4 | 2/2 | 5 | 28 | 2 | Y | Y | 2 | Y | Y | 4/2 | 49 | Y | N |
| PIC32MX470F512L | 100 | TQFP | 512+12 | 128 | 51 | 5/5/5 | 5 | 2/2 | 5 | 28 | 2 | Y | Y | 2 | Y | Y | 4/2 | 81 | Y | Y |
| | 124 | VTLA | | | | | | | | | | | | | | | | | | |
| PIC32MX470F512LB (see Note 4) | 100 | TQFP | 512+12 | 128 | 51 | 5/5/5 | 5 | 2/2 | 5 | 28 | 2 | Y | Y | 2 | Y | Y | 4/2 | 81 | Y | Y |
| | 124 | VTLA | | | | | | | | | | | | | | | | | | |

**Note 1:** All devices feature 12 KB of Boot Flash memory.
**2:** Four out of five timers are remappable.
**3:** Four out of five external interrupts are remappable.
**4:** This PIC32 device is targeted to specific audio software packages that are tracked for licensing royalty purposes. All peripherals and electrical characteristics are identical to their corresponding base part numbers

**TABLE 4: PIN NAMES FOR 100-PIN DEVICES**

**100-PIN TQFP (TOP VIEW)(1,2,3)**

PIC32MX330F064L
PIC32MX350F128L
PIC32MX350F256L
PIC32MX370F512L



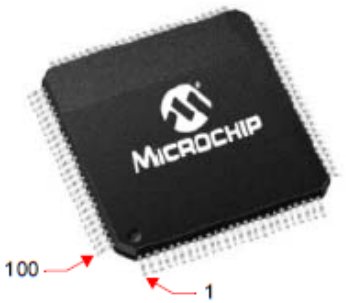

| Pin # | Full Pin Name | Pin # | Full Pin Name |
|---|---|---|---|
| 1 | RG15 | 36 | Vss |
| 2 | VDD | 37 | VDD |
| 3 | AN22/RPE5/PMD5/RE5 | 38 | TCK/CTED2/RA1 |
| 4 | AN23/PMD6/RE6 | 39 | RPF13/RF13 |
| 5 | AN27/PMD7/RE7 | 40 | RPF12/RF12 |
| 6 | RPC1/RC1 | 41 | AN12/PMA11/RB12 |
| 7 | RPC2/RC2 | 42 | AN13/PMA10/RB13 |
| 8 | RPC3/RC3 | 43 | AN14/RPB14/CTED5/PMA1/RB14 |
| 9 | RPC4/CTED7/RC4 | 44 | AN15/RPB15/OCFB/CTED6/PMA0/RB15 |
| 10 | AN16/C1IND/RPG6/SCK2/PMA5/RG6 | 45 | Vss |
| 11 | AN17/C1INC/RPG7/PMA4/RG7 | 46 | VDD |
| 12 | AN18/C2IND/RPG8/PMA3/RG8 | 47 | RPD14/RD14 |
| 13 | MCLR | 48 | RPD15/RD15 |
| 14 | AN19/C2INC/RPG9/PMA2/RG9 | 49 | RPF4/PMA9/RF4 |
| 15 | Vss | 50 | RPF5/PMA8/RF5 |
| 16 | VDD | 51 | RPF3/RF3 |
| 17 | TMS/CTED1/RA0 | 52 | RPF2/RF2 |
| 18 | RPE8/RE8 | 53 | RPF8/RF8 |
| 19 | RPE9/RE9 | 54 | RPF7/RF7 |
| 20 | AN5/C1INA/RPB5/RB5 | 55 | RPF6/SCK1/INT0/RF6 |
| 21 | AN4/C1INB/RB4 | 56 | SDA1/RG3 |
| 22 | PGED3/AN3/C2INA/RPB3/RB3 | 57 | SCL1/RG2 |
| 23 | PGEC3/AN2/C2INB/RPB2/CTED13/RB2 | 58 | SCL2/RA2 |
| 24 | PGEC1/AN1/RPB1/CTED12/RB1 | 59 | SDA2/RA3 |
| 25 | PGED1/AN0/RPB0/RB0 | 60 | TDI/CTED9/RA4 |
| 26 | PGEC2/AN6/RPB6/RB6 | 61 | TDO/RA5 |
| 27 | PGED2/AN7/RPB7/CTED3/RB7 | 62 | VDD |
| 28 | VREF-/CVREF-/PMA7/RA9 | 63 | OSC1/CLKI/RC12 |
| 29 | VREF+/CVREF+/PMA6/RA10 | 64 | OSC2/CLKO/RC15 |
| 30 | AVDD | 65 | Vss |
| 31 | AVss | 66 | RPA14/RA14 |
| 32 | AN8/RPB8/CTED10/RB8 | 67 | RPA15/RA15 |
| 33 | AN9/RPB9/CTED4/RB9 | 68 | RPD8/RTCC/RD8 |
| 34 | CVREFOUT/AN10/RPB10/CTED11PMA13/RB10 | 69 | RPD9/RD9 |
| 35 | AN11/PMA12/RB11 | 70 | RPD10/PMCS2/RD10 |

**Note**
1: The RPn pins can be used by remappable peripherals. See Table 1 for the available peripherals and Section 12.3 "Peripheral Pin Select" for restrictions.
2: Every I/O port pin (RAx-RGx), with the exception of RF6, can be used as a change notification pin (CNAx-CNGx). See Section 12.0 "I/O Ports" for more information.
3: RPF6 (pin 55) and RPF7 (pin 54) are only remappable for input functions.

**TABLE 4: PIN NAMES FOR 100-PIN DEVICES (CONTINUED)**

**100-PIN TQFP (TOP VIEW)(1,2,3)**

PIC32MX330F064L
PIC32MX350F128L
PIC32MX350F256L
PIC32MX370F512L




| Pin # | Full Pin Name | Pin # | Full Pin Name |
|---|---|---|---|
| 71 | RPD11/PMCS1/RD11 | 86 | VDD |
| 72 | RPD0/RD0 | 87 | RPF0/PMD11/RF0 |
| 73 | SOSCI/RPC13/RC13 | 88 | RPF1/PMD10/RF1 |
| 74 | SOSCO/RPC14/T1CK/RC14 | 89 | RPG1/PMD9/RG1 |
| 75 | VSS | 90 | RPG0/PMD8/RG0 |
| 76 | AN24/RPD1/RD1 | 91 | TRCLK/RA6 |
| 77 | AN25/RPD2/RD2 | 92 | TRD3/CTED8/RA7 |
| 78 | AN26/RPD3/RD3 | 93 | PMD0/RE0 |
| 79 | RPD12/PMD12/RD12 | 94 | PMD1/RE1 |
| 80 | PMD13/RD13 | 95 | TRD2/RG14 |
| 81 | RPD4/PMWR/RD4 | 96 | TRD1/RG12 |
| 82 | RPD5/PMRD/RD5 | 97 | TRD0/RG13 |
| 83 | PMD14/RD6 | 98 | AN20/PMD2/RE2 |
| 84 | PMD15/RD7 | 99 | RPE3/CTPLS/PMD3/RE3 |
| 85 | VCAP | 100 | AN21/PMD4/RE4 |

**Note 1:** The RPn pins can be used by remappable peripherals. See Table 1 for the available peripherals and **Section 12.3 "Peripheral Pin Select"** for restrictions.

**2:** Every I/O port pin (RAx-RGx), with the exception of RF6, can be used as a change notification pin (CNAx-CNGx). See **Section 12.0 "I/O Ports"** for more information.

**3:** RPF6 (pin 55) and RPF7 (pin 54) are only remappable for input functions.

## 1.0 DEVICE OVERVIEW

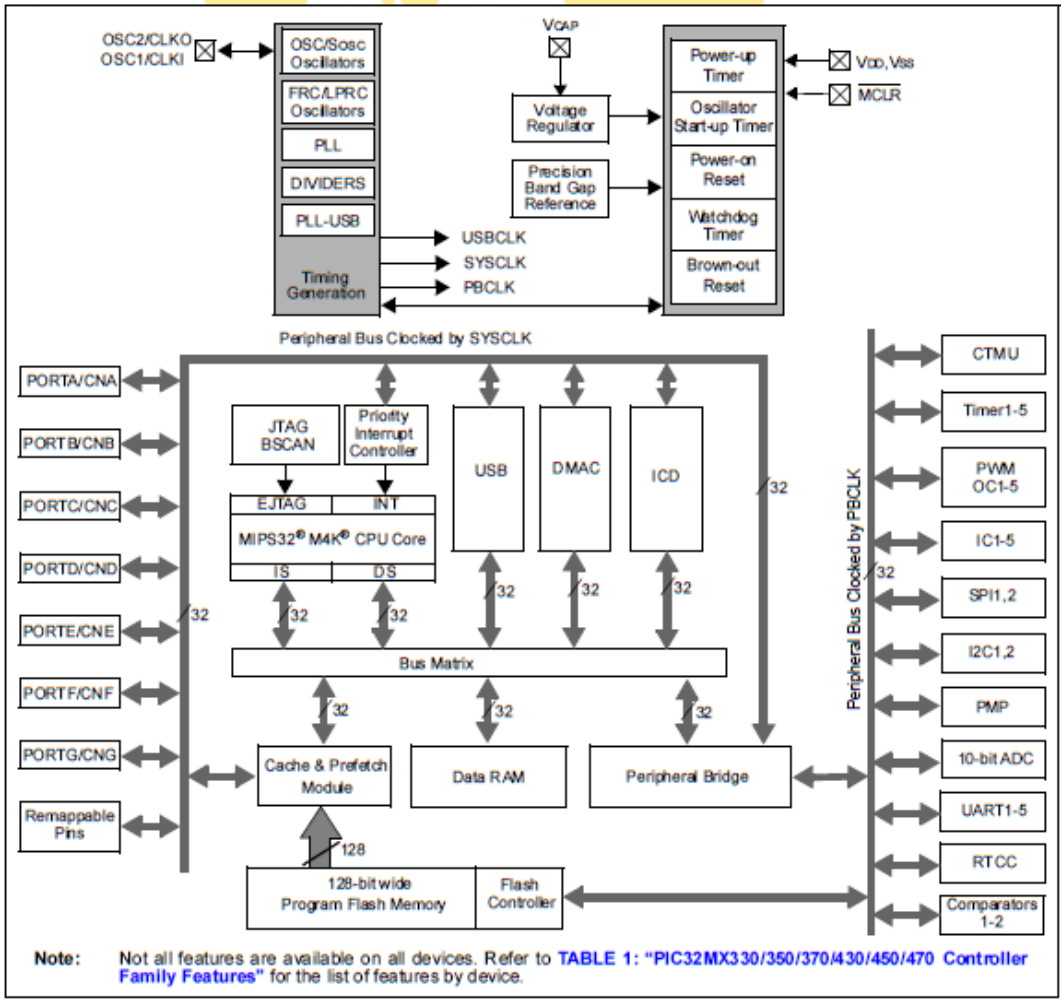
**Note:** This data sheet summarizes the features of the PIC32MX330/350/370/430/450/470 family of devices. It is not intended to be a comprehensive reference source. To complement the information in this data sheet, refer to the documents listed in the *Documentation > Reference Manual* section of the Microchip PIC32 web site (www.microchip.com/pic32).

This document contains device-specific information for PIC32MX330/350/370/430/450/470 devices.

Figure 1-1 illustrates a general block diagram of the core and peripheral modules in the PIC32MX330/350/370/430/450/470 family of devices.

Table 1-1 lists the functions of the various pins shown in the pinout diagrams.

**FIGURE 1-1: PIC32MX330/350/370/430/450/470 BLOCK DIAGRAM**



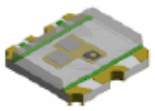

**Note:** Not all features are available on all devices. Refer to **TABLE 1: "PIC32MX330/350/370/430/450/470 Controller Family Features"** for the list of features by device.

# 1.6 x 1.6 mm Full Color SMT LED

## VS NRD8

### Description

This is Full Color Top View SMD LED with dimension 1.6mm (L) * 1.6 mm (W) * 0.35 mm (H), common anode configuration.
The unique of these LED are made from AlGaInP chips and InGaN technology that provide high luminous intensity at low driving current.
The lens design provides 120 degree viewing angle and suitable for lighting indicator purpose.



### Applications

- Industrial control systems signal indicator
- Automotive features
- Front panel indicator
- Status indication

RoHS

Pb
pb-free

**Electronic Optical Characteristics (at 20mA):**

| Part Number | Emitted Color | λ (nm) | | Lens Color | Iv(mcd) | | View Angle | VF(V) | |
|---|---|---|---|---|---|---|---|---|---|
| | | λD | λP | | Min. | Max. | | Typ. | Max. |
| VS NRD8 | Red | 624 | 632 | Clear | 71 | 180 | 120 | 2.0 | 2.4 |
| | Green | 525 | 518 | | 112 | 285 | | 3.3 | 3.8 |
| | Blue | 470 | 468 | | 28 | 71 | | 3.3 | 3.8 |

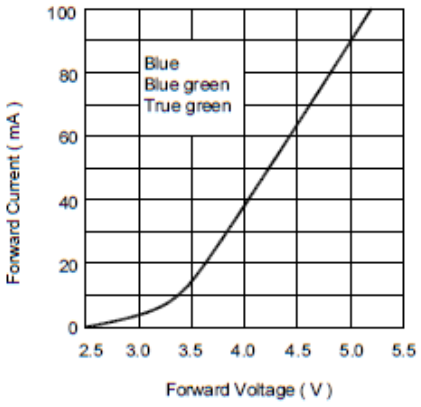
**Absolute Maximum Ratings (at Ta=25℃)**

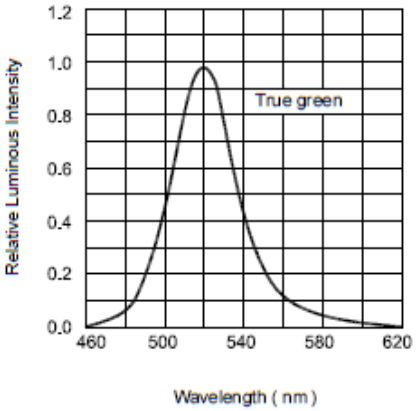
| Emitted Color | $P_D$ (mW) | $I_F$(mA) | $I_{FP}$(mA) | $I_R$(uA) @VR=5V | Topr (℃) | Tstg (℃) |
|---|---|---|---|---|---|---|
| Green | 110 | 25 | 100 * | 50 | -40~+85 | -40~+100 |
| Red | 60 | 25 | 60 * | 10 | -40~+85 | -40~+100 |
| Blue | 110 | 25 | 100 * | 50 | -40~+85 | -40~+100 |

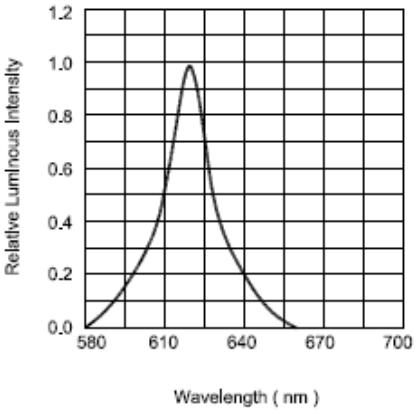
Note: Please take note the Absolute Maximum Rating values. Any operation beyond the specified ratings in this table will result degradation of LED life-span and may cause LED to fail.

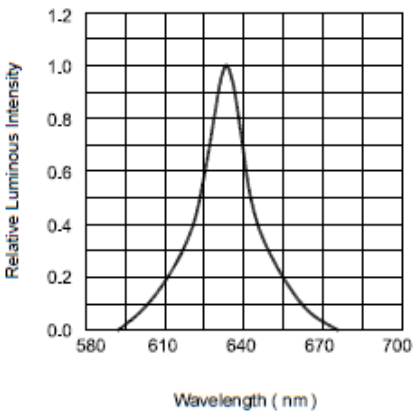
*$I_{FP}$ = Peak Forward Current under 1/10 duty, 1KHz condition

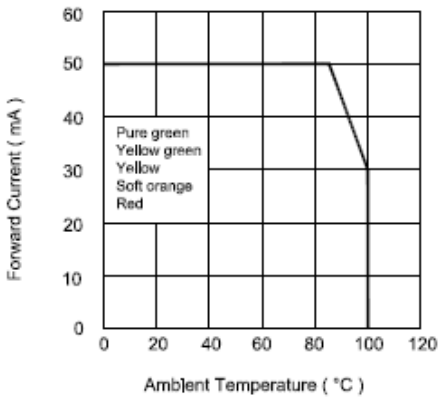
## Optical Characteristics Curves

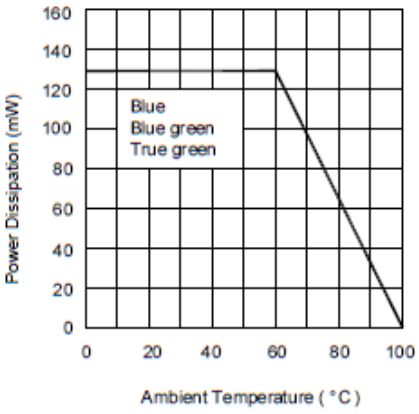


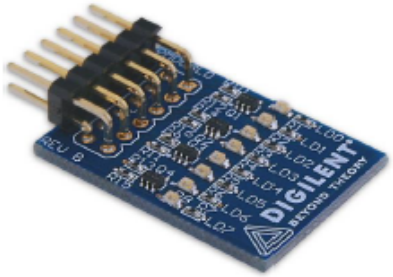
1300 Henley Court
Pullman, WA 99163
509.334.6306
www.digilentinc.com

# Pmod 8LD™ Reference Manual

Revised April 15, 2016
This manual applies to the Pmod 8LD rev. B

## Overview

The Digilent Pmod 8LD has eight high-bright LEDs that are driven by logic-level transistors so that each LED can be individually illuminated from a logic high signal.



*The Pmod 8LD.*

Features include:

- Eight high brightness green LEDs
- BJTs for low-power logic level control
- Small PCB size for flexible designs 1.1 in × 0.8 in (2.8 cm × 2.0 cm)
- 2×6-pin Pmod port with GPIO interface

## 1 Functional Description

The Pmod 8LD utilizes individual transistors so that each LED can be turned on or off independently. In order to active an LED, the associated pin on the pin header must receive about 1mA of current.

## 2 Interfacing with the Pmod

The Pmod 8LD communicates with the host board via GPIO pins. Correspondingly, to turn a particular LED on, the pin must be driven to a logic high state and driven to a logic low state to turn a LED off. With the parallel arrangement of these LEDs it is possible to turn on (or off) multiple LEDs simultaneously.

A pin description and diagram are provided below.

| Header J1 | | |
|---|---|---|
| Pin | Signal | Description |
| 1 | LD0 | LED 0 |
| 2 | LD1 | LED 1 |
| 3 | LD2 | LED 2 |
| 4 | LD3 | LED 3 |
| 5 | GND | Power Supply Ground |
| 6 | VCC | Power Supply (3.3/5V) |
| 7 | LD4 | LED 4 |
| 8 | LD5 | LED 5 |
| 9 | LD6 | LED 6 |
| 10 | LD7 | LED 7 |
| 11 | GND | Power Supply Ground |
| 12 | VCC | Power Supply (3.3/5V) |

*Table 1. Pinout description table.*

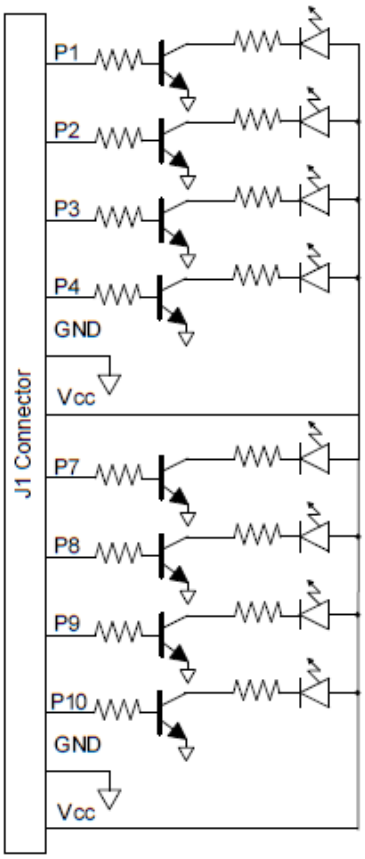



*Figure 1. Pmod 8LD module circuit diagram.*

## 3 Physical Dimensions

The pins on the pin header are spaced 100 mil apart. The PCB is 1.1 inches long on the sides parallel to the pins on the pin header and 0.8 inches long on the sides perpendicular to the pin header.

Luckylight

## Features:

- 0.28″ (inch) digit height.
- Excellent segment uniformity.
- Sold state reliability.
- Industrial standard size.
- Low power consumption.
- The product itself will remain within RoHS compliant Version.

## Descriptions:

- The KW4-281XXX series is a lager 7.0 mm (0.28″) high seven segments display designed for viewing distances up to 7 meters.
- These displays provide excellent reliability in bright ambient light.
- These devices are made with white segments and black surface.

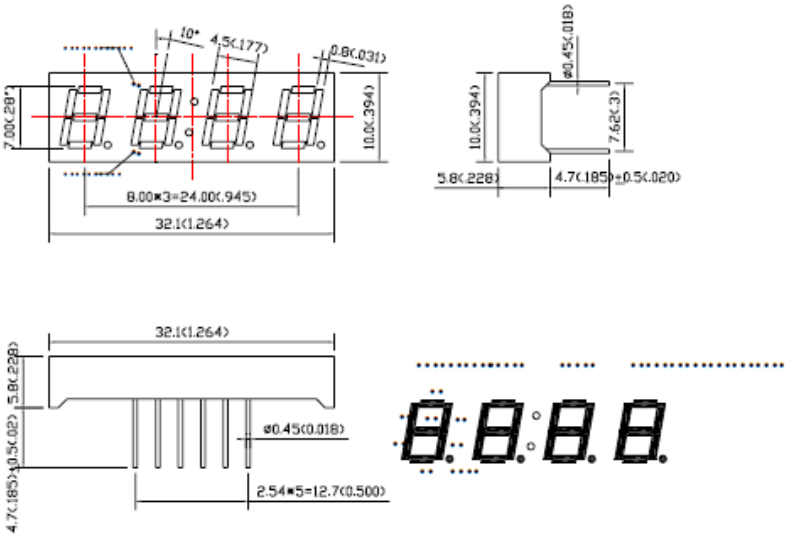
## Applications:

- Audio equipment.
- Instrument panels.
- Digital read out display.

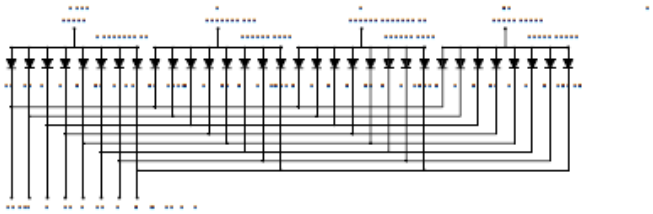
## Device Selection Guide:

| Model No. | Chip Material | Source Color | Description |
|---|---|---|---|
| KW4-281ASB | GaAlAs | Super Bright Red | Common Anode |
| KW4-281CSB | | Super Bright Red | Common Cathode |

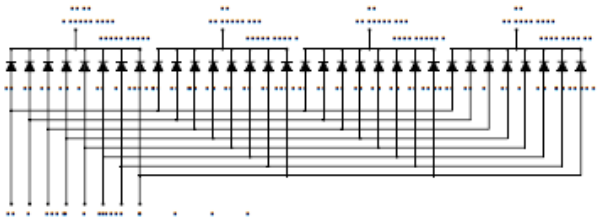
Package Dimension:




KW4-281ASB



KW4-281CSB



Notes:

1. All dimensions are in millimeters (inches).
2. Tolerance is ± 0.25 mm (.010") unless otherwise noted.
3. Specifications are subject to change without notice.

Luckylight

## Absolute Maximum Ratings at Ta=25• •

| Parameters | Symbol | Max. | Unit |
|---|---|---|---|
| Power Dissipation Per Segment | PD | 60 | mW |
| Peak Forward Current Per Segment (1/10 Duty Cycle, 0.1ms Pulse Width) | IFP | 100 | mA |
| Forward Current Per Segment | IF | 25 | mA |
| Dating Linear From 50• • | | 0.4 | mA/• • |
| Reverse Voltage | VR | 5 | V |
| Operating Temperature Range | Topr | -40• •to +80• • | |
| Storage Temperature Range | Tstg | -40• •to +85• • | |
| Soldering Temperature | Tsld | 260• •for 5 Seconds | |

## Electrical Optical Characteristics at Ta=25• •

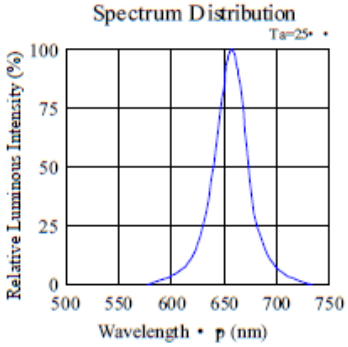
| Parameters | Symbol | Min. | Typ. | Max. | Unit | Test Condition |
|---|---|---|---|---|---|---|
| Luminous Intensity | Iv | 2.0 | 4.0 | --- | mcd | IF=20mA (Note 1) |
| Luminous Intensity Matching Ratio (Segment To Segment) | $I_{v-m}$ | --- | --- | 2:1 | | IF=10mA |
| Peak Emission Wavelength | λp | --- | 660 | --- | nm | IF=20mA |
| Dominant Wavelength | λd | --- | 640 | --- | nm | IF=20mA (Note 2) |
| Spectral Line Half-Width | • ƛ | --- | 20 | --- | nm | IF=20mA |
| Forward Voltage | VF | --- | 1.80 | 2.40 | V | IF=20mA |
| Reverse Current | IR | --- | --- | 50 | μA | VR=5V |

Notes:

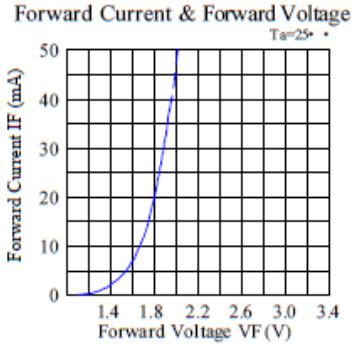
1. Luminous intensity is measured with a light sensor and filter combination that approximates the CIE eye-response curve.
2. The dominant wavelength (λd) is derived from the CIE chromaticity diagram and represents the single wavelength which defines the color of the device.

Typical Electrical / Optical Characteristics Curves
(25• •Ambient Temperature Unless Otherwise Noted)

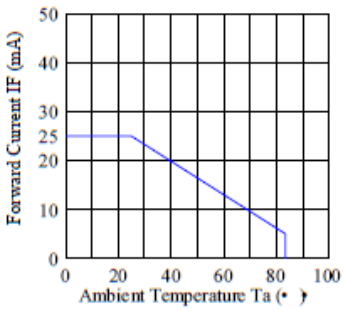
Spectrum Distribution



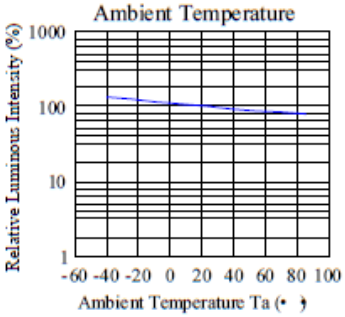

Forward Current & Forward Voltage




Forward Current Derating Curve




Luminous Intensity & Ambient Temperature




Luminous Intensity & Forward Current