# UNIVERSIDAD CATÓLICA DEL URUGUAY FACULTAD DE INGENIERÍA Y TECNOLOGÍAS SISTEMAS EMBEBIDOS - 2022

**Laboratorio 1**

*Introducción a la placa de desarrollo del curso y al MPLAB X IDE*

# Objetivo

Interiorizarse con el hardware de la placa de desarrollo del curso, el PIC32, y el entorno de desarrollo MPLAB X IDE. Comenzar a escribir código en lenguaje de programación C para aplicaciones embebidas. Desarrollar funciones básicas de E/S. Emplear las herramientas de compilación y depuración que ofrece el entorno de desarrollo.

# Evaluación y metodología de trabajo

A partir de este laboratorio, y para todos los laboratorios restantes y el proyecto final, se trabajará en grupos (siempre el mismo). El docente asistirá a los estudiantes constantemente realizando sugerencias, devoluciones y aclarando conceptos cada vez que se necesite.

Se evaluará el trabajo de cada grupo durante el laboratorio, y se considerarán para dicha evaluación los siguientes criterios:

1. Organización del grupo, involucramiento de cada uno de los integrantes.
2. Completitud y correctitud de las soluciones a las tareas indicadas en el laboratorio en el tiempo de clase.
3. Respuestas a las preguntas planteadas en el laboratorio y/o preguntas que los docentes puedan realizar.
4. Utilización de buenas prácticas de programación.
   1. El código entregado **debe** compilar sin errores ni warnings. En caso de que por algún motivo excepcional sea necesario ignorar un warning, explicar porqué en el informe.
   2. En el código no deben haber funciones, variables, comentarios, etc. que no se utilicen. Por ejemplo, si al crear un archivo hay funciones de ejemplo, estas se deben quitar antes de entregar el código. E. Entrega en fecha de las soluciones propuestas.

# Ejercicios

## Primera Parte

Introducción al MPLAB X IDE, creación de proyectos y primeros pasos.

### 1. Creación de un proyecto en MPLAB X IDE

1. Abrir el MPLAB X IDE y crear un nuevo proyecto vacío (Standalone Project) para el PIC32MM0256GPM064, Hardware Tool: Simulator, y compilador XC32. Darle un nombre al proyecto, por ejemplo “Laboratorio” y guardarlo en el directorio que deseen.
2. Analizar el árbol de directorios creado en el navegador del proyecto (vista

“Projects”, no “Files”).

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

* **Header Files** llevará la firma de los métodos que estará guardado en un archivo “.h” o “.inc”.
* **Important Files** agregar .pdf o cualquier documento que no está en el resto de las carpetas, pero son importantes.
* **Linker Files** carpeta donde podemos agregar scripts de vinculación a nuestro proyecto. Únicamente debe existir un archivo en esta carpeta, si hay más de uno tomará el primer archivo. Extensiones “.ld”, “.gld” o “.lkr”.
* **Source Files** son los archivos ejecutables del programa, que tendrán la extensión *.c*.
* **Libraries** extensiones utilizables de código ya definido y creado por otros programadores. Extensiones “.a” o “.lib”.
* **Loadables** son archivos y proyectos para combinar o remplazar los archivos Hexa de nuestro proyecto en c. Extensiones “.o”.

1. Agregar un archivo ***main.c*** con formato xc32. Para ello, hacer clic derecho en la carpeta *Source Files -> New xc32\_newfile.c* y llamarlo main.c.
2. Analizar el archivo generado: ¿Qué secciones tiene? ¿Cuáles van a usarse en el *main.c*? ¿Cuáles no y por qué?

* **Descriptive File Name** – Será esencial para todos los archivos que sea desarrollado por nosotros, ya que nos detalla que realizará cada código o como está compuesto a muy alto nivel.
* **Included Files** – Parte esencial del main, ya que deberán ser incluidos los “.c”, “.h” entre otros posibles para el correcto funcionamiento del programa.
* **File Scope or Global Data** – Puede llegar a ser utilizado, dependiendo la complejidad del programa a ejecutar. En este laboratorio no será utilizado.
* **Local Functions** – No serán usadas en este archivo, sino que serán definidas en los archivos “.h”. Aquí solo se instanciarán.
* **Interface Functions** – No serán usadas en este archivo, sino que serán definidas en los archivos “.h”.

1. Reemplazar la función **ExampleInterfaceFunction** por la **main** con la siguiente estructura:

int main( void ){ while(1){

Nop(); } return 0;

}

### 2. Bits de Configuración

1. Los dispositivos PIC32 tienen varios registros que contienen los ***bits de configuración***. Estos bits especifican la operación fundamental del dispositivo, como el modo del oscilador, el watchdog timer, el modo de programación y la protección de código. Si no se configuran correctamente los bits pueden provocar un error de código o un dispositivo que no se ejecuta correctamente.

Estos ***bits de configuración*** se deben configurar en el código a través de la directiva ***#pragma config setting\_name = value\_name.***

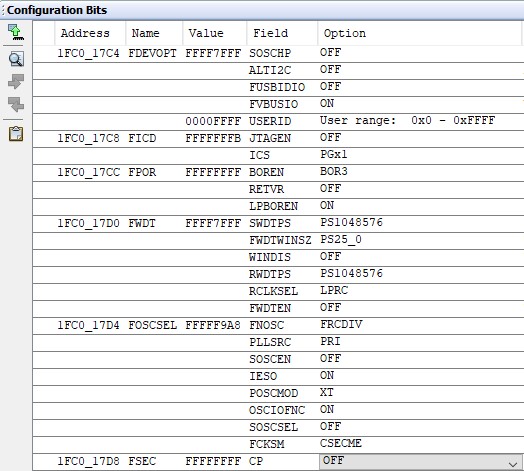
El nombre de configuración y valor son específicos del dispositivo y pueden determinarse mediante el documento *PIC32ConfigSet.html* ubicado en el directorio de instalación del compilador, carpeta ***docs***.

Ver el archivo PIC32ConfigSet.html y analizar los distintos parámetros configurables.

Tenemos diferentes configuraciones de los bits, ya sea si queremos configurar los diferentes osciladores, clocks, los pines de control (si son por puertos o USB), boton de reset, el voltaje, los canales de comunicación, el Watchdog Timer, si el codigo puede ser editable o no (protegido).

1. Afortunadamente, MPLAB X ofrece la configuración de los bits de configuración a través de una interfaz gráfica. Ir al menú Window -> Target Memory Views ->

Configuration Bits, y en la ventana que se abre realizar la siguiente configuración:



Luego de seteados los parámetros, dar click en *Generate Source Code to Output* y copiar en el ***main.c*** ,antes de cualquier directiva #include, el código generado.

1. Analizar qué significa la configuración elegida.

Tenemos 6 secciones de configuration bits:

**FDEVOPT:** es donde configura las opciones del dispositivo, ya sea un identificador de usuario, si los pines son controlados por el USB o por los puertos, el oscilador secundario, etc.

**FICD:** configura los registros del debug como por que canal se comunican.

**FPOR:** configura los registros reguladores de voltaje y bajo consumo.

**FWDT:** configura los registros del Watchdog Timer.

**FOSCSEL:** configura los registros del oscilador, ya sea los clock, los osciladores secundarios y los modos de osciladores.

**FSEC:** configura los registros de la protección del código.

### 3. Compilación

1. Agregar al archivo **main.c** la siguiente definición y las siguientes variables:

#define ARRAY\_SIZE 12

int exampleData; char exampleArray[ARRAY\_SIZE];

1. Compilar el proyecto creado (en caso de errores corregirlos) y ver los archivos generados en la carpeta del proyecto *Laboratorio.X*. Analizar particularmente las carpetas *build* y *dist*.

¿Qué tipos de archivos se encuentran en la carpeta *dist*?

* *.elf*
* *.hex*
* *.map*
* *.xml*

1. Buscar las variables antes declaradas en el archivo *.map* dentro de la carpeta *dist*. ¿Qué tamaño ocupan en la memoria y en qué direcciones están alojadas?

* Tamaño 0xc y dirección de memoria 0x0000000080000004

1. Sustituir ARRAY\_SIZE por 10 y ver qué ocurre con exampleArray en la memoria.

* Tamaño 0xa y dirección de memoria 0x0000000080000004

### 4. Ejecución, Depuración

1. En las propiedades del proyecto verificar que el SNAP esté asignado como herramienta de hardware.
2. Agregar un breakpoint en la línea de Nop(), y clickear en *Debug Project.*
3. Ver qué sucede, avanzar paso a paso, observar el contenido de las variables a través de Watch, y el código generado en *Execution Memory.*

* En esta instancia se nos rompe los breakpoints, por lo que no podemos visualizar las variables a través del Execution Memory en acción. Ya que al ser una instrucción *nop* y una línea de C pueden ser muchas operaciones dentro de Assembler, capaz no es posible seguirlo, en este caso, paso a paso a un *nop*.

Una captura de pantalla de un celular

Descripción generada automáticamente con confianza media

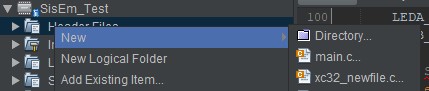
## Segunda Parte

En esta parte del laboratorio se realizará el “Hola Mundo” de los Sistemas Embebidos: Encender un LED.

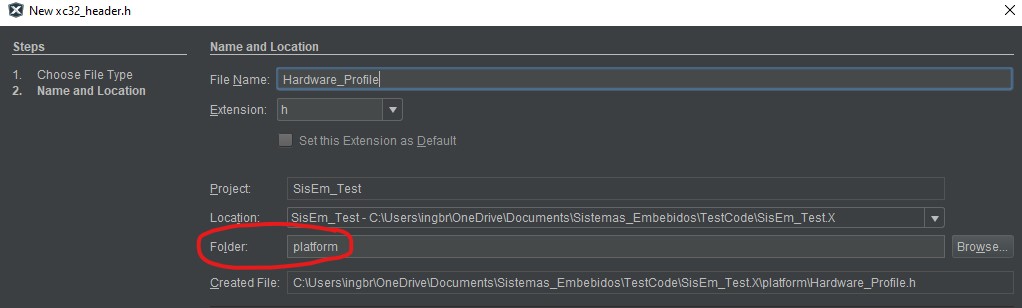
Luego se realizará el “parpadeo” de ambos LEDs de la placa, implementando una función de Delay (bloqueante).

### 1. Creación de archivo de configuración del hardware

1. Para comenzar a trabajar sobre la programación modular, crear una carpeta lógica llamada ***platform*** en el directorio raíz del proyecto, y luego un archivo llamado ***HardwareProfile.h*** en dicha carpeta. Para ello, en el árbol de directorios del proyecto en el IDE, hacer click derecho sobre header files y crear un nuevo directorio y llamarlo platform:



Luego, crear el archivo HardwareProfile.h, haciendo click derecho sobre Header Files, y creando un ***xc32\_header.h***. Agregarlo dentro de la carpeta “física” platform:



1. En el nuevo archivo, reemplazar, en las directivas ***#ifndef*** y ***#define***, la etiqueta *\_EXAMPLE\_FILE\_NAME\_H* por *\_HARDWARE\_PROFILE\_H*. Discutir para que se utilizan estas directivas.

La directiva #define es una condición especial para comprobar si un macro identificador está definido o no. Con esta directiva controlamos a voluntad las zonas de código en que se considerara definido.

La directiva #ifndef es la opuesta a #ifdef. Esta #ifndef verifica si un identificador ha sido definido, o si su definición ha sido eliminada con un #def, la condición es V, de lo contrario es F. Ayuda a la compilación.

1. Dado que ambos LEDs, LEDA y LEDB, están conectados al PIC a sus pines RA7 y RB14 respectivamente, se tienen que configurar ambos pines como salidas. Para ello agregar los siguientes macros en ***HardwareProfile.h***

#define LEDA\_SetDigitalOutput() (TRISAbits.TRISA7 = 0)

#define LEDB\_SetDigitalOutput() (TRISBbits.TRISB14 = 0)

1. Analizar la estructura de datos TRISAbits y ver otra forma de utilizarla. ¿Cómo sería la otra forma de utilizar el unión?

A diferencia de utilizar “TRISAbits.TRISA4 = 1”, lo cual setea directamente el bit deseado, este caso el pin 4, Union compara lo que ya tenía el registro w de TRISAbits y le hace un “AND” del registro nuevo al guardado, ejemplo: TRISAbits.w |= 0b00010000.

1. Basados en los macros anteriores, y sabiendo que los puertos de salida pueden ser controlados mediante el registro *LAT*, crear/definir los siguientes macros:

#define LEDX\_SetHigh() // Pone en nivel alto

#define LEDX\_SetLow() // Pone en nivel bajo #define LEDX\_Toggle() // Invierte de nivel

Reemplazando X por A y B para ambos LEDs.

### 2. Creación de una función de delay

a. Crear una nueva carpeta en el directorio raíz del proyecto llamada ***utils***, y dentro de ella se van a crear dos archivos: ***utils.c*** y ***utils.h***.

Dentro de utils.c, incluir a ***utils.h,*** y crear la siguiente función:

void UT\_delay( void ){ int n=UT\_DELAY\_CYCLES;

while(n>0){ n--;

}

}

Mientras que en utils.h se define la constante **UT\_DELAY\_CYCLES** utilizada en ***utils.c*** como:

#define UT\_DELAY\_CYCLES 500

Y el encabezado de la función:

void UT\_delay( void );

### 3. Implementación del parpadeo de LEDs

a. Incluyendo a ***HardwareProfile.h*** y ***utils.h***, modificar el ***main.c*** para que en la placa queden ambos leds parpadeando indefinidamente.

Webgrafía:

* <https://docs.microsoft.com/en-us/cpp/preprocessor/hash-ifdef-and-hash-ifndef-directives-c-cpp?view=msvc-170>
* <https://ccia.ugr.es/~jfv/ed1/c++/cdrom3/TIC-CD/web/tema17/teoria_4.html>
* <https://www.zator.com/Cpp/E4_9_10e.htm>