Departamento de Ingeniería Electrónica
ETSI de Telecomunicación
Universidad Politécnica de Madrid

Práctica 1:
Introducción al entorno MPLAB X y simulación de programas en ensamblador y C

Autores:

Pedro José Malagón Marzo

Profesores del Dpto. de Ingeniería Electrónica

Fecha última revisión: Abril 2017

Contenido

Tabla de figuras

1. Introducción

La sesión práctica 1 consiste en utilizar realizar programas en ensamblador de MIPS y en C para un PIC de la familia PIC32MX. Los objetivos más importantes de esta práctica son:

- Familiarización con el entorno de desarrollo MPLAB IDE de Microchip:
  - Simulador
  - Depuración de programas
- Consolidación de conceptos básicos vistos en clase:
  - Manipulación de bits
  - Control de periféricos: GPIO

Al igual que en la práctica 0 utilizaremos MPLAB X IDE para editar y simular el programa. Utilizaremos el compilador XC32 para compilar.

En primer lugar abriremos un proyecto existente en ensamblador, con un programa en ensamblador, que iremos modificando y ampliando para conseguir lo que se pide. Aprenderemos a visualizar el estado de los pines GPIO y provocar cambios en las entradas. Además veremos otras utilidades de depuración: variables, registros y memoria.

En segundo lugar crearemos un proyecto en C e implementaremos las mismas funciones que hayamos tenido que hacer en C, comprobando que obtenemos un resultado equivalente.

El resultado será un fichero comprimido (.zip) con un conjunto de capturas de pantalla, un fichero ensamblador y un fichero en C con todas las funciones que se hayan pedido. Por favor, sigue la nomenclatura indicada.

2. Proyecto de ejemplo: ports_control.X

En primer lugar tenemos que abrir el programa MPLAB IDE (buscar en el menú inicio).

Vamos a trabajar con un proyecto de ejemplo proporcionado por Microchip. Se trata del proyecto ports_control.X, que realiza operaciones sobre el puerto A de GPIO. El código original, en otras instalaciones, se puede encontrar en el directorio de instalación de xc32:

PATH_DE_MICROCHIP\xc32\v1.40\examples\assembly_examples\ports_control

En nuestro caso ya está disponible en la carpeta del entorno :

C:\Usuarios\usuario\MPLABXProjects

El proyecto tiene un único fichero fuente ports_control.S, en la carpeta Source Files. Este fichero tiene dos funciones:

- mPORTAClearBits que recibe un único argumento: la máscara con los pines del puerto A que hay que poner a 0.
- main, que prepara el argumento pasado a mPORTAClearBits (máscara para bit 7).

La extensión típica para el fichero ensamblador es .s o .asm. En este caso se utiliza .S (en mayúscula) para poder utilizar el preprocesador de C (así lo interpreta el compilador xc32). Gracias a ello el fichero hace uso de etiquetas (típicas del preprocesador de C) como a0 o

IOPORT_BIT_7, que están definidas en el fichero xc.h. Los ficheros de cabecera incluídos con <> son los del sistema. Uno de los directorios del sistema es

C:\Archivos de programa\Microchip\xc32\v1.40\pic32mx\include

Los registros de control de un puerto de GPIO son:

- TRISx: cada bit controla la dirección del pin (0 es salida, 1 es entrada)
- PORTx: cada bit refleja el valor del pin (0 es nivel bajo, 1 es nivel alto)
- LATx: cada bit refleja el valor del registro de salida del pin. Este valor coincide con el pin si está configurado como salida, y puede no coincidir si es entrada.

Para modificar un solo bit de un registro, manteniendo el resto con su valor original, normalmente es necesario leer el valor actual del registro, realizar una operación binaria (OR o AND) con la máscara adecuada, y guardar el resultado en el registro. Se necesitan 3 operaciones (Read/Modify/Write o RMW) para hacerlo. Si observamos el código, se hace solo en una operación. No se accede al registro LATA en este caso, sino al registro LATACLR. Casi todos los registros de control de periféricos tienen asociados 3 registros que facilitan la modificación de bits en una única instrucción. La función que se realiza depende del sfijo del registro: CLR (para poner bits a 0), SET (para poner bits a 1) e INV (para cambiar bits). Los 4 registros (LATA, LATACLR, LATASET y LATAINV) son consecutivos y cada uno ocupa 32-bits, aunque luego solo algunos de ellos tienen sentido, porque no haya tantas GPIO (se puede ver en la **sección 11.2 CLR, SET and INV Registers del datasheet**).

Los nombres de los registros están disponibles en el *datasheet* del componente. Para que el compilador los pueda utilizar, están definidos los nombres, asociados a las direcciones reales del registro, en el fichero ensamblador

C:\Archivos de programa\Microchip\xc32\v1.40\pic32-libs\proc\32MX250F128B\p32mx250f128b.S

El fichero para C (p32mx250f128b.h), que vimos en clase, con las estructuras de acceso a cada bit de los registros, tiene la palabra extern porque la variable ya está declarada en este fichero con la palabra clave .global NOMBRE_DEL_REGISTRO (.global LATACLR en este caso).

El siguiente paso es generar el código máquina: compilar. Se compila el proyecto marcado como Main Project (**en negrita**). Si no es el abierto, póngalo como Main Project haciendo click en el botón derecho del ratón sobre el nombre del proyecto, y seleccionando la opción Set as Main Project. La compilación se realiza a través del menú *Run -> Build Main Project* () o a través del menú *Run -> Clean and Build Main Project* (). En el segundo caso, se limpian todos los ficheros intermedios generados en compilaciones anteriores y se compila desde cero todo el proyecto. El proyecto debe compilar correctamente.

3. Primer proyecto: p1.X

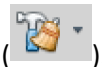
En el anexo 1 detallamos los pasos a seguir para crear un proyecto para nuestro dispositivo utilizando el compilador gratuito XC32, que incluye todas las herramientas para convertir los ficheros en ensamblador o C a código máquina del MIPS disponible en el PIC32. El proyecto incluye un conjunto de etiquetas de preprocesador para indicar el dispositivo y el lenguaje utilizado. Según las etiquetas definidas se accede a una parte del fichero xc.h, tal y como se ha visto en clase.

A continuación vamos a realizar nuestro primer programa, inspirado en el ejemplo. Cree un nuevo proyecto y llámelo p1. Los ficheros fuente, tanto en ensamblador como en C, se muestran en la carpeta *Source Files*. No es posible crear en el entorno un fichero .S mayúscula de forma correcta, por lo que hay que añadir uno ya existente, creado fuera del entorno (en este caso el del ejemplo ports_control). Hacemos click con el botón derecho en la carpeta y seleccionamos *Add Existing Item* ...

Navega por el explorador de archivos hasta seleccionar el fichero del ejemplo, que está en:

C:\Usuarios\usuario\MPLABXProjects\ports_control\source\ports_control.S

Marca las opciones *Relative* (para que siga funcionando si copias el proyecto a otro sitio) y *copy* (para que haga una copia y mantengamos el ejemplo como referencia).

El siguiente paso es generar el código máquina: compilar. La compilación se realiza a través del menú *Run -> Build Main Project* () o a través del menú *Run -> Clean and Build Main Project* ().

3.1. Ejercicio. Modifica ports_control.S

El objetivo es tener una nueva función, mPORTAToggleBits, con un argumento, que invierta los pines (Toggle) del puerto A que se correspondan a los bits del argumento que estén a 1. En una rutina main1 se configurarán los pines del puerto y, en un bucle infinito, se llamará a la función mPORTAToggleBits con un argumento para invertir los 3 bits menos significativos.

Las fases son:

1. Modificar el nombre de la etiqueta main para que sea main0
2. Duplicar la rutina main0 y llamarla main1.
3. Añadir una nueva rutina main, con 3 líneas de código, antes de main0. Tiene que saltar a main1.

```
main:
        j       main1
        nop
```

4. Crear, además de la función mPORTAClearBits, la función mPORTAToggleBits. Esta función, en vez de poner a 0, invierte los bits que están a 1 en el argumento. ¿Se puede simplificar la función utilizando otro registro? Si es así, hazlo.
5. Modificar la rutina main1 para que configure las GPIO correctamente (salidas digitales) y llame, en un bucle infinito, a la función mPORTAToggleBits para modificar los bits 0, 1 y 2 del puerto (RA0, RA1 y RA2). El programa no debería llegar nunca al bucle infinito existente endless.

Para modificar un registro como puede ser TRISA, es importante realizarlo en 3 fases, tal y como se hace en el ejemplo con LATACLR:

1) Cargar en un registro (s0) la dirección con la pseudoinstrucción **la**

2) Cargar en otro registro (s1) la constante que se quiera almacenar (o leer la variable)

3) Almacenar el registro s1 en la dirección almacenada en s0

Una vez el proyecto compile correctamente vamos a simularlo. El simulador tiene asociadas unas ventanas que debemos activar a través del menú *Window -> Simulator*. Activaremos las 3, y apareceran pestañas en la ventana de abajo para poder visualizarlas: *Logic Analyzer, I/O Pins* y *Stimulus.*

En primer lugar seleccionaremos *Logic Analyzer*, que nos permite ver el valor de los pines que seleccionemos, como si hubiésemos puesto un osciloscopio a medir la GPIO. Para seleccionar los pines a visualizar pulsamos el icono  .

Seleccionamos los pines que queremos modificar en el programa: RA0, RA1, RA2 y RA3, quedando el resultado mostrado en la siguiente figura

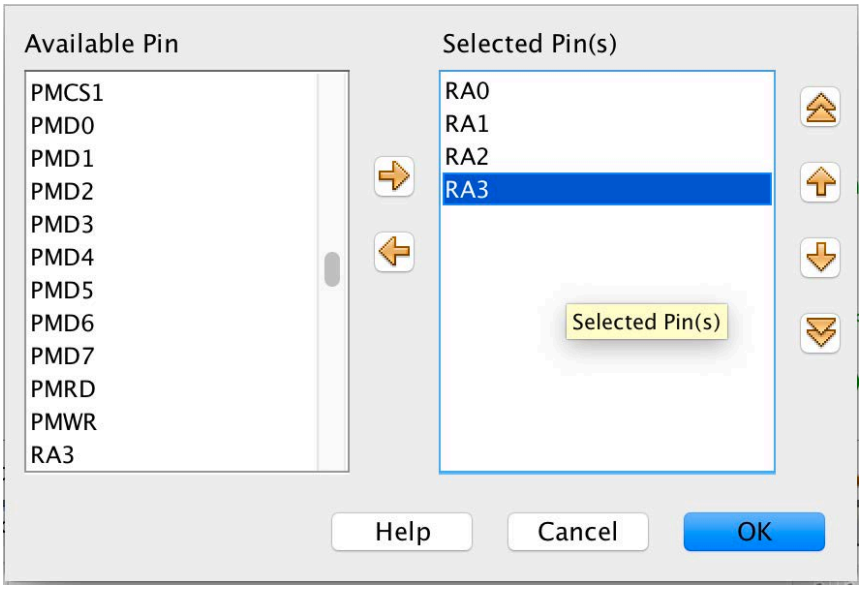


Figura 1. Selección de pines para el Logic Analyzer

Ahora seleccionamos la pestaña *I/O Pins*. Esta pestaña muestra el estado de los pines: modo, valor y módulo asociado. Recordamos que el número de pines es limitado y hay muchos periféricos disponibles. Tanto para señales de entrada como de salida a un módulo se puede elegir entre varios pines como alternativa (**sección 11.3 Peripheral Pin Select en el datasheet**). Para las entradas, los periféricos tienen un registro de control (nombre de la señal más R al final, como INT4R o U2RXR) en el que se elige qué pin de qué puerto se conecta. Para las salidas, cada pin tiene un registro de control (RPA0R o RPB3R) en el que se elige qué señal de salida se conecta a este pin (qué módulo lo controla).

El periférico del conversor analógico-digital (ADC) no es configurable, y se controla con el registro ANSELx (ANSELA, ANSELB, ...): cada bit controla si el pin está siendo utilizado como entrada analógica del ADC (1) o como digital (0).

En la ventana *I/O Pins* añade 4 filas, una para cada pin de interés, quedando el resultado mostrado en la siguiente figura

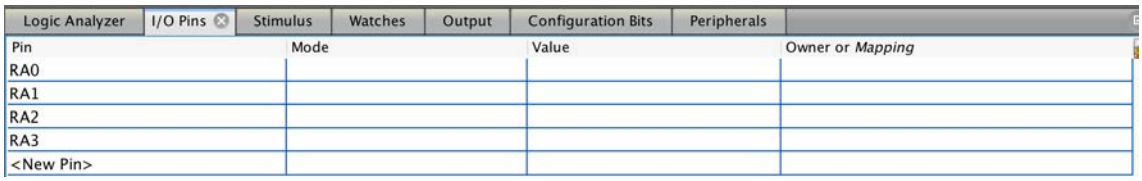


Figura 2. I/O Pins con registros del puerto A

Por último, antes de simular, vamos a poner un punto de parada (*breakpoint*) al comienzo del programa. Los puntos de parada detienen el programa y nos permiten acceder al valor de memoria, registros y variables que tiene en ese momento el programa. De esta manera podemos comprobar si el comportamiento es el adecuado. Los puntos de parada se ponen y se quitan con un simple click de ratón en el número de línea en el que se quiere poner. En este caso, hay que poner un punto de parada en la primera instrucción después del main (**j main1**). El símbolo de stop en vez del número de línea indica que hay un punto de parada.

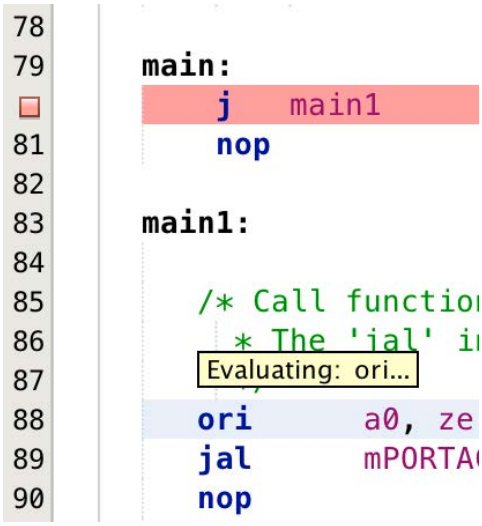


Figura 3. Breakpoint inicial

Ya estamos preparados para simular el programa. Se hace utilizando el menú *Debug -> Debug Main Project* o con el icono  .

En el momento de lanzar el simulador se activan todos los iconos de control de la ejecución del simulador o depurador en la barra de herramientas, tal y como muestra la siguiente figura:

Figura 4. Barra de herramientas del depurador

El anexo 2 explica la función de cada botón del simulador.

Para poder visualizar correctamente los iconos de depuración puede que tengamos que personalizar la barra de herramientas del entorno, a través del menú *View -> Toolbars*. Tenemos que tener seleccionadas las opciones según la siguiente imagen

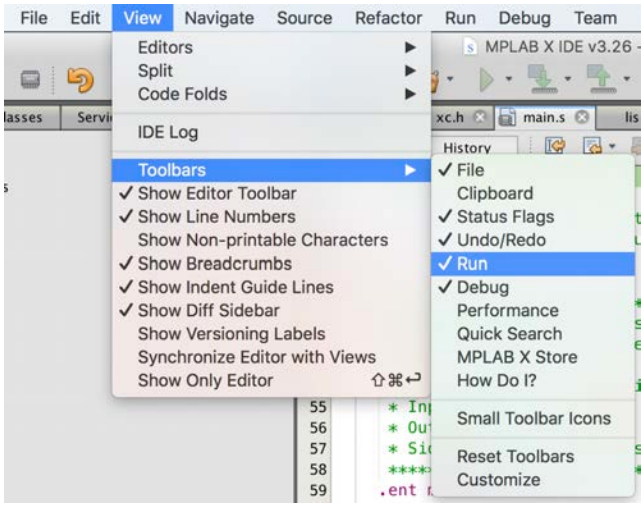


Figura 5. Personalización de la barra de herramientas

Hay que depurar el programa hasta que funcione. Recomendamos que se haga con ejecución paso a paso, comprobando el valor de los pines en *Logic Analyzer* y en *I/O Pins*. Si no es el correcto, ten en cuenta lo que hemos comentado anteriormente sobre la dirección (TRISx) y sobre los ADC (ANSELx).

Una vez terminado, realiza una captura (botón Print pantalla) de la ventana *Logic Analyzer* y otra de *I/O Pins*. Guárdalas en los ficheros **logic_1.jpg** y **io_1.jpg** respectivamente, utilizando el programa Paint de Windows.

4. Entradas en el simulador: Estímulos

En el apartado anterior hemos visto las herramientas disponibles para ver el valor de las salidas. En el simulador podemos asignar valores de entrada de los pines, simulando el valor que pondrían elementos externos. La ventana *Stimulus* nos permite controlarlos de forma cómoda.

Hay 5 pestañas diferentes para simular entradas. En este primer ejemplo vamos a ver el más simple: *Asynchronous*. Esta ventana permite cambiar el valor de un pin mediante un botón (la columna *Fire*). Es necesario seleccionar el pin y la acción que se desea realizar. La figura muestra las posibles acciones que se pueden realizar.

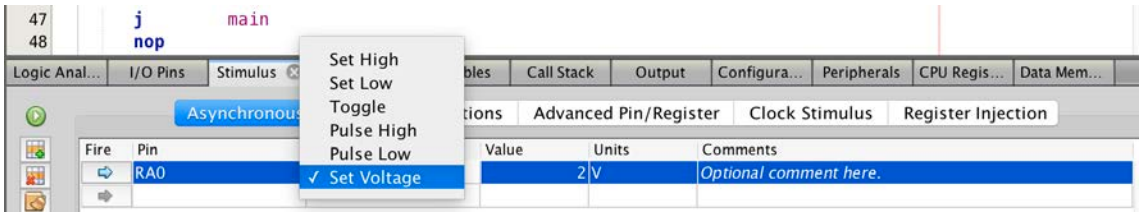


Figura 6. Selección de estímulos

Set High: Pone el pin a 1

Set Low: Pone el pin a 0

Toggle: Cambia el valor del pin

Pulse High: Pone el pin a 1 tantos tiempo como se indique en *Value*. Las unidades de tiempo pueden ser tiempo (ns, us, ms) o ciclos de reloj (independiente de la frecuencia de reloj).

Pulse Low: Pone el pin a 0 tantos tiempo como se indique en *Value*. Las unidades de tiempo pueden ser tiempo (ns, us, ms) o ciclos de reloj (independiente de la frecuencia de reloj).

Set Voltage: Pone el valor de tensión que se desee en V (para simular entradas analógicas).

Añade los pines RA0 a RA3 a la ventana de estímulos. Añade al *Logic Analizer* y al *I/O Pins* los pines RB0 a RB3.

4.1. Ejercicio. Función mPORTA2PORTB.

Debes reutilizar el programa previo. Crea una nueva función mPORTA2PORTB que lea el valor que hay en el puerto A y ponga ese valor en el puerto B. Utiliza la ventana de estímulos para variar su valor y comprobar su correcto funcionamiento. Llama a esta función de forma indefinida desde una nueva rutina main2 (inspirada en main1), a la que se tiene que saltar desde la rutina main.

Una vez hayas terminado realiza una captura (botón Print pantalla) de la ventana **Logic Analyzer** y otra de **I/O Pins**. Guárdalas en los ficheros **logic_2.jpg** y **io_2.jpg**, utilizando el programa Paint de Windows.

5. Big Endian y Little Endian

Para distinguir si un sistema es *big-endian* o *little-endian* tenemos que ver cuál es el orden en el que se almacenan los bytes en memoria: el más significativo en la dirección más baja (*big-endian*) o el menos significativo en la dirección más baja (*little-endian*). Si tenemos en cuenta que podemos hacer operaciones sobre 32-bits, sobre 16-bits y sobre 8-bits, podemos comprobar fácilmente el *endianness* del sistema.

En clase hemos hecho la operación almacenando un valor de 32-bits en memoria y cargando en un registro el byte 1 de esa zona de memoria. Utilizando los estímulos que acabamos de ver y el puerto A ¿puedes comprobarlo? Ten en cuenta que los registros del puerto A tienen 32-bits, aunque sólo haya 5 disponibles en nuestro microcontrolador. En una operación de lectura los pines no disponibles dan siempre 0.

5.1. Ejercicio. Endianness

Crea una nueva función endianness que te ayude en la tarea. Añade una rutina main3 (inspirada en main2) que llame a la función de forma indefinida, y modifica main para que salte a main3. La función, parecida a la anterior, debe utilizar carga de 8-bits en vez de 32-bits (**lb** en vez de **lw**). Al leer un solo byte, si es el menos significativo tendrá el valor de los estímulos, mientras que el más significativo tendrá todo 0. ¿El sistema es *Little Endian* o *Big Endian*? Adjunta una captura del simulador que justifique la respuesta. El nombre de la captura tiene que ser **big.jpg** si consideras que es *big endian* y **little.jpg** si consideras que es *little endian*.

6. Funciones en C

Desde un fichero ensamblador se puede llamar a funciones de un fichero en C y viceversa. Cada fichero se compila por separado y, al enlazar (linker), se genera un fichero ejecutable que contiene todas las funciones. Si desde un fichero se llama a una función de la que no se tiene implementación, el *linker* se queja con el mensaje *undefined reference to*.

6.1. Ejercicio. Implementación en C

El objetivo de este ejercicio es implementar funciones equivalentes a las que hemos hecho en ensamblador en un fichero C, que llamamos ports_control.c. Para crear el fichero seleccionamos la carpeta *Source Files* y seleccionamos el menú *File -> New File*. Escogemos el tipo *C -> C Source File*.

En este fichero tenemos que hacer la implementación de las siguientes funciones:

```
void mPORTAClearBitsenC (int arg);

void mPORTAToggleBitsenC (int arg);

void mPORTA2PORTBenC (void);

void endiannessenC (void);
```

Las funciones ya suponen que los pines se han configurado correctamente.

En el caso de endiannessenC, la mejor manera de cargar un byte de una dirección de memoria data es almacenar la dirección en una variable de tipo char*. Leer el contenido de esta variable es equivalente a cargar un byte de memoria en registro. Para forzar la asignación de la dirección del registro PORTA, que es de tipo int, hay que hacer un *casting*, para que el programador indique que sabe que está cambiando el tipo (si no sale un aviso). La instrucción para almacenar la dirección de PORTA quedaría como sigue:

```
char* ptr = (char*)(&PORTA);
```

Para poder llamar a las funciones y comprobar el funcionamiento debemos sustituir en el fichero ports_control.S todas las llamadas a las funciones originales. Añadiremos al final el sufijo enC para que llame a las funciones implementadas en C. Al depurar paso a paso el simulador cambia de fichero cuando llega. Comprueba que todas tengan el mismo resultado. Es posible ver el código ensamblador generado por el compilador en el menú *Window -> Debugging -> Disassembly*. Se ve que cada instrucción en C se corresponde con varias en ensamblador.

Todas las imágenes y ficheros generados, con la nomenclatura indicada, se tienen que añadir a un fichero P1_NOMBRE_APELLIDO_puesto_XX.zip. NOMBRE tiene que ser sustituido por el nombre del alumno, APELLIDO por el apellido y XX tiene que ser sustituido por el número de puesto. Ambos miembros de la pareja tienen que generar el fichero (se puede copiar y cambiar el nombre).

El fichero .zip se puede crear seleccionando los ficheros, pulsando el botón derecho, seleccionando Enviar a -> fichero comprimido.

ANEXO 1: Creación de un proyecto Blink.X

Se crea un proyecto nuevo a través del menú *File -> New project*, o utilizando el icono 

1. Choose project: Microchip Embedded -> Standalone Project

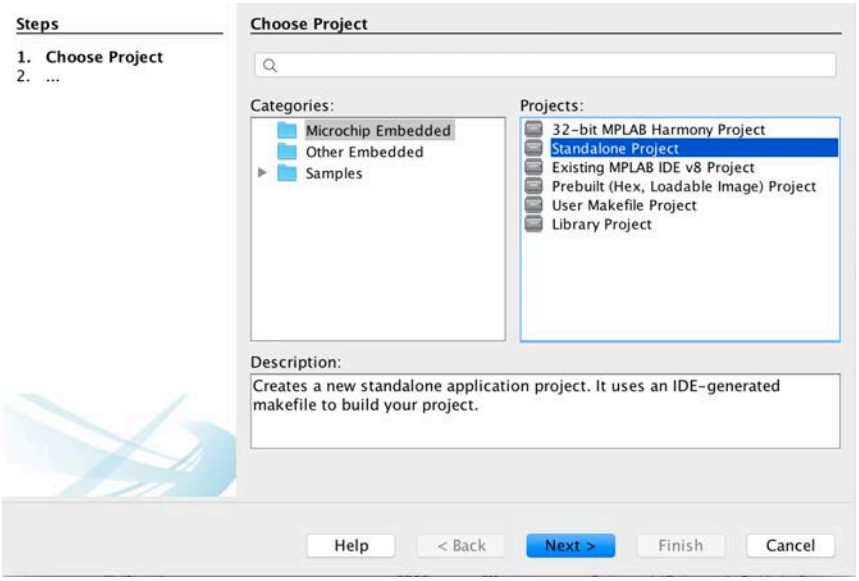


Figura 7. Tipo de proyecto para PIC32

2. Select Device: Family 32-bit MCUs (PIC32), Device: PIC32MX250F128B

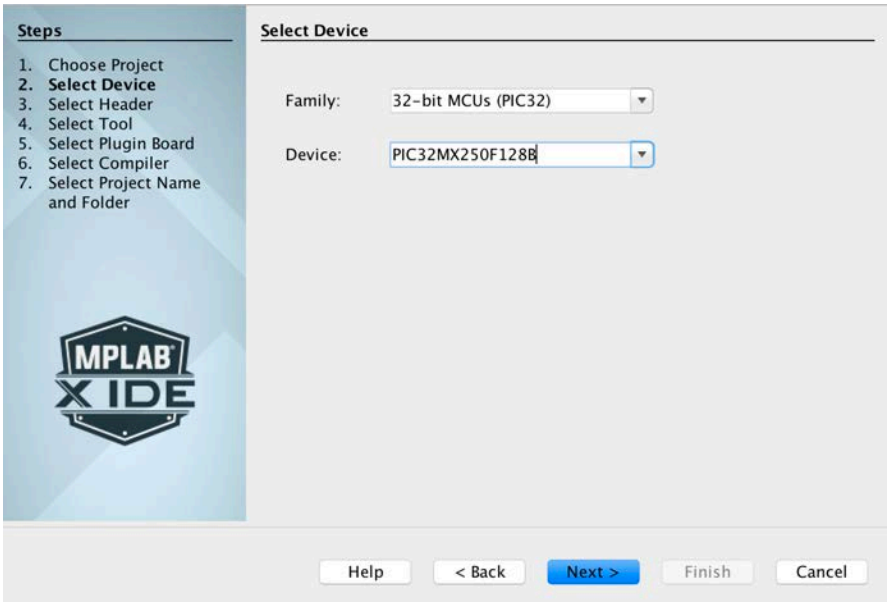


Figura 8. Selección de dispositivo

3. Select tool: Simulator

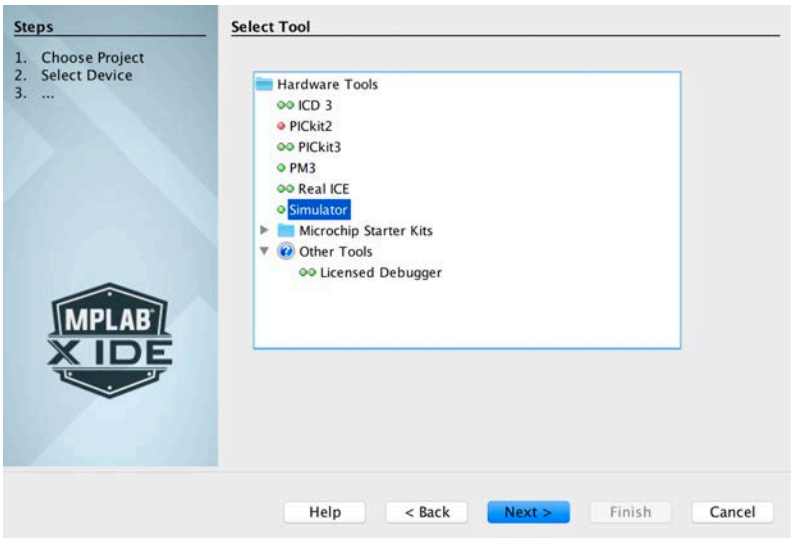


Figura 9. Escoger el simulador como hardware

6. Select compiler: XC32 (v1.40) (C:\Archivos de programa\Microchip\xc32\...)

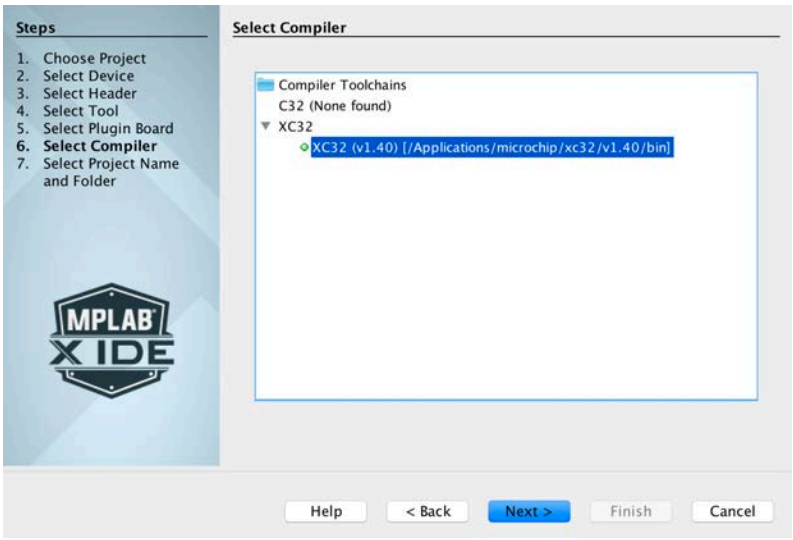


Figura 10. Escoger el compilador XC32

7. Select Project Name and Folder: dejar todos los campos por defecto y poner el nombre que corresponda. Por ejemplo el programa se puede llamar blink (típico primer programa).

El campo "Set as main project" indica que este es el proyecto que se va a compilar y simular cuando se utilicen las herramientas del menú.

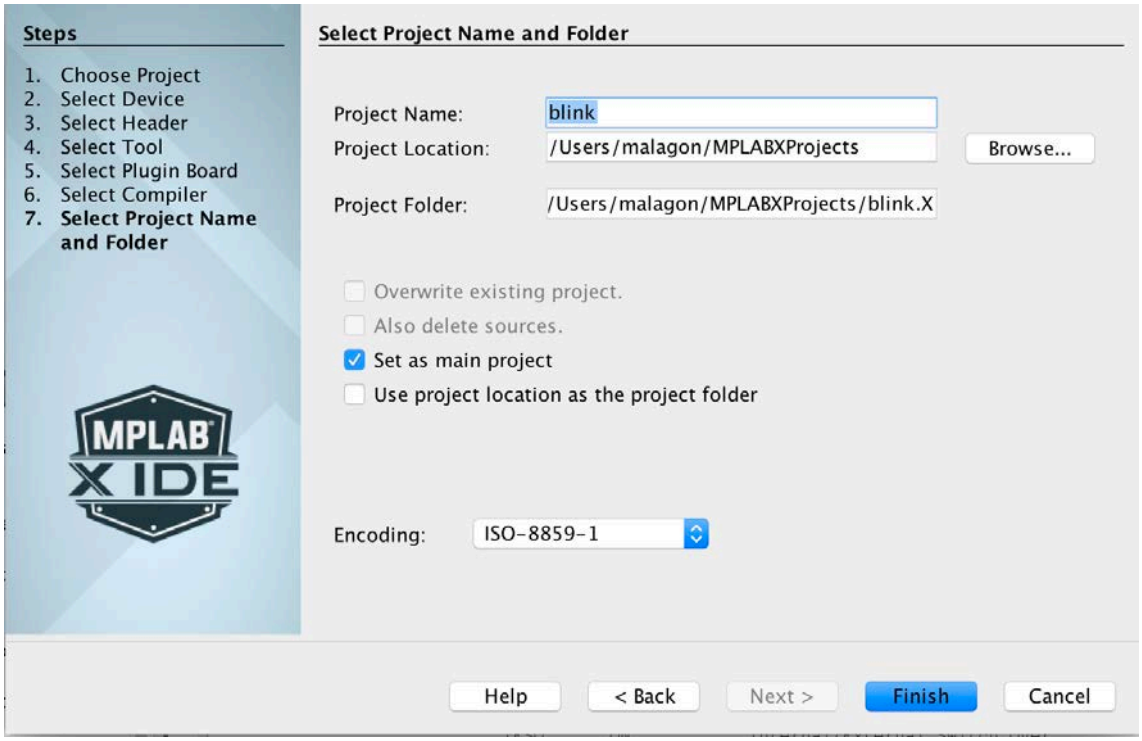


Figura 11. Nombre de proyecto y establecer como Main Project

Pulsamos *Finish*. En este momento aparece en el lateral izquierdo unas carpetas generadas por el proyecto.

ANEXO 2: Botones del simulador

 Para el proceso de depuración. Detiene el programa.

 Pause. Si el programa se está ejecutando de forma libre (no paso a paso) se puede parar, además de con los puntos de parada, pulsando este botón. Esto activa el resto de opciones disponibles.

 Reset. Comienza de nuevo el programa y la depuración.

 Continue. Ejecuta hasta que encuentre un punto de parada.

 Step over. Ejecuta la siguiente instrucción. Si es una llamada a función, la ejecuta como si fuera una única instrucción (en ensamblador, jal)

 Step into. Ejecuta la siguiente instrucción. Si es una llamada a función, sigue la ejecución paso a paso dentro de la función.

 Step out. Ejecuta hasta que encuentre un retorno de función (en ensamblador jr $ra)

 Run to cursor. Ejecuta hasta que llegue al cursor. Es equivalente a poner un punto de parada donde está el cursor y quitarlo según se para.

 Set PC to cursor. Empieza a ejecutar desde la instrucción en la que se ha puesto el cursor. Es interesante para depurar funciones concretas.

 Focus cursor at PC. Pone en la pantalla la instrucción que se va a ejecutar a continuación.