Sistemas con Microprocesadores

Práctica 6: Desarrollo y simulación con el entorno de desarrollo del fabricante del microcontrolador (MPLAB X).

Resumen:

Implementaremos en C un programa multiplicador de frecuencia para el microcontrolador ATmega328P. El microcontrolador recibirá una señal de reloj por la patilla PC0 y generará una señal de una frecuencia 4 veces mayor en la patilla PB0.

Entrega:

Entregar un PDF de 1 página aproximadamente que contenga:

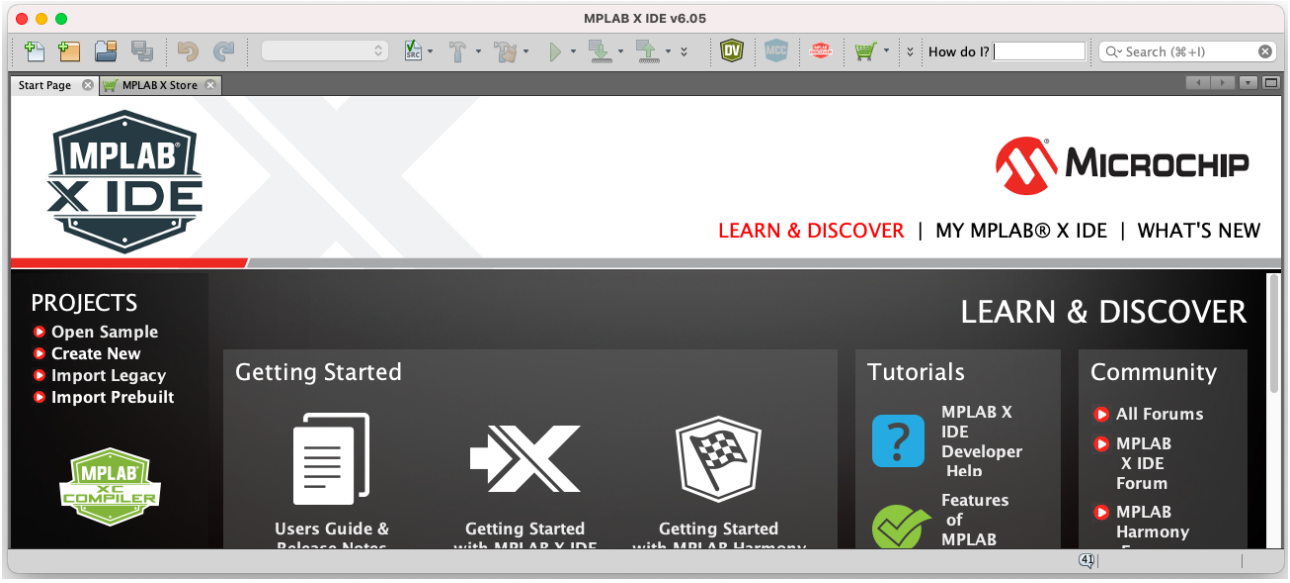
- Nombre y apellidos.
- Código fuente del programa (pegado como texto, no como imagen).
- Captura de la ventana de MPLAB X mostrando los cronogramas de la señal de entrada y la señal de salida (del cuádruple de frecuencia).

(Antes de pegar el código fuente en el documento PDF lo colorearemos, por ejemplo, utilizando la página: https://highlight.hohli.com/?language=cpp)

- Entregar también el fichero .c con el código del multiplicador.

Introducción:

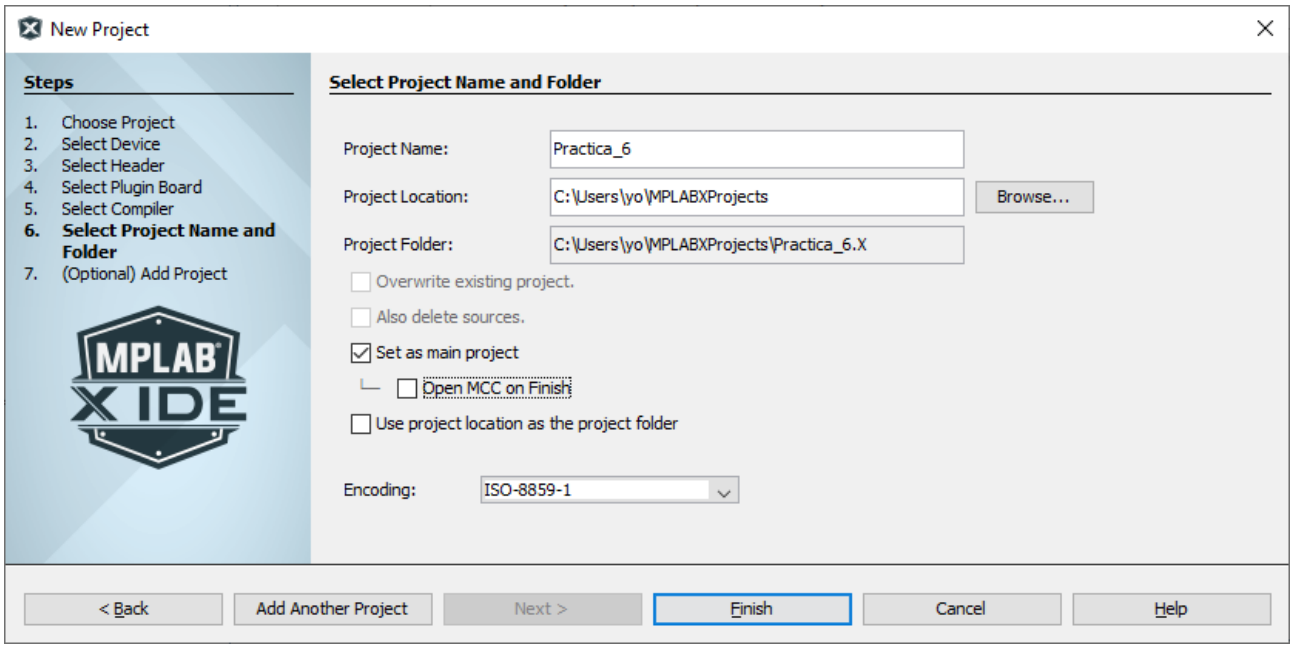
MPLAB® X es un programa que implementa un entorno de desarrollo integrado (IDE) que se utiliza para desarrollar aplicaciones para microcontroladores de la empresa Microchip (incluidos los de la marca Atmel) y controladores de señales digitales (DSC).



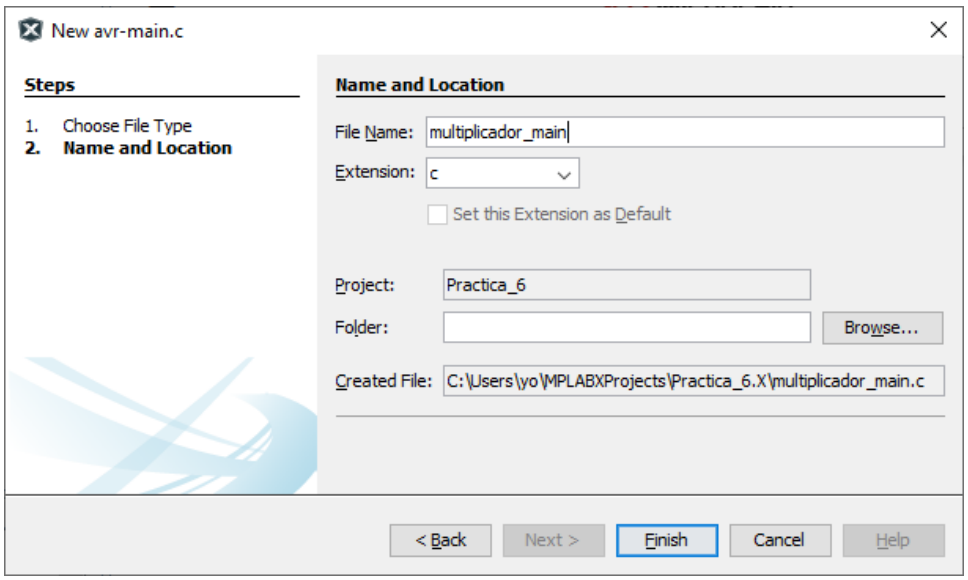
Desarrollo:

Para el desarrollo de esta práctica podemos utilizar un ordenador del aula de prácticas arrancando la imagen de disco *smp10*. También podemos realizar la práctica en nuestro ordenador personal. En este caso, descargamos e instalamos la última versión del entorno MPLAB X. Durante el proceso de instalación podemos excluir el soporte para microcontroladores de 16 y 32 bits para disminuir el espacio consumido en disco. También debemos descargar e instalar la última versión del compilador de C de MPLAB para microcontroladores de 8 bits PIC y AVR: XC8. El asistente de instalación de MPLAB nos puede ofrecer hacer la instalación del compilador automáticamente al final. Ambos ficheros de instalación están disponibles en el sitio web microchip.com. Al arrancar MPLAB nos puede ofrecer instalar actualizaciones. Es probable que nos resulten beneficiosas si vemos que nos da fallos o en ocasiones deja de responder.

Una vez dentro de MPLAB crearemos un proyecto nuevo (Menú *File -> New Project*). En el asistente de creación de proyectos seleccionamos un proyecto de tipo *Application Project(s)*, el cual está dentro de la categoría *Microchip Embedded*. En el siguiente paso especificamos el microcontrolador del proyecto: Familia: *8-bit AVR MCUs* y dispositivo: ATmega328P. También indicamos la herramienta que utilizaremos de depuración/simulación, en nuestro caso: *Simulator*. En el siguiente paso indicamos el compilador a usar, que será alguna versión del compilador XC8 (en la categoría XC8). Finalmente, le damos un nombre al proyecto y deseleccionamos la opción *Open MCC on Finish*, ya que no utilizaremos el configurador de código para generar código fuente.



Una vez creado, en la parte superior del panel de la izquierda del MPLAB nos aparecerá el árbol de nuestro proyecto (inicialmente sin ficheros). Con el botón derecho del ratón pinchamos en la carpeta *Source Files*, y en el menú contextual seleccionamos *New -> avr-main.c*. En la ventana del asistente le damos un nombre al fichero a crear y finalizamos.



Ya podemos empezar a escribir el código fuente de nuestro programa. El asistente nos incluye un código provisional mínimo en el fichero creado. Vamos a modificarlo para crear un sencillo programa que genera una señal cuadrada en la patilla PB0 del microcontrolador. Esta patilla está conectada al pin 8 en la placa Arduino UNO (se puede ver en el esquemático de la placa: Arduino_Uno_Rev3-schematic.pdf). Primero vamos a incluir en nuestro código un fichero de cabecera para poder llamar a funciones para crear retardos (`util/delay.h`). Justo antes de la línea donde incluimos este fichero declararemos la macro `F_CPU` para que el código de este fichero de cabecera conozca la frecuencia de reloj de nuestro microcontrolador, de esta forma puede calibrar los retardos generados:

```
#include <avr/io.h>
#define F_CPU 16000000 // 16 Mhz (frec. cristal del Arduino UNO)
#include <util/delay.h>

#define DDR_PUERTO_SALIDA DDRB // Regist. de direcc. puerto B (PB)
#define PUERTO_SALIDA PORTB // Registro de salida del puerto B

int main(void)
{
   // Configura la patilla PB0 como salida mediante el bit menos
   // significativo del registro DDR
   DDR_PUERTO_SALIDA = 0x01;
   uint8_t valor_puerto = 0x00;
   while (1)
   {
      PUERTO_SALIDA = valor_puerto; // Establece valor de salida
      // Cambia el valor del bit menos significativo usando XOR
      valor_puerto = valor_puerto ^ 0x01;
      _delay_us(10);
   }
}
```

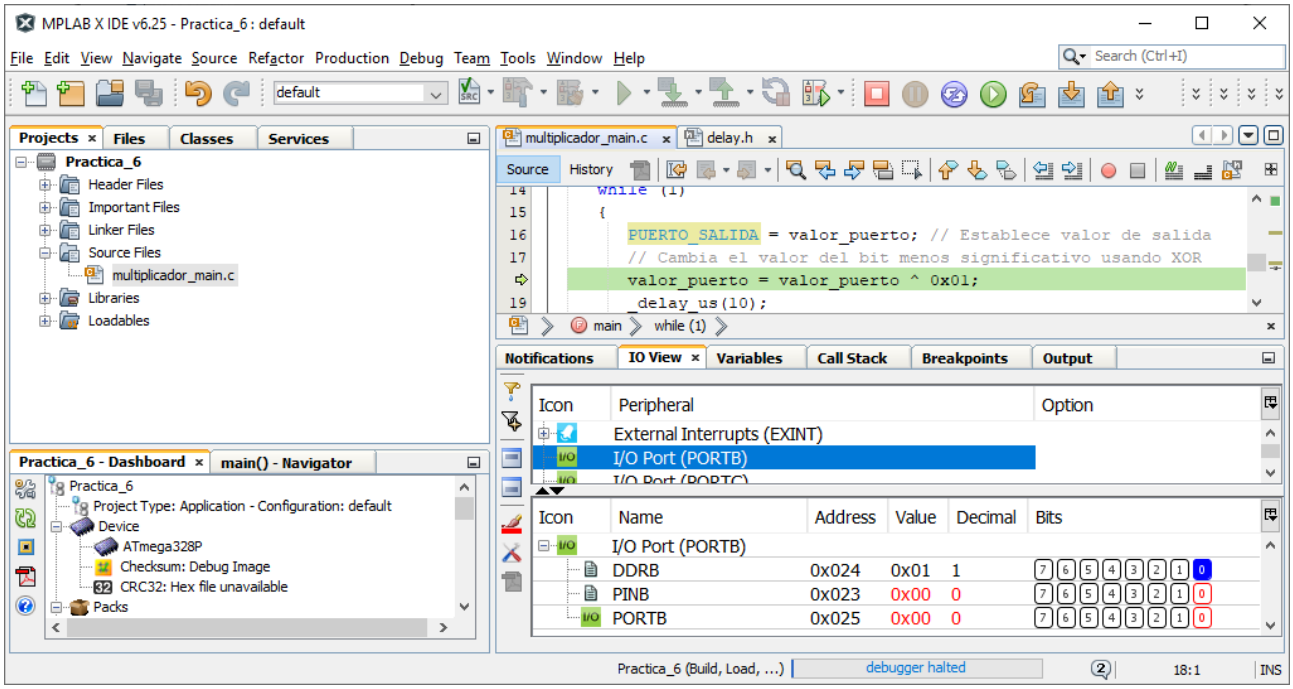
Para configurar la patilla PB0 como salida utilizamos el registro DDR (*Data Direction Register*) correspondiente al puerto B (`DDRB`) y estableceremos el bit menos significativo (el bit 0) a 1. A continuación ya podemos modificar el valor de la patilla mediante el bit menos significativo del registro del puerto B (`PORTB`). En nuestro código utilizamos la operación de bit XOR para cambiar

el valor del bit en cada iteración. Finalmente, utilizamos la función de la biblioteca de AVR para introducir un retardo de 10 microsegundos en cada iteración.

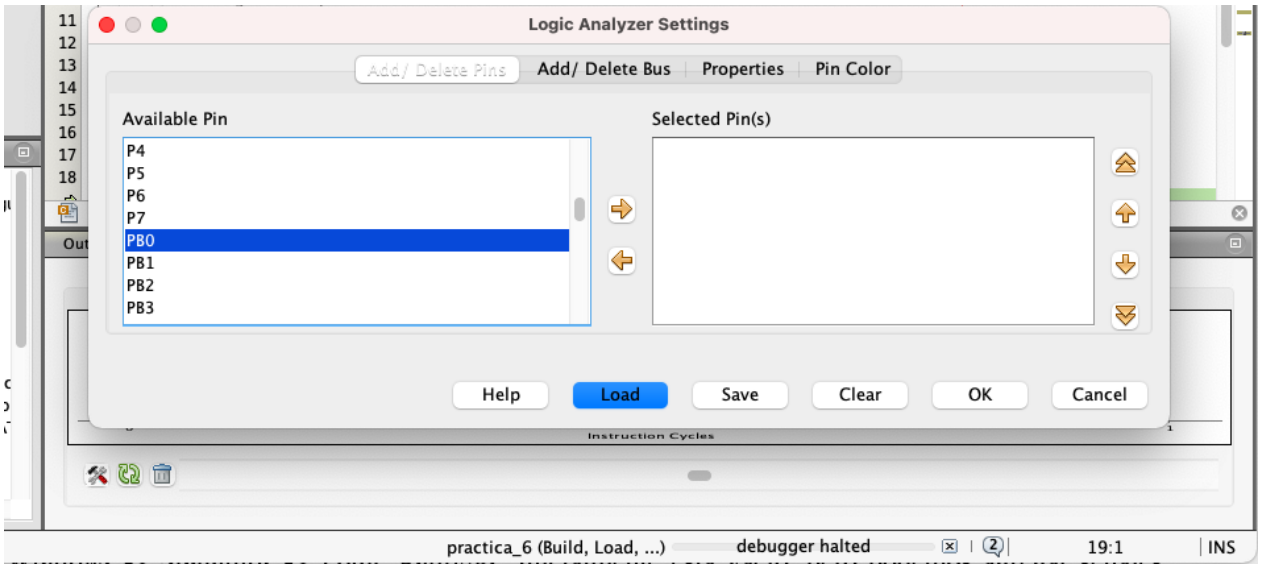
Ya podemos utilizar la opción *Debug Main Project* del menú *Debug* para compilar e iniciar la simulación. Si se ha compilado sin errores, podemos pausar la simulación iniciada con la opción *Pause* del mismo menú. Es probable que la ejecución se detenga en un punto del programa dentro de una función de las bibliotecas. Para salir de ahí, situamos el puntero del ratón sobre una línea de código dentro del bucle `while` que haya generado código máquina (por ejemplo, la que le asigna un valor al puerto), pulsamos el botón derecho del ratón y en el menú contextual elegimos *Run to Cursor*. Si no se nos detiene, podemos establecer un punto de ruptura en el mismo menú contextual.

Por defecto nuestro programa se compila con la optimización activada, por tanto, hay líneas de código fuente de nuestro programa que podrían no haberse transformado en código máquina. Esto puede impedirnos detener la ejecución del programa en estas líneas. La optimización de la compilación se podría desactivar (finalizando previamente la simulación) en la opción *Set Project Configuration* del menú *Production* (*Conf: [default] -> XC8 Global Options -> XC8 Compiler -> Option categories: Optimizations -> Optimization level: 0*), pero para esta práctica la dejaremos activada para que nuestro programa sea rápido.

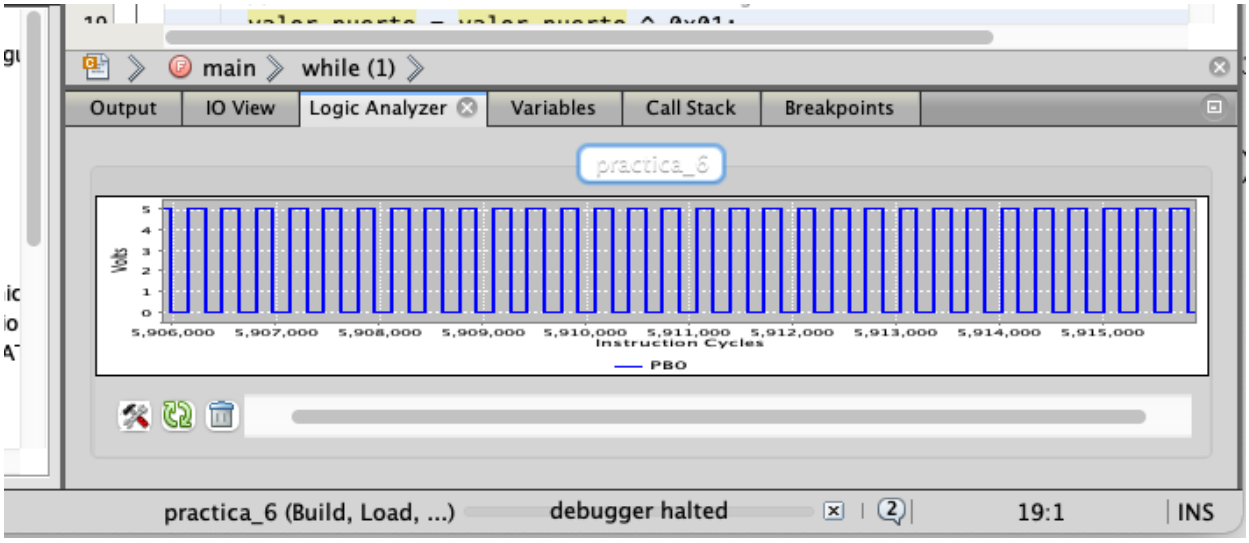
Una vez pausada la simulación podemos ver el valor de las variables situando el puntero del ratón sobre ellas en el código y esperando un par de segundos. También podemos inspeccionar el valor que toman los registros del microcontrolador abriendo la pestaña *IO View* en el menú *Windows -> Debugging -> IO View*. En esta pestaña podemos seleccionar *I/O Port (PORTB)* y veremos como va cambiando el valor (columna *Value*) del puerto `PORTB` al ir ejecutando el programa paso a paso pulsado repetidamente la tecla F8.



Para ver cómo va cambiando la señal generada por la patilla PB0 a lo largo del tiempo utilizaremos un cronograma. Estos cronogramas los podemos ver abriendo la pestaña *Logic Analyzer* en el menú *Windows -> Simulator -> Logic Analyzer*. Inicialmente aparece vacío, pero podemos agregar una o más señales pulsando el botón con el icono de un martillo y una llave (abajo a la izquierda de la nueva pestaña). Se nos abre una ventana de configuración, seleccionamos la patilla PB0 y pulsamos la flecha que apunta a la derecha para añadirla:



A continuación, podemos pinchar la pestaña *Pin Color* de esta misma ventana para asignarle el color deseado a la señal en el cronograma. Y finalmente, cerramos la ventana con OK.
Para ver la señal podemos reanudar la simulación (menú *Debug -> Continue*), esperar un par de segundos, y pausarla de nuevo (menú *Debug -> Pause*):

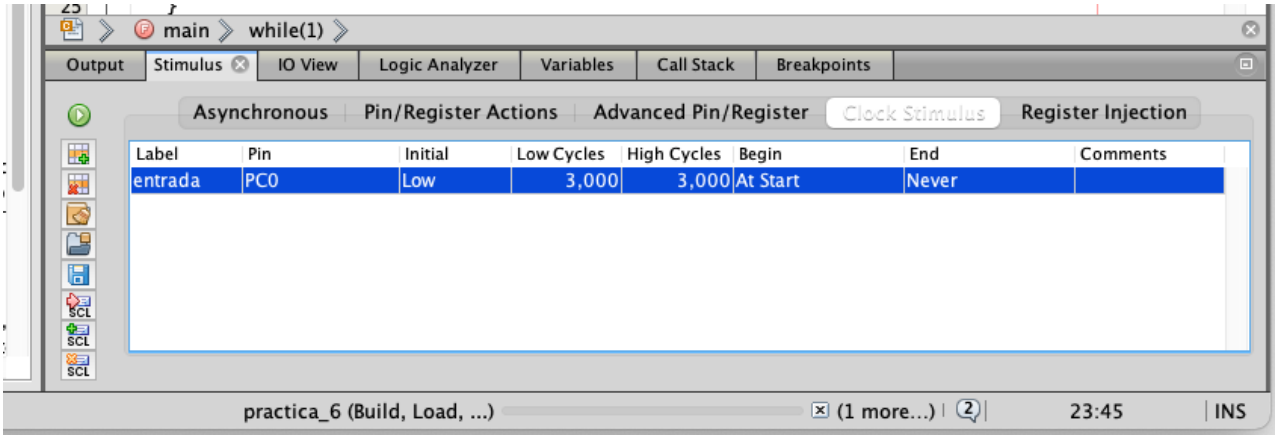


En este panel, con el ratón podemos seleccionar un tramo de la señal para ampliarla, o podemos volver a la vista inicial usando el menú contextual.

Ahora veremos cómo podemos introducir señales del exterior (estímulos) en las patillas del microcontrolador. Pero antes, para ver el efecto de estos estímulos, vamos a ampliar nuestro programa y configurar la patilla PC0 como entrada (esta patilla está conectada al pin A0 en la placa Arduino UNO). Para esto utilizaremos el registro DDR del puerto C de forma parecida a como configuramos el puerto B de salida. Y dentro del bucle `while` añadiremos otro bucle `while` el cual pausará el programa mientras esta patilla esté a 0. Para hacer lecturas de una patilla utilizamos otro registro diferente al usado para escritura: el registro PIN (*Port Input register*). En concreto, para la patilla PC0 leemos el bit menos significativo del registro `PINC`.

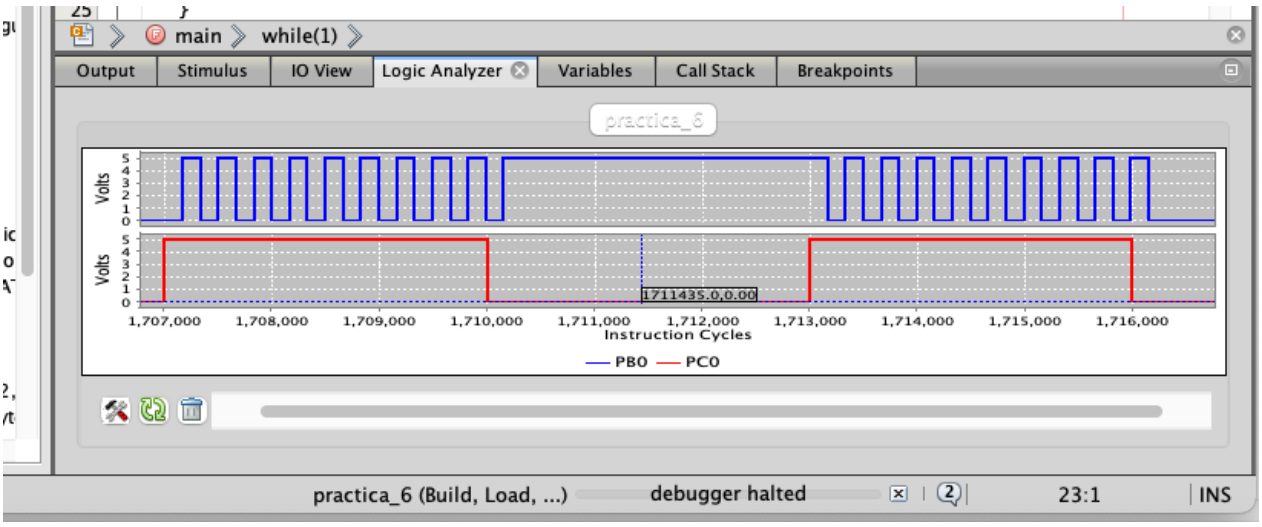
```
    while((PUERTO_ENTRADA & 0x01) == 0); // Espera un valor alto
```

Para añadir los estímulos abriremos la pestaña *Stimulus* mediante el menú *Windows -> Simulator -> Stimulus*. Tenemos disponibles distintos tipos de estímulos, por ejemplo, los asíncronos, los cuales se activan manualmente cuando pulsamos un botón. Pero en nuestro caso utilizaremos una señal de reloj como estímulo. Para esto, pulsamos la opción *Clock Stimulus* (dentro de la misma pestaña *Stimulus*). Nos aparece una tabla en la que rellenaremos la primera fila con un nombre para el reloj, la patilla que lo va a recibir (*Pin* PC0), y el periodo de la señal en baja y en alta en ciclos (*Low Cycles*: 3000, *High Cycles*: 3000).



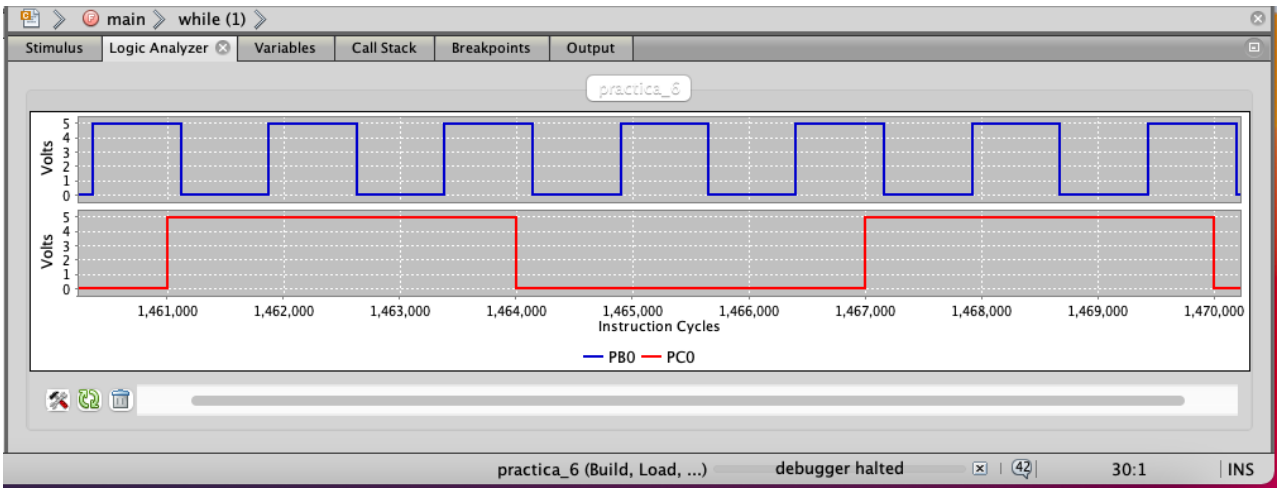
Es imprescindible activar el estímulo pulsando el botón redondo y verde a la izquierda de la pestaña de estímulos. Una segunda pulsación de este botón desactivaría el estímulo.

Es conveniente añadir la señal generada por el estímulo a la pestaña de cronogramas. Para esto, finalizamos la simulación (menú *Debug -> Finish Debugging Session*), volvemos a la pestaña *Logic Analyzer* y añadimos la patilla PC0 con un color diferente a la señal PB0. Iniciando de nuevo la simulación (menú *Debug -> Debug Main Project*), esperando un par de segundos y pausándola (menú *Debug -> Pause*) podemos ver las nuevas señales en el cronograma:



En esta práctica el programa debe tomar la señal estímulo a través de la patilla PC0 y generar una señal del cuádruple de frecuencia mediante la patilla PB0. Utilizaremos como estímulo la señal de reloj de 3000+3000 ciclos de reloj. Dado que esta señal tiene bastante frecuencia eliminaremos de nuestro bucle código innecesario que lo haga iterar menos rápido, como el retardo del final.

Una posible implementación para generar la señal de salida mediante software utilizaría un único bucle `while`. Dentro de este bucle se mediría repetidamente el periodo de la señal de entrada y se conmutaría la patilla de salida:
Para esto, creamos una variable contador, que puede ser un entero sin signo, la cual incrementamos en cada iteración del bucle (un tamaño de entero corto (16 bits) es más que suficiente para todos los contadores de este programa).
Creamos una variable de estado para comprobar cuándo ha habido un cambio de estado en la patilla de entrada (como hacíamos para detectar pulsaciones de botones, pero detectando cambios de 0 a 1 y de 1 a 0). En cada cambio de estado consultamos la variable contador y a continuación la reiniciamos a 0. Este valor consultado del contador lo dividimos entre 4 (con desplazamiento de bits) y lo almacenamos en otra variable. De esta forma, en esta otra variable tendríamos siempre una medida (de un octavo) del periodo total de la señal de entrada (medido en número de iteraciones del bucle).
Para conmutar el valor de la patilla de salida lo hacemos como en el primer código de ejemplo de este guion, pero solamente conmutamos cada *x* iteraciones del bucle, donde *x* es la medida de un octavo del periodo de la señal de entrada. Es decir, necesitamos otra variable para contar cuántas iteraciones han pasado desde la última conmutación.
Con esta implementación la frecuencia de la señal generada iría adaptándose continuamente a la frecuencia que tenga la señal de entrada. El resultado sería el siguiente:



Para aumentar la velocidad de ejecución, y por tanto la frecuencia de la señal que es capaz de generar el microcontrolador, procuraremos no utilizar variables de más tamaño del que necesitemos, y evitaremos abusar de operaciones costosas computacionalmente como la división (ya que ATmega328P no tiene instrucciones para dividir por hardware).
Si en algún momento dejan de visualizarse las señales en los cronogramas, es posible que tengamos que detener la simulación (menú *Debug -> Finish Debugging Session*) e iniciarla de nuevo en lugar de simplemente pausarla y reanudarla.

Si cuando intentamos abrir MPLAB se cuelga y no termina de iniciarse o no termina de cargar nuestro proyecto, es posible que haya que reiniciar el ordenador. Si esto no resuelve el problema, es posible que se hayan corrompido los ficheros de configuración de MPLAB o del proyecto que hemos creado. Para solucionar esto podemos cerrar MPLAB y borrar sus ficheros de configuración para que cree unos nuevos al iniciarse: https://microchipdeveloper.com/mplabx:persistence-data También podemos cerrar MPLAB, borrar el directorio del proyecto que hemos creado y crear un nuevo proyecto (haciendo previamente una copia de los ficheros de código fuente fuera del directorio).