Departamento de Ingeniería Electrónica
ETSI de Telecomunicación
Universidad Politécnica de Madrid

Práctica 2:
Simulación de un pulsómetro a partir de una señal de ECG implementado en C

Autores:

Pedro José Malagón Marzo

Profesores del Dpto. de Ingeniería Electrónica

Fecha última revisión: Mayo 2017

Contenido

1. Introducción

La sesión práctica 2 consiste en utilizar comprender un programa en C para un PIC de la familia PIC32MX, que implementa un pulsómetro utilizando periféricos del PIC32. Los objetivos más importantes de esta práctica son:

- Familiarización con el entorno de desarrollo MPLAB IDE de Microchip:
  - Simulador: consola y estímulos
  - Depuración de programas
- Aplicación y consolidación de configuración en C de periféricos vistos en clase:
  - GPIO entrada (botón) y salida (zumbador/pitido):
  - Interrupciones: habilitación, configuración y función ISR
  - Temporizador: interrupción periódica
  - ADC: activación automática con timer e interrupción
  - Puerto serie: consola

Al igual que en la práctica 1 utilizaremos MPLAB X IDE para editar y simular el programa. Utilizaremos el compilador XC32 para compilar.

El sistema presentado es un pulsómetro implementado en C para PIC32 que detecta la forma QRS en la señal de ECG. El sistema utiliza el algoritmo Pan-Tompkins[1] para detección de la curva QRS. Este algoritmo aplica un filtro paso alto simple, para descartar la desviación en continua, y mide la energía disponible de la señal resultante en una ventana temporal (filtro paso bajo). Al superar un umbral calculado de forma dinámica, se indica que se ha detectado una nueva forma QRS, siempre que haya pasado una ventana de tiempo respecto de la anterior. La implementación del algoritmo se ha adaptado de un proyecto disponible en github[2]. En la práctica no vamos a entrar en detalle del algoritmo. El alumno interesado puede analizarlo en otro momento si lo considera de interés, para relacionarlo con asignaturas de procesado de señal o de señales biomédicas.

El proyecto que contiene el programa está disponible en moodle en el fichero comprimido **beat_rate.zip** y se llama **beat_rate.X**. En él podemos encontrar distintos ficheros que implementan las funciones de soporte de los distintos módulos. El nombre de cada fichero refleja su funcionalidad. Parte de los ficheros está sin implementar, para que el alumno lo complete según se le vaya preguntando.

Para resolver a las preguntas puede consultar las transparencias de clase, el datasheet del microcontrolador PIC32MX250F128B o el datasheet de cada uno de los módulos implicados.

Como primer paso, descarga el fichero del proyecto y descomprime su contenido en la carpeta C:\Usuarios\usuario\MPLABXProjects. Si no aparece el proyecto puede utilizar el menú *File -> Open project* para abrirlo. Una vez abierto póngalo como proyecto principal, para compilarlo y simularlo (seleccionar el proyecto, botón derecho y seleccionar *Set as main project*).

---

[1] PAN.J, TOMPKINS. W.J, "A Real-Time QRS Detection Algorithm". IEEE TRANSACTIONS ON BIOMEDICAL ENGINEERING, MARCH 1985

2 https://github.com/blakeMilner/real_time_QRS_detection.git

En la parte inferior izquierda de la pantalla hay un *dashboard* con datos del proyecto. Se puede pulsar el icono  para abrir las opciones del proyecto. Podéis seleccionar el simulador. Hay dos campos muy importantes para este proyecto:

1. *Oscillator Options: 1 MHz*
2. *Uart2 IO Options: Enable Uart2 IO, Output Window*

Fuera de la configuración, 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.*

El resultado de la práctica consiste en rellenar el fichero practica2_soluciones.doc y completar el código en C del proyecto. Se subirá el fichero doc y un fichero comprimido con la carpeta del proyecto, similar al que habéis descargado. Para comprimir, con el explorador de archivos se navega hasta la carpeta c:\Usuarios\usuario\MPLABXProjects, se selecciona la carpeta beat_rate.c, se hace click en el botón derecho y se envía a fichero comprimido, con el nombre P2_NOMBRE_APELLIDO_puesto_XX.zip.

2. Fichero main.c

El fichero coordina la inicialización del sistema, llamando a las funciones de inicialización de cada uno de los módulos, e implementa una máquina de estados de la aplicación:

- `INIT`: estado en el que entra el dispositivo en el arranque. Produce una secuencia de 3 pitidos para indicar al usuario que ha arrancado y pasa al siguiente estado, `NOT_SAMPLING`.

- `NOT_SAMPLING`: estado en el que el dispositivo está esperando a que el usuario pulse un botón para empezar una medida. Mientras tanto, está en modo de bajo consumo. Al pulsar el botón pasa al estado `SAMPLING`.

- `SAMPLING`: estado en el que se mide la señal de ECG y se busca la curva QRS para detectar el pulso. Las muestras se capturan de forma periódica con un ADC que almacena el dato en un buffer (fichero data). Cada vez que se detecta la curva se muestra el valor por consola y, tras 8 medidas, se pasa al estado `PRINTING`.

- `PRINTING`: en este estado se hace la media de las medidas realizadas y se imprime por pantalla. Se pasa al estado `END`

- `END`: este estado reproduce un único pitido largo y pasa al estado `NOT_SAMPLING` para esperar la siguiente medida.

La máquina de estados se implementa con un `switch`, donde cada sentencia `case` corresponde a un estado diferente. Se proporciona un `enum` al principio del fichero con los posibles nombres de los estados. Hay que sustituir los valores que vienen en los `case` actuales (1, 2, 3, 4, 5) por los nombres adecuados para cada estado (`INIT`, `NOT_SAMPLING`, `SAMPLING`, `PRINTING`, `END`). Para ello hay que entender el código C proporcionado y saber qué se está haciendo en cada estado, identificando las descripciones planteadas anteriormente con el código en C.

En los estados se activan y desactivan módulos. Rellene la tabla para indicar la situación de cada módulo en cada uno de los estados, pudiendo ser los valores activo o inactivo. Para responder, considere que la etiqueta `ADC_STORED` no está definida, aunque lo esté originalmente.

3. Fichero timer.c

El sistema utiliza el temporizador Timer 3 para poder activar el ADC automáticamente. La frecuencia deseada para muestrear la señal ECG es de 250 Hz.

La función `timer_setup (int interrupt)` está sin implementar. A continuación se plantea la estructura que debe tener. Implemente la función utilizando esta estructura, calculando los valores que hay que poner en cada caso, para generar interrupciones con una frecuencia de 250 Hz. La frecuencia de reloj de los periféricos es 1 MHz. La prioridad de la interrupción está reflejada en la rutina de atención a la interrupción. El temporizador comienza inactivo y se activa manualmente con la macro `TIMER_START`, como se ha visto en el fichero main.c. Recuerde la fórmula para el temporizador de 16-bits:

$$\Delta T = T_{PB} \times PS \times (PR + 1)$$

```
void
timer_setup (int interrupt)
    T3CONbits.ON = XXX;
    T3CONbits.TCS = XXX;
    T3CONbits.TCKPS = XXX;
    PR3 = XXX;

    IPC3bits.T3IP = XXX;
    IPC3bits.T3IS = 0;

    IFS0bits.T3IF = 0;
    IEC0bits.T3IE = (interrupt != 0);
}
```

4. Fichero config.h

Este fichero contiene etiquetas (`define`) de configuración del sistema.

Las etiquetas que empiezan por `PEEP` indican la duración de los pitidos. ¿En qué unidades están expresadas estas duraciones? Consulta los ficheros main.c y timer.c para obtener la respuesta adecuada.

La etiqueta `BUZZER_MASK` contiene una constante con la máscara para manipular el bit del puerto asociado al zumbador (pitido). Si el zumbador se conecta en el pin RA2, ¿qué valor tiene que tener la constante?

5. Simulación sin ADC

En la pestaña *Logic Analyzer* añadimos el pin RA2 con el icono  para poder ver su evolución en el tiempo (ver cuándo va a pitar el sistema).

Una vez se ha llegado a este punto se puede simular el sistema. Para ello hay que provocar un estímulo en la entrada de interrupción asociada al botón, en la pestaña *Stimulus*. Añade un evento que provoque un pulso a nivel alto en la entrada (ver fichero button.c para saber cuál) durante 20 ciclos de reloj cada vez que se active (pulsar *Fire*). Para la simulación es útil poner *breakpoints* en puntos de interés, como los puntos de transición entre estados.

Recomendamos poner los siguientes *breakpoints*:

1. main.c: 69. GIE (inicialización finalizada).
2. main.c: 84. OSCCONbits.SLPEN = 0;(saliendo de estado)
3. main.c: 101. OSCCONbits.SLPEN = 1; (saliendo de estado)
4. main.c: 110. OSCCONbits.SLPEN = 1; (saliendo de estado)
5. main.c: 122. rate_index++; (QRS detectada)
6. main.c: 146. PORTASET = BUZZER_MASK; (saliendo de estado)

Para comprobar si las interrupciones están funcionando correctamente recomendamos poner un *breakpoint* en la primera instrucción de cada una de ellas. En el caso de la interrupción del temporizador o del ADC, como se llaman periódicamente muchas veces, lo más adecuado es eliminar el *breakpoint* una vez visto que funciona.

Al simular se tiene que obtener, en la pestaña de output, UART2 output (consola) mensajes del sistema, entre ellos mensajes con los valores medidos y la media. Algunas trazas se imprimen siempre y otras sólo si está en modo depuración (etiqueta DEBUG definida en config.h, tal y como está originalmente). Copia la salida de consola, desde el mensaje de inicio, en el fichero de soluciones. Además tienes que haber visto cómo varía el pin RA2 (pitido) según se va ejecutando.

6. Fichero data.c

En este fichero se implementa el buffer de acumulación de muestras. ¿Qué tipo de buffer es? ¿Qué función se utiliza para almacenar una nueva muestra? ¿Qué función se utiliza para comprobar si hay muestras disponibles?

7. Fichero adc.c

La etiqueta ADC_STORED se define para que no se utilice el ADC, sino que se utilice el array `adc_buf` con muestras ya almacenadas en el programa. De esta manera se puede depurar el correcto funcionamiento del programa, los algoritmos, la configuración de interrupciones o el buffer antes de comprobar la conexión con el sensor ECG externo. Comente en el fichero config.h la etiqueta ADC_STORED para utilizar el módulo ADC.

El ADC tiene que estar configurado para que capture un dato de la entrada AN0 con una frecuencia de 250 Hz, establecida en el temporizador del fichero timer.c. El ADC provocará una interrupción por cada nuevo dato capturado. El dato será un entero de 32-bits sin signo.

La función adc_setup está sin implementar. A continuación mostramos una implementación parcial, en la que hay que poner los valores correctos en vez de XXX. Sustituya los valores para que cumpla la especificación y añada la implementación al proyecto.

```
void
adc_setup () {
    AD1CON1 = XXX;
    AD1CON2 = 0;
    AD1CON3 = 0;
    AD1CHSbits.CH0SA = XXX;
    AD1CSSL = 0;

    IPC5bits.AD1IP = XXX;
    IPC5bits.AD1IS = 0;
    IFS0bits.AD1IF = 0;
    IEC0bits.AD1IE = 1;
```

```
}
```

8. Simulación con ADC

Detén la simulación antes de continuar, si no la has detenido antes.

Para poder simular el ADC necesitamos indicar cómo varía la señal analógica conectada a la entrada. Para ello hemos generado un fichero SCL (*Stimulus Control Language*)[3] que indica la evolución de la señal en el pin AN0 (fichero `data.an0.scl` en la carpeta del proyecto). Es un fichero similar a un banco de pruebas VHDL (testbench), con un lenguaje derivado de VHDL. El fichero se ha generado a partir de una base de datos de señales ECG muestreadas a 250 Hz. Para añadir el fichero seleccionamos la ventana *Stimulus* en la parte inferior de la pantalla. Tenemos disponibles unos botones a la izquierda, entre los que encontramos el de *Attach SCL file* 

, que nos permite añadir un nuevo fichero de simulación.

En la pestaña *I/O Pins* podemos añadir el pin AN0 para ver su configuración y el valor que tiene en tensión. Inicialmente debería tener un valor de 799 mV, que es el primer valor que se asigna en el fichero data.an0.scl, y cada 4 ms el valor cambia (250 Hz de frecuencia de muestreo).

Si simulamos el sistema, ¿qué valor nos da? ¿Obtenemos un resultado similar al anterior? Copia la salida de consola, desde el mensaje de inicio, en el fichero de soluciones.

[3] http://microchipdeveloper.com/mplabx:scl