

SISTEMAS EMPOTRADOS

3º Grado en Ingeniería Informática

PRÁCTICA 7: Programación con el puerto serie

7.1. Introducción

En esta práctica vamos a aplicar las dos prácticas anteriores. Partimos de la práctica 5, en las que se generaban dos señales por un puerto de salida utilizando un solo *timer* y trabajando por interrupciones, se va a realizar un interfaz de usuario que cambie la frecuencia de las señales.

Se van a generar dos señales cuadradas cuyo semiperiodo será programable por el usuario a través del Terminal y por tanto podremos variar la frecuencia de estas señales digitales. Para descartar cualquier valor el usuario podrá pulsar la tecla ESC. Por otro lado, seguimos utilizando el puerto serie de la práctica anterior (práctica 7).

7.2. Objetivos

El objetivo de esta práctica será realizar un pequeño menú que se comunique con el usuario utilizando uno de los puertos serie de la placa MCB2300 y conectando al otro lado de la línea un ordenador personal que simule un terminal, por ejemplo, con el *Hyperterminal* de YAT disponible en la plataforma *Moodle*.

7.3. Material necesario

- Ordenador personal con *Windows*.
- Placa de desarrollo MCB2300 de *Keil*.
- Adaptador USB–JTAG de la familia ULINK para depurar programas.
- Dos cables USB A–B.
- Cable serie RS232 o adaptador USB–UART, en su caso.

7.4. Desarrollo de la práctica

El interfaz de usuario que se mostrará en la pantalla del terminal tendrá un aspecto similar a este:

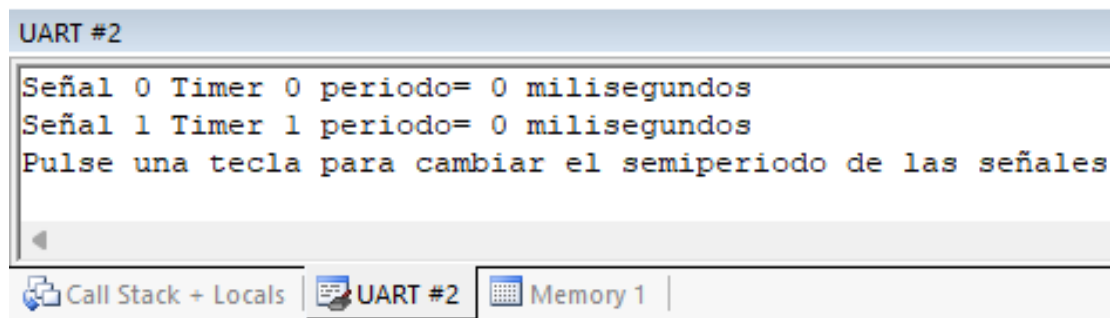


Figura 7-1: Interfaz de usuario inicial.

Una vez pulsada una tecla se pasará al menú de usuario que hará las siguientes preguntas:

Señal-Timer 0 ó 1:

La aplicación espera a que el usuario pulse 0 ó 1 y mostrará el eco en pantalla. Si se pulsa una tecla distinta de 0 ó 1 la aplicación no mostrará eco y descartará la tecla.

A continuación preguntará por el semiperiodo. Se aceptarán valores de dos dígitos y se descartarán las teclas distintas de un valor BCD:

Indique el semiperiodo utilizando dos dígitos 01 a 99:

A continuación, indicará el *timer* que se ha cambiado y la nueva frecuencia mostrando algo similar a esto:

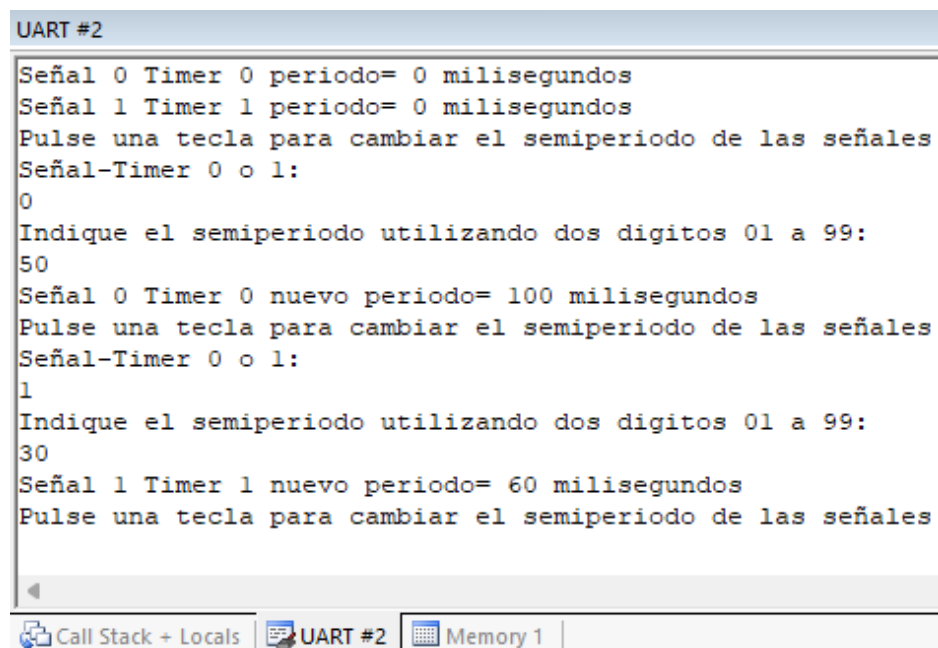


Figura 7-2: Interfaz de usuario a obtener.

- **Indicaciones:**

Separar el menú del resto de la aplicación. Para ello se creará una función con el menú.

El prototipo será:

```
UINT32 menuTimer(UINT32* timer, UINT32* semiperiodo)
```

La función devolverá un valor que indique a la función que llame, si el usuario ha abortado el menú pulsando la tecla ESC. Como argumentos tendrá el puntero a las variables que quedarán modificadas de la función que llama. El primer argumento estará relacionado con el *timer* (señal) y el segundo con el nuevo periodo.

Necesitaremos los ficheros de *retarget.c* y *serial.c* al igual que en la práctica anterior. También necesitaremos los ficheros *IRQ.c* y *HAL.c* de la práctica de los *timers software* con interrupciones. El manejador de proyectos tendrá una apariencia como la siguiente:

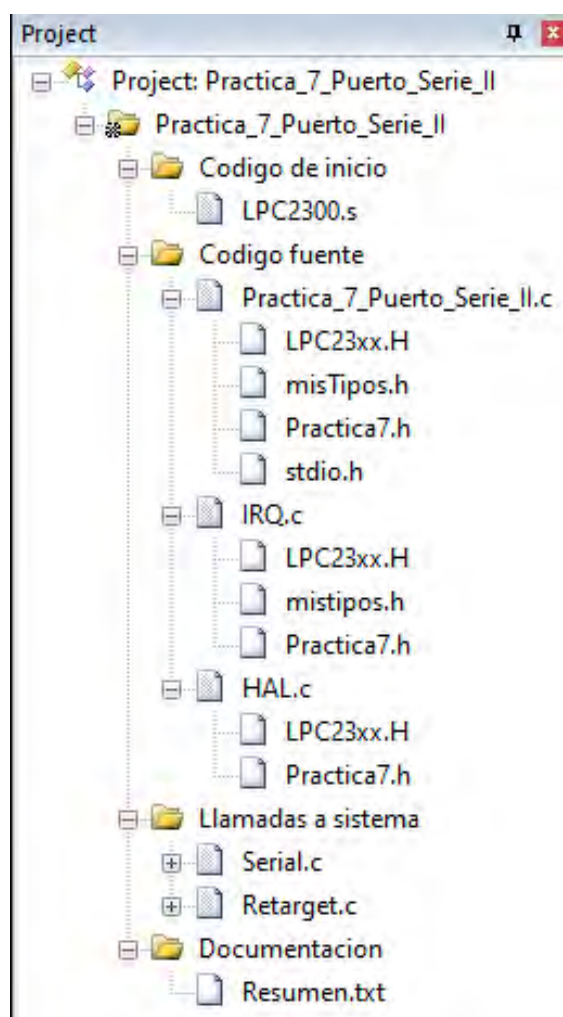


Figura 7-3: Manejador de proyectos.

La parte de código que tiene el fichero serial.c para configurar el puerto serie del microcontrolador LPC2378 de nuestra placa es la siguiente:

```

36 void init_serial (void) {                                /* Initialize Serial Interface */
37     #ifdef UART0                                           /* Enable TxD0 and RxD0 */
38         PINSEL0 |= 0x00000050;
39     #elif defined (UART1)
40         PINSEL0 |= 0x40000000;
41         PINSEL1 |= 0x00000001;
42     #endif
43     UxFDR = 0;                                             /* Fractional divider not used */
44     UxLCR = 0x83;                                          /* 8 bits, no Parity, 1 Stop bit */
45     UxDLL = 78;                                           /* 9600 Baud Rate @ 12.0 MHZ PCLK */
46     UxDLM = 0;                                            /* High divisor latch = 0 */
47     UxLCR = 0x03;                                          /* DLAB = 0 */
48 }

```

Figura 7-4: Serial.c para configurar el puerto serie.

Se observa que en este fichero se ha definido el reloj de 12 MHz del LPC2378 y una velocidad de transmisión de 9600 baudios. Por tanto, nuestra conexión del *hyperterminal* tendremos que definirla para 9600 baudios, 8 *bits*, sin paridad, un *bit* de parada y sin control de flujo.

El generador de velocidad de transmisión es un pre-escalador de dieciséis *bits* que divide el PCLK para generar una señal de reloj en la UART igual a 16 veces la velocidad de transmisión. Por tanto, la fórmula utilizada para calcular la velocidad de transmisión de la UART es:

$$\text{Divisor} = \text{Pclk} / (16 \times \text{BAUD})$$

En el caso de 12 MHz

$$\text{Divisor} = 12.000.000 / (16 \times 9600) = (\text{approx}) 78 \text{ o } 0x4E$$

Con el valor aproximado de 78 se obtiene una velocidad de transmisión real de 9615 baudios. Normalmente no es posible obtener una velocidad de transmisión exacta para la UART, sin embargo, se trabaja con un error máximo del 5% en la temporización.

Tendremos que conectar con un cable serie RS232 nuestra placa por su COM1 a un puerto serie del ordenador, y observar el menú a través del *hyperterminal* de Windows. También podremos utilizar un adaptador USB-UART si nuestro ordenador no dispone de conector de puerto serie.

Los valores obtenidos podremos medirlos en el osciloscopio, al igual que en las prácticas anteriores. Recordar que las señales son cuadradas y lo que hemos programado es el valor del semiperiodo.

7.5. Ficheros auxiliares

- Fichero de capa de abstracción del hardware HAL.c:

```

Practica_7_Puerto_Serie_II.c  Practica7.h  HAL.c  IRQ.c  misTipos.h  Serial.c  Resumen.txt
1  /*****
2  /* HAL.C: funciones generales que acceden al hardware
3  /* Sistemas Empotrados. Universidad de Córdoba
4  /*****
5  #include <LPC23xx.H>          /* LPC23xx definitions
6  #include "Practica7.h"
7  /* Import external IRQ handlers from IRQ.c file
8  extern __irq void T0_IRQHandler (void);
9  extern void init_serial (void);
10 /*****
11 /* pinesSignalInit
12 /*****
13 /* Esta función configura los pines P4.24 y P4.25 como salida
14 /*****
15 void pinesSignalInit(void)
16 {
17     PINSEL9 = 0x00000000;
18     PINMODE9 = 0x00000000;
19     FIO4DIR3 = 0x03;
20 }
21 /*****
22 /* timer0Init
23 /*****
24 /* Esta función configura el timer 0 con los parámetros que no cambian durante*/
25 /* la aplicación
26 /*****
27 void timer0Init(void)
28 {
29     TOPR = 0x00; /* activa el preescalador a cero */
30     //controlador de interrupciones
31     VICVectAddr4 = (unsigned long)T0_IRQHandler; /* Set Interrupt Vector
32     VICVectCntl4 = 15; /* Priority use it for Timer0 Interrupt
33     VICIntEnable = (1 << 4); /* Enable Timer0 Interrupt
34 }
35 /*****
36 /* hardwareInit
37 /*****
38 /* Esta función se llama al comienzo del programa para inicializar el Hardware*/
39 /*****
40 void hardwareInit(void)
41 {
42     pinesSignalInit(); // Configura los pines del circuito
43     timer0Init(); // Inicializa el timer 0
44     init_serial(); // Inicializa el puerto serie
45 }
46

```

Figura 7-5: Fichero de la capa de abstracción del hardware.

- Fichero que contiene la rutina de servicio a la interrupción (ISR) IRQ.c:

```

Practica_7_Puerto_Serie_II.c  Practica7.h  HAL.c  IRQ.c  misTipos.h  Serial.c  Res
1  /*****
2  /* IRQ.C: Rutinas de servicio a la interrupción (ISR)
3  /* Sistemas Empotrados. Universidad de Córdoba
4  /*****
5  #include <LPC23xx.H>
6  #include "Practica7.h"
7  #include "mistipos.h"
8  extern UINT32 timersw0, timersw0InitialCount;
9  extern UINT32 timersw1, timersw1InitialCount;
10 extern BOOL signal0High;
11 extern BOOL signal1High;
12 /* Timer0 IRQ
13 irq void T0_IRQHandler (void) {
14     timersw0--;
15     timersw1--;
16     if (timersw0==0) /* comprueba si el timer sw0 ha finalizado */
17     {
18         timersw0=timersw0InitialCount;
19         if (signal0High==TRUE)
20         {
21             SIGNAL0_PIN_LOW;
22             signal0High=FALSE;
23         }
24         else
25         {
26             SIGNAL0_PIN_HIGH;
27             signal0High=TRUE;
28         }
29     }
30     if (timersw1==0) /* comprueba si el timer sw1 ha finalizado */
31     {
32         timersw1=timersw1InitialCount;
33         if (signal1High==TRUE)
34         {
35             SIGNAL1_PIN_LOW;
36             signal1High=FALSE;
37         }
38         else
39         {
40             SIGNAL1_PIN_HIGH;
41             signal1High=TRUE;
42         }
43     }
44     T0IR = 1;
45     VICVectAddr = 0;
46 }

```

Figura 7-6: Fichero IRQ Rutina de servicio a la interrupción.

- **Fichero header para definiciones Practica7.h:**

```
1 /*****
2 /* Practica7.h: definiciones para la práctica 7
3 /* Sistemas Empotrados. Universidad de Córdoba
4 /*****
5 #define SIGNAL0_PIN_HIGH FIO4SET3 = 0x01; /* Pin señal 0 a alto P4.24 */
6 #define SIGNAL0_PIN_LOW FIO4CLR3 = 0x01; /* Pin señal 0 a bajo P4.24 */
7 #define SIGNAL1_PIN_HIGH FIO4SET3 = 0x02; /* Pin señal 1 a alto P4.25 */
8 #define SIGNAL1_PIN_LOW FIO4CLR3 = 0x02; /* Pin señal 1 a bajo P4.25 */
```

Figura 7-7: Fichero para las definiciones.

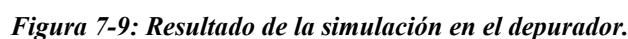
- **Fichero para definición de tipos de variables misTipos.h:**

```
1 /*****
2 /* misTipos.h: definiciones de tipos de variables para practicas con LPC2378
3 /* Sistemas Empotrados. Universidad de Córdoba
4 /*****
5
6 #ifndef __misTipos_H
7 #define __misTipos_H
8
9 /* Byte */
10 #define UINT8 unsigned char
11 #define INT8 char
12
13 /*16 bits */
14 #define UINT16 unsigned short int
15 #define INT16 short int
16
17 /*32 bits WORD para el LPC2378 */
18 #define UINT32 unsigned int
19 #define INT32 int
20
21 /* Tipos para control */
22 #define STATUS UINT32
23
24 /* Booleanas */
25 #define BOOL UINT32
26 #define FALSE (unsigned int)0x00000000
27 #define TRUE (unsigned int)0x00000001
28
29 /* flags */
30 #define FLAG BOOL
31 #define ESC 0x1B
32 #define OK 0x01
33 #endif
34
```

Figura 7-8: Fichero para las definiciones de tipos de variables.

7.6. Resultados de la simulación

Los periféricos que se deben de visualizar, por ser los que se han utilizado en esta práctica, son: GPIO4, Timer 0, UART1 y VIC.



Página 8 de 8