IGNACIO ORTEGA Y ROCIO RUIZ

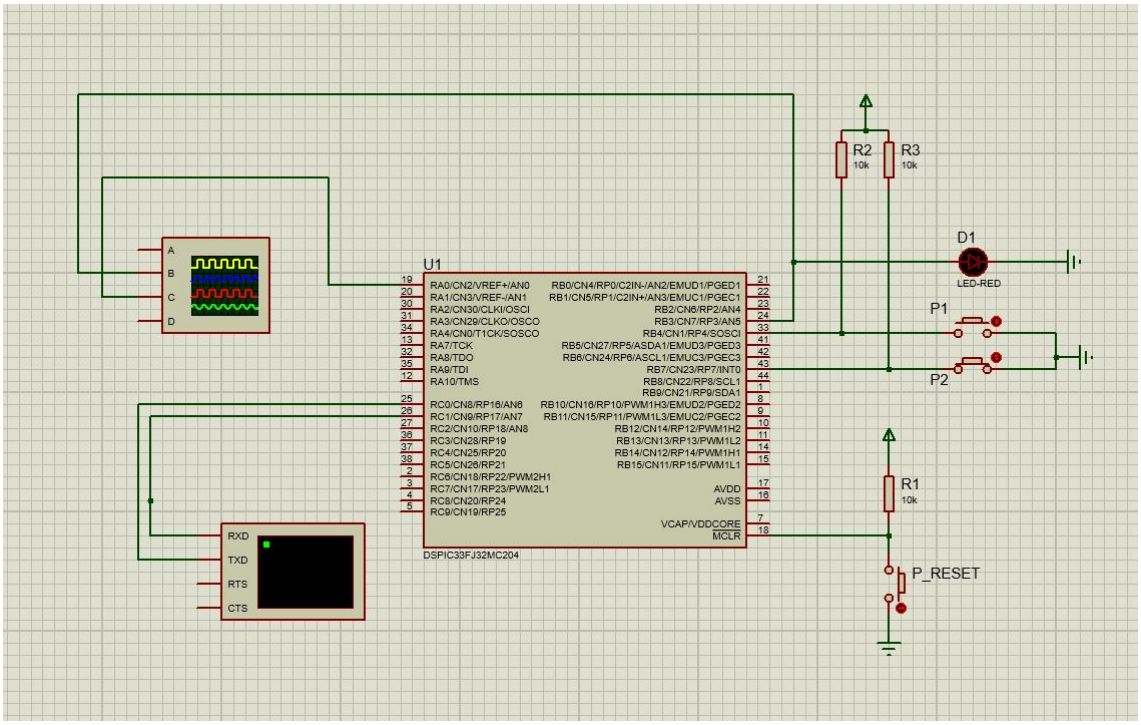
PRÁCTICAS

SISTEMAS EMPOTRADOS

Mar, 27 oct. 2020

PRACTICA 2

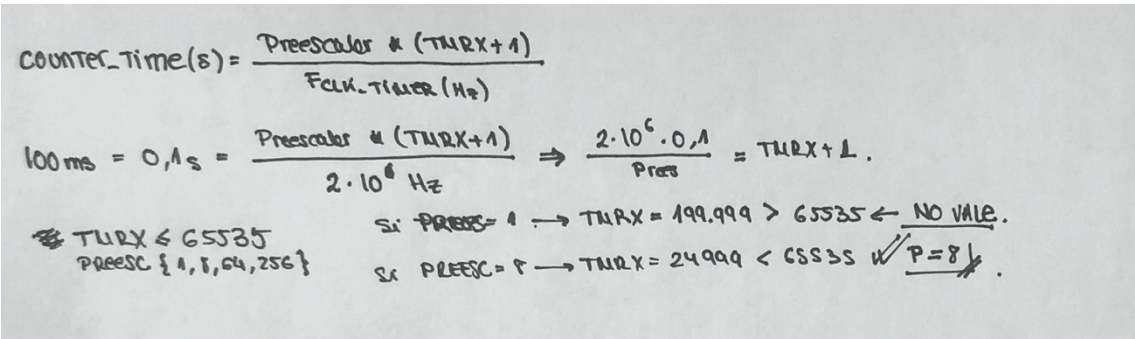
En esta práctica hemos aprendido a configurar y utilizar los temporizadores en el dsPIC33fj32mc204. Para la realización de estos ejercicios primero hemos realizado la configuración de los temporizadores 1 y 2. También hemos adaptado las configuraciones hasta ahora realizadas en clase para una frecuencia de CPU de 2MHz.



Ejercicio 1

Se pide implementar un bucle infinito, cuyo contenido debe procesarse cada 100 usando el Timer1. Y para ello, seleccionar el *Preescaler* que mejor se adapte.

Para el primer temporizador de 16-bits hemos utilizado un *Preescaler* de 8, ya que es el que mejor se adaptaba a los 100 ms. El *Preescaler* de 1 superaba el máximo de *TMRx* posible. Este ha sido el procedimiento para calcularlo:



La configuración de nuestro timer1, por tanto, ha quedado así:

```
83  void  timer1_config(){
84      T1CONbits.TON   = 0; //comprobamos que esta apagado
85      T1CONbits.TSIDL = 0;
86      T1CONbits.TCS   = 0;
87      T1CONbits.TGATE = 0;
88      T1CONbits.TSYNC = 0;
89      T1CONbits.TCKPS = 1; //Preescaler de 8
90      PR1 = 24999; //Configuracion del periodo para 0,1 segundo
91      T1CONbits.TON = 1;
92  }
```

Cada vez que el timer1 alcanza el tiempo máximo, la etiqueta de estado de la interrupción del timer1(T1IF) se pone a 1. Utilizamos esto para asegurarnos de que cada iteración en el bucle se realiza cuando los 100 ms han pasado.

```
165  int main(void)
166  {
167      //Configurar el oscilador para hacer funcionar la CPU a 2 MHz a partir de un reloj de entrada de 8MHz
168      //Fosc = Fin * M/(N1 * N2), Fcy = Fosc/2
169      //Fosc = 8M * 2/(2 * 2) = 4 MHz para un reloj de 8MHz de entrada
170      //Fcy = Fosc/2 = 4/2 = 2MHz (Frecuencia CPU)
171      PLLFBD = 0;                     // M  = 2
172      CLKDIVbits.PLLPOST = 0;         // N1 = 2
173      CLKDIVbits.PLLPRE  = 0;         // N2 = 2
174      while(OSCCONbits.LOCK != 1);    // Esperar a un PLL estable
175
176
177      AD1PCFGL         = 0xFFFF;      // Primer paso. Todos los pines configurados como pines digitales
178      TRISBbits.TRISB3 = 0;   //B3 como salida (LED))
179      LATBbits.LATB3   = 0;
180      TRISBbits.TRISB4 = 1;   //B4 como entrada (P1)
181      TRISBbits.TRISB7 = 1;   //B7 como entrada (P2))
182
183      uart_config(baud_9600);
184      timer1_config();
185      timer2_config();
186
187      while(1)
188      {
189          while(!IFS0bits.T1IF); //Se pone a 1 cuado se alcanza la temporizacion que tenemos
190          IFS0bits.T1IF = 0;
```

Ejercicio 2

Se pide imprimir por pantalla cada 200 milisegundos el estado de dos pulsadores. El estado puede corresponder a “Pulsado” o “No Pulsado”. Por otro lado, cada 100 ms parpadeará un LED rojo que indicará que el ciclo de tiempo generado por el Timer1 funciona correctamente, con ayuda de un osciloscopio.

Lo primero que hemos implementado es una condición que nos asegure que solo se ejecutan las instrucciones de dentro cuando el timer1 ha llegado por segunda vez a 100 ms. Dentro de esta condición miramos si cada pulsador está presionado o no y sacamos su estado actual por terminal virtual haciendo uso de las funciones EnviarString() y EnviarCaracter() estudiadas en clase.

```
193             if(time_count++ == 1)
194             {
195                 memset(txbuffer, '/0', sizeof(txbuffer));
196                 if(P1_pressed){
197                     sprintf(txbuffer, "P1: Pulsado. ");
198                 }else{
199                     sprintf(txbuffer, "P1: No pulsado. ");
200                 }
201                 TMR2 = 0;
202                 EnviarString(txbuffer);
203                 aux = TMR2;
204                 printf("TIMER2: %ull.  \r\n", aux*1000);
205                 if(P2_pressed){
206                     sprintf(txbuffer, "P2: Pulsado. ");
207                 }else{
208                     sprintf(txbuffer, "P2: No pulsado. ");
209                 }
210                 TMR2=0;
211                 EnviarString(txbuffer);
212                 aux = TMR2;
213                 printf("TIMER2: %ull.  \r\n", aux*1000);
214 
215                 time_count = 0;          // Reseteamos variable encargada de medir 2x250ms = 500ms
216             }
217             Led_Red = !PORTBbits.RB3;
218 
219         }
220 
221     }
```

Ejercicio 3

Se pide configurar otro *Timer* para medir con exactitud el tiempo que tarda la trama anterior de datos en enviarse a través de la UART. El tiempo que toma la transmisión debe mostrarse por pantalla, como añadido al mensaje de texto del paso anterior. Este tiempo debe estar expresado en milisegundos.

Para ello hemos configurado el Timer2 de la siguiente manera:

```
94  void  timer2_config(){
95      //T2CON.T32 = 0;
96      T2CONbits.TON   = 0; //comprobamos que esta apagado
97      T2CONbits.TSIDL = 0;
98      T2CONbits.TCS   = 0;
99      T2CONbits.TGATE = 0;
100     T2CONbits.TCKPS = 1;//Preescaler de 8.
101     PR2 = 24999; //Configuracion del periodo para 0,1 segundos.
102     T2CONbits.TON = 1;
103 }
```

Para saber el tiempo que tarda en realizar cada envío a través de la UART, debemos reiniciar el registro TMR2 cada vez que se inicie la cuenta. Por ello, cada vez que llamamos a la función EnviarString, lo reiniciamos.

```
TMR2=0;
EnviarString(txbuffer);
aux = TMR2;
printf("TIMER2: %ull.  \r\n", aux*1000);
```

RESULTADO DE LOS TRES EJERCICIOS

Esto es lo que muestra la Terminal Virtual como resultado.

```
P1: No pulsado. TIMER2: 4311.
P2: No pulsado. TIMER2: 4311.
P1: No pulsado. TIMER2: 4311.
P2: Pulsado. TIMER2: 3111.
P1: No pulsado. TIMER2: 4311.
P2: Pulsado. TIMER2: 3111.
P1: No pulsado. TIMER2: 4311.
P2: No pulsado. TIMER2: 4311.
P1: No pulsado. TIMER2: 4311.
P2: No pulsado. TIMER2: 4311.
P1: Pulsado. TIMER2: 3111.
P2: Pulsado. TIMER2: 3111.
```