TPL N°2 INTERRUPCIONES EXTERNAS - TIMERS

Profesor			
Docentes Auxiliares			
Integrantes			
Curso	R20 _ _	Fecha	/ / 2014
Apellido Nombres		***Legajo***	***Calificación individual***
Calificación Grupal			

Metodología	Práctica guiada por el equipo Docente
Objetivos	• Identificar el significado de WEAK y ALIAS • Comprender el funcionamiento de las Interrupciones Externas. • Analizar la ISR que atiende a la interrupción y ser capaz de modificarla. • Inicializar y realizar la ISR para e Systick. • Ser capaz de realizar otras temporizaciones a partir de la del systick. • Reconocer las diferentes características de la inicialización de un timer. • El Timer como temporizador y contador. • Reconocer el uso de MATCH y CAPTURE

Observaciones

PUNTOS DE INTERÉS EN LA PLACA

Figura 1

***Tecla 1**: Pin9 P2[13] / EINT3 / Reserved / I2STX_SDA*

***Tecla 2** : Pin59 P1[26] / MCOB1 / PWM1.6 / CAP0.0*

CONSIDERACIONES GENERALES

- Contesten todas las preguntas solicitadas en el informe.
- Capturen y peguen las pantallas en donde se les pide que realice cambios en el programa, en donde incorpora breakpoints y en las que ustedes consideren que enriquecerán su informe.
- Suban el informe en la wiki prevista para ello en la página del curso, según la indicación de su profesor.

Ejercicio N°1- Interrupciones Externas

1. Instale el TPL2A.zip.
2. Compile el proyecto y debería encontrar el siguiente mensaje de error.

```
15:39:03 **** Incremental Build of configuration Debug for project TPL2-A ****
make all
Building file: ../TPL2-A-Drivers/cr_startup_lpc176x.c
Invoking: MCU C Compiler
arm-none-eabi-gcc -D__REDLIB__ -DDEBUG -D__CODE_RED -I"F:\Facultad\Informatica 2\CL2014\TPL\TPL2\TPL2-A\TPL2-A-Aplicacion"
-I"F:\Facultad\Informatica 2\CL2014\TPL\TPL2\TPL2-A\TPL2-A-Drivers" -I"F:\Facultad\Informatica 2\CL2014\TPL\TPL2\TPL2-A\TPL2-A-Headers" -O0 -g3
-Wall -c -fmessage-length=0 -fno-builtin -ffunction-sections -fdata-sections -mcpu=cortex-m3 -mthumb -MMD -MP
-MF"TPL2-A-Drivers/cr_startup_lpc176x.d" -MT"TPL2-A-Drivers/cr_startup_lpc176x.d" -o "TPL2-A-Drivers/cr_startup_lpc176x.o"
"../TPL2-A-Drivers/cr_startup_lpc176x.c"
../TPL2-A-Drivers/cr_startup_lpc176x.c:433:6: error: redefinition of 'EINT3_IRQHandler'
../TPL2-A-Drivers/cr_startup_lpc176x.c:110:6: note: previous definition of 'EINT3_IRQHandler' was here
make: *** [TPL2-A-Drivers/cr_startup_lpc176x.o] Error 1

15:39:07 Build Finished (took 3s.828ms)
```

Figura 2

¿Por qué?, Arreglen esta situación y continúen con los siguientes puntos.

3. Coloquen breakPoints en donde indica la Figura 3

```
433 void EINT3_IRQHandler( void )
434 {
435     EXTINT |= ( 1 << EINT3 );
436
437     if( GetPIN( 0 , 22 , 1 ) )
438         SetPIN( 0 , 22 , 0 );
439     else
440         SetPIN( 0 , 22 , 1 );
441 }
```

Figura 3

4. Teniendo la precaución de poner la llaves del dipswitch en ON, ejecuten el programa y pulsen la Tecla 1.
   a. ¿Porque consideran que se deben tomarse la precaución antes mencionada?
   b. ¿Qué sucedió cuando pulsaron la Tecla 2?

c. Expliquen por qué, el cuerpo del main puede ser tan solo un : while(1);

5. Repitan varias veces el punto 4 y determinen en qué momento entra la interrupción ¿Cuando pulsan o cuando sueltan la tecla?
6. Modifiquen el programa para que se comporte exactamente al revés de lo que observaron en el punto 5.
7. Seguramente aprendieron que las funciones de interrupción deben ser cortas y eficientes. ¿Cómo modificarían el main y la función de interrupción, para estar más cerca de ese objetivo?

Ejercicio N°2- Timers - SistickTimer

1. Instalen el TPL2B.zip.
2. Compilen el proyecto. No deberían tener ningún Error ni Warning
3. Copien y peguen la función :

```
void SysTick_Handler(void)
{
    if( GetPIN( 0 , 22 , 1 ) )
        SetPIN( 0 , 22 , 0 );
    else
        SetPIN( 0 , 22 , 1 );
}
```

que se encuentra en el archivo **cr_startup_lpcx.c** en el archivo **TPL2-B-Aplicacion.c**.

4. En esta situación tendrán 2 veces la misma función dentro de su proyecto. Déjelo así y compilenlo. ¿ Que sucedió?, ¿Que mensajes obtuvo?. Justifiquen su respuesta
5. Cada cuánto interrumpe la función de interrupción según quedo configurada en **`InicSysTick()`**
6. Coloquen breakpoints donde consideren necesario para verificar que entra la interrupción.
7. Continuando con lo manifestado en el punto 7 del ejercicio 1. ¿Cómo modificarían el programa para que cuando sea ejecutado, y sin ser bloqueante, pueda observarse el led parpadeando a una frecuencia de 1 Hz?

Ejercicio N°3- Timers /Counters (Como Temporizador)

1. Instalen el TPL2C.zip.
2. Compilen sin errores y sin warnings
3. Describan línea por línea que tarea realiza la siguiente función. Revise el/los headers que acompañan a los fuentes que se proveen con el proyecto para entender el significado de los defines. Deberán ayudarse, asimismo, por la HOJA de DATOS de la Cátedra.

```
void Inicializar_Timer(void)
{
    PCONP |= 1 << 1 ;
    PCLKSEL0 |= 0 << 2 ;

    T0MR0 = 0x7fffff;
    T0MR1 = 0xffffff ;

    T0MCR  = ( ( 1 << MR0I ) | ( 0 << MR0R ) | ( 0 << MR0S ) ) ;
    T0MCR |= ( ( 1 << MR1I ) | ( 1 << MR1R ) | ( 0 << MR1S ) ) ;


    T0TCR &= ( ~( 1 << CE ) ) ;
    T0TCR |= (    1 << CR ) ;

    T0TCR &= ( ~( 1 << CR ) ) ;
    T0TCR |= (    1 << CE ) ;

    ISER0 |= ( 1 << NVIC_TIMER0 ) ;
}
```

4. De acuerdo a lo analizado en el punto 3, ¿cuál será el comportamiento del Timer0?
5. Coloquen breakpoints en el programa de tal manera que justifiquen su explicación.
6. Cambien las líneas de código:

```
T0MR0 = 0x7fffff;
T0MR1 = 0xffffff ;
```

Por:

```
T0MR0 = 0xffffff;
T0MR1 = 0x7fffff ;
```

y describan el funcionamiento. ¿qué pasó?

Ejercicio N°4- Timers / Counters (Como Contador)

1. Instalen el TPL2D.zip.
2. Agreguen lo necesario en las líneas de puntos, para que el proyecto se comporte del mismo modo que el ejercicio 3, pero que tome la entrada de pulsos desde la entrada de CAP0 Deberán ayudarse con la HOJA de DATOS de la Cátedra.

```
void Inicializar_Timer(void)
{
    PCONP |= 1 << 1 ;
    PCLKSEL0 |= 0 << 2 ;

    T0MR0 = 0x7fffff;
    T0MR1 = 0xffffff ;

    ..............;    // Contador de pulsos falling edge en P1.26
    ..............;    // Pone en cero los bits de control del CAP0.0


    T0MCR  = ( ( 1 << MR0I ) | ( 0 << MR0R ) | ( 0 << MR0S ) ) ;
    T0MCR |= ( ( 1 << MR1I ) | ( 1 << MR1R ) | ( 0 << MR1S ) ) ;

    T0TCR &= ( ~( 1 << CE ) ) ;
    T0TCR |= (    1 << CR ) ;

    T0TCR &= ( ~( 1 << CR ) ) ;
    T0TCR |= (    1 << CE ) ;

    ISER0 |= ( 1 << NVIC_TIMER0 ) ;
}
```

3. Pulsen la Tecla 2 para generar los pulsos de entrada en CAP0. ¿Con los valores cargados en T0MR0, pudo verificar algo?. ¿Qué debería hacer?.
4. Coloquen breakspoints donde consideren que sea necesario, para verificar el correcto funcionamiento del proyecto.