Desarrollo de software para usar el temporizador interno y depuración de aplicaciones

Objetivos

- Utilizar el temporizador interno de la CPU en modo interrogación.
- Usar el depurador SDK para fijar un punto de interrupción y ver el contenido de variables y posiciones de memoria

Procedimiento

Crear el proyecto en Vivado

1. Abrir el proyecto anterior y guardarlo como **Lab05PS**. Para ello seleccionar el menu **File -> Project -> Save As...**. Verificar que la opción **Create Project Subdirectory** este seleccionada. Presionar **OK**. Se creará el nuevo proyecto **Lab05PS** en su correspondiente directorio.

   Como se utilizará el temporizador del CPU, que esta siempre presente, no es necesario modificar el hardware del proyecto y el archivo de configuración (bitstream) ya fue generado.

2. En el menu **File** seleccionar **Launch SDK**. En la ventana emergente, presionar **OK**

Crear la Aplicación en SDK

1. En la pestaña **Project Explorer** cerrar los proyectos **lab4**, **lab4_bsp** y **System*wrapper*hw*platfrom*2**

2. En el menu **File** crear un nuevo proyecto de aplicación **File -> New -> Application Project**.

3. En la entrada **Project Name** denominar el proyecto **lab5**. En la entrada **Board Support Package** verificar que este seleccionada la opcion **Create New**. Presionar **Next**.

4. Seleccionar la plantilla **Empty Application** y presionar **Finish**.

5. En la pestaña **Project Explorer** seleccionar el proyecto **lab5 > src** . Presionar el boton derecho del raton y en el menu desplegable seleccionar la opción **Import**.

6. En la ventana emergente expandir la categoria **General** y dar doble click en **File System**. Presionar **Next**.

7. Navegar hasta el archivo **lab5.c** y seleccionarlo. Presionar **Finish**.

   El proyecto se recompilara con errores debido a que el archivo esta incompleto.

Verificar la documentacion de la API del periférico Scutimer

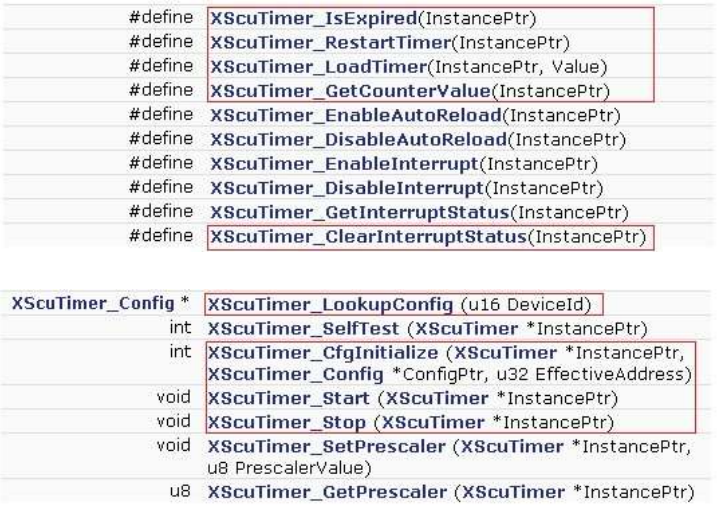
1. En el proyecto **lab5_bsp**, abrir el archivo** system.mss**.

2. En el periférico **ps7_scutimer** presionar el enlace **Documentation** para abrir la documentacion correspondiente en un navegador.

3. Presionar el enlace **File List** para ver los archivos asociados al periférico.

4. Presionar sobre el archivo *xscutimer.h **para ver las funciones asociadas al manejo de este periférico. Analizar las funciones** XScuTimer_LookupConfig( )** y **XScuTimer_CfgInitialize( )** que se utilizan para inicializar el temporizador.

   Otras funciones para utilizar el temporizador:



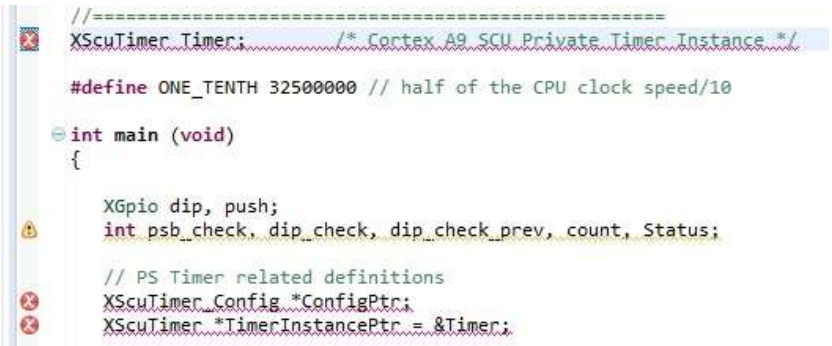
#define	**XScuTimer_IsExpired**(InstancePtr)
#define	**XScuTimer_RestartTimer**(InstancePtr)
#define	**XScuTimer_LoadTimer**(InstancePtr, Value)
#define	**XScuTimer_GetCounterValue**(InstancePtr)
#define	**XScuTimer_EnableAutoReload**(InstancePtr)
#define	**XScuTimer_DisableAutoReload**(InstancePtr)
#define	**XScuTimer_EnableInterrupt**(InstancePtr)
#define	**XScuTimer_DisableInterrupt**(InstancePtr)
#define	**XScuTimer_GetInterruptStatus**(InstancePtr)
#define	**XScuTimer_ClearInterruptStatus**(InstancePtr)

XScuTimer_Config *	**XScuTimer_LookupConfig** (u16 DeviceId)
int	**XScuTimer_SelfTest** (**XScuTimer** *InstancePtr)
int	**XScuTimer_CfgInitialize** (**XScuTimer** *InstancePtr, **XScuTimer_Config** *ConfigPtr, u32 EffectiveAddress)
void	**XScuTimer_Start** (**XScuTimer** *InstancePtr)
void	**XScuTimer_Stop** (**XScuTimer** *InstancePtr)
void	**XScuTimer_SetPrescaler** (**XScuTimer** *InstancePtr, u8 PrescalerValue)
u8	**XScuTimer_GetPrescaler** (**XScuTimer** *InstancePtr)

Funciones para utlizar el Temporizador

Corregir los errores de código

1. En la herramienta SDK, en la pestaña **Problems** (parte inferior de la pantalla), dar doble click en la entrada **Errors** para ver los errores de compilación. Dar doble click en el error **unknown tupe name 'XSCUTimer'** ubicado en la linea 7. Se abrira el código fuente con la linea 7 seleccionada.



```
//====================================================
XScuTimer Timer;        /* Cortex A9 SCU Private Timer Instance */

#define ONE_TENTH 32500000 // half of the CPU clock speed/10

int main (void)
{

   XGpio dip, push;
   int psb_check, dip_check, dip_check_prev, count, Status;

   // PS Timer related definitions
   XScuTimer_Config *ConfigPtr;
   XScuTimer *TimerInstancePtr = &Timer;
```

Primer error en el código

2. Agregar la cabecera **XScuTimer.h** en la linea 5. Guardar el archivo. El proyecto se recompilará sin errores.

```
C     #include “xscutimer.h”
```

1. Mas abajo en el código (linea 31) hay espacios en blanco con comentarios soble el código que es necesario agregar.

```
31    // Initialize the timer
32
33    // Read dip switch values
34    dip_check_prev = XGpio_DiscreteRead(&dip, 1);
35    // Load timer with delay in multiple of ONE_TENTH
36
37    // Set AutoLoad mode
38
39    // Start the timer
40
```

Lugares con código faltante

El codigo a agregar es la configuración del temporizador. Se debe:

- Inicializar el temporizador
- Cargar el valor ONE_TENTH*dip*check*prev como valor de conteo
- Habilitar el modo **Autoload**
- Arrancar el temporizador

2. Completar el codigo faltante usando funciones de la *API functions list*. Al guardar el archivo verificar que el compilado no genere errores.

   Funciones a utilizar:

   ```
   C XScuTimer_LookupConfig( ) XScuTimer_CfgInitialize( )
   XScuTimer_LoadTimer( ) XScuTimer_EnableAutoReload( ) XScuTimer_Start( )
   ```

3. Mas abajo (linea 57) hay otros espacios en blanco para completar con la actualización del valor de conteo en funcion de las posiciones de los interruptores.

```
if (dip_check != dip_check_prev) {
    xil_printf("DIP Switch Status %x, %x\r\n", dip_check_prev, dip_check);
    dip_check_prev = dip_check;
    // load timer with the new switch settings

    count = 0;
}
if(XScuTimer_IsExpired(TimerInstancePtr)) {
        // clear status bit

        // output the count to LED and increment the count
```

Código a completar

EL codigo a completar es:

- Actualizar el valor de conteo a **ONE_TENTH*dip_check**
- Poner a cero *InterruptStatus*
- Actualizar el valor de los leds
- Incrementar la variable de conteo *count*

Para eso se necesitan las funciones:

```
C    XScuTimer_LoadTimer( )    XScuTimer_ClearInterruptStatus ( )
LED_IP_mWriteReg ( )
```

1. Completar el codigo con las funciones, guardar el archivo y verificar que no haya erroes de compilado

2. El código completo es el siguiente:

```C // Include scutimer header file #include "xscutimer.h"
//================================================== XScuTimer Timer; /* Cortex A9 SCU Private Timer Instance */
```

```
#define ONE_TENTH 32500000 // half of the CPU clock speed/10

int main (void)

{

 XGpio dip, push;
 int psb_check, dip_check, dip_check_prev, count, Status;

 // PS Timer related definitions
 XScuTimer_Config *ConfigPtr;
 XScuTimer *TimerInstancePtr = &Timer;

 xil_printf("-- Start of the Program --\r\n");

 XGpio_Initialize(&dip, XPAR_SWITCHES_DEVICE_ID);
 XGpio_SetDataDirection(&dip, 1, 0xffffffff);

 XGpio_Initialize(&push, XPAR_BUTTONS_DEVICE_ID);
 XGpio_SetDataDirection(&push, 1, 0xffffffff);

 count = 0;

 // Initialize the timer
 ConfigPtr = XScuTimer_LookupConfig (XPAR_PS7_SCUTIMER_0_DEVICE_ID);
 Status = XScuTimer_CfgInitialize   (TimerInstancePtr, ConfigPtr,  ConfigPtr-
>BaseAddr);
 if(Status != XST_SUCCESS){
    xil_printf("Timer init() failed\r\n");
    return XST_FAILURE;
 }
 // Read dip switch values
 dip_check_prev = XGpio_DiscreteRead(&dip, 1);
 // Load timer with delay in multiple of ONE_TENTH
 XScuTimer_LoadTimer(TimerInstancePtr, ONE_TENTH*dip_check_prev);
 // Set AutoLoad mode
 XScuTimer_EnableAutoReload(TimerInstancePtr);
 // Start the timer
 XScuTimer_Start (TimerInstancePtr);
 while (1)
 {
    // Read push buttons and break the loop if Center button pressed
    psb_check = XGpio_DiscreteRead(&push, 1);
  if(psb_check > 0)
    {
      xil_printf("Push button pressed: Exiting\r\n");
      XScuTimer_Stop(TimerInstancePtr);
      break;
    }
    dip_check = XGpio_DiscreteRead(&dip, 1);
  if (dip_check != dip_check_prev) {
      xil_printf("DIP Switch Status %x, %x\r\n", dip_check_prev,
dip_check);
      dip_check_prev = dip_check;
          // load timer with the new switch settings
      XScuTimer_LoadTimer(TimerInstancePtr, ONE_TENTH*dip_check);
      count = 0;
  }
  if(XScuTimer_IsExpired(TimerInstancePtr)) {
```

```
            // clear status bit
            XScuTimer_ClearInterruptStatus(TimerInstancePtr);
            // output the count to LED and increment the count
            LED_IP_mWriteReg(XPAR_LED_IP_S_AXI_BASEADDR, 0, count);
            count++;
     }
    }
    return 0;

} ```
```

Verificar la funcionalidad en el hardware

1. Conectar la placa Arty Z7-20

2. Seleccionar la pestaña **SDK Terminal** en la parte inferior de la pantalla

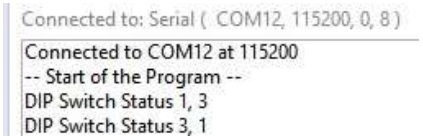
3. Presionar el boton + y verificar los parametros de puerto COM y velocidad de transmision si es necesario (normalmente estan los parametros del Lab1). Presionar **OK**. En la pantalla del terminal aparecera el texto **Connected to COMxx at 115200**

4. Seleccionar el menu **Xilinx -> Program FPGA**.

5. Presionar el boton **Program** para configurar el bloque PL (que contiene la IP Propia).

6. En el panel **Project Explorer** seleccionar el proyecto **lab5**. Presionar el boton derecho del raton y en el menu desplegable seleccionar **Run As -> Launch on Hardware (System Debugger)** para programar el bloque PS y ejecutar la aplicación

   Cambiar las posiciones de los interruptores y verificar que los leds cambian de color implementando un contador binario con un retardo dado por la posicion de los interruptores, siendo el conteo mas lento cuando los dos interruptores estan encendidos. Por otra parte en la pestaña **SDK Terminal** se puede ver el estado anterior y el actual de los interruptores



Ventana SDK Terminal

Ejecutar el depurador

1. Cambiar a la perspectiva **Debug** (ubicado arriba a la derecha).

2. En la pestaña **Debug** (arriba a la izquierda) seleccionar el procesador que esta ejecutando el código **ARM Cortex-A9 MPCore #0 (Runnig)**. Presionar el boton derecho del raton y en el menu desplegable seleccionar **Suspend**

   En este punto se puede agregar variables globales presionando el boton derecho del raton en la pestaña **Variables** y en el menu desplegable seleccionando **Add Global Variables...**. Como la aplicación no tiene variables globales, no se agregara nada.

3. En la pestaña **lab5.c** (en el centro a la izquierda), dar doble click en el numero de linea para colocar un punto de interrupccion (breakpoint). Colocar los puntos de interrupcion segun se ve en la siguiente figura.

```
26    XGpio_Initialize(&push, XPAR_BUTTONS_DEVICE_ID);
27    XGpio_SetDataDirection(&push, 1, 0xffffffff);
28
29    count = 0;
30
31    // Initialize the timer
32    ConfigPtr = XScuTimer_LookupConfig(XPAR_PS7_SCUTIMER_0_DEVICE_ID);
33    XScuTimer_CfgInitialize  (TimerInstancePtr, ConfigPtr, ConfigPtr->BaseAddr);
34    // Read dip switch values
35    dip_check_prev = XGpio_DiscreteRead(&dip, 1);
36    // Load timer with delay in multiple of ONE_TENTH
37    XScuTimer_LoadTimer  (TimerInstancePtr, ONE_TENTH*dip_check_prev);
38    // Set AutoLoad mode
39    XScuTimer_EnableAutoReload   (TimerInstancePtr);
40    // Start the timer
41    XScuTimer_Start(TimerInstancePtr);
```

```
44        // Read push buttons and break the loop if Center button pressed
45        psb_check = XGpio_DiscreteRead(&push, 1);
46        if(psb_check > 0)
47        {
48            xil_printf("Push button pressed: Exiting\r\n");
49            XScuTimer_Stop(TimerInstancePtr);
50            break;
51        }
52        dip_check = XGpio_DiscreteRead(&dip, 1);
53        if (dip_check != dip_check_prev) {
54            xil_printf("DIP Switch Status %x, %x\r\n", dip_check_prev, dip_check);
55            dip_check_prev = dip_check;
56            // load timer with the new switch settings
57            XScuTimer_LoadTimer(TimerInstancePtr, ONE_TENTH*dip_check);
58            count = 0;
59        }
60        if(XScuTimer_IsExpired(TimerInstancePtr)) {
61                // clear status bit
62            XScuTimer_ClearInterruptStatus(TimerInstancePtr);
63                // output the count to LED and increment the count
64            LED_IP_mWriteReg(XPAR_LED_IP_S_AXI_BASEADDR, 0, count);
65            count++;
66        }
67    }
```

Puntos de Interrupción

Un punto de interrupción se representa con un circulo azul en el margen izquierdo del código fuente. El primer punto de interrupción es cuando el contador se inicia a cero. El segundo punto de interrupcion esta en la función que lee la posición de los interruptores. El tercer punto de interrupción esta antes de finalizar el programa porque se presiono algun pulsador. El cuarto punto de interrupción esta cuando finalizo el contador y se va a actualizar el estado de los leds de la IP Propia.

4. Presionar la flecha hacia abajo en el boton de menu **Debug** (ubicado en la parte superior). En el menu desplegable seleccionar la opción **System Debugger using Debug_lab5.elf on local**. En la ventana emergente presionar** OK** para finalizar la ejecución actual e iniciar una nueva ejecución del depurador.

   La aplicación se reiniciará y comenzara a ejecutarse hasta detenerse en la primer sentencia de la función **main()**. Esto se indica con un tilde en la linea correspondiente (linea 12). Cuando se realiza la depuración de una aplicación, siempre hay un punto de interrupción implicito en la primer linea de la función main(). En la pestaña **Variables** se vera un valor de *count* aleatorio distinto de cero porque la variable todavia no fue inicializada.

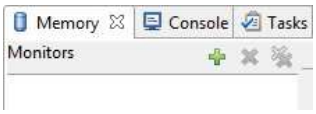
5. Presionar el boton **Resume** ( ). Se ejecutaran las lineas de código hasta el primer punto de interrupción (inicialización de *count*). El valor de *count* todavia es aleatorio y distinto de cero porque la variable todavia no fue inicializada.

6. Presionar el boton **Step Over** ( ) o la tecla **F6** para ejecutar la siguiente sentencia (inicialización de *count*). Se verifica que la variable *count* ahora vale 0_ y en la pestaña **Variables** la entrada correspondiente a *count* esta resaltada en amarillo para indicar que cambio el valor de la variable.

7. Presionar nuevamente el boton **Resume** ( ). Se ejecutaran las lineas de código hasta el segundo punto de interrupción (función que lee los interruptores). Se verifica que se inicializo el temporizador (puntero *ConfigPtr* resaltado en amarillo)

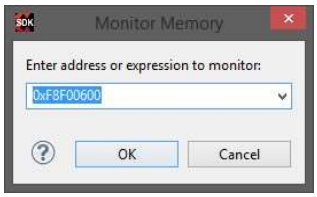
8. Presionar nuevamente el boton **Step Over** ( ) o la tecla **F6** para ejecutar la siguiente sentencia (función que lee los interruptores). Se verificará que la variable *dipcheckprev* cambión su valor y refleja la combinación de posiciones de los pulsadores.

9. Presionar la pestaña **Memory** abajo a la derecha para seleccionarla. Presionar el simbolo **"+"** en verde para agregar un **Memory Monitor**



Pestaña Memory

10. En la ventana emergente ingresar la posicion de memoria del registro de carga del temporizador (*0xF8F00600*), y presionar **OK**.



Ver el contenido de una dirección de memoria

La dirección del registro de carga se calcula como la dirección base del temporizador, que esta en el archivo *xparameters.h* `# XPAR_PS7XPAR_PS7_SCUTIMER_0_BASEADDR` , al que se le suma un desplazamiento que esta definido en el archivo _*xscutimer_hw.h* (`#define XSCUTIMER_LOAD_OFFSET`) . En este caso el valor de desplazamiento es 0, por lo que la dirección del registro de carga es la dirección base del periférico.

11. Verificar que los interruptores no esten en la posicion "00" y presionar el boton **Step Over** ( ) o la tecla **F6** para ejecutar la siguiente sentencia, que cargara el el valor al registro de carga.

    Verificar en la pestaña **Memory** que el contenido de la dirección 0xF8F00604 esta en rojo, indicando que el contenido ha cambiado. Verificar que el contenido de la dirección 0xF8F00600 es equivalente al valor dip*check*prev*32500000 en hexadecimal (bytes en orden 0 -> 3 o BigEndian, el byte mas significativo en la posicion de memoria mas baja). Por ejemplo, para dip*check*prev = 3; el valor es 0x05CFBB60.

12. Presionar nuevamente el boton **Resume** ( ). Se ejecutaran las lineas de código hasta el cuarto punto de interrupción (función que escribe en un registro de la IP Propia, actualizando el encendido de los leds). No se detuvo la ejecución en el tercer punto de interrupción porque no se presiono ningun pulsador. Se verifica que se actualizo el valor de lectura de los interruptores (variable **dip_check** resaltado en amarillo).

13. Presionar el boton **Step Over** ( ) o la tecla **F6** para ejecutar la siguiente sentencia (escritura del registro de la IP Propia). Se verifica que los leds se apagan porque *count* vale 0.

14. Dar doble click en el cuarto punto de interrupción (función que escribe en un registro de la IP Propia, actualizando el encendido de los leds). Se vera que el punto azul desaparece, lo que indica que se quitó el punto de interrupción y el programa se ejecutará en forma continua.

15. Presionar nuevamente el boton **Resume** ( ). El programa se ejecuta actualizando el valor de los leds en forma continua..

16. Cambiar la posicion de los interruptores y observar como esto afecta a la velocidad a la que se encienden los leds.

17. Presionar un pulsador y observar que el programa se detiene en el tercer punto de interrupción. Observar que el regisgtro de carga tiene otro valor (ya que la configuración de los interruptores es otra) y el **registro de contro** (*control register*) en el desplazamiento 0x08 esta en rojo. Esto se debe a que se escribión en el registro de control para detener el temporizador.

18. Terminar la sesión y cerrar las herramientas SDK y Vivado.

Conclusion

En esta práctica se analizo el periférico temporizador interno, se vio su documentación, API, y registros. Se inicializo y controlo el temporizador mediante las funciones apropiadas de su API. Se verifico la funcionalidad en el hardware y se utilizaron las funcionalidades de depuración de la herramienta SDK para ver el contenido de variables y posiciones de memoria y pausar la ejecución en distintas partes del código.