

Microarquitecturas y Softcores

Práctica 5

Escribiendo Software para un Timer y Depurando

Introducción

Esta práctica lo guiará a través del proceso de escritura de una aplicación de software que utiliza el timer de la CPU. Usted se referirá a la API timer en el SDK para crear y depurar la aplicación. La aplicación que desarrollará monitoreará los valores de los switches dip e incrementará una cuenta en los LEDs. La aplicación terminará cuando el botón push central sea presionado.

Objetivos

Después de completar esta práctica será capaz de:

- Utilizar el timer del CPU en modo polling
- Usar el depurador del SDK para establecer breakpoints y ver el contenido de las variables y la memoria

Procedimiento

Esta práctica está separada en pasos que consisten en sentencias generales que proveen información sobre las instrucciones detalladas que le siguen. Siga estas instrucciones detalladas para avanzar dentro de esta práctica.

Esta práctica está compuesta por 4 pasos principales: Abrir el proyecto en Vivado, crear un proyecto de software en SDK, verificar la operación en hardware, y lanzar el depurador para depurar el diseño.

Descripción del Diseño

Se usará el diseño de hardware creado en la práctica 4 para usar el timer de la CPU. Desarrollará el código para usarlo.

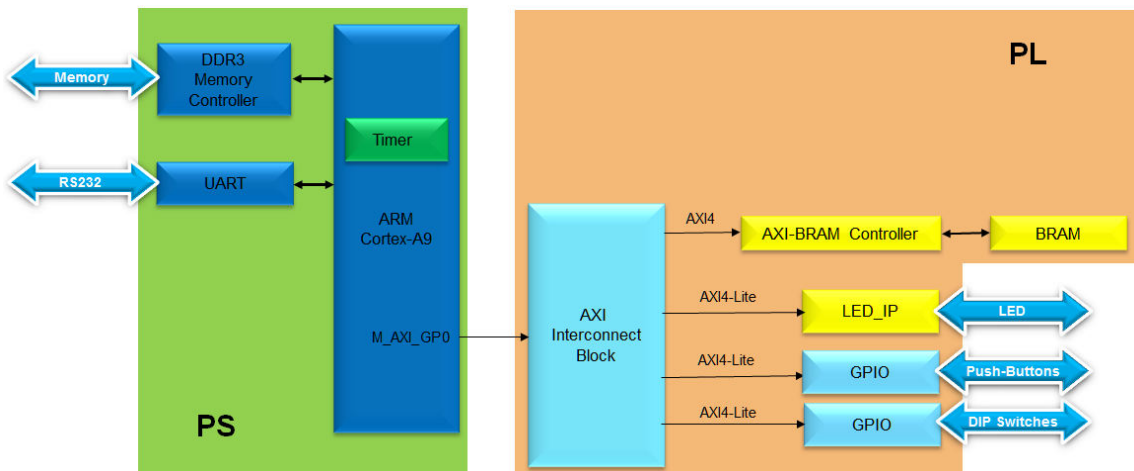
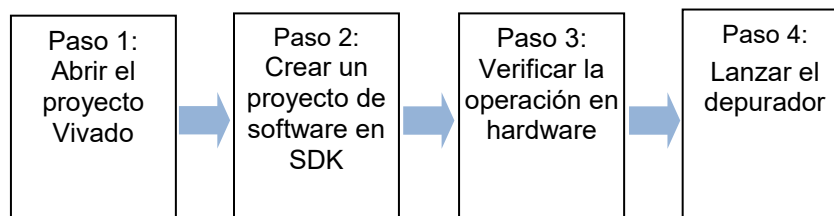


Figura 1. Diseño actualizado de la práctica anterior

Flujo General para esta práctica



Abrir el Proyecto

Paso 1

1-1. Abrir el proyecto anterior, y guardarlo como lab5. Abrir el Block Design.

- 1-1-1. Iniciar Vivado, y abrir el proyecto lab4.xpr que creó en la práctica anterior usando el link **Open Project** en la página Getting Started.

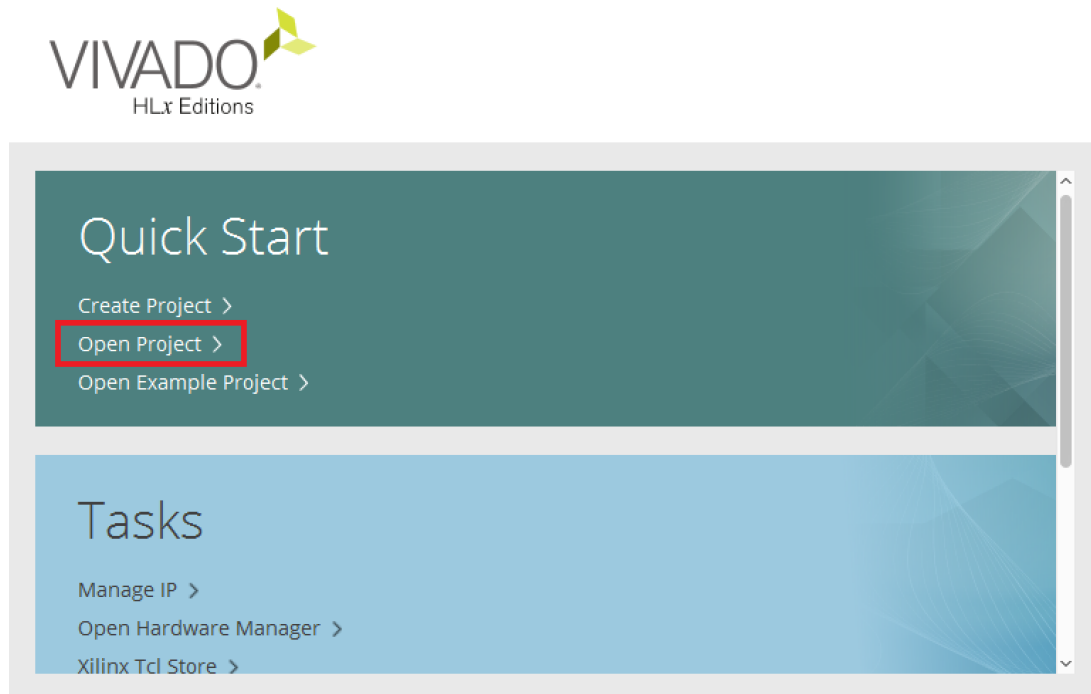


Figura 2. Abrir proyecto existente

- 1-1-2. Seleccionar **File ► Project ► Save As ...** para abrir el cuadro de diálogo *Save Project As*. Introducir **lab5** como nombre de proyecto. Asegúrese de que la opción *Create Project Subdirectory* esté marcada. Presionar **OK**.

Esto creará el directorio lab5 y guardará el proyecto y directorio asociado con el nombre lab5.

Ya que estaremos usando el timer del CPU, que está siempre presente, no necesitamos modificar el diseño de hardware.

- 1-1-3. Abrir el *Block Design (IP INTEGRATOR ► Open Block Design)*. Notará que el estado cambia a *Synthesis and Implementation Out-of-date* ya que el proyecto fue *salvado como ...* Ya que el bitstream ya ha sido generado y estará en el directorio exportado, podemos ignorar de manera segura cualquier advertencia acerca de esto.

- 1-1-4. En Vivado, seleccionar **File ► Launch SDK**. Presionar **OK**.

Aparecerá una ventana advirtiéndole que el diseño está desactualizado. Ya que no hemos realizado ningún cambio podemos ignorar la advertencia. Presionamos **Yes**.

Crear un Proyecto de Software en SDK

Paso 2

2-1. Cerrar proyectos creados previamente. Crear un proyecto de aplicación nuevo vacío llamado lab5 utilizando la ya existente plataforma de software standalone_bsp_0. Importar el archivo fuente lab5.c.

2-1-1. En el *Project Explorer* en SDK, hacer botón derecho sobre *lab4*, *lab4_bsp* y *system_wrapper_hw_platform_2* y seleccionar **Close Project**

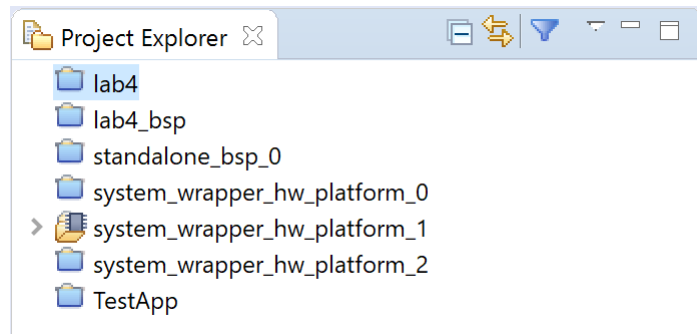


Figura 3. Cerrando proyectos que no se utilizarán

2-1-2. Seleccionar **File ► New ► Application Project**.

2-1-3. Nombrar el proyecto como **lab5**, y para el Board Support Package, (Dejar *Create New* para el *Board Support Package*) y hacer click en **Next**.

2-1-4. Seleccionar **Empty Application** y hacer click en **Finish**.

2-1-5. Seleccionar **lab5 ► src** en el project explorer, hacer botón derecho, y seleccionar **Import**.

2-1-6. Expandir la categoría **General** y hacer doble click sobre **File System**.

2-1-7. Navegar (Browse ...) hasta la ubicación del archivo provisto **lab5.c** y hacer click en **OK**, y luego en **Finish**.

Notará que hay multiples errors de compilación. Esto es esperable debido a que el código está incompleto. Más adelante se lo completará.

2-2. Referirse a la documentación de la API Scutimer.

2-2-1. Abrir el **system.mss** de lab5_bsp

2-2-2. Hacer click sobre el link **Documentation** correspondiente al periférico **scutimer** (*ps7_scutimer_0*) debajo de la sección *Peripheral Drivers* para abrir la documentación en un navegador web.

Peripheral Drivers

Drivers present in the Board Support Package.

axi_bram_ctrl_0	bram	Documentation	Import Examples
buttons	gpio	Documentation	Import Examples
led_ip	led_ip		
ps7_afi_0	generic		
ps7_afi_1	generic		
ps7_afi_2	generic		
ps7_afi_3	generic		
ps7_coresight_comp_0	coresightps_dcc	Documentation	
ps7_dds_0	ddrps	Documentation	
ps7_ddrc_0	generic		
ps7_dev_cfg_0	devcfg	Documentation	Import Examples
ps7_dma_ns	dmaps	Documentation	Import Examples
ps7_dma_s	dmaps	Documentation	Import Examples
ps7_globaltimer_0	generic		
ps7_gpv_0	generic		
ps7_intc_dist_0	generic		
ps7_iop_bus_config_0	generic		
ps7_l2cachec_0	generic		
ps7_ocmc_0	generic		
ps7_pl310_0	generic		
ps7_pmu_0	generic		
ps7_ram_0	generic		
ps7_ram_1	generic		
ps7_scuc_0	generic		
ps7_scugic_0	scugic	Documentation	Import Examples
ps7_scutimer_0	scutimer	Documentation	Import Examples
ps7_scuwdt_0	scuwdt	Documentation	Import Examples
ps7_slcr_0	generic		

Figura 4. Documentación de los drivers de periféricos

2-2-3. Hacer click sobre la pestaña **File List** para ver los archivos disponibles relacionados con la API timer.

2-2-4. Navegar al directorio source, {Instalación Xilinx}
 \SDK\2018.1\data\embeddedsd\XilinxProcessorIPLib\drivers\scutimer_v2_1\src y abrir **xscutimer.h** para ver las diferentes funciones disponibles.

Observar las funciones *XScuTimer_LookupConfig()* y *XScuTimer_CfgInitialize()* que deben ser llamadas antes de que la funcionalidad *timer* pueda ser accesible.

Observar las diferentes funciones disponibles para interactuar con el timer, incluyendo:

#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)

Figura 5. Funciones útiles

2-3. Corregir los errores

- 2-3-1. En SDK, en la pestaña **Problems**, hacer doble click sobre el error *unknown type name* Esto abrirá el archivo fuente y lo posicionará en el lugar del error.

```

#include "xparameters.h"
#include "xgpio.h"
#include "led_ip.h"
// Include scutimer header file

//=====
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);

```

Figura 6. Primer error

- 2-3-2.** Agregar la sentencia include en la parte superior del archivo lab5.c para el *XScuTimer.h*. Guardar el archivo y el error debería desaparecer.

```
#include "xscutimer.h"
```

- 2-3-3.** Deslizar para abajo el archivo y notar que hay unas pocas líneas dejadas intencionalmente en blanco con comentarios.

```

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

```

Figura 7. Lugares a completar en el código

El timer necesita ser inicializado, necesita ser cargado con el valor `ONE_TENTH*dip_check_prev`, necesita ser habilitado el modo AutoLoad, y finalmente necesita ser iniciado.

- 2-3-4.** Usando la lista de funciones, llenar las líneas dejadas en blanco. Guardar el archivo y corregir los errores si los hubiera. (Si fuera necesario utilizar el código completo provisto en el sitio).

Funciones necesarias: XScuTimer_LookupConfig()
XScuTimer_CfgInitialize()
XScuTimer_LoadTimer()
XScuTimer_EnableAutoReload()
XScuTimer_Start()

- 2-3-5.** Deslizar para abajo el archivo un poco más y notar que existen unas cuantas líneas más dejadas en blanco con comentarios asociados.

```
62     if (dip_check != dip_check_prev) {
63         xil_printf("DIP Switch Status %x, %x\r\n", dip_check_prev, dip_check);
64         dip_check_prev = dip_check;
65         // load timer with the new switch settings
66
67         count = 0;
68     }
69     if(XScuTimer_IsExpired(TimerInstancePtr)) {
70         // clear status bit
71
72         // output the count to LED and increment the count
```

Figura 8. Más código para ser completado

El valor de ONE_TENTH*dip_check necesita ser escrito al timer para actualizarlo, el InterruptStatus necesita ser limpiado, y el LED necesita ser escrito, y la variable de cuenta incrementada.

Funciones necesarias: XScuTimer_LoadTimer()
XScuTimer_ClearInterruptStatus()
LED_IP_mWriteReg()

- 2-3-6.** Usando la lista de funciones, completar las líneas faltantes. Guardar el archivo y corregir los errores si los hubiere.


```
1 #include "xparameters.h"
2 #include "xgpio.h"
3 #include "led_ip.h"
4 // Include scutimer header file
5 #include "xscutimer.h"
6
7 //=====
8 XScuTimer Timer;          /* Cortex A9 SCU Private Timer Instance */
9
10 #define ONE_TENTH 32500000 // half of the CPU clock speed/10
11
12 int main (void)
13 {
14
15     XGpio dip, push;
16     int psb_check, dip_check, dip_check_prev, count, Status;
17
18     // PS Timer related definitions
19     XScuTimer_Config *ConfigPtr;
20     XScuTimer *TimerInstancePtr = &Timer;
21
22     xil_printf("-- Start of the Program --\r\n");
23
24     XGpio_Initialize(&dip, XPAR_SWITCHES_DEVICE_ID);
25     XGpio_SetDataDirection(&dip, 1, 0xffffffff);
26
27     XGpio_Initialize(&push, XPAR_BUTTONS_DEVICE_ID);
28     XGpio_SetDataDirection(&push, 1, 0xffffffff);
29
30     count = 0;
31
32     // Initialize the timer
33     ConfigPtr = XScuTimer_LookupConfig (XPAR_PS7_SCUTIMER_0_DEVICE_ID);
34     Status = XScuTimer_CfgInitialize (TimerInstancePtr, ConfigPtr, ConfigPtr->BaseAddr);
35     if (Status != XST_SUCCESS){
36         xil_printf("Timer init() failed\r\n");
37         return XST_FAILURE;
38     }
39
40     // Read dip switch values
41     dip_check_prev = XGpio_DiscreteRead(&dip, 1);
42     // Load timer with delay in multiple of ONE_TENTH
43     XScuTimer_LoadTimer(TimerInstancePtr, ONE_TENTH*dip_check_prev);
44
45     // Set AutoLoad mode
46     XScuTimer_EnableAutoReload(TimerInstancePtr);
47
48     // Start the timer
49     XScuTimer_Start (TimerInstancePtr);
50
```

```

51  while (1)
52  {
53      // Read push buttons and break the loop if Center button pressed
54      psb_check = XGpio_DiscreteRead(&push, 1);
55      if(psb_check > 0)
56      {
57          xil_printf("Push button pressed: Exiting\r\n");
58          XScuTimer_Stop(TimerInstancePtr);
59          break;
60      }
61      dip_check = XGpio_DiscreteRead(&dip, 1);
62      if (dip_check != dip_check_prev) {
63          xil_printf("DIP Switch Status %x, %x\r\n", dip_check_prev, dip_check);
64          dip_check_prev = dip_check;
65          // load timer with the new switch settings
66          XScuTimer_LoadTimer(TimerInstancePtr, ONE_TENTH*dip_check);
67          count = 0;
68      }
69      if(XScuTimer_IsExpired(TimerInstancePtr)) {
70          // clear status bit
71          XScuTimer_ClearInterruptStatus(TimerInstancePtr);
72          // output the count to LED and increment the count
73          LED_IP_mWriteReg(XPAR_LED_IP_S_AXI_BASEADDR, 0, count);
74          count++;
75      }
76  }
77  return 0;
78  }
79

```

Figura 9. Código completo

Verificar la Operación en Hardware

Paso 3

3-1. Conectar la placa a la PC con el cable micro-usb. Establecer una comunicación serial con alguna terminal.

3-1-1. Asegurarse de que existe un cable micro-USB conectado entre la placa y la PC.

3-1-2. Iniciar una terminal. Si se lo desea se puede utilizar la existente en el entorno. Seleccionar el puerto COM apropiado y configurar una velocidad de 115200.

3-2. Programar la FPGA seleccionando Xilinx ► Program FPGA y asignando el archivo system_wrapper.bit. Ejecutar la aplicación TestApp y verificar la funcionalidad.

3-2-1. Seleccionar Xilinx ► Program FPGA.

3-2-2. Hacer click en el botón **Program** para programar la FPGA.

- 3-2-3.** Seleccionar **lab5** en *Project Explorer*, hacer botón derecho y seleccionar **Run As ► Launch on Hardware (GDB)** para descargar la aplicación, ejecutar ps7_init y lab5.elf

Dependiendo de la configuración de los switches verá los LEDs implementando un contador binario con el correspondiente retardo.

Modificar los DIP switches y verificar que los LEDs responden acordemente al retardo configurado por aquellos. También notar que en la terminal se muestra la configuración previa y actual de los switches cada vez que éstos se modifican.

```
-- Start of the Program --  
DIP Switch Status 1, 3  
DIP Switch Status 3, 7
```

Figura 10. Salida de la terminal

Lanzar el Depurador

Paso 4

4-1. Lanzar el Depurador y depurar

- 4-1-1.** Hacer click derecho sobre el proyecto **Lab5** en la vista *Project Explorer* y seleccionar **Debug As ► Launch on Hardware (GDB)**.

El archivo lab5.elf será descargado y si se lo solicita, hacer click en **Yes** para detener la ejecución del programa.

- 4-1-2.** Hacer click en **Yes** si se pregunta cambiar a la *Debug perspective*.

En este momento usted podría agregar variables globales haciendo click derecho en la pestaña **Variables** y seleccionando **Add Global Variables ...**. Todas las variables globales tendrían que mostrarse y usted podría seleccionar las que les interesara. Ya que no tenemos variables de este tipo, no lo haremos.

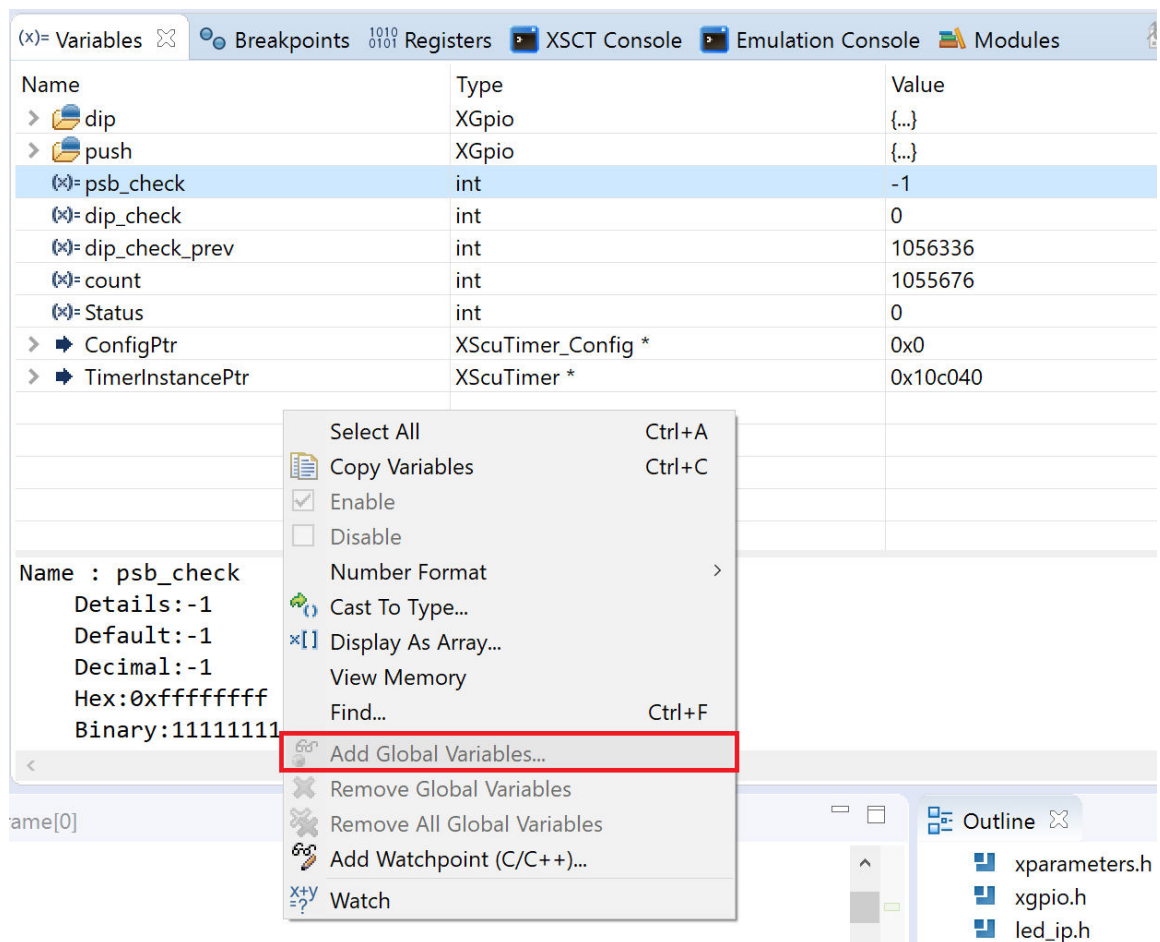


Figura 11. Agregado de variables globales a ser visualizadas

- 4-1-3.** Hacer doble click en el margen izquierdo del archivo lab5.c para establecer un breakpoint en las líneas mostradas en la imagen siguiente. Un breakpoint ha sido establecido cuando un "tick" y un círculo azul aparecen en el margen izquierdo al lado de la línea correspondiente. (Los números de las líneas pueden diferir mínimamente en su archivo).


El primer breakpoint es cuando la variable count es inicializada en 0. El segundo es para capturar un fallo en la inicialización del timer. El tercero es cuando el programa está por leer la configuración de los dip switch. El cuarto breakpoint es cuando el programa está por terminar debido a la presión del botón del centro. El quinto breakpoint es cuando el timer ha expirado y va a escribir en LED.

```


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  Status = XScuTimer_CfgInitialize (TimerInstancePtr, ConfigPtr, ConfigPtr->BaseAddr);
34  if(Status != XST_SUCCESS){
35      xil_printf("Timer init() failed\r\n");
36      return XST_FAILURE;
37  }
38  // Read dip switch values
39  dip_check_prev = XGpio_DiscreteRead(&dip, 1);
40  // Load timer with delay in multiple of ONE_TENTH
41  XScuTimer_LoadTimer(TimerInstancePtr, ONE_TENTH*dip_check_prev);
42  // Set AutoLoad mode
43  XScuTimer_EnableAutoReload(TimerInstancePtr);
44  // Start the timer
45  XScuTimer_Start (TimerInstancePtr);
46
47  while(1)
48  {
49      if(psb_check > 0)
50      {
51          xil_printf("Push button pressed: Exiting\r\n");
52          XScuTimer_Stop(TimerInstancePtr);
53          break;
54      }
55      dip_check = XGpio_DiscreteRead(&dip, 1);
56      if (dip_check != dip_check_prev) {
57          xil_printf("DIP Switch Status %x, %x\r\n", dip_check_prev, dip_check);
58          dip_check_prev = dip_check;
59          // load timer with the new switch settings
60          XScuTimer_LoadTimer(TimerInstancePtr, ONE_TENTH*dip_check);
61          count = 0;
62      }
63      if(XScuTimer_IsExpired(TimerInstancePtr)) {
64          // clear status bit
65          XScuTimer_ClearInterruptStatus(TimerInstancePtr);
66          // output the count to LED and increment the count
67          LED_IP_mWriteReg(XPAR_LED_IP_S_AXI_BASEADDR, 0, count);
68          count++;
69      }
70  }
71  }
72  return 0;
73 }

```

Figura 12. Estableciendo los breakpoints

- 4-1-4.** Hacer click en el botón **Resume** () o F8 para continuar ejecutando el programa hasta alcanzar el primer breakpoint.

En la pestaña Variables notar que la variable *count* debe tener un valor distinto a 0.

- 4-1-5.** Hacer click en el botón **Step Over** () o presionar F6 para ejecutar una sentencia. Luego de

avanzar un paso, notará que el valor de la variable **count** cambió a 0.

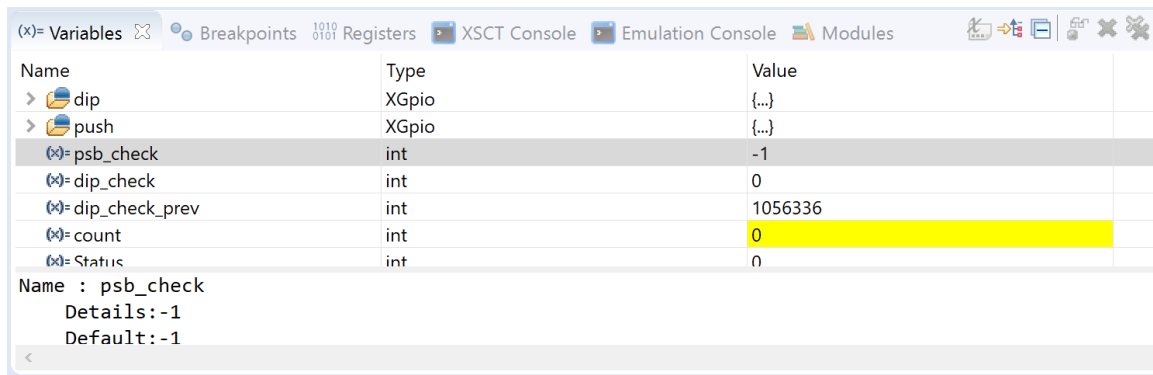


Figura 13. Visualizando el contenido de variables

- 4-1-6.** Hacer click nuevamente en el botón **Resume** y verá que varias líneas del código son ejecutadas y la ejecución es suspendida en el tercer breakpoint. El segundo breakpoint es saltado. Esto ocurre porque la inicialización del timer fue correcta.
- 4-1-7.** Hacer click en el botón **Step Over** (F6) para ejecutar una sentencia. Luego de avanzar un paso, notará que el valor de la variable **dip_check_prev** cambió a un valor dependiente de la configuración de los switches en su placa. En el caso de la imagen se muestra una configuración de los switches de "11".

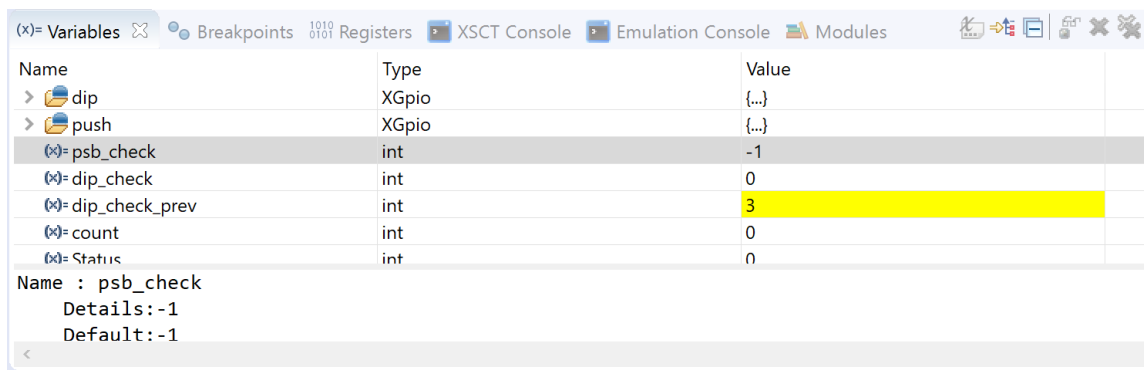


Figura 14. Visualizando el contenido de una nueva variable

- 4-1-8.** Hacer click en la pestaña memory. Si no la ve, ir a **Window ► Show View ► Memory**.

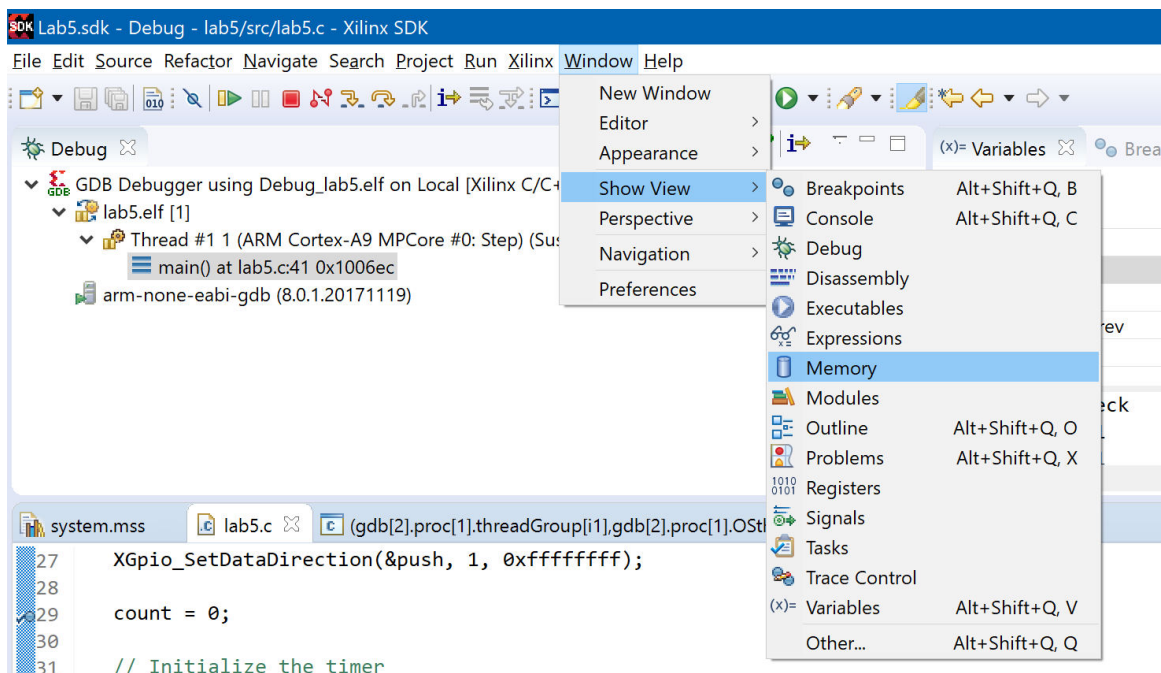


Figura 15. Haciendo visible un sector de memoria

4-1-9. Hacer click en el botón  para agregar un monitor de memoria

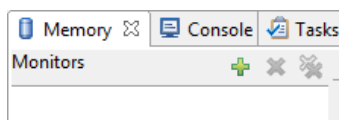


Figura 16. Monitor de ubicaciones de memoria

4-1-10. Introducir la dirección del counter load register (0xF8F00600), y hacer click en **OK**.

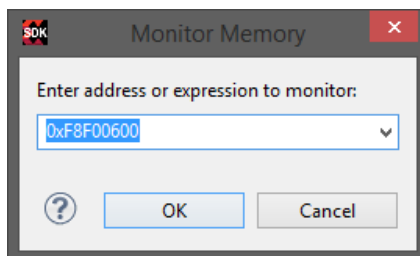


Figura 17. Monitoreando una dirección de memoria

La dirección se construye como:

DIRECCIÓN BASE DEL PERIFÉRICO + OFFSET DEL REGISTRO

En nuestro caso sería:

XPAR_PS7_SCUTIMER_0_BASEADDR + XSCUTIMER_LOAD_OFFSET

La dirección base se puede encontrar en el archivo xparameters.h, y el offset haciendo doble click en el archivo xscutimer.h en la ventana outline seguido de otro doble click sobre xscutimer_hw.h.


```
# XSCUTIMER_HW_H
# xil_types.h
# xil_io.h
# xil_assert.h
# XSCUTIMER_LOAD_OFFSET
# XSCUTIMER_COUNTER_OFFSET
# XSCUTIMER_CONTROL_OFFSET
# XSCUTIMER_ISR_OFFSET
```

Figura 18. Offset de la memoria

- 4-1-11.** Asegurarse de que los switches no están colocados en “00” (posición de reset) y hacer click en el botón **Step Over** para ejecutar una sentencia que cargará el registro del timer.

Notar que la dirección 0xF8F00600 se ha coloreado de rojo ya que el contenido ha cambiado. Verificar que el contenido es el mismo que el valor: $\text{dip_check_prev} * 32500000$. Verá su equivalente en hexadecimal (mostrando bytes en el orden 0 -> 3).

Por ej. para $\text{dip_check_prev} = 3$; el valor es 0x05CFBB60; (reversado: 0x60BBCF05)

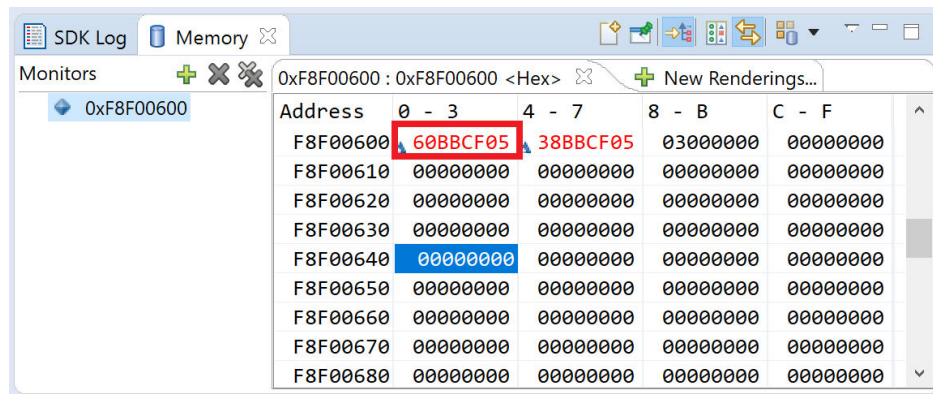


Figura 19. Visualizando los valores de los registros en memoria

- 4-1-12.** Hacer click en el botón **Resume** para continuar la ejecución del programa. El programa parará en la escritura sobre el Puerto LED (saltando el cuarto breakpoint ya que el botón central no ha sido presionado).

Notar que el valor del registro contador es distinto que el previo ya que el timer fue iniciado y la cuenta regresiva ha empezado.

- 4-1-13.** Hacer click en el botón **Step Over** para ejecutar una sentencia que escribirá en el puerto LED y que debería apagar los LEDs en $\text{count}=0$.

- 4-1-14.** Hacer double-click sobre el quinto breakpoint, el que escribe al puerto LED, para que el programa pueda ejecutarse libremente.

- 4-1-15.** Hacer click sobre el botón **Resume** para continuar con la ejecución de programa. En esta oportunidad correrá continuamente cambiando el patrón de titlado del LED en función de la configuración de frecuencia de los switches.

- 4-1-16.** Modificar los switches para cambiar el retardo y observar el efecto.

4-1-17. Presionar un botón y observar que el programa se suspende en el cuarto breakpoint. El contenido del registro timer así como el registro de control (offset 0x08) están en rojo ya que el valor del contador ha cambiado como así también el valor del registro de control debido a la llamada de la función de detención del timer. (En el monitor de memoria, debe hacer botón derecho sobre la dirección que está siendo monitoreada y hacer click en *Reset* para refrescar la vista de la memoria).

4-1-18. Cerrar la sesión de la terminal.

4-1-19. Cerrar el SDK y Vivado.

4-1-20. Desconectar la placa.

Conclusión

Este trabajo lo llevó a través de un desarrollo de software que utilizó un timer de CPU. Estudió la documentación de la API, usó llamadas a función apropiadas y consiguió la funcionalidad buscada, que verificó en hardware. Adicionalmente, usó el depurador del SDK para ver el contenido de variables y de la memoria, y fue avanzando, paso a paso, a través de diferentes partes del código.