Software Writing for Timer and Debugging

Introduction

This lab guides you through the process of writing a software application that utilizes the private timer of the CPU. You will refer to the timer's API in the SDK to create and debug the software application. The application you will develop will monitor the switch values and increment a count on the LEDs. The application will exit when the center push button is pressed.

Objectives

After completing this lab, you will be able to:

- Utilize the CPU's private timer in polled mode
- Use SDK Debugger to set break points and view the content of variables and memory

Procedure

This lab is separated into steps that consist of general overview statements that provide information on the detailed instructions that follow. Follow these detailed instructions to progress through the lab.

This lab comprises 4 primary steps: Open the project in Vivado, create a SDK software project, verify operation in hardware, and launch the debugger and debug the design.

Design Description

You will use the hardware design created in lab 3 to use CPU's private timer (see **Figure 1**). You will develop the code to use it.

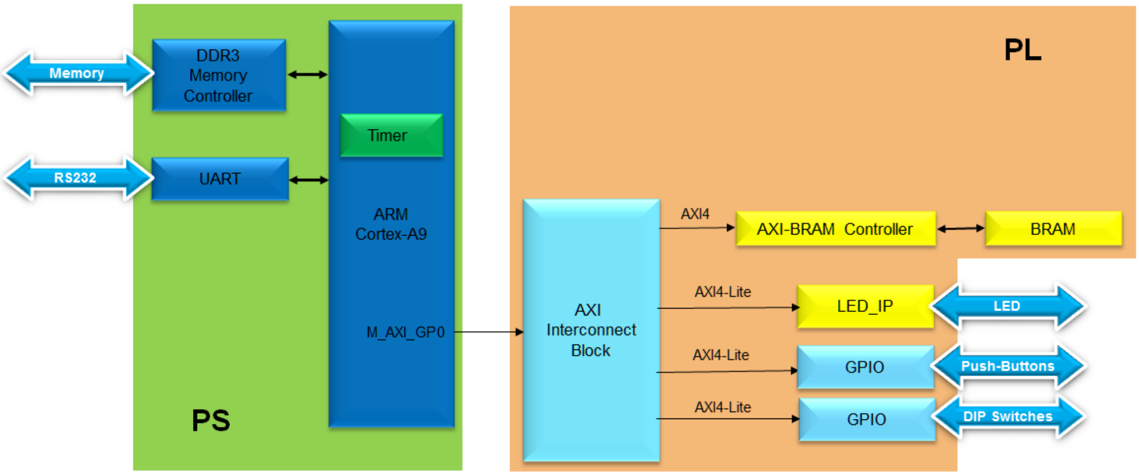
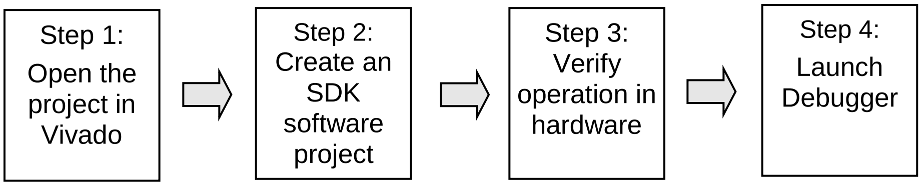


Figure 1. Design updated from Previous Lab

General Flow for this Lab


Step 1: Open the project in Vivado
Step 2: Create an SDK software project
Step 3: Verify operation in hardware
Step 4: Launch Debugger

In the instructions below;

{***sources***} refers to: C:\Xilinx_trn\Zynq_base\lab_sources
{***labs***} refers to : C:\Xilinx_trn\Zynq_base

Open the Project in Vivado Step 1

1-1. Use the lab4 project from the last lab and save it as *lab5.* Open the project in Vivado and then export to SDK.

1-1-1. Start the Vivado if necessary and open the lab4 project (lab4.xpr) you created in the previous lab using the **Open Project** link in the Getting Started page.

1-1-2. Select **File > Project > Save As …** to open the *Save Project As* dialog box. Enter **lab5** as the project name. Make sure that the *Create Project Subdirectory* option is checked, the project directory path is *C:/Xilinx_trn/Zynq_base* and click **OK**.

This will create the lab5 directory and save the project and associated directory with lab4 name.

Since we will be using the private timer of the CPU, ***which is always present****, we don't need to modify the hardware design.*

1-1-3. Open the Block Design. You may notice that the status changes to synthesis and implementation out-of-date as the project was *saved as*. Since the bitstream is already generated and will be in the exported directory, we can safely ignore any warning about this.

1-2. Export the hardware along with the generated bitstream to SDK.

1-2-1. Click **File > Export > Export Hardware.**

1-2-1.1. Click on the checkbox of *Include the bitstream*

1-2-1.2. Click **OK** and then click **Yes** to overwrite.

1-2-2. Select **File > Launch SDK** and click **OK.**

1-2-3. Click **Yes**.

Create an SDK Software Project Step 2

2-1. Close previously created projects. Create a new empty application project called lab5 utilizing already existing standalone_bsp_0 software platform. Import the lab5.c source file.

2-1-1. In the *Project Explorer* in SDK, ***right click+cntl*** on *lab4, lab4_bsp* and *system_wrapper_hw_platfrom_2* and select **Close Project**

2-1-2. Select ***File > New*** > ***Application Project*****.**

2-1-3. Name the project **lab5**, and for the board Support Package, (Leave *Create New* for the *Board Support Package* and name as *lab5_bsp)* and click **Next.**

2-1-4. Select **Empty Application** and click **Finish.**

2-1-5. Select ***lab5 > src*** in the project explorer, right-click, and select **Import.**

2-1-6. Expand **General** category and double-click on **File System.**

2-1-7. Browse to C:\Xilinx_trn\Zynq_base\lab_sources\lab5 folder, click **OK**.

2-1-8. Select **lab5.c** and click **Finish.**

You will notice that there are multiple compilation errors. This is expected as the code is incomplete. You will complete the code in this lab.

2-2. Refer to the Scutimer API documentation.

2-2-1. Open the **system.mss** from **lab5_bsp**

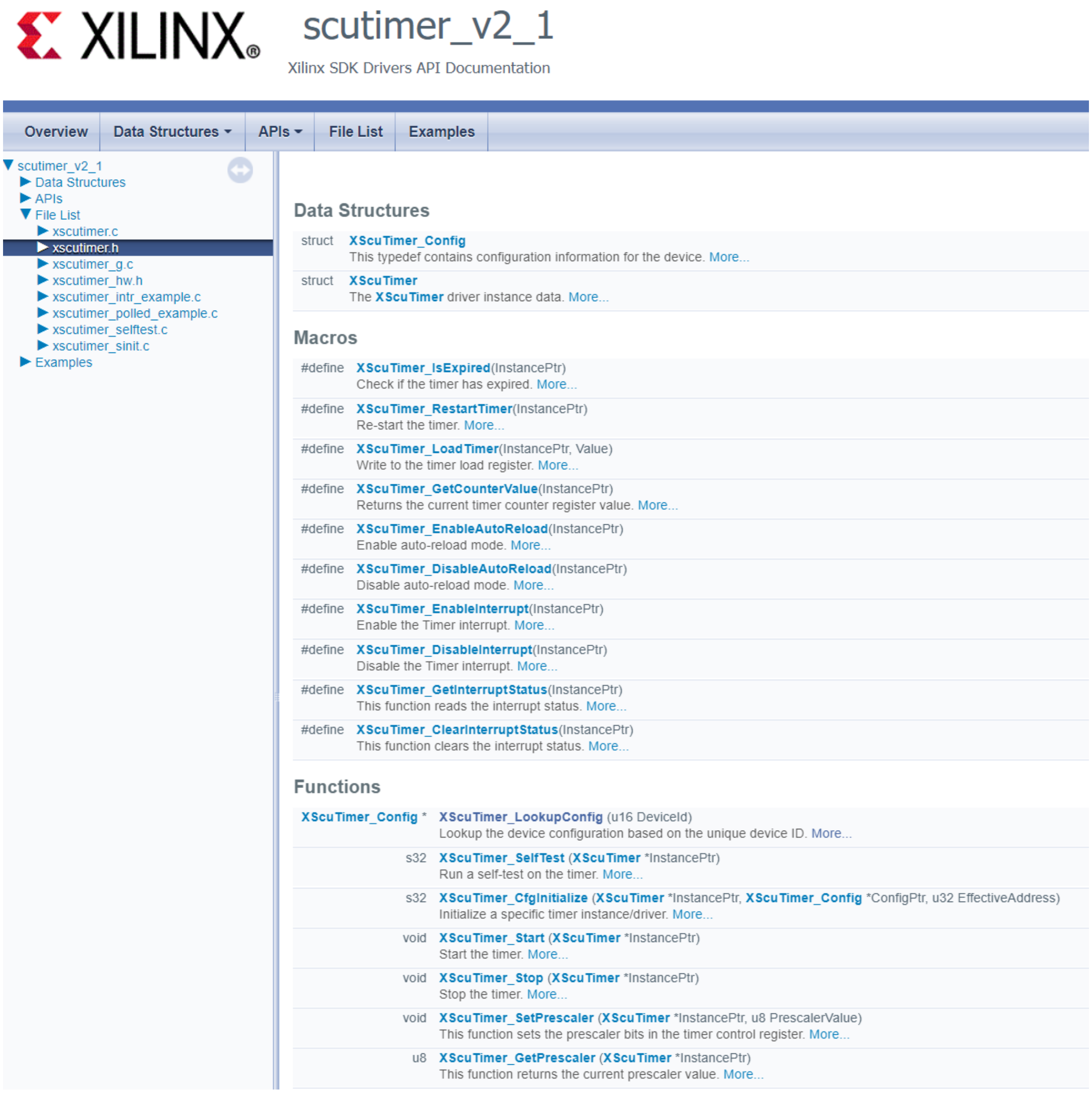
2-2-2. Click on **Documentation** link corresponding to **scutimer** (*ps7_scutimer*) peripheral under the *Peripheral Drivers* section to open the documentation in a default browser window.

2-2-3. In the window appeared click on the **File List** link to see available files related to the private timer API.

2-2-4. In the window appeared click on the **xscutimer.h** file.

Look at the *XScuTimer_LookupConfig*( ) and *XScuTimer_CfgInitialize*( ) API functions which must be called before the timer functionality can be accessed.

Look at various functions available to interact with the timer hardware.



Data Structures

struct	**XScuTimer_Config** This typedef contains configuration information for the device. More...
struct	**XScuTimer** The **XScuTimer** driver instance data. More...

Macros

#define	**XScuTimer_IsExpired**(InstancePtr) Check if the timer has expired. More...
#define	**XScuTimer_RestartTimer**(InstancePtr) Re-start the timer. More...
#define	**XScuTimer_LoadTimer**(InstancePtr, Value) Write to the timer load register. More...
#define	**XScuTimer_GetCounterValue**(InstancePtr) Returns the current timer counter register value. More...
#define	**XScuTimer_EnableAutoReload**(InstancePtr) Enable auto-reload mode. More...
#define	**XScuTimer_DisableAutoReload**(InstancePtr) Disable auto-reload mode. More...
#define	**XScuTimer_EnableInterrupt**(InstancePtr) Enable the Timer interrupt. More...
#define	**XScuTimer_DisableInterrupt**(InstancePtr) Disable the Timer interrupt. More...
#define	**XScuTimer_GetInterruptStatus**(InstancePtr) This function reads the interrupt status. More...
#define	**XScuTimer_ClearInterruptStatus**(InstancePtr) This function clears the interrupt status. More...

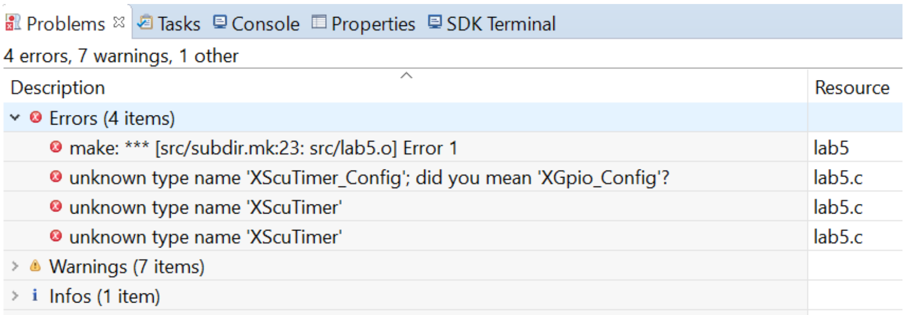
Functions

XScuTimer_Config *	**XScuTimer_LookupConfig** (u16 DeviceId) Lookup the device configuration based on the unique device ID. More...
s32	**XScuTimer_SelfTest** (**XScuTimer** *InstancePtr) Run a self-test on the timer. More...
s32	**XScuTimer_CfgInitialize** (**XScuTimer** *InstancePtr, **XScuTimer_Config** *ConfigPtr, u32 EffectiveAddress) Initialize a specific timer instance/driver. More...
void	**XScuTimer_Start** (**XScuTimer** *InstancePtr) Start the timer. More...
void	**XScuTimer_Stop** (**XScuTimer** *InstancePtr) Stop the timer. More...
void	**XScuTimer_SetPrescaler** (**XScuTimer** *InstancePtr, u8 PrescalerValue) This function sets the prescaler bits in the timer control register. More...
u8	**XScuTimer_GetPrescaler** (**XScuTimer** *InstancePtr) This function returns the current prescaler value. More...

Figure 2. Useful Functions

2-3. Correct the errors

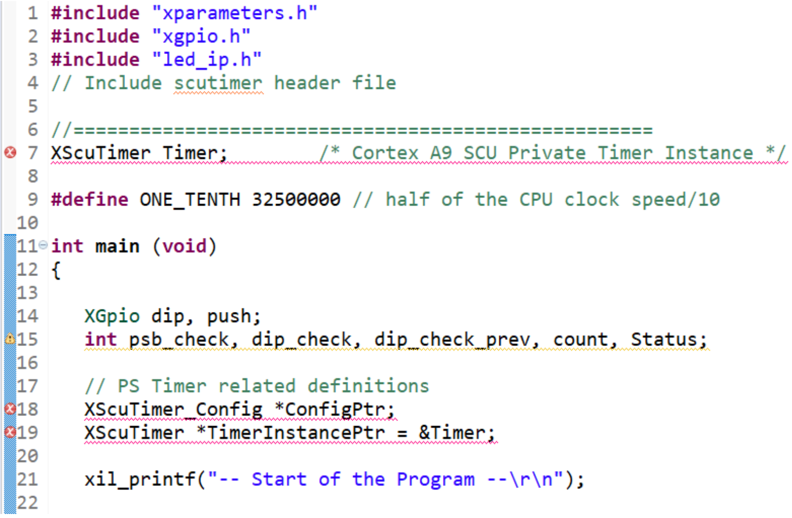
2-3-1. In SDK, in the **Problems** tab, double-click on the *unknown type name* **x** for the parse error.



4 errors, 7 warnings, 1 other

Description	Resource
Errors (4 items)	
make: *** [src/subdir.mk:23: src/lab5.o] Error 1	lab5
unknown type name 'XScuTimer_Config'; did you mean 'XGpio_Config'?	lab5.c
unknown type name 'XScuTimer'	lab5.c
unknown type name 'XScuTimer'	lab5.c
Warnings (7 items)	
Infos (1 item)	

This will open the source file and bring you around to the error place.



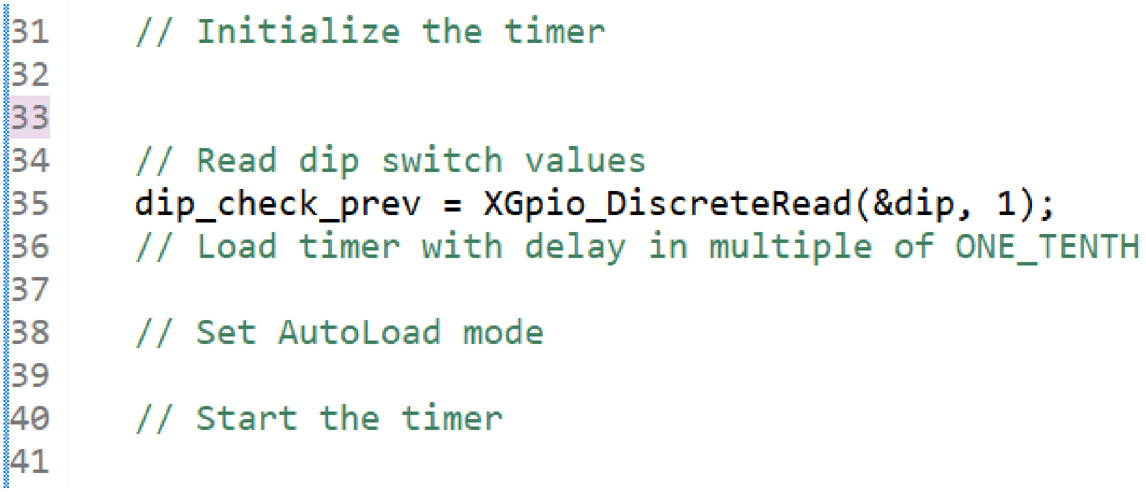
```
 1 #include "xparameters.h"
 2 #include "xgpio.h"
 3 #include "led_ip.h"
 4 // Include scutimer header file
 5
 6 //====================================================
 7 XScuTimer Timer;        /* Cortex A9 SCU Private Timer Instance */
 8
 9 #define ONE_TENTH 32500000 // half of the CPU clock speed/10
10
11 int main (void)
12 {
13
14     XGpio dip, push;
15     int psb_check, dip_check, dip_check_prev, count, Status;
16
17     // PS Timer related definitions
18     XScuTimer_Config *ConfigPtr;
19     XScuTimer *TimerInstancePtr = &Timer;
20
21     xil_printf("-- Start of the Program --\r\n");
22
```

Figure 3. First error

2-3-2. Add the include statement at line 5 of the file for the *XScuTimer.h*. Save the file and the errors should disappear.

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

2-3-3. Scroll down the file and notice that there are few lines intentionally left blank with some guiding comments.



```
31    // Initialize the timer
32
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
41
```

Figure 4. Fill in Missing Code

- *Lines 32, 33: The timer needs to be initialized,*
- *Line 37: The timer needs to be loaded with the value ONE_TENTH*dip_check_prev,*
- *Line 39: AutoLoad needs to be enabled,*
- *Line 41: The timer needs to be started.*

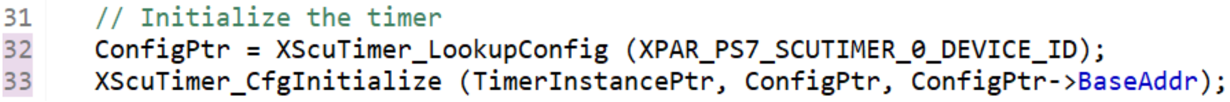
2-3-4. Using the API functions list, fill those lines. Save the file and correct errors if any. (Use the completed code further on in this workbook as a guide if necessary.)

Functions needed:	XScuTimer_LookupConfig( )
	XScuTimer_CfgInitialize( )
	XScuTimer_LoadTimer( )
	XScuTimer_EnableAutoReload( )
	XScuTimer_Start( )

2-3-4.1. In lines 32, 33 of lab5.c add the following functions:

ConfigPtr = XScuTimer_LookupConfig (XPAR_PS7_SCUTIMER_0_DEVICE_ID);

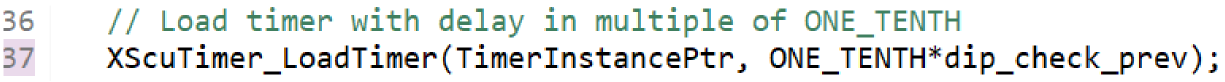
XScuTimer_CfgInitialize (TimerInstancePtr, ConfigPtr, ConfigPtr->BaseAddr);



```
31    // Initialize the timer
32    ConfigPtr = XScuTimer_LookupConfig (XPAR_PS7_SCUTIMER_0_DEVICE_ID);
33    XScuTimer_CfgInitialize (TimerInstancePtr, ConfigPtr, ConfigPtr->BaseAddr);
```

2-3-4.2. In line 37 of lab5.c add the following function:

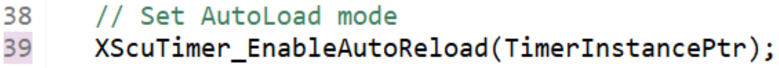
XScuTimer_LoadTimer(TimerInstancePtr, ONE_TENTH*dip_check_prev);



```
36    // Load timer with delay in multiple of ONE_TENTH
37    XScuTimer_LoadTimer(TimerInstancePtr, ONE_TENTH*dip_check_prev);
```

2-3-4.3. In line 39 of lab5.c add the following function:

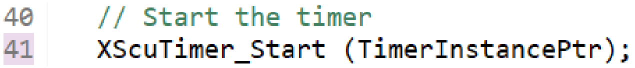
XScuTimer_EnableAutoReload(TimerInstancePtr);



```
38    // Set AutoLoad mode
39    XScuTimer_EnableAutoReload(TimerInstancePtr);
```

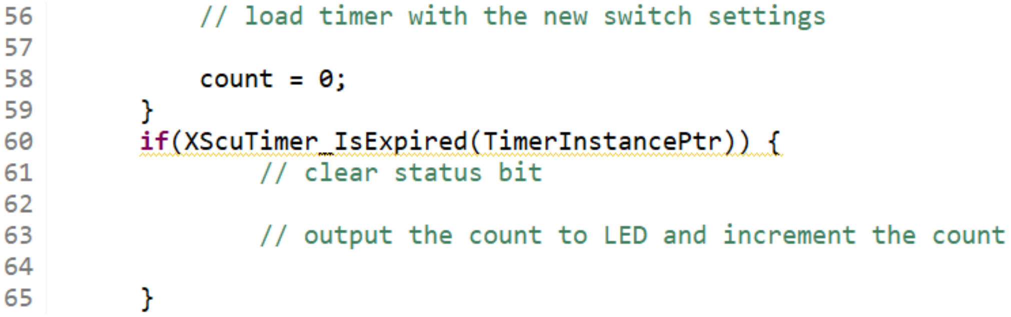
2-3-4.4. In line 41 of lab5.c add the following function:

XScuTimer_Start (TimerInstancePtr);



```
40    // Start the timer
41    XScuTimer_Start (TimerInstancePtr);
```

2-3-5. Scroll down the file further (down to line 56) and notice that there are few more lines intentionally left blank with some guiding comments.



```
56            // load timer with the new switch settings
57
58            count = 0;
59        }
60        if(XScuTimer_IsExpired(TimerInstancePtr)) {
61                // clear status bit
62
63                // output the count to LED and increment the count
64
65        }
```

Figure 5. More Code to be completed

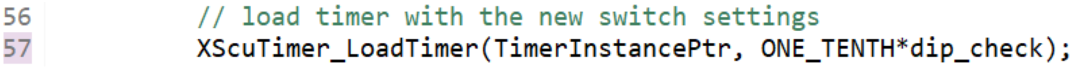
- *Line 57: The value of ONE_TENTH*dip_check needs to be written to the timer to update the timer.*
- *Line 62: The InterruptStatus needs to be cleared.*
- *Line 64: , and the LED needs to be written to, and the count variable incremented.*

Functions needed: *XScuTimer_LoadTimer( )*
XScuTimer_ClearInterruptStatus ( )
LED_IP_mWriteReg ( )

2-3-6. Using the API functions list, complete those lines.

2-3-6.1. In line 57 of lab5.c add the following function:

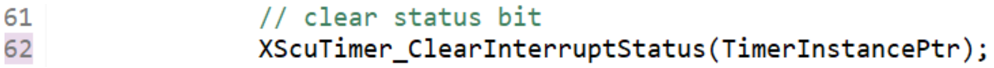
XScuTimer_LoadTimer(TimerInstancePtr, ONE_TENTH*dip_check);



```
56            // load timer with the new switch settings
57            XScuTimer_LoadTimer(TimerInstancePtr, ONE_TENTH*dip_check);
```

2-3-6.2. In line 62 of lab5.c add the following function:

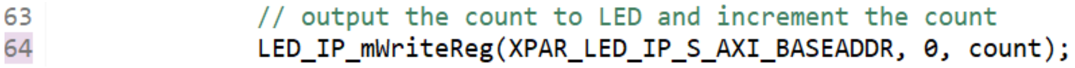
XScuTimer_ClearInterruptStatus(TimerInstancePtr);



```
61              // clear status bit
62              XScuTimer_ClearInterruptStatus(TimerInstancePtr);
```

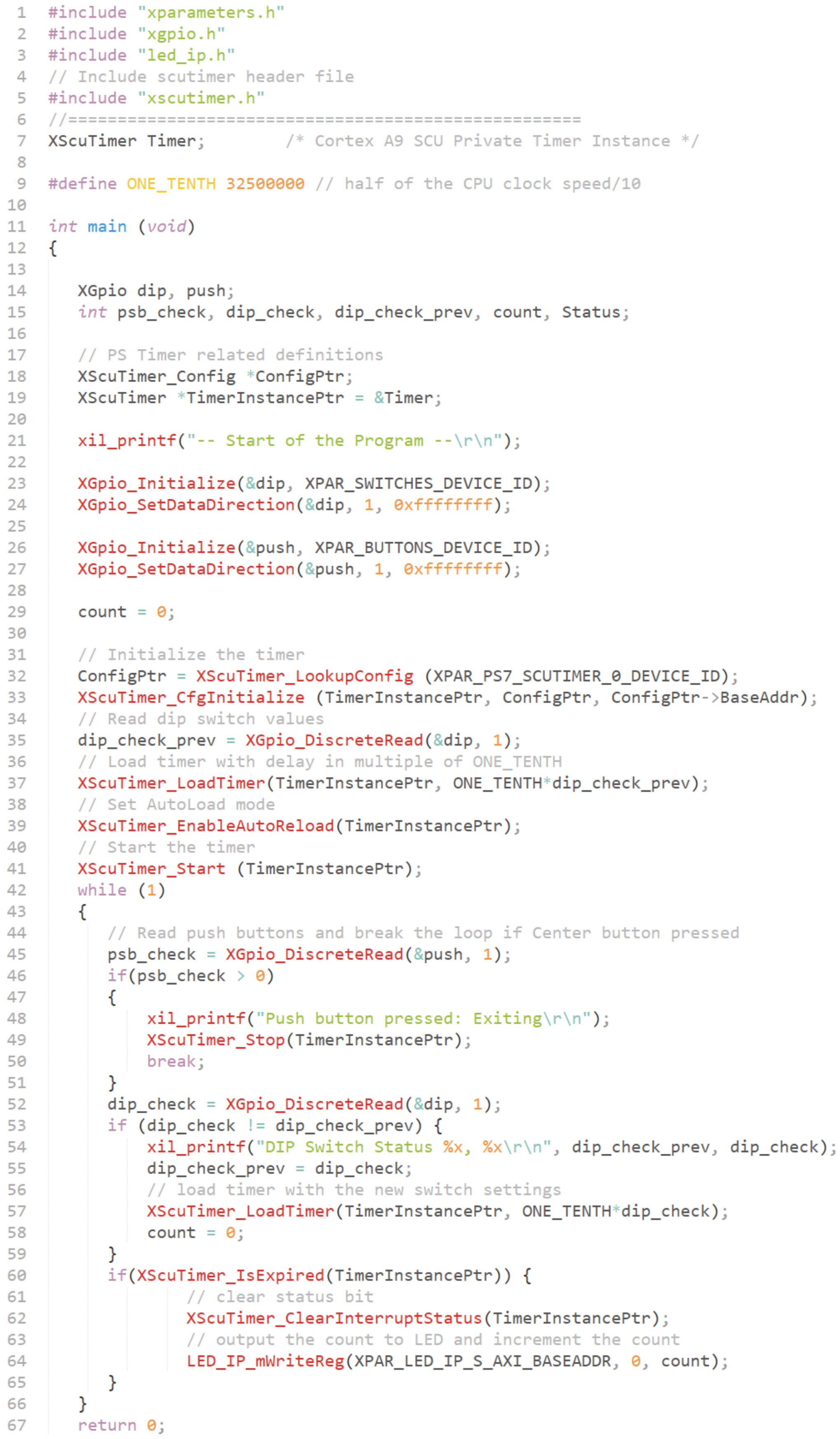
2-3-6.3. In line 64 of lab5.c add the following function:

LED_IP_mWriteReg(XPAR_LED_IP_S_AXI_BASEADDR, 0, count);



```
63              // output the count to LED and increment the count
64              LED_IP_mWriteReg(XPAR_LED_IP_S_AXI_BASEADDR, 0, count);
```

2-3-7. Save the file and correct errors if necessary.



```c
#include "xparameters.h"
#include "xgpio.h"
#include "led_ip.h"
// 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);
    XScuTimer_CfgInitialize (TimerInstancePtr, ConfigPtr, ConfigPtr->BaseAddr);
    // 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);
        }
    }
    return 0;
```

Figure 6. The completed Code

Verify Operation in Hardware Step 3

3-1. Connect the board with micro-usb cable(s) and power it ON. Establish the serial communication using SDK’s Terminal tab.

3-1-1. Make sure that micro-USB cable(s) is(are) connected between the board and the PC. Turn ON the power

3-1-2. Select the Terminal tab. If it is not visible then select **Window > Show view > Terminal**.

3-1-3. Click on and if required, select appropriate COM port (depends on your computer), and configure it with the parameters as shown. (These settings may have been saved from previous lab).

3-2. Program the FPGA by selecting Xilinx > Program FPGA and assigning system_wrapper.bit file. Run the TestApp application and verify the functionality.

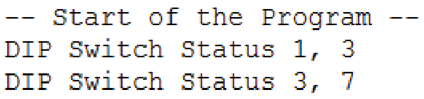
3-2-1. Select **Xilinx** > **Program FPGA.**

3-2-2. Click the **Program** button to program the FPGA.

3-2-3. Select **lab5** in *Project Explorer*, right-click and select **Run As > Launch on Hardware (GDB)** to download the application, execute ps7_init, and execute lab5.elf

Depending on the switch settings you expect to see LEDs implementing a binary counter with corresponding delay, BUT you don’t see it. There is a bug in the source code.

Flip the DIP switches and verify that in the Terminal window the previous and current switch settings are displayed whenever you flip switches.



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

Figure 7. Terminal window output

Launch Debugger Step 4

4-1. Launch Debugger and debug

4-1-1. Right-click on the **Lab5** project in the Project Explorer view and select **Debug As > Launch on Hardware (GDB)**.

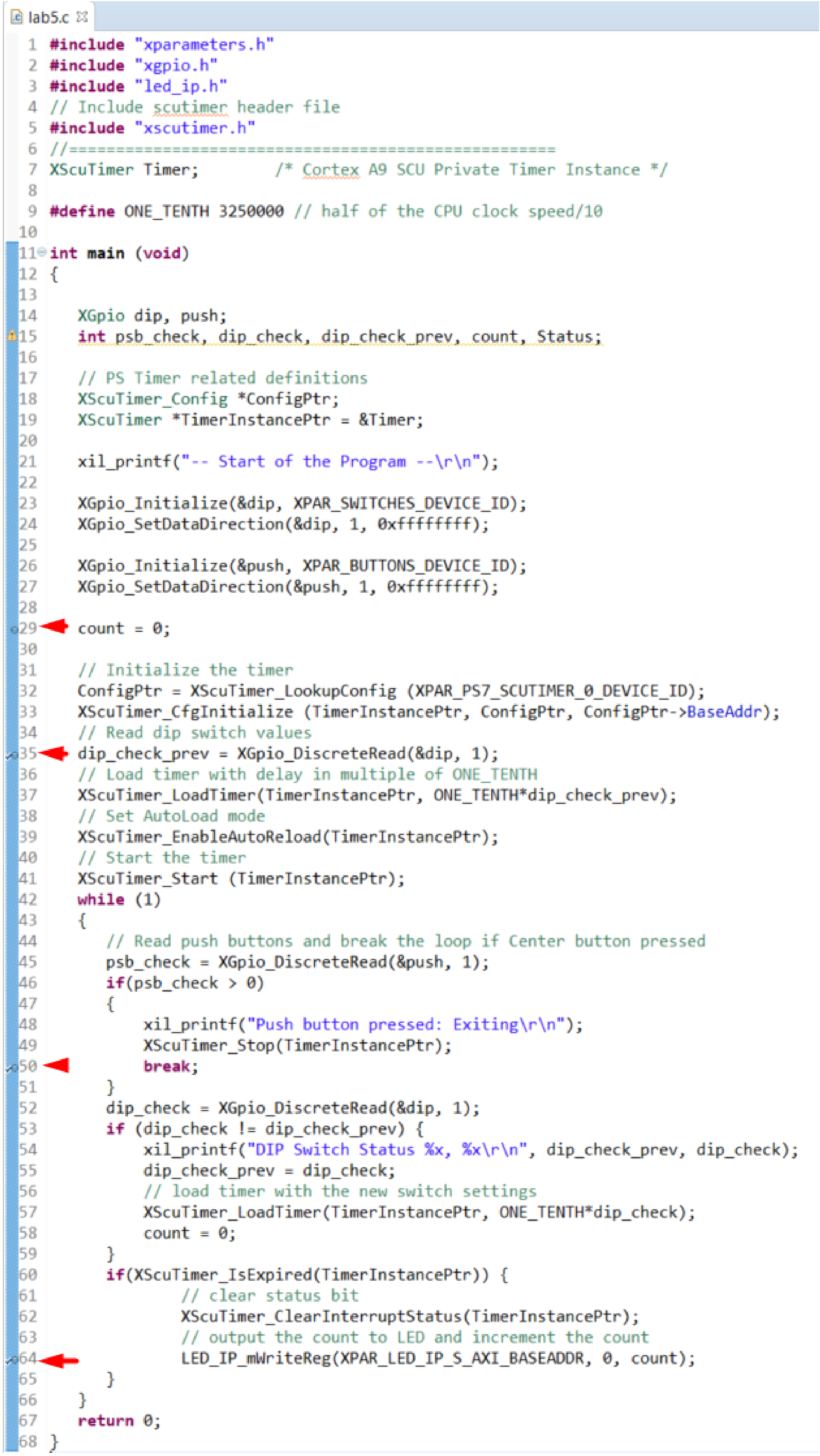
The lab5.elf file will be downloaded and if prompted, click **Yes** to stop the current execution of the program.

4-1-2. Click **Yes** if prompted to change to the *Debug perspective*.

At this point you could have added global variables by right clicking in the **Variables** tab and selecting **Add Global Variables …** All global variables would have been displayed and you could have selected desired variables. Since we do not have any global variables, we won't do it.

4-1-3. Double-click in the left margin to set a breakpoint on various lines in **lab5.c** shown below. A breakpoint has been set when a "tick" and blue circle appear in the left margin beside the line when the breakpoint was set. (The line numbers may be slightly different in your file.)

The first breakpoint is where count is initialized to 0. The second breakpoint is to catch if the timer initialization fails. The third breakpoint is when the program is about to read the dip switch settings. The fourth breakpoint is when the program is about to terminate due to pressing of center push button. The fifth breakpoint is when the timer has expired and about to write to LED.



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

#define ONE_TENTH 3250000 // 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);
    XScuTimer_CfgInitialize (TimerInstancePtr, ConfigPtr, ConfigPtr->BaseAddr);
    // 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);
        }
    }
    return 0;
}
```

Figure 8. Setting breakpoints

4-1-4. Click on the **Resume (▶)** button to continue executing the program up until the first breakpoint is reached.

In the Variables tab you will notice that the *count* variable may have value other than 0.

4-1-5. Click on the **Step Over** (↷) button or press F6 to execute one statement. As you do step over, you will notice that the **count** variable value changed to 0.

4-1-6. Click on the **Resume** button again and you will see that several lines of the code are executed and the execution is suspended at the third breakpoint..

4-1-7. Click on the **Step Over** (F6) button to execute one statement. As you do step over, you will notice that the **dip_check_prev** variable value changed to a value depending on the switch settings on your board.

4-1-8. Click on the memory tab. If you do not see it, go to **Window > Show View > Memory.**

4-1-9. Click the + sign to add a Memory Monitor

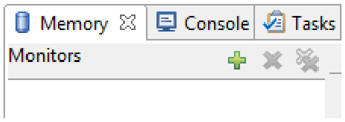
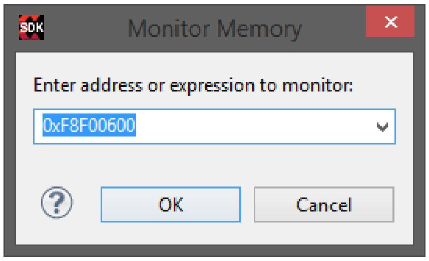


Figure 9. Monitor memory location

4-1-10. Enter the address for the private counter load register (0xF8F00600), and click **OK.**

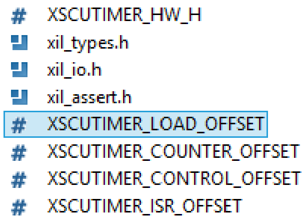


Enter address or expression to monitor:

0xF8F00600

Figure 10. Monitoring a Memory Address

You can find the address by looking at the xparameters.h file entry to get the base address (# XPAR_PS7_SCUTIMER_0_BASEADDR), and find the load offset double-clicking on the xscutimer.h in the outline window followed by double-clicking on the xscutimer_hw.h and then selecting XSCUTIMER_LOAD_OFFSET.



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

Figure 11. Memory Offset

4-1-11. Make sure the DIP Switches are not set to “0000” and click on the **Step Over** button to execute one statement which will load the timer register.

Notice that the address 0xF8F00604 has become red colored as the content has changed. Verify that the content is same as the value: `dip_check_prev`***32500000**. You will see hexadecimal equivalent (displaying bytes in the order 0 -> 3).

E.g. for dip_check_prev = 1; the value is 0x01EFE920; (reversed: 0x20E9EF01)

4-1-12. Click on the **Resume** button to continue execution of the program. The program will stop at the writing to the LED port (skipping fourth breakpoint as center push button as has not occurred).

Notice that the value of the counter register is changed from the previous one as the timer was started and the countdown had begun.

4-1-13. Click on the **Step Over** button to execute one statement which will write to the LED port and which should turn OFF the LEDs as the count=0.

4-1-14. Double-click on the fifth breakpoint, the one that writes to the LED port, so the program can execute freely.

4-1-15. Click on the **Resume** button to continue execution of the program. This time it will continuously run the program changing LED lit pattern at the switch setting rate.

4-1-16. Flip the switches to change the delay and observe the effect.

4-1-17. Press a push button and observe that the program suspends at the fourth breakpoint. The timer register content as well as the control register (offset 0x08) is red as the counter value had changed and the control register value changed due to timer stop function call. (In the Memory monitor, you may need to right click on the address that is being monitored and click *Reset* to refresh the memory view.)

Try to find and work around the issue with counter value on LED. Tip: Try to find in the source code where counter is incremented.

4-1-18. Correct source code. Save it. Rerun and check that it works correctly.

4-1-19. Terminate the session by clicking on the **Terminate (🟥)** button.

4-1-20. Exit the SDK and Vivado.

4-1-21. Power OFF the board.

Conclusion

This lab led you through developing software that utilized CPU's private timer. You studied the API documentation, used the appropriate function calls and achieved the desired functionality. You verified the functionality in hardware. Additionally, you used the SDK debugger to view the content of variables and memory, and stepped through various part of the code.