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 dip 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 4 to use CPU's private timer (see **Figure 1**). You will develop the code to use it.

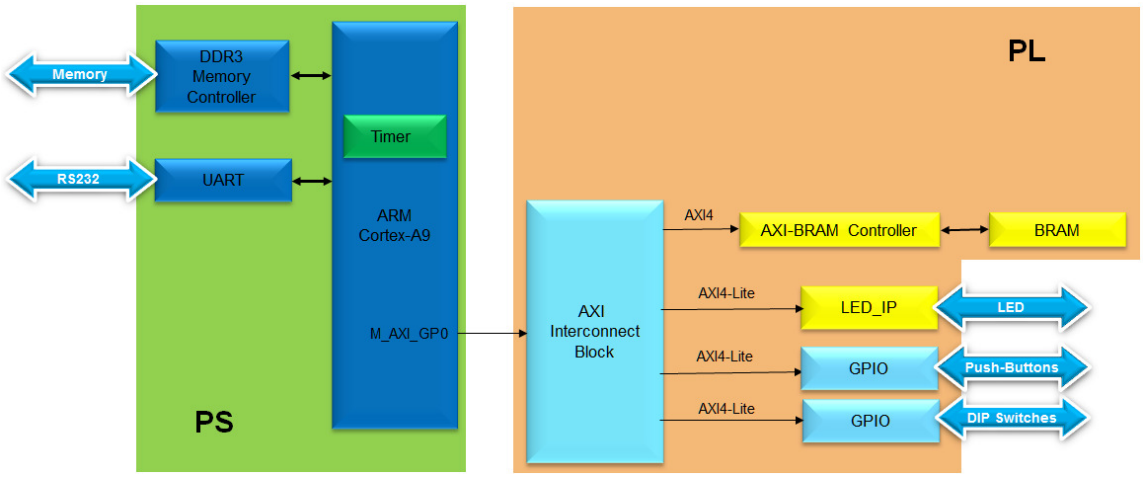
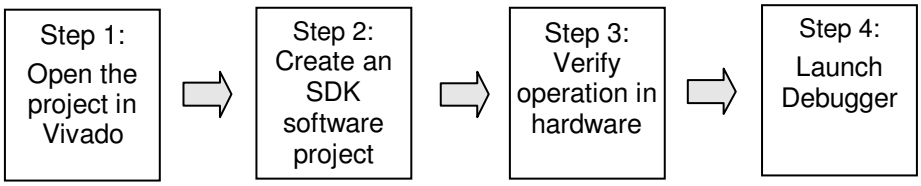


Figure 1. Design updated from Previous Lab

General Flow for this Lab



Open the Project in Vivado Step 1

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

1-1-1. If you wish to continue using the design that you created in the previous lab, open the lab4 project from the previous lab, or open the lab4 project in the labsolution directory, and *Save it as* lab5 to the labs/ directory

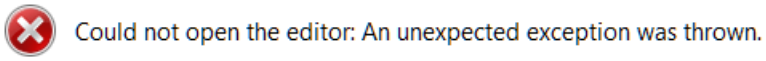
Since we will be using the private timer of the CPU, which is always present, we don't need to modify the hardware design.

1-1-2. Open the Block Design and notice that the status changes to synthesis and implementation out-of-date. Since the bitstream is already generated and will be in the exported directory, we can safely ignore the warning.

1-1-3. Launch SDK by selecting **File > Export > Export Hardware for SDK** in Vivado

1-1-4. Check the **Launch SDK** box only and click **OK**.

You may see the following message when SDK opens. Ignore this warning.



Create an SDK Software Project Step 2

2-1. 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 on lab4 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, select **Use Existing** (standalone_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:\xup\embedded\sources\lab5** folder, select **lab5.c** and click **OK**, and then 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. Select the **system.mss** tab (if it is not open, open it from standalone > system.mss)

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. Click on the **Files** link to see available files related to the private timer API.

2-2-4. Click on the **xscutimer.h** link to see various functions and data structures available in the API.

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, including

```
#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)
```

Figure 2. Useful Functions

2-3. Correct the errors.

2-3-1. In SDK, in the **Problems** tab, double-click on the first red *unknown type name* **x** for the parse error. This will open the source file and bring you around to the error place.

```
#include "led_ip.h"
// Include scutimer header file

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

#define ONE_SECOND 333000000 // half of the CPU clock speed

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;
```

Figure 3. First error

2-3-2. Add the include file for the *XScuTimer.h*. Save the file and the errors should disappear.

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

```
count = 0;

// Initialize the timer

// Read dip switch values
dip_check_prev = XGpio_DiscreteRead(&dip, 1);
// Load timer with delay in multiple of ONE_SECOND

// Set AutoLoad mode

// Start the timer
```

Figure 4. Fill in Missing Code

2-3-4. Using the API functions list, fill those lines. Save the file and correct errors if any.

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

```
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
```

Figure 5. More Code to be completed

2-3-6. Using the API functions list, complete those lines. Save the file and correct errors if necessary.

```
// Include scutimer header file
#include "xscutimer.h"
   // Initialize the timer
ConfigPtr = XScuTimer_LookupConfig(XPAR_PS7_SCUTIMER_0_DEVICE_ID);
Status = XScuTimer_CfgInitialize(TimerInstancePtr, ConfigPtr,
             ConfigPtr->BaseAddr);
if (Status != XST_SUCCESS) {
     return XST_FAILURE;
}

// Load timer with delay in multiple of ONE_SECOND
XScuTimer_LoadTimer(TimerInstancePtr, ONE_SECOND*dip_check_prev);
// Set AutoLoad mode
XScuTimer_EnableAutoReload(TimerInstancePtr);
// Start the timer
XScuTimer_Start(TimerInstancePtr);

    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_SECOND*dip_check_prev);
        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++;
     }
```

Figure 6. Portions of the completed Code

Verify Operation in Hardware Step 3

3-1. Make sure that the JP7 is set to select USB power. Connect the board with a micro-usb cable and power it ON. Establish the serial communication using SDK's Terminal tab.

3-1-1. Make sure that the JP7 is set to select USB power.

3-1-2. Make sure that a micro-USB cable is connected to the JTAG PROG connector (next to the power supply connector). Turn ON the power.

3-1-3. Select the 

tab. If it is not visible then select **Window > Show view > Terminal**.

3-1-4. 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 Tools > Program FPGA and assigning system_wrapper.bit file. Run the TestApp application and verify the functionality.

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

3-2-2. Click on the **Search** button of the Bitstream field, select *system_wrapper.bit*, and click **OK**.

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

3-2-4. 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 will see LEDs counting with corresponding delay.

Flip the DIP switches and verify that the LEDs light with corresponding delay according to the switch settings. Also notice 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 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.

```
27        XGpio_Initialize(&push, XPAR_BTNS_4BIT_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        // Read dip switch values
40        dip_check_prev = XGpio_DiscreteRead(&dip, 1);
41        // Load timer with delay in multiple of ONE_SECOND
42        XScuTimer_LoadTimer(TimerInstancePtr, ONE_SECOND*dip_check_prev);
43        // Set AutoLoad mode
44        XScuTimer_EnableAutoReload(TimerInstancePtr);
```

```
51            if(psb_check & 0x01)
52            {
53                xil_printf("Center pushbutton pressed: Exiting\r\n");
54                XScuTimer_Stop(TimerInstancePtr);
55                break;
56            }
57            dip_check = XGpio_DiscreteRead(&dip, 1);
58            if (dip_check != dip_check_prev) {
59                xil_printf("DIP Switch Status %x, %x\r\n", dip_check_prev, dip_check);
60                dip_check_prev = dip_check;
61                // load timer with the new switch settings
62                XScuTimer_LoadTimer(TimerInstancePtr, ONE_SECOND*dip_check);
63                count = 0;
64            }
65            if(XScuTimer_IsExpired(TimerInstancePtr)) {
66                    // clear status bit
67                    XScuTimer_ClearInterruptStatus(TimerInstancePtr);
68                    // output the count to LED and increment the count
69                    LED_IP_mWriteReg(XPAR_LED_IP_S_AXI_BASEADDR, 0, count);
70                    count++;
71            }
72        }
73        return 0;
74    }
```

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. The second breakpoint is skipped. This is due to successful timer initialization.

4-1-7. Click on the **Step Over** 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

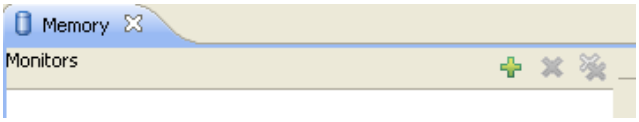


Figure 9. Monitor memory location

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

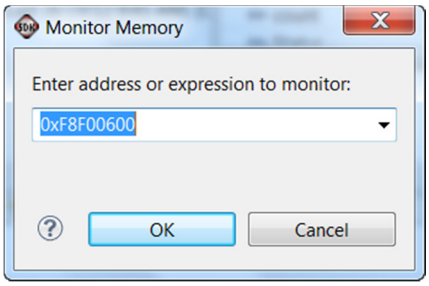


Figure 10. Monitoring a Memory Address

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.

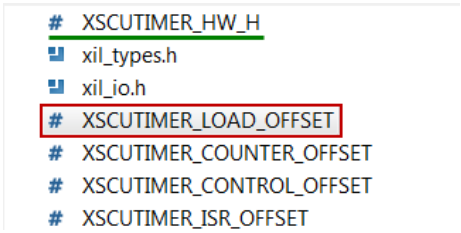


Figure 71. Memory Offset

4-1-11. 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*325000000`. You will see hexadecimal equivalent (displaying bytes in the order 0 -> 3).

E.g. for dip_check_prev = 1; the value is 0x13D92D40; (reversed: 0x402DD913)

4-1-12. Click on the **Resume** button to continue execution of the program. It 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 center 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.)

4-1-18. Terminate the session by clicking on the **Terminate (** **)** button.

4-1-19. Exit the SDK and Vivado.

4-1-20. 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.