# Laboratory 3 – I/O Operations – LED Applications

## I. LED Switch

### Objectives

- Develop an application to control the 16-bit LEDs using 16-bit slide switches.

### Activity Summary

1. Launch **Xilinx Vitis** after opening and exporting the hardware platform
2. Create a new application named **LedSwitch**
3. Program the FPGA, setup Run Configuration, and run the application
4. Repeat activity 3 for changes made to the application source file

### Specifications

The application implements real-time control where each slide switch directly controls its corresponding LED.
**Functionality:**

- When a switch is turned ON (up position), the corresponding LED turns ON
- When a switch is turned OFF (down position), the corresponding LED turns OFF
- All 16 switches independently control their respective LEDs in real-time

**Console Output:**

- Initialization successful: Displays "Successfully Initialized"
- Initialization failed: Displays "Initialization Failed" and program terminates

**Example:**

- Switch 0 ON → LED 0 lights up
- Switches 0, 3, 7 ON (others OFF) → LEDs 0, 3, 7 light up (others remain OFF)
- All switches ON → All 16 LEDs light up

### Activities

1. **Open and Export Hardware Design**

   Follow the procedure described in Lab 1 instructions to unzip and open **Lab 3 hardware Files** and export the hardware design files **Top.xsa (including bitstream)**.

**As we mentioned in Lab 1 instructions to export the hardware platform and create the XSA file:**

1. Click **File → Export → Export Hardware** (Figure 1a)
2. The Export Hardware Platform wizard opens (Figure 1b). Click **Next** to start the process.
3. In the Output window (Figure 1c), select **Include bitstream** to ensure the complete hardware implementation is included, then click **Next**.
4. In the Files window (Figure 1d), specify the XSA filename (default: "top") and export location. The system shows "The XSA will be written to: top.xsa". Click **Next**.
5. Review the summary (Figure 1e) showing that a hardware platform named 'top' will be created as top.xsa with post-implementation model and bitstream. Click **Finish** to complete the export.

The XSA file is now ready for use in Vitis IDE.

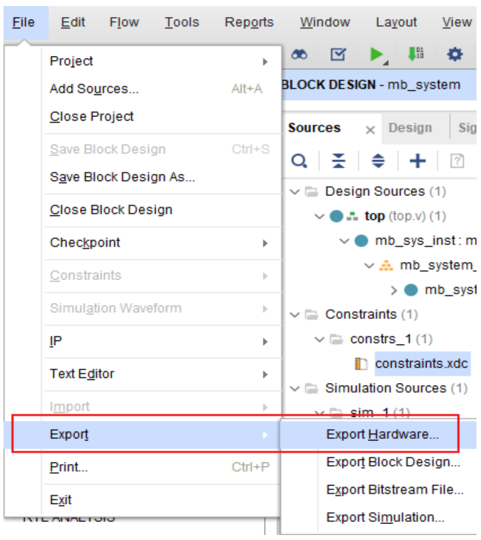
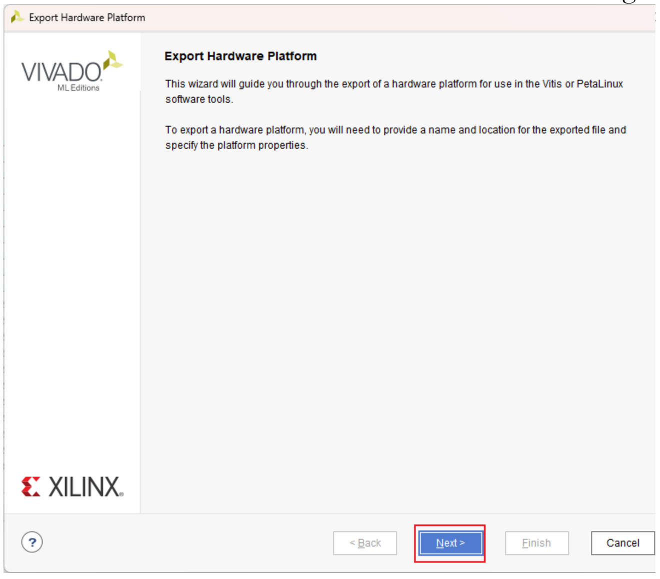


Figure 1(a)

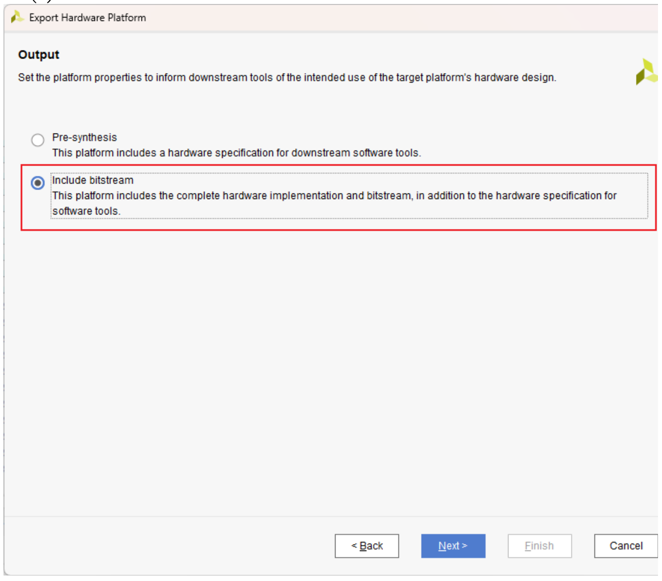

Export Hardware Platform

**Export Hardware Platform**

This wizard will guide you through the export of a hardware platform for use in the Vitis or PetaLinux software tools.

To export a hardware platform, you will need to provide a name and location for the exported file and specify the platform properties.

Figure 1(b)


Export Hardware Platform

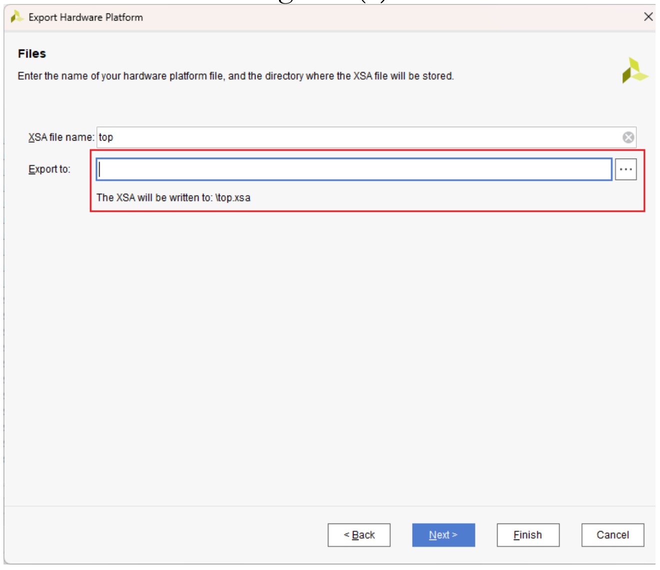
**Output**

Set the platform properties to inform downstream tools of the intended use of the target platform's hardware design.

○ Pre-synthesis
This platform includes a hardware specification for downstream software tools.

[✓ Include bitstream]
This platform includes the complete hardware implementation and bitstream, in addition to the hardware specification for software tools.

Figure 1(c)


Export Hardware Platform

**Files**

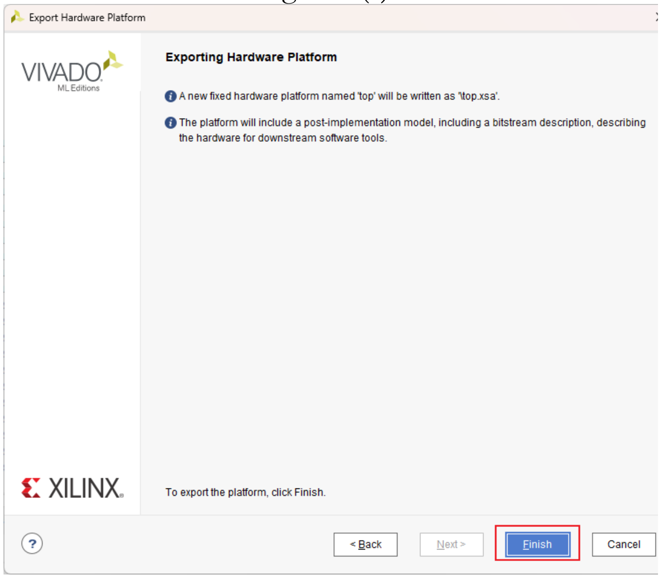
Enter the name of your hardware platform file, and the directory where the XSA file will be stored.

XSA file name: top

Export to:

The XSA will be written to: \top.xsa

Figure 1(d)


Export Hardware Platform

**Exporting Hardware Platform**

A new fixed hardware platform named 'top' will be written as '\top.xsa'.

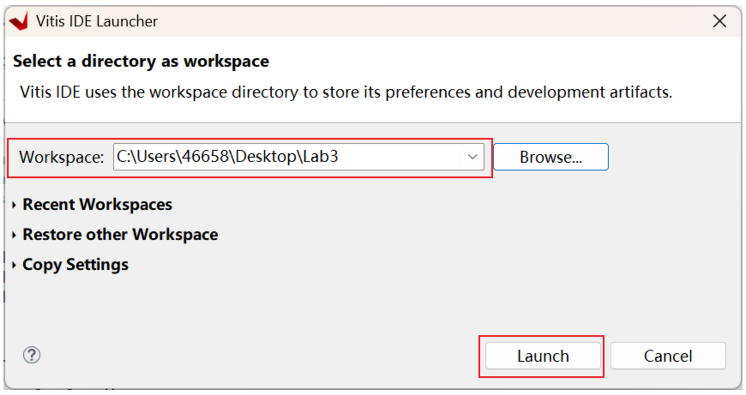
The platform will include a post-implementation model, including a bitstream description, describing the hardware for downstream software tools.

To export the platform, click Finish.

Figure 1(e)

## 2. **Launch Vitis**

Create the new folder in the PC, named Lab3, then open Vitis and select the file path of Lab3 as the workspace, and select **Launch**. **(Note: If you use Lab PC， please do not create the workspace on the uni-cloud, such as Desktop)**


Vitis IDE Launcher

**Select a directory as workspace**

Vitis IDE uses the workspace directory to store its preferences and development artifacts.

Workspace: C:\Users\46658\Desktop\Lab3

Figure 2

3. **Create a New Application Project**

Based on your images, here's the revised description for creating an application project:

**Create a New Application Project**

Launch Vitis IDE and follow these steps to create an application project:

**Step 1: Launch Vitis IDE** The Vitis IDE main interface will open (Figure 3a), displaying the welcome page. In the PROJECT section, click **Create Application Project** to create an application project.

**Step 2: Select Platform** In the New Application Project wizard (Figure 3b), select **Create a new platform from hardware (XSA)**, then click the **Browse...** button to locate the XSA file exported earlier from Vivado (the top.xsa file as shown in Figure 3c).

**Step 3: Configure Platform** On the platform selection page (Figure 3d), browse and select the top.xsa file from your project directory. The XSA file contains the hardware specification exported from Vivado.

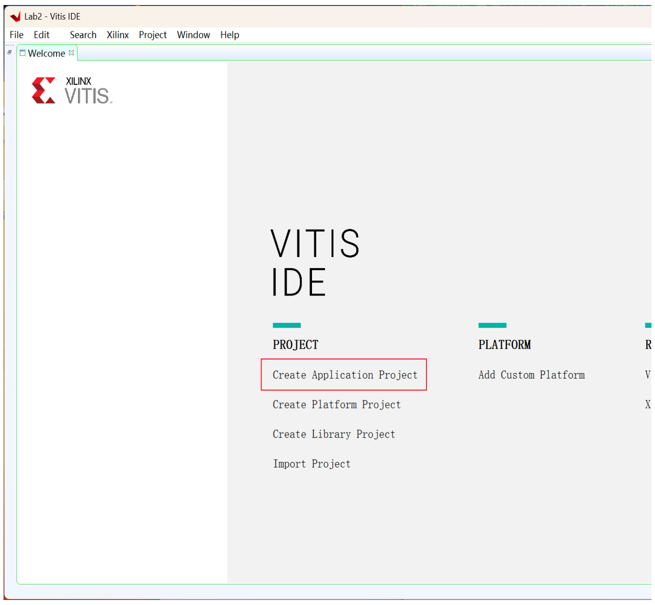
**Step 4: Configure Application Project** On the Application Project Details page (Figure 3e), enter the application project name (such as " **LedSwitch** "). The system will automatically create an associated system project " **LedSwitch**_system" with target processor shown as "microblaze_0".

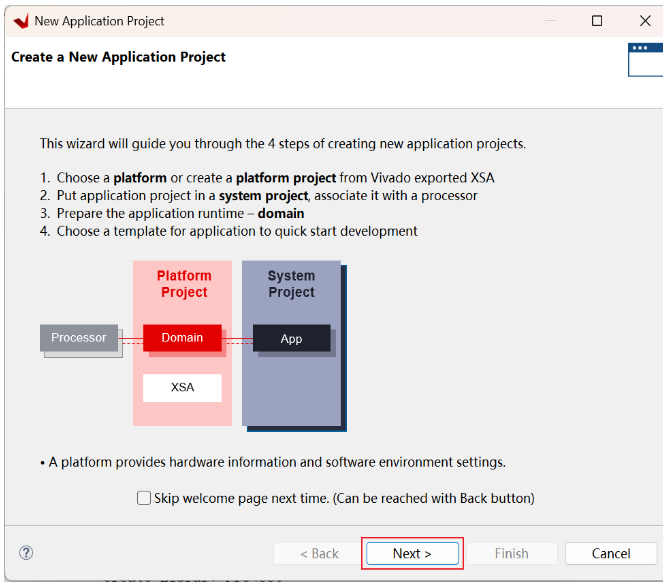
**Step 5: Select Domain** On the domain selection page (Figure 3f), the system displays details for the "standalone_microblaze_0" domain, including processor microblaze_0 and 32-bit architecture.

**Step 6: Choose Template** On the template selection page (Figure 3g), select the **Hello World** template from the embedded software development templates. This template creates a "Hello World" program written in C.

**Step 7: Complete Creation** Click **Finish** to complete the application project creation. Vitis IDE will create the application project and open the complete development environment, displaying the project structure in the Explorer.

Once created, you can begin developing and debugging your Lab3 embedded application.




New Application Project

**Create a New Application Project**

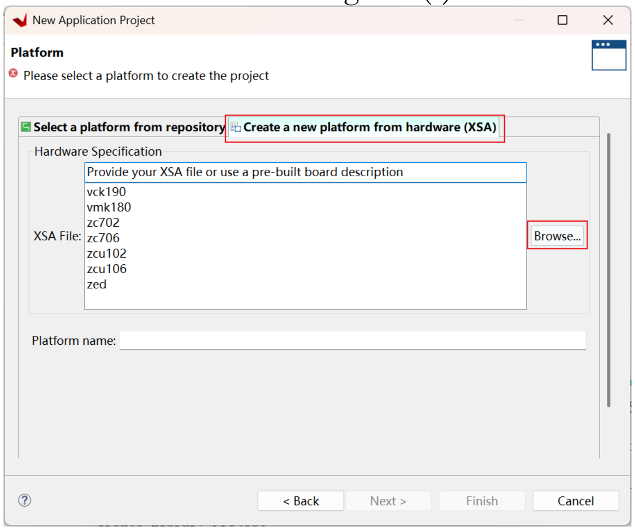
This wizard will guide you through the 4 steps of creating new application projects.

1. Choose a **platform** or create a **platform project** from Vivado exported XSA
2. Put application project in a **system project**, associate it with a processor
3. Prepare the application runtime – **domain**
4. Choose a template for application to quick start development

• A platform provides hardware information and software environment settings.

☐ Skip welcome page next time. (Can be reached with Back button)

Figure 3(a) Figure 3(b)


New Application Project

**Platform**

Please select a platform to create the project

Select a platform from repository
Create a new platform from hardware (XSA)

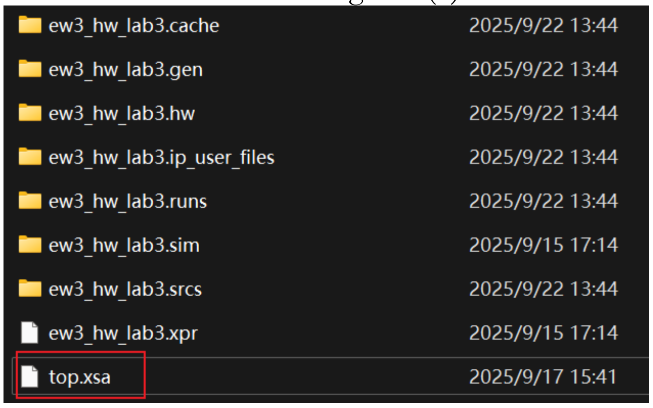
Hardware Specification

Provide your XSA file or use a pre-built board description

XSA File: vck190 vmk180 zc702 zc706 zcu102 zcu106 zed

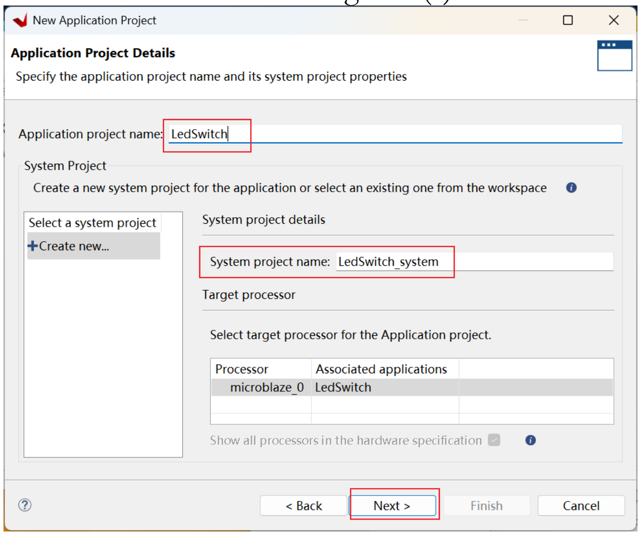
Platform name:

< Back
Next >
Finish
Cancel



```
ew3_hw_lab3.cache        2025/9/22 13:44
ew3_hw_lab3.gen          2025/9/22 13:44
ew3_hw_lab3.hw           2025/9/22 13:44
ew3_hw_lab3.ip_user_files 2025/9/22 13:44
ew3_hw_lab3.runs         2025/9/22 13:44
ew3_hw_lab3.sim          2025/9/15 17:14
ew3_hw_lab3.srcs         2025/9/22 13:44
ew3_hw_lab3.xpr          2025/9/15 17:14
top.xsa                  2025/9/17 15:41
```

Figure 3(c) Figure 3(d)


New Application Project

**Application Project Details**

Specify the application project name and its system project properties

Application project name: LedSwitch

System Project

Create a new system project for the application or select an existing one from the workspace

Select a system project

+Create new...

System project details

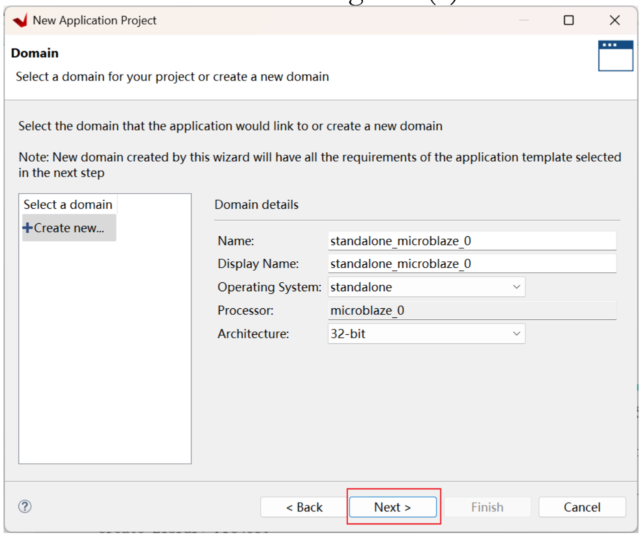
System project name: LedSwitch_system

Target processor

Select target processor for the Application project.

| Processor | Associated applications |
|---|---|
| microblaze_0 | LedSwitch |

Show all processors in the hardware specification


New Application Project

**Domain**

Select a domain for your project or create a new domain

Select the domain that the application would link to or create a new domain

Note: New domain created by this wizard will have all the requirements of the application template selected in the next step

Select a domain

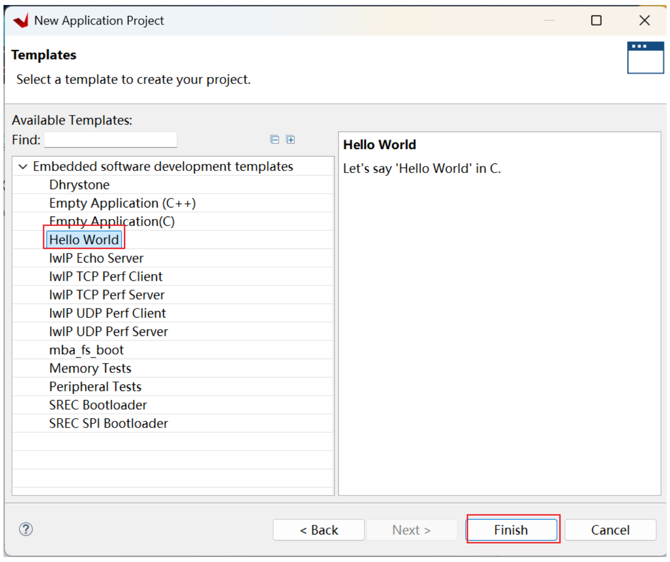
+Create new...

Domain details

| | |
|---|---|
| Name: | standalone_microblaze_0 |
| Display Name: | standalone_microblaze_0 |
| Operating System: | standalone |
| Processor: | microblaze_0 |
| Architecture: | 32-bit |

Figure 3(e)

Figure 3(f)


New Application Project

**Templates**

Select a template to create your project.

Available Templates:

Find:

Embedded software development templates
- Dhrystone
- Empty Application (C++)
- Empty Application(C)
- Hello World
- lwIP Echo Server
- lwIP TCP Perf Client
- lwIP TCP Perf Server
- lwIP UDP Perf Client
- lwIP UDP Perf Server
- mba_fs_boot
- Memory Tests
- Peripheral Tests
- SREC Bootloader
- SREC SPI Bootloader

**Hello World**

Let's say 'Hello World' in C.

Figure 3(g)

## 4. Create and Rename the Source File

- As shown in the Figure 4, in **Project Explorer**, expand the **LedSwitch** → **src** folder.
- Right-click on helloworld.c, select **Rename…**, and rename the file to led_switch.c.

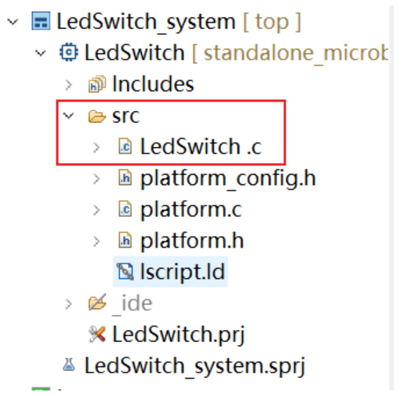


Figure 4.

## 5. Include Required Header Files

- Open led_switch.c.
- Remove the following two lines of code:

  print("Hello World\n\r");

  print("Successfully ran Hello World application");
- As shown in Figure 5, add the following #include directives at the top of the file:

```
#include "stdio.h"      // For standard I/O functions (printf, scanf, etc.)
#include "platform.h"   // For platform initialization and cleanup functions
#include "xgpio.h"      // For GPIO driver functions and XGpio structure definitions
```

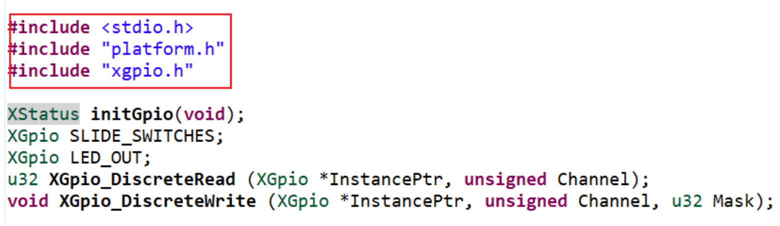
XStatus initGpio(void); // Function: Initialize GPIO

XGpio SLIDE_SWITCHES; // Variable: GPIO object for slide switches

XGpio LED_OUT; // Variable: GPIO object for LEDs

XGpio_DiscreteRead() - Function: Reads data from the specified GPIO channel

XGpio_DiscreteWrite() - Function: Writes data to the specified GPIO channel



```
#include <stdio.h>
#include "platform.h"
#include "xgpio.h"

XStatus initGpio(void);
XGpio SLIDE_SWITCHES;
XGpio LED_OUT;
u32 XGpio_DiscreteRead (XGpio *InstancePtr, unsigned Channel);
void XGpio_DiscreteWrite (XGpio *InstancePtr, unsigned Channel, u32 Mask);
```

Figure 5

xgpio.h defines the XGpio struct and the function prototypes for GPIO operations.

**Reference:**

General-Purpose Input/Output https://en.wikipedia.org/wiki/General-purpose_input/output

MicroBlaze with Basys3 GPIO using Vivado/Vitis https://sites.google.com/a/umn.edu/mxp-fpga/home/vivado-notes/xilinx-vivado16-2-and-embedded-processing-using-microblaze-and-basys3/xilinx-vivado16-2-and-embedded-processing-using-microblaze-with-basys3-gpio

## 6. Obtain Device IDs

- In **Project Explorer**, locate the **LedSwitch_bsp** folder.
- Expand top → microblaze_0 → standalone_microblaze_0→ bsp → microblaze_0→ include.
- Open xparameters.h.
- As shown in Figure 6, Search for the LED definition (typically named XPAR_GPIO_LEDS_DEVICE_ID).

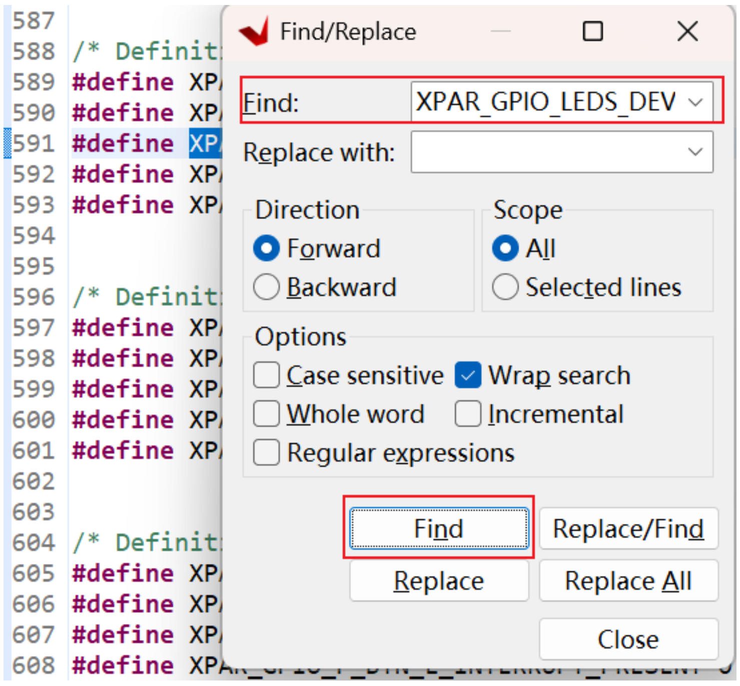
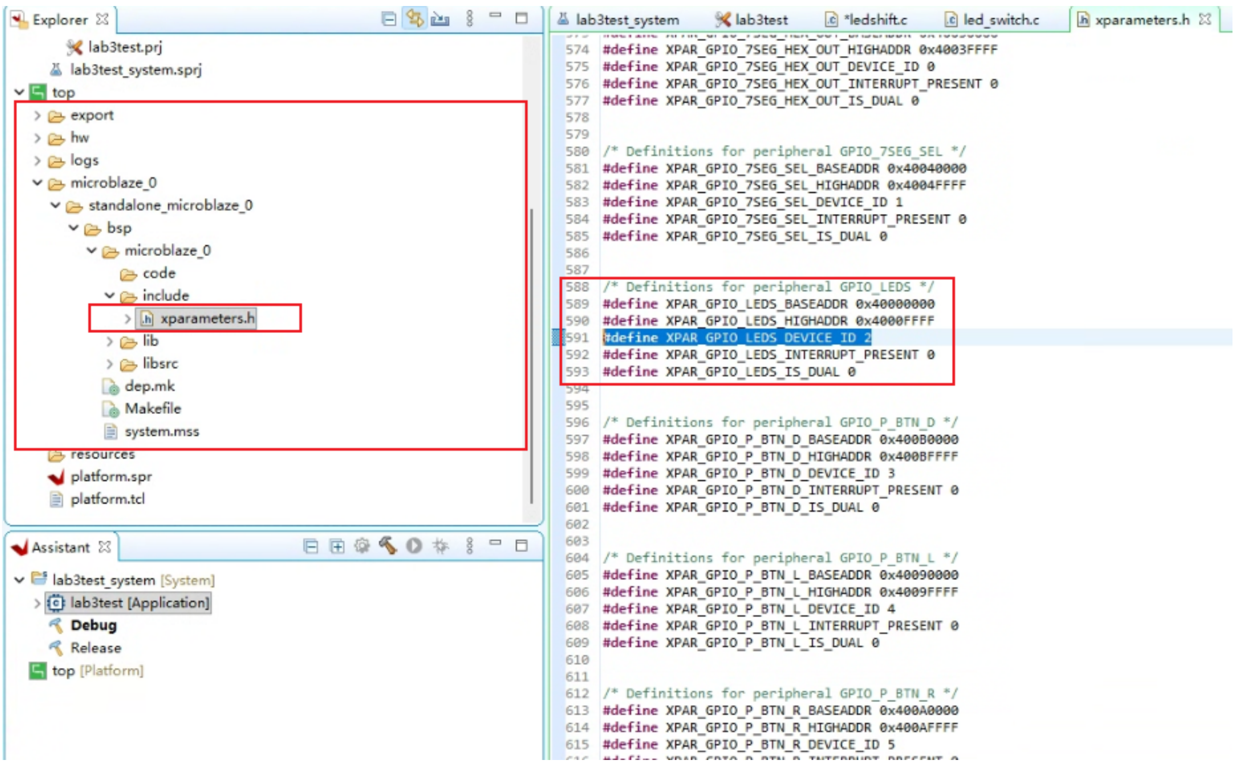


Figure 6.

- Confirm XPAR_GPIO_LEDS_DEVICE_ID device ID Number, As shown in Figure 7, the ID Number is 2 (**#define** XPAR_GPIO_LEDS_DEVICE_ID 2).



```
#define XPAR_GPIO_7SEG_HEX_OUT_HIGHADDR 0x4003FFFF
#define XPAR_GPIO_7SEG_HEX_OUT_DEVICE_ID 0
#define XPAR_GPIO_7SEG_HEX_OUT_INTERRUPT_PRESENT 0
#define XPAR_GPIO_7SEG_HEX_OUT_IS_DUAL 0


/* Definitions for peripheral GPIO_7SEG_SEL */
#define XPAR_GPIO_7SEG_SEL_BASEADDR 0x40040000
#define XPAR_GPIO_7SEG_SEL_HIGHADDR 0x4004FFFF
#define XPAR_GPIO_7SEG_SEL_DEVICE_ID 1
#define XPAR_GPIO_7SEG_SEL_INTERRUPT_PRESENT 0
#define XPAR_GPIO_7SEG_SEL_IS_DUAL 0


/* Definitions for peripheral GPIO_LEDS */
#define XPAR_GPIO_LEDS_BASEADDR 0x40000000
#define XPAR_GPIO_LEDS_HIGHADDR 0x4000FFFF
#define XPAR_GPIO_LEDS_DEVICE_ID 2
#define XPAR_GPIO_LEDS_INTERRUPT_PRESENT 0
#define XPAR_GPIO_LEDS_IS_DUAL 0


/* Definitions for peripheral GPIO_P_BTN_D */
#define XPAR_GPIO_P_BTN_D_BASEADDR 0x400B0000
#define XPAR_GPIO_P_BTN_D_HIGHADDR 0x400BFFFF
#define XPAR_GPIO_P_BTN_D_DEVICE_ID 3
#define XPAR_GPIO_P_BTN_D_INTERRUPT_PRESENT 0
#define XPAR_GPIO_P_BTN_D_IS_DUAL 0


/* Definitions for peripheral GPIO_P_BTN_L */
#define XPAR_GPIO_P_BTN_L_BASEADDR 0x40090000
#define XPAR_GPIO_P_BTN_L_HIGHADDR 0x4009FFFF
#define XPAR_GPIO_P_BTN_L_DEVICE_ID 4
#define XPAR_GPIO_P_BTN_L_INTERRUPT_PRESENT 0
#define XPAR_GPIO_P_BTN_L_IS_DUAL 0


/* Definitions for peripheral GPIO_P_BTN_R */
#define XPAR_GPIO_P_BTN_R_BASEADDR 0x400A0000
#define XPAR_GPIO_P_BTN_R_HIGHADDR 0x400AFFFF
#define XPAR_GPIO_P_BTN_R_DEVICE_ID 5
```

Figure 7.

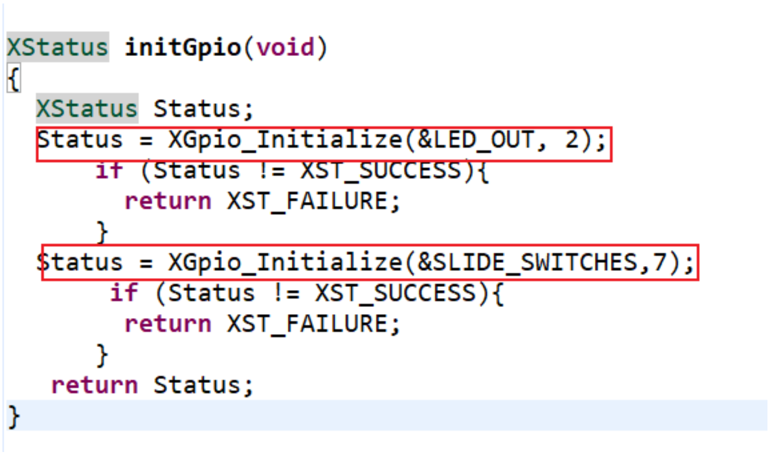
## 7. Create the initGpio() Function

- Define a **Function** initGpio() in **global scope** (not inside main()).

- As shown in Figure 8, Declare an XGpio instance for each device you intend to access (e.g., LEDs and switches).

  XGpio_Initialize(&LED_OUT, 2)

  XGpio_Initialize(&SLIDE_SWITCHES,7)



```
XStatus initGpio(void)
{
  XStatus Status;
  Status = XGpio_Initialize(&LED_OUT, 2);
      if (Status != XST_SUCCESS){
        return XST_FAILURE;
      }
  Status = XGpio_Initialize(&SLIDE_SWITCHES,7);
       if (Status != XST_SUCCESS){
        return XST_FAILURE;
      }
   return Status;
}
```

Figure 8.

- If the return value is not XST_SUCCESS, **return** XST_FAILURE to terminate the program.

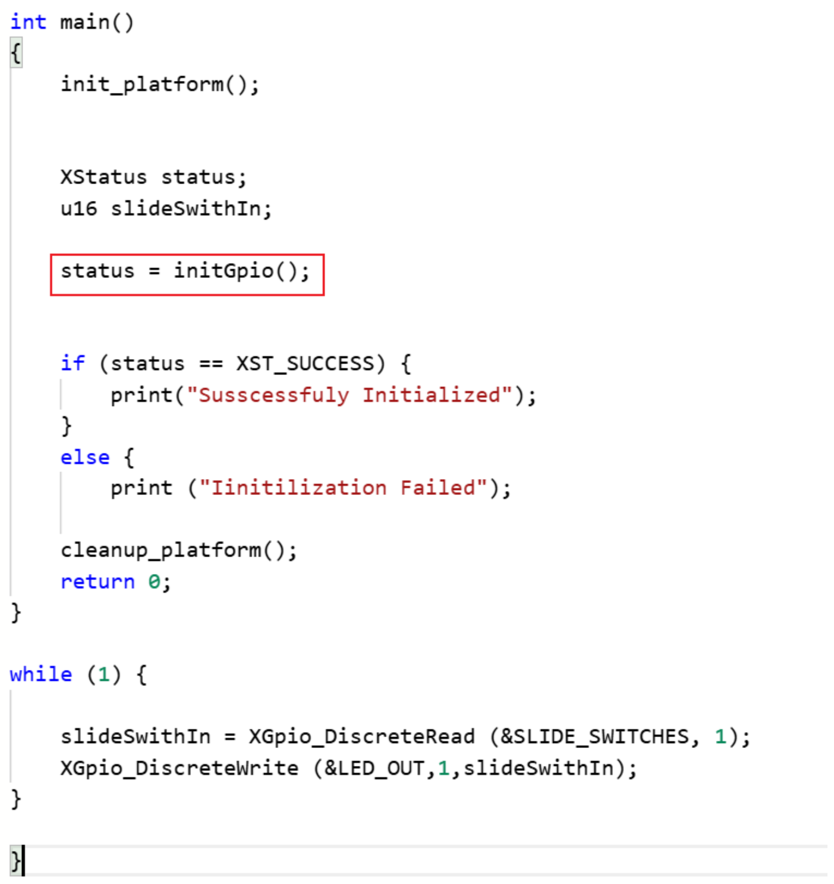
  **if** (Status != XST_SUCCESS){

  **return** XST_FAILURE;

  }

**8. Modify main() Function**

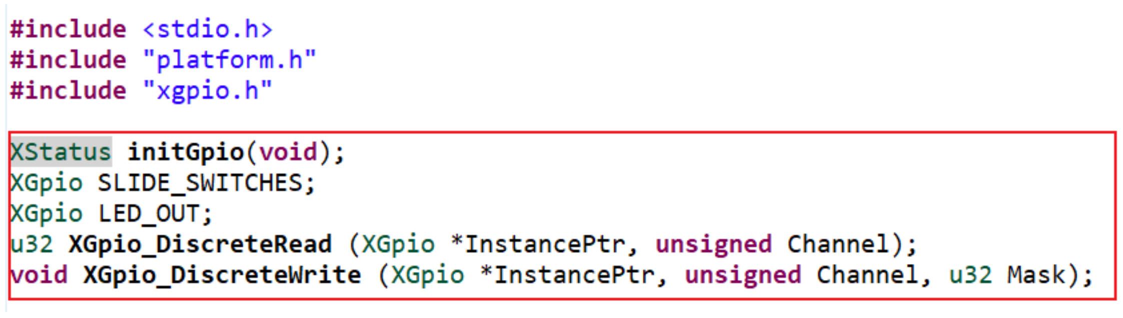
- As shown on Figure 9, Inside main(), ensure all user code is between init_platform(); and cleanup_platform();.
- Call initGpio() before performing any GPIO operations.



```
int main()
{
    init_platform();


    XStatus status;
    u16 slideSwithIn;

    status = initGpio();


    if (status == XST_SUCCESS) {
        print("Susscessfuly Initialized");
    }
    else {
        print ("Iinitilization Failed");

    cleanup_platform();
    return 0;
}

while (1) {

    slideSwithIn = XGpio_DiscreteRead (&SLIDE_SWITCHES, 1);
    XGpio_DiscreteWrite (&LED_OUT,1,slideSwithIn);
}

}
```

Figure 9.

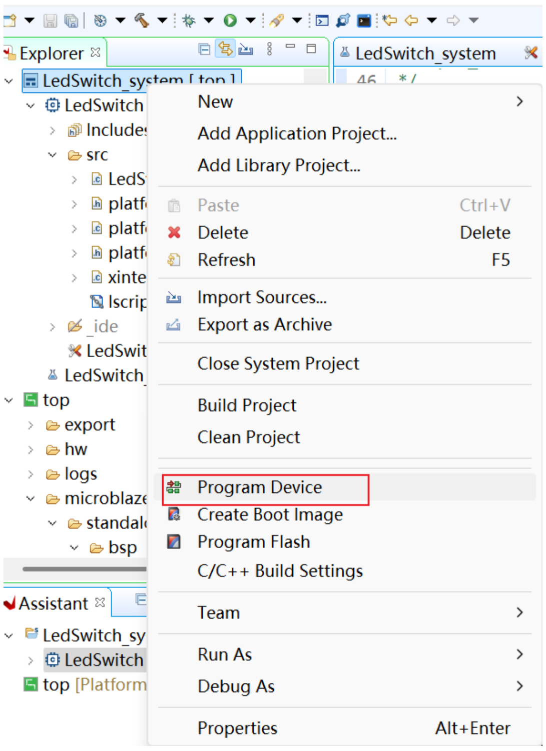
As shown in Figure 10, the code defines XStatus initGpio(void), a function that initializes GPIO devices and returns a status code (XStatus) indicating success or failure. It also declares two XGpio instances: SLIDE_SWITCHES for reading slide switch inputs and LED_OUT for controlling LED outputs. The function XGpio_DiscreteRead() reads the current state of a specified GPIO channel, returning 32-bit data where each bit represents a pin's high/low state. Meanwhile, XGpio_DiscreteWrite() writes 32-bit mask data to a specified channel to control output pin states. A typical application initializes the GPIO using initGpio(), then uses XGpio_DiscreteRead() to read switch positions and XGpio_DiscreteWrite() to control corresponding LEDs, implementing switch-controlled LED functionality.



```
#include <stdio.h>
#include "platform.h"
#include "xgpio.h"

XStatus initGpio(void);
XGpio SLIDE_SWITCHES;
XGpio LED_OUT;
u32 XGpio_DiscreteRead (XGpio *InstancePtr, unsigned Channel);
void XGpio_DiscreteWrite (XGpio *InstancePtr, unsigned Channel, u32 Mask);
```

Figure 10

9. Once you have completed writing the **LedSwitch.c** code, Vitis IDE will display the complete development environment. The project structure shows both the system project and application project. First, you need to program the FPGA with the hardware design by right-clicking on the **LedSwitch** project and selecting **Program Device** from the context menu (Figure 11a). This step configures the FPGA with the hardware bitstream containing the MicroBlaze processor and other peripherals. Next, to build the application, right-click on the **LedSwitch** project and select **Build Project** from the context menu (Figure 11b). This compiles the C source code into an executable file (.elf) that can run on the MicroBlaze processor.



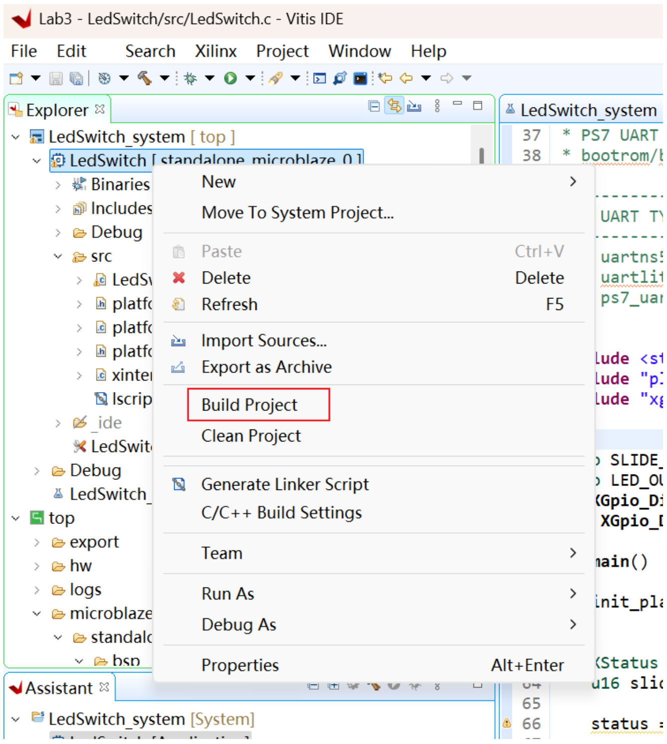
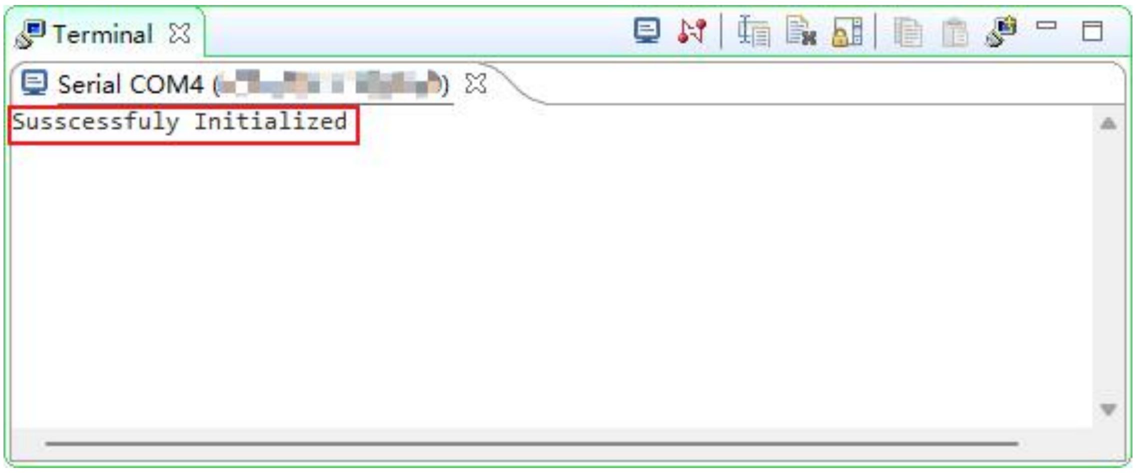


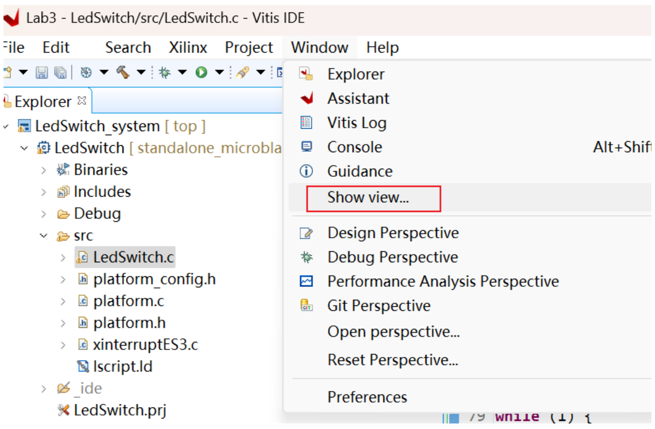
Figure 11(a) Figure 11(b)

10. **Figure 12** shows what the Console should look like when the application is successfully run, displaying "Successfully Initialized" if GPIO initialization succeeds. Toggle the slide switches to control the corresponding LEDs in real-time. To view the output and run the application, follow **Section 4 Steps 5-8** of Lab 1 instructions (also illustrated in **Figure 13(a)-(d)**): open the Terminal view, configure the Serial Terminal settings, and set up the run/debug configuration. If the activity on Ledswitch operations has been successfully completed, proceed to **Section 2 LED Shifting**.



```
Susscessfuly Initialized
```

Figure 12



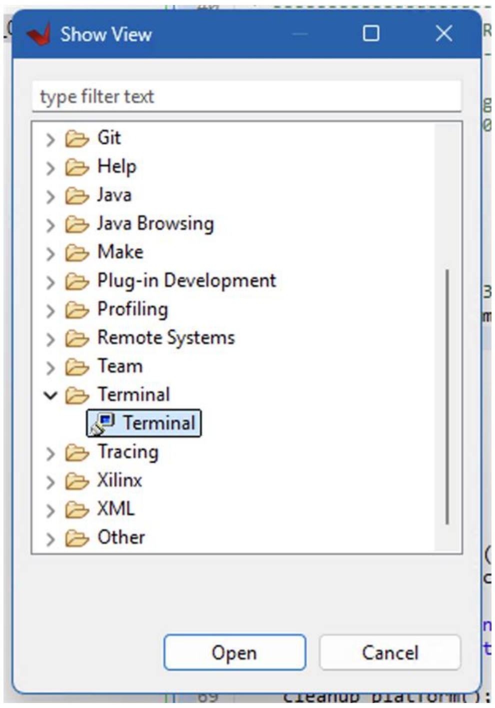
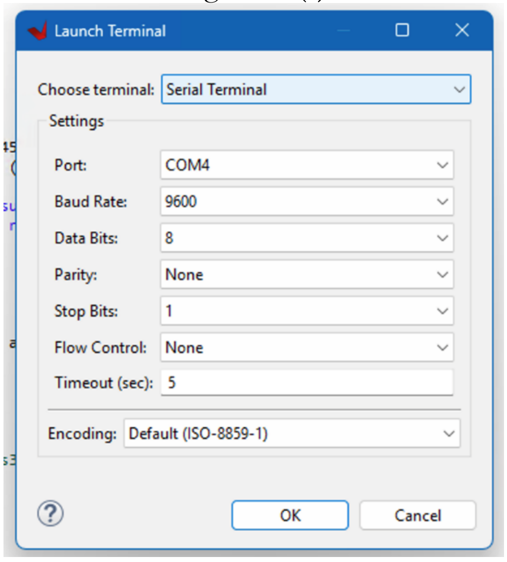


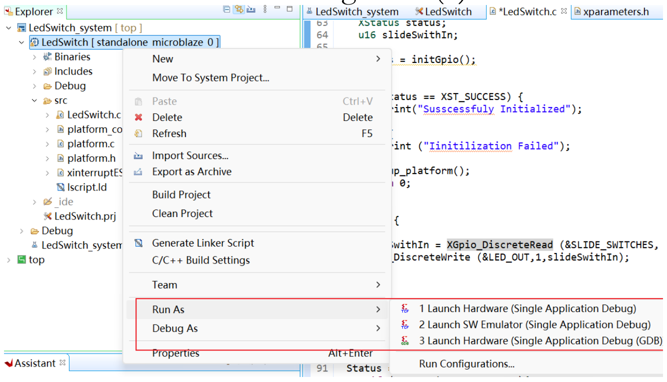
Figure 13(a) Figure 13(b)



Choose terminal: Serial Terminal

Settings

| | |
|---|---|
| Port: | COM4 |
| Baud Rate: | 9600 |
| Data Bits: | 8 |
| Parity: | None |
| Stop Bits: | 1 |
| Flow Control: | None |
| Timeout (sec): | 5 |
| Encoding: | Default (ISO-8859-1) |



```
    XStatus status;
    u16 slideSwithIn;

  = initGpio();

 atus == XST_SUCCESS) {
 int("Susscessfuly Initialized");

 int ("Iinitilization Failed");

 p_platform();
  0;

 {

 withIn = XGpio_DiscreteRead (&SLIDE_SWITCHES,
 DiscreteWrite (&LED_OUT,1,slideSwithIn);

  Status
```

Figure 13(c) Figure 13(d)

If you run the application successfully, the Console will display "Successfully Initialized" as shown in Figure 12, and you can toggle the slide switches to control the corresponding LEDs.

## II. LED Shifting

### Objectives

- Create an application to shift LED positions
- Implement interrupt-based execution
- Learn and understand the role and necessity of interrupts in FPGA-based systems (BASYS 3 + MicroBlaze).

### Activity Summary

1. Create a new application project called LedShift
2. Write the LED shifting code
3. Program the FPGA, setup Run Configuration, and run the application
4. Repeat activity 3 for changes made to the application source file

### Specifications

The LED Shifting application creates a single LED that moves from right to left across the 16-bit LED display with automatic rotation. The functionality is as follows:

**Initial State:** Start with LED pattern 0000000000000001 (rightmost LED ON)
**Shift Direction:** Left shift with zero padding on the right
**Shift Speed:** One position shift every 0.1 seconds (controlled by hardware timer interrupt)
**Rotation:** When the LED reaches the leftmost position, it wraps around to the rightmost position

**Example Sequence:**
0000000000000001 → (0.1s) →
0000000000000010 → (0.1s) →
0000000000000100 → (0.1s) →
0000000000001000 → (0.1s) →
... continuing left ...
1000000000000000 → (0.1s) →
0000000000000001 → (repeats cycle)

### Activities

The activities here involve shifting the position of the end LED from the first LED on the Basys3 board. Follow the steps below:

1. As shown in Figure 14, Create a new application with the name **LedShift** and rename **helloworld.c** to **led_shift.c**.

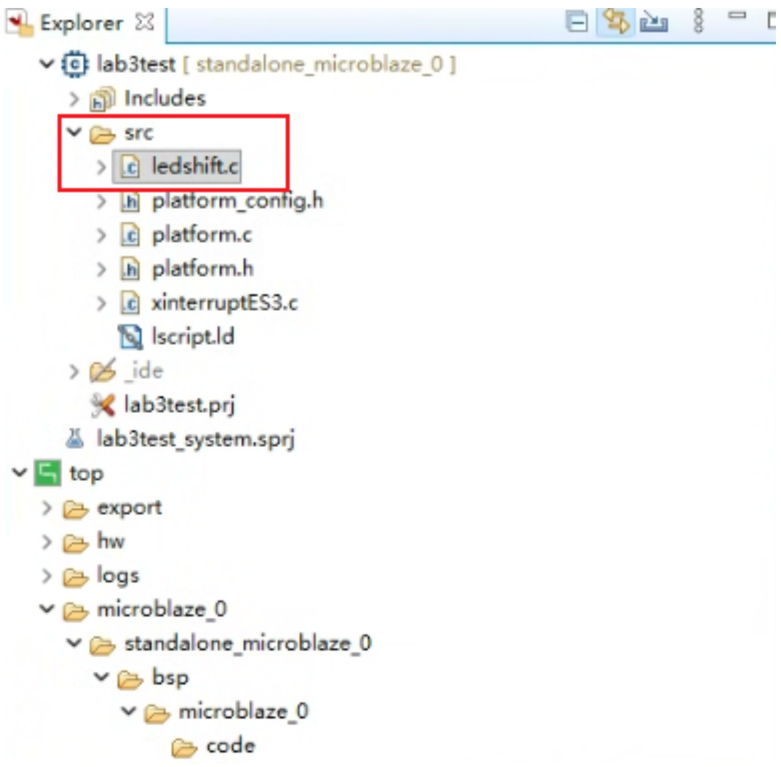
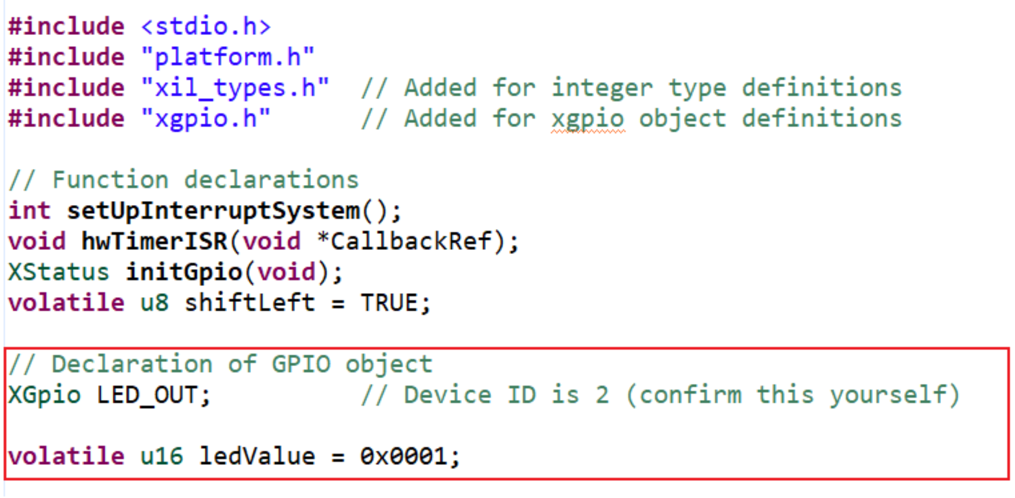


Figure 14

2. As shown in Figure 15, open the led_shift.c file and declare an unsigned 16-bit integer (`ledValue`) and load it with the value 0x0001. Remember to declare `ledValue` as a **`volatile`**



```
#include <stdio.h>
#include "platform.h"
#include "xil_types.h"  // Added for integer type definitions
#include "xgpio.h"      // Added for xgpio object definitions

// Function declarations
int setUpInterruptSystem();
void hwTimerISR(void *CallbackRef);
XStatus initGpio(void);
volatile u8 shiftLeft = TRUE;

// Declaration of GPIO object
XGpio LED_OUT;          // Device ID is 2 (confirm this yourself)

volatile u16 ledValue = 0x0001;
```

Figure 15

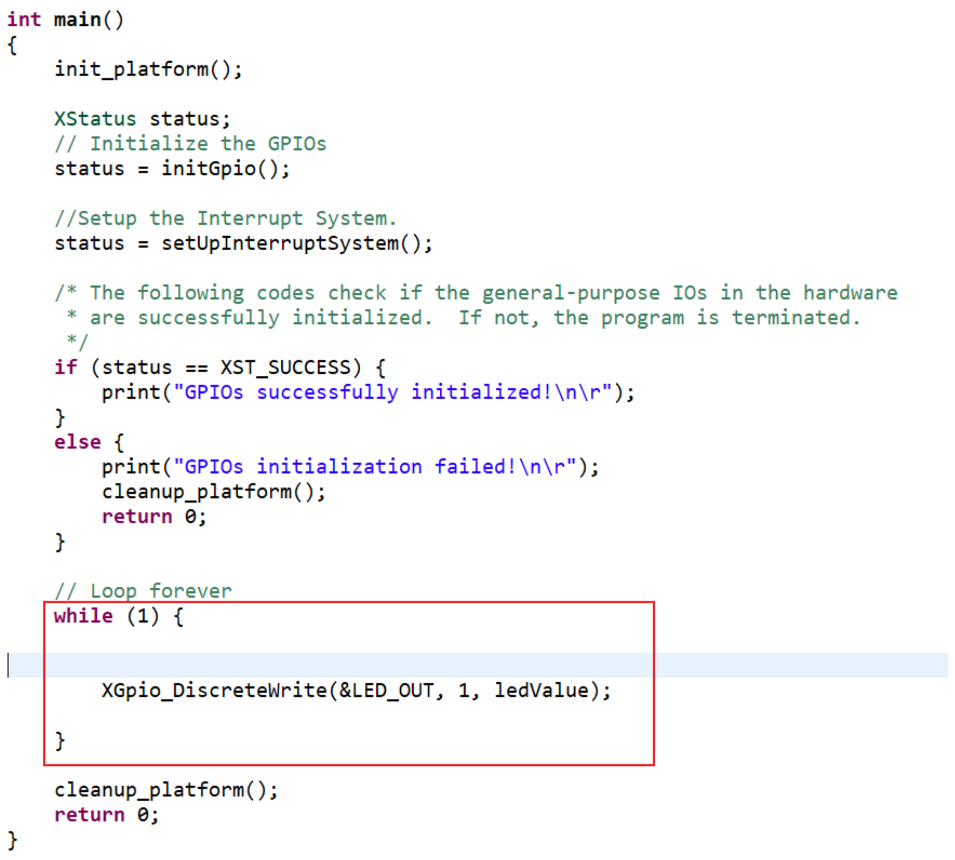
// Declaration of GPIO object

XGpio LED_OUT;       // Device ID is 2 (confirm this yourself)

volatile u16 ledValue = 0x0001;  // Start with rightmost LED ON, will shift left progressively(binary: 0000000000000001)

3. Initialize the GPIOs, check if the general-purpose IOs in the hardware are successfully initialized similar to Section 1 LedSwitch application.

4. As shown in Figure 16, in the `main()`, output the content of the `ledValue` variable to the LED port.



```
int main()
{
    init_platform();

    XStatus status;
    // Initialize the GPIOs
    status = initGpio();

    //Setup the Interrupt System.
    status = setUpInterruptSystem();

    /* The following codes check if the general-purpose IOs in the hardware
     * are successfully initialized.  If not, the program is terminated.
     */
    if (status == XST_SUCCESS) {
        print("GPIOs successfully initialized!\n\r");
    }
    else {
        print("GPIOs initialization failed!\n\r");
        cleanup_platform();
        return 0;
    }

    // Loop forever
    while (1) {


        XGpio_DiscreteWrite(&LED_OUT, 1, ledValue);

    }

    cleanup_platform();
    return 0;
}
```

Figure 16

5. As shown in Figure 17, copy the **xinterruptES3.c** file to the **src** folder. As shown in Figure 18, write an **hwTimerISR**, which rotates the content of ledValue to the left every interrupt event using the shift instruction. **Note that the hardware timer gives an interrupt every 0.1 second.**

**Why Use Interrupts for LED Shifting?**

Interrupts allow the system to perform LED shifting at precise time intervals without blocking the main program. The hardware timer generates an interrupt every 0.1 seconds, triggering the Interrupt Service Routine (ISR) hwTimerISR(). This approach enables:

Precise timing: Hardware timer ensures accurate 0.1-second intervals

Non-blocking operation: Main program can perform other tasks while waiting

Efficient resource usage: CPU is not constantly checking the timer

**How Interrupt-Based Shifting Works:**

Timer Interrupt: Hardware timer triggers every 0.1 seconds

ISR Execution: hwTimerISR() decrements the counter

Counter Check: Main loop checks if counter reaches zero

LED Rotation: When counter = 0, LED pattern shifts left and counter resets

Repeat: Process continues indefinitely

**Reference:** What is ISR? https://embeddedwala.com/EmbeddedSystems/embedded-c/what-is-isr

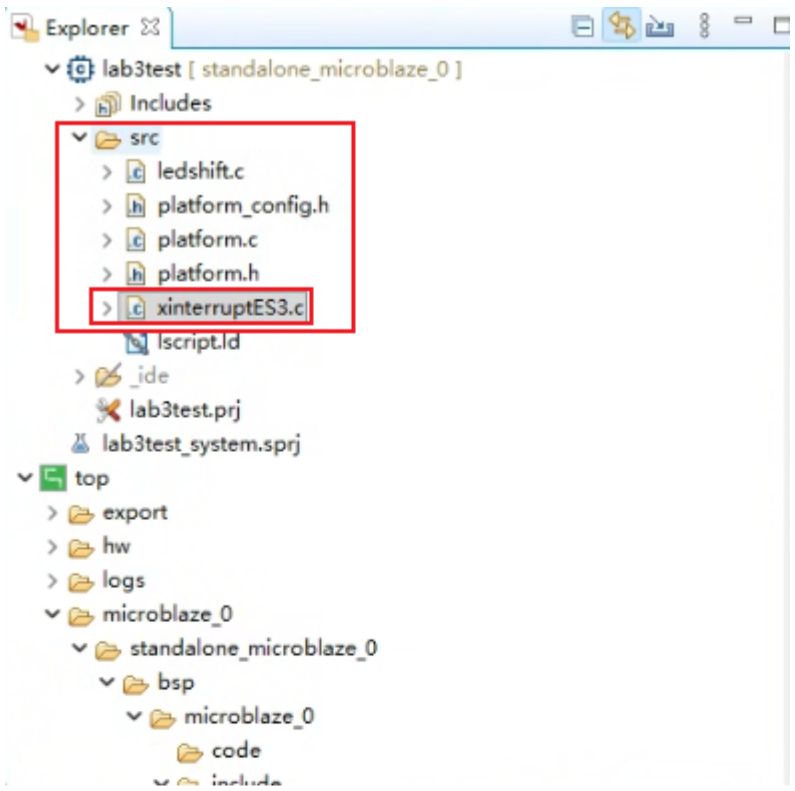
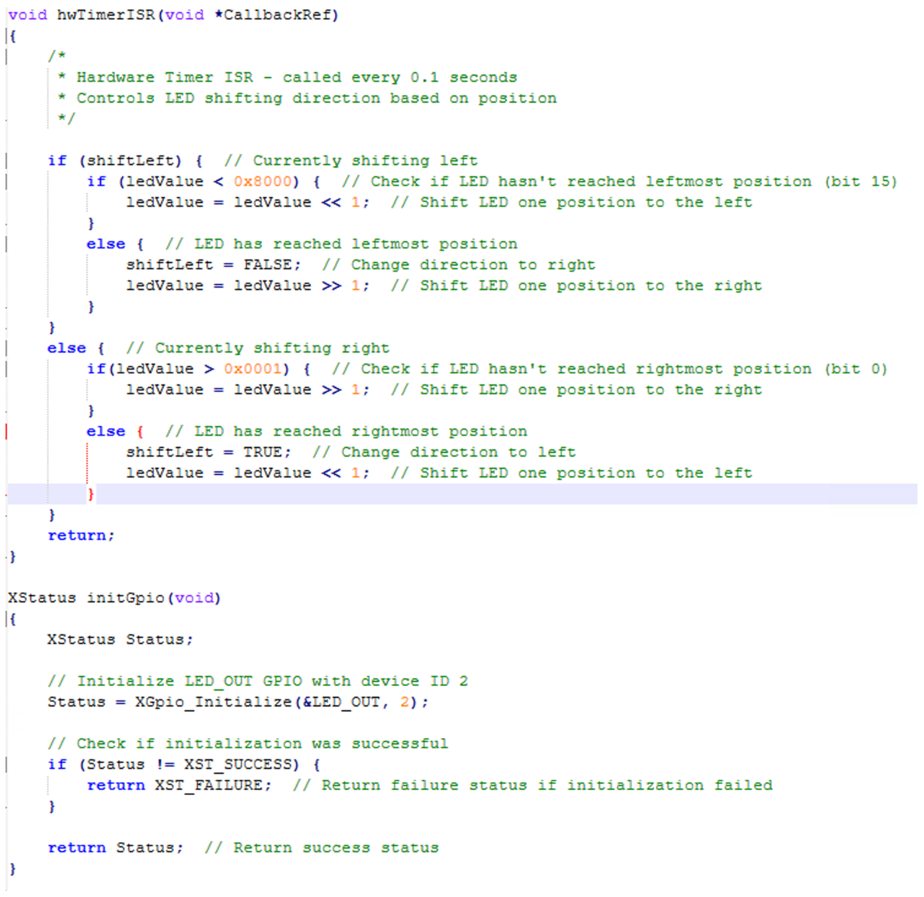


Figure 17.



```
void hwTimerISR(void *CallbackRef)
{
    /*
     * Hardware Timer ISR - called every 0.1 seconds
     * Controls LED shifting direction based on position
     */

    if (shiftLeft) {  // Currently shifting left
        if (ledValue < 0x8000) {  // Check if LED hasn't reached leftmost position (bit 15)
            ledValue = ledValue << 1;  // Shift LED one position to the left
        }
        else {  // LED has reached leftmost position
            shiftLeft = FALSE;  // Change direction to right
            ledValue = ledValue >> 1;  // Shift LED one position to the right
        }
    }
    else {  // Currently shifting right
        if(ledValue > 0x0001) {  // Check if LED hasn't reached rightmost position (bit 0)
            ledValue = ledValue >> 1;  // Shift LED one position to the right
        }
        else {  // LED has reached rightmost position
            shiftLeft = TRUE;  // Change direction to left
            ledValue = ledValue << 1;  // Shift LED one position to the left
        }
    }
    return;
}

XStatus initGpio(void)
{
    XStatus Status;

    // Initialize LED_OUT GPIO with device ID 2
    Status = XGpio_Initialize(&LED_OUT, 2);

    // Check if initialization was successful
    if (Status != XST_SUCCESS) {
        return XST_FAILURE;  // Return failure status if initialization failed
    }

    return Status;  // Return success status
}
```

Figure 18

6. Test the LED shifting on the Basys3 board by running the application. The light should rotate every 0.1 second across the LEDs from right to left.

## III. LED Advanced Shifting

### Objectives

- Create an application with variable LED shifting and speed changes.

### Activity Summary

1. Create a new application named **LedAdvancedShift**
2. Implement advanced LED shifting logic with varying speed
3. Program the FPGA, setup Run Configuration, and run the application
4. Repeat activity 3 for changes made to the application source file

### Specifications

The LED Advanced Shifting application implements a progressively accelerating LED chasing pattern that automatically resets. The functionality is as follows:

**Initial State:** Start with LED pattern 0000000000000001 (rightmost LED ON)
**Shift Direction:** Left circular shift (same as Section II)
**Variable Speed Control:** LED shifting speed progressively increases, then resets
**Speed Progression:**
Stage 1: One shift every 2.7 seconds (counter = 27, timer interrupt every 0.1s)
Stage 2: One shift every 0.9 seconds (counter = 9)
Stage 3: One shift every 0.3 seconds (counter = 3)
Stage 4: One shift every 0.1 seconds (counter = 1)
Reset: After Stage 4, speed resets to Stage 1 and cycle repeats

**Example Behavior Timeline:**

```
  Time 0.0s:  0000000000000001  (wait 2.7s, Stage 1)
  Time 2.7s:  0000000000000010  (wait 0.9s, Stage 2)
  Time 3.6s:  0000000000000100  (wait 0.3s, Stage 3)
  Time 3.9s:  0000000000001000  (wait 0.1s, Stage 4)
  Time 4.0s:  0000000000010000  (wait 2.7s, Reset to Stage 1)
  Time 6.7s:  0000000000100000  (wait 0.9s, Stage 2)
  Time 7.6s:  0000000001000000  (wait 0.3s, Stage 3)
  ... continues with repeating 4-stage speed cycle ...
```

**Visual Effect:** Creates a dynamic chasing light that accelerates progressively, then suddenly slows down to create a periodic pulsing acceleration pattern

### Activities

1. As shown in Figure 19, in order to control the speed of led shifting, a **rotate function** has been written for you (see **rotate.c** in the Lab3 – Codes folder). There are two functions in this file. Use the rotateLeft() function. Try and see if you can explain how these functions work.

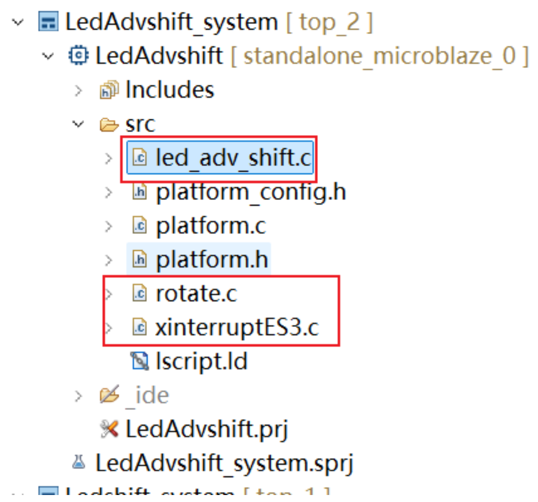
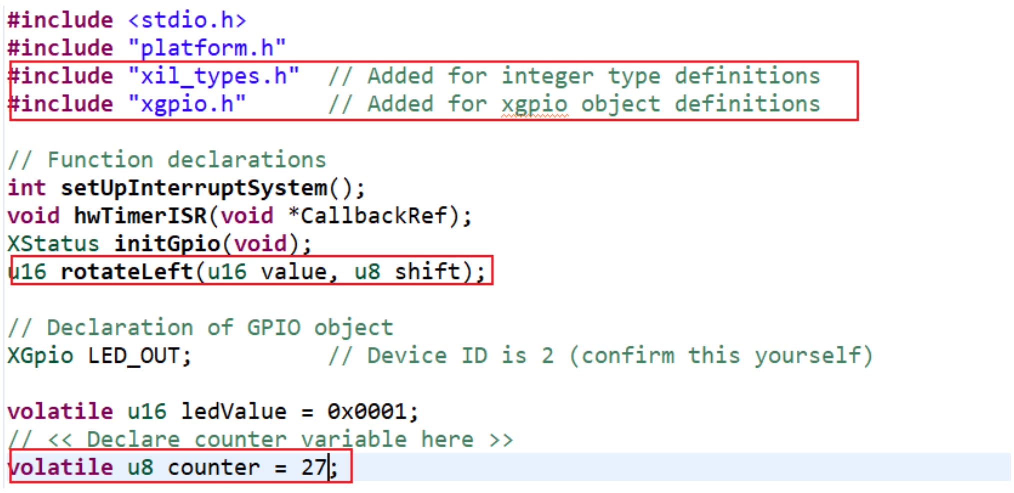


Figure 19.

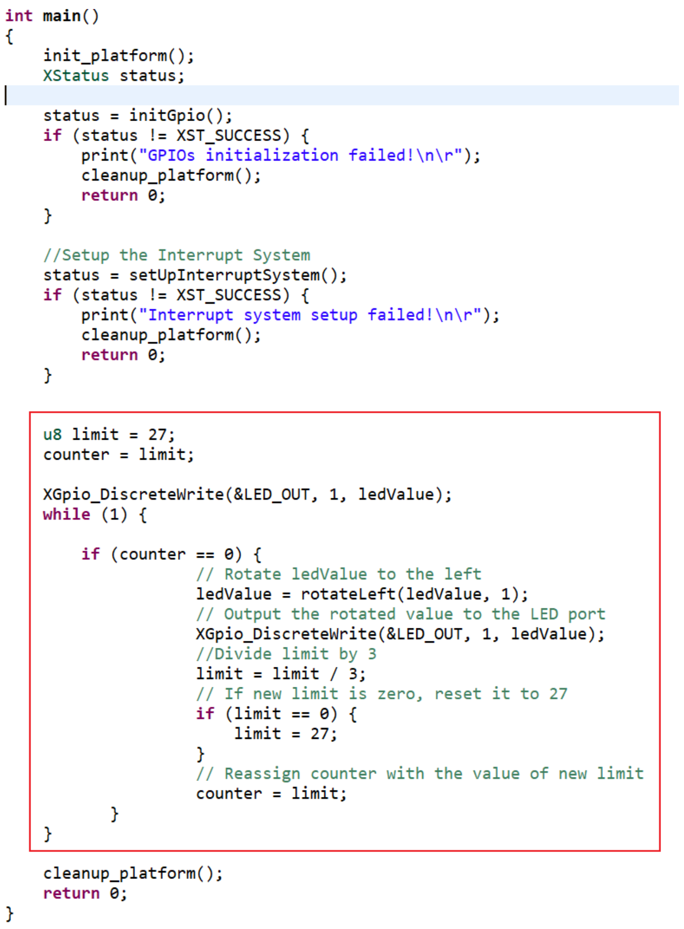
2. As shown in Figure 20 the rotateLeft function declaration takes a 16-bit unsigned integer value and an 8-bit shift parameter, performing left circular bit shifting to create LED pattern effects like chasing or flowing lights. The volatile counter variable is initialized to 27.



```c
#include <stdio.h>
#include "platform.h"
#include "xil_types.h"  // Added for integer type definitions
#include "xgpio.h"      // Added for xgpio object definitions

// Function declarations
int setUpInterruptSystem();
void hwTimerISR(void *CallbackRef);
XStatus initGpio(void);
u16 rotateLeft(u16 value, u8 shift);

// Declaration of GPIO object
XGpio LED_OUT;          // Device ID is 2 (confirm this yourself)

volatile u16 ledValue = 0x0001;
// << Declare counter variable here >>
volatile u8 counter = 27;
```

Figure 20

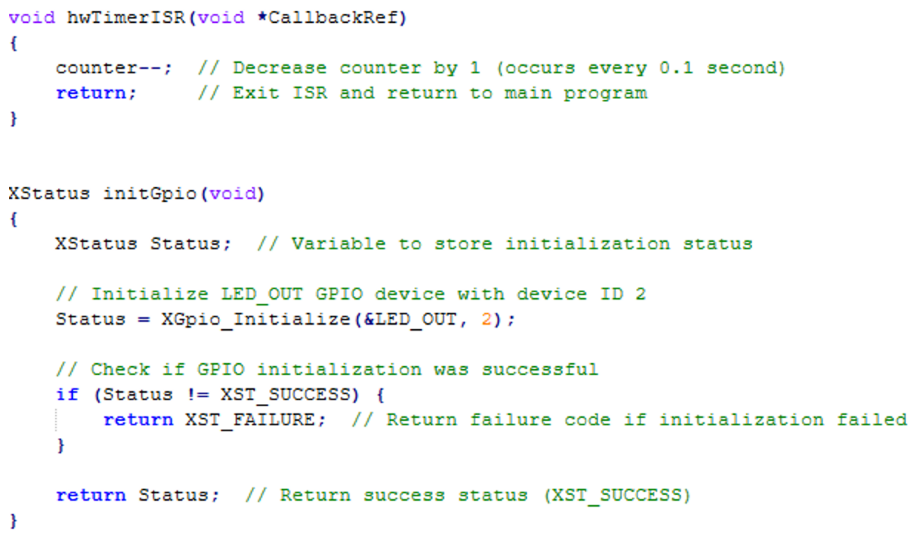
3. As shown in Figure 21, repeat this code creates a LED chasing light effect. It sets a timer limit to 27 and initializes a counter. When the counter reaches zero, it rotates the LED value left by one position using rotateLeft function, outputs it to GPIO, then speeds up the rotation by dividing the limit by 3. When the limit becomes zero, it resets to 27 to restart the cycle, creating a progressively faster LED chasing effect that periodically resets to the original speed.



```c
int main()
{
    init_platform();
    XStatus status;

    status = initGpio();
    if (status != XST_SUCCESS) {
        print("GPIOs initialization failed!\n\r");
        cleanup_platform();
        return 0;
    }

    //Setup the Interrupt System
    status = setUpInterruptSystem();
    if (status != XST_SUCCESS) {
        print("Interrupt system setup failed!\n\r");
        cleanup_platform();
        return 0;
    }

    u8 limit = 27;
    counter = limit;

    XGpio_DiscreteWrite(&LED_OUT, 1, ledValue);
    while (1) {

        if (counter == 0) {
                    // Rotate ledValue to the left
                    ledValue = rotateLeft(ledValue, 1);
                    // Output the rotated value to the LED port
                    XGpio_DiscreteWrite(&LED_OUT, 1, ledValue);
                    //Divide limit by 3
                    limit = limit / 3;
                    // If new limit is zero, reset it to 27
                    if (limit == 0) {
                        limit = 27;
                    }
                    // Reassign counter with the value of new limit
                    counter = limit;
            }
    }

    cleanup_platform();
    return 0;
}
```

Figure 21

4. As shown in Figure 21, the hwTimerISR function needs to be modified to be counter-controlled.



```
void hwTimerISR(void *CallbackRef)
{
    counter--;  // Decrease counter by 1 (occurs every 0.1 second)
    return;     // Exit ISR and return to main program
}


XStatus initGpio(void)
{
    XStatus Status;  // Variable to store initialization status

    // Initialize LED_OUT GPIO device with device ID 2
    Status = XGpio_Initialize(&LED_OUT, 2);

    // Check if GPIO initialization was successful
    if (Status != XST_SUCCESS) {
        return XST_FAILURE;  // Return failure code if initialization failed
    }

    return Status;  // Return success status (XST_SUCCESS)
}
```

Figure 21

5. This code implements a counter-controlled LED shifting system with the following logic:
    a) **Counter Check**: When the counter reaches zero, execute the following operations
    b) **LED Rotation**: Rotate ledValue to the left and output the result to the LED port using the rotate function from the previous LedShift application
    c) **Speed Control**: Divide the limit value by 3 using a custom division subroutine to increase the shifting speed
    d) **Reset Mechanism**: If the new limit becomes zero, reset it to 27 (1B in hex) to prevent infinite acceleration
    e) **Counter Update**: Reassign the counter with the new limit value
    f) **Overall Effect**: This creates a dynamic LED chasing light that progressively accelerates with each cycle, automatically resetting to the original speed when reaching maximum velocity, producing a visually appealing periodic acceleration pattern.

6. Test the LedAdvshift, the LED rotating speed starts with one shift after 2.7 seconds, and increases every time an interrupt occurs (respectively to one shift after 0.9 second, one shift every 0.3 second, and one shift every 0.1 second). After 4 interrupts, the period is reset to its original value and the above process restarts again.

## References:

You can go to the lab3 reference in the learn section, or directly copy the link to your browser:

1. General-Purpose Input/Output https://en.wikipedia.org/wiki/General-purpose_input/output

2. MicroBlaze with Basys3 GPIO using Vivado/Vitis https://sites.google.com/a/umn.edu/mxp-fpga/home/vivado-notes/xilinx-vivado16-2-and-embedded-processing-using-microblaze-and-basys3/xilinx-vivado16-2-and-embedded-processing-using-microblaze-with-basys3-gpio

3. What is ISR? https://embeddedwala.com/EmbeddedSystems/embedded-c/what-is-isr