Laboratory 3 – I/O Operations – LED Applications

I. LED Switch

Objectives

- To build an application to control the 16-bit LEDs by 16-bit slide switches

Activity Summary

1. Launch Xilinx SDK after opening and exporting the hardware platform
2. Create the LedSwitch application
3. Program the FPGA (if neessary), setup Run Configuration, and run the application
4. Repeat activity 3 for changes made to the application source file

Activities

Create the LedSwitch application and a led_switch.c souce file based on the HelloWorld template. The first thing to do in order to access the switches and the LEDs is to initialize their port device drivers. To do this, we call the `XGpio_Initialize(XGpio *InstancePtr, u16 DeviceId)` function for each of the hardware device we intend to access. This initialization has to be done once. The initialization function returns an `int` and takes as parameters, an `XGpio` object pointing to the instance of the device, and the ID of the device. The `XGpio` is a struct defined in the `xgpio.h` header file for Xilinx GPIO device. As the device ID is assigned automatically by the build tool, you have to obtain it from the `xparameters.h` header file. As you should have noticed, each new application you create automatically has an accompanying **appName_bsp** in the Project Explorer window. If you expand this you can locate all the header files that can be used in the project. The header files are in the **include folder** under the **microblaze_0 folder**. Do this for the LedSwitch_bsp and open the `xparameters.h` file. You should find that the LEDs are named GPIO_LEDS with device ID 2 (confirm if this has not changed). For the LedSwitch application all the GPIO device initializations are grouped together in a function called `initGpio()`. Note that if the GPIOs are not successfully initialized, we terminate the program. However, remember to include the `xgpio.h` header file.

In order to read and write to the hardware I/Os, there are two functions that can be used on Xilinx SDK for the MicroBlaze processor. The DiscreteRead and DiscreteWrite functions (seeFigure 1) allow the reading and writing of values from and to the I/Os connected to the processor. The InstancePtr has been explained already. For all the devices connected to the processor, the channel number 1 is used. The read function returns a 32-bit unsigned integer while the write function writes a 32-bit unsigned integer using the `Mask` parameter of the function.

```
u32 XGpio_DiscreteRead(XGpio *InstancePtr, unsigned Channel);
void XGpio_DiscreteWrite(XGpio *InstancePtr, unsigned Channel, u32 Mask);
```

Figure 1

The following steps are suggested:

- Write the code in Figure 2 to the `initGpio()` function
- Write the code in Figure 3 to the `main()`
- Save the application and run it
- Slide any of the slide switches and the corresponding LED should light up

```c
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 2

```c
XStatus status;
u16 slideSwitchIn;

// Initialize the GPIOs
status = initGpio();

/* 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) {

    // Read the values on the 16 slide switches
    slideSwitchIn = XGpio_DiscreteRead(&SLIDE_SWITCHES, 1);
    // Write the values read from the slide switches to the LEDs
    XGpio_DiscreteWrite(&LED_OUT, 1, slideSwitchIn);

}
```

Figure 3

II. LED Shifting

Objectives

- To write an application to shift the LEDs
- To use the interrupt to execute instructions

Activity Summary

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

Activities

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

1. Create a LedShift application and led_shift.c source file
2. 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`**
3. Initialize the GPIOs, check if the general-purpose IOs in the hardware are successfully initialized similar to LedSwitch application.
4. In the `main()`, output the content of the `ledValue` variable to the LED port as shown in Figure 4.

```
XGpio_DiscreteWrite(&LED_OUT, 1, ledValue);
```

Figure 4

5. Copy the **xinterruptES3.c** file to the **src** folder. Write an ISR, 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. [**Hint:** Check Lab 2, Further Exercise 2 for shifting the values.]
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

   You probably have noticed that the shifting stops when the light gets to LD15. A more interesting thing to do is to have the light start again at LD0. To do this, 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 here. Try and see if you can explain how these functions work.
7. Repeat the previous steps, but this time, rotate the LEDs from left (LD15) to right (LD0)
8. Repeat steps 1 to 5, but first shifting from right to left, and then from left to right when the light gets to the end, and then from right to left, repeating this continuously
9. Repeat step 7, but this time, when the light gets to the last LED on the left or the right, blink it 10 times using the for loop before going in the other direction [**Hint**: To know when the light gets to the last LED, check the content of `ledValue`]

III. LED Advanced Shifting

Objectives

- To build an application to shift the LEDs with different shifting periods

Activity Summary

1. Create a new application (LedAdvancedShift)
2. Write the LED advanced shifting code
3. Program the FPGA (if necessary), setup Run Configuration, and run the application
4. Repeat activity 3 for changes made to the application source file

Activities

1. A template is provided under the filename “led_adv_shift.c” with placeholders for code you need to write

2. Name a variable `counter`, and another `limit`

3. Define `limit` with 27 (1B in hex)

4. Assign the value of `limit` to `counter`

5. Every time an interrupt occurs, decrement the **`counter`** value. [**Hint**: This needs to be done in the Interrupt method]

6. If `counter` = 0 then

   - Rotate `ledValue` to the left and output the result to the LED port [**Hin**t: Use rotate.c function from LedShift application]
   - Divide the value of `limit` by 3 using the division subroutine you developed in previous labs. If new `limit` = 0 then reset `limit` value to 27 (1B in hex)
   - Reassign `counter` with the value of the new `limit` variable value

7. Test the application, 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.