Lab 1: Interfacing to Input/Output Devices

Due date: February 23, 2024

Student Names: Parth Dadhania
Het Bharatkumar Patel

Course Section: B1

Lab Section: H31

Abstract:

In this lab, we were tasked with the development of a FreeRTOS application to interface with the Digilent Zybo Z7 board's input/output devices, using the Xilinx Vivado SDK on our host PCs for development and debugging purposes. The primary focus was on interfacing a keypad and a 7-segment LED display for Part 1, and extending functionalities to buttons, switches, and LEDs in Part 2. Utilizing the Xilinx SDK facilitated seamless development and debugging, allowing us to efficiently manage hardware configurations and FreeRTOS based applications. The exercises involved reading inputs from the keypad and displaying corresponding outputs on the 7-segment display, while also exploring the integration of additional peripherals to enhance the system's interactivity and functionality. The lab aimed to provide practical experience with real-time operating system tasks, queues, and synchronization mechanisms, as well as an in-depth understanding of hardware interfacing in embedded systems.

Design Section:

Part 1: Keypad and 7-Segment Display Interface:

In Part 1, our code facilitates the interaction between the keypad and the 7-segment display (SSD). We start by defining the SSD device ID using *#define SSD_DEVICE_ID XPAR_AXI_SSD_DEVICE_ID*, which identifies the SSD component within the Xilinx SDK environment. The SSD is then initialized and its GPIO direction set to output (*XGpio SSDInst*), preparing it for data display.

```
/*************************** Enter your code here ****************************/
    // TODO: Initialize SSD
    XGpio_Initialize(&SSDInst, SSD_DEVICE_ID);
    // TODO: Set SSD GPIO direction to output after initialization
    XGpio_SetDataDirection(&SSDInst, 1, 0x00);
/****************************************************************************/
```

The above code snippet initializes the Seven Segment Display (SSD) and sets its GPIO pins as outputs. *XGpio_Initialize* prepares the SSD with the configuration specified by *SSD_DEVICE_ID*, ensuring the device is ready for use. Following this, *XGpio_SetDataDirection* configures all GPIO pins connected to the SSD (specified by the channel number 1) to operate in output mode (0x00), enabling the SSD to display data. This setup is essential for the SSD's operational readiness to visually represent keypad inputs in the lab project.

Within the *keypadTask*, we introduce a delay constant *xDelay = 10 / portTICK_RATE_MS* to manage the refresh rate of the SSD, reducing flicker.

```
/*************************** Enter your code here ****************************/
      // TODO: Use the SSD_decode function to convert 'current_key' to its
      // binary representation and store it in 'ssd_value'.
      ssd_value = SSD_decode(current_key, 0);
```

```
      // TODO: Write 'ssd_value' to the right digit of the SSD.
      XGpio_DiscreteWrite(&SSDInst, 1, ssd_value);
      // TODO: Modify the above code to display 'ssd_value' on the left SSD digit
instead.

      // TODO: Implement alternating display logic.
      // Display 'current_key' on the right SSD, then pause. Next, display
      // 'previous_key' on the left SSD and pause again.
      // Experiment with delay timings to reduce flickering.
      ssd_value = SSD_decode(current_key, 1);
      XGpio_DiscreteWrite(&SSDInst, 1, ssd_value);
      vTaskDelay(xDelay);
      ssd_value = SSD_decode(previous_key, 0);
      XGpio_DiscreteWrite(&SSDInst, 1, ssd_value);
      vTaskDelay(xDelay);
      // TODO: Add a xil_printf statement to output the value of 'status'
      // whenever it changes.

/*****************************************************************************/
```

The above code segment effectively implements several key requirements for Lab 1, focusing on displaying keypad inputs on a Seven Segment Display (SSD). Initially, it uses *SSD_decode* to convert the *current_key* pressed on the keypad into a binary format suitable for the SSD, storing this value in *ssd_value*. This value is then displayed on the right digit of the SSD using *XGpio_DiscreteWrite*. To fulfill the lab's requirement of understanding SSD interfacing, we also needed to demonstrate how to alter the display from the right to the left digit of the SSD. Moreover, it introduces an alternating display logic that rapidly switches the display between the *current_key* and *previous_key*, achieving the effect of simultaneously showing both digits. This alternation, coupled with a delay (*vTaskDelay*), aims to minimize flickering, aligning with the lab's objective to experiment with display timings to find an optimal viewing experience.

To summarize, the code file *lab_1_part_1.c* establishes the integration of the keypad with the Seven Segment Display (SSD) on a Digilent Zybo Z7 board under a FreeRTOS environment, highlighting real-time input handling and output display. It involves initializing the keypad and SSD, setting SSD pins as outputs, and continuously monitoring for keypad inputs. Upon detecting a keypress, it converts the input to a binary format suitable for the SSD using the *SSD_decode* function, then employs an alternating display logic to visually represent current and previous keypresses on the SSD. This approach demonstrates essential embedded system concepts such as real-time task scheduling, peripheral interfacing, and dynamic output display, aligning with Lab 1's objectives of creating an interactive and responsive system.

Part 2: Expanded Peripheral Interface:

The code *greenLedsValue = XGpio_DiscreteRead(&swInst, SW_CHANNEL);* is part of the implementation for the **A5** keypad command in Lab 1 Part 2. It reads the current state of switches from the *swInst* GPIO instance and assigns this state directly to *greenLedsValue*. This assignment effectively maps each switch's on/off state to a corresponding green LED, providing a direct interaction where the position of each switch determines the on/off state of each LED. This functionality showcases a basic but crucial aspect of embedded systems, demonstrating how user inputs (through switches) can control outputs (LEDs) in real-time, allowing for a tangible understanding of digital input/output operations in hardware interfaces.

```
/*************************** Enter your code here ****************************/
                // TODO: Rotate 'greenLedsValue' left or right based on
                // 'message.action' ('L' or 'R'), and mask with 0xF.
                // Refer to shift logic in case 's' for guidance.
                if(message.action == 'R') {
                  greenLedsValue = (greenLedsValue << 1) | (greenLedsValue >> 3);
                } else if(message.action == 'L') {
                  greenLedsValue = (greenLedsValue >> 1) | (greenLedsValue << 3);
                }
                greenLedsValue &= 0xF;
/*****************************************************************************/
```

The code snippet above is associated with the **D4** command for the Lab 1 part 2 project, which involves rotating the state of green LEDs. When the 'R' action is detected, it signifies a right rotation, and the code shifts *greenLedsValue* to the left by one bit and moves the bit that fell off on the left to the rightmost position, effectively rotating the bits to the right. Conversely, when the 'L' action is detected, indicating a left rotation, the code shifts *greenLedsValue* to the right by one bit, moving the bit that fell off on the right to the leftmost position, thus rotating the bits to the left. After the rotation, *greenLedsValue* is masked with 0xF to ensure only the four least significant bits are retained, corresponding to the four green LEDs. This operation demonstrates a fundamental technique in embedded systems for creating cyclic patterns with LEDs, providing a visual representation of binary data manipulation.

```
/*************************** Enter your code here ****************************/
                // TODO: Update the RGB LED color depending on the value of
                // 'message.action'
                if (message.action == '+') {
                    RGBState.color += 1;
                } else if (message.action == '-') {
                    RGBState.color -= 1;
                }
/*****************************************************************************/
```

The above code segment is part of the **EC** command implementation for the Lab 1 Part 2 project, designed to adjust the RGB LED color. This snippet increments or decrements the

RGBState.color based on the *message.action* value, which is determined by user input. If the action is BTN3 ('+'), the color value increases, leading to a change in the RGB LED color towards the next in the sequence. Conversely, a BTN2 ('-') action decreases the color value, moving the color towards the previous in the sequence. This functionality allows users to interactively explore the RGB LED's color spectrum by pressing buttons to cycle through colors, enhancing understanding of how RGB LEDs can be programmed to display various colors in embedded systems.

```
static void HandleA3Command(Message* message)
{
/*************************** Enter your code here ****************************/
    // TODO:
    unsigned int buttonVal = 0;
        const TickType_t xDelay = 500 / portTICK_PERIOD_MS;

        u8 greenLedsValue = 1;

        while (1) {
            XGpio_DiscreteWrite(&greenLedsInst, LEDS_CHANNEL, greenLedsValue);

            greenLedsValue = greenLedsValue << 1;

            if (greenLedsValue > 8) {
                greenLedsValue = 1;
            }

            buttonVal = XGpio_DiscreteRead(&btnInst, BTN_CHANNEL);
            if (buttonVal & BTN1) {
                xil_printf("btn1 pressed, exiting A3 command handler\n");
                break;
            }

            vTaskDelay(xDelay);
        }
/******************************************************************************/
}
```

The code above implements the **A3** command from Lab 1 Part 2, which is designed to create a "chasing" LED pattern on the green LEDs. Initially, *greenLedsValue* is set to 1, lighting up the first LED. In each iteration of the loop, this value is shifted left, moving the lit LED one position to the right. When *greenLedsValue* shifts beyond the last LED, it is reset to 1, starting the pattern over. The loop continuously checks for BTN1 press to exit, providing a way to stop the LED chase. This functionality demonstrates bitwise operations and control flow, key concepts in embedded systems programming, allowing the user to interact with the hardware through button presses and visual feedback from the LEDs.

In summary, code file *lab_1_part_2.c* implements a comprehensive system for interfacing with a keypad and a seven-segment display (SSD), alongside managing RGB and green LEDs on a Xilinx FPGA platform using FreeRTOS. It begins with initializing various peripherals including the keypad, SSD, RGB LEDs, green LEDs, buttons, and switches. The main functionality is divided into several tasks: *keypadTask* for detecting key presses, *sevenSegTask* for displaying the pressed keys on the SSD, *commandTask* for executing specific commands based on the key combinations and button presses, *RGBLedTask* for controlling the RGB LED based on commands, and *GreenLedTask* for manipulating the green LEDs in response to keypad inputs and switch states. Communication between tasks is facilitated through queues, allowing for the exchange of key presses and command messages. The code structure encourages modularity and real-time responsiveness, adhering to the requirements of Lab 1 which focuses on understanding the integration and control of different hardware components through software in a real-time operating system environment.

Testing Suite:

Part 1:

Test Case	Description	Expected Result	Actual Result	Rationale
1	Pressing 'A' on the keypad	'A' appears on right SSD digit	'A' appears on right SSD digit	Ensures keypad 'A' key correctly maps to the right SSD digit
2	Pressing '8' on the keypad	'8' appears on right SSD digit while 'A' moves to left SSD digit	'8' appears on right SSD digit while 'A' moves to left SSD digit	Ensures the SSD display reads 'A8' ('A' on the left and '8' on the right) with non-perceived flickering

Part 2:

Test Case	Description	Expected Result	Actual Result	Rationale
E7	Pressing 'E7' on the keypad	Toggles RGB LED on/off depending on its last state	Toggles RGB LED on/off depending on its last state	Ensures toggling of RGB LED
EC	Pressing 'EC' on the keypad	RGB LED toggles state from one color to another depending on inputs	RGB LED toggles state from one color to another depending on inputs	Ensures RGB LED changes colors depending on the user input

EF	Pressing 'EF' on the keypad	Flashes the RGB LED at the set frequency (if greater than 0)	Flashes the RGB LED at the set frequency (if greater than 0)	Ensures that the RGB LED flashes at the set frequency
A5	Pressing 'A5' on the keypad	Turns on the Green LEDs depending on the values of the switches	Turns on the Green LEDs depending on the values of the switches	Ensures the turning on of the Green LEDs
D5	Pressing 'D5' on the keypad	Shifts value shown by the Green LEDs one bit to the left or right	Shifts value shown by the Green LEDs one bit to the left or right	Ensures that the Green LEDs shifts one bit to the left or right
D4	Pressing 'D4' on the keypad	Rotates value shown by the Green LEDs one bit to the left or right	Rotates value shown by the Green LEDs one bit to the left or right	Ensures that the Green LEDs rotates one bit to the left or right
A3	Pressing 'A3' on the keypad	Green LEDs light up sequentially, cycling from the first to the last, then repeat	Green LEDs light up sequentially, cycling from the first to the last, then repeat	Ensures the functionality to control green LEDs in a sequential pattern and checks the interrupt mechanism with a button (BTN1) press

Conclusion:

Throughout this lab, we meticulously met every outlined goal, effectively demonstrating our ability to integrate and manage various hardware components such as keypads, SSDs, and LEDs through a Xilinx FPGA board within a FreeRTOS framework. Our successful execution of the lab's tasks without encountering significant issues highlights our strong foundation in embedded systems and our adeptness in applying theoretical knowledge to practical scenarios. This experience has not only solidified our understanding of complex hardware-software interactions but also emphasized the critical role that thorough testing and debugging play in the seamless functionality of integrated systems.