ECSE 324 G02 Lab 3 Report

Adrian Wang 260769387
Cheng Chen 260775674

Part 1 Slider Switches and LEDs program

At the beginning of the lab, a basic I/O driver program is implemented to make the nine slider switches to be able to control the corresponding LEDs. The project folder structure is given by the lab manual. The “asm” folder stores the assembly code formatted as .s file and the inc folder stores the C language header file packaging the subroutines written in assembly. To access the I/O, official Altera DE1-SoC Computer documentation provides the addresses of the needed I/O and its corresponding usage. In this part, the “read” subroutine simply loads the content in the switches’ address into the register to be returned (R0) and “write” for the LEDs program does a similar task in opposite direction. Then the main C program with the inputs of the header file uses the write and read subroutine so that whichever slider switch is switched upwards, the LED next to it lights up.

Part 2 Entire Basic I/O Program

In this part, two more complexed I/O systems, HEX display, and pushbuttons are used. The HEX display driver defines the methods to clear or light up all segments of the display, as well as display specific numbers in hexadecimal. All the methods take the one-hot encoding of the HEX displays defined as a datatype using enumerations. Within the subroutine, loops and masking methods are used to check each digit of the input and carryout corresponding actions or do nothing since there could be multiple HEX displays encoded in the input. The challenging part of this step is that the displays have two different addresses with one for HEX 0-3 and another for HEX 4 and 5. Meanwhile, the specific values need to be written into different digits in the 32-bit memory of the address. Therefore, after the counter is initialized and the bits are checked, if the counter reaches 3, it means that the HEX 4 is being selected. In this case, the counter changes by subtracting by 4, and the target address is updated. To be able to write the specific digits with others unchanged, the contents to be written are shifted according to the counter. For example, the digit 7-14 controls HEX 1 so that the content needs to be shifted left by 8 bit for one time. Similarly, HEX 3 requires the content to be shifted three times. Finally, the write subroutine defines the 16 different contents to be written with

respect to the input and uses similar approaches as the “clear” and “flood” subroutine to process the input, and update the target address.

Finally, the four pushbuttons are introduced to the program to control the HEX displays 0-3 and the slider switches are used to define the input value to be displayed on the 7-segments.

Part 3 Polling Based Stopwatch

In this part, a stopwatch was created which displays the current time on the hex displays. Three subroutines were created which are HPS_TIM_config_ASM, HPS_TIM_read_INT_ASM and HPS_TIM_clear_INT_ASM. The configuration subroutine used a counter to configure each of the four timers. To configure the timer, the data stored in its address was modified. Bits M, I and E were initialized as specified in the manual. After the initialization, the data is then stored back to its address. The read subroutine reads when the timer will be expired by reading the S bit of each timer. The clear subroutine simply resets the timer.

There was one timer used for the time count and another timer used for polling information from the pushbuttons. The pushbuttons are regularly checked to see if the timer needed to be reset, paused or restarted. There were three variables in the first timer which are centiseconds, seconds and minutes. Whenever centiseconds reached 100, it was reset to zero. Whenever seconds reached 60, it was reset to 0.

The biggest challenge we faced was that we had a hard time trying to understand the manual. However, after some discussions with the TAs, we were able to solve it.

Part 4 Interrupt Based Stopwatch

In this part, an interrupt based stopwatch was implemented. Interrupted based timer works in a different way when compared to polling based stopwatch: it monitores the interruption bit of the timer to update the timer accordingly. One thing to note is that only one timer is used for interrupt based stopwatch. With the help of the int_setup file given by the instructor, we were able to receive interruption from the pushbutton keys flag. When a pushbutton is pressed, the program will be interrupted and read the pushbutton keys subroutine which reads the edgecap register and clear the edgecap register after that.

The biggest challenge we faced was that we had a hard time trying to understand how interruption works. However, after some discussions with the TAs, we were able to solve it.