Carlo D'Angelo 260803454
Mostafa Abdelwahab 260551876
March 29th, 2019
ECSE 324 – Computer Organization

Lab 3: Basic I/O, Timers and Interrupts

"The hardware setup of the I/O components is fairly simple to understand. The ARM cores have designated addresses in memory that are connected to hardware circuits on the FPGA, and these hardware circuits in turn interface with the physical I/O components" (Lab 3 instruction sheet).

Slider Switches and LEDs Program

The slider switches on the DE1-SoC board are connected to an input parallel port. This port comprises a 10-bit read-only *Data* register (1 bit for each slider switch), which is mapped to address 0xFF200040.

The red LEDs on the DE1-SoC board are driven by an output parallel port. The port contains a 10-bit *Data* register (1 bit for each LED), which is mapped to address 0xFF200000.

The state of the slider switches and LEDs are represented by a logical '0' (i.e., OFF) and a logical '1' (i.e., ON). This state can be read and, in the LEDs' case, written by the processor using word accesses. Moreover, the upper bits not used in the registers are simply ignored.

The key of this program is that you read the input data from the slider switch *Data* register and write that data to the LED *Data* register. In doing so, the LEDs will light up or turn off when their corresponding (i.e., same bit location) slider switch is switched ON or switched OFF. To implement this program, we wrote two subroutines: read_slider_switches_ASM and write_LEDs_ASM. The former returns an integer (32 bits) representing the state of the slider switched and the latter takes in an integer argument representing which LEDs will be turned ON or turned OFF.

Entire Basic I/O Program

For this part of the lab, we first reused the code from the first section to implement the slider switches and LEDs part.

We, then, wrote code to use the pushbutton switches and HEX displays. Similarly to the first part, the pushbutton switches and HEX displays are connected to their own parallel ports respectively, so the processor can easily access their *Data* registers by using the correct memory address (0xFF200050 for the pushbuttons or 0xFF200020 and 0xFF200030 for the HEX displays).

The key of the pushbutton and HEX display part was to use the PB_edgecap_is_pressed_ASM subroutine to indicate whenever a pushbutton is pressed, the read_slider_switches_ASM subroutine to know what hexadecimal symbol to draw on the HEX displays and to know the state of the 9th slider switch, the HEX_write_ASM subroutine to actually write that hexadecimal symbol to the HEX displays and the HEX_clear_ASM subroutine to clear the HEX displays when the 9th slider switch is ON.

Polling Based Stopwatch

The polling based stopwatch is based on 2 timers: the first timer is has a timeout of 10 ms, the second timer a timeout of 5 ms. The 5 ms-timer is the "backbone" for the polling mechanism. While in an infinite loop, the c program checks whether the press buttons (to start, pause, reset) have been pressed when the 5 ms timer runs out of time then this timer is reset again. There is an int "counting" behaving as a boolean to indicate whether the stopwatch is running or is stopped. Each time the 10 ms timer runs out of time, an integer is incremented by 1 (if the stopwatch is running). This int storing time in increments of 10 ms is then used to display time (the hundredths are displayed on HEX-0 by doing time %10, the tenths are displayed on HEX-1 by doing (time/10) % 10 and so on.

Interrupt Based Stopwatch

The interrupt-based stopwatch uses a 10 ms tunout time to measure time and displays it the same way as the previous stopwatch. Instead of using a second timer to poll the push buttons, this stopwatch enables the interrupt bits (setting the corresponding mask bits to zero) in the timer and push button registers. We then wrote an Interrupt service routine that changes the value of a global variable when the interruption occurs. Each time the 10-ms timer runs out, the interrupt changes the value of hps_tim0_int_flag to 1, and then the c program can detect that this cycle has finished (and sets hps_tim0_int_flag back to zero). When any button is pressed, the corresponding ISR copies the value stored in the edgecapture register of the pushbutton to the global variable PB_int_flag so that the c program recognizes which button has been pressed. We then reset the edge capture register back to zero.