Engineering 304
Lab #5: NIOS II Assembly Language – I/O Devices

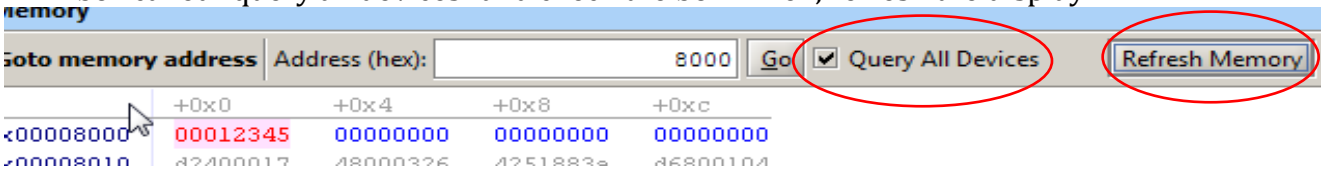
Purpose
The purpose of this lab is to learn about how to interface with I/O devices.

Overview
In this lab you will work with a NIOS processor system that includes the memory assignments shown in the table below. Note that only the Base Address is provided. You will need to use the documentation for the particular I/O device to determine what control registers are found at that base address. The questions listed in this writeup can be answered using the "Answer Sheet…" document, also found on the shared drive.

Module Name	Base Address	DE2 Item	Port Type
On-chip Memory	0x0 (through 0x7fff)	Internal memory (32k)	Memory
SW	0x8000	18 slide switches	PIO port (input)
LEDR	0x8040	18 Red LEDs	PIO Port (output)
SevenSeg3to0	0x8080	HEX3,2,1, and 0	PIO Port (output)
SevenSeg7to4	0x80c0	HEX7,6,5, and 4	PIO Port (output)
KEY	0x8100	Four Push buttons	PIO Port (input) w/ IRQ 1
Timer	0x8140	Internal timer	Programmable Timer w/ IRQ 0

Part I – Basic I/O Using the Memory Tab
Perform the following steps to become familiar with using various I/O ports on the DE2 board.

1. Create a new folder for this project and download the *.sof, *.jdi, and *.qsys files from the share drive and place them in this folder. Note that <u>these are new files</u>, implementing a different system than was used in the previous labs.
2. Use the Altera Debug Monitor to create a new project in that folder using the above 3 files. For now, simply use the assembly code "sampleprog.s" when asked for a file to compile/assemble.
3. Load the compiled code into the debugger and switch to the memory tab. Locate the check box called "query all devices" and check the box. Then, refresh the display.



4. Locate the SW port in memory (see the above table) and identify which word represents the port's DATA register. Change some of the slide switches and refresh the memory tab.
   <u>Q1:</u> What hex number do you find in the port's DATA register when SWitches 1, 5, 10, 14, and 17 are up and the rest are down?
5. Locate the LEDR port in memory and identify which word represents the port's DATA register. Try changing the value in the port's DATA register to see what happens.
   <u>Q2:</u> What hex number would you write into the LEDR port's DATA register to turn on LEDs 16, 13, 11, 10, 6, 3, and 1 , keeping the other LEDs off?
6. Locate the KEY port in memory and identify which words represent the port's DATA and EDGE registers. Check the state of both registers before pressing any KEYs. Next, press <u>and hold</u> KEY3 while clicking on the "Refresh Memory" button in the debugger. Compare the register states to their previous values prior to pressing the key. Next, release the

button and compare the register states to what they were prior to and during a button press. Do the same with KEY2 and KEY1. (KEY0 will reset the system, so leave that alone for now.) Write a 0 to the EDGE capture register. Then, press and release KEY3 and KEY1 at the same time.

Q3: What HEX values do you find in the DATA and EDGE capture registers after pressing and releasing KEY 3and and KEY1?

Q4: Why are they different and how might those registers used?

7. Locate the SevenSeg3to0 port in memory and identify which word represents the port's DATA register. Try writing different values into the port's DATA register to see what happens (suggestion: try patterns with most of the bits set to 0 instead of 1). Look for a pattern the relates the bits of each byte in the register with a particular segment that is turned on in each display.

   Q5: What hex number would need to be written into the port's DATA register to turn on only the middle segment in the HEX displays 3, 2, 1, and 0?

   Q6: What do bits 7, 15, 23, and 31 of the DATA register do to the seven segment display?

8. Write a HEX number to the SevenSeg7to4 port's DATA register to spell "dEAd" and another HEX number to the SevenSeg3to0 port's DATA register to spell "bEEF". (Hint: you may want to draw the seven segment patterns on a piece of paper.) Demonstrate for your instructor so you can be signed off.
9. Consider the assembly instructions necessary to read from the SW port.

   Q7: What instruction would load the gp register with the base address of the SW port?

   Q8: What instruction would read the DATA register of the SW port into r8, assuming the gp register has the base address?

   Q9: What instruction would read the EDGE register of the SW port into r9?

Part II – Changing Displays Over Time

In this part, you will write some simple assembly code that displays a counter value found in a register. The count value in the register is incremented until it reaches the value 0x00040000 at which it resets to zero and starts to count up again. The pseudo-code for this routine is:

```
int i = 0;
while (1) {
  write out the value of i to the LEDR port's DATA reg using STWIO instruction
  i++;
  if (i == 0x40000)
    i = 0;
}
```

"Action to be done"

1. Write a simple assembly code file to do the above operation; a template file is provided on the shared drive to use as a starting point for your code, but save it with a better name. The address and port register offsets are defined in the nios_defs.s file which is automatically included in the template file. You should review the nios_defs.s file to become familiar with what it does, but you should not need to modify it. After you have written your assembly code, compile and load the program to the DE2 board.
2. Run the program with a breakpoint in the main loop, repeating the loop at least 15 times.

   Q10: What do you expect to see on the LEDs if you let the code run freely without the breakpoint?

3. Next, remove the breakpoint and rerun the program without breakpoints.

   Q11: What did you see on the red LEDs (describe it using specifics) when the loop runs freely?

4. Next, add a delay mechanism (using assembly instructions) inside the while(1) loop. A simple delay mechanism can be built by incorporating a counting operation such that the action to be done is only done after the count reaches its maximum value (see the pseudocode below.) Estimate what that delay maximum count value (DELAYMAXCOUNT) would be so that the red LED port is updated once every 0.5 seconds, each time i is incremented. Test it and adjust the delay max count value a few times to see how close you can get to 0.5 seconds.

   Q12: What inner loop max count value did you come up with that gives approximately 0.5 seconds of delay to the outer while loop? _____

```
int i,j = 0;
while (1) {
  if (j == DELAYMAXCOUNT) { /* delay done, so adjust the LEDR state */
    write out i to the LEDR port's DATA register using STWIO instruction
    i++;
    if (i == 0x40000)
       i = 0;                          "Action to be done"
     j = 0;
    }
    else
      j++;  /* do nothing but increment j */
}
```

5. Save a copy of your assembly code and append "_CounterDelay" to the filename, do not modify this snapshot in the future steps.
6. To achieve a more exact delay, revise your code to use the built-in timer. First, determine how many times a 50 MHz clock would tick in 0.50000 seconds.

   Q13: What is that number written out as a 32-bit number in hex (note: this is the "PERIOD")?

   Q14: What would the two 16-bit numbers (PERIODH and PERIODL) be for the PERIOD value from Q13?

7. Next, remove the delay code from the previous part. Add code before your main loop to initialize the important timer port registers: Use the value for the PERIOD (see Q13) to setup the PERIODH and PERIODL registers and initialize the CONTROL and STATUS registers so that the timeout flag has been cleared and the timer has been started and setup to run continuously without generating interrupts. Finally, add code in the while(1) loop to wait until the TO bit is set from a timer timeout event that happens elsewhere in hardware before doing the action to be done. Test your code to see how it works.

```
int i = 0;
Initialize the timer;
while (1) {
   if (TO flag is set) { /* delay done, so adjust the LEDR state */
     Clear the TO flag;
     write out i to the LEDR port's DATA register using STWIO instruction
     i++;
     if (i == 0x40000)
       i = 0;                          "Action to be done"
   }
}
```

8. Save a copy of your assembly code and append "_TimerDelay" to the filename, do not modify this snapshot in the future steps.

Part III – Switch and Button Presses

Copy your *_TimerDelay.s file from Part II and modify the copy for this version. In this version, you will add the capability of reading the state of switches 15..0 as the new value for the timer port's Period High (PERIODH) register. The new state of the switches will be recorded and used only when the push button KEY3 has been pressed and released. Your code should initialize the timer's PERIOD to the default 0.5 seconds (from Part II) and then change the PERIOD after KEY3 is pressed. The loop should still update the displays when the timer times out, but the loop will need to also look for the KEY3 "edge" flag to be set and act on that event when it occurs. Be sure to re-enable the timer after you have updated a period register since it automatically stops when either PERIODH or PERIODL are modified. If you do not have a well-thought out plan, you should begin with pseudo code to describe the operations in the loop. Below is a generic version of pseudocode that outlines what should happen each time around the while loop.

```
while (1) {
  if (timer timeout occurred) {
     … {do an action}
  }
  if (KEY3 pressed) {
     … {do an action}
  }
}
```

Test your code. Either have your instructor sign off when you demonstrate it for him or record a video that clearly shows you demonstrating your working system. Do not email such a video; upload it to a place (e.g. OneDrive, google drive, DropBox, YouTube) from which you can share the link – make the link publicly accessible so that any grader can view it. Include a properly formatted printout of your code in your writeup.

Save a copy of your assembly code and append "_TimerDelayWithKey" to the filename.

Hand In:

1) The lab answer sheet (with signoff for Part 3 or a link to your video)
2) Properly formatted printouts of your assembly code files.

Extra Credit:

Replicate the DE-2 power-up light patterns (only the HEX displays and the red LEDs) as much as possible using assembly code and the I/O devices available to you. Demonstrate for your instructor and provide a printed code listing (properly formatted) with a short explanation of how you built the system.