ECE3073 Computer Systems

Laboratory Session 1
Altera Monitor and Nios Machine Code

WEEK 2, 2023

1. Objectives

This laboratory provides an introduction to using the Altera Monitor Program and the facilities it provides for examining/modifying registers and memory together with single-stepping and running Nios programs. You will also make use of the disassembly function of the Monitor. These facilities of the monitor give you a lot of tools for testing a debugging assembler and C-code programs. In addition you will investigate using the monitor to 'patch in' additional code into an already existing program. This technique is useful if you want to modify a program for which you do not have the source code.

These short laboratory exercises were written to help develop your understanding of:

- The operation of the Altera Monitor Program disassembler
- Using the Monitor Program to single step through a program
- Using break points
- Examining and changing register contents
- Patching an assembler program using the Monitor

Supporting information about the Monitor can be found in the Altera Monitor Program Tutorial.

Equipment

§ DE2 FPGA Development Board
§ A USB memory device provided by you to backup your design files

2. Preliminary work

a) What is the part number of the FPGA on your DE2 board?

……

b) Write Nios II assembler instructions that will invert bits 3 to 0, set (turn on) bits 11 to 8 and reset (turn off) bits 31 to 12 in register r4. All other bits in r4 should be left unchanged. Finish off this piece of code with a jump immediate to address 0x00000040:

……

……

……

……

c) Now convert the four assembler instructions that you wrote for part b into machine code. Write the binary and hexadecimal representation of each:

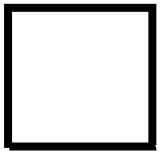
……

……

……

……

Demonstrator to initial the box to the right at the beginning of the lab to indicate satisfactory completion of the preliminary work. A cross indicates that preliminary work was not completed before the lab.



3. First Exercise

In this first exercise you will use the Altera Monitor Program to download a pre-written hardware design and assembler program into the DE2 system. More detail of these following steps is given in the Altera Monitor program Tutorial (available on Moodle).

a) Create a new folder for the files required for this exercise. Make sure the folder name is unique and the path does not contain any spaces. Don't save files on the desktop folder as it a network mapped folder. Try "C:\Users\<your username>".

b) Start the Monitor Program either using the desktop shortcut or by searching for it in the windows start menu.

If you get the message that "License file not specified" then copy the file "quartus2.ini" (on Moodle) to "C:\Users\<your username>"

c) Create a new project by selecting **File > New Project**. Use your new folder as project directory and call the project your initials followed by lab1 (no spaces). Then click Next.

d) In the 'Specify a system' window select a system should be 'DE2 Basic Computer'. Then click **Next**.

e) For 'Specify a program type' choose 'Assembly Program' and check the 'include a sample program with the project' box and select the sample program 'Getting Started'. Then click **Next**.

f) Click **Next** to skip 'Specify program details'.

g) In the 'Specify system parameters' window, check that Host Connection is 'USB-Blaster [USB-0]', Processor 'CPU' and Terminal device 'JTAG_UART'. Then click **Next**.

h) 'Specify program memory setting' you should leave the values as set. Then click **Finish**.

i) Make sure your DE2 board is powered on and plugged into the computer, then accept the offer to download the hardware design to the DE2 board.

Now the DE2 board should be configured as a Nios II microprocessor system.

j) To complete the system we are now going to compile the sample program 'getting_started.s' using the **Actions > Compile** menu item or click on the associated icon (you can check out the function of the icons by 'hovering' the cursor over them).

k) When the code is successfully compiled you should download it to the DE2 system using the **Actions > Load** menu item or click on the associated icon.

4. Second Exercise

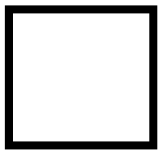
i) Run the program by clicking on the **Actions > Continue**, F3 or the associated menu item. Confirm that the program does the following:

- Displays the DE2 board's SW switch states on the red lights (LEDR)
- Displays the KEY1, KEY2, and KEY3 pushbutton states on the green lights (LEDG)
- Shows a rotating pattern on the HEX displays. If KEY1, KEY2, or KEY3 is pressed, the pattern is changed to correspond to the settings of the SW switches. KEY0 will stop the execution of the program.

Then stop program execution, either from the monitor program menu or by using KEY0.

ii) Using the Disassembly Window

Pseudo operations are instructions where the original source code instruction is not the same as the disassembled instruction/s as displayed in the Disassembly Window. Use this fact (or refer to the Nios instruction reference on Moodle) to identify all of the pseudo operations in "getting_started.s" and highlight them on the program listing at the end of these notes.

Demonstrator initials for satisfactory identification of pseudo operations 

iii) Single Stepping – reset the processor (**Actions > Restart** or clicking on the associated icon) and single step the program (**Actions > Single Step**, F2 or associated icon) up to the delay loop. Before you single step the program you should anticipate what the result of the next instruction will be in terms of changes to registers including the program counter, changes to memory, lights on the DE2 board, etc. After each single step check that you anticipated correctly.

iv) Using Breakpoints – An instruction breakpoint provides a means of stopping a Nios II program when it reaches a specific address. A simple procedure for setting an instruction breakpoint is:

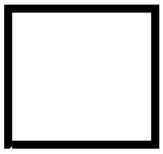
- In the Disassembly window, scroll to display the instruction address that will have the breakpoint. As an example, scroll to the instruction at the label NO_BUTTON, which is address 0x0000005C. In a larger program it may be more convenient to type the label into the Goto instruction box and then click on **Go**.

- Click on the gray bar to the left of the address 0x0000005C in the Disassembly window. The Monitor Program displays a red dot next to the address to show that an instruction breakpoint has been set. Clicking the same location again removes the breakpoint.

- Once the instruction breakpoint has been set, run the program. The breakpoint will trigger when the pc register value equals 0x0000005C. Control then returns to the Monitor Program, and the Disassembly window highlights in yellow the instruction at the breakpoint. A corresponding message is shown in the Info & Errors pane.

v) Examining and changing register contents - The Registers window on the right-hand side of the Monitor Program display shows the value of each register in the Nios II processor and allows the user to edit most of the register values. The number format of the register values can be changed by right-clicking in the Registers window. Each time program execution is halted, the Monitor Program updates the register values and highlights any changes in red. The user can also edit the register values while the program is halted. Any edits made are visible to the Nios II processor when the program's execution is resumed.

As an example of editing a register value, first scroll the Disassembly window to the label DELAY, which is at address 0x00000074. Set a breakpoint at this address and then run the program. After the breakpoint triggers and control returns to the Monitor Program, notice that there is a large value in register r7. This value is used as a counter in the delay loop. Double-click on the contents of register r7 and edit it to the value 1. Press Enter on the computer keyboard, or click away from the register value to apply the edit. Now, single-step the program to see that it exits from the delay loop after one more iteration, when r7 becomes 0.

Demonstrator initials that you are able to modify r7 to exit the wait loop early. 

5. Third exercise

In this exercise you are to modify the assembler code in "getting_started.s" and use the functions available in the Monitor Program to test your modifications. The changes to the program will be made by directly adding NIOS machine code instructions to program memory.

As it originally stands "getting_started.s" reads the state of the slider switches on the DE2 board and then immediately writes those values to the red LEDs. The changes that you will make to the program will modify the data read from the slider switches before writing it to the LEDs. These modifications are described in the preliminary work section 2(b). You should already have machine code instructions for performing these operations in answer to section 2(c) of the preliminary work.

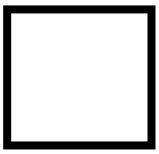
All of the required changes are as follows:

1) Overwrite the ldwio r4,0(r15) instruction at label DO_DISPLAY: with an unconditional branch (br) to memory address 0x00000090. Of course you will need to work out the corresponding machine code. Make sure to note down the machine code value that you replaced as you will use it later.

   Note that to change the machine code at this location you will need to change the monitor display from "Disassembly" to "Memory".

Record your machine code for the branch here:

……

Demonstrator initials for correct machine code for the branch instruction 

2) Write the machine code for the ldwio r4, 0(r15) instruction that you overwrote in part (1) at memory location 0x00000090.

3) Then in the following successive memory locations (groups of 4 bytes) insert the 4 machine code instructions from your preliminary work.

These memory changes should also be performed in the Memory Window (see the Altera Monitor Program Tutorial page 19 if you have any problems).

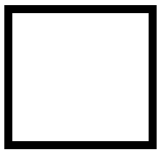
Note that the Disassembly window will not immediately reflect the changes that you have made to memory. To disassemble your added code do the following:

- restart the project **File > Open Recent Project >** your project
- reply **YES** if it asks about ending the current session and **NO** to the offer to download the system associated with the project
- connect to system **Actions > Connect to System**

The disassembly of your code will appear in the Disassembly window. However, labels, comments and pseudo code are not available. Also the jump immediate at location 0x000000A0 appears as a "?" because the disassembler does not support it (However, the Nios processor will execute the instruction correctly).

At this point **DO NOT** assemble and download the original program. This will overwrite the changes you have made.

Single step through your code to ensure that it works correctly before selecting run mode.

Show the demonstrator that your modified program works correctly. 

When you have completed the lab or when time is running out then start the assessment quiz for this laboratory exercise. The demonstrator will enter the password and record your mark (out of 5, one for each check box).

Ensure that your mark for this exercise is entered before you leave the lab.

```
/*********************************************************************************
 * This program demonstrates use of parallel ports in the DE2 Basic Computer
 *
 * It performs the following:
 *      1. displays the SW switch values on the red LEDR
 *      2. displays the KEY[3..1] pushbutton values on the green LEDG
 *      3. displays a rotating pattern on the HEX displays
 *      4. if KEY[3..1] is pressed, uses the SW switches as the pattern
*********************************************************************************/
        .text                   /* executable code follows */
        .global _start
_start:

                                /* initialize base addresses of parallel ports */
        movia           r15, 0xFF200040        /* SW slider switch base address */
        movia           r16, 0xFF200000        /* red LED base address */
        movia           r17, 0xFF200050        /* pushbutton KEY base address */
        movia           r18, 0xFF200010        /* green LED base address */
        movia           r20, 0xFF200020        /* HEX3_HEX0 base address */
        movia           r21, 0xFF200030        /* HEX7_HEX4 base address */
        movia           r19, HEX_bits
        ldw             r6, 0(r19)             /* load pattern for HEX displays */

DO_DISPLAY:
        ldwio           r4, 0(r15)             /* load slider switches */
        stwio           r4, 0(r16)             /* write to red LEDs */

        ldwio           r5, 0(r17)             /* load pushbuttons */
        stwio           r5, 0(r18)             /* write to green LEDs */
        beq             r5, r0, NO_BUTTON
        mov             r6, r4                 /* copy SW switch values onto HEX
displays */
WAIT:
        ldwio           r5, 0(r17)             /* load pushbuttons */
        bne             r5, r0, WAIT           /* wait for button release */

NO_BUTTON:
        stwio           r6, 0(r20)             /* store to HEX3 ... HEX0 */
        stwio           r6, 0(r21)             /* store to HEX7 ... HEX4 */
        roli            r6, r6, 1              /* rotate the displayed pattern */

        movia           r7, 100000             /* delay counter */
DELAY:
        subi            r7, r7, 1
        bne             r7, r0, DELAY

        br              DO_DISPLAY

/*********************************************************************************
        .data                                  /* data follows */

HEX_bits:
        .word 0x0000000F
```

6. Fourth Exercise

Use CPULATOR Following link : https://cpulator.01xz.net/ Demonstrate that you can compile, disassemble the code and also able to manipulate the memory map as demonstrated in the lab demo video in the Moodle. Upload the screenshots (in pdf format)of the CPULATOR screen showing the memory map and disassembly.