University of Toronto
Faculty of Applied Science and Engineering
Department of Electrical and Computer Engineering

Final Examination
ECE 253F - Digital and Computer Systems
Friday December 10, 2010

Duration: 2.5 Hours
Examiner: J. Rose

ANSWER ALL QUESTIONS ON THESE SHEETS, USING THE BACK SIDE IF NECESSARY.
1. No calculator and no cellphones are allowed.
2. The number of marks available for each question (and parts of questions) is indicated in the square brackets [].
3. There are two extra blank pages at the end of the test for rough work.

AIDS ALLOWED: (i) The Custom Course Textbook (including prior years’ versions) or Fundamentals of Digital Logic with Verilog Design, 1st or 2nd Edition.
(ii) One 8.5 x 11” two-sided aid sheet. (Does not need to be official aid sheet)

__________________________________________________________________________________

Last Name: ________________________________________________
First Name: ________________________________________________
Student Number: ____________________________________________

Total Marks Available/Achieved:

<table>
<thead>
<tr>
<th>Question</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marks Available</td>
<td>12</td>
<td>8</td>
<td>12</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>8</td>
<td>70</td>
</tr>
<tr>
<td>Marks Achieved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Q1. Short Answer Questions.** For each of the following questions, give the answer as indicated, or a 1 or 2 sentence answer.

(a) [2] Consider a memory (that might be a main memory attached to a processor) that contains 256 addresses and stores 2 bytes at each address.

(i) How many address wires are in its Address bus?
   Answer:
   8 (1 mark)

(ii) How many wires are in its Data Out bus?
   Answer:
   16 (1 mark)

(b) [2] What part of the execution of the Nios II instruction eret causes processor interrupts to be re-enabled?

   Answer:
   When ctl1 (estatus) is copied into ctl0 (status). 2 marks
   (this sets the PiE bit back to enabled)
   (just talking about the PiE bit being turned on isn’t right)

(c) [2] Under what circumstances would the use of a cache memory actually decrease the performance of a processor?

   Answer:
   When there was no locality of reference in the memory requests; for example, the data requests were random across the entire main memory. 2 marks.

   Possible alternative: if memory is much faster than processor
Q1, continued.

(d) [2] In the first 5 labs of this course, you “programmed” the FPGA chip using mostly the Verilog Hardware Description Language. In the last 4 labs, you programmed the same chip, but using the Nios II Assembly Language. Explain how it is possible to program this chip in these two different ways.

Answer:

An FPGA can be programmed to become any digital circuit, using Verilog (or schematics for that matter), which is what we did in the first part of the course. In the second part of the course the FPGA was first programmed to become a processor (by the Altera Monitor), and that Nios II processor is itself programmable through Assembly/machine level instructions.

(e) [2] Suppose that you wrote a recursive subroutine, which received its parameters on the stack. You made a mistake, though: you forgot to have a recursion termination condition. What would be the first thing to go wrong when that subroutine is executed?

Answer:

The stack would continue to grow downwards in memory, with each recursive call, until it over-ran the text & data section, completely messing up execution of the program. (infinite loop -> ran out of memory)

(f) [2] What is the key difference between a fully-encoded state machine and a 1-hot encoded state machine?

Answer:

1-hot uses 1 flip-flop per state, where as the number of flip-flops in a fully encoded machine is ceiling (log2(# states))

1-hot can have only 1 state bit high at a time, fully-encoded can have more than one. (not quite correct).
Q2. [8] **Assembly language programming and subroutines.** Be sure to read this question completely, as there some specific rules about your implementation given.

You are to write a Nios II Assembly Language subroutine, **CountBig**, that searches through a list of word-size numbers and counts how many of them are larger than 7245 (base 10). The subroutine is passed the beginning address of the word list in register **r4**. The address of the final word in the list is given in register **r5**. (That is, the list can be of any length > 0 words, with [r4] giving the beginning address and [r5] the final address of the word on the list). The subroutine should return the result in register **r2**.

Your subroutine may only use registers **r16** to **r23**, in addition to the ones mentioned above. For all of the registers (except r2) that you use as part of your subroutine, you must save and the prior contents of these registers on the stack, and restore them after use. You may assume that the stack register is set appropriately prior to the call of your subroutine.

**Be sure to provide comments that say what your subroutine is doing, these will be worth marks.**

Use this space for your rough work; place your final answer on the following page.
Solution:

CountBig:

/* Push registers to be used onto stack */
subi sp, sp, 12
stw r16, 0(sp)
stw r17, 4(sp)
stw r18, 8(sp)

mov r16, r4  # grab a copy of starting address
mov r2, r0  # initialize count of big numbers to 0
movia r17, 7245 # put number to be compared in r17

Loop:
ldw r18, 0(r16) # get next number from list
ble r18, r17, NotBig # if not bigger than 7245 don’t count
addi r2, r2, 1 # increment bigger counter

NotBig:
addi r16, r16, 4 # advance address
ble r16, r5, Loop # if don’t hit end, continue

/* pop saved registers, off stack */

ldw r16, 0(sp)
ldw r17, 4(sp)
ldw r18, 8(sp)

addi sp, sp, 12

ret # return from subroutine

Worth 8 marks.

2 marks for proper save and restore of all used registers
2 marks for a loop that properly traverses the list
2 marks for counting properly
1 mark for the ret statement and using r2 to return a value
1 mark for comments

saving/restoring r4/r5 is OK (if used)
-0.5 for syntax errors
Q3. **Processor Parallel Input/Output & Polling.**

You’ve been hired as an Imagineer at Disney World, and are tasked with writing the software that drives a part of a mechanical wizard - specifically the wizard’s arm which is holding a wand. The arm and hand are controlled by motors that are themselves controlled by a custom-built (by someone else) Wand Control Unit (WCU), as illustrated in the picture below.

Fortunately, the computer that is connected to the WCU is an Altera DE2 Basic Computer, just like the one you’ve used in the ECE 253 lab.

![Diagram of DE2 Basic Computer and Wand Control Unit](image)

The WCU is sophisticated; all you have to do is send it an 8-bit code to have it execute an entire wand movement. For example, code 0x22 is called ‘Expelliarmus’ and is a striking motion, while code 0x4b is ‘Obliviate’ which causes the arm to go through more of a twisting motion. The WCU can receive one of 256 different such codes/commands.

A code/command is sent over bits 0-7 of the JP1 cable from the DE2 computer to the WCU, which begins ‘executing’ the command as soon as a new command is sent (you can’t send the same command twice in a row). Since it takes quite a bit of time to make the arm move, the computer must wait until the wand is finished moving to send a subsequent command. Bit #8 of the JP1 cable is an input from the WCU to the DE2, and is set to 1 when the wand/arm is finished moving. The program running on the DE2 must check this bit before sending a new command to the WCU. When a new command is sent to the WCU, bit #8 is reset to 0, by the WCU.

You are to write a Nios II assembly language program that sends 100 byte-sized commands to the WCU. These commands are stored in memory, as consecutive bytes, beginning at the location `Wand_Com`. `Wand_Com` is set up as shown in the code fragment below, and the 100 memory locations are initialized by other software that you do not have to write.

Recall that the JP1 connector is controlled by the data register at memory-mapped address 0x10000060, as well as the direction register at memory-mapped address 0x10000064. Note that the JP1 base address value is defined for you in the code fragment below.

Be sure to provide comments in your code that say what it is doing, these are graded.
SOLUTIONS SOLUTIONS SOLUTIONS SOLUTIONS SOLUTIONS SOLUTIONS SOLUTIONS

Q3, continued.

Front code fragment:

```assembly
.equ JP1, 0x10000060

.sections .data
Wand_Com .skip 100

.sections .text

{Your Code Goes Here}

Answer:

Solution:
_start movia r8, JP1
    movia r9,0xff      # set the data direction register so that bits 0-7 are outputs, bit 8 is input
    stwio r9, 4(r8)    # +4 is direction register
    movi r10,100      #initialize byte counter
    movia r11,Wand_com # r11 keeps track of which byte we’re on

Loop:  ldb r12, 0(r11)       # get the next byte (make sure it isn’t a word)
      stwio r12,0(r8)     # send it to the WCU
      /* Poll to wait till done */
Poll:  ldwio r13,0(r8)
      andi r13,r13,0x100  #and out everything but bit 8
      beq r13,r0,Poll     /* Done */
      addi r11,r11,1      #increment byte count
      subi r10,r10,1      # decrement loop counter
      bgt r10,r0,Loop

END:   br END
.end
```

Worth 12 marks - 2 for setting Direction Register, 3 for outer loop, 2 for correct transmission of data to WCU, and 3 for the correct polling loop, 2 marks for comments.
- OK if use 1 for input, 0 for output of direction register
- if worked with 7 bits, not 8, -.5, totally off -1
- outer loop if infinite - -2; infinite loop but with counter just -1
- data transmission wrong - ldw instead of ldb -1, but don’t penalize twice

Consider the schematic diagram shown below, which has four separate blocks in it. You are to give the complete Verilog HDL code for this circuit, in a hierarchical form: each block should be described as a separate module, and joined together using a fifth module. You should use the highest-level behavioural form (which is the most concise form) of Verilog that you are able to create. Higher marks will be given for solutions that are more behavioural in form.

Every input to the circuit that is not driven should be made a primary input of the top-level module. The output ALUout should be the top-level module’s primary output. Assume that the carry-out of the adder can be ignored, and that the Counter Reset signal is synchronous and active high.

Begin your answer here, and continue on the next page:

Solution: 2 marks for each module, including top-level module; -1 for each module not behavioral -1 for every syntax/logic error in each module; -1 for active low reset

module q4 (Clock, DataIn, SelPath, EnableA, ResetC, ALUOut);
input Clock, SelPath, EnableA, ResetC;
input [9:0] DataIn;
output [9:0] ALUOut;
wire [9:0] MuxOut, RegAOut, CountOut;
mux10 muxA(ALUOut, DataIn, SelPath, MuxOut);
reg10 regA(Clock, EnableA, MuxOut, RegAOut);
count10 Counter(Clock, ResetC, CountOut);
Q4, continued

adder10 Adder (RegAOut, CountOut, ALUOut);
endmodule

module mux10 (In0, In1, Select, Out);
    input Select;
    input [9:0] In0, In1;
    output reg [9:0] Out;

    always @(In0, In1, Select)
        if (!Select)
            Out = In0;
        else
            Out = In1;
endmodule

module reg10 (Clock, Enable, D, Q);
    input Clock, Enable;
    input [9:0] D;
    output reg [9:0] Q;

    always @(posedge Clock)
        if (Enable) Q <= D;
endmodule

module count10 (Clock,Reset,Count);
    input Clock, Reset;
    output reg [9:0] Count;

    always @(posedge Clock)
        if (Reset)
            Count <= 0;
        else
            Count <= Count + 1;
endmodule

module adder10 (In0, In1, Out);
    input [9:0] In0, In1;
    output reg [9:0] Out;

    always @(In0, In1)
        Out = In0 + In1;
endmodule
Q4, continued.
Q5. [10] **Combined Finite State Machine and Processor Architecture.** Consider the processor and main memory system shown below. The datapath shown is just part of a complete processor that is similar to the simplified processor discussed in class.

In this question you are required to design part of the finite state machine that implements the *execution* (but not the instruction fetch) of the instruction `and r1, r1, (r2)`. This instruction computes the bitwise AND of register `r1` with the contents of the memory location pointed to by register `r2`, and places the result in register `r1`. Your state machine should also increment the program counter register (PC) by 4, after executing the the `and` instruction.

In the processor shown, registers `r1` and `r2` exist as separate entities. The ALU shown has two possible functions, Add and AND, determined by the control signal entering the left side of the ALU, `Add/And`. When `Add/AND = 1` the ALU function is Add; when `Add/AND = 0` the ALU function is bitwise AND.

You should also assume the following about the datapath and memory shown:

(i) Each register shown - MAR, MDR, PC, R1 and R2, is clocked by the same clock and has an enable signal (even though it isn’t shown on the figure). The enable signals are active high.

(ii) The processor interface to the memory works in the following way: the memory is connected to the same clock as the processor registers. A memory read operation takes place over two clock cycles: in the first cycle, the Address bus must be set to the address that is to be read, the R/W signal must be set to 1, and the Memory Enable signal must be set to 1. In the
second clock cycle, the memory **Data Out** bus will contain the contents of the addressed memory location.

You are to give the design of the part of the state machine that executes the instruction above and then increments the PC by 4 in the *state transition table* given below. Assume that your machine is placed into the state **START** to begin, and you should send it to that state at the end. There are many outputs from the state machine - all the register enable signals (grouped together under ‘Enables’ in the table, the three multiplexer controls (grouped under Muxes) and two other control signals for the memory and ALU.

Note again that you need only do the execution of the instruction, and should not do the instruction fetch. You should also increment the PC by 4. Recall that all Enables are Active High. Use the following page for any rough work that you need. Do not assume that you will need all of the rows in the table that are provided, extra are given.

Answer:

<table>
<thead>
<tr>
<th>Current State</th>
<th>Next State</th>
<th>Enables</th>
<th>Muxes</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAR</td>
<td>MDR</td>
<td>R1</td>
<td>R2</td>
<td>PC</td>
</tr>
<tr>
<td><strong>START</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Page 13 of 21
Solution: any blank space is either 0 or doesn’t matter.
2 marks for each correct state.
-0.25 for mistakes until used

if two states are merged - i.e. didn’t load register, then initiate memory, -2.5 (1.5 left)
3 states merged - 5
redundant or useless states -1
PC or R1 enables active in states where they should not be -0.25
wrong states -2

<table>
<thead>
<tr>
<th>Current State</th>
<th>Next State</th>
<th>Enables</th>
<th>Muxes</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>MAR</td>
<td>MDR</td>
<td>R1</td>
</tr>
<tr>
<td>Start</td>
<td>S2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td>S3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S3</td>
<td>S4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S4</td>
<td>S5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S5</td>
<td>Start</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The DE2 Basic Computer Nios II assembly language program below responds to button presses of some of the pushbuttons. Assume that a human presses the buttons in the following order: KEY1, KEY2, KEY3, KEY2, and that the buttons are pressed 1 second apart in time.

You are to determine the contents of registers \( r_8 \) and \( r_9 \) about half a second after each button is pushed (i.e. giving the processor plenty of time to respond). Place your answers in the table on the page after next.

### Pushbutton Register Information That May Be Useful

<table>
<thead>
<tr>
<th>Address</th>
<th>31</th>
<th>30</th>
<th>...</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x10000050</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>KEY3</td>
</tr>
<tr>
<td>Unused</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x10000058</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Mask bits</td>
</tr>
<tr>
<td>0x1000005C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Edge bits</td>
</tr>
</tbody>
</table>

```assembly
.equ Pushbuttons, 0x10000050      # Pushbuttons base address
.equ INTERRUPT_MASK, 0x10000058    # Pushbuttons int mask
.equ EDGE_CAPTURE, 0x1000005C       # Pushbuttons edge capture
.equ STACK, 0x1000

.section .text
.global _start
_start:
    movia sp, STACK
    movia r15, Pushbuttons
    movi r5, 0b1010
    stw r5, 8(r15)    # Set Push Buttons Interrupt Mask
    movi r5, 0b10      # Set Processor Int. Enable Register
    wrctl ctl3, r5      # to enable interrupts from Buttons/IRQ1
    movi r5, 0b1       # Enable Processor Interrupts
    wrctl ctl10, r5     # Enable Processor Interrupts
    mov r8, r0          # r8 is used to communicate with ISR
    movi r14, 10       # r14 is used in main loop below
```
Q6, continued

MAIN_LOOP:
  blt     r8, r14, MAIN_LOOP  # keep an eye on r8
  movi r5, 0b1110              # adjust Pushbuttons interrupt mask
  stwio r5, 8(r15)
  br     MAIN_LOOP

/* Interrupt Service Routine */

_isr:
  subi ea, ea, 4              # correct the return address
  ldwio r10, 12(r15)          # Read Edge-Capture register
  andi r10, r10, 0b0010
  beq r10, r0, Not1
  movia r8, 0x5
  movia r9, 0xd1ff
  br     Done

Not1:
  ldwio r10, 12(r15)          # Read Edge-Capture register
  andi r10, r10, 0b0100
  beq r10, r0, Not2
  movia r8, 0x12
  movia r9, 0xbaff
  br     Done

Not2:
  ldwio r10, 12(r15)          # Read Edge-Capture register
  andi r10, r10, 0b1000
  beq r10, r0, Done
  movia r8, 0x15
  movia r9, 0xc00f

Done:
  stwio r0, 12(r15)           # Clear the edge-capture register
  eret
Q6, continued.

Answer: Note that ‘after’ means 0.5 seconds after the button/key is pressed - i.e. plenty of time for the processor to respond to the button press.

<table>
<thead>
<tr>
<th>Event</th>
<th>Contents of r8</th>
<th>Contents of r9</th>
</tr>
</thead>
<tbody>
<tr>
<td>After Push Button Key1 pressed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>After Push Button Key2 pressed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>After Push Button Key3 pressed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>After Push Button Key2 pressed</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Solution: (Higher marks given for line 2 of the table - keeping it the same, since interrupts on button 2 aren’t enabled until after 3 is pressed)

Line 2 nuance: 2 for realizing that key2 interrupt interrupt is disabled, and 2 for correct values. (No full marks if line 1 not equal to line 2, even if value correct for line 2)

<table>
<thead>
<tr>
<th>Event</th>
<th>Contents of r8</th>
<th>Contents of r9</th>
<th>Marks</th>
</tr>
</thead>
<tbody>
<tr>
<td>After Push Button Key1 pressed</td>
<td>5</td>
<td>d1ff</td>
<td>2</td>
</tr>
<tr>
<td>After Push Button Key2 pressed</td>
<td>5</td>
<td>d1ff</td>
<td>4</td>
</tr>
<tr>
<td>After Push Button Key3 pressed</td>
<td>0x15</td>
<td>c00f</td>
<td>2</td>
</tr>
<tr>
<td>After Push Button Key2 pressed</td>
<td>0x12</td>
<td>baff</td>
<td>2</td>
</tr>
</tbody>
</table>
Q7. Logic Optimization.

(a) [4] Determine the minimal sum-of-products expression for the following four-input logic function (in which \( D \) represents don’t care):

\[
f(x_1, x_2, x_3, x_4) = \sum m(0, 5, 7, 8, 10, 11) + \sum D(2, 4, 6)
\]

Note that the ordering of the variables to minterms is as given, i.e. the minterm 2 corresponds to \( x_1x_2x_3x_4 = 0010 \).

Use the following Karnaugh map in your solution:

\[
\begin{array}{c|cccc}
X_1X_2 & 00 & 01 & 11 & 10 \\
\hline
X_3X_4 & 00 & & & \\
& 01 & & & \\
& 11 & & & \\
& 10 & & & \\
\end{array}
\]

Answer:

Solution: \( f = x_2'x_4' + x_1x_2'x_3 + x_1'x_2 \)

4 marks for correct solution, -1 for each term that is wrong/extra.

\[
\begin{array}{c|cccc}
X_1X_2 & 00 & 01 & 11 & 10 \\
\hline
X_3X_4 & 1 & X & 1 & \\
& 1 & & & \\
& 1 & & & \\
& X & X & 1 & \\
\end{array}
\]
Q7, continued.

(b) [4] Using Boolean Algebra, prove or disprove whether the following statement is true:

\[ \bar{a}\bar{c} + \bar{a}bc + a\bar{b}\bar{c} + abc = \bar{a}\bar{c} + \bar{a}b + ab \]

Answer: First, state if the statement is True or False:

Second, provide the proof here. In your proof, show which boolean algebra rules you’re using at each step.

Solution: it is True (1 mark)

Proof: 

\[
\begin{align*}
&= a'(c' + bc) + ab'(c' + c) \\
&= a'(c' + bc) + ab \\
&= a'(c' + b) + ab \\
&= a'c' + a'b + ab' \\
\end{align*}
\]

Q.E.D.

Note: if the proof was correct, mistakes in the name of the theorem were ignored.
Extra page, intentionally left blank.