University of Toronto
Faculty of Applied Science and Engineering

Final Exam
December 2014

ECE253 – Digital and Computer Systems

Examiner – Prof. Stephen Brown

Print:

First Name ______________________ Last Name ______________________

Student Number ______________________

1. There are **7** questions and **18** pages. Do **all** questions. The duration of the exam is 2.5 hours.
2. **ALL WORK IS TO BE DONE ON THESE SHEETS.** You can use the back of the pages if you need more space. Be sure to indicate clearly if your work continues elsewhere.
3. Closed book. One 2-sided hand-written aid sheet is permitted.
4. No calculators are permitted.

1 [18]	
2 [16]	
3 [8]	
4 [8]	
5 [5]	
6 [18]	
7 [12]	
Total [85]	

1. Short answers:

[2 marks] (a) Perform the following additions of 2's complement numbers.

	i.		ii.
	0 0 1 0 1 1 0 0		1 1 1 1 1 1 1 1
	0 0 1 0 0 1 0 0		1 1 1 1 1 1 1 1
	0 0 1 0 1 0 1 1		1 1 1 1 1 1 1 1
+	0 0 1 0 1 0 1 0	+	0 1 0 1 0 1 0 1

[2 marks] (b) For the numbers in part (a) are the results you calculated correct 2's complement sums, or not?

Answer for (a) i. ______________________________

Answer for (a) ii. ______________________________

[4 marks] (c) Consider the Nios II code fragment shown below. When this code is being executed, an interrupt occurs when Nios II is executing the instruction add r1, r2, r3. Assume that interrupts are enabled, and that the interrupt is generated by the interval timer. Assume the following values for Nios II registers: r1 = 1, r2 = 2, r3 = 3.

```
        .text
        .global   _start

_start: call      somesubroutine
        movia     r15, 0x10000040
        ldw       r6, 0(r15)
        add       r1, r2, r3
        ...
```

Fill in the values that the registers listed below will have when Nios II reaches, but has not yet executed, the first instruction of the exception handler. Assume that the main program is stored in the memory starting at address 0x400.

pc ______________ **ea** ______________ **status** ______________

estatus ______________ **ipending** ______________ **r1** ______________

[2 marks] (d) In part (c) of this question, you were told that the main program is stored in the memory starting at address 0x400. Would it be okay if this main program were stored in the memory starting at address 0 instead? Explain your reasoning.

Answer ____

____

____

[2 marks] (e) Prove the following Boolean relation using algebraic manipulation in two steps, using exactly two identities. Show your work and specify which identity is used in each of your two steps.

Identity

$(x + xy)z + x\overline{z} = x$

[2 marks] (f) The following Boolean relation can be proved using algebraic manipulation in one step, using exactly one identity. Show your work and specify which identity can be used.

Identity

$((\overline{w \oplus x}) + \overline{y}) \cdot ((\overline{w \oplus x}) + z) = (\overline{w \oplus x}) + \overline{y}z$

[2 marks] (g) Prove the following Boolean relation using algebraic manipulation in two steps, using exactly two identities. Show your work and specify which identity is used in each of your two steps.

Identity

$xz + yz + x + y = x + y$

[2 marks] (h) Prove the following Boolean relation using algebraic manipulation in three steps, using exactly three identities. Show your work and specify which identity is used in each of your three steps.

Identity

$wy + xy + yz + (\overline{w} \cdot \overline{x})z = (w + x) \cdot y + (\overline{w + x}) \cdot z$

[10 marks] 2. Karnaugh maps:

$x_3x_4 \backslash x_1x_2$	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	1	0	1	1
10	1	0	1	1

(i)

$x_3x_4 \backslash x_1x_2$	00	01	11	10
00	1	1	0	1
01	0	0	0	1
11	1	0	0	0
10	1	0	1	1

(ii)

$x_3x_4 \backslash x_1x_2$	00	01	11	10
00	1	1	0	d
01	0	1	1	0
11	0	0	1	1
10	d	0	1	1

(iii)

$x_3x_4 \backslash x_1x_2$	00	01	11	10
00	0	1	1	0
01	0	0	1	1
11	0	0	1	1
10	0	1	1	0

(iv)

(a) For the function in Karnaugh map (i) above list **all minimal sum-of-products** solutions:

(b) For the function in Karnaugh map (ii) above list **all minimal product-of-sums** solutions:

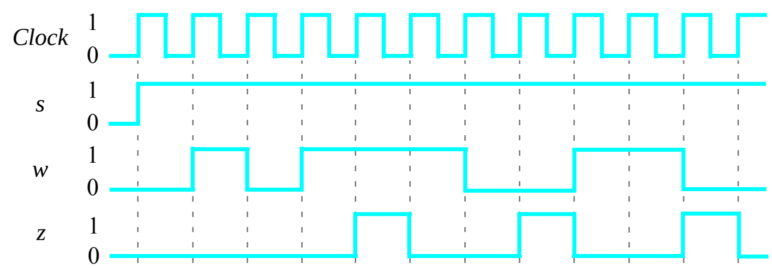
(c) For the function depicted in Karnaugh map (iii) above list **all prime implicants**:

(d) For the function depicted in Karnaugh map (iv) above, let $g = x_3 \oplus x_4$. Fill in the logic expression below. Make the simplest expression you can, using g as indicated.

$f = \qquad\qquad \cdot(g) + \qquad\qquad \cdot(\overline{g})$

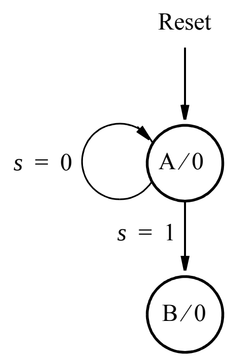
3. Finite State Machines:

[8 marks] (a) Consider a finite state machine with inputs s and w. Assume that the FSM begins in a reset state called A, as depicted below. The FSM remains in state A as long as $s = 0$, and it moves to state B when $s = 1$. Once in state B the FSM examines the value of the input w in the next three clock cycles. If $w = 1$ in exactly two of these clock cycles, then the FSM has to set an output z to 1 in the following clock cycle. Otherwise z has to be 0. The FSM continues checking w for the next three clock cycles, and so on. The timing diagram below illustrates the required values of z for different values of w.



You are to complete the state diagram below for this FSM. Use as few states as possible.

Answer:



[4 marks] (b) Given the state-assigned table shown below, draw a corresponding state diagram.

	Present state	Next state		Output
		$x = 0$	$x = 1$	z
	$y_2y_1y_0$	$Y_2Y_1Y_0$	$Y_2Y_1Y_0$	
A	000	000	001	0
B	001	001	100	0
C	010	010	001	0
D	011	001	010	1
E	100	011	100	1

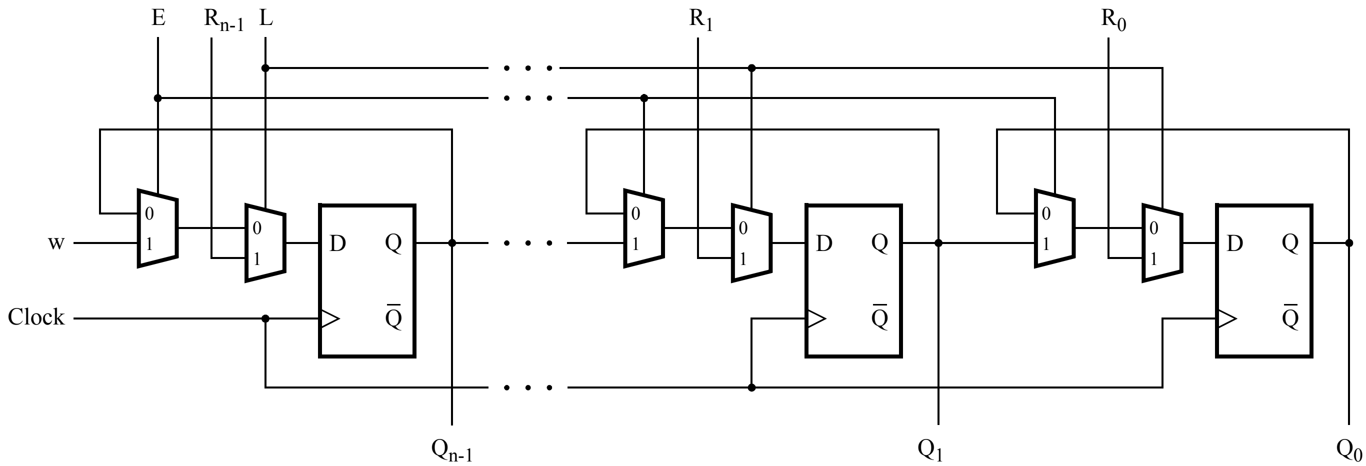
Answer:

[4 marks] (c) Synthesize minimal sum-of-products implementations of the functions Y_0 and z.

Answer:

4. Verilog Code:

Consider the n-bit shift-register circuit shown below.



[4 marks] (a) Write a Verilog module named *MUXDFF* for one stage of this circuit, including both the flip-flop and multiplexers.

Answer:

... continued on the next page

[4 marks] (b) Write a top-level Verilog module for the shift register, assuming that $n = 4$. Instantiate four copies of your *MUXDFF* subcircuit in your top-level module. Assume that you are going to implement the circuit on the DE2 board. Connect the R inputs to the *SW* switches, connect *Clock* to KEY_0, E to KEY_1, L to KEY_2 and w to KEY_3. Connect the outputs to the red lights *LEDR*.

Answer:

5. Nios II Assembly Language Code Debug:

Two implementations of the bubble sort algorithm are shown below. Both versions of the code are very similar, but the one on the left has an error that has been fixed in the implementation on the right.

```
        .text
        .global _start
_start:
        movia   sp, 0x7FFFFC
        movia   r9, LIST

BEGIN_SORT:
        ldw     r20, 0(r9)

RESTART_SORT:
        add     r18, r0, r0
        addi    r19, r0, 1
        addi    r4, r9, 4

SORT_LOOP:
        call    SWAP
        or      r18, r18, r2

        addi    r19, r19, 1
        addi    r4, r4, 4
        bne     r19, r20, SORT_LOOP

        addi    r20, r20, -1
        bne     r18, r0, RESTART_SORT

END:    br      END
```

```
        .text
        .global _start
_start:
        movia   sp, 0x7FFFFC
        movia   r9, LIST

BEGIN_SORT:
        ldw     r20, 0(r9)

RESTART_SORT:
        add     r18, r0, r0
        addi    r19, r0, 1
        addi    r4, r9, 4

SORT_LOOP:
        beq     r19, r20, END_FOR
        call    SWAP
        or      r18, r18, r2

        addi    r19, r19, 1
        addi    r4, r4, 4
        br      SORT_LOOP

END_FOR:
        addi    r20, r20, -1
        bne     r18, r0, RESTART_SORT

END:    br      END
```

[3 marks] (a) Describe the error in the code on the left, and explain how it has been fixed.

Answer ______________________________

____

____

____

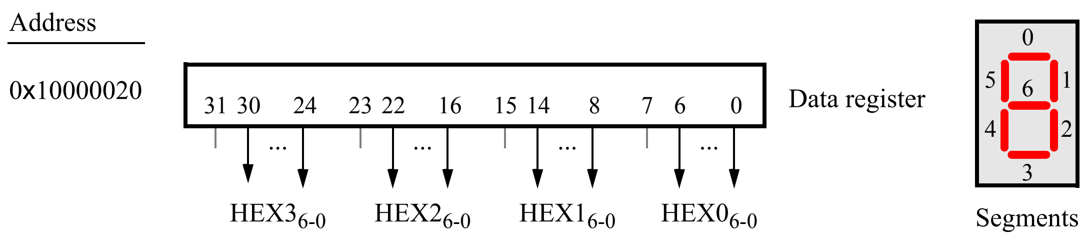
____

[2 marks] (b) Would the implementation on the left fail to sort properly for all input data? If not, then explain what property is needed in the list of data to be sorted such that the implementation on the left would give a correct result.

```
/* Swap list elements; r4 points to the first element; return 1 in r2 if swap performed */
SWAP:
        addi    sp, sp, -12
        stw     r5, 0(sp)               /* save */
        stw     r6, 4(sp)               /* save */
        stw     ra, 8(sp)               /* save */

        add     r2, r0, r0              /* initialize return value to 0 */
        ldw     r5, 0(r4)               /* get the first list element from memory */
        ldw     r6, 4(r4)               /* get the second list element */
        bgt     r5, r6, SKIP_SWAP       /* are the list elements already sorted? */

        stw     r6, 0(r4)               /* swap the list elements */
        stw     r5, 4(r4)
        addi    r2, r0, 1               /* set return value to 1 */

SKIP_SWAP:
        ldw     r5, 0(sp)               /* restore */
        ldw     r6, 4(sp)               /* restore */
        ldw     ra, 8(sp)               /* restore */
        addi    sp, sp, 12
        ret

LIST:   .word   10, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

        .end
```

[18 marks] 6. Write a Nios II Assembly Language Program:

Write an assembly language program that displays a decimal value between 0 and 3 on the seven-segment display *HEX*0 on the DE2 board. Assume that you are using the DE2 Basic Computer. In this computer system the parallel port connected to the seven-segment displays *HEX*3 − 0 is memory mapped at the address `0x10000020`. The figure below shows how the display segments are connected to the parallel port.



The number displayed on *HEX*0 should be initialized to "0". Pressing KEY_2 should increment the number (to a maximum of 3), and pressing KEY_3 should decrement the number (to a minimum of 0). Pressing KEY_1 should reset the number to 0. The parallel port connected to the pushbutton *KEYs* has the base address `0x10000050`, as illustrated below.

Address	31 30 ··· 4	3 2 1 0	
0x10000050	Unused	$KEY_{3\text{-}0}$	Data register
Unused	Unused		
0x10000058	Unused	Mask bits	Interruptmask register
0x1000005C	Unused	Edge bits	Edgecapture register

Your program has to use polled I/O to read the *Data* register in the KEY port to see when a button is being pressed. You do not need to use the *Interruptmask* or *Edgecapture* registers for this question.

The beginning part of your program is shown on the following page. Fill in the missing parts of the code. If you need more space, there is an extra lined page at the end of the exam (Page 17).

Note that the main program calls a subroutine named *SEG7_CODE*. This subroutine is passed the decimal digit between 0 and 3 and returns a bit code that can be written to *HEX*0. You are to fill in the code for *SEG7_CODE* on Page 13.

```
        .text
        .global  _start
_start:     movi    r4, 0                 # r4 holds the digital number (0 to 3)
            movia   r5, 0x10000050
            movia   r6, 0x10000020
            stw     r4, (r6)              # clear the display
LOOP:       ldw     r1, (r5)              # read the KEY port
            beq     r1, r0, DISPLAY
```

```
DISPLAY:    call    SEG7_CODE             # convert decimal number to 7-seg code
            stw     r2, (r6)              # display value on HEX display
            br      LOOP
```

```
/* Subroutine to convert the digits from 0 to 3 to bit patterns for a HEX display.
 * Parameters: r4 = the decimal value of the digit to be displayed
 * Returns: r2 = bit pattern to be written to the HEX display
 */
SEG7_CODE:
```

[12 marks] 7. Trace a Nios II Program:

Consider the Nios II program shown below. Note that the address that each instruction would have in the memory is shown to the left of the code.

```
                      .text
                      .global   _start
            _start:
00000000              movia     sp, 0x20000

00000008              ldw       r4, N(r0)          /* pass parameter in r4 */
0000000C              call      DOSUTHIN           /* result will be in r2 */
00000010              stw       r2, F(r0)

            END:
00000014              br        END                /* wait here */

            /* Do suthin', baby! */
            DOSUTHIN:
00000018              subi      sp, sp, 8
0000001C              stw       r16, 0(sp)         /* save */
00000020              stw       ra, 4(sp)          /* save return address */
00000024              mov       r16, r4

00000028              addi      r2, zero, 1
0000002C              beq       r4, r2, DIDSUTHIN

00000030              subi      r4, r4, 1
00000034              call      DOSUTHIN
00000038              mul       r2, r16, r2
            DIDSUTHIN:
0000003C              ldw       r16, 0(sp)         /* restore */
00000040              ldw       ra, 4(sp)          /* restore return address */
00000044              addi      sp, sp, 8
00000048              ret                          /* return value is in r2 */

            N:        .word     3
            F:        .word     0

                      .end
```

(a) What does this code "do"?

Answer ____

____

(b) If this program is executed on the Nios II processor, what would be the values of the Nios II registers shown below the **first** time the code reaches, but has not yet executed, the instruction `ldw r16, 0(sp)`, at the label *DIDSUTHIN*. Also, show in the space below the contents of the stack in memory at this point in time (fill in the memory addresses on the left, and show the data stored in each location). For memory values that are not known, if any, write N/A in the corresponding box.

pc		ra		sp	
r2		r4		r16	

Memory Address	Content
1FFFC	
20000	

Extra answer space for any question on the test, if needed:

Extra answer space for writing assembly language code, if needed:

Boolean Identities

10*a*.	$x \cdot y = y \cdot x$	*Commutative*
10*b*.	$x + y = y + x$	
11*a*.	$x \cdot (y \cdot z) = (x \cdot y) \cdot z$	*Associative*
11*b*.	$x + (y + z) = (x + y) + z$	
12*a*.	$x \cdot (y + z) = x \cdot y + x \cdot z$	*Distributive*
13*a*.	$x + x \cdot y = x$	*Absorption*
14*a*.	$x \cdot y + x \cdot \overline{y} = x$	*Combining*
15*a*.	$\overline{x \cdot y} = \overline{x} + \overline{y}$	*DeMorgan's theorem*
16*a*.	$x + \overline{x} \cdot y = x + y$	
17*a*.	$x \cdot y + y \cdot z + \overline{x} \cdot z = x \cdot y + \overline{x} \cdot z$	*Consensus*