University of Toronto
Faculty of Applied Science and Engineering

Final Exam
December 2016

ECE253 – Digital and Computer Systems

Examiner – Prof. Stephen Brown

Print:

First Name ______________________ Last Name ______________________

Student Number ______________________

1. There are **7** questions and **24** 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 blank pages included at the end of the exam (Pages 20 – 22) if you need more space. Be sure to indicate clearly if your work continues elsewhere.
3. Closed book. One 2-sided aid sheet is permitted.
4. No calculators are permitted.

1 [15]	
2 [10]	
3 [10]	
4 [12]	
5 [12]	
6 [11]	
7 [15]	
Total [85]	

[15 marks] 1. Short answers:

[2 marks] (a) Perform the following number-base conversions. Assume that all numbers are represented using eight bits. If it is not possible to perform a conversion write *NaN* as the result.

i. $(-25)_{10}$ to signed binary (2's complement)

Answer ....

ii. $(240)_{10}$ to signed binary (2's complement)

Answer ....

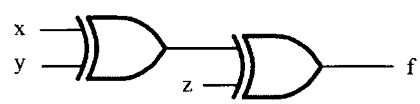
iii. $(-240)_{10}$ to signed binary (2's complement)

Answer ....

iv. $(100)_{10}$ to signed binary (2's complement)

Answer ....

[2 mark] (b) Consider the circuit shown below. Give a sum-of-products implementation of f.



Answer

[2 marks] (c) Draw a circuit that can be used to multiply two one-bit numbers, $p = a \times b$.

[2 marks] (d) Verilog code is similar to C code. True or false? Briefly explain your answer.

Answer ____

____

____

____

____

____

____

____

[3 marks] (e) Consider the ARM code fragment shown below. When this code is being executed, an interrupt occurs when the ARM processor is executing the instruction CMP R1, R3. Assume that interrupts are enabled, and that the interrupt is generated by the timer that you used in Lab Exercise 10. Assume the following values for ARM registers: R1 = 1, R2 = 2, R3 = 3.

```
        .text
        .global   _start

_start: BL        DOSUTHIN
        LDR       R10, =0xFF200040
        LDR       R6, [R10]
        CMP       R1, R3
        ...
```

Fill in the values that the registers listed below will have when the ARM processor reaches, but has not yet executed, the branch instruction at the IRQ exception vector. Assume that the main program is stored in the memory starting at address 0x20.

PC ____________________ **LR** ____________________ **R1** ____________________

[2 marks] (f) What is the purpose of the ARM processor's exception vector table, and what is usually stored there in a typical usage scenario?

Answer ____

____

____

____

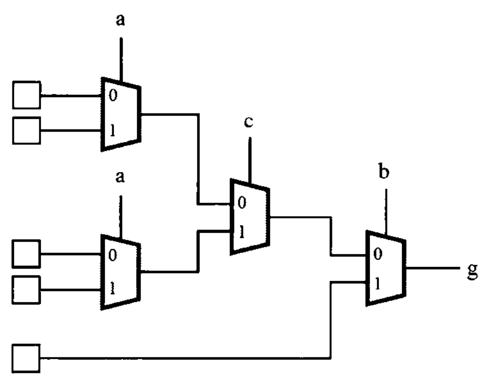
[2 marks] (g) Use Boolean algebra to minimize the following function. Show your work and specify which identity is used in each of your steps.

Identity

$x_1\overline{x}_2\overline{x}_3 + x_1x_2 + \overline{x}_1x_2x_3$

[10 marks] 2. Multiplexers :

(a) Consider the function $g = b + a \oplus c$. You are to implement this function using the circuit structure shown below, by filling in 0 or 1 in each of the boxes that are selected by the 2-to-1 multiplexers.



(b) Implement the following logic functions using only 2-to-1 multiplexers.

i. $f = x_1 x_2 + x_1 x_3$

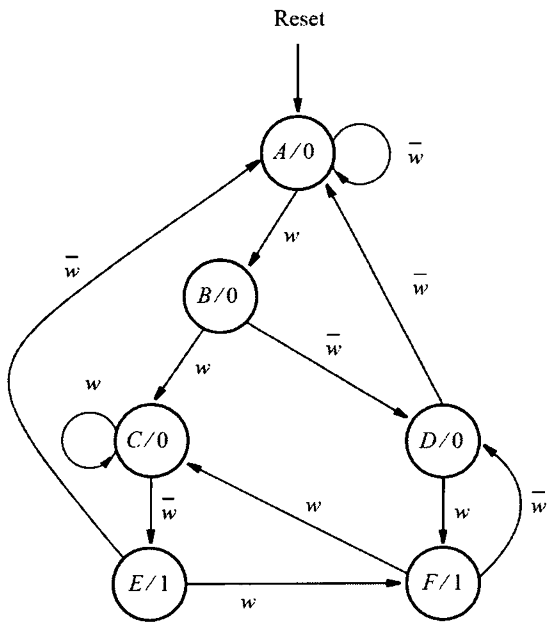
ii. $g = \overline{(x_1 x_2)} + x_3$

iii. Using a D flip-flop, a 4-to-1 multiplexer, and any other needed gates, design a positive-edge-triggered *JK* flip-flop. The *JK* flip-flop has two data inputs *J* and *K*, and behaves as follows at the rising clock edge:

J K	Behaviour
0 0	The flip-flop retains its state
0 1	The flip-flop is reset to 0
1 0	The flip-flop is set to 1
1 1	The flip-flop output toggles

[10 marks] 3. Finite State Machines:

Consider the finite state machine below, with input w and output z.



(a) Using the state assignment

$$y_6y_5y_4y_3y_2y_1 = 000001(A), 000010(B), 000100(C), 001000(D), 010000(E), 100000(F)$$

derive the expressions below:

$Y_1 =$ ____

$Y_5 =$ ____

$z =$ ____

(b) For the next part, using the state assignment

$$y_5y_4y_3y_2y_1 = 000(A), 001(B), 010(C), 011(D), 100(E), 101(F)$$

fill in the state-assigned table on the following page.

State-assigned table:

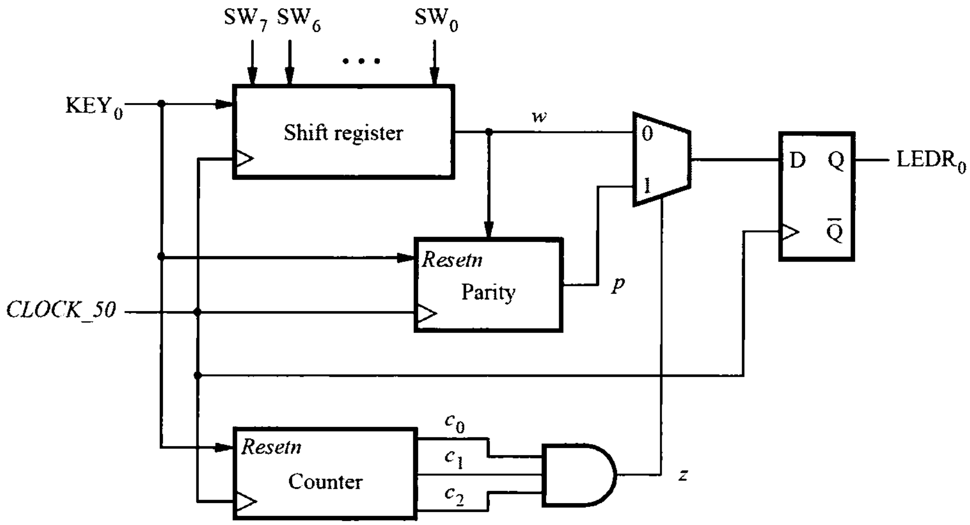
	Present state $y_3y_2y_1$	Next state $w = 0$ $Y_3Y_2Y_1$	Next state $w = 1$ $Y_3Y_2Y_1$	Output z
A	000			
B	001			
C	010			
D	011			
E	100			
F	101			

In the space below use K-maps to derive minimal sum-of-products expressions for Y_2 and z.

Answer

[12 marks] 4. Verilog code:

Consider the circuit shown below.



Verilog modules for the *shift register* and *parity* subcircuits are shown below.

```
module shift (input Ln, Clock, input [7:0] R, output y);
   reg [7:0] Q;
   always @ (posedge Clock)
   if (Ln == 0)
      Q <= R;
   else
   begin
      Q[7] <= 1'b0;
      Q[6:0] <= Q[7:1];
   end
   assign y = Q[0];
endmodule

module parity (input w, Clock, Resetn, output p);
   reg Q;
   always @ (posedge Clock)
      if (Resetn == 0)
         Q <= 1'b0;
      else
         Q <= Q ^ w;
   assign p = Q;
endmodule
```

On the following page you are to write Verilog modules for the counter, and for the top-level circuit.

Provide a Verilog module for the counter in the space below. It is just an up-counter.

Answer

Provide Verilog code for the top-level module. Include the *shift register*, *parity*, and *counter* modules as subcircuits, and write Verilog code for the rest of the circuit elements. Additional space is provided on the next page.

Answer

... Additional Space for the Verilog Top-level Module

[12 marks] 5. Trace an ARM Program:

Consider the ARM code 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                    LDR       SP, =0x20000

00000004                    LDR       R4, =X
00000008                    LDR       R0, [R4], #4
0000000C                    LDR       R1, [R4], #4
00000010                    BL        BINGO
00000014                    STR       R0, [R4]

00000018          END:      B         END                  /* wait here */

0000001C          BINGO:    PUSH      {R3, LR}
00000020                    MOV       R3, R0
00000024                    CMP       R1, R0
00000028                    BLT       BONGO

0000002C          BANGO:    SUB       R1, R1, R3
00000030                    BL        BINGO
00000034                    MOV       R1, R0

00000038          BONGO:    MOV       R0, R1
0000003C                    POP       {R3, PC}

                  X:        .word     2
                  Y:        .word     5
                  M:        .space    4

                            .end
```

(a) What does this code "produce"? That is, what is the relationship between X, Y and M?

Answer ____

____

(b) If this program is executed on the ARM processor, what would be the values of the ARM registers shown below the **first** time the code reaches, but has not yet executed, the instruction at address 0x34. 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.

R0		R1		R3	
R13		R14		R15	

Memory Address	Content
1FFFC	
20000	

[11 marks] 6. Assembly Language Subroutines:

Consider the ARM program shown below. This program executes in an endless loop that calls two subroutines: `PARSE` and `READ_KEY`.

You are to write the `PARSE` and `READ_KEY` subroutines. The `PARSE` subroutine is supposed to examine the machine code passed to it in register *R0* and identify which instruction the machine code represents. As shown in the code, the possible instructions that `PARSE` may be passed are AND, ORR, EOR, and B. `PARSE` has to indicate which instruction it identifies by writing to the HEX2-0 displays on a DE1-SoC board. The `PARSE` subroutine is discussed further on the following page.

The `READ_KEY` subroutine reads from the pushbutton KEYs port on the DE1-SoC board, and waits for any KEY to be pressed. After a KEY is pressed, the subroutine returns to the main program. The purpose of `READ_KEY` is to wait for the user to press a KEY, and then to return.

```
            .global     _start
_start:     LDR         SP, =0x3FFFFFFC

TOP:        LDR         R5, =INST_LABEL
            LDR         R0, [R5], #4
            BL          PARSE
            BL          READ_KEY

            LDR         R0, [R5], #4
            BL          PARSE
            BL          READ_KEY

            LDR         R0, [R5], #4
            BL          PARSE
            BL          READ_KEY

            LDR         R0, [R5], #4
            BL          PARSE
            BL          READ_KEY
            B           TOP

/* Code below this line is not executed */
INST_LABEL: AND         R0, R1, R2
            ORR         R1, R2, R3
            EOR         R2, R3, R4
            B           INST_LABEL
/* Code above this line is not executed */
            .end
```

(a) In the space on the following page, write the code for the `READ_KEY` subroutine. The registers in the pushbutton KEY port are shown at the end of this exam.

Answer Space for READ_KEY

(b) The PARSE subroutine that is called by the main program should display the following on HEX2-0: And for AND, Orr for ORR, Eor for EOR, and --- for B. The displays should appear as shown below.

And Orr Eor ---

The registers in the HEX display are shown at the end of this exam. Since the main program runs in an endless loop it should produce the following sequence of outputs on HEX2-0:

```
And
Orr
Eor
---
And
...
```

You will need to know the format of the logic instruction machine code, which is:

31 28	27 26	25	24 21	20	19 16	15 12	11 0
Cond	00	I	OpCode	S	Rn	Rd	Operand 2

For purposes of this question your code needs to examine only bits 21 − 24, which are called the *Opcode*. This field specifies the type of instruction: it has the values 0000 for AND, 0001 for EOR and 1100 for ORR.

Write the code for the PARSE subroutine in the space on the following page.

Answer Space for PARSE

[15 marks] 7. Exceptions :

Consider the code shown below, which sets up an exception vector table for interrupts. The main program has to first set up the stack pointers and enable interrupts for the pushbutton KEYs. To configure the GIC, assume that you are given a subroutine named `CONFIG_GIC`. The CPSR register and ARM mode bits are shown at the end of the exam. **You are to fill in the missing code below.**

```
            .section .vectors, "ax"
            B        _start          // reset vector
            .word    0               // undefined instruction vector
            .word    0               // software interrrupt vector
            .word    0               // aborted prefetch vector
            .word    0               // aborted data vector
            .word    0               // unused vector
            B        SERVICE_IRQ     // IRQ interrupt vector
            .word    0               // FIQ interrupt vector

            .text
            .global  _start
_start:














MAIN_LOOP:  AND      R0, R1, R2
            EOR      R3, R4, R5
            ORR      R6, R7, R8
            B        MAIN_LOOP       // main program simply repeats the loop
```

After setting up the stacks and enabling interrupts the main program enter an endless loop. The purpose of the program is as follows: when a user presses a pushbutton KEY, the corresponding interrupt service routine should display on HEX2-0 which instuction in the endless loop will be executed *next* when the interrupt service routine returns to the main program.

Complete the `SERVICE_IRQ` code below. It has to check if the KEYs port caused the interrupt. If so, it should call a subroutine named `KEYS_ISR` to handle the interrupt. The KEYs port uses interrupt ID number 73. The `SERVICE_IRQ` code has to pass to `KEYS_ISR` the machine code, in register *R0*, of the *next* instruction that will be executed in the main program on return from the interrupt.

You are to fill in the missing code below.

```
                .global   SERVICE_SVC
SERVICE_IRQ:    PUSH      {R0-R7, LR}
                /* Read the interrupt ID from the ICCIAR in the GIC */
                LDR       R4, =0xFFFEC100
                LDR       R5, [R4, #0xC]    // read interrupt ID

CHECK_KEYS:







                BL        KEY_ISR           // pass R0 as a parameter to KEY_ISR
EXIT_IRQ:       /* Write to the End of Interrupt Register (ICCEOIR) in the GIC */
                STR       R5, [R4, #0x10]   // write to ICCEOIR



                SUBS      PC, LR, #4
```

Write the `KEYS_ISR` subroutine on the following page. To control HEX2-0 you can simply call the `PARSE` subroutine that you wrote in Question 6. An example of output that could be produced if the main program were executed and the user pressed a few KEYs might be:

```
Orr
And
Eor
---
Orr
...
```

```
        .global    KEYS_ISR
KEYS_ISR:
```

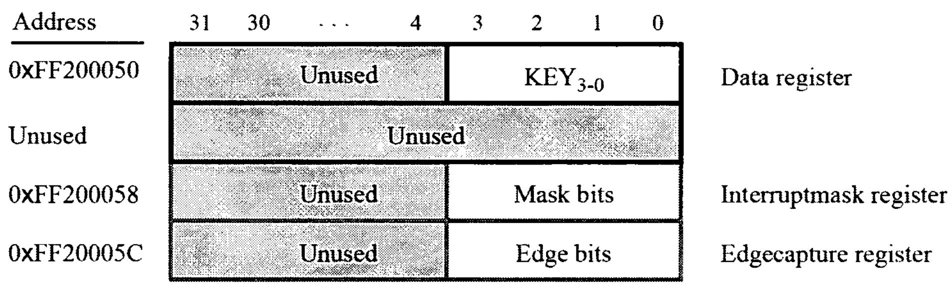
Extra answer space for any question on the test, if needed:

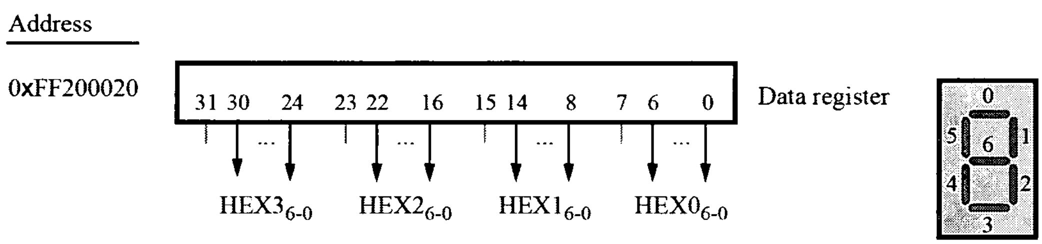
Extra answer space for any question on the test, if needed:

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

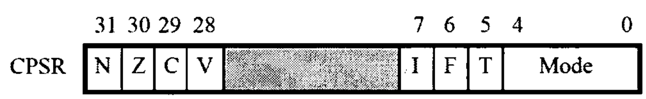
Boolean Identities

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*





Segments



User Mode: 10000
FIQ Mode: 10001
IRQ Mode: 10010
Supervisor Mode: 10011
Abort Mode: 10111
Undefined Mode: 11011

THIS PAGE INTENTIONALLY LEFT BLANK