UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING

ECE253F – Digital and Computer Systems
Final Examination

December 15, 2021 2:00pm - 4:30pm
Duration: 150 minutes

Examiners: Profs. N. Enright Jerger and J. Anderson

Please enter your name and student number in the spaces provided above as it appears on Quercus. It is important that your name exactly match the Quercus gradebook.

Exam Type D: Examiner specified aids: One single sheet of letter size paper (8.5 x 11 inch), both sides may be used.

Calculator Type 4: No calculators or other electronic devices are allowed.

All questions are to be answered on the examination paper. Your answer **MUST** be fully contained on the same page as the question. **Any material written on the back of each page will be ignored.**. Exams will be scanned – please write clearly in pen or dark pencil.

Please state any assumptions you make when answering a question.

The number of marks for each question are indicated. The exam has **19 pages**, including this one.

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Total
6	3	6	8	12	5	7	4	8	8	12	79

Question 1 [6 Marks]

Fill in the following table with the appropriate number conversions. If there is no possible answer, explain why.

Decimal	8-bit 2's complement	8-bit unsigned binary
103	1) **0110 0111**	2) **0110 0111**
3) **-70**	**1011 1010**	4) **can't represent negative number**
5) **212**	6) **Number is too large**	**1101 0100**

You may use the space below for your calculations. Please indicate which part you are solving by writing the corresponding number from the boxes above.

1 point for each answer

Question 2 [3 Marks]

Consider the following k-map:

cd \ ab	00	01	11	10
00	1	1	0	1
01	0	1	0	0
11	1	0	1	1
10	1	0	1	X

[2 marks] a) Using product terms, list all of the essential prime implicants.

$\overline{a}b\overline{c}, \overline{b}\,\overline{d}, \overline{b}c, ac$

0.5 for each

[1 marks] b) Derive the minimum cost SOP (sum-of-products) expression.

$f = \overline{a}b\overline{c} + \overline{b}\,\overline{d} + \overline{b}c + ac$

deduct 0.25 for each mistake

Question 3 [6 Marks]

Consider the following sequence of ARM assembly code:

```
   .data
XYZ: .word 0, 0xFFABCDFF, 1, PTR, 2
PTR: .word -2, -4
   .text
   .global _start
_start:
   LDR R0, =XYZ
   LDR R1, [R0, #4]
   LDR R2, [R0, #12]
   LDR R2, [R2]
   AND R2, R2, #0xFF
   CMP R2, #0
   BGE ABC
   LDR R3, =0x008945FF
   EOR R4, R1, R3
   LDR R5, [R0, #8]
   LSL R5, R5, #5
   B END
ABC: MOV R3, #0xF
   STR R3, [R0]
   LSL R3, #16
   AND R4, R1, R3
   ASR R5, R1, #4
END: B END
```

In the table on the next page, give the value in hexadecimal in each of the listed registers when the processor reaches the instruction at label END. Your answer should include all 8 hexadecimal digits stored in the register. If the value cannot be determined from the sequence of instructions, write ”UNKNOWN”.

R1	0xFFAB CDFF	**R4**	0x000B 0000
R2	0x0000 00FE	**R5**	0xFFFA BCDF
R3	0x000F 0000	**LR**	UNKNOWN

1 pt for each answer

Question 4 [8 Marks]

Write an ARM assembly language subroutine that implements restoring division. This subroutine takes 2 arguments: Divisor (R0) and Dividend (R1). It will return its results of Quotient in R0 and Remainder in R1. You can assume the stack pointer has been initialized for you to be 0x20000. You can assume the Divisor and Dividend are unsigned positive numbers. Comment your code for full marks.

The restoring division algorithm is as follows:

1) Initialize the Remainder to 0.
2) Shift the Remainder to the left by 1 bit. Set the least significant bit of the Remainder to be equal to the most significant bit of the Dividend. Shift the Dividend to the left by 1 bit.
3) Subtract the Divisor from the Remainder to get the new Remainder.
4) Test the new Remainder:
    - If positive, shift a 1 in the least significant bit of the Quotient.
    - If negative, shift a 0 in the least significant bit of the Quotient and add the Divisor back to the Remainder to "restore" it back to its original value.
5) Repeat from step 2, until all bits in the Dividend are consumed.

```
DIVIDE: PUSH {R4,R5} // save r4, r5 before use
        MOV R2, #0 //R2 is remainder
        MOV R4, #0 // R4 is quotient
LOOP:   CMP R5, #32 // All bits of dividend consumed?
        BEQ RETURN
        ADD R5, #1
        LSL R2, #1
        LSR R3, R1, #31 // MSb of dividend
        ORR R2, R2, R3
        LSL R1, #1
        SUB R2, R2, R0
        CMP R2, #0
        LSL R4, #1 // shift quotient by 1
        BGE ONE
        ADD R2, R2, R0 // if negative, restore remainder
        B LOOP
ONE:    ORR R4, #1 // if remainder positive, put 1 in LSb
        B LOOP
RETURN: MOV R1, R2 // move remainder into R1
        MOV R0, R4 // move quotient into R2
        POP {R4,R5} // restore r4, r5 before return
        MOV PC, LR
```

Question 4 continued . . .

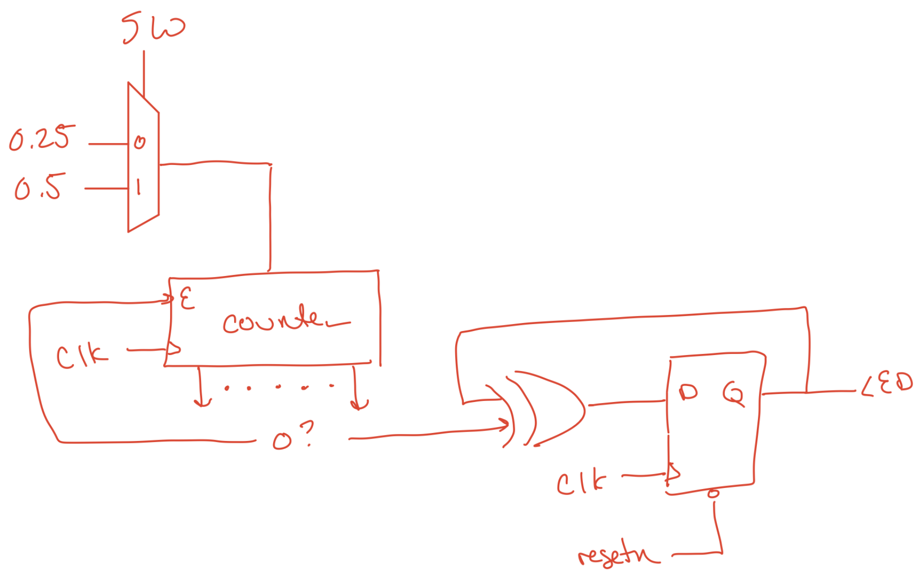
1 point for saving/restoring registers
1 point for returning values in correct registers
1 point for MOV PC, LR
1 point for consuming all bits of dividend
1 point for correct shifting of MSb of dividend into remainder
1 point for subtracting divisor from remainder
1 point for testing positive/negative of new remainder
1 point for correctly shifting 0 or 1 into quotient

Question 5 [12 Marks]

Consider a system that has the following hardware functionality:

- If the input Switch is set to 0, it blinks an LED on and off with a period of 0.5s. The off time is 0.25s and the on time is 0.25s
- If the input Switch is set to 1, it blinks an LED on and off with a period of 1s. The off time is 0.5s and the on time is 0.5s
- On a reset, the LED should be off and restart the pattern according to the value of the switch.

[4 marks] a) Draw a schematic to implement the described functionality.



1 mark for mux/switch

1 mark for counter

1 mark for updating LED

1 mark for correct connections

[8 marks] b) Write a Verilog module to implement the functionality described above.

The top-level Verilog module is given below. Clock_10 is a 10MHz clock input. Resetn is an active low asynchronous reset. You may write other modules as required. You may not assume you have any other modules available to you.

```
    module blinking_light(clock_10, resetn, switch, LED);
      input clock_10, resetn, switch;
      output LED;


wire [22:0] quarter, half, mux_out;
reg dff;
reg [22:0] counter;

assign quarter = 2500000;
assign half = 5000000;
mux2to1 u1 (quarter, half, switch, mux_out);

always@(posedge clock_10, negedge resetn)
begin
    if (!resetn)
        counter <= mux_out;
        dff <= 0;
    else
    begin
        if (counter == 0)
            counter <= mux_out;
            dff <= dff ^ 1;
        else
            counter <= counter-1;
    end
end

assign LED = dff;
endmodule
```

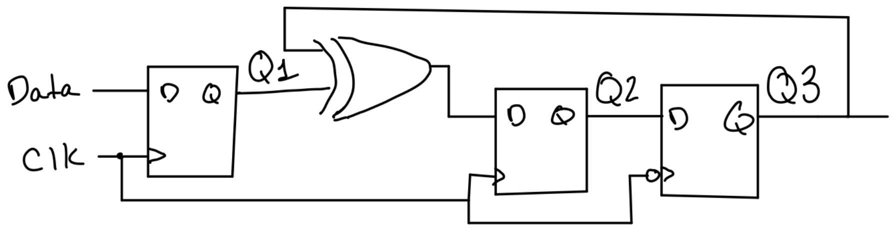
Question 5 continued . . .

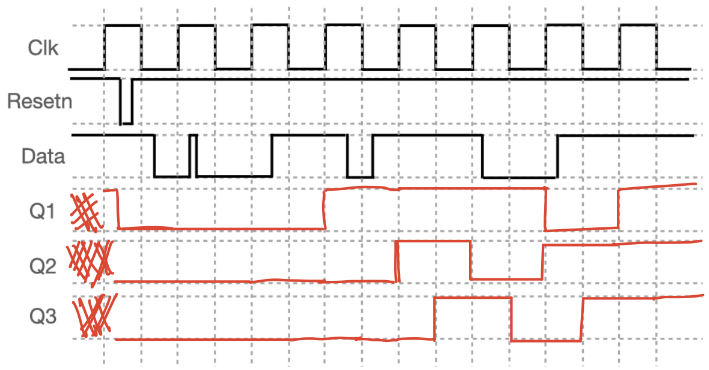
```
module mux2to1 (x, y, sel, out);
    input [22:0] x, y;
    input sel;
    output reg [22:0];

    always@(*)
    begin
        if (sel)
            out = y;
        else
            out = x;
    end
endmodule
```

1 mark for mux
1 mark for correct sensitivity list
1 mark for correct reset
1 mark for correct counter update
1 mark for correct counter values
1 mark for correct LED assignment
1 mark for correct <= assignment in always block
1 mark for flip flop to hold LED value

Question 6 [5 Marks]

Given the following circuit and input waveforms for Clk, Data and Resetn, draw the corresponding output waveforms. Each flip-flop has an active low, asynchronous reset.

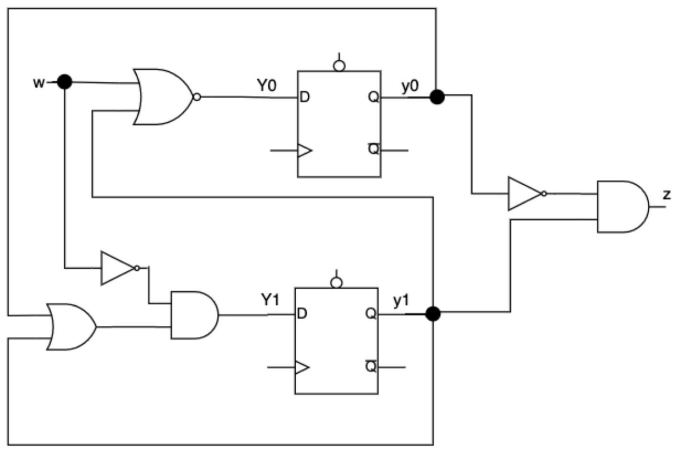




1 mark for reset
2 marks for updates on posedge (Q1, Q2)
1 mark for updates on negedge (Q3)
1 mark for XOR

Question 7 [7 Marks]

[6 marks] a) The circuit below is a sequence-recognizer FSM that detects a specific sequence of 3 bits on input w. Output z is set to 1 when the sequence is detected. For clarity, the clock and reset is not shown in the figure.



Give the state-assigned table for the FSM:

	Curr state		Next state w=0		Next state w=1		Output
	y1	y0	Y1	Y0	Y1	Y0	z
	0	0	0	1	0	0	0
	0	1	1	1	0	0	0
	1	0	1	0	0	0	1
	1	1	1	0	0	0	0

$\underbrace{\quad\quad}_{1\text{ mark}}$ $\underbrace{\quad}_{1}$ $\underbrace{\quad}_{1}$ $\underbrace{\quad}_{1}$ $\underbrace{\quad}_{1}$ $\underbrace{\quad}_{1}$

[1 mark] b) What sequence is detected by the FSM?

0, 0, 0

1 mark

Question 8 [4 Marks]

The logic equation for the carry-out function in a ripple-carry adder is: $C_{out} = xy + xC_{in} + yC_{in}$. Use Boolean algebra to prove this is equivalent to $C_{out} = xy + (x \oplus y)C_{in}$.

$C_{out} = xy + (x \oplus y)C_{in}$

1 mark $= xy + x\bar{y}C_{in} + \bar{x}yC_{in}$ // distrib

1 mark $= xy\underbrace{(C_{in} + \overline{C_{in}})}_{=1} + x\bar{y}C_{in} + \bar{x}yC_{in}$

1 mark $= xyC_{in} + xy\overline{C_{in}} + x\bar{y}C_{in} + \bar{x}yC_{in}$

$= xC_{in}(y + \bar{y}) + xy(C_{in} + \overline{C_{in}})$

1 mark $+ yC_{in}(x + \bar{x})$

$= xC_{in} + xy + yC_{in}$

Question 9 [8 Marks]

You are to design a leading-one finder circuit. This circuit has many applications in digital logic and computer arithmetic.

Input A_{3-0} is a 4-bit binary input. Output Z_{2-0} is a binary output that represents the position of the (most-significant '1' in input A) + 1. For example, if $A = 0010$, then $Z = 010$, since A's most-significant '1' is in the 1st position, and $1 + 1 = 2$ (010 in binary). If $A = 0001$, then $Z = 001$. As another example, if $A = 1001$, then $Z = 100$, since A's most-significant '1' is in the 3rd position. If $A = 0000$, then $Z = 000$.

[4 marks] a) Give the truth table for the leading-one finder circuit.

A_3	A_2	A_1	A_0	Z_2	Z_1	Z_0	
0	0	0	0	0	0	0	0.5
0	0	0	1	0	0	1	0.5
0	0	1	0	0	1	0	1 mark
0	0	1	1	0	1	0	
0	1	0	0	0	1	1	1 mark
0	1	0	1	0	1	1	
0	1	1	0	0	1	1	
0	1	1	1	0	1	1	
1	0	0	0	1	0	0	1 mark
1	0	0	1	1	0	0	
1	0	1	0	1	0	0	
1	0	1	1	1	0	0	
1	1	0	0	1	0	0	
1	1	0	1	1	0	0	
1	1	1	0	1	0	0	
1	1	1	1	1	0	0	

Question 9 continued . . .

[4 marks] b) Derive the minimized logic equations for Z_{2-0}.

$Z_2 = A_3$ 1 mark

$Z_1 = \overline{A_3}(A_2 + A_1)$ 1 mark

$Z_0 = \overline{A_3}(A_2 + \overline{A_1}A_0)$

$= \underbrace{\overline{A_3}A_2}_{1\ \text{mark}} + \underbrace{\overline{A_3}\,\overline{A_1}A_0}_{1\ \text{mark}}$

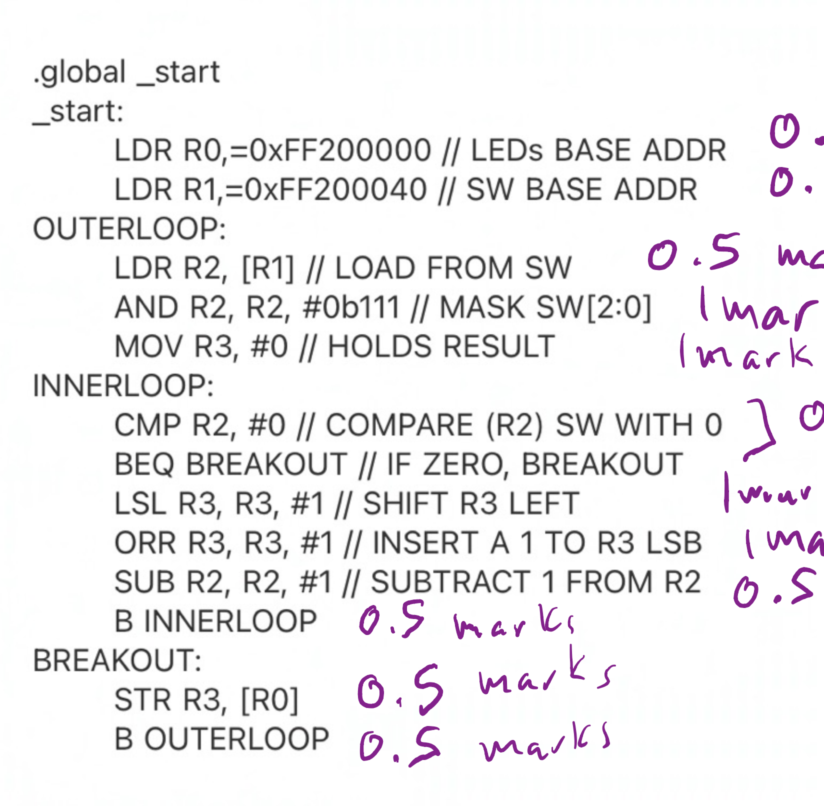
Question 10 [8 Marks]

You are to write an ARM assembly language program that converts a 3-bit binary code to 7-bit thermometer code. The table illustrates the conversion:

3-bit binary	7-bit thermometer
000	0000000
001	0000001
010	0000011
011	0000111
100	0001111
...	
111	1111111

The 3-bit binary code is input on SW_{2-0}. The 7-bit thermometer code must be output to $LEDR_{6-0}$. Your program should use memory-mapped I/O to load/store from the switches and red LEDs. Your program should use a loop to determine the thermometer code. No marks will be given for solutions written in the style of "if-else if-else if-else if-..."

Finally, as the user adjusts the SW inputs, the program should update the thermometer code output accordingly to reflect each new input. Meaning that, your program should use *polling* on the SW inputs to detect any changes, and then update the thermometer code accordingly. On the DE1-SoC system, $LEDR$ are mapped at address: 0xFF200000; SW are mapped at address: 0xFF200040.



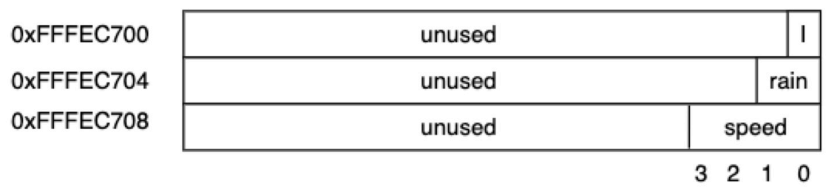
```
.global _start
_start:
        LDR R0,=0xFF200000 // LEDs BASE ADDR     0.5 marks
        LDR R1,=0xFF200040 // SW BASE ADDR       0.5 mark
OUTERLOOP:
        LDR R2, [R1] // LOAD FROM SW        0.5 marks
        AND R2, R2, #0b111 // MASK SW[2:0]   1mark
        MOV R3, #0 // HOLDS RESULT          1mark
INNERLOOP:
        CMP R2, #0 // COMPARE (R2) SW WITH 0   ] 0.5
        BEQ BREAKOUT // IF ZERO, BREAKOUT      ]
        LSL R3, R3, #1 // SHIFT R3 LEFT         1mark
        ORR R3, R3, #1 // INSERT A 1 TO R3 LSB   1mark
        SUB R2, R2, #1 // SUBTRACT 1 FROM R2     0.5 marks
        B INNERLOOP   0.5 marks
BREAKOUT:
        STR R3, [R0]   0.5 marks
        B OUTERLOOP    0.5 marks
```

Question 10 continued . . .

Question 11 [12 Marks]

You are to write an ARM assembly program that uses interrupts in an automotive application for automatic rain-detecting windshield wipers. When rain is detected, an interrupt will be generated. The heaviness of the detected rain will be represented by a 2-bit value. When the interrupt happens, your program should convert the rain strength to a 4-bit wiper speed by *multiplying* the rain strength by 4. Your program should then write the computed wiper speed to a memory-mapped address that controls the wipers.

The rain-detection and wiper hardware is accessed through the following memory address map. Bit I must be set to '1' to enable interrupts. When an interrupt occurs, bits "rain" can be read to detect the rain strength. Storing any value to these rain bits *clears* the interrupt. Bits "speed" control the wiper speed, as discussed above.



The IRQ ID for the rain-detection hardware is 88. The mode-code for supervisor mode is (0b10011); the code for IRQ mode is (0b10010). Bits 4-0 of the CPSR contain the mode; bit 7 is the IRQ mask bit.

[3 marks] a) Write the code to enable interrupts for the rain-detection hardware, to set up SP for the relevant ARM modes, and to enable interrupts in the CPSR. Use address 0x10000 as the SP for supervisor mode; use address 0x20000 as the SP for IRQ mode. Comment your code for full marks.

```
MOV  R0, #0b10010   // IRQ mode          ] 0.5
MSR  CPSR, R0                            ] 0.5
LDR  SP, #0X20000   // SP for IRQ
MOV  R0, #0b10011   // SUPERVISOR        ] 0.5
MSR  CPSR, R0
LDR  SP, #0X10000   // SP for SVC        ] 0.5
LDR  R0, =0XFFFEC700 // base add         ] 0.5
MOV  R1, #1
STR  R1, [R0]   // enable inter. in rain HW
MOV  R0, #b 00010011                     ] 0.5
MSR  CPSR, R0  // enable inter in ARM
```

Question 11 continued . . .

[3 marks] b) Write the section of the SERVICE IRQ subroutine to determine which device is interrupting. Your code should call the WIPER_ISR subroutine if rain-detection hardware is triggering the interrupt, and behave appropriately otherwise. You will be asked to write the WIPER_ISR subroutine in part (c). Comment your code for full marks.

```
SERVICE_IRQ:
  PUSH RO-R5, LR
  LDR R4, =MPCORE_GIC CPUIF
  LDR R5, [R4, #ICCIAR] // read the interrupt ID
  // WRITE YOUR CODE HERE
1 mark  CMP R5, #88  // check ID if 88
1 mark  BNE    ENDIRQ  // if !88 exit
0.5     BL WIPER_ISR  // call subroutine

0.5 ENDIRQ:
  STR R5, [R4, #ICCEOIR]
  POP RO-R5, LR
  SUBS PC, LR, #4
```

Question 11 continued . . .

[6 marks] c) Write the WIPER_ISR subroutine. Comment your code for full marks.

```
WIPER_ISR:
    LDR R0, =0xFFFEC700   // base addr          1 mark
    LDR R1, [R0, #4]      // load Rain bits     1 mark
    STR R1, [R0, #4]      // clear interrupt    1 mark
    LSR R1, #2  // *=4                          1 mark
    STR R1, [R0, #8]      // store to Speed bits 1
    MOV PC, LR  // return                       1 mark
```