

**UNIVERSITY OF TORONTO  
FACULTY OF APPLIED SCIENCE AND ENGINEERING**

**ECE253F – Digital and Computer Systems  
Final Examination**

**December 12, 2019 2:00pm - 4:30pm  
Duration: 150 minutes**

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

**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. There is one extra page at the end and you may use the back of a page. If you use more than the given space, please direct the marker to the appropriate page and indicate clearly on that page which question(s) you are answering there. It is your responsibility to make sure the marker can find your solution.

The number of marks for each question are indicated.

The examination has **18 pages**, including this one.

Last Name: \_\_\_\_\_ First Name: \_\_\_\_\_

Student Number: \_\_\_\_\_ UTORID: \_\_\_\_\_

This page is only for marking purposes.

**MARKS**

1	2	3	4	5	6	7	8	9	10	Total
/6	/4	/6	/6	/9	/12	/7	/8	/16	/10	/84

**Question 1 [6 Marks]**

Fill in the following table with the appropriate number conversions:

8-bit 2's complement	decimal	hexadecimal
11001110	-50	C E
1100 0111	-57	C 7
0101 1001	89	5B

1 mark each

**Question 2 [4 Marks]**

Consider the following ARM assembly language program:

```
.text
.global _start
_start:
    LDR R1, =LIST
    MOV R2, #0x0
    MOV R4, R2
LOOP:
    ADD R5, R1, R2
    LDR R3, [R5]
    ADD R4, R4, R3
    CMP R3, #0
    BEQ END
    BLT END
    ADD R2, R2, #0x4
    B LOOP
END:
    B END
LIST:
    .word 5, 8, 9, -2, 6, -1, 0
.end
```

State the values stored in registers R2, R3 and R4 after the execution of the program (i.e. when the program reaches the "END: B END" instruction).

marks

1 | R2 = XC  $(12)_{10}$

2 | R3 = XF FFFF FFFE  $(-2)_{10}$

| R4 = X14  $(20)_{10}$

**Question 3 [6 Marks]**

Consider the following state table for an FSM with input  $w$  and output  $z$ .

mark scheme  
in red.

Curr State	Next State		Output $z$
	$w=0$	$w=1$	
A	F	B	0
B	E	H	1
C	A	A	0
D	F	A	0
E	A	A	0
F	C	G	1
G	D	F	1
H	D	B	1

Use state minimization to determine which states are equivalent to other states. List the sets of states that are equivalent to each other. Show your work for full marks.

$$P_0 = (ABCD \not{EF} GH)$$

①  $P_1 = (ACDE) (BFGH) // \text{diff output } z$

$$\begin{array}{ll} ACDE \quad w=0 \Rightarrow (FAFA) \\ \quad \quad \quad w=1 \Rightarrow (BAAA) \end{array} \quad \quad \quad \begin{array}{ll} BFHG \quad w=0 \Rightarrow (ECD) \\ \quad \quad \quad w=1 \Rightarrow (HGF) \end{array}$$

→ A, D are distinct ①

$$P_2 = (A)(D) (C E) (BF) (GH)$$

$$CE \quad w=0 \Rightarrow (AA) \\ w=1 \Rightarrow (AA)$$

$$\begin{array}{ll} BFHG \quad w=0 \Rightarrow (ECD) \\ \quad \quad \quad w=1 \Rightarrow (HGF) \end{array}$$

GH distinct from BF ①

$$P_3 = (A)(D) (C E) (BF) (GH)$$

$$BF \quad w=0 \Rightarrow (EC) \\ w=1 \Rightarrow (HG)$$

$$\begin{array}{ll} GH \quad w=0 \Rightarrow (DD) \\ \quad \quad \quad w=1 \Rightarrow (FB) \end{array}$$

GH are equiv ①

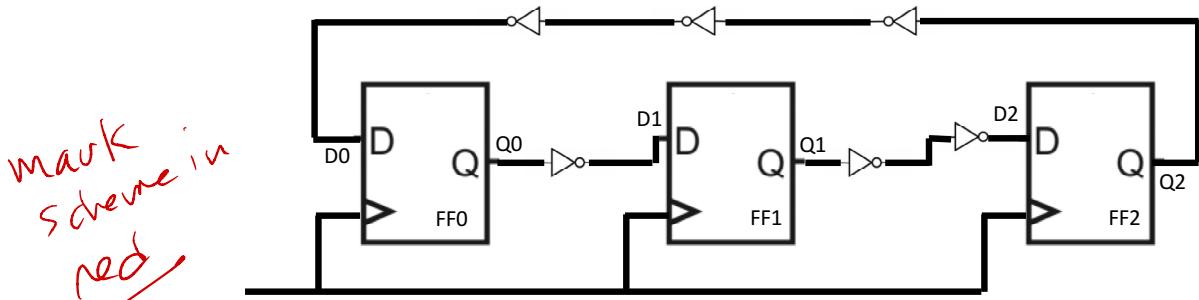
5

① C, E are equivalent

① B, F are equiv

**Question 4 [6 Marks]**

Consider the following circuit having inverters and D-type flip-flops.



Assume the delays in the circuit, and the setup and hold times, are as follows:

component	delay
$t_{inv}$	1.5ns
$t_{cQ}$	1ns
$t_{su}$	0.5ns
$t_{hold}$	1.2ns

a) [2 marks] What is the minimum clock period for the circuit? Show your work for full marks.

$$\begin{aligned} T_{min} &= 3t_{inv} + t_{cQ} + t_{su} \\ &\approx 4.5 + 1 + 0.5 = 6 \text{ ns} \end{aligned}$$

b) [2 marks] Is there a hold-time violation? Show your work for full marks.

fastest path  $Q_0 \rightarrow Q_1 \Rightarrow t_{cQ} + t_{inv} = 1 + 1.5 = 2.5 \text{ ns} > t_{hold}$   $\therefore \text{violation}$

c) [2 marks] Assume the clock arrival time at FF0 can be delayed (i.e., it is possible to introduce clock skew), by how much should it be delayed to minimize the clock period of the circuit? State the new minimum clock period for the circuit. Show your work for full marks.

longest path  $Q_2 \rightarrow D_0$  differ by inv delay } 1 mark  
 2nd longest Path  $Q_1 \rightarrow D_2$   
 $\therefore$  delay arrival at FF0 by  $t_{inv} = 1.5 \text{ ns}$   
 $\text{new } T_{min} = 2t_{inv} + t_{cQ} + t_{su} = 3 + 1 + 0.5 = 4.5 \text{ ns}$  } 1 mark  
 All 3 paths now "effectively" have 2  $t_{inv}$

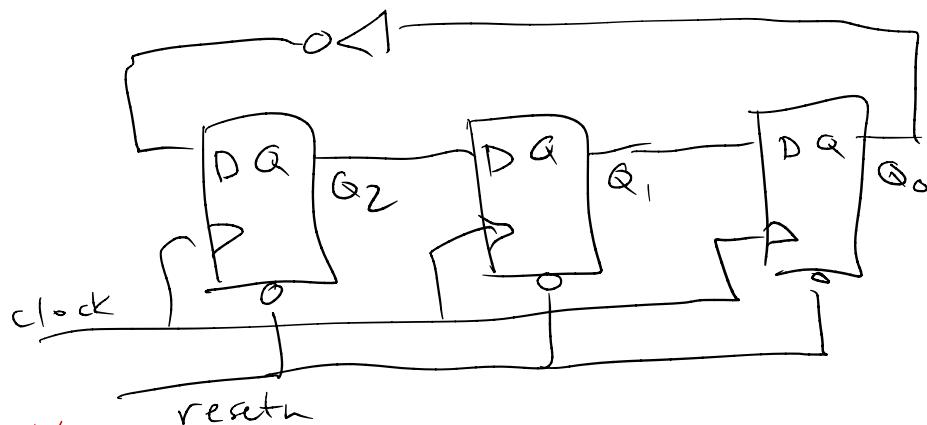
**Question 5 [9 Marks]**

Consider the Verilog code for special type of counter, called a Johnson counter.

```
module johnson(clock, resetn, Q);
    input clock, resetn;
    output reg [2:0] Q;

    always@ (posedge clock, negedge resetn)
    begin
        if (!resetn)
            Q <= 3'b0;
        else
            begin
                Q[2] <= ~Q[0];
                Q[1] <= Q[2];
                Q[0] <= Q[1];
            end
    end
endmodule
```

a) [3 marks] Show the circuit schematic that corresponds to the functionality described in the Verilog. Use any gates and flip-flops you may need.



marks

O.S.: resetn shown

O.S.: clock shown

1 : 3 FFs

O.S.: inv in feedback path

O.S.: shift connectivity b/w FFs

O.S.: shift connectivity b/w FFs

Question 5 continued ...

b) [3 marks] Assume that  $resetn = 1$ , and that at the 0<sup>th</sup> clock cycle,  $Q_2Q_1Q_0 = 000$ , i.e. each flip-flop stores a 0. Show the counter values over the next 6 clock cycles.

	$Q_2$	$Q_1$	$Q_0$	
Clock cycle 1:	1	0	0	
Clock cycle 2:	1	1	0	1 mark each
Clock cycle 3:	1	1	1	
Clock cycle 4:	0	1	1	
Clock cycle 5:	0	0	1	
Clock cycle 6:	0	0	0	

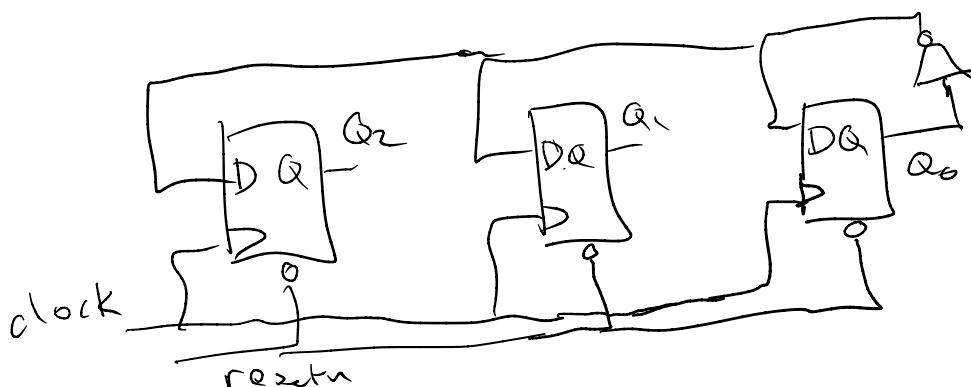
c) [3 marks] If the three assignment statements in the *always* block are changed from *non-blocking* into *blocking* assignments as follows:

```

 $Q[2] = \sim Q[0];$ 
 $Q[1] = Q[2];$ 
 $Q[0] = Q[1];$ 

```

Show the circuit schematic corresponding to the modified Verilog that uses blocking assignments.



- 1 mark: 3 FFs shown
- 1 mark: all 3 FFs have same signal
- 1 mark: on D input
- 1 mark: inverter correct.

## **Question 6 [12 Marks]**

Write an ARM assembly language program that displays a 10-bit binary value on the red LEDs on the DE1-SoC board. The value should initially be 512 (0b1000000000). When  $KEY_3$  is pressed, an interrupt should be generated and cause the displayed value to *decrease* by 1. When  $KEY_0$  is pressed, an interrupt should be generated should be generated and cause the displayed value to *increase* by 1. If the displayed value is 0, pressing  $KEY_3$  should produce no effect; likewise, if the displayed value is 1023 (0b111111111), pressing  $KEY_0$  should produce no effect. The memory-mapped locations for the  $KEYS$  and  $LEDR$  are shown below. The IRQ ID for the  $KEYS$  is 73. 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.



- a) [4 marks] Write the code to enable interrupts for the KEYS, to set up SP for the relevant ARM modes, to enable interrupts in the CPSR, and to display the initial value on *LEDR*. Use address 0x10000 as the SP for supervisor mode; use address 0x20000 as the SP for IRQ mode. The value displayed on *LEDR* should be stored at a memory location labeled CURR\_VALUE. Your code should continually read a value from this location and display it on *LEDR*. Comment your code.

O.S	SP for SVC	MOV R1, #0b11010011 // SVC mode, disable interrupts MSR CPSR, R1 // change mode LDR SP, = 0x1000 // SP for SVC mode LDR R1, = 0xFF200058 // int mask keys MOV R2, #0b1001 // enable for key3, key0 STR R2, [R1] // store to enable interrupts R1, #0b11010010 // IRQ mode, disable interrupts MSR CPSR, R1 // change mode LDR SP, 0x20000 // SP for IRQ mode MOV R1, #0b01010011 // enable interrupts in SVC mode MSR CPSR, R1 LDR R2, = 0xFF200000 // addr of LEDR LDR R3, = CURR_VALUE // addr of curr-value LOOP: LDR R1, [R3] // load currvalue STR R1, [R2] // store to LEDR B LOOP
O.S	SP for IRQ	R1, #0b11010010 // IRQ mode, disable interrupts MSR CPSR, R1 // change mode LDR SP, 0x20000 // SP for IRQ mode MOV R1, #0b01010011 // enable interrupts in SVC mode MSR CPSR, R1 LDR R2, = 0xFF200000 // addr of LEDR LDR R3, = CURR_VALUE // addr of curr-value LOOP: LDR R1, [R3] // load currvalue STR R1, [R2] // store to LEDR B LOOP
O.S	enable interrupts	R1, #0b01010011 // enable interrupts in SVC mode MSR CPSR, R1 LDR R2, = 0xFF200000 // addr of LEDR LDR R3, = CURR_VALUE // addr of curr-value LOOP: LDR R1, [R3] // load currvalue STR R1, [R2] // store to LEDR B LOOP
O.S	continuous loop	CURR_VALUE: .word 0b1000000000 // initial value

NOTE: OKAY if J didnt properly set FIQ bit  
CPSR C6] highlighted

Question 6 continued ...

b) [4 marks] Write the section of the SERVICE\_IRQ subroutine to determine which device is interrupting. Your code should call the KEY\_ISR subroutine if KEYS are triggering the interrupt, and behave appropriately otherwise. You will be asked to write the KEY\_ISR subroutine in part (c).

SERVICE\_IRQ:

```
PUSH R0-R5, LR  
LDR R4, =MPCORE_GIC_CPUIF  
LDR R5, [R4, #ICCIAR] // read the interrupt ID
```

// WRITE YOUR CODE HERE

| { CMP R5, #73 // check if key IRQ

| BNE ERROR

| BL KEY\_ISR // must be a key IRQ, BL to subroutine

| B EXIT\_IRQ // exit after handling IRQ

ERROR:

| B ERROR // unknown IRQ !

EXIT\_IRQ:

```
STR R5, [R4, #ICCEOIR]  
POP R0-R5, LR  
SUBS PC, LR, #4
```

Question 6 continued ...

c) [4 marks] Write the KEY\_ISR subroutine. Be sure you use the stack to store the previous contents of any registers used in your subroutine.

push pop O.S | PUSH {R1-R5}  
get curr value O.S | LDR R5, =CURRENt-VALUE // address of curr-value  
check which key O.S | LDR R4, [RS] // current value to R4  
check overflow O.S | LDR R2, =0xFC20005C // edge capture addr  
LDR R3, [R2] // edge capture value  
CMP R3, #0x8 // check Key3  
BNE KEY0 // if !Key3, must be Key0  
CMP R4, #0 // check if curr-value > 0  
BEQ ENDISR // branch if 0  
SUB R4, R4, #1 // decrease value  
STR R4, [RS] // store  
B ENDISR

KEY0:  
check overflow O.S | LDR R1, =0b11111111 // check if curr-value  
is 11111111  
CMP R4, R1 // clamping check  
BEQ ENDISR  
ADD R4, R4, #1 // increase value  
STR R4, [RS] // store

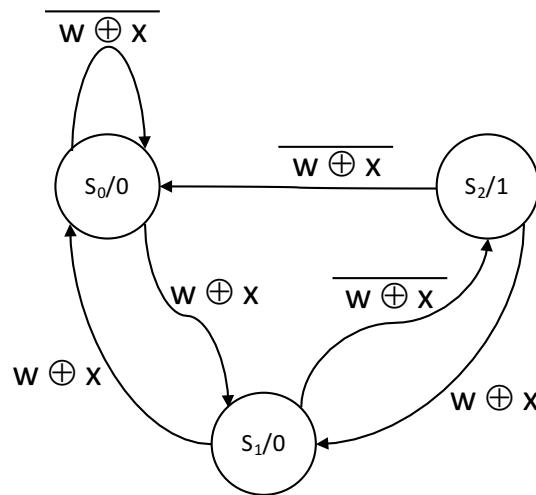
ENDISR:

POP {R1-R5}

return O.S MOV PC, LR

**Question 7 [7 Marks]**

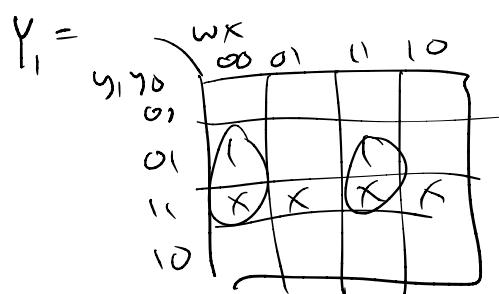
Consider the following state diagram for a finite state machine with inputs  $x$  and  $w$ , and output  $z$ .  
 $\oplus$  means exclusive-OR.



Using the state codes below, derive the next-state logic functions and the output logic function. Use  $Y_1$  and  $Y_0$  as the signal names for the next-state logic; use  $z$  as the output signal name. Use Karnaugh maps to minimize your logic functions, taking advantage of don't-cares if possible. Show your work; you can also use the next page.

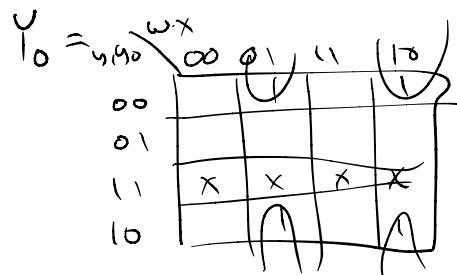
State	$y_1y_0$	Current		$y_1 y_0$ next			output $z$
		$w, x$	$y_1 y_0$	00	01	00	
$S_0$	00	11	11	10	01	00	1
$S_1$	01	00	00	01	01	00	0
$S_2$	10	10	10	00	00	10	0
		00	00	01	01	00	

*2 marks  
state table*



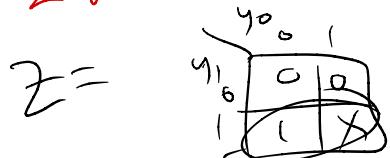
$$Y_1 = \overline{w} \overline{x} y_0 + w x y_0$$

*2 marks*



$$Y_0 = \overline{y}_0 \overline{w} x + \overline{y}_0 w \overline{x}$$

*2 marks*



$$Z = y_1$$

*1 mark*

*(-0.5 if didn't use X)*

Question 7 continued ...

$$Y_1 =$$

$$Y_0 =$$

$$z =$$

### Question 8 [8 Marks]

You are to write an assembly-language subroutine that computes the  $n^{th}$  number in the Fibonacci sequence. The  $n^{th}$  Fibonacci number is computed as:

$$Fib(n) = Fib(n - 1) + Fib(n - 2)$$

Note that  $Fib(0) = 0$  and  $Fib(1) = 1$ . Your subroutine must be recursive. Equivalent C code for such a subroutine is shown below.

```
int FIBONACCI(int N)
{
    if (N < 2)
        return N;
    else
        return FIBONACCI(N - 1) + FIBONACCI(N - 2);
}
```

You need to provide a main program that calls your FIBONACCI subroutine. The value of the argument N should be loaded from a memory location with label N, and passed to your subroutine. You can assume that  $N > 1$ . A skeleton has been provided for you to fill in.

```
.data
N: .word 10
.text
.global _start
_start:
    LDR SP, =0x20000    +1 initialize stack.
    LDR R4, =N
    LDR R0, [R4]    +1 load N from mem
    MOV R1, #0
    BL FIBONACCI    +1 use BL
END: B END

FIBONACCI:           // return via R1
    PUSH {R0, LR}    +1 push
    MOV R2, R0
    CMP R2, #2    +1 condition
    BLT return
    SUB R0, R0, #1
    BL FIBONACCI
    ADD R1, R2
    SUB R0, R0, #1
    BL FIBONACCI
    ADD R1, R2
    +1 recursion
```

+1 return: POP {R0, PC}  
+1 pop  
+1 return

14

**Question 9 [16 Marks]**

Consider the following ARM assembly language program:

```
.text
.global _start
_start:
    MOV R0, #0xFF
    MOV R1, #-1
    MOV R6, #0x8F000000
    LDR R7, =LIST
    LDR R8, [R7,#8]
    ADD R7, R7, #4
    LDR R9, =RESULT
    LDR SP, =0x20000
    BL FUNC1
    STR R6, [R7]
END: B END
FUNC1:
    PUSH {R6,R7,R8,LR}
    LSL R2, R0, #4
    EOR R3, R2, R1
    ASR R1, R6, #3
    BL FUNC2
    POP {R6,R7,R8,PC}
FUNC2:
    PUSH {R2,R3,R9,LR}
THIS: ADD R2, R2, R3
    POP {R2,R3,R9,PC}
LIST: .word -2, -3, -4, -5, 3, 4
RESULT: .word 0
.end
```

Fill in the table on the following page to indicate the contents of the stack when the instruction at label "THIS" executes. The table may contain more rows than needed. The last row is partially completed for you. Be sure to clearly specify the full hexadecimal value for each entry in the "Data" column. Also, give the name of the register each value corresponds to when it was pushed to the stack. Assume the first instruction of this program is stored at memory address 0x00000000.

+1 mark for data register name  
 +1 mark for

Memory Address	Data (in hex)	Register name
0x001FFE0	0x00000FF0	R2
0x001FFE4	0xFFFFF00F	R3
0x001FFE8	0x00000068	R9
0x001FFFC	0x00000040	LR
0x001FFF0	0x8F000080	R6
0x001FFF4	0x00000054	R7
0x001FFF8	0xFFFFFFF0	R8
0x001FFFC	0x00000024	LR
0x00020000	—	—

**Question 10 [10 Marks]**

*2 marks each*

Short answer:

- (a) Why is the CPSR saved when an interrupt occurs?

*so that the flags & mode can be registered when the ISR finishes*

- (b) If the following instruction is executed: `CMP R1, R2` with  $R1 = 0xFFFF0000$  and  $R2 = 0x00010000$ , what will be the values of the flags (C, V, N, Z) in the CPSR.

$$C = 1, V = 0, N = 1, Z = 0$$

- (c) Why is BL used to branch to a subroutine instead B?

*need to save instruction to return to in link register*

- (d) Consider the following short program (the first instruction is at memory address 0x0):

```
_start:  
    MOV R1, #0  
    ADD R1, R1, #1  
    MOV R15, #4
```

What does this program do?

*executes an infinite loop of ADD & mov*

- (e) Prior to learning ARM assembly, we introduced a simple processor in lecture. This processor has 6-bit instructions and 4 registers. Give one reason this processor has fewer registers than the ARM processor.

*instructions are narrower so don't have enough bits for more registers*

*This page has been left blank intentionally. You may use it for answers to any questions.*