**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 | CE |
| 11000111 | -57 | C7 |
| 01011011 | 91 | 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 = x FFFFFFFE $(-2)_{10}$

1 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 | Next State | | Output |
|---|---|---|---|
| State | w=0 | w=1 | z |
| 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.

$P_0 = (ABCDEFGH)$

1 $P_1 = (ACDE)(BFGH)$ // diff output z

ACDE w=0 ⇒ (FAFA)
w=1 ⇒ (BAAA)

BFGH w=0 ⇒ (ECDD)
w=1 ⇒ (HGFB)

⇒ A, D are distinct 1

GH distinct from BF 1

$P_2 = (A)(D)(CE)(BFGH)$

CE w=0 ⇒ (AA)
w=1 ⇒ (AA)

BFGH w=0 ⇒ (ECDD)
w=1 ⇒ (HGFB)

$P_3 = (A)(D)(CE)(BF)(GH)$

BF w=0 ⇒ (EC)
w=1 ⇒ (HG)

GH w=0 ⇒ (DD)
w=1 ⇒ (FB)

1 C, E are equivalent

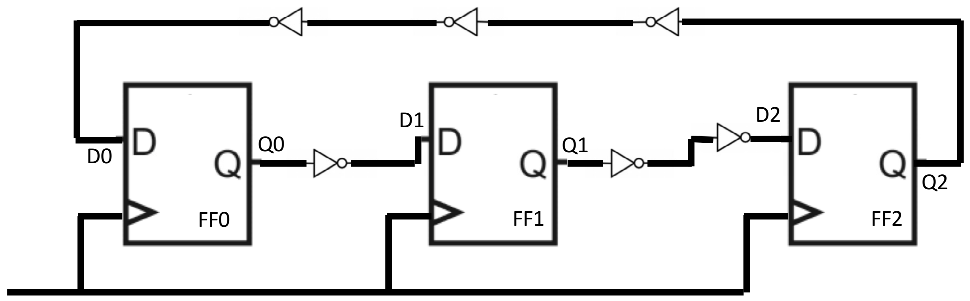
G, H are equiv 1

1 B, F are equiv

**Question 4** [6 Marks]

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

mark scheme in red



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.

$$T_{min} = 3t_{inv} + t_{cQ} + t_{su}$$
$$= \underset{0.5}{4.5} + \underset{0.5}{1} + \underset{0.5}{0.5} = \underset{0.5}{6ns}$$

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

fastest path $Q_0 \rightsquigarrow Q_1 \Rightarrow \underset{0.5}{t_{cQ}} + \underset{0.5}{t_{inv}} = 1 + 1.5 = \underset{0.5}{2.5ns} > \underset{0.5}{t_{hold}}$

$\therefore$ no 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 $Q_2 \to D_0$
2nd longest path $Q_1 \to D_2$ ) differ by 1 $\underline{\text{inv}}$ delay } 1 mark

$\therefore$ delay arrival at FF0 by $t_{inv} = 1.5ns$

new $T_{min} = 2t_{inv} + t_{cQ} + t_{su} = 3 + 1 + 0.5 = 4.5ns$ } 1 mark

All 3 paths now "effectively" have 2 $\underline{t_{inv}}$

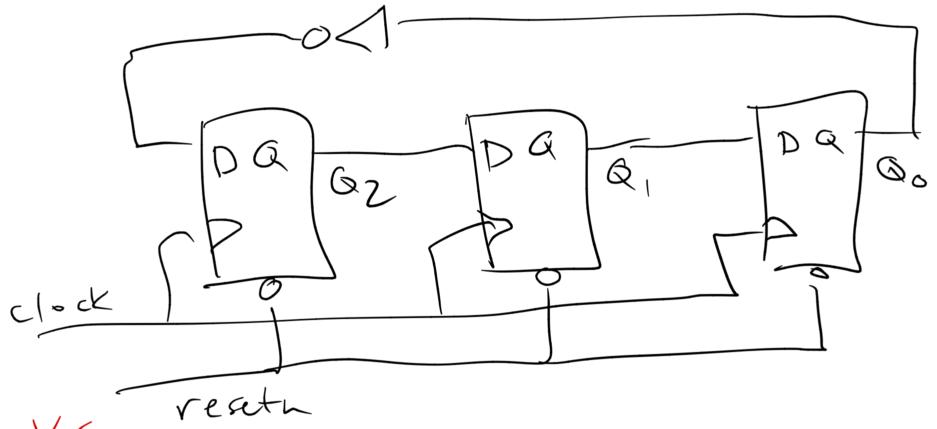
**Question 5** [9 Marks]

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.



<u>marks</u>

0.5 : resetn shown

0.5 : clock shown

1 : 3 FFs

0.5 : inv in feedback path

0.5 : shift connectivity b/w FFs

Question 5 continued . . .

b) [3 marks] Assume that $resetn = 1$, and that at the 0th 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.

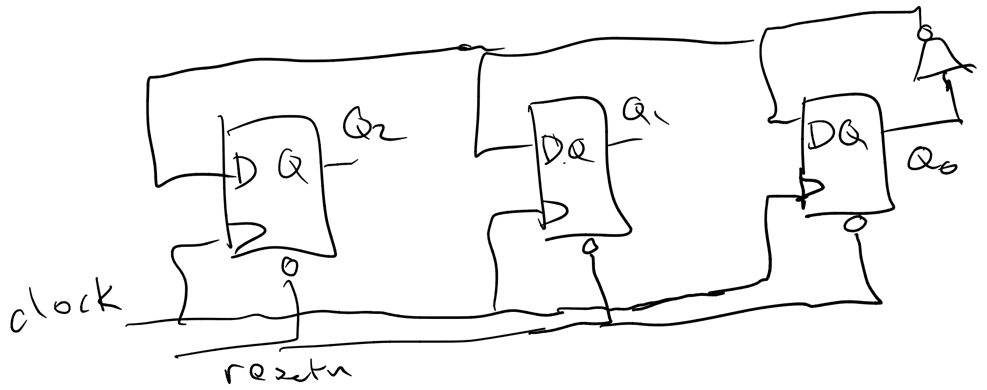
| | Q2 | Q1 | Q0 |
|---|---|---|---|
| Clock cycle 1: | 1 | 0 | 0 |
| Clock cycle 2: | 1 | 1 | 0 |
| Clock cycle 3: | 1 | 1 | 1 |
| Clock cycle 4: | 0 | 1 | 1 |
| Clock cycle 5: | 0 | 0 | 1 |
| Clock cycle 6: | 0 | 0 | 0 |

1/2 mark each

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

```
Q[2] = ~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

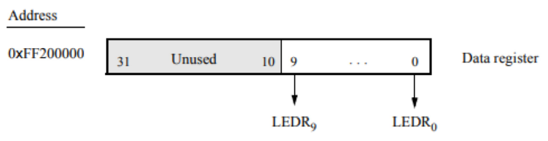
1 mark: all 3 FFs have same signal on D input

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 (0b1111111111), 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.

| Address | 31 30 . . . 4 | 3 2 1 0 | |
|---|---|---|---|
| 0xFF200050 | Unused | $KEY_{3-0}$ | Data register |
| Unused | Unused | | |
| 0xFF200058 | Unused | Mask bits | Interruptmask register |
| 0xFF20005C | Unused | Edge bits | Edgecapture register |



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.

```
0.5 SP for SVC      MOV  R1, #0b11010011 // svc mode, disable interrupts
                    MSR  CPSR, R1  // change mode
                    LDR  SP, =0x10000 // SP for SVC mode
1   Keys enable     LDR  R1, =0xFF200058 // int mask keys
                    MOV  R2, #0b1001 // enable for keys3, keys0
                    STR  R2, [R1] // store to enable interrupts
0.5 SP for IRQ      MOV  R1, #0b11010010 // IRQ mode, disable interrupts
                                          // change mode
                    MSR  CPSR, R1
                    LDR  SP, 0x20000 // SP for IRQ mode
0.5 enable          MOV  R1, #0b01010011 // enable interrupts in SVC
    interrupts                               mode
                    MSR  CPSR, R1
                    LDR  R2, =0xFF200000 // addr of LEDR
                    LDR  R3, =CURR_VALUE // addr of curr-value
1   continuous     LOOP:
    loop            LDR  R1, [R3] // load currvalue
                    STR  R1, [R2] // store to LEDR
                    B    LOOP
0.5                CURR_VALUE:
                    .word 0b1000000000 // initial value
                                 9
```

NOTE: OKAY if didnt properly set FIQ bit CPSR[6] 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 RO-R5, LR
    LDR R4, =MPCORE_GIC CPUIF
    LDR R5, [R4, #ICCIAR] // read the interrupt ID
```

**// WRITE YOUR CODE HERE**

```
1 {  CMP  R5, #73 // check if key IRQ
  {  BNE  ERROR
1    BL   KEY_ISR // must be a key IRQ, BL to subroutine
1    B    EXIT_IRQ // exit after handling IRQ


   ERROR:
1      B ERROR // unknown IRQ !
```

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

Question 6 continued . . .

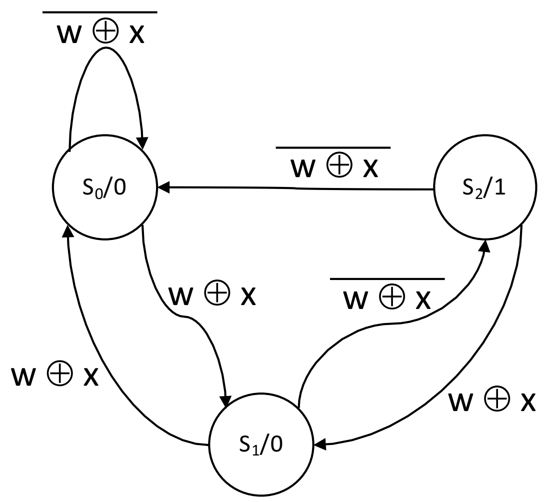
c) [4 marks] Write the KEY_ISR subroutine. Be sure you use the stack to store the previous contents of any registers used in your subroutine.

```
push pop 0.5            PUSH {R1-R5}
                        LDR R5, =CURR_VALUE // address of curr_value
get curr value 0.5  |   LDR R4, [R5] // current value to R4
                    |   LDR R2, =0xFC20005C // edge capture addr
0.5                 |   LDR R3, [R2] // edge capture value
check which key     |   CMP R3, #0x8 // check key3
                    |   BNE KEY0 // if !key3, must be key0
                    |   CMP R4, #0 // check if curr_value 0
check underflow 0.5 |   BEQ ENDISR // branch if 0
                        SUB R4, R4, #1 // decrease value
subtract & store 0.5 |  STR R4, [R5] // store
                        B ENDISR

                    KEY0:
                        LDR R1, =0b11111111 // check if curr_value is 11111111
                    |   CMP R4, R1 // clamping check
check overflow 0.5  |   BEQ ENDISR
                        ADD R4, R4, #1 // increase value
add & store 0.5     |   STR R4, [R5] // store
                    ENDISR:
                        POP {R1-R5}
return 0.5              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 | $y_1y_0$ |
|---|---|
| $S_0$ | 00 |
| $S_1$ | 01 |
| $S_2$ | 10 |

| Current | $w,x$ | $Y_1 Y_0$ next | | | output |
|---|---|---|---|---|---|
| $y_1 y_0$ | 11 | 10 | 01 | 00 | $z$ |
| 00 | 00 | 01 | 01 | 00 | 0 |
| 01 | 10 | 00 | 00 | 10 | 0 |
| 10 | 00 | 01 | 01 | 00 | 1 |

2 marks state table

$Y_1 =$

| $y_1y_0$ \ $wx$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 01 | | | | |
| 01 | 1 | | 1 | |
| 11 | X | X | X | X |
| 10 | | | | |

$Y_1 = \bar{w}\bar{x}y_0 + wxy_0$

2 marks

$Y_0 =$

| $y_1y_0$ \ $w x$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | | 1 | | 1 |
| 01 | | | | |
| 11 | X | X | X | X |
| 10 | | 1 | | 1 |

$Y_0 = \bar{y}_0\bar{w}x + \bar{y}_0 w\bar{x}$ 2 marks

$z =$

| $y_1$ \ $y_0$ | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | X |

$z = y_1$ 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 in 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
                     return: POP {R0, PC}
                             +1 pop
                             +1 return
                     +1 recursion
```

**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
+1 mark for register name

| Memory Address | Data (in hex) | Register name |
|---|---|---|
| 0x0001FFE0 | 0x00000FF0 | R2 |
| 0x0001FFE4 | 0xFFFFF00F | R3 |
| 0x0001FFE8 | 0x00000068 | R9 |
| 0x0001FFEC | 0x00000040 | LR |
| 0x0001FFF0 | 0x8F000000 | R6 |
| 0x0001FFF4 | 0x00000054 | R7 |
| 0x0001FFF8 | 0xFFFFFFFC | R8 |
| 0x0001FFFC | 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 restored 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.*