<u>Honor code:</u> I affirm that I will not give or receive any unauthorized help on this exam, and that all work will be my own.

Your name (print):_____ Signature:_____

CPEG 422/622 MIDTERM April 13, 2020

The midterm is made of **5** questions for a total of **100** points.

The questions have been ordered according to their difficulty.

Keep that in mind as you allocate your time to each problem.

Good luck!

Question	**Points**
1	/10
2	/15
3	/20
4	/25
5	/30
Total	/100

1 VHDL (10 points)

The following VHDL testbench tries to generate two random numbers A and B that can be used to test 16-bit adders and multipliers.

```
1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.ALL;
3.  entity tb_test is
4.  end tb_test;

5.  architecture tb_test of tb_test is

6.       component LFSR
7.       Port (
8.            clock :  in STD_LOGIC;
9.            reload:  in STD_LOGIC;
10.               D :  in STD_LOGIC_VECTOR (15 downto 0);
11.              en :  in STD_LOGIC;
12.               Q : out STD_LOGIC_VECTOR (15 downto 0)
13.          );
14.      end component;

15.      signal A,B,seedA,seedB : std_logic_vector(15 downto 0);
16.      signal clk,start,reset: std_logic;
17.      constant clk_period : time := 10 ns;

18.  begin

19.        clk_set:process
20.        begin
21.             clk <= '0'; wait for clk_period/2;
22.             clk <= '1'; wait for clk_period/2;
23.        end process;

24.        rst: process
25.        begin
26.             seedA<="1111111111101110";
27.             seedB<="0000000000011001";
28.             reset<='1';
29.             start<='0';
30.             wait for 10 ns;
31.             reset<='0';
32.             start<='1';
33.             wait ;
34.        end process;

35.        A_gen:  LFSR PORT MAP(clk, reset,seedA, start, A);
36.        B_gen:  LFSR PORT MAP(clk, reset,seedB, start, B);

37. end tb_test;
```

(Part A) [6pts] Please answer each of the following questions and give your reasoning.

There are two processes in the testbench. Are they executed concurrently or sequentially?

Concurrently

Are Line 26 and 27 executed concurrently or sequentially?

Sequentially, as they are inside one process.

Are Line 35 and 36 executed concurrently or sequentially?

Concurrently, as they are onside any process.

(Part B) [4pts] Suppose the LFSR unit has been implemented and you want to implement the whole design on the FPGA to evaluate hardware, power, and timing. Can the testbench be synthesized and implemented on the FPGA? Why or why not?

Testbench cannot be synthesized. It includes wait statements that are not synthesizable.

2 Arithmetic Questions (15 points)

(Part A) [8pts] Choose the best answer and give a short explanation to justify your answer (no justification, no points).

1) **True / False**: Booth's algorithm performs addition/subtraction alternately.
   The sequence is always sub, add, sub, add ….

2) Booth's multiplier **always / sometimes / never** overflows.
   It always adds two values of different signs, or subtracts two values of the same sign

3) **True / False**: A Booth's multiplication sequence may have more addition operations than subtraction.
   The sequence is always sub, add, sub, add …. it thus may have more sub than add but not the other way.

4) **True / False**: A Booth's multiplication sequence may have more subtraction operations than addition.
   Same reason as above

5) The first operation of Booth's multiplication is always an **addition / subtraction**. After the first operation, the partial product has the **same / opposite** sign as the multiplicand.
   The first operation is sub, it subtracts multiplicand from 0, thus has the opposite sign.

6) If the multiplier is positive, the total number of addition/subtraction operations is **even / odd**. If the multiplier is negative, the total number of addition/subtraction operations is **even / odd**.
   A positive multiplier, such as 0110, have equal number of 01 and 10 transitions. A negative multiplier ends with 1 at the left most position, thus will have one more subtraction than addition.

(Part B) [7pts] You are given two floating point numbers following the *Blue Hen 422* representation, which is very similar to IEEE 754 representation except that *Blue Hen 422* has 24 bits in total. Normalized numbers of *Blue Hen 422* also have a hidden 1 in the fraction.

Name	Size	Sign	Exponent	Fraction	Bias value
Blue Hen 422	24 bits	1 bit	7 bits	16 bits	63

The two numbers are A = 1 1000001 0000000000000101 and B = 0 1000001 0000000000000000.

1) [2pts] Which number is larger?

   B is larger as it is positive, while A is negative.

2) [5pts] What is the sum of these two numbers? Please show the steps for computing the sum and represent it in normalized *Blue Hen 422* representation.

   The two values have the same exponent; thus, their fraction can be directly added.
   As they have different signs, we will perform A – B for the fraction part.
   Fraction: A – B = 1.0000 0000 0000 0101 - 1.0000 0000 0000 0000
   = 0.0000 0000 0000 0101 = 1.01 x 2^ -14
   We will subtract 14 from the exponent, to get the normalized representation,
   so the exponent is 1000001 – 0001110 = 0110011
   Final answer: S = 1, Exponent = 0110011, Fraction = 0100 0000 0000 0000 (omit the hidden 1)

   Number: 1 0110011 0100 0000 0000 0000 = 0xb34000

3 Adder Timing (20 points)

As you probably remember from the lecture and observed in your project 1 implementation, the longest path in an n-bit adder is typically the path from the least significant bit of inputs (i.e., A_0 or B_0) to the most significant bit of sum (i.e., S_{n-1}). In this question you are asked to compute and compare the critical paths of carry-ripple adders (CRA) and carry-lookahead adders (CLA) under the following timing assumptions.

Type	Wire	2-input AND/OR	2-input XOR	3-input AND/OR	4-input AND/OR	5-input gate
Delay	0	1 ns	1.8 ns	1.5 ns	2 ns	N/A

(Part A) [4pts] For 4-bit CRA and 4-bit CLA that we studied in class, please list the sequence of gates that the longest path goes through and compute the longest path.

CRA: delay of critical path A_0/B_0 to S_3 is _____9.6_____ ns.
List of gates that the path goes through: 2-input XOR, (2-input AND, 2-input OR)*3, 2-input XOR

4-bit CRA path A0/B0 -> C1 -> C2 -> C3 -> S3
A0/B0 -> C1: 2-input XOR, 2-input AND, 2-input OR, delay = 1.8+1+1 = 3.8 ns
C1 -> C3 = 2 x the delay of C1->C2 (2-input AND, 2-input OR), delay = 2*2 = 4 ns
C3 -> S3: 2-input XOR, delay = 1.8ns
Total delay = 3.8 + 4 + 1.8 = 9.6

CLA: delay of critical path A_0/B_0 to S_3 is _____7.6_____ ns.
List of gates that the path goes through: 2-input XOR, 4-input AND, 4-input OR, 2-input XOR

4-bit CLA path A0/B0 -> P0 -> C3 -> S3
A0/B0 -> P0: 2-input XOR, delay = 1.8 ns
P0 -> C3: 4-input AND, 4-input OR, delay = 2 + 2 = 4ns
C3 -> S3: 2-input XOR, delay = 1.8ns
Total delay = 1.8 + 4 + 1.8 = 7.6

(Part B) [6pts] For 16-bit CRA and 16-bit CLA (i.e., the 2-level structure with 5 CLA generators) that we studied in class, please list the sequence of gates that the longest path goes through and compute the longest path.

CRA: delay of critical path A_0/B_0 to S_{15} is _____33.6_____ ns.
List of gates that the path goes through: 2-input XOR, (2-input AND, 2-input OR)*15, 2-input XOR

16-bit CRA path A0/B0 -> C1 -> C2 -> C3 -> … -> C15 -> S3
A0/B0 -> C1: 2-input XOR, 2-input AND, 2-input OR, delay = 1.8+1+1 = 3.8 ns
C1 -> C15 = 14 x the delay of C1->C2 (2-input AND, 2-input OR), delay = 14*2 = 28 ns
C15 -> S15: 2-input XOR, delay = 1.8ns
Total delay = 3.8 + 28 + 1.8 = 33.6

CLA: delay of critical path A_0/B_0 to S_{15} is _____13.6_____ ns.
List of gates that the path goes through: 2-input XOR, 4-input AND, (4-input AND, 4-input OR)*2, 2-input XOR

16-bit CLA path A0/B0 -> P0 -> PG0 -> C12 -> C15 -> S15
A0/B0 -> P0: 2-input XOR, delay = 1.8 ns
P0 -> PG0: 4-input AND, delay = 2ns
PG0 -> C12: this is similar to P0->C3 in 4-bit CLA, it goes through 4-input AND, 4-input OR, delay = 4ns
C12 -> C15: this is similar to C0->C3 in 4-bit CLA, it goes through 4-input AND, 4-input OR, delay = 4ns
C15 -> S15: 2-input XOR, delay = 1.8ns
Total delay = 1.8 + 2 + 4 + 4 + 1.8 = 13.6

(Part C) [6pts] Not every input combination will activate the critical path. Specifically, in order for a value change in A_0 to propagate to S_{15}, the rest bits of A and B need to form some property.
The following table shows three values of A. Please choose the value of B carefully, so that a transition in A_0 causes a transition in S_{15}.

A[15:0]	B[15:0]
0000 0000 0000 0001	x111 1111 1111 1111
0110 1001 0011 1100	x001 0110 1100 0011
1000 0111 0100 0101	x111 1000 1011 1011

For A0 to propagate to S15, a change in A0 needs to cause a change in C15. B15 can be a don't care, but the other values of B are fixed and should be a value that let A0 propagate.
To give you a counter example, for the first case in the table, assume B = 1111 1111 1111 0111. When B3 is 0, a change in A0 propagates to S3 and then stops there and cannot propagate any further. Other bits of B (except for B15) being 0 would cause the same problem.

(Part D) [4pts] Does the property of inputs A and B listed in (Part C) regarding triggering the critical path hold for both carry-ripple adders and carry-lookahead adders? Why or why not? Please analyze each type of adder and give your conclusion.

The property holds for both type of adders. The critical path is activated **if and only if** the change in the input causes the change in the output, no matter what type of implementation it uses.

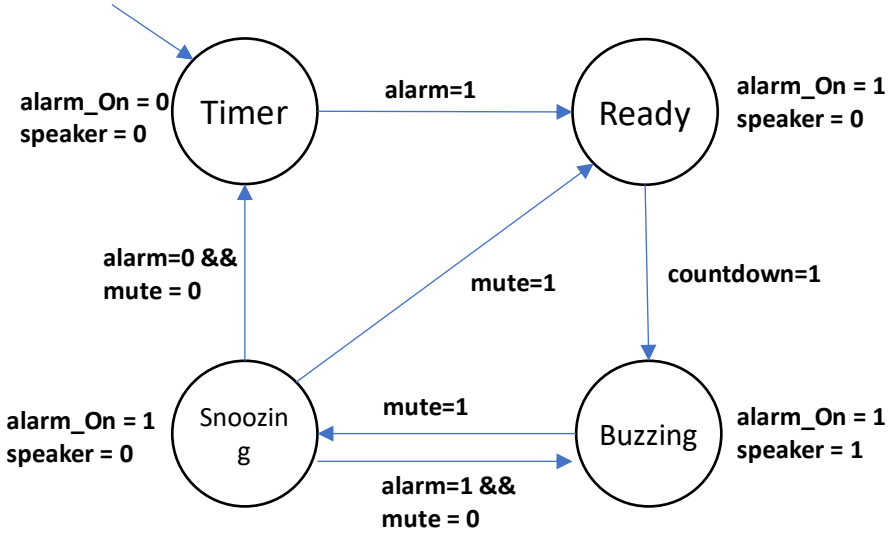
4 FSM (25 points)

In this question we will design an alarm clock. It has four modes:

- **Timer**: This is the default mode of the alarm clock. If the "alarm" button is ON, the alarm will be in Ready mode; otherwise it will remain in timer mode.
- **Ready**: The "alarm_on" light is ON in this mode. When "countdown" condition becomes true, the alarm will be in Buzzing mode; otherwise it will remain in Ready mode.
- **Buzzing**: The "speaker" is ON in this mode. When "mute" button is pressed (ON), the alarm will be in snoozing mode; otherwise it will remain in Buzzing mode.
- **Snoozing**: The "speaker" is OFF in this mode. If "mute" button is pressed (ON), the alarm will be in Ready mode; else if "alarm" button is OFF, the alarm will be in Timer mode and "alarm_on" light will be OFF; otherwise the alarm will be in Buzzing mode automatically.

(Part A) [8pts] Please draw the MOORE FSM of the alarm clock.

This is according to the description. Self-transitions are not shown.



(Part B) [5pts] Please complete the entity declaration of the alarm clock by filling in the input and output ports. There are **seven** ports in total, and two of them are common in most sequential circuits.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity FSM is
 Port(
      reset:       in STD_LOGIC;
      clock:       in STD_LOGIC;
      alarm:       in STD_LOGIC;
      countdown:   in STD_LOGIC;
      mute:        in STD_LOGIC;
      alarm_On:    out STD_LOGIC;
      speaker:     out STD_LOGIC
      );
end FSM;
```

(Part C) [12pts] Assume the use of Gray code for the four states: Timer (00), Ready (01), Buzzing (11), Snoozing (10). Please complete the following alarm clock implementation by finishing the output and next state logic.

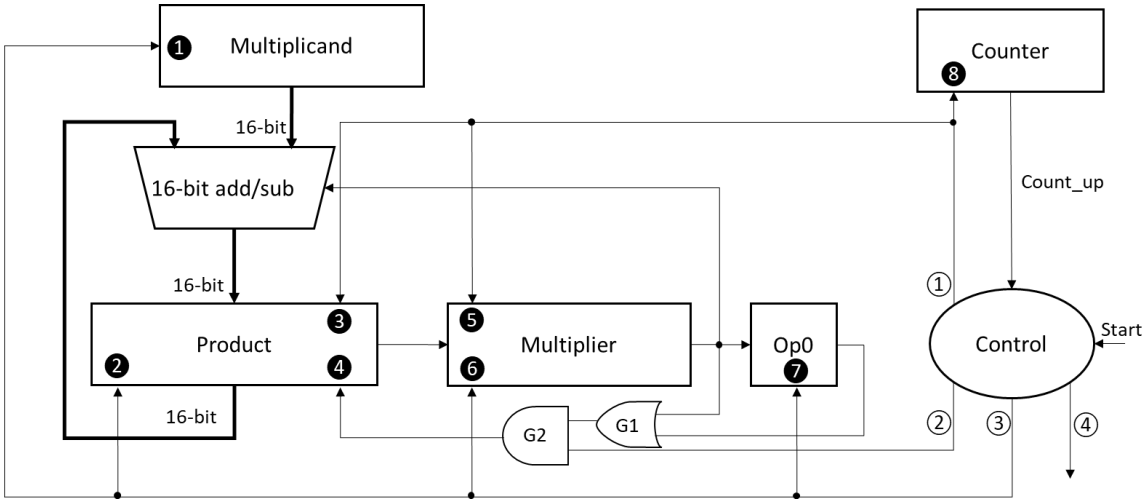
```
architecture Behavioral of FSM is
  signal state,next_state: std_logic_vector(1 downto 0);
begin
 sync:process(clock)
 begin
    if(rising_edge(clock)) then
      if(reset='1') then
       state<="00";
      else
       state<=next_state;
      end if;
    end if;
 end process;

 next_state_logic:process(alarm,countdown,mute,state)
 begin
       case state is
         when "00" =>
              if (alarm = '1') then next_state <= "01";
              end if;
         when "01" =>
              if (countdown = '1') then next_state <= "11";
              end if;
         when "11" =>
              if (mute = '1') then next_state <= "10";
              end if;
         when "10" =>
              if (mute = '1') then next_state <= "01";
              elsif (alarm = '0') then next_state <= "00";
              else next_state <= "11";
              end if;
       end case;
 end process;
```

```
  outputs:process(state)
  begin
       case state is
         when “00” =>
              speaker <= ‘0’;
              alarm_On <= ‘0’;
         when “01” =>
              speaker <= ‘0’;
              alarm_On <= ‘1’;
         when “11” =>
              speaker <=1;
              alarm_On <= ‘1’;
         when “10” =>
              speaker <= ‘0’;
              alarm_On <= ‘1’;
       end case;
  end process;
end Behavioral;
```

5 Booth Multiplication (30 points)

The following figure shows the block diagram of a **16-bit** Booth's multiplier. The bold lines are data signals, while the thin lines are control signals.



(Part A) [6pts] Please choose the correct signal/port names showing how the multiplier is implemented.

Signals:

① _____ (INIT/LOADA/SHIFT/DONE)

② _____ (INIT/LOADA/SHIFT/DONE)

③ _____ (INIT/LOADA/SHIFT/DONE)

④ _____ (INIT/LOADA/SHIFT/DONE)

Ports:

❶ _____ (CLEAR/CLOCK/LOAD/SHIFT)

❷ _____ (CLEAR/CLOCK/LOAD/SHIFT)

❸ _____ (CLEAR/CLOCK/LOAD/SHIFT)

❹ _____ (CLEAR/CLOCK/LOAD/SHIFT)

❺ _____ (CLEAR/CLOCK/LOAD/SHIFT)

❻ _____ (CLEAR/CLOCK/LOAD/SHIFT)

❼ _____ (CLEAR/CLOCK/LOAD/SHIFT)

❽ _____ (CLEAR/CLOCK/LOAD/SHIFT)

(Part B) [10pts] The diagram shown on the previous page has a bug. Gate G1 in the figure is an OR gate, which should be an XOR gate in the correct implementation. As a result of this bug, the multiplier cannot work correctly for certain input combinations.

For each of the multiplicand/multiplier pairs given below, determine whether the final product produced by this buggy multiplier will be correct or not. Please mark your choices and **briefly explain your reasoning**.

Multiplicands	**Multipliers**	**Result Correctness**
1100 1110 1011 0001	1010 1010 1000 0001	CORRECT / INCORRECT
1011 1010 0100 1001	0110 1010 1001 1010	CORRECT / INCORRECT
0011 1011 1010 1111	1001 0010 1010 0101	CORRECT / INCORRECT
0010 1010 1010 1001	0001 1010 1010 1001	CORRECT / INCORRECT

The OR and XOR gates differ when the inputs are 11. So, if in the multiplier there is no "11" in any two adjacent bit positions, the results will be correct, otherwise the result will be incorrect.

(Part C) [10pts] Suppose the bug is fixed. Now G1 is replaced with an XOR gate, and the design is correct. In our project 2 implementation, it always takes 2 cycles (first LoadA then Shift) to process each bit of the multiplier. A smart student wants to optimize it. She changes the design so that if there is no need to add or subtract, the LoadA cycle can be skipped.
For each of the A/B pairs given below, we are evaluating whether swapping the two numbers (i.e., performing B * A instead of A * B) is beneficial for saving cycles. Please make the decision by determine (1) the # of addition and subtraction when performing A (multiplicand) * B (multiplier), (2) the # of addition and subtraction when performing B (multiplicand)* A (multiplier), and (3) whether it is beneficial to swap.

A	**B**	**A * B**		**B * A**		**Beneficial to Swap?**
		Add #	**Sub #**	**Add #**	**Sub #**	
1100 1110 1011 0001	1011 1000 0111 1011	3	4	4	5	Yes / No
1011 1010 0100 1000	0110 1011 1001 1110	4	4	4	5	Yes / No
0011 1011 1010 1111	1001 0010 1110 0110	4	5	4	4	Yes / No
0010 1010 1000 1001	0001 1110 1110 1001	4	4	6	6	Yes / No

In the new design, if we perform fewer addition/subtraction, we can save time.
The number of addition = the number of "10" in the multiplier/multiplicand.
The number of subtraction = the number of "01" in the multiplier/multiplicand.

(Part D) [4pts] If the Booth's multiplier is used to multiply two floating point numbers (only the fraction parts), will it give correct results? Why or why not? Please give a brief explanation.

It will not give correct results. Floating point numbers have hidden ones. If you include hidden ones, the unsigned numbers will be treated as negative by the multiplier (since Booth's multiplier operates on 2's complement), causing errors.
If you want to get correct results, you need to use 25-bit multiplier for IEEE 754 single-precision numbers, and put a leading 0 in both the multiplier and the multiplicand.

(Extra Sheet)