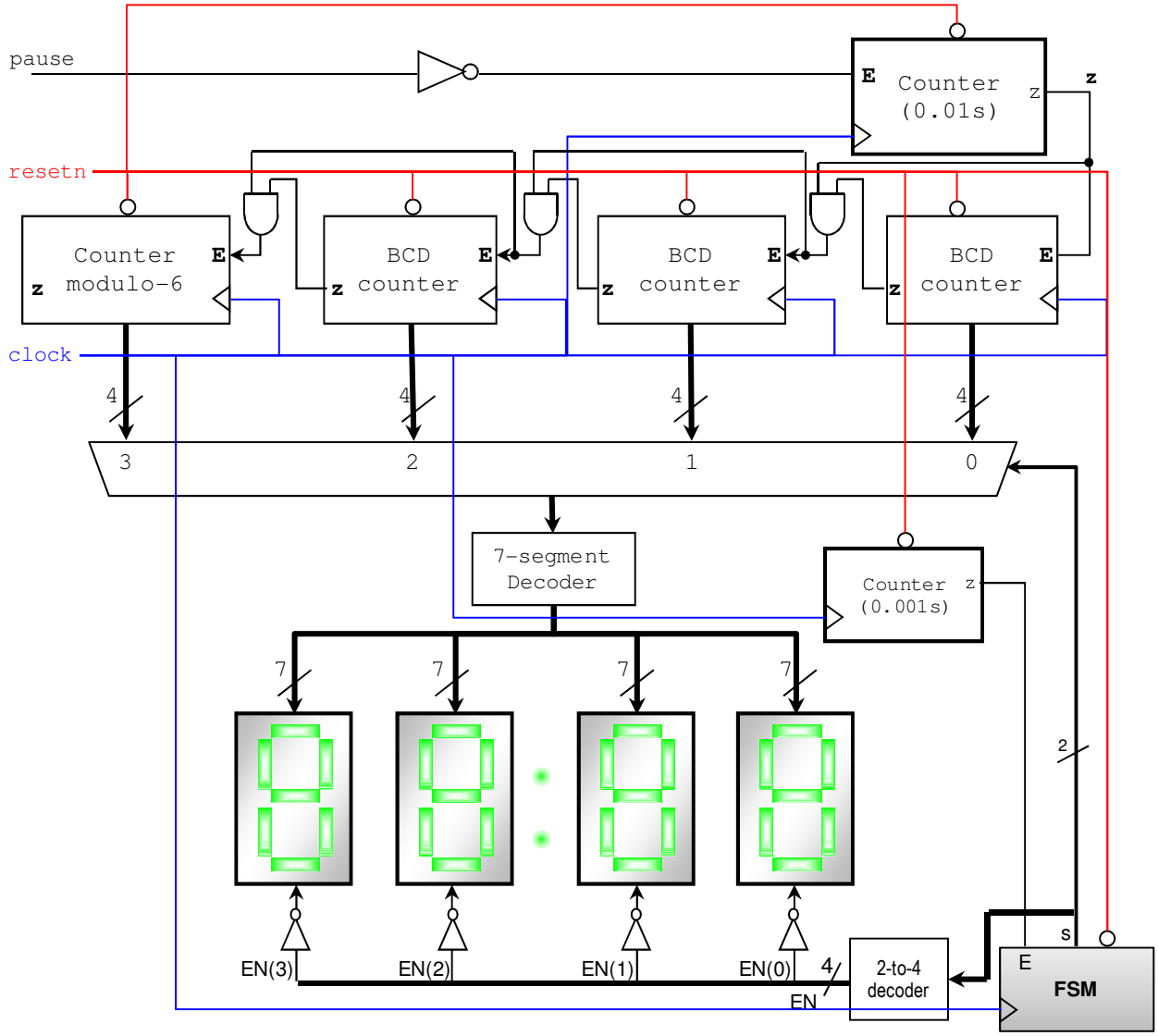
Homework 4

(Due date: November 26th @ 9:30 am)
Presentation and clarity are very important!

PROBLEM 1 (20 PTS)

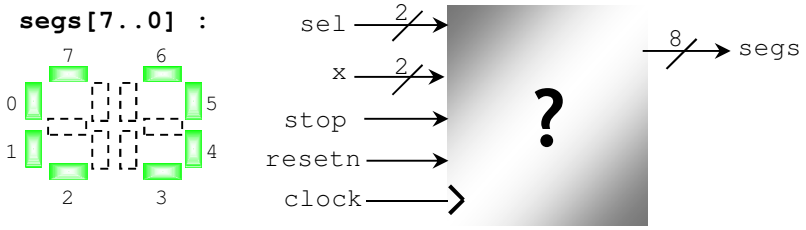
- **Digital Stopwatch**: The architecture of a digital stopwatch is provided. We require counters with an output 'z'. This output is asserted (for one clock cycle) when the counter reaches its maximum count.
  - ✓ Counter (0.01s). It counts up to $10^6 - 1$, asserting 'z' when the count reaches $10^6 - 1$. For an input clock frequency of 100 MHz, 'z' is asserted every 0.01 s.
  - ✓ BCD counter: It counts up to 9, asserting 'z' when the count reaches 9.
  - ✓ Modulo-6 counter: It counts up to 5, asserting 'z' when the count reaches 5.
- *NEXYS3 implementation details*: Only one 7-segment display can be used at a time → we serialize the four BCD outputs. In order for each digit to appear bright and continuously illuminated, we illuminate each digit for only 1ms every 4 ms. This is taken care of feeding the output 'z' of the counter to 0.001s to the enable input of the FSM.

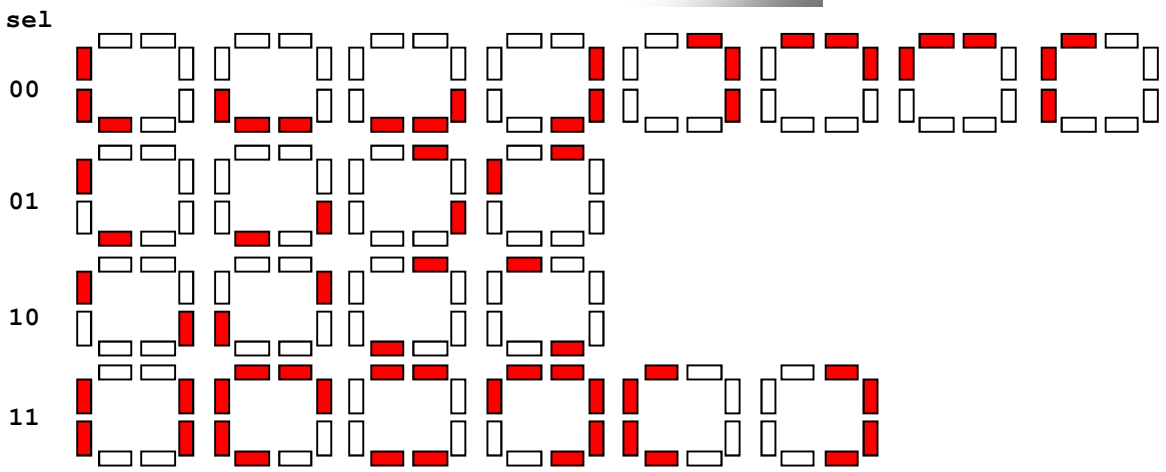


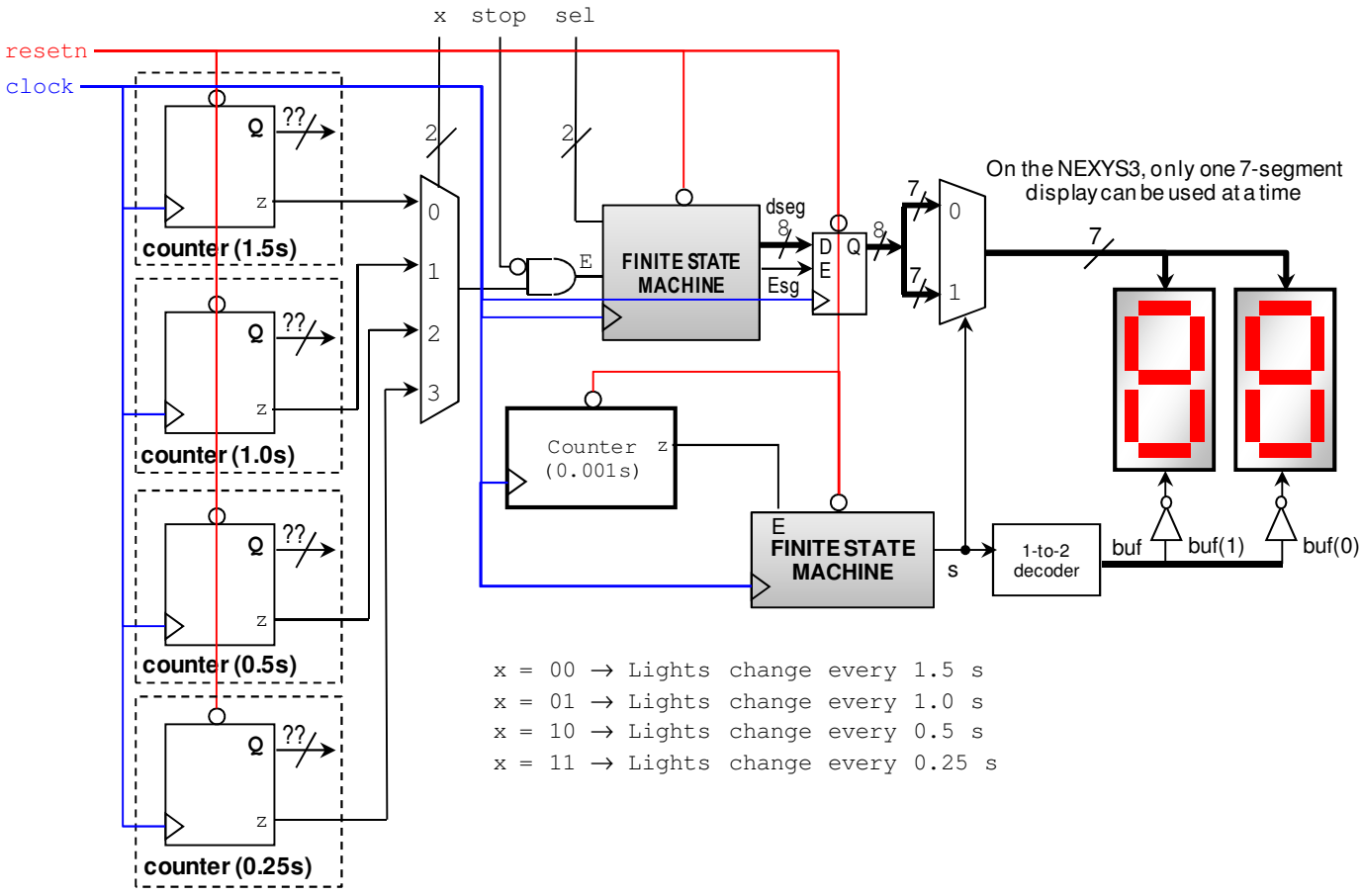
- ✓ Provide the Finite State Machine of the serializer in ASM (Algorithmic State Machine) form.
- ✓ Provide the ***VHDL description*** of the entire circuit: FSM + Datapath circuit.
  Tip: If you want to simulate your circuit, do not include the Counter to 0.01s and the Counter to 0.001s (instead, set the output signal 'z' to '1'). Otherwise, you might not be able to simulate your circuit.

PROBLEM 2 (25 PTS)

- Design of a configurable lights' pattern generator. `sel`: selects the pattern. `stop`: freezes the pattern when `stop=1`. Two 7-segment displays are used. The datapath circuit is provided.
- Input '`x`': Selects the rate at which the lights' pattern change (every 1.5, 1.0, 0.5, or 0.25 seconds)





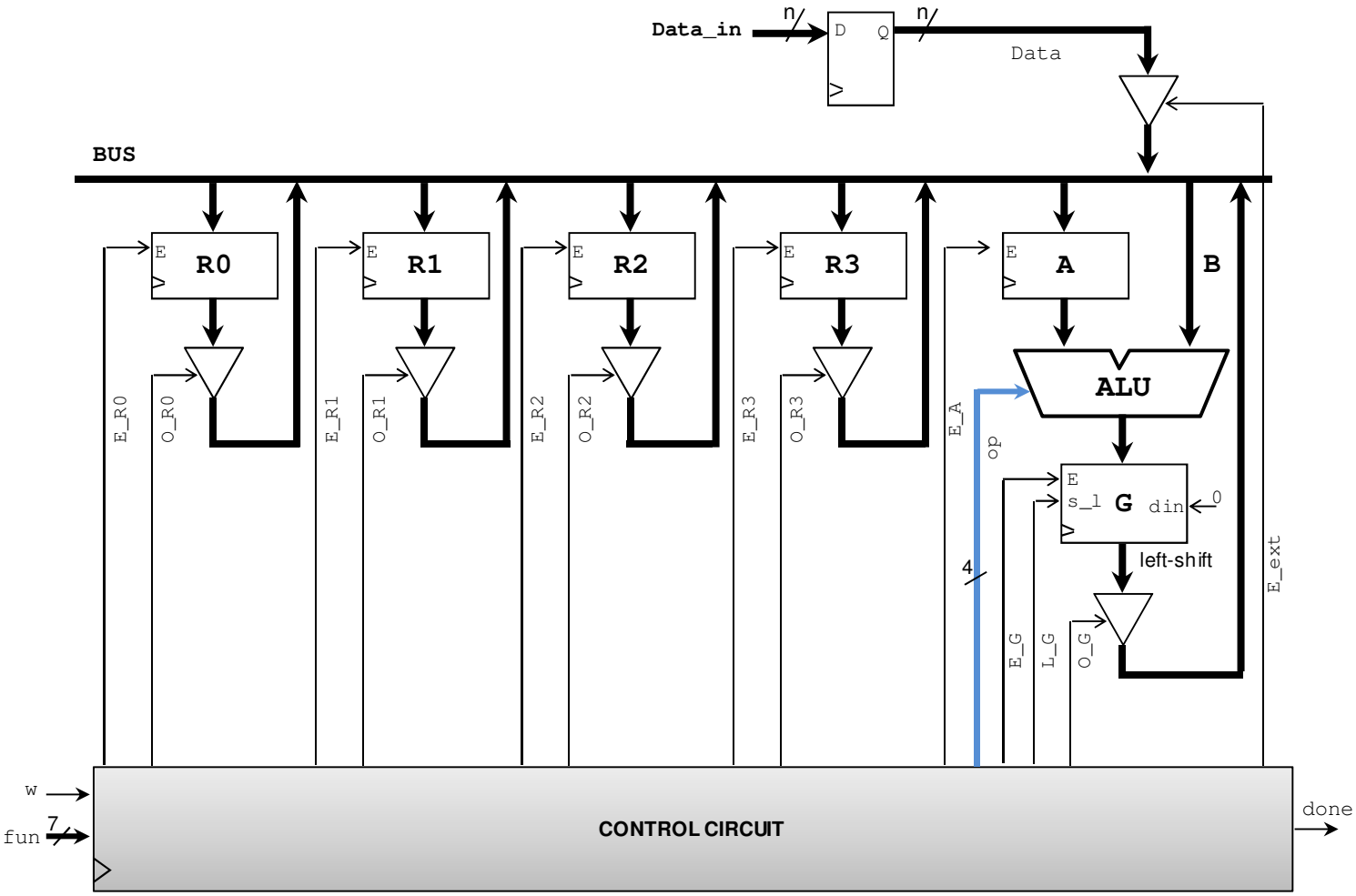



```
x = 00 → Lights change every 1.5 s
x = 01 → Lights change every 1.0 s
x = 10 → Lights change every 0.5 s
x = 11 → Lights change every 0.25 s
```

- ✓ Provide both Finite State Machines in ASM form. ASM: Algorithmic State Machine.
- ✓ Provide the VHDL description of the entire circuit: FSMs + Datapath circuit.

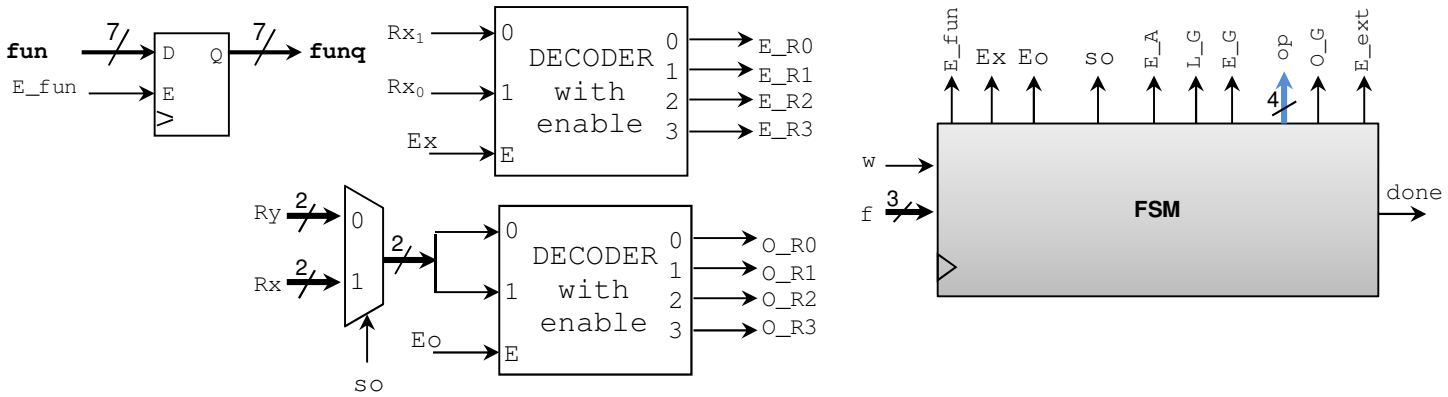
PROBLEM 3 (20 PTS)

- The figure depicts the architecture of a simple processor.



- Register G: Parallel Access left-shift register with enable. `s_l = 1` → Load, `s_l = 0` → Left-shift
- The figure below depicts the datapath and the FSM of the Control Circuit:

$funq = |f_2|f_1|f_0|Ry_1|Ry_0|Rx_1|Rx_0|$



- The following table specifies the behavior of the Arithmetic-Logic Unit (ALU). This ALU is purely combinatorial.

op	Operation	Function	Unit
0000	y <= A	Transfer 'A'	Arithmetic
0001	y <= A + 1	Increment 'A'	
0010	y <= A – 1	Decrement 'A'	
0011	y <= B	Transfer 'B'	
0100	y <= B + 1	Increment 'B'	
0101	y <= B – 1	Decrement 'B'	
0110	y <= A + B	Add 'A' and 'B'	
0111	y <= A – B	Subtract 'B' from 'A'	
1000	y <= not A	Complement 'A'	Logic
1001	y <= not B	Complement 'B'	
1010	y <= A AND B	AND	
1011	y <= A OR B	OR	
1100	y <= A NAND B	NAND	
1101	y <= A NOR B	NOR	
1110	y <= A XOR B	XOR	
1111	y <= A XNOR B	XNOR	

- Operation: Every time w = '1', we store the input fun, then proceed to execute it.

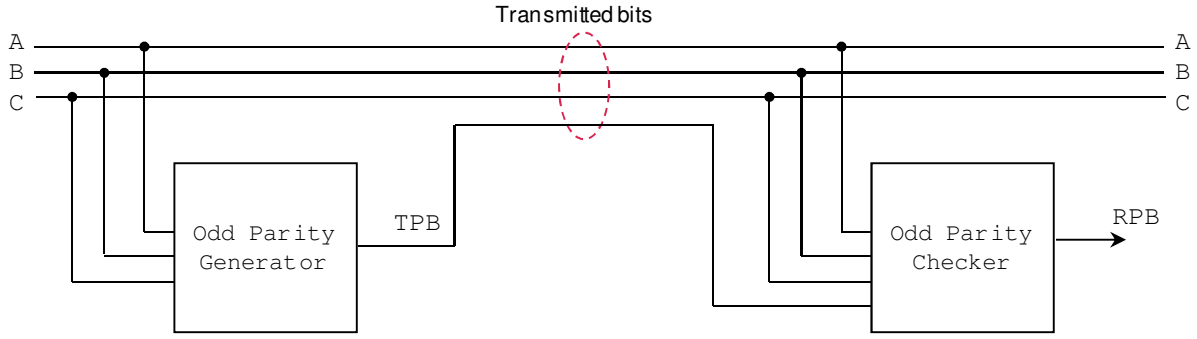
- The 7 bits of the stored input, called $funq$, are arranged as: $funq = |f_2|f_1|f_0|Ry_1|Ry_0|Rx_1|Rx_0|$. The first 3 bits specify the operation, while the other 4 bits specify the registers over which the operations are applied.

f	Operation	Function
000	Load Rx, Data	Rx ← Data
001	Add Rx, Data	Rx ← Rx + Data
010	Not Rx	Rx ← NOT (Rx)
011	Sub Rx, Ry	Rx ← Rx – Ry
100	Add Rx, Ry	Rx ← Rx + Ry
101	Move Rx, Ry	Rx ← Ry
110	sla Rx	Rx ← left-shift Rx
111	Addi Rx, 2	Rx ← Rx + 2

✓ Design the Finite State Machine (provide it in ASM form) that can issue the correct set of signals for all the listed operations.

PROBLEM 4 (10 PTS)

- For the following error-detection system, provide the truth table and sketch the circuits of i) the Odd Parity Generator, and ii) Odd parity checker.

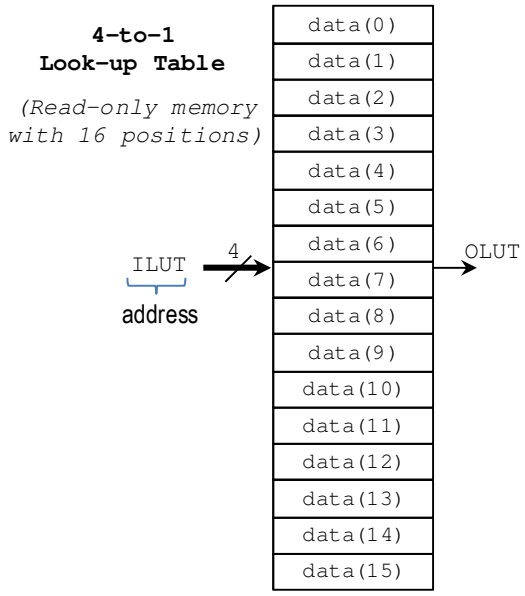


- Sketch the schematics of an n-bit comparator of two numbers represented in 2's complement.
  Required outputs: `AgeB, AlB, AeB`. If `A >= B` → `AgeB = 1`. If `A < B` → `AlB = 1`. If `A = B` → `AeB = 1`. You can use adders (specify the number of bits) and XOR gates.
  Make sure that your circuit works in all circumstances (consider that A and B may cause overflow, you need to avoid it).

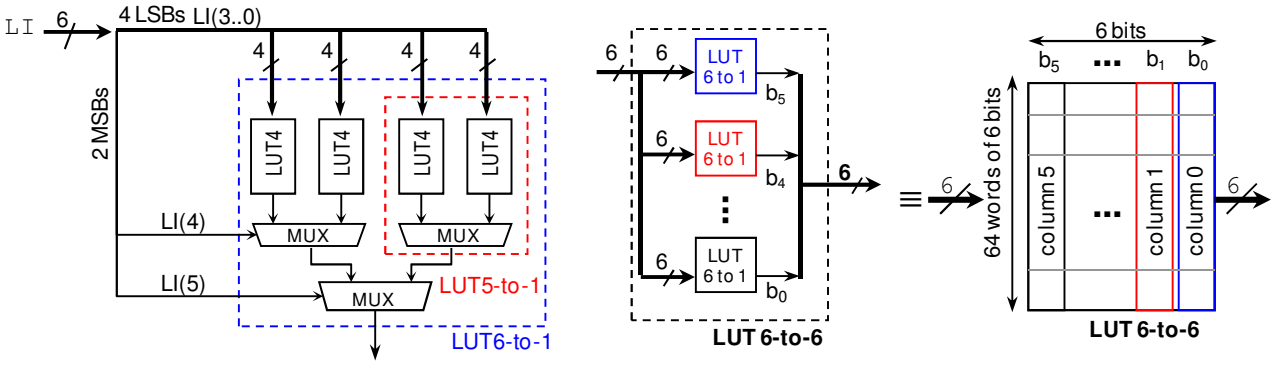
PROBLEM 5 (25 PTS)

- **LUT connected to a bidirectional port**: An LUT4-to-1 (also called LUT4) can implement a 4-input function. The VHDL code is provided below. Note that the values stored in the LUT are constant, hence those values are entered as a parameter in the VHDL code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity my4to1LUT is
        generic (data: std_logic_vector(15 downto 0):=x"FEAB"); -- LUT 4-to-1 contents
        port ( ILUT: in std_logic_vector (3 downto 0);
               OLUT: out std_logic);
end my4to1LUT;

architecture Behavioral of my4to1LUT is

begin
        with ILUT select
             OLUT <= data(0) when "0000",
                     data(1) when "0001",
                     data(2) when "0010",
                     data(3) when "0011",
                     data(4) when "0100",
                     data(5) when "0101",
                     data(6) when "0110",
                     data(7) when "0111",
                     data(8) when "1000",
                     data(9) when "1001",
                     data(10) when "1010",
                     data(11) when "1011",
                     data(12) when "1100",
                     data(13) when "1101",
                     data(14) when "1110",
                     data(15) when "1111",
                    '0' when others;

end Behavioral;
```



- LUTs with more than four inputs can be built by grouping several LUT 4-to-1 and MUXs. The figure below shows how an LUT 6-to-1 can be built.
- An LUT with more than one output can also be built. The figure below show how we can create an LUT 6-to-6 (we just use six LUT 6-to-1).



You are asked to design a system with an LUT 6-to-6. This will be accomplished in a series of steps:

✓ Provide the VHDL code of an LUT 6-to-1. You can built it by i) grouping four LUT4-to-1 and MUXs, or ii) using only one VHDL file (similar to `my4to1LUT.vhd`). The entity should look like this:

```
entity my6to1LUT is
        generic (data: std_logic_vector(63 downto 0):=x"FEAB97CA003E19CC"); -- 64 values
        port ( ILUT: in std_logic_vector (5 downto 0);
               OLUT: out std_logic);
end my6to1LUT;
```

- ✓ Write the VHDL code for an LUT 6-to-6. You must built it by grouping six LUT 6-to-1. The entity should look like this:

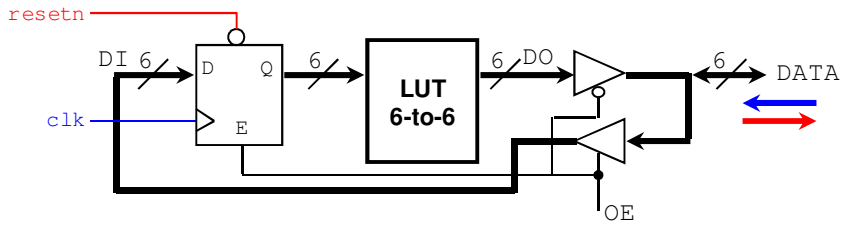
```
entity my6to6LUT is
        generic (data5: std_logic_vector(63 downto 0):=x"FEAB97CA003E19CC";  -- column 5
                 data4: std_logic_vector(63 downto 0):=x"AABBCCFFEE99098A";  -- column 4
                 data3: std_logic_vector(63 downto 0):=x"E595BEBECAFEDADA";  -- column 3
                 data2: std_logic_vector(63 downto 0):=x"FACE09093E3EECAB";  -- column 2
                 data1: std_logic_vector(63 downto 0):=x"DECAFFFF09EA3200";  -- column 1
                 data0: std_logic_vector(63 downto 0):=x"ACADE412BAFE125E"); -- column 0
        port ( ILUT: in std_logic_vector (5 downto 0);
               OLUT: out std_logic_vector (5 downto 0));
end my6to6LUT;
```

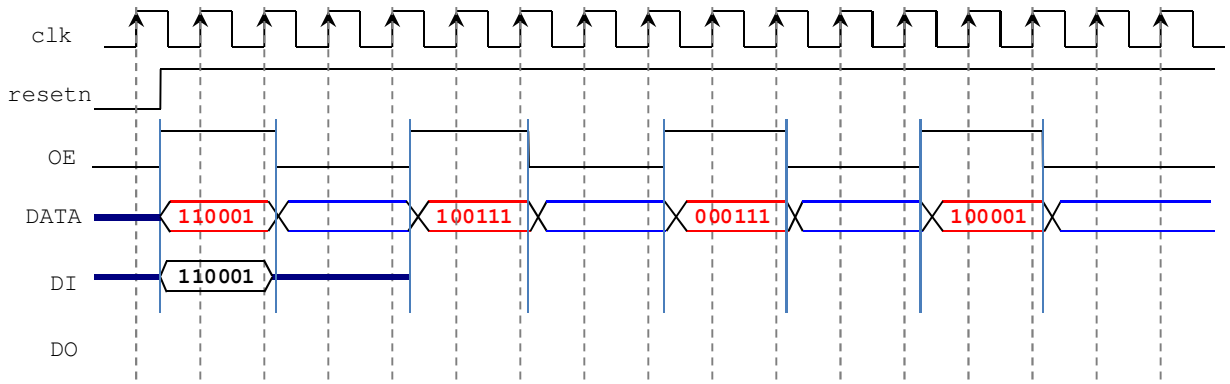
  Important: When instantiating the `my6to1LUT` component, we use the `port map` instruction to make interconnections. Now, we also need to provide the correct parameter to each `my6to1LUT` component. This is done usually the `generic map` instruction.

- ✓ **Final System:** Provide the VHDL code of the circuit depicted below. Use the Structural Description by interconnecting the following components: i) LUT 6-to-6, ii) 6-bit register, and iii) Tri-state buffers. Important: The port 'DATA' can be input or output at different times. Use INOUT in your VHDL code.

- ✓ *LUT6-to-6 contents*: We want this LUT 6-to-6 to provide the following function: $OLUT = \lceil ILUT^{0.95} \rceil$
  Example: $ILUT = 35\ (010011_2) \rightarrow OLUT = \lceil 35^{0.95} \rceil = 30\ (011110_2)$
  Compute the contents of the LUT 6-to-6 and provide each column in hexadecimal format as:
  `data5, data4, data3, data2, data1, data0` (generic input parameters)

- ✓ Complete the Timing diagram shown below. Note that the port DATA is input at some times, and output at other times.





EXTRA CREDIT (+10 PTS)

- Demonstrate the circuits of Problems 1 and 2 working on the NEXYS3 board.