FINAL REVIEW

A1Q1 write VHDL description of an 8-to-1 two-bit mux

```
library ieee;
use ieee.std_logic_1164.all;

entity MUX is
   port(a,b,c,d,e,f,g,h: in std_logic_vector(1 downto 0);
     sel: in std_logic_vector(2 downto 0);
     output: out std_logic_vector(1 downto 0)
   );
end MUX;

architecture BEHAVIOURAL of MUX is
begin
   process(sel,a,b,c,d,e,f,g,h) --combinational
   begin
      case sel is
         when "000" => output <= a;
            .....
         when others => output <= h;
      end case;
   end process;
end BEHAVIOURAL;
```

AIQ2 write VHDL of 3-input 8-output decoder

a decoder takes an input # and sets that # bit off the output to one (asserts) and all other bits to zero

```
library ieee;
use ieee.std_logic_1164.all;

entity DECODER is
    port(
        input: in std_logic_vector(2 downto 0);
        output: out std_logic_vector(7 downto 0)
    )
end DECODER;

architecture BEHAVIOURAL of DECODER is
begin
  process(input)
  begin
    case input is
        when "000" => y <= "00000001";
        when "001" => y <= "00000010";
            . . . . .
        when others => y <= "10000000";
    end case;
  end process;
end BEHAVIOURAL;
```

AIQ3 write VHDL for the following table

En	w_1	w_2	y_0	y_1	y_2	y_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	x	x	0	0	0	0

```
library ieee;
use ieee.std_logic_1164.all;

entity DECODER is
    port(
        w : in std_logic_vector(1 downto 0);
        en : in std_logic;
        y : out std_logic_vector(3 downto 0)
    );
end DECODER;

architecture BEHAVIORAL of DECODER is
begin
    process (en, w)
    begin
        if en='1' then
            case w is
                when "00" => y<= "1000";
                when "01" => y<= "0100";
                . . .
            end case;
        else
            y <= "0000";
        end if;
    end process;
end behavioral;
```

AIQ4 what is wrong with the following code for a decoder

```
entity demux is
      port( a   : in std_logic;
            sel : in std_logic_vector( 2 downto 0 );
            q   : out std_logic_vector( 7 downto 0 ) );
end demux;
architecture eece353 of demux is
begin
      process(sel,a)
      begin
            case(sel) is
                  when "000" => q(0) <= a;
                  when "001" => q(1) <= a;
                  ...
                  when others => q(7) <= a;
            end case;
      end process;
end eece353;
```

it's not driving all other bits of q with logic

to fix it put q<="00000000"; at the beginning.

to expand on this, the current code would tell the circuit to maintain their old values which would imply memory so it would create registers

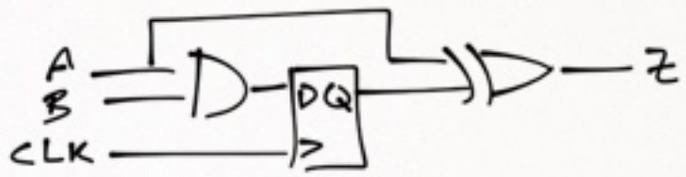
AIQ5 write VHDL for JK-flip-flop

J	K	Q
0	0	previous val of Q
0	1	0
1	0	1
1	1	Q'

```
entity JKFF is
    port(
        clk, j, k : in std_logic;
        q : out std_logic
    );
end JKFF;

architecture BEHAVIOURAL of JKFF is
begin
    process(clk);
        variable temp_q : std_logic;
    begin
        if rising_edge(clk) then
            if J='0' then
                if k='1' then
                    temp_q := '0';
                end if;
            else
                if k='0' then
                    temp_q := '1';
                else
                    temp_q := not temp_q;
                end if;
            end if;
        end if;
        q<=temp_q;
    end process;
end BEHAVIOURAL;
```

-- I forgot to use a variable to store information without reading in an out bit

A1Q6 write VHDL for following circuit

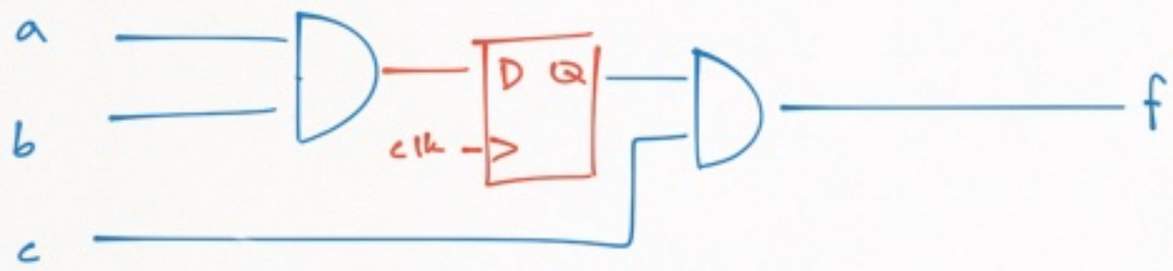



```
architecture BEHAVIOURAL of CIRCUIT is
begin
signal temp_q : std_logic;
    process(clk)
    begin
        if rising_edge(clk) then
            temp_q <= a and b;
        end if;
    end process;

    process (q, temp_q)
    begin
       z <= a xor temp_q;
    end process;
end BEHAVIOURAL;
```

-- I made the whole thing synchronous when the Z is sensitive to a

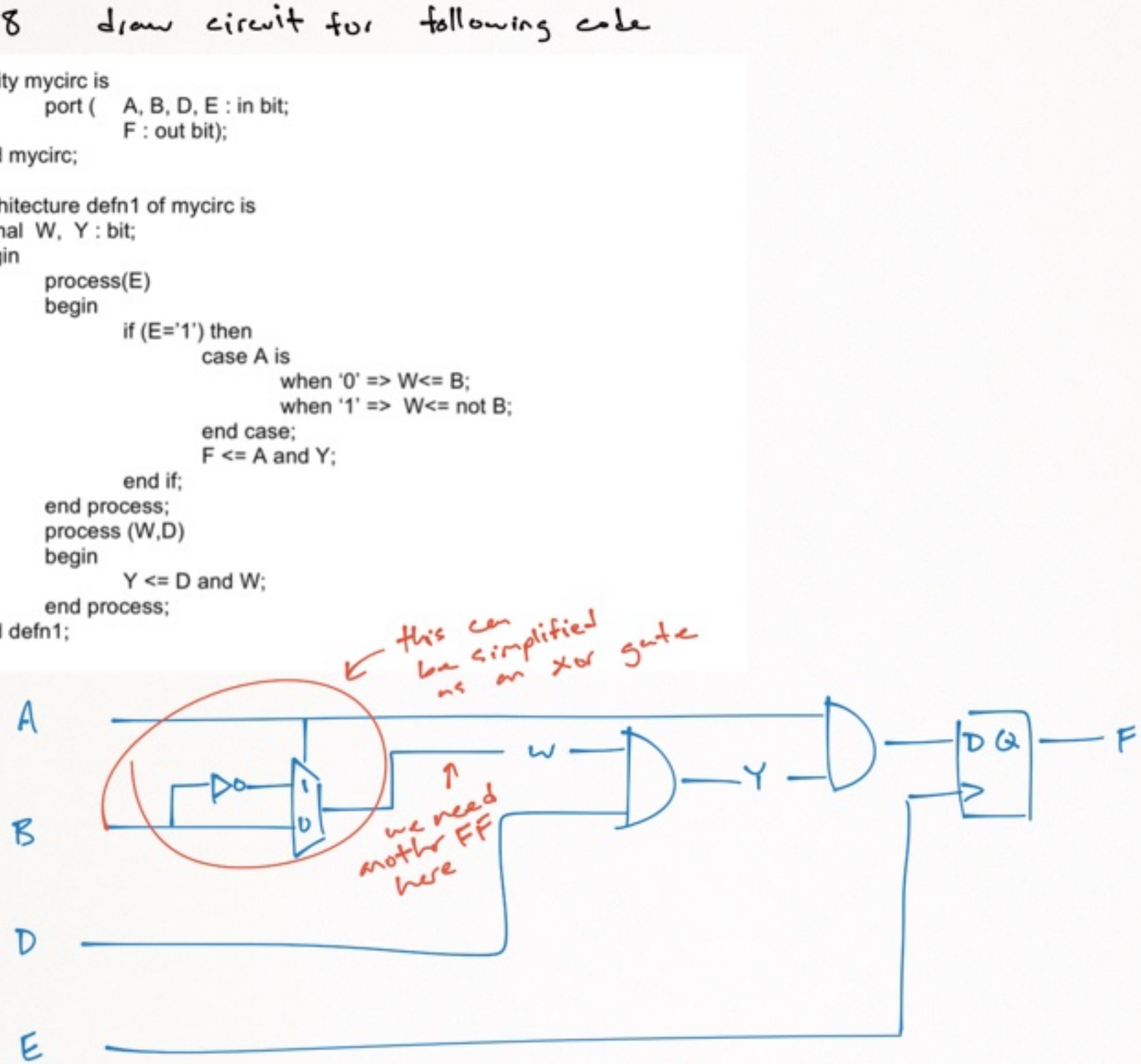
A1Q7 draw circuit for the following code

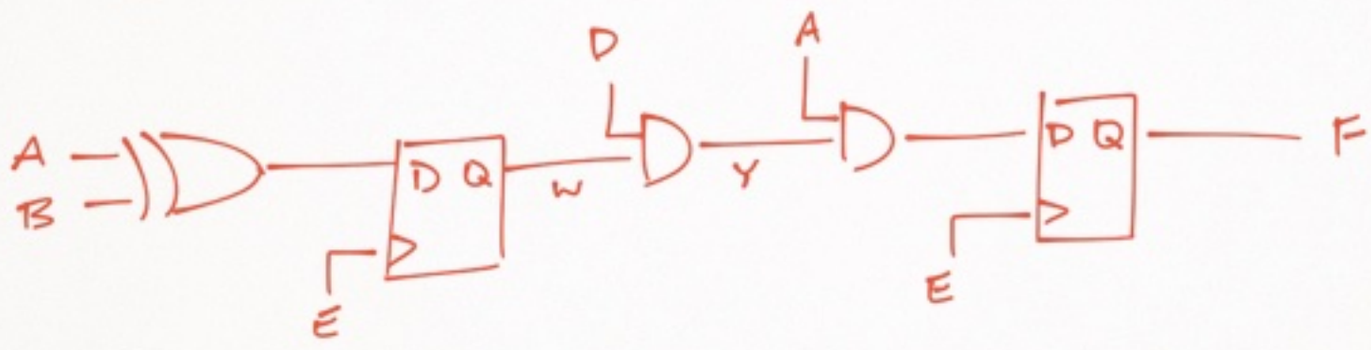
```
entity mycirc is
        port (    A, B, C, CLK : in bit;
                  F : out bit);
end mycirc;

architecture defn1 of mycirc is
signal  W : bit;
begin
        process(CLK)
        begin
                if (CLK='1') then
                    W <= A and B;
                end if;
        end process;

        F <= W and C;

end defn1;
```



AIQ8 draw circuit for following code

```
entity mycirc is
        port (    A, B, D, E : in bit;
                  F : out bit);
end mycirc;

architecture defn1 of mycirc is
signal  W,  Y : bit;
begin
        process(E)
        begin
                if (E='1') then
                        case A is
                                when '0' => W<= B;
                                when '1' =>  W<= not B;
                        end case;
                        F <= A and Y;
                end if;
        end process;
        process (W,D)
        begin
                Y <= D and W;
        end process;
end defn1;
```



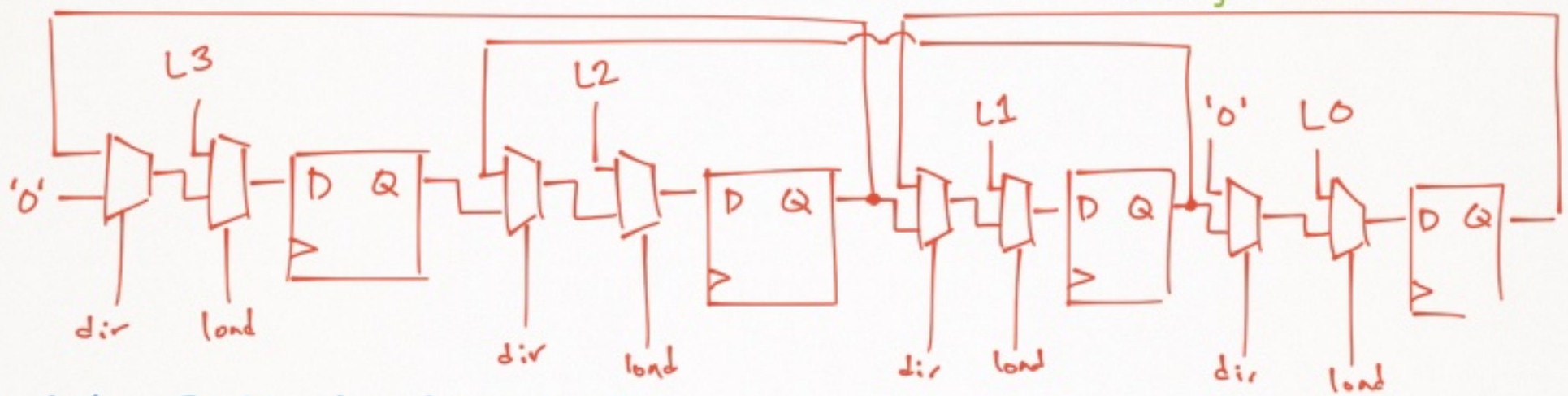


A2Q2 draw schematic of a 4-bit universal shift register. use D-flip-flops, muxs, & gates that has parallel load capability

A2Q3 write VHDL using single process

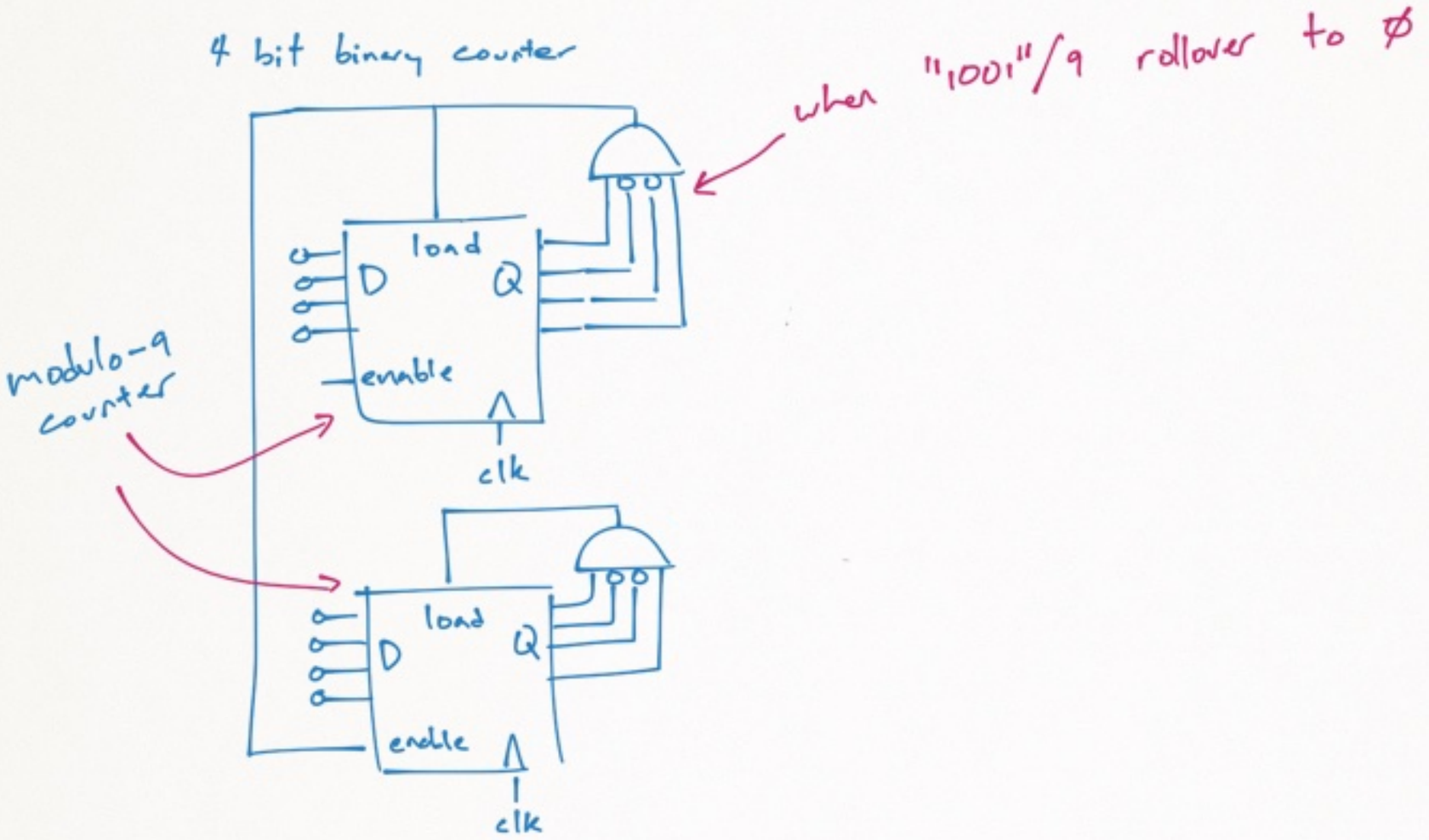
↳ left & right directionality

↳ cascaded FF with each feeding into the next




```
architecture BEHAVIOURAL of SHIFTREG
begin
    process (clk)
        variable reg : std_logic_vector(3 downto 0);
    begin
        if rising_edge(clk) then
            if load ='1' then
                reg := L;
            elsif dir = '0' then
                reg := '0' & reg(3 downto 1);
            else
                reg := reg(2 downto 0) & '0';
            end if;
            outsig <= reg(0);
        end if;
    end process;
end BEHAVIOURAL;
```

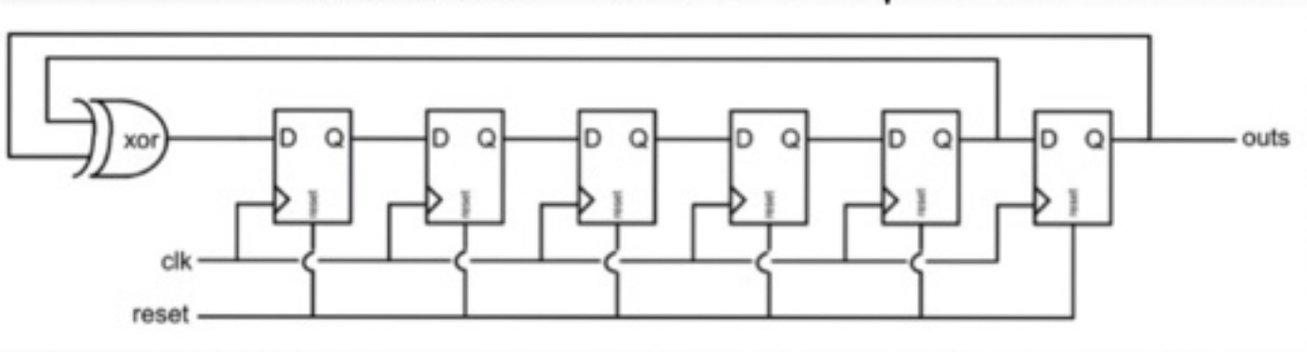
A203 design 8-bit BCD counter using 4 bit binary counters



A2Q4 write VHDL for it

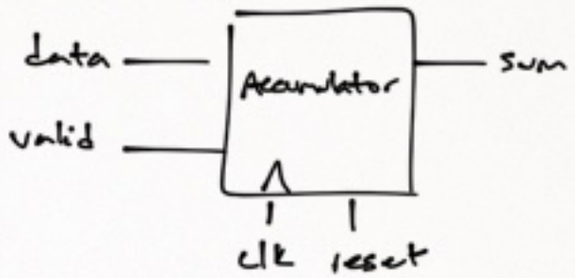
```
architecture BEHAVIOURAL of Q4 is
begin
   process(clk)
      variable BCD0 : unsigned(3 downto Ø) := "0000";
      veriable BCD1 : unsigned(3 downto Ø) := "0000";
   begin
      if rising_edge(clk) then
         if BCD0 = "1001" then
            BCD0 := "0000";
            if BCD1 = "1001" then  BCD1 := "0000";
            else BCD1 := BCD1 + 1;
            end if;
         else
            BCD0 := BCD0 + 1;
         end if;
         output <= std_logic_vector(BCD1 & BCD0);
      end if;
   end process;
end BEHAVIOURAL;
```

A2Q5 consider the following linear-feedback shift register (LFSR), it is asynchronous write VHDL that uses one process




```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numerical.all;   use ieee.numeric_std.all;

entity LFSR is
    port( clk, reset : in std_logic;
          outs : out std_logic
    );
end LFSR;

architecture BEHAVIOURAL of LFSR is
begin
    process( clk, reset )
        variable reg : std_logic_vector(5 downto 0);
    begin
        if reset ='1' then
            reg := "000000";
        else
            if rising_edge(clk) then
                reg := (reg(5) xor reg(4)) & reg(5 downto 1);
                                (reg(1) xor reg(0))
            end if;
        end if;
        outs <= reg(5);
                reg(0);
    end process;
end BEHAVIOURAL;
```

A2Q6 write VHDL for a 16-bit accumulator

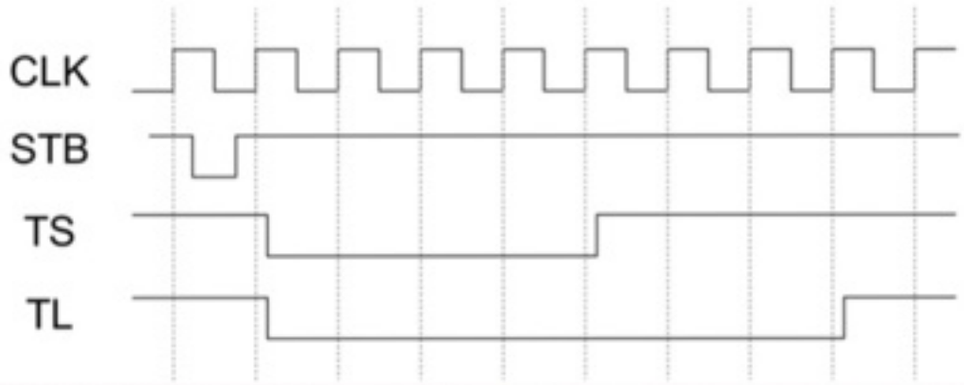


if valid is '1' whatever is stored in accumulator + data is output in sum each clk cycle

```
entity ACCUMULATOR is
    port( clk, reset, valid : in std_logic;
          data : in std_logic_vector(15 downto 0);
          sum : out std_logic_vector(15 downto 0)
    )
end ACCUMULATOR;

architecture BEHAVIOURAL of ACCUMULATOR is
begin
  process (clk)
     variable stored : unsigned(15 downto 0);
  begin
     if rising_edge(clk) then
        if reset = '1' then
            stored := 0;
        else
            if valid ='1' then
                stored := unsigned(data) + stored;
            end if;
        end if;
     end if;
     sum <= std_logic_vector(stored);
  end process;
end BEHAVIOURAL;
```

A2Q7 write VHDL for this:

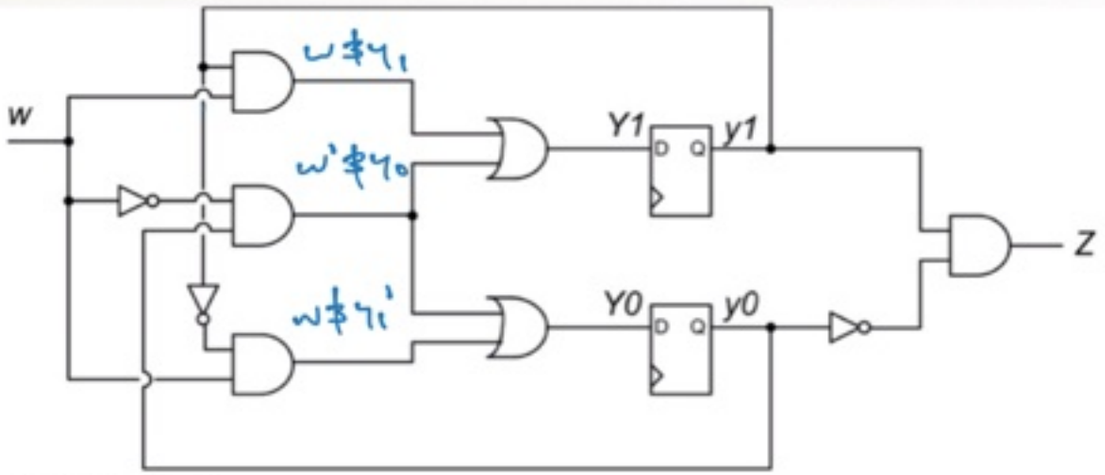
7. Write synthesizable VHDL code to specify the following behaviour. When input STB goes low, outputs TL and TS go low at the next rising clock edge. When input STB goes high, TS goes high after 4 clock ticks, at TL goes high after 7 clock ticks. If STB goes low again before TL goes high, the count should reset (start counting again at 0). (If this was being marked, marks would be deducted for solutions which are excessively complex or long.)




```
architecture BEHAVIOURAL of RANDOM_WAVEFORM is
begin
    process(clk)
      variable counter : integer := 0;
    begin
       if rising_edge(clk) then
            if STB ='0' then
                 TS ='0';
                 TL ='0';
                  counter =0;
            else
                if counter = 4 then
                     TS = '1';
                elsif counter = 7 then
                     TL = '1';
                end if;
                counter = counter + 1;
            end if;
      end if;
```

```
architecture BEHAVIOURAL of TIMER is
    signal count: unsigned(3 downto 0) := "0000";
begin
    process(clk,STB)
    begin
        if STB = '0' then
           count <= "1000";
        elsif rising_edge(clk) then
           if count > 0 then
              count <= count - 1;
           end if;
        end if;
    end process;

    process(count)
    begin
        if count >3 and count <8 then TS<='0';
        else TS <='1';
        end if;
        if count >0 and count <8 then TL<='0';
        else TL <='1';
        end if;
    end process;
```

A3Q1 consider the following circuit



Assume the following:

- Delay of each logic gate: 1 ns
- Set up time of each flip-flop: 0.2 ns
- Hold time of each flip-flop: 0 ns
- Clk-to-Q delay of each flip-flop: 0.5 ns

$t_{cp} = t_{cycle} - t_{su} - t_{clktoq}$

setup requirement:

$T \geq t_{clktoq} + t_{p\text{-}max} + t_{su}$

$t_{p\text{-}min} \geq t_{hold}$

↑ for our purposes, $t_{hold} = 0ns$

a) what's the maximum frequency of the clock in this circuit?

critical path = 0.5ns + 3(1ns) + 0.2ns = 3.7ns

↳ max delay from any flip-flop output to any flip-flop input

$$\text{max frequency} = \frac{1}{\text{critical path}} = \frac{1}{3.7} 10^9 \text{ Hz} = 270 \text{ MHz}$$

b) what's the max possible hold time?

$t_{hold} \leq t_{p\text{-}min}$, $t_{p\text{-}min} = 2(1ns)$

c) write VHDL for the above circuit using single process

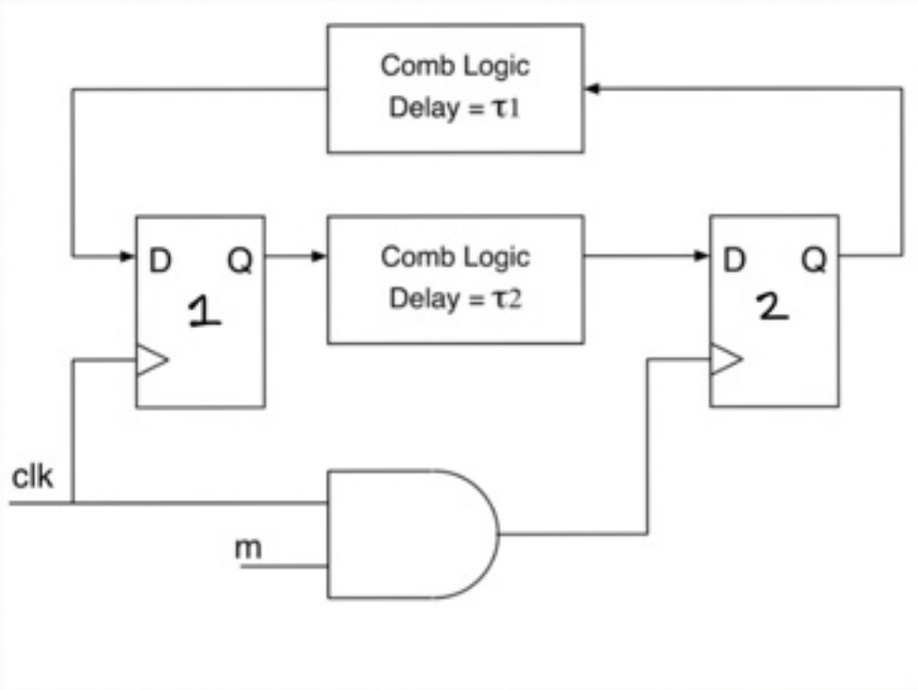
```
if rising_edge(clk) then
    y0 := (not w and y0) or (w and not y1);
    y1 := (not w and y0) or (w and y1);
    z <= not y0 and y1;
```

```
signal y : std_logic_vector(1 downto 0)
process(clk)
variable y_temp : std_logic_vector(1 downto 0)
if rising_edge(clk) then
    y_temp(0) := (not w and y_temp(0)) or ...
    y_temp(1) := (not w and ...
    y <= y_temp;
    z <= not y_temp(0) and y_temp(1);
```

A3Q2 consider the following circuit

Assume the following:

Set up time of each flip-flop: 0.3 ns
Hold time of each flip-flop: 0.2 ns
Clk-to-Q delay of each flip-flop: 0.5
Delay of the AND gate: 1 ns



lets say we want to run this at 100MHz

a) what's the max value of $\tau 1$?

$$T \geq t_{clk\text{-}to\text{-}q} + t_{p\text{-}max} + t_{su}$$

$$\frac{1}{100}10^{-6} = 10ns \geq 0.5ns + t_{p\text{-}max} + 0.3ns$$

$$t_{p\text{-}max} = 9.2ns$$

HOWEVER, NOTE THE CLK REACHES FF1 1ns EARLY RELATIVE TO THE CLK AT FF2

SO $t_{p\text{-}max} = 9.2ns - 1ns = \underline{8.2ns}$

b) what's the maximum $\tau 2$?

same as above but the clock reaches FF2 1ns late as oppored to early relative to FF1

$t_{p\text{-}max} = \underline{10.2ns}$

c) min value of $\tau 1$?

for $\tau 1$, the signal at FF1 will always be delayed 1ns from the signal at FF2, so

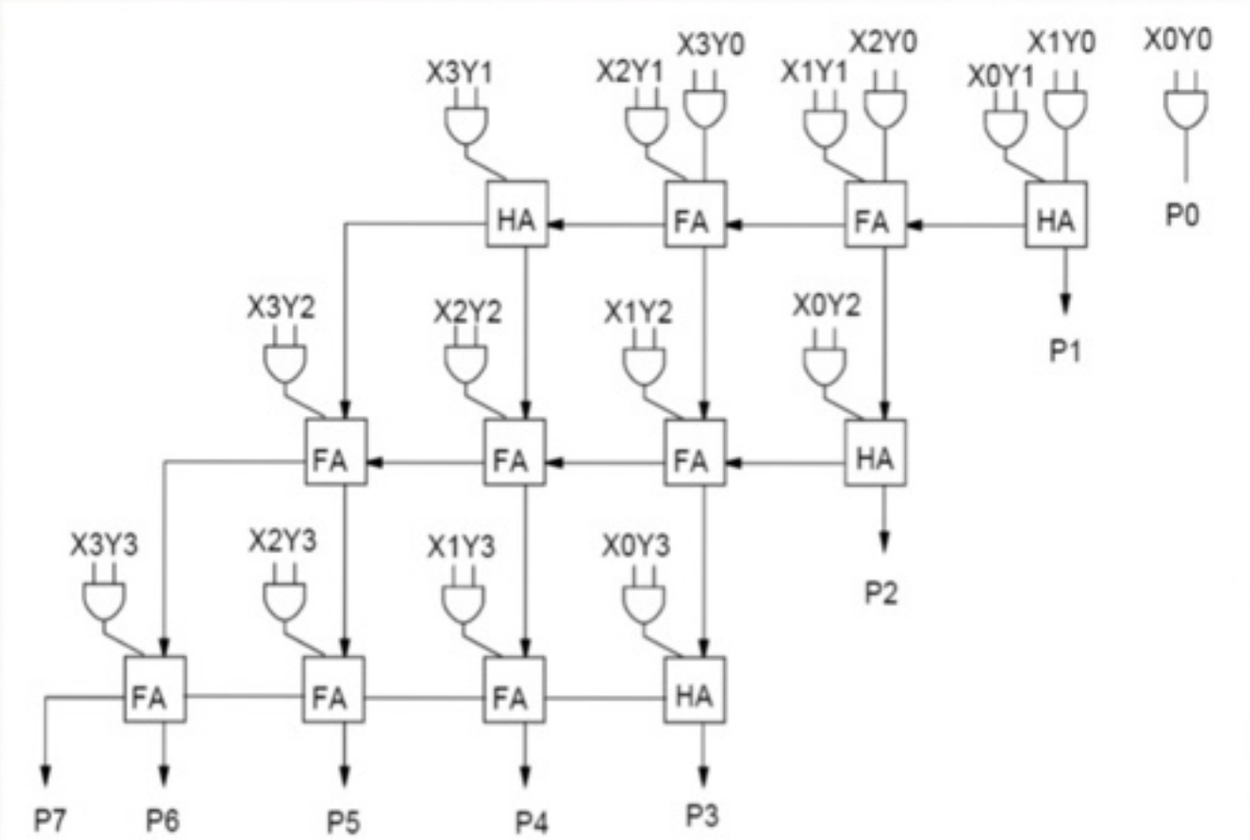
$t_{p\text{-}min} \geq t_{hold}$ $t_{p\text{-}min} = 1ns + \tau_1 \geq 0.2ns$ so $\underline{\tau 1 = \emptyset}$ and the condition still be satisfied

d) min $\tau 2$? $t_{p\text{-}min} \geq 0.2ns + 1ns = \underline{1.2ns}$

A3Q3 processor run @ 2GHz, delay of logic gates is 0.08ns
setup, hold, and clk-to-q time = 0. what's max number of gates in critical path?

$\frac{1}{2GHz} = 0.5(10^{-9}) = 0.5ns = N(0.08ns)$, $N \approx 6$ gates

A3Q4 what's the critical path of the multiplier?

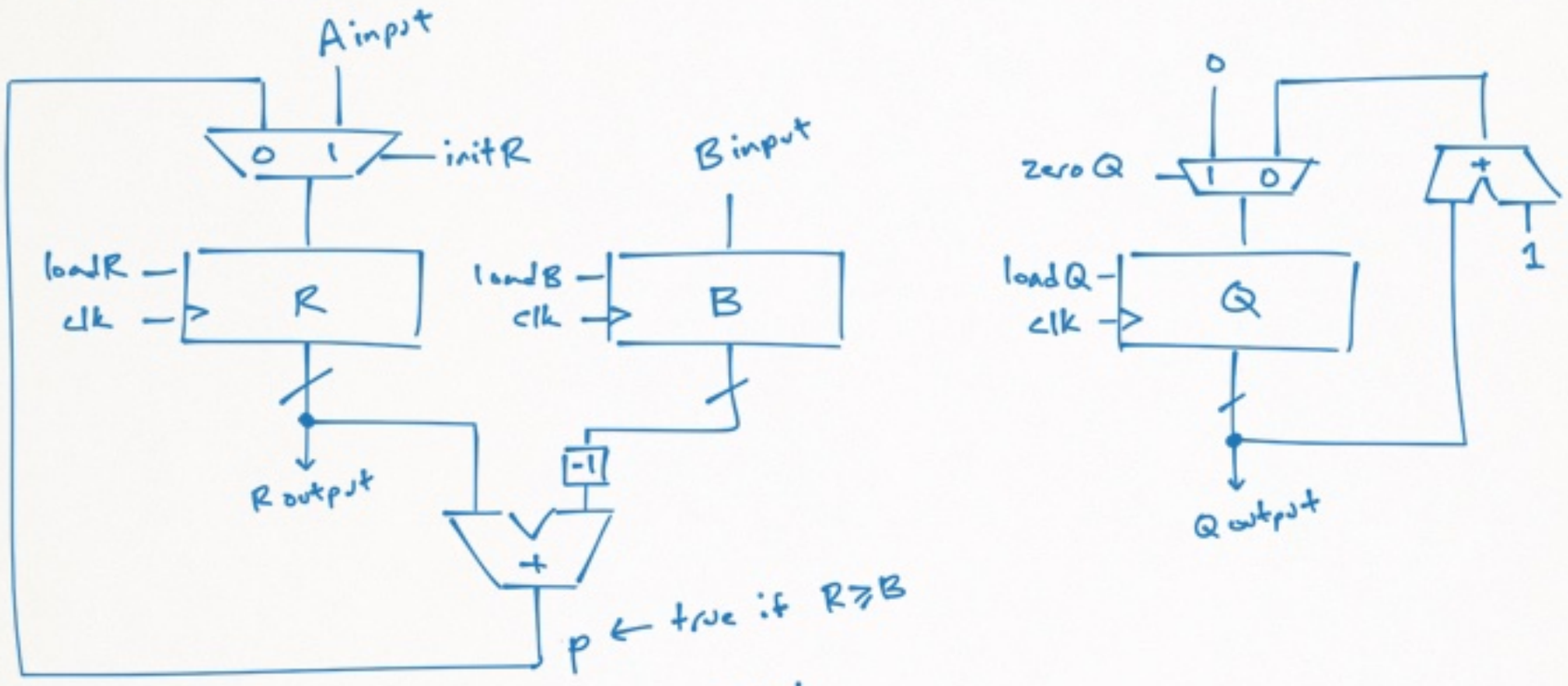


the delay of a F.A. = 3Δ

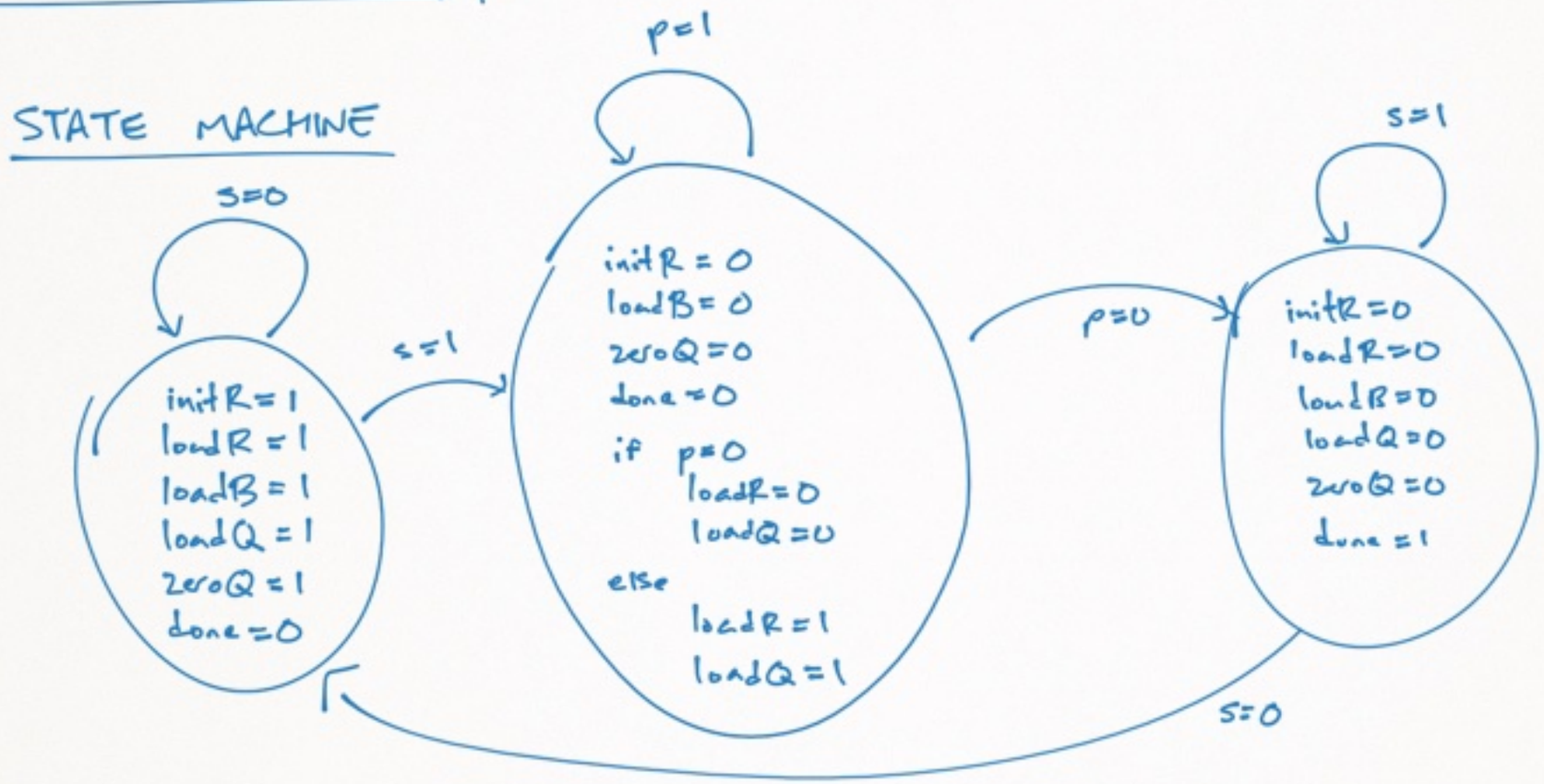
A3Q5 design a circuit to implement a divider using the algorithm

```
Q = 0
R = A
while(R ≥ B){
    R = R - B
    Q = Q + 1
}
```

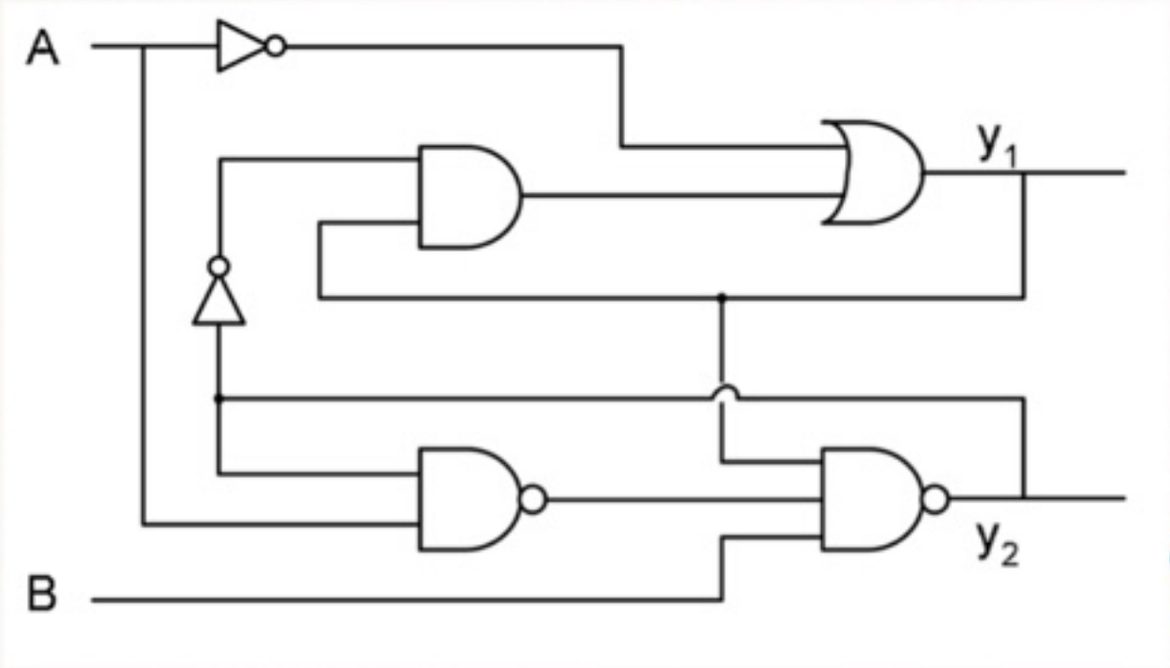
DATAPATH

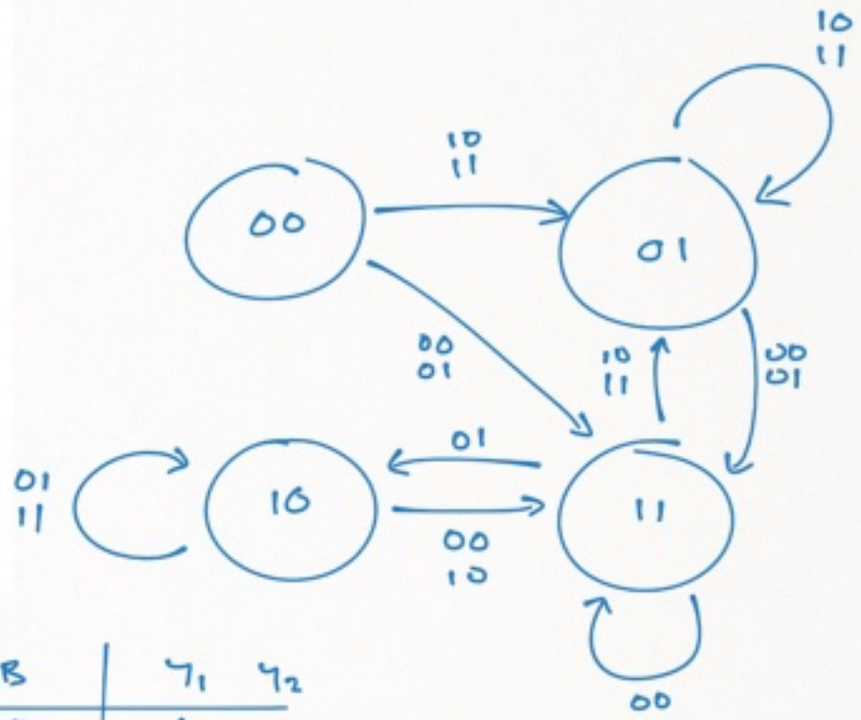


STATE MACHINE



A4Q1 draw state machine for following asynchronous state machine





$$Y_1 = \bar{A} + (\bar{Y}_2 \cdot Y_1)$$

$$Y_2 = \overline{(B \cdot \overline{(A \cdot Y_2)} \cdot Y_1)}$$

$$= \overline{B \cdot Y_1 \cdot (\bar{A} + \bar{Y}_2)}$$

$$= \bar{B} + \bar{Y}_1 + \overline{(\bar{A} + \bar{Y}_2)}$$

$$\overline{\bar{A} + \bar{Y}_2}$$

A=0 Y=0 $\quad \overline{1+1} = \bar{1} = 0$

A=1 Y=0 $\quad \overline{1+0} = \bar{1} = 0$

A=1 Y=1 $\quad \overline{0+0} = \bar{0} = 1$

Y_1	Y_2	A	B	Y_1	Y_2
0	0	0	0	1	1
0	0	0	1	1	1
0	0	1	0	0	1
0	0	1	1	0	1
0	1	0	0	1	1
0	1	0	1	1	1
0	1	1	0	0	1
0	1	1	1	0	1
1	0	0	0	1	1
1	0	0	1	1	0
1	0	1	0	1	1
1	0	1	1	1	0
1	1	0	0	1	1
1	1	0	1	1	0
1	1	1	0	0	1
1	1	1	1	0	1

A4Q2 consider following asynchronous state machine equations

$Y_1 = AB' + Y_1 Y_2 B$

$Y_2 = AB + Y_1$

Are there any static hazards (potential glitches) if these are implemented directly? if so, how can you eliminate them?

Y_1

Y_1Y_2 / AB	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	0	0	1	0
10	1	1	1	1

Y_1Y_2B

AB'

this fixes glitch, add term Y_1Y_2A

these are not covered which poses a hazard

Y_2

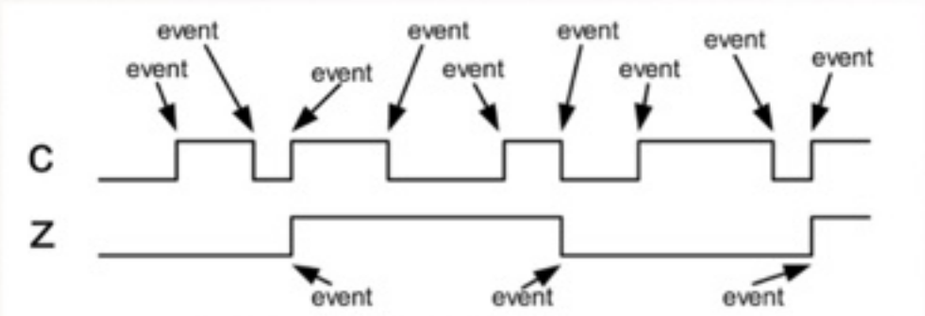
Y_1Y_2 / AB	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	1	1	1	1
10	0	0	1	1

Y_1

AB

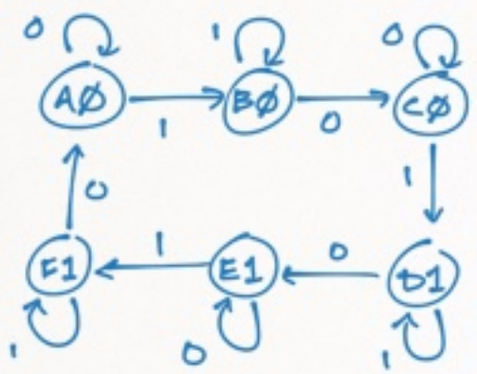
$Y_1 = AB' + AY_1Y_2 + BY_1Y_2$

A4Q3 design an asynchronous state machine



the circuit produces an event on Z for every third event on C

state machine



toggling C moves you through states, every 3 states has different output of Z

states	
A	000
B	001
C	011
D	111
E	101
F	100

consecutive states differ by 1-bit

state	next state C=0	next state C=1	output
A, 000	000	001	0
B, 001	011	001	0
C, 011	011	111	0
D, 111	101	111	1
E, 101	101	100	1
F, 100	000	100	1

y_2 y_1 y_0

y_1 (rows y_2y_1, columns y_0C)

y_2y_1 \ y_0C	00	01	11	10
00	0	0	0	1
01	–	–	1	1
11	–	–	1	0
10	0	0	0	0

$y_1 = y_0\bar{y}_2\bar{C} + y_1C + y_0y_1\bar{y}_2$

y_2 (rows y_2y_1, columns y_0C)

y_2y_1 \ y_0C	00	01	11	10
00	0	0	0	0
01	–	–	1	0
11	–	–	1	1
10	0	1	1	1

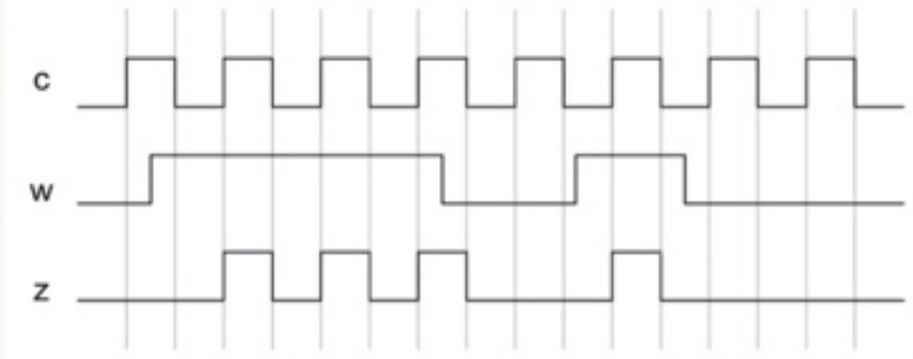
$y_2 = Cy_1 + Cy_2 + y_0y_2$

y_0 (rows y_2y_1, columns y_0C)

y_2y_1 \ y_0C	00	01	11	10
00	0	1	1	1
01	–	–	1	1
11	–	–	1	1
10	0	0	0	1

$y_0 = y_0\bar{C} + y_1y_0 + \bar{y}_2y_0 + \bar{y}_2C$

A4Q4



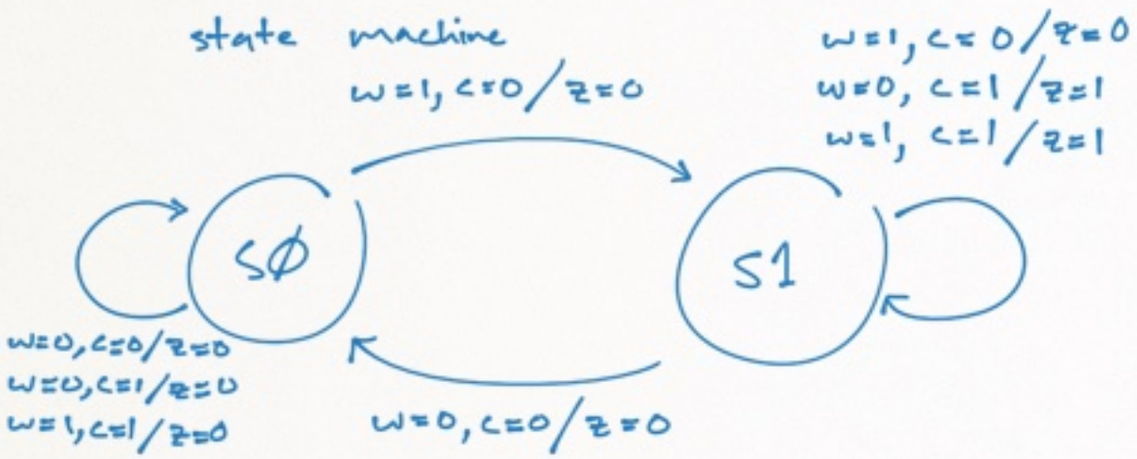
design asynchronous circuit that meets the specifications: clk c, input w, output z

z = clock when w=1, otherwise z=0

if w goes from 0→1, z turns on next rising edge

if w goes from 1→0, z turns on next falling edge

state machine



current state	next state wc=00, 01, 10, 11				output wc=00, 01, 10, 11			
S0	S0	S0	S1	S0	0	0	0	0
S1	S0	S1	S1	S1	0	1	0	1

next state

S \ wc	00	01	11	10
0	0	0	0	1
1	0	1	1	1

output

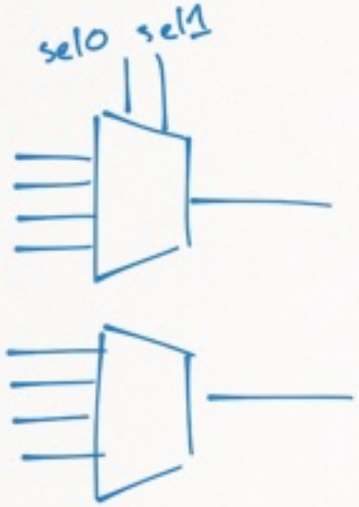
S \ wc	00	01	11	10
0	0	0	0	0
1	0	1	1	0

next state = $SC + SW + W\bar{C}$ output, $z = SC$

A4Q5 write in VHDL

```
entity WAVE is
    port (w,c : in  std_logic;
          z   : out std_logic
    );
entity WAVE;

architecture BEHAVIOURAL of WAVE is
   signal y: std_logic;
begin
   y <= (y and c) or (y and w) or (w and not c);
   y <= y and c;
end BEHAVIOURAL;
```

A4Q6 show how to implement 8-to-1 bit mux using 4-input L.U.T.s



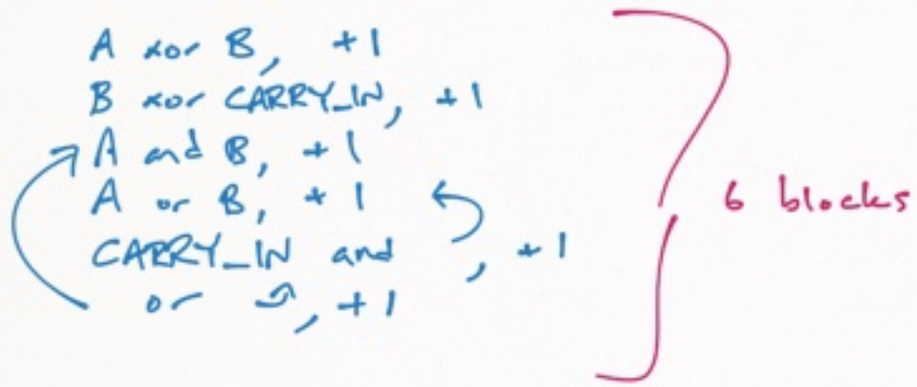
A4Q7 An n-input lookup table can implement ANY function of n inputs
how many different n-input functions are there?

an n-input L.U.T. has 2^n rows

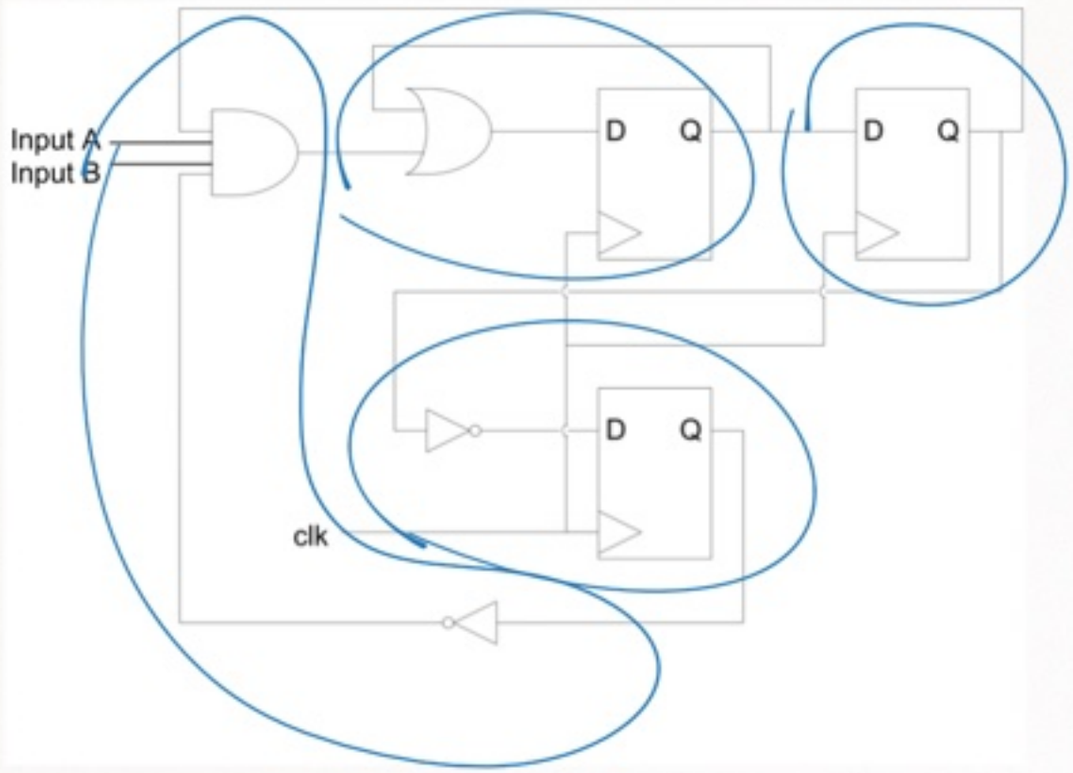
the output of any n-input LUT could be 1 or Ø, so there are 2^{2^n}

A4Q8 consider the code is compiled and mapped to FPGA consisting of 2-input L.U.T.s
How many 2-input L.U.T.s would be required?

```
architecture BEHAVIOURAL of FULL_ADDER is
begin
    SUM <= A xor B xor CARRY_IN;
    CARRY_OUT <= (A and B) or (CARRY_IN and (A or B));
end BEHAVIOURAL;
```

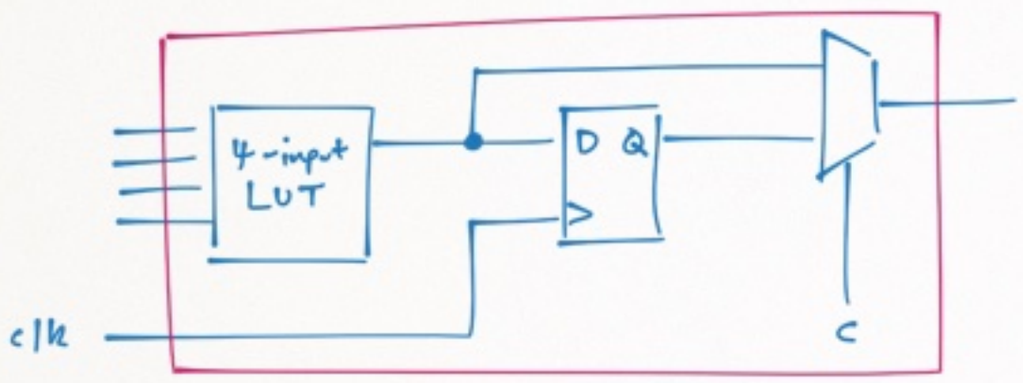


A4Q9



how many FPGA logic blocks are required to implement the circuit?

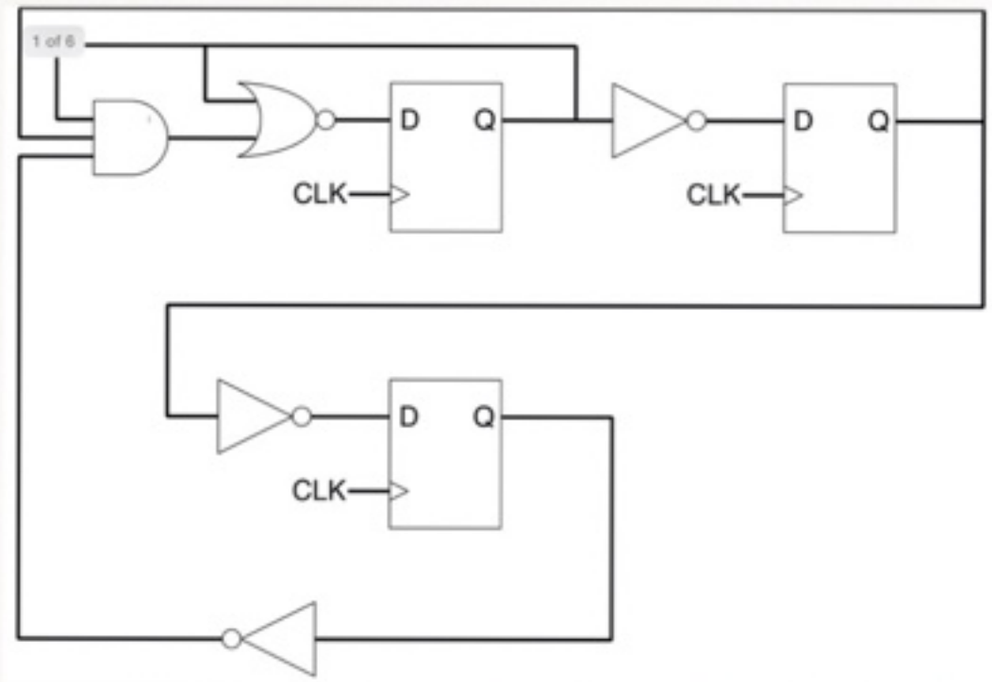
one logic block looks like this:



- this can incorporate a single gate and a flip-flop into one logic block
- the not gate and the and gate can be combined into one logic block
- then the flip-flop can get its own as well

4 logic blocks total

P1Q1



FF hold-time is 0.1ns
FF setup time is 0.2ns
FF clk-to-q is 0.4ns
inverter delay is 0.5ns
AND delay is 1ns
NOR delay is 0.8ns

a) what is the min. clk period

TGSUPMAXCQ, THLM

$T \geq t_{setup} + t_{p\_max} + t_{clktoq} = 0.2ns + 0.4ns + 0.5ns + 1ns + 0.8ns = \underline{2.9ns}$

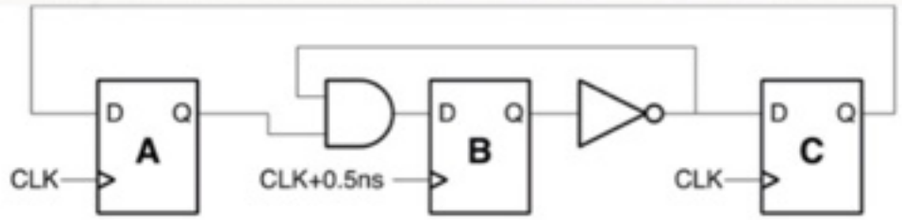
b) what is max clock period?

there is none, $\underline{\infty}$

c) what would be the max hold time?

$t_{hold} \leq t_{p\_min} = \underline{0.5ns}$

P1Q2



Assume the following:

- Hold time of each flip-flop is 0.5ns
- Setup time of each flip-flop is 0.2ns
- Clock-to-Q delay of each flip-flop is 0.1ns
- Delay of the NOT gate is 0.4ns
- Delay of the AND gate is 0.8ns
- As shown on the diagram, the clock connected to flip-flop B always arrives 0.5ns late

a) what's the minimum clock-period of this circuit

$T \leq t_{setup} + t_{p\text{-}max} + t_{clk\text{-}to\text{-}q}$

A→B : 0.8ns − 0.5ns
B→C : 0.4ns + 0.5ns
C→A : 0ns
B→B : 0.4ns + 0.8ns

$T \leq 0.2 + 0.3 + 0.1ns = 0.6ns$
$T \leq 0.2 + 0.9 + 0.1ns = 1.2ns$
$T \leq 0.3ns$
$T \leq 1.2ns + 0.2ns + 0.1ns = 1.5ns$

b) which flip-flops have a hold time violation?

$t_{hold} \leq t_{p\text{-}min}$
$0.5 \leq C \rightarrow A = 0ns$
$0.5 \leq A \rightarrow B = 0.3ns$

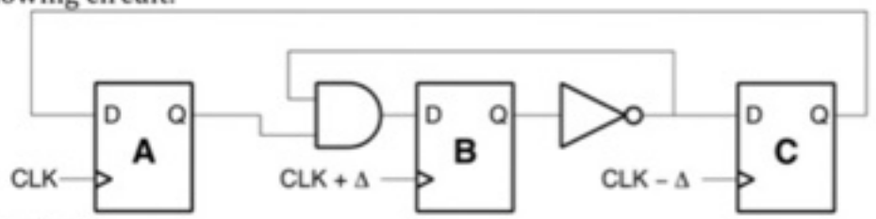
A FF has violation
B FF has violation

c) to fix hold-time violations, what flip-flops should you get?

ones with no hold time

P1Q3

3. Consider the following circuit:



Assume the following:

- Hold time of each flip-flop is 0.25ns
- Setup time of each flip-flop is 0.2ns
- Clock-to-Q delay of each flip-flop is 0.1ns
- Delay of the NOT gate is 0.4ns
- Delay of the AND gate is 0.8ns
- As shown on the diagram, the clock connected to flip-flops B and C suffer from *jitter* causing their clock edges to arrive early or late by an amount Δ; if an edge arrives early for B, it always arrives late for C by the same amount, and vice-versa. For any given clock edge, the value of Δ ranges from –0.2ns to +0.3ns.

a. What is the minimum clock period of this circuit? **You must show your work.**

$T \geq t_{setup} + t_{p\_max} + t_{clktoq}$

A→B: $0.2 + 0.1 + 0.8 - \Delta = 1.1 - \Delta$

B→C: $0.2 + 0.1 + 0.4 + 2\Delta = 0.7 + 2\Delta$

C→A: $0.2 + 0.1 + 0 = 0.3$

B→B: $0.2 + 0.1 + 0.4 + 0.8 = 1.5$

b) which flip-flops have violations?

$t_{hold} \leq t_{p\_min}$

$0.25ns \leq \emptyset$ A ✗

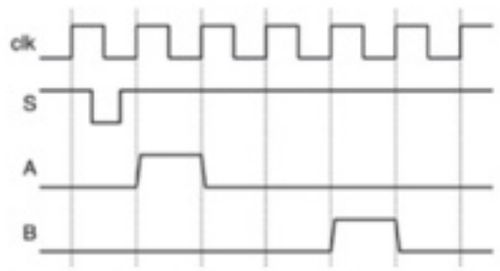
$0.25ns \leq 0.8 - 0.3$ B ✓

$0.25ns \leq 0.4 + 2\Delta = 0.4 - 0.6 = -0.2$ C ✗

c) how to fix?

P1Q5

Write synthesizable VHDL code that corresponds to the behaviour below. When the input signal S goes high (at any time, **irrespective of what the clock is doing**), a timer is started. The timer outputs two 1-cycle pulses: pulse A starts with the first rising clock edge that occurs immediately after S goes high, while pulse B starts on the fourth rising edge of clock after S goes high. A block diagram and timing diagram below illustrate the desired behaviour. (Ignore any metastability issues that this problem might face. Also, assume that S will not go low again until after the timer is finished, ie after the falling edge of B.)




```
library IEEE;
use ieee.std_logic_1164.all;

entity q2 is
  port( S, clk : in std_logic;
        A, B   : out std_logic );
end q2;

architecture behavioural of q2 is
```

```
begin
  signal blah : std_logic;
  process(S)
      if S='1' then
          blah='1';
      end if;
end process;
 process(clk)
  variable counter : std_logic_vector := "00";
    if rising_edge(clk) then
        if blah='1' and counter="00" then
            A='1';
        elsif blah='1' and counter="10" then
            B='1'; counter="00";
        else
            A='0';
            B='0';
        end if;

        counter=counter+1;
end process;
```

this was his solution which seems like bad form because both clk & s are in the same sensitivity list

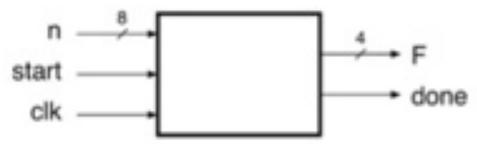
```
process(clk, s)
begin
    if s='0' then
        count <="1000";
    else
        if rising_edge(clk) then
            count<='0' & count(3 downto 0);
            A<=count(3);
            B<=count(0);
        end if;
    end if;
end process;
```

P1Q6

Given an 8-bit unsigned integer n, design a circuit that calculates a 4-bit unsigned integer F, the *square root* of n. i.e. $F = \sqrt{n}$. The algorithm is given below.

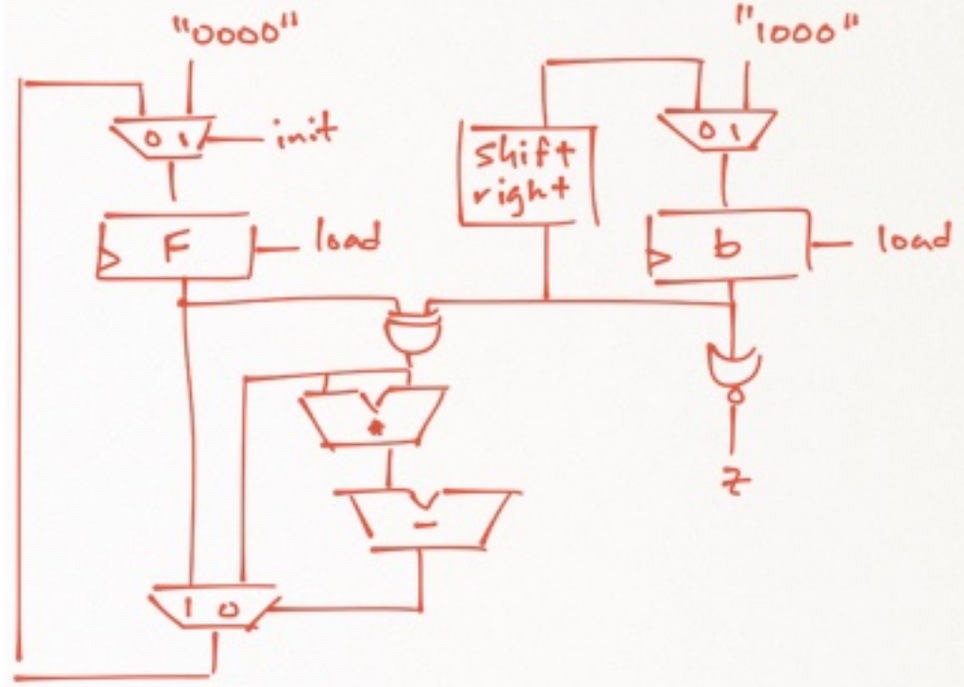
```
b := "1000"          -- b indicates which bit to examine
F := "0000"          -- builds up solution in F
while b != 0
    F := F XOR b     -- sets bit in F
    if (F*F > n) then
       F := F XOR b  -- clears bit from F
    end if
    b := b / 2       -- shifts b right by 1 position
end while
```

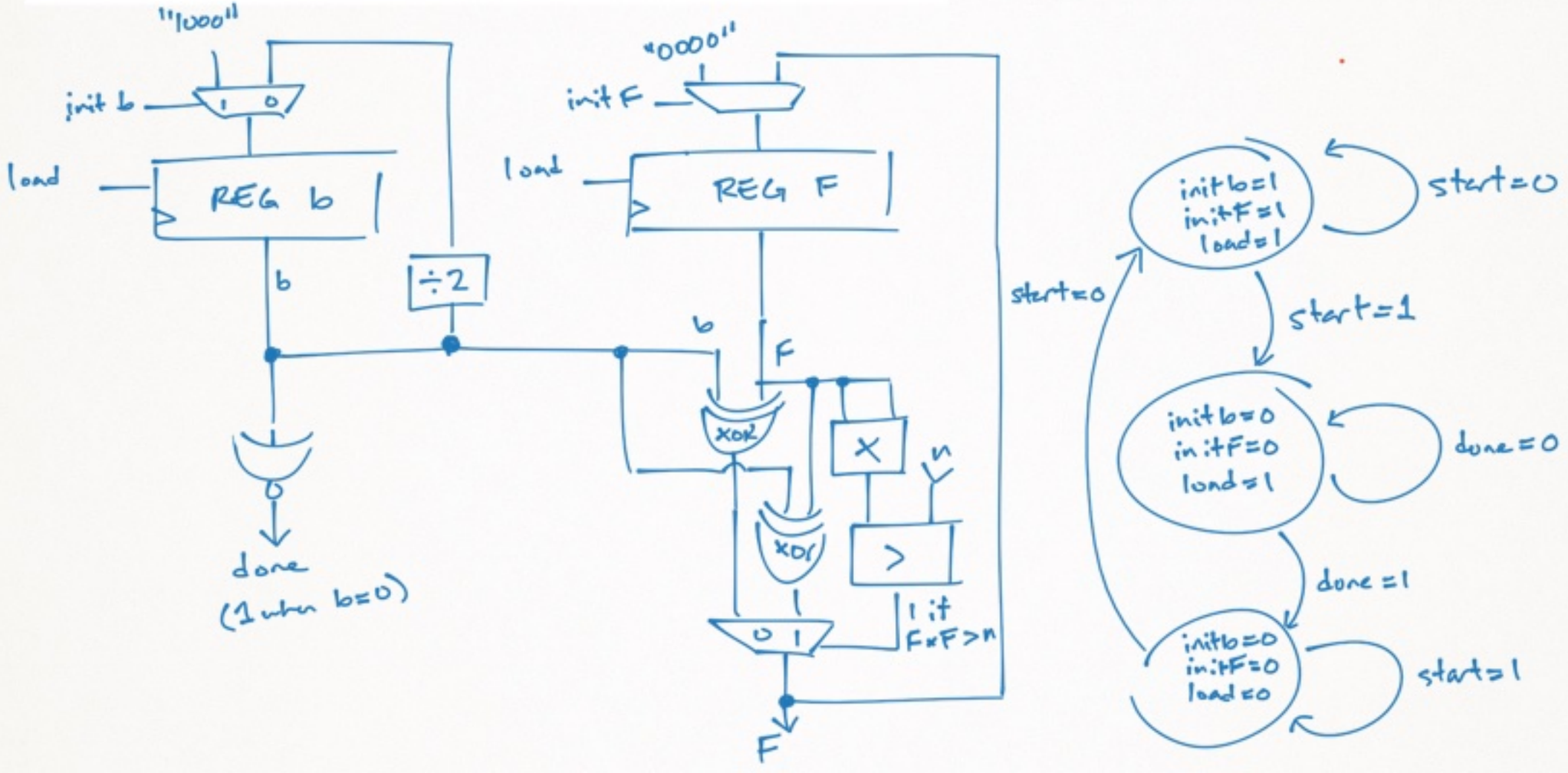
The inputs and outputs of your circuit are as shown below.



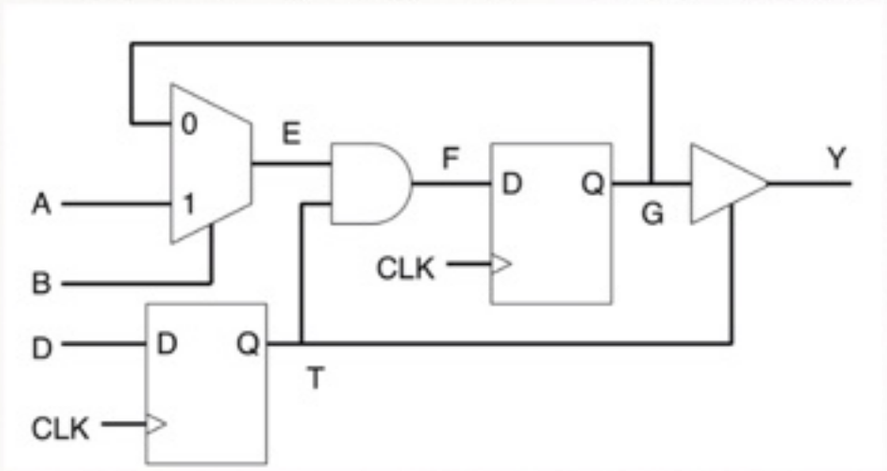
The user must supply an 8-bit value on the input bus **n**, and assert **start** (in the same clock cycle). The circuit will take several clock cycles to calculate the answer. When the circuit has computed the square root, it will supply this value on the output bus **F**, assert **done** and wait until the **start** signal is returned to 0. This is the same timing behaviour as the datapath circuits we talked about in class.

You must design your circuit using a datapath controlled by a state machine. Give your answer in terms of **one or more schematics (for the datapath) and/or one or more state diagrams (for the controlling state machine)**. Marks will be deducted for solutions that are overly large or complex.





MIQ1 write VHDL with same behaviour as circuit




```
library ieee;
use ieee.std_logic_1164.all;

entity CIRCUIT is
   portmap(A,B,D,E : in std_logic;
           Y : out std_logic
   )
end CIRCUIT;

architecture BEHAVIOURAL of CIRCUIT is

begin
process(clk)

begin
   if rising_edge(clk) then

      if B='1' then
         E:= A;
      else
         E:= G;
      end if;
      G:= E and T;

      if T='1' then
         Y:= G;
      else
         Y:= 'X';
      end if;
      T:= D;

if B='1' then
   F:= T and A;
else
   F:= T and G;
end if;
G<= F;

if D='1' then
   Y<= F;
else
   Y<='Z';
end if;
T<= D;
```

M1Q2

```
entity CIRCUIT2 is
      port (  A,B,C : in  std_logic;
              Y     : out std_logic  );
end CIRCUIT2;
architecture BEHAVIOURAL of CIRCUIT2 is
      signal   D,E,Q : std_logic;
      signal   V     : std_logic;
begin
      process(A)
      begin
            if rising_edge(A) then
                  V <= C and E;
                  if C = '1' then
                        D <= V;
                        Q <= B xor D;
                  else
                        Q <= V;
                  end if;
                  Y <= Q;
            end if;
      end process;
      E <= V;
```

Draw schematic for this VHDL