ECE 327 Solution to Final

2016t1 (Winter)

		Total Marks	Approx. Time	Page
Q1	DFD	20	25	2
Q2	The New, the Old, and the Midterm Leftovers	25	20	5
Q3	Latch Design	12	15	10
Q4	Latch Usage	8	15	12
Q5	Elmore	15	20	14
Q6	Power and Performance	20	25	18
Totals		100	120	

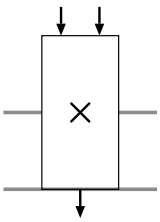
Q1 (20 Marks) DFD

(*estimated time: 25 minutes*)

Your task is to design a dataflow diagram for the expression: $a + b\times(c+d) + c\times(e+f)$.

NOTES:

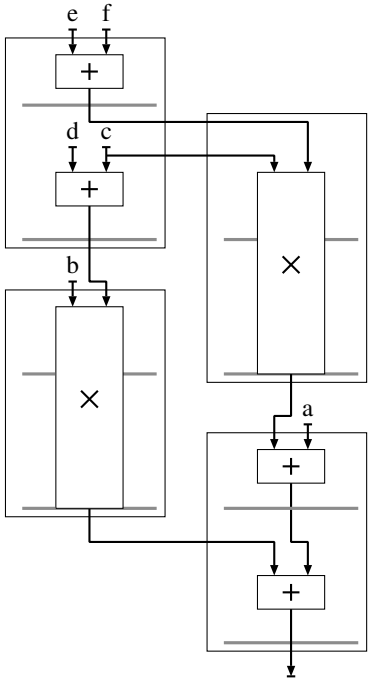
1. Outputs shall be registered
2. Optimization goals, in order of decreasing importance:
   (a) Maximize throughput
   (b) Minimize number of multipliers
   (c) Minimize number of adders
   (d) Minimize number of registers
   (e) Minimize clock period
   (f) Minimize latency
   (g) Minimize number of inputs
3. Description of the multiplier:
   - Latency=2.
   - Throughtput=0.5.
   - Combinational inputs and registered outputs. An internal register is used for the output and to store the intermediate value between the two clock cycles. This internal register does not count toward the registers used in the design.
   - You shall draw a multiplier as shown below.



4. You may schedule the input values to arrive in any order, but you may read each input value only once.
5. You do *not* need to do any allocation.
6. The *only* algebraic optimizations you may use are commutativity and associativity.

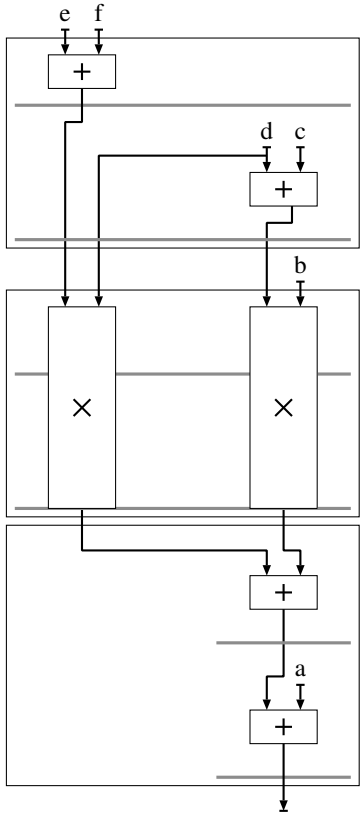
Answer:

Optimal design *20 marks max*



Throughput:	*1/2*
Latency:	*5*
Clock period:	*flop + max(add, 1/2mul)*
Number of muls:	*2*
Number of adds:	*2*
Number of regs:	*2*
Number of inputs:	*4*

Suboptimal design *15 marks max*



Throughput:	*1/2*
Latency:	*6*
Clock period:	*flop + max(add, 1/2mul)*
Number of muls:	*2*
Number of adds:	*2*
Number of regs:	*5*
Number of inputs:	*4*

Marking:

+12 marks *Dataflow diagram drawing*

+8 marks *Analysis*

Q2 (25 Marks) The New, the Old, and the Midterm Leftovers

(*estimated time: 20 minutes*)

Of the four framgments of VHDL code below, one is synthesizable and good, the other three are either illegal, unsynthesizable, or synthesizable but bad coding practices.

For each of the code fragments Q2a–Q2d:

1. Answer whether the code is *legal*
2. If the code is *illegal*: explain why, and proceed to the next code fragment.
3. Answer whether the code is *synthesizable*.
4. If the code is *unsynthesizable*: explain why, and proceed to the next code fragment.
5. Answer whether the code adheres to good coding practices, according to the guidelines for ECE 327.
6. If the code does *not follow good coding practices*: explain why.
7. For the *one* fragment that is synthesizable and good practice, in Q2e, you will calculate the minimum number of FPGA cells needed to implement the circuit.

NOTES:

1. The signal declarations are:

```
clk           : std_logic;
st            : std_logic_vector( 2 downto 0 ); -- one hot state
a, b, c, d, y : unsigned( 15 downto 0 );
z             : unsigned(  7 downto 0 );
```

2. When calculating the number of FPGA cells for the good synthesizable fragment:
   - Optimizations are allowed, so long as the externally visible input-to-output behaviour of the system does not change.
   - For full marks, you must justify your answer with a drawing and/or text.

Q2a

```
y <=   a + b + c when st(0) = '1'
  else a + c + d when st(1) = '1'
  else b + c + d when st(2) = '1'
  else (others => '0');

process begin
  wait until rising_edge(clk);
  z <= y( 7 downto 0 );
end process;
```

Answer:

Legal, synth, good.

Q2b

```
y <=   a + b + c when st(0) = '1'
  else a + c + d when st(1) = '1'
  else b + c + d when st(2) = '1';

process begin
  wait until rising_edge(clk);
  z <= y( 7 downto 0 );
end process;
```

Answer:

Legal, synth, bad: latch because "when" conditions do not cover all possiblities.

Q2c

```
process begin
  wait until rising_edge(clk);
  y <=   a + b + c when st(0) = '1'
    else a + c + d when st(1) = '1'
    else b + c + d when st(2) = '1';
end process;

z <= y( 7 downto 0 );
```

Answer:

Illegal: conditional assignment is a concurrent statement. Only sequential statements are allowed inside a process.

Q2d

```
process begin
  wait until rising_edge(clk);
  case st is
    when "001" => y <= a + b + c;
    when "010" => y <= a + c + d;
    when "100" => y <= b + c + d;
  end case;
end process;

z <= y( 7 downto 0 );
```

Answer:

Illegal: cases do not cover all possibilities.

Marking:

+3.5 marks *each subquestion*

+0.5 marks *each box*

2 marks *explanation*

+1 mark *answered all subquestions*

Q2e (10 Marks) Area analysis for synthesizable and good code fragment:

Answer:

Original code from Q2a:

```
y <=   a + b + c when st(0) = '1'
  else a + c + d when st(1) = '1'
  else b + c + d when st(2) = '1'
  else (others => '0');

process begin
  wait until rising_edge(clk);
  z <= y( 7 downto 0 );
end process;
```

Area analysis:

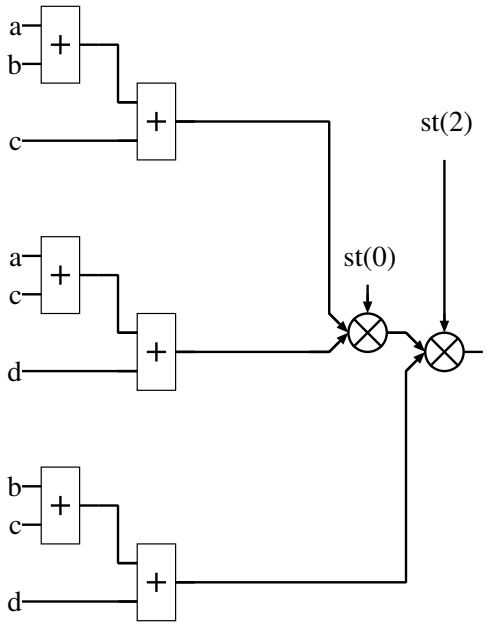
- *Optimization: do all arithmetic on 8 bits, because output only 8 bits in the end.*
- *Optimization: delete last `else` clause and test for `st(2)`, because with one-hot encoding the first three `when` clauses cover all possibilities,*

```
y <=   a + b + c when st(0) = '1'
  else a + c + d when st(1) = '1'
  else b + c + d;  -- st(2) = '1' as a comment
```

- *Optimization: recognize that we need only two adders and some multiplexers. Transform expressions to create adders with a 2:1 mux on one input, because that pair of operations can fit into 1 LUT.*

  There are several techniques to do the transformation.

  First, we could use "mux-pushing" to move multiplexers through adders. We begin with the picture below.



A second technique is textual. We begin by arranging operands into columns with common terms:

```
y <=   a + c + b when st(0) = '1'
  else a + c + d when st(1) = '1'
  else b + c + d;  -- st(2) = '1' as a comment
```

We see that all three cases use `c` *as their middle operand.*

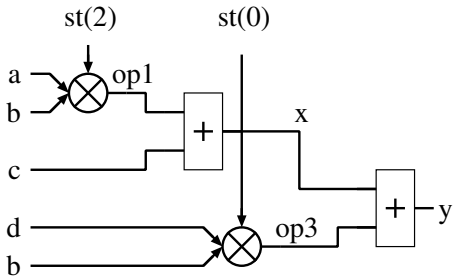
For the first operand, `st(2)` *uses* `b` *and the other cases use* `a`*.*

For the third operand, `st(0)` *uses* `b` *and the other cases use* `d`*.*

Write multiplexers explicitly:

```
op1 <= b when st(2) = '1' else a;
op3 <= b when st(0) = '1' else d;
x <= op1 + c;
y <= w + op3;
```

In both the mux-pushing and textual case, we end up with following picture:



- `op1` *and* `x` *will fit into one LUT per bit.*
- `op3` *and* `y` *will fit into one LUT per bit.*
- `z` *will use one flip-flop per bit.*
- *We have more LUTs than flip-flops, so LUTs determine the total number of FPGA cells.*

total of 16 cells

Marking:

+2.5 marks *reduce all operations to 8 bits*

+2.5 marks *remove else clause*

+2.5 marks *optimize to 2 muxes and 2 adders*

+2.5 marks *combine mux and adder into 1 LUT*

-2 marks *1 LUT/flop per bit*

-2 marks *cells = max(LUTs, flops)*

-2 marks *functionally incorrect optimization*

Q3 (12 Marks) Latch Design

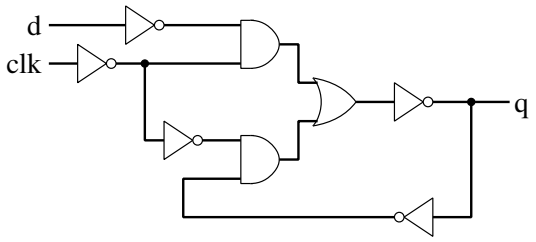
(*estimated time: 15 minutes*)

For each of the circuits below, answer whether it is a correct latch. If it is a correct latch, analyze the timing parameters. If it is not a correct latch, explain how the behaviour is incorrect or how to fix the design.

NOTES:

1. All gates have a delay of 1.
2. There are extra copies of the circuits on the next page.

1



Answer:

Incorrect: delay(clk→load-enable→join) < delay(clk→store-enable→join)

Marking:

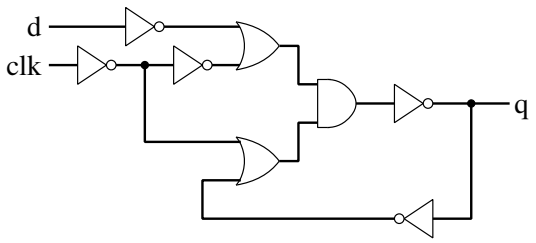
4 marks *Incorrect*

- **4 marks** *correct justification*
- **2 marks** *mostly correct justification*

2 marks *Correct*

- **+0.5 marks** *active hi/lo = lo*
- **+0.5 marks** *clock-to-q = 4*
- **+0.5 marks** *setup = 3*
- **+0.5 marks** *hold = 0*

2



Answer:

Correct, active-low

Clock-to-Q	*5*
Setup	*4*
Hold	*1*

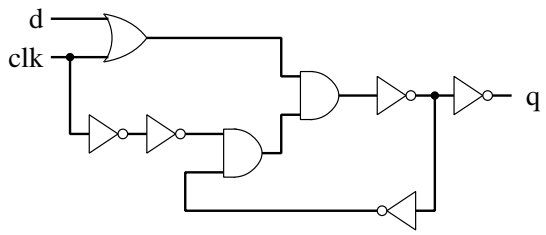
Marking:

4 marks *Correct*

- **1 mark** *each for timing parameters*

2 marks *Incorrect with significant correct information*

3



Answer:

Incorrect: the join gate is not a correct multiplexer

Marking:

4 marks *Incorrect*

- **4 marks** *correct justification*
- **2 marks** *mostly correct justification*

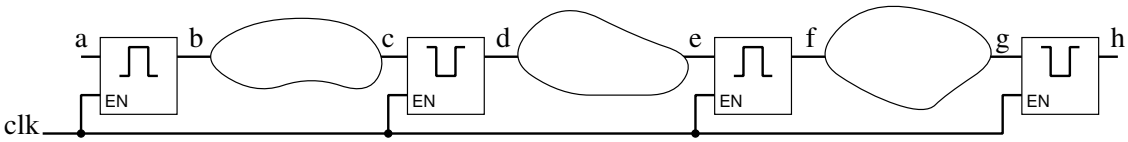
2 marks *Correct*

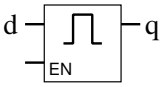
- **+0.5 marks** *active hi/lo = lo*
- **+0.5 marks** *clock-to-q = 4*
- **+0.5 marks** *setup = 2*
- **+0.5 marks** *hold = 0*

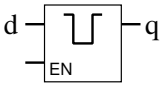
Q4 (8 Marks) Latch Usage

(*estimated time: 15 minutes*)

Some high-speed pipelines in ASICs use latches rather than flip-flops. Latches must be used in an alternating pattern of active-high and active-low, as shown below.



Active-high latch 

Active-low latch 

Q4a (5 Marks) Behaviour

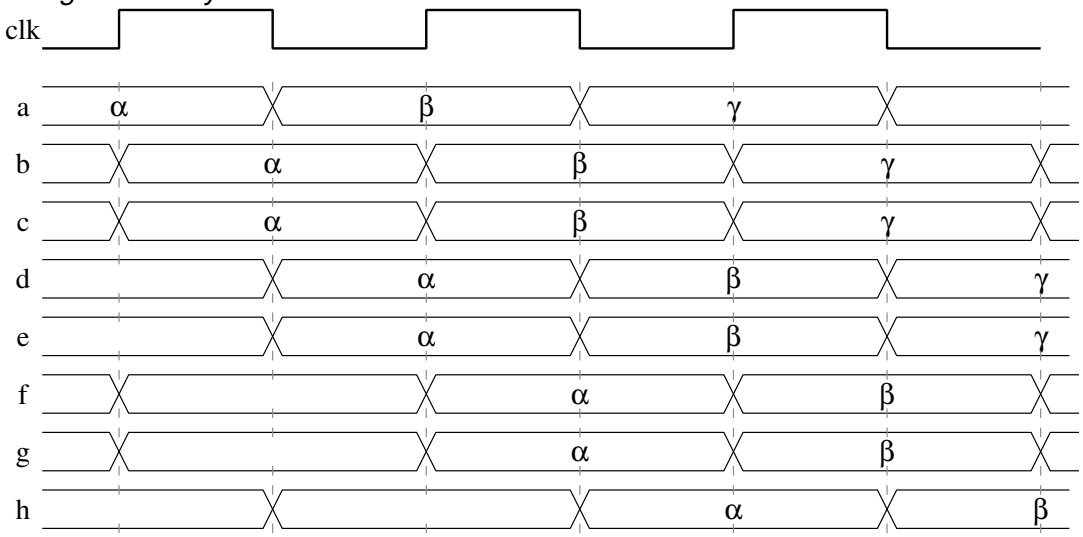
Draw the execution-trace/waveform for the signals `b`, `c`, and `d`.

NOTES:

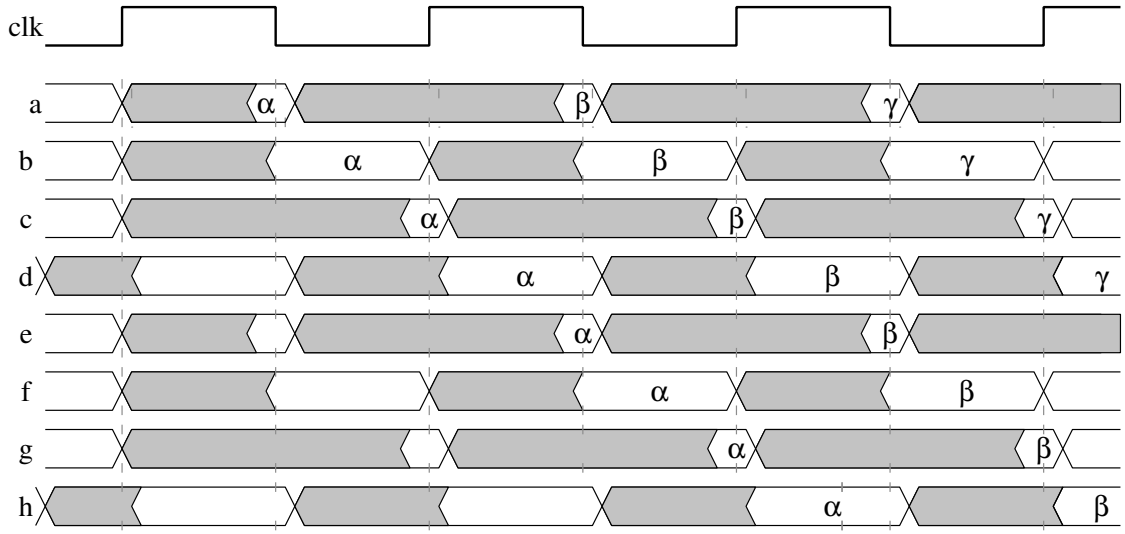
1. Use zero-delay simulation semantics.

Answer:

1. *Using zero-delay simulation:*



2. *Not part of the answer, but provided as additional information, the behaviour showing the minimum windows of stable values on each signal:*



Marking:

+1 mark *zero delay*

+2 marks *latches hold value in store mode*

+2 marks *polarity of latches is correct*

Q4b (3 Marks) Timing Parameters

Answer which *one* of the timing parameters below causes much more difficulty in latch-based pipelines than in flop-based pipelines. **For full marks, you must justify your answer.**

Answer:

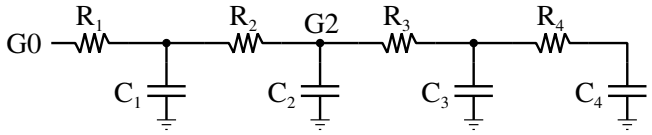
Clock skew *In a flop-based pipeline, clock skew is handled simply by extending the clock period. With a latch-based pipeline, clock skew can cause data to slip through two latches at the same time.*

Q5 (15 Marks) Elmore

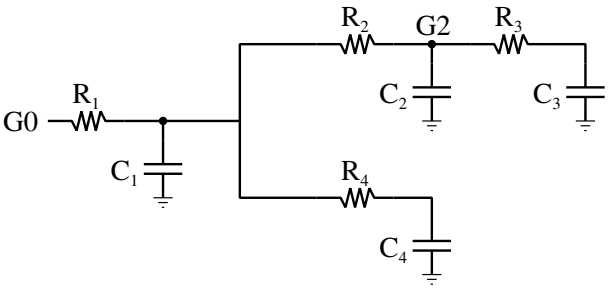
(*estimated time: 20 minutes*)

In this question, you will analyze the circuits below (A, B, C, and D) with respect to the Elmore delay from G0 to G2.

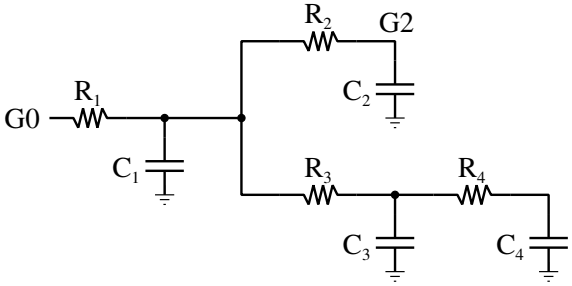
A



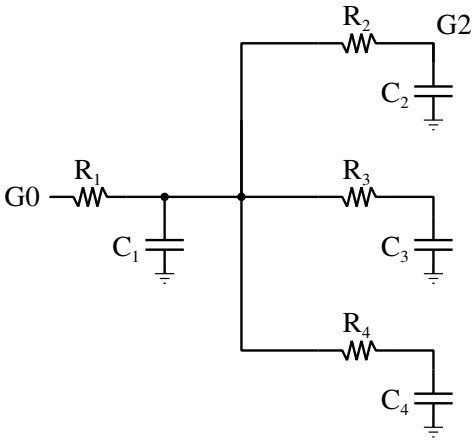
B



C



D



Q5a (8 Marks) Ranking Circuits

Rank the circuits (A, B, C, and D) in terms of the delay from G0 to G2, from smallest delay (fastest) to largest delay (slowest).

NOTES:

1. If multiple circuits have the same delay, write the identifiers for the circuits (A, B, C, or D) on the same line in the ranking.
2. Each resister Ri and capacitor C_i has the same value in each circuit.

   For example, the R_1 resistors have the same value in each circuit, and the R_2 resistors have the same value in each circuit; but the value of R_1 might be different from the value of R_2.

Answer:

This justification is provided for explanation, it is not part of the expected answer.

The delay from G0 to G2 is proportional to the voltage drop from G0 to G2. The voltage drop is the sum of the voltage drops across the resistors on the path from G0 to G2. The voltage drop across a resistor is proportional to the current flowing through the resistor, which is proportional to the sum of the downstream capactitors.

To minimize delay, we want to minimize resistance along the path from G0 to G2 and minimize the capacitance that is downstream from each resistor on the path.

All of the circuits have the same resistors on the path from G0 to G2: R1 and R_2*.*

All of the circuits have the same capactitors downstream from R_1*:* C_1*,* C_2*,* C_3*, and* C_4*.*

The difference between the circuits comes down to the capacitors that are downstream from R_2*.*

	Resistors
A	C_2, C_3, C_4
B	C_2, C_3
C	C_2
D	C_2

Circuits C, and D are the fastest and are equally fast, because they have only C_2 *downstream from* R_2*, and all other circuits have* C_2 *and additional capacitors downstream from* R_2*.*

By similar reasoning, B is faster than A.

	Circuit
Fastest 1	*C, D*
2	*B*
3	*A*
Slowest 4	

Marking:

+2 marks *each correct ordering for a circuit*

Q5b (7 Marks) Resistance and Capacitance

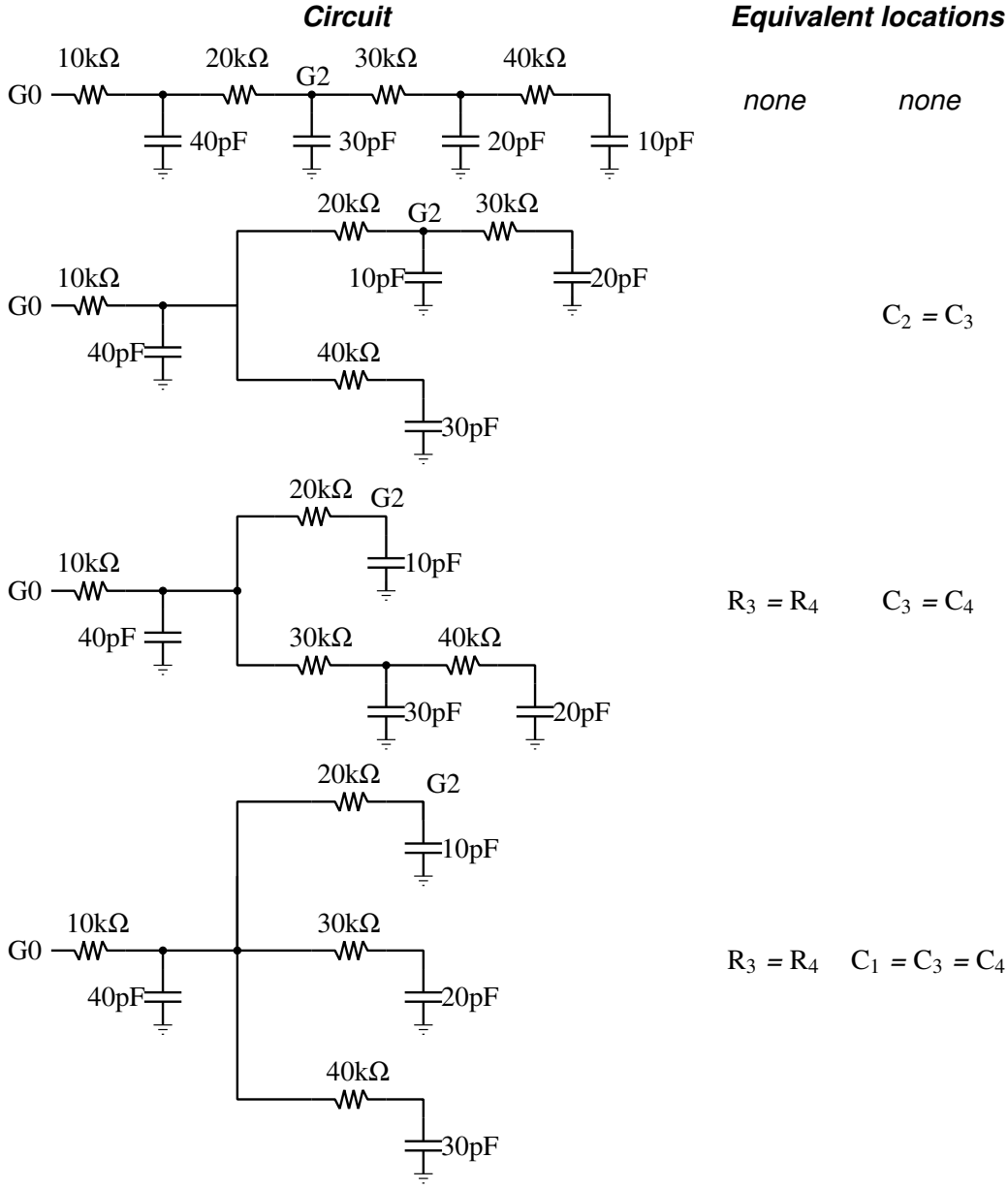
You are given one each of a 10 kΩ, 20 kΩ, 30 kΩ, and 40 kΩ resistor and one each of a 10 pF, 20 pF, 30 pF, and 40 pF capacitor. Your task is to choose the location ($R_1 \ldots R_4$) for each resistor and ($C_1 \ldots C_4$) for each capacitor so that you minimize the delay from G0 to G2 for your fastest circuit.

On the copy of your *fastest* circuit below, label each resistor and capacitor with the value (*e.g.*, 10 kΩ) that would minimize the delay.

NOTES:

1. Annotate ***only*** your fastest circuit. Leave the other three circuits blank. If multiple circuits are equally fast, arbitrarily choose one of the fastest to annotate.
2. Use each value (*e.g.*, 10 kΩ) exactly once.
3. If multiple locations of a resistor or capacitor will have an equal effect on the delay, list those equivalent locations below.

Answer:



Marking:

+1.5 marks *each resistor and capacitor that matters in the delay*

+1 mark *equivalent resistor locations*

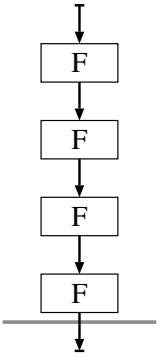
+1.5 marks *equivalent capacitor locations*

Q6 (20 Marks) Power and Performance

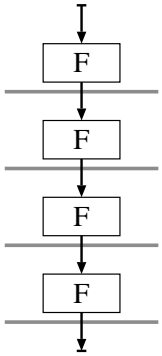
(*estimated time: 25 minutes*)

You have been promoted to be the technical leader of the F4 group for the next generation Waterluvian filter. The F4 module is the most important module in the Waterluvian filter and its design is guarded with the highest levels of security.

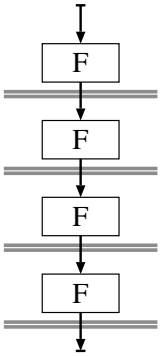
Each of your engineers, Aarti, Bob, and Clio, have drawn their dataflow diagram. It is your task to evaluate their dataflow diagrams in terms of MPPS/Watt, where MPPS=mega-pixels per second. Using Aarti's design as a baseline, estimate the relative MPPS/Watt of Bob and Clio's designs (*e.g.*, "Bob's MPPS/Watt will be $e^{2\pi}$ times the MPSS/Watt of Aarti's"). **For full marks, you must justify your answer.**



Aarti's original (A)



Bob's option (B)



Clio's option (C)

NOTES:

1. All designs will be run with the same supply voltage.
2. Each design will be run at its maximum clock speed.

Answer:

1. *MPPS/Watt is a measure of energy per pixel. We are optimizing for the total energy consumed to process a pixel.*
2. *Assumption: delay of a register is neglible compared to that of F.*
3. *Assumption: total area of registers is neglible compared to that of F.*
4. *Assumption: leakage power is negligible.*
5. *Assumption: short circuiting power is negigible, or lump switching and short circuiting power together and approximate them by area.*
6. *Equations:*

$$MPPs \propto Tput \times ClkSpeed$$

$$Power \propto Area \times ClkSpeed$$

$$MPPs/Watt \propto Tput/Area$$

7. *Analysis:*

	Aarti	***Bob***	***Clio***
Tput	*1*	*1/4*	*1*
Area	*4F*	*1F*	*4F*
MPPs/Watt	*1/(4F)*	*1/(4F)*	*1/(4F)*
Clock period	*4F*	*1F*	*1F*
MPPS	*1/(4F)*	*1/(4F)*	*1/F*
Power	*1*	*1*	*4*

8. *Final answer: All options have the same MPPs/Watt.*

If assume that area of register is same as F and that delay of register is negligible:

	Aarti	***Bob***	***Clio***
Tput	*1*	*1/4*	*1*
Area	*5F*	*2F*	*8F*
MPPs/Watt	*1/(5F)*	*1/(8F)*	*1/(8F)*
Relative MPPs/Watt	*1*	*5/8*	*5/8*
Clock period	*4F*	*1F*	*1F*
MPPs	*1/4*	*1/4*	*1*
Power	*5/4*	*1/2*	*8*

Marking:

+3 marks *Assumptions*
+6 marks *Analysis of DFD (latency, throughput, clockspeed, area)*
+6 marks *Energy and performance*
+6 marks *Conclusion and final answer*
+3 marks *Clarity*

(*MDA: describe options as leaving data-dependency graph unchanged, but changing the clock-cycle boundaries in DFD (compare to 2014t1-final) *)

(*MDA: optimize for MIPS/Watt: Howie Hifreq and Laura Lovolt *)

(*MDA: additional question would be how could increase crit path from Flop+Add to Flop+2Add without increasing area: answer: if fully pipelined, then just delete registers. If not fully pipelined, then the design has reuse, so going to 2Add will eliminate some potential reuse, which will either decrease throughput or increase area. *)