ECE-327 Solution to Midterm

2017t1 (Winter)

All requests for re-marks must be submitted in writing to Mark Aagaard before noon on Thursday June 30.

A random collection of midterms were scanned. Exams that are submitted for re-marking will be verified against this set.

		Total Marks	**Approx. Time**	**Page**
Q1	VHDL Simulation	25	15	2
Q2	The Flu, the Fever, and the Nausea	25	15	4
Q3	DFD	25	25	6
Q4	FSM	25	15	10
Totals		100	70	

Q1 (25 Marks) VHDL Simulation

(*estimated time: 15 minutes*)

For the VHDL program below, calculate the values for the signals c0, c1, c2, and at 45ns.

NOTES:

1. All of the processes are in the *same architecture*.
2. The signals a0, a1, a2, b0, b1, b2, c0, c1, and c2, are declared to be unsigned(15 downto 0).
3. Don't panic. Although the code is long, it is systematic:
   - The a signals drive the b signals
   - The b signals drive the c signals

Category of each signal			
timed	**comb**	**clk1**	**clk2**
a0		a1	a2
	b0, b1, b2		
	c0	c1	c2

4. **For full marks you must justify your answer, using text and/or the waveform diagram. You may, but are *not* required to, show a delta-cycle simulation.**

```
process begin
  clk1 <= '0';
  wait for 10 ns;
  clk1 <= '1';
  wait for 10 ns;
end process;

clk2 <= clk1;

process begin
  reset <= '1';
  wait for 11 ns;
  reset <= '0';
  wait;
end process;

process begin
  a0 <= ( others => '0' );
  wait for 10 ns;
  a0 <= a0 +    1;
  a0 <= a0 +    2;
  wait for 10 ns;
  a0 <= a0 +    4;
  a0 <= a0 +    8;
  wait for 10 ns;
  a0 <= a0 +   16;
  a0 <= a0 +   32;
  wait for 10 ns;
  a0 <= a0 +   64;
  a0 <= a0 +  128;
  wait for 10 ns;
end process;
```

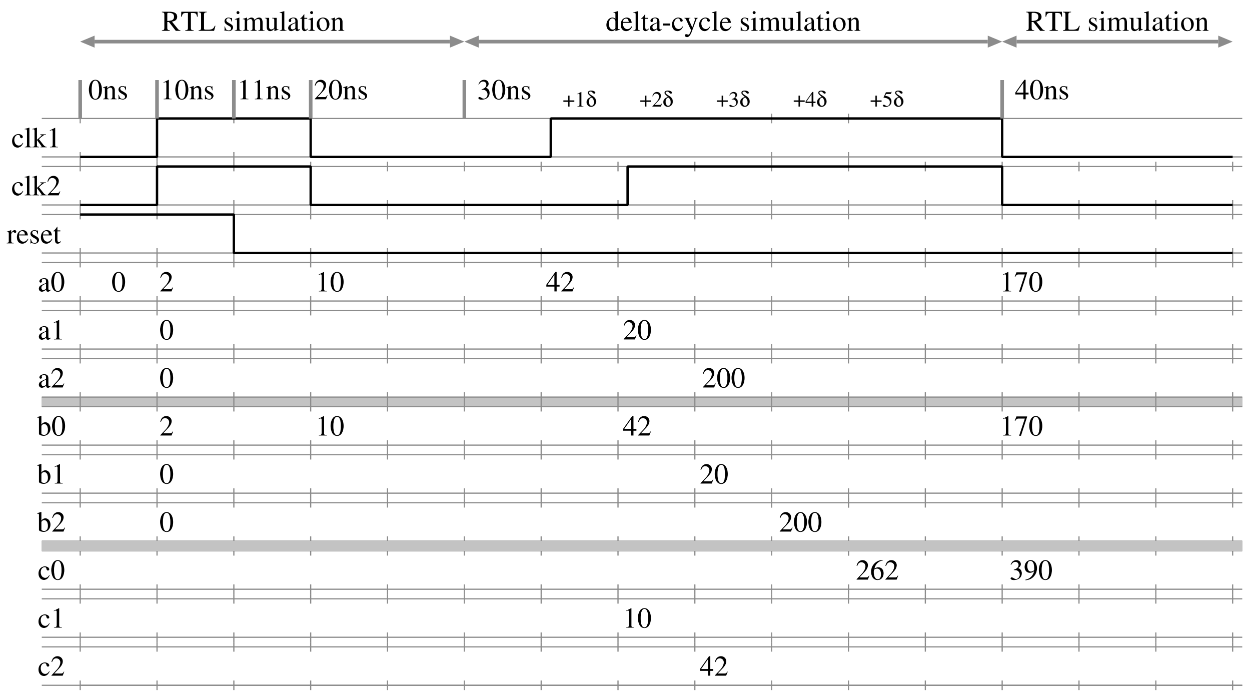
```
process begin
  wait until rising_edge( clk1 );
  if reset = '1' then
    a1 <= ( others => '0' );
  else
    a1 <= a1 +  20;
  end if;
end process;

process begin
  wait until rising_edge( clk2 );
  if reset = '1' then
    a2 <= ( others => '0' );
  else
    a2 <= a2 + 200;
  end if;
end process;

b0 <= a0;
b1 <= a1;
b2 <= a2;

c0 <= b0 + b1 + b2;

process begin
  wait until rising_edge( clk1 );
  c1 <= b0 + b1 + b2;
end process;

process begin
  wait until rising_edge( clk2 );
  c2 <= b0 + b1 + b2;
end process;
```

Answer on the next page

Answer:

The rising clock edges occur at 10 ns and 30 ns. The only effect of the edges at 10 ns is to reset a1 and a2 to 0. The values of the c signals at 10 ns are irrelevant, because the c signal are updated based only on the b signals. So, the values of the c signals at 10 ns will be overwritten at 30 ns (c1 and c2) and at 45 ns (c0) based on the values of the b signals.

The critical clock edges occur at 30 ns.



	value at 45ns
c0	390
c1	10
c2	42

Some example incorrect answers:

c0	*c1*	*c2*	*mistakes*
475	*15*	*63*	*incorrect update to a0*
475	*15*	*15*	*incorrect update to a0; clk1==clk2*
283	*15*	*63*	*incorrect update to a0; missed a0 change at 40ns*

20 = 0x14 = 1 0100

200 = 0xC8 = 1100 1000

Marking:

- **+3 marks** *clk1 is correct*
- **+3 marks** *clk2 is correct*
- **+3 marks** *a0 gets last asn in each sim cycle*
- **+3 marks** *a0, b0, c0 updated at 40 ns*
- **+3 marks** *a1, a2 correct*
- **+3 marks** *b1, c1 updated on clk1*
- **+3 marks** *b2, c2 updated on clk2*
- **+2 marks** *arithmetic correct*
- **+2 marks** *neatness and clarity*

Q2 (25 Marks) The Flu, the Fever, and the Nausea

(*estimated time: 15 minutes*)

For each of the code fragments Q2a–Q2e:
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, and proceed to the next code fragment
7. Calculate the number of flip-flops that will be synthesized from the code.

NOTES:

1. The signal declarations are:

```
clk, a        : std_logic;
b, c, d, e, f : unsigned( 7 downto 0 );
```

Q2a

```
d <=   b + c when d(0) = '1'
  else b - c;
```

Answer:
Legal, synth, bad (comb loop through d(0)). If said "Legal, synth, good", then part marks for 0 flops.

Q2b

```
process begin
  c <= b;
  wait until rising_edge( clk );
end process;
```

Answer:
Legal, unsynth (statement before wait). If said "Legal, synth, good", then part marks for 8 flops.

Q2c

```
process begin
  wait until rising_edge( clk );
  d <= b + 2;
  f <= e + 4;
end process;

e <= d;
```

Answer:
Legal, synth, good. 16 flops.

Q2d

```
process begin
  wait until rising_edge( clk );
  if a = '1' then
    c <= b + 2;
    d <= b + c;
  else
    d <= b - c;
  end if;
end process;
```

Answer:
Legal, synth, good. 16 flops

Q2e

```
process ( b, c ) begin
  d <= b + 2;
  d <= c + 5;
end process;
```

Answer:
Legal, synth, good. 0 flops

Marking:

- **3 marks** *Legal, synth, good.*
- **2 marks** *Number of flops or justification*

Q3 (25 Marks) DFD

(*estimated time: 25 minutes*)

In this question, you will design and analyze a dataflow diagram for the equation:

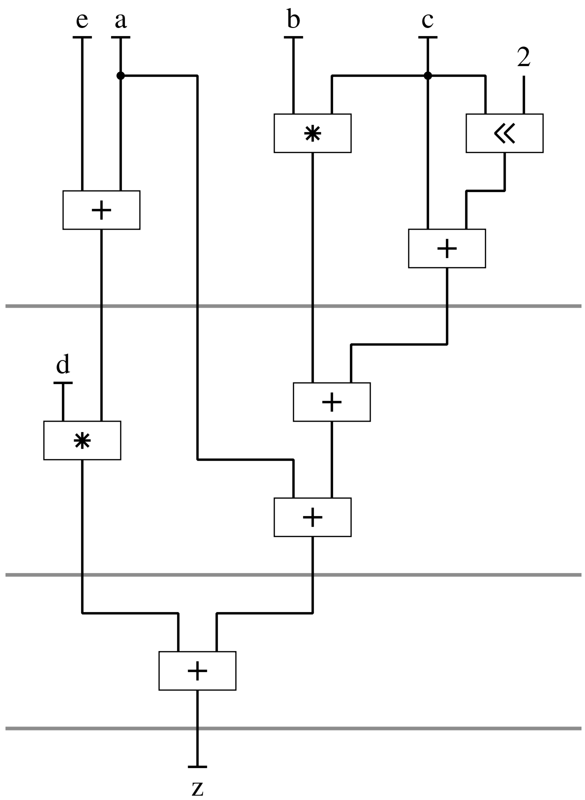
```
z = a + a*d + 5*c + b*c + d*e
```

NOTES:

1. Inputs shall be combinational and outputs shall be registered.
2. The delay of a multiplier is approximately twice that of an adder.
3. Optimization goals in order of decreasing importance:
   (a) minimize number of *multipliers*
   (b) minimize *clock period*
   (c) minimize *latency*
   (d) minimize number of *adders*
   (e) minimize number of *registers*
   (f) minimize number of *input ports*
   (g) minimize number of *output ports*
4. Input values may be read in any clock cycle, but each input value shall be read exactly once.
5. Algebraic optimizations are allowed, as long as the final value of z is correct.
6. You do *not* need to perform allocation.

Answer:

```
z = d*(a + e) + c + (c sll 2) + b*c
```

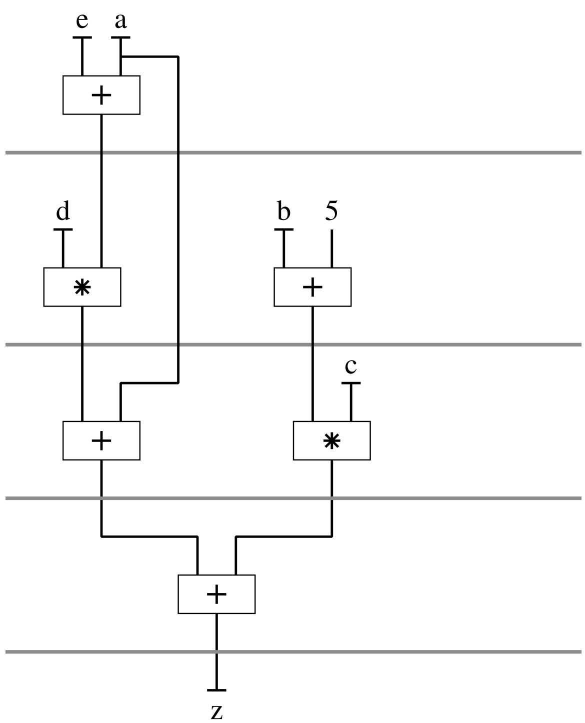


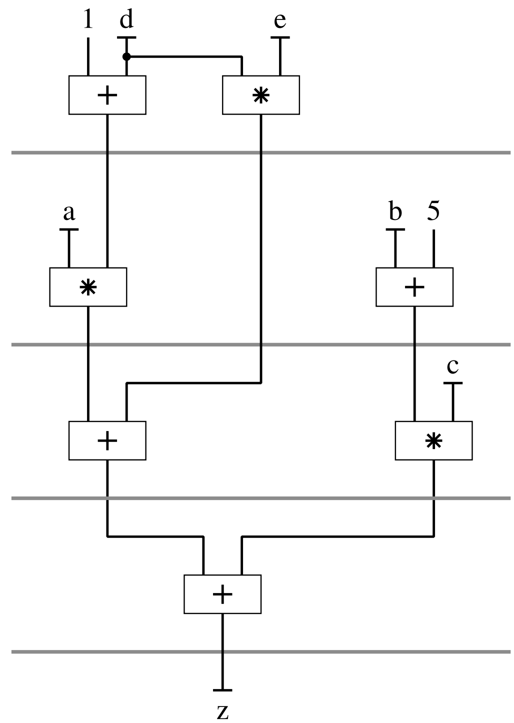
Latency	*3*
Clock period	*max(2add, mul) + flop*
Number of multipliers	*1*
Number of adders	*2*
Number of registers	*4*
Number of input ports	*4*
Number of output ports	*1*

18/18 marks max

`z = a + d*(a+e) + c*(5+b)`

`z = a*(d + 1) + c*(b+5) + d*e`





16/18 marks max

Latency	4
Clock period	mul + flop
Number of multipliers	1
Number of adders	1
Number of registers	3
Number of input ports	2
Number of output ports	1

DFD: 18 marks max

Marking:

- **-4 marks** *2 multipliers*
- **-3 marks** *clock period = mul + add + flop*
- **-2 marks** *latency = 5*
- **-1 mark** *each extra adder, register, or input*
- **-1 mark** *inputs not comb*
- **-1 mark** *outputs not reg*
- **-1 mark** *datapath components not comb*
- **-1 mark** *constant uses an input or reg*
- **-1 mark** *read an input multiple times*
- **-1 – -3 marks** *DFD is difficult to understand*

Analysis: 7 marks max

Marking:

- **1 mistake** *answer is off by 1*
- **2 mistakes** *answer is off by 2 or more*
- **3 mistakes** *missing answer*
- **7 marks** *0 mistakes*
- **6 marks** *1 mistake*
- **5 marks** *2 mistakes*
- **4 marks** *3–4 mistakes*
- **3 marks** *5–6 mistakes*
- **2 marks** *7–8 mistakes*
- **1 mark** *9 mistakes*

Q4 (25 Marks) FSM

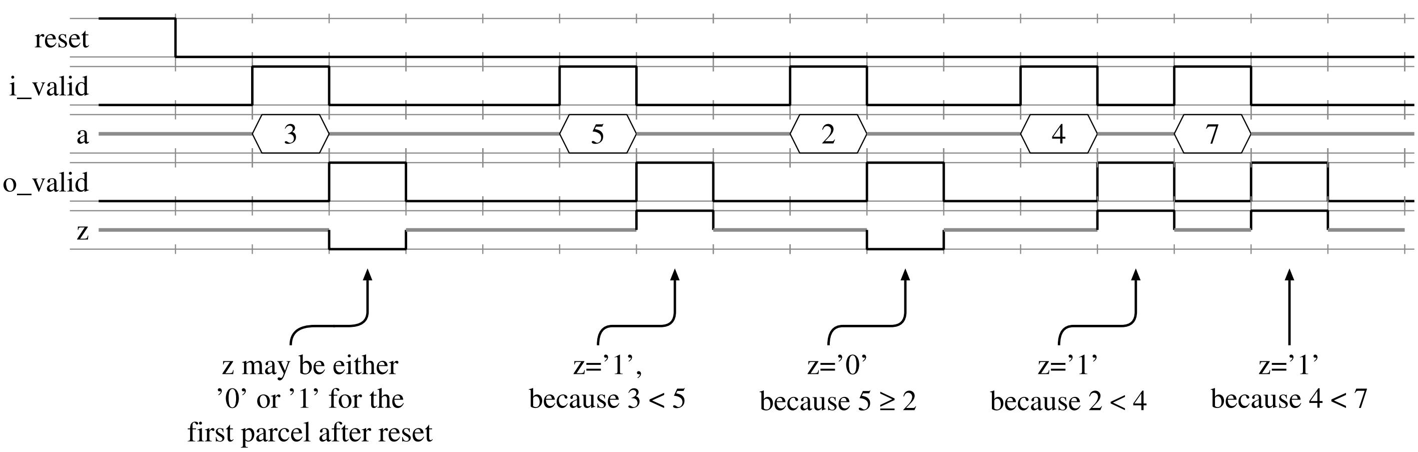
(*estimated time: 15 minutes*)

You've just been promoted to a manager and have been assigned the exciting project of designing a new state machine. Because you are manager, you delegate the design work to the three employees you supervise. Your task is to evaluate each design to determine whether it is correct.

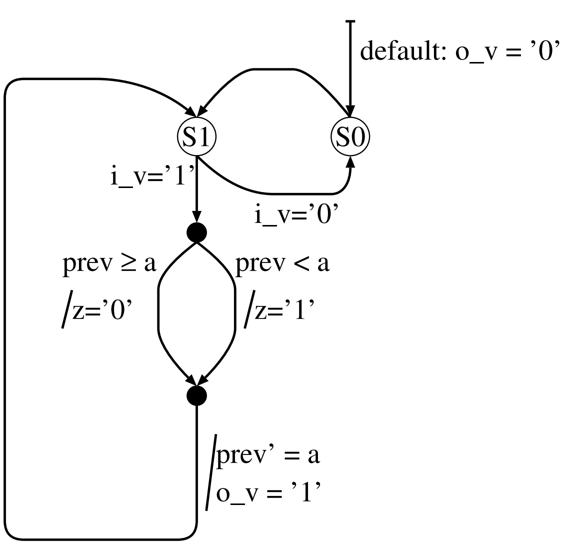
The state machine has one data input (`a`) and one data output (`z`). For each parcel, the value of `z` shall be `'1'` if the current parcel's `a` is greater than the previous parcel's `a`.

NOTES:

1. The system shall use a parcel schedule of unpredictable number of bubbles.
2. There is no requirement for the value of `z` for the first parcel after reset is deasserted. In other words, for the first parcel after reset, the value of `z` may be either `'0'` or `'1'`.
3. There is no requirement for the latency between `i_valid='1'` and `o_valid='1'`. In other words, the latency may be any number of clock cycles.
4. If the state machine is correct, answer what the latency is and what is the minimum number of bubbles between parcels.
5. If the state machine is incorrect, explain either how the machines behaviour differs from the system description or how the state machine could be modified to fix the incorrect behaviour.
6. In the diagrams below, "`i_valid`" has been shortened to "`i_v`" and "`o_valid`" has been shortened to "`o_v`".



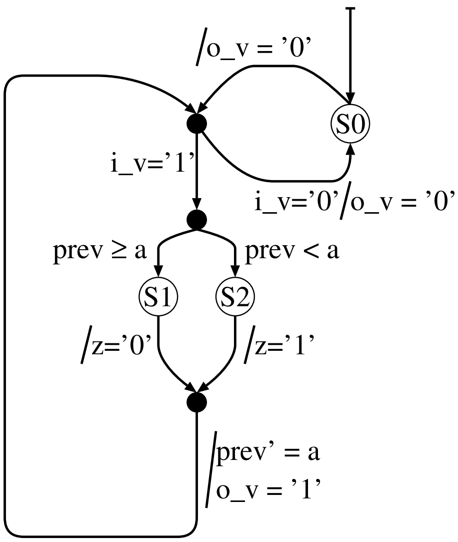
Q4a



Answer:

Incorrect. Will miss `i_valid='1'` on transition between S0 and S1. If answered "correct": latency=0, bubbles=0.

Q4b



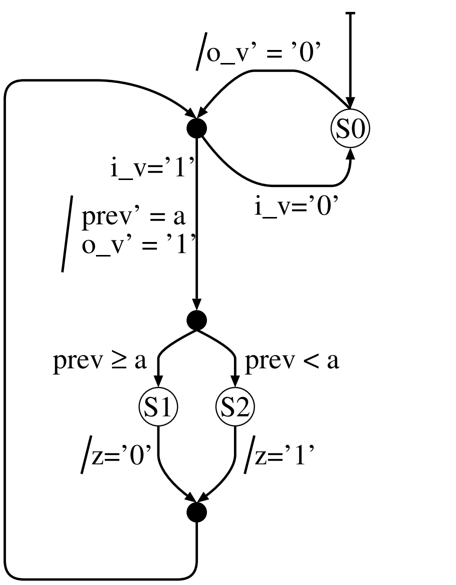
Answer:

Incorrect.

- `o_valid='1'` *is overwritten by the assignment* `o_valid='0'` *on the edge to S0 if a new parcel arrives in the same cycle as the current parcel exits.*
- `/prev'=a` *is a registered assignment. If there are no bubbles, then the* `prev<gea` *comparisons read* `prev` *in the same clock cycle as the registered is performed, and so get the old value of* `prev`*.*
- `a` *is read in two different clock cycles, once in the comparison with* `prev` *and once in the next clock cycle when* `prev` *is updated.*

If answered "correct": latency=1, bubbles=0.

Q4c



Answer:

Incorrect. If there are one or more bubbles between parcels, then `o_valid='1'` *for 2 consecutive clock cycles. If answered "correct": Latency = 1. Bubbles = 0.*

Marking:

- **+1 mark** *gave complete answers for all fsms*
- **+8 marks** *each fsm*
  - **+2 marks** *correct/incorrect*
  - **+6 marks** *justification, if answered "incorrect"*
  - **+3 marks** *latency, if answered "correct"*
  - **+3 marks** *bubbles, if answered "correct"*