ECE 327 Solution to Midterm

2014t2 (Spring)

All requests for re-marks must be submitted in writing to Mark Aagaard before 8:30am on Friday March 7.

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

Q1 (10 Marks) VHDL Semantics

(*estimated time: 7 minutes*)

Is it possible for a simulation round not to contain any delta cycles? **Justify your answer in terms of VHDL simulation semantics.**

Answer:

Yes, if the simulation round contains exactly one simulation cycle. In the first simulation cycle of a simulation round, timed processes are executed. If none of the timed processes that run change the value of any signals, then there will not be any more simulation cycles in the simulation round.

Q2 (20 Marks) The Yellow, the Red, and the Goaaalllll!!!!

(*estimated time: 10 minutes*)

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. If the code does *follow good practices*: draw the circuit that would most likely result from synthesizing the code.

NOTES:

1. If the VHDL code includes an implicit state machine: draw the gates, wires, and flops for the datapath. All of the arithmetic and logical operators in the VHDL code (*e.g.*, "+", "−", "<", and "`xor`") are considered part of the datapath.
2. You may draw the control portion of the circuit as a cloud or black-box that drives the appropriate signals in the datapath.
3. The signal declarations are:

```
clk     : std_logic;
a, b    : unsigned( 7 downto 0 );
m, n, p : std_logic_vector( 0 to 3 );
```

Q2a

```
for_i : for i in 0 to 3 generate
  if_yes : if i = 0 generate
    m(i) <= p(3);
  end generate;
  if_no : if i /= 0 generate
    m(i) <= p(i-1);
  end generate;
  p(i) <= m(i) xor n(i);
end generate;
```

Answer:
Legal, synth, bad (comb loop).

Q2b

```
process begin
  a <= (others => '0');
  wait until rising_edge(clk);
  loop
    a <= a + 1;
    wait until rising_edge(clk);
  end loop;
end process;
```

Answer:
Legal, unsynth (asn before wait).

Q2c

```
b <= a + 1;
process (clk) begin
  if rising_edge(clk) then
    a <= b;
  end if;
end process;
```

Answer:
Legal, synth, good.

Q2d

```
process (clk) begin
  if rising_edge(clk) then
    a <= b;
  end if;
  if rising_edge(clk) then
    b <= a + 1;
  end if;
end process;
```

Answer:
Legal, unsynth (2 if-rising-edge.

Q3 (15 Marks) Area Analysis

(*estimated time: 10 minutes*)

Calculate the mininum number of FPGA cells needed to implement the VHDL code below.

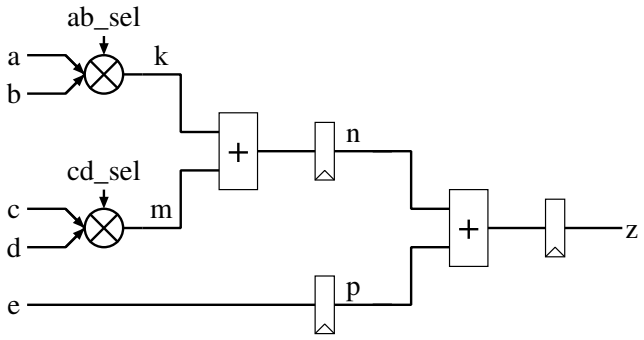
NOTES:

1. The signals `ab_sel` and `cd_sel` are `std_logic`.
2. The signals `a`, `b`, `c`, `d`, `e`, `k`, `m`, `n`, `p`, and `z` are 12-bit `unsigned`.
3. Optimizations are allowed, so long as the externally visible input-to-output behaviour of the system does not change.
4. For full marks, you must justify your answer with a drawing and/or text.

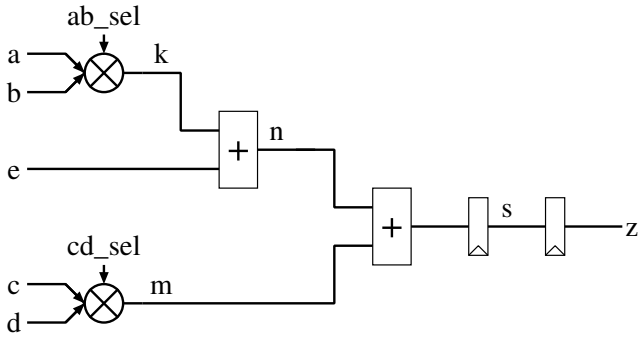
```
k <= a when ab_sel = '1' else b;
m <= c when cd_sel = '1' else d;
process (clk) begin
  if rising_edge(clk) then
    n <= k + m;
    p <= e;
    z <= n + p;
  end if;
end process;
```

Answer:

Original circuit:



Optimized circuit:



signal	*circuitry*	*LUTs per bit*	*flops per bit*	*num bits*	*LUTs total*	*flops total*
n	*mux and adder*	*1*	*0*	*12*	*12*	*0*
s	*mux and adder, reg*	*1*	*1*	*12*	*12*	*12*
z	*reg*	*0*	*1*	*12*	*0*	*12*
				Total	***24***	***24***

Total number of cells = Max(24 LUTs, 24 flops) = 24 cells.

We can put a 2:1 mux and an adder in the same LUT.

Each cell has a LUT and a flip-flop, so we can fit both the adder and the reg for `z` into one cell per bit.

Marking:

10 marks *24 cells with correct justification*

If scaled by number of bits *Sum of the following:*

- **2 marks** *Baseline*
- **1 marks** *Need a 2:1 mux, two adders, and a flop*
- **1 mark** *At most 4 inputs + carry in per LUT*
- **1 mark** *At most 1 output + carry out per LUT*
- **2 marks** *Do not include cells for* `a`, `b`, `c`, `d`, `e`
- **2 marks** *Use both LUT and flop in same cell*
- **1 mark** *Put 2:1 mux in same cell as an adder*

If did not scale by number of bits *One of the following:*

- **4 marks** *2 cells with explanation of 2 adds + mux + reg*
- **3 marks** *2 cells with justification that have 5 inputs and can do 4 inputs / cell*
- **1 mark** *1 cell*

Penalties

- **-2 marks** *missing register for* `z`
- **-2 marks** *missing optimization for mux pushing*

Q4 (15 Marks) Function Table and Encoding

(*estimated time: 15 minutes*)

This question will examine a function table and encoding for the pseudocode specifications of `y` and `z` given below.

NOTES:

1. Inputs:
   - The signal `a` is a `std_logic_vector` that is a one-hot encoding of a size; where the size is either small, medium, or large.
   - The signal `b` is a 3-bit `unsigned`
2. Outputs:
   - The signal `y` is a color, which is one of `red`, `blu`, or `grn`
   - The signal `z` is a 8-bit `unsigned`.
3. Any condition not defined by the specifications below is a don't care.

```
if b then
  y = blue;
elsif a == lg then
  y = grn;
else
  y = red;
```

```
if a == sm or a == md then {
  if b then
    z = 3;
  else
    z = 5;
}  # there intentionally is not an else clause
```

Q4a (3 Marks) One-Hot Encoding

Define the encoding for `a`.

NOTES:

1. The table shows 5 bits for `a`, if you do not need all 5 bits, draw an $\times$ through the label of any bits that you do not need (*e.g.*, ~~`a(2)`~~).

Answer:

	~~a(4)~~	~~a(3)~~	a(2)	a(1)	a(0)
small			0	0	1
medium			0	1	0
large			1	0	0

Q4b (6 Marks) Function Table

Draw one function table that defines the behaviour of both `y` and `z`.

NOTES:

1. Requirement: Each output value shall appear in exactly one cell.

Answer:

		y	b = 0	b = 1
a	small	100	red	blu
	medium	010	red	blu
	large	001	grn	blu

		z	b = 0	b = 1
a	small	100	3	5
	medium	010	3	5
	large	001	—	—

		y	z
---	0		5
---	1	blu	3
1--	0	grn	
0--	0	red	

Q4c (6 Marks) Code

Using your encoding for `a`, write if-then-else statements, in either VHDL or pseudocode, for `y` and `z`.

NOTES:

1. Optimization goal: Minimize the total cost of the conditions:
   - Each if-then-else statement has a cost of 1
   - Each AND, OR, and NOT has a cost of 1
   - Each *n*-bit equality test (=) has a cost of *n*
2. If you use VHDL, you may pretend that if-then-else conditions may be `std_logic`. That is, you *may* write `if a(0) then ...` and do *not* need to write `if a(0)='1' then ...`
3. You may choose either to combine the code for `y` and `z`, or to use separate if-then-else statements for `y` and `z`.

Answer:

```
if b then
  y = blue;
  z = 3;
else
  z = 5;
  if a(2) then
    y = grn;
  else
    y = red;
```

Total cost = 2

Q5 (20 Marks) State Machine

(*estimated time: 15 minutes*)

This question examines a state machine that implements the equation `z = a + b`.

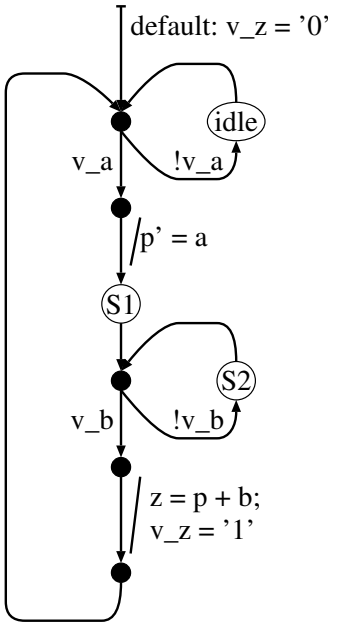
NOTES:

1. The variables `a` and `b` are both part of the same parcel.
2. There is an unpredictable number of clock cycles between when `a` arrives and when `b` arrives, but `b` arrives *at least* one clock cycle later than `a`.
3. The signal `v_a = '1'` when `a` has valid data. The signal `v_b = '1'` when `b` has valid data.
4. The state machine shall assign `v_z = '1'` for one clock cycle when `z` is valid.
5. There is an unpredictable number of bubbles between when `b` arrives and when the next value of `a` arrives.
6. Inputs and outputs may be either registered or combinational.

Q5a (15 Marks) State Machine Design

Draw a state machine that implements the specification.

Answer:



Q5b (5 Marks) Throughput

What is the maximum throughput of your state machine?

Answer:

1 parcel per clock cycle

Q6 (20 Marks) Design with Memory

(*estimated time: 15 minutes*)

This question examines the implementation of the pseudocode specification:

```
M[a+1] = b;
M[a]   = M[a+1];
M[c]   = M[c] - M[a];
z      = M[c]
```

NOTES:

1. Inputs shall be *registered*
2. Outputs may be *either combinational or registered*
3. The system shall support an *indeterminate number of bubbles*
4. Memory has registered inputs and combinational outputs (same as in class)
5. The memory may be *either dual-ported or single-ported*.
6. Optimization goals in order of decreasing importance:
   (a) minimize *latency* to `z`
   (b) minimize *clock period*
   (c) minimize *area*
      i. input ports

ii. adders and subtracters
iii. registers (*excluding* memory)
iv. output ports
v. use single-ported memory instead of dual-ported memory

7. Input values may be read in any clock cycle, but each input value shall be read exactly once.
8. Optimizations to the pseudocode are allowed, as long as the final values of `z` and `M` are correct.

Q6a (15 Marks) Dataflow Diagram

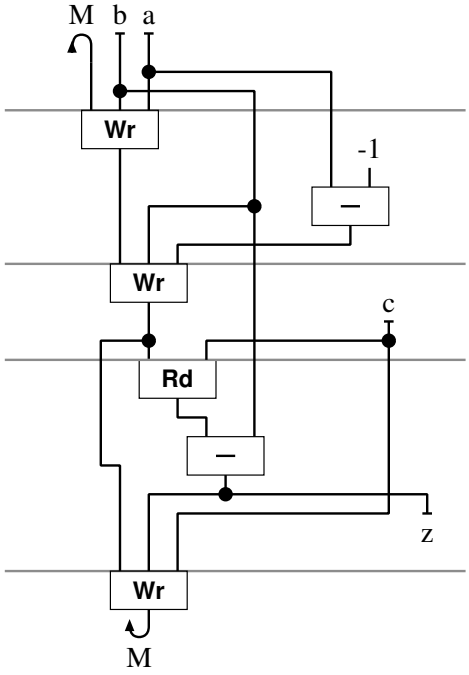
Draw a dataflow diagram for the system.

Answer:

1. *Pseudocode optimizations*

```
M[a]   = b;
M[a+1] = b;
p      = M[c] - b;
z      = p;
M[c]   = p;
```

2. *Dataflow diagram*



Marking:

+3 marks *functional correctness*
+2 marks *optimal latency*
+2 marks *DFD syntax is correct*
+2 marks *mem operations on clock cycle boundaries*
+1 mark *DFD uses M*
+1 mark *M is an inter-parcel variable*
+1 mark *use dual-port memory*
+1 mark *M has one write port and one read port*
+1 mark *Wr produces M*
+1 mark *anti-dependency arrow*
+1 mark *registered inputs*
+1 mark *combinational outputs*
+1 mark *2 registers*
+1 mark *3 inputs*
+1 mark *1 adder unit (a – 1 == a + -1)*

Q6b (5 Marks) Memory Ports

How many ports does your memory have:

Briefly justify that your choice of number of memory ports produced the most optimal design.

Answer:

Memory has 1 port (single-ported).

Dual ported memory would not reduce the latency.

Because we cannot do two write operations in the same clock cycle, the writes to `M[a]` *and* `M[a+1]` *must be done in separate clock cycles.*

Because either `a` *or* `a-1` *might equal* `c`*, the read of* `M[c]` *must be done after the writes to* `M[a]` *and* `M[a-1]` *(RAW dependencies).*

Because of the WAR dependency from `M[c]` *to* `M[c]`*, the write to* `M[c]` *must be done after the read.*

Because of the chain of dependencies, we cannot every schedule two memory operations in the same clock cycle to reduce the latency.