E&CE 327 Midterm Solution

2011t1 (Winter)

All requests for re-marks must be submitted in writing to Mark Aagaard before 9:30am on Wed March 2.

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

Q1 (24 Marks) VHDL Semantics

(estimated time: 15 minutes)

This question analyzes the VHDL semantics of the two processes below. Q1a deals with simulation steps, Q1b deals with simulation cycles, and Q1c deals with register-transfer-level simulation.

NOTES:

1. The marks in this question are dependent mostly upon the quality of the justifications, which will be marked according to *correctness, clarity, and conciseness.*
2. You do *not* need to draw delta-cycle or register-transfer-level simulation charts, but you *may* do so if it is helpful.

```
proc_a : process ( clk ) begin
  if rising_edge( clk ) then
    if ce = '1' then
      q <= d;
    end if;
  end if;
end process;
```

```
proc_b : process ( clk, ce ) begin
  if rising_edge( clk ) and ce = '1' then
    q <= d;
  end if;
end process;
```

Q1a (8 Marks) Differences in delta-cycle simulation chart

If you were to fill out two ECE-327 style delta-cycle simulation charts, one for `proc_a` and one for `proc_b`, is there a scenario where the input signals (`clk`, `ce`, and `d`) have the same behaviour, but the chart for `proc_a` would be different from the chart of `proc_b`? **For full marks, you must justify your answer.**

NOTES:

1. Ignore the difference that one chart has "`proc_a`" and one has "`proc_b`".
2. If there would be a difference, describe the behaviour on `clk`, `ce`, and `d` that would lead to the difference and describe the difference in the two charts.
3. This question includes differences at the granularity of *simulation steps*: an individual column in the delta-cycle simulation chart.

Answer:

Yes, there is a scenario where the two simulation charts will be different. If `ce` changes in a simulation cycle in which `clk` remains constant, then at the end of this simulation cycle, `proc_b` will resume and become postponed. In the next simulation cycle, `proc_b` will activate and become active. In comparison, `proc_a` will remain suspended.

Marking:

- **8 marks** *"Yes", with excellent justification*
- **6 marks** *"Yes", good justification, but no mention of postponed and suspended modes*
- **6 marks** *"Yes", count evaluation of conditions for `if` statements as simulation steps.*
- **5 marks** *"Yes", but unclear as to effect on simulation chart.*
- **4 marks** *"No", references values of signals, not process modes.*
- **3 marks** *use of VHDL semantics terminology*
- **2 marks** *some amount of correct justification*

Q1b (8 Marks) Simulation-cycle differences

Is there a scenario where the signals `clk`, `ce`, and `d` have the same behaviour for both `proc_a` and `proc_b`, but where at the end of a *simulation cycle*, the signal `q` in `proc_a` has a different value than the signal `q` in `proc_b`? **For full marks, you must justify your answer.**

NOTES:

1. If there would be a difference, describe the behaviour on `clk`, `ce`, and `d` that would lead to the difference and describe the difference in the two `q` signals.

Answer:

No, there is no scenario where the two `q` signals will have different behaviour in delta-cycle simulation.

There are three situations that could lead to different behaviours, we will discuss each in turn.

If both `proc_a` and `proc_b` run in a simulation cycle, then their two `q` signals will have the same behaviour, because the conditions in the `if` statements leading to the assignment to `q` are the same.

The only scenario in which `proc_b` runs in a simulation cycle but `proc_a` does not is if `ce` changes but `clk` remains constant (as described in the previous part of this question). In this scenario, `proc_b` will not execute the assignment to `q`, because there will not have been a rising edge on the clock.

There is no scenario where `proc_a` runs but `proc_b` does not, because the sensitivity list for `proc_b` has a superset of the signals in `proc_a`'s sensitivity list.

Marking:

- **8 marks** *"No" with excellent justification*
- **6 marks** *"No" with correct but incomplete justification*
- **5 marks** *"Yes", describes differences in process modes (similar to correct answer to part A)*
- **3 marks** *"Yes", incomplete justification, but got part A correct*
- **3 marks** *correct use of VHDL semantics terminology*
- **2 marks** *some amount of correct justification*

Q1c (8 Marks) Register-transfer-level differences

Is there a scenario where the signals `clk`, `ce`, and `d` have the same behaviour for both `proc_a` and `proc_b`, but where, when doing *register-transfer-level* simulation, the signal `q` in `proc_a` has a different behaviour than the signal `q` in `proc_b`? **For full marks, you must justify your answer.**

NOTES:

1. If there would be a difference, describe the behaviour on `clk`, `ce`, and `d` that would lead to the difference and describe the difference in the two `q` signals.

Answer:

No, there is no scenario where the two q signals will have different behaviour in register-transfer-level simulation. RTL simulation gives the same values as delta-cycle simulation at the end of the simulation round. Because the two q signals will have the same behaviour in delta-cycle simulation, they will also necessarily have the same behaviour in RTL simulation.

Marking:

8 marks	*"No" with excellent justification*
6 marks	*"No", correct, but incomplete justification*
5 marks	*"No", describe synthesized hardware*
5 marks	*"Yes", could have combinational loops in other code*
4 marks	*"Yes", correct answer to delta-cycle simulation*
3 marks	*correct use of RTL semantics terminology*
2 marks	*some amount of correct justification*

Q2 (25 Marks) The Return of the Three

(estimated time: 20 minutes)

NOTES:

1. For each of the code fragments Q2a–Q2e, answer whether the code is legal VHDL.
2. If the code is legal VHDL, answer whether it is synthesizable.
3. If the code is synthesizable, answer whether it adheres to good coding practices, according to the guidelines for E&CE 327.
4. If the the code is not legal, or is not synthesizable, or does not follow good coding practices, explain why.
5. Each of the code fragments is the body of an architecture.
6. The signals are declared as follows:

```
clk, a, b, c, d, d0, d1, q, x, y  :  std_logic
m, n                              :  unsigned(3 downto 0)
```

Q2a

```
process (clk) begin
  if rising_edge(clk) then
    q <= d0;
  else
    q <= d1;
  end if;
end process;
```

legal, unsynth: if rising_edge with else

Q2b

```
process (a, b, c) begin
  if a = '1' then
    x <= b and c;
    y <= b or  c;
  else
    y <= not b;
  end if;
end process;
```

legal, synth, bad: latch: for combinational process, must assign to all signals in both branches of if-then-else

Q2c

```
m(0) <= n(0);
m_for : for i in 1 to 3 generate
  process ( m, n ) begin
    m(i) <= m(i-1) xor n(i);
  end process;
end generate;
```

legal, synth, good;
or legal, synth, bad: doing Boolean operation (xor) on numeric signal

Q2d

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

legal, unsynth: wait-for is unsynthesizable

Q2e

```
process ( a, n ) begin
  if a = '1' then
    m <= to_unsigned( 0, 4 );
  else
    m <= n + 1;
  end if;
end process;
process ( a, m ) begin
  if a = '1' then
    n <= to_unsigned( 0, 4 );
  else
    n <= m + 2;
  end if;
end process;
```

legal, synth, bad: combinational loop

Marking:

In the set of tables below, the X-axis represents the submitted answer and the Y-axis represents the correct answer. The points on the axes are: (illegal; legal, unsynthesizable; legal, synthesizable, bad; legal, synthesizable, good)

The charts on the diagonal, which have double frames, represent the situation when the submitted answer is correct.

Inside a frame, the "Y" and "N" refer to "yes" and "no" for "legal", "synthesizable" and "good". The second column is the marks earned for the "Y" and "N" answers. The third column is the maximum possible mark earned for the explanation.

Each part of a submitted answer that is correct is marked in bold.

The maximum marks that an answer can earn is shown on the bottom line — the actual marks may be less, depending on the quality of the explanation.

For example, if the correct answer is legal, synthesizable, bad practice, then we will be looking at the third row. If the submitted answer is legal, unsynthesizable, then we will be looking at the second column. This earns a maximum of 3 marks: 1 mark for answering "legal" correctly and up to 2 marks for the explanation.

N *1 2* **3**	Y 0 N 0 2 **2**	Y 0 Y 0 N 0 2 **2**	Y Y Y **0.5**
N 0 2 **2**	**Y** *1* **N** *1 2* **4**	**Y** 1 Y 0 N 0 2 **3**	**Y** 1 Y Y **1**
N 0 2 **2**	**Y** 1 N 2 **3**	**Y** *1* **Y** *1* **N** *1 2* **5**	**Y** 1 **Y** 1 Y **2**
N 0 2 **2**	**Y** 1 N 0 2 **3**	**Y** 1 **Y** 1 N 0 2 **4**	**Y** **Y** **Y** **5**

By the charts the total number of marks for the question is 23. To add up to a total of 25, 1 additional mark was earned if at least 3 subquestions were answered and 1 additional mark (for a total of 2 additional marks) was earned if all 5 subquestions were answered.

Q3 (26 Marks) Dataflow Diagrams

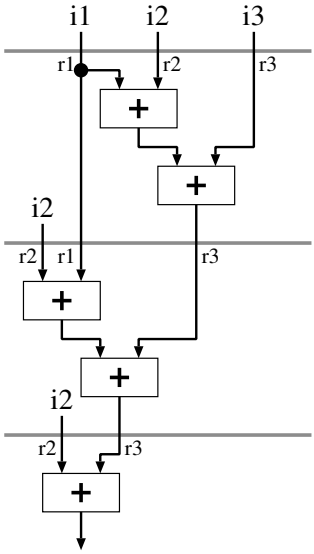
(estimated time: 20 minutes)

For each of the code fragments A, B, C, and D on the next page, answer whether the code fragment has the same behaviour as the dataflow diagram below.

If a VHDL code fragment does *not* have the same behaviour as the dataflow diagram, give the first signal (`r1`, `r2`, `r3`, or `o1`) and clock cycle where the behaviour differs, where clock cycles are counted from the first data values being on the inputs at cycle 0.

NOTES:

1. "Same behaviour" means that the registers and the output have the same values as in the dataflow diagram at the end of each clock cycle.
2. Each of the VHDL code fragments is legal, synthesizable VHDL.
3. The signal `clk` is declared as `std_logic` and each of the other signals is declared as `unsigned(7 downto 0)`.
4. Each of the code fragments is the body of an architecture.



Marking:

Correct answer is "yes"

6 marks *Correct answer*

Correct answer is "no"

2 marks *Correctly answers same behaviour = "no"*

5 marks *Correct signal and cycle for first differece*

3 marks *Signal and cycle have different behaviour, but not the first signal and cycle with different behaviour*

2 marks *Signal has different behaviour, but in the given clock cycle, the signal has the same behaviour as the dataflow diagram.*

Code fragment A

```
1   process begin
2     wait until rising_edge(clk);
3     r1   <= i1;
4     r2   <= i2;
5     r3   <= i3;
6     add1 <= r1 + r2;
7     add2 <= add1 + r3;
8     wait until rising_edge(clk);
9     r2   <= i2;
10    r3   <= add2;
11    add1 <= r1 + r2;
12    add2 <= add1 + r3;
13    wait until rising_edge(clk);
14    r2   <= i2;
15    r3   <= add2;
16    add1 <= r2 + r3;
17    o1   <= add1;
18  end process;
```

incorrect (datapath components should be combinational, but they use registered assignments) r3 in cycle 2

Code fragment B

```
1   process begin
2     wait until rising_edge(clk);
3     r1 <= i1;
4     r2 <= i2;
5     r3 <= i3;
6     wait until rising_edge(clk);
7     r2 <= i2;
8     r3 <= r1 + r2 + r3;
9     wait until rising_edge(clk);
10    r2 <= i2;
11    r3 <= r1 + r2 + r3;
12  end process;
13  o1 <= r2 + r3;
```

correct

Code fragment C

```
 1  process begin
 2    wait until rising_edge(clk);
 3    r1   <= i1;
 4    wait until rising_edge(clk);
 5    wait until rising_edge(clk);
 6  end process;
 7  process begin
 8    wait until rising_edge(clk);
 9    r2   <= i2;
10    wait until rising_edge(clk);
11    r2   <= i2;
12    wait until rising_edge(clk);
13    r2   <= i2;
14  end process;
15  process begin
16    wait until rising_edge(clk);
17    r3   <= i3;
18    wait until rising_edge(clk);
19    r3   <= r1 + r2 + r3;
20    wait until rising_edge(clk);
21    r3   <= r1 + r2 + r3;
22  end process;
23  o1 <= r2 + r3;
```

correct

Code fragment D

```
 1  process begin
 2    wait until rising_edge(clk);
 3    r1   <= i1;
 4  end process;
 5  process begin
 6    wait until rising_edge(clk);
 7    r2   <= i2;
 8  end process;
 9  process begin
10    wait until rising_edge(clk);
11    r3   <= i3;
12    wait until rising_edge(clk);
13    r3   <= add2;
14    wait until rising_edge(clk);
15    r3   <= add2;
16  end process;
17  add1 <= r1   + r2;
18  add2 <= add1 + r3;
19  o1   <= r2 + r3;
```

incorrect (r1 should have a chip-enable): r1 in cycle 2

Q4 (25 Marks) Performance

(estimated time: 20 minutes)

You are developing a new design of the Waterluvian filter that will be implemented on an FPGA chip (the definition of a Waterluvian filter is irrelevant to this question). Your project leader just returned from a conference where she learned about a new FPGA chip. She has asked you to analyze the tradeoffs between remaining with your current FPGA chip, or delaying the project and switching to the new chip.

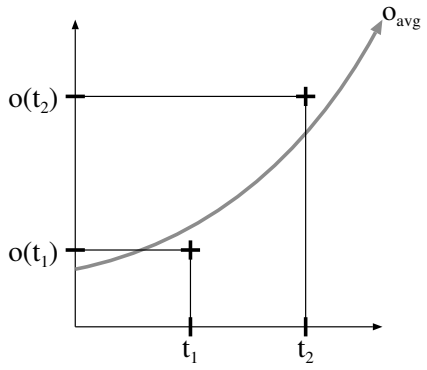
- The current prediction is that with the current FPGA chip, the average Waterluvian filter at the time you complete the project will have the 5% more area than your filter and be 25% faster than your filter.
- The average *performance* of Waterluvian filters doubles every 24 months.
- The average *area* of the circuits that implement Waterluvian filters has remained constant over time.
- The new FPGA chip claims that it will provide a 20% improvement in clock speed with just 85% of the area of the FPGA chip you are using now.
- The *optimality* of Waterluvian filters is measured by the ratio of performance to area.

If you keep your design the same and switch to the new chip, how long can you delay your project and have an *optimality* that is at least 7% more than the average *optimality* at the time you complete the project?

If you cannot achieve an optimality that is 7% more than the average, then calculate the optimality of your filter compared to the average if you use the new FPGA chip and delay the project by four months.

If you do not know how to answer the question for optimality, you may earn part marks by writing a $\surd$ in the box at the end of this question and answering the question for *performance* rather than optimality.

Answer:



$$o_{avg}(t_1) = \frac{1.25}{1.05} \times o(t_1) \quad (1)$$

$$o_{avg}(t) = o_{avg}(t_1) \times 2^{\frac{t-t_1}{24}} \quad (2)$$

$$o(t_2) = \frac{1.2}{0.85} \times o(t_1) \quad (3)$$

$$\frac{o(t_2)}{o_{avg}(t_2)} = 1.07 \quad (4)$$

Solve for $t_2 - t_1$.

To get $t_2 - t_1$, we will need to use (2), which means that we first need to solve for $\frac{o_{avg}(t_2)}{o_{avg}(t_1)}$.

Because we are solving for a ratio, setup chain of ratios:

$$\begin{aligned} & & (4) & & (3) & & (1) \\ \frac{o_{avg}(t_2)}{o_{avg}(t_1)} &= \frac{o_{avg}(t_2)}{o(t_2)} & \times & \frac{o(t_2)}{o(t_1)} & \times & \frac{o(t_1)}{o_{avg}(t_1)} \\ &= \frac{1}{1.07} & \times & \frac{1.2}{0.85} & \times & \frac{1.05}{1.25} \\ &= 1.1080 \end{aligned}$$

Substitute into (2):

$$\begin{aligned} \frac{o_{avg}(t_2)}{o_{avg}(t_1)} &= 2^{\frac{t_2-t_1}{24}} \\ 1.1080 &= 2^{\frac{t_2-t_1}{24}} \\ \log_2 1.1080 &= \frac{t_2-t_1}{24} \\ 24 \times \log_2 1.1080 &= t_2 - t_1 \\ 24 \times \frac{\log_{10} 1.1080}{\log_{10} 2} &= t_2 - t_1 \\ t_2 - t_1 &= 3.55 months \end{aligned}$$

If we switch to the new FPGA chip, we can delay the project by 3.55 months and still achieve an optimality that is 7% more than average.

Marking:

3 marks

$$\begin{aligned} o_{avg}(t_1) &= \frac{1.25}{1.05} \times o(t_1) \\ &= 1.1904 o(t_1) \\ &= \frac{1}{0.84} \times o(t_1) \end{aligned}$$

8 marks

$$o_{avg}(t) = o_{avg}(t_1) \times 2^{\frac{t-t_1}{24}}$$

3 marks

$$\begin{aligned} o(t_2) &= \frac{1.2}{0.85} \times o(t_1) \\ &= 1.4117 \times o(t_1) \\ &= \frac{1}{0.7083} \times o(t_1) \end{aligned}$$

3 marks

$$\begin{aligned} \frac{o(t_2)}{o_{avg}(t_2)} &= 1.07 \\ &= \frac{1}{0.9345} \end{aligned}$$

5 marks *Overall strategy and clarity*
3 marks *Taking log to remove exponential*
-1 mark *Simple arithmetic mistake*
-5 marks *Using performance rather than optimality*
-8 marks *Treating rate of performance increase as linear*