ECE-327Midterm

2018t1 (Winter)

Instructions and General Information

- 100 marks total
- If you need extra *scratch* paper, request some from a proctor. **All answers to be marked must be on the exam paper.** Scratch paper will not be marked.
- The proctors and instructors will **not answer questions**, except in cases where an error on the exam is suspected. If you are confused about a question, write down your assumptions or interpretation.
- **Justifications** of answers will be marked according to correctness, clarity, and concision.
- Write neatly.
- To earn part marks, you must show the formulas you use and all of your work.

		Total	**Approx.**	
		Marks	**Time**	**Page**
Q0	!!Almost Free!!	2	2	2
Q1	VHDL Semantics	20	10	3
Q2	Code Optimization	20	15	4
Q3	FSM	20	15	9
Q4	DFD	25	18	11
Q5	Memory	15	10	13
Totals		100	70	

Q0 (2 Marks) !!Almost Free!!

(*estimated time: 2 minutes*)

Q0a (1 Mark) Best part

What is the best part of the course?

Q0b (1 Mark) Most improve

What one thing could be done to most improve the course for the remainder of the term?

Q1 (20 Marks) VHDL Semantics

(*estimated time: 10 minutes*)

In VHDL, when do assignments to signals become visible?

Over the past few years, you have been very successful at creating new hardware description languages that are variations of VHDL (Synflopsys, Why Wait, *etc.*). You have earned billions of dollars and are now retired, living on your own tropical island. Despite your luxurious life of leisure, you still have the urge to explore new hardware description languages.

Your next language, the Island Description Language for Electronics (Idle) uses the same syntax as VHDL, but changes the semantics so that assignments to signals become visible *at the end of the simulation cycle in which the assignment was executed*.

If you compare Idle and VHDL by simulating the same program in an Idle simulator and a VHDL simulator, will each signal have the same value in both simulators at the beginning of every nanosecond of simulation? **For full marks, you must justify your answer.**

All programs will have the same behaviour in VHDL and Idle ☐
Some programs will have the same behaviour in VHDL and Idle ☐
No programs will have the same behaviour in VHDL and Idle ☐

Justification:

Q2 (20 Marks) Code Optimization

(*estimated time: 15 minutes*)

In this question, you will analyze the VHDL program `payette`, whose entity is on this page and whose architecture is on the next page.

NOTES:

1. Signal and state names are *not* related to the purpose of the signal or state, except `clk`, `reset`, and `state`.

Q2a (5 Marks) Encoding

Based upon the ECE-327 guidelines, which state-encoding should be used with `payette`? **For full marks, you must justify your answer.**

Show your encoding for the states:

SA	
SB	
SC	
SD	

Q2b (15 Marks) Optimizations

On the next page, use the recommended state encoding from Q2a to optimize the code.

NOTES:

1. Your optimizations shall *not* affect the clock-cycle by clock-cycle behaviour of `p`, `y`, or `z`.
2. You may change, add, and delete lines of code.

```
entity payette is
  port ( clk, reset : in std_logic;
         a : in std_logic;
         b : in signed( 7 downto 0 );
         y : out std_logic;
         z : out signed( 7 downto 0 )
       );
end entity;
```

The Next Page Contains a Copy for Scratch Work

```
architecture main of payette is
  signal p : signed( 7 downto 0 );
  type state_ty is ( SA, SB, SC, SD );
  signal state : state_ty;
begin

  process begin
    wait until rising_edge(clk);
    if reset = '1' then
      state <= SD;
    else
      if state = SD or state = SA then
        if a = '0' then
          state <= SA;
        else
          state <= SC;
        end if;
      elsif state = SC then
        state <= SB;
      else
        state <= SD;
      end if;
    end if;
  end process;

  process begin
    wait until rising_edge(clk);
    if state = SB then
      p <= b;
    elsif state = SA
          or (state = SC and a='1')
    then
      p <= p + b;
    end if;
  end process;

  z <= p;
  y <=   '1' when p > 17
    else '0';

end architecture;
```

Extra Copy for Scratch Work

```
architecture main of payette is
  signal p : signed( 7 downto 0 );
  type state_ty is ( SA, SB, SC, SD );
  signal state : state_ty;
begin

  process begin
    wait until rising_edge(clk);
    if reset = '1' then
      state <= SD;
    else
      if state = SD or state = SA then
        if a = '0' then
          state <= SA;
        else
          state <= SC;
        end if;
      elsif state = SC then
        state <= SB;
      else
        state <= SD;
      end if;
    end if;
  end process;

  process begin
    wait until rising_edge(clk);
    if state = SB then
      p <= b;
    elsif state = SA
          or (state = SC and a='1') then
      p <= p + b;
    end if;
  end process;

  z <= p;
  y <=   '1' when p > 17
    else '0';

end architecture;
```

This Page Intentionally Blank

Q3 (20 Marks) FSM

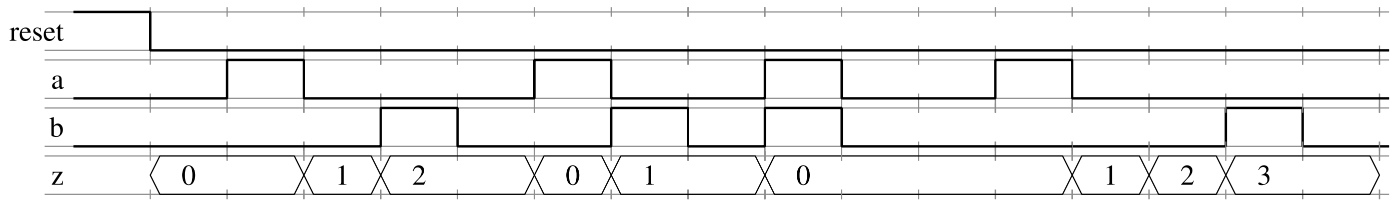
(*estimated time: 15 minutes*)

Draw a state machine that counts the number of clock cycles between `a='1'` and `b='1'`.

NOTES:

1. The system has three inputs: `reset`, `a`, and `b`.
2. The system has one output: `z`.
3. From the first clock cycle when `reset` is deasserted (changes from `'1'` to `'0'`) up to but not including the clock cycle when `a='1'`, `z` shall be `0`.
4. In the clock cycle when `a='1'`, `z` shall be `0`.
5. From the clock cycle after `a='1'` up to and including the clock cycle when `b='1'`, `z` shall increment in each clock cycle.
6. From the clock cycle after `b='1'` up to but not including the clock cycle when `a='1'`, `z` shall hold its value constant, showing the number of clock cycles between `a='1'` and `b='1'`.
7. After `a` changes from `'1'` to `'0'`, it is guaranteed to remain `'0'` until the clock cycle after `b='1'`.
8. Marks will be earned for the state machine being syntactically correct, functionally correct, simple and elegant, and drawn neatly.

Example execution:



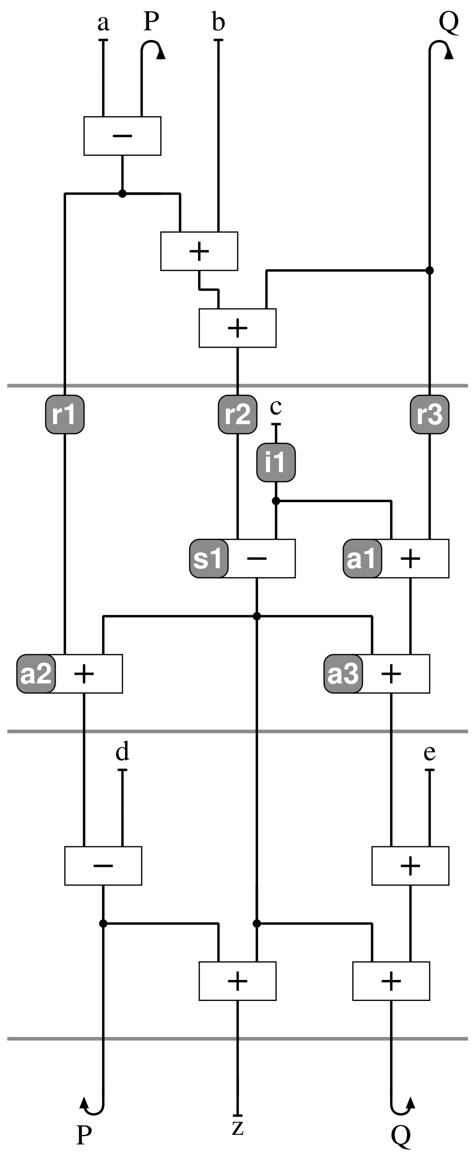
Q4 (25 Marks) DFD

(*estimated time: 18 minutes*)

Complete the allocation for the dataflow diagram below, draw the control table, and analyze your allocation.

NOTES:

1. Optimization goal: minimize number of multiplexers
2. You may *not* use any scheduling optimizations.
3. You may *not* change any of the existing allocations.
4. The *only* algebraic optimization you may use is commutativity.

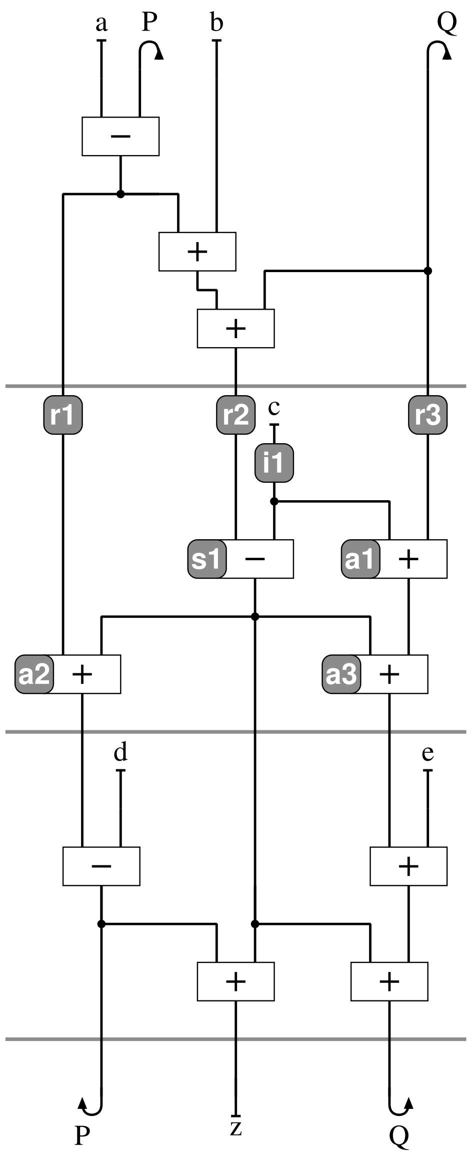


Control Table

Analysis

2:1 Multiplexers	
Latency	
Throughput	
Clock period	

Extra copy for scratch work



Q5 (15 Marks) Memory

(*estimated time: 10 minutes*)

Q5a (8 Marks) Optimization

Optimize the pseudocode below as if it were to be used as the specification for a dataflow diagram.

NOTES:

1. Optimization goals, in order of *decreasing* priority:
   (a) Minimize overall latency of the program
   (b) Minimize latency to z
2. It is guaranteed that `a` < `d`.
3. The variables `a`, `b`, `c`, and `d` do *not* change their values during execution.
4. Use the standard type of dual-port memory from ECE-327.
5. Scheduling and other optimizations are allowed, as long as they do *not* affect the final values of `M` or `z`.

```
M[a+1] = b
M[a+2] = c
M[d]   = M[a+2]
M[d+1] = M[a]   + M[a+2]
e      = M[a]
z      = M[e]
```

This question continues on the next page

Q5b (5 Marks) Latency

If you were to design a DFD to implement your optimized pseudocode, what would be the minimum possible latency to `z` and the minimum possible overall latency?

Latency to `z`	
Overall latency	

Q5c (2 Marks) Single-Port Memory

If you switch to single-port memory, will you need to increase the overall latency of the program? **For full marks, you must justify your answer.**

This Page Intentionally Blank