M A S S A C H U S E T T S I N S T I T U T E O F T E C H N O L O G Y
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

6.375 Complex Digital Systems
Spring 2008 - Quiz - March 21, 2008
80 Minutes

NAME: _____ SCORE: _____

Please write your name on every page of the quiz.

Not all questions are of equal difficulty, so look over the entire quiz and budget your time carefully.

Please carefully state any assumptions you make.

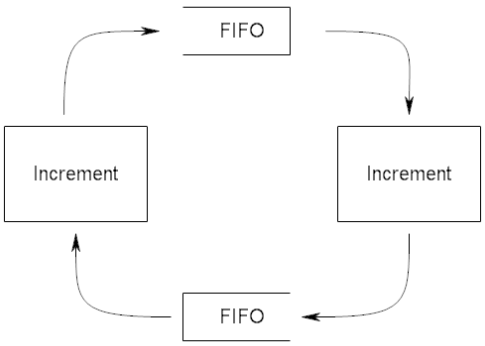
Enter your answers in the spaces provided below. If you need extra room for an answer or for scratch work, you may use the back of each page but please *clearly indicate where your answer is located.*

You must not discuss the quiz's contents with other students who have not yet taken the quiz. If, prior to taking it, you are inadvertently exposed to material in a quiz — by whatever means — you must immediately inform the instructor or a TA.

	Points	**Score**
Problem 1	20	
Problem 2	25	
Problem 3	25	
Problem 4	10	
Problem 5	20	

Problem 1 : Throughput (20 total points)

Ben Bitdiddle made a dual upcounter:



using the following code:

```
module temp();
   Reg#(Bit#(1)) stage <- mkReg(0);
   FIFO#(int)    fifoA <- mkFIFO1();
   FIFO#(int)    fifoB <- mkFIFO1();
   rule init (stage == 0);
      fifoA.enq(1);
      fifoB.enq(1);
      stage <= 1;
   endrule
   rule inc1 (stage == 1);
      let temp = fifoA.first();
      fifoA.deq();
      $display("Inc1: %d", temp);
      fifoB.enq(temp+1);
   endrule
   rule inc2 (stage == 1);
      let temp = fifoB.first();
      fifoB.deq();
      $display("Inc2: %d", temp);
      fifoA.enq(temp+1);
   endrule
   rule exit ((fifoA.first() == 6) || (fifoB.first() == 6));
      $finish();
   endrule
endmodule
```

He was expecting to see the following display:

Inc1: 1 Inc2: 1 Inc1: 2 Inc2: 2 Inc1: 3 Inc2: 3 Inc1: 4 Inc2: 4 Inc1: 5 Inc2: 5

1.1 (10 points)
The code compiled, but on simulation, he did not see any display statements. Why is the code not executing correctly?

Solution:
Since fifoA and fifoB are single element FIFOs, they are full after rule init. Rule inc1 and inc2 have implicit predicates of fifoA.notFull and fifoB.notFull. This leads to a deadlock where neither rule fires.

1.2 (10 points)
Modify the code to get the desired execution. You can use library elements such as those used in the labs.

Solution:
Making fifoA and fifoB 2-element FIFOs resolves the deadlock.

```
FIFO#(int)    fifoA <- mkFIFO();
FIFO#(int)    fifoB <- mkFIFO();
```

Problem 2 : Bluespec Semantics (25 total points)

Consider the code given below:

```
module sem();
   Reg#(int)     count <- mkReg(1);
   Reg#(int)     a <- mkReg(1);
   Reg#(int)     b <- mkReg(2);
   Reg#(int)     c <- mkReg(3);
   rule counter (True);
      count <= count + 1;
   endrule
   rule mod1 (True);
      a <= b + c;
   endrule
   rule mod2 (True);
      b <= c + count;
   endrule
   rule mod3 (True);
      c <= b + count;
   endrule
   rule exit (count >= 4);
      $finish();
   endrule
endmodule
```

2.1 (6 points)

What are the sequential composability conditions deduced by the compiler?

Solution:

The following composability relations are seen:

```
exit < counter
mod2 or mod3 < counter
mod1 < mod2 or mod3
mod2 C mod3
```

2.2 (9 points)

Using the above conditions, assume an overall order and determine the values of all the state elements at finish.

Solution:

Any order which satisfies the above conditions is valid. Assuming the following order of execution:

```
exit < mod1 < mod2 < counter
```

Value of state elelments:
Cycle 0: a = 1; b = 2; c = 3; count = 1
Cycle 1: a = 5; b = 4; c = 3; count = 2
Cycle 2: a = 7; b = 5; c = 3; count = 3
Cycle 3: a = 8; b = 6; c = 3; count = 4

2.3 (10 points)
Modify the code such that all the rules fire every cycle in the following order:

```
counter | mod1 < mod2 < mod3 < exit
```

You can use EHRs of any order.

Solution:
Use EHR components:

```
EHR3_Reg#(int)     count <- mkEHR3_Reg(1);
Reg#(int)          a <- mkReg(1);
EHR2_Reg#(int)     b <- mkEHR2_Reg(2);
EHR2_Reg#(int)     c <- mkEHR2_Reg(3);

rule counter (True);
   count.write_0(count.read_0 + 1);
endrule

rule mod1 (True);
   a <= (b.read_0 + c.read_0);
endrule

rule mod2 (True);
   b.write_0(c.read_1 + count.read_1);
endrule

rule mod3 (True);
   c.write_1(b.read_1() + count.read_1());
endrule

rule exit (count.read_2 == 4);
   $finish();
endrule
```

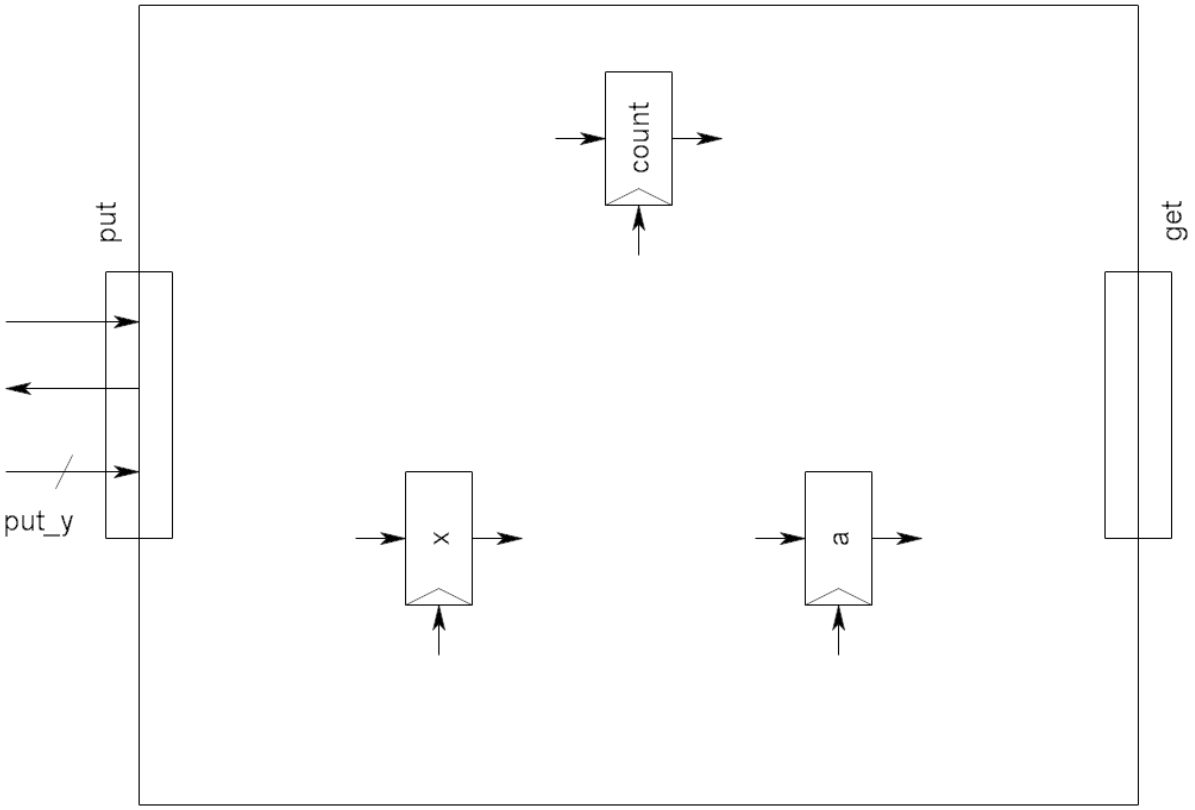
Problem 3 : Bluespec Synthesis (25 total points)

Consider the code shown below:

```
module mkMultBySixDyn(Foo#(int));
  Reg#(int) a     <- mkReg(0);
  Reg#(int) x     <- mkReg(0);
  Reg#(int) count <- mkReg(0);
  rule mulDyn (count>0 && count<6);
    count <= count+1;
    a     <= a+x;
  endrule
  method Action put (int y) if (count==0);
    a <= y; x <= y;
    count <= 1;
  endmethod
  method ActionValue#(int) get if (count==6);
    count <= 0;
    return a;
  endmethod
endmodule
```

3.1: 15 points

Sketch the hardware produced on compiling this code. Label the interface signals, scheduling logic and signals corresponding to CAN_FIRE_mulDyn and WILL_FIRE_mulDyn.



3.2: 10 points
The rule mulDyn is replaced by the following rule:

```
rule mulStat (count>0 && count<6);
  for(int i = 1; i<6; i++)
     if(count==i)
      begin
        count <= i+1; a <= a+x;
      end
endrule
```

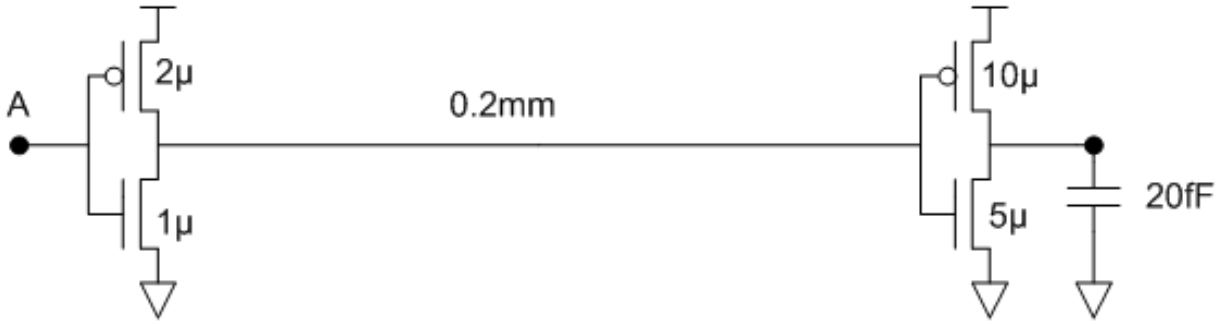
How does this change affect the hardware generated? Which implementation mulDyn or mulStat has more adders? More muxes? Compare the overall area and critical paths of the implementations.

Solution:
a) The for loop in MultByStat is statically elaborated to generate a sequence of add statements each gated by the value of count. Assuming no optimization, MultbySixStat has more adders, more muxes, larger area and a longer critical path.
Bluespec actually optimizes number of adders, and MultByStat actually has just one adder compared to 2 adders for MultByDyn. Other answers remain the same. No points deducted for missing the optimization.

b) In the hardware sketch, the following are important:
Update logic for state variables, adders, can fire and will fire signals, methods' rdy and enable signals.

Problem 4 : RC Delay (10 points)

Assume you have an inverter (nmos width = 1 μm, pmos width = 2 μm) driving an interconnect of length 0.2 mm as shown in the figure. The interconnect has an inverter (nmos width = 5 μm, pmos width = 10 μm) at the other end which drives a flipflop whose input capacitance is 20 fF. Calculate the delay of the circuit from point A to point B (see figure). You can use a π model (as shown in the attached slide) for the bitline and assume that the transistors turn on after one RC time constant. The process parameters are given in the table below.



Process Parameters	**Value**
PMOS gate capacitance per μm of transistor width	1.5fF/μm
NMOS gate capacitance per μm of transistor width	1.5fF/μm
PMOS drain capacitance per μm of transistor width	0.3fF/μm
NMOS drain capacitance per μm of transistor width	0.3fF/μm
PMOS effective on resistance	6.6k$\Omega\mu$m
NMOS effective on resistance	3.3k$\Omega\mu$m
Metal 2 wire resistance per μm of length	0.4Ω/μm
Metal 2 wire capacitance per μm of length	0.2fF/μm

Solution:
$Delay = R_{eff} * (C_{dn} + C_{dp} + C_w/2) + (R_{eff} + R_w) * (C_w/2 + 5 * C_{gn} + 5 * C_{gp}) + R_{eff}/5 * (5 * C_{dn} + 5 * C_{dp} + C_{load})$
$= 3.3k*(0.3f+0.6f+20f)+(3.3k+80)*(20f+5*1.5f+5*3f)+3.3k/5*(5*0.3f+5*0.6f+20f)$
$= 228.79ps$

Problem 5 : Power (20 total points)

The following two unit designs implement the same signal processing function, with the following performance characteristics.

Unit	Throughput (million tasks/sec)	Vdd (volts)	Energy/Task (nanojoules)
Unit 1	20	1.2	5
Unit 2	10	1.2	3

The following table lists the effect of changing supply voltage on circuit delay and energy per operation, **normalized** to that at 1.2V.

Vdd	Delay	Energy/Task
1.5	0.7	1.9
1.4	0.8	1.5
1.3	0.9	1.2
1.2	**1.0**	**1.0**
1.1	1.2	0.8
1.0	1.5	0.6
0.9	2.0	0.4
0.8	10.0	0.3
0.7	non-functional	non-functional

a) Which unit is the most energy efficient for a minimum throughput of million 12.5 tasks/second? Show your work. **6 points**

Solution:
At $Vdd = 1V$*, Unit 1 has throughput of* $20/1.5 = 13.3M/s$ *with Energy/Task of* $5 * 0.6 = 3nJ$.
At $Vdd = 1.4V$*, Unit 2 has throughput of* $10/0.8 = 12.5M/s$ *with Energy/Task of* $3 * 1.5 = 4.5nJ$.
For a throughput of 12.5M/s, Unit 1 will have Energy/Task less than 3nJ, and is thus more efficient for the required throughput.

b) Which unit gives the highest performance at a maximum power dissipation of 30mW? Show your work. **6 points**

Solution:
At $Vdd = 0.9V$*, Unit 1 has throughput of* $20/2 = 10M/s$*, Energy/Task of* $5 * 0.4 = 2nJ$ *giving power dissipation of* $10M/s * 2nJ = 20mW$.
At $Vdd = 1.2V$*, Unit 2 has throughput of* $10M/s$*, Energy/Task of* $3nJ$ *giving power dissipation of* $10M/s * 3nJ = 30mW$.
For 30 mW power, Unit 1 has a throughput higher than 10M/s and thus has a higher performance for power budget.

c) Assume the signal processing function is perfectly parallelizable. What is the lowest power parallel configuration to process 20 million tasks/second? Ignore the area cost. List which unit is used, the operating voltage, and the number of parallel instances. **8 points**

Solution:
Unit 2 operating at 0.8V with 20 units has the lowest power of 18mW.