ECE 327 Midterm Solution

2013t2 (Spring)

All requests for re-marks must be submitted in writing to Mark Aagaard before 8:30am on Friday June 28.

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

Q1 (10 Marks) Simulation Semantics

(estimated time: 7 minutes)

Your manager has asked you to evaluate a new register-transfer-level simulator for VHDL named "Unifortl: the uniform RTL simulator". The distinguishing feature of Unifortl is that all processes are simulated using the same rules (*i.e.*, all categories of processes are treated uniformly).

The simulation rules are, for each moment in time:

1. Run the non-combinational processes in *any order*, processes read *old values* of signals (with the exception that clocked processes read the current value of the clock signal).
2. Run the combinational processes in *any order*, processes read *old values* of signals.

Will Unifortl simulate *all* VHDL programs correctly, *some* VHDL programs correctly, or *no* VHDL programs correctly?

NOTES:

1. Unifortl is said "not to simulate a VHDL program correctly" if either Unifortl is unable to simulate the program, or Unifortl simulates the program but produces incorrect results.
2. If Unifortl is able to simulate some VHDL programs correctly, then describe the set of programs that *are simulated correctly*.

Answer:

(This answer is much longer than necessary.)

Unifortl will simulate **some** *VHDL programs correctly. Specifically, Uniforrtl will simulate correctly VHDL programs where there are no combinational or intra-clock cycle dependencies between processes, except for the clock signal driving registers.*

To run a combinational processes correctly in zero-delay simulation, events must appear to propagate instaneously through combinational circuitry.

Normal RTL simulation achieves this illusion by supporting only programs without combinational loops, running the combinational processes in topological order, and having the combinational processes read the new values of signals.

Unifortl differs from normal RTL simulation by running the combinational processes in any order and reading the old values of signals. Thus, combinational processes will not see the new values of signals and events will not propagate instanteously through the combinational circuitry.

Marking:

10 marks max *some: programs where there are no combinational dependencies between processes*
9 marks max *some: programs without comb processes*
8 marks max *all: if run at granularity of VHDL simulation cycles*
8 marks max *none: because can't simulate comb processes correctly*
6 marks max *some: programs without combinational loops*
5 marks max *none*
50% of max possible mark *justification*

Q2 (25 Marks) The Mud, the Rain, and the Pollen

(estimated time: 20 minutes)

NOTES:

1. For each of the code fragments Q2a–Q2d:
   (a) Answer whether the code is *legal*
   (b) If the code is *illegal*: explain why, and proceed to the next code fragment.
   (c) Answer whether the code is *synthesizable*.
   (d) If the code is *unsynthesizable*: explain why, and proceed to the next code fragment.
   (e) Answer whether the code adheres to good coding practices, according to the guidelines for ECE 327.
   (f) If the code does *not follow good coding practices*: explain why.
2. For Q2a–Q2b, if the code is synthesizable: draw the gate-level circuit that would most likely result from synthesizing the code. If the VHDL code includes an implicit state machine: draw the gates, wires, and flops for the datapath; you may draw the control portion of the circuit as a cloud or black-box that drives the appropriate signals in the datapath.
3. For Q2c–Q2d, if the code is legal: compute the value of `z` at 30 ns and fill in the value on the waveform diagram.
4. The signals `clk` and `a` are `std_logic`.
5. The signals `b`, `c`, `d`, `e`, and `z` are `unsigned( 7 downto 0 )`.
6. The waveforms for `clk`, `a`, `b`, and `c` are the same for each of Q2a–Q2d. The VHDL code for these signals is:

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

```
process begin
  a <= '0';
  wait for 5 ns;
  a <= '1';
  wait for 20 ns;
  a <= '0';
  wait for 15 ns;
end process;
```

```
process begin
  b <= to_unsigned( 11, 8 );
  c <= to_unsigned( 21, 8 );
  loop
    wait for 5 ns;
    b <= b + 1;
    c <= c + 1;
  end loop;
end process;
```

Note on baseline[1]

Q2a

```
a0 : if a =  '0' generate
  z <= b;
end generate;
a1 : if a /= '0' generate
  z <= c;
end generate;
```

Answer:

Illegal: condition for generate must be static

Marking:

3.5 marks max Illegal
- **+2.0 marks** baseline[1]
- **+1.5 marks max** justification

2.0 marks max Legal, unsynth
- **+0.5 marks** baseline[1]
- **+1.5 marks max** justification

1.0 marks max Legal, synth, bad
- **+1.0 marks max** justification

0.5 marks max Legal, synth, good
- **+0.5 marks max** picture of ckt

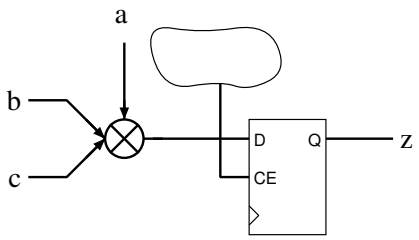
Q2b

```
process begin
  wait until rising_edge(clk);
  if a = '0' then
    z <= b;
  else
    wait until rising_edge(clk);
    z <= c;
  end if;
end process;
```

Answer:

Legal, synth, good

If said illegal, unsynth, or bad: justification would be that missing wait statement in `then` clause.



Marking:

7.5 marks max Legal, synth, good
- **+4.0 marks** baseline[1]
- **+3.5 marks** circuit drawing

6.5 marks max Legal, synth, bad
- **+1.5 marks** baseline[1]
- **+3.5 marks max** circuit drawing
- **+1.5 marks max** justification

3.0 marks max Legal, unsynth
- **+1.5 marks** baseline[1]
- **+1.5 marks max** justification

1.5 marks max Illegal
- **+1.5 marks max** justification

circuit drawing
- **+1.5 marks** mux with a, b, c
- **+1.0 marks** z is a flop
- **+1.0 marks** cloud driving ce for flop

[1]Baseline mark is earned only if justification or circuit drawing is answered. **Half of baseline if circuit/justification is missing**

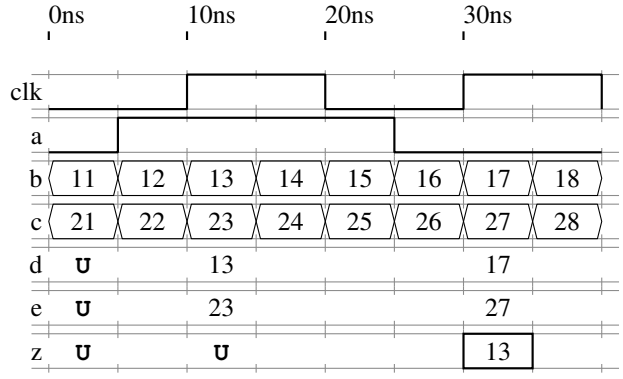
Q2c

```
process begin
  d <= b;
  e <= c;
  wait until rising_edge(clk);
  if a = '0' then
    z <= d;
  else
    z <= e;
  end if;
end process;
```

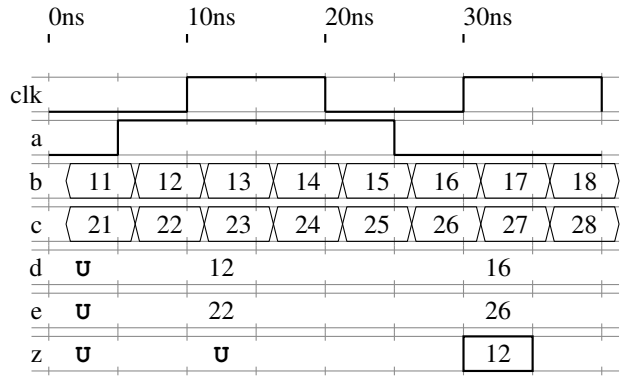
Answer:

Legal, unsynth: wait must be the first statement in the process

If noticed that `clk`, `b`, and `c` change in the same *simulation cycle:*



If treated `b` and `c` as if they were the outputs of combinational circuitry and changed in the simulation cycle after *`clk`:*



Marking:

6.5 marks max Legal, unsynth
- **+1.5 marks** baseline[1]
- **+3.5 marks max** value of z
- **+1.5 marks max** justification

5.5 marks max Legal, synth, bad
- **+0.5 marks** baseline[1]
- **+3.5 marks max** value of z
- **+1.5 marks max** justification

3.5 marks max Legal, synth, good
- **+3.5 marks max** value of z

1.5 marks max Illegal
- **+1.5 marks max** justification

value of z
- **3.5 marks** z=12
- **3.0 marks** z=13
- **2.0 marks** z=16
- **1.0 marks** z=17
- **0.5 marks** z=22,23,25,26

Q2d

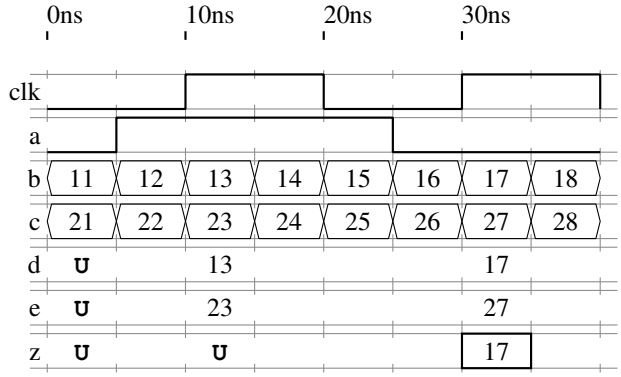
```
process begin
  wait until rising_edge(clk);
  d <= b;
  e <= c;
end process;

process (a,d,e) begin
  if a = '0' then
    z <= d;
  else
    z <= e;
  end if;
end process;
```

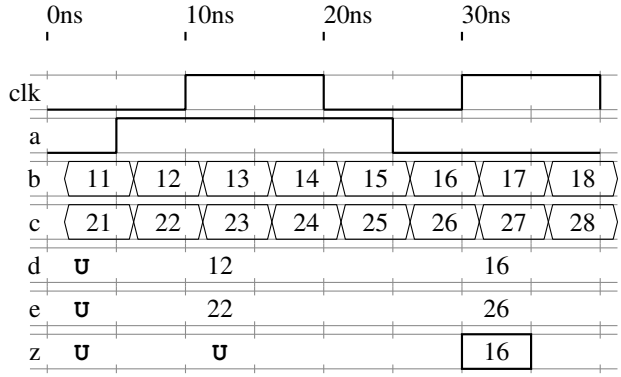
Answer:

Legal, synth, good

If noticed that `clk`*,* `b`*, and* `c` *change in the* same *simulation cycle:*



If treated `b` *and* `c` *as if they were the outputs of combinational circuitry and changed in the simulation cycle* after `clk`*:*



Marking:

7.5 marks max Legal, synth, good
- **+4.0 marks** baseline[1]
- **+3.5 marks max** value of z

6.5 marks max Legal, synth, bad
- **+1.5 marks** baseline[1]
- **+3.5 marks max** value of z
- **+1.5 marks max** justification

5.0 marks max Legal, unsynth
- **+3.5 marks max** value of z
- **+1.5 marks max** justification

1.5 marks max Illegal
- **+1.5 marks max** justification

value of z
- **3.5 marks** z=17
- **3.0 marks** z=16
- **2.0 marks** z=13
- **1.0 marks** z=12
- **0.5 marks** z=22,23,25,26

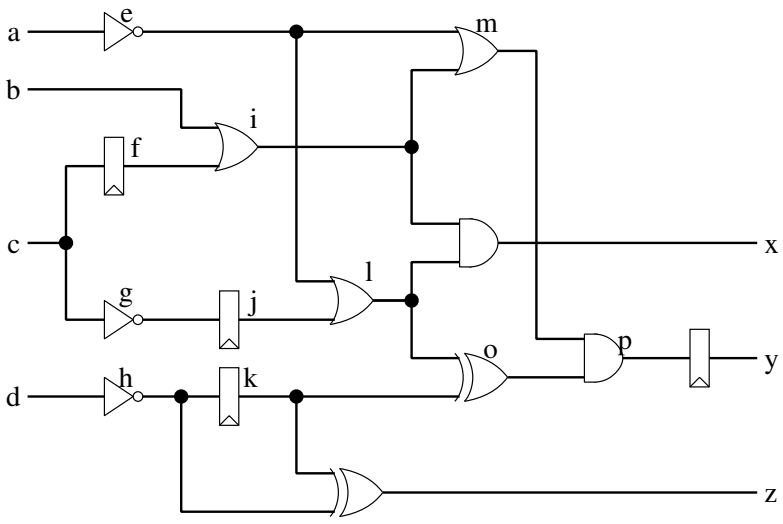
Q3 (23 Marks) FPGA Cells

(estimated time: 15 minutes)

Design an FPGA implementation of the gate-level circuit shown below that uses the minimum number of FPGA cells. Use the FPGA cells on the following page to answer the question.

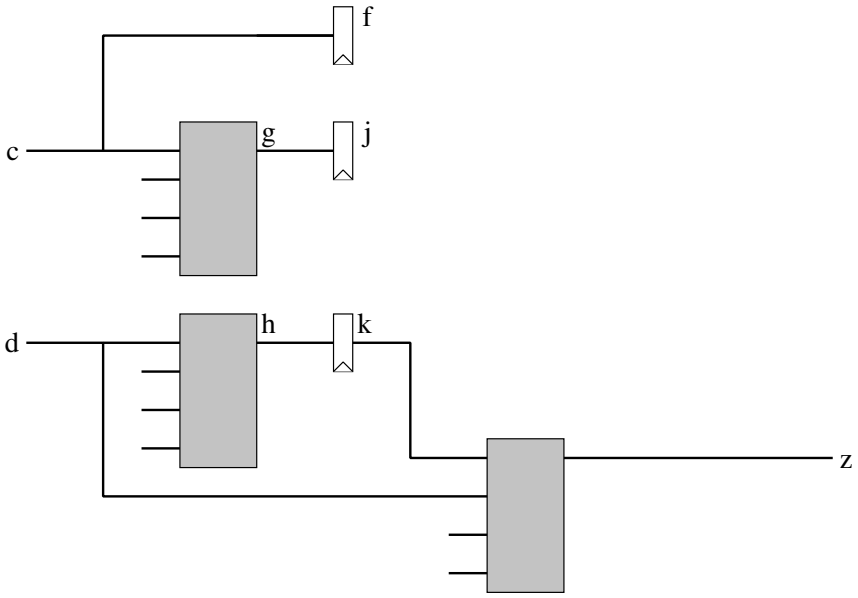
NOTES:

1. The primary inputs of the circuit are: `a`, `b`, `c`, and `d`.
2. The primary outputs of the circuit are: `x`, `y`, and `z`.
3. Do not perform any logic optimizations.
4. For each FPGA cell that you use:
   - Label the input and output ports of the cell using the signal names from the gate-level circuit for ports that you use and **NC** (for no-connect) for ports that you do not use.
   - Show the configuration for the internal multiplexer.

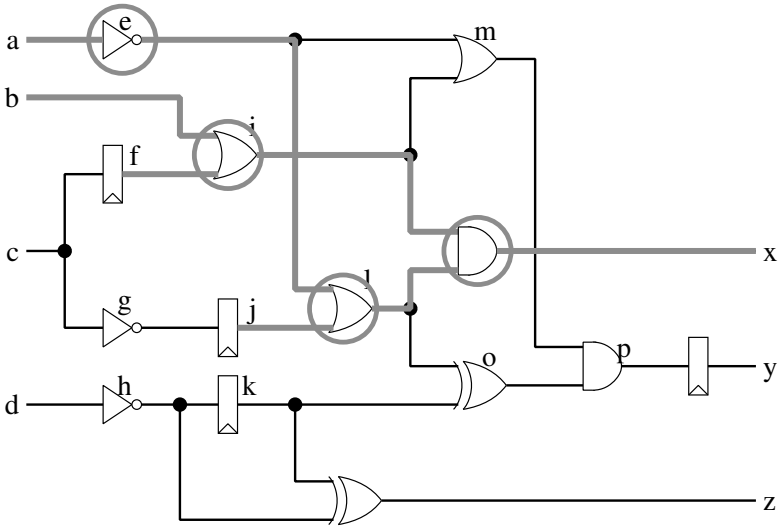


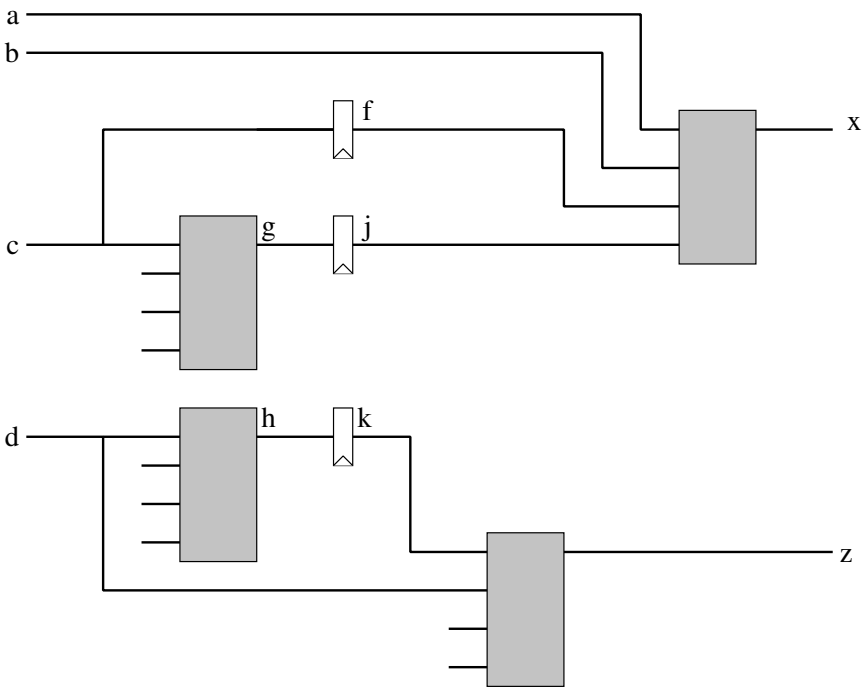
Answer:

1. *Easy cells (flops and outputs with only a few inputs to fanin-cone)*



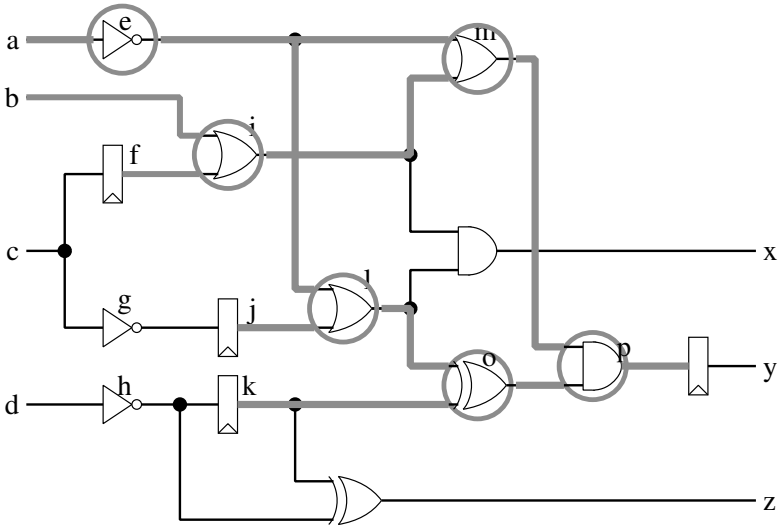
2. *Add x*



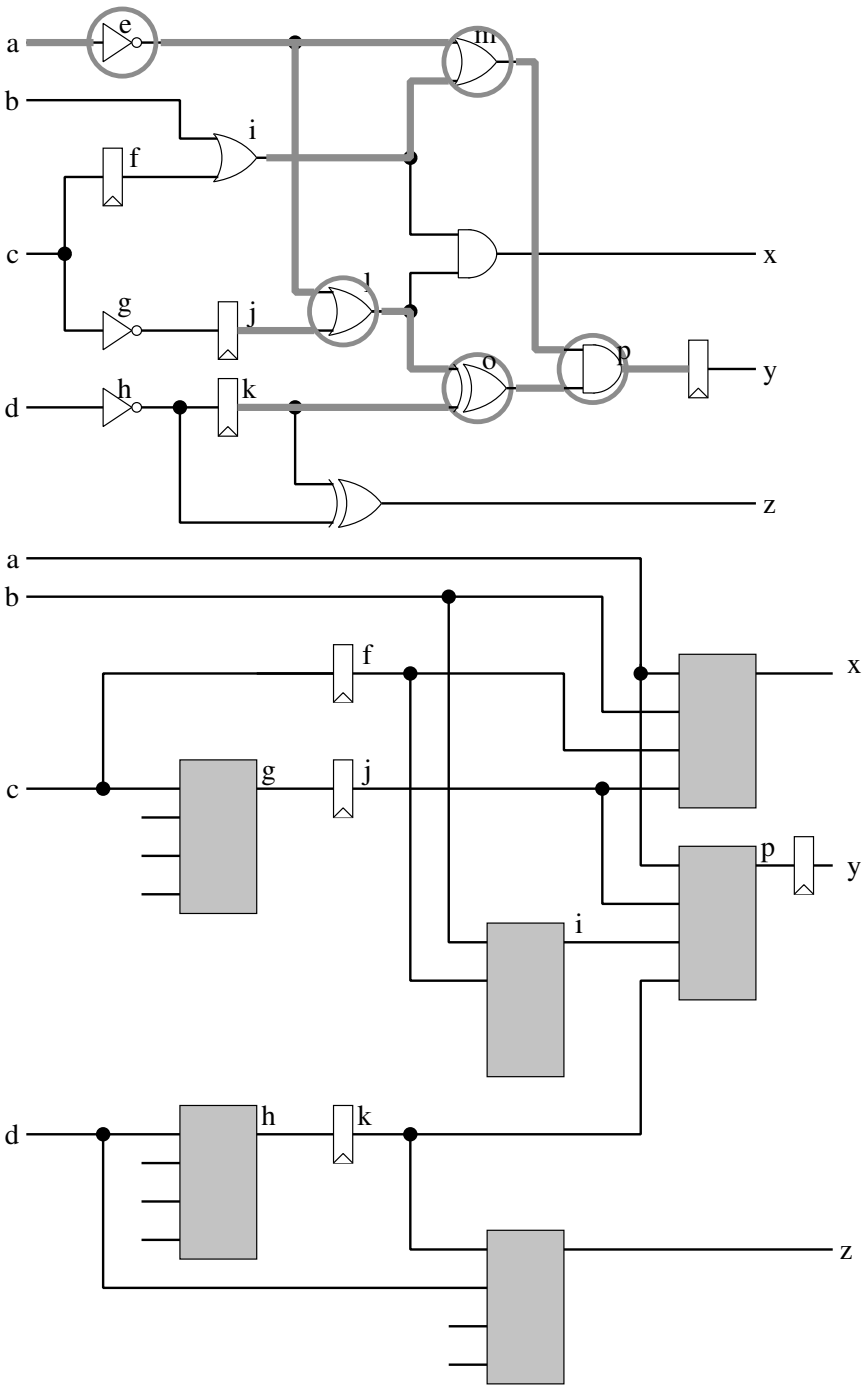


3. *Add y*

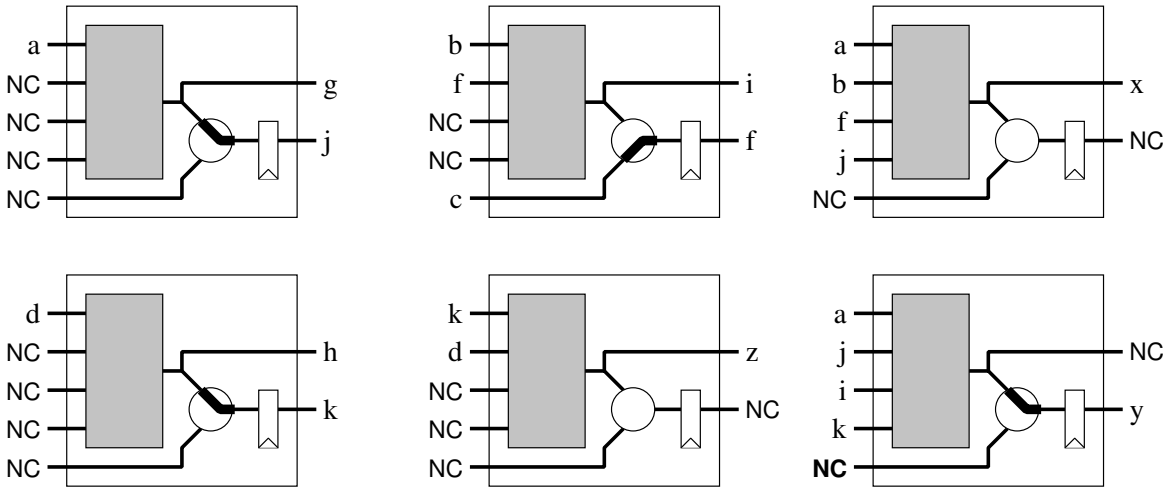
   Try complete fanin-cone for y



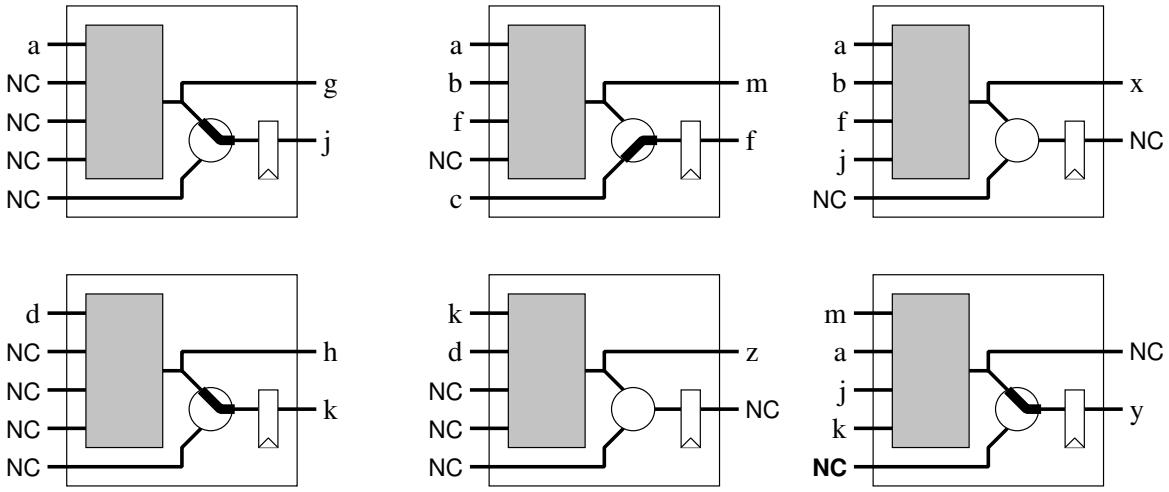
   Observe that y has 5 inputs. Need to remove some gates to reduce number of inputs to 4. Best choices are to remove either `i` *or* `m`*: requires only one additional cell.*


a
b
c
d
e
f
g
h
i
j
k
l
m
o
p
x
y
z
a
b
c
d
f
g
j
h
k
i
p
x
y
z

4. Place LUTs and flops into cells



5. Another possible answer



Marking:

23 marks *correct answer, 6 FPGA cells*
21 marks *functionally correct, 7 FPGA cells*
20 marks *functionally correct, 8 FPGA cells*
16 marks *functionally correct, 9 FPGA cells*
14 marks *1 gate per lut (10 cells)*
10 marks *at least FPGA cell completely correct*
Conceptual mistakes
- *not stopping at flop*
- *not sharing cell for unrelated lut and flop*
- *missing NCs*
- *missing mux configurations*
- *missing signal (1 mistake per signal)*

-4 marks *1 mistake*
-8 marks *2 mistakes*
-11 marks *3 mistakes*
-14 marks *4 mistakes*

-2 marks *incorrect mapping of signal to flop vs comb*
-1 mark *x stops at i*
-1 mark *labeling p (should be NC)*
-1 mark *each small mistake*

Q4 (22 Marks) State Machines

(estimated time: 15 minutes)

You are part of the design team for "Woggles", a combined smart-watch and ski-goggles. A senior manager is coming for a design review in the afternoon. Your project leader has to go out to lunch with the manager, and has left you to design the last state machine in the system before the design review.

The functional requirements of the state machine are given below, where "`a`" means that `a` is true (in VHDL: equal to `'1'`) and "`!a`" means that `a` is false (in VHDL: not equal to `'1'`):

Functional requirements:

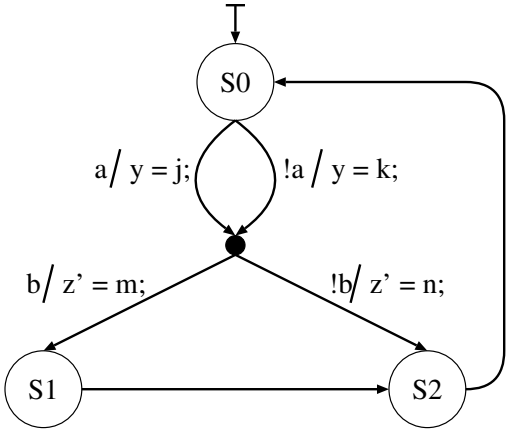
1. The inputs are: `a`, `b`, `j`, `k`, `m`, and `n`.
2. The outputs are: `y` and `z`.
3. The initial state is `S0`.
4. If (the current state is `S0`) and `a`, then in the current clock cycle assign `j` to `y`.
5. If (the current state is `S0`) and `!a`, then in the current clock cycle assign `k` to `y`.
6. If (the current state is `S0`) and `b`, then in the next clock cycle assign `m` to `z`.
7. If (the current state is `S0`) and `!b`, then in the next clock cycle assign `n` to `z`.
8. If (the current state is `S0`) and `b`, then the next state shall be `S1`.
9. If (the current state is `S0`) and `!b`, then the next state shall be `S2`.
10. If the current state is `S1`, then the next state shall be `S2`.
11. If the current state is `S2`, then the next state shall be `S0`.

NOTES:

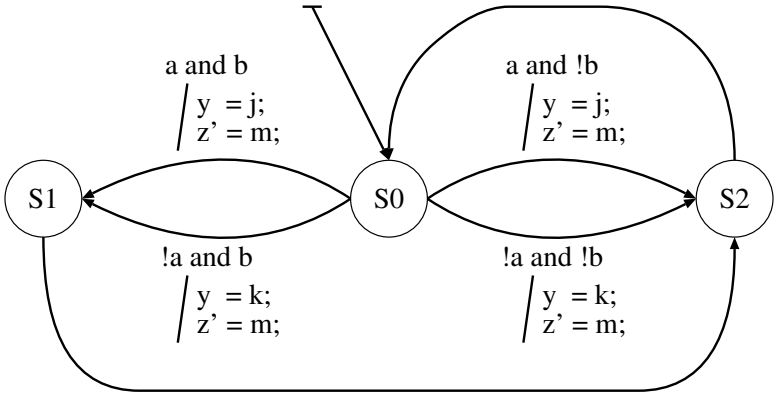
1. You may choose `y` and `z` to be either registered or combinational. Both may be combinational, both registered, or one combinational and one registered. One of the optimization goals below is to minimize the number of registers.
2. Your design shall satisfy the functional requirements.
3. Your design shall use the state-machine notation and semantics from ECE-327.
4. Your goals, in order of decreasing importance are
   (a) *minimum* number of clocked (*i.e.*, real) states
   (b) *minimum* number of edges
   (c) *minimum* number of transient states
   (d) *minimum* number of registers

Answer:

Aesthetically most pleasing design:



Removes one transient state, but has more complicated conditions and duplicates combinational assignments.



Marking:

Syntactic mistakes

- *Missing initial state*
- *Not using "/" to delimit condition vs assignment*
- *Multiple condition/assignment pairs per edge*

Functional mistakes

- *Combinational vs registered assignment*
- *Sampling condition in wrong state*
- *Not transitioning to correct state*

Optimality deficiencies

- *Extra transient state with duplicate assignments to* a
- *y is registered*

-3 marks (19) *1 mistake*
-6 marks (16) *2 mistakes*
-8 marks (14) *3 mistakes*
-10 marks (12) *4 mistakes*
-1 mark *each small mistake*

Q5 (20 Marks) Datapath Design

(estimated time: 15 minutes)

Draw a dataflow diagram that implements the specification:

spec was intended to be z = Q + a - c

```
P = P - a + b + c;
Q = Q + d;
z = P + a - c;
```

NOTES:

1. Inputs shall be *registered*
2. Outputs may be *either registered or combinational*
3. Optimization goals in order of decreasing importance:
   (a) Minimize *adders*
   (b) Minimize *subtracters*
   (c) Minimize *clock period*
   (d) Minimize *latency to* `z` (latency from first input to `z`)
   (e) Minimize *registers*
   (f) Minimize *inputs*
4. Input values may be read in any clock cycle, but each input value shall be read exactly once.
5. Optimizations to the pseudocode are allowed, so long as the final value on `P`, `Q`, and `z` are correct.

Q5a (15 Marks) Dataflow Diagram

Answer:

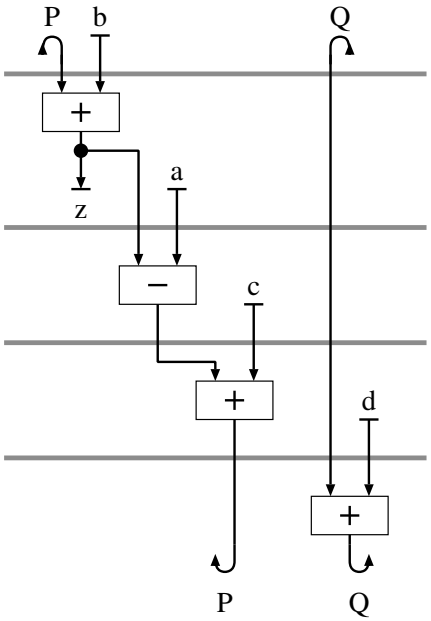
Correct semantics *z sees the new value of P*

Expression for z simplifies to `P + b`*:*

```
z = P + b;
P = P - a + b + c;
Q = Q + d;
```

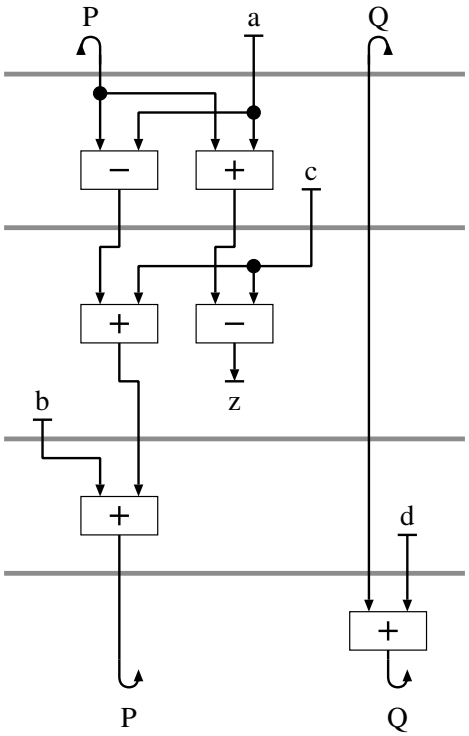
Compute `P + b` *just once:*

```
z = P + b;
P = z - a + c;
Q = Q + d;
```



Latency to z	*1*
Minimum clock period	*flop + max(Add, Sub)*
Number of inputs	*1*
Number of registers	*4*
Number of adders	*1*
Number of subtracters	*1*
Best state encoding	*valid bit*

Incorrect semantics *z sees the old value of P*



Latency to z	*2*
Minimum clock period	*flop + max(Add, Sub)*
Number of inputs	*1*
Number of registers	*4*
Number of adders	*1*
Number of subtracters	*1*
Best state encoding	*valid bit*

Marking:

- **2 marks** *P and Q are inter-parcel variables with correct semantics*
- **2 marks** *algebraic optimizations*
- **2 marks** *one adder*
- **2 marks** *one subtracter*
- **2 marks** *clock period = flop + max(ADD, SUB)*
- **1 marks** *z is combinational (minimize latency)*
- **1 marks** *z is produced before Q*
- **1 marks** *minimal number of registers*
- **1 marks** *minimal number of inputs*
- **1 mark** *inputs are registered*

Q5b (5 Marks) Analysis

Answer:

Latency to z	*2*
Minimum clock period	*flop + min(Add, Sub)*
Number of inputs	*3*
Number of registers	*5*
Number of adders	*1*
Number of subtracters	*1*
Best choice for type of state encoding	*valid bit*

Marking:

- **5 marks** *correct answer*
- **4 marks** *1 mistake*
- **3 marks** *2 mistakes*
- **2 marks** *3–4 mistakes*
- **1 marks** *5–6 mistakes*