ECE 327 Solution to Midterm

2016t2 (Spring)

All requests for re-marks must be submitted in writing to Mark Aagaard before noon on Thursday June 30.

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

		Total Marks	**Approx. Time**	**Page**
Q1	VHDL	25	15	2
Q2	Area Analysis	25	15	4
Q3	FSM	25	20	6
Q4	DFD	25	20	8
Totals		100	70	

Q1 (25 Marks) VHDL

(*estimated time: 15 minutes*)

On the first day of your dream co-op job at Amabagookify, the company goes bankrupt and you lose your job. You decide to create a startup company based on your idea for a new hardware description language, "Why Wait?", abbreviated as "yW". To get the company going quickly, you decide to base yW on VHDL, but eliminate `wait` statements and sensitivity lists, because they are too complicated.

Q1a (3 Marks) Disadvantage

What is the most significant disadvantage of eliminating wait statements?

Answer:

Timed processes in testbenches use wait statements. Without at least one timed process, we can't have a clock or other external inputs that change value.

Q1b (22 Marks) Semantics

So far, you have decided that in each simulation cycle:

- each process is executed exactly once
- the processes are executed in the order in which they appear in the architecture.

Can you define the rest of the semantics of yW such that a program with a combinational loop will have the same behaviour as in VHDL?

If **yes**, briefly describe the required semantics.

If **no**, write a sample program with a combinational loop and explain why the program would be illegal in yW or how its behaviour in yW would be different from that in VHDL.

Answer:

***Yes**, we can define semantics such that a combinational loop will have the same behaviour in yW and VHDL. The two criteria for zero-delay simulation are:*

- *How to create the illusion that processes execute in parallel within a simulation cycle.*
  yW will use projected assignments, just as VHDL does. This will give the correct behaviour for processes appearing to execute in parallel during a simulation cycle.
- *How to create the illusion that events propagate instantaneously through combinational circuitry*
  In yW, a simulation cycle will be an infinitesimally small amount of time, just like VHDL.

In addition, we need to address how to end a simulation round. Simulation cycles will continue within a simulation round until no signal's visible value changes. At this point, time will advance.

The remainder of the answer is not part of the expected answer.

Within a simulation cycle, all processes must have the appearance of executing in parallel. Because projected values are invisible within the current simulation cycle, the order in which we execute the processes does not matter.

In VHDL, a process with a sensitivity list is executed only if a signal in its sensitivity list changes value. In yW, processes will execute even if no signal that they depend on changes value. This will not affect the values of signals, because if none of the signals that a target signal reads changes value, then when we compute the new value for the target signal, we will see that its value also does not change.

Marking:

+4 marks	*correct description of combinational loop*
+5 marks	*correct description of ordering of process execution*
+4 marks	*processes must appear to execute in parallel*
+5 marks	*VHDL and yW can both use projected assignments*
+4 marks	*VHDL and yW can both use sequence of simulation cycles to make a simulation round*

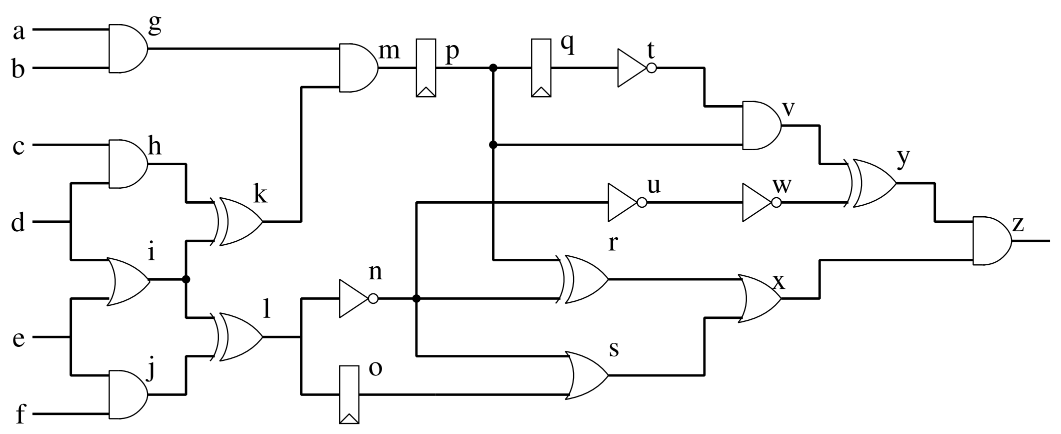
Q2 (25 Marks) Area Analysis

(*estimated time: 15 minutes*)

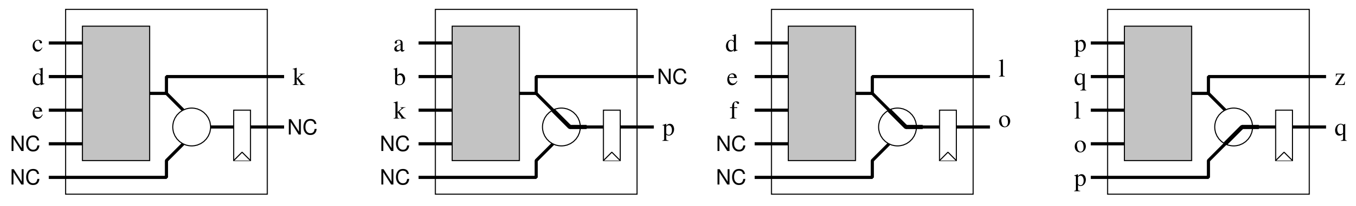
Design an FPGA implementation of the gate-level circuit shown below that uses the minimum number of FPGA cells. Use the FPGA cells on the following page to answer the question.

NOTES:

1. The primary inputs of the circuit are: `a`, `b`, `c`, `d`, `e`, and `f`.
2. The primary output of the circuit is: `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:



Marking:

Conceptual mistakes

- *missing signal o, p, q, z (1 mistake per signal)*
- *incorrect fanin for a LUT or flop*
- *missing LUT or flop (*i.e.*, a signal without a LUT or flop)*
- *not stopping at flop*
- *not sharing cell for unrelated lut and flop*
- *using a LUT as a wire*
- *missing NCs*
- *missing mux configurations*

-4 marks *1 mistake*

-8 marks *2 mistakes*

-11 marks *3 mistakes*

-14 marks *4 mistakes*

Optimality mistakes *mistakes in mapping gates to LUTs*

-2 marks *1 extra cell*

-4 marks *2 extra cells*

-5 marks *3 extra cells*

-2 marks *incorrect mapping of signal to flop vs comb*

-1 mark *labeling m (should be NC)*

-1 mark *each small mistake*

Q3 (25 Marks) FSM

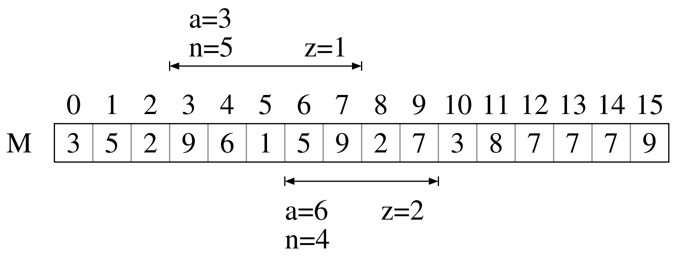
(*estimated time: 20 minutes*)

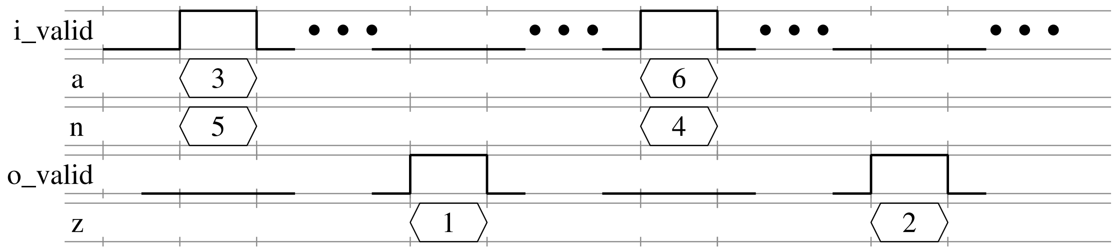
Design a state machine that finds the minimum value in a memory array.

NOTES:

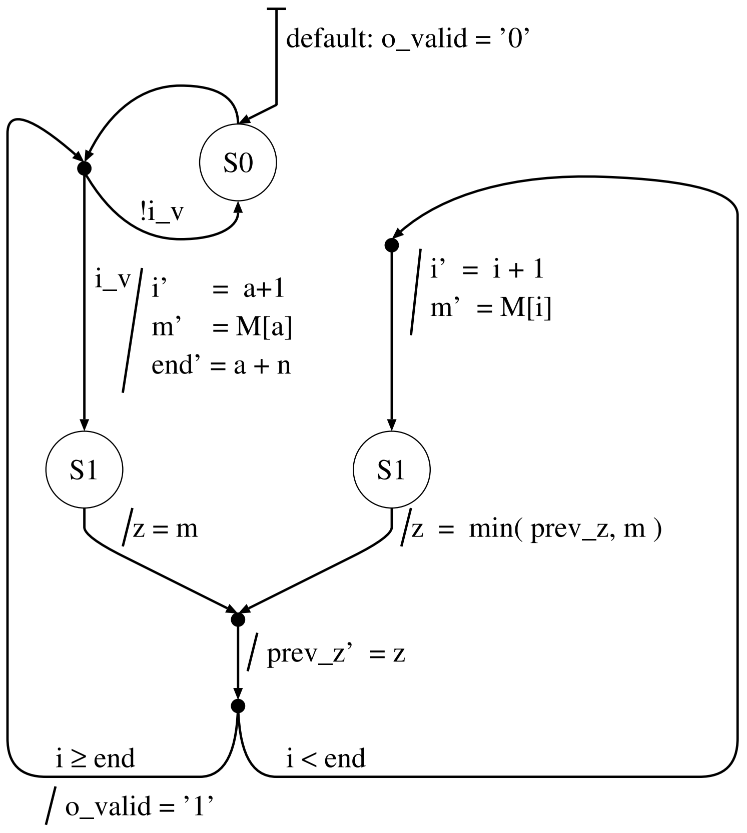
1. The inputs to the system shall be `i_valid`, `a` and `n`.
2. The outputs shall be `o_valid` and `z`.
3. The system shall have an internal *single-port* memory array, named `M`.
4. The system shall set the output `z` to be the minimum value in `M` between addresses `a` and `a+n-1`, inclusive (*i.e.*, including `M[a]` and `M[a+n-1]`, see examples below.)
5. The environment will set the inputs `a` and `n` in the same clock cycle as `i_valid='1'`.
6. The environment will set `i_valid='1'` for exactly one clock cycle, then `i_valid` will remain `'0'` until after `o_valid='1'`. An unpredictable number of clock cycles after the system sets `o_valid='1'`, the environment will set `i_valid='1'`.
7. The system shall set `o_valid='1'` for exactly one clock cycle when it sets `z` to be the correct output value.
8. The system shall ignore the possibility of overflow.

Example:





Answer:



Marking:

+2 marks	*consistent reg vs comb asns for each variable*
+2 marks	*no combinational loops*
+2 marks	*consistency and completeness of transitions*
+2 marks	*initialization*
+2 marks	*read a and n in same clock cycle as* `i_valid='1'`
+2 marks	*memory is read correctly*
+2 marks	*min value is computed correctly*
+2 marks	`o_valid='1'` *for output*
+2 marks	*first iteration (M[a]) is correct*
+2 marks	*stopping condition and last iteration is correct*
+2 marks	*works correctly when n=1*
+3 marks	*clarity and elegance*

Q4 (25 Marks) DFD

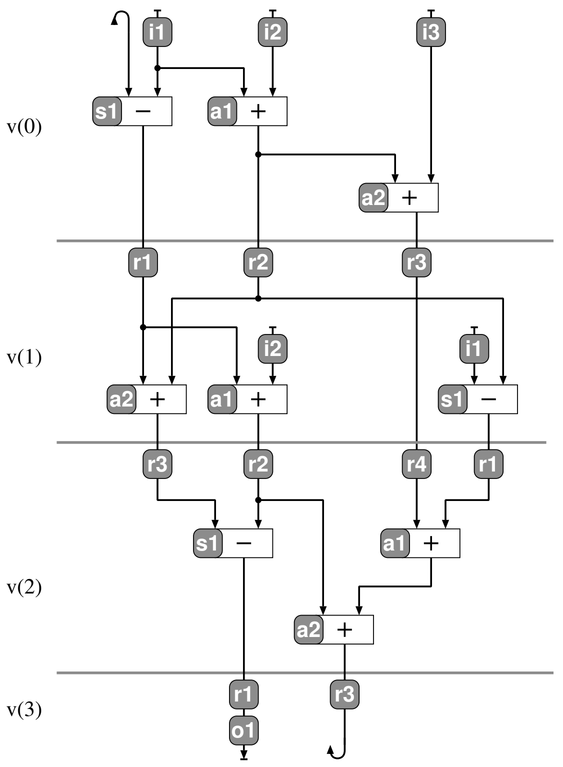
(*estimated time: 20 minutes*)

For the dataflow diagram below: perform allocation, then draw the control table.

NOTES:

1. The system shall use a parcel schedule of “unpredictable number of bubbles”.
2. You may use 2:1 multiplexers, and may combine 2:1 multiplexers to create larger multiplexers.
3. The optimization goals, in order of highest priority to lowest, are to minimize the number of:
   (a) adders and subtracters
   (b) input and output ports
   (c) registers
   (d) 2:1 multiplexers (including the 2:1 multiplexers used to build larger multiplexers)
   (e) chip-enables
4. In the control table, clearly show:
   - where *each multiplexer* and *chip-enable* is used
   - the *total* number of multiplexers and the *total* number of chip-enables used.
5. You may *not* perform scheduling optimizations on the dataflow diagram.
6. The *only* algebraic optimization that you may perform is *commutativity*.
7. You shall *not* perform don’t-care instantiation

Answer:



	a1		a2		s1		r1		r2		r3		r4		o1	
	src1	**src2**	**src1**	**src2**	**src1**	**src2**	**ce**	**d**	**ce**	**d**	**ce**	**d**	**ce**	**d**		
v(0)	i1	i2	a1	i3	r3	i1	1	s1	1	a1	1	a2	—	—	╳	
v(1)	r1	i2	r1	r2	i1	r2	1	s1	1	a1	1	a2	1	r3	╳	
v(2)	~~r4~~ r1	~~r1~~ r4	~~r2~~ a1	~~a1~~ r2	r3	r2	1	s1	1	a1	—	—	—	—	╳	
idle	—	—	—	—	—	—	—	—	—	—	0	—	—	—	╳	
const															r1	
mux	1	1	1	1	1	1										6
ce											1					1

Marking:

- **+2 marks** *3 inputs and 1 output*
- **+2 marks** *4 registers*
- **+2 marks** *2 adders and 1 subtracter*
- **+2 marks** *format of table is correct*
- **+2 marks** *3 clock cycles*
- **+2 marks** *idle state with ce='0' for interpcl var*
- **+2 marks** *correct allocation for sub.src1 in cycle 0*
- **+2 marks** *applied commutativity to addition (1 mark if not helpful)*
- **+2 marks** *don't care values*
- **+2 marks** *correct count for number of muxes and chip enables*
- **+2 marks** *optimal number of chip enables*
- **+3 marks** *optimal number of multiplexers*
- **-1 mark** *each small mistake not listed above*