University of Washington – Computer Science & Engineering

Autumn 2016 Instructor: Justin Hsia 2016-12-12

CSE 369 QUIZ 3

Name: _Perry_Perfect________________

UWNetID: _perfect______________________

Please do not turn the page until 10:30.

Instructions

- This quiz contains 4 pages, including this cover page. You may use the backs of the pages for scratch work.
- Please clearly indicate (box, circle) your final answer.
- The quiz is closed book and closed notes.
- Please silence and put away all cell phones and other mobile or noise-making devices.
- Remove all hats, headphones, and watches.
- You have 30 minutes to complete this quiz.

Advice

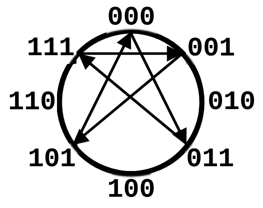
- Read questions carefully before starting. Read *all* questions first and start where you feel the most confident to maximize the use of your time.
- There may be partial credit for incomplete answers; please show your work.
- Relax. You are here to learn.

Question	Points	Score
(1) Counters	12	12
(2) Shift Registers	9	9
(3) Routing Elements	9	9
Total:	**30**	**30**

Question 1: Counters [12 pts]

Implement a counter that goes through the following state sequence: **000 → 011 → 111 → 001 → 101 →** 000 → ... using a *minimal number of 2-input logic gates.*

Fun side note – this counter creates a star on the number wheel:



PS_2	PS_1	PS_0	NS_2	NS_1	NS_0
0	0	0	0	1	1
0	0	1	1	0	1
0	1	0	X	X	X
0	1	1	1	1	1
1	0	0	X	X	X
1	0	1	0	0	0
1	1	0	X	X	X
1	1	1	0	0	1

NS_2	00	01	11	10
0	0	X	X	X
1	1	1	0	0

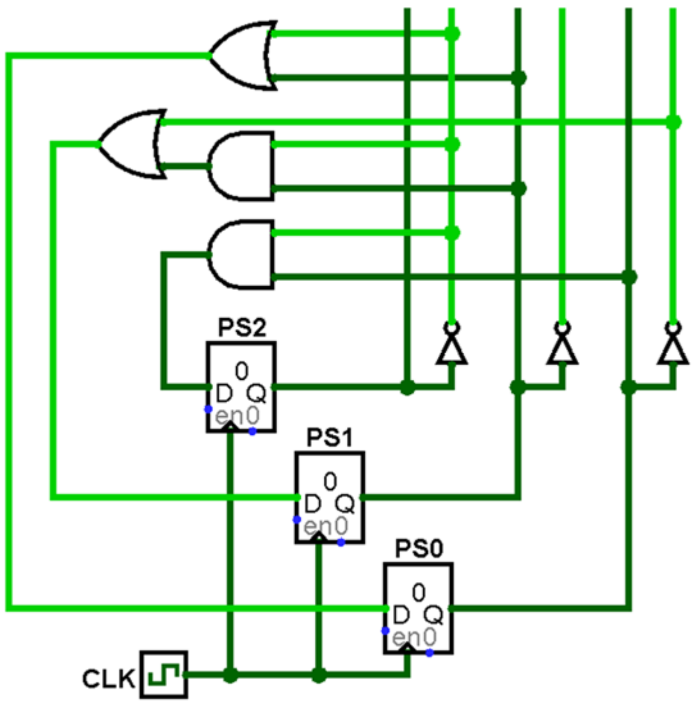
NS_1	00	01	11	10
0	1	X	X	X
1	0	1	0	0

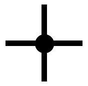
NS_0	00	01	11	10
0	1	X	X	X
1	1	1	1	0

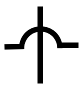
$$NS_2 = \overline{PS_2}PS_0$$
$$NS_1 = \overline{PS_2}PS_1 + \overline{PS_0}$$
$$NS_0 = \overline{PS_2} + PS_1$$

Rubric:

- 1 pt each column of truth table
- 2 pt each K-map (filling and simplification)
- 1 pt each signal convert Boolean expression to logic gates



Wire connection: 

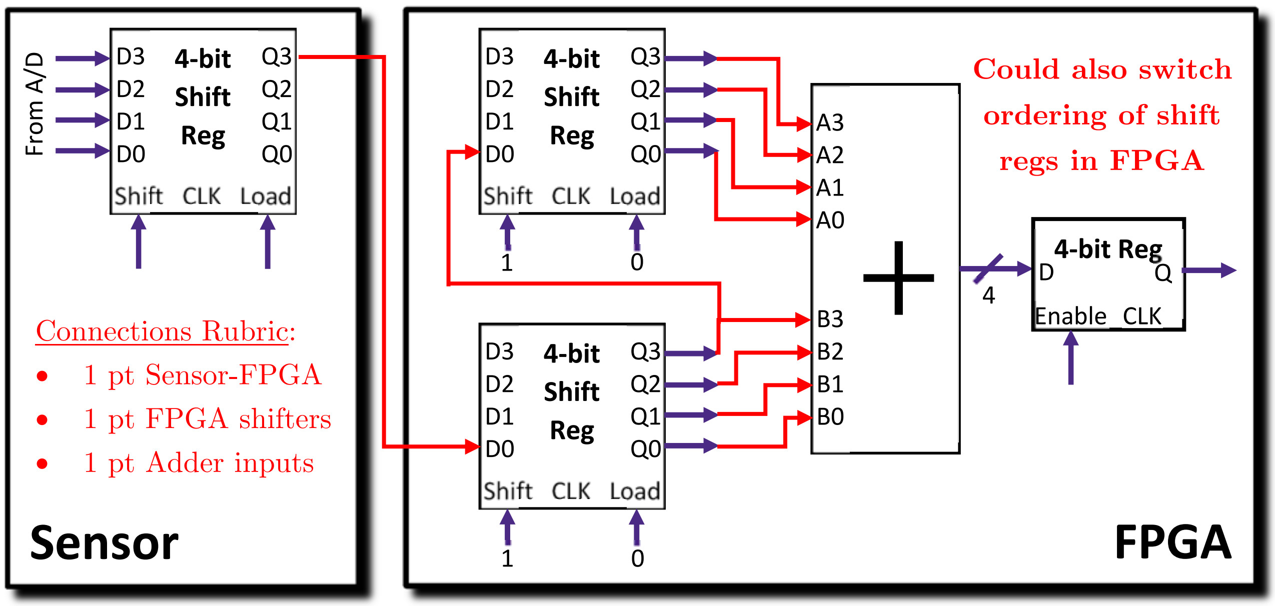
Wire crossing: 

Question 2: Shift Registers [9 pts]

We are reading **serialized** data from a sensor that includes a 4-bit analog-to-digital (A/D) converter and a shift register. We sum up **consecutive readings** of sensor data and store the result in an enabled 4-bit register. We only have access to 4-bit shift registers with parallel load capability. ***Serial loading is done from the D0 input.***

(A) Connect the circuit elements shown below to allow for the behavior described above. [3 pt]

(B) Assuming the first sensor data is available on Cycle 0, fill in the tables at the bottom for the unspecified inputs (Shift and Load of the sensor shift register and Enable to the result register) to produce the behavior described above. [6 pt]



Could also switch ordering of shift regs in FPGA

Connections Rubric:

- 1 pt Sensor-FPGA
- 1 pt FPGA shifters
- 1 pt Adder inputs

	Sensor Shift Reg		Ungraded (for your use)				Result Reg
Cycle	Shift	Load	Q0	Q1	Q2	Q3	Enable
0	0	1	X	X	X	X	0
1	1	0	I0	I1	I2	I3	0
2	1	0	X	I0	I1	I2	0
3	1	0	X	X	I0	I1	0
4	0	1	X	X	X	I0	0
5	1	0	I4	I5	I6	I7	0
6	1	0	X	I4	I5	I6	0
7	1	0	X	X	I4	I5	0
8	0	1	X	X	X	I4	0
9	1	0	I8	I9	I10	I11	1

Shift/Load Rubric:

- 2 pt: load/shift cycle pattern of 4 (half credit if 5)
- 1 pt: Load is complement of Shift

Enable Rubric:

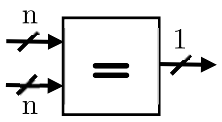
- 1 pt: 0 *after* first load/shift cycle
- 2 pt: 1 *after* second load/shift cycle (half credit if off-by-one)

Question 3: Routing Elements [9 pts]

A **cache** is a hardware device that stores the most recently accessed blocks of memory for a CPU. Implement the logic to **check for a cache hit** in a small 4-block direct-mapped cache below. *Ignore all other normal cache functionality.* As a reminder, the procedure is as follows:

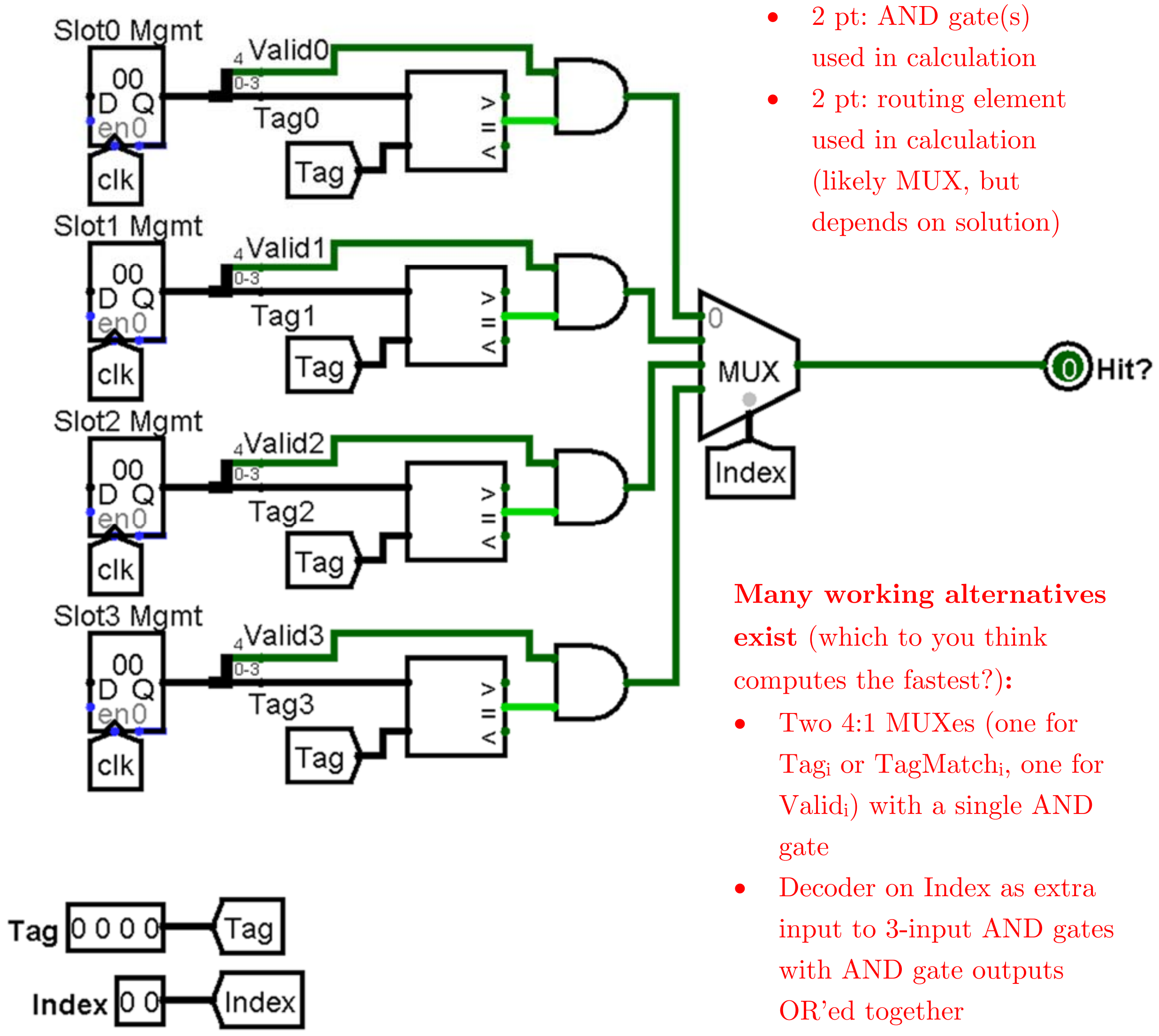
- Split the requested data's address into Tag, Index, and Offset fields (provided for you)
- The value of the Index bits (2 bits) tells you which cache line to check
- A cache hit occurs if the cache line is **Valid (1 bit) and the Tag (4 bits) matches**

We represent the cache line management bits below using registers. Assume you can freely use equality logic blocks:



Hit? Rubric:

- 2 pt: equality block used in calculation
- 2 pt: AND gate(s) used in calculation
- 2 pt: routing element used in calculation (likely MUX, but depends on solution)



Many working alternatives exist (which to you think computes the fastest?)**:**

- Two 4:1 MUXes (one for Tag_i or TagMatch_i, one for Valid_i) with a single AND gate
- Decoder on Index as extra input to 3-input AND gates with AND gate outputs OR'ed together