ECE/CS 552: Introduction to Computer Architecture
ASSIGNMENT #5
Due Date: At the beginning of lecture, December 1st, 2010

This homework is to be done individually. Total 5 Questions, 100 points

1. (10 pts.) Consider a computer memory system with the following properties:
40-bit virtual byte address
16 KB pages
36-bit physical byte address
1) (5 pts.) What is the total size of a single-level forward page table for each process on this processor, assuming that the valid, protection, dirty, and use bits take a total of 4 bits and that all the virtual pages are in use? (Assume that disk addresses are not stored in the page table).
2) (3 pts.) If the L1 cache is physically addressed and a TLB is used for fast virtual-to-physical translation, show the required operations of a data memory read access that hits in L1 cache.
3) (2 pts.) To maintain single-cycle access latency on L1 hits, sometimes L1 caches are virtually addressed. What must be done for a virtually addressed L1 cache during a context switch? Describe at least one way that guarantees the processes to access virtual addresses of their own address spaces.

Solution:
1) The total size is equal to the number of entries times the size of each entry. Each page is 16 KB, and thus, 14 bits of the virtual and physical address will be used as a page offset. The remaining 40 – 14 = 26 bits of the virtual address constitute the virtual page number, and there are thus 2^{26} entries in the page table, one for each virtual page number. Each entry requires 36 – 14 = 22 bits to store the physical page number and an additional 4 bits for the valid, protection, dirty, and use bits. We round the 26 bits up to a full word per entry, so this gives us a total size of $2^{26} \times 32$ bits or 256 MB.
2) i. Access the TLB
ii-a. If TLB hits, fetch the physical address from the TLB and access L1 cache
ii-b. If TLB misses, access the page table in the memory (location of page table is stored in page table base register)
Update the TLB, and then retry the instruction.
3) To prevent a process from accessing data that belongs to another process, either set all the cache lines as invalid during a context switch, or use pid (process id) to distinguish virtual address for different processes.

2. (10 pts.) A program repeatedly performs the following 3-step process:
It reads in a 4KB block of data from disk, does some processing on that data, and then writes out the result as a 4KB block elsewhere in the disk. Each block is continuous and randomly located on a single track on the disk (assume that different blocks are on different tracks). The disk drive rotates at 10,000 RPM, has an average seek time of 8ms, and has a transfer rate of 50MB/sec. The controller overhead is 2ms. No other program is using the disk or processor, and there is no overlapping of disk operation with processing. The processing step take 20 million clock cycles and the clock rate is 5GHz. What is the overall speed of the system in blocks processed per second?

Solution:

(Seek time + Rotational delay + Overhead) * 2 + Processing time

(0.008 sec + 0.5 / (10000/60) sec + 0.002) * 2 + (20 million cycles)(5 GHz) sec = (.008 + .003 + .002)*2 + .004 = 30 ms

Block processed/second = 1/30 ms = 33.3

Transfer time is 80 μsec and thus is negligible.

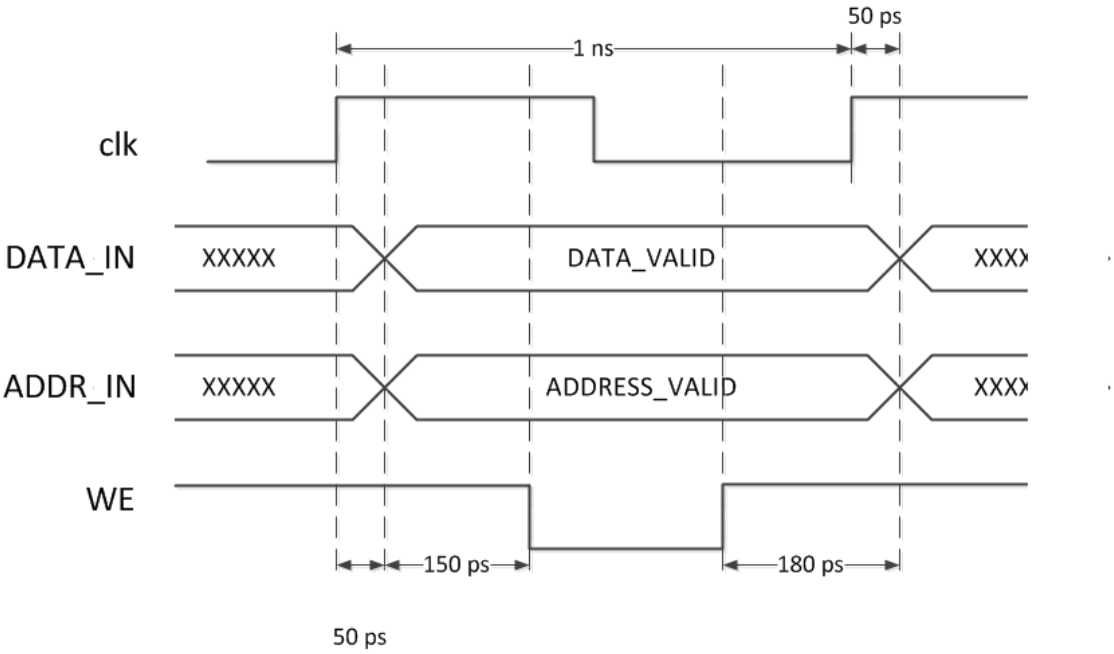
3. (10 pts.) In the following table, do the multiplication in 2-bit Booth's encoding. The multiplicand and multiplier are in 7-bit 2's complement representation. Show all the partial products, and show the final result of the computation.

Multiplicand					0	0	1	1	0	1	1		
Multiplier					1	1	0	1	0	0	1		
Partial Products	Encoded multiplier:							0	-1	-2	1		
					0	0	1	1	0	1	1		
			1	1	0	0	1	0	0	_	_	_	
								+1	_	_	_		
			1	0	0	1	0	0	_	_	_	_	
							+1		_	_	_	_	
Result	…1	1	0	1	1	0	0	1	0	0	1	1	
										(0xFD93)			

4. (25 pts.) Phased Memory Design

Timing constraints exist in real-world memories. Assume we have a simple latch-based memory. WE is the write enable signal. When WE is 0, DATA_IN is written into MEM [ADDR_IN]. When WE is 1, MEM [ADDR_IN] is read on DATA_OUT.

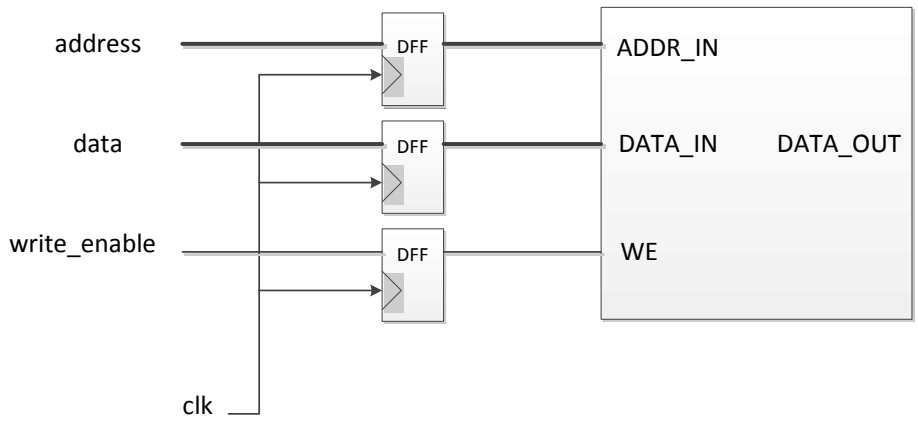
Below is the timing waveform for the memory module. The clock signal clk has a period of 1 ns.



Timing constraints for the memory module and the DFF are:

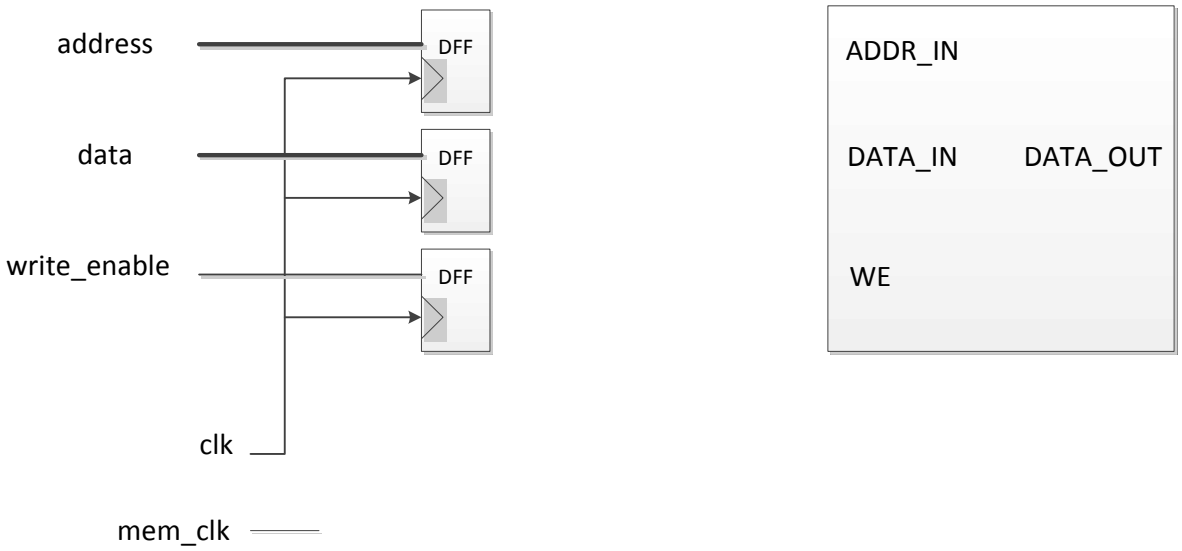
a. DFF propagation delay is 50 ps.
b. Setup times for ADDR_IN and DATA_IN are both 150 ps. That is, ADDR_IN and DATA_IN must be stable for at least 150 ps before WE changes from 1 to 0.
c. Hold times for ADDR_IN and DATA_IN are both 180 ps. That is, ADDR_IN and DATA_IN must be stable for at least 180 ps before WE changes from 0 to 1. Violating setup and hold time constraints may result in wrong data being written into wrong location.

1) (5 pts.) Explain why the following “straightforward” design does not work.



Answer: Because the WE signal is asserted immediately after the address and data become valid, and unasserted immediately after the address and data become invalid. It violates both the setup and hold time constraints of the memory.

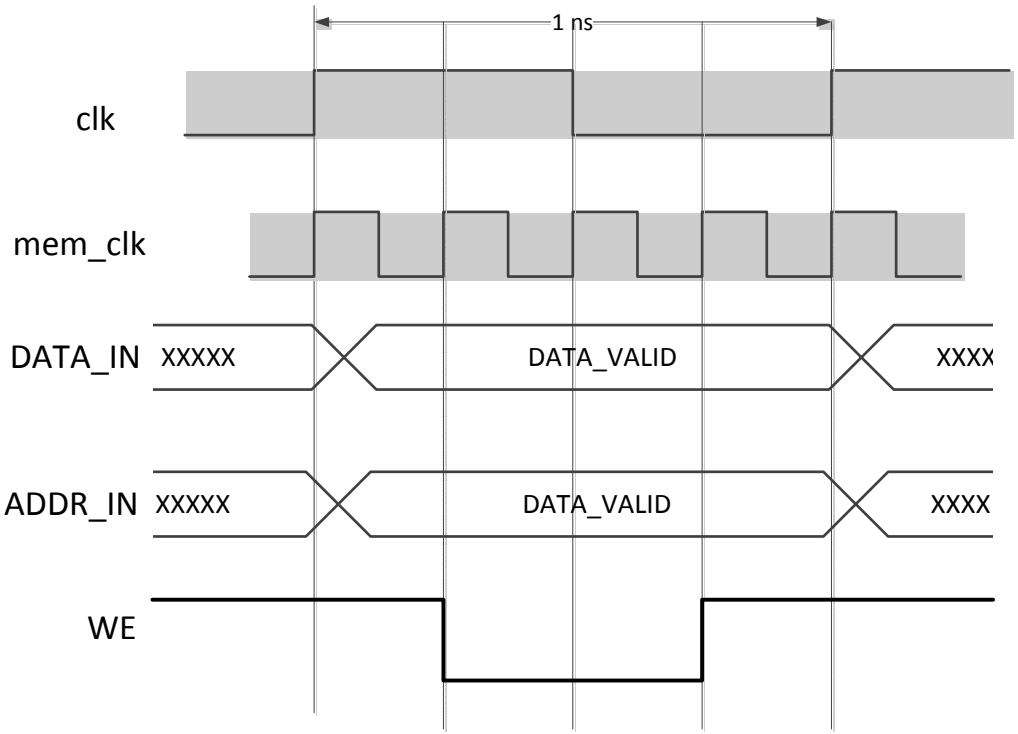
2) (15 pts.) Assume we have another clock signal mem_clk that has a period of 250 ps, which is 4 times as fast as clk. By using mem_clk, we are able to divide the clk into four phases, and generate different control signal in each phase. Draw peripheral control logic in the following diagram to generate correct WE signal for the memory when it is being written. You may use logic gates and flip-flops. Indicate the initial value in the sequential component, if necessary.



This problem is not well defined since there is no information about the gate delay. Any design that satisfied the timing constraints in the first waveform figure is acceptable.

3) (5 pts.) Draw the new waveforms for the input signals when the memory is being written in one clock cycle. Prove that your design did not violate the timing constraints.

One example:

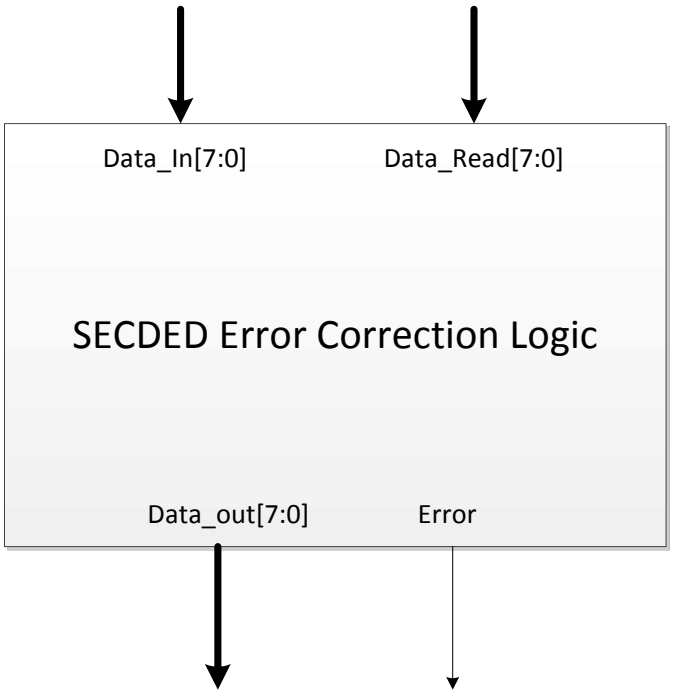


5. (45 pts.) Design in Quartus: SECDED Logic Design.
Design a single-error-correct-dual-error-detect (SECDED) error correction logic in Quartus II. The protected data width is 8 bits. The width of ECC is 5 bits: 4 check bits c1 – c4 and 1 overall parity bit p. The memory should be able to correct single-bit errors at any position (including the ECC bits), and be able to detect double-bit errors (but cannot correct them). The parity check matrix for the ECC (m=8, k=4) is:

Bit index	0001 1	0010 2	0011 3	0100 4	0101 5	0110 6	0111 7	1000 8	1001 9	1010 10	1011 11	1100 12	1101 13
Bit name	c_1	c_2	d_1	c_4	d_2	d_3	d_4	c_8	d_5	d_6	d_7	d_8	p
C_1	X		X		X		X		X		X		
C_2		X	X			X	X			X	X		
C_4				X	X	X	X					X	
C_8								X	X	X	X	X	
P	X	X	X	X	X	X	X	X	X	X	X	X	

1) (15 pts.) Design the logic of ECC generator using only logic gates. The ECC should take the 8-bit data and compute its 4-bit check bits and the parity bit.

2) (30 pts.) Design the SECDED error correction logic. It has two input: Data_In [7:0], Data_Read [7:0], and two outputs: Data_Out [7:0] and Error. Data_In [7:0] represents the actual data that writes to a memory location, and Data_Read [7:0] represents the value that read from the same memory location, which may or may not have errors in it. The number of error bits is the number of different bits between Data_In [7:0] and Data_Read [7:0].



The Data_Out[7:0] output should return the correct data when there is no error, and when there is a correctable single-bit error (so your design must correct the corrupted bit).

The Error output should be zero when there is no error, and should be one only when an uncorrectable double-bit error between is detected.

Add functional blocks and proper connections to complete this diagram. Besides logic gates, you are allowed to use the following functional blocks:

a. ECC generator that you designed in (1).

b. Bit-wise XOR module for two multi-bit values.

c. Decoder with any input width. You are most likely to use 4-to-16 decoder.

You should turn in:

1) Schematics of the ECC generator and the SECDED logic.

2) Simulation result of the following input sequence (you can add more).

Error type	Data_In[7:0]	Data_Read[7:0]
No error	00000000	00000000
	11111111	11111111
	10101010	10101010
1-bit error	00000000	00000001
	11111111	01111111
	10101010	10111010
2-bit error	00000000	10000001
	11111111	11110011
	10101010	10110010

Output should be identical with Data_In when there is no error or 1-bit error. When there is 2 bit error, the “error” signal should be high.