EECS150: Components and Design Techniques for Digital Systems

University of California
Dept. of Electrical Engineering and Computer Sciences

Mid Term 3
Fall 2004

Last name: __________________________  First name_______________________
Student ID: __________________________  Login: ______________________
Lab meeting time: ________________  TA's name: ____________________

(Sorry to ask this next question, but with 100 students packed closely together there may be a wide range of behavior.)
Student to my left is ________________________   __________________
Student to my right is ________________________   __________________

No notes. No calculators! This booklet contains 10 numbered pages, including room to show your work. Please, no extra stray pieces of paper. The exam contains 5 substantive questions and 100 points (plus 15 possible extra credit points), so just over 1/2 a point per minute. Read through the questions before you start. You have 3 hours, so relax, work thoughtfully and give clear answers. Good luck!

I certify that my answers to this exam are my own work. If I am taking this exam early, I certify that I shall not discuss the exam questions, the exam answers, or the content of the exam with anyone until after the scheduled exam time. If I am taking this exam in scheduled time, I certify that I have not discussed the exam with anyone who took it early.

Signature: ______________________________________

BE SURE YOUR NAME IS ON EVERY PAGE
Problem 1 (20pts). Hamming Error Correction Codes

As you should recall from lecture, Hamming Single Error Correction (SEC [7, 4]) codes on 4-bit data words use 3 parity bits. They correct a single error in any one of the 7 code word bits.

Hamming code word: \[ p_1 \ p_2 \ d_1 \ p_3 \ d_2 \ d_3 \ d_4 \]
Position: \[ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \]
- \( p_1 \) covers positions 1, 3, 5, 7 (group 1)
- \( p_2 \) covers positions 2, 3, 6, 7 (group 2)
- \( p_3 \) covers positions 4, 5, 6, 7 (group 3)

Parity bits are assigned to force **even parity** over their respective groups. To correct a single error in an Hamming code, the check bits \((c_3 \ c_2 \ c_1)\) are generated by finding the parity of the corresponding groups. The check bits \((c_3 \ c_2 \ c_1)\) give the position of the error in the 7-bit code word.

Design a combination logic circuit to perform **single error correction** on a 7-bit Hamming code word. You should produce a schematic composed any any standard combinational logic blocks (gates, comparators, adders, etc) you need.

![Diagram of combination logic circuit for single error correction in Hamming code word]
Problem 2 (20pts). Timing and Delays

The following diagram shows a simple 2-bit ripple adder with full-adder cells constructed from 2 half-adder cells (the entire circuit has been provided for your convenience):
You are to draw the precise timing of the output of the 2-bit ripple adder shown above. You are to assume that each gate has a propagation delay of one time unit (this is called the unit delay model). In the timing diagram below, each dotted line signifies one time unit.

Notice that all outputs start at “Don’t Know” or \( 1'bX \) and that additional space has been provided so that you may draw the waveforms of intermediate nodes, if you choose to.
**Problem 3 (20pts).** Instruction Set Architecture & RTL

Using the datapath from class (shown below) you will implement the following operation:

\[
\text{sw rs rt offset} \quad \text{memory[rt + offset] = [rs]}
\]

The memory and the register file both have asynchronous read and synchronous write. IE: reads are instantaneous whereas writes are completed on clock edge.

a. 10pts: Write out the RTL for completing this operation. Once you have all the required operations you will need to arrange them into four cycles.

b. 10pts: Fill in the table below with 0, 1 or X for all the control signals. For ALUOp, use the values (2'b00: A + B, 2'b01: A – B, 2'b10: A + 1, 2'b11: A -1)

<table>
<thead>
<tr>
<th>Cycle</th>
<th>MRLoad</th>
<th>IRLoad</th>
<th>RegMemEn</th>
<th>MemWrite</th>
<th>MASel</th>
<th>PCreset</th>
<th>PCLoad</th>
<th>MemEn</th>
<th>MemWrite</th>
<th>MASel</th>
<th>MemEn</th>
<th>MemWrite</th>
<th>ALUOp</th>
<th>SRCB</th>
<th>SICA</th>
<th>RegWrite</th>
<th>SICRD</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Problem 4 (20pts+5EC). Multiplier

The block diagram below shows a datapath for an iterative unsigned 12-bit multiplier.

It operates by first loading the 12-bit operands in registers A and B and clearing P. In each iteration, the partial product is accumulated into P and a portion of B. When the operation is done, the entire B operand has been shifted out and the 24-bit product is in registers P and B.

In class, we discussed a 1-bit per step iterative multiplier, so 12 iterations would be required to complete the full 12-bit multiply. We also discussed an array multiplier, which formed all the partial products and accumulated them in a large combinational array of full adders, requiring only 1 cycle.

Here we are going to design a hybrid adder data path that multiplies A by 3 bits of B in each step, so four iterations are required for the full 12-bit multiply. We’ll do it in steps.

Notice that for this problem we are not going to design the controller.

a. 5pts: Replace the “?” marks with the correct Verilog bit selections for the associated data connections.
b. 15pts: Show below how the adder array is implemented using Full Adders (FA). Be sure the carries are handled correctly. Show which bits go to $P$ and which go to $B$.

c. **EXTRA CREDIT (5pts):** Show, below, an implementation of $B$ and $P$ using D-FlipFlops and MUXes.
Problem 5 (20pts+10EC). Controller and Datapath Design

In this problem you will be designing the read controller for a CAM or Contents Addressable Memory: essentially a look-up-table; a memory which stores <Key, Value> pairs. An AddEntry operation consists of giving the CAM a <Key, Value> pair which it will then store (along with a Valid bit). A later read given the same <Key> will return the original <Value>. Shown below is a small example of the operation of a CAM.

- Reset, CAM is empty
  - All Valid bits are 1'b0
  - SRAM Handles this (Not your controller)
- Read <0x0000>, Finishes but fails
- AddEntry <0x0000, 0x0001> Succeeds
- Read <0x0000>, Succeeds returning <0x0001>
- AddEntry <0xF0A3, 0xB720> Succeeds
- AddEntry <0x0000, 0x1234> Succeeds (Overwrites <0x0000, 0x0001>)
- Read <0xF0A3>, Succeeds returning <0xB720>
- Read <0x0000>, Succeeds returning <0x1234>

You will be implementing the read portion of this CAM using an SRAM with asynchronous read and synchronous write. That means it will return the data infinitely quickly (not a cycle later), but writes are not effective until the edge of the clock. Shown below is the format of the 33 bit entries in your SRAM consisting of a Valid bit, a 16 bit Key and a 16 bit Value.

```
| Valid | Key | Value |
```

Steps for a Read Operation (Which you must implement)
There is no correspondence between steps and clock cycles
1. Start from Address 0x00
2. Check the <Valid, Key, Value> at the current Address
   - Succeed if this entry is Valid and the Key matches
   - Otherwise if we are at Address 0xFF then Fail
3. Increment the Address
4. Return to step 2

Steps for an AddEntry Operation (EXTRA CREDIT)
There is no correspondence between steps and clock cycles
1. Start from Address 0x00
2. Check the <Valid, Key, Value> at the current Address
   - If this entry is Valid and the Key matches, change the Value stored
   - If this entry is not Valid, then store the new <Key, Value> pair
   - Otherwise if we are at Address 0xFF then Fail
3. Increment the Address
4. Return to step 2
a. 16pts: Shown below is an empty block diagram for the datapath of your CAM. Fill it in with any necessary components. If you would like, feel free to use Verilog or logic equations to complement your schematic if necessary. You only need to implement the Read operation, you may assume the SRAM is preloaded with the necessary values.

**EXTRA CREDIT (10pts):** Implement the AddEntry operation in addition to the read operation.

There is a timing diagram on the next page! Please refer to it!
Shown below is a simple timing diagram for:

- AddEntry<0xF0A3, 0xB720>  Succeeds
- Read <0xF0A3>, Succeeds returning <0xB720>

Notice that the number of cycles per operation is completely unknown and dependant on your implementation.

b. 2pts: Explain BRIEFLY how you would modify this CAM to make it more efficient, by turning it into something like a Hash Table. Hint: you will be using the Key to make the search for the correct <Valid, Key, Value> entry faster.

c. 2pts: Explain BRIEFLY one or two difficulties in adding a Delete operation. Notice that adding Delete is extremely difficult and you need not go into detail about all the problems involved.