CSE140 – Spring2013

CSE140 Homework #7 - Solution

You must *SHOW ALL STEPS* for obtaining the solution. Reporting the correct answer, without showing the work performed at each step will result in getting 0 points for that problem.

1. Adder:
You are asked to design an N-bit adder. Suppose that the only elements you can use are in library *lib3* with 2-input NAND, 2-input NOR and 1-input INVERTER. Each of them has a propagation delay of 1ps, and a 1-bit full adder has 6ps maximum propagation delay.

(a) Briefly explain why the delay of 1-bit full adder is 6X that of a combinational gate in the given library. Assume the XOR gate is implemented using 4 NAND gates as shown below.

An XOR can be implemented by three stages of 2-input NAND gates. Therefore the total delay is roughly six times of that of a 2-input NAND gate.

(b) If you are asked to design a 1-bit full adder using only 2-input NAND gates, how many gates would you like to budget in order to minimize the cost? Draw the circuit diagram to illustrate.

We can implement a 1-bit full adder using 9 2-input NAND gates. The circuit diagram is as below.

(c) Calculate the total delay of a 16-bit ripple-carry adder based on the given library timing
(d) Design a 16-bit carry-lookahead adder using blocks of size \( k=4 \) and calculate its total delay.

You can assume a CLA with basic 1-bit adder as below. The computation of negative generate \((G')\) and propagate \((P)\) do not require additional logic. Totally there are just two XOR gates in series, which add up to 8 NAND gates.

A 4-bit CLA block can be designed in the following way using gates from only lib3. Totally we need 4 INVs and 11 NANDs for one block.

As a result, we can compute the delay as follows.

\[
t_{pg}=2\text{ps}, \quad t_{pg\text{ block}}=6\text{ps}, \quad t_{AND\text{ OR}}=2\text{ps}, \quad t_{FA}=6\text{ps}
\]

\[
t_{CLA}=t_{pg}+t_{pg\text{ block}}+(16/4-1)t_{AND\text{ OR}}+4t_{FA}=1+6+3*2+4*6=37\text{ps}
\]

(e) If the operational frequency is not a major concern, but achieving minimum cost in terms of the number of transistors is, would you prefer to use a ripple-carry adder, or a carry-lookahead? How much cost, in terms of number of transistors, could you save for an 16-bit adder using the lower-cost architecture? Assume each block of the carry-lookahead adder contains 4 bits.

Ripple-carry adder will be the better choice as it is more cost efficient.

A 16-bit ripple-carry adder, implemented using 9 NAND gates, will use \(16*9*4=576\) transistors.

A 16-bit carry-lookahead adder, implemented as (d), will use 4 INVs and 11 NANDS or NORs in one 4-bit block, which requires 52 transistors per block thus 13 transistors per bit. The 16-bit adder will consume \(16*(8*4+13)=720\) transistors. Also, if you consider the \(C_{out}\) computation in each 1-bit adder, you need another NAND as shown in 1(b). So it may adds up to \(16*(9*4+13)=784\) transistors.

2. ALU:
(a) Design an ALU with similar functionality as the one shown in table 5.1 to implement $A + B$, $A - B$, NAND, NOR, AND, OR, EQUAL and NOT EQUAL using three control bits. Give a table to illustrate how you would map each function to some control pattern. Draw your ALU design with two 1-bit inputs and one 1-bit output based on the table you just generated.

<table>
<thead>
<tr>
<th>control</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>A NEQ B</td>
</tr>
<tr>
<td>001</td>
<td>$A + B$</td>
</tr>
<tr>
<td>010</td>
<td>$A \text{ AND} B$</td>
</tr>
<tr>
<td>011</td>
<td>$A \text{ OR} B$</td>
</tr>
<tr>
<td>100</td>
<td>$A \text{ EQ} B$</td>
</tr>
<tr>
<td>101</td>
<td>$A - B$</td>
</tr>
<tr>
<td>110</td>
<td>$A \text{ NOR} B$</td>
</tr>
<tr>
<td>111</td>
<td>$A \text{ NAND} B$</td>
</tr>
</tbody>
</table>

(b) Fabricate your single-bit ALU design into a block with clearly defined interfaces (signal inputs and outputs). Assemble a two-bit ALU at block level based on the block you just fabricated.

3. Multiplier:
You are given the architecture of the NxN multiplier shown in Figure 5.18(c) on page 252 of the textbook, suppose a 2-AND gate has 2ps delay, while a full adder has 3ps delay.

(a) Compute the propagation delay of the entire multiplier.

$$8t_{FA} + 1t_{AND} = 26\text{ps}$$
(b) How would you modify the architecture to enable multiplication of negative numbers? Suppose that you are given a negative multiplicand and a positive multiplier in two's complement formats, briefly discuss how you should modify the architecture.

Sign bit extension during each level of addition.

4. Counter design:

(a) Design a 3-bit counter that counts on each clock tick only odd numbers from $1$ up to 7 ($1, 3, 5, 7, 1, 3, \ldots$). Once it reaches 7, it sets a ripple carry out bit to 1. Otherwise ripple carry out bit is zero. It also has an enable and set inputs. When set = 1, the counter value is set to 1. The counter counts only when enable = 1, otherwise it holds its current value.

(b) Fabricate your design in part (a) into a block with interface definition. Design a 6-bit odd number counter that counts $1, 3, 5, 7, 9, 11, \ldots, 63, 1, 3, 5, \ldots, 63, 1, \ldots$ at block level based on the block you just fabricated.
5. RTL design:

Exercise 5.16 from page 288 of Vahid's book.

Using the RTL design method shown in Table 5.1 to create an RTL design that outputs the maximum value found in a register file \( A \) consisting of 64 32-bit numbers.

**Step 1 - Capture a high-level state machine**

*Inputs:* go (bit), \( A_{\text{data}} \) (32 bits)

*Output:* \( A_{\text{addr}} \) (6 bits), max (32 bits)

*Local Registers:* \( \text{max}_{\text{tmp}} \) (32 bits), index (6 bits)

**Step 2 - Create a datapath**
Step 3 - Connect the datapath to a controller

Step 4 - Derive the controller’s FSM

Inputs: go, index_Lt_64, data_gt_max (bit)

Outputs: A_addr_ld, index_clr, index_ld, max_tmp_clr, max_tmp_ld, max_ld (bit)