Summer 2011 CSE 140L
Lab 2 Assignment

Alex Orailoglu
06/29/11
Lab 2

Part 1: 2’s complement Multiplication
- Build an 8-bit two's complement multiplier using Robertson’s algorithm.

Part 2: Error correction and detection
- Implement a logic to detect a single bit error in a 7 bit received message and correct it.
2’s Complement Multiplication

Basic Idea
- Accumulate a partial sum in multiple steps.
- Only the rightmost bit of the multiplier is checked on every step. If it is 1, the multiplicand is added to the partial sum.
- Shift the partial sum and the multiplier one bit to the right on every step.
  - Make the partial sum line up with the multiplicand;
  - Always check the rightmost bit of multiplier, no need to check higher bits;
  - Lower bits of the partial sum shifted into the multiplier register for storage.
- Possible correction step at the end for negative numbers.
**Multiplication Example**

\[ 1010_2 \times 1100_2 \]

### Step 1:

- ** Accumulator:** 0000
- ** Multiplier:** 1100
- ** Multiplicand:** 1010

*Right shift*

### Step 2:

- ** Accumulator:** 0000
- ** Multiplier:** 0110
- ** Multiplicand:** 1010

*Right shift*

### Step 3:

- ** Accumulator:** 0000
- ** Multiplier:** 0011
- ** Multiplicand:** 1010

*Accumulator = Accumulator + Multiplicand*

- ** Accumulator:** 1010
- ** Multiplier:** 0011
- ** Multiplicand:** 1010

*Right shift*

**Correction Step:**

- ** Accumulator:** 1101
- ** Multiplier:** 0001
- ** Multiplicand:** 1010

*Accumulator = Accumulator - Multiplicand*

- ** Accumulator:** 0011
- ** Multiplier:** 0001
- ** Multiplicand:** 1010

**Result:** 0011000
• The most significant bit of the accumulator during a shift
  – Once the accumulator is non-zero, you need to shift in the sign of the multiplicand.

• The conversion of an adder into an adder/subtractor.
  – Add if $A/S = 0$; subtract if $A/S = 1$.

• The $Mult$ input of the controller
  – You need to drive it **high** if and only if Robertson's algorithm requires an operation.
Multiplier Components

- **8-bit Shift Register:**
  - When $LD$ is set, the parallel data input is loaded into the shift register;
  - When $SRt$ is set, the content of the register is shifted 1-bit right, with the most significant bit fed by $VIin$.

- **8-bit Adder:**
  - Output $Q(7:0) = A(7:0) + B(7:0) + CI$;
  - You should convert it into an adder/subtractor with an input A/S.
    - If $A/S=0$, $Q(7:0) = A(7:0) + B(7:0)$.
    - If $A/S=1$, $Q(7:0) = A(7:0) - B(7:0)$.

- **Control Unit:**
  - Handles the multi-cycle process by telling each device what to do on every cycle until the entire process is finished.
  - Every other device simply does what it’s told on each cycle.

- **Your job:** add necessary logic & connections.
Multiplication Process

Robertson’s Multiplication Algorithm:
• Load multiplier and multiplicand, clear accumulator A and flip-flop F;
• Repeat 7 times the following two sub-steps (a & b):
  a) If the rightmost bit of the multiplier is 1 then A = A + Multiplicand;
  b) Shift A and multiplier 1-bit right;
• Correction step: if the sign bit of the multiplier is 1 then A = A - Multiplicand;
• The 15-bit result is in A(7:0)multiplier(7:1).

Controller outputs at each cycle:

<table>
<thead>
<tr>
<th>Cycle</th>
<th>Init</th>
<th>LoadA</th>
<th>Shift</th>
<th>A/S</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1,3,5,7,9,11,13</td>
<td>0</td>
<td>1/0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2,4,6,8,10,12,14</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>1/0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Error detection and correction

- Given 7 bits binary number (4 bit data and 3 parity bits), implement a logic to correct a **single bit error** (1 may have become a 0 and conversely 0 a 1) in the 7 bit number.
- Use only XNOR gates (and one 3-8 Decoder to be found in LogicWorks library)
- You should be able to complete the design well within 20 2-1 input XNOR gates (and a decoder).
Error detection and correction - implementation

- Your logic should use the 7 bits binary message to generate a 3-bit number, which when non-zero indicates an error at that bit position of the message.
- Using the 3-to-8 decoder provided to you in the library, convert this 3 bit binary number to individual 8 lines.
  - One of the lines denotes the absence of an error.
  - The other 7 lines correspond to the 7 possible fault locations.
- Implement a logic to use those 8 lines from the decoder to correct the error in the received message.