ELEC6234
Embedded Hardware Design

Embedded hardware multipliers
Unsigned binary multiplication

\[
\begin{array}{cccc}
A_3 & A_2 & A_1 & A_0 \\
X & B_3 & B_2 & B_1 & B_0 \\
\hline
A_3B_0 & A_2B_0 & A_1B_0 & A_0B_0 \\
A_3B_1 & A_2B_1 & A_1B_1 & A_0B_1 \\
A_3B_2 & A_2B_2 & A_1B_2 & A_0B_2 \\
A_3B_3 & A_2B_3 & A_1B_3 & A_0B_3 \\
\end{array}
\]

2N-bit product

Hardware implementations:
1. Sequential – adder, shift register, state machine
2. Combinational – adders and AND gates for partial products \( A B_i \)
Example

Multiply $7 \times 5$ using unsigned 3-bit binary representation

\[
\begin{array}{c}
111 & \text{M multiplicand} \\
\times 101 & \text{Q multiplier} \\
111 & (\text{ADD 7, } Q_0 = 1) \\
0000 & (\text{don’t ADD, } Q_1 = 0) \\
111 & (\text{ADD 7 shifted by 2 bits, } Q_2 = 1) \\
\end{array}
\]

\[100011 \quad (\text{result } = 35)\]
Same example cycle by cycle for sequential hardware implementation

<table>
<thead>
<tr>
<th>C</th>
<th>A</th>
<th>Q = 5</th>
<th>Initial Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0 0</td>
<td>1 0 1</td>
<td>Initial Values</td>
</tr>
<tr>
<td>0</td>
<td>1 1 1</td>
<td>1 0 1</td>
<td>ADD: A := A+M</td>
</tr>
<tr>
<td>0</td>
<td>0 1 1</td>
<td>1 1 0</td>
<td>SHIFT (cycle 1)</td>
</tr>
<tr>
<td>0</td>
<td>0 0 1</td>
<td>1 1 1</td>
<td>SHIFT (cycle 2)</td>
</tr>
<tr>
<td>1</td>
<td>0 0 0</td>
<td>1 1 1</td>
<td>ADD: A:=A+M</td>
</tr>
<tr>
<td>0</td>
<td>1 0 0</td>
<td>0 1 1</td>
<td>SHIFT (cycle 3)</td>
</tr>
</tbody>
</table>

A,Q= 0 1 0 0 0 1 1 = 35

Note: here the double length accumulator AQ shares storage with multiplier Q
Sequential unsigned multiplier - hardware implementation

C, A and Q can right shift by one bit
Combinational unsigned multiplier

<table>
<thead>
<tr>
<th></th>
<th>X3</th>
<th>X2</th>
<th>X1</th>
<th>X0</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Y3</td>
<td>Y2</td>
<td>Y1</td>
<td>Y0</td>
</tr>
<tr>
<td>+</td>
<td>X3Y0</td>
<td>X2Y0</td>
<td>X1Y0</td>
<td>X0Y0</td>
</tr>
<tr>
<td>+</td>
<td>X3Y1</td>
<td>X2Y1</td>
<td>X1Y1</td>
<td>X0Y1</td>
</tr>
<tr>
<td>+</td>
<td>X3Y1</td>
<td>X2Y1</td>
<td>X1Y1</td>
<td>X0Y1</td>
</tr>
</tbody>
</table>

Propagation delay:
~ 2N adders
Signed multiplication

- Can extend addition to 2n bits
- Negative multiplicand, positive multiplier
  - n cycles

\[
\begin{array}{cccccccccccc}
-5 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\
5  & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\
\hline
1  & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\
0  & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1  & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0  & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
-25 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
\hline
-128 & -128 & +64 & +32 & +4 & +2 & +1 \\
+103  & & & & & & & & & & & \\
\end{array}
\]
**Signed multiplication**

- Positive or negative multiplicand, negative multiplier
  - 2n cycles, but can be reduced to n cycles using a special treatment of the upper half (see next slide)

<table>
<thead>
<tr>
<th>7</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>-3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

```
0 0 0 0 0 0 1 1 1 1
0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 0 0 0
0 0 1 1 1 1 0 0 0 0
1 1 1 0 0 0 0 0 0 0
1 1 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0
```

\[ \times \quad = \]

```
0 0 0 0 0 0 1 1 1 1
0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 0 0 0
0 0 1 1 1 1 0 0 0 0
1 1 1 0 0 0 0 0 0 0
1 1 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0
```

\[ = \]

\[ -128 \quad -128 \quad +64 \quad +32 \quad +8 \quad +2 \quad +1 \]

\[ +107 \]
Combinational signed multiplication extending addition to 2n bits – much extra circuitry
Hardware efficient combinational signed multiplier
Baugh-Wooley multiplier

- n-1 bit addition
- Special treatment of sign bits
- Special treatment of last stage
- Note the use of FA Co outputs
- Red line shows the longest propagation delay
Embedded multipliers in FPGAs

Altera Cyclone: multipliers arranged in columns surrounded by Logic Array Blocks (LABs)

Each multiplier is configured as one 18x18 multiplier or two 9x9 multipliers.

For multiplications greater than 18x18, multiple multipliers are blocked together. There is no restriction on the data width.
Numbers of embedded multipliers in Cyclone III and IV families

<table>
<thead>
<tr>
<th>Device Family</th>
<th>Device</th>
<th>Embedded Multipliers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cyclone III</td>
<td>EP3C5</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>EP3C10</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>EP3C16</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td>EP3C25</td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>EP3C40</td>
<td>126</td>
</tr>
<tr>
<td></td>
<td>EP3C55</td>
<td>156</td>
</tr>
<tr>
<td></td>
<td>EP3C80</td>
<td>244</td>
</tr>
<tr>
<td></td>
<td>EP3C120</td>
<td>288</td>
</tr>
<tr>
<td>Cyclone III LS</td>
<td>EP3CLS70</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>EP3CLS100</td>
<td>276</td>
</tr>
<tr>
<td></td>
<td>EP3CLS150</td>
<td>320</td>
</tr>
<tr>
<td></td>
<td>EP3CLS200</td>
<td>396</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Device Family</th>
<th>Device</th>
<th>Embedded Multipliers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cyclone IV GX</td>
<td>EP4CGX15</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>EP4CGX22</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>EP4CGX30</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>EP4CGX50</td>
<td>140</td>
</tr>
<tr>
<td></td>
<td>EP4CGX75</td>
<td>198</td>
</tr>
<tr>
<td></td>
<td>EP4CGX110</td>
<td>280</td>
</tr>
<tr>
<td></td>
<td>EP4CGX150</td>
<td>360</td>
</tr>
</tbody>
</table>

Altera DE2 Development Kit:
EP4CE115 – 266 multipliers
## Cyclone V Hardware Resources

<table>
<thead>
<tr>
<th>Resource</th>
<th>Member Code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A2</td>
</tr>
<tr>
<td>Logic Elements (LE) (K)</td>
<td>25</td>
</tr>
<tr>
<td>ALM</td>
<td>9,434</td>
</tr>
<tr>
<td>Register</td>
<td>37,736</td>
</tr>
<tr>
<td>Memory (Kb)</td>
<td></td>
</tr>
<tr>
<td>M10K</td>
<td>1,760</td>
</tr>
<tr>
<td>MLAB</td>
<td>196</td>
</tr>
<tr>
<td>Variable-precision DSP Block</td>
<td>25</td>
</tr>
<tr>
<td>18 x 18 Multiplier</td>
<td>50</td>
</tr>
<tr>
<td>PLL</td>
<td>4</td>
</tr>
</tbody>
</table>
Multiplier block architecture
Inferring multipliers from SystemVerilog code

Read “Recommended HDL Coding Styles”,

// unsigned multiplier
module unsigned_mult (output logic [15:0] out, input logic [7:0] a, b);
    assign out = a * b;
endmodule

// signed multiplier with I/O registers
module signed_mult (output logic signed [15:0] out, input logic signed [7:0] a, b, input logic clk);
    logic signed [7:0] a_reg, b_reg;
    logic signed [15:0] mult_out;

    assign mult_out = a_reg * b_reg;

    always_ff @ (posedge clk)
    begin
        a_reg <= a;
        b_reg <= b;
        out <= mult_out;
    end
endmodule
// signed multiplier-accumulator (a*b + c*d) with registers
module signed_mult_accum (output logic [15:0] out, input logic [7:0] a, b, c, d input logic clk, aclear ,clk_enable);
    logic signed [7:0] a_reg, b_reg,c_reg,d_reg;
    logic signed [15:0] mult0, mult1;

    always_ff @ (posedge clk or posedge aclear)
        if ( aclear) // clear all registers
            begin
                a_reg <= 8'b0;
                b_reg <= 8'b0;
                c_reg <= 8'b0;
                d_reg <= 8'b0;
                mult0 <= 16'b0;
                mult1 <= 16'b0;
                out <= 16'b0;
            end
        else begin // multiply and add
            a_reg <= a;
            b_reg <= b;
            c_reg <= c;
            d_reg <= d;
            mult0 <= a_reg * b_reg;
            mult1 <= c_reg * d_reg;
            out <= mult0 + mult1;
        end
endmodule
Altera’s Qsys System Integration Tool is used to design digital hardware systems that contain components such as processors, memories, input/output interfaces, timers, multipliers, etc.

Qsys allows a designer to choose components via a GUI. It then automatically generates the hardware system that connects all of the components together.

Qsys introduction and tutorial:

Also refer to: “Recommended HDL Coding Styles” in Altera Quartus Handbook vol. 1