ELEC6234
Embedded Hardware Design

Embedded hardware multipliers

Example

Multiply 7 x 5 using unsigned 3-bit binary representation

<table>
<thead>
<tr>
<th>M</th>
<th>111 multiplicand</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>101 multiplier</td>
</tr>
<tr>
<td></td>
<td>0000 (don't ADD, Q_0 = 0)</td>
</tr>
<tr>
<td></td>
<td>111 (ADD 7 shifted by 2 bits, Q_2=1)</td>
</tr>
<tr>
<td></td>
<td><strong>100011</strong> (result = 35)</td>
</tr>
</tbody>
</table>

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

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

Signed multiplication
- Can extend addition to 2n bits
- Negative multiplicant, positive multiplier
  - n cycles

Signed multiplication
- Positive or negative multiplicant, negative multiplier
  - 2n cycles, but can be reduced to n cycles using a special treatment of the upper half (see next slide)
### 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>EP4C10</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>EP4C100</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>EP4C15</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>EP4C14</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>EP4C12</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>EP4C17</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>EP4C1150</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>EP4C11500</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>EP4C115000</td>
<td>20</td>
</tr>
<tr>
<td>Cyclone IV GX</td>
<td>EP4C22</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>EP4C20</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td>EP4C200</td>
<td>140</td>
</tr>
<tr>
<td></td>
<td>EP4C23</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>EP4C2110</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>EP4C2150</td>
<td>300</td>
</tr>
</tbody>
</table>

**Altera DE2 Development Kit:**
### Cyclone V Hardware Resources

<table>
<thead>
<tr>
<th>Resource</th>
<th>Member Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic Elements (LSE) (K)</td>
<td>25 40 77 150 401</td>
</tr>
<tr>
<td>ALM</td>
<td>9543 18480 29080 56480 113560</td>
</tr>
<tr>
<td>Register</td>
<td>37734 79850 116330 235920 454240</td>
</tr>
<tr>
<td>Memory (Kn)</td>
<td>1760 3080 4660 6860 12200</td>
</tr>
<tr>
<td>Variable-precision DSP Block</td>
<td>196 303 424 836 1717</td>
</tr>
<tr>
<td>18 x 18 Multiplier</td>
<td>50 132 300 312 684</td>
</tr>
<tr>
<td>PLL</td>
<td>4 4 6 7 8</td>
</tr>
</tbody>
</table>

### Multiplier block architecture

- **Data In:** Input A, B, C, D
- **Data Out:** Output E, F

### Inferring multipliers from SystemVerilog code

Read "Recommended HDL Coding Styles", Quartus Prime Standard Edition Handbook Version 17.1 [Oct 2017], Vol 1, Chapter 12


#### Signed multiplier

```verilog
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

```verilog
module signed_mult (output logic signed [15:0] out, input logic signed [7:0] a, b);
assign out = a * b;
endmodule
```

#### Signed multiplier accumulator

```verilog
module signed_mult_accum (output logic [15:0] out, input logic [7:0] a, b, c, d input logic clk, aclr_enable);
logic signed [15:0] s_reg, d_reg,
logic signed [15:0] mul0, mul1;
always_ff @ (posedge clk or posedge aclr_enable) if (aclr_enable) begin
  s_reg <= 8'b0;
  d_reg <= 8'b0;
  mul0 <= 16'b0;
  mul1 <= 16'b0;
  out <= 16'b0;
end else begin
  s_reg <= a;
  d_reg <= b;
  mul0 <= s_reg * b_reg;
  mul1 <= c_reg * d_reg;
  out <= mul0 + mul1;
end
endmodule
```

### Inferring multiply-add hardware

```verilog
/* 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, aclr_enable];
logic signed [15:0] s_reg, d_reg,
logic signed [15:0] mul0, mul1;
always_ff @ (posedge clk or posedge aclr_enable) if (aclr_enable) begin
  s_reg <= 8'b0;
  d_reg <= 8'b0;
  mul0 <= 16'b0;
  mul1 <= 16'b0;
  out <= 16'b0;
end else begin
  s_reg <= a;
  d_reg <= b;
  mul0 <= s_reg * b_reg;
  mul1 <= c_reg * d_reg;
  out <= mul0 + mul1;
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