The Spartan 3e FPGA

- What’s inside the chip?
  - How does it implement random logic?
  - What other features can you use?
- What do all these things mean?
  - LUT, Slice, BRAM, DCM, IOB, CLB...
- Two important documents
  (linked to the class web site)
  - Spartan3e Family Complete Data Sheet
  - Spartan3e User Guide
What’s on the chip?

- CLB (Configurable Logic Blocks)
  - Logic and flip flops
  - 1,164 CLBs on our chip
  - Each CLB is 4 Slices
  - 500k total “system gates”
What’s on the chip?

• IOB (Input Output Blocks)
  • Communicate off chip
  • Our chip has 232 total pins in a 320 BGA package

• BRAM (Block RAM)
  • On-chip SRAM
  • 18k bits per block
  • 20 blocks on our chip
What’s on the chip?

- Multiplier
  - Custom 18x18 multiplier
  - One per RAM block...

- DCM (Digital Clock Manager)
  - Clock generation and distribution
  - Four on our chip
What’s on the chip?

- Programmable Interconnect
  - Connect everything together
  - Perhaps the most critical part of the chip!

CLB: Configurable Logic Block

- 4 “Slices” per CLB
  - The slices work together to make logic, flip flops, distributed RAM, or shift registers
  - Connected to other CLBs through Switch Matrix
Left and Right Slices

- SRL16 = 16-bit shift register
- RAM16 = 16-bit RAM (16x1 bit memory)
- LUT4 = four-bit lookup table (16x1 bit memory)
- SLICEM = slice that can be memory or logic
- SLICEL = slice that can only be logic

What's Really in a Slice?
LUT 4 – Basic Building Block

- RAM memory 4-bit input, 1-bit output
- Can implement any logic function of up to 4 inputs

Example: Implement 3-input AND

Assume A0 - A2 are used

<table>
<thead>
<tr>
<th>A0</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>x</td>
<td>1</td>
</tr>
</tbody>
</table>

Patrick Schaumont
Spring 2008
Slice Muxes extend LUT4

Once CLB – up to LUT7
Top Half of a SliceM (left)
Logic-only (combinational)

Logic + register (sequential)
Just register

Fast Carry Path (arithmetic)
Fast Carry Path (arithmetic)
Mapping to CLBs

- Each LUT can go through a flip flop
  - So, these circuits map to the same number of Slices

Mapping to CLBs

- How about these?
Mapping to CLBs

- How about these?

CLB Summary

- Each CLB = 4 slices
- Each slice contains
  - 2 LUT-4
    - LUT can be random logic, or 16x1bit RAM or SR
  - 2 flip flop
  - MUXs
  - Carry logic
- ISE reports how many slices you use
  - among lots of other things...
IO Blocks

- Connections to the outside world
  - Each pin can be configured a large number of ways
  - Different signaling voltages and drive currents

<table>
<thead>
<tr>
<th>Single-Ended I/O STANDARD</th>
<th>V_{CCO} Supply/Compatibility</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.2V</td>
</tr>
<tr>
<td>LVTTL</td>
<td>-</td>
</tr>
<tr>
<td>LVC莫斯33</td>
<td>-</td>
</tr>
<tr>
<td>LVC莫斯25</td>
<td>-</td>
</tr>
<tr>
<td>LVC莫斯18</td>
<td>-</td>
</tr>
<tr>
<td>LVCmos15</td>
<td>-</td>
</tr>
<tr>
<td>LVC莫斯12</td>
<td>Input/Output</td>
</tr>
<tr>
<td>PCI33_3</td>
<td>-</td>
</tr>
<tr>
<td>PCI66_3</td>
<td>-</td>
</tr>
</tbody>
</table>

NOTE! No 5v!

<table>
<thead>
<tr>
<th>Single-Ended I/O STANDARD</th>
<th>V_{CCO} Supply/Compatibility</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.2V</td>
</tr>
<tr>
<td>LVTTL</td>
<td>-</td>
</tr>
<tr>
<td>LVC莫斯33</td>
<td>-</td>
</tr>
<tr>
<td>LVCmos25</td>
<td>-</td>
</tr>
<tr>
<td>LVCmos18</td>
<td>-</td>
</tr>
<tr>
<td>LVCmos15</td>
<td>-</td>
</tr>
<tr>
<td>LVCmos12</td>
<td>Input/Output</td>
</tr>
<tr>
<td>PCI33_3</td>
<td>-</td>
</tr>
<tr>
<td>PCI66_3</td>
<td>-</td>
</tr>
</tbody>
</table>
Inside an IOB

Interconnect

• Actually the most important part of the FPGA!
  - Consumes the most area on the die
  - Consumes the most power on the die
  - In most cases, wires limit the performance
• But, hardly mentioned in the datasheet
  - People are more impressed with logic
Interconnect

- RAM-programmable switches
  - 2,270,208 bits of configuration RAM!
  - Compare to 368,640 total bits of Block RAM
  - or 74,752 total bits of Distributed RAM (LUTs)
- Hierarchical organization
  - Many fast, short wires with small drive
  - Fewer longer wires with high drive
  - LOTS of work goes into picking just the right mix!

CLBs connect to 'switch matrix' which connects to the on-chip network

Each switch matrix interconnects many different wires
Four types of wires

Clock Routing

- Routed on a separate dedicated network
  - Another reason to avoid gated clocks
- Recursive “Fish bone” network that minimizes clock skew
- Clocks come from off-chip, or from a DCM
Spartan XC3E500S

<table>
<thead>
<tr>
<th>Device</th>
<th>System Gates</th>
<th>Equivalent Logic Cells</th>
<th>Rows</th>
<th>Columns</th>
<th>Total CLBs</th>
<th>Total Slices</th>
<th>Distributed RAM bits</th>
<th>Block RAM bits</th>
<th>Dedicated Multipliers</th>
<th>DCMs</th>
<th>Maximum User I/O</th>
<th>Maximum Differential I/O Pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>XC3S1000E</td>
<td>100K</td>
<td>1,190</td>
<td>22</td>
<td>15</td>
<td>249</td>
<td>160</td>
<td>10K</td>
<td>72K</td>
<td>4</td>
<td>2</td>
<td>153</td>
<td>40</td>
</tr>
<tr>
<td>XC3S2000E</td>
<td>250K</td>
<td>5,508</td>
<td>34</td>
<td>25</td>
<td>612</td>
<td>2,148</td>
<td>33K</td>
<td>216K</td>
<td>12</td>
<td>4</td>
<td>172</td>
<td>69</td>
</tr>
<tr>
<td>XC3S2500E</td>
<td>500K</td>
<td>11,146</td>
<td>40</td>
<td>34</td>
<td>1,164</td>
<td>4,056</td>
<td>73K</td>
<td>360K</td>
<td>20</td>
<td>4</td>
<td>232</td>
<td>92</td>
</tr>
<tr>
<td>XC3S3200E</td>
<td>1200K</td>
<td>18,512</td>
<td>50</td>
<td>36</td>
<td>2,168</td>
<td>8,072</td>
<td>130K</td>
<td>504K</td>
<td>28</td>
<td>8</td>
<td>304</td>
<td>124</td>
</tr>
<tr>
<td>XC3S1600E</td>
<td>1600K</td>
<td>33,192</td>
<td>58</td>
<td>58</td>
<td>3,688</td>
<td>14,752</td>
<td>291K</td>
<td>648K</td>
<td>96</td>
<td>8</td>
<td>376</td>
<td>156</td>
</tr>
</tbody>
</table>

Block RAM

We’ve seen details of these already…
Behavioral Template

```verilog
parameter RAM_WIDTH = <ram_width>;
parameter RAM_ADDR_BITS = <ram_addr_bits>;

reg [RAM_WIDTH-1:0] <ram_name> [2**RAM_ADDR_BITS-1:0];
reg [RAM_WIDTH-1:0] <output_data>;

// The following code is only necessary if you wish to initialize the RAM
// contents via an external file | use (readmemh for binary data)
initial
  (readmemh("<data_file_name>", <ram_name>, <begin_address>, <end_address>));

always @(posedge <clock>) begin
  if (enable) begin
    if (write_enable) begin
      <ram_name>[<address>] <= <input_data>;
      <output_data> <= <ram_name>[<address>];
    end
  end
  <output_data> <= <ram_name>[<address>];
end
```

CS/EE 3710

---

Structural Template

```verilog
// RAM16_09_99 : In order to incorporate this function into the design
// instance is the body of the design code. The instance name
// declaration is RAM16_09_99 inst and the port declarations within the
// code may be changed to property reference and
// connect this function to the design. All inputs
// and outputs must be connected.

// <<<<< Cut code below this line >>>>>

// RAM16_09_99 : Virtex-5/II-Pro, Spartan-3/III 2k x 8 + 1 Parity Bit Dual-Port RAM
// Xilinx HDL Language Template, version 10.1.3

RAM16_09_99 @
  .INIT(1'1[16000]), // Value of output RAM registers on Port A at startup
  .INIT(1'1[32000]), // Value of output RAM registers on Port B at startup
  .EVVAL(1'1[16000]), // Port A output value upon EOR assertion
  .EVVAL(1'1[32000]), // Port B output value upon EOR assertion
  .WRITE_MODE("WRITE_FIRST"); // WRITE_FIRST, READ_FIRST or NO_CHANGE
  .WRITE_MODE("WRITE_FIRST"); // WRITE_FIRST, READ_FIRST or NO_CHANGE

// The following INIT_xx declarations specify the initial contents of the RAM
// Address 0 to 63
  .INIT(0'1[16'1[00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00]]),
  .INIT(0'1[16'1[00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00]]),
  .INIT(0'1[16'1[00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00]]),
  .INIT(0'1[16'1[00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00]]),
  .INIT(0'1[16'1[00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00]]),
  .INIT(0'1[16'1[00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00]]),
  .INIT(0'1[16'1[00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00]]),
  .INIT(0'1[16'1[00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00]]),
  .INIT(0'1[16'1[00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00]]),
  .INIT(0'1[16'1[00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00]]),
  .INIT(0'1[16'1[00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00]]),
  .INIT(0'1[16'1[00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00]]),
  .INIT(0'1[16'1[00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00]]),
  .INIT(0'1[16'1[00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00]]),
  .INIT(0'1[16'1[00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00]]),
  .INIT(0'1[16'1[00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00]]),
  .INIT(0'1[16'1[00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00]]),
  .INIT(0'1[16'1[00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00]])
```

CS/EE 3710
Table 6-4: Single-Port and Dual-Port Distributed RAMs

<table>
<thead>
<tr>
<th>Primitve</th>
<th>RAM Size (Depth x Width)</th>
<th>Type</th>
<th>Address Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM16XS</td>
<td>16 x 1</td>
<td>Single-port</td>
<td>A3, A2, A1, A0</td>
</tr>
<tr>
<td>RAM32XS</td>
<td>32 x 1</td>
<td>Single-port</td>
<td>A4, A3, A2, A1, A0</td>
</tr>
<tr>
<td>RAM64XS</td>
<td>64 x 1</td>
<td>Single-port</td>
<td>A5, A4, A3, A2, A1, A0</td>
</tr>
<tr>
<td>RAM16XD</td>
<td>16 x 1</td>
<td>Dual-port</td>
<td>A3, A2, A1, A0</td>
</tr>
</tbody>
</table>

Figure 6-3 shows generic single-port and dual-port distributed RAM primitives. The A[WX] and DPR[0] signals are address buses.

Table 6-5: Dual-Port RAM Function

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>WE (mode)</td>
<td>WCLK</td>
</tr>
<tr>
<td>0 (read)</td>
<td>X</td>
</tr>
<tr>
<td>1 (read)</td>
<td>0</td>
</tr>
<tr>
<td>1 (write)</td>
<td>T</td>
</tr>
<tr>
<td>1 (write)</td>
<td>1</td>
</tr>
</tbody>
</table>
Distributed RAM

parameter RAM_WIDTH = <ram_width>;
parameter RAM_ADDR_BITS = <ram_addr_bits>;

reg [RAM_WIDTH-1:0] <ram_name> [(2^RAM_ADDR_BITS)-1:0];
wire [RAM_WIDTH-1:0] <output_data>;
<reg_or_wire> [RAM_ADDR_BITS-1:0] <read_address>, <write_address>;
<reg_or_wire> [RAM_WIDTH-1:0] <input_data>;
always @(posedge <clock>)
  if (<write_enable>)
    <ram_name>[<write_address>] <= <input_data>;
assign <output_data> = <ram_name>[<read_address>];

Dual-Port Distributed RAM
Distributed RAM

```verilog
RAM16X1D #(  
  .INIT(16'h0000) // Initial contents of RAM
)
  RAM16X1D_inst |
  .BPO(SPO), // Read-only 1-bit data output for BPO
  .SP0(SPO), // R/W 1-bit data output for A0-A3
  .A0(A0), // R/W address[0] input bit
  .D(D), // Write 1-bit data input
  .ERP0(ERP0), // Read address[0] input bit
  .ERP1(ERP1), // Read address[1] input bit
  .ERP2(ERP2), // Read address[2] input bit
  .ERP3(ERP3), // Read address[3] input bit
  .WCLK(WCLK), // Write clock input
  .WE(WE) // Write enable input
);```

Dual-Port Distributed RAM

---

Digital Clock Manager (DCM)

DCMs integrate advanced clocking capabilities directly into the FPGA’s global clock distribution network. Consequently, DCMs solve a variety of common clocking issues, especially in high-performance, high-frequency applications:

- **Eliminate Clock Skew**, either within the device or to external components, to improve overall system performance and to eliminate clock distribution delays.
- **Phase Shift** a clock signal, either by a fixed fraction of a clock period or by incremental amounts.
- **Multiply or Divide an Incoming Clock Frequency** or synthesize a completely new frequency by a mixture of clock multiplication and division.
- **Condition a Clock**, ensuring a clean output clock with a 50% duty cycle.
- **Mirror, Forward, or Rebuffer a Clock Signal**, often to deskew and convert the incoming clock signal to a different I/O standard—for example, forwarding and converting an incoming LVTTL clock to LVDS.
- **Any or all the above functions**, simultaneously.
Digital Clock Manager (DCM)

a. Global Buffer Inputs and Clock Buffers Drive a Low-Skew Global Network in the FPGA

b. A Digital Clock Manager (DCM) Inserts Directly into the Global Clock Path
Clock Skew

Figure 3.19: Eliminating Skew on Internal Clock Signals
module mult18x18sio(a, b, clk, prod);
    input [7:0] a;
    input [7:0] b;
    input clk;
    output [15:0] prod;
    reg [15:0] prod;
    always @(posedge clk) prod <= a*b;
endmodule
Synthesis Output (mips example)

<table>
<thead>
<tr>
<th>Logic Utilization</th>
<th>Used</th>
<th>Available</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Slice Flip Flops</td>
<td>73</td>
<td>3,912</td>
<td>15%</td>
</tr>
<tr>
<td>Number of 4 input LUTs</td>
<td>193</td>
<td>3,912</td>
<td>25%</td>
</tr>
</tbody>
</table>

Logic Distribution

<table>
<thead>
<tr>
<th></th>
<th>Used</th>
<th>Available</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of occupied Slices</td>
<td>123</td>
<td>4,096</td>
<td>25%</td>
</tr>
<tr>
<td>Number of Slices containing only related logic</td>
<td>123</td>
<td>123</td>
<td>100%</td>
</tr>
<tr>
<td>Number of Slices containing unrelated logic</td>
<td>0</td>
<td>123</td>
<td>0%</td>
</tr>
</tbody>
</table>

Total Number of 4 input LUTs

<table>
<thead>
<tr>
<th></th>
<th>Used</th>
<th>Available</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number used as logic</td>
<td>161</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number used for Dual Port RAMs</td>
<td>32</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Number of bonded I/O pins

<table>
<thead>
<tr>
<th></th>
<th>Used</th>
<th>Available</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of bonded</td>
<td>19</td>
<td>232</td>
<td>0%</td>
</tr>
<tr>
<td>Number of RAMB16s</td>
<td>1</td>
<td>20</td>
<td>5%</td>
</tr>
<tr>
<td>Number of BUF3MUXs</td>
<td>1</td>
<td>24</td>
<td>4%</td>
</tr>
</tbody>
</table>
Synthesis Output (mips example)
Implement Output (mips example)
Conclusion

- FPGAs are complex beasts!
  - Made to be very general and flexible
- ASIC vs. FPGA?
  - Rule of thumb, FPGA about 5 times slower clock than ASIC
  - FPGAs consume more power
  - FPGAs are bigger for the same function
  - ASICs are much more expensive to develop
    - NRE – Non-Recurring Engineering

ASIC vs. FPGA

<table>
<thead>
<tr>
<th>FPGA &amp; ASIC Design Advantages</th>
<th>ASIC Design Advantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Faster time-to-market - no layout, masks or other manufacturing steps are needed</td>
<td>Full custom capability - for design since device is manufactured to design specs</td>
</tr>
<tr>
<td>No upfront NRE (nonrecurring expenses) - cost typically associated with an ASIC design</td>
<td>Lower unit costs - for very high volume designs</td>
</tr>
<tr>
<td>Simpler design cycle - due to software that handles much of the routing, placement, and timing</td>
<td>Smaller form factor - since device is manufactured to design specs</td>
</tr>
<tr>
<td>More predictable project cycle - due to elimination of potential respins, wafer capacities, etc</td>
<td>Higher raw internal clock speeds</td>
</tr>
<tr>
<td>Field reprogramability - new bitstream can be uploaded remotely</td>
<td></td>
</tr>
</tbody>
</table>