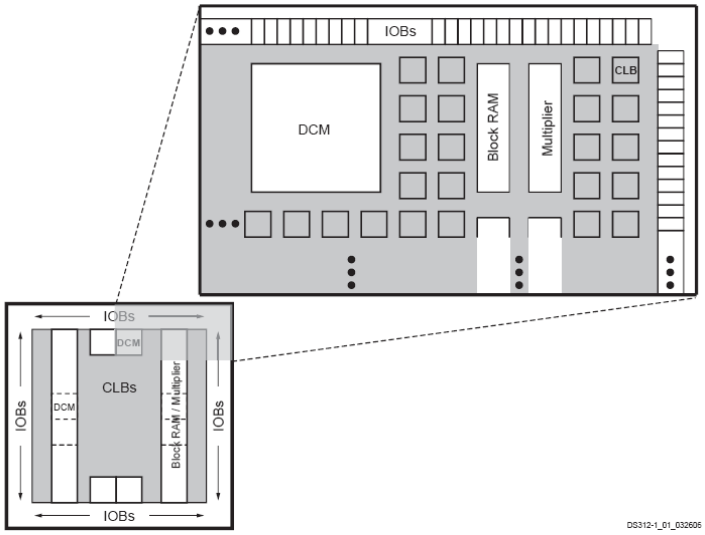
The Spartan 3e FPGA

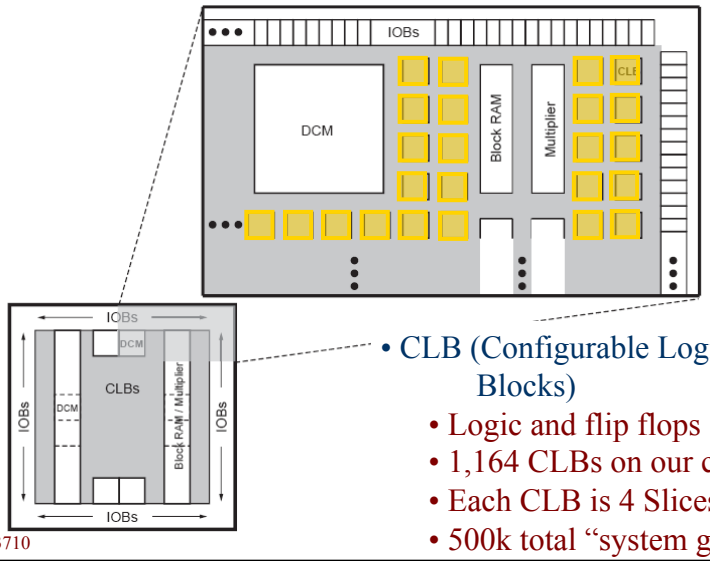
The Spartan 3e FPGA

- 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?

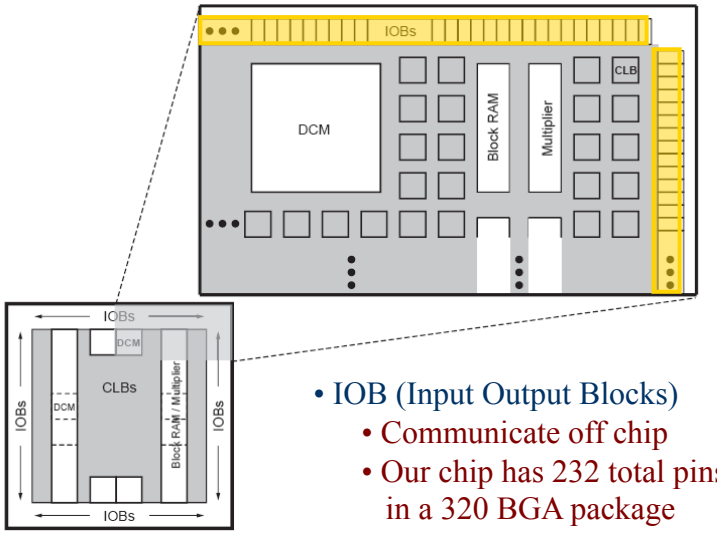


What's on the chip?



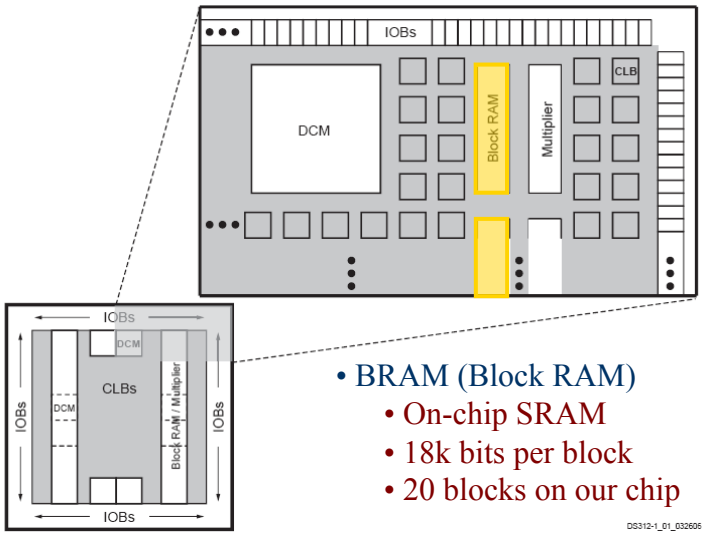
- CLB (Configurable Logic Blocks)
  - Logic and flip flops
  - 1,164 CLBs on our chip
  - Each CLB is 4 Slices
  - 500k total "system gates"

What's on the chip?



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

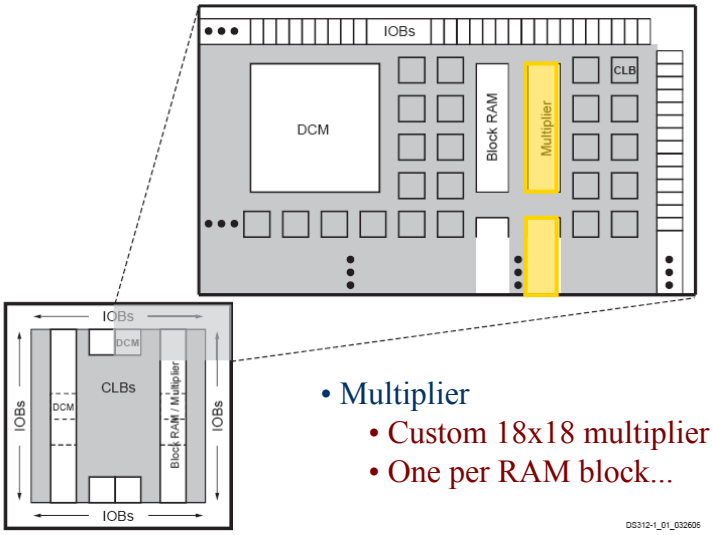
What's on the chip?



- BRAM (Block RAM)
  - On-chip SRAM
  - 18k bits per block
  - 20 blocks on our chip

DS312-1_01_032606

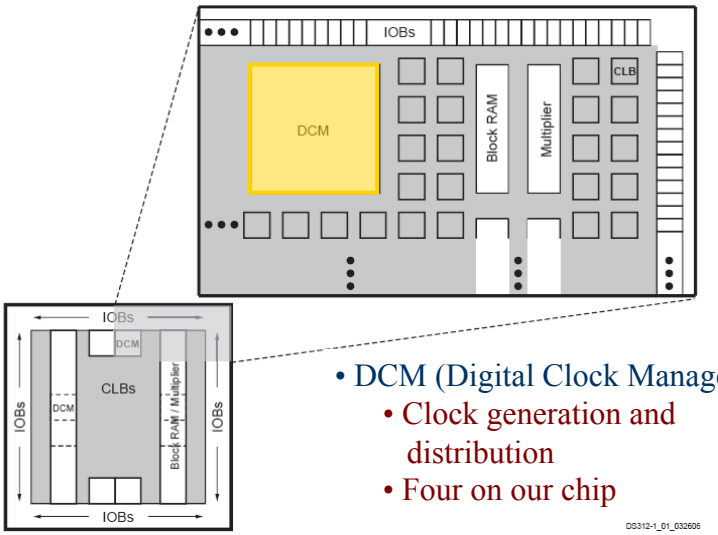
What's on the chip?



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

DS312-1_01_032606

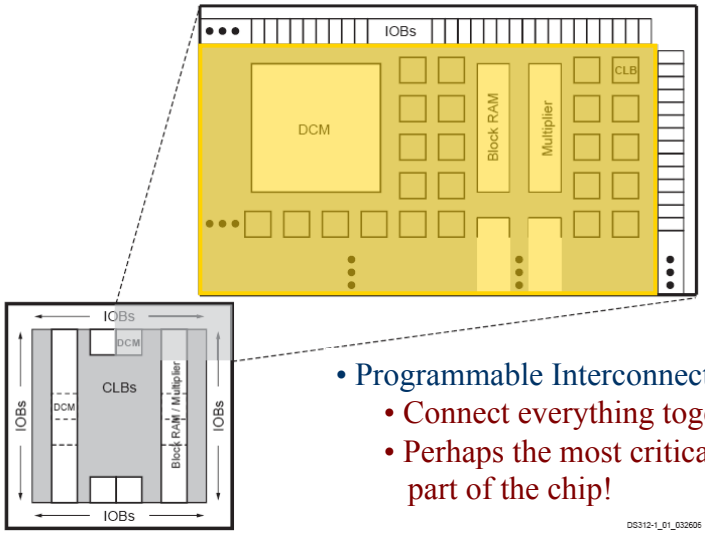
What's on the chip?



- DCM (Digital Clock Manager
  - Clock generation and distribution
  - Four on our chip

DS312-1_01_032606

What's on the chip?

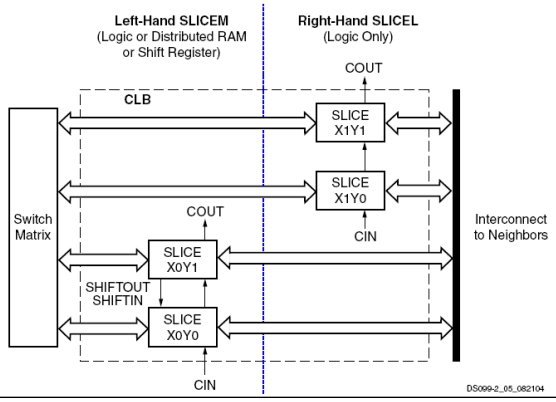


- Programmable Interconnect
  - Connect everything together
  - Perhaps the most critical part of the chip!

DS312-1_01_032605

CLB: Configurable Logic Block

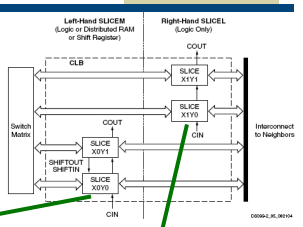
- 4 "Slices" per CLB
  - The slices work together to make logic, flip flops, distributed RAM, or shift registers
  - Connected to other CLBs through Switch Matrix

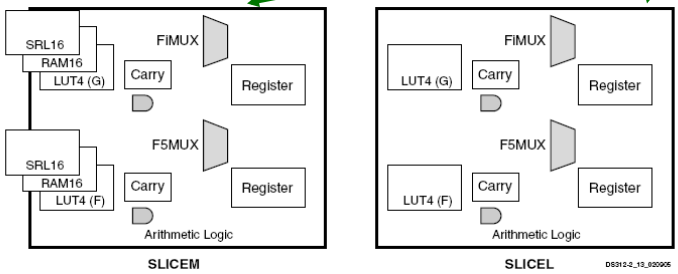


DS099-2_05_082104

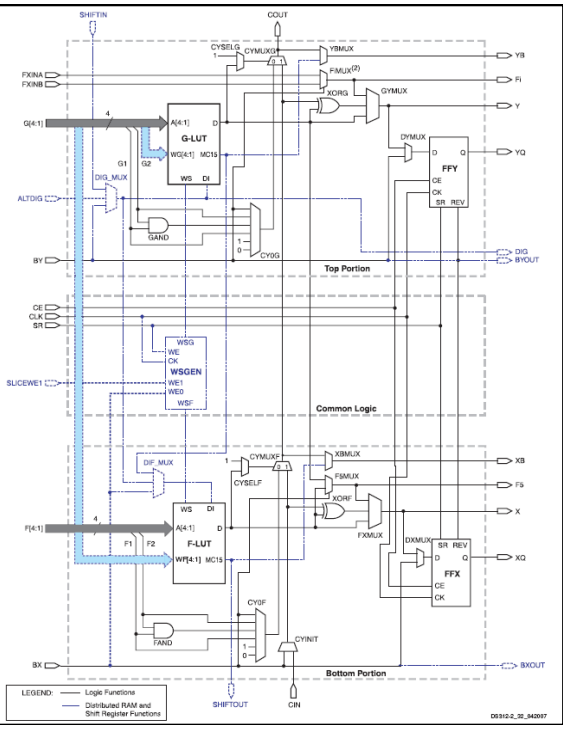
Left and Right Slices

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



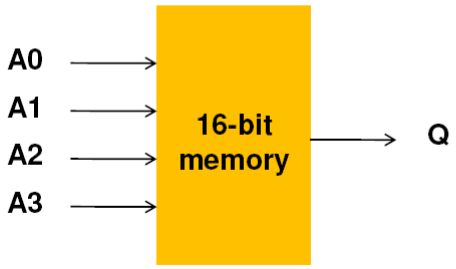


What's Really in a Slice?



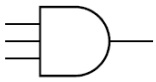
LUT 4 – Basic Building Block

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

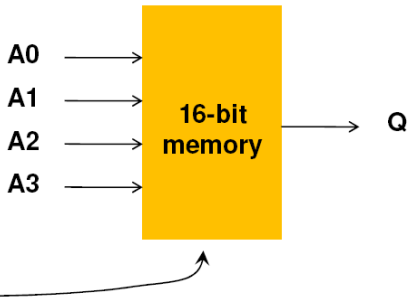


LUT 4 – Basic Building Block

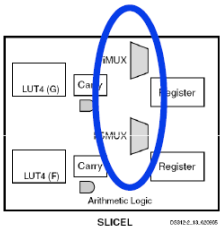
- Example: Implement 3-input AND
- Assume A0 - A2 are used

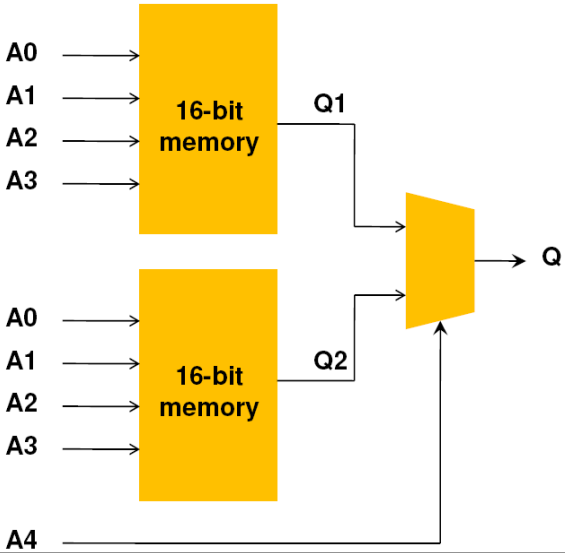


A0	A1	A2	A3	Q
0	0	0	x	0
0	0	1	x	0
0	1	0	x	0
0	1	1	x	0
1	0	0	x	0
1	0	1	x	0
1	1	0	x	0
1	1	1	x	1



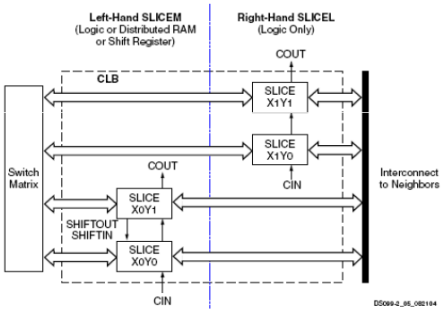
Slice Muxes extend LUT4

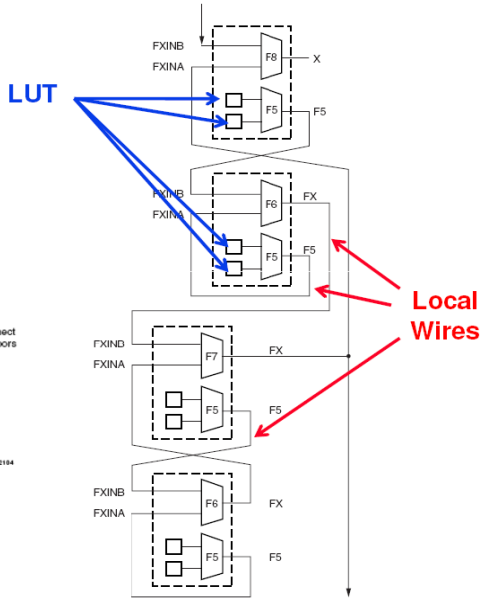




Patrick Schaumont
Spring 2008

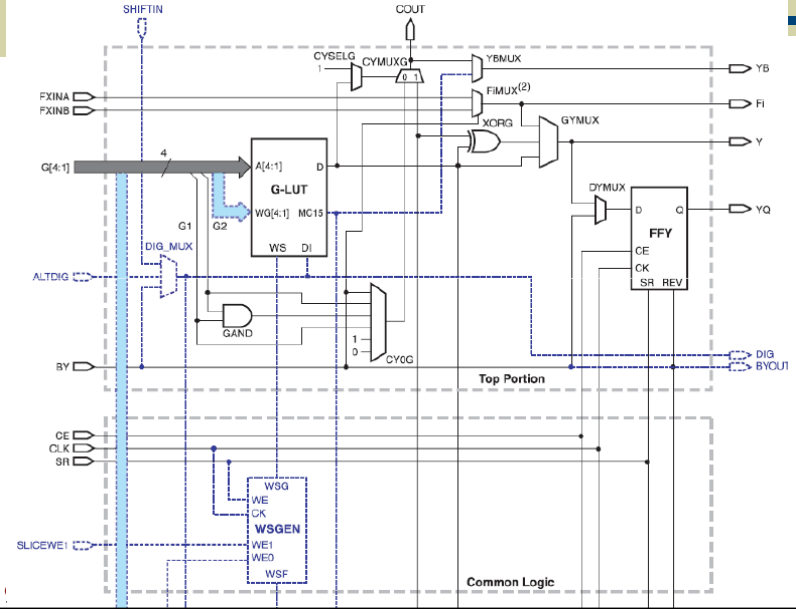
Once CLB – up to LUT7



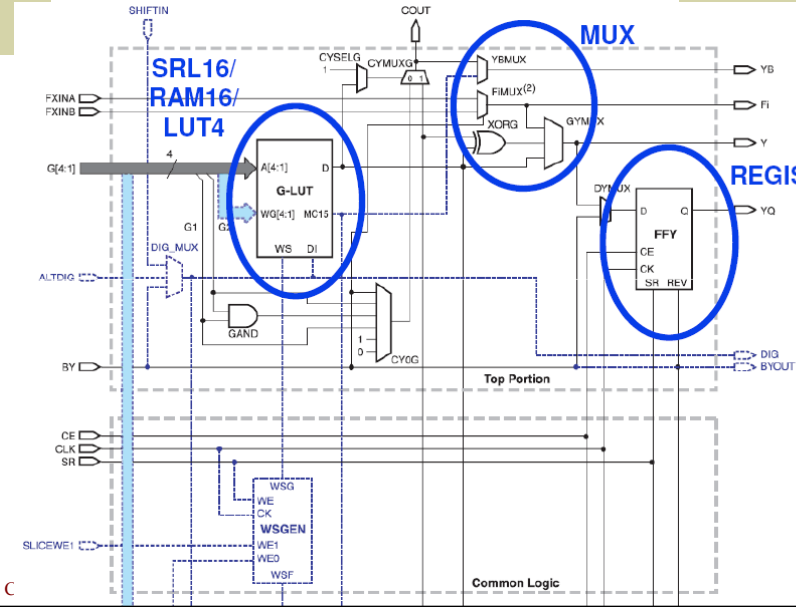


Patrick Schaumont
Spring 2008

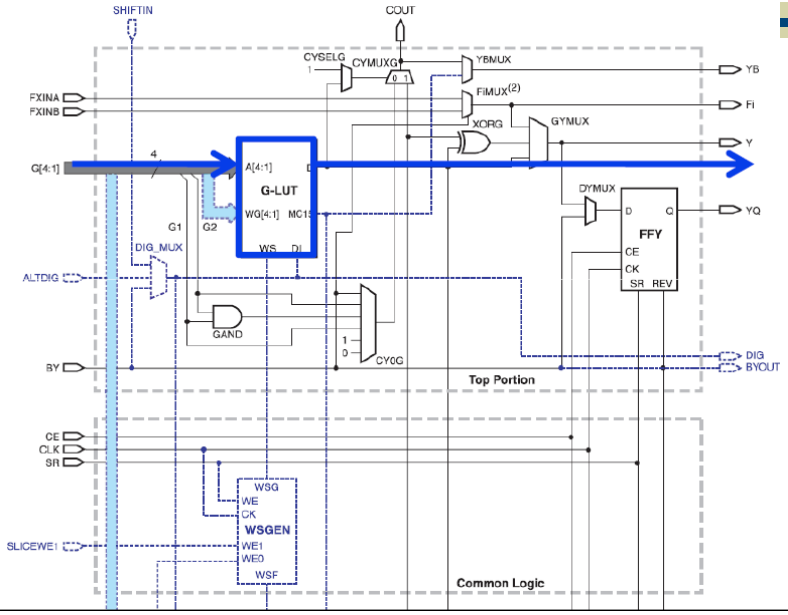
Top Half of a SliceM (left)



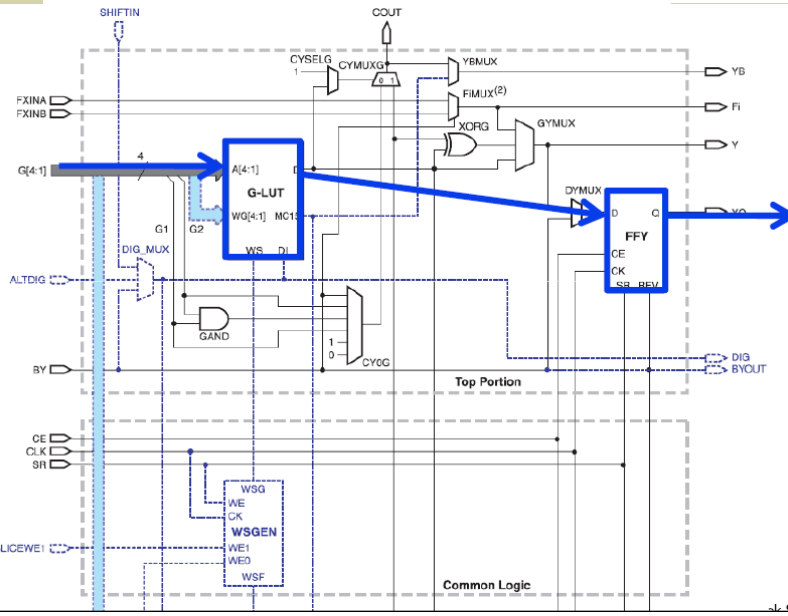
Top Half of a SliceM (left)



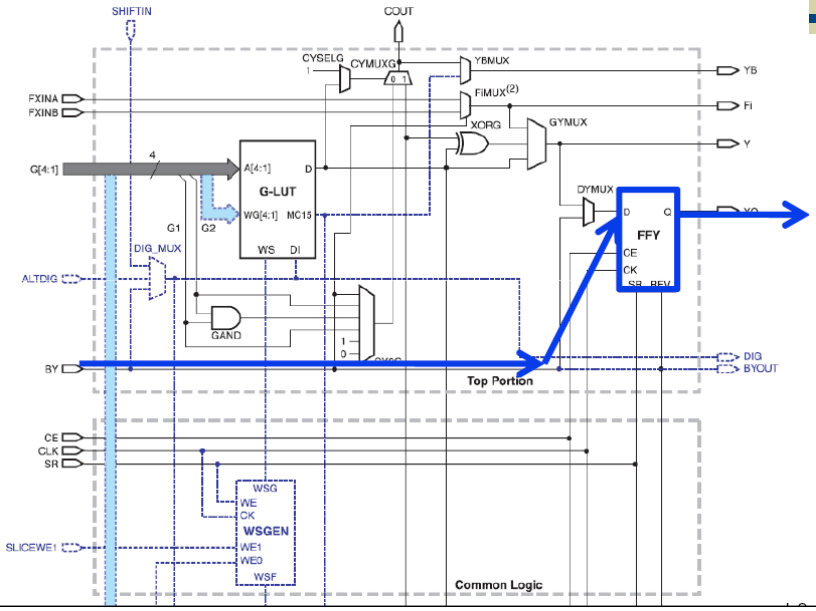
Logic-only (combinational)



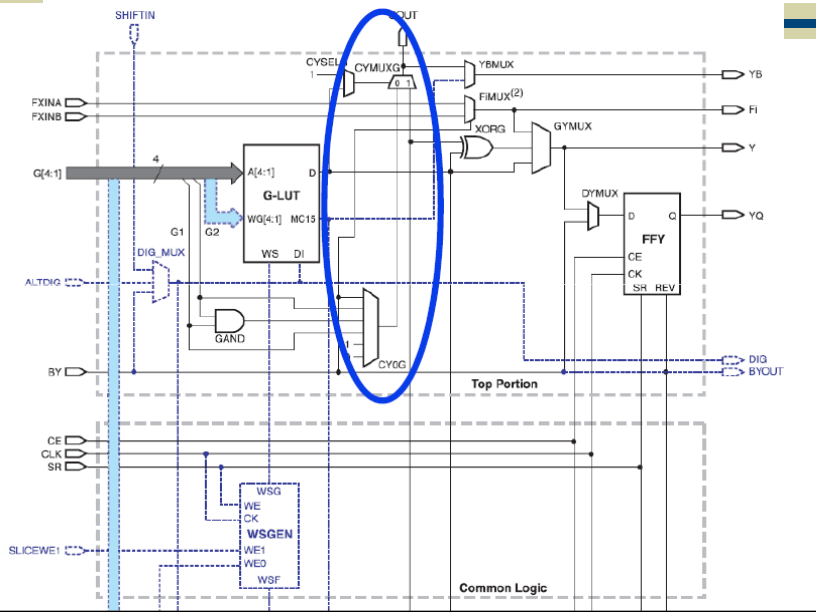
Logic + register (sequential)



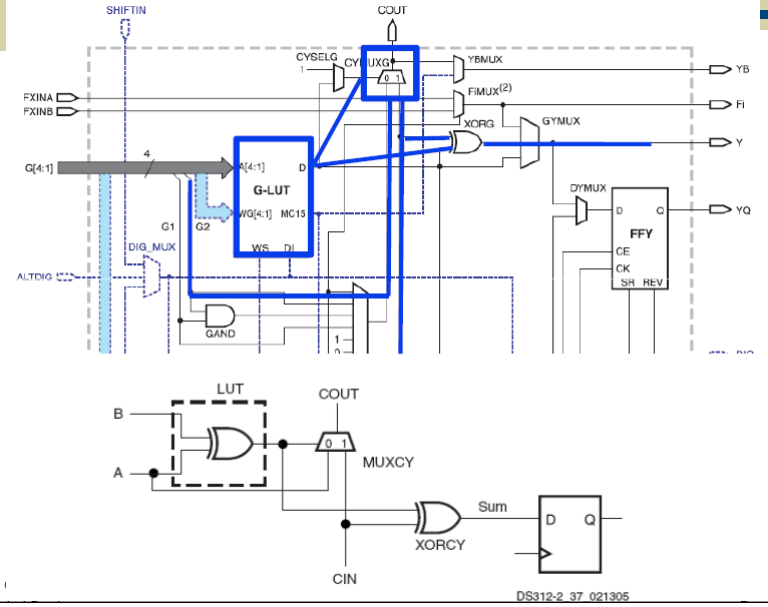
Just register



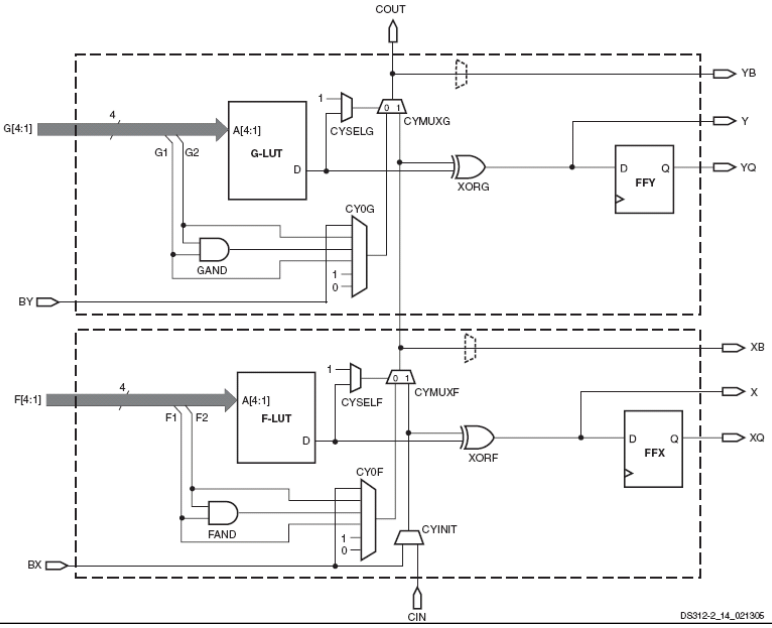
Fast Carry Path (arithmetic)



Fast Carry Path (arithmetic)

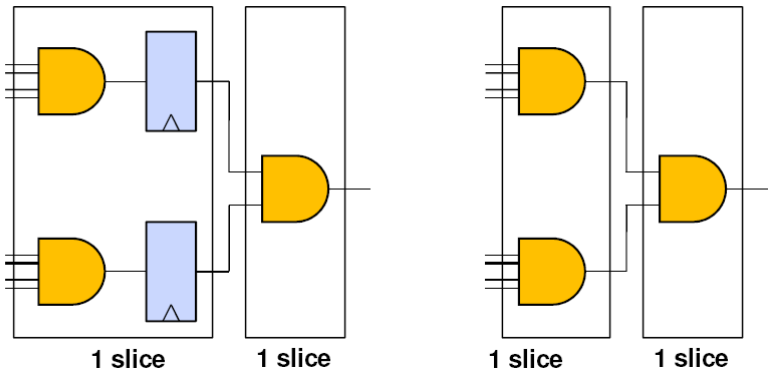


Fast Carry Path (arithmetic)



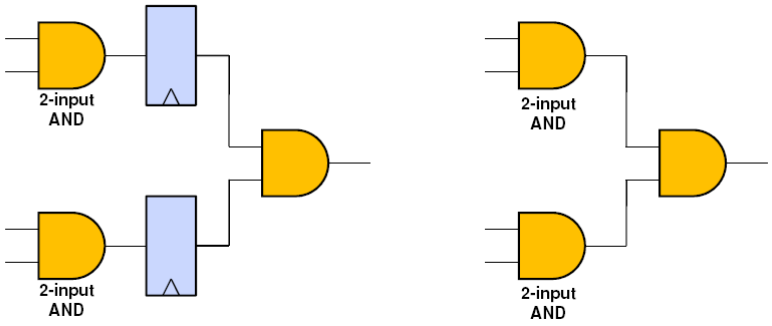
Mapping to CLBs

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



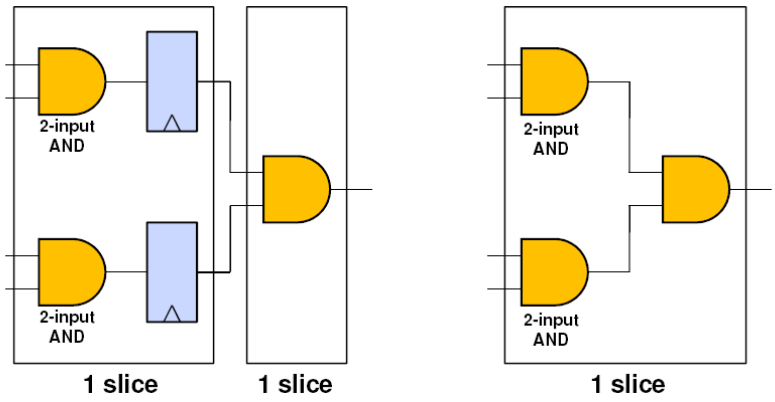
Mapping to CLBs

- 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

Single-Ended IOSTANDARD	V_{CCO} Supply/Compatibility				
	1.2V	1.5V	1.8V	2.5V	3.3V
LVTTL	-	-	-	-	Input/ Output
LVCMOS33	-	-	-	-	Input/ Output
LVCMOS25	-	-	-	Input/ Output	Input
LVCMOS18	-	-	Input/ Output	Input	Input
LVCMOS15	-	Input/ Output	Input	Input	Input
LVCMOS12	Input/ Output	Input	Input	Input	Input
PCI33_3	-	-	-	-	Input/ Output
PCI66_3	-	-	-	-	Input/ Output

IO Blocks

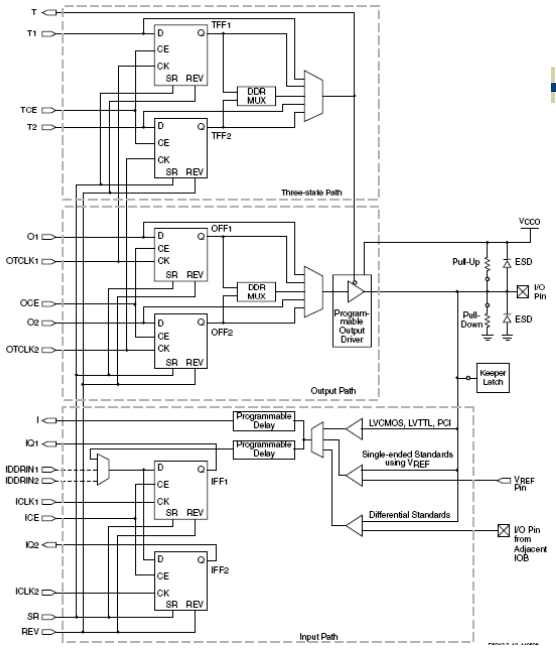
- Connections to the outside world
  - Each pin can be configured a large number of ways
  - Different signaling voltages and drive currents

NOTE! No 5v!

IOSTANDARD	Output Drive Current (mA)					
	2	4	6	8	12	16
LVTTL	✓	✓	✓	✓	✓	✓
LVCMOS33	✓	✓	✓	✓	✓	✓
LVCMOS25	✓	✓	✓	✓	✓	-
LVCMOS18	✓	✓	✓	✓	-	-
LVCMOS15	✓	✓	✓	-	-	-
LVCMOS12	✓	-	-	-	-	-

Single-Ended IOSTANDARD	V_{CCO} Supply/Compatibility				
	1.2V	1.5V	1.8V	2.5V	3.3V
LVTTL	-	-	-	-	Input/ Output
LVCMOS33	-	-	-	-	Input/ Output
LVCMOS25	-	-	-	Input/ Output	Input
LVCMOS18	-	-	Input/ Output	Input	Input
LVCMOS15	-	Input/ Output	Input	Input	Input
LVCMOS12	Input/ Output	Input	Input	Input	Input
PCI33_3	-	-	-	-	Input/ Output
PCI66_3	-	-	-	-	Input/ Output

Inside an IOB



Interconnect

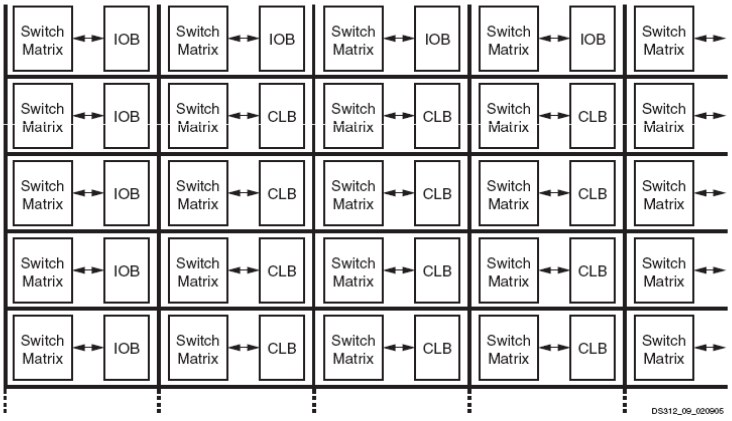
- Actually the most important part of the FPGA!
  - Consumes the most area on the die
  - Consumes the most power on the die
  - In most cases, wires limit the performance
- But, hardly mentioned in the datasheet
  - People are more impressed with logic

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!

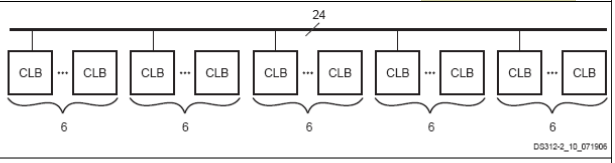
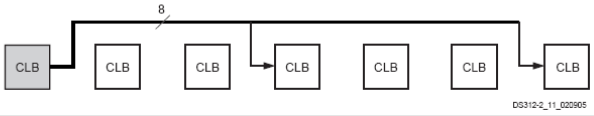
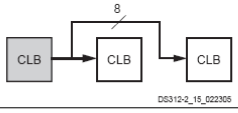
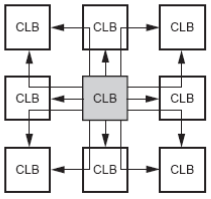
Interconnect

- CLBs connect to 'switch matrix' which connects to the on-chip network
- Each switch matrix interconnects many different wires



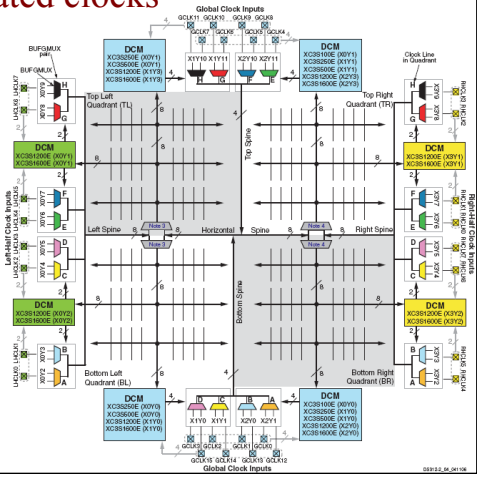
Interconnect

Four types of wires

Horizontal and Vertical Long Lines (horizontal channel shown as an example)	
Horizontal and Vertical Hex Lines (horizontal channel shown as an example)	
Horizontal and Vertical Double Lines (horizontal channel shown as an example)	
Direct Connections	

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

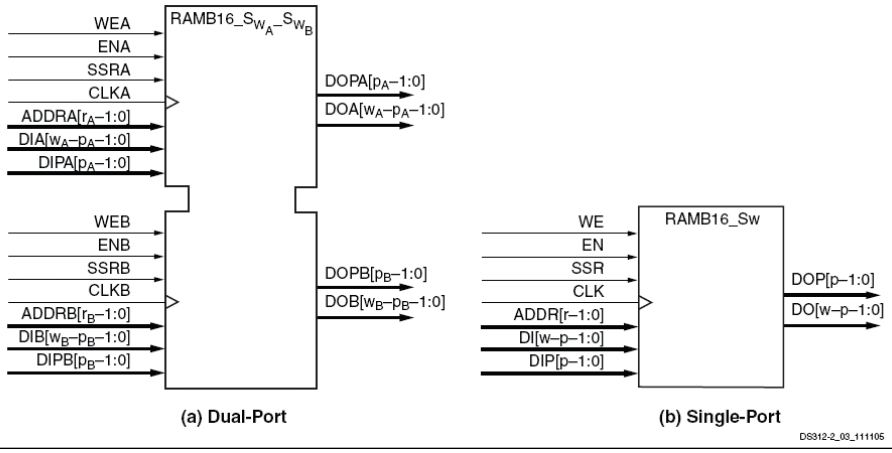


Spartan XC3E500S

Device	System Gates	Equivalent Logic Cells	CLB Array (One CLB = Four Slices)				Distributed RAM bits[1]	Block RAM bits[1]	Dedicated Multipliers	DCMs	Maximum User I/O	Maximum Differential I/O Pairs
			Rows	Columns	Total CLBs	Total Slices						
XC3S100E	100K	2,160	22	16	240	960	15K	72K	4	2	108	40
XC3S250E	250K	5,508	34	26	612	2,448	38K	216K	12	4	172	68
XC3S500E	500K	10,476	46	34	1,164	4,656	73K	360K	20	4	232	92
XC3S1200E	1200K	19,512	60	46	2,168	8,672	136K	504K	28	8	304	124
XC3S1600E	1600K	33,192	76	58	3,688	14,752	231K	648K	36	8	376	156

Block RAM

We've seen details of these already…



Behavioral Template

```
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_dataA>, <output_dataB>;

<reg_or_wire> [RAM_ADDR_BITS-1:0] <addressA>, <addressB>;
<reg_or_wire> [RAM_WIDTH-1:0] <input_dataA>;

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

always @(posedge <clock>) begin
   if (<enableA>) begin
      if (<write_enableA>)
         <ram_name>[<addressA>] <= <input_dataA>;
      <output_dataA> <= <ram_name>[<addressA>];
   end
   if (<enableB>)
      <output_dataB> <= <ram_name>[<addressB>];
end
```

Dual-port
1 R/W
1 R

Structural Template

```
//  RAMB16_S9_S9 : In order to incorporate this function into the design,
//    Verilog    : the following instance declaration needs to be placed
//   instance    : in the body of the design code.  The instance name
//  declaration  : (RAMB16_S9_S9_inst) and/or the port declarations within the
//     code      : parenthesis may be changed to properly reference and
//               : connect this function to the design.  All inputs
//               : and outputs must be connected.

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

   // RAMB16_S9_S9: Virtex-II/II-Pro, Spartan-3/3E 2k x 8 + 1 Parity bit Dual-Port RAM
   // Xilinx HDL Language Template, version 10.1.3

   RAMB16_S9_S9 #(
      .INIT_A(9'h000),  // Value of output RAM registers on Port A at startup
      .INIT_B(9'h000),  // Value of output RAM registers on Port B at startup
      .SRVAL_A(9'h000), // Port A output value upon SSR assertion
      .SRVAL_B(9'h000), // Port B output value upon SSR assertion
      .WRITE_MODE_A("WRITE_FIRST"), // WRITE_FIRST, READ_FIRST or NO_CHANGE
      .WRITE_MODE_B("WRITE_FIRST"), // WRITE_FIRST, READ_FIRST or NO_CHANGE
      .SIM_COLLISION_CHECK("ALL"), // "NONE", "WARNING_ONLY", "GENERATE_X_ONLY", "ALL"

      // The following INIT_xx declarations specify the initial contents of the RAM
      // Address 0 to 511
      .INIT_00(256'h00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00),
      .INIT_01(256'h00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00),
      .INIT_02(256'h00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00),
      .INIT_03(256'h00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00),
      .INIT_04(256'h00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00),
      .INIT_05(256'h00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00),
      .INIT_06(256'h00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00),
```

Structural Template

```
    // The next set of INITP_xx are for the parity bits
    // Address 0 to 511
    .INITP_00(256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INITP_01(256'h0000000000000000000000000000000000000000000000000000000000000000),
    // Address 512 to 1023
    .INITP_02(256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INITP_03(256'h0000000000000000000000000000000000000000000000000000000000000000),
    // Address 1024 to 1535
    .INITP_04(256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INITP_05(256'h0000000000000000000000000000000000000000000000000000000000000000),
    // Address 1536 to 2047
    .INITP_06(256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INITP_07(256'h0000000000000000000000000000000000000000000000000000000000000000)
) RAMB16_S9_S9_inst (
    .DOA(DOA),      // Port A 8-bit Data Output
    .DOB(DOB),      // Port B 8-bit Data Output
    .DOPA(DOPA),    // Port A 1-bit Parity Output
    .DOPB(DOPB),    // Port B 1-bit Parity Output
    .ADDRA(ADDRA),  // Port A 11-bit Address Input
    .ADDRB(ADDRB),  // Port B 11-bit Address Input
    .CLKA(CLKA),    // Port A Clock
    .CLKB(CLKB),    // Port B Clock
    .DIA(DIA),      // Port A 8-bit Data Input
    .DIB(DIB),      // Port B 8-bit Data Input
    .DIPA(DIPA),    // Port A 1-bit parity Input
    .DIPB(DIPB),    // Port-B 1-bit parity Input
    .ENA(ENA),      // Port A RAM Enable Input
    .ENB(ENB),      // Port B RAM Enable Input
    .SSRA(SSRA),    // Port A Synchronous Set/Reset Input
    .SSRB(SSRB),    // Port B Synchronous Set/Reset Input
    .WEA(WEA),      // Port A Write Enable Input
    .WEB(WEB)       // Port B Write Enable Input
);

// End of RAMB16_S9_S9_inst instantiation
```

Distributed RAM

Table 6-4: **Single-Port and Dual-Port Distributed RAMs**

Primitive	RAM Size (Depth x Width)	Type	Address Inputs
RAM16X1S	16 x 1	Single-port	A3, A2, A1, A0
RAM32X1S	32 x 1	Single-port	A4, A3, A2, A1, A0
RAM64X1S	64 x 1	Single-port	A5, A4, A3, A2, A1, A0
RAM16X1D	16 x 1	Dual-port	A3, A2, A1, A0

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

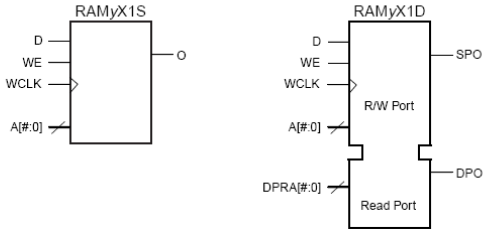
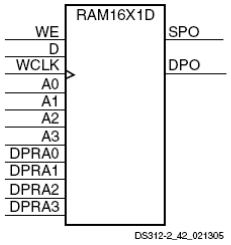


Table 6-5: **Dual-Port RAM Function**

Inputs			Outputs	
WE (mode)	WCLK	D	SPO	DPO
0 (read)	X	X	data_a	data_d
1 (read)	0	X	data_a	data_d
1 (read)	1	X	data_a	data_d
1 (write)	↑	D	D	data_d
1 (read)	↓	X	data_a	data_d

Distributed RAM



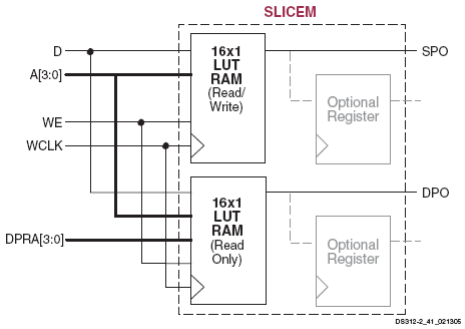


Figure 26: **RAM16X1D Dual-Port Usage**

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

```
RAM16X1D #(
   .INIT(16'h0000) // Initial contents of RAM
) RAM16X1D_inst (
   .DPO(DPO),     // Read-only 1-bit data output for DPRA
   .SPO(SPO),     // R/W 1-bit data output for A0-A3
   .A0(A0),       // R/W address[0] input bit
   .A1(A1),       // R/W address[1] input bit
   .A2(A2),       // R/W address[2] input bit
   .A3(A3),       // R/W address[3] input bit
   .D(D),         // Write 1-bit data input
   .DPRA0(DPRA0), // Read address[0] input bit
   .DPRA1(DPRA1), // Read address[1] input bit
   .DPRA2(DPRA2), // Read address[2] input bit
   .DPRA3(DPRA3), // Read address[3] input bit
   .WCLK(WCLK),   // Write clock input
   .WE(WE)        // Write enable input
);
```

Dual-Port Distributed RAM

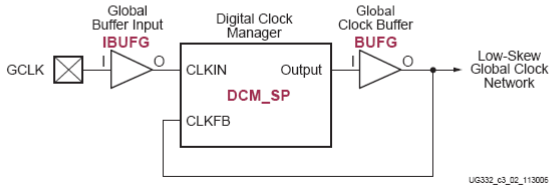
Digital Clock Manager (DCM)

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**.

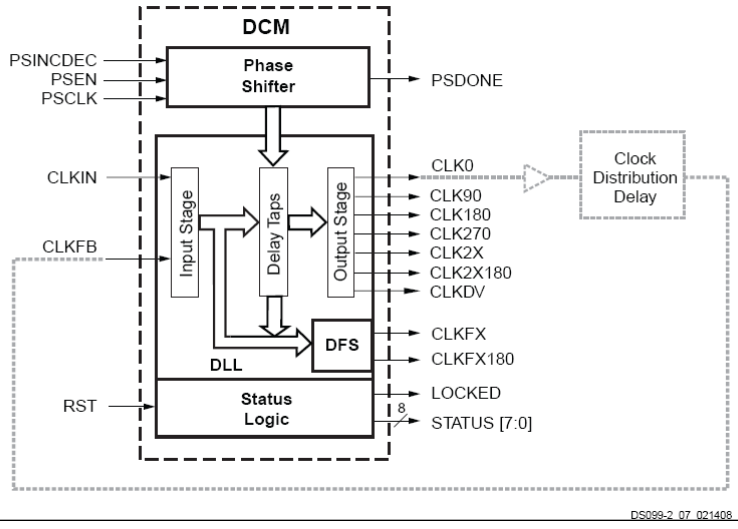
Digital Clock Manager (DCM)

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

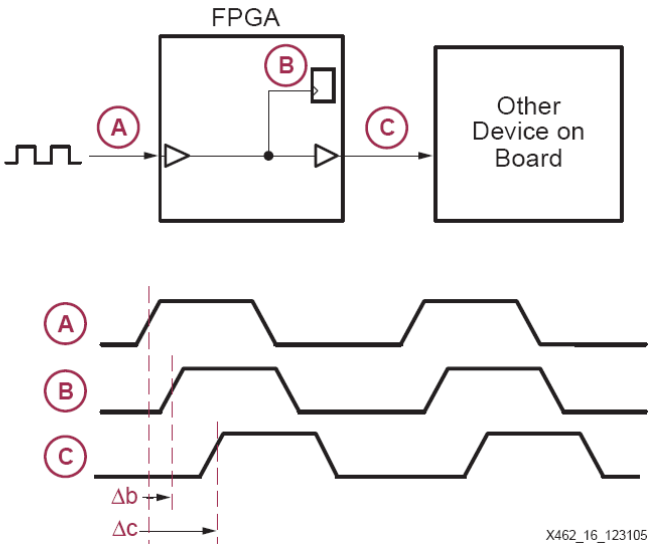


b. A Digital Clock Manager (DCM) Inserts Directly into the Global Clock Path

Digital Clock Manager (DCM)



Clock Skew



Clock Skew

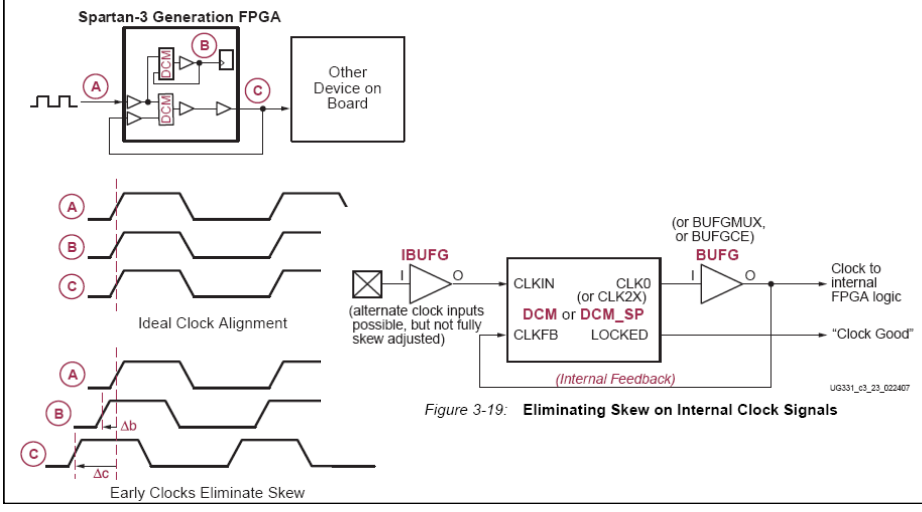


Figure 3-19: **Eliminating Skew on Internal Clock Signals**

Multipliers

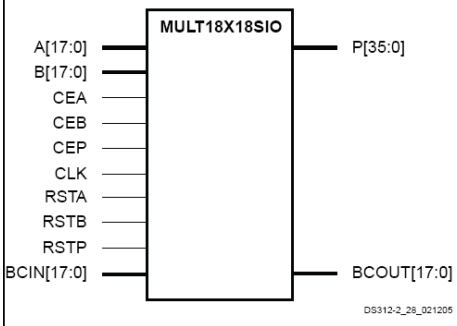


Figure 11-1: **MULT18X18SIO Primitive**

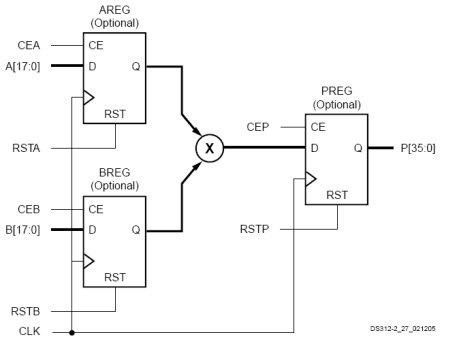


Figure 11-3: **Principle Ports and Functions of Dedicated Multiplier Blocks**

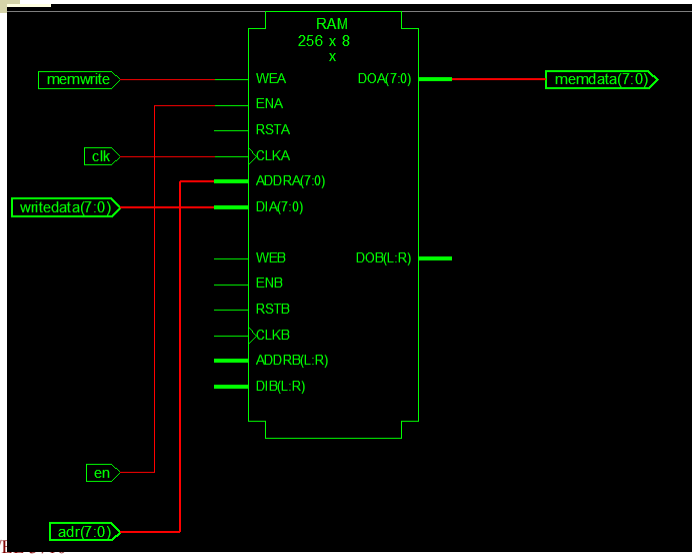
Multipliers

```
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
```

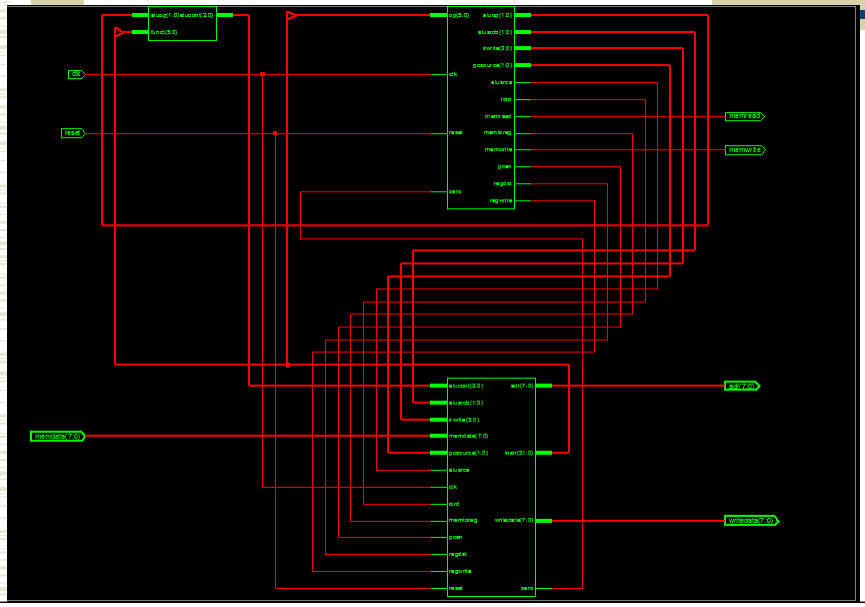
Synthesis Output (mips example)

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	79	9,312	1%
Number of 4 input LUTs	193	9,312	2%
Logic Distribution			
Number of occupied Slices	123	4,656	2%
Number of Slices containing only related logic	123	123	100%
Number of Slices containing unrelated logic	0	123	0%
Total Number of 4 input LUTs	193	9,312	2%
Number used as logic	161		
Number used for Dual Port RAMs	32		
Number of bonded IOBs			
Number of bonded	19	232	8%
Number of RAMB16s	1	20	5%
Number of BUFGMUXs	1	24	4%

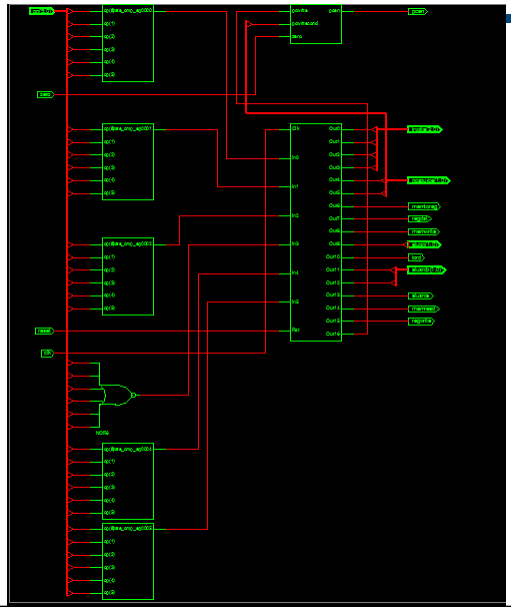
Synthesis Output (mips example)



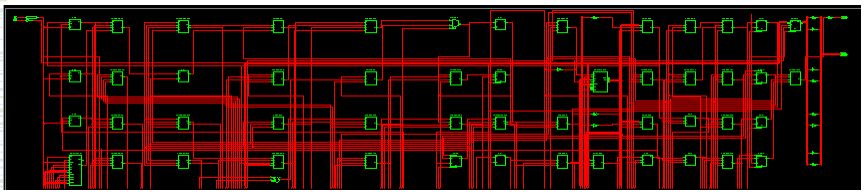
Synthesis Output (mips example)



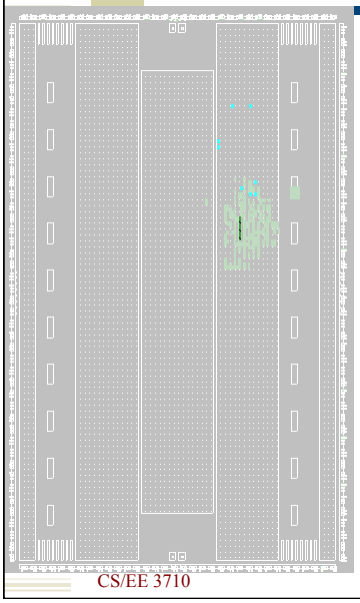
Synthesis Output (mips example)

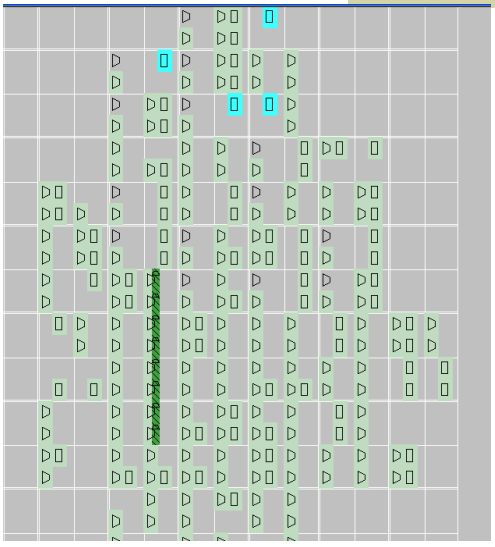


Synthesis Output (mips example)

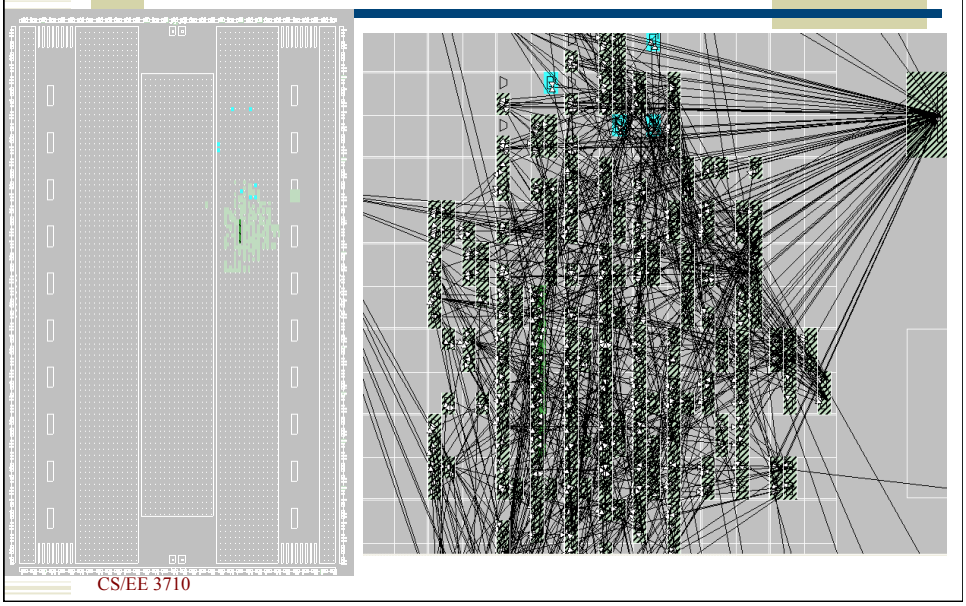


Implement Output (mips example)

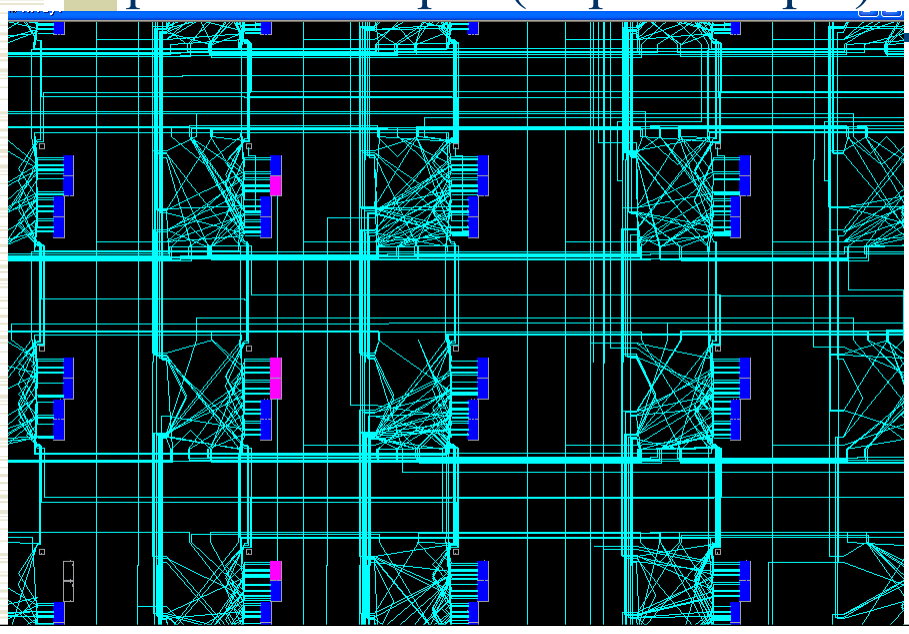




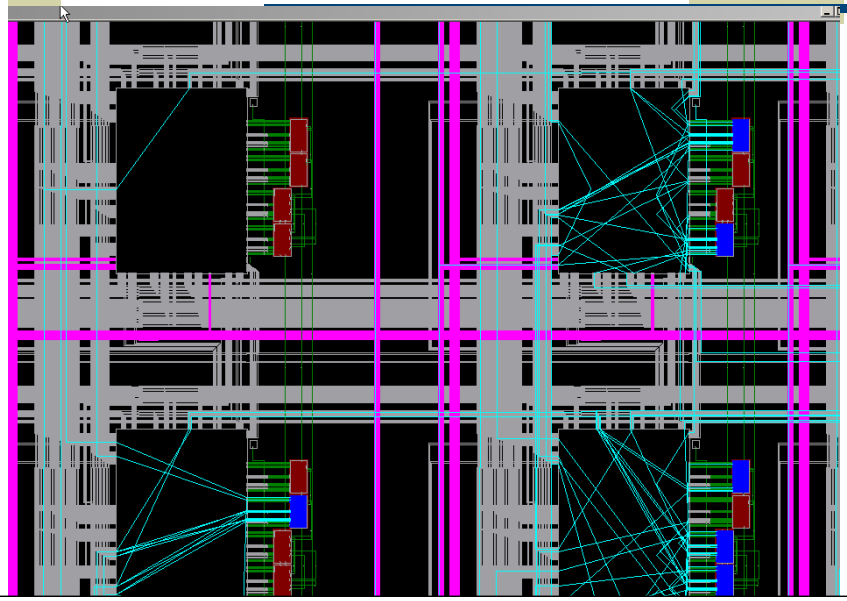
Implement Output (mips example)



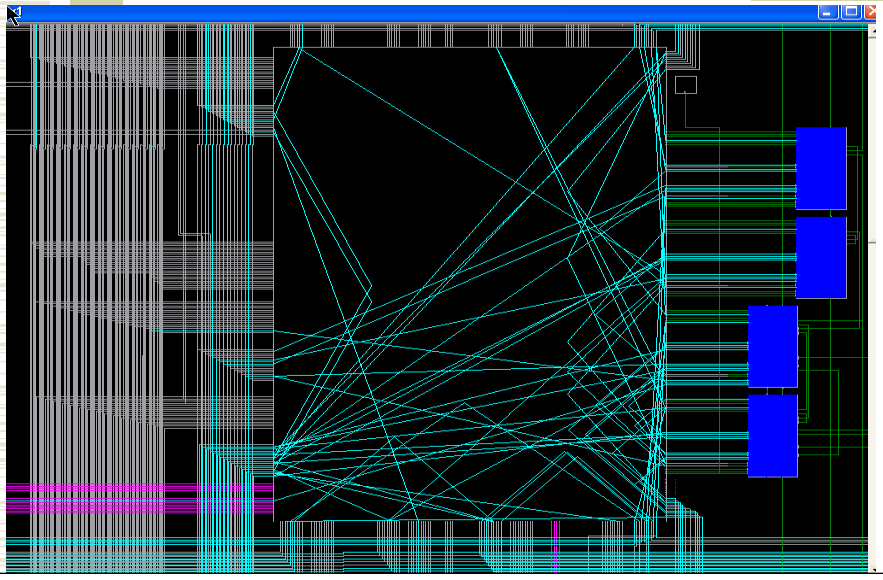
Implement Output (mips example)



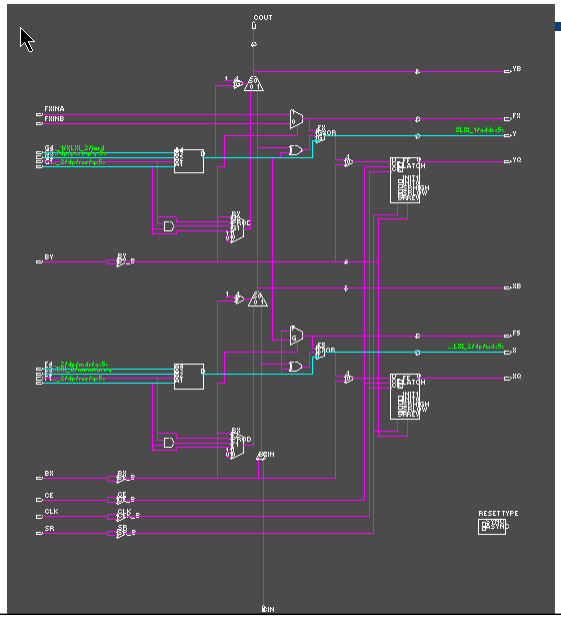
Implement Output (mips example)



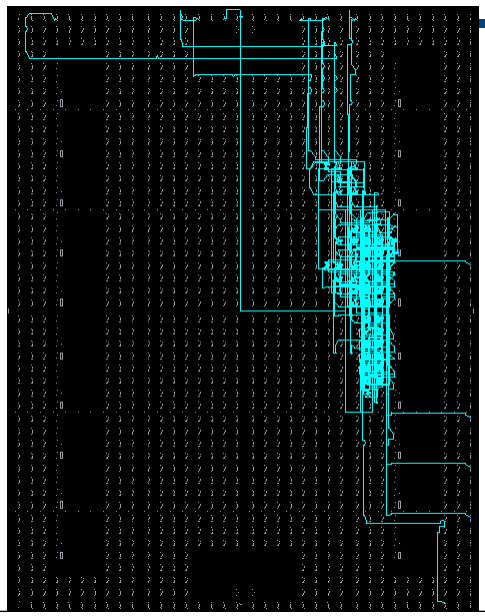
Implement Output (mips example)



Implement 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

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