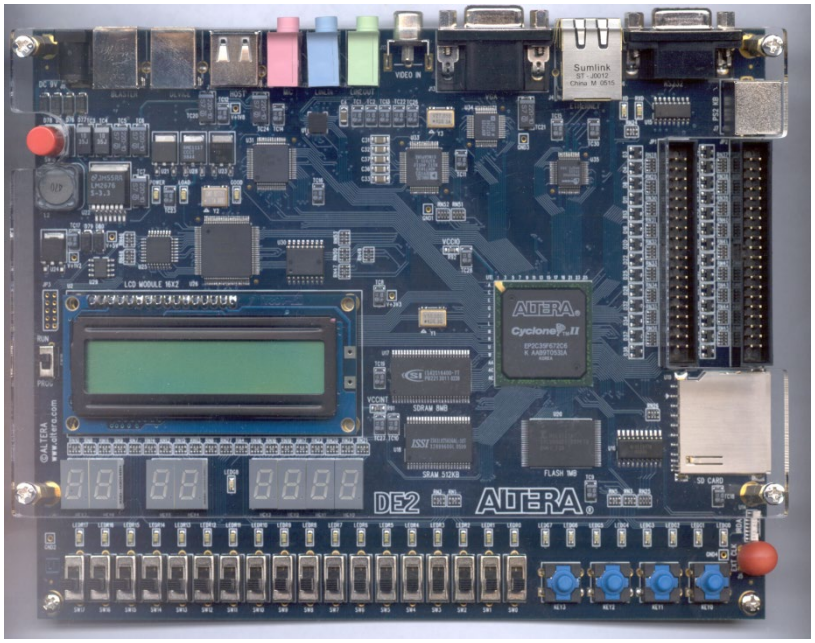
CPE 322
Digital Hardware Design Fundamentals

Electrical and Computer Engineering
UAH

Designing With FPGAs Architectural Issues and Features



Designing with FPGAs

Topics to be covered:

- Review of Generic FPGA Features
- Tradeoffs arising from the structure of the basic FPGA building blocks – some hand mapped examples
- Large Boolean Function Decomposition – Shannon's Expansion
- FPGA Support for commonplace operations – Carry and Cascade Chains
- Dedicated Memory and Multipliers in FPGAs

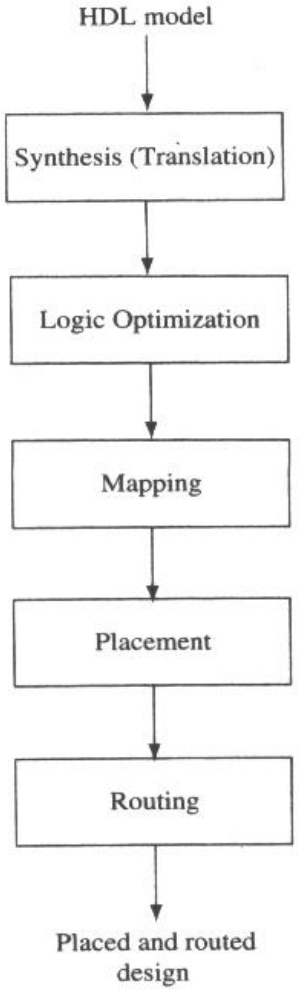
Chapter 6: Designing with FPGAs

Topics to be covered:

- Example Real-world FPGA Logic Blocks
- Cost of Reconfigurability
- FPGAs and One-Hot State Assignment
- Capacity Metrics; Maximum Gates versus Usable Gates
- FPGA Design Flow – modeling, synthesis, optimization, mapping, place and route

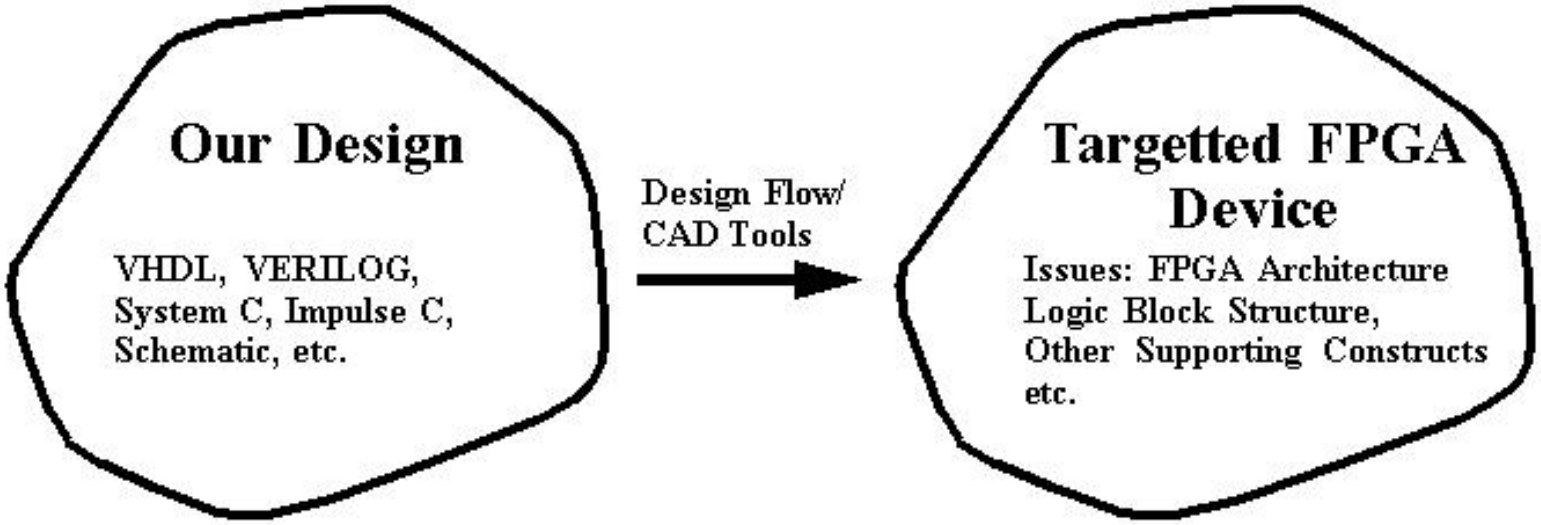
Implementing Functions in FPGAs

Partitioning a Design in an FPGA



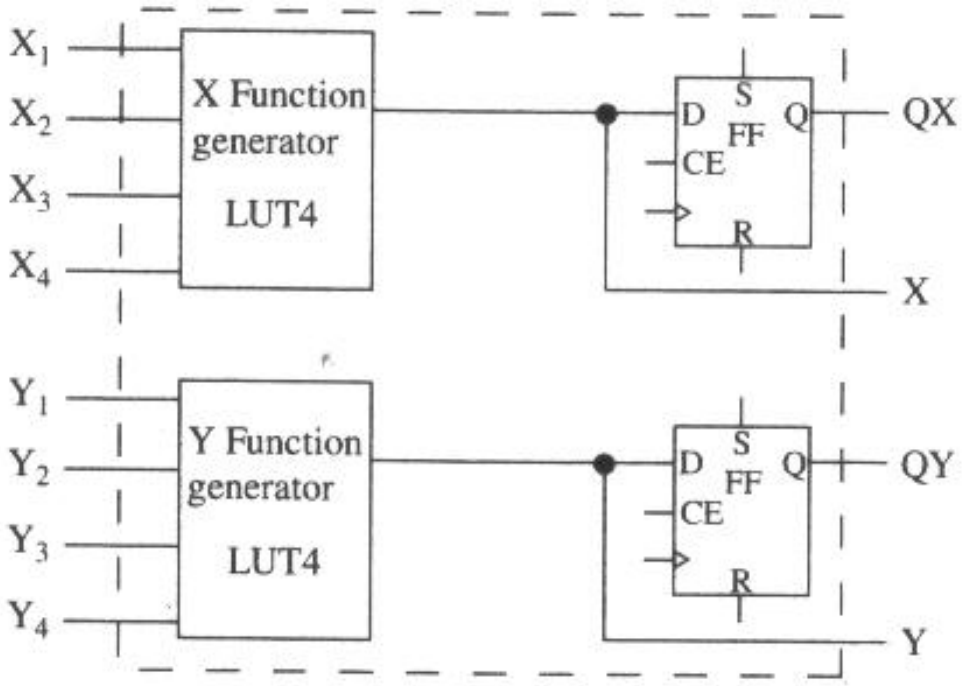
Implementing Functions in FPGAs

Partitioning a Design in an FPGA



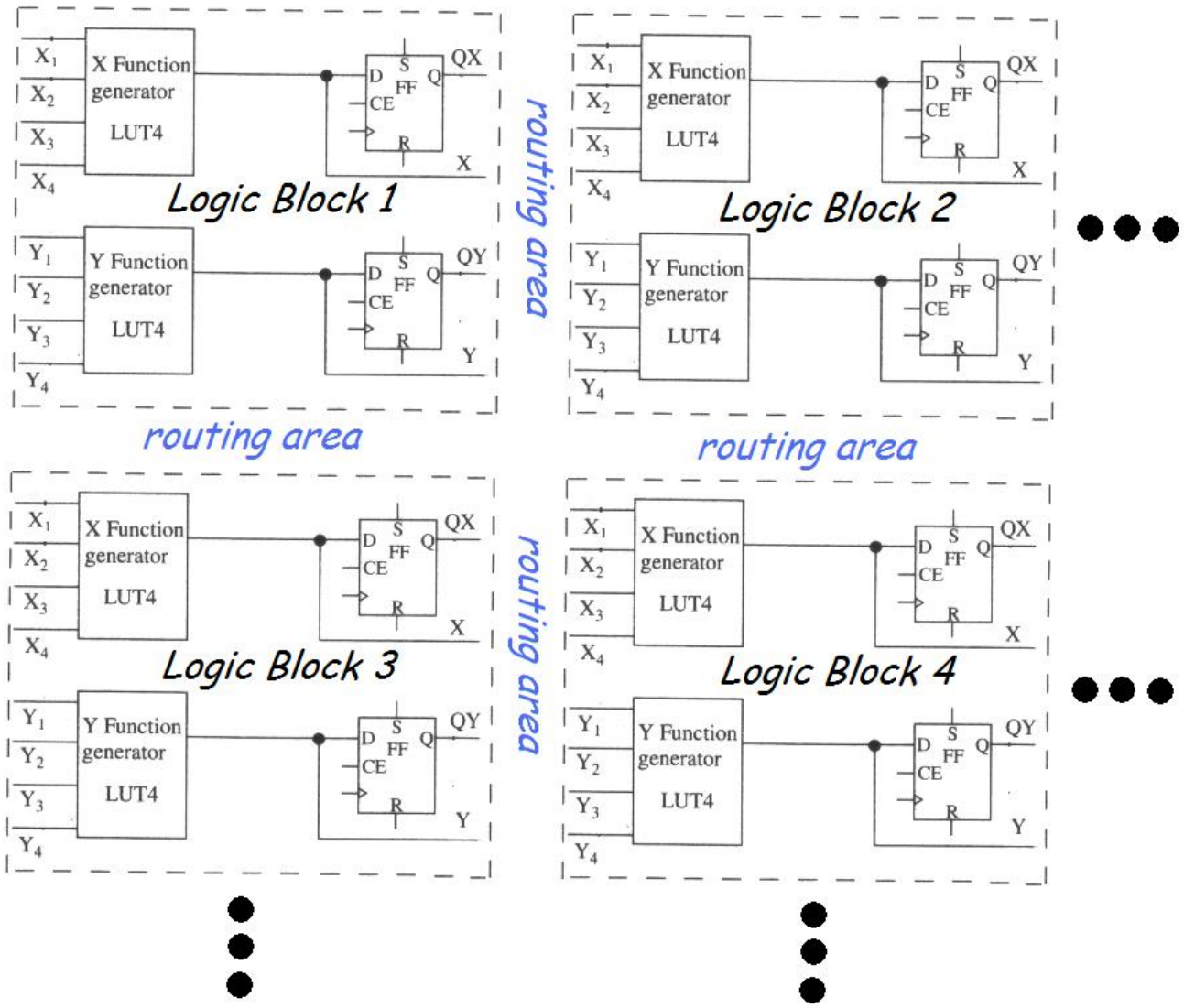
Implementing a Desired Function in an FPGA – Hand-mapped 4-to-1 MUX example

- Desired Function:
  - 4-to-1 Multiplexer
- Targeted FPGA Logic Block



(a)

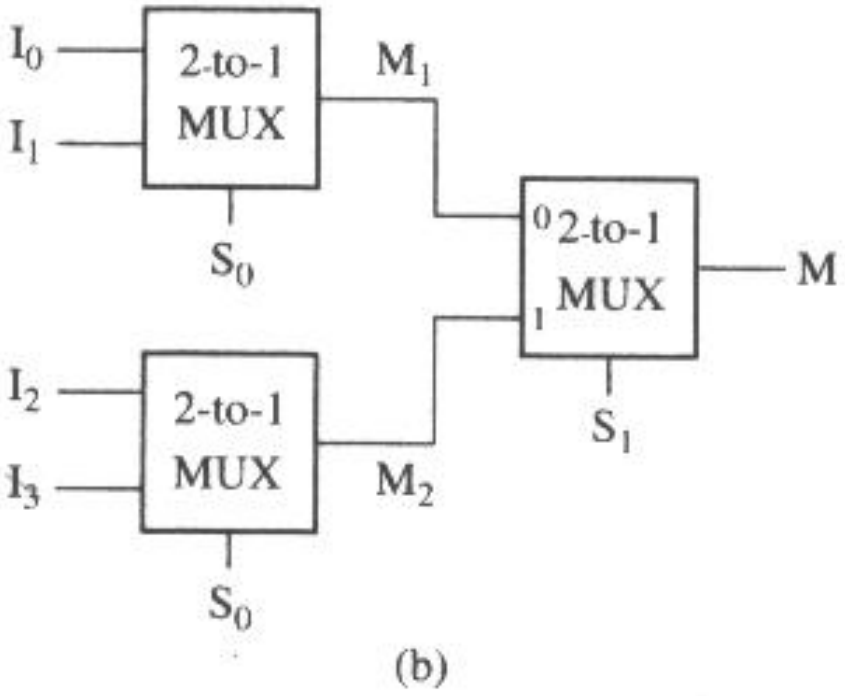
Implementing a Desired Function in an FPGA – Hand-mapped 4-to-1 MUX example



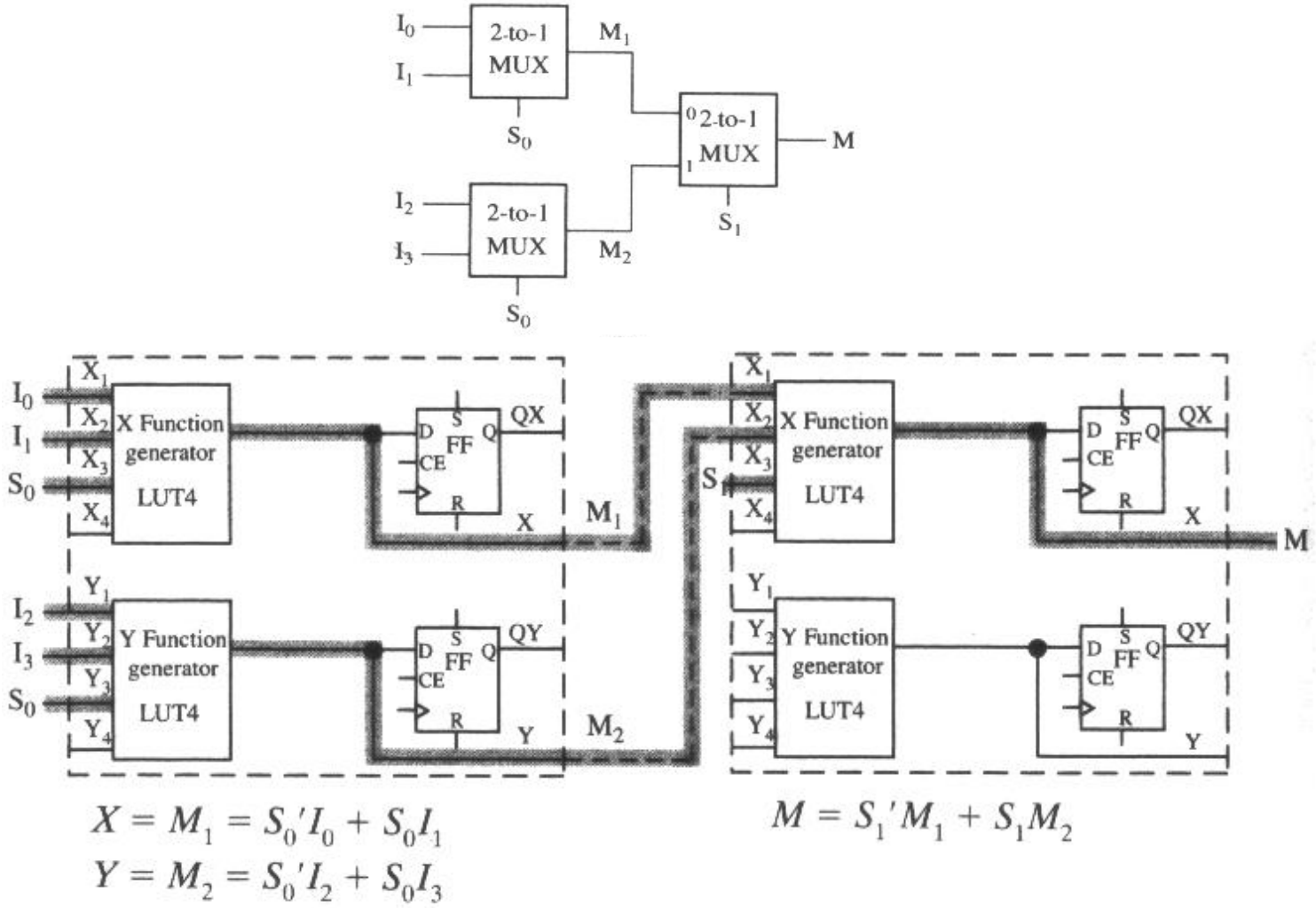
Implementing a Desired Function in an FPGA – Hand-mapped 4-to-1 MUX example

- 4-to-1 Multiplexer
  - $M = S'_1S'_0I_0 + S'_1S_0I_1 + S_1S'_0I_2 + S_1S_0I_3$ ***(6 variables – will not fit in LUT4)***
- 4-to-1 MUX can be decomposed into 3 2-to-1 MUXs

$$M_1 = S_0'I_0 + S_0I_1$$
$$M_2 = S_0'I_2 + S_0I_3$$
$$M = S_1'M_1 + S_1M_2$$



Implementing a Desired Function in an FPGA – Hand-mapped 4-to-1 MUX example

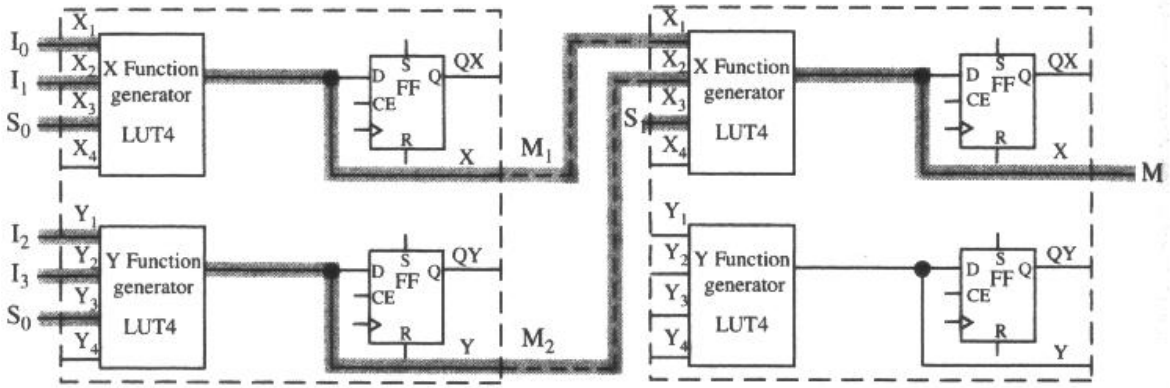


$$X = M_1 = S_0'I_0 + S_0I_1$$

$$Y = M_2 = S_0'I_2 + S_0I_3$$

$$M = S_1'M_1 + S_1M_2$$

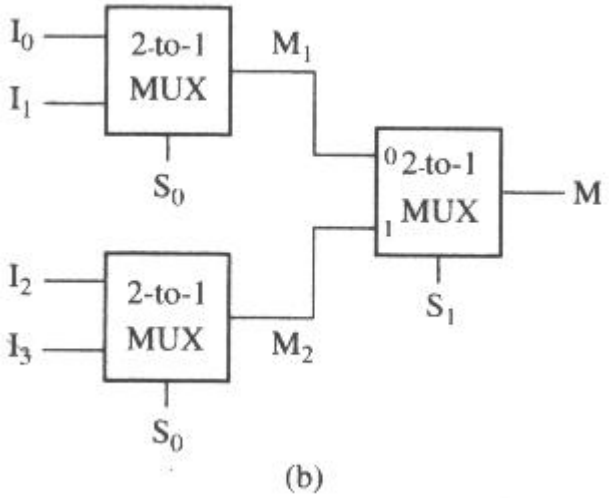
Implementing a Desired Function in an FPGA – Hand-mapped 4-to-1 MUX example



$X = M_1 = S_0'I_0 + S_0I_1$

$Y = M_2 = S_0'I_2 + S_0I_3$

$M = S_1'M_1 + S_1M_2$



Inputs				Output
X_4	$X_3(S_0)$	$X_2(I_1)$	$X_1(I_0)$	X
x	0	0	0	0
x	0	0	1	1
x	0	1	0	0
x	0	1	1	1
x	1	0	0	0
x	1	0	1	0
x	1	1	0	1
x	1	1	1	1

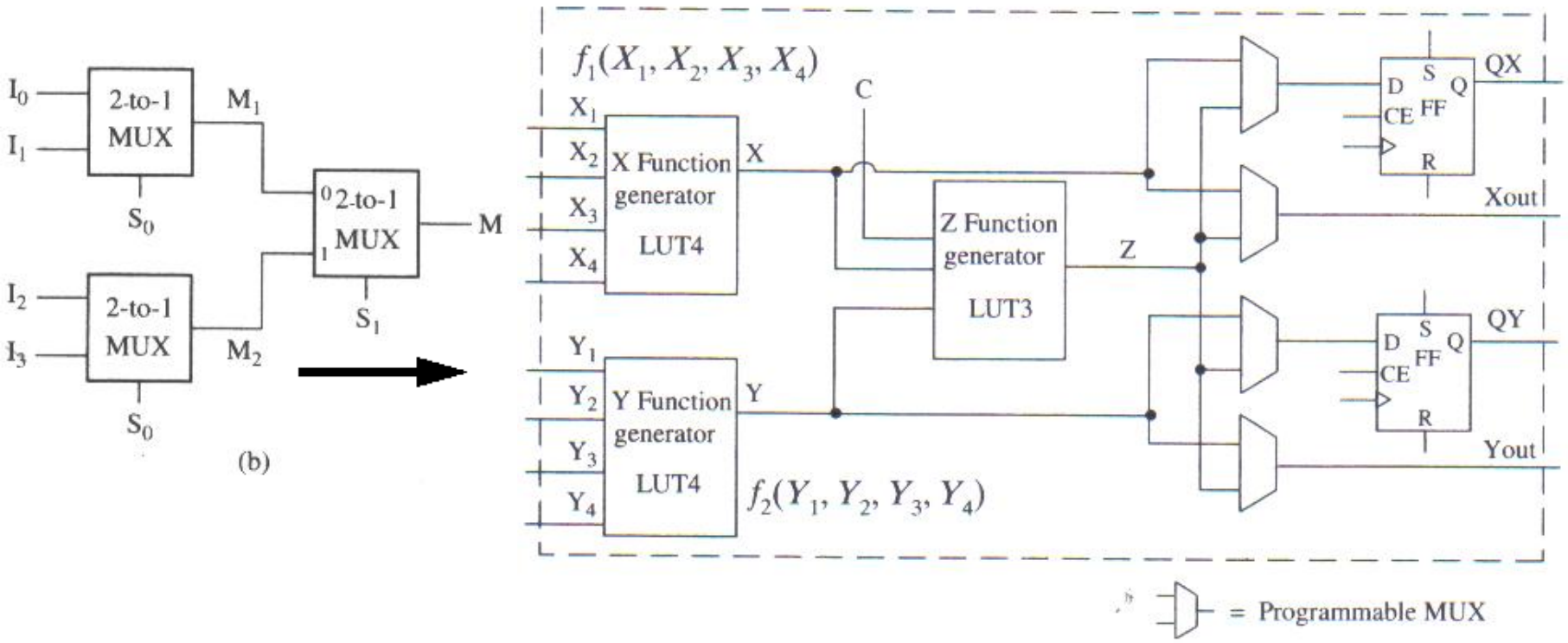
LUT-M1 – 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1

LUT-M2 – 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1

LUT-M – 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1

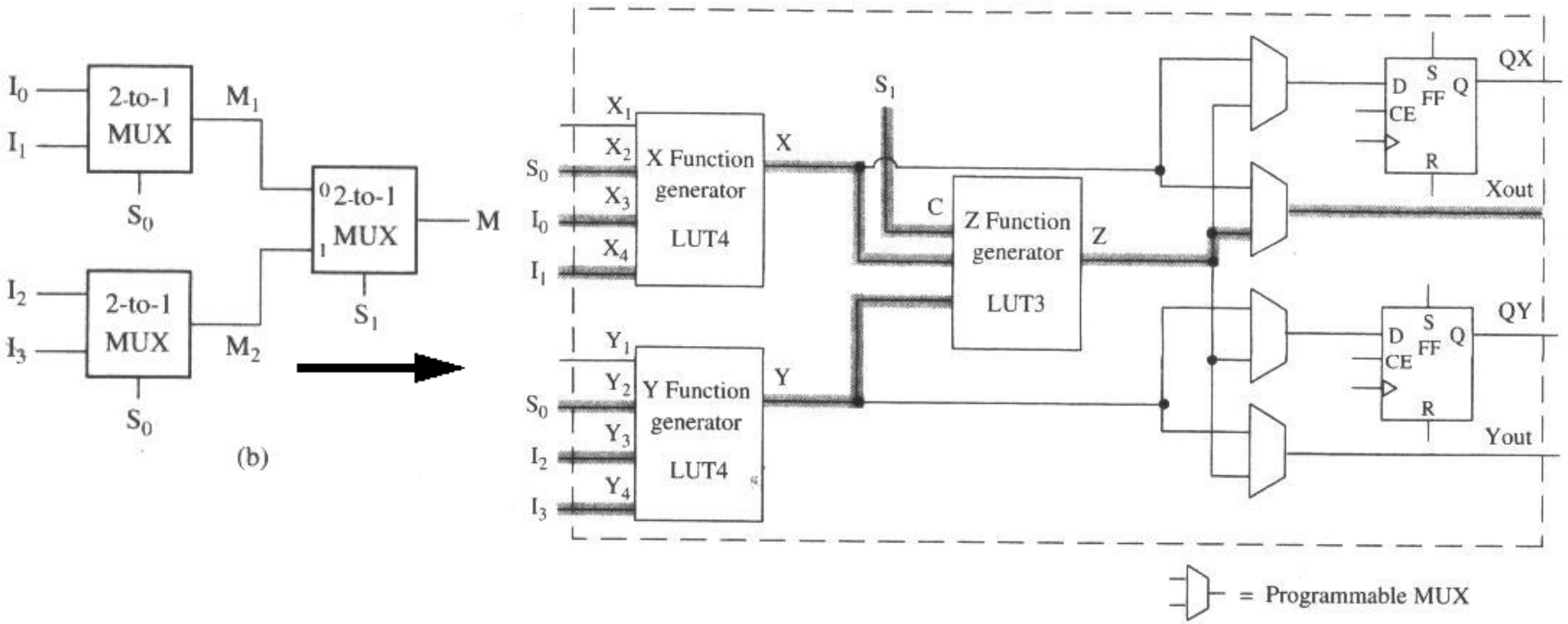
Implementing a Desired Function in an FPGA – Hand-mapped 4-to-1 MUX example

Another FPGA might have a different Logic Block Configuration – for example:



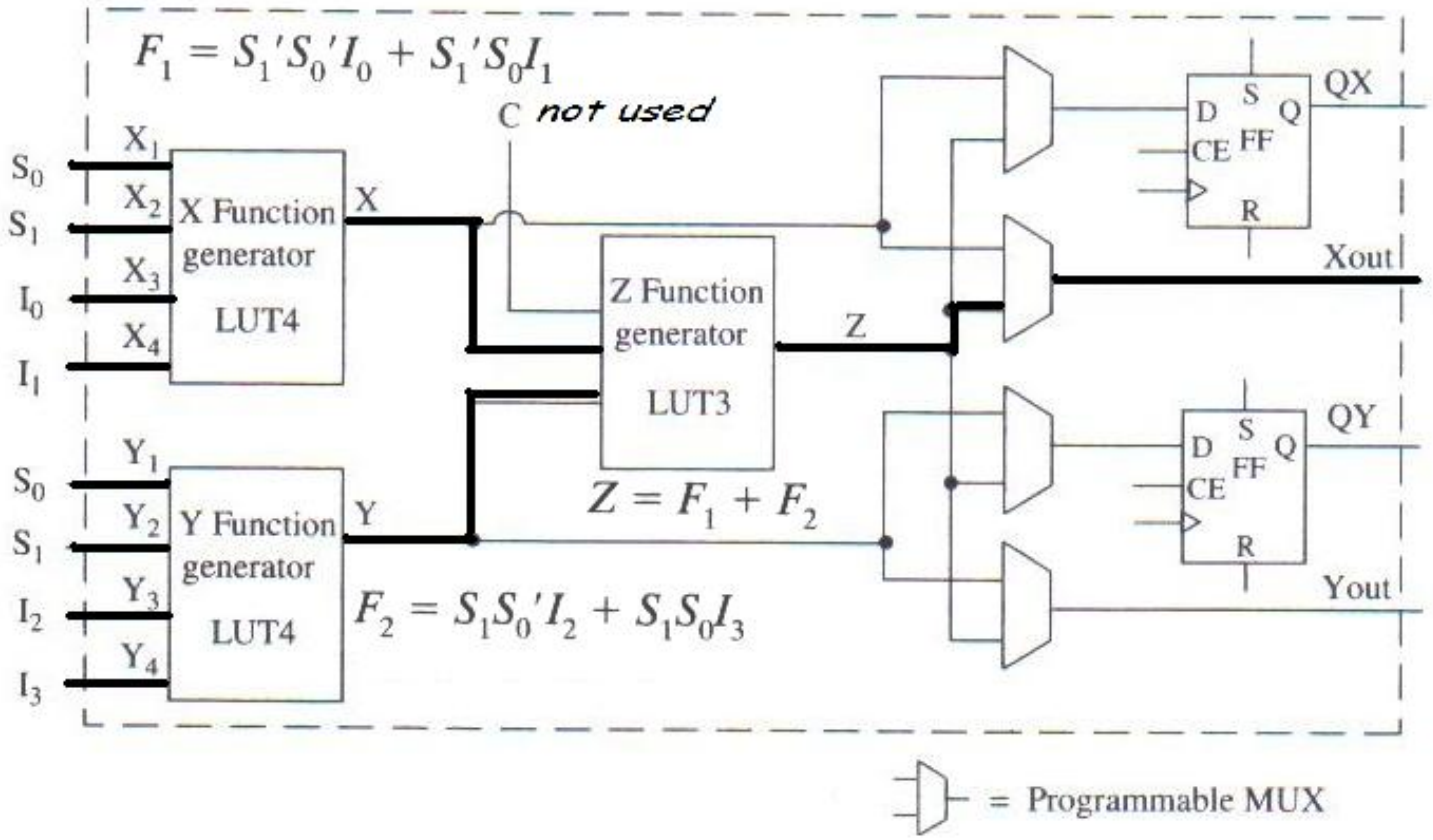
Implementing a Desired Function in an FPGA – Hand-mapped 4-to-1 MUX example

Another FPGA might have a different Logic Block Configuration – for example:

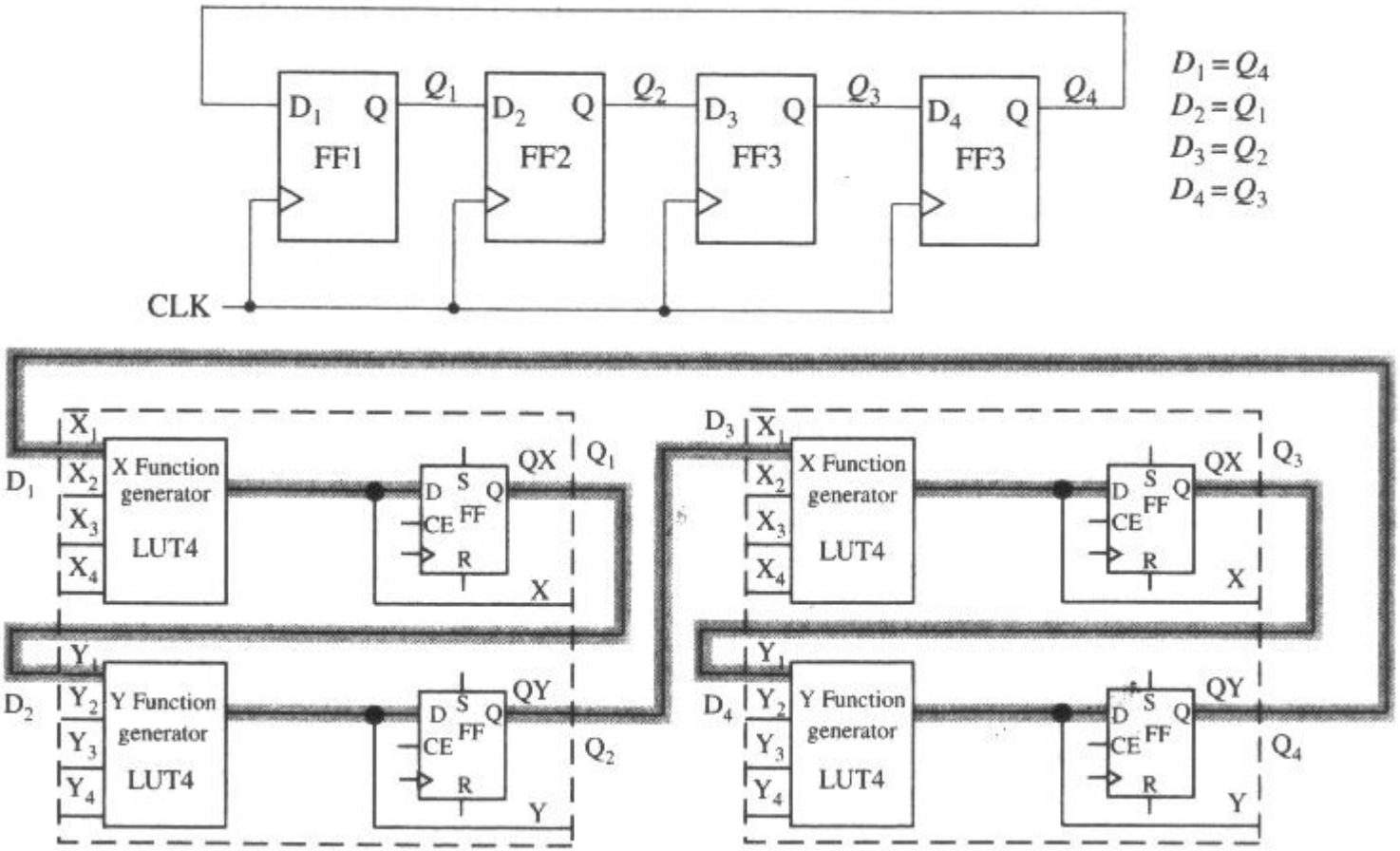


Implementing a Desired Function in an FPGA – Hand-mapped 4-to-1 MUX example

There can also be multiple ways to implement a Function within the same FPGA. Consider this equally valid representation of a 4-to-1 MUX!



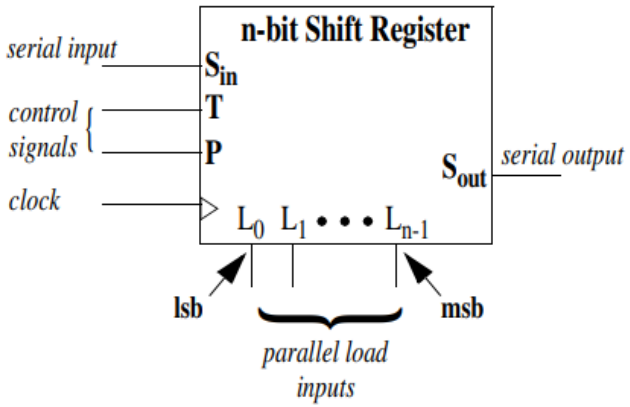
Implementing a Desired Function in an FPGA Hand-mapped ring counter example



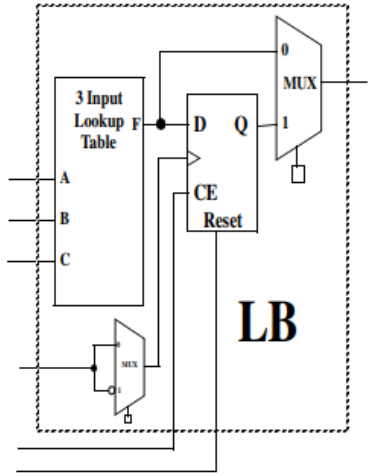
Note: function generator LUT4s largely unused!!
(wastes 64 SRAM cells)

Example Exam 2 Question:

An multiple bit right shift register with parallel load and a single serial output is to be implemented within an FPGA that has a standard island style architecture where the basic logic blocks have the configurations shown below:



Shift Register Module to be Designed

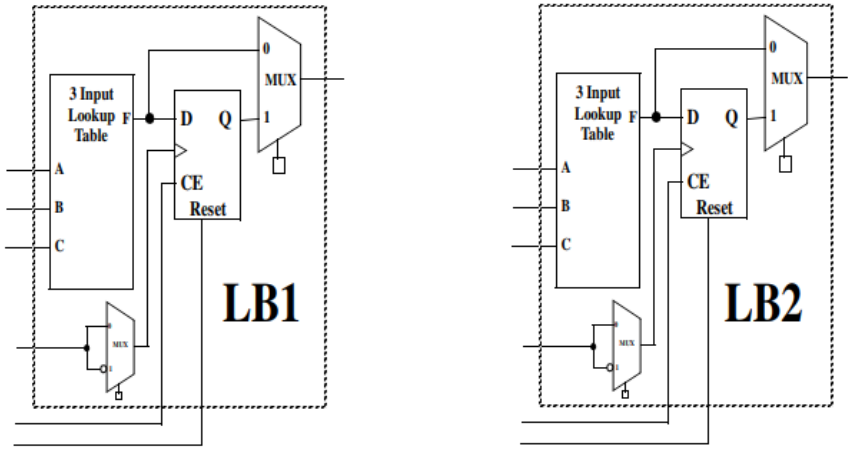


Internal Logic Block Architecture of Targeted FPGA

The Control Signals P and T operate as follows: P='0', do nothing keep current state; P = '1' and T='1' right shift on next rising edge clock; P='1' and T='0' load values from parallel load inputs into shift register on the next rising clock edge.

Example Exam 2 Question:

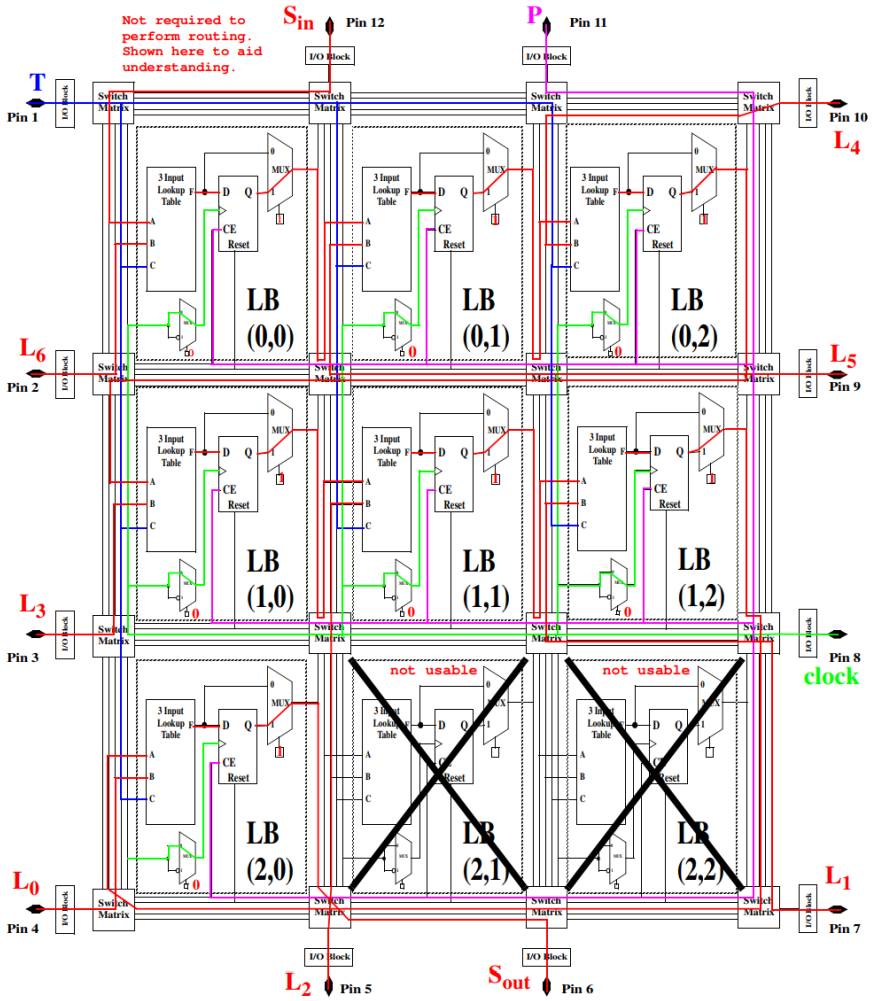
a) On the figure below create a two-bit shift register using these logic blocks as basic building elements of your design. Show the necessary internal connections between subcomponents as well as the logical connections between the logic blocks themselves. Also label signals that enter and leave the logic blocks that must originate or terminate on external FPGA I/O pin locations. Such signals include the serial input, control signals, parallel load inputs and the serial output. Use the names shown on the figure.



b) What are the logic equations that should be implemented by the lookup tables in the two logic blocks in part (a) of this problem.

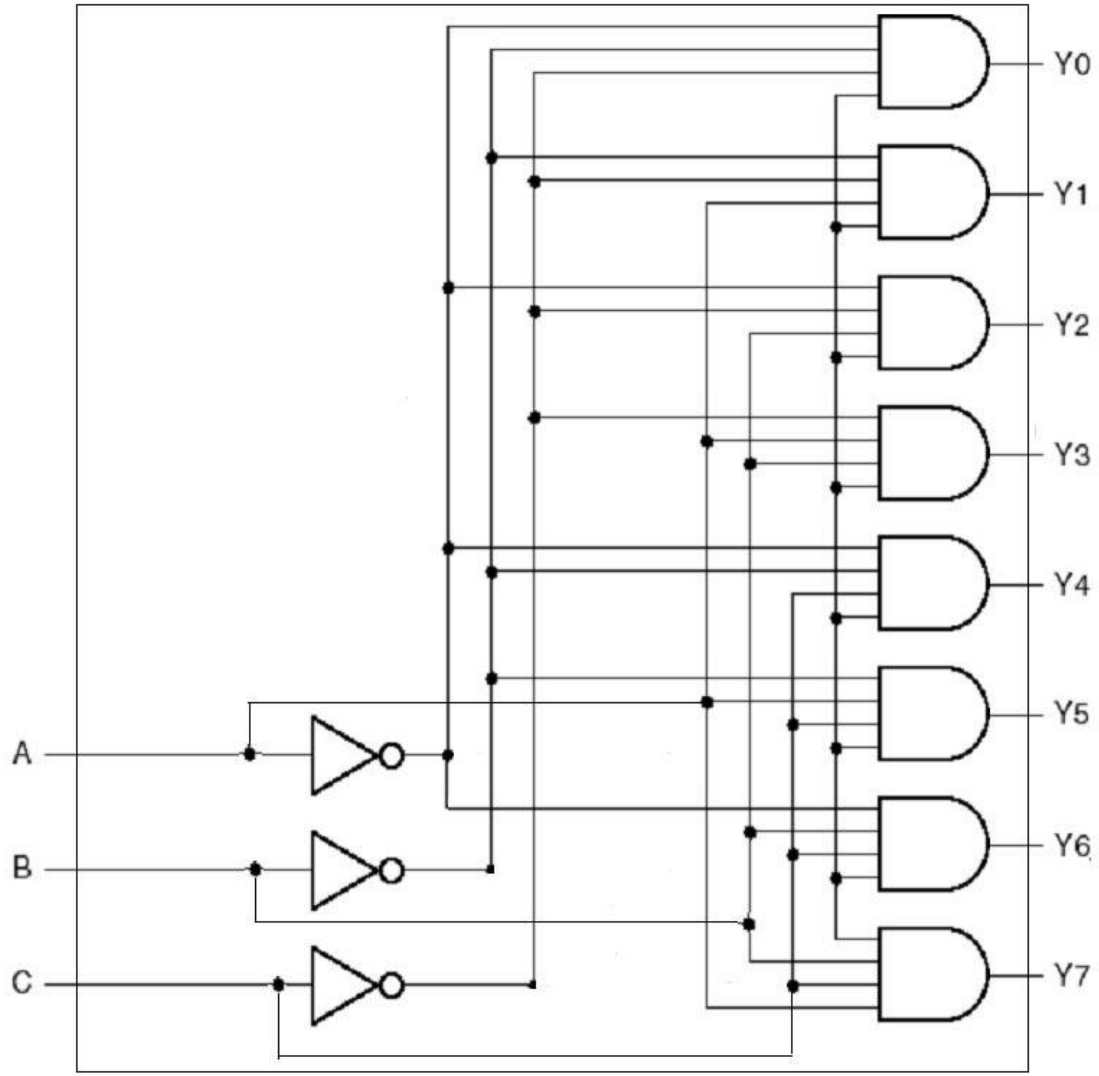
c) What is the maximum value of n for the n-bit shift register that would fit in the very small FPGA, whose floor plan is shown in the figure on the next page could support? What resource makes this value of n the limiting factor?

Example Exam 2 Question:

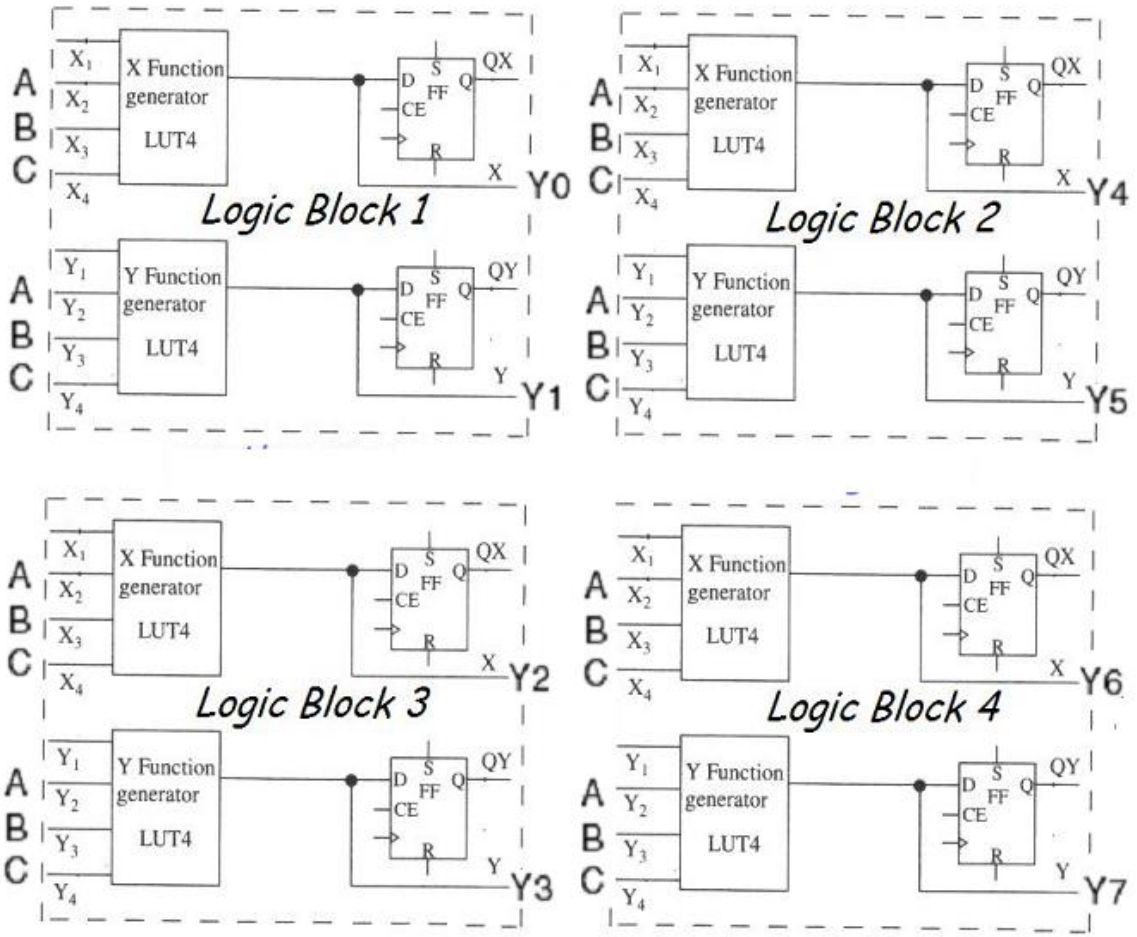


Floor-plan of Targeted island-style FPGA Device

Implementing a Desired Function in an FPGA Hand-mapped 3-to-8 decoder



Implementing a Desired Function in an FPGA Hand-mapped 3-to-8 decoder



Note: Very Expensive Implementation

(uses 128 SRAM cells – 4 Logic Blocks – 8 FF wasted)

Shannon's Decomposition

$$Z(a, b, c, d, e, f) = a' \cdot Z(0, b, c, d, e, f) + a \cdot Z(1, b, c, d, e, f) = a' Z_0 + a Z_1$$

This expansion can be continued for any number of variables

Example: $Z = abcd'ef' + a'b'c'def' + b'cde'f$

Setting $a = 0$ gives

$$Z_0 = 0 \cdot bcd'ef' + 1 \cdot b'c'def' + b'cde'f = b'c'def' + b'cde'f$$

and setting $a = 1$ gives

$$Z_1 = 1 \cdot bcd'ef' + 0 \cdot b'c'def' + b'cde'f = bcd'ef' + b'cde'f.$$

$$Z = a' Z_0 + a Z_1$$

Shannon's Decomposition
(6 variable function to two 5 variable functions)

Example: $Z = abcd'ef' + a'b'c'def' + b'cde'f$

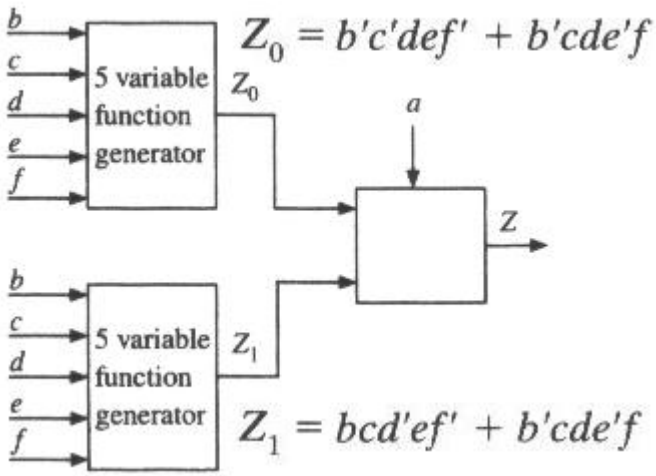
Setting $a = 0$ gives

$$Z_0 = 0 \cdot bcd'ef' + 1 \cdot b'c'def' + b'cde'f = b'c'def' + b'cde'f$$

and setting $a = 1$ gives

$$Z_1 = 1 \cdot bcd'ef' + 0 \cdot b'c'def' + b'cde'f = bcd'ef' + b'cde'f.$$

$$Z = a'Z_0 + aZ_1$$

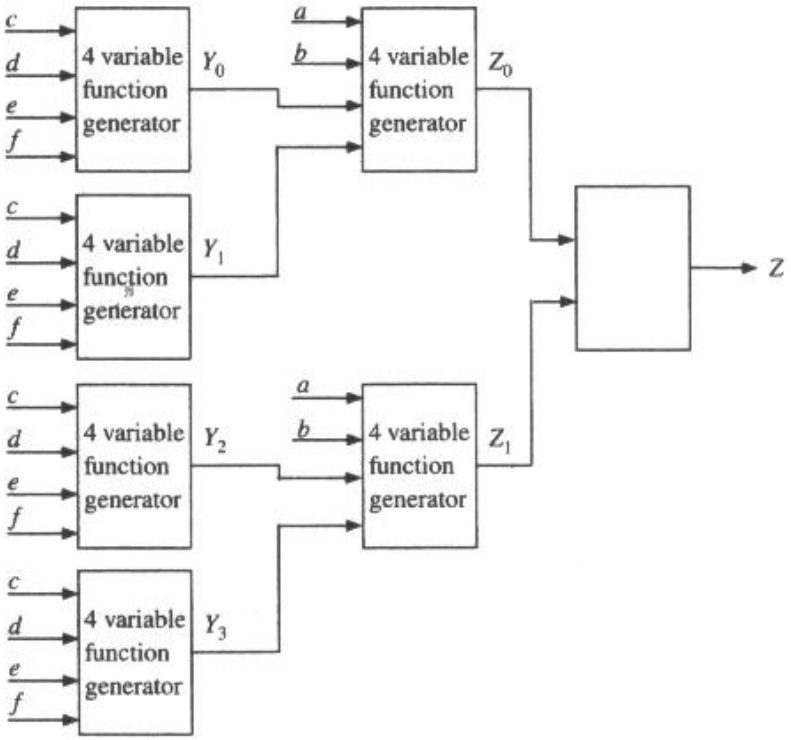


Shannon's Decomposition

(6 variable function to six 4 variable functions)

Example: $Z = abcd'ef' + a'b'c'def' + b'cde'f$

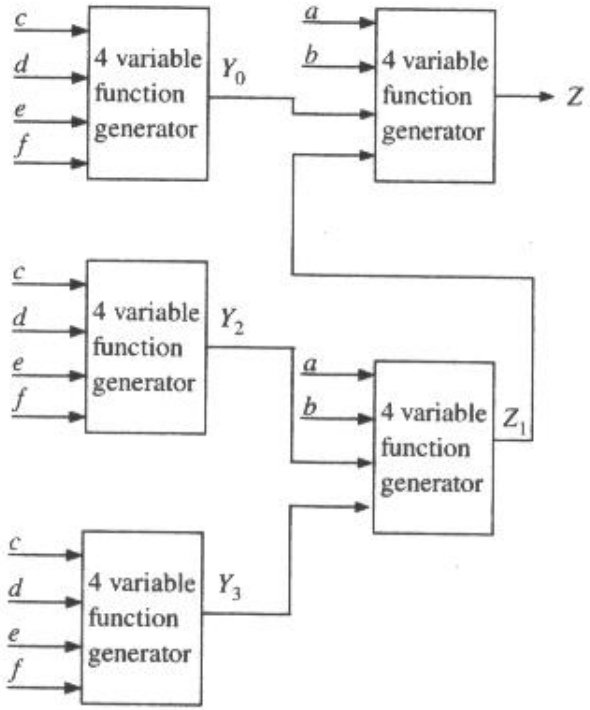
$$Z(a,b,c,d,e,f) = a'b' \cdot Z(0,0,c,d,e,f) + a'b \cdot Z(0,1,c,d,e,f) + ab' \cdot Z(1,0,c,d,e,f) + ab \cdot Z(1,1,c,d,e,f)$$
$$= a'b' \cdot Y_0 + a'b \cdot Y_1 + ab' \cdot Y_2 + ab \cdot Y_3$$



Shannon's Decomposition
(6 variable function to five 4 variable functions)

Example 2: $Z = abcd'ef' + a'b'c'def' + b'cde'f$

- Substituting $a = b = 0$ gives $Y_0 = c'def' + cde'f$
- Substituting $a = 0, b = 1$ gives $Y_1 = 0$
- Substituting $a = 1, b = 0$ gives $Y_2 = cde'f,$
- Substituting $a = b = 1$ gives $Y_3 = cd'ef'$



$$Z = a'b' \cdot Y_0 + ab' \cdot Y_2 + ab \cdot Y_3$$

Possible because $Y_1 = 0$

Shannon's Decomposition
(7 variable function to six 5 variable functions)

$$\begin{aligned} Z(a,b,c,d,e,f,g) &= a'b' \cdot Z(0,0,c,d,e,f,g) + a'b \cdot Z(0,1,c,d,e,f,g) \\ &\quad + ab' \cdot Z(1,0,c,d,e,f,g) + ab \cdot Z(1,1,c,d,e,f,g) \\ &= a'b' \cdot Y_0 + a'b \cdot Y_1 + ab' \cdot Y_2 + ab \cdot Y_3 \end{aligned}$$

Example 3:

$$Z = c'de'fg + bcd'e'fg' + a'c'def'g + a'b'd'ef'g' + ab'defg'$$

- Substituting $a = b = 0$ gives $Y_0 = c'de'fg + c'def'g + d'ef'g'$
- Substituting $a = 0, b = 1$ gives $Y_1 = c'de'fg + cd'e'fg' + c'def'g$
- Substituting $a = 1, b = 0$ gives $Y_2 = c'de'fg + defg'$
- Substituting $a = b = 1$ gives $Y_3 = c'de'fg + cd'e'fg'$

$$Z_0 = a'b' \cdot Y_0 + a'b \cdot Y_1 \qquad Z = Z_0 + ab' \cdot Y_2 + ab \cdot Y_3$$

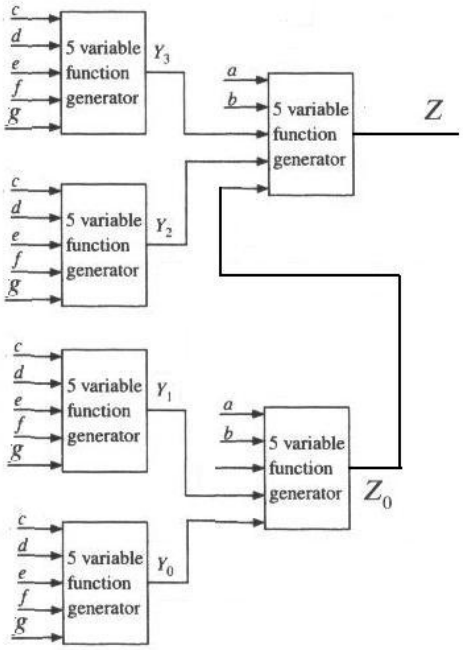
Shannon's Decomposition
(7 variable function to six 5 variable functions)

Example 3:

$$Z = c'de'fg + bcd'e'fg' + a'c'def'g + a'b'd'ef'g' + ab'defg'$$

- Substituting $a = b = 0$ gives $Y_0 = c'de'fg + c'def'g + d'ef'g'$
- Substituting $a = 0, b = 1$ gives $Y_1 = c'de'fg + cd'e'fg' + c'def'g$
- Substituting $a = 1, b = 0$ gives $Y_2 = c'de'fg + defg'$
- Substituting $a = b = 1$ gives $Y_3 = c'de'fg + cd'e'fg'$

$$Z_0 = a'b' \cdot Y_0 + a'b \cdot Y_1 \qquad Z = Z_0 + ab' \cdot Y_2 + ab \cdot Y_3$$



Any 7 Variable function can be implemented with six or fewer LUT5s!

Shannon's Decomposition
(Implementing 7 variable function using LUT4s w/ MUXES

Note: to Reduce the number of LUTs needed many FPGAs provide Multiplexers in addition to LUT4s.

Implement a seven-variable function using four-input LUTs and 2-to-1 multiplexers.

Shannon's expansion can be used to obtain the following decompositions:

7-variable function generator = two 6-variable function generators + a 2-to-1 mux . . . (i)

6-variable function generator = two 5-variable function generators + a 2-to-1 mux . . . (ii)

5 variable function generator = two 4-variable function generators + a 2-to-1 mux . . . (iii)

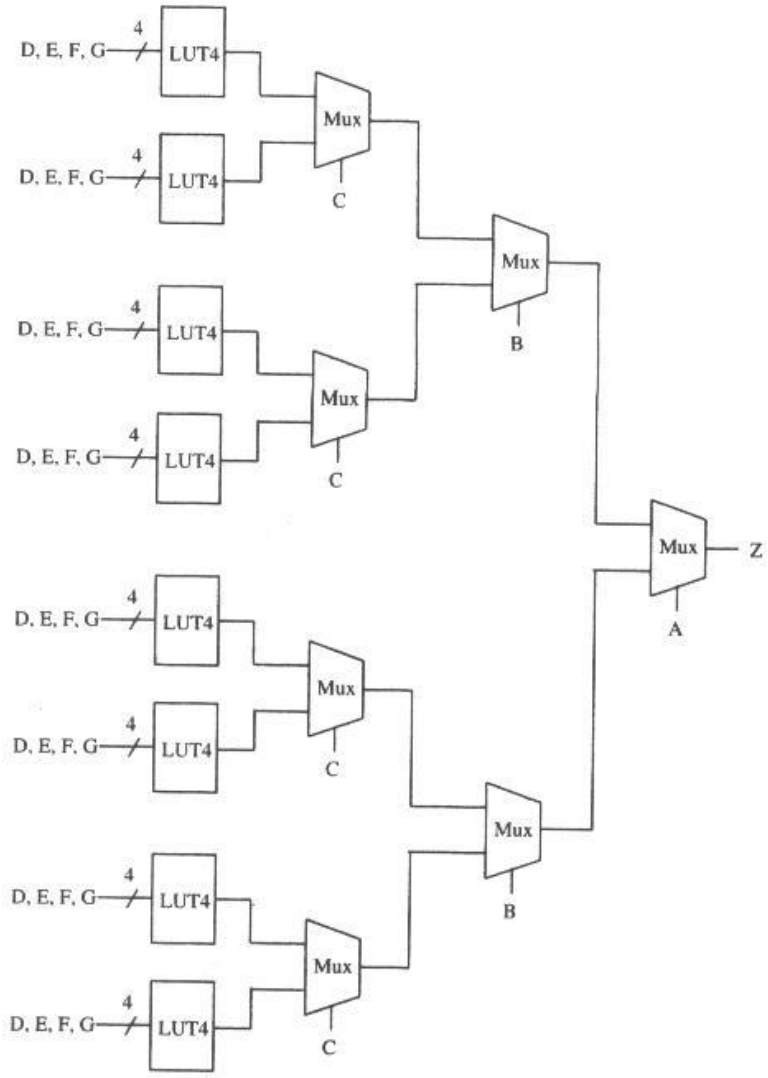
Substituting (iii) into (ii), we obtain

6-variable function generator = four 4-variable function generators + three 2-to-1 muxes . . . (iv)

Substituting (iv) into (i), we obtain

7-variable function generator = eight 4-variable function generators + seven 2-to-1 muxes

Shannon's Decomposition
(Implementing 7 variable function using LUT4s w/ MUXES

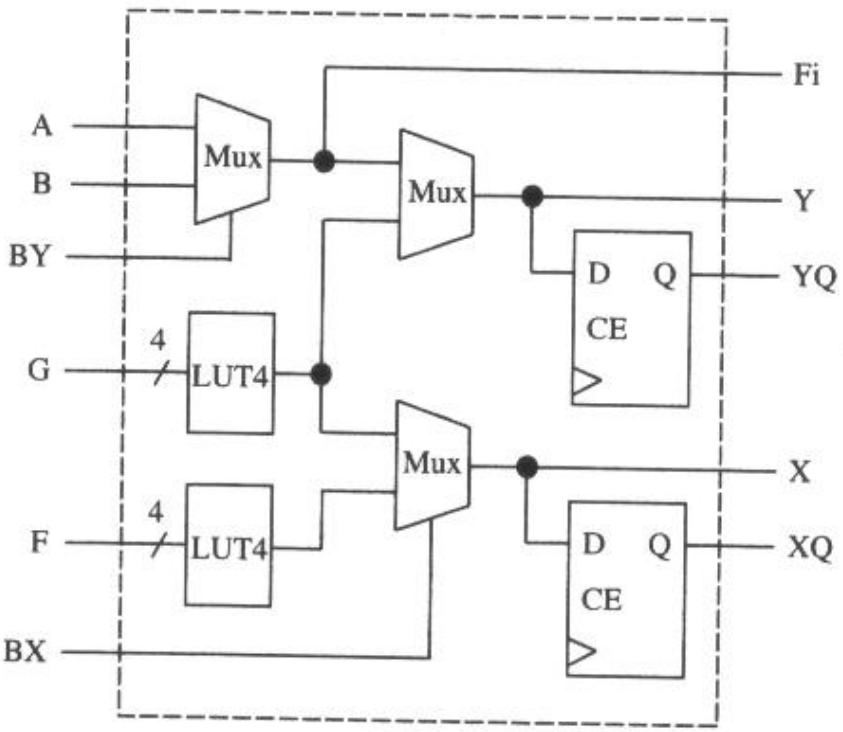


Shannon's Decomposition

(Implementing 7 variable function using 4 Xilinx Spartan Slices)

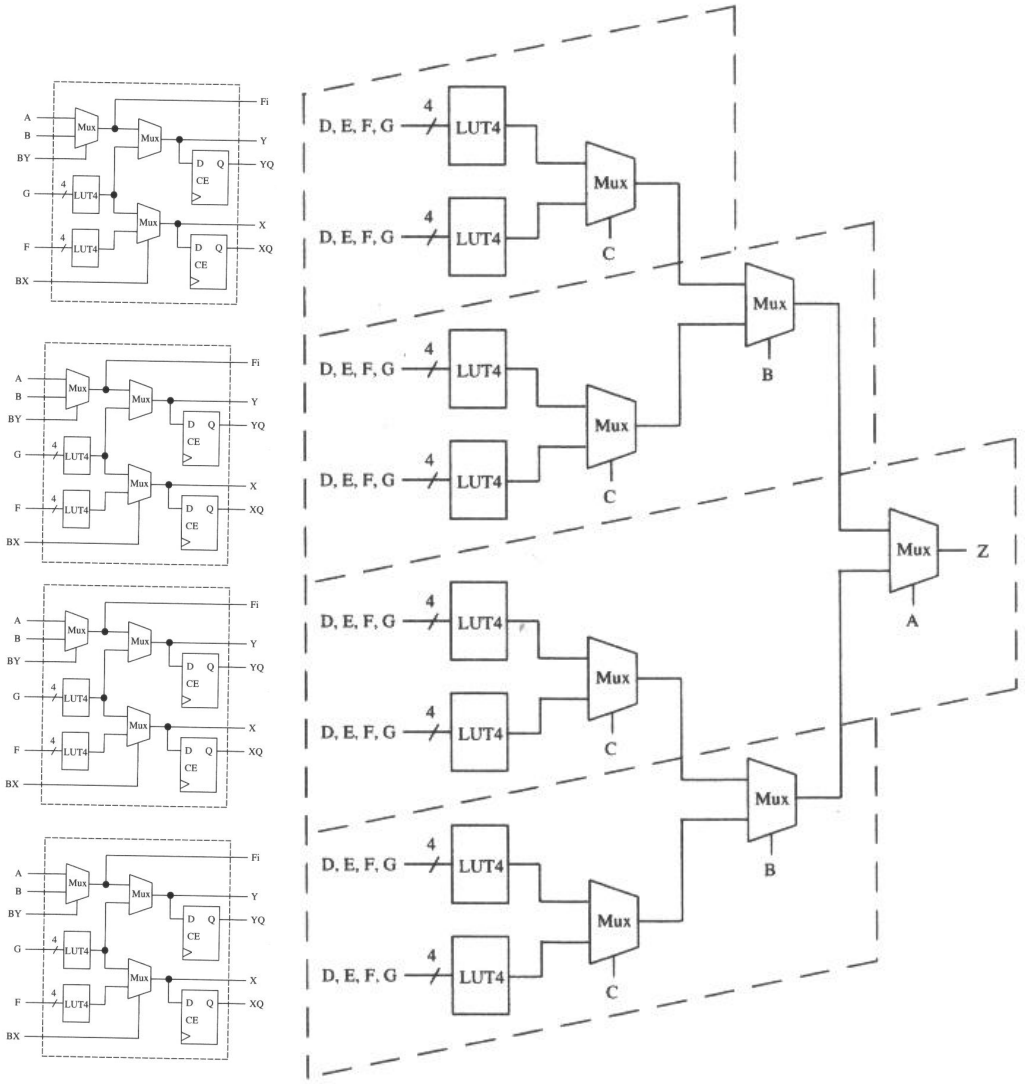
Example: Parity Generation Logic

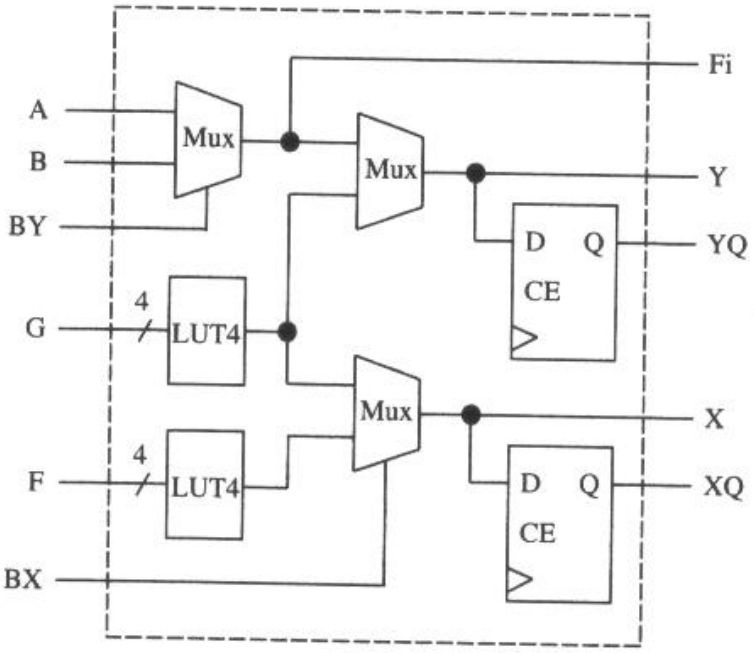
$$Z = A \oplus B \oplus C \oplus D \oplus E \oplus F \oplus G$$



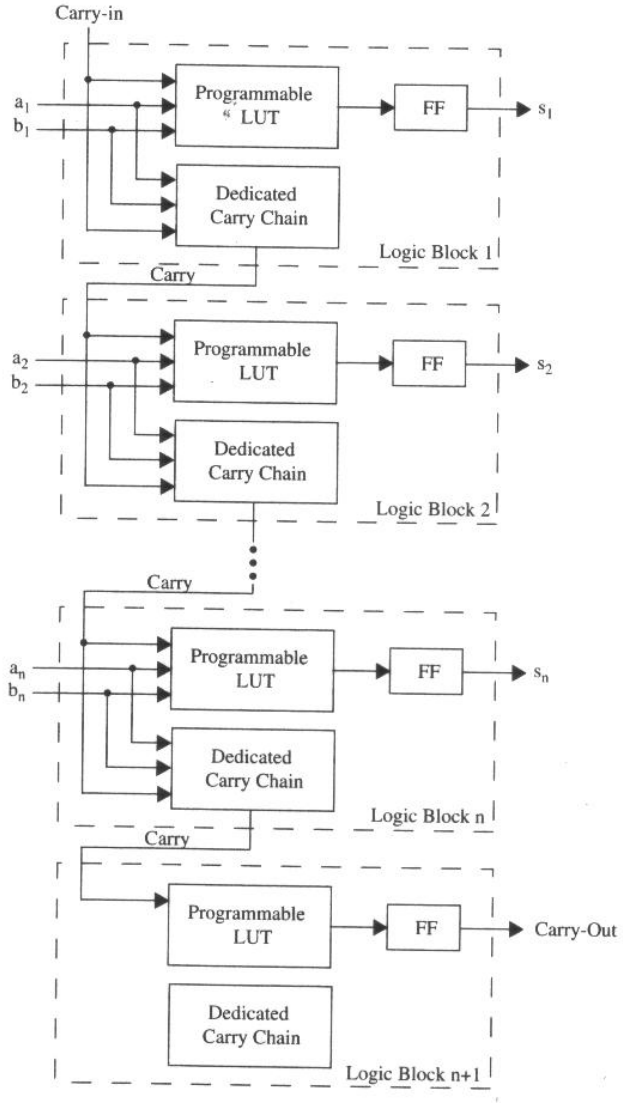
Shannon's Decomposition

(Implementing 7 variable function using 4 Xilinx Spartan Slices)

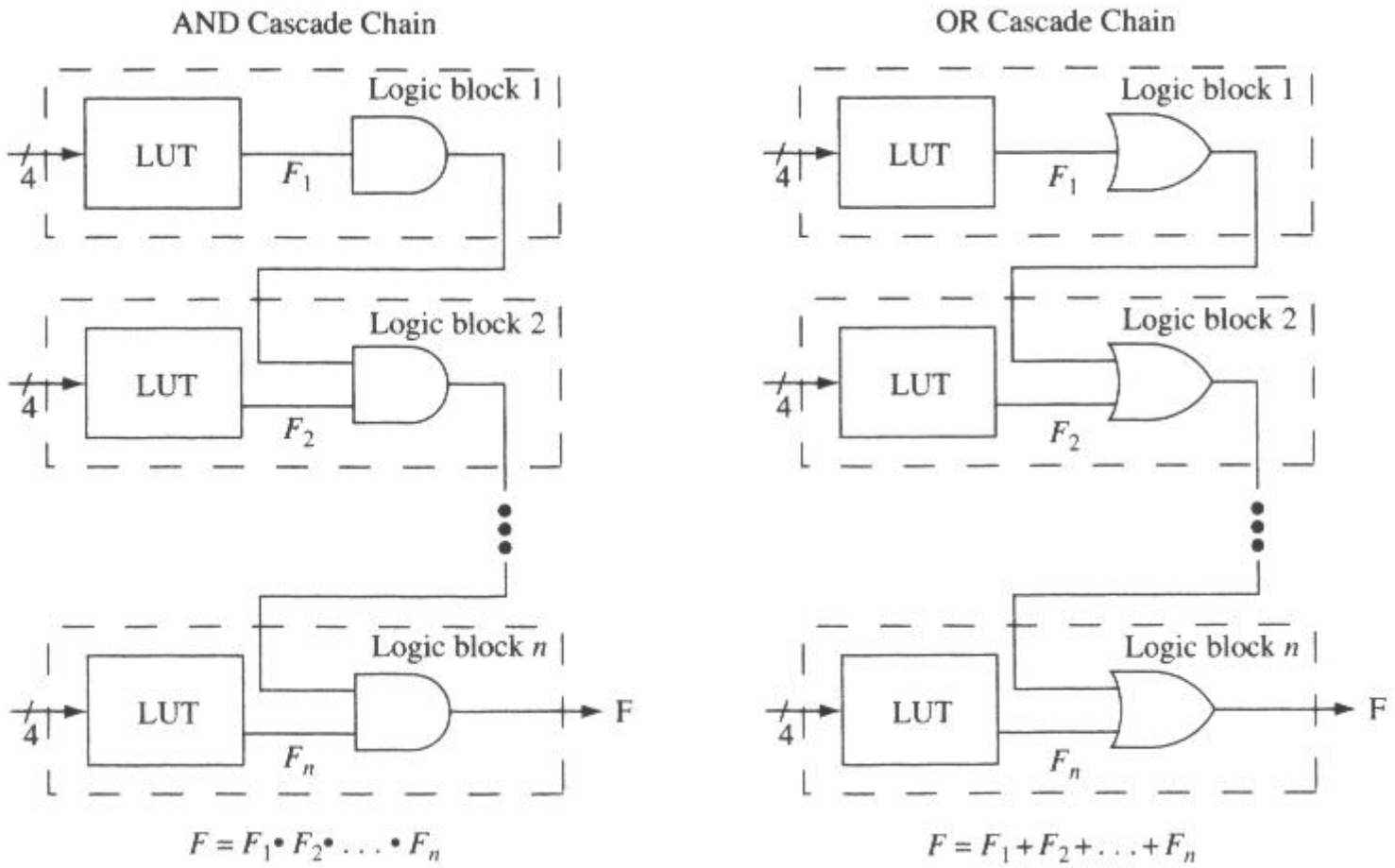




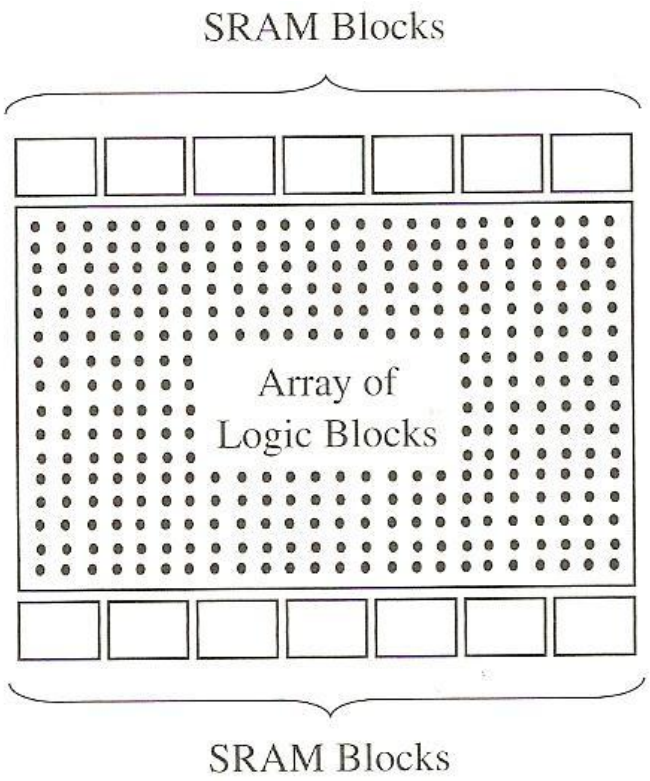
Commonplace Operation Support (Carry Chains in FPGAs)



Commonplace Operation Support (Cascade Chains in FPGAs)



Commonplace Operation Support (Dedicated Memory in FPGAs)



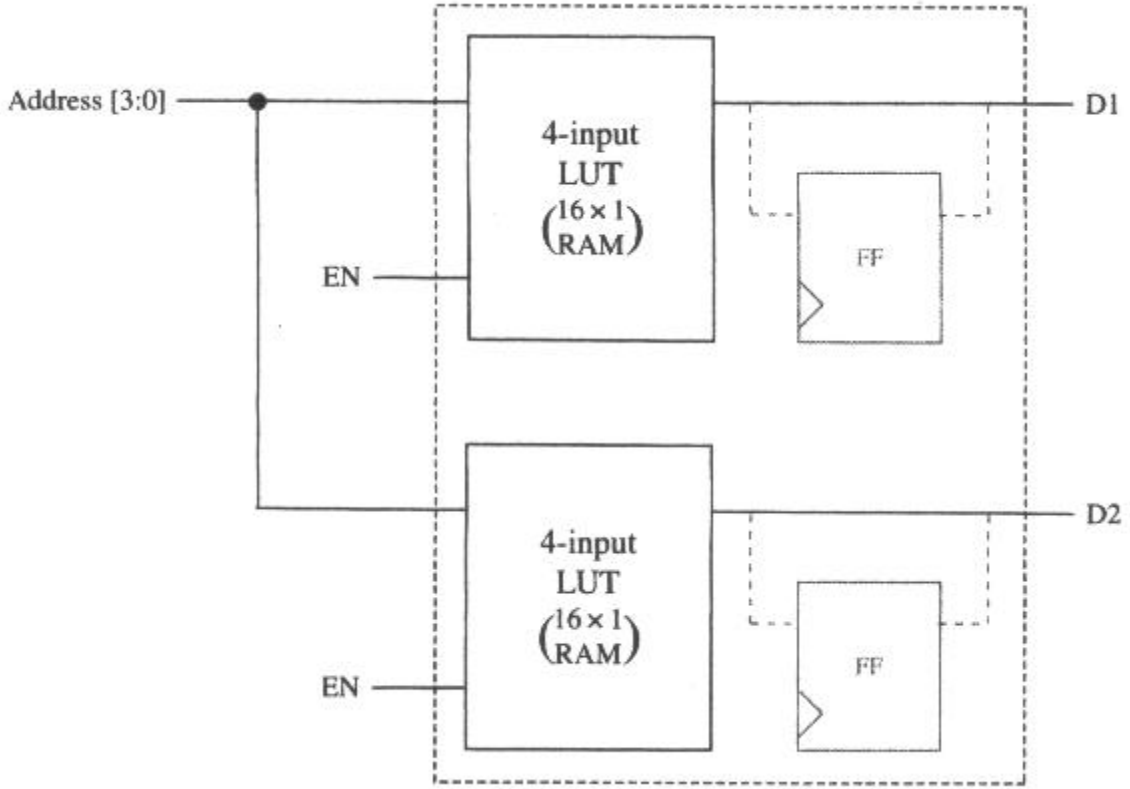
Commonplace Operation Support (Dedicated Memory in FPGAs)

FPGA Family	Dedicated RAM Size (Kb)	Organization
Xilinx Virtex 5	1152–10368	64–576 18Kb blocks
Xilinx Virtex 4	864–9936	48–552 18Kb blocks
Xilinx Virtex-II	72–3024	4–168 18Kb blocks
Xilinx Spartan 3E	72–648	4–36 18Kb blocks
Altera Stratix II	409–9163	104–930 512b blocks 78–768 4Kb blocks 0–9 512Kb blocks
Altera Cyclone II	117–1125	26–250 4Kb blocks
Lattice SC	1054–7987	56–424 18Kb blocks
Actel Fusion	27–270	6–60 4Kb blocks

Dedicated Memory – Variable Data Widths

Width	Depth	Addr Bus	Data Bus
1	32K	15 bits	1 bit
2	16K	14 bits	2 bits
4	8K	13 bits	4 bits
8	4K	12 bits	8 bits
16	2K	11 bits	16 bits

Creating Memory from LUTs
(16 x 2 memory)



Creating Memory from LUTs

Verilog HDL Model that typically infers LUT-based memory

```
module Memory (input CLK, MemWrite, input [11:0] Address,
               input [31:0] Data_In, output [31:0] DATA_Out);

   reg [31:0] DataMEM [0:511];

   initial
     begin
        $readmemh("initial_ram.txt",DataMEM);
     end

   always @(posedge CLK)
      begin
      if (MemWrite)DataMEM[Address] = Data_In;    // Synchronous Write
      end

   assign DATA_Out = DataMEM[Address];            // Asynchronous Read

endmodule
```

Creating Memory from LUTs

Verilog HDL Model that typically infers LUT-based memory

```
module Memory (input CLK, MemWrite, input [11:0] Address,
               input [31:0] Data_In, output [31:0] DATA_Out);

   reg [31:0] DataMEM [0:511];

   initial
     begin
        $readmemh("initial_ram.txt",DataMEM);
     end

   always @(posedge CLK)
      begin
      if (MemWrite)DataMEM[Address] = Data_In;    // Synchronous Write
      end

   assign DATA_Out = DataMEM[Address];            // Asynchronous Read

endmodule
```

Flow Summary	
Flow Status	Successful - Mon Apr 07 19:48:32 2014
Quartus II 32-bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Full Version
Revision Name	Memory
Top-level Entity Name	Memory
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	23,890 / 33,216 (72 %)
Total combinational functions	23,069 / 33,216 (69 %)
Dedicated logic registers	16,384 / 33,216 (49 %)
Total registers	16384
Total pins	78 / 475 (16 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

Creating Memory from LUTs

Verilog Model that typically infers Dedicated memory

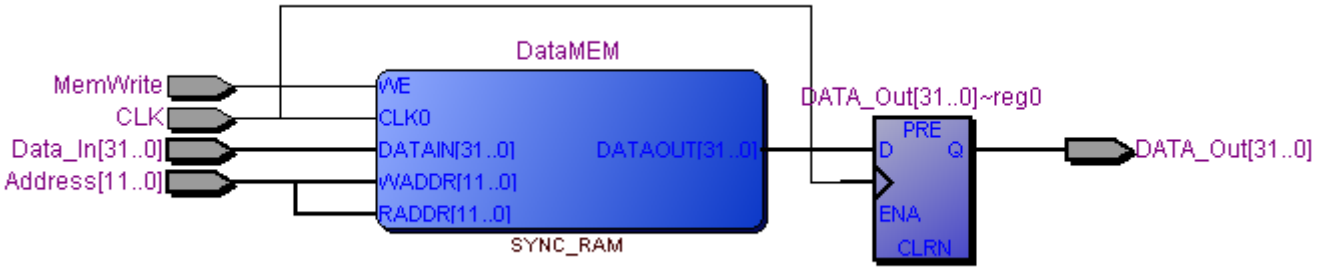
```
module Memory (input CLK, MemWrite, input [11:0] Address,
               input [31:0] Data_In, output reg [31:0] DATA_Out);

   reg [31:0] DataMEM [0:4095];

   initial
     begin
        $readmemh("initial_ram.txt",DataMEM);
     end

   always @(posedge CLK)
      begin
      if (MemWrite)DataMEM[Address] = Data_In;   // Synchronous Write
      DATA_Out = DataMEM[Address];               // Synchronous Read
      end

endmodule
```

Flow Summary	
Flow Status	Successful - Mon Apr 07 19:23:32 2014
Quartus II 32-bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Full Version
Revision Name	Memory
Top-level Entity Name	Memory
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	33 / 33,216 (< 1 %)
Total combinational functions	32 / 33,216 (< 1 %)
Dedicated logic registers	33 / 33,216 (< 1 %)
Total registers	33
Total pins	78 / 475 (16 %)
Total virtual pins	0
Total memory bits	131,072 / 483,840 (27 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

Creating Memory from LUTs

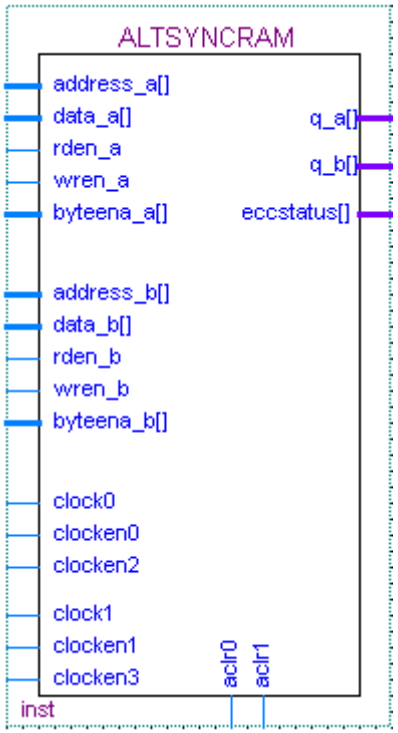
Verilog Model that typically infers Dedicated memory

```
module Memory (input CLK, MemWrite, input [11:0] Address,
               input [31:0] Data_In, output reg [31:0] DATA_Out);

   reg [31:0] DataMEM [0:4095];

   initial
     begin
        $readmemh("initial_ram.txt",DataMEM);
     end

   always @(posedge CLK)
      begin
      if (MemWrite)DataMEM[Address] = Data_In;    // Synchronous Write
      DATA_Out = DataMEM[Address];                // Synchronous Read
      end

endmodule
```



Structural Technique for Implementing Memory Elements in the FPGA's Dedicated Block RAM Memory

```
// Memory module in Verilog HDL in a manner that explicitly utilizes
// the Altera Megafunction library in Quartus II. This will utilize
// 64 M4K dedicated memory elements for a total storage of
// 262144 bits (It also uses 48 LE's).
module rom (input [14:0] address, input clock, output [7:0] q);
   altsyncram   C1 (
      .clock0 (clock),
      .address_a (address),
      .q_a (q),
      .aclr0 (1'b0),
      .aclr1 (1'b0),
      .address_b (1'b1),
      .addressstall_a (1'b0),
      .addressstall_b (1'b0),
      .byteena_a (1'b1),
      .byteena_b (1'b1),
      .clock1 (1'b1),
      .clocken0 (1'b1),
      .clocken1 (1'b1),
      .clocken2 (1'b1),
      .clocken3 (1'b1),
      .data_a ({8{1'b1}}),
      .data_b (1'b1),
      .eccstatus (),
      .q_b (),
      .rden_a (1'b1),
      .rden_b (1'b1),
      .wren_a (1'b0),
      .wren_b (1'b0));
```




```
defparam
   C1.clock_enable_input_a = "BYPASS",
   C1.clock_enable_output_a = "BYPASS",
   C1.init_file = "E:/example/rom.hex",
   C1.intended_device_family = "Cyclone II",
   C1.lpm_hint = "ENABLE_RUNTIME_MOD=NO",
   C1.lpm_type = "altsyncram",
   C1.numwords_a = 32768,
   C1.operation_mode = "ROM",
   C1.outdata_aclr_a = "NONE",
   C1.outdata_reg_a = "UNREGISTERED",
   C1.power_up_uninitialized = "FALSE",
   C1.ram_block_type = "M4K",
   C1.widthad_a = 15,
   C1.width_a = 8,
   C1.width_byteena_a = 1;

endmodule
```

Using LUTS to Implement a 4x4 Multiplier
(using LUTs – i.e. distributed RAM)

```
module LUTmult(input [3:0] Mplier, Mcand, output reg [7:0] Product);

   reg [7:0] prod_rom [0:255];

   initial
      begin:ROM_LOAD
      reg [8:0] i;
      for (i=0;i<256;i=i+1)
         prod_rom[i] = {4'b0000,i[7:4]}*{4'b0000,i[3:0]};
      end
```

columns

rows

```
(x"00", x"00",
x"00", x"01",
x"00", x"02",
x"00", x"03",
x"00", x"04",
x"00", x"05",
x"00", x"06",
x"00", x"07",
x"00", x"08",
x"00", x"09",
x"00", x"0A",
x"00", x"0B",
x"00", x"0C",
x"00", x"0D",
x"00", x"0E",
x"00", x"0F",
```

Flow Summary

Flow Status	Successful - Tue Apr 08 09:08:21 2014
Quartus II 32-bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Full Version
Revision Name	LUTmult
Top-level Entity Name	LUTmult
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	67 / 33,216 (< 1 %)
Total combinational functions	67 / 33,216 (< 1 %)
Dedicated logic registers	0 / 33,216 (0 %)
Total registers	0
Total pins	16 / 475 (3 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

```
00", x"00", x"00",
0D", x"0E", x"0F",
", x"1C", x"1E",
27", x"2A", x"2D",
34", x"38", x"3C",
41", x"46", x"4B",
4E", x"54", x"5A",
5B", x"62", x"69",
68", x"70", x"78",
75", x"7E", x"87",
82", x"8C", x"96",
8F", x"9A", x"A5",
9C", x"A8", x"B4",
A9", x"B6", x"C3",
B6", x"C4", x"D2",
C3", x"D2", x"E1");
```

```
   always @ (Mplier or Mcand)
      Product = prod_rom[{Mplier,Mcand}]; // read Product LUT

endmodule
```

Using LUTS to Implement a 4x4 Multiplier
(using Dedicated Block RAM)

```
module LUTmult(input [3:0] clk, Mplier, Mcand, output reg [7:0] Product);

   reg [7:0] prod_rom [0:255];

   initial
      begin:ROM_LOAD
      reg [8:0] i;
      for (i=0;i<256;i=i+1)
         prod_rom[i] = {4'b0000,i[7:4]}*{4'b0000,i[3:0]};
      end
```

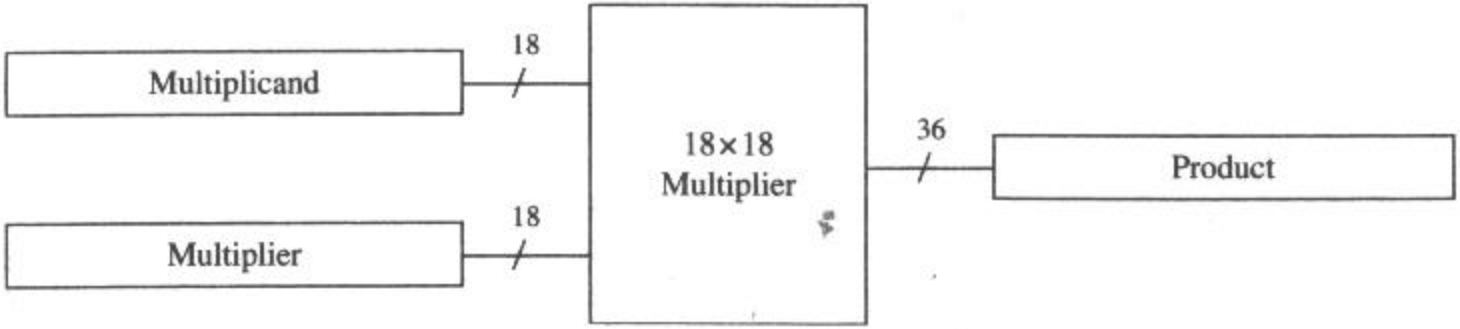
columns

rows

Flow Summary	
Flow Status	Successful - Tue Apr 08 09:19:39 2014
Quartus II 32-bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Full Version
Revision Name	LUTmult
Top-level Entity Name	LUTmult
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	0 / 33,216 (0 %)
Total combinational functions	0 / 33,216 (0 %)
Dedicated logic registers	0 / 33,216 (0 %)
Total registers	0
Total pins	20 / 475 (4 %)
Total virtual pins	0
Total memory bits	2,048 / 483,840 (< 1 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

```
   always @ (posedge clk)
      Product = prod_rom[{Mplier,Mcand}]; // read Product LUT
                                          // synchronously

endmodule
```

Common Operation Support (Multipliers in FPGAs)




```
module multiplier (input [31:0] A,B, output [63:0] C);

   assign C = A * B;

endmodule
```

Flow Summary	
Flow Status	Successful - Tue Apr 08 09:27:18 2014
Quartus II 32-bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Full Version
Revision Name	multiplier
Top-level Entity Name	multiplier
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	80 / 33,216 (< 1 %)
Total combinational functions	80 / 33,216 (< 1 %)
Dedicated logic registers	0 / 33,216 (0 %)
Total registers	0
Total pins	128 / 475 (27 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	8 / 70 (11 %)
Total PLLs	0 / 4 (0 %)

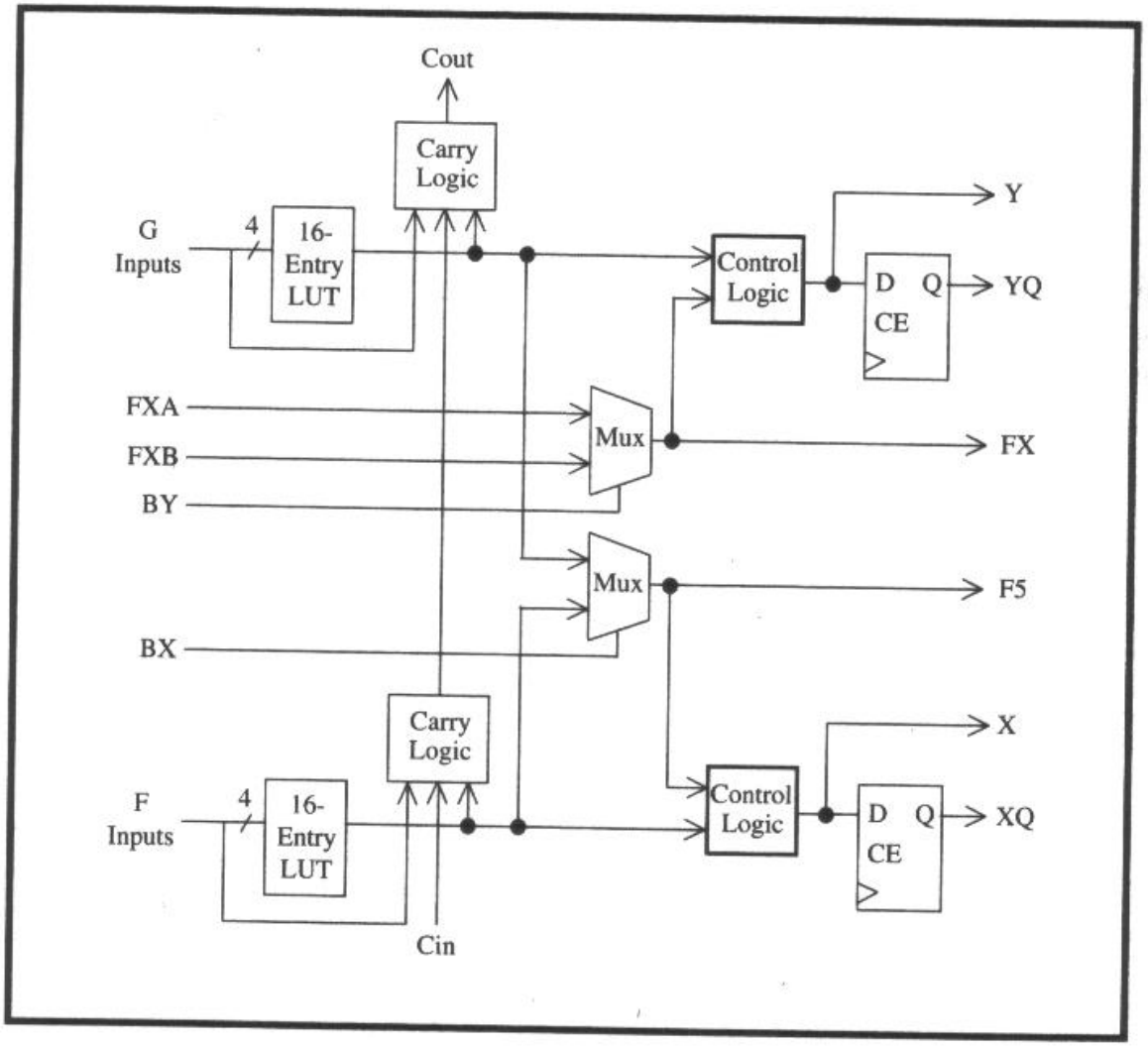
FPGA	Dedicated Multipliers
Xilinx Virtex-4, Virtex-II Pro/X, Spartan-3E, Spartan 3/3L	18 × 18 multipliers
Altera Stratix II Cyclone II	18 × 18 multipliers

Common Operation Support

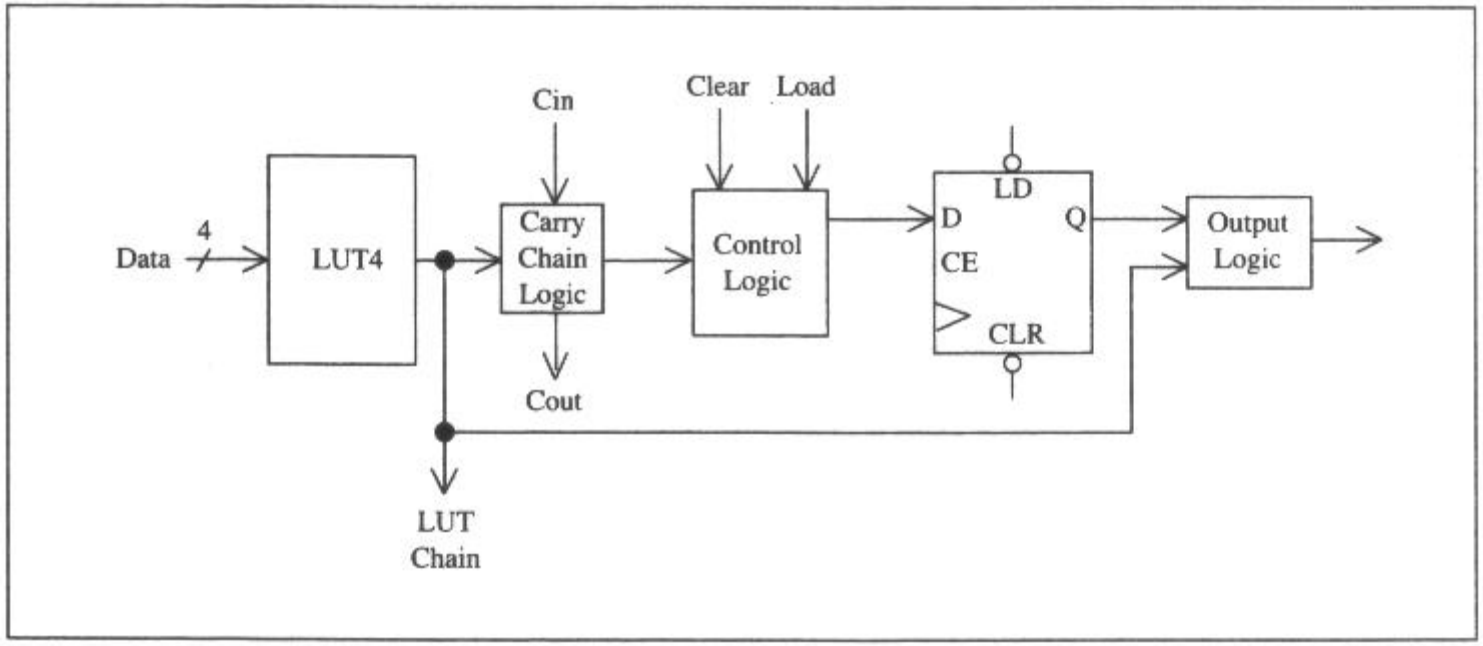
(high end logic, embedded processors, dsp, soft/hard ip cores)

FPGA	Embedded Processor
Xilinx Virtex-4, Virtex-II Pro/X	IBM 400 MHz PowerPC
Xilinx Spartan-3E, Spartan 3/3l	MicroBlaze PicoBlaze
Altera Stratix II Cyclone II	Nios II
Altera APEX APEX II	ARM, MIPS, Nios
Altera Excalibur	ARM 9
Actel Fusion	ARM7

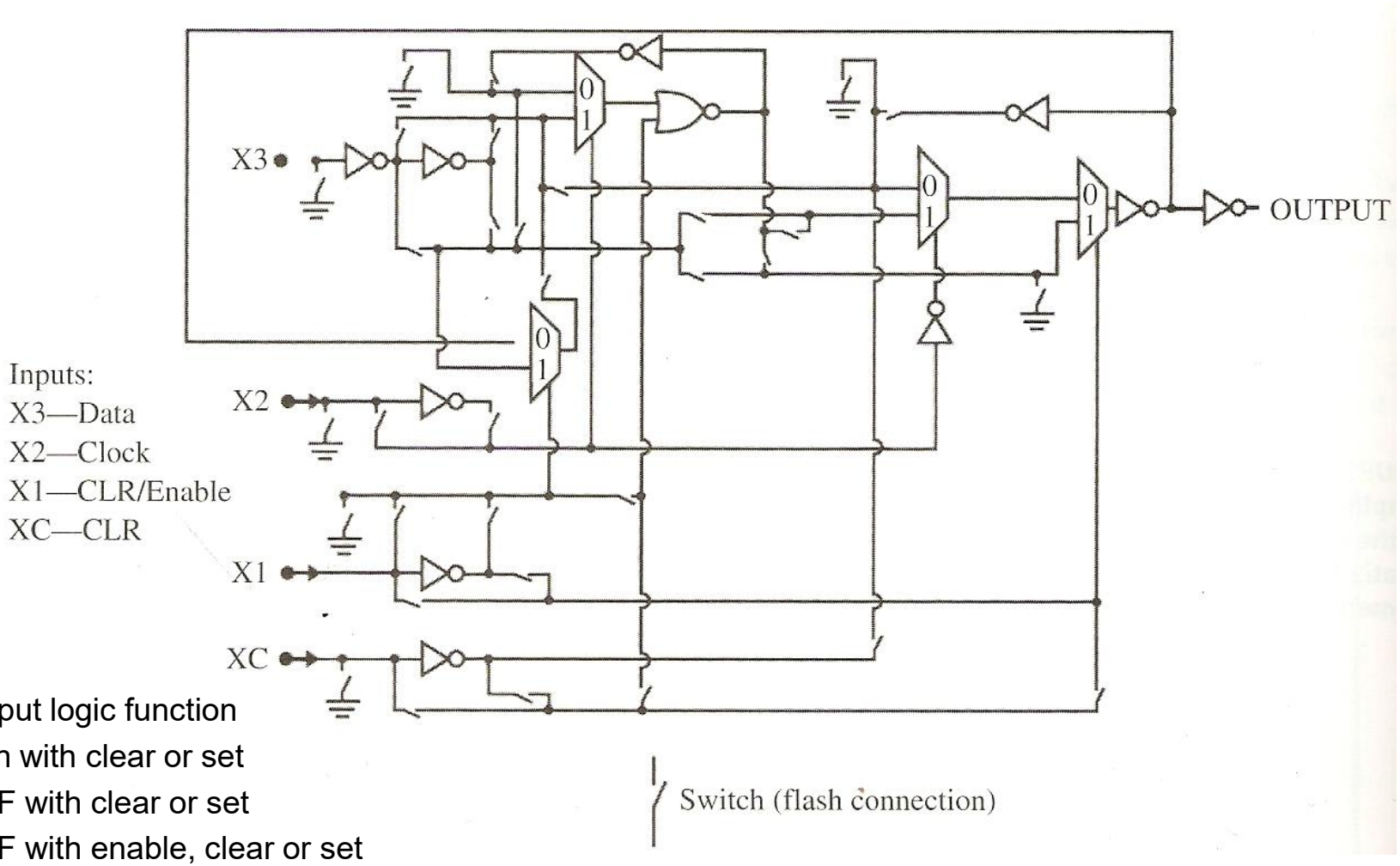
Logic Blocks of some Commercial FPGAs (Xilinx Spartan Slice)



Logic Blocks of some Commercial FPGAs (Altera Stratix Logic Element -- LE)



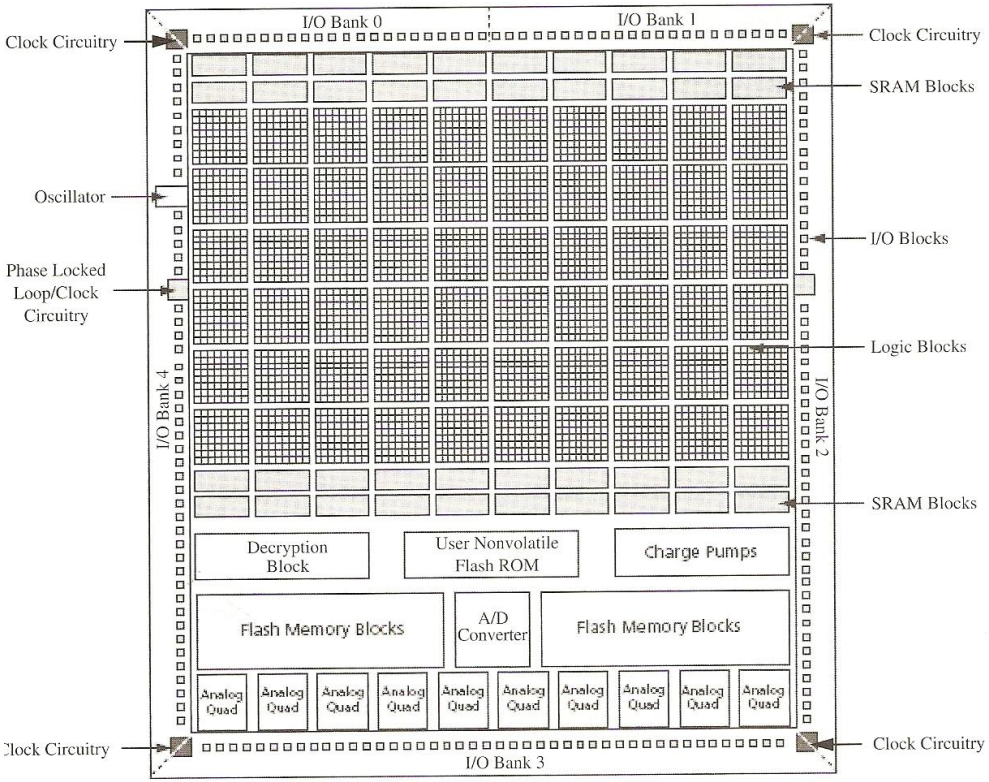
Logic Blocks of some Commercial FPGAs (Actel Fusion & ProAS1C Logic Block)



a 3-input logic function
a latch with clear or set
a D-FF with clear or set
a D-FF with enable, clear or set

Notes on Actel Fusion Architecture

The Actel Fusion architecture, shown in Figure 3-40, provides several specialized components, including embedded RAM, decryption, and A/D converters. At the core of the chip are tiles of logic blocks (VersaTiles in Actel terminology). The embedded RAM is in the form of rows of SRAM blocks above and below the tiles of logic blocks. Several specialized components appear below the SRAM blocks in the bottom. There is a dedicated decryption unit that implements the AES decryption algorithm. (AES stands for Advanced Encryption Standard, which has been the cryptograhic standard for the U.S. government since 2001.) There is an analog-to-digital converter (ADC) that accepts inputs from several analog quads, which are circuitry to condition analog signals received by the FPGA. The analog quads contain circuitry to monitor and condition signals according to voltage, current, and temperature.



Granularity

- FPGA Granularity – working definition –
  complexity of the Logic Blocks
  - Fine grain; FPGAs with very simple logic blocks
  - Coarse grain; FPGAs with complex logic blocks

Fine grain ← range →Coarse grain

simple LBs (smaller DP) complex LBs (wider DP)

more LBs in FPGA less LBs in FPGA

easier to map harder to map

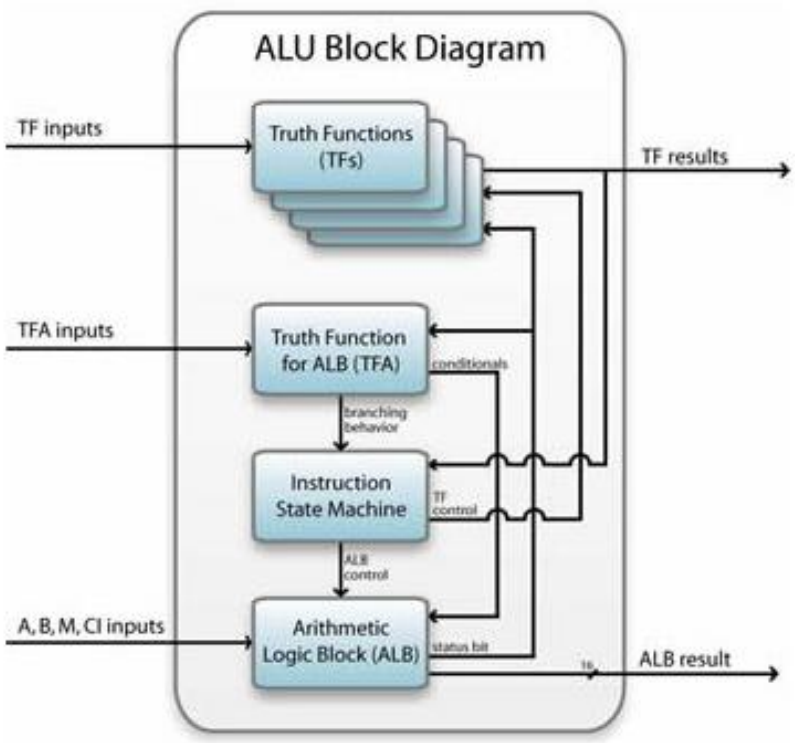
harder to place/route easier to place/route

can waste LBs can waste internal LB resources

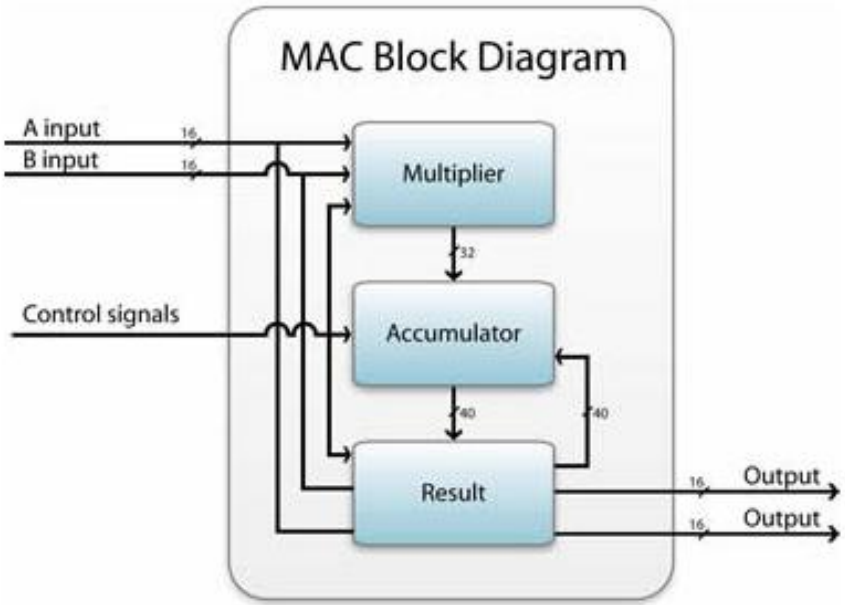
Actel Fusion ←→Altera Stratix ←→ Xilinx Spartian

MathStar's Field Programmable Logic Array
(Very Coarse-Grain Logic Blocks)

Arithmetic Logic Unit (ALU)

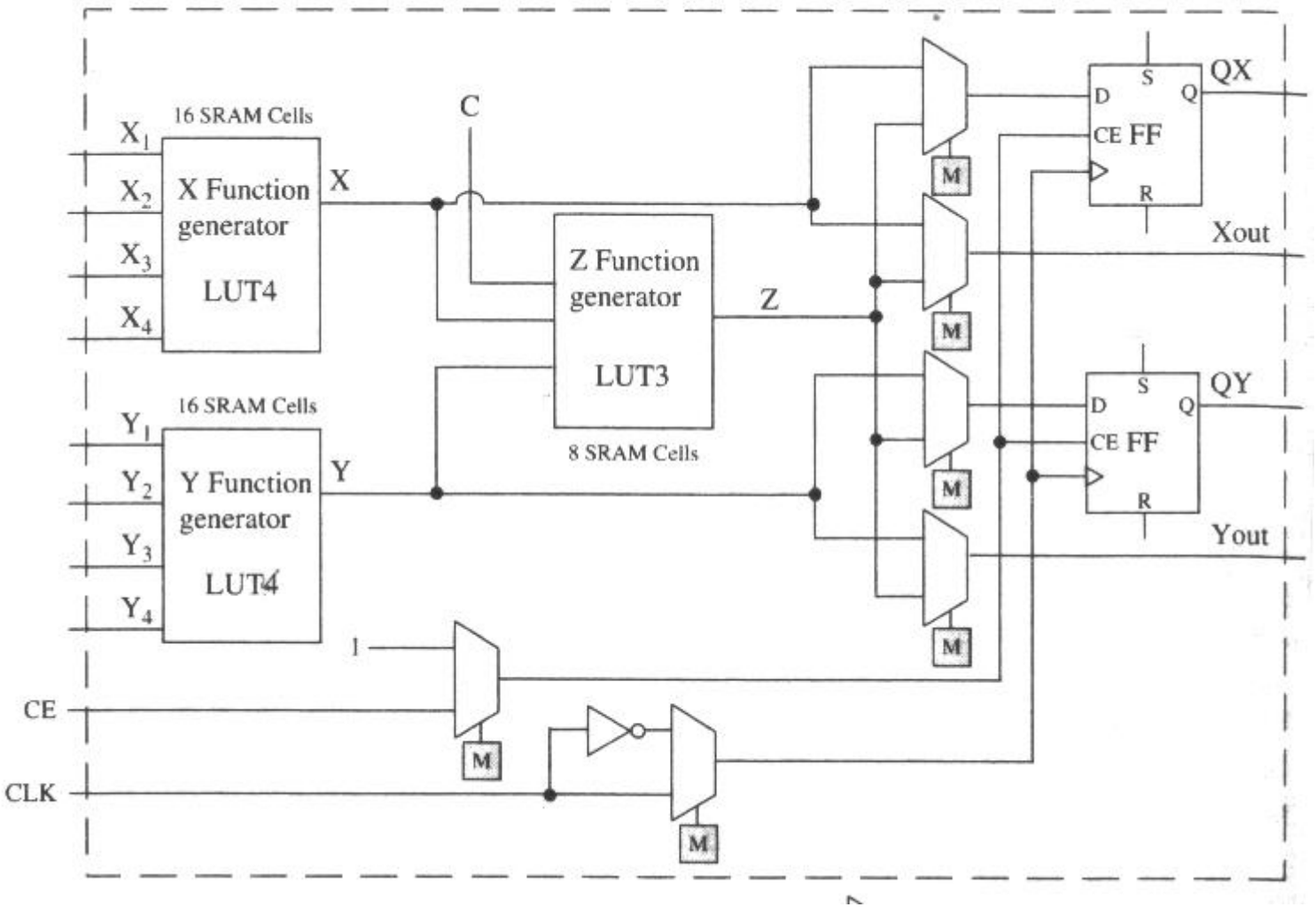


Multiply - Accumulator (MAC)

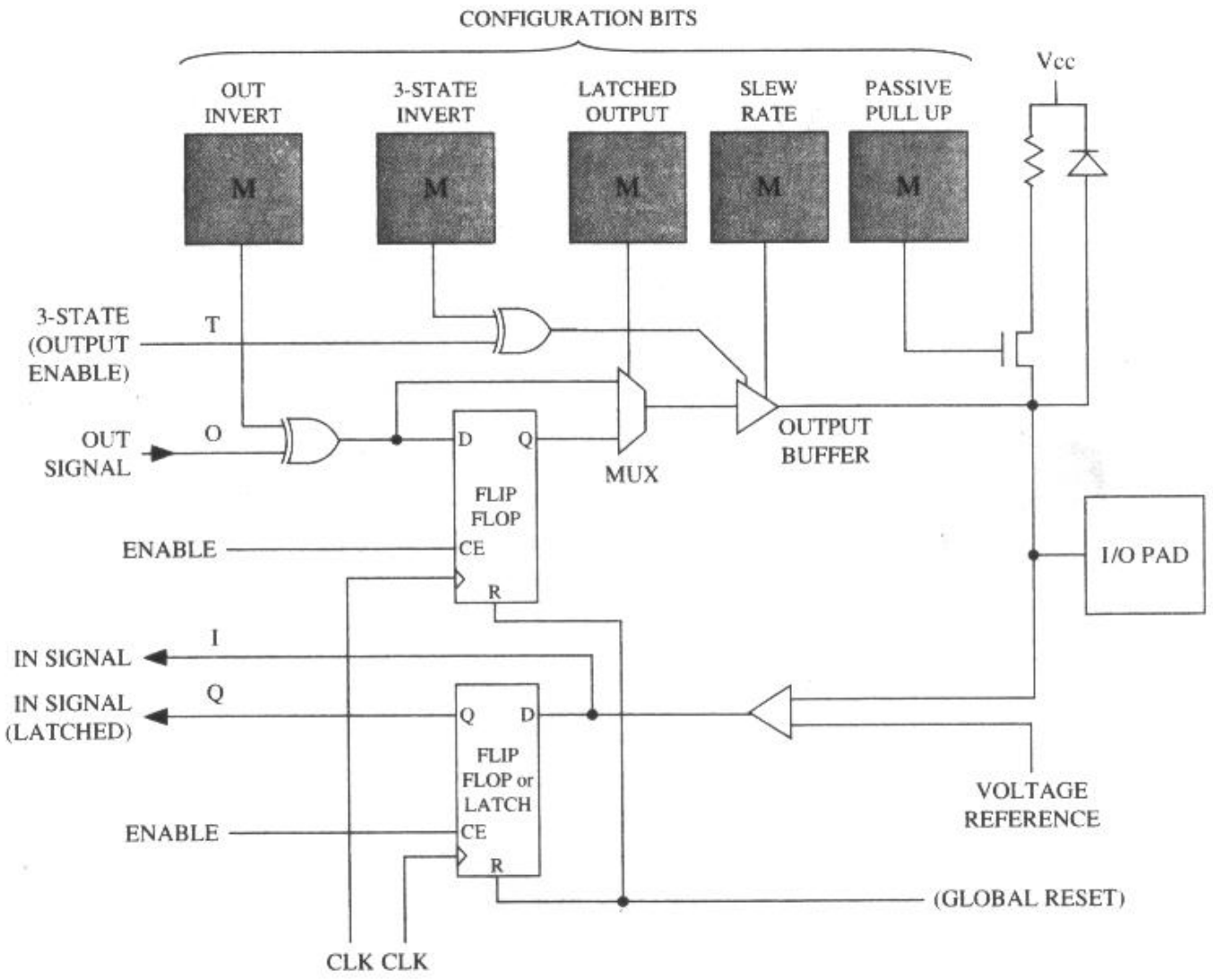


Cost of Reconfigurability

(programmable points in a logic block)



Cost of Reconfigurability
(programmable points in a FPGA I/O Block)



Cost of Reconfigurability

Vendor	Device Family	Device	# of Configuration Bits	# of Logic Blocks	# of LUTs	# Usable I/O Pins
Xilinx	Virtex-5	XC5VLX30 XC5VLX330	8.4M 79.7M	4,800 51,840	19,200 207,360	400 1200
Xilinx	Virtex-II	XC2V40 XC2V8000	0.3M 26.2M	256 46,592	512 93,184	88 1108
Xilinx	Spartan 3E	XC3S100E XC3S1600E	0.6M 6.0M	960 14,752	1,920 29,504	108 376
Altera	Stratix II	EP2S15 EP2S180	4.7M 49.8M	6,240 71,760	12,480 143,520	366 1170
Altera	Stratix	EP1S10 EP1S80	3.5M 23.8M	10,570 79,040	10,570 79,040	426 1238
Altera	Cyclone II	EP2C5 EP2C70	1.3M 14.3M	4,608 68,416	4,608 68,416	158 622

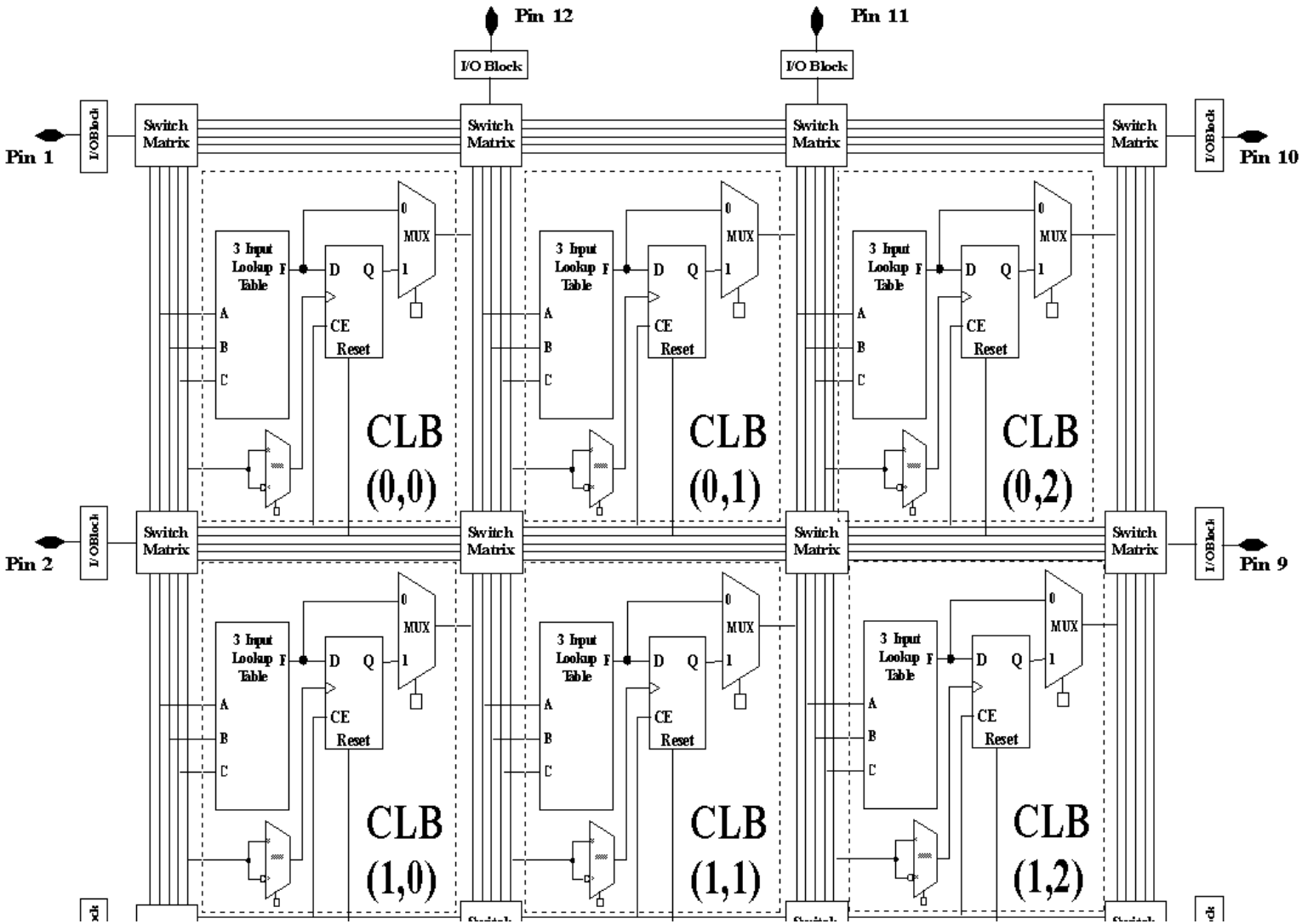
State Machine Implementation in FPGAs

- In FPGAs, it may not be important to minimize the number of flip-flops used in the design
- Goal should be to reduce the total number of logic cells used and the interconnections between the cells
- The manner in which the state assignments are performed can facilitate this goal.
- Often this means that we need to use more flip-flops than the minimum of $\log_2(N)$, where N is the number of states.

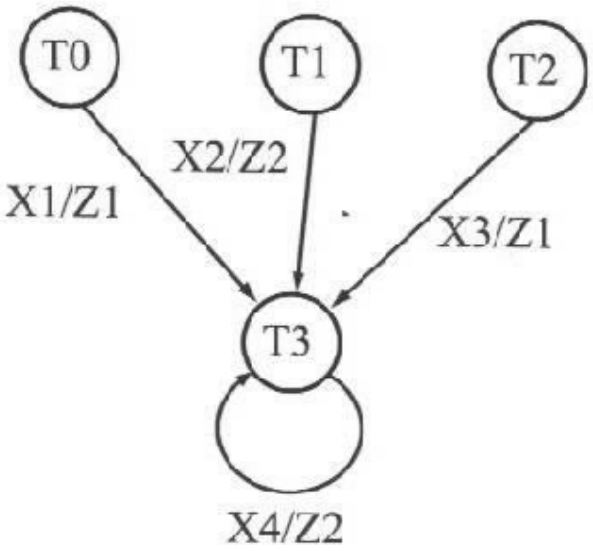
One-Hot State Assignment

- The one-hot assignment method uses one flip-flop for each state
  - State machine with N states requires N flip-flops (not $\log_2(N)$ flip-flops as before!
- Often FPGAs have a flip-flop with each cell. This flip-flop cannot be used by other circuitry when any of the other components of the cell are used
  - Therefore the availability of usable flip-flops in the FPGA may not be the limiting constraint.
- Often one-hot state assignments result in simpler state assignments and output equations which are easier to route and fit in the lookup tables.

Simplified View of an FPGA Architecture



One-Hot State Assignment



One-hot state assignment for flip-flops Q0 Q1 Q2 Q3:

T0: 1000, T1: 0100, T2: 0010, T3: 0001

$$Q3^{+} = X1\ Q0 + X2\ Q1 + X3\ Q2 + X4\ Q3$$

$$Z1 = X1\ Q0 + X3\ Q2$$

$$Z2 = X2\ Q1 + X4\ Q3$$

One-Hot State Assignment

- When a one-hot assignment is used, resetting the system requires that one flip-flop be set to 1 instead of being reset to 0.
- Could be a problem if flip-flops do not have a preset input
  - Two workarounds in this case
    - Replace one of the flip-flop variables with its complement in all the equations ($Q_3+ = X_1Q_0'$ instead of $Q_3+ = X_1Q_0'$ then

      So: $Q_0Q_1Q_2Q_3 = 0000$, $Q_3+ = X_1Q_0'$ instead of $Q_0Q_1Q_2Q_3 = 1000$ and $Q_3+ = X_1Q_0$.
    - Add (i.e. OR) the term $Q_0'Q_1'Q_2'Q_3'$ to the equation Q_0+ then after the first clock cycle Q_0+ will equal 1 and you will be in the correct starting state!

Metrics for FPGA Capacity

Vendor	FPGA Product	Capacity (Approx) in Gates/LUTs
Xilinx	Spartan-II Spartan-IIE Spartan-3 Virtex-5 Virtex Virtex-E Virtex-II	15K to 200K 50K to 600K 50K to 5M 19,200 to 207,360 LUTs 57,906 to 1,124,022 71,693 to 4,074,387 40K to 8M
Altera	ACEX 1K APEX II FLEX 10K Stratix/Stratix II	56K to 257K 1.9M to 5.25M 10K to 50K 10,570 to 132,540 logic elements
Lattice Semiconductor	LatticeECP2 Lattice SC ispXPGA MachXO LatticeECP	6K to 68K LUTs 15.2K to 115.2K LUTs 139K to 1.25M 256 to 2280 LUTs 6.1K to 32.8K LUTs
Actel	Axcelerator eX ProASIC3 MX	125K To 2M 3K to 12K 30K to 3M 3K to 54K
Quick Logic	Eclipse/EclipsePlus Quick RAM pASIC 3	248K to 662K 45K to 176K 5K to 75K
Atmel	AT40K AT40KAL	5K to 40K 5K to 50K

Metrics for FPGA Capacity

(equivalent gate count, etc)

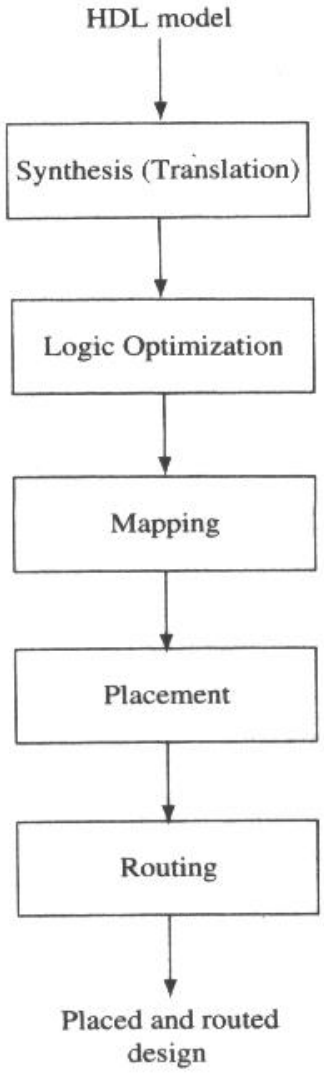
PREP Benchmarks

The **Programmable Electronics Performance Company (PREP)** was a non-profit organization that gathered and distributed a series of benchmarks for programmable ASICs. The nine PREP benchmark circuits in the PREP 1.3 suite were as follows:

1. An 8-bit datapath consisting of a 4-to-1 MUX, a register, and a shift register
2. An 8-bit timer-counter consisting of two registers, a 4-to-1 MUX, a counter, and a comparator
3. A small state machine (8 states, 8 inputs, and 8 outputs)
4. A larger state machine (16 states, 8 inputs, and 8 outputs)
5. An ALU consisting of a 4 × 4 multiplier, an 8-bit adder, and an 8-bit register
6. A 16-bit accumulator
7. A 16-bit counter with synchronous load and enable
8. A 16-bit prescaled counter with load and enable
9. A 16-bit address decoder

PREP's online information included Verilog and VHDL source code and test benches (provided by Synplicity). PREP also made additional synthesis benchmarks available, including a bit-slice processor, multiplier, and R4000 MIPS RISC microprocessor.

FPGA CAD Design Flow



Mapping

- Mapping: process of binding technology-dependent circuits of the target technology to the technology-independent circuits in the design (produced by translation/synthesis phase)
- In FPGAs it focuses upon dividing the synthesized design into sub-blocks where each sub-block can fit into a logic block, I/O block, or some other logic entity within the FPGA
- Where these blocks are to be placed and how they are connected within the FPGA is handled in the place and rout phase of the design flow

Placement

- Placement: Process of taking defined logic sub-blocks and assigning them to physical locations with the target implementation.

- In FPGAs it focuses upon assigning each sub-block to specific Logic blocks, I/O bocks, or dedicated memory

Routing

- Routing: process of interconnecting the sub-blocks in the design.
- Note: place and route are dependent upon each other but can be performed as separate steps.
- In FPGAs routing involves providing the interconnection between the Logic Blocks, I/O blocks, and the other hardware entities after the necessary logic functionality has been assigned to these entities.

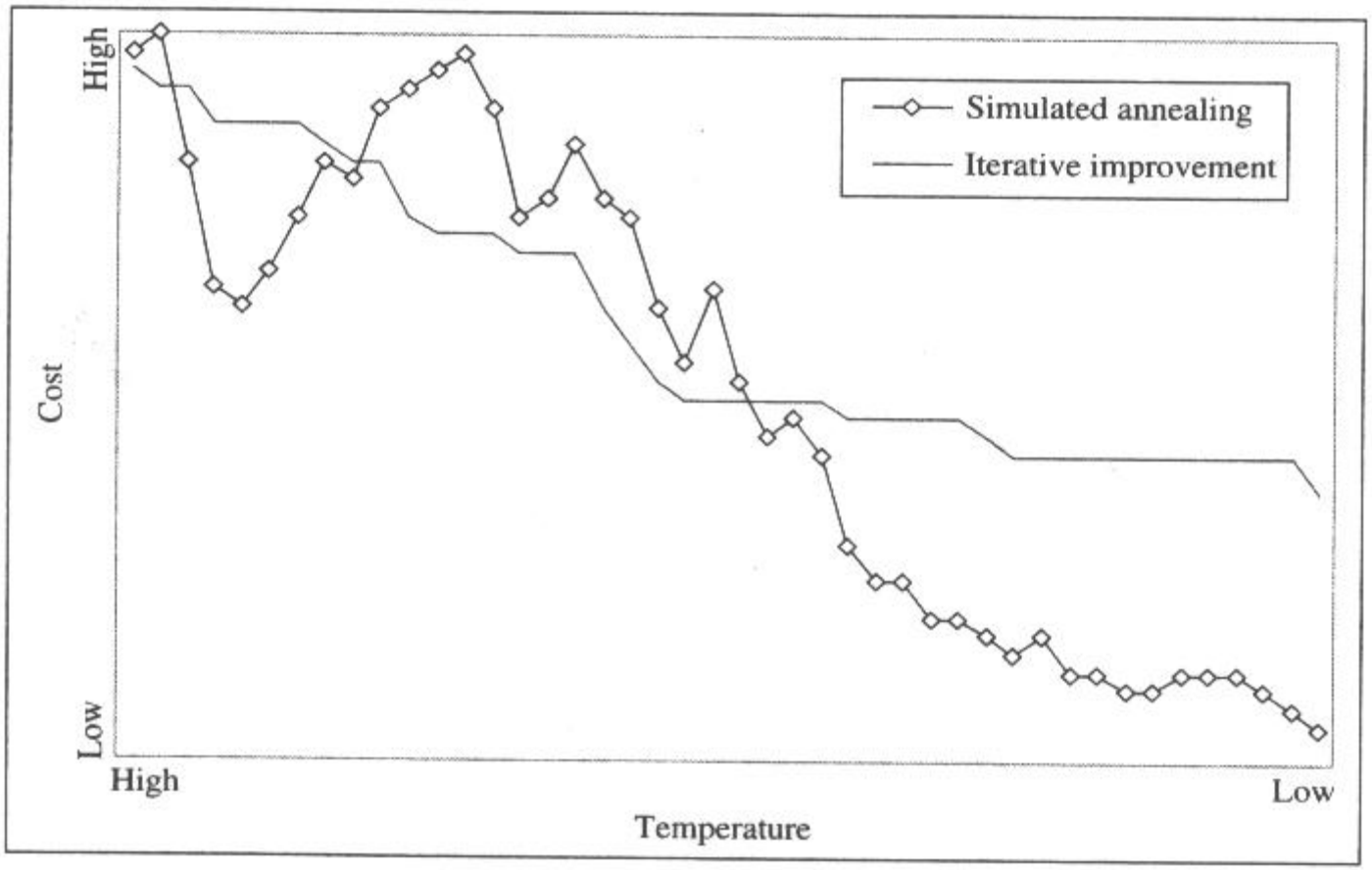
Mapping, Place and Route

- Automated Heuristics are used by the CAD tool to perform these operations.
- Most techniques create an initial solution to the problem and then try to improve upon the solution by selectively altering a portion of the solution.
- Algorithms based upon greedy/iterative improvement techniques employ a greedy strategy and will only accept a new solution if it is better than a previous one
- These algorithms can get trapped at a local optimization point that can be much worse than the global one

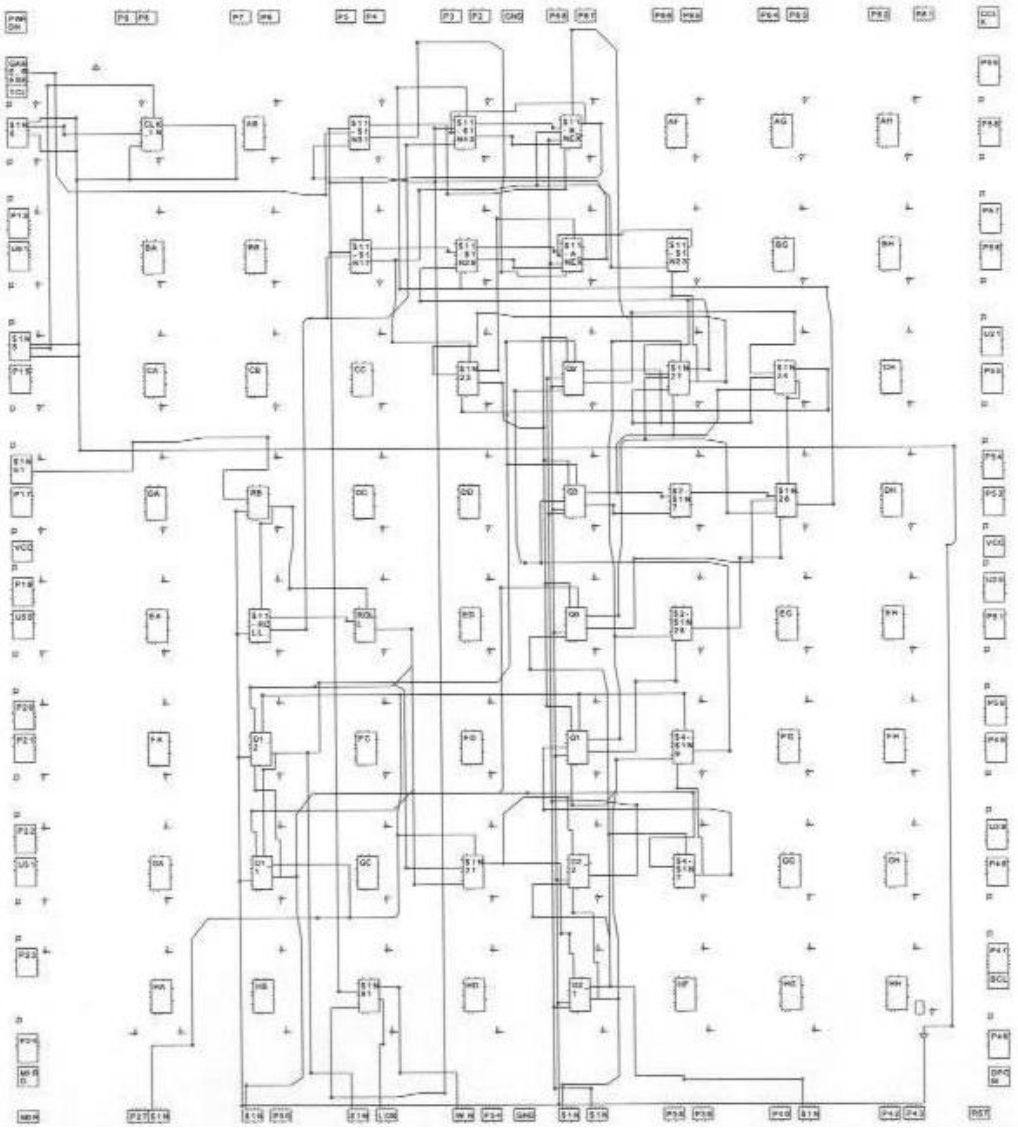
Mapping, Place and Route

- Hill Climbing Heuristics attempt overcome this problem – they are most often used in modern CAD tool environments
- Simulated Annealing is one such hill-climbing technique
- Is is based upon the process of Annealing Molten Metal
- It allows solutions that are worse than a previous solution to be accepted in a probablistic manner during different iterations.
  - As the heuristic nears completion the algorithm reverts to a simple greedy iterative approach – only accepting solutions that are better than the current one.

Simulated Annealing for Place and Route



FPGA, Mapping, Place and Route

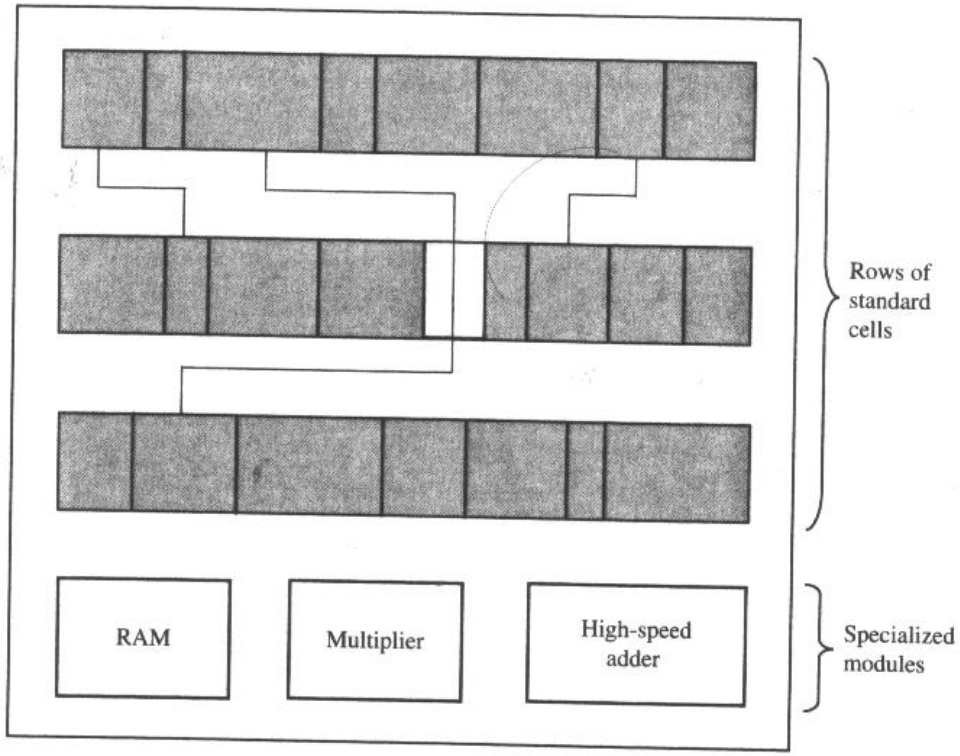


Mapping, Place and Route

- Of course the same design can be implemented in other non-FPGA based technologies as well.
- Example Standard Cell

Standard Cell Approach Standard cell design is a common technique for integrated circuit design. The design is mapped into a library of standard logic gates. Typically NOT, AND, NAND, OR, NOR, XOR, XNOR, and so on are available. CAD tools that support standard cell design methodology will also usually contain a library of complex functions and standard building blocks such as multiplexers, decoders, encoders, comparators, and counters. The design is mapped into a form that contains only cells available in the library. The cells are placed in rows that are separated by routing channels. Some cells may be used only for routing between rows of cells. Such cells are called feedthrough cells. For the standard cell methodology to be effective, the height of cells should be the same. But it is possible to include memory modules, specialized arithmetic modules, and so on.

Mapping, Placement, and Routing



Place and Route

- Placement: taking the defined logic block and I/O blocks