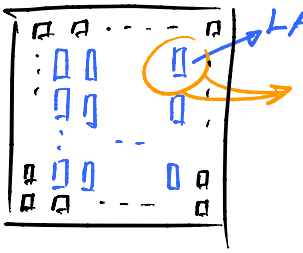
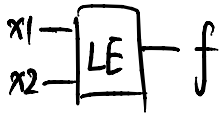
B6.5 FPGA (Field programmable gated array)

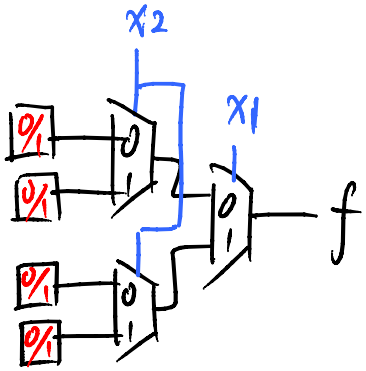


each LE can store a truth table that represents the logic function that is implemented. ie If the LE has k inputs, then it can represent any k-input logic function

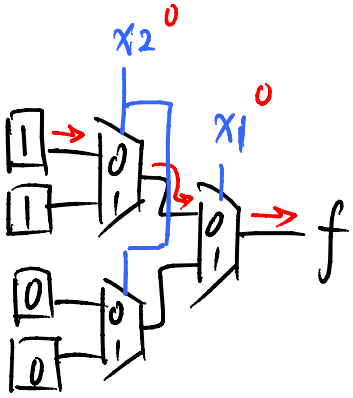
eg. k=2

x_1	x_2	f
0	0	0/1
0	1	0/1
1	0	0/1
1	1	0/1





x_1	x_2	f
0	0	1
0	1	1
1	0	0
1	1	0



⇐ called Look-up table (LUT)

on DE2, there 35,000 4-input LUTs.

3.2 Ripple-carry adder

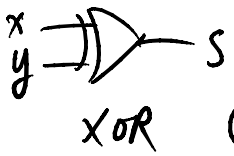
recall adding 2 bits:

$$\begin{array}{r} x \\ +\ y \\ \hline c\ s \end{array}$$

x	y	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$c = xy$

$s = \bar{x}y + x\bar{y}$



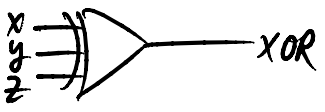
XOR ⊕

3-input XOR

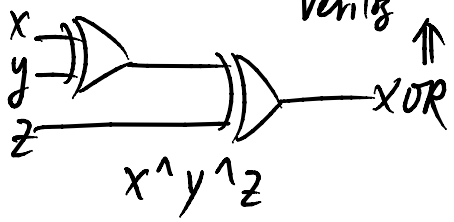
x	y	z	XOR
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

detecting "odd" number of inputs being "1"

$XOR = x \oplus y \oplus z$



⇒



extend to adding multi-bits: $A = a_2 a_1 a_0$ and $B = b_2 b_1 b_0$

eg. A = 010, B = 111

```
    1100        C3 C2 C1 C0
(A)  010           a2 a1 a0
(B) +111       +)  b2 b1 b0
-------        ------------
    1001        C3 S2 S1 S0
```

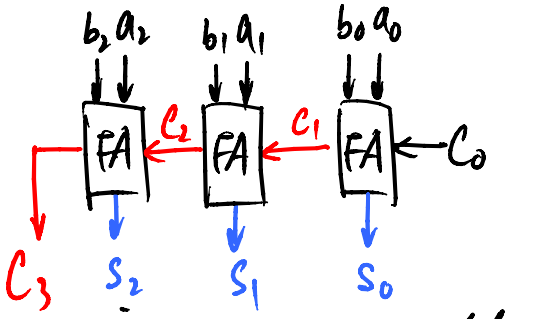
c_i	b_i	a_i	c_{i+1}	S_i
0	0	0	0	0
0	0	1	0	1 ←
0	1	0	0	1 ←
0	1	1	1 ←	0
1	0	0	0	1 ←
1	0	1	1 ←	0
1	1	0	1 ←	0
1	1	1	1 ←	1 ←

```
  Ci+1 Ci
     ai
+)   bi
-------
  Ci+1 Si
```

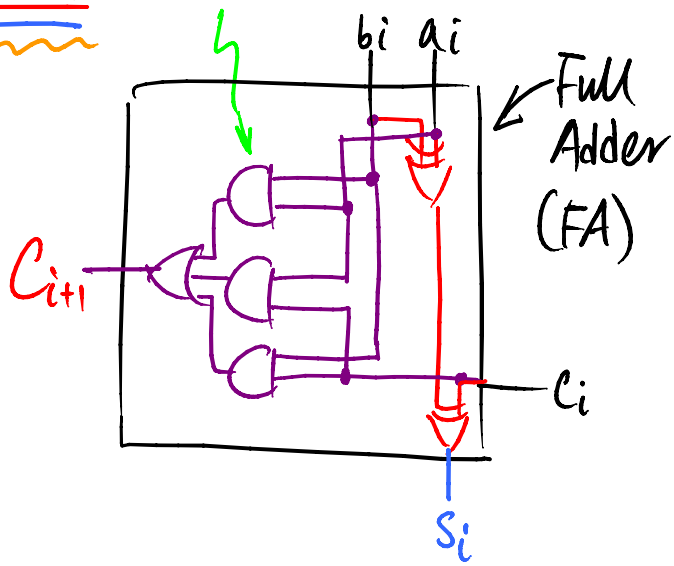
simplified logic expressions

$S_i = a_i \oplus b_i \oplus c_i$

$C_{i+1} = \bar{c}_i b_i a_i + c_i \bar{b}_i a_i + c_i b_i \bar{a}_i + c_i b_i a_i = b_i a_i + c_i a_i + c_i b_i$



ripple-carry adder




```
module Carry_Adder(A, B, Cin, S, Cout);
  input [2:0] A, B;
  input Cin;
  output [2:0] S;
  output Cout;
  wire C1, C2; // internal connections
  FA Bit0 (A[0], B[0], Cin, S[0], C1);
  FA Bit1 (A[1], B[1], C1 , S[1], C2);
  FA Bit2 (A[2], B[2], C2 , S[2], Cout);
endmodule
```

```
module FA (input a, b, Cin, output s, Cout);
  assign s = a^b^Cin;   XOR gate (⊕)
  assign Cout = (a&b)|(a&Cin)|(b&Cin);
endmodule
```

Verilog: procedural statement "Always" block

Recall 2-to-1 mux


```
module 2to1mux (input x1, x2, s, output f);
  assign f = (~s & x1) | (s & x2);
endmodule
```

alternative Verilog code for 2-to-1 mux (using if-else)

```
reg f;                          *3
always@(x1, x2, s)
  if (!s)                       *2
    f = x1;
  else
    f = x2;
endmodule
```

```
f = x1;
if (s == 1)
  f = x2;
```

*1 any signal assigned a value inside a "always" block must be declared as reg

*2 "if-else" statements must be inside a "always" block

*3 sensitivity list = variables that will affect the assigned signal, alternative notation

```
always@(*)        <- all input signals
```

*4 if more statements are used inside the "always" block, you need to use

```
always@(*)
  begin
    :
  end
```

note: if-else is treated as a single statement

"Case" statement

→ must be used inside a "always" block

```
case (s)
  1'b0: f = x1;
  1'b1: f = x2;
  default: ...        <- cover all remaining cases.
endcase
```