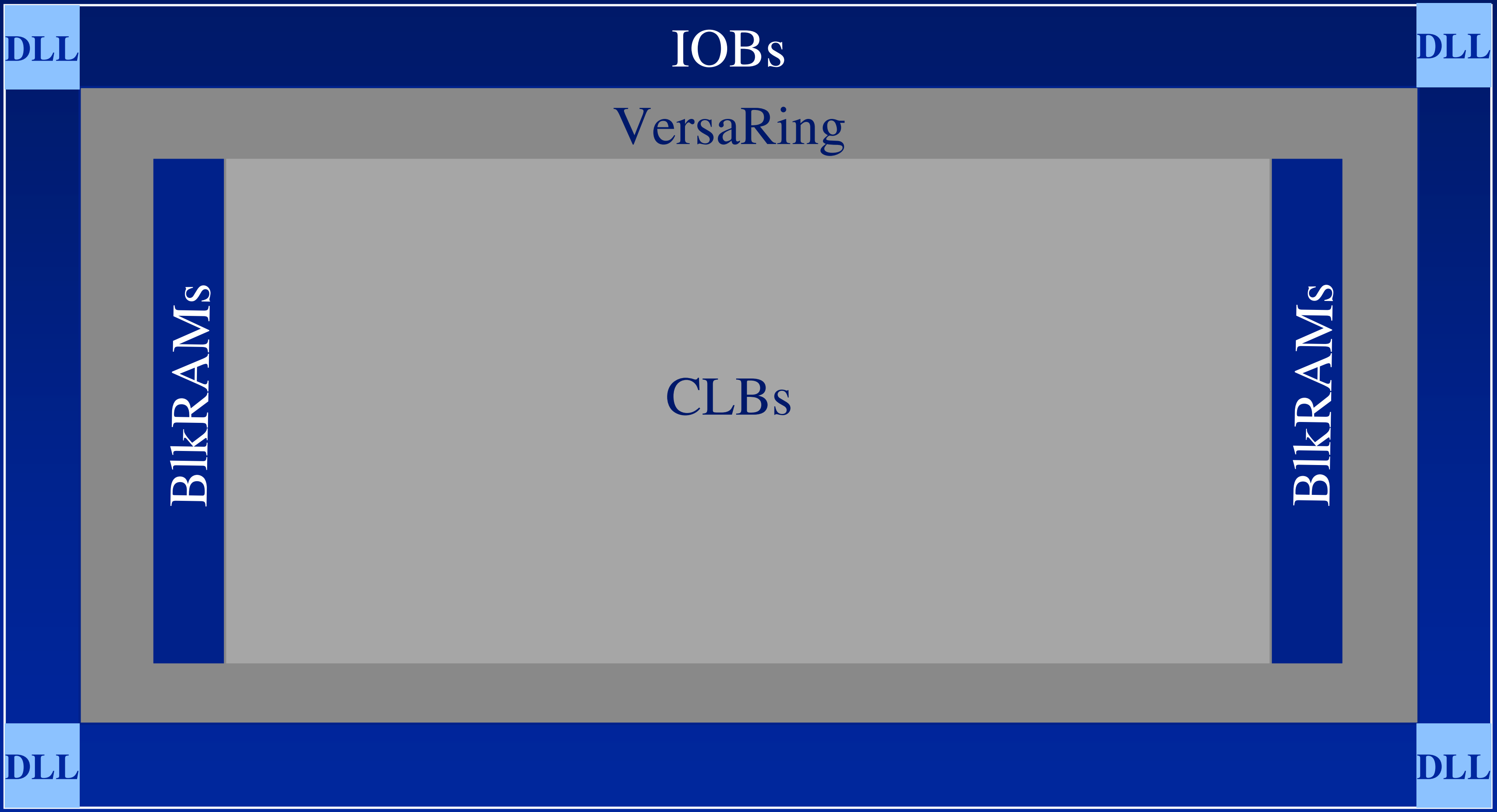
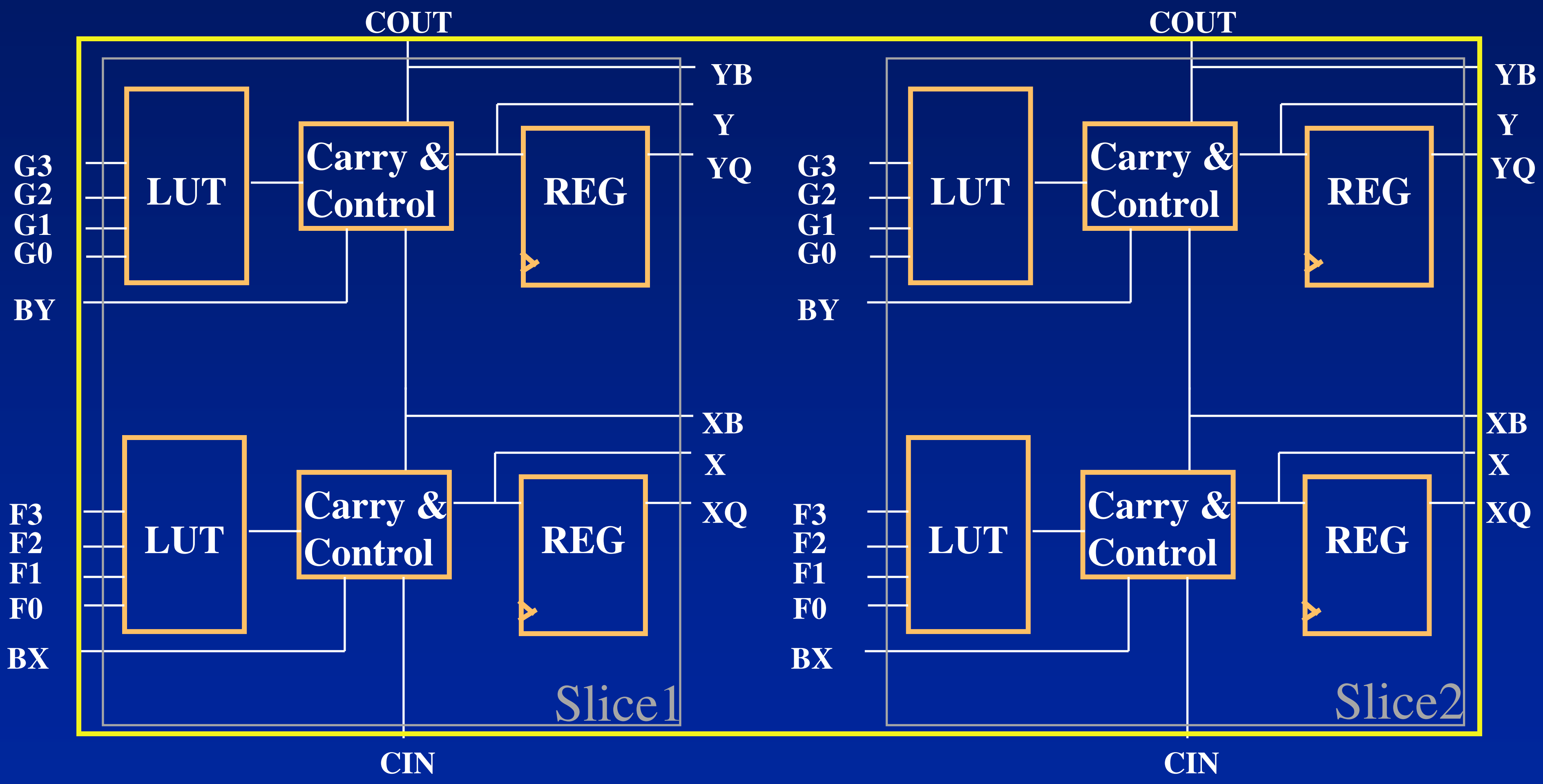
XILINX®

Virtex^{TM} Architecture

VIRTEX™ Architecture

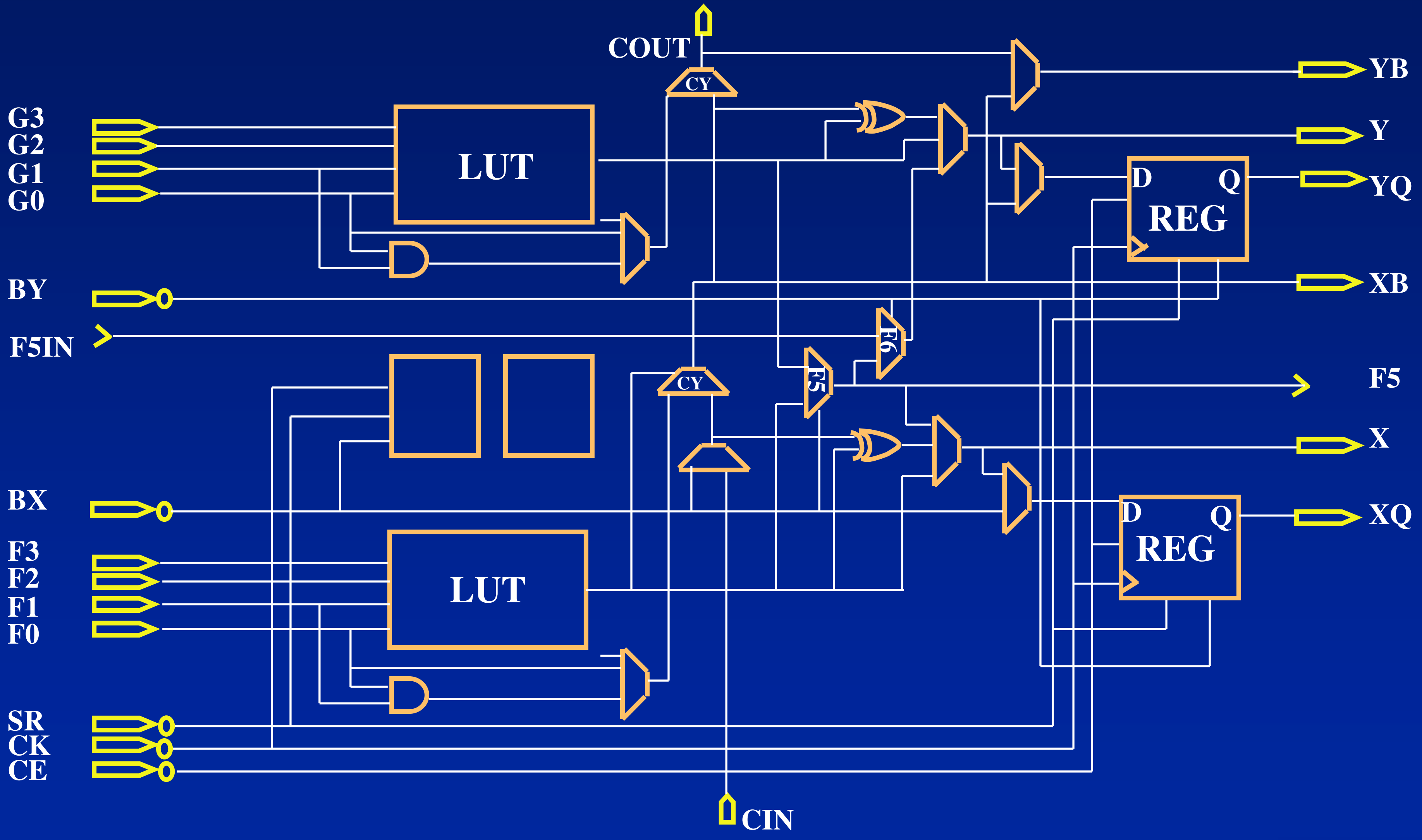


VIRTEX™ CLB

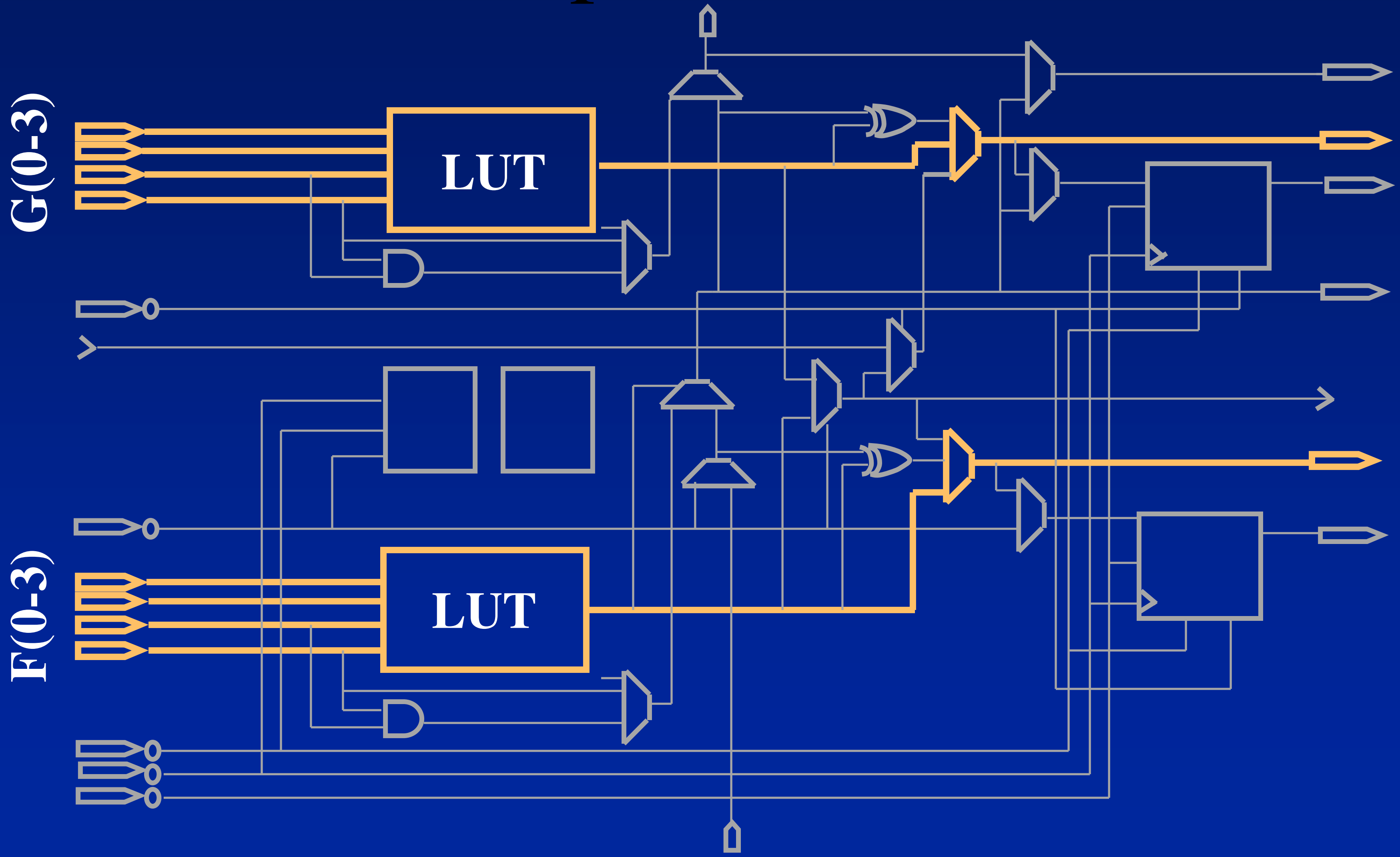


COUT
YB
Y
YQ
G3
G2
G1
G0
BY
LUT
Carry & Control
REG
XB
X
XQ
F3
F2
F1
F0
BX
LUT
Carry & Control
REG
Slice1
CIN
COUT
YB
Y
YQ
G3
G2
G1
G0
BY
LUT
Carry & Control
REG
XB
X
XQ
F3
F2
F1
F0
BX
LUT
Carry & Control
REG
Slice2
CIN

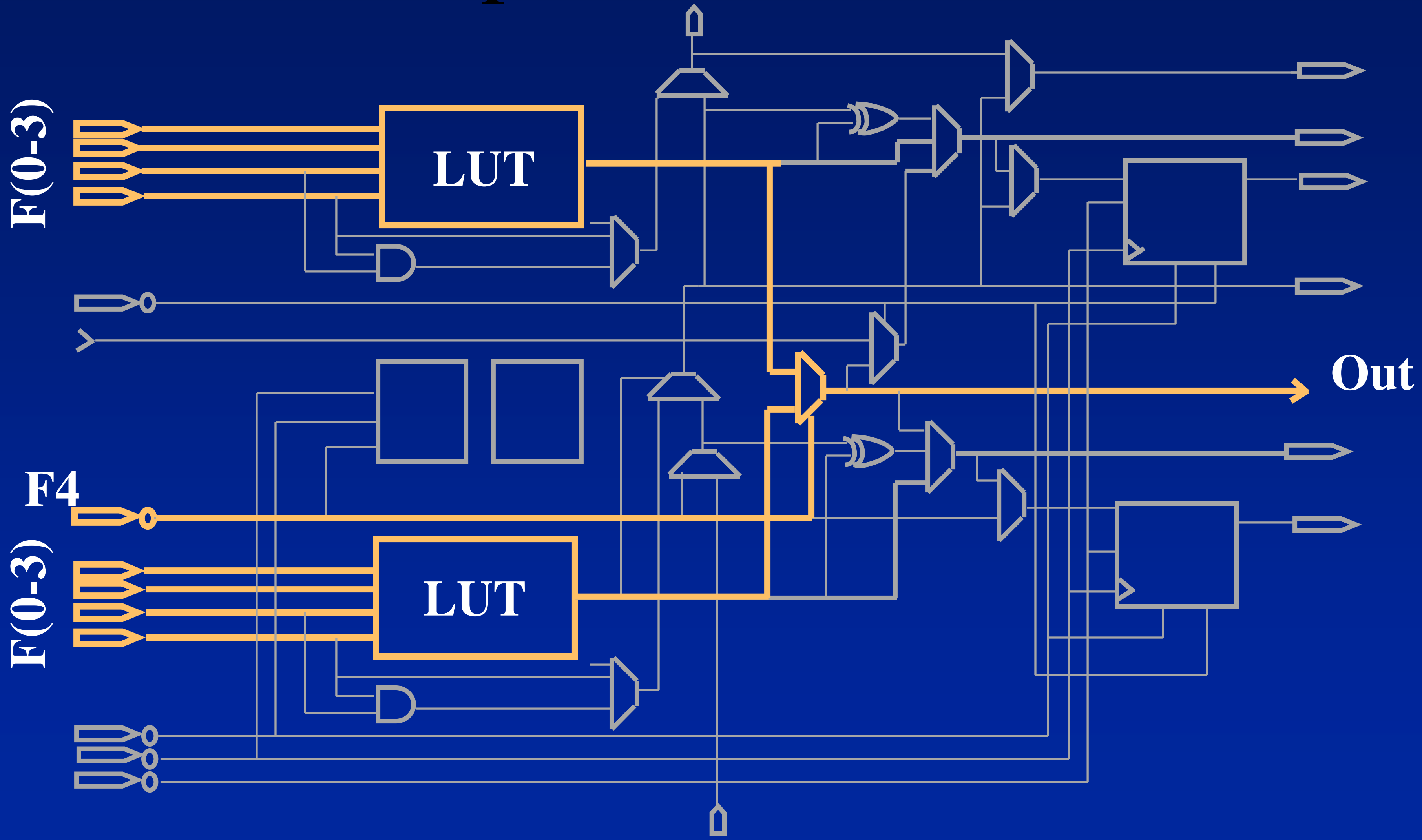
VIRTEX^{TM} SLICE



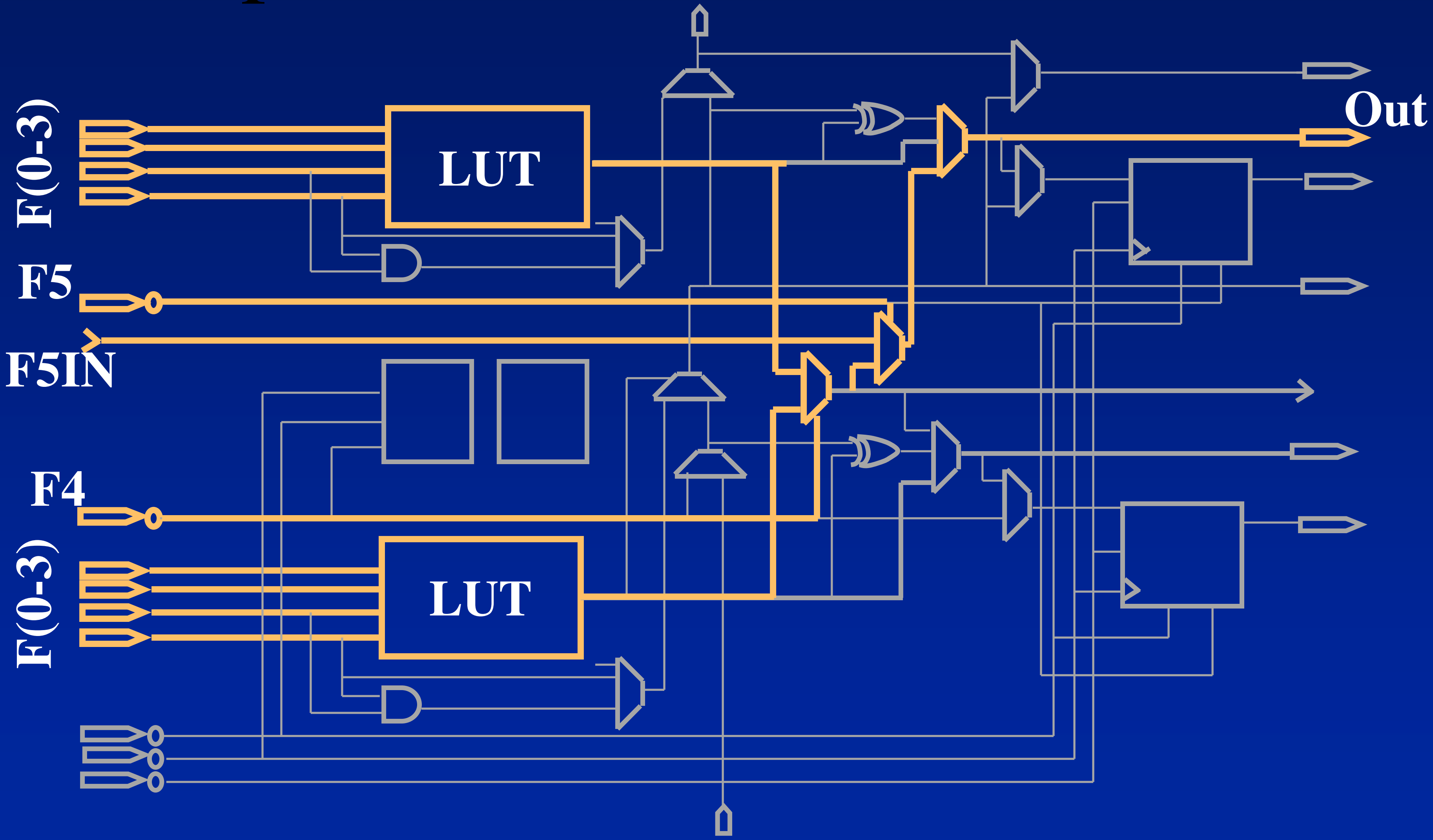
Two 4-input functions



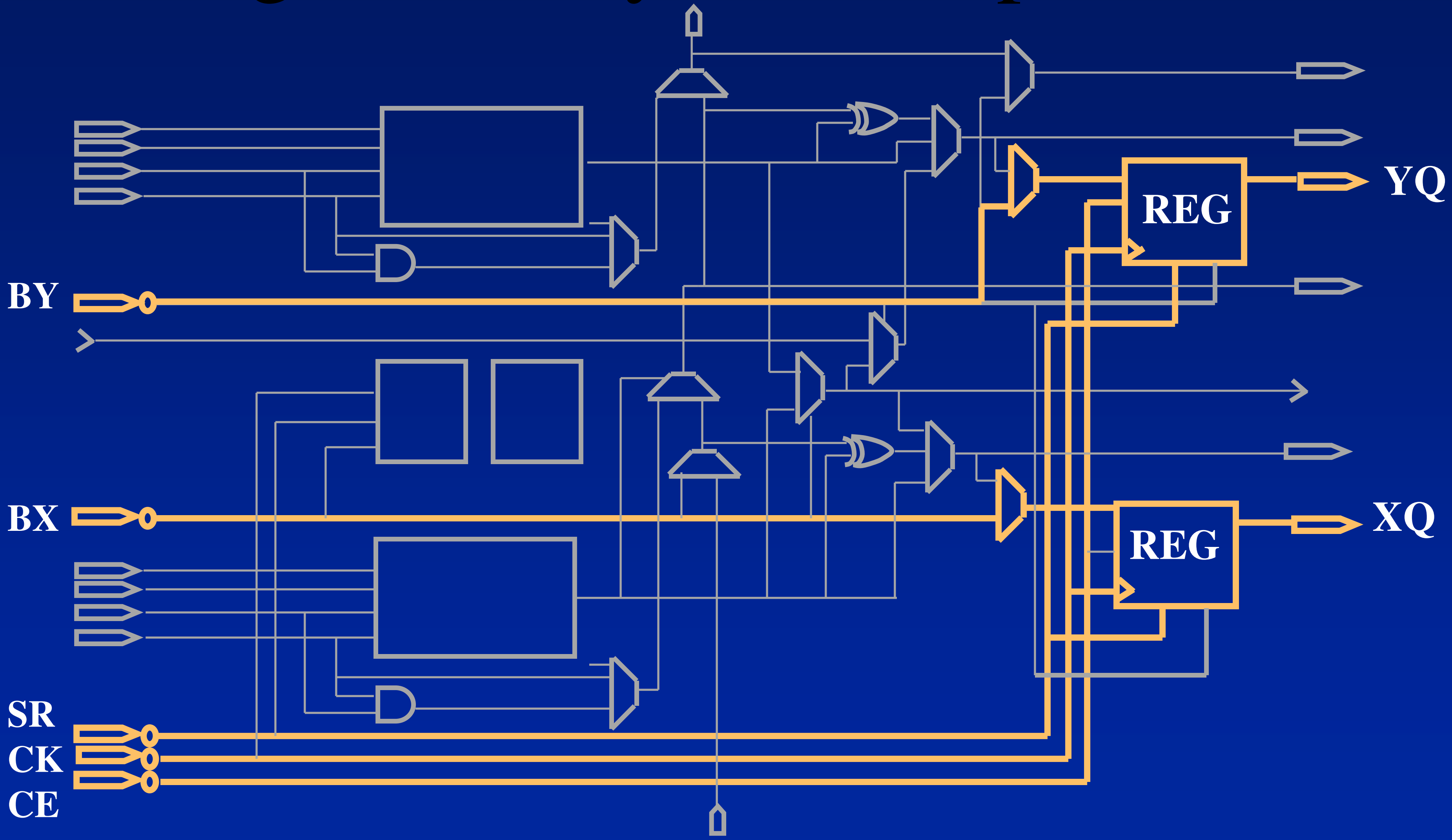
5-input function



6-input function (half shown)

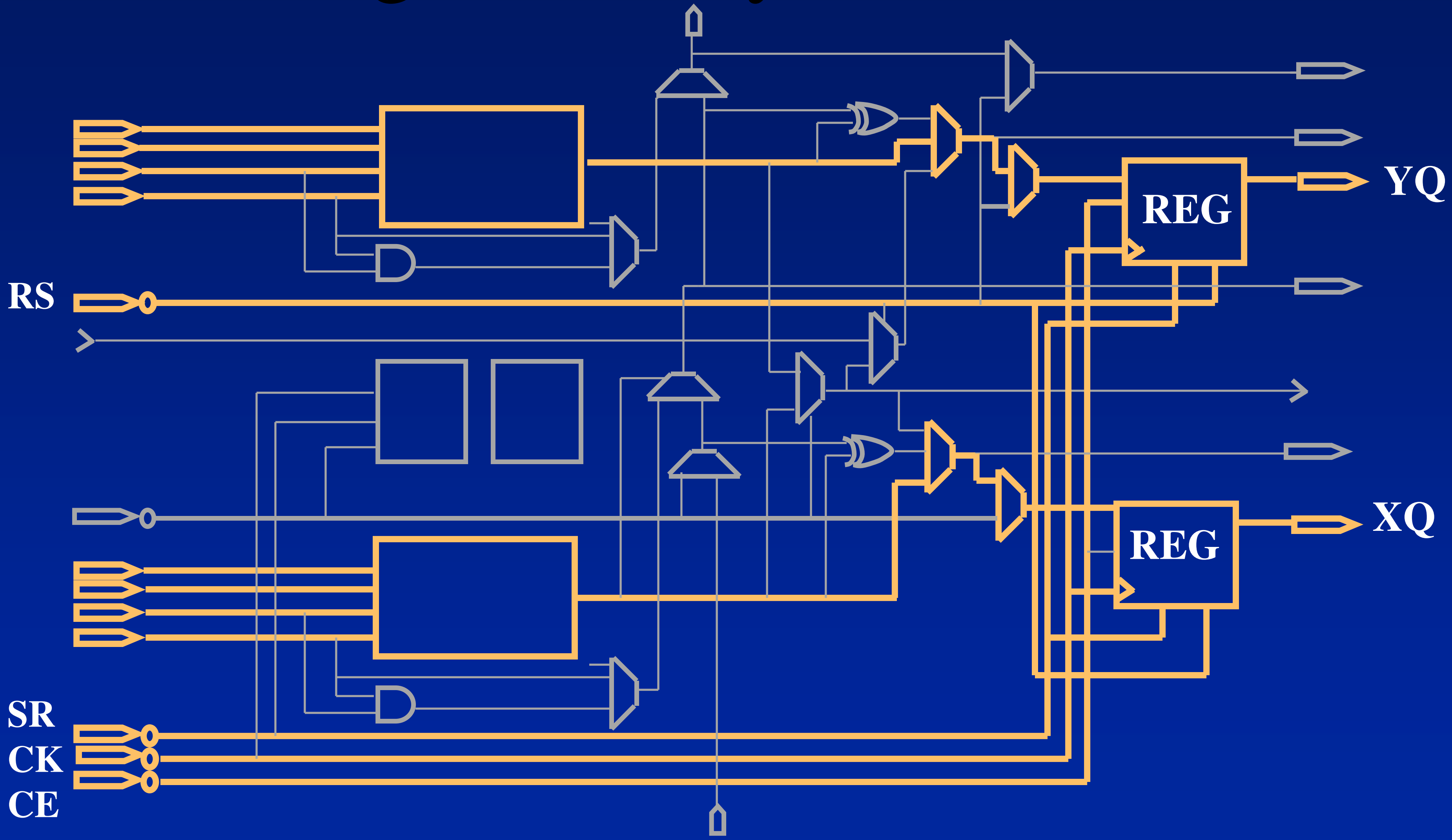


2 reg driven by direct inputs



XILINX

2 reg driven by LUTs



1-bit Full Adder

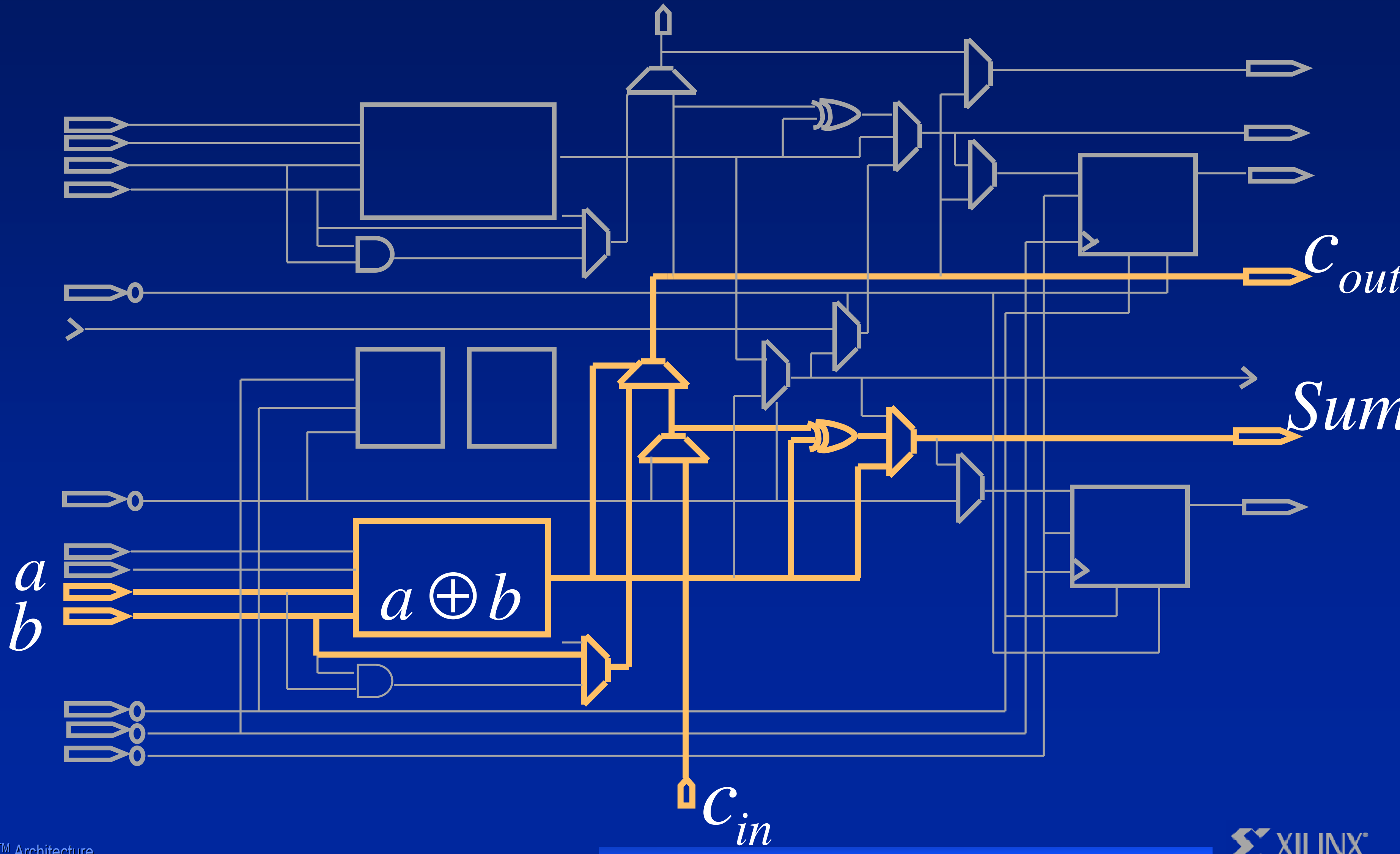
- Adding a and b;   carry in Cin

- $Sum = (a \oplus b) \oplus c_{in}$

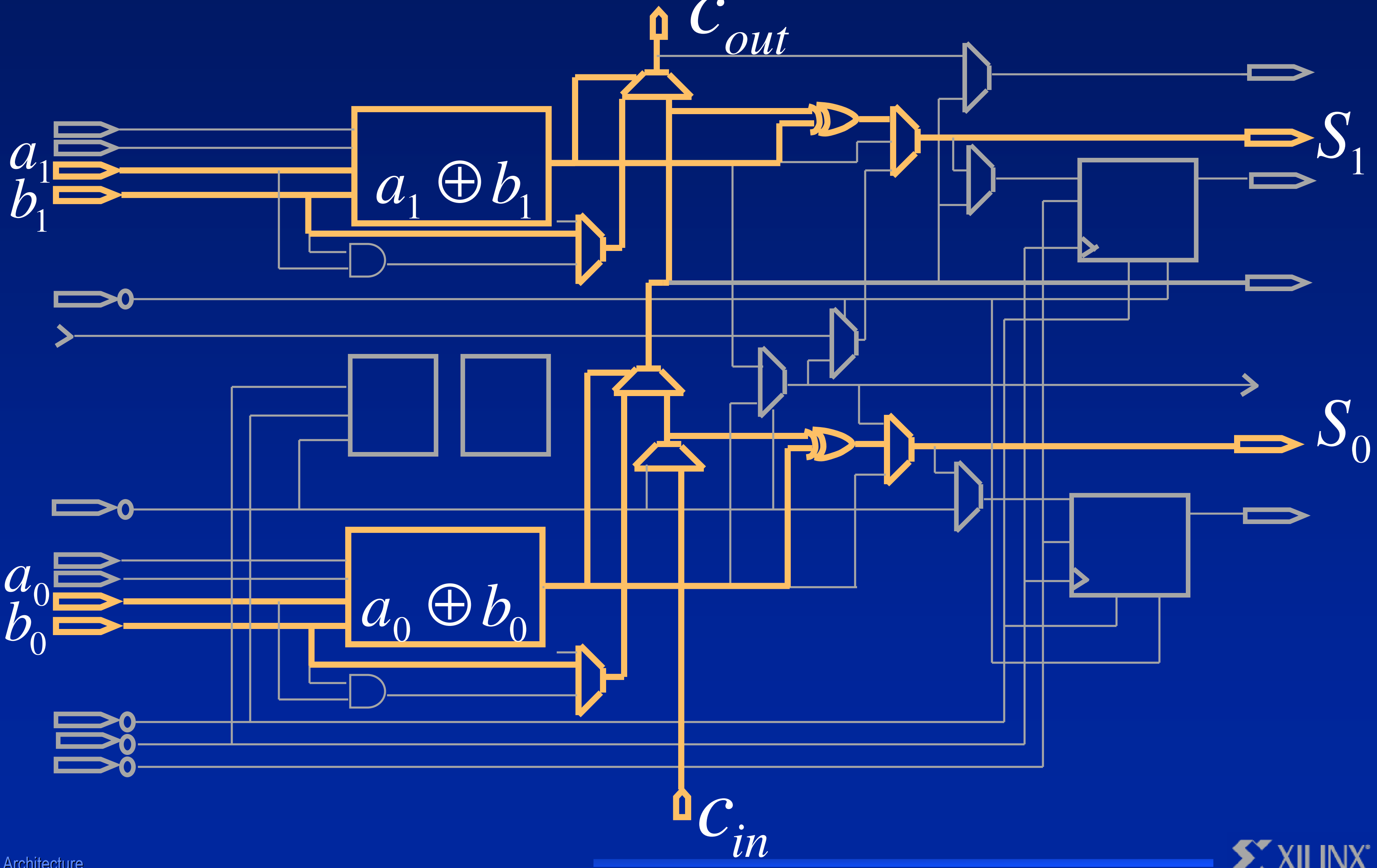
- $c_{out} = ab + ac_{in} + bc_{in}$

$$= (a \oplus b) \cdot c_{in} + \overline{(a \oplus b)} \cdot b$$

1-bit full adder

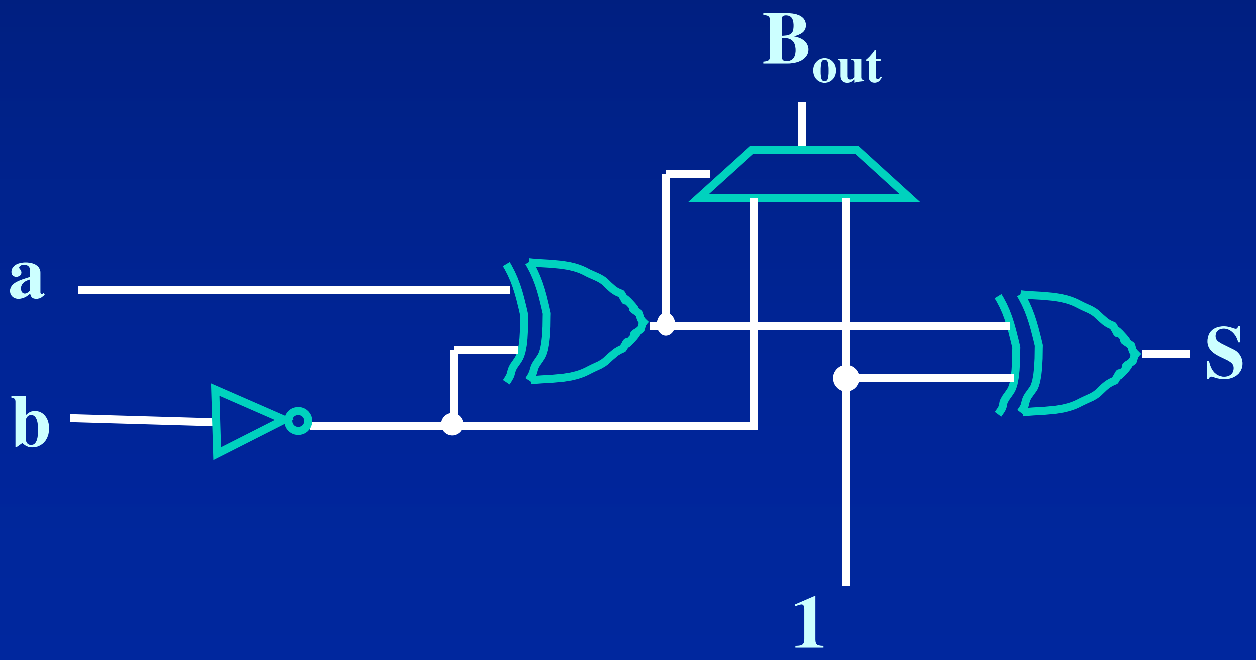


2-bit adder cascaded

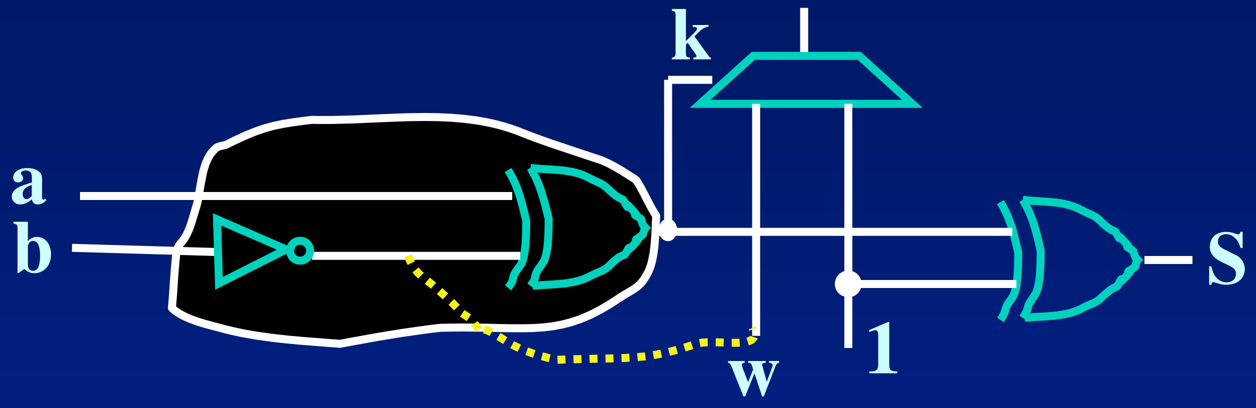


1-bit Subtractor

$$a - b = a + \bar{b} + 1$$

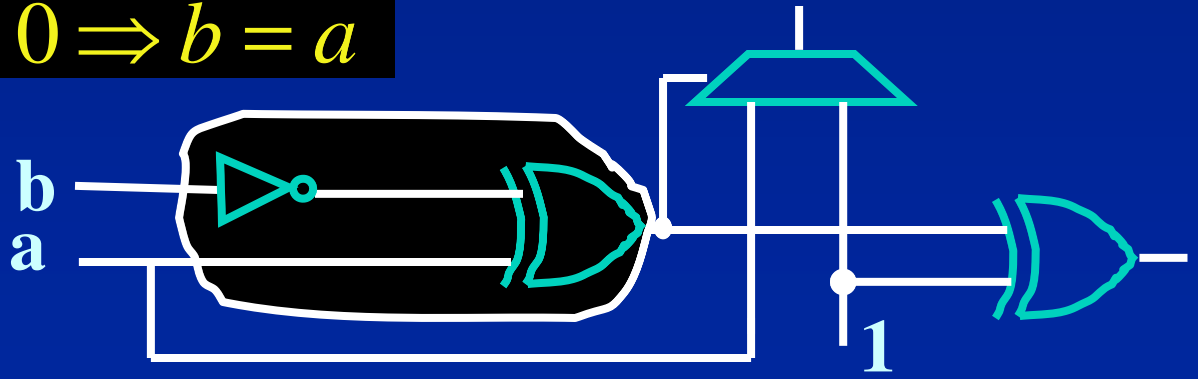


1-bit Subtractor

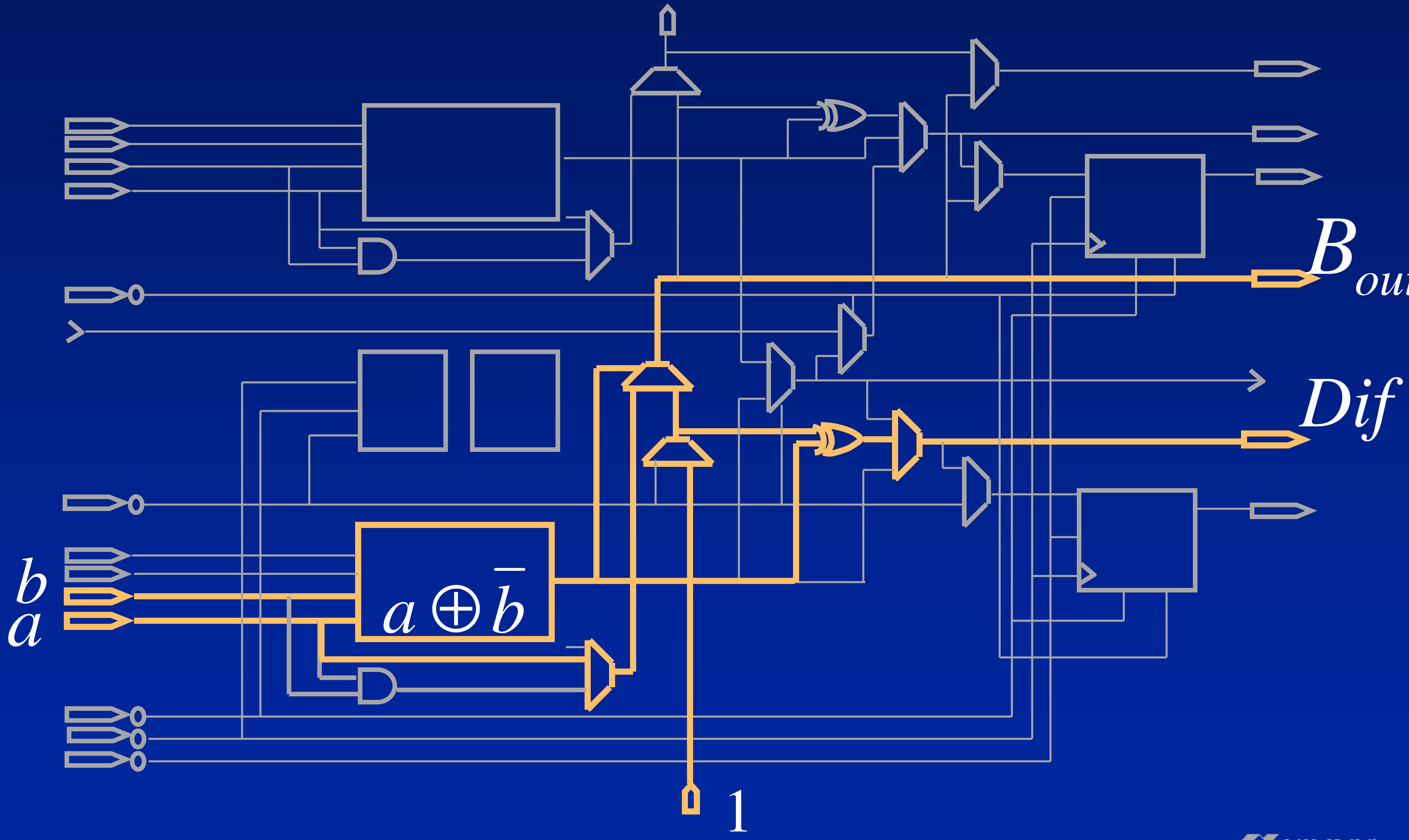


$$when\ (k == 0)\ reqd.\ w = \bar{b}$$

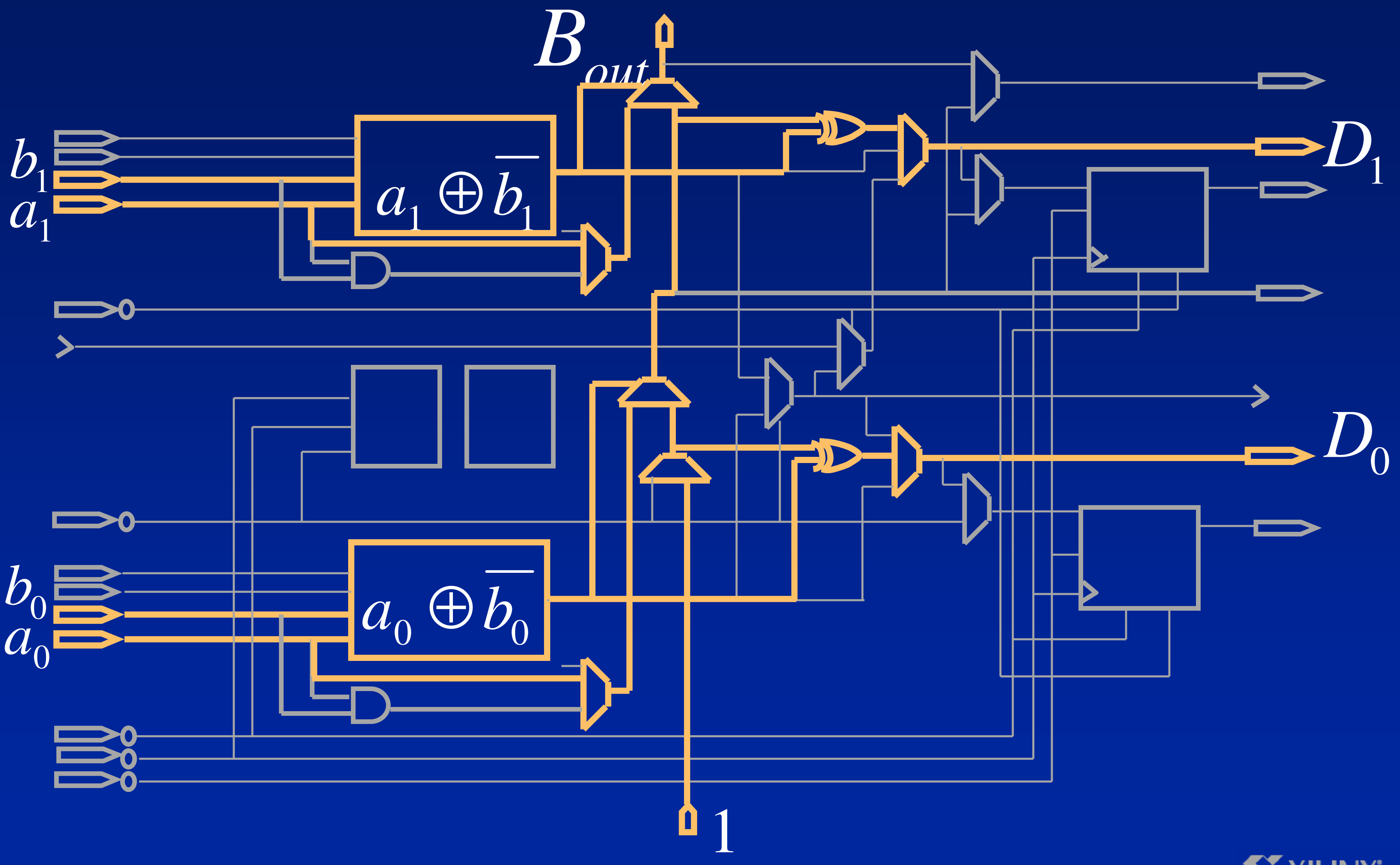
$$k = 0 \Rightarrow a \oplus \bar{b} = 0 \Rightarrow \bar{b} = a$$



1-bit subtractor

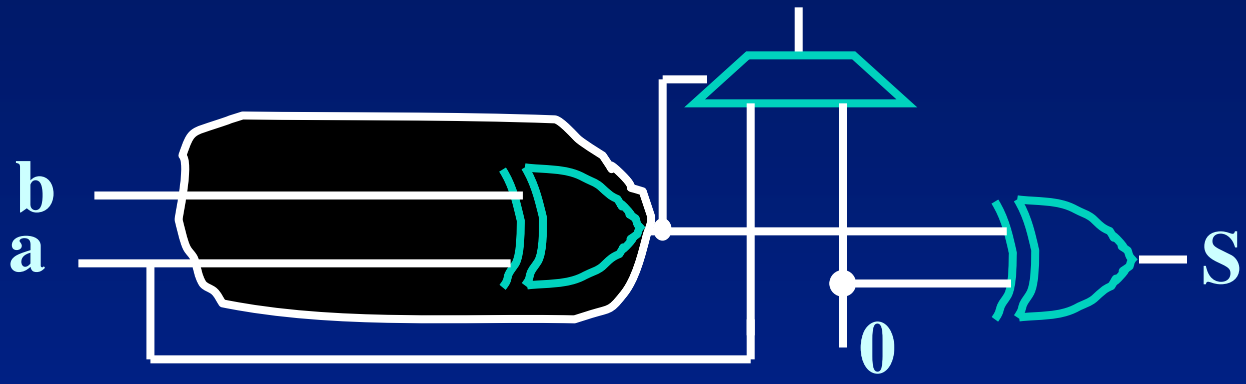


2-bit subtractor

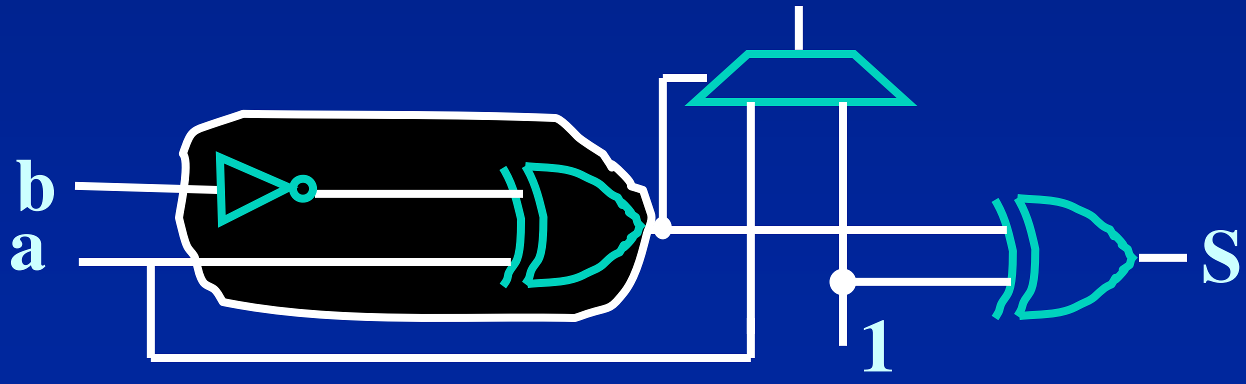


1-bit add-sub

$sa = 0 \Rightarrow a + b$



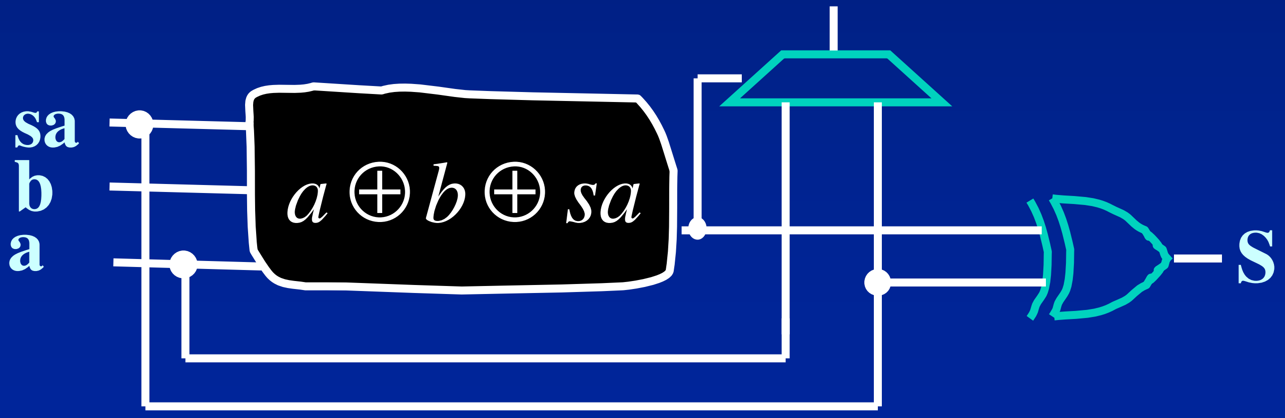
$sa = 1 \Rightarrow a - b$



1-bit add-sub

$sa = 0 \Rightarrow a + b$

$sa = 1 \Rightarrow a - b$



Scaling Accumulator

$$Y = X + 2^{-1}Y$$

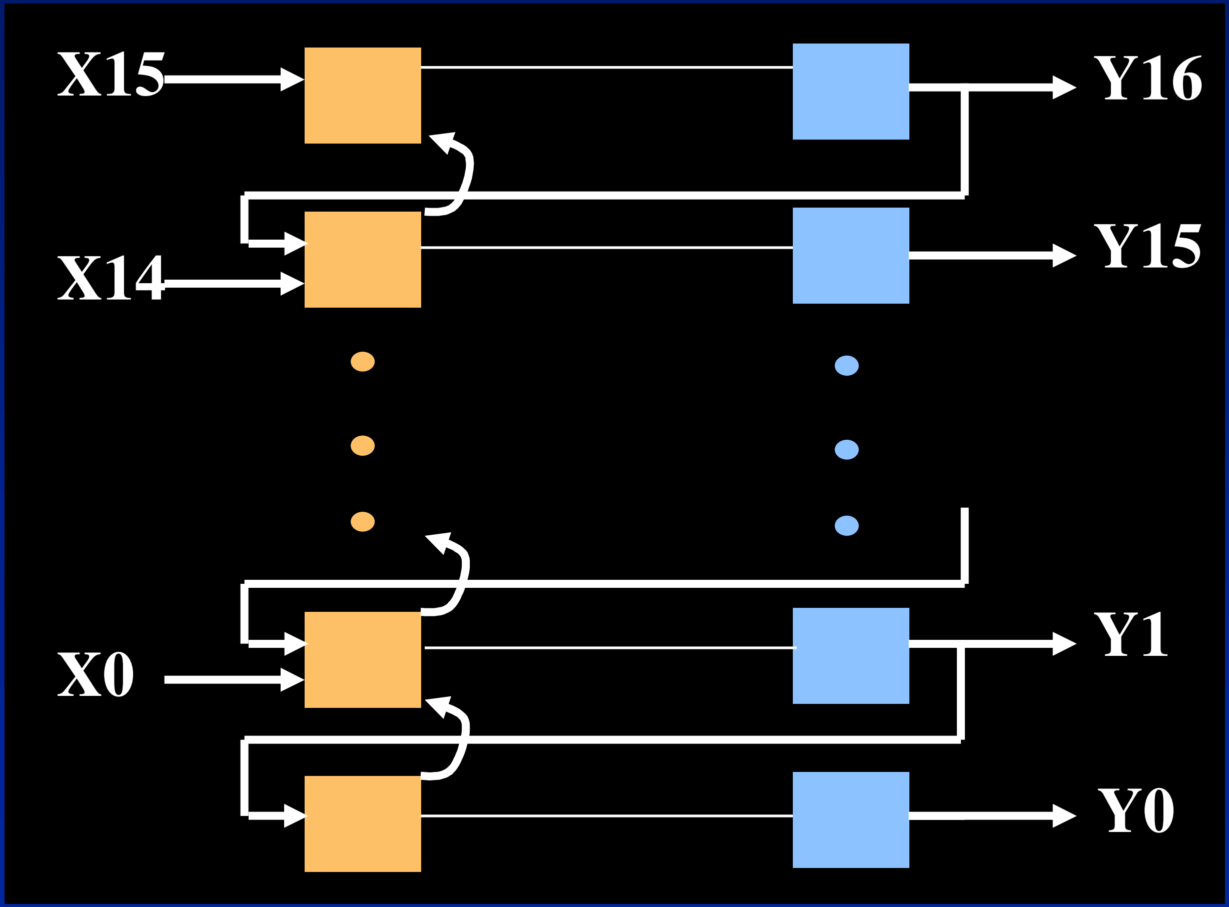
- X : 16-bit input
- Y : 17-bit output

Scaling Accumulator

$$Y = X + 2^{-1} Y$$

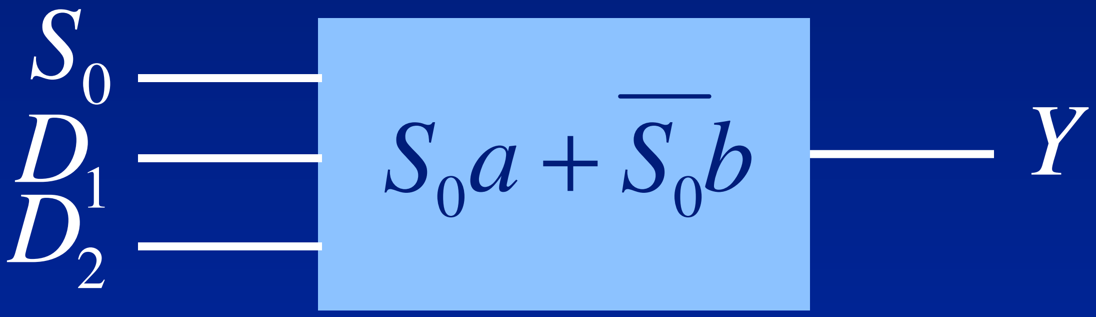
🟧	**2-bit adder**
🟦	**Register**

N-bit scaling acc:
N/2 slices or
N/4 CLBs



2:1 Multiplexer

$$Y = S_0 D_1 + \overline{S_0} D_2$$



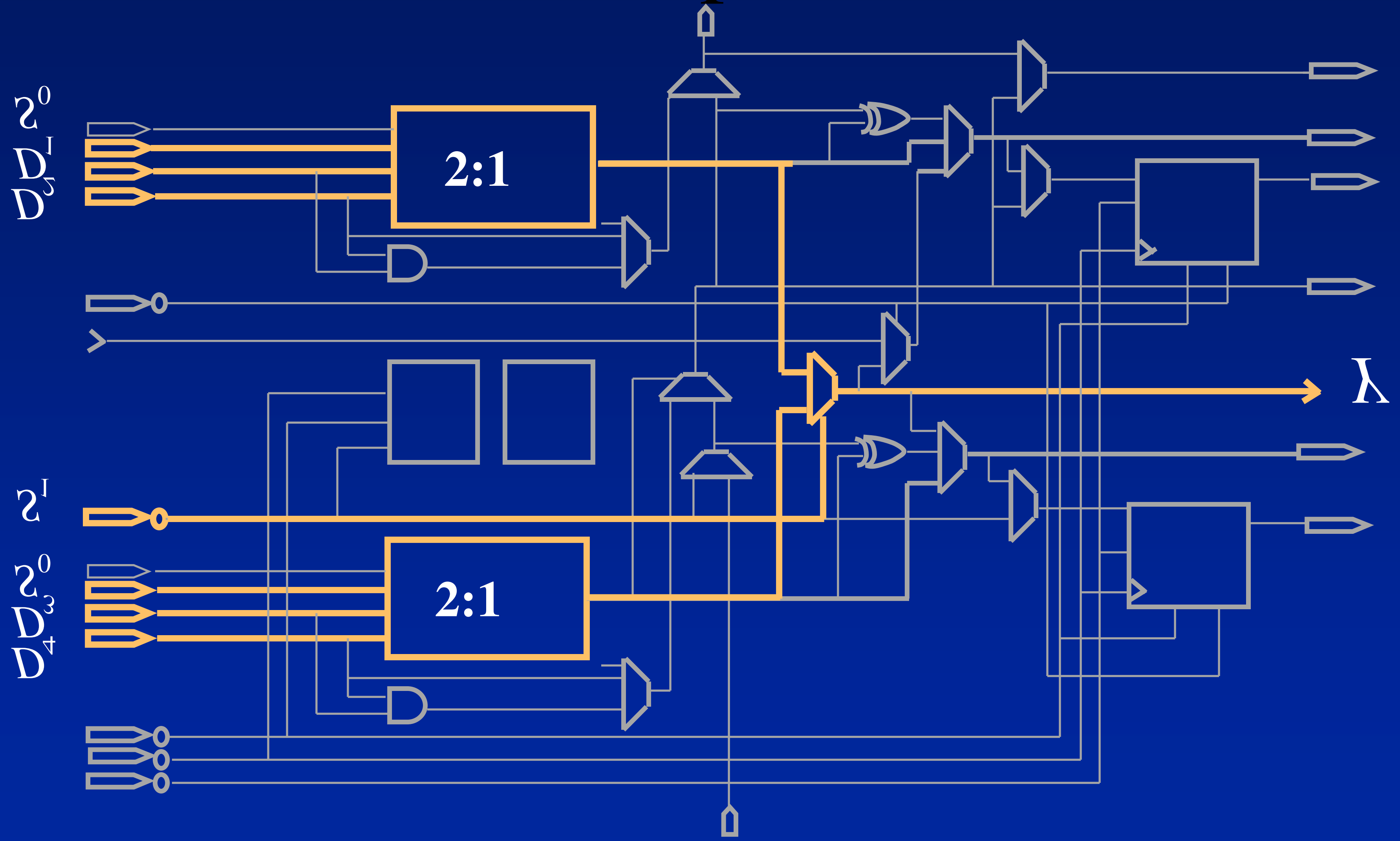
- Requires 1 LUT

4:1 Multiplexer

$$Y = \overline{S_0}\,\overline{S_1}D_1 + S_0\overline{S_1}D_2 + \overline{S_0}S_1D_3 + S_0S_1D_4$$

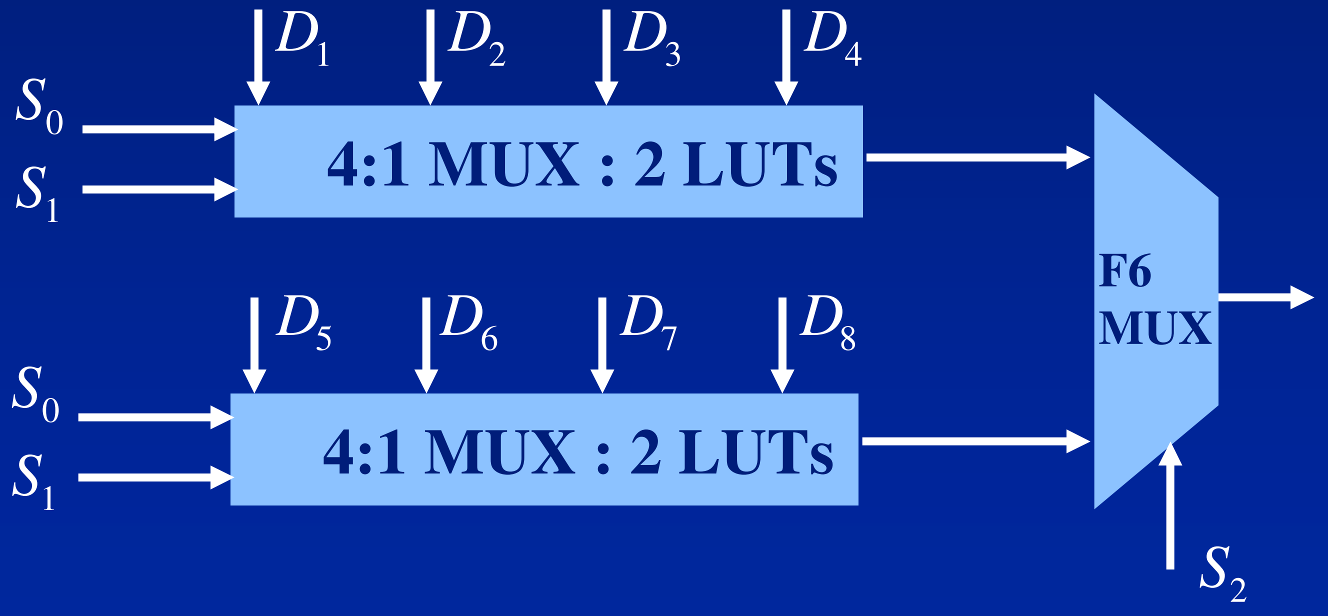
- 6 input function
- naïve implementation will require 4 LUTs
- exploit partitioning of variables
  — can implement using just 2 LUTs (1 slice)

4:1 Multiplexer



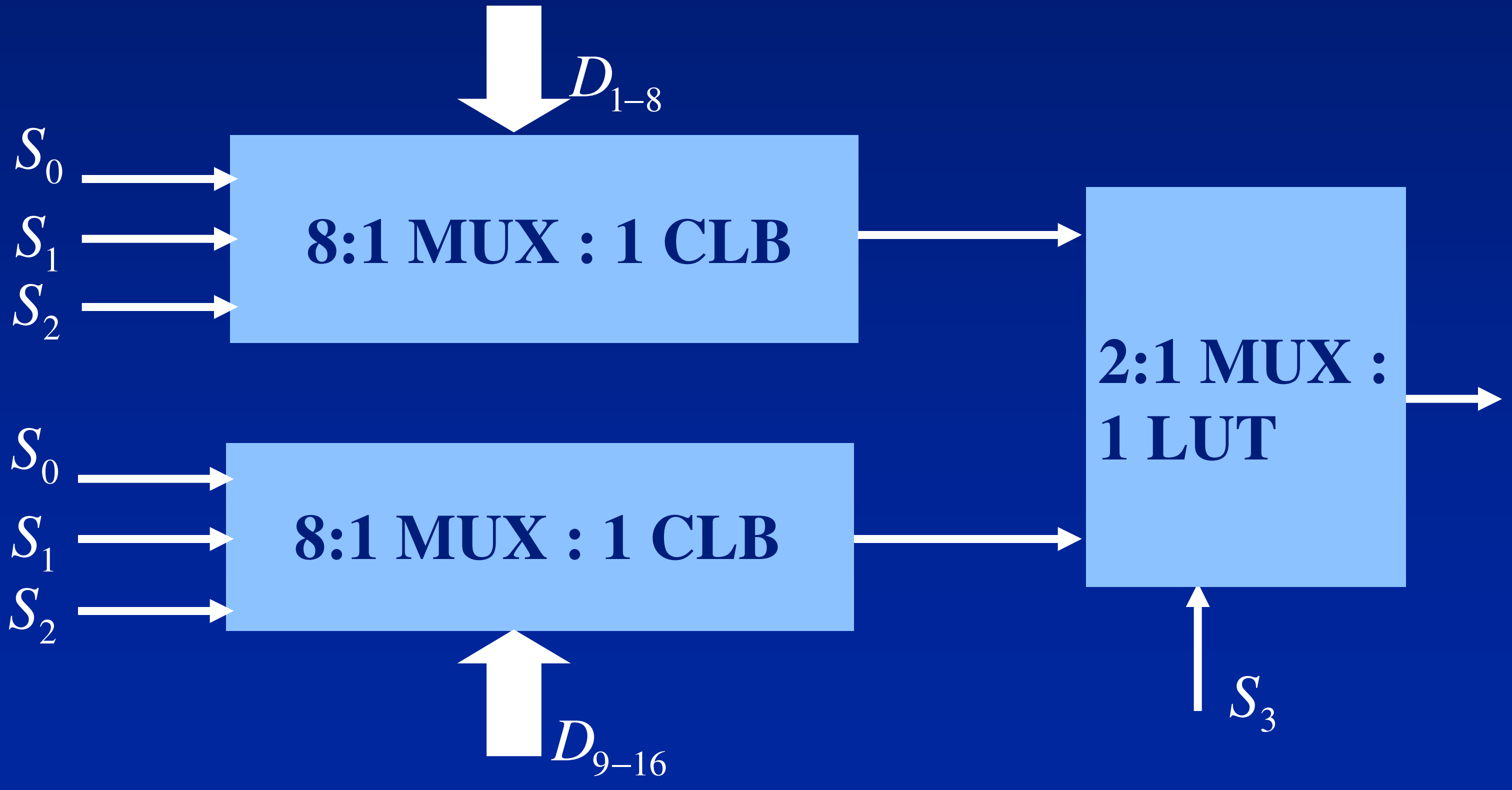
8:1 Multiplexer

- 11 input function
- can be implemented with 4 LUTs (1 CLB)

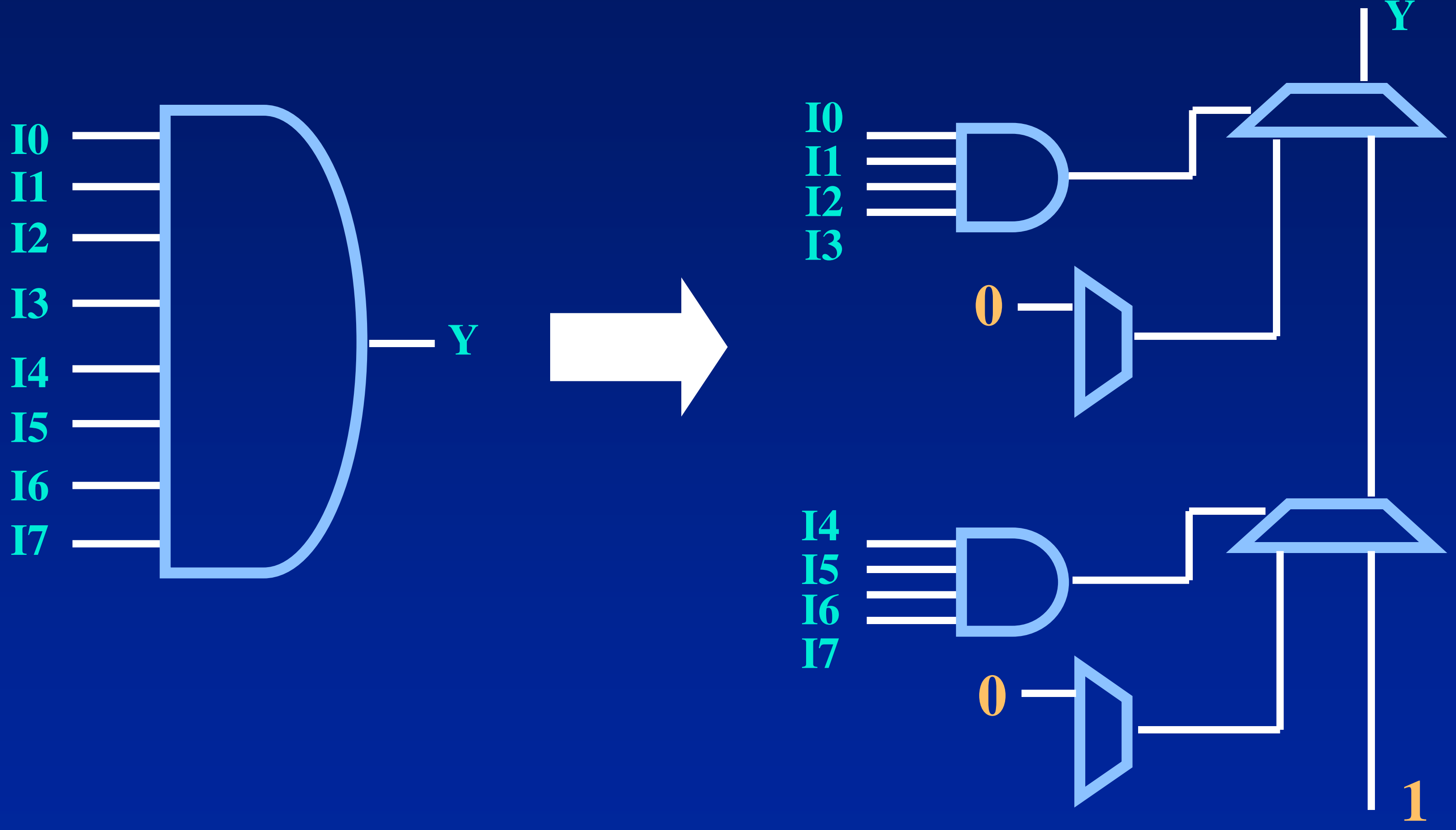


16:1 Multiplexer

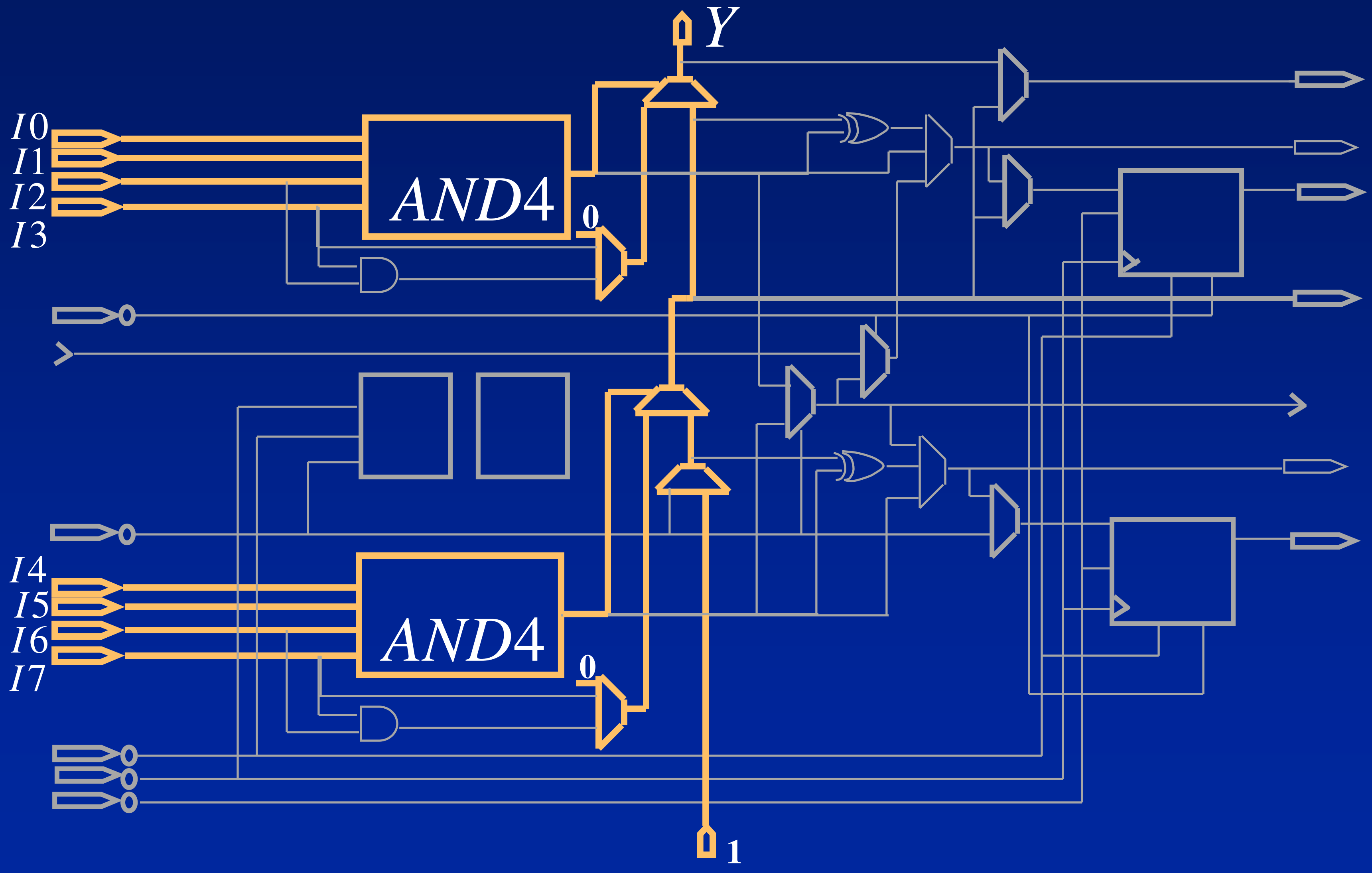
- Build larger muxes using [8:1] [4:1] [2:1] modules



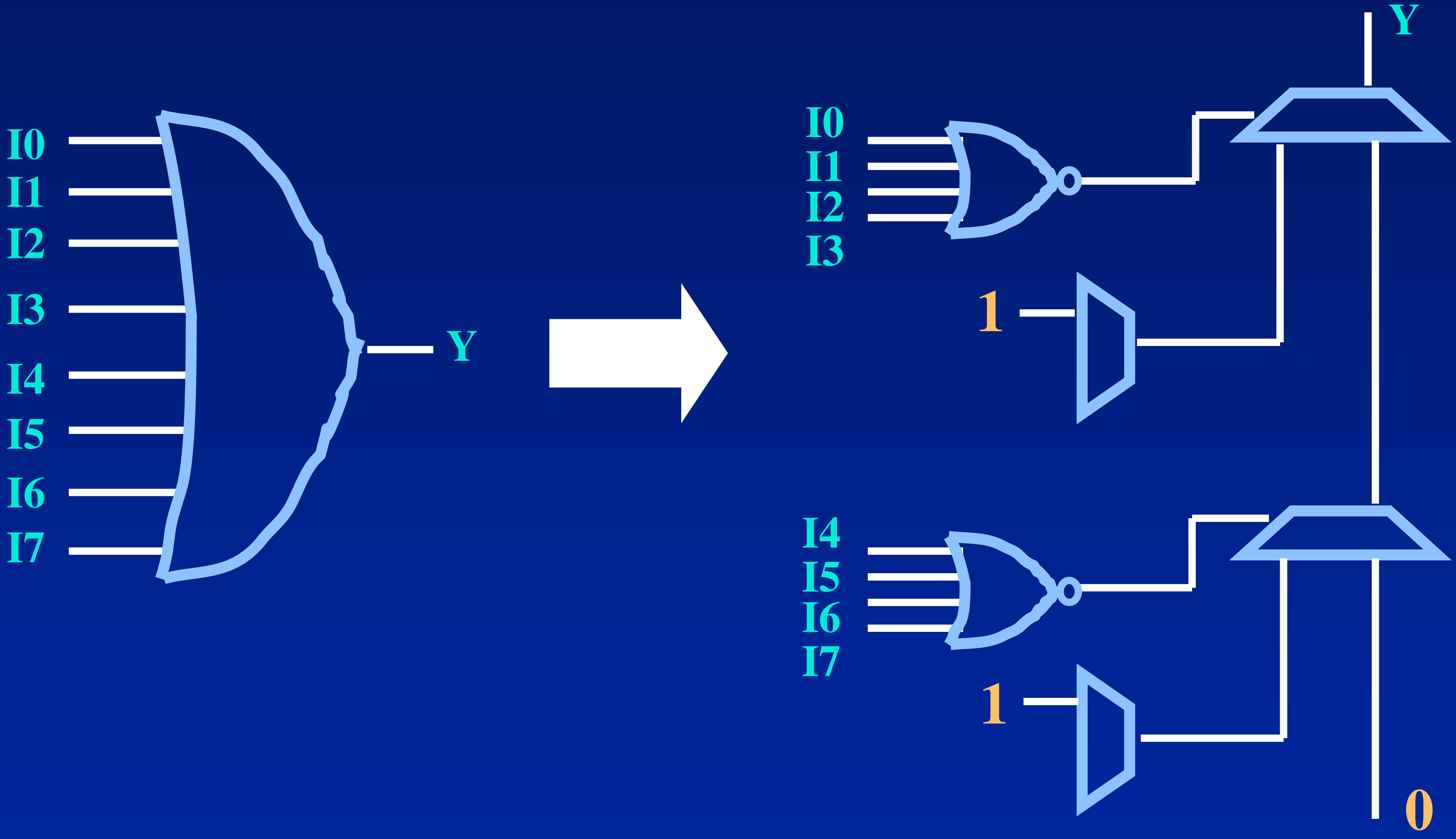
Wide-AND



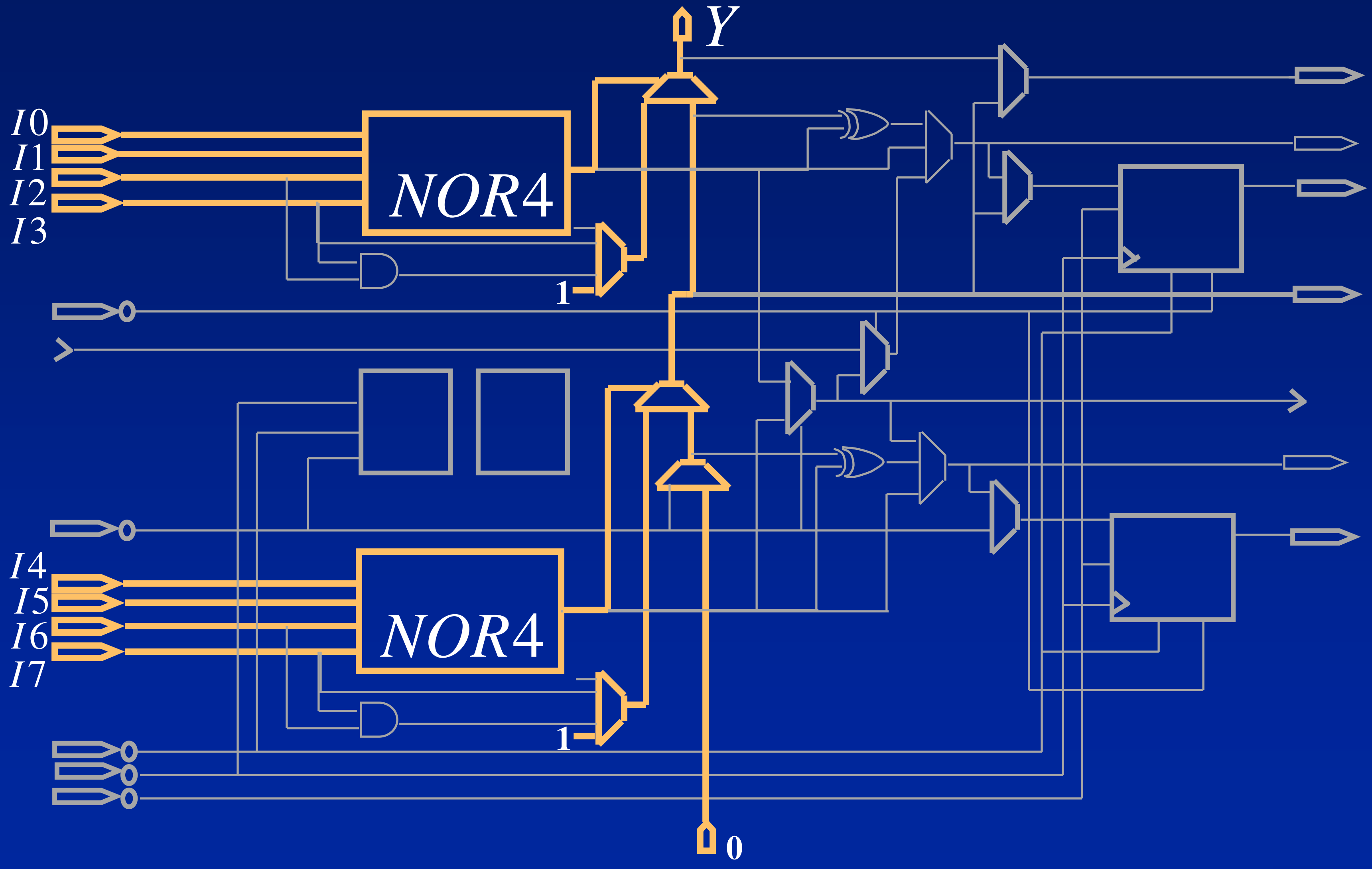
8-bit Wide-AND



Wide-OR



8-bit Wide-OR



Multiplier

$$Y = a \times b$$

- a : 16-bit multiplicand
- b : 16-bit multiplier
- Y : 32-bit output

Multiplier

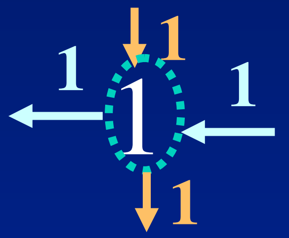
$$
\begin{array}{r}
a_N \; a_{N-1} \; \ldots \; a_2 \; a_1 \; a_0 \\
\times \qquad b_M \; b_{M-1} \; \ldots \; b_2 \; b_1 \; b_0 \\
\hline
a_N b_0 \; \ldots \; a_2 b_0 \; a_1 b_0 \; a_0 b_0 \\
a_N b_1 \; \ldots \; a_2 b_1 \; a_1 b_1 \; a_0 b_1 \quad\;\; \\
\vdots \qquad\qquad\qquad\qquad\qquad\qquad \\
a_N b_M \; \ldots \; a_2 b_M \; a_1 b_M \; a_0 b_M \qquad\qquad\qquad \\
\hline
\end{array}
$$

Multiplier

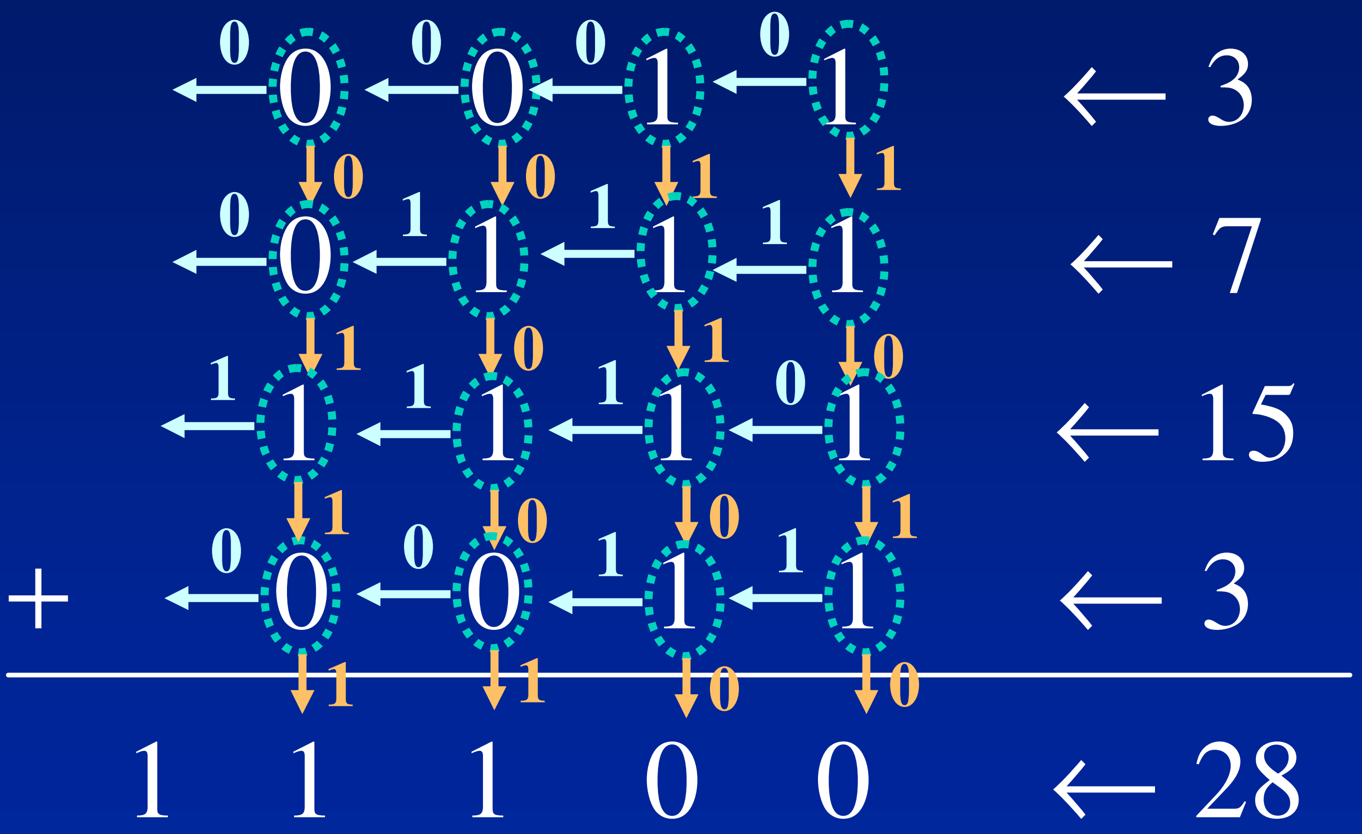
$$\begin{array}{ccccccl}
 & 1 & 11 & 111 & 11 & & \\
 & & 0 & 0 & 1 & 1 & \leftarrow 3 \\
 & & 0 & 1 & 1 & 1 & \leftarrow 7 \\
 & & 1 & 1 & 1 & 1 & \leftarrow 15 \\
+ & & 0 & 0 & 1 & 1 & \leftarrow 3 \\
\hline
 & 1 & 1 & 1 & 0 & 0 & \leftarrow 28
\end{array}$$

Multiplier

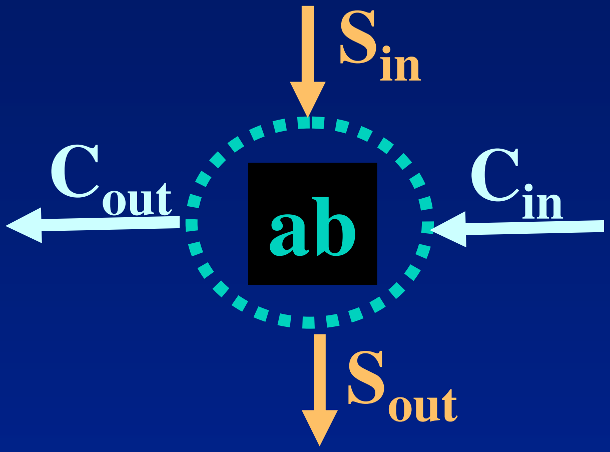
$$\begin{array}{cccccccl}
 & & 0 & 0 & 1 & 1 & & \leftarrow 3 \\
 & & 0 & 1 & 1 & 1 & & \leftarrow 7 \\
 & & 1 & 1 & 1 & 1 & & \leftarrow 15 \\
+ & & 0 & 0 & 1 & 1 & & \leftarrow 3 \\
\hline
 & 1 & 1 & 1 & 0 & 0 & & \leftarrow 28
\end{array}$$



Multiplier



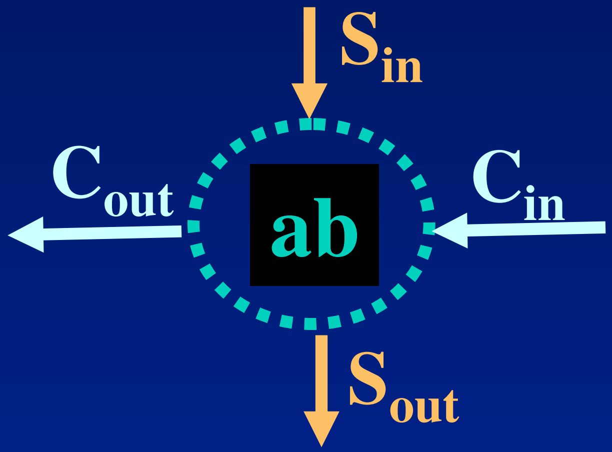
Multiplier

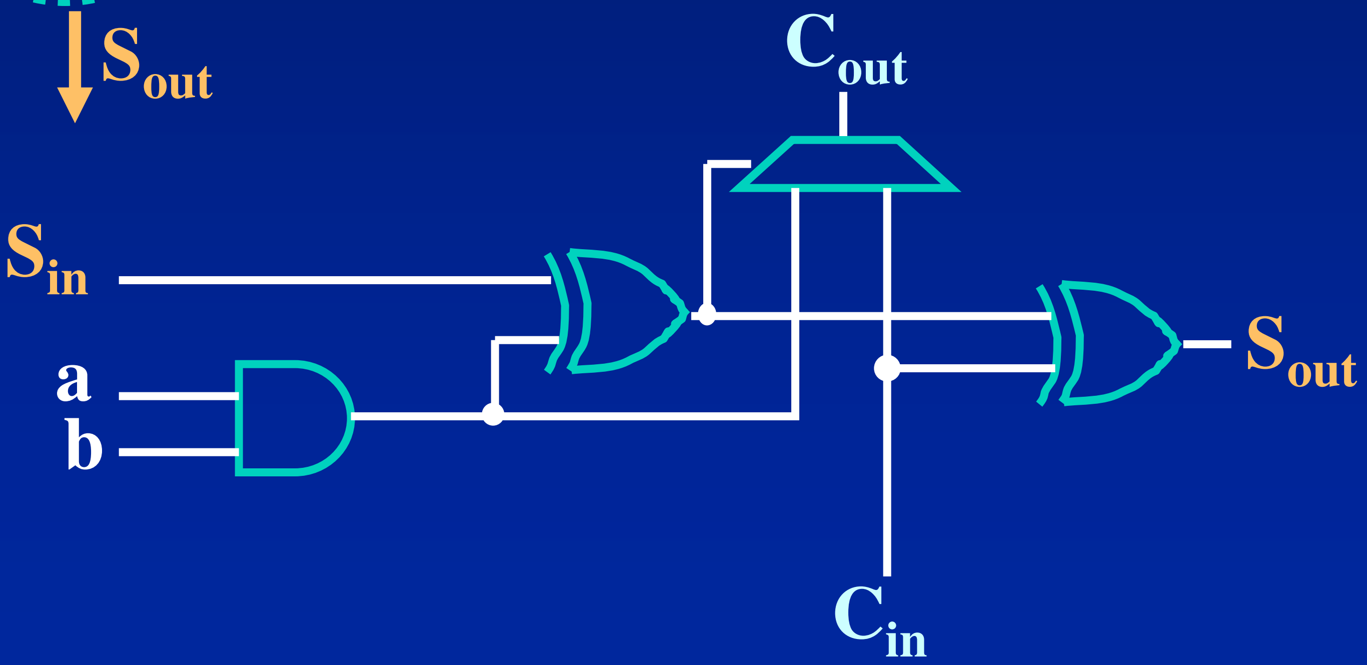


$$S_{out} = \left(ab \oplus S_{in}\right) \oplus c_{in}$$

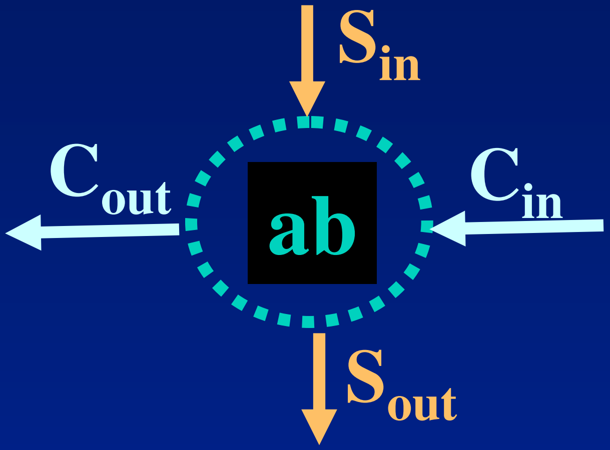
$$C_{out} = abS_{in} + S_{in}C_{in} + abC_{in}$$

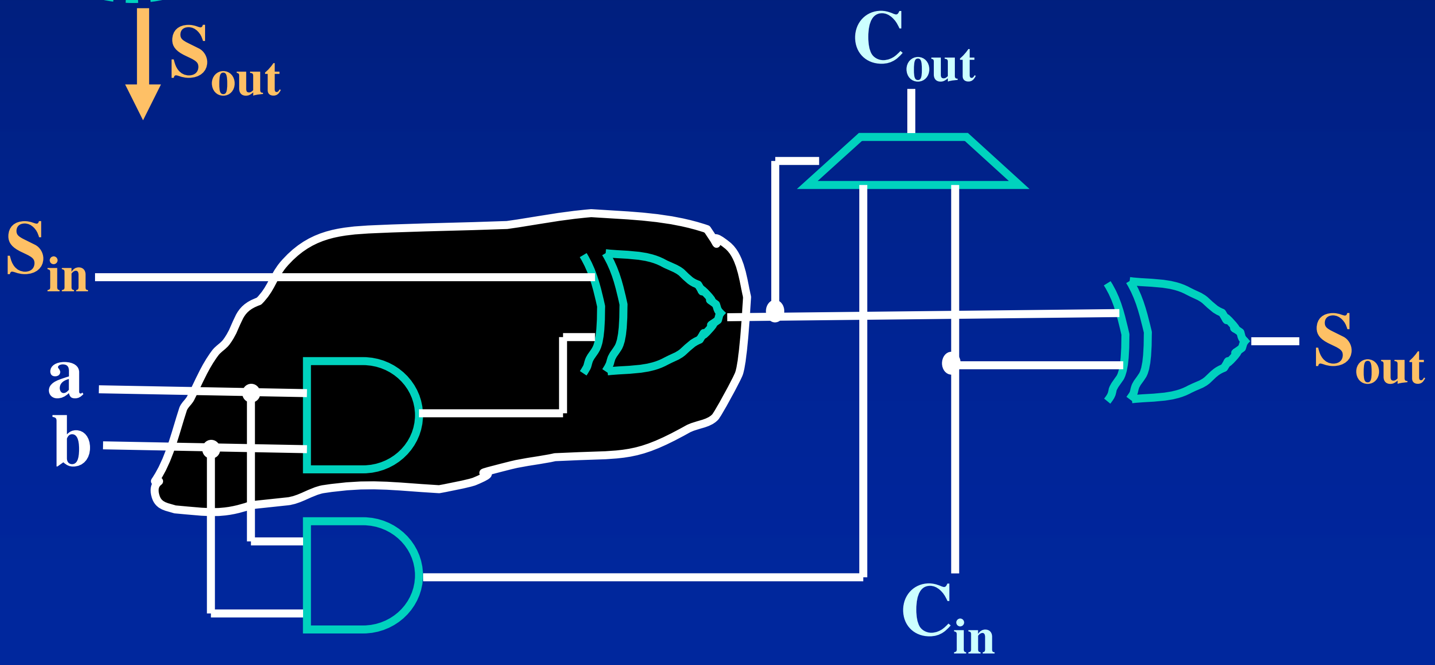
Multiplier



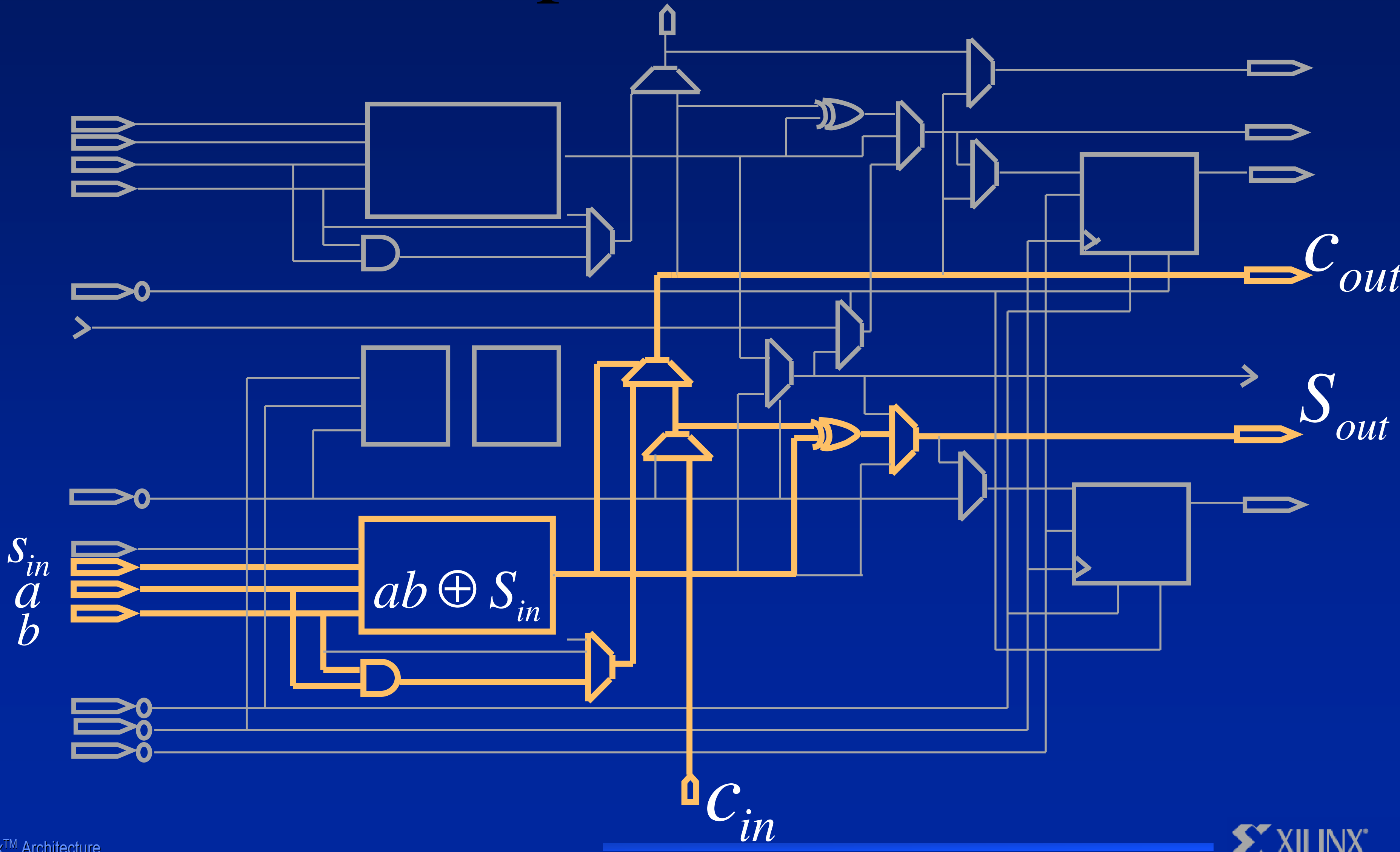


Multiplier

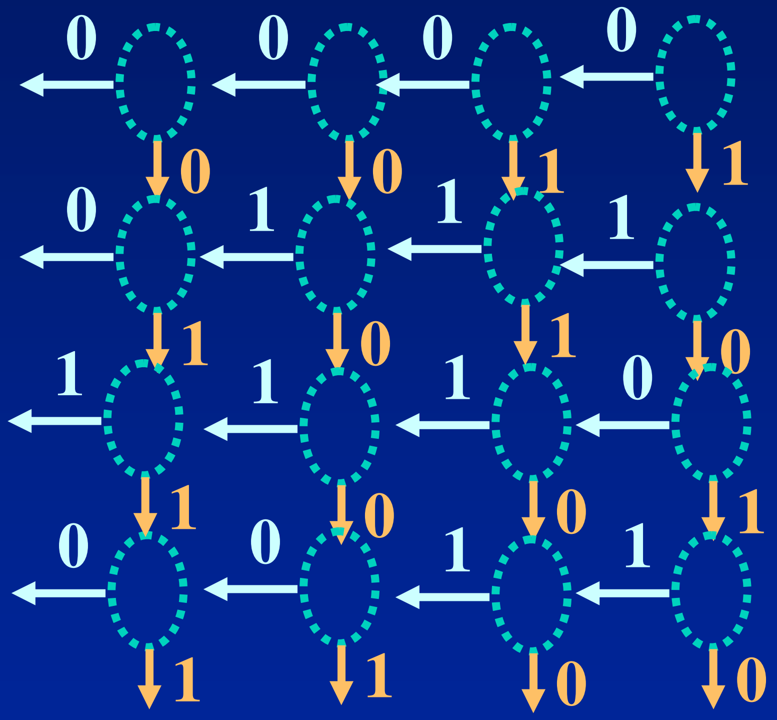




Multiplier core



N-bit Multiplier



- Area: N^2 LUTs

  $= N^2/2$ slices

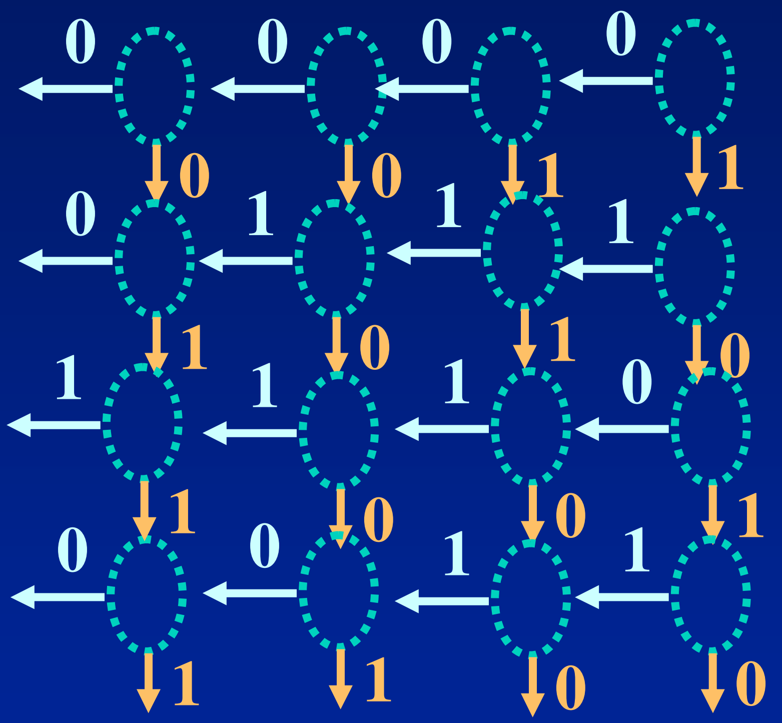
  $= N^2/4$ CLBs

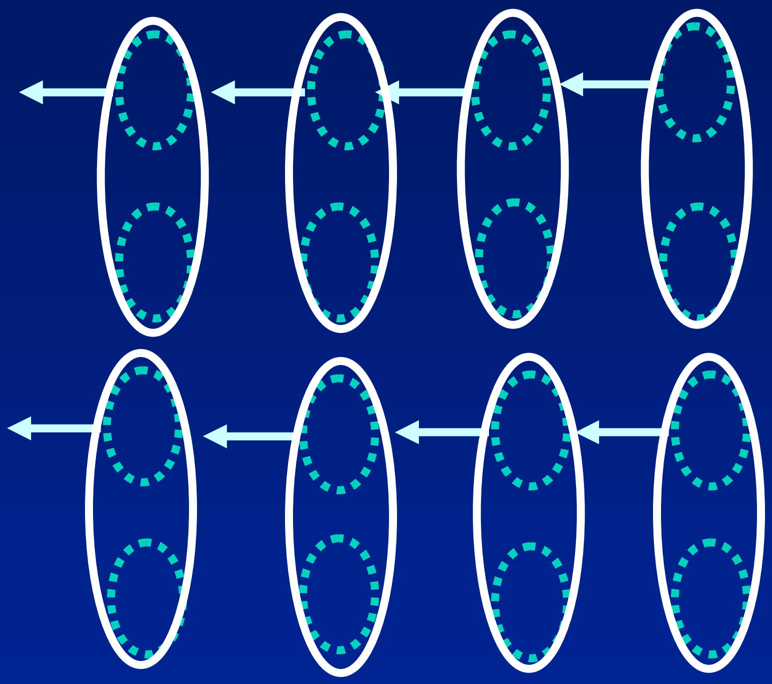
- Speed : $2Nc + Ns$

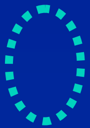
  c = carry delay = 0.1ns

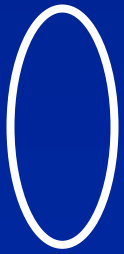
  s = slice delay = 3ns

N-bit Multiplier (efficient)



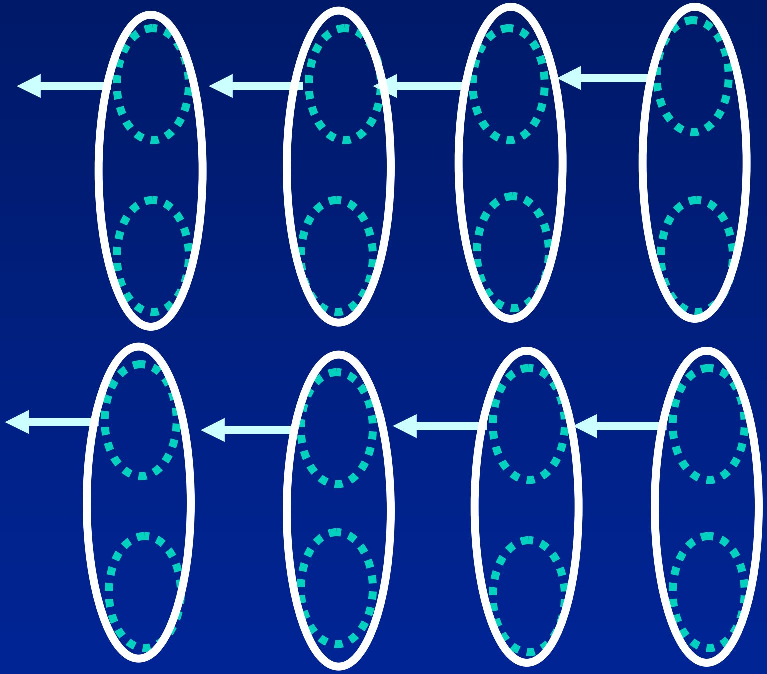


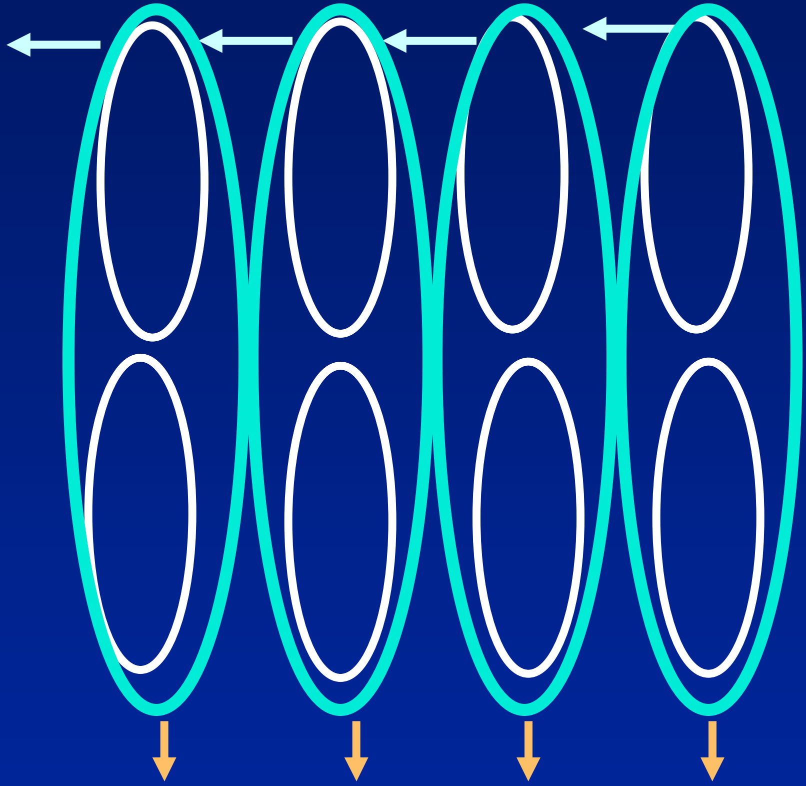
 $ab + S_{in} + c_{in}$

$ab + cd + c_{in}$ 

XILINX

N-bit Multiplier (efficient)

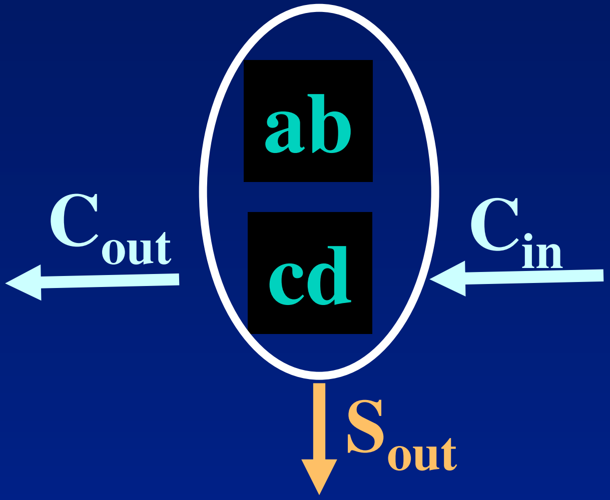


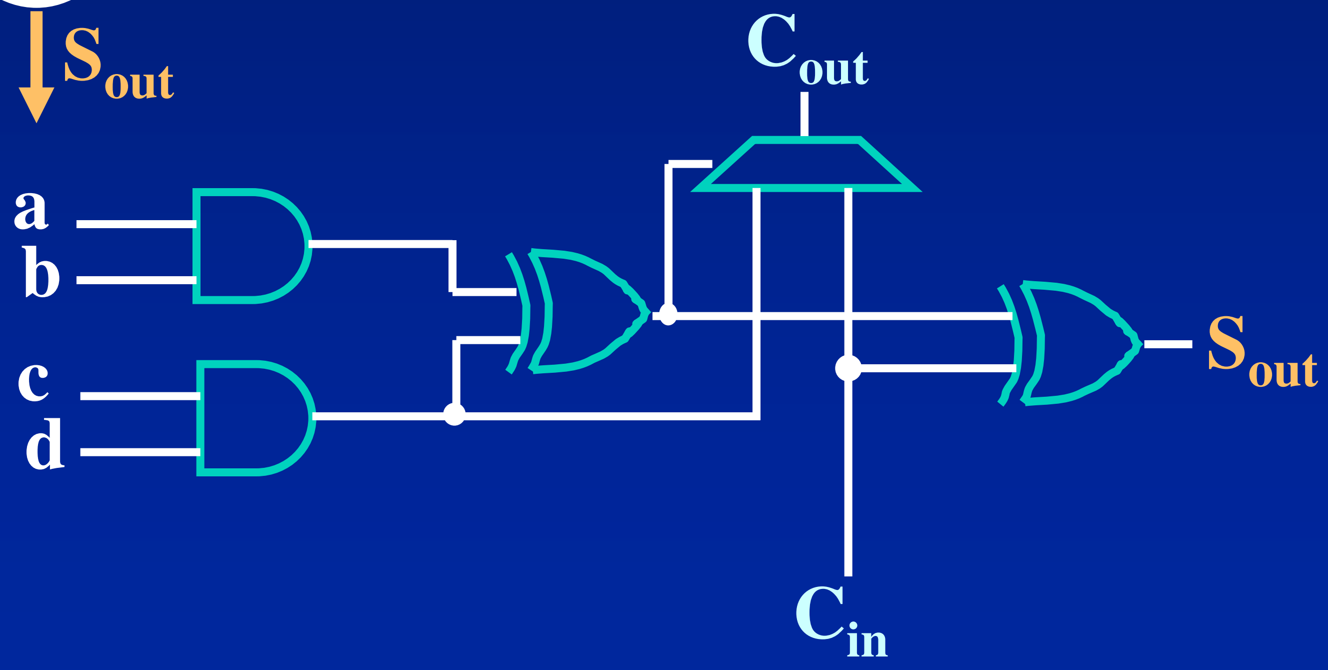


○ $ab+cd+c_{in}$

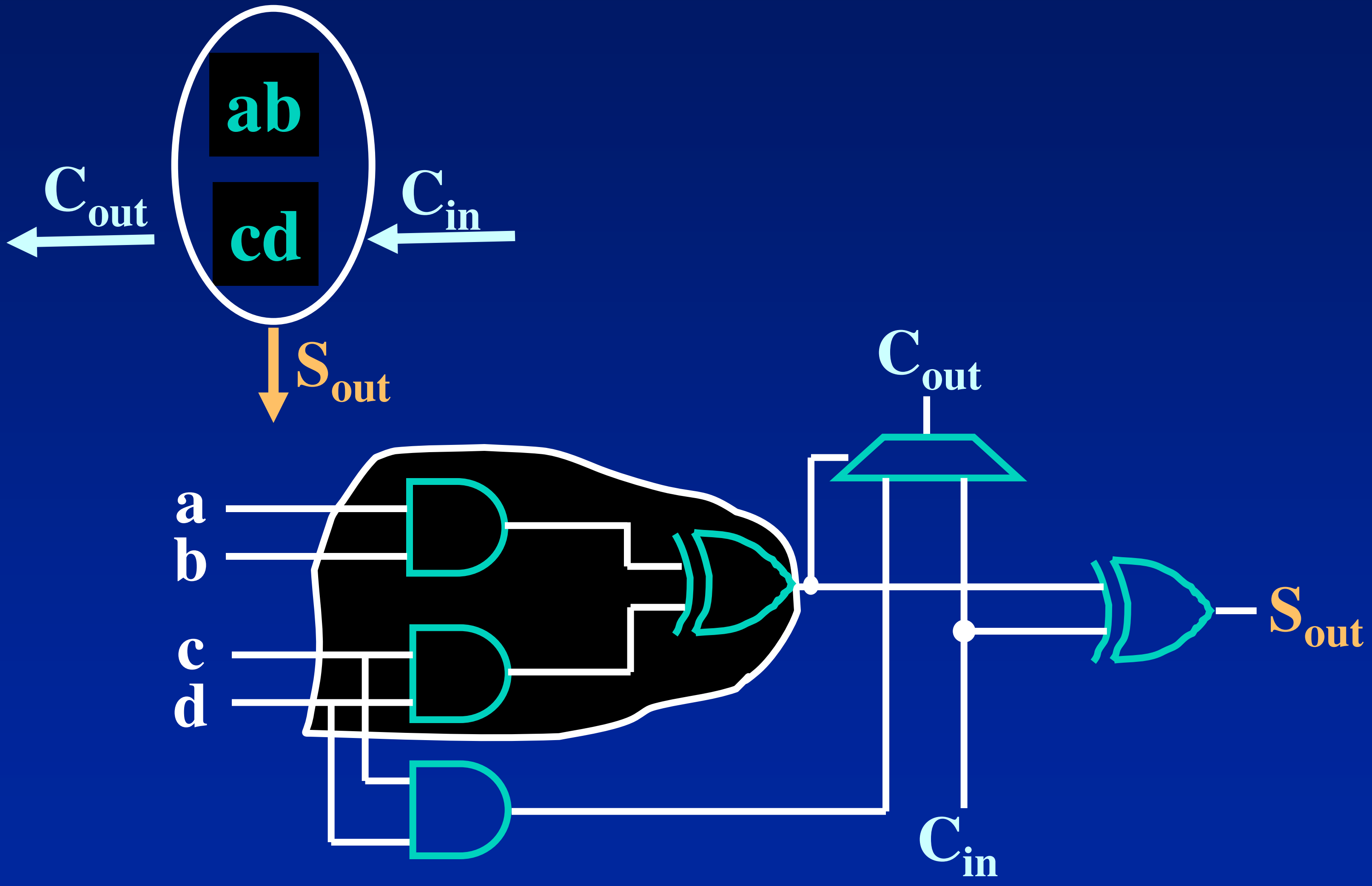
$a+b+c_{in}$ ○

Multiplier (efficient)

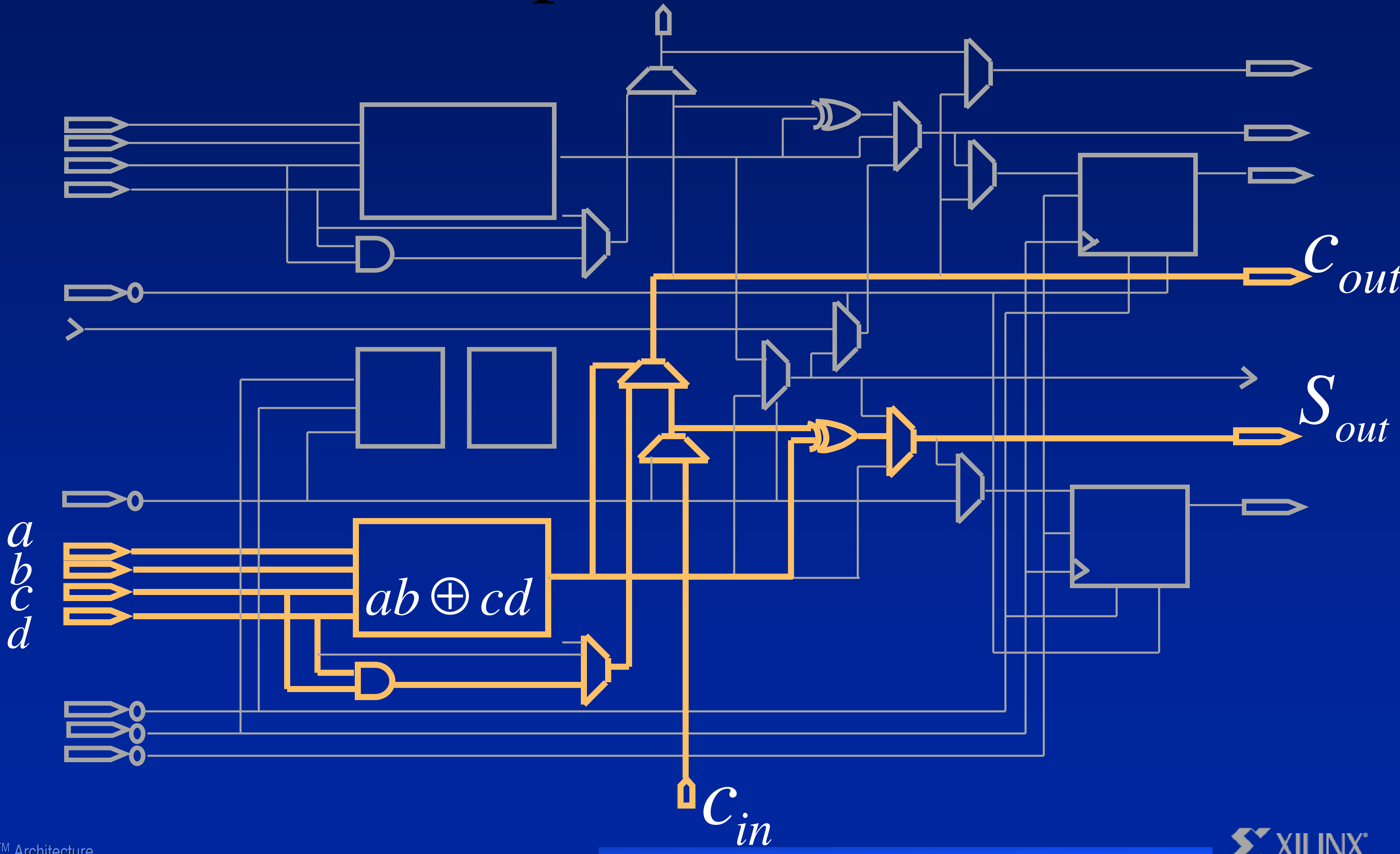




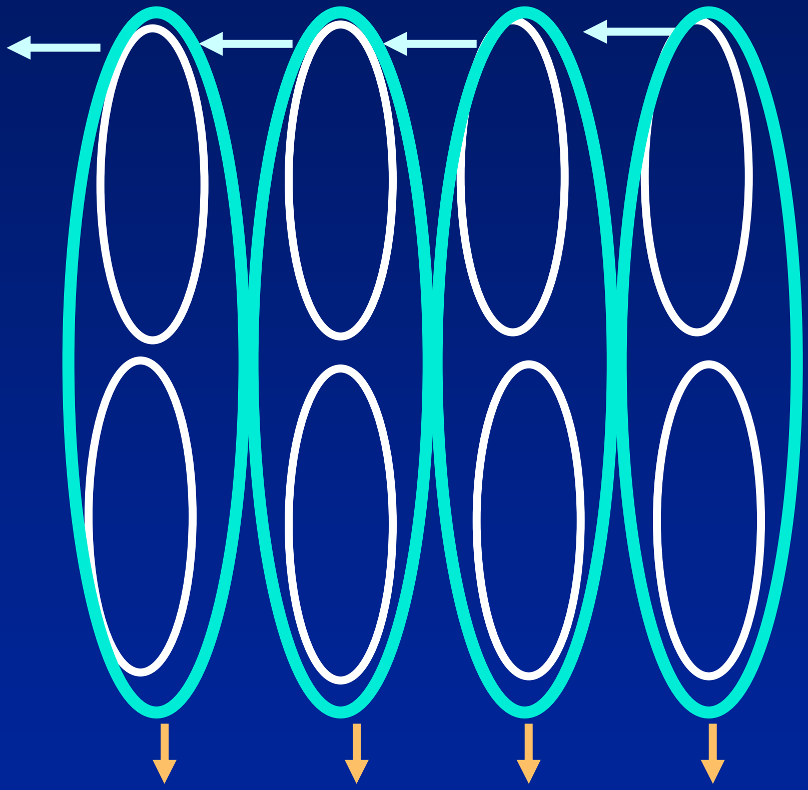
Multiplier(efficient)



Multiplier core



N-bit Multiplier (efficient)



- Area (LUTs):

$$v = \log_2 N$$

$$area =$$

$$\sum_{k=1}^{v-1} N \times \left(\frac{N}{2^k} + 1 \right)$$

- Speed : 2Nc + vs

Constant-Coeff Multiplier

$$Y = k \times X$$

- K : 8-bit constant coefficient
- X : 8-bit variable input
- Y : 16-bit output

Constant-Coeff Multiplier

$$\boxed{Y = k \times X}$$

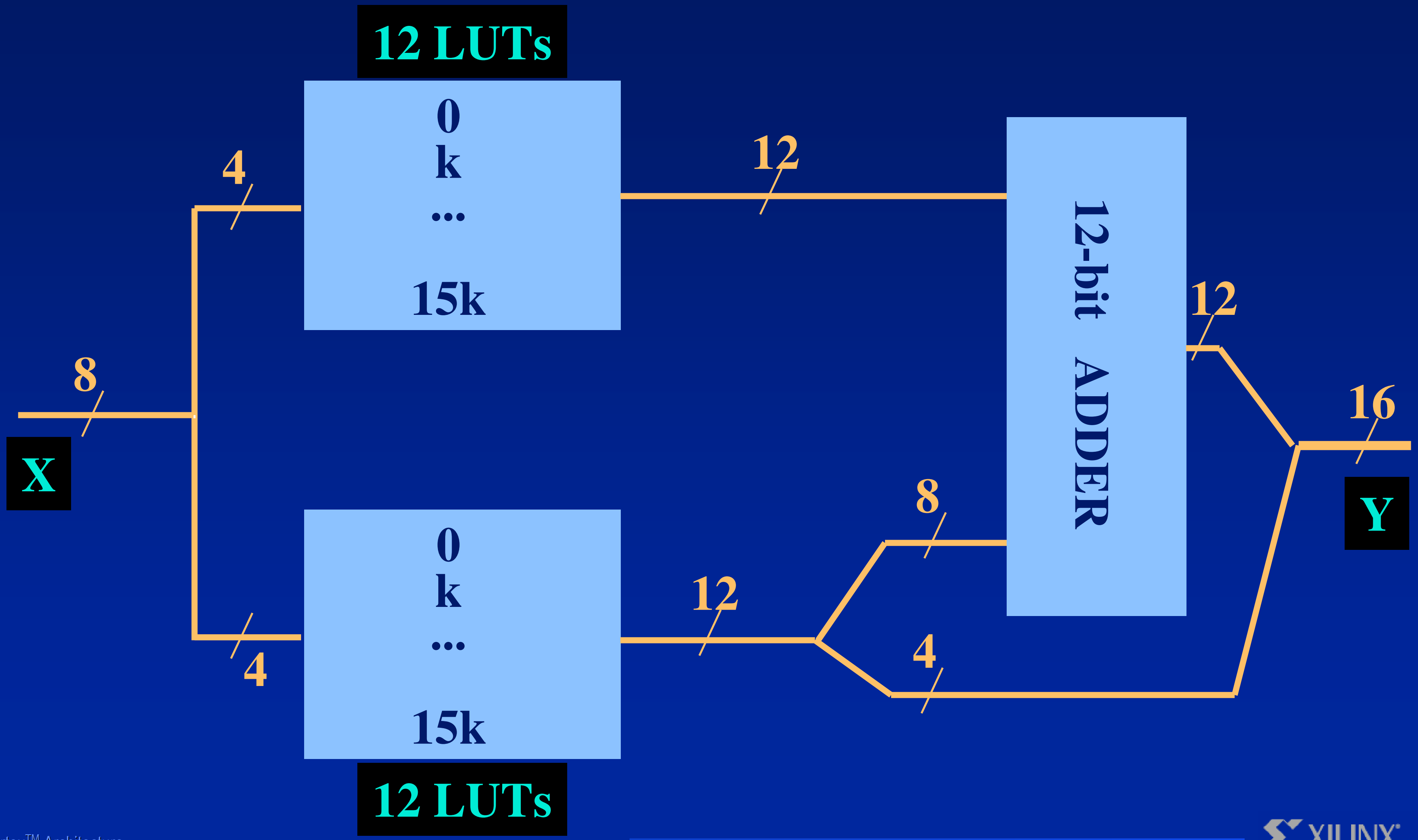
- K : 8-bit constant coefficient
- X : 8-bit variable input
- Y : 16-bit output

$$Y = 2^0 k \times X_0 + 2^1 k \times X_1 + \ldots + 2^7 k \times X_7$$

$$= \left[2^0 k \times X_0 + \ldots + 2^3 k \times X_3\right] +$$

$$\left[2^0 k \times X_4 + \ldots + 2^3 k \times X_7\right] \times 2^4$$

Constant-Coeff Multiplier



Const-coeff mult: 8-bits

- Area:
  - 2 LUTs 12 bit wide => 24 LUTs
  - 1 adder 12 bit wide => 12 LUTs
  - Total area = 36 LUTs
- Speed:
  - 2s (2 levels of LUT)

Constant-coeff: Booth-style

- Each run of 1 represents one ADD and one SUB
- Isolated 1 represents one ADD

$$11011011 \times 01000111$$
$$= 11011011000000 +$$
$$110110110000 -$$
$$11011011$$

Constant-coeff: Booth-style

Coeff = 10001011

- Area
  - A = add shift-1 and shift-2 => 11 LUTs
  - B = add shift-4 and shift-7 => 13 LUTs
  - C = A + B => 16 LUTs
  - Total area = 11+13+16=40 LUTs (> 36 LUTs)
- Speed
  - 2s or s
- Better to use KCM

Constant-coeff: Booth multipler

Coeff = 00011101

- Area
  - A = add shift-0 and shift-5 => 14 LUTs
  - C = A - shift-1 => 15 LUTs
  - Total area = 14+15=29 LUTs (< 36 LUTs)
- Speed
  - 2s or s
- Better to use Booth-style

Distributed Arithmetic

- Linear time-invariant networks

$$y(n) = \sum_{k=1}^{N} A_k \cdot x_k(n)$$

$$y(n) = response\ of\ \ network\ at\ time\ n$$

$$x_k(n) = k^{th}\ input\ \ \mathrm{var}\ at\ time\ n$$

$$A_k = weight\ \ factor$$

Distributed Arithmetic

$$x_k = -x_{k0} + \sum_{b=1}^{B-1} x_{kb} \cdot 2^{-b}$$

$$y = \sum_{k=1}^{N} A_k \cdot \left[-x_{k0} + \sum_{b=1}^{B-1} x_{kb} \cdot 2^{-b} \right]$$

$$= -\sum_{k=1}^{N} x_{k0} \cdot A_k + \sum_{k=1}^{N} \sum_{b=1}^{B-1} x_{kb} \cdot A_k 2^{-b}$$

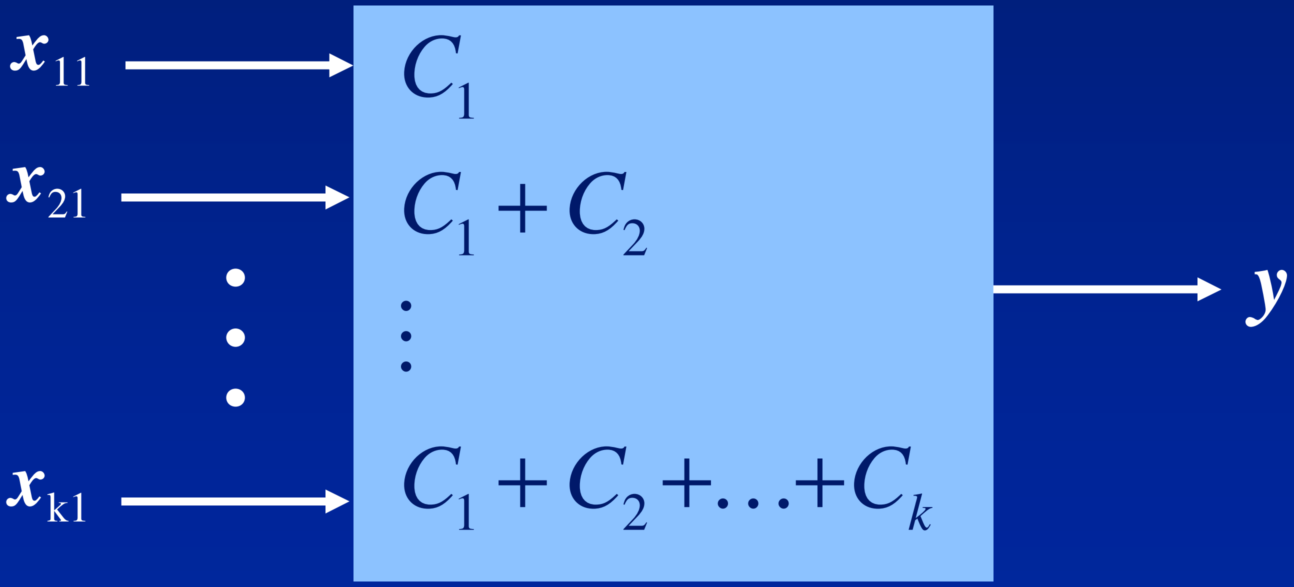
Distributed Arithmetic

- Expanded form

$$
\begin{aligned}
y = & -\left[x_{10} \cdot A_1 + x_{20} \cdot A_2 + \ldots + x_{k0} \cdot A_k\right] \\
& + \left[x_{11} \cdot A_1 + x_{21} \cdot A_2 + \ldots + x_{k1} \cdot A_k\right] 2^{-1} \\
& \vdots \\
& + \left[x_{1(B-1)} \cdot A_1 + x_{2(B-1)} \cdot A_2 + \ldots + x_{k(B-1)} \cdot A_k\right] 2^{-(B-1)}
\end{aligned}
$$

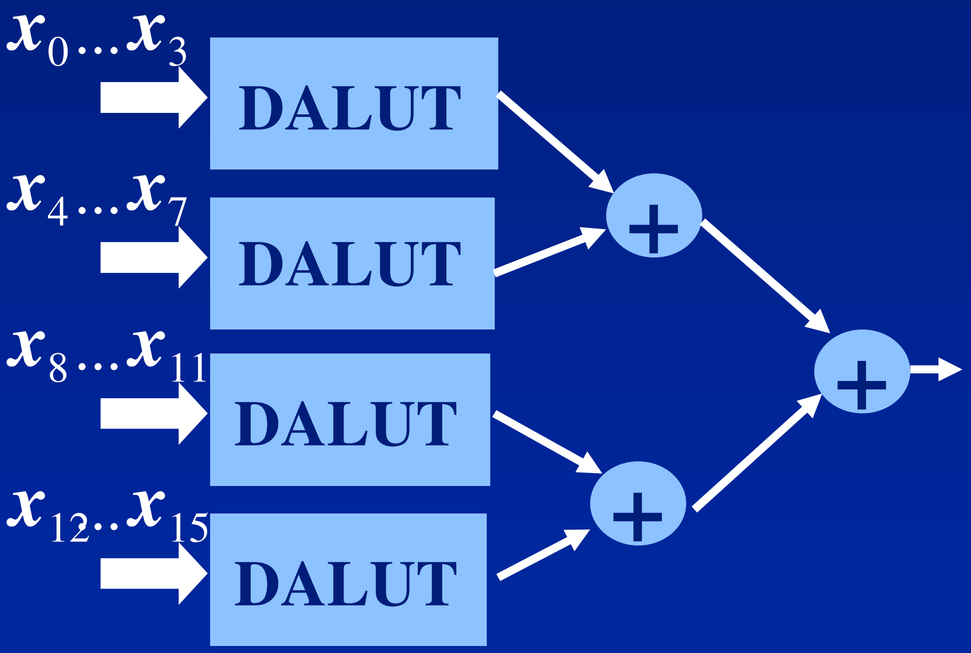
DALUT

$$\left[x_{11} \cdot C_1 + x_{21} \cdot C_2 + \ldots + x_{k1} \cdot C_k\right]$$



Distributed Arithmetic

- K=16 => 2^{16} DALUT contents! => 512xW CLBs
- Break DALUTs to 4/5 inputs and add

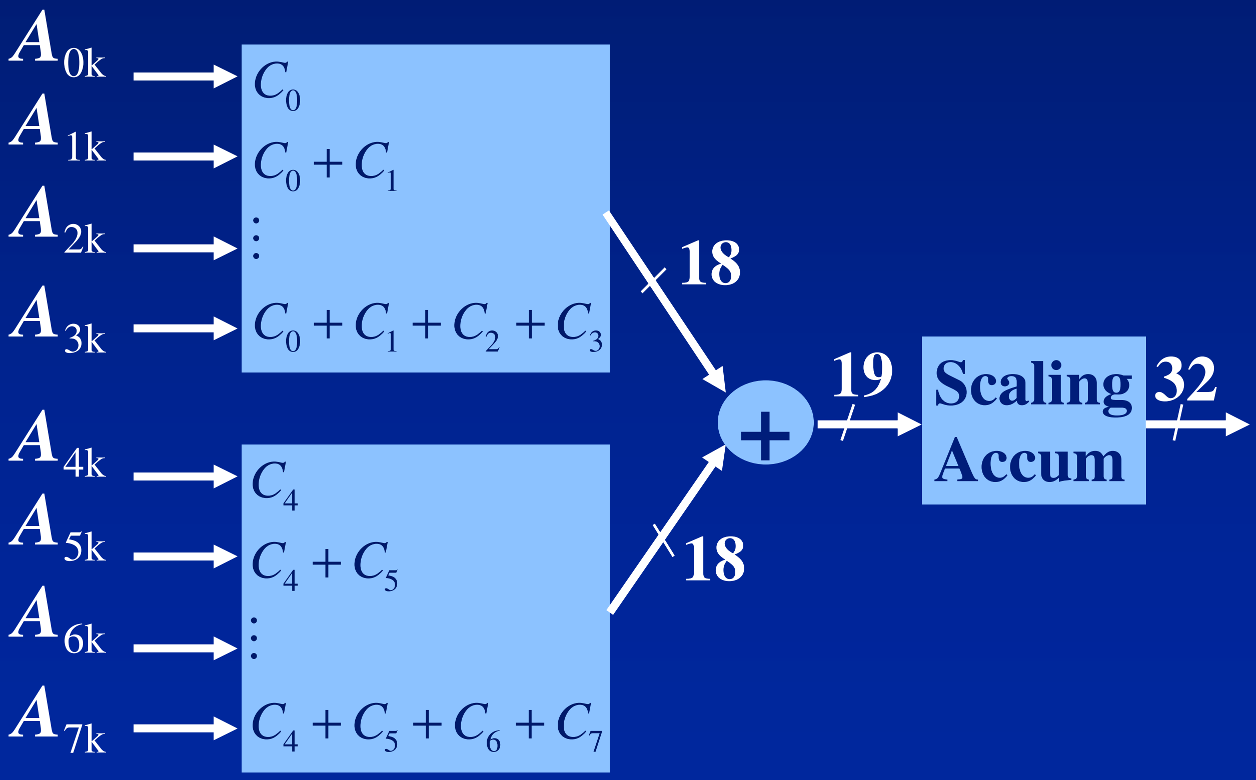


- Needs W CLBs
- Delay = 3s

Area efficient

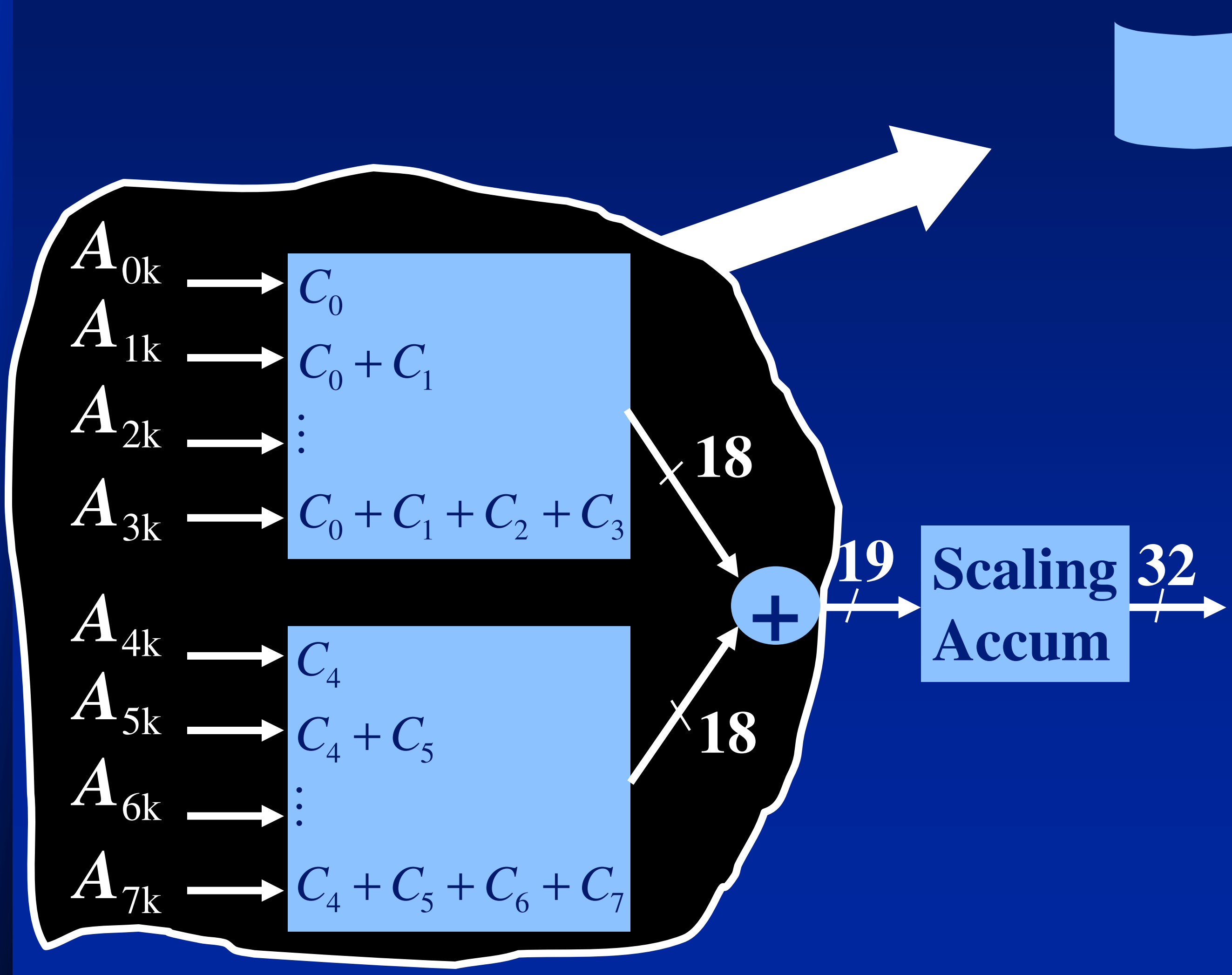
$$Y = C_0 \cdot A_0 + C_1 \cdot A_1 + \ldots + C_7 \cdot A_7$$

- C_k: 16-bit
- A_k: 16-bit

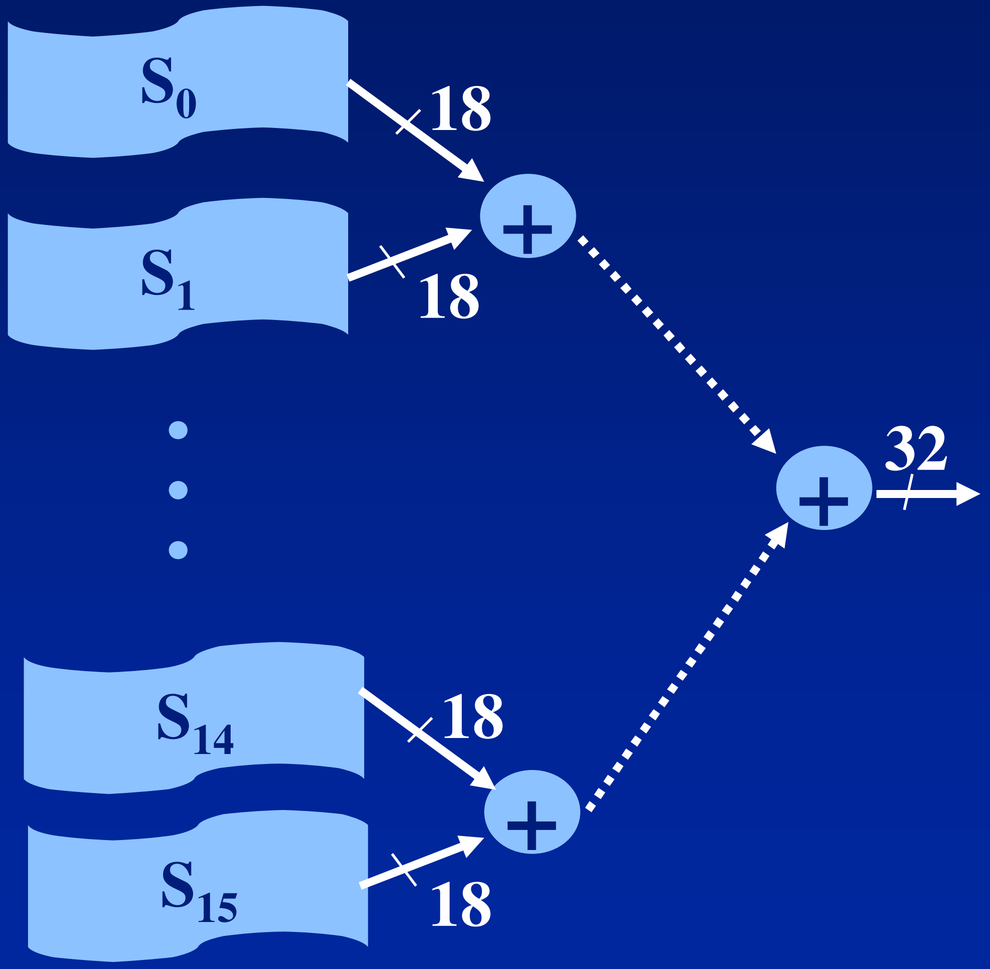


- Area: [2*18]+19+32 = 87 LUTs
- Speed: [3s*16] = 48s

Speed efficient



Speed efficient



- Area:
  [2*18*16]+
  [8*20]+ [4*23]+
  [2*28]+ [32]
  = 916 LUTs (9X)
- Speed:
  s+4s = 5s (9X)
- Intermediate configurations possible

Conclusions

- Described the latest generation of Xilinx FPGA: VIRTEX™
- Architecture is very well-suited for DSP core functions:
  - Adders / Subtractors
  - Wide AND / Ors
  - Comparators / Decoders
  - Multiplexers
  - Multipliers
  - Distributed Arithmetic