Advanced Digital Logic Design – EECS 303

http://ziyang.eecs.northwestern.edu/eecs303/

Teacher:	Robert Dick
Office:	L477 Tech
Email:	dickrp@northwestern.edu
Phone:	847–467–2298

NORTHWESTERN
UNIVERSITY

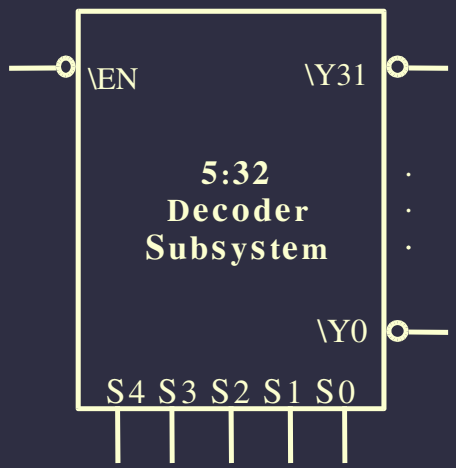
Outline

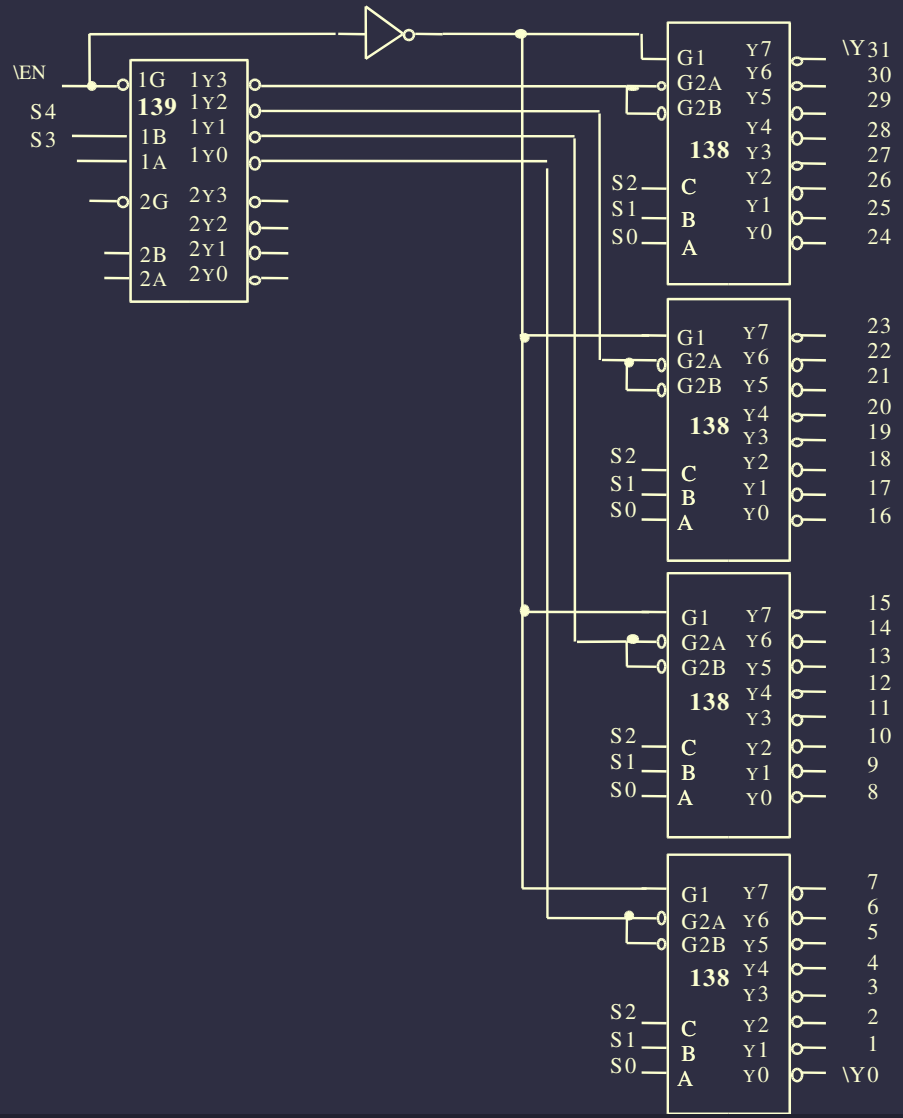
1. Implementation technologies
2. Homework

Section outline

1. Implementation technologies
    - Review of MUX composition
    - Steering logic
    - ROMs
    - FPGAs
    - Transformations for CMOS

5:32 decoder/demultiplexer





5:32 decoder/demultiplexer implementation details

- Why is G1 connected to an inverted active-low enable signal?
- Why are 2A, 2B, and 2G not connected on the 74139 part?
- What would happen if this design were used and the parts were TTL (I don't expect you to know this one already)?
- How about CMOS?

Section outline

1. Implementation technologies
   - Review of MUX composition
   - Steering logic
   - ROMs
   - FPGAs
   - Transformations for CMOS

Tally circuit example

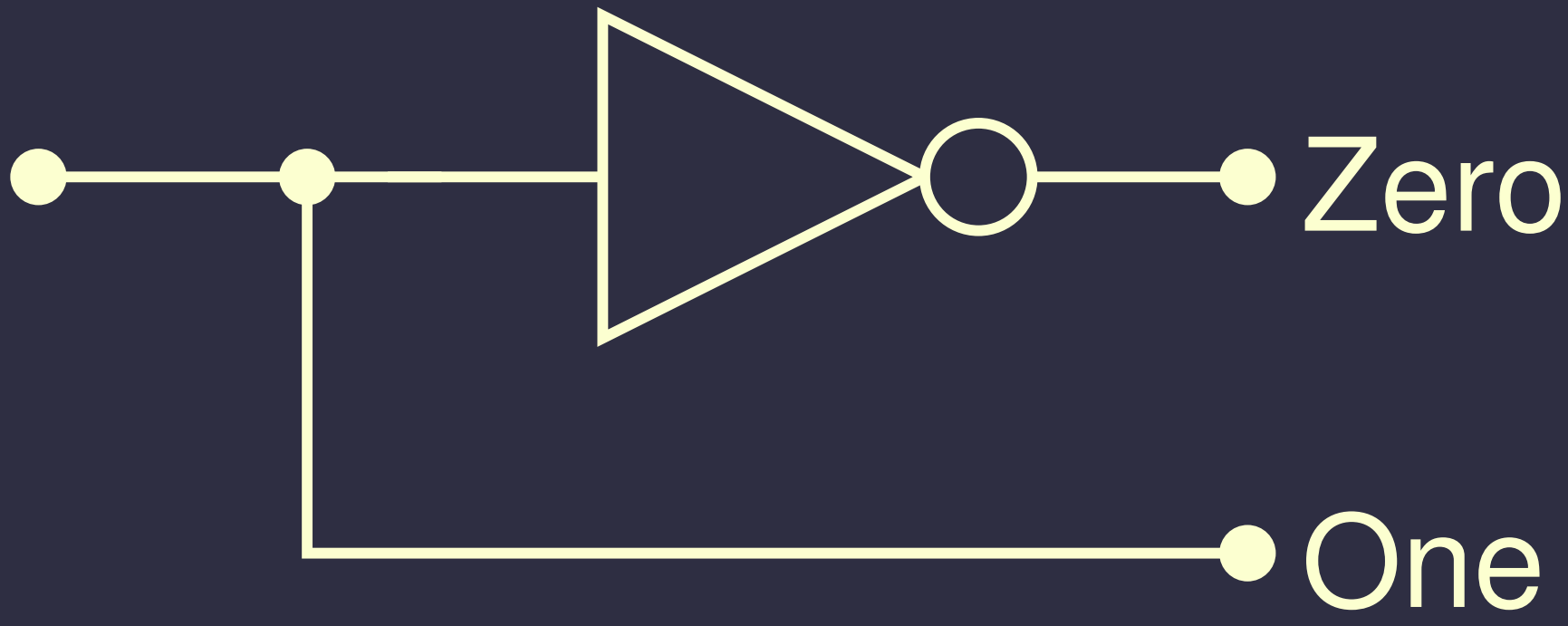
- Given n-input circuit
- Count number of 1s in input

I_1	Zero	One
0	1	0
1	0	1

Tally circuit example

I_1	Zero	One
0	1	0
1	0	1

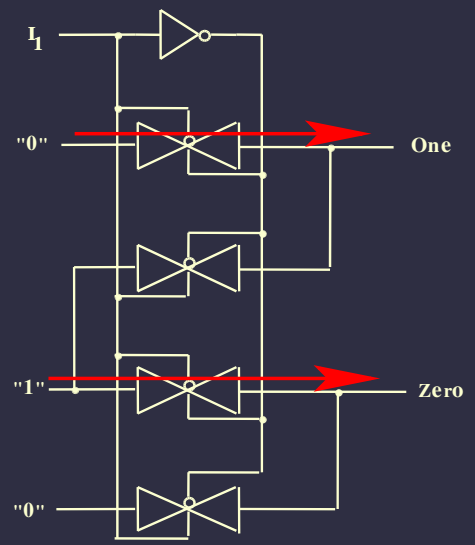
Can implement using logic gates



Tally circuit example

I_1	Zero	One
0	1	0
1	0	1

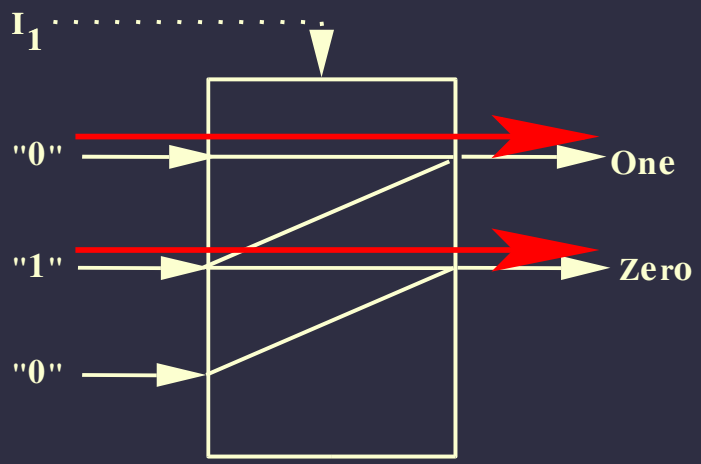
Can implement using TGs



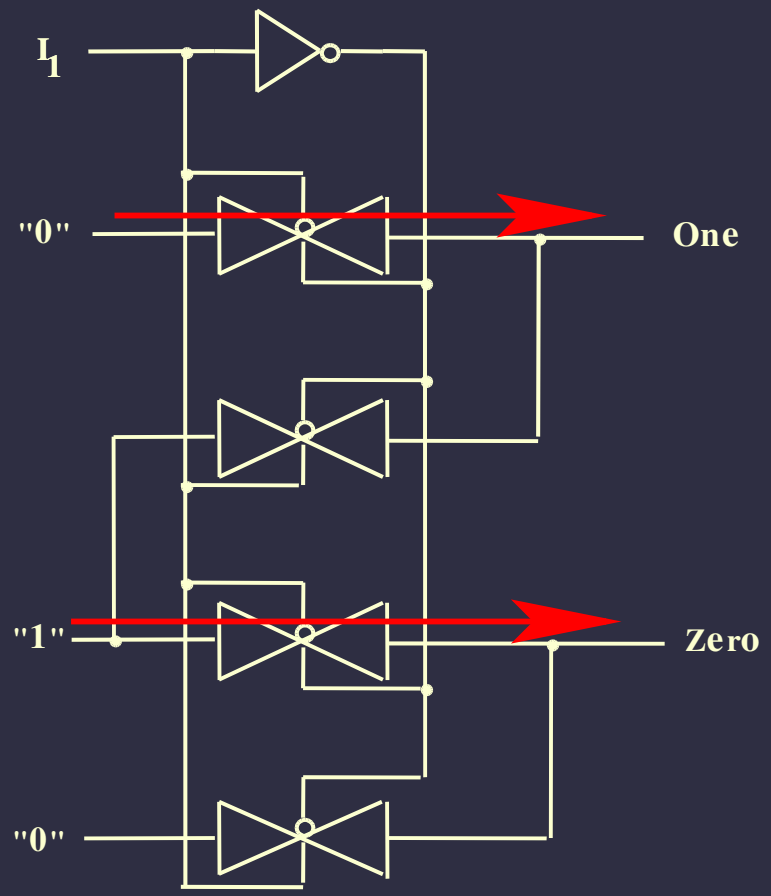
Tally circuit example

I_1	Zero	One
0	1	0
1	0	1

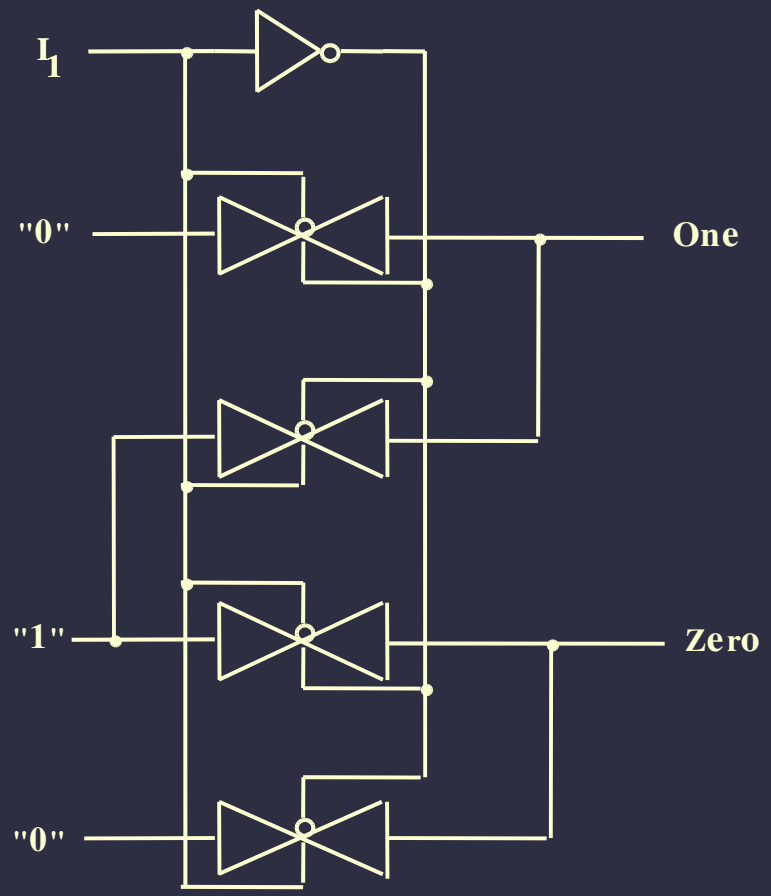
Can implement using TGs



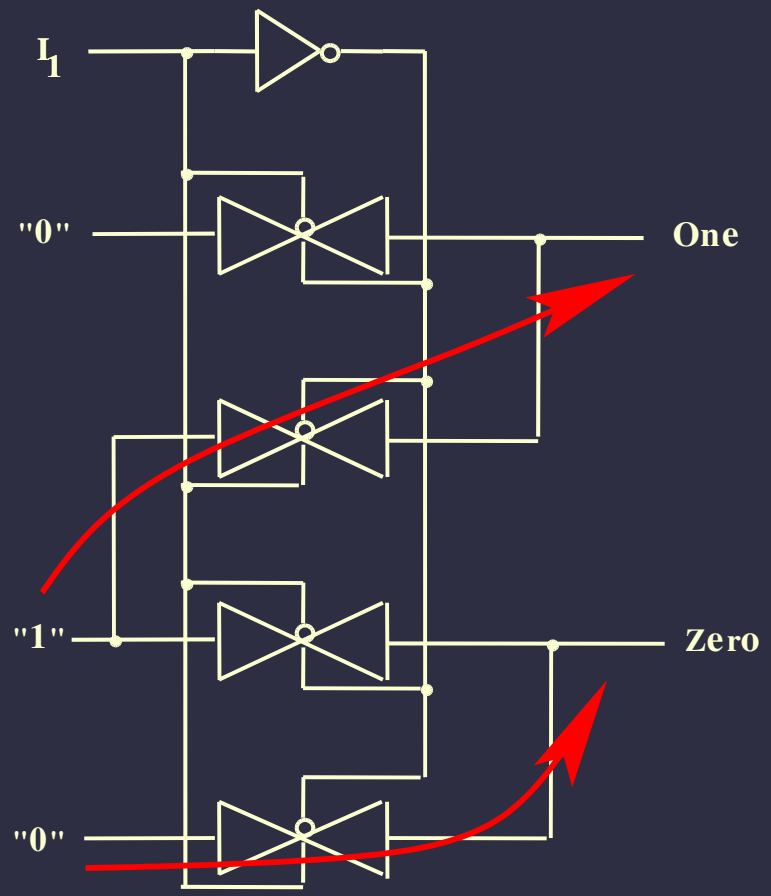
TG tally circuit



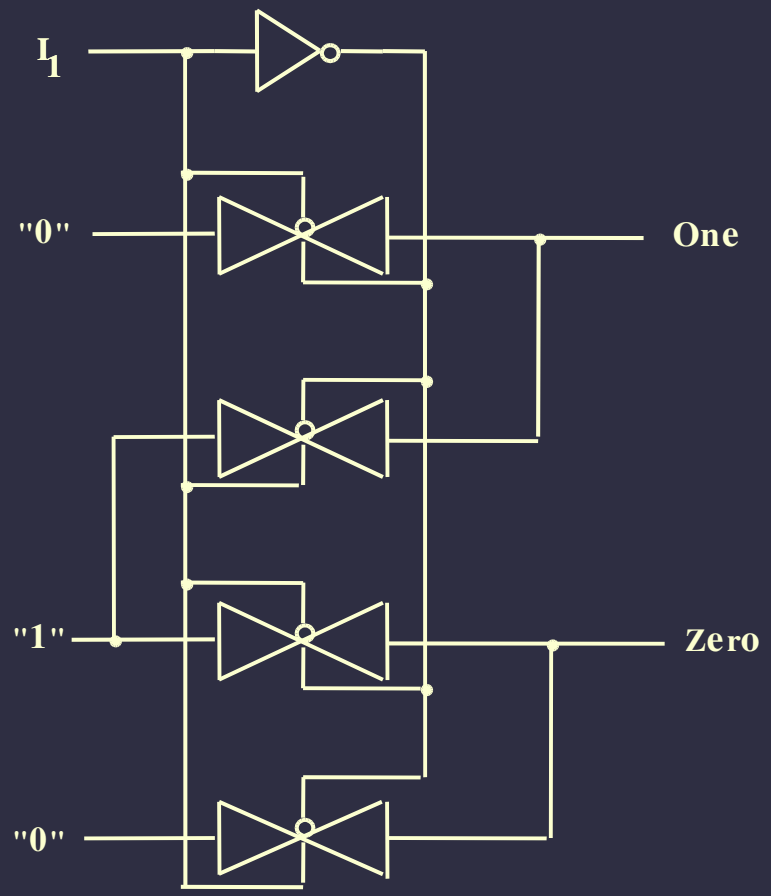
TG tally circuit



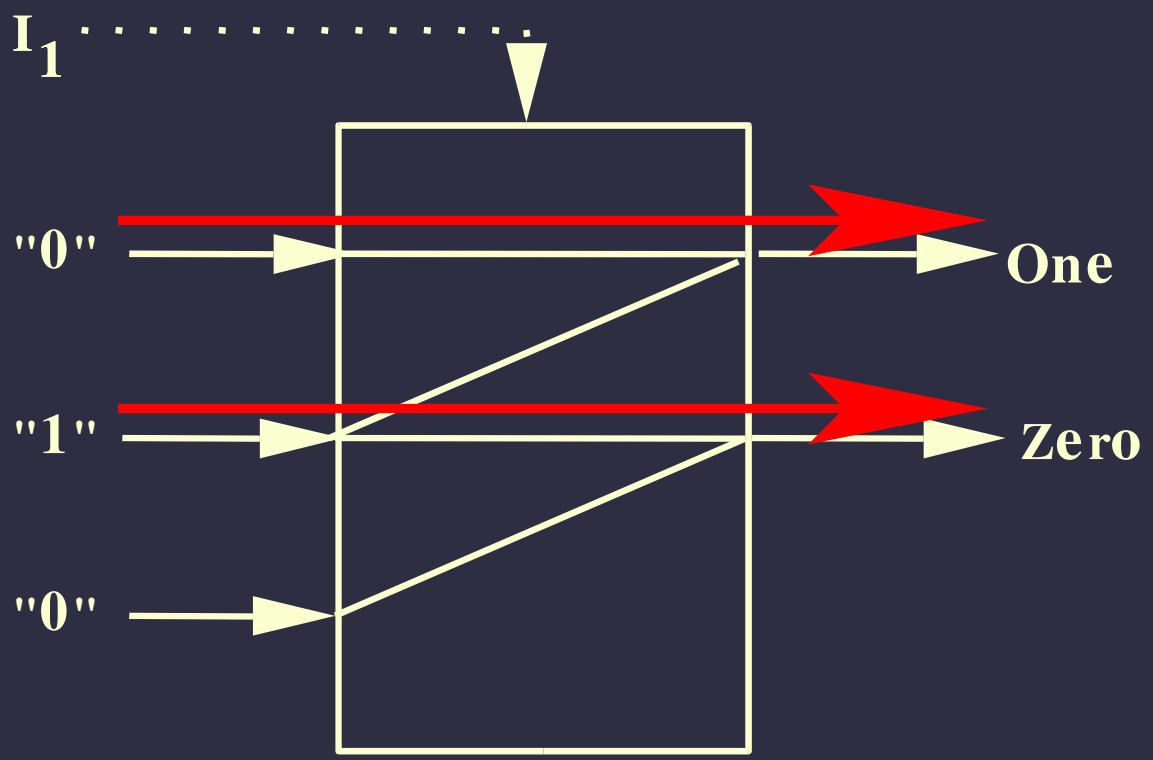
TG tally circuit



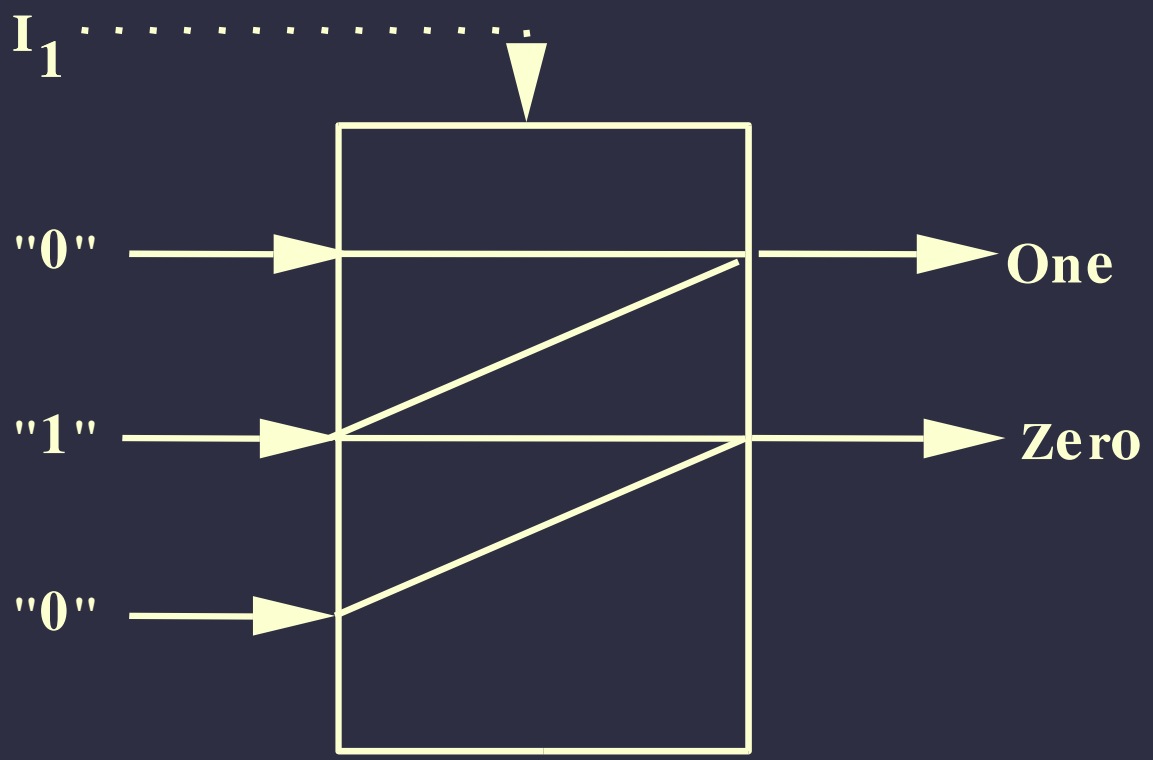
TG tally circuit



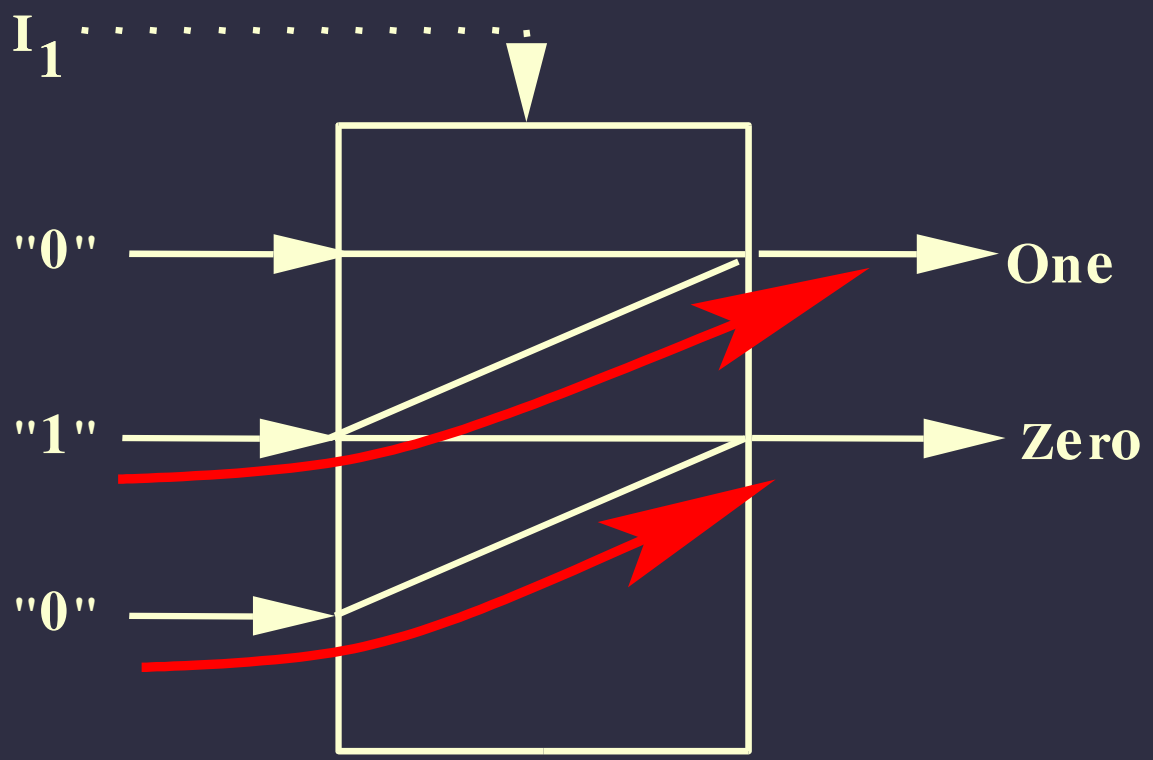
TG tally circuit



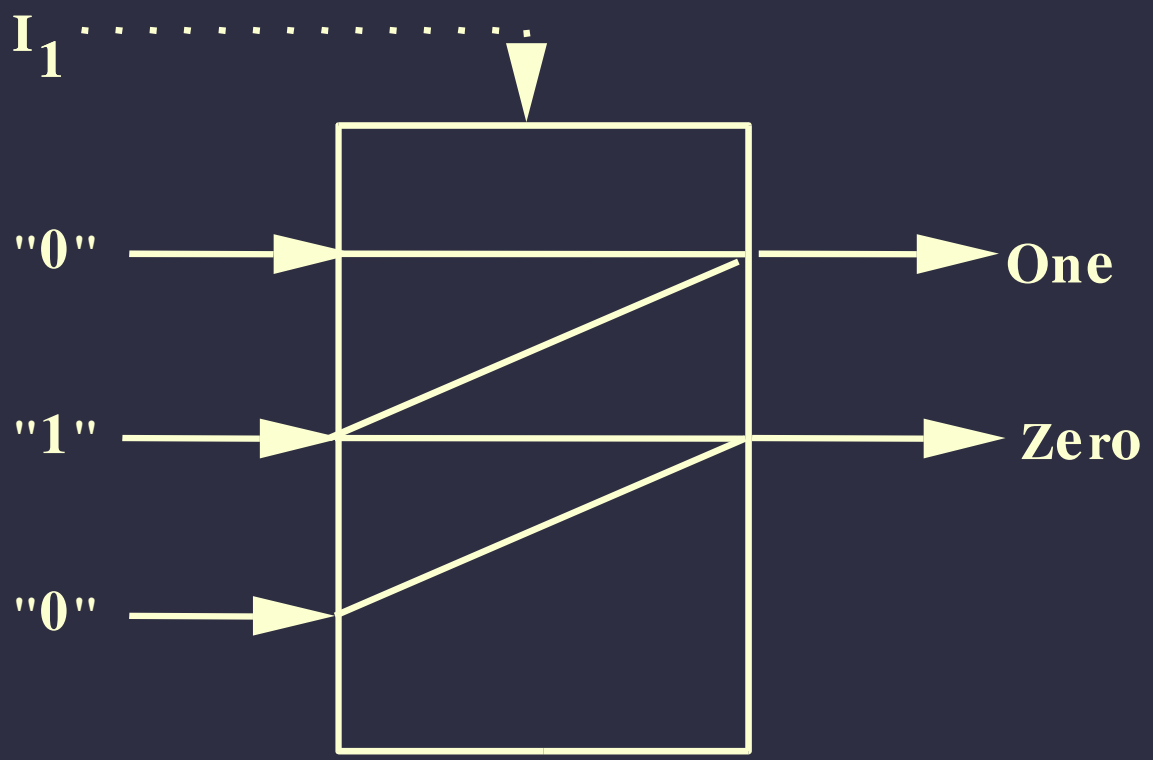
TG tally circuit



TG tally circuit

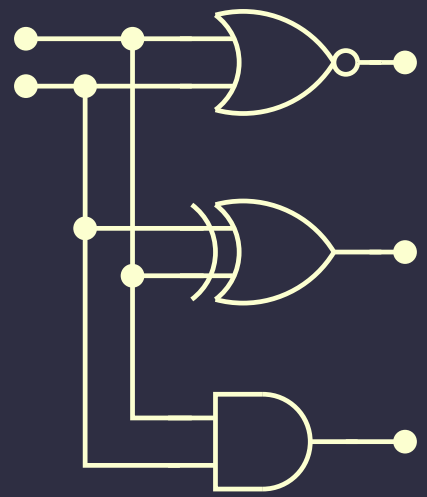


TG tally circuit



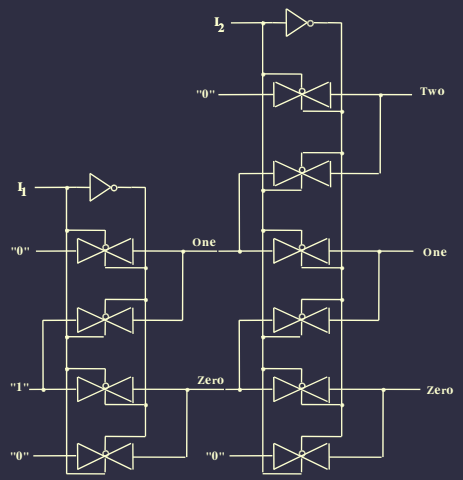
4-input tally

I_1	I_2	Zero	One	Two
0	0	1	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1



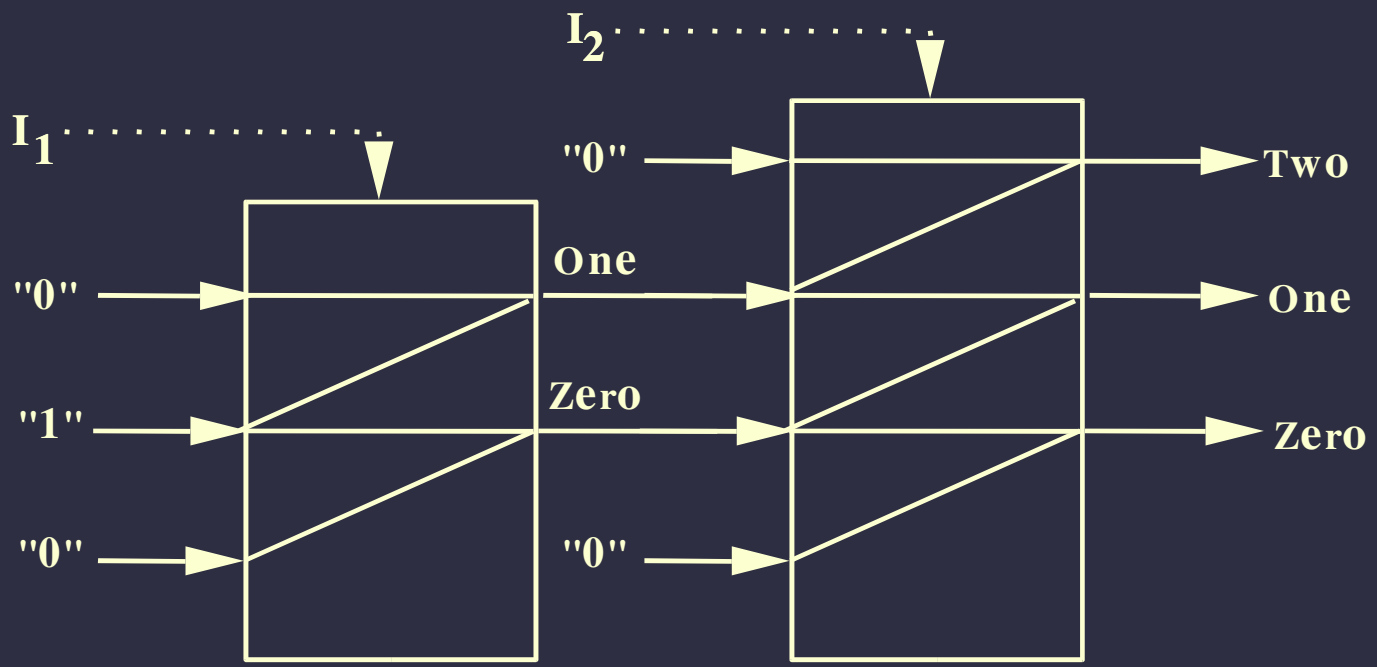
4-input tally

I_1	I_2	Zero	One	Two
0	0	1	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1

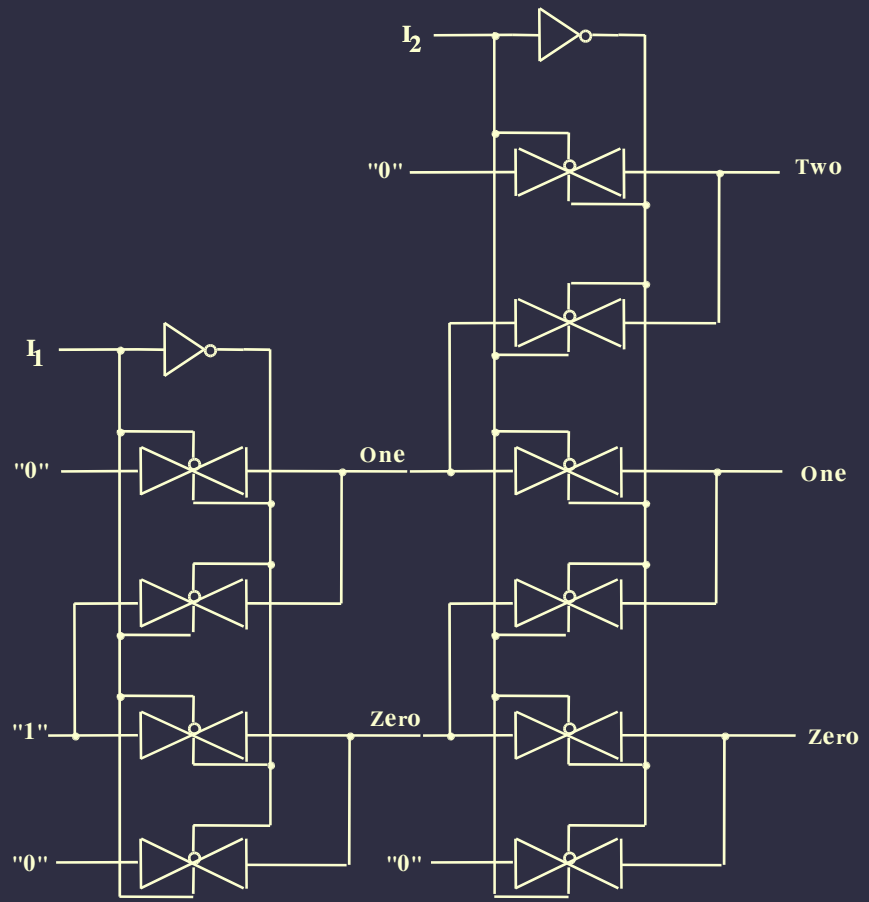


4-input tally

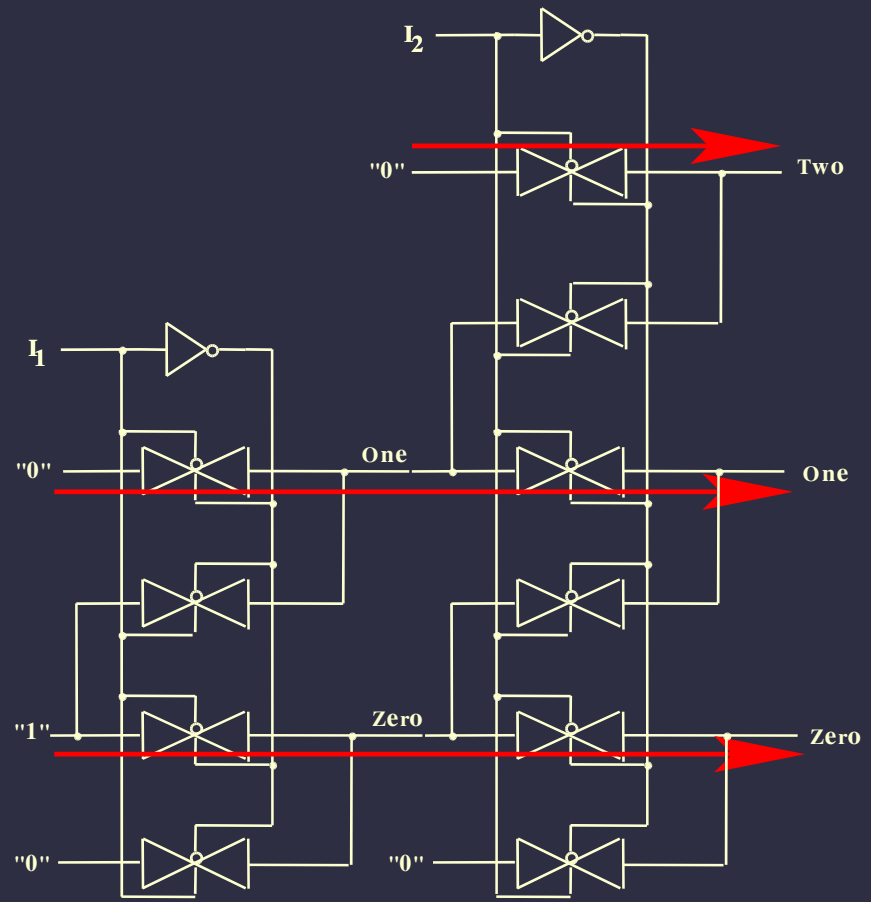
I_1	I_2	Zero	One	Two
0	0	1	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1



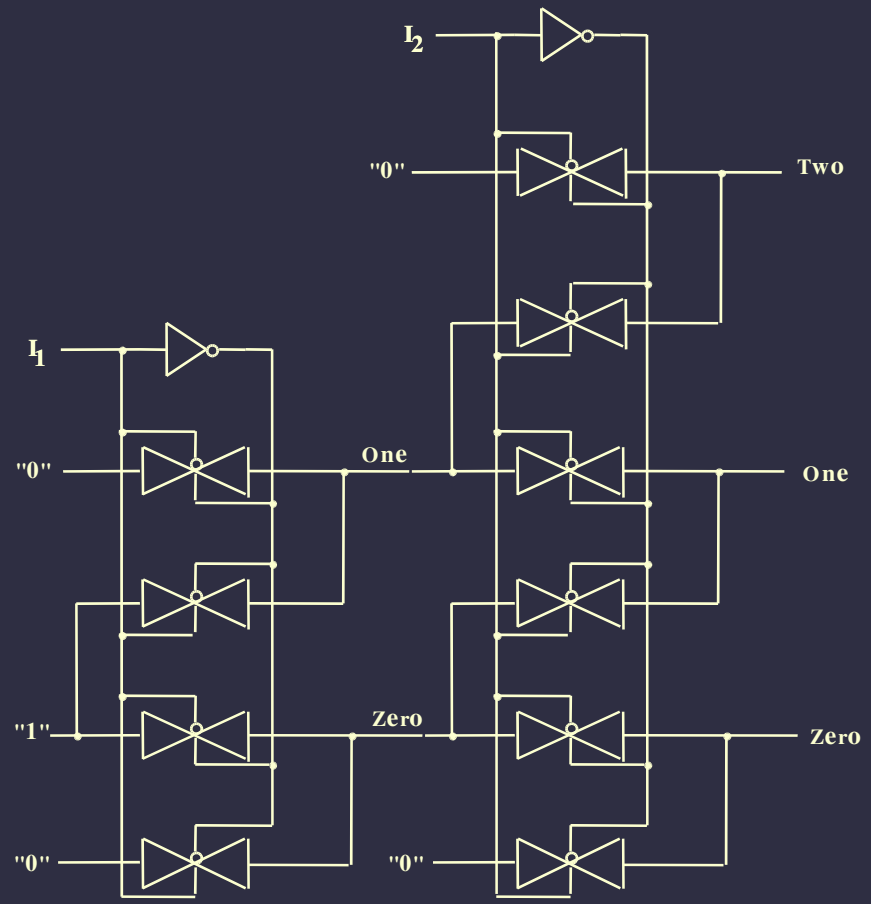
TG tally circuit



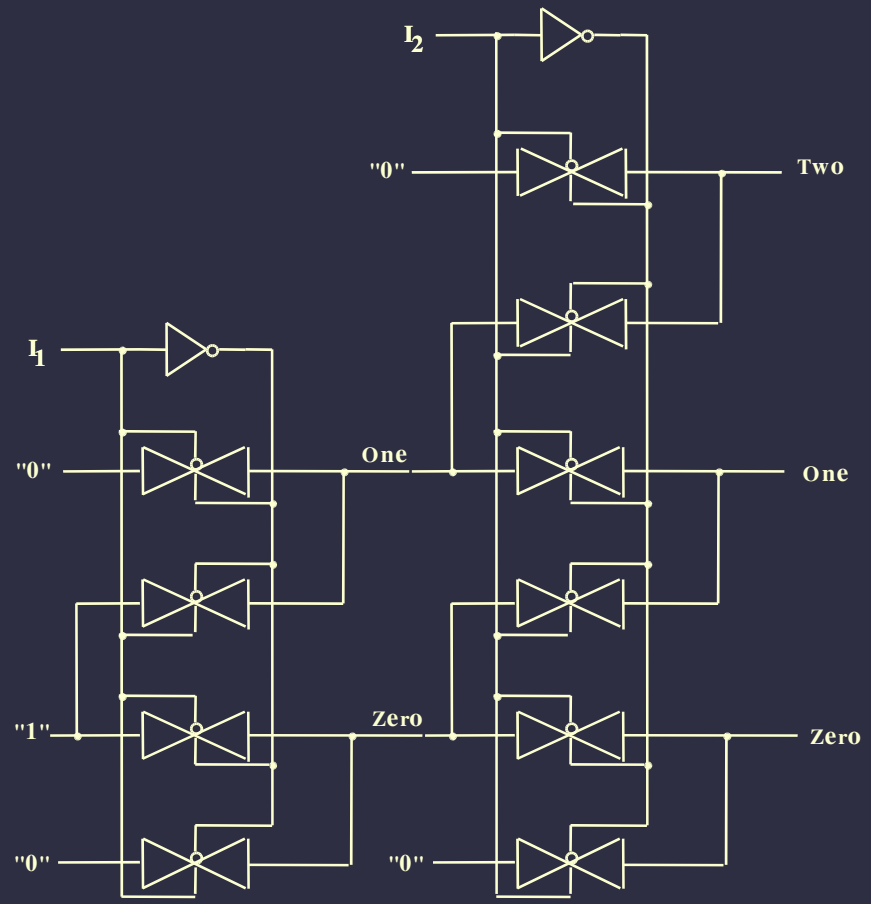
TG tally circuit



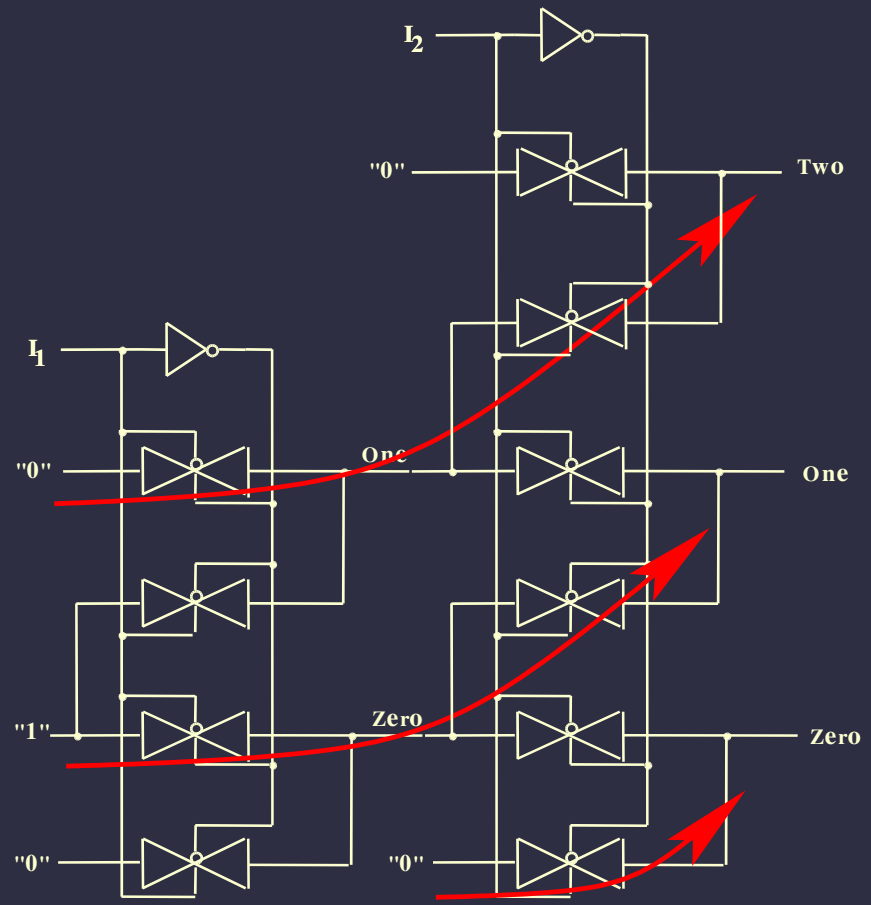
TG tally circuit



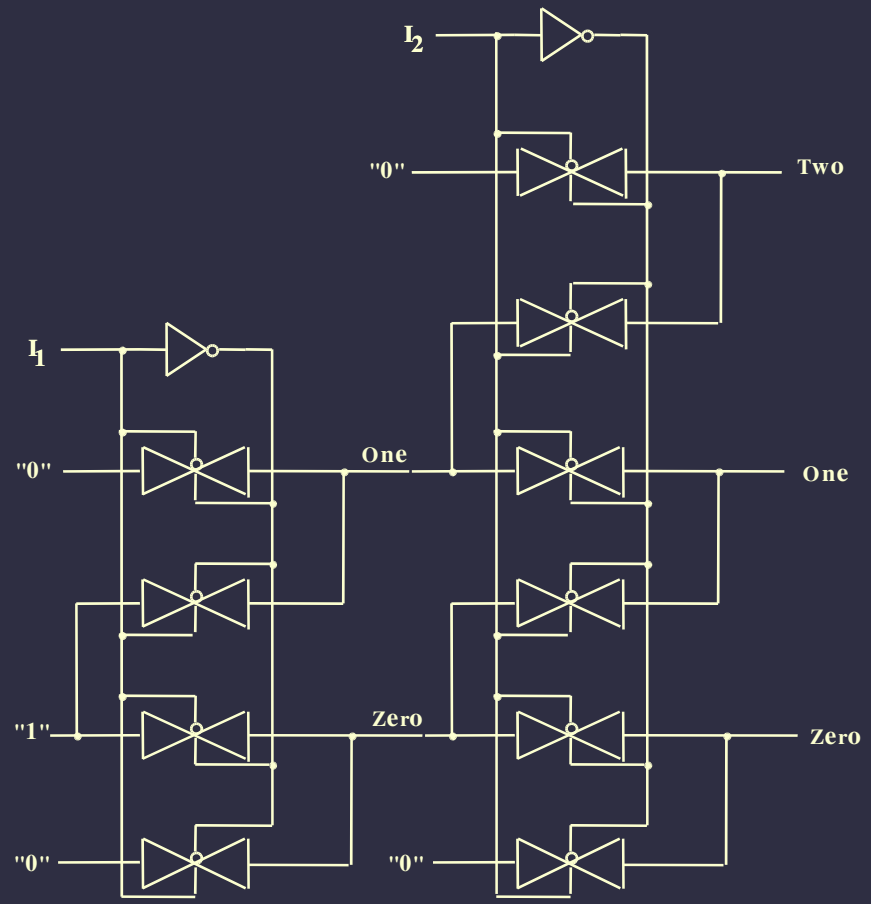
TG tally circuit



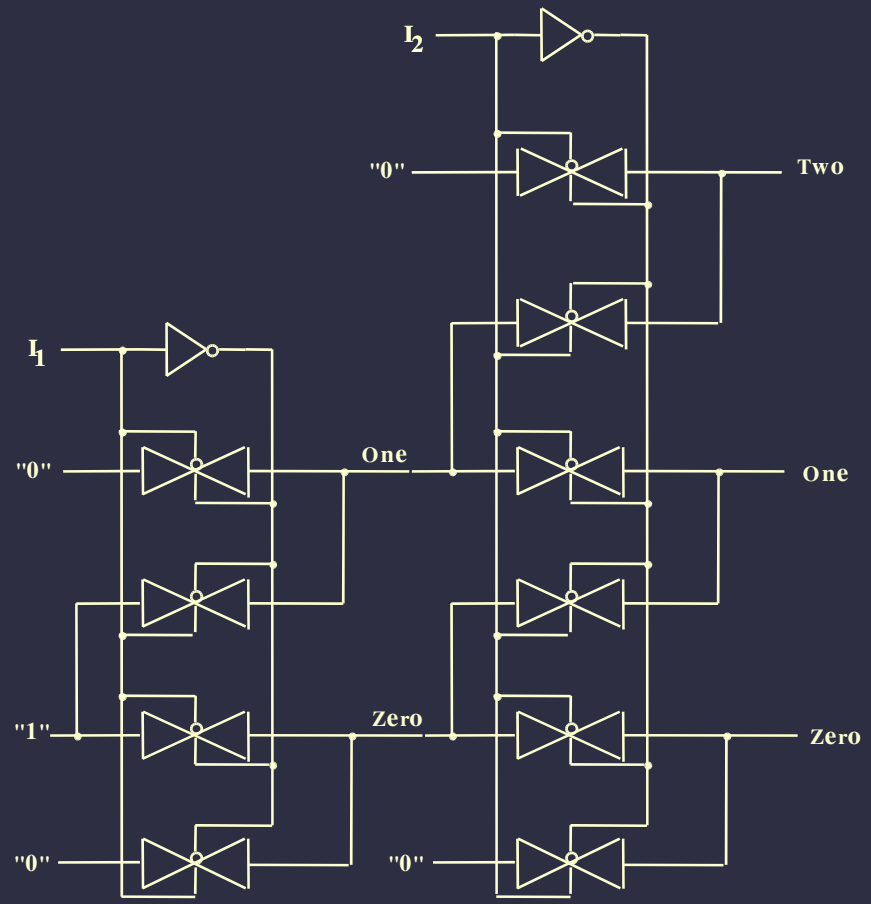
TG tally circuit



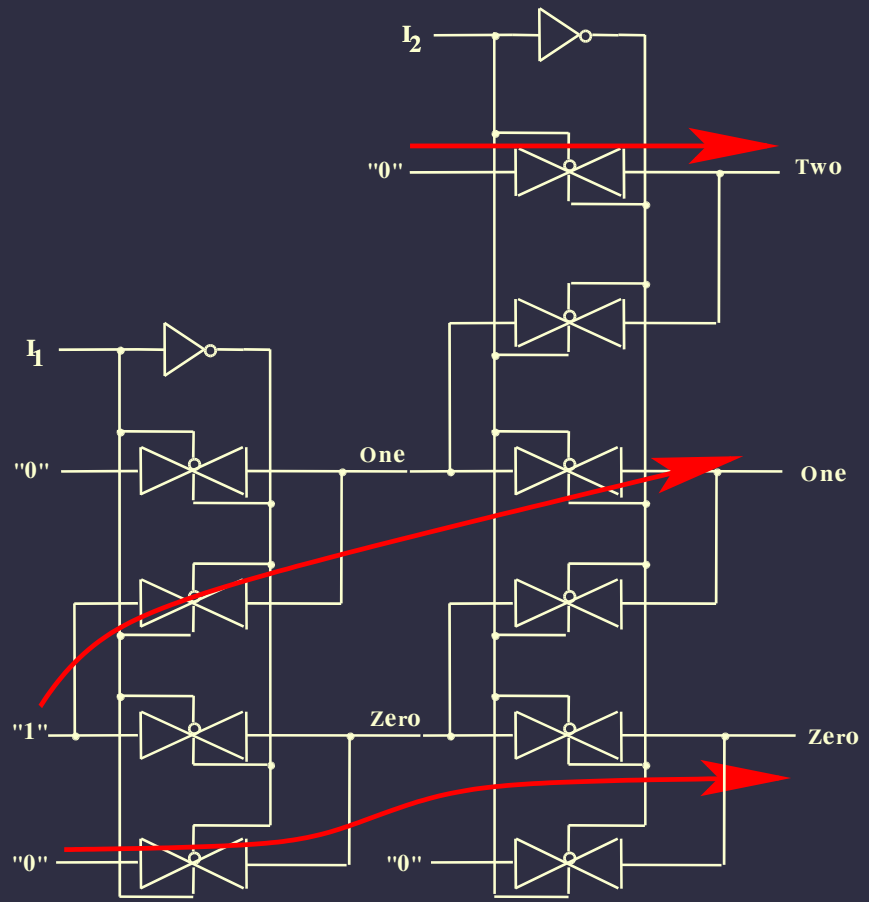
TG tally circuit



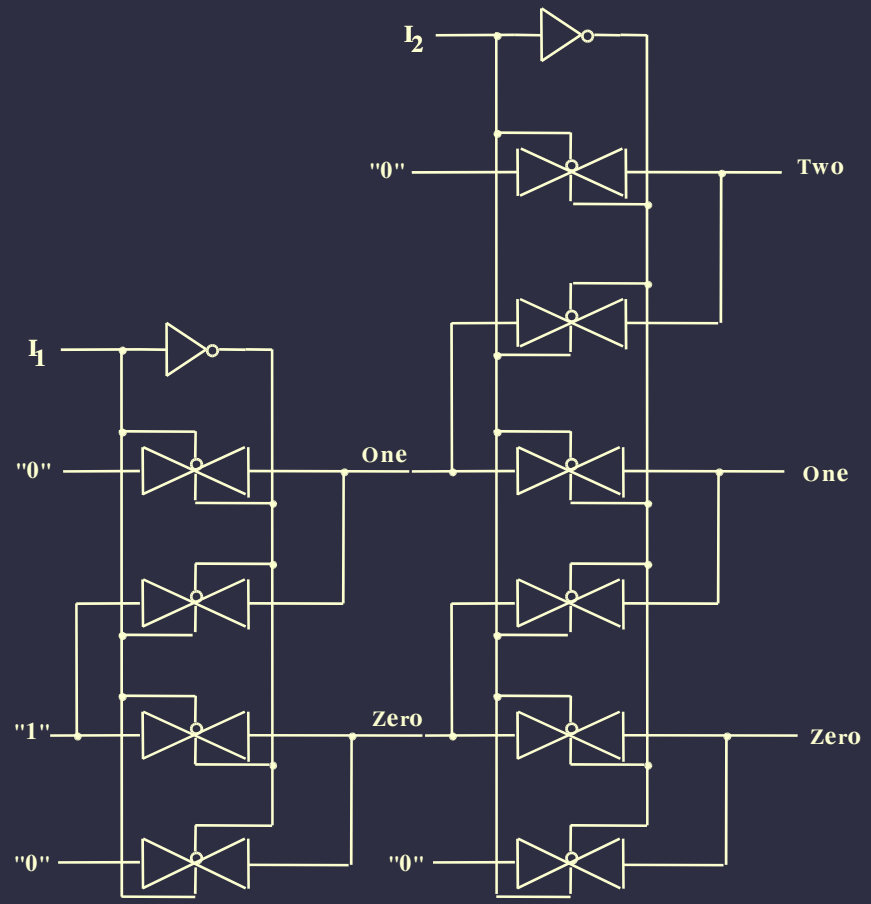
TG tally circuit



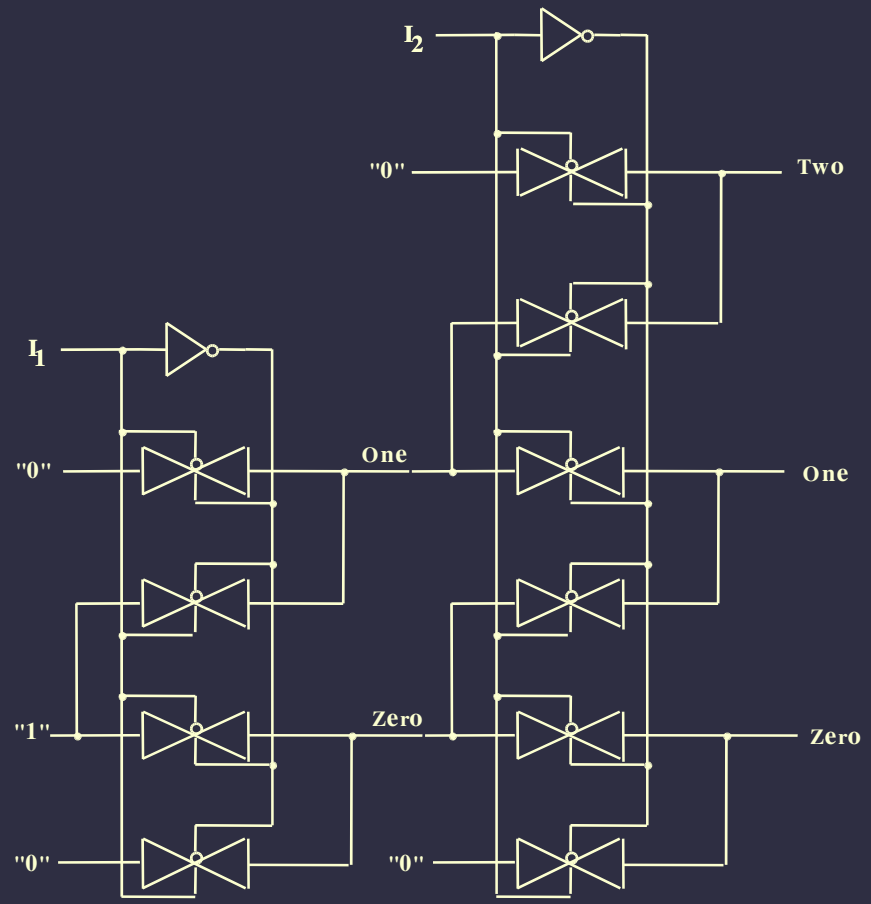
TG tally circuit



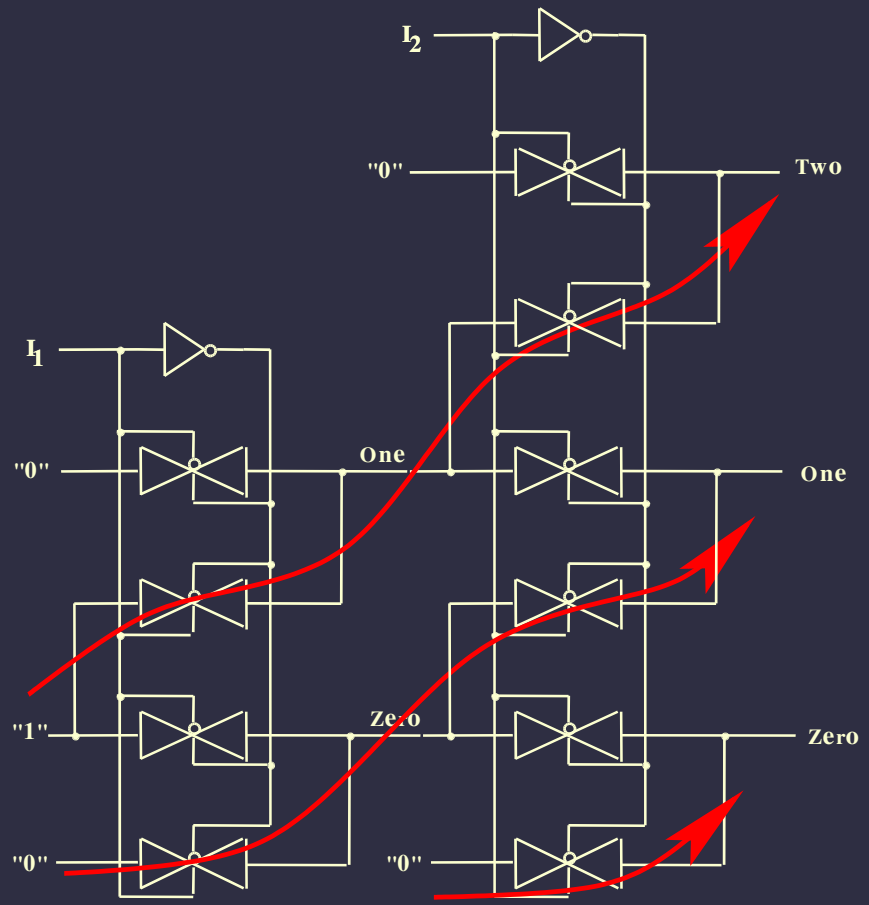
TG tally circuit



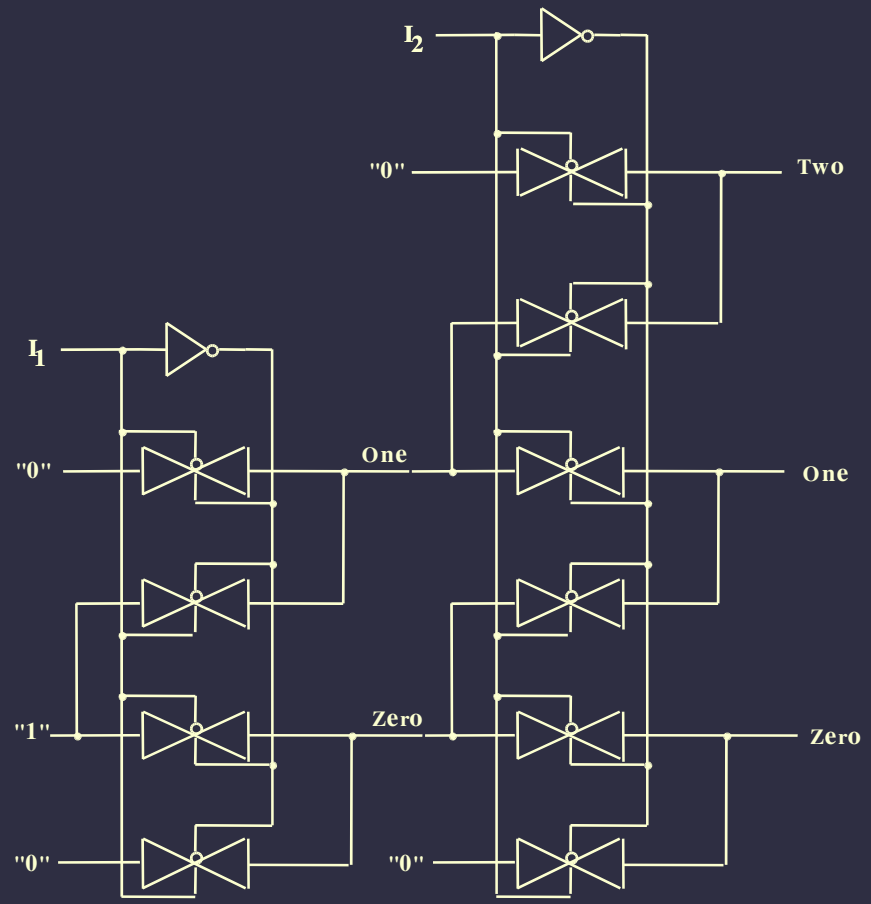
TG tally circuit



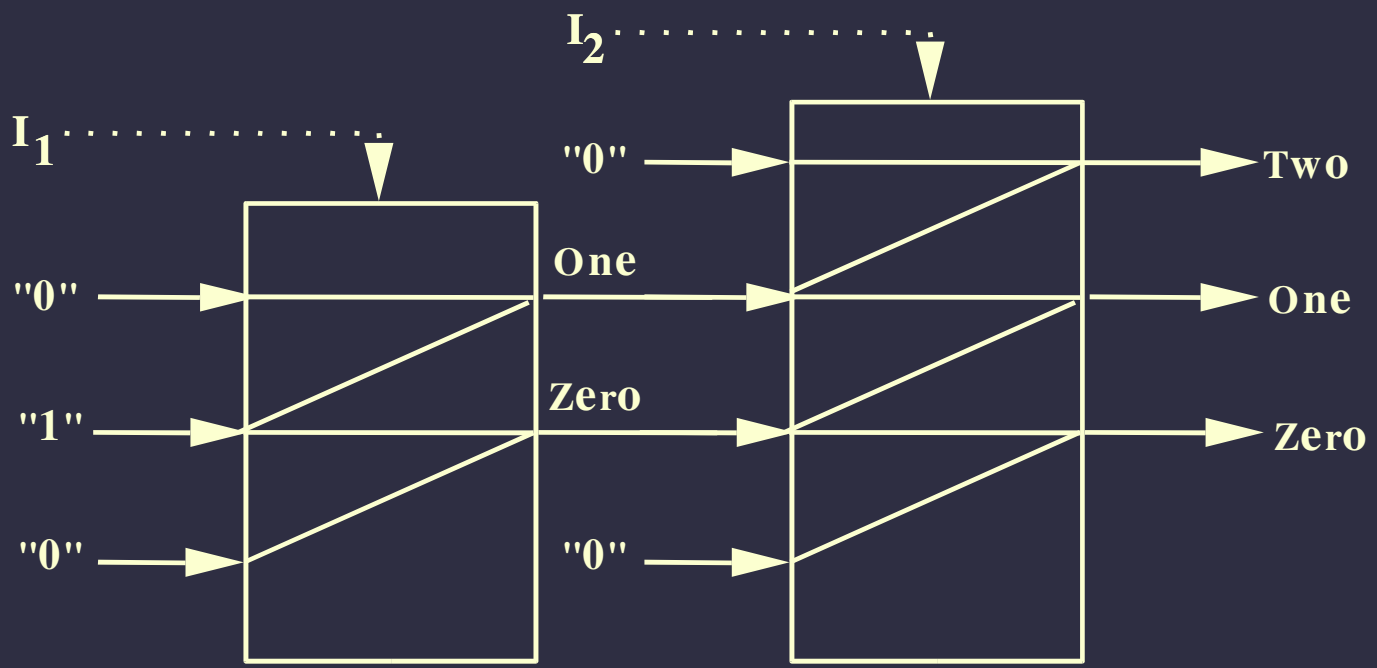
TG tally circuit



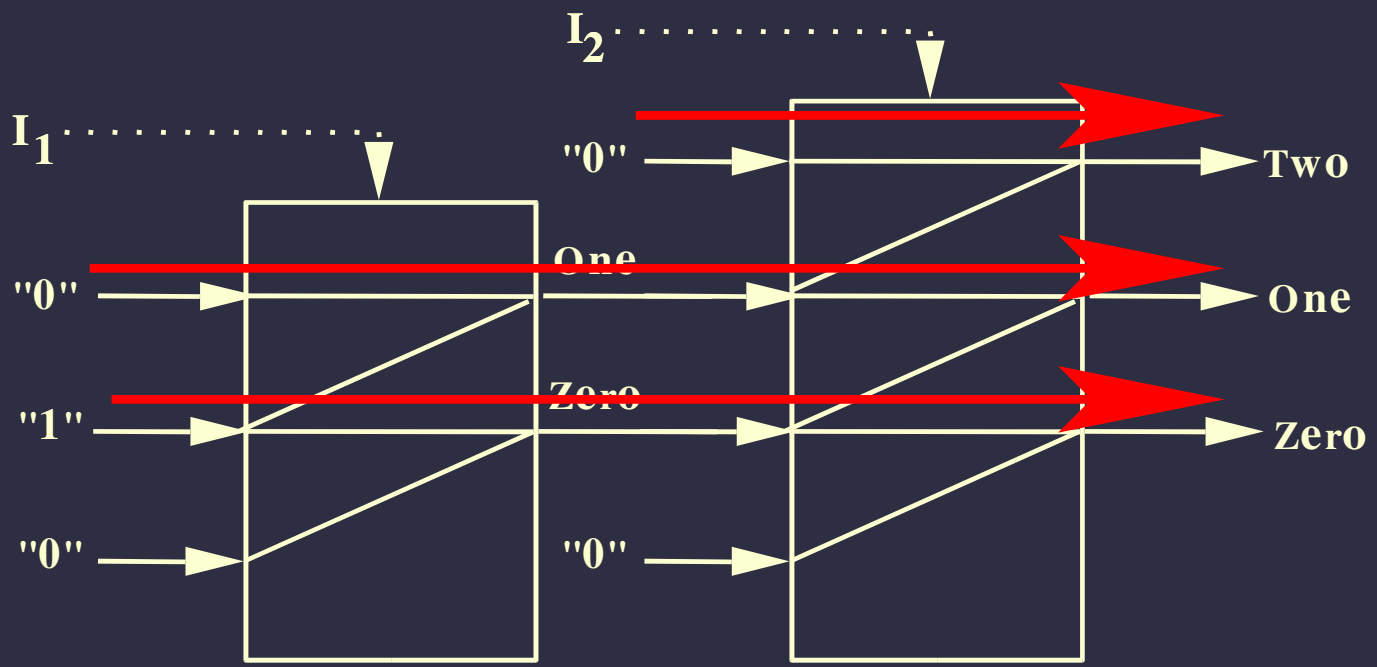
TG tally circuit



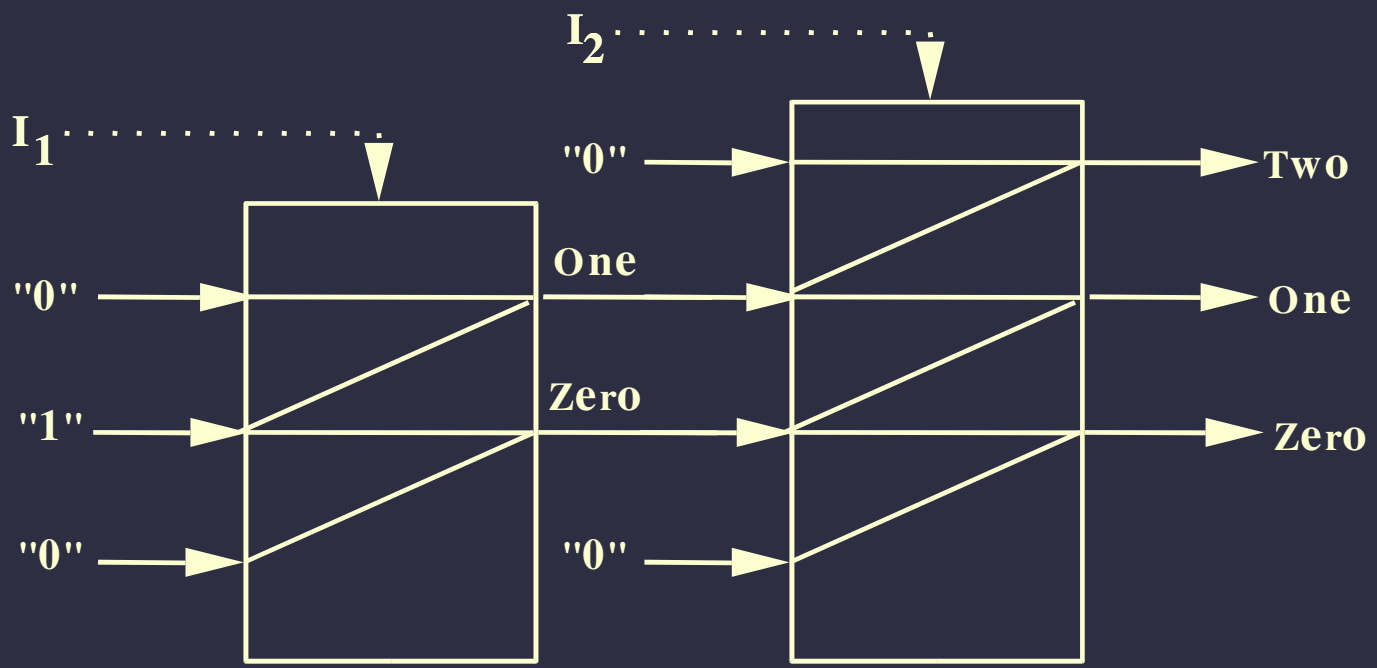
TG tally circuit



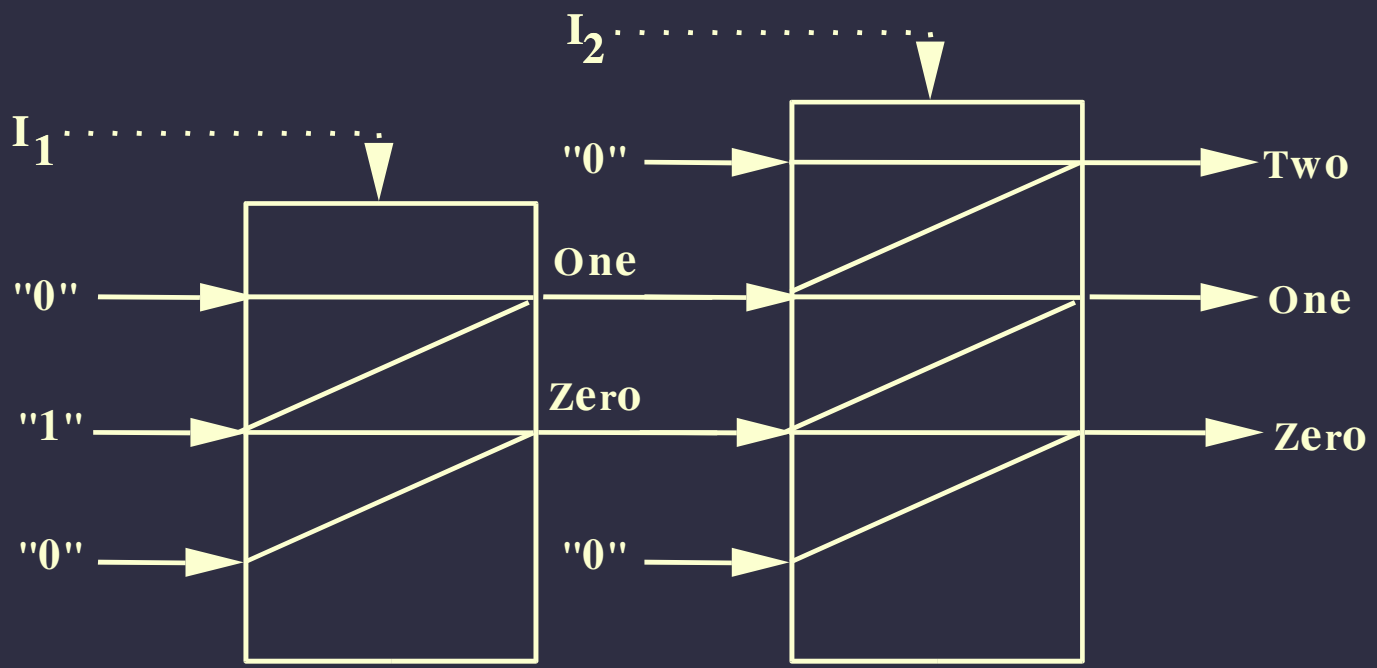
TG tally circuit



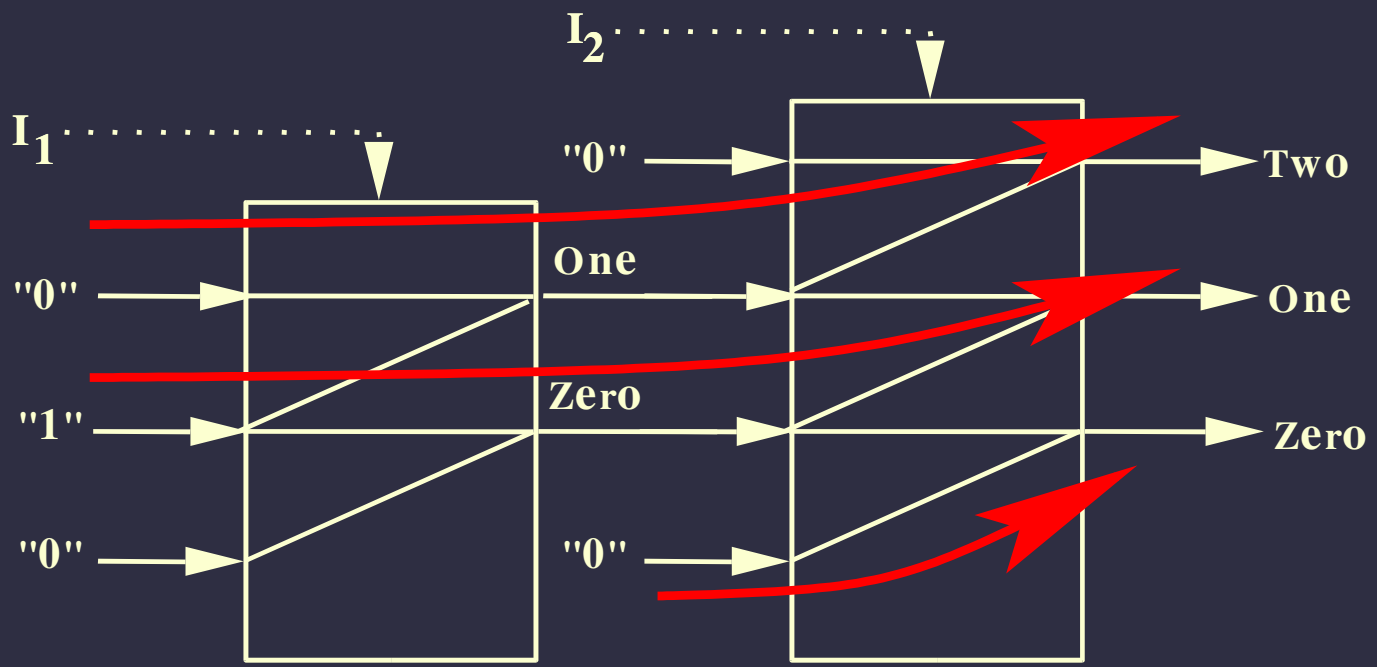
TG tally circuit



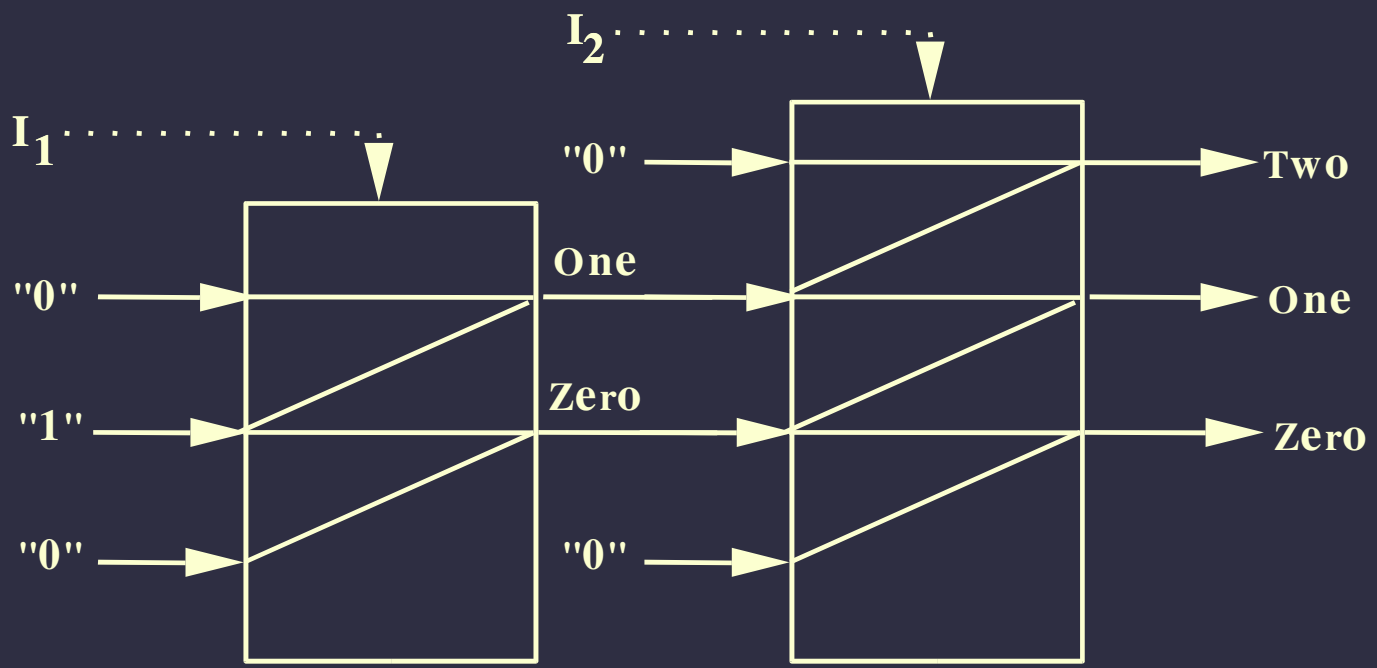
TG tally circuit



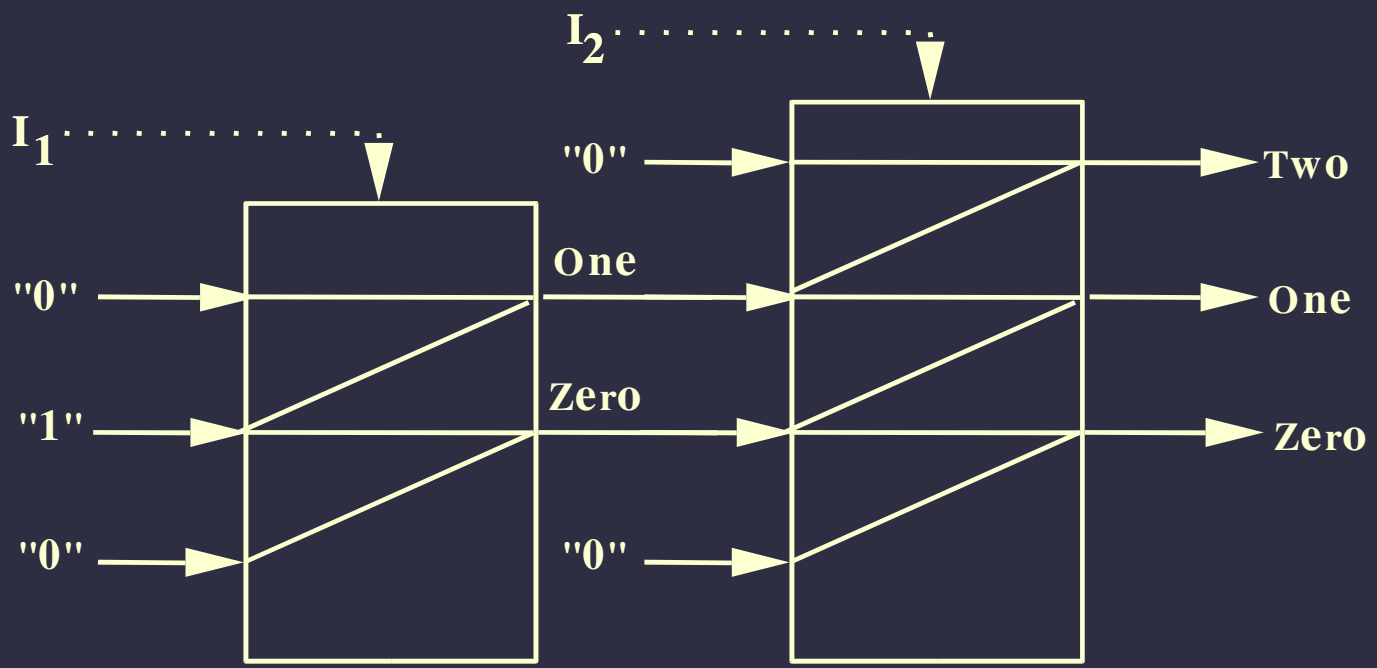
TG tally circuit



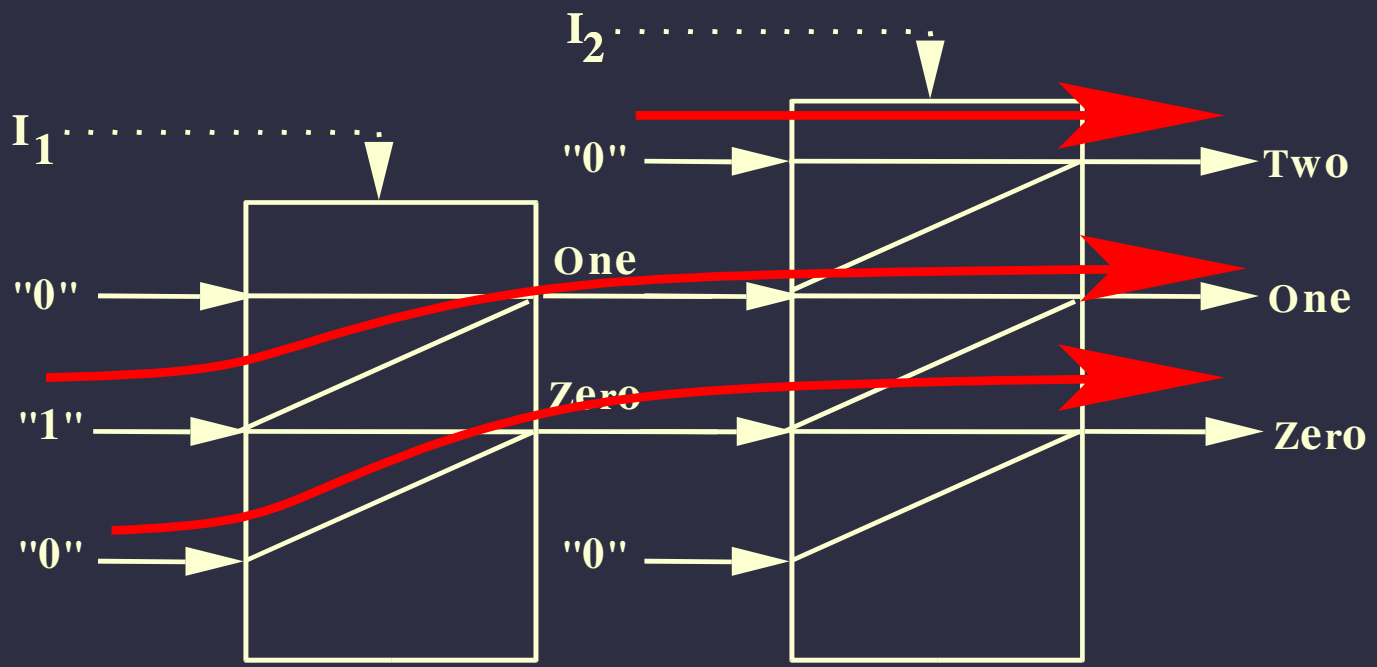
TG tally circuit



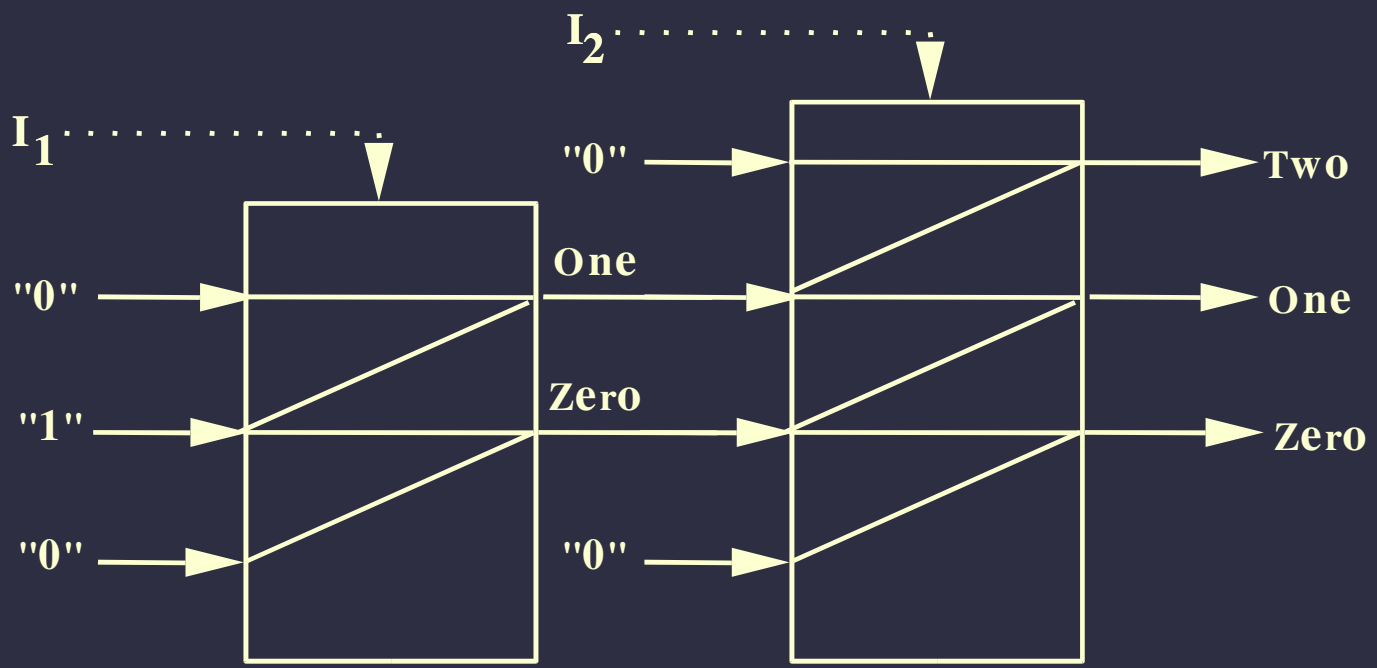
TG tally circuit



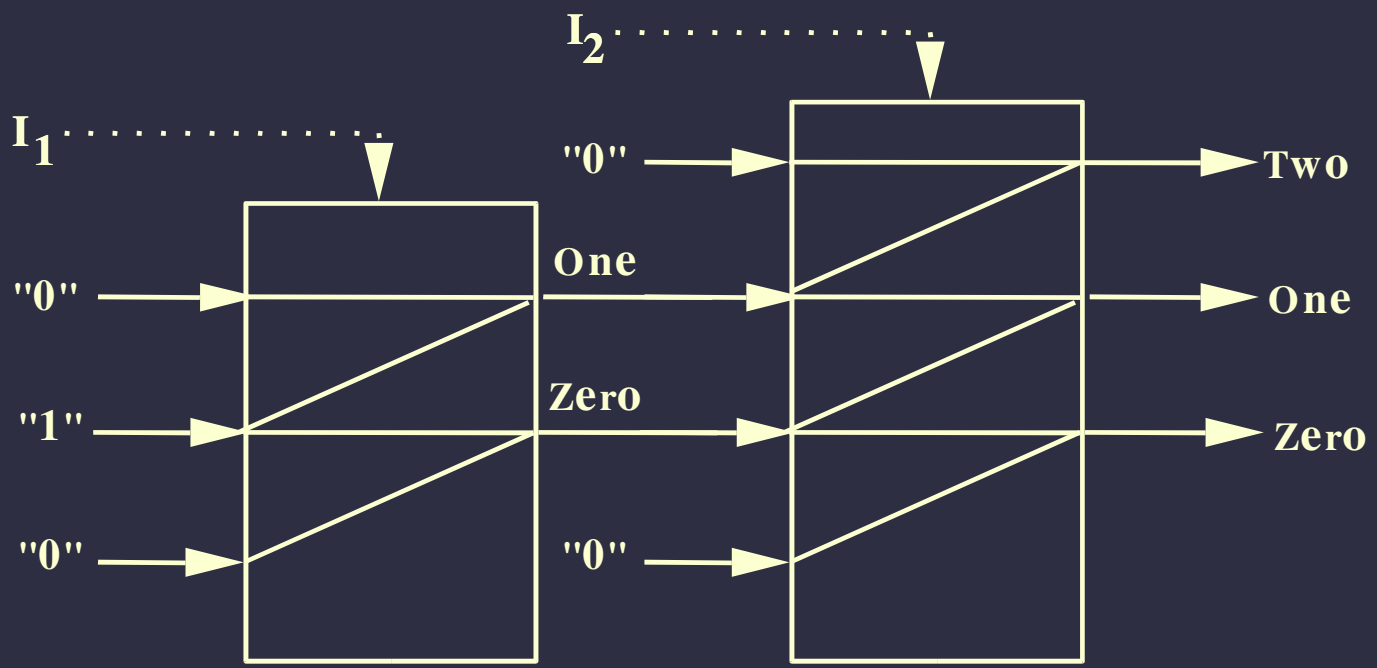
TG tally circuit



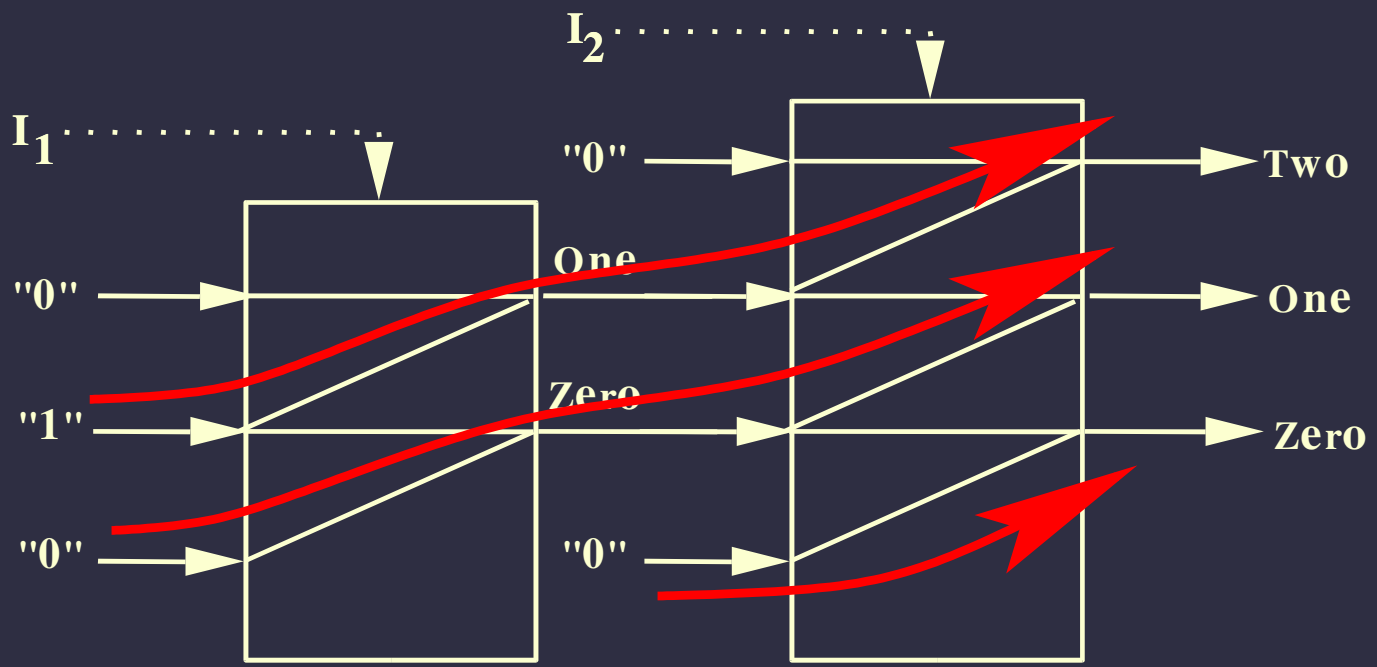
TG tally circuit



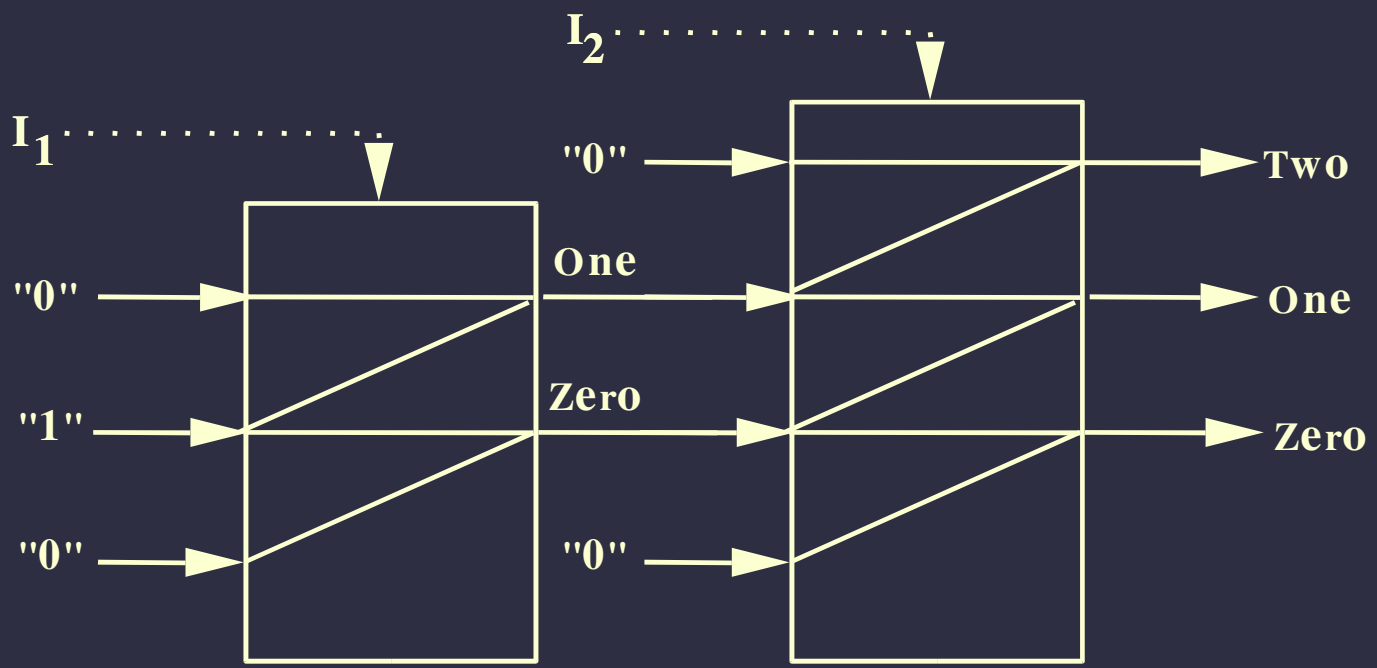
TG tally circuit



TG tally circuit



TG tally circuit



Section outline

1. Implementation technologies
   - Review of MUX composition
   - Steering logic
   - ROMs
   - FPGAs
   - Transformations for CMOS

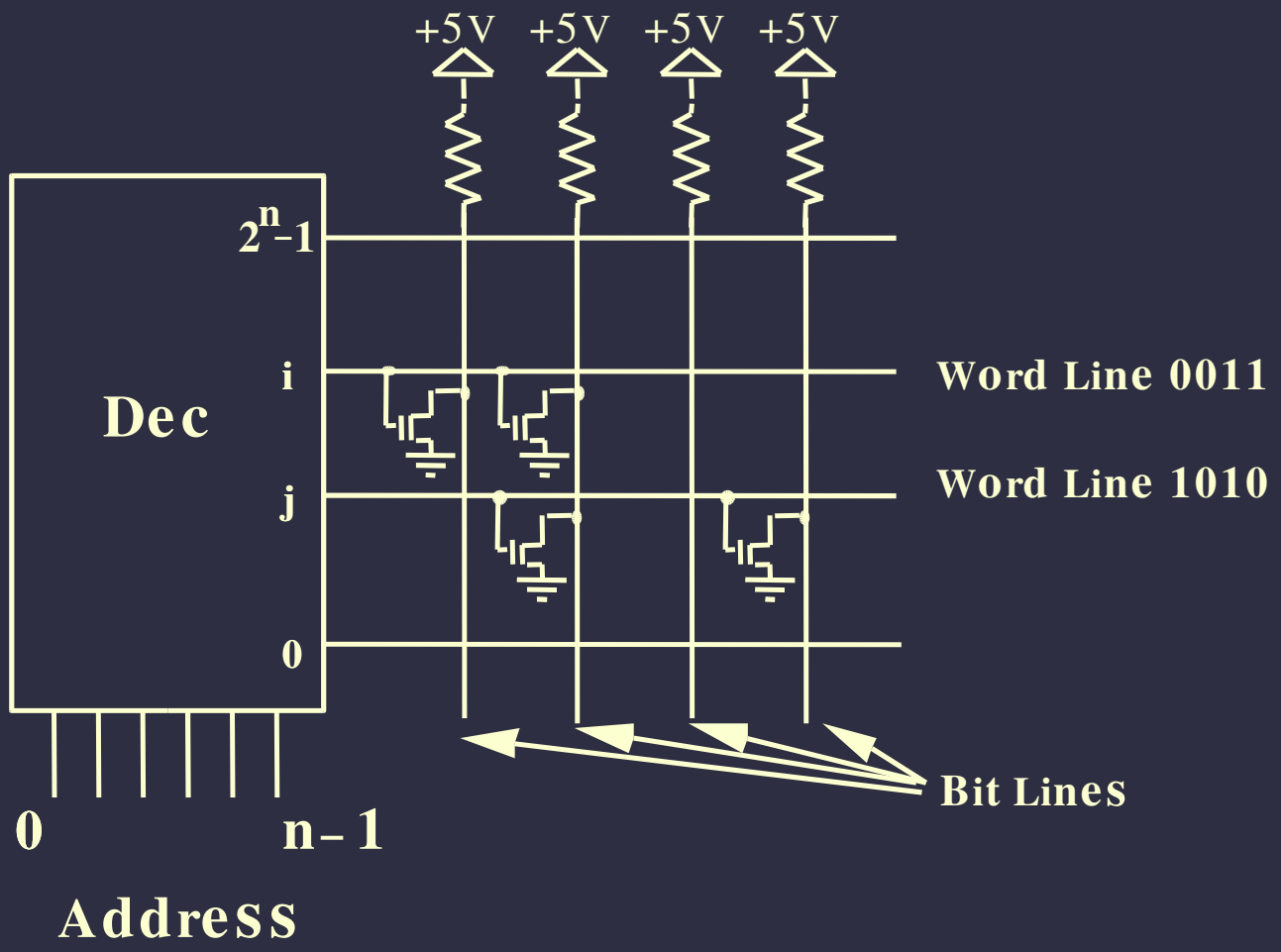
ROMs, FPGAs, and multi-level minimization

- Programmable read-only memories (PROMs)
- Field-programmable gate arrays (FPGAs)
- Programmable devices for prototyping

Programmable read-only memories (PROMs)

- 2-D array of binary values
- Input: Address
- Output: Word

PROM


+5V
+5V
+5V
+5V
2^n-1
i
j
0
Dec
Word Line 0011
Word Line 1010
Bit Lines
0
n–1
Address

Implementing logic with PROMs

$$F_0 = \overline{A}\ \overline{B}\ C + A\ \overline{B}\ \overline{C} + A\ \overline{B}\ C$$
$$F_1 = \overline{A}\ \overline{B}\ C + \overline{A}\ B\ \overline{C} + A\ B\ C$$
$$F_2 = \overline{A}\ \overline{B}\ \overline{C} + \overline{A}\ \overline{B}\ C + A\ \overline{B}\ \overline{C}$$
$$F_3 = \overline{A}\ B\ C + A\ \overline{B}\ \overline{C} + A\ B\ \overline{C}$$

Truth table

A	B	C	F_3	F_2	F_1	F_0
0	0	0	0	1	0	0
0	0	1	0	1	1	1
0	1	0	0	0	1	0
0	1	1	1	0	0	0
1	0	0	1	1	0	1
1	0	1	0	0	0	1
1	1	0	1	0	0	0
1	1	1	0	0	1	0

Truth table

A	B	C	F_3	F_2	F_1	F_0
0	0	0	0	1	0	0
0	0	1	0	1	1	1
0	1	0	0	0	1	0
0	1	1	1	0	0	0
1	0	0	1	1	0	1
1	0	1	0	0	0	1
1	1	0	1	0	0	0
1	1	1	0	0	1	0

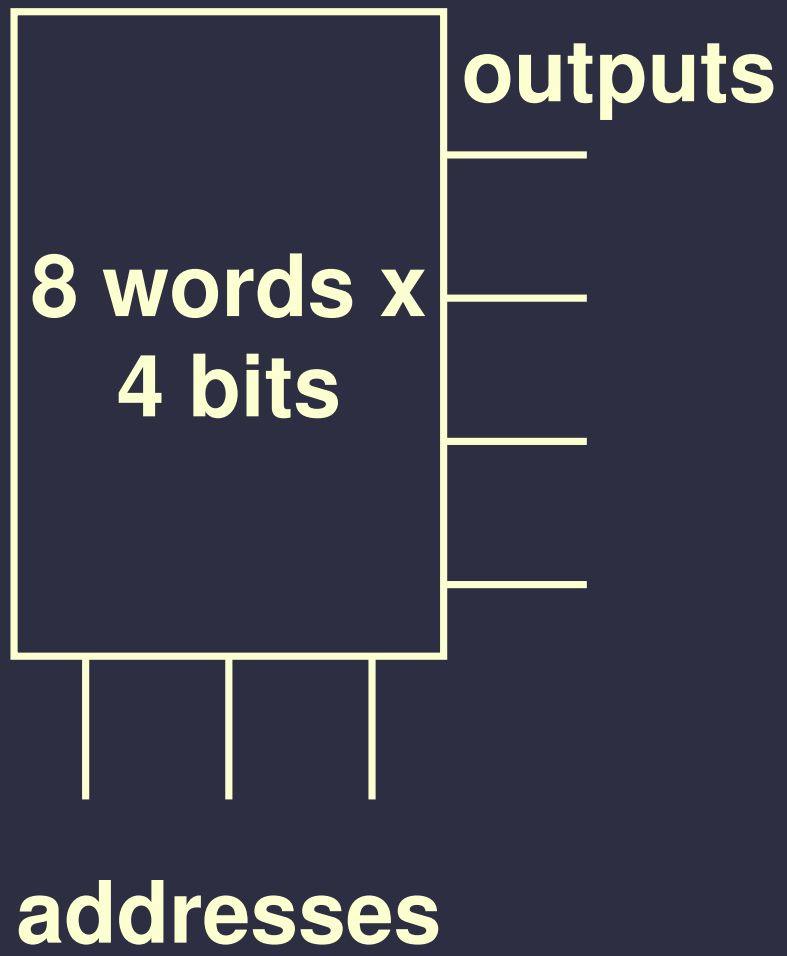
Address

Truth table

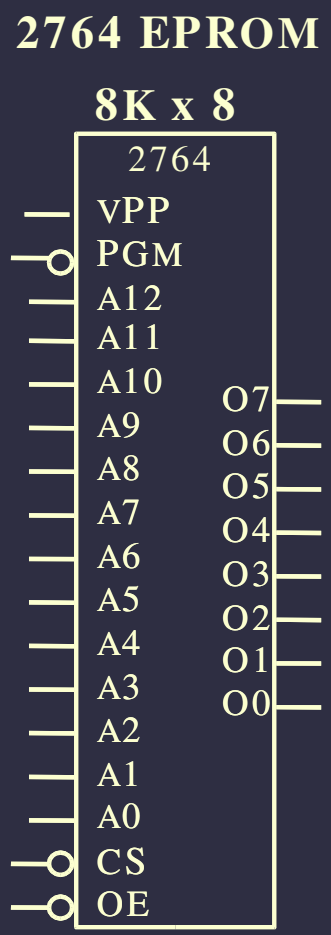
A	B	C	F_3	F_2	F_1	F_0
0	0	0	0	1	0	0
0	0	1	0	1	1	1
0	1	0	0	0	1	0
0	1	1	1	0	0	0
1	0	0	1	1	0	1
1	0	1	0	0	0	1
1	1	0	1	0	0	0
1	1	1	0	0	1	0

Word

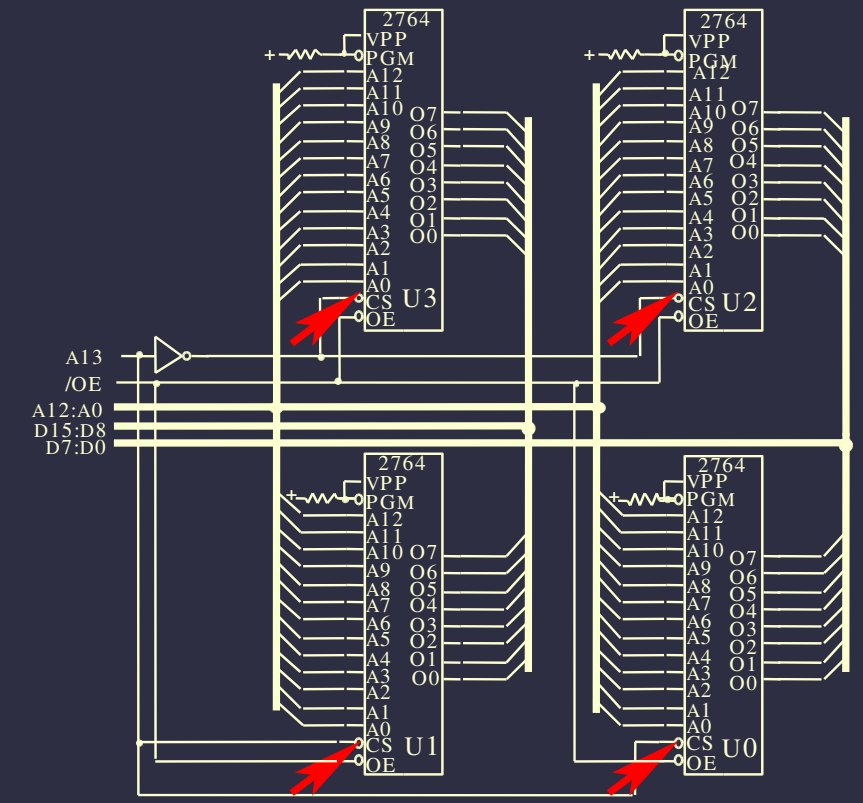
PROM suitable for implementing example



Memory composition

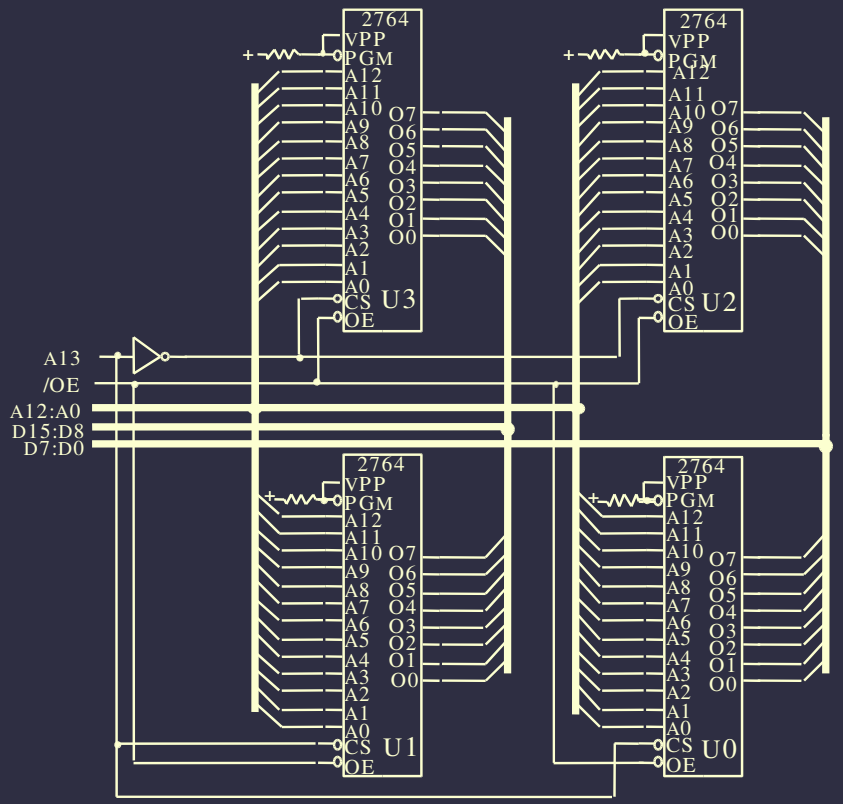


Memory composition



16K x 16

Memory composition



16K x 16

Chip select

PLA/PAL vs. PROM

PLA

- Takes advantage of don't-cares
  - Good at random logic
- Good when product terms shared

PAL

- More area-efficient for certain designs
- OR-plane can't be programmed, usually no sharing

PLA/PAL vs. PROM

PROM

- Design trivial
- Can't take advantage of don't-cares
  - Area-inefficient
- Product/sum terms not shared

Section outline

1. Implementation technologies
   - Review of MUX composition
   - Steering logic
   - ROMs
   - FPGAs
   - Transformations for CMOS

Field-programmable gate arrays (FPGAs)

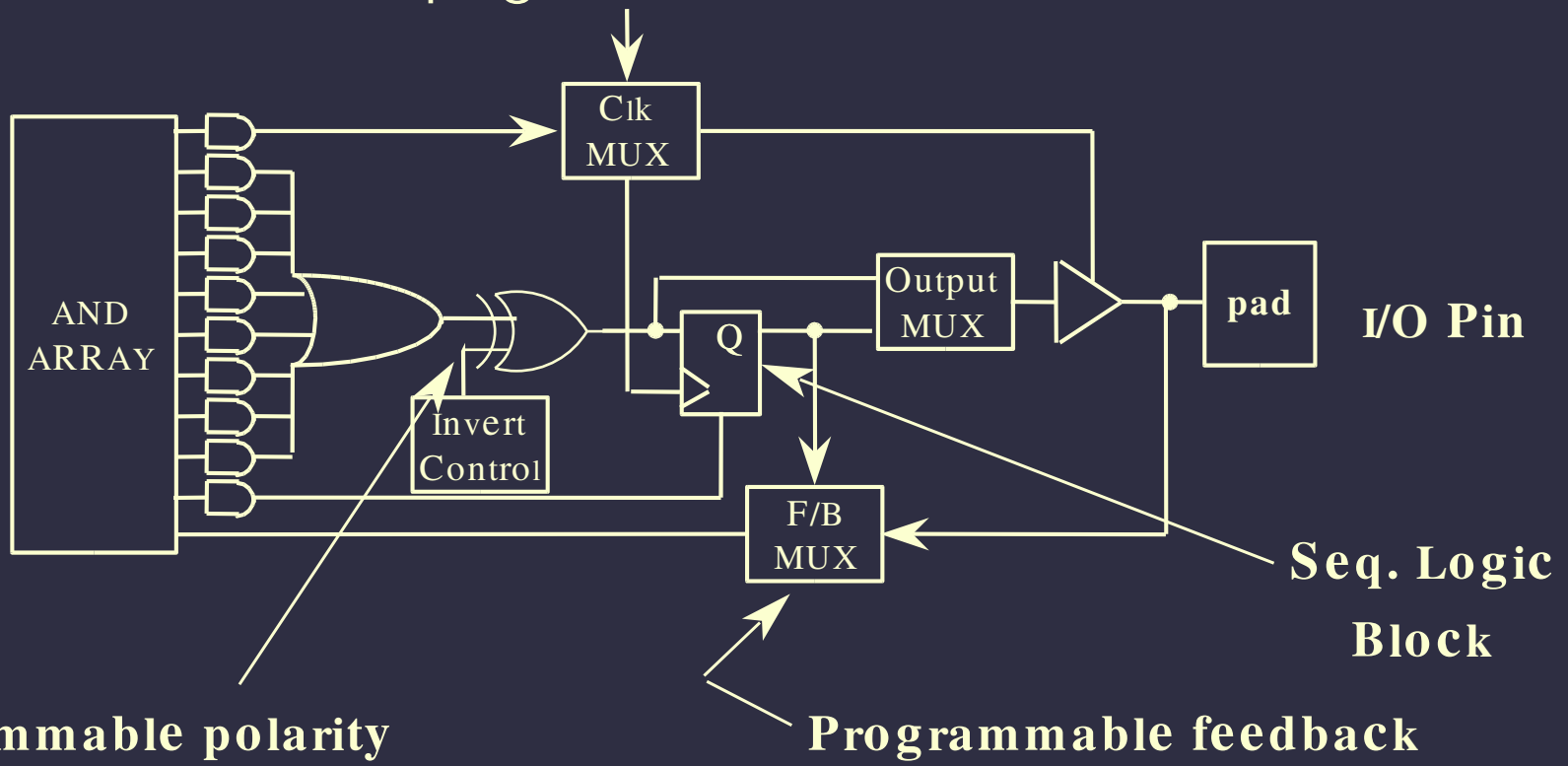
- PLAs
  - 10–100 gate equivalent
- FPGAs
  - Altera
  - Actel
  - Xilinx
  - 100–1,000,000 gate equivalent

Altera EPLDs

- Each has from 8–48 macrocells
- Macrocell behavior controlled with EPROM bits
- Can be used sequentially
- Has synchronous and asynchronous modes

Altera erasable programmable logic devices (EPLDs)

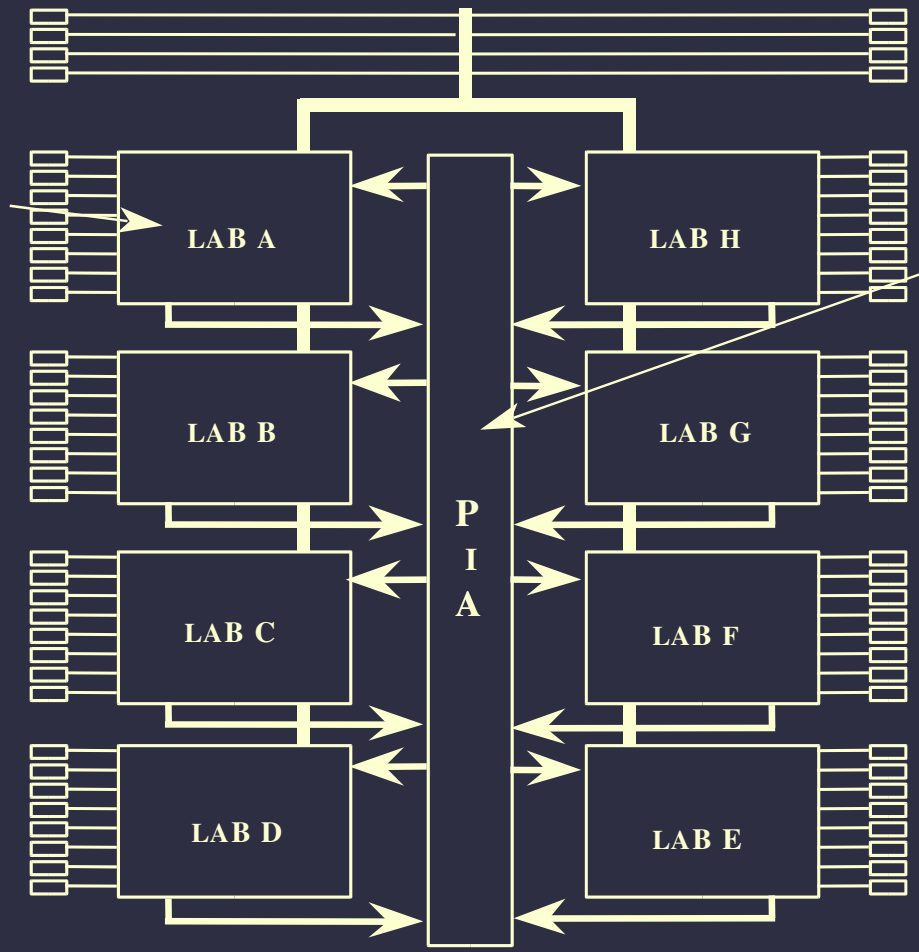
Composed of many *macrocells* – 8 product term AND/OR array with programmable MUXs



Multiple array matrix (MAX)

- Altera macrocells quite limited
  - Can't share product terms between macrocells
- Workaround: Connect together macrocells with programmable interconnect

Multiple array matrix (MAX)



Logic Array Blocks

(Similar to macrocells)

Global Routing: Programmable Interconnect

Array EPM5128:

8 Fixed Inputs

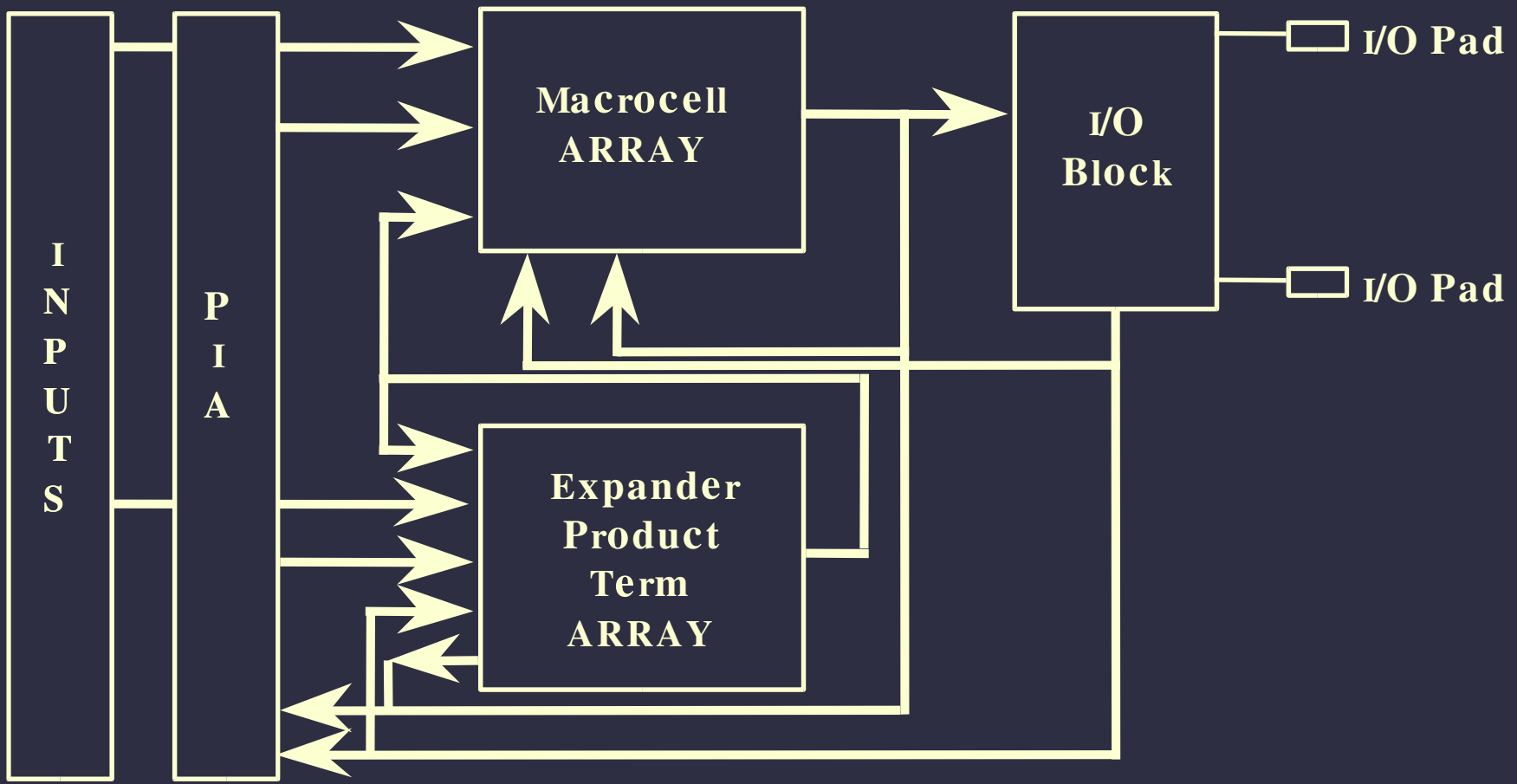
52 I/O Pins

8 LABs

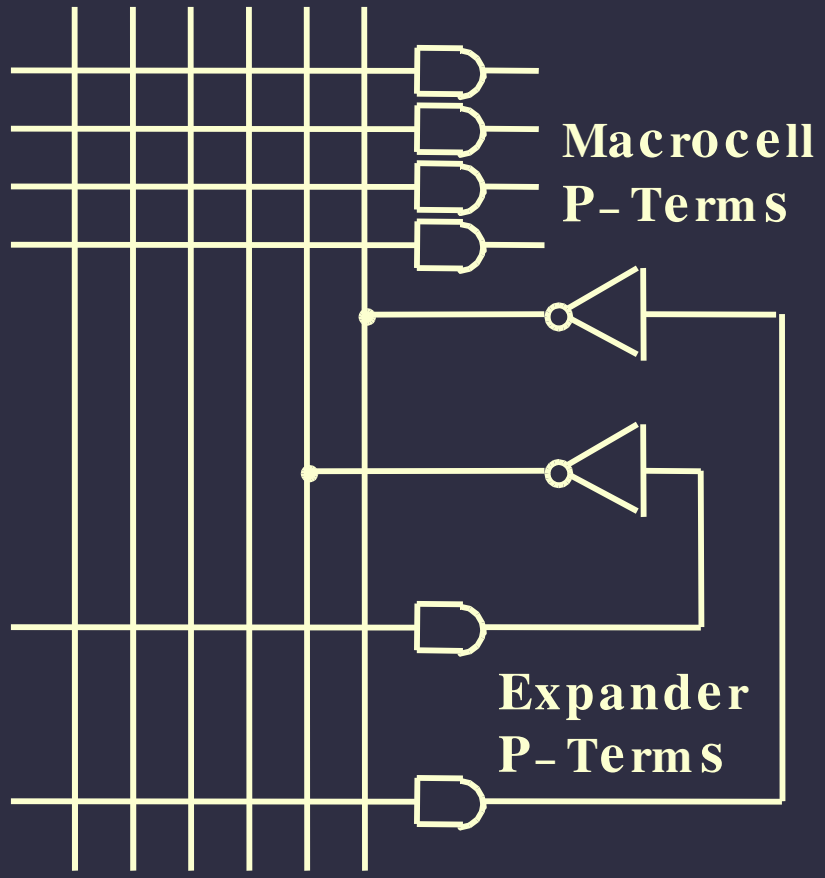
16 Macrocells/LAB

32 Expanders/LAB

MAX expander terms

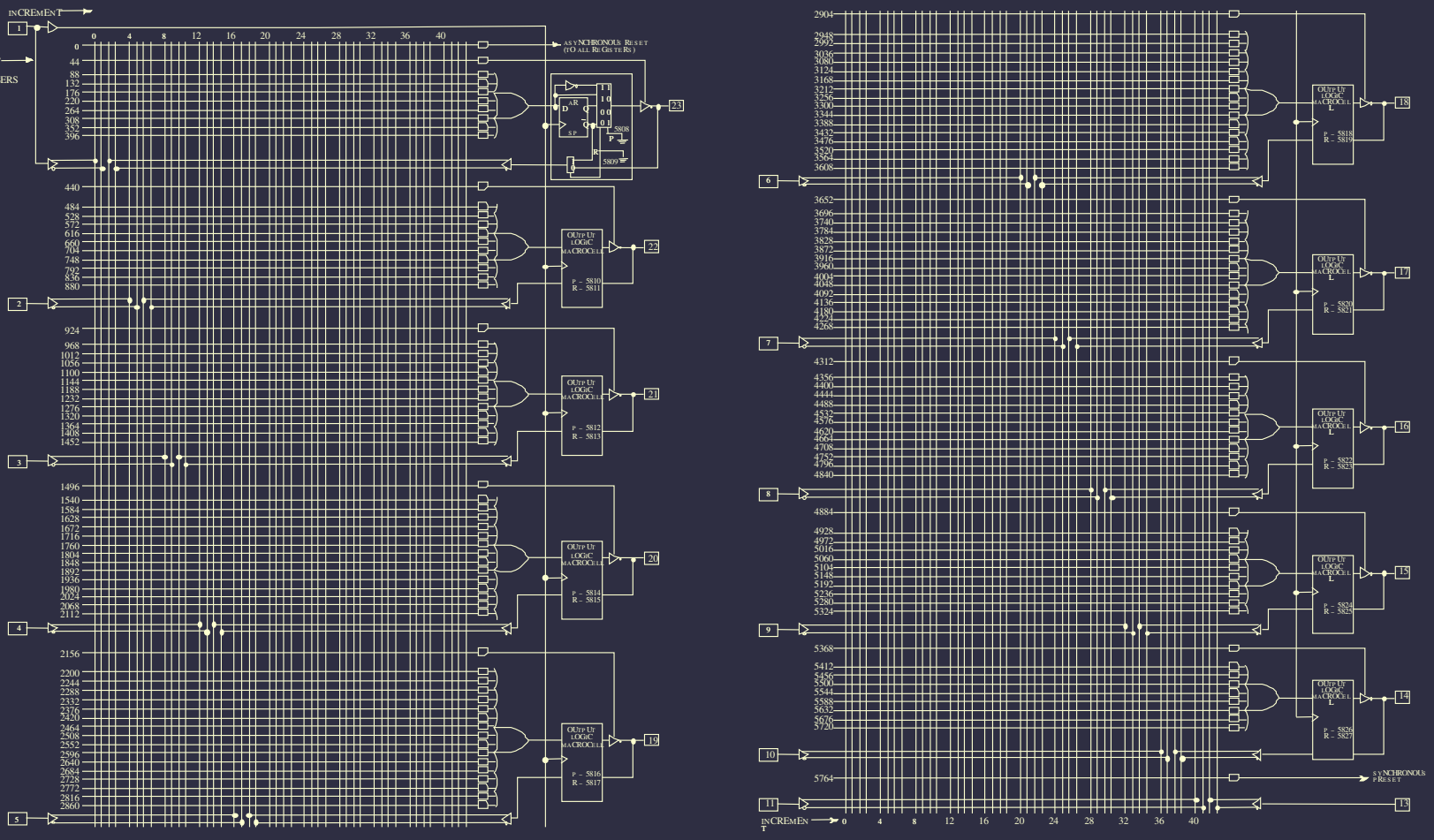


MAX expander terms



Expander product terms shared among all macrocells

Altera 22V10 PAL



Altera 22V10 PAL

- Many product terms per output
- Latches and MUXs associated with outputs
- 22 IO pins
- 10 may be used as outputs

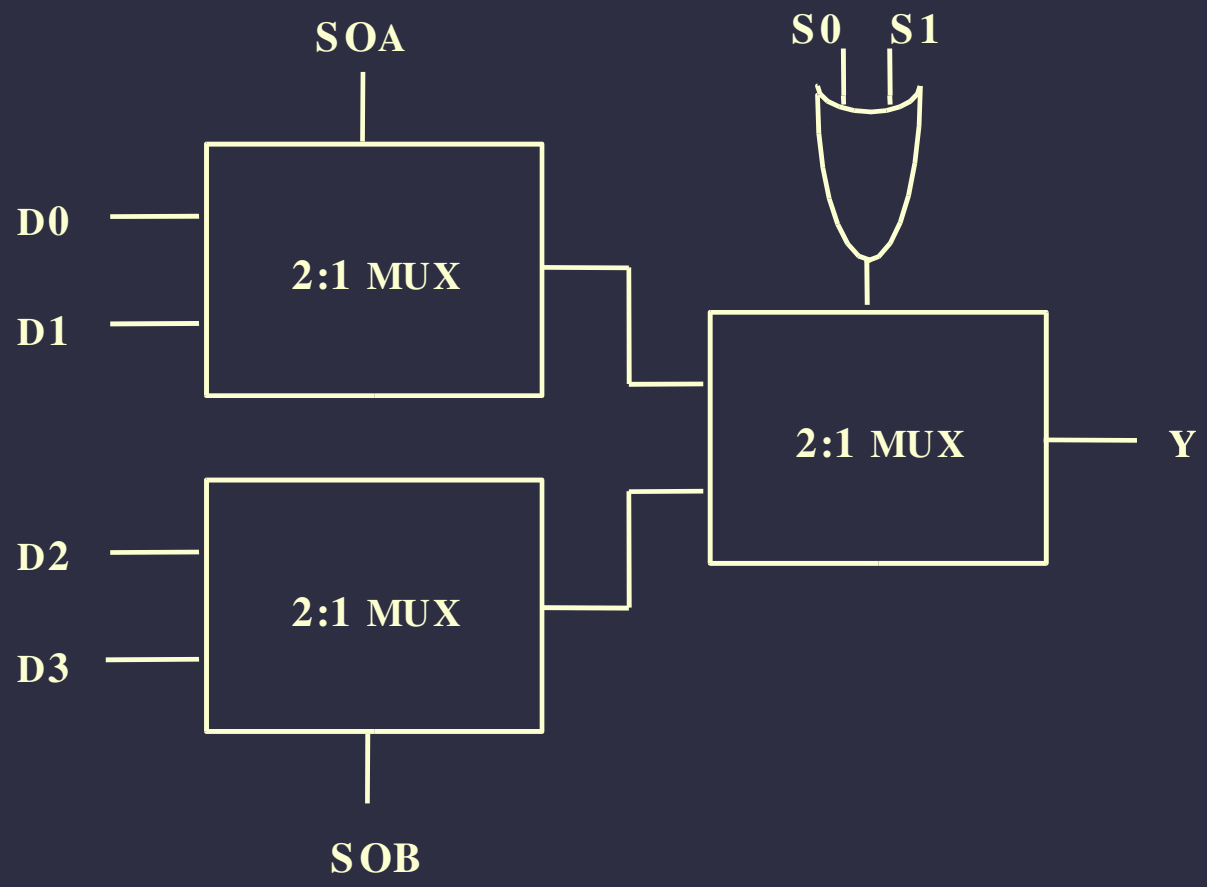
Actel programmable gate arrays

- Rows of programmable logic blocks
- Rows of interconnect
- Columns of interconnect
- Attach to rows using antifuses

Actel programmable gate arrays

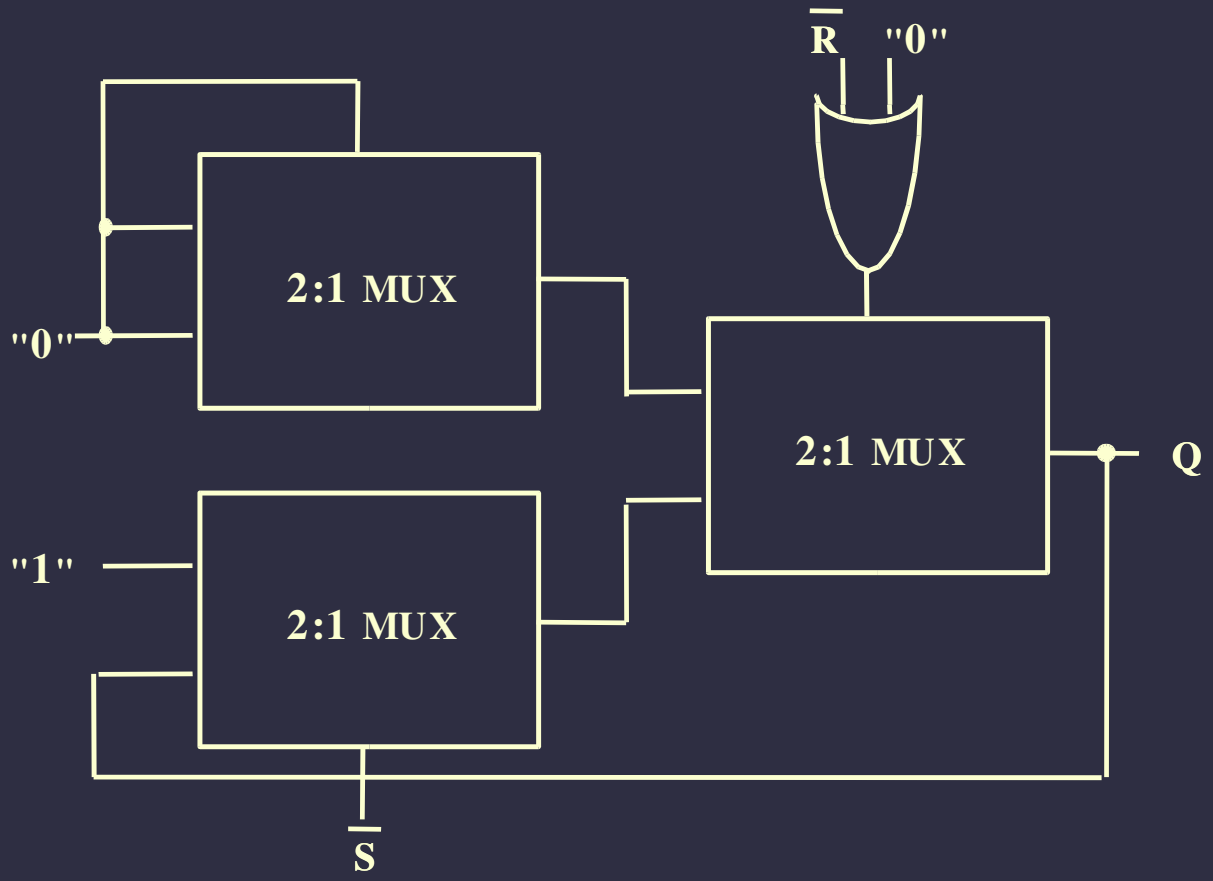
- Each combinational logic block has 8 inputs, 1 output
- No built-in sequential elements
- Build flip-flops using logic blocks

Actel logic block



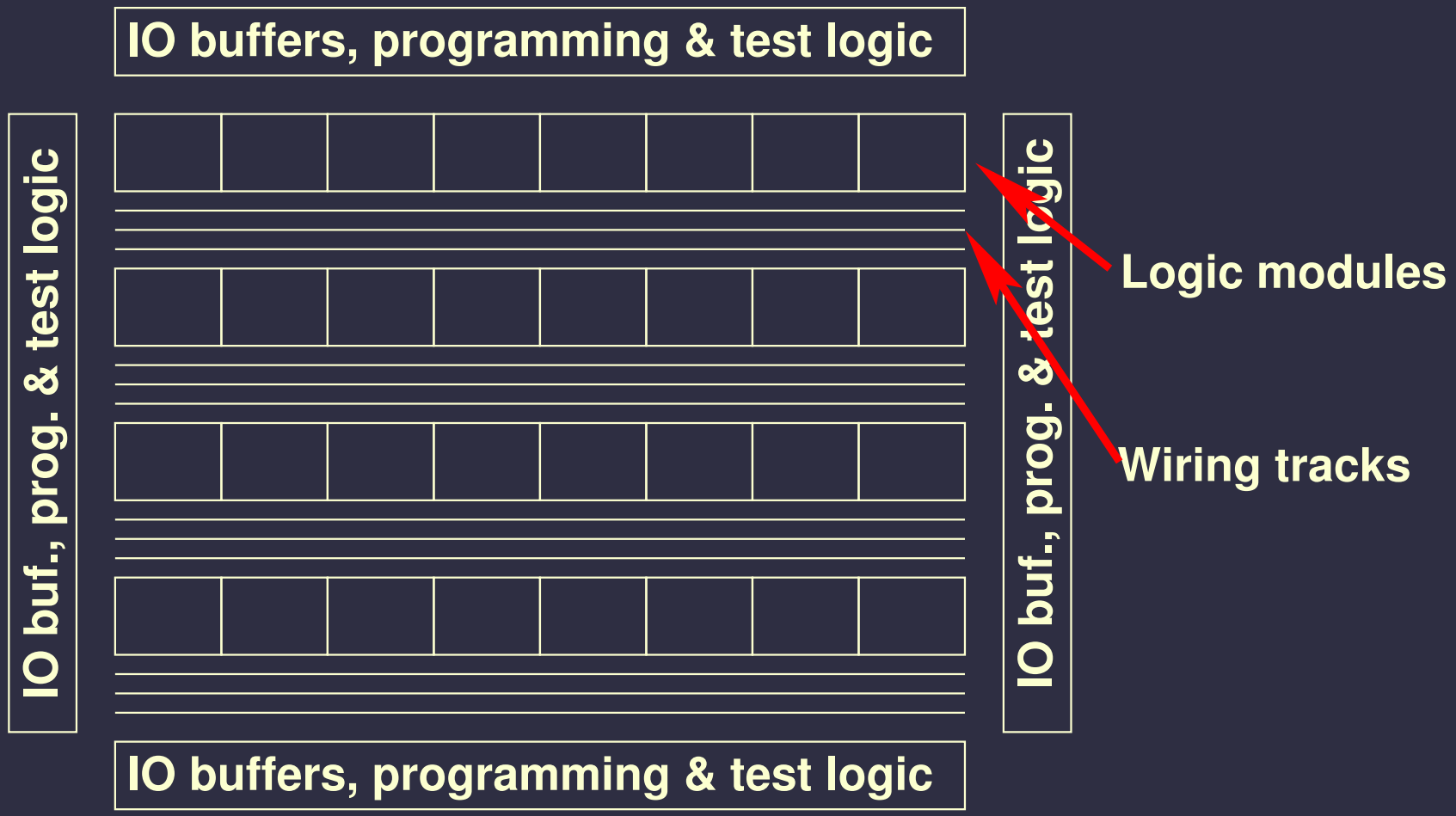
Modified 4:1 MUX

Actel logic block

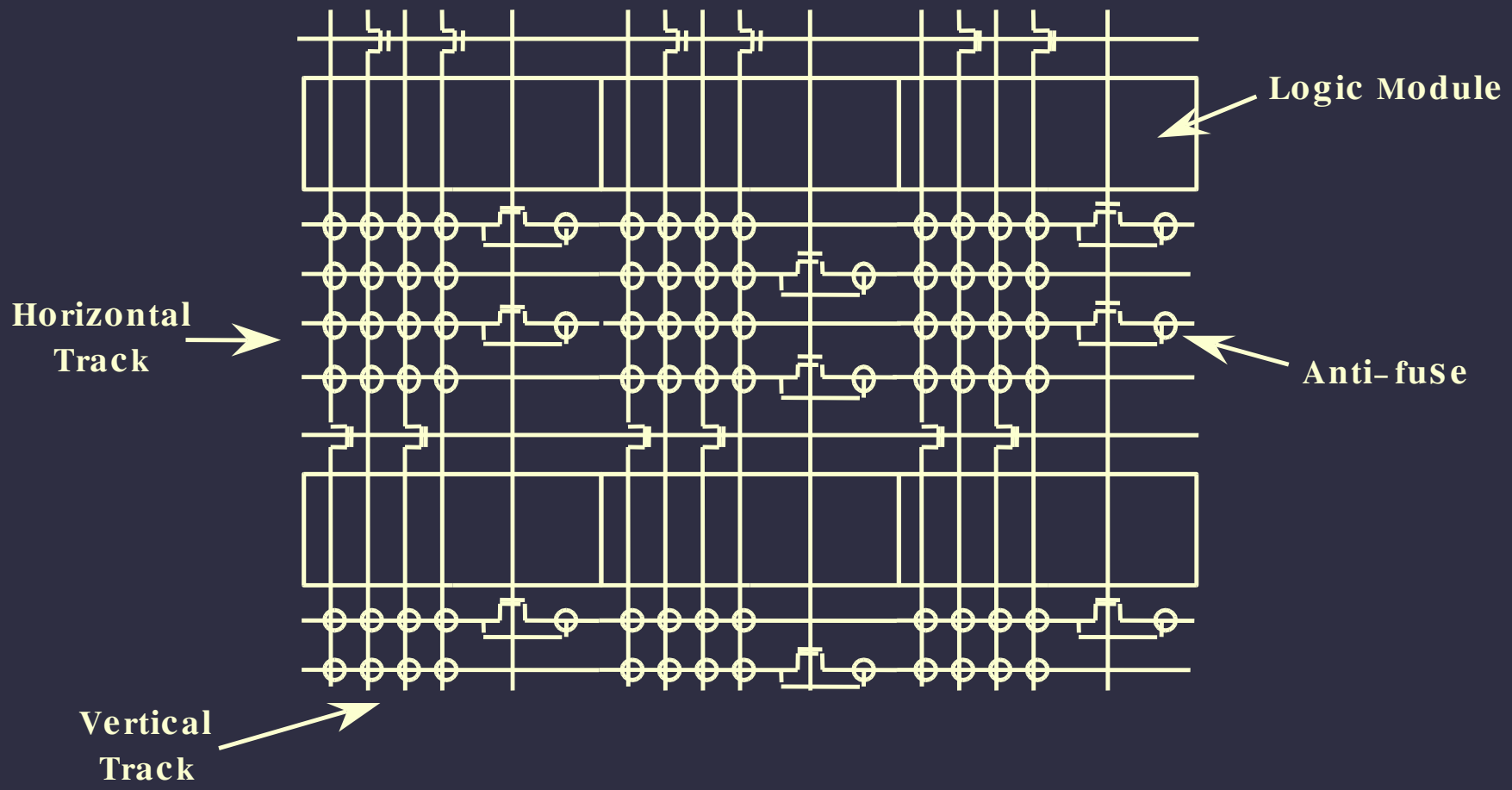


Cross-couple for sequential use

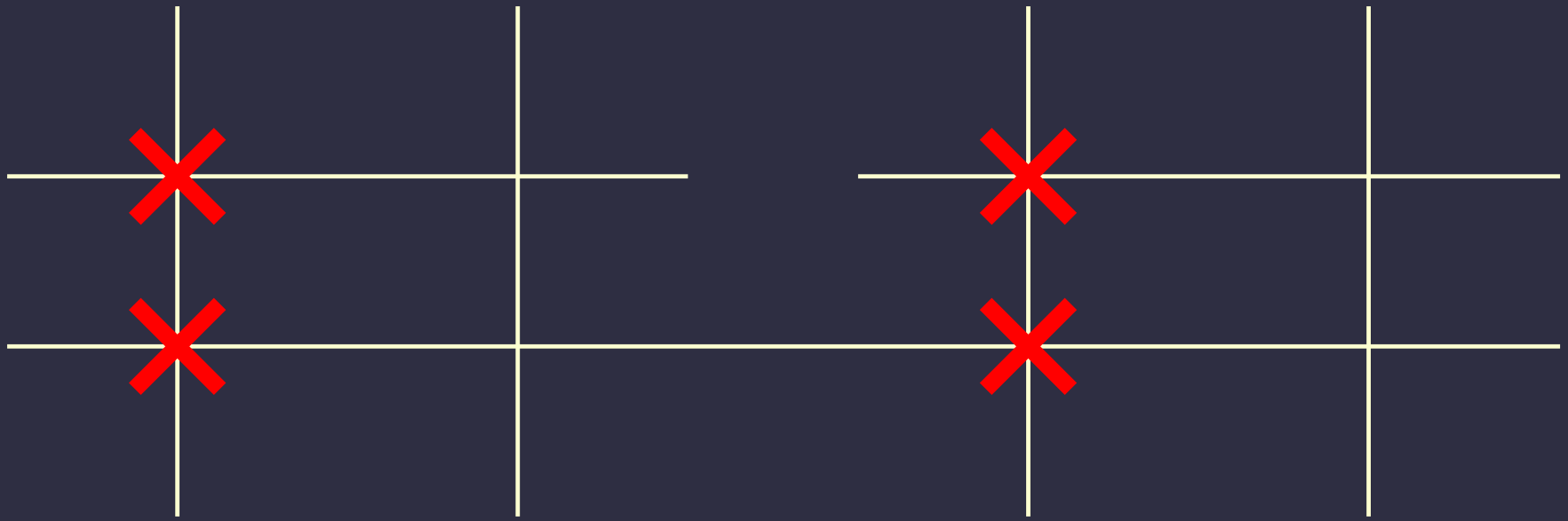
Actel programmable gate arrays



Actel interconnect

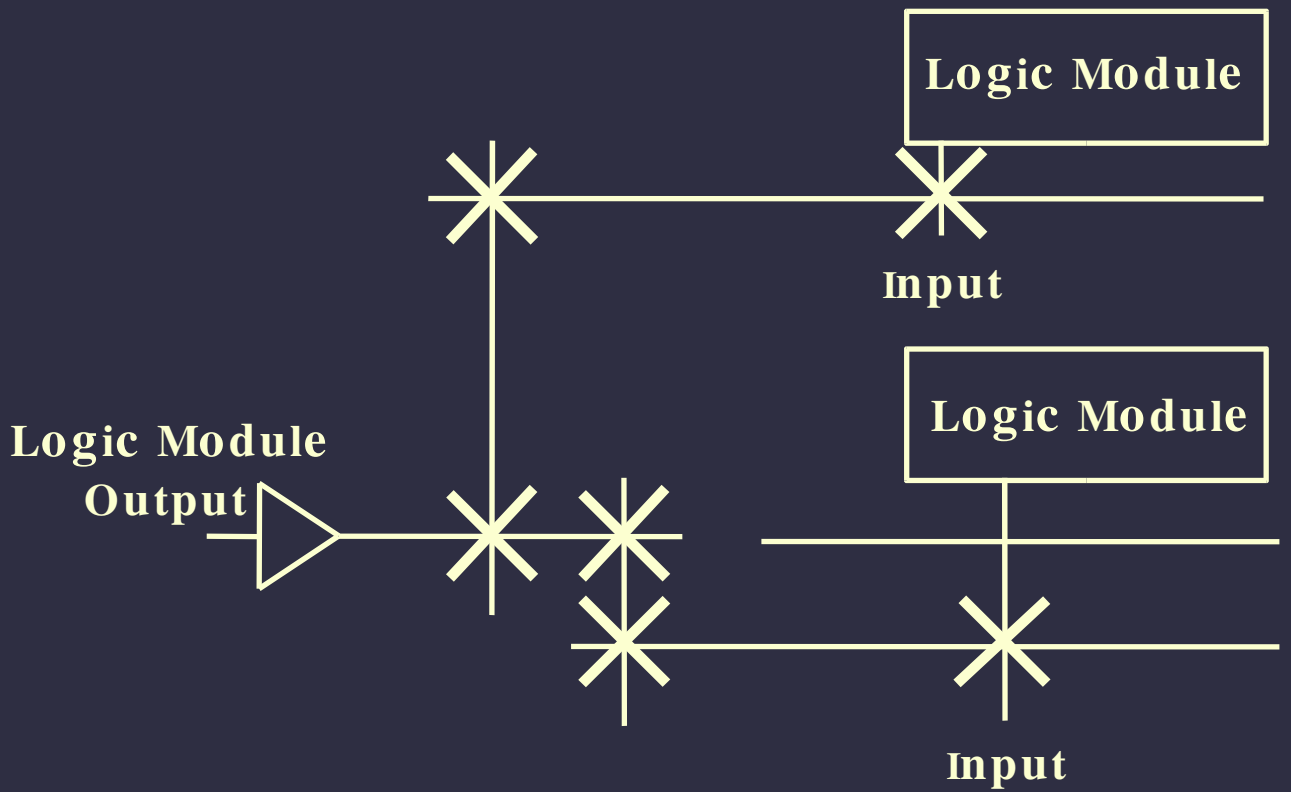


Antifuse routing



Build long routing lines from short segments

Actel routing example



- Minimize number of antifuse hops for critical path
- 2-3 hops for most interconnections

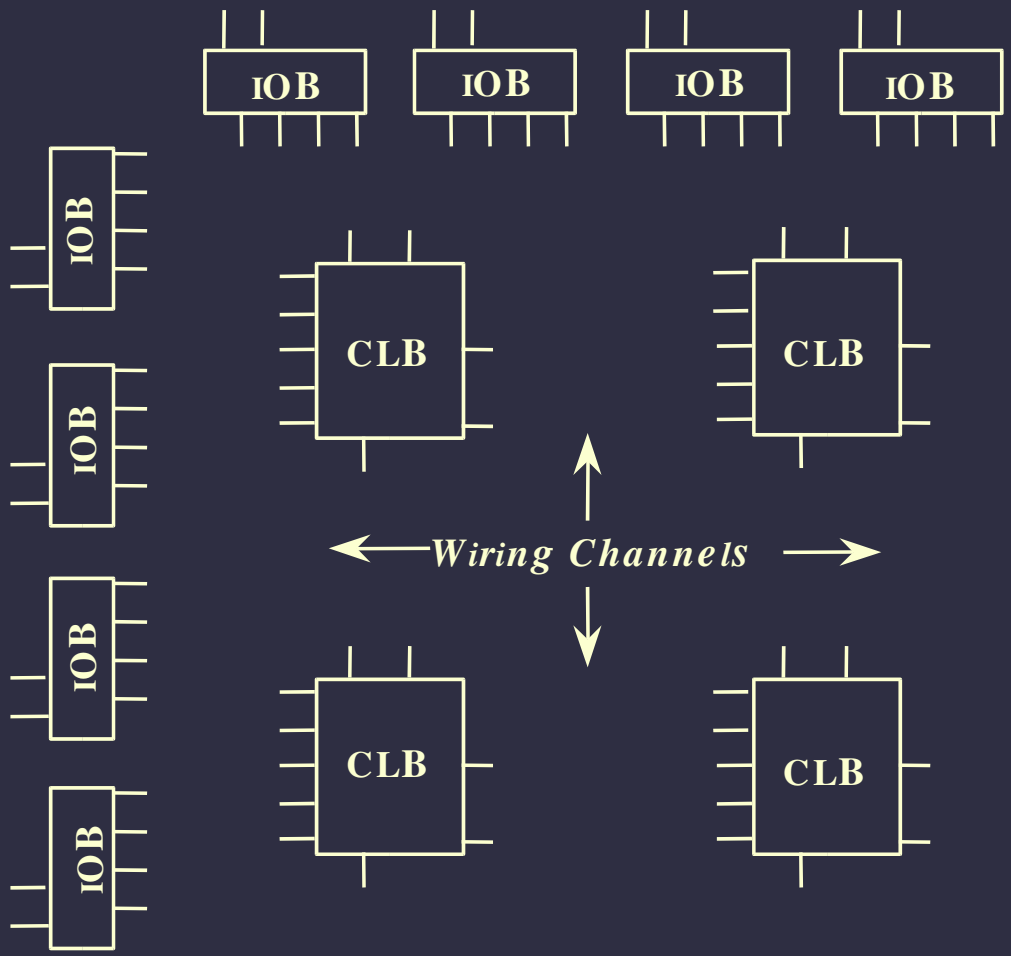
Xilinx logic cell arrays (LCAs)

- CMOS static RAM
  - Run-time programmable
- Serial shift-register based programming
- Program on power-up (external PROM)

Xilinx LCA components

- Configurable logic blocks (CLBs)
- IO blocks (IOBs)
- Wiring channels

Xilinx LCAs


IOB
IOB
IOB
IOB
IOB
IOB
IOB
IOB
CLB
CLB
Wiring Channels
CLB
CLB

Xilinx LCA features

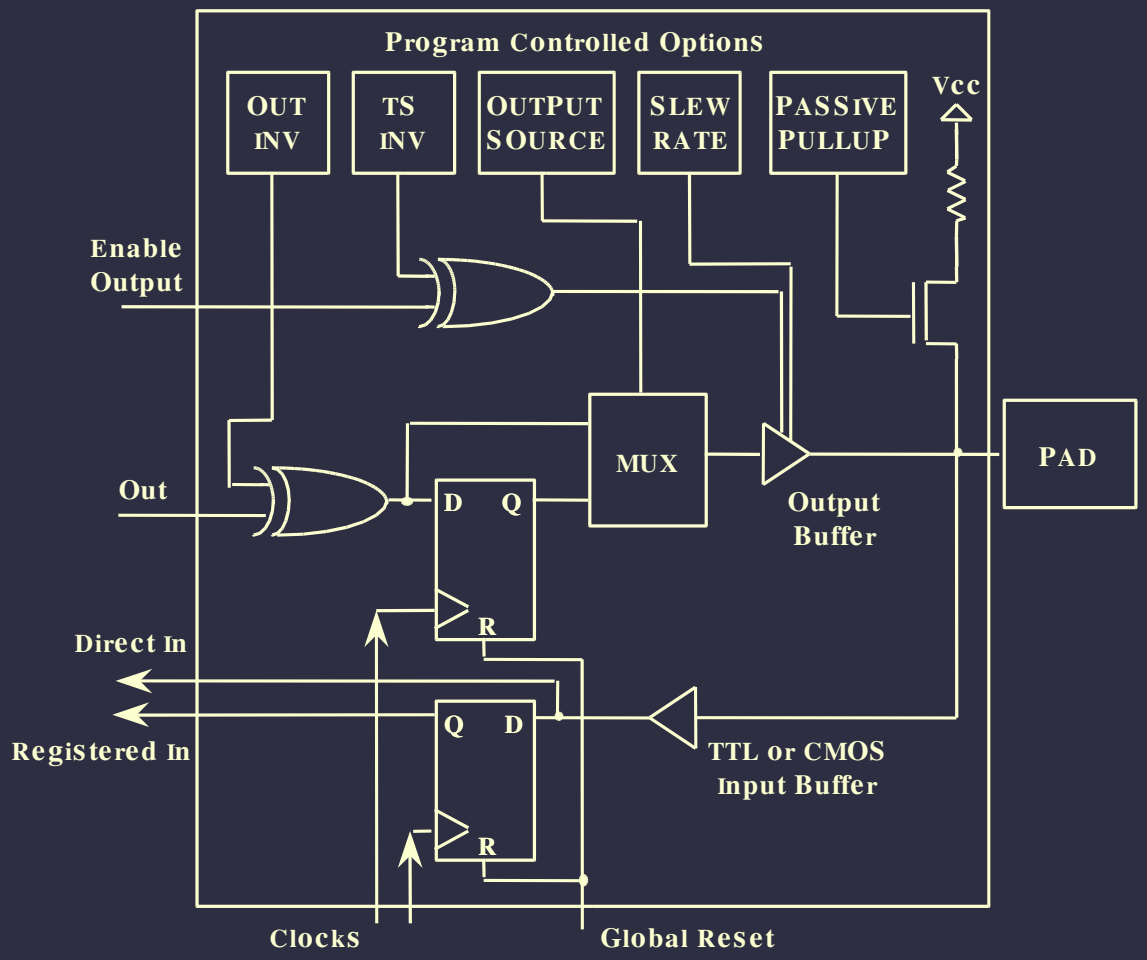
Inputs

- Input variables
- Tri-state (high-Z) enable bit for output
- Output clocks

Xilinx LCA features

- Output the input bit
- Contains internal flip-flops for inputs and outputs
- Fast and slow outputs available, e.g., 5 ns vs. 30 ns
  - Slower option limits slew rate
  - Lower noise
  - Lower power consumption

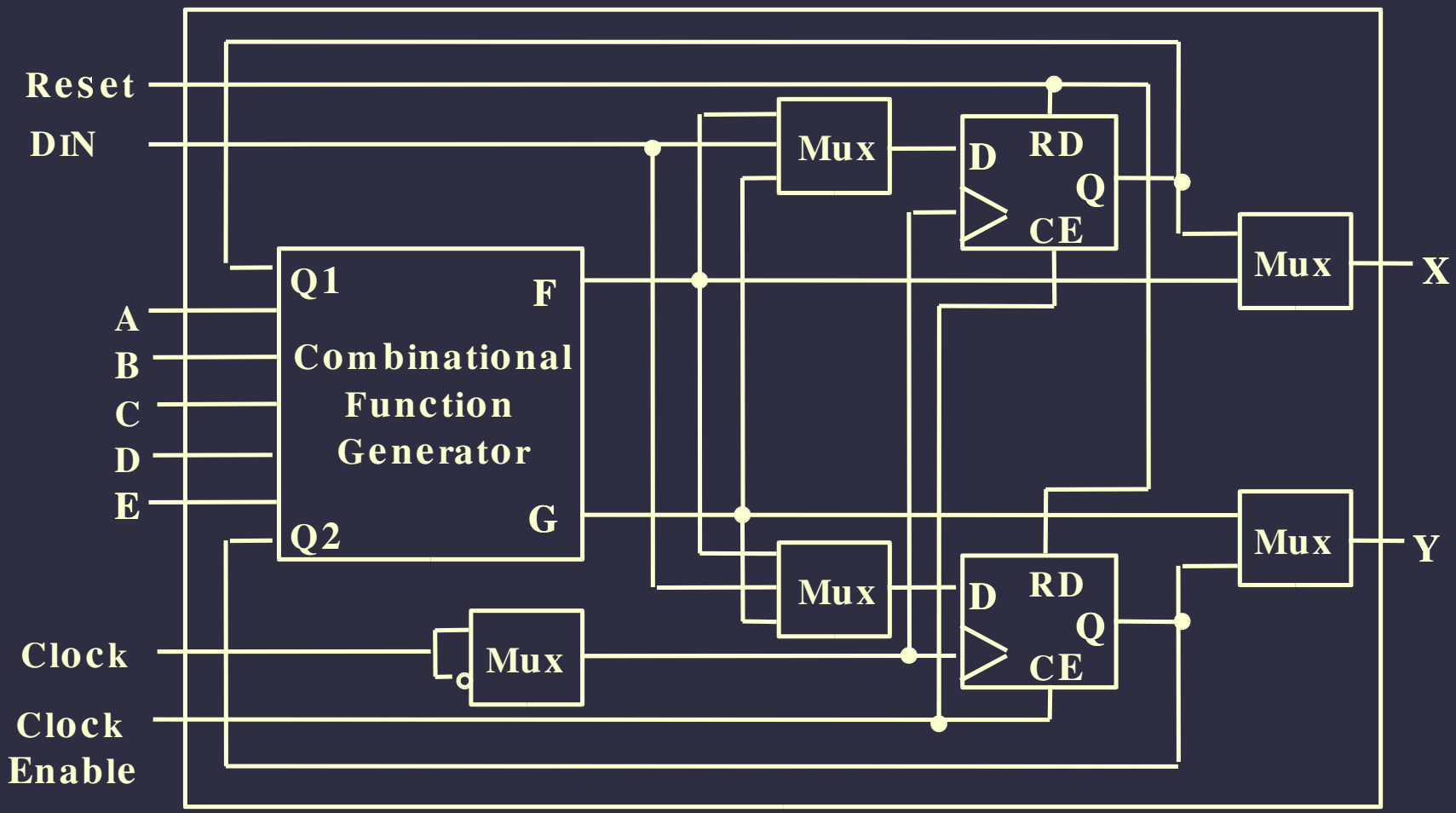
Xilinx LCA


Program Controlled Options
OUT INV
TS INV
OUTPUT SOURCE
SLEW RATE
PASSIVE PULLUP
Vcc
Enable Output
Out
D
Q
MUX
PAD
Output Buffer
R
Direct In
Registered In
Q
D
TTL or CMOS Input Buffer
R
Clocks
Global Reset

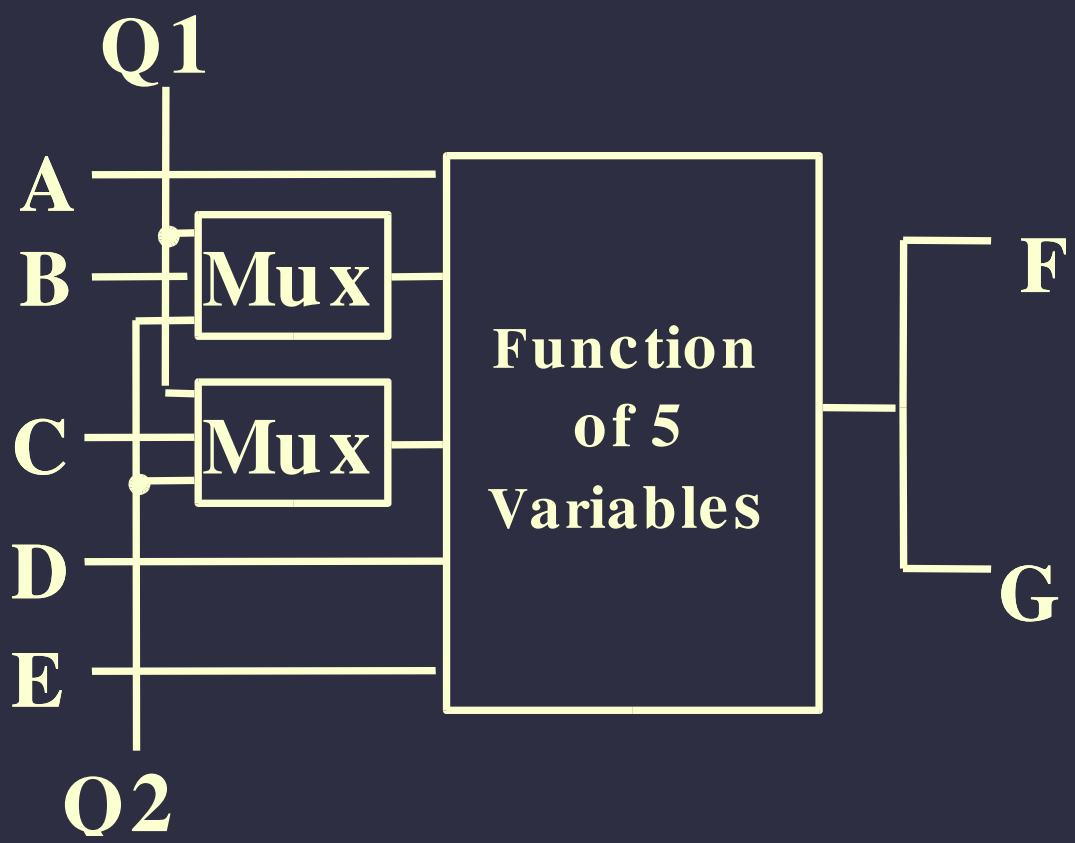
Xilinx CLB

- 2 flip-flops
- General function of 4 variables
- 2 non-general functions of 5 variables
- Certain special-case functions of 6 variables
- Global reset
- Clock
- Clock enable
- Independent input, DIN

Xilinx CLB

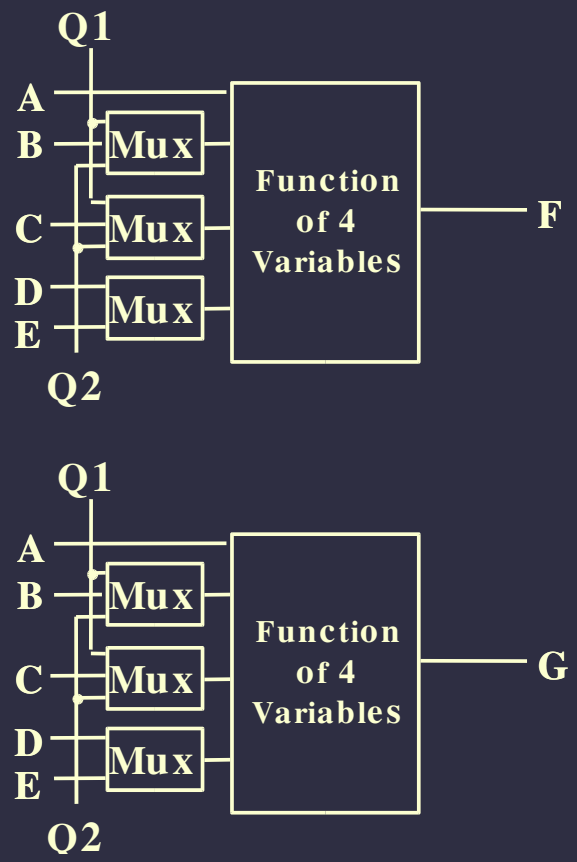

Reset
DIN
A
B
C
D
E
Clock
Clock Enable
Q1
F
Combinational Function Generator
G
Q2
Mux
Mux
Mux
Mux
D
RD
Q
CE
D
RD
Q
CE
X
Y

Function generator



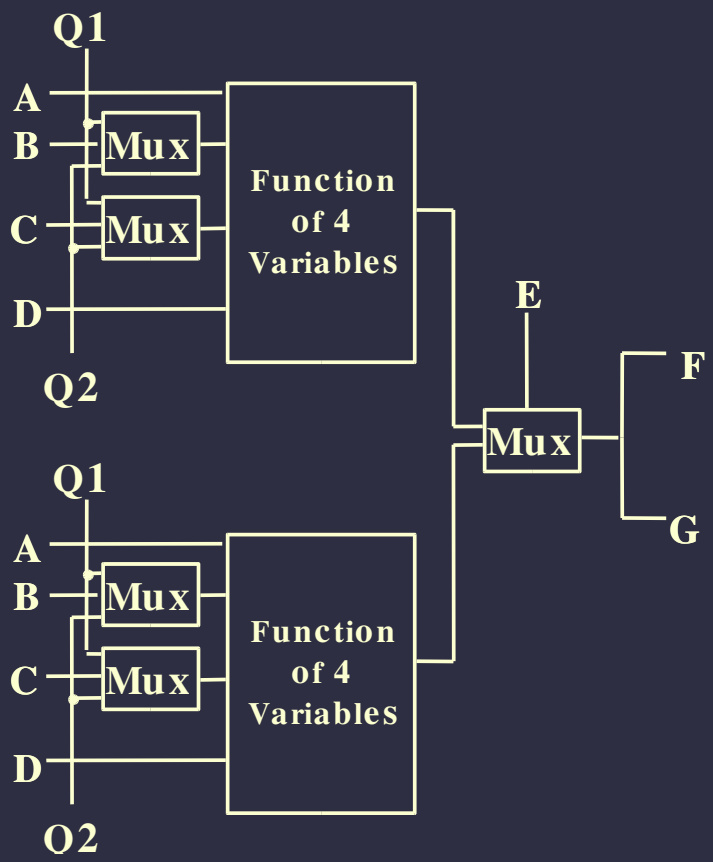
Two constrained functions of five variables

Function generator



Two arbitrary functions of four variables

Function generator



Certain limited functions of 6 variables

PARITY5 CLB cost example

- Determine whether the number of 1s is even or odd
- $F = A \oplus B \oplus C \oplus D \oplus E$
- Implement using 1 CLB

2-bit comparator CLB cost example

$$A\ B = C\ D \text{ or } A\ B > C\ D$$

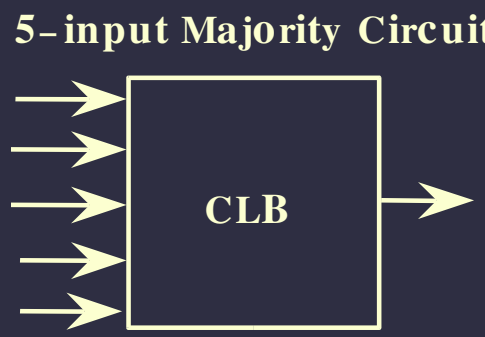
$$GT = A\ \overline{C} + A\ B\ \overline{D} + B\ \overline{C}\ \overline{D}$$

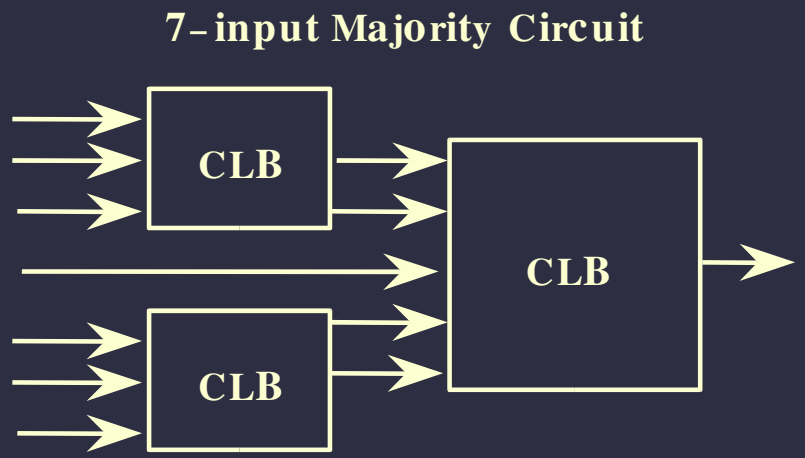
$$EQ = \overline{A}\ \overline{B}\ \overline{C}\ \overline{D} + \overline{A}\ B\ \overline{C}\ D + A\ \overline{B}\ C\ \overline{D} + A\ B\ C\ D$$

Only 1 CLB required

Majority CLB cost example

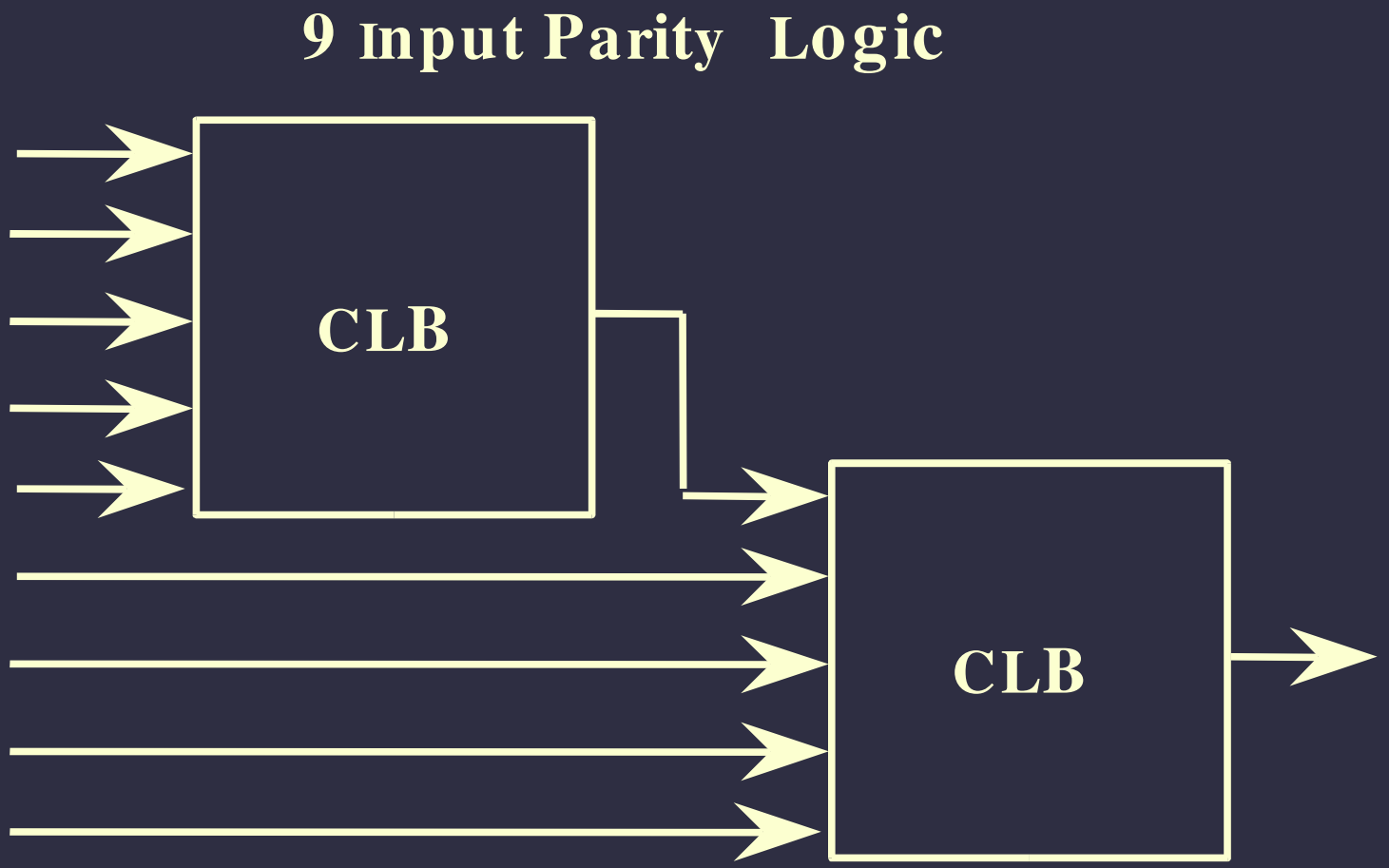
High whenever $\lceil n/2 \rceil$ outputs are high





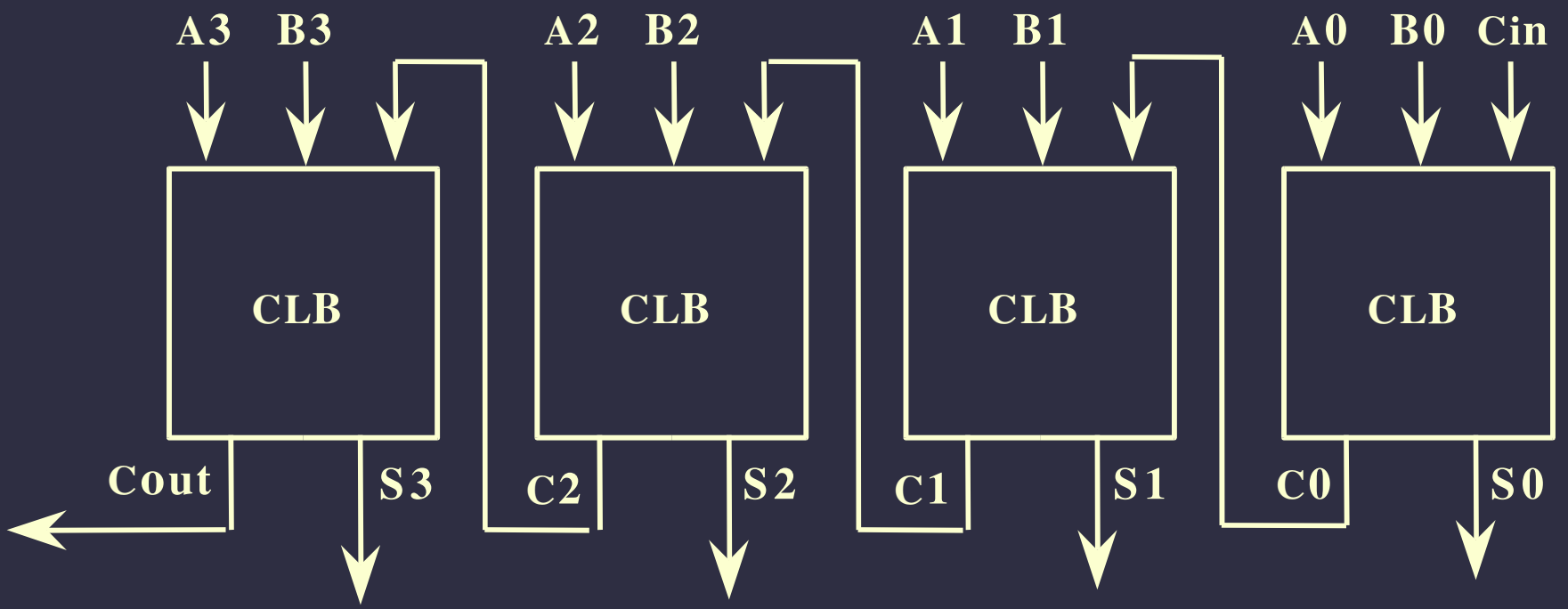
Large parity CLB cost example

2 levels allow up to 25 inputs



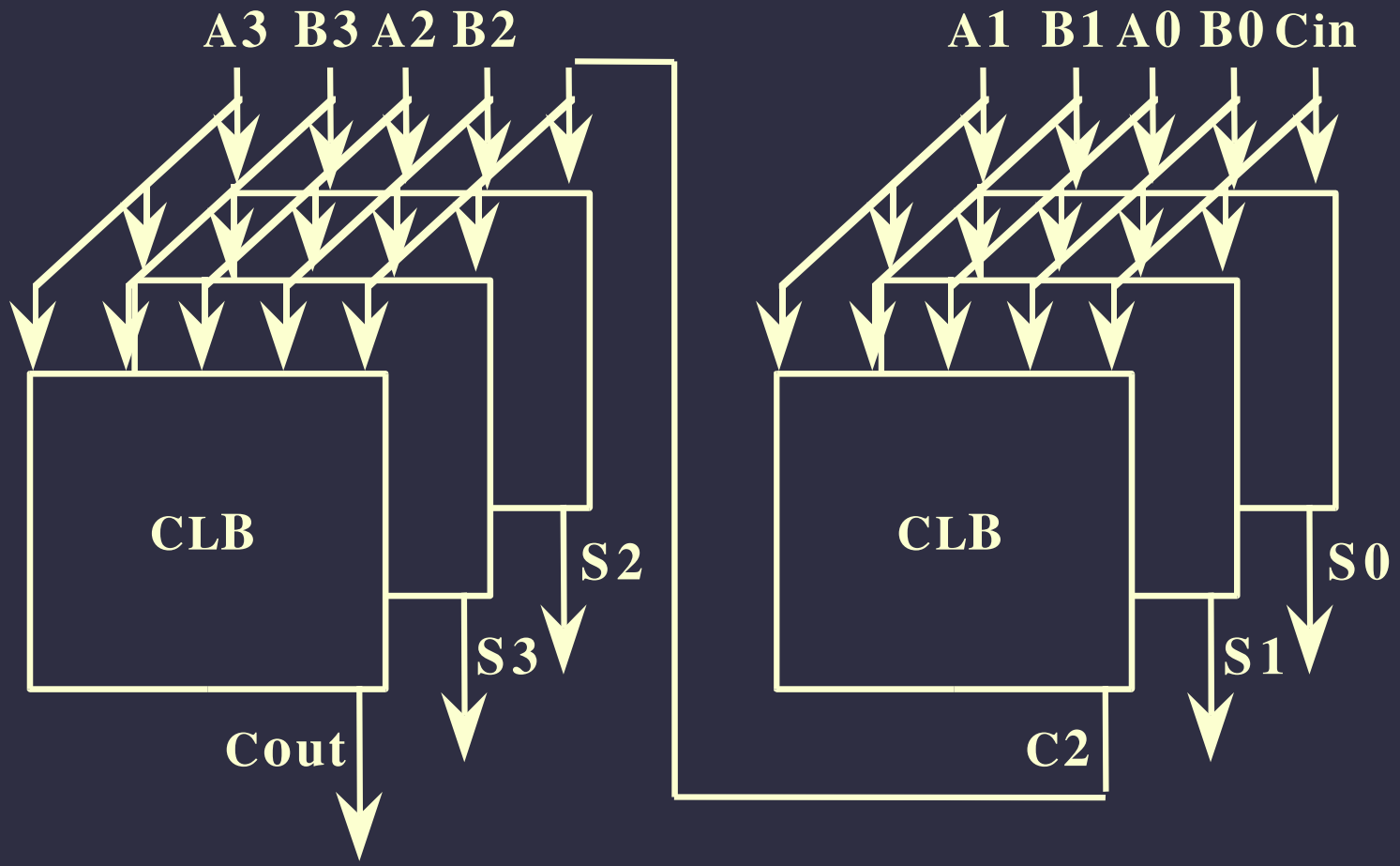
4-bit adder CLB cost example

Full adder, 4 CLB delays to final carry out (CO), 4 CLBs



4-bit adder CLB cost example

Composition from 2 2-bit adders give 2 CLB delay, 6 CLB cost



Xilinx interconnect

- Short direct connections
- Global long lines
- Horizontal/vertical long lines
- Switching matrix connections

Xilinx interconnect

- Hierarchical routing organization
- Some designs are constrained by routing resources
- Can use logic CLBs to control routing
- Substantial communication power consumption

Xilinx interconnect

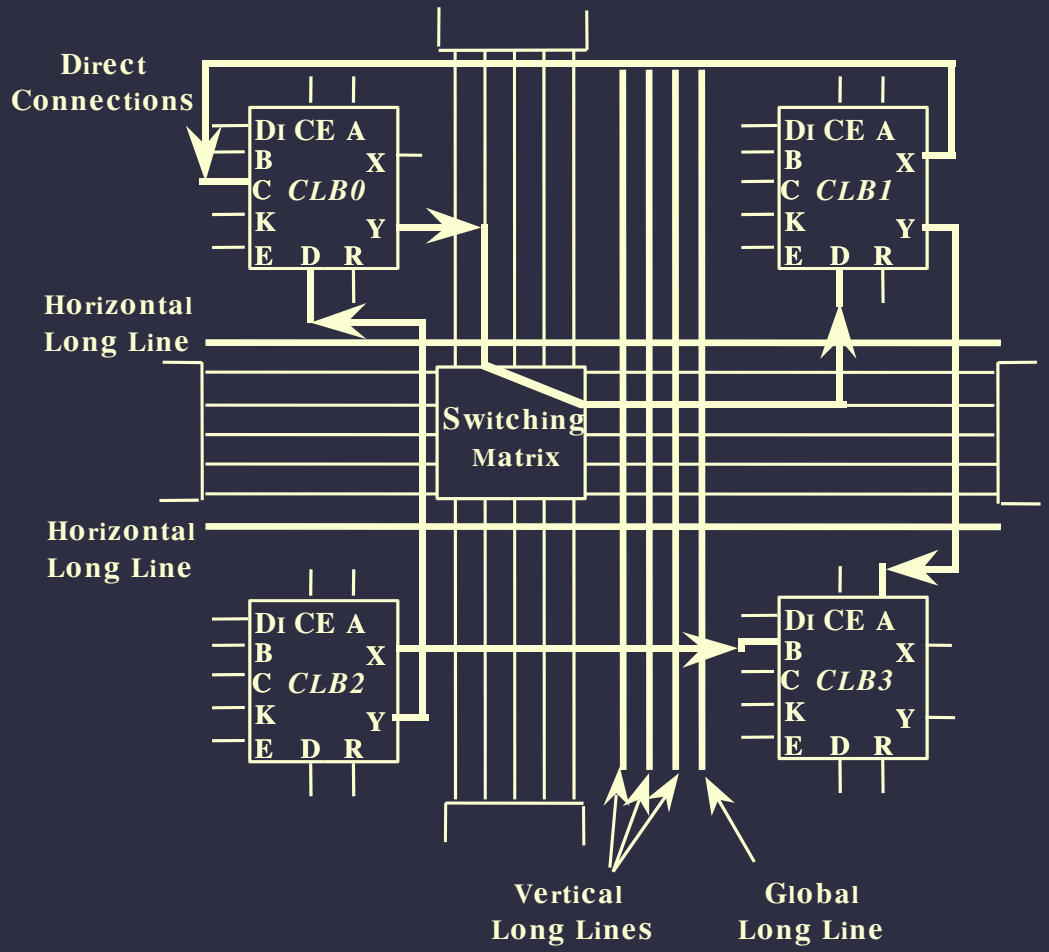
Interconnect

Direct Connections

Global Long Line

Horizontal/Vertical Long Lines

Switching Matrix Connections



Example Xilinx parts

Parameter	XC4024	XC3195	XC2018
Number of FFs	2,560	1,320	174
Number of IOs	256	176	74
Number of logic inputs per CLB	9	5	4
Function generators per CLB	3	2	2
Fast carry logic	yes	no	no
Number of logic outputs per CLB	4	2	2
RAM bits	32,768	0	0

Dynamic reconfiguration

- Serial configuration slow
  - Parallelize
- Full reconfiguration slow
  - Partial reconfiguration
- Reconfiguration slow
  - Use configuration cache

FPGA application examples

- Prototyping
- Constant coefficient multiplication
- Direct HW implementation of problem instance, e.g.,
  - 3SAT
  - Design rule checking (DRC)

Prototype designs

- Discrete packages
  - Slow
  - Error-prone
- Custom layout requires circuit fabrication
  - Slow
  - Expensive for small runs
  - Can't be changed

Programmable devices in prototyping

- Multiplexers (MUXs) and demultiplexers (DMUXs)
  - Wiring them up is tedious and error-prone
- Programmable array logic (PAL) and programmable logic array (PLA)
  - Fuses blown, write-once
- Generic array logic (GAL)
  - Electrically reprogrammable

Programmable devices in prototyping

- Programmable read-only memories (PROMs)
  - Inefficient for implementing random logic
  - Write-once
- Erasable programmable read-only memories (EPROMs)
  - Can be erased
  - Erasure slow (UV)
  - Expensive package window

Programmable devices in prototyping

- Electrically erasable programmable read-only memories (EEPROMs)
  - Erasure fast
  - Packaging less expensive
  - Potential for in-circuit erasure
- Field-programmable gate arrays (FPGAs) are ideal
  - If market size small, ship FPGAs
- In-circuit programming practical

Section outline

1. Implementation technologies
   - Review of MUX composition
   - Steering logic
   - ROMs
   - FPGAs
   - Transformations for CMOS

DeMorgan's Law for CMOS

$$\overline{(A+B)} = \overline{A}\ \overline{B}$$
$$\overline{(AB)} = \overline{A} + \overline{B}$$
$$A + B = \overline{\overline{A}\ \overline{B}}$$
$$AB = \overline{\overline{A} + \overline{B}}$$

DeMorgan's Law for CMOS

- OR is the same as NAND with complemented inputs
- AND is the same as NOR with complemented inputs
- NAND is the same as OR with complemented inputs
- NOR is the same as AND with complemented inputs

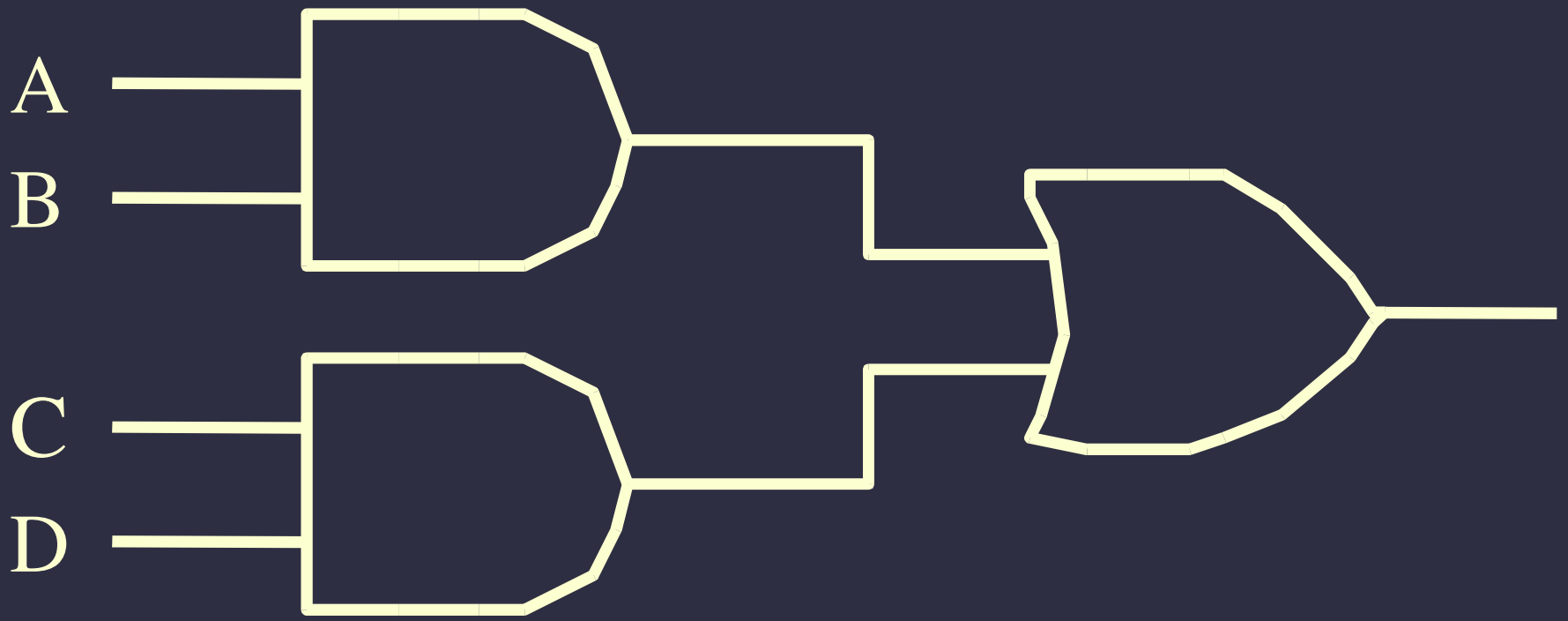
DeMorgan's Law for OR/NAND

A	$\overline{A}$	B	$\overline{B}$	A + B	$\overline{(\overline{A}\ \overline{B})}$	$\overline{A} + \overline{B}$	$\overline{(AB)}$
0	1	0	1	0	0	1	1
0	1	1	0	1	1	1	1
1	0	0	1	1	1	1	1
1	0	1	0	1	1	0	0

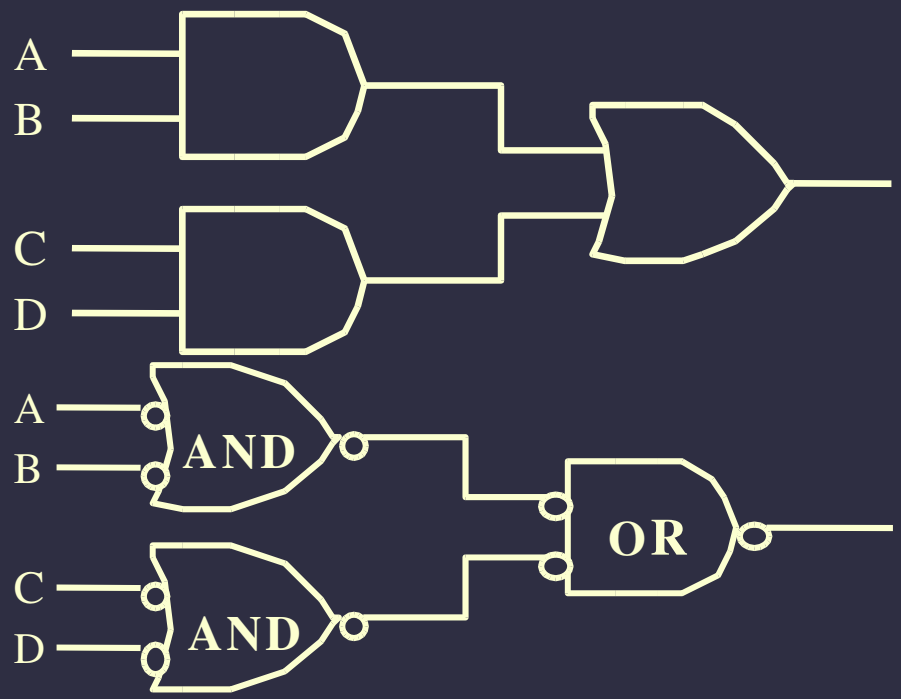
DeMorgan's Law for AND/NOR

A	$\overline{A}$	B	$\overline{B}$	A B	$\overline{(\overline{A}+\overline{B})}$	$\overline{A}$ $\overline{B}$	$\overline{(A+B)}$
0	1	0	1	0	0	1	1
0	1	1	0	0	0	0	0
1	0	0	1	0	0	0	0
1	0	1	0	1	1	0	0

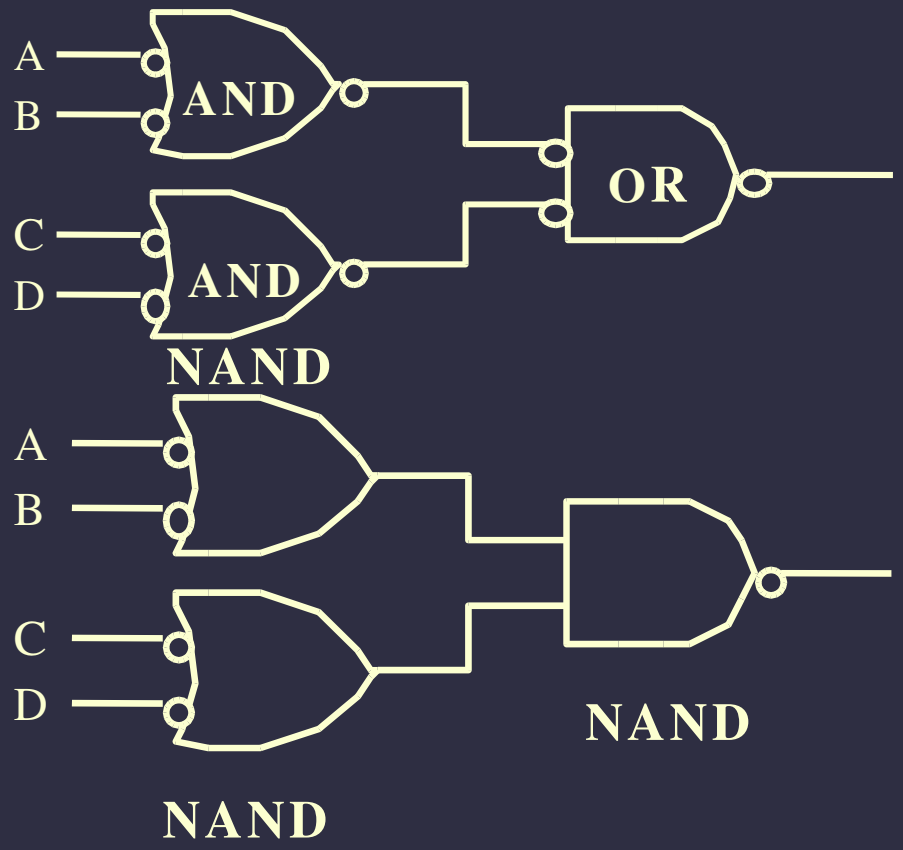
AND/OR → NAND/NOR



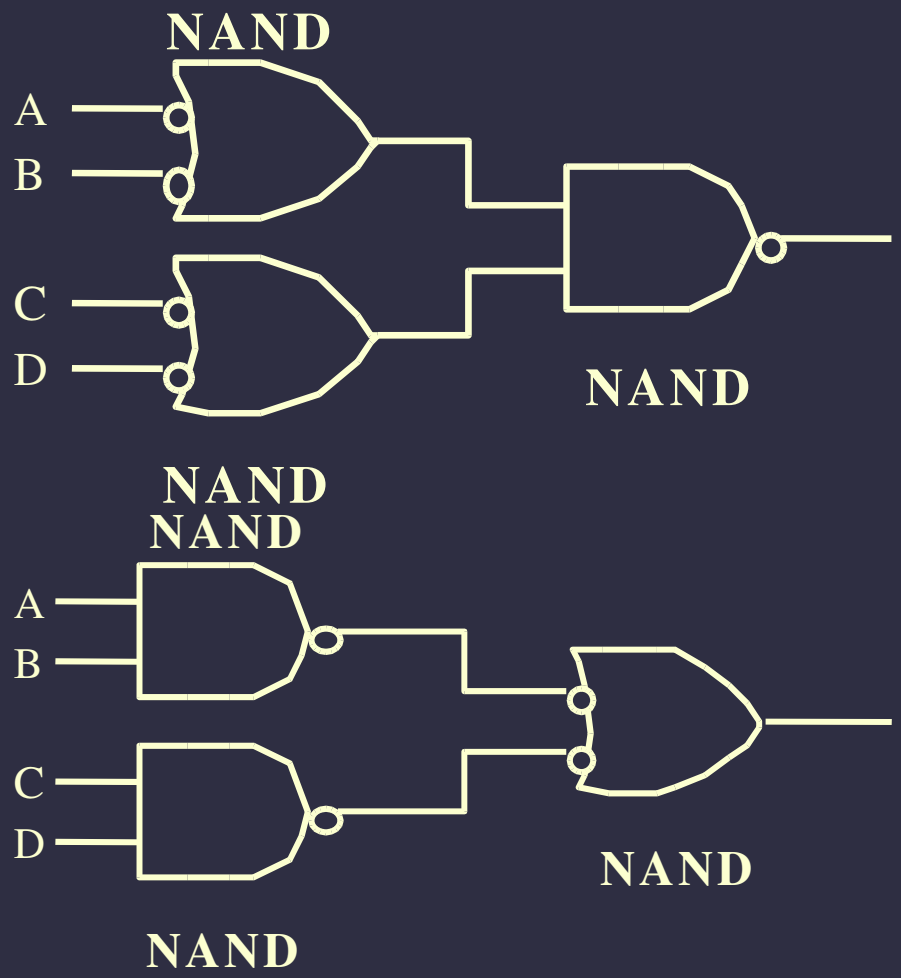
AND/OR → NAND/NOR



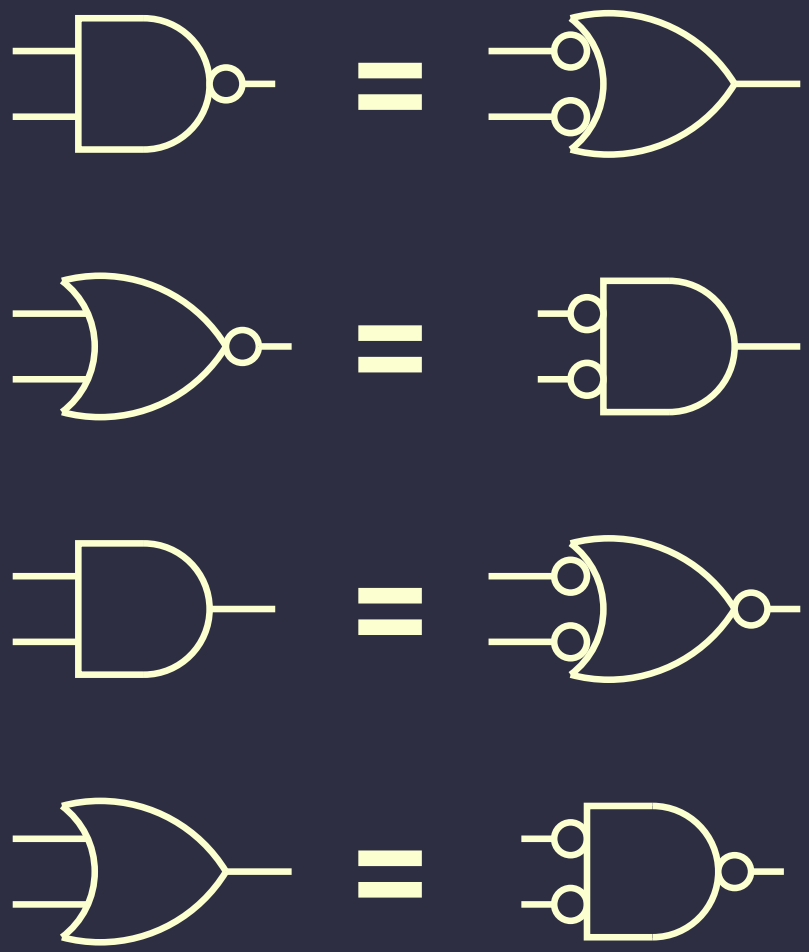
AND/OR → NAND/NOR



AND/OR → NAND/NOR



AND/OR/NOT network to NAND/NOR



Outline

1. Implementation technologies
2. Homework

Homework

- Review for midterm exam on Thursday
- Will post solutions to homework tonight
- Responsible for all reading, assignments, labs

Review

- Two-level transformations and minimization
- Multi-level minimization
- Design with various implementation technologies