# Digital Systems


2. Fundamentals of Digital Circuits
    - 2.12 Representation of Boolean Function
    - 2.13 NAND and NOR Logic
    - 2.14 Transmission Gate
    - 2.15 Fan-In and Fan-Out
    - 2.16 Standard Combinatorial Circuit
    - 2.17 Realization of Combinatorial Circuits using Memories

Theorem:

Every boolean function can be represented by only using inverters, AND or OR gates.

Proof:

The canonical disjunctive normal form is one such respresentation.

Theorem :

Every boolean function can be represented by only using inverters, AND and OR gates with only two inputs.

Theorem :

Every boolean function can be represented by only using inverters, and AND gates with only two inputs.

Theorem :

Every boolean function can be represented by only using inverters and OR gates with only two inputs.

Theorem :

Every boolean function can be represented by only using NAND gates with only two inputs.

Every boolean function in disjunctive Normalform can be converted to pure NAND logic very simply:

By replacing all AND and OR gates by NAND gates.

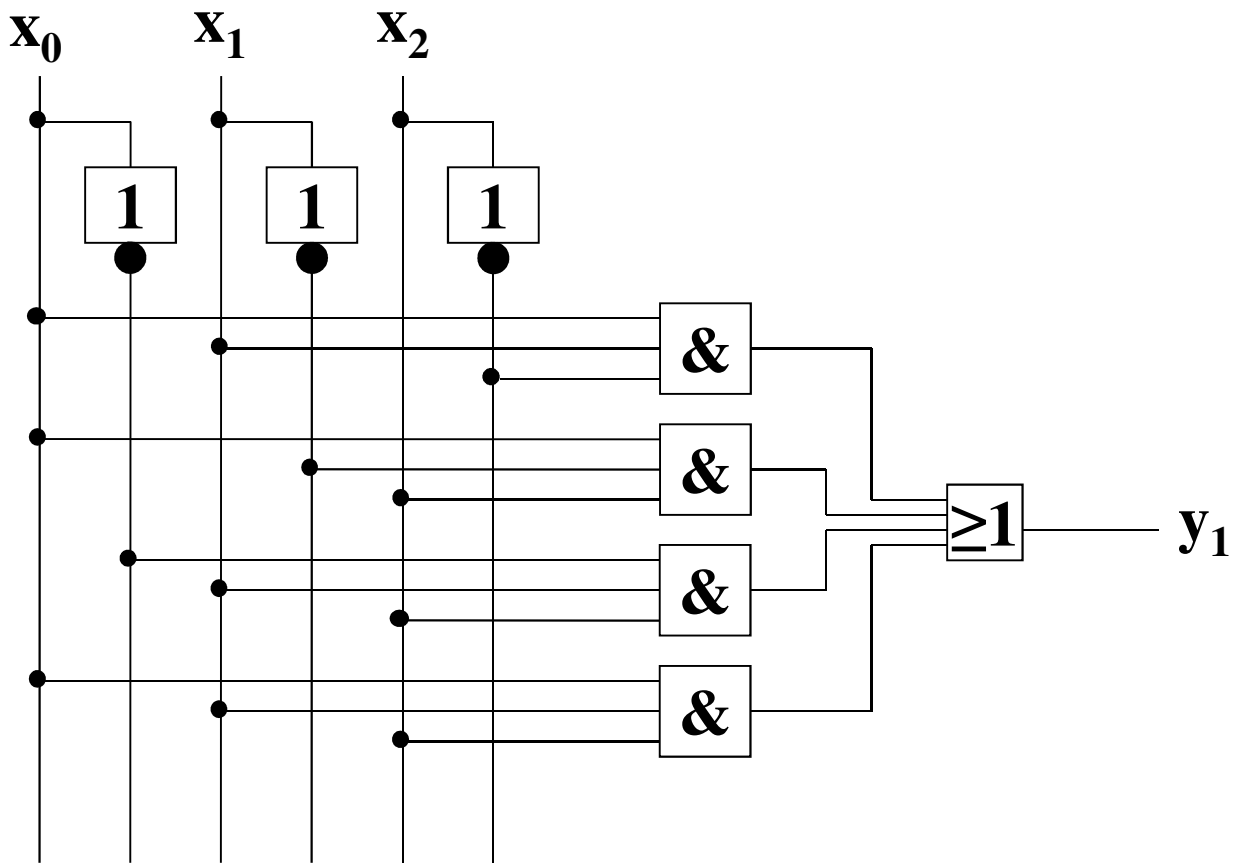
Example:

$$x_0 x_1 \overline{x_2} + x_0 \overline{x_1} x_2 + \overline{x_0} x_1 x_2 + x_0 x_1 x_2 =$$

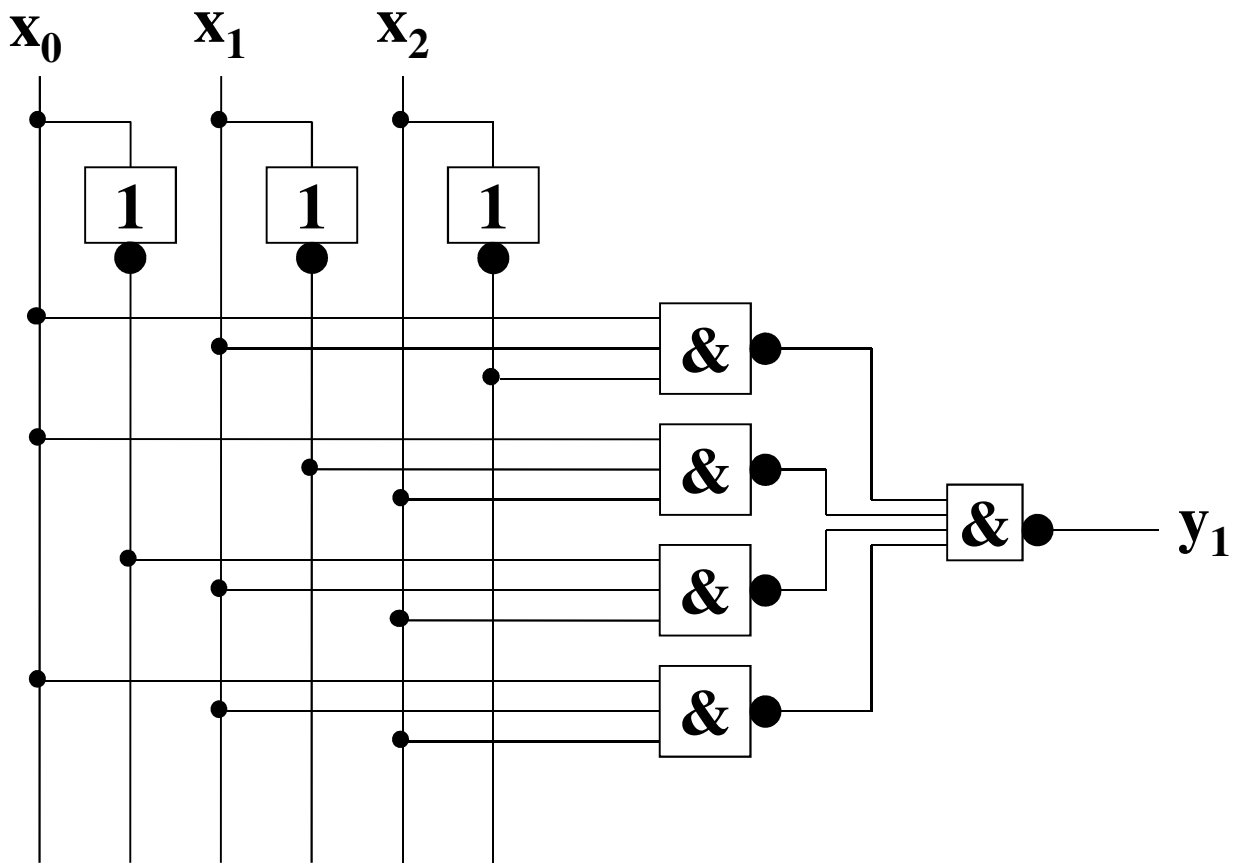
$$\overline{\overline{x_0 x_1 \overline{x_2} + x_0 \overline{x_1} x_2 + \overline{x_0} x_1 x_2 + x_0 x_1 x_2}} =$$

$$\overline{\overline{x_0 x_1 \overline{x_2}} \cdot \overline{x_0 \overline{x_1} x_2} \cdot \overline{\overline{x_0} x_1 x_2} \cdot \overline{x_0 x_1 x_2}}$$

# An Example of a Function in DNF:



The same Function only using NAND Logic:



Every boolean function in conjunctive Normalform can be converted to NOR logic very simply:

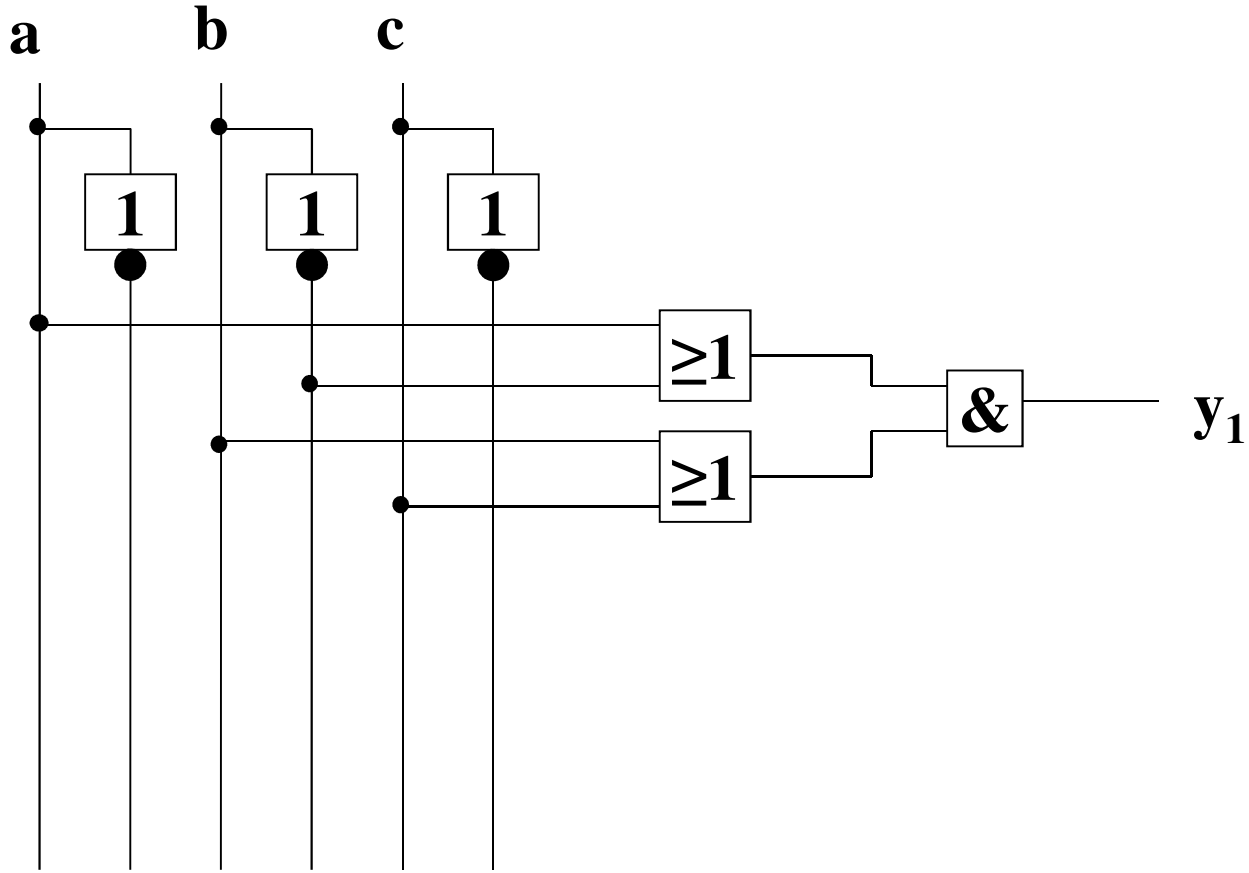
By replacing all AND and OR gates by NOR gates.

Example:

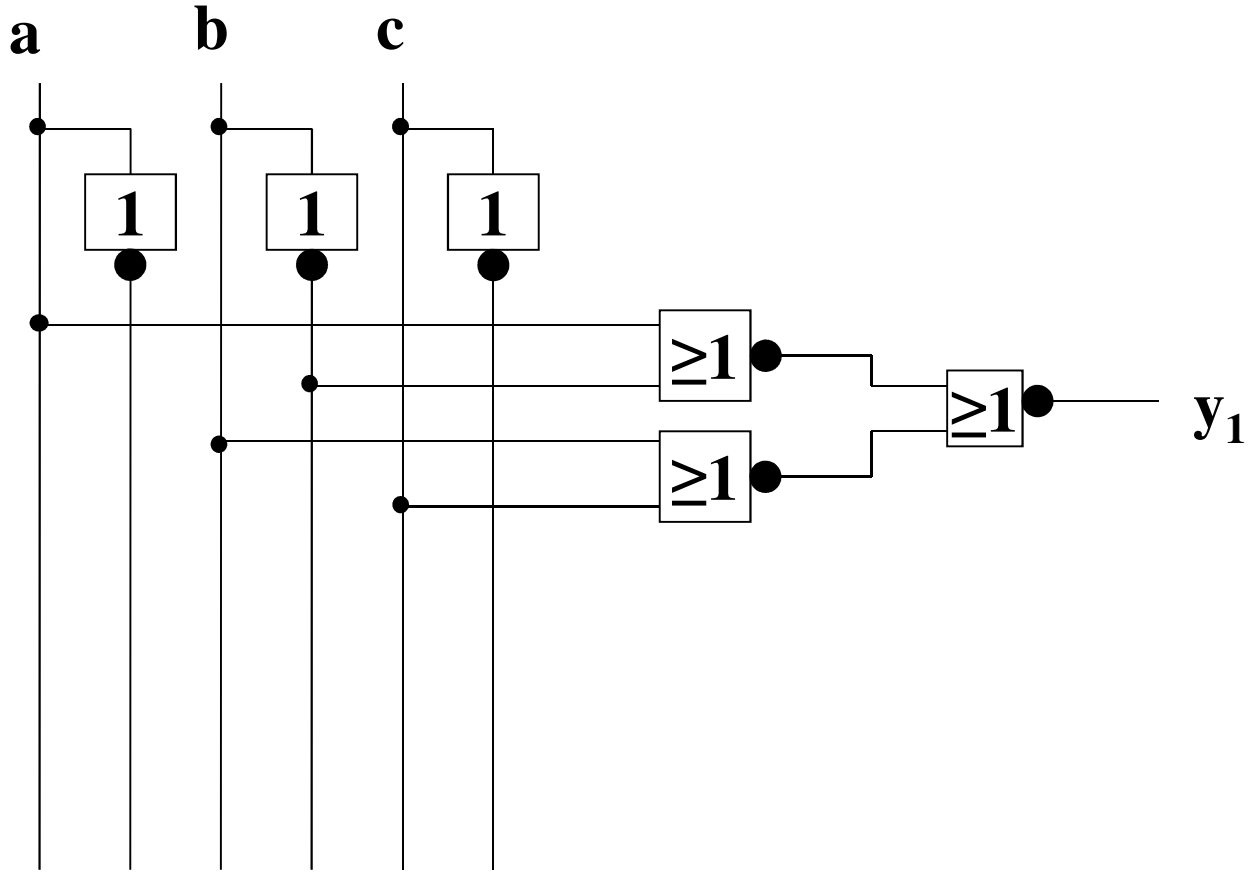
$$y_1 = (a + \overline{b}) \cdot (b + c) = \overline{\overline{(a + \overline{b}) \cdot (b + c)}}$$

$$= \overline{\overline{(a + \overline{b})}} + \overline{\overline{(b + c)}}$$

# An Example of a Circuit Diagram of a Function in CNF:



The same Function only using NOR Logic:

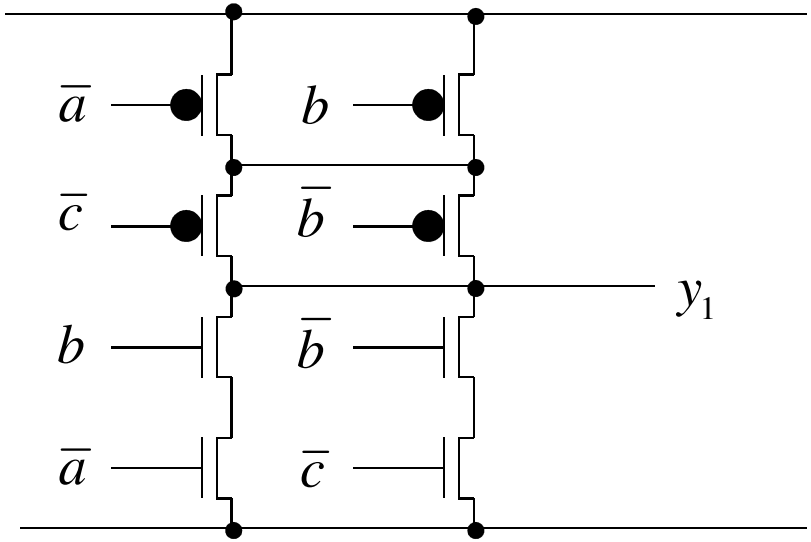


Example: The same Function as CMOS Gate

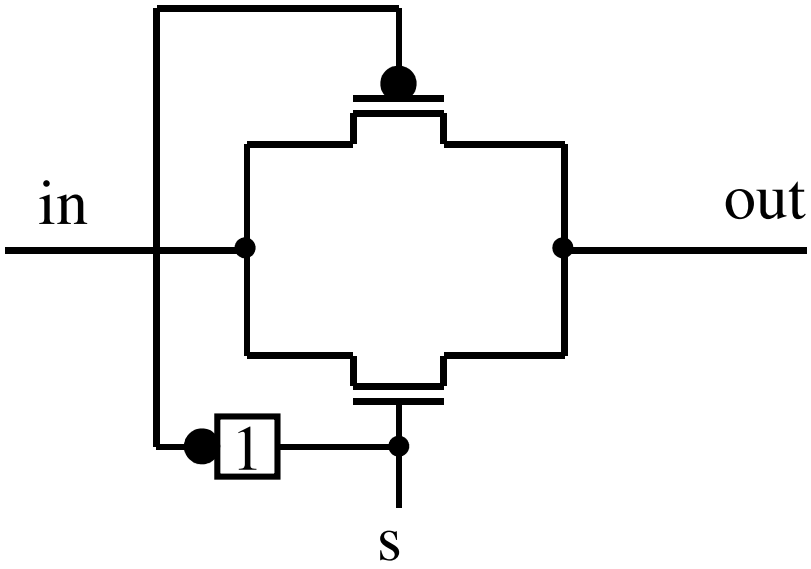
$$y_1 = (a + \overline{b}) \cdot (b + c) = \overline{\overline{(a + \overline{b}) \cdot (b + c)}}$$

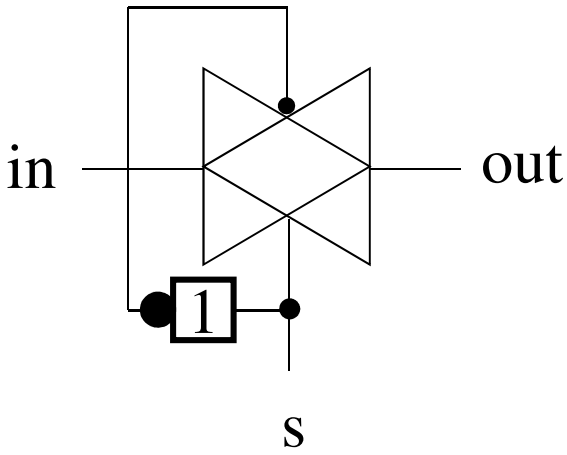
$$= \overline{\overline{(a + \overline{b})} + \overline{(b + c)}}$$

$$= \overline{(\overline{a} \cdot b) + (\overline{b} \cdot \overline{c})}$$



Transmission gates have the functionality of gates: They pass signals from the data input to the output if the steering input (s=steering input, enable) is 1. The interesting part with transmission gates is the fact that their output is not connected if s=0. That means in that case the output is neither 0 nor 1 but at high impedance state. The output of other gates should never be connected to each other, because undefined signals could be produced but the outputs of transmission gates can be wired together if it is ensured that always exactly one signal has a logic 1.



## 2.15 Fan-In and Fan-Out:

Fan-In: The number of inputs of a gate determines the number of transistors in row, that the signals have to pass to drive the output.

Fan-Out: The number of inputs of the same size that has to be charged or discharged determines the capacity that the gate has to charged or discharged while switching.

The resistance and the capacitance forms a low pass filter describing the switching behavior of the gate. The switching process can be described by the exponential curve of a low pass. R*C is the time that the charging and discharging process needs. Hence this time is proportional to R as well as to C. That’s why fan in and fan out has to be limited in real circuits.

Charging Process (Change from 0 to 1)

$$U_a = U_e - U_e \cdot e^{-\frac{t}{RC}} = U_e \cdot (1 - e^{-\frac{t}{RC}})$$

$$\frac{U_a}{U_e} = 1 - e^{-\frac{t}{RC}}$$

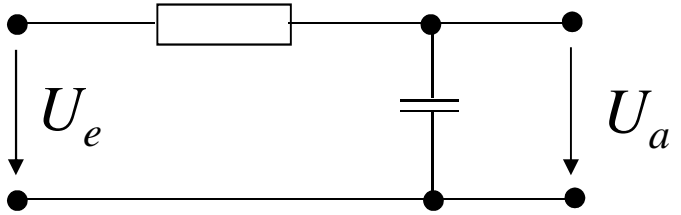
$$1 - \frac{U_a}{U_e} = e^{-\frac{t}{RC}}$$

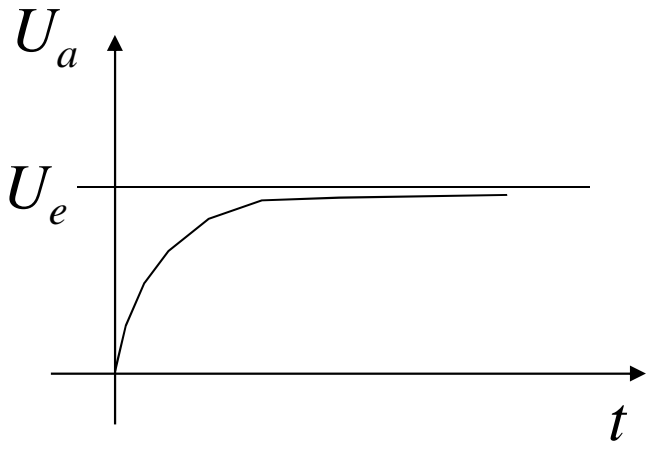
$$Assumption: \frac{U_a}{U_e} = 0{,}9$$

$$\ln(1 - 0{,}9) = -\frac{t}{RC}$$

$$-2{,}3RC \approx -t$$

$$t \approx 2{,}3RC$$



Discharging Process (Change from 1 to 0)

$$U_a = U_e \cdot e^{-\frac{t}{RC}}$$

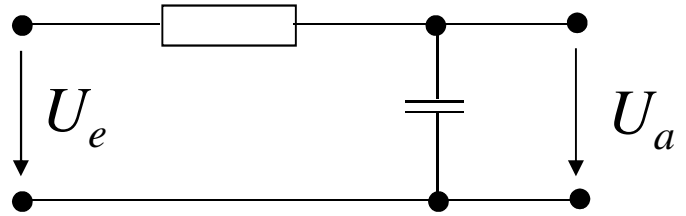
$$\frac{U_a}{U_e} = e^{-\frac{t}{RC}}$$

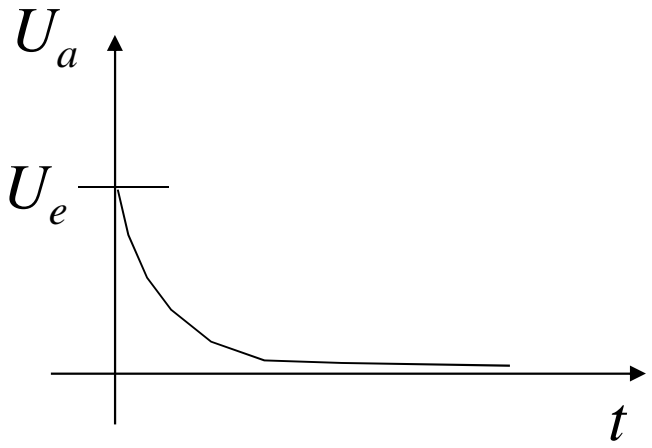
$$Assumption: \frac{U_a}{U_e} = 0{,}1$$

$$\ln 0{,}1 = -\frac{t}{RC}$$

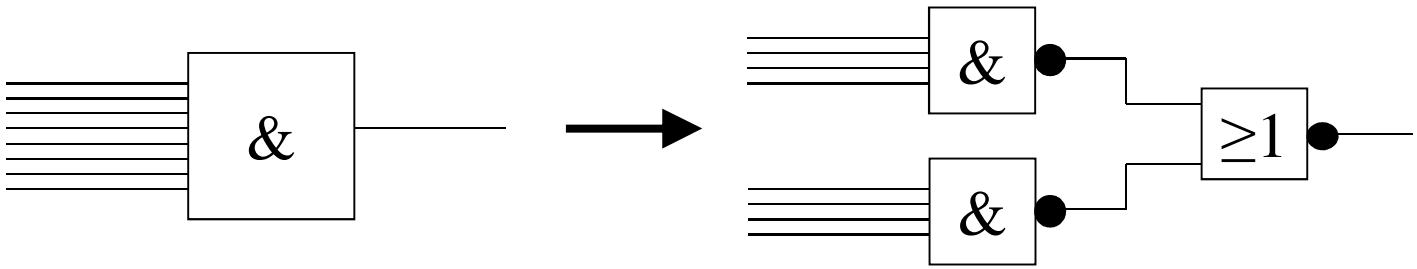
$$-2{,}3RC \approx -t$$

$$t \approx 2{,}3RC$$



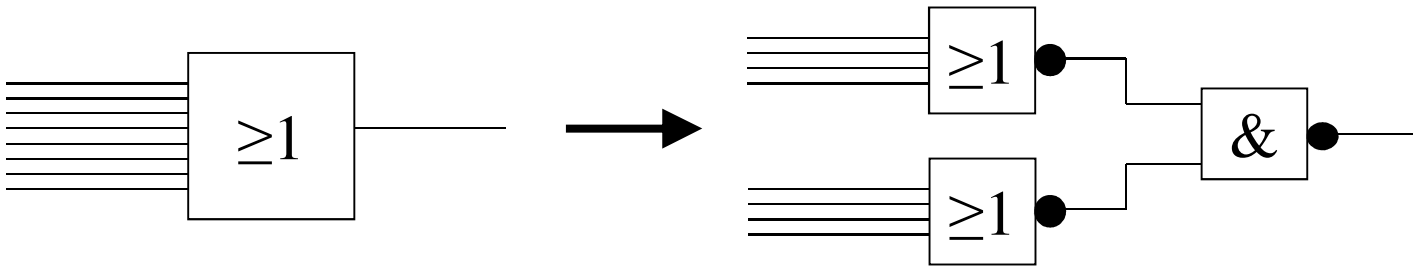



What can we do if we have gates with a too large fan-in?



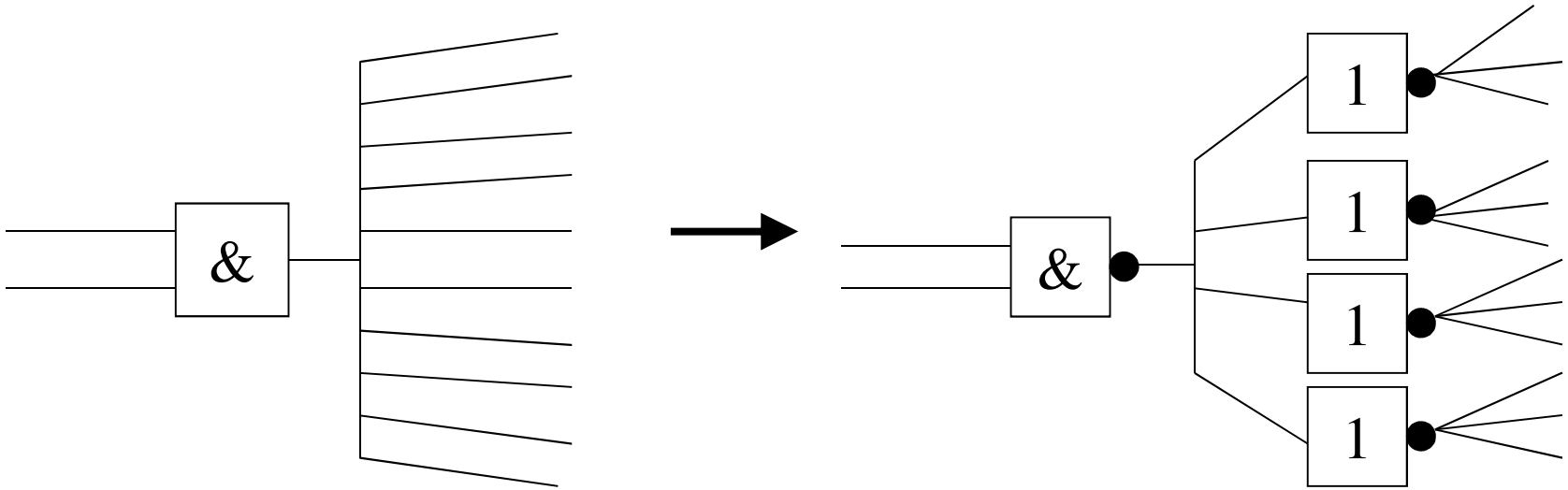

$$a \cdot b \cdot c \cdot d = \overline{\overline{a \cdot b \cdot c \cdot d}} = \overline{\overline{a \cdot b} + \overline{c \cdot d}}$$

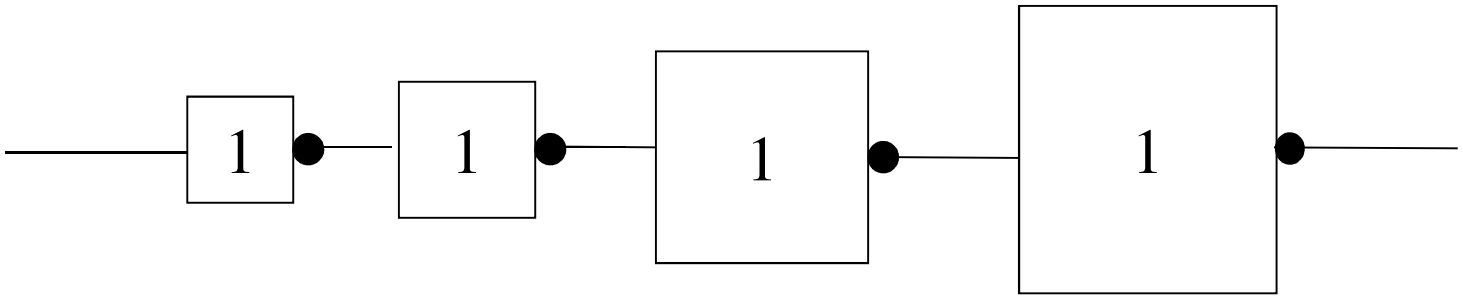
2. Example: OR gate with too large Fan-in



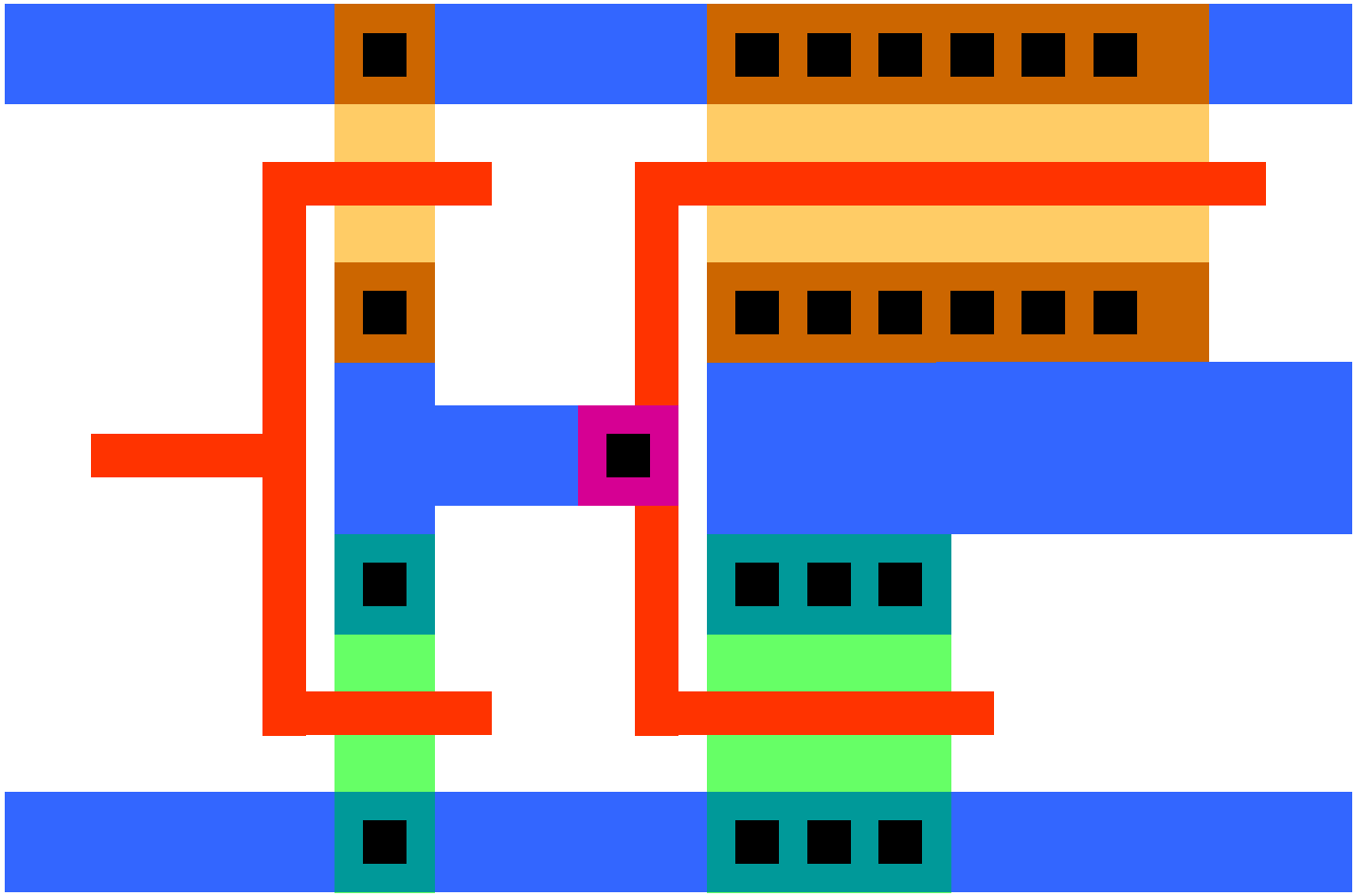

$$a+b+c+d=\overline{\overline{a+b+c+d}}=\overline{\overline{a+b}\cdot\overline{c+d}}$$

What can we do if we have gates with a too large fan-out?



# Top View of an Inverter Pair as Driver on a Chip



## 2.16 Standard Combinatorial Circuit

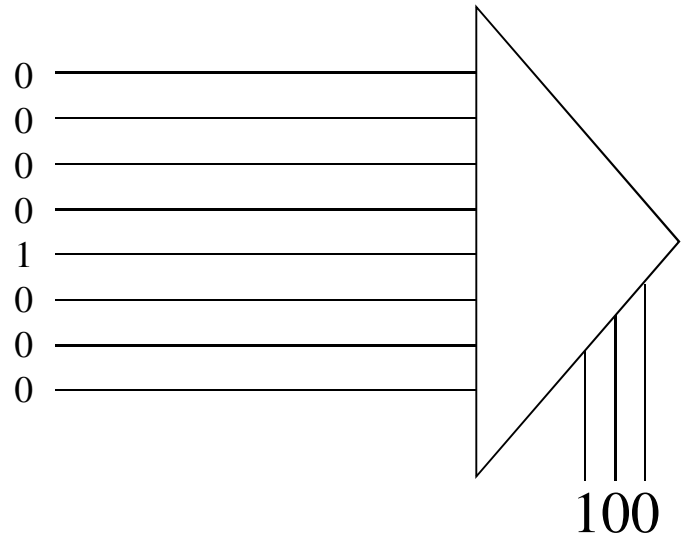
Up till now, we have created ourselves a tool box to create circuits out of function descriptions. This is like our basics and what will follow next are many operations like division, square root, power of, logarithms, integrals, etc... operations using this basics. This is the stage that we are at right now.

Now, we learn the basic circuits which are used in common hardware systems on and on. It is our goal that we will develop a feeling for what an adder does, a multiplexer, a counter etc.

The Coder

A coder is a combinatorial circuit that gets a message in a defined code as an input and changes this code to a different one and gives this as an output. Remember the binary Aiken coder for the decimal numbers.

Very often one would like one of the codes to be a 1-out-of-N code. This is a coding of numbers from 0 to N-1 where exactly one bit is a at logic 1 and all the others 0. A 1-out-of-N code has exactly N different code words. How to build a coder that converts a 1-out-of-N code into a binary code?

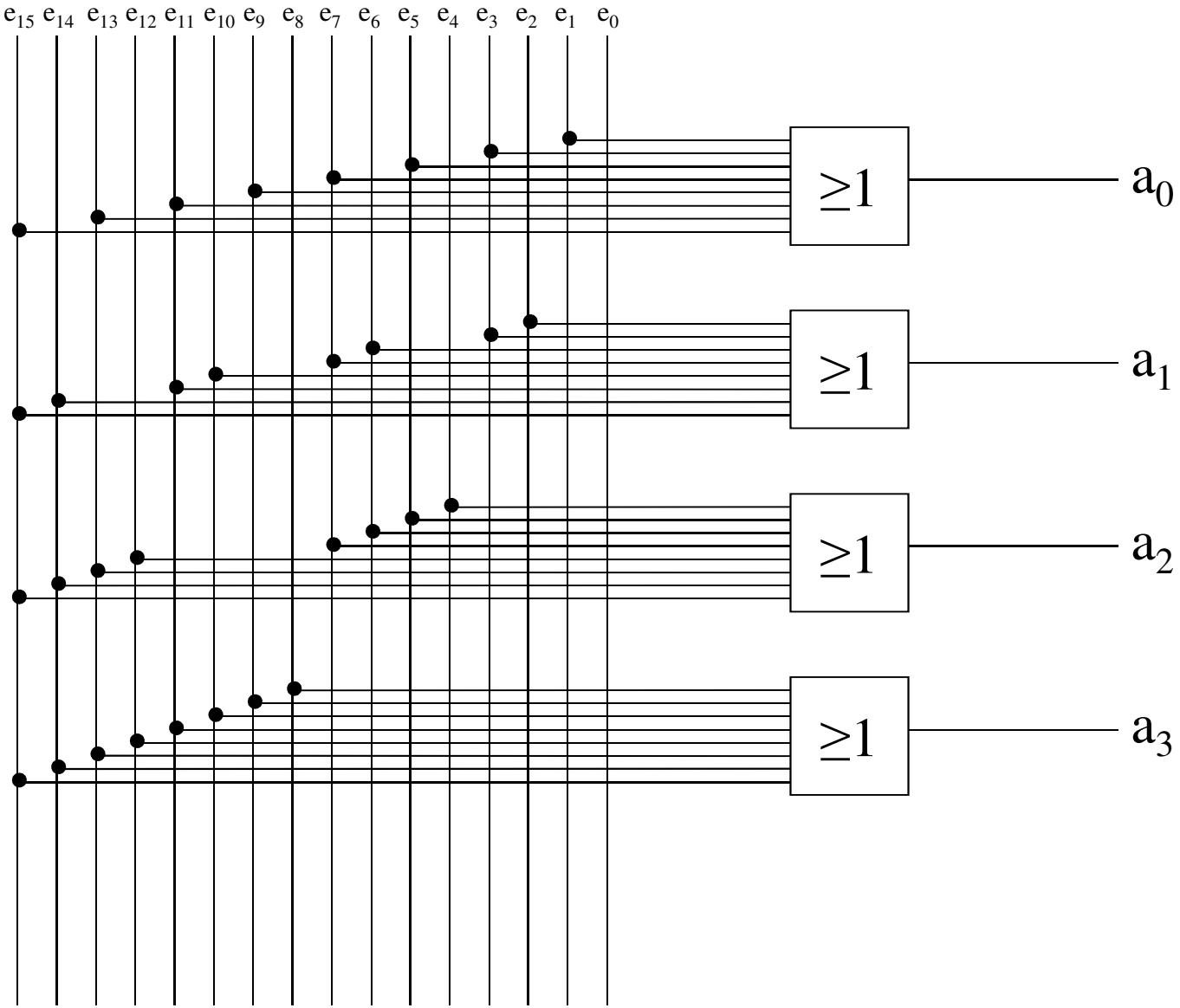


Relevant Part of the Truth Table for a 1-out-of-16-in-Binary Coder

| $e_{15}$ | $e_{14}$ | $e_{13}$ | $e_{12}$ | $e_{11}$ | $e_{10}$ | $e_9$ | $e_8$ | $e_7$ | $e_6$ | $e_5$ | $e_4$ | $e_3$ | $e_2$ | $e_1$ | $e_0$ | $a_3$ | $a_2$ | $a_1$ | $a_0$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

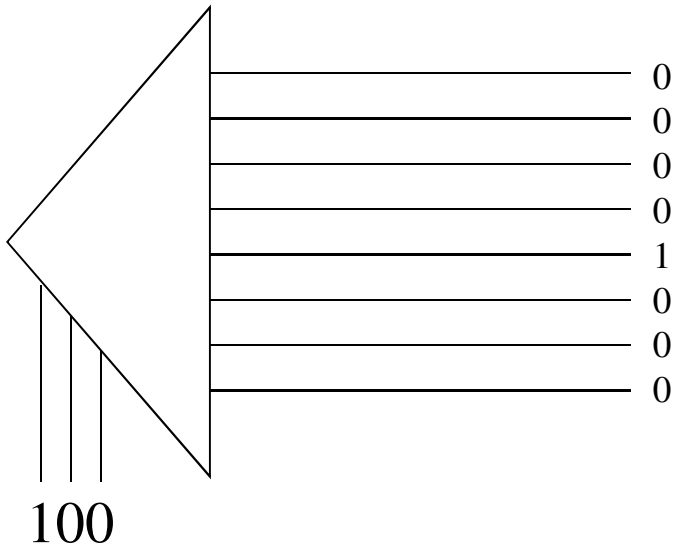
All other values are X (dont' care)

# 1-out-of-16-in-Binary Coder



The Decoder

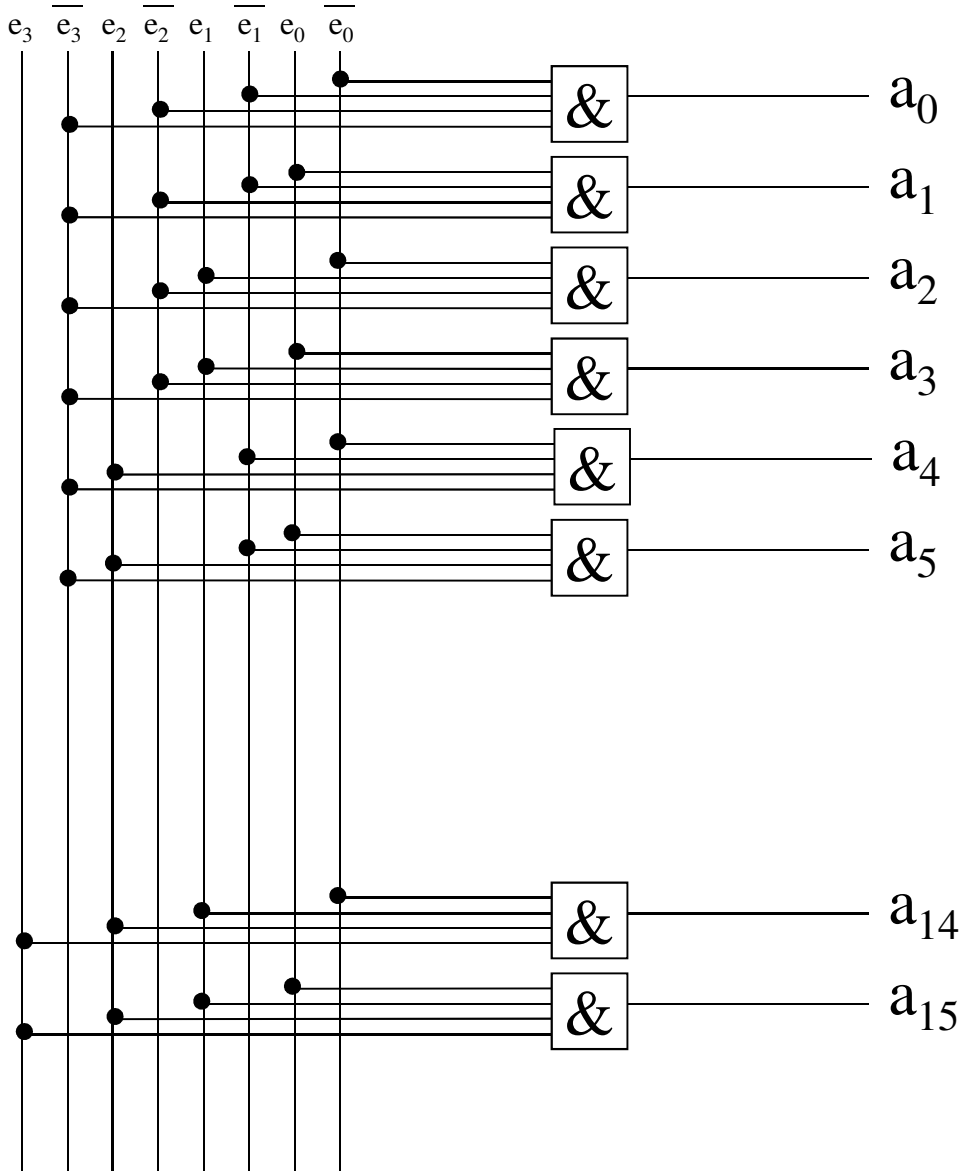
Now is the opposite whereby the input is a binary code and the output is 1-out-of-N code.



# Truth table for binary-in-1-out-of-16-Decoder

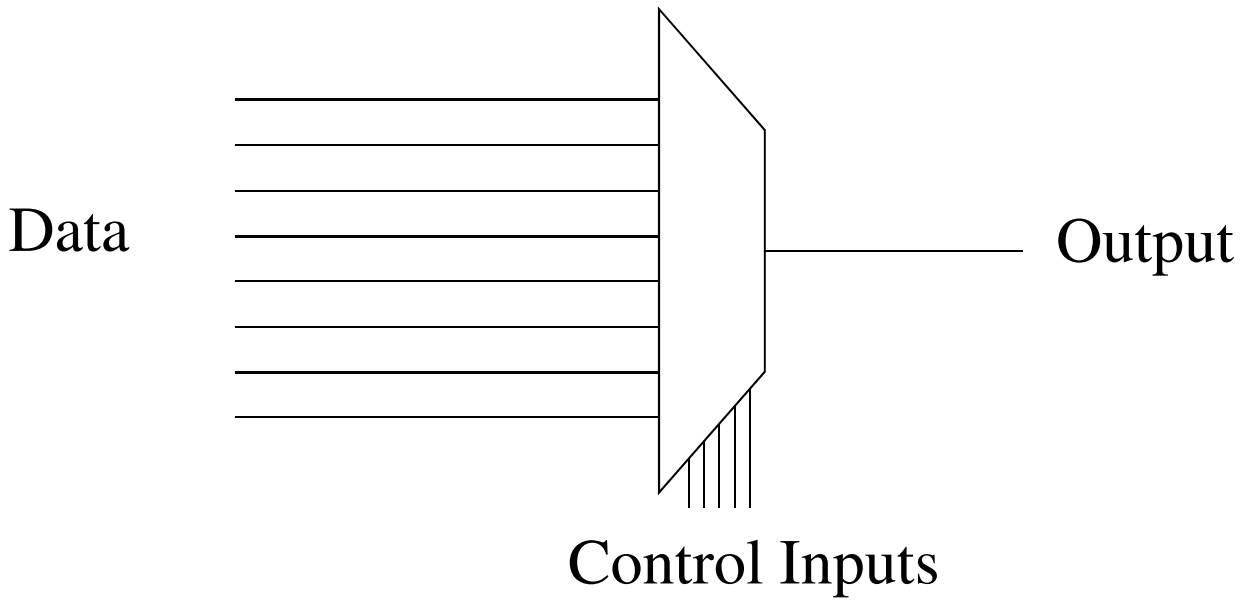
| $e_3$ | $e_2$ | $e_1$ | $e_0$ | $a_{15}$ | $a_{14}$ | $a_{13}$ | $a_{12}$ | $a_{11}$ | $a_{10}$ | $a_9$ | $a_8$ | $a_7$ | $a_6$ | $a_5$ | $a_4$ | $a_3$ | $a_2$ | $a_1$ | $a_0$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# Binary-in-1-out-of-16-Decoder



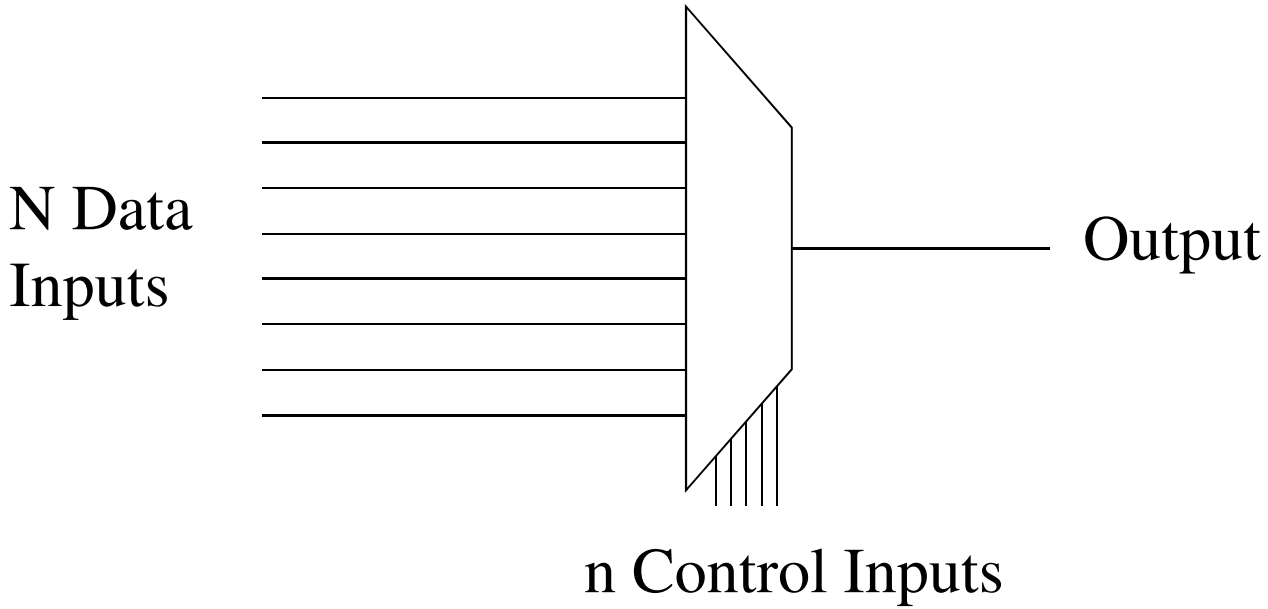
The Multiplexer

A mutliplexer is a combinatorial circuit that chooses 1 out of several inputs and sends that signal unchanged as an output. One can think of a multiplexer like a switch. Multiplexers have data inputs and control inputs. Depending on the signals of the control input, the correct data inputs will be chosen.

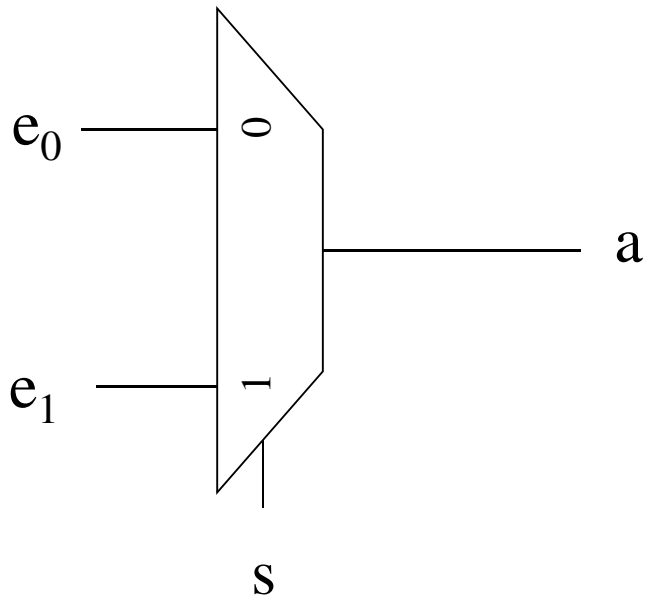


Let the data inputs be represented with $e_i$ and, i=0,...,N-1, the control inputs with $s_i$, i=0,..,n-1, the output with a. Then, N has to be $N \leq 2^n$. The function of the multiplexer is described as

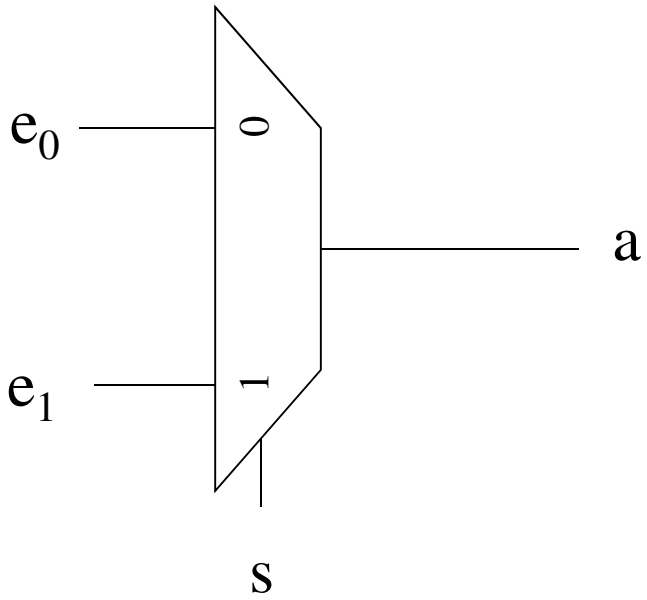
$a = e_i$, if $(i)_{10}=(s_{n-1}s_{n-2}...s_0)_2$



Example: A 2-to-1-Multiplexer. It has two data inputs $e_0$ and $e_1$ and one control input s. When the control input is s=0, then $a=e_0$ and when s=1, then $a=e_1$.




| s | $e_1$ | $e_0$ | a |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |




| s | $e_1$ | $e_0$ | a |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

a

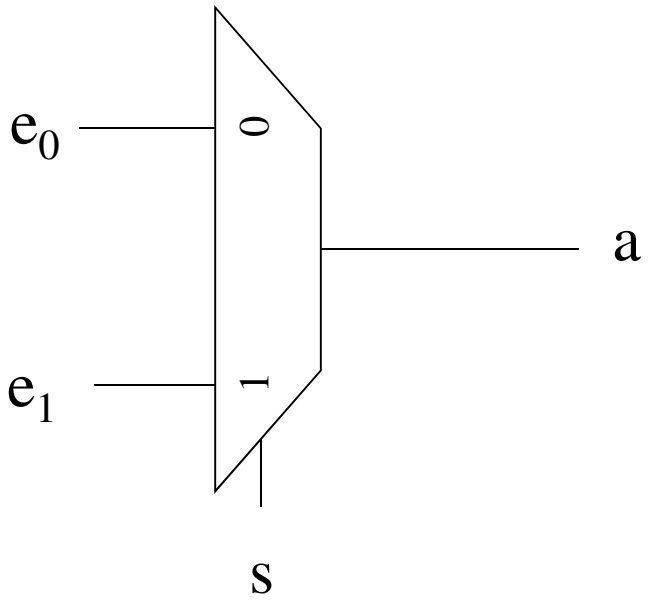
| s \ $e_0$ | | | |
|---|---|---|---|
| | 0 | 1 | 1 | 0 |
| | 1 | 1 | 0 | 0 |

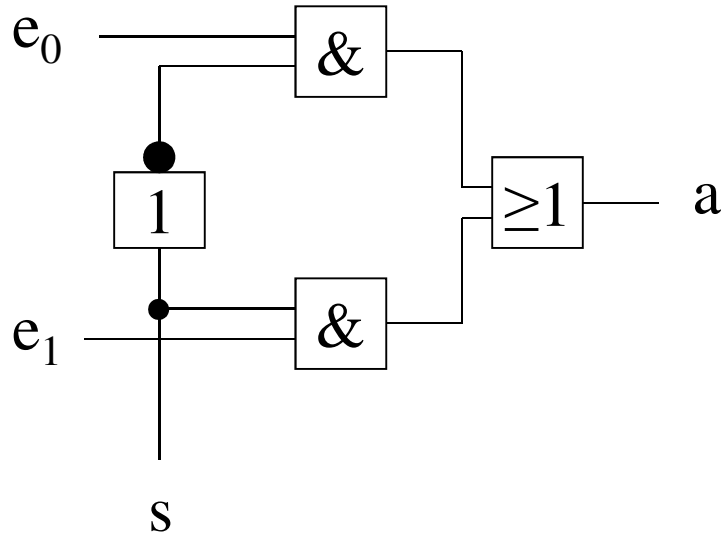
$e_1$

$$a = \bar{s}e_0 + se_1$$

Realization:

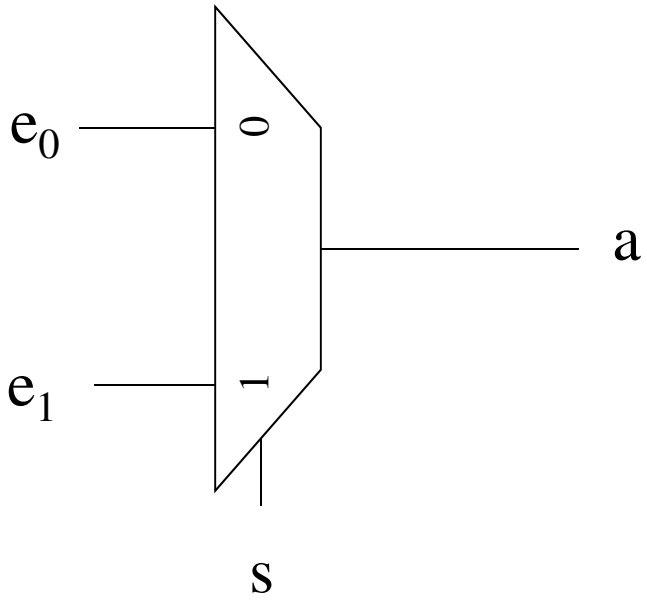
$$a = \bar{s}e_0 + se_1$$

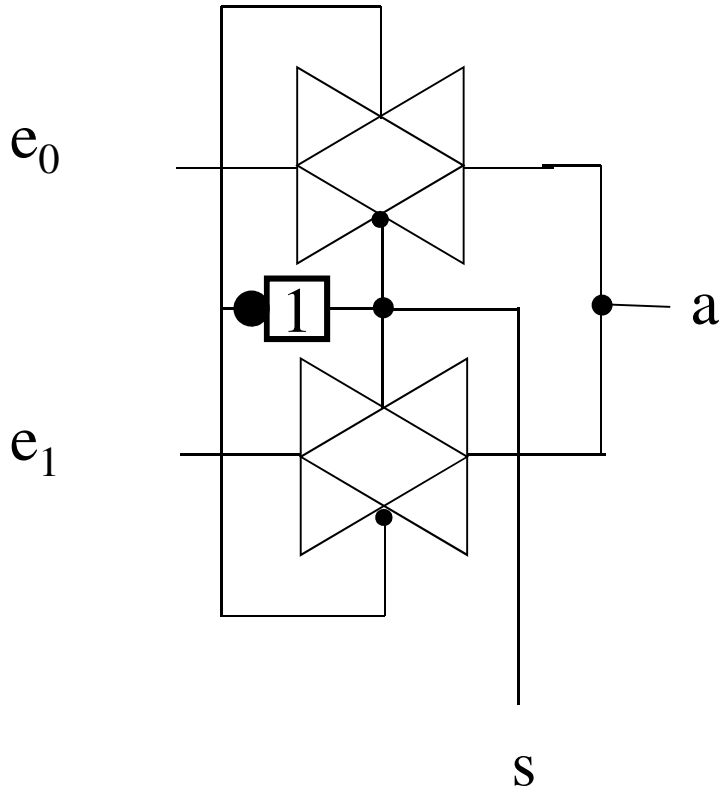


2. Realization Possibility:

$$a = \bar{s}e_0 + se_1$$



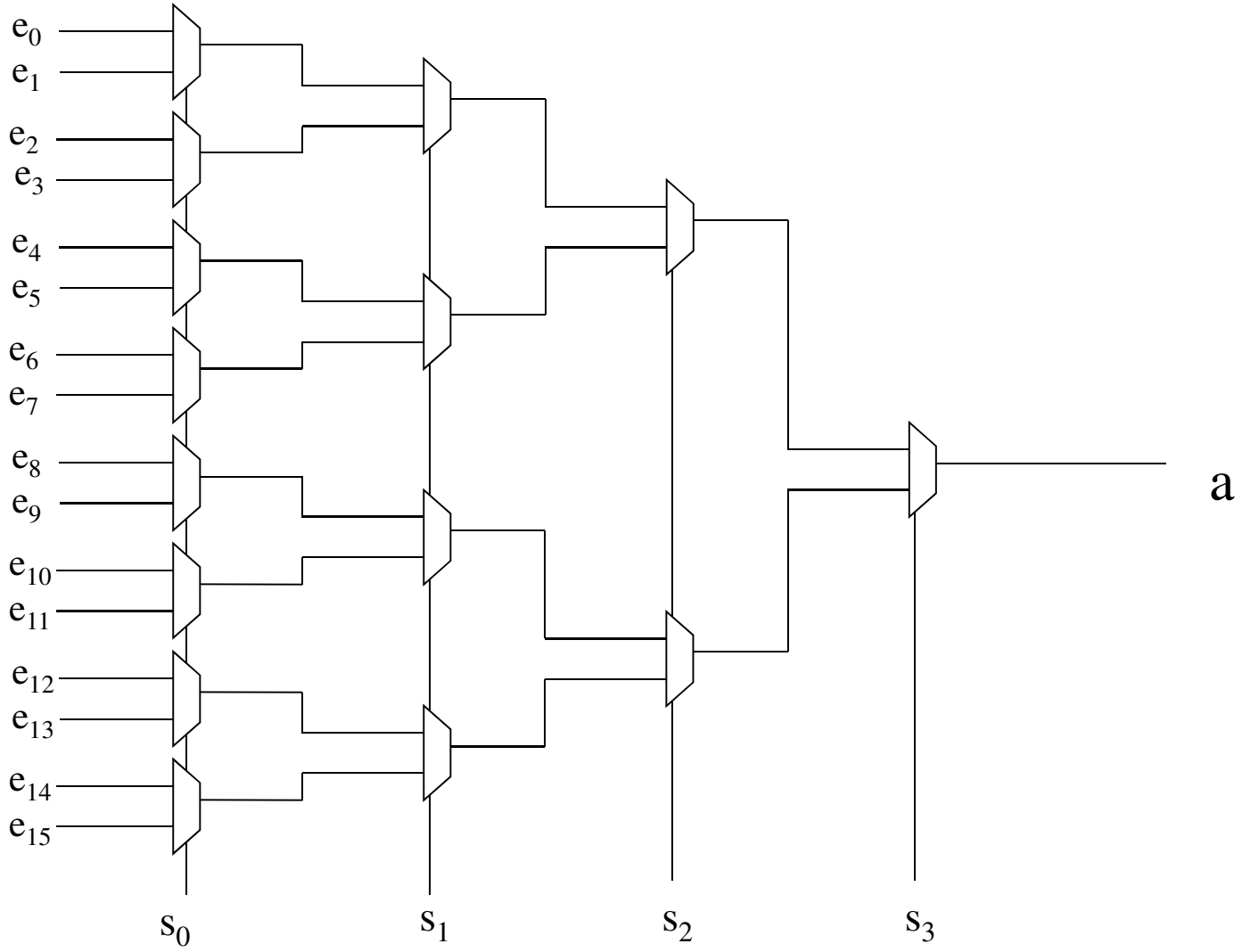



# Truth Table for a 16-to-1-Multiplexer

| $s_3$ | $s_2$ | $s_1$ | $s_0$ | a |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | $e_0$ |
| 0 | 0 | 0 | 1 | $e_1$ |
| 0 | 0 | 1 | 0 | $e_2$ |
| 0 | 0 | 1 | 1 | $e_3$ |
| 0 | 1 | 0 | 0 | $e_4$ |
| 0 | 1 | 0 | 1 | $e_5$ |
| 0 | 1 | 1 | 0 | $e_6$ |
| 0 | 1 | 1 | 1 | $e_7$ |
| 1 | 0 | 0 | 0 | $e_8$ |
| 1 | 0 | 0 | 1 | $e_9$ |
| 1 | 0 | 1 | 0 | $e_{10}$ |
| 1 | 0 | 1 | 1 | $e_{11}$ |
| 1 | 1 | 0 | 0 | $e_{12}$ |
| 1 | 1 | 0 | 1 | $e_{13}$ |
| 1 | 1 | 1 | 0 | $e_{14}$ |
| 1 | 1 | 1 | 1 | $e_{15}$ |

Heavily condensed representation. Complete truth table would have $2^{20} \approx 1.000.000$ rows.

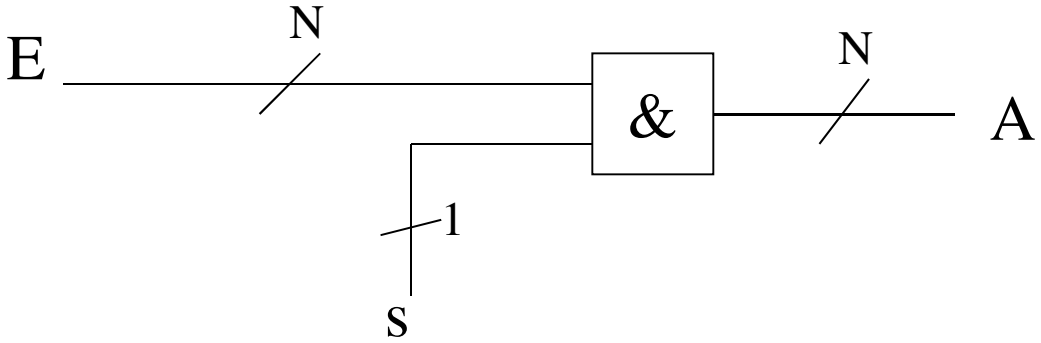
Possible Realization:



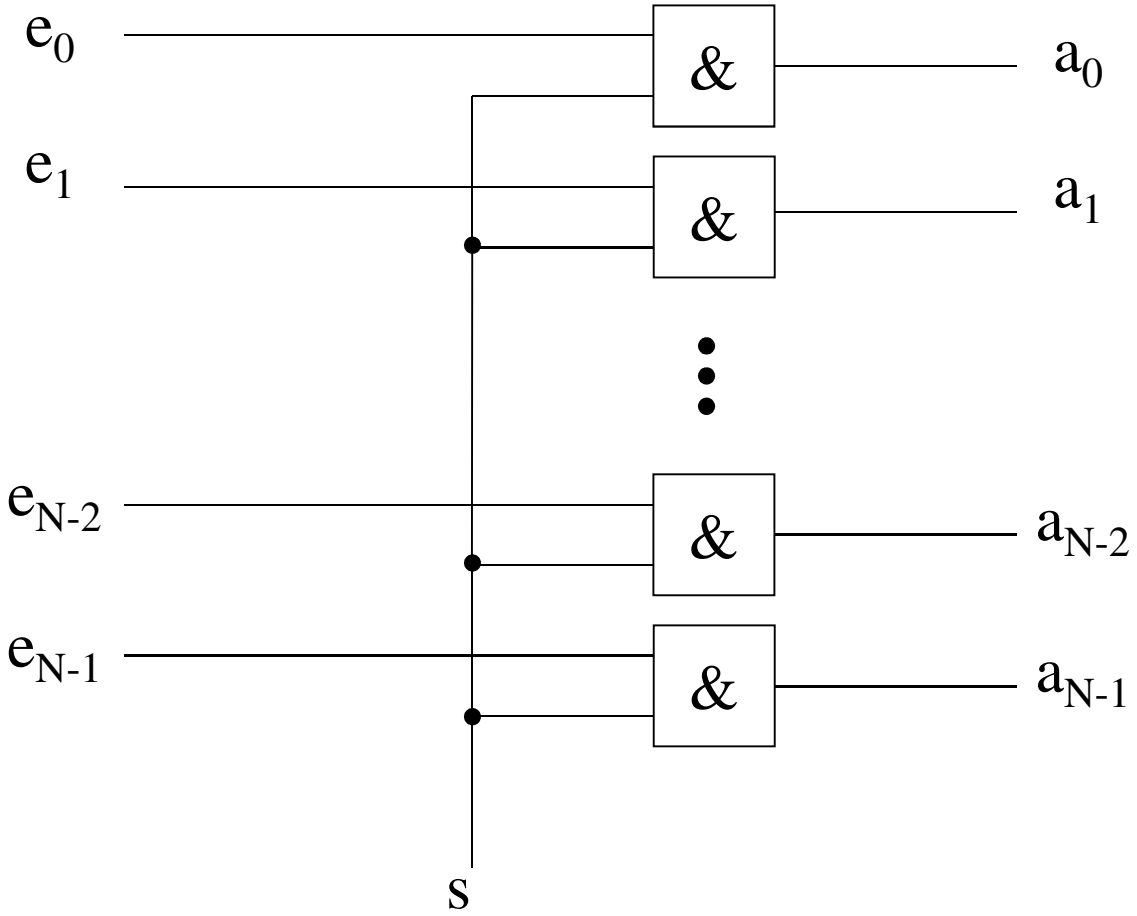
The data-path (Datenweg) switch

A data-path switch is a combinatorial switch, that transfers a given amount of inputs, $e_i$ unchanged to a corresponding amount of outputs, if a control input is being set to 1 (i=0,...,N-1).
If the control input is 0, all outputs are 0.

$$a_i = \begin{cases} e_i, \text{ if } s=1 \\ 0, \text{ if } s=0 \end{cases}$$

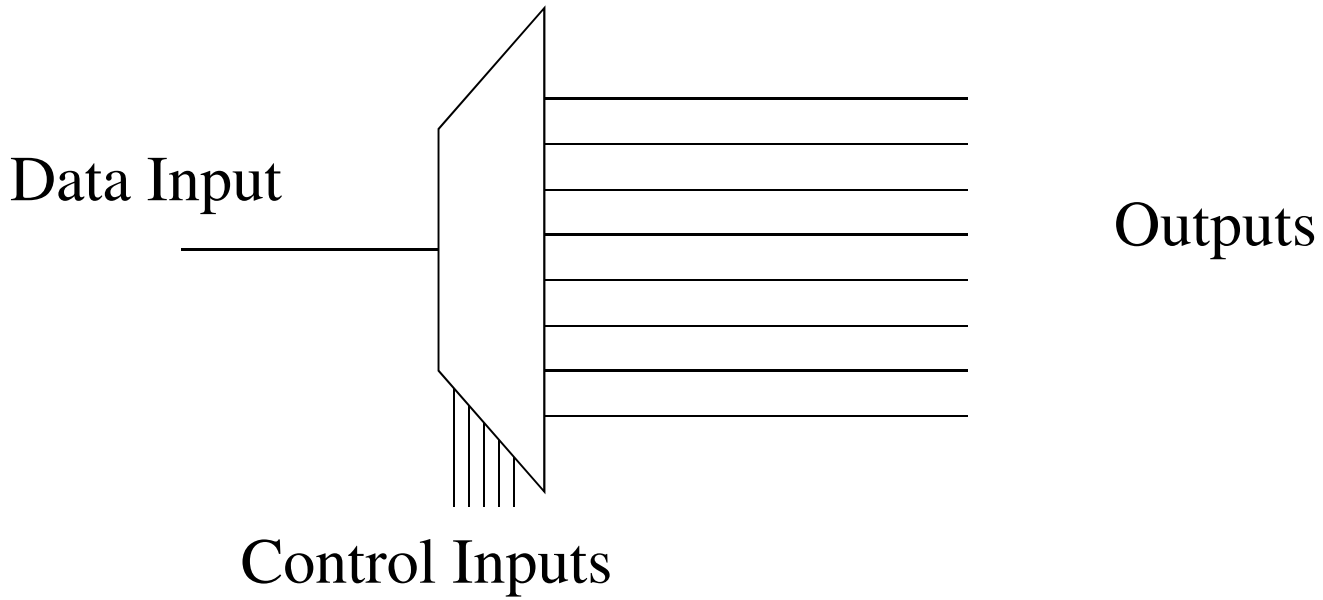


Realization of the data path switch



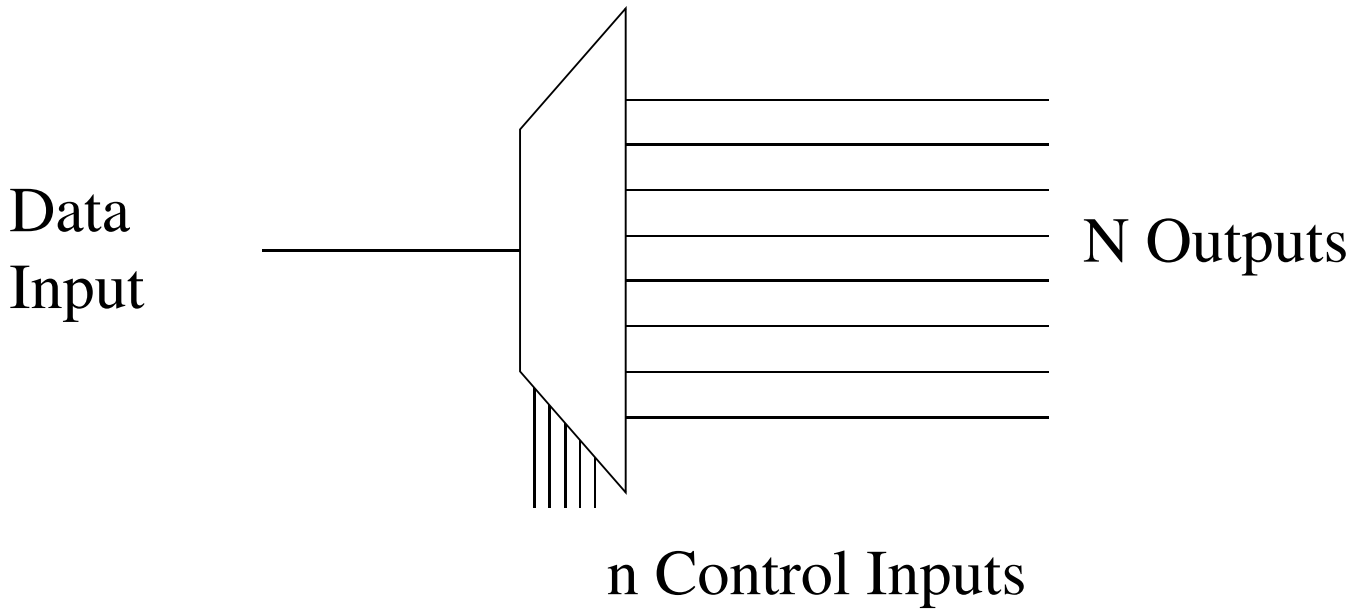
The Demultiplexer

A Demultiplexer is a combinatorial circuit that can switch its single data input to 1-out-of-N outputs. The control inputs determines which output is chosen.



Let the data outputs be represented with $a_i$, i=0,...,N-1, and the control inputs be represented with $s_i$, i=0,..,n-1, and the input with e. Then, N has to be $N \leq 2^n$. The function of the demultiplexer is described by

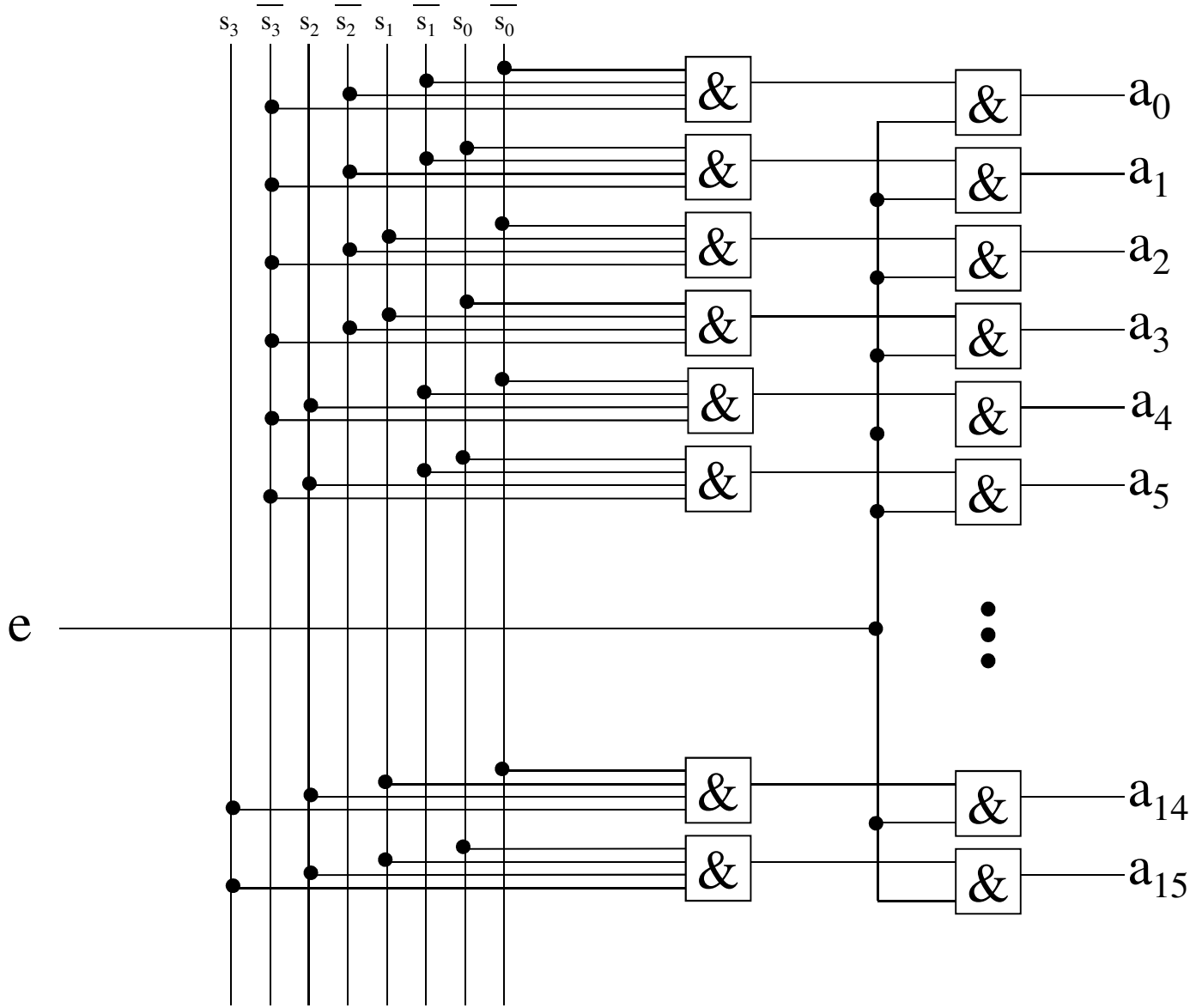
$a_i = e$, if $(i)_{10}=(s_{n-1}s_{n-2}...s_0)_2$ , else $a_i = 0$



# Truth table for 1-out-of-16-Demultiplexer

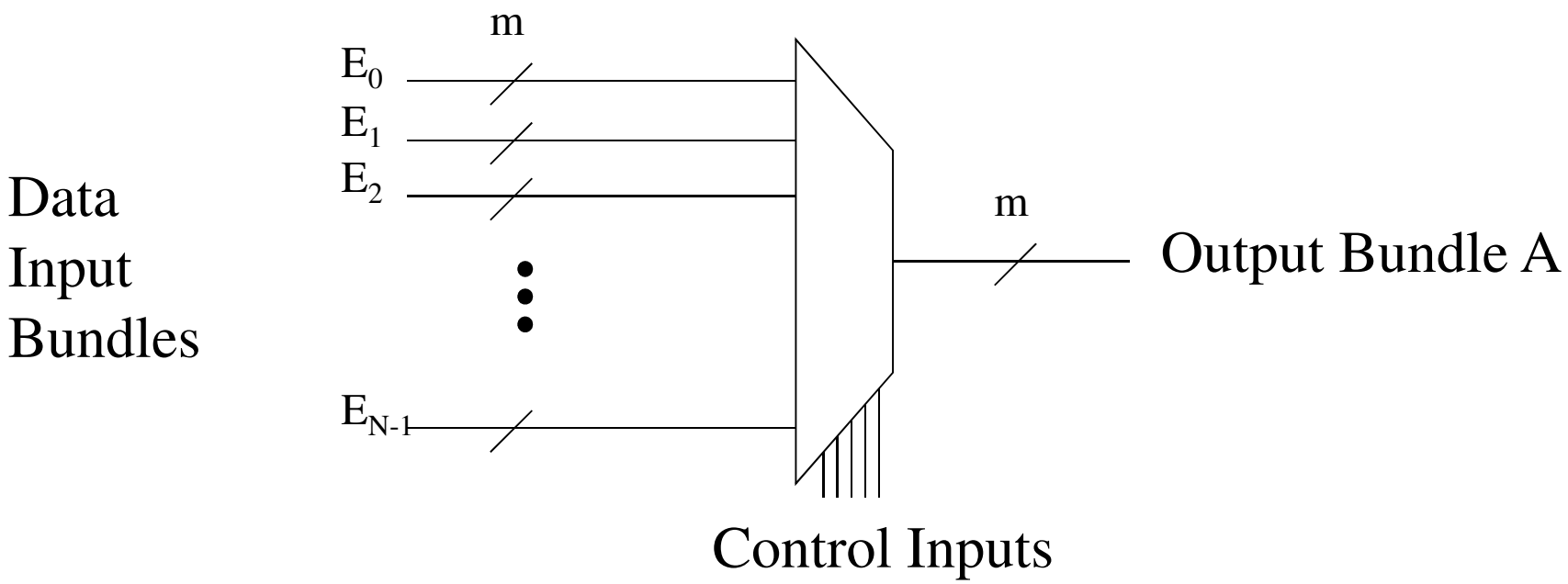
| e | $s_3$ | $s_2$ | $s_1$ | $s_0$ | $a_{15}$ | $a_{14}$ | $a_{13}$ | $a_{12}$ | $a_{11}$ | $a_{10}$ | $a_9$ | $a_8$ | $a_7$ | $a_6$ | $a_5$ | $a_4$ | $a_3$ | $a_2$ | $a_1$ | $a_0$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | X | X | X | X | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# Realization of a 1-out-of-16-Demultiplexer



The Data Path Multiplexer

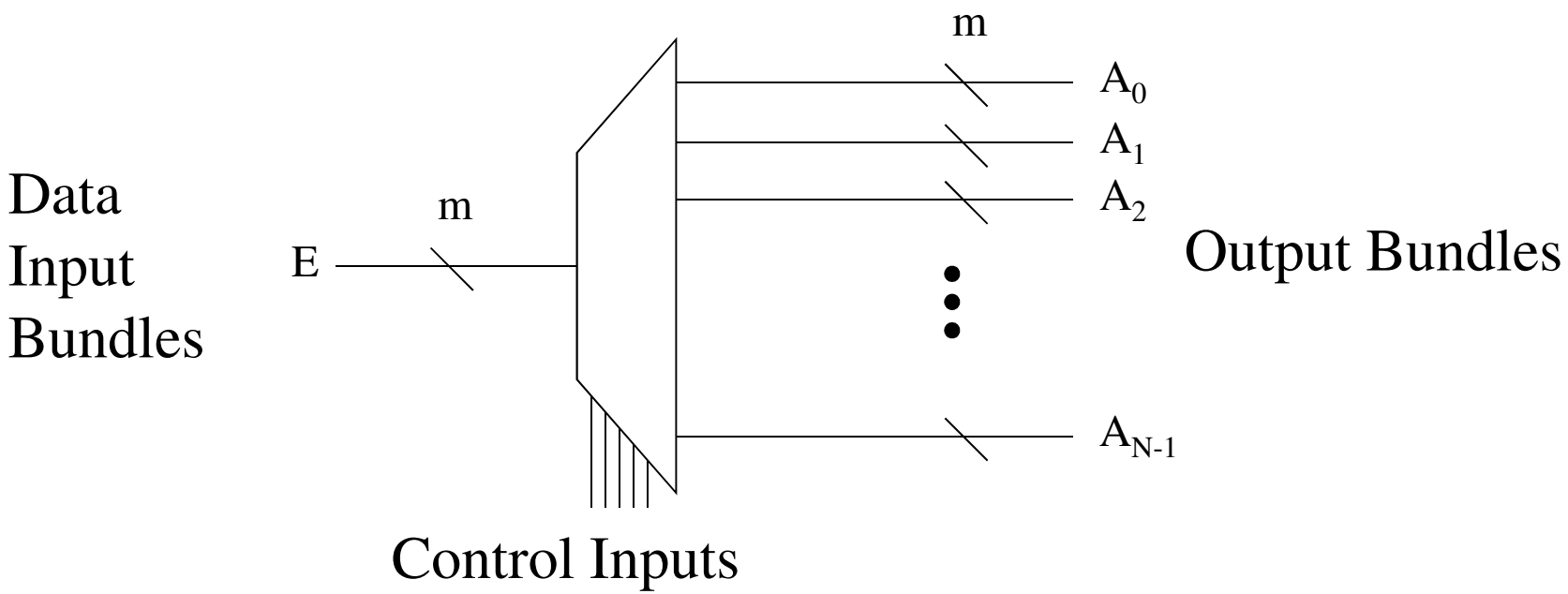
A data path multiplexer is a combinatorial circuit which consists of m multiplexers in parallel. It can be used to switch an amount of output, m. All parallel wires will be switched by the control inputs to one and the same output bundle.




$a_i = e_{k,i}$ is valid for all i=0,...,m-1, only if $(k)_{10} = (s_{n-1}...s_0)_2$

The Data Path Demultiplexer

A data path demultiplexer is a combinatorial circuit which consists of m demultiplexers in parallel. It can be used to switch an input bundle of the amount m to 1 of N outputs wired bundles.




$a_{k,i} = e_i$ is valid for all i=0,...,m-1, only if $(k)_{10} = (s_{n-1}...s_0)_2$
All other $a_{k,i}$ are 0.

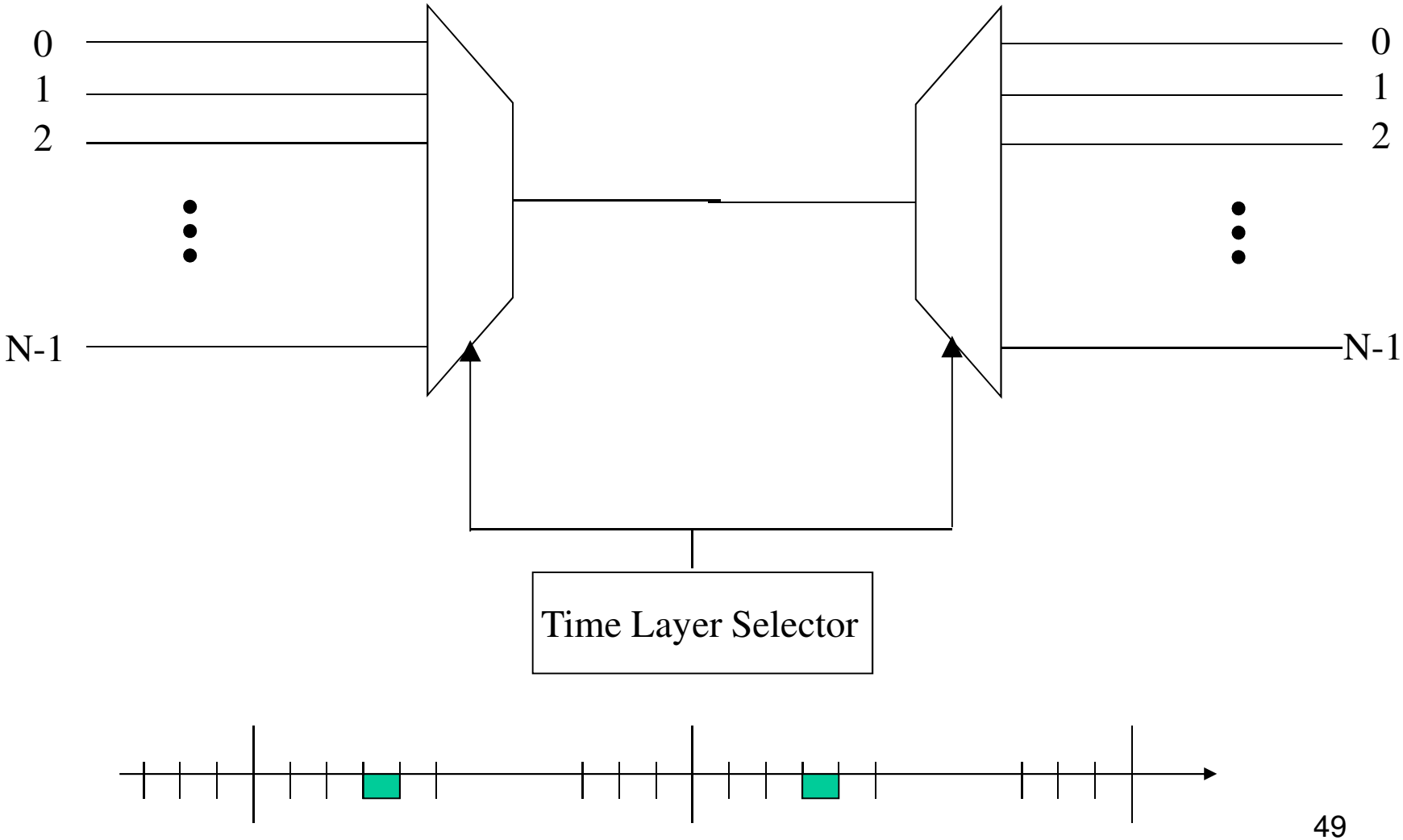
# Simplified Truth Table of a Data Path Demultiplexer

| $s_{n-1}$ | $s_{n-2}$ | ..... | $s_2$ | $s_1$ | $s_0$ | Output Bundles |
|---|---|---|---|---|---|---|
| 0 | 0 | .... | 0 | 0 | 0 | $A_0 = E, A_i = 0$ for $i \neq 0$ |
| 0 | 0 | .... | 0 | 0 | 1 | $A_1 = E, A_i = 0$ for $i \neq 1$ |
| 0 | 0 | .... | 0 | 1 | 0 | $A_2 = E, A_i = 0$ for $i \neq 2$ |
| 0 | 0 | .... | 0 | 1 | 1 | $A_3 = E, A_i = 0$ for $i \neq 3$ |
| 0 | 0 | .... | 1 | 0 | 0 | $A_4 = E, A_i = 0$ for $i \neq 4$ |
| ⋮ | | | | | | ⋮ |
| 1 | 1 | .... | 1 | 1 | 0 | $A_{N-2} = E, A_i = 0$ for $i \neq N-2$ |
| 1 | 1 | .... | 1 | 1 | 1 | $A_{N-1} = E, A_i = 0$ for $i \neq N-1$ |

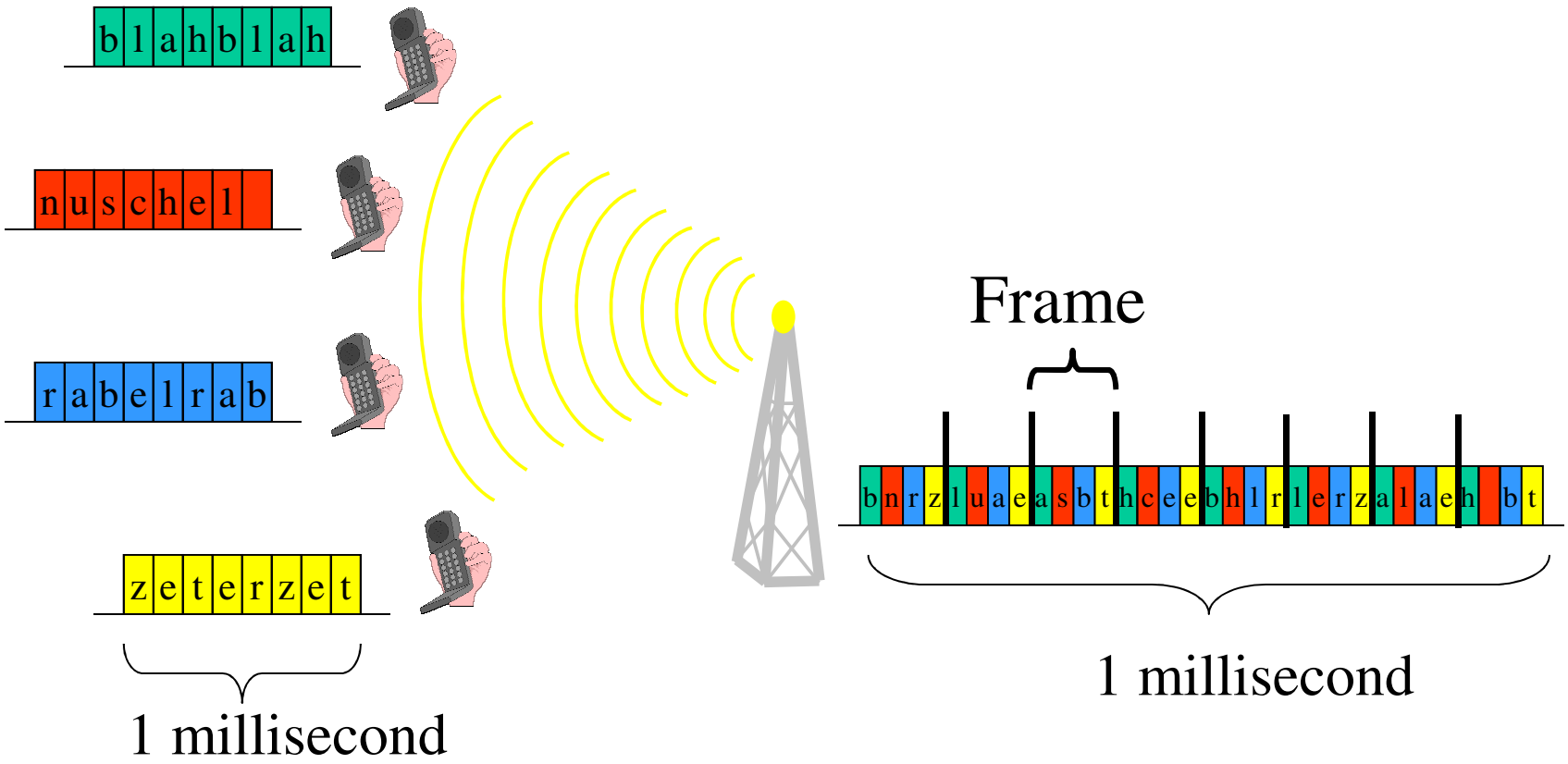
Time multiplexing transmission

Often it is necessary to use one wire for several different communications procedure. A simple example is your telephone connection at home. Using this technique (eg. ISDN), two telephone conversations and maybe an additional data transfer (over DSL) from a PC at the same time has to be handled. This works by alternating the connection for very short time frames for each contributing communication pair. This technique is known as **time multiplex technique**.
It can be done by a combinatorial circuit consisting of a multiplexer and a demultiplexer. Both of them are controlled by the same control unit, the so-called **time layer selector**.

Time multiplexing technique (Time Division Multiplex Access, TDMA)



# Example for time multiplexing: GSM network



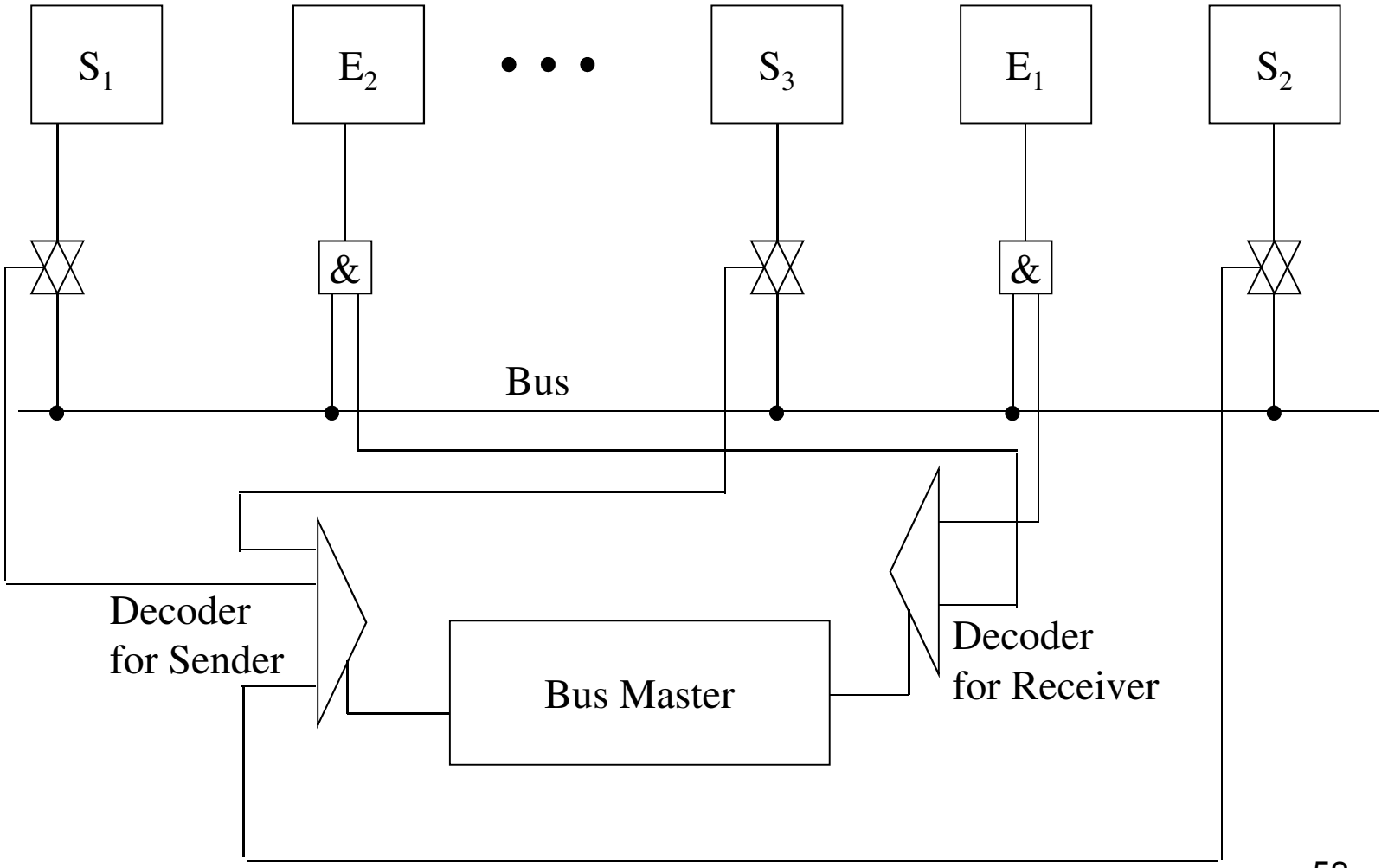
## The Data Bus

A time multiplexed method is very static because every communication pair is having a fixed part of the bandwidth of the wire. In lots of applications, it is much more important that the transmission bandwidth can be arranged dynamically to suit all the communication pairs, based on their needs. This leads to the concept of the data bus.

A data bus is a wire or a wire bundle. Through this, different connected devices can exchange data. These devices can write or read on this bus. Writing devices are also senders and reading devices are receivers. Of course, sometimes the same device, can act as a sender and a receiver.

In its most simplest form, the data bus has a shared wire (or wired bundle) which every contributor (device) uses it to write on or read from it. This is done by using transmission gates or data path switches for reading. The control wires are again served by a central unit (bus master, bus controller). This unit handles the allocation of the bandwidth of the bus for the communication pairs based on their needs.

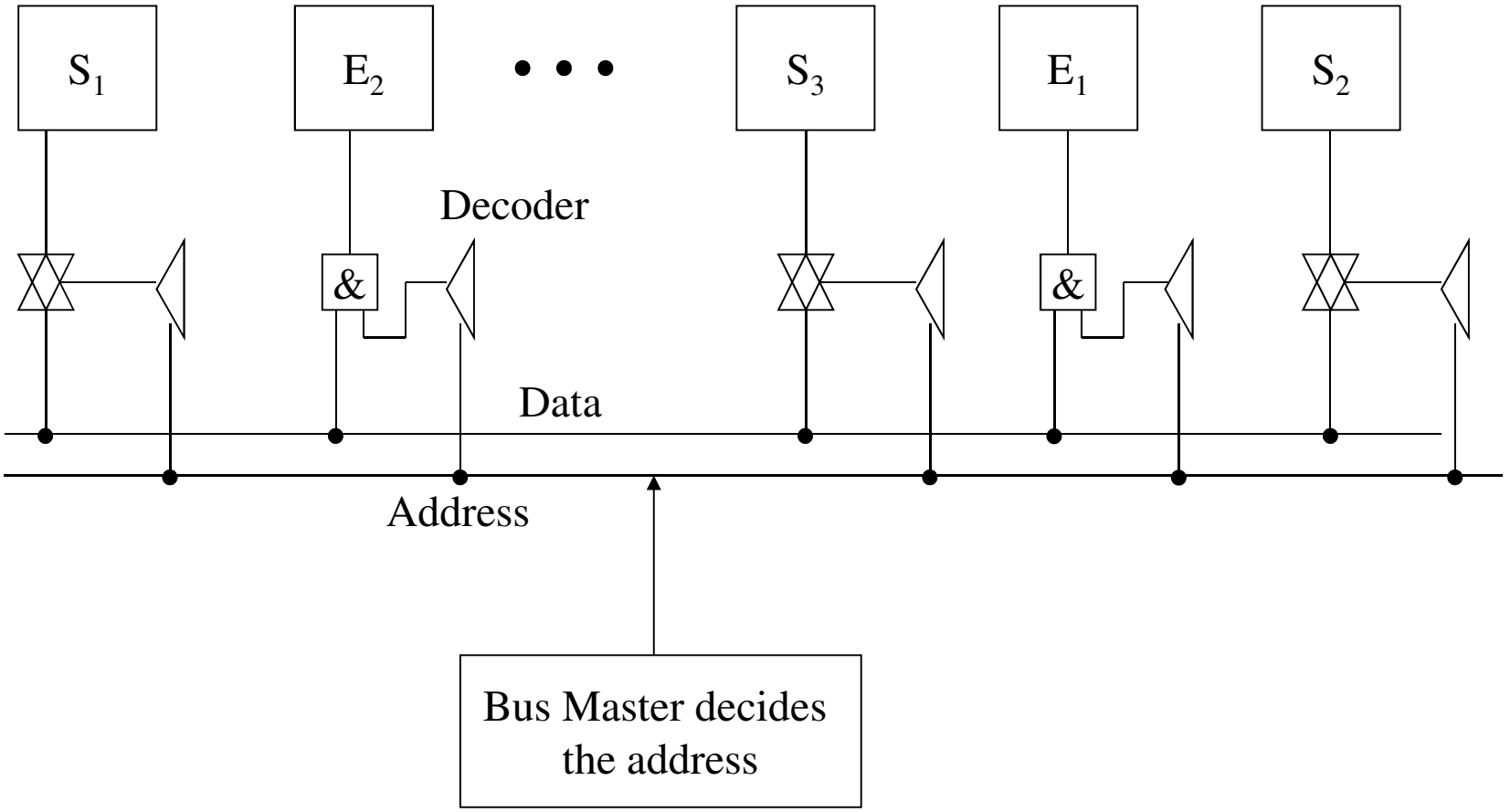
# Data Bus


S1
E2
S3
E1
S2
&
&
Bus
Decoder
for Sender
Bus Master
Decoder
for Receiver

The disadvantage is obvious: The bus master (bus controller, arbitrator) has to have a direct wire from every decoders to each potential sender and receiver. The wire has to be set to 1 at the right time if a sender wants to write something to one or several receivers.

This disadvantage can be avoided by adding an address wire bundle to the bus. Now the bus master needs only to put the addresses of the communicating pairs on the address bus and each sender and receiver has a decoder that reads the address bus. In case the decoder of the sender or the receiver reads its own address from the bus, the decoder generates a 1 for its transmission gates, and its data path switch respectively.

# Data Bus with Address Bus



In reality, it is very often that a communication device exchange lots of data continuously before it switches to another device. In this case, the selected address remains the same for a long time, while the data changes constantly. It is uneconomical to create its own dedicated wire bundle for the addresses (over a long time).

Instead we use the existing wire bundle for the data in time multiplex and also for the addresses. More exactly: in the beginning of a communication, the addresses of the senders and of the receiver/receivers are placed on the bus. Now the devices control their transmission gates and their data path switches. Subsequently, data will be transmitted continuously until the communication process has finished.

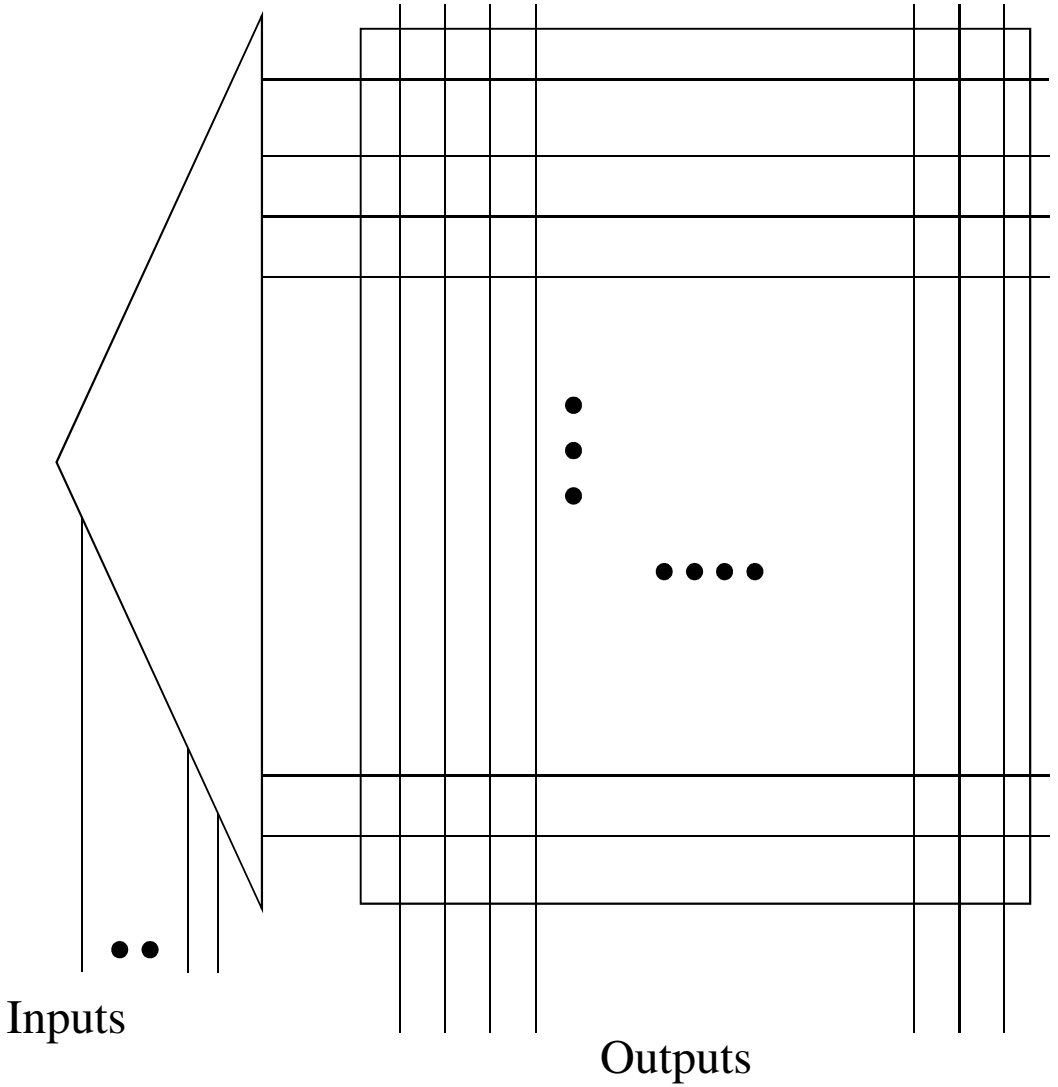
For such a bus, a protocol is needed that determines the interpretation and usage of the signals on the bus. E.g. every connected device to the bus has to be informed whether addresses or data is currently transmitted over the bus and also the end of the transmission has to be signaled.

## Combinatorial Circuit Realization as Memory

In the modern circuit technology, it is often too expensive to do a single realization or even a dedicated chip for every individual combinatorial circuits. Hence there are many used techniques on how to use existing universal chips, to make them fulfill the functionality of a given combinatorial circuit.

The simplest of these is the realization of combinatorial circuits as memory: the truth table of boolean functions will be written into the memory. The inputs will be connected to the address wires of the memory. The outputs of the memory are the outputs of the combinatorial circuits.

# Circuit Realization for Memory

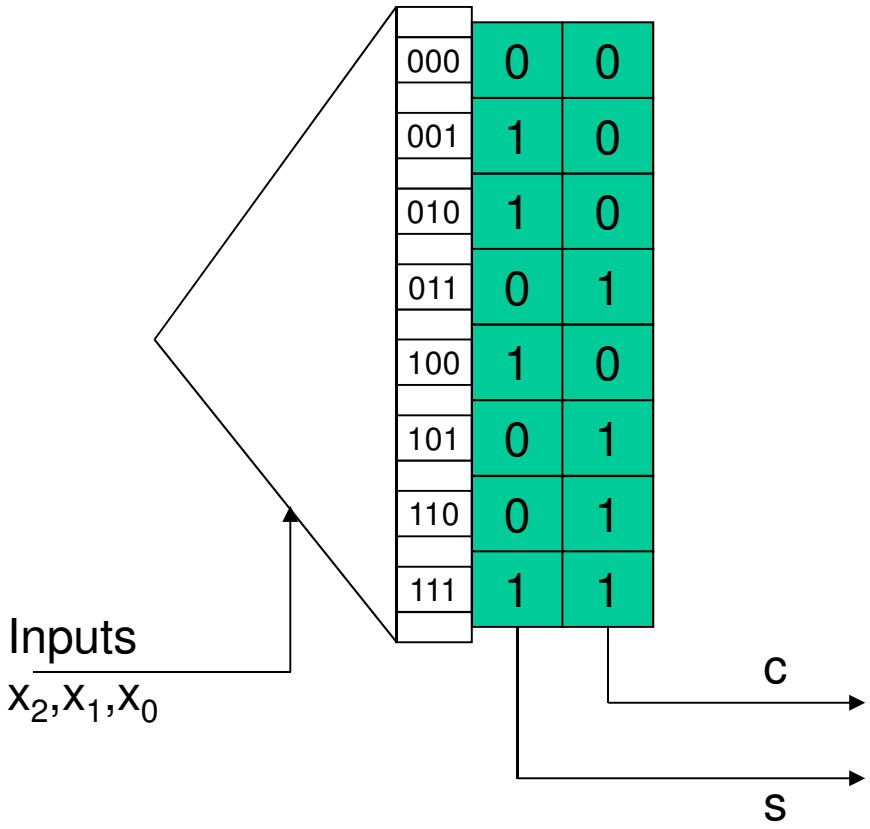


Example: A Full Adder, realised using a 8x2-bit ROM

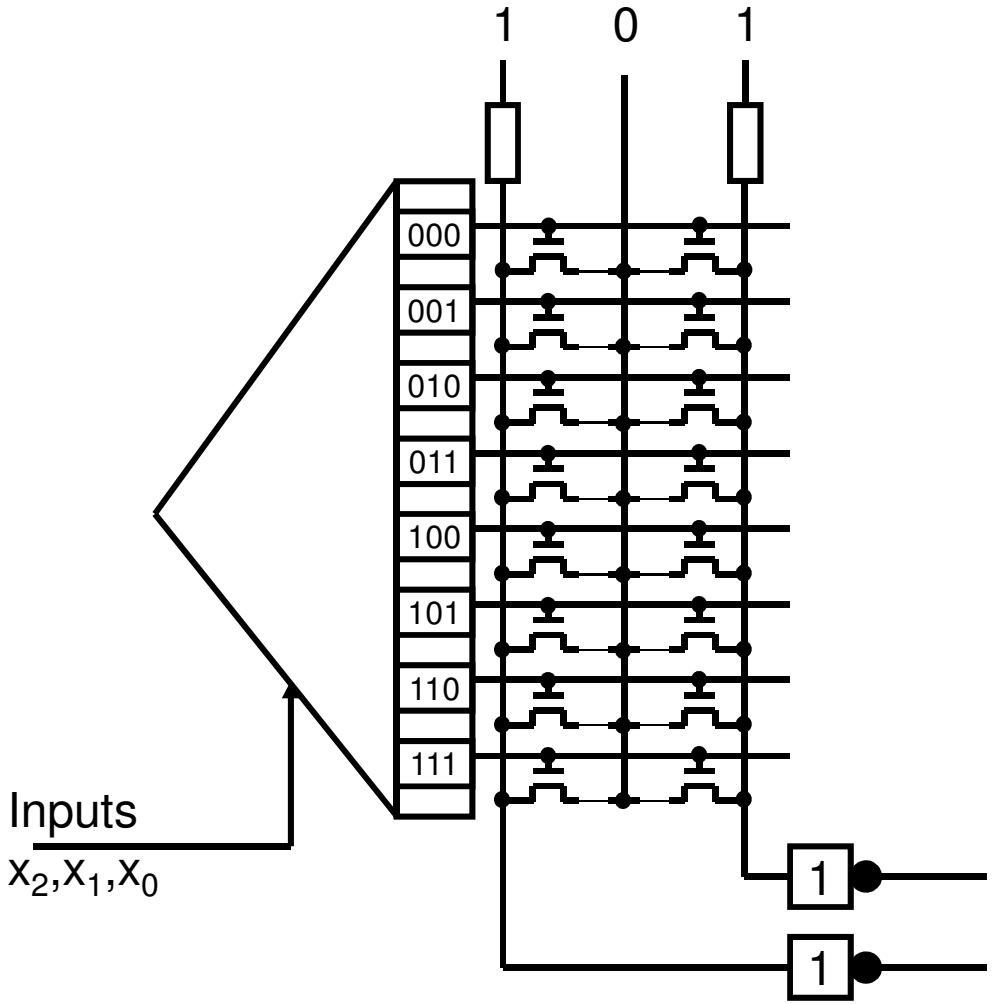
Truth Table

| $x_2$ | $x_1$ | $x_0$ | s | c |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

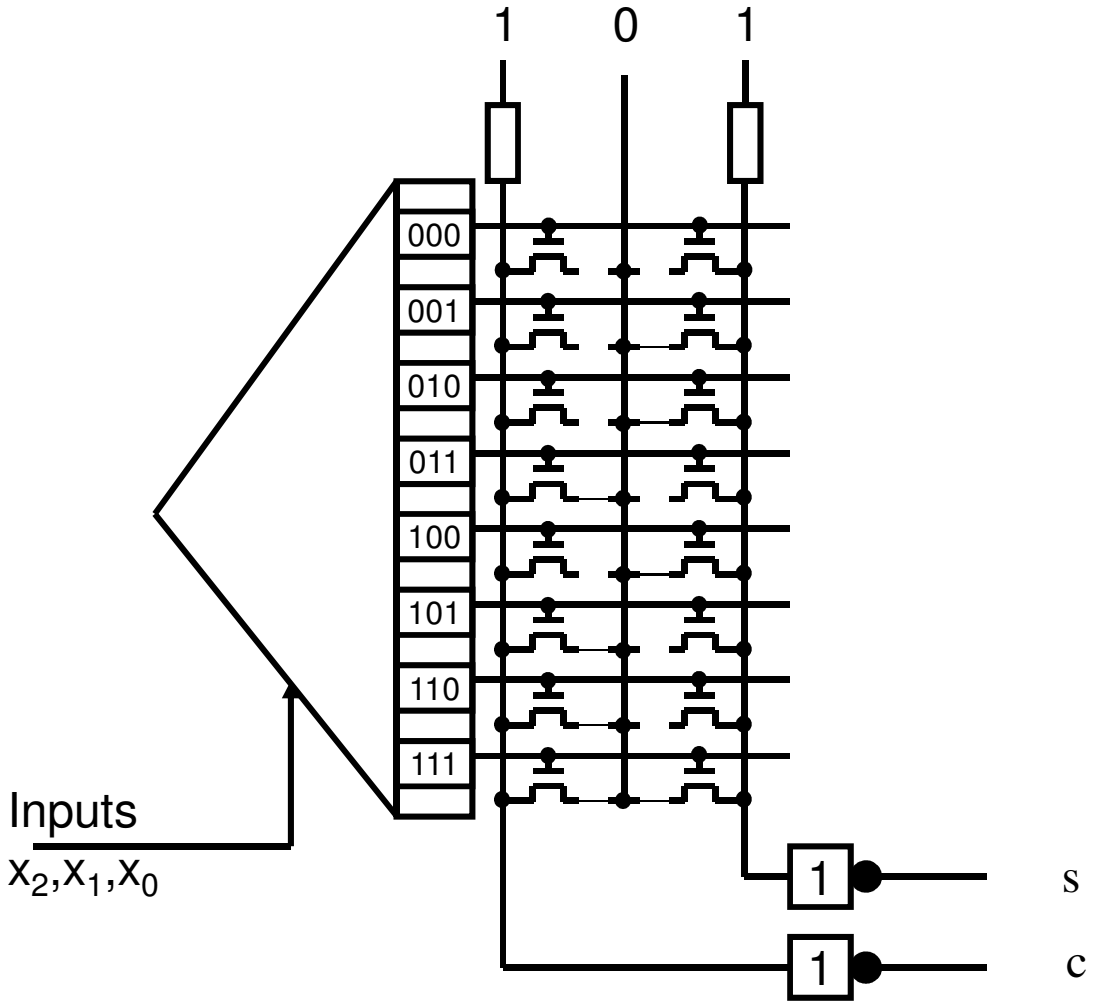
ROM



Example: A 8x2-bit ROM (Antifuse technique, i.e. not programmable)
before burning to become a full adder



Example: A full adder, realised using a 8x2-bit ROM (Antifuse technique, i.e. not programmable)



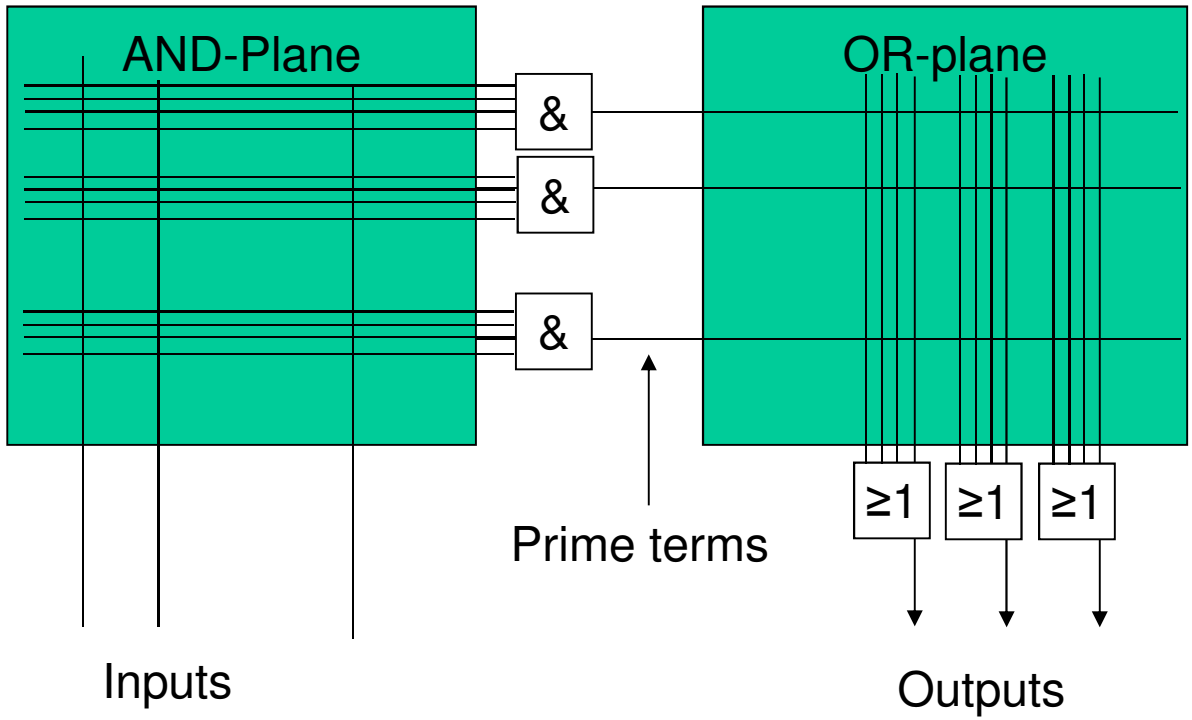
Combinatorial circuit realization in the form of PLA (Programmable Logic Array)

A different way of realising a combinatorial circuit is the PLA. It is based on the idea of a free form for a boolean function with n inputs and m outputs. This form can be “filled out” (burnt) depending on the individual functions.

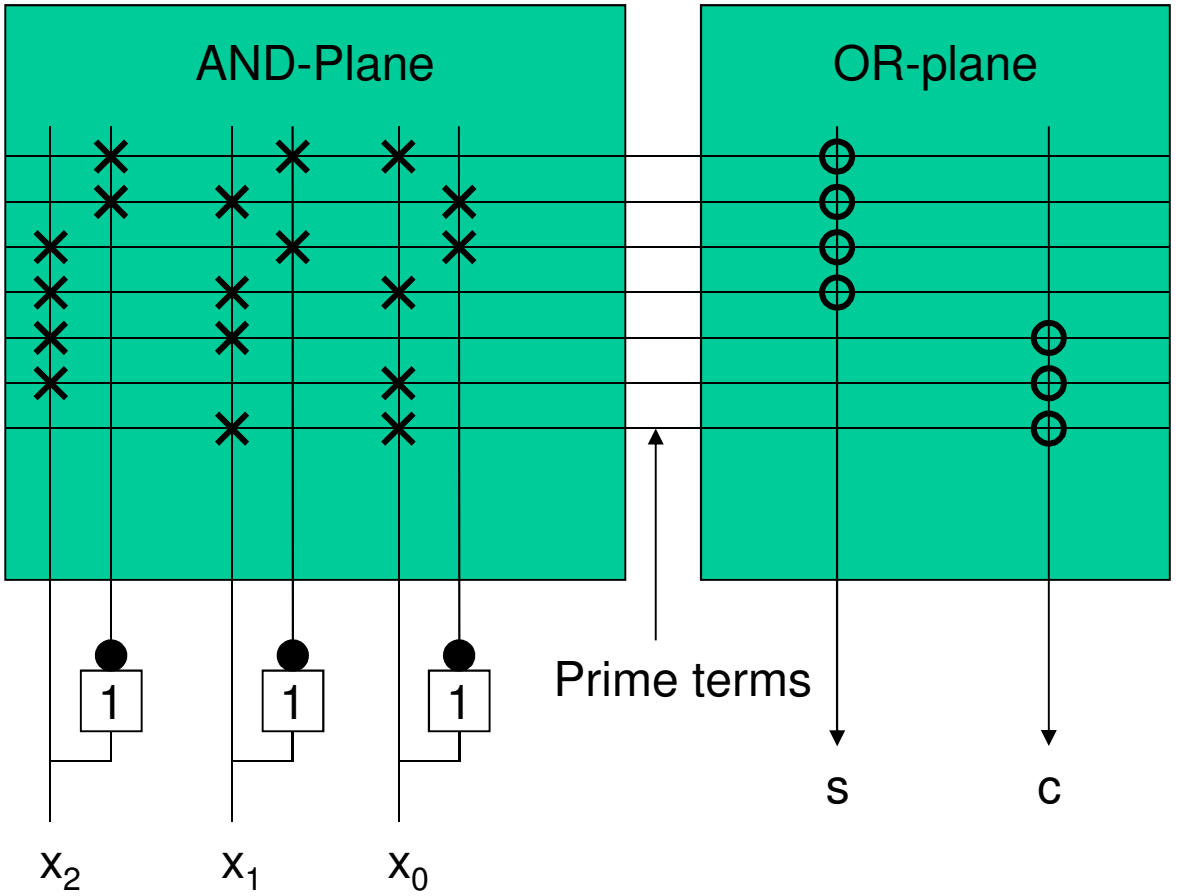
The AND plane generates the product terms (makes sense to put the product terms in DMF, hence prime terms) The OR plane connects these in a disjunction. Of course, the actual realization is not done by AND and OR but instead done using NAND gates. We know this technique already.

It makes sense to use the following notation to represent it in a PLA: You only mark the AND connection with crosses if the wires have influence on the outputs and the same goes for the OR connection using circles.

Realization of a Combinatorial Network as PLA with AND and OR Plane



Example: Full Adder as PLA with AND and OR Plane



Example: Function $y_1 = \overline{x}_0 x_1 \overline{x}_2 + \overline{x}_0 \overline{x}_1 x_2$ and $y_0 = \overline{x}_0 \overline{x}_1 x_2$
realized as PLA with AND- and OR-Plane