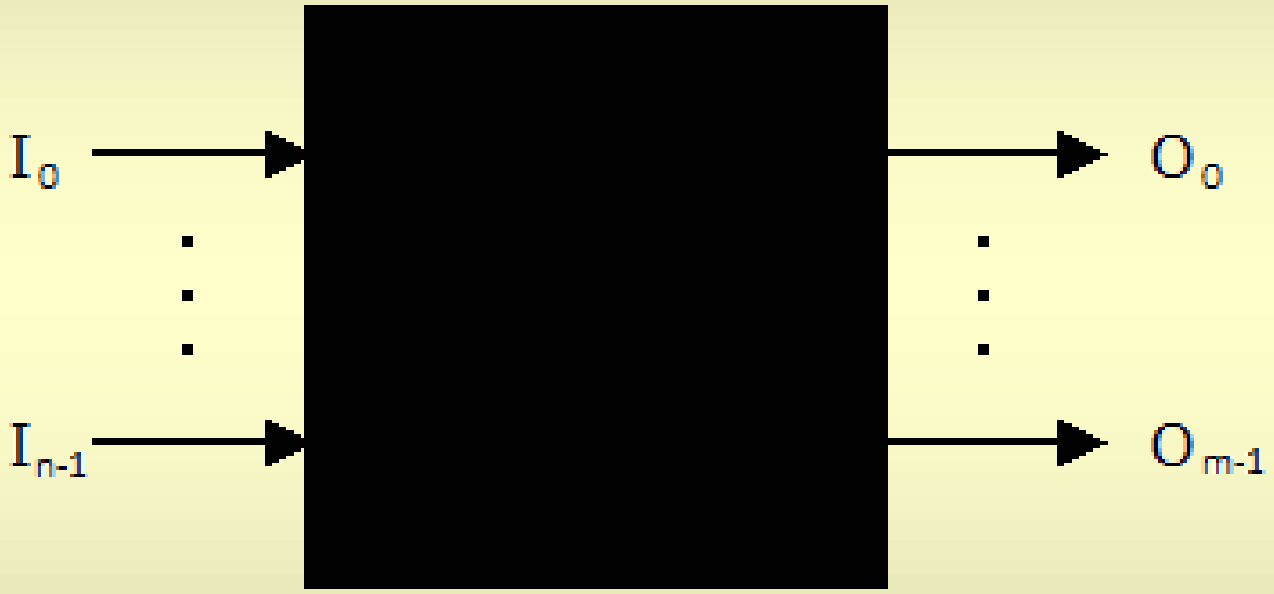
II.5. Combinational Circuits

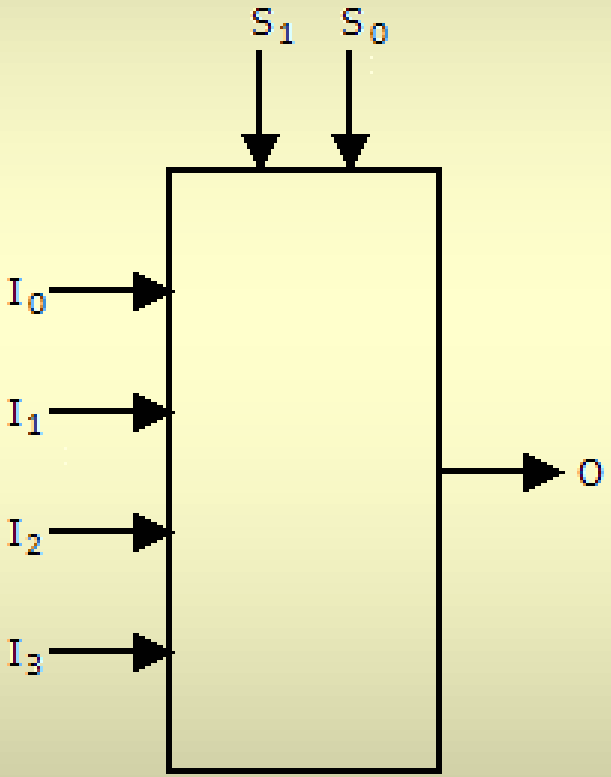


- output values depend only on the input values at the current moment

The Multiplexer

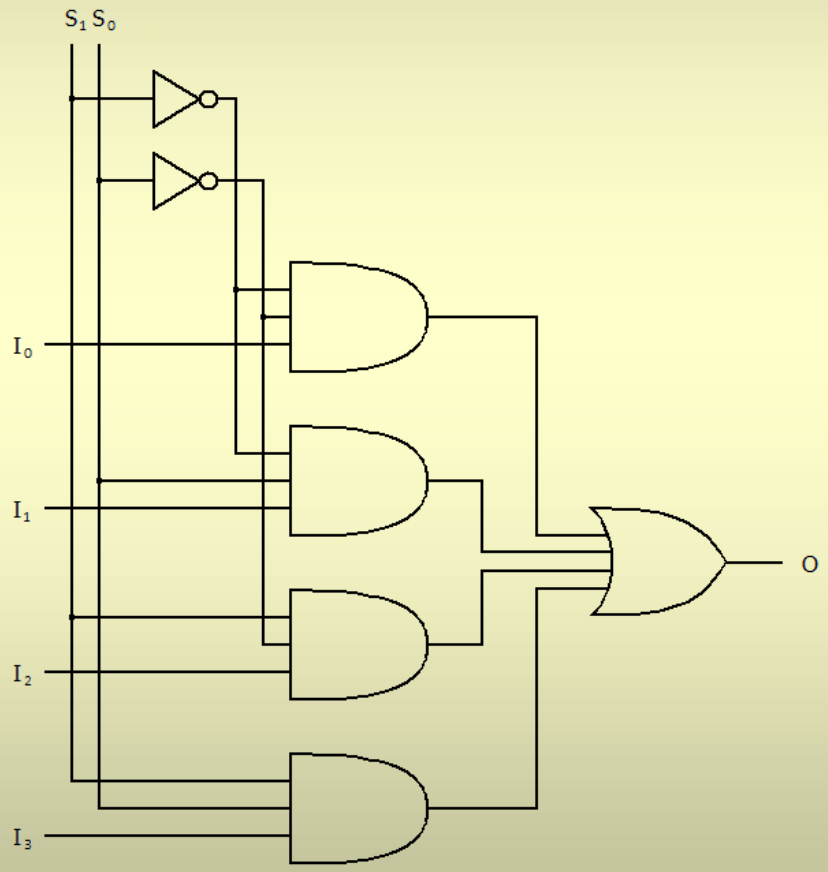
- 2^n inputs (data)
- n selection inputs (control variables)
  - control (address) bits
- a single output
- each data input corresponds to a DNF term
- one of the inputs (data bit) is selected - copied to the output

Multiplexer 4→1 (*n*=2)



S_1	S_0	O
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Logic Diagram (4→1)

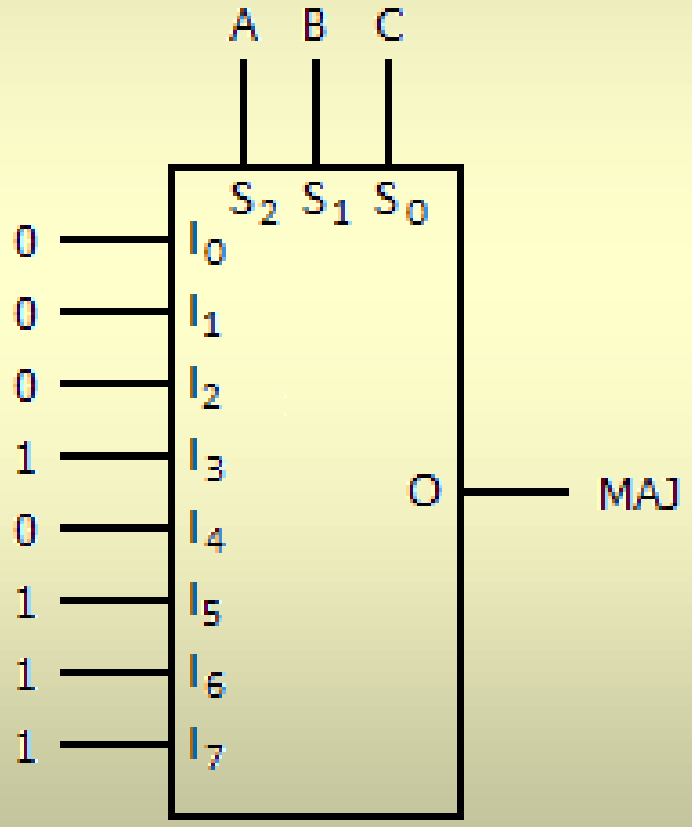


Implementation of Boole functions

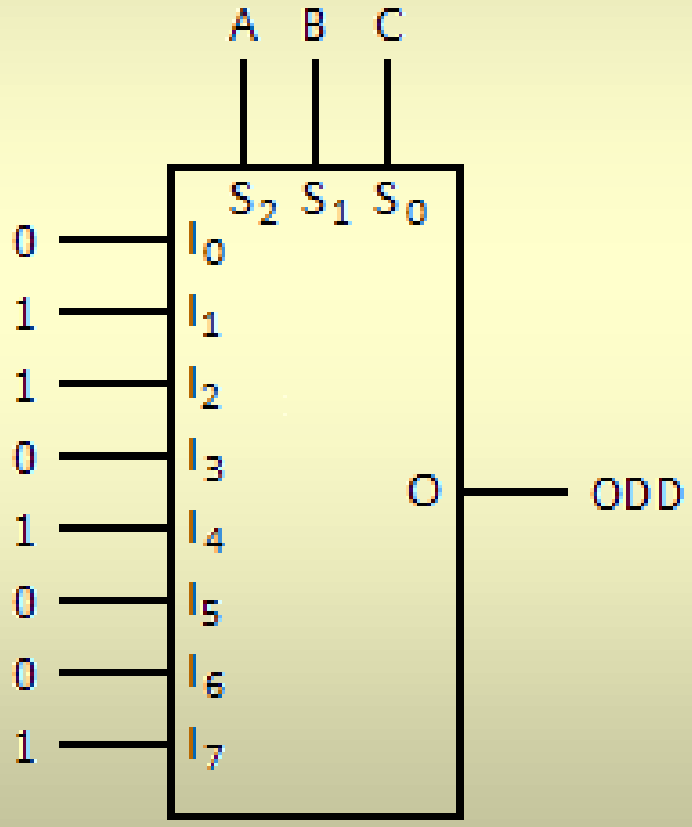
- selection inputs make a number
- which is the index of the data input to be selected for the output
- we can thus implement Boole functions through the multiplexer
  - data inputs - the outputs corresponding to the rows in the truth table
  - selection inputs - inputs of the Boole function

Examples

majority of 3



odd

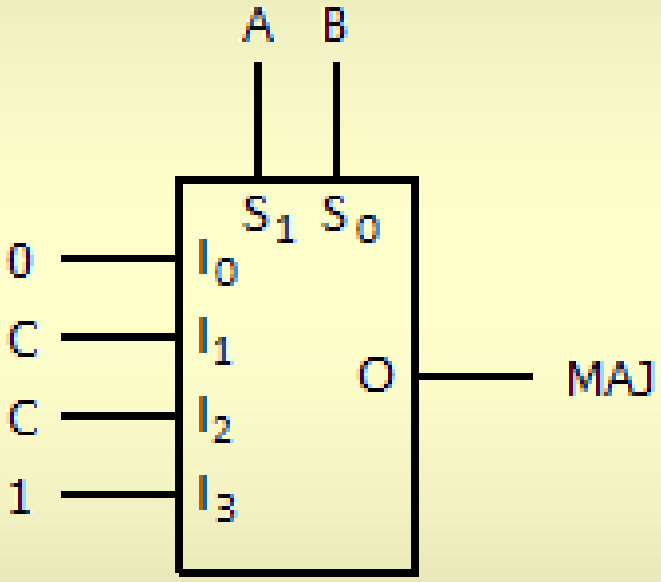


Efficient Implementation - folding

majority of 3

A	B	C	
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

A	B	
0	0	0
0	1	C
1	0	C
1	1	1

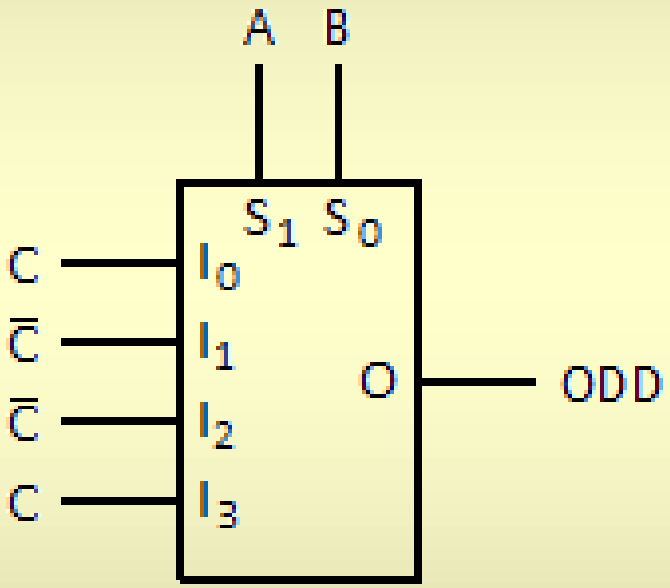


Efficient Implementation - folding

A	B	C	
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

odd

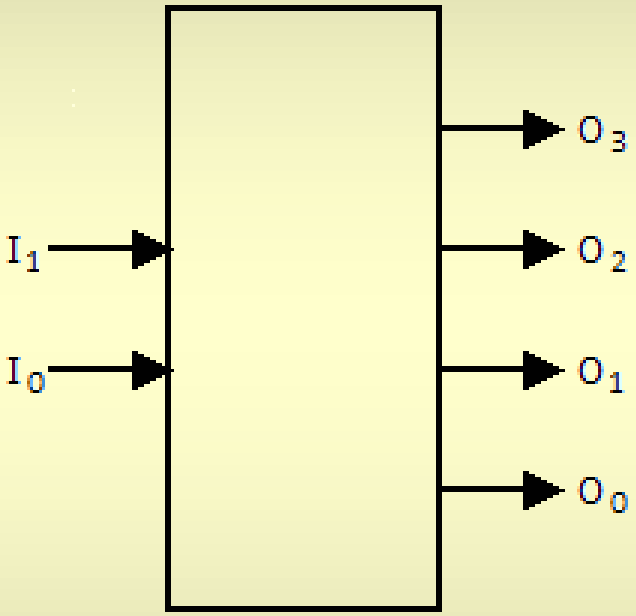
A	B	
0	0	C
0	1	$\overline{C}$
1	0	$\overline{C}$
1	1	C



The Decoder

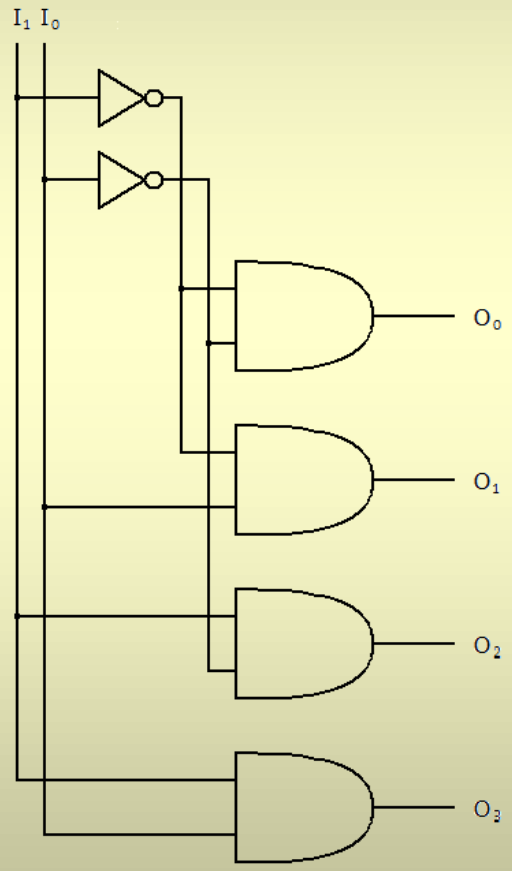
- n inputs
- 2^n outputs
- at each moment, exactly one of the outputs is activated
  - the one whose index is the number made by the inputs
  - each output corresponds to a DNF term written with the input variables

Decoder *n*=2



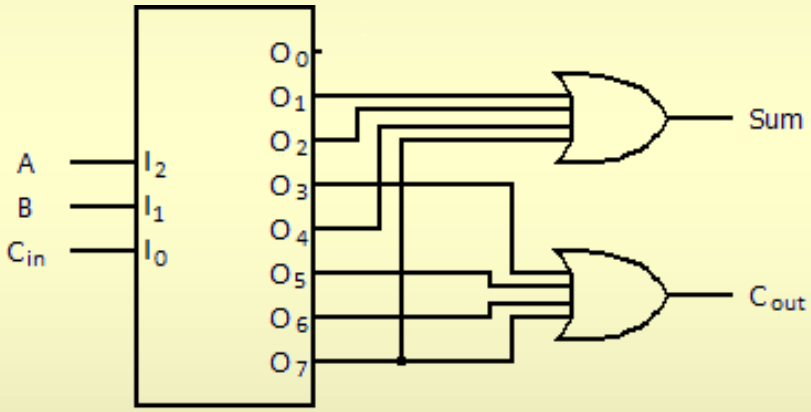
I_1	I_0	O_3	O_2	O_1	O_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Logic Diagram (*n*=2)



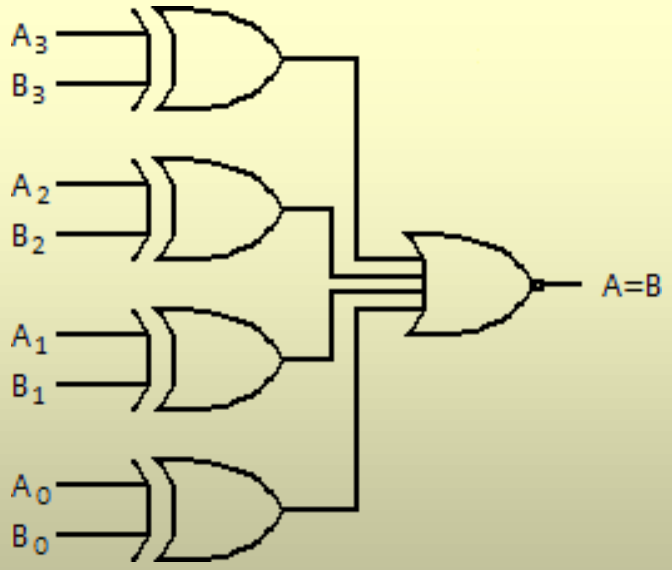
Addition - Decoder Implementation

A	B	C_{in}	Sum	C_{out}
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



The Comparison Circuit

- comparison operators: = , > , < , ≥, ≤
  - example of implementation: 4-bit equality
  - homework: complete comparison (<, =, >)

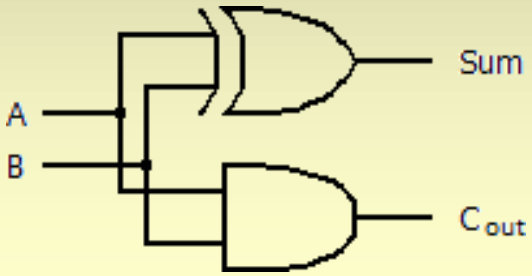


Adders

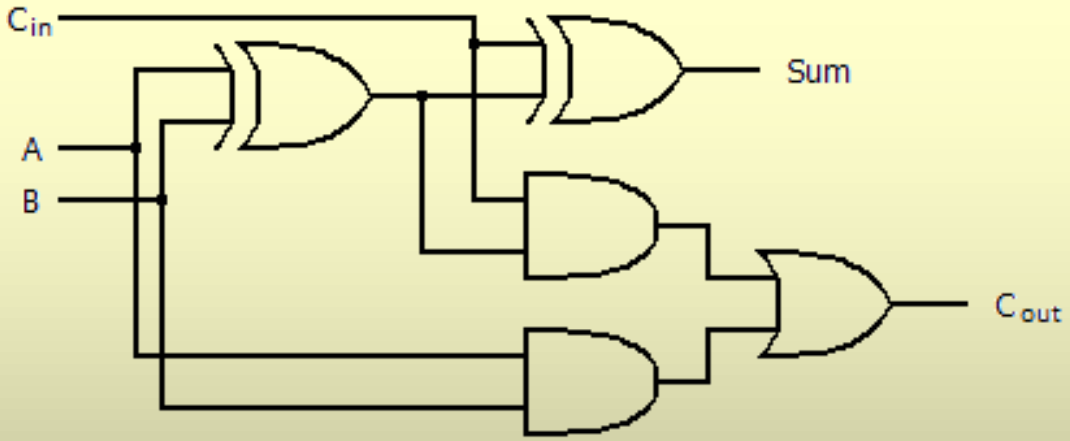
- half-adder
  - Adds the two input bits
  - outputs: a sum bit and a carry bit
  - cannot be extended for adding longer numbers
- full adder
  - Adds the three input bits (including carry in)
  - same outputs: a sum bit and a carry (out) bit
  - can be extended for multiple digits (bits)

Logic Diagrams

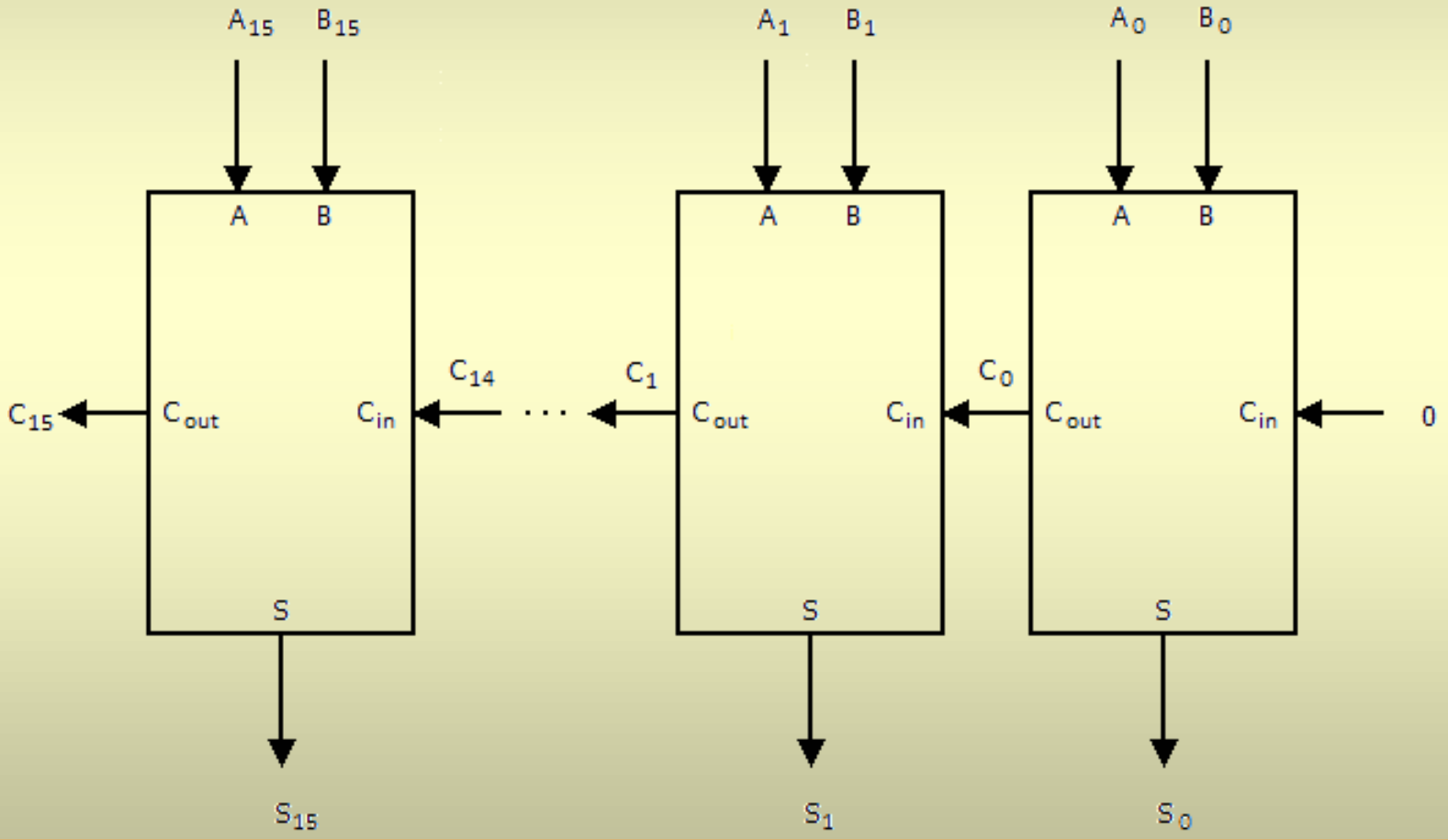
A	B	Sum	C_{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



A	B	C_{in}	Sum	C_{out}
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



Ripple-carry Adder (16 bits)



Ripple-carry Adders

- this kind of addres uses carry propagation
- advantage: the same (simple) circuit, repeated
- drawback: low speed
  - for each rank, one must wait for the result from the previous rank
  - so the delay is proportional to the number of bits

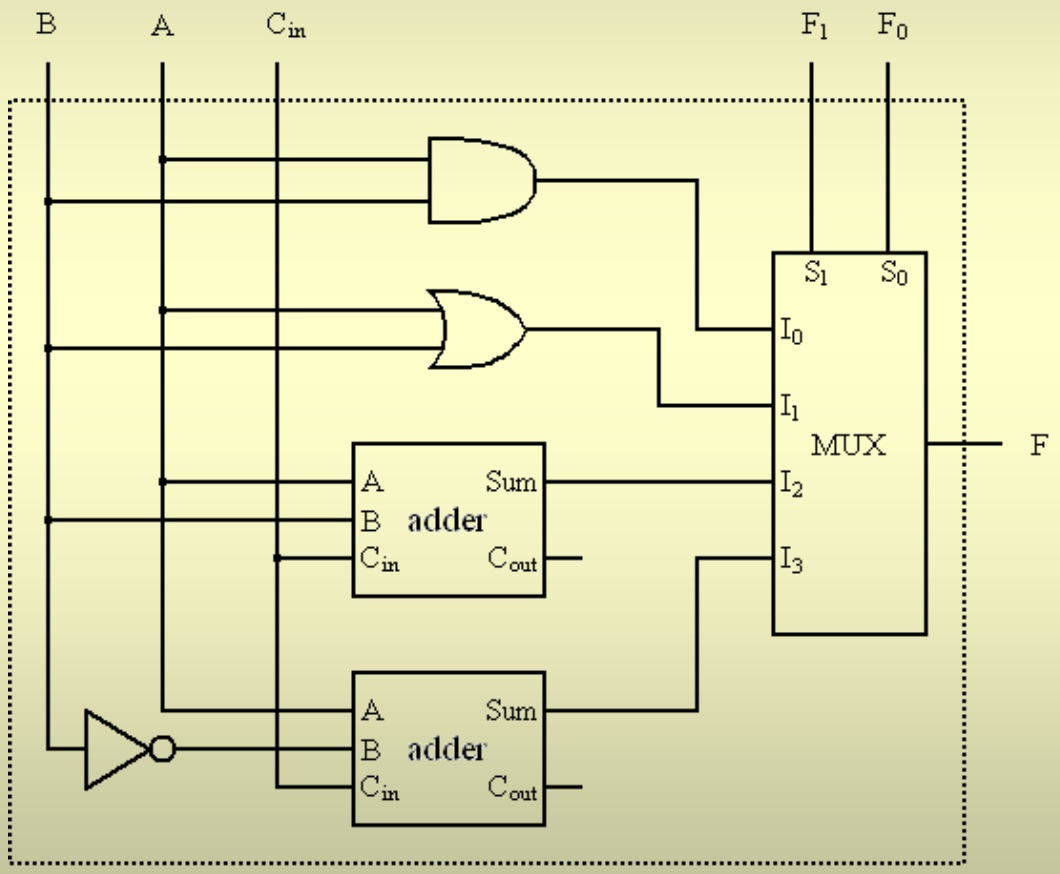
Speed-up (1)

- carry lookahead adder
  - carry in - generated independently for each rank
    - $C_0 = A_0 B_0$
    - $C_1 = A_0 B_0 A_1 + A_0 B_0 B_1 + A_1 B_1$
    - ...
    - $C_i = G_i + P_i C_{i-1} = A_i B_i + (A_i + B_i) \cdot C_{i-1}$
    - ...

Speed-up (2)

- carry lookahead adder (continued)
  - advantage - hish speed
    - eliminates the delay caused by carry propagation
  - drawback - requires complex additional circuits
  - usually - combination of the two methods
- carry selection adder
  - for each rank, compute the sum for both C_{in}=0 and C_{in}=1, the select the correct result

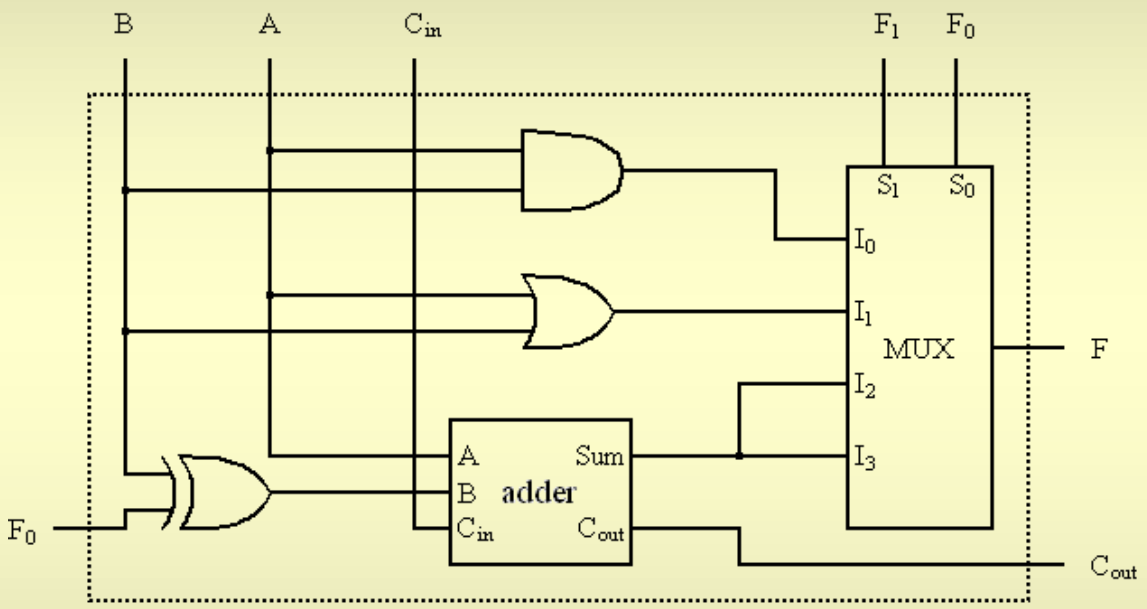
Elementary Arithmetic Logic Unit (1 Bit)



F_1	F_0	F
0	0	A and B
0	1	A or B
1	0	A+B
1	1	A-B

F_1,F_0 - control signals

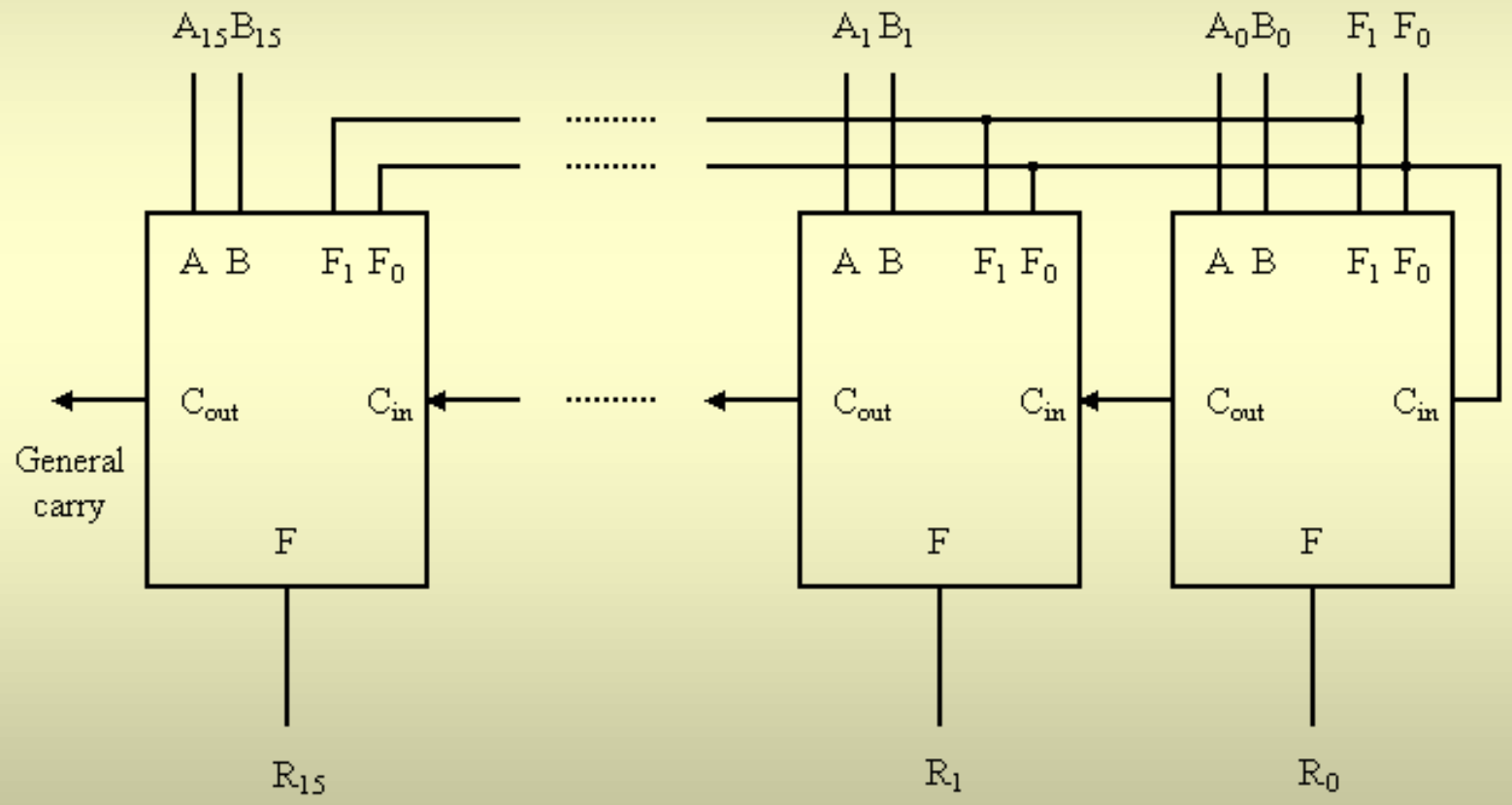
Improved Design



F_1	F_0	F
0	0	A and B
0	1	A or B
1	0	A+B
1	1	A-B

F_1,F_0 - control signals

Arithmetic Logic Unit (16 Bits)

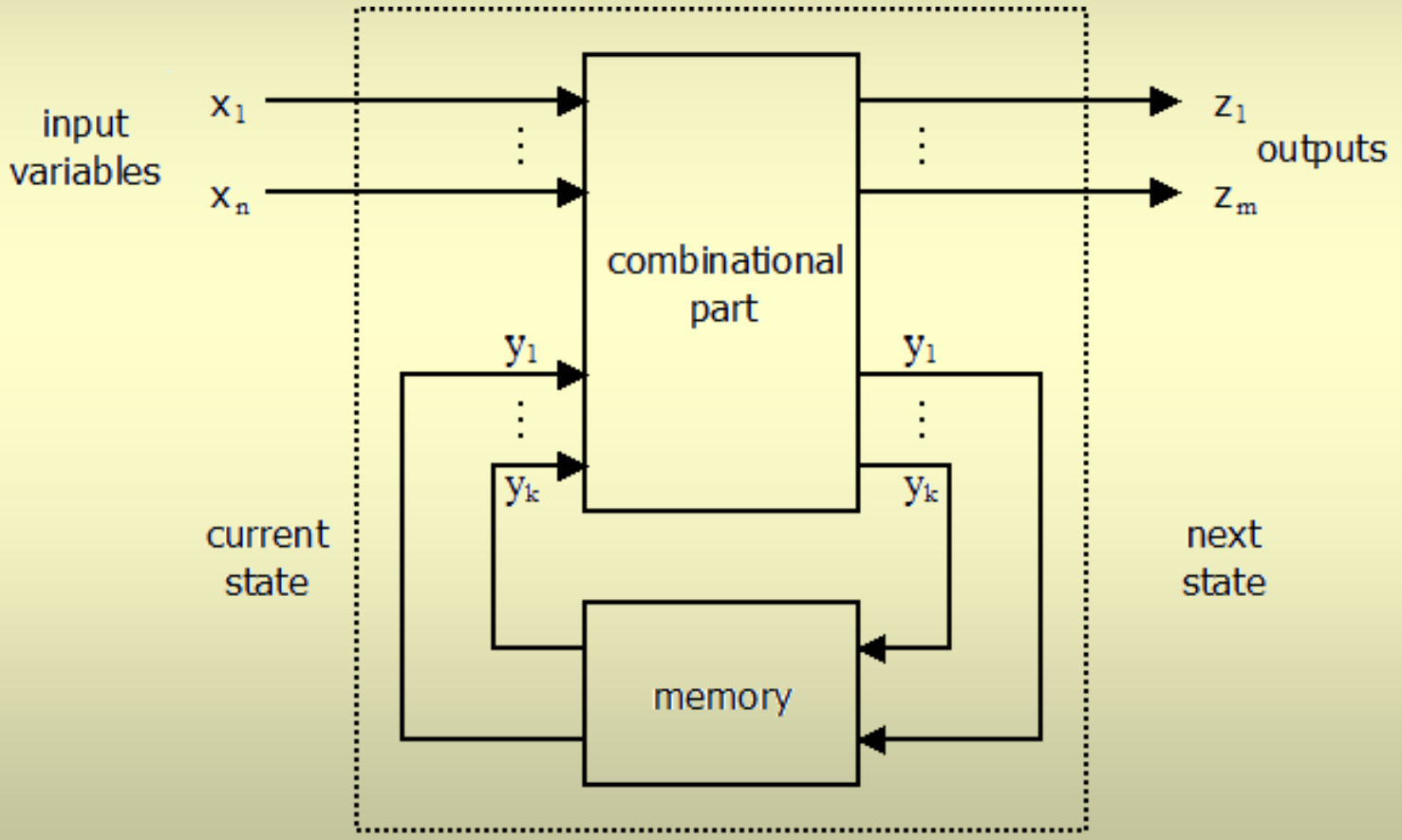


III. Sequential Circuits

Sequential Circuit

- at each moment, the output depends on
  - the input
  - the internal state
- so, for the same input, different output values may emerge, at different moments
- internal state
  - memorized by the circuit
  - evolves in time

Block Diagram



State Evolution

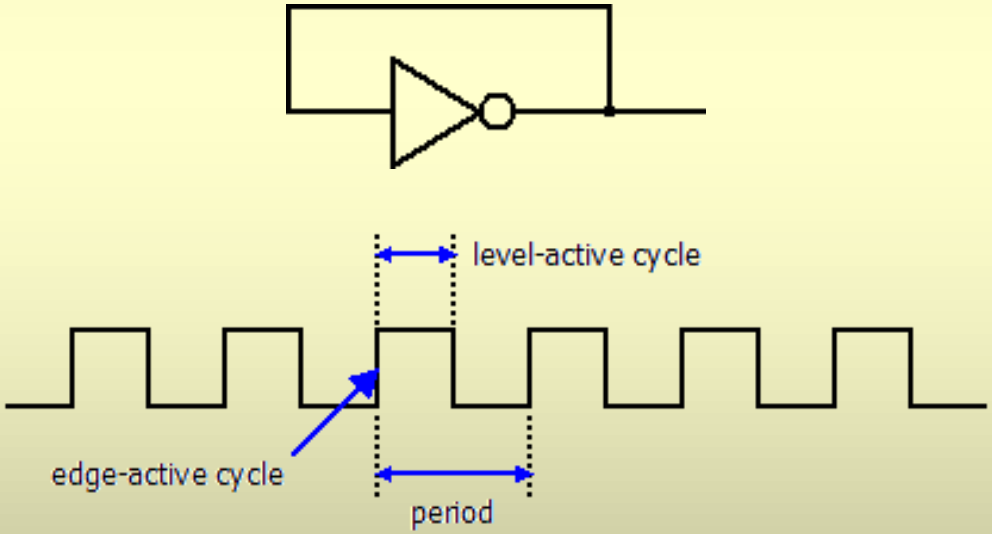
- the state changes at certain moments
  - synchronous: at regular time moments
    - provided by a special signal (clock)
  - asynchronous: when an event occurs
    - events are defined depending on the circuit's activity
  - why doesn't it change permanently?
    - sending the signals through gates and communication lines - with delays
    - so signals are considered after they have stabilized

The Clock

- periodic signal
  - active cycle - the percentage of the period when the signal is active
  - depends on what active signal means
    - level-active (during value 0 or 1)
    - edge-active (changing from 0 to 1 or the reverse)
- period duration
  - long enough for the signals to stabilize

Implementation

- the simplest logic diagram
  - the feed-back connection is essential



- usually, more complex schemes are used

Types of Sequential Circuits

- bit-level - bistable circuits
- depending on how the clock is detected
  - *latch* - level-active
  - *flip-flop* - edge-active
- multi-bit circuits
  - registers, counters
  - made of multiple bistable circuits

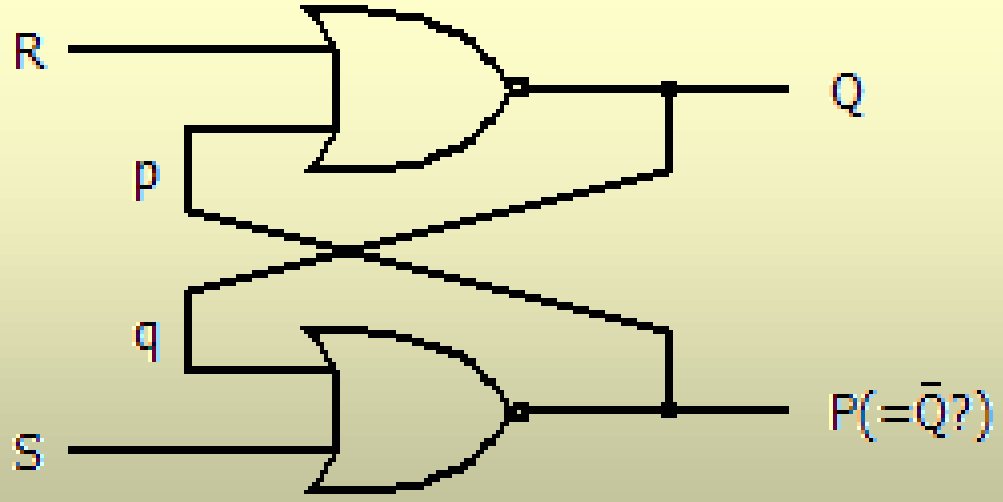
III.1. Bistable Circuits

Bistable Circuit

- what should a bit-implementation circuit look like ?
- specifications
  - can write either 0 or 1 to it
  - memorize the last written value until a new one is written
  - can read the last written value
- cannot be a combinational circuit (memorization)

No-clock RS Bistable Circuit

- two inputs (R,S), two outputs (Q,P), two feed-back connections
  - the circuit implements a single bit: $P = \overline{Q}$



How It Works (1)

- at first sight we have simultaneously

  $q = Q$ and $p = P$

  $Q = \overline{p + R}$

  $P = \overline{q + S}$

- in fact, when inputs change, outputs do not change instantly
  - due to gate propagation delays
  - so the behavior can be studied with truth tables

How It Works (2)

- consider (q,p) the current output values
- and (Q,P) the next output values
  - depending on (q,p) and on inputs (R,S)
  - they will become current values after the propagation delays

Karnaugh Diagram

outputs: QP

qp\RS	00	01	11	10
00	11	10	00	01
01	01	00	00	01
11	00	00	00	00
10	10	10	00	00

$$Q = \overline{p + R}$$

$$P = \overline{q + S}$$

Stable States

- in principle, (Q,P) changes permanently
- but when (Q,P)=(q,p), we have a stable state
  - we want to find these stable states
  - the circuit can be controlled if passing only through stable states

qp\RS	00	01	11	10
00	11	10	**00**	01
01	**01**	00	00	**01**
11	00	00	00	00
10	**10**	**10**	00	00

Functioning (1)

initial state (q,p)	evolution (q,p)	conclusion
R=0, S=1		
00	00 → 10 stable	the circuit evolves towards the stable state (Q,P)=(1,0) in all cases
01	01 → 00 → 10 stable	
10	10 stable	
11	11 → 00 → 10 stable	
R=1, S=0		
00	00 → 01 stable	the circuit evolves towards the stable state (Q,P)=(0,1) in all cases
01	01 stable	
10	10 → 00 → 01 stable	
11	11 → 00 → 01 stable	

Functioning (2)

initial state (q,p)	evolution (q,p)	conclusion
R=0, S=0		
00	00 → 11 → 00 → ...	for q=p, the circuit oscillates indefinitely for q≠p, the circuit holds the state (stable)
01	01 stable	
10	10 stable	
11	11 → 00 → 11 → ...	
R=1, S=1		
00	00 stable	the circuit evolves towards the stable state (Q,P)=(0,0) in all cases
01	01 → 00 stable	
10	10 → 00 stable	
11	11 → 00 stable	

Functioning (3)

- we remind the condition $P=\overline{Q}$

(R,S)=(0,0): keep the current state (memorization)

(R,S)=(0,1): stabilize at Q=1 (set)

(R,S)=(1,0): stabilize at Q=0 (reset)

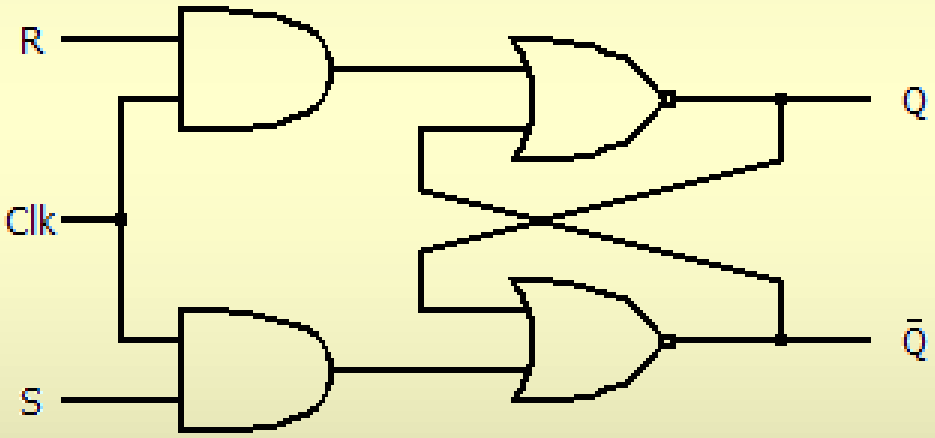
(R,S)=(1,1): forbidden combination

  – because P=Q - does not implement a bit

Synchronous Sequential Circuits

- add a synchronization signal (clock) to the RS bistable circuit
- starting from which other bistable circuits can be designed
  - D, JK, T
- all are latches (level-active)

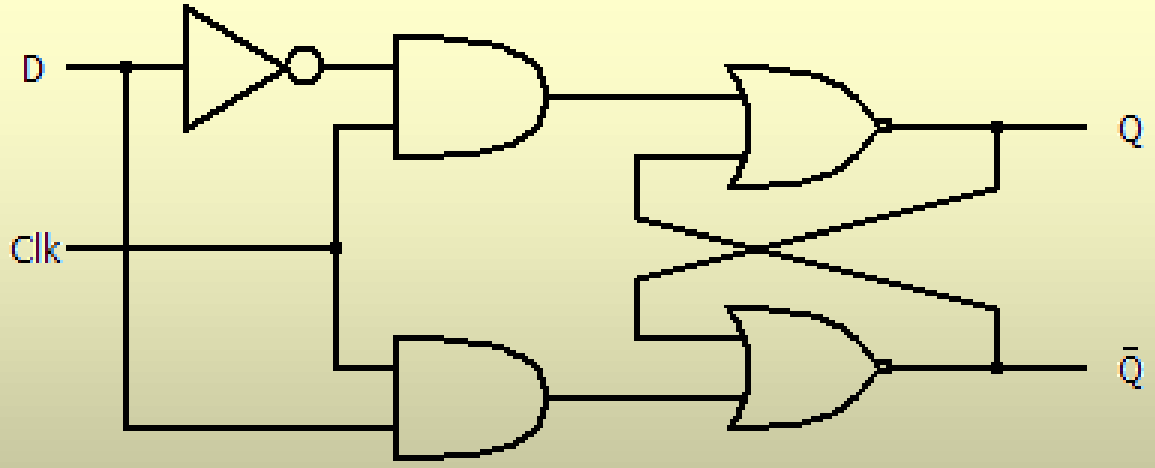
RS Latch



$q(=Q_t)$	R	S	Q_{t+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	*
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	*

D Latch

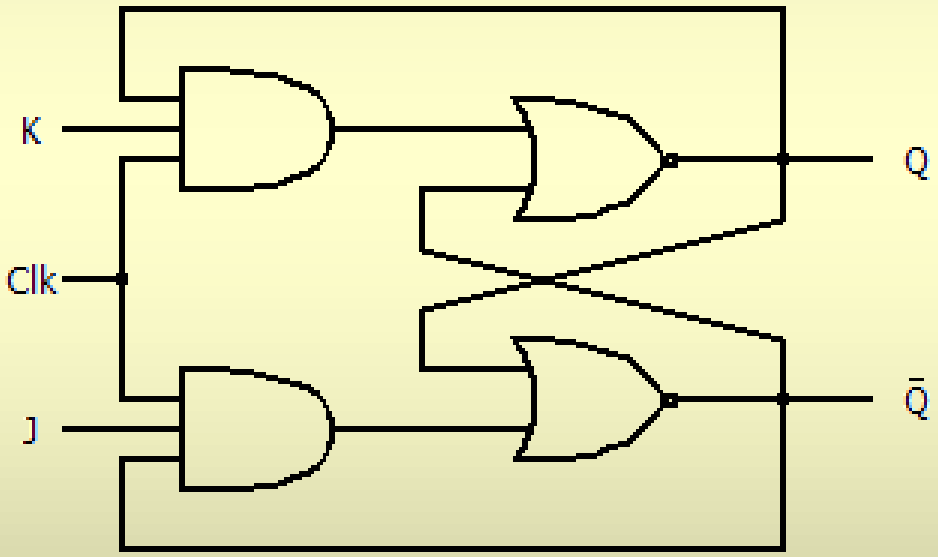
- models only R≠S situations
- eliminates the forbidden combination
- with this circuit, the current state does not actually depend on the previous one



$q(=Q_t)$	D	Q_{t+1}
0	0	0
0	1	1
1	0	0
1	1	1

JK Latch

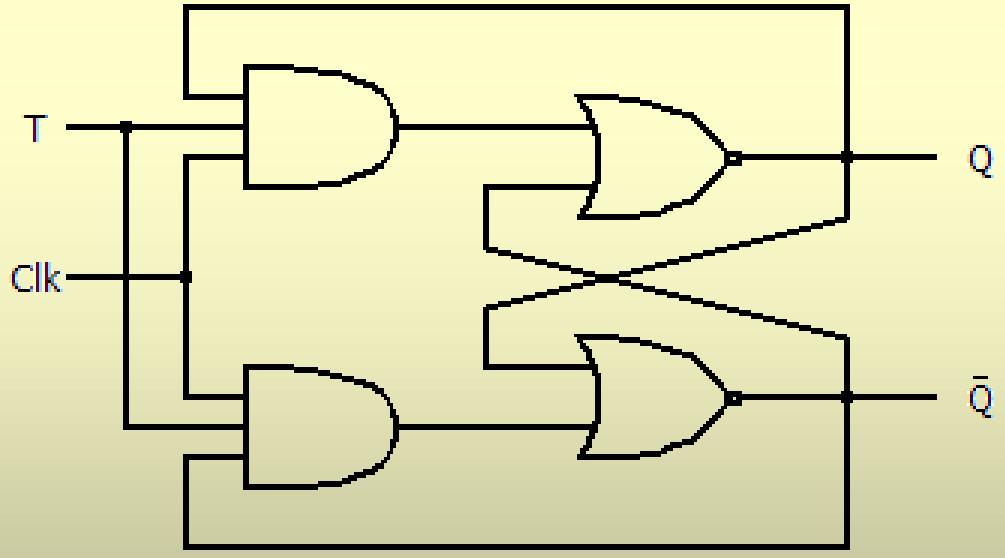
- eliminates the RS forbidden combination



$q(=Q_t)$	J	K	Q_{t+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

T Latch

- derived from the JK latch
- models only J=K situations



$q(=Q_t)$	T	Q_{t+1}
0	0	0
0	1	1
1	0	1
1	1	0

State Evolution

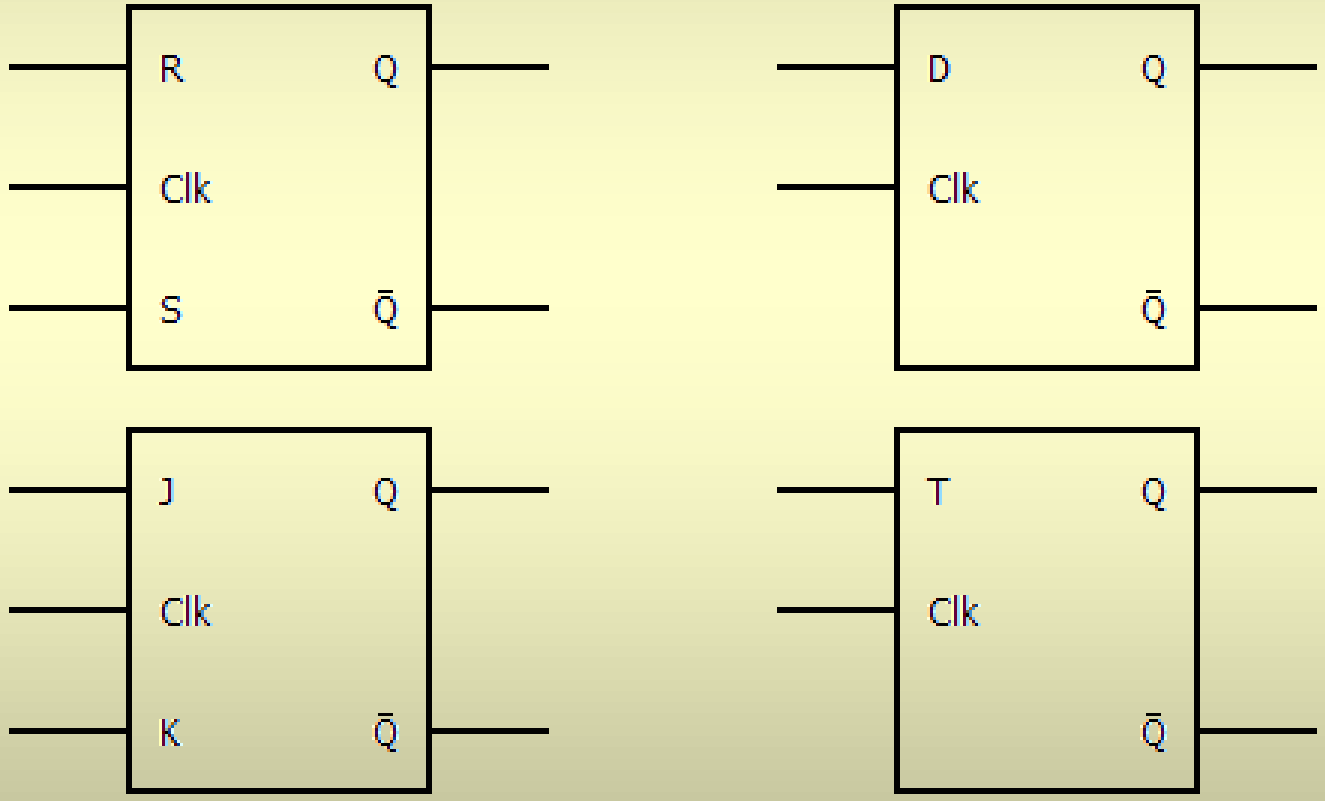
R	S	Q_{t+1}	
0	0	Q_t	unchanged
0	1	1	write 1
1	0	0	write 0
1	1	*	forbidden

J	K	Q_{t+1}	
0	0	Q_t	unchanged
0	1	0	write 0
1	0	1	write 1
1	1	$\overline{Q_t}$	invert

D	Q_{t+1}	
0	0	write 0
1	1	write 1

T	Q_{t+1}	
0	Q_t	unchanged
1	$\overline{Q_t}$	invert

Block Diagrams for Latches



Homework

- implement and analyze the behavior of the no-clock RS circuit using NAND gates instead of the NOR gates
- the same for the RS, D, JK, T latches