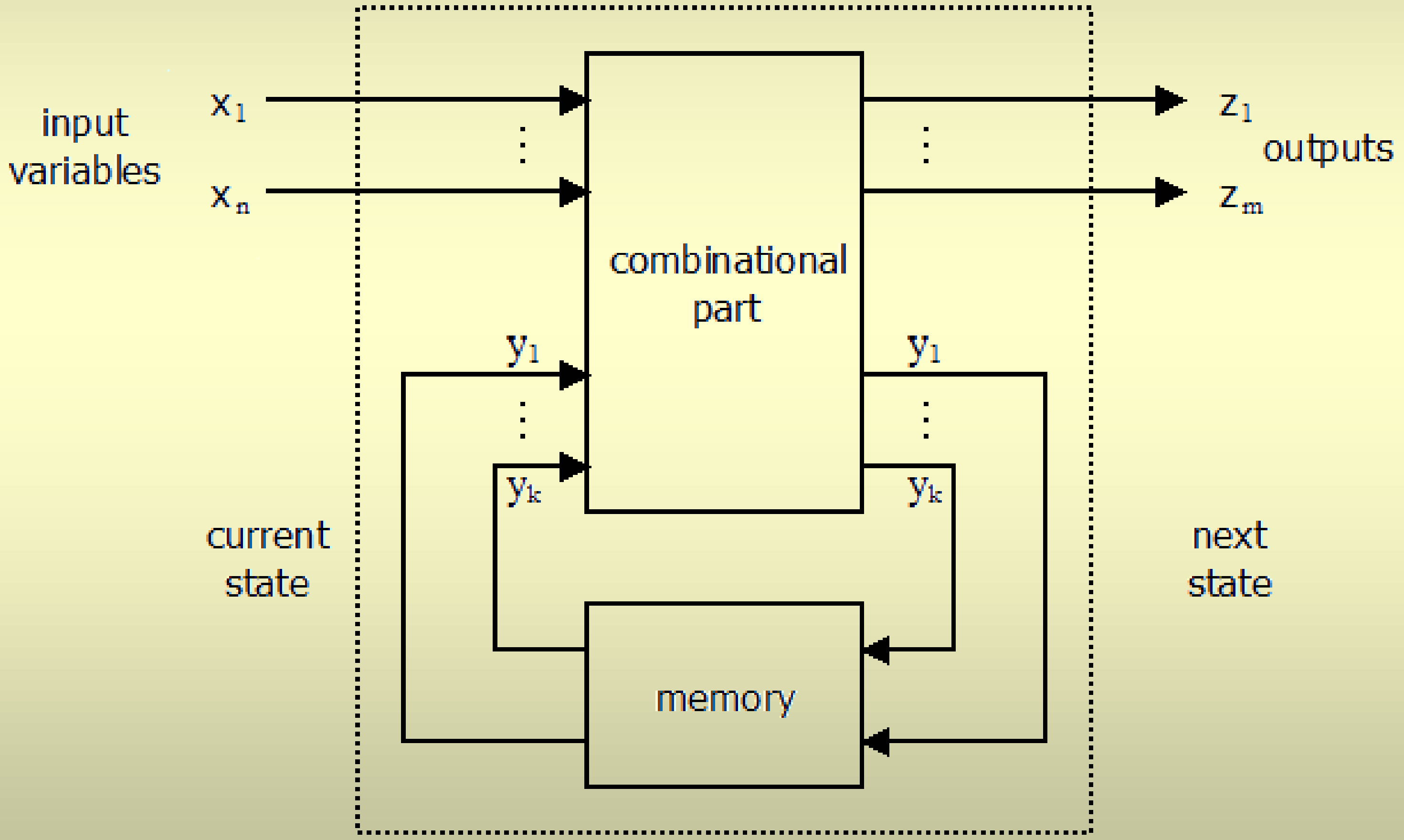
III. Sequential Circuits

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

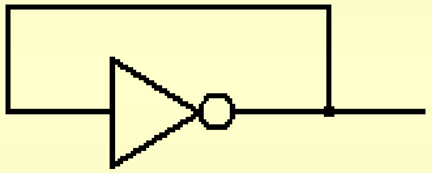
- the state changes at certain moments
  - synchronous: at regular time moments
    - provided by a special signal (clock)
  - asynchronous: when an event occurs
    - events are defined depending on the circuit's activity
  - why doesn't it change permanently?
    - sending the signals through gates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

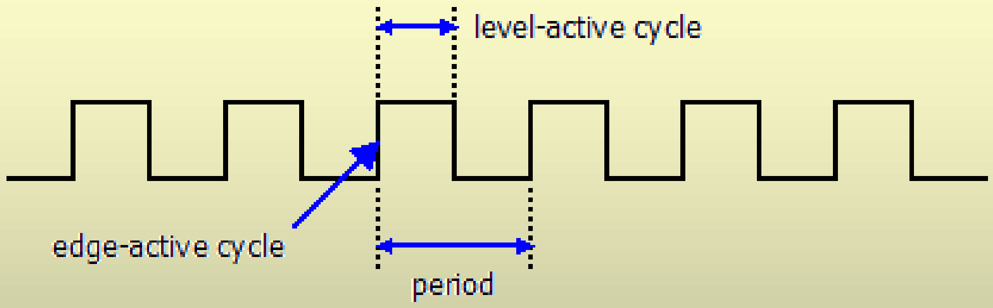
The Clock

- 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

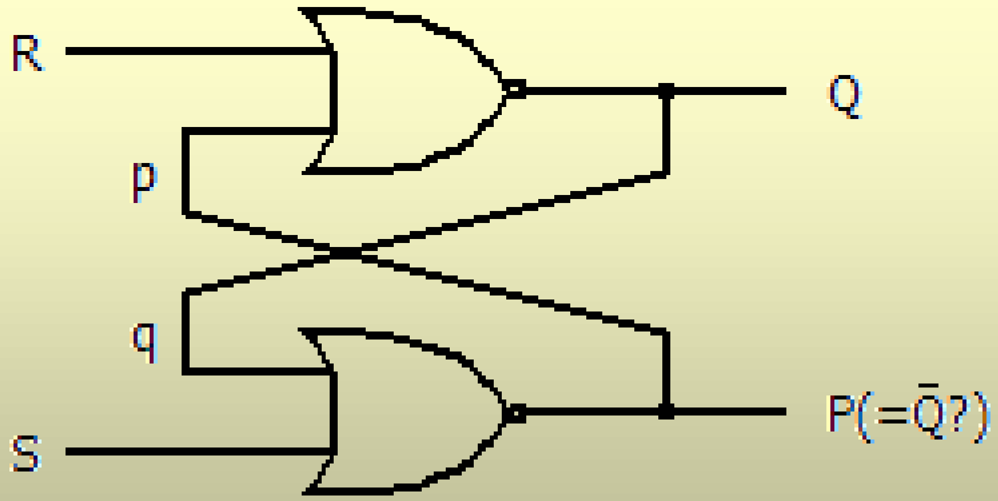
III.1. Bistable Circuits

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)

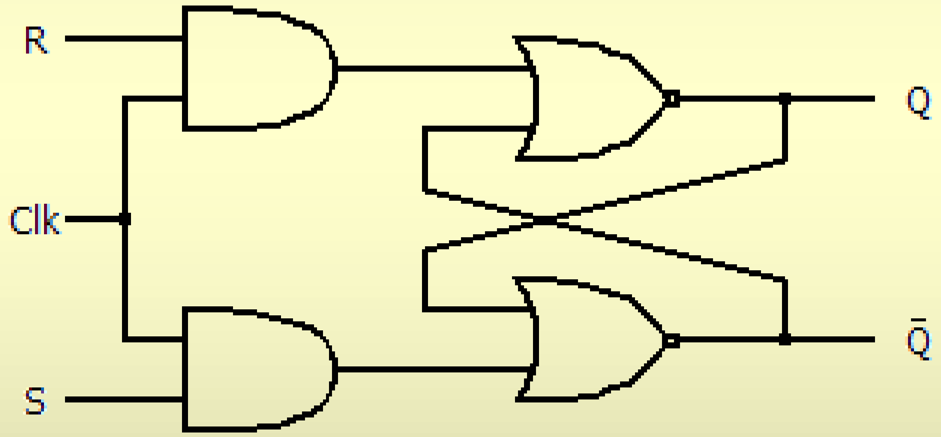
(R,S)=(1,1): forbidden combination

  – 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)

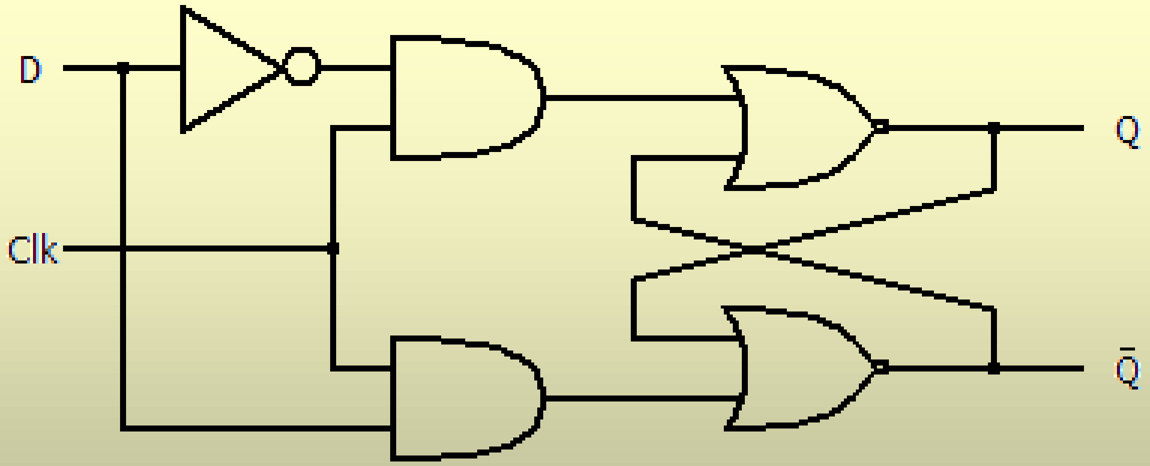
RS Latch



$q(=Q_t)$	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

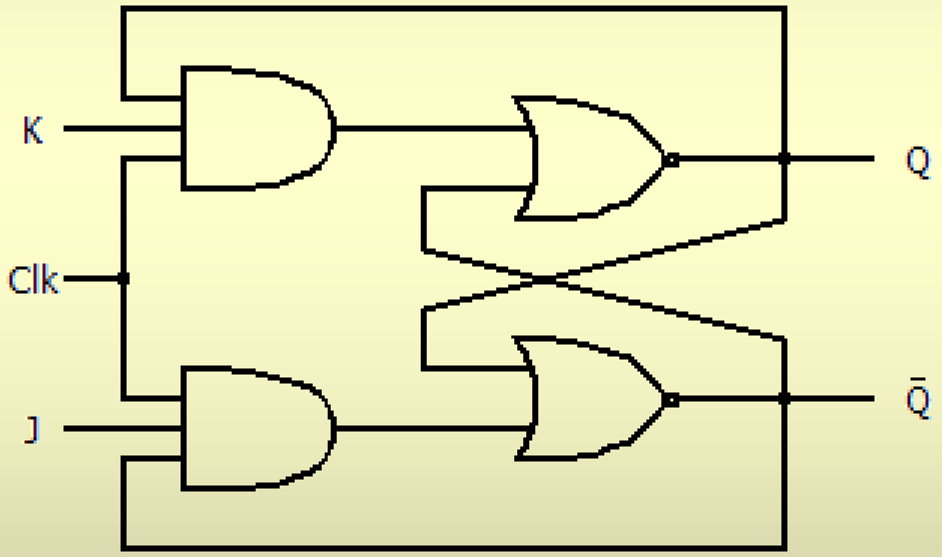
- models only R≠S situations
- eliminates the forbidden combination
- with this circuit, the current state does not actually depend on the previous one



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

JK Latch

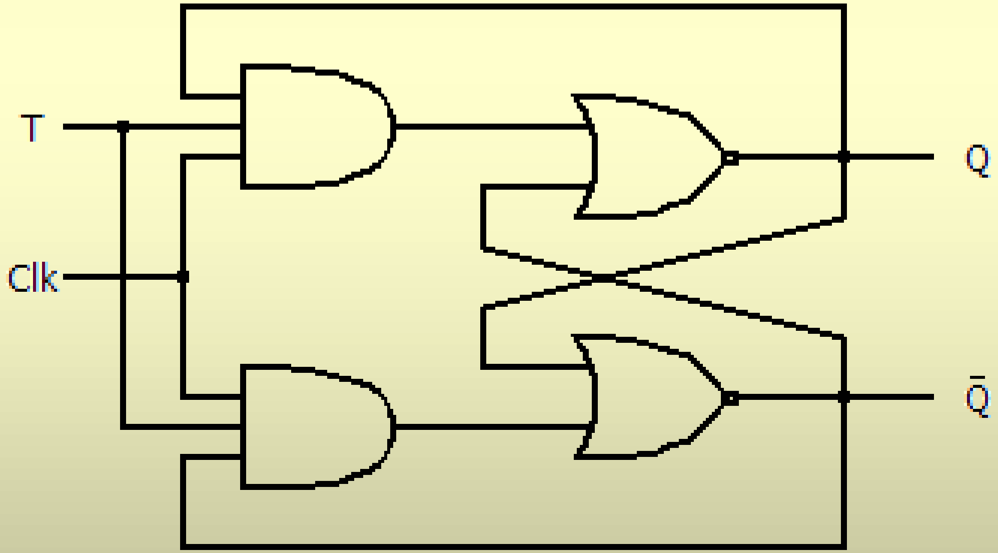
- eliminates the RS forbidden combination



$q(=Q_t)$	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

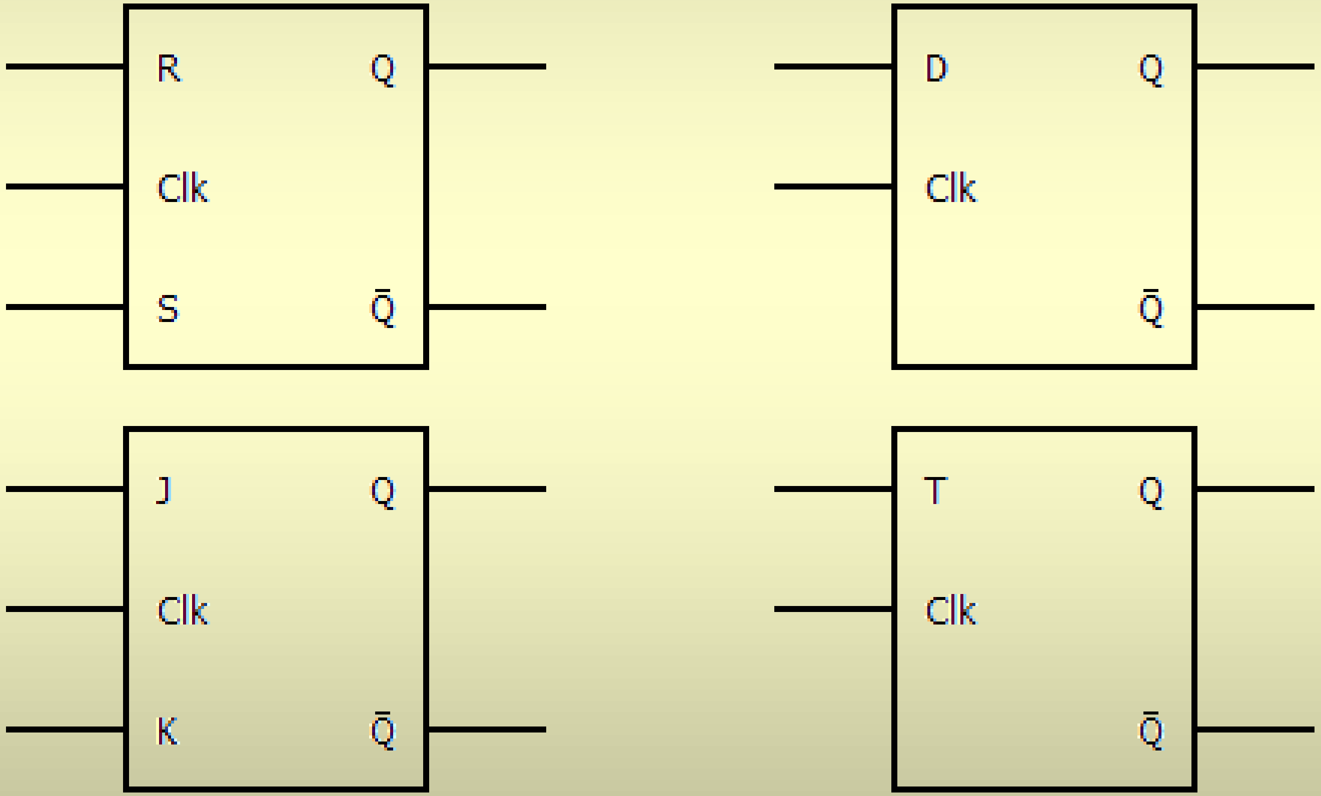
R	S	Q_{t+1}	
0	0	Q_t	unchanged
0	1	1	write 1
1	0	0	write 0
1	1	*	forbidden

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



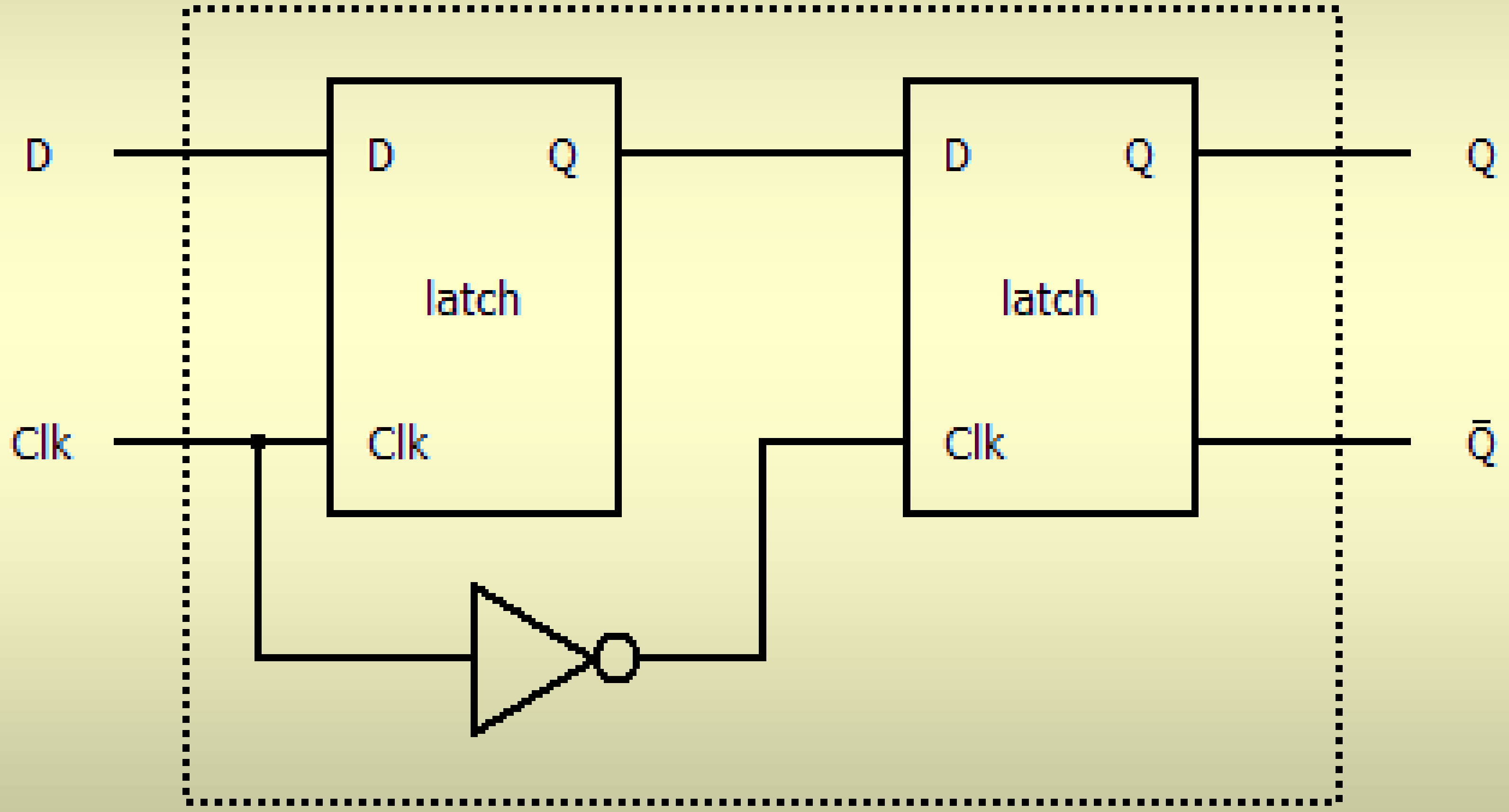
Homework

- 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

Flip-flop

- inputs are considered on the rising (or falling) edge of the clock signal
- ways of making a flip-flop
  - electronics - derive the clock signal
    - results in an impulse-like signal
  - use latches → master-slave circuits

Master-slave D Flip-flop



Latch vs. Flip-flop

- each category has its utility
- flip-flops - used for controlling digital systems
  - the edge of the clock signal is very short compared with the clock period
    - i.e., it can be considered as a moment
  - during each clock period, the system makes exactly one step of its evolution
- latches - asynchronous systems

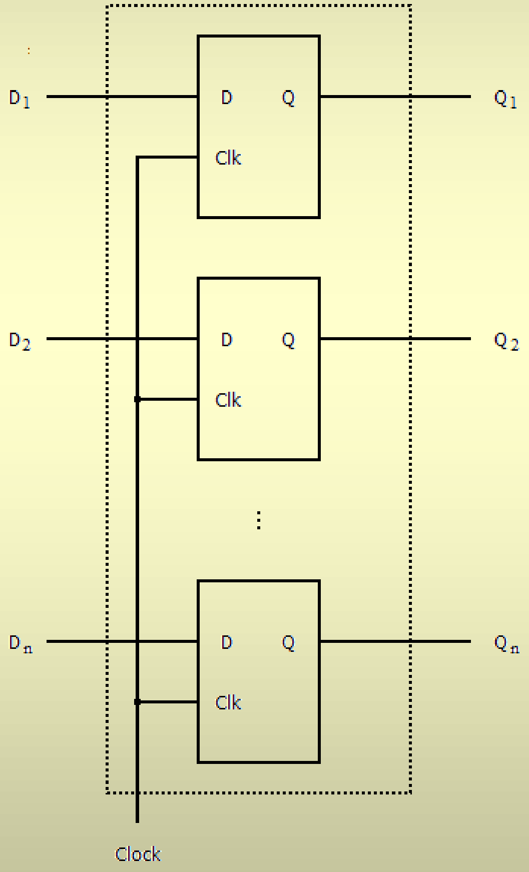
III.2. Complex Sequential Circuits

Registers

- a bistable circuit implements a single bit
  - not very useful in practice
- we can use several bistable circuits together
  - all receive the same command at the same time
  - such a circuit is called register
- types of registers
  - parallel registers
  - shift (serial) registers

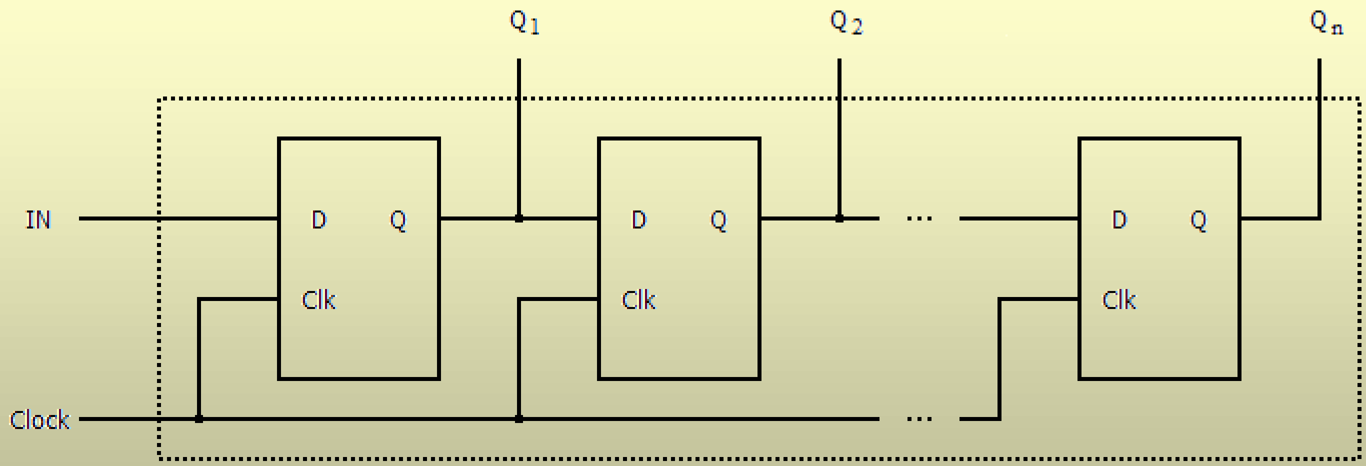
Parallel Register

- implementation with D bistable circuits
  - can be latches or flip-flops, as needed
- the same command (clock)
  - all bits change at the same moments
- natural extension of the bistable circuit

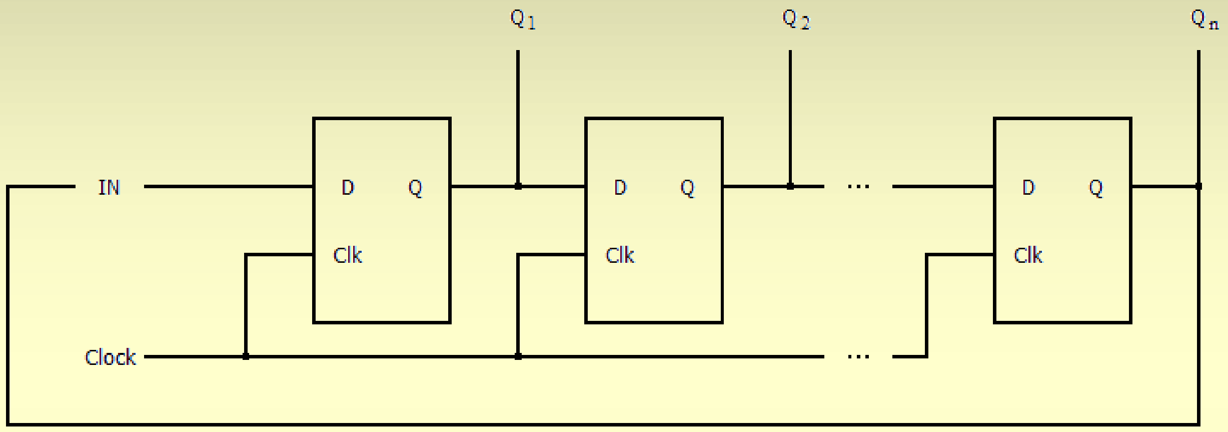


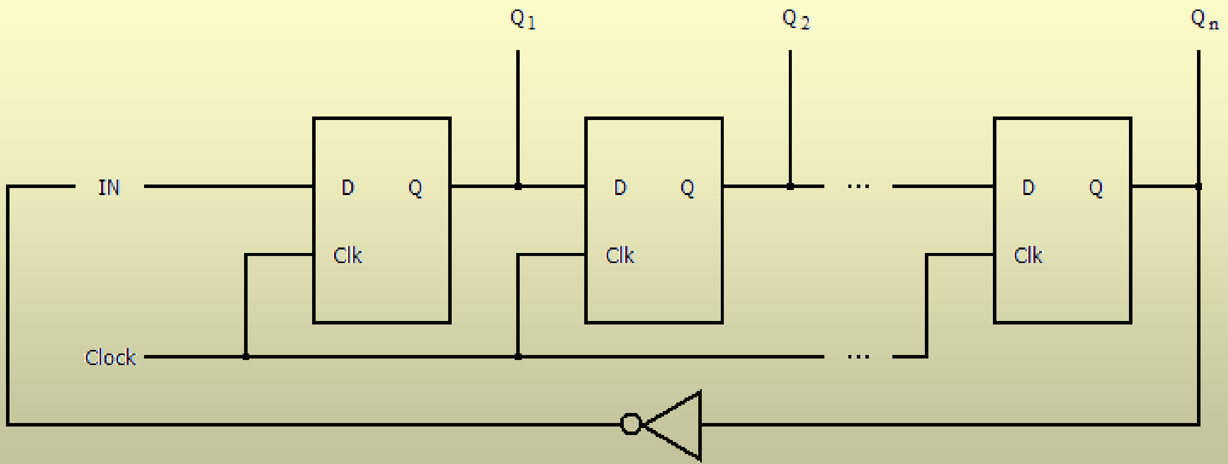
Classic Shift Register

- memorizes the last *n* values applied on the input
- can be implemented only with flip-flops
  - homework: why?



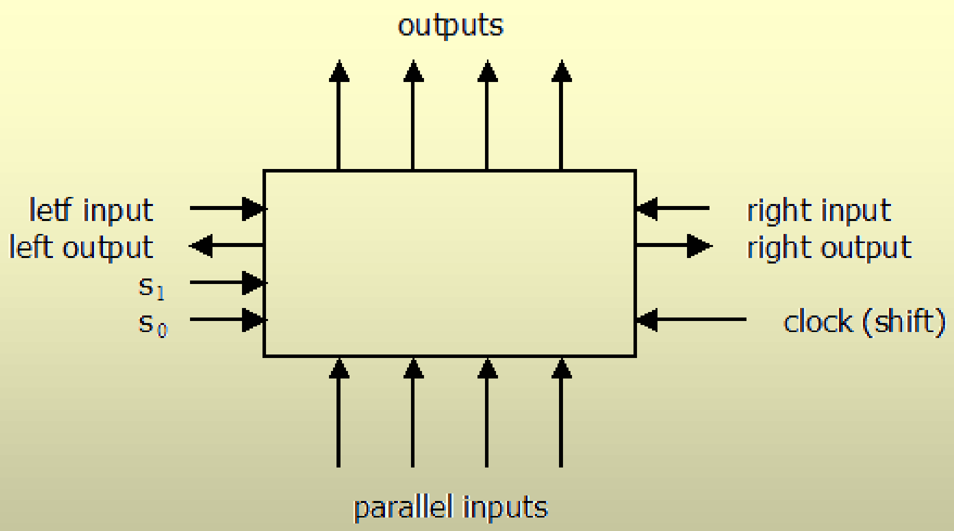
Other Shift Registers





Universal Shift Register

- serial or parallel inputs and outputs
- right or left shift operations
- one can use any of the features above, as needed



s_0	s_1	function
0	0	unchanged
0	1	shift right
1	0	shift left
1	1	parallel load

Designing a Sequential Circuit (1)

- finite state machine (automaton)

1. determine the states of the circuit
2. determine the state transitions
   - how the next state and the outputs depend on the inputs and the current state
3. state encoding
   - using the necessary number of bits
4. write the truth table for the state transitions

Designing a Sequential Circuit (2)

5. minimization
6. implementation
   - the state - memorized by flip-flops
   - combinational part - from the minimization
     - the inputs of the combinational part (current state) are collected from the outputs of the flip-flops and the input variables
     - the outputs of the combinational part (next state) are applied at the inputs of the flip-flops

Binary Counter

- at each moment keeps an *n*-bit number
- at each clock "tick" - incrementation
  - could also be decrementation
  - after the maximum value, 0 comes next
  - no inputs, only state variables
    - which keep the current value of the number
  - outputs are identical to the state variables

Example: *n*=4

current state				next state				current state				next state			
q_3	q_2	q_1	q_0	d_3	d_2	d_1	d_0	q_3	q_2	q_1	q_0	d_3	d_2	d_1	d_0
0	0	0	0	0	0	0	1	1	0	0	0	1	0	0	1
0	0	0	1	0	0	1	0	1	0	0	1	1	0	1	0
0	0	1	0	0	0	1	1	1	0	1	0	1	0	1	1
0	0	1	1	0	1	0	0	1	0	1	1	1	1	0	0
0	1	0	0	0	1	0	1	1	1	0	0	1	1	0	1
0	1	0	1	0	1	1	0	1	1	0	1	1	1	1	0
0	1	1	0	0	1	1	1	1	1	1	0	1	1	1	1
0	1	1	1	1	0	0	0	1	1	1	1	0	0	0	0

Example: *n*=4

- by minimization we get the equations below

$$d_0 = \overline{q_0} = q_0 \oplus 1$$

$$d_1 = \overline{q_1} \cdot q_0 + q_1 \cdot \overline{q_0} = q_1 \oplus q_0$$

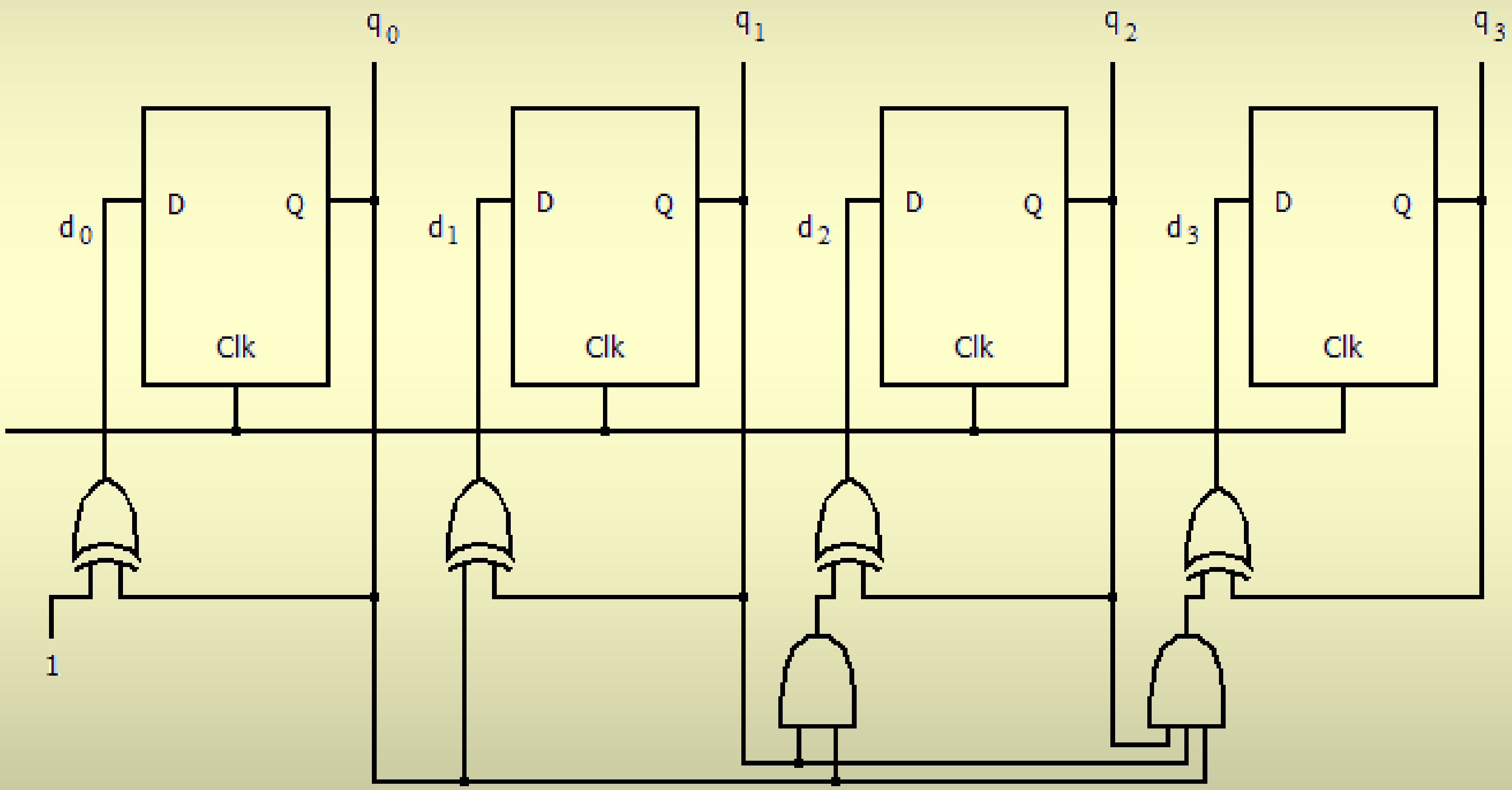
$$d_2 = \overline{q_2} \cdot q_1 \cdot q_0 + q_2 \cdot \overline{q_1} + q_2 \cdot \overline{q_0} = q_2 \oplus (q_1 \cdot q_0)$$

$$d_3 = \overline{q_3} \cdot q_2 \cdot q_1 \cdot q_0 + q_3 \cdot \overline{q_2} + q_3 \cdot \overline{q_1} + q_3 \cdot \overline{q_0} =$$

$$= q_3 \oplus (q_2 \cdot q_1 \cdot q_0)$$

– state implementation - D flip-flops

Implementation



Microprogramming (1)

- alternative implementation technique
  - the state is still memorized by flip-flops
  - combinational part - implemented by a ROM circuit
  - the inputs of the Boole functions are applied to the address inputs of the ROM
  - the outputs of the Boole functions are collected from the data outputs of the ROM

Microprogramming (2)

- implementation of the combinational part
  - start from the truth table
  - to each location - write the desired output values
- advantage - flexibility
  - any change of the automaton requires only the rewriting of the contents of the ROM
- drawback - low speed
  - ROM circuits are slower than logic gates

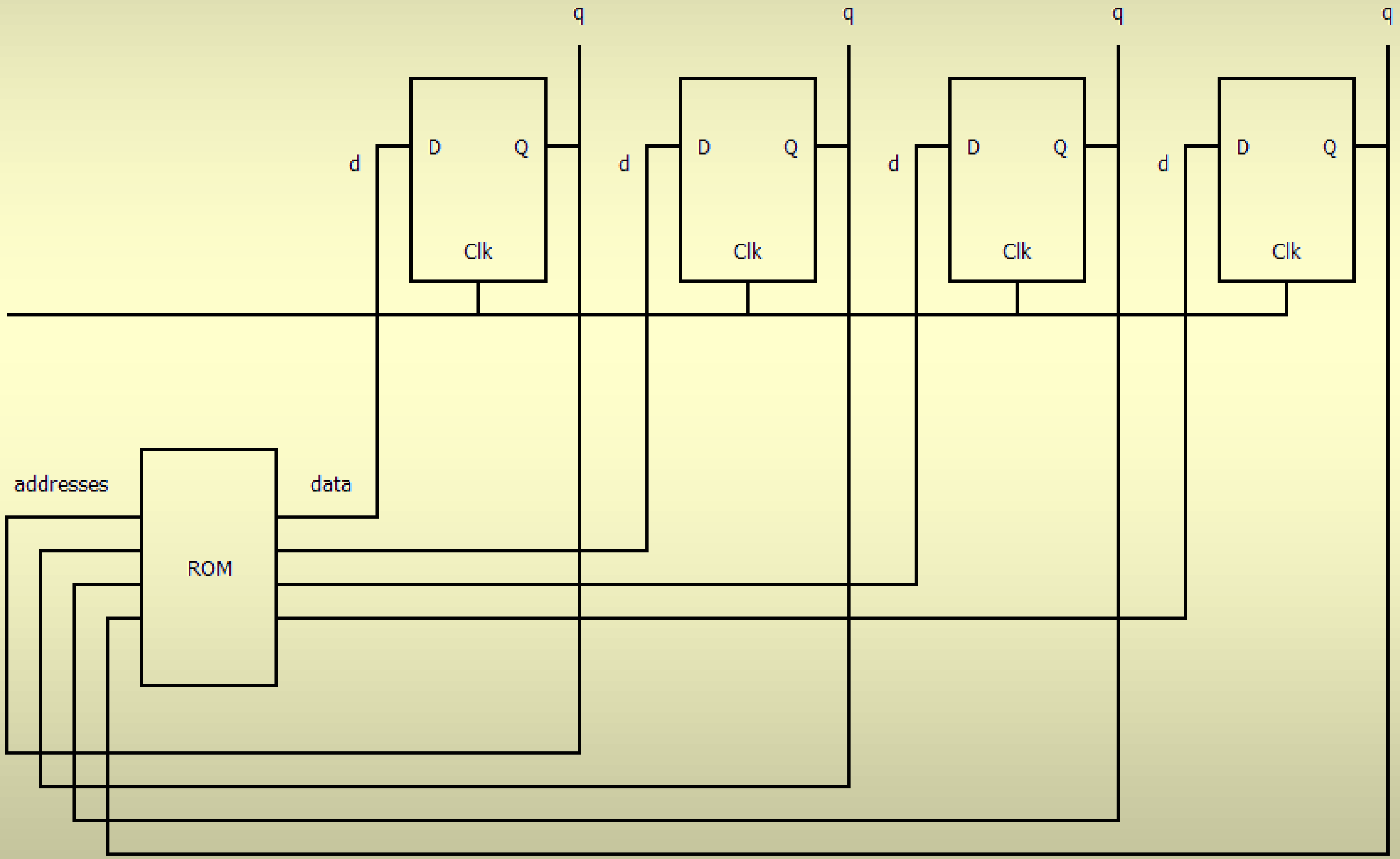
The Same Example

- there are 16 (= 2^4) states
  - encoded with 4 state bits
- so the ROM circuit will have
  - 2^4 addresses $\rightarrow$ 4 address bits
    - 16 locations
  - 4 data bits $\rightarrow$ locations are 4 bits wide
  - in this example there are no inputs and outputs of the system - only state bits

The Contents of the ROM

address	value
0	0 0 0 1
1	0 0 1 0
2	0 0 1 1
3	0 1 0 0
4	0 1 0 1
5	0 1 1 0
6	0 1 1 1
7	1 0 0 0
8	1 0 0 1
9	1 0 1 0
10	1 0 1 1
11	1 1 0 0
12	1 1 0 1
13	1 1 1 0
14	1 1 1 1
15	0 0 0 0

Implementation



IV. Internal Representations

- elementary internal representations
  - they are part of the computer's architecture
  - so they are implemented in hardware
  - directly accessible to the programmers
- more complex data structures
  - based on elementary representations
  - defined and accessible to the programmers by software

Elementary Representations

- numerical data
  - integer/rational numbers
  - only certain subsets of these sets
- alpha-numerical data
  - characters etc.
- instructions
  - the only system-specific representations
  - thus non-standardized and non-portable

Studying the Representations

- numerical representations

  $\mathrm{repr}(n_1)\ \mathit{op}\ \mathrm{repr}(n_2) = \mathrm{repr}(n_1\ \mathit{op}\ n_2)$???
- example - if we add two integer variables, will the result fit into its destination?
- representation errors
  - approximations
  - overflows

Sending the Information

- between various physical media
  - between computers/systems
  - between the components of a computer/system
- transmission errors may occur
  - due to perturbations/incorrect working
  - digital signal - some bits are inverted
  - we wish to detect to occurrence of such errors
  - and even fix them, where possible (correction)

Ways of Detection/Correction

- use additional *redundant* bits
- parity - 1 additional bit
  - allows detecting the occurrence of a (1 bit) error
  - odd/even parity: odd/even number of bits 1
- Hamming code
  - 4 information bits, 3 additional bits
  - detection/correction of multiple errors simultaneously

Example: Odd Parity

- transmitter
  - has to send value $(110)_2$
  - 2 bits of value 1 (even) $\rightarrow$ the additional bit is 1
  - sends $(1101)_2$
- receiver
  - receives the bit string
  - if the number of bits of value 1 is even - error
  - else - eliminate the parity bit and get $(110)_2$