CO221 Logic Networks

Dr. S.D. Dewasurendra

Dept. of Computer Engineering, Faculty of Engineering, UOP

Lecture No. 5

Topic	Time Allocated			
	L	T	P	A
Sequential logic circuits and Memory Elements: SR flip flops, Gated, edge triggered and Master-slave operation, JK flip flop, D flip-flop, T flip-flop, Registers, Serial/Parallel conversion, Codes-Error correction and Detection	6		4	2
Design of Synchronous Seq. circuits: Analysis of Synchronous circuits, Mealy -Moore Networks and Models, State diagrams and state tables State minimization, State assignment, Assignment Rules, Next state and output equation realization, Design of Counters, ROM utilization of Sequential circuits.	5		4	4
Analysis and Design of Asynchronous Seq. circuits: Analysis of Asynchronous circuits, Design Procedure, Flow tables, Reduction of state and flow tables, Race free State assignment, Hazards in asynchronous circuits.	5		4	

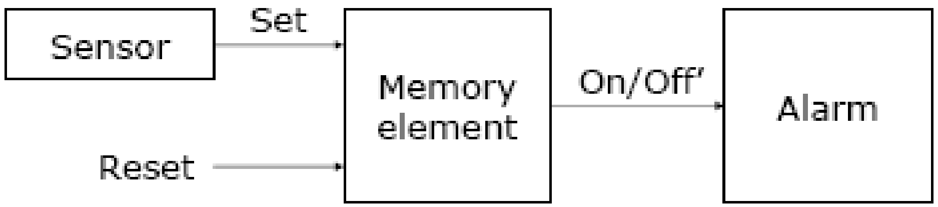
Course Website:
http://www.ce.pdn.ac.lk/cms/co211
23 – 03 - 2020

Motivation

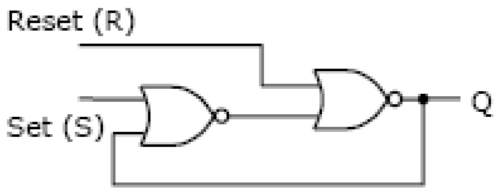
Alarm control system

- Suppose we wish to construct an alarm circuit such that the output remains active (on) even after the sensor input that triggered the alarm goes off
  - A typical car alarm is representative of this type of circuit

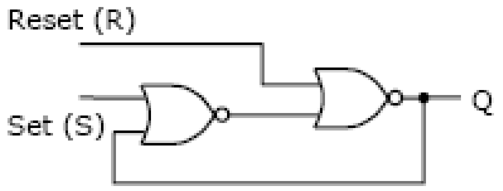
- The circuit requires a **memory element** to remember that the alarm has to be active until a ***reset signal*** arrives



- The following circuit, constructed with NOR gates does just that:

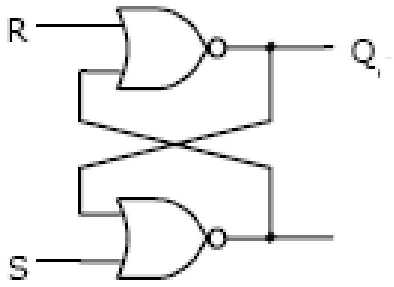


- Inputs, ***Set and Reset, provide the means to*** changing the state, ***Q (Alarm), of the circuit***
- This circuit is referred to as a basic ***latch***



S-R latch operation

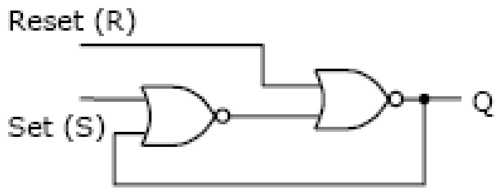
This latch can be redrawn as follows:



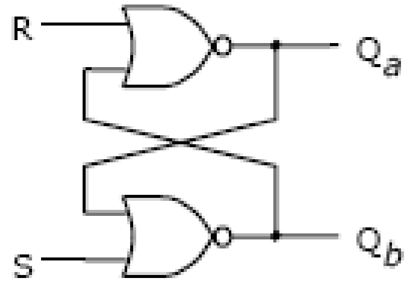
S(t)	R(t)	Q(t)	Q(t+ε)	
0	0	0	0	Memory states
0	0	1	1	
0	1	0	0	
0	1	1	0	
1	0	0	1	
1	0	1	1	
1	1	0	-	Uncertain outputs
1	1	1	-	

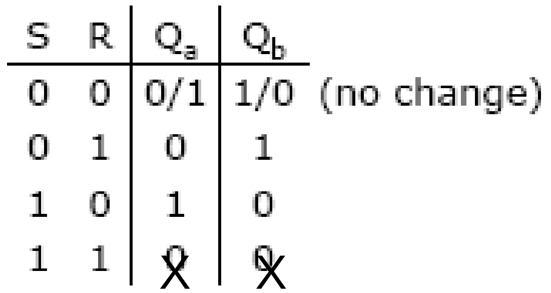
- Notion of state
- Karnaugh map for Next State ->
- Characteristic Equation of the latch
- A sequential function

Basic SR Latch



The latch can be redrawn thus,

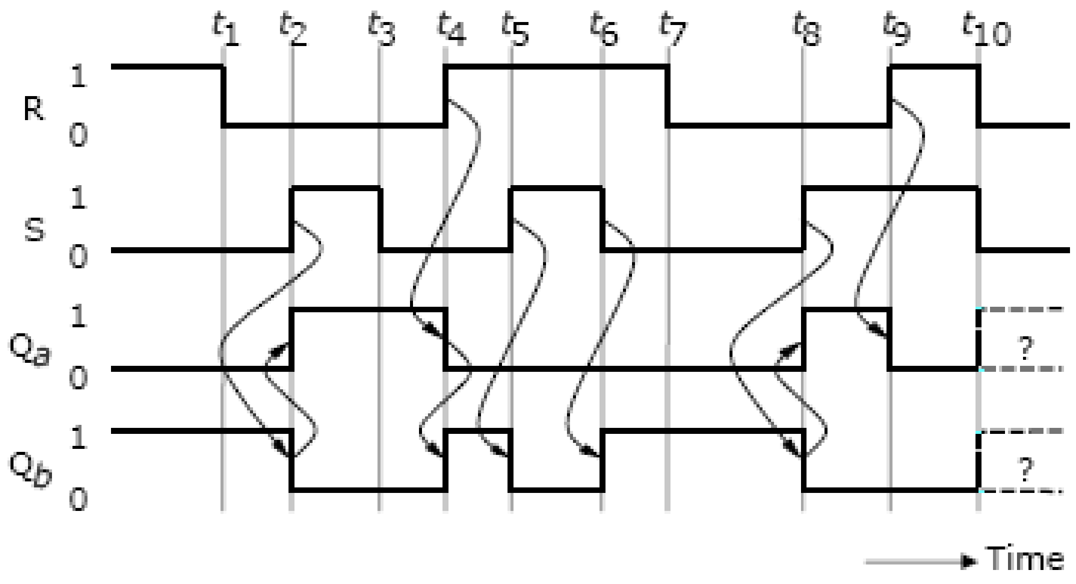




S	R	Q_a	Q_b	
0	0	0/1	1/0	(no change)
0	1	0	1	
1	0	1	0	
1	1	~~0~~ X	~~0~~ X	

Basic SR latch timing diagram

Characteristic Table



- When S=1 and R=1,
  Q_a=Q_b=0
  (there are actually problems with this state as we will see)

- If the ***propagation delays from Q_a and Q_b*** are exactly the same, the oscillation at time t_{10} would continue indefinitely

- In a real circuit there would probably be some (mostly insignificant) difference in the delays and the latch would eventually settle into one of its two stable states (but we don't know which one it would be)

- Thus the S=R=1 combination is generally considered an ***unallowed combination in the*** SR latch

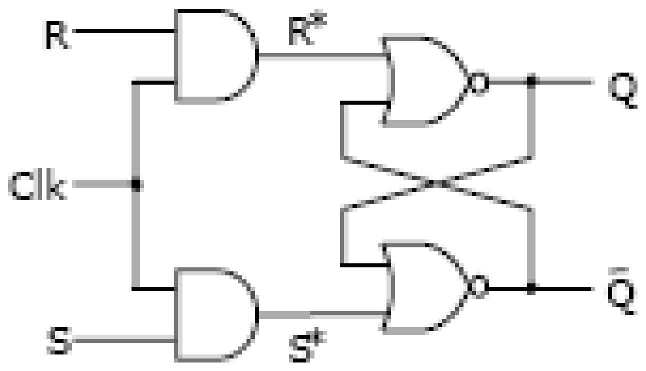
Gated SR latch

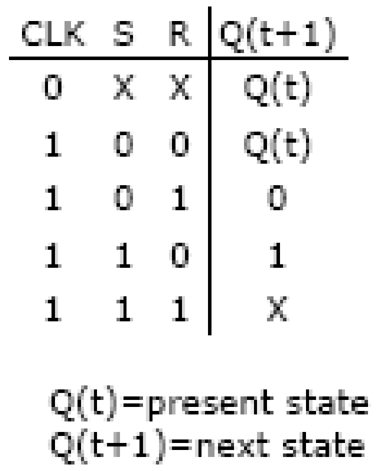
• The basic SR latch changes its state whenever its inputs change

• It may be desirable to add an enable signal to the basic SR latch that allows

us to control when the circuit can change states

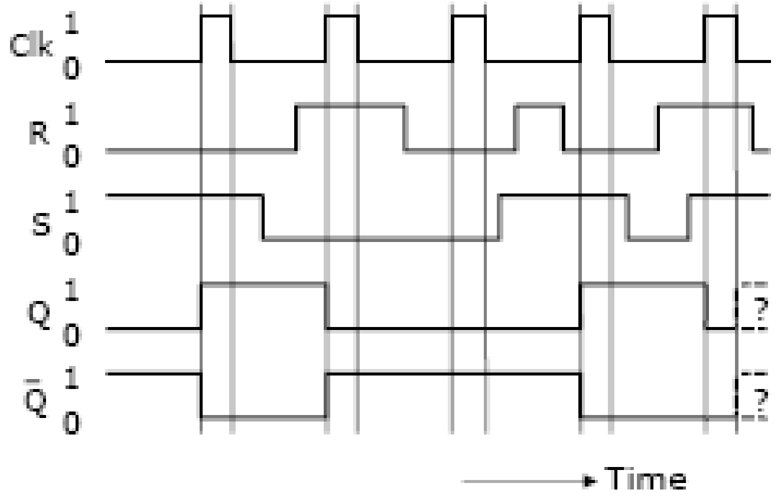
• Such a circuit is referred to as a ***gated SR latch***



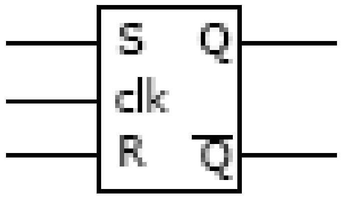


CLK	S	R	Q(t+1)
0	X	X	Q(t)
1	0	0	Q(t)
1	0	1	0
1	1	0	1
1	1	1	X

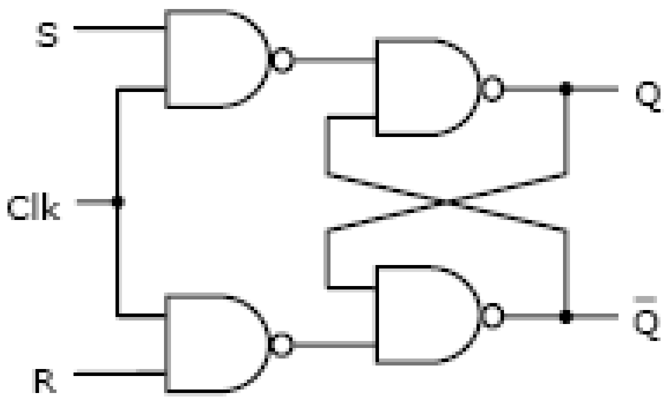
Q(t)=present state
Q(t+1)=next state



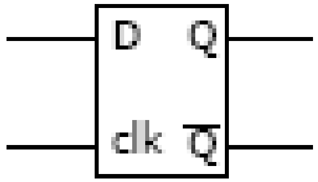
Graphical symbol for Gated SR latch



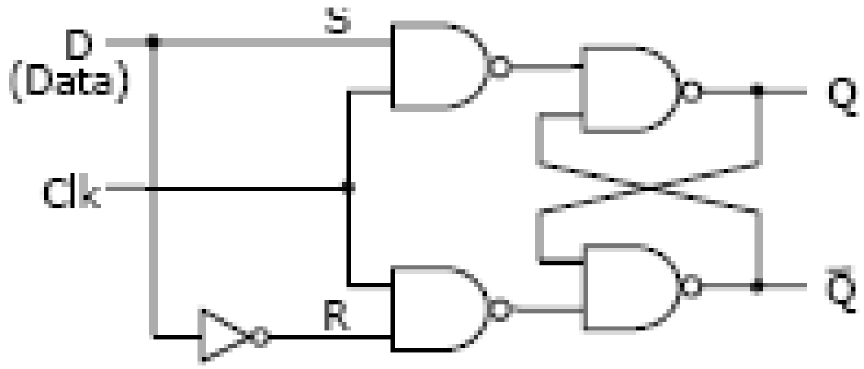
Gated SR latch with NAND gates

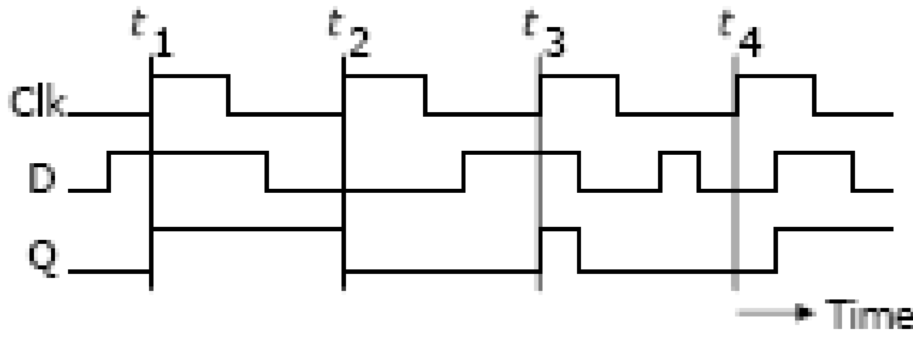


Gated D latch



CLK	D	Q(t+1)	
0	X	Q(t)	-memory
1	0	0	-follow D
1	1	1	



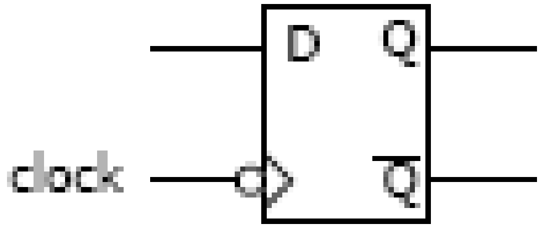


- Another useful latch has a single data input, **D**, and it stores the value of this input under the control of a clock signal
- This is referred to as a ***gated D latch***
  - Useful in circuits where we want to store some value
  - The output of an adder/subtractor circuit would be one example

Level versus edge sensitivity

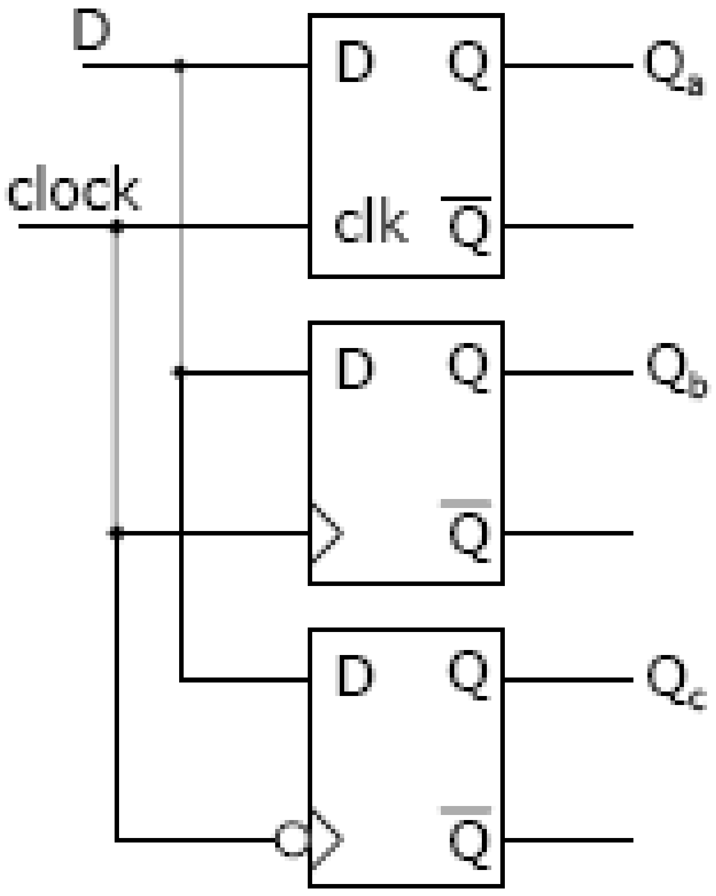
- Since the output of the D latch is controlled by the level (0 or 1) of the clock input, the latch is said to be ***level sensitive*:** they can change states **more than once** during the 'active' period of the clock signal
  - All of the latches we have seen up to now were **level sensitive**
- It is possible to design a storage element for which the output only changes at the point in time when the clock changes from one value to another: **so that it can change its state no more than once during a clock period**
- Such circuits are said to be ***edge triggered***

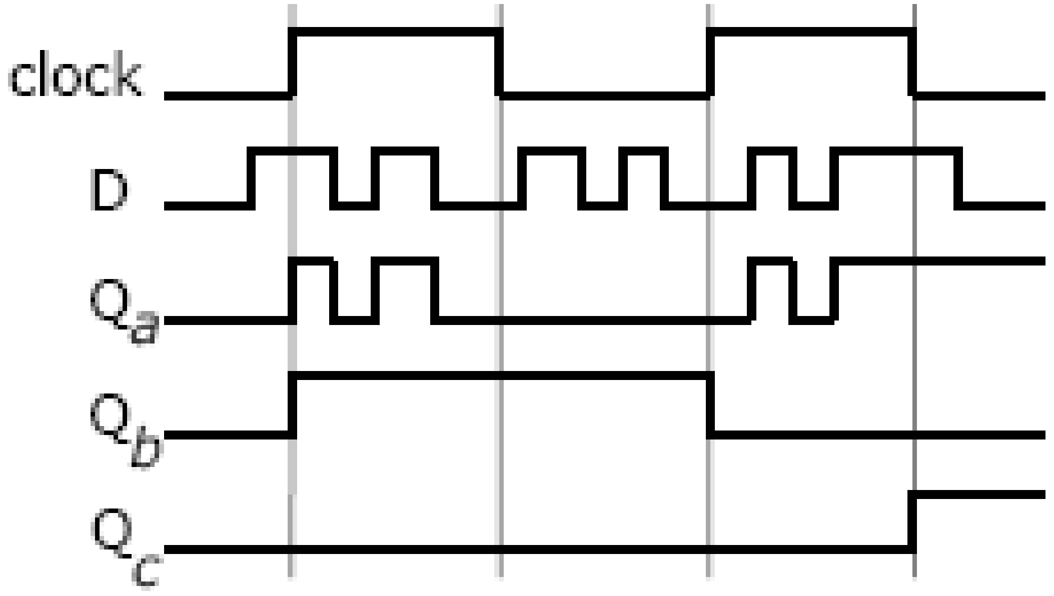
D flip-flop



- Two types of circuits with such behaviour
  - Master-slave **flip-flop**
  - Edge-triggered **flip-flop**

Comparing D storage elements

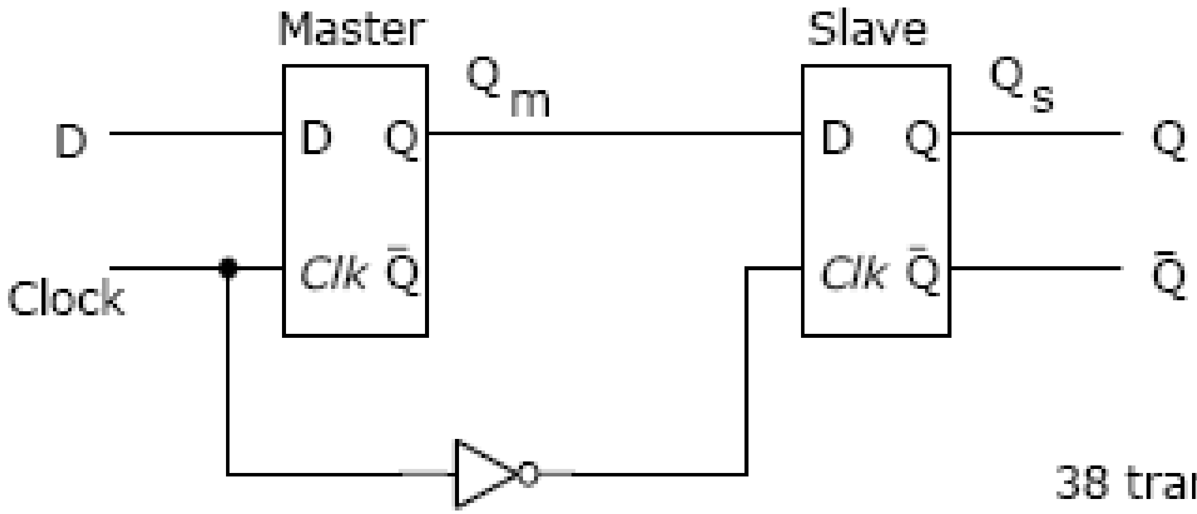




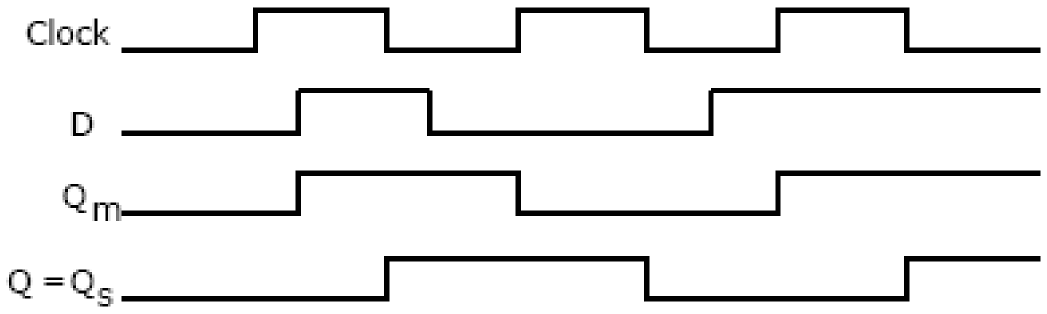
Characteristic eqn of edge-triggered D Flip-flop: $\mathbf{Q^+ = D}$

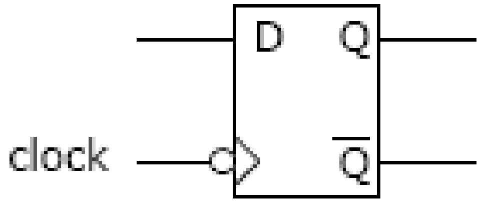
Constructing an Edge-triggered D Flip-flop

Master-slave configuration using 2 gated D latches and one inverter
- – The first D latch, the ***master***, changes its state while clock = 1
- – The second, the ***slave***, changes its state while clock = 0



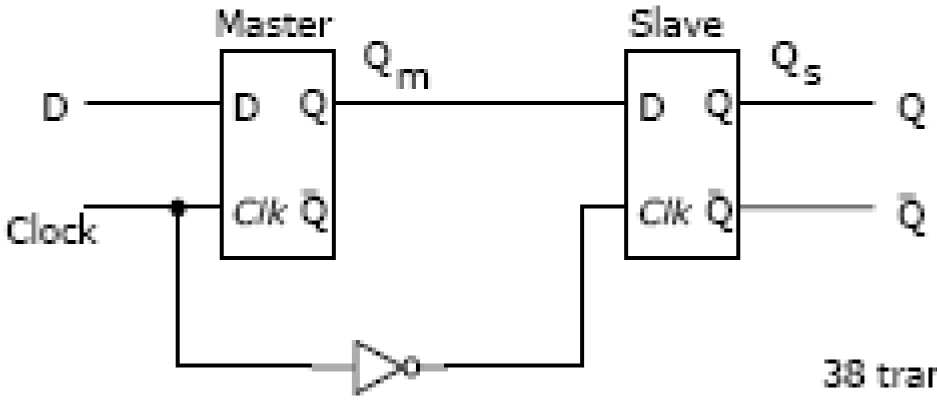
38 transistors

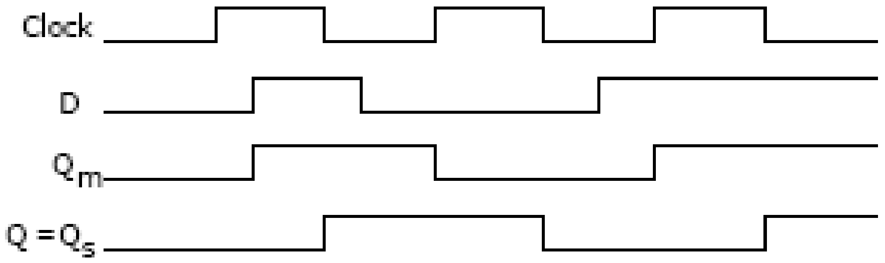




Falling Edge-triggered D Flip-flop

Master-slave D flip-flop ctd.





38 transistors

• When clock=1, the master tracks the values of the D input signal and the slave does not change

– Thus Q_m follows any changes in D and Q_s remains constant

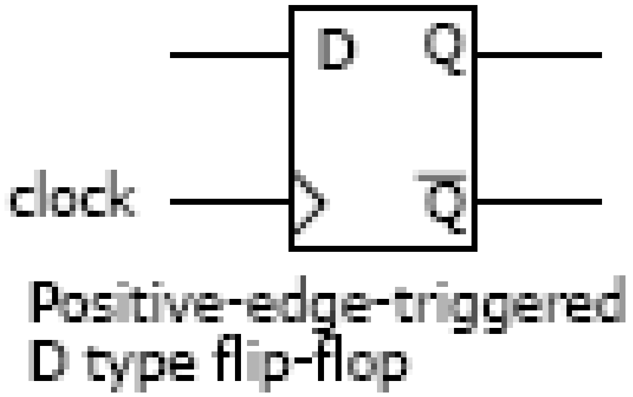
• When the clock signal changes to 0, the master stage stops following the changes in the D input signal

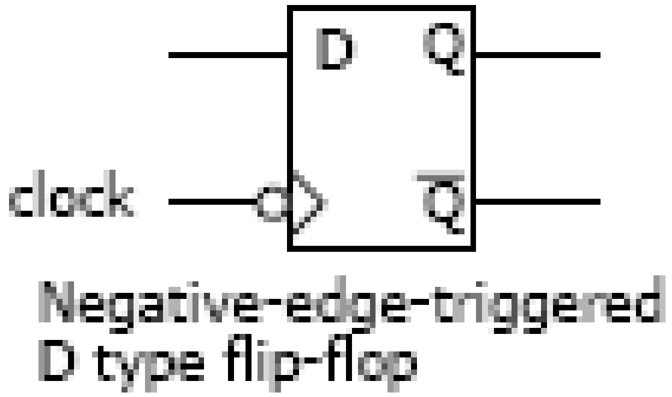
- At the same time, the slave stage responds to the value of Qm and changes states accordingly

• Since Q_m does not change when clock=0, the **slave stage undergoes at most one change of state during a clock cycle**

• From an output point of view, the circuit changes Q_s (its output) at the ***negative edge*** ***o***f the clock signal

- The previous circuit responds on the falling-edge of the clock signal
- A rising-edge triggered D flip-flop can be constructed by moving the inverted clock input to the master

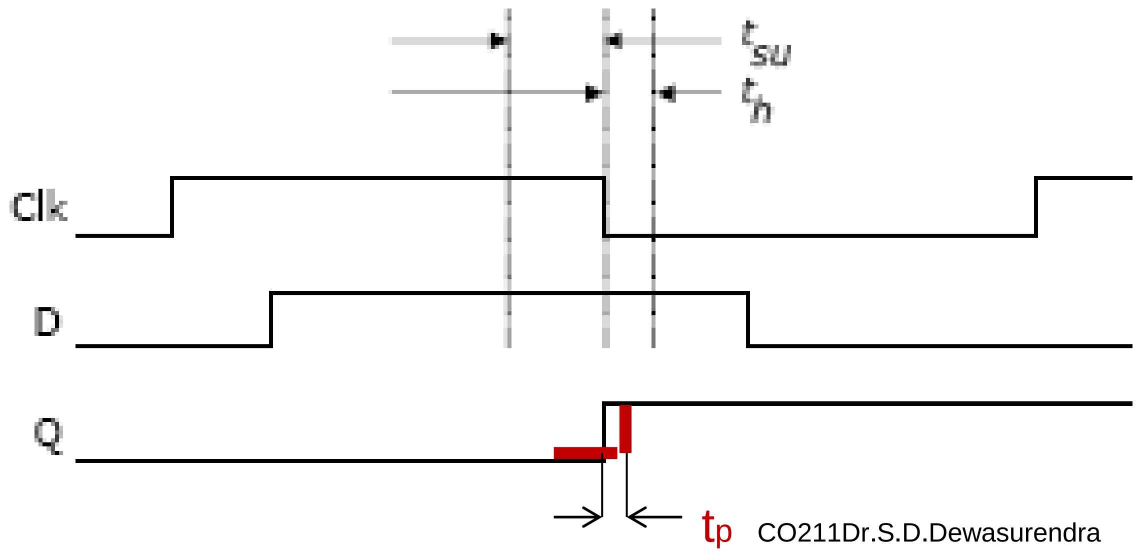




We can use the same strategy for constructing SR, J-K and T Flip-flops

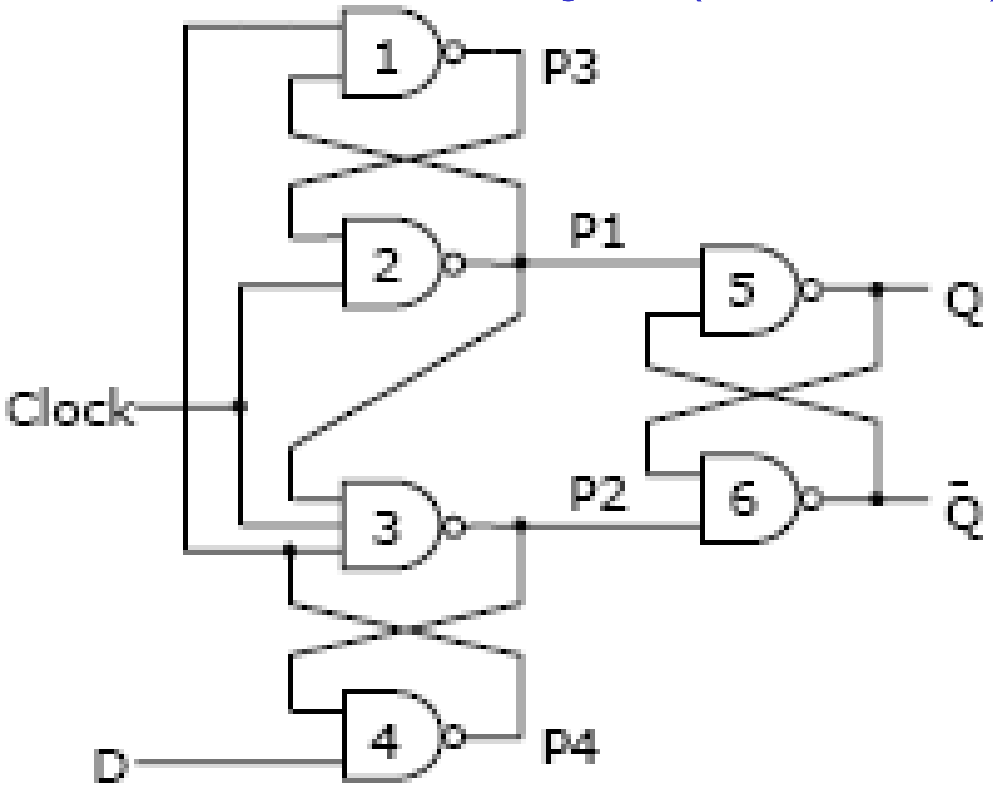
Effects of propagation delays – setup and hold times

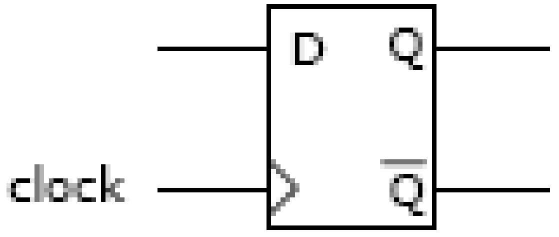
- Previously we have ignored the effects of propagation delay. In practical circuits, it is essential to account for these delays
- For the gated D latch (and others as well), it is important that the value of D does not change at the time the active edge of the clock, clk, is changing from 1 to 0 (or 0 to 1)
  - The designer must make sure the signal is stable when the critical change in the clock takes place
- The minimum time the D signal must remain stable prior to the negative edge (1->0) of the clock signal is called the ***setup time*** (t_{su})
- The minimum time the D signal must remain stable after the negative edge of clock is the ***hold time*** (t_h)
  - Typical CMOS values are: t_{su}=3ns and t_h=2ns



(Positive) Edge-triggered D flip-flop

- A circuit, similar in functionality to the master-slave D flip-flop, can be constructed with 6 NAND gates (24 transistors)





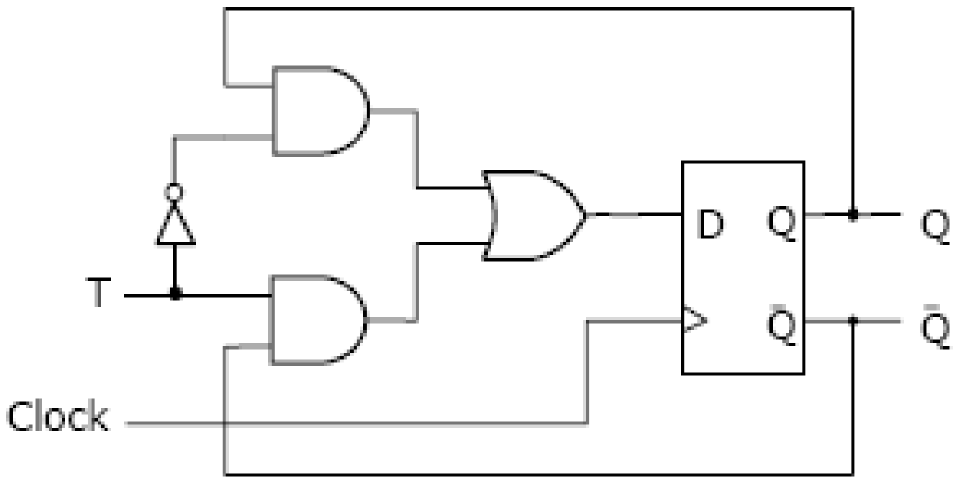
When clk = 0 => P1 = P2 =1 => this maintains the output latch in its present state;

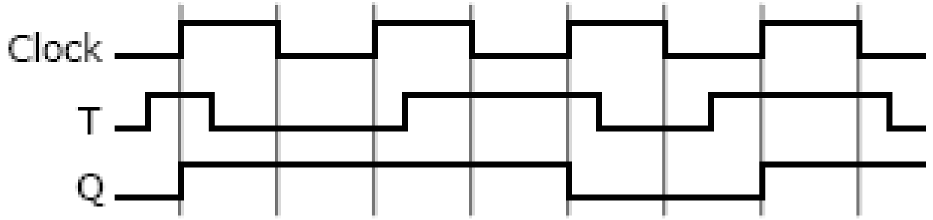
At the same time P3 => D and P4 => $\overline{D}$

When clk changes to 1 => P1 => $\overline{D}$ and P2 => D; which **sets** Q = D and $\overline{Q}$ = $\overline{D}$

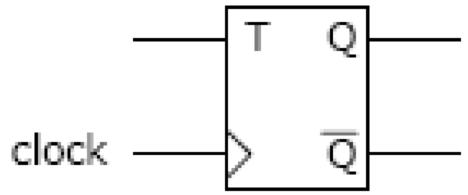
T flip-flop

- Another flip-flop type, the ***T flip-flop***, can be derived from the basic D flip-flop presented
- Feedback connections make the input signal D equal to the value of Q or Q’ under the control of signal labelled T





T	Q(t+1)
0	Q(t)
1	Q’(t)



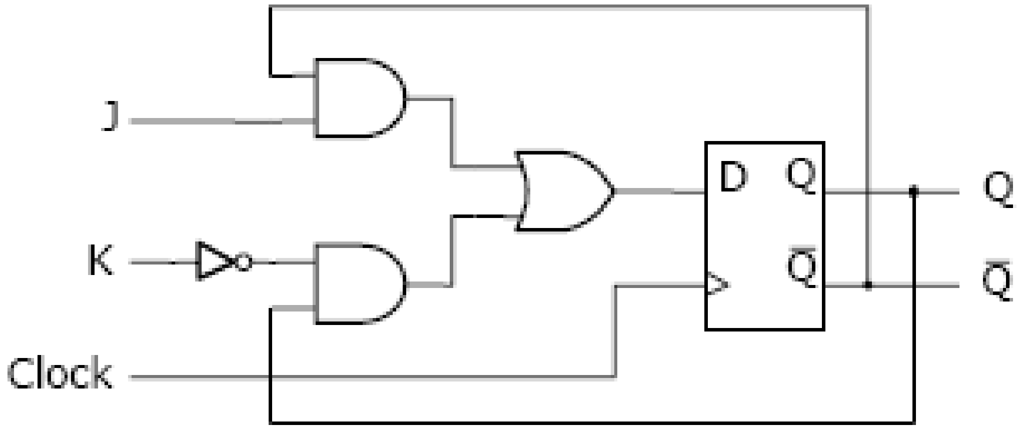
- The name T derives from the behavior of the circuit, which ‘toggles’its state when T=1

Q(t + 1) = T Q’ + T ’ Q

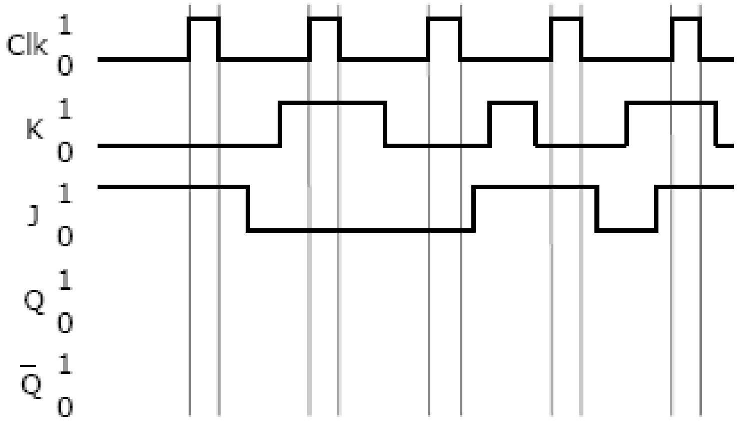
- – This feature makes the T flip-flop a useful element when
- – constructing counter circuits

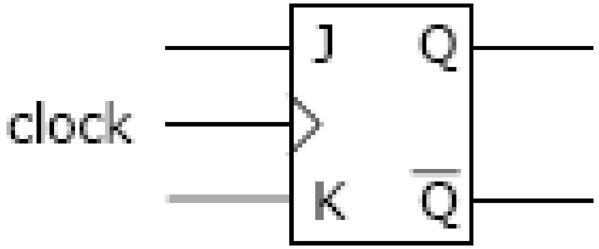
JK flip-flop

- The ***JK flip-flop*** can also be derived from the basic D flip-flop such that D=JQ'+K'Q
- The JK flip-flop combines aspects of the SR and the T flip-flop
  - It behaves as the SR flip-flop (where J=S and K=R) for all values except J =K=1
  - For J=K=1, it toggles like the T flip-flop



J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	Q'(t)

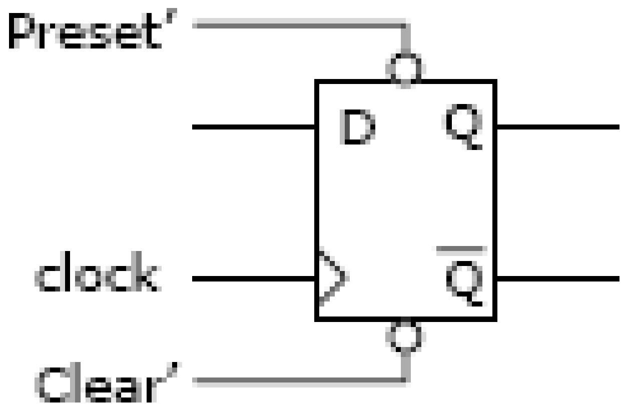




Q(t + 1) = JQ' + K'Q

Clear and preset inputs

- It may be desirable to specifically set (Q=1) or clear (Q=0) a flip-flop
- Practical flip-flops often have ***preset*** and ***clear*** inputs
  - Generally, these inputs are ***asynchronous*** (they do not depend on the clock signal)



As long as **Preset' = 0, Q=1**
As long as **Clear' = 0, Q=0**

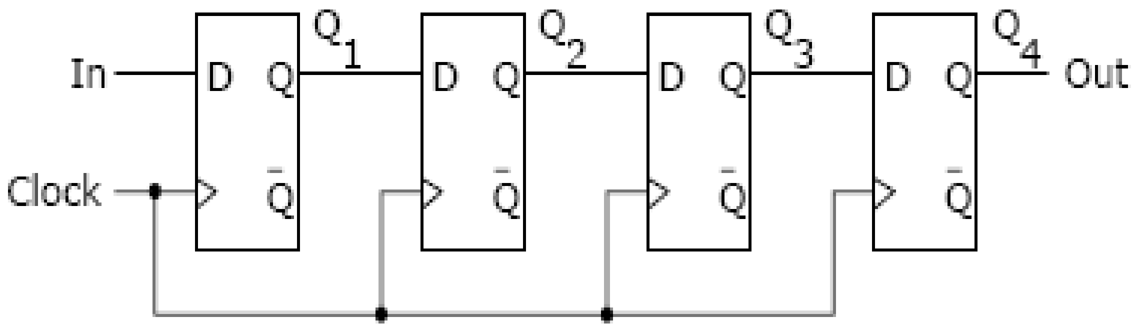
Registers

- A flip-flop stores **one bit** of information
- When a set of *n* flip-flops is used to **store *n* bits** of data, we refer to these flip-flops as a ***register***
  - Common register usages include
    - Holding a data value output from an arithmetic circuit
    - Holding a count value in a counter circuit
- A common clock signal is typically used for each flip-flop in a register

Shift register

A register that provides the ability to shift its contents by a single bit

- May be to the right or left (or possibly both)

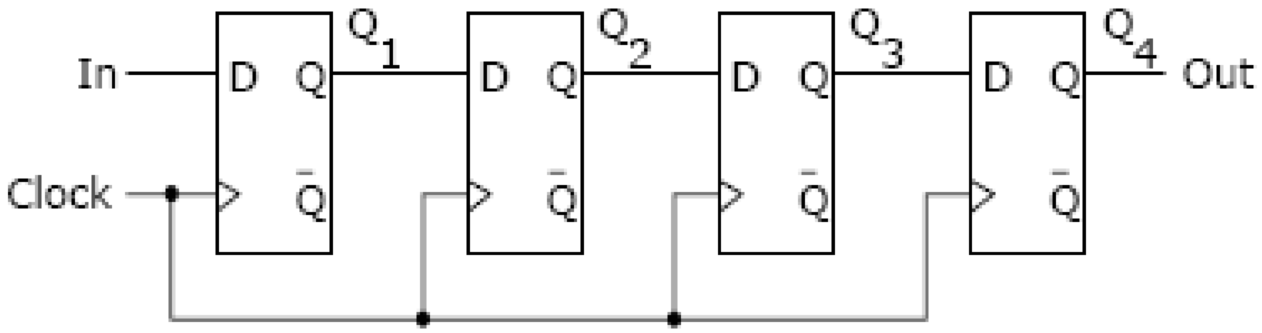


Shift right register

Shift right register

- Data is shifted to the right in a serial fashion using the ***In*** input
- Positive edge triggered
  - Contents of each flip-flop are transferred to the next flip-flop at each positive edge of the clock
- **Level sensitive devices** would **not** be appropriate for this circuit
- The transfer of new information into a register is referred to as ***loading*** the register

Serial Loading

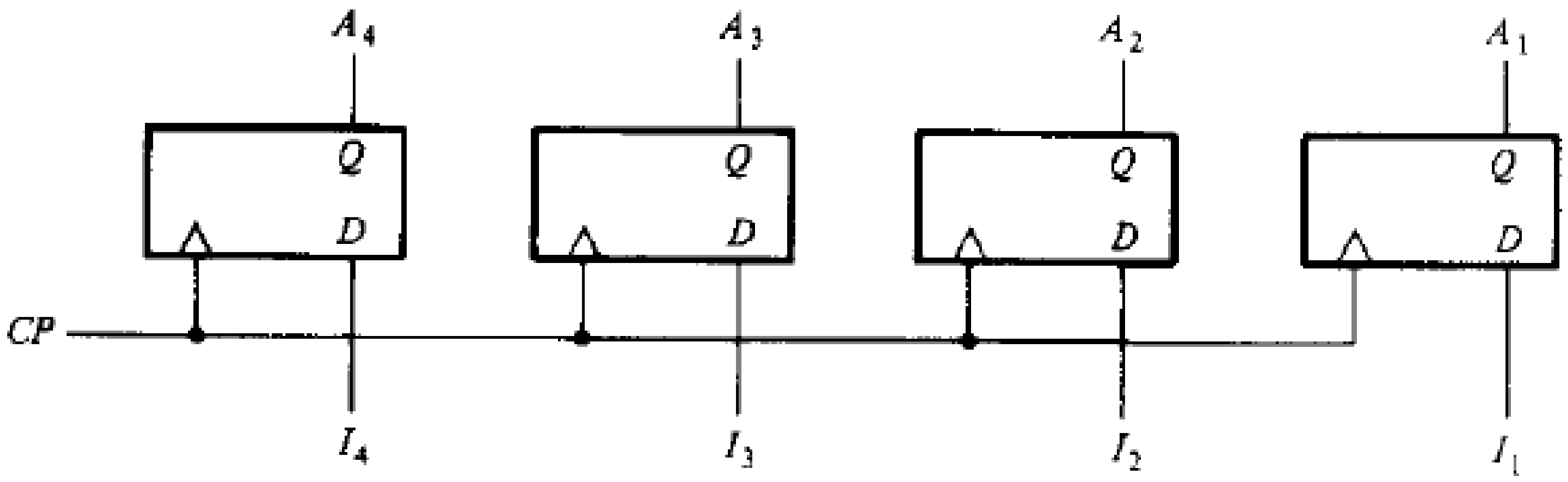


	In	Q_1	Q_2	Q_3	Q_4
t_0	1	0	0	0	0
t_1	0	1	0	0	0
t_2	1	0	1	0	0
t_3	1	1	0	1	0
t_4	1	1	1	0	1
t_5	0	1	1	1	0
t_6	0	0	1	1	1
t_7	0	0	0	1	1

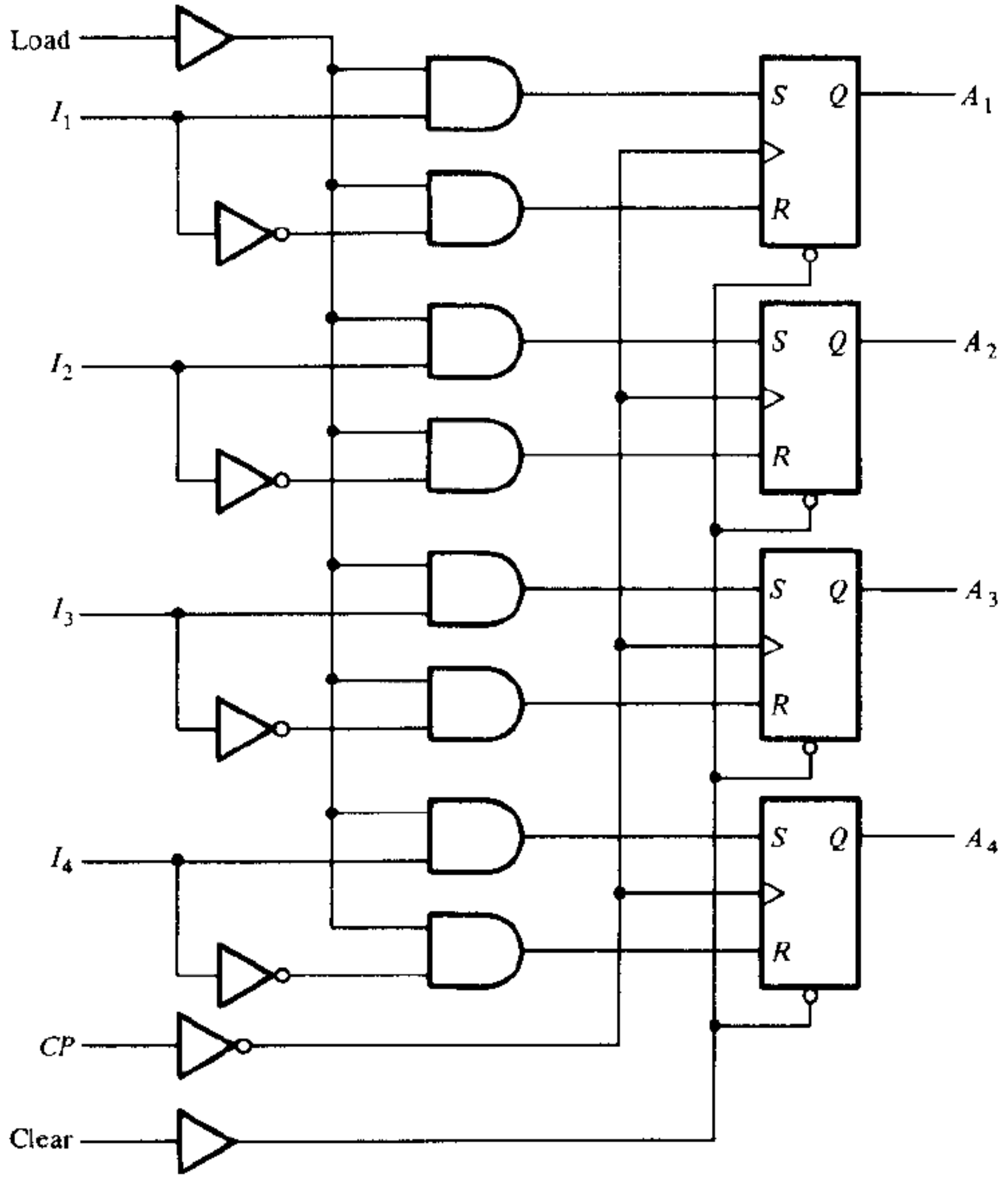
- the transfer is *1*-bit at a time, the transfer is said to be ***serial.***

Register loading ctd.

Parallel Loading



- transfer is *n*-bits (4 bits in this case) at a time, the transfer is said to be in ***parallel.***

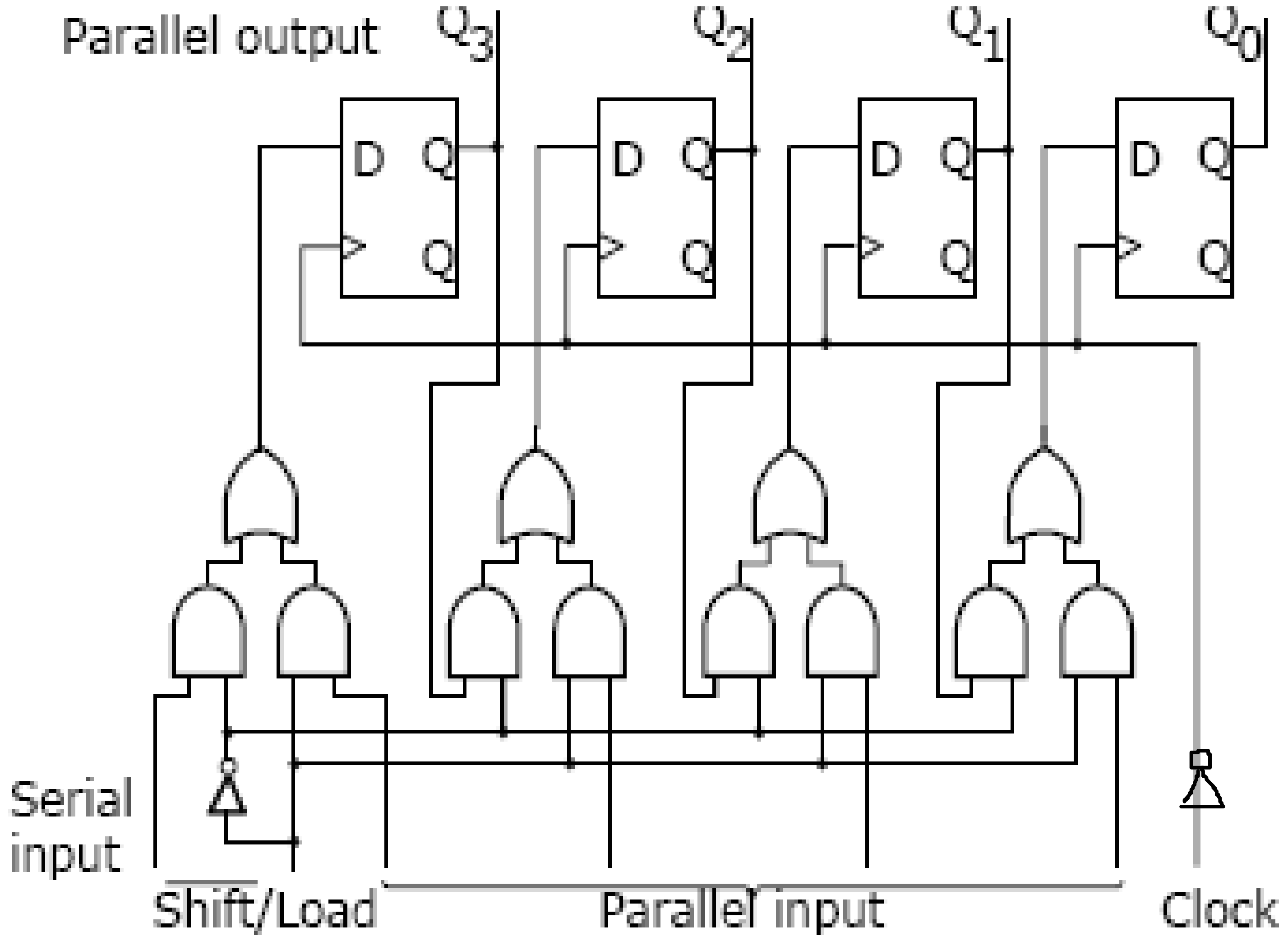


4-bit register with parallel load

The inverter in the CP path reduces the loading of the master clock generator (now its output is connected to only one gate –the inverter- instead of the four gate inputs).

The buffer in the 'Load' line plays the same role.

Parallel-access shift register



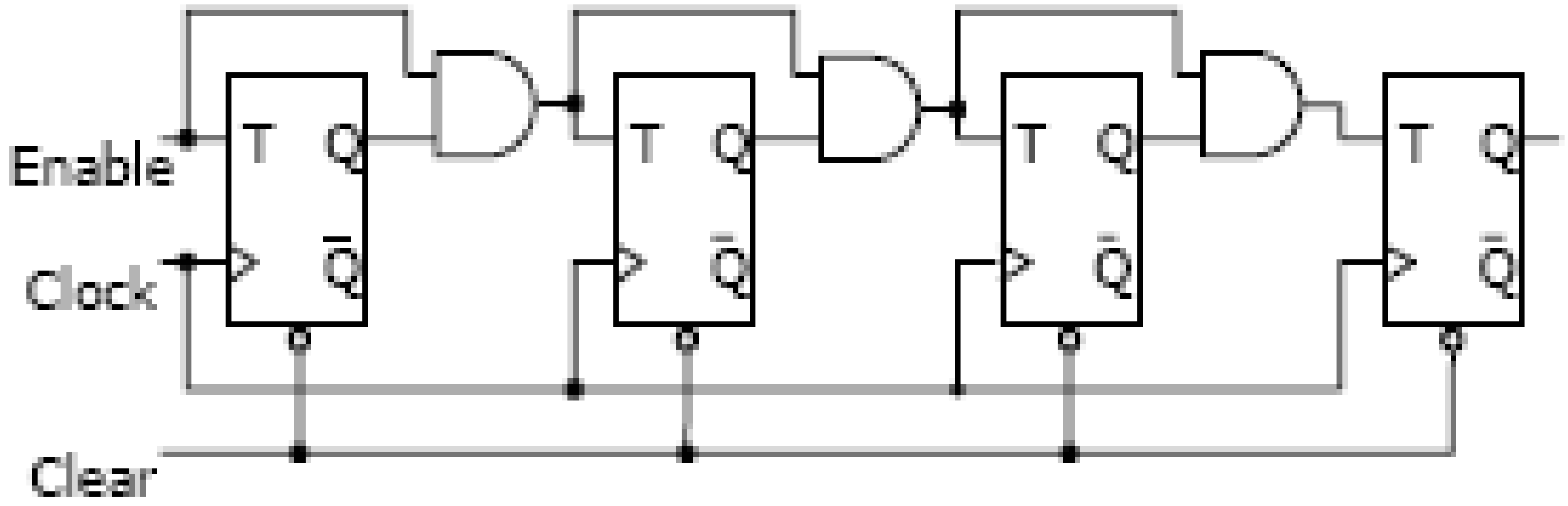
Shift Registers ctd.

- Data transfer in computer systems is a common function
  - If the transfer is *n*-bits at a time, the transfer is said to be in ***parallel***
  - If the transfer is *1*-bit at a time, the transfer is said to be ***serial***
- To transfer data serially, data can be loaded into a register in parallel (in one clock cycle) and then shifted out one bit at a time
  - **Parallel-to-serial data conversion**
- If bits are received serially, after *n* clock cycles the contents of a register can be accessed in parallel as an *n*-bit item
  - **Serial-to-parallel conversion**
- The clock pulses generated by a **Master-clock generator** are applied to all flip-flops and registers in a circuit.
- A separate **control signal** is then used to decide which specific clock pulse will have an effect on a particular register.

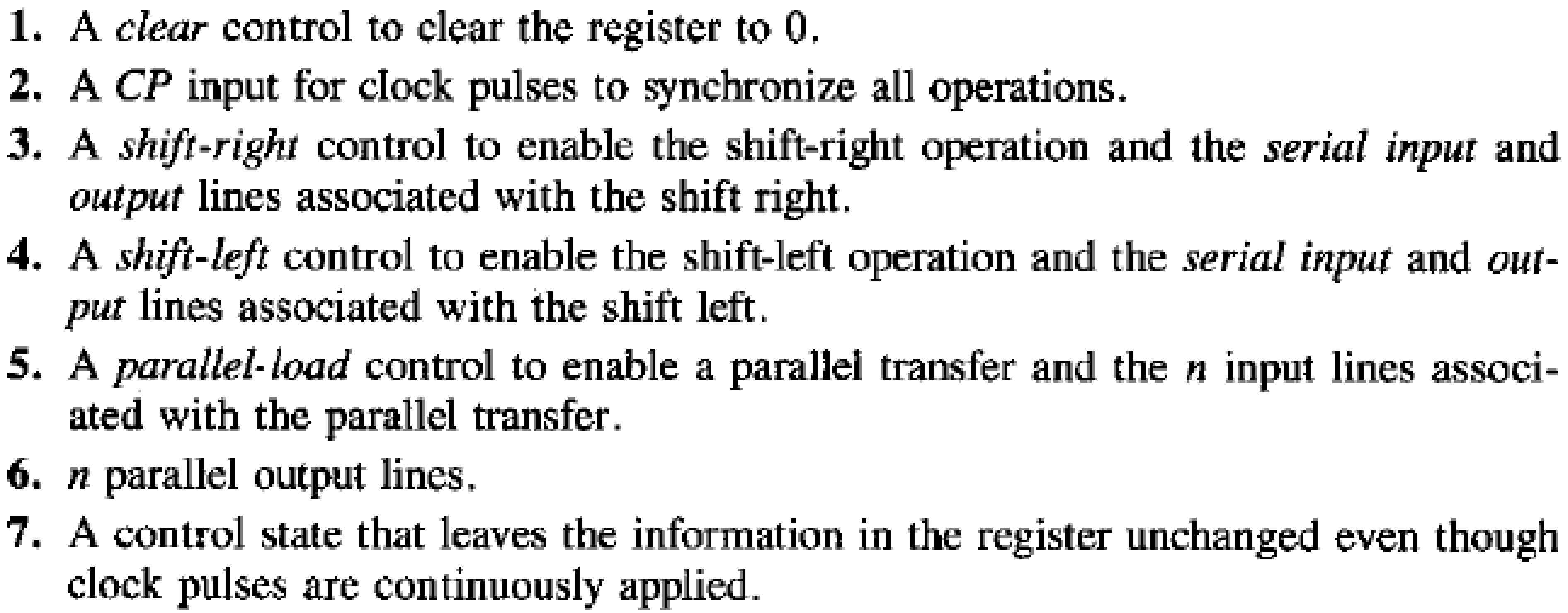
Shift Registers ctd.

- To fully synchronise the system, we must ensure that all clock pulses arrive at the **same time** to all inputs of all flip-flops so that they can all change simultaneously.
- A **clear** input can be used to clear the register to all 0's prior to its clocked operations.

Enable and clear capability

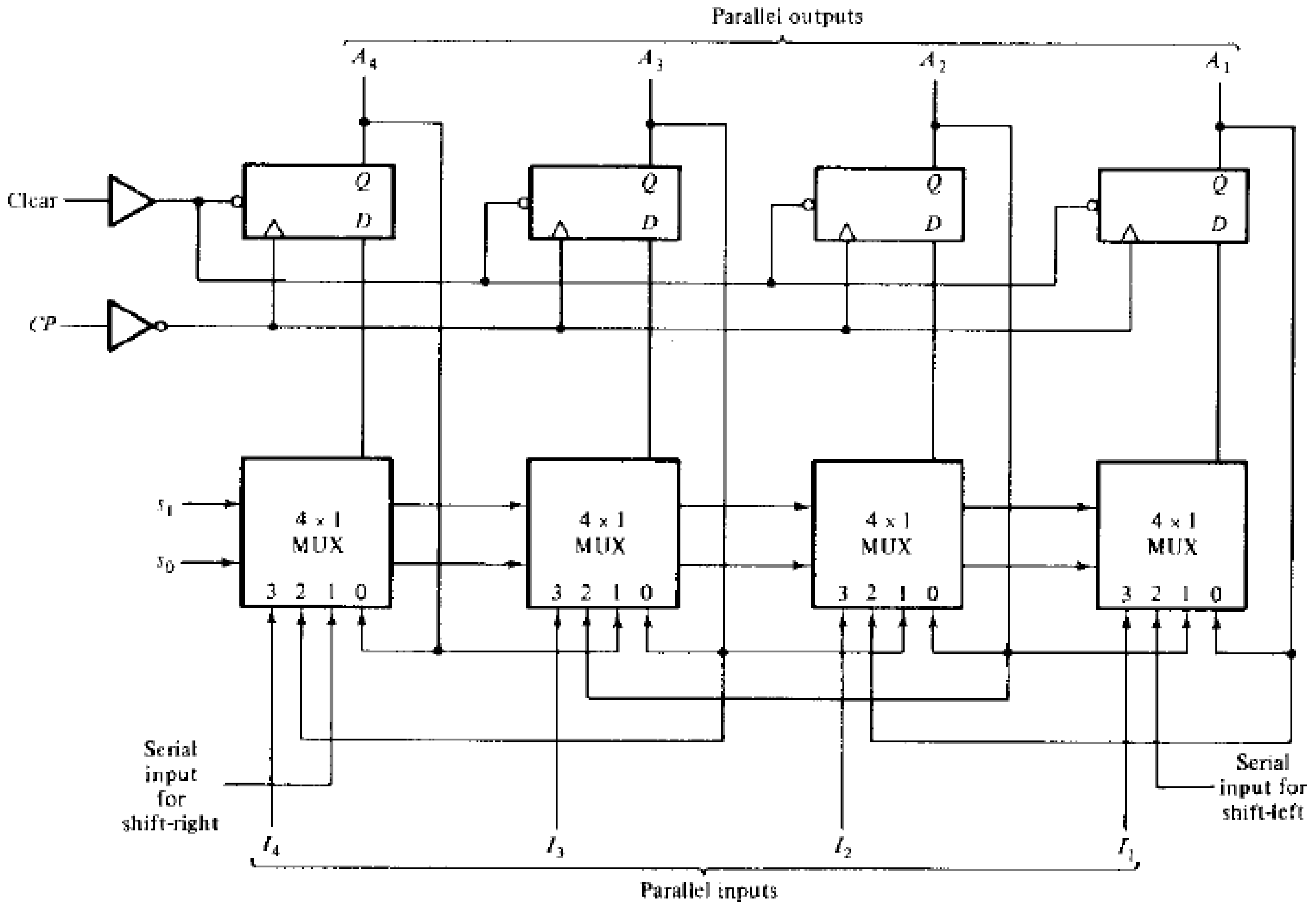


The most general shift Register will have the capabilities listed below:



1. A *clear* control to clear the register to 0.
2. A *CP* input for clock pulses to synchronize all operations.
3. A *shift-right* control to enable the shift-right operation and the *serial input* and *output* lines associated with the shift right.
4. A *shift-left* control to enable the shift-left operation and the *serial input* and *output* lines associated with the shift left.
5. A *parallel-load* control to enable a parallel transfer and the *n* input lines associated with the parallel transfer.
6. *n* parallel output lines.
7. A control state that leaves the information in the register unchanged even though clock pulses are continuously applied.

Register with all the capabilities 1 – 7 listed above:



Codes - Error Correction and Detection

Error detection codes

Errors in binary information can occur while being transmitted, stored or retrieved. When **single bit** or **multiple bit** errors occur, it is necessary to detect such errors and correct the error/s whenever possible.

Error detection and correction is done using the technique of **redundancy**: use of additional bits.

One of the most common ways to detect errors employ **parity bits**.

A parity bit is an extra bit included with binary data to make the total number of 1's either odd (**odd parity**) or even (**even parity**).

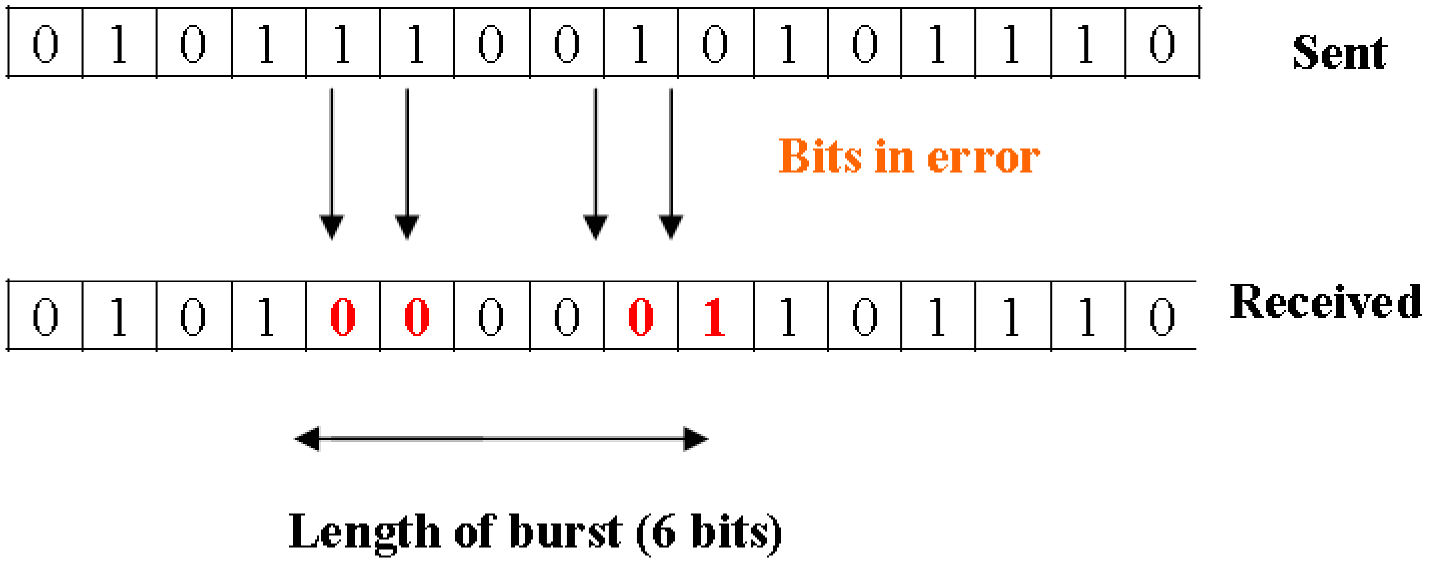
At the end of transmission, storage or retrieval, the number of 1's is checked again to see whether the parity has **remained unchanged**

- in which case we consider that there has not been any error introduced to data due to the operation.

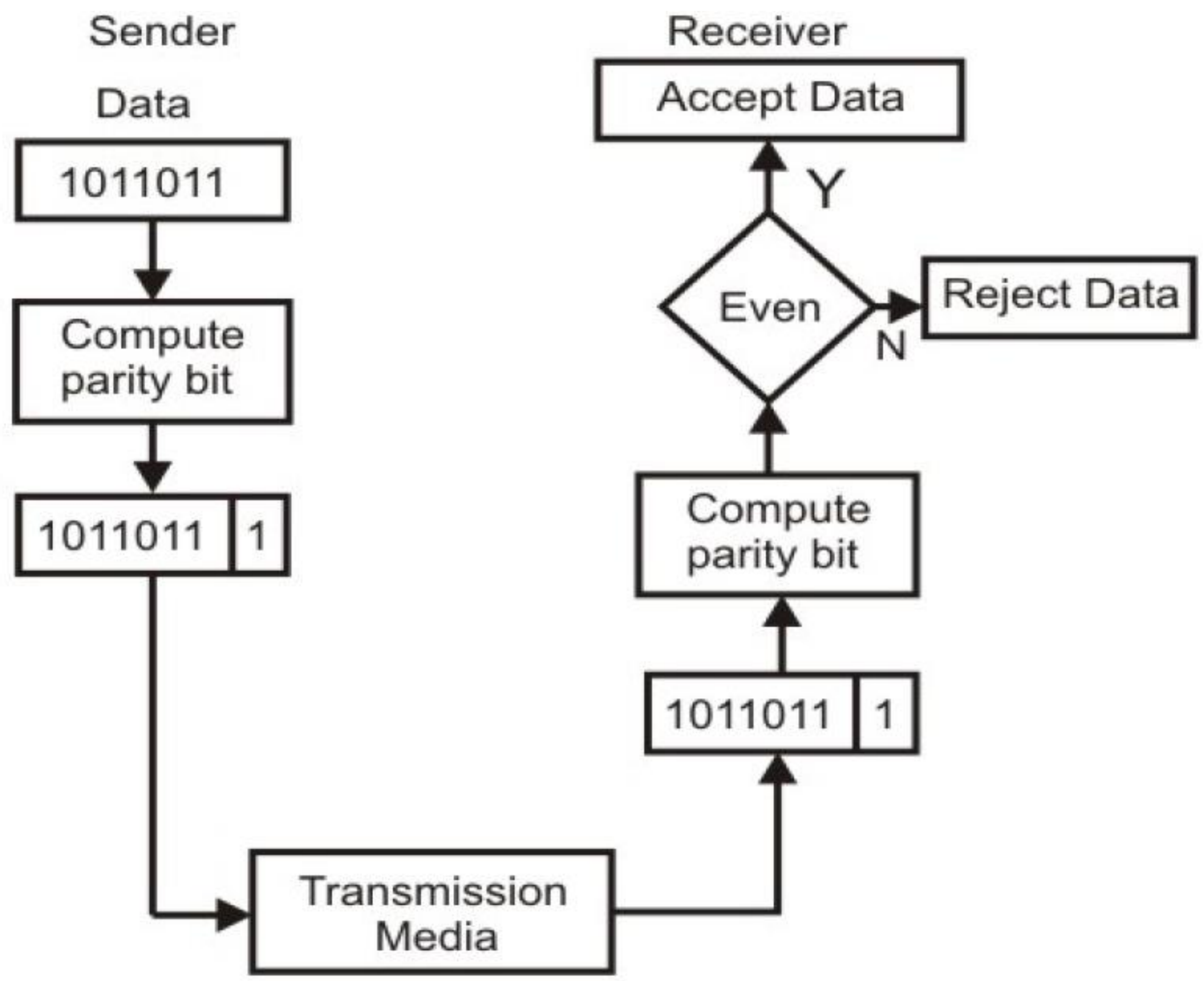
- Otherwise we will know that there has been a corruption of data.

Simple parity check can detect **all single bit errors** and also multiple bit errors when **the total number** of corrupted bits is **odd**.

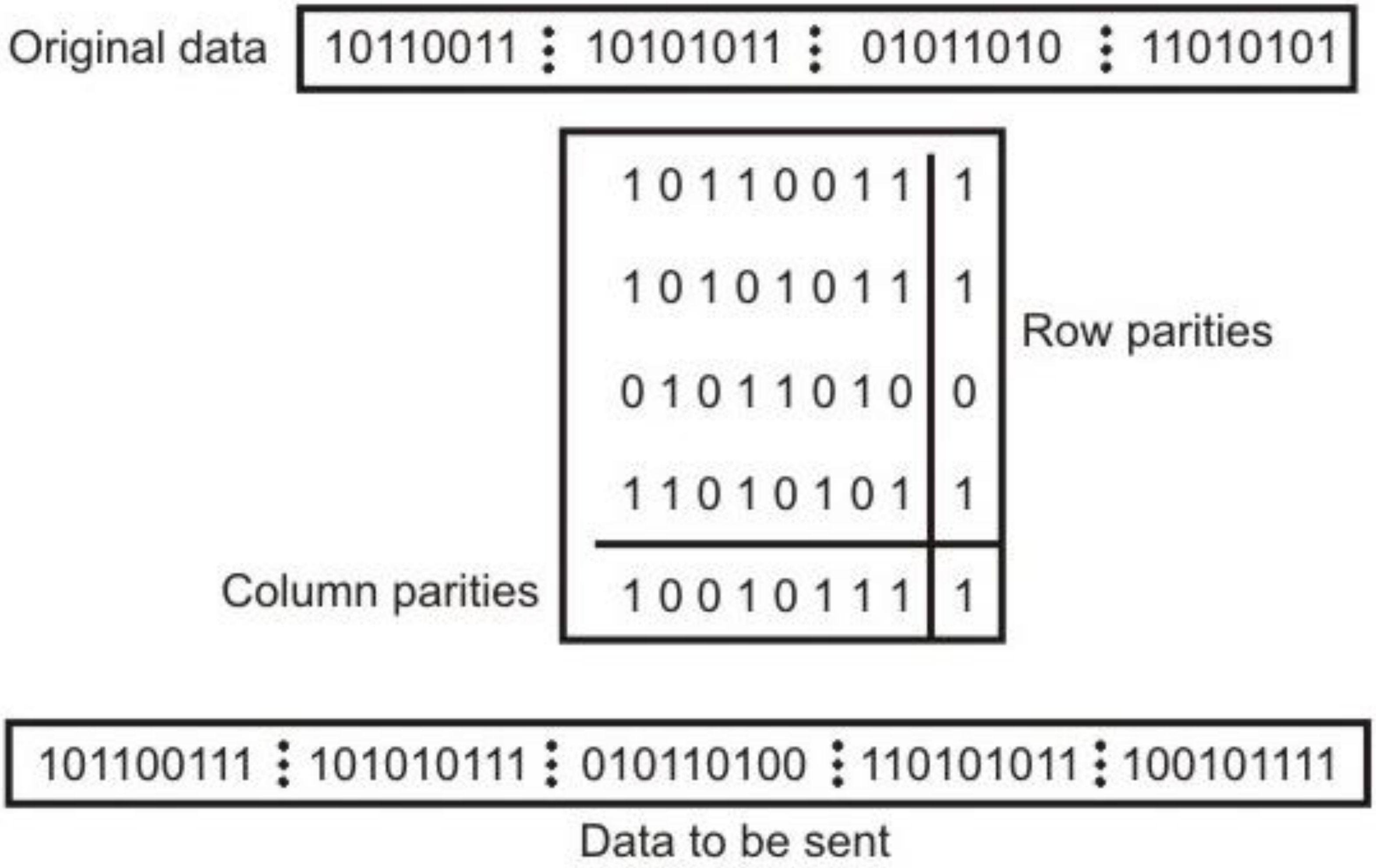
Burst Error


0 1 0 1 1 1 0 0 1 0 1 0 1 1 1 0
Sent
Bits in error
0 1 0 1 0 0 0 0 0 1 1 0 1 1 1 0
Received
Length of burst (6 bits)

Even parity Checking scheme



Two-dimension Parity Check



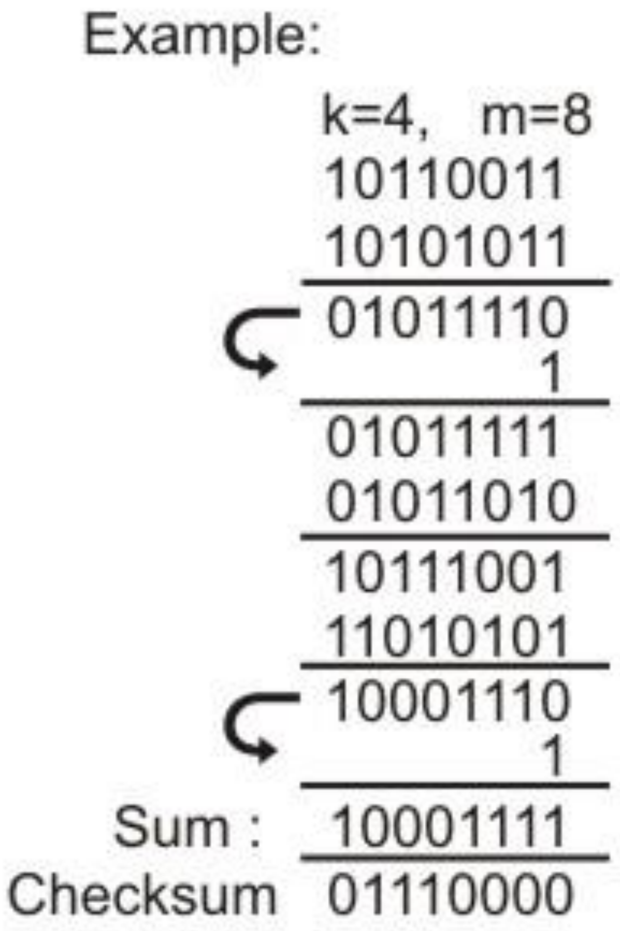
Checksum

In checksum error detection scheme, the data is divided into k segments each of m bits.

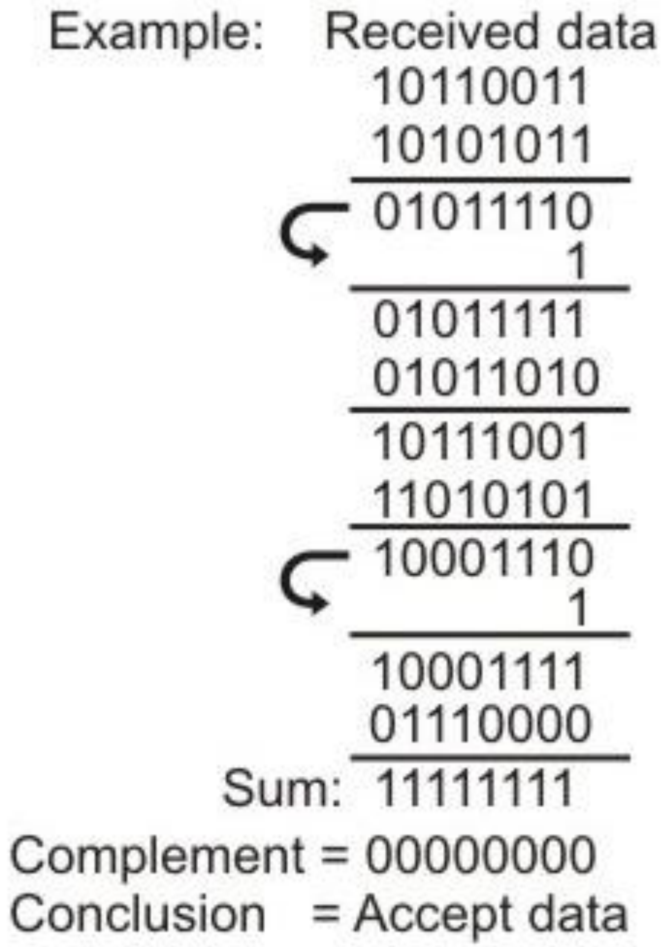
In the sender's end the segments are added using 1's complement arithmetic to get the sum. The sum is complemented to get the checksum. The checksum segment is sent along with the data segments.

At the receiver's end, all received segments are added using 1's complement arithmetic to get the sum. The sum is complemented. If the result is zero, the received data is accepted; otherwise discarded,

Sent Data



```
Example:
             k=4,   m=8
             10110011
             10101011
             --------
           ↪ 01011110
                    1
             --------
             01011111
             01011010
             --------
             10111001
             11010101
             --------
           ↪ 10001110
                    1
             --------
Sum :        10001111
             --------
Checksum     01110000
```



```
Example:    Received data
             10110011
             10101011
             --------
           ↪ 01011110
                    1
             --------
             01011111
             01011010
             --------
             10111001
             11010101
             --------
           ↪ 10001110
                    1
             --------
             10001111
             01110000
             --------
Sum:         11111111
Complement = 00000000
Conclusion   = Accept data
```

Error detection codes ctd.

Two dimensional parity check, **Checksum** and **Cyclic Redundancy Check** (CRC) based methods of error **detection** use multiple check bits to improve error detection capability. However they have **no error correction** capability.

Error-correcting code

One of the most common error correcting codes is the **Hamming Code**.

In the Hamming code, **k parity bits** are added to an **n-bit data word**, forming a new word of **n + k** bits.

The **bit positions** are numbered in sequence from 1 to n + k, left to right.

Positions numbered as **powers of 2** are reserved for **parity bits**. The remaining bits are **data bits**.

Example: Consider the 8-bit data word 11000100. A Hamming code with four parity bits can be generated for this data word as follows:

Bit position:	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
	P1	**P2**	**1**	**P4**	**1**	**0**	**0**	**P8**	**0**	**1**	**0**	**0**

Error-correcting code

Bit position:	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
	P1	**P2**	**1**	**P4**	**1**	**0**	**0**	**P8**	**0**	**1**	**0**	**0**

The four parity bits P1, P2, P4 and P8 are in positions 1, 2, 4 and 8, respectively. The eight bits of the data word are in the remaining positions.

Each parity bit is calculated as follows:

$$P_1 = \text{XOR of bits } (3, 5, 7, 9, 11) = 1 \oplus 1 \oplus 0 \oplus 0 \oplus 0 = 0$$

$$P_2 = \text{XOR of bits } (3, 6, 7, 10, 11) = 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 0$$

$$P_4 = \text{XOR of bits } (5, 6, 7, 12) = 1 \oplus 0 \oplus 0 \oplus 0 = 1$$

$$P_8 = \text{XOR of bits } (9, 10, 11, 12) = 0 \oplus 1 \oplus 0 \oplus 0 = 1$$

P1 checks the positions 3, 5, 7, 9 and 11: {001**1**, 010**1**, 011**1**, 100**1**, 101**1**}
P2 checks the positions 3, 6, 7, 10 and 11: {00**1**1, 01**1**0, 01**1**1, 10**1**0, 10**1**1}
P4 checks the positions 5, 6, 7 and 12: {0**1**01, 0**1**10, 0**1**11, 1**1**00}
P8 checks the positions 9, 10, 11 and 12: {**1**001, **1**010, **1**011, **1**100}

Error-correcting code ctd.

• **Bit position:**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
12 bit composite word stored in memory	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**0**
•	**P1**	**P2**	**1**	**P4**	**1**	**0**	**0**	**P8**	**0**	**1**	**0**	**0**

Thus, each parity bit is set so that the total number of 1's in the checked positions, including the parity bit, is always **even**.

When the 12 bits are read from memory, they are checked again for possible Errors. The parity is checked over the same combination of bits including the parity bit. The four check bits are evaluated as follows:

$$C_1 = \text{XOR of bits } (1, 3, 5, 7, 9, 11)$$

$$C_2 = \text{XOR of bits } (2, 3, 6, 7, 10, 11)$$

$$C_4 = \text{XOR of bits } (4, 5, 6, 7, 12)$$

$$C_8 = \text{XOR of bits } (8, 9, 10, 11, 12)$$

Error-correcting code ctd.

Then we can form the Number

$$C = C_8C_4C_2C_1 = 0000$$

Which is called the **syndrome**. C = 0000 indicate that no error has occurred.

Now, if C = 0101, then we will know that there is an error in bit 5 of the composite code.

The error can then be corrected by the **complementing** the corresponding bit (bit 5 in this case).

Hamming code can **detect and correct** only a **single error**.

More about timing & clocks …

-