Intro to Logic Design

What are digital circuits?

They are circuits that operate on signals that are True or False.

1	0
ON	OFF
HIGH	LOW
1	0

- True signal is indicated by High voltage. (5V)
- False signal is indicated by Low voltage. (0 or 2V)

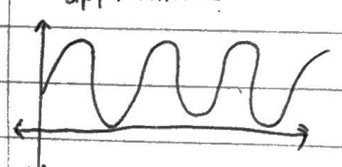
What is the difference between Analog and Digital Circuits?

Analog

- Signal can have any value V.
- Sensitive to noise & parameter variance.

Required for:

- power systems, audio applications

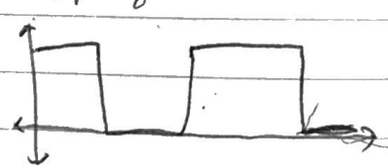


Digital

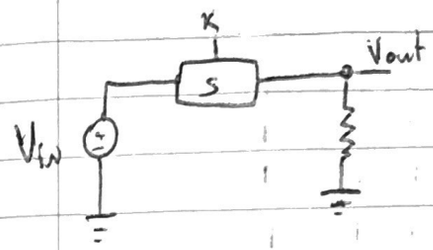
- Signals are valued True or False
- Insensitive to noise & parameter variance.

Required for:

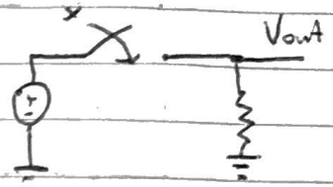
- digital electronics, digital computing



Switch



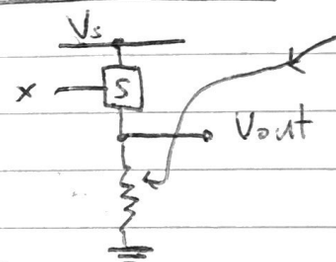
OR



$x = 1$, the switch is closed then $V_{out} = V_{IN}$

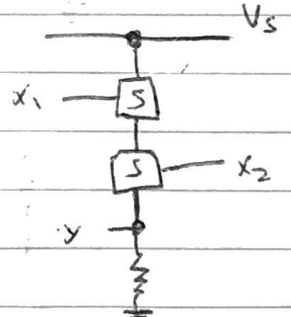
$x = 0$, the switch is open then $V_{out} = 0V$

Alternative Version



- Logical "ON" (True) is 1
- Logical "OFF" (False) is 0

Series Switch

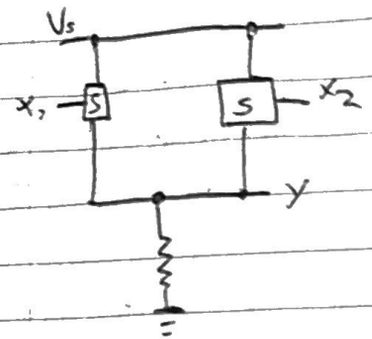


Truth table		
x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

And Gate

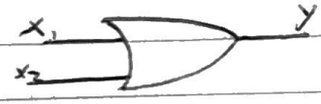
- $y = x_1 \cdot x_2 = x_1 x_2$

Parallel Switches



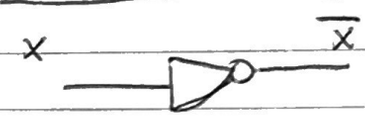
Truth Table		
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

OR Gate

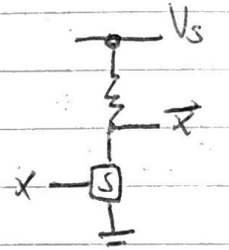


$y = x_1 + x_2$
↑
or

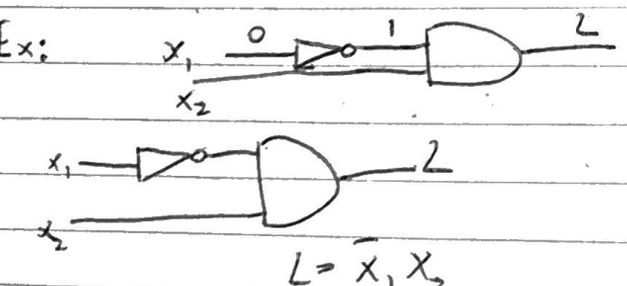
NOT Gate / Inverter



x	$\bar{x}$
0	1
1	0



Ex:



x_1	x_2	2
0	0	0
0	1	1
1	0	0
1	1	0

$L = \bar{x}_1 x_2$

What is a decimal number system?

- number system has 10 symbols: 0, 1, 2 ...
- represents larger numbers

What is a binary number system?

- two symbols : 0 and 1
- represents numbers: ones, twos, fours, eights ...

Ex: $10101 \rightarrow 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$

$= 16 + 0 + 4 + 1 = 21$

Ex 53 →

53	21	5	1	
−32	−16	−4	1	= 110101
22	5	1	0	

Proof Tables

When making a table ensure to follow binary pattern.

Ex:

1	1	1	
0	1	1	
1	0	1	
0	1	1	
1	1	0	...
0	0	0	
1	1	0	
0	1	0	
1	0	...	

- Each column the # of same consecutive elements doubles.

Boolean Algebra: Single Value Theorem

AND	OR	NOT
$x \cdot 0 = 0$	$x + 0 = x$	$\bar{\bar{x}} = x$
$x \cdot 1 = 1$	$x + 1 = 1$	
$x \cdot x = x$	$x + x = x$	
$x \cdot \bar{x} = 0$	$x + \bar{x} = 1$	

→ if $x = 1$, $\bar{x} = 0$
if $x = 0$, $\bar{x} = 1$

Multivariable Theorems

Commutative Law

- $x \cdot y = y \cdot x$ and $x + y = y + x$

Associative Law

- $x \cdot (y \cdot z) = (x \cdot y) \cdot z$ and $x + (y + z) = (x + y) + z$

Absorbition Law

$x + xy = x$ → proof →
$x + xy = x \cdot 1 + x \cdot x$
$= x(1 + y)$
$= x(1)$
$= x$

$x(x + y) = x$

Distributive Law

$x(y + z) = xy + xz$ and $x + yz = (x + y)(x + z)$

Combining Law

- $xy + x\bar{y} = x$ → proof →
$x(y + \bar{y})$
$x(1)$
x

- $(x + y)(x + \bar{y}) = x$ → proof →
$= xx + yx + x\bar{y} + \bar{y}y$
$= x + xy + x\bar{y} + 0$
$= x(1 + y + \bar{y})$
$= x(1)$
$= x$

Other Law or "Like Absorbtion Law"

$x + \bar{x}y = x + y$ $\qquad$ $x(\bar{x} + y) = xy$

DeMorgan's Law

$\overline{x \cdot y} = \bar{x} + \bar{y}$ $\qquad$ $\overline{x + y} = \bar{x} \cdot \bar{y}$

Multivariable: $\overline{ABC} = \bar{A} + \bar{B} + \bar{C}$ or $\overline{A + B + C} = \bar{A} \cdot \bar{B} \cdot \bar{C}$

Consensus Law

$xy + yz + \bar{x}z = xy + \bar{x}z$

Avoid these Common Law Mistakes pt. 1

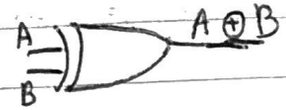
$\bar{A} + \bar{B} \neq \overline{A + B}$ $\qquad$ $\bar{A} \cdot \bar{B} \neq \overline{A \cdot B}$

Logic Gates

Exclusive OR - XOR - $\oplus$

- one or the other, not both.
- only works with odd number of inputs

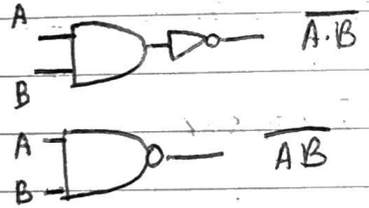
A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0



$$A \oplus B = \bar{A}B + A\bar{B} = (A+B)\cdot(\bar{A}+\bar{B})$$

NAND - NOT AND

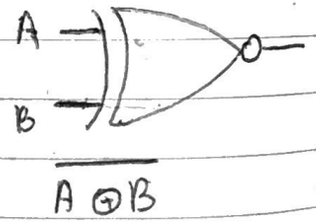
A	B	$\overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0



XNOR - (XOR + NOT)

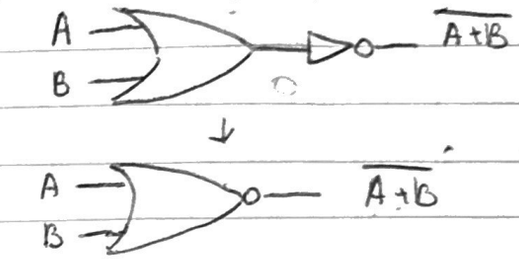
A	B	$\overline{A \oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1

$$\overline{A \oplus B} = \bar{A}\bar{B} + AB$$



NOR - NOT OR

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



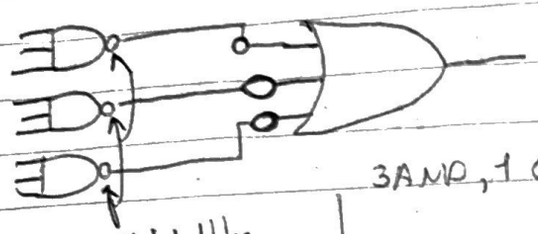
Multi inputs

XOR counts the number of true inputs and outputs true when an odd number of input are true.

XNOR outputs true when an even number of inputs are true.

Sum of Product	Product of Sum
• look for rows that = 1 • find product to make row true • OR in between, in final form	• look for rows that = 0 • find sum to make row true • AND in between, in final form

How to turn SOP to only NANDs?

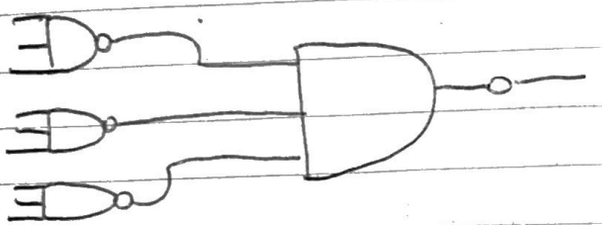


$\bar{x}+\bar{y}+\bar{z} = \overline{x \cdot y \cdot z}$

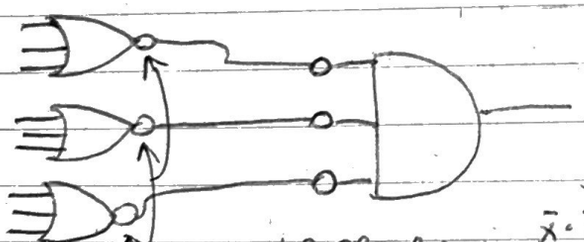
3 AND, 1 OR

added bubbles

↓ 4 NAND



How to turn POS to only NORs?

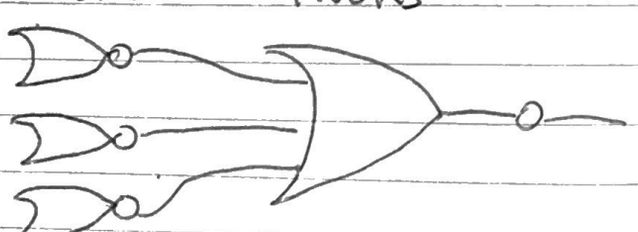


$\bar{x} \cdot \bar{y} \cdot \bar{z} = \overline{x+y+z}$

3 ORs, 1 AND

added bubbles

↓ 4 NORs



Karnaugh Map

What is it? Implementation method for a digital circuit.

two variable case

Truth table

x_1 x_2	
0 0	m_0
0 1	m_1
1 0	m_2
1 1	m_3

K-map

→

x_2 \ x_1	0	1
0	m_0	m_2
1	m_1	m_3

*Ex:

x_2 \ x_1	0	1
0	1	0
1	1	1

$\bar{x}_1 + x_2 = f$

large variable case

groups of two are made

CD \ AB	00	10	11	01
00				
01				
11				
10				

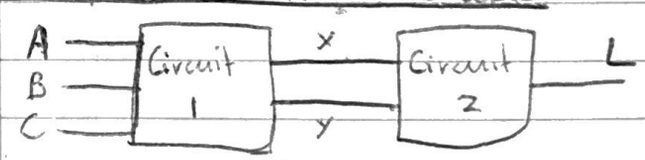
→ look at it as

00 → into symbols

$\bar{Y}\bar{Z}$ $Y\bar{Z}$ $Y2$ $\bar{Z}Y$

K-map with Indeterminate Cases

Ex:



3 essential cases:
- $x=0, y=0$
- $x=0, y=1$
- $x=1, y=0$

don't care about $x=1, y=1$.

Circuit 2

x y	L
0 0	0
0 1	1
1 0	1
1 1	x

→

x \ y	0	1
0	0	1
1	1	x

→ $f = x\bar{y} + xy = x+y$

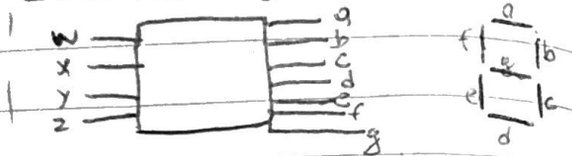
Binary Coded Decimal (BCD)

What is it? Common way to represent decimal numbers in decimals.

decimal	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	?

↓ splits into two BCDs → 0001 0000

BCD to 7-seg. Display



Binary Subtraction

General Method:

$$\begin{array}{r} 1001 \\ 0111 \\ \hline 0010 \end{array} \quad \rightarrow \quad \begin{array}{r} 1111 \\ 0101 \\ \hline 1010 \end{array} \quad ;$$

Alternate Method:

1. 2s complement of lower num / 2nd num.
2. Then add new num to 1st num.

Ex:

$$\begin{array}{r} 1000 \\ + \; 1111 \\ \hline 1000 \\ + \quad 1 \\ \hline 1001 \end{array} \quad = \quad \begin{array}{r} \downarrow \\ 0000 \\ + \quad 1 \\ \hline 0001 \end{array}$$

2s complement: 0101 1111
1) 1s complement → 1010 0000
2) add 1 → + 1 → 1010 0001

1s complement: 0101 1111
1) "inverse nums" → 1010 0000

Registers

Data is stored in them at low level. Values are represented in binary.

- Fixed size of 8 bits - 1 byte.
- Size is usually 2^N.

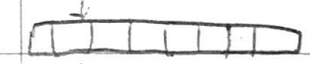
1 byte

2 bytes

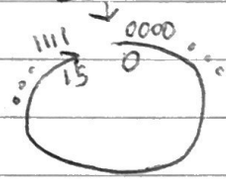
What is an Overflow? when math with registers produces incorrect results due to overflow. Like 2 x 1111 1111 = will yield a num > 8 bits.

What is a better way to represent a register?

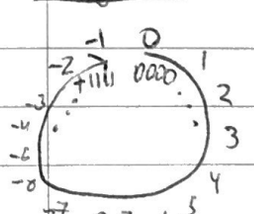
ok



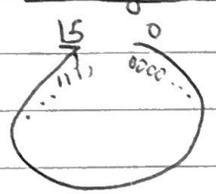
better



Signed

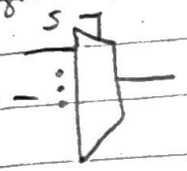


Unsigned

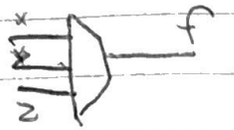


Multiplexes

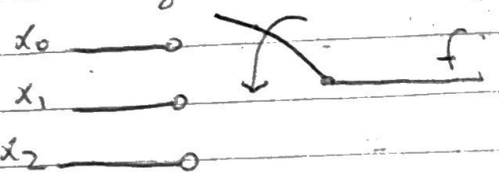
Circuit that combines multiple inputs into a single output.



Ex:

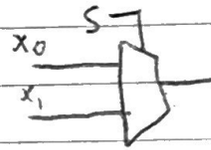


- Works using synchronized switching.



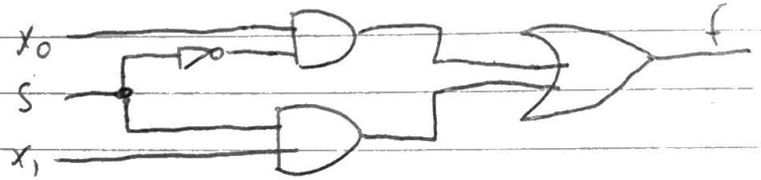
N = # control bits
2^N = # inputs

Example 2-1 Mux



- When $S=0$ then $f=x_0$
- When $S=1$ then $f=x_1$

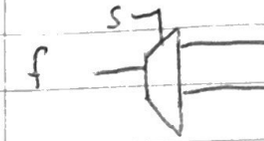
$\longrightarrow f = \bar{S}x_0 + S x_1$



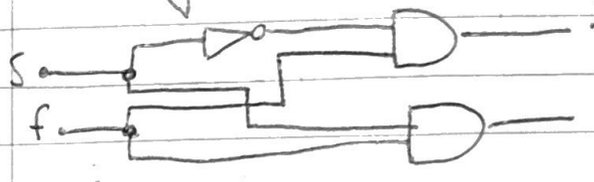
N - num of control bits
S - num of inputs

Demultiplexes

Circuit that "deciphers" single input into multiple outputs



$x_0 = \bar{S}f \quad x_1 = Sf$



Clock signal: square wave signal that is periodic, alternates between 1 and 0.

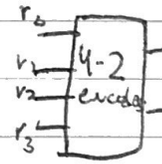
Ex 1: S_0

Ex 2: S_1

Encoder: digital circuit that takes a set of bits and represents them compactly/compresses them.

- "lossless" or "lossy"

Ex:



$e_0 = r_2 + r_3$

$e_1 = r_1 + r_3$

r_0	r_1	r_2	r_3	e_0	e_1
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

Decoder: digital circuits that recovers encoded bits into original bits.
- "loss less"

e_0	e_1	r_0	r_1	r_2	r_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

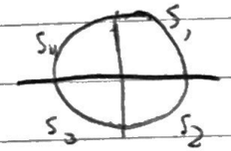
$r_0 = \bar{e}_0 \cdot \bar{e}_1$

$r_1 = \bar{e}_0 \cdot e_1$

$r_2 = e_0 \cdot \bar{e}_1$

$r_3 = e_0 \cdot e_1$

Wheel Encoder or Rotary Encoder:
uses a set of "sensors" to identify an angle.

Ex. 

- 4 sensors
- motor can only be in 1 section

S_0	S_1	S_2	S_3
0	1	0	0

Ex 2: 360 (locations) sensors → 9 encoded bits.

Priority Encoder.
Only communicates the most important output that is true.

Ex:
Increasing priority ↓
1. no alerts
2. trash needs to be taken out
3. House is on fire

Note: in digital circuits there are often signals with some form of redundancy called "other encoders".

Logic Gates with Feedback:

AND gate

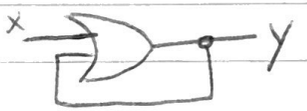
start y=0

x	y
0	0
1	0

; start y=1

x	y
0	1 → 0
1	1

OR gate



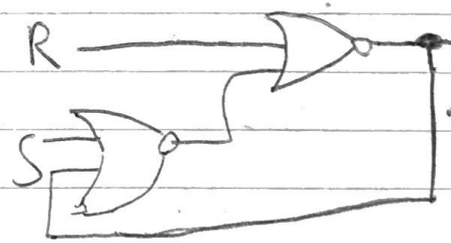
y=0

x	y
0	0
1	0 → 1

y=1

x	y
0	1
1	1

Basic Latch



- if $Q_0 = 0$

S	R	Q^+
0	0	0
0	1	0
1	0	1
1	1	0

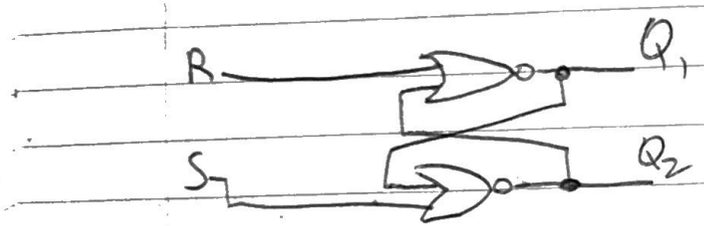
- if $Q_0 = 1$

S	R	Q^+
0	0	1
0	1	0
1	0	1
1	1	0

Summary

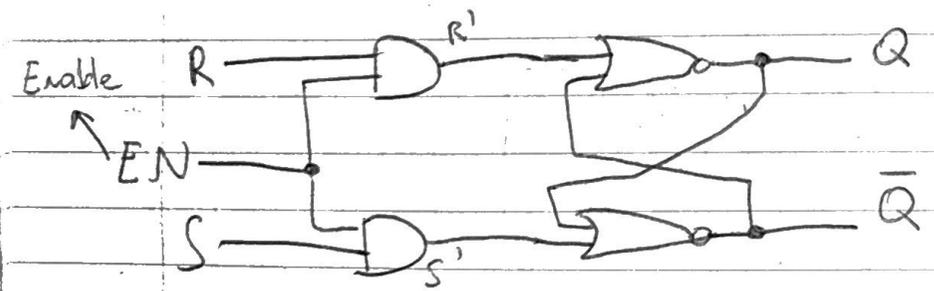
- $S=0, R=0$: $Q \to Q^+$ ($0 \to 0$, $1 \to 1$) - hold case
- $S=0$, $R=1$: Set Q to zero - "Reset"
- $S=1$, $R=0$: Set Q to one - "Set"
- $S=1$, $R=1$: Set Q to zero - "Reset"

SR Latch - common form



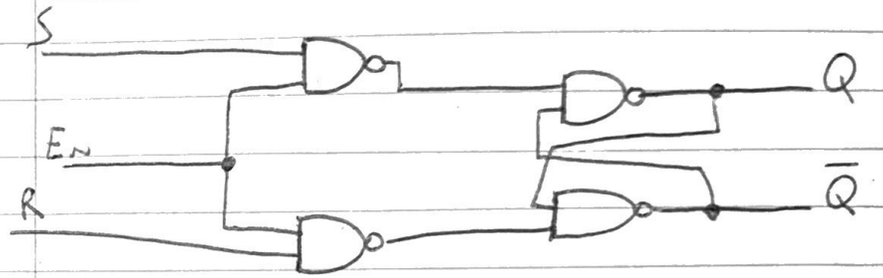
S	R	Q_1	Q_2	
0	0	Q	Q_1	hold
0	1	0	1	Reset
1	0	1	0	Set
1	1	0	0	not needed

Gated SR Latch

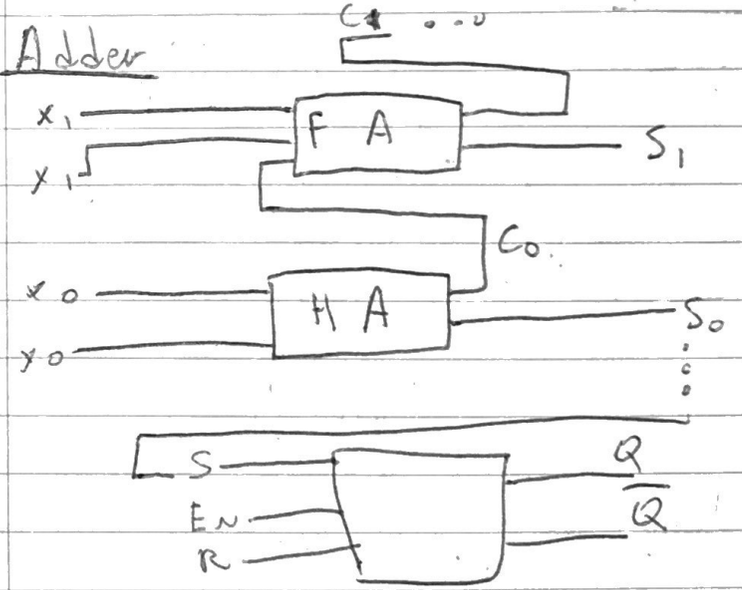


E_N	R	S	Q^+	$\overline{Q^+}$	
0	x	x	Q	$\overline{Q}$	hold
1	0	0	Q	$\overline{Q}$	hold
1	0	1	1	0	set
1	1	0	0	1	reset

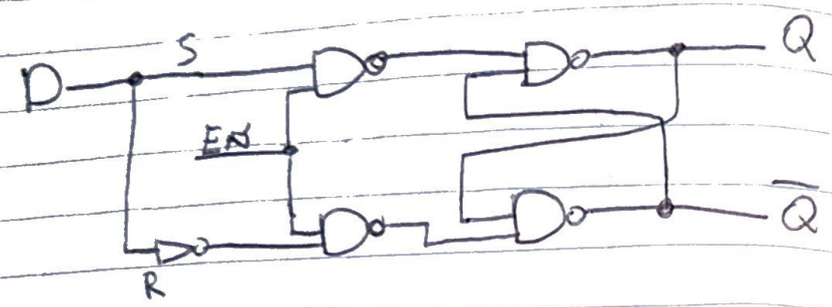
Gated SR Latch (With NAND gates)



N Bit Adder



Gated D Latch



- Value of D is stored into Q when $EN=1$.

EN	D	Q	$\bar{Q}$	
0	x	Q	$\bar{Q}$	hold
1	0	0	1	reset
1	1	1	0	set

All flip flops are 1 bit memory elements that store 0 or 1.

D Flip Flops

What is it? simplest flip flop.
- 1 data input - D ... - 1 output - Q
- 1 clock input - CLK

Why is it? Stores value of D once a clock switches ↓

$$Q_{N+1} = D_N \quad / \quad Q_{next} = D$$

JK Flip Flops

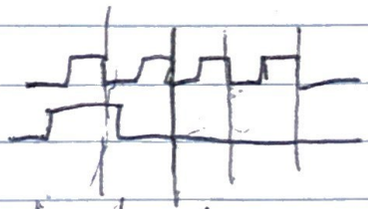
- 2 inputs, J (set) & K (Reset)
- 1 clock input - Clk
- 1 output - Q

Why is it? Designed to solve "invalid state" aka S = R = 1 of SR Latch. Main feature is that it can be configured to 4 states:
- hold - set - reset - toggle

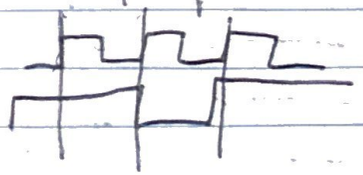
$$Q_{next} = (J \;\&\; !Q) \text{ OR } (!K \;\&\; Q)$$
$$Q_{N+1} = (J_N \text{ AND } \bar{Q}_N) \text{ OR } (\bar{K}_N \text{ AND } Q_N)$$

When values are stored in Flip Flop?

(Negative side) 1) Falling Edge, when clk signal is falling to 0.



(Positive side) 2) Rising Edge, when clk signal is rising to 1.



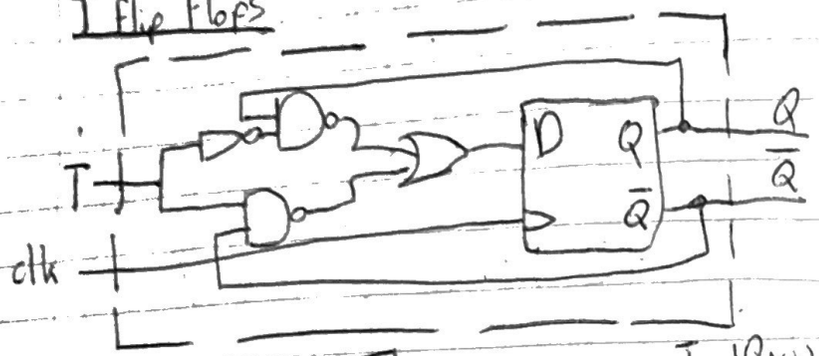
Misc Info

Preset & Clear pins set value of Q and Q NOT at init phase. They are "active low" so when set to high they are off.

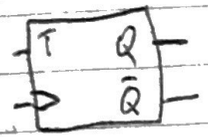
Counters

Use logic gates and flip flops to count binary

T Flip Flops

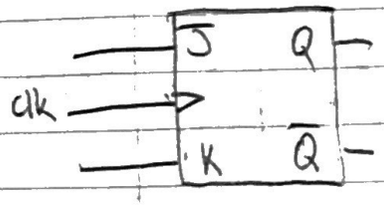


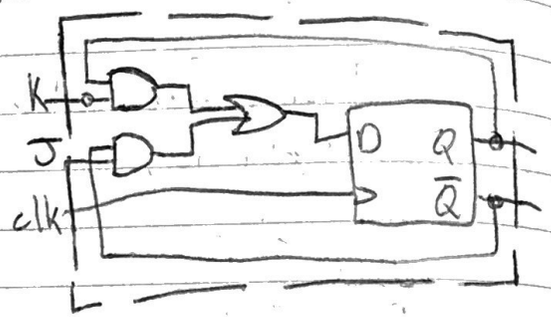
Symbol:



T	Q_{N+1}
0	Q_K hold
1	$\bar{Q}_K$ toggle

JK Flip Flops Symbol



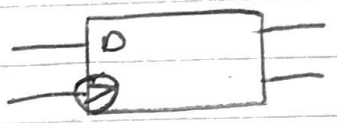


J	K	Q_{K+1}	
0	0	Q_K	"hold"
0	1	0	"reset" / K - "kill" because Q ↓
1	0	1	"set" / J - "jump" because Q ↑
1	1	$\bar{Q}_K$	"toggle"

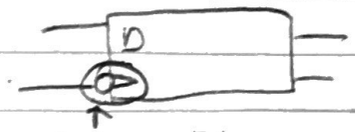
Reminder on Flip Flops: Value of Q always depends on the previous value of Q.

- depend on initial conditions

How to know if its Falling or Rising Edge?



Rising edge



Falling Edge

D Flip Flops Truth Table

Clk	D	Q	$\bar{Q}$
Edge	0	0	1
Edge	1	1	0

How to analyze Flip Flop Circuits?

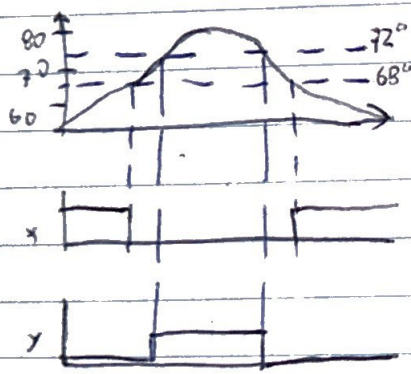
1) Use initial Q, $\bar{Q}$ and plot.
2) Update (A, B, J...) signals and evaluate gates.
3) Use J|K and whatever (D or T) to find next value Q.
4) Plot Q, $\bar{Q}$ for next clock period.
5) Repeat from # step 2 to find new Q.

State Machines

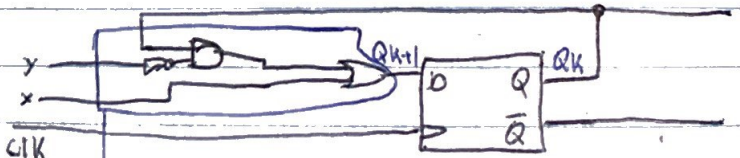
Digital circuit that use flip flops, inputs, outputs and state (stored memory). Next state is controlled by current state and inputs using "transitional logic". Outputs are determined by "output logic".

Example

- goal to regulate temp near 70° F.
- turn on heater when temp is ~~at~~ below 68° F.
- turn off heater when tem is over 72° F.



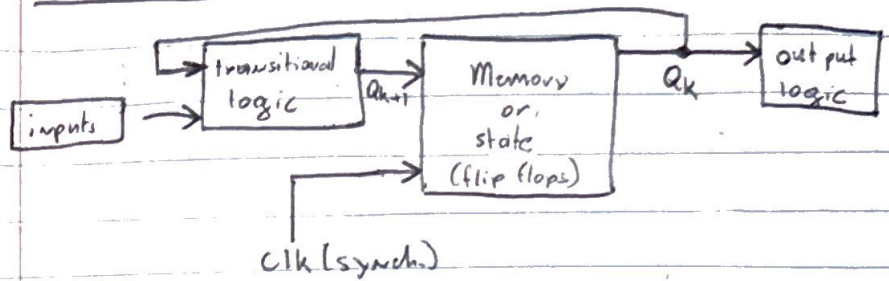
State Machine with D flip flops



transitional logic ←

Backwards "sort off" structure.

Schematic



Types of State Machine

Moore machine - output is completely determined by the current state.

Mealy machine - output is determined by both current state and inputs.

How to Design State Machines?

Using D flip flops...

1. draw state transition diagram
2. define all states (binary)
3. fill out state table for next states and outputs
4. Use boolean algebra / K-map to define (Q_{k+1} / next) logic.
5. Use boolean algebra / K-map to define output logic.
6. Draw circuit.
7. Simulate in Logisim / ModelSim
8. Build it!

Using JK flip flops...

4.5 Convert "Q_{k+1}" logic to JK logic meaning one expression for J and another for K.

Note: just an extra step compared to D flip flops.

Ex 1:

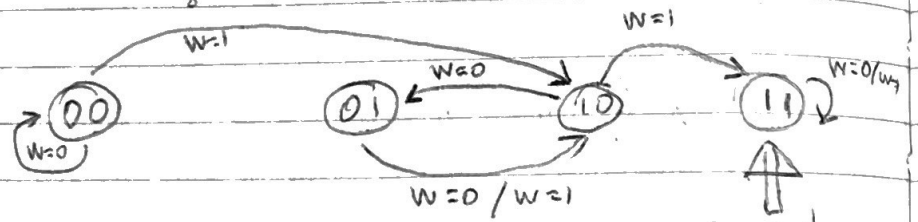
Current AB	Next AB_{+1} w=0	Next AB_{+1} w=1	Output Z
00	00	10	1
01	10	10	0
10	01	11	0
11	11	11	1

$Z = \overline{B \oplus A}$

$B_{k+1} = A_k$

$A_{k+1} = w + B_k$

Transition diagram:



input → w

state → AB

output → Z

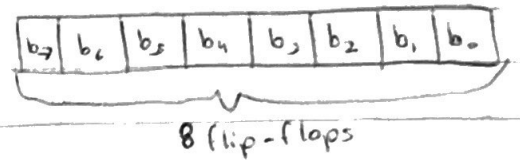
init → 00

Registers

- collection of flip flops used to store bits.
- sizes are powers of 2.

vocab: byte is 8 bits.

Ex:

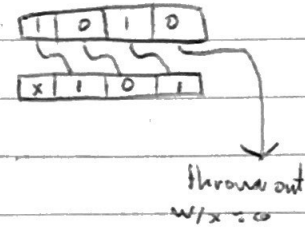


Shift Registers

- circuit designed to shift bits in a specific order.

Ex:

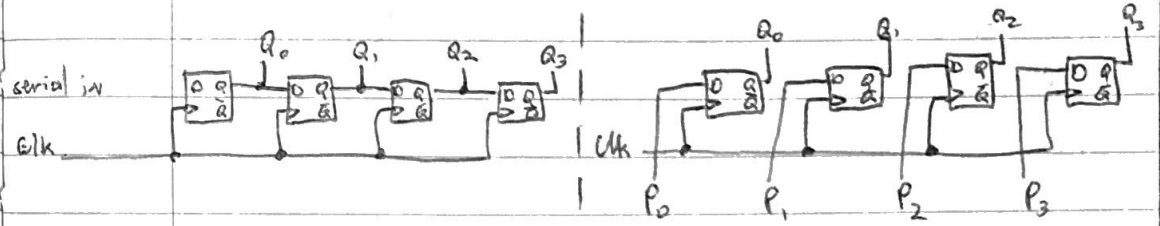
10 10 → ? 1 0 1



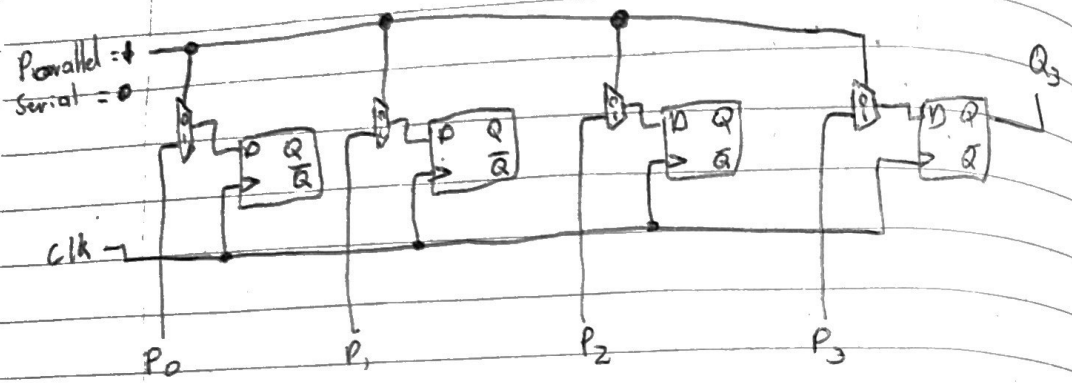
- shifting right → ÷ 2.
- shifting left → · 2.

Serial vs Parallel Load Registers

4 bit serial load register | 4 bit parallel load register



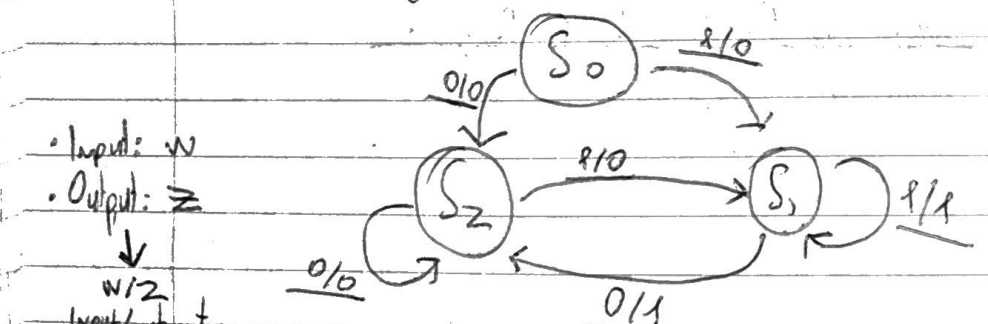
How can we do both at the same time?



- if seled bit is 1, store all parallel inputs.
- if seled bit is 0, load serial and shift bits.

Mealy Machine ...

It's state diagram:



its state table:

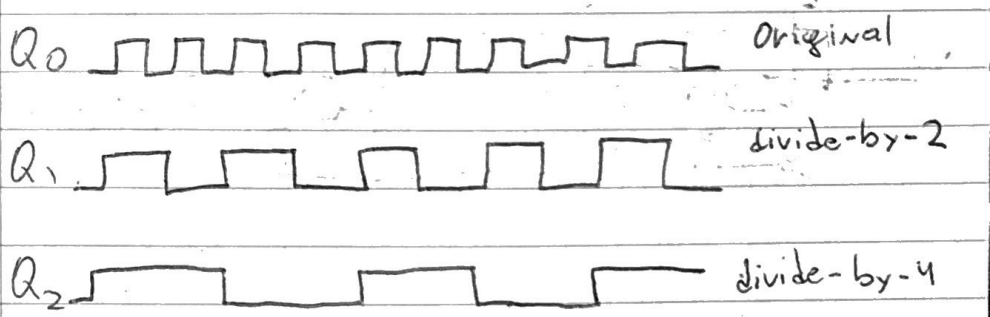
Current	Next		Output	
	w=0	w=1	w=0 z	w=1 z
S_0	S_2	S_1	0	0
S_1	S_2	S_1	1	1
S_2	S_2	S_1	0	0

Example - Mealy Machine:

Frequency Divider Circuit ⇔ divide-by-x circuit

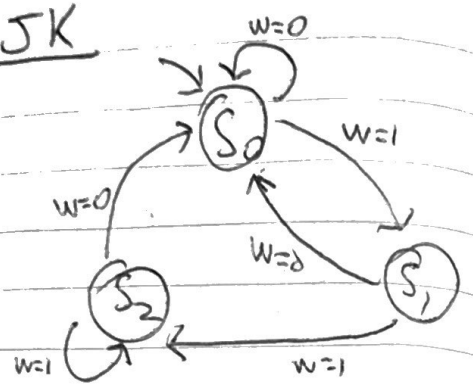
Circuit that is used to reduce the frequency of clock signal.

Ex: clk



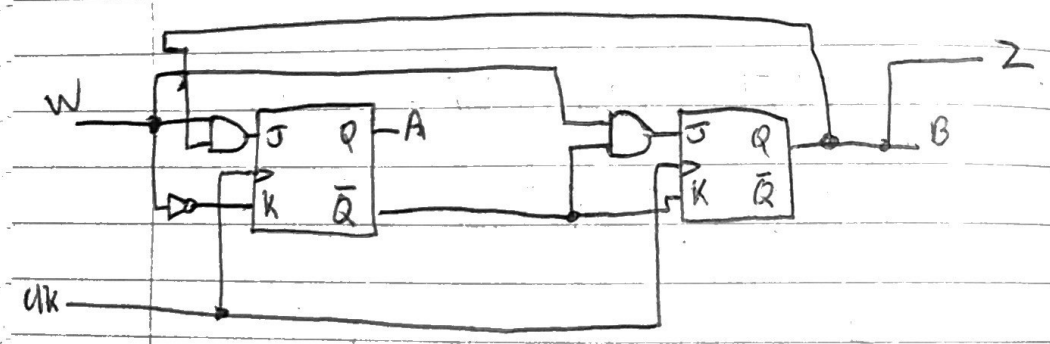
State Machine Using JK

Q_k	Q_{k+1}	JK
0	0	0 x
0	1	1 x
1	0	x 1
1	1	x 0



	Current AB	Next w=0 AB	J_AK_A	J_BK_B	Next w=1 AB	J_AK_A	J_BK_B	Output z
S_0=00	00	00	0 x	0 x	01	0 x	1 x	0
S_1=01	01	00	0 x	x 1	10	1 x	x 1	1
S_2=10	10	00	x 1	0 x	10	x 0	0 x	0
	11	xx	x x	x x	xx	xx	x x	X

$J_A = w \cdot B_k$ $K_A = \bar{w}$ $J_B = w \cdot \bar{A}_k$ $K_B = 1$ or $K_B = B_k$



Sequence Detectors

State machine circuit that monitors a bit stream to search for and identify a sequence.

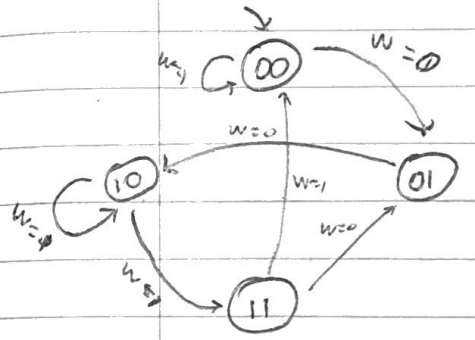
- Output is 0 until sequence is detected, when detected output is 1 for 1 clock cycle.

Ex: sequence detector for 11

In: 001001101110... (w)

Out: 000000100110... (z)

Ex: Sequence Detector for 001



AB	w=0 AB	w=1 AB	z
00	01	00	0
01	10	00	0
10	10	11	0
11	01	100	1

Alternative Approach

Store all bits in flip flops, and wait until sequence is stored.

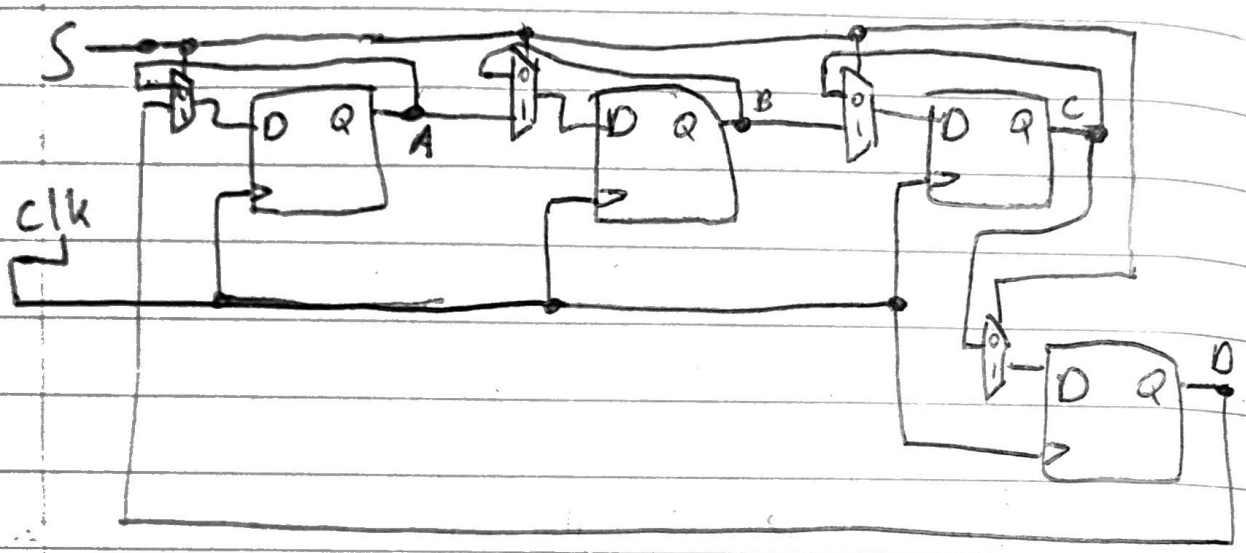
Ex: 101 → 3 flip flops; 11111 11111 11111 → 15 flip flops

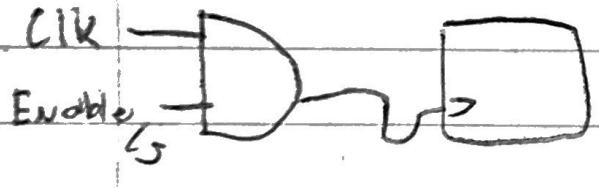
4 bit circular shift

S=0 - hold

S=1 - shift

K	A B C D
0	1 0 1 1
1	1 1 0 1
2	1 1 1 0
3	0 1 1 1





Current ABCD	Next S=0 ABCD	Next S=1 ABCD
	0 0 0 0	0 0 0 0
	0 0 0 1	1 0 0 0
	⋮	⋮
	⋮	⋮