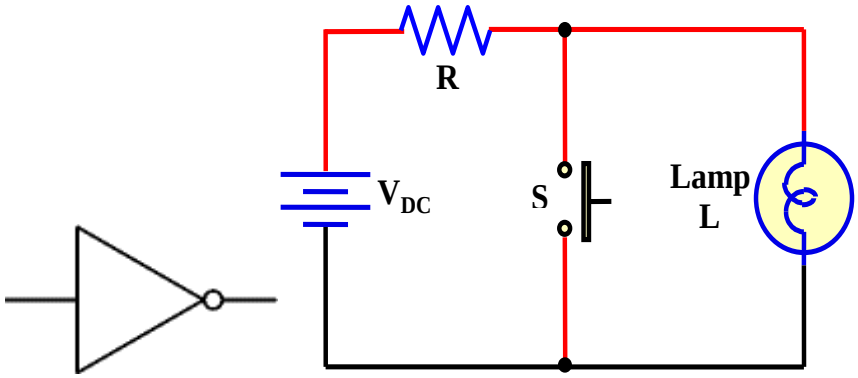
1.1 Logic Gates

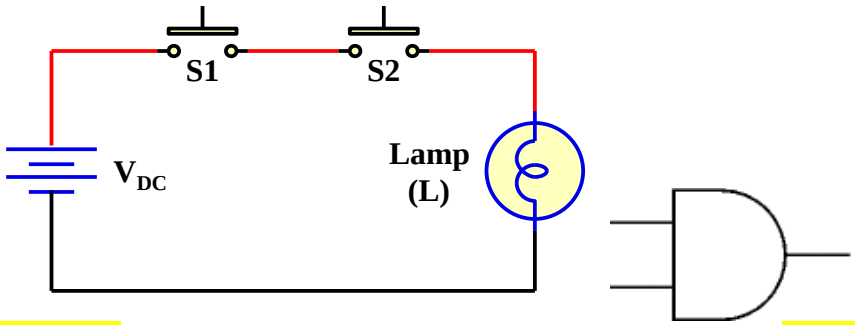
1.1.1 Basic Logic Gates

NOT B can represented as $\neg B$ and it is called a **negation.** The symbol '~' is sometimes used in place of '¬' in some logical books. While most digital electronic books just use the complement symbol. So **NOT B** is represented as B' or as $\bar{B}$. Next figure shows NOT symbol, inverter implementation of NOT, and truth table (TT) of NOT.



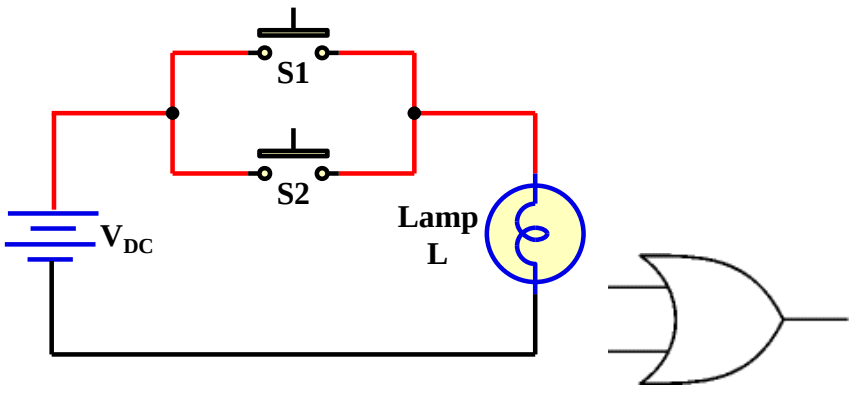
INPUTS	OUTPUT
A	NOT A
0	1
1	0

A AND B can represented as $A \wedge B$ and it is called a **conjunction.** Some logical books use the symbol '&' instead. While most digital electronic books just use the multiplication symbol. So **A AND B** is represented as $A.B$ or just as AB . Next figure shows in-series implementation of AND, AND symbol, and TT of AND.



INPUTS		OUTPUT
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

A OR B can represented as $A \vee B$ and it is called a **disjunction.** Most digital electronic books just use the addition symbol. So **A OR B** is represented as $A + B$.
Next Figure shows in- parallel implementation of OR, OR symbol, and the TT of OR.



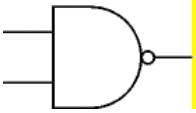
INPUTS		OUTPUT
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

1.1.2 Universal Logic Gates

A universal gate is a gate that can be used by itself, without any other logic gates, to constitute any of the basic gates. Examples of a universal gate are:

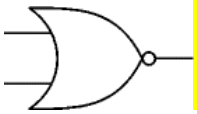
- NAND, implemented as AND-NOT
- NOR, implemented as OR-NOT

A NAND B can represented as $A \mid B$ and it is called a **Sheffer stroke.**
Next Figure shows NAND symbol and its TT.



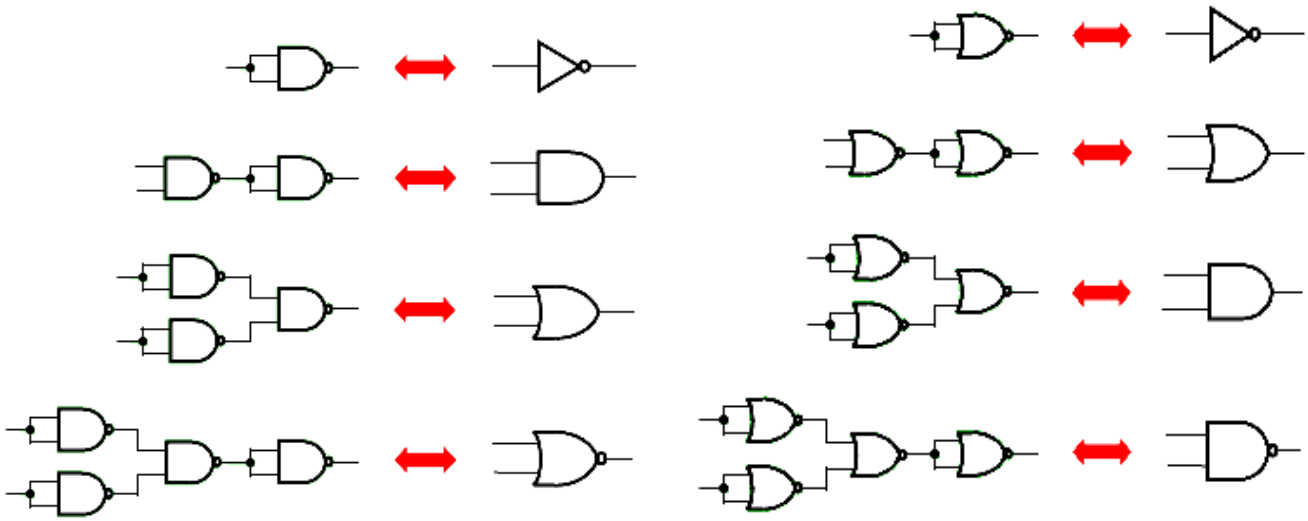
			A NAND B
A	B	T=A AND B	$= \overline{T}$
0	0	0	**1**
0	1	0	**1**
1	0	0	**1**
1	1	1	**0**

A NOR B can represented as $A \downarrow B$ and it is called a **Peirce's arrow** or **Quine dagger**.
Next Figure shows the NOR symbol and its TT



			A NOR B
A	B	T=A OR B	$= \overline{T}$
0	0	0	**1**
0	1	1	**0**
1	0	1	**0**
1	1	1	**0**

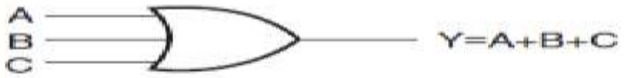
Next figure shows the universality of NAND and NOR



1.2 Logic Circuits

1.2.1 Logic expressions

Logic expressions provide a mathematical way for describing logic circuits.
Next Figure shows 3-input and 4-input ORs.



(a)

(b)

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

(c)

Observation 1.1

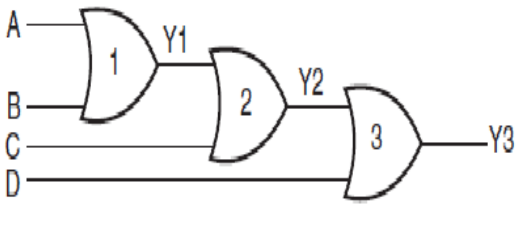
The total number of possible combinations of 1 and 0 values for n-inputs is 2^n.

Examples:

a) For 2-inputs → Number of combinations = $2^n = 2^2 = 4$.

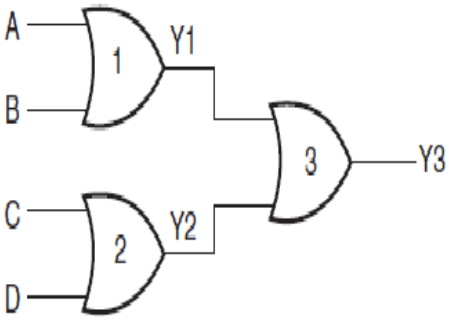
b) For 3-inputs → Number of combinations = $2^n = 2^3 = 8$.

The 4-input is implemented using many 2-input OR gates, as shown in next figure



(a)

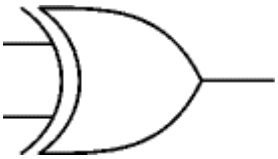
A+B+C+D as ((A+B)+C)+D,



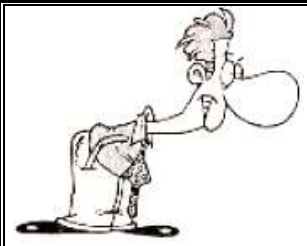
(b)

A+B+C+D as (A+B)+(C+D)

A special gate is exclusive-OR, abbreviated as **XOR** and symbolized as ⊕.
Next figure shows the **XOR** gate symbol, then the TT of **XOR**.



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

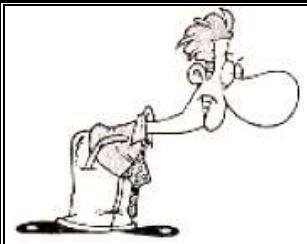


Outside Scope of COAT 1.1

Manufacturing Families of Digital Logic

Logic families classified based on manufacturing technologies to:

- "Diode Logic"(DL)
- "Resistor Transistor Logic"(RTL)
- "Diode Transistor Logic"(DTL)
- "High threshold Logic"(HTL)
- "Transistor Transistor Logic"(TTL)
- "Integrated Injection Logic"(I2L)
- "Emitter coupled logic"(ECL)
- "Metal Oxide Semiconductor Logic"(MOS)
- "Complementary Metal Oxide Semiconductor Logic"(CMOS)



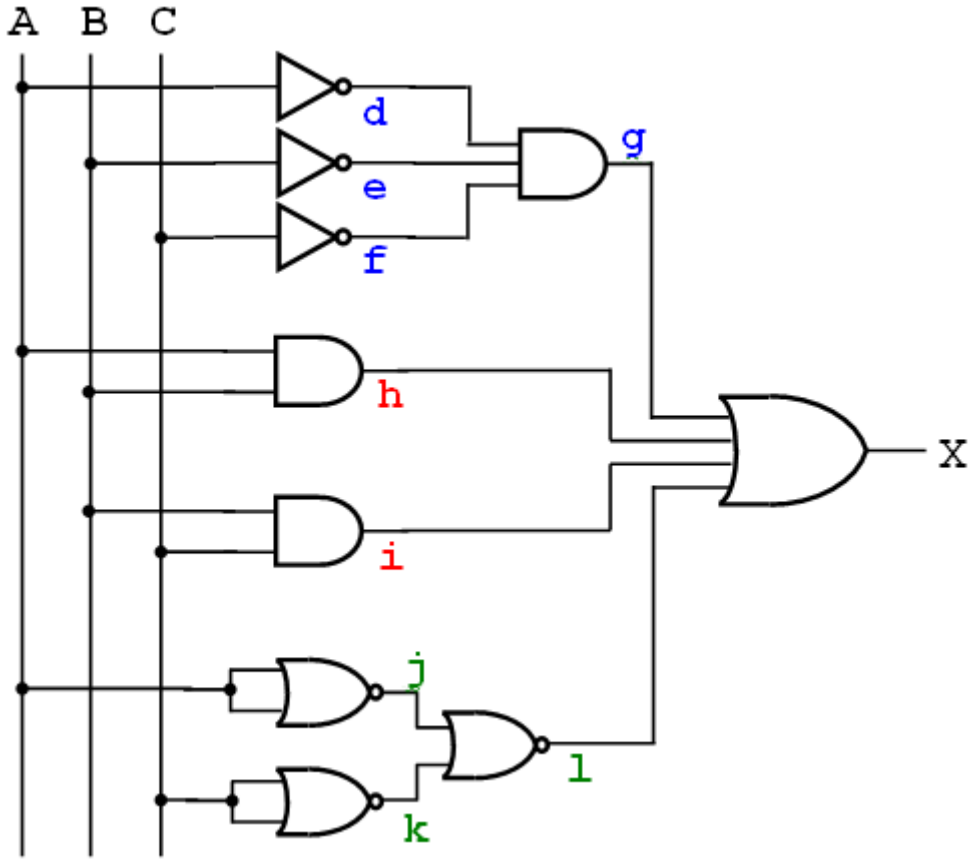
Outside Scope of COAT 1.2

Classification of Integrated Circuits

- "Small Scale Integration" (SSI) upto 10 transistors. Such as AND, OR, NOT gates.
- "Medium Scale Integration" (MSI) upto 100 transistors. Such as adders, decoders, counters, flip-flops and multiplexers.
- "Large Scale Integration" (LSI) upto 1,000 transistors. Such as I/O chips, ALUs.
- "Very-Large Scale Integration" (VLSI) upto 10,000 transistors. Such as PLDs.
- "Super-Large Scale Integration" (SLSI) upto 100,000 transistors .Such as microprocessor chips, micro-controllers.
- "Ultra-Large Scale Integration" (ULSI) more than 1 million transistors. Used in CPUs, GPUs, and FPGAs.

1.2.2 Sum of product

How to describe a logic circuit such as that shown in next figure? What is the standard mathematical representation to describe relation between inputs(A,B,C) and output(X)?



Actually there are many proposed canonical normal forms such as

- "Sum of Products" (SoP), a.k.a. conjunctive normal form (CNF)
- "Product of Sums" (PoS), a.k.a. disjunctive normal form (DNF)

A product, or a minterm, is an expression formed only with AND on variables. A sum, or a maxterm, is an expression formed only with OR on variables. CNF is dual to DNF through De Morgan laws. We shall stick to **CNF** /**SoP** throughout **COAT**.

First, we need to calculate the intermediate temporary nodes in a TT,

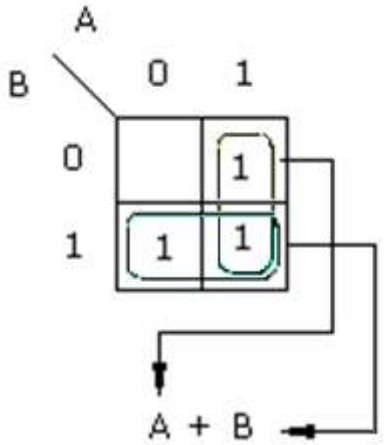
Input			Temporary nodes									Ouptut
A	B	C	d	e	f	g	h	i	j	k	l	X
0	0	0	1	1	1	1	0	0	1	1	0	1
0	0	1	1	1	0	0	0	0	1	0	0	0
0	1	0	1	0	1	0	0	0	1	1	0	0
0	1	1	1	0	0	0	0	1	1	0	0	1
1	0	0	0	1	1	0	0	0	0	1	0	0
1	0	1	0	1	0	0	0	0	0	0	1	1
1	1	0	0	0	1	0	1	0	0	1	0	1
1	1	1	0	0	0	0	1	1	0	0	1	1

The output may now be described in CNF /SoP as

$$\bar{A}\bar{B}\bar{C} + \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$$

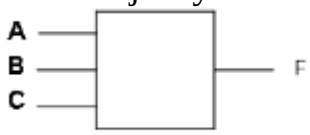
1.2.3 Minimization and Simplification Karnaugh-maps, or K-maps.

The map for a 2-input OR gate looks like this:



Example: design the majority circuit for 3 inputs.

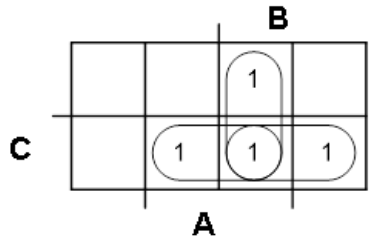
A logic circuit has three inputs, labelled X_2, X_1, and X_0. The single output, labelled F, is required to be 1 if a majority of the inputs are at 1.



Step 1: convert the problem statement into a TT:

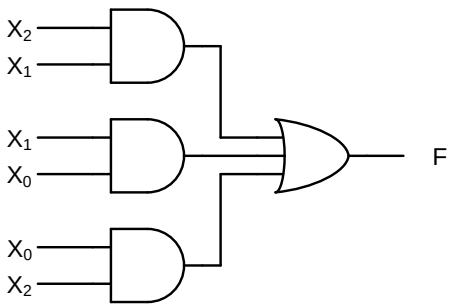
X_2	X_1	X_0	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Step 2: Simplify Boolean expression for the function using Karnaugh map

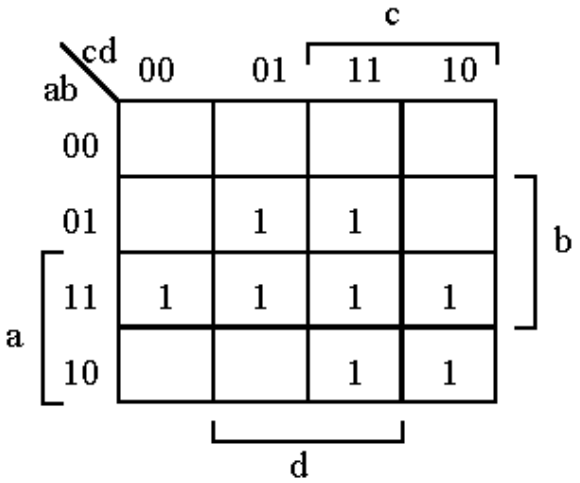


$$A.B.\overline{C} + A.\overline{B}.C + \overline{A}.B.C + A.B.C$$

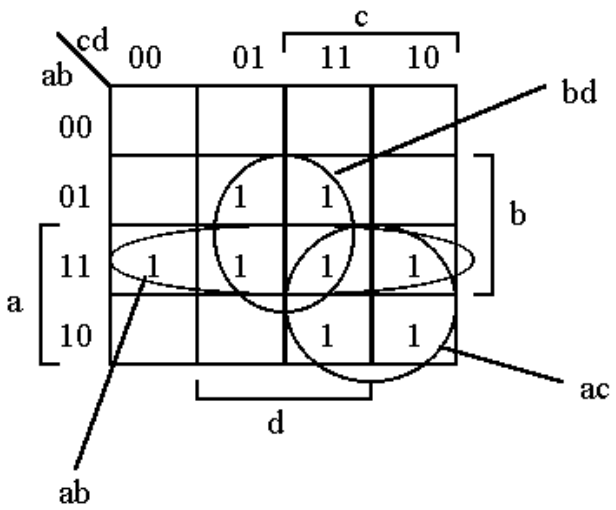
Step 3 is to give the corresponding logic diagram.
Boolean equation in sum-of-products form.



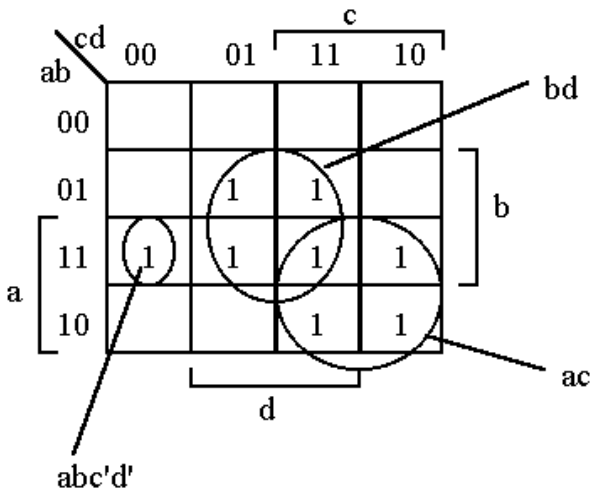
Example: Consider
q = a'bc'd + a'bcd + abc'd' + abc'd + abcd + abcd' + ab'cd + ab'cd'



Grouping the 1s together results in the following.



The expression for the groupings above is q = bd + ac + ab
This expression requires 3 2-input **and** gates and 1 3-input **or** gate.
We could have accounted for all the 1s in the map as shown below, but that results in a more complex expression requiring a more complex gate.



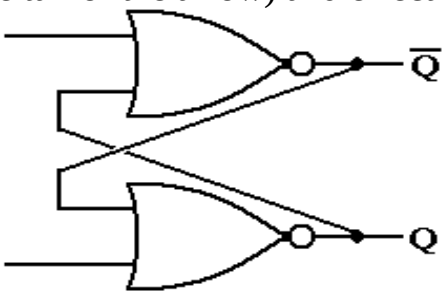
The expression for the above is **bd + ac + abc'd'**. This requires 2 2-input **and** gates, a 4-input **and** gate, and a 3 input **or** gate. Thus, one of the **and** gates is more complex (has two additional inputs) than required above. Two inverters are also needed.

1.3 Sequential circuits

Combinational circuits, presented until now, do not have the ability to store information. Sequential circuits, such as Flip-Flops (FF) and registers, can store information.

1.3.1 LATCH

The most primitive memory element is the latch. An NOR latch consists of a cross-coupled pair of NOR gates as shown in next figure.
The arrows indicate cause (the tail of the arrow) and effect (the head) relationships.

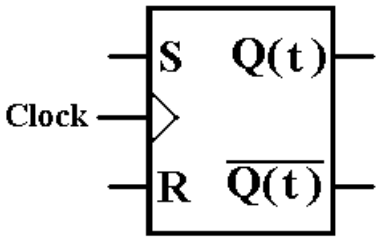


1.3.2 Flip-Flops

Circuits constructed from such *unclocked* devices as latches are referred to as *asynchronous circuits*. The design of asynchronous circuits is much more involved than that of *synchronous* circuits, in which changes of the memory element outputs are synchronized by a *clock*.

The SR Flip–Flop

SR flip–flop is defined based on SR latch.



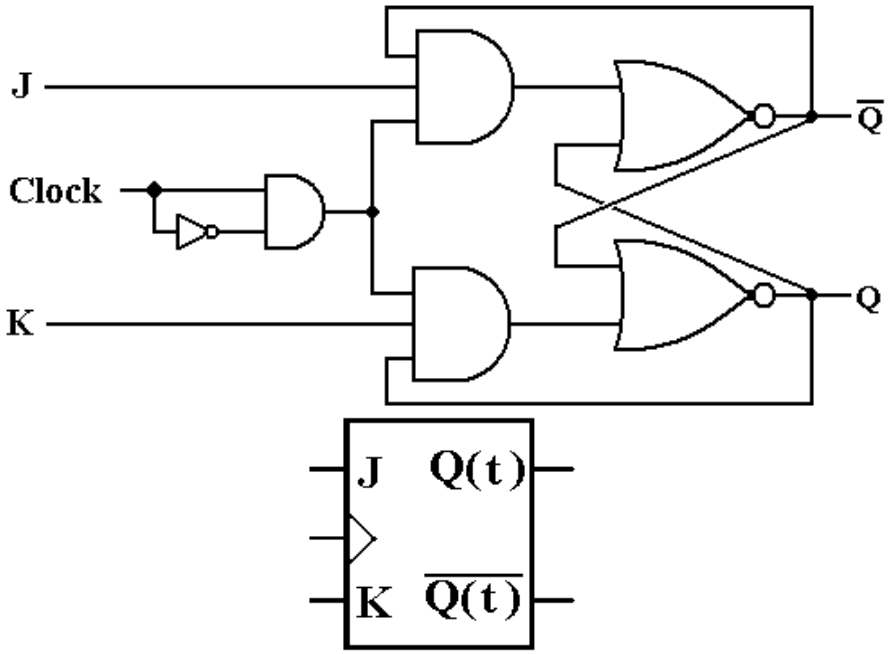
Characteristic Table

S	R	Q(t + 1)
0	0	Q(t)
0	1	0
1	0	1
1	1	ERROR

Excitation Table

Q(t)	Q(t + 1)	S	R
0	0	0	d
0	1	1	0
1	0	0	1
1	1	d	0

The JK Flip–Flop



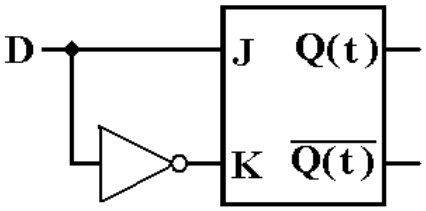
Characteristic Table

J	K	Q(t + 1)
0	0	Q(t)
0	1	0
1	0	1
1	1	$\overline{Q}(t)$

Excitation Table

Q(t)	**Q(t + 1)**	**J**	**K**
0	0	0	d
0	1	1	d
1	0	d	1
1	1	d	0

The D Flip–Flop



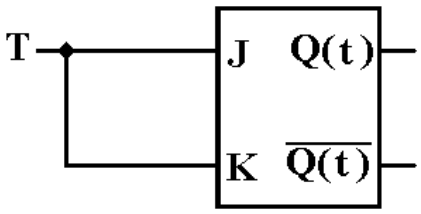
Characteristic Table

D	Q(t + 1)
0	0
1	1

Excitation Equation

D = Q(t + 1)

The T Flip–Flop



Characteristic Table

T	Q(t + 1)
0	Q(t)
1	$\overline{Q}$ (t)

Excitation Table

Q(t)	Q(t + 1)	T
0	0	0
0	1	1
1	0	1
1	1	0

Excitation Equation

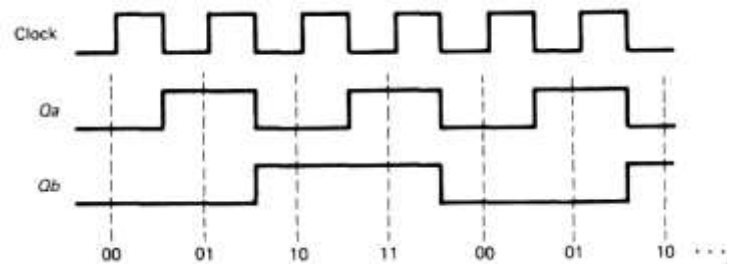
$T = Q(t) \oplus Q(t + 1)$

1.3.3 Counters

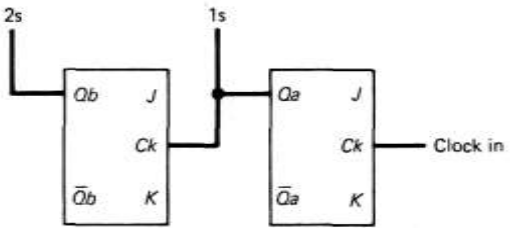
A counter is a sequential circuit that goes through a set of given states on successive clock cycles. (e.g., 0 through 7, 0 through 15, and so on).

If a *J-K* flip-flop is used as a *T* flip-flop (by tying *J* and *K* high), the output changes state on each clock cycle. Therefore, after two clock cycles the output of the flip-flop completes one cycle. If the output of this flip-flop were then used as the clock input to a second *T* flip-flop, the output of the second flip-flop would cycle at half the rate of the first flip-flop. Each additional flip-flop added in this manner would cycle at half the rate of the preceding flip-flop.

Timing Diagram for 2-Bit Ripple Counter



2-Bit Ripple Counter

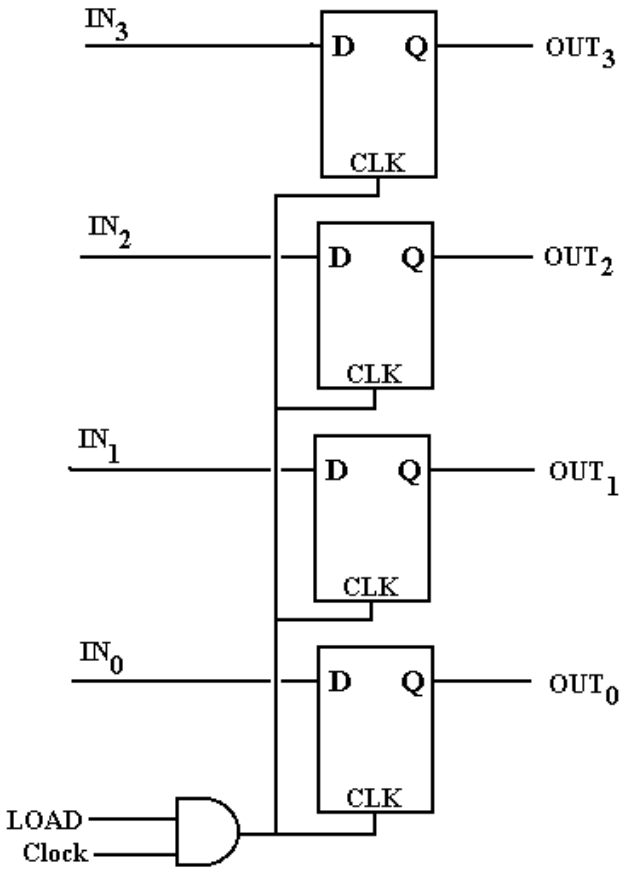


As can be seen, by considering the output of the first flip-flop *(Qa)* as the Is digit and the output of the second flip-flop *(Qb)* as the 2s digit, we have a simple binary counter sequencing from 0 through 3 and then repeating. Asynchronous ripple counters are often referred to as *ripple counters*, for short.

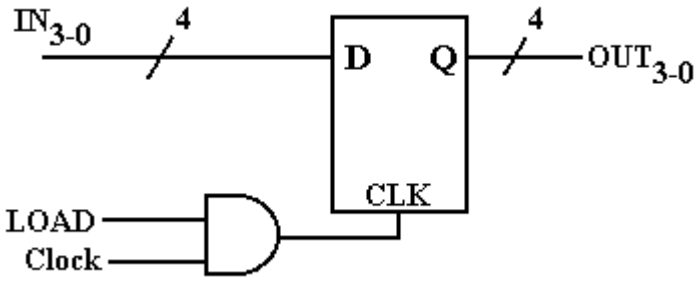
1.3.4 Registers

A register is a storage device capable of holding binary data. It is best viewed as a collection of flip-flops, usually D flip-flops. To store N bits, a register must have N flip-flops, one for each bit to be stored. We show a design for a four-bit register with a synchronous LOAD.

In this example, the 4-bit register is implemented by four D flip-flops.
Note the input CLK comes from an AND gate that puts out the logical AND of the system clock (Clock) and the LOAD signal. When LOAD is 0, the flip-flops are cut off from the input and do not change state in response to the input. The design calls for LOAD to be 1 for almost one clock pulse, so that the system clock and LOAD are both high for 1/2 clock cycle. At this time, the register is loaded.



The next figure shows a short-hand notation used when drawing a 4-bit register that contain a number of flip-flops identically configured.



Common Registers

1. AR: Address register. This register holds the memory address. Only the least-significant 12 bits are significant. The output of the register is directly connected to the address lines of memory.
2. PC: Program counter. Only the least significant 12 bits are significant.
3. DR: Data register. This register holds data for ALU operation. The output of the register is directly connected to the ALU on data line 0.
4. AC: The accumulator. The output of the register is directly connected to ALU data line 1.
5. IR: The instruction register. This register holds the current instruction
6. TR: Temporary register. Holds temporary data.

The inputs of these registers, except the AC, are connected to the bus. AC receives its input from the ALU. Each of these registers have the following control lines: LD, INR, and CLR. Ld will load the data into the register, INR will increment the register, and CLR will clear the register.

Memory

Memory has 16 bit input data lines connected to the bus, and 16 bit output lines connected at bus address 7. The memory is 4096 words long and is word addressed. The address lines are 12 bits wide (2^{12}=4096) and is connected to the address register AR.

Input/Output

Input/output is performed via two 8 bit registers, INPR and OUTR. INPR is connected to the ALU, while the input lines of OUTR are connected to the bus. I/O is request based. When an input character is available, a flip-flop FGI is to be turned on by the i/o unit. This flag will be turned off by the CPU when the input is read. Similarly, when the output is written to to OUTR, a flip-flop FGO will be turned off by the CPU. This flop-flop can be turned on by external hardware.

Interrupt Structure

The CPU has IEN flip-flop that enables interrupts. When an interrupt is received, the CPU stores the current PC at location 0 and jumps to location 1. Typically there will be an unconditional branch to the service routine. The routine will branch back to the address in location 0. A flip flop R is used to initiate interrupt service. It is turned on when the interrupt is acknowledged and is turned off once the PC has been saved and the PC has been set to 1.

Instruction Set

The opcode consists of 4 bits while the remaining 12 bits are used either to address memory or encode inherent (non-memory) instructions.

The instruction sets consists of 6 memory addressed instructions. The most significant bit determines the type of addressing, either direct or indirect. The other 3 bits determine the instruction as follows:

000: AND

001: ADD

010: LDA

011: STA

100: BUN (Branch unconditional)

101: BSA (Branch and save return address)

110: ISZ (Increment memory and skip next instruction if zero)

RTL Descripion [Mano- RTL]

One way to describe and synthesize the machine is to describe the action of the computer at each clock cycle. For example, consider the LDA instruction. Here is the sequence of events

T0	Transfer PC to AR	AR ← PC
T1	Fetch instruction and increment PC	IR ← Memory, PC ← PC+1
T2:	Decode and xfer address	AR ← IR(11 -- 0)
T3	Indirection if needed	AR ← Memory, if IR(15) =1
T4:	Fetch data	DR ← Memory
T5:	Move to AC	AC ←DR, DONE

Note based on the above timing information, we first generate a sequencer which will have lines T0, T1, T2, etc. A counter (register) connected to a decoder generates this sequencer. The control module will then turn on the following signals:

T0	Connect PC to BUS, Turn on LD line of AR
T1	Connect Mem to Bus, Turn on LD line of AR and INC line of PC
T2:	Connect IR to bus, Turn on LD line of AR
T3	If I(15) connect Mem to Bus and turn on LD line of AR
T4:	Connect Mem to Bus, Turn on LD line of DR
T5:	Set ALU to pass through and turn on LD line of AC, Turn on CLR line of Sequence counter

1.4 Programmable Logic Devices (PLDs)

There are many PLDs such as :

- "Programmable Read Only Memory"(PROM) has a fixed AND array(constructed as a decoder) and programmable connections for the output OR gates array. The PROM implements Boolean functions in sum-of-minterms form.
- "Programmable Array Logic"(PAL) device has a programmable AND array and fixed connections for the OR array.
- "Programmable Logic Array"(PLA) has programmable connections for both AND and OR arrays. So it is the most flexible type of PLD.
- "Complex Programmable Logic Device" (CPLD)
- "Field Programmable Gate Array" (FPGA).

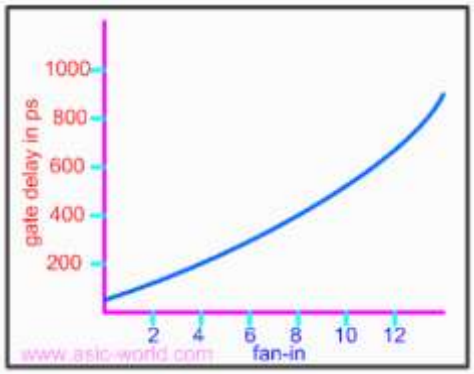
PLA (Programmable Logic Array):

In PLAs, a number (***k***) of AND gates is usedwhere ***k < 2n,*** (***n*** is the number of inputs).
Each of the AND gates can be programmed to generate a product term of the input variables and does not generate all the minterms as in the ROM.
The AND and OR gates inside the PLA are initially fabricated with the links (fuses) among them.
The specific Boolean functions are implemented in sum of products form by opening appropriate links and leaving the desired connections.

1.5 Physical considerations

1.5.1 Fan – in:

Fan-in is the number of inputs a gate has, like a two input AND gate has fan-in of two, a three input NAND gate as a fan-in of three. So a NOT gate always has a fan-in of one. The figure below shows the effect of fan-in on the delay offered by a gate for a CMOS based gate. Normally delay increases following a quadratic function of fan-in.

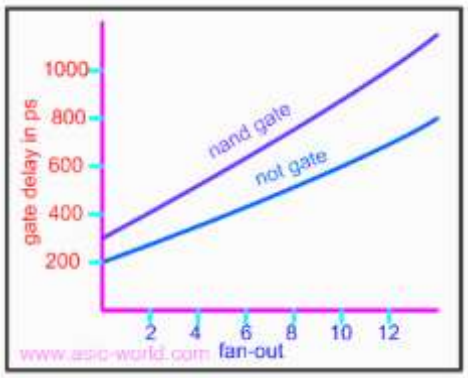


1.5.2 Fan – out:

The number of gates that each gate can drive, while providing voltage levels in the guaranteed range, is called the standard load or fan-out. The fan-out really depends on the amount of electric current a gate can source or sink while driving other gates. The effects of loading a logic gate output with more than its rated fanout has the following effects.

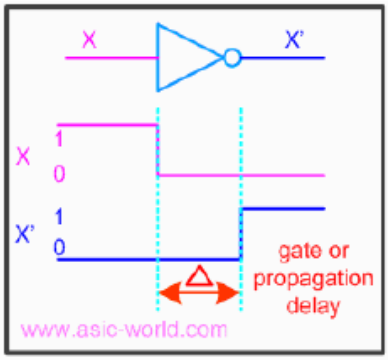
In the LOW state the output voltage VOL may increase above VOLmax.
In the HIGH state the output voltage VOH may decrease below VOHmin.

The operating temperature of the device may increase thereby reducing the reliability of the device and eventually causing the device failure.
Output rise and fall times may increase beyond specifications
The propagation delay may rise above the specified value.
Normally as in the case of fan-in, the delay offered by a gate increases with the increase in fan-out.



1.5.3 Gate Delay

Gate delay is the delay offered by a gate for the signal appearing at its input, before it reaches the gate output. The figure below shows a NOT gate with a delay of "Delta", where output X' changes only after a delay of "Delta". Gate delay is also known as propagation delay.
Gate delay is not the same for both transitions, i.e. gate delay will be different for low to high transition, compared to high to low transition. Low to high transition delay is called turn-on delay and High to low transition delay is called turn-off delay.



Outside Scope of COAT 1.3

Other Physical Considerations

- Wire Delay.
- Noise Margin.
- Power Dissipation.
- Skew.
- Voltage threshold