Digital Logic: Implementation

D.K. Sharma, D. Chakraborty, K. Chatterjee, B.G. Fernandes,
J. John, P.C. Pandey, N.S. Shiradkar, K.R. Tuckley

EE Department
IIT Bombay, Mumbai

Book References

"Digital Fundamentals" by Thomas L. Floyd, Pearson Education.
"Digital Design" by M. Morris Mano, Pearson Education.

January 17, 2024

Review

In the previous lecture, we had learnt about Boolean algebra, its axioms and theorems and their use in expressing and manipulating logic expressions.
We had also learnt how to minimize logic expressions to simplify their implementation.

In this lecture, we discuss:

- combinational and sequential logic,
- how digital logic is implemented in hardware,
- number systems using binary and hexadecimal representations,
- digital to analog conversion (DAC) and analog to digital conversion (ADC).

Book References

"Digital Fundamentals" by Thomas L. Floyd, Pearson Education.
"Digital Design" by M. Morris Mano, Pearson Education.

Combinational and Sequential Logic

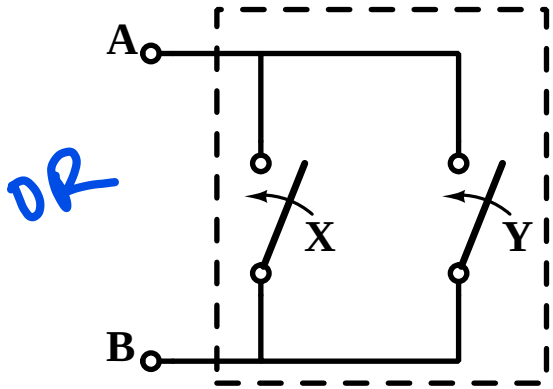
- The logic functions we have seen up to now produce an output as a function of the combination of current values of input variables. Logic of this kind is known as **Combinational Logic**.
- However, some applications require the output to generate a **sequence** of values as a function of a **sequence** of values at the input. Logic functions of this kind constitute **sequential** logic.
- Sequential logic requires
  - A way to mark points in time which separate the "previous" value from the "current value" on any any input or output, and
  - means of storing information about previous values of data (**memory**).
- The signal, which marks points in time separating different elements of the input and output sequences, is known as the **clock**.

Implementing Digital Logic

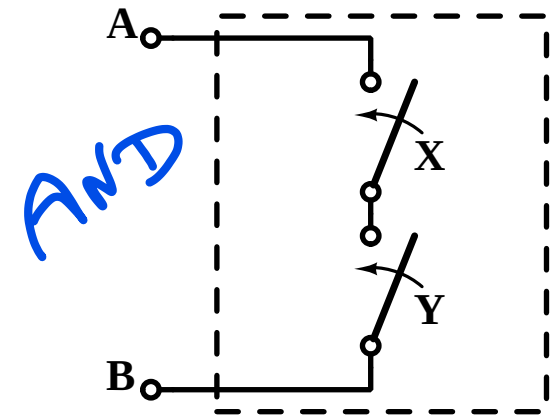
How can we implement Digital Logic in hardware?
One possibility is to use electronic devices as switches. Let us see how switches may be used to implement logic functions.

First, consider two switches connected in parallel across nodes A and B.

- Obviously, node A is connected to node B if either or both switches are ON.
- This can be used to implement the logical OR.



- When the switches are connected in series, node A is connected to node B only if *both* switches are ON.
- This can be used for implementing AND logic.

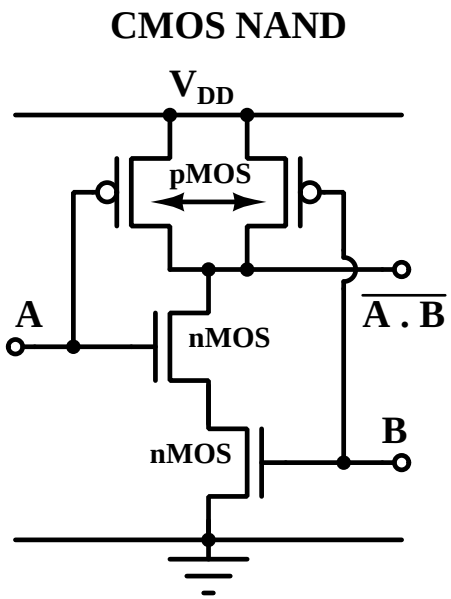


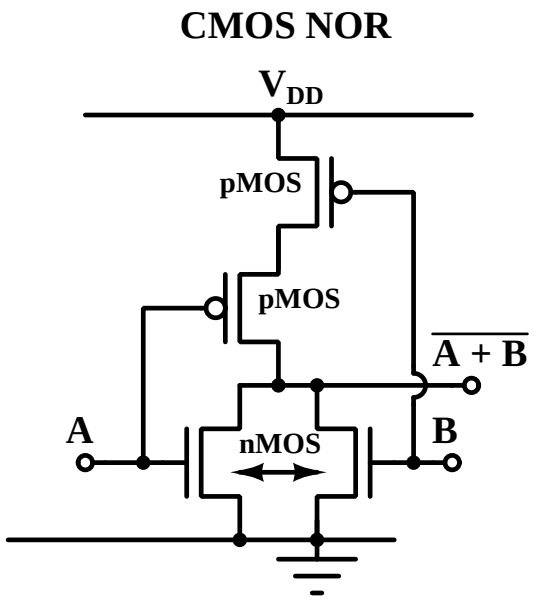
Logic Gates

- In the early days, relays were used as controllable switches to implement logic functions.
- More recently, logic circuits have been implemented using diodes, bipolar transistors and MOS transistors.
- These days, digital circuits are implemented using a combination of n and p channel MOS transistors. This is known as complementary MOS or CMOS logic.
- TTL (transistor-transistor-logic) was the dominant technology for a long time and is still in use. It used bipolar transistors and operates with a 5 V supply. Most TTL gates have type numbers of the form 74xx.
- CMOS B series uses complementary MOS transistors and can operate with supply voltages of 3 to 15V. Typical type numbers are CD 4xxx.
- A TTL compatible logic series implemented with CMOS is also available and carries type numbers like 74Cxx.

CMOS Logic Gates

CMOS gates implement switch based logic that we had discussed earlier. A '1' at the input turns the nMOS ON and the pMOS OFF, while a '0' at the input turns the nMOS OFF and the pMOS ON.



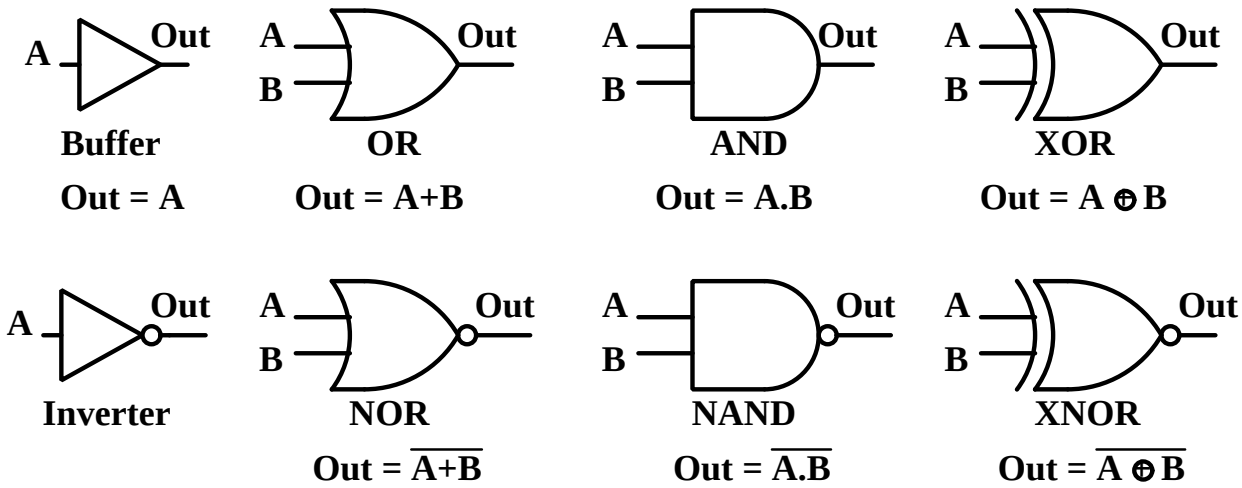


NAND output is pulled to ground when **both** inputs are '1'.
It is pulled up to V_{DD} when **either or both** inputs are '0'.

NOR output is pulled to ground when **either or both** inputs are '1'.
It is pulled up to V_{DD} when **both** inputs are '0'.

Logic Gates and symbols

Irrespective of the technology used for implementing the logic, standard symbols are used to represent logic functions.
The term "Logic Gate" is used for each logic function implemented in hardware. The figure below shows the symbols used for different types of logic gates.



If you leave an input to a TTL gate unconnected, it is taken as a '1'
Unused inputs to CMOS gates must **never** be left unconnected - this may result in heavy current flow and may damage the IC.

Some commonly used logic gates

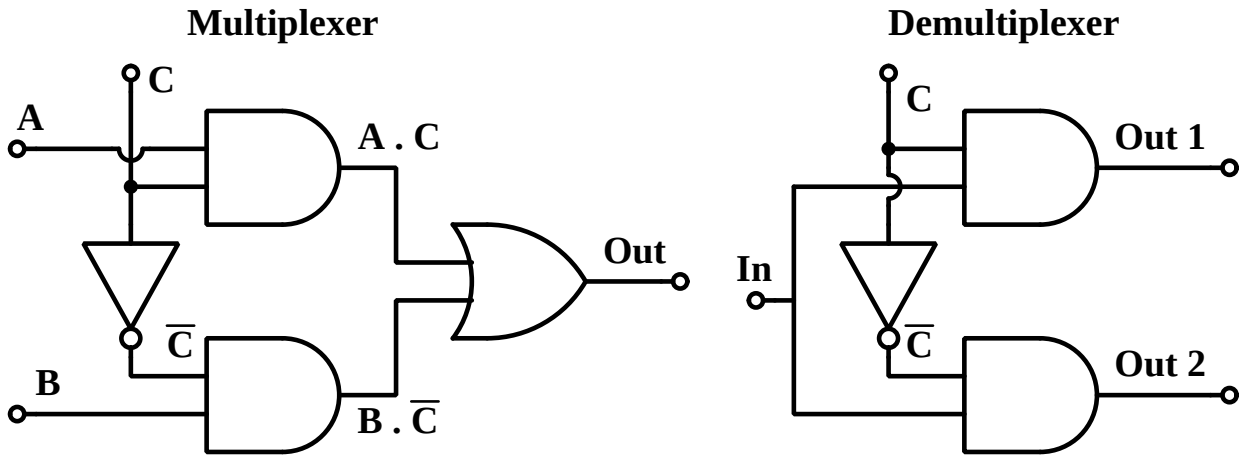
Type numbers of TTL gates typically begin with 74 which is followed by a sub type of TTL like low power Schottky (LS) or High speed (H) and a number denoting the logic function.

CMOS B series has type numbers like CD 4xxx, while the TTL compatible CMOS logic gates have type numbers like 74Cxx.

Logic Function	No of gates on chip	TTL implementation	CMOS B Series
Inverter	6	7404	4069
2 input AND	4	7408	4012
2 input NAND	4	7400	4011
4 input AND	2	7421	4082
4 input NAND	2	7420	4012
2 input OR	4	7432	4071
2 input NOR	4	7402	4001
2 input XOR	4	7486	4070

Multiplexing and Demultiplexing

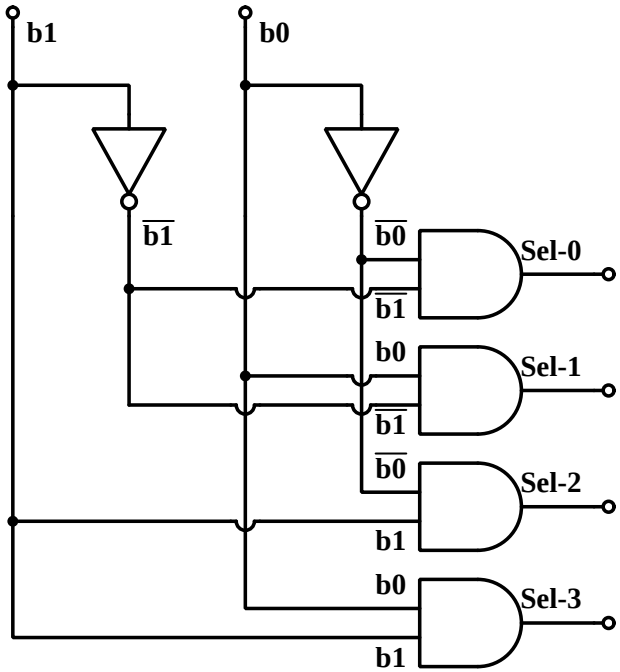
- We often want to select one out of two bits A and B depending on the value of C being '1' or '0'. The function $C \cdot A + \overline{C} \cdot B$ evaluates to A when C = '1' and to B when C = '0'.
- This can be used to select one out of two bits and put the selected bit on the output line. Circuits of this kind are called **multiplexers.**



- **Demultiplexers** do the opposite. The input data comes on a single line and depending on a line number code, it is put on one out of multiple output lines.

Decoding

What can we do if we want to multiplex from or demultiplex to more than two lines?
– We use multi-bit **decoders** which provide select lines to replace C and $\overline{C}$ in the previous example.

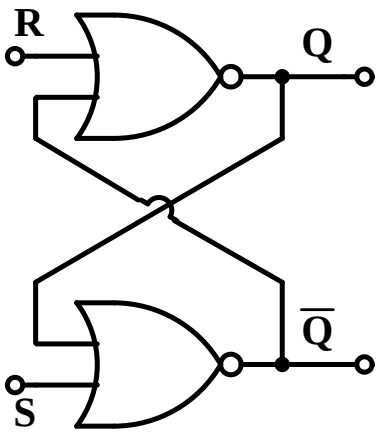


The circuit on the left shows a 2 bit decoder which provides 4 Select lines. The select signals replace C and $\overline{C}$ when we have more than 2 lines to multiplex or demultiplex.

Apart from their use in multiplexers and demultiplexers, these are widely used for address decoding and enabling selected devices connected to processors etc.

Storage Elements: RS Latch

Digital circuits require storage elements in addition to combinational logic functions. Consider the cross connected NOR gates in the circuit shown below.



For R = '0' and S = '0', the circuit can remain in one of these two states:

1. Q = '0' and $\overline{Q}$ = '1', (reset state)
2. Q = '1' and $\overline{Q}$ = '0'. (set state)

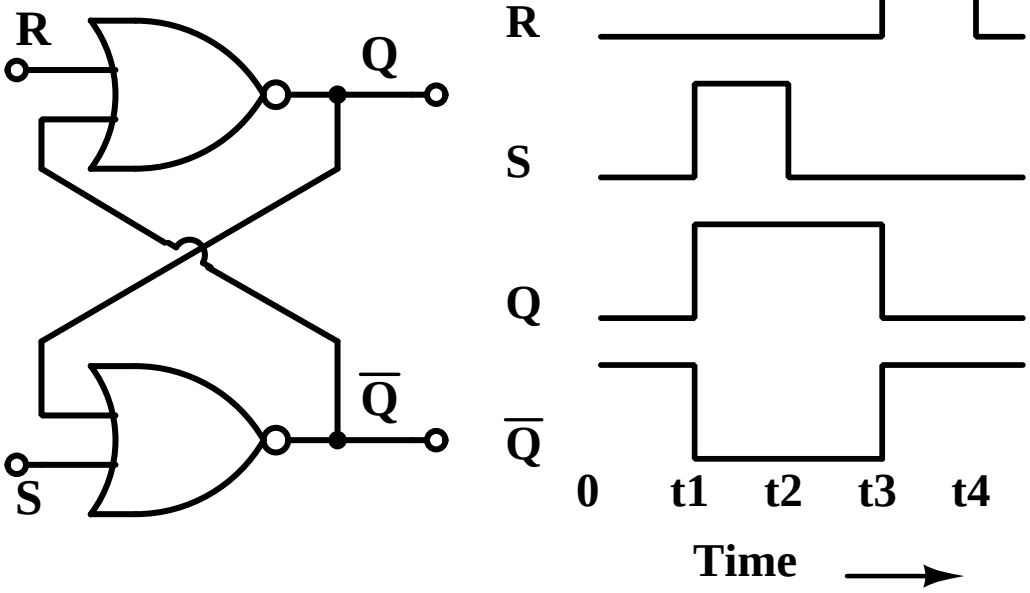
It is easy to see that for $R = 0$, $S \rightarrow$ '1' will force the circuit into its 'set' state (Q = 1, $\overline{Q}$ ='0').

$$S \rightarrow \text{'1'} \Rightarrow \overline{Q} = \text{'0'}, \quad R = \text{'0'}, \ \overline{Q} = \text{'0'} \Rightarrow Q = \text{'1'}$$

Now even if S returns to '0', the circuit will remain in its set state. Thus this circuit "remembers" that S had been '1' in the past.

Similarly, R = '1' while S = '0' will force the circuit in its 'reset state' and it will remain in reset state even after R returns to '0'.

RS Latch: set and reset action



Initially R = S = '0' and the latch is in reset (Q = '0', $\overline{Q}$ = '1') state.

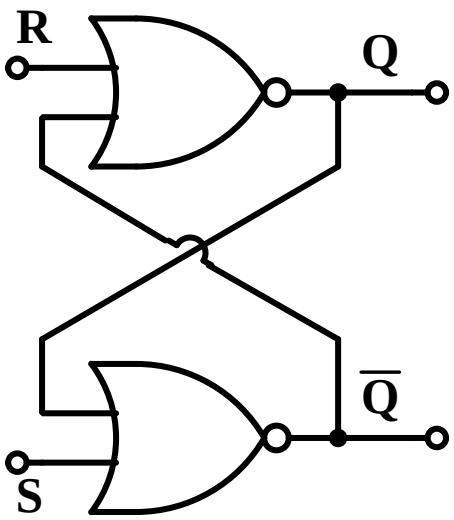
At t1, S goes to '1' and it forces $\overline{Q}$ to '0' unconditionally. Since R = '0' and $\overline{Q}$ = '0', Q goes to '1'.

At t2, S returns to '0', and the latch remains in set (Q = '1', $\overline{Q}$ = '0') state.

At t3, R goes to '1'. It forces Q to '0' unconditionally. Since S = '0' and Q = '0', $\overline{Q}$ goes to '1'.

at t4, R returns to '0'. With R = 0, S = 0, the latch remains in reset (Q = '0', $\overline{Q}$ = '1') state.

RS Latch: set and reset action



Since the S input sets Q to '1', it is called the 'Set' input.
R input returns Q to '0' and is termed as the 'Reset' input.
The latch retains its set (Q = '1', $\overline{Q}$ = '0') state even after S returns to '0'. Thus, it "remembers" that S had gone to '1' before it returned to '0'.

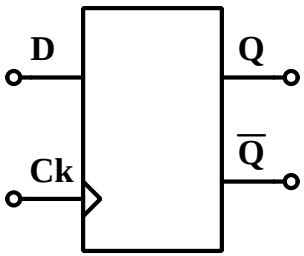
Similarly, the latch retains its reset (Q = '0', $\overline{Q}$ = '1') state even after R returns to '0'. Thus it "remembers" that R had gone to '1' before it returned to '0'.

If Set and Reset are simultaneously applied, both Q and $\overline{Q}$ are forced to '0'.
The latch will go to set or reset state when R and S return to '0', depending on which input is returned to '0' last.

Other storage circuits: D flipflop

There are other useful circuits which show a ‘memory’.

Many of these are used in sequential circuits with a clock.
Their action occurs whenever the clock has a specified transition, say from ‘0’ to ‘1’. This transition is known as the **active edge of the clock**.

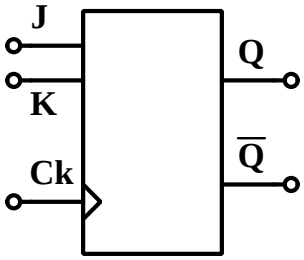


D flipflop: This is circuit with a ‘D’ (Data) input and a clock. At the active clock edge, it copies the value of D to its Q output and the complement appears at $\overline{Q}$.

The symbol for a D flipflop is shown above. The little triangle at the clock input shows that the flipflop transfers D to Q on the rising edge of the clock. A little circle is placed with the triangular symbol if the active edge is the falling edge of the clock.

Many circuit diagrams omit the indication for the active edge.

Other circuits with memory: JK flipflop



JK flipflop:
This circuit has two inputs J and K –
These act similarly to S and R inputs in the RS latch.

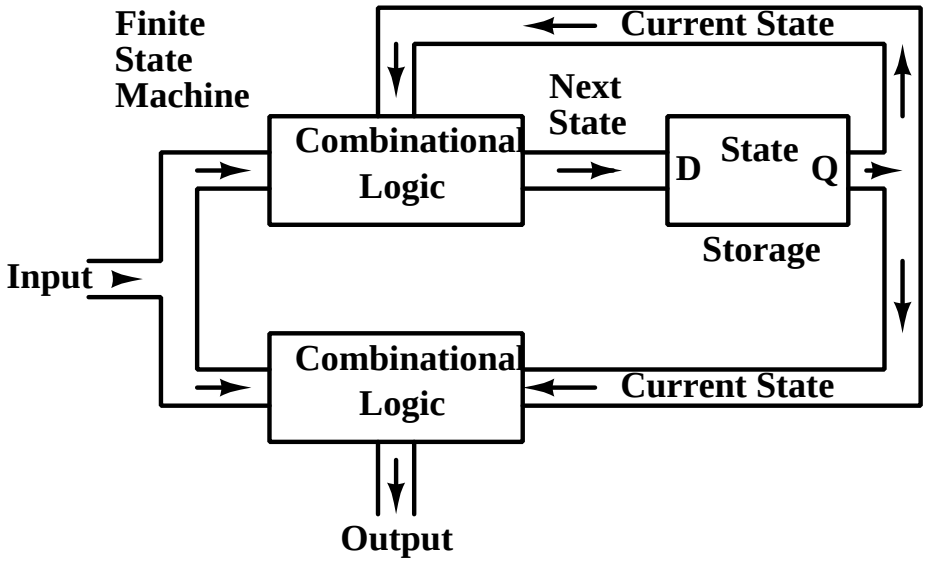
At the active edge of the clock,
If J = '0' and K = '0', the flipflop outputs remain unchanged.
If J = '1' and K = '0', Q goes to '1' and $\overline{Q}$ goes to '0'.
If J = '0' and K = '1', Q goes to '0' and $\overline{Q}$ goes to '1'.
If J = '1' and K = '1', it complements the values at Q and $\overline{Q}$.
(This and the presence of clock distinguish it from RS latch).

Toggle flipflop: In this, the J and K inputs are tied together and are used as the T input.
When T = '0', the flipflop outputs Q and $\overline{Q}$ remain unchanged.
When T = '1', it toggles *ie* complements the values at Q and $\overline{Q}$ at the active clock edge.

Finite State Machines

In a sequential circuit, the output sequence is a function of input sequence.
The sequences between which we want to establish a functional relationship have to be of finite length to be implementable.

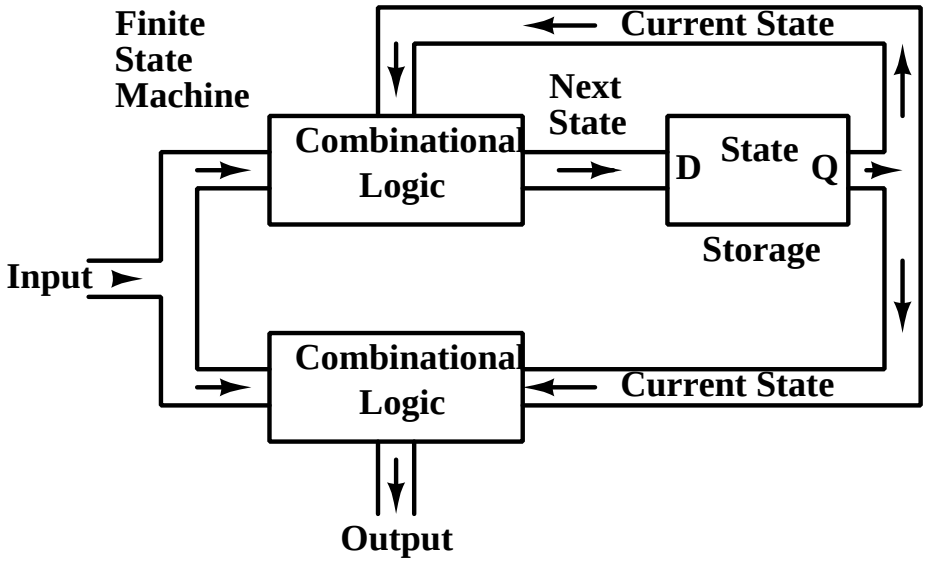
- A new set of input values are presented during each new clock period.
- It is not necessary to store all the past values of input variables.



A (possibly multi-bit) **state** is stored, which is a digital value with sufficient information about the past inputs so that the current element of the output sequence can be generated as a **combinational** function of the state and the current inputs.

Finite State Machines

- In each clock period, as fresh inputs arrive, a new value of **state** is generated which is a combinational function of the current value of state and current inputs.
- The state is thus a recursive function of previous inputs.

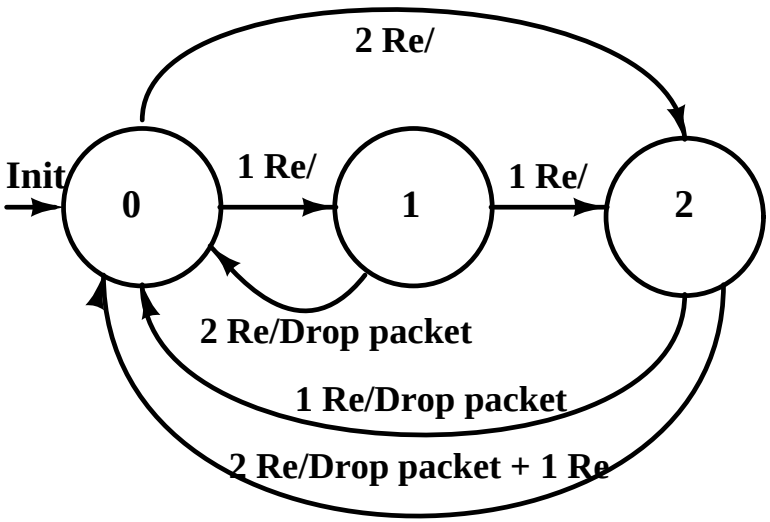


The **state** represents the "history" of input variables as relevant to production of the output sequence.

Systems which use this approach to generate sequential logic are called **Finite State Machines** or FSMs.

FSM example: Penny in the slot machine

Consider a Penny in the slot machine which accepts 1 Re and 2Re coins. It dispenses a packet worth 3 Rs. and returns coins if required.

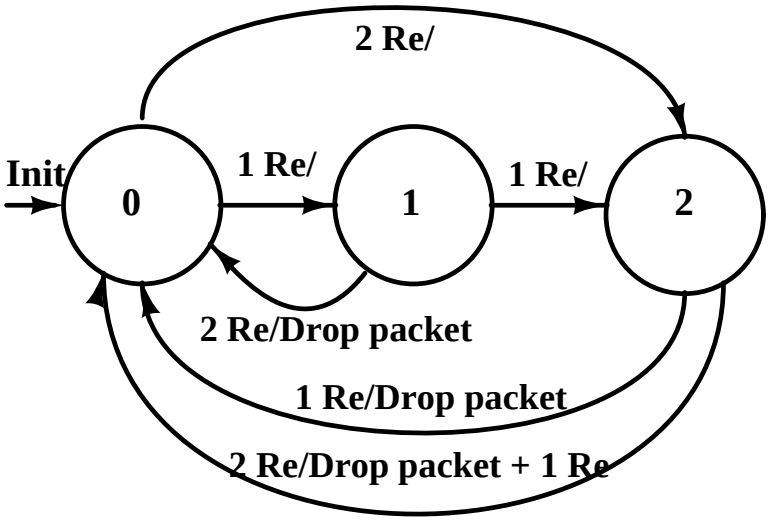


The state diagram on the left shows the evolution of 'states' in response to a sequence of inputs.

- At any time, one of two inputs may occur – insertion of a 1 Re coin or insertion of a 2 Re coin.
- In response, the machine may – i) do nothing, ii) Drop the packet or iii) Drop the packet and a 1 Re coin.

The machine can be designed with 3 states and combinational logic to determine the action and the next state in response to specific inputs.

FSM example: Penny in the slot machine



At power on, the machine wakes up in state 0.

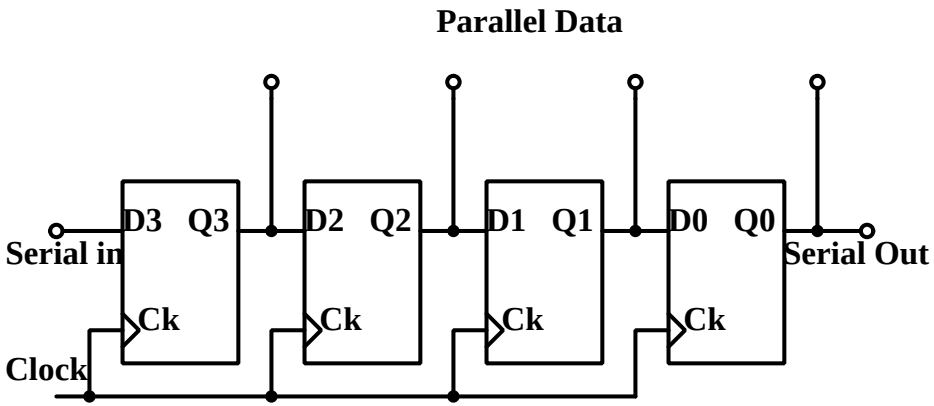
- At any time, The machine is in one of the three states shown.
- In any state, only two external inputs are possible: Arrival of a 1 Re coin or arrival of a 2 Re coin.

Curr st	Input	Output	Next St	Input	Output	Next St
0	1 Re	Nothing	1	2 Re	Nothing	2
1	1 Re	Nothing	2	2 Re	Drop packet	0
2	1 Re	Drop packet	0	2 Re	Drop packet + 1 Re	0

We can implement this with 2 flipflops to encode the state number and combinational logic to determine next state and output from current input and state.

Registers

- A group of D flipflops in parallel can be used to store a multi-bit value. This is called a **register**.
- We can also connect D flipflops in series as shown below. This arrangement is called a **Shift Register**.



At each active edge of the clock, $Q0 \rightarrow$ Serial Out, $Q1 \rightarrow Q0$, $Q2 \rightarrow Q1$, $Q3 \rightarrow Q2$, Serial In $\rightarrow Q3$.
Thus data is shifted to the right by one location.

- This can be used for serial to parallel conversion by feeding data in serially and after all data has been fed, collecting it in parallel.
- Shift register are also used for parallel to serial conversion. Data is loaded in parallel to all the D flipflops and shifted out serially.
- These are also used for data delay, as a serial memory etc.

Number Systems

- We have already seen how we can represent natural numbers in a base-2 system (binary representation).
  The bit pattern $b_{n-1} \ldots b_i \ldots b_1 b_0$ represents the number:
  $$N = \sum_{i=0}^{n-1} 2^i \cdot b_i.$$
- Given a decimal number N, how do we determine b_i?
  - Take N and divide by 2. The remainder is b_0.
  - Repeat this process by dividing the quotient by 2 and retaining the remainder as $b_1, b_2 \ldots$. Continue till the quotient becomes 1 or 0. This is then the most significant bit.
  - For somewhat larger numbers, it is more efficient to convert the number to base-16 or Hex format, by using the above procedure but dividing by 16 every time.
  - The resulting Hex number can be converted to binary easily by replacing each hex digit by its binary equivalent.

An n bit binary number can represent any natural number in the range $0 \leq N \leq 2^n - 1$.

Representation of Natural numbers

Let us illustrate the conversion procedure by a few examples.

Given $N = 55$.
Divide successively by 2.

$55 = 27 * 2 + 1$, so $b_0 = 1$
$27 = 13 * 2 + 1$, so $b_1 = 1$
$13 = 6 * 2 + 1$, so $b_2 = 1$
$6 = 3 * 2 + 0$, so $b_3 = 0$
$3 = 1 * 2 + 1$, so $b_4 = 1$
Quotient = 1, go $b_5 = 1$

Thus decimal $55 =$ binary 110111

Given $N = 1000$. Divide by 16.
$1000 = 62 * 16 + 8$,
so least significant hex digit is 8.
$62 = 3 * 16 + 14$,
so the next digit is 14 or E.
$3 < 16$, so the most significant digit is 3. Thus $1000 =$ 3E8 in Hex.
Expanding each hex digit to binary, this can be written as:

$$0011 \mid 1110 \mid 1000.$$

Binary addition

- Binary numbers can be added just like decimal numbers, with a carry being transferred to a more significant bit position whenever the sum exceeds 1.
- We can represent positive numbers between 0 and $2^n - 1$ using n bits.
- What happens if the sum exceeds the largest representable number?

Consider an example using 4 bit numbers. Using 4 bits we can represent numbers between 0 and 15

Now consider the addition of 11 to 7 in binary arithmetic using 4 bits. 1011 + 0111 = 10010. The overflowing 5th bit can be used to signal that overflow has occurred and the result is not valid.

However, this provides us with a clue to a method for representing negative numbers.

Representing Negative numbers

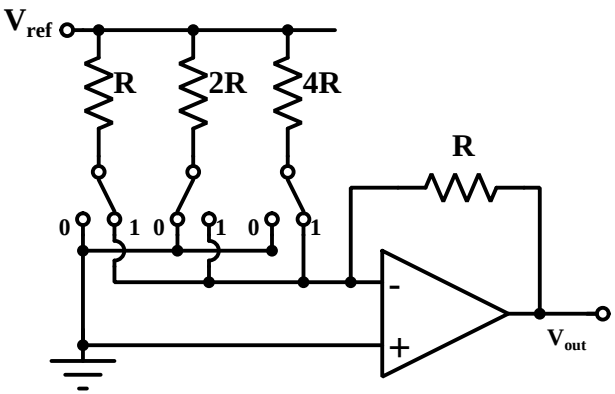
- In arithmetic, we define a negative number by the property: $x + (-x) = 0$.
- To represent negative numbers, we take the magnitude of the number and then choose a number which when added to it will give zeros in the defined bit width, (albeit with an overflow).
- For example, the decimal number 5 is 0101 using 4 bit representation. When we add decimal 11 to it we get 0101 + 1011 = 10000. (5+11=16). If we ignore the overflow (fifth bit), the result is zero.
- Thus, if x = 0101, (-x) = 1011. Now, x + (-x) = 0, if we ignore the overflow.
- Obviously, we cannot use 1011 to represent 11 as well as -5! So we adopt a convention where numbers with 0 in the most significant position are positive, while numbers with 1 in the most significant position represent negative numbers.

Representing Negative numbers

- We use the convention where numbers with 0 in the most significant position are positive, while numbers with 1 in the most significant position represent negative numbers.
- With this convention, signed 4 bit numbers can be represented in the range -8 to + 7, inclusive of Zero.
- How do we find the number which when added to x will give zero (with overflow)?
- Notice that when we add a number to its complement (changing all 0's to 1 and 1's to zero), the sum will have 1 in all positions. Now if add 1 to this, we'll get all zeros with an overflow.
- So to get the negative of a number, we take its complement (also called 1's complement) and add 1 to it. (This is called 2's complement).

Digital to Analog Conversion

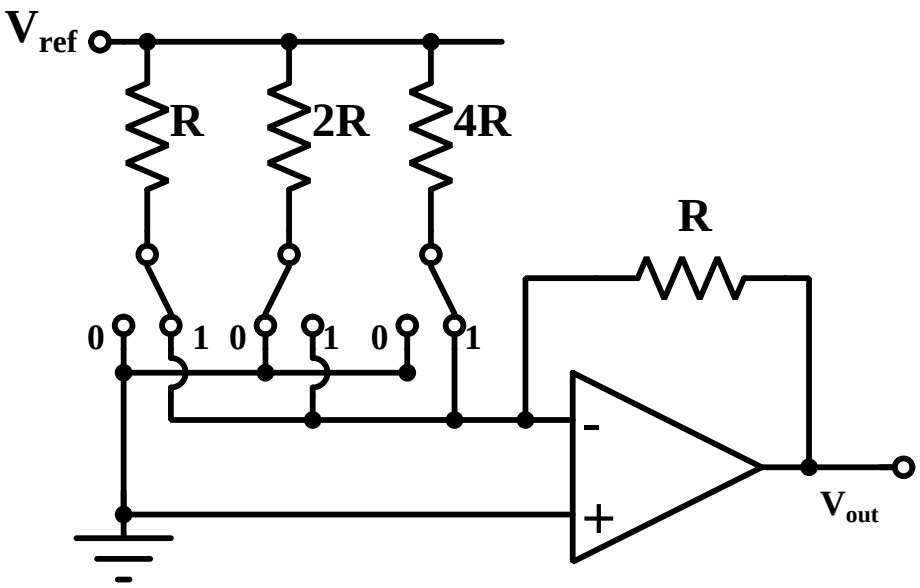
Consider the production of an analog quantity – say voltage, proportional to a given digital numbers. This is Digital to Analog conversion or DAC.



A digital number: $b_{n-1} \ldots b_i \ldots b_1 b_0$ represents $\sum_0^{n-1} 2^i b_i$.
If we use resistors with resistance values proportional to 2^i and use these to draw current from a reference voltage V_{ref} (with the other end of the resistors at ground potential), the currents will be in binary ratio.

We put a two way switch at the end other than V_{ref} which connects either to ground or to the virtual ground of an opamp.
Since these two choices are at the same potential, the current remains unchanged if we switch it between either of these.

Digital to Analog Conversion



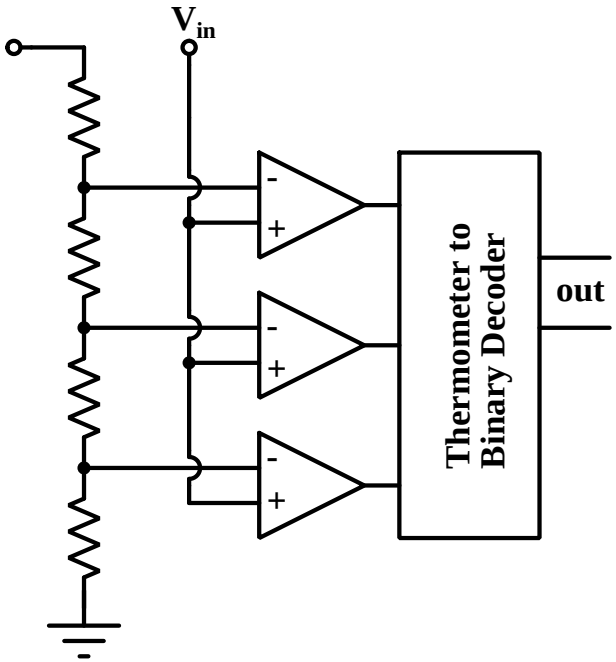
Each two way switch is controlled by a bit of the digital number which we want to convert. Since the two way switches direct the current either to ground or to the virtual ground of an opamp, the current through the resistors remains unchanged if we switch it between either of these.

The current going into the inverting terminal is then proportional to the digital number. The output voltage is $-R$ times this current and thus proportional to the digital number.

This method of D to A conversion is intuitive, but getting accurate values of resistors over a wide range is a difficult task.

Analog to Digital Conversion

A flash ADC is a fast converter which uses a separate comparator for each digital value possible. A resistor divider creates equally spaced N-1 reference levels.



- A bank of comparators compares the input to each of the reference levels.
- All comparators with reference level below the input value go to '1' while all those above it go to '0'.
  This is known as **Thermometer coding**.
- A decoder is then used to transform the thermometer code to binary.

Flash converters are used for A to D conversion with a small number of bits, since its complexity grows exponentially with each additional bit.

Analog to Digital Conversion

- Many other types of A to D converters are available, providing a compromise between speed, power and complexity.
- A successive approximation ADC uses a DAC whose Digital input is provided by a register. The ADC sequentially adjusts the bits of this register and compares the output of the DAC with the input voltage, till they match. This type of ADC is widely used in medium speed applications.
- A dual slope ADC provides low complexity at low conversion rates. It converts its input voltage to a time interval, which is measured using a counter. This kind of ADC is used by most DMMs.

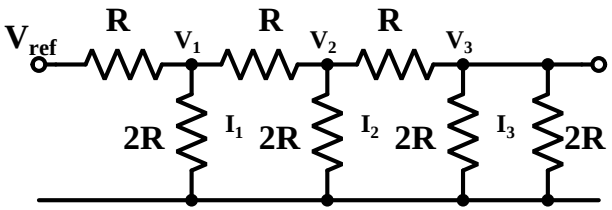
From the user's view point, a "start conversion" signal is sent to the ADC (of whatever type) and when the conversion is complete, it sends an "end of conversion" signal. The output of the ADC can be read as a digital number once the conversion is complete.

And That's all

for now!

(for Digital Design).

DAC with R-2R ladder network



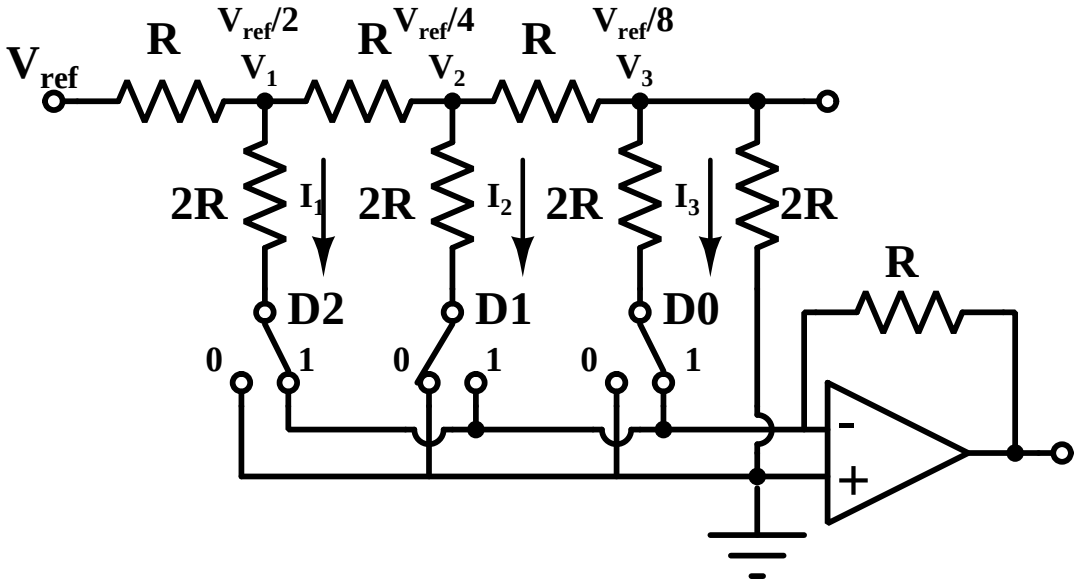
Consider a network with resistors of value R in series and those of value 2R in parallel in a ladder network as shown on the left.

Notice the extra 2R resistor at the end of the chain.

- At the end, we have 2 resistors to ground, each of value 2R. This is equivalent to a single resistor of value R.
- This equivalent resistance of R and the series resistance R from V_2 form a potential divider such that $V_3 = V_2/2$.
- The series resistor from V_2 and the equivalent resistor R to ground provide a total resistance of 2R to ground from V_2.
- Extending this argument, $V_2 = V_1/2$ and $V_1 = V_{ref}/2$
- Correspondingly, the current to ground through the 2R resistors is in binary ratio.
- by inserting a two way switch at the ground end and switching between ground and virtual ground, we can generate a voltage proportional to the digital value.

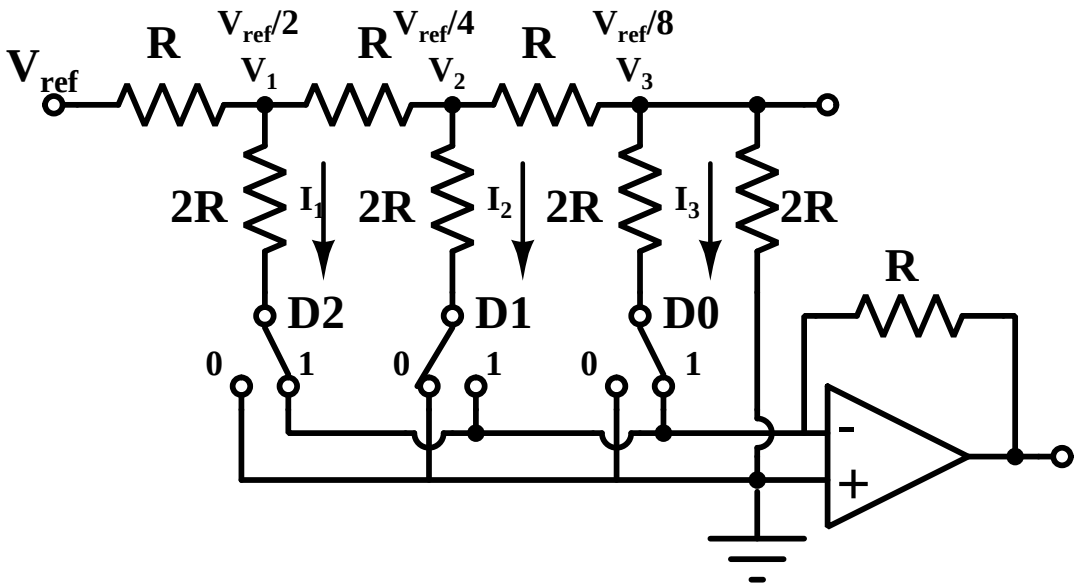
DAC with R-2R ladder network

The Circuit below shows a D to A converter using the R-2R ladder network.



- Bits of the digital value choose whether the current through the 2R resistors will go to ground or to virtual ground.
- Total current going into the virtual ground is converted to a voltage using a feedback resistor.

DAC with R-2R ladder network



- Implementing the DAC is easier this way, because it requires only two resistance values. In fact, two identical resistors are put in series to get the 2R value.
- Accurate matching of resistors is much easier here compared to the binary weighted resistor case.

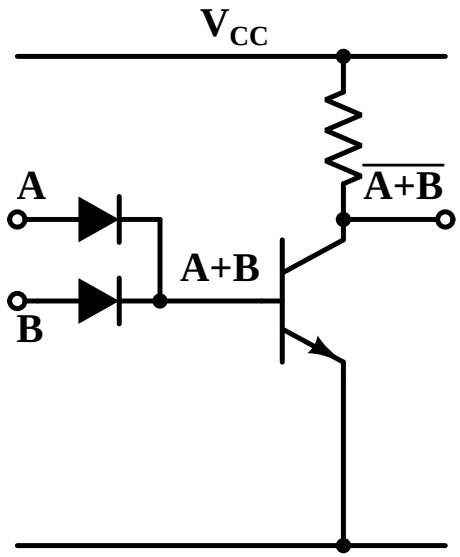
ADC with successive approximation

- This is a widely used architecture for ADCs.
- It uses a DAC and a comparator.
- At the first step, the most significant bit for the DAC is set.
- The comparator indicates if the unknown voltage is above or below the DAC output. If the unknown value is below the DAC value, the bit which was last set is cleared.
- Now the next significant bit for the DAC input is set. The unknown voltage is compared to the DAC output again.
- This process is continued till the least significant bit has been determined.

Logic families: DTL

Earliest implementation of Digital logic used relays connected in series/parallel to implement logic.

With the advent of semiconductor devices, diodes and transistors were used to implement digital logic. The circuit on the right is a DTL NOR gate.
A HIGH voltage (close to V_{CC}) represents a logic '1', while a LOW voltage (close to ground) represents a logic '0'.



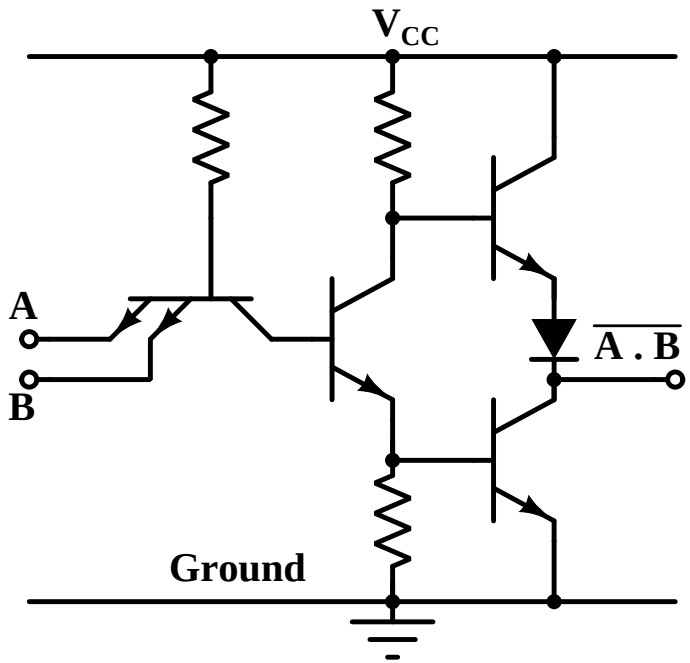
When either A or B (or both) are HIGH, base current flows, saturating the bipolar transistor. This pulls the output LOW.
The transistor is OFF when both A and B are LOW as there is no base current. The output is then connected to V_{CC} through the collector resistor and is HIGH.

Logic families: TTL

Diode-Transistor Logic was eventually replaced by Transistor-Transistor Logic or TTL.

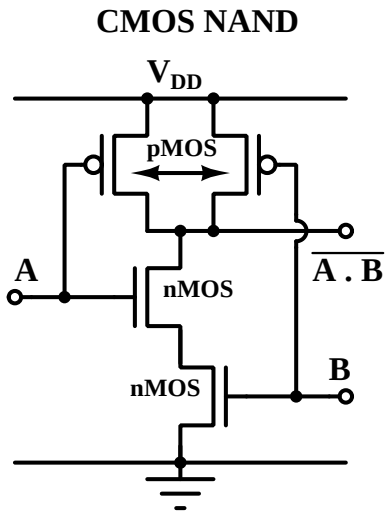
This was the dominant digital technology for implementation of digital logic for a long time.

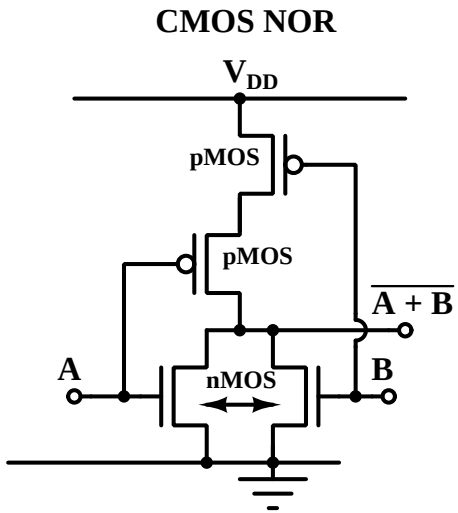
- TTL replaces the diodes at the input in DTL by a bipolar transistor with multiple emitters.
- The circuit on the right shows a NAND gate implemented in TTL technology.
- The output stage is called a “totem pole” circuit.



Logic families: CMOS

- Modern digital circuits use MOS transistors.
- A ‘1’ at the gate input turns the nMOS ON and the pMOS OFF
- A ‘0’ at the gate input turns the nMOS OFF and the pMOS ON.



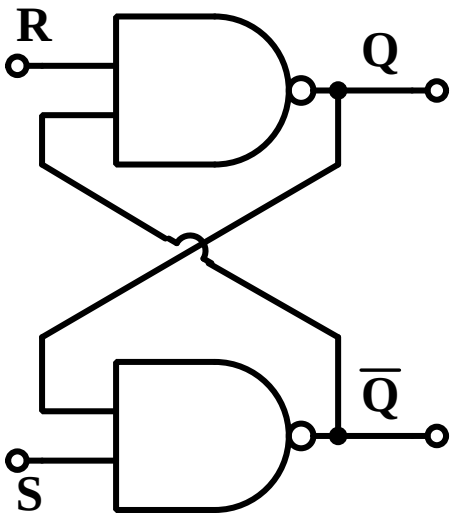


NAND output is pulled to ground when **both** inputs are ‘1’. It is pulled up to V_{DD} when **either or both** inputs are ‘0’.
NOR output is pulled to ground when **either or both** inputs are ‘1’. It is pulled up to V_{DD} when **both** inputs are ‘0’.

By using complementary MOS transistor types (n channel and p channel), high speed can be obtained at reasonably low power consumption. This is known as CMOS logic.

RS Latch with cross connected NAND gates

Remember the duality property of Boolean logic?



We can also construct an RS latch using cross connected NAND gates. Its operation is similar to the cross connected NOR.

Here the circuit is idle when R and S inputs are at '1', while the 'set' and 'reset' action is triggered by S or R going to '0'.

Like the cross connected NOR circuit, Set and Reset are not supposed to go to their active level ('0' in this case) simultaneously.

If both R and S inputs go to their active level ('0') simultaneously, Q as well as $\overline{Q}$ will go to '1'.

The output will settle to one of the two stable states depending on which of the two inputs is removed from the active state last.