

Digital Logic: Design and Implementation

Dinesh Sharma
Joseph John
P.C. Pandey
Kushal Tuckley

EE Department
IIT Bombay, Mumbai

Book References

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

April 8, 2023

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 about Karnaugh maps – a graphical method for logic minimization.

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**.

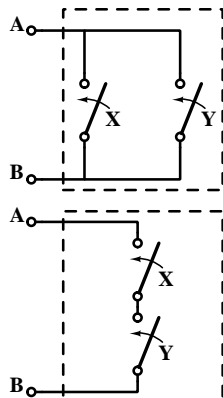
Implementing Digital Logic

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.

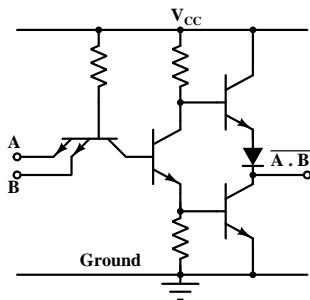


Logic Gates

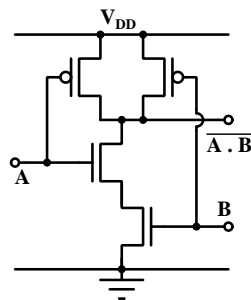
- In the early days, relays were used as controllable switches to implement logic functions.
- More recently, logic circuits have been implemented using diodes, bipolar transistors and MOS transistors.
- These days, digital circuits are implemented using a combination of n and p channel MOS transistors. 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.

Logic Gates

Implementation of two input NAND gates in TTL and CMOS technologies is shown in the circuit diagrams below:



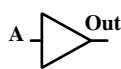
A Two input NAND gate
Implemented in TTL technology



A Two input NAND gate
Implemented in CMOS technology

Logic Gates and symbols

Irrespective of the technology used for implementing the logic, standard symbols are used to represent logic functions.



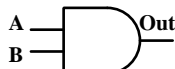
Buffer

$$\text{Out} = A$$



OR

$$\text{Out} = A + B$$



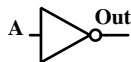
AND

$$\text{Out} = A \cdot B$$



XOR

$$\text{Out} = A \oplus B$$



Inverter

$$\text{Out} = \overline{A}$$



NOR

$$\text{Out} = \overline{A + B}$$



NAND

$$\text{Out} = \overline{A \cdot B}$$



XNOR

$$\text{Out} = \overline{A \oplus B}$$

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.

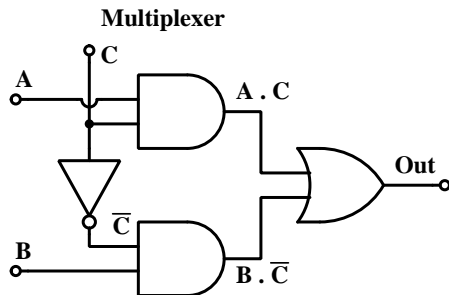
Logic Function	No of gates on chip	TTL implementation	CMOS B Series
Inverter	6	7404	4069
2 input AND	4	7408	4012
2 input NAND	4	7400	4011
4 input AND	2	7421	4082
4 input NAND	2	7420	4012
2 input OR	4	7432	4071
2 input NOR	4	7402	4001
2 input XOR	4	7486	4070

A Multiplexer

- We often need to choose one of several inputs.
- The specific input to be selected can be indicated by a digitally encoded number C .
- Digital circuits which implement this are called multiplexers.
- For example, we may 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 the two input bits A and B and put the selected bit on the output line.

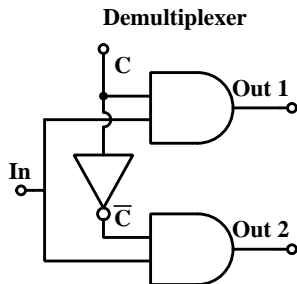


A De-multiplexer

- **De-multiplexers** perform the opposite function to multiplexers.
- The input data comes on a single line and depending on a line number code, it is put on a selected output line.

De-multiplexing can be easily done for two lines.

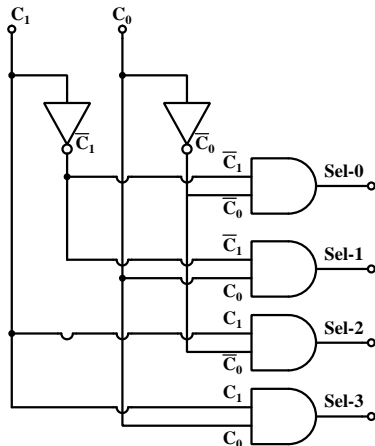
- We put the AND of the incoming data with C on one of the output lines.
- We put the AND of the incoming data with \overline{C} on the other line.



Now when $C = 1$, the incoming data stream will be replicated on the upper line and when $C = 0$, it will be replicated on the lower line.

Decoding

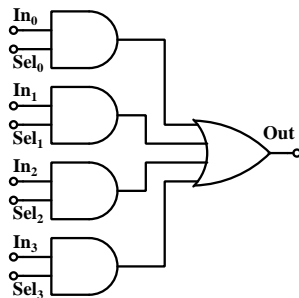
What can we do if we want to multiplex from or de-multiplex to more than two lines?



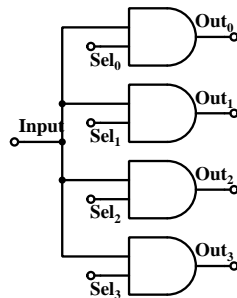
- We need to use multiple bits C_i in place of a single bit C to specify which input bit to choose in a multiplexer (and which line to choose for a de-multiplexer).
- From C_i , we generate multiple select lines such that exactly one of the select lines is '1' and all others are '0' for various combinations of C_i .
- Such circuits are called **decoders**.

Multi-bit Multiplexers and De-multiplexers

Multiple select lines generated by decoders can be used to implement multi-bit multiplexers and de-multiplexers.



Multiplexer

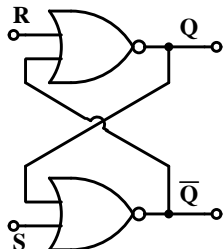


De-multiplexer

Apart from their use in multiplexers and de-multiplexers, decoders 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)

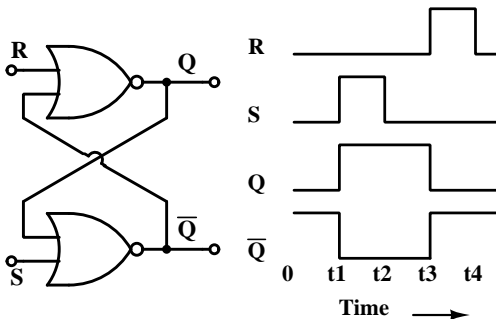
It is easy to see that $R = 0, S \rightarrow '1'$ will force the circuit into its 'set' state ($Q = 1, \overline{Q} = '0'$).

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

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 ($Q = '0'$, $\bar{Q} = '1'$) state.

At t1, S goes to '1' and it forces \bar{Q} to '0' unconditionally.

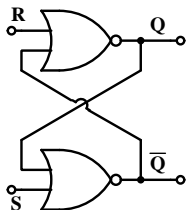
Since $R = '0'$ and $\bar{Q} = '0'$, Q goes to '1'.

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

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

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

RS Latch: set and reset action



Since $S=1$ sets Q to '1', it is called the 'Set' input.
 $R=1$ 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 (S) returned to '0'.

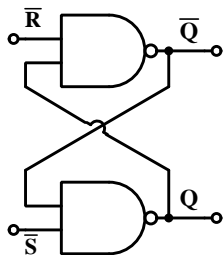
Similarly, the latch retains its reset ($Q = 0$, $\overline{Q} = 1$) state even after R returns to '0'. It "remembers" that R had gone to '1' before it returned to '0'.

If Set and Reset are simultaneously '1', both Q and \overline{Q} are forced to '0'. The latch will come to set or reset state when one or both of R and S return to '0'.

If both were '1' and both return to '0', the state of the flipflop will depend on which input returned to '0' last. This is known as a **race condition**.

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 \overline{R} and \overline{S} inputs are at '1', while the 'set' and 'reset' action is triggered by \overline{S} or \overline{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 \overline{R} and \overline{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.

Other storage circuits: D flipflop

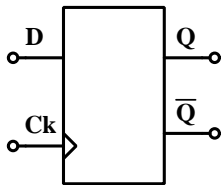
There are other useful circuits which show a 'memory'.

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**.

Memory elements triggered by a clock edge are known as **Flipflops**.

D flipflop: This is a 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 on the right.

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 in 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.

However, this is a clocked flipflop and not a latch.

Also, the JK flipflop has a well determined behaviour when both J and K are '1'.

At the active edge of the clock,

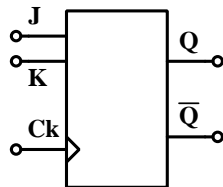
If J = '0' and K = '0', the flipflop outputs remain unchanged.

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} .

Notice that J = '1' and K = '1' is a valid input to the JK flipflop.

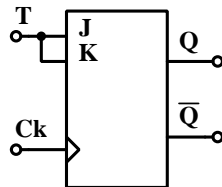


Other circuits with memory: Toggle flipflop

Toggle flipflop:

As the name implies, a Toggle flipflop changes its state (from '1' to '0' and from '0' to '1') at the arrival of the active clock edge if the toggle input (T) is '1'

It can be implemented using a JK flipflop with the J and K inputs tied together and used as the T input.



When $T = '0'$, outputs Q and \overline{Q} of the flipflop remain unchanged.
When $T = '1'$, it toggles (complements the values of Q and \overline{Q}) at the active clock edge.

An application of toggle flipflops is in **counter** circuits.

These are multi-bit storage circuits whose outputs (interpreted as numbers) increment or decrement on arrival of the clock.

Sequential Circuits

- In a sequential circuit, the **sequence of output values** is a function of **the sequence of input values**.
- Obviously for the circuit to be practically implementable, the sequences between which we want to establish a functional relationship have to be of finite length.
- As in the case of design of combinational circuits, it will be helpful if we have a standard way of implementing such circuits.
- Since the output at any point of time depends on current inputs **as well as** previous inputs, we need to store the previous inputs in some form.
- However, a brute force implementation of storing all previous inputs will be inefficient – even impractical.
- Sequential circuits may be implemented efficiently using **Finite State Machines**.

- In a sequential circuit, a new set of input values is presented during each new clock period.
- However, it is not necessary to store all the past values of input variables.
- Instead, a (possibly multi-bit) **state** is stored, which is a digital value with sufficient information about the past inputs, so that the current output can be generated as a **combinational** function of the **current state** and the **current inputs**.
- In each clock period, as fresh inputs arrive, a new value of **state** is generated which is also a combinational function of the current value of state and current inputs.
- The state is thus a recursive function of all previous inputs.

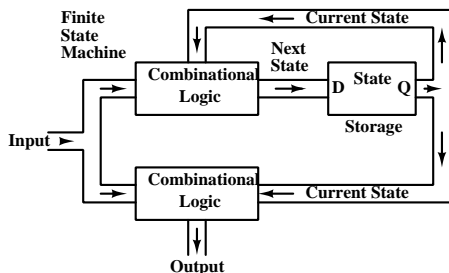
Finite State Machines

In each clock period, as fresh inputs arrive, a new value of **state** needs to be generated which is a combinational function of the current value of **state** and the current inputs.

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

We need two combinational circuits now.

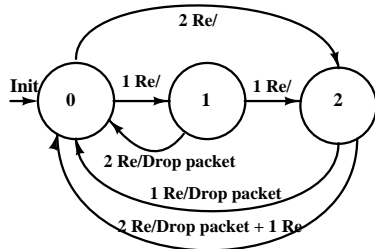
- 1 One to generate the outputs from the current state and current inputs and
- 2 another to generate the new value of the state from its stored value and fresh inputs.



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.

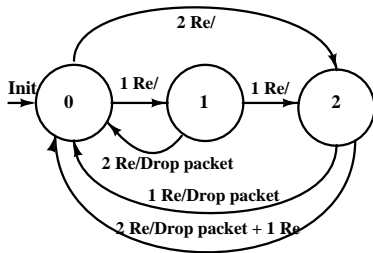


The state diagram on the left shows the evolution of 'states' in response to a sequence of inputs.

- 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 –
 - do nothing,
 - Drop the packet or
 - 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.

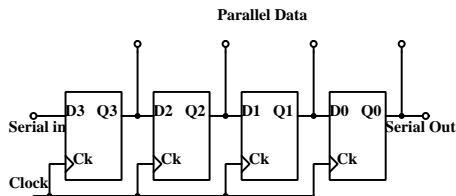
- At any time, The machine is in one of the three states shown.
- In any state, only two external inputs are possible: Arrival of a 1 Re coin or arrival of a 2 Re coin.

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	packet	0
2	1 Re	packet	0	2 Re	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,
 $Q0 \rightarrow \text{Serial Out}$, $Q1 \rightarrow Q0$,
 $Q2 \rightarrow Q1$, $Q3 \rightarrow Q2$,
 $\text{Serial In} \rightarrow Q3$.

Thus all bits of data are shifted to the right by one location.

- This can be used for serial to parallel conversion by feeding data in serially and after the data has been fed, collecting it in parallel.
- Shift registers 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 $b_{n-1} \dots b_i \dots b_1 b_0$ 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 $b_1, b_2 \dots$. 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.

$$55 = 27 * 2 + 1, \text{ so } b_0 = 1$$

$$27 = 13 * 2 + 1, \text{ so } b_1 = 1$$

$$13 = 6 * 2 + 1, \text{ so } b_2 = 1$$

$$6 = 3 * 2 + 0, \text{ so } b_3 = 0$$

$$3 = 1 * 2 + 1, \text{ so } b_4 = 1$$

$$\text{Quotient} = 1, \text{ go } b_5 = 1$$

Thus decimal $55 = \text{binary } 110111$

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.

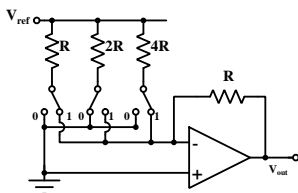
Representing Negative numbers

- 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 number.

This is Digital to Analog conversion or DAC.



A digital number: $b_{n-1} \dots b_i \dots b_1 b_0$
represents $\sum_{i=0}^{n-1} 2^i b_i$.

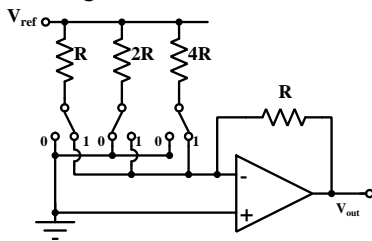
If we use resistors with resistance values proportional to 2^i and use these to draw current from a reference voltage V_{ref} (with the other end of the resistors at ground potential), the currents will be in binary ratio.

The two way switch at the bottom of each resistor connects either to ground or to the virtual ground of an opamp.

Since these two choices are at the same potential due to virtual short, the current through the resistors remains unchanged when we switch it between either of these positions.

Digital to Analog Conversion

Each two way switch in the circuit shown below is controlled by a bit of the digital number which we want to convert.

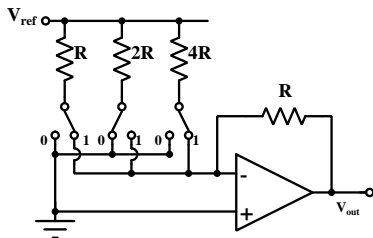


- If the bit is '0', current goes to ground.
- If the bit is '1', current goes to the I-V converter formed by the opamp.

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 when we switch it between either of these.

Let $I_0 = V_{ref}/(2^{n-1}R)$ where n is the number of bits in the digital word. In the 3 bit converter shown above, $I_0 = V_{ref}/4R$.

Digital to Analog Conversion



- If bit i is '0', current into the I-V converter is 0. (Goes to ground).
- if bit i is '1', current into the I-V converter is $2^i I_0$.
- So the total current into the I-V converter is $I_0 \sum 2^i b_i$.

Since the digital word represents $N = \sum 2^i b_i$, the total current into the I-V converter is $N I_0$.

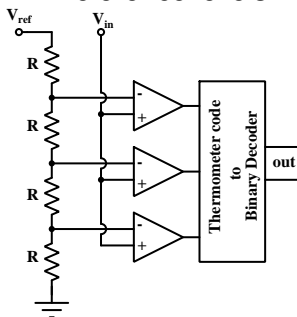
Then the output voltage is:

$$-R I_{total} = -R N I_0 = -R N \frac{V_{ref}}{2^{n-1} R} = -N \frac{V_{ref}}{2^{n-1}}$$

Thus the output voltage is proportional to the digital word N .

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).