Lab 2 - 135 mins / 3 lectures.

Main objects

A. Combination circuit.
B. Addition and Subtraction circuits.
C. Multiplexer and DeMultiplexer.
D. Encoder and Decoder.
E. Application with MSI Combination circuits.

Resources

Object	Software / File	Book	Slide	Student's task
Full adder	Fulladder.circ	[1] p.368		Download and install to your PCs
Multiplexer	Lb2_2.circ	[1] p.105	Ch.7 p.34	Experiment
DeMUX		[1] p.114	Ch.7 p.36	Experiment
App on communication	N/A	[1] p.115	Ch.7 p.38	Experiment and advance work.
Encoder	Lab2_4.circ	[1] p.123		
Decoder	Lab2_5.circ	[1] p.116		
Applications on MUX/DeMUX		[1] p.		

Preferences

[1] Brian Holdsworth, Clive Woods, [2002], Digital Logic Design, 4th edition, Newnes, Oxford.

[2] https://www.electronics-tutorials.ws/combination/ access on Jan 14th.

Table of Contents

Part 1. Combination circuit

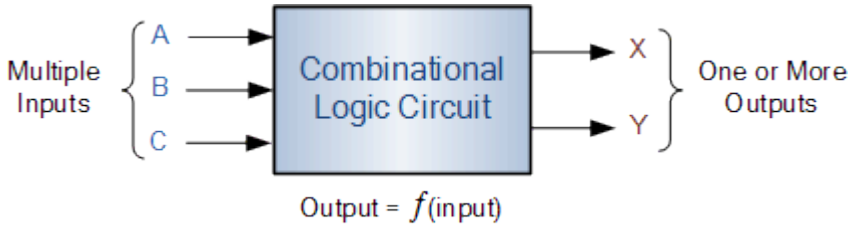


Figure 1. Defination of Combination circuit.

Combinational Logic Circuits are memoryless digital logic circuits whose output at any instant in time depends only on the combination of its inputs

How to specific a combination circuit.

The three main ways of specifying the function of a combinational logic circuit are:

1. **Boolean Algebra** – This forms the algebraic expression showing the operation of the logic circuit for each input variable either True or False that results in a logic "1" output.
2. **Truth Table** – A truth table defines the function of a logic gate by providing a concise list that shows all the output states in tabular form for each possible combination of input variable that the gate could encounter.
3. **Logic Diagram** – This is a graphical representation of a logic circuit that shows the wiring and connections of each individual logic gate, represented by a specific graphical symbol, that implements the logic circuit.

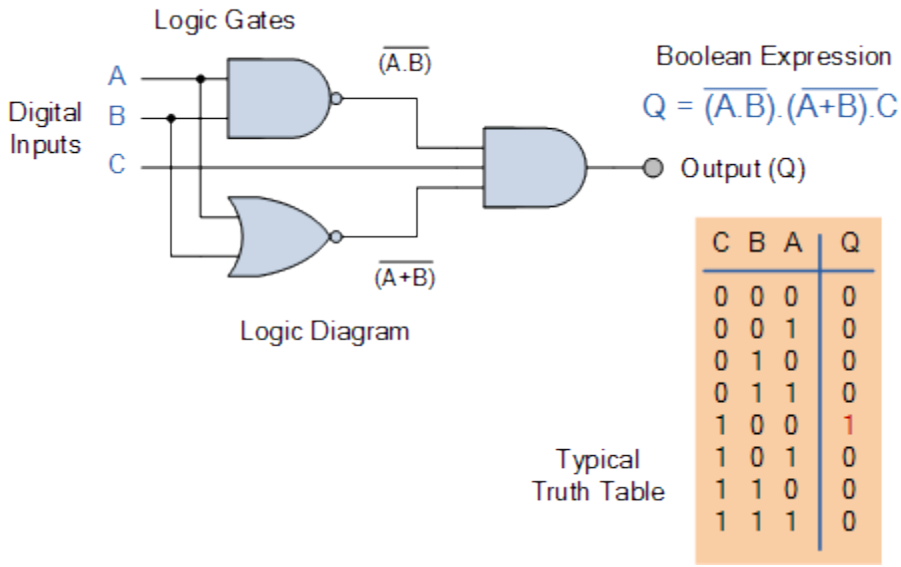


Figure 2. Three ways of specifying the function of a combinational circuit.

As combinational logic circuits are made up from individual logic gates only, they can also be considered as “decision making circuits” and combinational logic is about combining logic gates together to process two or more signals in order to produce at least one output signal according to the logical function of each logic gate. Common combinational circuits made up from individual logic gates that carry out a desired application includeMultiplexers, De-multiplexers, Encoders, Decoders, Full and Half Adders etc.

Classification of Combinational Logic

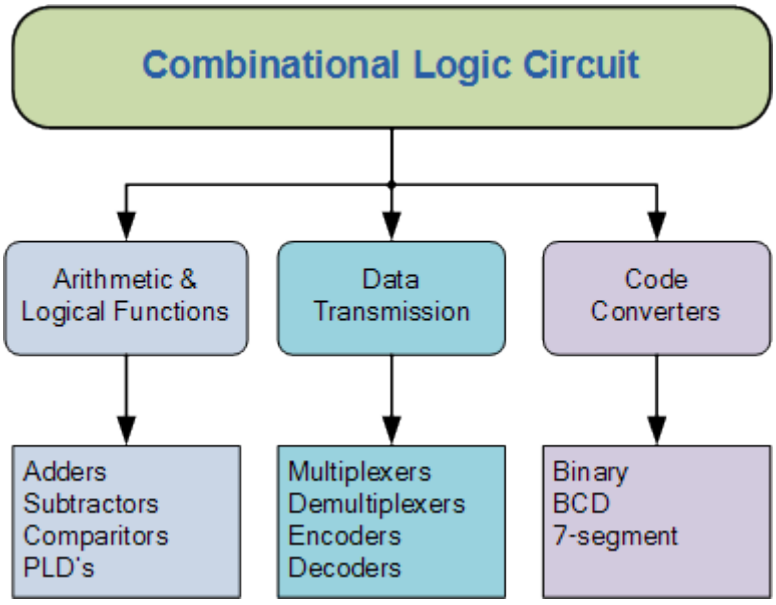


Figure 3. Classification of Combinational Logic.

Have you done the “Half Adder” and “BCD to 7-segments LED IC driver” in previous LAB? They are totally combination circuits!

Part 2. Binary adder

Another common and very useful combinational logic circuit which can be constructed using just a few basic logic gates allowing it to add together two or more binary numbers is the **Binary Adder**.

A basic Binary Adder circuit can be made from standard AND and Ex-OR gates allowing us to "add" together two single bit binary numbers, A and B.

The addition of these two digits produces an output called the SUM of the addition and a second output called the CARRY or Carry-out, (C_{OUT}) bit according to the rules for binary addition. One of the main uses for the *Binary Adder* is in arithmetic and counting circuits. Consider the simple addition of the two denary (base 10) numbers below.

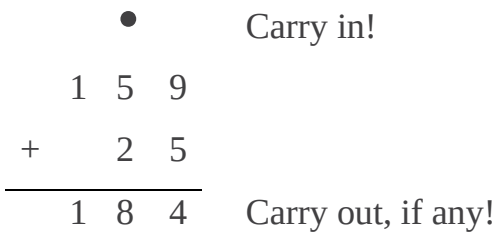


Figure 4. Example of Addition with Decimal numbers.

From our maths lessons at school, we learnt that each number column is added together starting from the right hand side and that each digit has a weighted value depending upon its position within the columns.

When each column is added together a carry is generated if the result is greater or equal to 10, the base number. This carry is then added to the result of the addition of the next column to the left and so on, simple school math's addition, add the numbers and carry.

The adding of binary numbers is exactly the same idea as that for adding together decimal numbers but this time a carry is only generated when the result in any column is greater or equal to "2", the base number of binary. In other words, 1 + 1 creates a carry.

Binary Addition

Binary Addition follows these same basic rules as for the denary addition above except in binary there are only two digits with the largest digit being "1". So when adding binary numbers, a carry out is generated when the "SUM" equals or is greater than two (1+1) and this becomes a "CARRY" bit for any subsequent addition being passed over to the next column for addition and so on. Consider the single bit addition below.

```
  • • •          Carry in!
    1 0 1
+     1 1
-------------
  1 0 0 0        Carry out, if any!
```

Figure 5. Example of Addition with Binary numbers.

Ex 1: Explain when "overflow" occour? How do we indentify an overflow addition?

Binary Adder Block Diagram

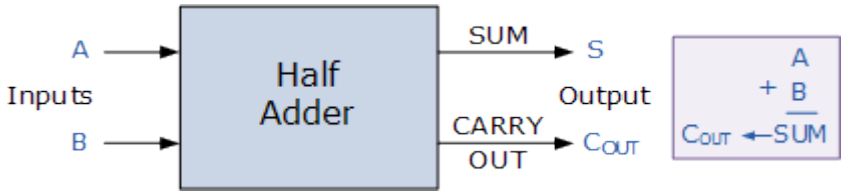


Figure 6. Block Diagram of Half adder.

For the simple 1-bit addition problem above, the resulting carry bit could be ignored but you may have noticed something else with regards to the addition of these two bits, the sum of their binary addition resembles that of an **Exclusive-OR** Gate. If we label the two bits as A and B then the resulting truth table is the sum of the two bits but without the final carry.

A Half Adder Circuit

A half adder is a logical circuit that performs an addition operation on two binary digits. The half adder produces a sum and a carry value which are both binary digits.

We have done it already in LAB1_1.circ. Review or build it again if you didn't finish.

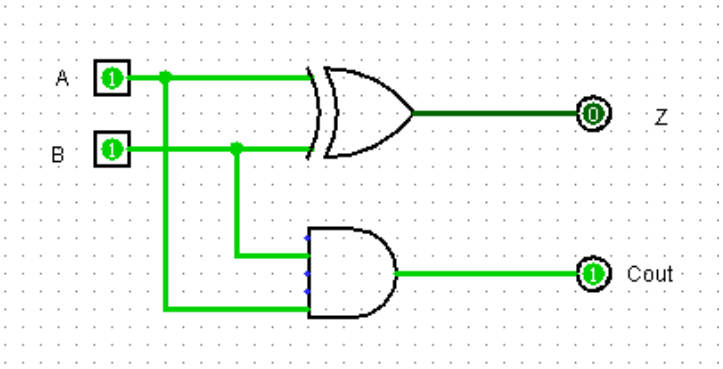


Figure 7. Circuit of a Half Adder.

A Full Adder Circuit

The main difference between the **Full Adder** and the previous **Half Adder** is that a full adder has three inputs. The same two single bit data inputs A and B as before plus an additional *Carry-in* (C-in) input to receive the carry from a previous stage as shown below.

Full Adder Block Diagram

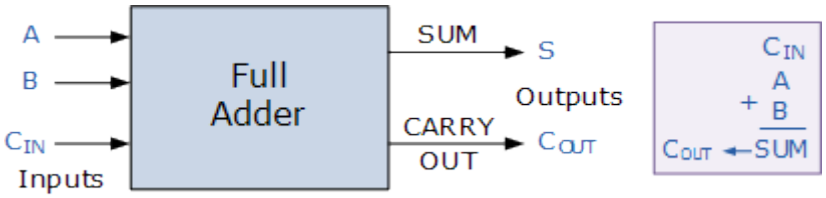


Figure 8. Block Diagram of Full Adder.

Then the **full adder** is a logical circuit that performs an addition operation on three binary digits and just like the half adder, it also generates a carry out to the next addition column. Then a *Carry-in* is a possible carry from a less significant digit, while a *Carry-out* represents a carry to a more significant digit.

In many ways, the full adder can be thought of as two half adders connected together, with the first half adder passing its carry to the second half adder as shown.

Full Adder Logic Diagram

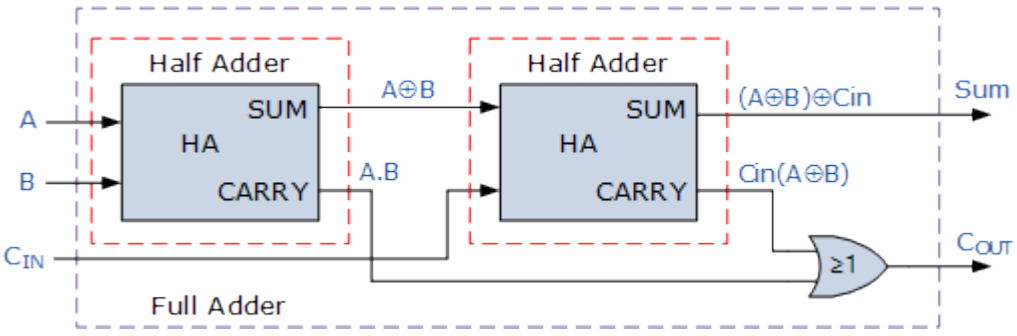


Figure 9. Diagram of a Full Adder built from 2 Half Adder.

As the full adder circuit above is basically two half adders connected together, the truth table for the full adder includes an additional column to take into account the *Carry-in*, C_{IN}input as well as the summed output, S and the Carry-out, C_{OUT} bit.

Full Adder Truth Table with Carry

Cin	B	A	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figure 10. Truth Table of a Full Adder.

Then the Boolean expression for a full adder is as follows.

For the **SUM** (S) bit: $SUM = (A \oplus B) \oplus Cin$

For the **CARRY-OUT** (Cout) bit: $CARRY\_OUT = A.B + Cin(A \oplus B)$

Implementation

After all, we have two methods to build a Full Adder

1. Logic design:
   - Input the expression or the truth table of Full Adder and procedure the final circuit.
   - This method cannot be used with huge inputs. Why not?
2. Block design:
   - Make the Half Adder become a subcircuit (IC A).
   - Use two IC A and some gates to make a Full Adder.

Ex 2: Full Adder is for 1-bit plus 1-bit number. Build a 4-bits adder and make it as a subcircuit.

?

- *How many inputs, outputs?*
- *Have you forgotten the Carry in and Carry out?*
- *Could it be done with method "Logic design"?*
- *How large the Truth Table will be?*
- *Method "Block design" is your only way!*

Part 3. The Multiplexer

Multiplexing is the generic term used to describe the operation of sending one or more analogue or digital signals over a common transmission line at different times or speeds and as such, the device we use to do just that is called a **Multiplexer**.

The multiplexer, shortened to "MUX" or "MPX", is a combinational logic circuit designed to switch one of several input lines through to a single common output line by the application of a control signal. Multiplexers operate like very fast acting multiple position rotary switches connecting or controlling multiple input lines called "channels" one at a time to the output.

Multiplexers, or MUX's, can be either digital circuits made from high speed logic gates used to switch digital or binary data or they can be analogue types using transistors, MOSFET's or relays to switch one of the voltage or current inputs through to a single output.

The most basic type of multiplexer device is that of a one-way rotary switch as shown.

Basic Multiplexing Switch

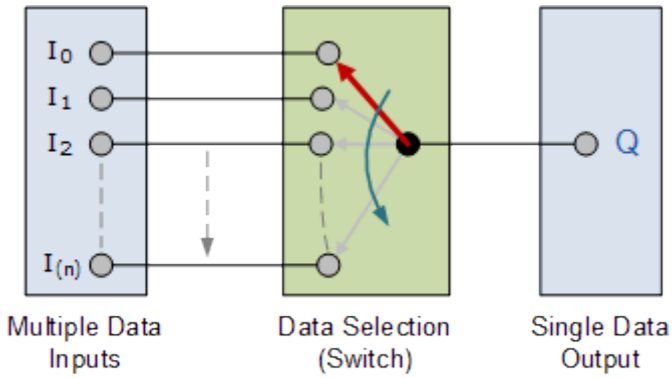


Figure 11. Basci Multiplexing Switch.

The rotary switch, also called a wafer switch as each layer of the switch is known as a wafer, is a mechanical device whose input is selected by rotating a shaft. In other words, the rotary switch is a manual switch that you can use to select individual data or signal lines simply by turning its inputs "ON" or "OFF". So how can we select each data input automatically using a digital device.

In digital electronics, multiplexers are also known as data selectors because they can "select" each input line, are constructed from individual Analogue Switches encased in a single IC package as opposed to the "mechanical" type selectors such as normal conventional switches and relays.

They are used as one method of reducing the number of logic gates required in a circuit design or when a single data line or data bus is required to carry two or more different digital signals. For example, a single 8-channel multiplexer.

Generally, the selection of each input line in a multiplexer is controlled by an additional set of inputs called control lines and according to the binary condition of these control inputs, either “HIGH” or “LOW” the appropriate data input is connected directly to the output. Normally, a multiplexer has an even number of 2^n data input lines and a number of “control” inputs that correspond with the number of data inputs.

Note that multiplexers are different in operation to *Encoders*. Encoders are able to switch an n-bit input pattern to multiple output lines that represent the binary coded (BCD) output equivalent of the active input.

We can build a simple 2-line to 1-line (2-to-1) multiplexer from basic logic NAND gates as shown.

2-input Multiplexer Design

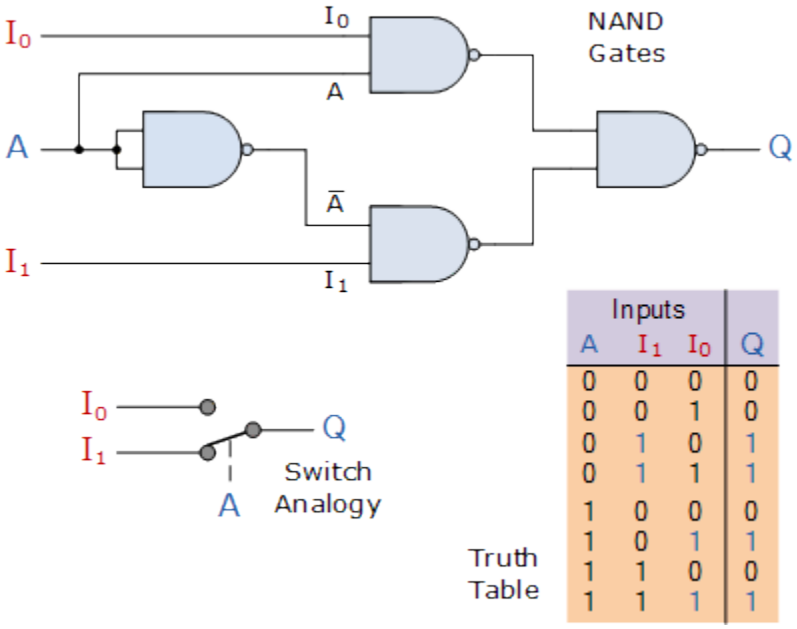


Figure 12. Design a 2-to-1 MUX.

The input A of this simple 2-1 line multiplexer circuit constructed from standard NAND gates acts to control which input (I_0 or I_1) gets passed to the output at Q.

From the truth table above, we can see that when the data select input, A is LOW at logic 0, input I_1 passes its data through the NAND gate multiplexer circuit to the output, while input I_0 is blocked. When the data select A is HIGH at logic 1, the reverse happens and now input I_0 passes data to the output Q while input I_1 is blocked.

So by the application of either a logic “0” or a logic “1” at A we can select the appropriate input, I_0 or I_1 with the circuit acting a bit like a single pole double throw (SPDT) switch.

As we only have one control line, (A) then we can only switch 2^1 inputs and in this simple example, the 2-input multiplexer connects one of two 1-bit sources to a

common output, producing a 2-to-1-line multiplexer. We can confirm this in the following Boolean expression.

$$Q = \bar{A}.\overline{I_0}.I_1 + \bar{A}.I_0.I_1 + A.I_0.\overline{I_1} + A.I_0.I_1$$

and for our 2-input multiplexer circuit above, this can be simplified too:

$$Q = \bar{A}.I_1 + A.I_0$$

We can increase the number of data inputs to be selected further simply by following the same procedure and larger multiplexer circuits can be implemented using smaller 2-to-1 multiplexers as their basic building blocks. So for a 4-input multiplexer we would therefore require two data select lines as 4-inputs represents 2^2 data control lines give a circuit with four inputs, I_0, I_1, I_2, I_3 and two data select lines A and B as shown.

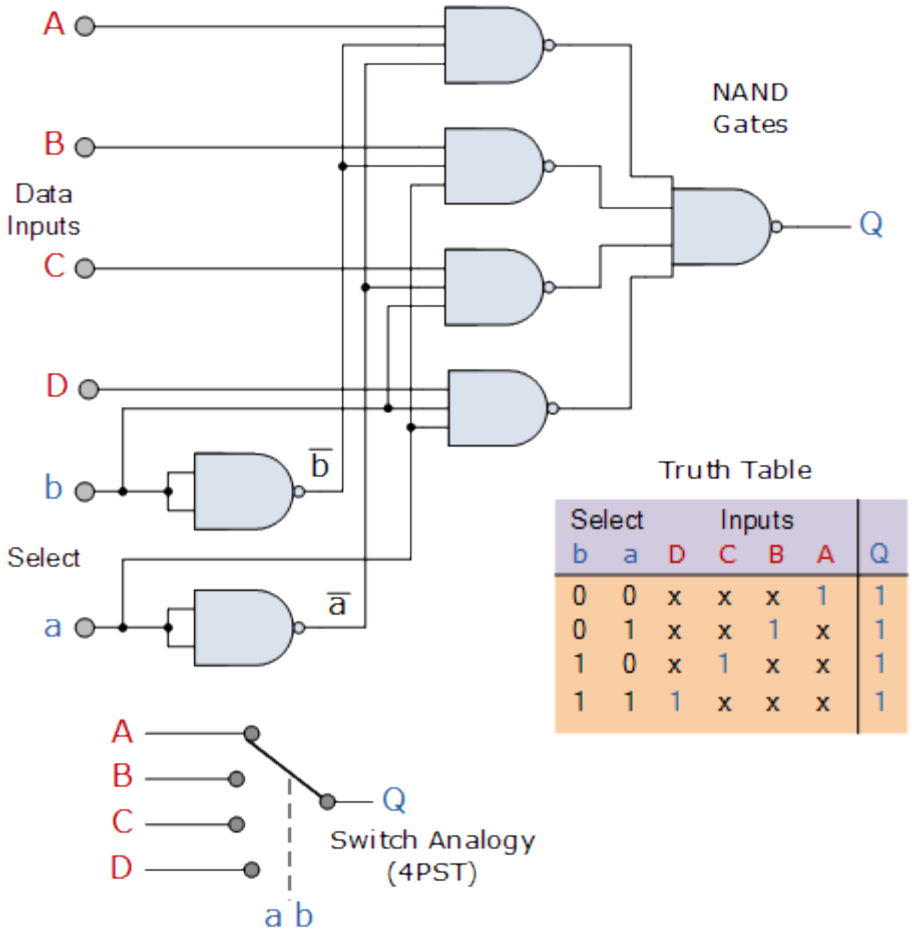
Ex 3: Finish the 2-to-1 MUX by all methods:

A, Draw the circuit.

B, Input the Truth table.

C, Input the Expression.

4-to-1 Channel Multiplexer



The Boolean expression for this 4-to-1 Multiplexer above with inputs A to D and data select lines a, b is given as:

$$Q = a.b.A + a.b.B + a.b.C + a.b.D$$

In this example at any one instant in time only ONE of the four analogue switches is closed, connecting only one of the input lines A to D to the single output at Q. As to which switch is closed depends upon the addressing input code on lines "a" and "b".

So for this example to select input B to the output at Q, the binary input address would need to be "a" = logic "1" and "b" = logic "0". Thus we can show the selection of the data through the multiplexer as a function of the data select bits as shown.

? **Ex 4: Finish the 4-to-1 MUX by all methods:**

A, Draw the circuit.

B, Input the Truth table.

C, Input the Expression.

Multiplexer Input Line Selection

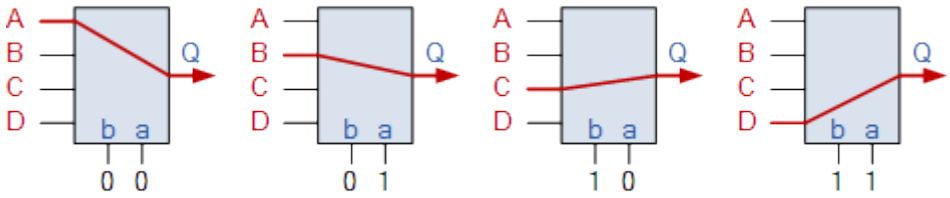


Figure 13. How a MUX work with diffent seletion input value.

Adding more control address lines, (n) will allow the multiplexer to control more inputs as it can switch 2^n inputs but each control line configuration will connect only ONE input to the output.

Then the implementation of the Boolean expression above using individual logic gates would require the use of seven individual gates consisting of AND, OR and NOT gates as shown.

4 Channel Multiplexer using Logic Gates

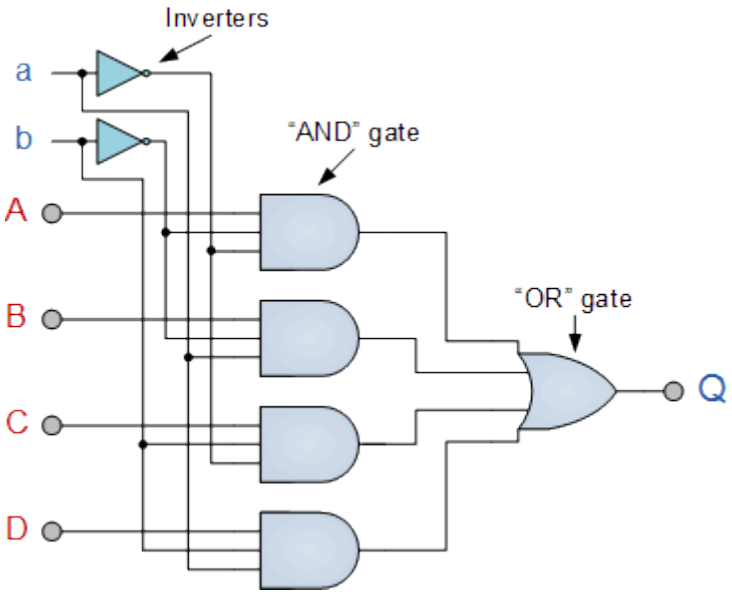


Figure 14. Logic gates diagram of 4-to-1 MUX.

The symbol used in logic diagrams to identify a multiplexer is as follows:

Multiplexer Symbol

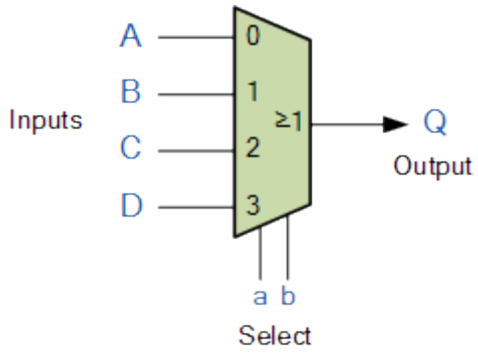


Figure 15. Multiplexer Symbol.

Multiplexers are not limited to just switching a number of different input lines or channels to one common single output. There are also types that can switch their inputs to multiple outputs and have arrangements or 4-to-2, 8-to-3 or even 16-to-4 etc configurations and an example of a simple Dual channel 4 input multiplexer (4-to-2) is given below:

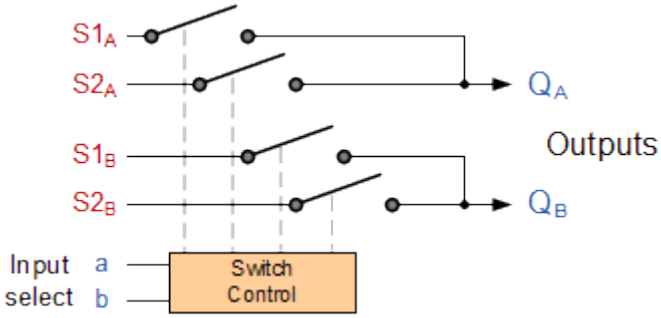


Figure 16. Diagram of a 4-to-2 MUX.

Here in this example the 4 input channels are switched to 2 individual output lines but larger arrangements are also possible. This simple 4-to-2 configuration could be used for example, to switch audio signals for stereo pre-amplifiers or mixers.

Ex 5: Finish the 8-to-1 MUX by using two 4-to-1 MUX and one 2-to-1 MUX

Preference: Slide Ch7. P.40

Ex 6: Finish the 8-to-2 MUX.

Could we use two 4-to-1 MUX?

Part 5. The DeMultiplexer

The data distributor, known more commonly as a Demultiplexer or “Demux” for short, is the exact opposite of the Multiplexer we saw in the previous tutorial.

The demultiplexer takes one single input data line and then switches it to any one of a number of individual output lines one at a time. The demultiplexer converts a serial data signal at the input to a parallel data at its output lines as shown below.

1-to-4 Channel De-multiplexer

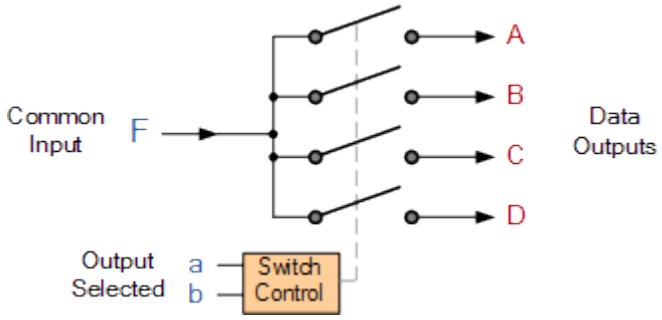


Figure 17. Diagram of 1-to-4 DeMUX.

a	b	F is connected to
0	0	A
0	1	B
1	0	C
1	1	D

Figure 18. Truth table of 1-to-4 DeMUX

The Boolean expression for this 1-to-4 Demultiplexer above with outputs A to D and data select lines a, b is given as:

$$F = a.b.A + a.b.B + a.b.C + a.b.D$$

The function of the Demultiplexer is to switch one common data input line to any one of the 4 output data lines A to D in our example above. As with the multiplexer the individual solid state switches are selected by the binary input address code on the output select pins “a” and “b” as shown.

Demultiplexer Output Line Selection

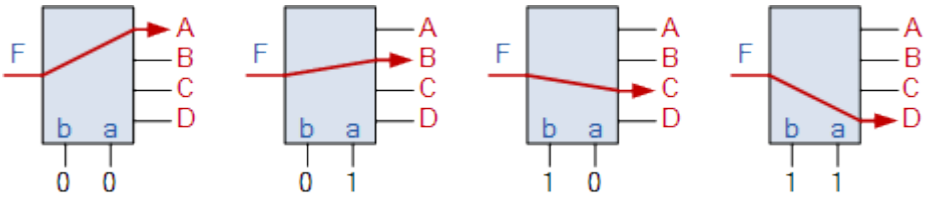


Figure 19. How a DeMUX work with diffent seletion input value.

As with the previous multiplexer circuit, adding more address line inputs it is possible to switch more outputs giving a 1-to-2^n data line outputs.

Some standard demultiplexer IC´s also have an additional “enable output” pin which disables or prevents the input from being passed to the selected output. Also some have latches built into their outputs to maintain the output logic level after the address inputs have been changed.

However, in standard decoder type circuits the address input will determine which single data output will have the same value as the data input with all other data outputs having the value of logic “0”.

The implementation of the Boolean expression above using individual logic gates would require the use of six individual gates consisting of AND and NOT gates as shown.

4 Channel Demultiplexer using Logic Gates

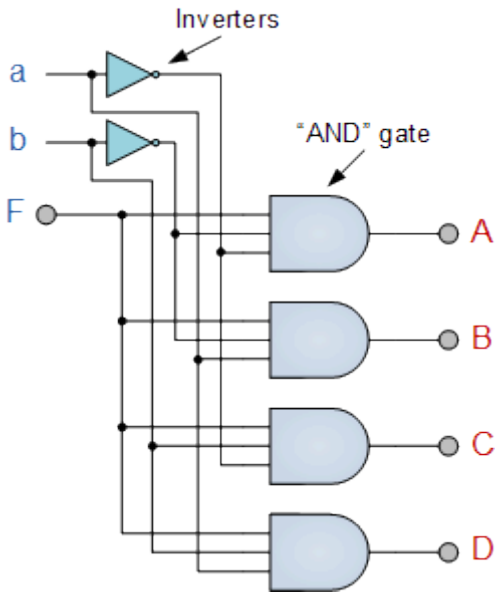


Figure 20. Diagram of Logic gates of a DeMUX.

The symbol used in logic diagrams to identify a demultiplexer is as follows.

The Demultiplexer Symbol

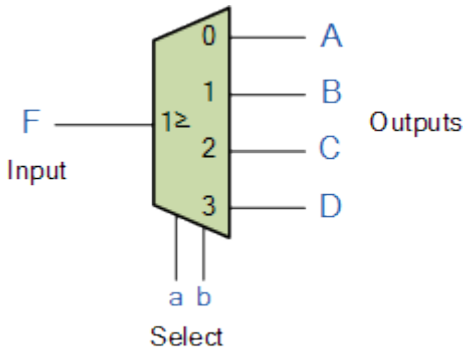


Figure 21. The 1-to-4 DeMUX symbol

Again, as with the previous multiplexer example, we can also use the demultiplexer to digitally control the gain of an operational amplifier as shown.

Ex 7: Finish the 1-to-4 DEMUX circuit

Preference: Slide Ch7. P.36

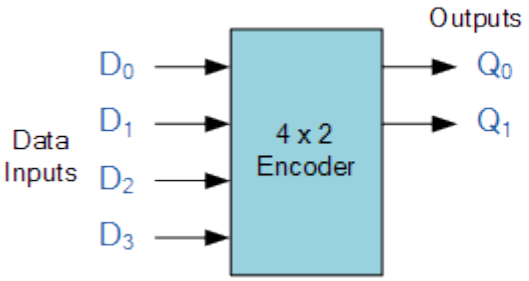
Part 6. The Encoder

Unlike a multiplexer that selects one individual data input line and then sends that data to a single output line or switch, Digital Encoder more commonly called a Binary Encodertakes ALL its data inputs one at a time and then converts them into a single encoded output. So we can say that a binary encoder, is a multi-input combinational logic circuit that converts the logic level “1” data at its inputs into an equivalent binary code at its output.

Generally, digital encoders produce outputs of 2-bit, 3-bit or 4-bit codes depending upon the number of data input lines. An “n-bit” binary encoder has 2^n input lines and n-bitoutput lines with common types that include 4-to-2, 8-to-3 and 16-to-4 line configurations.

The output lines of a digital encoder generate the binary equivalent of the input line whose value is equal to “1” and are available to encode either a decimal or hexadecimal input pattern to typically a binary or “B.C.D” (binary coded decimal) output code.

4-to-2 Bit Binary Encoder



Inputs				Outputs	
D_3	D_2	D_1	D_0	Q_1	Q_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1
0	0	0	0	x	x

Figure 22. Diagram of Block design of 4-to-2 Encoder and its truth table.

One of the main disadvantages of standard digital encoders is that they can generate the wrong output code when there is more than one input present at logic level “1”. For example, if we make inputs D_1 and D_2 HIGH at logic “1” both at the same time, the resulting output is neither at “01” or at “10” but will be at “11” which is an output binary number that is different to the actual input present. Also, an output code of all logic “0”s can be generated when all of its inputs are at “0” OR when input D_0 is equal to one.

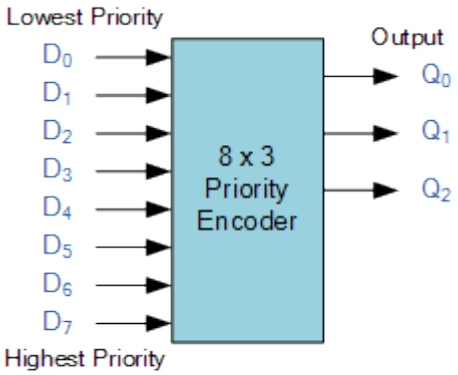
One simple way to overcome this problem is to “Prioritise” the level of each input pin. So if there is more than one input at logic level “1” at the same time, the actual output code would only correspond to the input with the highest designated priority. Then this type of digital encoder is known commonly as a Priority Encoder or P-encoder for short.

Priority Encoder

The Priority Encoder solves the problems mentioned above by allocating a priority level to each input. The priority encoders output corresponds to the currently active input which has the highest priority. So when an input with a higher priority is present, all other inputs with a lower priority will be ignored.

The priority encoder comes in many different forms with an example of an 8-input priority encoder along with its truth table shown below.

8-to-3 Bit Priority Encoder



Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	x	0	0	1
0	0	0	0	0	1	x	x	0	1	0
0	0	0	0	1	x	x	x	0	1	1
0	0	0	1	x	x	x	x	1	0	0
0	0	1	x	x	x	x	x	1	0	1
0	1	x	x	x	x	x	x	1	1	0
1	x	x	x	x	x	x	x	1	1	1

X = dont care

Figure 23. Diagram of Block design of 8-to-3 Priority Encoder.

Priority encoders are available in standard IC form and the TTL 74LS148 is an 8-to-3 bit priority encoder which has eight active LOW (logic “0”) inputs and provides a 3-bit code of the highest ranked input at its output.

Priority encoders output the highest order input first for example, if input lines “D2“, “D3” and “D5” are applied simultaneously the output code would be for input “D5” (“101”) as this has the highest order out of the 3 inputs. Once input “D5” had been removed the next highest output code would be for input “D3” (“011”), and so on.

The truth table for a 8-to-3 bit priority encoder is given as above. Where X equals “dont care”, that is logic “0” or a logic “1”.

From this truth table, the Boolean expression for the encoder above with data inputs $D_0$to D_7 and outputs Q_0, Q_1, Q_2 is given as:

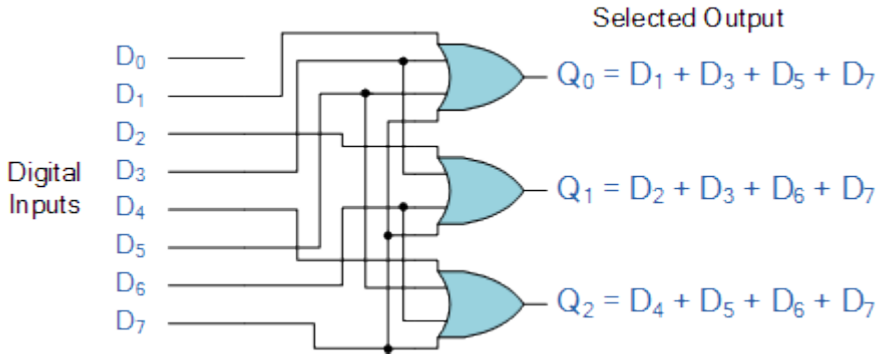


Figure 24. Digital Encoder using Logic Gates.

Digital Encoder Applications

Keyboard Encoder

Priority encoders can be used to reduce the number of wires needed in a particular circuits or application that have multiple inputs. For example, assume that a microcomputer needs to read the 104 keys of a standard QWERTY keyboard where only one key would be pressed either “HIGH” or “LOW” at any one time.

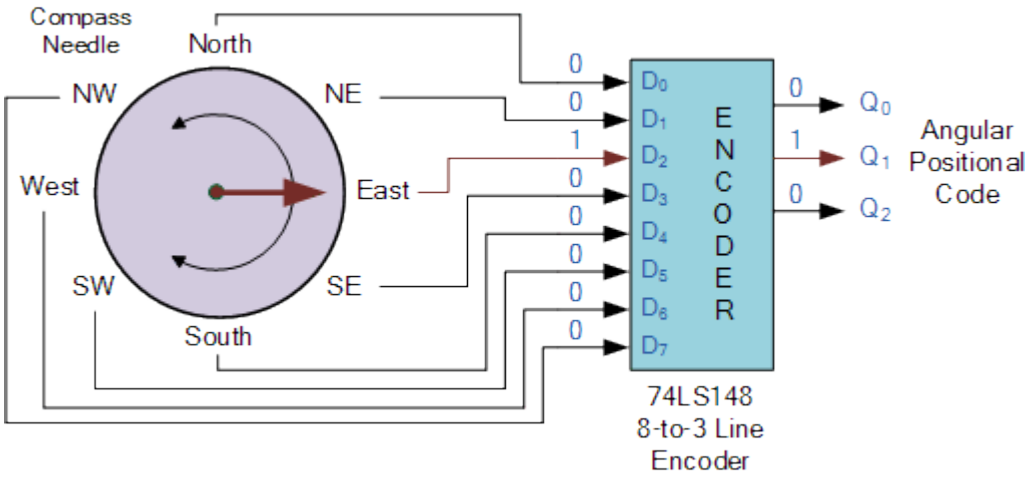
One way would be to connect all 104 wires from the individual keys on the keyboard directly to the computers input but this would be impractical for a small home PC. Another alternative and better way would be to interface the keyboard to the PC using a priority encoder.

The 104 individual buttons or keys could be encoded into a standard ASCII code of only 7-bits (0 to 127 decimal) to represent each key or character of the keyboard and then input as a much smaller 7-bit B.C.D code directly to the computer. Keypad encoders such as the 74C923 20-key encoder are available to do just that.

Positional Encoders

Another more common application is in magnetic positional control as used on ships navigation or for robotic arm positioning etc. Here for example, the angular or rotary position of a compass is converted into a digital code by a 74LS148 8-to-3 line priority encoder and input to the systems computer to provide navigational data and an example of a simple 8 position to 3-bit output compass encoder is shown below. Magnets and reed switches could be used at each compass point to indicate the needles angular position.

Priority Encoder Navigation



Interrupt Requests

Other applications especially for **Priority Encoders** may include detecting interrupts in microprocessor applications. Here the microprocessor uses interrupts to allow peripheral devices such as the disk drive, scanner, mouse, or printer etc, to communicate with it, but the microprocessor can only "talk" to one peripheral device at a time so needs some way of knowing when a particular peripheral device wants to communicate with it.

The processor does this by using "Interrupt Requests" or "IRQ" signals to assign priority to all the peripheral devices to ensure that the most important peripheral device is serviced first. The order of importance of the devices will depend upon their connection to the priority encoder.

IRQ Number	Typical Use	Description
IRQ 0	System timer	Internal System Timer.
IRQ 1	Keyboard	Keyboard Controller.
IRQ 3	COM2 & COM4	Second and Fourth Serial Port.
IRQ 4	COM1 & COM3	First and Third Serial Port.
IRQ 5	Sound	Sound Card.
IRQ 6	Floppy disk	Floppy Disk Controller.
IRQ 7	Parallel port	Parallel Printer.

IRQ 12	Mouse	PS/2 Mouse.
IRQ 14	Primary IDE	Primary Hard Disk Controller.
IRQ 15	Secondary IDE	Secondary Hard Disk Controller.

Because implementing such a system using priority encoders such as the standard 74LS148 priority encoder IC involves additional logic circuits, purpose built integrated circuits such as the 8259 Programmable Priority Interrupt Controller is available.

Part 7. Binary Decoder

The name “Decoder” means to translate or decode coded information from one format into another, so a binary decoder transforms “n” binary input signals into an equivalent code using 2^n outputs.

Binary Decoders are another type of digital logic device that has inputs of 2-bit, 3-bit or 4-bit codes depending upon the number of data input lines, so a decoder that has a set of two or more bits will be defined as having an *n*-bit code, and therefore it will be possible to represent 2^n possible values. Thus, a decoder generally decodes a binary value into a non-binary one by setting exactly one of its *n* outputs to logic “1”.

If a binary decoder receives n inputs (usually grouped as a single Binary or Boolean number) it activates one and only one of its 2n outputs based on that input with all other outputs deactivated.

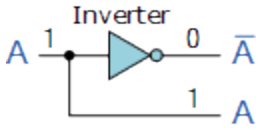


Figure 25. A 1-to-2 decoder.

So for example, an inverter (NOT-gate) can be classed as a 1-to-2 binary decoder as 1-input and 2-outputs (21) is possible because with an input A it can produce two outputs A and A (not-A) as shown.

Then we can say that a standard combinational logic decoder is an n-to-m decoder, where $m \leq 2n$, and whose output, Q is dependent only on its present input states. In other words, a binary decoder looks at its current inputs, determines which binary code or binary number is present at its inputs and selects the appropriate output that corresponds to that binary input.

A Binary Decoder converts coded inputs into coded outputs, where the input and output codes are different and decoders are available to “decode” either a Binary or BCD (8421 code) input pattern to typically a Decimal output code. Commonly available BCD-to-Decimal decoders include the TTL 7442 or the CMOS 4028. Generally, a decoders output code normally has more bits than its input code and practical “binary decoder” circuits include, 2-to-4, 3-to-8 and 4-to-16 line configurations.

An example of a 2-to-4 line decoder along with its truth table is given as:

A 2-to-4 Binary Decoders

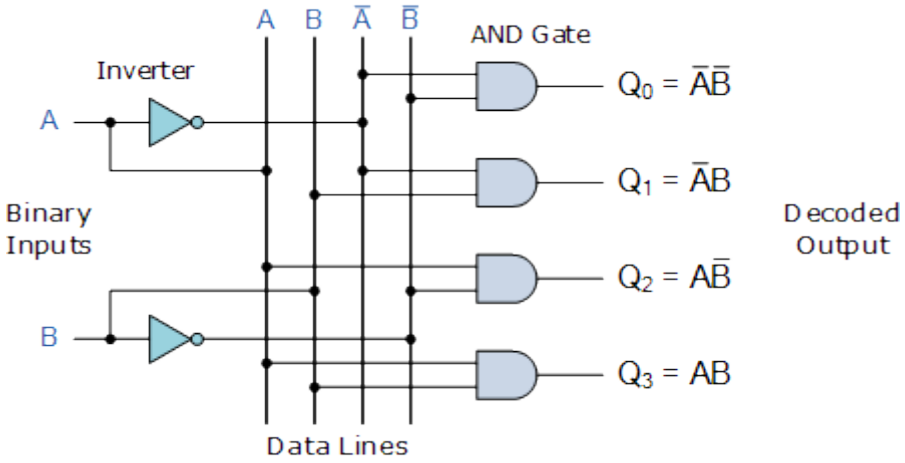
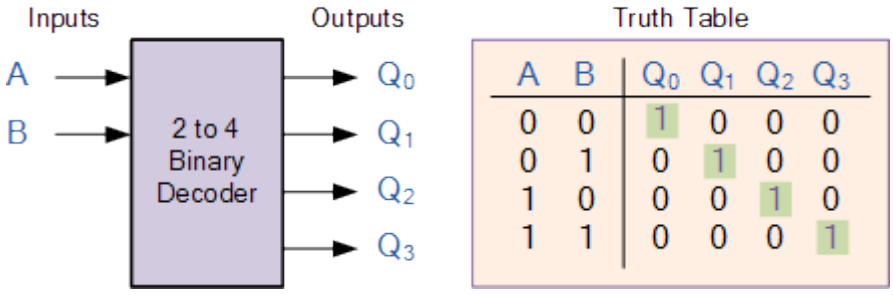


Figure 26. Logic gates design of a 2-to-4 Binary Decoders.



A	B	Q_0	Q_1	Q_2	Q_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

Figure 27. Block design and Truth table of a 2-to-4 Binary Decoders.

This simple example above of a 2-to-4 line binary decoder consists of an array of four AND gates. The 2 binary inputs labelled A and B are decoded into one of 4 outputs, hence the description of 2-to-4 binary decoder. Each output represents one of the miniterms of the 2 input variables, (each output = a miniterm).

The binary inputs A and B determine which output line from Q0 to Q3 is "HIGH" at logic level "1" while the remaining outputs are held "LOW" at logic "0" so only one output can be active (HIGH) at any one time. Therefore, whichever output line is "HIGH" identifies the binary code present at the input, in other words it "de-codes" the binary input.

Some binary decoders have an additional input pin labelled "Enable" that controls the outputs from the device. This extra input allows the decoders outputs to be turned "ON" or "OFF" as required. These types of binary decoders are commonly used as "memory address decoders" in microprocessor memory applications.

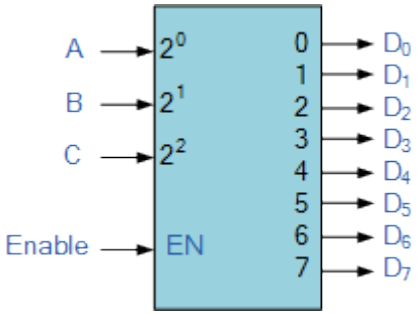


Figure 28. A 3-to-8 decoder with Enable control.

74LS138 Binary Decoder

We can say that a binary decoder is a demultiplexer with an additional data line that is used to enable the decoder. An alternative way of looking at the decoder circuit is to regard inputs A, B and C as address signals. Each combination of A, B or C defines a unique memory address.

We have seen that a 2-to-4 line binary decoder (TTL 74155) can be used for decoding any 2-bit binary code to provide four outputs, one for each possible input combination. However, sometimes it is required to have a **Binary Decoder** with a number of outputs greater than is available, so by adding more inputs, the decoder can potentially provide 2^n more outputs.

So for example, a decoder with 3 binary inputs ($n = 3$), would produce a 3-to-8 line decoder (TTL 74138) and 4 inputs ($n = 4$) would produce a 4-to-16 line decoder (TTL 74154) and so on. But a decoder can also have less than 2^n outputs such as the BCD to seven-segment decoder (TTL 7447) which has 4 inputs and only 7 active outputs to drive a display rather than the full 16 (2^4) outputs as you would expect.

Here a much larger 4 (3 data plus 1 enable) to 16 line binary decoder has been implemented using two smaller 3-to-8 decoders.

A 4-to-16 Binary Decoder Configuration

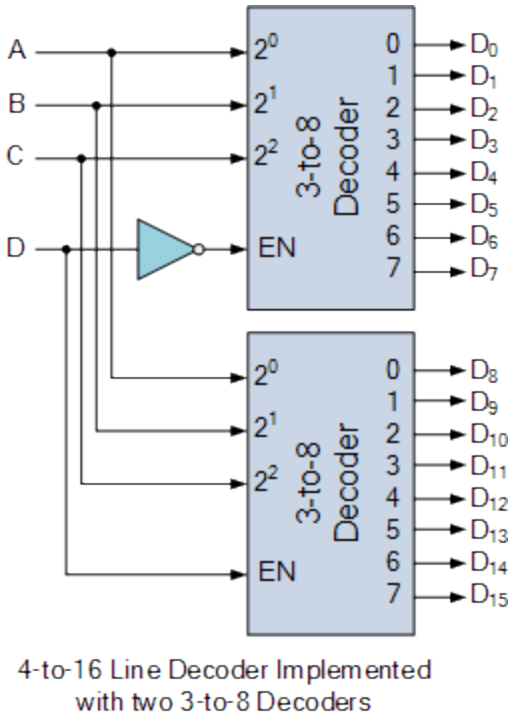


Figure 29. Make a larger Decoder.

Inputs A, B, C are used to select which output on either decoder will be at logic "1" (HIGH) and input D is used with the enable input to select which encoder either the first or second will output the "1".

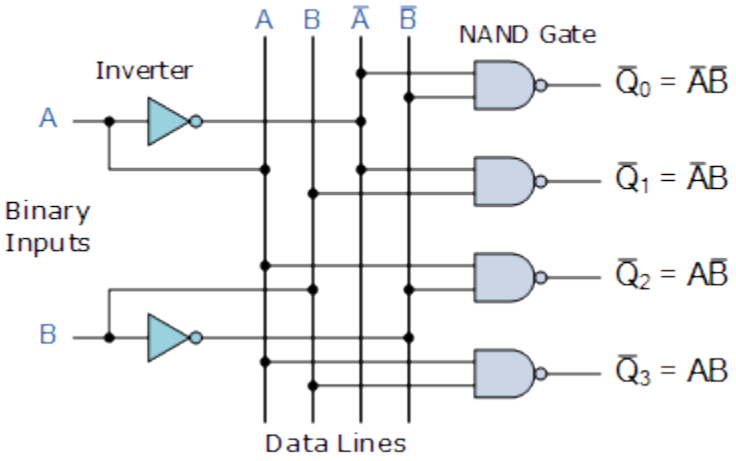
However, there is a limit to the number of inputs that can be used for one particular decoder, because as n increases, the number of AND gates required to produce an output also becomes larger resulting in the fan-out of the gates used to drive them becoming large.

This type of active-"HIGH" decoder can be implemented using just Inverters, (NOT Gates) and AND gates. It is convenient to use an AND gate as the basic decoding element for the output because it produces a "HIGH" or logic "1" output only when all of its inputs are logic "1".

But some binary decoders are constructed using NAND gates instead of AND gates for their decoded output, since NAND gates are cheaper to produce than AND's as they require fewer transistors to implement within their design.

The use of NAND gates as the decoding element, results in an active-"LOW" output while the rest will be "HIGH". As a NAND gate produces the AND operation with an inverted output, the NAND decoder looks like this with its inverted truth table.

2-to-4 Line NAND Binary Decoder



Truth Table

A	B	Q_0	Q_1	Q_2	Q_3
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

Figure 30. Using Line NAND to build a Decoder.

Then for the NAND decoder, only one output can be LOW and equal to logic "0" at any given time, with all the other outputs being HIGH at logic "1".

Decoders are also available with an additional "Enable" input pin which allows the decoded output to be turned "ON" or "OFF" by applying a logic "1" or logic "0" respectively to it. So for example, when the enable input is at logic level "0", (EN = 0) all outputs are "OFF" at logic "0" (for AND gates) regardless of the state of the inputs A and B.

Generally, to implement this enabling function the 2-input AND or NAND gates are replaced with 3-input AND or NAND gates. The additional input pin represents the enable function.

Memory Address Decoder

Binary Decoders are most often used in more complex digital systems to access a particular memory location based on an "address" produced by a computing device. In modern microprocessor systems the amount of memory required can be quite high and is generally more than one single memory chip alone.

One method of overcoming this problem is to connect lots of individual memory chips together and to read the data on a common "Data Bus". In order to

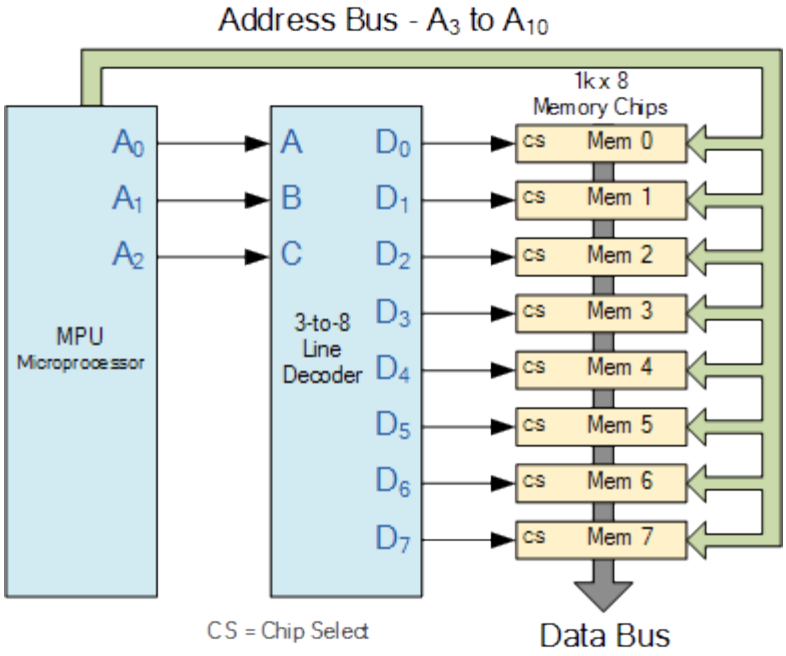
prevent the data being "read" from each memory chip at the same time, each memory chip is selected individually one at time and this process is known as **Address Decoding**.

In this type of application, the address represents the coded data input, and the outputs are the particular memory element select signals. Each memory chip has an input called **Chip Select** or **CS** which is used by the MPU (micro-processor unit) to select the appropriate memory chip when required. Generally a logic "1" on the chip select (CS) input selects the memory device while a logic "0" on the input de-selects it.

So by selecting or de-selecting each chip one at a time, allows us to select the correct memory address device for a particular address location. The advantage of address decoding is that when we specify a particular memory address, the corresponding memory location exists ONLY in one of the chips.

For example, Lets assume we have a very simple microprocessor system with only 1Kb (one thousand bytes) of RAM memory and 10 memory address lines available. The memory consists of 128×8-bit (128×8 = 1024 bytes) devices and for 1Kb we would need 8 individual memory chips but in order to select the correct memory chip we would also require a 3-to-8 line binary decoder as shown below.

Memory Address Decoding



The binary decoder requires only 3 address lines, (A_0 to A_2) to select each one of the 8 chips (the lower part of the address), while the remaining 8 address lines (A_3 to A_{10}) select the correct memory location on that chip (the upper part of the address).

Having selected a memory location using the address bus, the information at the particular internal memory location is sent to a common “Data Bus” for use by the microprocessor. This is of course a simple example but the principals remain the same for all types of memory chips or modules.

Binary Decoders are very useful devices for converting one digital format to another, such as binary or BCD type data into decimal or octal etc and commonly available decoder IC’s are the TTL 74LS138 3-to-8 line binary decoder or the 74ALS154 4-to-16 line decoder. They are also very useful for interfacing to 7-segment displays such as the TTL 74LS47 which we will look at in the next tutorial.

Part 8. Digital Comparator.

Digital or Binary Comparators are made up from standard AND, NOR and NOT gates that compare the digital signals present at their input terminals and produce an output depending upon the condition of those inputs.

For example, along with being able to add and subtract binary numbers we need to be able to compare them and determine whether the value of input A is greater than, smaller than or equal to the value at input B etc. The digital comparator accomplishes this using several logic gates that operate on the principles of Boolean Algebra. There are two main types of Digital Comparator available and these are.

1. Identity Comparator – an Identity Comparator is a digital comparator with only one output terminal for when A = B, either A = B = 1 (HIGH) or A = B = 0 (LOW)

2. Magnitude Comparator – a Magnitude Comparator is a digital comparator which has three output terminals, one each for equality, A = B greater than, A > B and less than A < B

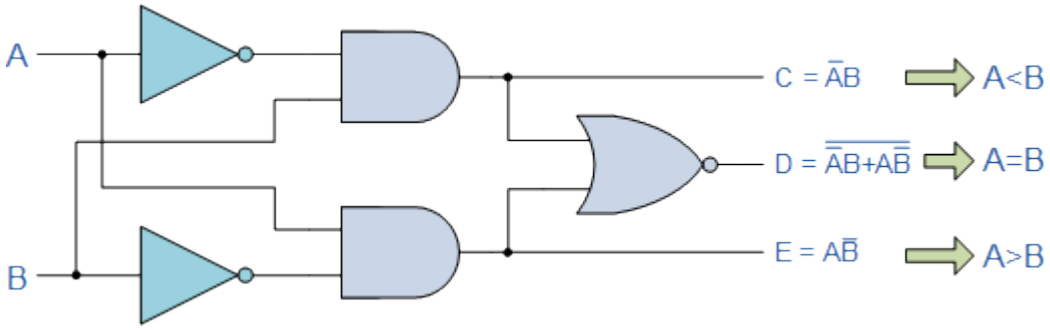
The purpose of a Digital Comparator is to compare a set of variables or unknown numbers, for example A (A1, A2, A3, …. An, etc) against that of a constant or unknown value such as B (B1, B2, B3, …. Bn, etc) and produce an output condition or flag depending upon the result of the comparison. For example, a magnitude comparator of two 1-bits, (Aand B) inputs would produce the following three output conditions when compared to each other.

A > B ; A == B ; A < B

Which means: A is greater than B, A is equal to B, or A is less than B

This is useful if we want to compare two variables and want to produce an output when any of the above three conditions are achieved. For example, produce an output from a counter when a certain count number is reached. Consider the simple 1-bit comparator below.

1-bit Digital Comparator Circuit



Then the operation of a 1-bit digital comparator is given in the following Truth Table.

Digital Comparator Truth Table

Inputs		**Outputs**		
B	A	A > B	A = B	A < B
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

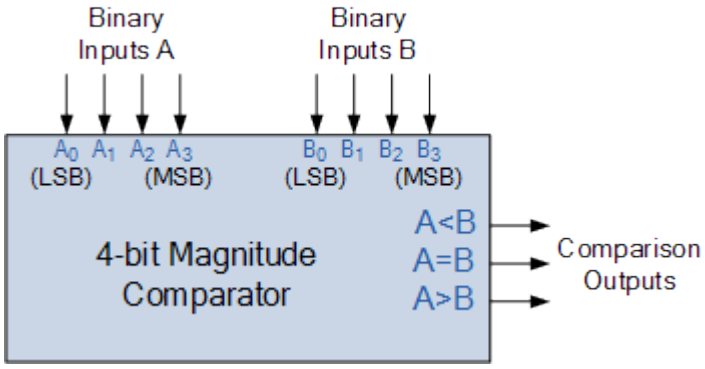
You may notice two distinct features about the comparator from the above truth table. Firstly, the circuit does not distinguish between either two "0" or two "1"'s as an output A = B is produced when they are both equal, either A = B = "0" or A = B = "1". Secondly, the output condition for A = B resembles that of a commonly available logic gate, the Exclusive-NOR or Ex-NOR function (equivalence) on each of the n-bits giving: $Q = A \oplus B$

Digital comparators actually use Exclusive-NOR gates within their design for comparing their respective pairs of bits. When we are comparing two binary or BCD values or variables against each other, we are comparing the "magnitude" of these values, a logic "0" against a logic "1" which is where the term **Magnitude Comparator** comes from.

As well as comparing individual bits, we can design larger bit comparators by cascading together n of these and produce a n-bit comparator just as we did for the n-bit adder in the previous tutorial. Multi-bit comparators can be constructed to compare whole binary or BCD words to produce an output if one word is larger, equal to or less than the other.

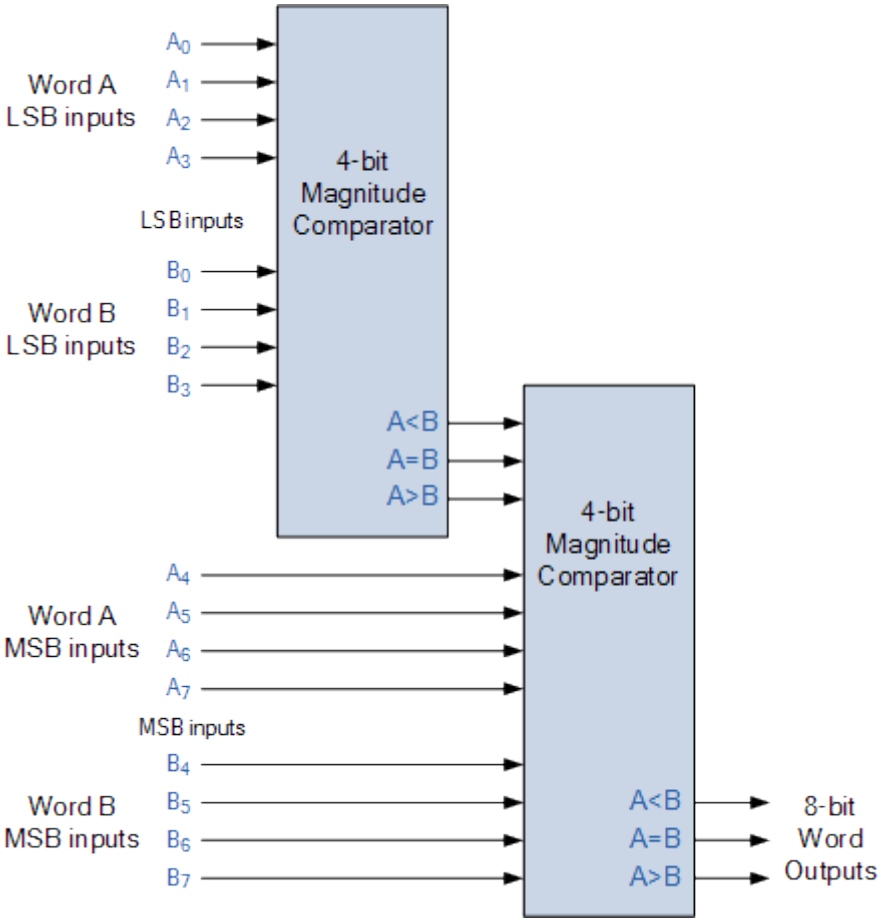
A very good example of this is the 4-bit **Magnitude Comparator**. Here, two 4-bit words ("nibbles") are compared to each other to produce the relevant output with one word connected to inputs A and the other to be compared against connected to input B as shown below.

4-bit Magnitude Comparator



Some commercially available digital comparators such as the TTL 74LS85 or CMOS 4063 4-bit magnitude comparator have additional input terminals that allow more individual comparators to be "cascaded" together to compare words larger than 4-bits with magnitude comparators of "n"-bits being produced. These cascading inputs are connected directly to the corresponding outputs of the previous comparator as shown to compare 8, 16 or even 32-bit words.

8-bit Word Comparator



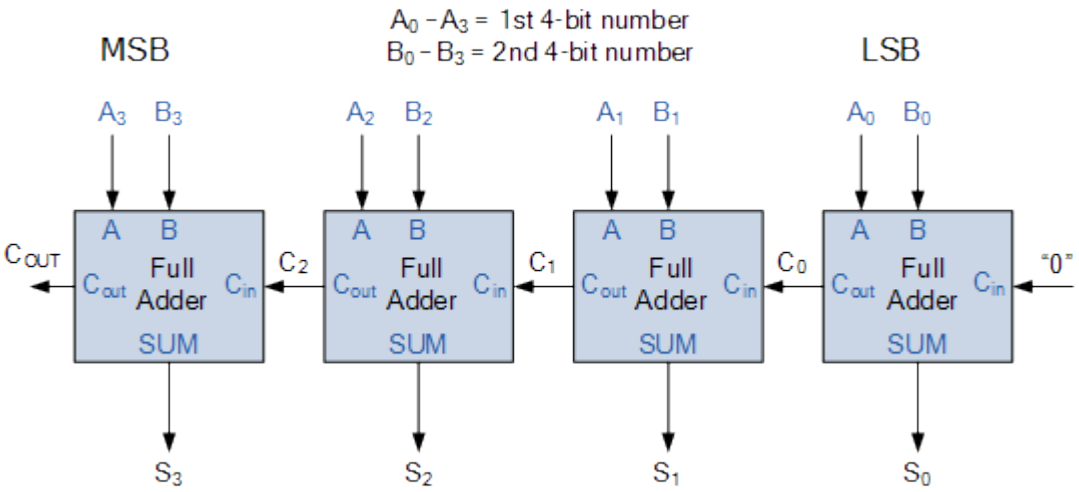
When comparing large binary or BCD numbers like the example above, to save time the comparator starts by comparing the highest-order bit (MSB) first. If equality exists, A = Bthen it compares the next lowest bit and so on until it reaches the lowest-order bit, (LSB). If equality still exists then the two numbers are defined as being equal.

If inequality is found, either A > B or A < B the relationship between the two numbers is determined and the comparison between any additional lower order bits stops. Digital Comparator are used widely in Analogue-to-Digital converters, (ADC) and Arithmetic Logic Units, (ALU) to perform a variety of arithmetic operations.

Excercises

1. A 4-bit Ripple Carry Adder.

A 4-bit Ripple Carry Adder



One main disadvantage of "cascading" together 1-bit **binary adders** to add large binary numbers is that if inputs A and B change, the sum at its output will not be valid until any carry-input has "rippled" through every full adder in the chain because the MSB (most significant bit) of the sum has to wait for any changes from the carry input of the LSB (less significant bit). Consequently, there will be a finite delay before the output of the adder responds to any change in its inputs resulting in a accumulated delay.

When the size of the bits being added is not too large for example, 4 or 8 bits, or the summing speed of the adder is not important, this delay may not be important. However, when the size of the bits is larger for example 32 or 64 bits used in multi-bit adders, or summation is required at a very high clock speed, this delay may become prohibitively large with the addition processes not being completed correctly within one clock cycle.

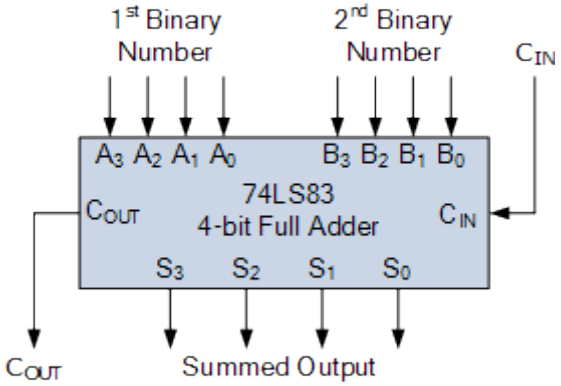
This unwanted delay time is called Propagation delay. Also another problem called "overflow" occurs when an n-bit adder adds two parallel numbers together whose sum is greater than or equal to 2n

One solution is to generate the carry-input signals directly from the A and B inputs rather than using the ripple arrangement above. This then produces another type of binary adder circuit called a Carry Look Ahead Binary Adder where the speed of the parallel adder can be greatly improved using carry-look ahead logic.

The advantage of carry look ahead adders is that the length of time a carry look ahead adder needs in order to produce the correct SUM is independent of the number of data bits used in the operation, unlike the cycle time a parallel ripple adder needs to complete the SUM which is a function of the total number of bits in the addend.

4-bit full adder circuits with carry look ahead features are available as standard IC packages in the form of the TTL 4-bit binary adder 74LS83 or the 74LS283 and the CMOS 4008 which can add together two 4-bit binary numbers and generate a SUM and a CARRYoutput as shown.

74LS83 Logic Symbol



2. Build a 4-bits Subtractor. https://www.electronics-tutorials.ws/combination/binary-subtractor.html
3. Using subcircuit “4-to-1 MUX” you’ve done in exercise 2 to make a “16-to-1” MUX.