Digital Electronics: Combinational Logic

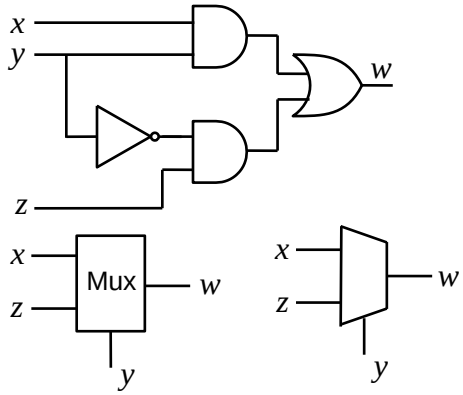
Beyond Simple Logic Gates

Multiplexor

- **Multiplexor (Mux)/selector** – chooses 1 of many inputs to steer to its single output under the direction of control inputs, e.g., if the input to a circuit can come from several places a Mux is one way to funnel the multiple sources selectively to the single ouput.

Multiplexor

- The hazard example is actually a 2-to-1 (2:1) Mux, i.e., it can select either input x or z to appear at output w under control of y



x	z	y	w
0	0	0	0
0	1	0	1
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	0
1	0	1	1
1	1	1	1

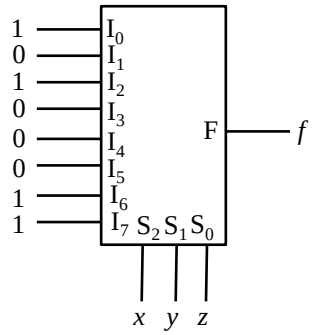
Multiplexor

- Clearly an n-to-1 (n:1) Mux is also possible. For example, an 8-to-1 (8:1) Mux will need 3 control inputs.
- A Mux can also be used to implement combinational logic functions. For example, an 8 input Mux can be used to implement functions having 3 variables expressed as a sum of minterms, i.e., DNF.

$$f = \bar{x}.\bar{y}.\bar{z} + \bar{x}.y.\bar{z} + x.y.\bar{z} + \bar{x}.y.z + x.y.z$$

Multiplexor

$$f = \bar{x}.\bar{y}.\bar{z} + \bar{x}.y.\bar{z} + x.y.\bar{z} + x.y.z$$



- The control inputs are used to select the minterms required at the output. The Mux is sometimes called a hardware *look-up* table.

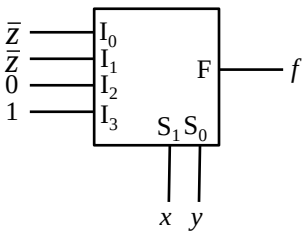
Multiplexor

- In this example if we use one of the inputs as a variable, then we can get away with a 4-to-1 (4:1) Mux

$$f = \bar{x}.\bar{y}.\bar{z} + \bar{x}.y.\bar{z} + x.y.\bar{z} + x.y.z$$

$$f = (\bar{x}.\bar{y} + \bar{x}.y).\bar{z} + x.y.(z + \bar{z})$$

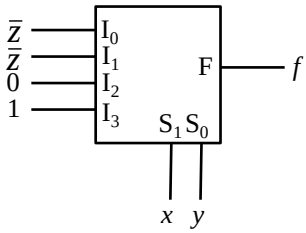
$$f = (\bar{x}.\bar{y} + \bar{x}.y).\bar{z} + x.y$$



Multiplexor

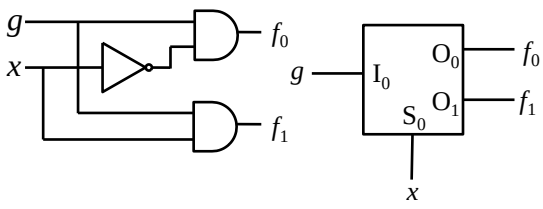
- We see it can also be designed via a truth table based approach, e.g.,

x	y	z	f	
0	0	0	1	$I_0 = \bar{z}$
0	0	1	0	
0	1	0	1	$I_1 = \bar{z}$
0	1	1	0	
1	0	0	0	$I_2 = 0$
1	0	1	0	
1	1	0	1	$I_3 = 1$
1	1	1	1	



Demultiplexor

- A demultiplexor is the opposite of a Mux, i.e., a single input is directed to exactly one of its outputs
- The truth table for a 1-to-2 (1:2) Demux (i.e., 1 control input and 2 outputs is:



g	x	f_0	f_1
0	0	0	0
1	0	1	0
0	1	0	0
1	1	0	1

Demultiplexor

- Clearly a larger Demux are also possible. For example, a 3-to-8 (3:8) Demux has 3 control inputs and 8 outputs.
- A related function is a *Decoder*. In this case the input g is permanently connected to a logic 1. This yields a 1-of-2 decoder (also known as a 1:2 decoder)

g	x	f_0	f_1
0	0	0	0
1	0	1	0
0	1	0	0
1	1	0	1

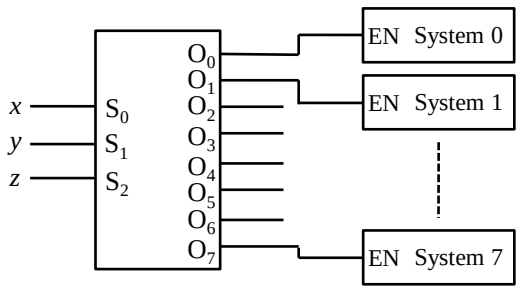
g =1

x	f_0	f_1
0	1	0
1	0	1

- See only one output is logic 1 at a time

Decoder

- Clearly an 1-of-n Decoder is possible. For example, a 1-of-8 Decoder (i.e., a 3:8 demux) has 3 control inputs and 8 outputs.
- A typical application would be to 'Enable (EN)' 1 out-of-n logic sub-systems.



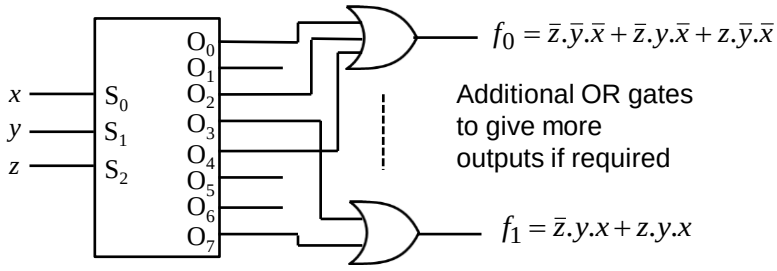
- So, letting x=1, y=z=0 will enable System 1

Decoder

- We can see that a 1-of-n Decoder will generate all the possible minterms having n variables.
- Consequently, a logical expression having DNF form can be implemented by ORing together the required minterms at the decoder output.
- Multiple output logic blocks can be created by using multiple OR gates at the decoder output, i.e., one for each output.

Decoder

- Decoder implementation of a 3 variable, 2 output combinational logic block.



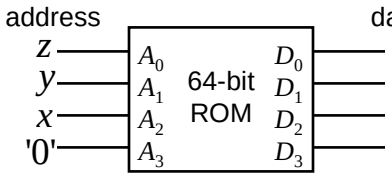
Even More Ways to Implement Combinational Logic

- We have seen how combinational logic can be implemented using logic gates (e.g., AND, OR), Mux and Demux.
- However, it is also possible to generate combinational logic functions using memory devices, e.g., Read Only Memories (ROMs)

ROM Overview

- A ROM is a data storage device:
  - Usually written into once (either at manufacture or using a programmer)
  - Read at will
  - Essentially is a look-up table, where a group of input lines (say n) is used to specify the *address* of locations holding m-bit *data* words
  - For example, if n = 4, then the ROM has 2^4 = 16 possible locations. If m = 4, then each location can store a 4-bit word
  - So, the total number of bits stored is $m \times 2^n$, i.e., 64 in the example (very small!) ROM

ROM Example



address (decimal)	x	y	z	f	data $D_3 D_2 D_1 D_0$
0	0	0	0	1	X X X 1
1	0	0	1	1	X X X 1
2	0	1	0	1	X X X 1
3	0	1	1	1	X X X 1
4	1	0	0	0	X X X 0
5	1	0	1	0	X X X 0
6	1	1	0	0	X X X 0
7	1	1	1	1	X X X 1

Design amounts to putting minterms in the appropriate address location

No logic simplification required

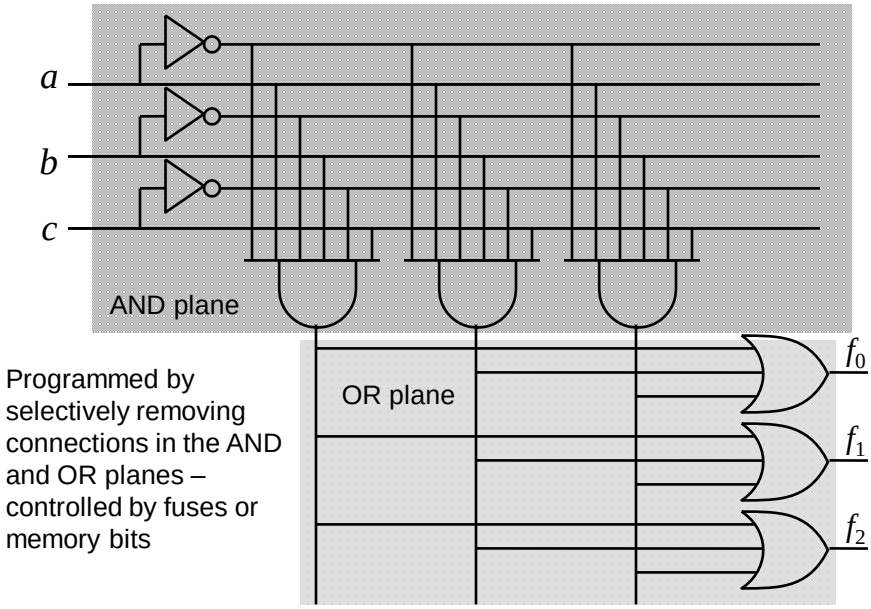
Useful if multiple Boolean functions are to be implemented, e.g., in this case we can easily do up to 4, i.e., 1 for each output line

Reasonably efficient if lots of minterms need to be generated

ROM Implementation

- Can be quite inefficient, i.e., become large in size with only a few non-zero entries, if the number of minterms in the function to be implemented is quite small
- Devices which can overcome these problems are known as programmable logic array (PLA)
- In PLAs, only the required minterms are generated using a separate AND plane. The outputs from this plane are ORed together in a separate OR plane to produce the final output

Basic PLA Structure

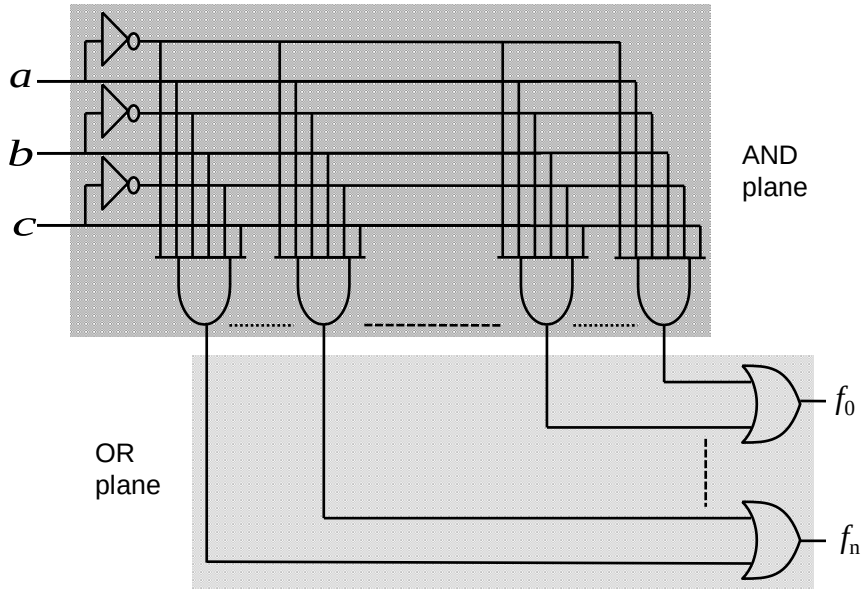


Programmed by selectively removing connections in the AND and OR planes – controlled by fuses or memory bits

Other PLA Style Structures

- In PLAs, only the required minterms are generated using a separate AND plane. Output from this plane are available to all OR gates to give the final output
- A modified structure known as Programmable Array Logic (PAL) does not have a programmable OR array and so outputs from the AND array can not be shared among the OR gates to give the final outputs.
- This simplifies the structure, but at the cost of lower efficiency

Basic PAL Structure



Other Memory Devices

- Non-volatile storage is offered by ROMs (and some other memory technologies, e.g., FLASH), i.e., the data remains intact, even when the power supply is removed
- Volatile storage is offered by Static Random Access Memory (SRAM) technology
  - Data can be written into and read out of the SRAM, but is lost once power is removed

Memory Application

- Memory devices are often used in computer systems
- The central processing unit (CPU) often makes use of busses (a bunch of wires in parallel) to access external memory devices
- The *address bus* is used to specify the memory location that is being read or written and the data bus conveys the data too and from that location
- So, more than one memory device will often be connected to the same data bus

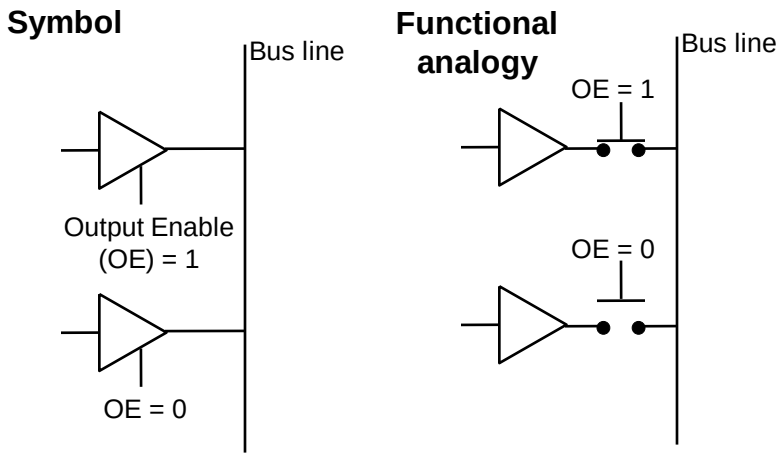
Bus Contention

- In this case, if the output from the data pin of one memory was a 0 and the output from the corresponding data pin of another memory was a 1, the data on that line of the data bus would be invalid
- So, how do we arrange for the data from multiple memories to be connected to the some bus wires?

Bus Contention

- The answer is:
  - *Tristate* buffers (or drivers)
  - Control signals
- A tristate buffer is used on the data output of the memory devices
  - In contrast to a normal buffer which is either 1 or 0 at its output, a tristate buffer can be electrically disconnected from the bus wire, i.e., it will have no effect on any other data currently on the bus – known as the *'high impedance'* condition

Tristate Buffer



Control Signals

- We have already seen that the memory devices have an additional control input (OE) that determines whether the output buffers are enabled.
- Other control inputs are also provided:
  - Write enable (WE). Determines whether data is written or read (clearly not needed on a ROM)
  - Chip select (CS) – determines if the chip is activated
- Note that these signals can be active low, depending upon the particular device