Digital Logic Design

Lecture # 8

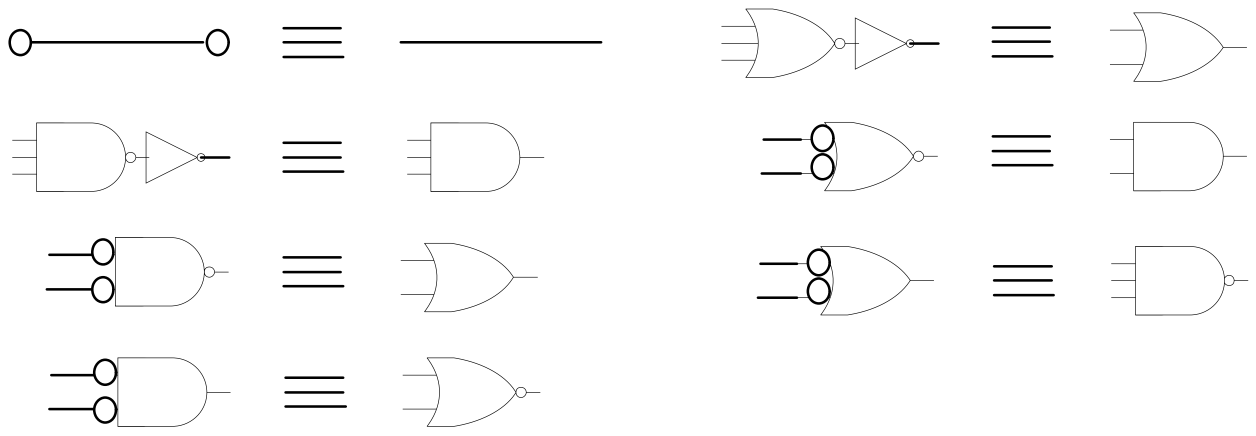
University of Tehran

Outline

- All NAND/NOR Gate Realization of a Circuit
- Active-Low and Active-High Logic
- RTL Design
- Decoder
- Realization of Switching Functions Using Decoders

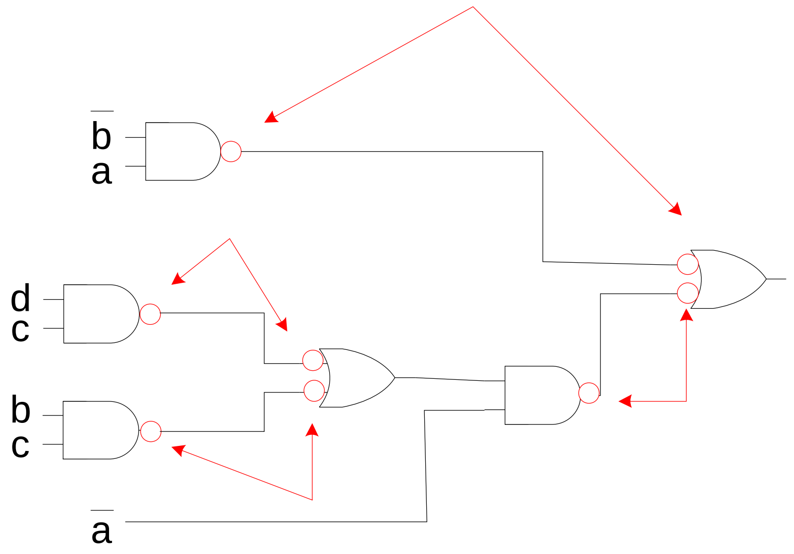
All NAND/NOR Gates Realization of a Circuit

- We need to turn expressions such as $a\overline{b} + \overline{a}(cd + bc)$ to a representation ready for an all nand/nor gate realization by applying De Morgan law recursively. We now want to obtain rules with which we can do the same act in a simpler and more visual form:



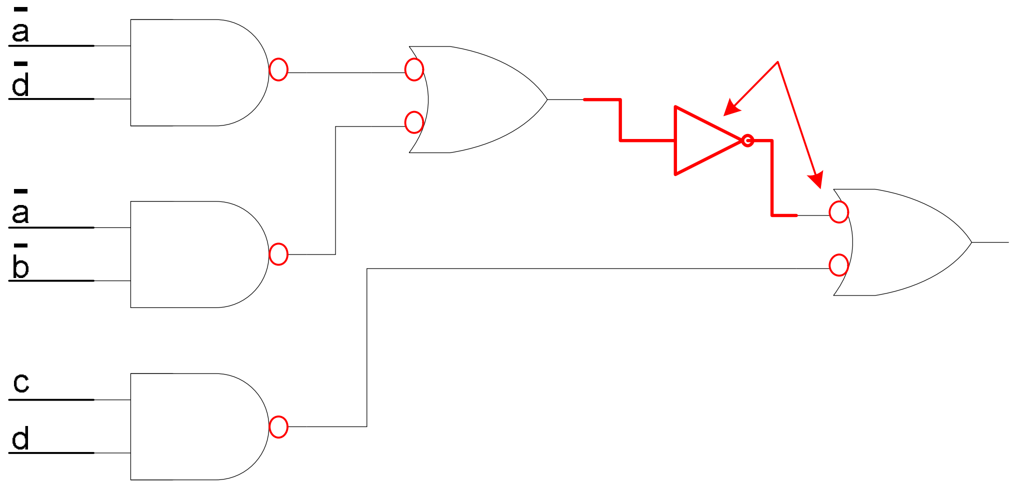
All NAND/NOR Gates Realization of a Circuit (continued...)

- For example, we want to convert the following representation to all nand gate representation:



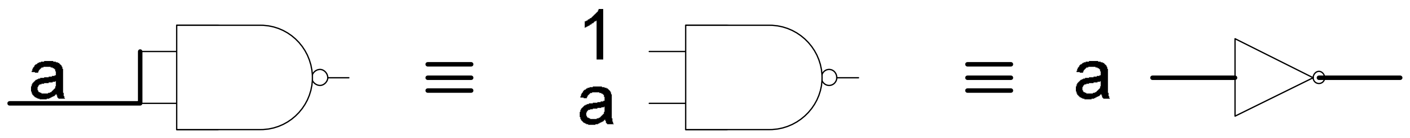
All NAND/NOR Gates Realization of a Circuit (continued...)

- Note: When we put an extra bubble somewhere in the circuit, another bubble must be accordingly placed somewhere to cancel it out.
- Example (In this example we have used an inverter to cancel off a bubble in one place.):



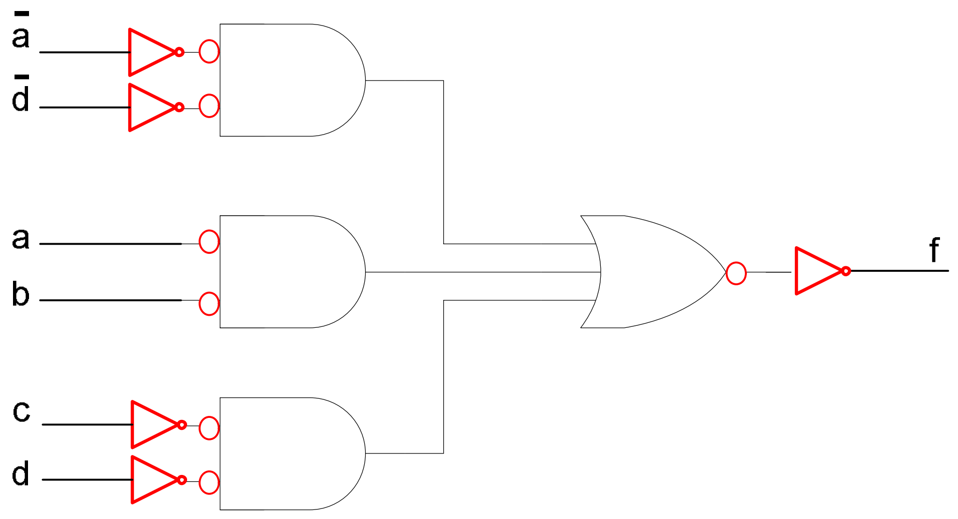
All NAND/NOR Gates Realization of a Circuit (continued...)

- If we had constructed the circuit in the last example in all and form, we would have had an increase in transistor use and delays (because of the pull ups and pull downs following each other in each and gate – and = nand + inverter – as discussed in lecture # 4.
- Quote: Inverters can also be considered nand gates:



All NAND/NOR Gates Realization of a Circuit (continued...)

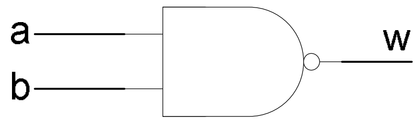
- Example: We want to construct an all nor representation of the following circuit:



- Quote: Using an all nand gate representation doesn't differ from an all nor gate representation in CMOS.

Active-Low and Active-High Logic

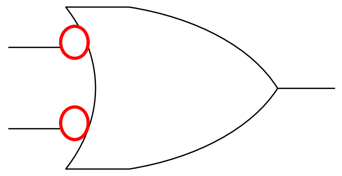
- So far we have naturally considered F to be logic 0 and T to be logic 1. Say we change our state of thought and look at the inputs as $T \equiv 1, F \equiv 0$ but on the output $T \equiv 0, F \equiv 1$. Then we would be saying that an input or output's definition of T or F can be desirably changed to one of these forms. With the above look to the meaning of T and F in inputs and outputs, the truth table of the NAND gate would be transformed to:



a	b	w
0 F	0 F	T 0
0 F	1 T	T 0
1 T	0 F	T 0
1 T	1 T	F 1

Active-Low and Active-High Logic (continued...)

- With the mentioned look at the meaning of T and F, the ‘nand’ gate can be called an ‘and’ gate with ACTIVE-LOW output. This state can be applied to any input or output signal. For instance the following gate is an ‘or’ gate with active-low inputs:



- Quote: Whenever the state of an input or output signal isn’t mentioned, we are implying ACTIVE-HIGH.

Active-Low and Active-High Logic (continued...)

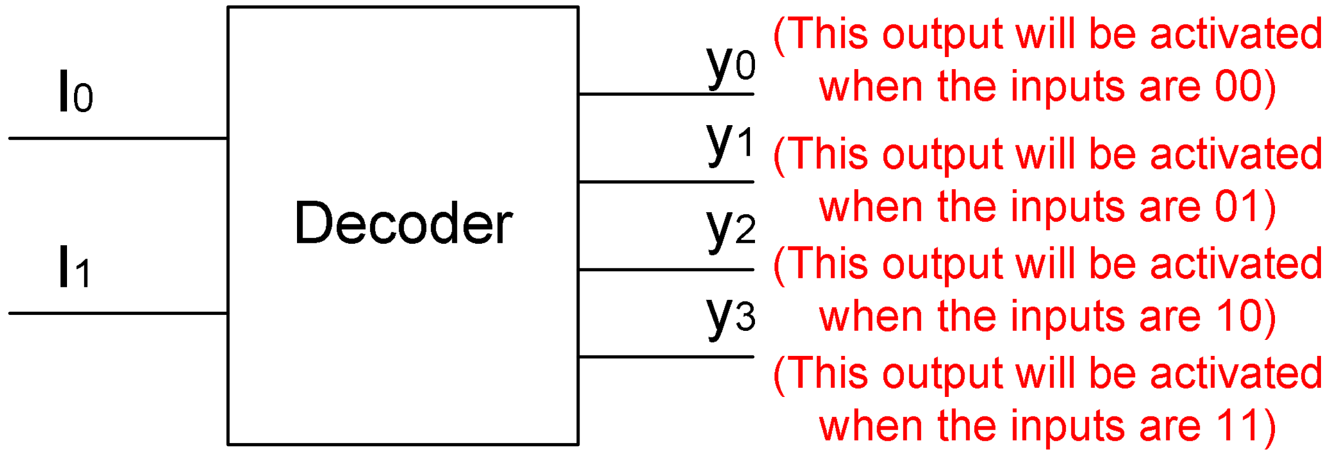
- A question that arises here in how we benefit of this state of thought? Consider for instance a package that gives us an active-low output, when using such a package, it’s output can be easily fed to a package with an active-low input, whereas feeding such a signal to a package with an active-high input causes a mismatch that must be fixed with glue logic. We were applying exactly the same concept when putting bubbles on the inputs and outputs of or/and gates to form a different representation of a particular circuit.

RTL Design

- So far all we have shown was based on gate level design. When considering functionality of elements as a base of our design, a gate level design isn’t meaningful enough. We will concentrate on functional packages from now on. This level of design referred to as RTL (Register Transfer Level) design and the components which are called RTL components are divided into combinational and sequential types. All of this certainly doesn’t mean our need for gates and transistors has been eliminated, especially when matching is needed between RTL components (glue logic).

Decoder

- We'll introduce this package by examining a binary decoder.

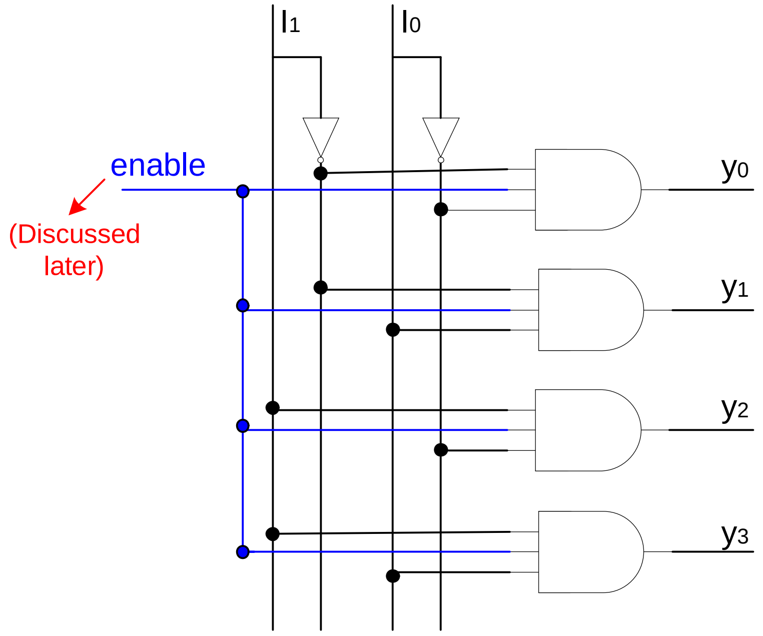


I_0
I_1
Decoder
y_0
(This output will be activated when the inputs are 00)
y_1
(This output will be activated when the inputs are 01)
y_2
(This output will be activated when the inputs are 10)
y_3
(This output will be activated when the inputs are 11)

Decoder (continued...)

- The decoder shown in the last slide is known as a 2-to-4 decoder and a simple structure for it can be the following circuit:

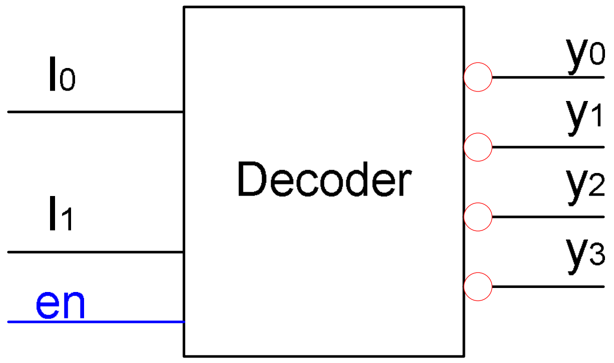
en	I_1	I_0	y_0	y_1	y_2	y_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	–	–	0	0	0	0



Decoder (continued...)

- Problems can occur in such a design. Firstly we have used and gates (constructed of nand+inverter in CMOS) when we don't know whether or not the user of our package really needs active-high outputs. We have also, for the same reason, used a lot of pull ups and pull downs and an excessive use of transistors.
- To resolve the mentioned problems, we change the and gates to nand gates and so our truth table will be:

en	I_1	I_0	y_0	y_1	y_2	y_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	–	–	1	1	1	1

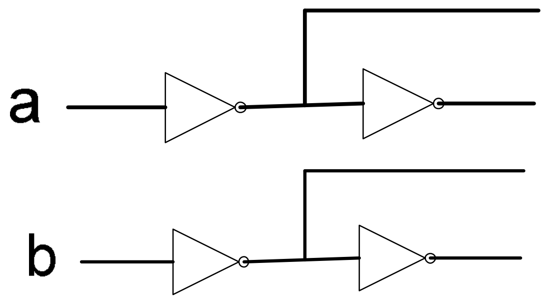


Decoder (continued...)

- Such a package still isn’t general as such and needs to have the ability of cascading, for the user to be able to construct larger decoders through combining smaller ones. Adding an extra input for enabling/disabling the output will help us in this arena as we will show later on.
- Looking at our design so far may cause some new questions. For instance have we told our user that any signal feeds to the EN input has been used to drive 4 gates? Have we not misled the user by not using a standard method?

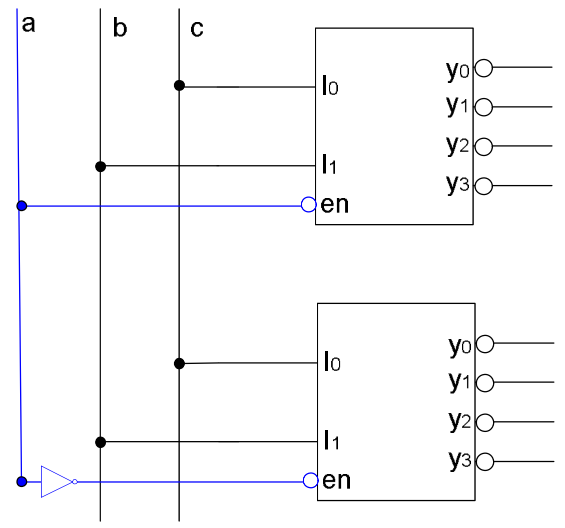
Decoder (continued...)

- There can be two possible ways to resolve this, one is to document the number of gates a particular input is used to drive. The second and preferable solution is to construct our structure by a convention that no input is used to drive more than one gate. To do this, we need to buffer the EN input through an inverter that will change these inputs' activity level to low. The same can be done for inputs 'a' and 'b' without changing their activity level, as shown in the following figure:

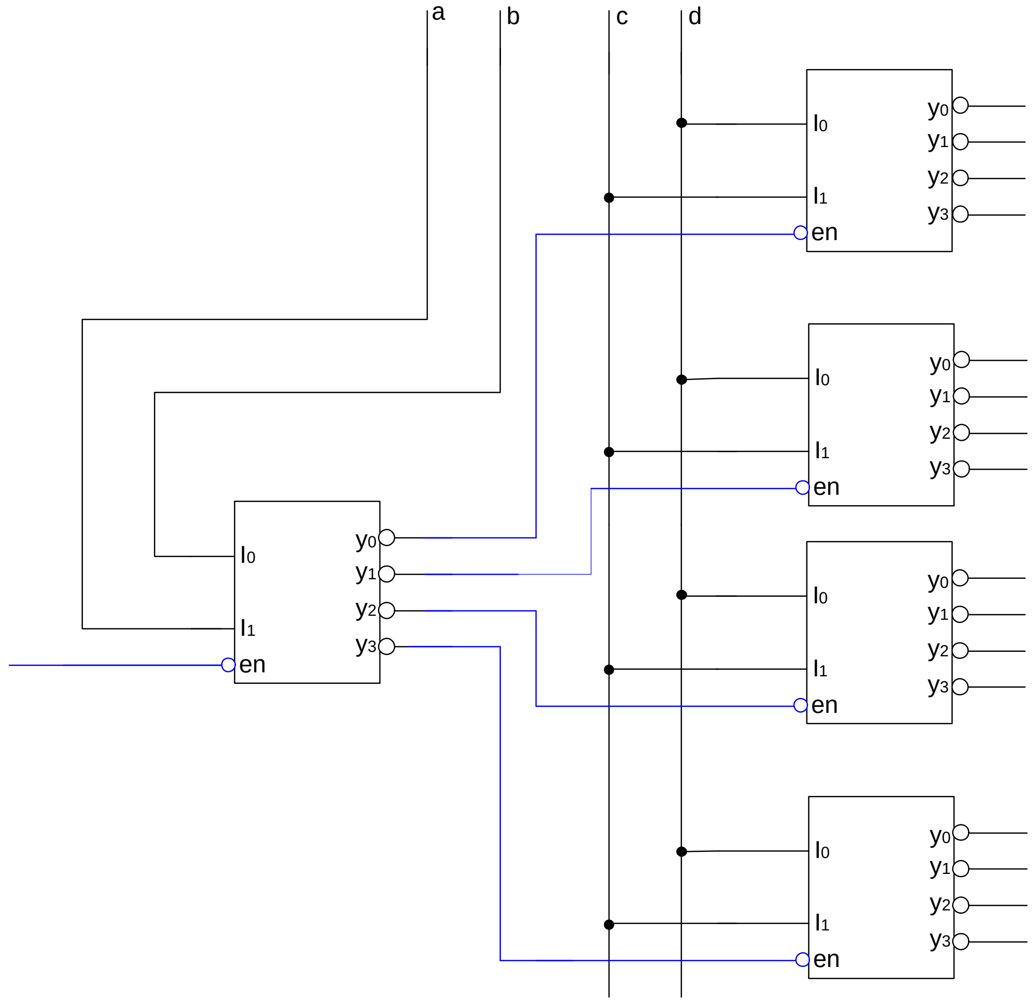


Decoder (continued...)

- We will now show how adding the EN input has given our package the ability of cascading. We'll show this fact by constructing a 3-to-8 and a 4-to-16 decoder using 2-to-4 decoders:

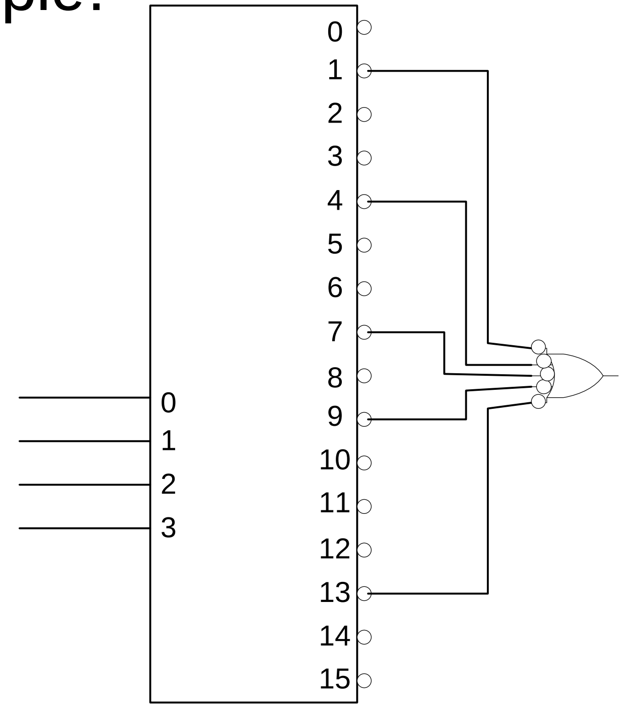


Decoder (continued...)



Realization of Switching Functions Using Decoders

- Decoders can be used to realize switching functions. To do this we will need to or outputs (relative to the corresponding minterms) of our decoder with an active-low inputs or gate. Example:



0
1
2
3
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15