

Introduction to Digital Systems

Part II (4 lectures)

2023/2024

Combinational Logic Blocks

Arnaldo Oliveira, Augusto Silva, Ioulia Skliarova

Lecture 5 contents

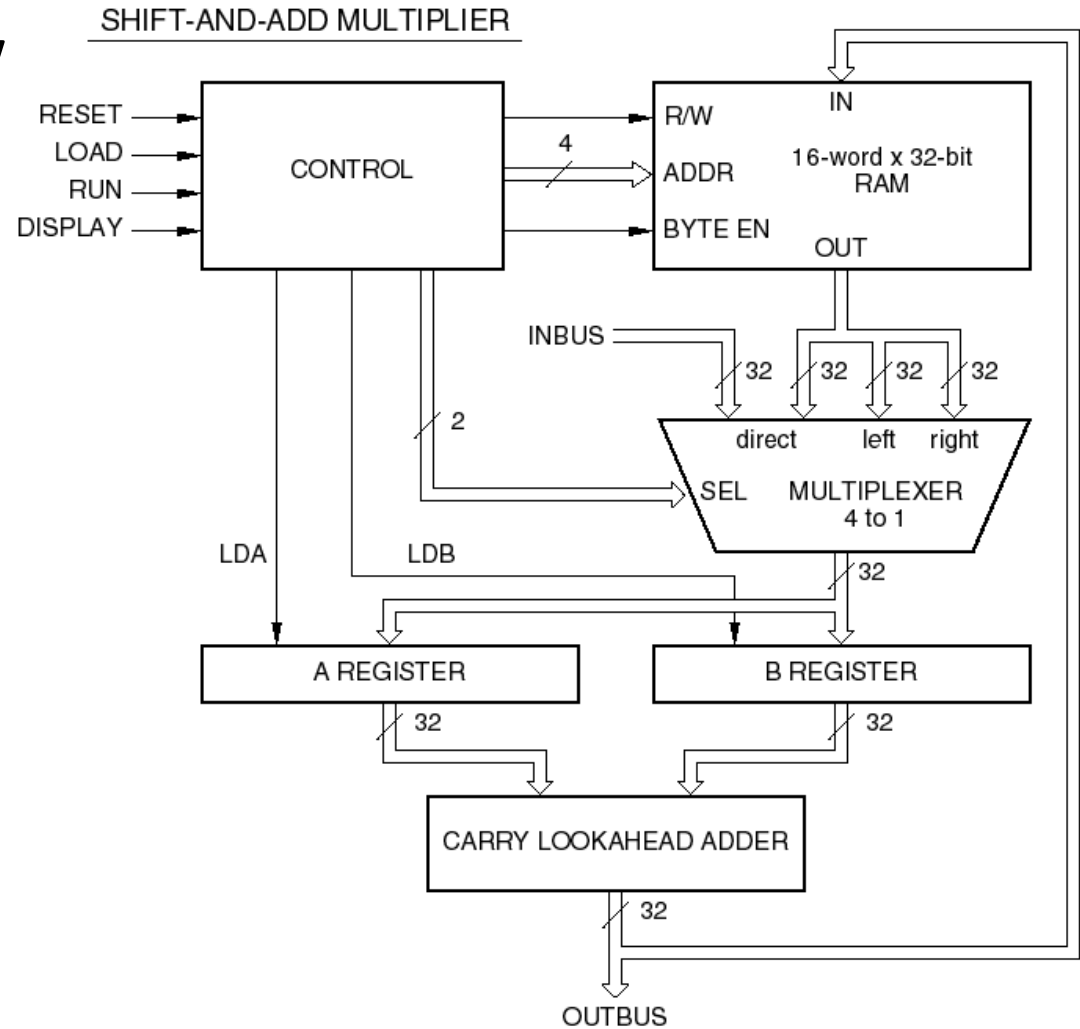
- About project documentation
- Block oriented combinational logic design
- Decoders
- Encoders

Documentation

- Essential in the whole project development cycle
- Block diagrams
- Logic circuits
- Hardware Description Languages
 - (VHDL, Verilog)
- Timing diagrams
- Electrical circuits
- Component specs

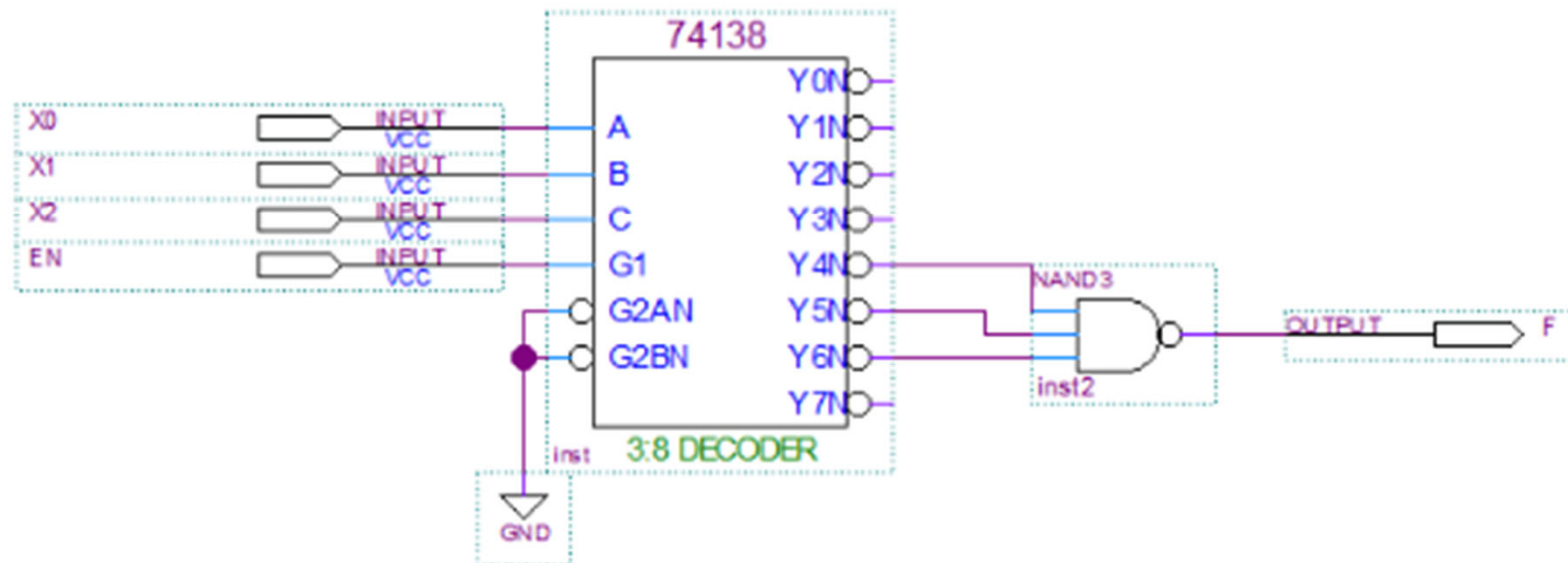
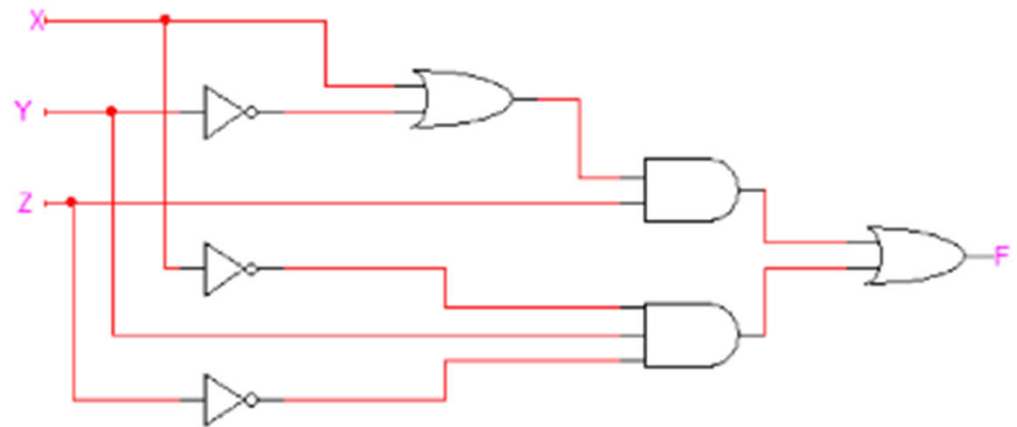
Block Diagrams

- Architectural view
- Functional hints



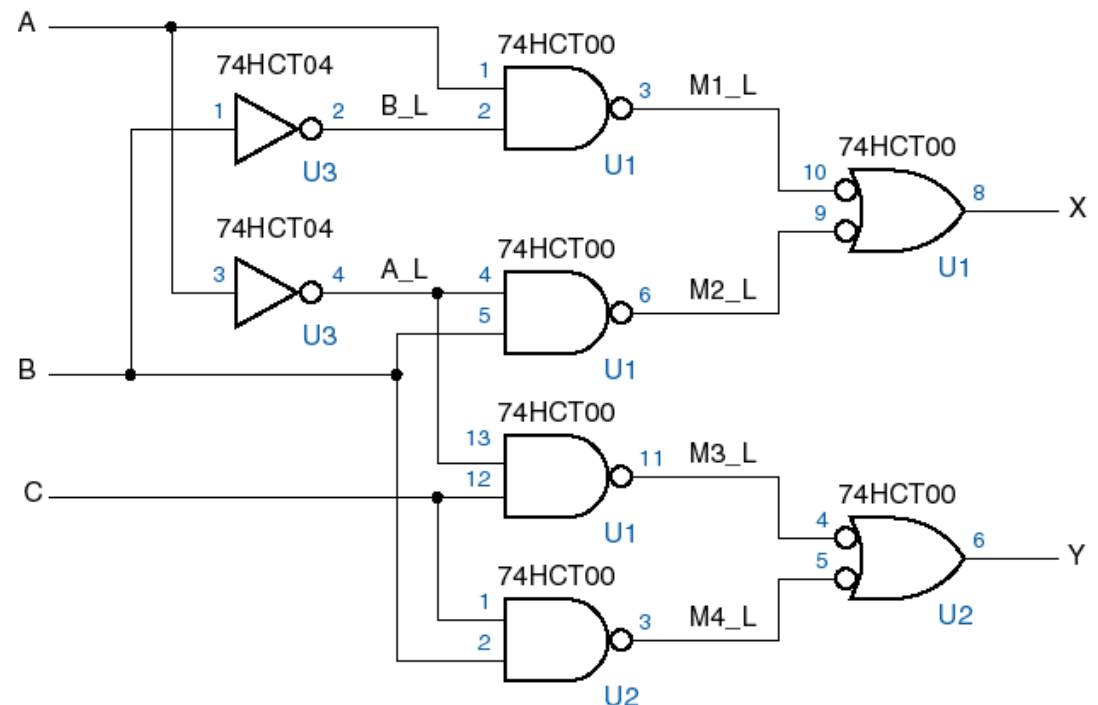
Logic Circuits

- Elementary gates
- Logic blocks
- Signal naming only



Electrical Circuits

- Component references
- Pin references
- Signal naming
- Connectors
- ...

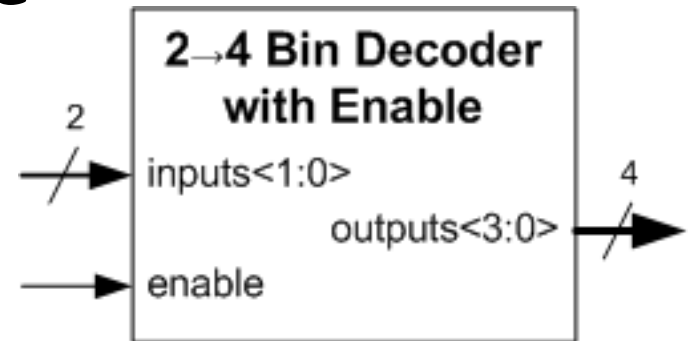


HDL's

- Coming soon ...
- Modeling hardware with code
- A flavor of VHDL

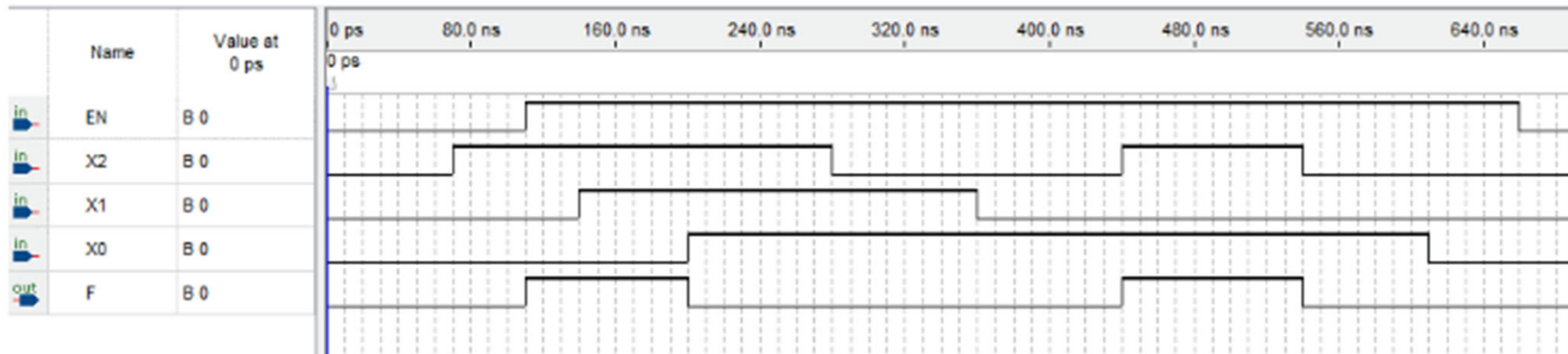
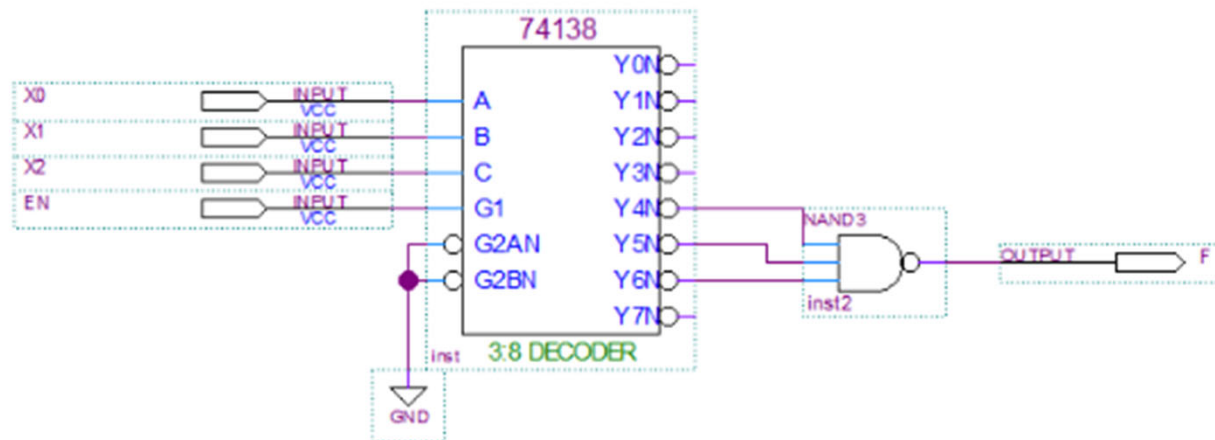
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity Dec2_4En is
    port(enable : in std_logic;
          inputs : in std_logic_vector (1 downto 0);
          outputs : out std_logic_vector (3 downto 0));
end Dec2_4En;
architecture BehavAssign of Dec2_4En is
begin
    outputs <= "0000" when (enable = '0') else
               "0001" when (inputs = "00") else
               "0010" when (inputs = "01") else
               "0100" when (inputs = "10") else
               "1000";
end BehavAssign
```



Timing Diagrams

- A core skill for analysis and simulation

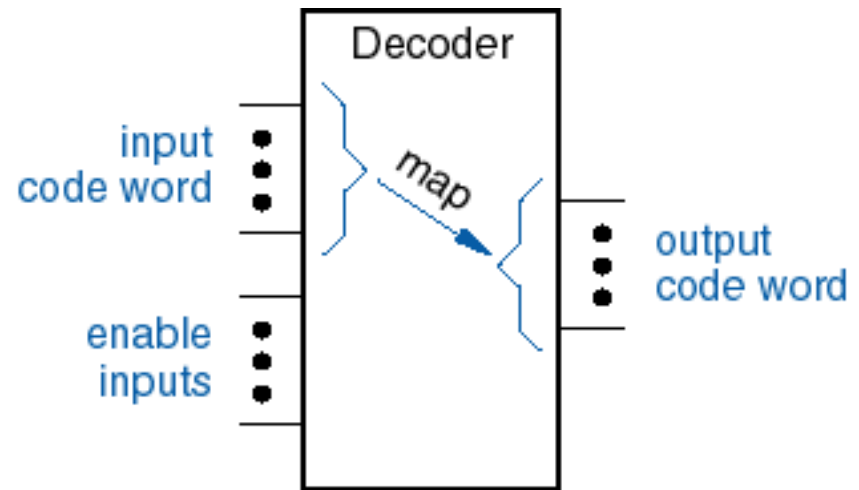


Beyond elementary gates

- Combinational logic blocks
- Encapsulation of specific behavior within a functional block
 - Decoders / Encoders
 - Multiplexers / Demultiplexers
 - Arithmetic blocks
 - Adders / Subtractors
 - Comparators
 - Multipliers
 - Arithmetic Logic Units (ALU)

Decoders

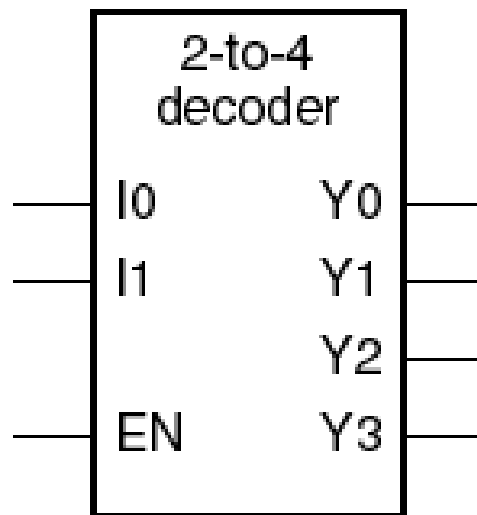
- Generic description



- A decoder implements a mapping between input code word and output code word
- Behavior is externally controlled by “enable” inputs

$n:2^n$ decoders

- Restrict the #inputs – #outputs relation to $n:2^n$
- Impose a “1 out of 2^n ” output code
- Then we have a standard $n:2^n$ binary decoder



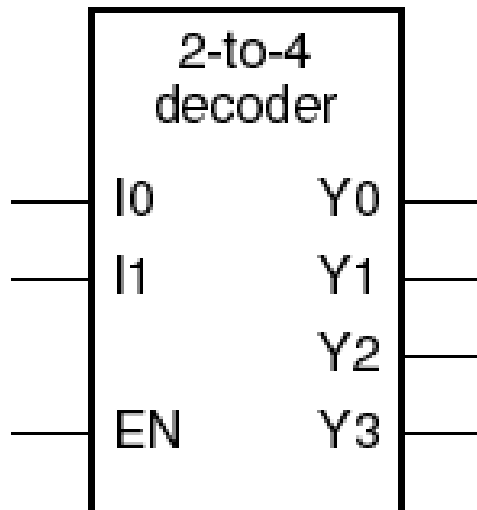
<i>Inputs</i>			<i>Outputs</i>			
EN	I_1	I_0	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

“x” (don’t care)

2:4 Decoder

- Things we have to know:

Block diagram



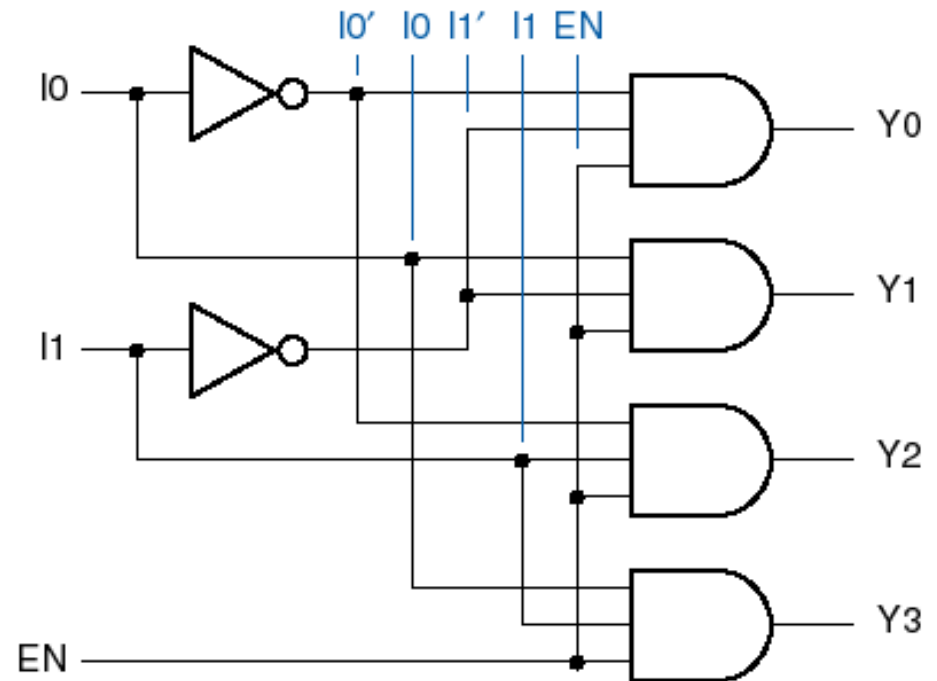
Functional Truth Table

<i>Inputs</i>			<i>Outputs</i>			
EN	I1	I0	Y3	Y2	Y1	Y0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

"1" out of 4 code words
Look at the diagonal layout

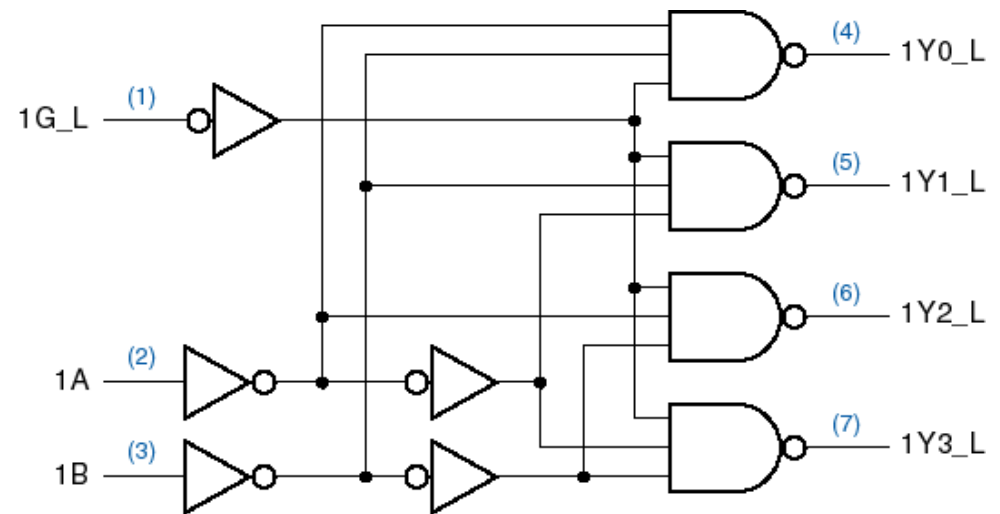
2:4 Decoder

- Things we have to know:
 - Write the output equations
 - Be aware of the role of the EN input

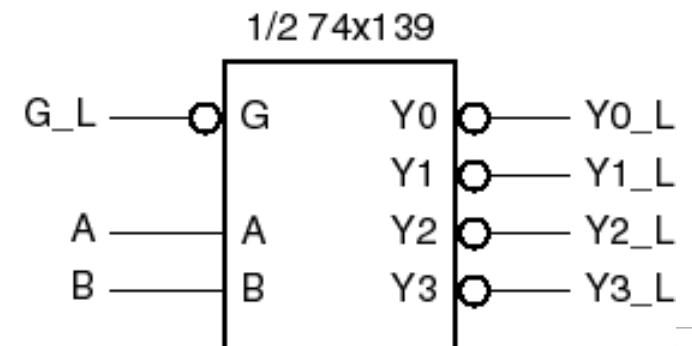
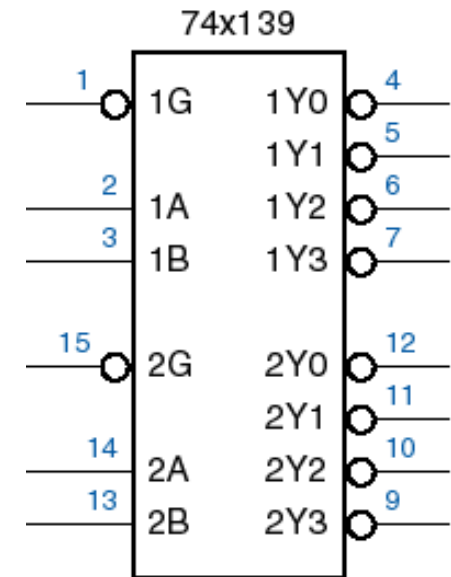
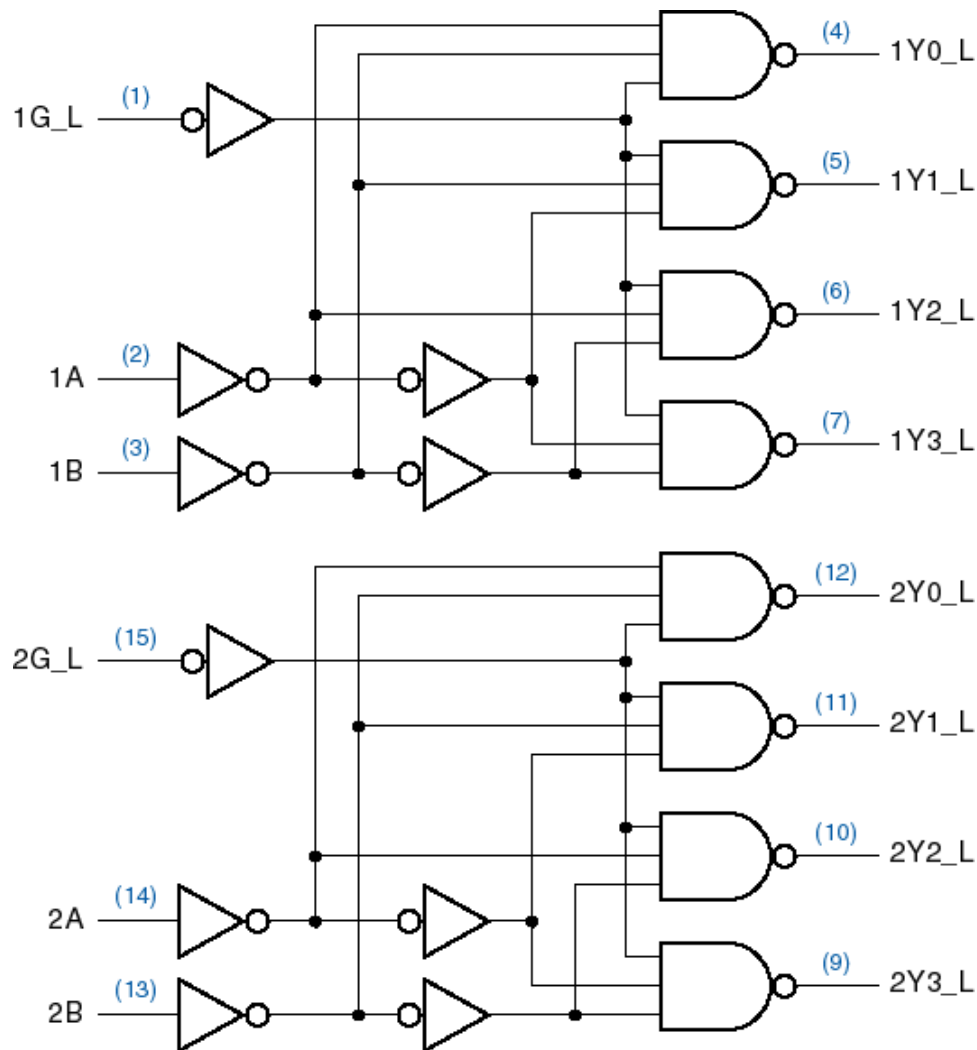


2:4 Decoder

- Active-low version
- Write the output equations
- Build the truth table

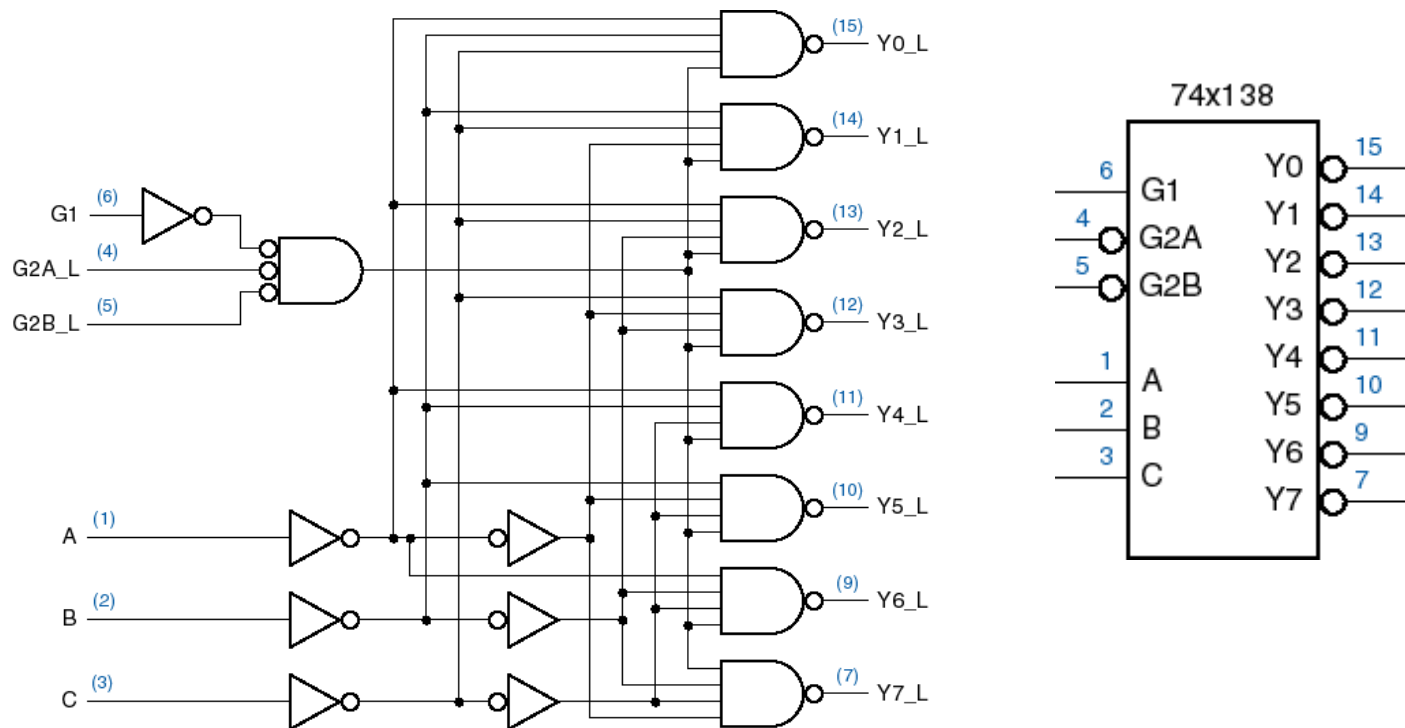


Commercial Models



3:8 Decoder

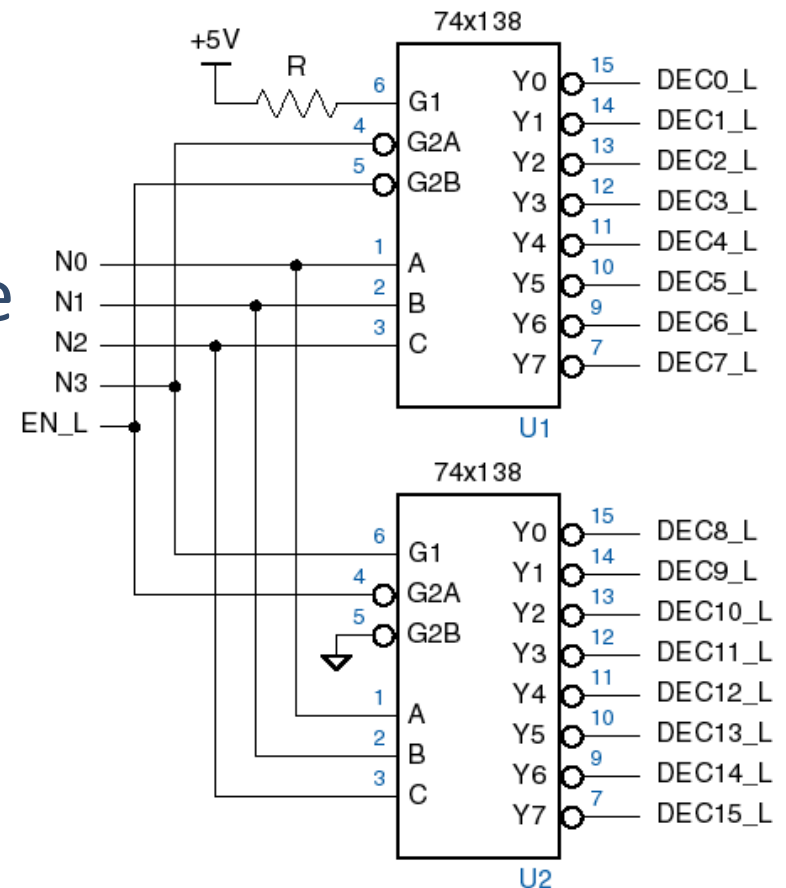
- Twofold increase in the number of output products



Build the Truth Table for the “enabled” mode

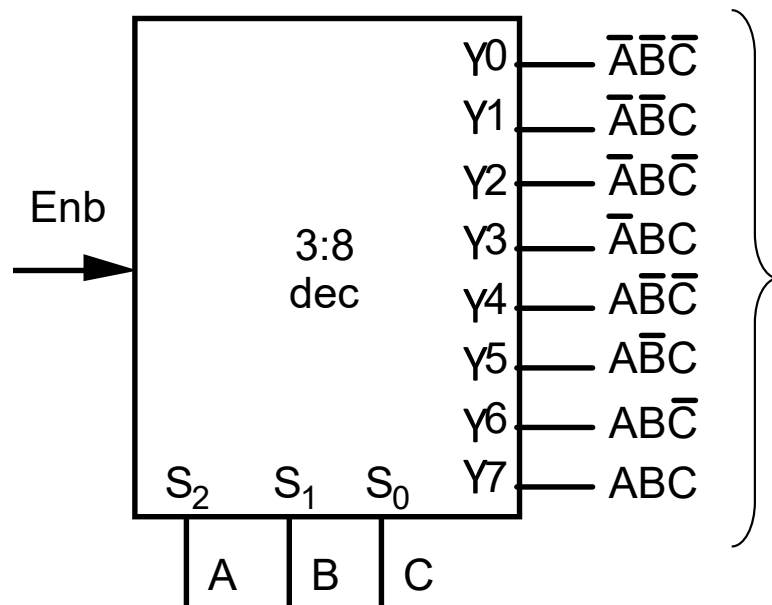
Scaling-up

- Decoder cascades
 - 4:16 with 2x(3:8)
 - Look at the role of the enable inputs
 - Figure out other cascading structures:
 - 4:16 from 2:4 Decoder blocks
 - 5:32 from 2:4 + 3:8 Decoder blocks



The Decoder as a minterm generator

- Recall the “internals” of the decoding block



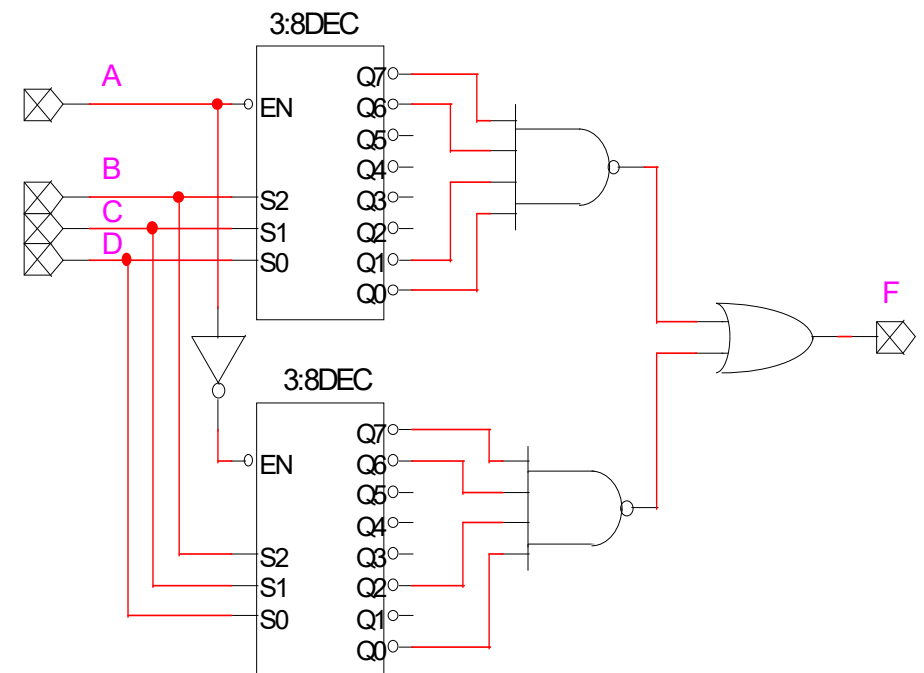
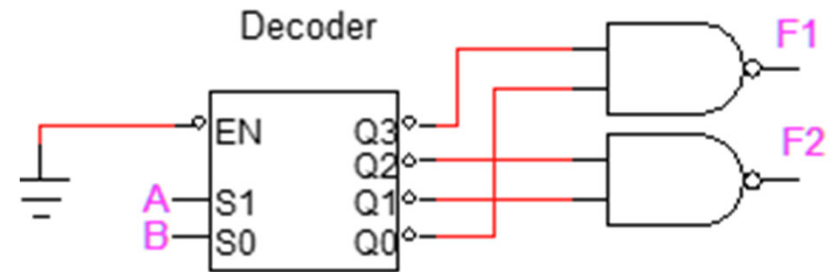
Each output equation is an “enabled” minterm

$$Y_i = Enb \cdot m_i(A, B, C) \quad i = 0, \dots, 2^n - 1$$

- A $n:2^n$ binary decoder is an implicit minterm generator for any n variables Boolean function

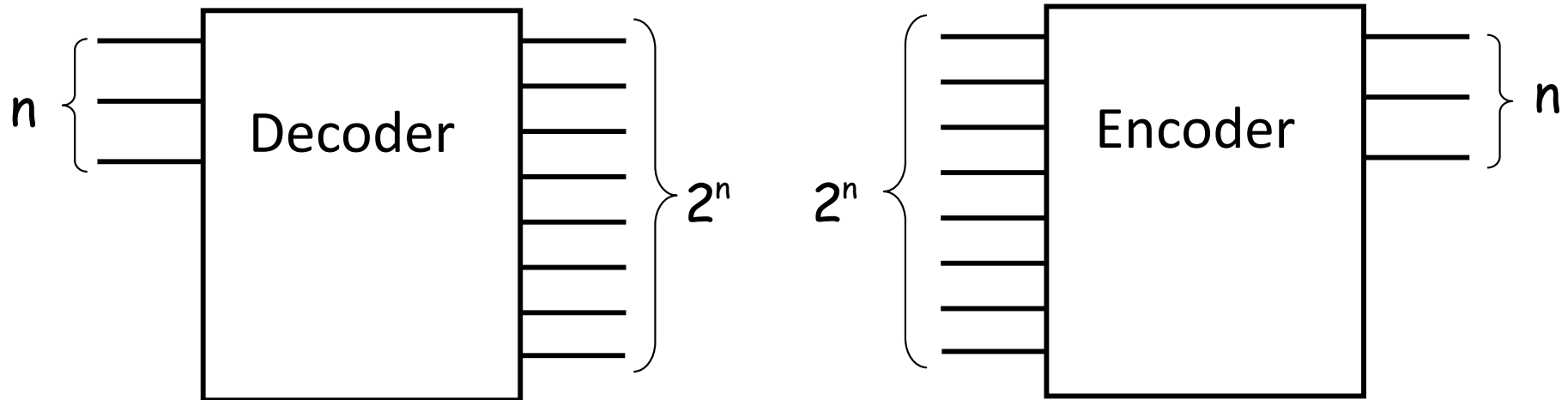
Canonical Implementations

- Write the SOP form for F_1 and F_2
- Obtain a minimal SOP form $F(A,B,C,D)$



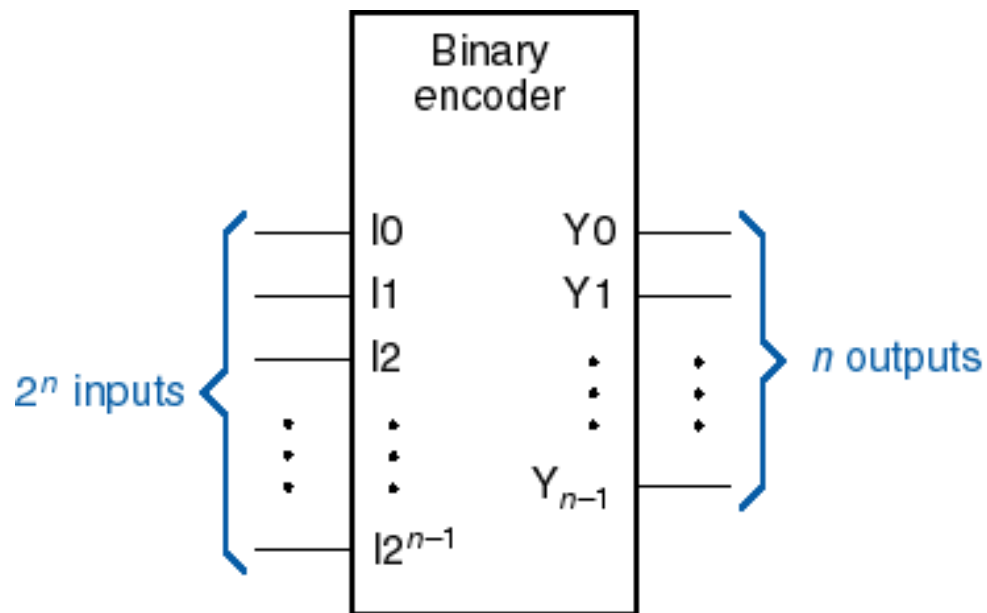
Encoders

- Functional inverse of the decoder

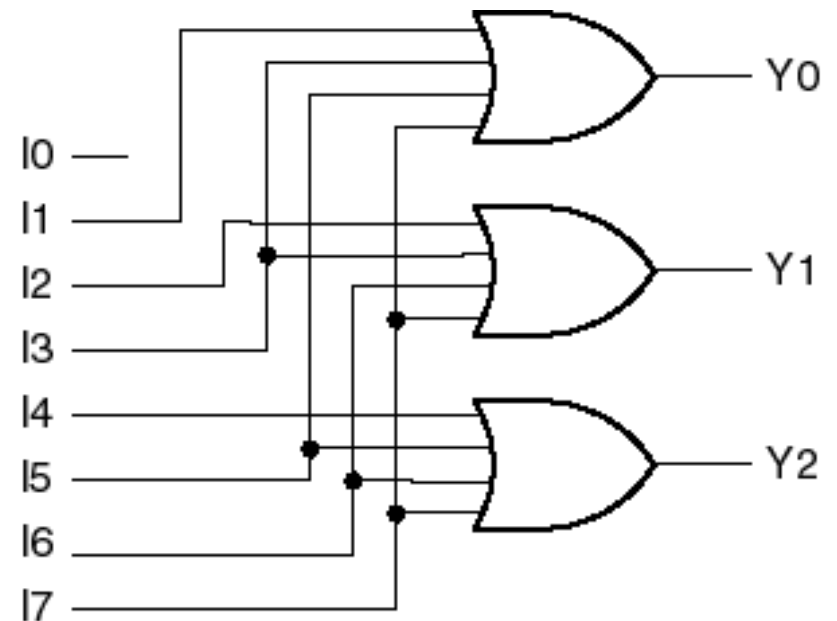


“Naïve” Binary encoders

- Why naïve?



Write the truth table



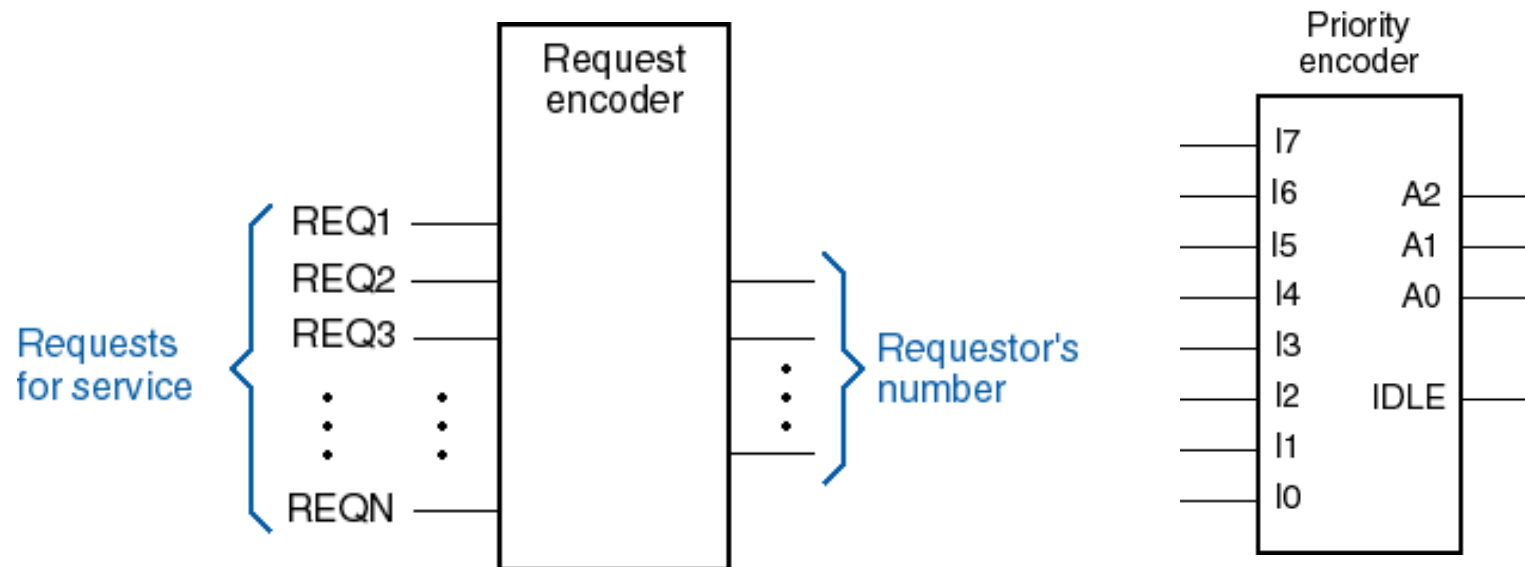
$$Y_0 = I_1 + I_3 + I_5 + I_7$$

$$Y_1 = I_2 + I_3 + I_6 + I_7$$

$$Y_2 = I_4 + I_5 + I_6 + I_7$$

The Priority issue

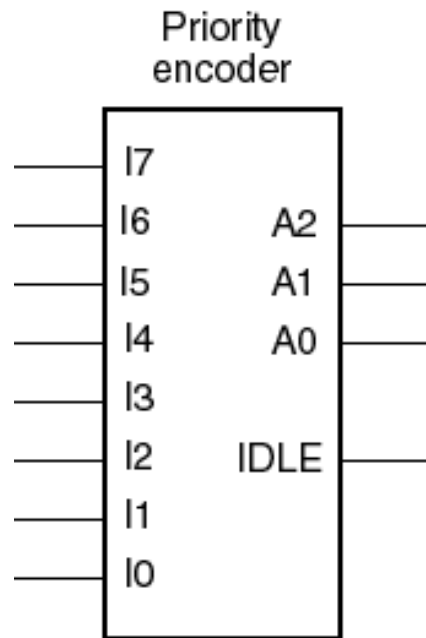
- In the previous “naïve” encoder what happens when I_3 and I_5 are asserted?



- Conflicts are resolved using a priority strategy

The Priority Encoder

- Let's define the following set of internal signals $H_n : H_0, H_1, \dots, H_7$



$$H_7 = I_7$$

$$H_6 = I_6 \cdot \bar{I}_7$$

$$H_5 = I_5 \cdot \bar{I}_6 \cdot \bar{I}_7$$

⋮

$$H_0 = I_0 \cdot \bar{I}_1 \cdot \bar{I}_2 \cdot \bar{I}_3 \cdot \bar{I}_4 \cdot \bar{I}_5 \cdot \bar{I}_6 \cdot \bar{I}_7$$

$$IDLE = \sum_{n=0}^{2^n-1} I_n = \prod_{n=0}^{2^n-1} \bar{I}_n$$

- Verify that priority is achieved when

$$A_0 = H_1 + H_3 + H_5 + H_7$$

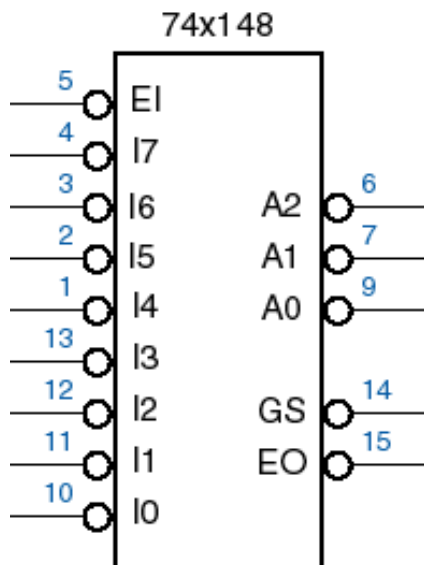
$$A_1 = H_2 + H_3 + H_6 + H_7$$

$$A_2 = H_4 + H_5 + H_6 + H_7$$

- What's the role of the IDLE output?

A commercial Model

- Active-low I/O
- Enable Input
- “Got Something Output”
- Enable Output

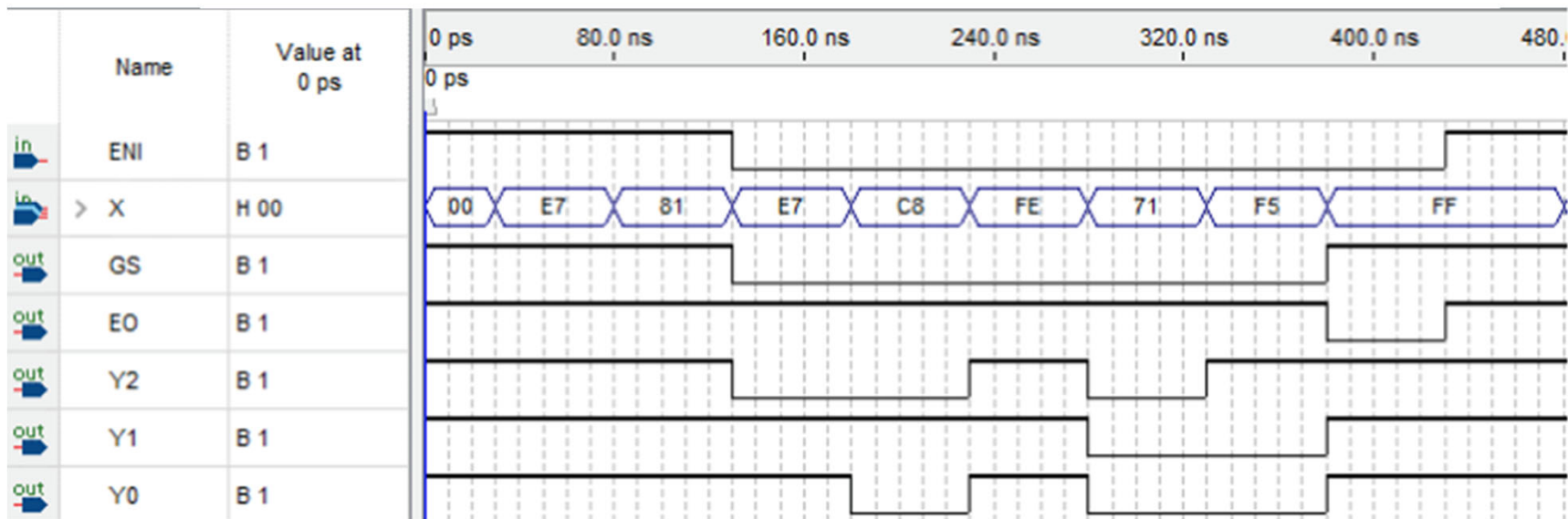
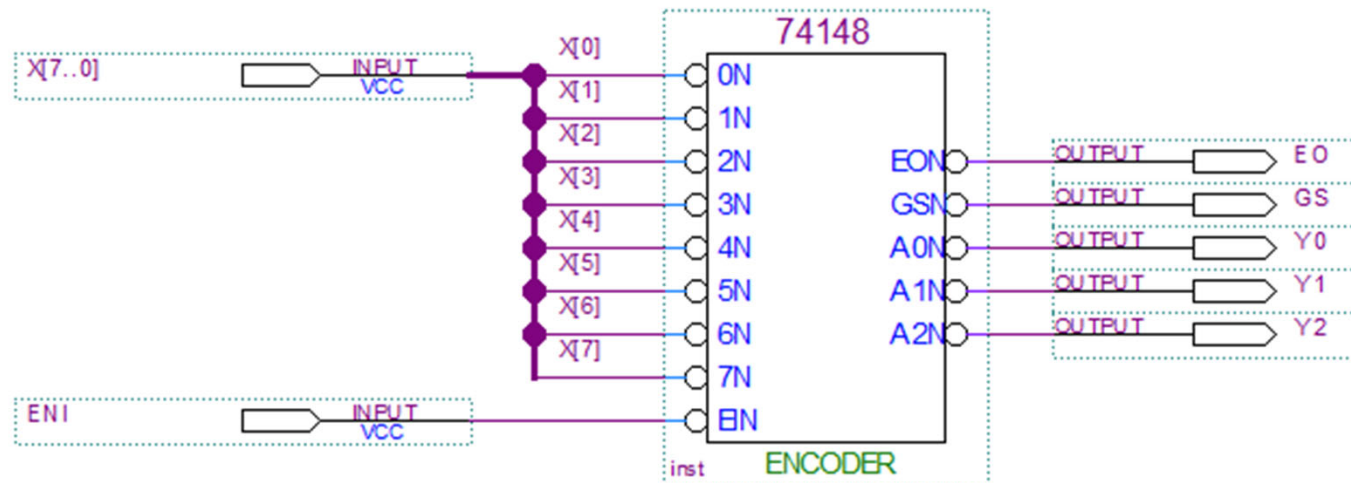


Inputs										Outputs				
EI_L	I0_L	I1_L	I2_L	I3_L	I4_L	I5_L	I6_L	I7_L		A2_L	A1_L	A0_L	GS_L	EO_L
1	x	x	x	x	x	x	x	x		1	1	1	1	1
0	x	x	x	x	x	x	x	0		0	0	0	0	1
0	x	x	x	x	x	x	0	1		0	0	1	0	1
0	x	x	x	x	x	0	1	1		0	1	0	0	1
0	x	x	x	x	0	1	1	1		0	1	1	0	1
0	x	x	x	0	1	1	1	1		1	0	0	0	1
0	x	x	0	1	1	1	1	1		1	0	1	0	1
0	x	0	1	1	1	1	1	1		1	1	0	0	1
0	0	1	1	1	1	1	1	1		1	1	1	0	1
0	1	1	1	1	1	1	1	1		1	1	1	1	0



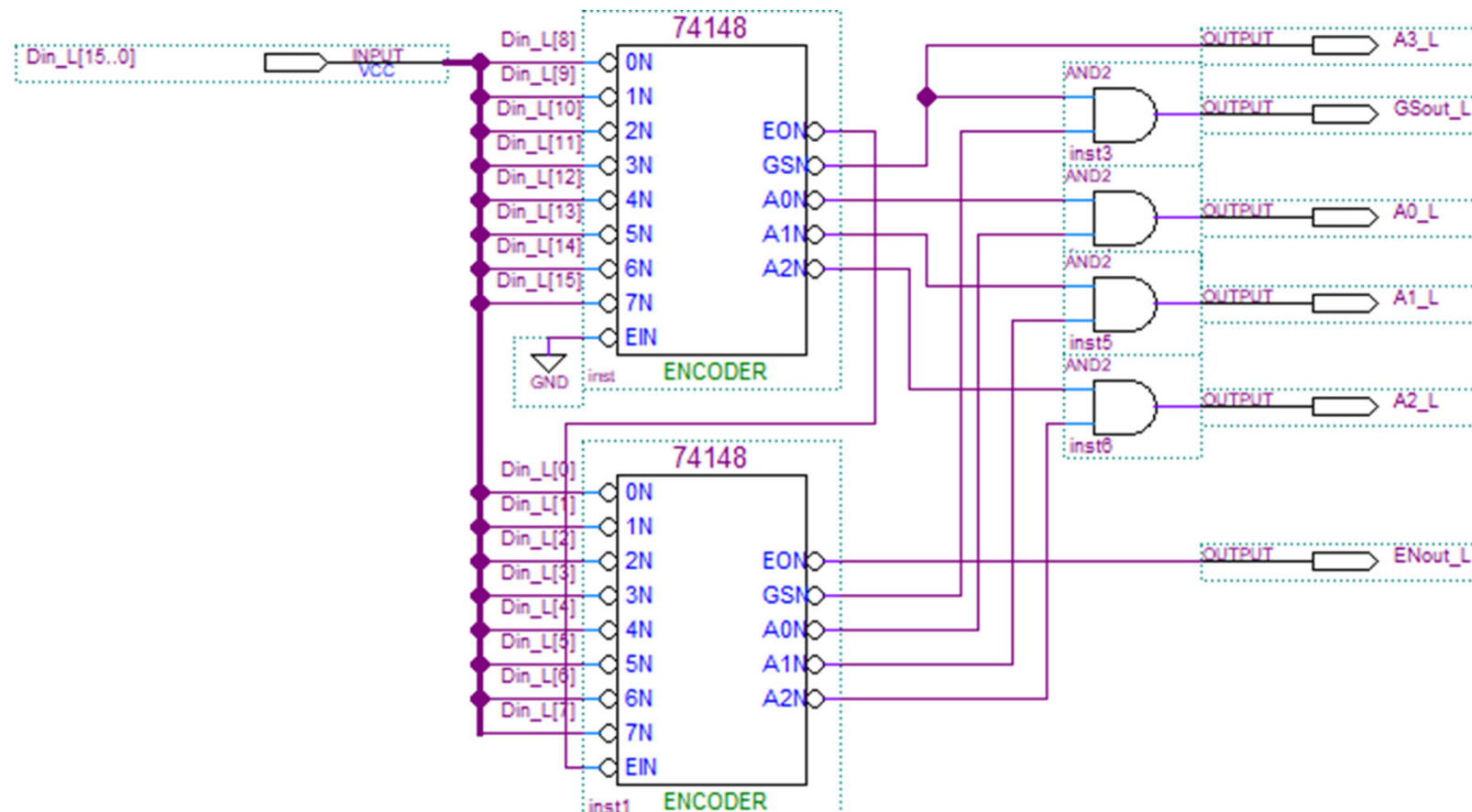
Exercise

- Explain the timing diagram



Scaling-up: PE16to4

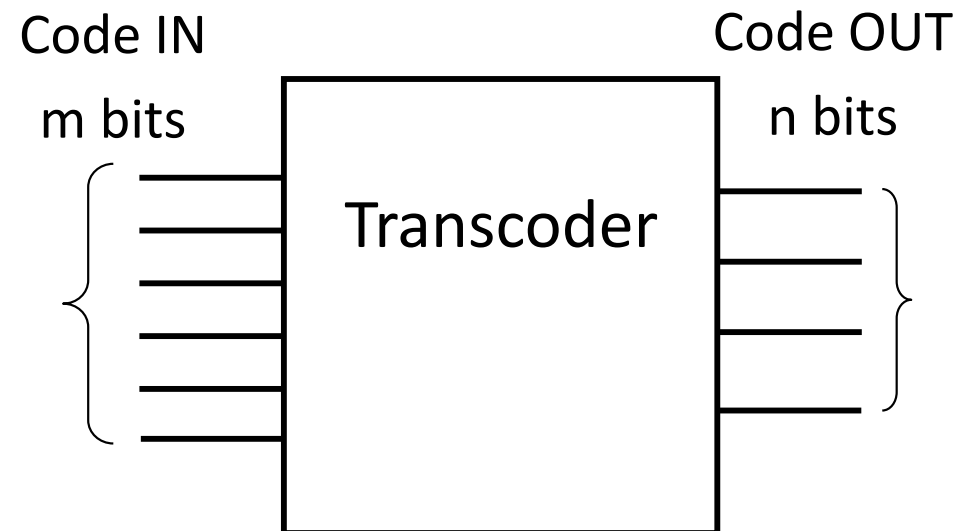
- Encoder cascading
- Note the priority enabling



Generic Decoding

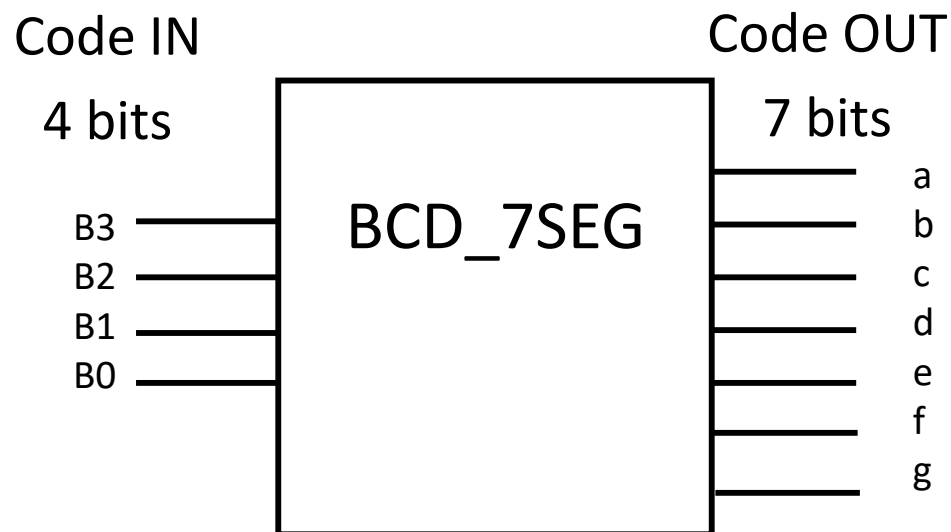
- “Transcoding”

Code IN	Code OUT
BCD ₈₄₂₁	7-Segs
BCD ₈₄₂₁	Gray
Others ...	

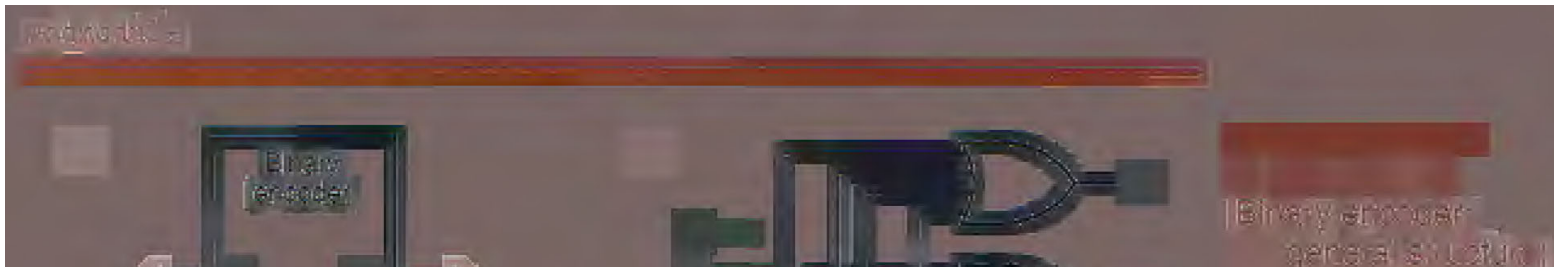


Example

- BCD₈₄₂₁ to 7-Segment decoder



- Write the truth table
- Obtain a minimal SOP for the segment “a”



Final Remarks

- Always recall
 - The block symbol
 - The types of inputs and outputs
 - Data
 - Control
 - The truth table
 - The output equations
- Design with encapsulated logic requires mastering all the functional details of each block