Faculty of Engineering

CSE115: Digital Design

Lecture 15: Decoders

Suggested Reading

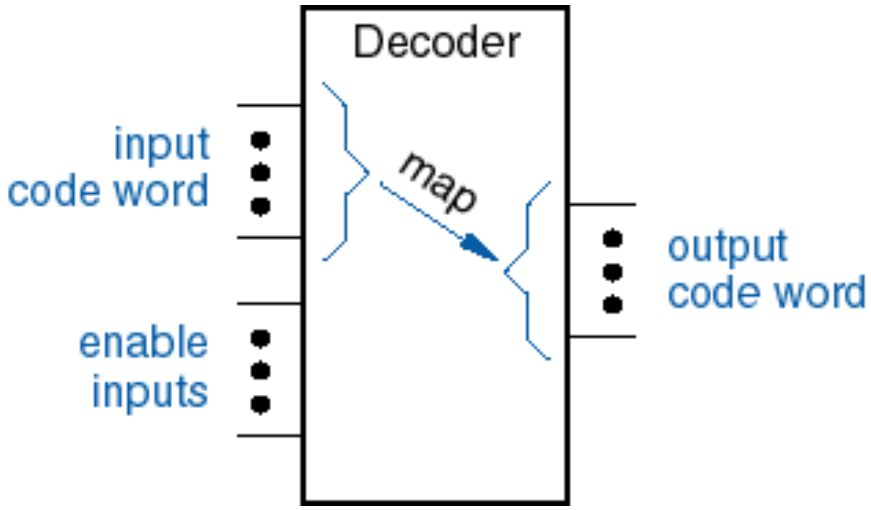
- Sections 5.4

Decoders

Converts input code words into output code words.

One-to-One mapping: Each input code produces **only one** output code.

Binary Code
Gray Code
BCD Code Any Code…

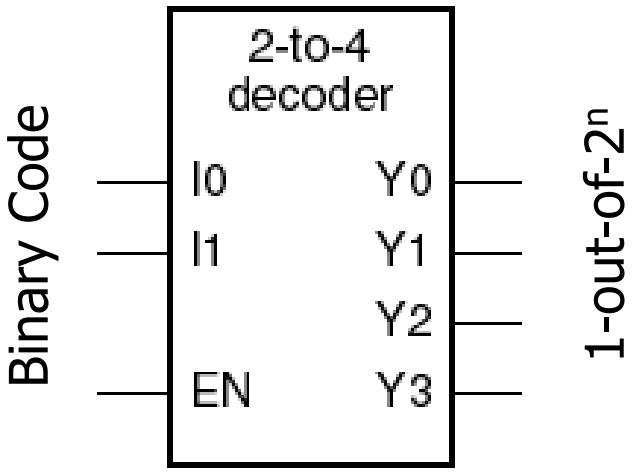


Typically ***n*** **inputs**, **2^n outputs**: 2-to-4, 3-to-8, 4-to-16, etc.

Binary 2-to-4 Decoder

n-to-2^n decoder: n inputs and 2^n outputs.

Example: 2-to-4 decoder

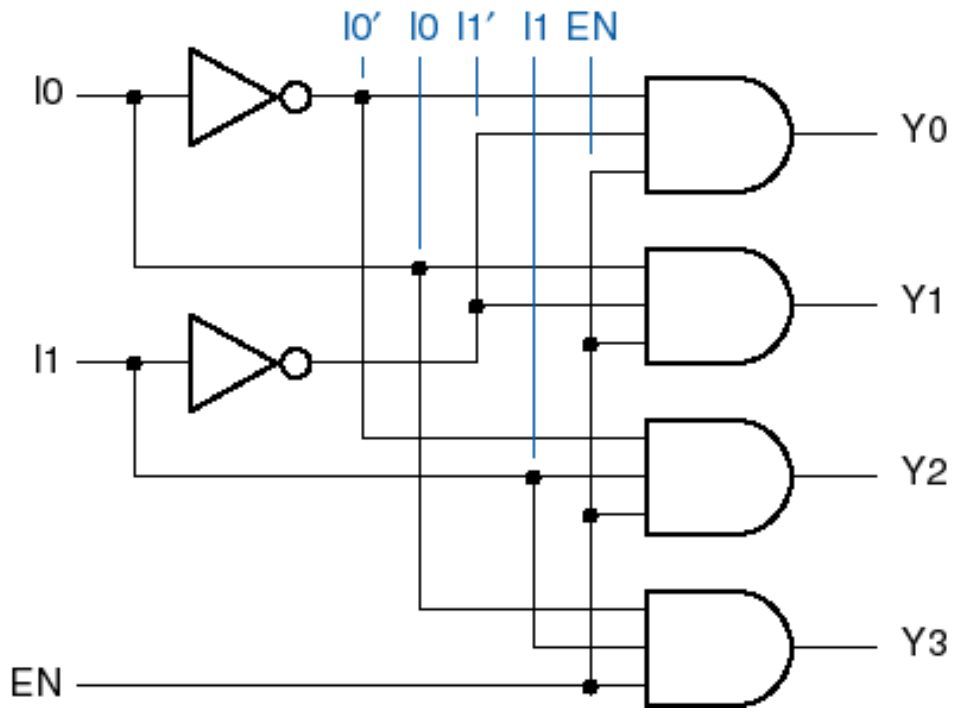


Inputs			Outputs			
EN	I1	I0	Y3	Y2	Y1	Y0
0	X	X	0	0	0	0
1	0	0	0	0	0	**1**
1	0	1	0	0	**1**	0
1	1	0	0	**1**	0	0
1	1	1	**1**	0	0	0

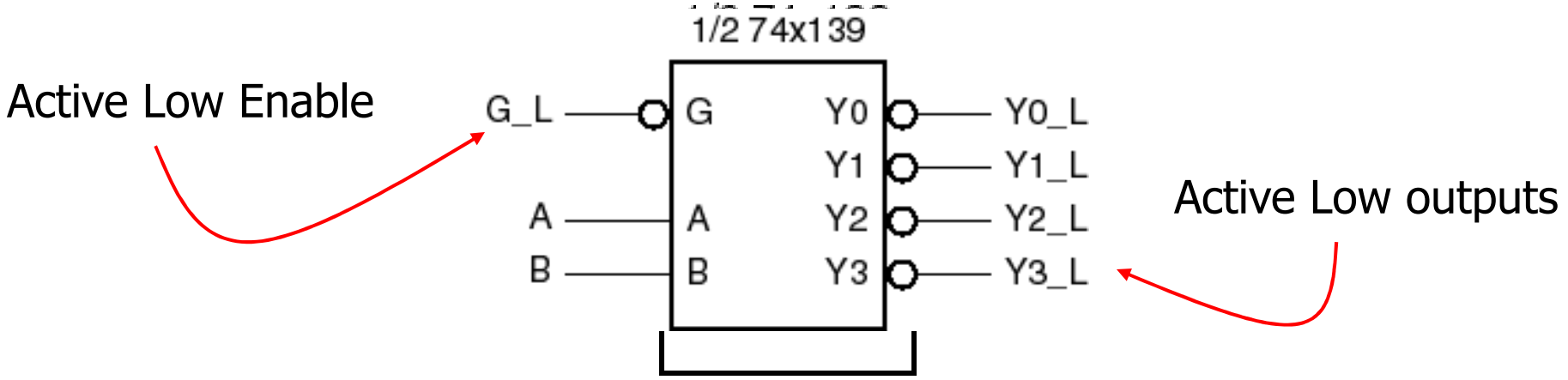
X: don't care

One output is asserted for each input code.

2-to-4- Decoder Logic Diagram

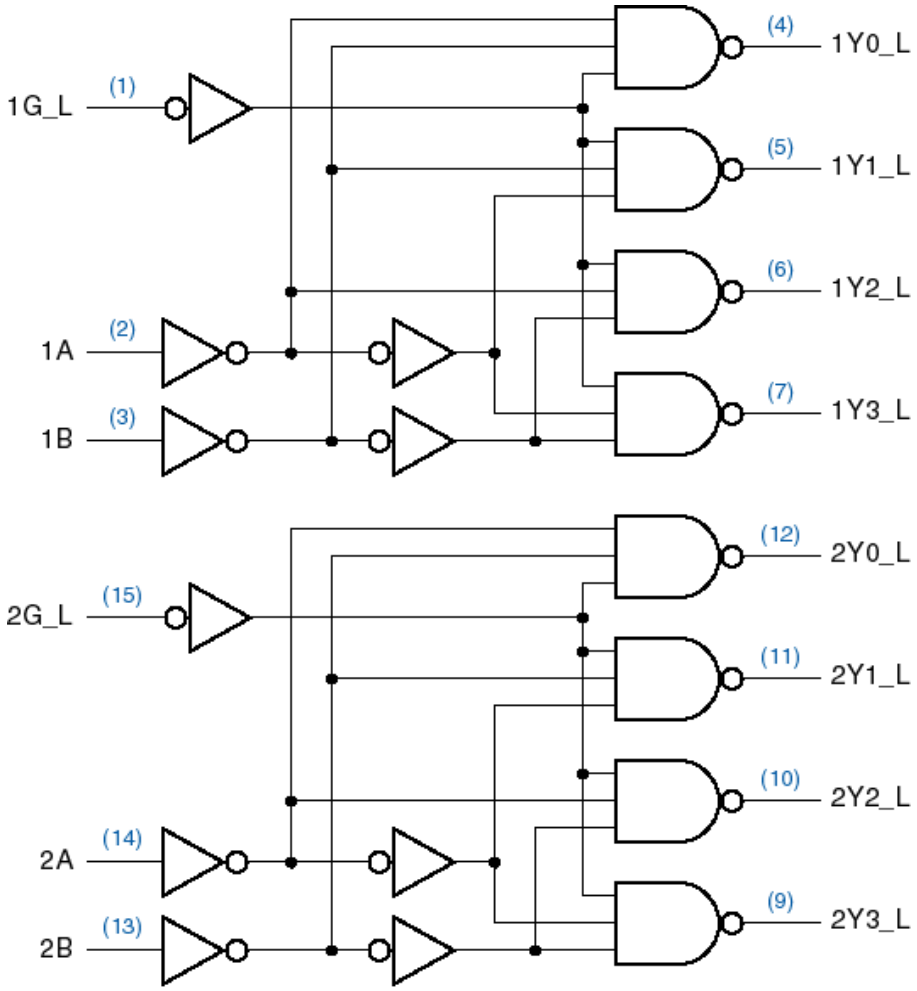


MSI Decoder



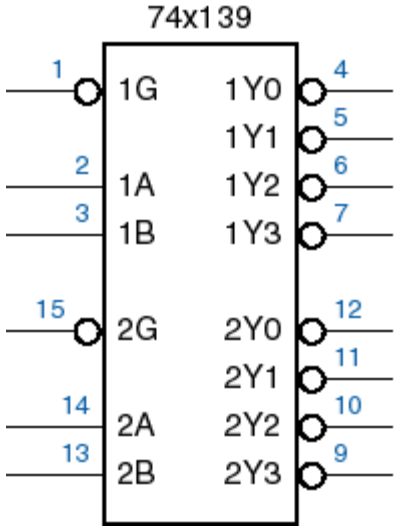
Inputs			Outputs			
/G	**B**	**A**	**/Y3**	**/Y2**	**/Y1**	**/Y0**
1	X	X	1	1	1	1
0	0	0	1	1	1	**0**
0	0	1	1	1	**0**	1
0	1	0	1	**0**	1	1
0	1	1	**0**	1	1	1

Complete 74x139 Decoder

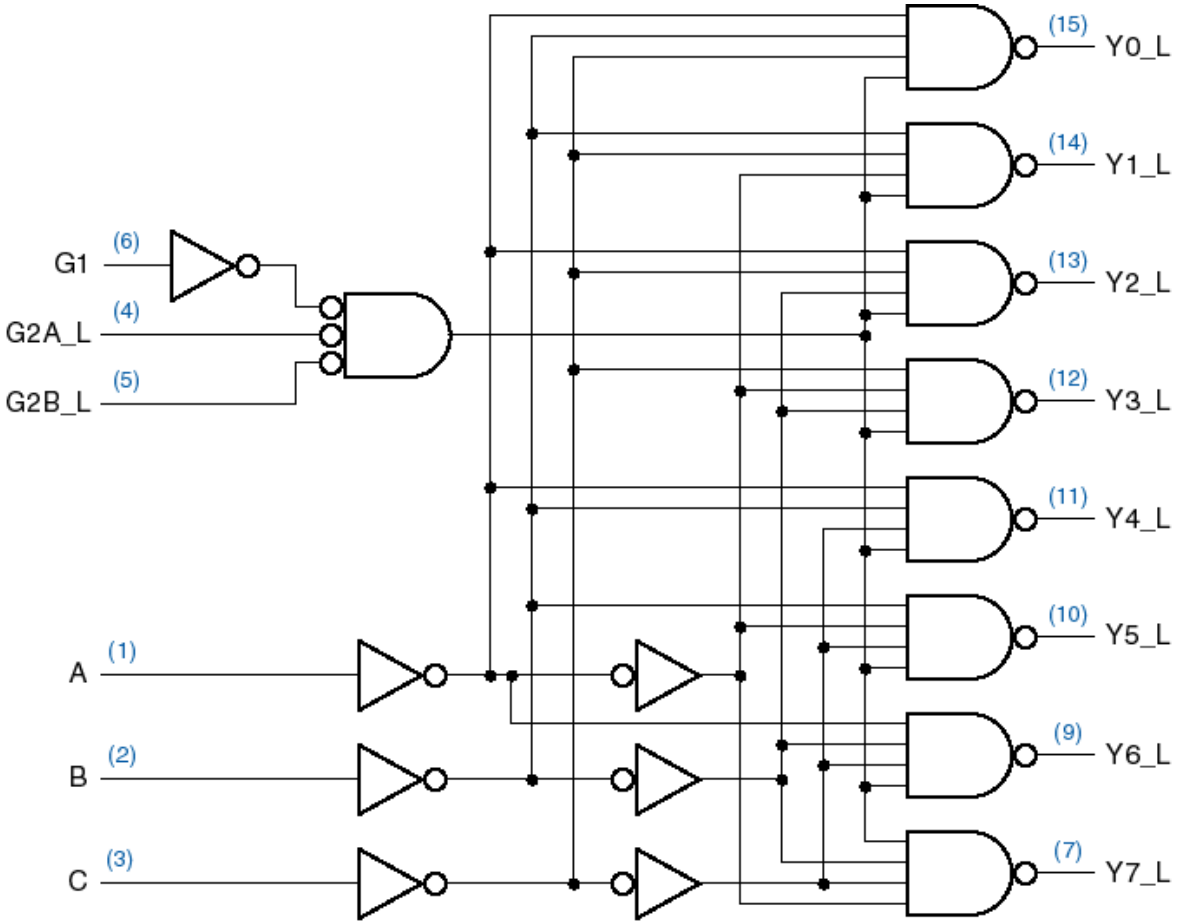


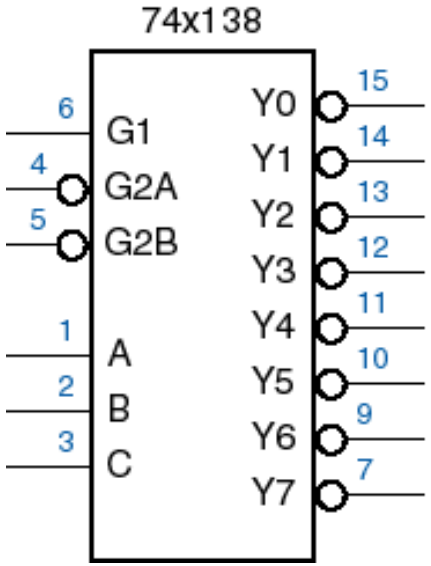
Input **buffering** → less load

NAND gates → faster



74x138 3-to-8 Decoder

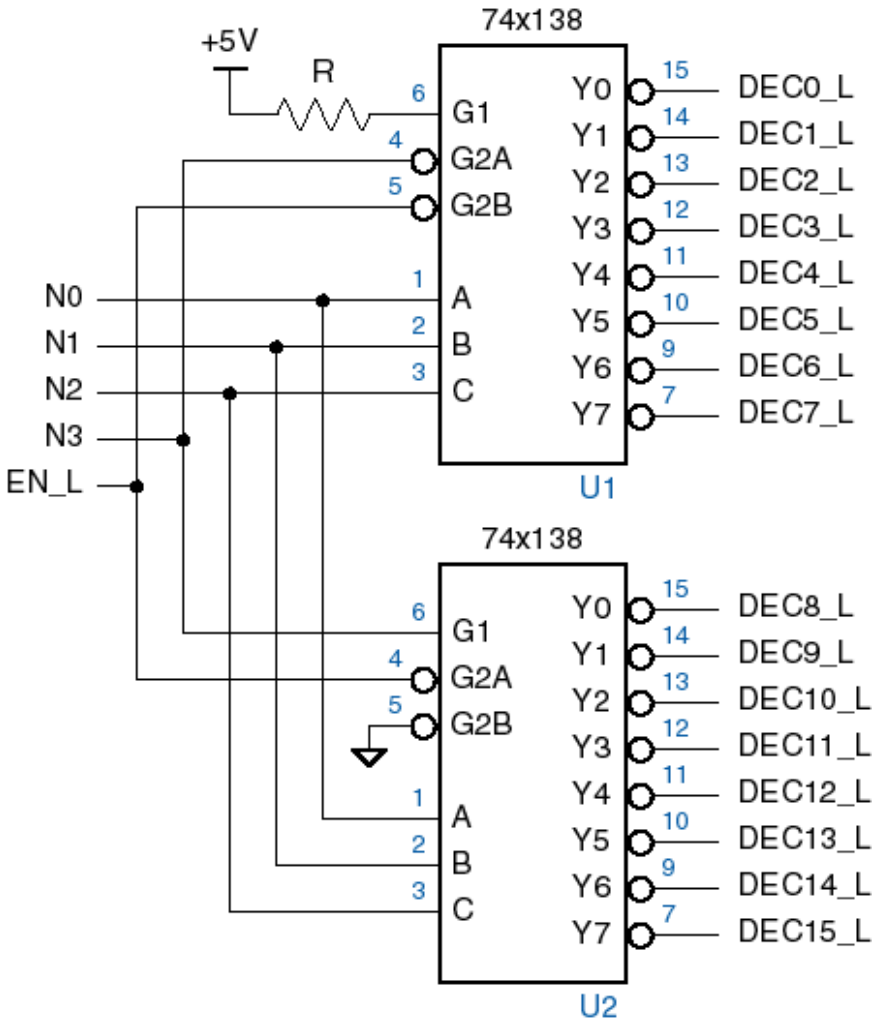




74x138 3-to-8-Truth Table

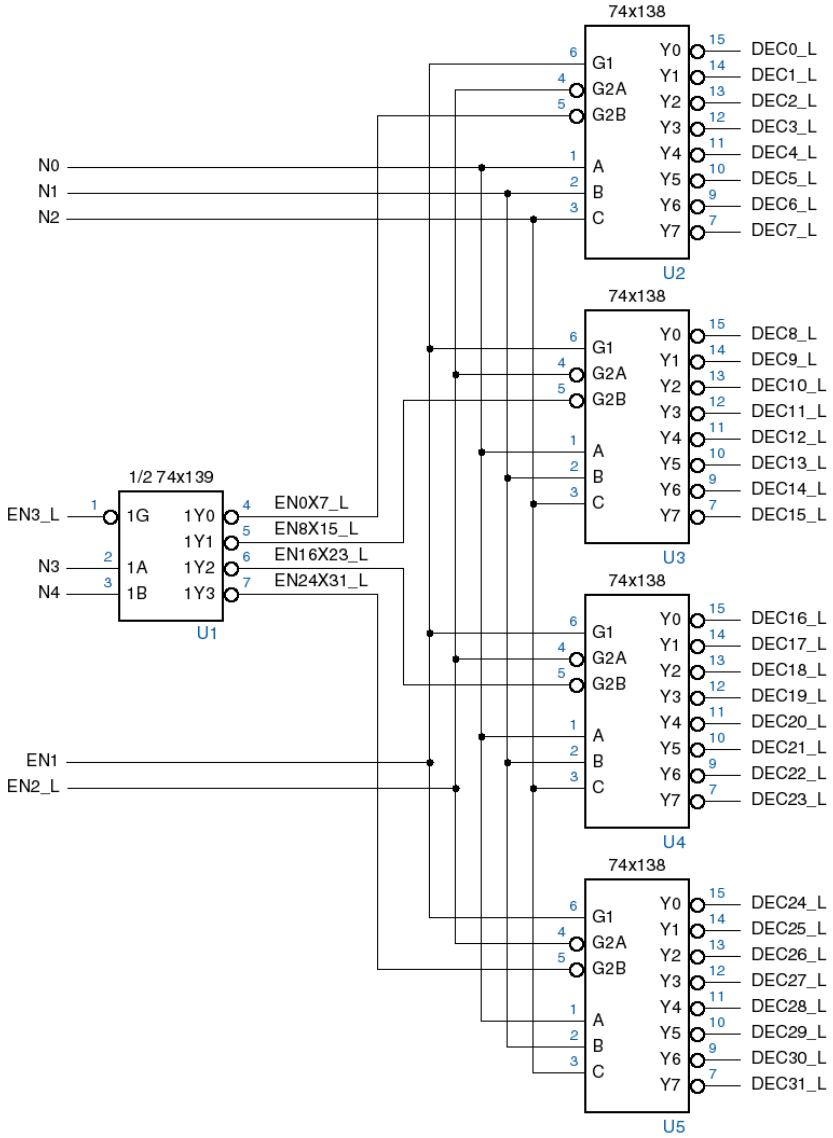
Inputs						Outputs							
/G1	/G2A	/G2B	C	B	A	/Y7	/Y6	/Y5	/Y4	/Y3	/Y2	/Y1	/Y0
0	X	X	X	X	X	1	1	1	1	1	1	1	1
X	1	X	X	X	X	1	1	1	1	1	1	1	1
X	X	1	X	X	X	1	1	1	1	1	1	1	1
1	0	0	**0**	**0**	**0**	1	1	1	1	1	1	1	**0**
1	0	0	**0**	**0**	**1**	1	1	1	1	1	1	**0**	1
1	0	0	**0**	**1**	**0**	1	1	1	1	1	**0**	1	1
1	0	0	**0**	**1**	**1**	1	1	1	1	**0**	1	1	1
1	0	0	**1**	**0**	**0**	1	1	1	**0**	1	1	1	1
1	0	0	**1**	**0**	**1**	1	1	**0**	1	1	1	1	1
1	0	0	**1**	**1**	**0**	1	**0**	1	1	1	1	1	1
1	0	0	**1**	**1**	**1**	**0**	1	1	1	1	1	1	1

Decoder Cascading: 4-to-16 Decoder



Two 3-to-8

More Cascading: 5-to-32 Decoder



Four 3-to-8

Decoder Applications

Microprocessor **memory** systems

- selecting different banks of **memory**

Microprocessor **input/output** systems

- selecting different devices

Microprocessor **instruction decoding**

- enabling different functional units

Memory chips

- enabling different rows of memory depending on address

Lots of other applications

Implementing the Canonical Sum

The binary decoder generates all minterms of n-variable logic function.

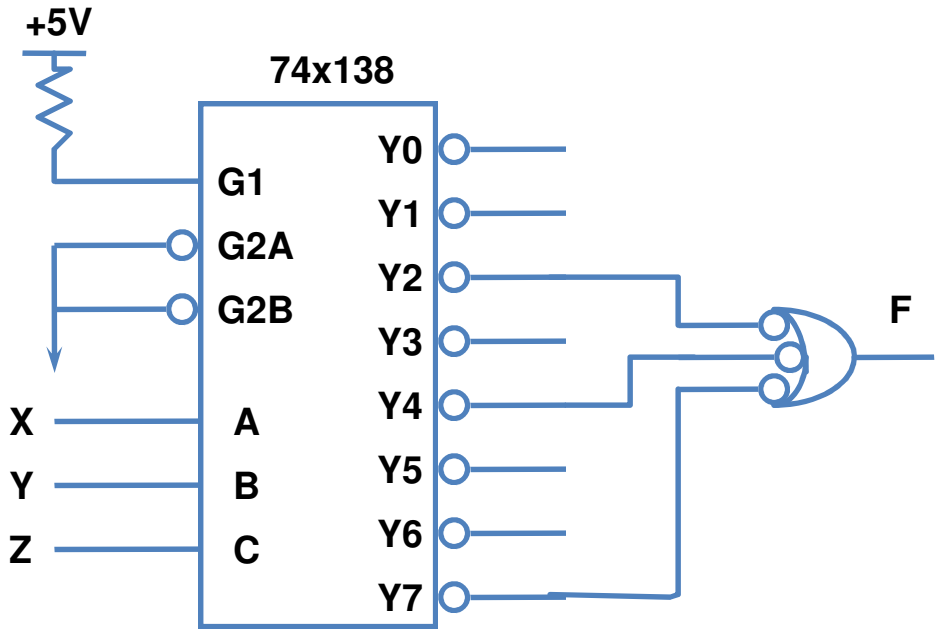
The canonical sum of a logic functions is obtained by adding all minterms of that function:

- Match the order of input bits

- Activate Enable inputs

Example:

$$F = \Sigma_{X,Y,Z}\,(2,4,7)$$



Logic design using Decoders

Advantages:

- Flexibility
- Multiple-output Logic functions

Disadvantages:

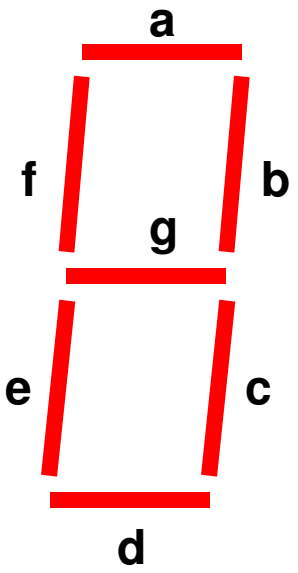
- Complexity: for large number of inputs

A practical alternative: PLD’s

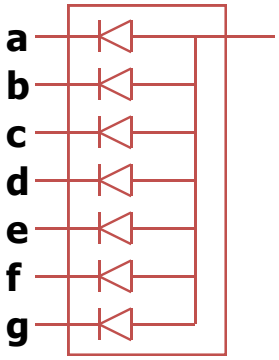
Seven-Segment Displays

Displays decimal numbers and some characters

LED (Light Emitting Diode) or **LCD** (Liquid Crystal Display)

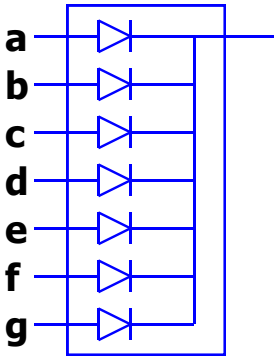


CommonAnode(CA)



requires Active Low inputs

CommonCathode(CC)



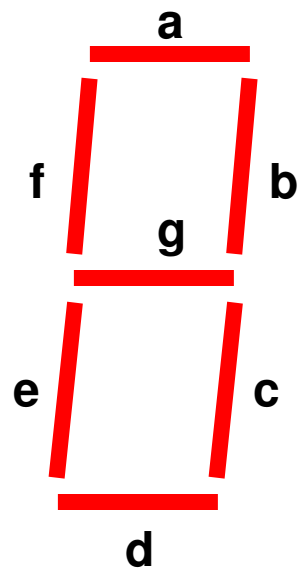
requires Active High inputs

Seven-Segment Decoders/Drivers

BCD Code

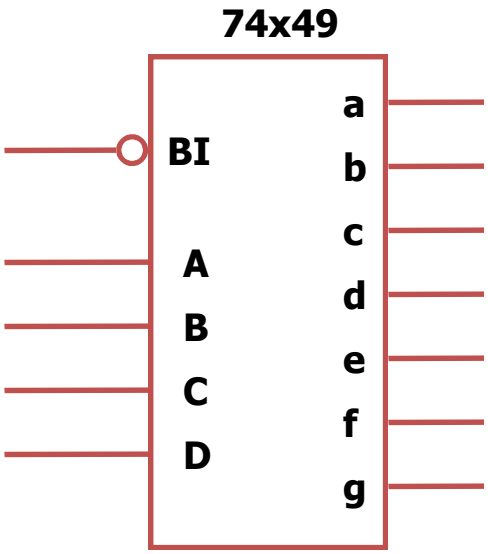
Seven-Segment Code

Input				Output						
D	C	B	A	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	0	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1

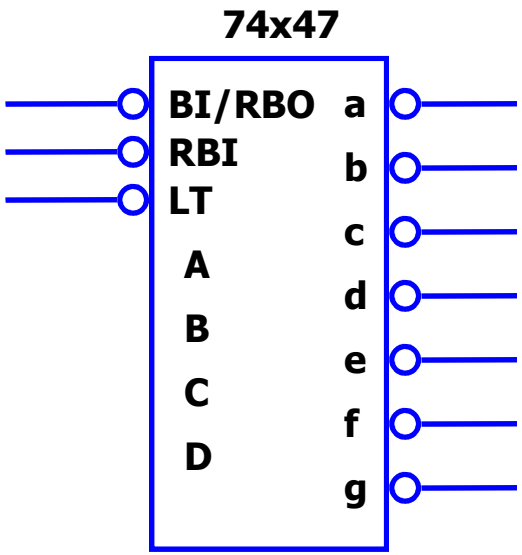


MSI Seven-Segment Decoders

CC Driver : 74x49



CA Driver : 74x47



74x49 Driver

Driving CC Seven-Segment Display