

University of Illinois at Urbana-Champaign
Dept. of Electrical and Computer Engineering

ECE 120: Introduction to Computing

Multiplexers (MUXes)

Task: Checking for a Lower-Case Letter

What if we also need logic to check whether an **ASCII** character is a lower-case letter.

In **ASCII**, 'a' is **1100001** (0x61),
and 'z' is **1111010** (0x7A).

Recall that 'A' is **1000001** (0x41),
and 'Z' is **1011010** (0x5A).

**Can we reuse our solutions
for upper-case letters?**

Of course we can!

Change C_5' to C_5 to Obtain $L(C)$ from $U(C)$

Let's again say that the **ASCII** character is in $C = C_6C_5C_4C_3C_2C_1C_0$.

By breaking up the truth table, we obtained

$$U(C) = C_6C_5'C_4' (C_3 + C_2 + C_1 + C_0) + C_6C_5'C_4 (C_3' + C_2')(C_3' + C_1' + C_0')$$

But lower-case characters are only different from upper-case in C_5 , which is 1 instead of 0.

$$L(C) = C_6C_5C_4' (C_3 + C_2 + C_1 + C_0) + C_6C_5C_4 (C_3' + C_2')(C_3' + C_1' + C_0')$$

Or just change the comparators' inputs.



Want Logic to Choose Between Two Signals

What if we want one design to check for either upper-case or lower-case letters?

In a few examples,

- we added a control signal S
- to select between functions.

Can we design logic

- that uses a control signal S to select
- between two arbitrary signals,
 D_1 (when $S = 1$) and D_0 (when $S = 0$)?

Truth Tables for a 2-to-1 Multiplexer

A full truth table for such logic appears to the right.

But we could shorten it as shown below...

S	D ₁	D ₀	Q
0	x	0	0
0	x	1	1
1	0	x	0
1	1	x	1

Unselected inputs do not matter (marked with “x”).

S	D ₁	D ₀	Q
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

We Normally Use the Most Compact Truth Table

In this case, we can even write outputs in terms of other inputs, as shown here.

S	D ₁	D ₀	Q
0	x	0	0
0	x	1	1
1	0	x	0
1	1	x	1

Unselected inputs do not matter (marked with “x”).

S	Q
0	D ₀
1	D ₁

Expression for a 2-to-1 Multiplexer

Let's solve with a K-map.

$$Q = S'D_0 + SD_1$$

		D_1D_0			
		00	01	11	10
S	0	0	1	1	0
	1	0	0	1	1

S	D_1	D_0	Q
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Expression for a 2-to-1 Multiplexer

But **Q** just selects **D₀** or **D₁** (as desired)!

$$Q = S'D_0 + SD_1$$

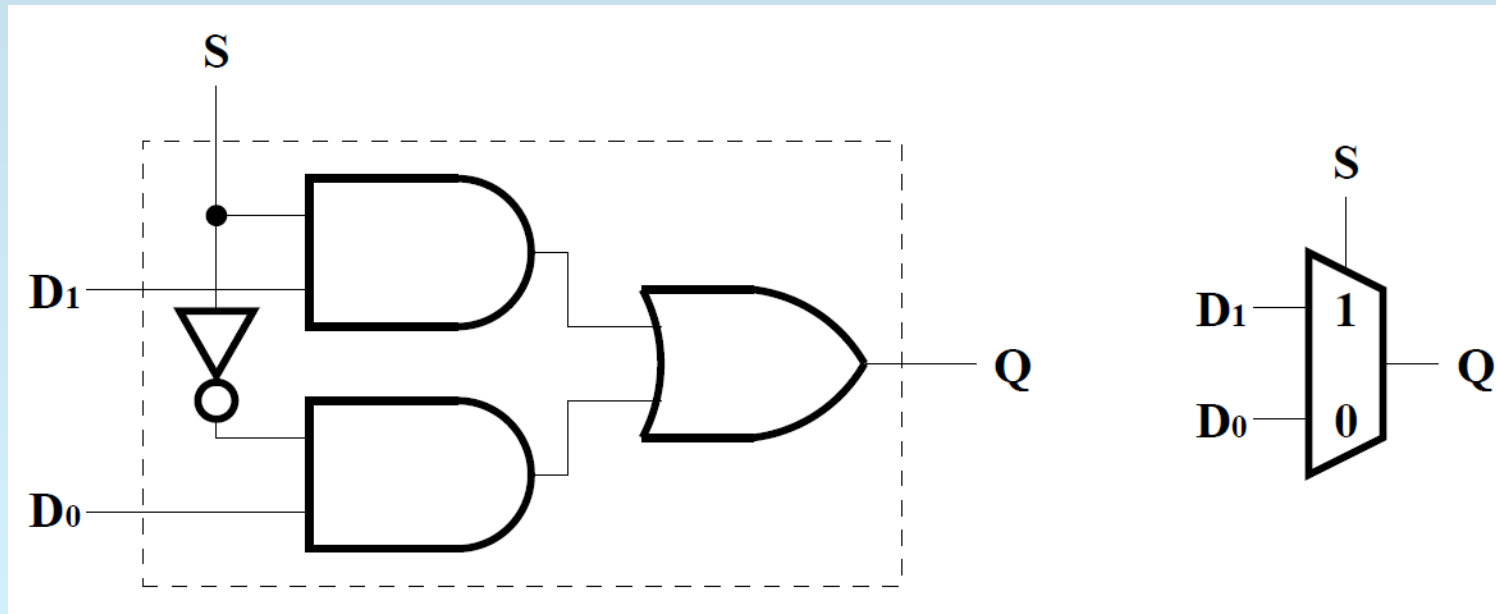
S	Q
0	D ₀
1	D ₁

Could we have just written this expression using the table to the right?

Q is **D₀** when **S = 0**, and **D₁** when **S = 1**...

Implementation and Symbolic Form of a 2-to-1 Mux

The circuit below shows a **2-to-1 mux (multiplexer)**, for which the symbolic form appears to the right.



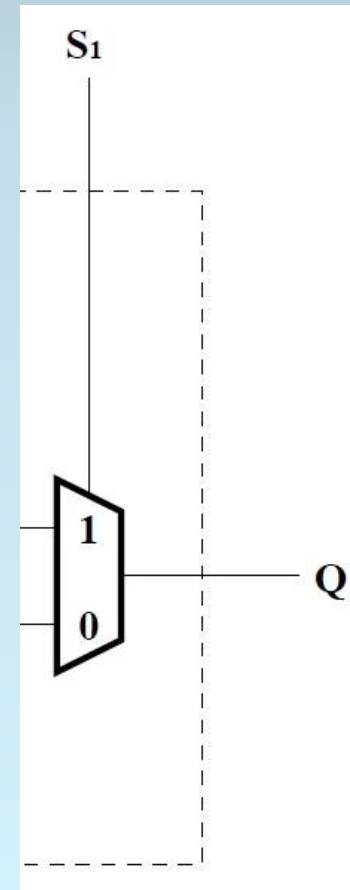
Selecting from More than Two Expressions

What if we want to select between four expressions, D_3 , D_2 , D_1 , and D_0 ?

One answer is to use muxes hierarchically:

- start by using one 2-to-1 mux (signal S_1)
- to decide between D_3 or D_2 and D_1 or D_0 .

D_3 or D_2
 D_1 or D_0

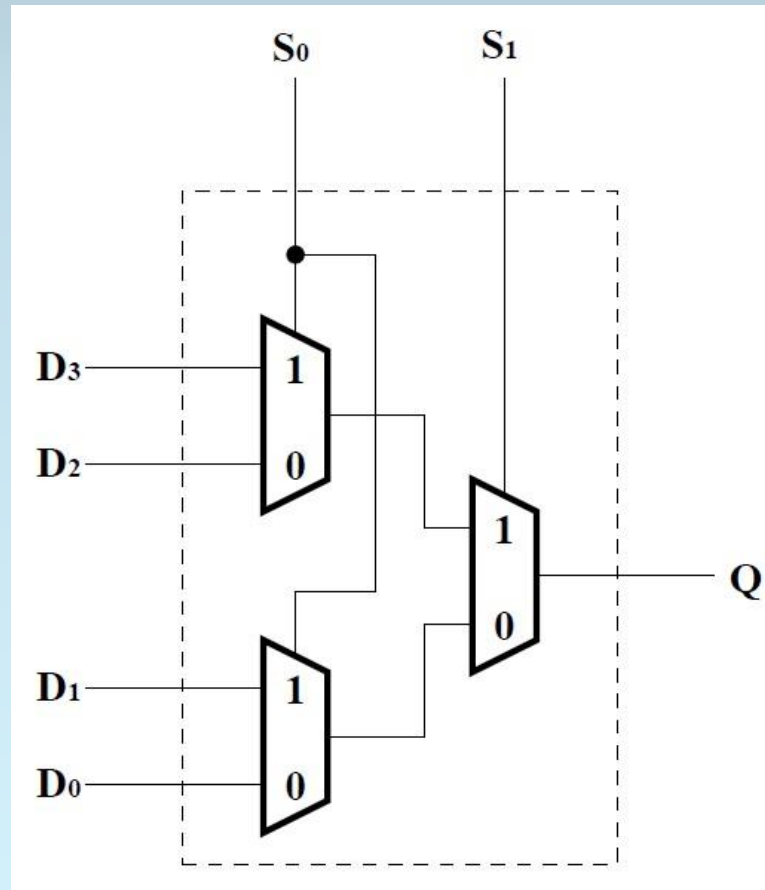


For the Second Level, Use More Muxes

But how do we deliver two expressions to each mux input?

Use more muxes (both controlled by S_0)!

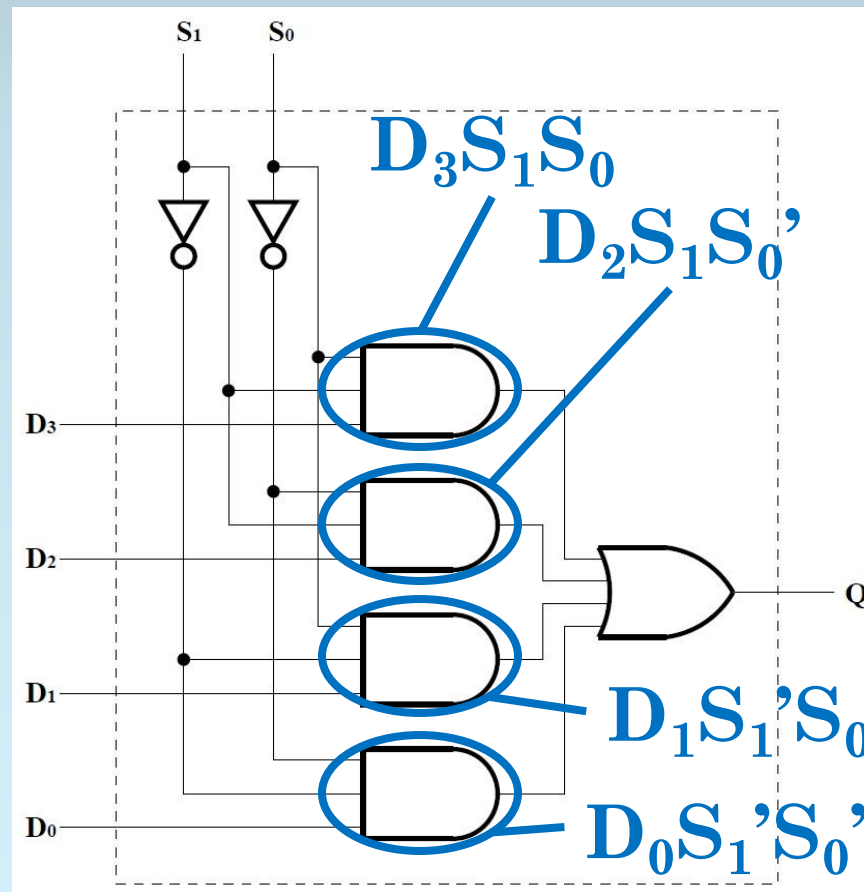
Notice that S_1S_0 then allows us to choose from four expressions.



AND Gates Represent Minterms ANDed with Data Inputs

For something as common as a mux, we typically build directly from gates.

Notice that **each AND gate produces a minterm of S_1, S_0 ANDed with the corresponding D_i .**

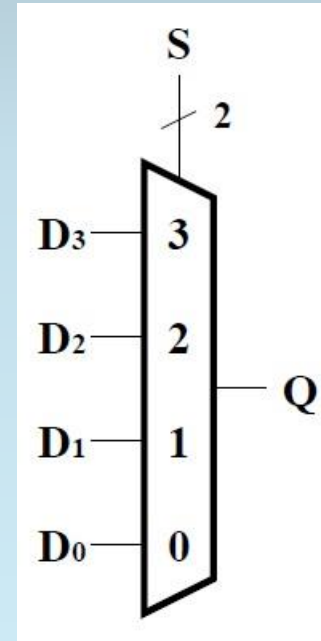


A 2^N -to-1 Mux Requires N Select Bits

The diagram to the right shows the **symbolic form of a 4-to-1 mux**.

We can, of course, further extend this idea to build 8-to-1 muxes, 16-to-1 muxes, and so forth.

When selecting amongst $P = 2^N$ inputs $D_{P-1} \dots D_0$ we need N bits of select input, $S_{N-1} \dots S_0$.



Can Use Sets of Muxes to Select Amongst Groups of Bits

We can also generalize the idea of multiplexers by

- using a common control signal
- to select between groups of inputs.

Generally,

- **an N-to-M multiplexer**
- **represents M separate (N/M)-to-1 muxes**
- each with $\log_2(N/M)$ select bit inputs
- (typically $N/M = 2^K$ for some integer **K**).

Example of a Set of Muxes with Common Select Input

For example, recall the design of the **N-bit** adder and subtractor.

We could have used a **2N-to-N** mux

- to choose between **B_i** and **B_i'** for the adder's **B** input
- based on a common (one-bit) control signal **S** .

(Previously, we used the nature of the mux' data inputs, **B_i** versus **B_i'** , to simplify each mux' logic to an XOR gate.)

Another Design Problem: Checking Four Types of ASCII

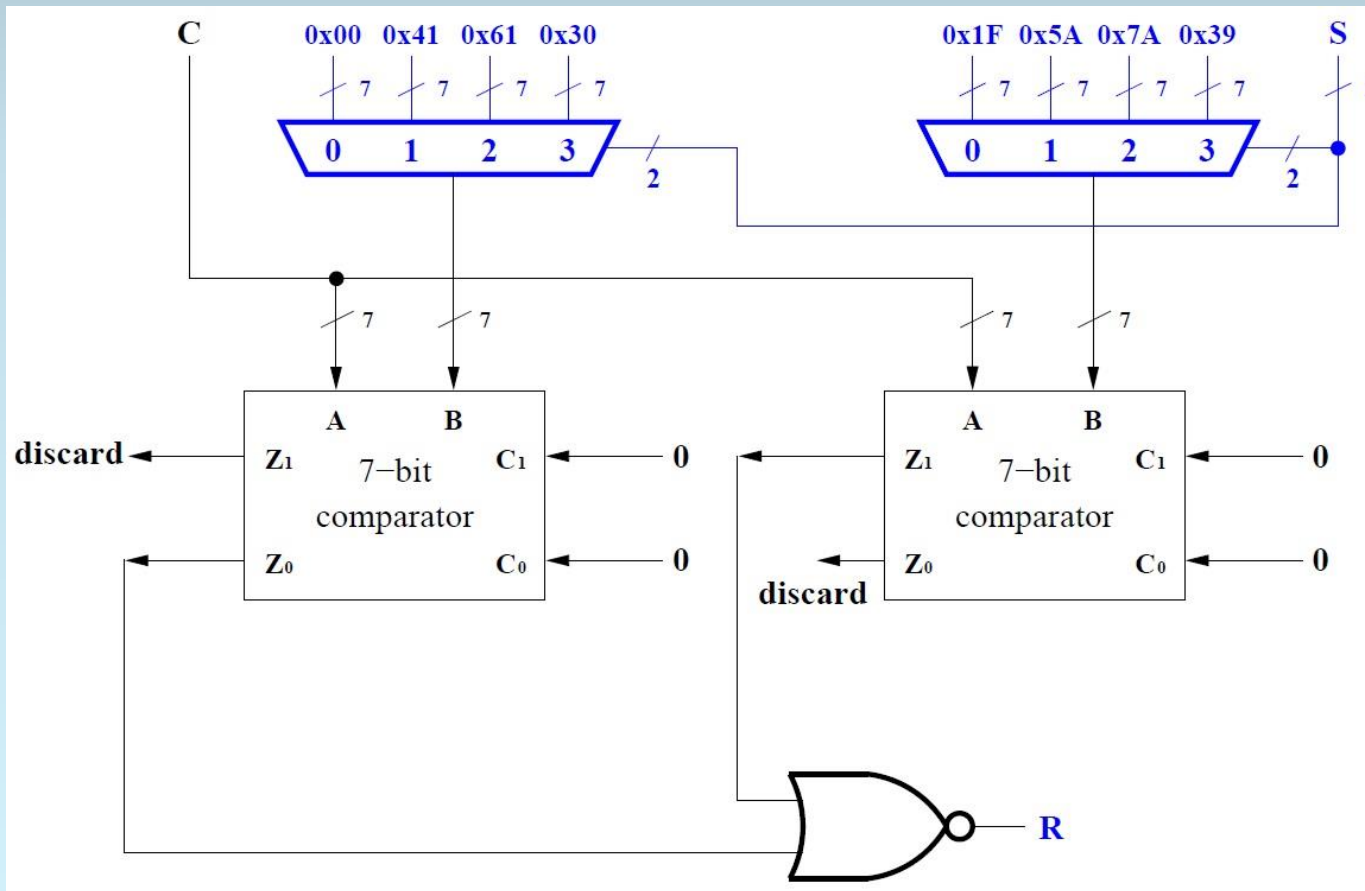
Now think again about our **ASCII** checker.

Say that we want four kinds of comparison:

- control characters (0x00 to 0x1F),
- lower-case letters (0x41 to 0x5A),
- upper-case letters (0x61 to 0x7A), and
- digits (0x30 to 0x39).

**How can we design logic to check
for any of the four types?**

The Answer? Use Muxes! Two 28-to-7 Muxes



University of Illinois at Urbana-Champaign
Dept. of Electrical and Computer Engineering

ECE 120: Introduction to Computing

Decoders

How Can We Decode a Representation?

What if we have

- designed a representation using **N** bits,
- and have an **N-bit** value in that representation,
- but need separate signals for each of the **2^N** possible bit patterns?

Applications include

- **naming sets of bits**, which we explore in 3-4 weeks, when we examine memory, and
- **generating arbitrary functions dynamically**, a technique used for many years in reconfigurable hardware devices.

A Decoder's Outputs Have Exactly One 1 Bit

In other words,

- **given a set of N bits,**
- we want to **generate a signal for each possible combination** of the N bits.

These **signals correspond to the minterms** on the N bits.

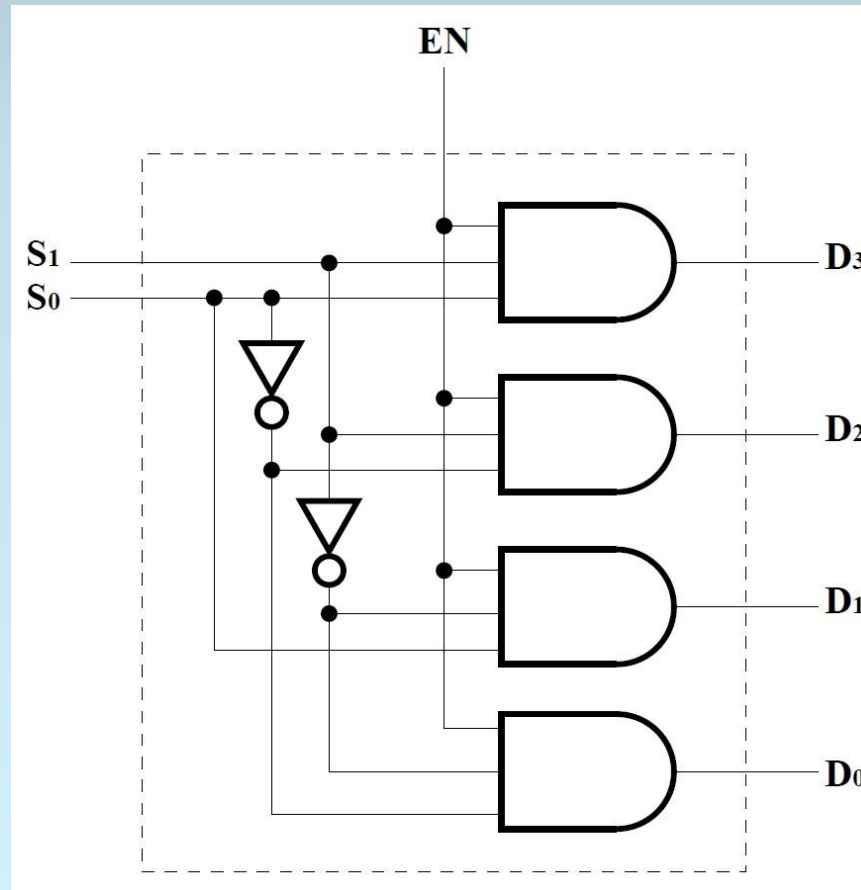
Exactly one of the resulting signals **has the value 1**.

A 2-to-4 Decoder with Enable Produces Zero or One 1 Bit

We call the logic to perform such a task a **decoder**.

An example of a **2-to-4 decoder** is shown to the right.

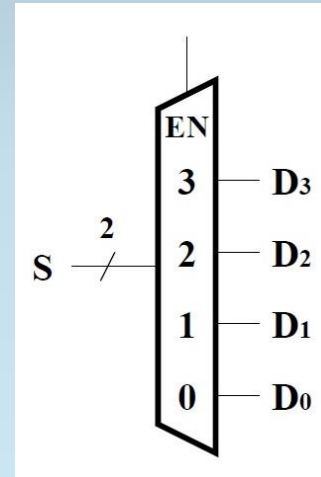
The decoder here has an enable signal. If **EN = 0**, all outputs are 0.



A Trapezoid that Widens Represents a Decoder

The symbolic form for a decoder is a trapezoid.

- The **N-bit** signal to be decoded enters from the smaller side.
- The **2^N** outputs exit from the larger side.

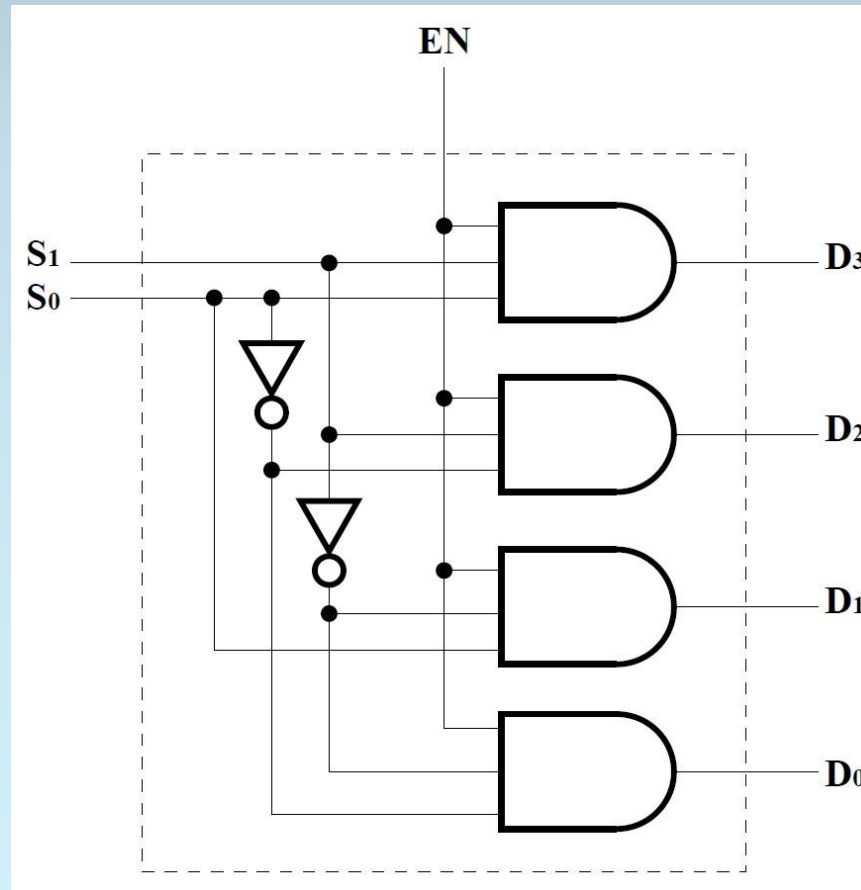


If the decoder has an enable signal, it is typically drawn on a small side of the trapezoid.

Decoders and Muxes Both Calculate All Minterms

Notice that the internal structure of a decoder looks similar to that of a mux.

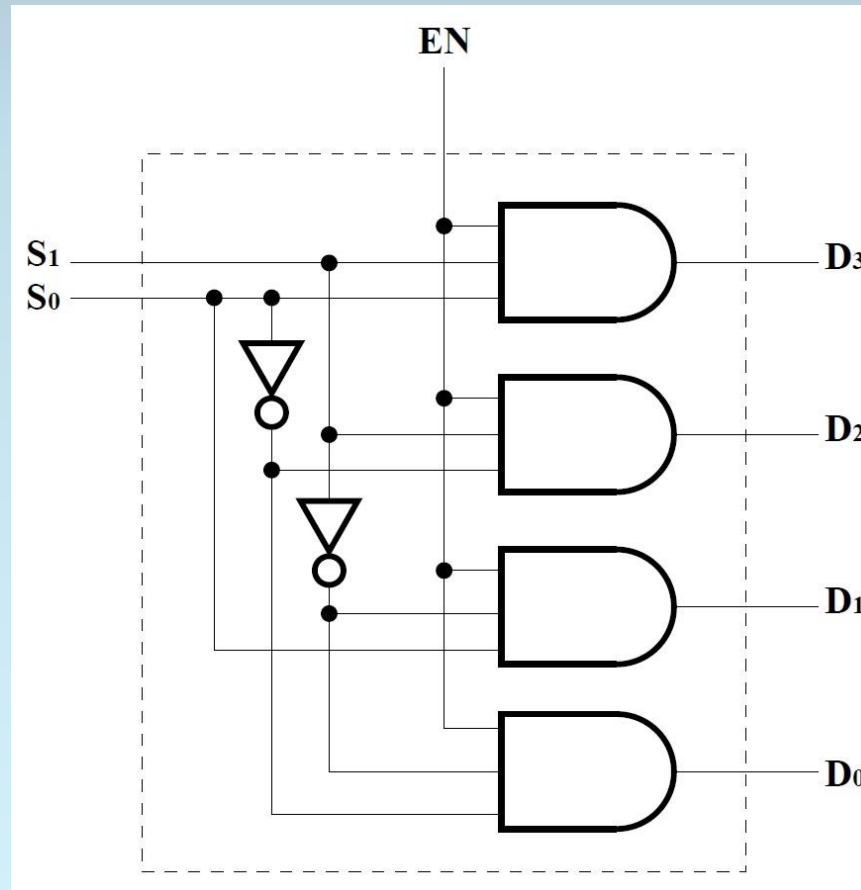
In a mux, minterms are combined with per-minterm inputs and ORed together in a single output.



Decoder Outputs Allow Separate Use of Each Minterm

The decoder allows the minterm signals to be used separately.

For example, each decoder output might control the mechanical release for a different product in a vending machine.



Early Reconfigurable Hardware Used Decoders

Or we can use a decoder

- to compose arbitrary functions on the inputs
- by ORing together the right set of minterms.

Doing so is equivalent to composing a function with a mux by connecting 0s and 1s to its inputs.

But a single decoder can be used construct many functions, while the mux allows only one function.

Such an approach allowed

- programmable logic arrays (PLAs)
- to dynamically produce arbitrary functions,
- thus providing reconfigurable hardware.