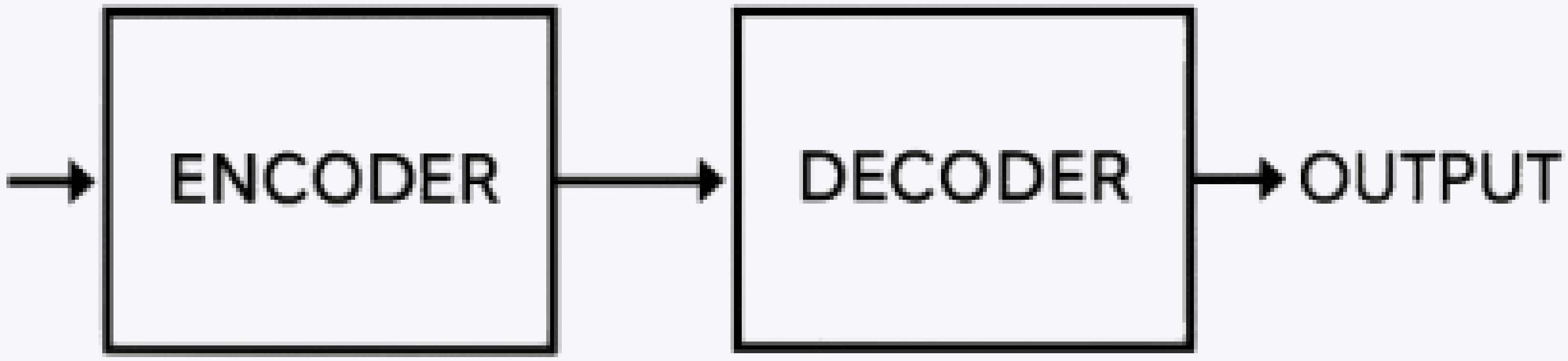
Encoders & Decoders

- What are encoders and decoders?
- Decoder logic, truth tables and gate-level implementations
- Encoder logic, truth tables and priority handling
- Area and delay analysis for each circuit
- VHDL implementation examples (decoder and encoder)
- System-level usage and design applications
- Summary table and comparisons
- Interactive quiz to reinforce concepts
- Homework assignment to practice

Encoders & Decoders: From Logic To VHDL

Welcome to this lecture on encoders and decoders. We'll go from what they are conceptually, all the way through logic design, performance considerations like area and delay, and their implementation in VHDL.

This is a key building block in digital systems, so pay attention—this logic shows up everywhere from memory access to keyboard scanning.



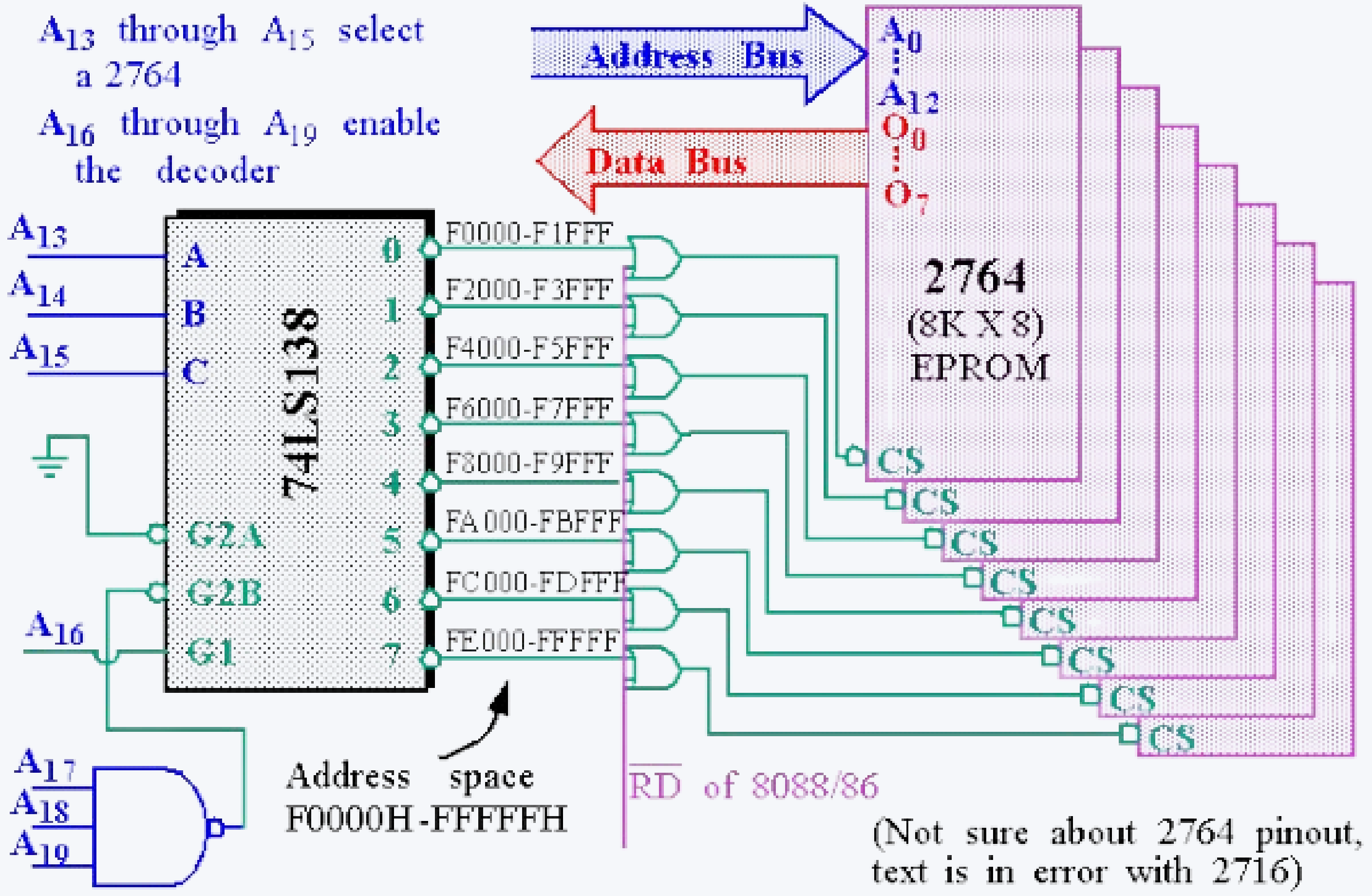
Note: a one-hot is a group of bits among which the legal combinations of values are only those with a single high (1) bit and all the others low (0). Example: 1000, 0100, 0010, 0001 (4 bits)

What are encoders and decoders?

Encoders: convert 1-hot input to binary code

Decoders: convert binary input to 1-hot output

Used in address decoding, multiplexing, keyboard scan, display control



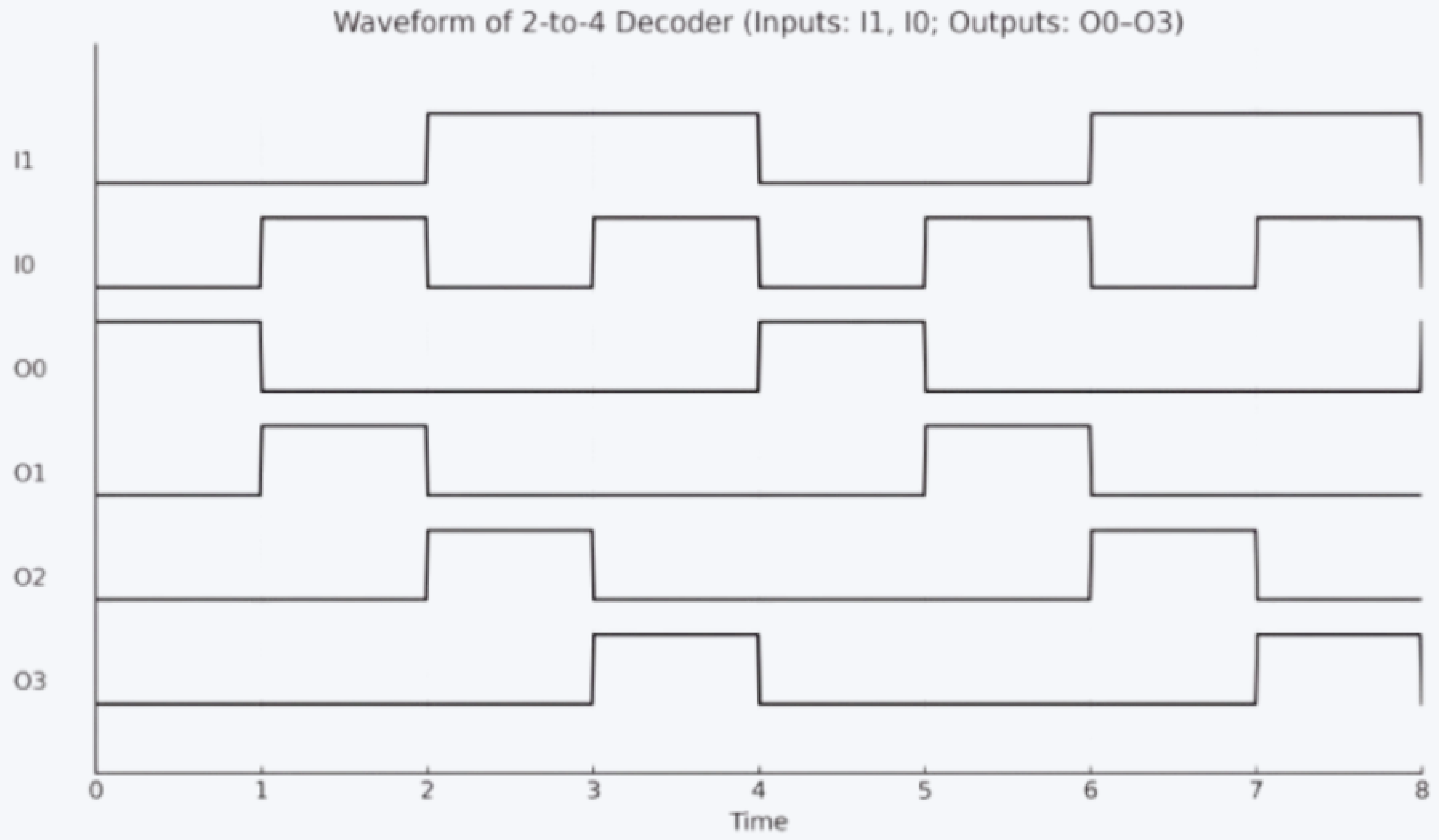
Encoders and decoders are complementary components. Decoders take a binary input and activate one of many outputs, while encoders do the reverse. You'll find them in systems where you need to either select something or identify where a signal is coming from.

Decoder: Logical Function

n inputs → 2^n outputs

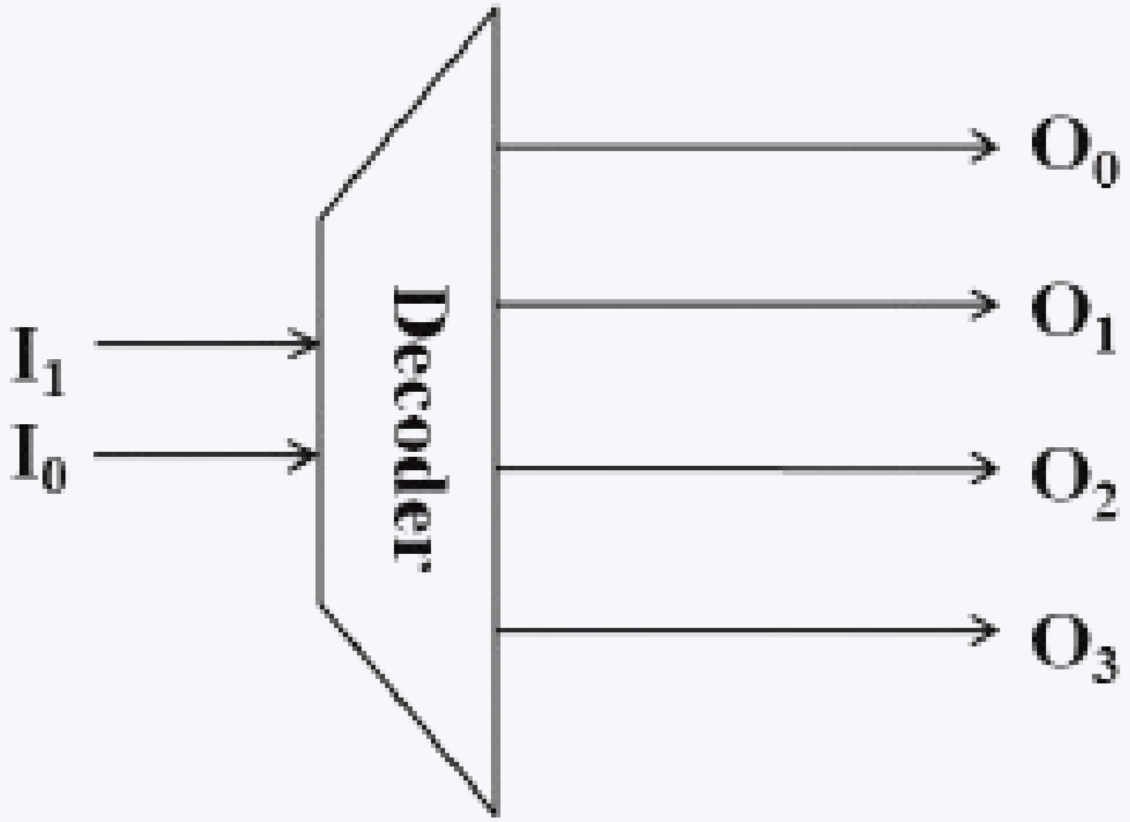
Only 1 output is active at a time

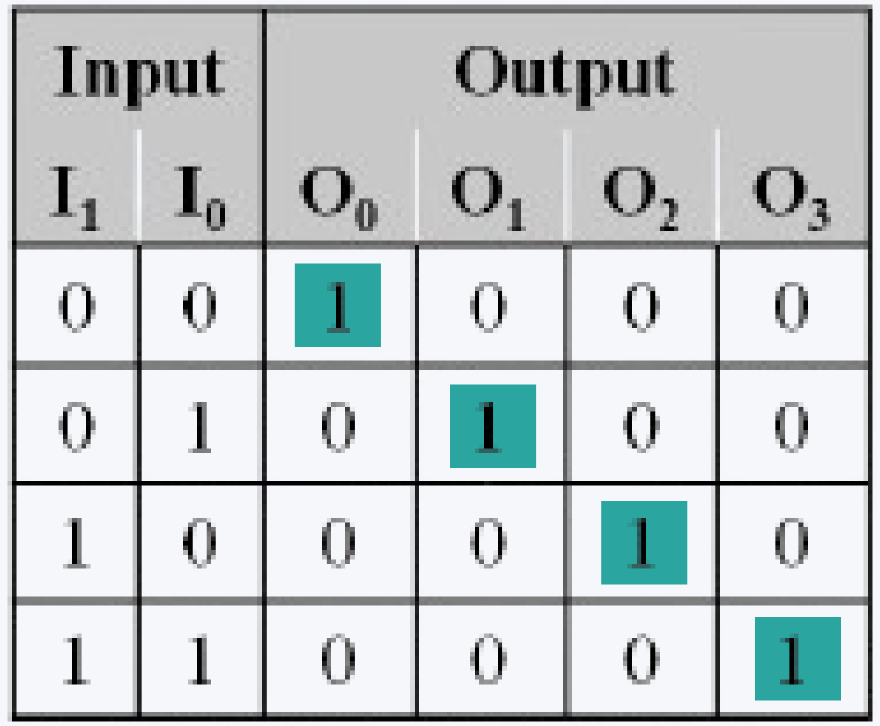
2-to-4 decoder: 2-bit input → 4 one-hot outputs



For a decoder, we have a binary number as input.
Based on that value, one of the 2^n outputs is set to logic 1.
All the others are 0.
This is useful, for example, when we want to enable a specific block of memory or a specific peripheral

Decoder: Truth Table (2-> 4)





Input		Output			
I_1	I_0	O_0	O_1	O_2	O_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Which line is active from input 11?

This truth table shows exactly how the binary input gets decoded. Input 00 selects output O0, input 01 selects O1, and so on. Only one output is active at a time, which is the core idea of decoding.

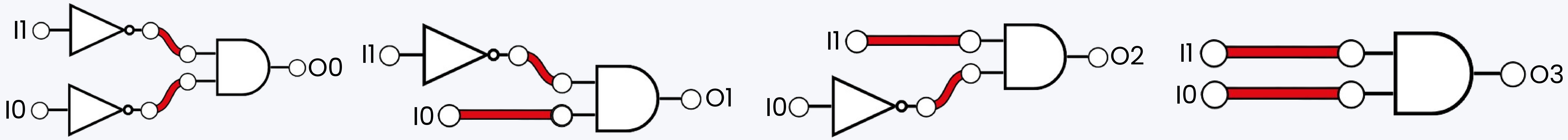
Decoder: Logic Equations

$$O0 = \neg I1 \cdot \neg I0$$

$$O1 = \neg I1 \cdot I0$$

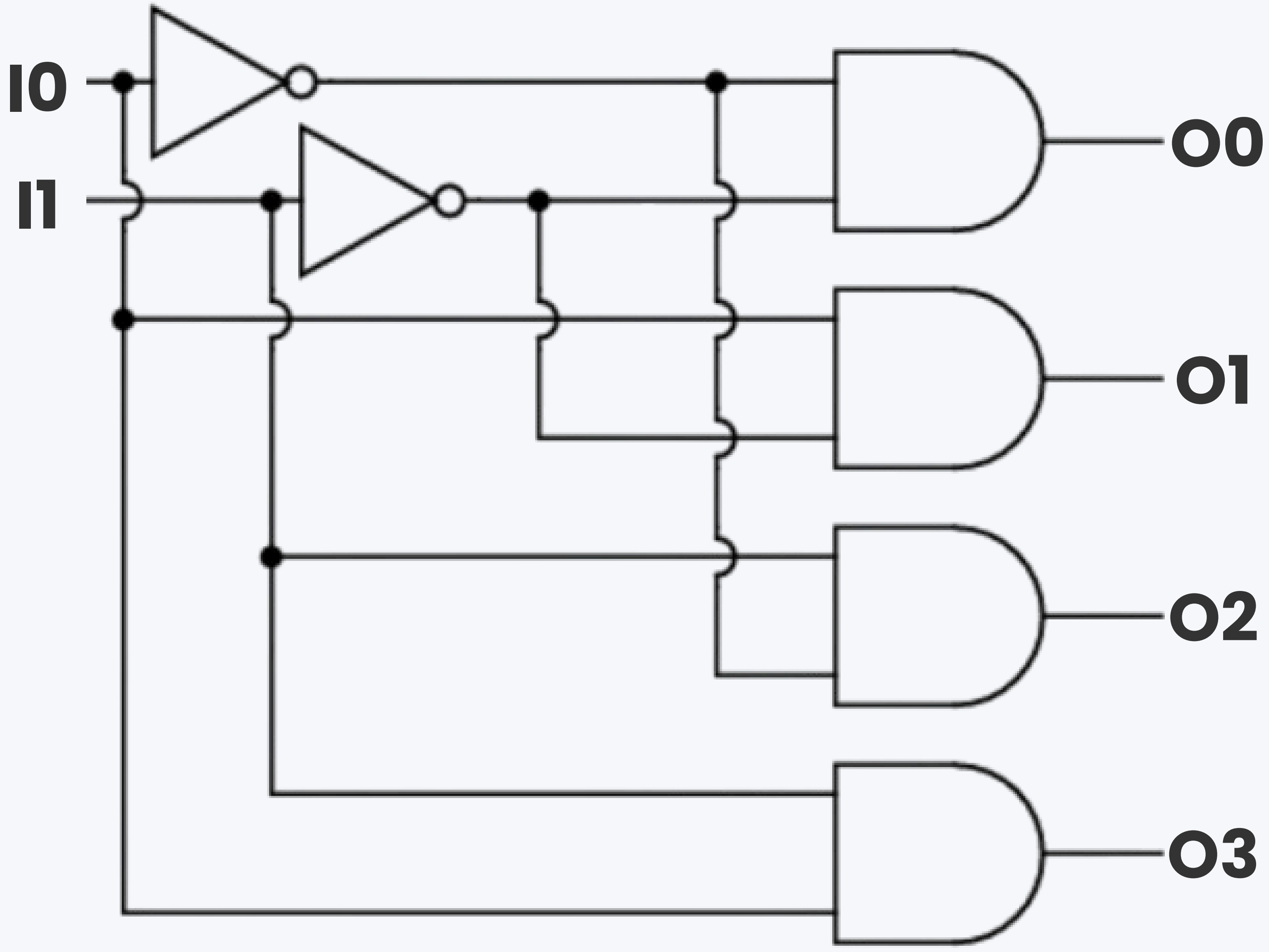
$$O2 = I1 \cdot \neg I0$$

$$O3 = I1 \cdot I0$$



Each output is expressed as a logical AND of input bits or their negations. These are simple expressions that translate directly to AND and NOT gates. This is how we implement the decoder in real logic circuits.

Decoder: Gate-Level Block



Here’s what the decoder looks like at gate level.
You’ll notice we use inverters to get the complemented inputs, and each AND gate represents a condition for one output. This is a clean and predictable design

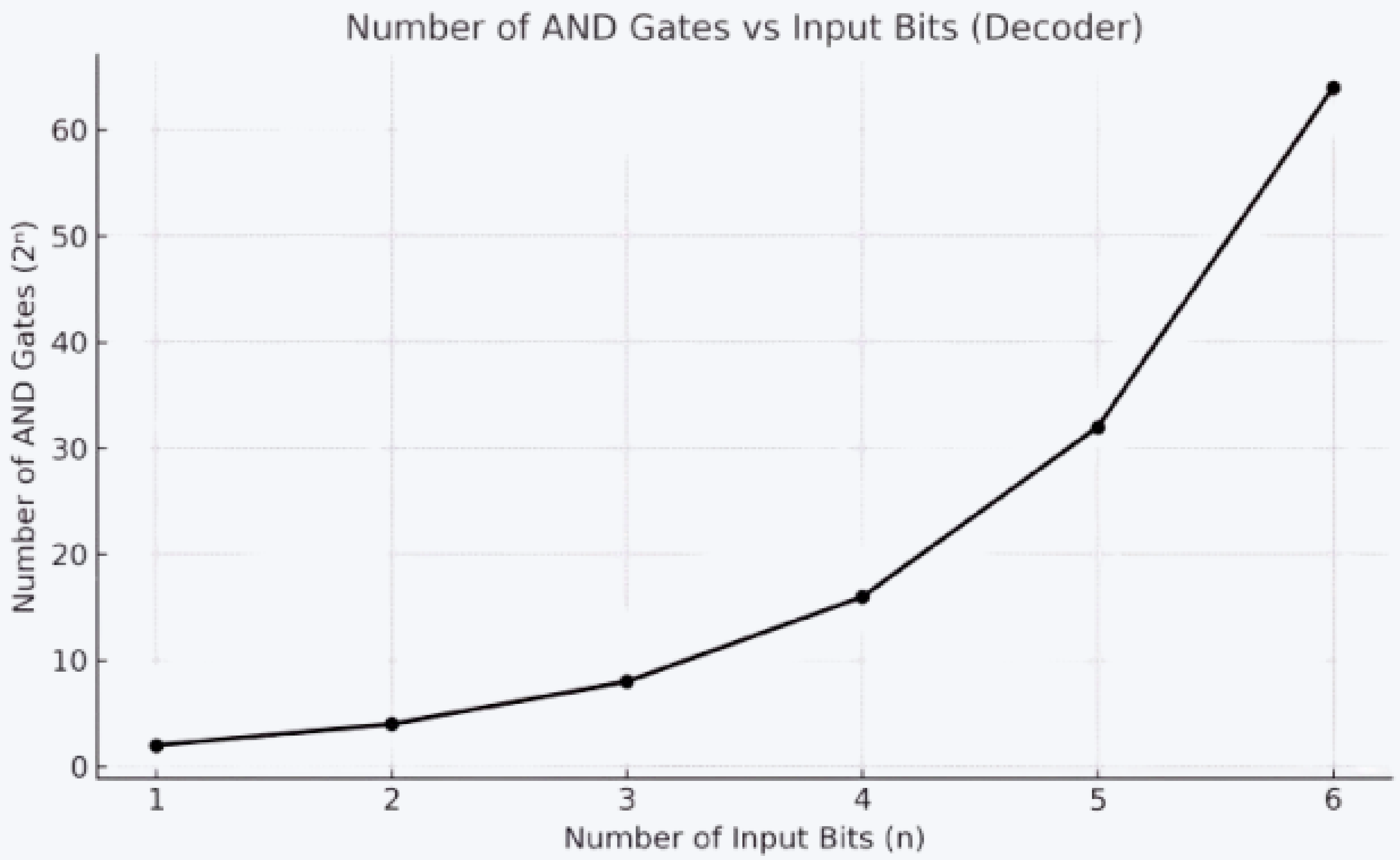
Decoder: Area and Delay

Area:

- 2^n AND gates
- n NOT gates
- Delay:

  Tinv + Tand_n

  Scales with n



As the number of input bits increases, the number of outputs—and therefore AND gates—increases exponentially.

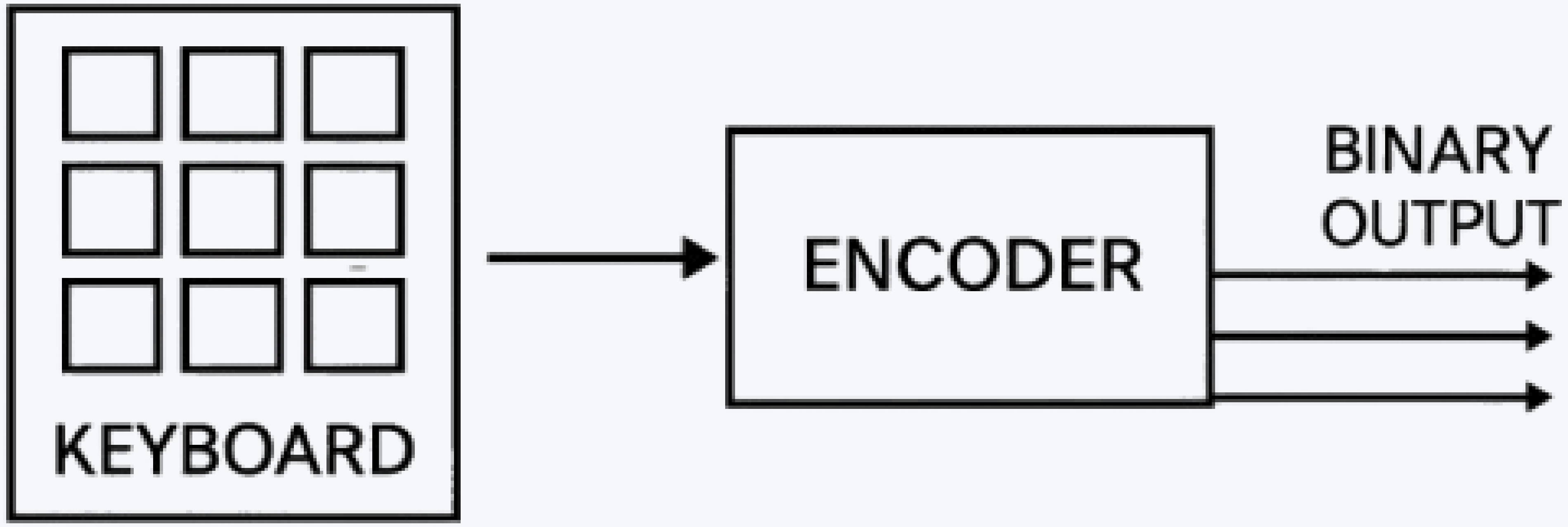
Delay depends mostly on inverter and AND gate depth, which is relatively small for low n, but noticeable for large decoders.

Encoder: Logical Function

2^n inputs → $\log_2(n)$ output bits

Only 1 input active at a time

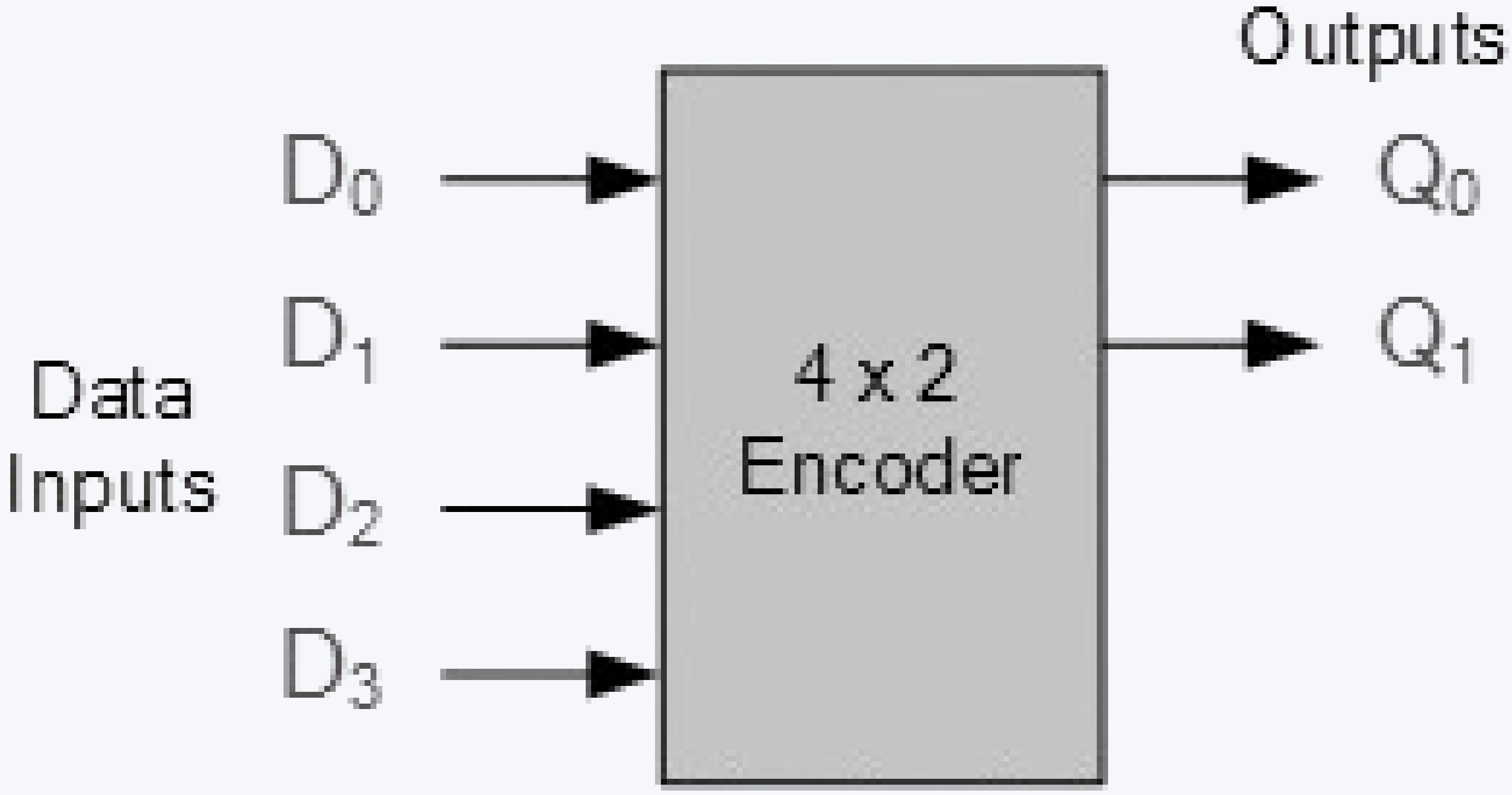
4-to-2 encoder: I0–I3 → Y1 Y0

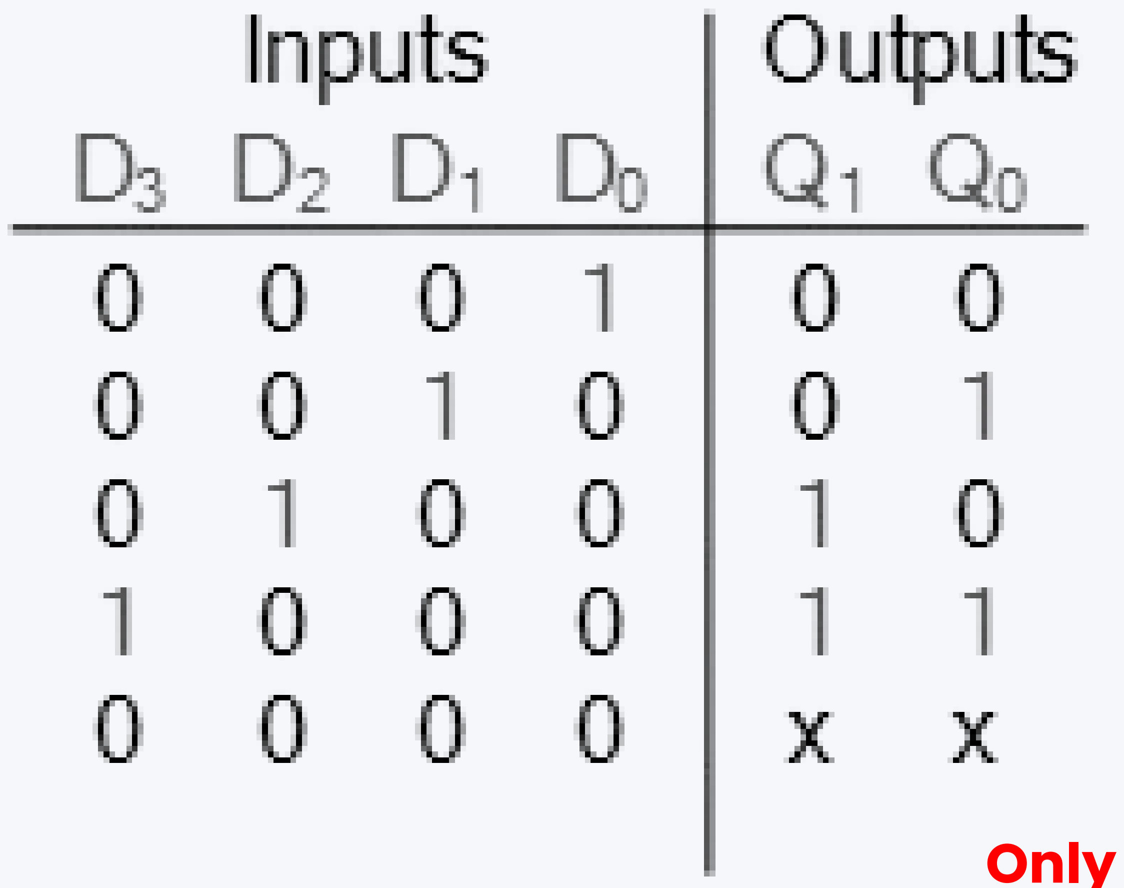


An encoder does the inverse of a decoder. Instead of taking a binary input and activating an output, it takes a set of one-hot inputs and compresses them into a binary output code.

Encoder: Truth Table (4 → 2)

What would happen if D2 and D3 were both on?





Inputs				Outputs	
D_3	D_2	D_1	D_0	Q_1	Q_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1
0	0	0	0	x	x

Only one input should be active!

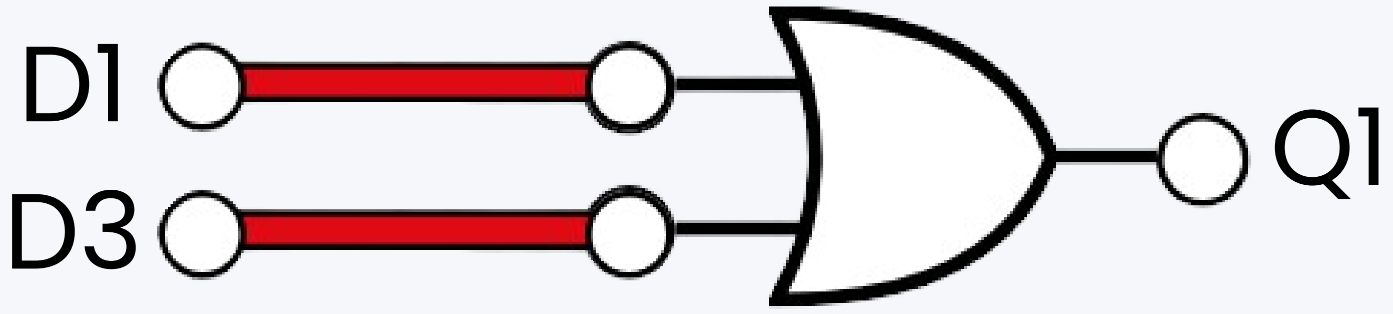
Here’s the reverse of our decoder truth table. Only one input is active at a time, and depending on which one it is, we output the binary representation of its index.

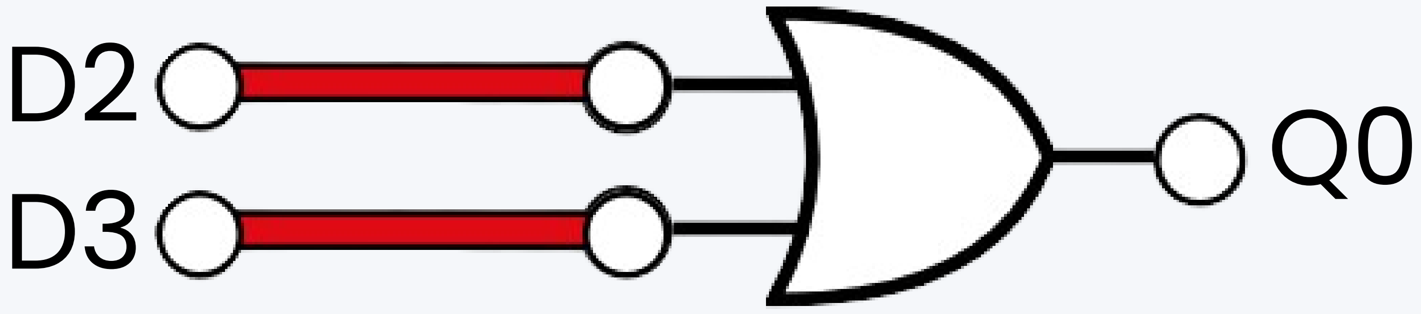
Encoder: Logical Equations

**The OR gate works only when one input is at HIGH level.
It would create ambiguity whenever more than one input in an equation were to be HIGH at the same time!**

$$Q1 = D1 \lor D3$$

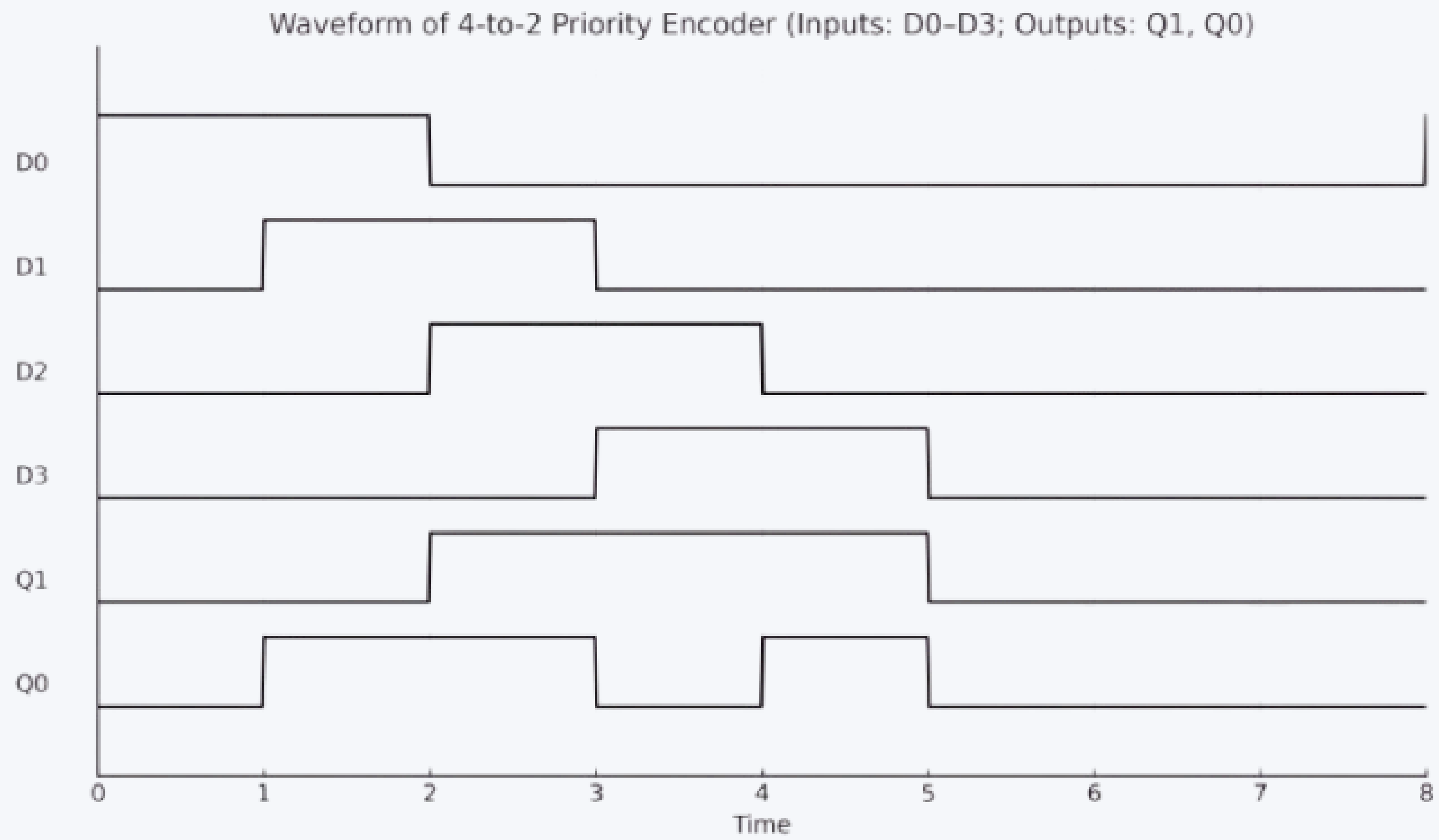
$$Q0 = D2 \lor D3$$

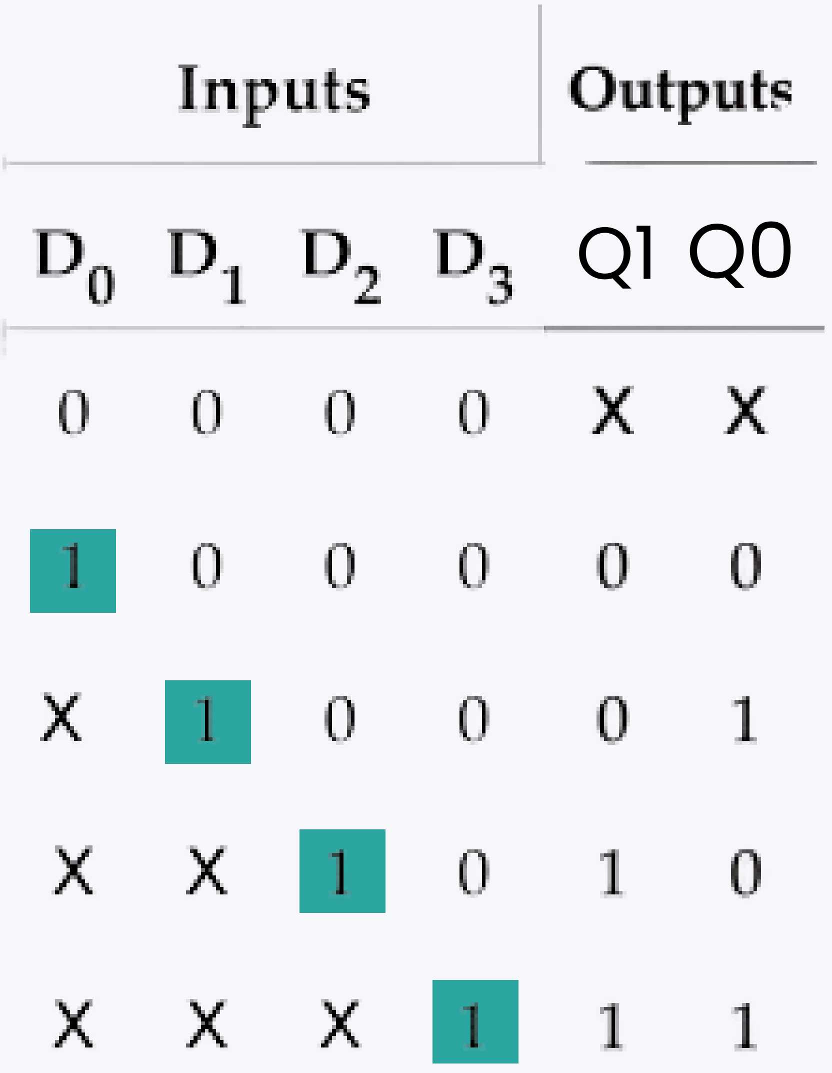




This is one way to write output expressions from the encoder truth table. But remember: this works correctly only when a single input is high. Multiple high inputs would create ambiguous results.

Priority Encoder





Inputs				Outputs	
D_0	D_1	D_2	D_3	Q1	Q0
0	0	0	0	X	X
1	0	0	0	0	0
X	1	0	0	0	1
X	X	1	0	1	0
X	X	X	1	1	1

If D3 = 1 → Q = “11”
Else if D2 = 1 → Q = “10”
Else if D1 = 1 → Q = “01”
Else if D0 = 1 → Q = “00”

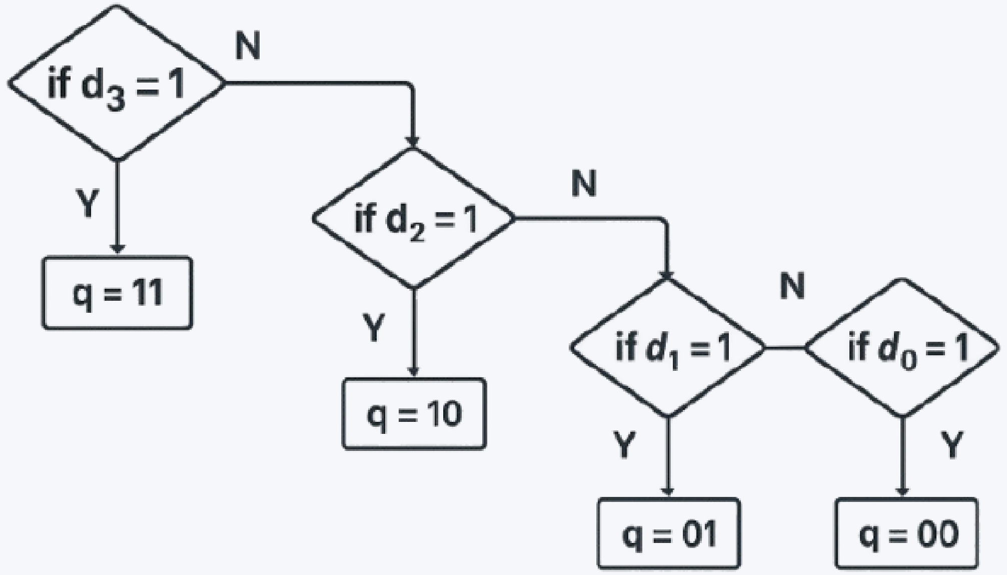
Enter the priority encoder. This version guarantees that if more than one input is active, the one with the highest priority (typically the most significant input) is selected. This is used in systems where multiple signals might be active but only one should be serviced.

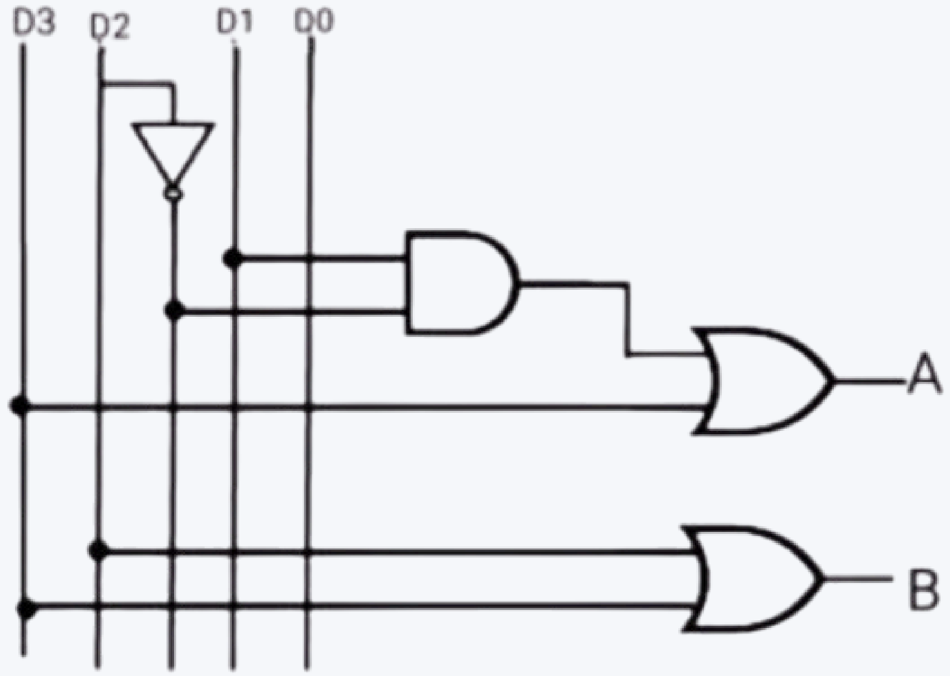
Priority Encoder: Logic View

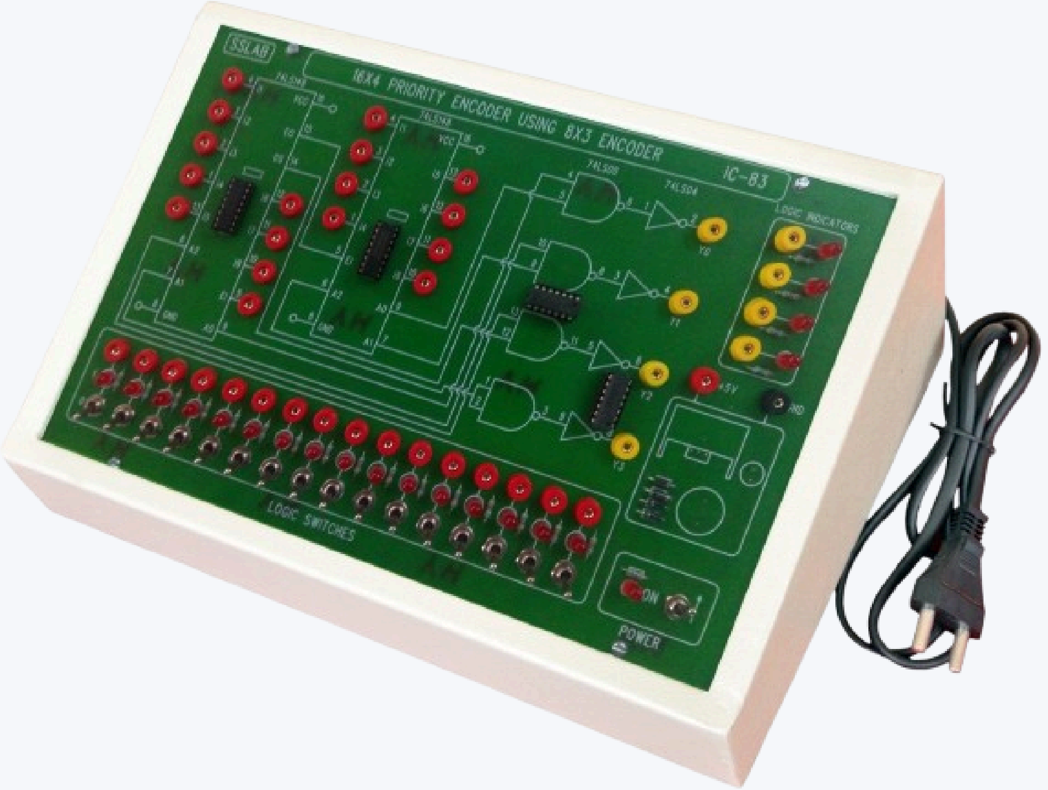
Conditions handled with if-elsif statements

Higher index = higher priority

Needed when multiple signals might be 1

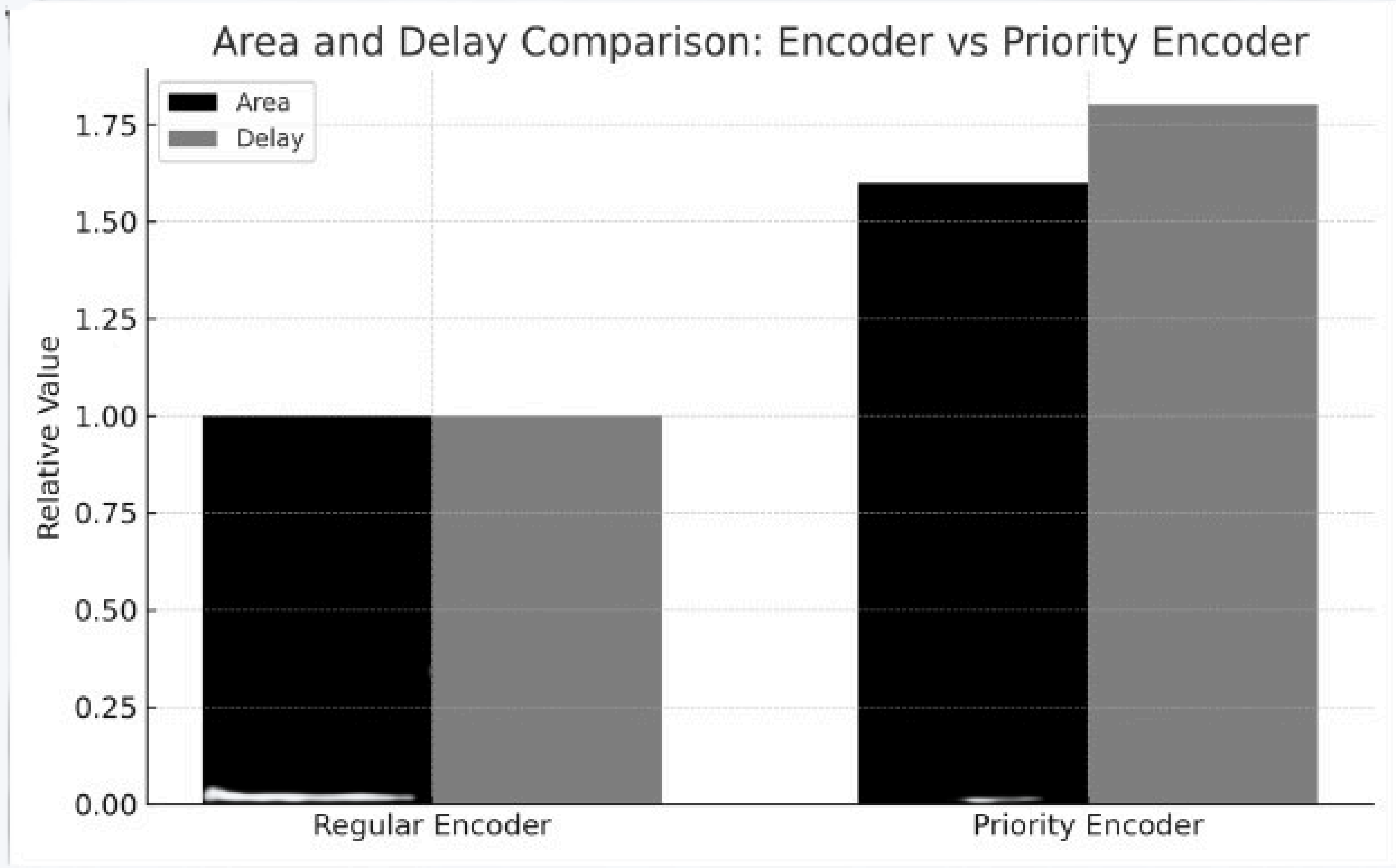






In hardware, this usually gets implemented with cascaded logic. It's more complex than a simple encoder but solves real-world issues in interrupt handling, bus arbitration, etc.

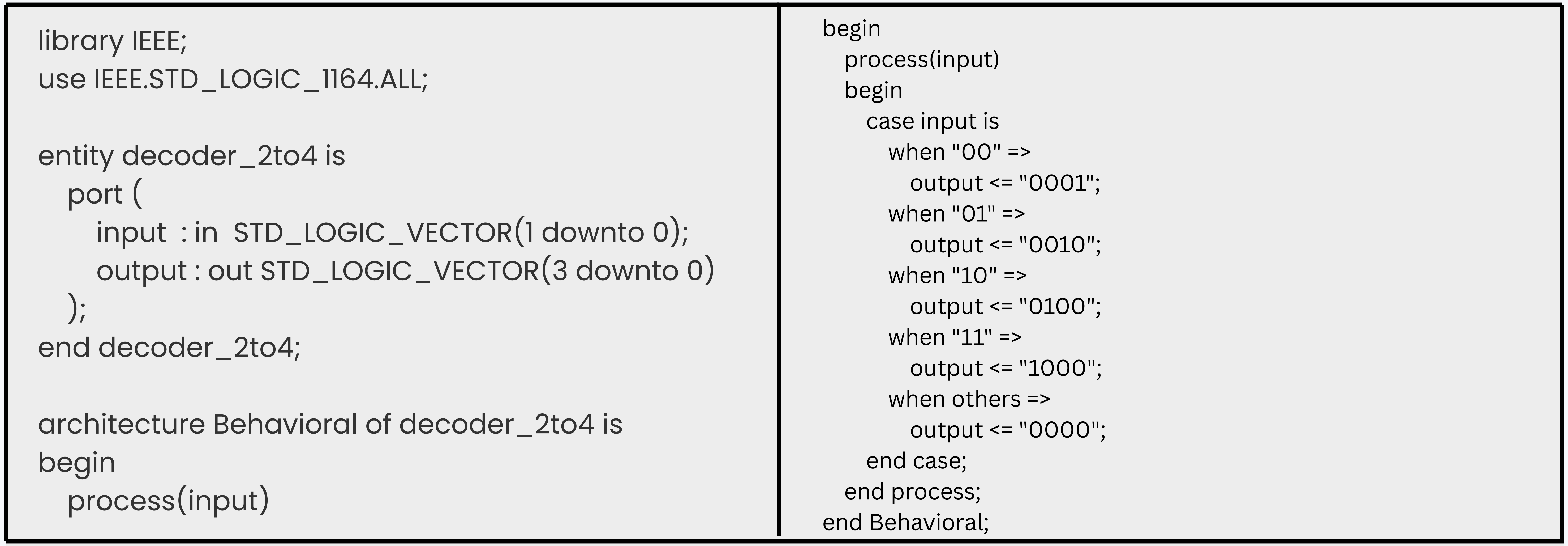
Priority Encoder: Area and Delay



More logic: ANDs, ORs, NOTs
Longer path → higher delay
Useful in CPU interrupt handlers

More logic means more gates, which means more delay. The path from input to output is longer, so it’s slower than a basic encoder. But it’s deterministic—which is crucial in many systems.

VHDL: 2-to-4 Decoder



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity decoder_2to4 is
    port (
        input  : in  STD_LOGIC_VECTOR(1 downto 0);
        output : out STD_LOGIC_VECTOR(3 downto 0)
    );
end decoder_2to4;

architecture Behavioral of decoder_2to4 is
begin
    process(input)
```

```
begin
  process(input)
  begin
    case input is
      when "00" =>
        output <= "0001";
      when "01" =>
        output <= "0010";
      when "10" =>
        output <= "0100";
      when "11" =>
        output <= "1000";
      when others =>
        output <= "0000";
    end case;
  end process;
end Behavioral;
```

Here's how you'd write a 2-to-4 decoder in VHDL. The case statement maps input values to output lines. Simple, readable, and synthesizable.

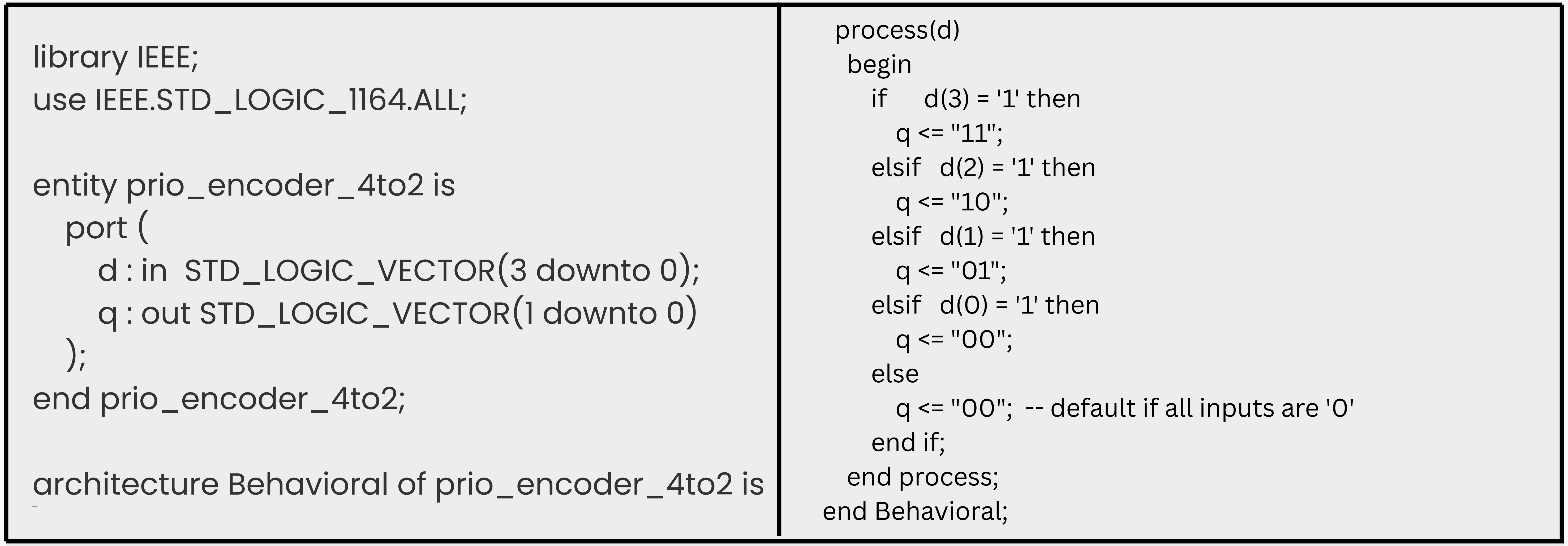
VHDL: 4-to-2 Encoder (Non-priority)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity encoder_4to2 is
  port (
    input  : in  STD_LOGIC_VECTOR(3 downto 0);
    output : out STD_LOGIC_VECTOR(1 downto 0)
  );
end encoder_4to2;

architecture Behavioral of encoder_4to2 is
begin
```

```
 begin
     case input is
       when "0001" =>
         output <= "00";
       when "0010" =>
         output <= "01";
       when "0100" =>
         output <= "10";
       when "1000" =>
         output <= "11";
       when others =>
         output <= "00";  -- default if invalid or multiple
bits set
     end case;
   end process;
end Behavioral;
```

This encoder assumes only one input is high. It works like a lookup table. But you must be careful: if multiple inputs are 1, the output is undefined.

VHDL: Priority Encoder



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity prio_encoder_4to2 is
  port (
    d : in  STD_LOGIC_VECTOR(3 downto 0);
    q : out STD_LOGIC_VECTOR(1 downto 0)
  );
end prio_encoder_4to2;

architecture Behavioral of prio_encoder_4to2 is
```

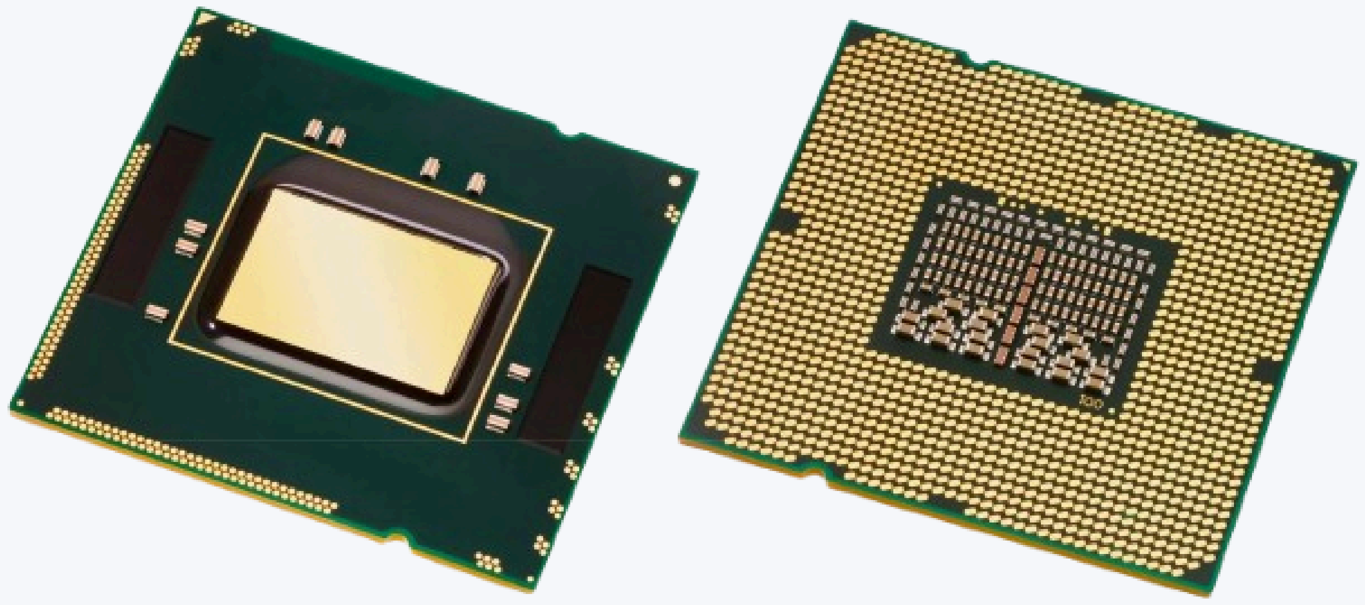
```
  process(d)
   begin
     if     d(3) = '1' then
       q <= "11";
     elsif  d(2) = '1' then
       q <= "10";
     elsif  d(1) = '1' then
       q <= "01";
     elsif  d(0) = '1' then
       q <= "00";
     else
       q <= "00";  -- default if all inputs are '0'
     end if;
   end process;
end Behavioral;
```

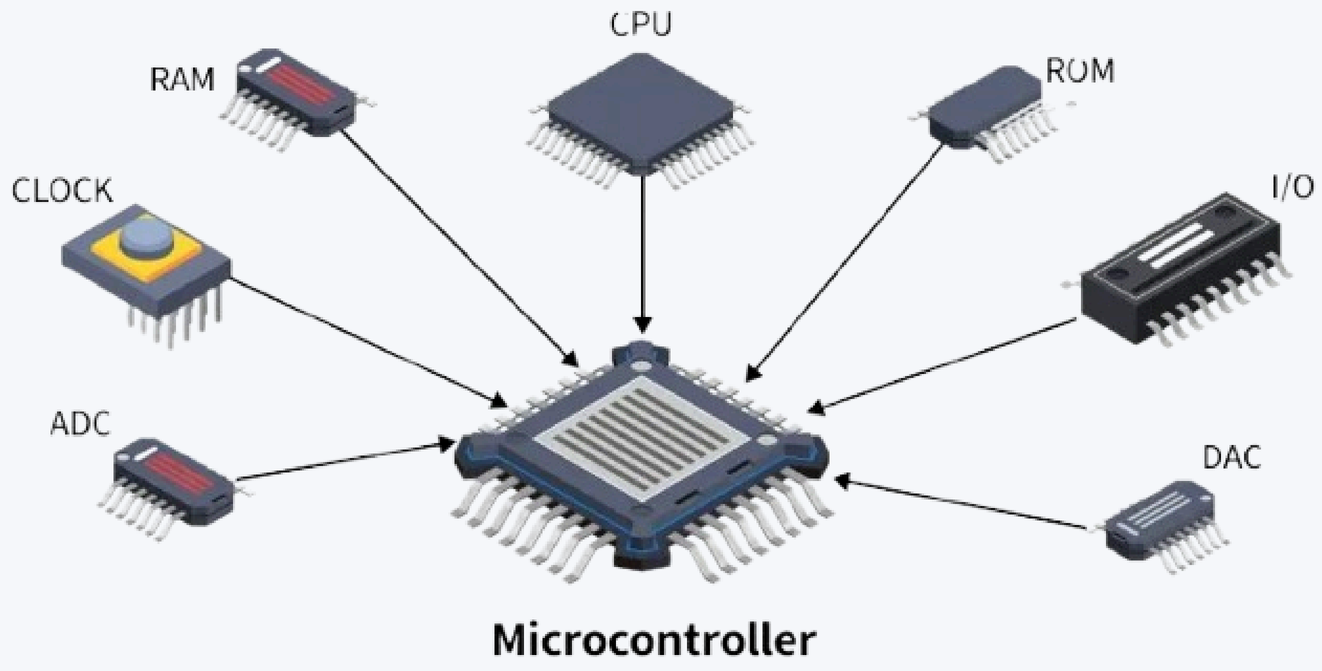
This version resolves conflicts. The if-elsif chain directly expresses the logic of the priority encoder. Most synthesis tools will generate hardware that reflects this priority.

System-Level Integration

Decoder → Address decoder, line selector

Encoder → Keyboard scanner, interrupt controller





Let’s step back. Where do these circuits get used? Decoders select among options based on binary input. Encoders do the opposite—they turn external signals into a manageable binary code.

Summary Table

Feature	Decoder	Encoder	Priority Encoder
Input	n bits	2^n lines	2^n lines
Output	2^n lines	$\log_2(n)$	$\log_2(n)$
Area	High	Low	Medium–High
Delay	Medium	Low	Medium–High

This table summarizes what we’ve covered. Each design has its trade-offs. You’ll want to consider area, delay, and deterministic behavior based on your use case.

Quiz & Discussion

Questions:

1. How many outputs for a 3-to-8 decoder?
2. What happens if two inputs are high in a non-priority encoder?
3. When is a priority encoder necessary?

Let’s check your understanding. Think carefully about each of these—it will help you decide when to use which architecture.

Conclusion & Assignment

Assignment:

Write a 3-to-8 decoder in VHDL

Simulate and test it

Bonus: Write a priority 8-to-3 encoder

Summary:

✓ Logical structure

✓ VHDL implementations

✓ Area vs delay tradeoffs

Today we’ve walked through the entire design process for encoders and decoders. Your homework is to implement a larger decoder and, optionally, a priority encoder. This will reinforce the concepts we explored together.