Decoders

- Another common circuit block. A decoder accepts n inputs and has 2^n outputs.
- The purpose of a decoder is to recognize (or "decode") the binary input pattern and set the corresponding output.
- Often, a decoder will have an *enable* signal. When the enable signal is low, all the outputs are 0. When the enable signal is high, the decoder performs its task.

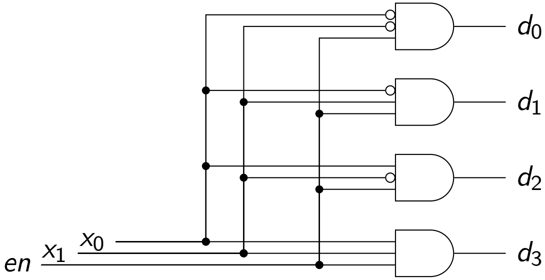
Decoders

- Example of a 2-to-4 decoder:

x_0	x_1	en	d_0	d_1	d_2	d_3
X	X	0	0	0	0	0
0	0	1	1	0	0	0
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	1

Decoders

- Circuit (unoptimized) for a 2-to-4 decoder:

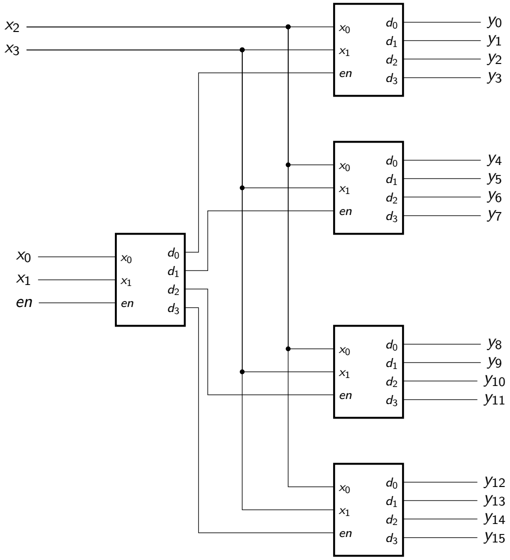


Decoder trees

- We can make larger decoders from smaller decoders.
- Example... 4-to-16 decoder built from 2-to-4 decoders.
  - Of the 4 inputs x_0, x_1, x_2 and x_3, the two most significant inputs x_0 and x_1 are used with one decoder to *enable* the appropriate decoder in the second stage.
  - The two least significant inputs x_2 and x_3 are used to generate the correct output.

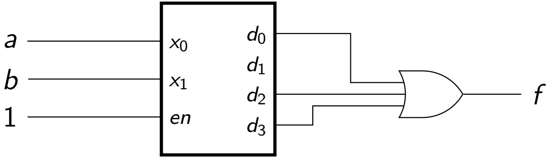
Decoder trees

- Circuit for the 4-to-16 decoder...



Function implementation with decoders

- Consider that a decoder is basically decoding the input pattern corresponding to a minterm.
- Consequently, if we have a decoder with n inputs and an **OR** gate, we can implement any n input function.
- Example... Implement $f = f(a, b) = \sum(0, 2, 3)$.



Encoders

- An *encoder* performs the inverse operation of a decoder — the encoder has 2^n inputs and generates n outputs. The output is a binary encoding of the input line which is set.
- Example of an 8-to-3 encoder...

d_0	d_1	d_2	d_3	d_4	d_5	d_6	d_7	x_0	x_1	x_2
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

- Encoder is implemented easily with **OR** gates; $x_0 = d_4 + d_5 + d_6 + d_7$, $x_1 = d_2 + d_3 + d_6 + d_7$ and $x_2 = d_1 + d_3 + d_5 + d_7$.

Priority encoders

- Simple encoder is a dumb circuit and has some problems:
  1. What if no input is set?
  2. What if multiple inputs are set?
- The solution to the first problem is to introduce another output, *valid*, which is set when any input is set, otherwise 0. This tells us when we have encoded an input value vs. having no input value.
- The solution to the second problem is to have *priority* and to design a so-called *priority encoder*. The output the circuit produces should be the encoding of the "highest indexed" (highest priority) input.

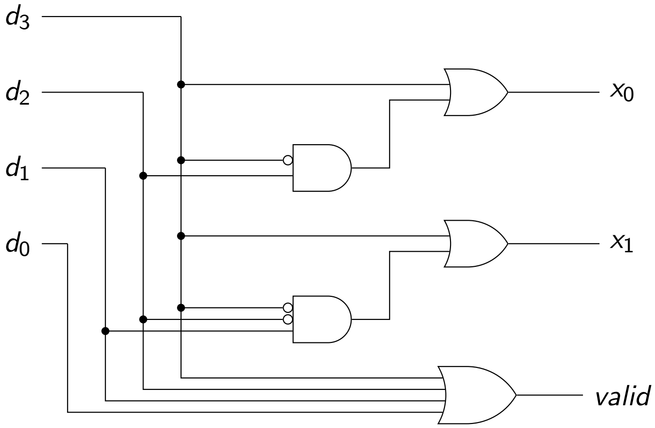
Priority encoders

- Operation of a 4-to-2 priority encoder:

d_0	d_1	d_2	d_3	x_0	x_1	*valid*	
0	0	0	0	0	0	0	$\leftarrow$ output not valid
1	0	0	0	0	0	1	
X	1	0	0	0	1	1	
X	X	1	0	1	0	1	
X	X	X	1	1	1	1	

Priority encoders

- Can write down (unoptimized) equations for the outputs and the valid signal easily to get a circuit.
- Example... circuit for the 4-to-2 priority encoder...



Priority encoders

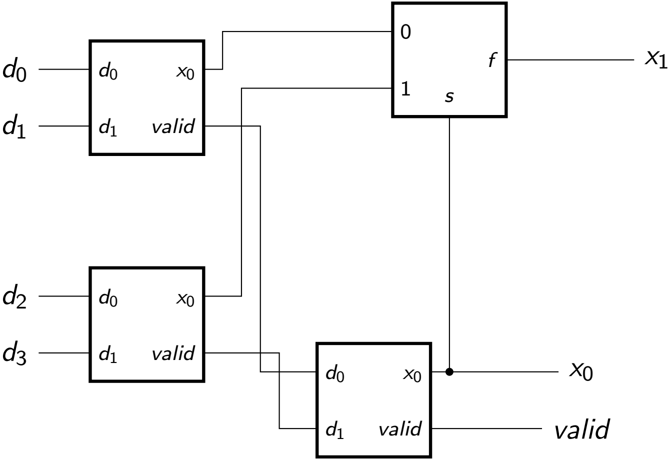
- For larger priority encoders, we will require gates with a larger number of inputs.
- It is interesting to consider whether or not we can construct larger priority encoders from smaller ones. We can if we use some additional multiplexers.
- Consider a 2-to-1 prioity encoder:

d_0	d_1	x_0	*valid*
0	0	0	0
1	0	0	1
X	1	1	1

- The logic equations are $valid = d_0 + d_1$ and $x_0 = d_1$ which are very simple.

Priority encoders

- Circuit for a 4-to-2 priority encoder using smaller priority encoders and some multiplexer...



- All gates inside of these blocks are 2-input gates.

Priority encoders

- We can go even further — we can design an 8-to-3 priority encoder from 4-to-2 priority encoders (plus some multiplexers).
- This also means we can design an 8-to-3 priority encoder using only 2-to-1 priority encoders.
- There is a pattern which emerges and it should be clear that we can continue the procedure to build any sized 2^n-to-n priority encoder from smaller encoders (and multiplexers).

Priority encoders

- 8-to-3 priority encoder from smaller circuits...