Chapter 6. Basic Combinational Logic Elements

We will consider basic building blocks of digital circuits, from two points of view.

View 1:
Read-only memories (ROMs) allow us to implement any truth table with a number of inputs and outputs determined by the size of the ROM.

When a function has too many inputs to fit into a single ROM, it can be decomposed to fit into multiple interconnected ROMs.

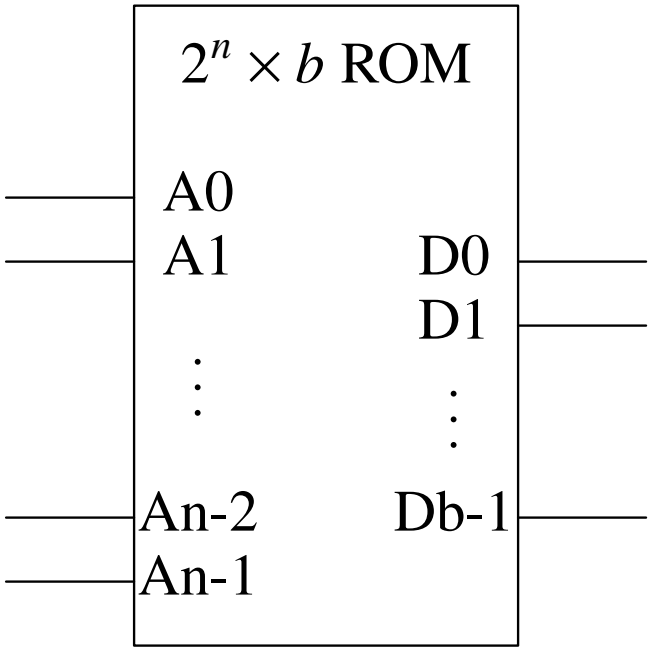
This is the basis for field-programmable gate arrays (FPGAs).

View 2:
There are commonly occurring operations such as decoding, selecting and comparing for which building blocks exist.

It is possible to combine these building blocks to form larger functions.

6.1. Read-Only Memories (ROMs)

A ROM is a combinational circuit with n inputs and b outputs.



Internally a ROM is a two-dimensional array with 2^n rows.
Each row stores a b-bit word.

The inputs $A_{n-1}, A_{n-2}, \cdots, A_1, A_0$ form an n-bit vector $A = A_{n-1}A_{n-2}\cdots A_1A_0$, called an address. The address A selects the corresponding row of the ROM.

The outputs $D_{b-1}, D_{b-2}, \cdots, D_1, D_0$, called the data outputs, form a b-bit word
$D = D_{b-1}D_{b-2}\cdots D_1D_0$.
The output D is equal to the word stored in the address given by A.

Once A is changed, the ROM outputs become unpredictable.
The outputs become stable after the propagation delay of the ROM.

A $2^n \times b$ ROM can store the truth table of a function with n inputs and b outputs.

Example of a truth table:
A decoder for $A_1 A_0$, with control input A_2.

A_2	A_1	A_0	D_3	D_2	D_1	D_0
0	0	0	0	0	0	1
0	0	1	0	0	1	0
0	1	0	0	1	0	0
0	1	1	1	0	0	0
1	0	0	1	1	1	0
1	0	1	1	1	0	1
1	1	0	1	0	1	1
1	1	1	0	1	1	1

The ROM needs to be programmed with the truth table.

ROM content:

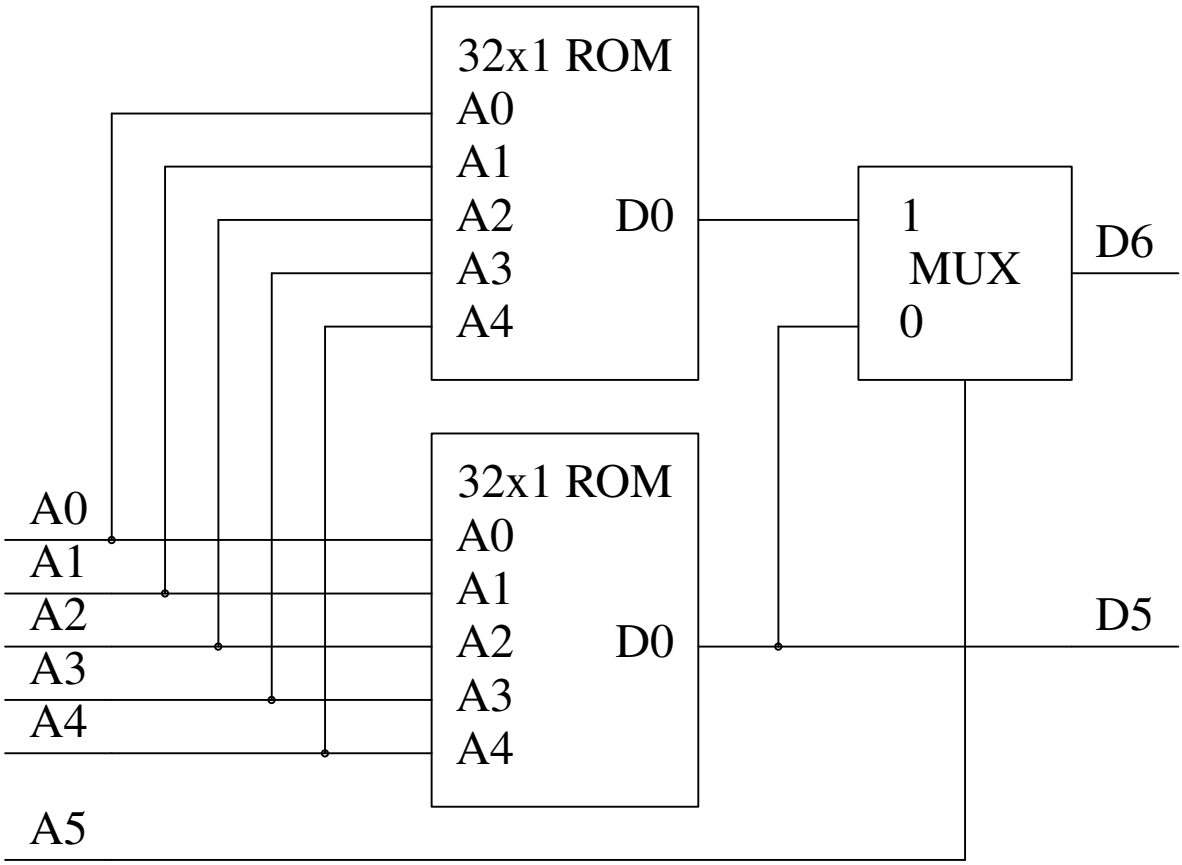
A2A1A0	D3	D2	D1	D0
0(000)	0	0	0	1
1(001)	0	0	1	0
2(010)	0	1	0	0
3(011)	1	0	0	0
4(100)	1	1	1	0
5(101)	1	1	0	1
6(110)	1	0	1	1
7(111)	0	1	1	1

FPGAs used small ROMs called *lookup tables (LUTs)* to implement logic functions.

A $2^n \times 1$ LUT can store any single-output logic function with up to n inputs by storing its truth table.
Small values of n are necessary to ensure that the LUTs are of practical size.

To implement a logic function using an FPGA, the function needs to be decomposed into an interconnection of smaller functions, each of which needs to fit in an available LUT.

LUTs in the Xilinx 7 series FPGAs have $n = 6$. Instead of a 64×1 ROM, an LUT is implemented as follows.



To implement an arbitrary function with $n = 6$ inputs:
The minterms with $A_5 = 0$ are programmed into the lower ROM.
The minterms with $A_5 = 1$ are programmed into the upper ROM.

To implement an arbitrary function with $n = 5$ inputs and two outputs:
The truth table for the first output is programmed into the lower ROM.
The truth table for the second output is programmed into the upper ROM.
$A_5 = 1$.

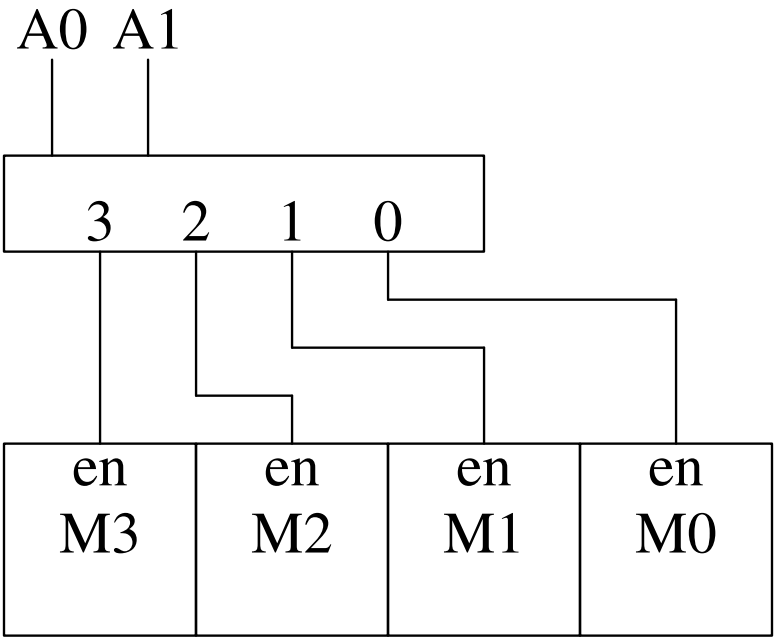
Read:
6.2 Combinational PLDs

6.3. Decoding and Selecting

A decoder can select a memory address, or a specific component, based on its address.

Example: Two address bits, $A_1 A_0$, are used for selecting one of four memories.

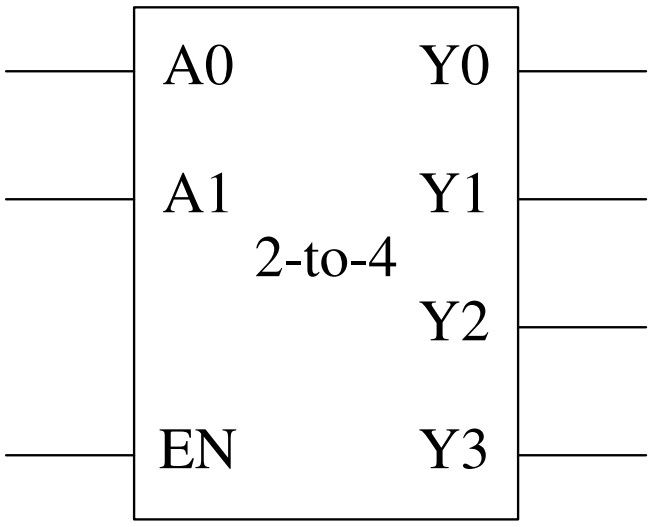
A_1	A_0	En_3	En_2	En_1	En_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0


A0 A1
3 2 1 0
en
M3
en
M2
en
M1
en
M0

A *decoder* is a multiple-input, multiple-output logic circuit that converts an input code word into an output code word

A decoder may have one or more enable inputs. The enable inputs must be asserted for the decoder to perform its normal function.
Otherwise, all the input code words are mapped into a single default output code word.

A logic symbol for a 2-to-4 decoder with an enable input:



Examples of input codes:

n-bit binary code with 2^n different input code words.
The decoder for the four memories is an example with $n = 2$.

With five memories, it is possible to use the input code words 000, 001, 010, 011 and 100 for the memories, and leave 101, 110 and 111 unused.

BCD code is the four-bit binary code for the decimal digits 0-9.
It includes the input code words
0000, 0001, $\cdots$, 1001.
The combinations 1010, $\cdots$, 1111 are not used.

Gray code is the code used for labeling the cells of a K-map.
$n = 1$: 0 1
$n = 2$: 00 01 11 10
$n = 3$: 000 001 011 010 110 111 101 100

Example of output code:

1-out-of-m code (or one-hot code).
The output code words have m bits.
Exactly one bit is asserted at any time.
With $m = 4$ we obtain the output code words 0001, 0010, 0100, 1000.
The decoder for the four memories is an example with $m = 4$.

A 1-out-of-*m* code can have an additional output code word for the case where the decoder is disabled.
Example: 0001, 0010, 0100, 1000 for selecting one of four memories, and 0000 for disabled (no memory is selected).

The input code typically has fewer bits than the output code.
Each input code word produces a different output code word (one-to-one mapping).

Binary Decoders

A binary decoder is an $n - to - 2^n$ decoder with an n-bit binary input code and a 1-out-of-2^n output code.

A binary decoder is used when it is necessary to activate exactly one of 2^n outputs based on an n-bit input value.

Truth table of a 2-to-4 binary decoder:

EN	A1	A0	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

Output functions:

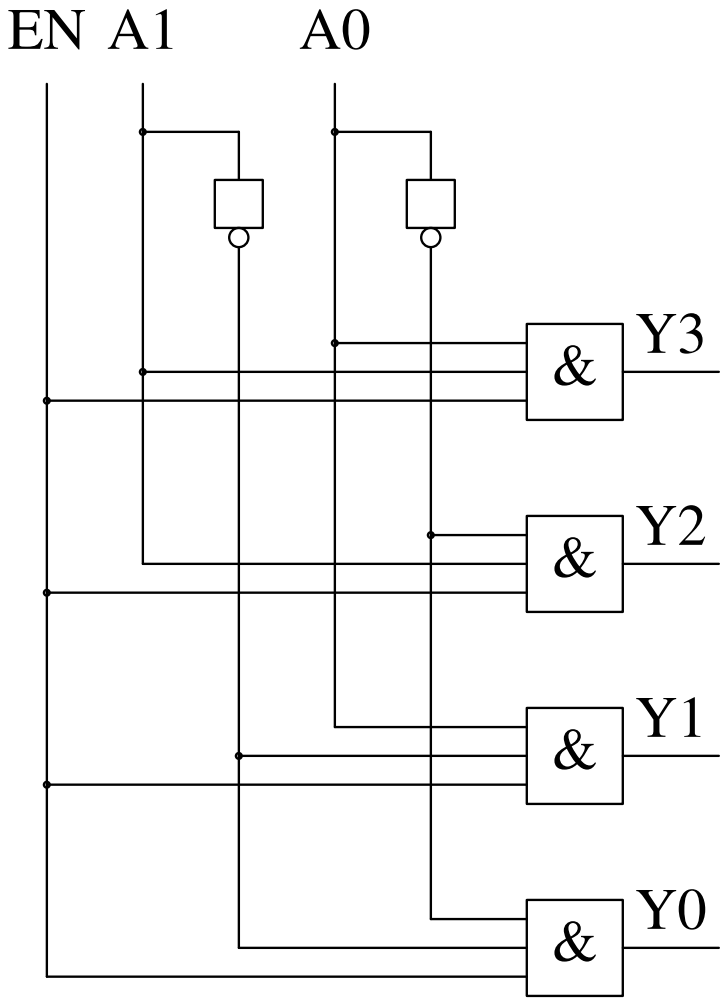
$Y_0 = EN \cdot A1' \cdot A0'$

$Y_1 = EN \cdot A1' \cdot A0$

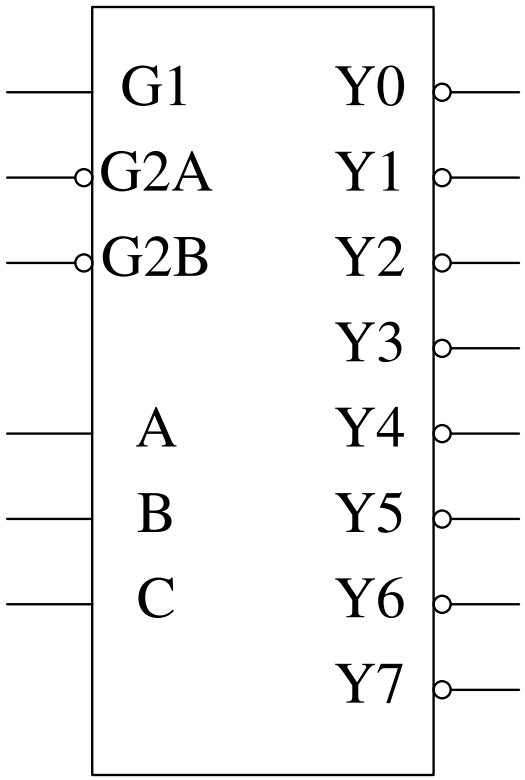
$Y_2 = EN \cdot A1 \cdot A0'$

$Y_3 = EN \cdot A1 \cdot A0$

Circuit:



This structure can be extended to any *n*.
Example: A 3-to-8 binary decoder with
active low outputs, and
three enable inputs, all of which must be asserted to enable the outputs.
Logic symbol:



Truth table:

G1	G2A	G2B	C	B	A
0	x	x	x	x	x
x	1	x	x	x	x
x	x	1	x	x	x
1	0	0	0	0	0
1	0	0	0	0	1
1	0	0	0	1	0
1	0	0	0	1	1
1	0	0	1	0	0
1	0	0	1	0	1
1	0	0	1	1	0
1	0	0	1	1	1

Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	0
1	1	1	1	1	1	0	1
1	1	1	1	1	0	1	1
1	1	1	1	0	1	1	1
1	1	1	0	1	1	1	1
1	1	0	1	1	1	1	1
1	0	1	1	1	1	1	1
0	1	1	1	1	1	1	1

Output functions:

$$EN = G1 \cdot G2A' \cdot G2B'$$
$$Y_0 = (EN \cdot C' \cdot B' \cdot A')'$$
$$Y_1 = (EN \cdot C' \cdot B' \cdot A)'$$
$$Y_2 = (EN \cdot C' \cdot B \cdot A')'$$
$$Y_3 = (EN \cdot C' \cdot B \cdot A)'$$
$$Y_4 = (EN \cdot C \cdot B' \cdot A')'$$
$$Y_5 = (EN \cdot C \cdot B' \cdot A)'$$
$$Y_6 = (EN \cdot C \cdot B \cdot A')'$$
$$Y_7 = (EN \cdot C \cdot B \cdot A)'$$

In general, a decoder with n address inputs contains n-input AND (or NAND) gates.
The number of AND (or NAND) gates is 2^n.

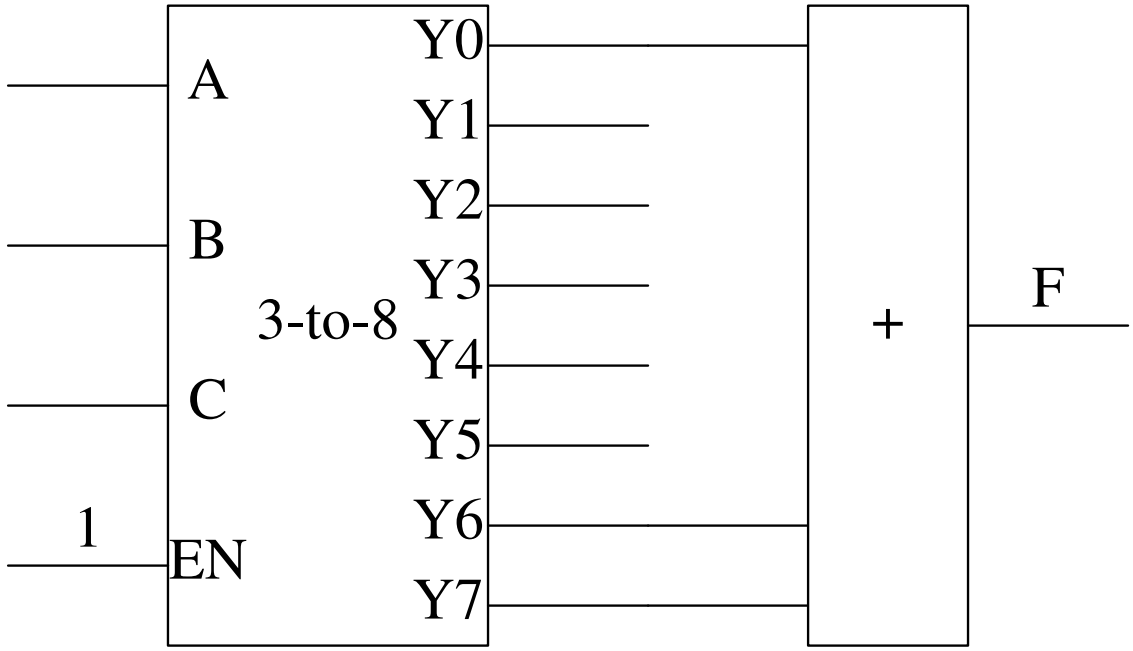
It is possible to implement any n-variable function with an $n - to - 2^n$ decoder and an OR gate.

This can be done based on the canonical sum of the function.

Example: $F(A, B, C) = A' \cdot B' \cdot C' + A \cdot B =$
$A' \cdot B' \cdot C' + A \cdot B \cdot C' + A \cdot B \cdot C$

The canonical sum provides the minterms where the function is 1.
The decoder has an output that produces a 1 for each minterm.
Connect to the inputs of an OR gate the decoder outputs corresponding to the 1-minterms of the function.

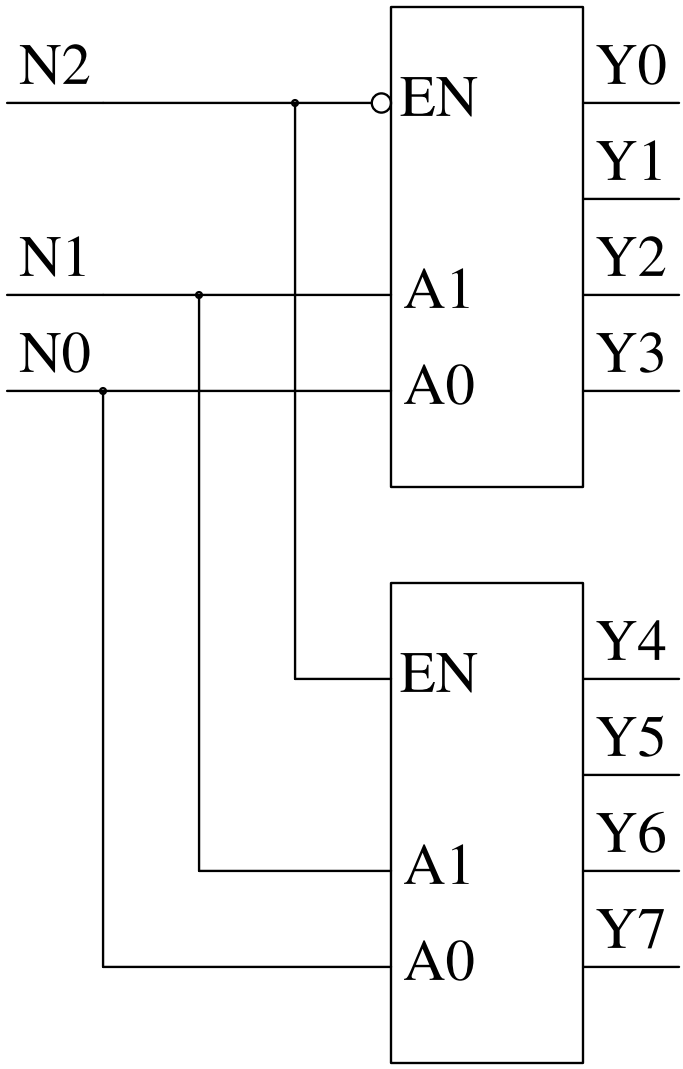
$$F(A, B, C) = A' \cdot B' \cdot C' + A \cdot B \cdot C' + A \cdot B \cdot C$$



This is typically not a minimal implementation.

It is possible to connect several binary decoders to form a larger decoder.

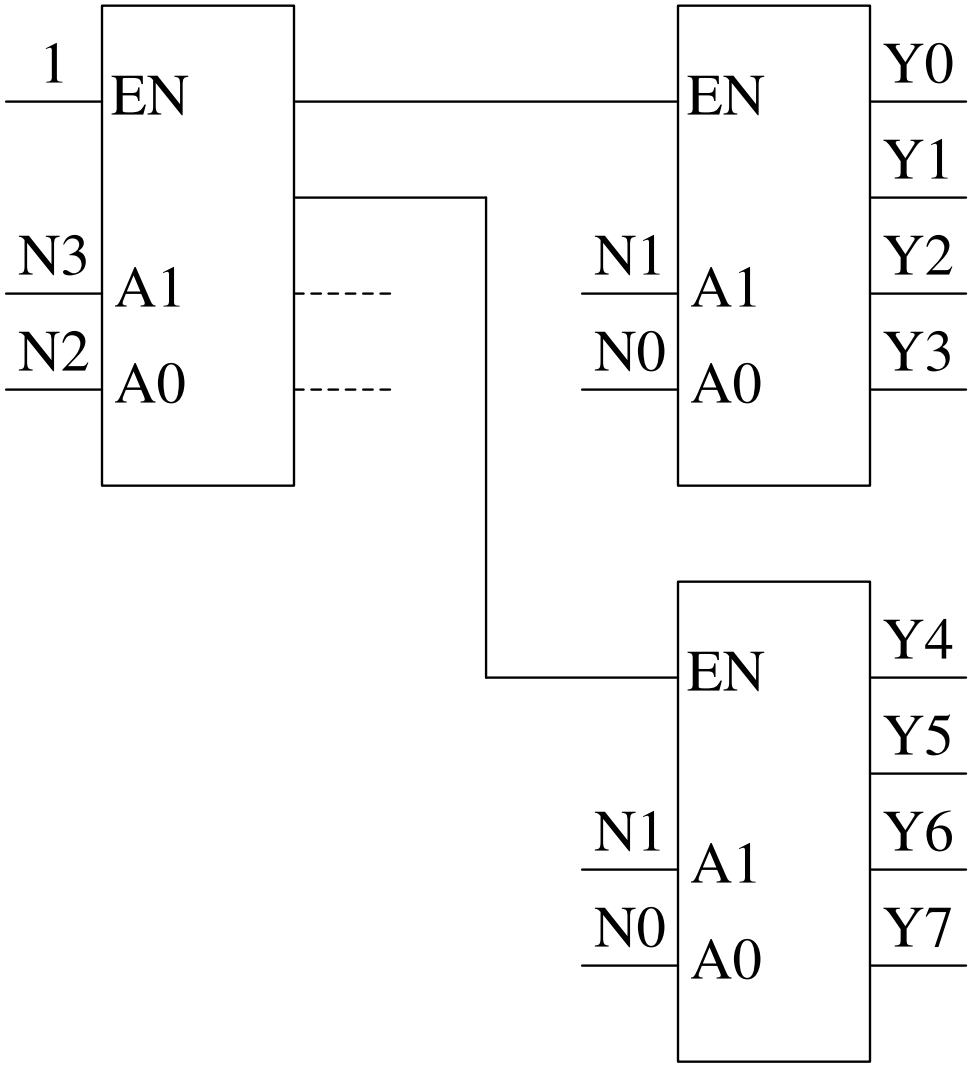
A 3-to-8 binary decoder from two 2-to-4 binary decoders:



Effectively, this partitions the truth table of the decoder into two parts:

N2	N1	N0	
0	0	0	top
0	0	1	decoder
0	1	0	
0	1	1	
1	0	0	bottom
1	0	1	decoder
1	1	0	
1	1	1	

A 4-to-16 binary decoder from five 2-to-4 binary decoders:



Effectively, this partitions the truth table of the decoder into four parts:

decoder zero N3	N2	N1	N0	
0	0	0	0	first
0	0	0	1	decoder
0	0	1	0	
0	0	1	1	
0	1	0	0	second
0	1	0	1	decoder
0	1	1	0	
0	1	1	1	
1	0	0	0	third
1	0	0	1	decoder
1	0	1	0	
1	0	1	1	
1	1	0	0	fourth
1	1	0	1	decoder
1	1	1	0	
1	1	1	1	

Read:
6.4.4. Larger Decoders (predecoding)

6.3.4. Decoders in Verilog

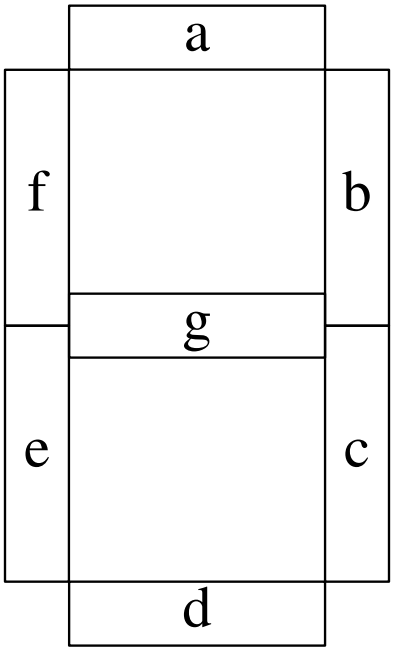
We will see one example in behavioral Verilog.
The section has many other examples.

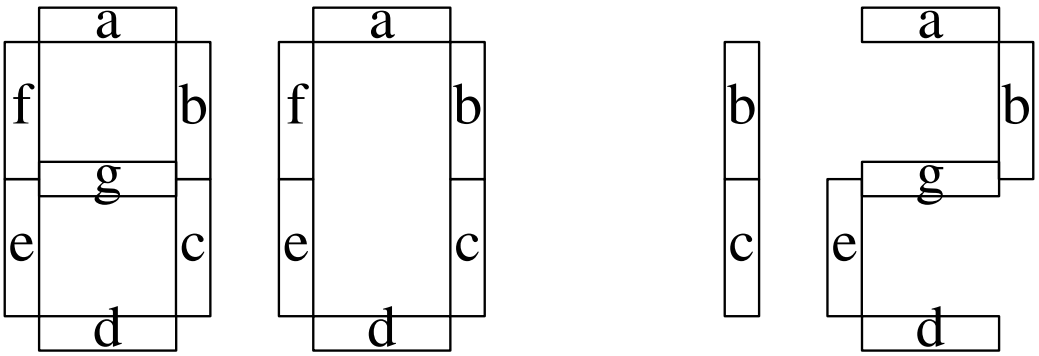
```
module VrDecoder(A,EN,Y);
parameter N=3, S=8;
    input [N-1:0] A;
    input EN;
    output reg [S-1:0] Y;
    always @(*)
    begin
        Y=0;
        if(EN==1) Y[A]=1;
    end
```

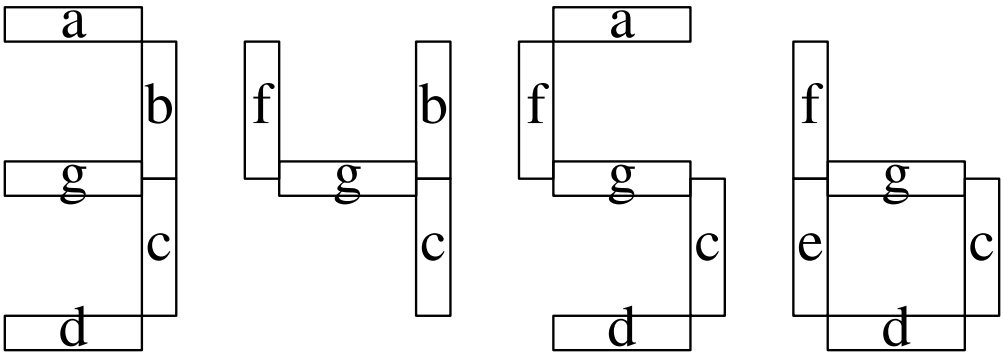
If $EN = 0$, $Y = 0$.
Otherwise, A may be one of the N-bit (3-bit) values $0, 1, \cdots, 7$.
The output given by A is assigned the value 1.

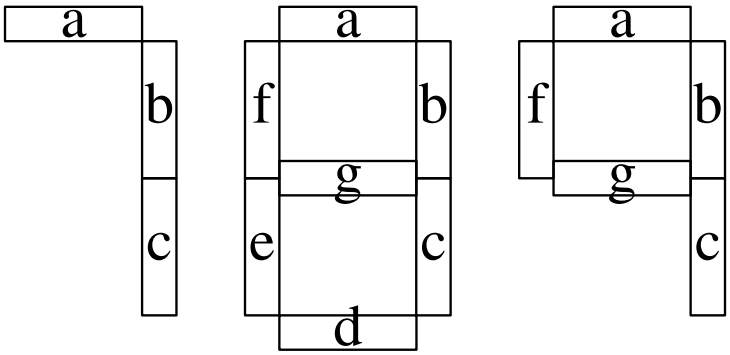
Seven-Segment Decoder

A seven-segment display is used for displaying digits by turning lights on or off.
The lights are arranged in segments such that every digit has a unique combination of segments.




a
f
b
g
e
c
d
a
f
b
e
c
d
b
c
a
b
g
e
d


a
b
g
c
d
f
b
g
c
a
f
g
c
d
f
g
e
c
d


a
b
c
a
f
b
g
e
c
d
a
f
b
g
c

A decoder for a seven-segment display has BCD input code and output code that corresponds to the segments a, b, $\cdots$, g.

Truth table:

digit	A3	A2	A1	A0	Ya	Yb	Yc	Yd	Ye	Yf	Yg
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	0	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1
-	1	0	1	0	x	x	x	x	x	x	x
-	1	0	1	1	x	x	x	x	x	x	x
-	1	1	0	0	x	x	x	x	x	x	x
-	1	1	0	1	x	x	x	x	x	x	x
-	1	1	1	0	x	x	x	x	x	x	x
-	1	1	1	1	x	x	x	x	x	x	x

Y_a:

A_1A_0 \ A_3A_2	00	01	11	10
00	1		x	1
01		1	x	1
11	1	1	x	x
10	1		x	x

$$Y_a = A_3 + A_1 \cdot A_0 + A_2 \cdot A_0 + A_2' \cdot A_0'$$

Binary Encoders

Similar to a decoder, an *encoder* is also a multi-input, multi-output logic circuit that converts coded inputs into coded outputs.

Relative to a decoder, an encoder maps the output code of a decoder to its input code.

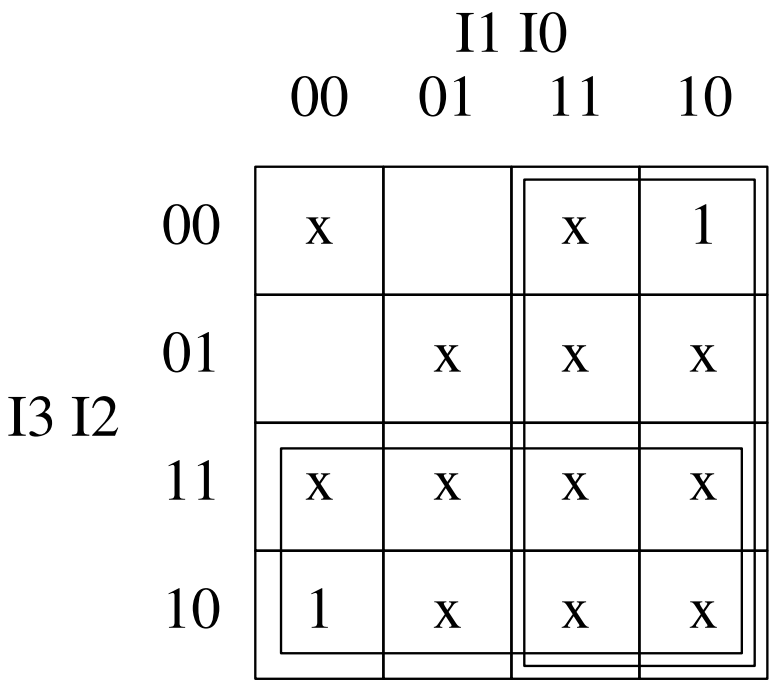
This implies that the output code of an encoder typically has fewer bits than the input code (opposite to a decoder).

The $2^n - to - n$ or *binary encoder* has the opposite function of a binary decoder.
The input code is the 1-out-of-2^n code.
The output code is the n-bit binary code.

A truth table for a 4-to-2 binary encoder:

I3	I2	I1	I0	Y1	Y0
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
0	0	1	1	x	x
0	1	0	1	x	x
0	1	1	0	x	x
0	1	1	1	x	x
1	0	0	1	x	x
1	0	1	0	x	x
1	0	1	1	x	x
1	1	0	0	x	x
1	1	0	1	x	x
1	1	1	0	x	x
1	1	1	1	x	x

A K-map for Y0:



Y0 = I1 + I3

It is possible to write this function without using a K-map by noting that:

Y0=1 for 0010 and Y0=x for any other combination where I1=1.
This yields the term I1.

Y0=1 for 1000 and Y0=x for any other combination where I3=1.
This yields the term I3.

Similar arguments for Y1 yield
Y1 = I2 + I3

The non-x terms for an 8-to-3 encoder:

I7	I6	I5	I4	I3	I2	I1	I0	Y2	Y1	Y0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

$Y0 = I1 + I3 + I5 + I7$

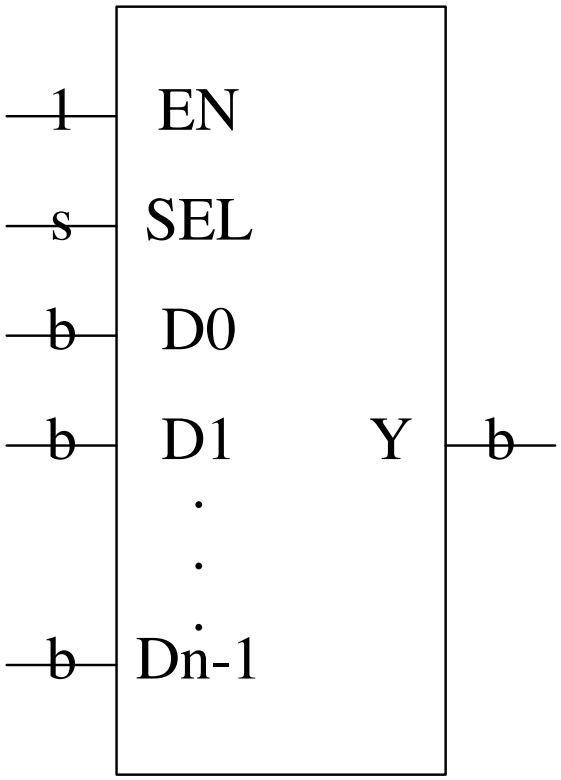
$Y1 = I2 + I3 + I6 + I7$

$Y2 = I4 + I5 + I6 + I7$

6.4. Multiplexers

In many applications it is necessary to select one of several sources of data, and transfer the data from the selected source to a destination.

A *multiplexer* connects data from one of *n* sources to its output.
The data is $b \geq 1$ bits wide.



The multiplexer has:

$n \cdot b$ data inputs.

b outputs.

s select inputs to select a source.
We have that $2^s = n$ or $s = \lceil \log_2 n \rceil$.

An enable input.
When EN=0 all the outputs are 0.

It is convenient to present the truth table of a multiplexer as follows:

EN	SEL	D0	D1	...	Dn-1	Y
0	x	x	x	...	x	0
1	0	d0	x	...	x	d0
1	1	x	d1	...	x	d1
				...		
1	n-1	x	x	...	dn-1	dn-1

A 4-input 1-bit multiplexer:
($b = 1$ and $n = 4$)

EN	S1	S0	D0	D1	D2	D3	Y
0	x	x	x	x	x	x	0
1	0	0	d0	x	x	x	d0
1	0	1	x	d1	x	x	d1
1	1	0	x	x	d2	x	d2
1	1	1	x	x	x	d3	d3

$$\begin{aligned} Y = EN \cdot (&S1' \cdot S0' \cdot D0 + \\ &+S1' \cdot S0 \cdot D1 + \\ &+S1 \cdot S0' \cdot D2 + \\ &+S1 \cdot S0 \cdot D3). \end{aligned}$$

An 8-input 1-bit multiplexer:
($b = 1$ and $n = 8$)

EN	S2	S1	S0	Y
0	x	x	x	0
1	0	0	0	d0
1	0	0	1	d1
1	0	1	0	d2
1	0	1	1	d3
1	1	0	0	d4
1	1	0	1	d5
1	1	1	0	d6
1	1	1	1	d7

$$
\begin{aligned}
Y = EN \cdot (&S2' \cdot S1' \cdot S0' \cdot D0 + \\
&+S2' \cdot S1' \cdot S0 \cdot D1 + \\
&+S2' \cdot S1 \cdot S0' \cdot D2 + \\
&+S2' \cdot S1 \cdot S0 \cdot D3 +. \\
&+S2 \cdot S1' \cdot S0' \cdot D4 + \\
&+S2 \cdot S1' \cdot S0 \cdot D5 + \\
&+S2 \cdot S1 \cdot S0' \cdot D6 + \\
&+S2 \cdot S1 \cdot S0 \cdot D7).
\end{aligned}
$$

When the data values are not independent, or when the data contains constants, the function of a multiplexer can be simplified.

When the function is sufficiently simplified, instead of using a multiplexer it may be cheaper to implement a minimal sum or minimal product.

Example 1:

With $b = 1$ and $n = 4$, it is known that

D2 = D0 and D3 = D1.

We obtain

$$
\begin{aligned}
Y = {} & S1' \cdot S0' \cdot D0 + S1' \cdot S0 \cdot D1 + \\
& + S1 \cdot S0' \cdot D2 + S1 \cdot S0 \cdot D3 = \\
& = S1' \cdot S0' \cdot D0 + S1' \cdot S0 \cdot D1 + \\
& + S1 \cdot S0' \cdot D0 + S1 \cdot S0 \cdot D1 = \\
& = (S1' \cdot S0' + S1 \cdot S0') \cdot D0 + \\
& + (S1' \cdot S0 + S1 \cdot S0) \cdot D1 = \\
& = S0' \cdot D0 + S0 \cdot D1
\end{aligned}
$$

We obtained a 2-input 1-bit multiplexer.

Example 2:

With $b = 1$ and $n = 4$, it is known that

D0 = 0 and D1 = D2 = D3 = 1.

We obtain

$$
\begin{aligned}
Y &= S1' \cdot S0' \cdot D0 + S1' \cdot S0 \cdot D1 + \\
&\quad + S1 \cdot S0' \cdot D2 + S1 \cdot S0 \cdot D3 = \\
&\quad = S1' \cdot S0' \cdot 0 + S1' \cdot S0 \cdot 1 + \\
&\quad + S1 \cdot S0' \cdot 1 + S1 \cdot S0 \cdot 1 = \\
&\quad = S1' \cdot S0 + S1 \cdot S0' + S1 \cdot S0 = \\
&\quad = S1' \cdot S0 + S1 \cdot (S0' + S0) = \\
&\quad = S1' \cdot S0 + S1 = \\
&\quad = S0 + S1
\end{aligned}
$$

It is possible to implement any n-input function with a 2^n-input 1-bit multiplexer.

Example: $F(A, B) = A + B$

$F(A, B) = A + B = A \cdot B + A \cdot B' + A' \cdot B$

Use:

$S_1 = B$

$S_0 = A$

$D_0 = 0$

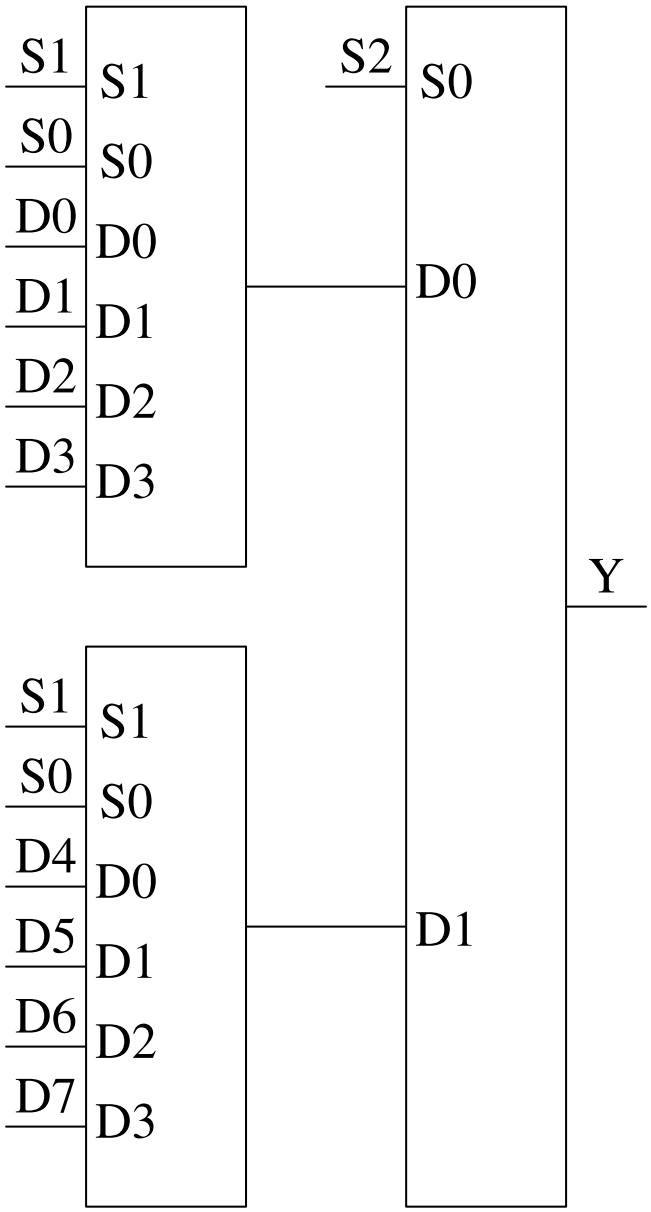
$D_1 = 1$

$D_2 = 1$

$D_3 = 1$

Expanding Multiplexers

Constructing a 1-bit, 8-input multiplexer from smaller multiplexers:



A *demultiplexer* has a function opposite to that of a multiplexer.
It transfers data from its input to one of *m* outputs.
The output is indicated by the select inputs.

A demultiplexer can be implemented using a binary decoder:
The address lines determine which output is selected.
The data is connected to the enable input.

Truth table of a 2-to-4 binary demultiplexer:

A1	A0	Y3	Y2	Y1	Y0
0	0	0	0	0	D
0	1	0	0	D	0
1	0	0	D	0	0
1	1	D	0	0	0

Truth table of a 2-to-4 binary demultiplexer:

D	A1	A0	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

Truth table of a 2-to-4 binary decoder:

EN	A1	A0	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