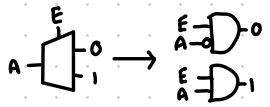
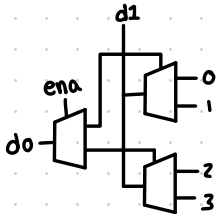
Decoders

remember:



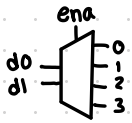
2x4 Decoder Cascade



d0	d1	ena	0	1	2	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

~d0

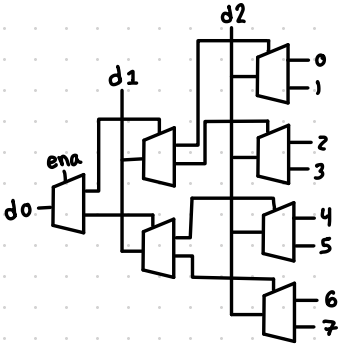
d0



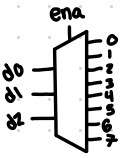
each output maps to a state (one-hot encoding)

↳ states will cycle through

3x8 Decoder Cascade

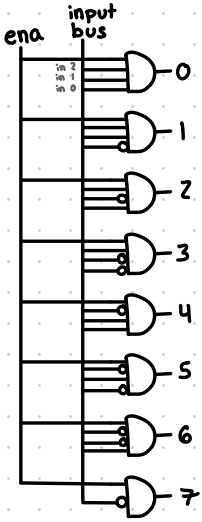


d0	d1	d2	ena	0	1	2	3	4	5	6	7
X	X	X	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0	0	0	0
0	0	1	1	0	1	0	0	0	0	0	0
0	1	0	1	0	0	1	0	0	0	0	0
0	1	1	1	0	0	0	1	0	0	0	0
1	0	0	1	0	0	0	0	1	0	0	0
1	0	1	1	0	0	0	0	0	1	0	0
1	1	0	1	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	1

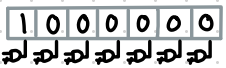


You can use AND gates to do the same logic - use the same logic table

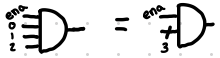
(we wanted to try this instead since it's more explicit to describe in verilog)



each AND gate controls 1 bit



All and gates have 3 inputs and enable

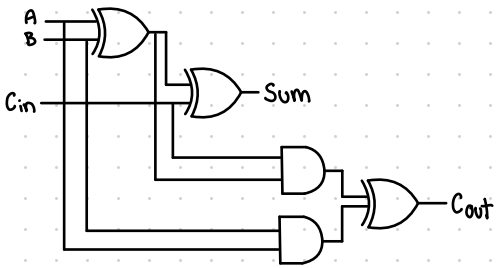


→ Verilog: `out[7] = ena & inb[1] & inb[2] & inb[3]`

Adder

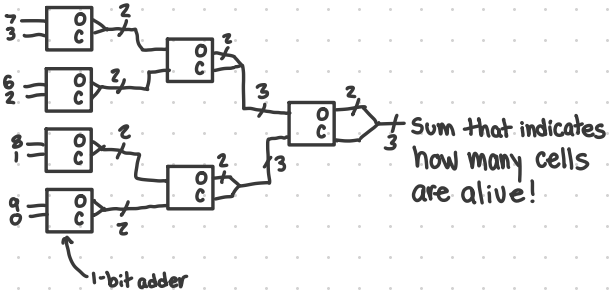
(1 bit full adder)

A	B	Cin	Cout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



Summer

input 8-bit bus from main
↳split into 4 2 input adders



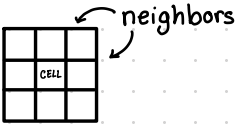
Conway's Game of Life Cell Module

Rules:

1) Any cell w/ 2-3 neighbors survive

2) Any dead cell w/ 3 neighbors is born

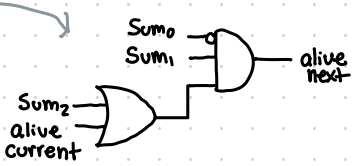
3) All other live cells die in the next generation, dead cells stay dead



↳ represented by 8-bit bus supplied by main

neighbour count	Sum out 0	Sum out 1	Sum out 2	alive current	alive next
0	0	0	0	0	0
1	0	0	1	0	0
2	0	1	0	0	0
3	0	1	1	0	1
4-7	1	X	X	0	0
0	0	0	0	1	0
1	0	0	1	1	0
2	0	1	0	1	0
3	0	1	1	1	1
4-7	1	X	X	1	0

& logic



LED DRIVER

note:
rows need to go low to light up the LED