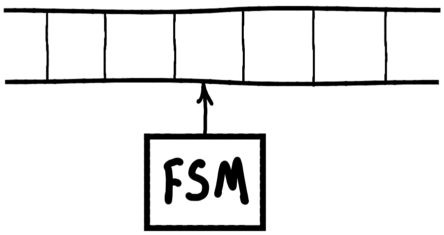
Finite State Machine Implementation



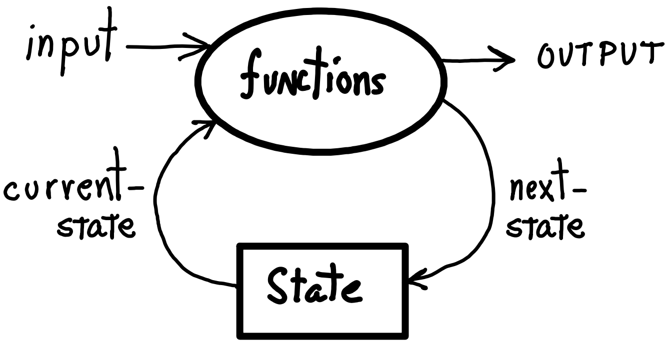
Build TMs ===> Build FSMs

Symbol sets {a, b, c, d} ===> sets of bit strings { 00, 01, 10, 11}

State sets {state-W, state-X, state-Y, state-Z} ===> {00, 01, 10, 11}

We need:

State Elements
Boolean Functions (Next-State, Output)



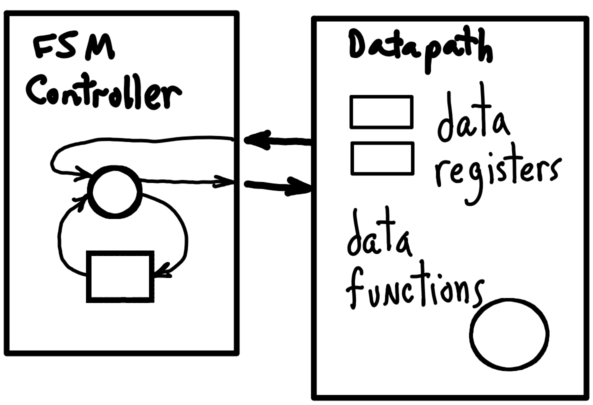
FSM has input/output, but from/to where?

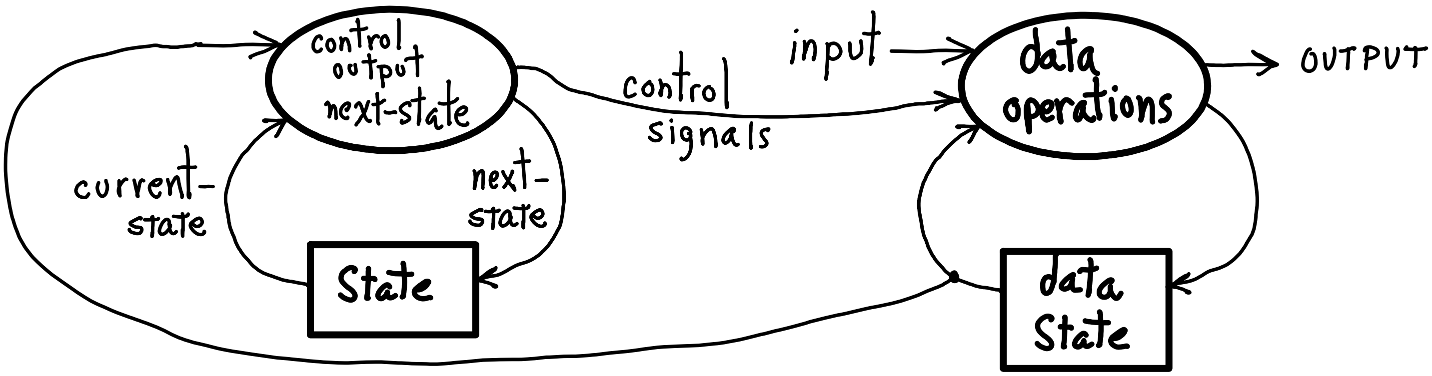
(1) Other FSMs

(2) Feedback loops

"Computers": Mentally Split State

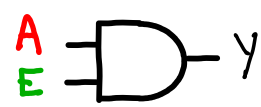
Control STATE	**and**	**Data STATE**
"control" state elements		**"data" state elements.**
next-state function		**data operation functions**





State Elements

gating a signal

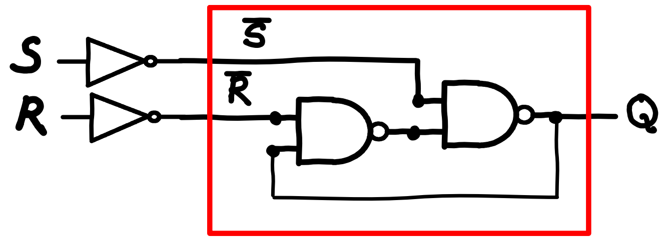


E = 1 : Y = A
E = 0 : Y = 0

BUILD a State element that does not cause feedback problems: D-FF

We have an S-R latch:

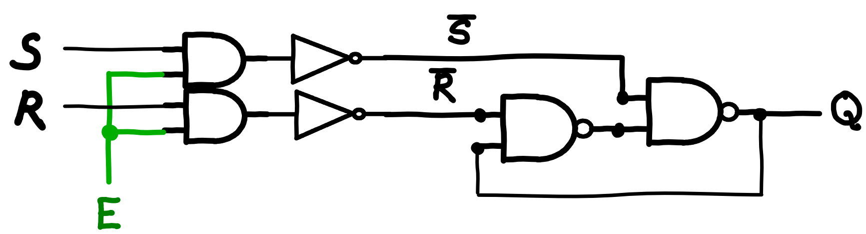
(S, R) == (0, 0) : Q is stable
(S, R) == (1, 0) : Q == 1
(S, R) == (0, 1) : Q == 0



We need to isolate its inputs:
GATING

E == 0 : Q is stable

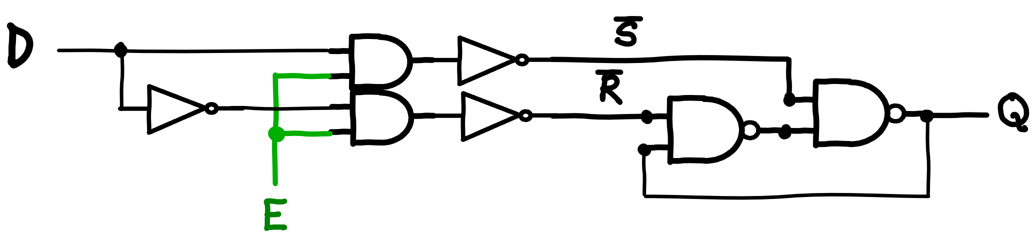
E == 1: Q can change



1-bit input:

E == 0: Q = D
E == 1: Q is stable

"D-latch w/ enable"

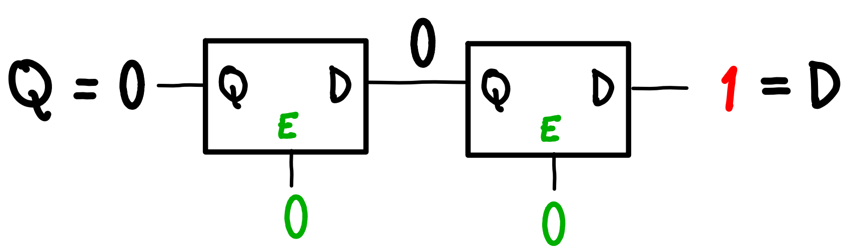


2-phase clocking

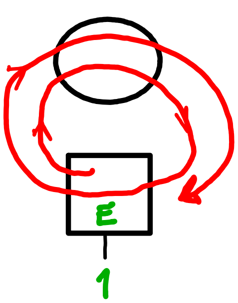
We need to prevent feedback from changing FSM state until we are ready.

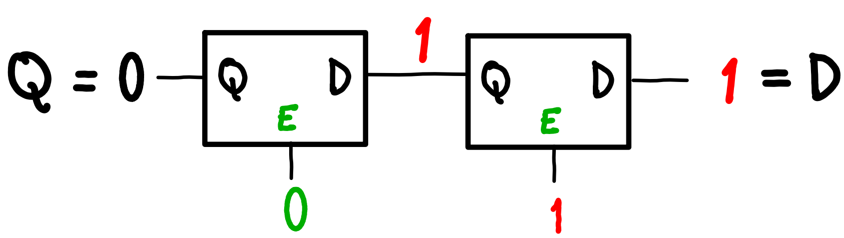
A D-latch acts like a wire when E == 1.

State changes continually w/o control.



Current state Q == 0. Next state = 1. Latches stable.



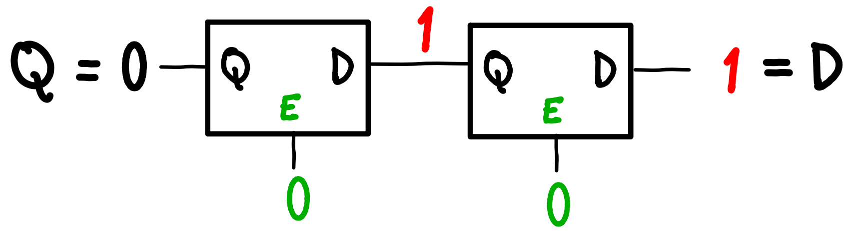


Current state Q == 0. Right latch propagates 1.

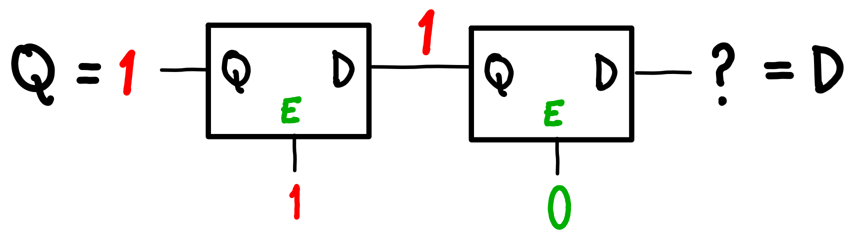
2-Phase Clocking

Independent signals for each latch's enable.

On breadboard:
connect Es to different switches.



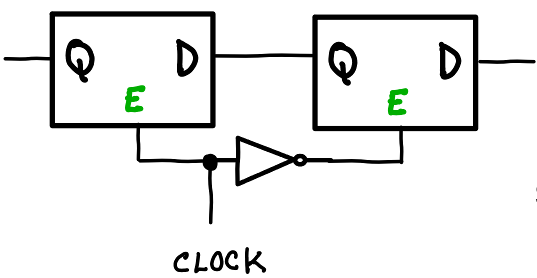
Current state Q == 0. Right latch stable.



Current state Q == 1. Left latch propagates 1.
Feedback may change D, but right latch is stable.

State Q changed when left E when from 0 to 1.

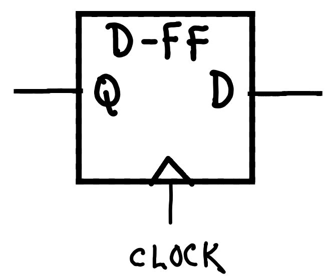
implement 2-phase clock



Positive Edge-Triggered D FlipFlop

State Q changes when clock makes 0-1 transition

Next state sampled on clock 1-0 transition

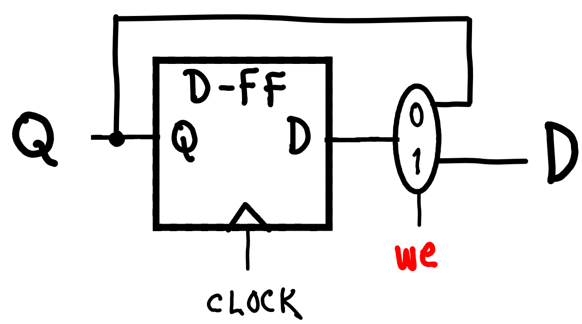


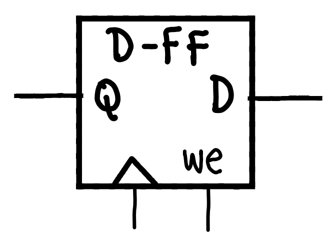
Control if D-FF will be written.

Add a write-enable.

we == 1: D propagates to input state changes

we == 0: Q propagates to input state is stable

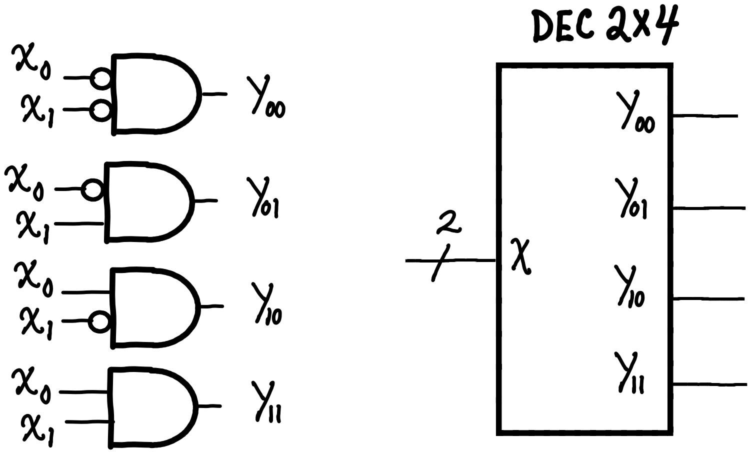




NO FEEDBACK path from Q to D?

Can use D-latch instead.

implementing functions: DEC, PLA, MUX, ROM



k-bit input DECODER

Generates all 2^k minterms.

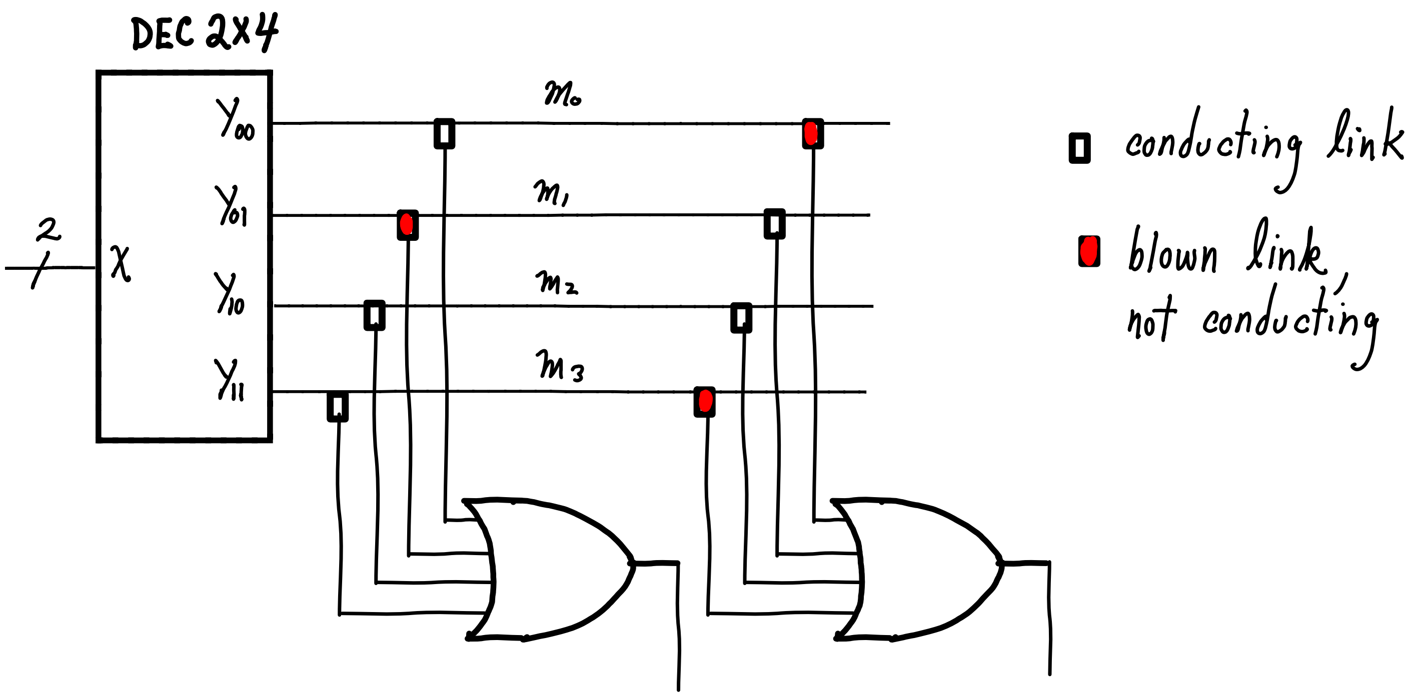
Exactly one ouput == 1

PLA (Programmable Logic Array): OR'ing minterms.

Share logic to implement multiple functions.

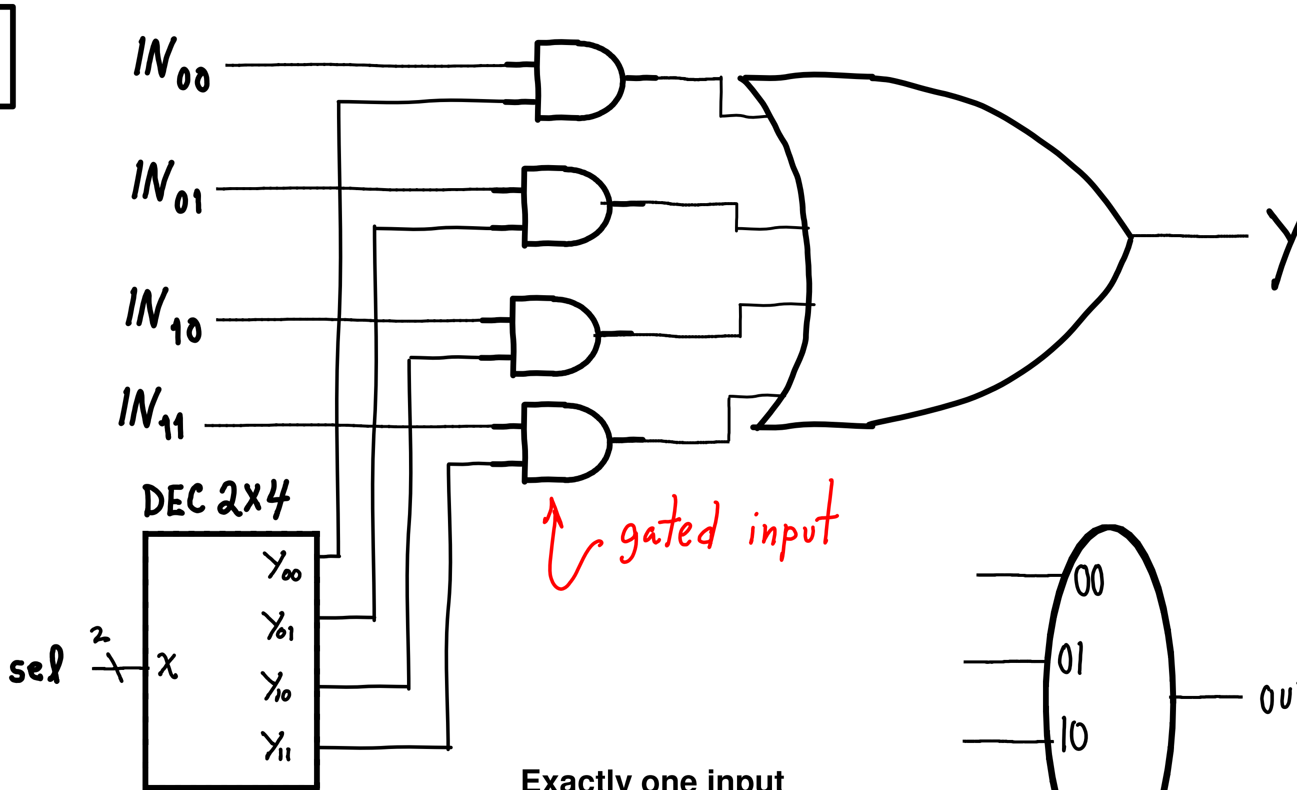
Programmable:

minterm lines can be "blown" to disconnect them:
select function's minterms.

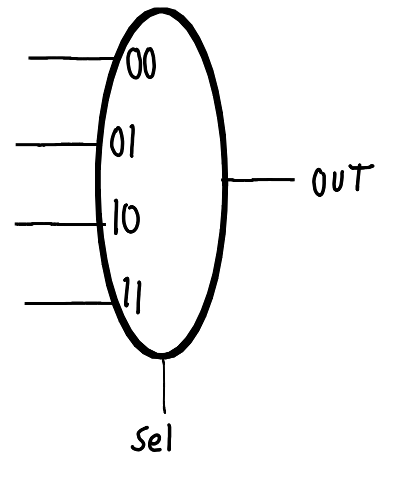


$f = m_0 + m_2 + m_3$ $\quad$ $g = m_1 + m_2$

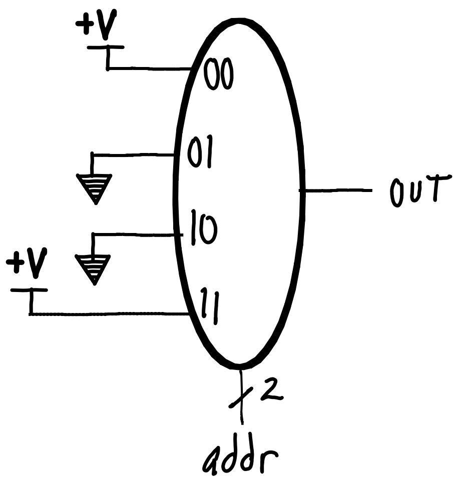
MUX



Exactly one input steered to output

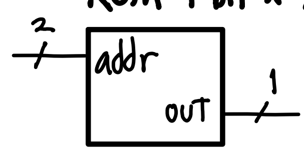


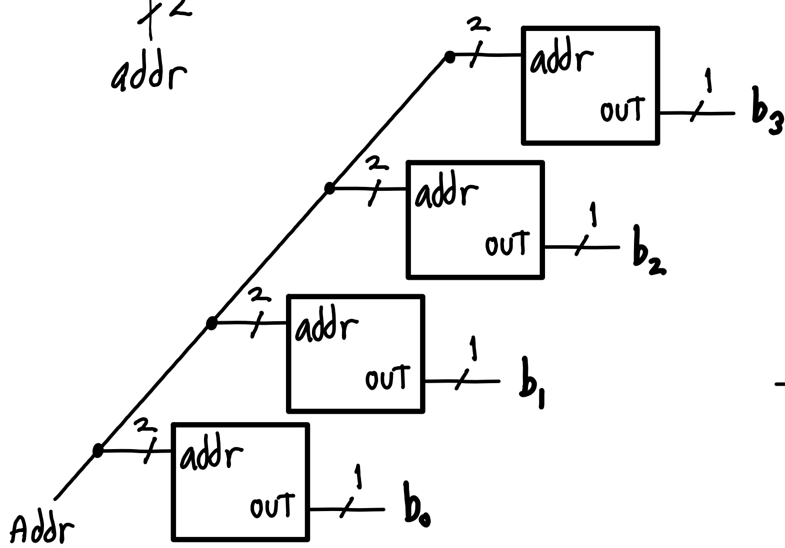
ROM



addr	content
00	1
01	0
10	0
11	1

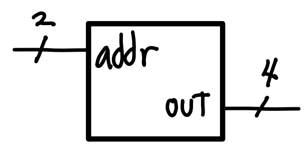
ROM 1 bit x 4 words





ROM

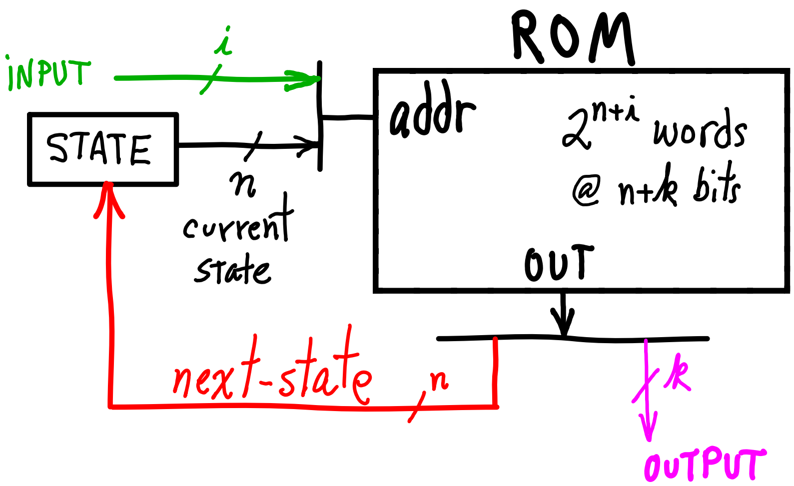
4-bit words
4 words



addr	b_3	b_2	b_1	b_0
00	1	1	1	1
01	0	0	0	0
10	0	1	1	0
11	1	0	0	1

0

FSM = ROM + STATE



FSM functions as ROM

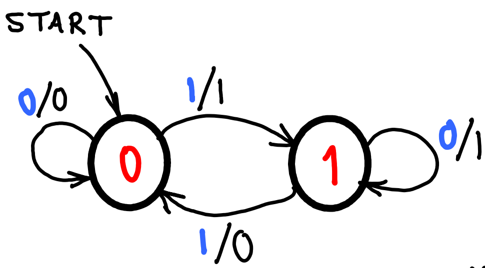
i input bits (2^i different inputs)

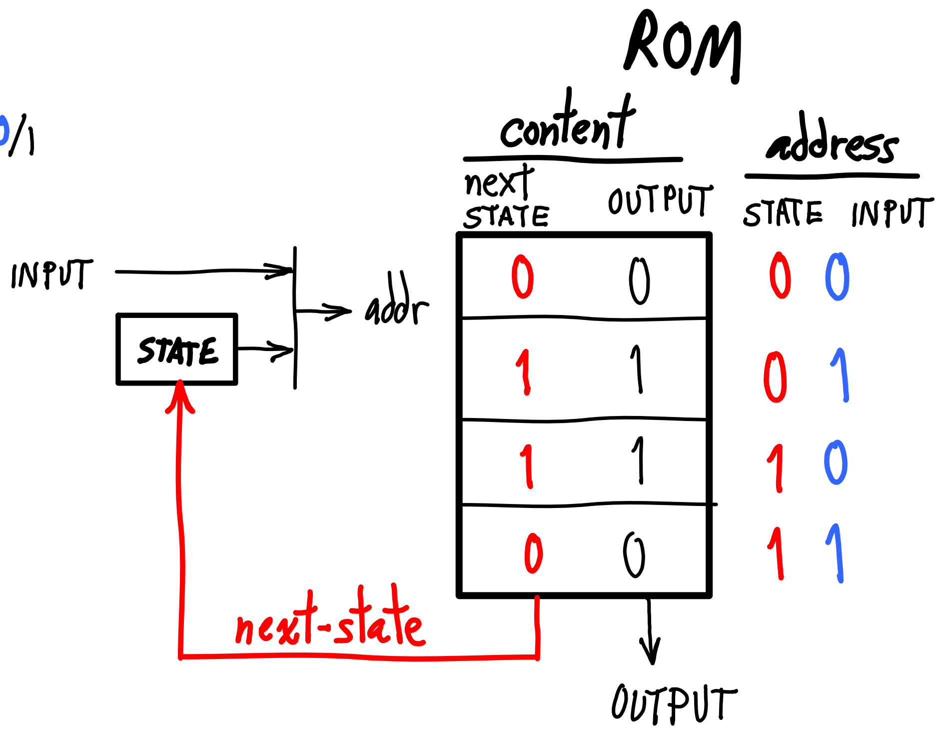
n state bits (2^n different states)

(2^i) X (2^n) combinations
===> 2^(i+n) words in ROM
i+n address bits

ROM word == (next-state, output)

output for two functions





ANY FSM can be built:

ROM + STATE

One ROM word per
(STATE, INPUT) combination

BUT, very big?

SMALLER?

One word per state (Moore Machine)?
Next-state function outside of ROM?

We can list all ROMs ===> list all FSMs ==> all TMs

ROM

address	content
00	00
01	11
10	11
11	00

Concatenate ROM content: ==> 00111100

Each is an integer (make unique, no leading 0s)