midterm notes

- $x + x'y = x + y$
- dual of boolean property: $\wedge \to \vee$, $\vee \to \wedge$, $1 \to 0$, $0 \to 1$ ex: $xy \to x+y$
- demorgan = $(a+b'+c)(d+e) = a'bc + d'e'$
- xor (exclusive or) $00 \to 0$, $01 \to 1$
- min/max terms start with zero $m(0)$
- $m(4) \to a\,b'\,c' \to 4$ → to give 1 (a b c : 4)
- $M(6) \to a'+b'+c \to 6$ → to give 0 (1 1 0 : 6), max 0 → $a+b+c$ (0,0,0), min 0 → abc (1,1,1)
- $[\min(a)]' = \max(a)$

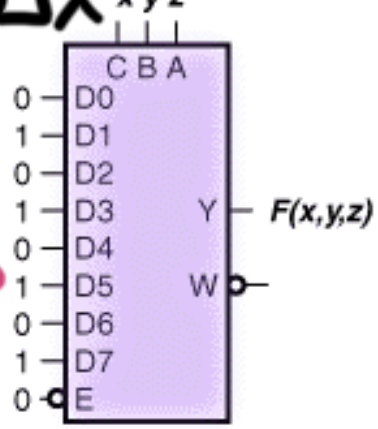
$x \oplus 1 = x'$
$x \oplus 0 = x$

latches → memory

An 8-line to 1-line multiplexer is connected as shown, where output Y = F(x,y,z) and z is the least significant input. Which of the following functions does Y generate?

1. F(x,y,z) = z
2. F(x,y,z) = y
3. F(x,y,z) = z'
4. F(x,y,z) = x
5. F(x,y,z) = x + y'

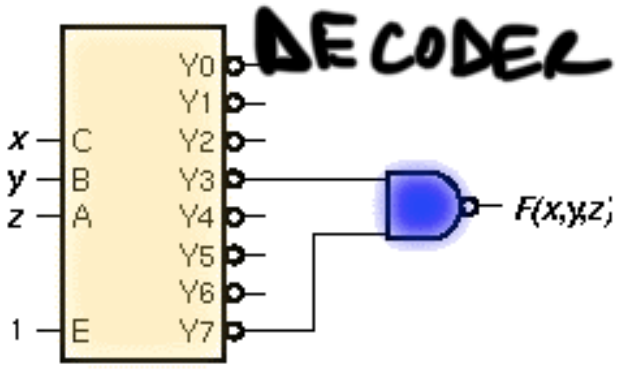
MUX



n → selectors

2^n inputs → 1 output

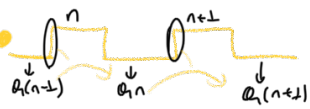
A 3-line to 8-line decoder is connected as shown. Where x, y and z are inputs (z is the least significant input digit) and F is an output. Which of the following expressions correctly describes F?



1. F(x,y,z) = z
2. F(x,y,z) = x
3. F(x,y,z) = z'
4. F(x,y,z) = yz
5. F(x,y,z) = x'

n input → 2^n output

- JK → q/0/1/q' D → d T → q / q'
- when to stop cycle = clock (with) / how = flip-flops made of latches



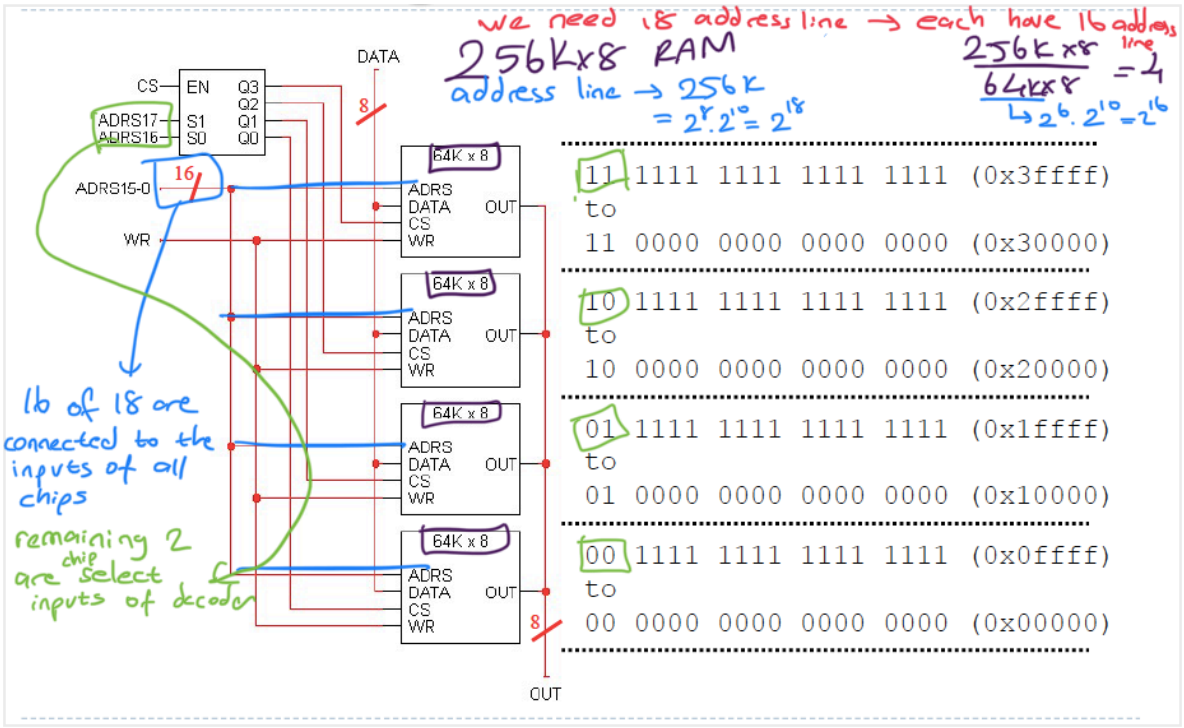
- latches are asynchronous sequential circuits (no clock)
- ripple counter = bit changes when previous bit changes 0111 → 0110 → 0100 → 0000 → 1000 (3 → 8)
- when a variable is connected to clock input we have to see positive edge so it must go from 0 to 1 (then that flip-flop can operate on its own input)

toggle = complement

ram chip
8K × 32 → address lines = 2^{13} ($3 \cdot 10$) 13
data lines = 32

K → 2^{10}
M → 2^{20}
G → 2^{30}

128 × 8 RAM → is in bits = 128 bytes
(0b)


we need 18 address line → each have 16 address line
256Kx8 RAM
address line → 256K
= 2^8.2^10 = 2^18
256Kx8 / 64Kx8 = 4
↳ 2^6.2^10 = 2^16
DATA
CS
EN
Q3
Q2
Q1
Q0
S1
S0
ADRS17
ADRS16
ADRS15-0
16
WR
8
64K x 8
ADRS
DATA
CS
WR
OUT
16 of 18 are connected to the inputs of all chips
remaining 2 are chip select inputs of decoder
11 1111 1111 1111 1111 (0x3ffff)
to
11 0000 0000 0000 0000 (0x30000)
10 1111 1111 1111 1111 (0x2ffff)
to
10 0000 0000 0000 0000 (0x20000)
01 1111 1111 1111 1111 (0x1ffff)
to
01 0000 0000 0000 0000 (0x10000)
00 1111 1111 1111 1111 (0x0ffff)
to
00 0000 0000 0000 0000 (0x00000)

rom size

inputs → k = address lines → $2^k \times n$
outputs → n = data lines

(and decoder)
ram = bunch of registers, time to access are same
↳ clock = dont have own clock, uses CPU's (memory unit)

access time = select and read
cycle time = complete writing

is used for a computers main memory

static ram (SRAM) vs dynamic ram (DRAM)

faster with caches

- fast, large in size, consume more power
- cost more, low capacity
- made up flip-flops (voltage) → using latches instead (faster, cheaper, but timing is problem)
- data remains valid as long as power is supplied

- capacitors (charge) data storage
- constan refreshing

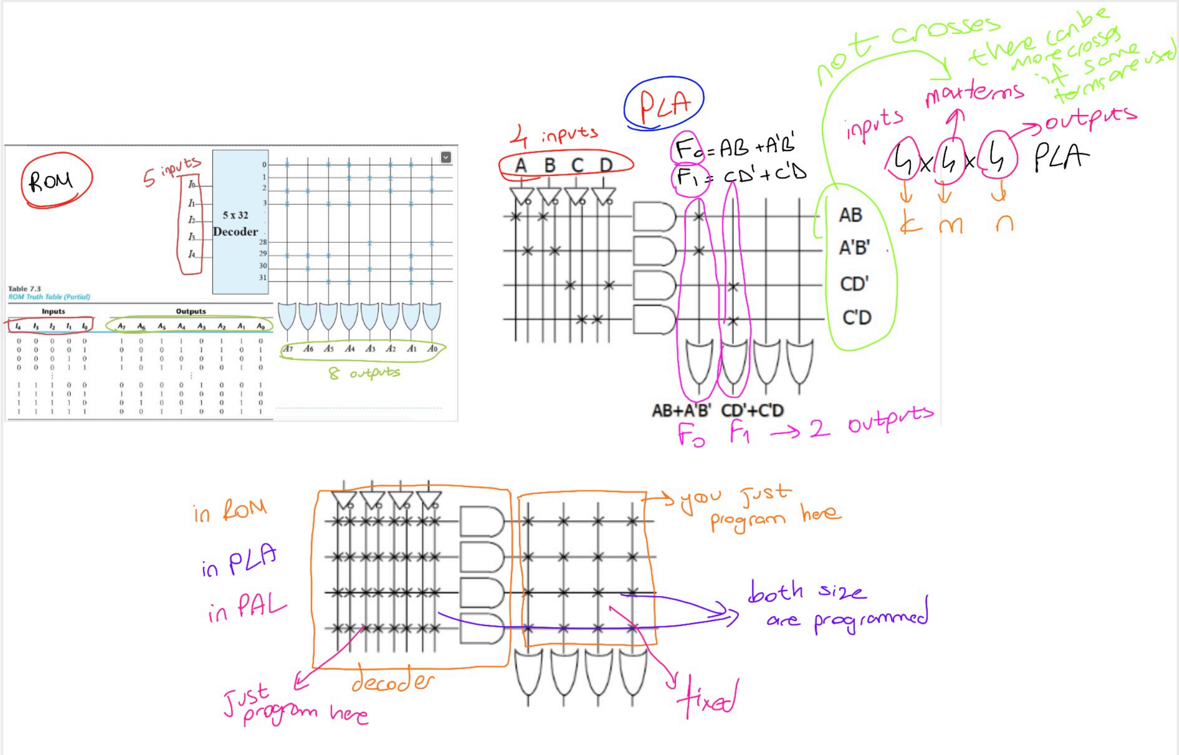
dynamic wit sma stati per. mem

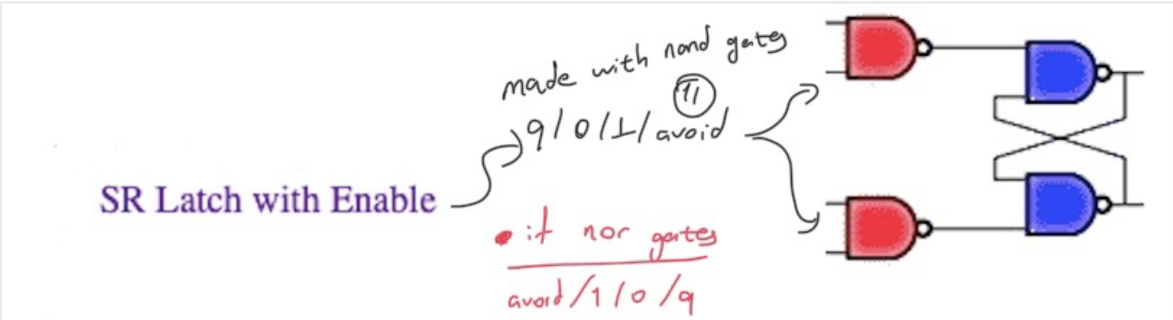
programmable logic devices

1. ROM → just a truth table + decoders + or gates → it is programmable bc of the different connections btw the decoder and or gates
   ↳ data is preserved even without power
   ↳ decoder generates all minterms, → inefficient
   inputs → decoder (fixed) → or array (programmable) → output
2. PLA → makes the decoder part programmable too (less genral / limited)
   ↳ shared product terms (to reduce the fan-in = # of inputs)
   inputs → and array (programmable) → or array (programmable) → outp
3. PAL → decoder part is programmable, or gates are not (they are fixed)
   ↳ so, limited but cheaper (should be used to program basic logic functions)
   inputs → and array (programmable) → or array (fixed) → outp

sequential programmable devices = programmable combinational logic + memory

1. FPGA (field-programmable gate array)
2. SPLD
   PAL/PLA + flip flops
3. CPLD
   several PLDs (complex)

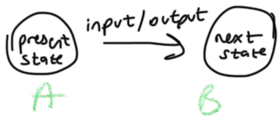




☆ two flip-flops means 2 bit memory

state diagram → n flip flops → 2^n nodes

m inputs → 2^m each nodes' outgoing arrows



FSM (finite-state machine) models:

1. mealy machine = current state + external input ⇒ output
2. moore machine = (just) current state ⇒ output

before assigning stages (A,B,C) to numbers, decide of the # flip fl…

how many flip-flops are needed? →
1 FF → 0/1 : 2 state
2 FF → 00/01/10/11 : 4 state
3 FF → 000/001 … /111 : 8 state
} $\log_2 n$

characteristic tables

J	K	$Q_t(t+1)$	operation
0	0	$Q_t(t)$	no change
0	1	0	reset
1	0	1	set
1	1	$Q_t'(t)$	complement

they show what ff inputs need for desired change

excitation tables

$Q_t(t)$	$Q_t(t+1)$	J	K	Operation
0	0	0	x	no change / reset
0	1	1	x	set / complement
1	0	x	1	reset / complement
1	1	x	0	no change / set

D	$Q_t(t+1)$	operation
0	0	reset
1	1	set

$Q_t(t)$	$Q_t(t+1)$	D	Operation
0	0	0	reset
0	1	1	set
1	0	0	reset
1	1	1	set

T	$Q_t(t+1)$	operation
0	$Q_t(t)$	no change
1	$Q_t'(t)$	complement

$Q_t(t)$	$Q_t(t+1)$	T	Operation
0	0	0	no change
0	1	1	complement
1	0	1	complement
1	1	0	no change

* you can reduce the # of states (if next states, inputs, and outputs are same present states)

register = group of flip-flops
each stores one bit

↳ parallel r. = all ffs are loaded in a single clock cycle

serial (shift) r. = shifts its binary information in a direction → keyboards, mouse, printers, usb

load 1100 to 1101 at 2t : 0011

serial → parallel: load after t, reads at once

parallel → serial: read at once, sends after t

universal shift register = capable of shifting right and left, and loading parallel

parallel adder = n adders, one unit of time, combinational circuit

serial adder = 1 full adder, n unit of time, sequential circuit

half adder = input A + input B → output Sum + Output Carry (add 2 one bit)

full adder = carry C + input A + input B → output Sum + Output Carry (add 3 o

counters = register which counts

① ripple c. = the ff's output triggers other ff (asynchronous) (no clock)

② synchronous c. = count the clock

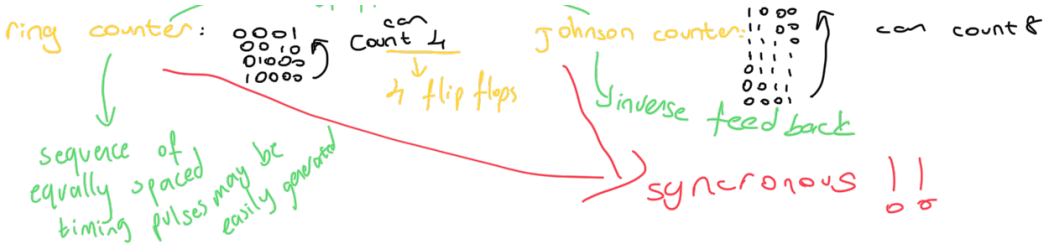
binary ripple counter = 0000 / 0001 / .... / 1111 : the bits change when previous bit goes from 0 to 1

synchronous binary counter = ff is complemented if all lower bits are 1 (the right ones)

↳ to add initial value, we can add parallel load feature

same # of ffs

0000



* if reset input is required you can add it as normal input:

R	Q_1	Q_0	$\rightarrow$	$Q_{1\,next}$	$Q_{2\,next}$
1	-	-		0	0
0	↑	↑		⋮	⋮

binary coded decimal:

0 → 0000
1 → 0001
⋮
9 → 1001

10 → 0001 0000
11 → 0001 0001
⋮

ring counter = overback counter → changes 2 bit a time

Johson → 1

hamming distance → 1 0 0 1 0 1 1 / 0 1 0 1 0 0 1 → 3 (different bits)

the transition back to the same state → means self loot