MultiOutput

Tri-State: aka three-state, refers to a type of circuit that can exist in three difference states

Disadvantages: need to avoid multiple drivers driving at the same time

E	A	Y
0	0	z
1	0	1
1	1	0

Find a minimum two-level sum of products solution for the following equation:

id	A	B	C	X	Y	Z
0	0	0	0	0	1	0
1	0	0	1	1	1	x
2	0	1	0	x	0	0
3	0	1	1	0	0	0
4	1	0	0	0	1	0
5	1	0	1	1	x	1
6	1	1	0	x	0	1
7	1	1	1	0	0	x

Find out the SOP with minimal gates for X, Y, and Z (ex. X = AB + CD + EF)

Minimal SOP of X will be $B'C$.
The mterm of X are $5, 1$.
The dterm of X are $2, 6$.

Minimal SOP of Y will be B'.
The mterm of Y are $4, 0, 1$.
The dterm of Y are 5.

Minimal SOP of Z will be $AB + AC$.
The mterm of Z are $5, 6$.
The dterm of Z are $7, 1$.

The minimal gates required is 4.
We need gates of each term AC,B'C,AB.
And extra function gate of 1.

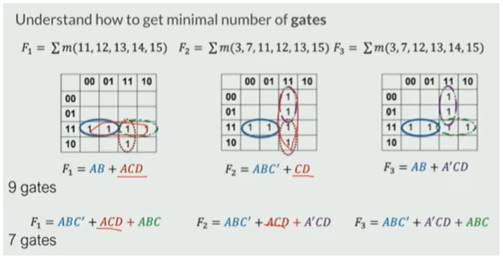
[Fanin: number of inputs on a gate**]** ↑ fanin = ↑ gate complexity, ↓ operation speed, ↑ input capacitance, ↑ power consumption, ↑ delay

[Fanout: number of loads that a gate drives (provides signal to other gates); measures loading capacity**]** ↑ fanout = ↑ load capacitance, ↓ operation speed, ↑ power consumption, ↑ delay

Each gate has a fixed delay, called **propagation delay (tpd)** ⇒ two components: **intrinsic delay** (inherent delay of the gate itself) and **loaded delay** (depends on the load/inputs that the gate is driving)

Load = Capacitance (simple model)

MultiOutput Minimization: process used to reduce the complexity of a circuit


Understand how to get minimal number of gates

$F_1 = \sum m(11,12,13,14,15)$ $F_2 = \sum m(3,7,11,12,13,15)$ $F_3 = \sum m(3,7,12,13,14,15)$

$F_1 = AB + ACD$ $F_2 = ABC' + CD$ $F_3 = AB + A'CD$

9 gates

$F_1 = ABC' + ACD + ABC$ $F_2 = ABC' + ACD + A'CD$ $F_3 = ABC' + A'CD + ABC$

7 gates

Tips: find similar implicates so you can reuse, remember that gates can have more than 2 inputs

Quine McCluskey: apply uniting theorem (X'Y + XY = Y) over and over again to reduce terms

Let $f(e,f,g,h) = \sum m(2,3,6,10,11,13,14)$

minterms	binary	matched
2	0010	V
3	0011	V
6	0110	V
10	1010	V
11	1011	V
13	1101	X
14	1110	V
–	–	–
–	–	–

minterms	binary	matched
2,10	-010	V
2,6	0-10	V
2,3	001-	V
3,11	-011	V
6,14	-110	V
10,14	1-10	V
10,11	101-	V
–	–	–
–	–	–

minterms	binary	matched
2,6,10,14	--10	X
2,3,10,11	-01-	X
–	–	–

implicant	minterms involved	2	3	6	10	11	13	14
f'g ✓100%	2,3,10,11 ✓100%	x	x		x	x		
gh' ✓100%	2,6,10,14 ✓100%	x		x	x			x
efg'h ✓100%	13 ✓100%						x	
✓100%	✓100%							
✓100%	✓100%							

minimal SOP = f'g+gh'+efg'h ✓100%

Combinatorial Timing

Longest Delay (Critical Delay Path): max time from input to output

- **Critical Path Input:** first input in this path

Shortest Delay: min time from input to output

Sequential Circuits:

Combinatorial Circuit: output depends on inputs

Sequential Circuit: output depends on inputs and current state

Sequential vs. Combinatorial Logicintr

Sequential Logic: output depends on current input and past inputs

- May contain storage elements like flip-flops or latches, and feedback, more complex
- State diagram, state and excitation tables, characteristic equation

Combinatorial Logic: output is a function of current input states

- No storage element or feedback
- Truth tables, boolean expressions, no feedback

Memory Storage Mechanism

Can introduce feedback loop, where output affects input

There are no feedback loops in combinatorial logic

When circuit power is applied, circuit is metastable until it settles on one of the two stables states

To read a bit value in memory, you measure voltage difference between the two inputs

To write a value, you just force the values in the two inputs

Basic Memory

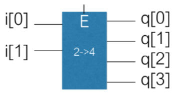
RAM: Random Access Memory, Memory can be quickly accessed, Most RAM is volatile memory, Bit values are lost if electrical power is removed

SRAM: Static Random Access Memory, Typically uses 6 transistors to store each bit, Stores single bit, Maintains data as long as power is supplied, Fast access, more expensive than DRAM, consumers more power and space

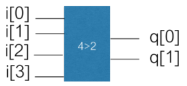
DRAM: Dynamic Random Access Memory, Typically uses 1 transistor and 1 capacitor to store each bit value, Each cell stores one bit of data, Requires periodic refresh cycles to maintain data → "volatile", Slow access, cheaper

Row Decoders: select the specific row of cells to be accessed during read/write operations, Decode the address give by CPU and activate the appropriate row, Helps improve overall speed of memory operation, To set a specific value, convert to binary (#0's = #NOTS) #terms - 1 = #2-input ANDS

E	i1	i0	o0	o1	o2	o2
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

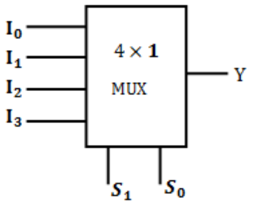


i0	i1	i2	i3	o1	o0
0	0	0	0	x	x
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1



Column Mux: select the specific column during data read/write operations, Helps isolate specific cell or group of cells for data transfer

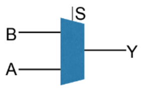
Mux:

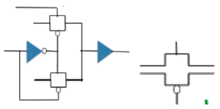


Truth table

S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

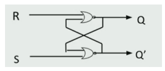
S	A	B	Y
0	0	?	0
0	1	?	1
1	?	0	0
1	?	1	1





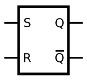
ROM: Read Only Memorym Given m-bit row and n-bit column, total number of cells = $2^m \times 2^n \times k$ if k exists

Latches



SR Latch: basic flip-flop

- Stores a single bit and has two stable states
- Two inputs: set and reset
- Two outputs: Q and Q'
- Reset = 1 → Q = 1, Q' = 1
- Set = 1 → Q = 1, Q' = 0
- S = 0, R = 0 means storing memory
- S = 1, R = 1: invalid state
    - If both pressed then release, Q will oscillate and become metastable and settle in real-life
- SR Latch **characteristic equation:** Q(t + ▵) = S + R'Q(t)
- A latch will store (remember) the value of the D input when the clock first goes high

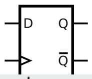


S	R	Q(t)		
0	0	0	0	hold
0	0	1	1	
0	1	0	0	reset
0	1	1	0	
1	0	0	1	set
1	0	1	1	
1	1	0	x	n/a
1	1	1	x	

Clock Signal

Frequency = 1/Period

D Latch: flip flop, built using SR latch



- Stores binary data
- Two states
- Level-triggered
- Two inputs: input D and clock/enable
- When the clock is high, the latch is transparent and follows D
- When the clock is low, the latch will keep whatever the last D value was
- D Latch **characteristic equation:** Q(t + 1) = D(t)

Clk	D	D'	S	R	Q	Q'
0	X	X'	0	0	Q_{prev}	Q'_{prev}
1	0	1	0	1	0	1
1	1	0	1	0	1	0

Edge Triggered Flip Flop:

- Stores whatever value is first seen (before) on every rise/fall of the clock
- Stores for one cycle and checks for new value every high/low

Latch vs Flip Flop:

- Latch can be considered as a door
- Flip flop is a two door entrance

Clock: repeating sequence

- Serves as a timing signal, particularly in sequential logic
- **Frequency:** how fast a signal oscillates
- **Duty Cycle:** ratio of the time the signal is high to the total time of one cycle
    - 50% = high for half cycle and low for other half (most typical instance)
- **Edge Triggering:** responding to state change in the clock rather than its level value

Clock Enabled Flip Flop: flip flop that includes a clock input and enable signal

- **(not enabled) D Flip-Flop:** captures and stores value on D input at the moment when both clock signal triggers and enable is active
- **(not enabled) SR Flip-Flop:** sets or resets the state based on the S and R when both the clock and enable signals are active
- **(enabled) Edge-Triggered Flip-Flops:** react specifically to a clock signal *change*

Flip Flop (aka register): used to store binary data

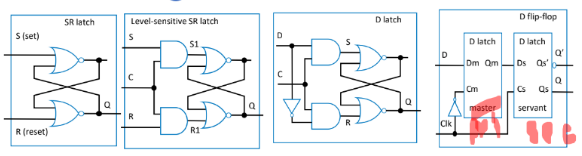
- Registers are used to store data temporarily

Flip-Flops vs. Latches:

- Flip-Flops are edge-triggered, latches are level=triggered
- Flips-flops are more complex

Bit Storage Overview for latches and flip flops

Bit Storage Overview

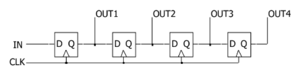


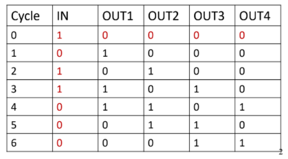
S=1 sets Q to 1, R=1 resets Q to 0. Problem: SR=11 yield undefined Q.	S and R only have effect when C=1. Can design outside circuit so SR=11 never happens when C=1. Problem: avoiding SR=11 can be a burden.	SR can't be 11 if D is stable before and while C=1, and will be 11 for only a brief glitch even if D changes while C=1. *Transition may cross many levels of latches.	Only loads D value present at rising clock edge, so values can't propagate to other flip-flops during same clock cycle. *Transition happens between two level of flip-flops.

Registers

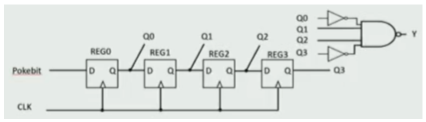
Basic Register: small set of flip-flops grouped together to store binary information
- Each flip-flop stores a bit

Shift Register: holds and shifts samples of input
- It's a sequence of flip-flops connected in a line
- During each clock cycle, data shifts from one flip-flop to the next

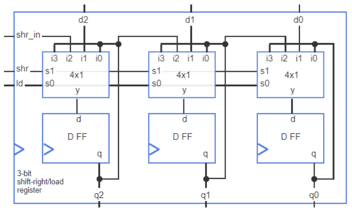




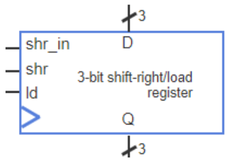
Cycle	IN	OUT1	OUT2	OUT3	OUT4
0	1	0	0	0	0
1	0	1	0	0	0
2	1	0	1	0	0
3	1	1	0	1	0
4	0	1	1	0	1
5	0	0	1	1	0
6	0	0	0	1	1



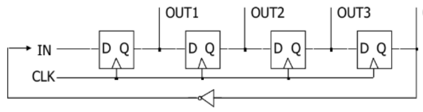
0110 → Y = 1



shr	ld	Register function
0	0	Maintain
0	1	Load
1	0	Shift-right
1	1	Maintain



Counters: count the number of times a particular event/process occurs
- **2-bit counter:** counts in binary from 0 to 3
  - There are two flip-flops that change in state, which represents the count



0000→1000→1100→1110→1111→0111→0011→0001—>0000

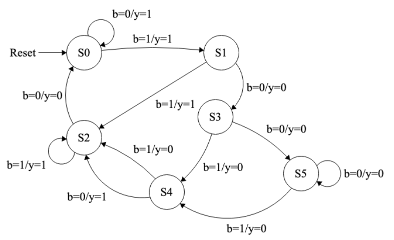
Finite State Machines

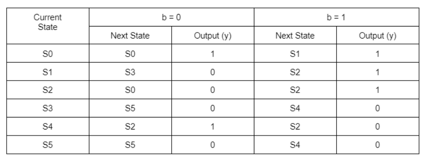
Specification: state tables, state diagram, behavior
- Combinatorial Logic: truth tables, boolean expressions, logic diagram (no feedback)
- Sequential Logic: state diagram, state and excitation tables, characteristic equation, logic diagrams (feedback)

Implementation: excitation table, Mealy/Moore machines, examples
- FSM consists of:
- Set of states
- Set of inputs, set of outputs
- Initial state
- Transitions from one state to another
- State must be well-defined
  - Only one state is valid at a time
  - Exactly one state must be valid at all times

FSM representation:
- State diagram - directed graph. Edges are state transitions
- State table

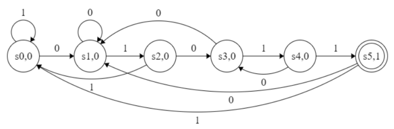
Mealy FSM + Design Process: *remember tables need outputs*
- Output is determined by both current state and inputs
- Mealy responds faster
- Require fewer states
- More complex to implement





Current State	b = 0		b = 1	
	Next State	Output (y)	Next State	Output (y)
S0	S0	1	S1	1
S1	S3	0	S2	1
S2	S0	0	S2	1
S3	S5	0	S4	0
S4	S2	1	S2	0
S5	S5	0	S4	0

Moore FSM + Design Process: *remember tables need outputs*
- Output is determined solely by current state
- Moore provides more output stability
- Require more states
- Easier to build



Mealy/Moore Steps:
- Specification
- State Diagram
- State Table (with abstract states)
- State Assignment
- Excitation Table
- Combinatorial Logic (B Algebra, JM, QM, Shannon, etc)
- Circuit

Abstract States

S(t)\x	0	1
S0	S1,0	S0,0
S1	S2,0	S0,0
S2	S2,0	S0,1

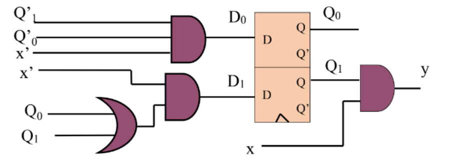
State Assignment
S0: 00
S1: 01
S2: 10

$Q_1(t)\ Q_0(t)$\x	0	1
00	01,0	00,0
01	10,0	00,0
10	10,0	00,1

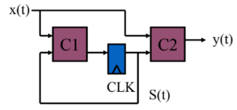
Excitation table → circuit(find d1(t), d2(t), y(t) with kmap):

id	Q_1Q_0x	D_1	D_0	y
0	000	0	1	0
1	001	0	0	0
2	010	1	0	0
3	011	0	0	0
4	100	1	0	0
5	101	0	0	1
6	110	X	X	X
7	111	X	X	X

ıtion Table – specify the next state inputs



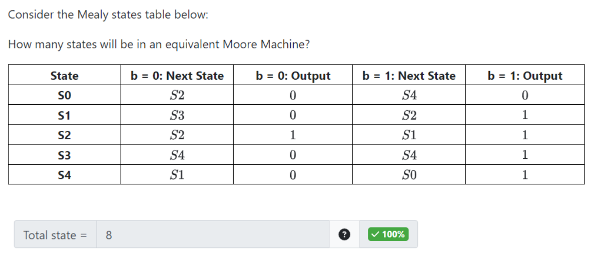
$D_1(t) = x'Q_0 + x'Q_1$
$D_0(t) = Q'_1Q'_0x'$
$y = Q_1x$



Mealy Machine

Mealy vs. Moore Machines:
- Mealy: outputs on the transitions
- Moore: outputs on the states
- Moore may have more states based on if Mealy states have two different outputs

Mealy to Moore:
- Identify unique (NS, y) pairs
- Replace each distinct (NS, y) pair with a distinct new state
- Insert rows for new states (S_n) with outpu y
- Append each present state with its output y
- Successor state of new state is successor state of mealy NS
- **Redundant States:** states with same output and same destination states for same inputs
- **Partition Method:** aka state minimization/state reduction
  - Group: fill out table and group states by transition and next state
  - Label: categorize successor groups (each group from above gets a different table and determine what next group for each row is by seeing which group the NS is in)
  - Finalize: determine which states are redundant



Consider the Mealy states table below:

How many states will be in an equivalent Moore Machine?

State	b = 0: Next State	b = 0: Output	b = 1: Next State	b = 1: Output
S0	$S2$	0	$S4$	0
S1	$S3$	0	$S2$	1
S2	$S2$	1	$S1$	1
S3	$S4$	0	$S4$	1
S4	$S1$	0	$S0$	1

Total state = 8 ✓ 100%

One-Hot Encoding: # states = # flip flops needed
- Encoding states order: 1000,0100,... (start the 1 from the left)