CS230 Notes : Soham Joshi

1 Circuits

A ciruit is a black box with

1. One or more discrete values input terminals
2. One or more discrete valued output terminals
3. Functional specification : reln bw inputs and outputs
4. Timing sp : Delay bw inputs changing and outputs resp

Difference :

1. Element : small circuit (black box)
2. Node : wire (input, output, internal)
   (a) Input : recv values from outside
   (b) Output : give values
   (c) Internal : Neither of the above two.

Types of digital circuits :

1. Combinational : Outputs depend only on the current values of the inputs, e.g. logic gates :- called memoryless
   (a) Outputs = f(inputs) :- specification usually via boolean eq (truth table), and K-maps to simplify the expressions
   (b) Timing : lower and upper bound specification (input -¿ output)
   (c) Popular ckt : full adder, $S = A \oplus B \oplus C_{in}$, $C = AB + BC_{in} + AC_{in}$
   (d) Single line with slash :- bus (multiple signals)
   (e) Rules : interconnected ckt elements st
      i. Each element is Combinational
      ii. Every node is either designated to input/exactly one output terminal of ckt element (`FANIN` not allowed)
      iii. No cyclic paths
2. Sequential ckt : has memory (chapter 3)

2 Boolean logic

Literal : var or its complement Minterm : Product involving all inputs to Function (e.g. for three inputs A,B,C, AB is not a minterm) Maxterm : Sum involving all inputs of function Operator precedence : `NOT` > `AND` > `OR` Canonical form :

1. sum-of-products form : Sum of all minterms where output is true
2. product-of-sums form : Product of all maxterms where output is false

Boolean algebra : axioms, theorems given below. Theorems of many vars :

Table 2.1 Axioms of Boolean algebra

	Axiom		Dual	Name
A1	$B = 0$ if $B \neq 1$	A1′	$B = 1$ if $B \neq 0$	Binary field
A2	$\overline{0} = 1$	A2′	$\overline{1} = 0$	NOT
A3	$0 \bullet 0 = 0$	A3′	$1 + 1 = 1$	AND/OR
A4	$1 \bullet 1 = 1$	A4′	$0 + 0 = 0$	AND/OR
A5	$0 \bullet 1 = 1 \bullet 0 = 0$	A5′	$1 + 0 = 0 + 1 = 1$	AND/OR

Table 2.2 Boolean theorems of one variable

	Theorem		Dual	Name
T1	$B \bullet 1 = B$	T1′	$B + 0 = B$	Identity
T2	$B \bullet 0 = 0$	T2′	$B + 1 = 1$	Null Element
T3	$B \bullet B = B$	T3′	$B + B = B$	Idempotency
T4		T3′	$\overline{\overline{B}} = B$	Involution
T5	$B \bullet \overline{B} = 0$	T5′	$B + \overline{B} = 1$	Complements

Table 2.3 Boolean theorems of several variables

	Theorem		Dual	Name
T6	$B \bullet C = C \bullet B$	T6′	$B + C = C + B$	Commutativity
T7	$(B \bullet C) \bullet D = B \bullet (C \bullet D)$	T7′	$(B + C) + D = B + (C + D)$	Associativity
T8	$(B \bullet C) + (B \bullet D) = B \bullet (C + D)$	T8′	$(B + C) \bullet (B + D) = B + (C \bullet D)$	Distributivity
T9	$B \bullet (B + C) = B$	T9′	$B + (B \bullet C) = B$	Covering
T10	$(B \bullet C) + (B \bullet \overline{C}) = B$	T10′	$(B + C) \bullet (B + \overline{C}) = B$	Combining
T11	$(B \bullet C) + (\overline{B} \bullet D) + (C \bullet D)$ $= B \bullet C + \overline{B} \bullet D$	T11′	$(B + C) \bullet (\overline{B} + D) \bullet (C + D)$ $= (B + C) \bullet (\overline{B} + D)$	Consensus
T12	$\overline{B_0 \bullet B_1 \bullet B_2 \ldots}$ $= (\overline{B_0} + \overline{B_1} + \overline{B_2} \ldots)$	T12′	$\overline{B_0 + B_1 + B_2 \ldots}$ $= (\overline{B_0} \bullet \overline{B_1} \bullet \overline{B_2})$	De Morgan's Theorem

Proving theorems in boolean algebra : Truth table!
Simplifying equations : Use theorems of many vars.
Minimized formula : Fewest number of implicants.
Implicant : Product term in the sum of products representation

Prime implicant : Cannot be combined with other implicants to form a new implicant with fewer literals.
All implicants in minimal equation must be prime.

Note : Expanding an implicant is sometimes useful in minimization.
Simplifying reduces the number of gates used to physically implement the function, thus making it smaller, cheaper, and possibly faster.

Table 2.4 Equation minimization

Step	Equation	Justification
	$\overline{A}B\overline{C} + AB\overline{C} + A\overline{B}C$	
1	$B\overline{C}\,(\overline{A} + A) + A\overline{B}C$	T8: Distributivity
2	$B\overline{C}\,(1) + A\overline{B}C$	T5: Complements
3	$B\overline{C} + A\overline{B}C$	T1: Identity

Table 2.5 Improved equation minimization

Step	Equation	Justification
	$\overline{A}B\overline{C} + AB\overline{C} + A\overline{B}C$	
1	$\overline{A}B\overline{C} + AB\overline{C} + AB\overline{C} + A\overline{B}C$	T3: Idempotency
2	$B\overline{C}(\overline{A} + A) + A\overline{B}(\overline{C} + C)$	T8: Distributivity
3	$B\overline{C}(1) + A\overline{B}(1)$	T5: Complements
4	$B\overline{C} + A\overline{B}$	T1: Identity

3 Gates

schematic : a diagram of a digital circuit showing the elements and the wires that connect them together.

Guidelines :

1. Inputs are on the left (or top) side of a schematic.
2. Outputs are on the right (or bottom) side of a schematic.
3. Whenever possible, gates should flow from left to right.
4. Straight wires are better to use than wires with multiple corners (jagged wires waste mental effort following the wire rather than thinking of what the circuit does).
5. Wires always connect at a T junction.
6. A dot where wires cross indicates a connection between the wires.
7. Wires crossing without a dot make no connection.

This style is called programmable logic array (PLA).
Example : priority circuit

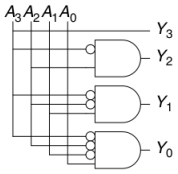


Figure 2.28 Priority circuit schematic

A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	X	0	0	1	0
0	1	X	X	0	1	0	0
1	X	X	X	1	0	0	0

Figure 2.29 Priority circuit truth table with don't cares (X's)

4 Multilevel combo logic

Layer of `AND` followed be level of `OR`.

Trick :

1. three input XOR = $(A \oplus B) \oplus C$ = two level of XORs.
2. 8 input XOR = eight input `AND` gates, one 128 `OR` gate. But it needs only 7 XORS.
3. Moving bubble from the end towards left, adding bubbles at input and AND to/from OR.

Special symbols :

1. X(contention) : unknown/illegal value. Driven to both 0 and 1 at the same time. (e.g. FANIN with one input 1, other 0)
2. Z(floating value) : driven by neither 0 nor 1. Misconception : Z is 0 (not true)

Tristate buffer : Active high enable. (1 enables buffer). Has three possible output states. 0 or 1 or Z (Z when the enable is False). Used in busses that connect multiple input chips.

K-maps : Easy visual way to minimize logic. Use fewest circles to connect all 1s.

5 Combinational building blocks

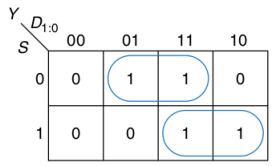
Multiplexer (mux): 2 : 1 mux :- S, D0, D1. S = i means Di is the output.

Wider multiplexers :- 4:1 :- two input state signals.

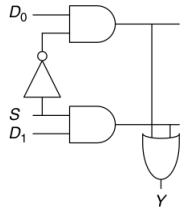
N:1 mux :- $log_2 N$ select lines. With a little cleverness, we can cut the multiplexer size in half, using only a $2^N - 1$ -input multiplexer to perform any N-input logic function. The strategy is to provide one of the literals, as well as 0s and 1s, to the multiplexer data inputs. First layer two 2:1 mux, second layer, one 2:1 mux. (hierarchial)

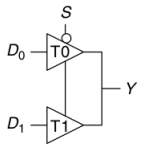
Decoders :

A decoder has N inputs and 2^N outputs. It asserts exactly one of its outputs depending on the input combination. (turns one input 1, rest 0 : "one-hot" outputs.)



$Y = D_0\overline{S} + D_1 S$

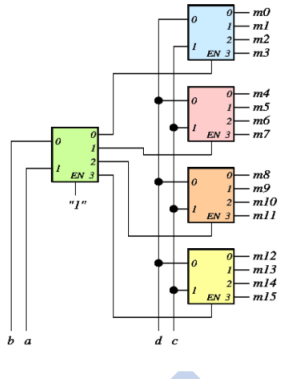




$Y = D_0\overline{S} + D_1 S$

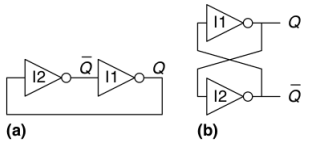
- Build a 4x16 decoder using 2x4 decoders (decoder tree)..

a	*b*	*c*	*d*	*EN*	*minterm*
0	0	0	0	1	m0
0	0	0	1	1	m1
0	0	1	0	1	m2
0	0	1	1	1	m3
0	1	0	0	1	m4
0	1	0	1	1	m5
0	1	1	0	1	m6
0	1	1	1	1	m7
1	0	0	0	1	m8
1	0	0	1	1	m9
1	0	1	0	1	m10
1	0	1	1	1	m11
1	1	0	0	1	m12
1	1	0	1	1	m13
1	1	1	0	1	m14
1	1	1	1	1	m15
X	X	X	X	0	0



6 Sequential Logic Design

Value depends on current state-variables. Design via finite-state machines.
Lateches and Flip flops :
bistable element (two stable states):- 1 bit of information : fundamental building block of memory. No input but two outputs. Analysis : take cases as Q = 0, Q = 1. and look for contradictions. i.e. look for stable points.



When power is first applied to a sequential circuit, the initial state is unknown and usually unpredictable. It may differ each time the circuit is turned on. Although the cross-coupled inverters can store a bit of information, they are not practical because the user has no inputs to control the state.

SR Latch :

SR latch : two cross coupled NOR gates. The inputs S and R stand for Set and Reset. To set a bit means to make it TRUE. To reset a bit means to make it FALSE. The outputs, Q and Q , are normally complementary. When R is asserted, Q is reset to 0 and Q does the opposite. When S is asserted, Q is set to 1 and Q does the opposite. When neither input is asserted, Q remembers its old value, Q prev . Asserting both S and R simultaneously doesn't make much sense because it means the latch should be set and reset at the same time, which is impossible. The poor confused circuit responds by making both outputs 0.

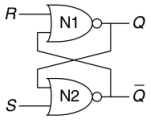


Figure 3.3 SR latch schematic

No matter what pattern of setting and resetting occurred in the past, all that is needed to predict the future behavior of the SR latch is whether it was most recently set or reset.

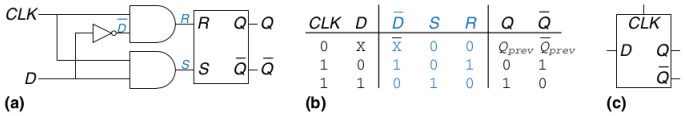
D Latch :

The SR latch is awkward because it behaves strangely when both S and R are simultaneously asserted. Moreover, the S and R inputs conflate the issues of what and when. Asserting one of the inputs determines not only what the state should be but also when it should change. Designing circuits becomes easier when these questions of what and when are separated. The D latch in Figure 3.7(a) solves these problems. It has two inputs. The data input, D, controls what the next state should be. The clock input, CLK, controls when the state should change.

Case	S	R	Q	$\bar{Q}$
IV	0	0	Q_{prev}	$\bar{Q}_{prev}$
I	0	1	0	1
II	1	0	1	0
III	1	1	0	0

Figure 3.5 SR latch truth table

Figure 3.6 SR latch symbol



CLK	D	$\bar{D}$	S	R	Q	$\bar{Q}$
0	X	$\bar{X}$	0	0	Q_{prev}	$\bar{Q}_{prev}$
1	0	1	0	1	0	1
1	1	0	1	0	1	0

Figure 3.7 D latch: (a) schematic, (b) truth table, (c) symbol

D Flip Flop : D flip-flop copies D to Q on the rising edge of the clock, and remembers its state at all other times.

A D flip-flop is also known as a master-slave flip-flop, an edge-triggered flip-flop, or a positive edge-triggered flip-flop. The triangle in the symbols denotes an edge-triggered clock input. The Q output is often omitted when it is not needed.

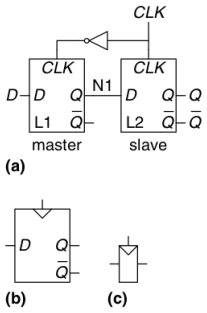


Figure 3.8 D flip-flop: (a) schematic, (b) symbol, (c) condensed symbol

Register : An N-bit register is a bank of N flip-flops that share a common CLK input, so that all bits of the register are updated at the same time.

Enabled flip-flop : An enabled flip-flop adds another input called EN or ENABLE to determine whether data is loaded on the clock edge. When EN is TRUE, the enabled flip-flop behaves like an ordinary D flip-flop. When EN is FALSE, the enabled flip-flop ignores the clock and retains its state.

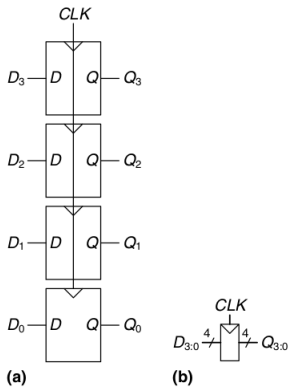


Figure 3.9 A 4-bit register: (a) schematic and (b) symbol

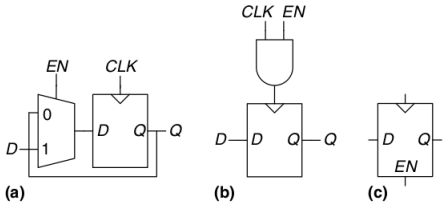


Figure 3.10 Enabled flip-flop: (a, b) schematics, (c) symbol

Resettable flip flops : A resettable flip-flop adds another input called RESET. When RESET is FALSE, the resettable flip-flop behaves like an ordinary D flip-flop. When RESET is TRUE, the resettable flip-flop ignores D and resets the output to 0. Resettable flip-flops are useful when we want to force a known state (i.e., 0) into all the flip-flops in a system when we first turn it on.

6.1 Finite state machines

1. These forms are called finite state machines (FSMs). They get their name because a circuit with k registers can be in one of a finite num- ber (2^k) of unique states.

2. An FSM has M inputs, N outputs, and k bits of state. It also receives a clock and, optionally, a reset signal.

3. An FSM consists of two blocks of combinational logic, next state logic and output logic, and a register that stores the state.

4. On each clock edge, the FSM advances to the next state, which was computed based on the current state and inputs.

5. There are two general classes of finite state machines, charac- terized by their functional specifications. In Moore machines, the outputs depend

only on the current state of the machine. In Mealy machines, the outputs depend on both the current state and the current inputs.

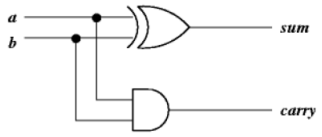
6. Finite state machines provide a systematic way to design synchronous sequential circuits given a functional specification.

General design :

1. For a Moore machine:
   (a) Write a state transition table.
   (b) Write an output table.
2. For a Mealy machine:
   (a) Write a combined state transition and output table.
3. Select state encodings–your selection affects the hardware design.
4. Write Boolean equations for the next state and output logic.
5. Sketch the circuit schematic.

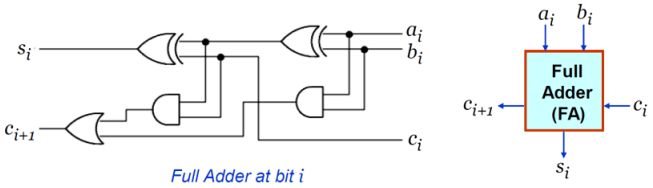
We will repeatedly use FSMs to design complex digital systems

7 Important Circuits

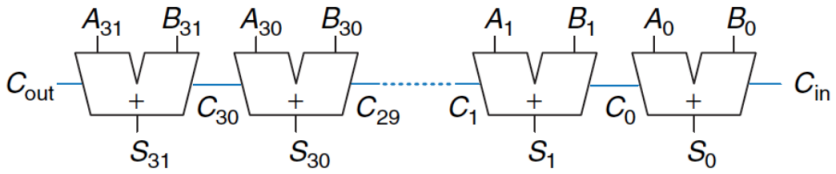


$Sum = \bar{a}b + a\bar{b} = a \oplus b$
$Carry = ab$

$$Sum = \bar{a}\bar{b}c + \bar{a}b\bar{c} + a\bar{b}\bar{c} + abc = a \oplus b \oplus c$$
$$C_{out} = ab + ac + bc = ab + c(a+b)$$



Ripple Carry Adder



Comparator (Equality Checker)