Digital Electronics - Combinational Logic

Kevalee Shah

April 16, 2019

Contents

1 Boolean Algebra

- `xor` - samer as or, but false for $1, 1$ input
- `and` rules:

  $a.0 = 0$

  $a.a = a$

  $a.1 = a$

  $a.\overline{a} = 0$

- `or` rules:

  $a + 0 = 0$

  $a + a = a$

  $a + 1 = 1$

  $a + \overline{a} = 1$

- Distribution:

  $a.(b + c + K) = a.b + a.c + K$

  $a + (b.c.K) = (a + b.(a + c).K$

- Absorption:

  $a + (a.c) = a$

  $a.(a + c) = a$

- A good trick to keep in mind: $a = a.b + a.\overline{b}$
  e.g. Prove that $a + (a.b) = a$

$$a = a.b + a.\overline{b}$$
$$\implies a.(a + c) = a.b + a.\overline{b} + a.b = a.b + a.\overline{b} = a.(b + \overline{b}) = a.1 = a$$

- DeMorgan's Theorem:

  $a + b + c = \overline{\overline{a}.\overline{b}.\overline{c}}$

  $a.b.c = \overline{\overline{a} + \overline{b} + \overline{c}}$

  $\overline{a + b + c} = \overline{a}.\overline{b}.\overline{c}$

  $\overline{a.b.c} = \overline{a} + \overline{b} + \overline{c}$

- `nand` and `nor` gates are simpler and faster
- $f = a.b + c.d$ is the same as $f = \overline{(\overline{a.b}).(\overline{c.d})}$
- negating the inputs into an `or` gate is the same as a `nand` gate
- negating the inputs into an `and` gate is the same as a `nor` gate

1.1 Karnaugh Maps

- Karnaugh maps are used for up to 5 variables
- Truth tables show how a function behaves for every combination of the input variables.
- Minterms are given where a function has value 1. It must contain all variables, either in complement or uncomplemented form. The variables are 'anded'. E.g. for inputs x, y, z, the minterm $x.\bar{y}.\bar{z}$ corresponds to $x = 1, y = 0, z = 0$
- Disjunctive Normal Form: when a boolean function is expressed as the disjunction of its minterms - ORing the minterms
- Sum of Products: a function expressed as the ORing of ANDed variables - doesn't have to be minterms
- Maxterms: given where a function has value 0, which is the same as the minterms of the complement function. DeMorgan can be used to get a POS expression. e.g.

$$f = \bar{x}.\bar{y}.\bar{z} + \bar{x}.\bar{y}.z + \bar{x}.y.\bar{z} + \bar{x}.y.z + x.y.z$$
$$\bar{f} = x.\bar{y}.\bar{z} + x.\bar{y}.z + x.y.\bar{z}$$
Using DeMorgan we get:
$$f = (\bar{x} + y + z).(\bar{x} + y + \bar{z}).(\bar{x} + \bar{y} + z)$$

- Conjunctive Normal Form: when the function is expresses as the conjunction of maxterms - ANDing max terms
- Product of Sums: a function expressed as the ANDing of ORed variables - dont have to be maxterms
- Karnaugh Maps: also known as K-maps, are a way of visualising and simplifying logical expressions. The table has the variables on top which are Gray Coded - only one variable changes at a time. Mark where the minterms are and group them together in order to simplify
- POS simplification can be done on the K-map by plotting the complemented function instead - plot the maxterms. Then use DeMorgan to get the complement of the complemented function - the original function
- Dont Cares: when the output value of certain inputs doesn't matter then we can plot these on the K-map as well, and only use them if it helps with simplification
- Cover: a term is said to cover a minterm if the minterm is part of that term
- Prime Implicant: a term that cannot be further combined
- Essential Prime Implicant: prime implicant that covers a minterm that no other prime implicant covers
- Cover Set: minimum set of prime implicants which include all essential terms pls any other prime implicants needed to cover all minterms

1.2 QM Method

1. Write out the minterms and dont cares in binary
2. Group by how many 1s there are
3. Separate each group with a line
4. Compare first group with second, second with third ... pairwise
5. If the pair only differs by one digit then the combined pair in a separate column, with a dash where the difference is
6. Repeat this method until no more pairings can be made
7. During the process if some terms cannot be paired then star them
8. Collect all starred terms and write out what input values correspond to the binary value

   $(2, 6) = 0_1 0$ corresponds to $\bar{a}.c.\bar{d}$
9. Create a table - the rows are the starred expressions, the columns are the minterms (DO NOT INCLUDE DONT CARES)
10. Put an x in the table where the row terms correspond to the headings
11. If a cross is found in a column where there are no other crosses, then you know that term has to be included. Draw a line across to indicate the term has been chosen, and then vertical lines on the other points to see which other minterms you now have because of that term.
12. Continue choosing the min number of starred terms so that all minterms appear.
13. Using the corresponding input values, write a simplified expression for f

2 Binary Adders

- Half Adder: adds together two single bit binary inputs, with no carry input but a carry output.

  $sum = a \bigoplus b$

  $c_{out} = a.b$
- Full Adder: adds together two single bit binary inputs, with a carry input and a carry output

  $sum = c_{in} \bigoplus a \bigoplus b$

  $c_{out} = b.a + c_{in}.(b + a) = c_{in}.(a \bigoplus b) + a.b$
- Ripple Carry Adder: In order to add two n-bit binary numbers we use a ripple carry adder, which is n full adders cascaded together, so that the c_{out} of one is the c_{in} of another.
- Subtraction: If we want to calculate $s = a - b$, we can write this as $s = a + (-b)$. This is now the addition of a with the 2's complement of b.

The 2's complement of b is inverting all the bits of b and then adding 1

We can invert bits by $\bar{b} = b \bigoplus 1$ - pass each bit of b through a `xor` gate

Adding one just means set c_0, the first carry in bit to 1

- In order to speed up a ripple carry adder:
  - Instead of having a compositional approach, we can just have one block with $2n + 1$ inputs and $n + 1$ outputs. This has a low delay, as only two gates would need to be used, but it requires inputs of $2n$ which as the numbers get bigger is impossible. It would have a very complex design
  - Instead use full adders, but generate the carry signals independently , and therefore we do not have to wait for the c_{out} of the previous adder to be generated, which reduces the number of gate propagation delays

2.1 Fast Carry Adder

To determine the boolean logic to generate the fast carry signals, we can consider the c_{out} generated by a full adder at stage i which is the c_{in} for adder $i + 1$

- We know that if $\bar{a_i}.\bar{b_i}$ the carry out is always 0 - carry kill

  $k_i = \bar{a_i}.\bar{b_i}$

- We know that if $a_i \bigoplus b_i$ (different inputs) then the carry out is the same as the carry in - carry propagate

  $p_i = a_i \bigoplus b_i$

- We know that if $a_i.b_i$ then the carry out is always 1, independent of the carry in - carry generate

  $g_i = a_i.b_i$

Using the above facts, and that $c_{i+1} = a_i.b_i + c_i.(a_i + b_i) = a_i.b_i + c_i.(a_i \bigoplus b_i)$
This gives $c_{i+1} = q_i + c_i.p_i$
We can keep on doing this recursively for all c Doing this calculation is much faster than waiting for each adder to generate the carry out to be the next carry in.
However in order to reduce complexity, often each 4-bit adder block generates c_4 using the fast carry logic, and blocks of 4-bit adders are using to make the larger $n-$bit adders.

3 Multilevel Logic

Using multilvel logic we can significantly reduce the number of wires and gates needed to implement the function, however there is a greater delay because of the increased levels of logic. Multilevel logic are more gate-efficient than two

level implementations but have worse gate propagation delays
e.g.

$$f = ADF + AEF + BDF + BEF + CDF + CEF + G$$

This can be implemented using 6 3-input AND gates, and a 7-input OR gate. This has in total 7 gates and nine literals.
Or

$$f = (A + B + C)(D + E)F + G$$

This shows that f can also be implemented using one 3-input OR gates, 2 2-input OR gates, and one 3-input AND gates - 4 gates and 9 literals.

3.1 Gate Propagation and Hazards

Logic gates are basically transistors and therefore they only have a finite switching speed. That means there is a finite delay before the output of a gate responds to the change in inputs. If there are many gates in a cascade there is an increase in time before the output is valid (e.g. ripple carry adder, MSB has to wait for the carry inputs)
This leads to a slower operation and can also give rise to hazards
Hazard: brief unwanted logic level changes at the output in response to changing inputs.

- Static - output undergoes momentary transition when one input changes, when it is supposed to remain unchanged
- Dynamic - output changes more than once when it is only meant to change once

Assumptions of timing diagrams: only two levels, when it reality the signal is more of a squiggle, instantaneous transition though it actually takes finite time
To remove a static 1 hazard - draw K-map, add another term that overlaps all the essential terms
To remove a static 0 hazard - draw K-map of complement function, add another term that overlaps all the essential terms

4 Other Logic Gates

4.1 Multiplexor

Device that selects betweene several inputs, under the direction of a control input, and forwards it to a signle output. For 2^n inputs, there are n control inputs. Diagram is like a trapezium. They are used to increase the amount of data that can be sent over the network, and can also be used to implement boolean functions. Can get $2:1, 4:1, 8:1$ muxes

4.2 Demultiplexor

Opposite of a multiplexor - single input to exactly one output. e.g. a $1:2$ demux has 1 control input, 1 input and 2 outputs. It takes a single input line and routes it to one of the serveral output lines. A demux of 2^n outputs has n inputs

4.3 Decoder

Same as a demultiplexor, but input is connected to one. $(1:2$ decoder). In general it decodes n bits of input into 2^n bits of output. A decoder doesnt have any select lines. Types are $1:2, 2:4, 3:8, 4:16$. It is used to decode instructions before execution. Every probable input condition only one output signal with produce the logic one.

4.4 ROM

- data storage device
- written into once
- read at will but cant rewrite
- look up table - n input lines specify addresses of locations holding m-bits
- e.g. if $n = 4$ then there are $2^4 = 16$ locations, and if $m = 4$ then each location can store a $4-$bit word
- Total number of bits store is very small - $m \times 2^n$
- fairly efficient if many minterms need to be generated
- inefficient when number of minterms is small

4.5 PLA

Programmable Logic Array - with a programmable AND array and a programmable OR array. A PLA only needs to have enough AND gates to decode as many unique terms as there are functions it will implement. It is more efficient to implement sparse output functions compared to a ROM , as not all minterms are decoded. For n inputs there are fewer than 2^n AND gates - only needs enough gates to decode unique terms. It is advantageous to minimize the expression, but need to keep in mind that product terms can be shared between functions.

4.6 PAL

Programmable Array Logic - with a programmable AND array but a fixed OR array. This means that it is cheaper, but cannot share product terms between functions. It is advantageous to minimize functions

5 Sequential Logic

Type of logic where output depends not only on the latest inputs, but also on the condition of earlier inputs. They contain memory elements - store data one bit per element
A snapshot of the memory is called the state. Bistables are when a one bit memory has two stable states, such as flip flops and latches
An asynchronous operation is one in which tthe output state changes directly in reponse to a change in inputs

However mostly all sequential circuits use synchronous operations - output is only allowed to change at a time specified by a global enabling signal - clock signal. The clock allows lots of state changes to appear simultaneously. It imposes some order on the state changes. Usually a square signal at a specified frequency.

6 Latches and Flip Flops

6.1 RS Latch

Two inputs - set and reset, with two outputs $Q, \bar{Q}$

- $S = 0, R = 0$: hold - $Q' = Q$
- $S = 0, R = 1$: reset - $Q' = 0$
- $S = 1, R = 0$: hold - $Q' = 1$
- $S = 1, R = 1$: hold - illegal

It is made up of two `NOR` gates - Gate 1 output is Q, input is $\bar{Q}$ and R. Gate 2 output is $\bar{Q}$ and input is Q and S

6.2 Transparent D-Latch

Similar to a RS Latch, but the output is only allowed to change if there is a valid enable signal. Now the inputs are D, EN. The input to R is the complement of D and EN and the input to S is D and EN. The complement ensures that R and S are both never the same and therefore there cannot be an illegal state. It is made up of two `NOR` gates, 2 `AND` gates and one `NOT` gate. When EN is 0 it doesnt matter what D is, the latch is on hold. When EN is 1, then D determines if the latch is to be reset or set.

6.3 JK Flip Flop

Same as a clocked RS latch, but instead of an illegal state, there is a toggle state - alternates between 0 and 1.

6.4 T Flip Flop

Same as JK, but the J and K are connected together to form one input - T. This has two states: hold and toggle.

6.5 D-Type

This is when there is one input, and the current input determines the next state.

- $Q = 0, D = 0 \rightarrow Q' = 0$
- $Q = 0, D = 1 \rightarrow Q' = 1$
- $Q = 1, D = 0 \rightarrow Q' = 0$
- $Q = 1, D = 1 \rightarrow Q' = 1$

This leads us to the equation $Q' = D$

7 Tables and Diagrams

7.1 State Transition Table

This shows every possible combination of the current state and inputs, and what the next state will be.
e.g. for the RS Latch the columns are $QRS|Q'$

A modified state transition table has columns for current state, next state and the additonal columns for the flip flops required to make this transition.

7.2 State Diagrams

Have a circle for every possible state. Draw arrows to show transitions from one state to another. On the arrows have labels to show which input corresponds to that transition.

7.3 Excitation Table

Similar to a state transition table, but it instead has the current state and next state together and shows the input required to make that transition
e.g. for a D-type flip flop, the columns are $QQ'|D$

8 Counters

Problems with using ripple counters (cascading T-type FFs in toggle mode) -

- outputs do not change at the same time
- hard to know when count output actually valid
- as FFs not clocked using the same clock - not synchronous
- Propogation delays build up and therefore limits the max clock speed before micounting occurs

The frequency of counter output signals - each has half the frequency. Double the repetition period of previous one.
The issues above leads to having a synchronous counter so that all the flip flops are connected to the same clock.
Drawing state transition table with D-type inputs can help determine the logic required to implement the counter. Remember to add unused counts as dont cares when doing the K-maps. e.g to count up to 7, can use the 8 count as don't care.

8.1 Shift registers

A cascade of FFs that share the same clock in which the output of each FF is connected to the data input of the next flip flop in the chain. This reuslts in a circuit that shifts by one position in the bit array - shifts in the next data and shifts out the last bit in the array. Two types - parallel in serial out or serial in and parallel out.

9 State Machines

- FSM - deterministic machine that produces outputs based on internal state and external inputs.
- States - set of internal memorised values that are shown as circles in the state diagram
- Inputs - external stimuli, labelled as arcs
- Outputs - results of the FSM

9.1 Moore Machine

FSM whose outputs depend only on present state. Generally it has more states than Mealey. Value of the output function is a function of the current state and the changes at the clock edges, whenever state change occurs. They have guaranteed timing characteristics as outputs come straight from clocked FFs. The are glitch free. Any Moore machine can be converted into a Mealey Machine, but the timing properties will not be the same In a state diagram, the states are in cirlces, and the labels on the arrows are only the inputs.

9.2 Mealey Machines

Output depends both upon the present state and the present input. Outputs depend on the timing of the inputs. Any Mealey machine can be converted into a Moore Machine, but the timing properties will not be the same. Generally has fewer states than a Moore machine. Value of the output is a function of the transitions and changes when the input logic on the present state is done. In a state diagram, the states are in the circles, and inputs and outputs appear on the arrows.