GATE CSE

GATE CSE Book

November 2016

GATE CSE

Preface

This book is made thanks to the effort of GATE CSE members and Praneeth who made most of the latex notes for GATE CSE. Remaining work of completing the syllabus is done by Ananya Mukhopadhyay and Arjun Suresh. Special thanks to Bikram Ballav for providing read from materials and suggestions. Also thanks to Praveen Saini and Ravi Singh for leading many discussions for GATECSE which are added either as links or contents in the book.

This book is not a complete note for GATE CSE, rather what to study and where to study from, which stuffs you might not notice while studying, more important, etc. and solutions to some of the most confusing questions. And some notes are also included though not complete - hence ***please never read only this material for GATE****. They are included just to highlight the parts to be read from the given standard textbook or weblinks. Being first revision there can also be many typos/grammatical errors. Still care has been made to ensure content is the best possible.*

Margin notes have been used to drive attention wherever necessary. Some topics given are given less importance by aspirants (may be coaching classes skipped them). Having seen most of the GATE questions multiple times as part of GATE Overflow I have tried to include all the topics– at least as a heading – so that no one misses any topic relevant for GATE. Some advanced topics are very very unlikely to be asked for GATE – still since I'm not 100% sure, have added them with a blue mark. Most people can skip those if time is an issue. Most frequently asked portions are added in **bold** in the syllabus detailed section for each subject.

Initial plan was to include GATECSE discussions happened in 2016 here, but could not do it as focus was more on noting down everything in syllabus. So, discussion links are given in Weblinks part of each subject. Subjects like Mathematics had no discussion.

Contents

1

FAQs

What is the most important subject for GATE?

Answer.

Who makes GATE question paper?

Answer.

What is the minimum bachelor percentage to get to IITs?

Answer.

What should be my preparation strategy?

Answer.

Which test series to follow for GATE?

Answer.

How to improve my accuracy during exam?

Answer.

2

Digital Logic

2.1 Syllabus

- Boolean algebra
- Combinational and sequential circuits
- Minimization
- Number representations and computer arithmetic (fixed and floating point)
- Booth's Algorithm

2.2 Syllabus Detailed

Types of Questions asked and what to focus more given in boxes

Using Mano [1] as reference.
NPTEL - Recommended Video

1. Boolean algebra - Boolean Functions, Canonical Forms. Chapter 2, sections 1-5.
2. Combinational circuits - Half/Full adder, Half/Full Subtractor, Code Conversion, **Binary Parallel Adder**, **MUX, DEMUX**, Encoder, Decoder, ROM, PLA. Chapters 4-5.
3. **Sequential circuits - SR FlipFlop, JK FlipFlop, T FlipFlop, D FlipFlop, Counters**, Synchronous Counters, Ripple Counters. Asynchronous Sequential Circuits- Hazards. Chapters 6, 7, 9.
4. **Minimization - K-MAP, SOP, POS, Prime Implicants**. Chapter 3.

No. of Boolean functions possible, simplification of Boolean functions

Determine the output of circuit, **propagation delay of adder, carry-look-ahead generation, minimum no. of NAND/NOR gates required**, static hazard

Flip-flops given and modulus of counter asked, for sequential circuit output after 't' cycles, counters using flipflops, static hazard

Minimization asked a lot, minimization with don't cares

5. Number representations and computer arithmetic - fixed and floating point, 1's complement, 2's complement, sign magnitude format for negative numbers, **IEEE 754 Floating Point Representation for single**/double precision, IEEE 754 Single Precision Ranges. Partly covered in Chapters 1, 8 and in Computer Organization. Also see Section 2.3 which details this beautifully.

As of now IEEE 754 representation is mostly asked and previous representations not required, range, precision, difference between 2's complement and 2's complement representation

2.2.1 Boolean algebra

Definition 2.1 (Duality Principle). *A Boolean expression remains valid if the operators ($\vee and \wedge$) and the identity elements (0 and 1) are interchanged. Hence, to show a Boolean expression as valid, it is enough to show for its dual.*

Precedence of Boolean Operators

() > NOT > AND > OR

Definition 2.2 (Literal). *A literal is a primed or unprimed variable.*

These corresponds to inputs when a logic circuit is made from the Boolean expression. Minimizing the literals and terms in the expression can reduce the cost of the circuit. Both may not be simultaneously possible. Minimizing literals can be done by algebraic simplification - but there is no definite algorithm for this. Only cut-and-try procedure works.

Boolean Functions

Each variable in a Boolean expression - formed with $(), \vee, \wedge, \neg$ and $=$ is mapped to $\{0, 1\}$. On the RHS it is either 0 or 1. i.e, each Boolean function is a mapping from the set of all combination of binary values possible for the input variables to either 0 or 1. Set of all possible combinations for n binary variables is 2^n and hence

Theorem 2.3. *The total number of Boolean functions over n variables is 2^{2^n}.*

Theorem 2.4. *DeMorgan's Theorem*

1. $(A + B + C + D + E + F)' = A'.B'.C'.D'.E'.F'$
2. $(A.B.C.D.E.F)' = A' + B' + C' + D' + E' + F'$

i.e., complement of a Boolean expression is equal to the expression with each literal being complemented and AND and OR interchanged – i.e., take the dual and complement each literal.

Canonical Forms

Min Term or Standard Product - 2^n minterms possible for n variables. For 3 variables x, y, z the minterms are

$$x'y'z', x'y'z, x'yz', x'yz, xy'z', xy'z, xyz', xyz$$

and are denoted by $m_0, m_1 \ldots m_7$ and similarly for n variables.

Max Term or Standard Sum - 2^n maxterms possible for n variables For 3 variables x, y, z the maxterms are

$$x' + y' + z', x' + y' + z, x' + y + z', x' + y + z', x + y' + z', x + y' + z, x + y + z', x + y + z$$

denoted by $M_0, M_1 \ldots M_7$ and similarly for n variables.

Any Boolean function can be expresses as:

1. Sum of minterms
2. Product of maxterms

Note 2.5. The complement of a Boolean function equals the minterms missing from the sum of minterms expression for the function.

Standard Forms

Canonical forms are not practically usable as each term must have all the literals present (either prime or unprime). Practically usable are

1. Sum of Products - ORing of product terms
2. Product of Sums - ANDing of sum terms

forms where each term can have any number of literals. These two form the standard forms. A Boolean expression having a mix of the two is not in standard form.

Minimization - Karnaugh's Map

Is a table of boxes. Each box represents a minterm. It presents all possible ways a Boolean function can be represented in standard form. Gives "Minimal" Sum of Products expression as well as Product of Sum expression where minimal is with respect to the number of literals. Both of these need not necessarily unique.

Definition 2.6 (Prime Implicants). *Is a product term obtained by combining maximum number of adjacent squares possible in the K-map*

Definition 2.7 (Essential Prime Implicants). *Is a minterm in a square that is covered by only one prime implicant in K-map*

Hence, an essential prime implicant will be part of any minimal sum of products expression but a prime implicant need not always be.

From K-map we can get both minimal sum of product form as well as minimal product of sum form. To get minimal sum of product expression, we combine the 1's and then write down the corresponding minterms and sum them. To get minimal product of sums form, we combine the 0's (empty squares marked by 0's) and get the minimal sum of product form of the complement function. Then we take its dual and complement each literal and this will give the minimal product of sum form.

Examples should be worked out as this has been asked many times for GATE.

NAND/NOR GATE

Steps to realize a Boolean circuit for a Boolean expression are as follows:

1. Convert to minimal sum of product form (K-map)
2. Draw a NAND gate for each term that has at least 2 literals which forms a set of first level NAND gates
3. Draw a single second level NAND gate
4. A term with a single literal requires an inverter in the first level or can be complemented and given as input to the second level NAND gate

For NOR gate realization we do the same procedure except that we use minimal product of sums form.

Tabulation Method

1. A systematic approach which guarantees to give a minimal standard form expression for a Boolean function – K-map is a trial-and-error method and becomes tedious with more number of variables
2. Also known as Quine-McCluskey method
3. Consists of 2 steps- first search for all terms that are candidates for inclusion in minimal form and then choose the ones which gives the minimal form
4. Is tedious for humans

2.2.2 Combinational and Sequential Circuits

Note 2.8. In combinational circuit, output depends only on the current input whereas in a sequential circuit output depends not only on the current input but also on previous inputs (memory element)

Half Adder

1. Sum=$A \oplus B$
2. Carry=AB

Full Adder

1. Sum=$A \oplus B \oplus C$
2. Carry=$AB + BC + CA$

Half Subtractor

1. Difference=$A \oplus B$
2. Borrow=$A - B$

Full Subtractor

1. Difference=$A \oplus B \oplus C$
2. Borrow=$(A - B) - C$

Decoder

Decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2^n (less in case of dont-cares) unique output lines.

Demultiplexer

A decoder with an enable input can function as a demultiplexer. A demultiplexer is a circuit that receives information on a single line and transmits this information on one of 2^n possible output lines.

Encoder

Inverse of a decoder. Has 2^n(or fewer) input lines and n output lines.

Multiplexer

Is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line. For two select lines A and B and four inputs I_0, I_1, I_2, I_3 the output is given by

1. $f(A, B) = A'B'I_0 + A'BI_1 + AB'I_2 + ABI_3$

For all the above, one need to know to determine the outputs when put in a circuit with other components

ROM

Any Boolean function can be expressed as a sum-of-products form. Now, for each minterm we have an output. By removing the links of the terms not included in a function, we can make a ROM represent the Boolean function of one of the output variables in the combinational circuit. For n input m output combinational circuit ROM must be $2^n \times m$. In short we store m outputs in ROM and selects them based on n inputs.

Types of ROMs

1. Programmable ROM (PROM)- Can be written only once
2. Erasable PROM (EPROM) - Can be erased using ultra-violet rays and reprogrammed
3. Electrically Erasable PROM (RPOM) - Can be erased using electrical signals and reprogrammed

ROMs are used to implement combinational circuit directly from their truth tables. . They are used in binary code conversions and in the design of control units of digital systems.

The size of ROM needed for circuits like multipliers have been asked in GATE

Definition 2.9 (Microprogrammed Control Unit). - *A control unit that utlizes ROM to store binary control information.*

Code Conversion

To facilitate the usage of output of one system as input of another. BCD, Excess-3 Gray codes are some common binary codes.

Examples have been asked in GATE

2.2.3 Sequential circuit

Flip-Flops

1. S-R Flip-Flop - $Q_n = S + R'Q$. Gets to indeterminate state when $S = R = 1$.
2. JK Flip-Flop - $Q_n = JQ' + K'Q$. No indeterminate state. Can be made by cross coupling either NAND gates or NOR gates
3. D Flip-Flop - $Q_n = Q$.
4. T Flip-Flop - $Q_n = Q'$

Edge/Level Triggering

Questions have been asked previously based on timing diagram.

State diagram Construction - if you know Automata nothing new here – but see previous year questions

Excitation Table: Tells the flipflop input condition which can cause a change from present state to new one.

2.2.4 Counters

An $n - bit$ counter can count from 0 to $2^n - 1$ and consists of n flipflops. The counting sequence need not be the same as binary numbers.

1. Synchronous Counter - A Clock Pulse is applied to all flipflops which triggers them.
   - Ring Counter: A circular shift register with only 1 flipflop being set at any time. The number of flip-flops required is the mod of the counter.
   - Johnson Counter : Is a $k - bit$ switch tail (complement output of last flipflop given as input to the first) ring counter that can provide outputs for $2k$ timing signals with $2k$ decoder gates. Only 2 input gates are required. The number of flip-flops required is one half the number of timing signals.
2. Asynchronous(Ripple) Counter - Output of one or more flipflops triggers other flipflops. No common clock. For n bit binary ripple counter required n flip-flops.

Definition 2.10 (RAM). *Memory cells can be accessed for information transfer to or from any desired random location and hence called Random Access Memory.*

2.2.5 Error Correcting Code

Hamming Code - In the Hamming code k parity bits are added to n data bits to get a code of $n + k$ bits. Can detect and correct only a single error. By adding an extra parity bit, Hamming code can detect double errors and correct a single one.

Please see text book for creation and detection of error using Hamming Code

1. Parity bit Generator
2. Parity bit checker

Please see text book

2.2.6 Asynchronous Sequential Circuits

1. The memory elements are unclocked FFs or time delay elements.
2. Similar to a combinatorial circuit with feedback
3. Are faster than synchronous sequential circuits
4. For proper operation, inputs must be allowed to change only after the system has reached a stable state

Definition 2.11 (Race Condition). *Happens in an asynchronous sequential circuit when two or more binary state variables change their value in response to change in an input variable.*

Definition 2.12 (Hazards). *Are unwanted switching transients that may appear at the output of a circuit because different paths exhibit different propagation delays.*

1. Happens in combinational circuit whenever the circuit moves from one minterm to another causing the output to be momentarily 0
2. Static 0 Hazard - Output going to 1 when it should remain 0
3. Static 1 Hazard - Output going to 0 when it should remain 1
4. Dynamic Hazard - Output changing 3 or more times when it should change from 0 to 1 or vice verse.
5. Avoided by adding redundant gates often by combining the minterms and getting a new minterm

A problem asked in GATE 2006

2.2.7 Number Representations

Conversion of numbers(binary to octal, octal to binary, binary to gray, gray to binary, Excess 3 code conversion)

One example for converting between Hexadecimal to Octal. Let input be 0xAB72

1. Convert to binary by taking each hex-digit and writing its binary:

$$1010\ 1011\ 0111\ 0010$$

2. Starting from right group the digits by pairs of 3 adding 0's to left if required)

$$1\ 010\ 101\ 101\ 110\ 010$$

3. Convert each of the 3 bits to octal digit

$$125,562$$

And we are done.

Please see text and previous GATE questions for more examples and other conversions

Complements

Used for simplifying subtraction. Two types:

1. $(r-1)'s$ complement: For a given number N in base r having n digits, $(r-1)'s$ complement is $(r^n-1)-N$.
   - 9's complement of $54678 = 99999 - 54678 = 45321$
   - 1's complement of $10110 = 11111 - 10110 = 01001$
2. $r's$ complement: For a given number N in base r having n digits, $r's$ complement is $(r^n) - N$, if $N \neq 0$ and 0 if $N = 0$. Can be obtained by adding 1 to the $(r-1)'s$ complement
   - 10's complement of $54678 = 100000 - 54678 = 45322$
   - 2's complement of $10110 = 100000 - 10110 = 01010$ (Reverse all bits before the right most 1)

Signed Numbers

Represented in

1. Sign Magnitude Form - leading bit is sign bit. 0 for positive and 1 for negative.
2. 1's complement Form - negative numbers represented in 1's complement form (swapping 0's and 1's)
3. 2's complement - negative numbers represented in 2's complement form (1's complement + 1)

Note 2.13. In all the above forms positive numbers are represented as it is. i.e., say for 11, 1's complement representation = 2's complement representation = sign magnitude representation = 00001010.

Important!!!

Note 2.14. Both 1's complement and sign magnitude forms have 2 representations for 0

Binary Codes

Are used to represent decimal digits. Different codes can be used as shown in the following table.

Decimal Digit	BCD (8421)	Excess-3	84-2-1	2421	Biquinary (5043210)
0	0000	0011	0000	0000	0100001
1	0001	0100	0111	0001	0100010
2	0010	0101	0110	0010	0100100
3	0011	0110	0101	0011	0101000
4	0100	0111	0100	0100	0110000
5	0101	1000	1011	1011	1000001
6	0110	1001	1010	1100	1000010
7	0111	1010	1001	1101	1000100
8	1000	1011	1000	1110	1001000
9	1001	1100	1111	1111	1010000

Note 2.15. Binary equivalent of a decimal number and binary code of a decimal number are different. For decimal digits from 0 to 9, these are the same for BCD code. But for 10, binary value is 1010 whereas binary code is 00010000

BCD is the most commonly used code. But it is not self-complementing where as Excess-3, 84-2-1 and 2421 are self complementing codes. i.e., in order to obtain the 9's complement of a number we can simply exchange 0's and 1's in the binary codes.

The biquinary code is a 7-bit binary code with error detection capability. Each decimal digit consists of five 0's and two 1's. If less than 2 or more than 2, 1 bits are detected by the receiver, it signals an error.

Error Detection Codes

Definition 2.16 (Parity Bit). *Is an extra bit included with the message to make the total number of 1's transmitted either even or odd.*

`GRAY code` - Advantage of GRAY code over binary is that only 1 bit changes when a number is incremented by 1. A typical application is when analog signals are used to represent the continuous movement of shaft position.

`ASCII` - Used to represent characters and commonly used in most systems. Is of 7 bits allowing 128 characters (extended ASCII has 8 bits). A-Z are from 65-90, a-z are from 97-122 and 0-9 are from 48-57.

2.2.8 Computer Arithmetic

Carry and Overflow condition

Carry and Overflow are two very similar term in digital logic. But we should not be confused between two. Carry is useful when we add or subtract two binary digits which are in unsigned format. Overflow is used when we add or subtract two terms which are in signed format.

But usually both flags are in place in the architecture and the required one is used based on the types of the operands (compiler generates code based on this).

Carry flag is set when there is a carry from the most significant bit position – this is why it is useful for unsigned numbers only.

The condition for overflow, $C_{out} \oplus C_{n-1} = 1$ (Carry out of the MSB `XOR`ed with carry in to the MSB). This is same as saying MSB of both the numbers to be added are 1 and that of the sum is 0, or MSB bits of both the numbers are 0 and that of the sum is 1. Overflow is an error in 2's complement arithmetic for signed numbers and ignored for unsigned numbers.

2.2.9 Real Number representations and computer arithmetic (fixed and floating point

1. **R**eal Numbers can be represented in fixed and floating point numbers. See 2.3 for fixed point representation. Floating point number representation consists of three parts
   a) Sign bit - 1 bit is enough
   b) Exponent
   c) Mantissa
2. single precision floating point number represented in 32 bits
3. double precision floating point number represented in 64 bits

As of now the most widely used floating point representation is IEEE 754. It has a single precision representation using 32 bits and a double precision one using 64 bits. Some of the most important features are (assuming single precision)

- 1 sign bit to the left most
- Followed by 8 exponent bits
- Followed by 23 mantissa bits

IEEE 754 assumes normalized representation if exponent is non-zero (when exponent is 0, number is considered denormalized). Normalized means the number starts like 1.xx. Now, since for a normalized representation 1 is always there at left position IEEE 754 ignores it and does not represent this in the mantissa bits. This allows it to have a precision of 24 bits even though only 23 mantissa bits are there. Only issue being implicit 1 restricts the smallest positive (or largest negative) number that can be represented.

Exponent is of 8 bits. So, the number of exponent values is 256. In order to represent negative exponent values, a bias "127" is used and the exponent value is always subtracted by this. i.e., 0 becomes -127 and 255 becomes +128. But these two values have special meanings (see 2.3), and normal exponent range is from -126 to +254.

So, a 32 bit number represented in IEEE 754 as xzy is

$$(-1)^x 1.y \times 2^{z-127}$$

where the mantissa bits y are appended to "1." and z is the decimal value corresponding to the exponent bits.

Example for IEEE 754 Representation

Consider the floating point number 0.75.
Binary value $= 0.11$.
Normalized value $= 1.1 \times 2^{-1}$
So, mantissa $= 0.1$ (Leading 1 implied for normalized numbers)
Exponent $= -1 + 127 = 126$
Sign bit $= 0$
So, the value represented will be

$$\underbrace{0}_{\text{sign}}\ \underbrace{01111110}_{\text{exponent}}\ \underbrace{10000000000000000000000}_{\text{mantissa}}$$

When we convert a decimal number to binary and the value cannot be exactly represented within 24 bits, we loose precision. If numbers can be exactly represented as the example above, we have no loss in precision.

2.2.10 Booth Algorithm

Consider an example: 01011×1110 which in decimal is $11 \times 14 = 154$. Usually if we do normal multiplication we add the multiplicand as Partial Product corresponding to each '1' in the multiplier. Thus for the given example we have 3 ADD operations. Booth Algorithm can better this by considering "groups of 1s" as to individual 1's. It can be shown by the following steps for the given example. Multiplicand (M): 01011, Multiplier (m): 01110 - 5 bits each, Product can be up to 10 bits. 2's complement of $M = 10101$

1. Initialize Product = Multiplier and top 5 bits as 0's. Also assume $P_{(-1)} = 0$

$$P = 00000\ 01110$$

2. For $i = 0$ to 4 do steps 3 and 4
3. $P_U = P_U + P_0 - P_{(-1)}, P_U$ denoting the upper 5 bits of P.
4. Arithmetic Shift Right P one bit (Leftmost bit replaced by carry bit, $P_{(-1)} = P_0$).
   For $i = 0$, $P_{(-1)} - P_0 = 0$, no ADD, ASR one bit

$$P = 00000\ 00111\ 0$$

   For $i = 1$, $P_{(-1)} - P_0 = -1$, SUB M (adds 2's complement) from left half of P

$$P = 10101\ 00111\ 0$$

   ASR one bit

$$P = 11010\ 10011\ 1$$

   For $i = 2$, $P_{(-1)} - P_0 = 0$, no ADD, ASR one bit

$$P = 11101\ 01001\ 1$$

   For $i = 3$, $P_{(-1)} - P_0 = 0$, no ADD, ASR one bit 11110 01011

$$P = 11110\ 10100\ 1$$

For $i = 4$, $P_{(-1)} - P_0 = 1$, ADD M to left half of P

$$P = 01001\ 10100\ 1$$

ASR one bit

$$P = 00100\ 11010\ 0$$

The final P is the required result which equals 154. We got result in 2 ADD operations (considering SUB as ADD of 2's complement)

m_i	m_{i-1}	Operation	$m_{i-1} - m_i$
0	0	Do nothing	0
0	1	Add M	1
1	0	Subtract M	-1
1	1	Do nothing	0

Note 2.17. Worst case of Booth's algorithm happens for alternate sequence of 1's and 0's in which case we require $b/2$ ADD operations and $b/2$ SUB operations for $b - bit$ multiplication.

Radix-4 Booth Recoding

Asked in GATE 2006 IT

This is a modification to Booth algorithm where we look at 3 bits at a time instead of 2 in normal Booth's algorithm.

m_{i+1}	m_i	m_{i-1}	Partial Product
0	0	0	0
0	0	1	1 * Multiplicand
0	1	0	1 * Multiplicand
0	1	1	2 * Multiplicand
1	0	0	-2 * Multiplicand
1	0	1	-1 * Multiplicand
1	1	0	-1 * Multiplicand
1	1	1	0

We need some adjustments here before doing multiplication compared to previous case. We need to make the multiplicand even no. of bits by extending the sign bit to left if required.

Multiplicand (M): 001011, Multiplier (m): 001110 - 6 bits each, Product can be up to 12 bits. 2's complement of $M = 110101$

1. Initialize Product = Multiplier and top 6 bits as 0's. Also assume $P_{(-1)} = 0$

$$P = 000000\ 001110$$

2. For $i = 0$ to 2 do steps 3 and 4
3. $P_U = P_U + PP$, P_U denoting the upper 5 bits of P and PP corresponding to the partial product given be the least 3 bits (including -1 position).
4. Arithmetic Shift Right P two bits (Leftmost 2 bits replaced by carry bit, $P_{(-1)} = P_0$).
   In our case:
   For $i = 0$, $P_1P_0P_{(-1)} = 100$, $PP = -2 * M$,

$$P = 101010\ 001110\ 0$$

ASR two bits

$$P = 111010\ 100011\ 1$$

For $i = 1$, $P_1P_0P_{(-1)} = 111$,$PP = 0$
ASR two bits

$$P = 111110\ 101000\ 1$$

For $i = 2$, $P_1P_0P_{(-1)} = 001$,$PP = 1 * M$

$$P = 001001\ \ 101000\ 1$$

ASR two bits

$$P = 000010\ 011010\ 0$$

The final P is the required result which equals 154. We got result in 2 ADD operations (considering SUB as ADD of 2's complement)

Here, we got the same number of ADD operations as the normal Booth's algorithm. But even in worst case, Booth recoding gets us the result in $b/2$ ADD/SUB operations as compared to b in the case of normal Booth algorithm.

2.3 Web Links

1. Webpage for Digital Logic
2. Previous GATE Questions in Digital Logic
3. GATECSE Discussion lead by Praveen Saini and Ravi Singh NPTEL - Recommended Video
4. IEEE 754 - Must see
5. Multipliers
6. Number Representation
7. Fixed Point Representation

Important Problems

1. `http://gateoverflow.in/65751/rs-complement`
2. `http://gateoverflow.in/29729/hexadecimal-subtraction`
3. `http://gateoverflow.in/31359/minimum-nand-gates-realization-exor-exnor-adder-subtractor`
4. `http://gateoverflow.in/28723/minimum-number-of-gates`
5. `http://gateoverflow.in/35739/number-of-nor-gates`
6. `http://gateoverflow.in/2206/gate2010_32`
7. `http://gateoverflow.in/17407/gate1992_04_b`
8. `http://gateoverflow.in/39575/gate-2016-2-07`
9. `http://gateoverflow.in/264/gate2005_62`
10. `http://gateoverflow.in/2776/gate1996_24`- beautiful question
11. `http://gateoverflow.in/39670/gate-2016-1-8`
12. `http://gateoverflow.in/1814/gate2006-38`

2.4 Exercise Solutions

3 Conclusion

"I always thought something was fundamentally wrong with the teaching system in India. So why not change it."

FAQs

References

[1] M. Morris Mano. *Digital Logic and Computer Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 1979. ISBN 0132145103.