ISSN 2278-3091
Volume 9, No.1, January – February 2020
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse122912020.pdf
https://doi.org/10.30534/ijatcse/2020/122912020

Design of Sequence Detector using Finite State Machine

S. G. Mundada[1], H. M. Agrawal[2]
[1]Assistant Professor, Department of Computer science Engineering, Shri Ramdeobaba College of Engineering and Management, Nagpur, India, mundadasg@rknec.edu
[2]Assistant Professor, Department of Computer science Engineering, Shri Ramdeobaba College of Engineering and Management, Nagpur, India, agrawalhm2@rknec.edu

ABSTRACT

Automata Theory is a tool which is used in multidisciplinary computing and scientific research. It is the basis behind the traditional model of computation and is used for many purposes such as controller circuit design, sequential circuit design etc. A Sequence detector is a sequential state machine used to detect consecutive bits in a binary string.. This technical paper examines various sequences and gives output as 1 if the sequence is detected. The types of the sequence examined are overlapping sequences. Realization of the designed algorithm for detecting the sequence is with the help of flip-flops. The flip-flops help to detect the pattern in the given string.

Key words : Finite automata with output, Sequence detector, Mealy Machine, Moore Machine, VHDL

1.INTRODUCTION

A **Finite Automata**, is a mathematical model of computation. The finite automata can change from one state to another in response to some external inputs; the change from one state to another is called a **transition**. A finite automata is defined by a list of its states, its initial state, and the conditions for each transition. [1].

The behavior of finite automata can be observed in many devices in modern society that perform a predetermined sequence of actions depending on a sequence of events with which they are presented. Examples are vending machines (event triggered), which dispense products when the proper combination with correct sum of coins are deposited, elevators (event triggered), whose sequence of stops is determined by the floors requested by riders, traffic lights (time triggered), which change sequence when cars are waiting, and combination locks (event triggered), which require the input of combination numbers in the proper order.

The finite automata has less computational power than some other models of computation such as the Turing machine. The computational power distinction means there are computational tasks that a Turing machine can do but a finite automata cannot. This is because a finite automata's memory is limited by the number of states it has. Finite automata are studied in the more general field of automata theory.

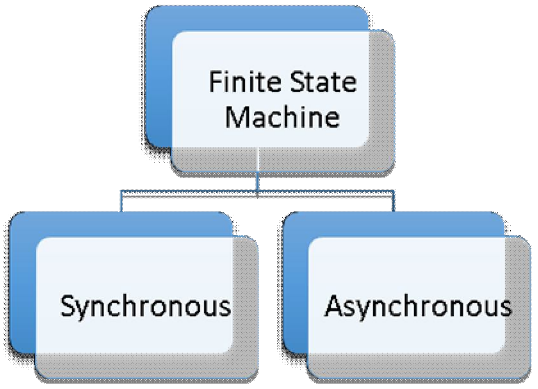


Figure 1: Synchronous & Asynchronous Finite Automata

A **Synchronous Finite Automata** is a finite automata whose output depends on the clock i.e. synchronous finite automata make use of clock to move from one state to next. Using a **clock** to control the synchronous movement between one state and the next allows the finite automata logic time to settle before the next transition and, hence, overcomes some logic delay problems that may arise. For this reason, synchronous systems are, by far, the most popular in digital electronics; and most HDLs used to define them are optimized for synchronous system design.

An **Asynchronous Finite Automata** is a finite automata whose transition between states is controlled by the eventinputs, so that the finite automata does not need to wait for a clock signal input. For this reason, asynchronous finite automata are sometimes called ‘event-driven’ finite automata.

2.SEQUENCE DETECTOR

Sequence detection is the act of recognizing a predefined series of inputs. Following example looks for a series of ones and zeroes similar to 1011. The overwhelming beauty of programmable logic is its ability to work with DSP (Digital Signal Processing). The capability of processing a great deal of information quickly cannot be overemphasized. Suppose one is working at the VLA in Socorro and they wants to scan thousands of bits of digital information looking for a series of bits that represents an RF signal from a distant galaxy. Or, wants to scan a digital image of a human lung looking for a digital signature that could represent a cancerous cell. These are very broad interpretations but it can be noticed where the benefit comes with being able to recognize a specific sequence. A sequence detector is a sequential circuit which is basically a circuit that can store information between operations. There are two main models for sequential circuits: Mealy and Moore model. A Mealy model circuit the output depends on the inputs and the state of the system, in a Moore model, the output of the system only depends on its state.[2].

There are basically two types of sequence detector depending on the type of sequence they identify, which are as follows:

- **Overlapping Sequence Detector:** In a sequence detector that allows overlap, the final bits of one sequence can be the start of another sequence. For example will be an 1101sequence detector. It raises an output of 1 when the last 4 binary bits received are 1101.
- **Non-Overlapping Sequence Detector:** The sequence detector with no overlap allowed resets itself to the start state when the sequence has been detected. After the initial sequence 1101 has been detected, the detector with no overlap resets and starts searching for the initial 1 of the next sequence.

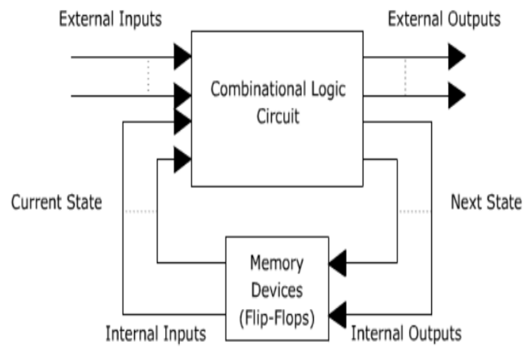


Figure 2: Sequence Detector

3.MEALY MACHINE AND MOORE MACHINE

Mealy machine: A simple Mealy machine has one input and one output. Each transition edge is labelled with the value of the input (shown in red) and the value of the corresponding output (shown in blue). The machine starts in state *S*i. (In this example, the output is the exclusive-OR of the two most-recent input values; thus, the machine implements an edge detector, outputting a one every time the input flips and a zero otherwise.)[3].

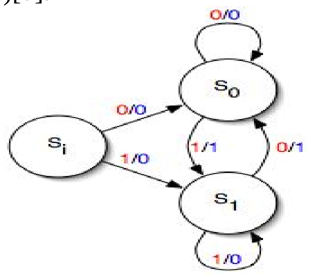


Figure 3 :A Simple Mealy Machine

Moore machine: Simple Moore machines have one input and one output. Output depends only upon the present state. Generally, it has more states than Mealy Machine. Input change can cause change in output change as soon as logic is done. In Moore machines, more logic is needed to decode the outputs since it has more circuit delays. [4].

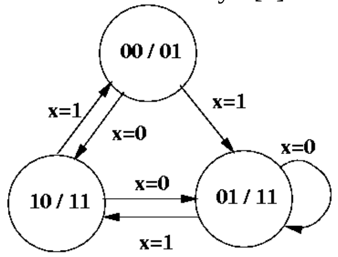


Figure 4: A Simple Moore Machine

4.VHDL

Most hardware designers use hardware description languages (HDLs) to describe designs at various levels of abstraction. A hardware description language is a high level programming language, with programming constructs such as assignments, conditions, iterations and extensions for timing specification, concurrency and data structure proper for modeling different aspects of hardware. The most popular hardware description languages are VHDL [4] and Verilog [5]. VHDL (VHSIC (Very High Speed Integrated Circuits) Hardware Description Language) [4] is an IEEE Standard since 1987 while Verilog was standardized in 1995.

VHDL is a programming language used to model a digital system by dataflow, behavioral and structural style of modeling. This language was first introduced in 1981 for the department of Defense (DoD) under the VHSIC program. It was originally developed under contract F33615-83-C-1003 from the United States Air Force awarded in 1983 to a team with Intermetrics, Inc. as language experts and prime contractor, with Texas instruments as chip design experts and IBM as computer system design experts. The language has undergone numerous revisions and has a variety of sub-standards associated with it that extend it in important ways.

5.FLIP-FLOPS

A flip-flop or latch is a circuit that has two stable states and can be used to store state information. A flip-flop is a bistable multivibrator. The circuit can be made to change state by signals applied to one or more Control inputs and will have one or two outputs, one for the normal value and one for the complement value of the stored bit. Memory elements in any sequential circuit are usually flip-flops.

The flip flops can be divided into 4 types:

- Set-Reset(SR) Flip-Flop
- Delay(D) Flip-Flop
- Toggle(T) Flip-Flop
- JK Flip-Flop

6.KARNAUGH MAP (K-MAP)

A Karnaugh map (K-map) is a pictorial method used to minimize Boolean expressions without having to use Boolean algebra theorems and equation manipulations. A K-map can be thought of as a special version of a truth table.

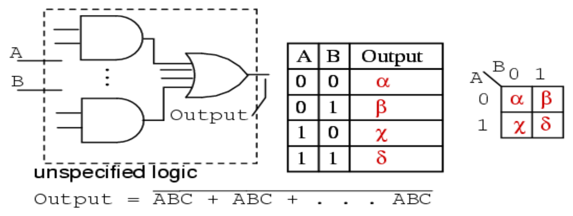
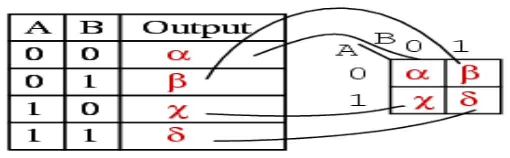


Figure 5 : K Map Representation

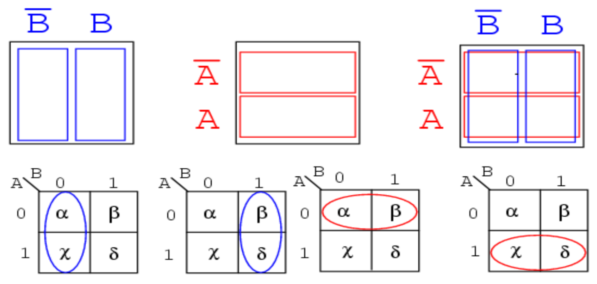
Four items as shown above, are just different ways of representing the same thing: an arbitrary 2-input digital logic function. First is logic gates, a truth table, a Karnaugh map, and a Boolean equation. The point is that any of these are equivalent. Two inputs A and B can take on values of either 0 or 1, high or low, open or closed, True or False, as the case may be. There are 2^2 = 4 combinations of inputs producing an output. This is applicable to all five examples.

These four outputs may be observed on a lamp in the relay ladder logic, on a logic probe on the gate diagram. These outputs may be recorded in the truth table, or in the Karnaugh map. Look at the Karnaugh map as being a rearranged truth table. The Output of the Boolean equation may be computed by the laws of Boolean algebra and transfered to the truth table or Karnaugh map.



The outputs of a truth table correspond on a one-to-one basis to Karnaugh map entries. Starting at the top of the truth table, the A=0, B=0 inputs produce an output α. Note that this same output α is found in the Karnaugh map at the A=0, B=0 cell address, upper left corner of K-map where the A=0 row and B=0 column intersect. The other truth table outputs β, χ, δ from inputs AB=01, 10, 11 are found at corresponding K-map locations.

Below, we show the adjacent 2-cell regions in the 2-variable K-map with the aid of previous rectangular Venn diagram like Boolean regions.



Cells α and χ are adjacent in the K-map as ellipses in the left most K-map below. Referring to the previous truth table, this is not the case. There is another truth table entry (β) between them. Which brings us to the whole point of the organizing the K-map into a square array, cells with any Boolean variables in common need to be close to one another so as to present a pattern that jumps out at us. For cells α and χ they have the Boolean variable B' in common. We know this because B=0 (same as B') for the column above cells α and χ. Compare this

ı B
) in
ı.

aly

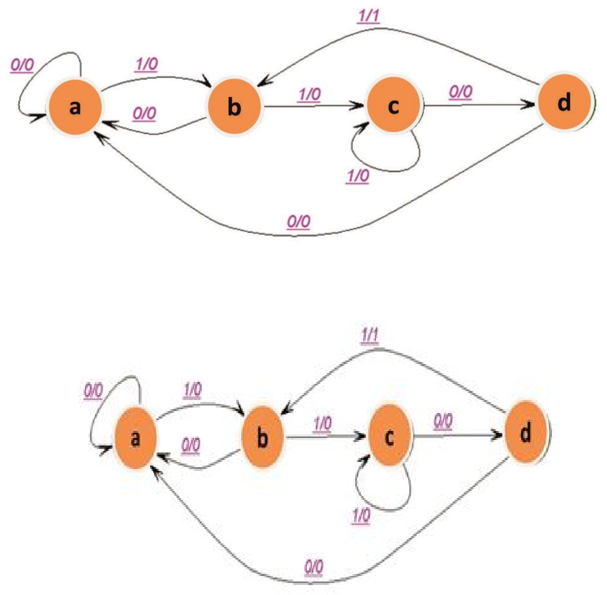


Figure 6 : Sequence Detector 1101

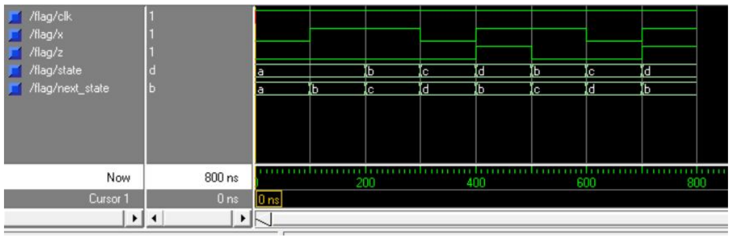
Implementation Details:

A. **Behavioral Modeling:**

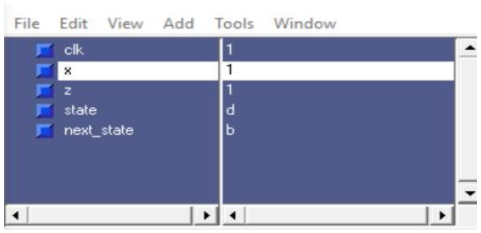
```
entity flag is port (
    clk,x:    in bit;
    z:        out bit);
end flag

architecture mealy of flag is
     type states is (a,b,c,d);
     signal state:  states := a;                -- initial value is a
     signal next_state:   states := a;          -- initial value is
              a begin
clock: process(clk)
         begin
           if clk'event and clk = '1' then
           state <= next_state;
          end if;
         end process clock;
state_trans:  process(state,x)  --reacts to changes in state and x
begin
next_state <= state;                            --update next state
                 case state is
              when  a  =>  if  x  =  '0'
                        then z <= '0';
                        next_state <=a;
                   else
                        next_state <= b;
                        z <= '0';
                   end if;
              when  b  =>  if  x  =  '1'
                        then      next_state
                        <= c;
                        z <= '0';
                   else
                        z <= '0';
                        next_state <= a;
                   end if;
              when  c  =>  if  x  =  '1'
                        then      next_state
                        <= c;
                        z <= '0';
                   else
                        next_state <= d;
                        z <= '0';
                   end if;
              when  d  =>  if  x  =  '0'
                        then      next_state
                        <= a;
                        z <= '0';
                   else
                        next_state <= b;
                        z <= '1';
                    end if;end case;
end process state_trans;
end mealy;
```

String Accepted –
Input - **"01101101"**
Output - **"00001001"**

The above string contains two instances of 1101. Hence, the output string contains two '1' at positions 5 and 8 respectively.



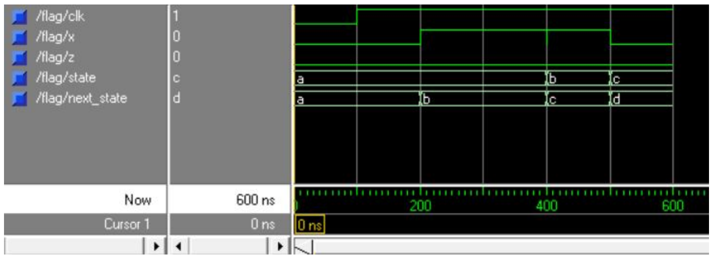
/flag/clk	1
/flag/x	1
/flag/z	1
/flag/state	d
/flag/next_state	b
Now	800 ns
Cursor 1	0 ns



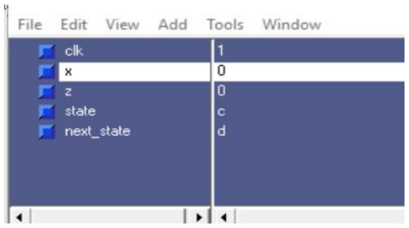
clk	1
x	1
z	1
state	d
next_state	b

String Not Accepted –
Input - **"001110"**
Output - **"000000"**

The above string do not contain any instance of 1101. Hence, the output has no zeroes.



/flag/clk	1
/flag/x	0
/flag/z	0
/flag/state	c
/flag/next_state	d
Now	600 ns
Cursor 1	0 ns



clk	1
x	0
z	0
state	c
next_state	d

B. Using Flip-flops

The following example generates a simple dataflow program in VHDL using **JK Flip-flop .**From the state diagram **state table** can be generated. The state table is as follows:

Present State	Next State		Output	
	X=0	X=1	X=0	X=1
S0	S0	S1	0	0
S1	S0	S2	0	0
S2	S3	S2	0	0
S3	S0	S1	0	1

The **binary representation** for above state table is as follows:

Present State	Next State				Output	
	X=0		X=1		X=0	X=1
	A1	B1	A2	B2		
0 0	0	0	0	1	0	0
0 1	0	0	1	0	0	0
1 0	1	1	1	0	0	0
1 1	0	0	0	1	0	1

Now to implement sequence detector flip-flops are require. Let the two required flip-flops be JK flip-flops. The above table along with JK flip-flop can be written as:

Present State		Input (x)	Next State		Input to Flip-flop				Output
A	B		A	B	Ja	Ka	Jb	Kb	
0	0	0	0	0	0	X	0	x	0
0	0	1	0	1	0	X	1	X	0
0	1	0	0	0	0	X	X	1	0
0	1	1	1	0	1	X	X	1	0
1	0	0	1	1	X	0	1	X	0
1	0	1	1	0	X	0	0	X	0
1	1	0	0	0	X	1	X	1	0
1	1	1	0	1	X	1	X	0	1

The equations for flip flops are determined using **K-Maps**. The k-map for the above flip flop:

AB / X	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{X}$	0	2	6 X	4 X
X	1	3 1	7 X	5 X

$J_A = BX$

AB / X	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{X}$	0 X	2 X	6 1	4
X	1 X	3 X	7 1	5

$K_A = B$

AB / X	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{X}$	0	2	6 X	4 1
X	1 1	3 X	7	5

$J_B = A\bar{X} + \bar{A}X$

AB / X	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{X}$	0 X	2 1	6 1	4 X
X	1 X	3 1	7	5 X

$K_B = \bar{A} + \bar{X}$

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{X}$	0	2	6	4
X	1	3	7 1	5

$Y = ABX$

Circuit diagram for sequence detector is as follows:

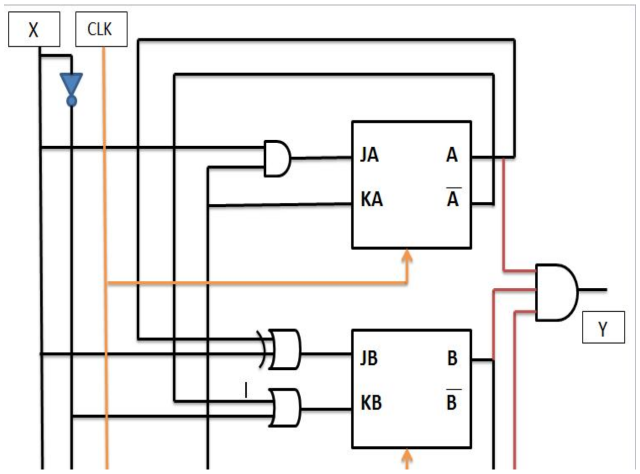
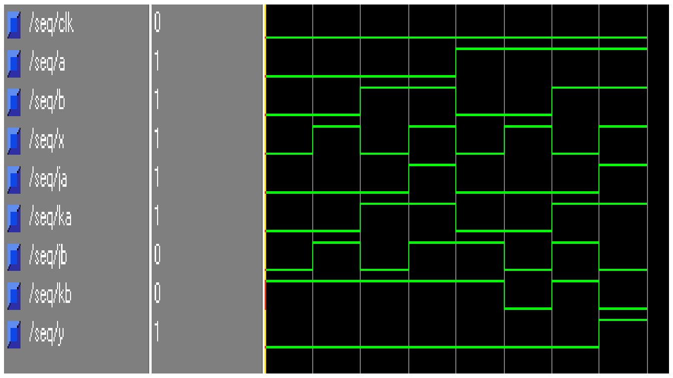


Figure 7: Circuit Diagram

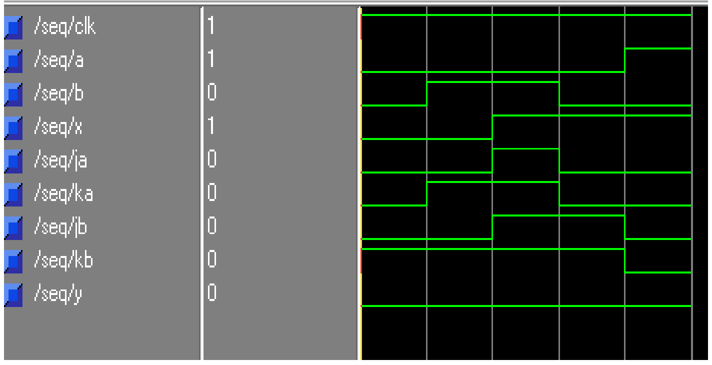
```
entity seq is
port (clk,a,b,x: in bit; ja,ka,jb,kb,y: out bit);
end seq;
architecture dataflow of seq is begin
                ja<=b and x;
                ka<=b;
                jb<=a xor x;
                kb<=(not x) or (not a);
                y<=a and b and x;
end dataflow;
```

String Accepted-
Input : **"0111101"**
Output : **"0000001"**



When the string 0111101 is scanned, the output at the end is one .Which means the desired sequence is detected and accepted.

String Not Accepted –
Input - **"0001"**
Output - **"0000"**



When the string 0001 is scanned, output is 0 at the end. This means that given input does not contain sequence 1101 and hence it is not detected. The output is always 0 because there is no sequence 1101 as input.

8.CONCLUSION

This paper shows the relationship between finite state machines and VHDL coding. A fundamental motivation to use VHDL is that it is a standard, technology independent language, and is therefore portable and reusable. The VHDL code for sequence detector 1101 has been successfully designed and implemented using two different approaches (i.e. via behavioral and by using flip-flop) through this paper.

This paper also helps in explaining the concept related to theory of automata .It has many applications. It can be used in the following:

- Designing ring counter,
- Serial adder,
- Serial data,
- Schmitt trigger.

REFERENCES

1. Hopcroft, J. E., Ullman, J. D., "Introduction to Automata Theory,Languages, and Computation", Addison-Wesley, 1979.
2. Golson, S. , "State Machine Design Techniques for Verilog and VHDL", Synopsys Journal of High-Level Design, pp. 1-2,1994
3. Mealy, G. H., "A method for synthesizing sequential circuits," Bell System Tech. J., Vol. 34, No. 5, pp. 1045–1079, 1955 https://doi.org/10.1002/j.1538-7305.1955.tb03788.x
4. Moore, E. F., "Gedanken experiments on sequential machines," Automata Studies. Princeton, NJ: Princeton University Press, pp.129–153, 1956. https://doi.org/10.1515/9781400882618-006
5. Mealy, G. H., "A method for synthesizing sequential circuits,"

Bell System Tech. J., Vol. 34, No. 5, pp. 1045–1079, 1955
https://doi.org/10.1002/j.1538-7305.1955.tb03788.x

6. IEEE Standard 1076-1993, IEEE Standard Description Language based on the VHDL Hardware Description Language, 1993.
7. IEEE Standard 1364-2001, IEEE Standard Description Language based on the Verilog Hardware Description Language, 2001.
8. Perrin, “Finite Automata”, Handbook of Theoretical Computer Science. Elsevier Science Published 1990.
9. H. R. Lewis, C. H. Papadimitriou.,”Elements of the Theory of Computation”, Prentice Hall 1981.
10. John C.Martin , ”Introduction to Languages and Theory of Computation”,Third edition, Tata McGraw Hill,2009