International Journal of Engineering Science Invention
ISSN (Online): 2319 – 6734, ISSN (Print): 2319 – 6726
www.ijesi.org Volume 2 Issue 4 ‖ April. 2013 ‖ PP.01-06

ASM Based Parity Detection Algorithm for Communication And Networking

Subiya Yaseen[1], Hussain Ahmed[2], Kehkeshan Jallal S[3] ,

[1]*(Department of Electronics & Communication , HKBK College of Engineering , India)*
[2]*(Department of Electronics & Communication , HKBK College of Engineering , India)*
[2]*(Department of Electronics & Communication , HKBK College of Engineering , India)*

ABSTRACT ***:*** *In Digital Systems, Counters can be used to store the number of times an event has occurred. A Bit counting circuit counts the number of 1's in the data. By knowing the size of the data and the number of 1's in the data, one can find what the data is, without trying the Brute Force method. Bit Counting Circuit is associated with ASM Charts including both Datapath and Control circuit. It is of interest in the study of Data Communication and Networking, where the parity bit can be calculated and bit errors can be detected, it is also of interest in Cryptography and Network*

Keywords: *ASM, Bit Counting, Parity , ECC, Cadence*

I. INTRODUCTION

The Algorithmic State Machine (ASM) method [1] is a method for designing finite state machines. It is used to represent diagrams of digital integrated circuits. The ASM diagram is like a state diagram but less formal and thus easier to understand. An ASM chart is a method of describing the sequential operations of a digital system.

The ASM method is composed of the following steps:
(1) Create an algorithm, using pseudo code, to describe the desired operation of the device,
(2) Convert the pseudo code into an ASM chart,
(3) Design the data path based on the ASM chart,
(4) Create a detailed ASM chart based on the datapath,
(5) Design the control logic based on the detailed ASM chart. ASM charts aid in designing Bit counting circuit that counts the number of 1's in the data.

II. A BIT COUNTING METHOD

The different blocks shown in the figure1 were implemented separately and then integrated together. Below are descriptions of each block and specifications for each.
The pseudo-code for a step-by-step procedure, to count the number of bits in a register, A, that have the value 1 is shown in Figure 1. It assumes that A is stored in a register that can shift its contents in the direction from left to right. Answer produced in the algorithm is stored in the variable named B. The algorithm terminates when A does not contain any more 1's, that is when A=0.In each iteration of the while loop,

```
B=0;
While A≠0 do
If a0=1 then
B=B+1;
end if;
Right-shift A;
End while;
```

Fig. 1 Pseudo-code for the bit counter.

If least significant bit (LSB) of A is 1, then B is incremented by 1, otherwise,B is not changed. A is shifted onebit to the right at the end of each loop iteration. Figure 2 gives the ASM chart that represents the algorithm I Figure 1. The state box for the starting state, *S1*, specifies Fig. 2. ASM Chart for the pseudo-codeinFig.1 B is initialized to 0.An assumption is made that an input signal *s*, exists, which is used to indicate when the data to be processed has been loaded into A, so that the machine can start. The decision box labeled *s* stipulates that the machine remains in state *S1* as long as *s* = 0.The conditional output box with Load A written inside indicates that A is loaded from external data inputs if *s* = 0 in state *S1.*

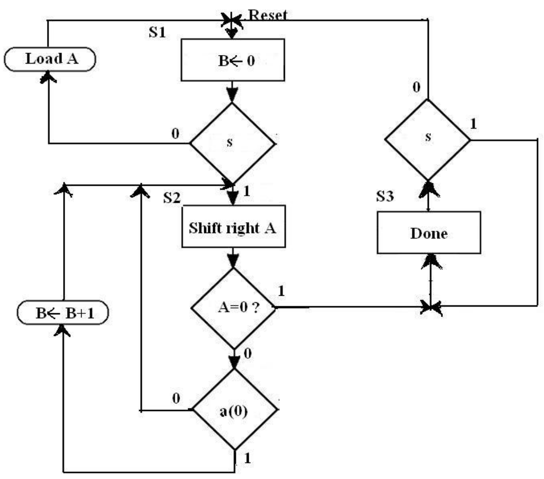


Fig. 2. ASM Chart for the pseudo-code in Fig. 1

When *s* becomes 1, the machine changes to state *S2*.The decision box below the state box for *S2* checks whether A=0.If so, the bit–counting operation is complete, hence the machine should change to *S3*.If not the FSM remains in state *S2*.The decision box at the bottom of the chart checks the value of a_0 .If a_0=1, B is incremented ,which is indicated in the chart as B←B+1.If a_0=0.Then B is not changed. In state *S3* B contains the number of bits in A that were 1.An output signal, *Done* is set to indicate that the algorithm is finished, the FSM stays in *S3* until s goes back to 0.

2.1 Bit Counter in Generalized Ceaser Cipher

Consider data being encoded using Generalized Ceaser Cipher where the algorithm used is C=(P+18)mod 26, C=Cipher text and P=Plaintext and key=18,Key size = 6 bits. In this cipher if Plaintext= hello ,for mathematicsl operations on plaintext and ciphertext,we assign numerical values to each letter(lower or uppercase) as shown in table 1 and 2, then Ciphertext=ZWDDG,here key=18,where Key size = 6 bits.

Table i. Numerical values of alphabets a to m

PT	A	b	c	d	e	f	g	h	i	j	k	L	M
CT	A	B	C	D	E	F	G	H	I	J	K	L	M
V	00	01	02	03	04	05	06	07	08	09	10	11	12

The encryption algorithm adds key to the plaintext and decryption algorithm subtracts key from ciphertext character.Key is any number. In the above example, given data size = 6 bits and using this algorithm ,availability of two 1's in the data is determined. The only numbers to be attempted are 3,5,6,9,10,12,17,18,20,24,33,34,36,40,48 instead of testing all 64 numbers from 0 to 63.Henceforth the key for an encryption operation can be guessed without trying all possible combinations.

2.2 Bit Counter in Parity Error Detection

While digital circuits are much more immune to noise than analog electrical circuits, they are not completely immune from interference. The effect of interference is occasionally to change the value of a signal from 0 to 1 or from 1 to 0. We call this a bit flip. One technique that is often used for error detection is parity, which refers to the number of bits that are 1 in a code word. Parity error checking involves increasing the code length by one bit, called the parity bit. In the even parity scheme, the parity bit in each augmented code word is set to 0 or 1 to ensure that the total number of 1 bits is even. For example, if the original code word is 1011, the augmented code word is 10111.In an even parity scheme, valid augmented code words have even parity, and invalid augmented code words have odd parity. If interference causes a 0 bit to change to 1, the number of 1bits is increased by one, making the parity odd. Similarly, if interference changes a 1 bit to 0, the number of 1 bits is decreased by one, again making the parity odd. So to check whether a bit has flipped, the number of 1 bits are counted, including the parity bit. If the count is odd, parity has been reversed, so an error has occurred. If the count is even, either no error has occurred, or an even number of bits have been flipped, which we can't detect. In many applications, the probability of two or more bits flipping is much lower than the probability of one bit flipping, so it is acceptable not to be able to detect an even number of bit flips.

A common approach is to use a parity tree, is to XOR all the bits in the information as shown in figure 3, For example, if the original code word is 1001 then parity bit p=1^0^0^1= 0, the augmented code word is 10010 since it keeps the overall propagation delay small and avoids using gates with large numbers of inputs. The tree at the left of the figure 3 generates the parity bit to augment an 4-bit code, creating a code of five bits with even parity. The tree at the right checks the augmented code and yields a 1 if there is a parity error.

Hence the error can be easily detected using the bit counting circuit.

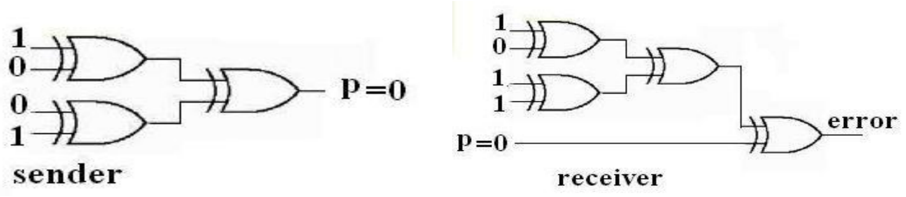


Fig. 3 Parity trees for generating (left) and checking (right) even parity for a 4 bit code 1001 with an error in the third bit.

2.3 Bit Counter in Error Detection and Correction using Errorcorrecting code (ECC)

In the case of memories, use of parity involves adding an extra bit cell to each memory location. When we write to a location, we compute the parity bit and store it in the extra cell. When we read a location, we check that the data, together with the parity bit, have the correct parity. If so, we assume the data is uncorrupted. Otherwise, we take appropriate action to deal with the error in the stored data.The problem with using parity to check for errors, is that it only allows us to detect a single bit flip in a stored code word. It does not allow us to identify which bit flipped, nor does it allow us to detect an even number of bit flips [2][3]. If we could identify the particular bit that flipped, we could correct the error by flipping the bit back to its original value, and then continue operating as normal. We could In order to be able to identify which bit flipped on occurance of an error, we need to consider the invalid code words that result from flipping each bit of each valid code word. Provided all of those invalid code words are distinct, we can use the value of the invalid code word to identify the flipped bit. One scheme for doing this is to use a form of *error correcting code* known as a *Hamming code* [4]. Since each data ECC bit has at least two 1 bits in its binary index (otherwise it would be a check bit), each data bit is included in the computation of at least two check bits. When the memory location is read, again, the entire ECC word is read. We recomputed the values of the check bits from the data ECC bits and compare them, using a bit-wise exclusive OR, with the check bits read from memory.

If the comparison result is 0000, the recomputed check bits match the read check bits, so all is well. However, if one of the stored ECC bits (either a data bit or a check bit) is flipped from the original, the comparison result, called the *syndrome*, will be other than 0000. It turns out to be the binary index of the ECC bit that has flipped. Thus, we can use the syndrome value to correct the error by flipping the indexed bit back.
Consider to Determine whether there is an error in the ECC word 000111000100, and if so, we need to correct it. The check bits computed from the data bits of the ECC word are

$e1=e3\oplus e5\oplus e7\oplus e9\oplus e11= 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$

$e2=e3\oplus e6\oplus e7\oplus e10\oplus e11= 1 \oplus 0 \oplus 1 \oplus 0 \oplus 0 =0$

$e4=e5\oplus e6\oplus e7\oplus e12= 0 \oplus 0 \oplus 1 \oplus 0 =1$

$e8=e9\oplus e10\oplus e11\oplus e12=0 \oplus 0 \oplus 0 \oplus 1 =1$

The syndrome is $1101 \oplus 1000 = 0101$. Thus, there is an error in bit *e*5 of the read ECC. That bit should be flipped back from 0 to 1, giving the corrected

ECC word 000111010100. Note that we have assumed that only one bit of the stored ECC word could be in error. If two or more bits flip, the checking process may incorrectly identify a single bit as having flipped, or it may yield an invalid syndrome. The problem arises from the fact that we have insufficient invalid code words to distinguish between single-bit errors and double-bit errors. A simple remedy is to add further check bits. If we add a check bit that is the exclusive-OR of all of the data bits, the resulting error-checking code allows us to correct any single-bit error and to detect (but not correct) any double-bit error. If we assume that errors are independent, the probability of a double-bit error is very low, so this scheme suffices in many applications. If extreme reliability and resilience to errors is required, we can further extend the error-checking code to enable correcting of multiple-bit errors. The details of how we might do this are beyond the scope of this book, referred for Further Reading.

III. ASM CHART IMPLIED TIMING INFORMATION

Bit counting circuit can be illustrated using the concept that the ASM chart implies timing information.Consider the ASM block for state S2.In traditioal flowchart ,when state S2 is entered the value of A would first be shifted to the right.Then the value of A is examined and if A's LSB is 1,immediately1 is added to B.But since the ASM chart represents the sequential circuit,changes in A and B,which represent the outputs of flip flops ,take place after the active clock edge.The same clock signal that controls the changes in the state of the machine also controls changes in A and B.Hence in state S2 the decision Boxes that check the conditions A=0 and A(0)=0 are executed before shifting the contents of A.If A=0 then the FSM will change to S3 on the next clock edge .On the other hand if A≠0,then the FSM does not change to S3, but remains in S2.At the same time ,A is still shifted, and B is incremented if A(0)=1.

3.1 . Data path Circuit

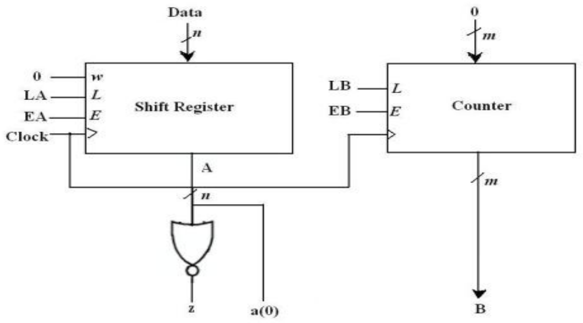


Fig. 4.Datapath for the ASM Chart in Fig. 2

By examining the ASM chart for the bit-counting circuit,the type of circuit elements needed to implement its datapath can be infered[5][6].Requirements include a shift register to shift from left to right to implement A.It must have the parallel-load capability because of the conditional output box in state *S1* that loads data into the register.An enable input is also required because shifting should occur only in state *S2*.A counter is needed for B,and it needs a parallel-load capability to initialize the count to 0 in state *S1*.It is not wise to rely on the counter's reset input to clear B to 0 in state *S1*.In practice the reset signal is used in a digital system for only two purpose: to initialze the circuit when power is first applied,and to recover from an error.The machine changes from state *S3* to *S1* as a result of *s* = 0. The datapath is depicted in Figure 3The serial input to the shift register,w, is connected to 0 because it is not needed.The load and the enable inputs on the shift register are driven by the signals LA and EA. The parallel input to the shift register is named *Data*, and its parallel output is A. An *n*-input NOR gate is used to test whether A =0 .The output of this gate,*z*, is 1 when A =0.Note that the figure indicates the n-input NOR gate by showing a single input connection to the gate,with the lable *n* attached to it. The counter has $m=log_2(N)+1$ bits, where $N= 2^x > n$, (e.g if n=1018,then N= 2^{10}> 1018,thus m=$\log_2(2^{10})+1$=11 bits), with parallel inputs connected to 0 and parallel outputs named B.It also has a parallel load input LB and enable input EB control signals

3.2 . Control Circuit

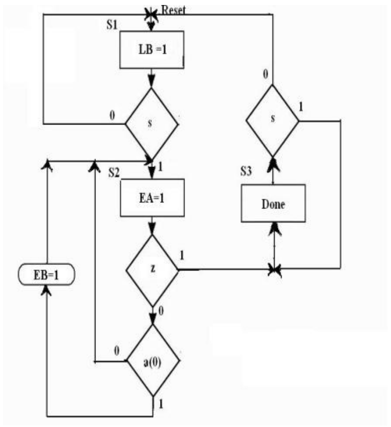


Fig. 5. ASM chart for the bit counter data path circuit

For convenience,second ASM chart that represents only the FSM needed for the control circuits [5][6] is shown in Figure 4.The FSM has the inputs s,a_0, and z that generates the output EA,LB,EB, and Done.In state S1,LB is asserted,so that 0 is loaded in parallel into the counter.For the control signals like LB,instead of wriring LB=1, LB is used to indicate that the signal is asserted .Assume that the external circuitry derives LA to 1 when valid data is presnent at the parallel inputs of the shift register,so that the shift register contents are initialized before *s* changes to 1. In state *S2* ,EA is asserted to cause a shift operation,and the count enabled for B is aserted only if a_0=1.

IV. CIRCUIT IMPLEMENTATION AND RESULTS

In order to experimentally verify the proposed scheme, a bit counting circuit is implemented in Quartus ii 7.1 and Cadence tool [7].

4.1 Simulation Results

The algorithm is described in Verilog code [8]. The code is tested for data input of n bits and simulation results are obtained in Quartus II 7.1 ,where the output variable B stores the No. of 1's in data and output is as shown in Figures 6,7,8,9 an Figure 10 for bit count, parity detection and error correction .

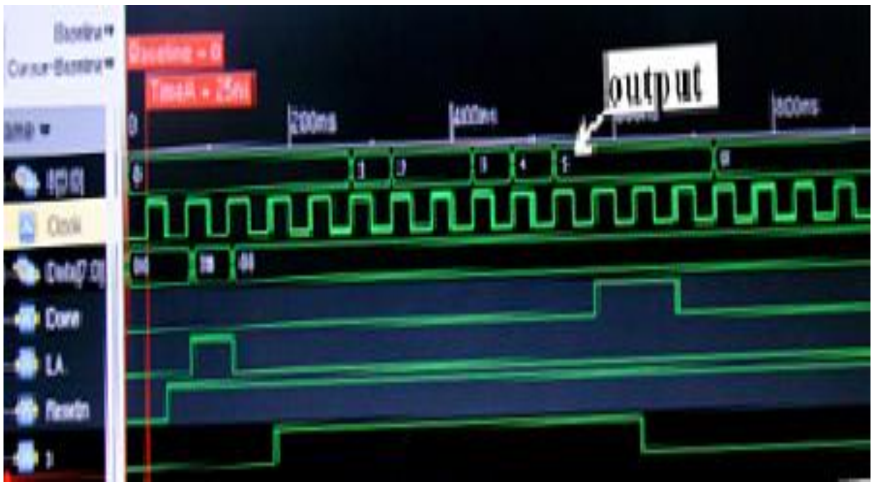


Fig 6: Simulation results with data as 3B and Bit counter output 5 in Cadence Simulator

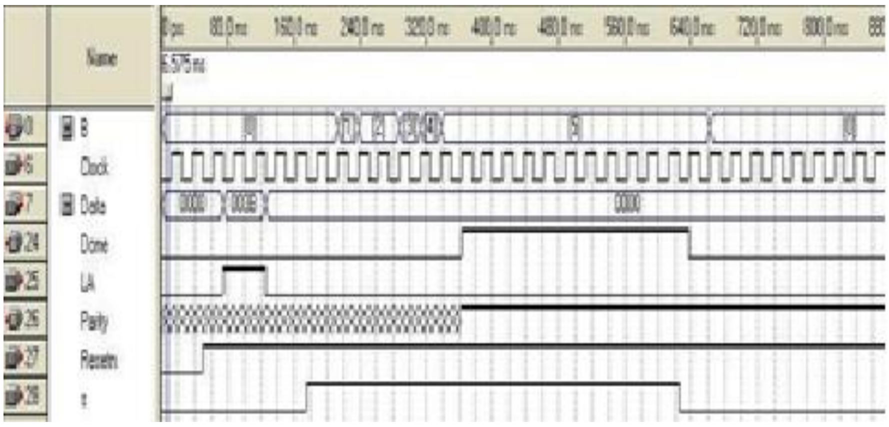
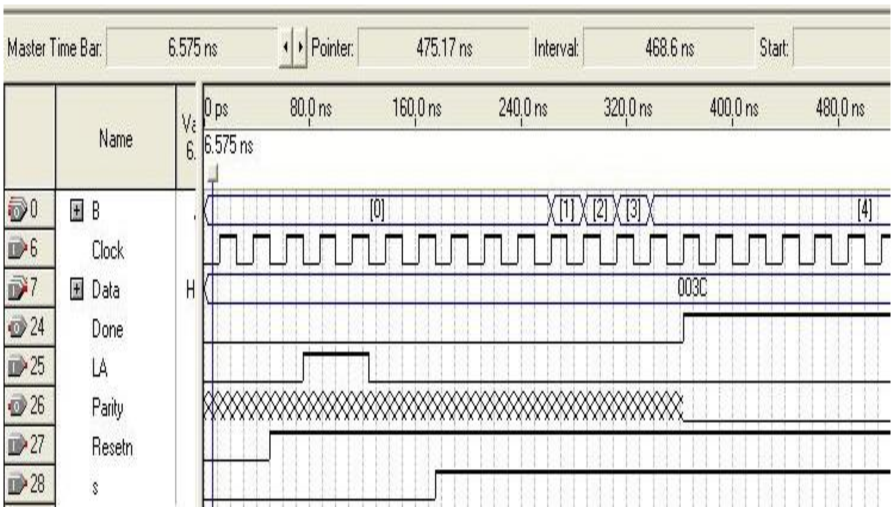


Fig.7. Simulation results of bit counter with parity bit set to 1 due to odd number (5) of one's in the input data 003B ,output as seen in Quartus II 7.1 Simulator.



Master Time Bar: 6.575 ns Pointer: 475.17 ns Interval: 468.6 ns Start:

Fig. 8. Simulation results of bit counter with parity bit set to 'zero' due to even number (4) of one's in the input data 003C ,output as seen in Quartus II 7.1 Simulator.

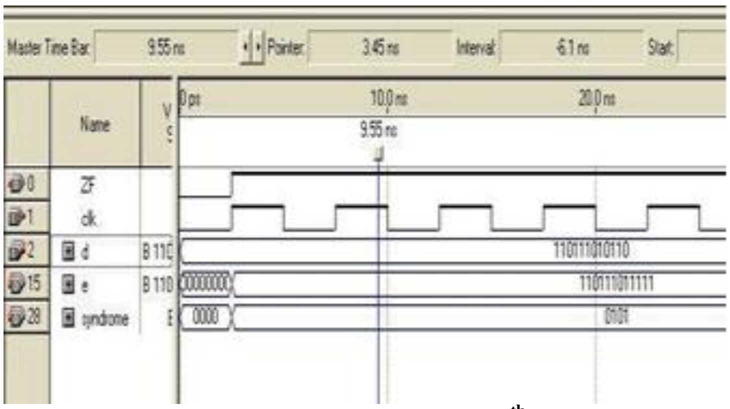


Fig. 9 Simulation Results of ECC with an error in 5th bit position indicated by syndrome as "0101" in Quartus II 7.1 Simulator.

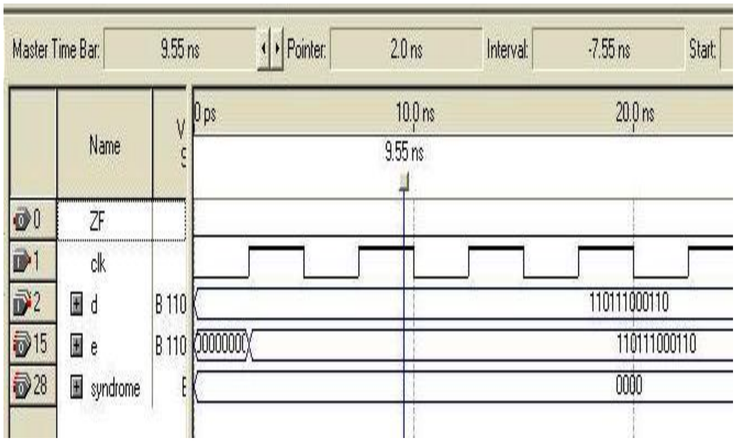


Fig. 10. Simulation Results of ECC with no error in any bit position indicated by Syndrome as "0000" in Quartus II 7.1 Simulator.

V. CONCLUSION

Digital system is designed that counts the number of 1's with the usage of ASM method and its implementation as a high-efficiency technique to detect the key in generalized Ceaser algorithm is illustrated. It can be deployable in complex Encryption algorithms such as DES,Triple DES and AES,where multiple keys are utilized,and establishing the value of keys require much effort . Also the concept of Error Correction can be increased to more than two bit error correction and detection.It finds numerous application in Data communication and networking,espeially at the transport layer while performing Checksum calculation.The sender can calculate the check bits and receiver can verify those check bits and conform whether the data received is same as the data send,if not,can also find the syndrome or erroneous bit.Consumption of resources such as time and speed becomes smaller.

ACKNOWLEDGEMENTS

The authors thank the reviewers for their valuable comments and suggestions that helped us to make the paper in its present form. Authors extend thanks to, Principal, H. K. B. K College of Engineering, Bangalore and the Management, for their support and co-operation in carrying out the work successfully in the institute.

REFERENCES

[1]. Arun Kumar Singh. *Digital Principles Foundation of Circuit Design and Application* (New Age Publishers)
[2]. Thomas, Donald, Moorby, Phillip, *The Verilog Hardware Description Language* (Kluwer Academic Publishers, Norwell , MA)
[3]. Janick Bergerdon, *Writing Testbenches: Functional Verification of HDL Model*, 2000.
[4]. R. W Hamming, Error Detecting and Error Correcting Codes, *The Bell Sytem Technical Journal* ,Vol 29 , 1950
[5]. Wawrzynek, Garp: a MIPS processor with a reconfigurable coprocessor, (FCCM'97, 1997)12 –21.
[6]. Mano And Kime, Logic and Computer Design Fundamentals , *Chapter 7,* " (Printice Hall 2000)
[7]. Janick Bergerdon, "Writing Testbenches: Functional Verification of HDL Models", 2000.
[8]. Tiwari, A. *Formal Semantics and Analysis Methods for Simulink Software*. 2000
[9]. Peter J. Ashenden, *Digital Design ,An Embedded Systems Approach using Verilog*, (Elsevier, 2008)