Journal of Communications Technology, Electronics and Computer Science, Issue 2, 2015
ISSN 2457-905X

Implementation and Verification of Synchronous FIFO using System Verilog Verification Methodology

Navaid Z. Rizvi[1], Rajat Arora[2] and Niraj Agrawal[3]
[1,2,3] School of ICT, Gautam Buddha University
Greater Noida, India
navaid@gbu.ac.in[1], arora.rajat08@gmail.com[2], neeraj6418@yahoo.com[3]

***Abstract* - Developing complex nature of patterns & concurrency of Integrated Circuits has made conventional coordinated test-benches an unworkable answer for testing. Nowadays, testing as a word has been substituted with check. Confirmation specialists need to guarantee what goes to the plant for assembling is an exact representation of the specification of configuration. Verification is the maximum time consuming stage in the whole design process, thus it has become a necessity to minimize the time required to encounter the confirmation prerequisites. The relentless growth in the complexity of the system, has led to the requirement of a more advanced, well organized and automated approach for creating verification environments. As the designs gets complex, the probability of the occurrence of bugs increases, this entailed the influx of the verification phase for authenticating the functionality of the IC's and to detect the bugs at an early stage. In this paper, the synchronous FIFO design is verified using System Verilog Verification Environment.**

Keywords—component; formatting; style; styling; insert (*key words*)

I. INTRODUCTION

This The continuous growth and complexity of digital design requires modem, systematic and automated approaches for creating test benches [1], given that up to 70% of the design period is spent in the authentication process [2] it has become even more critical that verification engineers design test benches that are at the advanced of the verification industry. The EDA vendors have recognized that a standardized approach for verification is required and this approach needs to upkeep the structures desired to shape a progressive confirmation atmosphere. Though, they don't actually identify in what way the architecture of the test bench surroundings must be fabricated. It is still the verification engineer's accountability to do this with the added pressure of making the environment re-usable for future chip sets.

With this in mind the main goal is to develop a new and more effective intuitive way of designing test benches. This paper describes the implementation of constrained random test stimuli, functional coverage, also describes an approach for creating test cases that allow the use of both constrained random tests within a single environment. The environment built should also have the capability to be easily modified where a Device under Test (DUT) of similar structure can be verified [3]. As circuits become more complex, the more capable verification tools must be used. This verification should takes place much earlier than the fabrication process. In this paper a verification environment is realized and implemented which may detect the maximum errors for proper functioning of the synchronous FIFO model.

The FIFO (First in First Out) is a sort of memory that is ordinarily used to cradle the information, has to utilize consistently between diverse systems at distinctive deferrals. The FIFO model permits the transmitter to send information, while the collector is in not functioning stage. The information then tops off the FIFO memory until the beneficiary starts emptying it. An overflow occurs as soon as the transmitter fills up the FIFO model and attempts to store more data before the receiver has read the data out. An underflow occurs when the receiver attempts to read data from the FIFO structure, but the transmitter did not fed any data into it. The Full and empty signals are used by the logic to throttle the transmitter and receiver respectively, in order to avoid these critical conditions. The Fig. 1 shows the functionality of the FIFO. The transmitter puts data into the FIFO, like filling a bucket with water. The newest data is on top, whereas the oldest data is on the bottom. The receiver gets data out of the FIFO like emptying the bucket using a faucet [4].

II. DESIGN AND WORKING

The Fig. 2 demonstrates the basic blocks through which the architecture of FIFO mode has been realized. This Design consists of the Dual Port RAM, Read control logic and Write control logic blocks. Instead of Dual port RAM, these memory arrays can be implemented with the help of flip flops but dual port RAM are chosen due to its simple design. This dual port ram allows simultaneous access of the read and write ports i.e.

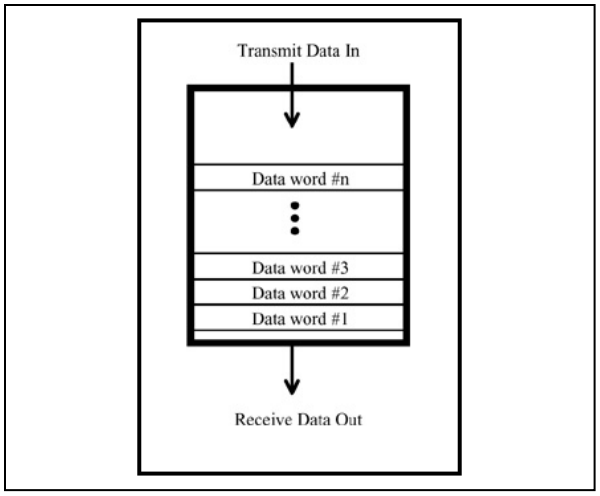


Fig. 1. FIFO functional diagram.

Read and Write operations can be implemented simultaneously. This design has limitation that simultaneous Read and Write access cannot take place from the same memory location.

The synchronous FIFO has a unique clock port for both the Read and Write operations. The data that is given on the data input port is written on the next empty location. But this happens only on the rising edge of the clock, when write enables and the data valid signals of the write control logic block are high. The written memory data is then read out from the Read control logic block. This progress when the read enable signal and the data valid signal of the block are high. There are unavailability for writing data in the FIFO. The Fifo_empty signal indicates that there are still vacant locations that can be utilized to write further in the memory.

The main role of dual port RAM component has been to write and read the data simultaneously. This special type of RAM has two unidirectional data ports, an input port for writing data and an output port for the reading data. Each port is assigned to have their own data and address buses. The write port has a signal named WRITE to allow writing of the data. The read port has a signal known as READ to enable the data output. The particular dual port RAM examined in this paper is synchronous and has a single clock for both ports, as depicted in the Fig. 3. Both reading and writing of data occur when the clock has its rising edge.

The functionality of the dual port RAM can be expressed with three conditions, the first occurs when the reset signal is high and all the output signals (data_out, wr_pointer, rd_pointer and status_counter) reset to zero value. The next condition occurs when write_enable signal is high and the incoming input data (data_in) is written in the memory block with the write addresses generated by the write pointer.

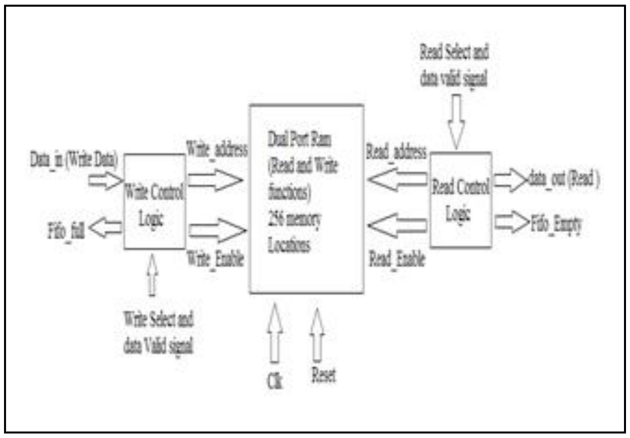


Fig. 2. Internal Block Diagram of the FIFO.

But this only will happen only on the next rising edge of the clock. The write enable signal is only generated when the FIFO is not full so as to avoid corruption of data. The third stage materialize when read_enable signal is high, then the data that is written in the memory is read out form the memory. The read pointer will generate the read address through which the data can be read, but this stage can only exists at the next rising edge of the clock. The Read enable signal is only generated when the FIFO is not empty so that any corrupted data could not be read out from the memory. In the presented Fig. 2, the block on the left side is the Write control logic block. This block is used to control the write operation of the implemented FIFO design The block basically generates a binary coded write pointer and this pointer gets incremented by one location every time, the input data is written into the design. Also, this block generates a full signal to avoid overwriting a data in the memory block. The block on the right side is the read control logic block, this block is used to control the read operation of the implemented FIFO design. The subunit generates a binary coded read pointer which gets incremented by one location every time, the written data is read from the design. Also, this block generates an empty signal such that no invalid data can be read from memory. The verification phase is most important step for any successful design.

III. VERIFICATION PLAN

The verification plannings are growing in a rapid manner hence it becomes more and more skillfully requirements to create a good plan before the verification has been finally started [5]. The verification plan must consist of the entire verification process [6] and creatiopn of a good plan will save a lot of tedious and unfruitful time later. The whole plan should include the time for the complition of process as well as the authentication of the coverage result [7]. The theory about verification planning has been reported in a well orgainined and systematic manner [8], which proposes a five-day approach for the complete verification planning process.

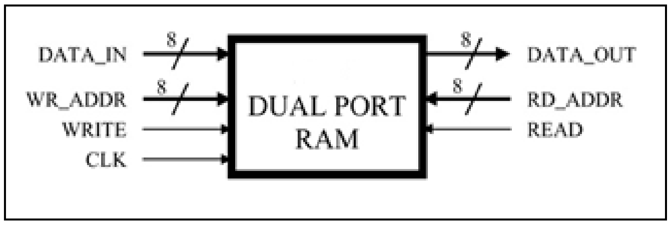


Fig. 3. Block Diagram Dual Port Ram.

However some adjustments are made to better fit System Verilog testbenches. The Block diagram of the verification plan has been shown in Fig. 4 and all its components are explained in this section.

The Interface block is not shown in the plan but this is one of the essential modules throughout the verification plan. In this particular block all the commonly used signals in both the design and the verification environment. The interconnect block bridge as an interfaced to the design under-test and the check environment. The interface embodies all the pin-level associations that are made to the DUT [9]. Basically an interface is a heap of nets or variables.

The transaction generator is used to generate different transactions depending on the test case configurations field selection. For the implemented FIFO design, the transactions are RESET, WRITE and READ operations. Using the above environment, the transaction generator is defined within the Driver block i.e. the coding to generate the read, write and reset transactions is done in the Driver block itself. Each implemented transaction will generate a test case that is a random data which will initiate by the driver and given to the DUT to perform the specific operation utilizing the functioning of the incoming signals.

The driver is the block that translates the transactions in to random inputs i.e. test cases. These are given to the DUT and the DUT performs specific operation depending upon the input given. The transaction generator generates a high level transaction like read, write or reset. The driver basically converts these transactions into actual inputs. A driver gets the information from the generator and drives it to the DUT by inspecting and driving the DUT signals [10]. It contains the hidden rationale to drive the pins of the DUT as indicated by situation gave to it from the sequencer. The scoreboard records the operations of the driver and then displays these operations systematically.

A screen, an aloof commodity, is just to monitor the specimens of the DUT signals [11] but cannot be used to drive them. A monitor collects information, extracts occasions, performs checking, scope and optionally prints follow data. It utilized the screen to sign sent to the pins of the DUT from the driver. As the name suggests, it basically monitors the operations performed by the driver and then it passes its data to the scoreboard to display the information

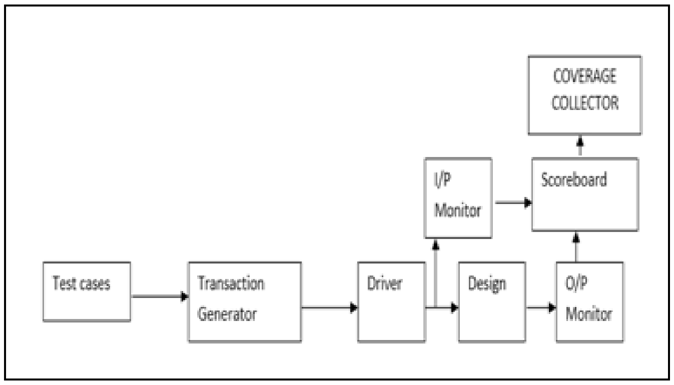
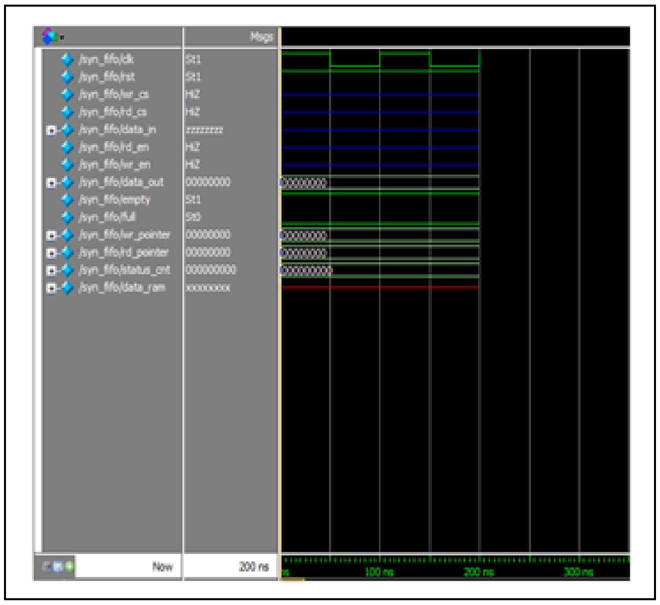


Fig. 4. Verification plan.

The scoreboard can also be named as tracker as it tracks all the operations. The dynamic data types and dynamic memory allocation makes the task much easier to implement scoreboard in the system Verilog. Normally, a scoreboard confirms whether there has been fitting operation of the configuration at a practical level. The Scoreboard basically stores the data and address when the write operation is done and displays the results. Furthermore, it also records the data and addresses that had been previously read. It matches the similarity of the data and display the outcome. The coverage collector mainly covers the coverage related issues of the block. This block have cover the groups and cover points that are used to estimate the functional coverage of the design.

IV. DESIGN OUTCOMES AND ANALYSIS

Initially the outcomes comprised of the Reset, Read and Write condition results. Fig. 5 depicts to the scenario in which all the output signals are reset to zero which is shown with the help of waveform generated in the tool.



```
/syn_fifo/clk         St1
/syn_fifo/rst         St1
/syn_fifo/wr_cs       HiZ
/syn_fifo/rd_cs       HiZ
/syn_fifo/data_in     zzzzzzzz
/syn_fifo/rd_en       HiZ
/syn_fifo/wr_en       HiZ
/syn_fifo/data_out    00000000
/syn_fifo/empty       St1
/syn_fifo/full        St0
/syn_fifo/wr_pointer  00000000
/syn_fifo/rd_pointer  00000000
/syn_fifo/status_cnt  000000000
/syn_fifo/data_ram    xxxxxxxx
```

Fig. 5. The Reset Waveform.

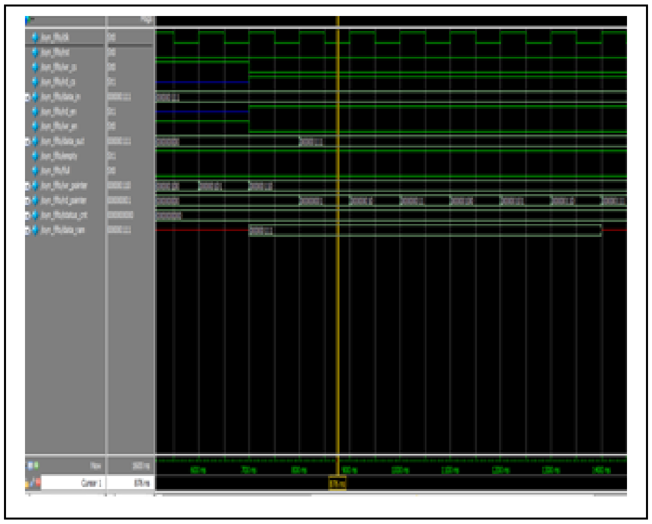


Fig. 6. Write Condition Waveform.

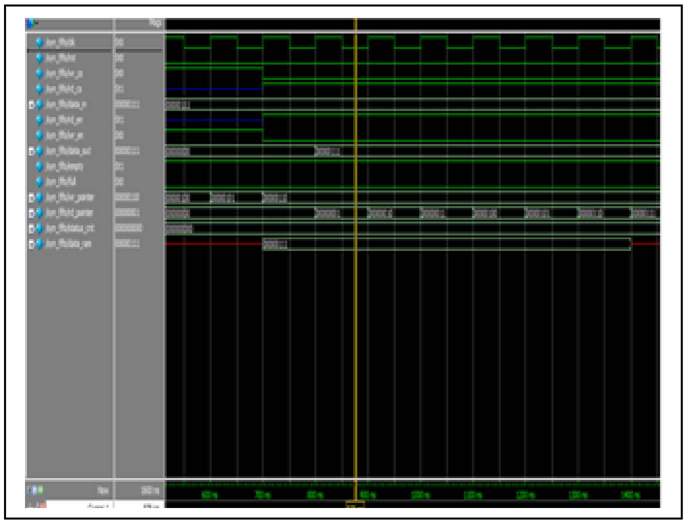


Fig. 7. Read Condition Waveform.

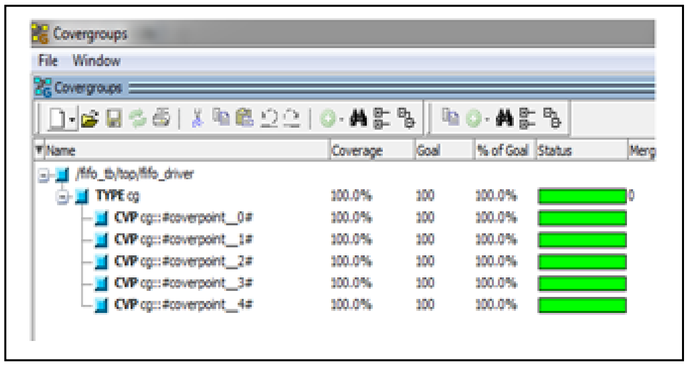
As shown reported in the waveform the initial signal is the clock signal. This is a free running signal and all other signals will have to change their values only on the rising edge of this signal. The next signal is the reset signal, this is the signal that dictates the reset condition for all other signals. This status has forced the value 1 as such that all the output signals would be reset to zero. The output signals data_out, read_pointer, write_pointer and the status_pointer are set to zero.

The Fig. 6 shows the scenario in which all data is written in the FIFO through waveform generated by the Modelsim tool. As exhibited in the figure the first signal is the clock signal that is a free running signal. All other signals will change the values on the rising edge of the clock. The reset signal in this case has forced to zero and this reset to zero value should be avoided. The data is given on the data_in signal (00001111). This data is the input data that is given in the FIFO memory. The wr_cs and wr_en signals are also set to 1, to activate the write control logic of the design. This block is responsible for writing the data in the memory. The Fifo_empty signal becomes 1 as the FIFO memory is still vacant but the Fifo_full signal is set to 0 as the FIFO is not full in this situation. The wr_pointer which is write pointer signal has been binary coded, gets incremented by one value. As many times the rising edge of the clock is coming, the wr_pointer gets incremented and the incoming data gets written into the FIFO memory on that write pointer location.

The Fig. 7 laid out the scenario in which the incoming data is read out from the FIFO. As depicted through the waveform, the first signal is the clock signal as elaborated in the write status explanation. The data that was written in the memory (00001111) with the help of data_in signal is now read out form the FIFO memory. The rd_cs and rd_en signals are set to 1 so to activate the read control logic of the design.This block is basically responsible for reading data from the memory. The Fifo_empty signal is 1 in this circumstances as the FIFO is still in empty state and Fifo_full signal is 0 as the FIFO is not full in this scenario. In this case, now the rd_pointer that is the read pointer signal gets incremented by 1 value. Now the read pointer will traverse through all those locations on which the data was written by the write pointer. When all the data written would be read out then the FIFO will again be completely empty. The written data is read out from the FIFO with the help of data_out signal.

V. Verification Results and Analysis

The main result is the coverage report as demonstrated in the Fig. 8. The practical coverage is the determination of the amount of the design usefullness having induced by the verification environment [13]. Initially the code includes, the type of cover groups to screen the stimuli being put on the DUT. The responses and reaction to the stimuli are additionally checked to figure out what usefulness has been worked out. The cover groups ought to be indicated in the plan of verification. Inside a scenario of test, their handiness is learned by dissecting the RTL code and comprehension of the data. The cover points turn out to be all the more capable inside the recreation when they are crossed together to inside



Name	Coverage	Goal	% of Goal	Status	Merg
/fifo_tb/top/fifo_driver					
TYPE cg	100.0%	100	100.0%	🟩	0
CVP cg::#coverpoint__0#	100.0%	100	100.0%	🟩	
CVP cg::#coverpoint__1#	100.0%	100	100.0%	🟩	
CVP cg::#coverpoint__2#	100.0%	100	100.0%	🟩	
CVP cg::#coverpoint__3#	100.0%	100	100.0%	🟩	
CVP cg::#coverpoint__4#	100.0%	100	100.0%	🟩	

Fig. 8. Coverage Report.

recognize more noteworthy levels of reflection of an outline [13]. The cover groups give an effective system in distinguishing zones of useful scope inside of a configuration.

The Fig. 8 shows the coverage report of the design. This report basically shows the coverpoints that has been created to cover the different situations of the design automatically. The coding part of this coverage report has been given below :

```
covergroup cg@(posedge cports.clk);
coverpoint cports.data_in{
 bins data_range   = {0,255};
}
coverpoint cports.rd_cs {
   bins  rd  = {0,1};}
coverpoint cports.wr_cs {
   bins  wr  = {0,1};}
coverpoint cports.rd_en {
   bins  ren = {0,1};}
coverpoint cports.wr_en {
   bins  wen = {0,1};
```

In the coding, a covergroup is created that is generated which is clearily visible through the coverage report. The coverpoints are created so as to cover the each important aspect of the design [12]. The **cports.rd_cs** and **cports.rd_en** coverpoint has covered both the scenarios in which the Read control logic is selected in one situation and not selected in the other. It covers these conditions as these signals are given 0 and 1 value randomly so as to cover both the situations. This is the reason that the coverage for these signals are coming 100% The **cports.wr_cs** and **cports.wr_en** covers both the scenarios in which the Write control logic is selected in one and not selected in the other phase. Both the situations are cover by the signals which are given 0 and 1 value. Due to this same reason that the coverage for these signals also are showing to be 100 %.The most essential coverpoint of these is the first one i.e. **cports.data_in** . This coverpoint basically cover the scenarios in which random test data is generated automatically and is given to the FIFO design. This coverpoint basically covers the conditions whether all types of data is given to the design or not in the specified range that is from 0 to 255. This range can also be visible through the coding. It checks whether all the types of data being written in the design and the same data that was written is read out form the design. The random data that is being written and read out form the design can be seen in the scoreboard report that is given in the next section.

The next is the scoreboard report [13] explained below. The scoreboard report that is generated with the help of Scoreboard block of the verification plan. The scoreboard report generated in this case records the transactions that are taking place in the verification environment. For the above design it records the Reset, Write or Read scenario are being implemented by the verification environment. It records the random test case data that is generated by the Write control logic to be written in the FIFO memory. It also records the test case data that is read out form the memory with the help of Read Control Logic. Furthermore, it compares both the data being written matches with the data read out. The Driver requests are given to the scoreboard through the monitor block. The generated scoreboard report for the design has been given below :

```
#         41ns : Write posting to scoreboard data = 3d
#         43ns : Read posting to scoreboard data = 3d
#         43ns : Write posting to scoreboard data = 34
#         43ns : Read posting to scoreboard data = 34
#         43ns : Write posting to scoreboard data = 8c
#         45ns : Read posting to scoreboard data = 8c
#         45ns : Write posting to scoreboard data = 8c
#         45ns : Read posting to scoreboard data = 8c
#         45ns : Write posting to scoreboard data = c6
#         47ns : Read posting to scoreboard data = c6
#         47ns : Write posting to scoreboard data = c0
#         47ns : Read posting to scoreboard data = c0
#         47ns : Write posting to scoreboard data = aa
#         49ns : Read posting to scoreboard data = aa
#         49ns : Write posting to scoreboard data = 80
#         49ns : Read posting to scoreboard data = 80
#         49ns : Write posting to scoreboard data = 77
#         51ns : Read posting to scoreboard data = 77
#         51ns : Write posting to scoreboard data = 65
#         51ns : Read posting to scoreboard data = 65
#         93ns : Terminating simulations
```

VI. Conclusion

One of the initial objectives of this paper was to design and implement a FIFO module. This task is successfully completed as the FIFO module is designed and implemented using the Verilog language. The next outcome was to use this FIFO module the building the verification environment. The last and most crucial target of this work was to implement the verification plan of realized FIFO Design using System Verilog Framework. The test cases which are the input data that need to be given was randomized and automated using the implemented verification model. The system Verilog functional coverage methodology is adopted that verifies the functionality of the design in most effective way. All the blocks of the verification plan are implemented using system Verilog language.

The outcomes were obtained as functional Coverage and the scoreboard reports using verification methodology. 100 % coverage has been achieved for the constraints that were applied to the design. Also the scoreboard report is generated using the scoreboard block of the verification plan. This report explained the various transactions that took place to achieve this coverage for the design.

If the plan are well organized and skillful they will reduce lot of undesirable design time. This work can be further extended y realizing other complex systems verification methodologies.

REFERENCES

[1] Chris Spear, "SystemVerilog for Verification", Springer, Vol 1 pp. 1-5, 2005.

[2] Janick Bergeron, "Writing Testbenches Using SystemVerilog" Springer, Vol I pp. 1-10 , 2006

[3] Takimoto, Y., "Recent activities on millimeter wave indoor LAN system development in Japan," *Dig. IEEE Microwave Theory and Techniques Society Int. Symp.*, 405-408, Jun. 1995.

[4] Bergeron, Janick, Writing testbenches: functional verification of HDL models, Springer, Edition 2003.

[5] Wang, Xin, Tapani Ahonen, and Jari Nurmi, "A synthesizable RTL design of asynchronous FIFO", System-on-Chip Proceedings. 2004 International Symposium on. IEEE, pp. 123-128, 2004.

[6] Yakovlev, Alexandre V., Albert M. Koelmans, and Luciano Lavagno. "High-level modeling and design of asynchronous interface logic." IEEE Design & Test of Computers, Vol 12.1, pp 32-40, 1995.

[7] K.K. Yi, "The Design of a Self-Timed Low Power FIFO Using a Word-Slice Structure", M.Phil Thesis, University of Manchester, September 1998.

[8] Chelceq T., Nowick, S.M., "Low-latency asynchronous FIFO's using token rings", Advanced Research in Asynchronous Circuits and Systems, Proceedings. Sixth International Symposium, Vol 2-6 , pp 210 – 220, April 2000.

[9] Andreas Meyer, "Principles of Functional Verification" Vol I, pp. 1-10, 2004.

[10] Doulos Ltd, "SystemVerilog Golden Reference Guide" Vol II, pp. 1-11, 2003.

[11] J. Bergeron, E. Cerny, A. Hunter, and A. Nightingale, "Verification Methodology Manual for System Verilog.", Springer, 2006.

[12] Synopsys, "System Verilog Assertions Checker Library Quick Reference", April, 2006.

[13] S. Vijayaraghavan and M. Ramanathan, "A Practical Guide for System Verilog Assertions". Springer, 2005.

[14] Keaveney, M., McMahon, A., O'Keeffe, N., Keane, K., & O'Reilly, J. "The development of advanced verification environments using system verilog.", Signals and Systems Conference, 208.(ISSC 2008), pp. 325-330, 2008.