International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-2 Issue-6, August 2013

VIP Architecture and Design Using OVM for IrDA Protocol

Muralidhara R, Siva Yellampalli

***Abstract*: This paper presents practical and efficient way of architecting and developing verification component (VIP) for IrDA protocol using OVM methodology, which supports transaction level modeling , coverage driven and self-checking and reusability across subsystem , system, Architectural exploration along with HW-SW co-simulation, Which can address Time to market and Bug free silicon.**

Keywords—Transcation level modeling, Verification IP, coverage driven, OVM, Time to market, Bug free silicon, resusbality.

I. INTRODUCTION

Verification consumes nearly 70% of design time during the SOC/ASIC design cycle [1], bug in Silicon will leads in to functional or performance failure due to which ASIC/SOC may need a replacement in the particular devices where these ASIC/SOC are used.

Quality of verification comes with amount of Logic simulation using random or directed stimulus with the functional and code coverage on RTL [2], to ensure Design is bug free which is again time consuming.

To ensure Verification is effective and not time consuming we need to have configurable, coverage driven, self-checking Verification components which can easily reused across, unit level, subsystem level and system level verification [3] and [4]

The quality of verification can be increased with minimal Test cases with the random stimulus which system Verilog supports with the coverage driven approach [4].

Transaction level modeling is achieved with OOPS concept and using OVM methodology [5], We are going to discuss on the VIP architecture and design using OVM for IrDA protocol which supports all above[1][2][3][4][5] and [6].

Manuscript published on 30 August 2013.

***** Correspondence Author (s)

Mr.Muralidhara.R, 6th sem MTECH-PT in VLSI and Embedded systems, VTU Extension Center, UTL technologies Ltd, Bangalore-560022, India.

DR. Siva Yellampalli, Professor, VTU Extension Center, UTL technologies Ltd, Bangalore-560022, India.

II. IRDA

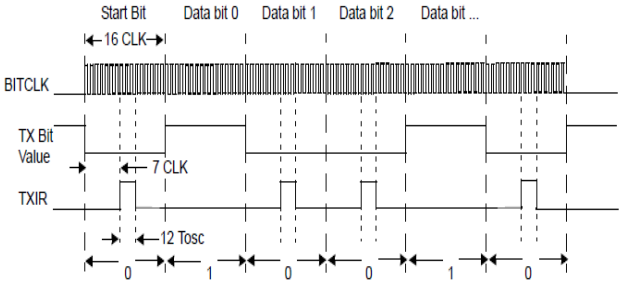

Figure 1: IrDA Encoding/Modulation

A. Encoding (Modulation)

In Figure 1, data that needs to be modulated will be TX bit value, which is nothing but serial Transmitted data of an UART for a 16 clock cycle, out of which 3 clock cycles 1.6 µs wide and 3 clock cycles long at 1.843 MHz

The data Rate which it needs to support during transmission is 115.2 kbps as shown in the above diagram.

BITCLK is the clock which is driven by oscillator and TX Bit Value is of 16 clock cycle, out of which after 7 clock cycle the modulated data has been hold by TXIR ,which will be send to the IrDA Diode part , modeling of diode part is not explained here.

After 7+3 clock cycle the remaining 6 clock cycle holds the bus low.

The default value of the bus is low and 0 is represented by 1 after modulation similarly 1is represented with 0 [6].

B. Decoding (Demodulation)

In Figure2 decoding happens whenever IrDA receives data in rx pin, same rx bit is sampled for 16 clock cycle.

Starting 3 clock cycle the rx data is sampled (1.6 micro second), and remaining 13 clock cycles the RX bus will be held at high, the below scheme is called RX mode in inverting mode, in non-inverting mode the same data will be inverted .

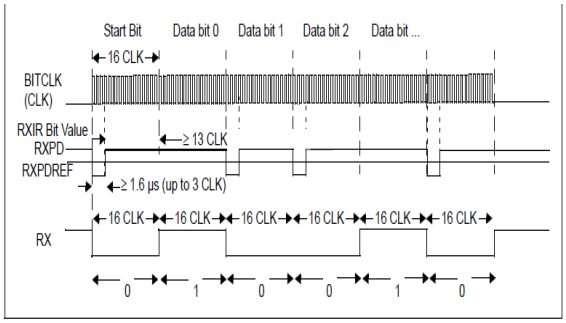

Figure 2: IrDA Decoding/Demodulation

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

Both encoding and decoding starts with start bit, and bits needs to be transmitted or received.
Figure 3 shows how Uart frame will converted in to IrDA Frame In functional verification we need a TLM modeling [5] of the IRDA protocol which supports all above [1][2][3][4] and [5].

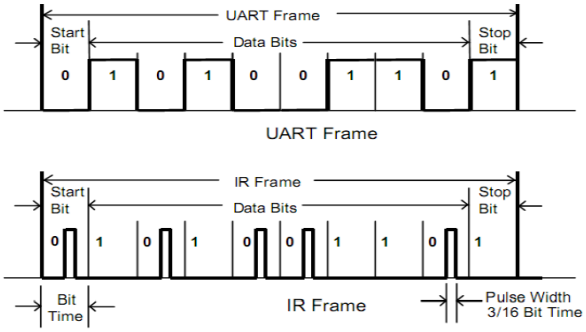

Figure 3: Uart to IrDA frame conversion

III. VIP ARCHITECTURE

VIP architecture has been divided in to four parts, which is purely OVM based, Figure 4 shows the OVC/VIP architecture in OVM methodology.

These are the four main blocks which needs to be architectured and implemented using OVM methodology and System Verilog language.

a. Interface

b. Tx-Agent or Master-Agent

c. Rx-Agent or Slave-Agent

d. Data modeling

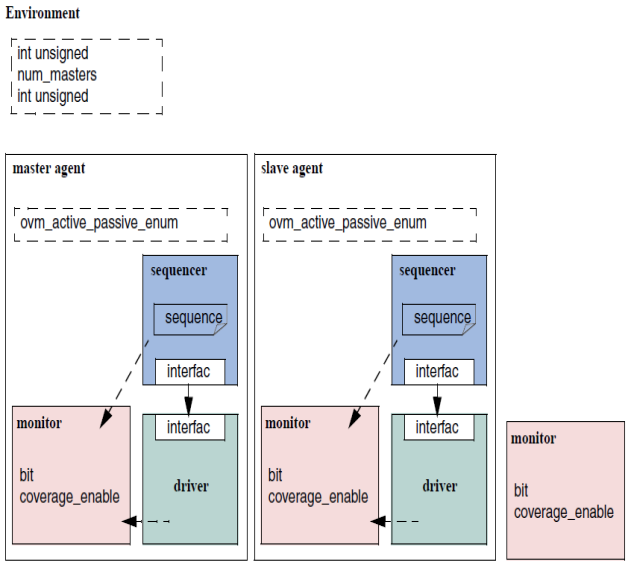

Figure 4: OVC/VIP architecture in OVM

a. Interface :
Interface block will have Mod ports for DUT and Test bench along with clocking block, by using the same interface block a physical instance and virtual interface instances in all the components of the agents as shown in the Figure 5, the same VIP is used for Gate level simulation by adding any intra and inter delays .

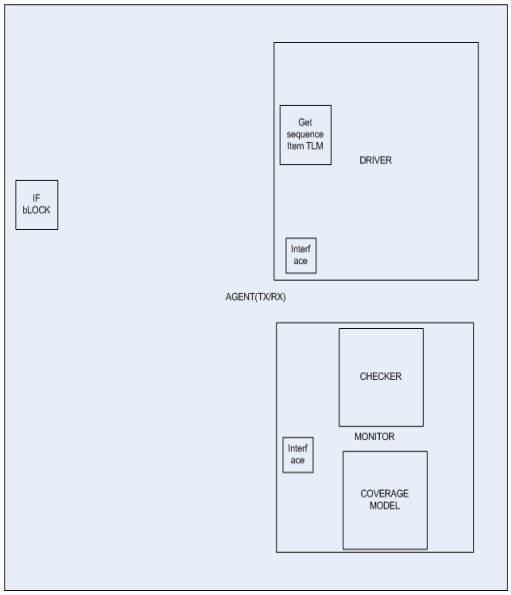

Figure 5: Interface block architecture and usage

Below piece of code is implemented in the interface block, Physical interface is passed from the Test bench using set_conifg , and assigned inside the agent and env using get_config to their sub blocks.

```
// Control flags
  bit   has_checks = 1→ To enable/disable Checker
bit   has_coverage=1→  To  enable/disable  Functional
coverage
 logic  DAT_in;  →For RX agent
 logic  DAT_out; →For TX agent
 logic  sig_reset; → For reset
clocking cb @(posedge clk); → Clocking  Block
modport DUT → Mod port for DUT
modport TB  → Mod port TB
```


Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

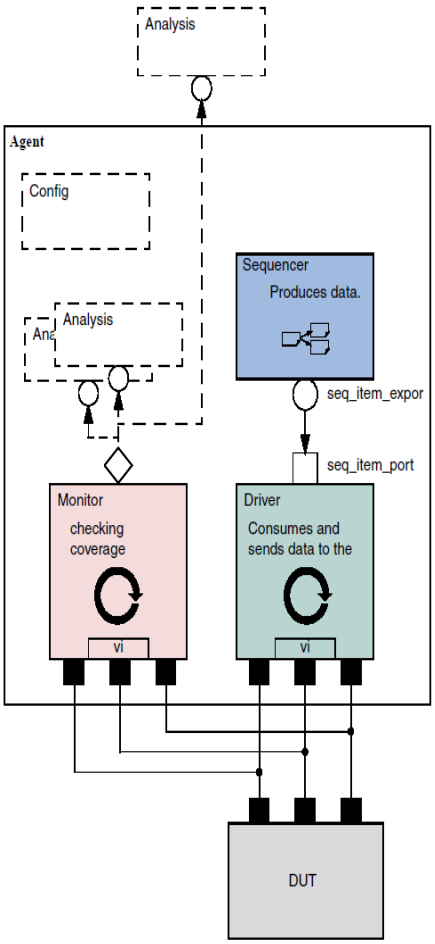

Figure 6: Agent Architecture

b. Tx-Agent or Master-Agent

Tx-Agent or Master-Agent as shown in Figure 4 and Figure 6 will have the below blocks:

- Configuration: which uses OVM config DB, through which Agent can be selected for active or passive ,OVM_ACTIVE will decides driver enable or disable.
- Monitor: This block always works in passive mode, its main function is to collect and convert TX port signal information to transaction modeling, through interface VI shown in Figure 6, also same monitor will have functional coverage model along with the checker, which enables self-checking and coverage driven verification, and same monitor can be reused [2][3][4].which supports random verification[5].
- Driver: which collects the transaction from the test case/sequences(TLM)/sequencer and drive the signals by consuming it as per IrDA protocol (Figure 1) using the interface block as shown in Figure 7
- Sequencer: which communicates between sequence and driver and collects the IrDA TLM transactions from the test/sequences and send it to the driver as shown in Figure7

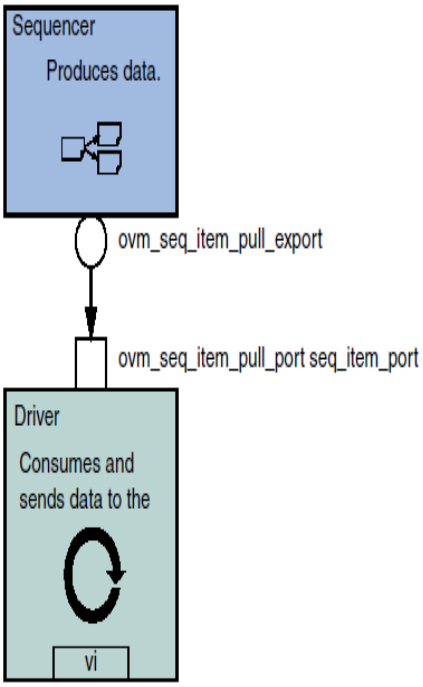

Figure 7: TLM flow from Sequencer to Driver

c. Rx-Agent or Slave-Agent

Rx-Agent or Slave-Agent architecture and blocks are same Tx-Agent or Master-Agent components but works on Rx Port as per IrDA protocol shown in Figure 2.

d. Data modeling

Data modeling will have total two parts which are listed below:

a. Sequence modeling
b. Coverage modeling

a. Sequence modeling: Which has details of how many bytes of data needs to be transmitted using TX-Agent, along with delay between each transfer and generating error ,which is against IrDA protocol, which can test the DUT for its correctness of functionality with the below sequence modeling which is implemented for IrDA TX/RX.

Data model/Transaction model details: common for TX/RX

- Uart_Irdi_trans_kind - READ,WRITE
- Uart_Irdi_error -ENABLE,DISABLE
- Uart_Irdi_invert_type -INVERT_EN,INVERT_DE
- Uart_Irdi_lp-LP,NONLP

Sequence details:

- Uart_Irdi_trans_kind;
- Uart_Irdi_error;
- size2;
- transmit_delay_min2 ;
- transmit_delay_max2 ;
- invert_type;

The same data modeling is used in checker for data and protocol checking , also through an analysis port same truncations are provide to the IP , Sub system or System level Scoreboard through analysis port as shown in Figure 6 to support self-checking and reusability [2][3][4].

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

b. Coverage modeling: Checker will check for the correctness of the IrDA protocol and update a coverage model, to show that functionalities which are covered in a random stimulus based verification, coverage model shows the functionality which are covered or hit by the Stimulus , the below coverage model is implemented for IrDA TX/RX
Functional coverage model:

- tans_kind : Read/Write
- delay : {1,2,4,8}-between each transfer;
- invert_type : invert en or invert de;
- decoding_trans:cross

tans_kind,error_kind,delay,invert_type(cross coverage);
Above coverage should hit only when checks passes.

IV. STATEMACHINES

Driver and monitor is coded with state machine, each state represent each functional state of the IrDA protocol.
TX agent state machines related to the TX side traffic generation from driver and collecting the signals and converting them in to transactions in monitor by using the interface block.
Rx agent state machines related to the RX side collecting the signals and converting them in to transactions in monitor by using the interface block.
a. Driver and Monitor state machine design for TX-Agent:

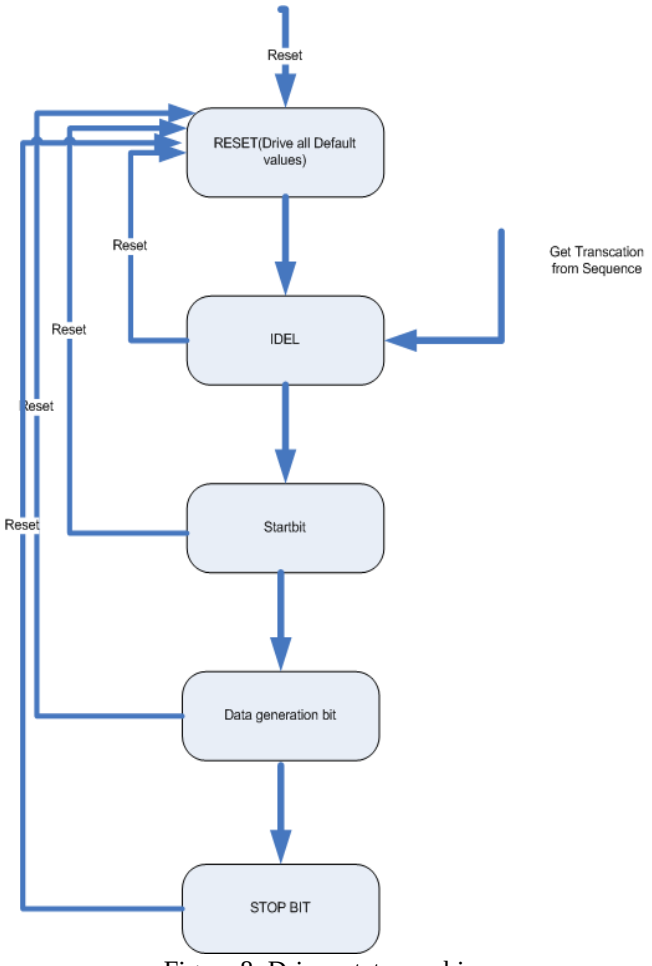

Figure 8: Driver state machine

RESET: This is the state where all the signals are driven to reset values and same thing will be monitored by monitor, from RESET to IDEL state movement happens, when there is no reset signal in the next positive edge of the clock.

IDEL: This is the state where TX master agent will be there immediately after reset, waiting for the transactions, which needs be generated by sequence and tests.
START bit: This is the state where Driver starts consuming the transactions; similarly monitor will be looking for the start bit, in the next clock cycle the state will be moved to DATAGENERATIONBIT state, if there is no reset.
DATA GENERATION BIT: This is the state where Driver starts consuming the transactions and generate 8bit data, 1 bit data in every sixteen clock cycle as per Figure1, in between if it finds reset it will go to RESET state.
Similarly monitor will start creating transactions from 16clock cycles of the interface until 16*8 clocks, to create a byte of transaction.
From this state state machine will move to the STOPbit state.
STOP BIT: This is the state where Driver starts consuming the transactions and generate stop bit, in between if it finds reset it will go to RESET state.
Similarly Monitor State machine will look for this signal to go to its RESET state.
b. Monitor state machine design for RX-Agent:

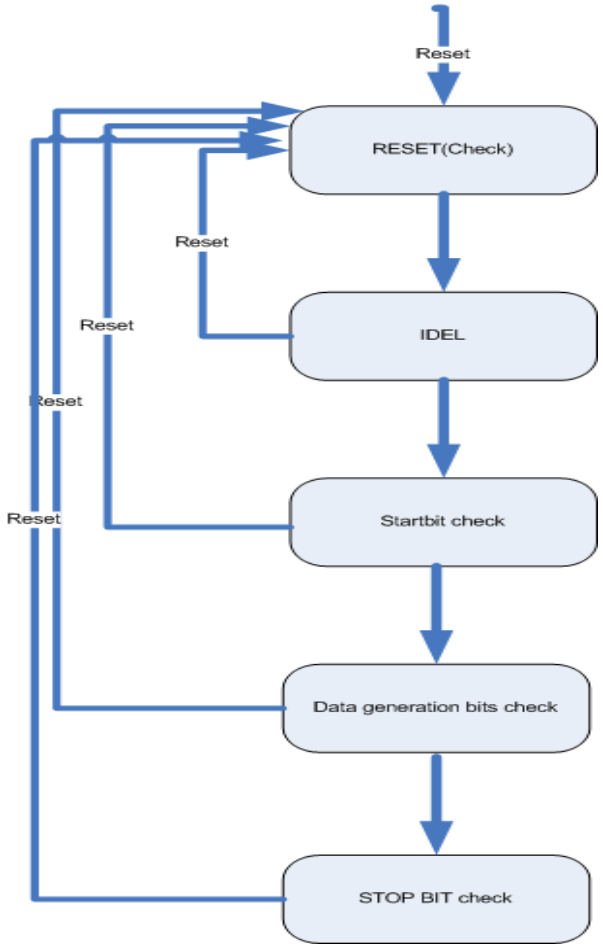

Figure 9: Rx agent Monitor state machine

This agent and monitor will run in continues loop.
RESET: This is the state where all the internal values of the transactions are converted in to reset values and same thing will be monitored by monitor, from RESET to IDEL state movement happens, when there is no reset signal in the next positive edge of the clock.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

IDEL: This is the state where RX master agent will be there immediately after reset, waiting for the start bit which needs be generated by TX port.

Startbit Check: This is the state where Driver look for TX side start bit, similarly monitor will be looking for the start bit, in the next clock cycle the state will be moved to DATAGENERATIONBIT state, if there is no reset.

Data generation bits check: This is the state where Driver starts looking for 8 bit bit data, 1 bit data in every sixteen clock cycles, in between if it finds reset it will go to RESET state.

From this state, state machine will move to the STOPbit state.

STOP bit check: This is the state where Monitor starts looking for the TX side stop generate bit, in between if it finds reset it will go to RESET state.

V. SIMULATION AND RESULTS

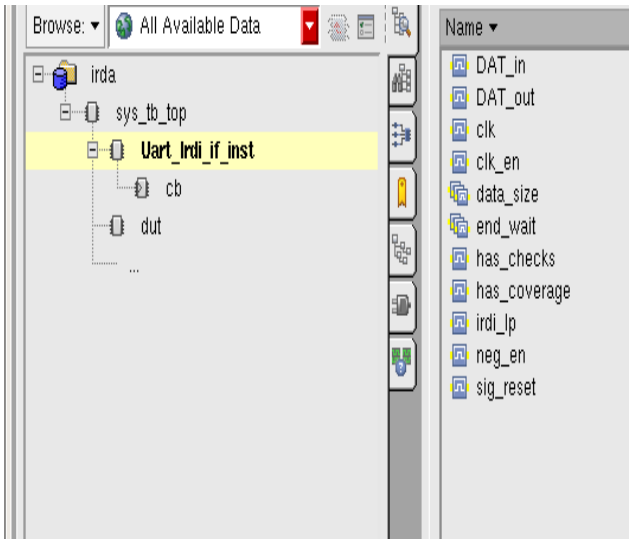

Figure 10: Loaded data during simulation

Simulation is conducted by connecting RX and TX agent VIP back to back.

Cadence NC-SIM used for Compilation and running simulation as shown in Figure 10 and Figure 11.

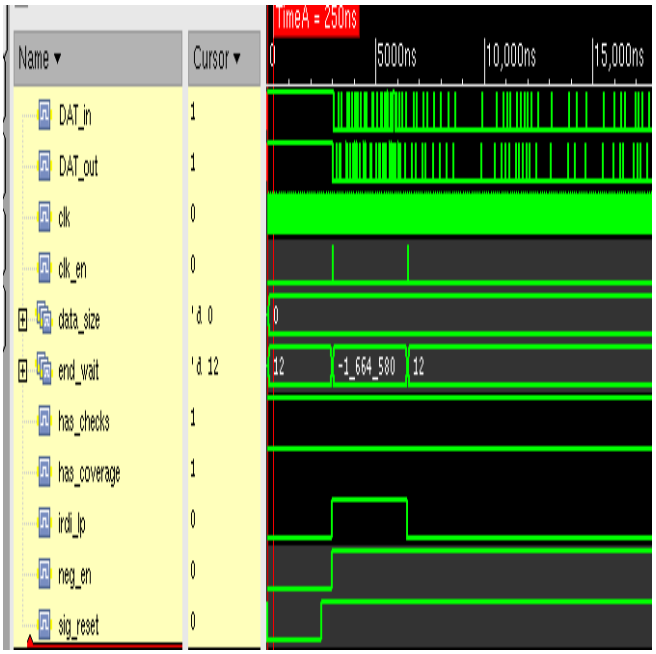

Figure 11: simulation on RX –TX agent

For checking the functional coverage Inclusive code coverage tool is used, results are as shown in Figure 12 and Figure 13

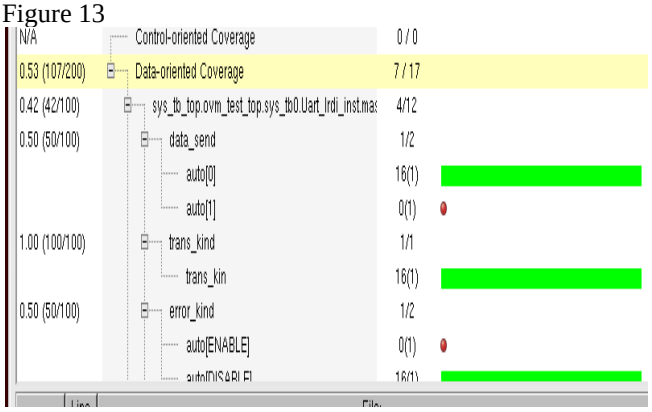

Figure 12: Functional coverage for the simulation part1

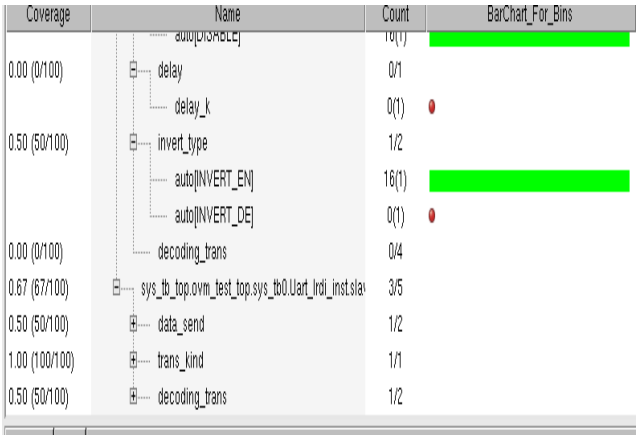

Figure 13: Functional coverage for the simulation part2

VI. CONCLUSION

With above IrDA VIP we are able to generate TX-RX random traffic, by using Transaction level modeling and Self checking feature debugging is very easy.

With the coverage driven verification we will know the functionalities which are verified and missing functionalities needs to be verified.

This VIP will make verification cycle fast, also all the above components can be reused across IP, Subsystem and System level of verification.

REFERENCES

[1] A. Raynaud. The new gate count: What is verification's real cost? Electronic Design, October 2003 link
[2] B.stohr, M.Simmons, J.Geishauser "FlexBench:Reuse of verification IP to increase the productivity " Design Automation and Test in Europe Conference and Exhibition, 2002
[3] Hu Zhaouhui, A.Pierres, Hu shiqing, Chen Fang "Practical and efficient SOC Verification flow by resuing IP testcase and Testbench" SOC design conference(ISOCC),2012 International.
[4] System verilog 3.1a Language Refrence Manual.link
[5] OVM 2.1.2 Methodology manual.link
[6] IrDA standard Protocol Stack Controller With Fixed 9600 Baud Communication Rate -2007-2011 Microchip Technology Inclink

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)