| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 7.122|

||Volume 9, Issue 5, May 2020||

Developing Verification IP (VIP) for AMBA LowPower Interface P-Channel Protocol using Universal Verification Methodology (UVM)

A.Vipula[1], Nikkitha.N[2], Dr Kiran Bailey[3], Loganath Ramachandran[4]

UG Student, Dept. of ECE, BMS College of Engineering, Bangalore, Karnataka, India[1]

UG Student, Dept. of ECE, BMS College of Engineering, Bangalore, Karnataka, India[2]

Assistant Professor, Dept. of ECE, BMS College of Engineering, Bangalore, Karnataka, India[3]

Founder, VeriKwest Systems Inc., Santa Clara, USA[4]

ABSTRACT: In today's world there in an ever-increasing complexity of System on Chip (SOC) Architectures with numerous IPcores encompassing various functionalities.System Level Verificationof these IP's is cumbersome process. Owing to advancementin the VLSI Industry, standards were developed and one amongthem is Universal Verification Methodology UVM in short. Usingthis methodology, Verification IP's are developed to verify thecorresponding IP cores and are reused as the need states. Thispaper deals with developing a Verification IP, for AMBA LowPower Interface P-Channel Protocol using Universal VerificationMethodology.

KEYWORDS:UVM, AMBA Low Power Specifications, VLSI, DUT

I.INTRODUCTION

During the last decades, various verification techniquesor methodologies have been developed by EDA Vendorsfor verification of ASIC Designs. Due to compatibility relatedissues from one vendor to another the need of the hour statesthe usage of common, universally accepted Verification Platform.Thereby the standard adopted is Universal VerificationMethodology (UVM).UVM is a standard to enable faster, easier development andsoftware reuse throughout the semiconductor industry. It is aset of class libraries from which references can be derived or inferred and is defined using the syntax and semanticsof SystemVerilog (IEEE 1800) (a unified Hardware Descriptionand Verification Language).UVM is now an IEEE standardwhich is widely acceptedthroughout.The main idea behind UVM is to help companies /industries developreusable, modular, efficient, and extensible testbench structures catering tocomplex designs under test (DUT).This paper presents VIP developedusing UVM for the **AMBA Low Power Interface P-Channel Protocol**.

AMBA (Advanced Microcontroller Bus Architecture) is anopen standard and registered trademark of ARM Ltd., which was introduced in 1996. It is freelyavailable which is used for the connection andmanagement of functional blocks in a system-on-chip (SoC).It facilitates right inceptive development of humongous multiprocessordesigns, with large numbers of controllers andperipherals built upon them. In this project the protocol underconsideration is the AMBA Low Power Interface P-Channelspecifications [3]. The two-low power AMBA specificationsare:

- Q-Channel used where simple run-stop quiescence semantics are suitable.
- P-Channel used for management of complex power scenarios where multiple power transitions are involved.

II.PROBLEM DEFINITION

A usual approach for simulation is building a non-standardsv environment tailoring to our protocol needs which can be done as starting phase to get basic functional overview of theprotocol. But as the complexity of the design increases such
approaches tend to fail the feasibility and unable to cover thecorner cases, hence the UVM approach with appropriate regressiontechniques can be deployed to solve larger intricaciesin the verification platforms.

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 7.122|

||Volume 9, Issue 5, May 2020||

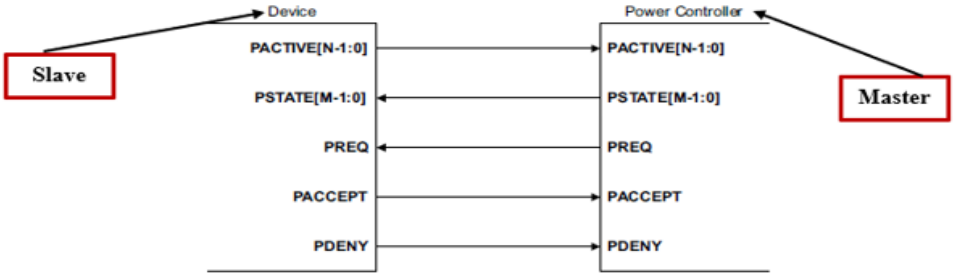

Fig. 1 Signal Mappings of the Power Controller and the Device

The UVM presents easier debugfeatures with much more increased parallelism involved. Wehave a power controller Design under Test (DUT) which needs to be verified,upon response from the Device where a certain number ofhandshake signals are involved between. The further details tothe same have been discussed in III. Fig 1 shows the signalsgroups and their directions for the Power Controller and theDevice.

III.PROPOSED SOLUTION

We have considered a **master - slave** based approach which enables us to understand the responsive of the mutually coexistent system as shown in Fig 1. The Power Controller is the master which is the DUT designed or the modelled RTL and VIP is developed for the slave that is the Device using standard UVM. The Programming Language used in this project is **SystemVerilog** with main structuring from **Object Oriented Programming Concept (OOPs).** Here we are checking the ACCEPT/DENY communication of the slave for the request from the master. For the corner case we are checking an ILLEGAL transaction form slave to see the response form the master. The underlying details to the same have been discussed in section IV.

IV. METHODOLOGY AND IMPLEMENTATION

4.1 MASTER DESIGN

Transition conditions	Current State	Next State	Actions
ENABLE_REQ == 1	P_STABLE	P_REQUEST	Drive PSTATE signal with new value Drives PREQ signal 1
PACCEPT == 1	P_REQUEST	P_ACCEPT	Drive PSTATE signal with new value Drives PREQ signal 1
PDENY == 1	P_REQUEST	P_DENIED	Drive PSTATE signal with new value. Drives PREQ signal 1.
1	P_ACCEPT	P_COMPLETE	Drive PSTATE signal with new value Drive PREQ signal 0
PACCEPT == 0	P_COMPLETE	P_STABLE	Drive PSTATE signal with new value Drives PREQ signal 0
1	P_DENIED	P_CONTINUE	Drive PSTATE signal with old value Drive PREQ signal 0
PDENY == 0	P_CONTINUE	P_STABLE	Drive PSTATE signal with old value Drive PREQ signal 0

Fig. 2 The Master(DUT) is modelled based on the following conditions.

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 7.122|

||Volume 9, Issue 5, May 2020||

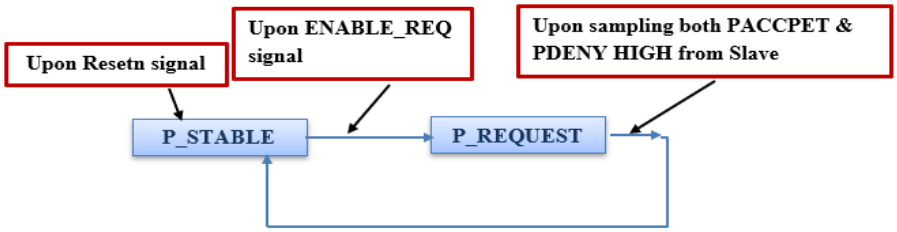

Fig. 3 Corner case or ILLEGAL Scenario flow

Master is referred as designed under test or DUT in short. Fig 2 shows the state transition details of the master designedusing **SystemVerilog** a Hardware Description and VerificationLanguage. The state transitions details are designed keepingthe [3] as reference. A posedge clock and negedge resetis deployed. The system goes to stable state upon RESETnassertion and upon deassertion further process is continuedaccording to the protocol specifications.Fig 3 show how an ILLEGAL scenario is dealt bythe master. Upon such an encounter the master moves to P_STABLE STATE.

4.2 SLAVE DESIGN

A slave corresponding to the master has been developedusing the UVM Framework shown in Fig4. All references tothe class libraries are derived from **UVM 1.1d** [4]. The slavesends PACCEPT and PDENY values upon request(PREQ) from master. The sending of these values to master is done ina channelized manner which is referred to as transactions inthe UVM context. The details of master-slave communication or handshake mechanism is discussed below.

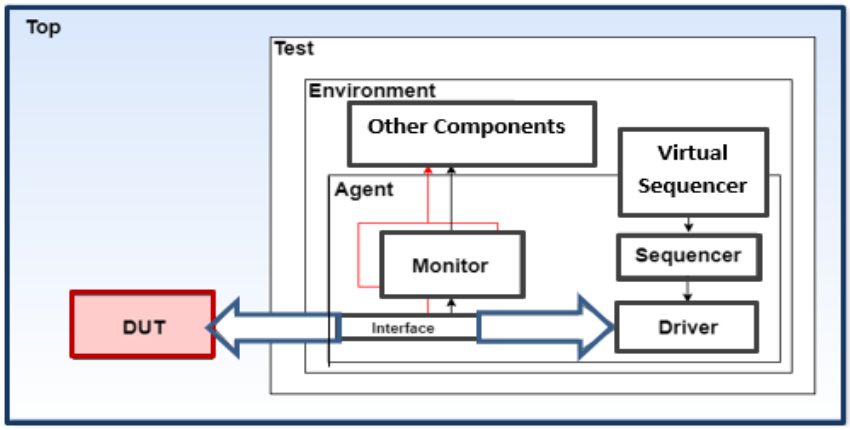

Fig. 4UVM framework

4.2.1 UVM DRIVER:

- An active entity which drives signals through Interface to the master(DUT).
- Transaction level objects is obtained from the sequencer by means of **Sequencer-Driver Handshake** mechanism to drive them through the Interface handles.

Fig5 gives a brief overview of the steps involved in developinga driver class.The *run phase* of the driver is where theactual logic of driver- master communication resides or happensthrough the virtual interface handle.Fig 6 show the structuraldefinition of driver-monitor communication task, a continuousprocess built in the run phase of UVM DRIVER.

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 7.122|

||Volume 9, Issue 5, May 2020||

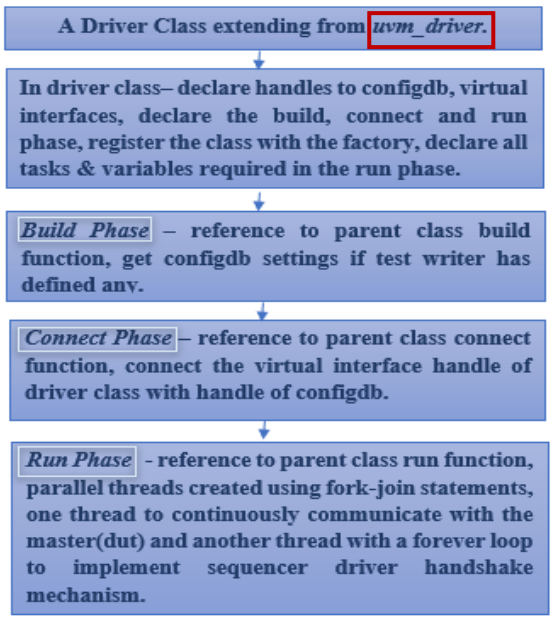

Fig. 5Flowchart of UVM DRIVER code

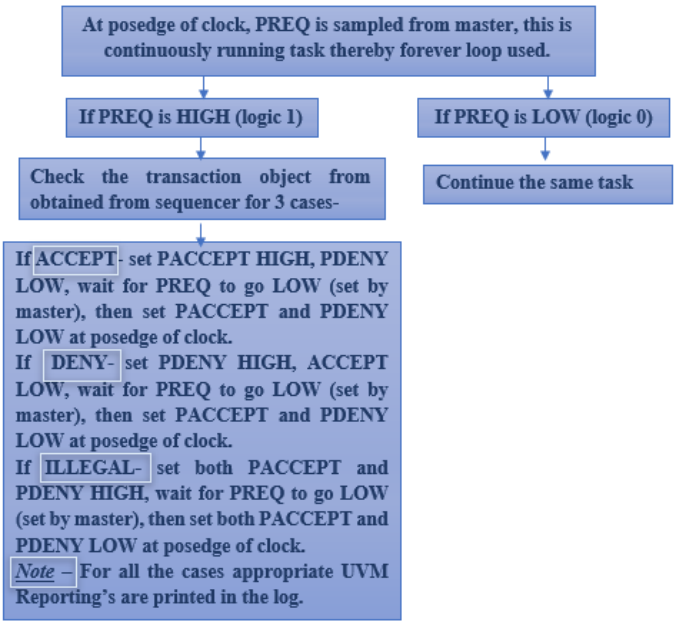

Fig. 6Task flow for driver and master communication in run phase of UVMDRIVER

4.2.2 UVM MONITOR:

- As the name suggests, it is responsible for capturing or recording the activity of the signals on the Interface.
- It translates these activities into transaction level objects and broadcasts them to various other components like scoreboard, coverages/collectors etc.

Fig7 shows the structuring of code for the UVM MONITORclass.

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 7.122|

||Volume 9, Issue 5, May 2020||

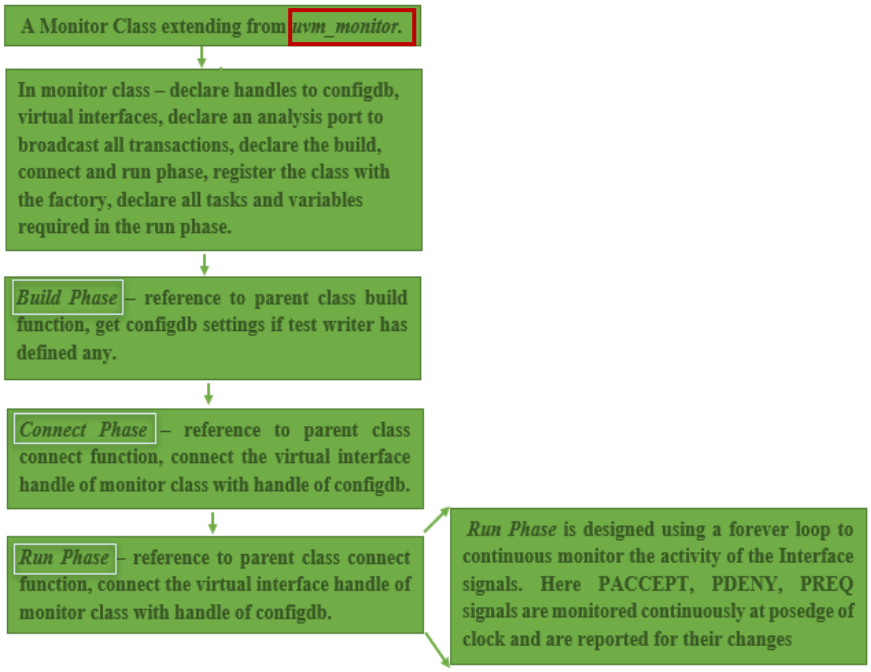

Fig. 7Flowchart of UVM MONITOR code

The highlighted parts in the Fig 6 and Fig 7 are class libraries or parent class where the driver and monitor codes (references) are extended or derived from.

4.2.3 UVM SEQUENCE:

- It is made up of several data items which can be puttogether with several permutation and combination tocreate varied interesting scenarios.
- They are executed by the assigned sequencer when a transaction request is made by tester.

```
SV/Verilog Testbench
task slave_basic_1_seq::body();
    req = p_channel_slave_transaction::type_id::create("req");
    start_item(req);

    // Randomize the sequence and start it on the sequencer.
  if(!req.randomize with {m_action==ILLEGAL_REQUEST; m_delay_before_action==0;})
    begin
        `uvm_fatal(get_type_name(), "[RAND_FAILED]: Randomization failed due to violation of
transaction constraints.")
    end

    finish_item(req);// till there is a item done from driver this will be hanging
    get_response(rsp);

endtask : body
```

Fig. 8. Constraint Driven Randomization of a sequence defined in sequence body

```
task slave_basic_1_seq::body();
    req = p_channel_slave_transaction::type_id::create("req");

    `uvm_do_with(req,{m_action==DENY_REQUEST; m_delay_before_action==0;})

    get_response(rsp);

endtask : body
```

Fig. 9. Constraint Driven Randomization of a sequence using UVM Macroas defined in sequence body

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 7.122|

||Volume 9, Issue 5, May 2020||

Fig 8 shows setting of a transaction by the test writer. A failure of randomisation results in failure of execution of the sequence therefore a *uvm_ fatal* reporting is being printed. Fig 9shows the same setting of transaction by using a **uvm macro.**

4.2.4 UVM SEQUENCER:

- It generates data transaction as class objects and passesthe sequences to the Driver for execution.
- It does high level to low level translation for the Driver to understand the task need to be performed.

4.2.5 UVM VIRTUAL SEQUENCER:

- It is used to have control of all the sequencers from central place.
- It is efficient to use a virtual sequencer when test cases are to be run in parallel.

4.2.6 UVM AGENT:

- It encapsulates the Driver, Sequencer and Monitor into single entity.
- It instantiates and connects components inside it via the Transaction Level Modelling (TLM) Interfaces.

4.2.7 UVM ENVIRONMENT:

- It is an encapsulation/wrapper class which can have multiple or single agents defined in them.
- It contains scoreboards, coverage metrices and collectorsor top-level monitor checkers.

4.2.8 UVM TEST:

- It is created as pattern to target checks and verify distinctparts of the design.
- A Verification Plan is available which has the requisitetestcases defined to check the DUT.

4.2.9UVM TOP:

- It is the root/base node in the hierarchy of UVM Framework.
- It is like a static container encompassing all the componentsalong where instantiation of all the verificationcomponents are done here.

V. RESULTS AND DISCUSSION

```
# ----------------------------------------------------------------------------------------------
# Name                                          Type                          Size  Value
# ----------------------------------------------------------------------------------------------
# uvm_test_top                                  testcase_l_test               -     @472
#   m_env                                       p_channel_env                 -     @479
#     p_channel_slave_agent_0                   p_channel_slave_agent         -     @492
#       slave_driver                            p_channel_slave_driver        -     @617
#         m_p_channel_slave_item_info_port      uvm_analysis_port             -     @640
#         rsp_port                              uvm_analysis_port             -     @632
#         seq_item_port                         uvm_seq_item_pull_port        -     @624
#       slave_monitor                           p_channel_slave_monitor       -     @757
#         m_p_channel_slave_item_collected_port uvm_analysis_port             -     @764
#       slave_sequencer                         p_channel_slave_sequencer     -     @648
#         rsp_export                            uvm_analysis_export           -     @655
#         seq_item_export                       uvm_seq_item_pull_imp         -     @749
#         arbitration_queue                     array                         0     -
#         lock_queue                            array                         0     -
#         num_last_reqs                         integral                      32    'd1
#         num_last_rsps                         integral                      32    'd1
#     scoreboard                                p_channel_scoreboard          -     @608
#       p_channel_slave_agent_0_driver_export   uvm_analysis_export           -     @881
#       p_channel_slave_agent_0_driver_fifo     uvm_tlm_analysis_fifo #(T)    -     @834
#         analysis_export                       uvm_analysis_imp              -     @873
#         get_ap                                uvm_analysis_port             -     @865
#         get_peek_export                       uvm_get_peek_imp              -     @849
#         put_ap                                uvm_analysis_port             -     @857
#         put_export                            uvm_put_imp                   -     @841
#       p_channel_slave_agent_0_monitor_export  uvm_analysis_export           -     @826
#       p_channel_slave_agent_0_monitor_fifo    uvm_tlm_analysis_fifo #(T)    -     @779
#         analysis_export                       uvm_analysis_imp              -     @818
#         get_ap                                uvm_analysis_port             -     @810
#         get_peek_export                       uvm_get_peek_imp              -     @794
#         put_ap                                uvm_analysis_port             -     @802
#         put_export                            uvm_put_imp                   -     @786
#     virtual_sequencer                         p_channel_virtual_sequencer   -     @499
#       rsp_export                              uvm_analysis_export           -     @506
#       seq_item_export                         uvm_seq_item_pull_imp         -     @600
#       arbitration_queue                       array                         0     -
#       lock_queue                              array                         0     -
#       num_last_reqs                           integral                      32    'd1
#       num_last_rsps                           integral                      32    'd1
# ----------------------------------------------------------------------------------------------
```

Fig. 10Ports used in UVM Testbench as obtained from log file

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 7.122|

||Volume 9, Issue 5, May 2020||

Fig. 11Simulator waveform for ACCEPT Transaction

```
# ----------------------------------------------------------------------------------------------------------------------------------
# Name                          Type                        Size  Value
# ----------------------------------------------------------------------------------------------------------------------------------
# req                           p_channel_slave_transaction -     @912
#   m_action                    action_type_e               2     ACCEPT_REQUEST
#   m_delay_before_action       integral                    32    'h0
#   begin_time                  time                        64    0
#   depth                       int                         32    'd2
#   parent sequence (name)      string                      8     subSeq_0
#   parent sequence (full name) string                      67    uvm_test_top.m_env.p_channel_slave_agent_0.slave_sequencer.subSeq_0
#   sequencer                   string                      58    uvm_test_top.m_env.p_channel_slave_agent_0.slave_sequencer
# ----------------------------------------------------------------------------------------------------------------------------------
```

Fig. 12 ACCEPT Transaction creation as obtained from log file

```
# UVM_INFO p_channel_slave_driver.sv(120) @ 0: uvm_test_top.m_env.p_channel_slave_agent_0.slave_driver [p_channel_slave_driver] stored transaction
# UVM_INFO p_channel_slave_driver.sv(183) @ 90: uvm_test_top.m_env.p_channel_slave_agent_0.slave_driver [p_channel_slave_driver] Request  not seen from master
# UVM_INFO p_channel_slave_driver.sv(183) @ 100: uvm_test_top.m_env.p_channel_slave_agent_0.slave_driver [p_channel_slave_driver] Request  not seen from master
# UVM_INFO p_channel_slave_driver.sv(183) @ 110: uvm_test_top.m_env.p_channel_slave_agent_0.slave_driver [p_channel_slave_driver] Request  not seen from master
# UVM_INFO p_channel_slave_driver.sv(183) @ 120: uvm_test_top.m_env.p_channel_slave_agent_0.slave_driver [p_channel_slave_driver] Request  not seen from master
# UVM_INFO p_channel_slave_driver.sv(183) @ 130: uvm_test_top.m_env.p_channel_slave_agent_0.slave_driver [p_channel_slave_driver] Request  not seen from master
# UVM_INFO p_channel_slave_driver.sv(183) @ 140: uvm_test_top.m_env.p_channel_slave_agent_0.slave_driver [p_channel_slave_driver] Request  not seen from master
# UVM_INFO p_channel_slave_driver.sv(163) @ 150: uvm_test_top.m_env.p_channel_slave_agent_0.slave_driver [p_channel_slave_driver] Request seen from Master
# UVM_INFO p_channel_slave_driver.sv(167) @ 150: uvm_test_top.m_env.p_channel_slave_agent_0.slave_driver [p_channel_slave_driver] Accept Passed
# UVM_INFO p_channel_slave_monitor.sv(102) @ 150: uvm_test_top.m_env.p_channel_slave_agent_0.slave_monitor [p_channel_slave_monitor] Request Noticed
# UVM_INFO p_channel_slave_monitor.sv(127) @ 180: uvm_test_top.m_env.p_channel_slave_agent_0.slave_monitor [p_channel_slave_monitor] Accept Seen
# UVM_INFO p_channel_slave_driver.sv(183) @ 280: uvm_test_top.m_env.p_channel_slave_agent_0.slave_driver [p_channel_slave_driver] Request  not seen from master
# UVM_INFO p_channel_slave_driver.sv(163) @ 290: uvm_test_top.m_env.p_channel_slave_agent_0.slave_driver [p_channel_slave_driver] Request seen from Master
# UVM_INFO p_channel_slave_driver.sv(167) @ 290: uvm_test_top.m_env.p_channel_slave_agent_0.slave_driver [p_channel_slave_driver] Accept Passed
# UVM_INFO p_channel_slave_monitor.sv(102) @ 290: uvm_test_top.m_env.p_channel_slave_agent_0.slave_monitor [p_channel_slave_monitor] Request Noticed
# UVM_INFO p_channel_slave_monitor.sv(127) @ 320: uvm_test_top.m_env.p_channel_slave_agent_0.slave_monitor [p_channel_slave_monitor] Accept Seen
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 400: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
```

Fig. 13 uvm_info statements from log file for ACCEPT Transaction

```
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   39
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     3
# [RNTST]     1
# [TEST_DONE]     1
# [UVMTOP]     1
# [p_channel_env]     1
# [p_channel_slave_agent]     2
# [p_channel_slave_driver]    17
# [p_channel_slave_monitor]     7
# [p_channel_slave_sequencer]     2
# [p_channel_virtual_sequencer]     1
# [testcase_1_test]     1
# [uvm_test_top.m_env.scoreboard]     2
# ** Note: $finish    : /usr/share/questa/questasim/linux_x86_64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 400 ns  Iteration: 53  Instance: /p_channel_testbench_top
# End time: 05:57:18 on Jun 11,2020, Elapsed time: 0:00:07
# Errors: 0, Warnings: 17
```

Fig. 14. UVM Report summary as obtained from log file

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 7.122|

||Volume 9, Issue 5, May 2020||

Fig 10 shows the details of ports used in the UVM Components. Constraint driven randomization of ACCEPT, DENY and ILLEGAL transactionswere run in the simulator and results were verified for the same. Fig 11 shows the waveform of the master (DUT)signals when simulation is carried out for ACCPET transaction. Similar waveforms are obtained for DENY and ILLEGALas well. Fig 12 show log for the creation of ACCEPT Transaction item. We used *uvm_info* statements as reporting mechanisms to check the overall UVM Process and for our readability purpose.Fig 13 shows some of the info statements used in the driverand monitor module of the UVM Testbench. A UVM ReportSummary as shown in Fig 14 is generated at end of simulationgiving counts of all the reporting's used in the test program.

VI.CONCLUSION AND FUTURE SCOPE

The UVM Testbench was capable of randomizing all thenecessary test cases according to the verification plan andresults were successfully obtained. The basic parts of the UVM Framework was well understood and learnings were inculcated. Corner case was dealtand covered. The need for the Standard Verification Platformwhich helps mainly in IP reuse, the flavour of pre-siliconverification and structured Verification Methodology was appreciated.
The future scope of the project can be extended to explorefurther intricacies in design which includes timings, clock gating concepts,synthesis etc.With respect to UVM Testbench;scoreboards, coverage metrices and assertions can be appendedto get deeper and wider insights of the complete protocol structure as well as to widen our horizon on UVM Framework.

ACKNOWLEDGMENT

The work reported in this paper is supported by **BMS College of Engineering**, Basavanagudi Bangalore-560019, byproviding Tools and Mentoring through Internship & SkillDevelopment Program. We would like to express our gratitudeto Mahesh R, Senior Engineer, Cisma (subsidiary ofVeriKwest Systems Inc.) Bengaluru, for providing guidancefor the project work.

REFERENCES

[1] C. Spear and G. Tumbush," System Verilog for Verification", Third Edition: A Guide to Learning the Testbench Language Features. Springer
Publishing Company, Incorporated, 2012.
[2] T Tarun Kumar, CY Gopinath, "Verification of I2C Master Core using System Verilog-UVM", International Journal of Science and Research
(IJSR), Volume 3 Issue 6, June 2014.
[3] https://static.docs.arm.com/ihi0068/c/IHI0068C low power interface spec.pdf
[4] Accellera Organization, Universal Verification Methodology (UVM) 1.1d Class Reference, March 2013
[5] IEEE Computer Society and the IEEE StandardsAssociation Corporate Advisory Group, "IEEEStandard for System Verilog - Unified Hardware
Design, Specification, and Verification Language",IEEE Std 1800™-2012, 21 February 2013.
[6] IEEE Computer Society "IEEE Standard Verilog®Hardware Description Language", IEEEStd 1364™-2005, 07 April 2006.
[7] Geng Zhong, Jian Zhou, Bei Xia, "Parameter and UVM, Making a Layered Testbench Powerful", IEEE 2013.
[8] Suchika Lalit, Ashish Prabhu "UVM Based Verification of CAN Protocol Controller Using System Verilog", International Journal on Recent and Innovation Trends in Computing and Communication, Volume: 3 Issue: 5, May 2015.
[9] Chris Spear, System Verilog for Verification, A Guide to Learning the Testbench, MA, Springer, p.15(2006).

BIOGRAPHY

A VIPULA is currently pursuing her undergraduate 4-year program in Electronics & Communication Engineering from BMS College of Engineering, Bangalore, India. Her area of interest lies in VLSI Design, Testing and Verification.

NIKKITHA N is currently pursuing her undergraduate 4-year program in Electronics & Communication Engineering from BMS College of Engineering, Bangalore, India. Her area of interest lies in VLSI Design and Embedded Systems.

Dr KIRAN BAILEY is a dedicated and hardworking academician with over 20 years of experience by working as an Assistant Professor in Department of Electronics & Communication, BMS College of Engineering, Bangalore, India

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 7.122|

||Volume 9, Issue 5, May 2020||

and has vast experience in curriculum design, lab set up and Industry interaction. She has more than 35 publications and has also filed for patents with her research interest lying in multiple VLSI Domains.

LOGANATH RAMACHANDRAN, Ph.D. is a verification expert with more than 25 years of EDA experience. He is the founder of VeriKwest System Inc., Santa Clara, USA. Loaganath served as a member of the IEEE committee that standardized SystemVerilog and currently serves on the technical programme committees for various international conferences including DVCON USA and DVCON INDIA.