International Journal of Engineering Research and Advanced Technology (IJERAT)

DOI: http://dx.doi.org/10.31695/IJERAT.2018.3279

E-ISSN : 2454-6135

Volume.4, Issue 6

June -2018

Verification cases and procedure for IP-core development

Dr.Manju Nanda[1] and P Rajshekhar Rao[2]

Sr. Principal Scientist [1], Research Scholar [2]

Aerospace Electronics & Systems Division[1]

CSIR- National Aerospace Laboratories[1]

Bangalore[1]

Department of Avionics[2]

Inst. Of Science & Technology[2]

Jawaharlal Nehru Technological University[2]

Kakinada[2]

India

ABSTRACT

The main objective is to perform the module level testing of requirement(MLITP) given and verify the completeness, compliance and correctness of the implemented functionality with reference to the Hardware Design Data (HDD) requirements followed by functional required document(FRD) mapped to hardware required document (HRD) .Basically this paper tells about for any module under functionality testing to perform as per the requirement, some mapping to be follow as per DO-254 which includes the safety analysis as priority and traceability requirement plays a vital activity to complete the functionality testing under test cases scenarios. Regardless of whether the design and verification tools can demonstrate that a plan is practically right under all conditions that were considered, design errors in the hardware can even now happen because of conditions beyond the scope of the design tools. The most ideal approach to maintain a strategic distance from these mistakes is to have an accomplished design and verification staff with learning of the device confinements. This will enable the group to distinguish potential issues while still in the design phase and permit error mitigation techniques to be incorporated.

Key Words: Hardware requirement document, tool usage, code coverage, IP-core, FPGA, Hardware functional requirement.

1. INTRODUCTION

Verification cases and procedure includes test environment like requirement-based verification (RBV).). Addition to RBV examinations, for example, elemental analysis, formal analysis, and safety specific analysis are performed. . In many regards, this procedure is a parallel to RTCA/DO-178 verification requirement for software. Concentrating on simply, the verification procedure, the RTCA/DO-254 confirmation stream for a DAL. A design can be abridged as takes after[1]:

1.1. Requirements based verification

Use guided test vectors to confirm all requirements.

1.2. Elemental analysis

Several measurements might be utilized however explanation scope best matches the necessities of basic examination.

1.3. Formal analysis

Is proposed for equipment with simultaneousness or adaptation to non-critical failure.

1.4. Robustness testing

Is added to guarantee the device capacities effectively in all legitimate conditions.

The hardware design plan describes the procedures, methods and standards to be applied and the processes and activities to be conducted for the design of the hardware item.

The hardware testing process involves three mode of testing they are:

1.4.1. Module level testing

In the unit level testing, a module is confirmed in its own test condition to demonstrate that the logics, control, and information ways are practically right. The objective of module level test is to guarantee that the segment/unit being tried complies with its determinations and is prepared to be coordinated with different subcomponents of the item. In unit level confirmations great coverage rate is normal. It is the little nuclear module in the IP core utilitarian piece, which can work autonomously. Module testing is unit trying, i.e. testing a specific module. Normally a task is separated into smallest atomic known as modules, testing these modules independently to test in the event that they are working is known as module testing. It is only the unit testing [2].

1.4.2. Integration level testing

Integration Testing comprises of the efficient combination and execution of product components. Various levels of integration testing are conceivable with a combination of equipment and programming parts at a few extraordinary levels. The objective of integration testing is to guarantee that the interfaces between the components are right and that the product segments consolidate to execute the product’s usefulness accurately [4].

1.4.3. Target level testing

Target level testing will addressed in high-level interface test plan (HLITP) document. Each module will be tested for the functionality. The effected terms or scenarios while implementing in advance tool verification they are

1.4.3.1. Functionality

Each module will be tested with respect to requirement, architecture, and design.

1.4.3.2. Performance

Each module will be tested to performance stated against the design.

1.4.3.3. Safety

Each module will be tested with respect to safety of IP- core with respect to Level A.

1.4.3.4. Coverage

Each module will be tested with respect to coverage for statement, branch, condition and toggle.

Test cases will be generated to monitor errors such as an invalid instructions operation code, divisible by zero, division overflow, and single event upset as shown in below Table 1.1.

Table1.1. Hardware terms for IP-core

Terms	descriptions
Compliance	**A Compliance is the satisfaction of a standard guidelines objectives**
Finding	**A Finding is identification of non-compliance to a standard guidelines objective**
Observation	**An Observation is identification of a potential process improvement**
Soft IP-core	**Soft IP-cores are the category of IP-core that comes to the user with the most lifecycle data. This data generally include register transfer level (RTL) descriptions in languages such as Verilog or VHDL. This allows a detailed analysis and optimization (and eventually customization) of the soft IP-cores for the intended application. Soft IP-cores still need to be synthesized, placed and routed (P & R) in the target Airborne Electronics Hardware (AEH) device.**
Firm IP-core	**Firm IP-cores are next in the decreasing level of design description, specified in technology-independent netlist level format. This allows the IP provider to hide the critical IP details and yet allow the IP user to perform some limited amount of analysis and optimization during placement, routing, and technology-dependent mapping of the IP block. Firm IP-cores still need to be placed and routed in the AEH device.**
Hard IP-core	**Hard IP-cores have the least design description and lifecycle data, specified in technology-dependent physical layout format using industry standard languages such as stream, polygon, or GDSII format. Hard IP-cores can be thought of as a "black box" that, due to the lack of knowledge about the internal detailed design, they cannot be fully analyzed and/or co-optimized. Hard IP-cores come with a detailed specification of integration requirements in terms of clock, testing, power consumption, interfaces and a host of other parameters. Hard IP-cores are embedded in the PLD/ASIC at the silicon level.**

2. OBJECTIVES

Hardware system process mapped to hardware requirement as per DO-254 which is allotted to the hardware item to be approved before the design implementation. In addition to the requirement tracer for compatibility the system performance for design requirement to correct the design code as well as FPGA specification. The mechanism for recognizing requirements attributes (e.g., validated, derived, safety-critical, and so on), as this can enable you to track the suitable exercises related with different classifications of necessities, for example, robustness testing of security basic properties and system level validation of derived requirements. ReqTracer, presented in the area Requirements Capture (Including Management and Traceability) can help with these approval assignments. Processing procedure or verification procedure is to verify the Complex Electronics Hardware which approaches DAL A/B. This guideline verifies the test results and reports generation as per design code level [7].

2. Overview of approach

At integration level, the functionality, safety and performance will be tested with the input and output signal, interfaces between the modules, data flow and signal control flow for the IP-core.

2.1. Functionality

- What is the trigger point to transmit the data?
- What pattern of the data is to be sent?
- In case the trigger does not come, then what?

DOI : http://dx.doi.org/10.31695/IJERAT.2018.3279

- In case the data pattern fails to transmit at required rate, then what?
- In case the clock fails, then what?
- Signals control flow analysis.

2.2. Safety

It will be tested for the safety of the IP-core and related hardware. Any failure in the system the component should able to retrieve by itself. Testing method will define the set of criteria to be applied in the testing of the IP-core. The criteria will be tested and demonstrated as follows.

Table1.2. Testing methods for IP-core

S.no	Mode of testing	Description	Tool usage
1.	Module level testing	Functionality	RTL design code to behavioural
2.	Integration level testing	Inter-module interaction, control flow of the information and data flow	Post synthesis verification model to function simulation. Post PNR verification model HDL to function simulation and static timing analysis.
3.	Target level testing	Bit stream generation	Chip scope pro-analyzer

PLD's hardware development cycle flow diagram with Validation/Verification process in parallel is shown in Figure 1.1. Activity flow down across design and verification/validation process is also shown.

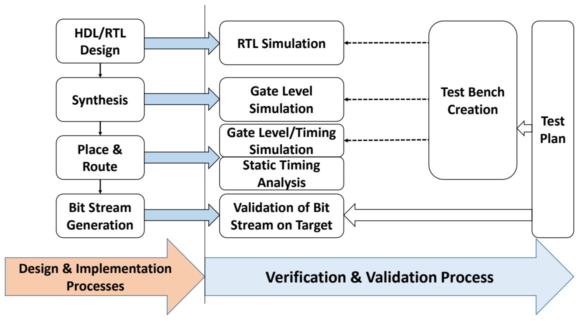

Figure1.1: Hardware Development Flow Diagram [9]

3. DETAILED DESIGN

The targets of the detailed design is to guarantee that the goals got from the framework prerequisites and equipment particulars can be meeting in the framework plan on a low level of the design. In the event that the framework depends on a FPGA the detailed outline can be composed in Verilog or VHDL code. The developments of test seats for framework and safety verification are likewise made in the point by point plan. Detailed design incorporates the majority of improvement work and spans hardware description language (HDL) coding through synthesis and furthermore place and route, however some may contend that place and route falls under the execution stage [10]. During the detailed design process, HDL code must be composed to certain design standards, confirmed, explored inside, reviewed remotely, kept under configuration management, and followed to program requirements they are:

3.1. Dealing with the design flow: HDL Designer gives an organized workplace that, among other benefits, enables a group to characterize the tool versions and scripts that will be keep running all through the process. Running a stage in the flow is as simple as clicking the appropriate button, for example, "Simulate" or "Synthesize". HDL Designer underpins Mentor Graphics design and verification tools, as well as numerous FPGA vendor tools.

3.2. Making/altering: HDL configuration is a content-based strategy to depict physical hardware and its functional behaviour. Since it is content based, it can be composed by means of a straightforward word processor. In any case, this can be an escalated, manual process. HDL Designer gives an option: a suite of advanced design editors (instead of straightforward content managers) to encourage design development, including Interface-Based Design (IBD) spreadsheets.

3.3. Imagining/archiving: Often made physically, HDL reports and artifacts are essential for enhancing the comprehension of the design (important for design reviews and code reuse) and are particularly valuable if the design must be reproduced later on. HDL Designer robotizes the creation such documents and artifacts as pictures or design representations.

3.4. Code checking: DO-254 requires that groups characterize the principles they will use in a design procedure, including the coding benchmarks they should stick to. These standards and rule assist maintain a strategic distance from downstream issues with the design or configuration process. In a DO-254 program, the design code can be physically investigated against these norms as a feature of the design surveys; however, this can be an excruciating, time-consuming, costly, and error-prone approach. A superior technique is to enrol the assistance of a tool to consequently do this kind of checking and after that essentially audit the outcomes as a major aspect of the design surveys. HDL Designer incorporates a HDL coding rules checker (or linter), that incorporates an arrangement of predefined (and modifiable) manage sets. Essential among these is the “DO-254 Rule set”, an arrangement of configuration checks got from genuine venture encounters with organizations doing wellbeing and mission-basic plan and assembled with contribution from roughly 20 individuals from the DO-254 User Group.

3.5. Evaluating/examining: The second DO-254 review, or SOI-2, is commonly a design audit. Preceding the official confirmation review, the design group ought to have had various interior surveys that cover the architecture, changes to requirements, coding, following HDL code to the requirements, etc. HDL Designer can help encourage such surveys by indicating source code, charts, venture chain of importance, and an assortment of other appropriate information, and after that catching these in a HTML site, which empowers venture audits crosswise over groups and topographies. The activities/results of the review can likewise be caught into this HTML arrangement to give evidence (a relic) of the survey procedure.

3.6. Tracing necessities: As it’s composed HDL code ought to likewise be connected back to the appropriate requirements, which is a procedure called &”labelling”. The far reaching altering situations of HDL Designer incorporate with Retrace’s (ReqTracer) “tagger” highlight to encourage the connecting of a HDL(hardware description language) usage to its necessities source [12].

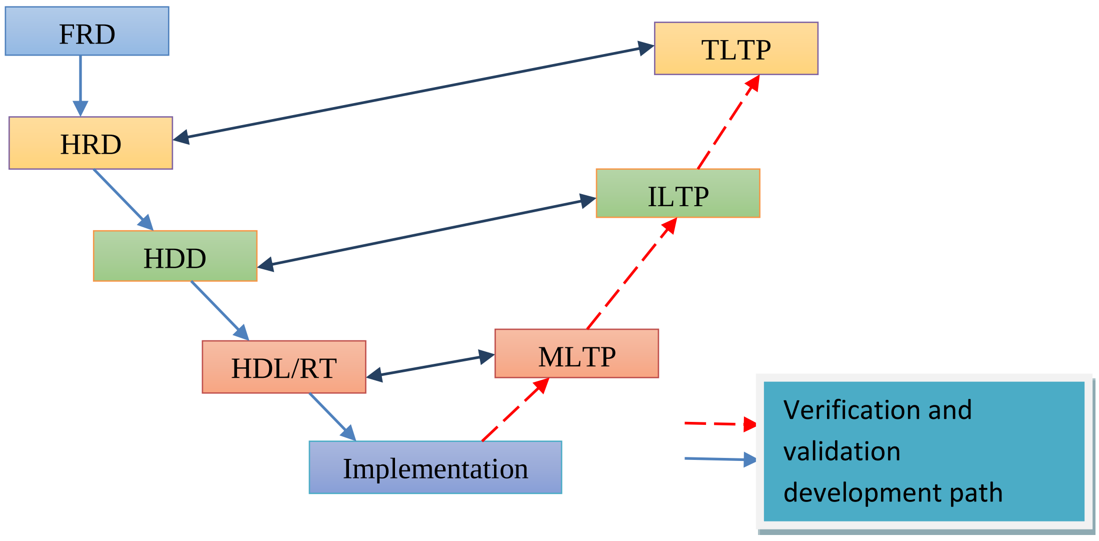

Figure.1.2. Requirement tracer [6]

4. METHODOLOGY

DOI : http://dx.doi.org/10.31695/IJERAT.2018.3279

To describe the extent level of individual module level testing on FPGA, nevertheless, the Verification and validation (V&V) approaches depicted will focus on the useful and execution parts of the prerequisites and particulars for the item. Methodologies for deciding if an product, fulfils its prerequisites and determinations as for wellbeing, compactness, ease of use, practicality, serviceability, security, and so on., although very important for many systems. It is possible to determine the applicability of various V&V approaches and techniques [14]

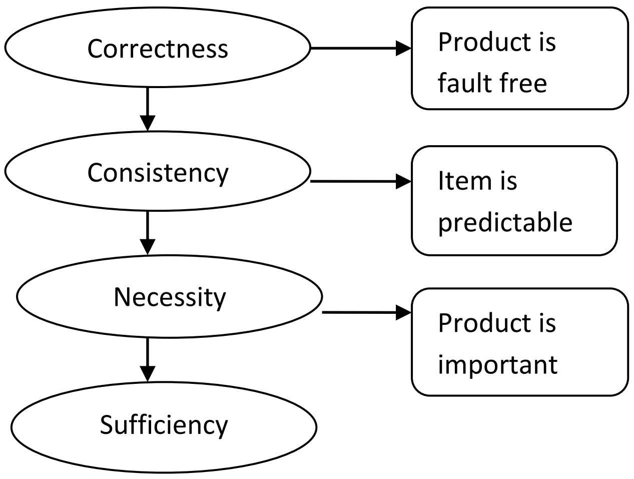

Figure1.3: Applicability of various V&V approaches [17]

5. TEST APPLICATION APPROACH

The approach for applying functional tests to the design. Ordinarily, this includes either preloading tests into on-chip memory and executing the test by means of the on-chip processor or applying the test all through the external interfaces to the device (test or functional interfaces).

5.1. Results Checking

Instructions to confirm the design's reactions to function tests. This can be done by self checking techniques, golden model (reference model) comparison, or comparing expected results files.

5.2. Test Definitions

Characterizes the tests that are to be performed and the model abstraction levels to which the tests are to be connected. In many occurrences, a similar test will be connected to several model levels. For each test, the verification technology or tool and the associated metrics ought to be incorporated. The metric shows when the test is finished. Metric can be characterized for combinations of tests. For instance, 100 percent statement coverage will be accomplished when executing the greater part of the simulation tests.

5.3. Test-bench Requirements

The test-bench necessities in light of investigating the contents of the verification definition table. The model sorts and abstraction levels, model sources, and test-bench components (checkers, stimulus, and so on) should be considered. For formal verification, characterize design properties and requirements.

5.4. Verification Metrics

Two classes of metrics ought to be tended to in the verification plan:

・ Capacity measurements: Identifies tool capacity assumptions (run times, memory size, disk size, and so on) and confirms that the assumptions made in creating the confirmation design remain constant hold true during the execution of that plan.

・ Quality measurements: Establishes when a verification undertaking is finished. Quality metrics incorporate functional coverage and code coverage.

5.5. Regression Testing

The technique for Regression testing is to regret the fault analysis done by simulation based testing. The test design detail when the regression tests are to be run (overnight, persistently, activated by change levels, and so on) and indicates the assets required for the regression testing. Regularly, once the plan has been acknowledged and checked at a specific level, a formal control methodology is put in place to deal with the plan updates to this brilliant model and the consequent re-verification. The test design ought to unmistakably state at what level of abstraction the regression tests are to be run and recognize the tests to be utilized. The regression test suite may be the full arrangement of tests recognized for the level of abstraction of the plan or a selected subset.

5.6. Directed Random Testing

The nature of a functional verification condition relies upon the stimulus that is connected to a DUT. A thorough test vector set can be composed utilizing all combinations of the input signals, yet this is not possible, since it builds the simulation time massively. In directed random testing, irregular address, data, and control signals are driven onto a bus, and at least one bus protocol checkers verifies that bus protocol violations do not happen because of these cycles. This testing approach is appropriate for bus validation. The test-benches are directed in that the test cycles created are not arbitrary nevertheless; make cycles that stress the design in particular ways. The pattern generators can be set to make particular exchange composes, for example, read, write, and read-modify write in a random sequence. For instance, 20 percent read, 30 percent write, 50 for read-modify-write. So also, data and address fields can be produced in a random succession, however inside determined cut-off points or utilizing a restricted arrangement of discrete esteems. These sorts of tests confirm corner conditions and successive or data-dependent circumstances that are difficult to distinguish in simulation. With this philosophy, any algorithmic errors are recognized and settled right on time in the design [16] [17].

6. RESULTS

6.1. Implementation

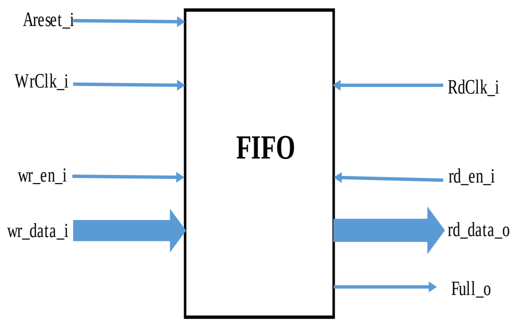

Figure1.4. Block diagram of FIFO

Figure.1.4. shows the block diagram of FIFO with six inputs and two outputs. The Buffer is works on the principle of Asynchronous FIFO. When Areset_i is high then all the signals are reset to its initial value. When the Areset_i is low and Wr_en_i is high then the 32 bit data from the previous module is written into the Buffer with respect to the WrClk_i pulses. When the Areset_i is low and Rd_en_i is high then the 32 bit data which is stored in the buffer will be fetched by the next module with respect to the RdClk_i pulses. If the Buffer is full then the Full_0 signal goes high which indicating that the data is available to read for the next module Module.

DOI : http://dx.doi.org/10.31695/IJERAT.2018.3279

6.2. Traceability

6.2.1. Hardware Design Data (HDD Requirement)

Hardware Design Data (HDD) requirements of Video Data Buffer which are to be tested as part of this module level testing are tabulated in below table for port mappings and for logical requirements.

Table 1.3. Generic configuration ports of FIFO

Requirement Identification	Generics Name	Value (in bits)	Interface Description
RDD_XXXX_006	**FIFO shall be interfaced with generic interfaces & corresponding values assigned as follows:**		
	IMPL_STYLE	**1**	**0 = distributed RAM based, 1 = Block RAM Based**
	FIFO_mode	**1**	**0 = STD mode FIFO, 1 = First word fall through (FWFT) FIFO**
	FIFO shall be interfaced with generic interfaces		
	Data_width	**32**	**data width of data in and out port**
	FIFO_depth_width	**13**	**FIFO depth configuration**

Table 1.4. Port map requirements for FIFO

Requirement Identification	Interface Name	Width (in bits)	Interface Description
RDD_XXXX_007	**FIFO module shall be interfaced with following port for reset input:**		
	Areset_i	1	**Areset_i**
RDD_XXXX_009	**FIFO module shall be interfaced with following port for writing into buffer:**		
	WrClk_i	**1**	**Pixel clock input interface for writing data**
	wr_data_i	**32**	**Output Data Stream as data Sequence in terms of**

Requirement Identification	Interface Name	Width (in bits)	Interface Description
			32 bits
	wr_en_i	1	Buffer Write Enable Signal. Control Signal to write the Data Into the FIFO. 1=Write; 0=Don't Write
	Full_o	1	FIFO Full signal output
RDD_XXXX_045			
	RdClk_i	1	
	rd_en_i	1	Buffer Read Enable Signal. Control Signal to read the Data from FIFO. 1=Read; 0=Don't Read
	rd_data_o	32	Data out

The above tables indicate the following interfaces as per the given requirement which captures the functional requirement followed by hardware requirement given in the design data requirement.

6.2.2. Libraries to include in Test Bench

Include the following libraries part of the test bench:

```
LIBRARY ieee;

LIBRARY modelsim_lib

USE ieee.std_logic_1164.ALL

use IEEE.STD_LOGIC_unsigned.ALL

use IEEE.STD_LOGIC_arith.ALL

use IEEE.STD_LOGIC_textio.ALL

use std.textio.all

use IEEE.numeric_std.ALL

use modelsim_lib.util.all;
```

6.2.3. Steps to Monitor Design's Local Signal

The following procedural steps allows tester to monitor the design's local signals inside the test bench simulation scenario:

1. Include the library such as modelsim_lib & use modelsim_lib.util.all;

DOI : http://dx.doi.org/10.31695/IJERAT.2018.3279

2. Generate the un-clocked process with WAIT Statement within it.

3. Use the init_signal_spy function call as mentioned below:

SYNTAX:

```
process

begin

                    init_signal_spy ("/<Source Signal with Path>/ ", "/<Destination Signal with Path>/" );

wait;

end process;
```

6.2.4. Steps to log the report in the transcript window

1. Create a folder ARINC818_Tx_Test in Drive (x) in local system.
1. Copy XXXXXXX.vhd from CC, XXXXXX_TB06 and expected output files for assertion based testing to be copied from CC,
2. Open the QuestaSim tool
3. Select file > Change Directory >Enter
4. Browse for folder window select the XXXXX_Test folder.
5. Select Compile > compile >enter
6. In Compile Source File window select both source code and Test bench files and enter Compile button.
7. If Create library pops-up “The library “work” does not exist. Do you want to create this library?” enter Yes to create work library.
8. Observe and make sure for the Errors:0 and warnings:0.
9. After compilation enter done button.
10. Select Simulate > start Simulation >enter
11. In start Simulation window> Design >enter
12. Select Work library and expand (click on +)
13. Select Test bench file name (.VHD) and Resolution 100ps and select OK.
14. Sim-Default right click on the test bench then add to >wave> All items in design >enter
15. All signal will be added to the Wave Default.
16. In transcript window run simulation for 1 sec to verify the parameters with different resolution and the total no. of frames per sec.
17. After run simulation completes then analyze the signals

Assertion is used to report the result of the test case in the transcript window

```
If test case is passed then

report "test case is passed" severity note;

else

assert false report "test case is failed" severity error;
```

6.2.5. Test Cases

The test cases are generated for the Normal scenarios for functional verification to cover the Hardware Design Data (HDD) requirements which are mentioned in the table 1.3.. Coverage Report will be generated based on the test cases.

6.2.5.1. Normal Test cases

All input and output of FIFO are to be verified for correct port mapping with Inst_Async_FIFO. The port interfaces are compared with known values as per the HDD requirements. Assertions are used to verify whether the respective port mapped signals are matched or not matched by comparing the expected and observed value.

6.3. Procedure

Test Case 1: Testing the port interfaces in port mapping level for all port mapped signals

(The input signals will be given to the module and check the output at the input /output interface of the instantiated FIFO buffer)

This test case covers above requirement. The objective of this test case is to test the Generic and Port mapping of the Async_FIFO instance in the module.

The following test conditions are to be followed:

1. Write a single Test bench with information as provided below.

2. Generate clock from test bench for input port WrClk_i of XXXX MHz (Time period= XXXX ns).

3. Generate clock from test bench for input port RdClk_i of XXXX MHz (Time period= XXXX ns).

4. All Inputs ports shall be assigned to any value based on the width of the input port.

5. Tapped the port signals of Inst_Async_FIFO to the test bench using init_signal_spy process.

6. Compare the values at the input ports of Inst_Async_FIFO to the input ports of buffer and assert true/false.

7. Compare the values at the output ports of buffer to the output ports of Inst_Async_FIFO and assert true/false.

6.4. Limitation

Firstly, Difficulty of Testing All Data- For most projects, it is illogical to endeavour to test the program with every single conceivable contribution, because of a combinatorial explosion. For those inputs selected, a testing prophet is expected to decide the accuracy of the yield for a particular test input. Secondly, Difficulty of Testing All Paths- For most projects, it is unfeasible to endeavour to test all execution ways through the product, because of a combinatorial explosion. It is also not possible to develop an algorithm for generating test data for paths in an arbitrary product, because of the powerlessness to decide path feasibility.

7. CONCLUSION

In HW/SW co-verification, integration and verification of the hardware and software occurs simultaneously. The Co-verification condition gives a graphical UI (GUI) that is steady with the present equipment test systems and programming emulators/debuggers that are utilized by the equipment and programming venture improvement groups. This empowers the software group to execute the software straight forwardly on the hardware design. Additionally, the hardware design is stimulated with real input stimulus in this way decreasing the endeavours required to creator the hardware test benches.

Highlights:

- Verifies both hardware and programming ahead of schedule in the design cycle. It offers adequate execution to run the interface certainty tests, code sections, and individual driver and utility code.

Restrictions:

- Co-verification situations accessible today don't offer adequate execution to run finish application programming over the target real-time operating system (RTOS) because of capacity and simulation speed problems.

DOI : http://dx.doi.org/10.31695/IJERAT.2018.3279

REFERENCES

[1] DO-254/ED-80, Design Assurance Guidance for Airborne Electronic Hardware, RTCA/ EUROCAE.

[2] MIL-STD-1629A, Procedures for Performing a Failure Mode, Effects and Criticality Analysis, US DoD, Washington D.C.

[3] Prakash Rashinkar, Peter Paterson, Leena Singh, 'System- on-a- chip verification', Kulwer Academic Publishers, Newyork, Boston, London, Moscow.

[4]Application Note, ASIC design guidelines, Atmel Corporation, 1999

[5]Cummings CE, Sunburst Design, Inc.; Mills D, LCDM Engineering (2002) Synchronous resets? Asynchronous resets? I am so confused! How will I ever know which to use? SNUG, San Jose

[6]Application Note, Clock skew and short paths timing, Actel Corporation, 2004

[7]ARP 4754 Certification considerations for highly-integrated or complex aircraft systems; SAE Systems Integration Requirements Task Group AS-1c, ASD, Society of Automotive Engineers Inc.

[8]ARP 4761 Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment, SAE Committee S-18, Draft 13a, Society of Automotive Engineers Inc.

[9]Bauer, M. and W. Ecker: "A VHDL-Based Hierarchical, Highly Flexible and Extendable Testbench Approach". International High Level Design Validation and Test Workshop, 1996.

[10]Swavely, G.W., Beaton, J., and W. Debany: "A Generic VHDL Testbench to aid in Development of Board-Level Test Programm", AUTOTESTCON, 1994.

[11]Heinkel, U. and W.H. Glauert: An Approach for a Dynamic Generation / Validation System for the Functional Simulation Considering Timing Constraints. European Design and Test Conference, 1996.

[12]Zinn, A., Ecker, W., Bauer, M. and B. Bernard:"Comparison of Sequential VHDL and C Revised". International HDL Conference, 2001.

[13]Sztipanovits, J.: "Engineering of Computer-Based Systems: An Emerging Discipline", Conference and Workshop on Engineering of Computer-Based Systems, 1998.

[14]Drager, S.L., Hanna, J.P., and R.G. Hillmann: "VHDL Model Verification and System Life Cycle Support", VHDL International User Forum, Spring 1996.

[15]Goldbach M., Grams H., Glauert W., Hartl W. and Voit G.: "Simulation-Based Test Programm Verifiaction Using the SZ Testsystem Environment", IMSTW, 1998.

[16]Garbe, H., Jentschel, H.J., and R. Kaminski: "Multilevel Simulation in the Design of Communication Systems", NE Science, 1994.

[17]Delvai M., Huber W., Rahbaran B., and A. Steininger: "An FPGA-Based Development Platform for the Virtual RealTime Processor Component SPEAR". IEEE International Workshop on Design and Diagnostics of Electronic Circuits and Systems, 2002.

[18]Lentz, K.P., Heller, J., and P.L. Montessoro: "System Verification using Multilevel Concurrent Simulation", IEEE Micro, 1999, p.60-67.

[19]M. H. Schulz and E. Auth, "Improved deterministic test pattern generation with applications to redundancy identification," IEEE Trans. on CAD, Vol. 8, No. 7, July 1989, pp. 811-816.

[20]Neil H.E. Weste and David Harris, CMOS VLSI Design: A Circuits and Systems Perspective (3rd Edition), Addison Wesley; 3rd edition (May 11, 2004), ISBN: 0321149017.

[21]Y. Kim, M.-H. Yang, Y. Lee, and S. Kang, ―A new low power test pattern generator using a transition monitoring window based on BIST architecture,‖ in Proc. Asian Test Symp. (ATS), Dec. 2005, pp. 230–235.

DOI : http://dx.doi.org/10.31695/IJERAT.2018.3279