Implementation and Verification of Rgb to Grayscale Converter Ip Using System Verilog

Shyamsundar Iyer, S.M.Sakthivel

Abstract*: In this paper, a generic rgb to gray scale converter intellectual property (IP) core is implemented and verified. First verification environment for the device under test (DUT) is implemented using system verilog class libraries and Qsys tool of Quartus software. Then the device under test (DUT) is verified by performing two tests. First test is video file reader test which is carried out to check the functionality of the device under test and another test is constrained random test which is performed to verify the device under test outputs with randomly generated expected outputs. Both the test are performed using Modelsim software.

Index Terms*: Intellectual Property (IP) core, Verification, Device Under Test (DUT), Verification Environment, System Verilog.

I. INTRODUCTION

A. Introduction to Verification

Verification is a procedure where the design is tested or verified in opposition to a given specification of the design before the chip is taped out. Verification is carried out in parallel with the implementation and development of the design and can be started from the time where the design architecture definition is in progress. The main aim of the verification process is to guarantee whether the design is functionally correct without any bug inside before the design is sent for fabrication and finally gets taped out of the company and reaches the customer for the use. However due to increasing complexities of the design, the extent of verification is also unfolding to include much more than the functionality of the design [1]. This includes verification of power and performance of the design, safety and security features of the design and design complexities with multiple asynchronous clock domains in the design. Design simulation (RTL Design) remains the primary focus for verification. Many other methodologies have come up like formal verification, power-aware simulations, FPGA or emulation prototyping, dynamic and static checks etc. These methodologies are also used for effectively verifying all the features of design before taping out the design. The process of verification is considered very important as part of the lifecycle of design. This is because any serious bugs in the design is not analysed and corrected according to the specification of the design will result into faulty chips thereby wasting the overall cost of the design process.

Revised Manuscript Received on May 10, 2019

Shyamsundar Iyer, School of Electronics Engineering, VIT University, Chennai, India.

S.M.Sakthivel, School of Electronics Engineering, VIT University, Chennai, India.

B. Need of Verification

Verification imbibes almost 60% to 75% of the effort of whole design process and plays a vital role in the design flow of multimillions of gate ASICs, so the process of verification becomes a must need technique involved in the design process. Majority of the chips requires redesign which is mainly due to the effect of the bugs present in the design.

The primary focus of the verification process will be determining the failures in the design thereby identifying the presence of bug in the design. Bug may also occur in the chip if there is a mistake when RTL designer designs or codes according to the given design specification. If this bug gets executed it may end up giving wrong results to the customers resulting in a failure of chip. A single mistake may lead to variety of failures in the system.

Not all the bug present in the system will result in to failure of the system. A bug present in the code which is dead will never result in failure. Also not all bugs occur due to error while coding. There are also possibilities that error might be in the design specifications itself.

So verification is in much need to tape out the chip without any bugs and to reach the customers in time. Verification of the design also depends upon the complexity of the design [2]. Also in today's increasing complexity of the design, verification process also became difficult to perform. To ease this situation many verification languages have come up to smoothly perform the verification of the design. Verification languages like Open Vera (OV), Specman e, System Verilog have helped verification engineers to verify even the complex design in a quite easier way.

However not only the complexity of the design will be an obstacle for the verification process but also the size of the design will create problem in both specification and functional level. This situation will ultimately lead to architectural level bugs come in to presence in the design which requires huge amount of effort to debug those bugs.

When a bug occurs at an unpredictable place, debugging becomes always a problem during the verification phase. Due to which even the most comprehensive verification techniques can completely miss these bugs which are created by uncertain functional combinations and unclear specification interpretations [3].

C. Verification Methodologies

Verification is a methodology used to illustrate how functionally the design is correct. The main purpose of performing the verification is to guide the design to do its function perfectly for which it is intended to. There are two methodologies which can help in performing

verification of the design before the chip is being taped out to the end customer [4] [5].

First one is the Assertion Based Verification (ABV) Methodology which works on the principle of assertions [6]. Assertion is a check performed against the design specification or any specific rule. Assertion can be used for checking the intent of the design. It also improves the ability to debug and observability. The effect of the bug can be observed early at a line of code before it progresses to the outputs. Assertions can also be used for checking whether the design specification is met in verification phase [7].

Assertion Based Verification (ABV) methodology basically defines how assertions are used, who writes the assertion cases, languages used for writing the assertions, where to write the assertion in the particular code, according to the problem which libraries of assertion to be used, how can we debug the assertion, how to use the assertion methodology for formal verification and how to use the assertions for coverage [8].

Next is the concept of Constrained Random Verification (CRV) Methodology. This concept will allow the user to generate Random key presses, in a Random Sequence. Of course, we are still not testing all possible sequences, but it is giving us far more confidence, as we know that this way of generating test vectors is testing a lot more than the traditional method [9]. If we run ten thousand Random sequences, and could not find a bug, the chances are DUT is indeed bug free. On the other hand, using traditional methods, the test coverage remains significantly low [10] [11].

D. Intellectual Property (IP) Cores

Intellectual Property (IP) core in the VLSI design industry terms is known as a reusable part of the logic or a reusable functionality of the design or a reusable part of a standard cell of a design or simply a layout of the design that is generally implemented with the scope of manufacturing and delivering the design to multiple customers which can be used as a building blocks in different higher level and complex chip designs [12].

In today's world of semiconductor industry, integrated chip designs a large amount of functionality of the design systems are getting consolidated into single chips which is later evolved as a system on chip (SOC) design. Designs which are implemented earlier such as IP cores design or a small logic blocks are the most important part of the system on chip design. This is because most of the system on chip designs do have a pre defined microcontroller or a microprocessor and many of other functionalities of the design system which are uniform and hence if the chip is designed according to the given manufacturer's specification once again, it can be re-used across the several part of the designs. IP cores are commonly accredited as either Soft IP cores or Hard IP cores.

Soft Intellectual Property cores are the blocks which are in general presented as synthesizable RTL models. These design blocks are created in one of the Hardware description language like SystemVerilog or Verilog hardware description languages. Sometimes intellectual property blocks are also synthesized and are presented as standard gate level netlist which can be then mapped to any design level technologies. This also comes under Soft IP cores. The advantage of Soft intellectual property cores is that those can be adapted in the back end design which includes partitioning, floor planning and placement and routing flow by a end user to map to any development technologies. Examles of soft intellectual property core includes dynamic random access memory controller, Ethernet physical address controller, AMBA- AXI, AMBA-AHB bus protocol controller IPs.

Hard Intellectual Property cores on the other hand are presented as layout of the designs in a layout plan like Graphic Data System (GDS) which is mapped to a process technology and can be unswervingly dropped by a customer to the final layout of the chip design. These intellectual property cores cannot be personalized for different process technologies. Phase Locked Loop's, Analog to Digital Converter or Digital to Analog Converter, Physical layer logic for Double Data Rate semiconductor memories, Peripheral Component Interconnect Express are usually developed and accredited as Hard Intellectual Property cores [13].

E. Literature Survey

In paper [14] "Design and Verification of an Efficient Wishbone-Based Network Interface For Network On Chip", a generic asynchronous first in first out based wishbone attuned plug and play network boundary for the design of network on chip is been implemented and verified. It is found that the advantages like minimum latency, high speed data transfer, higher throughput and higher speed has been achieved and through which it concludes that the proposed verification environment performs better in terms of speed, latency and throughput as compared to all other previously used architectures.

In paper [15] "Design and Verification for Data Acquisition Interface Adc_Usb Ip Core", it achieves better data acquisition and efficient data transfer to PC because ADC_USB IP core solves the communication problem by achieving efficient data transfer between PC and data acquisition equipment. The IP can also be applied to other SOC systems.

II. DEVICE UNDER TEST (DUT)

In the verification process, we as a design verification engineer are going to verify the device under test (DUT) by sending the input to the device under test (DUT) to check the behavior of the device under test (DUT). We should be checking the behavior of the device under test (DUT) by sending both the correct and error input stimulus, in both the cases the design verification engineer is bound to observe the device under test (DUT) for its behavior, as the device under test (DUT) is behaving as per the expected design specification, if not then there is surely an existence of a bug.

In the process of verification, we will implement the environment for verification process as well as for the test bench implementation to assign the required input stimuli for the device under test (DUT) and more

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

importantly to observe and monitor the progress of the design under test (DUT) so that the design verification engineer can check how far the device under test (DUT) is giving the user the correct results.

The functionalities of the Test bench and Verification environment is to produce the required stimulus for the device under test (DUT), send this generated input stimuli to the device under test (DUT), the device under test will analyze the input data and will generate the required output according to its design specifications, these outputs are captured from the device under test (DUT), next step is to ensure that there is a match between the actual results from the device under test (DUT) with the expected results which is pre-calculated using some mathematical models and finally measuring the overall progress of the verification process.

In the figure mentioned below describes the Device Under Test (DUT), to which input stimulus are applied and corresponding output is collected and analyzed. The output of the device under test(DUT) is compared with the one reference model.

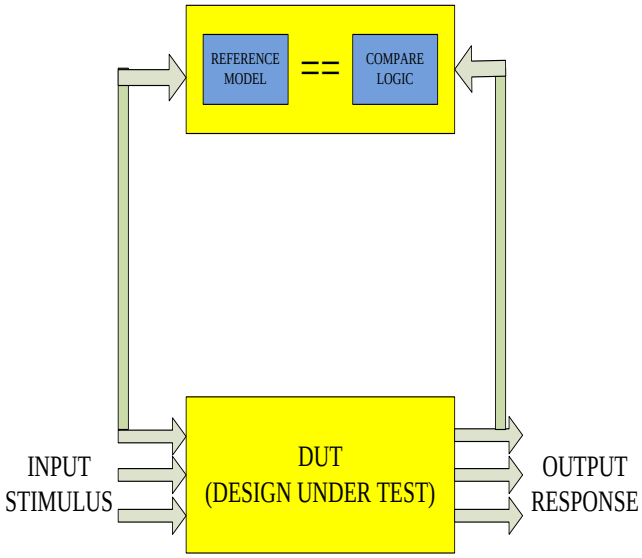

Fig 1. Device Under Test (DUT) Example

If the obtained output is equal to that of the reference model then the test case is passed or we can say that the DUT is verified successfully. But if the output is not equal to that of the reference model then the testcase has failed to verify the DUT. And the bug is still present in the DUT and we have to take proper action to eliminate the bug.

Here Design Under Test (DUT) is a RGB to Gray scale Converter IP. There are two methods to convert a given video from rgb to grayscale. Average method and Weighted or also called as Luminosity method [16].

In average method we take the average of the three colors that is red, green and blue to convert the video into grayscale. The formula to calculate the grayscale output is:

$$\text{Grayscale} = [(R+G+B)/3].$$

We have to convert a rgb video inputs into the output as a grayscale video, but the resultant video was turned out as a fully black video. The problem is aroused due to the reality that the three dissimilar colors have their own diverse wavelengths and each of the color also have their own contributions towards the development of the video or image.

Weighted method used for conversion of rgb video into the grayscale video has a solution to the problem caused by the average method. The red color part of the rgb video has more wavelength than the other two colors that is green and blue. Green color part of the rgb video comprises not only less wavelength as compared to the red color but also green color part will provide more comforting consequence to the eyes. This situation means demand the design verification engineer to decrease the involvement of red color and increase the role of the green color and put blue color part in between these two. So the new grayscale output formula that is obtained is:

Newly obtained grayscale image = ((0.30 x Red) + (0.59 x Green) + (0.11 x Blue)).

According to this newly obtained grayscale image or video shows that red color has contributed thirty percentage of the total, green color has contributed fifty nine percentage of the total and blue has contributed remaining eleven percentage of the total, which is smaller as compared to red and green contribution to the finally obtained output grayscale image or video [17].

Since input that we send to device under test (DUT) is a video file, we will first initialize the number of bits per symbol as 8 and symbols per beat as 3 under the parameter keyword using SystemVerilog language definition. Here only one symbol is considered as black or white color. The size of the input video file is 24 bits, video field width and height are 16 bits each. Width and height defines pixels of the input video file. Device under test consist of two interfaces, decoder, user algorithm block and encoder.

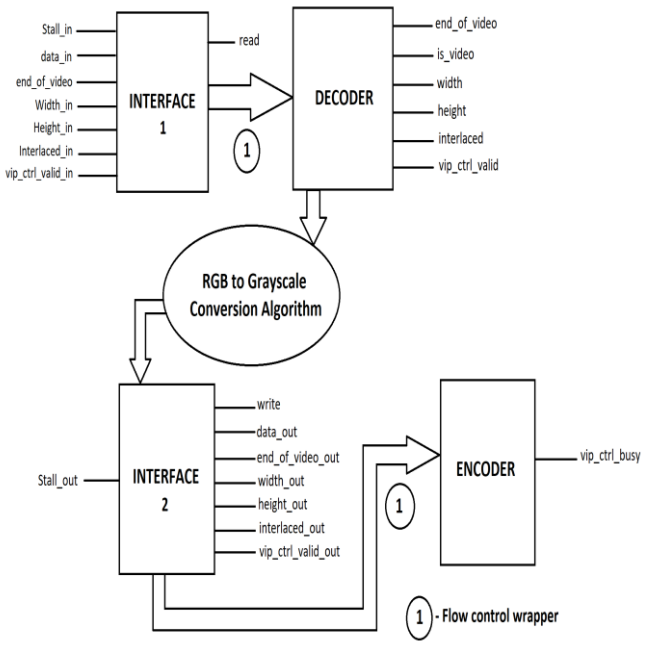

Fig 2. Device under test block diagram.

In the figure above interface 1 reads the input video file along with other signals such as stall_in which signals read event to occur whenever video sequence is available at the input, end_of_video signal to indicate there is no more video sequence at the input to read, vip_ctrl_valid_in signal to avoid collision of the video packets. Decoder decodes these signals from interface 1 and streams the required signals to the user algorithm block.

In user algorithm block, each of the color constants like red factor, green factor and blue factor are allotted size of 8 bits each. Value of red factor is assigned as 76, value of green factor is assigned as 150 and value of blue factor with 29.

Next step is to break down the input video file into the red color component, green color component and blue color component. Here least significant bit (LSBs) are assigned as blue and most significant bit (MSBs) are assigned as red according to the new color convention. And each color components are allotted with the size of 8 bits. Finally the gray output is calculated by multiplying each of the color constants with their corresponding color components.

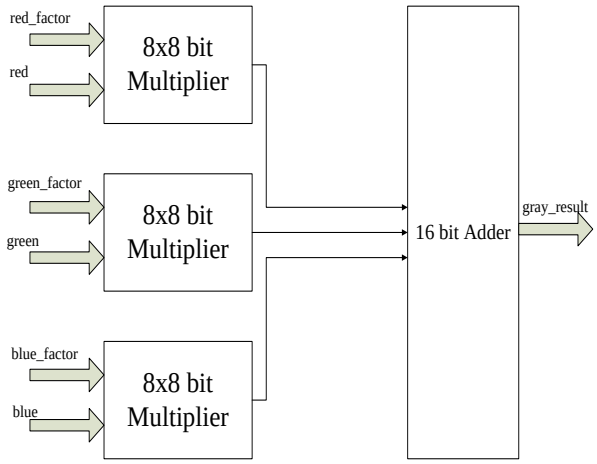

Fig 3. User algorithm block diagram.

The size of the gray output is 16 bits. For multiplication 8 bit multiplier is incorporated and for addition 16 bit adder is used which is shown in above figure. Since the input video file is of size 24 bits, we take the middle value of size 8 bits of the obtained result and concatenate three times to send it as an output video file to the interface 2. Interface 2 along with stall_out signal as an input will write the file.

Other outputs from interface 2 are width and height of the video field, end_of_video_out signal to indicate end of video processing by the DUT. These outputs are send to the encoder which encodes the output video packets and sends it to the sink bus functional model which will eventually sends the video file to the output mailbox where the file is written into the output file. This process happening outside the DUT is explained in the next section.

III. VERIFICATION ENVIRONMENT

A. Introduction

Verification environment and test bench is generally used to check whether the device under test is functionally is correct or not. This is done by generating a predefined input sequences and driving to the device under test. Then the device under test outputs are captured, analyzed and finally compared with the expected outputs [18].

Verification environment is formed by grouping the several components necessary for the specific task or operations. In verification environment, separate classes are written for each specific operation [19]. This includes classes for generating stimulus, driving the input files through interfaces and buses and for monitoring the flow of the operation etc. and these classes will be named on the operation [20].

The below mentioned figure indicates the components present in the avalon-st video verification Environment. Block colored with yellow indicates elements of class library present for the test bench environment, block colored with green indicates the bus functional model (BFM) of the avalon-st interface and block colored with blue indicates the device under test (DUT) and the test method points to represent themselves in the environment which is done with the help of exclusive features of SystemVerilog hardware verification language. Input video packets and input control packets are sent to the device under test (DUT). The output reactions are collected and then analyzed from the device under test (DUT) and the obtained final output video files are written to an output file [21] [22]. Even though the verification environment is implemented by creating the class libraries required for the device under test (DUT) present in the environment, other test bench environments can be predictable to this same test structure, with respect to the Veri -log hardware description language module level accessibility and object or class level accessibility.

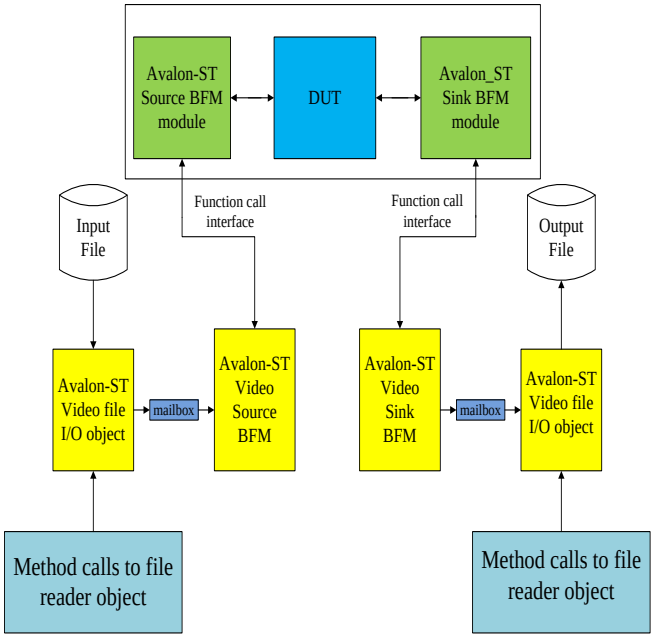

Fig 4. Verification Environment.

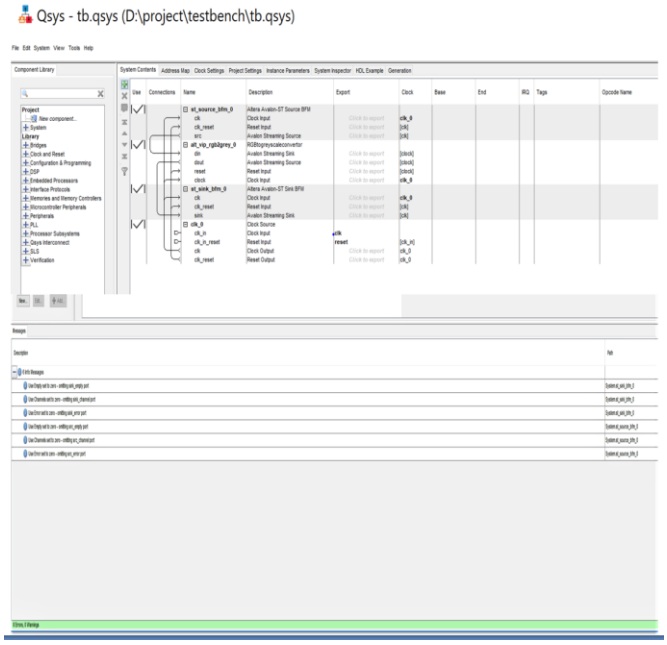

Fig 5. Verification Environment implementation on Qsys tool.

The class libraries called in this verification environment make use of the source and sink bus functional model (BFM) and provide the functionalities such as it encompasses the avalon-st video regular observance testing, creates a certain group of modus operandi failures that the device under test (DUT) can be tested against which can be configured by using simple method calls to the class libraries with the help of the test bench, it provides facility to read video file and to write the output file in order to facilitate device under test (DUT) testing with authentic time input video sequences. The library code is easily understandable to new design verification engineers and has all the features of a better object oriented based design code and hence it can be effortlessly extended for further requirements in the design in future and it uses SystemVerilog's commanding verification functionalities which include mailboxes and randomize methods which allows the design verification engineer to design and implement complex bus environments required for rigorously verifying the device under test (DUT).

B. Implementation

Verification environment is created using Altera Quartus Prime 7.2 tool in which by copying the verification files to the local directory and by invoking Qsys system integration tool [23] we will generate the netlist files and other required classes file using System Verilog language. Once you click the generate icon, the system refreshes and shows the RGB to grayscale convertor in between source and sink bus functional models (BFMs), this is our device under test (DUT).

We have to exit the generation completed information message pop up icon, and close the Qsys tool. Qsys tool now will generate the netlist file required for the verification and other required simulation files for assisting further verification process. The implementation on Qsys tool can be in the figure below. After implementing the verification environment, we will perform two verification process to verify our device under test using ModelSim [24]. One is file reader test for video verification and another one is constrained random verification [25].

IV. RESULTS AND DISCUSSIONS

First test which is performed here is the video file reader test. It is the simplest test which is performed with the help of libraries of different classes. The file reader test for video verification, reads the input video file and then translates into the video objects. The test then sends the video objects in to the device under test (DUT) using the bus functional models (BFM). And then it retranslates back the video file to the video output and writes the video output file again.

The test bench for video verification has four main sections of code inside it. First, the test to verify the video will precisely give information about the numbers of bits per channel and channels per pixel used in defining the input video file, as most of the classes defined for this project will be requiring this information. Next step is to import class packages, define the clock and reset necessary for the device under test to work and finally the netlist which is instantiated with connections necessary for clock and reset in place. The resets defined for the bus functional models are all initially set to active high. The last part of the code creates some of the objects that are later used in the code. The parameter is standard across all object instances of the classes as per the initialization of the bits per channel and channels per pixel which are constant in any given design system.Figure 6 and 7 represents the output that specifies the communication establishment happening between Source and Sink Bus Functional Module (BFM) while running the Video File Reader Test.

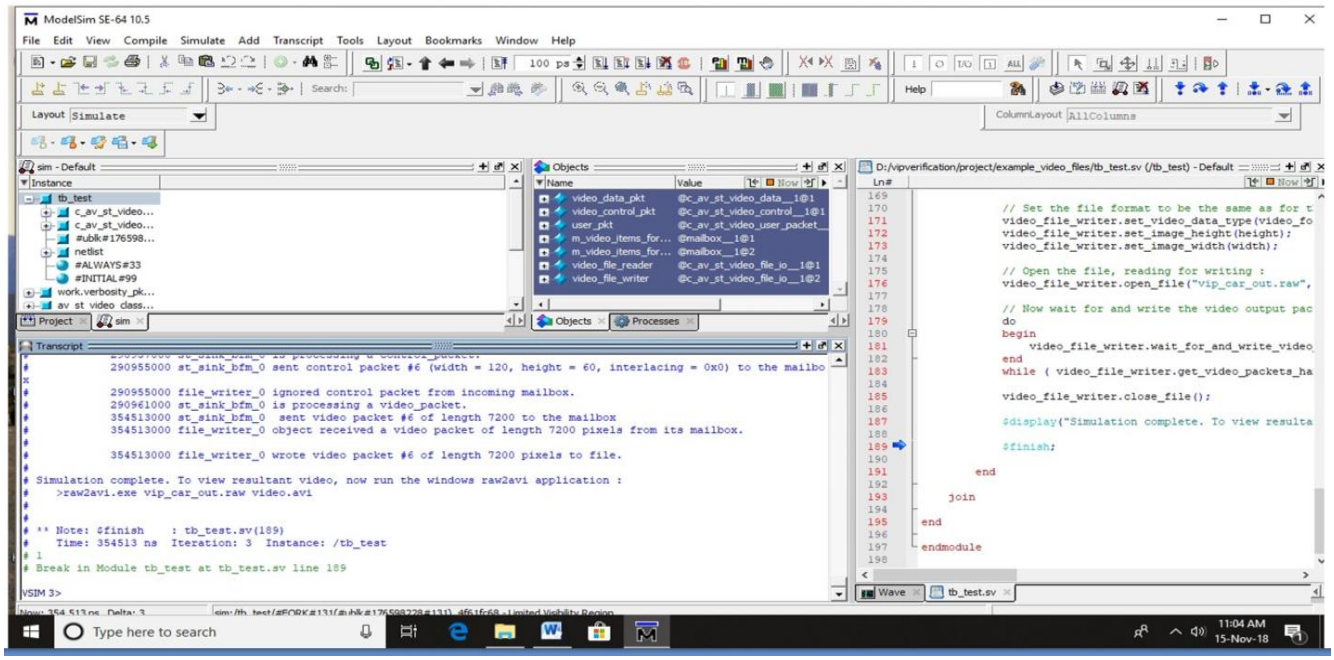

Fig 6. ModelSim output of Video File Reader Test.

```
#          0: INFO: tb_test.netlist.st_sink_bfm_0.__hello: - Hello from altera_avalon_st_sink_bfm.
#          0: INFO: tb_test.netlist.st_sink_bfm_0.__hello: -  $Revision: #1 $
#          0: INFO: tb_test.netlist.st_sink_bfm_0.__hello: -  $Date: 2011/08/15 $
#          0: INFO: tb_test.netlist.st_sink_bfm_0.__hello: -  ST_SYMBOL_W      = 8
#          0: INFO: tb_test.netlist.st_sink_bfm_0.__hello: -  ST_NUMSYMBOLS          = 3
#          0: INFO: tb_test.netlist.st_sink_bfm_0.__hello: -  ST_CHANNEL_W     = 1
#          0: INFO: tb_test.netlist.st_sink_bfm_0.__hello: -  ST_ERROR_W       = 1
#          0: INFO: tb_test.netlist.st_sink_bfm_0.__hello: -  ST_EMPTY_W       = 2
#          0: INFO: tb_test.netlist.st_sink_bfm_0.__hello: -  ST_READY_LATENCY = 1
#          0: INFO: tb_test.netlist.st_sink_bfm_0.__hello: -  ST_MAX_CHANNELS  = 0
#          0: INFO: tb_test.netlist.st_sink_bfm_0.__hello: -  ST_BEATSPERCYCLE = 1
#          0: INFO: tb_test.netlist.st_sink_bfm_0.__hello: -  USE_PACKET       = 1
#          0: INFO: tb_test.netlist.st_sink_bfm_0.__hello: -  USE_CHANNEL      = 0
#          0: INFO: tb_test.netlist.st_sink_bfm_0.__hello: -  USE_ERROR        = 0
#          0: INFO: tb_test.netlist.st_sink_bfm_0.__hello: -  USE_READY        = 1
#          0: INFO: tb_test.netlist.st_sink_bfm_0.__hello: -  USE_VALID        = 1
#          0: INFO: tb_test.netlist.st_sink_bfm_0.__hello: -  USE_EMPTY        = 0
```

```
#          0: INFO: tb_test.netlist.st_source_bfm_0.__hello: - Hello from altera_avalon_st_source_bfm.
#          0: INFO: tb_test.netlist.st_source_bfm_0.__hello: -  $Revision: #3 $
#          0: INFO: tb_test.netlist.st_source_bfm_0.__hello: -  $Date: 2011/08/21 $
#          0: INFO: tb_test.netlist.st_source_bfm_0.__hello: -  ST_SYMBOL_W      = 8
#          0: INFO: tb_test.netlist.st_source_bfm_0.__hello: -  ST_NUMSYMBOLS          = 3
#          0: INFO: tb_test.netlist.st_source_bfm_0.__hello: -  ST_CHANNEL_W     = 1
#          0: INFO: tb_test.netlist.st_source_bfm_0.__hello: -  ST_ERROR_W       = 1
#          0: INFO: tb_test.netlist.st_source_bfm_0.__hello: -  ST_EMPTY_W       = 2
#          0: INFO: tb_test.netlist.st_source_bfm_0.__hello: -  ST_READY_LATENCY = 1
#          0: INFO: tb_test.netlist.st_source_bfm_0.__hello: -  ST_MAX_CHANNELS  = 0
#          0: INFO: tb_test.netlist.st_source_bfm_0.__hello: -  ST_BEATSPERCYCLE = 1
#          0: INFO: tb_test.netlist.st_source_bfm_0.__hello: -  USE_PACKET       = 1
#          0: INFO: tb_test.netlist.st_source_bfm_0.__hello: -  USE_CHANNEL      = 0
#          0: INFO: tb_test.netlist.st_source_bfm_0.__hello: -  USE_ERROR        = 0
#          0: INFO: tb_test.netlist.st_source_bfm_0.__hello: -  USE_READY        = 1
#          0: INFO: tb_test.netlist.st_source_bfm_0.__hello: -  USE_VALID        = 1
#          0: INFO: tb_test.netlist.st_source_bfm_0.__hello: -  USE_EMPTY        = 0
```

Fig 7. Communication between Source and Sink BFM.

Input video file. Output video file.

Fig 8. Output of Video File Reader Test.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

The file reader test done for video verification is valuable for examining the video functionality of the device under test (DUT) for any types of input video file. Nevertheless, this test is not appropriate to experiment the device under test (DUT) with a range of differently sized and formatted video input fields.The constrained random test is also effortlessly accumulated using the libraries defined for the classes. In constrained random verification, the randomized input video packets, input control packets and input user packets are produced using the SystemVerilog's built-in features of constrained random verification. Then the randomly generated packets are sent to device under test through the source bus functional model (BFM) as well as the packets are also sent directly to the scoreboard through one of the mailbox. The DUT processes the video packets and send it to the mailbox through the sink bus functional model (BFM). The scoreboard retrieves the packets from both the mailboxes and determines a test pass or fails result.

This test bench code launches the source and sink bus functional model (BFM), then randomly produces either a video data packet, control data packet or a user data packet needed for the device under test (DUT). This generation of these input packets are achieved by basically invoking the randomize method on the objects that are created previously at the end of this test bench code, next step involves putting the objects that are created according to type of the input packets in the source bus functional model's mailbox, a copy of this objects are made and is streamed to a reference mailbox at the end used by the scoreboard.

Ultimately, the test bench code indicates to the scoreboard at the end that a video data has been sent and the scoreboard waits for the output from the device under test to be investigated. It is also hinted by an event from the scoreboard. Now the only part that is left out is to develop a scoreboard that will recover the video data objects from the two scoreboard mailboxes and will compare the outputs received from the device under test (DUT) with the reference outputs.

In figure 9 the scoreboard is expected to perceive the device under test (DUT) giving the output as a grayscale video output data. You must model the data to emulate the behavior of individual device under tests accurately. The scoreboard successfully matches the randomly generated video, control and user packets to that of the packets obtained from the device under test (DUT).

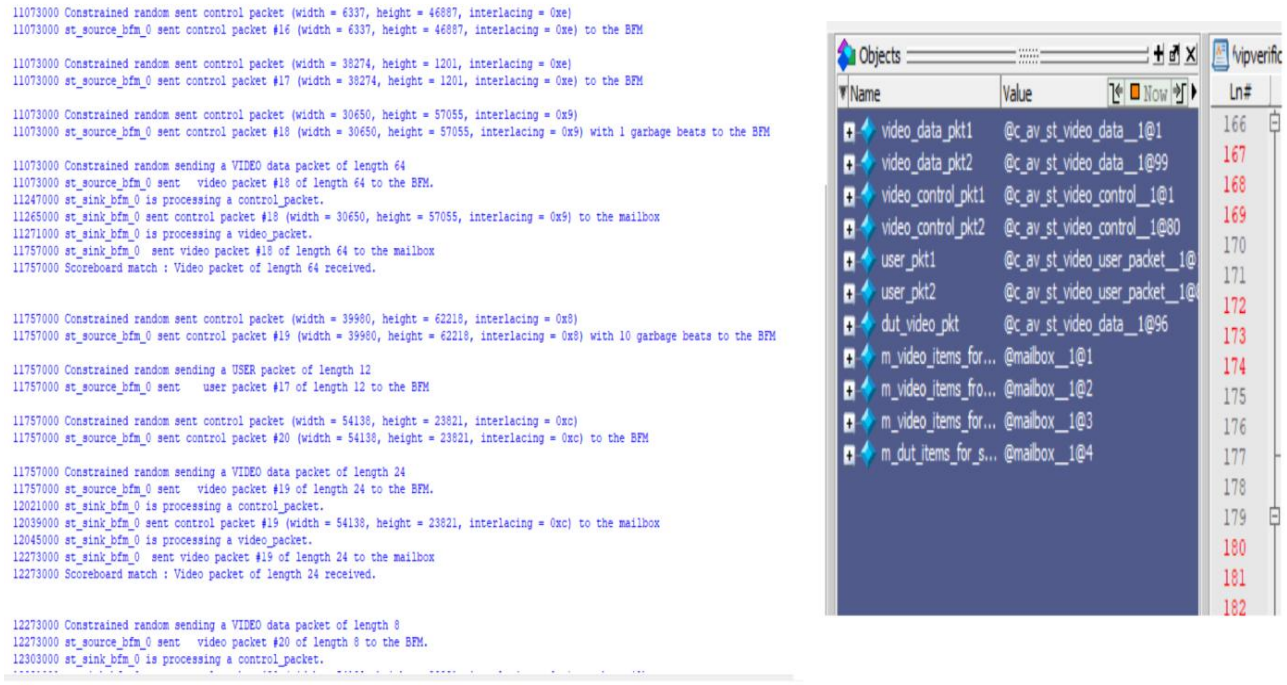

Fig 9. Output of Constrained Random Test.

V. CONCLUSION

In this paper RGB to Grayscale Converter IP is implemented and verified by performing two tests. One is file reader test for video verification and another one is constrained random verification. In file reader test for video verification, the functionality of the device under test (DUT) which means whether it is properly converting the input video into a grayscale video or not. In Constrained Random verification, the test will generate random video, user and control packets and send it directly to the scoreboard and also to the device under test. Scoreboard determines whether the randomly

generated values are same as that to the values generated by the device under test. Future work would be obtaining code coverage by testing the device under test with required test cases.

REFERENCS

1. Accellera Home: System Verilog 3.1a LRM: (2004)
2. Amir Hekmatpour, Alley, C.; Stempel, B.; Coulter, J.; Salehi, A.; Shafie, A.; Palenchar, C.,(2005) **"**A HETEROGENEOUS FUNCTIONAL VERIFICATION PLATFORM**"**- Custom Integrated Circuits Conference, Proceedings of the IEEE 2005 Volume , Issue , 18-21 Sept. 2005.
3. Aniruddha Baljekar, NXP Semiconductors India Pvt. Ltd., Bangalore, INDIA, (2006), "RE-USE OF VERIFICATION ENVIRONMENT FOR VERIFICATION OF MEMORY CONTROLLER".
4. Jayanta Bhadra, Magdy S. Abadir, Li-C. Wang, Sandip Ray, (2007) "A SURVEY OF HYBRID TECHNIQUES FOR FUNCTIONAL VERIFICATION," IEEE Design and Test of Computers, vol. 24, no. 2, pp. 112-122, June 2007
5. St. Pierre, M. Yang, S.-W. Cassiday, D., Thinking Machines Corp., Cambridge, MA, (2002) "FUNCTIONAL VLSI DESIGN VERIFICATION METHODOLOGY FOR THE CM-5MASSIVELY PARALLEL SUPERCOMPUTER"- Computer Design: VLSI in Computers and Processors, 1992. ICCD '92. Proceedings., IEEE 1992 International Conference.
6. Eduard Cerny. Synopsys, Inc. Marlborough, USA , Dmitry Korchemny Intel *Corp* , in (2007) "USING SYSTEMVERILOG ASSERTIONS FOR CREATING PROPERTY-BASED CHECKERS" www.eda.org/ovl/pages/pdfs/dvcon07_cerny.pdf.
7. Hans Eveking, Braun, M. Schickel, M. Schweikert, M. Nimbler, V., Comput. Syst. Group, Darmstadt Univ. of Technol., Darmstadt, (2007) "MULTI- LEVEL ASSERTION-BASED DESIGN FORMAL METHODS AND MODELS FOR CODESIGN", 5th IEEE/ACM International Conference.
8. Kuang-Chien Chen, (2003) **"**ASSERTION-BASED VERIFICATION FOR SOC DESIGNS"- ASIC, 2003. Proceedings. 5th International Conference on, Volume 1, Issue , 21-24 Oct. Page(s): 12 - 15 Vol.1.
9. Charles H. P. Wen, Li-C. Wang, Kwang-Ting Cheng, (2006) "SIMULATION-BASED FUNCTIONAL TEST GENERATION FOR EMBEDDED PROCESSORS"- IEEE Transactions on Computers, vol. 55, no. 11, pp. 1335-1343.
10. F. Corno, G. Cumani, M. Sonza Reorda, G. Squillero, (2003) "FULLY AUTOMATIC TEST PROGRAM GENERATION FOR MICROPROCESSOR CORES," date, vol. 1, pp.11006, Design, Automation and Test in Europe Conference and Exhibition.
11. Eugene Zhang, E. Yogev, E. Cisco Syst. Inc., USA, (1997) "FUNCTIONAL VERIFICATION WITH COMPLETELY SELF-CHECKING TESTS"- Verilog HDL Conference, IEEE International**.**
12. Anjali Vishwanath, Ranga Kadambi Infineon Technologies Asia Pacific Pte Ltd (2007) "VERIFICATION PLANNING FOR THE CORE BASED DESIGNS" .
13. K.Aditya, M.Sivakumar, Fazal Noorbasha, T.Praveen Blessington, (2012) "DESIGN AND FUNCTIONAL VERIFICATION OF A SPI MASTER SLAVE CORE USING SYSTEMVERILOG", International Journal of soft computing and engineering (IJSCE), ISSN: 2231-2307, Volume-2, Issue-2.
14. K.Swaminathan,G.Lakshminarayanan, Seok-Bum Ko, (2014) DESIGN AND VERIFICATION OF AN EFFICIENT WISHBONE-BASED NETWORK INTERFACE FOR NETWORK ON CHIP, Computers and Electrical Engineering, Elsevier.
15. Golla Mahesh, SM Sakthivel, (2015) "Functional verification of the Axi2OCP bridge using system verilog and effective bus utilization calculation for AMBA AXI 3.0 protocol", International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS) 1-5.
16. Tarun kumar, Karun verma, (2010) "A THEORY BASED ON CONVERSION OF RGB IMAGE TO GRAY IMAGE", International Journal of Computer Applications (0975-8887), Volume-7.
17. Fradi Marwa, Elhadjyoussef Wajih, Mohsen Machhout, (2017) "THE DESIGN OF AN EMBEDDED SYSTEM (SOPC) FOR AN IMAGE PROCESSING APPLICATION", International Conference on Advanced systems and electric technologies (IC_ASET).
18. Han Ke; Deng Zhongliang; Shu Qiong, (2007) **"**VERIFICATION OF AMBA BUS MODEL USING SYSTEMVERILOG"- Electronic Measurement and Instruments, 2007. ICEMI apos;07. 8th International Conference on, Volume , Issue , Aug. 16 2007-July 18 2007 Page(s):1-776 - 1-780.
19. Ivan Petkov, Amblard, P. Hristov, M. Jerraya, A. TIMA Lab., IMAG, Grenoble, France, (2005) "SYSTEMATIC DESIGN FLOW FOR FAST HARDWARE/SOFTWARE PROTOTYPE GENERATION FROM BUS FUNCTIONAL MODEL FOR MPSOC"- Rapid System Prototyping, (RSP 2005). The 16th IEEE International Workshop on, Publication Date: 8-10 June 2005, On page(s): 218- 224.
20. Min-An Song; Ting-Chun Huang; Sy-Yen Kuo, (2006) "A FUNCTIONAL VERIFICATION ENVIRONMENT FOR ADVANCED SWITCHING ARCHITECTURE ELECTRONIC DESIGN, TEST AND APPLICATIONS", DELTA 2006. Third IEEE International Workshop on Volume , Issue , 17-19
21. Nick Heaton and Ed Flaherty, CommsDesign, (2001) "A VERIFICATION ENVIRONMENT FOR AN EMBEDDED PROCESSOR": www.commsdesign.com/.
22. Sakthivel.S.M. Murali .M, Umadevi. S, (2017) "Verification IP for AMBA AXI Protocol using System Verilog", International Journal of Applied Engineering Research (IJAER) 12(17) 6534-6541
23. Akshay S.Bharadwaj, Deepak Mankkadan (2012) "IMPLEMENTATION OF AN ETHERNET BRIDGE USING AVALON MEMORY-MAPPED INTERFACE", Emerging Research in Electronics, Computer Science and Technology: Proceedings of International Conference, ICERECT published by Springer India.
24. Shweta Sharma, SM Sakthivel (2018) "Design and Verification of AMBA AXI3 Protocol", VLSI Design: Circuits, Systems and Applications, Springer, Singapore. Springer Lecture notes in Electrical Engineering 247-259
25. https://www.intel.com/content/dam/alterawww/global/en_US/pdfs.

AUTHOR PROFILE

Shyamsundar Iyer, pursuing M.tech VLSI Design from VIT University Chennai. Interested in front end design especially in design verification phase.