IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)
Volume 3, Issue 6 (Nov. – Dec. 2013), PP 44-50
e-ISSN: 2319 – 4200, p-ISSN No. : 2319 – 4197
www.iosrjournals.org

Design & Assertion Based Verification of Avalon Interrupt Interface

Arshiya Sultana[1], Sowmya L[2]
[1](Department of Electronics & Communication Engineering, M V J College of Engineering, India)
[2](Department of Telecommunication Engineering, M V J College of Engineering, India)

Abstract : *The complexity of ASIC design has increased at an enormous rate in the last few years. This gives rise to difficulties in verification as well. Assertion Based Verification (ABV) provides a solution to this problem. Assertions are used to capture specifications and design intent in an executable form. They help in detecting the bugs faster and closer to the source. In addition to this ABV has a number of advantages such as, detection of corner cases, help in documentation and provide improved design reuse. With the introduction of many standard languages to capture assertions the ABV methodology has gained a lot of popularity. In addition to this many of the simulation tools available today also provide assertion coverage. In this project Avalon Interrupt Interface was designed and verified using assertions. The project carried out involved six stages: RTL design, linting, behavioral simulation, synthesis, Gate Level Simulation (GLS) and adding of assertions. In this project, an RTL design for Avalon Interrupt Interface was developed in VHDL. Testbench was written in Verliog. The design is verified using Accellera PSL assertions along with the test bench. The results were simulated using Aldec Riviera-PRO 2011.02 simulation tool.*

Keywords : *Assertions, individual request, interrupt, priority based, PSL.*

I. INTRODUCTION

The recent developments in the field of VLSI have led to increase in complexity of ASIC design. With this increase in complexity of design, the verification engineers are also faced with problems of finding new ways to verify a design. The purpose of verification is to confirm that the design complies with the specification. Verification forms a very important aspect of ASIC design and it is estimated that over 70 % of the design cycle for any new device is spent in the verification process. Moreover, a design containing bugs could cause severe delays in time to market, because major part of the design has to be redone. The old day techniques of verification like manual verification, test bench verification that treat the design as a black box are time consuming and hence not suitable for present day complex designs. While designer productivity underwent a breakthrough gate-per-hour increase with the adoption of synthesis, a comparable breakthrough did not happened in verification productivity.

Fig.1 shows the gap between design and verification. Verification effort, including development cost and time, required engineering resources and the probability of undetected bugs, all increase non-linearly as design complexity increases. So the verification engineers are faced with the challenge of finding new verification techniques. ABV methodology is a revolution in the field of verification.

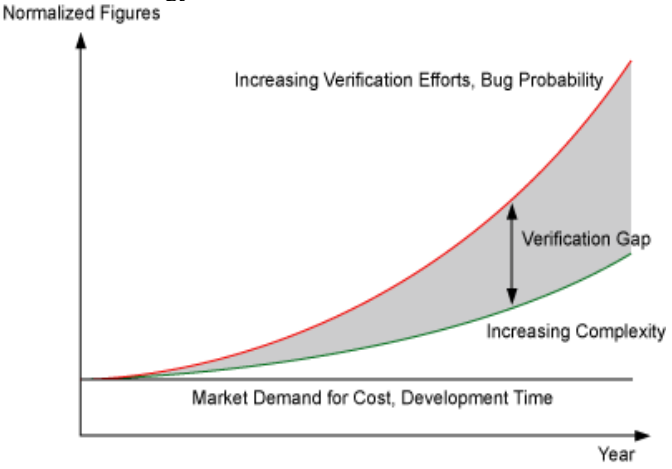

Fig 1 Design – Verification gap

ABV can be defined as the use of assertions along with design to provide added visibility to its internal state and allow white box testing [1]. It is a combination of simulation and formal verification. An assertion captures

expected behavior when a particular input is applied to the design [2]. Assertions can be used to verify designs that use complex functionalities and also employ reusable design components [2].

The ABV methodology has a number of advantages as listed below:

Detection of bugs closer to source: It enables detection of bugs closer to the source hence saving debug time. This reduces the time to market.

Better than test bench alone: This is because assertions are part of the source code and describes the internal functionality of the module they create a faster and better way to check the system. Implementing assertions with a typical test bench creates improved system-level results by improving the overall coverage with less effort.

Documentation: In addition to finding bugs, assertions encapsulated in an RTL provide an excellent form of documentation. Other engineers can review the assertions to understand the lower level specifications of how to interface with another block. Assertions formally document protocols, interfaces and assumptions in an unambiguous form that clarifies a designer's interpretation of the specification and design intent.

Detect corner cases: ABV enables detection of corner cases. An example of corner case is that a FIFO is being written when it is already full. This would result in an error, which can be easily detected with the help of an appropriate assertion.

Improved Design Re-Use: Assertions are quite simply assumptions about how a specific block should operate, both by itself and in relation to the surrounding design blocks. Because of this, subsequent re-use of the specific modules becomes much easier.

Assertions act as monitors or checkers spread throughout the design thus help in detection of more bugs and provide maximum coverage. The assertions are executed and their success or failure can be saved in a log for review. Today many industry standard languages are available to write assertions. In this paper we present a design for Avalon Interrupt Interface in VHDL and the capture assertions using PSL.

II. PROPERTY SPECIFICATION LANGUAGE

The design specifications are generally represented in English like language and are not machine executable. The Accelera PSL [3] was introduced to overcome this shortcoming. PSL is based on the Sugar language developed by IBM. PSL was designed to be used along with test bench simulation. PSL allows assertions to be written as embedded comments in the design or as a separate verification unit (vunit). The vunit can be compiled and bound to the main design file.

1.1 Flavors of PSL

PSL is defined for four different flavors which cut the language across the axis of HDL compatibility: one for each hardware description languages SystemVerliog, Verilog, VHDL and GDL. The syntax of each flavor conforms to the syntax of the corresponding HDL in a number of specific areas. A given flavor of PSL is compatible with the corresponding HDL's syntax in those areas. In this project VHDL flavor was used. In each of the flavors, all expressions of the Boolean layer, as well as modeling layer code, are written in the respective HDL syntax.

1.2 Layers in PSL

The PSL language is formally structured into four distinct layers: Boolean layer, Temporal layer, Verification layer and Modeling layer [3]. These layers cut the language across the axis of functionality. Fig. 2 shows an example highlighting the different layers in PSL.

Boolean Layer: This layer is used to build expressions that represent design behavior and are used by other layers. It is the supplier of Boolean expressions to the heart of the language, which is the temporal layer.

Temporal Layer: This layer is the heart of the language. It is used to define properties, which describe behavior over time.

Verification Layer: The verification layer provides directives to the verification tool. The directives are used tell what to do with the properties described by other layers.

Modeling Layer: The modeling layer is used to define auxiliary code that is not part of design but needed for verification.

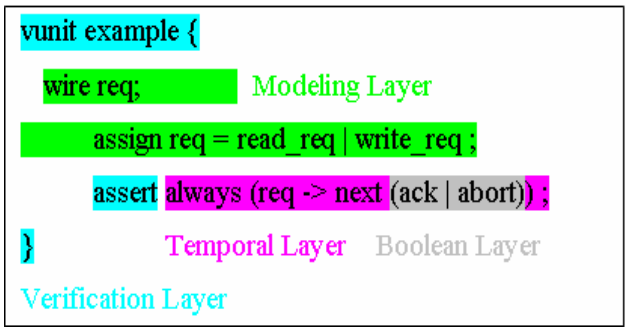

Fig. 2 Layers in PSL

III. AVALON INTERFACE

The Avalon Interface family developed by Altera was defined to allow components in an FPGA to be easily connected [4]. The interface family consists of six different interface types:

- Avalon Memory Mapped Interface (Avalon-MM) – used as an address-based read/write interface.
- Avalon Streaming Interface (Avalon-ST) – used for unidirectional flow of data.
- Avalon Memory Mapped Tristate Interface – used to support off-chip peripherals.
- Avalon Clock – used to provide clock and reset signals to components.
- Avalon Interrupt – used by components to signal events to other components.
- Avalon Conduit – used to export a set of signals to top level where they can be connected to other components.

The Avalon standard interfaces are designed into the components available in the SOPC Builder [5]. SOPC Builder is software made by Altera that automates connecting soft-hardware components to create a complete computer system that runs on any of Altera's various FPGA chips.

1.3 Avalon Interrupt Interface

Avalon Interrupt Interface is used in master-slave type connections to allow the slave (sender) to signal an event to the master (receiver). The master then sends an acknowledgment for the request appropriately. The interrupt request signal of the sender remains high until the acknowledgment is sent. Avalon Interrupt Interface supports two types of interrupt schemes:

1. Individual Requests – It is also known as software priority [5] and supports up to 32 senders. In this scheme the receiver expects a separate interrupt request (irq) bit from each sender. The relative priority is not defined and depends on the implementation [4].
2. Priority Encoded – It is also known as hardware priority [5] and supports up to 64 senders. In this scheme the receiver expects a single interrupt request bit and a six bit interrupt request number (irqnum) corresponding to the highest priority sender [4].

IV. PROJECT OVERVIEW

The project carried out involved six stages: RTL design, linting, behavioral simulation, synthesis, Gate Level Simulation (GLS) and adding of assertions as shown in Fig 3.

TABLES I and II list the tools and languages used at different stages of the project.

TABLE I
TOOLS USED

Process	Tools Used
Design entry	gVim editor
Simulation and ABV	Aldec Riviera-Pro 2011.02
Linting	Aldec ALINT 2010.06
Synthesis	Xilinx ISE 11.1

TABLE II
LANGUAGES USED

Process	Language used	IEEE Standard
RTL Design	VHDL	IEEE 1076
Testbench	Verilog	IEEE 1364
Assertions	PSL	IEEE 1850

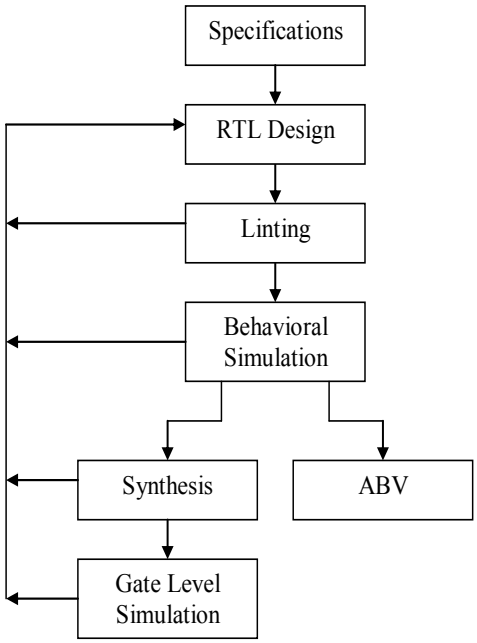

Fig 3 Project flow

4.1 RTL Design

In integrated circuit design, Register Transfer Level (RTL) is a level of abstraction used in describing the operation of a synchronous digital circuit. Register transfer level abstraction is used in HDLs like Verilog and VHDL to create high-level representations of a circuit, from which lower-level representations and ultimately actual wiring can be derived.

An RTL design in VHDL was developed. The proposed design of Avalon Interrupt Interface consists of two main components namely sender and receiver. There can more than one sender but only a single receiver component is allowed. The priority generator component is used only for priority encoded scheme. As already stated the receiver supports two interrupt schemes namely: Individual Requests and Priority Encoded. The irqscheme signal is used to distinguish between the two interrupt schemes. The function of Avalon Interrupt Interface with a single sender and receiver can be summarized as shown in Fig. 4.

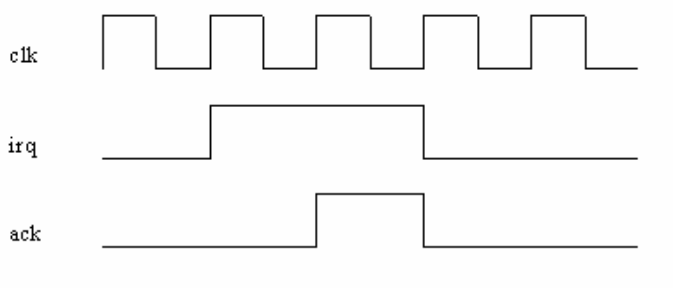

Fig. 4 Interrupt and acknowledge

Individual requests: In this case irqscheme signal is set to low value. The receiver receives 32-bit irq signal corresponding to the 32 distinct senders. The priority assignment used is as follows: sender corresponding to Least Significant Bit (LSB) of irq signal i.e., irq0 is assigned the highest priority and sender corresponding to Most Significant Bit (MSB) i.e., irq31 is assigned the lowest priority. As an example if sender0, sender1 and sender2 send irq signals simultaneously then ack0 is asserted first followed by ack1 and then ack2.

The design consists of 32 senders labeled as sender0 to sender31 and a single receiver as shown in Fig. 5. The clk signal is used as a positive edge triggered clock for the design. The interrupt (intr) signal is used to trigger the irq signal.

As an example a single pulse on intr0 triggers irq0 which is synchronous with the clock (clk) signal. In response to this request the receiver generates an acknowledgment (ack0) in the next cycle itself. The ack signal is high for one cycle and low for one cycle before the next acknowledgement is sent. The sender's functioning can be reset using rst signal, which is also synchronous with positive edge of the clock. When rst signal is set high all irq's corresponding to the 32 senders are reset to low irrespective of the intr signals. The receiver consists of interrupt enable (isr_enable) signal which is active low. When isr_enable is set high the receiver does

not respond to any request i.e., all ack signal are reset while irq signals remain unchanged. When isr_enable goes low again the requests are acknowledged. The rst and isr_enable signals used here are synchronous.

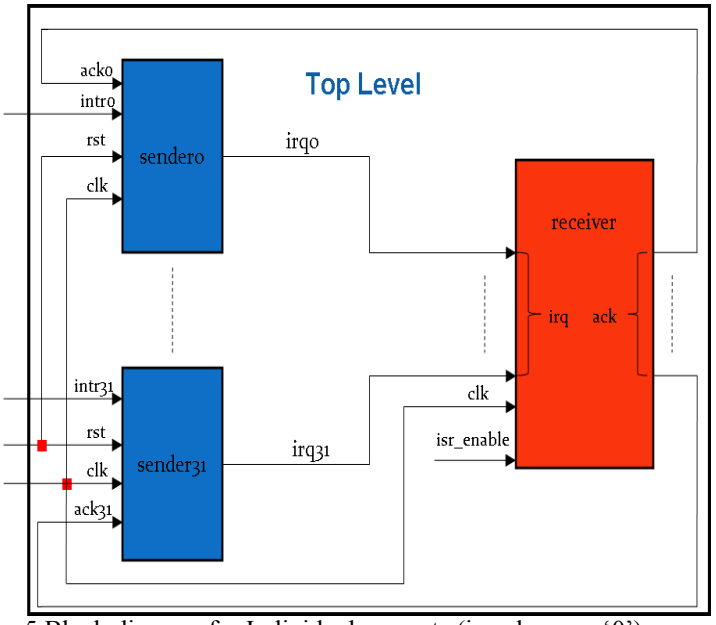

Fig. 5 Block diagram for Individual requests (irqscheme = '0')

Priority Encoded: In this case irqscheme signal is set to high value. The receiver gets a single bit irq signal which is the logical OR of all 64 sender irq signals.

The block labeled prioritygen in Fig. 6 is the priority generator block. Prioritygen performs two important functions:logical ORing of the 64 irq signals and generation of the six bit irqnum to indicate the highest priority sender currently being acknowledged. This block adds one cycle delay to the design. So the minimum cycle delay between an irq and ack is one cycle for priority encoded whereas for individual requests it is zero as explained earlier.

The irqscheme signal is used as an enable signal to prioritygen, thus enabling the block to function only for the case of priority encoded. Here each of the senders was assigned a fixed priority. The ack signal is high for one cycle and low for two cycles to enable prioritygen to calculate irq and irqnum. An additional valid signal is present, which is used by the priority generator to indicate that the value of irqnum calculated is correct. The clk, rst, intr, irq, isr_enable and ack signals have the same function as in individual requests.

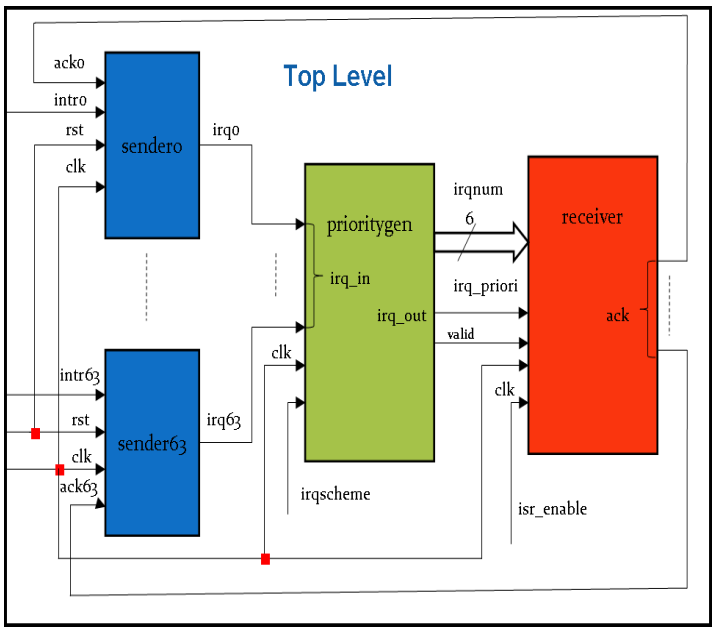

Fig. 6 Block diagram for Priority encoded (irqscheme = '1').

As a last step both cases for irqscheme = '1' and '0' were combined to obtain the final design.

4.2 Linting

Linting also known as Design Rule Checking is used to check the HDL source code against a set of design rules. It helps to detect suspicious use of programming language or identify use of poor coding style. Linting tools detect errors which are normally not captured by the compiler, an example is the use of a condition which is always constant. Some of the rules checked by Linting are:

- Design rules
- Synthesizable description rules
- Reusability rules
- Synchronous design rules
- Initial reset rules
- Clocks and reset description rules
- Combinational logic description rules
- Sequential description rules
- Clock Domain Crossing (CDC) rules
- Design For Testability (DFT) rules

Today, many linting tools are available in the market. In our project the Aldec ALINT 2010.06 [6] was used. It supports IEEE VHDL, Verilog and mixed language designs. Linting carried on the design was successful.

4.3 Behavioral Simulation

Behavioral simulation is performed using a pre-synthesis Hardware Description Language (HDL) description of the design. Behavioral simulation allows you to verify syntax and functionality without timing information. During design development, most verification is accomplished through behavioral simulation. Errors identified early in the design cycle are inexpensive to fix compared to functional errors identified during silicon debug.

A testbench is built to functionally verify the design by providing meaningful scenarios to check that given certain input, the design performs to specification. Writing a testbench is as complex as writing the RTL code itself. For writing testbenches it is important to have the specifications of Design Under Test (DUT). Specifications need to be understood clearly and a test plan, which basically consists of the test cases, needs to be made. In a simple testbench the DUT is instantiated and stimulus is applied as shown in Fig. 7

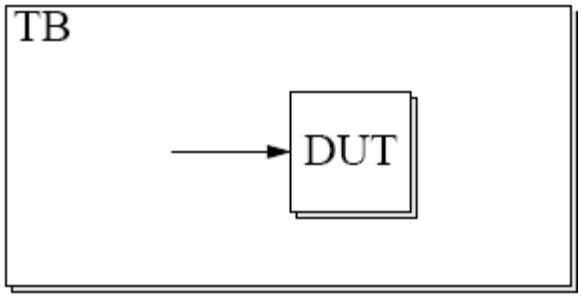

Fig. 7 DUT instantiation in a testbench

In our case a test bench written in Verilog was used to perform behavioral simulation.

4.4 Synthesis and Gate Level Simulation

In electronics, logic synthesis is a process by which an abstract form of desired circuit behavior, typically RTL, is turned into a design implementation in terms of logic gates. Logic synthesis is one aspect of electronic design automation. Synthesis is a mostly automated process using a synthesizer tool that converts your RTL-level design source code into a single gate-level file in a HDL called netlist, which can be fed into a place-and-route tool to create a programming file for your FPGA.

When writing RTL code it must be noted that we are designing hardware and not simply writing a computer program. So, poorly written code may end in poorly designed hardware. Even if the RTL code gives proper results when simulated, it may not give the same results after synthesis. In order for the code to be synthesized properly certain conventions must be followed. For example a VHDL synthesizer cannot synthesis time delays. So, expressions of the form "after time expression" will be ignored or have to be removed.The tool used for synthesis was Xilinx ISE 11.1. Using this tool a Verilog netlist was generated.

Once the design is synthesized, it's time to simulate the gate-level code generated by the synthesizer, in order to verify that the design still meets the intended functionality and timing. This is known as Gate Level Simulation (GLS). In GLS the Verilog netlist generated by synthesis is simulated along with the test bench written earlier for behavioral simulation. The results are matched with that obtained in behavioral simulation.

4.5 Assertions

Assertions were added to the design in separate units known as vunit using PSL. Simulation with the assertions was carried out using Aldec Riviera-Pro 2011.02. The list of assertions added are as shown.

Sender

- If intr is raised and rst signal is zero then irq must also be raised.
- If rst is raised irrespective of intr, the irq signal must be zero.
- When ack signal is received by the sender in response to an irq, then the irq signal must go low until and unless another intr is raised.
- If at any time an irq is raised it must be eventually acknowledged before the end of simulation.

Receiver

- Whenever isr_enable is set any request must not be acknowledged.
- When irqscheme = 0 i.e., for individual requests ack must be high for one cycle and low for one cycle.
- When irqscheme = 1 i.e., for priority encoded ack must be high for one cycle and low for two cycle before the next acknowledge.
- The value of irqnum must not exceed 64

Priority Generator

- The irq_priori signal must be set when at least one of the irq signals is high

Top Level

- At no time must the number of requests acknowledged be greater than one.
- For priority encoded scheme when valid = '0' then, there should be no acknowledge.

V. CONCLUSION

A An RTL design of Avalon Interrupt Interface was developed in VHDL. The design was verified using testbench in Verilog and assertions in PSL. Since the design and testbench are in two different languages, the project demonstrates the technique of mixed-HDL simulation. In addition to this the project also introduces a new and efficient technique for verification of ASICs i.e., ABV. The synthesis of the design was also successful.

REFERENCES

[1] Ben Cohen, Srinivasan Venkataramanan and Ajeetha Kumari, "*Using PSL/Sugar for formal and dynamic verification: Guide to Property Specification Language for Assertion-Based Verification*", VhdlCohen publishing, 2nd Edition, pp. 1-187, 2004
[2] Assertion-Based Verification, 2003. Available: http://www.synopsys.com/Tools/Verification/Documents/assertion_based_wp.pdf
[3] Property Specification Language Reference Manual, (June 9, 2004), Version 1.1. Available: http://www.eda.org/vfv/docs/PSL-v1.1.pdf
[4] Avalon Interface Specifications, (Aug 2010), Available: http://www.altera.com/literature/manual/mnl_avalon_spec.pdf
[5] SOPC Builder User Guide, Available: http://www.altera.com/literature/ug/ug_sopc_builder.pdf
[6] Aldec ALINT, Available : http://www.aldec.com/products/alint