International Journal of Research in Signal Processing, Computing & Communication System Design
ISSN: 2395 – 3187
Vol: 01, Issue: 01, Jan-June, 2015

Verification of GPIO Core Functions using Universal Verification Methodology

K. Niranjan Reddy[1], U.DhanaLakshmi[2], Dr. PVY Jaya Sree[3]
nianreddy@gmail.com[1], udattudhana@gmail.com[2], pvyjayasree@gitam.edu[3]
Professor & HOD, ECE[1], Asst Professor[2], Assoc Professor[3]
Malla Reddy Engineering College for Women, Hyderabad, Telangana State, India[1,2]
Gitam Institute of Technology, Gitam University, Visakhapatnam, AP, India[3]

Abstract—The OPB GPIO design provides a general purpose input/output interface to a 32-bit On-Chip Peripheral Bus (OPB). The GPIO IP core is user-programmable general-purpose I/O controller. That is use is to implement functions that are not implemented with the dedicated controllers in a system and require simple input and/or output software controlled signals. It is one of the important peripheral that is listed on any FPGA board. In this project we are atomizing the operation of the GPIO by writing the code in SYSTEM-VERILOG and simulating it in QUESTA MODELSIM. The main aim of this project is to verify the output by using GPIO pins depending up on the preference the code. We verify the GPIO modules by using UVM [Universal verification Methodology]. The functional verification of the RTL design of the GPIO is carried out for the better optimum design.

Index Terms— GPIO,OPB,QUESTA MODELSIM, System Verilog, FPGA.

I. INTRODUCTION

The GPIO module is part of Inicore's IP module family. This general purpose input/output controller provides some unique features that eases system integration and use. Each GPIO port can be configured for input, output or bypass mode. All output data can be set in one access. Single or multiples bits can be set or cleared independently. Every GPIO port can serve as an interrupt source and has its own configuration options: • Level sensitive, single edge triggered or level change • Active high or low respectively rising edge or falling edge • Individual interrupt enable register and status flags The core provides several synthesis options to ease the system integration and minimize the gate count: • Selectable CPU bus width: default options are 8/16/32-bit • Selectable number of GPIO ports • CPU read back enable.

II. GPIO(GENERAL PURPOSE I/O)

GPIO is a generic pin on a chip whose behavior (including whether it is an input or output pin) can be controlled (programmed) through software. GPIO pins have no special purpose defined, and unused by default. The idea is that sometimes the system integrator building a full system that uses the chip might find useful to have a handful of additional digital control lines, and having these available from the chip can save the hassle of having to arrange additional circuitry to provide them. For example, the Realtek ALC260 chips (audio codec) have 4 GPIO pins, which go unused by default. Some system integrators (Acer laptops) employing the ALC260 use the first GPIO (GPIO0) to turn on the amplifier used for the laptop's internal speakers and external headphone jack.

A. Architecture of GPIO

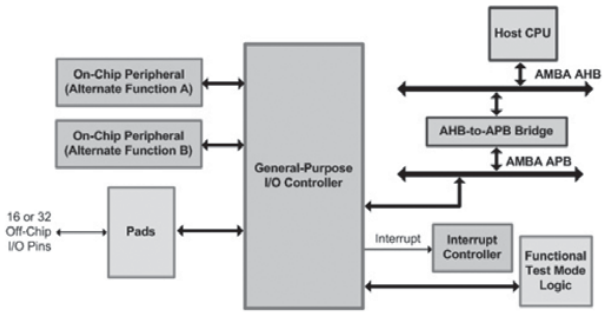

Fig. 1 Architecture of GP I/O

i. Clocks: The GPIO core has two clock domains. All registers except RGPIO_IN are in system clock domain. RGPIO_IN register can be clocked by system clock or by external clock reference.

ii. APB Interface: The host interface is implemented using a 32 bit APB compliant slave interface.

iii. GPIO Registers

The GPIO IP Core has several software accessible registers. Most registers have the same width as number of general-purpose I/O signals and they can be from 1 – 32 bits. The host through these registers programs type and operation of each general-purpose I/O signal or three-state outputs, appropriate open-drain or three-state I/O cells must be used. Part of external interface is also ECLK register. It can be used to register inputs based on external clock reference. General-purpose inputs can generate interrupts so that software does not have to be in poll mode all the time when sampling inputs. Switching output drivers into open-drain or three-state mode will disable general-purpose outputs. To lower number of pins of the chip, other on-chip peripherals can be multiplexed together with the GPIO pins. For this purpose, auxiliary inputs can be multiplexed on general-purpose outputs.

iv. Hardware Reset:

Following hardware reset all general-purpose I/O signals are set into input mode. Meaning, all output drivers are disabled. All interrupts are masked, so that inputs would not generate any spurious interrupts. Gpio_eclk signal is not used to latch inputs into RGPIO_IN register; instead system clock is used General-Purpose I/O as Polled Input. To use general-purpose I/O as input only, corresponding bit in RGPIO_OE register must be cleared to select input mode. Bit RGPIO_CTRL[INTE] and corresponding bit in RGPIO_INTE register must be cleared as well ,to disabled generation of interrupts. Bit RGPIO_IN register reflects registered value of general-purpose input signal. RGPIO_IN is updated on positive edge of system clock or if RGPIO_ECLK appropriate bit is set, on gpio_eclk edge. Which clock edge is selected is defined by value of RGPIO_NEC appropriate bit.

v. General-Purpose I/O as Input in Interrupt Mode:

To use general-purpose I/O as input with generation of interrupts, corresponding bit in RGPIO_OE register must be cleared to select input mode. Corresponding bit in RGPIO_PTRIG register must be set to generate an interrupt on positive edge event on general-purpose input. To generate an interrupt on negative edge event, corresponding bit in RGPIO_PTRIG register must be cleared. If we are enabling interrupts for the first time, we also need to clear interrupt status register RGPIO_INTS. Last, RGPIO_CTRL[INTE]bit and corresponding bit in RGPIO_INTE register must be set to enable generation of interrupts. Bit RGPIO_IN register reflects registered value of general-purpose input signal. RGPIO_IN is updated on positive edge of system clock or if RGPIO_ECLK appropriate bit is set, on gpio_eclk edge. Which clock edge is selected, is defined by value of RGPIO_NEC appropriate bit. Which input caused an interrupt is recorded in interrupt status register RGPIO_INTS. Inputs that caused an interrupt since last clearing of RGPIO_INTS have bits set. Interrupt can be de-asserted by writing zero in RGPIO_INTS register and control register bit RGPIO_CTRL[INTS]. Another way to de-assert interrupts is to disable them by clearing control bit RGPIO_CTRL[INTE].

Vi .General-Purpose I/O as Output

To enable general-purpose I/O output driver, corresponding bit in RGPIO_OE must reset. Corresponding bit in RGPIO_OUT register must be set to the value that is required to be driven on output driver. Corresponding bit in RGPIO_INTE register must be cleared to disable generation of spurious interrupts. Clearing bit in RGPIO_OE register will disable output driver and enable three-state or open-drain. General-Purpose I/O as Bi-Directional I/O. To use general-purpose I/O as bi-directional signal, corresponding bit in RGPIO_OE must be toggled to enable or disable three-state or open-drain mode of bi-directional driver. Corresponding bit in RGPIO_OUT register must be set to the value that is required to be driven on output driver. Corresponding bit in RGPIO_INTE register must be cleared to disable generation of spurious interrupts. If input should generate interrupts, corresponding bit in RGPIO_INTE register must be set and if required also corresponding bit in RGPIO_PTRIG should be set. Corresponding bit RGPIO_IN register reflects registered value of general-purpose input signal. RGPIO_IN is updated on positive edge of system clock or if RGPIO_ECLK bit is set, on gpio_eclk edge. Which clock edge is selected, is defined by value of RGPIO_NEC bit. If an interrupt is enabled and pending, it can be de-asserted by writing zero in RGPIO_INTS register and control register bit RGPIO_CTRL[INTS]. Another way to dessert interrupts is to disable them by clearing control bit RGPIO_CTRL[INTE]General-Purpose I/O driven by Auxiliary Input To drive general-purpose output with auxiliary input, corresponding bit in RGPIO_OE must be set to enable output driver. Corresponding bit in RGPIO_AUX must be set to enable multiplexing of auxiliary input onto general-purpose output.

III. UNIVERSAL VERIFICATION METHODOLOGY

The UVM (Universal Verification Methodology) was introduced in December 2009, by a technical Sub committee of Accellera. UVM uses Open Verification Methodology as its foundation.

Accellera released version UVM 1.0 EA on May 17, 2010.UVM Class Library provides the building blocks needed to quickly develop well-constructed and reusable verification components and test environments. It uses system Verilog as its language. All three of the simulation vendors (Synopsys, Cadence and Mentor) support UVM today which was not the case with other verification methodology. Today, more and more logic is being integrated on the single chip so verification of it is a very challenging task. More than 70 percent of the time is spent on the verification of the chip. So it is a need of an hour to have a common verification methodology that provide the base classes and framework to construct robust and reusable verification environment. UVM provides that.

In this paper , all the terminology related to UVM is introduced along with the sample example. In first phase uvm components are introduced. In second phase some of the features related to UVM are introduced and in final phase small environment is built using UVM from the scratch.

2. Test Bench architecture

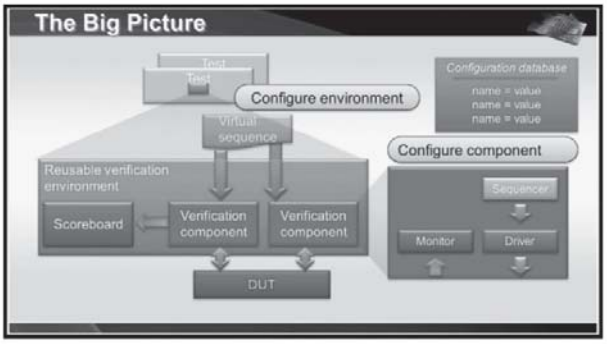

Fig. 1 Test bench architecture

The following subsections describe the components of a verification component.

- Data Item (Transaction)
- Driver (BFM)
- Sequencer
- Monitor
- Agent
- **Environment**

Data Item (Transaction)

Data item are basically the input to the device under test. All the transfer done between different verification components in UVM is done through transaction object. Networking packets, instructions for processor are some examples of transactions. From the top level test many data items are generated and applied to the dut so by intelligently randomizing the data items object we can check corner cases and maximize the coverage on the device under test.

Driver (BFM)

Driver as the name suggest, drive the dut signals. It basically receives the transaction object from the sequencer and converts it in to the pin level activity. So for example it can generate read or write signal, write address and data to be transferred. It is the active part of the verification logic.

Sequencer

Sequencer is the component on which the sequences will run. The dut needs to be applied a sequence of transaction to test its behavior. So sequence of transaction is generated and it is applied to driver whenever it demands by the sequencer.

Monitor

A monitor is the passive element of the verification environment. It just sample the dut signal from the interface but does not drive them. It collect the pin information , package it in form of a packet and then transfer it to scoreboard or other components for coverage information.

Agent

Agent is basically a container. It contains driver, monitor and sequencer. Driver and sequencer are connected in agent. Agent has two modes of operation: passive and active. In active mode it drives the signal to the dut. So driver and sequencer are instantiated in active mode. In passive mode it just sample the dut signals does not drive them. So only monitor is instantiated in passive mode. Normally there is one agent per interface like AHB or APB.

Scoreboard

Scoreboard is a verification component that checks the response from the dut against the expected response. So it keeps track of how many times the response matched with the expected response and how many time it failed.

Environment

Environment is at the top of the test bench architecture, it will contain one or more agents depend on design. If more than one agents are there then it will be connected in this component. Agentsare also connected to other components like scoreboard in this component.

IV. RESULTS AND VERIFIACTION

The GP I/O is carried out for the functional verification using the UVM technique for both the read and write operation. The functional verification is of the RTL design is of the GPIO is yields the complete code coverage.

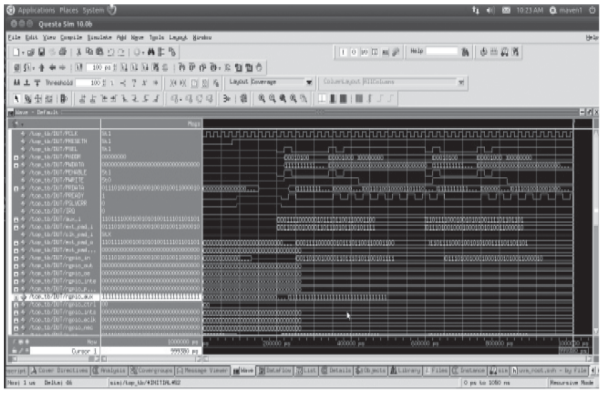

Fig 3 simulation showing GPIO Functional Verification for read operation

Functional Verification of GPIO Core Using UVM

As verification methodology plays a important phase in the circuit design. The read operation of the GPIO is carried out in XILINX for RTL design and the verification methodology is carried out using Questasim 10.0b. The design is carried out using in HDL and the verification is carried out in UVM. The GPIO is set up as DUT for the functional verification and the code coverage is determined using Modelsim is obtained for 100%

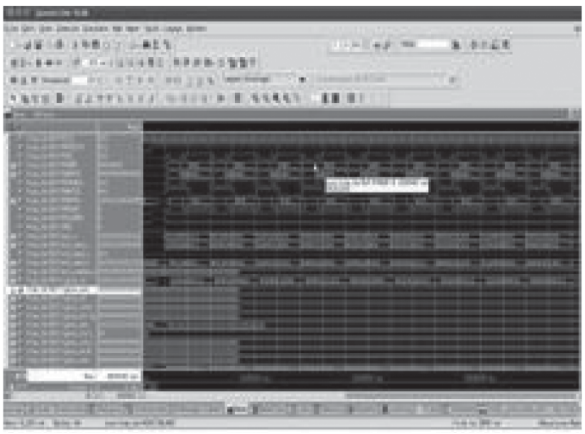

Fig 4 simulationresults for GPIO functional verification for write operation

ModelSim Coverage Report

Number of tests run:	4
Passed:	4
Failed:	0

List of tests included in report...

Design Coverage Summary:		Coverage Summary by Type:			
Weighted Average:	100.0%	Weighted Average:			100.0%
Design Scope	Coverage (%)	Coverage Type	Bins	Hits	Coverage (%)
gpio_pkg	100.0%	Covergroup	12	12	100.0%
gpio_coverage	100.0%				

Fig 5 Functional code coverage of GPIO

V. CONCLUSION

In this we have verified the GPIO core based on UVM technique using Questasim simulator and Modelsim. The code coverage is obtained for the RTL design and 100% code coverage is extracted. This methodolgy provides the complete coverage of the RTL design so as to acquire the fault free Protocol design of GPIO. So that can be implemented in real time systems. This can be further implemented for the ASIC implementation and SOC Applications.

REFERENCES

[1] D.Gajski et al, “Essential Issues for IP Reuse”, Proceedings of ASP-DAC, pp.37-42, Jan. 2000

[2] C.K.Lennard et al, “Industrially proving the SPIRIT Consortium Specifications for Design Chain Integration”, Proceedings of DATE 2006, pp. 1-6, March 2006

[3] K.Cho et al, “Reusable Platform Design Methodology For SOC Integration And Verification”, Proceedings of ISOCC 2008, pp. I-78-I-81, Nov. 2008

[4] W.Kruijtzer et al, “Industrial IP integration flows based on IP-XACT standards” proceedings of DATE 2008, pp. 32-37, March 2008

[5] M.Strik et al, “subsystem Exchange in a Concurrent Design Process Environment” Proceedings of DATE 2008, pp. 953-958, March 2008

[6] GensysIO, http://www.atrenta.com/solutions/gensys-family/gensys-io.htm

[7] SocratesSpinner