Functional Verification of GPIO Core using OVM

L. Veera Raju, B. Kali Vara Prasad, A. L. G. N. Aditya, A. Jhansi Rani, D. Naga Dilip Kumar

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 OVM [Open 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

Manuscript received on April 26, 2012.

L. VeeraRaju, M.tech VLSI, K L University, Vijayawada, India, 9966461747., (e-mail: veera.raju143@gamil.com).

B. KaliVara Prasad, E.C.E Dept, K L University, Vijayawada, India, Phone 9440568944. (e-mail: baditakali@rediffmail.com).

A. L. G. N. Aditya, M.tech VLSI, K L University, Vijayawada, India, Phone 9014936978, (e-mail: adityasind@hotmail.com).

A. Jhansi Rani, M.tech VLSI, K L University, Vijayawada, India, 9985799965., (e-mail:Jhansi.atluri@gamil.com).

D. NagaDilip Kumar, M.tech VLSI, K L University, Vijayawada, India, 9052290361., (e-mail:dilipnaga@gamil.com)

II. GPIO(GENERAL PURPOSE I/O)

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 go 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

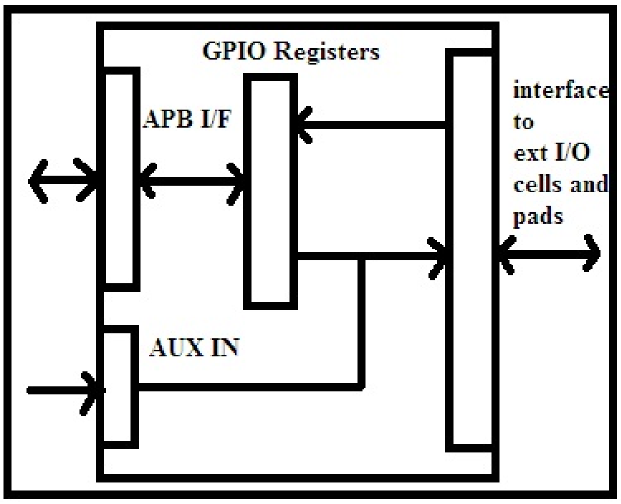


Fig.1 Architecture of GP I/O

*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.GPIO Registers.The GPIO IP Core has several software accessible registers. Most registers have the samewidth as number of general-purpose I/O signals and they can be from 1 – 32 bits. Thehost through these registers programs type and operation of each general-purpose I/Osignal.

*iii. Auxiliary Inputs:*The auxiliary inputs can bypass RGPIO_OUT outputs based on programming ofRPGIO_AUX register. Auxiliary inputs are used to multiplex other on-chip peripheralson GPIO pins.Interface to External I/O Cells and PadsExternal interface connects GPIO core to external I/O ring cells and pads. To supportopen-drain or three-state outputs, appropriate open-drain or three-state I/O cells must beused.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 inpoll mode all the time when sampling inputs.Switching output drivers into open-drain or three-state mode will disable general-purposeoutputs.To lower number of pins of the chip, other on-chip peripherals can be multiplexedtogether with the GPIO pins. For this purpose, auxiliary inputs can be multiplexed ongeneral-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 wouldnot generate any spurious interrupts. Gpio_eclk signal is not used to latch inputs intoRGPIO_IN register; instead system clock is usedGeneral-Purpose I/O as Polled InputTo use general-purpose I/O as input only, corresponding bit in RGPIO_OE register mustbe cleared to select input mode. Bit RGPIO_CTRL[INTE] and corresponding bit inRGPIO_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 appropriatebit is set, on gpio_eclk edge. Which clock edge is selected, is defined by value ofRGPIO_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 inRGPIO_OE register must be cleared to select input mode. Corresponding bit inRGPIO_PTRIG register must be set to generate an interrupt on positive edge event ongeneral-purpose input. To generate an interrupt on negative edge event, corresponding bitin 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 ofinterrupts.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 appropriatebit is set, on gpio_eclk edge. Which clock edge is selected, is defined by value ofRGPIO_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. Interruptcan be de-asserted by writing zero in RGPIO_INTS register and control register bitRGPIO_CTRL[INTS]. Another way to de-assert interrupts is to disable them by clearingcontrol 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 beset. Corresponding bit in RGPIO_OUT register must be set to the value that is required tobe driven on output driver. Corresponding bit in RGPIO_INTE register must be cleared todisable generation of spurious interrupts.Clearing bit in RGPIO_OE register will disable output driver and enable three-state oropen-drain.General-Purpose I/O as Bi-Directional I/OTo use general-purpose I/O as bi-directional signal, corresponding bit in RGPIO_OEmust be toggled to enable or disable three-state or open-drain mode of bi-directionaldriver. Corresponding bit in RGPIO_OUT register must be set to the value that isrequired to be driven on output driver. Corresponding bit in RGPIO_INTE register mustbe 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 correspondingbit in RGPIO_PTRIG should be set.Corresponding bit RGPIO_IN register reflects registered value of general-purpose inputsignal. RGPIO_IN is updated on positive edge of system clock or if RGPIO_ECLK bit isset, 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_OEmust 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. OPEN VERIFICATION METHODOLOGY

OVM provides the best framework to achieve coverage-driven verification (CDV). CDV combines automatic test generation, self-checking test benches, and coverage metrics to significantly reduce the time spent verifying a design. The purpose of CDV is to:

■ Eliminate the effort and time spent creating hundreds of tests.

■ Ensure thorough verification using up-front goal setting.

■ Receive early error notifications and deploy run-time checking and error analysis to

Simplify debugging. The CDV flow is different than the traditional directed-testing flow. With CDV, you start by setting verification goals using an organized planning process. You then create a smart test bench as shown in fig2 that generates legal stimuli and sends it to the DUT. Coverage monitors are added to the environment to measure progress and identify non-exercised functionality. Checkers are added to identify undesired DUT behavior. Simulations are launched after both the coverage model and testbench have been implemented. Verification then can be achieved. Using CDV, you can thoroughly verify your design by changing testbench parameters or changing the randomization seed. Test constraints can be added on top of the smart infrastructure to tune the simulation to meet verification goals sooner. Ranking technology allows you to identify the tests and seeds that contribute to the verification goals, and to remove redundant tests from a test-suite regression.

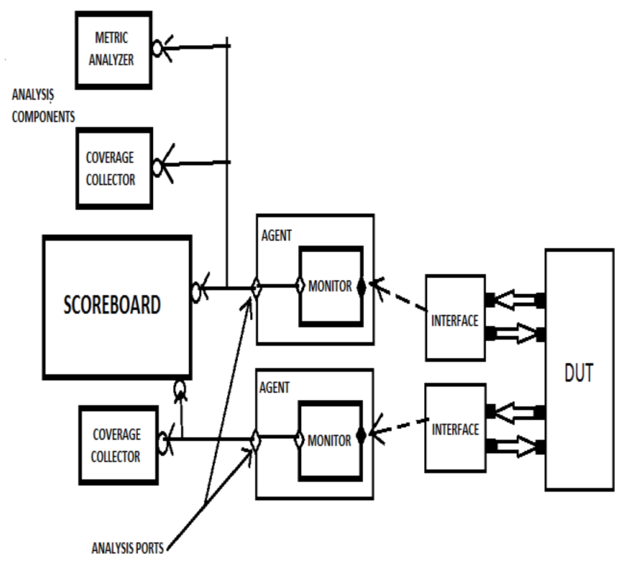


Fig 2 OVM Test Bench setup

OVC Overview

- The following subsections describe the components of an OVC:
- Data Item (Transaction)
- Driver (BFM)
- Sequencer
- Monitor
- Agent
- Environment

A. Data Item:

Data items represent the input to the DUT. Examples include networking packets, bus transactions, and instructions. The fields and attributes of a data item are derived from the data item's specification. For example, the Ethernet protocol specification defines valid values and attributes for an Ethernet data packet. In a typical test, many data items are generated and sent to the DUT. By intelligently randomizing data item fields using System Verilog constraints, you can create a large number of meaningful tests and maximize coverage.

B. Driver (BFM):

A driver is an active entity that emulates logic that drives the DUT. A typical driver repeatedly receives a data item and drives it to the DUT by sampling and driving the DUT signals. (If you have created a verification environment in the past, you probably have implemented driver functionality.) For example, a driver controls the read/write signal, address bus, and data bus for a number of clocks cycles to perform a write transfer.

Sequencer:

A sequencer is an advanced stimulus generator that controls the items that are provided to the driver for execution. By default, a sequencer behaves similarly to a simple stimulus generator and returns a random data item upon request from the driver. This default behavior allows you to add constraints to the data item class in order to control the distribution of randomized values. Unlike generators that randomize arrays of transactions or one transaction at a time, a sequencer captures important randomization requirements out-of-the box. A partial list of the sequencer's built-in capabilities includes.

C. Monitor:

A monitor is a passive entity that samples DUT signals but does not drive them. Monitors collect coverage information and perform checking. Even though reusable drivers and sequencers drive bus traffic, they are not used for coverage and checking. Monitors are used instead. A monitor: Collects transactions (data items). A monitor extracts signal information from a bus and translates the information into a transaction that can be made available to other components and to the test writer.

■ Extracts events: The monitor detects the availability of information (such as a transaction), structures the data, and emits an event to notify other components of the availability of the transaction. A monitor also captures status information so it is available to other components and to the test writer.

■ Performs checking and coverage

Agent: Sequencers, drivers, and monitors can be reused independently, but this requires the environment integrator to learn the names, roles, configuration, and hookup of each of these entities. To reduce the amount of work and knowledge required by the test writer, OVM recommends that environment developers create a more abstract container called an agent. Agents can emulate and verify DUT devices. They encapsulate a driver, sequencer, and monitor. OVCs can contain more than one agent. Some agents (for example, master or transmit agents) initiate transactions to the DUT, while other agents (slave or receive agents) react to transaction requests. Agents should be configurable so that they can be either active or passive. Active agents emulate devices and drive transactions according to test directives. Passive agents only monitor DUT activity.

D. Environment :

The environment (env) is the top-level component of the OVC. It contains one or more agents,as well as other components such as a bus monitor. The env contains configuration properties that enable you to customize the topology and behavior and make it reusable. For example, active agents can be changed into passive agents when the verification environment is reused in system verificationUnits

IV. RESULTS AND VERIFIACTION

The GP I/O is carried out for the functional verification using the OVM 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.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

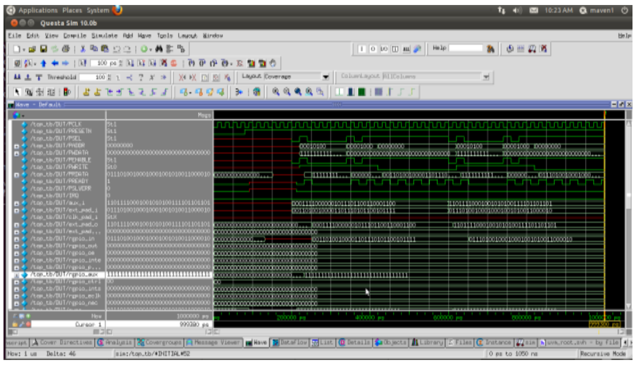


Fig 3 simulation showing GPIO Functional Verification for read operation

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 OVM.

The GPIO is set up as DUT for the functional verification and the code coverage is determined using Modelsim is obtained for 100%

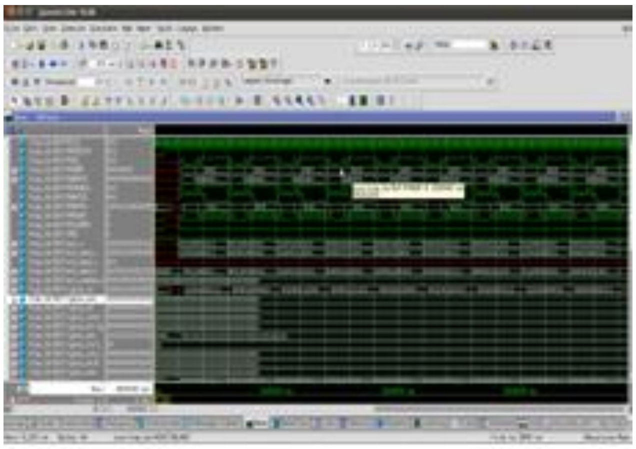
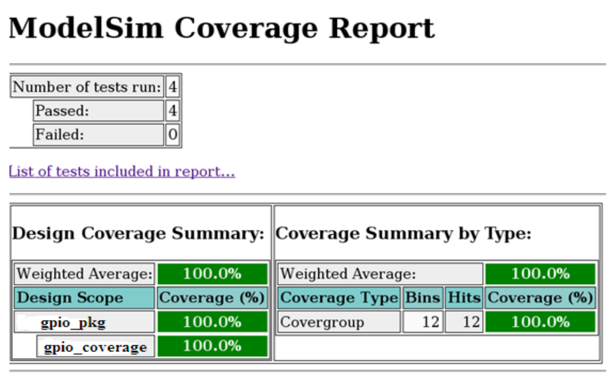


Fig 4 simulation results for GPIO functional verification for write operation.



ModelSim Coverage Report

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

List of tests included in report...

Design Coverage Summary:

Weighted Average:	100.0%
Design Scope	**Coverage (%)**
gpio_pkg	100.0%
gpio_coverage	100.0%

Coverage Summary by Type:

Weighted Average:			100.0%
Coverage Type	**Bins**	**Hits**	**Coverage (%)**
Covergroup	12	12	100.0%

Fig 5 Functional code coverage of GPIO.

V. CONCLUSION

In this we have verified the GPIO core based on OVM 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, http://www.duolog.com/technical-documents

AUTHORS PROFILE

L. VeeraRaju, was born in vijayawada, krishna (Dist.), AP, India. He received B.Tech. in Electronics & Communication Engineering from Prasad V. Potluri Siddhartha college of Engineering, Vijayawada (Dist.,),AP, India ,M.Tech from KL University , Vijayawada, AP, India.

B. K. V. Prasad, was born in Krishna(Dist.,),AP, India. He received B.E inE.C.EfromBharathidaanuniversityTrichy,T.N.India,M.Tech from RGMCET, Kurnool(Dist.,) affiliated to JNTUHyderabad. Pursuing part-time Ph.D.from JNTUH, Hyderabad, AP, India. Hisresearchinterest include automatic circuitreconfiguration and design of fault modelcircuits. He has published 10 publicationsin various journals, conferences at National and Internationallevel and contributed papers in conferences held at Lasvegas,USA in 2009.

A. L. G. N. Aditya, was born in vizag,(dist),AP, India. He received B.Tech. in Electronics & Communication Engineering from TPIST,AP. India ,M.Tech from KL University , Vijayawada, AP, India. He has undergone 8 international conferences and 1 publishment in IEEE

Jhansirani.Atluri, was born in Narasaraopet, Guntur (Dist.), AP, India. She received B.Tech. in Electronics & Communication Engineering from St. Ann’s College of Engg. ,Chirala ,prakasam (Dist.,),AP, India ,M.Tech from KL University , Vijayawada, AP, India

D. NagaDilip Kumar, was born inpagulapadu, PrakasamDist.,),AP, India. HereceivedB.Tech in E.C.E from JntuniversityHyderabad, A.P, india. Pursuing M.Techfrom KLU. His research interest includesTesting and Verification of fault models