International Journal of Engineering Science Invention
ISSN (Online): 2319 – 6734, ISSN (Print): 2319 – 6726
www.ijesi.org ||Volume 6 Issue 8|| August 2017 || PP. 88-97

Design Validation of Inter-IC (I²C) Bus and Implementation in Real-Time PCI Application

*M. Srinu, B.Krishna
Asst. Professor, Department of ECE, Mahaveer Institute of Science & Technology-Hyd,
Assistant Professor, Department of ECE, AAR Mahaveer Engineering College-Hyd,
Corresponding Author: M. Srinu

Abstract: *Validation is the activity that determines the correctness of the design that is being created. It ensures that the design does meet the specifications required and operates properly. This paper will focus on developing an exhaustive, reusable and configurable environment to validate the Inter-IC (I²C) interface on one of Intel's silicon products. Inter-IC (I²C) provides a way of serial communication between CPU and low speed peripherals with in short range. In this paper used a specimen based validation environment for validation of Inter-IC (I²C) protocol. Specimen provides inbuilt constraint solver, which allows fast and easy test generation. This paper will also focus on developing the validation environment to validate the System Management Bus (SMBus) and PCA9555 IO expander, which are usage models of Inter-IC (I²C) bus. System Management Bus (SMBus) provides a control bus for system. PCA9555 IO expander provides 16 bits of General Purpose parallel Input/output (GPIO) expansion for Inter-IC (I²C) bus applications. Hot plug operation allows addition or removal of components that would expand or shrink the system without significant interruption to the operation of the system. Implementing Peripheral Component Interconnect (PCI) Hot plug functionality with System Management Bus (SMBus) and PCA9555 IO expander reduces the required number of IO pins. Such Peripheral Component Interconnect (PCI) Hotplug validation environment can be developed by re-using System Management Bus (SMBus) and PCA9555 validation environments to validate Peripheral Component Interconnect (PCI) Hotplug operation on Intel Peripheral Component Interconnect (PCI) Express Controller.*
Keywords: *I^2c, SMBus, GPIO, PCI, SPI, ISA, VESA, AGP, IPMI, NXP, API, BFM.*

Date of Submission: 08-08-2017 Date of acceptance: 31-08-2017

I. INTRODUCTION

Validation is the activity of the correctness of the design is created. It ensures that the design does meet the specifications required and operates properly. In the IC design process is mapped into its implementation correctly in terms of Validation. The number of gates increases in modern integrated circuits coupled with the use of Intellectual Property (IP) cores and advances in design re-use methodologies are contributing to larger, more complex and highly integrated designs. These increased complexity results of designs take more effort and time to verify. Validation tasks commonly accounts for 50% to 80% of the chip's development schedule. So, validation is the bottleneck in delivering today's highly integrated electronic systems and chips.

A bus is a set of physical connections which can be shared by multiple hardware components in order to communicate with one another. The purpose of buses is to reduce the number of "pathways" required for communication between the components, by carrying out all communications over a single data channel. If only two hardware components communicate over the line is called a hardware port. Hardware components may include the CPU, main memory, and I/O devices. The four components are connected with ports is shown in fig.1.1 and components are connected with bus is shown in fig.1.2. It is observed that number of pathways needed for communication is less when components are connected with bus.

Different types of buses are common in practice are SPI (Serial Peripheral Interface) , I²C(Inter Integrated Circuit), ISA (Industry Standard Architecture), VESA (Video Electronics Standards Association), PCI (Peripheral Component Interconnect), USB (Universal Serial Bus), AGP (Advanced Graphics Port), PCI Express.

A. I²C BUS

I²C Bus was developed by NXP Semiconductors. It is a simple bidirectional two wire bus for efficient inter-IC control. This bus is called the Inter-IC or I²C-bus.

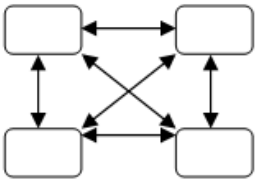


Fig. 1.1: Components Connected with ports

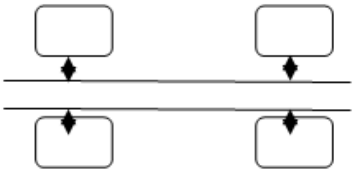


Fig. 1.2: Components connected with bus

I²C is peripherals and low manufacturing costs are more important than speed. It is a serial communication protocol. I²C connects many peripheral devices to CPU. It is also used for reading monitors and sensors. A particular strength of I²C is that a microcontroller can control a network of device chips with just two general-purpose I/O pins and software. Additionally, I²C-bus is used in a variety of control architectures such as System Management Bus (SMBus), Power Management Bus (PMBus), Intelligent Platform Management Interface (IPMI), and PCA9555 IO expander.

As I²C is used in different architectures to inter connect different component in the system, it is very important to ensure the correct operation of I²C.So exhaustive validation is required.

II. VALIDATION ENVIRONMENT

The objective of pre-silicon validation is to verify the correctness and sufficiency of the design. This approach typically requires modeling the complete system, where the model of the design under test may be (Register Transfer Level) RTL, and other components of the system behavioral or bus functional models. The validation environment functional models are connected to the design under test (RTL) for validation. The goal is to subject the Design under Test (DUT) to real-world-like input stimuli. Pre-silicon validation aims are (1) Validate design sufficiency. (2) Validate design correctness. (3) Verify implementation correctness. (4)Uncover unexpected system component interactions. The Block diagram of validation environment is shown in the fig. 2.1.

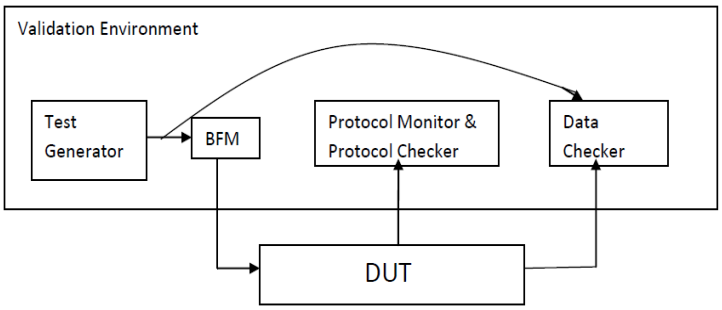


Fig. 2.1 Components of Validation Environment

The test generation methodology is critical in building a system level pre- silicon validation environment capable of generating real-world-like input stimuli. A dynamic test generator is more effective in creating very interesting, reactive test sequences. An automated test generation tool should be capable of handling direct testing, pseudo-random testing and reactive testing. In direct testing, users specify the sequence of events to generate. It is efficient for verifying known cases and conditions. Pseudo-random testing is useful in uncovering unknown conditions or corner cases. Pseudo-random test generation, where transactions are generated from user-defined constraints, can be interspersed with blocks of direct sequences of transactions at periodic intervals to re-create real-life traffic scenarios in a pre-silicon validation environment. Dynamic test generation also facilitates reactive test generation. It implies a change in test generation when a monitored event is detected during simulation.

BFMs will drive the generated input stimulus to DUT. It provide a transaction-level (API) Application Programming Interface are designed to handle concurrency and parallelism. It is suitable to use in an automated test generation environment. It also offers a high degree of controllability for the model behavior to emulate a real device with real operating characteristics through programmable delay registers and configuration registers [1]. The bus protocol monitors provide dynamic protocol checking and can be used in automated test generation environments. They provide dynamic bus state information, which can be used to provide dynamic feedback to user tests or automated test controllers. The bus protocol checkers will check whether transactions are pausing according to the protocol or not. The Data checker receives data from the DUT output interface. Based on the input stimuli provided, it will check whether output data is correct or not. The test generator utilizes transaction generators to create constraint-based concurrent sequences of transactions at the different interfaces of the DUT. The controller can generate transactions pseudo-randomly, for a user specified sequence. It can also perform specific tasks or dynamically reload input constraints upon a certain event occurring during simulation.

III. SPECMAN

Spec man is a development environment for the e language. The main difference is that an e code can never execute stand-alone without Spec man. Spec man automates the entire validation process from the validation of individual blocks, to the full chip all the way to the project level, promoting productivity, quality and predictability. Spec man is a comprehensive environment that contains engines like constraint solver, coverage engine etc. for all aspects of validation: Automatic generation of functional tests, Data and assertion checking and Functional coverage analysis.

A. Spec man Capabilities

Significant reuse is not possible with VHDL & Verilog. Spec man provides reusable methodology for developing reusable validation environment. Then spec man allows the development of reusable validation environments. Spec man has a constraint Solver which allows fast and easy test generation for functional validation. By specifying constraints, one can quickly and easily target the generator to create any test in functional test plan. These tests can be generated on-the-fly based on the current design state, making it possible to generate even hard-to-reach corner cases.

The Data and Assertion Checking is the Powerful temporal constructs enable to capture complex protocols for assertion checking. On-the-fly data checking and generation allows context-specific expected values. The Functional Coverage Analysis is an executable functional test plan measures the progress of validation functional analysis automatically identifies holes in the test coverage. Validation schedules become more predictable because functional coverage is a meaningful and direct measure of the completeness of validation.

The HDL Simulator Interfaces are integrated with Spec man. Internal signals of the device under test can be sampled and driven. 100 percent controllability and observability otherwise inaccessible internal signals allow all engines of Spec man full access to signal values during the simulation.

B. e Reusable Methodology (eRM)

e language allows the creation of the validation environment as an independent, plug and play package called *e language* Validation Component(eVC) [4]. Each eVC consists of complete validation environment as shown in fig. 3.1.

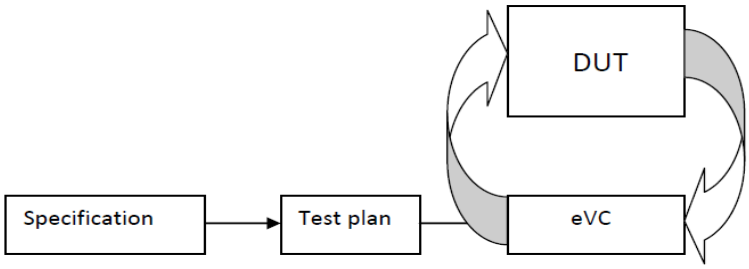


Fig. 3.1: Validation Environment with eVC

These eVCs can be used as a validation environment integrated with a larger environment. To use these eVCs as plug and play devices Spec man defines a methodology called ***e*** Reusable Methodology (eRM) which defines guidelines and best-known-methods for ***e***VC development. The typical eVC architecture is shown in fig. 3.2.

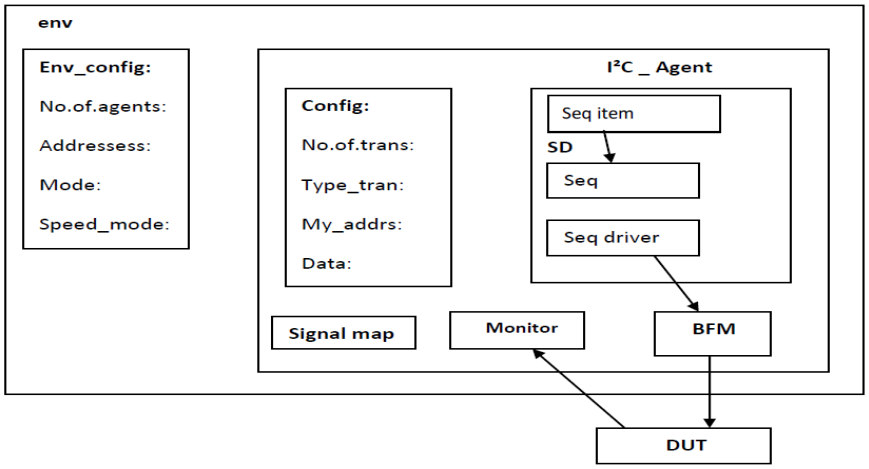


Fig. 3.2: Typical e Verification Component (eVC)

The agents are unit's instantiated within the environment. In the eVC, I²C has one agent, which is transmitting agent. These agents have the following components are Configuration, Signal Map, Sequence Driver, Bus Functionality Module (BFM) and Monitor.

IV. INTER IC CONNECT (I²C) BUS

NXP Semiconductors developed a simple bidirectional two-wire I²C bus for efficient inter-IC control. All I²C-bus devices incorporate an on-chip interface which allows them to communicate directly with each other via the I²C-bus. The I^2c-bus is required only two bus lines are a serial data line (SDA) and a serial clock line (SCL) [3]. Each device connected to the bus is software addressable by a unique address. It is a true multi-master bus including collision detection and arbitration to prevent data corruption if two or more masters simultaneously initiate data transfer. It is used for Serial 8-bit oriented, bidirectional data transfers. The operating speeds are up to 100 kbps in the Standard-mode, up to 400 kbps in the Fast-mode, up to 1 Mbps in Fast-mode and up to 3.4 Mbps in the High-speed mode. The number of ICs can be connected to the same bus is limited by a maximum bus capacitance.

A. I²C Bus Protocol

The two wires carry information between the devices connected to the bus. Each device is recognized by a unique address (whether it is a microcontroller, LCD driver, memory or keyboard interface) and can operate as either a transmitter or receiver, depending on the function of the device. An LCD driver is only a receiver whereas a memory can both receive and transmit data. The devices can be considered as masters or slaves for transmitters and receivers when performing data transfers. A master is the device which initiates a data transfer on the bus and generates the clock signals to permit that transfer. At that time, any device addressed is considered a slave.

The I²C-bus is a multi-master bus means that more than one device capable of controlling the bus can be connected to it. As masters are usually microcontrollers, let's consider the data transfer between two microcontrollers connected to the I²C-bus. There is possibility that more than one master could try to initiate a data transfer at the same time. To avoid this chaos an arbitration procedure has been developed. This procedure relies on the wired-AND connection of all I²C interfaces to the I²C-bus. If two or more masters are try to put information onto the bus, the first to produce a 'one' when the other produces a 'zero' will lose the arbitration. When the clock signals during arbitration are a synchronized combination of the clocks generated by the masters using the wired-AND connection to the SCL line.

Generation of clock signals on the I²C-bus is always the responsibility of master devices. Each master generates its own clock signals when transferring data on the bus. Bus clock signals from a master can be altered when they are stretched by a slow slave device holding down the clock line.

B. I²C Bus Validation Environment

I²C Bus validation environment preparing test plan and validation of each item in the test plan. Spec man's eRM methodology is used for developing the validation environment for I²C bus. Developed and eVC for I²C as shown in fig. 4.10

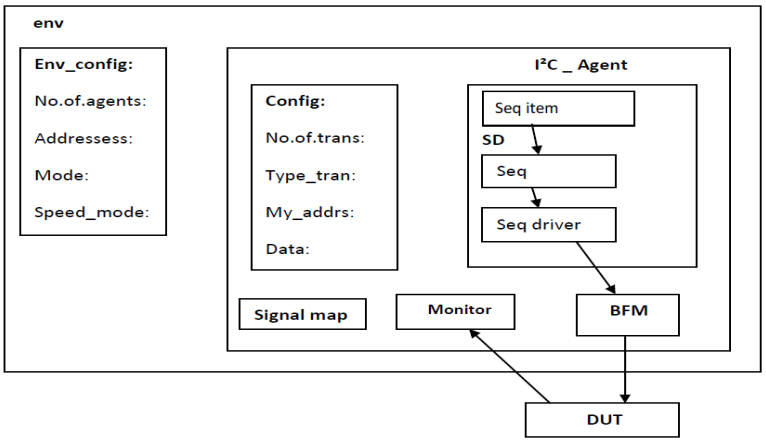


Fig. 4.10: eVC for I²C

B.1 Test plan

Test plan is a list of scenarios to be validated. For I²C the following scenarios must to be validated to ensure the proper operation of I²C. (1) Master transmitter operation. (2) Master receiver operation. (3) Slave transmitter operation. (4) Slave receiver. (5) Arbitration mechanism. (6) Acknowledgement and Not Acknowledgement generation.

B.2 Validation

It is the operation of I²C bus and developed I²C protocol checker. BFM collects the data from sequence driver and drives to I²C RTL. I²C protocol checker will monitor the transaction happening on I²C and flags an error, if transactions are not happening according to the I²C bus protocol. Implementation of I²C BFM is the most complex aspect in I²C eVC. Based on the I²C specification developed a single state diagram which can behave as either master or slave or both depending on the configuration mode value. The State diagram of I²C BFM is shown in fig. 4.11.

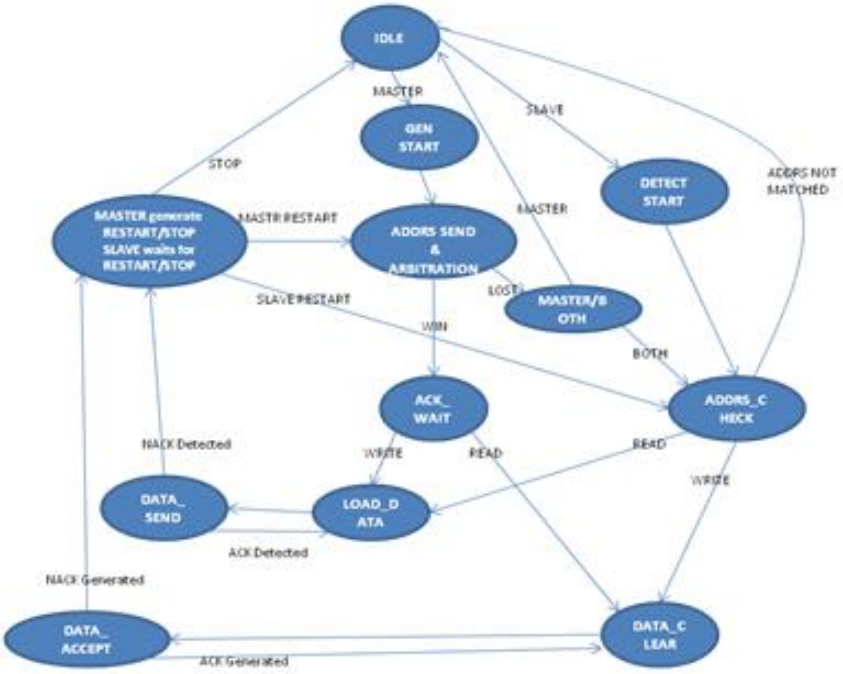


Fig. 4.11: State diagram for I²C BFM

B.3 Other Uses of I²C Bus Protocol

The I²C-bus is used as the communications protocol for several system architectures. These architectures have added command sets and application-specific extensions in addition to the base I²C specification. The derived architectures are going to the similar protocol and physical interfaces of I²C bus. Some of architectures are derived from I²C bus protocols are: (1) SMBus- System Management Bus. (2) PMBus- Power Management Bus. (3) PCA9555 IO expander.

V. SYSTEM MANAGEMENT(SM) BUS

It is a two-wire interface through which simple system can communicate with the rest of the system. It provides a control bus to the system. It is based on the principles of operation of I²C.The SMBus uses I²C hardware and I²C hardware addressing, but adds second-level software for building special systems. Its specifications include an Address Resolution Protocol that can make dynamic address allocations. Dynamic reconfiguration of the hardware and software allow bus devices to be 'hot-plugged' and used immediately, without restarting the system. The devices are recognized automatically and assigned unique addresses. This is the advantage of results in a plug-and-play user interface.

***A.* SMBus Protocol**

The SMBus protocols are a subset of the data transfer formats defined in the I²C specifications [4]. SMBus is having two additional features that are Command code and Packet Error Checking. The Command code will be used for processing the data received.

A.1 Packet Error Checking

Implementation of Packet Error Checking by SMBus devices is optional. It is implemented by appending a Packet Error Code (PEC) at the end of each message transfer. Each protocol has two variants: one with the Packet Error Code (PEC) byte and one without. PEC is calculated based on all message bytes.

B. SMBus validation Environment

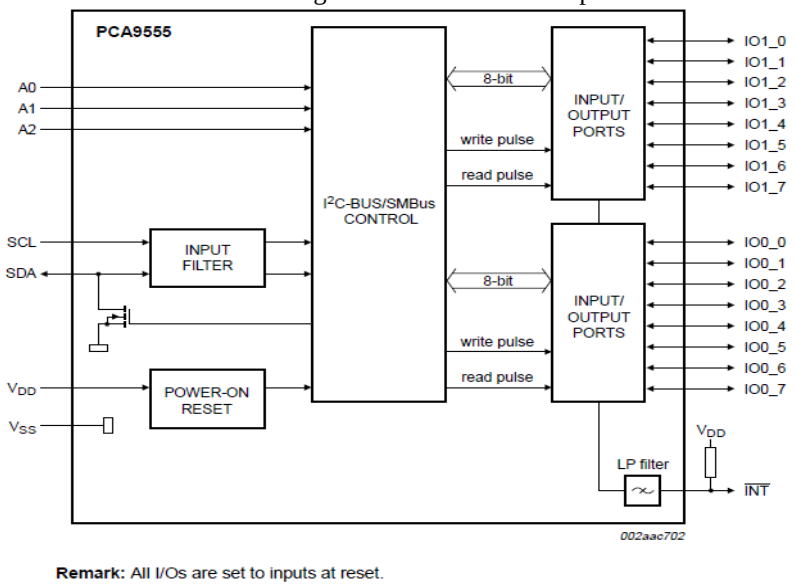
It can be developed by reusing the I²C bus validation environment and it requires software to support the additional features command code and packet error checking. SMBus protocol checker can be developed by reusing the I²C Bus protocol checker, modifying it to support for identifying the first data byte received as command code, Read and Write transaction according to SMBus protocol. SMBus validation environment, preparing test plan and validating the each item in the test plan.

B.1 Test plan: Reception and transmission of command code and Packet Error Checking is not supported in the SMBus RTL.

B.2 Validation: The only additional feature validated in case of SMBus is Command code.

VI. PCA9555 IO EXPANDER

It is a 24-pin CMOS device provides 16 bits of General Purpose parallel Input/output (GPIO) expansion for I²C -bus/SMBus applications. It provide a simple solution when additional I/O pins are required for ACPI power switches, sensors, push buttons, LEDs, fans, etc. The PCA9555 consists of two 8-bit configurations of input, output and polarity inversion registers. The system master can enable the I/Os as either inputs or outputs by writing to the I/O configuration bits. The data for each Input or Output is kept in the corresponding Input or Output register. The polarity of the read register can be inverted with the Polarity Inversion register. Three hardware pins (A0, A1 and A2) vary the I²C-bus address and allow up to eight devices to share the same I²C-bus/SMBus. The block diagram of PCA9555 IO Expander is shown in fig. 6.1.



Remark: All I/Os are set to inputs at reset.

Fig. 6.1: Block diagram of PCA9555 [5]

A. PCA9555 IO Expander Validation Environment

PCA9555 validation environment can be developed by reusing the SMBus validation environment. The sequence driver of SMBus to generate command code and checker should get the information like command code and the values updated in the port registers based on the information.

A.1 Test Plan

The Validation of PCA9555 is mostly around the validating command code. If command code is 0 or 1, then PCA9555 should read the port bits, command code is 2 or 3, then PCA9555 should write to the port bits and command code is 4 or 5, then the data followed by the command code updates the Inversion registers. During read operation, the read value must be inverted in accordance with respective inversion register value. If command code is 6 or 7, then the data followed by the command code will update the configuration registers. Some of the ports bit behave as input and some output based on values written to configuration register.

A.2 Validation

The complete validation of PCA9555 is required to validate all the scenarios covered in the test plan.

VII. PCIE HOTPLUG

Addition/Removal of components would expand/shrink the system without significant interruption to the operation of the system is called as Hotplug operation. A well-known example of this functionality is the Universal Serial Bus (USB) allows the user to add or remove peripheral components such as a mouse, keyboard

or printer etc. PCI (Peripheral Component Interconnect) is a scalable I/O serial bus technology set to replace PCI bus [6]. In all modern PCs, from consumer laptops and desktops to enterprise data servers, the PCI bus serves as the primary motherboard-level interconnect. Hotplug technology allows for replacement of a failed adapter while the remaining adapters provide end-users with uninterrupted service. Hotplug technology will support three major processes are hot replacement, hot upgrade, and hot expansion.

A. Implementation of PCI Hotplug

PCI Hotplug functionality can be implemented by using SMBus and PCA9555 and without using SMBus and PCA9555. Implementing the Hotplug functionality with SMBus and PCA9555 IO expander doesn't require one dedicated pin for each of Hotplug interface signals. So it require reduce the number of IO pins. All Hotplug interface signals are connected to IO expander PCI Hotplug logic communicates with IO expander using SMBus. The block diagram of interfacing Hotplug signals to PCI using SMBus and IO expander is shown in the fig. 7.2

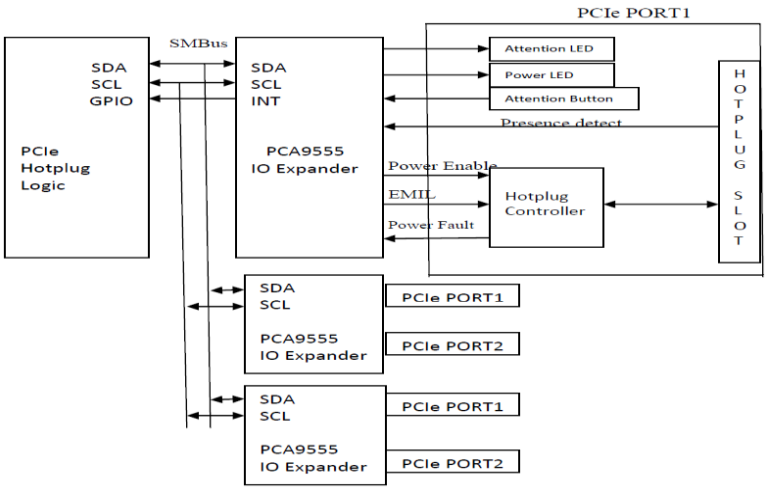


Fig. 7.2: PCI Hotplug implementation using SMBus and PCA9555

All Hotplug interface signals are directly connected to the PCI Hotplug logic. So this implementation requires more number of IO pins. The block diagram of interfacing Hotplug signal directly to PCI is shown in the fig. 7.3.

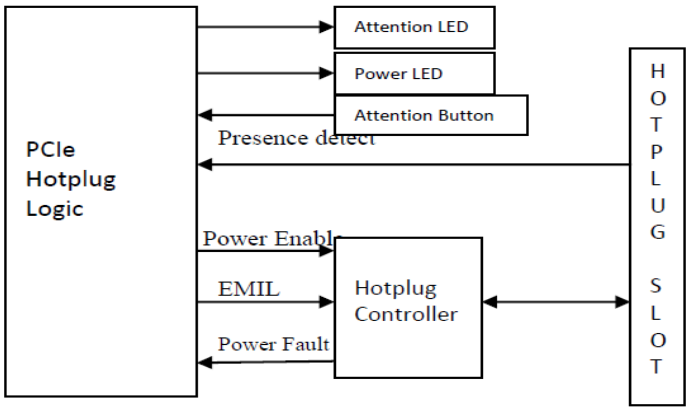


Fig. 7.3: Interfacing Hotplug signal to PCI Hotplug logic directly

B. PCI Hotplug Validation Environment

The Hotplug test generator is developed for generating Hotplug transactions. PCA9555 test generator should get input transactions from Hotplug test generator. To emulate the behavior of the Hotplug software to control the output signals at the Hotplug interface. The Hotplug event response of software must acknowledge the user by controlling Indicators and it should control the power on/off. To enable the PCA9555 protocol checker to check the transactions and to develop a Hotplug monitor which update Hotplug Checker and the Hotplug registers like Slot Status registers. Hotplug Checker should flag an error if Hotplug registers are not updated properly and the transaction is not met. The block diagram of Hotplug validation environment is shown in fig. 7.4

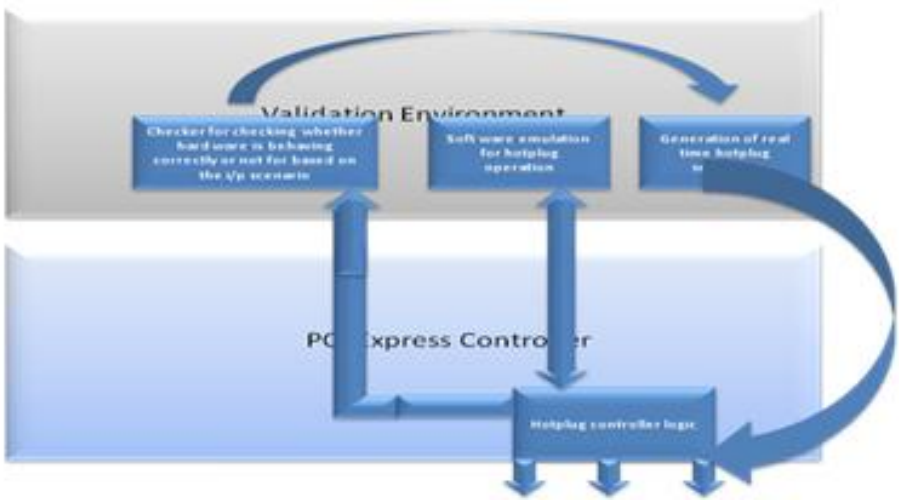


Fig. 7.4: Hotplug Validation Environment

B.1 Test Plan

Validate each step in Hot Insertion or Removal Selectively disable each Hotplug component by programming Slot Capability registers and ensure that no operation should happen on the disabled component. Interrupt generation on Hotplug events can be disabled using Slot Control register. Disabled Interrupt generation for each of the Hotplug event and ensure no interrupt should be generated. During Hotplug operation, Slot Status register reflect the status of Hotplug interface signal. On each Hotplug event, check whether Slot Status register is getting updated or not and if interrupt generation is enabled interrupt should be generated. Randomize the Hotplug Validation Environment to validating the different scenario.

B.2 Validation

The complete validation of PCI Hotplug operation is to validate all the scenarios in the test plan.

VIII. RESULT ANALYSIS

The I²C Protocol Checker Output waveform when RTL master writing to I²C validation environment is shown in fig.8.1.

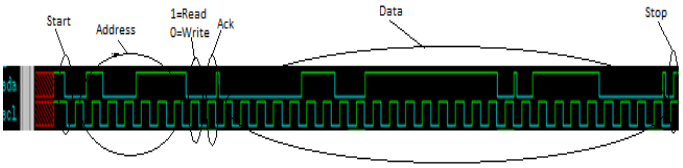


Fig. 8.1: I²C protocol checker output when RTL master writing to I²C validation environment.

The I²C Protocol Checker Output waveform when RTL master Reading I²C validation environment is shown in fig.8.2.

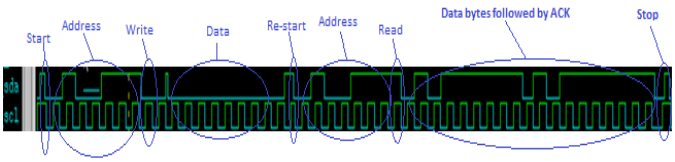


Fig. 8.2: I²C protocol checker output when RTL master Reading I²C validation environment.

Number of test cases developed = 8
Number of Bugs found = 0
Validation Result = RTL is bug free

The SMBus protocol checker output waveform when SMBus master writing to SMBus slave is shown in fig.8.3.

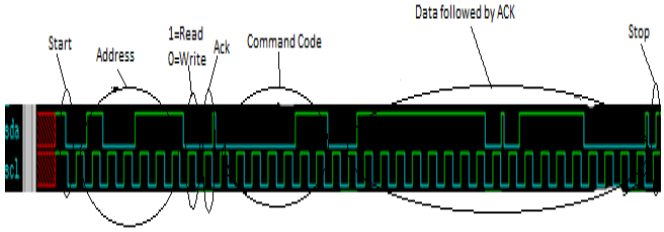


Fig. 8.3: SMBus protocol checker output when SMBus master writing to SMBus slave.

The SMBus protocol checker output waveform when SMBus master Reading SMBus slave is shown in fig.8.4.

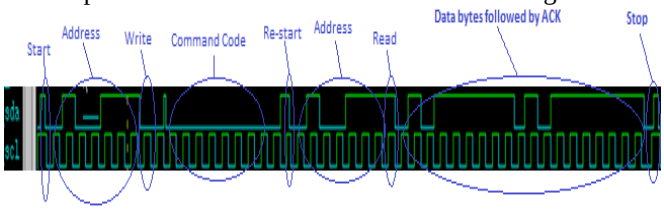


Fig. 8.4: SMBus protocol checker output when SMBus master Reading SMBus slave.

Number of test cases developed = 7
Number of Bugs found = 0
Validation Result = RTL is bug free

The PCA9555 protocol checker output waveform when PCA9555 master is writing to two ports of the slave is shown in fig.8.5.

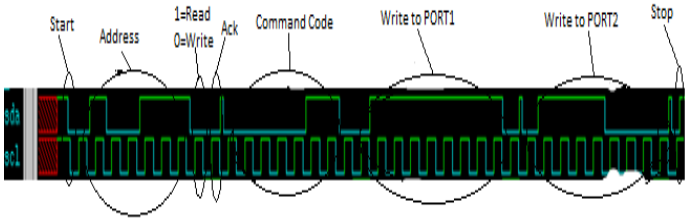


Fig. 8.5: PCA9555 protocol checker output when PCA9555 master is writing to two ports of the slave.

The PCA9555 protocol checker output waveform when PCA9555 master is reading two ports of the slave is shown in fig.8.6.

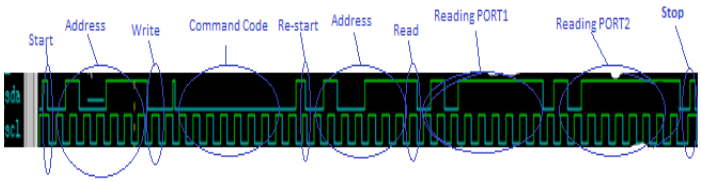


Fig. 8.6: PCA9555 protocol checker output when PCA9555 master is reading two ports of the slave.

Number of test cases developed = 9
Number of Bugs found = 0
Validation Result = RTL is bug free

The Hotplug monitor output waveform while validating presence detect bit is shown in fig.8.7.

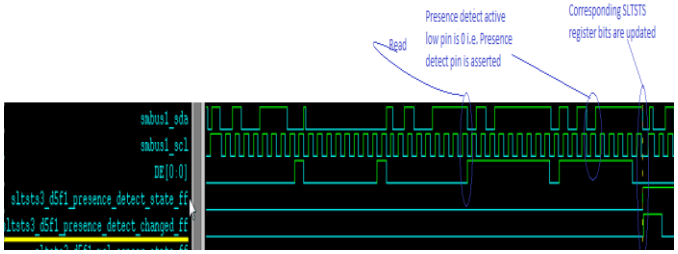


Fig. 8.7: Hotplug monitor output while validating presence detect bit

The Hotplug monitor output waveform while validating MRL status bit is shown in fig.8.8.

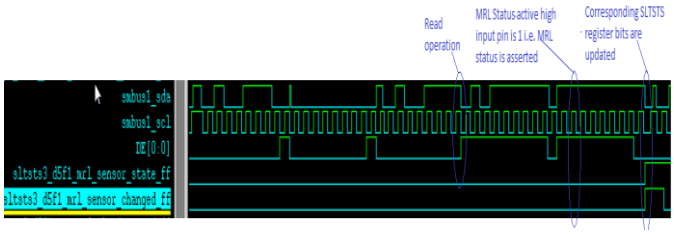


Fig. 8.8: Hotplug monitor output while validating MRL status bit

The Hotplug monitor output waveform while validating Attention button bit is shown in fig.8.9.

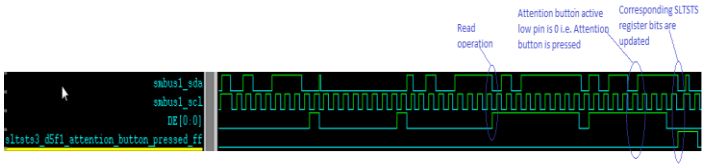


Fig. 8.9: Hotplug monitor output while validating Attention button bit

Number of test cases developed = 27
Number of Bugs found = 5
Validation Result = RTL needs fixes for the Bugs

IX. CONCLUSION

As validation tasks commonly accounts for 50% to 80% of the chip's development schedule, it is very important to develop a re-usable validation environment. Spec man provides re-usable methodology for developing validation environment.

An exhaustive, reusable and configurable environment is developed to validate the I²C bus protocol. SMBus and PCA9555 IO expander are two applications of I²C bus. Its validation environment is developed by re-using the I²C validation environment and validated SMBus and PCA9555 IO expander protocols.

Implementing PCI Hotplug operation using SMBus and PCA9555 IO expander reduces the number of IOs required. PCI Hotplug validation environment is developed by re-using SMBus and PCA9555 validation environments. Validated PCI hot insertion and PCI hot removal sequences.

REFERENCES

[1] David Dempster and Michael Stuart., "Verification Methodology Manual Techniques for Verifying HDL Design", 1st edn, Biddles Ltd., Guildford and King"s Lynn, Great Britain, 2002.
[2] "e Reuse Methodology(eRM) Developer Manual", Version 4.3.5, Versity Design.
[3] "UM10204 I²C-bus specification and user manual", Revision 3, 19th June 2007.
[4] "System Management Bus Specification", Rev. 1.1, 11th Dec 1998.
[5] "PCA9555 16-bit I²C-bus and SMBus I/O port with interrupt", Revision 08, 22th October 2009.
[6] "PCI Express™ Base Specification", Revision 1.0a, 15th April 2003.
[7] Bryan Le, "Enabling Hot-Plug with IDT PCI Express® Gen2 System Interconnect Switches", 20th June 2009.
[8] "e Language Reference Manual", preliminary, Versity Design.

International Journal of Engineering Science Invention (IJESI) is UGC approved Journal with Sl. No. 3822, Journal no. 43302.

M. Srinu. "Design Validation of Inter-IC (I²C) Bus and Implementation in Real-Time PCI Application." International Journal of Engineering Science Invention (IJESI), vol. 6, no. 8, 2017, pp. 88-97.