FPGA-based Solution for Automation of Post-Silicon Validation

Debabrata Ghosh & Felix Paul

Senior Verification Engineer, Infineon Technologies, Bangalore, India
E-mail : debabrata.ghosh@infineon.com, felix.paul@infineon.com

Abstract : **Post Silicon Validation (PSV) is the key task to ensure needed quality is achieved in silicon before product reaches customer. Hence, PSV necessitates stressing the system and its internal components in terms of Use-Case scenarios. Also, with more and more sophisticated requirements, the complexity of the chip is increasing day by day, and achieving time to market (TTM) is pivotal for product and business success. The proposed approach of FPGA-based solution for validation automation helps to accelerate validation cycle by automation. The objective of the effort is to provide an efficient solution of Automation of Silicon Validation that will reduce the manual effort and increase the reliability of the test setup. The design should also provide the flexibility to change the configuration on-the-fly. The paper presents a method of changing FPGA glue logic with the help of three pins connected and controlled by the Microcontroller. These three wires can cater to a large number of pin-interconnect.**

I. INTRODUCTION

Post Silicon Validation for SoCs typically involves IP-level validation and this poses challenges in terms of connecting IPs to external world .External connection can be external loop back, connection to slave devices or connections to protocol specific connectors.

Currently when an IP is present within the Microcontroller and it needs external connection to any other IP or slave device, a small extender board is used (it is connected to the Target Board's Connector pins) and nature of connection is through wires. External connections are not reliable and could potentially waste validation and debug time.

Commonly followed approach would be to chose a daughter board (mounted with all different kind of slave devices) and connect the same to the target board (carrying the microcontroller).Microcontroller's port pins have to be hard wired in external loopback or IP to slave connections have to be hard wired or use jumper alternatives. Also when the next variant of microcontroller comes out, porting the solution to new pin-mapping is tedious in case of wired connections. Peripheral IP's signals might come out from a different port pins, resulting in incorrect connections to slave devices or external loopback, as the slave board had already hard wired connections.

Current approach of FPGA based solution of connectivity brings portability in addressing different variant SoC product-line and reliable than traditional approach of wired solution.

II. PROBLEM OR SCOPE OF WORK

The Post-Silicon validation requires a lot of manual effort, which involves wiring between ports of microcontroller as well as need to change the connection on-the-fly. Problem with wiring results in loose connections, incorrect connections, introduction of skew and delay in overall Post-Silicon process. Also in the traditional way of manual validation, there is lack of flexibility or compatibility to switch to future products.

The scope of work is thus can be categorized broadly into following divisions:

Introduction of automation and automated regression in Post-Silicon validation process, which will remove manual connectivity problems.

A method to change the connectivity on-the-fly as and when required by Microcontroller.

A design to introduce flexibility such that some of the selective IP modules can be directly connected to each other by bypassing the Automation setup.

III. SOLUTION

A. Choosing FPGA for Automation

To address the problem of manual wiring and introducing automation, we choose FPGA as the hardware for Automation. FPGA can serve the purpose of Automation in the following ways:

WQManual wiring can be removed by introduction of FPGA in between the ports. This can be achieved easily by connecting all the ports of microcontroller to the FPGA. Hence, any port can communicate to each

other through FPGA. This removes manual intervention completely.

We can program our desired design into FPGA, which will serve any activity that Microcontroller was supposed to expect from port connecting to slave device.

We propose a method to send the command to FPGA, on reception of which it will change the connectivity on-the-fly. FPGA needs to be programmed such a way that it will interpret the command sent by the Microcontroller and do the necessary connections or any other action on run-time.

B. Introducing Flexibiity in design for Automation

The problem of FPGA is its inherent delay, introduced by it. So, high-speed signals cannot be routed through FPGA. Also, analog modules need to bypass FPGA for connectivity. Hence, we need a flexible solution for automation, which will bypass FPGA for certain modules.

The flexibility can be addressed at hardware level, where a choice can be made for direct connection to bypass FPGA. Figure 1 describes the pictorial view of the flexible design.

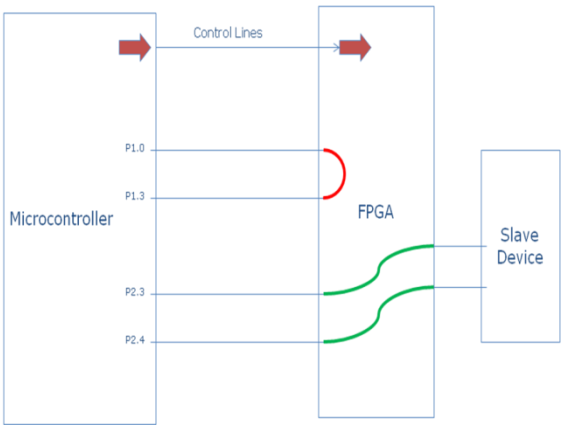


Fig. 1 : Communication between Microcontroller and FPGA. Microcontroller sends the control signals to connect ports through FPGA or to connect to Slave Devices

IV. DETAILED DESIGN

The design consists of following parts:

Hardware design:

Target Board: Microcontroller comes mounted on this board and all external peripheral IP's port pins are bought out on the Samtec connector pins.

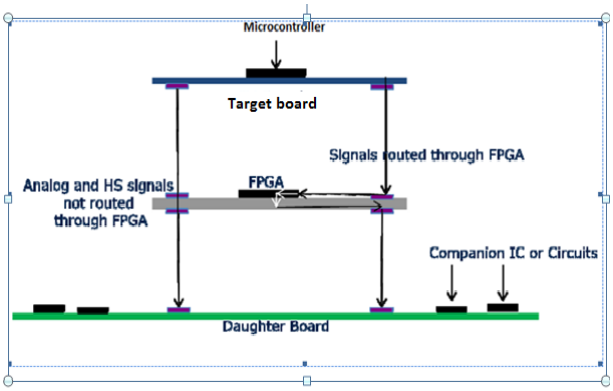
Daughter Board: A board with all slave devices, connectors of different form factors, level shifters etc. are mounted

FPGA Board: A board with FPGA mounted and all required Microcontroller signals and slave signals connected to it.

High–speed and Analog signals are bypassed for direct connection at the hardware schematic level.

Figure 2 describes the elevation view of hardware involved.

FPGA driver design: To control desired configuration and design and to change the configuration on-the-fly, Microcontroller needs to send command to FPGA. A piece of software driver is required to be designed, which will run at the Microcontroller side. The driver will export an API, sendFPGAcmd (char * module, int cfg), which will be called by the IP modules for sending command to FPGA. The API will consist of module no. and config no. as input parameter.



Hardware diagram of FPGA with other boards.

Microcontroller will send the command to FPGA by three GPIO lines. One of the two GPIO lines will send the data command to FPGA and the other line will send clock, which will synchronize the data. To synchronize in an efficient manner, we can take help of a third GPIO line, 'data_ready' line, which will indicate the valid data on the above-mentioned two GPIO lines. To synchronize the data and clock, we adopted the convention that, data will be sent at the middle of the –ve level of the clock. On the receiver side, FPGA will scan the data at the positive edge of the clock, provided 'data_ready' signal is active.

The overall interaction between Microcontroller and FPGA is explained in Figure 3. Validation testcase will have to call the API sendFPGAcmd (), which will send the control command to FPGA in encoded format. In turn, FPGA will decode the command and will make desired connection(s) as per the IP module and config no. combination.

We propose to send the data command to FPGA as encoded by One Hot encoding scheme. The advantage of using One Hot encoding is that only one '1' will be

sent at a time and hence RTL will be lighter at FPGA side during decoding the logic. As a consequence, the switching logic will also be faster (delay will be less). Also, memory consumed for logic design at FPGA side will be less.

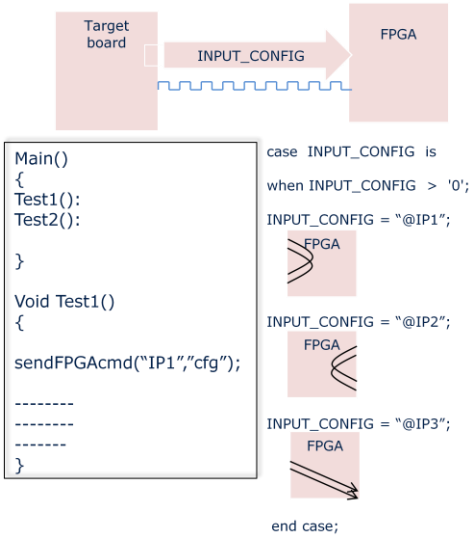


Fig. 3 : Validation testcase will call sendFPGAcmd() API, which will provide the input config to FPGA. Then FPGA will make desired connection as per the specific config..

HDL logic design at FPGA: This is the principal part of the work, since it will actually implement the logic at the FPGA side. Based on the command received from Microcontroller, the logic will decode the command and will perform desired connectivity and/or additional tasks.

Logic needs to be designed such that it should be capable to change the configuration on-the-fly. Design [3] should take care of the fact that- number of the level of look-up table to be minimum, in order to minimize the delay introduced by the FPGA.

We designed such that there will be logically two parts of the string command sent by the Microcontroller: module part and config part. Module part will consist of One-Hot encoded string, i.e., all the bits will be '0' except only one bit as '1' indicating the intended IP module. Similarly, Config part will be One-Hot encoded string, which will indicate which configuration is intended to trigger.

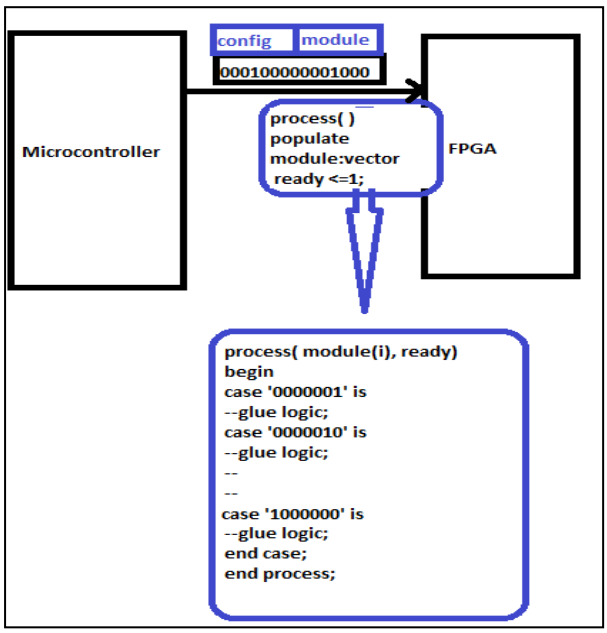
There will be one VHDL [3] 'process ()', corresponding to each module. Each process will be sensitive to the corresponding bit of the module. Within each such process, there will be multiplexing logic to select particular configuration. Config bits will be treated as selection criteria for multiplexers. Thus multilevel One Hot-encoded module and config parts can cater to select any configuration of any IP module.

There will be a master process, which will receive the command from the Microcontroller and populate a vector say,

receive :STD_LOGIC_VECTOR (n downto 0);

Initially, all the bits of 'receive' vector are '0'. Any change of a bit will trigger corresponding module-specific process(). Also, there will be a signal 'ready', which will be set by receiver process and will trigger the other processes as well. Figure 4 explains how Microcontroller command will be structured and the high-level design of the HDL logic.

Thus Microcontroller can control the FPGA to implement any predefined configuration. The glue-logic inside each configuration can serve the desired purpose of the configuration. It can be as simple as the connections of set of pair of port pins to get connected or it could be drive some slave devices, which are connected to FPGA by daughter card.



The high-level design of HDL logic at FPGA

There are couples of benefits of this design:

- We can switch to any predefined configuration on the fly.
- Validation engineer need not take care if the port-mappings, and hence less possibility of wrong connection due to human error.
- Microcontroller can directly control the FPGA by sending commands. No PC-based commands and hence manual intervention is minimum. It will be particularly more useful in case of regression. From validation point of view, we just need to call the function

sendFPGAcmd(char *module, int cfg) as per the testcase requirement. For regression, it will be series of call of the function with different parameters. A script can be easily written to the sequential call of that function and thus automation can be achieved.

V. RESULT

Finally, we completed the coding of FPGA driver part and tested the code with the hardware setup of FPGA board, placed in between Target board and Daughter board.

Following snapshot in Figure 5 shows actual hardware as a result of design of FPGA board.

We developed the VHDL code in and verified the RTL after synthesis using Xilinx [1] environment.

As the future scope of work, one can enhance the automation methodology by using the techniques of FPGA emulation. In this approach, the slave devices will be emulated in the FPGA itself along with their associated communication protocols.

VI. ACKNOWLEDGMENT

Thanks to Sadashivaiah Shivaprasad for his vision, guidance and overall management support for this project. Thanks to Ramasamy Subramanian and Ali Shaik Imtiyaj Mohamad for their technical guidance. Thanks to Kativarapu Srikanth and Umamaheswara Rao for hardware design. Thanks to Renukaswamy Pratap, Subhashini Revuri and Surendra Shamanna for their contribution in coding, debugging and testing. We extend our thanks to Hanumath Sastry and Basab Kundu for their valuable suggestions during review of this paper.

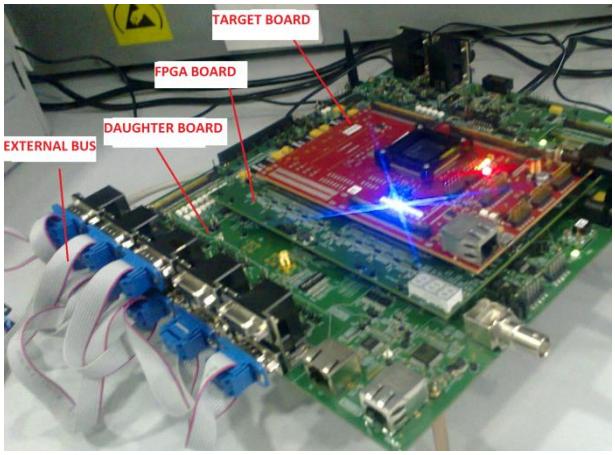


Fig. 5 : Custom hardware of FPGA board, placed in between Target board(microcontroller) and Daughter board.

VII. REFERENCES

[1] Xilinx Corp. www.xilinx.com

[2] Infineon Technologies www.infineon.com

[3] Doglous J Smith. "HDL Chip Design – A practical guide for designing, synthesizing and simulating ASICs and FPGAs using VHDL or Verilog", Doone publications, 1996.

◈◈◈