SAND2016-0687C

Extending the Constrained Random Simulation Methodology into Physical Device Verification for Processor-based ASICs

Anita L. Schreiber
High Integrity Software Systems, Dept.02622
Sandia National Laboratories
Albuquerque, New Mexico 87185 USA
alschre@sandia.gov

Melissa N. Wirtz
Embedded Systems/C-Series Modules
National Instruments
Austin, Texas 78759 USA
melissa.wirtz@ni.com

Abstract: *The coverage and efficiency of constrained random simulations for verifying an ASIC have long been recognized. However, this level of test coverage is often missing from tests of the actual device. A process to extend the test coverage provided in constrained random simulations to software-driven tests performed on actual devices in hardware systems will be discussed.*

Keywords: UVM; OS-VVM; Constrained Random Coverage; Intelligent Coverage; Processor-based ASIC; ASIC Verification

Introduction

It is often very challenging to provide a high level of coverage when verifying an ASIC with actual physical interfaces. When moving from simulation to real hardware, the reality of physics comes into play. Real device delays, board parasitics, routing delays, and clock skews are just a few of the complications of physical device verification versus simulation. Directed tests can verify functionality for a small set of parameters, but the use of directed testing for a wide variety of configurations can be time consuming and ineffective. Closing the gap between the level of verification that is obtained in a simulation environment and the level of verification that is achievable in physical hardware is a desired goal of ASIC and system designers.

One method of achieving this goal is possible with processor-based ASICs. By converting the random simulation parameters into software running on the processor within the ASIC to exercise the various IO interfaces, random and intelligent test coverage of the physical ASIC can be achieved. Though the process discussed in this paper utilizes Open Source VHDL Verification Methodology (OS-VVM) [1], an intelligent test bench methodology that allows mixing of "Intelligent Coverage" (coverage driven randomization) with directed, algorithmic, file based, and constrained random test approaches, the same process and principles can be applied using System Verilog and Unified Verification Methodology (UVM). The method described is tool independent. For simplicity, this paper will use the terms OS-VVM and Constrained Random Simulations to refer to these approaches. An overview of the process will first be described followed by an example implementation and results.

Block Diagram of the General Processing ASIC (GPA)

A simplified block diagram of the device verified through this process is shown in Figure 1.

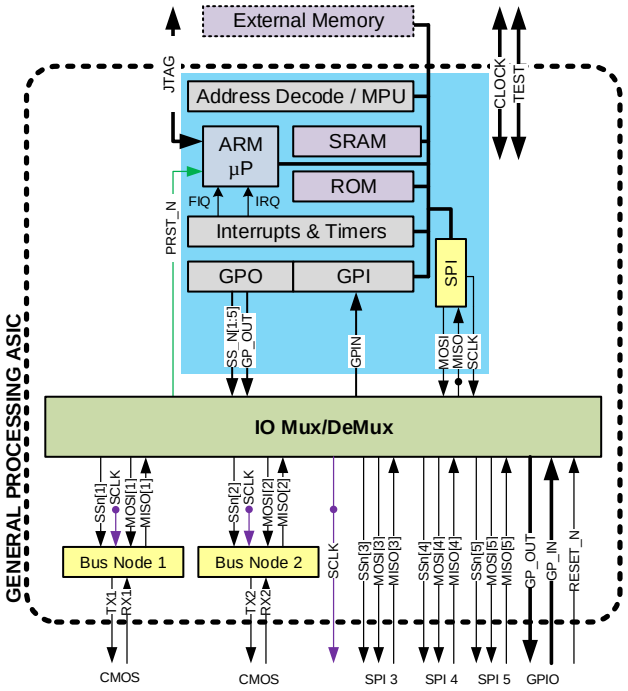

Figure 1: GPA Simplified Block Diagram

Built around an ARM processor, the GPA contains a processor subsystem consisting of internal memory and external memory interfaces with accesses controlled by an address decode / memory protection unit (MPU). In addition to interrupts and timers, general purpose inputs and outputs are available along with a Serial Peripheral Interface (SPI) master. The SPI master communicates with up to five SPI slaves; two are internal interfaces to the custom Bus Nodes within the ASIC while the remaining three are available to connect to external SPI slaves.

Process Overview

The process of extending the random simulations of a processor-based ASIC involves three main steps:

1. Set up / execute the constrained random simulation to get the randomized parameter sets for each interface to be tested.
2. Convert the resulting interface parameters to processor-accessible value sets.
3. Execute the randomized hardware tests.

Step 1: Set up and execute the constrained random simulation

Interface and parameter identification: To set up the constrained random simulation, the ASIC interfaces and software parameters for the interface drivers must first be identified along with the allowed ranges and bounds for these parameters.

For the GPA, the first interface to be tested was the SPI master through the external SPI ports. The GPA software driver for the SPI master allowed configuration of the following parameters with the constraints as shown in Table 1:

Table 1. SPI Parameters and Allowed Values

Parameter	Description	Range
Clock Divide	System Clock Divisor to generate SPI Clock (SCK)	8 - 65534
SPI Mode	Polarity (CPOL) and Phase (CPHA)	0 - 3
Length	SPI transaction length in bits	1 - 8192
Data	SPI data byte value	0 - 255

The processor interface to the two custom Bus Nodes of the GPA was also through the SPI master, however, with a limited range of system clock divisors and a fixed SPI mode. The testing of this interface therefore was built upon the work done for the SPI ports with the two Bus Nodes connected to each other external to the GPA. Each node was configured to either transmit the data or receive the data. For simplicity since both nodes had to be at the same clock frequency, the clock divisor was set at the minimum value to produce the maximum bus clock frequency as a worst-case condition. The parameters and the allowed range of values that were randomized are shown in Table 2.

Table 2. Bus Node Parameters and Allowed Values

Parameter	Range
Node TX	0 – transmit 1 – receive
Node data value	0 - 255
Transaction length (bits)	1 - 2032
Destination Port – Node 1	16 - 65535
Destination Port – Node 2	16 - 65535

After the parameters and legal ranges are identified, coverage bins for randomization of these parameters must be determined along with any cross coverage between bins that is required. Techniques for setting up the randomization and coverage parameters will not be discussed in this paper, the reader is encouraged to research this based on whether OS-VVM or UVM is being utilized.

Development of the RTL file for simulation: Next, the RTL file to be simulated is developed. In addition to containing the processes for randomizing the interface parameters, this RTL file contains a process to create a text file with each set of randomized values based off the values created by the coverage bins of the simulation. This process is called after each randomization cycle to output the values to the text file.

Note that it is not required, but very helpful, to include the RTL of the interface under test in the simulation so that the randomized output and transactions generated can be visualized in the resulting simulation waveforms. If the interface RTL is included, the text file creation process is called each time the RTL interface is provided randomized parameters.

For testing of the GPA SPI master, randomization bins were defined for the parameters in Table 1. Note that to insure all SPI modes were exercised at each clock frequency, the random SPI mode value was subsequently inverted until all combinations had been exercised. Likewise, the data value resulting from the randomization was inverted as many times as necessary to achieve the designated SPI length.

In addition to instantiating the RTL for the SPI master, a procedure was developed to output the resulting random values for each of the SPI parameters to a text file. Part of this file is shown in Table 3. 520 randomized value sets were created as a result of the simulation.

Table 3. SPI Random Values File

Clock Divide	SPI Mode	Length	Data
8286	11 00 01 10	7993	241 14
6320	10 01 00 11	5791	107
1284	01 10 11 00	794	69

Similarly, the RTL file for testing the GPA Bus Nodes was developed, using the same technique of inverting the random value for the Node TX values and for expanding the data. A snippet of the resulting text file is shown in

Table 4. 64 randomized value sets were created as a result of the simulation.

Table 4. Bus Node Random Values File

Node 1 TX	Node 2 TX	Length	Data	Dest. Port 1	Dest. Port 2
1 0	0 1	72	196	18155	17708
1 0	1 0	77	125	22166	22200
0 1	0 1	105	255	27910	28634

Step 2: Convert the resulting interface parameters to processor-accessible value sets

Now that randomized value sets have been created that produce the desired coverage across the interface parameters, the next step is to convert the values into data that can be accessed by test software running on the processor.

Using the memory map of the processor, a section of memory to contain these values is defined and a script is developed to write each value set into the defined memory region using the selected debugger memory write commands. Note that the procedure defined in the RTL could also directly write out the debugger memory write command script. However, the creation of a file that simply contains the resulting random values can be easily reviewed for coverage and allows flexibility to use different software debuggers in the future without having to re-run the simulation.

Test software to run on the ASIC processor must also be developed using the available software drivers for the interface under test. This software first reads the number of random value sets to be executed and then calls the interface software driver for each set of values, comparing the resulting data with the expected result.

For testing the GPA, Python 3 was selected as the scripting language to convert the SPI and Bus Node random value files to debugger memory write commands. The first command writes the number of random values sets to be executed and the consecutive memory write commands write the randomized value sets as shown in Figure 2.

```
memwrite 4 0x10011000 520
memwrite 4 0x10011004 8286
memwrite 4 0x10011008 1
memwrite 4 0x1001100c 1
memwrite 4 0x10011010 7993
memwrite 4 0x10011014 241
memwrite 4 0x10011018 8286
memwrite 4 0x1001101c 0
memwrite 4 0x10011020 0
```

Figure 2. Debugger Memory Write Command File

A test program was then developed for the ARM processor of the GPA to run both the randomized tests of the SPI master and the Bus Node ports. This purpose of this program was to read the randomized parameters from the designated memory locations, call the interface driver with the randomized values, and initiate the transfer. It also compared the resulting data with the expected data after each transaction and logged any errors.

Step 3: Execute the randomized hardware tests

The final step of the process is to execute the randomized hardware tests. The hardware containing the ASIC to be tested and all interface modules must be properly configured. If desired, any scopes and logic analyzers should be connected and set up at this time. The software debugger must also be properly set up and connected to the processor within the ASIC. Using the debugger command file, the processor memory is then initialized with the random value sets. At this point, the processor test software is downloaded to the processor and executed.

The testing of the SPI and Bus Nodes of the GPA was done using an FPGA-based development board that provided connections for daughter cards containing the GPA as shown in Figure 3. This test platform allowed early GPA prototypes to be tested throughout the entire GPA development cycle. Daughter cards containing the external memory needed for the GPA were provided and all of the interfaces of the GPA were brought out to connectors.

The SPI was tested by looping the MOSI output line to the MISO input line of the SPI connector. The Bus Nodes were tested by connecting the nodes to each other. This test platform not only verified the physical interfaces of the GPA but the memory interface timing as well.

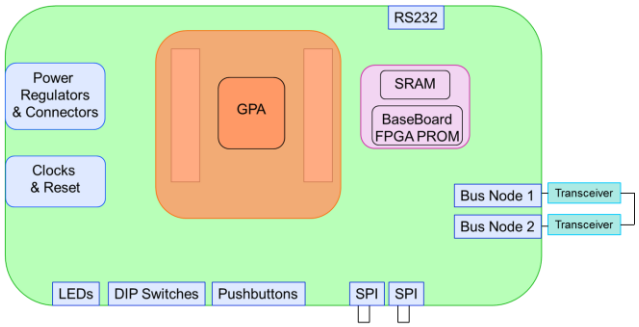

Figure 3. GPA Development Test Platform

A logic analyzer was connected to the SPI port to monitor the randomized transactions. During the test, it was visibly evident that the SPI modes as well as the SCK frequency were looping through the various randomized value sets. Figure 4 shows a small section of the resulting waveforms.

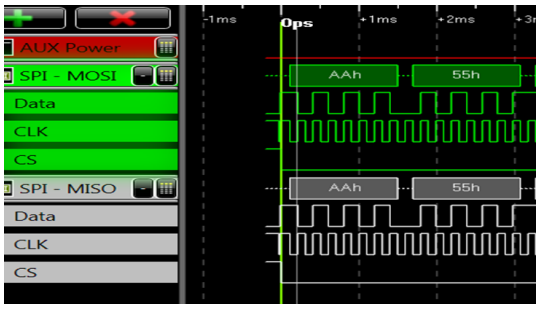

Figure 4. Logic Analyzer Output

Summary and Results

The process discussed in this paper is straightforward to implement without a large investment (depending on the tool suite used) and provides a more complete physical verification of a processor-based ASIC than achievable with directed tests. The GPA described in this paper is currently being used in three different embedded processing applications, each with unique interface requirements and has successfully passed all system tests to date.

Acknowledgements

The authors wish to thank Brent Meyer of Sandia National Laboratories for his initial work with OS-VVM which provided the basis for the GPA SPI and Bus Node random simulations.

References

1. Open Source VHDL Verification Methodology (OS-VVM).

   http://osvvm.org/