FPGA Based SHAKTI C-Class Verification

Omkar Bhilare
Dep. of Electronics Eng.
Veermata Jijabai Technological Institute
omkarbhilare45@gmail.com

Lavanya J
Dep. of Computer Science and Eng.
Indian Institute of Technology Madras
lavanya.jagan@gmail.com

V. Kamakoti
Dep. of Computer Science and Eng.
Indian Institute of Technology Madras
kama@cse.iitm.ac.in

***Abstract*—C-Class is a member of the SHAKTI family of processors. It is an extremely configurable and commercial-grade processor class. VAJRA is a SOC based on a 64-bit C-class 6 stage in order core supporting RV64IMACSU extension. Verification is a very important process in chip design. Usually, the software RTL simulation tests on host computers take a very large amount of time. This paper explores the possibility of FPGA-based verification of SHAKTI processors to reduce verification time. AAPG is a tool that is intended to generate random RISC-V programs to test RISC-V cores. In this paper, we ran AAPG tests on FPGA and compared the spike's golden signature dump with the FPGA signature dump. This paper also explores the self checking tests generated by AAPG. Self-checking tests have the advantage of running on FPGA or silicon without much intervention from the host, thereby accelerating the speed verification significantly.**

***Index Terms*—C-Class, FPGA, RTL, AAPG, RV64IMACSU, ISA, Verification**

I. Introduction

Verification is the most crucial aspect of ASIC design. Modern ASIC chips are very complex and contain millions of transistors, which increases the chances of getting an error in the chip during the design process. The goal of ASIC verification is to make ASIC bug-free and to meet design requirements and specifications. ASIC Verification process is one of the crucial parts of the ASIC design process. It consumes as much as 70-80% of the total ASIC design time. Since modern ASICs are getting complex and complex, it is necessary to accelerate the verification flow so that more ASICs can be taped out in a shorter duration.

The use of FPGAs in the verification flow has been increasing in recent years. To understand how FPGAs assist in the verification flow one should understand the other industry-standard verification methodologies. The industry uses majorly four kinds of verification technologies, which are as follows:

- **Dynamic Verification:**
  Dynamic Verifications are based on Software Simulation. The simulation-based verification testbenches use HVLs(Hardware Verification Languages) like System Verilog, Python to verify the RTL Designs. Constrained-random, Assertions, Code, and Functional Coverage are the most popular techniques which are based on dynamic verification technology.
  Code coverage is a measure of how extensive the testbench is written. Functional Coverage is a verification approach that has gained popularity in recent years. Functional Coverage is the determination of how much functionality of a design has been verified by the Testbench. Most of the time design is subjected to input which is constrained random. Assertions are used to Flag the error if occurred.
  The main advantage of Dynamic Verification is the high control and visibility in the flow. The bugs can be found in the RTL code with the help of assertion and waveform analysis. The disadvantage of Dynamic Verification is the slow speed of software simulation.
- **Static Verification:**
  Static verification is the process of verifying the RTL design against some user predefined rules without executing the design. In the early stages of ASIC, formal verification is used to find bugs in the RTL design.
  Formal Verification methods are efficient at early design stages, but they are difficult to set up and proper experience, EDA tools are required for the same. At the later stages of ASIC design, dynamic verification is usually used.
- **Hardware Emulation Engines:**
  It takes billions of verification cycles to boot an operating system and execute software applications. The high throughput required for nowadays verification process can only be achieved using specialized hardware engines or FPGA prototyping. The Hardware emulation engines like Cadence Palladium Z1 can be used for faster hardware and software integration.
  The Advantage of Hardware emulation engines is their speed. But its major disadvantage is the cost of the actual device.
- **FPGA Prototyping:**
  FPGAs are the cheap alternatives to Hardware Emulation Engines. In the last decade or so there are lots of improvements happened in the FPGA market. The FPGAs are much cheaper and state of art FPGA toolchains are very easy to use. With the rise in the complexity of ASICs and the increasing demand to shorten the time to the market of a chip, FPGA prototyping remains a key solution.
  The major advantage of FPGA prototyping is the fast speed and lost cost. But the disadvantage of FPGA-based verification is the limited control and visibility in the flow. In this paper, we worked on developing a new FPGA framework that has FPGAs speed advantages but as well

Comparision of RTL Verification Approach				
RTL Verification Approach	**Speed**	**Controllability**	**Visiblity**	**Cost**
Dyamic and Static Verification	Very Slow	High	Full	Low
Hardware Emulation Engines	Very Fast	High	Full	Very High
FPGA Prototyping	Fast	Low	Low	Low
SHAKTI FPGA Framework	Fast	moderate	moderate	Low

TABLE I: Comparision of RTL Verification Approach

proper control and visibility are ensured in the flow with the help of checksums, python wrapper, etc.

II. SHAKTI FPGA FRAMEWORK

SHAKTI is an open-source initiative by RISE group at IIT-Madras with the aim to produce production grade processors, complete System on Chips (SoCs), development boards and SHAKTI-based software platform

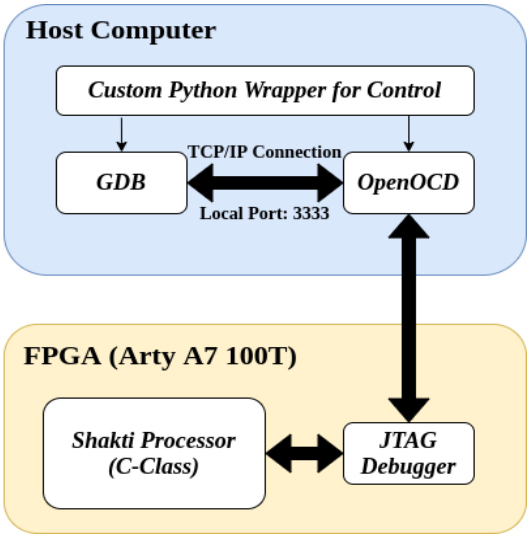
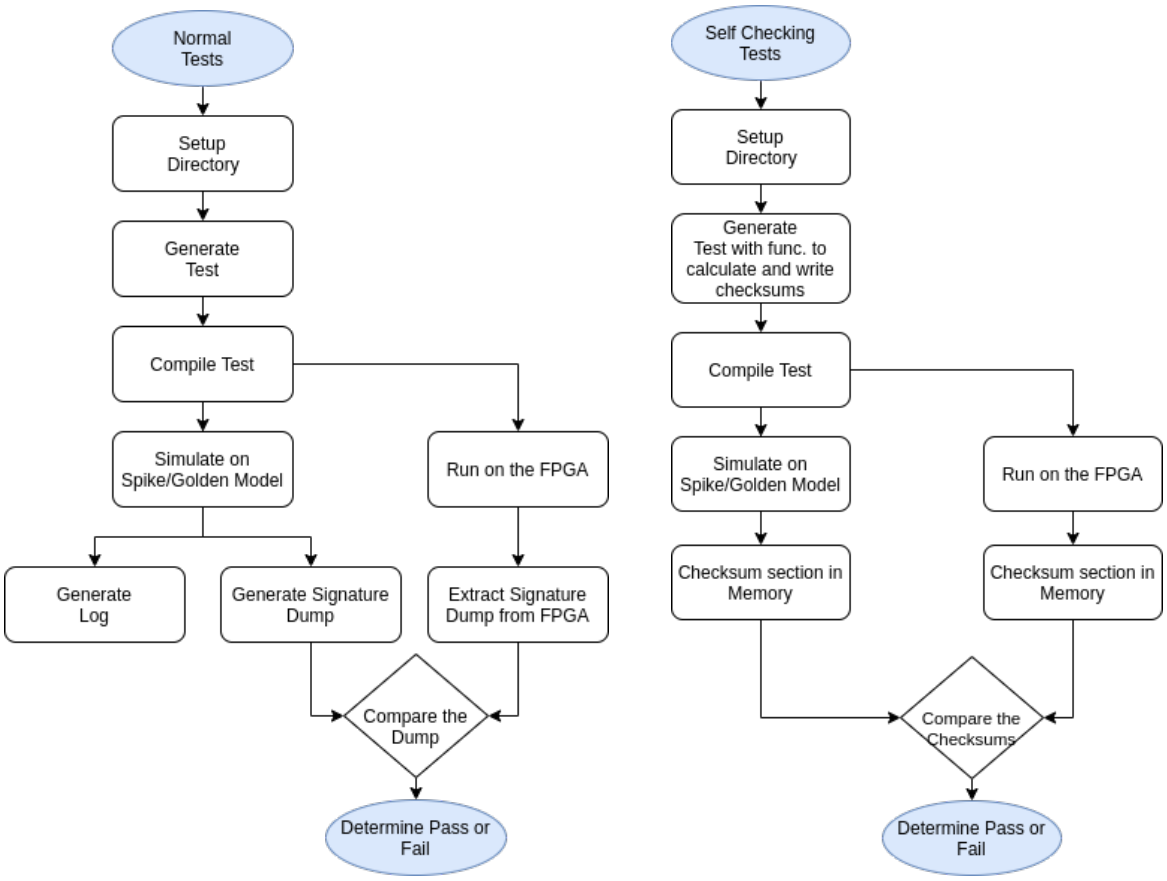


Fig. 1: Shakti FPGA framework

The SHAKTI FPGA framework consists of following things:

1) **RISCV GDB:**
RISCV GDB is the GNU debugger for the RISC-V platforms. RISC-V GDB is a debugger that lets you control the flow of assembly code which is running on the FPGA.
In this framework, we have used the following GDB features:
   a) *file Code.riscv*: Selects the binary of code which needs to be loaded into FPGA.
   b) *load*: Loads the all sections from binary to the FPGA.
   c) *break function-name*: Used for creating breakpoints in the ASM code.
   d) *dump binary memory mem.bin start end*: Used for Dumping the FPGA signature to the binary file stored in host computer.
2) **OpenOCD:**
Open On-Chip Debugger (OpenOCD) provides an interface for the RISCV-GDB to connect to the target device (Arty A7-100T). It allows the RISC-V GDB (gdbserver) to connect to the target processor which is running on the FPGA.
3) **JTAG Debugger:**
An additional hardware device is required to allow the Host Computer to communicate and introspect the target hardware. These devices are often referred to as Debugger devices or Debug adapters. The compiled program is transferred to the target hardware using the Debugger device. The FPGA device that we have used has on board JTAG debugger.
4) **Automated Assembly Program Generator (AAPG):**
Arbitrary Assembly Program Generator (AAPG), is a code level verification tool built in-house as part of the Shakti Program. It generates constrained random tests to verify RISC-V cores. It can generate normal and self-checking tests. In normal tests, the general memory range dump are compared between golden signature and DUT(Device Under Test).
Self Checking tests are a feature in AAPG that setup an assembly test with checksums placed at regular intervals throughout the test, and the values of these checksums are provided in a designated reference data section. The concept of Self Checking tests arose for verification at the Field Programmable Gate Array (FPGA) level. Currently, one can create and use a self-checking test, as a single entity/program to be run in the environment needing verification and identify if the test is passing or failing. [2]
5) **AAPG On FPGA:** Custom Python Wrapper
AAPG On FPGA (AOF) is a python wrapper that is used on HOST Computer to generate tests and compare the golden and FPGA dump.
The features of AOF as follows:
   a) *Extraction and Comparison of FPGA and Golden Signature(spike) Dumps:*
   The Python Wrapper automatically runs AAPGTests on the FPGA and afterward extracts the FPGA Signature into the host computer with the help of GDB. It also compares the FPGA and golden signature Dump.
   b) *Multiple Tests Support*:
   The Python wrapper can also automatically run an N number of tests one after another on the FPGA. The Steps followed by wrapper for multiple tests are as follows:
      i) *python signature.py -tests=N* : Following com-



(a) Normal AAPG Tests on the FPGA

(b) Self-Checking AAPG Tests on the FPGA

Fig. 2: FPGA Test Flow

mand runs N number of tests on the FPGA one after other.

ii) The first wrapper sets up the test number 0, runs the AAPG test on FPGA, and extracts the Dump from FPGA. Afterward, compare the FPGA and golden Signature dump.

iii) If both dumps are the same, then the wrapper deletes the last test folder and set up the next test.

iv) This is done until N-1 Tests. If any tests are fail then they will be automatically stored in folder named *fail tests-number*for further debugging.

c) *Increases Visibility of the FPGA*:
The main drawback of FPGA-based prototyping for debugging and verification of processors is the less visibility and control in the FPGA flow. AAPG has a feature called Self-Checking Tests. In these tests at regular intervals, the checksums of CPU Status are stored in the memory of the CPU at a particular location. This set of checksums is nothing but a signature. The golden signature (set of checksums) generated by spike will be always true. The python wrapper compares the golden signature generated by the spike to the signature of the FPGA.
The python wrapper can detect exactly where is the mismatch in the checksum and accordingly find the group of instructions causing issue resulting in more visibility and control in the verification flow.

III. AAPG Tests on FPGA

1) **Normal AAPG Tests on the FPGAs:**
AAPG is used to generate constrained random RISC-V assembly tests. After the tests are generated AAPG also compiles it on the Spike and generates a golden signature dump from it. Afterward, the Python wrapper runs the same constrained random RISC-V assembly test on the FPGA and obtains a memory dump from the FPGA. Both the memory dumps are compared and if found any mismatch python wrapper points to the exact memory address.

2) **Self-Checking AAPG Tests on the FPGAs:**
The config file of AAPG has a section labelled self-checking, which contains:

a) **Rate**: This takes an integer in the range (1,infinity) as input. It controls the number of instructions

between two check sums. If rate is 10, then a checksum will be added every 10 instructions. [2]

In Self-Checking AAPG tests, these sets of checksums are embedded in a particular part of memory. First, the test program is run on the golden spike model from which we get the golden signature dump. Then python wrapper runs these Self-Checking AAPG tests on the FPGA and extracts the set of checksums from the memory region and compares them with the reference golden dump. Depending on the comparison test result is decided.

IV. Results

In this paper, we have shown how FPGAs can accelerate the traditional Verification flow.

Targeted SOC: We have tested the FPGA framework on Vajra SOC. VAJRA(C64-A100) [3] is an SoC built around C-class. This SoC is a single-chip 64-bit C-class micro controller with 4KB of ROM and 256MB DDR3 RAM.

Software Golden Model: We have used Spike [4], the RISC-V ISA [5] Simulator to obtain golden signature dump.

FPGA Board: The Vajra SOC was emulated on Arty A7-100 FPGA. Arty is a ready-to-use development platform designed around the Artix-7™ Field Programmable Gate Array (FPGA) from Xilinx.

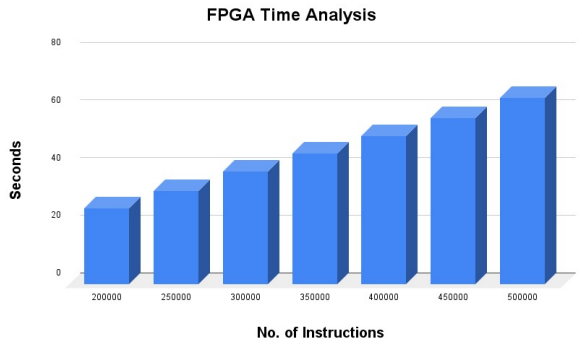


Fig. 3: Instruction Test

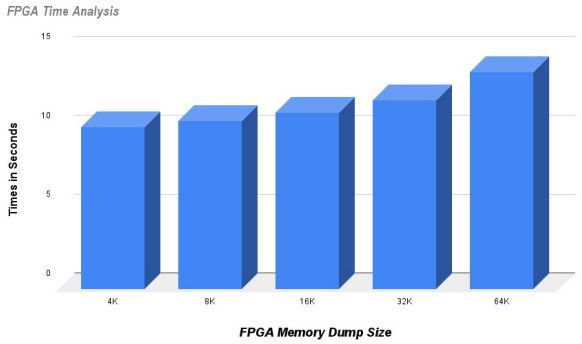


Fig. 4: Memory Dump test

These tests were run on Shakti's Vajra SOC. The Major Advantage of FPGA in verification flow is Speed. As shown in Figure 3, 5,00,000 RISC-V Instruction took only 64 seconds to run on the FPGA.

Visibility and control are required in any Verification Framework. Traditional FPGA flow has usually less visibility and control but with the help of checksums in AAPG and python wrapper, we can see where is the mismatch.

V. Future Improvements

Instead of using separate host machines and FPGA, in the future, we are planning to explore Hybrid architectures provided by Xilinx such as Zynq. The Zynq-7000 SoC family integrates the software programmability of an ARM-based processor with the hardware programmability of an FPGA.

In the future, we would also like to explore cloud-based options provided by amazon like the Amazon F1 instance for the framework.

VI. Conclusion

SHAKTI FPGA framework is a completely automatic system to generate single or multiple tests suitable to run on FPGAs. It also automatically compares golden signature and FPGA dump. In case of a mismatch, it also gives the exact address for further debugging. It increased the verification flow speed as well as maintained control and visibility in the flow.

References

[1] D. Kim, C. Celio, S. Karandikar, D. Biancolin, J. Bachrach and K. Asanović, "DESSERT: Debugging RTL Effectively with State Snapshotting for Error Replays across Trillions of Cycles," 2018 28th International Conference on Field Programmable Logic and Applications (FPL), 2018, pp. 76-764, doi: 10.1109/FPL.2018.00021.

[2] ABISHEK TAIKAD SHYAMSUNDER, "SELF CHECKING TESTS FROM AAPG", 2019, SHAKTI, IITM.

[3] SHAKTI, IITM, 2020, VAJRA gitlab.com/shaktiproject/sp2020/c64-a100

[4] RISC-V foundation, 2020, RISC-V Spike, https://github.com/riscv-software-src/riscv-isa-sim

[5] "The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 20191213", Editors Andrew Waterman and Krste Asanovi´c, RISC-V Foundation, December 2019