No gain without RISC- A novel approach for accelerating and streamlining RISC-V Processor functional verification with register change dump methodology

Damandeep Singh Saini, Chethan Ammanahalli Siddaramiah, Anil Deshpande, Raviteja Gopagiri, Somasunder Kattepura Sreenath, Niharika Sachdeva
Samsung Semiconductor India Research (SSIR)
Bagmane Goldstone Building, Mahadevpura
Bangalore – 560037

Abstract- **Welcome to the era of intelligence! The invention of a transistor was undoubtedly remarkable which eventually led to the most sophisticated, complex and intellectual piece of engineering marvel - The Processors With the advent of AI/ML and 5G, there's no doubt that Processors, which are the main Computational power engines of today's era, to become super main stream in bringing intelligence breakthrough in Al, a big-time reality. We understand how a small - scale start-up to Multi-Billion Dollar Technology companies would be eager to board the wave of AI for which the basic pre-requisite would be a "PROCESSOR". Therefore, we see an exponential growth in the domain of Open-Source "RISC-V" Processors now-a-days, in fact IEEE predicts 73% of processors will be RISC-V by 2027. Hence forth, one of the biggest challenges for any company using RISC-V would be on how they verify the Processor with their incremental design (hardware) changes with minimal cost expense and maximum reliability. So, in this paper, we propose a reliable, proven and a cost-effective strategy of making the verification of RISC-V processor streamline using Register Change Dump (RCD) and Silicon proven Test vectors (Binaries) alone. This process of standardizing processor verification would be a game changer in Time-to-Market and Cost efficiency across the companies.**

Keywords— RISC-V, Processors, open-source, AI/ML, 5G, IoT, UVM, Verification

I. Introduction

Till now, Processor related verification has been well-renowned and been a proven pathway for almost all the Technology companies since decades. This traditional approach of processor verification involves paying a hefty royalty to the IP patented processor technology like ARM, along with additional cost for support, simulator and development trades (Fig I). Not to forget additional bucks needed for 3rd Party EDA development and performance tools with licensing fees for the generation of mere crypto (encrypted) RTL (register-transfer-level) code. To catch-up with the revolutionary period of AI/ML, IOT (Internet of Things), 5G and be the first to capitalize this market, there's a definite need for an open-source processor "development" and "verification". The novelty this paper brings on the table for the verification part where we recommend strategies and share results to address the above-mentioned bottleneck and help the community to lead with innovation rather than bounded by economic factors.

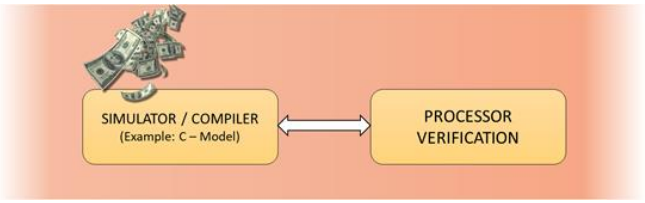


Fig I

To begin with our recommendation, there are essentially four components needed to do the End-to-End Processor verification of RISC-V:

1) The Instruction Binary / Test Vector (ROM Code)
2) A file named as RCD (Register Change Dump)
3) The RISC-V RTL Code (DUT: Device Under Test)
4) UVM supported script for converting RCD to RISC-V Register Dumps (For Scoreboarding checks)

A) *ROM Code (Instruction Binary Test Vector):*

Also sometimes referred as BOOT Code, it's essentially a simple text file holding the instructions derived from ISA (instruction Set Architecture) in Binary / Hex format. As per the general computer hardware architecture, the processors are connected to a memory either directly or via High-Level Caches and are expected to keep fetching instructions from the memory in a particular defined order. This binary then basically guides what the processor is expected to execute based on the application. The ROM Code can be generated by open-source compilers.

B) *.RCD (Register Change Dump):*

This is a generalized industry standard file with a fixed format holding the information of any change in Processor's Register values, along with information of any external data transfer with the memory or Cache. It is essentially an alternate to Reference Model / Simulator and can be generated by either directly simulating the RTL code and enabling the Dump vector options in the code or, it can be derived from RISC-V open-source simulators after feeding the desired binary to the simulator.

C) *RISC-V RTL Code:*

It is essentially an open-source RTL (Verilog) code for RISC-V Processors. Based on the configuration, there can be different flavors of RISC-V processors catering to dedicated needs of the end customer.

D) *UVM Script (.RCD to RISC-V Register Dumps):*

This is a simple System-Verilog or any other UVM supported DPI based script which converts the .RCD information to RISC-V register models which could then be used in scoreboard for comparison with the actual RTL simulation.

II. Related Work

So far, we understood the pre-requisite components of our proposal. Herein, we share the detailed novel idea of its implementation.

Going forward, we recommend that with every release of RISC-V processor RTL code, a corresponding set of pre-verified Test Vectors (Binary) be released along with is .RCD file by the concerned RISC-V committee. Now these components act as a Golden Reference model/file for the verification purpose. So now even if any RISC - V developer performs incremental changes in the RTL code, say to either enhance the clock speed by optimizing the critical paths or introduce tightly coupled buffers for improving performance, or say even changes it to Out-of-order from in-order pipelining, the ultimate output of the Processor would still be the same and should be un-affected by these incremental changes. Hence, these 2 components (Binary and. RCD) can still act as valid reference models for processor verification. This concept provides an excellent balance between the design flexibility and cost optimization for the IP development.

Once we have the. RCD and Binary file(s) for the corresponding RISC-V RTL code, we can initiate the UVM based RTL simulations to do the Register change comparison using the component (D) mentioned above. The idea is whenever there's a change in RISC-V Internal Register value, this same changed value should be compared to the expected value from the .RCD file. Clearly, this novel idea shared here is deliberately chosen to be Non-Cycle-

Accurate to enable us for playing around with memory latencies which usually forms a corner-case bug in most of the designs due to unexpected stalls in the pipeline. Hence, using this concept we indeed don't limit or resist the developer from generating artificial stalls or Hazard scenarios related to memory. This proposed verification flow is sufficient enough to find any bugs related to these scenarios.

To summarize, the overall idea is to maximize the creativity and minimize the operational costs and time-to-market by enabling developers close the design loop with minimal effort. Few additional remarkable points to note are as below.

a) Since the Test Vectors would be tied to a particular RISC-V configuration, hence, the code closure for the design can be almost achieved by simply running this suite of open-source test vectors.
b) There are many industries specific Test Vectors (Binary code) which are Self-Testing. Hence, for such scenarios, even .RCD can be opted out.
c) With this verification proposal, one can use the information from .RCD file to generate “Fault Attack Errors" in the RISC-V internal registers during the RTL simulation itself. This could potentially deplete the need of using highly expensive EDA tools required for Fault Attack tests.
d) With this Open-Source community development proposal, the implementation of any novel Scenario (test Vector) by a developer could then be verified by any other developer working on the same configuration of RISC-V Processor. This would definitely lead to a more Robust Verification framework across the globe.

One of the key ground-breaking implementation of this paper is the introduction of “Auto-Restoring” feature wherein, the RISC-V processor can be initialized from anywhere in-between the programming sequence by defining just the “CLOCK” edge number from the .RCD file and then, the UVM testbench initializes all the RISC-V processor’s internal registers upto the point of the above mentioned CLOCK edge number, by auto generating the instructions needed to configure internal registers and continues executing the ROM Code from the last CLOCK edge Program Counter (PC) Value. This technique not only enables us to cut-short the simulation time for faster Code closure but also helps in testing just the RTL bug fix right at the expected ROM code execution instead of running the entire Binary again. More details to be shared in the IMPLEMENTATION Section.

III. IMPLEMENTATION

The following figure (Fig II) shows the recommended implementation of the overall RISC-V open source verification framework where for example, the RISC-V open-source repository is holding groups of different flavors of RISC-V configuration and once the developer shortlists the required configuration (configuration 3 in this case), the checkout files would then cross the boundary and be used for only UVM based RTL simulations.

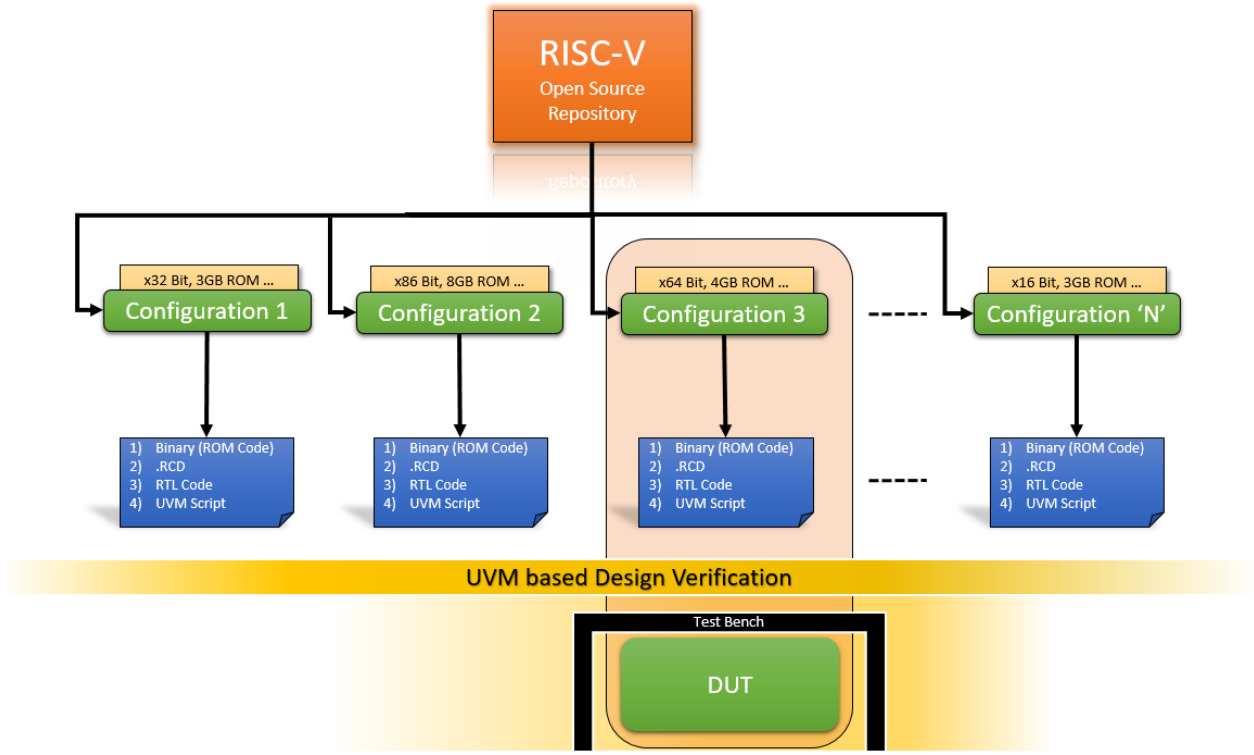
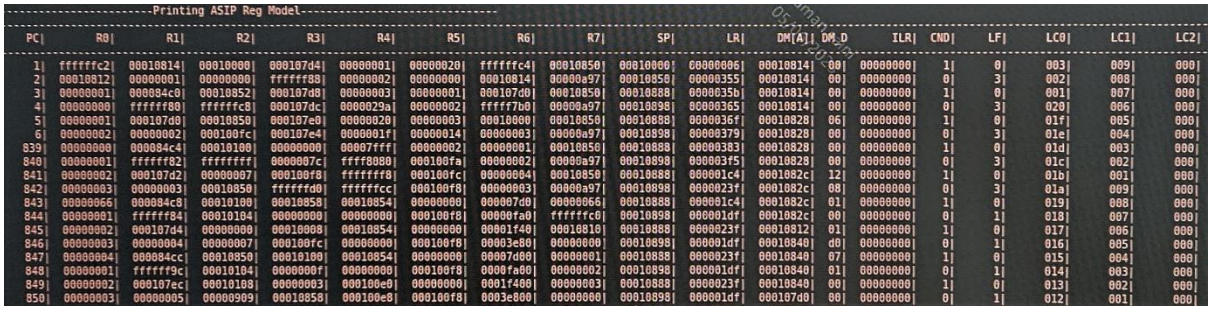


Fig II.

Once the Testbench is stitched across the modified RISC-V RTL (DUT), the verification team can use the UVM based script to decode the .RCD file and generate a more readable RISC- V internal register value dumps along with scoreboard variables which could then be compared to the RTL simulations for any potential bug.

The following figure (Fig III) shows the output of "UVM script" whose input is the corresponding .RCD file from the particular configuration.



```
----------Printing ASIP Reg Model----------
| PC| R0| R1| R2| R3| R4| R5| R6| R7| SP| LR| DM[A]| DM_D| ILR| CND| LF| LC0| LC1| LC2|
|  1| ffffffc2| 00010814| 00010000| 000107d4| 00000001| 00000020| ffffffc4| 00010850| 00010000| 00000006| 00010814| 00| 00000000| 1| 0| 003| 009| 000|
|  2| 00010812| 00000001| 00000000| ffffff88| 00000002| 00000000| 00010814| 00000a97| 00010850| 00000355| 00010814| 00| 00000000| 0| 3| 002| 008| 000|
|  3| 00000001| 000084c0| 00010852| 000107d8| 00000003| 00000001| 000107d0| 00010850| 00010888| 0000035b| 00010814| 00| 00000000| 1| 0| 001| 007| 000|
|  4| 00000000| ffffff80| ffffffc8| 000107dc| 0000029a| 00000002| fffff7b0| 00000a97| 00010898| 00000365| 00010814| 00| 00000000| 0| 3| 020| 006| 000|
|  5| 00000001| 000107d0| 00010850| 000107e0| 00000020| 00000003| 00010000| 00010850| 00010888| 0000036f| 00010828| 06| 00000000| 1| 0| 01f| 005| 000|
|  6| 00000002| 00000002| 000100fc| 000107e4| 0000001f| 00000014| 00000003| 00000a97| 00010898| 00000379| 00010828| 00| 00000000| 0| 3| 01e| 004| 000|
| 839| 00000000| 000084c4| 00010100| 00000000| 00007fff| 00000002| 00000001| 00010850| 00010888| 00000383| 00010828| 00| 00000000| 1| 0| 01d| 003| 000|
| 840| 00000001| ffffff82| ffffffff| 0000007c| ffff8080| 000100fa| 00000002| 00000a97| 00010898| 000003f5| 00010828| 00| 00000000| 0| 3| 01c| 002| 000|
| 841| 00000002| 000107d2| 00000007| 000100f8| fffffff8| 000100fc| 00000004| 00010850| 00010888| 000001c4| 0001082c| 12| 00000000| 1| 0| 01b| 001| 000|
| 842| 00000003| 00000003| 00010850| fffffd0| ffffffcc| 000100f8| 00000003| 00000a97| 00010898| 0000023f| 0001082c| 08| 00000000| 0| 3| 01a| 009| 000|
| 843| 00000066| 000084c8| 00010100| 00010858| 00010854| 00000000| 000007d0| 00000066| 00010888| 000001c4| 0001082c| 01| 00000000| 1| 0| 019| 008| 000|
| 844| 00000001| ffffff84| 00010104| 00000000| 00000000| 000100f8| 00000fa0| ffffffc6| 00010898| 000001df| 0001082c| 00| 00000000| 0| 1| 018| 007| 000|
| 845| 00000002| 000107d4| 00000000| 00010008| 00010854| 00000000| 00001f40| 00010810| 00010888| 0000023f| 00010812| 01| 00000000| 1| 0| 017| 006| 000|
| 846| 00000003| 00000004| 00000007| 000100fc| 00000000| 000100f8| 00003e80| 00000000| 00010898| 000001df| 00010840| d0| 00000000| 0| 1| 016| 005| 000|
| 847| 00000004| 000084cc| 00010850| 00010100| 00010854| 00000000| 00007d00| 00000001| 00010888| 0000023f| 00010840| 07| 00000000| 1| 0| 015| 004| 000|
| 848| 00000001| ffffff9c| 00010104| 0000000f| 00000000| 000100f8| 0000fa00| 00000002| 00010898| 000001df| 00010840| 01| 00000000| 0| 1| 014| 003| 000|
| 849| 00000002| 000107ec| 00010108| 00000003| 000100e0| 00000000| 0001f400| 00000003| 00010888| 0000023f| 00010840| 00| 00000000| 1| 0| 013| 002| 000|
| 850| 00000003| 00000005| 00000909| 00010858| 000100e8| 000100f8| 0003e800| 00000000| 00010898| 000001df| 000107d0| 00| 00000000| 0| 1| 012| 001| 000|
```

Fig III.

As shown in the figure III, the UVM script automatically generates RISC-V Register modelling information in a more user-friendly and readable format based on the .RCD file. The verification team can then decide on inserting assertions or standalone checkers for comparing the expected value from the script with the change in RTL register value. The effectiveness of this technique lies on the fact that the comparison happens in real-time for any change in RTL register value, hence any issue / bug identified, as in the mismatch between the "expected" and "observed" values, the user doesn't have to wait till the End of the simulation. The moment, any mismatch is reported in the simulation

during the RTL execution, the user can decide to either kill the test and debug it or ignore it based on the severity and design limitations.

Adding to the discussion on "Auto-Restore" feature, the following Fig IV illustrates the overall concept dealing with the restoring of RISC-V processor internal state to resume the execution of the ROM code from anywhere in-between of the simulation.

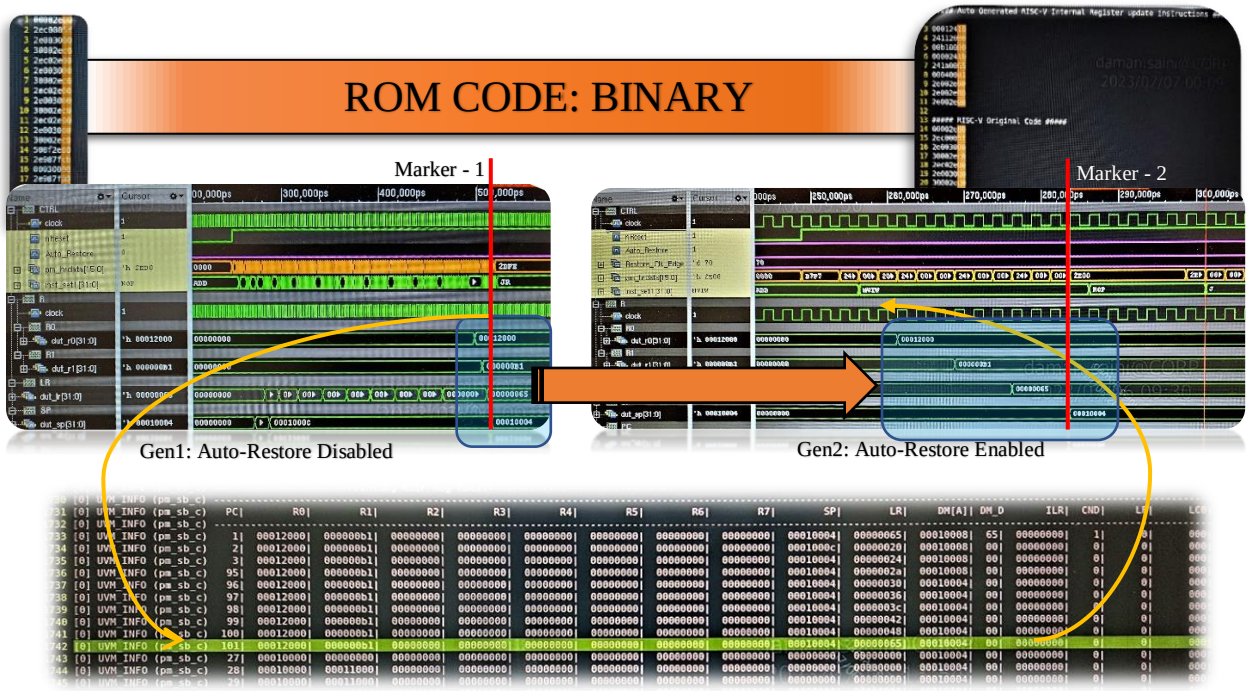


Fig IV.

Consider the "Gen1" scenario where a regular Boot Code is running post Reset de-assertion and the Processor mis-functions somewhere at the highlighted Red "Marker – 1". After fixing the RTL / TB issue, the user can simply enable "Auto_Restore" option in their testbench and mention the "Restore_Clk_Edge" number using the .RCD file from where the execution is expected to resume from. Using the above mentioned "UVM Script", the TB will calculate the expected values of all the internal registers / states of the RISC-V Processor till the "Restore_Clk_Edge" value and will also automatically generate few MVIW (Move Immediate Word) instructions to configure the RISC-V processor with the last instruction being that of updating PC value. Once these newly generated instructions are executed and the last PC update instruction is fetched in the pipeline, the memory is re-programmed using back-door entry to return to the original Binary ROM code. The same is depicted in Red colored "MARKER – 2" where once all the auto generated instructions are executed, the processor makes a JUMP to the last PC value of Marker – 1, thus continuing the code execution from where it was restored.

IV. RESULTS

As a general fact, two individuals working on processor verification were asked to report the figures based on the following parameters. One of them was asked to follow this paper's proposed framework for verification versus another person who continued verifying the processor with "traditional Approach" by developing / updating the RTL simulator and comparing the final results. The outcome was discussed and documented in the following table (Table I) based on various parameters.

Parameter	# of days (Traditional approach)	# of days (Proposed Framework)	Improvement (%)
Getting hands on Processor RTL Code	7	1	85.7
Getting the stuff ready to hand over all the required files to DV team for verification	10	3	70
Time taken to debug and close issues	28	15	46.4
Time for Code Coverage Closure	6	2	66.7

Table I

Clearly, there's been a significant improvement in Time-to-Market with this proposed framework. The results related to costing was undoubtably best in this new framework approach. Due to confidentiality, the costing data will NOT be shared here.

V. Conclusion & Future Scope

The impact of using this proposed verification methodology would be as follows.

1) Its stands unbiased for providing equal platform for any small scale startup to a multi billion dollar company for competing in the international market for upcoming technology trend.

2) The basic ecosystem needed for any processor development and verification is greatly reduced leading to rapid adoption and TTM (Time to Market).

3) Being an open source, the development rate could be exponential and any collaboration on world-wide would be a win situation between two or more than two parties.

4) For future scope:

a) We intend to design a workflow for measuring dynamic power consumption directly using .RCD file.

b) To extend the verification concept on RISC-V for multi processor architecture.

VI. REFERENCES

[1] Getting started with RISC-V verification (RISC – V): https://riscv.org/blog/2020/05/getting-started-with-risc-v-verification

[2] Abdelfattah Munir; Mina Magdy; Samer Ahmed; Sherouk Nasr; Sameh El-Ashry; Ahmed Shalaby. "Fast Reliable Verification Methodology for RISC – V Without a Reference model", IEEE 2018.

[3] EEWeb, RISC -V's Verification Challenge: https://www.eeweb.com/risc-vs-cpu-verification-challenge

[4] Jihye Lee; Whijin Kim; Sohyeon Kim; Ji-Hoon Kim, "Post-Quantum Cryptography Coprocessor for RISC-V CPU Core", IEEE 2022

[5] RISC-V Foundation. (2017, July) RISC-V Specifications [Online].: https://riscv.org/ specifications

[6] Mike Bartley, Lavanya Jagan, G S Madhusudan & Neel Gala, "RISC-V Design Verification Strategy" , Verification Horizons

[7] Dake Liu, "ASIP (Application Specific Instruction-set Processors) design", 2009 IEEE 8th International Conference on ASIC