PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)

Electronic City Campus, Bengaluru – 560 100, Karnataka, India

Report on

"Design of Decode-BOX and Verification of (Data and Instruction) Cache, Register File, Bypass and RISC-V Bit-Manipulation Hardware IPs"

Submitted by

Karthik B K (PES2201800185)

January - April 2022

Under the guidance of

Neel Gala

Dr. Neel Gala

Co-Founder / CTO
InCore Semiconductors
Pvt. Ltd.

Dr. Madhura Purnaprajna

Professor, Department of ECE
PES University EC Campus

Prof. Vinay Reddy

Assistant Professor,
Department of ECE
PES University EC Campus

FACULTY OF ENGINEERING
DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

BACHELOR OF TECHNOLOGY

CERTIFICATE

This is to certify that the report titled

“Design of Decode-BOX and Verification of (Data and Instruction) Cache, Register File, Bypass and RISC-V Bit-Manipulation Hardware IPs”

is a bonafide work carried out by

Karthik B K (PES2201800185)

In partial fulfillment for the completion of the 8th-semester course work in the Program of Study B.Tech in Electronics and Communication Engineering, under rules and regulations of PES University, Bengaluru during the period January – April 2022. It is certified that all corrections/suggestions indicated for internal assessment have been incorporated in the report. The report has been approved as it satisfies the 8th-semester academic requirements in respect of due internship work.

Signature with date & seal
Dr. Neel Gala
Direct Supervisor (External)

Signature with date & seal
Dr. Madhura Purnaprajna
Internal Supervisor

Signature with date & seal
Prof. Vinay Reddy
Internal Supervisor

Signature with date & seal
Dr. Ajey S N R
Chairperson

DECLARATION

I, **Karthik B K**, hereby declare that the project report titled, '**Design of Decode-BOX and Verification of (Data and Instruction) Cache, Register File, Bypass and RISC-V Bit-Manipulation Hardware IPs'**, is our original work under the guidance of **Dr. Neel Gala,** Co-Founder / CTO, InCore Semiconductors Pvt. Ltd., **Dr. Madhura Purnaprajna**, Professor and **Prof. Vinay Reddy**, Assistant Professor, ECE Department and is being submitted in partial fulfillment of the requirements for completion of 8th Semester course work in the Program of Study, B.Tech in Electronics and Communication Engineering.

Date and Place: Bengaluru, KA

Name, and Signature of the Candidates:

1. **Karthik B K (PES2201800185)**

InCore Semiconductors Pvt. Ltd.

and

PES University Electronic City Campus

Faculty of Engineering
Department of Electronics and Communication Engineering

Design of Decode-BOX and Verification of (Data and Instruction) Cache, Register File, Bypass and RISC-V Bit-Manipulation Hardware IPs

April 20, 2022

Internship Report

Karthik B K <bkkarthik@pesu.pes.edu>
SRN: PES2201800185

Dr. Neel Gala, InCore Semiconductors
Dr. Madhura Purnaprajna, PES University
Prof. Vinay Reddy, PES University

Contents

1 Abstract

RISC-V is a modular and open source instruction set architecture which is capable of withstanding both low power and high performance applications. During these 18 weeks, we have designed an automated framework to generate truly optimised, correct-by-construction decoders. We have also contributed to the verification of the existing data and instruction cache subsystems in Chromite[7], along with the RISC-V Bit-Manipulation[9] hardware IPs.

In this report, we discuss the 3 tasks completed during this internship in sufficient detail. In section 4.1, we describe the verification methodology (UATG[6]) used with the Chromite[7] RISC-V. In section 4.2, we discuss the design methodology for Decode-BOX, and the new opcodes format that was suggested as part of this work. Decode-BOX has not been described in full detail, as it is not completely open-source at the time this report was written. In section 4.3, we discuss the verification methodologies used with the RISC-V Bit-Manipulation[9] hardware IPs developed by InCore Semiconductors Pvt. Ltd., using CoCoTb[8]. Not all test information and cover points have been discussed in detail, as the project is not fully open-source at the time this report was written.

The Decode-BOX designs were not integrated with any RISC-V core at the time this report was written, which would be required to evaluate the performance and functional correctness.

Keywords: *RISC-V, Cache, Decode-BOX, Opcodes, CoCoTb, Bitmanip*

List of Tables

List of Figures

2 About the Company

InCore is a leading provider of core and uncore IPs. At InCore, our commitment to open source goes back over 3 decades and we leverage our academic roots to create an extensive open source based RISC-V core portfolio. Our drive for leadership in RISC-V innovation is anchored on the bedrock principle of open source and the ethos of intellectual commons. During this internship, we contributed to 3 different projects at InCore Semiconductors. Lightning fast solution realization at the least possible cost - we leverage cutting edge open source tools and methodologies to give our customers a state of the art agile development environment. InCore partners with leading companies to provide IP blocks, product design services, tools and foundry services - allowing our customers to realize optimal designs with minimal time to market. Beyond standard Processor IPs - InCore provides customization services for cores, fabrics, Peripheral IPs and SoCs to realize the optimal SoC.

3 Introduction

RISC-V is in it's early stages, where most of the extensions have not been ratified. To be able to preserve the modular nature of the instruction set architecture which expects to grow in a modular fashion in the near future is of paramount importance. When a processor IP has been designed, verifying it can be looked at from two perspectives, the ISA and the micro-architecture. From the ISA perspective, we say that the core has to be compliant to the ISA. RISC-V CTG [5]. From the micro-architectural perspective, we are required to generate tests as per our specific micro-architectural configurations/assumptions. With this background, the following internship objectives were proposed:

---

1. *Verification of (data and instruction) cache, regfile and bypass using UATG [6]:* ***μ****Architectural Test Generator - a python-based open source framework for generating assembly tests for functional verification.*

2. *Design of Decode-BOX: a highly configurable and open source python-based automated framework for generating truly-optimised correct-by-construction RISC-V instruction decoder IPs.*

3. *Verification of b-box: RISC-V Bit Manipulation Instruction (BMI) hardware IPs for multiple ratified and unratified (RV32/64: Zba, Zbb, Zbc, Zbe, Zbp, Zbs) sub-extensions.*

---

4 Internship Activities

4.1 Verification of Chromite RISC-V using UATG[6]

4.1.1 Chromite RISC-V

Chromite[7] is an open-source core generator, based on the SHAKTI C Class core developed at PS CDISHA at the Indian Institute of Technology Madras . The core generator emits synthesizable, production quality RTL of processors based on the open RISC-V ISA. The core generator can produce variants of a commercial grade 6-stage in-order core supporting the RV[64/32]GCSUN (or its subsets) extensions of the RISC-V ISA, from the same high-level source code. Chromite leverages the high level abstraction offered by Bluespec System Verilog to build highly parameterized, compact and powerful library components (like arithmetic units, branch predictors, caches, mmu, etc) that can be seamlessly integrated to create a solution catered to our needs.

4.1.2 UATG[6]: μArchitectural Test Generator

UATG - **μ**Architectural Test Generator - is an open-source python based framework developed by InCore to generate RISC-V Assembly tests for functional verification of the RISC-V cores (currently supports in-house and SHAKTI cores only). While complements of UATG, like RISCV-CTG[5], AAPG[3], etc work at the ISA level and focus on generating tests to maximize ISA coverage, UATG is more focused on micro-architecture driven tests. Micro-architectural components like caches, branch predictors, scoreboards, etc which are more implementation driven and less ISA defined, require a dedicated set of tests to be written to test them thoroughly.

Also, as InCore is primarily focused on core-generators like Chromite[7], a significant portion of the micro-architectural features are also heavily parameterized, and creating individual tests for each configuration is not possible. What is required is an equally parameterized set of verification tests for these micro-architectural features.

UATG addresses the above issues, and provides a minimal framework which allows one to create parameterized set of tests which UATG can generate and filter based on the input configuration of the target device. Along with generating tests, UATG also provides hooks to define and generate a parameterized set of cover points which can help indicate the health of the test-suite being run on a target.

The parameterized tests in UATG are written as python-plugins which can generate assembly programs. These python programs are required to follow a certain API as outlined by UATG and produce artifacts which UATG can use to generate the final Assembly tests. These python plugins have the capability to generate relevant tests based on the configuration of the target DUT or skip generation completely if certain features in the configuration have been disabled.

UATG has ensured that the framework and tests are decoupled and thus the tests themselves can be hosted as a separate directory or repository and can be fed into UATG to generate Assembly tests. You can find examples of python-plugins for some of Chromite's modules in the chromite-uatg-tests repository

The tests generated using UATG can be run on the DUT in the conventional way or by using a framework like RiVer Core[4]. The UATG plugin for RiVer Core automatically selects the uatg plugin to generate the tests, run it on the DUT, obtain coverage as well as compare the logs from DUT, and reference and finally provide a comprehensive report of your test's health!

4.1.3 Testing the Cache Subsystem

A total of 18 tests were written to test the dcache subsystem. These tests are available here. The following scenarios were tested:
- Fill the cache completely based on the size mentioned in the core64.yaml input.
- Try to fill the fill-buffer completely.
- Perform cache line thrashing.
- Perform cache set thrashing.
- Perform all possible types of load/store access (byte, hword, word, dword).
- Perform a load/store hit in the RAMS.
- Perform a load/store hit in the Fill-buffer.
- Perform an I/O operation.
- Perform a store-to-load forwarding scenario from the store-buffer.
- Perform a replacement on all sets.
- Check if fence.i works properly.
- Check if performance counters are correctly incremented.
- Check to see if we can perform simultaneous io and cached ops.
- Perform an operation whose lower and upper half-words are in different lines/ways of the icache.

The test description for some of the above mentioned tests is given below: *dcache*fill*01.py*

- Perform a `fence` operation to clear out the data cache subsystem and the fill buffer.
- Load some data into a temporary register and perform `numerous store operations` to fill up the cache.
- Each loop in ASM has an unconditional `jump` back to that label, a branch takes us out of the loop.
- Each iteration, we visit the next `set`.
- The total number of iterations is parameterized based on YAML input.

*dcache*fill*02.py*

- Perform a `fence` operation to clear out the data cache subsystem and the fill buffer.
- In each iteration, we visit the next way in the same set. Once all the ways in a set are touched, we visit the next set.
- The total number of iterations is parameterized based on YAML input.

*dcache*fill*03.py*

- Perform a `fence` operation to clear out the data cache subsystem and the fill buffer.
- Perform `numerous load operations` to fill up the cache.
- In each iteration, we visit the next way in the same set. Once all the ways in a set are touched, we visit the next set.
- The total number of iterations is parameterized based on YAML input.

*dcache*fill*04.py*

- Perform a `fence` operation to clear out the data cache subsystem and the fill buffer.
- Load some data into a temporary register and perform `numerous load operations` to fill up the cache.
- Each loop in ASM has an unconditional `jump` back to that label, a branch takes us out of the loop.
- Each iteration, we visit the next `set`.
- The total number of iterations is parameterized based on YAML input.

*dcache*fill*buffer_01.py*

- Perform a `fence` operation to clear out the data cache subsystem and the fill buffer.

- Load some data into a temporary register and perform `numerous store operations` to fill up the cache.
- Each loop in ASM has an unconditional `jump` back to that label, a branch takes us out of the loop.
- Each iteration, we visit the next `set`.
- The total number of iterations is parameterized based on YAML input.
- Once the cache is full, we perform numerous `consecutive store operations`.
- The number of iterations is parameterized based on the YAML input such that the fill_buffer is completely full.
- Post filling the caches, we perform a series of `nop` instructions to ensure that the fill buffer is empty.

*dcache*fill*buffer_02.py*

- Perform a `fence` operation to clear out the data cache subsystem and the fill buffer.
- Perform `numerous load operations` to fill up the cache.
- In each iteration, we visit the next way in the same set. Once all the ways in a set are touched, we visit the next set.
- The total number of iterations is parameterized based on YAML input.
- Once the cache is full, we perform numerous `consecutive load operations`.
- The number of iterations is parameterized based on the YAML input such that the fill_buffer is completely full.
- Post filling the caches, we perform a series of `nop` instructions to ensure that the fill buffer is empty.

*dcache*line*thrashing.py*

- Perform a `fence` operation to clear out the data cache subsystem and the fill buffer.
- First the cache is filled up using the following logic. For an *n-way* cache system, in each set there is *only 1 non dirty way* and the remaining *n-1 ways are dirty*.
- Now a series of `nop` operations are done inorder the ensure that the fillbuffer is empty and the cache is completely full.
- This is followed by a large series of back to back `store operations` with an address that maps to a single set in the cache. This ensures that the fillbuffer gets filled and the line thrashing process begins.
- Now after the fill buffer is full, with each store operation a cache miss is encountered and the non-dirty line in the set will be replaced.
- This process is iterated to test each cache line.

*dcache*set*thrashing.py*

- Perform a `fence` operation to clear out the data cache subsystem and the fill buffer.
- First the cache is filled up using the following logic. All the ways of a set should either be *dirty or clean.*
- This is followed by a large series of back to back `store operations` with an address that maps to a single set in the cache. This ensures that the fillbuffer gets filled and the set thrashing process begins.
- Now after the fill buffer is full, with each store operation a cache miss is encountered.
- This process is iterated to test each cache set.

*dcache*load*store_op.py*

- Perform a `fence` operation to clear out the data cache subsystem and the fill buffer.
- `Store` a single `byte` using `sb` and `load` it back using `lbu`.
- `Store` a `half word` using `sh` and `load` it back using `lhu`.
- `Store` a `word` using `sw` and `load` it back using `lwu`
- For the above three cases, the `load` should be identical to the store, as it is unsigned.
- `Store` a `double word` using `sd` and `load` it back using `ld`
- The following test cases are storing part of a double word where the remaining bits are set.

- `Load` from the same locations again, but this time allow the data to be sign extented.
- For the sign extended loads, compare with the sign extended versions of the test data.
- Always branch out if the load is not equal.
- `Store` a `double word` and then modify only half of it using `sw`, Then immediately `load` the entire `double word` and check if the modification has updated the value from the store buffer.

4.1.4 Testing the Register File

The RISC-V specification dictates that the register *x0* must be hard-wired to zero. This is being tested by performing some operations which attempt to write to the *x0* register. This test is available here.

4.1.5 Testing Bypass for Functional Units

Branch operation happens if bypass doesn't happen correctly. Available here, Bypassing was checked for the following scenarios:
- ALU operations in the base ISA.
- MULDIV operations in the base ISA.
- Checking pipeline flushes and invoking trap handler by creating misaligned loads.
- Checking bypass operation using signature region.

4.2 Designing Decode-BOX

Decode-BOX: is a python based open source framework that generates truly-optimised and correct by construction instruction decode stage for any subset of the RISC-V ISA. The decoder implements the ISA dependent and Microarchitecture dependent components separately. The microarchitecture specification is highly configurable using YAML files.

Decode-BOX is a standalone framework that generates synthesizable and highly optimised Bluespec System Verilog description of the decode unit. The framework uses riscv-opcodes as a source to generate all the components that are purely dependent on the ISA specification. Whenever the ISA changes, Decode-BOX will get an update to incorporate those changes. Decode-BOX as a utility is targeted towards design engineers who wish to implement a fully optimised decode unit that meets the RISC-V ISA specification and is highly configurable in terms of the microarchitecture they want to implement.

4.2.1 The Need for Decode-BOX

When considering an ISA like RISC-V which is significantly modular and expects to grow modularly, being able to generate a truly-optimised decode stage will make a significant impact on the performance of our core.

4.2.2 Mechanism Inside Decode-BOX: A Brief

Decode-BOX uses “riscv-opcodes“ and PyEDA[1] to deal with all parts that depend on the instruction set architecture specification. The new format of *riscv-opcodes* will be discussed in section <>, allows us to easily parse the opcodes and the fields to be decoded for each instruction. One of the most basic fields to be decoded, would be the register addresses.

4.2.3 Decoding Register Operand Addresses

The first of the variable-fields to be decoded are the operand addresses. We iterate through all instructions to be supported and create a mask of which operands are to be decoded for each instruction. These masks (outputs) along with the encodings (inputs) of the instruction are written into a PLA file. This PLA is then optimised using the Espresso Logic Minimizer.

The PLA takes a 32-bit input and generates a 3-bit/4-bit output based on the maximum length of the mask. These bits are now used to mask the address decoding as described in the diagram below:

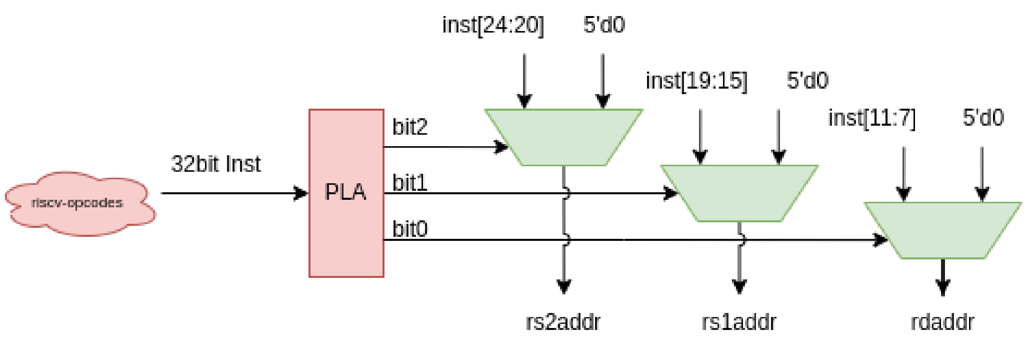

Figure 1: Decoding Register Operand Addresses

We generate the *PLA* file using python, and for RV32I, it looks something like this:

```
#See LICENSE.incore for license details
#decode_box version: 0.0.0,
#user bkkarthik
#file generated at: Thursday, 17. February 2022 04:41PM,
#command : 'decode_box generate -isa RV32I -bd ./build -ld ./lib -tf ./sample_config/types.yaml'

.i 32
.o 3
.p 43
.ilb i31 i30 i29 i28 i27 i26 i25 i24 i23 i22 i21 i20 i19 i18 i17 i16 i15 i14 i13 i12 i11 i10 \\
i9 i8 i7 i6 i5 i4 i3 i2 i1 i0
.ob o2 o1 o0
.type fr
-----------------100-----0000011 011                #lbu
-----------------000-----0100011 110                #sb
-----------------101-----1100011 110                #bge
0000000----------001-----0010011 011                #slli
0000000----------000-----0110011 111                #add
0000000----------001-----0110011 111                #sll
-----------------000-----0010011 011                #addi
0000000----------011-----0110011 111                #sltu
00000000000100000000000001110011 000                #ebreak
00010000010100000000000001110011 000                #wfi
-----------------010-----0010011 011                #slti
-----------------110-----1100011 110                #bltu
-----------------010-----0000011 011                #lw
-------------------------1101111 001                #jal
01111011001000000000000001110011 000                #dret
-------------------------0110111 001                #lui
0000000----------110-----0110011 111                #or
-----------------001-----0100011 110                #sh
-----------------100-----1100011 110                #blt
-----------------001-----1100011 110                #bne
0000000----------101-----0110011 111                #srl
0000000----------101-----0010011 011                #srli
-----------------000-----0001111 011                #fence
-----------------100-----0010011 011                #xori
0100000----------101-----0110011 111                #sra
-----------------110-----0010011 011                #ori
0100000----------101-----0010011 011                #srai
-----------------111-----1100011 110                #bgeu
00110000001000000000000001110011 000                #mret
-----------------111-----0010011 011                #andi
-----------------010-----0100011 110                #sw
-----------------101-----0000011 011                #lhu
0000000----------010-----0110011 111                #slt
0000000----------100-----0110011 111                #xor
-----------------001-----0000011 011                #lh
0000000----------111-----0110011 111                #and
00000000000000000000000001110011 000                #ecall
-----------------011-----0010011 011                #sltiu
-----------------000-----1100011 110                #beq
-----------------000-----1100111 011                #jalr
0100000----------000-----0110011 111                #sub
-----------------000-----0000011 011                #lb
```

```
-------------------------0010111 001                #auipc
.e
```

This is the un-optimized PLA file. We now invoke *Espresso* from PyEDA[1] as described in the figure below: Upon optimizing the

Unoptimised PLA
Espresso Logic Minimizer
Optimised PLA

Figure 2: Invoking Espresso Logic Minimizer

PLA file using Espresso from PyEDA[1], we obtain the ‘optimized‘ PLA file, which looks something like this:

```
#See LICENSE.incore for license details
#decode_box version: 0.0.0,
#user bkkarthik
#file generated at: Thursday, 17. February 2022 04:41PM,
#command : 'decode_box generate -isa RV32I -bd ./build -ld ./lib -tf ./sample_config/types.yaml'
.i 32
.o 3
.ilb i31 i30 i29 i28 i27 i26 i25 i24 i23 i22 i21 i20 i19 i18 i17 i16 i15 i14 i13 i12 i11 i10 \\
i9 i8 i7 i6 i5 i4 i3 i2 i1 i0
.ob o2 o1 o0
.p 6
-------------------------01--0-- 100
--------------------------10-0-- 100
-------------------------0-1-0-- 011
---------------------------00--- 010
-----------------------------1-- 001
--------------------------00---- 011
.e
```

This is proof that the Espresso Logic Minimizer is able to heavily optimise the PLA file. Using the above ‘optimised‘ PLA file, we create a function in BluespecSystemVerilog as follows:

```
function Bit#(3) fn_create_op_addr_mask (Bit#(32) i);
    Bit#(3) o=?;

    o[2] = (~i[6] & i[5] & ~i[2]) | (i[5] & ~i[4] & ~i[2]);
    o[1] = (~i[6] & i[4] & ~i[2]) | (~i[4] & ~i[3]) | (~i[5] & ~i[4]);
    o[0] = (~i[6] & i[4] & ~i[2]) | (i[2]) | (~i[5] & ~i[4]);

    return o;
endfunction: fn_create_op_addr_mask
```

We now use this ‘mask‘ to decode the operand address as follows:

```
function Tuple3#(Bit#(5), Bit#(5), Bit#(5)) fn_dec_op_addr(Bit#(32) inst);
    Bit#(3) mask;

    mask = fn_create_op_addr_mask(inst);

    return tuple3(mask[0]==1?inst[11:7]: 0, mask[1]==1?inst[19:15]:0, mask[2]==1?inst[24:20]:0);
endfunction:fn_dec_op_addr
```

Similarly, we decode:
- Operand address

- Immediate values
- Memory access type
- Functional unit type
- Instruction Legality
- Cause value for traps
- Word32 status
- ALU recoded ops

All these functions are delivered as a single BluespecSystemVerilog file that can be built into HDL like Verilog. This makes Decode-BOX a fully-automated framework that generates truly-optimised and correct by construction, synthesizable instruction decoder for any subset of the RISC-V ISA.

4.2.4 New Opcodes Format

riscv-opcodes[2] is being used as the base for all decoding that is defined by the instruction set architecture's specification. This repository enumerates standard RISC-V instruction opcodes and control and status registers. It also contains a script to convert them into several formats (C, Scala, LaTeX). Artifacts (encoding.h, latex-tables, etc) from this repository are used in other tools and projects like Spike, PK, RISC-V Manual, etc.

This project follows a very specific file structure to define the instruction encodings. All files containing instruction encodings start with the prefix `rv`. These files can either be present in the root directory (if the instructions have been ratified) of the `unratified` directory. The exact file-naming policy and location is as mentioned below:

1. `rv_x` - contains instructions common within the 32-bit and 64-bit modes of extension X.
2. `rv32_x` - contains instructions present in rv32x only (absent in rv64x e.g.. brev8)
3. `rv64_x` - contains instructions present in rv64x only (absent in rv32x, e.g. addw)
4. `rv_x_y` - contains instructions when both extension X and Y are available/enabled. It is recommended to follow canonical ordering for such file names as specified by the spec.
5. `unratified` - this directory will also contain files similar to the above policies, but will correspond to instructions which have not yet been ratified.

When an instruction is present in multiple extensions and the spec is vague in defining the extension which owns the instruction, the instruction encoding must be placed in the first canonically ordered extension and should be imported(via the `\$import` keyword) in the remaining extensions.

The encoding syntax uses `\$` to indicate keywords. As of now 2 keywords have been identified : `\$import` and `\$pseudo_op` (described below). The syntax also uses `::` as a means to define the relationship between extension and instruction. `..` is used to defined bit ranges. We use `#` to define comments in the files. All comments must be in a separate line. In-line comments are not supported.

Instruction syntaxes used in this project are broadly categorized into three:

- **regular instructions** :- these are instructions which hold a unique opcode in the encoding space. A very generic syntax guideline for these instructions is as follows: `<instruction name> <instruction args> <bit-encodings>` Examples: `lui rd imm20 6..2=0x0D 1..0=3 beq bimm12hi rs1 rs2 bimm12lo 14..12=0 6..2=0x18 1..0=3` The bit encodings are usually of 2 types:
  - *single bit assignment* : here the value of a single bit is assigned using syntax `<bit-position>=<value>`. For e.g. `6=1` means bit 6 should be 1. Here the value must be 1 or 0.
  - *range assignment*: here a range of bits is assigned a value using syntax: `<msb>..<lsb>=<val>`. For e.g. `31..24=0xab`. The value here can be either unsigned integer, hex (0x) or binary (0b).
- **pseudo_instructions** (a.k.a pseudo_ops) - These are instructions which are aliases of regular instructions. Their encodings force certain restrictions over the regular instruction. The syntax for such instructions uses the `\$pseudo_op` keyword as follows: `\$pseudo_op <extension>::<base-instruction> <instruction name> <instruction args> <bit-encodings>` Here the `<extension>` specifies the extension which contains the base instruction. `<base-instruction>` indicates the name of the instruction this pseudo-instruction is an alias of. The remaining fields are the same as the regular instruction syntax, where all the args and the fields of the pseudo instruction are specified.

Example: `\$pseudo_op rv_zicsr::csrrs frflags rd 19..15=0 31..20=0x001 14..12=2 6..2=0x1C 1..0=3`

If a ratified instruction is a pseudo_op of a regular unratified instruction, it is recommended to maintain this pseudo_op relationship i.e. define the new instruction as a pseudo_op of the unratified regular instruction, as this avoids existence of overlapping opcodes for users who are experimenting with unratified extensions as well.

- **imported_instructions** - these are instructions which are borrowed from an extension into a new/different extension/sub-extension. Only regular instructions can be imported. Pseudo-op instructions cannot be imported. Example: `\$import rv32_zkne::aes32esmi`

4.3 Verifying the RISC-V Bitmanip Hardware IPs

The RISC-V Bitmanip Extension[9] includes implementation guidelines and architecture for the instructions in the bitmanip extension (both ratified and un-ratified).The hardware IPs for implementing the bitrmanip instructions have been developed by InCore Semiconductors Pvt. Ltd. We use CoCoTb[8]: a COroutine based COsimulation TestBench environment for verifying the Verilog RTL generated using BluespecSystemVerilog using Python.

4.3.1 CoCoTb[8]: COroutine based COsimulation TestBench environment

cocotb is completely free, open source (under the BSD License) and hosted on GitHub. cocotb requires a simulator to simulate the HDL design and has been used with a variety of simulators on Linux, Windows and macOS. A typical cocotb testbench requires no additional RTL code. The Design Under Test (DUT) is instantiated as the toplevel in the simulator without any wrapper code. cocotb drives stimulus onto the inputs to the DUT (or further down the hierarchy) and monitors the outputs directly from Python.

A test is simply a Python function. At any given time either the simulator is advancing time or the Python code is executing. A test can spawn multiple coroutines, allowing for independent flows of execution. We look at CoCoTb from a unit-testing perspective, and write coverage sections and tests to hit those cover points.

4.3.2 Using CoCoTb for functional verification

A typical cocotb testbench requires no additional RTL code. The Design Under Test (DUT) is instantiated as the toplevel in the simulator without any wrapper code. cocotb drives stimulus onto the inputs to the DUT (or further down the hierarchy) and monitors the outputs directly from Python, as desribed in figure 3.

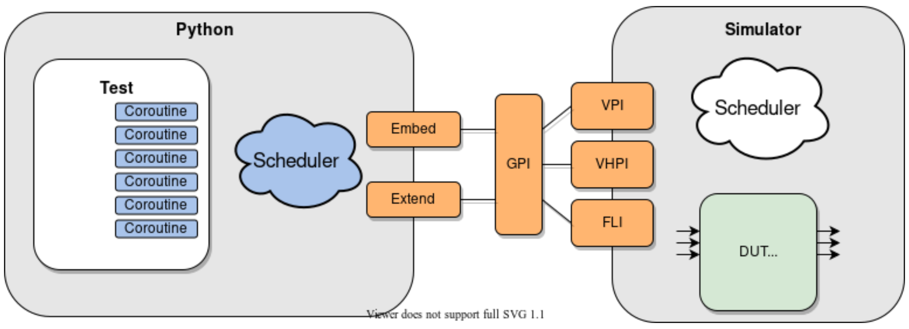

Figure 3: An Overview of the CoCoTb Environment

In cocotb, we can access all internals of your design, e.g. signals, ports, parameters, etc. through an object that is passed to each test. In the following we'll call this object dut.

Let's create a test file *test_my_design.py* containing the following:

```
# test_my_design.py (simple)

import cocotb
from cocotb.triggers import Timer


@cocotb.test()
async def my_first_test(dut):
    """Try accessing the design."""

    for cycle in range(10):
        dut.clk.value = 0
        await Timer(1, units="ns")
        dut.clk.value = 1
        await Timer(1, units="ns")

    dut._log.info("my_signal_1 is \%s", dut.my_signal_1.value)
    assert dut.my_signal_2.value[0] == 0, "my_signal_2[0] is not 0!"
```

This will first drive 10 periods of a square wave clock onto a port clk of the toplevel. After this, the clock stops, the value of my_signal_1 is printed, and the value of index 0 of my_signal_2 is checked to be 0.

The test shown is running sequentially, from start to end. Each await expression suspends execution of the test until whatever event the test is waiting for occurs and the simulator returns control back to cocotb. It's most likely that you will want to do several things "at the same time" however - think multiple always blocks in Verilog or process statements in VHDL. In cocotb, you might move the clock generation part of the example above into its own async function and start() it ("start it in the background") from the test:

```
# test_my_design.py (extended)

import cocotb
from cocotb.triggers import Timer
from cocotb.triggers import FallingEdge


async def generate_clock(dut):
    """Generate clock pulses."""

    for cycle in range(10):
        dut.clk.value = 0
        await Timer(1, units="ns")
        dut.clk.value = 1
        await Timer(1, units="ns")


@cocotb.test()
async def my_second_test(dut):
    """Try accessing the design."""

    await cocotb.start(generate_clock(dut))  # run the clock "in the background"

    await Timer(5, units="ns")  # wait a bit
    await FallingEdge(dut.clk)  # wait for falling edge/"negedge"

    dut._log.info("my_signal_1 is \%s", dut.my_signal_1.value)
    assert dut.my_signal_2.value[0] == 0, "my_signal_2[0] is not 0!"
```

Note that the generate_clock() function is not marked with @cocotb.test() since this is not a test on its own, just a helper function.

Now, we create a test environment as described in figure 4.

The testbench would consist of the components described in table 1.

4.3.3 The RISC-V Bit-manipulation Extension[9]

The RISC-V Bit-manipulation Extension[9] consists of hundreds of instructions distributed over various sub-extensions. The following figure visually describes the overlapping that exists in the ratified and unratified sub-extensions:

4.3.4 Instructions in the Zbp Sub-Extension

The riscv-bitmanip extension lists the following instructions to be a part of the Zbp sub-extension: ANDN, ORN, XNOR, ROL[W], ROR[I][W] , XPERM.N[BHW], GREV[I][W] / REV8, GORC[I][W] / ORC.b, [UN]SHFL[I][W] , PACK[U]|[W] , PACK[H]

4.3.5 Verifying the Zbp Sub-Extension

We wrote 15+ tests for verifying the DUT, including tests where we considered the order in which these outputs are encountered in the output side monitor of the test environment. We also evolved the test framework by including checks that would count the number of clock cycles based on the hardware configuration of the design under verification.

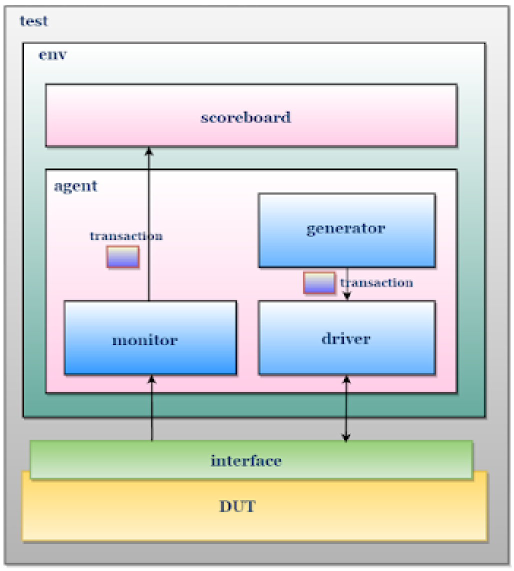

Figure 4: TestBench Architecture

Name	Type	Description
transaction	class	Defines the pin level activity generated by agent (to drive to DUT through the driver) or the activity has to be observed by agent (Placeholder for the activity monitored by the monitor on DUT signals)
generator	class	Generates the stimulus (create and randomize the transaction class) and send it to Driver
driver	class	Receives the stimulus (transaction) from a generator and drives the packet level data inside the transaction into pin level (to DUT
monitor	class	Observes pin level activity on interface signals and converts into packet level which is sent to the components such as scoreboard
agent	class	An agent is a container class, which groups the class's (generator, driver, and monitor) specific to an interface or protocol
scoreboard	class	Receives data items from monitors and compares them with expected values. Expected values can be either golden reference values or generated from the reference model
environment	class	The environment is a container class for grouping higher level components like agent's and scoreboard
test	program	The test is responsible for, - Configuring the testbench - Initiate the testbench components construction process - Initiate the stimulus driving
testbench_top	class	This is the topmost file, which connects the DUT and TestBench. It consists of DUT, Test and interface instances, the interface connects the DUT and TestBench

Table 1: Components of a TestBench

All these tests were run for various configurations of the DUV, for both RV32 and RV64. All tests resulted in 100% coverage at all times, and no tests failed. These tests included some common testing patterns like walking/marching and soak test.

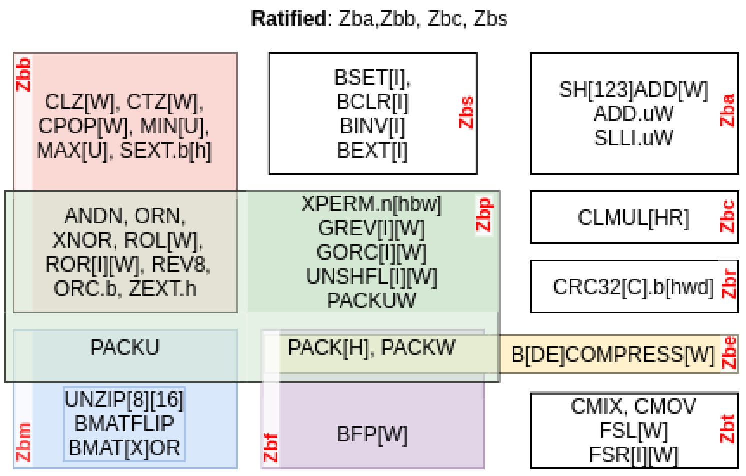
Ratified: Zba,Zbb, Zbc, Zbs
Zbb
CLZ[W], CTZ[W], CPOP[W], MIN[U], MAX[U], SEXT.b[h]
BSET[I], BCLR[I] BINV[I] BEXT[I]
Zbs
SH[123]ADD[W] ADD.uW SLLI.uW
Zba
ANDN, ORN, XNOR, ROL[W], ROR[I][W], REV8, ORC.b, ZEXT.h
XPERM.n[hbw] GREV[I][W] GORC[I][W] UNSHFL[I][W] PACKUW
Zbp
CLMUL[HR]
Zbc
CRC32[C].b[hwd]
Zbr
PACKU
PACK[H], PACKW
B[DE]COMPRESS[W]
Zbe
UNZIP[8][16] BMATFLIP BMAT[X]OR
Zbm
Zbf
BFP[W]
CMIX, CMOV FSL[W] FSR[I][W]
Zbt

Figure 5: Instructions in the RISC-V Bit-Manipulation Extension

5 Conclusions

20+ tests were written in the process of verifying the cache sub-systems and 15+ tests were written in the process of verifying the bitmanip hardware IPs. Decode-BOX was successfully designed, developed, and documented. Future work would include integrating the decoder IPs into a core pipeline and benchmarking the performance improvement.

6 Acknowledgements

I would like to thank my internship supervisor at InCore Semiconductors Pvt. Ltd., Dr. Neel Gala, for closely working with me and supervising me, hosting regular meetings (once a week) with me to see my progress. I would like to thank Mr. Babu P S, Mr. Alenkruth K M, and Mr. Purushothaman Palani, Mr. Pawan Kumar Incore Semiconductors Pvt. Ltd., for helping out with my work. I would also like to thank Dr. Madhura Purnaprajna and Prof. Vinay Reddy, PES University, who were involved in the interviews and selection process for the internship position, for giving me an opportunity to work at InCore Semiconductors Pvt. Ltd. and for acting as my internal assessors for this internship. I would also like to thank Kishan S, my good friend and colleague. Finally, I would like to thank PES University, for giving me an opportunity to work in an external company environment as a part of my Bachelor's course.

7 Appendix

The internship took place in the months of January to April. Most of the work was done either at my place of residence when working from home. The first three weeks were spent writing tests for the data and instruction caches within chromite-uatg-tests. The next 5 weeks were spent designing and developing decode-box. The next 4 weeks were spent writing tests for the bitmanip hardware IPs. Attached below are my activity logs, which serve as a proof of attendance, as the contain the timestamps of each of my commits during the period of this internship.

7.1 Activity log 1 - Verifying data and instruction caches, Register File, Bypass using UATG

```
2022-01-04 11:43:57 +0530 caches: Add tests for dcache subsystem
2022-01-04 11:54:50 +0530 index.yaml: add dcache tests
2022-01-04 14:43:54 +0530 caches: add test for hits in fill_buffer
2022-01-04 16:34:59 +0530 Added cache replacement python code for replacing entire cache three times
2022-01-04 18:48:26 +0530 caches: add test for hit in RAM
2022-01-04 18:48:58 +0530 caches: Follow a 80 char word limit
2022-01-04 18:49:25 +0530 index: add new tests
2022-01-05 10:56:13 +0530 fill_04: Improve code readability
2022-01-05 11:12:37 +0530 fill_buffer_01: Improve code readability
2022-01-05 11:17:33 +0530 fill_buffer_02: improve code readabiltiy
2022-01-05 11:35:01 +0530 line_thrashing: improve code readability
2022-01-05 12:04:45 +0530 load_store_types: improve code readability
2022-01-05 12:32:23 +0530 set_thrashing: improve code readability
2022-01-05 13:39:06 +0530 Improved code readability
2022-01-05 16:40:26 +0530 caches: Add dcache_fill_05
2022-01-05 16:56:48 +0530 Added 11 variations of load/store
2022-01-05 17:49:04 +0530 Added docstring for dcache_fill_all
2022-01-05 22:02:16 +0530 Added random load/store buffer
2022-01-06 10:20:03 +0530 load_store)_types: add tests for critical word
2022-01-06 10:38:17 +0530 load_store_types: add tests for critical word
2022-01-06 11:52:45 +0530 dcache: Add more test schemes for critical words in load_store_types
2022-01-06 14:44:02 +0530 Ignore log files
2022-01-06 14:44:29 +0530 Use dwords in rvtest_data instead of words
2022-01-06 15:18:05 +0530 atomic operations on data cache
2022-01-07 10:49:50 +0530 caches: align rvtest_data to word_size
2022-01-07 14:00:18 +0530 caches: add test for reading during eviction
2022-01-07 14:24:26 +0530 caches: update test to read during replacement
2022-01-07 15:04:12 +0530 Update checks for replacement policy in the YAML
2022-01-07 18:09:30 +0530 dcache: fix some typos and rename tests
2022-01-10 12:16:52 +0530 Filling icache
2022-01-10 12:17:43 +0530 Updating index.yaml
2022-01-10 12:19:26 +0530 Updating docstring in icache_fill
2022-01-10 17:35:14 +0530 Removed compressed
2022-01-11 09:34:19 +0530 Rename files
2022-01-11 12:20:47 +0530 icache: Add set_thrashing
2022-01-11 14:12:12 +0530 icache: Add test to check for races in the bus
2022-01-11 17:04:03 +0530 use icache params from YAML
2022-01-11 17:09:02 +0530 Self modifying code to check fence.i functionality
2022-01-11 17:58:55 +0530 set_thrasing: more passes on each set
2022-01-11 21:48:05 +0530 Exit only if fence.i works correctly
2022-01-12 16:03:10 +0530 icache: Add a better variant of set_thrashing
2022-01-12 16:16:30 +0530 Recuce size in .align directives
2022-01-12 17:01:28 +0530 Update alignments
2022-01-12 18:34:41 +0530 icache: add tests for RR replacement policy
2022-01-12 19:04:28 +0530 Exhaustive critical word test
2022-01-12 23:16:18 +0530 Updated Docstring
2022-01-13 12:33:35 +0530 icache: Add tests for PLRU replacement policy
2022-01-13 12:43:48 +0530 Updating index.yaml
2022-01-13 16:54:03 +0530 Regfile branch
2022-01-13 17:06:31 +0530 Correcting errors in python file
2022-01-13 17:30:54 +0530 bypass: trap: fix syntax in asm
2022-01-14 10:13:10 +0530 dcache: Add co-author info
2022-01-14 10:33:46 +0530 Add co-author info
2022-01-14 10:49:31 +0530 Alignment
```

```
2022-01-14 10:57:38 +0530 80 char limit
2022-01-14 11:12:18 +0530 ASM alignment
2022-01-14 11:51:39 +0530 Corrected functional errors and added docstring
2022-01-18 10:10:42 +0530 Directly return the dict with asm and sig
2022-01-18 10:19:28 +0530 Remove check_log and generate_covergroups
2022-01-18 10:44:12 +0530 Using instruction_constants instead of created lists
2022-01-18 10:46:10 +0530 Fix return type error
2022-01-18 11:00:09 +0530 Add test for x0 being zero.
2022-01-18 12:27:29 +0530 Changed variable names and formatting style
2022-01-19 15:05:26 +0530 Revert "Changed variable names and formatting style"
2022-01-19 15:17:21 +0530 changed variables names
2022-01-19 16:15:38 +0530 caches: Add check_logs and generate_covergroups
2022-01-21 16:30:18 +0530 Jumping across last instructions in a set
2022-01-22 10:00:04 +0530 caches: initialise registers before using them
2022-01-22 10:15:39 +0530 rvtest_data: prevent illegal memory accesses
2022-01-22 10:26:05 +0530 execute: Check XLEN
2022-01-22 13:26:20 +0530 Added 32/64 isa len check and split load_store_types and critical
2022-01-28 13:23:56 +0530 Corrected functional errors, illegal accesses, naming convention
2022-02-02 13:31:34 +0530 Changed loading begin address to rectify errors
2022-02-02 13:48:11 +0530 Correction of addresses for loading and storing
```

7.2 Activity log 2 - Design and Development of Decode-BOX

```
2022-01-18 08:57:25 +0000 Initial commit
2022-01-18 16:19:44 +0530 initial commit
2022-01-19 13:11:21 +0530 Adding gitignore
2022-01-19 16:44:35 +0530 Add instruction semantics for RV32/64[I,M]
2022-01-19 16:50:19 +0530 removed isem_rv32m
2022-01-19 16:56:33 +0530 isem: Add rs1type and rs2type
2022-01-19 17:30:41 +0530 rename fields according to the schema
2022-01-20 10:11:20 +0530 incremental instructions in isem_rv64m
2022-01-20 10:45:14 +0530 Correction to asm syntax of sw
2022-01-20 15:06:46 +0530 Add rv32f instructions
2022-01-20 15:45:11 +0530 Atomic branch
2022-01-20 15:48:19 +0530 FU Correction
2022-01-20 16:57:12 +0530 rv64a
2022-01-20 17:15:09 +0530 rv32f: Fix asm syntax
2022-01-20 17:15:25 +0530 Add rv64f instructions
2022-01-20 20:06:38 +0530 added bsv function generator from pla
2022-01-21 13:59:26 +0530 chamge fu_type to float
2022-01-21 13:59:39 +0530 Add rv32d instructions
2022-01-21 13:59:48 +0530 Add rv64d instructions
2022-01-21 14:00:43 +0530 change fu_type to muldiv
2022-01-21 14:01:12 +0530 rv64m: change fu_type to muldiv
2022-01-21 14:03:09 +0530 remove [M] from this branch
2022-01-23 12:39:19 +0530 isem_configs: move YAML files
2022-01-24 09:34:06 +0530 Add socumentation for isem_schema
2022-01-24 09:34:27 +0530 Zicsr: Add SYSTEM instructions
2022-01-24 11:34:33 +0530 Zbb except psuedo ops of Zbp
2022-01-24 11:44:12 +0530 Zba subextension
2022-01-24 11:48:29 +0530 Zbc subextension
2022-01-24 11:57:48 +0530 Update functional units
2022-01-24 11:57:57 +0530 Changed std_ext to list, changed fu from alu to bbox
2022-01-24 12:11:22 +0530 Zbs subextension
2022-01-24 12:12:28 +0530 change asm_syntax and immidiate values
2022-01-24 12:24:10 +0530 PseudoI added for Zbb
2022-01-24 12:48:34 +0530 Added f12 in schema
2022-01-24 13:20:24 +0530 change iformat for immidiate type csr ops
2022-01-24 13:21:16 +0530 add fence.i instruction
2022-01-24 13:21:55 +0530 Change std_extension to a list
2022-01-24 13:23:12 +0530 Fix type in std_extension
2022-01-24 13:25:16 +0530 make list for std_extension and fix typo
2022-01-24 13:27:39 +0530 make list in std_extensioin and fix typo
2022-01-24 14:41:56 +0530 Std extension correction
2022-01-24 18:19:18 +0530 Fix major opcode for rv32m
2022-01-24 18:20:17 +0530 Fix f7 for rv32m
2022-01-24 18:26:20 +0530 Fix asm syntax and f3 fields
2022-01-24 18:35:34 +0530 Std extension for base ISA
2022-01-24 18:37:53 +0530 Added latest decode-box.py file from dev branch
2022-01-24 18:50:29 +0530 fix asm syntax, major opcode, f3 fields
2022-01-24 18:51:05 +0530 Remove zbc.yaml
2022-01-24 19:11:00 +0530 Remove zbc form branch FD
2022-01-25 07:39:04 +0530 Funct field corrections, added f6 in schema
2022-01-25 09:28:35 +0530 change immidiate fields  in csr ops
2022-01-25 10:49:32 +0530 Merge remote-tracking branch 'origin/aext' into my-dev
2022-01-25 10:49:58 +0530 Merge remote-tracking branch 'origin/mext' into my-dev
2022-01-25 10:51:58 +0530 Merge remote-tracking branch 'origin/fdext' into my-dev
```

```
2022-01-25 10:52:46 +0530 Merge remote-tracking branch 'origin/bext' into my-dev
2022-01-25 10:53:22 +0530 Merge remote-tracking branch 'origin/Zicsr' into my-dev
2022-01-25 05:29:10 +0000 Merge branch 'my-dev' into 'dev'
2022-01-25 11:49:02 +0530 Corrected typo in std_ext
2022-01-25 12:14:25 +0530 change r*type
2022-01-25 12:25:10 +0530 changed r*type and rs2 fields
2022-01-25 12:38:14 +0530 fix r*types and fields
2022-01-25 12:45:57 +0530 change fields
2022-01-25 12:46:12 +0530 Add new fields to schema
2022-01-25 12:51:54 +0530 Add trap and pseudo_op nodes
2022-01-25 15:26:03 +0530 P extension, schema modifications
2022-01-25 18:35:14 +0530 P extension and corrections
2022-01-26 13:49:39 +0530 Add pycache to gitignore
2022-01-26 13:49:54 +0530 Remove decode-box.py
2022-01-26 13:50:14 +0530 Add CLI support
2022-01-26 13:50:23 +0530 Add logger from UATG
2022-01-26 17:54:33 +0530 Add more logger statements
2022-01-26 17:55:10 +0530 Append a newline char after every new yaml.
2022-01-26 18:44:48 +0530 Added fields for aq and rl in schema
2022-01-27 12:06:02 +0530 Changed rs2 to f5_0
2022-01-27 12:09:12 +0530 Added pbox in schema
2022-01-27 12:11:46 +0530 Added rs3 in schema
2022-01-27 14:45:40 +0530 Revert "Added rs3 in schema"
2022-01-27 17:18:31 +0530 generate reference models
2022-01-27 17:49:26 +0530 Fix list-extensions
2022-01-27 17:59:57 +0530 adding missing newline character and deindentation of string
2022-01-27 18:10:27 +0530 optimized reference bsv case functions
2022-01-28 12:50:36 +0530 z extensions and mapping
2022-01-28 12:52:57 +0530 remove bsv string from logger, opt.bsv in one file
2022-01-28 12:57:08 +0530 Remove encodings from logger and move it to build/misc/encodings.txt
2022-01-28 12:59:22 +0530 Disable verilog generation
2022-01-28 14:08:42 +0530 Add header info to all files in build
2022-01-28 16:12:10 +0530 Check for executables
2022-01-28 17:04:49 +0530 Add support to change build directory
2022-01-28 20:23:53 +0530 Added seperation for Zifencei and i
2022-01-28 22:56:31 +0530 Removed B and added individual subextensions
2022-01-31 09:26:07 +0530 make consistent use of tabs and spaces
2022-01-31 18:22:35 +0530 Added PLA rd,rs1,rs2 select mask
2022-02-01 09:15:19 +0530 make consistent use of tabs and spaces
2022-02-01 10:11:56 +0530 Change encodings source from isem to riscv opcodes
2022-02-01 11:18:52 +0530 Create instruction dict and generate reference from this dict
2022-02-01 11:58:08 +0530 Using instr_dict as source instead of encodings.txt
2022-02-01 12:11:35 +0530 make mask a new field instead of a new dict
2022-02-01 12:45:00 +0530 Create PLA with addr_mask
2022-02-01 12:49:09 +0530 change typestr in addr_mask.pla
2022-02-01 13:42:46 +0530 Parse rv32 opcodes as well when xlen is 64
2022-02-01 13:46:12 +0530 fix pla type field
2022-02-01 22:34:28 +0530 stop tracking riscv-opcodes
2022-02-01 22:35:10 +0530 cleanup for create_inst_dict
2022-02-01 22:46:54 +0530 remove prints, dump instr_dict and show error on isa string for lowercase
2022-02-02 10:12:42 +0530 fix all the directory paths as part of the class initialization
2022-02-02 10:16:46 +0530 use yaml to dump instruction dict and fix build directory paths
2022-02-02 10:34:35 +0530 Add sanity check for duplicate encodings
2022-02-02 10:35:22 +0530 utils: init: Add default values for args
2022-02-02 12:42:17 +0530 Create addr_mask and it's PLA
```

```
2022-02-02 12:43:54 +0530 Pass values to the constructor in main instead of default args
2022-02-02 13:00:15 +0530 Prevent unpacking more values than present
2022-02-02 14:12:32 +0530 updated constants.py to include all regex patterns and instruction var LUT
2022-02-02 14:12:46 +0530 change default verbosity to info
2022-02-02 14:13:24 +0530 added support for imported and pseudo ops to create instr dict
2022-02-02 15:03:10 +0530 fixed function for duplicate encoding checks
2022-02-02 15:11:40 +0530 fix single value regex to include hex/binary values
2022-02-02 16:24:02 +0530 adding more graceful exits for corner case mistakes
2022-02-02 16:35:44 +0530 adding ISA regex to ensure unwated strings  are not parsed and processed
2022-02-02 16:55:15 +0530 cleaned up header insertion mechanism
2022-02-02 18:06:03 +0530 Create optimised PLA and BSV for addr_mask
2022-02-02 19:51:43 +0530 ignore build directory
2022-02-02 19:52:11 +0530 minor clean up and add package/endpackage to bsv files created along with header
2022-02-02 20:03:39 +0530 print number of instructions
2022-02-02 20:03:53 +0530 move espresso_exe as a class variable
2022-02-02 23:20:31 +0530 Split string into multiple strings and corrected indentation in output bsv
2022-02-03 12:21:33 +0530 create immediate mask
2022-02-03 12:37:54 +0530 keep only immmidiates instead of removing the others
2022-02-03 20:56:37 +0530 create a encodings.bsv once instruction dict is created
2022-02-03 22:38:53 +0530 Function for generating 32 bit immediate using mask and instruction
2022-02-04 09:44:21 +0530 move all_immediate_fields to constants.py
2022-02-04 12:19:17 +0530 Added variable mask lengths
2022-02-04 12:41:36 +0530 move imm_dec_Dict to constants
2022-02-04 12:59:32 +0530 Create bsv for decoding immediates
2022-02-04 13:11:59 +0530 move hi and lo immediates as a single immediate
2022-02-04 13:13:46 +0530 insert header to immediate_mask.bsv
2022-02-04 13:14:13 +0530 fix typo in filename
2022-02-04 14:35:43 +0530 work in progress: adding operand types to constants
2022-02-07 16:55:27 +0530 Generate PLA and BSV to check instruction legality
2022-02-07 16:57:01 +0530 Add exit message for successful run
2022-02-07 17:22:09 +0530 Add mechanism to include multiple CSRs for instr legality
2022-02-07 17:53:55 +0530 Partial op3_type and bsv bracket correction
2022-02-07 17:56:26 +0530 Syntax error correction
2022-02-07 17:57:12 +0530 Syntax error correction
2022-02-07 18:18:22 +0530 Generate reference bsv models for both rs*addr and immediates
2022-02-07 18:42:51 +0530 Added op3types, op4types, corrected bsv syntax in imm_dec_dict
2022-02-08 10:07:25 +0530 Move all optimised BSV functions to a single file in optbsv_dir
2022-02-08 10:12:47 +0530 Add header to ref.bsv
2022-02-08 11:22:33 +0530 Removal of \r characters
2022-02-08 11:27:15 +0530 Single package decode_opt in one file decode_opt.bsv
2022-02-08 12:31:53 +0530 Correct bsv errors
2022-02-08 12:33:11 +0530 Flattening a level
2022-02-08 12:38:37 +0530 Added enum encodings
2022-02-08 12:39:08 +0530 Removed aqrl from op3_type
2022-02-08 14:06:31 +0530 utils: Generate exhaustive reference BSV
2022-02-08 14:14:17 +0530 Merge branch 'rvp' of https://gitlab.com/incoresemi/ex-box/decode-box into rvp
2022-02-08 14:25:47 +0530 Extra brackets for syntax error
2022-02-08 14:29:53 +0530 Indentation fix
2022-02-08 14:51:49 +0530 Create pla for operand type decoding
2022-02-08 14:58:54 +0530 Correction of errors in op2_type
2022-02-08 16:05:06 +0530 move file pointer for decoder_opt.bsv as a class variable
2022-02-08 16:19:26 +0530 Create optimised BSV for operand types
2022-02-08 16:24:39 +0530 fix bsv syntax error in reference models
2022-02-08 16:39:33 +0530 update misa per instruction while checking legality
2022-02-08 16:54:21 +0530 Added schema to constants
```

```
2022-02-08 16:54:42 +0530 Added mechanism to validate filtered instr dict
2022-02-08 21:25:45 +0530 add constant space so the bsv is more readable
2022-02-08 21:26:19 +0530 fix for optypes in reference bsv creation
2022-02-08 23:09:45 +0530 Added rd_type decoding
2022-02-08 23:21:41 +0530 Indentation and formatting changes
2022-02-09 09:47:48 +0530 Add rdtype to reference BSV
2022-02-09 12:17:27 +0530 format using yapf and replace logger statements with lambdas
2022-02-09 14:54:48 +0530 Add lib_dir as a command line arguement and cleanup for utils
2022-02-09 15:11:36 +0530 Replace op_type in decoder_ref with enum encoding
2022-02-10 10:12:34 +0530 Add instruction name as a comment in the unoptimised pla file.
2022-02-10 13:05:04 +0530 Validate filtered_inst dictionary using a schema in src.constants
2022-02-10 20:54:41 +0530 Added fu_type
2022-02-10 23:32:56 +0530 Adding rv64i and modification to rv32zbb
2022-02-10 23:33:23 +0530 Added fu_type dictionary
2022-02-10 23:33:54 +0530 Added fu_type_mask generation
2022-02-10 23:39:52 +0530 Change instruction schema to accomodate for fu_type
2022-02-10 23:40:24 +0530 Added bsv generation for fu_type_mask
2022-02-10 23:40:55 +0530 fu_type change
2022-02-10 23:41:47 +0530 YAPF formatting
2022-02-11 10:41:22 +0530 add enum encodings to fu_types
2022-02-11 10:41:36 +0530 use enum_encodings to create fu_type_mask
2022-02-11 10:44:51 +0530 change type of fu_type in schema
2022-02-11 10:45:12 +0530 clean up of fu_type and it’s mask
2022-02-11 12:24:14 +0530 apply fu_type mask and decode fu_type_enum in decode_opt.bsv
2022-02-11 12:25:48 +0530 fix typo in rf_type in reference bsv
2022-02-11 12:43:01 +0530 add fu_type to reference.bsv
2022-02-11 12:57:32 +0530 Correct errors in decode_opt.bsv
2022-02-11 12:59:51 +0530 fn_dec_fu_type: declare mask before use
2022-02-11 13:09:19 +0530 dumo the filtered_inst as a yaml instead of txt
2022-02-11 13:25:16 +0530 Added f for fstring
2022-02-11 13:26:44 +0530 add lib_dir to all file headers
2022-02-11 13:41:04 +0530 add a check for missing fu_type
2022-02-11 17:00:21 +0530 Add mechanism to check instruction legality based on misa and mstatus
2022-02-11 17:24:37 +0530 Corrected BSV errors
2022-02-11 17:25:04 +0530 Merge branch ’rvp’ of https://gitlab.com/incoresemi/ex-box/decode-box into rvp
2022-02-14 09:39:42 +0530 src: constants: added some comments
2022-02-14 11:09:14 +0530 src: utils: create_bsv: replace empty expressions with 0
2022-02-14 11:14:37 +0530 Changed enum encodings and output size for pla and bsv generation
2022-02-14 11:31:05 +0530 fixes for bsc compile of reference decoder
2022-02-14 11:34:06 +0530 Fix bsv errors
2022-02-14 14:57:09 +0530 rename rf_type to rd_type
2022-02-14 14:57:29 +0530 let all the default types have an empty list
2022-02-14 14:58:03 +0530 default drive of output of don’t care.
2022-02-14 14:59:23 +0530 remove optype fields from ref_bsv.
2022-02-14 14:59:39 +0530 default immediate value to be 0.
2022-02-14 15:00:07 +0530 updated create_dec_optype to handle all operand types
2022-02-14 15:00:37 +0530 commented out futype functions
2022-02-14 15:03:27 +0530 remove enum_enc from operand_tyeps dict
2022-02-14 15:09:25 +0530 src: constants: replace ’.’ with ’_’ in instruction names
2022-02-14 15:16:11 +0530 Removing multiple searchings by using replace ’_’ and ’.’
2022-02-14 16:00:07 +0530 adding a bsv compile stage at the end
2022-02-14 16:54:36 +0530 src: utils: create reference bsv
2022-02-14 17:08:40 +0530 utils: fix typo in reference bsv
2022-02-14 18:02:29 +0530 clean up for logger statements
2022-02-14 20:03:15 +0530 reverse positions in enums and fields of pla
```

```
2022-02-14 20:35:33 +0530 fix addr_mask generation
2022-02-14 21:04:49 +0530 adding missing shamtw4 in list of immediates
2022-02-14 21:05:17 +0530 fix encoding for branch immediates
2022-02-14 21:08:43 +0530 reverse the immediate priority encoding pattern for correct behavior
2022-02-14 22:22:40 +0530 added fu_types to schema and changed dict structure
2022-02-14 22:23:19 +0530 utils: decode fu_types
2022-02-14 22:49:39 +0530 add instructions from rv_b
2022-02-14 22:50:53 +0530 Add instructions from rv_zbe
2022-02-14 22:51:43 +0530 Add instructions from rv_zbf
2022-02-14 22:53:09 +0530 Add instructions from rv_zbp
2022-02-14 22:55:02 +0530 Add instructions from rv_zbr
2022-02-14 22:55:47 +0530 Add instructions from rv_zbt
2022-02-14 22:57:30 +0530 Add instructions from rv32_zbp
2022-02-14 23:00:04 +0530 Add instructions from rv32_zbt
2022-02-14 23:01:03 +0530 Add instructions from rv64_zbe
2022-02-14 23:01:49 +0530 Add instructions from rv64_zbf
2022-02-14 23:04:15 +0530 Add instructions from rv64_zbm
2022-02-14 23:07:14 +0530 Add instructions from rv64_zbpbo
2022-02-14 23:08:06 +0530 Add instructions from rv64_zbr
2022-02-14 23:09:18 +0530 Add instructions form rv64_zbt
2022-02-14 23:10:08 +0530 src: constants: formatting to improve code readability
2022-02-14 23:14:53 +0530 Add instructions from rv64_zbp
2022-02-14 23:17:12 +0530 Add instructions from rv64_zbp to operand_types
2022-02-15 00:36:21 +0530 Added missing B extension instructions
2022-02-15 00:36:51 +0530 Added the calling of del_setup in the end
2022-02-15 10:30:23 +0530 utils: pseudo_ops: Replace '.' with '_' in instr name
2022-02-15 11:05:49 +0530 utils: atexit: handle file closure at exit
2022-02-15 11:45:56 +0530 utils: ref: add fu_types with enumeration
2022-02-15 11:59:15 +0530 utils: atexit: ensure file closure before checking bsv syntax
2022-02-15 12:02:29 +0530 call del_setup to close file before compiling bsv
2022-02-15 12:07:01 +0530 move fsw to memory
2022-02-15 12:08:39 +0530 move all Xret ops to system fu_type
2022-02-15 12:52:59 +0530 make trap default fu_type
2022-02-15 12:53:39 +0530 change decoder for immediate to leverage one-hot mask scheme
2022-02-15 12:56:01 +0530 utils: ref: Add default case
2022-02-15 12:58:07 +0530 utils: ref: use a local copy of filtered_inst
2022-02-15 13:16:03 +0530 Add XRF in op3_type
2022-02-15 13:16:20 +0530 utils: ref: add xrf in op3_type
2022-02-15 13:17:24 +0530 utils: ref: make a deep copy of the class variable
2022-02-15 13:18:02 +0530 src: yapf formatting to improve code readability
2022-02-15 14:03:49 +0530 remove duplicates from types and add a check for detecting duplicates
2022-02-15 14:31:35 +0530 cleanup ref-bsv creation
2022-02-15 14:35:29 +0530 default immediate to 0
2022-02-15 14:49:45 +0530 move types as a single dictionary
2022-02-15 15:01:09 +0530 unregister del_setup with atexit once it's been called
2022-02-15 15:08:07 +0530 add verify as a cli option
2022-02-15 15:16:11 +0530 move verify as a sub option
2022-02-15 15:18:45 +0530 main: generate: fix verify option checking in generate
2022-02-15 15:40:27 +0530 main: move --verify as a flag in CLI
2022-02-15 15:42:15 +0530 main: remove unused imports
2022-02-15 15:46:27 +0530 remove isem_configs
2022-02-15 15:51:29 +0530 sample_config: add types.yaml
2022-02-15 15:56:33 +0530 src: utils: comment out fu_type mask mechanism
2022-02-15 15:57:56 +0530 src: utils: remove all redundant functions
2022-02-15 16:00:23 +0530 main: cli: show default values for all options
```

```
2022-02-15 16:08:47 +0530 Removed types dictionary
2022-02-15 16:09:14 +0530 Added types as formal arguments
2022-02-15 16:10:40 +0530 Added YAML file as cli subcommand
2022-02-15 16:28:52 +0530 Removed parsing and made changes to typesfile subcommand
2022-02-15 16:29:40 +0530 Added YAML parsing in init and using types in ref_bsv, opt_bsv
2022-02-15 16:55:42 +0530 Changed option
2022-02-15 16:56:50 +0530 Changed shallow copy and header command
2022-02-15 17:03:48 +0530 staging a buggy but better version of instr legality
2022-02-15 17:06:42 +0530 check if bsc executable is available before using it.
2022-02-15 17:12:38 +0530 src: utils: prevent espresso errors for empty pla
2022-02-15 17:16:19 +0530 Revert "src: utils: prevent espresso errors for empty pla"
2022-02-15 17:30:50 +0530 Added node decode_types
2022-02-15 17:33:02 +0530 Logging types.yaml path and added node in yaml parsing
2022-02-16 14:14:55 +0530 Added test-vectors generation
2022-02-16 14:48:15 +0530 Changed generation of binary combinations, added no dont care support
2022-02-16 15:03:39 +0530 Handle bsc not found in PATH
2022-02-16 15:18:26 +0530 utils: Clean up for verify_bsv
2022-02-16 17:24:59 +0530 utils: ref: fix typo in if statements for r*addr
2022-02-16 17:36:24 +0530 Changed test vector generation
2022-02-16 18:00:44 +0530 Corrected list_index error and added comments
2022-02-16 18:16:46 +0530 utils: verify: log number of test vectors and cleanup
2022-02-16 18:42:05 +0530 Changed the number of bits for b type and s type
2022-02-16 22:50:41 +0530 Seperated out case for B type and S type, hence removing all duplicates
2022-02-16 23:16:01 +0530 Considering hi and lo as seperate fields
2022-02-17 10:34:10 +0530 ignore local .python-version file
2022-02-17 10:39:06 +0530 fix list append
2022-02-17 10:46:32 +0530 utils: write_pla: fix typo in logger statement
2022-02-17 11:21:35 +0530 docs: Initial commit
2022-02-17 11:40:57 +0530 docs: Initial commit
2022-02-17 13:01:53 +0530 docs: add some basic intro
2022-02-17 13:02:08 +0530 docs: add installation procedure
2022-02-17 13:31:55 +0530 Added commands in toctree
2022-02-17 13:32:24 +0530 Documentation for command of decode_box
2022-02-17 13:34:07 +0530 Modified list-extensions command to accomaodate for latest changes in decoder utils
2022-02-17 13:34:54 +0530 Made changes in del_setup and init to accomodate for list-extensions command
2022-02-17 14:13:29 +0530 docs: version: fix typos and links
2022-02-17 14:13:58 +0530 gitignore: track txt files
2022-02-17 14:14:26 +0530 docs: add requirements
2022-02-17 14:14:55 +0530 docs: conf: use PKG-INFO for version
2022-02-17 14:15:20 +0530 docs: minor cleanup
2022-02-17 14:15:58 +0530 add previously untracked package info
2022-02-17 14:16:54 +0530 docs: Makefile: change project name
2022-02-17 16:08:43 +0530 Split big lines into multiple lines to improve code readibility and clean
2022-02-17 16:53:24 +0530 docs: added tutorial
2022-02-17 17:03:14 +0530 Reformatting and improving content
2022-02-17 20:13:11 +0530 RST file for generation of test vectors for verify
2022-02-17 20:15:34 +0530 Changed formatting and added more description
2022-02-18 10:47:06 +0530 docs: add mechanism till decoding immediates
2022-02-18 12:11:46 +0530 docs: add mechanism
2022-02-18 14:42:43 +0530 docs: minor cleanup for commands
2022-02-19 11:23:41 +0530 src: fix multiple typos to improve code readability
2022-02-21 09:14:43 +0530 src: utils: add missing poitional arguement
2022-02-21 09:30:49 +0530 sample_config: types: fix typo in op2_type
2022-02-21 16:48:44 +0530 src: add support for instruction grouping in types.yaml
2022-02-21 16:56:43 +0530 mappings: move sgn, mv and class under MISC for F/D
```

```
2022-02-21 22:42:38 +0530 Added A,B,P extensions
2022-02-22 09:11:22 +0530 sample_config: remove op2_type for flw, fld
2022-02-22 09:30:56 +0530 docs: mechanism: add instruction grouping mechanism
2022-02-22 10:19:19 +0530 docs: add info about checked_types.yaml
2022-02-22 11:01:52 +0530 sample_config: add groupings.yaml
2022-02-22 12:03:08 +0530 sample_config: fix indentation in types.yaml
2022-02-22 12:14:21 +0530 Added unratified sub-extensions of B
2022-02-22 12:15:08 +0530 Sub-grouping of sub-extensions of B
2022-02-22 12:16:00 +0530 Added sub-extensions instead of instructions
2022-02-22 12:16:42 +0530 Corrected expansion errors
2022-02-22 12:51:57 +0530 Sorted types
2022-02-22 12:52:35 +0530 Corrections
2022-02-22 12:52:58 +0530 Generation of sorted checked_types
2022-02-22 12:57:28 +0530 Changed indentation
2022-02-22 12:57:49 +0530 Dumping into YAML files using different library
2022-02-23 10:50:38 +0530 update README.md
2022-02-23 11:01:32 +0530 readme: add steps to build documentation
2022-02-23 13:10:52 +0530 added CHANGELOG.md
2022-02-23 07:43:15 +0000 Merge branch ’rvp’ into ’dev’
2022-02-26 12:25:19 +0000 Pyeda espresso
2022-02-26 12:25:19 +0000 Merge branch ’pyeda_espresso’ into ’dev’
2022-02-27 13:53:18 +0000 Comment_correction
2022-02-27 13:53:18 +0000 Merge branch ’comment_correction’ into ’dev’
2022-03-02 11:53:29 +0000 Instruction legality
2022-03-02 11:53:30 +0000 Merge branch ’instruction_legality’ into ’dev’
2022-03-04 05:42:58 +0000 utils: create_inst_dict: use rv_system instead of opcodes-system
2022-03-04 05:42:58 +0000 Merge branch ’update-opcodes’ into ’dev’
2022-03-07 16:50:49 +0530 added restructured source code
2022-03-07 16:58:03 +0530 source: verify: use object notation of instr
2022-03-08 12:30:19 +0530 create RST table for encodings
2022-03-08 14:30:36 +0530 utilities: add fields in encodings rst
2022-03-08 14:36:45 +0530 minor cleanup
2022-03-08 14:37:06 +0530 yapf formatting
2022-03-08 14:55:34 +0530 verify: full: use obj notation
2022-03-09 16:58:35 +0530 populate arg lut with fields from compressed
2022-03-09 16:58:49 +0530 clean up support for new pseudo op format
2022-03-09 16:59:05 +0530 patch for utilities
2022-03-09 17:04:53 +0530 fix docs for getting version
2022-03-09 18:16:42 +0530 Added encodings rst generation
2022-03-09 18:20:03 +0530 docs: conf: add required extensions
2022-03-09 18:26:47 +0530 utilities: minor cleanup in rst generation
2022-03-09 18:27:53 +0530 utilities: fix error with write_bool
2022-03-09 18:30:30 +0530 decoder: minor cleanup for obj notation
2022-03-09 18:34:59 +0530 utilities: rst: use code syntax for instr name
2022-03-09 20:21:08 +0530 Fixed subscripting issue
2022-03-09 20:21:29 +0530 Fixed generation of encoding rst file
2022-03-10 09:29:44 +0530 populate arg lut with fields from compressed
2022-03-10 09:40:53 +0530 minor cleanup in utils
2022-03-10 10:20:18 +0530 utilities: use _ in instruction names
2022-03-10 11:59:15 +0530 utilities: redirect error streams as warnings in the logger
2022-03-10 12:55:34 +0530 dumpo all fields into instr yaml
2022-03-10 12:58:12 +0530 wavedrom format for encodings table
2022-03-10 13:30:08 +0530 fix malformed rst table
2022-03-10 13:31:33 +0530 docs: add empty encodings.rst
2022-03-10 13:31:53 +0530 create docs in generate
```

```
2022-03-10 13:36:31 +0530 disable automatic generation of docs
2022-03-10 13:37:05 +0530 fix instruction names in encodings
2022-03-10 13:41:11 +0530 move encodings.rst to build/misc
2022-03-10 13:45:29 +0530 encodings: replace . with _ in instr names
2022-03-10 13:52:24 +0530 utils: rst: hardwire max cols to 80 for encodings
2022-03-10 13:53:25 +0530 minor cleanup and yapf formatting
2022-03-10 19:35:35 +0530 utilities: rst: color the fixed bits in wavedrom
2022-03-10 20:31:11 +0530 log: use stdout instead of stderr for printing logs
2022-03-11 11:24:43 +0530 Corrected aq,rl bits for encodings rst
2022-03-11 11:30:25 +0530 add all hidden files in gitignore
2022-03-11 12:06:34 +0530 minor cleanup in main
2022-03-11 13:20:50 +0530 utilities: cleanup and beter docstrings
2022-03-11 13:33:53 +0530 minor cleanup in decoder
2022-03-11 13:34:17 +0530 initialize all fields of the instruction
2022-03-11 14:11:44 +0530 added sanity checks for each instruction
2022-03-11 14:17:08 +0530 add address mask to the Instruction
2022-03-11 14:18:18 +0530 add sanity check for address mask
2022-03-11 15:47:10 +0530 decode and check trapcause values
2022-03-11 15:58:14 +0530 fix default trapcause value
2022-03-14 09:03:05 +0530 skip making array syntax in BSV constants
2022-03-14 09:45:27 +0530 handle environment calls based on prv
2022-03-14 11:39:57 +0530 fix error in trapcause values for PLA
2022-03-14 11:40:15 +0530 move logger statements into debug
2022-03-14 11:58:40 +0530 fix for variables in pla name
2022-03-14 12:07:31 +0530 Replaced 'b by just number to fix bsc errors
2022-03-14 12:08:02 +0530 YAPF formatting
2022-03-15 10:02:13 +0530 update docs
2022-03-15 10:02:47 +0530 minor cleanup
2022-03-15 10:42:31 +0530 delete the old source tree
2022-03-15 10:43:36 +0530 minor cleanup
2022-03-15 10:48:59 +0530 yapf formatting
2022-03-15 10:59:31 +0530 Update CHANGELOG.md
2022-03-15 11:04:58 +0530 Bump version 0.1.1-restructuring -> 0.1.2
2022-03-15 12:31:05 +0530 log on debug level instead of info for test vec generation
2022-03-15 18:12:30 +0530 added mem_type decoding
2022-03-15 18:40:03 +0530 added mem_type to ref_bsv
2022-03-15 20:08:56 +0530 Added comments
2022-03-16 11:10:09 +0530 Added sanity checks for memory_mask
2022-03-16 12:06:16 +0530 decoder: mem_types within dec_types
2022-03-16 12:23:35 +0530 correction in ref_bsv generation
2022-03-16 13:04:55 +0530 consistency in refbsv and optbsv enum for mem_type
2022-03-16 13:21:17 +0530 removed sorted for types
2022-03-16 16:33:50 +0530 instr: add utility functions for each instr
2022-03-16 16:34:57 +0530 const: add recoded op dict
2022-03-16 16:35:18 +0530 decode recoded op mask and make BSV + minor cleanup
2022-03-16 16:35:35 +0530 main: invoke decoding recoded op
2022-03-16 16:38:44 +0530 utilities: minor cleanup with spacing
2022-03-16 16:53:17 +0530 work in progress: ref bsv generation for recoded_op
2022-03-16 21:56:52 +0530 Added dec_word32 todo: verification.
2022-03-16 22:00:46 +0530 add comments and correct indentation for write_bsv
2022-03-17 08:55:47 +0530 fix type name assignment
2022-03-17 09:12:39 +0530 cleanup for dec_word32
2022-03-17 09:12:55 +0530 added sanity check for word32 op
2022-03-17 09:16:36 +0530 yapf formatting for better code readability
2022-03-17 09:34:49 +0530 legality, trapcause, word32 in ref bsv
```

```
2022-03-17 09:56:44 +0530 move rec_op masks to constants
2022-03-17 09:56:55 +0530 recoded_op in ref_bsv
2022-03-17 10:00:41 +0530 fix pla type for trapcause
2022-03-17 13:02:21 +0530 minor clean up
2022-03-17 13:02:50 +0530 add docs for recoded_op, mem_access and word32 status
2022-03-17 17:03:50 +0530 major cleanup
2022-03-17 17:30:16 +0530 update changelog
2022-03-22 04:55:06 +0000 Merge branch 'restructuring' into 'dev'
2022-04-01 09:34:30 +0530 add png versions of all drawio images
2022-04-01 09:37:07 +0530 use image directive instead of drawio-image
2022-04-01 10:39:34 +0530 Bump version 0.1.2 -> 0.1.3
2022-04-01 10:41:06 +0530 update CHANGELOG
```

7.3 Activity log 3 - Verification of RISC-V Bitmanip hardware IPs

```
2021-10-29 15:49:13 +0000 Initial commit
2021-10-29 21:23:00 +0530 initial verif framework
2021-10-29 15:55:07 +0000 Update README.md
2021-10-29 15:55:41 +0000 Update README.md
2021-11-13 19:27:00 +0530 Changes to alias signal
2021-11-29 13:02:41 +0530 remove unwanted imports
2021-12-05 00:47:08 +0530 DUT enable signal fix in test
2021-12-07 19:54:08 +0530 Adding input and output handling as co-routines in CoCoTb env.
2021-12-10 10:37:27 +0530 Fix for XLEN
2021-12-10 10:40:31 +0530 Fixing test_Zbc.py file
2021-12-10 11:01:04 +0530 Reverting "info" to "debug" in out_monitor
2021-12-13 12:32:10 +0530 adding coverage template
2021-12-16 12:26:12 +0530 issue_template to file bugs
2022-02-11 18:50:30 +0530 Pythonic reference model is verified against Spike model
2022-02-12 00:21:15 +0530 Updated valid_rdy signal as changed in the DUT.
2022-02-17 11:22:14 +0530 fix for bitwalker
2022-02-28 10:10:54 +0530 design models per sub-extension (Zbb-incomplete)
2022-03-22 11:54:12 +0530 work in progress: models: add reference models
2022-03-22 13:08:38 +0530 models: add generalised shuffle instructions
2022-03-22 13:13:39 +0530 minor cleanup
2022-03-22 14:37:47 +0530 models: added crossbar permutation instructions
2022-03-22 14:56:18 +0530 models: added generalized or-combine
2022-03-22 14:56:37 +0530 minor cleanup
2022-03-22 16:49:42 +0530 Work in progress: added andn, orn, xnor, pack*
2022-03-22 17:28:43 +0530 models: zbp: fix conditions to select instruction
2022-03-22 18:24:24 +0530 stop tracking hidden files and directories
2022-03-22 18:24:56 +0530 select only zbp tests
2022-03-22 18:25:10 +0530 fix return type causing error in cocotb
2022-03-22 18:25:29 +0530 add empty test for zbp
2022-03-22 18:25:50 +0530 select zbp in test_list.yaml
2022-03-23 14:59:22 +0530 added zbp instructions
2022-03-23 14:59:39 +0530 test_Zbp: added round robin test
2022-03-23 16:40:49 +0530 send and validate src* separately
2022-03-23 16:40:53 +0530 fix reference model for gorc32/64
2022-03-23 17:48:01 +0530 models: gorci32/64: fix funct in condition
2022-03-24 15:28:58 +0530 Modification of march string in spike
2022-03-24 16:43:40 +0530 fix for imm type instructions and their shamt
2022-03-24 18:25:33 +0530 coverage: immidiates in g*i instructions
2022-03-25 09:59:12 +0530 move test specific information from constants
2022-03-25 09:59:20 +0530 remove constants
2022-03-25 10:45:35 +0530 test for walking zeroes in
2022-03-25 12:08:50 +0530 test for grev’s bit retention property
2022-03-25 12:12:52 +0530 coverage: fix var in transfer function
2022-03-25 12:24:58 +0530 Added bit_marcher to generate marching patterns
2022-03-25 12:25:23 +0530 Added bit walker for rs1/op1
2022-03-25 12:25:52 +0530 Added bit marcher test for rs1/op1
2022-03-25 12:27:53 +0530 Added bit walker to rs2
2022-03-25 12:28:34 +0530 Added bit marcher to rs2
2022-03-25 15:41:08 +0530 Added all 1s test for logical with negate
2022-03-25 15:41:44 +0530 Added all 0s test for logical with negate
2022-03-25 15:44:30 +0530 Added all alternating 1s and 0s test for logical with negate
2022-03-25 15:45:41 +0530 Corrected march bit_width to 32 for rs1 and rs2
2022-03-25 16:06:37 +0530 coverage: outer bit retention for (un)shuffle
2022-03-26 01:21:49 +0530 zip_outer_bits: remove redndant entries from cover point bins
```

```
2022-03-28 10:23:52 +0530 minor cleanup
2022-03-28 11:38:33 +0530 coverage for walking for rs1
2022-03-28 11:38:33 +0530 coverage for walking for rs2
2022-03-28 11:38:33 +0530 minor cleanup
2022-03-28 11:38:33 +0530 coverage for marching for rs1
2022-03-28 11:38:33 +0530 coverage for marching for rs2
2022-03-28 11:38:33 +0530 coverage-for-all-1s-for-logical-negate-instructions
2022-03-28 11:38:33 +0530 coverage-for-all-0s-for-logical-negate-instructions
2022-03-28 11:38:33 +0530 coverage-for-alternate-1s-and-0s-for-logical-negate
2022-03-28 11:40:08 +0530 Update todo
2022-03-28 12:28:40 +0530 minor cleanup
2022-03-28 12:37:15 +0530 Minor cleanup
2022-03-28 12:39:36 +0530 Minor cleanup
2022-03-28 14:24:56 +0530 utils: add patter_walk utility for generating input patterns
2022-03-28 14:38:10 +0530 add pattern entry sequence in pattern_walk
2022-03-28 15:01:57 +0530 Partial correction to models
2022-03-28 15:02:21 +0530 Inputs based on XLEN
2022-03-28 15:04:31 +0530 Minor cleanup
2022-03-28 15:18:05 +0530 corrected bit width
2022-03-28 15:48:16 +0530 Correction in grev
2022-03-28 16:43:58 +0530 fix models for grev32/64
2022-03-28 16:44:16 +0530 fix tests for r* walking/marching
2022-03-28 16:48:43 +0530 fix models for shfl*
2022-03-28 17:22:54 +0530 fix all reference models
2022-03-28 17:42:30 +0530 enable all tests
2022-03-28 18:02:06 +0530 minor cleanup
2022-03-29 10:45:47 +0530 fix reference models for rotate instructions
2022-03-29 11:13:13 +0530 Fixed bracket issue, corrected models for shfl
2022-03-29 11:17:45 +0530 Minor cleanup
2022-03-29 11:36:04 +0530 Added user_config.py
2022-03-29 11:39:58 +0530 Revert "Added user_config.py"
2022-03-29 12:08:14 +0530 Added function swap_n_bits
2022-03-29 12:16:39 +0530 fix ref models for generalized reverse
2022-03-29 13:31:42 +0530 add or-combine utility
2022-03-29 13:32:01 +0530 fix irc models and minor cleanup
2022-03-29 13:32:19 +0530 enable only round robin test
2022-03-29 16:00:47 +0530 coverage: add out of bounds condition for crossbar perm instructions
2022-03-29 16:00:58 +0530 minor cleanup in models
2022-03-29 22:48:55 +0530 Corrected or_n_bits
2022-03-30 02:42:52 +0530 Correction to gorc* model, verif req
2022-03-30 03:22:09 +0530 Minor cleanup
2022-03-30 03:22:37 +0530 Fixed logical not, pack* model
2022-03-30 03:43:26 +0530 Corrected models for rori, roriw
2022-03-30 04:11:26 +0530 Removed print, shfl model progress
2022-03-30 10:33:56 +0530 models: fix encoding for packu
2022-03-30 11:39:05 +0530 fix xperm_h models
2022-03-30 11:47:17 +0530 fix crossbar permutation ref models
2022-03-30 11:58:09 +0530 fix gorc models and minor cleanup
2022-03-30 12:17:16 +0530 fix grev* ref models
2022-03-30 12:29:15 +0530 fix models for *shfli*
2022-03-30 13:00:33 +0530 minor cleanup
2022-03-30 15:36:47 +0530 Added xlen check
2022-03-30 15:39:28 +0530 Added xlen check for all test functions
2022-03-30 15:40:13 +0530 Added xlen check
2022-03-30 18:26:20 +0530 fix orc, rev (i)w models
```

```
2022-03-31 10:45:16 +0530 fix pack*w ref models
2022-03-31 10:58:41 +0530 fix (un)shfl*w models
2022-03-31 11:08:11 +0530 coverage: add bins for round robin test
2022-03-31 11:11:12 +0530 Fix for rolw rorw roriw
2022-03-31 11:22:27 +0530 test: imm_walking: disable walking over the bit 26
2022-03-31 11:27:42 +0530 enable all tests
2022-03-31 11:59:47 +0530 test: add test for pack_carry_bit
2022-03-31 12:03:00 +0530 coverage: add pack_carry_bits cover point
2022-03-31 12:24:48 +0530 Soak test
2022-04-01 19:47:40 +0530 Added instruction data logger
2022-04-01 20:55:03 +0530 Completely exhaustive test
2022-04-03 20:52:55 +0530 test: sign/zero extension for pack* instructions
2022-04-04 09:52:09 +0530 move test utils into helper
2022-04-04 09:52:32 +0530 use helper.py instead of utils
2022-04-04 10:53:53 +0530 coverage: fix coverpoint for imm_walking test
2022-04-04 11:03:33 +0530 coverage: fix bins for pack_carry_bit test
2022-04-04 11:11:41 +0530 coverage: fix cover points for grev tests
2022-04-04 11:22:53 +0530 coverage for soak test
2022-04-04 12:14:28 +0530 fix docstring for xperm test
2022-04-04 12:15:26 +0530 minor cleanup for better code readability
2022-04-04 12:30:09 +0530 enable all tests
2022-04-04 12:50:50 +0530 minor cleanup
2022-04-04 13:20:25 +0530 Fix coverage for rs1 walking
2022-04-04 13:37:05 +0530 Fix for exhaustive test
2022-04-04 16:51:19 +0530 work in progress: test for in order execution
2022-04-04 21:19:18 +0530 Adding ready signal of DUT to cocotb environment
2022-04-04 22:36:09 +0530 Replaced test_vecs with instr
2022-04-05 10:01:20 +0530 Changed yield RisingEdge(tb.dut.CLK)
2022-04-05 15:21:47 +0530 Correction in round robin loop variable
2022-04-05 16:01:50 +0530 Minor cleanup and logging instruction name
2022-04-05 16:12:06 +0530 add mechanism to check order of outputs from DUT
2022-04-05 16:44:12 +0530 don’t wait for outputs
2022-04-05 16:51:08 +0530 minor cleanup
2022-04-05 19:32:13 +0530 Grouping of instructions
2022-04-06 10:14:59 +0530 set-wise selection of instructions for in-order test
2022-04-06 12:27:03 +0530 minor cleanup
2022-04-06 16:18:30 +0530 enable coverage sectiona based on test_list
2022-04-06 17:28:01 +0530 Checking if dut is ready inside innermost loop
2022-04-07 11:37:53 +0530 drive ma_request_instr signal along with inputs in inner loop and minor cleanup
2022-04-07 12:13:06 +0530 Add documentation for Zbp tests
2022-04-08 11:33:38 +0530 Fix coverage for rs1 walking
2022-04-08 11:34:37 +0530 Fix coverage for soak test
2022-04-08 11:39:47 +0530 Fix coverage for rs2 walking
2022-04-08 12:31:54 +0530 Selective test
2022-04-08 12:34:52 +0530 Updated docstring
2022-04-08 13:30:30 +0530 Change invert to false for coverage fix
2022-04-11 11:26:33 +0530 wip: cycle count verif
2022-04-11 11:53:33 +0530 reset clock cycles in imon
2022-04-11 12:14:57 +0530 Revert "reset clock cycles in imon"
2022-04-11 12:15:11 +0530 Revert "wip: cycle count verif"
2022-04-11 15:49:33 +0530 Add utility to compute required cycles for each instruction based on hw_config
2022-04-11 16:31:34 +0530 Added module name in logger
2022-04-11 18:28:13 +0530 Cycle count
2022-04-12 10:41:20 +0530 Coorected assertion error and removed print
2022-04-12 10:43:11 +0530 Setting clock cycles to 0 at drive_inputs
```

```
2022-04-12 11:20:32 +0530 wip: clock_cycle_counter
2022-04-12 12:01:27 +0530 debugging: simply print clock edges
2022-04-12 16:07:25 +0530 add mechanism to cycle_count
2022-04-12 17:10:42 +0530 utils: fix expected cycle count
2022-04-12 17:11:50 +0530 utils: cycle_count: return 1 cycle by default
2022-04-12 17:14:37 +0530 test: set recv_output flag to false after check
2022-04-12 17:38:19 +0530 add clock_cycle counter to all tests and minor cleanup
2022-04-12 17:54:00 +0530 return 1 as default clk count
2022-04-12 17:58:23 +0530 minor fix in exp cycle count
2022-04-13 09:53:24 +0530 add missing clock counter in soak test
2022-04-13 10:14:48 +0530 Fix for in-order execution
2022-04-13 10:16:45 +0530 cleanup the logger region
2022-04-13 10:25:00 +0530 Added docstring for instr_map
2022-04-13 12:45:27 +0530 change exceptions into test failures
2022-04-18 10:28:45 +0530 Fix of march/mabi combination
2022-04-18 12:44:17 +0530 models: fix gorci32-32 model
2022-04-18 12:54:37 +0530 WIP: coverage based on XLEN
2022-04-18 13:09:16 +0530 minor fix for using XLEN in coverage
2022-04-18 14:04:58 +0530 coverage for 32 bit. TODO: pack_carry_bit, xperm_rs2_out_of_bounds
2022-04-18 15:40:40 +0530 Fix for 'bins' addition; 32 bit cov for xperm, pack
2022-04-18 16:18:29 +0530 fix overflow in pack_carry_bits for 32-bit test
2022-04-18 16:25:17 +0530 wait until all expected outputs are recieved
2022-04-19 10:44:46 +0530 avoid walking over all bits in imm fields
2022-04-19 12:10:15 +0530 fix coverage bins to comply with changes in da3a9a02
2022-04-19 12:24:09 +0530 coverage: fetch correct bins
2022-04-19 13:08:05 +0530 Moved instruction dictionary and instr_map to helper.py
2022-04-19 13:35:46 +0530 Changed to ref_model_results to accomodate for clock+exp_clock
2022-04-19 13:36:07 +0530 Added clock+exp_clock to ref_model_results
2022-04-19 13:36:50 +0530 Changed for clock_counter in ref_model_results
2022-04-19 13:39:59 +0530 Removed redundancy in cocotb_env for append
2022-04-19 14:50:09 +0530 fix bins32 for xperm rs2 in bounds test
2022-04-19 15:23:03 +0530 Set clock to 0 in init
2022-04-19 15:23:47 +0530 Instruction hex related functions/dictionaries
2022-04-19 15:24:16 +0530 Moved instr_map
2022-04-19 15:24:26 +0530 Moved get_cycles
2022-04-19 15:24:46 +0530 Import encodings
2022-04-19 15:25:17 +0530 Merge branch 'zbp' of https://gitlab.com/incoresemi/ex-box/b-box-verif into zbp
2022-04-19 15:36:12 +0530 Changed file name
2022-04-19 15:37:43 +0530 changed import
2022-04-19 15:38:11 +0530 changed import
2022-04-20 12:03:10 +0530 tests: enable all tests except exhaustive_test
2022-04-20 12:03:34 +0530 yapf formatting for better code readability
```

References

[1] Christopher Drake. “PyEDA: Data Structures and Algorithms for Electronic Design Automation”. In: Jan. 2015, pp. 25–30. DOI: `10.25080/Majora-7b98e3ed-004`.

[2] Neel Talakshi Gala. 2022. URL: `https://github.com/incoresemi/riscv-opcodes/tree/migration-to-new-format`.

[3] Neel Talakshi Gala. *RISC-V AAPG*. URL: `https://gitlab.com/shaktiproject/tools/aapg`.

[4] Neel Talakshi Gala. *RiVer Core Verification Framework*. URL: `river-core.readthedocs.io`.

[5] Neel Talakshi Gala and Pawan Kumar. *RISC-V Compatibility Test Generator*. URL: `https://riscv-ctg.readthedocs.io/`.

[6] Neel Talakshi Gala, Alenkruth K Murali, and Purushothaman Palani. *UATG: A python-based open source microarchitectural test generation framework for RISC-V functional verification*. URL: `https://uatg.readthedocs.io/`.

[7] Neel Talakshi Gala et al. *Chromite Core Generator*. URL: `chromite.readthedocs.io`.

[8] Chris Higgs. *CoCoTb: a COroutine based COsimulation TestBench environment for verifying VHDL and SystemVerilog RTL using Python*. 2014. URL: `docs.cocotb.org`.

[9] Phillipp Tomsich and Andrew Waterman. *RISC-V Bitmanip Extension*. June 2021. URL: `https://github.com/riscv/riscv-bitmanip`.

Signature Certificate

Reference number: JDJHA-KFHC5-REDPE-K5IBJ

Signer	Timestamp	Signature
Neel Gala Email: neelgala@incoresemi.com Sent: Signed:	 25 Apr 2022 07:52:52 UTC 25 Apr 2022 07:52:52 UTC	Neel Gala IP address: 49.207.204.62 Location: Bengaluru, India

Document completed by all parties on:

25 Apr 2022 07:52:52 UTC

Page 1 of 1