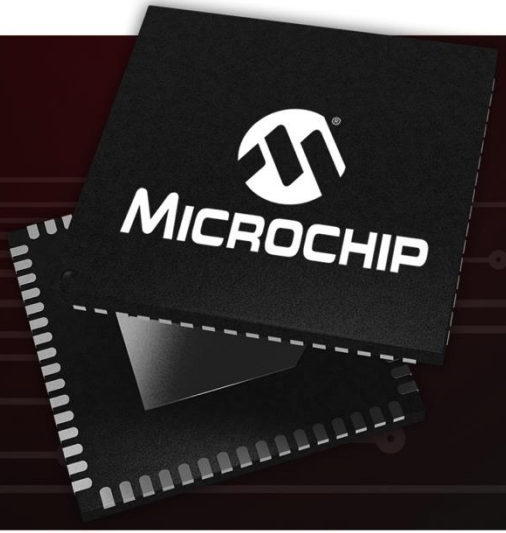
MICROCHIP



A Leading Provider of Microcontroller, Security, Mixed-Signal, Analog & Flash-IP Solutions




Building Better Soft RISC-V IP Cores Through Mi-V™ Verification and Compliance Testing

RISC-V Workshop Zurich, 2019

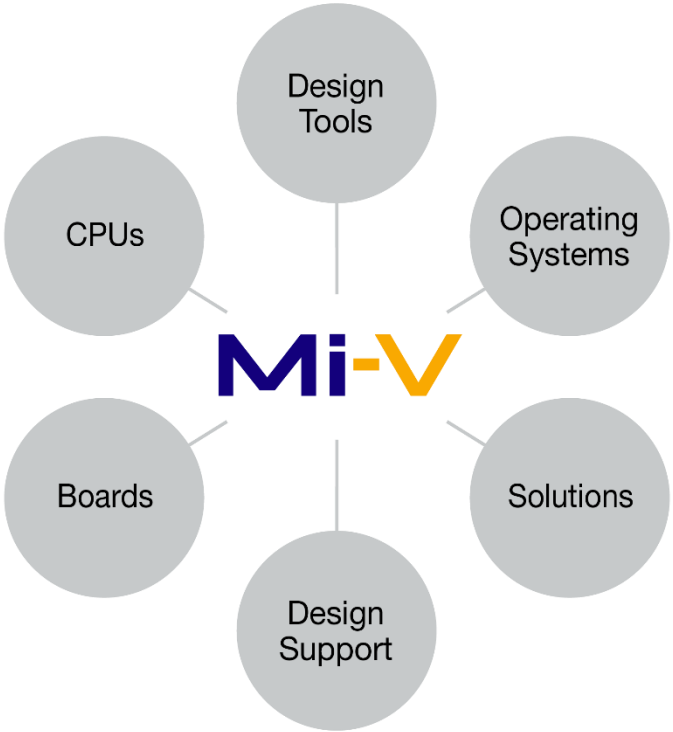
Stuart Hoad & Hugh Breslin

Overview

- Introduction to Mi-V™
- Why we need a compliance and verification platform
- Cross platform verification and compliance testing of Mi-V processor cores
- Applying formal model checking to core verification

Mi-V RISC-V Ecosystem

- **A continually expanding, comprehensive suite of tools and design resources to fully support RISC-V designs**
- **Aims to increase adoption of RISC-V ISA and Microchip's soft CPU product family**
- **Supports development using Microchip’s soft-CPUs and RISC-V SoC FPGAs**



Mi-V Compliance and Verification

•We'd like to be able to integrate and use open-source cores.

- In some cases, these have unknown verification and unknown compliance to the RISC-V specification(s).
- Whilst the base cores are chosen with some (varying) level of verification expected, we may need to make changes/additions, so these must be verified.

•We'd like all cores to be compliant.

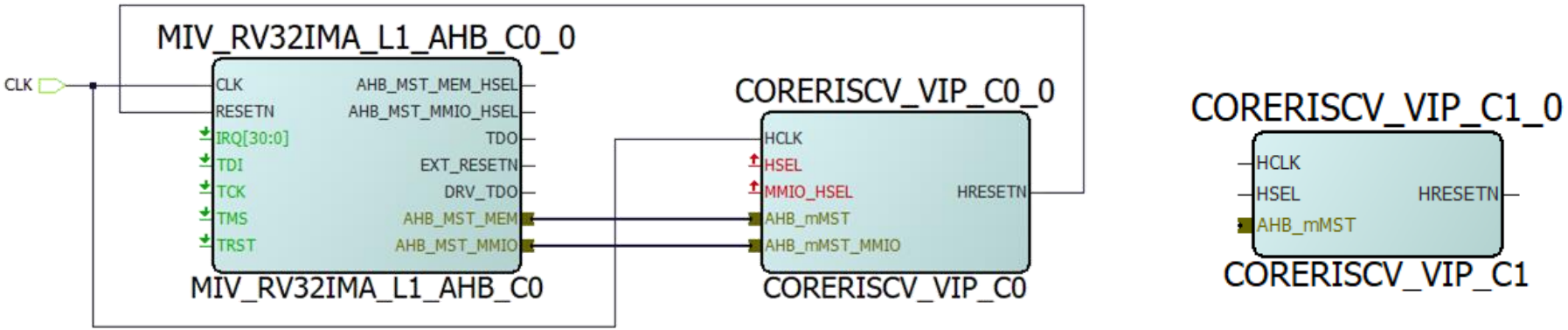
- We would like all cores to be interoperable with all tools and all other cores, therefore they must be compliant.
- Microchip contributes to the compliance working group.
- This is an opportunity to test out the compliance suite across multiple platforms and improve it as part of the ongoing development of RISC-V.
- Propose methodology for user and privileged "super-compliance."

Mi-V Compliance and Verification

- **RISC-V VIP**
  - Compliance
  - Torture
  - Simulation environment
- **Mi-V Compliance Suite**
  - Compliance
  - Hardware/emulation/simulation environment
- **Benchmarks**

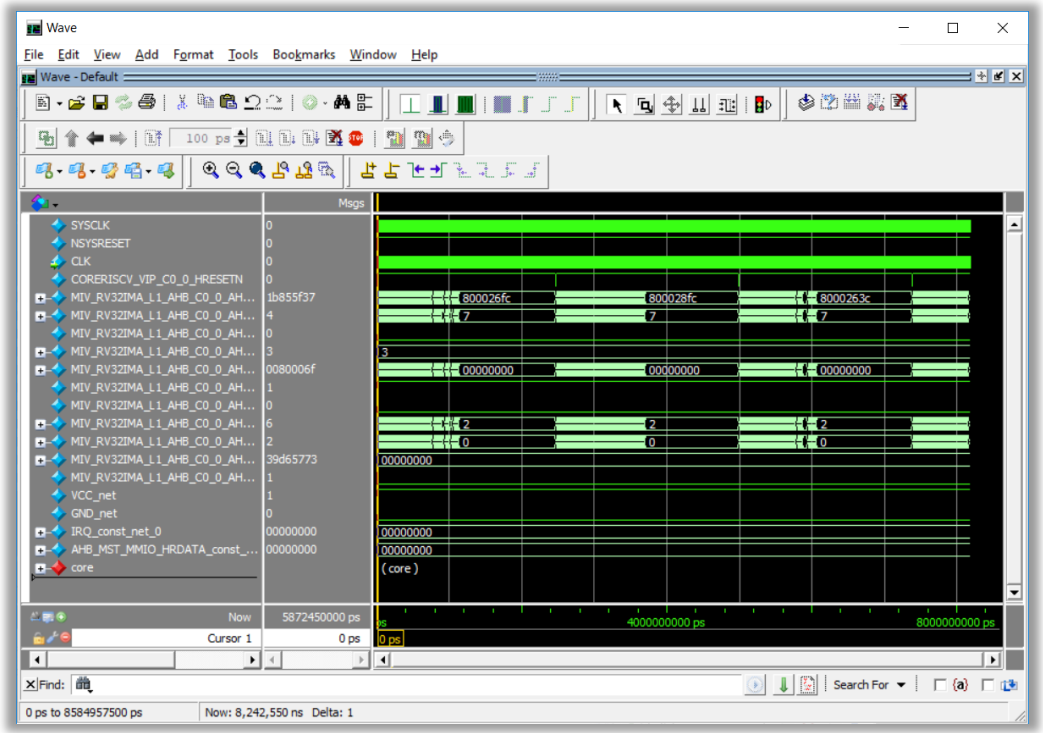
RISC-V VIP

- **Uses an AHB interface**
- **Can pass selections of test hex files to the core and compare the resulting memory dump with reference signatures**

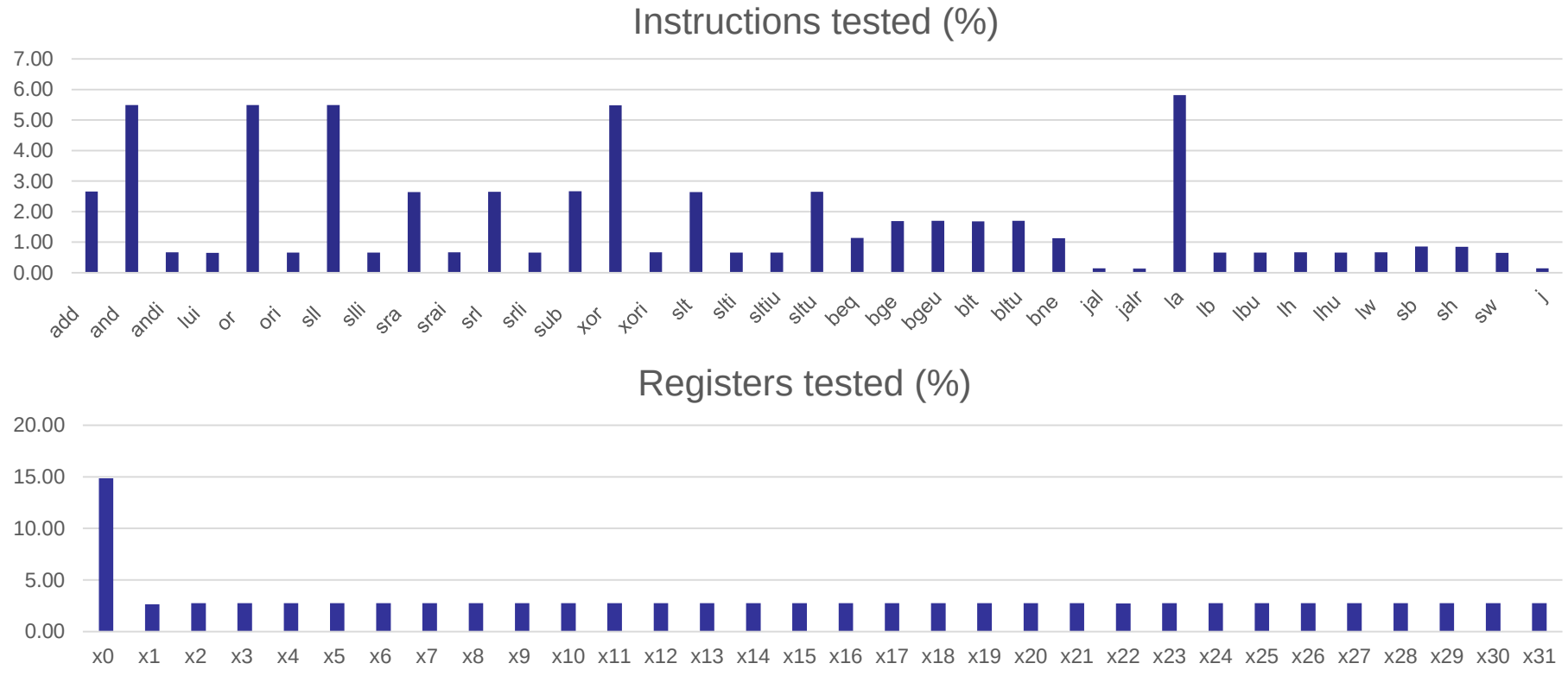


RISC-V VIP

- **Can be configured to run**
  - Compliance tests
  - Torture tests
- **Compliance tests**
  - RISC-V compliance GitHub
  - Support for I, M and C extensions
- **Torture tests**
  - Generated from Clifford Wolf's scripts
  - Support for I, M & C extensions
  - 1000 tests for 8 configurations: 8000 tests
  - Coverage



RISC-V VIP


Instructions tested (%)
7.00
6.00
5.00
4.00
3.00
2.00
1.00
0.00
add
and
andi
lui
or
ori
sll
slli
sra
srai
srl
srli
sub
xor
xori
slt
slti
sltiu
sltu
beq
bge
bgeu
blt
bltu
bne
jal
jalr
la
lb
lbu
lh
lhu
lw
sb
sh
sw
j
Registers tested (%)
20.00
15.00
10.00
5.00
0.00
x0
x1
x2
x3
x4
x5
x6
x7
x8
x9
x10
x11
x12
x13
x14
x15
x16
x17
x18
x19
x20
x21
x22
x23
x24
x25
x26
x27
x28
x29
x30
x31

Mi-V RISC-V Compliance

- **Make each test a function, and call it from a main**
- **Use global defines to enable/disable tests**

*main.S

```
180     Individual test call examples, the ISA for these tests must be defined in "src/compliance_defines.h":
181     call ADD_TEST
182 */
183
184 call ADD_TEST
185
186 /*
187 * Test calls below are based on extensions defined in compliance_defines.h
188 * ITERATIONS is defined in compliance_defines.h
189 */
190     li x6, ITERATIONS
191     li x7, 0
192
193     1:
194
195     #ifdef I_EXTENSION_TESTS
196
197         call ADD_TEST
198         call ADDI_TEST
199         call AND_TEST
200         call ANDI_TEST
201         call AUIPC_TEST
202         call BEQ_TEST
203         call BGE_TEST
```

I-ADD-01_converted.S

```
31
32 #include "compliance_defines.h"
33
34 #ifndef ADD_S
35
36     #define ADD_S
37
38     .global registers
39     .global ADD_TEST
40
41     .extern print_signatures
42     .extern endTest
43
44     ADD_TEST:
45
46     mv  x25,    x1
47     call    save_state;
48
49     #ifdef SIM_TEST
50         li  x30,    0xFFFF0000
51         sw  x30,    (x31)
52     #endif
53
54     li  x30,    0x00000001
55     sw  x30,    (x31)
56
57     #ifdef HW_TEST
58       call    testNumber
59     #endif
60
61     sw  x0, (x31)
62
63     # Load testdata
```

Mi-V RISC-V Compliance

- **Create a buffer and print the content at the end of each test to minimize the size**

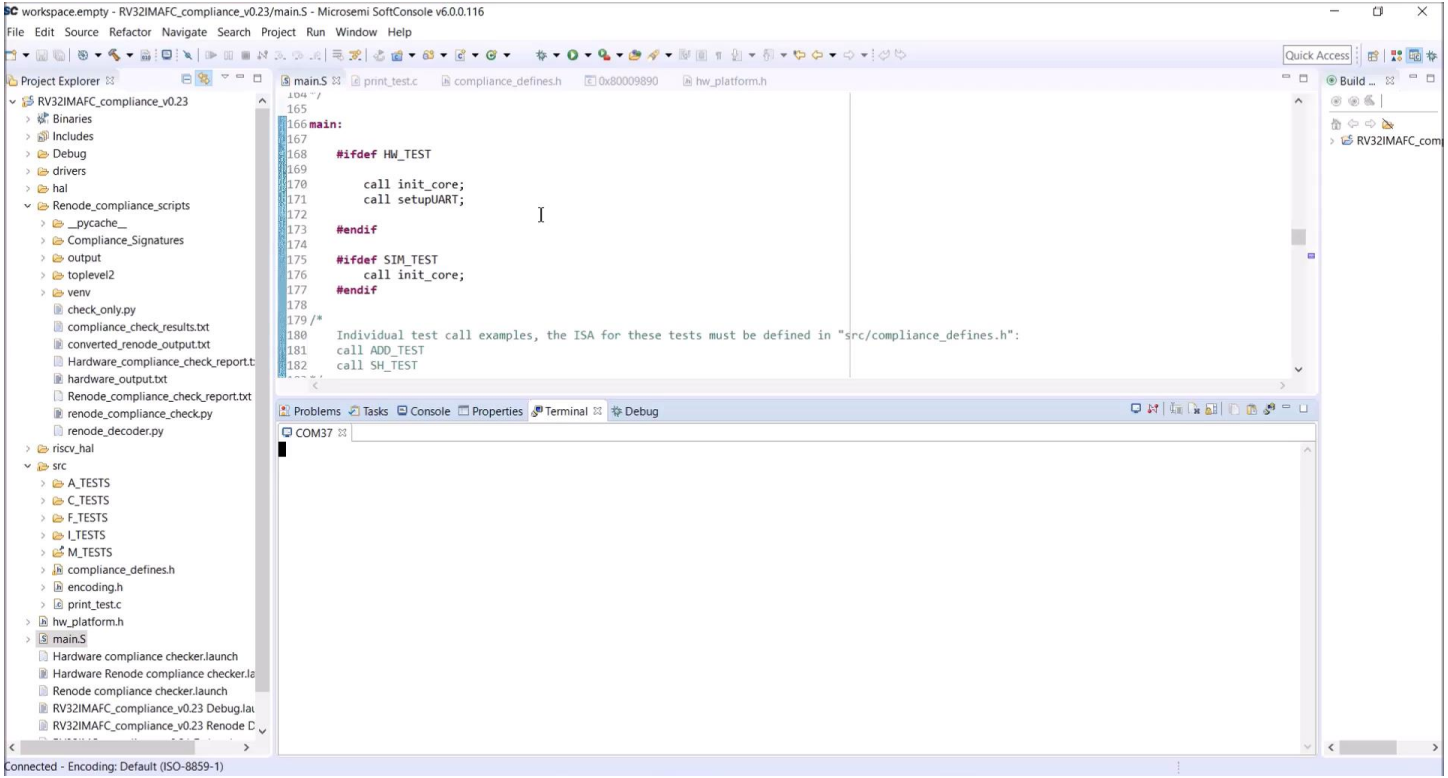
```
15
16  OUTPUT_ARCH( "riscv" )
17  ENTRY(_start)
18
19
20  MEMORY
21  {
22      ram (rwx) : ORIGIN = 0x80000000, LENGTH = 120k
23      uart (rwx) : ORIGIN = 0x80019000, LENGTH = 2K
24      results (rwx) : ORIGIN = 0x60000000, LENGTH = 1K
25  }
26
27  RAM_START_ADDRESS    = 0x80000000;        /* Must be the same
28  RAM_SIZE             = 120k;               /* Must be the sam
29  STACK_SIZE           = 2k;                /* needs to be calc
30  HEAP_SIZE            = 2k;                /* needs to be calc
31  UART_SIZE            = 1K;
32  REGISTER_SAVE_SIZE   = 1K;
33  RESULTS_SIZE         = 1K;
34
35  SECTIONS
36  {
37    .text : ALIGN(0x10)
38    {
```

```
62   */
63
64  #ifdef HW_TEST
65  #define __STORE_LOC __buffer_start // defines store location a
66  #endif
67
68  #ifdef SIM_TEST
69  #define __STORE_LOC  __signatures_start // defines store locat
70  #endif
71
72
```

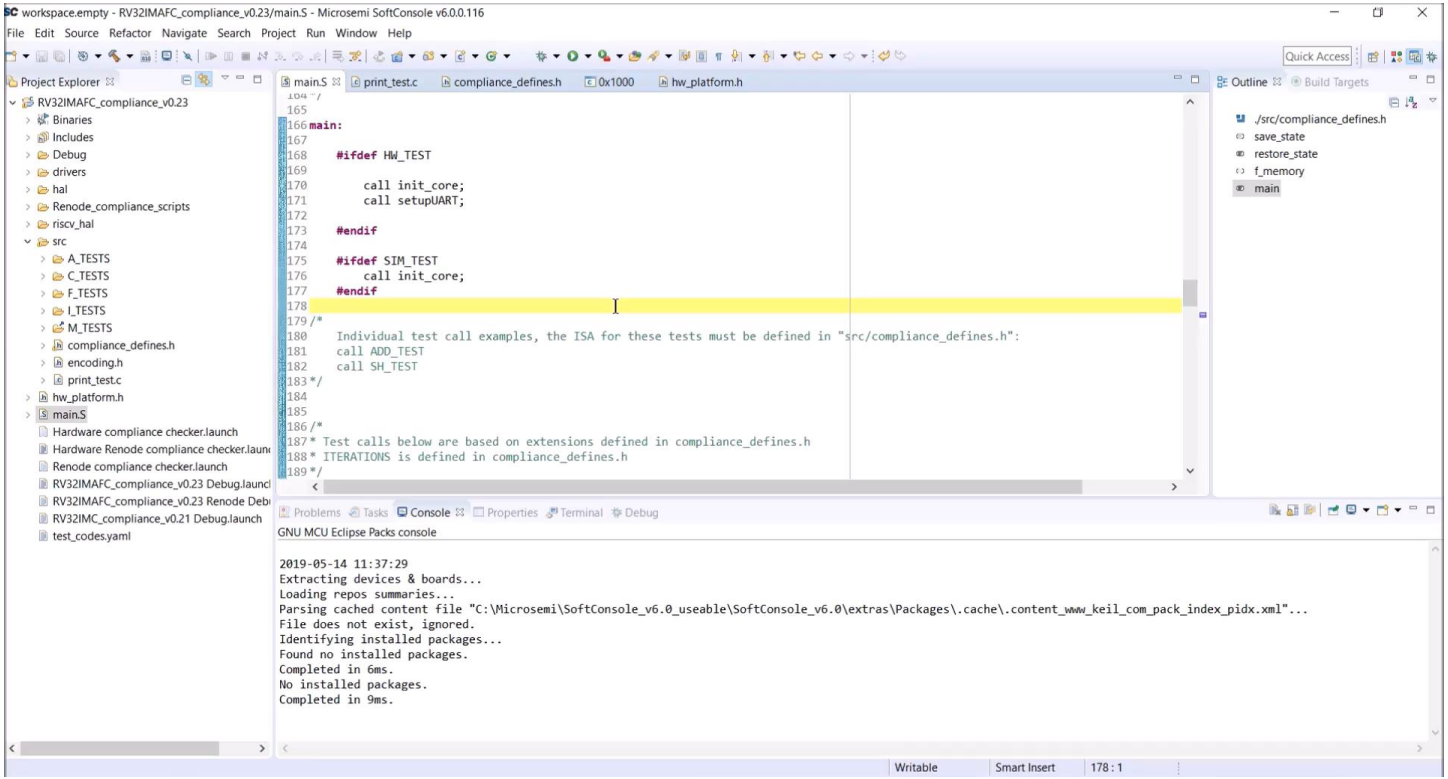
I-ADDI-01_converted.S

```
145 # Test
146
147     addi    x19,    x18,    1
148     addi    x20,    x18,    0x7FF
149     addi    x21,    x18,    0xFFFFFFFF
150     addi    x22,    x18,    0
151     addi    x23,    x18,    0xFFFFF800
152
153     la  x6,     __STORE_LOC
154     lw  x5, (x6)
155     add x6, x6, x5
156
157     sw  x18, 0(x6)
158     sw  x19, 4(x6)
159     sw  x20, 8(x6)
160     sw  x21, 12(x6)
161     sw  x22, 16(x6)
162     sw  x23, 20(x6)
163
164     la x6, __STORE_LOC
165     add x5, x5, 24
166     sw  x5, (x6)
167
```

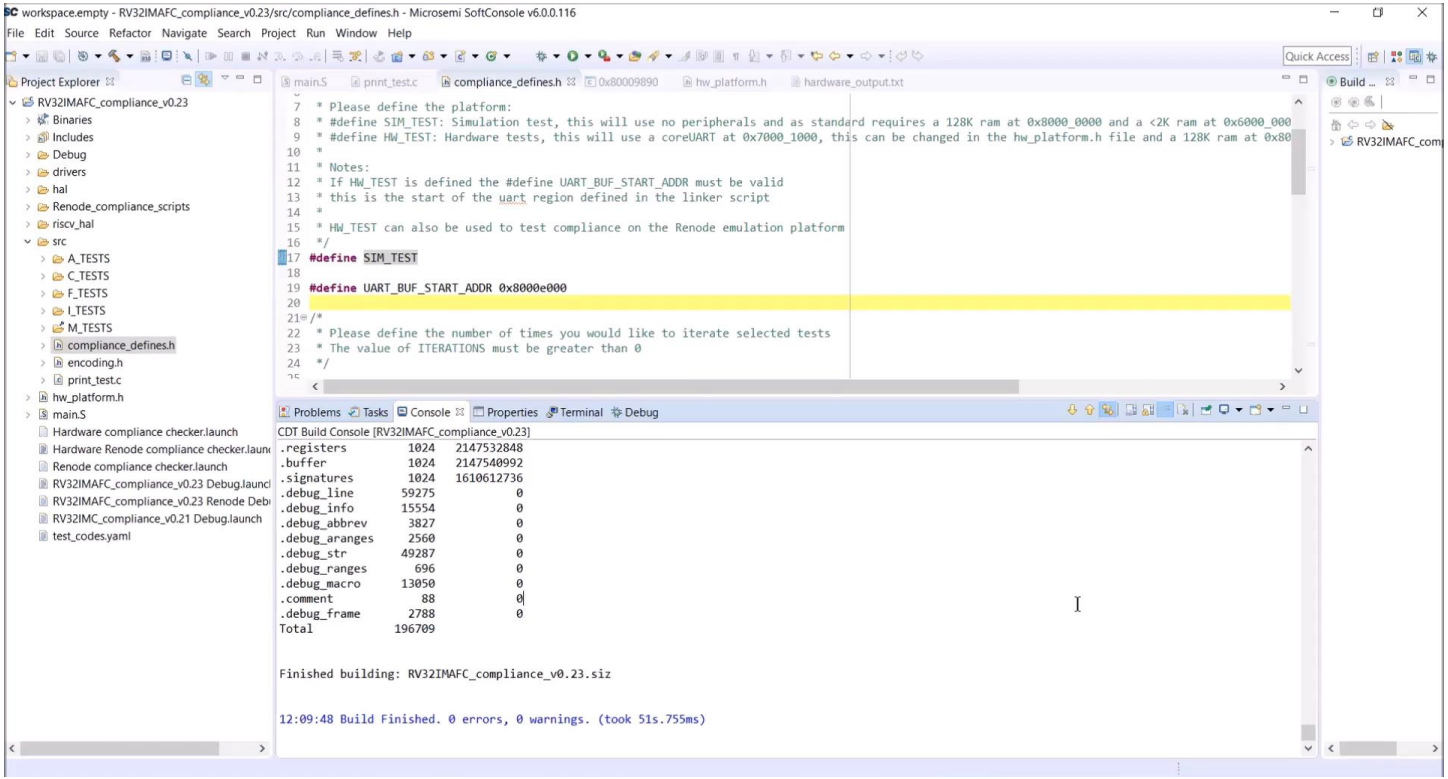
RISC-V Compliance Hardware Platform



RISC-V Compliance Emulation Platform



RISC-V Compliance Simulation Platform



Formal Methods in Mi-V RISC-V Core Development

•Simulation (and emulation) is insufficient for processor verification.

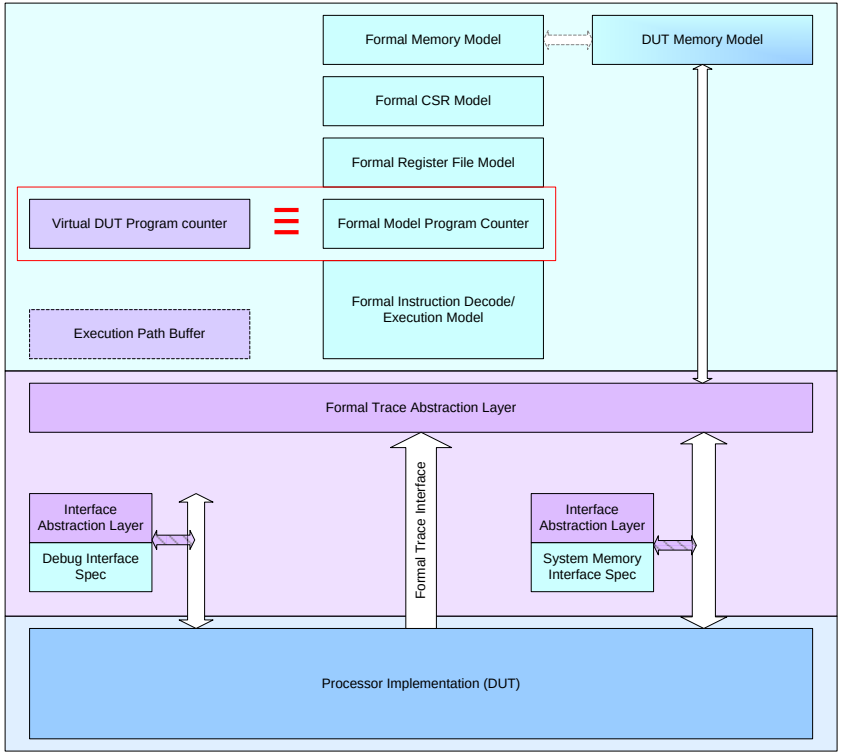
- Processors are complex (even simple ones!).
- Processors have many corner cases, in fact a lot of the logic is to manage such cases.
- Can simulation ever hit all the permutations, both spatially and temporally?
- Do we know what to hit, i.e. is the coverage sufficient?

•Formal methods are good for processor verification.

- Very good for finding corner case issues quickly.
- Processors are well specified and easily constrained (have well-defined interfaces).
  - RISC-V is particularly well specified!
- Allow aggressive design.
- Super-compliance.

Formal Methods in Mi-V RISC-V Core Development

- **Compare RTL implementation to a formal model**
  - Approach used is similar to other approaches checking architectural transformation of state such as ARM® ISA-Formal, RISC-V Formal (Clifford Wolf).
- **Mi-V Formal differs significantly due to the targets and goals**
  - Not assuming designs are our own, therefore requires a portable, black-box approach
  - Cannot rely on most state being visible
- **Mi-V Formal checks equivalence to an architectural model by comparing the program counter of retired instructions**



Known Knowns, Known Unknowns…

- **Some model states are unknown, some are architecturally undefined, some have a range of allowable behavior – how do we check equivalency with an implementation?**
  - Initialize everything in both model and implementation – hard to do and inefficient
  - Use a model that deals with it

- **Created SystemVerilog ISA model that allows known/unknown state (and propagation of it) to be explicitly defined with "known value" logic.**
  - Avoids the need for any "directed" constraints to avoid undesirable behavior that would normally be discounted as "bad software." *Only* constrained by the architecture definition (not over-constrained).
  - "Known value" propagation allows us to aggressively abstract memories, resets and arithmetic operations, etc., by only caring about the concrete values in a small subset of cases.

Known Knowns, Known Unknowns…

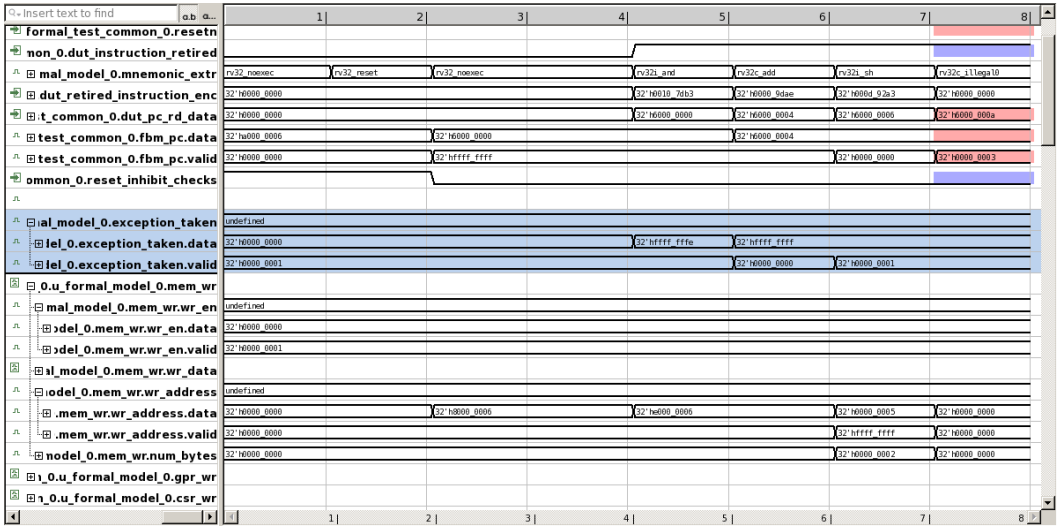
- **“Known value” logic model**
  - Created 4-state logic model.
  - Defined ISA model based on this logic. In SystemVerilog this means a know-value datatype (tuple of value, known/unknown-state) and functions for all logic/arithmetic functions.
  - Example:
    A = {0101, 1111} would be a value of 0101 with all bits known.
    If B = {0111, 0110} then A&B = {0101, 0110), and A+B = {1010, 0000}.

- **Propagation of failures**
  - Example:
    CSR has incorrect access privilege in implementation where implementation incorrectly allows access.

csrrw x0, mtvec, x0	<model pc = {0,F}, impl pc =0>
csrxxx x?, <offending_csr>, x0	<model pc = {4,F}, impl pc =4>
any instruction	<model pc = {**0**,F}, impl pc =**8**>

Formal Methods in Mi-V RISC-V Core Development

- **Mi-V Formal has been focused on design bring-up (bug-hunting) and on privileged architecture compliance on machine-mode cores.**
- **Highly efficient, highly effective!**
- **Bugs found include:**
  - Incorrect CSR accesses, privileges and bit writability
  - Incorrect trap behavior
  - Incorrect exception return behavior
  - Getting stuck in debug mode
  - Pipeline interlock errors
  - Decode errors
  - Fetch and load/store ordering
  - …



Summary

- For development and release of Mi-V RISC-V cores, Microchip has employed a suite of traditional techniques and combined them with formal model checking techniques to enhance quality and ensure compliance.
- Traditional techniques have been run on multiple platforms, utilizing open-source compliance, random-instruction set simulation and benchmarks, with added coverage measurement.
- Formal techniques have been developed to provide better/faster verification of new cores as well as privileged ISA compliance.
- Mi-V core RTL and compliance/verification suite, are available as part of Microchip's Libero software and open-sourced on GitHub at https://github.com/RISCV-on-Microsemi-FPGA.