RVVI
RISC-V Verification Interface
Discussion in OpenHW
01-21-2021

Preamble

The thoughts and implementation work done here, have been developed from the experiences gathered during the implementation of the Step/Compare approaches take with the RISCV developments for both EMMicro, SiLabs and OpenHW (and other 3rd party companies)

This also leans and expands on previous work by Symbiotics in describing their RVFI (RISC-V Formal Interface)

Reference: https://github.com/riscv-verification/RVVI

Problem Statement

When verifying an RTL Micro-Architectural Implementation of the RISC-V Specification, the provision of an Architectural Reference Model is a crucial element in ensuring that implemented behavior is not ‘lost in translation’

For this to be successful it is imperative that when executing instructions, or assigning asynchronous events, both representations are able to be controlled and observed in such a manner that ensures that state comparisons can be definitively trusted

RVVI
RISC-V Verification Interface
(in reality, a tale of two interfaces)

- State Interface
  - Provision of notification, data, internal state
- Control Interface
  - Control of execution
  - Observation of execution progress

RVVI – State Interface

- The state interface reports evaluations which have completed to a result, this could be an instruction which
  - Retires successfully
  - Causes a trap to occur (alignment, ecall, abort)
  - Causes a halt to occur (wfi)
- This could be an event which
  - Causes a mode switch (debug request)
  - Causes an Interrupt to be taken (asynchronous exception)

RVVI – State Interface Data Container

Name	Description
notify	event to inform a listener of new state for interrogation
valid	flag to indicate a successful retirement
trap	flag to indicate an exception
halt	flag to indicate a halt condition
intr	flag to indicate first instruction of a trap a handler (required?)
order	instruction count for a (valid) instruction
insn	instruction content for valid, Trap or Halted instruction
isize	size of instruction (insn) 16/32 bits
mode	operating mode (Machine, User, Supervisor, …)
ixl	current XLEN for mode of operation
decode	string decode of current instruction
pc	program counter value for instruction (valid / trap)
pcnext	program counter value for nest instruction to be executed
x	array of registers containing INTEGER / X
f	array of registers containing FLOAT / F
csr	array of registers containing CSR values

RVVI – State Interface
Data Container SystemVerilog

```
interface RVVI_state #(
    parameter int ILEN = 32,
    parameter int XLEN = 32
);

    event            notify;

    bit              valid; // Retired instruction
    bit              trap;  // Trapped instruction
    bit              halt;  // Halted  instruction

    bit              intr;  // Flag first instruction of trap handler
    bit [(XLEN-1):0] order;
    bit [(ILEN-1):0] insn;
    bit [2:0]        isize;
    bit [1:0]        mode;
    bit [1:0]        ixl;

    string           decode;

    bit [(XLEN-1):0] pc;
    bit [(XLEN-1):0] pcnext;

    // Registers
    bit [(XLEN-1):0] x[32];
    bit [(XLEN-1):0] f[32];
    bit [(XLEN-1):0] csr[string];

endinterface
```

RVVI – Control Interface

- The control interface contains both data and methods
- The methods control the execution - in fact this closely resembles a GDB/RSP server interface providing the client with a 'Run/Stop Control' capability.
- In the case of this environment, the client requesting Run/Control is the testbench
- In the simplest case we consider, we want the testbench to control on-demand instruction execution, which the state interface notifier indicates a result of either
  - valid, trap or halt
- Additionally the associated state variables are assigned values dependant upon the result above

RVVI – Control Interface Data / Methods Container

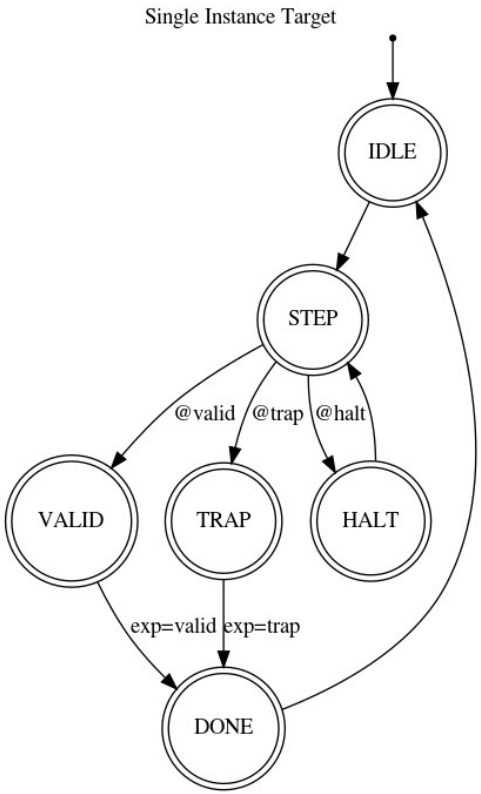
Variable	Values
cmd	IDLE, STEPI, STOP, CONT

Method	Visibility	Description
idle()	private	function to set the cmd variable to indicate a completed command cmd = IDLE
stepi()	public	function to request the target attempt to execute the instruction indicated by the pcnext on the state interface cmd = STEPI
stop()	public	function to request the target stop executing at the next opportunity (state notifies of valid or trap condition) cmd = STOP
cont()	public	function to request the target to continue until some pre-defined condition is satisfied, or an unexpected exception cmd = CONT

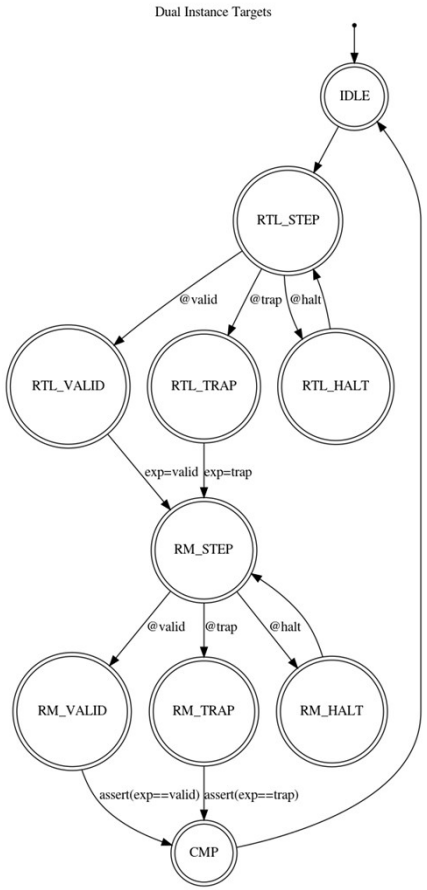
RVVI – Control Interface Data / Methods Container SystemVerilog

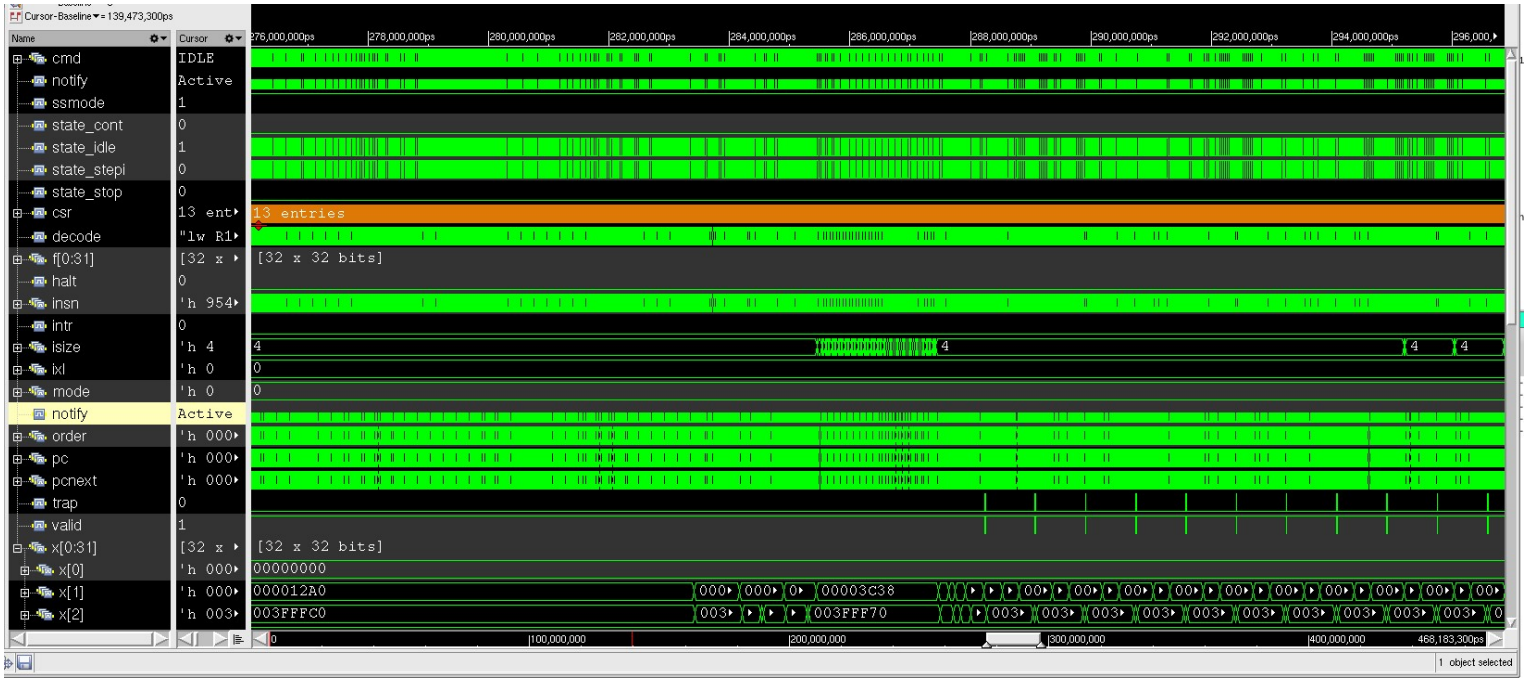
```
typedef enum { IDLE, STEPI, STOP, CONT } rvvi_c_e;
interface RVVI_control;

    event     notify;

    rvvi_c_e  cmd;
    bit       ssmode;

    bit       state_idle;
    bit       state_stepi;
    bit       state_stop;
    bit       state_cont;

    initial ssmode = 1;

    assign state_idle  = (cmd == IDLE);
    assign state_stepi = (cmd == STEPI);
    assign state_stop  = (cmd == STOP);
    assign state_cont  = (cmd == CONT);

    function automatic void idle();
        cmd = IDLE;
        ->notify;
    endfunction
    function automatic void stepi();
        cmd = STEPI;
        ->notify;
    endfunction
    function automatic void stop();
        ssmode = 1;
        cmd = STOP;
        ->notify;
    endfunction
    function automatic void cont();
        ssmode = 0;
        cmd = CONT;
        ->notify;
    endfunction

endinterface
```

RVVI – Testbench State Flow



RVVI – Testbench State Flow





RVVI – Further Thoughts

RVVI – Further Thoughts
Notification of Interrupt Taken

- We need a method of telling the RM the instruction boundary in which to perform processing of an active interrupt source
- An interrupt is only taken when active, enabled and then processed at a specific time by the RTL microarchitectural state machine. The instruction at which the interrupt exception is taken, is the boundary in which we are interested
- In the RM interrupts are considered when presented, at each instruction execution, we need to control the scheduling as seen by the microarchitecture
  - **In fact this is a solved issue using 'deferint', but I think we need to formalize this as part of the RVVI**

RVVI – Further Thoughts memory aborts

The majority of memory exception behavior, is determined by the core itself, there are a few exceptions which are generated by the memory subsystem, these include (but may not be limited to)

- Instruction Access Fault (cause=1)
- Load Access Fault (cause=5)
- Store/AMO Access Fault (cause=7)

One of our DV customers is making use of the ability to pass this information back to the RM so that it can take an exception, and set the mcause code accordingly

RVVI – Further Thoughts
Custom Exceptions

Additionally we may need to consider the communication of custom exceptions

- cause 24 - 31
- cause 48 - 63

RVVI –

So who volunteers to create the CV32E40P with the RVVI ☺

How do we assist the development of the next core to adopt RVVI

Discuss …

Other interfaces considered

GDB / RSP (Remote Serial Protocol)

Arm CADI (Component Architecture Debug Interface)

SystemC CCI (Configuration Control and Inspection)