2023
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

Random Testcase Generation and Verification of Debug Unit for a RISC-V Processor Core

Sneha Mishra, Lu Hao, Ajay Sharma, Afshan Anjum,
Lucia Franco, Sourav Roy, Jeff Scott

NXP

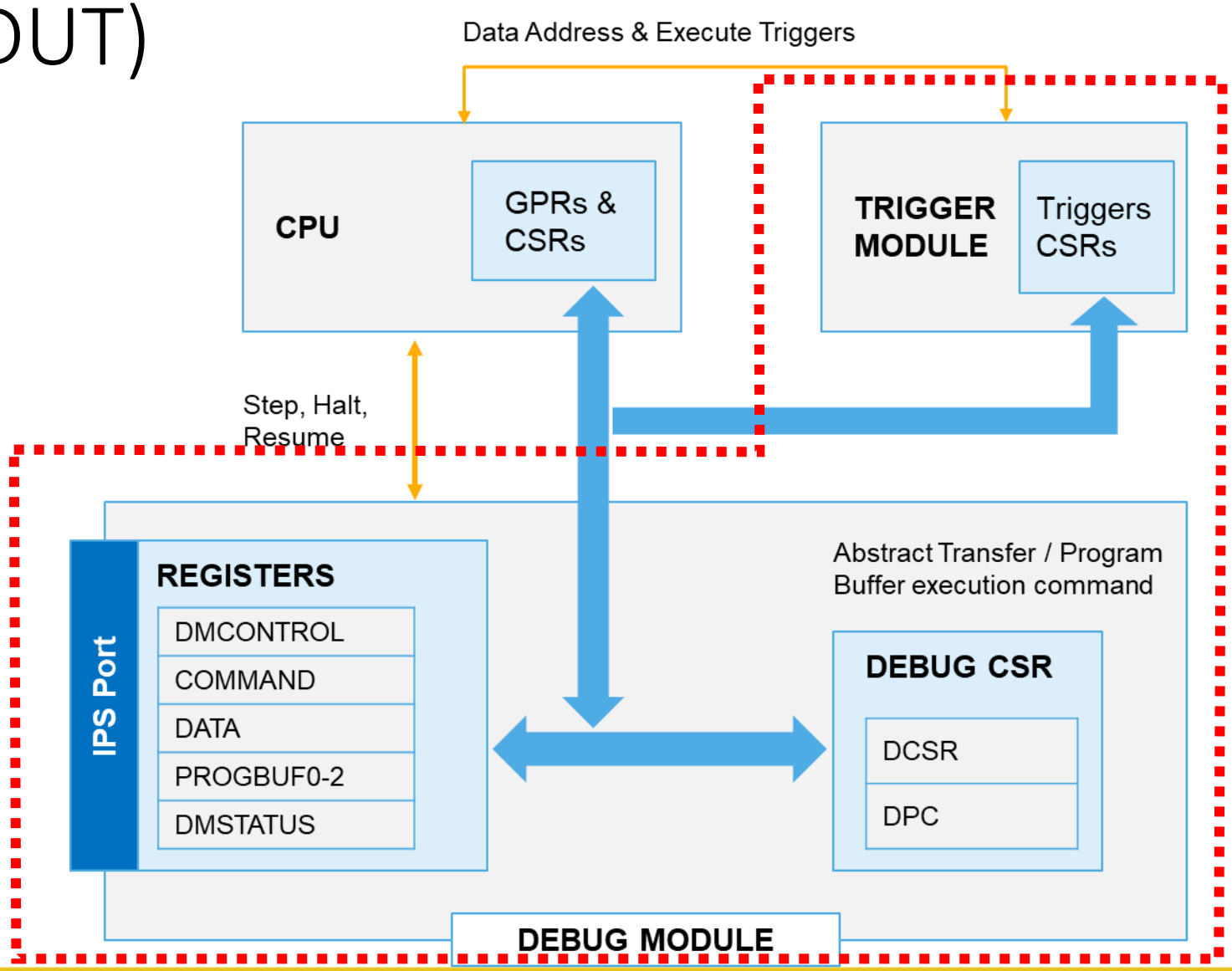
accellera
SYSTEMS INITIATIVE

Table of contents

- Design-Under-Test (DUT)
- Verification Challenges & Solution
  - RAPTOR - Random test case generator
- Verification methodology
  - Debug verification environment overview
  - Verification Targets
    - Run control and External Triggers
    - Data and Instruction Triggers
    - Abstract command and Program Buffer execution
  - Debug Checkers
- Results and Coverage

Design-Under-Test (DUT)

- CPU can be debugged from the very first instruction
- **RUN Control**:
  - Halt, resume
  - Single-step
- Halt groups, Resume groups and External Triggers
- Instruction and data triggers (breakpoints and watchpoints)
- Abstract commands:
  - Register transfer
  - Program buffer execution



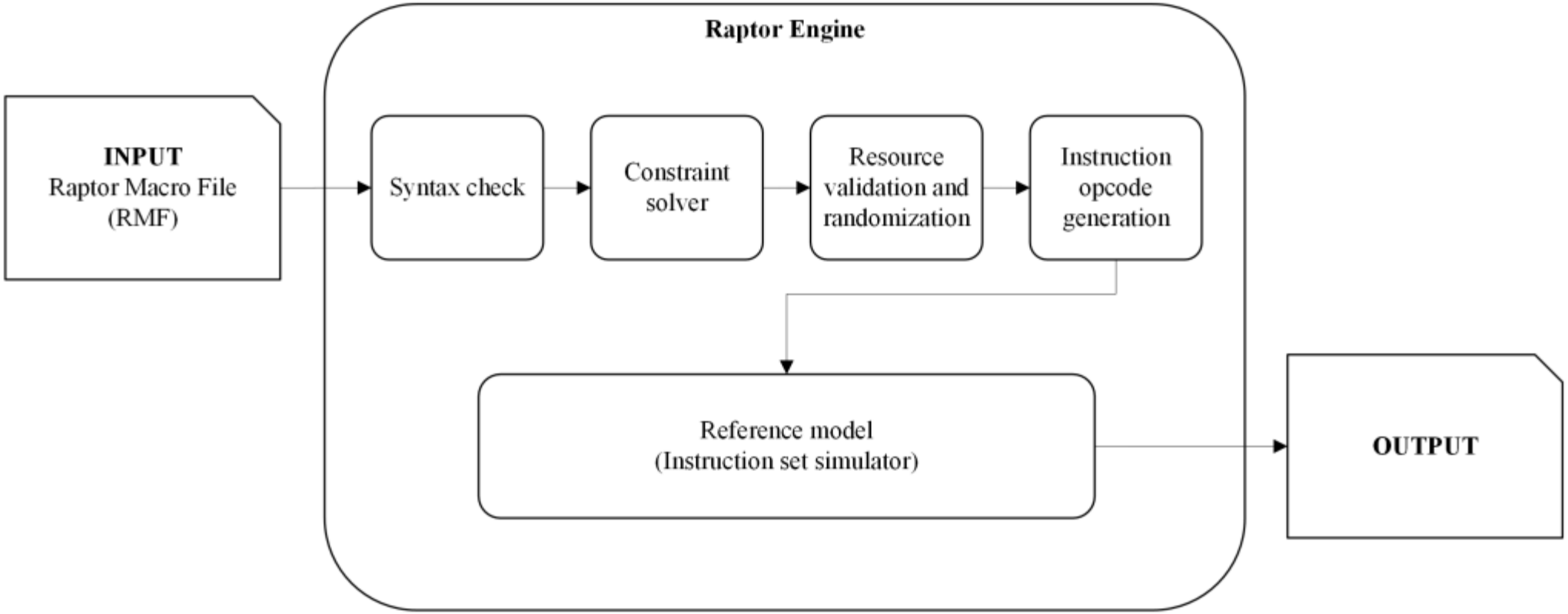
Verification challenges

- Putting the core in different configurations with varying mix of instructions in core's pipeline, along with external stimuli such as interrupts, debug exceptions, bus faults, and then bombarding with debug requests randomly during simulation
- Exercising random instruction stream within debug mode session with varying debug control configurations
- Correctness checking. Involves synchronizing of the asynchronous events with the functional model, generating correct set of expectations in terms of registers and memory updates

RAPTOR - Random test case generator

- Provides a specific language that offers well-known control flow structures (i.e., if, while, foreach, etc.) and supports querying generator and processor state, thus allowing users to dynamically bias or change the test flow during runtime based on current conditions.
- Instruction randomization
  - Generate instructions randomly but under user constraints
- Events randomization
  - If a register is modified during the debug session, the state of the processor is modified. The simulation would then deviate from the intended test flow and would produce a failure.
  - We need to handle this case when incorporating non-recoverable debug events into the test creation flow in Raptor

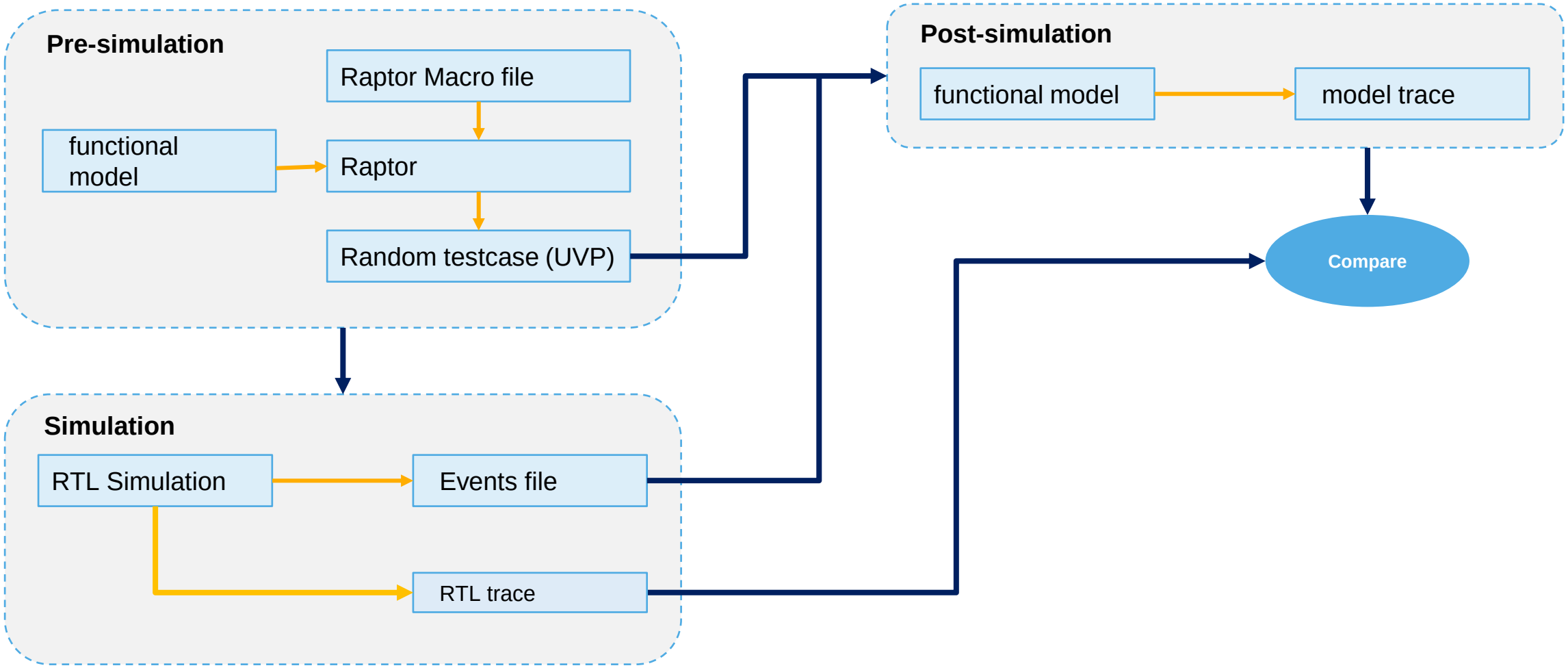
RAPTOR - Random test case generator



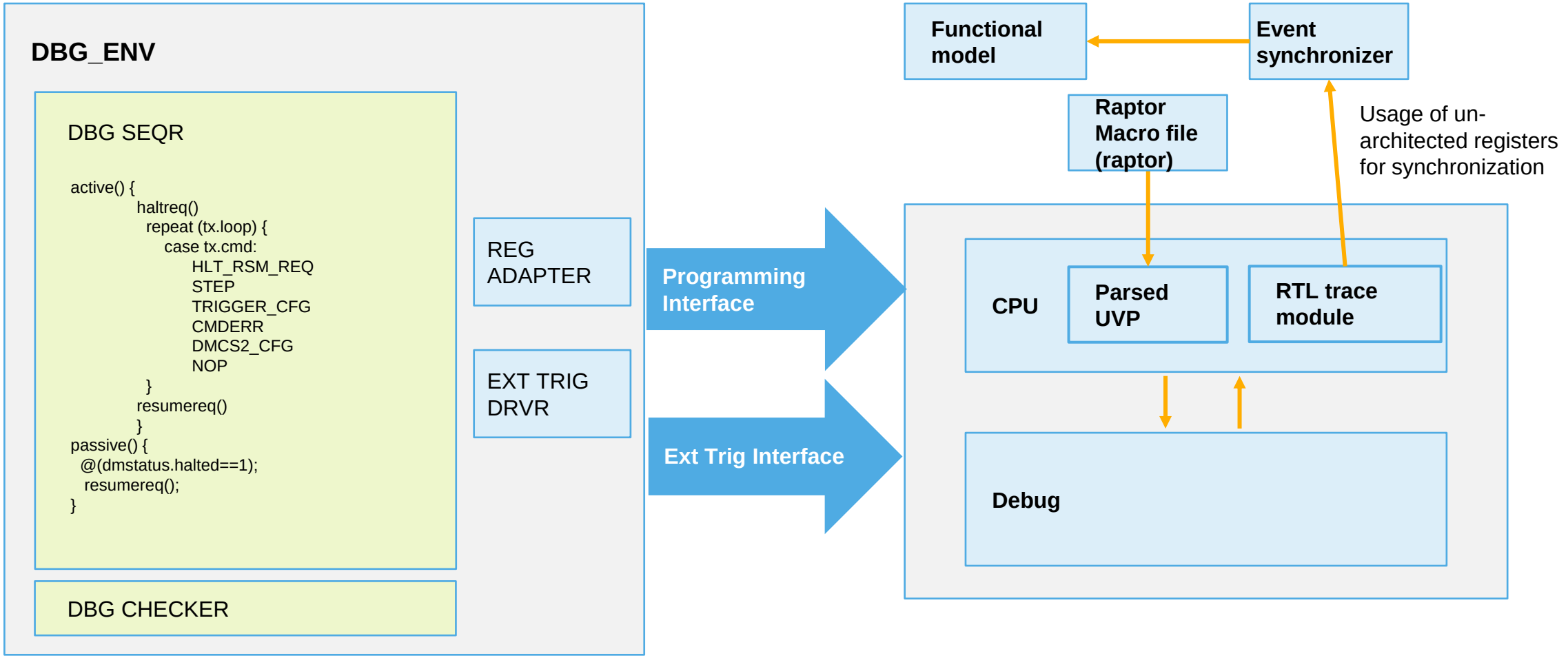
Verification Methodology

A. Debug Verification Environment
B. Verification Target – Run Control and External Triggers
C. Verification Target – Data and Instruction Triggers
D. Verification Target – Abstract Command and Program Buffer Execution
E. Debug Checkers

Debug Verification Environment



Verification Target – Run Control and External Triggers



Verification Target – Data and Instruction Triggers

- Instruction trigger

```
tdata2 = PC + random_range( <start_addr> , <end_addr> );
tdata1 = random_range( 0 , 0xFFFFF );
repeat(count) {
  call any_inst(); // select instruction randomly from instruction list
}
```

- Data trigger

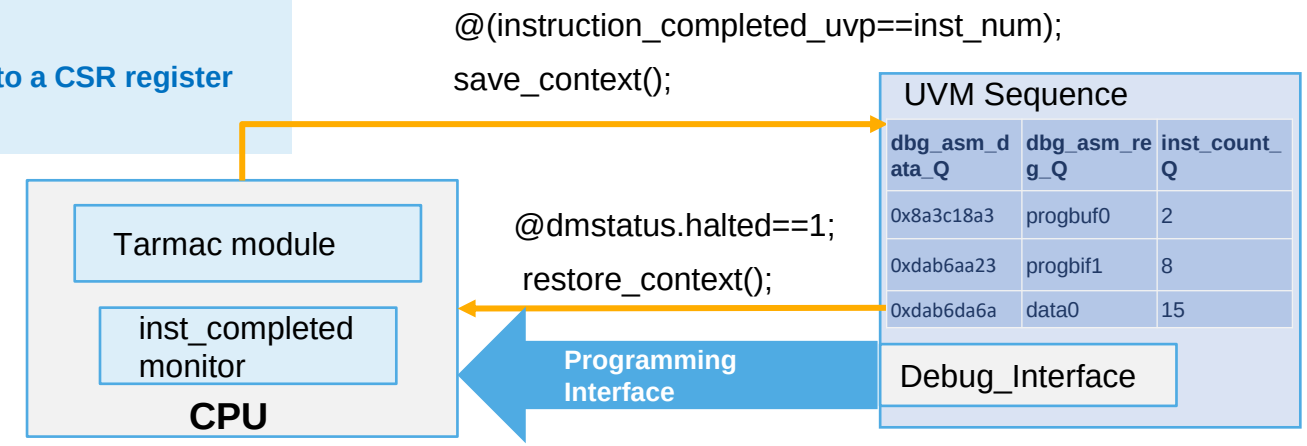
```
tdata2 = random_range( <start_addr> , <end_addr> );
tdata1 = random_range( 0 , 0xFFFFF );
gpr10 = tdata2;
Repeat (count) {
  call ld_st(addr=gpr10); // randomly generate a load/store insruction; GPR10 contains the target address
  call any_inst(); // select instruction randomly from instruction list
}
```

- The above macros are called in different raptor menus in conjunction with async inputs like external interrupts, halt requests, load store faults etc.

Verification Target – Abstract Command and Program Buffer Execution

Example UVP file

```
#1   "MRET“                          // Instruction #1
#2   "SH 14,29,0xe08"                // Instruction #2
#E1 progbuf0=>0x8a3c18a3   // Event #1: write to progbuf0
#E2 command=>0x240000      // Event #2: abstract command - execute program buffer
..
..
#8     "CSRRS 0,28,0x342"            // Instruction #8
#E3 dmcontrol=>0x80000001 // Event #3: write to dmcontrol
#E4 progbuf1=>0xdab6aa23   // Event #4: write to progbuf1
#E5 command=>0x240000      // Event #5: abstract command - execute program buffer
..
..
#15   "SW 10,27,0xdf5“              // Instruction #15
#E6 dmcontrol=>0x80000001  // Event #6: write to dmcontrol
#E7 data0=>0xdab6da6a          // Event #7: write to data0
#E8 command=>0x2307a2      // Event #8: abstract command – write to a CSR register
```

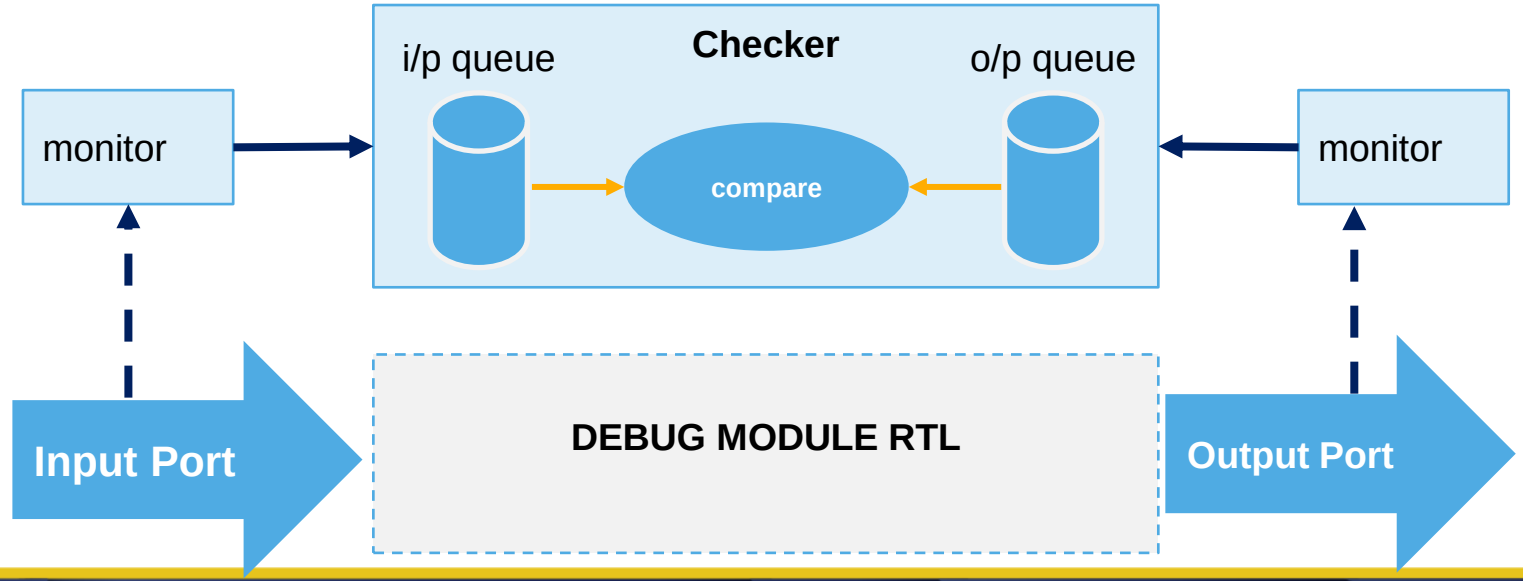


Verification Target – Abstract Command and Program Buffer Execution

- Raptor provides users several ways to place instructions in the program buffer. For example:
  - a) progBuff(”progbuf0”, 0x123452b7); // direct opcode
  - b) progBuff(”progbuf1”, ‘call # LH(realAddr=0x9000)); // biased-random instruction: load from address 0x9000
  - c) progBuff(”progbuf2”, instr()); // function call contains biased instructions
  - d) progBuffs(instrs()); // function call containing multiple biased instructions

Debug Checkers

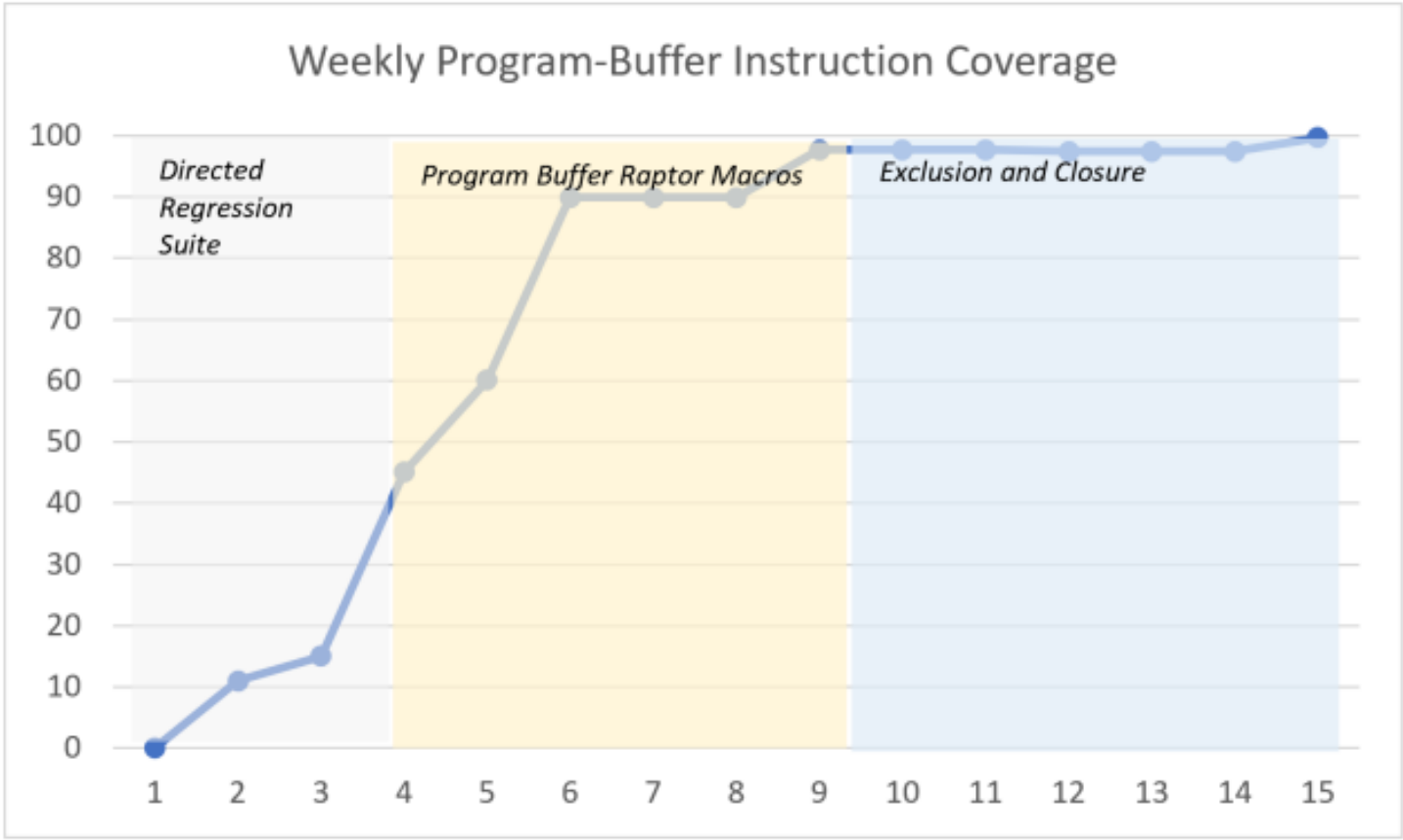
- Exhaustive checking of signal-level protocol
  - CPU and debug interactions
  - Debug state transitions (halt request, resume ack, etc)
  - External notification on the output port for non-debug trigger actions
  - Number of steps during single stepping



Results and Coverage

- Achieved high coverage of the debug module
  - Achieved 100% functional coverage of the debug module. E.g. coverage of triggers and instructions executed from the program buffer.
  - Achieved 100% code coverage utilizing random verification and Unreachability (UNR) analysis with formal tools.
- Multiple corner case bugs and interesting architectural issues around custom instructions
- Several interesting scenarios were unearthed that were not thought of during conventional directed verification of the debug module
  - Program buffer execution of various categories of instructions in different privilege modes
  - Debug features crossed with other external stimuli like interrupts, bus faults etc.
  - Debug features crossed with decode class exceptions (ecall, mis-aligned faults etc.)
  - Debug features crossed with low power mode stimuli
  - External debug and internal debug mode cross scenarios

Results and Coverage



Questions

accellera
SYSTEMS INITIATIVE