A RISC-V SystemC-TLM simulator

CARRV 2020

Màrius Montón

Outline

- Introduction
- Simulator
- Tests
- Conclusions

Motivation

Motivation

Develop a simple simulator based on a RISC-V CPU

- As a embedded processor
  - Small CPU
  - Simple memory scheme
- Using a simple toolchain
  - Out-of-the-box binary from gcc
  - Easy tools
  - No semi-hosting facilities
- And easy expandable
  - Attach new peripherals
  - Add new RISC-V extensions
  - Modify CPU architecture

Simulator

Simulator

SystemC as language

- C++ based, well known language
- Add-ons HW to C++
- Simulation based
- Possibility to synthesis with external tools

TLM-2 as modeling

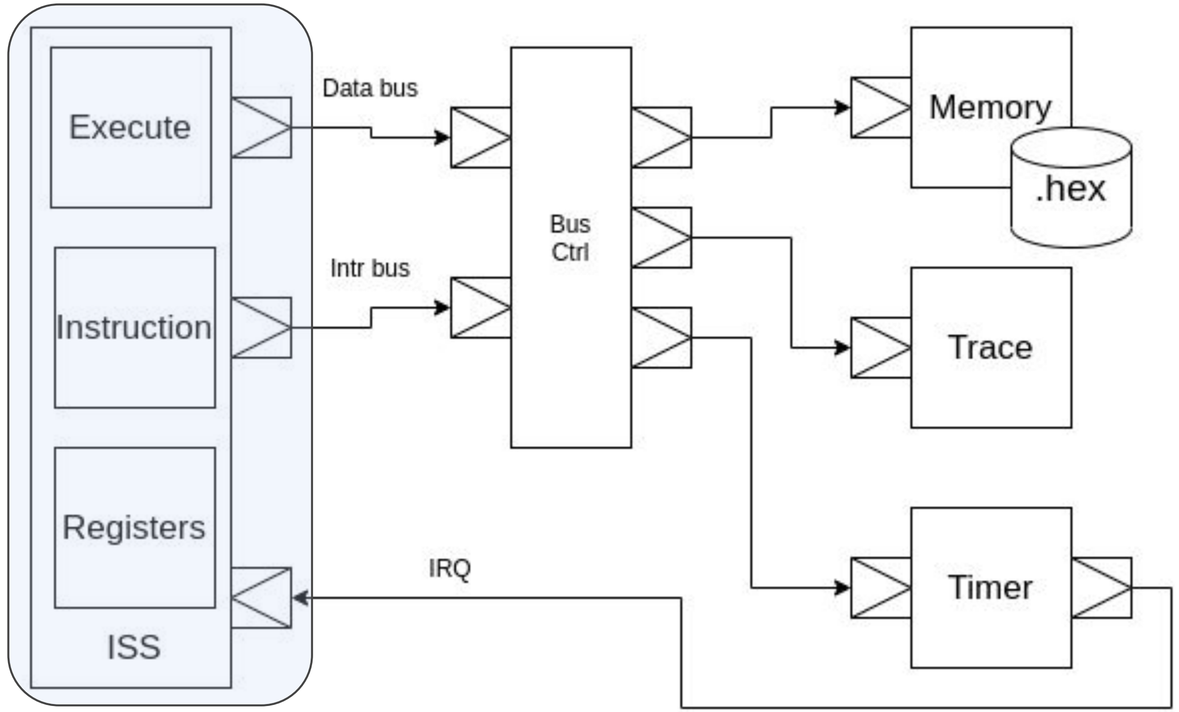
- Transaction based
- Common interface

Simulator

TLM Transactions & sockets

- Communication channel
- Abstraction of a bus
  - Details not important
  - Information about time and address/data
- Increase simulation speed
- Sockets encapsulates all this
  - Initiator/Target <--> Master/Slave
  - Interchangeable

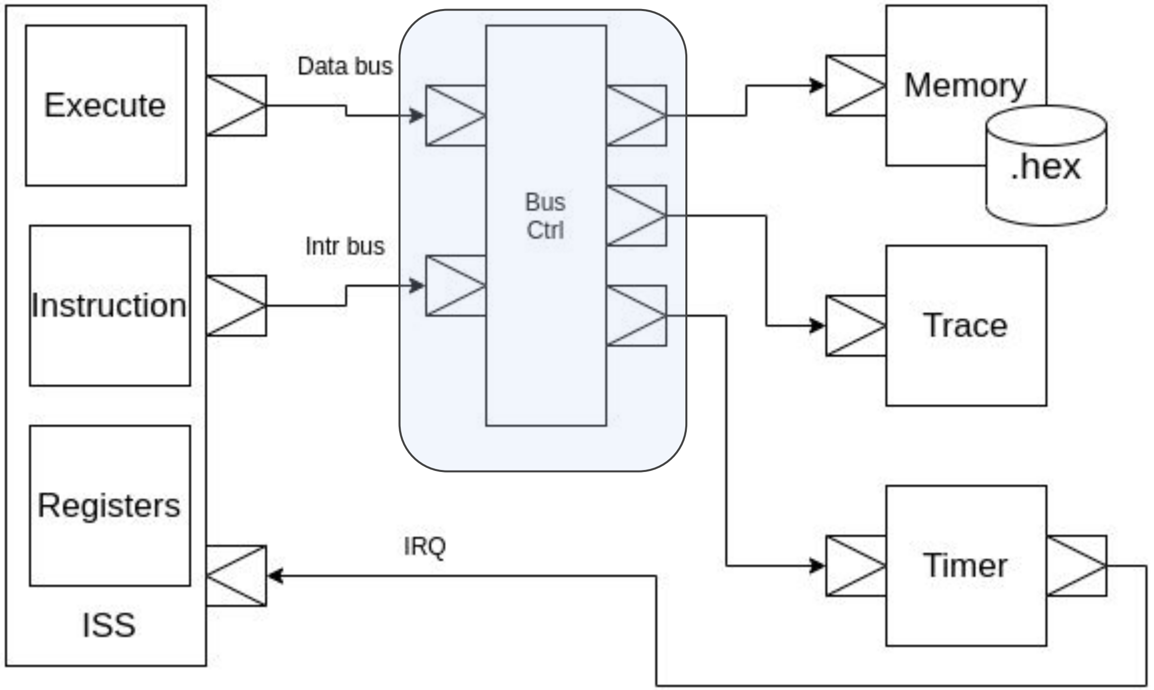
Simulator



Instruction set simulator

- Execute and decode
  - Extensions
- Register file
  - x0-x31
  - PC
  - CSR
- Harvard
  - Data / Instr. Bus
- IRQ port

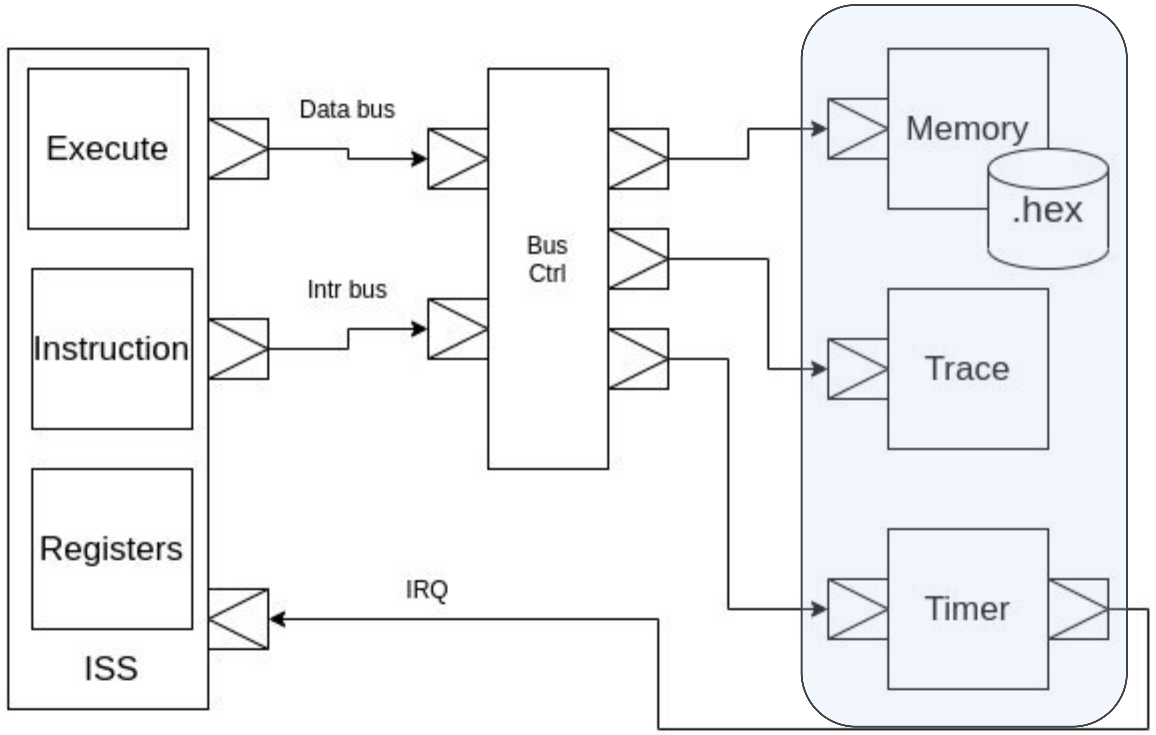
Simulator



Bus controller

- Data / Instr. Input sockets
- Out sockets
    - To memory
    - Peripherals
        - Trace
        - Timer
        - …
- Memory map

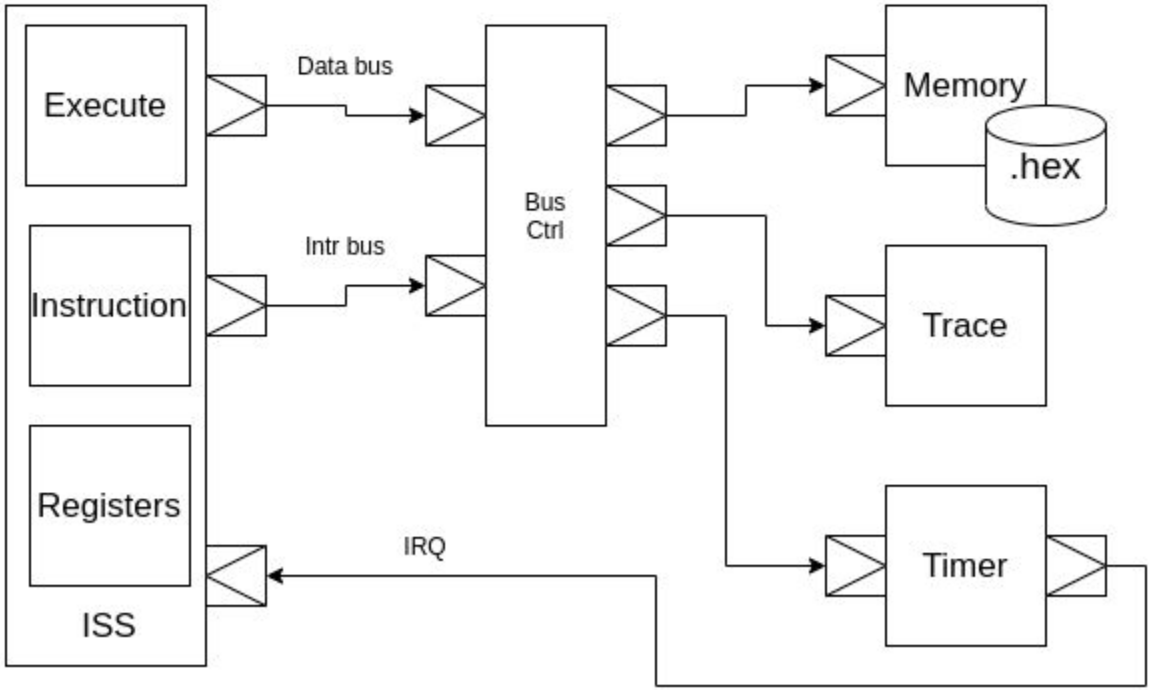
Simulator



Peripherals

- Memory
  - Exe file pre-loaded
- Trace for debug/console
- Timer, trigger IRQ

Simulator



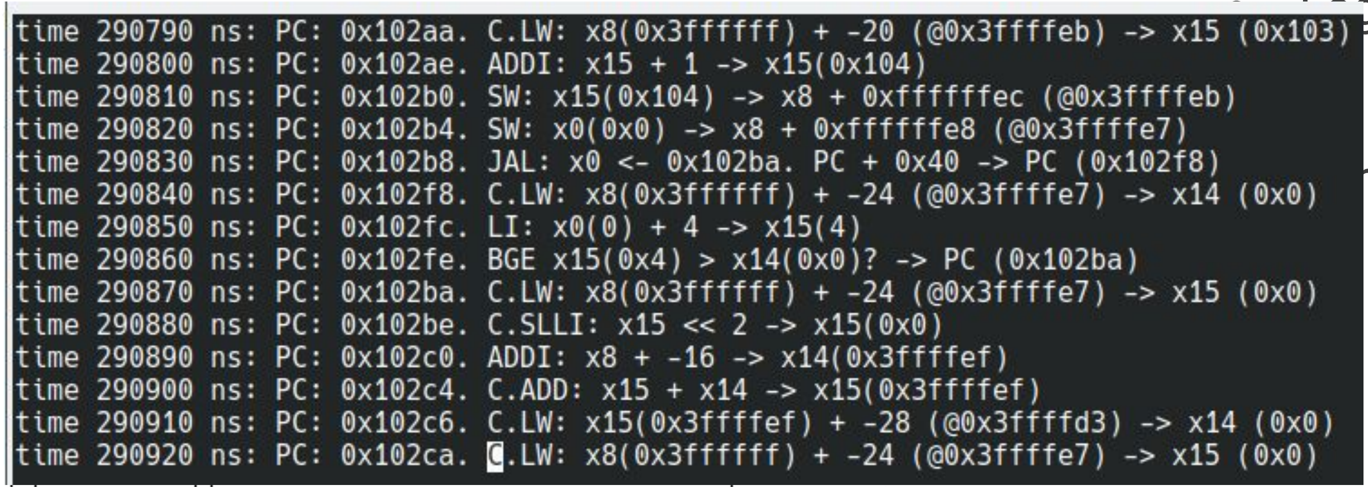
Simulation helper

- Log execution
  - Log file
  - Operands and result
- Performance metrics
  - Memory accesses
  - Registers accesses
  - Instructions executed

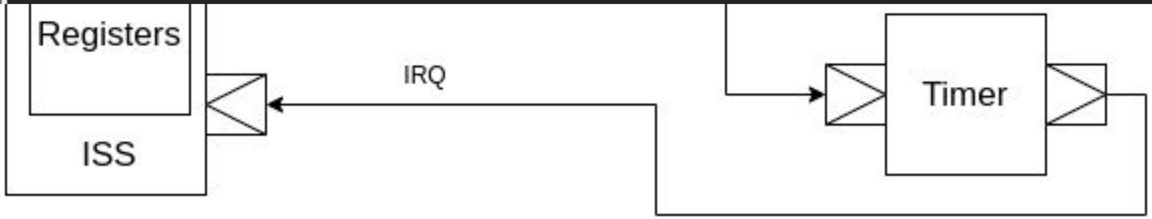
Simulator

Simulation helper

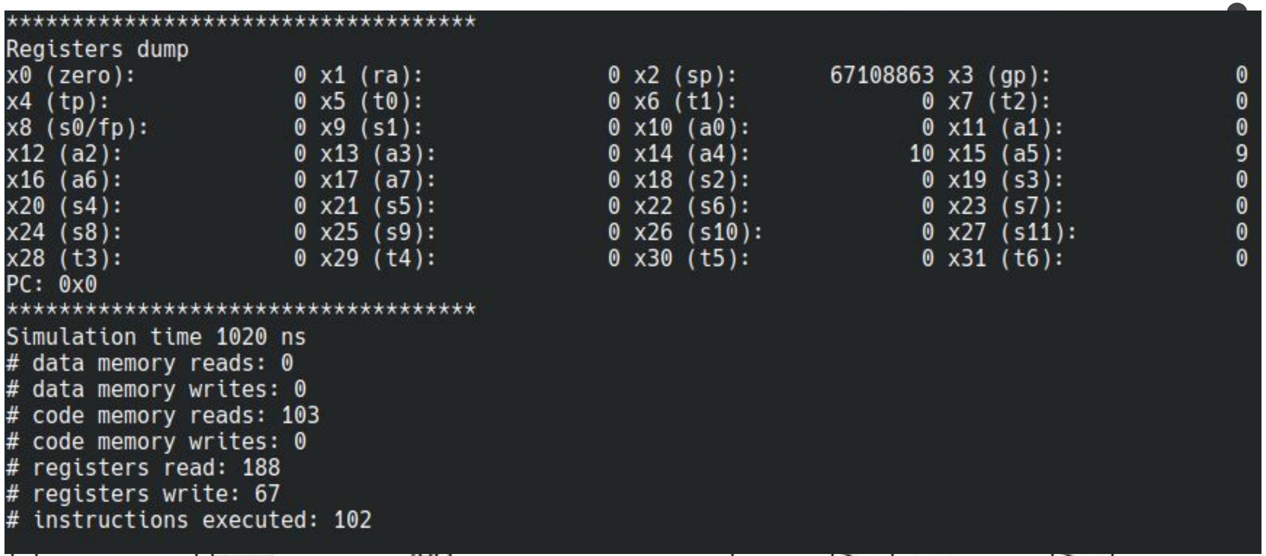
- Log execution
  - Log file
  - Operands and result
- rformance metrics
  - Memory accesses
  - Registers accesses
  - Instructions executed



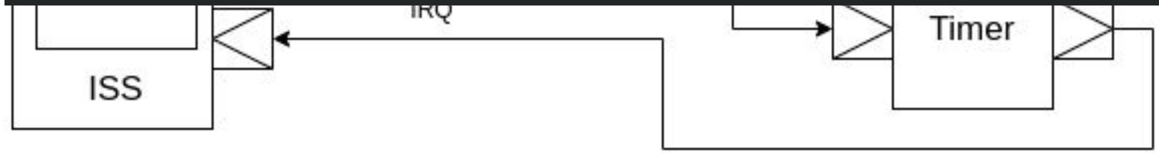
```
time 290790 ns: PC: 0x102aa. C.LW: x8(0x3fffff) + -20 (@0x3fffeb) -> x15 (0x103)
time 290800 ns: PC: 0x102ae. ADDI: x15 + 1 -> x15(0x104)
time 290810 ns: PC: 0x102b0. SW: x15(0x104) -> x8 + 0xffffffec (@0x3fffeb)
time 290820 ns: PC: 0x102b4. SW: x0(0x0) -> x8 + 0xffffffe8 (@0x3fffe7)
time 290830 ns: PC: 0x102b8. JAL: x0 <- 0x102ba. PC + 0x40 -> PC (0x102f8)
time 290840 ns: PC: 0x102f8. C.LW: x8(0x3fffff) + -24 (@0x3fffe7) -> x14 (0x0)
time 290850 ns: PC: 0x102fc. LI: x0(0) + 4 -> x15(4)
time 290860 ns: PC: 0x102fe. BGE x15(0x4) > x14(0x0)? -> PC (0x102ba)
time 290870 ns: PC: 0x102ba. C.LW: x8(0x3fffff) + -24 (@0x3fffe7) -> x15 (0x0)
time 290880 ns: PC: 0x102be. C.SLLI: x15 << 2 -> x15(0x0)
time 290890 ns: PC: 0x102c0. ADDI: x8 + -16 -> x14(0x3fffef)
time 290900 ns: PC: 0x102c4. C.ADD: x15 + x14 -> x15(0x3fffef)
time 290910 ns: PC: 0x102c6. C.LW: x15(0x3fffef) + -28 (@0x3fffd3) -> x14 (0x0)
time 290920 ns: PC: 0x102ca. C.LW: x8(0x3fffff) + -24 (@0x3fffe7) -> x15 (0x0)
```



Simulator



```
************************************
Registers dump
x0 (zero):              0 x1 (ra):                0 x2 (sp):       67108863 x3 (gp):              0
x4 (tp):                0 x5 (t0):                0 x6 (t1):              0 x7 (t2):              0
x8 (s0/fp):             0 x9 (s1):                0 x10 (a0):             0 x11 (a1):             0
x12 (a2):               0 x13 (a3):               0 x14 (a4):            10 x15 (a5):             9
x16 (a6):               0 x17 (a7):               0 x18 (s2):             0 x19 (s3):             0
x20 (s4):               0 x21 (s5):               0 x22 (s6):             0 x23 (s7):             0
x24 (s8):               0 x25 (s9):               0 x26 (s10):            0 x27 (s11):            0
x28 (t3):               0 x29 (t4):               0 x30 (t5):             0 x31 (t6):             0
PC: 0x0
************************************
Simulation time 1020 ns
# data memory reads: 0
# data memory writes: 0
# code memory reads: 103
# code memory writes: 0
# registers read: 188
# registers write: 67
# instructions executed: 102
```



Simulation helper

- Log execution
  - Log file
  - Operands and result
- Performance metrics
  - Memory accesses
  - Registers accesses
  - Instructions executed

Simulator

Pure bare-metal simulator

- ECALL, EBREAK → implemented to help debugger, not calling OS
  - ECALL Stops simulation
  - EBREAK Raise Breakpoint exception
- Need to implement _write() _read() functions in sim code
- Support full C std libraries for sim code
- FreeRTOS porting

Docker version

- Not need to compile anything, just hit & run
- Performance penalty

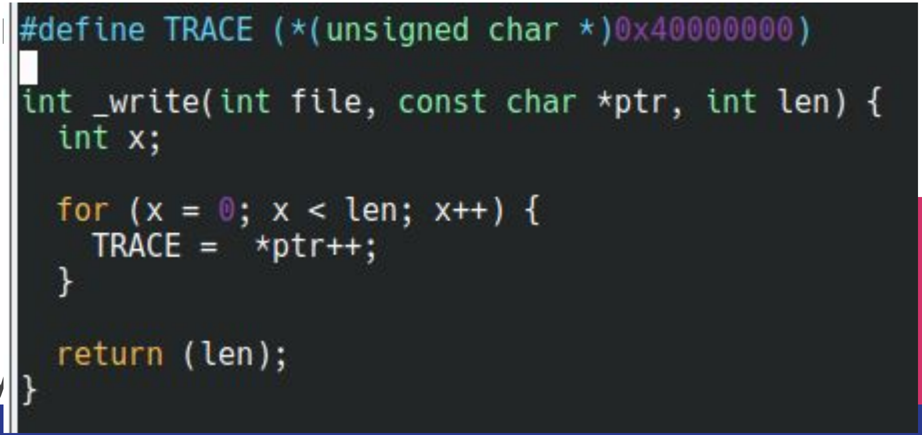
Simulator

Pure bare-metal simulator

- ECALL, EBREAK → implemented to help debugger, not calling OS
  - ECALL Stops simulation
  - EBREAK Raise Breakpoint exception
- Need to implement _write() _read() functions in sim code
- Support full C std lib
- FreeRTOS porting

Docker version

- Not need to compile
- Performance penalty



```c
#define TRACE (*(unsigned char *)0x40000000)

int _write(int file, const char *ptr, int len) {
  int x;

  for (x = 0; x < len; x++) {
    TRACE =  *ptr++;
  }

  return (len);
}
```

Simulator

Pure bare-metal simulator

- ECALL, EBREAK → implemented to help debugger, not calling OS
    - ECALL Stops simulation
    - EBREAK Raise Breakpoint exception
- Need to implement _write() _read() functions in sim code
- Support full C std libraries for sim code
- FreeRTOS porting

Docker version

- Not need to compile anything, just hit & run
- Performance penalty

Simulator

Tool-chain

- Used gcc for RISC-V
- Only small CFLAGS required

```
CFLAGS = -Wall -I. -O0 -static -march=rv32imac -mabi=ilp32
--specs=nosys.specs
```

- Default linker script
- Uses HEX file from elf output

```
> objcopy -Oihex file.elf file.hex
```

Tests

Tests

Check ISS correctness - Compliance tests

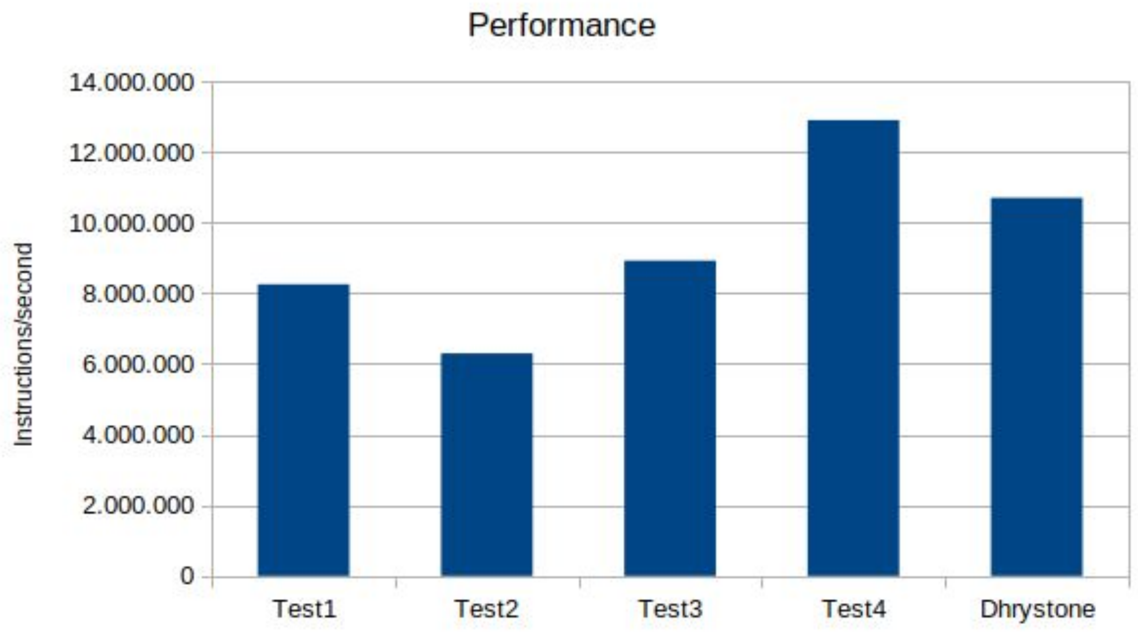
- riscv/riscv-tests passed

- riscv-compliance test passed

Check whole simulator - C programs

- Simple C programs, using libraries
- FreeRTOS porting
- Dhrystone

Tests



- Consistent
- Penalty using trace
- Penalty using Log

Conclusions

Conclusions

Simulator is working fine

- Complex programs running OK
- No cross-tools modifications
- Easy to use and understand

Need to add more components

- Add I/O peripherals
- Add FLASH memory for instr.
- Model a real MCU

Conclusions

Increase performance, but similar to other SystemC simulators

Open-Source

https://github.com/mariusmm/RISC-V-TLM