Exploring RISC-V based platforms within VPSim simulation tool for High Performance Computing

Ayoub Mouhagir, Mohamed Benazouz and Lilia Zaourar

Paris-Saclay University, CEA, LIST, F-91120 Palaiseau, France (firstname.lastname@cea.fr)

Context

The RISC-V Instruction Set Architecture has gained much attention recently due to its open-source nature, flexibility, and greater customization. RISC-V-based platforms are becoming increasingly popular in the embedded systems industry. However, designing and implementing such systems could take time and effort. In order to address these challenges, virtual prototyping tools are widely adopted so that designers can model, simulate, test, and optimize their complex systems in early design phases. VPSim is a tool developed to speed up the SW/HW co-validation of various computer architectures, including RISC-V-based systems.

Virtual Prototyping Simulator

With systems' increasing complexity and scale, virtual prototyping solutions should meet new and evolving requirements regarding performance, ease, rapidity, and automation of system modeling and design space exploration. In this context, VPSim addresses the challenges above and strives to offer maximum flexibility to its users to compose their systems. Figure 1 gives a global overview of VPSim's environment. It includes a large library of models comprising ARM and RISC-V CPUs and models of several peripherals and buses. VPSim is distinguished by its ability to host third-party subsystems using standard and non-standard interfaces such as , Python, and HW designs [1]. It can also interface with other modeling tools, and simulators within a Functional Mockup Interface (FMI) based co-simulation [2, 3].
VPSim's CPU models can be imported from providers such as QEMU, ARM Fast Models, Open Virtual Platforms, etc. The main provider is the open-source system emulator QEMU, which allows unmatched simulation speed. Such high performance is achieved by abstracting micro-architectural (pipeline, out-of-order) and architectural (memory hierarchy) aspects that do not impact the functional behavior. VPSim leverages the QEMU dynamic binary translator and the abstraction levels offered by /TLM 2.0 to provide the best possible trade-offs between accuracy/performance. It integrates QEMU by running its CPU and peripheral models within SystemC threads. These QEMU models are enriched with performance information through VPSim large library modeling memory hierarchy components (caches, NoCs) [4]. VPSim provides a flexible and highly configurable framework for architectural exploration and SW design of complex systems among all the RISC-V-based platforms. For example, VPSim can simulate a full SMP RISC-V platform or instrument a dedicated RISC-V-based accelerator into a larger system.

Platform composition/simulation

VPSim provides a user-friendly interface based on Python to compose and build the simulated platforms. It saves considerable development effort, as many low-level aspects of building a platform are abstracted. A single VPSim executable can simulate an infinite number of architectures without recompilation. It also saves considerable time when testing different configurations. The workflow of using VPSim is as follows: the user describes the targeted platform via Python front-end platform composition. This platform description must include details about the simulated components, the connections between these components, the simulation parameters, and the software that will run on the virtual platform. After, VPSim builds a simulation using its library of components. Once the simulation is set up, the user can control it at a high level, including launching, stopping, and pausing the simulation and debugging and profiling the software. Additionally, the user can access various run-time statistics that can be used for performance evaluation and design space exploration.

Full RISC-V based platform

Figure 2b describes a minimal system with one RISC-V core, bus, memory, and UART. VPSim provides the special components (**BlobLoader**,**ElfLoader**) to load flat and ELF binaries. We can load and execute the app binary on the simulated platform. Many statistics can be retrieved : real execution time & simulation time of the app, memory reads/writes, stats from interconnect and cpu,etc. Listing 1 shows a composition code snippet of a single RISC-V core platform. A corresponding Python class represents each component(`Memory`,`Bus`, `etc`) imported from VPSim binary.

Listing 1: *Snippet of a basic single RISC-V core platform composition in VPSim*

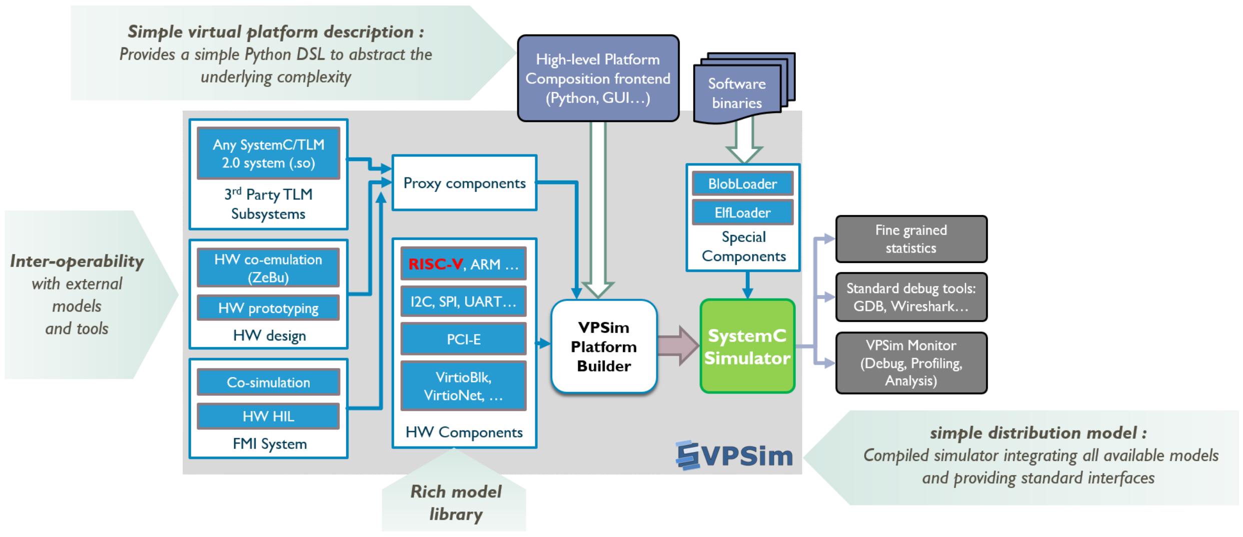
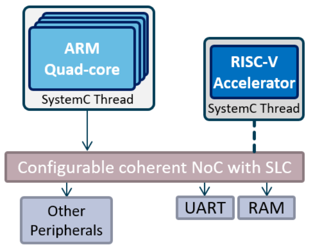
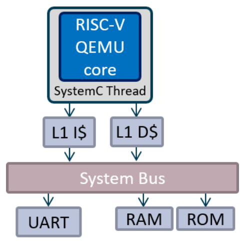

Simple virtual platform description :
Provides a simple Python DSL to abstract the underlying complexity
High-level Platform Composition frontend (Python, GUI…)
Software binaries
Any SystemC/TLM 2.0 system (.so)
3rd Party TLM Subsystems
Proxy components
BlobLoader
ElfLoader
Special Components
Fine grained statistics
Standard debug tools: GDB, Wireshark…
VPSim Monitor (Debug, Profiling, Analysis)
Inter-operability with external models and tools
HW co-emulation (ZeBu)
HW prototyping
HW design
RISC-V, ARM …
I2C, SPI, UART…
PCI-E
VirtioBlk, VirtioNet, …
HW Components
VPSim Platform Builder
SystemC Simulator
Co-simulation
HW HIL
FMI System
VPSim
simple distribution model :
Compiled simulator integrating all available models and providing standard interfaces
Rich model library

Figure 1: *Virtual Prototyping Simulator (VPSim) overview.*



(a) *ARM-based platform with RISC-V accelerator*



(b) *RISC-V single core*

Figure 2: *Simulated RISC-V based platforms in VPSim*

```python
import vpsim
class RiscvPlatform(vpsim.System):
  def __init__(self, app):
    core = vpsim.RiscV('cpu', cpu_id=0, iss=iss)
    sysbus = vpsim.Interconnect(latency= , ..)
    ram = vpsim.Memory(base_address=, size=, ..)
    # Connect components
    core('to_icache') >> sysbus
    core('to_dcache') >> sysbus
    sysbus >> ram
    ...
```

RISC-V core as an accelerator

Figure 2a presents a quad-core ARM-based platform with a RISC-V core connected as an accelerator. The aim is to demonstrate the ability of VPSim to compose platforms and perform data offloading. First, the host ARM-based platform prepares series of data and operations to be performed on the accelerator side. These series are then offloaded to the memory associated with the RISC-V core. Afterwards, the host initiates computation by writing to a specific control register. Once the RISC-V core has finished processing the requested operations, it raises an interrupt to the host that in turn gathers and checks the obtained results. Finally, execution statistics (ARM cores, RISC-V core and memory hierarchies) are retrieved and displayed by the end of simulation.

Discussion

Virtual prototyping simulators are essential tools for designers, allowing them to rapidly test and optimize their designs in a virtual environment, reducing the time and cost associated with physical prototypes conception and testing. As RISC-V continues to gain popularity, tools like VPSim will become increasingly important in the design process. By prioritizing user-friendliness, integration, adaptability, and user feedback, VPSim can still be improved with more features.

References

[1] Fast virtual prototyping for embedded computing systems design and exploration, A. Charif, et. al, RAPIDO/HIPEAC 2019.

[2] Multilevel simulation-based co-design of next generation HPC microprocessors, L. Zaourar, et. al., PMBS/SC2021.

[3] Co-simulation of a Model Predictive Control System for Automotive Applications, C. Bernardeschi et. al.,SEFM/CoSim-CPS, 2022.

[4] Decoupling processor and memory hierarchy simulators for efficient design space exploration, F. Jebali et. al., RAPIDO/HIPEAC 2022.

[5] S. Salah Eddine et. al., Fast Virtual Prototyping of Cyber-Physical Systems Using and FMI: ADAS Use Case, RSP'19.