UNTETHER AI

Enhanced Dynamic Hybrid Simulation Framework for Hardware-Software Verification

Victor Besyakov
Untether AI Toronto, Ontario, Canada
https://www.untether.ai.

2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES
VIRTUAL
FEBRUARY 28 - MARCH 3, 2022

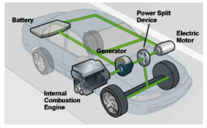
Background

At best, a hybrid simulation can be described as a combination of "***two or more different simulation techniques into one simulation environment in order to benefit from the different simulation characteristics, e.g., different levels of accuracy and simulation speed***"
Conceptually, it can be categorized into two groups:

- ❑ static or spatial
- ❑ dynamic or temporal

Static hybrid simulation (traditional approach) - offers the possibility to simulate system "*at different levels of abstraction*" User can decide for each simulation run "*if an abstract simulation is sufficient or if a detailed simulation is necessary*" .**Issue: Simulation performance is always limited by the "slowest" part.**

Dynamic hybrid simulation – "*offers the user the possibility to switch the simulation technique, i.e. the level of abstraction, during runtime*".
Ideally, a Dynamic hybrid simulation should be capable "to speed up the simulation without sacrificing the accuracy".
Static and semi-dynamic hybrid simulations are special cases of the dynamic hybrid simulation.
Issue: Switching between "fast" and "slow" simulation "is a very complex and architecture dependent task , since the entire state of the simulated program" must be coherent for all hybrid simulation engines.



Acronyms & references

- ❑ ISS - Instruction Set Simulator (https://en.wikipedia.org/wiki/Instruction_set_simulator)
- ❑ TLM or TLM 2.0 - Transaction-Level Modeling (https://en.wikipedia.org/wiki/Transaction-level_modeling)
- ❑ IPC - Inter-Process Communication (https://en.wikipedia.org/wiki/Inter-process_communication)
- ❑ "The Shunt" - Client/Server TCP/IP socket based communication library (https://github.com/xver/Shunt)
- ❑ Kraemer, S. Design and Analysis of Efficient MPSoC Simulation Techniques. Ph. D. Dissertation, RWTH Aachen Univeristy 2011

Why Dynamic Hybrid Simulation ?

- ❑ ***Enables*** shift left HW development and debug on SoC RTL rather than on first silicon and/or prototyping
- ❑ ***Provides*** a more realistic stimulus for RTL verification
- ❑ ***Reduces*** simulation time and Allows finding/fixing system-level bugs in HW, rather than as SW
- ❑ ***Creates*** a common platform for HW, SW development, and prototyping (emulation) and Enables HW and SW early integration

Enhanced dynamic hybrid platform architecture

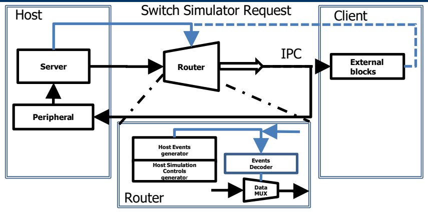


Figure. 1 Generic dynamic hybrid platform

Hybrid platform (refer to Figure 1 and Figure 2)consists of two main subsystems, called **Host** and **Client**, which communicate with each other via a **Data Router** and the **IPC** channels (dotted lines).
The implementation of the **IPC** may vary depending on system requirements. The most conventional approach is to use a standard TCP/IP socket API.
In a typical hybrid system, the **Host** subsystem is the fastest part of the simulation. The **Server** can be either an ISS or even an actual processor unit.
The reconfigurable **Data Router** governs data flow between **Host** and **Client**. It also implements an adaptive layer that maps associated interfaces to the **IPC** protocol. In the case of static simulation, the direction of the data stream always remains the same. For dynamic simulation, data flows can be altered based on the Switch Simulator request events see Table one and Figure 2.

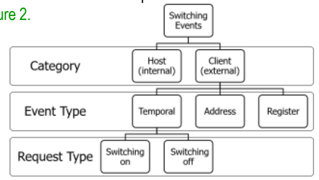


Figure. 2 Events Hierarchy

TABLE ONE TYPES AND SWITCHING EVENTS

Event Name	Transaction type	Event Trigger
Temporal	N/A	If the simulated time is within the specified interval.
Address	Read and Write Transaction	If the transaction address falls within the specified address range.
Register	Read and Write Transaction	If the address and the value of the present transaction are equal to the corresponding values in the reference table.

Hybrid Simulation Framework implementation Example

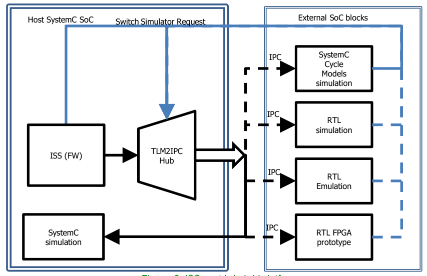


Figure. 3 ISS centric hybrid platform

Framework Components: Data Router

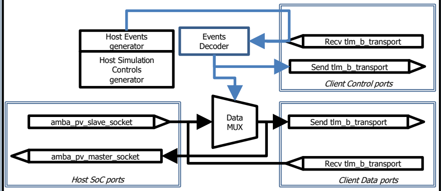
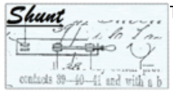


Figure. 4 Router block diagram (one Client example)

The **Data Router** manages data traffic between three types of input-output ports (see Figure. 4 and Figure. 5).
The **Client Control ports** are dedicated to the exchange of control information between the **Client** and **Host** subsystems. This Control flow consists of simulation controls and routing events. Each external port pair interacts independently with the data router and has its own TCP/IP socket.
The second pair is a SystemC simulation data path (Host SoC ports). They enable a connection between the Fast CPU model and the rest of the SoC Host system.
In operational terms, data routing can be static (default mode) or dynamic. When the simulation is launched, the data router obtains the default routing table configuration in the initialization step. In the case of a static hybrid simulation, the routing table remains unchanged during the entire simulation.
With dynamic routing, **Data MUX** updates the routing table on a regular basis based on information from internal or external switching events. The event receiver is responsible for processing all events from internal sources and external client control ports.
In addition to the traffic directing function, the router is responsible for establishing the TCP/IP socket-based communication. It also records and stores inbound and outbound transactions in the traffic database, and manages (accepts or rejects) Direct Memory interface (DMI) requests from System C. The external data ports (**Client Data Ports**) and the operation of the data ports are described in this poster (see Host-Client data flow).

Framework Components: Inter-Process Communication



The open-source communication library "The Sunt" *is* selected as the TCP/IP application layer. It covers all SystemC, System Verilog, and C clients and facilities to establish TLM 2.0 connections. The main reason for selecting a TCP/IP protocol is that the hybrid platform requires communication between remote applications.

The primary objective of the IPC is to provide a unified reusable data exchange protocol for all potential hybrid platform clients. In practice, the selection of the communication methods is a compromise between simulators interoperability, performance, and data transfer overhead, just to name a few.

Host-Client data flow

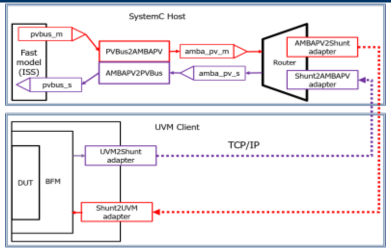


Figure. 5 Host-Client data flow

An example of client-host integration is shown in Figure 5

Host subsystem is represented by the ARM ISS Fast Processor model . It interacts with a hybrid system's peripherals through the ARM Programmer View (PV) ports (**pvbus_m** and **pvbus_s**).
The PV Bus provides functionally accurate communication between bus masters and slaves, but doesn't have any notion of the data exchange protocol.

The PVBus2AMBAPV and AMBAPV2PVBus components implement the required protocol conversion, such as PV to AMBA and vice versa.

Host-to-Client Read request (red lines in Figure 5 **) :**

Host: Read request is sent from the processor model output to the ama_pv_m port via the PVBus2AMBAPV protocol converter.

The Router fetches the AXI bus transaction and forwards it to the AMBAPV2Shunt adapter. It converts read request from the amba_pv and sent it over TCP/IP to the corresponding client.

Client: The Shunt2UVM adapter gets an ingress transaction, converts it to the regular uvm_tlm_generic_payload, and passes it to AXI BFM.

Client -to-Host Read response (purple lines in Figure 5):

Then BFM returns the results of the read transaction. The following components of the data flow diagram will be involved: UVM2Shunt adapter, TCP/IP packet, Shunt2AmbaPV adapter, amba_pv_s port, AMBAPV2PVBus, and finally the PVbus input port of the processor.