CREATING HETEROGENEOUS SIMULATIONS WITH SST AND SYSTEMC

Sabbir Ahmed
Booz Allen Hamilton
ahmed_sabbir@bah.com

Noel S. Wheeler
Laboratory for Physical Sciences
nwheeler@lps.umd.edu

Robert P. Mrosky
Laboratory for Physical Sciences
rmrosky1@lps.umd.edu

ABSTRACT

Implementing new computer system designs involves careful study of both programming models and hardware design and organization, a process that frequently introduces distinct challenges. Hardware and software definitions are often simulated to undertake these difficulties. Structural Simulation Toolkit (SST), a parallel event-based simulation framework that allows custom and vendor models to be interconnected to create a system simulation [1], is one such toolkit. However, SST must be able to support models implemented in various frameworks and languages. SystemC is a popular hardware-level modeling language composed of C++ classes and macros [2]. Establishing communication with SystemC modules would allow SST to interface numerous existing synthesizable hardware models. SST-SystemC Interoperability Toolkit (SSTSCIT) is a library developed to provide interoperability between SST and SystemC. SSTSCIT aims to achieve this capability in a modular design without interfering with the kernels by concealing the communication protocols in black box interfaces. This project includes a demonstration of the interoperability by simulating a vehicular traffic intersection with traffic lights driven by SystemC processes. The modular implementation of the black box interface allowed for sufficient flexibility in establishing communication between the different systems. This design can, therefore, be configured to interoperate SST with various model simulation frameworks and even hardware to achieve further heterogeneity.

1 Introduction

The increasing size and complexity of systems require engineers heavily rely on simulation techniques during the development phases. Typically, simulations of these complex systems require both custom and off-the-shelf logic functionality in ASICs or FPGAs. High-level commercial tools simulate and model these components in their native environments. On the other side, developers create the register transfer level (RTL) models representing the systems to simulate them with computer-aided design (CAD) tools and test benches. These duplicative strategies require a method that simulates the entire system in one heterogeneous model.

Successful attempts have made to establish interoperability between Structural Simulation Toolkit (SST) and the Python-based RTL language, PyRTL [3]. This project establishes interoperability between SST and SystemC and demonstrates the possibility of extending to further systems due to its modular design. SST is an event-based framework that has the capabilities to simulate not only functionality but timing, power or any other information required. Each SST components can be assigned a clock to synchronize tasks. They communicate events with each other via SST links by triggering their corresponding event handlers. The SST models are constructed in C++ and consist of the functionality of the element, the definition of each links' ports and the event handlers. The models are connected and initialized through the SST Python module. SystemC is a set of C++ classes and macros that deliberately mimics hardware description languages like VHDL and Verilog. The system-level modeling language provides an event-driven simulation kernel along with signals, events and synchronization primitives. Implementing a heterogeneous system to synchronize signals and events between the two kernels would allow the developers to work cooperatively and efficiently.

2 Black Box Interface

SSTSCIT conceals the communication implementation in black box driver files. This strategy allows the SST component to connect with the SystemC process via SST links as if it were a component itself.

The interface consists of:

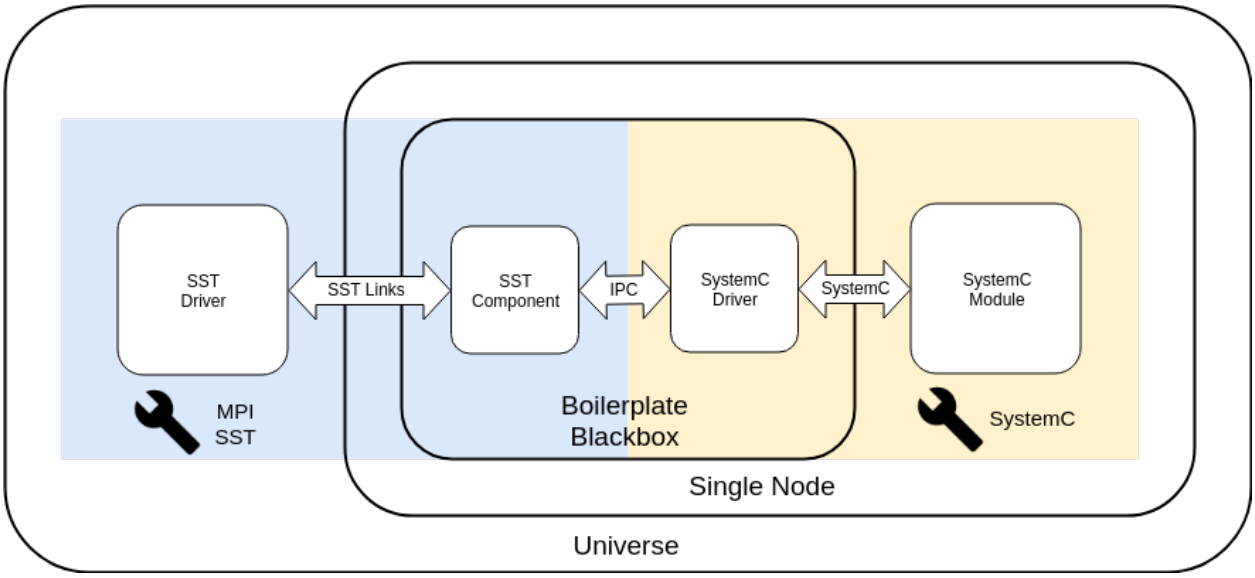

Figure 1: Components of SSTSCIT

1. A SystemC driver
2. An SST component

Each SystemC modules must have their corresponding driver file to interoperate within the black box interface. It is possible to interoperate multiple SystemC modules with a single driver file. However, the additional communication lines must be accounted for in the corresponding black box SST component.

The toolkit includes a Python class that generates the boilerplate code required for the black box interface.

3 Communication

3.1 Inter-Black Box Communication

The two components inside the black box interface are spawned in the same node and therefore communicate via interprocess communication (IPC) transports. The following is a list of supported IPC transports:

1. Unix domain sockets
2. ZeroMQ

It is possible to add custom IPC protocols to the interface by implementing a derived class of `sigutils::SignalIO` with customized sending and receiving methods.

3.2 SST-Black Box Communication

An SST component can interface the black box via standard SST links.

3.3 SystemC-Black Box Communication

A SystemC module can be interfaced by a standard source file inclusion.

4 Proof of Concept: Traffic Intersection Simulation

A simulation model has been developed to demonstrate the project. The model simulates a traffic intersection controlled by two traffic lights. A flow of traffic is simulated through a road only when its traffic light generates a green

Figure 2: Black Box Interface Data Flow Diagram. Arrows Highlighted in Red Indicate Communication Signals.

or yellow light with the other generating a red light. The number of cars in a traffic flow is represented by random number generators.

The concept of this simulation is derived from the original project that established interoperability between SST and PyRTL.[3]

4.1 SystemC Drivers

The simulation project includes a SystemC module and its driver, `traffic_light_fsm`, that interacts with the SST component `traffic_light`. The module is a clock-driven FSM of three states representing the three colors of a traffic light: green, yellow and red. The FSM proceeds to the next state when indicated by its internal counter initialized in the beginning. The input variables to the module include: the three durations for the three colors of the light, `green_time`, `yellow_time` and `red_time`, the preset variable `load` to initialize the FSM, and `start_green` to indicate if the first state should be "green" or "red".

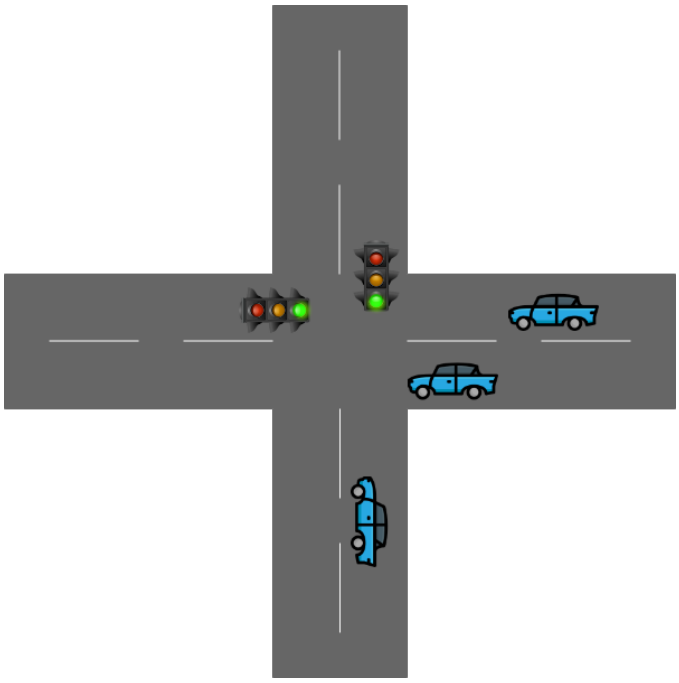

Figure 3: Simple Two-Road Intersection

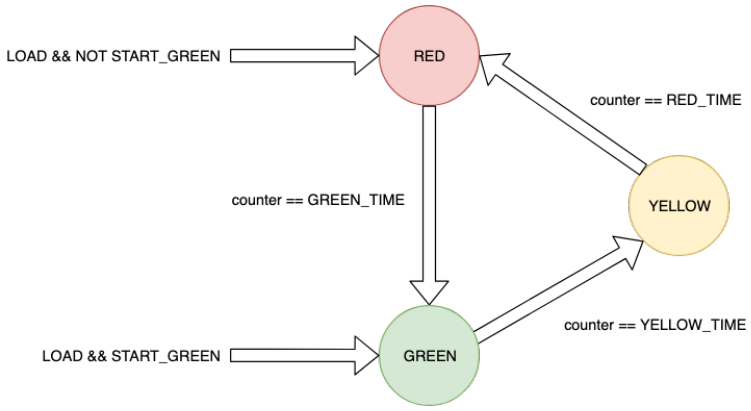

Figure 4: Traffic Light Finite State Machine

4.2 SST Components

The project also consists of three SST components: `car_generator`, `traffic_light_controller` and `intersection`. All the components with the exception of most of `traffic_light_controller` were inherited from the original project.

4.2.1 car_generator

The `car_generator` component consists of a random number generator that yields 0 or 1. The output is redirected to `intersection` via SST links.

4.2.2 traffic_light

The `traffic_light` component generates the light colors of the traffic lights using a simple finite state machine (FSM). The component delegates the FSM portion of its algorithm to the SystemC module `traffic_light_fsm` and inter-procedurally communicates with it via UNIX domain sockets. The component initializes the FSM with the SST parameters and sends its output to `intersection` via SST links every clock cycle.

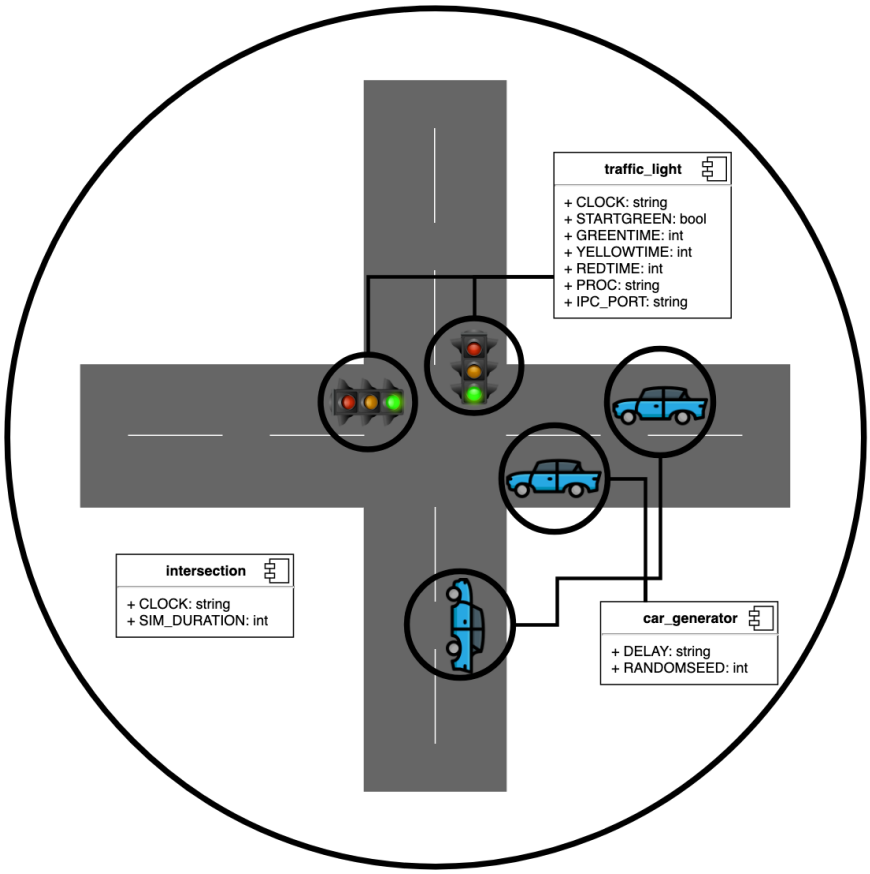

Figure 5: Simple Two-Road Intersection Represented by SST Components

4.2.3 intersection

`intersection` is the main driver of the simulation. The component is able to handle n instances of `traffic_light` subcomponents and therefore expects the same number of `car_generator` instances. For the purposes of this simulation, two instances of the subcomponent pairs are set up. The driver keeps track of the number of cars generated and the color of the light per clock cycle for each subcomponent pairs and stores them in local variables. The variables are summarized in the end to generate statistics about the simulation.

The component does not check for any collisions in the intersection, i.e. comparing if both the `traffic_light` subcomponents yield "green" during the same clock cycle. The SST Python module is responsible for setting up the `traffic_light` components with the proper initial values.

4.3 Example Simulation

A sample output of the simulation has been generated and provided below. The SST components were linked along with the parameters in the SST Python module.

```
traffic_light-Traffic Light 0 -> GREENTIME=30, YELLOWTIME=3, REDTIME=63, STARTGREEN=0
traffic_light-Traffic Light 1 -> GREENTIME=60, YELLOWTIME=3, REDTIME=33, STARTGREEN=1
car_generator-Car Generator 0 -> Minimum Delay Between Cars=3s, Random Number Seed=151515
car_generator-Car Generator 1 -> Minimum Delay Between Cars=5s, Random Number Seed=239478
intersection-Intersection -> sim_duration=24 Hours
intersection-Intersection -> Component is being set up.
traffic_light-Traffic Light 0 -> Component is being set up.
traffic_light-Traffic Light 0 -> Forking process "/path/to/traffic_light_fsm.o"...
traffic_light-Traffic Light 0 -> Process "/path/to/traffic_light_fsm.o" successfully synchronized
traffic_light-Traffic Light 1 -> Component is being set up.
traffic_light-Traffic Light 1 -> Forking process "/path/to/traffic_light_fsm.o"...
traffic_light-Traffic Light 1 -> Process "/path/to/traffic_light_fsm.o" successfully synchronized
```

```
intersection-Intersection -> ------------------------------------
intersection-Intersection -> -------- SIMULATION INITIATED --------
intersection-Intersection -> ------------------------------------
intersection-Intersection -> Hour | Total Cars TL0 | Total Cars TL1
intersection-Intersection -> -----+---------------+---------------
intersection-Intersection ->    1 |           618 |           369
intersection-Intersection ->    2 |          1238 |           719
intersection-Intersection ->    3 |          1851 |          1074
intersection-Intersection ->    4 |          2432 |          1426
intersection-Intersection ->    5 |          3041 |          1774
intersection-Intersection ->    6 |          3688 |          2121
intersection-Intersection ->    7 |          4290 |          2467
intersection-Intersection ->    8 |          4892 |          2813
intersection-Intersection ->    9 |          5467 |          3175
intersection-Intersection ->   10 |          6054 |          3525
intersection-Intersection ->   11 |          6644 |          3885
intersection-Intersection ->   12 |          7228 |          4233
intersection-Intersection ->   13 |          7813 |          4607
intersection-Intersection ->   14 |          8435 |          4973
intersection-Intersection ->   15 |          9047 |          5337
intersection-Intersection ->   16 |          9656 |          5691
intersection-Intersection ->   17 |         10255 |          6059
intersection-Intersection ->   18 |         10843 |          6428
intersection-Intersection ->   19 |         11448 |          6791
intersection-Intersection ->   20 |         12025 |          7140
intersection-Intersection ->   21 |         12617 |          7499
intersection-Intersection ->   22 |         13225 |          7867
intersection-Intersection ->   23 |         13807 |          8223
intersection-Intersection ->   24 |         14400 |          8580
intersection-Intersection ->
intersection-Intersection -> ----------------------------------------
intersection-Intersection -> ---------- SIMULATION STATISTICS ----------
intersection-Intersection -> ----------------------------------------
intersection-Intersection -> Traffic Light | Total Cars | Largest Backup
intersection-Intersection -> --------------+------------+--------------
intersection-Intersection ->             0 |      14400 |            18
intersection-Intersection ->             1 |       8581 |             7
intersection-Intersection -> Destroying Intersection...
car_generator-Car Generator 1 -> Destroying Car Generator 1...
car_generator-Car Generator 0 -> Destroying Car Generator 0...
traffic_light-Traffic Light 0 -> Destroying Traffic Light 0...
traffic_light-Traffic Light 1 -> Destroying Traffic Light 1...
Simulation is complete, simulated time: 86.4 Ks
```

Listing 1: Sample Simulation Output

References

[1] SST Simulator - The Structural Simulation Toolkit. `sst-simulator.org`.

[2] 1666-2011 - IEEE Standard for Standard SystemC Language Reference Manual. IEEE, 9 Jan. 2012, `standards.ieee.org/standard/1666-2011.html`.

[3] Mrosky, Robert P, et al. *Creating Heterogeneous Simulations with SST and PyRTL.*