SST-SystemC Interoperability Toolkit

Sabbir Ahmed

April 23, 2019

Abstract

SST-SystemC Interoperability Toolkit (SSTSCIT) is a collection of header files developed to provide interoperability between Structural Simulation Toolkit (SST) and SystemC. SST is a parallel event based simulation framework that allows custom and vendor models to be interconnected to create a simulation environment [1]. SystemC is a system-level modeling language composed of C++ classes and macros [2]. SSTSCIT aims to provide the capability to interoperate the two systems without interfering with any of the kernels by concealing the communication protocols in black box interfaces. This project provides a demonstration of the interoperability by simulating a traffic intersection, where the traffic lights are determined by SystemC processes.

1 Introduction

This collection of header files provides methods to transmit and receive signals between SST components and SystemC modules. The toolkit provides a black box interface that can be interfaced with both SST and SystemC via their internal communication transports.

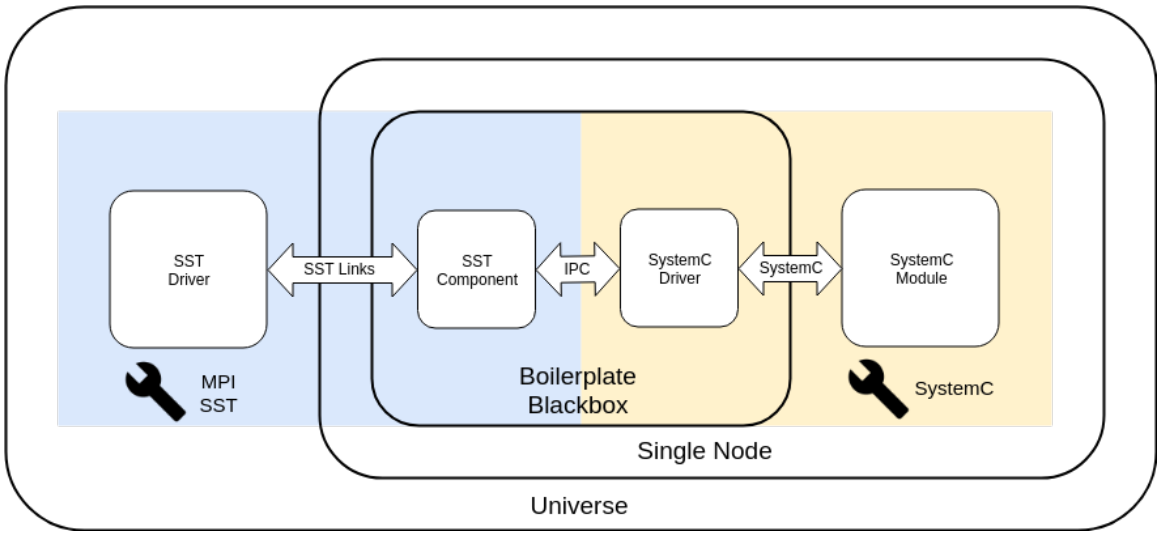

Figure 1: Components of SSTSCIT

2 Components

2.1 Black Box Interface

The black box interface consists of:

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.

2.2 UML

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 model can interface the black box via standard SST links.

The following snippets demonstrate an SST link transmitting a unidirectional signal from the SST environment to the black box interface.

```
// parent_sst.cpp

// register a string event in the class declaration
SST_ELI_DOCUMENT_PORTS(
    { "demo_din", "Demo model data in", { "sst.Interfaces.StringEvent" }},
    ...
)

// initialize the link in the class declaration
SST::Link *demo_din;

// set up the SST link in the constructor
demo_din = configureLink("demo_din");

// trigger the event in the clocked function
demo_din->send(new SST::Interfaces::StringEvent(...));
```

```
// blackboxes/demo.cpp

// register the same string event in the class declaration
SST_ELI_DOCUMENT_PORTS(
    { "demo_din", "Demo model data in", { "sst.Interfaces.StringEvent" }},
    ...
)

// initialize the same link in the class declaration
SST::Link *demo_din;

// set up the SST link in the constructor with an event handler
demo_din = configureLink(
```

```
    "demo_din",
    new SST::Event::Handler<demo>(this, &demo::handle_event)
);

// receive and parse the event in the event handler
void demo::handle_event(SST::Event *ev) {
    auto *se = dynamic_cast<SST::Interfaces::StringEvent *>(ev);
    if (se) {
        std::string _data_in = se->getString();
        ...
    }
    delete ev;
}
```

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

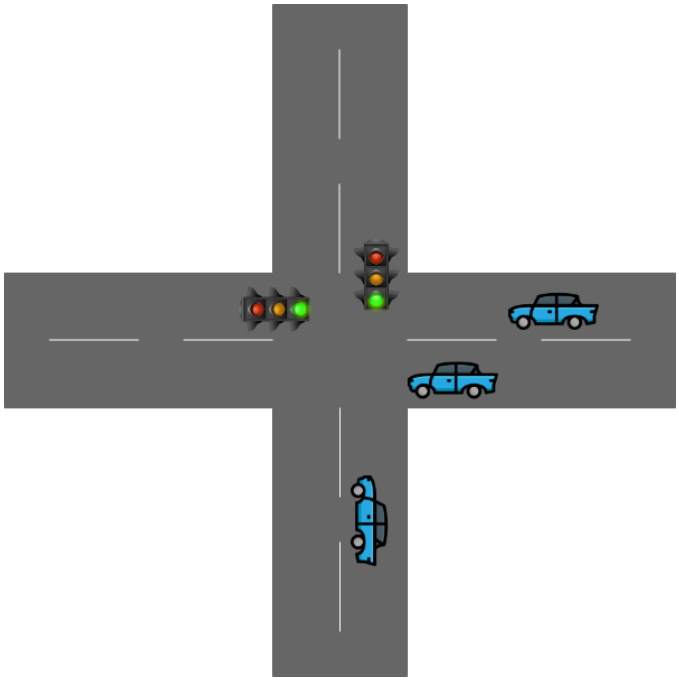

Figure 2: Simple Two-Road Intersection

The concept of this simulation is derived from the original project that established interoperability between SST and PyRTL - a Python based hardware description language.[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".

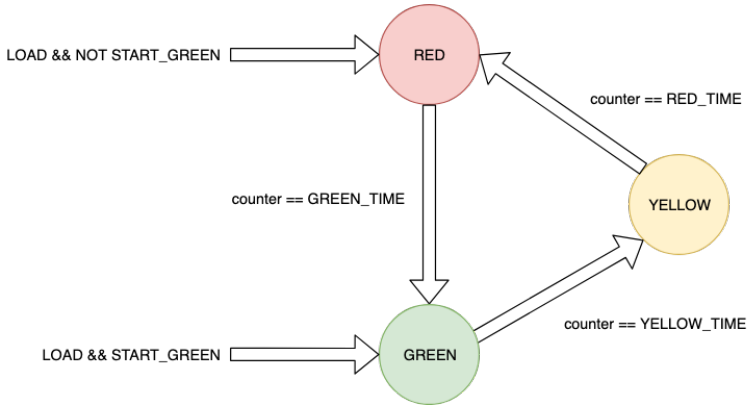

Figure 3: Traffic Light Finite State Machine

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.

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.

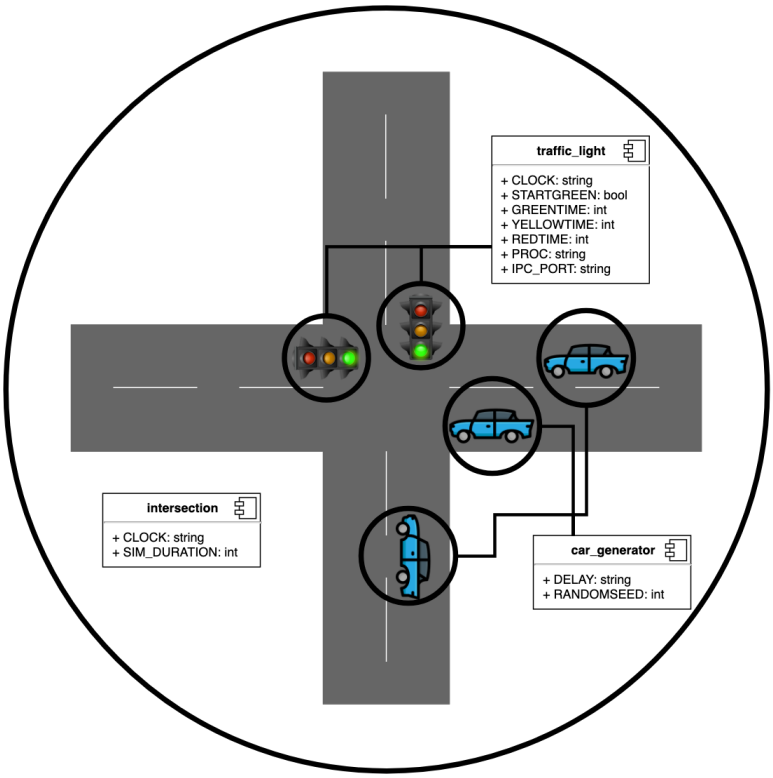

Figure 4: Simple Two-Road Intersection

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.*