SystemC Programming

Naveen Yadav

February 3, 2026

1 What is SoC?

An SoC is literally a *system on a chip*, consisting of both silicon (an integrated circuit that integrates most or all components of a computer or electronic system.) and embedded software. The following components are usually included on a single substrate or microchip:

1. On-chip central processing unit (CPU),

2. Memory interfaces,

3. Input/output (I/O) devices and interfaces,

4. Secondary storage interfaces,

5. Components such as radio modems and a graphics processing unit (GPU).

In general, such systems may contain

1. Digital circuits to process signals that represents data as a *sequence of discrete values*; at any given time it can only take on, at most, one of a finite number of values.

2. Analog circuits to process signals that represent a quantity as a *continuous-time* signal. For example, in an analog audio signal, the instantaneous signal voltage varies continuously with the pressure of the sound waves.

3. Mixed-signals circuits to process both analog and digital circuits on the same integrated circuit (IC). For example, an analog-to-digital converter (ADC) is a typical mixed-signal circuit.

SoC design involves complex algorithm and architecture development and analysis similar to that performed in *system design*. In a nutshell, it involves understanding of component parts and their subsequent interaction with one another.

2 What is SystemC?

It is a language for for System-level design, modeling and verification.

3 Compiling SystemC programs

Lets assume that we have built an environment module for SystemC

```
 -------------------------------------------------------------------
/home/nyadav/privatemodules/systemc:
setenv          SYSTEMC_HOME /home/nyadav/opt/SystemC
setenv          SYSTEMC_INCLUDE /home/nyadav/opt/SystemC/include
setenv          SYSTEMC_LIB /home/nyadav/opt/SystemC/lib64
setenv          SYSTEMC_MAN /home/nyadav/opt/SystemC/share
prepend-path    LD_LIBRARY_PATH /home/nyadav/opt/SystemC/lib64
prepend-path    C_PATH /home/nyadav/opt/SystemC/include
prepend-path    C_INCLUDE_PATH /home/nyadav/opt/SystemC/include
prepend-path    CPLUS_INCLUDE_PATH /home/nyadav/opt/SystemC/include
prepend-path    MANPATH /home/nyadav/opt/SystemC/share
prepend-path    CMAKE_MODULE_PATH /home/nyadav/opt/SystemC/lib64/cmake
```

4 SystemC Tutorial

I am following this website: Learn SystemC

SystemC header file: To use the SystemC class library features, an application shall include either of the `C++` header files specified below:

Listing 1: Including the SystemC header.

```
/* Add all of the names from the namespaces
   sc_core and sc_dt to the declarative region
   in which it is included. The namespace sc_core
   is defined inside sc_object.h -- Abstract base class
       of all SystemC 'simulation' objects.
*/
#include <SystemC.h>
/* SystemC.h is provided for backward
   compatibility with earlier versions of
```

```
    SystemC and may be deprecated in future
    versions of this standard. The better
    way to do it is using ‘SystemC‘ header.
*/
#include <SystemC>
```

SystemC entry point: While a normal C++ programs entry point is the main () function, SystemC user has to use

```
int sc_main(int argc, char* argv[])
```

as the entry point. This is because SystemC library has the main() function already defined, therefore main() will call sc_main() and passes the command-line parameters.

4.1 SystemC module

A SystemC module is a class (or struct) that inherits the sc_core :: sc_module base class. A SystemC module is the *smallest container of functionality with state, behavior, and structure for hierarchical connectivity*. It is the *principle* structural building block of SystemC. Syntactically, it is a C++ class, which inherits a SystemC basic class, the sc_core :: sc_module, and is used to represent a component in real systems.

4.2 How to define a SystemC module

A SystemC module can be defined in three ways

1. As a struct inheriting a SystemC basic class, the sc_core :: sc_module

   ```
   struct module_name: public sc_core::sc_module {};
   ```

2. As a class inheriting a SystemC basic class, the sc_core :: sc_module

   ```
   struct module_name: public sc_core::sc_module {};
   ```

3. Using the SystemC defined macro SC_MODULE, which is equivalent to the first method of defining a SystemC module

   ```
   SC_MODULE(module_name) {};
   ```

The first two methods are similar except that the members of a struct have public access by default, while the members of a class have private private by default.

4.3 How to use a SystemC module

SystemC module objects have certain properties which sets them apart from normal classes.

1. Objects of class sc_core::sc_module can only be constructed during elaboration[1]. It is an error to instantiate a module during simulation, as module instantiation happens during elaboration phase.
2. Every class derived (directly or indirectly) from sc_core::sc_module must have at least one constructor.
3. Every constructor must have *one and only one* parameter of class sc_core::sc_module_name, however it may have further parameters of classes other than class sc_core::sc_module_name.
4. While creating an instance of SystemC module, a string-valued argument must be passed to the constructor[2].

4.4 SystemC Module Constructor

Each C++ classes has a constructor – a special member function used to initialize objects, such that each data member is given a well-defined initial value. A *default* constructor is *auto-generated* if an explicit constructor is not provided. As discussed above, every SystemC module must have a *unique name*, which is provided when instantiating a module object. Therefore, we need a constructor with atleast one parameter (of type sc_core::sc_module_name).

SC_CTOR: For convenience, SystemC provides a macro (SC_CTOR) when declaring or defining a constructor of a module. The macro SC_CTOR is defined as

Listing 2: Macro SC_CTOR definition.

```
// Function like macros (note the newline usage)
#define SC_CTOR(module) \
typedef module SC_CURRENT_USER_MODULE; \
```

[1] Elaboration is the execution of statements prior to sc_core:: sc_start (). The primary purpose is to create internal data structures to support the semantics of simulation. Also check out the meaning in VHDL and FPGA terminology here: https://vhdlwhiz.com/terminology/elaboration/#elaboration.

[2] It is good practice to make this string name the same as the C++ variable name through which the module is referenced, if such a variable exists.

```
module(::sc_core::sc_module_name)
```

The macro SC_CTOR has limitations:

1. It can only be used where C++ rules permit a constructor to be declared.
2. It has only one argument, the name of the module class being constructed.
3. It can not add user-defined arguments to the constructor[3].
4. Since SC_CTOR has a constructor function declaration, it can only be placed inside class header.

SC_HAS_PROCESS: In many cases, we may want to have a constructor with additional arguments. For this, SystemC v2.0 provides an alternative constructor using a C++ macro called SC_HAS_PROCESS that is defined as

Listing 3: Macro SC_HAS_PROCESS definition.

```
// Function like macros (note the newline usage)
#define SC_HAS_PROCESS(module) \
typedef module SC_CURRENT_USER_MODULE;
```

In both the cases, the module name (which is a type in itself in C++) is further defined as SC_CURRENT_USER_MODULE, as it allows to register member functions to simulation kernel via SC_METHOD/SC_THREAD/SC_CTHREAD.

[3] Module objects which need additional arguments for instantiation must be provided such constructors explicitly.