Chapter 8

SYSTEMC KERNEL APPLICATION PROTOCOL INTERFACE (API)

Designed to contain only one simulation kernel, SystemC does not provide an API to tidily add an extension to the simulation kernel. Nor are we aware of any efforts of providing an API for kernel development in SystemC. The existing simulation kernel is specified in the *sc_simcontext* class along with several global function definitions such as *sc_start(...)*. We provide an API that better incorporates multiple kernels and provides a medium through which kernels can gain access to its counterpart kernels. Once again, our approach is in limiting the changes in the current source by overlaying the current implementation with our API with the introduction of a class called *sc_domains*. The problems that we address by adding this encapsulating API class are as follows:

1 For each implemented kernel, one should be able to execute each of them independent of others.

2 Every implemented kernel must be able to access every other kernel for multi-domain heterogeneous simulation.

1. System Level Design Languages and Frameworks

Our API class that we call *sc_domains* is shown in Listing 8.1 and a class diagram is shown in Figure 8.

The *sc_domains* class contains pointers to each implemented kernel. We present two methods by which a kernel can be represented. The first is by dynamically allocating the kernel such as in [Listing 8.1, Line 45] for the *de_kernel* or by having a list of multiple kernels as in [Listing 8.1, Line 22] for *sdf_domain*. The *sdf_domain* could have been easily

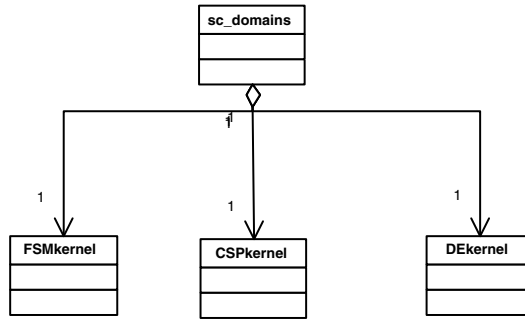

Figure 8.1. Class Diagram for *sc_domains*

implemented as a pointer to a *vector* list, but we choose to do this to show that this approach can also be used. Furthermore, not all kernels require more than one instance valid in the system. For example, the DE kernel requires only one instance of itself. However, there can be multiple SDF or FSM models in a heterogeneous model requiring multiple SDF graphs (SDFG) to be present in the system same applied with multiple FSM models.

Initialization functions and its helper functions must all be member functions of the class *sc_domains*. An example of a helper function is *split_processes()* that is an SDF initialization helper function. Its purpose is to separate SDF method processes from regular DE method processes. Similarly, *find_sdf_graph()* is a helper function that finds a specific SDFG in the entire model.

For a kernel designer it is important to adhere to some general guidelines in adding a kernel to SystemC using the API we provide. Most of these guidelines are intuitive. They are as follows:

1 Ensure that every added kernel is encapsulated in a class such as *sdf_graph*. This class must give enough access to the *sc_domains* class to enable execution of the functions from the *sc_domains* class.

2 Include a pointer (recommended) to an instance of a kernel type as a data member of *sc_domains*. If for some reason there can be multiple instances of the kernel, such as in the case of SDF then a list of pointers to the kernels can be used. Whether to use pointers to kernels or object instances of kernels in implementation is up to the kernel designer.

3 Initialization functions for the kernel must be called from function *init_domains* that is responsible for initializing all kernels. A kernel designer can implement additional functions specific for their kernel

Listing 8.1. API Class *sc_domains*

```
1
2 class sc_domains  {
3
4   // public functions
5   public:
6     sc_domains();
7     ~sc_domains();
8
9     void init_de();
10    void init_sdf();
11    void init_domains(const sc_time & duration, string in);
12
13    void split_processes();
14
15    sdf_graph * find_sdf_graph(string sdf_prefix);
16
17    // take the input from user
18     bool user_input(string in);
19
20    // make these private after the hack works
21    sc_simcontext* sdfde_kernel;  // DE kernel
22    vector<sdf_graph*> sdf_domain; // SDF kernel
23    void sdf_trigger(string topname);
24
25    string user_order;
26
27    // DE functions
28    void initializeDE();
29    bool isDEinitialized();
30
31    // CSP functions
32    void initializeCSP();
33    bool isCSPinitialized();
34
35    // FSM functions
36    void initializeFSM();
37    bool isFSMinitialized();
38
39    void clean_sdf(const string & str);
40    void clear_de();
41    DEkernel * get_de_kernel();
42    CSPkernel * get_csp_kernel();
43
44  private:
45    DEkernel * de_kernel;
46    CSPkernel * csp_kernel;
47    FSMkernel * fsm_kernel;
48 };
```

in *sc_domains*, for example in *sdf_domain*, *split_processes()* is used to split the runnable process list by removing the SDF method processes except for the top-level SDF method process.

Table 8.1 displays some of the member functions in *sc_domains* and their purposes. A file static instance of *sc_domains* is instantiated. The object is called the *Manager*. The API class acts as a manager having access to all models and MoCs that are currently in the system. All kernel instances have access to this *Manager* instance through which they

Listing 8.2. *init_domains* from *sc_domains* class

```
1// initial the domains
2void sc_domains::init_domains(const sc_time & duration, string
     in ) {
3
4  if (in.size() > 0)
5    user_input(in);
6
7  init_de();
8  split_processes();
9  init_sdf();
10 model.sdfde_kernel->de_initialize2();
11}
```

get access to any other kernel implemented and instantiated allowing for interaction between them. This way the *Manager* object can find a particular model in a specific MoC and execute it accordingly. This setup for *sc_domains* exists to allow for behavioral hierarchy for the future.

Table 8.1. Few Member Functions of class *sc_domains*

Functions	Description
sdf_trigger(...)	Global SDF specific function to execute the SDF graph.
init_domains(...)	Function that invokes all initialization member functions for every kernel in *sc_domains*.
split_processes()	SDF specific function to split SDF function block processes from regular SystemC method processes.
init_de()	Create an instance of the DE kernel.
init_sdf()	Initialization function for SDF kernel. Traverses all SDFGs and constructs an *executable schedule* if one exists.
find_sdf_graph(...)	Helper function to find a particular SDF graph for execution.
initializeCSP()	Prepare *csp_kernel* such that instances of CSP models can be inserted
initializeFSM()	Prepare *fsm_kernel* such that instances of FSM models can be inserted
get_de_kernel()	Return the pointer *de_kernel*.
get_csp_kernel()	Return the pointer *csp_kernel*.
get_fsm_kernel()	Return the pointer *fsm_kernel*.

We provide these guidelines to simply help kernel designers in using the introduced API and we impose no restrictions as to a particular method of addition. This *sc_domains* class shown in Listing 8.1 simply encapsulates all the kernel classes requiring a certain alteration to the

Listing 8.3. *init_sdf* from *sc_domains* class

```
1 void sc_domains::init_sdf() {
2
3   if (sdf_domain.size() == 0) {
4     schedule::err_msg(" No SDF system ", "WW");
5     return;
6   }
7
8   for (int sdf_graphs = 0; sdf_graphs < (signed) sdf_domain.
        size() ; sdf_graphs++) {
9
10    // Extract the address of first SDF
11    sdf_graph * process_sdf_graph = sdf_domain[sdf_graphs];
12    // Calculate schedule for this SDFG if one not already
        calculated
13    if (process_sdf_graph->result == NULL)
14      process_sdf_graph->sdf_create_schedule();
15  }
16}
```

Listing 8.4. *sdf_trigger()* from *sc_domains* class

```
1 void sc_domains::sdf_trigger(string topname)  {
2   string sdfname = topname+".";
3   sdf_graph * run_this;
4
5   for (int sdf_graphs = 0; sdf_graphs < (signed) model.
        sdf_domain.size(); sdf_graphs++) {
6     // pointer to a particular SDF graph
7
8     sdf_graph * process_sdf_graph = model.sdf_domain[sdf_graphs
          ];
9
10    if (strcmp(process_sdf_graph->prefix.c_str(), sdfname.c_str
          ())==0 ) {
11
12      run_this = process_sdf_graph;
13
14      if (( run_sdf == true)){
15        // execute the SDF METHODs
16        run_this->sdf_simulate(sdfname);
17        run_sdf = false;
18      }/* END IF */
19    }/* END IF */
20  }/* END FOR */
21}/* END sdf_trigger */
```

global function calls such as *sc_start(...)* and the introduction of MoC specific simulation functions such as *sc_csp_start(...)*. The addition of global and member functions in existing classes are described in the MoC's respective chapter and the API mainly supports the exchange of information about these multi-MoC models.

This API is not the most evolved nor is it the most robust, but it provides a mechanism and an approach to organizing and allowing kernel designers to think and consider additional improvements in managing

their kernel implementation in SystemC. The API is also not fully complete, for example, we do not support multiple models for the CSP MoC as yet. Ideally, we envision the *Manager* object to manipulate the client QuickThread coroutine instances as well, but these considerations are still under investigation.

Finally, Listings 8.2, 8.3 and 8.4 show some of the API functions.