Center for Embedded and Cyber-Physical Systems
University of California, Irvine

Generating Synthetic Data Flow Models in SystemC TLM based on TGFF

Yanda Li, Yutong Wang, Rainer Dömer

Technical Report CECS-24-02
May 9, 2024

Center for Embedded and Cyber-Physical Systems
University of California, Irvine
Irvine, CA 92697-2620, USA
(949) 824-8919

yandal5@uci.edu
http://www.cecs.uci.edu

Generating Synthetic Data Flow Models in SystemC TLM based on TGFF

Yanda Li, Yutong Wang, Rainer Dömer

Technical Report CECS-24-02
May 9, 2024

Center for Embedded and Cyber-Physical Systems
University of California, Irvine
Irvine, CA 92697-2620, USA
(949) 824-8919

yandal5@uci.edu
http://www.cecs.uci.edu

Abstract

While SystemC is the standard System-Level Description Language (SLDL) for embedded system design, openly accessible benchmark models in SystemC are rare. This report addresses this problem by providing an automatic model generator for SystemC Transaction Level Modeling (TLM). Taking a data flow graph produced by Task Graphs For Free (TGFF)[1] as input, the proposed model generator produces SystemC TLM 1.0 and 2.0 benchmark models. This report demonstrates that the generated models simulate correctly and can also be used as benchmarks for evaluating compiler designed for Grid of Processing Cells(GPC) many-core platform.

Contents

List of Figures

Generating Synthetic Data Flow Models in SystemC TLM based on TGFF

Yanda Li, Yutong Wang, Rainer Dömer
Center for Embedded and Cyber-Physical Systems
University of California, Irvine
Irvine, CA 92697-2620, USA
yandal5@uci.edu
http://www.cecs.uci.edu

Abstract

While SystemC is the standard System-Level Description Language (SLDL) for embedded system design, openly accessible benchmark models in SystemC are rare. This report addresses this problem by providing an automatic model generator for SystemC Transaction Level Modeling (TLM). Taking a data flow graph produced by Task Graphs For Free (TGFF)[1] as input, the proposed model generator produces SystemC TLM 1.0 and 2.0 benchmark models. This report demonstrates that the generated models simulate correctly and can also be used as benchmarks for evaluating compiler designed for Grid of Processing Cells(GPC) many-core platform.

1 Introduction

Embedded computer systems perform an essential role in people's daily lives due to their various applications, from personal devices to industrial operations. While computers have experienced many substantial improvements, one major feature of common computing systems is the processing cores and the main memory are separated. Since all the processing cores need to communicate through one central memory bus, a memory traffic jam problem that limits the computing speed heavily will happen. The Grid of Processing Cells(GPC) is a new processor architecture where processing cores are placed on the same chip in a 2D array structure and communicate through their own local shared memory. This new architecture avoids the memory traffic jam limitation and improves the performance of computer systems[2]. A Compiler for GPC produces SystemC Models in the new proposed GPC architecture for simulation purpose, however, the lack of benchmark models make the evaluation of this compiler's performance hard.

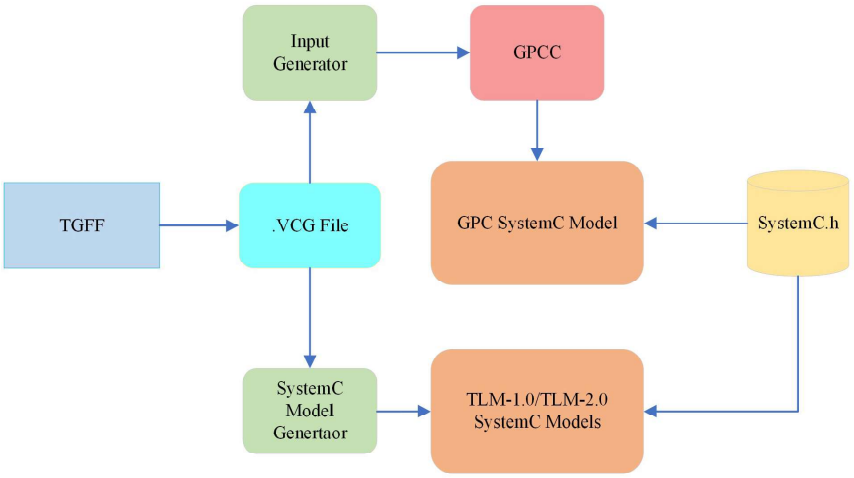


Figure 1: Design flow of taking TGFF's output to generate TLM models and GPC mapping graph

TGFF is a graph generator initially designed to generate task graphs with task sets and communications among tasks based on given input parameters[1]. This report will introduce tools taking a data flow graph from the TGFF graph generator as an input then generates synthetic SystemC benchmark models, input for the GPC compiler. The generated models can be utilized to evaluate the performance of the GPC compiler by comparing their simulation results with those of models produced by the compiler.

2 Task Graphs from TGFF

2.1 Task Graph's Meaning and Usage

TGFF is a tool that generates task graphs based on input ".tgffopt" option file. The program produces both text-based and visualized task graphs. By changing the input parameters in the ".tgffopt" file, task graphs with various number of tasks and connection features can be generated, so the SystemC TLM models produced based on these graphs will be sufficient to serve as benchmark models for statistical evaluations.

Figure 2 shows a sample task graph produced by the TGFF graph generator. The nodes represent the tasks, and the edges represent the data flow among tasks. To transform this graph into SystemC TLM models, each node will be converted to a SystemC node module, and the modules will be connected based on the edges in the graph. Since this visualized graph is stored in a readable ".vcg" text file, the SystemC TLM module generator will read the ".vcg" file and store the information about the nodes and edges to a graph data structure, so it can be used to produce a C++ file representing the SystemC TLM module.

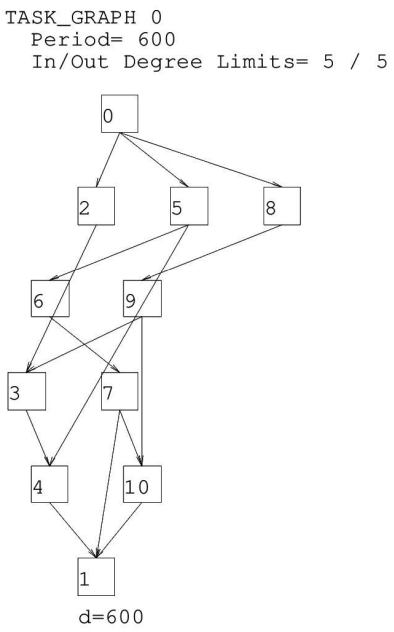


Figure 2: Sample task graph by TGFF

```
node: { title: "t0_8" label: "t0_8 (6)"  color: white }
node: { title: "t0_9" label: "t0_9 (15)"  color: white }
node: { title: "t0_10" label: "t0_10 (11)"  color: white }

edge: { thickness: 2 sourcename:"t0_0" targetname: "t0_2"  label: "(41)" }

edge: { thickness: 2 sourcename:"t0_2" targetname: "t0_3"  label: "(13)" }

edge: { thickness: 2 sourcename:"t0_3" targetname: "t0_4"  label: "(0)" }

edge: { thickness: 2 sourcename:"t0_4" targetname: "t0_1"  label: "(16)" }

edge: { thickness: 2 sourcename:"t0_0" targetname: "t0_5"  label: "(13)" }
```

Figure 3: Sample task graph description

2.2 Adjusting TGFF Graph Configuration for SystemC TLM Modeling

As previously mentioned, features of task graphs can be changed by adjusting the input parameters in the ".tgffopt" file. The generated SystemC module will serve as a Design under Test(DUT) unit that will be placed under a testbench to perform simulations, so the generated SystemC TLM model must have a proper structure compatible with the testbench. Thus, the input parameters must be set correctly to ensure that TGFF generates proper graphs representing the desired SystemC TLM

models. Figure 4 shows the input parameters for TGFF used to generate the task graph shown in figure 2 along with comments explaining the function of each input parameters.

Since the DUT will have one module accepting the input and one module sending the output to the platform inside the testbench, the graph must start with one node and end with only one node. This is achieved by setting the parameters "series_must_rejoin" and "gen_series_parallel" to 1 and not setting "prob_multi_start_nodes" parameter. These two parameters must be set correctly since the SystemC TLM model generator assumes that there will always be only one starting and one ending node in the graph. Otherwise, the model generator will not produce any meaningful output. The remaining parameters shown in Figure 4 set the graph's organization. Not all parameters are set in this ".tgffopt" file, detailed documentation of parameters can be found in the TGFF's manual[3].

```
#integrate tgff option test
#example of parallel series task graph
#the graph will have one starting node and join at one ending node

#Basic functions
tg_cnt 1
task_cnt 5 2
seed 3

#Determine the transmition type number (tested, not impact the actual graph, may be more details in the paper)
#trans_type_cnt 3

#can set input/output arcs, task type. (may not necessary using series algorithms)

#generate parallel series graph
gen_series_parallel 1

#Force the graph to join at the end node
series_must_rejoin 1

#How long is the chain
series_len 3, 1

#How many chains
series_wid 4, 2

# This allows generation of arcs which cross over between parallel chains
# Local xover is for within one set of series parallel tasks (means can jump over)
# Global xover allows arcs from one set of series parallel tasks to another(from one chain to other chain)
series_local_xover 2
series_global_xover 1

#output graphs
tg_write
eps_write
vcg_write
```

Figure 4: .tgffopt TGFF input file for task graph

3 Generation of SystemC TLM Models and Inputs for GPCC

3.1 Data Structure

Based on the flowchart of GPC compiler input mapper and SystemC TLM model's generation shown in figure 1, both the GPC compiler input mapper and the SystemC TLM model comes from the ".vcg" file. The first step is to extract the information of graphs from the ".vcg" file produced by TGFF and store it in appropriate data structure, so the information can be used by a C++ program. Figure 5 shows the data structure representing the graph read from the ".vcg" file.

The node's structure contains two pointers to lists of its input and output nodes, while the edge structure contains two pointers to its input and output nodes. The input node represents the source sending the data, and the output node represents the node receiving the data. Each node in the graph is stored in a node structure, similar to the edges. two main lists store all the nodes and all the edges correspondingly. Information is extracted from the ".vcg" file is achieved by using the regular expression function(regex) in standard C++ library to search for nodes and the connection between them, based on the ".vcg" file format. The advantages of using this data structure will be discussed in later sections about the detailed implementation of generators.

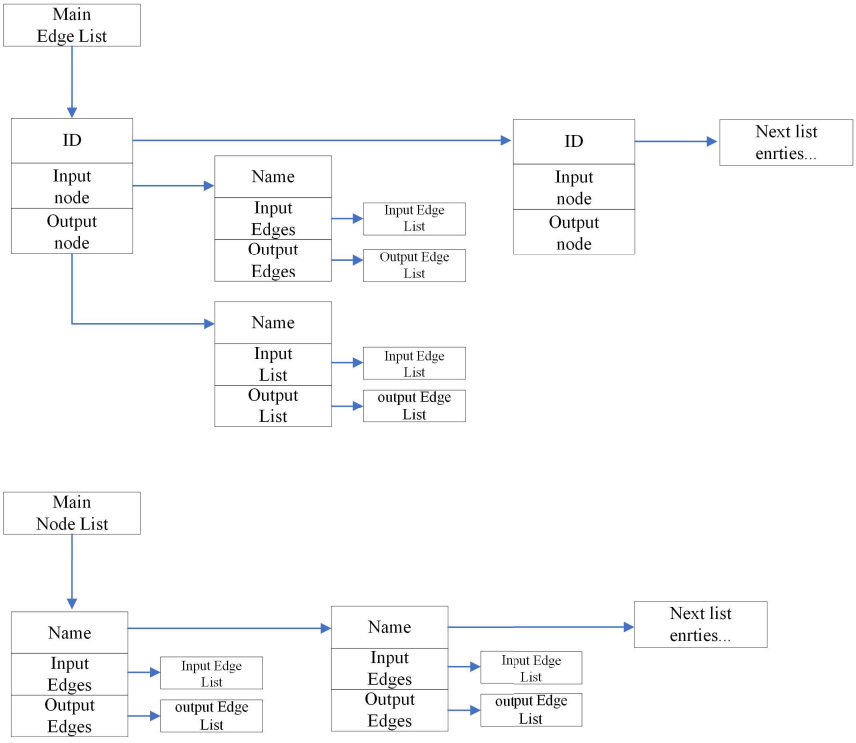
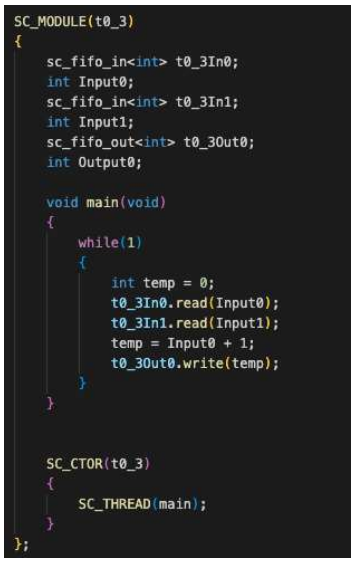


Figure 5: Data Structure to represent the graph

3.2 SystemC TLM-1.0 Model Generator

Transaction-Level Modeling 1.0(TLM-1.0)provides concepts for high-level transaction modeling. In TLM-1.0, modules communicate through channels and ports, with less focus on how transactions should be represented and timed[4]. SystemC TLM-1.0 models generated will have modules connecting through "sc_fifo" channels without specifying the time delay inside each module.



```
SC_MODULE(t0_3)
{
    sc_fifo_in<int> t0_3In0;
    int Input0;
    sc_fifo_in<int> t0_3In1;
    int Input1;
    sc_fifo_out<int> t0_3Out0;
    int Output0;

    void main(void)
    {
        while(1)
        {
            int temp = 0;
            t0_3In0.read(Input0);
            t0_3In1.read(Input1);
            temp = Input0 + 1;
            t0_3Out0.write(temp);
        }
    }


    SC_CTOR(t0_3)
    {
        SC_THREAD(main);
    }
};
```

Figure 6: Generated SystemC TLM-1.0 node module

```
t0_0_UNIT.input.bind(ImgIn);
t0_0_UNIT.t0_0Out0.bind(Edge_0);
t0_0_UNIT.t0_0Out1.bind(Edge_4);
t0_0_UNIT.t0_0Out2.bind(Edge_8);
t0_1_UNIT.t0_1In0.bind(Edge_3);
t0_1_UNIT.t0_1In1.bind(Edge_7);
t0_1_UNIT.t0_1In2.bind(Edge_11);
t0_1_UNIT.output.bind(ImgOut);
t0_2_UNIT.t0_2In0.bind(Edge_0);
t0_2_UNIT.t0_2Out0.bind(Edge_1);
t0_3_UNIT.t0_3In0.bind(Edge_1);
```

Figure 7: Generated SystemC TLM-1.0 module's connection

As previously stated, each node represents a module, and each edge represents a connection between modules. For the module to communicate in the TLM-1.0 model through channels, it should have corresponding ports connected to the channels and local variables to read and store the data received. Figure 6 shows a generated SystemC node module's content, indicating it has 2 input edges, 1 output edge, and 3 local variables for reading and sending the data. Since the node's data structure stores 2 lists of its input and output edges, the generation of all node modules can be done effectively by iterating through the main node list and then iterating through its input and output lists. Similarly, since the edge structure holds its input and output nodes, as shown in figure 7when generating the channels to connect all node modules, it will only require iterating through the main edge list.

The generated model will be placed under the testbench for simulation. Currently, inside each node module, the processing task simply increments the received data value by 1, and the initial data value sent by the stimulus is set to be 1, and the data value received by the monitor will reflect

the length of each chain connecting the nodes in the graph. By checking the data value received with the original graph, it can be determined whether the generated model is organized correctly.

```
Final node: t0_1
The final signal value 0 is 4:
The final signal value 1 is 4:
The final signal value 2 is 4:
      0 s: Monitor received frame 54 with      0 s delay.
      0 s: Monitor received frame 55 with      0 s delay.
      0 s: Monitor exits simulation.
```

Figure 8: Simulation result of generated SystemC TLM-1.0 model

3.3 SystemC TLM-2.0 Model Generator

TLM-2.0 models support a more detailed definition of transaction type and description of time[4]. In SystemC TLM-2.0 models, node modules no longer communicate simply through "sc_fifo" channels but share the information using local memory blocks, which is a more realistic and meaningful model for evaluating the GPC compiler's performance.

For node modules to communicate through memory blocks in the TLM-2.0 model, each edge will create a socket on the node module for reading or sending the data to the memory blocks. Since data cannot be read from and written to the memory at the same time, the memory block will be connected to a Mux that connects to the sockets on node modules, and selecting the data goes into or out from the memory block.

```
178
179  SC_MODULE(t0_2)
180  {
181      tlm_utils::simple_initiator_socket<t0_2> In0;
182      int Input0;
183      sc_event &sigIn0;
184      tlm_utils::simple_initiator_socket<t0_2> Out0;
185      int Output0;
186      sc_event &sigOut0;
187
188      void main(void)
189      {
190          HostedSM OutSM0(Out0, sigOut0, 0x00, 0x1000, ON_CHIP_MEMORY_SIZE);
191          MemQ<int,1> Out_Q0(OutSM0, offsetof(DUT_Channels, data));
192
193          wait(MEMQ_INITIALIZATION_DELAY);
194          RemoteSM  InSM0(In0, sigIn0, 0x00, ON_CHIP_MEMORY_SIZE);
195          MemQ<int, 1> In_Q0(InSM0, offsetof(DUT_Channels, data));
196
197          while(1)
198          {
199              int tmp = 0;
200              In_Q0.Pop(Input0);
201              tmp = Input0 + 1;
202              Out_Q0.Push(tmp);
203          }
204      }
205
206      SC_HAS_PROCESS(t0_2);
207
208      t0_2(sc_module_name n, sc_event &sigIn0, sc_event &sigOut0)
209  :sc_module(n)
210      ,sigIn0(sigIn0)
211      ,sigOut0(sigOut0)
212      {
213          SC_THREAD(main);
214          SET_STACK_SIZE
215      }
216  };
```

Figure 9: Generated SystemC TLM-2.0 node module

The graph representation file generated by TGFF does not include any other information beyond

the nodes and edges, the TLM-2.0 model is built by replacing each channel in the TLM-1.0 model with one socket on both the input and output nodes and a memory block stores the data for their communication and a 2-to-1 Mux selects whether data goes into or comes out of the memory block.

As shown in Figure 9, the memory block is hosted by the input node, so each node that passes data out will need a delay due to memory initialization, and the node will read the data by accessing the corresponding hosted memory block remotely. For the node to receive the correct data, it needs to have the same sc_event as the sender.

Similar to the TLM-1.0 model, generating the TLM-2.0 model requires requires first generating all the nodes and corresponding sockets, local sc_event, hosted or remote access memory, and local variables by iterating through the main node list. Because the mux, memory blocks, and sc_event all represent the connection between nodes, it can be generated properly by iterating through the edges list. Finally, a SystemC TLM-2.0 model can be built by connecting all components and assigning the sc_event signal properly.

Simulation using the generated model can be achieved by connecting it to a modified testbench that communicates with the model using memory blocks. same as the SystemC TLM-1.0 model, the processing function inside each node module simply increments the data value received by 1, so the meaning of the output data remains the same, representing the length of chains connecting the nodes. However, since the TLM-2.0 model requires the node to wait for memory initialization, there will be a time delay for the monitor to receive the output data.

```
Final node: t0_1
The final signal value 0 is 4:
The final signal value 1 is 4:
The final signal value 2 is 4:
2012260 ns: Monitor received frame 19 with  5625 ns delay.
2012896 ns: Monitor received frame 20 with  5625 ns delay.
2012896 ns: Monitor exits simulation.
```

Figure 10: Simulation results of generated SystemC TLM-2.0 model

3.4 Generation of GPC compiler's Input

The basic idea for evaluating the performance of the GPC compiler is to compare the performance of models generated by the GPC compiler with the performance of models produced by previous generators. GPC compiler takes a ".json" file that describes the connections among nodes as its input. Although directly generating a ".json" file from a C++ program is difficult, a Python file serves as a mapper used to produce the ".json" file that can be generated from the C++ program. The generation process is similar to previous model generators since the mapper only needs nodes and edges' information to produce a ".json" file for the GPC compiler's input.

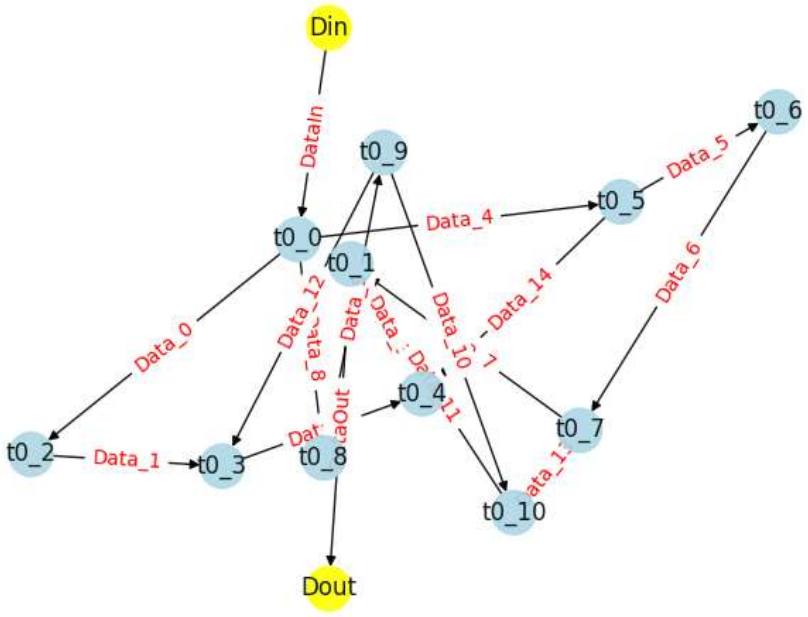


Figure 11: Graph produced by GPC compiler input mapper

At this time, the GPC compiler is not finalized yet, and can only produce output based on specific inputs, so there is no effective way to validate whether the generated ".json" file works properly for the GPC compiler. However, the mapper will also produce a graph showing the connections of nodes and edges. By checking the details in the graph produced by the mapper with the original graph produced by TGFF, the correctness of the mapper can be partly verified.

4 Experiments and Results

Benchmark	Nodes	Depth	LoC	TLM-1.0	LoC	TLM-2.0
Simple-1	8	5	311	0	547	2012871
Simple-2	10	10	361	0	619	2012846
Medium-1	40	10	1635	0	2895	2013519
Medium-2	49	11	1826	0	3136	2012996
Large-1	110	14	4319	0	7539	2026808
Large-2	93	13	3926	0	7008	2023472

Table 1: Generated benchmark model and simulation results

The table above shows the experiment result of generated TLM-1.0 and TLM-2.0 models with different complexity. Column "Nodes, Depth" and "LoC" shows the number of nodes, the length

of the longest data path, and the lines of code in generated file, respectively. Column "TLM-1.0" and "TLM-2.0 shows the total simulated time of corresponding models in nano seconds(ns). Since TLM-1.0 model focus less on transaction timing compare with TLM-2.0 model, and the generated TLM-1.0 model don't have any time definition. Data presents by the table shows that the simulated time for the TLM-1.0 model stays at 0 ns due to no timing definition in generated models, and for the TLM-2.0 model the overall simulated time increases with the increase in model complexity. Thus, these examples meet the expectation and can serve as valid benchmarks for performance evaluation. Commands in the .tgffopt files corresponding to above models are provided in the appendix.

5 Conclusion and Future Work

This report introduces tools developed based on the TGFF graph generator to generate synthetic examples and input the GPC compiler for the GPC compiler's performance evaluation. The report briefly discusses the background information about the GPC compiler, the motivation to design these generators, and the technical details of them.

As previously stated, for validation purposes, currently all generated models have node modules that only increment the received data value. In the future, more meaningful and realistic computation tasks can be assigned to these nodes to make the simulation results reflect the models' performance more comprehensively. For the TLM-2.0 SystemC model, since the TGFF only gives the information about nodes and edges in the graph, for now, each edge is replaced with one memory block. Further improvement to the TLM-2.0 model generator can be done by grouping the edges or nodes and assigning memory blocks based on groups, which will produce a more realistic model, and once the GPC compiler is finalized, all these improvements will offer a more accurate evaluation of the GPC compiler's performance.

References

[1] R.P. Dick, D.L. Rhodes, and W. Wolf. Tgff: Task graphs for free. In *Proceedings of the Sixth International Workshop on Hardware/Software Codesign. (CODES/CASHE'98)*, March 1998.

[2] Rainer Dömer. A grid of processing cells (gpc) with local memories. Technical Report CECS-22-01, April 2022.

[3] Keith Vallerio. *Task Graphs for Free (TGFF v3.0)*, September 2017.

[4] Ieee standard for standard systemc® language reference manual. *IEEE Std 1666-2023 (Revision of IEEE Std 1666-2011)*, pages 1–618, 2023.

6 Appendix

```
# Large-2
tg_cnt 1
task_cnt 36 6
seed 4
gen_series_parallel 1
series_must_rejoin 1
series_wid 15, 3
series_local_xover 15
series_global_xover 15
tg_write
eps_write
vcg_write

# Large-1
#Basic functions
tg_cnt 1
task_cnt 36 6
seed 1
gen_series_parallel 1
series_must_rejoin 1
series_wid 15, 3
series_local_xover 15
series_global_xover 15
tg_write
eps_write
vcg_write

# Medium-2
#Basic functions
tg_cnt 1
task_cnt 12 4
seed 4
gen_series_parallel 1
series_must_rejoin 1
series_wid 8, 2
series_local_xover 6
series_global_xover 6
tg_write
eps_write
vcg_write
```

```
# Medium-1
tg_cnt 1
task_cnt 12 4
seed 1
gen_series_parallel 1
series_must_rejoin 1
series_wid 8, 2
series_local_xover 6
series_global_xover 6
tg_write
eps_write
vcg_write

# Simple-2
#Basic functions
tg_cnt 1
task_cnt 3 1
seed 4
gen_series_parallel 1
series_must_rejoin 1
tg_write
eps_write
vcg_write

# Simple-1
tg_cnt 1
task_cnt 3 1
seed 1
gen_series_parallel 1
series_must_rejoin 1
tg_write
eps_write
vcg_write
```