Simulating real-time scheduling in multi-processor systems

Matthijs van Drunen

1 Introduction

Embedded systems are computer systems that are embedded within a larger system. In some embedded systems, time is critical to the proper functioning of the system. In this case, we call such systems *real-time*.

CSDF graphs (Bilsen et al. 1996) can be used to model streaming programs. The model consists of actors (i.e. tasks) that are connected using channels. On each invocation, every actor reads tokens from its connected input channels and writes tokens to its connected output channels. An example CSDF graph is given in Figure 1. In this graph, actors are represented nodes and channels by edges. The sequences on the input and output ports of an actor represent the number of tokens that an actor wil consume (in case of an input port) and produce (in case of an output port). For example, actor A_1 in Figure 1 will produce one token on channels E_1 and E_3 and no tokens on E_2 on its first and second invocation. On its third invocation, it will produce one token on E_3 and E_2, but none on E_1. After this, the production and consumption sequences will repeat. Actor A_2 will consume one token from E_1 and produce one token on E_4 on every invocation.

It is possible to execute a program modeled as an acyclic CSDF graph using a set of real-time periodic tasks (Bamakhrama 2014). In this report, we propose an infrastructure to simulate the behavior of Cyclo-Static Dataflow (CSDF) graphs using real-time scheduling algorithms. Earliest Deadline First (EDF), proposed by Liu and Layland 1973, is one of such algorithms. At each point in time, it will schedule the task with the nearest deadline.

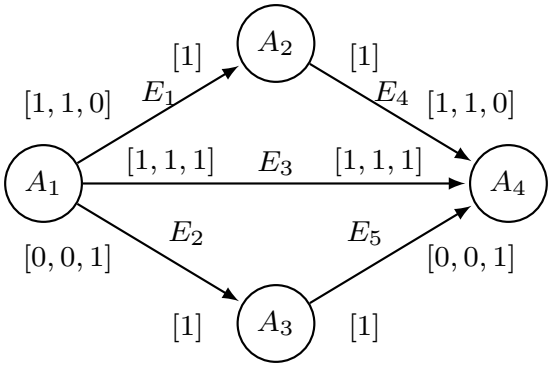


Figure 1: An example of a CSDF graph (Bamakhrama 2014).

2 Definitions

In this section, we will introduce the terminology that is used throughout this report. A *system* consists of m *processors* $\{\pi_1, \pi_2, ..., \pi_m\}$. These processors can execute tasks from a set of tasks

$T = \{T_1, T_2, ..., T_n\}$. We assume that all tasks are modelled using the *real-time periodic* task model. Each *task* is defined as a tuple $T_i = (S_i, C_i, P_i, D_i)$, where S_i is the start time of task T_i, C_i its worst-case execution time, P_i its period and D_i its deadline. Each periodic task will be invoked at a constant time interval. Task T_i will be invoked at times $r_{i,k} = S_i + kP_i$ where $k \in \mathbb{N}$. If task T_i is invoked at time instant $r_{i,k}$, it should finish execution before $d_{i,k} = r_{i,k} + D_i$. A *scheduler* is a function $\sigma(t) = (\sigma_1(t), \sigma_2(t), ..., \sigma_m(t))$, where each $\sigma_i(t)$ indicates the task running on processor i:

$$\sigma_i(t) = \begin{cases} 0 & \text{if } \pi_i \text{ is idle at time instant } t. \\ j & \text{if } \pi_i \text{ is running task } T_j \text{ at time instant } t. \end{cases}$$

A *global* scheduling algorithm allows all tasks to be migrated among all processors during execution, whereas a *partitioned* scheduling algorithm does not permit any migration at all. A *hybrid* scheduling algorithm allows a subset of tasks to migrate among a subset of processors.

3 Simulator architecture

The simulation of tasks is implemented using SystemC (Panda 2001) modules. The communication channels between tasks are FIFOs with a limited capacity. If a write is performed when a FIFO is full, or a read when it is empty, the simulation will be stopped (blocking is not allowed when tasks are sceduled according to the framework in [Bamakhrama 2014]). All components are connected to a single clock signal. The FIFOs only allow one read or write per clock cycle. Processes are also implemented as SystemC modules, as subclasses of `sc_schedulable_module`. An example is shown in Listing 1. These modules are connected to an instance of `sc_scheduler`. Instead of calling SystemC's `wait()` function, processes should call `wait_ticks(n)` where `n` is the number of clock cycles of the simulated work. This will cause the process to wait for a signal from the scheduler to run for `n` clock cycles. Pseudo code for `wait_ticks(n)` is shown in Algorithm 1

Figure 2 shows the relation between the different components of the simulator. Only the relevant attributes and methods are shown. The most important components will be discussed in the following subsections.

```
class Pf2 : public sc_schedulable_module, public Process {
public:
    sc_in<bool> clk;
    sc_fifo_in<int> IP1;
    sc_fifo_out<int> OP1;

    void run() {
        while (true) {
            wait_ticks(read_delay);
            IP1.read();
            wait_ticks(exec_delay);
            wait_ticks(write_delay);
            OP1.write(1);
        }
    }
    ...
};
```

Listing 1: Part of the implementation of a SystemC module for actor A_3 in the CSDF graph shown in Figure 1.

Algorithm 1 Pseudo code for the `wait_ticks` function.

```
1: function WAIT_TICKS(n)
2:     r ← n
3:     while r ≠ 0 do
4:         WAIT(run_event)          ▷ run_event is implemented using SystemC's sc_event
5:         r ← r − 1
6:     end while
7: end function
```

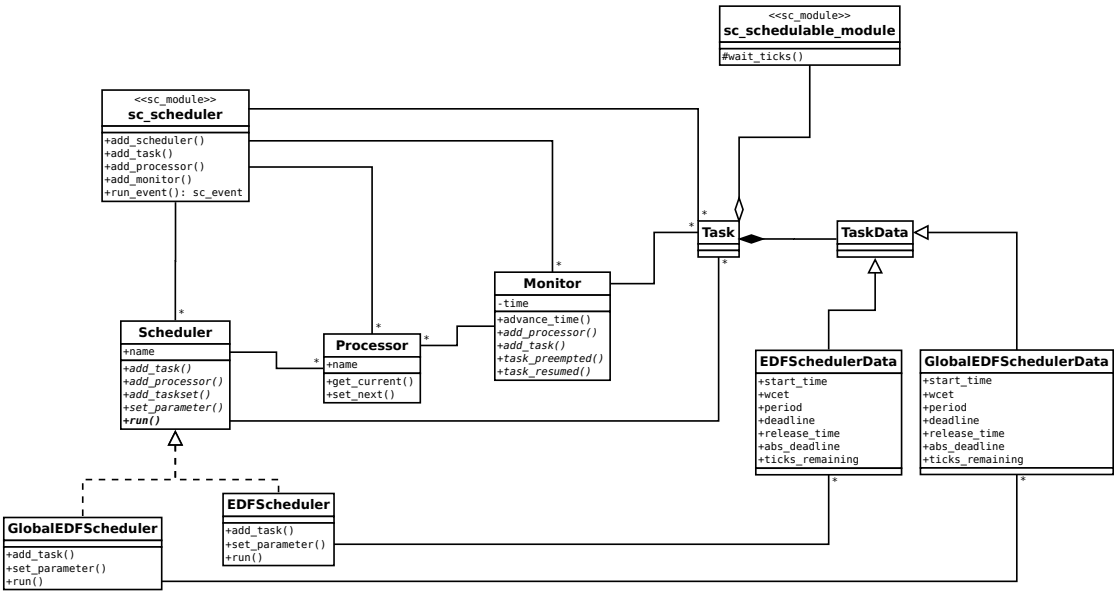


Figure 2: A UML class diagram showing the relations between the different components of the simulator.

3.1 Schedulers

The `sc_scheduler` component calls upon implementations of subclasses of `Scheduler` to perform the actual scheduling. A scheduler's `run()` function is called every clock tick. This function should assign to each processor the task that it will run for that clock cycle using the `set_next()` function. A processor can also be idle for a cycle, which is achieved by passing `NULL` as argument to `set_next()`.

Every subclass of `Scheduler` can implement the `add_processor`, `add_task` and `set_parameter` methods defined in Table 1. The `run` method must always be implemented.

Function	Description
`add_processor`	A processor is added to the simulation.
`add_task`	A task is added to the simulation.
`set_parameter`	A parameter (e.g. wcet, deadline) is set for a task.
`run`	This function gets called every clock cycle. The scheduler should assign tasks to each processor in this function.

Table 1: Functions that can be implemented by schedulers.

3.2 Monitors

Subclasses of the `Monitor` class are used to monitor the simulation. Such classes can be used for instance to output log files or generate statistics of the simulation. Monitor objects respond to events that are generated during the simulation. These events are described in Table 2. Initially, all tasks are assumed to be not running. The function `get_time()` can be used to obtain the cycle in which an event has occured.

Event	Description
`add_processor`	A processor is added to the simulation.
`add_task`	A task is added to the simulation.
`task_preempted`	A task is being preempted.
`task_resumed`	A task is resuming execution.

Table 2: Different events that are generated during simulation.

Currently, two monitors that are implemented are `GraspMonitor` and `StatsMonitor`. `GraspMonitor` will output a trace of the simulation to a file that follows the GRASP format. This file can be visualized using the GRASP tool (Holenderski et al. 2010). `StatsMonitor` outputs statistics (e.g. total execution time, processor utilization) to the standard output.

3.3 System specification

The schedulers, tasks, processors, FIFO sizes and mappings are specified in an XML file. The simulator uses information from this file to construct a model of the system. All possible parameters are described in Table 3. The FIFO sizes can be left out, in which case the default value (16) is assumed. An example of a system specification in XML format is shown in Listing 2. The tasks `Psrc`, `Pf1`, `Pf2` and `Psnk` refer to actors A_1, A_2, A_3 and A_4 in Figure 1, respectively.

Type	Parameter	Description
`scheduler`	`name`	The name of the scheduler.
	`algorithm`	The scheduling algorithm. Currently, the only allowed value is `EDF`.
	`type`	The type of scheduling algorithm. Valid options are `global`, `partitioned` and `hybrid-semipartitioned`.
	`mapping`	The name of the mapping that is to be used by this scheduler. The mapping must be defined in the XML file.
`task`	`name`	The name of the task.
	`wcet`	The WCET of the task. The total WCET (that is passed to the scheduler) is computed as the value of `wcet` + `readDelay` + `writeDelay`.
	`readDelay`	The number of clock cycles it takes to complete all FIFO reads.
	`writeDelay`	The number of clock cycles it takes to complete all FIFO writes.
	`startTime`	The start time (in clock cycles) of the task.
	`period`	The period (in clock cycles) of the task.
	`deadline`	The deadline (relative to the start time) of the task.
	`priority`	The priority of the task.
	`type`	The type of the task. Valid options are `fixed` and `migrating`.
`processor`	`name`	The name of the processor.
	`scheduler`	The name of a scheduler for this processor. This should refer to a scheduler defined in the XML file.
`fifo`	`name`	The name of the FIFO. This should match the name of the `fifo_fsl` SystemC component.
	`size`	The maximum number of tokens that can be stored in the FIFO.

Table 3: A description of the parameters for the different components that are specified in the XML file.

4 Results

In order to demonstrate the simulator, it was run on a number of different system specifications. Figure 3 and Table 4 show the tasks that are running on each processor for the 4-processor system specified in Listing 2, where the tasks are scheduled using partitioned EDF. The results for the 2-processor system (also using partitioned EDF) specified in Listing 3 are shown in Figure 4 and Table 5.

In order to demonstrate the migrations of tasks, global EDF was used to schedule tasks on a 4-processor system. The results are shown in Figure 5 and Table 6.

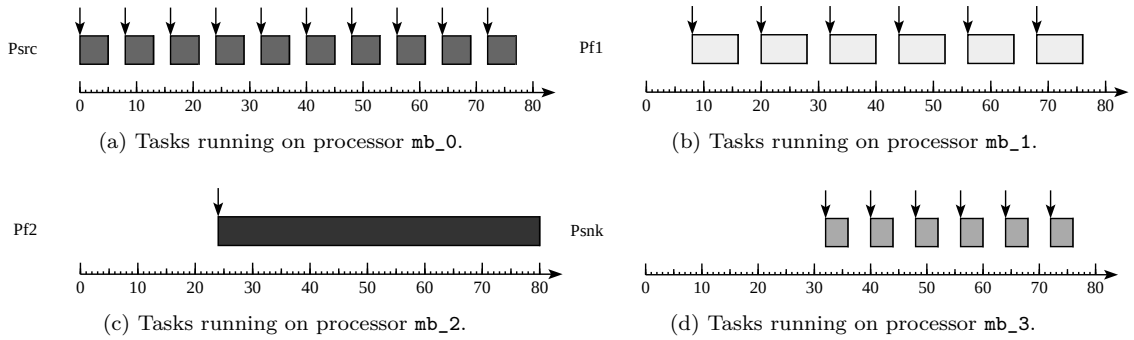
Task	Execution time (ticks)	Migrations
`Psrc`	50	0
`Pf1`	48	0
`Pf2`	55	0
`Psnk`	24	0
Total	177	0

(a) Task statistics.

Processor	Utilization
`mb_0`	0.625
`mb_1`	0.6
`mb_2`	0.6875
`mb_3`	0.3
Total	2.2125

(b) Processor statistics.

Table 4: Statistics for the simulation of the system specified in Listing 2.



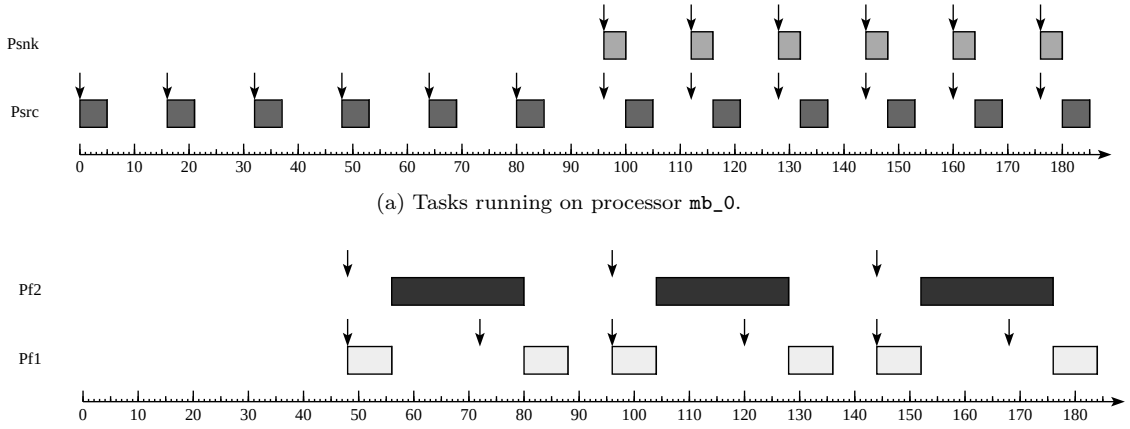
(a) Tasks running on processor `mb_0`.

(b) Tasks running on processor `mb_1`.

(c) Tasks running on processor `mb_2`.

(d) Tasks running on processor `mb_3`.

Figure 3: GRASP plots showing the schedule for the simulation of the system specified in Listing 2. The arrows denote the release times.



(a) Tasks running on processor `mb_0`.

(b) Tasks running on processor `mb_1`.

Figure 4: GRASP plots showing the schedule for the simulation of the system specified in Listing 3.

Task	**Execution time (ticks)**	**Migrations**
`Psrc`	60	0
`Pf1`	48	0
`Pf2`	72	0
`Psnk`	24	0
Total	204	0

(a) Task statistics.

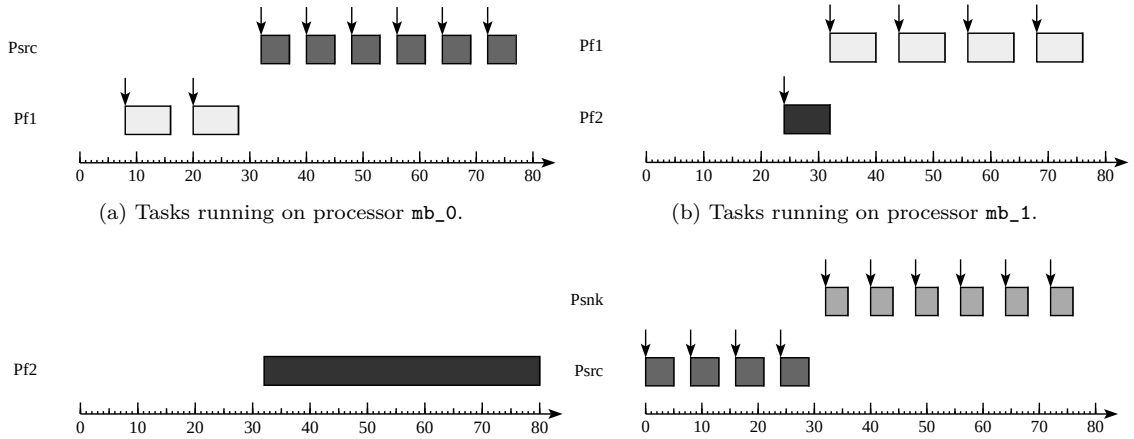
Processor	**Utilization**
`mb_0`	0.4375
`mb_1`	0.625
Total	1.0625

(b) Processor statistics.

Table 5: Statistics for the simulation of the system specified in Listing 3.

```xml
<?xml version="1.0" standalone="no" ?>
<system name="mySystem">
    <scheduler name="sched_0" algorithm="EDF" type="partitioned" mapping="mapping_0" />
    <scheduler name="sched_1" algorithm="EDF" type="partitioned" mapping="mapping_1" />
    <scheduler name="sched_2" algorithm="EDF" type="partitioned" mapping="mapping_2" />
    <scheduler name="sched_3" algorithm="EDF" type="partitioned" mapping="mapping_3" />

    <task name="Psrc" wcet="3" readDelay="0" writeDelay="2" startTime="0"
            period="8" deadline="8" priority="1" type="fixed" />
    <task name="Pf1" wcet="4" readDelay="2" writeDelay="2" startTime="8"
            period="12" deadline="12" priority="2" type="fixed" />
    <task name="Pf2" wcet="20" readDelay="2" writeDelay="2" startTime="24"
            period="24" deadline="24" priority="3" type="fixed" />
    <task name="Psnk" wcet="2" readDelay="2" writeDelay="0" startTime="32"
            period="8" deadline="8"  priority="4" type="fixed" />

    <processor name="mb_0" scheduler="sched_0" />
    <processor name="mb_1" scheduler="sched_1" />
    <processor name="mb_2" scheduler="sched_2" />
    <processor name="mb_3" scheduler="sched_3" />

    <fifo name="E1" size="2" />
    <fifo name="E2" size="2" />
    <fifo name="E3" size="5" />
    <fifo name="E4" size="3" />
    <fifo name="E5" size="2" />

    <mapping name="mapping_0">
       <processor name="mb_0">
           <task name="Psrc" />
       </processor>
    </mapping>
    <mapping name="mapping_1">
       <processor name="mb_1">
           <task name="Pf1" />
       </processor>
    </mapping>
    <mapping name="mapping_2">
       <processor name="mb_2">
           <task name="Pf2" />
       </processor>
    </mapping>
    <mapping name="mapping_3">
       <processor name="mb_3">
           <task name="Psnk" />
       </processor>
    </mapping>
</system>
```

Listing 2: Example of a system specification for the example in Figure 1 on 4 processors using partitioned EDF.



(a) Tasks running on processor `mb_0`.

(b) Tasks running on processor `mb_1`.

(c) Tasks running on processor `mb_2`.

(d) Tasks running on processor `mb_3`.

Figure 5: GRASP plots showing the schedule for the simulation of the system specified in Listing 4.

Task	**Execution time (ticks)**	**Migrations**
`Psrc`	50	1
`Pf1`	48	1
`Pf2`	56	1
`Psnk`	24	0
Total	178	3

(a) Task statistics.

Processor	**Utilization**
`mb_0`	0.575
`mb_1`	0.5
`mb_2`	0.6
`mb_3`	0.55
Total	2.225

(b) Processor statistics.

Table 6: Statistics for the simulation of the system specified in Listing 4.

```xml
<?xml version="1.0" standalone="no" ?>
<system name="mySystem">
    <scheduler name="sched_0" algorithm="EDF" type="partitioned" mapping="mapping_0" />
    <scheduler name="sched_1" algorithm="EDF" type="partitioned" mapping="mapping_1" />

    <task name="Psrc" wcet="3" readDelay="0" writeDelay="2" startTime="0"
            period="16" deadline="16"  priority="1" type="fixed" />
    <task name="Pf1" wcet="4" readDelay="2" writeDelay="2" startTime="48"
            period="24" deadline="24" priority="2" type="fixed" />
    <task name="Pf2"  wcet="20" readDelay="2" writeDelay="2" startTime="48"
            period="48" deadline="48" priority="3" type="fixed" />
    <task name="Psnk" wcet="2" readDelay="2" writeDelay="0" startTime="96"
            period="16" deadline="16" priority="4" type="fixed" />

    <processor name="mb_0" scheduler="sched_0" />
    <processor name="mb_1" scheduler="sched_1" />

    <!-- Fifo sizes are assumed to be 16 (default value) -->

    <mapping name="mapping_0">
       <processor name="mb_0">
           <task name="Psrc" />
           <task name="Psnk" />
       </processor>
    </mapping>
    <mapping name="mapping_1">
       <processor name="mb_1">
           <task name="Pf1" />
           <task name="Pf2" />
       </processor>
    </mapping>
</system>
```

Listing 3: A system specification for the CSDF graph in Figure 1 using partitioned EDF on 2 processors.

```xml
<?xml version="1.0" standalone="no" ?>
<system name="mySystem">
    <scheduler name="sched_0" algorithm="EDF" type="global" mapping="mapping_0" />

    <task name="Psrc" wcet="3" readDelay="0" writeDelay="2" startTime="0"
            period="8" deadline="8" priority="1" type="migrating" />
    <task name="Pf1"  wcet="4" readDelay="2" writeDelay="2" startTime="8"
            period="12" deadline="12" priority="2" type="migrating" />
    <task name="Pf2"  wcet="20" readDelay="2" writeDelay="2" startTime="24"
            period="24" deadline="24" priority="3" type="migrating" />
    <task name="Psnk" wcet="2" readDelay="2" writeDelay="0" startTime="32"
            period="8" deadline="8" priority="4" type="migrating" />

    <processor name="mb_0" scheduler="sched_0" />
    <processor name="mb_1" scheduler="sched_0" />
    <processor name="mb_2" scheduler="sched_0" />
    <processor name="mb_3" scheduler="sched_0" />

    <fifo name="E1" size="2" />
    <fifo name="E2" size="2" />
    <fifo name="E3" size="5" />
    <fifo name="E4" size="3" />
    <fifo name="E5" size="2" />

    <mapping name="mapping_0">
       <processor name="mb_0">
           <task name="Psrc" />
           <task name="Pf1" />
           <task name="Pf2" />
           <task name="Psnk" />
       </processor>
       <processor name="mb_1">
           <task name="Psrc" />
           <task name="Pf1" />
           <task name="Pf2" />
           <task name="Psnk" />
       </processor>
       <processor name="mb_2">
           <task name="Psrc" />
           <task name="Pf1" />
           <task name="Pf2" />
           <task name="Psnk" />
       </processor>
       <processor name="mb_3">
           <task name="Psrc" />
           <task name="Pf1" />
           <task name="Pf2" />
           <task name="Psnk" />
       </processor>
    </mapping>
</system>
```

Listing 4: A system specification for the CSDF graph in Figure 1 using global EDF on 4 processors.

References

Bamakhrama, Mohamed A. (2014). "On Hard Real-Time Scheduling of Cyclo-Static Dataflow and its Application in System-Level Design". PhD thesis. Universiteit Leiden, Leiden, Netherlands.

Bilsen, Greet et al. (1996). "Cycle-static dataflow". In: *Signal Processing, IEEE Transactions on* 44.2, pp. 397–408.

Holenderski, Mike et al. (2010). "Grasp: Tracing, visualizing and measuring the behavior of real-time systems". In: *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, pp. 37–42.

Liu, C. L. and James W. Layland (1973). "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment". In: *J. ACM* 20.1, pp. 46–61. ISSN: 0004-5411. DOI: 10.1145/321738.321743. URL: http://doi.acm.org/10.1145/321738.321743.

Panda, Preeti Ranjan (2001). "SystemC-a modeling platform supporting multiple design abstractions". In: *System Synthesis, 2001. Proceedings. The 14th International Symposium on*. IEEE, pp. 75–80.

A Getting Started

The source code for the simulation is stored in a git repository. The following dependencies are required to build the project.

- cmake 2.8 (`http://cmake.org/`)
- libxml2 (`http://xmlsoft.org/`)
- SystemC 2.3.0 (`http://www.accellera.org/downloads/standards/systemc`)

The following sequence of commands can be used to download the source code and compile the project on a Linux system:

```
git clone https://github.com/mdartu/csdfrtsim.git
cd csdfrtsim
mkdir build
cd build
cmake ../src/
# if cmake is unable to find the systemc library,
# use the following command to specify the location:
# cmake -DSYSTEMC_INCLUDE_DIR=/path/to/systemc/include \
#       -DSYSTEMC_LIB_DIR=/path/to/systemc/lib ../src/
make
```

In order to test the simulator, it may be run using one of the supplied system specification files:

```
./test.bin ../examples/test_global.xml
```