A Complete SystemC Process Instrumentation Interface and Its Application to Simulation Performance Analysis

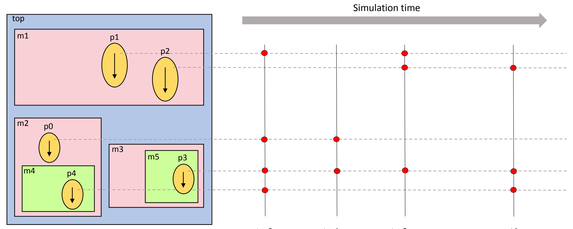
Bishnupriya Bhattacharya, Chandra Sekhar Katuri, Vincent Motel
Cadence Design Systems

Abstract

In the Electronic System Level domain, the SystemC language uses non-preemptive processes to simulate hardware parallelism in C++. Processes are a fundamental construct of the language, which makes modeling of hardware easier than in plain C++. Since processes are a key design element of SystemC, there are various application domains (e.g. debugging, performance analysis) where a generic mechanism to associate user-defined callbacks with process state changes will prove to be beneficial. We propose such a generic interface for registering user-defined callbacks triggered at SystemC process state changes. We also present an application case study for such an interface for simulation performance profiling in a virtual platform and in an approximately-timed memory controller model.

SystemC processes

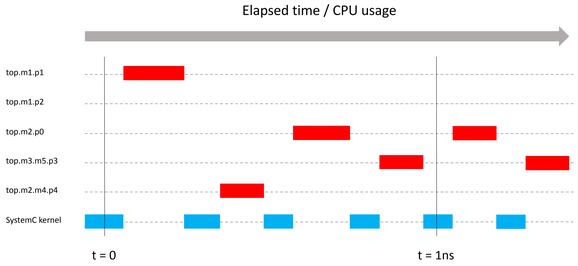
- Non-preemptive processes simulate hardware parallelism



Hierarchical view of the simulation

⇒ This models the parallel nature of electronic systems

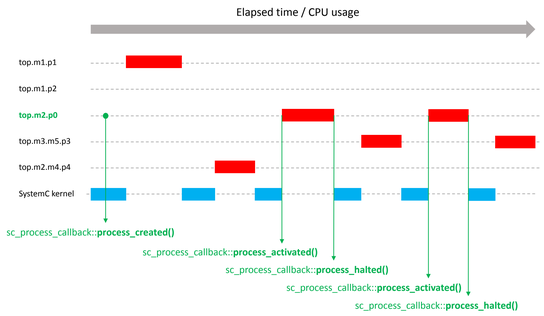
- From the start of the simulation to the end of the simulation, all executable code is initiated from one or more processes
- But the simulation execution is sequential



- Process execution follows a co-routine semantics: processes voluntarily yield control, the SystemC scheduler never preempts a process forcefully
- Two kinds of processes
  - SC_METHOD: executes completely from beginning to end in each execution, and does not maintain any local state
  - SC_THREAD: maintains its own local state, and on execution can voluntarily suspend itself using the wait() construct
- Static processes are created in the initial phases
- Dynamic processes are created during the simulation

SystemC Process Instrumentation Interface

We propose a SystemC language extension to register user-defined callbacks triggered at process state changes.

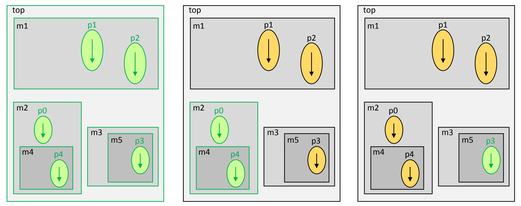


Class sc_process_callback

sc_process_callback
virtual void process_created(sc_process_handle process) virtual void process_activated(sc_process_handle process) virtual void process_halted(sc_process_handle process) virtual void process_terminated(sc_process_handle process) virtual void process_control_construct_invoked(sc_process_handle, process_control_construct_t) virtual void reset_signal_changed(sc_process_handle, const char*, reset_active_t)

users_callback_class

Callback registration



All processes of the design

bool **sc_add_process_callback_all**(
*sc_process_callback *callback*
);
bool **sc_remove_process_callback_all**(
*sc_process_callback *callback*
);

All processes in a module

bool **sc_add_process_callback_module**(
sc_module mod, sc_process_callback *callback*
);
bool **sc_remove_process_callback_module**(
sc_module mod, sc_process_callback *callback*
);

A specific process

bool **sc_add_process_callback**(
sc_process_handle process,
*sc_process_callback *callback,*
sc_descendant_inclusion_info = SC_NO_DESCENDANTS
);
bool **sc_remove_process_callback**(
sc_process_handle process,
*sc_process_callback *callback,*
sc_descendant_inclusion_info = SC_NO_DESCENDANTS
);

+ one method for dynamic processes

*bool sc_spawn_options::***add_process_callback**(*sc_process_callback *);*

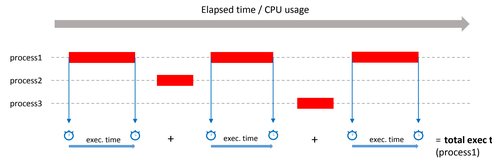
Implementation of the process instrumentation interface

- A subset of the process callback interface has been implemented in the Cadence Incisive® simulator
  - In the 15.10 release
  - Some constructs for advanced use models are not implemented yet: *process_control_construct_invoked()* and *reset_signal_changed()*, *sc_spawn_options::add_process_callback()*, callback registration for individual process handle
  - No performance degradation when process callbacks are not registered
- The natural next step is to propose the specification of this interface to Accellera System Initiative for future language standardization

  ⇒ It would allow to make models and applications relying on the callbacks compatible with other simulators.

Application to Simulation Performance Analysis

The process instrumentation interface can be used to make accurate simulation performance analysis, by measuring the exact CPU time spent in each process execution.

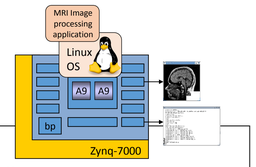



```
#include <time.h>
timespec exec_start_ts;
timespec exec_end_ts;

In process_activated() callback:
    clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &exec_start_ts);

In process_halted(), process_terminated() callbacks:
    clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &exec_end_ts);
    double exec_time = 1e-9 * (exec_end_ts.tv_nsec - exec_start_ts.tv_nsec) + (exec_end_ts.tv_sec - exec_start_ts.tv_sec);
```

The CPU time usage can be accumulated in simple per-process statistical bins, so that at the end of the simulation, the total CPU time usage of each process instance can be reported. In combination with process activation counts, the average CPU time of the activation of each process can be computed directly.

We applied this performance analysis application to two designs.

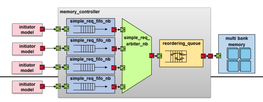
Profiling the virtual platform of Xilinx Zynq®-7000 "All Programmable SoC", running MRI image processing in hardware and software




```
-- summary
  total CPU usage          =    279.423191s
  total non-process time   =      5.329797s
  total process time       =    274.093394s
  number of process activations = 1516532, average 180.736967 us/activation
--list of contributions
    1.90743 %     5.329797 s                    3.514 us/activation - total non-process time (kernel, profiling itself)
    -- processes sorted by total exec time --
   95.26493 %   266.192300 s   517404 activations    514.477 us/activation - zynq.zynq_ps.armfm_core.ARMCortexA9MPx2CT32.run
    1.99865 %     5.586694 s    12000 activations    465.391 us/activation - zynq.ui.run
    0.28222 %     0.788582 s   859786 activations      0.917 us/activation - zynq.zynq_ps.usb1.start_schedule_th
    0.20792 %     0.580972 s     1281 activations    453.530 us/activation - zynq.bp.run
    0.11588 %     0.323805 s        2 activations 161902.519 us/activation - zynq.term_1.xterm_thread
    0.07198 %     0.201141 s    68698 activations      2.928 us/activation - zynq.term_0.xterm_thread
    0.05826 %     0.162791 s        1 activation  162790.992 us/activation - zynq.ddr_ram.reset_process
    0.03854 %     0.107692 s        3 activations  35897.204 us/activation - zynq.loader.reset_thread
    0.02024 %     0.056542 s    14120 activations      4.004 us/activation - zynq.term_0.rx_fifo_method
    0.00714 %     0.019961 s     2400 activations      8.317 us/activation - zynq.fb_ram.run
    0.00567 %     0.015841 s    17395 activations      0.911 us/activation - zynq.zynq_ps.armfm_core.run_standbywfi0
    0.00531 %     0.014839 s     4819 activations      3.079 us/activation - zynq.term_1.rx_fifo_method
    0.00349 %     0.009750 s     1112 activations      8.768 us/activation - zynq.zynq_ps.slirp_wrapper.poll_slirp
    0.00316 %     0.008818 s    11414 activations      0.773 us/activation - zynq.zynq_ps.armfm_core.run_standbywfi1
    0.00163 %     0.004544 s     1133 activations      4.011 us/activation - zynq.zynq_ps.armfm_core.bridge_ints27.signal
    0.00141 %     0.003935 s        1 activation    3934.544 us/activation - zynq.smc_sram_1.reset_process
    0.00140 %     0.003910 s        1 activation    3910.188 us/activation - zynq.smc_sram_0.reset_process
    0.00082 %     0.002304 s      595 activations      3.872 us/activation - zynq.zynq_ps.armfm_core.bridge_ints44.signal
    0.00078 %     0.002178 s        1 activation    2177.840 us/activation - zynq.logiCVCML.run
    0.00046 %     0.001298 s     1133 activations      1.146 us/activation - zynq.zynq_ps.armfm_core__ints27_s2w_convert
```

Performance report on the Zynq Virtual platform (top 20 processes).
Process corresponding to the processor model is highlighted in blue, it largely dominates the activity.
Process corresponding to the hardware accelerator is in green, its impact on simulation performance is very low.

Profiling the TLM model (approximately-timed) of a memory controller

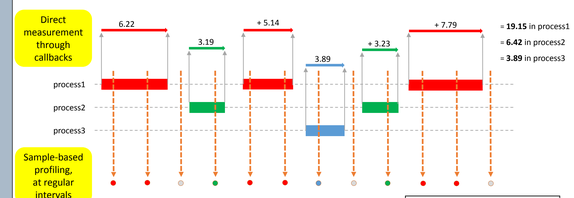



```
-- summary
  total CPU usage          =     19.445972s
  total non-process time   =      3.196789s
  total process time       =     16.249182s
  number of process activations = 3372820, average 4.817684 us/activation
--list of contributions
   16.43934 %     3.196789 s                    0.948 us/activation - total non-process time (kernel, profiling itself)
    -- processes sorted by total exec time --
   39.31143 %     7.644490 s   259107 activations     29.503 us/activation - TOP.mem_controller_0.smart_mem_q.send_transaction
   10.11653 %     1.967258 s   178498 activations     11.021 us/activation - TOP.im_2.generate_traffic
    9.33906 %     1.816071 s   171553 activations     10.586 us/activation - TOP.im_0.generate_traffic
    5.33261 %     1.036977 s    96177 activations     10.782 us/activation - TOP.im_3.generate_traffic
    2.66017 %     0.517297 s   242127 activations      2.136 us/activation - TOP.mem_controller_0.r_arb.transmit_responses
    2.17587 %     0.423119 s   501250 activations      0.844 us/activation - TOP.mem_controller_0.smart_mem_q.accept_requests
    1.97593 %     0.384239 s    38695 activations      9.930 us/activation - TOP.im_1.generate_traffic
    1.97324 %     0.383717 s   585061 activations      0.656 us/activation - TOP.mem_controller_0.r_arb.arbitrate_and_send_trans
    1.93752 %     0.376769 s    89170 activations      4.225 us/activation - TOP.mem_controller_0.fifo_2.transmit_responses
    1.80048 %     0.365678 s    85658 activations      4.269 us/activation - TOP.mem_controller_0.fifo_0.transmit_responses
    1.06135 %     0.206390 s    48073 activations      4.293 us/activation - TOP.mem_controller_0.fifo_3.transmit_responses
    1.00635 %     0.195694 s   242126 activations      0.808 us/activation - TOP.memory_0.end_of_busy_state_method
    0.83354 %     0.162090 s   126331 activations      1.283 us/activation - TOP.mem_controller_0.fifo_2.send_requests
    0.81893 %     0.159249 s   120481 activations      1.322 us/activation - TOP.mem_controller_0.fifo_0.send_requests
    0.77634 %     0.150966 s   179744 activations      0.840 us/activation - TOP.mem_controller_0.fifo_2.accept_requests
    0.74205 %     0.144300 s   171597 activations      0.841 us/activation - TOP.mem_controller_0.fifo_0.accept_requests
    0.43911 %     0.085389 s    58443 activations      1.461 us/activation - TOP.mem_controller_0.fifo_3.send_requests
    0.43163 %     0.083935 s    97895 activations      0.857 us/activation - TOP.mem_controller_0.fifo_3.accept_requests
    0.41969 %     0.081612 s    19229 activations      4.244 us/activation - TOP.mem_controller_0.fifo_1.transmit_responses
    0.16556 %     0.032194 s    22471 activations      1.433 us/activation - TOP.mem_controller_0.fifo_1.send_requests
    0.16043 %     0.031198 s    38457 activations      0.811 us/activation - TOP.mem_controller_0.fifo_1.accept_requests
    0.00128 %     0.000248 s      241 activations      1.031 us/activation - TOP.im_1.sync_process
    0.00086 %     0.000168 s      241 activations      0.696 us/activation - TOP.im_0.sync_process
    0.00058 %     0.000113 s      161 activations      0.701 us/activation - TOP.im_2.sync_process
    0.00011 %     0.000021 s       34 activations      0.627 us/activation - TOP.im_3.sync_process
```

Performance report on the memory controller architectural model (complete).

Comparison to Sample-based Profiling

Comparison to sample-based profiling
A very different way of measuring process time



Comparison to sample-based profiling, on memory controller model: top process

MAIN CONTRIBUTOR PROCESS OF MEMORY CONTROLLER SIMULATION, ON 10 RUNS

	Sample-based		Time from callbacks
	# hits	equiv. time	
Sim run 1	2619	7.865	7.681326
Sim run 2	2571	7.721	7.601298
Sim run 3	2685	8.063	7.704177
Sim run 4	2589	7.775	7.614316
Sim run 5	2619	7.865	7.666795
Sim run 6	2620	7.868	7.701617
Sim run 7	2595	7.793	7.670353
Sim run 8	2678	8.042	7.668937
Sim run 9	2702	8.114	7.69941
Sim run 10	2573	7.727	7.599858
Average	**2625.1**	**7.883**	**7.661**
Std. dev.	**47.435**	**0.142**	**0.041**
Dispersion	**1.81 %**	**1.81 %**	**0.53 %**

We ran the exact same simulation multiple times, with both profiling methods enabled on each run for a reliable comparison, and we computed the dispersion of the results.

3.4 x lower dispersion
=> Higher accuracy

Comparison to sample-based profiling, on memory controller model: top 8 processes

TOP 8 PROCESSES OF MEMORY CONTROLLER SIMULATION, ON 10 RUNS

	Sample-based			Time measured by callbacks		
Process	average	std. dev.	dispersion	average	std. dev.	dispersion
mem_controller_0.smart_mem_q.send_transaction	7.883183	0.142448	**1.81 %**	7.6608087	0.0409176	**0.53 %**
im_2.generate_traffic	2.180781	0.036418	**1.67 %**	1.9783610	0.0251477	**1.27 %**
im_0.generate_traffic	1.967868	0.045213	**2.30 %**	1.8125638	0.0248937	**1.37 %**
im_3.generate_traffic	1.109910	0.071067	**6.40 %**	1.0332264	0.0122282	**1.18 %**
mem_controller_0.r_arb.transmit_responses	0.723123	0.048850	**6.76 %**	0.5104633	0.0082844	**1.62 %**
mem_controller_0.smart_mem_q.accept_requests	0.803303	0.050115	**6.24 %**	0.4081642	0.0070231	**1.72 %**
im_1.generate_traffic	0.406607	0.018195	**4.47 %**	0.3849621	0.0043943	**1.14 %**
mem_controller_0.r_arb.arbitrate_and_send_trans	0.769069	0.035148	**4.57 %**	0.3731437	0.0071083	**1.90 %**

Dispersion is consistently lower with direct measurement by process callbacks
=> Useful for incremental optimization of simulation performance

Conclusion

We have proposed a complete interface that allows user's application code to be notified of all the state changes that occur in a SystemC process's lifetime, and execute a user-defined callback action, without a need for the user to modify the SystemC kernel. This interface is very generic and enables diverse and useful application areas, which we have demonstrated on a profiling case study.
In the simulation performance analysis domain, profiling solutions based on this process interface can produce very accurate results, which are useful for incremental optimization of model performance.