A Sampalib and SystemC™ comparison

Thierry Grellier, sampalib.org

Abstract

While SystemC™ is designed as a multi paradigm modeling language supporting multiple abstraction levels, Sampalib only focus on the transaction level modeling which actually covers the main usage of SystemC™ and comes with a richer toolset and better performances thanks to its focus. Hence a truly cycle accurate model can exceed 500 KHz while playing a significant activity in the system.

Introduction

SystemC™[1] is now a well known library though it now tends to be a language in itself specified by IEEE. It is a combined effort of the EDA industry to deliver a C++ HDL with the hope that this language will make easier interoperability, and combining multiple abstraction level to increase simulation speed. Yet some task forces are still working at specifying what the higher level of abstraction is. The trend is to call it TLM, although it is already itself divided into 4 levels by OCP-IP[2]. TL1 would probably be the level of choice when it would meet the performance constraints. This effort at upraising abstraction is because one still faces simulation speed issue although this is already a vast improvement against RTL. But upraising abstraction may not deliver accurate enough information, and it is difficult to work with another kernel that the reference one which isn't very efficient.

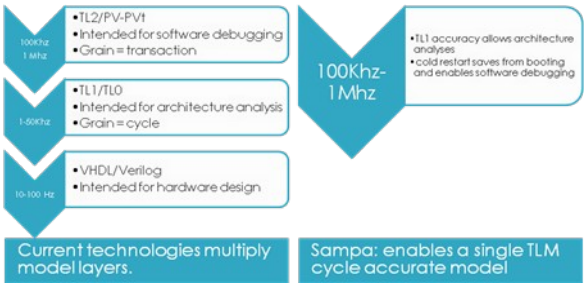

Figure 1: simulation speed according to abstraction level

Sampalib[3] is a new open source library tailored for the architecture analysis. This library includes core language features but also a set of components aiming at being extended. It is delivered with a comprehensive toolset but also some predefined component models. It thus isn't

bounded to a C++ library, and it is actually developed both in C++, with the help of QT[4] and in Lua[5].

The later is an interpreted language used for the configuration of a model and to develop associated tools. Plans are also to allow fast model prototyping with this language.

This library is delivered under a dual license:

- An open source license, the GPL v2[6],
- And a proprietary license that allows delivering closed source models but requires a subscription.

Concepts

Both library offers very similar features, but some differences are reveling strong design decision for the languages architecture. Figure 2 summarizes the key concepts and features in the core libraries.

	module	Port	interface	Event	Method threads	signals	clock	configuration	trace	Delta cycles	Persistence
SystemC	yes	yes	SystemC Specific	Simple	yes	yes	yes	no	Only signals	yes	no
Sampa	yes	No Bind_xx()	Pure C++	Several kinds	Methods only	Event Gated variable	yes	yes	yes	yes	yes

Figure 2: concepts present in libraries

Modules

The concept of module is equivalent in both libraries. This is name hierarchical element. In Sampalib however, one tend to use constructor with only one parameter, which is the name of the module. Having a configuration mechanism actually avoids adding parameters in constructor signature. The hope is that a dynamic factory of module can be used to have plug-in components.

Port

There is no generic concept of port in Sampalib. Being at transaction level, there is no data connection between modules, but rather point to point connections that can be expressed with a pointer or a transactional endpoint object. Thus ports are identified by public methods named bind_PORTNAME(INTERFACE*). This is then the responsibility of the binding method to propagate the binding and or perform checks. This allows to get rid of specific interface notion, and may save from using virtual functions (like for clocks).

In SystemC™, HDL-like ports were needed, and then the port concept was generalized, with the decision that ports must be bound (no dangling) and that number of binding must be checked. Then ports are using a lot of pointer and enforced the declaration of interfaces with pure virtual methods. This

also uses a lot of templates, which slows down compilation, and bloat code.

Events

Event model is a bit simpler in SystemC™ than in Sampalib which has several kinds of events. Hence there are clocked event whose notification isn't expressed as absolute time, but according to a number of edges. Only these clocked events can be canceled.

A dual category of event also exists, with single subscriber (mostly internal control of a module) and with several subscribers. This distinction allows some optimizations.

Another semantic difference is that Sampalib events are never triggered in the same delta cycle they are notified.

Finally it is possible to combine arbitrary and/or expression of events.

Methods/Threads

To model concurrency SystemC™ uses 3 kinds of processes. It was once reduced to 2, but then came back to 3 for the needs of behavioral synthesis. These are:

- clock sensitive threads which are stack-less,
- sc_methods which are stack-less too, and sensitive to events (including change value events from signals) at the entry point and looping. These are almost HDL processes, but sensitivity can be changed within the state machine.
- sc_threads which are stack-full and can use wait() calls with dynamic sensitivity. Reaching end of function exit the threads, so that often a while(1) {} loop is used.

Sampalib only has the equivalent of sc_methods. Having an equivalent to the sc_thread was envisioned, but it would have behaved like sc_methods, that is with looping at the end of the call. Basically a Sampalib process would have distinguishing a method from a thread only when a wait() call occurs, in which case the stack would have been preserved, for the current process, and another would have been allocated from a pool for the next process to run. This can save a lot of processing compared to SystemC™™, because there is then only a single queue of process to manage.

However the decision was not to introduce a wait() feature. This was motivated by several reasons:

- Using threads means that part of the simulation state is in stacks, which may limit the ability to persist the state of the simulation in checkpoints.

- Using wait() implies context switch which are costly in terms of simulation speed, and thus often discouraged in guidelines for modeling SystemC™ models.
- Modeling hardware is modeling state machine, so sc_methods style are natural to use.
- It is always possible to first write a thread and then transform it into a state machine

The main reason remains persistence, and not supporting wait() may be felt as boring limitation. Depending on future needs, it may be introduced in Sampalib, but not with introducing a new kind of process.

Signals

SystemC™ provides signal like any HDL, that is variable whose next state is committed after all processes have been executed, that is at the end of a delta cycle.

Sampalib doesn't have signals but gated variables. The variables are committed to their next state when their gating event is triggered.

As an event is always triggered the delta cycle after the notification it is easy to emulate a signal. But it is generally not needed in transactional event, where control event may be notified several cycles after the assignment.

Gated variables also allow modeling registers more easily and efficiently than in HDL, thanks to clocked events which can be notified for the next edge event.

Clock

Both libraries introduce a clock concept. But being more HDL oriented, SystemC™ propagate clock as a Boolean signal, while Sampalib uses a full interface, which is generally used for the triggering of gated variables or clocked events. A Sampalib clock internally manages a circular buffer of clock event. The size of the buffer is the maximum latency expected on the clock.

This allows dealing with timings in terms of clock cycles, which is more natural for hardware design, but also allows changing clock periods more easily without having to rescheduling all pending events.

Then insertion and retrieval of a clocked timed event is O(1) whereas timed notification are O(ln(num of timed notification)). Given the number of timed events occurring in a transactional model, this is quite a huge improvement.

Configuration

A model often needs to be parameterized at elaboration time, eventually during run-time.

There is no standard mechanism offered by SystemC™. Hence TLM working group or OCP-IP propose what one generally uses

for that purpose: text files. Some are tempted by xml files. As long as heterogeneous configuration mechanisms are used, one faces the problem of integrating them and avoiding multiplying the configuration files.

There can be also some inter-modules dependencies between the parameters that can't be captured in the model itself. Using pure data file as an input make the calculation exogenous to the model and leads to risk of incoherently manually change the parameters.

This is why Sampalib provides a C++ API to retrieve parameters from a lua script which can calculate them. This script language was initially designed for this purpose. It has then evolved[7] to a full and powerful[8] programming language while preserving very natural syntax for configuration. It is also very easy to embed in a C++ program.

Trace

SystemC™ only offers traces such as VCD, which is mainly for RTL level. At transactional level OCP-IP offers monitors that allow dumping channel states at each cycle for TL1 which produces huge traces with high noise ratio. And users often add their textual traces or other in the.

In the end, heterogeneous traces don't help much analysis because they can't be combined and cause major slowdown in simulation when activated or sometimes even when not activated but present (like OCP monitors).

Sampalib offers a unified trace framework that allows keeping all data and logging only reveling events.

It is possible to configure library so that trace processing belong to another thread than the simulation. Choice was made to use printf for user traces rather than cout for efficiency reason although formatting is potentially more error prone. The simulation speed with traces on is thus still very high.

Then the output allows an easy post processing that filters the useful event with lua scripting. A channel activity comparison script, provided in package is an example of this.

Persistence

Simulation can often take time, and share long initial time. This can make debugging very boring when a crash occurs quite late.

As we have seen, a full usage of SystemC™ features, such as the sc_threads, makes that part of simulation state resides in sc_threads stacks that can hardly persist. Sampalib has thus chosen to support only sc_methods, but it also enforces programming constraints to benefit from an underlying persistence library, Post++[9]. Hence periodic checkpoints of the

simulation can be stored, at almost no runtime penalty.

It is thus possible to resume a simulation in a later run. This is called cold restart. A hot restart fork the process when it executes, to preserve a checkpoint (the parent process) but in the same run only. A hot restart could be useful feature yet for SystemC™.

Protocol

Transaction level protocol modeling is very supported by SystemC™ itself. But OCP-IP, and a OSCI-TLM workgroup provides two solutions for that purpose.

However OCP-IP channels are not very efficiently designed, because this is exclusively event based communication and copies a lot the transaction payloads.

OSCI-TLM models can hardly model cycle accurately transactions because it is intended for using delay notation. But delays don't scale that well (log(n)) and are difficult to combine without losing a lot of accuracy. It is also a template intensive framework requiring writing a lot of code for few features.

Sampalib provides a simple protocol intended for TL1 abstraction level. It is simpler to model components and accurate. To stay efficient and contrary to OCP-IP, it uses method calls until the other endpoint when a payload is sent, only once through the channel that doesn't keep a copy of the payload. The role of the channel is to broadcast an event the clock cycle after the acceptance of the payload. This event can be used to gate a variable to store the payload. The payload uses a copy on write semantic for the less mutable fields which allows a passing by value semantic.

The figure below sums-up the phases of a word transaction cycle.

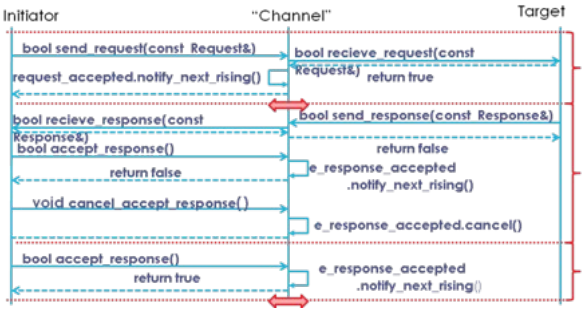

Figure 3: transaction protocol

Conclusion

An architecture detailed below is delivered with Sampalib and has been tested on the following environment, a laptop with a T7200 2GHz processor, under Linux ubuntu 64 on WMware, and compiled with g++ 4.1.

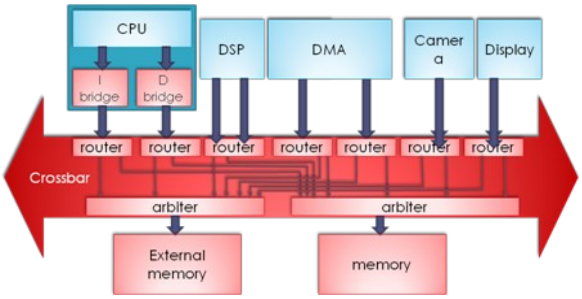

Figure 4 example of architecture model

The total compilation time for the library and the example is 30s, and the runtime for 10 million cycles is 16s with a bus at 133MHz and arbitration and routing decided at each cycles (not burst grain). That is 625KHz. This with concurrent traffic as such 30Mb/s on the camera, 30Mb/s on the display, 3 MB/s on DMA and 5% miss ratio on the CPUs.

Sampalib is thus an alternative to model SoC. It introduces valuable concepts that may be worth introducing in the SystemC™ kernel. Revising the SystemC™ language is a slow process, and there is limited acceptation for non binary compatibility for integration reasons. There are many efforts to make SystemC™ evolving, in SystemC™ workgroups, in EDA tools or independent initiative such as greensocs[10].

However Sampalib shall still keep a better efficiency both in terms of compile time and runtime thanks to its focus that makes it a much smaller kernel. This focus shouldn't actually prevent from using it for all current usage of SystemC™.

Sampalib value also comes of a rich set of features that have been neglected or left to side-product so far (efficient high level trace mechanism, configuration mechanism, predefined components and associated analysis tools), or are very difficult to achieve given the language specification (simulation checkpoints).

1http://www.SystemC.org
2http://www.ocpip.org/pressroom/schedule/speaking/papers_presentations/2005_09_28_FDL05_OCPIP_tim.pdf
3http://www.sampalib.org
4http://www.trolltech.com
5http://www.lua.org
6http://www.gnu.org/licenses/old-licenses/gpl-2.0.html
7http://www.river-valley.tv/conferences/tex/tug2007/index.html#Roberto_Ierusalimschy
8http://shootout.alioth.debian.org/
9http://www.garret.ru/~knizhnik/post.html
10http://www.greensocs.com/SystemC/SystemCNeeds