Multi-Threaded SystemC and external interfaces

Session 1.2

accellera SYSTEMS INITIATIVE

SYSTEMC™ EVOLUTION DAY
MAY 3, 2016 | MUNICH | GERMANY

Presentation Copyright Permission

– A non-exclusive, irrevocable, royalty-free copyright permission is granted by **GreenSocs** to use this material in developing all future revisions and editions of the resulting draft and approved Accellera Systems Initiative **SystemC** standard, and in derivative works based on this standard.

Multi-threaded?

- There are two approaches to multi-threading:
  1. Fine Grained Parallel SystemC Execution, as discussed this morning.

  2. Course Grained multi-threading:
     - Putting one or more sub-systems into a separate host thread.
     - Or – combining several simulation kernels.

- First issue – Synchronizing between simulation kernels

Inter-Simulation Synchronization

"Asynchronous Wait"

Guillaume Delbergue

And

Mark Burton

Existing SystemC Constructs

- async_request_update provides a mechanism to notify a SystemC event in a thread safe manner.
  - Member function async_request_update shall cause the scheduler to queue an update request for the current primitive channel in a thread-safe manner with respect to the host operating system. The intent of async_request_update is that it may be called reliably from an operating system thread other than those in which the SystemC kernel and any SystemC thread processes are executing.

- We have a way to inject async events, but we don’t have a way to ‘wait’ for async events.

Waiting for an async event

- We don't know when the event will arrive, We don't know how much 'simulation time' should pass.
- Events may trigger other activity, if you are waiting for an async event, you may run out of normal events, but the simulation is not over !
- Hence two additional semantics:
  1. When you are waiting for an async event, do not advance simulation time.
  2. When you are waiting for an async event, do not stop the simulation if you run out of (normal) events.

Asynchronicity

- Asynchronicity is a property of an event.
- But, then you may want to asynchronously wait for a time
  - We propose to add a time class : sc_async_time
- All functions (wait, notify) have the normal semantics, with the additional semantic rules for async events.
- (you can do all of this with an async function, rather than events, but when you ‘async_wait’ for a list of events, you ‘taint them all’ as asynchronous, which is maybe not what you want)

Additional functions

- N.B. Wait(event, time) has the semantic of waiting for events (the logical combination of an event list), or the time.
- We would like a semantic that says “At this point in the simulation, I would like to sync with an external simulator”
  - I may have to process events at this time, but I should not advance time while I wait for the other simulator to sync.
- Hence we could have :
  - While (not_in_sync) { wait(async_sync_event, 0); }
- BUT this will consume 100% of CPU as we spin waiting
- Ideally we would like to say
  - Wait(time t AND async event e)
- meaning “wait for the condition to be true that time is t AND the event e has fired”.
- In order to achieve this, clearly the kernel would have to pause time at ‘t’ waiting for the event.
  - But that is not the meaning of ‘and’ in the context of events. The ‘AND’ construct means wait for both events to have happened, time could move on.

Many choices

- A ‘true and’ wait (wait time and event (list).
- Ignore the issue (burn more trees)
- Add new semantics to something else (e.g. sc_pause…)
- A totally new function

- We propose sync_async(time, async_event)…

- Sync_async means ‘wait until the time and the async event occurs’ (which means the kernel may have to pause, waiting for the event, if it has already reached the time).
- It is only defined for async events.

- cleaner than ‘overloading’ wait, though it seems a pitty.

Wait, sync, async, time….

	Sc_time	Sc_async_time	Sc_event	Sc_async_event
Wait	Normal wait	Do not advance time, exit at time, (simulation may end).	Normal wait	wait for the event to happen, time may advance, the simulation should not end
Sync_async	n/a			wait for the event to happen (do so immediately, do not run more delta cycles)

Nb some debate as to whether wait(0) == wait(async(0)) – you choose!

Wait, sync, async, time….

	Sc_event		Sc_async_event	
	Sc_time	Sc_async_time	Sc_time	Sc_async_time
wait	Normal wait	Do not advance time, exit on event OR time, (simulation may end).	Normal wait	Do not advance time, wait for the events to happen, or for time to be caused by other events to be reached
Sync_async	n/a		Wait for both the time and the event. If the event doesn't arrive by time, then simulation should wait for the event (executing any delta cycles as appropriate). If the event arrives before time, then allow systemc time to advance to time.	Wait for both the time and the event. If the event doesn't arrive by time, then simulation should wait for the event (executing any delta cycles as appropriate), If the event arrives before time, wait until time (the simulation may end)

- We are able to simply sync between two systemc's running in different threads.

- Each kernel runs a small system, using 'quantums'
- At the end of each quantum, the two kernels 'sync' using an async event.

- For each kernel, there are two possibilities, either they have already reached the end of their quantum, and wait for the other to catch up, or they receive the other kernels notification before they have themselves finished.
- Sync_async (time, async_event) provides exactly that semantic, waiting for the time (not allowing the kernel to move forward from that time) and the event to happen.

Summary of proposed changes

- Add new class sc_async_event,
  - Two Semantics:
    1. Do Not Advance time to this event
    2. If there are primitive channels blocking for events of this type, then don't stop the simulation.

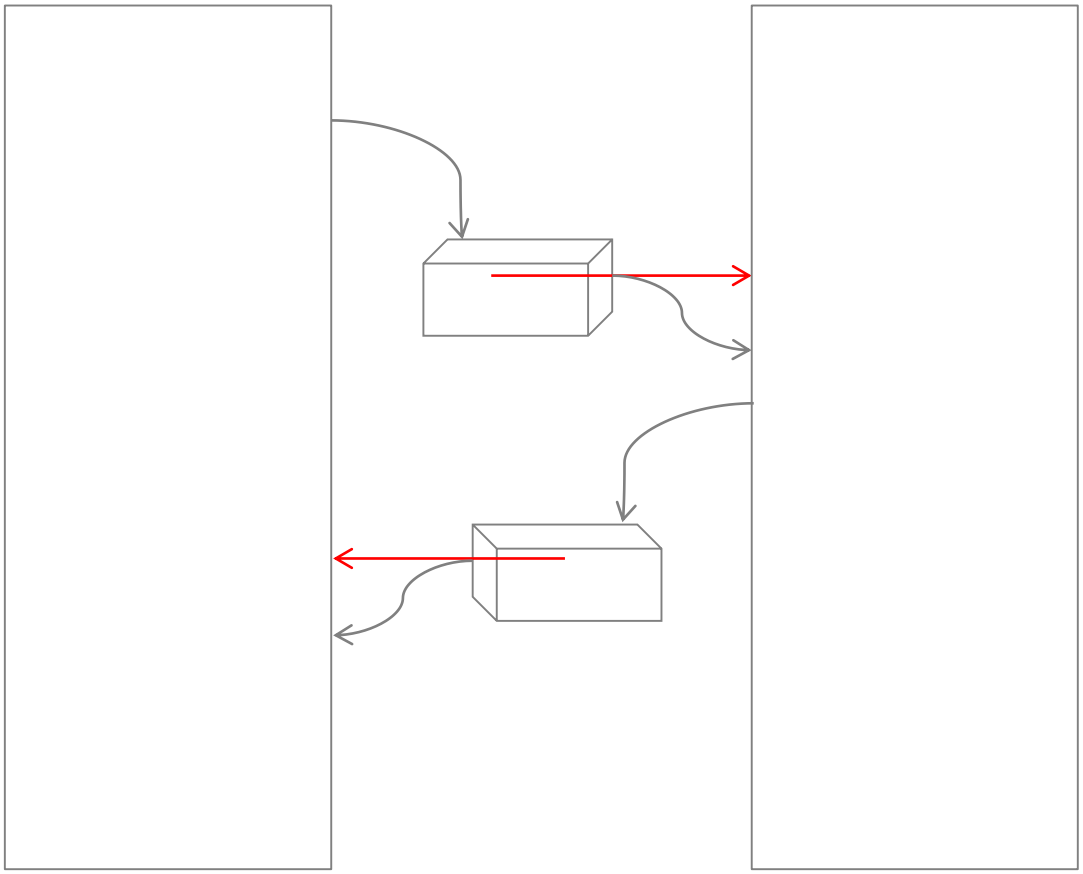
- Add a new type of 'wait' like statement called sync_async
  - Semantics :
    - Async_sync(time T, event(list)) – wait until both the event has happened ***and*** it is time T. If the event has happened, you must continue to run delta cycles, but you may not allow time to advance. (Hence on completion, SystemC time will be T, ***and*** the event will have occurred).
- Add a new sc_async_time type.
  - Hence it is possible to wait (or notify) a time, with the same async semantics.

Inter-Simulation communication: TLM + ?

Just TLM2 ?

- TLM 2 certainly helps, BUT
- For LT models - good news, you can play loose with time (within the quantum).
  - So you do not need to sync kernels on each transaction.
- For AT models – Can we have temporal decoupling?
  - Theoretically NO
  - Theoretically, all cross simulation communication should be sync'd !
- And then – THREAD SAFETY…

Thread swap -tastic



Post to letter box
Async notify,
Other thread (finally) picks up
…

And all the same the other way

Are we happy?

- Insist all models are thread safe
  - (or handle switching threads themselves)
  - (That may also means making sure all SystemC kernel activity is thread safe!)
  - BUT – this could work if we say that is must be the case for models called from external simulations…?

- Hack-a-day your own solution, if you know the model you are communicating with is thread-safe – good luck.

- We standardize some mechanism to identify thread safe models, in all other cases, you must take the pain.
  - E.g. using an extension on the tlm-port.
  - Or a CCI mechanism

- For Sync :
  - Proposal with code that is a simple addition to SystemC and will enable sync between simulators

- For communication:
  - TLM looks like a good start, but AT will suffer a lot of sync's, and LT needs to know when it's thread-safe.
  - (Apart from that, an LT interface looks much like any other remote function call interface – e.g. RPI)
    - [Fill in the gap - a good proposal]

Multi-Threaded SystemC and external interfaces

Session 1.2

accellera SYSTEMS INITIATIVE

SYSTEMC™ EVOLUTION DAY
MAY 3, 2016 | MUNICH | GERMANY

Summary

- For Sync :
  - Proposal with code that is a simple addition to SystemC and will enable sync between simulators

- For communication:
  - TLM looks like a good start, but AT will suffer a lot of sync's, and LT needs to know when it's thread-safe.
  - (Apart from that, an LT interface looks much like any other remote function call interface – e.g. RPI)
    - [Fill in the gap - a good proposal]

Summary of proposed changes : Sync

- Add new class sc_async_event,
  - Two Semantics:
    1. Do Not Advance time to this event
    2. If there are primitive channels blocking for events of this type, then don't stop the simulation.

- Add a new type of 'wait' like statement called sync_async
  - Semantics :
    - Async_sync(time T, event(list)) – wait until both the event has happened *and* it is time T. If the event has happened, you must continue to run delta cycles, but you may not allow time to advance. (Hence on completion, SystemC time will be T, *and* the event will have occurred).
- Add a new sc_async_time type.
  - Hence it is possible to wait (or notify) a time, with the same async semantics.