Accellera Federated Simulation Standard (FSS)
Proposed Working Group

Mark Burton
Problem Statement

Different simulation approaches and standards...

Avionics
VISTAS / VHTNG

Space
SMP2

Semi’s
SystemC
TLM
IP-XACT

Automotive
openADx
openDRIVE
openSCENARIO
openCRG
openPASS

Mechatronics
FMI / FMU

How to bring these industries and simulation approaches together?
A “Core Team” has been established in 2019 to exchange knowledge and best-practices

- Inventorize existing simulation standards, its usage and coverage
- Understand requirements, potential overlaps, and points of interaction

Foster and initiate actions to improve and co-ordinate standards development and integration of simulation technologies

- Collaborative action to make standards evolve according to our needs (e.g., interoperability, scalability, …)
- Explore cross-industry collaboration between Standards Developing Organizations and Consortia supporting open innovation and collaboration

Core Team members*

Airbus
Aptiv
AVL
Bosch
Collins Aerospace
IRT Saint-Exupery
NXP
Qualcomm
Shokubai
Spacebel
Federated Simulation Standard – Proposed WG

• Charter
  – Cross-industry collaboration to improve the interoperability of product and environment simulation using existing and new open standards

• Scope
  – Develop a standard (API) and open infrastructure to enable cross-industry interoperability of simulation frameworks

• Purpose the Proposed Working Group
  – Identify industry interest and requirements for a standard / API covering addressing interoperability of simulation

• Leadership
  – Chair: Martin Barnasconi (NXP), vice-chair: Mark Burton (Qualcomm)

• Envisioned Stakeholders
  – Companies active in different industry segments (e.g., Semiconductors, Automotive, Avionics, Space, ... )
  – Companies active in different stages of the value chain (Tier2, Tier1, OEM)
FSS: Enabling cross-industry interoperability of simulation frameworks

• Approach: Leveraging and connecting existing standards and industry formats
  – Not re-invent wheels

• Introduce standardized interfaces
  – Enabling interoperability between simulation frameworks

• Targeting a scalable simulation and modeling ecosystem
  – Support models and simulation domains used at different levels of the ‘OSI stack’
The problem

For some the problem is “only” the serial interfaces

How do we connect Engine controller A to device B.

But our problem is deeper….

How do I re-use models
How do I connect models of one type to another
How do I even connect models of the ‘same’ type!
And How do we deal with HW/SW ‘connections’….
DATA Exchange (easy?)

TIME Sync (HARD?)
Every simulation environment has a different notion of “time”

Many have multiple “times”:
(Wall clock, simulation time, local time, quantum time . . . )
Interactions

Abstracting data is not trivial…

But…

Each have a notion of ‘time’
Ensuring that each is “happy” _IS_ hard!
SystemC VP

QEMU/CPU → ‘I2C’ controller → ‘I2C’ device

Data: Not Rocket Science

Time .....?
When Synchronisation becomes n-way:
- central controllers
- “global” notions of time

But when the simulations beginning combined do not share these?
- Adapters/shims are only possible when the ‘concepts’ of time match
- If time is variously ‘abstracted’ things are more tricky. . .
Federated Simulation Standard – Ideas (1)

Main idea is to introduce a ‘message passing’ and ‘adapters’ approach to bring different models / simulation domains together

- Approach should support system models and simulation domains used at different levels of the ‘OSI stack’
- Assess available standards and their capabilities to enable interoperability
- Aim is not to replace existing standards, but to standardise how they can be adapted to work with each other
Example: S/W and H/W

S/W expects timer interrupts at the end of each period, but....

A Virtual Platform may not know that time it is !!! Interrupts might fire too quickly....
• Synchronous Application can execute in zero simulation time. Time is an intrinsic artifact of event types and loops.

• Asynchronous Application polls for events every $\Delta t$

• RTOS is formed by combining Asynchronous and Synchronous

• Audio, Video, HMI follows Synchronous model

• Synchronous resembles Software in the loop

• Asynchronous resembles Hardware in the loop

• Assess available standards and their capabilities to enable interoperability
HW/SW Interaction

- Simulation needs awareness that certain events must be completed before or after corresponding Milestone Marker.

- Solutions to this exist... but
  - “We’re not talking” to each other.
  - Not universally adopted
  - Not connected

- Exit to real world. Connect to real HW

---

Synchronous Application

Interrupt

Event B

Event A

Environment

Asynchronous Application

Milestone Markers

Δt

Quantum time

Simulation Time

If simulation Time <= Wall Clock

Wall Clock

Exit to real world. Connect to real HW
Don’t throw the baby out with the bathwater!

- Lots of standards exist
- All have good/bad points
- Plan is to link/reuse
- NOT replace
How will we work?

- **SystemC**
  - TLM
  - IEEE 1666-2011
  - IEEE 1666-2023
- **VHTNG**
  - ED247/ VISTAS
  - ED247 Revb (DDS)
- **HLA / IEEE1516**
  - HLA on DDS
  - HLA on Zenoh

Groups:
- **Group 1**
- **Group 2**
- **Group 3**

Demonstrators:
- **dcp**
- **FMI2**
- **FMI3**
How does this relate to SystemC?

• What parts are there?

• Are there enough?
Sync primatives : Do we have enough?

<table>
<thead>
<tr>
<th>Primative</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sc_suspend_all()</td>
<td>Suspend all systemc threads if none are unsuspendable.</td>
</tr>
<tr>
<td>sc_unsuspend_all()</td>
<td>Unsuspend.</td>
</tr>
<tr>
<td>sc_suspendable()</td>
<td>Mark suspendable.</td>
</tr>
<tr>
<td>sc_unsuspendable()</td>
<td>Mark unsuspendable, such that systemc can not suspend all.</td>
</tr>
</tbody>
</table>

Class async_event

Wrap “request_update” (the only thread safe method in SystemC) in a convenient sc_core::sc_event type.

async_attach_suspending/async_detach_suspending to ensure SystemC does not quit on event starvation.

NB “request_update” events are executed by the kernel even if the kernel is suspended.

Class RunOnSysC

Convenience layer to schedule a lambda expression to be run by the SystemC thread. (NB this will run on the next delta cycle). Provides:

bool run_on_sysc(std::function<void()> job_entry, bool wait = true)

realtimelimiter

A module which prevents time from advancing beyond realtime.
“Cloud TLM”? 

- Basis of any “external” interface: (un)suspend interface

Requirements:
- Requires sc (un) suspend

Code available: [github:quic/qbox](https://github:quic/qbox)
Bidirectional serial socket

- Simple set of standard TLM sockets, can cover most serial interfaces
- Not ‘standard’ just one way of modelling interfaces

- SystemC models of UARTS, NICs, I2C, ...

- 4x tlm 2.0 GP interfaces.
- Not all fields used, but protocol used for compatibility.
- (May be sent over RPC remote)
- Convenience layer provided to enqueue data

Code available github:quic/qbox
RPC tlm

- ‘n’ inputs/outputs, etc..
- Relies on (un)suspend interface, and ‘asynchronous’ events.
- Pass TLM-2.0 interface over RPC

CCL parameter database is shared (names de-mangled)

Code available
github:quic/qbox
Sync policies

- TLM 2.0
  ONLY DETERMINISTIC MODE

- ‘parallel’ TLM 2.0
  With a fixed Quantum.

Code available
github:quic/qbox
Sync policies

- ‘Windowed’ quantum

- Unconstrained

Each `b_transport` indicates a time, which can be used to allow SystemC to advance.

Code available: [github:quic/qbox](github:quic/qbox)
Conclusion

• Lets get married

• Lets start the conversation

• Lets work on bringing standards together.