

Issues impeding on transactor portability and common use model

Background:

Transactor portability from simulation to hardware-based acceleration and lack of consistent use model of SCE-MI implementations continue to impede on the adoption of SCE-MI, Transaction-based acceleration (TBA) and Co-Emulation Modeling methodology.

Several issues are impeding on transactor portability and common use model. These issues with the recommended approach to resolving the issues are listed below. The issues that need to be addressed in SCE-MI 2.0 are listed first under in the SCE-MI 2.0 bucket in order of priority. The issues that could be addressed in a subsequent SCE-MI release (3.0) or as a parallel longer term effort are listed in the “Other” Bucket in order of priority.

SCE-MI 2.0 Bucket:

1. Support common interfaces for simulation and acceleration – Elimination of the uclock/cclock mechanism for as broad a class of BFM's as possible. *Already presented [1]*

2. Native support of variable-length messages (VLMs) – *Already presented [1]*.

3. Define minimal transactor architecture requirements – SCE-MI 1.1 does not define any minimal transactor architecture requirements besides defining the semantics of transporting messages between message ports and message proxy ports. This leaves transactor modelers with many questions unanswered. Best practices would suggest that a transactor should contain a BFM on the HW side communicating with proxy model on the SW side implementing a given bus protocol through a mutual agreement between these two partitions. As a matter of fact, SCE-MI 1.1 even defines the BFM on the HW side as the ‘transactor’ as opposed to best practices. This lack of architectural definition may be misleading to IP vendors attempting to building transactors used in both simulation and emulation.

Furthermore, lack of transactor architecture definition also impedes on building portable transactor architecture where the proxy models could be developed in various HVLs (such as SystemC/Vera/e/ System Verilog) based on the user chosen HVL w/o requiring any modifications of the reciprocal BFM on the HW side. The principle of separating the implementation language from the interface definition of the proxy model could open the door for variety of portable transactors serving different segments of the HVL market.

As a minimum an application note containing some guidelines should be defined to suggest that a transactor would contain HW and SW side partitions in its architecture to support all common HVLs, C and C++ on the SW side in addition already mentioned Verilog/VHDL support on the HW side.

4. Clearly define *alternating vs. concurrent* use model – SCE-MI 1.1 doesn't make any explicit assumptions if the HW/SW side is running concurrently or in an alternating mode. It makes however an implicit assumption of concurrent use model. While alternating mode provides better determinism and consistency with simulation results, a concurrent use mode may increase performance. But it may also cause indeterminism (at least with simulation) if the user is not careful enough when writing the tests.

I recommend defining these two use models more distinctively with appropriate semantics changes when one or the other is being used to accomplish consistent simulation results in each use model across implementations. I would also suggest that it would be dealt with separately from the variable length messages and the use of batching of messages. The concurrent use model is independent of whether messages are of small/large size, fixed/variable size or delivered individually/ in a batch of messages.

5. Provide SCE-MI infrastructure support for *implicit reactivity control* – I use the term ‘implicit reactivity control’ as the capability to deliver SCE-MI messages in batches under the control of the end user. Batching of messages is derived from the level of granularity defined by the test when delivering or receiving a sequence of transactions. It is orthogonal to whether fixed or variable length messages are used and it is also orthogonal to using concurrency vs. alternating

co-simulation in streaming mode. I would like to focus on transactor portability issues under reactivity control.

While some types of interface protocols lean better to communicating messages/transactions in batches over others (for example, unidirectional networking interfaces as opposed to bi-directional tightly coupled processor interfaces), reactivity control of transactions on a given interface should be left to the end user's decision who writes his/her application level test program and should not require modification of the transactor model. This implies that the transactor should not be required to be aware of when messages are transferred in batches and should not need to be explicitly configured to support batch transfers (Unless the modeler wishes to configure it at his/her discretion). Only the SCE-MI infrastructure will be configured to loosen some of its existing semantic requirements (as described below) to support transfer of a batch of messages.

To be clear, SCE-MI 1.1 supports the arrival of messages in batches in both directions (as long as messages ordering requirements are satisfied). For instance, if the cclocks are stopped, multiple messages can be allowed to accumulate on an output port before it is serviced and the infrastructure can move all messages as a batch if it wants to. For input ports, if the test sends multiple messages to an input port, SCE-MI infrastructure is certainly free to move them as a batch.

However, in both cases the determination to queue messages into a batch is conducted by the producer of the messages (i.e. the transactor modeler) as opposed to the end user that determines the reactivity tradeoffs assumed by its tests. By SCE-MI 1.1 spec semantics, Receive() callback must be called on the SW side when service request is issued on the HW side which implies that batching of multiple output messages can only occur at the control of the BFM (via clock control). This forces the BFM to be (unnecessarily) involved in reactivity control, requires modification of a BFM model configuration interface for user level reactivity control or forces the creation of multiple derivatives of the same BFM for each 'batching figure'.

The simplest mechanism to supporting batching of messages could be using a FIFO that would store and convey a batch of output messages. The depth of the FIFO would determine the number messages that would be transferred in a single synchronization and service requests would be issued when FIFO is full. This will deviate from the current semantics of SCE-MI 1.1 where service requests are issued at the control of the BFM and will require updating the SCE-MI spec. Similarly, a FIFO could be used on an input port by IsReady sensitive transactors, but service requests could be issued when input FIFO is empty and avoided when the buffer is capable of satisfying the service request. This will again deviate from the current semantics of SCE-MI 1.x where service requests are issued when the BFM is ready to receiving the next message and not when the FIFO is empty.

Possible enhancements to the SCE-MI interface would allow reactivity control on a per message port basis that will modify the service request semantics from 'message driven' service request (as per SCE-MI 1.1) to 'FIFO driven' service request. Other means will be required to set/modify FIFO depth on a per-port basis.

Another implication of the above use model is the support of delayed arrival of messages on buffered ports. This implies that output messages on multiple output channel may not arrive in the order they were sent. However, messages do arrive in order within a channel and arrival is deterministic between runs using the same FIFO settings. SCE-MI 1.1 prohibits this scenario from happening and thus this topic will need to be reconsidered.

Given that multiple messages may be batched into a output buffer, multiple messages may be flushed and Receive() could be called sequentially multiple times, once for each message. If the Receive() callback code is called multiple times before it conveys the messages to its target(s) (which is conceivable if for example SystemC notify() is used only after all Receive() callbacks complete execution in the context of a single thread), it is quite possible that each Receive() callback will place the current message over the previous message rather than queue the messages for subsequent processing. This would require some guidelines and examples that would recommend to the transactor modeler to queue messages coming out on output port for further processing.

6. Provide basic infrastructure support for transaction recording support – Transaction level recording and viewing is evolving as important debug technique that support transaction-level abstraction.

The use of this capability is mostly dependent on the ability of the transactor to gather the necessary information (mainly transaction ‘begin’ and ‘end’ times) and record the transaction with these attributes into a data base for future retrieval. There are several issues that need to be addressed in this regards. But only the following issue should be addressed in SCE-MI 2.0.

SCE-MI only provides automatic message level cycle stamping which does not allow the BFM to make an explicit reading of transaction ‘begin’ and ‘end’ time at the beginning and end of the transaction. While the SCE-MI 2.0 interface definition should not include a use model definition for transaction recording, it should provide the means for BFMs to read and convey ‘begin’ and ‘end’ times. This issue is addressed by the Cadence proposal for SCE-MI 2.0. [2].

7. Provide SCE-MI spec classification across target users – While the SCE-MI spec (section 4.3) defines three categories of users, it doesn’t follow through on this classification throughout the specification. It is left to each target audience (I’ll call it class) to figure out which sections of the spec apply to each class. The end user is left to deal unnecessary with the SCE-MI spec complexity on issues that have been addressed by the SCE-MI infrastructure implementor or the transactor implementor.

Transaction-based acceleration and Co-Emulation technology will be adopted much easier if the end user required knowledge will be kept to the bare minimum and the transactor implementor would be shielded from specifications that clearly target the SCE-MI infrastructure implementor. In additional, Accellera copyright restrictions prohibit pulling the relevant sections out of the spec into the respective target audience end user documentation. This forces end users and transactor implementor to dig into the entire SCE-MI spec instead of simply reading the end user product document or pointed only to the portions in the spec applicable to them.

8. Allow for mixed signal/transaction level to co-exist – traditional co-simulation was based on signal level interaction between the hardware side and the software side. SCE-MI 1.1 elevated this communication to the message/transaction level. However this requires creating SCE-MI compliant transactors on ALL interfaces.

It is conceivable that some users will move to using Transaction based acceleration/Co-Emulation methodology in a gradual manner where some interfaces will continue using signal level interaction. This may apply for example to cases where some interfaces are not highly active, or even for ease of use reasons where the testbench wants to read or modify certain signals values infrequently during simulation or during a debug session. While SCE-MI should not include definition of signal based acceleration semantics, it should not contain requirements that impede on such usage either and allow the two use models to co-exist.

“Other” Bucket Issues:

9. Provide timed testbench support – Advanced verification environments may require mix level simulation of RTL and TLMs that support timed constructs. Timed constructs in the TLM may be ‘predictable’ or ‘assumed’ vs. ‘actual time’ implemented at the RTL. Testbenches may also wish to control simulation attributes (such as end of initialization or end of simulation) using simulation time. This is a highly common practice among simulation users. But when running in SCE-MI 1.1 accelerated mode, SCE-MI doesn’t mandate the view of common time across the partitions. Contrasting this to simulation, common time is supported implicitly in simulation mode where implicit synchronization is maintained between the hardware side partition and the software side partition.

Users should not be forced to deal with the above limitation when moving from simulation to acceleration mode and SCE-MI could be enhanced to support a common view of time across hw/sw side boundaries (if the software side supports the notion of time of course). Given that not all HVLs support the notion of time, a possible solution would define a time-based synchronization paradigm parallel to the event-based synchronization paradigm supported by SCE-MI 1.1 today.

Using timed constructs in the proxy models should be discouraged to insure portability of

transactors across various implementations and the use model would only allow other abstracted components (such as TLMs and stimuli generators) to use timed constructs.

10. Provide HVL native API support – Assuming different HVLs would be used for creating transactor proxy models, SCE-MI SW side API presents today the ‘lowest common denominator’ API. As such, it does not present an easy-to-use/native interface for each of the HVLs.

To maintain language neutrality on one hand, while making SCE-MI easy to use for native HVL modeling, a native SW HVL language API that encapsulates the SCE-MI SW side API could be defined. For example a thin SystemC layer (named adaptor) could present a SystemC TLM API to the proxy modeler making it easier to build a proxy model in SystemC that interfaces with SCE-MI on one side and with other SystemC TLMs on the other side through a common TLM API.

Given that SCE-MI is and should be maintained as HVL language neutral, providing the native HVL layer in open source could address the language neutrality issue and be used by all users and proxy models running in a specific HVL environment.

11. Provide native blocking/non-blocking interfaces – While this issue relates to input and output ports, I’ll focus on SCE-MI input message ports. SCE-MI offers the Send() method as a non-blocking method for sending input messages. However if the proxy model wishes to use a blocking interface (in a threaded environment such as SystemC) that will block sending the message until the BFM requests it, it would need to implement code that throttles input messages based on IsReady. The use of IsReady() wrt the Send() method semantics is left to the modelers who may do what they wish. For example, the proxy model may send one message per IsReady call back, 10 messages or even send no new message. SCE-MI 1.1 semantics doesn’t mandate any relationships between IsReady() and Send().

Given that many HVL software transactions use the blocking vs. non-blocking semantics, it would have been nice if either a single method or two methods that define if Send() will be blocking or non-blocking will be supported (like Send and Nb_Send) in the native HVL. In addition, the SCE-MI spec semantics could be tightened at the interface level to insure a consistent behavior of a blocking interface wrt IsReady/Send semantics that all transactors will comply with.

Last, the use of blocking interface entails the use of threading package. SCE-MI 1.1 rightfully does not mandate the use of threads and does not assume any threading package. A native HVL adaptor could satisfy this requirement by providing a blocking and non blocking interface that uses its native threading package. The adaptor will use IsReady(), Send() and Receive() under the hood but provide native HVL functions for transaction level communication.

12. Provide common methodology for transaction recording support – Transaction level recording and viewing is evolving as important debug technique that support transaction-level abstraction. The following describes the broader set of issues required to support it.

- a) SCE-MI only provides message level cycle stamping today which does not allow the BFM to make an explicit reading of transaction ‘begin’ and ‘end’ time at the beginning and end of the transaction. This issue is addressed by the Cadence proposal.
- b) The delivery of ‘begin’ and ‘end’ time is based on cycle stamp rather than simulation time. SystemC/SCV (that supports transaction recording as part of the standard) for example supports the notion of time. SCE-MI 1.1 doesn’t provide standard means for converting cycle stamp information to absolute time natively viewed by the user. This can be addressed however by the HVL language specific adapter described above.
- c) Given that transaction recording may be costly (performance wise) it must be provided with means to turn it ON or OFF. While again this could be handled by the transactor, ideally the transactor should not be aware if the user wishes to turn recording ON or OFF. But even if it does, common functions for turning recording ON and OFF should be used to build portable transactors across implementations.

References:

1. Enhancements to SCEMI 1.0, First Principles, Matt Kopser, Apr 8, 2004.
2. Standard Co-Emulation Modeling Interface (SCE-MI), Cadence Proposed DRAFT, Version CPD.a, April 20, 2004