

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 and Transaction-based acceleration (TBA) and Co-Emulation Modeling methodology.

Some of the following portability and use model issue have interface or modeling ramifications, but it is hard to simply associate them with these two buckets. In the last ITC discussion, we identified the “other bucket” (in contrast to the interface and modeling buckets). Hopefully a better term will be defined when the issues and limitations described here will be analyzed. For now, I have taken the liberty of raising issues and sharing my thoughts in areas that touch on various buckets while paying less attention as to which of the buckets the issue belongs to.

Limitations:

Lack of SCE-MI infrastructure support for reactivity control – I use the term ‘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 the discussion in Per’s and Russ’s write-ups dealing with using concurrency vs. alternating co-simulation in streaming mode. Naturally when the size of a batch (of messages) is so large that it spans through the entire test, the batch can be viewed as ‘streaming’. I would like to focus on transactor portability issues in a test environment that requires 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 user’s decision via his/her test program and should not require modification of the transactor model. This implies that the transactor should continue to produce/consume messages w/o being aware when messages are transferred in a batch and should not be explicitly configured to support batch transfers. Only the SCE-MI infrastructure should be configured to loosen some of its existing semantic requirements (as described below) to support transfer of a batch of messages (defined as larger than one message or even a fully streaming message).

To be clear, SCE-MI 1.x supports the arrival of messages in batches in both directions (as long as messages ordering requirements are satisfied). For instance, if the clocks 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 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 as opposed of having the test configure the SCE-MI infrastructure to allow batching of messages. 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 of fixed length messages that would be batched and service requests would be issued when FIFO is full. This will deviate from the current semantics of SCE-MI 1.x where service requests are issued at the control of the BFM and it requires 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.x) 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 ports may not arrive in the order set by the cycle stamp tagging of each message. The SCE-MI 1.1 prohibits this scenario from happening and thus this topic will need to be reconsidered.

Batching of messages modeling implications – Given that multiple messages may be batched into a output buffer, the 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 override the previous message rather than queue the messages for subsequent processing. SCE-MI doesn't mandate any different behavior today, this some guidance need to be added that would recommend to the transactor modeler to queue messages coming out on output port for further processing. This would be more natural if reactivity control will be directly supported at the SCE-MI infrastructure level.

Lack of native 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.

- 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 Cadence proposed enhancements to SCE-MI.
- 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 (and not the notion of cycle stamp). SCE-MI doesn't provide standard means for converting cycle stamp information to absolute time natively viewed by the user. This can be addressed however by HVL language specific adaptor as described latter.
- 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 method for turning recording ON and OFF should be used to build portable transactors across implementations.

Lack of mixed signal/transaction level support – traditional co-simulation was based on signal level interaction between the hardware side and the software side. SCE-MI 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. SCE-MI doesn't define if the above mixed signal/transaction use model is supported and leaves the decision whether to implement this capability to each vendor.

Lack of 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 accelerated mode, SCE-MI doesn't mandate the view of common time across the partitions. Contrasting this to simulation, common time is supported in simulation 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).

Besides the use model implications, some of the software side partition of the transactor (to be named proxy model) may use timed constructs while others do not. This will cause portability issues for the proxy model part of the transactor across SCE-MI implementations if SCE-MI would continue to assume that the software side is untimed.

Clearly defining alternating vs. concurrent use model – SCE-MI today doesn't make any assumptions if the HW/SW side is running concurrently or in an alternating mode. 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 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, uses of batching or streaming of messages. The use of concurrent use is independent of whether the messages are of small/large size, fixed/variable size or delivered individually/ in a batch or as part of a long stream of messages.

Lack of minimal transactor architecture requirements – SCE-MI 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 the partitions. As a matter of fact, SCE-MI even defines the BFM on the hw side as the 'transactor' as opposed to what third party IP vendors would view as best practices. This lack of architectural definition may discourage building common transactors by IP vendors as the functionally on the HW side would be considered highly limiting and ambiguity is left on the SW side.

Furthermore, lack of transactor architecture definition also impedes on building scalable transactors models where the proxy models could be developed in various HVLS such as SystemC/C/C++/Vera/Specman/ 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 scalable transactors serving different segments of the market.

As a minimum some guidelines should be defined to suggest that a transactor would contain hw and sw side partitions with indications to scalability of the architecture to support all common HVLS on the sw side in addition already mentioned Verilog/VHDL support on the hw side.

Lack of 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. SCE-MI is not even a fully object oriented interface for C++ models.

To maintain language neutrality on one hand, while making the API easy to use for native HVL modeling, a native SW API in the native HVL language could be defined that encapsulates the SCE-MI SW side API. 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.

Lack of 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 that will block sending the message until the BFM requests it, it could 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 semantics doesn't mandate any relationships between IsReady and send.

Given that many 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 addition, the SCE-MI spec semantics could be tightened to insure a consistent behavior of a blocking interface wrt IsReady/Send semantics that all transactor will comply with.

Last, the use of blocking interface entails the use of threading package. SCE-MI 1.x 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 IsReady and Send() and Receive() under the hood but uses more native HVL constructs for transaction level communication.

Lack of SCE-MI spec classification across target users – While SCE-MI spec (section 4.3) defines three categories of users, it doesn't follow 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 resolved 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.