Qualifying precision of abstract SystemC models using the SystemC Verification Standard

Franco Carbognani[1], Christopher K. Lennard[2], C. Norris Ip[3], Allan Cochrane[2], Paul Bates[2]
[1]Cadence European Labs, [2]ARM Ltd, [3]Cadence Design Systems

Abstract

The increasing complexity of Systems on Chip (SoC) has introduced the need for abstract executable specifications (models) covering both hardware and embedded software. The new capabilities of SystemC 2.0, such as those added for transaction-based communication and test-bench Specification and monitoring, facilitate this SoC modeling. However, an obstacle to the adoption of abstract modeling as standard design practice is the lack of well establishes methodologies for the assessment of model precision. We describe such a methodology based on the SystemC Verification Standard implemented by Cadence's TestBuilder-SC. This methodology enables comparison of high-level (transaction level) SoC models in SystemC against implementation RTL models.

An application of the methodology is presented, based on the AMBA Class Library (ACL) for SystemC being developed by ARM in collaboration with EDA partners. The key elements of the methodology are:

1. *A completely reusable testbench that can be used for simulation and verification of the design at both high-level (transaction level) of abstraction and RTL implementation level.*
2. *A single database format is used so that data collected from simulations at each level can easily be processed and compared*

We present an example of effective validation of ARM PrimeXsys-platform IP components against their RTL implementation.

1 Introduction

One of the greatest pressures on the IP provider today is that of providing fast, and highly accurate models. There are many models of varying speed and accuracy that exist as part of EDA design-support packages, but only two system-level that have gained industry trust: (1) Untimed programmers-models (e.g., ARMulator), and (2) cycle-callable, cycle-accurate models (e.g., ARM Cycle-Callable Models). The former is necessary for soft prototypes. Soft prototyping enables SW development to commence prior to the finalization of SoC architectural decisions. These models are certifiably accurate in context in that they assume system-communication will be performed correctly, and offer register accurate visibility into the system (sufficient for application developers). The latter model type is required for SoC co-verification providing sufficient speed for HW, driver and OS validation with certifiable timing accuracy.

There is, however, a large gap between co-verification effective clocking speeds (low 100k cycles per second) and soft-prototype models (mid 10M cycle per second). In this gap fall two critical tasks: hardware-dependent middle-ware development, and HW/SW corner-case testing. Design models that target this gap suffered a serious flaw to industrial adoption: they trade simulation speed for simulation accuracy, but that accuracy is not quantified. Adoption of these models necessitates that a qualification process for their accuracy be defined. This paper addresses this issue.

2 Qualification of abstract system-level models

The delivery of an architectural platform necessitates packaging of the connected HW components and ported OS with guaranteed reference methodology to Si implementation. However, system-houses are finding that ESW development for a product is now taking longer than HW development. This is exacerbated by the TTM driven platform environment. Consequently, ESW development must commence prior to finalization of the HW design, and this ESW development often requires more HW and timing dependent detail than a soft-prototyping environment provides. Furthermore, during the validation phase, the comprehensiveness of corner-case or regression-based

testing benefits from increased simulation speed. Transaction based verification methods, and improved hierarchical strategies for full system verification and validation are opening the market to abstract system-models. SoC validation encompass three major steps: Level 1 Test - exhaustive checking of the connectivity and memory access, cycle and pin accurate; Level 2 Test - HW corner-case testing of communication paths in the SoC, including interrupt handling – cycle accurate with single 'ports' for data/address word and control-state transfer; and Level 3 Test: HW/SW corner-case testing with full OS – cycle approximate with abstract data and control passing. Level 1 Test involves RTL or RTC models, Level 2 involves transaction-based cycle-callable models with abstract memory descriptions, and Level 3 involves block-transfer models with abstract memory descriptions.

Both the support of HW-dependent SW authoring, and comprehensive SoC validation, require fast-but-close-to-timing-accurate system models: a move from Level 2 to Level 3 operational abstraction.

There are two issues associated in the move for Level 2 to Level 3 models, the first is guaranteeing of 'sufficient' timing accuracy of the model when instantiated in a particular system, the second is the recognition of functional accuracy in the sense of correct execution against properties rather than cycle-accurate timing.

Relating to the first issue, in isolation a timing-approximate component model for Level 3 style validation and development is not quantifiable. The reason for this is inability to statistically guarantee adequate spread-over-time of the exercising the any particular operational condition approximated by the model. That is, over a particular performance benchmark suite, it may be shown that a component model has a Y% timing inaccuracy, but this increases to an, e.g., 5xY%, under particular operation. However, with the embedding of the component within a market-targetted platform, the relevance of the performance benchmarking of a model can be assured and modeling accuracy quantified [3]. This allows the designer to choose their desired simulation-speed against a statistically guaranteed timing-accuracy trade-off.

Relating to the second issue, for Level 3 design and validation, verification principles must move away from cycle-based comparison of register or signal values. Instead, this necessitates a move to transaction-based specification of operational principles. This ARM example of this paper, and the ACL supported by that is a Level 2 type model which is cycle-accurate but uses transaction-based communication. However, the ACL may be utilized in the support of Level 3 models due to its handling of burst transfers. The model qualification and validation concepts described extend to Level 3 models.

3 The Performance Validation Methodology

The validation methodology we are presenting will be focusing on the assessment of timing accuracy of transaction level models (TLM) and will be termed as performance validation.

The two key elements of our methodology are:

1. Creation of effective reusable test benches that can be used for simulation and verification for designs at both high-level (transaction-level) and RTL implementation level.
2. Creation of databases with transaction level timing information from simulation of designs at both high-level and RTL implementation level for effective comparison of timing characteristics.

The concept of testbench reuse is not new, but its realization is often difficult particularly when applied to the validation of equivalent models written using different abstractions. If, however, there are certain features common to both models and it is sufficient to test both models within this subset, testbench reuse becomes possible. We have found that performance validation falls within this subset. For the same reasons, the metrics of interest in performance validation, span both modeling domains. This allows the output of the same information to the same database and the use a common set of post processing techniques.

If the TLM model has an RTL counterpart that has been validated, the RTL model can be used as the golden representation to which to compare the TLM model. At the TLM level the testbench will produce test stimulus in the form of transactions. In order to keep a unique stimulus generator, the TLM testbench must be able to interface to the RTL environment through a transaction-level interface, and be able to output transaction to the same database used in the TLM domain. We found that a combination of TestBuilder-SC (implementing the SystemC Verification Standard) and Transaction Explorer (TxE) provided all the right features to enable this flow:

TestBuilder-SC integrates with the NC-Sim simulator and provides the facilities for writing transaction level testbenches for RTL models. TestBuilder's Transaction Verification Modules (TVMs) provide a transaction-level interface to various buses. They translate a C++ bus transaction to the pin-level signaling required to complete the transaction through hardware.

The automatic transaction recording facilities of the TVMs output transaction to the SST2 database format. TxE can then be used to construct queries on the database to derive the average latency information needed for validation and comparison of models. Figure 1 illustrates how all of these tools fit into the validation flow.

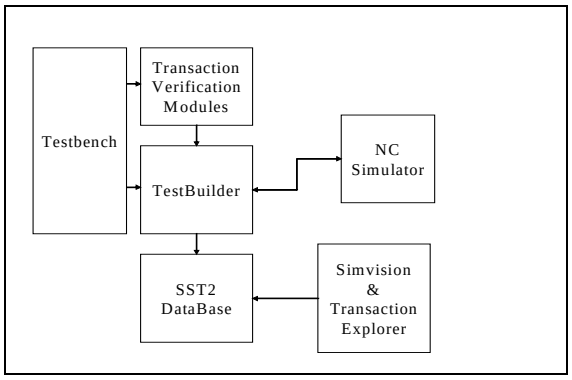

Figure 1: Cadence Tools Flow

4 The SystemC Verification Standard

The SystemC 2,0 Standard [5][6] and the SystemC Verification [7] provide a sufficiently flexible environment to execute the methodology described in Section 3. SystemC 2.0 was standardized in 2001 and has been incorporated into NC-Sim for simulation that takes both SystemC description and Verilog/VHDL description. The new SystemC Verification Standard proposed by Cadence in early 2002 has been accepted by the SystemC Steering Committee in September 2002. In the remainder of this section, we provide a summary of the SystemC Verification Standard, and describe how it enables reusable test benches. The creation of transaction-level database will be described in Section 5.5.

The SystemC Verification Standard version 1.0 contains the following features:

- data introspection : manipulation of arbitrary data type through the scv_extensions_if abstract interface and the scv_smart_ptr template.
- constrained and weighted randomization : creation of Boolean constraints through the scv_expression class, and discrete distribution through the scv_bag class.
- transaction recording : recording of transactions into a database through the scv_tr_stream class and the scv_tr_generator template.
- miscellaneous features such as sparse arrays, HDL connection, exception handling, and debugging interface.

Using Transaction Verification Modules (TVMs), the same test can be used for simulation of designs at both high-level and RTL implementation level. A Master TVM transaction level interface could be defined as:

```
class mybus_master_task_if : virtual
public sc_interface {
  public:
    virtual void write_tx
      ( bus_data_t& data ) = 0;
    virtual unsigned int  *read_tx
      ( bus_data_t& data ) = 0;
    virtual void write_tx
      ( const bus_data_h data_h ) = 0;
    virtual unsigned int  *read_tx
      ( bus_data_h  data_h ) = 0;
};
```

A test for such interface can be implemented as:

```
class test : public sc_module {
public:
  sc_port<mybus_master_task_if> tvm;
  SC_CTOR(test) { SC_THREAD(body); }
  void body() {
    scv_smart_ptr<bus_data_t> ctrl;
    ctrl->next();
    tvm1->write_tx(…,
      ctrl->read(), …);
    …
  }
};
```

This test contains a port with the abstract Master TVM interface. As a result, this test can be used for the TLM description, by using a TVM that will interface to the TLM model, and also for the RTL design description, by using a TVM that will interface to the RTL implementation.

In this specific case the TLM Master TVM will be basically an “adapter” between TestBuilder-SC transaction class and native AMBA bus transaction class, but more in general the concept of TLM TVMs has been introduced so that they implement the complete interface of existing RTL TVMs and allow the complete reuse of existing TestBuilder Testbenches.

Figure 2 shows the resulting validation flow. Both the TLM and RTL TVMs output transactions to the SST2 database. More specifically TLM TVMs record transaction using the methodology detailed in Section 5.5.

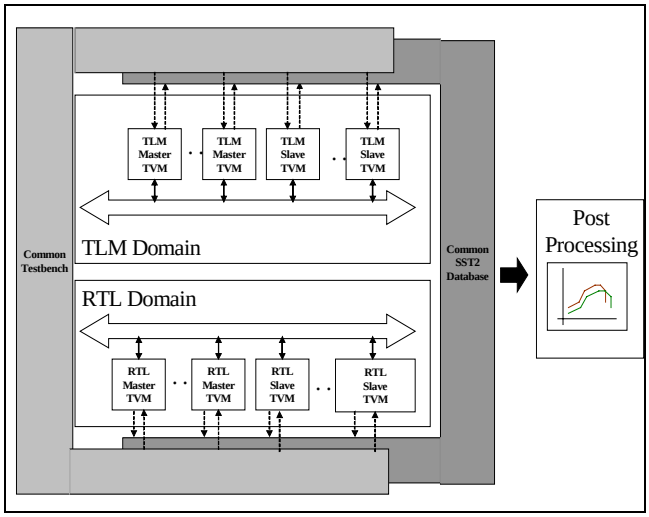

Figure 2: Performance Validation Flow

5 The AMBA Class Library

While ARM has a history in CPU core modeling, the support of platforms with complex multi core system simulation requires fast, accurate behavioral models for the peripherals and AMBA bus. There is an increasing need for a design environment that meets the needs of newer platform-based designs. In particular, we need an environment that:

- brings hardware and software together early in the design cycle
- provides a full System-on-Chip (SoC) design environment including support for multi-core
- delivers a consistent validation environment such that we have a continuous route from capturing design intent to validating design implementation.

SystemC provides ARM with the de facto industry-standard needed for behavioral modeling. The C-like structure of the language is perfect for functional modeling, and its position as a standard ensures that SystemC models are easily adopted and reused across the industry (an essential property for any IP development). SystemC, with its support of C++ class libraries and operator overloading also provides a basis for encapsulating functional hierarchy, fundamental to the emerging transaction-based verification strategies. ARM has developed an AMBA Class Library (the "ACL"), which is a model of the MultiLayer AMBA fabric. The model supports transaction level communication but in a way that allows accurate modeling of pipelined processors and to system level models of peripheral blocks.

5.1 Bus Model

The bus model that has been developed is ARM's MultiLayer AMBA fabric. The protocol supported by this fabric is AHBlite; The AHBlite protocol is very similar to the standard AHB 2.0 protocol except that split/retries are not available and slaves use HREADY to control the bus when the transfer ends.

The release of SystemC version 2.0 allowed higher-level approaches to bus modeling to be taken. This is documented in [3]. These new high level modeling approaches were adopted in the ACL design

The bus model developed was designed to be cycle accurate. By this we mean that at the rising edge of the clock, the signals on the bus would match those in the RTL. Although the ACL does not use signals to represent bus transactions they can be derived from the state of the bus at any time. Cycle accurate modeling reduces the problems of verifying the ACL and the models attached to it against the RTL verification suites.

With this in mind three means of accessing the bus are offered:

- Non-blocking access
- Blocking access
- Debug or back-door access

5.2 Non blocking access

Bus masters that want to be very accurate modeling AHB pipeline require non-blocking access to the bus. Here follows an example of a non-blocking master interface that could constitute part of the final ACL:

```
class AMBA_nonblocking_interface
: virtual public sc_interface {
public:
  virtual AMBA_burst_status
  nonblocking_burst(
    const AMBA_input_stage* const
master,
    AMBA_addr_t address,
    AMBA_data_t const data,
    AHB_lite_control AHB_control,
    AMBA_burst_control burst_control
  ) = 0;
};
```

Matching the pipelined address and data phases of the AHBlite protocol requires multiple calls to the above method. A bus master starts a burst with an initial call to nonblocking_burst() to set up the burst data structures and request access to the bus. The master then makes repeated calls to nonblocking_burst() to check on the status of the individual transfers within the burst. The bus then calls the slave at each one of these calls to nonblocking_burst(), the slave returns a value indicating that the current transfer is complete, waiting or an error has occurred. This value is then passed back to the master by the bus. The master must repeat the call to nonblocking_burst() at each clock cycle until the slave has completed all the transfers or has aborted the burst and returned an error. These subsequent calls equate to the address and data phases within the transfer.

By making bus transaction calls at the burst level, rather than the individual transfer level, it is expected that bus and peripheral models can be optimized for simulation efficiency without sacrificing cycle accuracy. Furthermore it is also expected that models may be able to offer the user the ability to forego 100% cycle accuracy whilst gaining significant simulation performance.

5.3 Blocking access

The blocking access allows masters to simply model bus access functions. The only drawback is that the master cannot normally re-access the bus without incurring a single

cycle delay. This can be overcome by slightly more sophisticated strategies on the part of the master but then due to the programming effort required it will be easier to use the non-blocking approach. The blocking interface is implemented internally by the bus using the non-blocking interface.

5.4 Debug Access

Finally, debug accesses allow things like debuggers, visualisers, etc. quick access to the contents of peripherals. These objects do not exist in a real system so they should not cause bus activity when reading or writing data to objects.

5.5 Embedding Transaction Recording into the AMBA class library

An important component of our methodology is the generation of databases with transaction-level timing information. Instead of using simple logging statements throughout the description, we use the transaction recording features in the new SystemC Verification Standard. Because we would like to give the user as much control over how recording should be done as possible, we adapted a style in which related transaction recording codes are captured in several recorder interfaces. For example the recorder interface associated with the nonblocking master interface presented above is defined as follows:

```
class nb_recorder_interface
: virtual public sc_interface {
public:
  scv_tr_generator
    <AMBA_addr_t, AMBA_data_t> &
  get_nb_tx_generator() = 0;
  …
};
```

This recorder interface becomes the contract between the master and the fabric. The template scv_tr_generator is a template in the new SystemC Verification Standard for generating a transaction of a specific type, with attribute types identified by the two template arguments. As long as the master provides a recorder implementing this interface, the bus fabric can generate appropriate transactions into the database. An example implementation of the nonblocking burst operation is shown below, which generates a transaction through the begin_transaction() and end_transaction() calls.

```
AMBA_burst_status
AMBA_RTL_bus_fabric::nonblocking_burst(
    const AMBA_input_stage* const master,
    AMBA_addr_t address,
    AMBA_data_t const data,
    AHB_lite_control AHB_control,
    AMBA_burst_control burst_control
  ) {
  …
  scv_tr_handle h = master->get_recorder()
    ->get_nb_tx_generator()
    .begin_transaction(address);
  …
  h.end_transaction(data);
}
```

This style provides the tests and the masters the flexibility to customize transaction recording for different simulation. For example, a straightforward recorder can be implemented as follows:

```
class basic_recorder
: public nb_recorder_interface {
  bool _on;
  scv_tr_stream tx_stream;
  scv_tr_generator
    <AMBA_addr_t, AMBA_data_t> nb_tx;
  scv_tr_generator
    <AMBA_addr_t, AMBA_data_t> dummy;
public:
  basic_recorder()
  : _on(false),
    tx_stream("basic","recorder"),
    nb_tx("nb", &tx_stream,
          "addr","data"),
    dummy() {}
  scv_tr_generator
    <AMBA_addr_t, AMBA_data_t>&
    get_nb_tx_generator() {
    if (_on) return nb_tx;
    else return dummy;
  }
  void set(bool on) { _on = on; }
};
```

This basic recorder creates a transaction stream by instantiating scv_tr_stream from the Verification Standard. A valid generator is then created with respect to this stream, and a dummy generator is created also. This basic recorder can be turned on and off by calling the method set(). When it is on, the implementation of the abstract interface returns a valid generator so that the bus fabric will be recording onto the appropriate transaction stream.

A test can instantiate this recorder and attach it to masters. Each master can have a dedicated recorder, or multiple master can share one, depending on the purpose of the simulation. The main goals we would like to achieve using this style of coding are precise controls over which databases, transaction streams, and generators are active and how they are shared among multiple modules. We have also partitioned the recording code into its own class to minimize the impact on the implementation of the actual design functionality in various levels of abstractions.

6 The Performance Validation Case Study

The performance validation methodology just described was applied to the validation of some of the PrimeXsys-platform IP blocks. Our case study was purposely kept very simple since we were more interested in bringing together

the complete flow of the proposed methodology then to a complex application of it. On the RTL side the system consists of the Verilog AHB Bus Fabric and Memory models from the ARM Micropack and two instantiations of the Cadence AMBA Master TVM. The resulting system is depicted in figure 3.

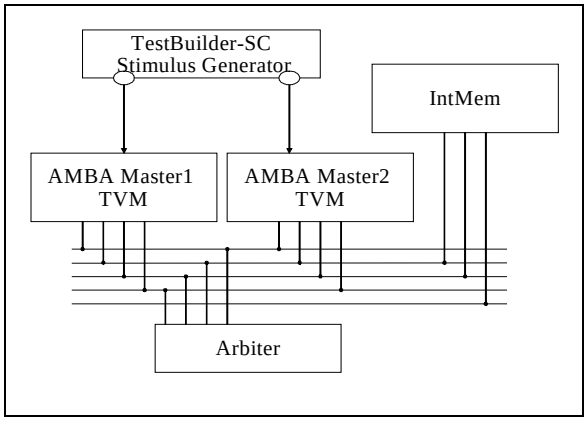

Figure 3: RTL Test Environment

On the TLM side the system is composed by the AHB Bus fabric from an early prototype version of the ACL, a memory block and two instantiations of the TLM Master TVM. The resulting system is depicted in figure 4.

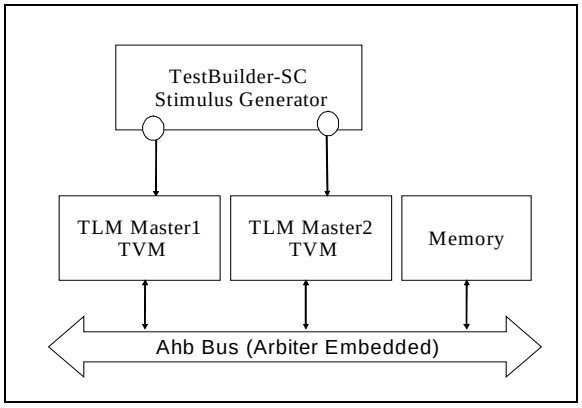

Figure 4: TLM Test Environment

6.1 Comparison of TLM against RTL simulations

We have defined a simple stimulus generator that is specifying a scenario in which the two Masters are accessing the bus one after the other with SINGLE-BURST. A total of 10,000 transactions were submitted and recorded into the SST2 database in both the TLM and RTL studies. A TxE script was written to process the averaged, maximum and minimum transactions duration for the two Masters. Those values were then compared between TLM and RTL simulations and difference calculated in percent. Results are shown in Table 1.

Table 1: Comparison results before TLM model modification

AvgRtl1	MinRtl1	MaxRtl1	AvgRtl2	MinRtl2	MaxRtl2
69.0ns	69.0ns	69.0ns	29.0ns	29.0ns	29.0ns
AvgTlm1	MinTlm1	MaxTlm1	AvgTlm2	MinTlm2	MaxTlm2
30.0ns	30.0ns	30.0ns	30.0ns	30.0ns	30.0ns
AvgErr	MinErr	MaxErr	TotErr		
29.9%	29.9%	29.9%	29.9%		

Looking in more details at the RTL simulation it was immediately clear that the resulted nearly 30% error was related to the fact that the Master2 (corresponding to ARM core) is selected as default granted master leading to different timing in the arbitration. With little modification on the TLM model side, in order to take in account of this behavior, the new comparison results on Table 2 show an error reduced to 2.4%.

Table 2: Comparison results after TLM model modification

AvgRtl1	MinRtl1	MaxRtl1	AvgRtl2	MinRtl2	MaxRtl2
69.0ns	69.0ns	69.0ns	29.0ns	29.0ns	29.0ns
AvgTlm1	MinTlm1	MaxTlm1	AvgTlm2	MinTlm2	MaxTlm2
70.0ns	70.0ns	70.0ns	30.0ns	30.0ns	30.0ns
AvgErr	MinErr	MaxErr	TotErr		
2.4%	2.4%	2.4%	2.4%		

7 Conclusion and future work

ARM is intending to support SystemC models compliant to an ACL supportive of both cycle-callable transaction level models, and cycle-approximate models. These models offer the speed and timing accuracy needed for HW-dependent SW development, as well as full-system validation with comprehensive HW/SW corner-case analysis. These strategies are fundamentally based around model and validation support of TLM methodologies.

We have shown in our work that our validation flow based on SystemC Verification Standard can effectively help in enabling ARM strategies. More sophisticated stimulus generators will be defined in order to address the problem of test coverage and we do intend to enhance the presented methodology via it's integration within the Cadence Verification Reuse Methodology (VRM).

8 References

[1] C. Norris, Stuart Swan, *Using Transaction-Based Verification in SystemC*, http://www.systemc.org/, June 2002

[2] Jon Connell, Bruce Johnson, *Early HW/SW Integration Using SystemC v2.0* , http://www.systemc.org/, Embedded Systems Conference - San Francisco, June 2002

[3] Christopher K. Lennard, Eric Granata: *The Meta-Methods: Managing design risk during IP selection and integration.* European IP 99 Conference - Edinburgh, November 1999

[4] Alain Clouard, Giorgio Mastrorocco, Franco Carbognani, Antoine Perrin, *Towards Bridging the Precision Gap between SoC Transactional and Cycle-Accurate Levels*, Design, Automation and Test in Europe conference - Munich, March 2002

[5] SystemC Stuart Swan, An Introduction to System Level Modeling in SystemC 2.0, white paper, www.SystemC.org.

[6] Thorsten Grotker, Stan Liao, Grant Martin, and Stuart Swan, System Design with SystemC, Kluwer Academic Publishers, 2002.

[7] The SystemC Verification Standard, version 1.0a, approved by the SystemC Verification Working Group and the Steering Committee, to appear at www.SystemC.org.

[8] C. Norris Ip and Stuart Swan, A Tutorial Introduction on the New SystemC Verification Standard, submitted for publication, September 2002.

[9] C. Norris Ip, Simulation Coverage Enhancement Using Test Stimulus Transformation, ICCAD 2000