Accurate Performance Exploration of System-on-Chip using TLM

Master of Science Thesis in System-on-Chip Design

by

Farooq Khalid Chuhtai

chughtai@kth.se

Supervisor

PhD Henrik Svensson (Ericsson AB)

Examiner

Prof. Ahmed Hemani (KTH)
Abstract

Increased complexity of system-on-chips (SoC) makes performance exploration with register transfer level (RTL) models to be both time consuming and to appear too late in the design cycle. Instead of RTL, transaction level models (TLM) have emerged as a dominant candidate for modeling of these complex SoCs. Transaction level models are abstracting implementation details and are therefore less complex to implement and execute much faster than RTL. However, an open issue is to define and measure how accurate TLM models are compared to RTL. In this thesis, accuracy of a TLM model is defined and quantified by introducing metrics that are used to compare RTL and TLM models. The metrics introduced are transaction time, start time, relative throughput and transaction reordering. From the metrics, an error between RTL and TLM is calculated and presented. The thesis discusses how to generate proper stimuli for models, collect the metric, and how to calculate and represent the error to the designer. The proposed methodology for accuracy measurement is using TLM2 definitions and may therefore be applied to any TLM model. This methodology is applied to experiment a multi-port memory that is modeled as an approximately timed TLM. Defined metrics are collected from the same scenarios applied to both RTL and TLM models. Accuracy measurements at system and sub-system level can be integrated with the verification environment to systematically develop and refine accurate TLM models for performance exploration.
Contents

Abstract ......................................................................................................................... iii
Contents ......................................................................................................................... iii
Acknowledgement ........................................................................................................ vi
List of Figures .............................................................................................................. vii
List of Tables ................................................................................................................ ix

1. Introduction ............................................................................................................... 1
   1.1 Thesis Contribution .......................................................................................... 1
   1.2 Related Work .................................................................................................. 2
   1.3 Report Structure ............................................................................................. 2

2. Performance Exploration ......................................................................................... 3
   2.1 Architectural Exploration of SoC .................................................................... 3
   2.1.1 Abstraction Level ...................................................................................... 3
   2.1.2 System Level Estimations ......................................................................... 4
   2.2 SystemC and Transaction Level Modeling (TLM) ......................................... 6
      2.2.1 Why SystemC/TLM ............................................................................. 7
      2.2.2 Why TLM 2.0 ..................................................................................... 8
      2.2.3 Typical use cases of TLM .................................................................... 9
      2.2.4 Level of Abstraction ........................................................................... 9
   2.3 Architectural Exploration through TLM ........................................................... 10

3. Metrics for Accuracy Measurements ...................................................................... 11
   3.1 Introduction ..................................................................................................... 11
   3.2 Metrics for Accuracy measurements ................................................................ 11
      3.2.1 Transaction Duration ........................................................................... 11
      3.2.2 Start Time ............................................................................................ 15
      3.2.3 Throughput .......................................................................................... 16
      3.2.4 Transaction Order ............................................................................... 17
   3.3 System Accuracy ............................................................................................. 18
   3.4 Sub-system analysis ......................................................................................... 18
   3.5 Accuracy driven design methodology ............................................................. 19

4. TLM Model ............................................................................................................... 22
   4.1 Introduction ..................................................................................................... 22
   4.2 TLM model of a Multi-Port Memory ............................................................... 22
      4.2.1 Loosely Timed Model ......................................................................... 22
      4.2.2 Approximately Timed Model ............................................................... 23

5. Experimental Setup ................................................................................................. 25
   5.1 Accuracy Measurement Environment ............................................................... 25
      5.1.1 Ideal Environment ................................................................................. 25
      5.1.2 Accuracy Measurement Environment Used ........................................... 25
      5.1.3 TLM/RTL Logs ..................................................................................... 26
   5.2 TLM Traffic Generators ................................................................................... 27
      5.2.1 Base Protocol Mapping ....................................................................... 27
      5.2.2 Top Module .......................................................................................... 28
5.3 Input Stimuli .................................................................................................................. 28
   5.3.1 Test Vectors ................................................................................................. 28
   5.3.2 No. of Transactions ....................................................................................... 30

6. Results and Analysis ........................................................................................................ 31
   6.1 TLM accuracy Analysis .................................................................................... 31
      6.1.1 Transaction Duration ............................................................................... 31
      6.1.2 Start Time ................................................................................................. 36
      6.1.3 Throughput ............................................................................................... 38
      6.1.4 Transaction Order ................................................................................... 38
   6.2 System Accuracy .................................................................................................... 41
   6.3 Sub-system Analysis ............................................................................................. 42
   6.4 Accuracy driven design methodology ............................................................... 43
   6.5 Simulation Time Comparison ............................................................................. 43

7.1 Conclusion .................................................................................................................... 46

7.2 Future Work ............................................................................................................... 47

Bibliography .................................................................................................................... 48
Acknowledgement

Performing my master’s thesis at Digital ASIC dept of Ericsson AB was a significant milestone in my life. During the course of the project I learned and gained much from a lot of talented people. I made many good friends too. In general, I am thankful to everyone who was involved with the project but there are some names that clearly stand out.

Most importantly, I would like to express my sincere gratitude for my supervisor at Ericsson, PhD Henrik Svensson who was always available with good advice and rich ideas. I am grateful for his patience, mentorship and motivation.

I am highly indebted to my manager, Pierre Rohdin, for creating a wonderful environment to work and for ably handling all the management of the research project.

I wish to express my appreciations for the support I received from Göran Knutson and his team, Frederic Pouyet, Mihai Stoian, Michael Carlsson and my fellow thesis workers. This team work made my thesis all the more enjoyable.

From my university, I would like to express greatest regard for my academic supervisor Mohammad Badawi and my examiner Prof. Ahmed Hemani.

In the end I am thankful to all my friends, my family members and most importantly my parents for their support, care and appreciation through thick and thin times during the whole thesis work.

Stockholm,

December 8, 2011
List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT</td>
<td>Approximately Timed</td>
</tr>
<tr>
<td>DUT</td>
<td>Design under Test</td>
</tr>
<tr>
<td>EDA</td>
<td>Electronic Design Automation</td>
</tr>
<tr>
<td>ERM</td>
<td>eVC Reuse Methodology</td>
</tr>
<tr>
<td>ESL</td>
<td>Electronic System Level</td>
</tr>
<tr>
<td>eVC</td>
<td>e Verification Component</td>
</tr>
<tr>
<td>HW</td>
<td>Hardware</td>
</tr>
<tr>
<td>IC</td>
<td>Integrated Circuit</td>
</tr>
<tr>
<td>IEEE</td>
<td>Institute of Electrical and Electronics Engineers</td>
</tr>
<tr>
<td>LT</td>
<td>Loosely Timed</td>
</tr>
<tr>
<td>OSCI</td>
<td>Open System C Initiative</td>
</tr>
<tr>
<td>OVM</td>
<td>Open Verification Methodology</td>
</tr>
<tr>
<td>RTL</td>
<td>Register Transfer Level</td>
</tr>
<tr>
<td>SoC</td>
<td>System on Chip</td>
</tr>
<tr>
<td>SW</td>
<td>Software</td>
</tr>
<tr>
<td>TLM</td>
<td>Transaction Level Modeling</td>
</tr>
<tr>
<td>UVM</td>
<td>Universal Verification Methodology</td>
</tr>
<tr>
<td>VHDL</td>
<td>VHSIC Hardware Descriptive Language</td>
</tr>
</tbody>
</table>
List of Figures

| Figure 2.1 | Gajski-Kuhn Y-chart (source [5]) | .......................................................... | 4 |
| Figure 2.2 | Design Space width versus abstraction level | .......................................................... | 5 |
| Figure 2.3 | Accuracy and simulation Speed trade-off (source [2]) | .................................................. | 5 |
| Figure 2.4 | SystemC language architecture (source [9]) | .................................................. | 6 |
| Figure 2.5 | TLM 2.0 Classes | .................................................. | 7 |
| Figure 2.6 | TLM based design flow [10] | .......................................................... | 8 |
| Figure 3.1 | Transaction duration of 40 ns | .......................................................... | 12 |
| Figure 3.2 | Base Protocol Phases | .......................................................... | 12 |
| Figure 3.3 | Example of Transaction start and end points | .................................................. | 13 |
| Figure 3.4 | Relative Error in transaction durations | .................................................. | 14 |
| Figure 3.5 | Differences in start timings for TLM and RTL transactions | ........................................ | 15 |
| Figure 3.6 | Multiport memory attached to 4 DSPs | .................................................. | 19 |
| Figure 3.7 | Contemporary design flow | .................................................. | 20 |
| Figure 3.8 | Proposed design flow | .................................................. | 20 |
| Figure 4.1 | Transaction Duration | .......................................................... | 23 |
| Figure 4.2 | TLM Model of a Multiport Memory | .......................................................... | 24 |
| Figure 5.1 | Accuracy Measurement Environment | .......................................................... | 25 |
| Figure 5.2 | Experimental Setup for accuracy measurement | .......................................................... | 26 |
| Figure 5.4 | Mapping between RTL and TLM transaction | .................................................. | 27 |
| Figure 5.5 | Top module for the experiments | .......................................................... | 28 |
| Figure 5.6 | A general test case | .......................................................... | 29 |
| Figure 5.7 | Test case selection | .......................................................... | 29 |
| Figure 6.1 | Histogram of transaction duration error for test1 | .................................................. | 33 |
| Figure 6.2 | Histogram of transaction duration error for test3 | .................................................. | 33 |
| Figure 6.3 | Histogram of transaction duration error for test3 and test5 | .................................................. | 34 |
| Figure 6.4 | Mean Time of Flight per port and its standard deviation | .................................................. | 35 |
| Figure 6.5 | Min and Max value per port | .......................................................... | 35 |
| Figure 6.6 | Error in start timings for all TLM transactions for all tests | .................................................. | 37 |
| Figure 6.7 | A measure of the disordering of transactions between TLM and RTL | .................................................. | 39 |
| Figure 6.8 | Transaction Reordering per port | .......................................................... | 40 |
| Figure 6.9 | Combined Error metric | .......................................................... | 41 |
| Figure 6.10 | Varying region of interest for different DSPs | .................................................. | 42 |
| Figure 6.11 | Accuracy driven design methodology | .......................................................... | 43 |
| Figure 6.12 | Example of timing information shown on the specman profiler | .................................................. | 44 |
List of Tables

Table 2.1 Use cases in TLM modelling (source [9]) ................................................................. 9
Table 3.1 An Example of Global Transaction order ................................................................. 17
Table 3.2 An Example of Max System disorder ......................................................................... 18
Table 5.1 Snapshot of RTL/TLM log files ................................................................................. 26
Table 5.2 List of test cases with their characteristics ................................................................. 30
Table 6.1 Mean relative, Mean absolute, Max, Min and Std Dev of the difference in
transaction durations between TLM and RTL ......................................................................... 32
Table 6.2 Average and relative difference in the start time of transactions ................................. 36
Table 6.3 Relative throughput error .......................................................................................... 38
Table 6.4 Measure of disordering of transactions for applied test cases ................................. 39
Table 6.5 Combined Error Metric ............................................................................................ 41
Table 6.6 CPU time comparison ............................................................................................... 45
Chapter 1

1. Introduction

In 1965, co-founder of Intel, Gordon E. Moore presented a paper suggesting that the number of transistors in an integrated circuit (IC) doubles approximately every two years [1]. This law has proved to hold up till now and has been the guiding principle in planning and research in the semi-conductor industry. This suggests that modern electronic devices and applications running on them are getting increasingly complex and demanding. The system designers face challenges to reduce the size, power and performance requirements of modern system-on-chip (SoC). Development of modern SoCs require greater control on complexity of the system. System architects face tough questions regarding the properties and number of components used for designing a complex system. They would like to experiment around these aspects to find the best mix or to identify the bottlenecks in the system.

In the realm of performance exploration, an executable language is required to model SoCs at a higher abstraction level as compared to RTL. By increasing the abstraction level, system architects can skip implementation details and keep enough temporal details to correspond to the performance shown by RTL.

Transaction Level Modeling (TLM) is increasingly used as a representation of hardware in a higher abstraction. TLM models are executable and their simulation performance/speed is magnitudes higher than the RTL. Since TLM models do not implement all design details, they can be implemented much earlier than RTL. Thus performance exploration results can be acquired much earlier in the design flow.

The challenge of increased complexity and early performance exploration has been well answered by the introduction of TLM. However, higher abstraction level means less accuracy which can be a limiting factor in performance exploration and very well undermine the advantages of using TLM. General requirement in performance exploration is to implement TLM models with sufficient accuracy. The goal of this thesis is to present a measurement for accuracy between RTL and TLM.

This thesis presents a systematic approach to quantify accuracy of any TLM model by collecting a set of proposed metrics. Accuracy estimates can be utilized as acceptance criteria for TLM models, i.e. if they are accurate enough to be used in performance exploration.

1.1 Thesis Contribution

The major contribution of this thesis is presentation of a methodology for accuracy quantification, applicable to any TLM 2.0 compliant model. The methodology involves experimental setup, selection of input stimuli and a set of metrics for accuracy and error measurements. The results and analysis presented in this report are reflections of accuracy measurement experiments performed at approximately timed model of a multiport memory. Same input is applied to RTL and TLM models. Metrics collected represent error in transaction duration, transaction start time, long term throughput and transaction order between RTL and TLM.
It is proposed that error measurements made through metrics collected be used to systematically develop accurate TLM models for performance exploration. One option would be addition of accuracy metrics around the verification environment built around both TLM and RTL DUT. This will allow system architects to get coverage of system accuracy under different test scenarios. It can also be used to compare accuracy of two different TLM models representing same RTL model. Accuracy of the TLM models can be systematically improved till the models are accurate enough for performance exploration.

1.2 Related Work

It is well established that accuracy drops as designs move from lower abstraction level to higher. Abstraction means lack of details and this drop is justified. Quantification of TLM accuracy remains an open issue. In [2], trade-off between accuracy and simulation performance is explained. TLM models at varying levels of detail are used to perform error measurements in terms of latency calculation on individual transactions and the finish time of system when all transactions have finished. The presented model is tested with 5000 transactions of linearly varying size and delay to the next transaction.

A similar approach is followed in [3]. Reusable testbench is applied to both RTL and TLM models. Average, minimum and maximum transaction durations for 10,000 transactions are compared for error calculation.

This thesis not only utilizes the metrics used earlier for accuracy measurements but also contributes some new metrics to offer detailed analysis.

1.3 Report Structure

The report is organized in the following manner:

- **Chapter 1** introduces the overall thesis and the contributions made. Some related work is also presented.
- **Chapter 2** gives an in-depth background in the field of design exploration of SoC. It focuses mainly on the use of TLM for this purpose.
- **Chapter 3** presents the metrics used for accuracy measurement.
- **Chapter 4** elaborates the TLM model of a multiport memory used for performing accuracy measurement experiments.
- **Chapter 5** offers an insight into the experimental setup used for collecting results. It gives out details of the traffic generators and stimuli used.
- **Chapter 6** explains the findings of the applied experiments.
- **Chapter 7** deals with the future work and conclusion of the thesis.
Chapter 2

2. Performance Exploration

This chapter gives an in-sight into performance exploration which is part of architectural exploration. The limitation of RTL and the need to raise the abstraction level for architectural exploration is presented. TLM is introduced as promising abstraction for area and performance estimates on modern SoCs. The advantage with TLM is simplified HW/SW co-simulation, early implementation and fast simulation speed. However, these gains are made at the expense of accuracy. Generally, higher abstraction level means faster simulation and more inaccuracy is observed. For good performance exploration, trade-off between abstraction, simulation performance and accuracy is required.

2.1 Architectural Exploration of SoC

In the present era a massive demand of electronic devices is seen. Many electronic gadgets and devices are part of our everyday life now. The devices themselves are getting smaller, more computationally intensive, have strict requirement on lower power without any compromise on the performance. The applications running upon the devices are getting more demanding as well. They are benefitting from the advancements made in the semi-conductor technology.

This complexity has made it hard for system architects to perform architectural exploration on SoCs to find the best mix of:

- Power
- Area
- Performance

The contemporary process of architectural exploration itself is not automated and therefore error prone. System architects relay on their experience, simulation and spread sheets to perform exploration.

2.1.1 Abstraction Level

With the increased complexity in ICs an important question is the level of abstraction at which the hardware should be modeled. This question can be explained from the Y-chart presented by Gajski and Kuhn in 1983 [4].
A few decades ago, it was possible to describe a system at physical level. As the complexity grew, the systems were raised to gate level and later to register transfer level. By raising the abstraction level some details of the model can be abstracted away. Throughout the 1990s two well known languages for describing a system in RTL domain were Verilog and VHDL.

When the models started getting even more complex, there was a need for raising the abstraction level. The answer to the problem of increasing complexity was shifting to System Level. The idea was to abstract away the details of the lower level domains. This will result in early development of models since they are less complex. Early development of hardware models has immense benefits. They can be utilized by software developers to develop and test their software earlier. Verification engineers can start creating the verification environment earlier. System architects themselves can test many different scenarios before finalizing one design/system for further development.

2.1.2 System Level Estimations

Architectural exploration at system level presents more possibilities for system architects. For instance, if a router is being designed, different arbitration schemes can be modeled on a system level and the best one can be selected. It is unlikely that same principle will be applied in RTL. Even if different routing algorithms are implemented in RTL, they will cost considerable amount of overhead in terms of cost and effort spent over them. System level exploration offers a larger design space as compare to lower abstraction level.
Widened design space by increasing abstraction is a challenge in itself as well. Now system designers have to explore a large set of scenarios. As an example, if a choice has to be made between 4 processors, 2 arbitration policies, 4 memory configurations and a NOC or a bus for interconnect, $4 \times 2 \times 4 \times 2 = 64$ different systems have to be explored. Many different combinations can be tested because of high simulation speed in System level. This however, comes at the loss of accuracy [3]. The drop in accuracy and increase in simulation speed can be represented as in figure 2.3.

A system architect must strike the right balance between rising abstraction level, simulation performance and accuracy for successful performance exploration.
2.2 SystemC and Transaction Level Modeling (TLM)

For modeling hardware at system level, an executable language is required. System C has emerged as a major candidate. It is a C++ class library that has hardware like data types and structures. It is both a high level language as well as a hardware descriptive language [4]. In 2005, it became an IEEE standard [7] and OSCI (Open SystemC Initiative) presented the Language Reference Manual for providing unambiguous semantics for SystemC language [9]. Since then it has been one of the most popular languages in electronic system-level (ESL). A free for distribution simulator, for running SystemC models, is provided by OSCI. This consortium started back in 1999 backed by major players in the EDA companies like Coware and Synopsis.

<table>
<thead>
<tr>
<th>Methodology-Specific Libraries</th>
<th>Layered Libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master/Slave Library, etc.</td>
<td>Verification Library</td>
</tr>
<tr>
<td></td>
<td>Static Dataflow, etc</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Primitive Channels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal, Mutex, Semaphore, FIFO, etc</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Core Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modules</td>
</tr>
<tr>
<td>Ports</td>
</tr>
<tr>
<td>Processes</td>
</tr>
<tr>
<td>Channels</td>
</tr>
<tr>
<td>Events</td>
</tr>
<tr>
<td>Event-driven Simulation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-valued Logic type</td>
</tr>
<tr>
<td>4-valued Logic Vectors</td>
</tr>
<tr>
<td>Bits and Bit Vectors</td>
</tr>
<tr>
<td>Arbitrary Precision Integers</td>
</tr>
<tr>
<td>Fixed-point types</td>
</tr>
<tr>
<td>C++ user-defined types</td>
</tr>
</tbody>
</table>

| C++ Language Standard |

Figure 2.4 SystemC language architecture (source [9])

Transaction Level Modelling is a library built on top of SystemC which itself is a class library of C++. It encapsulates the communication between different hardware modules through system calls. The idea is to separate communication from implementation. OSCI has so far published two drafts for the Language Reference manual. For the purpose of this thesis, the latest draft TLM 2.0 has been utilized.

TLM 2.0 class library consists of core interfaces, initiator target sockets, base protocol, generic payload and utilities. In addition all the constructs of TLM 1.0 are also supported. Using these constructs, any hardware module can be modelled at higher abstraction level.
2.2.1 Why SystemC/TLM

The advantages for using TLM over RTL are:

1. Simulation speed
   Immense speed-up is observed when simulating in TLM as compared to RTL. Some estimates suggest 1000 times faster simulation speed [10].

2. Fast and early implementation
   As compared to RTL, the TLM models can be much simpler and easier to implement. A measure for that purpose is the lines of code comparison. The TLM models are simpler because they exclude much of the design details. In [6], the author reports an implementation speed up of nearly a factor of 10 when modeling TLM versus RTL.

3. Accuracy
   Experiments have shown that TLM models can be designed with sufficient amount of accuracy. With cycle accurate modeling in TLM, nearly 100% accuracy can be achieved. This is the reason why TLM models can be trusted and used for architectural exploration.

4. HW/SW co-design
   In conventional RTL based design flow, the SW developers have to wait till they have some hardware prototype on which they can run their application. Thus the SW and HW teams cannot work in parallel with each other. Since TLM in essence models hardware, systems developed in TLM can be used by SW groups for early software development.
In traditional design flow, software is implemented in C/C++ and hardware in VHDL/Verilog. In TLM based design flow both hardware and software can be implemented in C/C++.

5. Interoperability

Since TLM is an IEEE standard and the latest draft puts some basic requirements for modeling, IPs of various SoC modules can be obtained off-the-shelf and used in the design process. It results in shortened time to market. Many EDA companies are providing IP libraries for common SoC architectures like interconnect buses. Reusability and interoperability within both in-house and with 3rd party vendors is due to the standardization of TLM.

2.2.2 Why TLM 2.0

TLM 2.0 specifically models memory mapped systems. Main benefit of using TLM 2.0 is interoperability. This is achieved through:
- Standardized Request/Response
- Base Protocol
- Ignorable extension

All blocks communicate through *generic payload*. It is intended for, but not restricted to be used for modeling memory mapped systems. It includes many attributes of a typical memory mapped protocol such as command type, address, data length, streaming width, direct memory interface etc. Users can add extensions to generic payload attributes but they should be ignorable.
Base Protocol defines a set of timing phases TLM models follow. Any TLM model can have extended phases for better protocol mapping. However, like extensions to the generic payload, the extended phases should be ignorable. This implies that any set of TLM base protocol compliant models should communicate with each other irrespective of extensions made.

### 2.2.3 Typical use cases of TLM

Usually TLM is used for the purposes listed in table 2.1.

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Coding Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Application Development</td>
<td>Loosely-timed</td>
</tr>
<tr>
<td>Software Performance Analysis</td>
<td>Loosely-timed</td>
</tr>
<tr>
<td>Hardware Architectural Analysis</td>
<td>Loosely-timed or Approximately-timed</td>
</tr>
<tr>
<td>Hardware Performance Verification</td>
<td>Approximately-timed or Cycle Accurate</td>
</tr>
<tr>
<td>Hardware Functional Verification</td>
<td>Untimed, Loosely-timed or Approximately-timed</td>
</tr>
</tbody>
</table>

Table 2.1 Use cases in TLM modelling (source [9])

Since the focus of this these is performance exploration, approximately timed models are used. The various notions of timing with TLM models are explained in the next sub heading.

### 2.2.4 Level of Abstraction

Although TLM seem like one level of abstraction (ESL), there are more abstractions within this one level. In fact what abstraction level TLM follows is debatable within scientific community itself. Transaction level may refer to a set of abstraction levels with varying functional and timing details [11].

Various options of the temporal details are represented in different coding styles in TLM 2.0. They are:

- Untimed
- Loosely-timed
- Approximately-timed
- Cycle accurate

The untimed models do not have any notion of time. Processes run at a pre determined synchronization point.

Loosely-timed models have two timing points; transaction start and end points. Blocking calls are used for communication between modules. This coding style is mostly utilized for virtual platform development and software performance analysis.
Approximately-timed models have multiple timing points. Delays can be annotated by timed event notifications. Non-blocking function calls are used which have multiple phases and explicit timing points for phase transition. Performance exploration is a typical use case for this style of coding.

Cycle accurate models can be implemented by extending the phases and timing points in the AT models. The reason to do so could be a requirement of accurate protocol mapping from RTL to TLM. Cycle accuracy can be achieved through ignorable extensions. Since there can be many implementations for modeling hardware, cycle accurate modeling cannot be standardized.

2.3 Architectural Exploration through TLM

System architects are interested in finding out the performance and area of their design. A combination of static and simulation based analysis can be applied to find out the area and performance of the system respectively.

When TLM model are used for architectural exploration, it is important to implement the models in a way that they have configuration parameters. For example, in a memory based system, the common performance parameters can be read/write access delays, arbitration delays, number of ports, port widths etc. Performance exploration can be done by tuning parameters from the model before or during the simulation [12].

In OSCI itself a working group by the title “Configuration, Control & Inspection Working Group” is in place and working for the standardization of configurable parameters [8]. This will allow any tool to connect and configure the parameters. The research group has presented the requirement specification but has not currently finalized a draft. Standardization of configuration parameters will help covering the huge design space in ESL. This will also make design exploration interoperable.
Chapter 3

3. Metrics for Accuracy Measurements

In this chapter, a set of metrics for accuracy measurements is presented. A combination of them can be applied on different levels of system to measure sub-system or system accuracy of any TLM model. These accuracy metrics can be a part of the verification environment themselves to systematically calculate accuracy.

3.1 Introduction

TLM has answered the problem of increasing complexity of SoC design. However, accuracy measurements between RTL and TLM design remain an open issue. This thesis gives quantitative analysis for accuracy.

In the scope of performance exploration, designers are interested in “timing accuracy”. In this report, this will be referred as accuracy only. In [2] it is argued that accuracy drops as the designers perform modeling in higher abstraction levels. As the pin level details are abstracted away when design moves from RTL to TLM, this drop in accuracy makes sense and needs to be quantified.

3.2 Metrics for Accuracy measurements

Following metrics are used for accuracy quantification:

- Transaction duration
- Start time
- Throughput
- Transaction reordering

From these metrics error vectors are formulated. Since lower abstraction levels have more details available, they are considered as ‘golden reference’. Thus error vectors are calculated by comparing TLM model results with RTL model.

Each error metric can have the following representations:

- Mean
- Max
- Min
- Standard Deviation

Furthermore, the error vectors can be plotted in the form of histograms, plots over all time/transactions or plots for a window of time/transactions representing trends in error.

3.2.1 Transaction Duration

Blocks represented in TLM communicate through transactions. In TLM2 communication is standardized through generic payload. One fundamental metric for accuracy measurement is transaction duration. This has been applied in [2] and [3] as well. Transaction duration is the time between the start and end time of any transaction.
In a TLM model of an initiator and a target, if a transaction shown in figure 3.1 is made, both initiator and target have to agree to the start time and end time of the transaction for effective communication and handshaking. Thus 4 timing phases are required. In AT models, they are represented as:

- BEGIN_REQ
- END_REQ
- BEGIN_RESP
- END_RESP

These phases can also be shown as in figure 3.2:

The order of the phases has to be maintained. However, some intermediate phases can be skipped or completed implicitly [9].
Since TLM allows any initiator or target to finish any transaction early and not wait for the subsequent phases, the exact definitions of ‘transaction start’ and ‘transaction end time’ would depend how the designer models them and annotate timing information accordingly. As for an example consider figure 3.3.

![Diagram](image.png)

Figure 3.3 Example of Transaction start and end points

In this scenario, transaction starts when initiator sends out BEGIN_REQ and finishes when the target sends BEGIN_RESP. END_REQ and END_RESP are completed implicitly. There are other options available for defining transaction duration according to [9]. An example can be the time between BEGIN_REQ and TLM_ACCEPTED sent back on the return path.

Whatever may be the definitions of transaction start time and end time in TLM, the transaction duration is simply the time between them. For RTL and TLM, this time can be calculated by applying equations 3.1 and 3.2 respectively.

\[
Trans_{duration_{RTL}} = End_{time_{RTL}} - Start_{time_{RTL}}
\]  
(3.1)

\[
Trans_{duration_{TLM}} = End_{time_{TLM}} - Start_{time_{TLM}}
\]  
(3.2)

In order to quantify error between the two transaction durations, relative error between the two has to be taken into account.

\[
Trans_{duration\_error} = \frac{|Trans_{duration_{TLM}} - Trans_{duration_{RTL}}|}{Trans_{duration_{RTL}}}
\]  
(3.3)

Relative transaction duration error can be negative or positive. Absolute values are considered to estimate total error. For clarification, equation 3.3 is applied to the following pair of transactions.
In transaction pair N, $\text{Trans\_duration}_{\text{TLM}}$ is less than $\text{Trans\_duration}_{\text{RTL}}$. In that case a negative value for $\text{Trans\_duration}_{\text{TLM}} - \text{Trans\_duration}_{\text{RTL}}$ is observed. In N it is -2. However in M it is +14. $\text{Trans\_duration\_error}$ for N and M is -0.0625 and 0.4375 respectively. To calculate the total error absolute values are considered. This way whether the error is due to shortened TLM transactions or elongated ones, all cases are covered.

$\text{Trans\_duration\_error}$ is unit less. It can be represented in percentage as well.

$$\text{Trans\_duration\_error\%} = \text{Trans\_duration\_error} \times 100$$

Thus transaction pair N shows 6 % error and M shows 43 % error in figure 3.4. Error percentage can be greater than 100 % too. It is up to the system designer to define, how much error his model can afford. If error is exceeding the defined limits, the model may have to be redesigned.

Equation 3.3 gives error on a single TLM transaction. This can be accumulated over the whole simulation length. The mean can be calculated as:

$$\text{Mean\_trans\_duration\_error} = \frac{\sum_{n}^{\text{No\_of\_transactions}} \text{Trans\_duration\_error}}{\text{No\_of\_transactions}}$$

Mean error gives a global value for measure of error. It may be interesting to see how this error changes with time. Thus the designer can take mean value for a certain number of TLM transactions.
Performance exploration usually investigates the corner cases. Therefore it is practical to take the minimum and maximum value of the difference between $Trans\_duration_{TLM}$ and $Trans\_duration_{RTL}$. This may show both negative and positive values. Min shows the earliest time the TLM transaction will be completed. Same way Max will represent the latest time the TLM transaction finishes after exceeding the time of finish for its RTL counterpart. The standard deviation is a representation of how much deviation transactions duration error show from the mean value.

### 3.2.2 Start Time

Difference in start timing of TLM and RTL transactions is utilized to quantify the error between them. Analysis can also be performed at the finish time of transactions. Such measurements were performed by [2]. Cumulative analysis on either start time or finish time of transactions is expected to yield similar results if the individual transaction durations are not much different.

Transaction start time reflects the transaction duration as well. If relative transaction duration error is calculated from equation 3.3, the difference will be expressed as the difference in start time. If the relative difference in transaction duration is high, start time difference is expected to be high as well.

$$\text{Error}_{\text{Start\_time}} = \text{Start\_time}_{\text{RTL}} - \text{Start\_time}_{\text{TLM}}$$  \hspace{1cm} (3.5)
If the values of equation 3.5 are accumulated for a number of transactions it can be analyzed if the error in start time is increasing, decreasing or following a random pattern. If it is increasing or decreasing, it will be interesting to see if it follows a linear trend. If so, TLM model will be showing systematic error.

In order to present one value for a given system, mean difference for the start time error can be utilized. Equation 3.6 represents the average difference in start timings between RTL and TLM.

\[
Mean_{\text{error}}_{\text{Start time}} = \frac{\sum_{0}^{\text{No. of transactions}} |\text{Start time}_{\text{RTL}} - \text{Start time}_{\text{TLM}}|}{\text{No. of transactions}}
\]  

(3.6)

For making different metrics comparable, relative values are used. Thus relative start time difference is calculated by dividing the start time difference between RTL and TLM with the start time of RTL. This can be expressed as in equation 3.7.

\[
Error_{\text{relative}}_{\text{Start time}} = \frac{\text{Start time}_{\text{RTL}} - \text{Start time}_{\text{TLM}}}{\text{Start time}_{\text{RTL}}}
\]

(3.7)

Mean relative error in start time can be found as in equation 3.6.

### 3.2.3 Throughput

Throughput calculations are a classic way of comparing different systems. They can be applied for accuracy comparisons as well.

\[
Throughput = \frac{\sum_{0}^{\text{No. of transactions}} \text{Data}}{\text{Total time}}
\]

(3.8)

Using equation 3.8 throughput for both TLM and RTL model can be found. Relative throughput can be found by applying equation 3.9:

\[
Throughput_{\text{relative}} = \frac{\text{Throughput}_{\text{TLM}}}{\text{Throughput}_{\text{RTL}}}
\]

(3.9)

If the systems are very much same, relative throughput will be unity. The further it is away from 1, the greater is error in it. This principle can be used to estimate the accuracy of the system as in equation 3.10.

\[
Error_{\text{Throughput}} = |Throughput_{\text{relative}} - 1|
\]

(3.10)
3.2.4 Transaction Order

Transaction order is an important analysis between RTL and TLM models. Not only does it show the accuracy measurement but also the behavior of the two. There might be scenarios in which maintaining the right order of transactions may be more important than any other characteristic for example a point to point communication between a DSP filter and memory.

It is important to define how transactions are ordered. In a point to point connection, transaction ordering is straightforward. However, in a multiport module, where multiple transactions can originate at the same time, an order has to be assigned.

Transaction order can be seen in two ways:
- Port Transaction Order
- Global Transaction Order

Port Transaction Order is the order of transactions on each port.

Global Transaction Order is the order of how the transactions are seen globally on the system.

Order is assigned in terms of increasing simulation time. When there is a situation that different ports finish transactions at the same time, order can be assigned in terms of increasing port id number.

Consider the table 3.1 as an example.

<table>
<thead>
<tr>
<th>Global Order</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLM Port ID</td>
<td>21</td>
<td>23</td>
<td>4</td>
<td>12</td>
<td>56</td>
<td>0</td>
<td>22</td>
<td>21</td>
<td>3</td>
<td>22</td>
</tr>
<tr>
<td>RTL Port ID</td>
<td>21</td>
<td>56</td>
<td>22</td>
<td>12</td>
<td>0</td>
<td>21</td>
<td>4</td>
<td>23</td>
<td>22</td>
<td>3</td>
</tr>
<tr>
<td>Disorder</td>
<td>0</td>
<td>6</td>
<td>4</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3.1 An Example of Global Transaction order

In TLM transactions, the global order for port 21 is <1, 8>

In RTL transactions, the global order for port 21 is <1, 6>

Absolute difference in the transaction order can be taken to find Displacement for port 21. It is <0, 2>.

Summing up the Displacement for port 21 gives the Disorder for all transactions on port 21 which is 0+2 = 2 in this case. Disorder for all the ports and eventually for the whole system can be found out.

\[
Port\_disorder_i = \sum_{0}^{No.\_of\_transactions} |TLM\_port\_order_i - RTL\_port\_order_i|
\]  

(3.11)

Equation 3.11 presents the disorder among TLM and RTL transactions on one port. By summing it up over all the ports of the system, system disorder can be found as in equation 3.12.
\[ \text{System\_disorder} = \sum_{i=0}^{\text{No\_of\_ports}} \text{Port\_disorder}_i \]

(3.12)

To find the relative system disorder, the value for System\_disorder is normalized by the maximum disorder that can occur. It is the total reversal of global order. Consider the following example:

<table>
<thead>
<tr>
<th>Global Order</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLM Port ID</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>RTL Port ID</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Disorder</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 3.2 An Example of Max System disorder

In the example shown in table 3.2, TLM and RTL ports exhibit the maximum disorder. Dividing the system disorder by the maximum system disorder keeps the error value below 1.

\[ \text{System\_disorder\_relative} = \frac{\text{System\_disorder}}{\text{Max\_system\_disorder}} \]

(3.13)

### 3.3 System Accuracy

Individual accuracy metrics are combined to formulate a combined error metric.

\[ \text{Error}_{\text{all\_metric}} = \sum_{i=\text{metrics}} \zeta_i \text{Error}_i \]

(3.14)

Equation 3.14 presents error for all 4 metrics discussed. Summation can have equal weights by putting same constant value for all error metrics. It can also have different weights for different error metrics to emphasize the effect of error measurement from different metrics. Error measurements from all metrics can further be combined for all test cases to present error estimation for the whole system under application of all test cases.

\[ \text{Error}_{\text{system}} = \sum_{\text{Test\_cases}} \zeta Error_{\text{all\_metric}} \]

(3.15)

### 3.4 Sub-system analysis

In a performance exploration scenario, accuracy comparison at the right level is important. Metrics for accuracy measurements can be applied at different levels of TLM model. They can be combined to present a global estimation of error between TLM and RTL models.

Consider the simple example shown in figure 3.6.
Suppose this system as a whole shows a `transaction_duration_error` of 30%. However, if the designer performs the accuracy measurements per port, it may be the situation that port 1 shows an error of 40% while the rest of the ports show errors not exceeding 5%. By dividing the system into sub systems, designer can isolate the problematic port and explore further within the device attached to port 1. Thus the system accuracy will improve.

### 3.5 Accuracy driven design methodology

In the contemporary design flow, following steps are followed:

1. A functional model is created based on the design specifications. This model usually omits any timing details. Mostly a high level language like C/C++ is used for functional modeling.
2. Timing is added to the model using SystemC/TLM constructs. Initially, a loosely-timed (LT) model is created followed by approximately timed (AT) model.
3. The LT and AT models can be used for functional verification and performance exploration respectively. LT model is used for software development.
4. Once the designer/system architect is satisfied with the performance of the TLM design under test (DUT), it can be implemented in RTL and verified for functional correctness and performance. This is followed by other subsequent steps in ASIC design flow.

Figure 3.7 explains the above steps. The test environment can be any verification methodology such as ERM, UVM or OVM.
It is proposed that accuracy measure metrics, explained in this chapter, be incorporated in the verification environment. In addition to them, a reference TLM model may also be included. This will result in an automated process of accuracy measurement between RTL and reference TLM model. The accuracy may be tested against varied input stimuli applied to both RTL and TLM. This way system architects can get coverage of test scenarios in which either model will perform with too much inaccuracy or error. Based on this coverage, system architect will also find the scenarios in which his system will perform optimally.
This methodology can also be applied in situations when system designers are looking to make next generation SoCs from already existing setup in RTL. First step in such a scenario would be to model existing RTL in TLM with sufficient accuracy. Accuracy metrics can help in this classification. Once the required accuracy is achieved, system architects can explore performance in TLM before reverting back to RTL.
Chapter 4

4. TLM Model

This chapter explains the TLM model used for experimentation. It is an AT model of a multiport memory. Some guidelines for implementing an RTL model in LT and AT model are presented from the experiences gained over the multiport memory.

4.1 Introduction

TLM can be used to model different hardware modules at varying levels of details referred as coding styles in TLM 2.0 language reference manual [9]. Each coding style is targeted for specific purpose. Untimed models are used for functional reference. Loosely timed models are utilized for early software development. Approximately timed and cycle accurate models are typically utilized for performance exploration. Thus it is important to identify the purpose of TLM models before modeling them in any coding style.

4.2 TLM model of a Multi-Port Memory

When modeling any hardware in TLM, it is recommended to start from simple structure and add details to it later. Thus, even if the purpose is to perform performance exploration on an AT model, it will be practical to start with LT model. LT and AT model has same structure, only the functional calls are changed from blocking to non-blocking. Often it is useful to implement both function calls in one TLM model so that it can be used in either style.

4.2.1 Loosely Timed Model

LT models are typically used for software development. Major concern in LT models is register-accuracy. In a memory model, register-accuracy means after any memory operation, contents transferred to or from the memory are correct and at the right place. Timing of their transfer is not a concern. Software developers need to model constructs that will affect software performance. For instance semaphores are often used to gain access to a shared resource. Similarly, model may have different operational modes like debug mode or supervision mode. Direct memory interface is often allowed at this level. Such details need to be incorporated in the LT model.

LT models use blocking transport calls for communication. These calls may return immediately or after some simulation time. The waiting time can be specified in the blocking transport call. It can also be annotated by calling wait statements within the function call. Since there are very few timing details available in this style of coding, it is not an ideal choice for performance exploration.
4.2.2 Approximately Timed Model

Since focus of this thesis is performance exploration, correct timing annotation is important. Approximately timed model, which uses non-blocking function calls, can have good timing annotation and is commonly used for performance exploration. When a read or write transaction is initiated from any device attached to the memory, it goes through the following processes.

- Arbitration
- Scheduling
- Read/write Process

TLM model for performance exploration has to take care of only these functions. Other features are not important and can be abstracted away. In a multiport memory, different attached devices can initiate memory operations at the same time. These initiators have to be served in a defined order. This is done through arbitration. Once arbitration is completed, the next phase is to schedule the memory operation. It depends upon the present and previous state of the memory address space. Based on that state, the start time of any memory transaction can be scheduled. When a transaction has gone through the arbitration and it is estimated when it will be scheduled, the actual time spent reading or writing is calculated by the amount of data involved in the operation.

![Transaction Duration Diagram](image)

The advantage of TLM models in the simplicity of implementation of details. For example, arbitration function may be implemented in RTL through many processes. In TLM, it may take just one process. This allows designers to experiment with different arbitration policies.

Similarly, TLM model may not implement the exact structure as long as the timings are accurately met. An example can be memory banking. In RTL memory model, address space may be divided in separate blocks or banks. In TLM memory model, address space can be implemented in one big chunk and the scheduler has to ensure that any transaction is accessing its requested address space at the right timings.

Address space of memory module implemented in TLM is usually byte addressable. If the RTL memory is half-word or even word addressable, designer should be careful when implementing the same in TLM.
Figure 4.2 represents the TLM model of a multiport memory. It can be attached to any number of hardware devices represented in TLM. The port width of TLM sockets correspond to the bus width in RTL.

As a designer, options for exploration would be no. of ports, bus width, arbitration delays, read/write delays, memory size, pipeline depths, etc. In TLM model, these parameters should be implemented as constructor arguments. This will allow designer to experiments with different options for these parameters. They may be changed before or during simulation. System’s performance profile can be drawn for all the different combinations of such aspects.
Chapter 5

5. Experimental Setup

This chapter describes the overall test environment created to measure accuracy of TLM models. Implementation of traffic generators utilizing information logged from RTL verification environment is discussed. Characteristics of different test cases are also presented.

5.1 Accuracy Measurement Environment

When measuring accuracy of any TLM model, it is required to apply same input stimuli to both RTL and TLM. By doing so and collecting accuracy metrics on RTL and TLM models, good estimates of TLM accuracy can be made. Accurate TLM models will present best estimates if performance exploration in conducted on them.

5.1.1 Ideal Environment

Ideally, accuracy measurement environment should look like figure 5.1:

![Figure 5.1 Accuracy Measurement Environment](image)

Verification environment can be in any methodology. Usually it is ERM, UVM or OVM. The DUT can be replaceable. The verification environment should have functionality to support both RTL and TLM DUT.

5.1.2 Accuracy Measurement Environment Used

For this thesis, verification environment for TLM models was not available. Thus test vectors from RTL's environment were collected and utilized in TLM traffic generators connected with TLM DUT. The whole process can be depicted as in figure 5.2.
5.1.3 TLM/RTL Logs

Logs saved in text files from TLM and RTL have similar format. It is depicted in table 5.1.

<table>
<thead>
<tr>
<th>Port Id</th>
<th>Transaction Start Time</th>
<th>Transaction End Time</th>
<th>Read/Write Operation</th>
<th>Memory address</th>
<th>Data size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>21</td>
<td>39</td>
<td>1</td>
<td>4bcc5d</td>
<td>4</td>
</tr>
<tr>
<td>24</td>
<td>21</td>
<td>67</td>
<td>0</td>
<td>1867d2</td>
<td>56</td>
</tr>
<tr>
<td>75</td>
<td>36</td>
<td>189</td>
<td>1</td>
<td>42641f</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>57</td>
<td>167</td>
<td>1</td>
<td>4429fa</td>
<td>632</td>
</tr>
</tbody>
</table>

Table 5.1 Snapshot of RTL/TLM log files
For each transaction, this information is logged on each row in the same order on a text file. Both text files from TLM and RTL model can be fed to any software which can handle large amount of numeric data and perform accuracy measurements on them. Matlab is a good example and is used in this thesis. Other candidates can be ‘Stata’ or ‘R’.

5.2 TLM Traffic Generators

An important feature of accuracy measurement methodology is to create TLM traffic generators which mimic RTL in such a manner that RTL input specification can be mapped to TLM 2.0 Base Protocol. By doing so, any pair of RTL and TLM model can be compared.

5.2.1 Base Protocol Mapping

Start and end time of TLM transactions represented in Base Protocol phases have to be mapped to triggering of certain events in RTL. In RTL of the multiport memory used in this thesis, a new transaction is initiated when \texttt{start\_rqst} signal is asserted. Similarly, a transaction is finished when \texttt{rqst\_done} is set high. This marks the start and end time on a transaction in RTL. This time is logged in the log files as well. The start time is mapped to the \texttt{BEGIN\_REQ} from initiator. End time is mapped to \texttt{BEGIN\_RESP} from target.

![Diagram of mapping between RTL and TLM transaction](image.png)

Figure 5.4 Mapping between RTL and TLM transaction
5.2.2 Top Module

Top module in TLM connects the traffic generators feeding on data logged from RTL to a multiport memory. This is shown in figure 5.5.

Each traffic generator or initiator has 1 simple initiator socket. Multiple initiators can be connected to the same DUT by utilizing multi-pass through target socket. This gives possibility to the system architect to test with as many attached devices as he wants through a configuration parameter.

5.3 Input Stimuli

This section discusses the input stimuli applied for accuracy comparisons. Both TLM and RTL models are applied the same input stimuli. Accuracy is very much dependent upon the applied input stimuli. Thus selecting the right test cases is vital.

5.3.1 Test Vectors

In the experiments conducted for this thesis, transactions performing read and write operations on the memory model where generated from the traffic generators. As the table 5.1 suggests, each transaction is destined to or from a specific device attached to the memory. This is inferred from port id. Since the traffic is generated for a memory controller, the transactions can only be read or write operations. Each transaction targets a particular address of the memory and is of a certain length. These aspects along with the timings of the transactions are logged in the text files as well.
Figure 5.6 shows 4 transactions of equal length in terms of data size and same timing gap among them. This forms the very basic test vector. Variations can be made by changing the following two parameters.

1. Individual Transaction length
2. Time interval between two successive transactions

Accordingly for any change in these two aspects, different accuracy is measured. This can be elaborated as follows:

Individual transaction length and gap between two transactions make up the two axes on the graph. If another aspect of the input transactions is altered for the experiments, it will be represented in another axis of the graph. System architect must have some specifications regarding the minimum and maximum transaction length and the gap between individual transactions. This makes a two dimensional area on the graph shown in figure 5.7. It is referred as Region of Interest. All points within this region will have some error estimation. This will be shown on the third axis of the graph.
Ideally, test vectors should be generated to cover the whole region of interest. This may not be practically possible because of time and memory limitations. In that case, systems designers will have to resort to directed test vectors. Usually corner cases are of much interest as the system is most vulnerable on them and may show the worst performance.

By varying the two axes of figure 5.7, test vectors generated for experiments are listed in Table 5.2.

<table>
<thead>
<tr>
<th>Test case name</th>
<th>Delay between two successive transactions in ns</th>
<th>Length of individual transactions in half-words</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Test2</td>
<td>2-202</td>
<td>4</td>
</tr>
<tr>
<td>Test3</td>
<td>2-202</td>
<td>4-128</td>
</tr>
<tr>
<td>Test4</td>
<td>2-202</td>
<td>4-1024</td>
</tr>
<tr>
<td>Test5</td>
<td>2-202</td>
<td>4-2048</td>
</tr>
<tr>
<td>Test6</td>
<td>202-2002</td>
<td>4-1024</td>
</tr>
</tbody>
</table>

Table 5.2 List of test cases with their characteristics

Test1 is a back to back stress test with the smallest size of data length. The data length is gradually increased from test2 to test5 keeping within the same gap limits between the transactions. Test5 has data lengths between 4 to 2048 half-words and delay between 2 to 202 ns. This means that test5 also covers scenarios in which data length is between 4 to 128 and 4 to 1024 half-words. However the number of iterations for all tests is the same. Thus the scenarios created within test1 till test4 will not be fully covered by test5. In effect if test5 is run for many iterations, there is no need to run the previous tests. On figure 5.7, test5 will be represented as a large area made up of areas covered by the test1 to test4.

5.3.2 No. of Transactions

Ideally, any system architect will be interested in testing his models for as many transactions as possible. However, there are practical limitations. Accuracy measurements require both TLM and RTL simulations. TLM models simulate magnitudes faster than RTL models but logging data from RTL verification environment is a time consuming process and forms a bottleneck in the accuracy comparison. A compromise between test collection time and enough data for statistical study has to be decided.

In [2] and [3], 5000 and 10,000 transactions are run on TLM model respectively. For RTL DUT used in this thesis, if 10,000 transactions are sent per port, it takes about 1 day to collect enough statistics to test TLM DUT. On the other hand if 1,000 transactions are sent per port, 5-6 hours are required. Multiport memory modelled for this thesis has 64 ports available which can be connected to any hardware devise. 1000 transactions per port make about 64,000 transactions for TLM model accuracy measurement.
Chapter 6

6. Results and Analysis

This part of report presents results of test cases applied to TLM model and measured accuracy with respect to defined metrics. After collecting different accuracy metrics individually, cumulated accuracy for TLM model is measured. Accuracy can be measured for all metrics and all the test cases. This quantifies error for the whole TLM model.

6.1 TLM accuracy Analysis

Accuracy metrics can be applied one by one on TLM model. Results collected can be combined to represent system accuracy as explained in the later part of this chapter.

6.1.1 Transaction Duration

From the data logged in both RTL and TLM model, transaction durations can be measured. If both models are 100% accurate, their transaction durations will be same. If this is not the case, relative error in transaction duration can be found using:

\[
\text{Trans\_duration\_error} = \frac{|\text{Trans\_duration}_{\text{TLM}} - \text{Trans\_duration}_{\text{RTL}}|}{\text{Trans\_duration}_{\text{RTL}}}
\]

This equation will give the relative error per transaction which can be collected for many transactions. From this information, system designers can estimate the average, minimum and maximum difference in transaction durations. Furthermore, it will be useful to measure the standard deviation to know how much TLM transactions deviate from RTL. These measurements will give designers a global perspective of their design and can be used to compare different TLM implementations of same RTL design.

Results calculated from transaction duration metrics are represented in table 6.1. Except mean absolute transaction duration error, all values are relative and thus unit less. The maximum and minimum values show the earliest and latest time taken by TLM transactions to finish.
<table>
<thead>
<tr>
<th>Test Case</th>
<th>Mean relative transaction duration error (%)</th>
<th>Mean absolute transaction duration error (ns)</th>
<th>Min transaction duration error (%)</th>
<th>Max transaction duration error (%)</th>
<th>Std Dev in transaction duration error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test1</td>
<td>10.86</td>
<td>13.186</td>
<td>-30.0</td>
<td>93.75</td>
<td>14.22</td>
</tr>
<tr>
<td>Test2</td>
<td>41.24</td>
<td>14.9596</td>
<td>-80.43</td>
<td>450.0</td>
<td>51.62</td>
</tr>
<tr>
<td>Test3</td>
<td>22.24</td>
<td>15.0275</td>
<td>-79.59</td>
<td>180.0</td>
<td>22.09</td>
</tr>
<tr>
<td>Test4</td>
<td>10.61</td>
<td>23.4269</td>
<td>-86.57</td>
<td>330.0</td>
<td>13.22</td>
</tr>
<tr>
<td>Test5</td>
<td>08.52</td>
<td>31.5394</td>
<td>-88.79</td>
<td>308.33</td>
<td>13.47</td>
</tr>
<tr>
<td>Test6</td>
<td>07.00</td>
<td>13.6776</td>
<td>-100.0</td>
<td>66.67</td>
<td>9.73</td>
</tr>
</tbody>
</table>

Table 6.1 Mean relative, Mean absolute, Max, Min and Std Dev of the difference in transaction durations between TLM and RTL

Test2 clearly shows worst performance at nearly 41% error. It has very small transaction lengths but large gap between two successive transactions. Keeping the same gap and increasing the transaction length as in test2 to test5, an improved accuracy is seen with the mean error dropping from 41% to just 8%. However, this is an improvement in relative error. Mean absolute error increases as transaction lengths are increased. Increasing the gap between transactions further increases the accuracy.

A system architect will be interested not only in the mean accuracy of the system but also the worst case scenarios. Min and max transaction duration error represent the corner cases in an individual test. To see how frequently corner cases show up, histogram is a good representation.
Figure 6.1 show how the error in transaction duration timing is spread.

Figure 6.2 Histogram of transaction duration error for test3
Figure 6.3  Histogram of transaction duration error for test3 and test5

If the mean relative error in transaction duration for test3 and test5 is plotted on top of each other as in figure 6.3, it can be observed that most of the transactions in test5 are aligned on zero. Transactions from test3 have a larger standard deviation and are thus more spread out. Thus test3 have more relative transaction duration error as compared to test5. Test5 on the other hand has worse min and max values.

An important point to ponder is the fact that the experiments and error measurements are concerned with the transaction duration difference on a system level. In this thesis, all the memory ports are behaving in the same fashion. All 64 traffic generator produce uniform distribution of traffic. If the traffic generators are to be configured to behave in a non-uniform manner for example mimic different traffic patterns for different devices attached to memory ports, statistics collected on some ports will differ significantly from those collected on the other ports. In that case, analysis can be made per port.
From the figure 6.4 it can be seen that all ports behave similarly. All have the same traffic pattern. They are only slightly different because of the randomization effects. Similarly, the deviation from the mean value is very much the same for all the ports.

Min and max values can be plotted per port as well. Such a case will be useful if some ports are sending small transactions while the others are sending long transactions. Since all port follow uniform distribution, only global min and max are important.

In conclusion transaction duration error is an important metric for TLM and RTL accuracy quantification. Error observed is very much dependent upon the input stimuli and it can be represented in different forms as histogram, plots over windows of time or transaction etc.
6.1.2 Start Time

In the experiments conducted for this thesis, a systematic error is observed. It accumulates over time and will grow as the simulations run longer.

In order to measure the difference in start time between any TLM and RTL transaction, the following equation is used.

\[ \text{Error}_{\text{start time}} = \text{Start time}_{\text{RTL}} - \text{Start time}_{\text{TLM}} \]

This difference can be collected for all transactions on all ports for all test cases. Similarly, mean difference in start time can also be calculated for all test cases.

\[ \text{Mean error}_{\text{start time}} = \frac{\sum_{\text{No. of transactions}} |\text{Start time}_{\text{RTL}} - \text{Start time}_{\text{TLM}}|}{\text{No. of transactions}} \]

In order to make this metric comparable with other accuracy measurement metrics, relative start time error is required.

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Mean Start Time Error (ns)</th>
<th>Mean Relative Start Time Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test1</td>
<td>12.4987</td>
<td>0.069</td>
</tr>
<tr>
<td>Test2</td>
<td>3.8420 \times 10^3</td>
<td>5.48</td>
</tr>
<tr>
<td>Test3</td>
<td>6.2402 \times 10^3</td>
<td>7.42</td>
</tr>
<tr>
<td>Test4</td>
<td>9.5732 \times 10^3</td>
<td>4.73</td>
</tr>
<tr>
<td>Test5</td>
<td>1.2230 \times 10^4</td>
<td>3.66</td>
</tr>
<tr>
<td>Test6</td>
<td>6.3329 \times 10^3</td>
<td>0.90</td>
</tr>
</tbody>
</table>

Table 6.2  Average and relative difference in the start time of transactions

Table 6.2 clearly shows an increase in mean start time error as the transaction length is increased. This can be correlated with table 6.1 showing transaction duration error. As the absolute error in transaction duration is increasing, start time error is increasing as well. The effect of larger transactions causing large start time error is normalized by taking relative values.
Figure 6.6  Error in start timings for all TLM transactions for all tests

Test1 does not show much error in start timings because all transactions are back to back and much small in length. Error seen is due to the slack left by transaction duration error. Since the transaction duration error is small for test1, the start time error is small as well. Test5 and test4 show maximum error in start timings because of large error in transaction durations.

It can be concluded that TLM models exhibit systematic error in start timings. This error is larger for transactions having large error in transaction durations.
6.1.3 Throughput

Applying throughput measurements on TLM and RTL model, results observed are listed in table 6.3.

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Throughput TLM Mbps</th>
<th>Throughput RTL Mbps</th>
<th>Relative Error throughput (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test1</td>
<td>$7.999 \times 10^3$</td>
<td>$7.9976 \times 10^3$</td>
<td>0.02</td>
</tr>
<tr>
<td>Test2</td>
<td>$3.0403 \times 10^4$</td>
<td>$2.8608 \times 10^4$</td>
<td>6.27</td>
</tr>
<tr>
<td>Test3</td>
<td>$4.2151 \times 10^5$</td>
<td>$3.9129 \times 10^5$</td>
<td>7.72</td>
</tr>
<tr>
<td>Test4</td>
<td>$1.3340 \times 10^6$</td>
<td>$1.2725 \times 10^6$</td>
<td>4.84</td>
</tr>
<tr>
<td>Test5</td>
<td>$1.5328 \times 10^6$</td>
<td>$1.5372 \times 10^6$</td>
<td>2.86</td>
</tr>
<tr>
<td>Test6</td>
<td>$3.7325 \times 10^5$</td>
<td>$3.6974 \times 10^5$</td>
<td>0.95</td>
</tr>
</tbody>
</table>

Table 6.3 Relative throughput error

Test1 shows very little error in relative throughput. An explanation for this behavior is the absence of any start time error in it. From test2 to test5, a drop in relative throughput error can be seen. It is because for the same gap between successive transactions, data length is increasing. Thus throughput of the model is increasing.

In relation to the transaction duration error, results for relative throughput error are similar. TLM model shows worst performance under test2 and test3 and same can be observed here.

In conclusion, it can be stated that TLM models show higher accuracy if error in the relative throughput is small.

6.1.4 Transaction Order

Port transaction order does not show any disorder due to the design’s protocol followed by TLM model used in this experiment. Any port cannot issue or queue a new transaction until the previous transaction is served.

Global transaction order shows some amount of disorder. Collecting this metric on all transactions on all ports, results observed for system disorder are listed in table 6.4. In relation to the maximum possible system disorder, the global transaction order is not very large. This is the reason why relative system disorder shows small values in table 6.4.
Table 6.4  Measure of disordered transactions for applied test cases

<table>
<thead>
<tr>
<th>Test Case</th>
<th>System Disorder</th>
<th>Relative System Disorder (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test1</td>
<td>318452</td>
<td>0.0155</td>
</tr>
<tr>
<td>Test2</td>
<td>730050</td>
<td>0.0356</td>
</tr>
<tr>
<td>Test3</td>
<td>5729395</td>
<td>0.2798</td>
</tr>
<tr>
<td>Test4</td>
<td>3923615</td>
<td>0.1916</td>
</tr>
<tr>
<td>Test5</td>
<td>3553899</td>
<td>0.1735</td>
</tr>
<tr>
<td>Test6</td>
<td>547453</td>
<td>0.0267</td>
</tr>
</tbody>
</table>

Figure 6.7  A measure of the disordering of transactions between TLM and RTL

To observe how individual ports show reordering on a global level, transaction disorder measurements can be made on port to port basis. This analysis highlights ports showing maximum amount of disorder on a global scale.
In figure 6.8, Y-axis for test1 and test6 is scaled down for better presentation. It also manifests little disorder as compared to other test cases.

From the analysis of transaction ordering, it can be concluded that a TLM model is much accurate to RTL model if the system_disorder is as small as possible.
6.2 System Accuracy

All metrics can be combined to present a combined error representation under a certain test. In order to combine different metrics, relative values are taken.

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Mean Transaction duration error (%)</th>
<th>Mean Start Time Error (%)</th>
<th>Relative Throughput error (%)</th>
<th>System Disorder Relative (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test1</td>
<td>10.86</td>
<td>0.0690</td>
<td>0.02</td>
<td>0.015</td>
</tr>
<tr>
<td>Test2</td>
<td>41.24</td>
<td>5.48</td>
<td>6.09</td>
<td>0.035</td>
</tr>
<tr>
<td>Test3</td>
<td>22.24</td>
<td>7.42</td>
<td>7.72</td>
<td>0.279</td>
</tr>
<tr>
<td>Test4</td>
<td>10.61</td>
<td>4.73</td>
<td>4.74</td>
<td>0.191</td>
</tr>
<tr>
<td>Test5</td>
<td>8.52</td>
<td>3.66</td>
<td>3.55</td>
<td>0.173</td>
</tr>
<tr>
<td>Test6</td>
<td>7.00</td>
<td>0.90</td>
<td>0.89</td>
<td>0.026</td>
</tr>
</tbody>
</table>

Table 6.5 Combined Error Metric

All entries in table 6.5 represent error in percentage. They can be linearly combined with equal weights to measure error for all metrics for one particular test case. The problem lies with the fact that not all the metrics have equal impact on system accuracy. For instance, relative system disorder is very small compared to other error metrics. It will not have a large impact on system accuracy if equal weights are assigned to all the error metrics. System designer should decide the significance of each metric and assign weights accordingly.
Figure 6.9 shows error measurements on the application of different test vectors. TLM model shows varying accuracy under varying input stimuli. Thus as a system designer, it is important to mark out the region of interest for input selection and test the model against meaningful test cases.

Error for all metrics under one test can be combined with error for all metrics under another test to form the overall system error. Using these combined system error measurements, system designers can assign one error count to one TLM model. This allows different models to be easily compared.

6.3 Sub-system Analysis

When measuring the accuracy of a memory controller attached to different devices, it is important to measure the accuracy of the memory controller at the right system hierarchy. Thus sub-system analysis should be applied rather than a global system analysis. This also means that input stimuli should be selected at a sub-system level.

The regions of interest from where different input vectors are generated may be overlapping or separate. If test vectors from one big region representing a global system analysis are applied, system designer may get unrealistic error measurements.

![Figure 6.10 Varying region of interest for different DSPs](image-url)
6.4 Accuracy driven design methodology

If error metrics are an integral part of the verification environment, error measurements can be utilized to systematically improve accuracy of any TLM model.

![Combined Error %](image)

Figure 6.11 Accuracy driven design methodology

Figure 6.11 shows the effect of improvement in TLM model. If a new version of any model shows linear improvement for all test vectors, it reflects that the system specifications were not correctly mapped in the earlier TLM model. This way, designers may keep improving models till they are accurate enough to be utilized for performance exploration.

6.5 Simulation Time Comparison

With all the RTL and TLM accuracy comparison presented earlier, it will be useful to present the comparison between simulation time as well. One important reason why there is a shift to ESL from RTL is the small simulation time in ESL as compared to RTL. What can be simulated in days in RTL can take merely a few minutes in TLM based simulation. The ideal goal is to combine the accuracy of RTL with the speed-up TLM models provide [13].

RTL simulation environment for this thesis consists of both RTL and e verification components. The e components are used to generate test cases and collect results after passing the test vectors through RTL. Since the comparison between TLM and RTL simulation speed is interesting, it is fair to separate the system time used in the e components of the verification environment.

Cadence ‘Specman’ tool allows user to setup profile before running any test case. The profiler gives detailed timing report clearly marking the time spent on RTL simulation and the other components. Depending upon the test, most of the time is spent in the RTL.
From the report presented in figure 6.12, it can be seen what percentage of user time was spent in RTL and what where the other time consuming processes.

Time for the TLM module is all the user plus cpu time required to run the program. This can be calculated from the ‘time’ command. For example if there is a program named tlm_model present as an executable file, following command can be used:

```
time tlm_model.exe
```

The time returned will be in the format:

```
1.406u 0.042s 0:04.96 29.0% 2+5k 0+1io 0pf+0w
```

This example model used 1.406 seconds of user time, 0.042 seconds of system time and 4.96 seconds of real time. The sum of the user and system times is the total CPU time of the process. The percentage (29.0%) indicates the percentage of the CPU's time that the process used while it ran.

For one test case the CPU time of process for RTL and TLM is presented in table 6.6:
<table>
<thead>
<tr>
<th>RTL</th>
<th>TLM</th>
</tr>
</thead>
<tbody>
<tr>
<td>14699 sec</td>
<td>17.152 sec</td>
</tr>
</tbody>
</table>

Table 6.6  CPU time comparison

Test runs on other longer scenarios suggest that TLM models run nearly 1000X faster than their RTL counterparts. The simulation speed gain is even more for test cases covering a larger area within the region of interest.
Chapter 7

7.1 Conclusion

Challenges faced by system architects today to find the best mix of power, area and performance in light of the increasing complexity of SoC can be solved by using transaction level modeling. TLM models abstract details present in RTL and lower levels. Thus they can be implemented much earlier in the design cycle. Their simulation performance is also many times better than RTL.

TLM models allow designers to explore different design parameters by simulation. Design parameters such as no. of ports, bus width, arbitration delays etc. can be configuration parameters which can be altered for every simulation run. However, all performance exploration in TLM would only be useful if RTL is accurately represented in TLM.

This thesis proposes a methodology and a set of metrics to quantify the accuracy between TLM and RTL. The methodology maps RTL signals to TLM 2.0 base protocol; applicable on any RTL and TLM design pair. A framework for selecting and applying input stimuli is discussed. Six test cases ranging from back to back transactions of equal length to transactions of varied lengths and gap in between are applied to an AT model of a multiport memory. In RTL design, any transaction on the memory goes through the process of arbitration, scheduling and read/write operation. These functionalities are implemented in the TLM model. Proposed metrics present a quantification of how accurately these functions are implemented.

Metrics applied for accuracy quantification are based on individual transaction’s duration, start time and global order. In addition to them, overall system throughput is also compared. Since all ports of memory receive input of uniform distribution, a global measurement of error is made. If there were different traffic generators for different ports, metrics would have been collected and analyzed on individual ports.

Experiments show that TLM model shows a systematic error in start time. This error is related with the transaction duration error. Transactions show varying amount of transaction duration error based upon the characteristics of input stimuli. Results for throughput and transaction reordering follow the trends from transaction duration error.

System accuracy can be measured by combining all metrics over all test cases. These measurements can be used to systematically improve TLM models. Designer can set acceptance criteria and iteratively apply accuracy measurements on different versions of TLM model until the time it is accurate enough for performance exploration. It is recommended to integrate accuracy measurements in the verification environment for automation of accurate TLM design process.
7.2 Future Work

Despite the detailed presentation about TLM and RTL accuracy quantification put forward in this thesis, further work is needed to be done. This can be elaborated in the following points:

- Combined Metric:

  For the measurement of overall system accuracy, all collected metrics over all tests are linearly combined. There may be scenarios were one metric might be more important that other, for example transaction duration may not matter as long as correct transaction order is maintained and vice versa. Thus different emphasis can be given to different metrics. Similarly, combination of different test cases can be a product instead of a sum. Thus combined metric for classifying system accuracy needs further exploration.

- TLM Models:

  Accuracy measurement were applied on an AT model of a multiport memory. It will be meaningful to perform this analysis on different versions of TLM implementation of same RTL model. One can see the effect of different coding styles to the accuracy of the model. Similarly, it will be interesting to collect the same set of metrics from different TLM and RTL pairs. These experiments may project novel and unforeseen findings.

- Test Cases:

  In this thesis work six test cases were applied. Each test case covers different aspects of input transactions. However, not all the options regarding input stimuli were covered. Reasons were practical limitations in RTL verification environment. In future, test cases covering more options of input stimuli can be applied.

- New Metrics:

  Four metrics were presented in this thesis. There can always be more aspects to be compared between RTL and TLM. They can be presented in the form of a new metric.
Bibliography


[12] Henrik Svensson, Reconfigurable Architectures for Embedded Systems, Department of Electrical and Information Technology, Faculty of Engineering, LTH, Lund University, 2008