

Transaction Level Modelling: A reflection on what TLM is and how TLMs may be classified

Mark Burton*, James Aldis†,
Robert Günzel‡ and Wolfgang Klingauf§

*GreenSocs, mark.burton@greensocs.com

†Texas Instruments, j-aldis2@ti.com

‡TU Braunschweig (E.I.S.), r.guenzel@tu-bs.de

§TU Braunschweig (E.I.S.), w.klingauf@tu-bs.de

Abstract

Transaction-level modelling (TLM) is a poorly-defined term, promising a level of abstraction like RTL (register transfer level), where the key feature is a ‘transaction’. But unlike registers, transactions are not well defined. While many feel they have an understanding of the term, indeed many are active proponents of this form of modelling, it remains elusive. Furthermore, multiple different levels of abstraction are offered in parallel under the umbrella of the term ‘TLM’. This paper will propose a definition of a transaction and a classification of the types of data contained within transactions, based on representations of the data movements that TLM involves. Our goal in writing the paper is to increase the common industry understanding of TLM concepts, which is essential if TLM is to provide efficiency gains approaching those which have been advertised for it.

1 Players

In order to describe transactions, we need some notion of the players that conduct them. In order to describe those players, we need an understanding of how they interact - the transactions. To break this vicious circle we will start by giving some loose terms to describe the players, and then revisit these once we have an understanding of the transactions.

In this early 2000s, a rumour spread that we would no longer be allowed to call object masters and slaves because of the connotations of the words. Maybe we should have heeded the advice, and rejected the words, as now we have a plethora of competing terms. Master, slave, initiator, target, source, sink, etc. Of these, the two most used sets of terms are master/slave (e.g., [2, 4]) and initiator/target (e.g., [1, 5]). Unfortunately these often are used interchangeably (e.g., in [3], [7]). Typically they have been used at the level of IP blocks to describe the interfaces on those blocks. Or sometimes they have been used to

describe the roles that the IP blocks perform as they exercise their interface. All of which is confusing. The result is consistent, every paper that talks about TLM defines the terms that it will use. We have no option but to follow suit. We will not define a different taxonomy, simply we must specify which of the many we are using.

In an entirely general context, a transaction will be created by something, which we could call the *initiator*. The transaction will be conducted between the initiator and one or more other players, which we could call the *targets*. There may be other players involved that are neither initiators nor targets but provide some service, for example routing communications associated with the transaction between the initiator and the targets.

It is useful to examine the more specific context of the most common application of TLM, in modelling the memory-mapped busses used in microelectronic components and systems, such as PCI, AMBA, ISA, VME, OCP, CoreConnect and so on. In such busses transactions consist of at least a command, either a read or a write and including an address, and a response. A memory-mapped bus protocol is asymmetric. There is an originator of the commands and a receiver of them, which are often referred to as ‘master’ and ‘slave’ respectively. Following the memory-mapped bus example, instead of initiators and targets we will refer to *system masters*, and *system slaves*. We will refer to all players as *system components* for the time being, to include all other objects. Note that other types of protocol, such as Ethernet, do not have masters and slaves, all endpoints being functionally equivalent. However, we believe that the concepts in this paper are not limited to memory-mapped busses.

The intention of this paper is *not* to propose a naming convention for TLM. The names are adopted temporarily, to aid the discussion.

2 Transactions

Having identified the main players in transaction level models, we come to the most sensitive question. What is a *transaction*?

There is nothing stopping a transaction being a single bit of information carried at one time from one IP to another. In short, it could be a representation of a wire.

Equally, a transaction could represent the full communication between two world powers discussion the release of some prisoners of war.

The word does include the notion that there will be at least two parties involved, and that whatever the communication is about, the transaction will represent the whole of it.

This all-encompassing notion of a transaction is important, but it begs the question of where to draw the line. For example, imagine that the CPU will carry out some other communication as a direct result of a reply from a memory. Is this part of the same

transaction or not? Likewise, if a software component reacts to one message by issuing another, should this be modelled as a single transaction being manipulated, or as two transactions?

2.1 The what how and when of TLM: data lifespans

We will view a transaction as nothing more than a container of data. Transactions are conducted between a system master and at least one system slave, possibly with the help of other system components, over a time period. System masters, system slaves and system components can be considered as combinations of hardware and software (sometimes called silicon based functionality), so a transaction has an existence in both space and time.

In general, a transaction's data can change during its travels. We identify 4 types of data lifespans each with unique purposes.

Table 1. End to end invariant data

| Type | Name | Set By |
|------------------------------------------------------------------------------------|--------------------------------------------------|--------------------------------------------------------------|
| 1. | What is to be transferred: end to end invariant. | System master / slave |
| Life span | | Relevance of data |
| Remains unchanged thought the life of a transaction, both temporally and spatially | | Relevant to any, or all components on the transactions path. |

Table 1 shows the end to end invariant data. For a memory mapped bus, this data would contain the command (read, write), the data, byte enables and the place where response data is to be stored. It is important to note that the data could change in terms of its validity (see below). This means that initially only the first bytes are valid and over time the number of valid bytes grows.

Table 2. Point to point invariant data

| Type | Name | Set By |
|-----------------------------------------------------------------------------------------------|------------------------------------------------------------------|--------------------------------------------------------------|
| 2. | How it is to be transferred: point to point invariant (mutable). | System master / slave or any system component |
| Life span | | Relevance of data |
| Temporally constant, but <i>may</i> be different at different spatial locations in the system | | Relevant to any, or all components on the transactions path. |

Table 2 shows the point to point invariant data. This type of data is the least explored, and the hardest to handle.

While it may be the case that the system master's initial settings for this data are suitable for all the links in the communication chain, it is equally possible that each link will determine its own specific settings for this data. What is important is that this data remains unchanged during the lifetime of the transaction having been set up initially. It represents the mechanisms and parameters of the local point-to-point communication, not the current state of that communication.

For example, within a memory mapped bus, this data would include: (base) address, tag/thread ID, burst sequence and burst length, etc..

The existence of this type of data is a critical finding in the field of transaction level modelling. Until now, this class of data has not been explicitly considered. Implementation of a TLM framework may choose to implement it by 'shoehorning' it into either type 1 or type 3, but it is important to understand that this type of data exists because there may be benefits in treating it differently.

Table 3. Point to point changeable data

| Type | Name | Set By |
|-----------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------|-------------------------------------------------------------------------------------------------------------|
| 3. | State of the transfer: point to point changeable | Every system component |
| Life span | | Relevance of data |
| Data is only relevant when the transaction is in transit between one system component and another. That is, it varies both temporally and spatially | | Data only relevant to the point-to-point link (i.e. only the receiving system component will use the data). |

Table 3 shows the point to point changeable data. This could include for example an indication of what form of transaction update is occurring (whether a new response or a request accept), or the address for this beat of the burst, or an indication of the end of burst, or a timing annotation indicating when the update occurs.

Table 4. Local data

| Type | Name | Set By |
|----------------------------------------------------------------------------------------------------------------------------|---------------------------|----------------------------------------------------------------|
| 4. | About the transfer: local | Any system component |
| Life span | | Relevance of data |
| Data can be manipulated by the system component at any time, but will only be available during the life of the transaction | | The data will only be relevant to the system component itself. |

Type 4 data is local data (table 4). This data relates to a specific transaction. In many TLM fabrics, the system components are left to handle this form of data themselves, but it is worth mentioning here as it is the data that the system component needs which specifically relates to this transaction. Given that, it is possible that some optimisations are possible in terms of how it is implemented.

Its use seems at first obscure, but can be used for housekeeping. E.g. a router could attach an object to a transaction that contains an integer, which is the decoded address of this transaction. So the router needs to perform the decode only once, and simply remember the result (as an instance of this form of data). From the perspective of a single system component, Type 4 is just a special type 3.

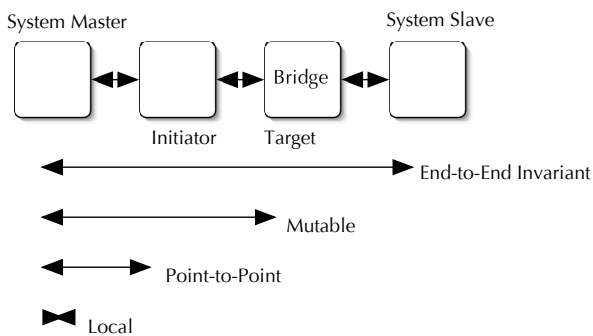


Figure 1. Data lifespans in transactions

Figure 1 depicts the four identified data types with an example transaction-level model. This taxonomy is the lynch-pin on which all of TLM modelling rests. With it, we may now describe what is, and what is not a transaction:

A transaction is the longest communication during which the invariant data, as set by a system master, remains valid.

This definition is of course dependent on the selection of which of the data set by the system master is to be considered invariant. There is, for the case of the memory-mapped bus, some scope for choice in this. For example a bus burst that gets chopped into two shorter bursts by a protocol adaptor could be modelled as a single transaction or as three.

We may also describe the other players in a system:

2.2 Bridges, adapters and transaction mutation

System components are objects that mutate transactions. Some mutations are small, some require entire new transactions to be formed.

If the mutation is small, and only affects either the mutable, point-to-point or local data, then the component is a *bridge*.

If the mutation requires a transaction's invariant data be changed - in other words, a new transaction will be created, even if there is a one-to-one mapping between them, then we call this component an

adapter. An adapter is both a system slave and a system master.

Notice how the role of a good TLM fabric can now be measured in terms of the way in which it facilitates the appropriate handling of these forms of data. In this case, a well constructed protocol will try to minimize the number of adapters needed and maximize those that can be implemented as bridges.

3 Data content types

We have identified 4 lifespans for data and we shall now address the question of what data is to be communicated.

Transaction level modelling is about system components that may be software or hardware. It is hard to classify all possible intents for TLM models but the purpose of modelling is very often either to ensure some sort of correct behaviour or to extract some performance metrics.

Overall there are three types of data that can be talked about:

- *Functional* information is used to convey data needed to perform functionally correctly.
- *Performance* information carries information about any measurable effect that the model is interested in - for instance time, or power.
- *Meta* information is used to pass data to the simulation framework.

It may be helpful to give some examples: A model may be constructed that has functionally correct behavior, but exhibits no performance measures at all. Even so, it may use a 'router' that is capable of being used with any number of system masters. In a SystemC implementation, it would have a multi-port. In order to function, each port may need to be uniquely identified. In reality, this identification would happen as a result of the physical connection that was active. In SystemC some sort of ID field is needed, this would be 'meta' information.

The same model may be extended to report information about the (simulated) time taken to perform some actions. In order to make this calculation, there will be the need to augment the purely functional model. Of course, this information may cause the functionality to change - the model is becoming more detailed, it is becoming lower in its abstraction.

Data of all three types could fall into any of the 4 data lifespans, so there are 12 possible categories of data in total. The exact collection of information in each of the 12 will be exceedingly domain-specific, and for some domains certain categories will simply not be relevant. Again, this is no more than a useful tool to analyse TLM fabrics, in order to determine how (and whether) a given modelling domain would be supported.

By way of example, the table 5 lists the sorts of data items that might be expected in the various categories for a memory mapped bus. (These are merely

suggestions. A given TLM fabric may place different items in different places, or may not support all 12 categories).

Table 5. Data items for busses

| | Functional | Performance | Meta |
|---------------------------------|-------------------|-----------------------------------|--------------------|
| end-2-end invariant | R/W, length | Bus burst info | Debug Switch |
| point-2-point invariant | Address | Thread/Tag ID | pv/pvt switch |
| point-2-point changeable | N/A | Phase (request/response/accepted) | time delay |
| local | IP specific | e.g. average delay | e.g. debug channel |

4 Abstraction levels

There are as many definitions of Abstraction Level as there are white papers on TLM modelling. None of them help. Our view is that we should not try and classify abstraction levels so much as understand what features a model has that together form its ‘abstraction’ - which becomes an n-dimensional space. We have identified at least 3 dimensions:

4.1 Abstraction and timing accuracy

Abstraction levels are almost exclusively associated with timing fidelity. This is erroneous, as abstraction can equally be applied to functional fidelity. In either case, the degree to which a model is an abstract representation of reality will be very specific to that individual model.

Some abstractions can be seen clearly, for instance the typical abstraction of simple boolean ‘wires’ to more complex data structures. Likewise complex pipelines are often reduced to simplistic state machines.

Others are less obvious, for instance whether a specific pipeline is being modelled or not may have no functional effect, but may change the timing fidelity by a small percentage on average, or significantly if that specific pipeline is being exercised.

In the end, there are only really 2 generic timing fidelity abstractions that can be measured: 100% the same as the RTL in all cases (cycle accurate), and the rest. Dividing the rest into ‘bands’ of abstraction, for instance saying ‘some pipelines have been abstracted away’ is not especially helpful to the user. Giving the user confidence about the accuracy of each individual model and indicating where that model will not be accurate may be much more help. In the end, the user must know that if the model is not 100% accurate, then the entire sequence of functional execution could be radically different from the real implementation.

4.2 Abstraction and technology

Another approach to the classification of timing fidelity abstractions would be to base the discussion on

technologies. Clearly models can be created using different technologies (or models of computation) which will permit different levels of timing fidelity. These technologies are not necessarily compatible with each other. This includes the interfaces the models use, how they use them, and the data types passed across them. This potential incompatibility is a key driver behind the desire to identify ‘abstraction levels’.

Identifying the technology used to create a model is therefore potentially useful, and may be associated in some way with ‘abstraction level’.

There are 2 technologies that have been proposed for TLM:

1. System master blocked (i.e. it stalls during communication): Here timing information is abstracted away to the degree that no activity will occur in a system master between the time that a transaction is sent out from the system master, and the moment when the transaction returns to that master. This technology typically assumes some sort of return. It can be implemented simply, by a function call. More accurate timing estimations can be added retrospectively but in no case can they affect time functional behaviour at a level of granularity that would ensure the system performed exactly as an RTL implementation would. This form of model can never be 100% accurate. We can distinguish two versions of this:
 - (a) non-blocking: immediate return of simulation control to the system master is required. This technology is likely to lead to the fastest simulations but the poorest timing fidelity.
 - (b) blocking: time may pass while the system master is blocked and other independent activity of the system be simulated.
2. System master free (i.e. it continues processing during communication): In this case, a system master may continue while a transaction is on its path. It may, for instance, emit other (parallel) transactions. Some mechanism is employed to indicate to the components of the system that an important timing point has been reached by a transaction – for instance that it has been responded to and the response data is now valid. Technically several techniques have been deployed to implement this form of modelling:
 - (a) cycle callable: on each cycle, data is passed between the components indicating the current state of the transaction
 - (b) event and shared memory message passing: the transaction is held in shared memory, when a timing event occurs, an event is generated and the listening components may read the updated state information from the shared memory object

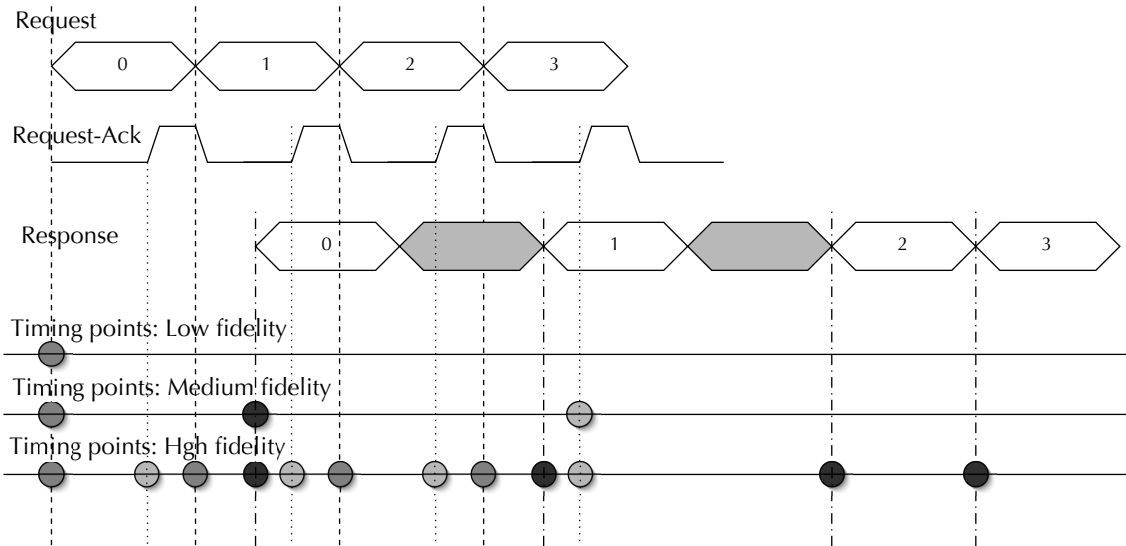


Figure 2. Timing points in transactions

- (c) pass-by-value message passing: the transaction data is passed from component to component at the correct time, with changes to its state being made as it passes through components.

While these are different technologies, they do not differ fundamentally in terms of how accurately they can model timing effects. In each case, a decision can be made about what is going to be modelled, and what is not. In all cases, these forms of model can be 100% accurate.

We seem to have arrived at the same 2 levels of abstraction. However, these various technologies are not compatible with each other without bridges or adapters, and in some cases not even with them. They are *probably* what people need to know when they talk about abstraction levels.

4.3 Abstraction and time points

A level of abstraction that has not been so widely discussed is that of the timing points in a protocol. We have noted above how each of the three popular technologies for system-master-free style modelling allows the designer to choose which timing points in a protocol they will model.

Hence, we can classify abstraction level in terms of which of the timing points are modelled. These are true abstractions, as smaller grained timing points are subsumed into higher level ones.

Figure 2 shows an example of a simple point-to-point memory-mapped bus protocol which is being used to support a transaction of duration 4 beats. Three options for timing points are shown, labelled low, medium and high fidelity. In the low fidelity, only the creation instant of the transaction is captured. This corresponds to the use of a System Master Blocked technology. In the medium fidelity case

the start of the first of the 4 responses is also captured, as well as the acceptance by the System Slave of the last of the requests. This permits the models to take into consideration the occupation of the request wires and the latency of the slave. The final case is the high fidelity one, where all the significant events of the protocol are captured as timing points and 100% cycle-accuracy is possible.

The timing points would in a TLM simulation generally be modelled by function calls.

This section and the preceding one both describe abstraction levels in terms of the theoretical capabilities of the TLM interfaces at the module boundaries. It is vital to point out that the abstraction level of a model need not be the same as that of its external interfaces. In fact it is common for a model with very low timing fidelity to be created with external interfaces which could in theory support a much higher fidelity, for example for ease of integration in some system. This adds a further issue to the classification of levels of abstraction: a model needs to be classified separately for its content and its interfaces? This will be explored in the following section.

4.4 Abstraction and use cases

Another popular approach is to examine use cases. Here, the interest is not so much in the model, what has been abstracted, or the models technology, rather the focus is on how that model may be used.

It is presumed to be possible make decisions about what sort of model to write and what technologies to deploy from the use case perspective. In this case, identifying a set of use cases should lead to a set of technologies and a suitable set of timing points. Hence it is relatively sensible to label a set of abstractions using the use cases that they are suitable for.

The difficulty with this approach is that the use cases are varied, and not static, and the technology choices for a given use case are not always fixed.

Use cases that have been proposed include ‘PV’ for programmers view, ‘AV’ for architects view, ‘VV’ for verification view. The intent of a view is that it provides a glimpse of a world from the viewers perspective, merely the part of the world the viewer wants to see. In other words, it describes the acceptable level of abstraction - what may be abstracted away. In the case of a programmers view, it is not just acceptable, but in some cases desirable to abstract away all the possible implementation detail (including timing effects) such that the programmer can only rely upon the system level architecture of the device when constructing their code. This may be desirable to ensure that one revision of the software is not tied to one revision of the hardware. Hence the name programmers view. The difficulty is that it is not clear whether this feature (of not being able to detect implementation details) is a requirement of a programmers view model, or an acceptable abstraction. In other words, does a hardware implementation of a device provide a suitable programmers view? Different technologies provide different types of model, with different execution speeds, and different abstractions, but they may all be used by programmers.

Similarly there are plenty of verification and architectural tasks that do not need any more timing detail than the programmer. And there are architectural tasks that require 100% cycle accuracy.

The term programmers view could be interpreted not as a use case but as a description of an abstraction: the model will contain only those elements of the system that would be visible to a programmer. This interpretation is attractive because it defines quite accurately the functional and timing fidelity of the model but does not imply that the purpose of the model is software development. Unfortunately this is not true for the terms architects view and verification view so, unless we find alternative terms, this approach to classification is broken.

5 Summary

We have attempted to review the language and concepts in common use in the world of transaction level modelling. Our conclusion is that the concepts are not well defined today and that the electronics industry will be hampered by this lack of definition. As IP reuse and adoption of ESL/TLM continue to expand, it will be essential for consistent modelling approaches to become established.

The discussion in this paper can be summarised as follows:

- Transactions are containers for information that is communicated between multiple components of a system;
- The information can be classified into 12 types, all of which should be efficiently supported by a

good TLM infrastructure. There are four classes of information ownership and stability:

- Temporally and spatially constant information defined by the system master when the transaction is created;
- Temporally constant information that may vary spatially, being defined by each component when it first sees the transaction;
- Spatially and temporally unstable information, local to the interface between any pair of components;
- Information private to some component;

and there are three type of data:

- Functional data;
 - Performance data;
 - Meta data;
- The classification of abstraction levels for TLM is extremely challenging. Different approaches are possible but none is fully satisfactory:
 - Generic definitions of timing accuracy is doomed to failure except for the trivial cases. Practically all ‘approximately accurate’ models will need module-specific descriptions of their accuracy.
 - Accuracy of models needs to be specified separately for the internal behaviour and external interfaces.
 - Classification based on external interface technology or external interface timing points is possible but protocol-specific and says nothing about the internal behaviour.
 - Classification based on expected use of the model is confusing and unhelpful.

Further work is required on this subject, in order to align the various standardisation efforts that are ongoing, in particular within OSCI (Open SystemC Initiative) [6] and SPIRIT [8].

References

- [1] A. Bernstein, M. Burton, and F. Ghenassia. How to bridge the abstraction gap in system level modeling and design. *Proc. ICCAD*, 2004.
- [2] L. Cai and D. Gajski. Transaction Level Modeling: An Overview. *Proc. CODES+ISSS*, 2003.
- [3] F. Ghenassia. *Transaction-Level Modeling with SystemC*. Kluwer Academic Publishers, 2006.
- [4] T. Grötter, S. Liao, G. Martin, and S. Swan. *System Design with SystemC*. Kluwer Academic Publishers, 2002.
- [5] A. Haverinen, M. Leciercq, N. Weyrich, and D. Wingard. SystemC based SoC Communication Modeling for the OCP Protocol. *OCP-IP White Paper*, 2002.
- [6] Open SystemC Initiative. OSCI Homepage. <http://www.systemc.org>.
- [7] A. Rose, S. Swan, J. Pierce, and J. M. Fernandez. Transaction Level Modeling in SystemC. *OSCI TLM Working Group*, 2005.
- [8] SPIRIT. The SPIRIT Consortium Homepage. <http://www.spirit-consortium.org>.