# TLM Modelling of 3D Stacked Wide I/O DRAM Subsystems

## A Virtual Platform for Memory Controller Design Space Exploration

Matthias Jung, Christian Weis, Norbert Wehn
University of Kaiserslautern, Germany
Microelectronic System Design Research Group
{jungma,weis,wehn}@eit.uni-kl.de

Karthik Chandrasekar
Delft University of Technology, Netherlands
Computer Engineering
k.chandrasekar@tudelft.nl

## ABSTRACT

Three-dimensional stacked Wide I/O DRAMs have been proposed as a promising solution to overcome the pin-limited memory performance growth, the power vs. bandwidth dilemma and the Memory Wall. This new DRAM architecture and organisation requires a new generation of DRAM memory controllers.

In this paper, we present a new methodology using virtual platforms to model the backend of a 3D-DRAM memory subsystem (channel controller and Wide I/O DRAM) with special *SystemC TLM2.0* phase extensions. This methodology enables us to explore the complete design space of memory controllers at the system level at very fast simulation speeds with precise timing accuracy. We show simulation speedups of up to 377x with a timing accuracy of 99% compared to an equivalent cycle and pin accurate *SystemC* based RTL simulation.

## Categories and Subject Descriptors

B.3.3 [**Performance Analysis and Design Aids**]: Simulation; B.3.1 [**Semiconductor Memories**]: Dynamic memory (DRAM)

## General Terms

Performance, Measurement

## Keywords

Virtual Platforms, TLM, Simulation acceleration, Design Space Exploration, 3D-Stacked DRAMs

## 1. INTRODUCTION

Today's high-performance and embedded applications are characterised by ever increasing demands on the memory bandwidth and capacity. Due to this reason the need for a higher number of I/Os of the memory subsystem is continuously growing [1]. However the number of I/O pins is limited by the package and power considerations. The energy consumed per bit for accessing off-chip memory is many times higher than the energy required for on-chip memory accesses. This is due to complex and power hungry I/O transceiver circuits that have to deal with the electrical characteristics of the interconnections between the chips.

Moreover, the maximum achievable memory bandwith in current off-chip DRAMs is lesser than the bandwidth requirements of modern high performance and embedded computing systems. This problem is known as the Memory Wall [2].

Three-dimensional stacked memories have been proposed as a promising solution for these problems. These memories reduce the distance between CPU and external RAM from centimetres to micrometres by means of TSV (through silicon via) technology. The available bandwidth has increased but more importantly this technology provides a major boost in energy efficiency in comparison to standard off-chip SDR or DDR/2/3 DRAM devices [3, 4]. The combination of high bandwidth communication with the lower power consumption of 3D integrated memory is an ideal fit for high-performance and embedded applications.

An optimal architecture of a 3D-DRAM with respect to bank structure, number of layers and channels is presented in [5]. Figure 1 shows a 3D-DRAM cube that is composed of four independent vertical channels, each consisting of 8 independent banks. Every channel has its own channel controller (CCx) that is connected via microbumps and TSVs to the DRAM layers.

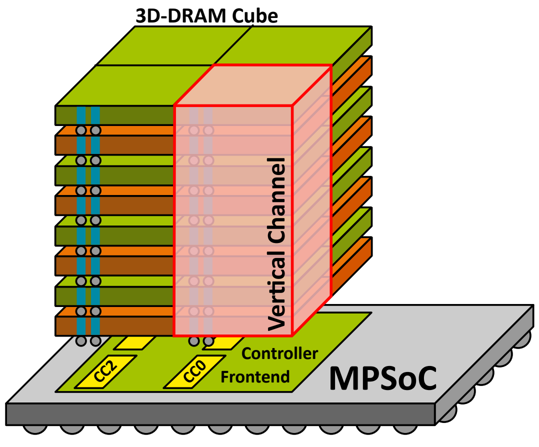

**Figure 1: 3D-DRAM Cube with Vertical Channel Architecture**

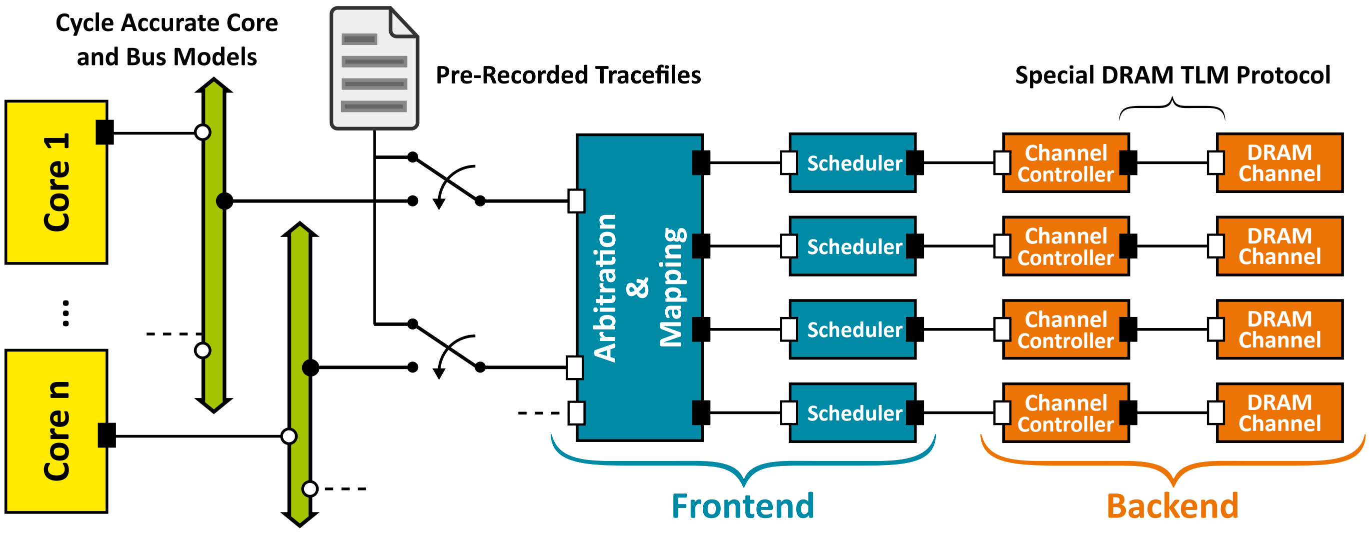

**Figure 2: Virtual platform for 3D-DRAM controller exploration with 4 channels**

These new memory architectures require a new generation of DRAM controllers to exploit the full bandwidth and energy efficiency of 3D-DRAMs. Some examples are presented in [3, 6, 7, 8, 9]. However, the design space for a 3D-DRAM controller is huge with respect to different mappings, configurations, scheduling and arbitration algorithms. Therefore a flexible model is needed for an effective and fast investigation. The exploration of these new controllers with traditional cycle and pin accurate (CA) *Register Transfer Level* (RTL) models provides the needed accuracy. However, the RTL models are inflexible in terms of the large design space and the very long simulation times. This is due to the large number of signals, processes and events that have to be simulated [10]. However it is possible to simulate at a higher level of abstraction without loosing simulation accuracy.

One way to achieve a higher level of abstraction is using *Transaction Level Modeling* (TLM) [11] for system level simulation. TLM has emerged as standard methodology for *Electronic System Level* (ESL) design. Since 2012, TLM is part of the IEEE1666 SystemC standard [12]. TLM can help speeding up the simulation by replacing all pin-level events with a single function call. For instance, a single bus transaction produces approximately 75 events in a RTL simulation compared to only a handful of events in a TLM simulation [13]. It is possible to reach speedup factors up to 10.000 [12].

However, simulation speed comes at the cost of reduced timing accuracy. For the purpose of modeling DRAM controller architecture, the standard TLM coding styles are not accurate enough to reflect a realistic behaviour. In this paper, we present an effective way to extend the *Approximately Timed* (AT) TLM protocol with DRAM related phases to achieve the needed accuracy.

The rest of the paper is structured as follows: Section 2 describes the state of the art regarding DRAM simulation. Section 3 presents the simulation environment using virtual platforms and the implementation of the DRAM controller backend model. In Section 4 the TLM model is compared to a cycle accurate implementation with respect to speedup and accuracy. Finally, the paper concludes with Section 5.

## 2. RELATED WORK

Various approaches exist to integrate and simulate the memory subsystem. *DRAMSim2* [14] is a memory system simulator that uses cycle accurate and highly detailed C++ models of the controller and the DRAM. Thus it is not suitable for fast and exhaustive system level investigations. The model of the *gem5* system simulator [15] is not detailed enough to reflect a realistic DRAM subsystem behaviour. A TLM based DRAM model is available from OCP-IP [16]. In contrast to our implementation it uses a clock based calculation of state and delay of DRAM and controller, which leads to an increase in simulation time. The commercial *DesignWare* TLM Library [17] from Synopsys and Sonics' *MemMax Memory Subsystem* [18] include AT DDR3 memory controller models that are not changeable and they do not disclose any details. With our proposed methodology we enable the exploration of the huge and complex design space of the Wide I/O DRAM subsystem using virtual platforms.

## 3. VIRTUAL PLATFORMS

The TLM2.0 platforms for design space exploration are created with the help of the *Synopsys Platform Architect* tool [19]. It permits the modeling of different types of system architectures with cycle accurate bus and core models. To get even faster simulations we record transaction traces and replay them with elastic trace players [20]. The ability to process traces from other simulators like *gem5* [15] or *Simplescalar* [21] opens up the opportunity of using multiple sources for analysis and explorations.

Figure 2 shows the architecture of the flexible virtual platform of our 3D-DRAM multi-ported memory subsystem. The platform contains either cycle accurate core and bus models (e.g. ARM processor models from Carbon) or elastic trace players for generating input data for the subsequent 3D-DRAM memory subsystem. The subsystem consists of a frontend and a backend part. The frontend contains an arbitration and mapping block that handles the incoming transactions and forwards them to the different channel schedulers according to specific priority schemes and mappings. The single channels of the subsystem are independent. Therefore each channel has its own scheduler and controller. The scheduler module collects transactions and reorders them with respect to latency and power savings and issues them to the backend with the channel controller that takes care of the correct use of the DRAM. This virtual platform gives us the flexibility for exhaustive explorations.

All connections are implemented in the TLM2.0 *Approximately Timed* (AT) coding style. An exception is the connection between controller and channel: For this connection we extended the TLM *non-blocking* protocol with DRAM specific phases. With these phase extensions we can achieve the required accuracy to observe e.g. the detailed impact of different address mappings or reordering algorithms of the scheduler. We separated the channel controller from the DRAM channel model to provide the interoperability to analyse various DRAM device models. The design of these phases and the implementation of the channel controller and the DRAM model is explained in the following sections.

## 3.1 Design of the Phases

The TLM non-blocking base protocol consists of following phases: `BEGIN_REQ`, `END_REQ`, `BEGIN_RESP` and `END_RESP`. Instead of simulating every clock cycle, the simulator is triggered only at the `BEGIN` (`<`) and `END` (`>`) phase events. Using the *JEDEC Wide I/O Single Data Rate* standard [22] we have defined additional application specific phases for the different DRAM commands by means of the `DECLARE_EXTENDED_PHASE()` macro. These phases are calibrated to the cycle accurate behaviour of the *Wide I/O* standard, although it could be easily adopted to any DRAM. Three of the important phases (ACT, PRE, RD) and their timing dependencies are presented in this section.

### *Activate - ACT*

DRAMs are organised in banks, rows and columns. To access data in a row of a certain bank in the DRAM, the Activate command (ACT) must be issued before any read or write operation can be executed. The ACT command opens an entire row of the memory array that is transferred into the bank's row buffer (sense amplifiers), which serves as a cache. Then the DRAM accepts a read or write command after the RAS to CAS delay $t_{RCD}$. The TLM phase-pair consists of `BEGIN_ACT` and `END_ACT` and its duration is equal to $t_{RCD}$ (see Figure 3).

The minimum time interval between successive ACT commands to the same bank is determined by the row cycle time of the device ($t_{RC}$). The minimum time interval between ACT commands to different banks is $t_{RRD}$ (see Figure 4).

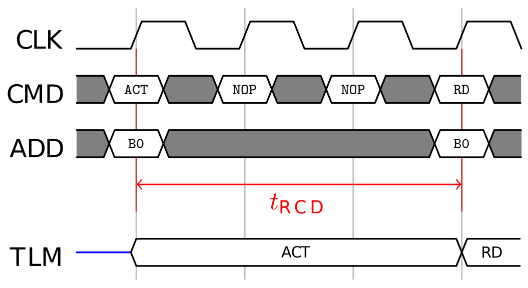

**Figure 3: The design of the ACT command phases**

### *Precharge - PRE*

If a certain row in a bank is active it must be precharged (PRE) before another row can be activated. The minimum time between PRE and ACT is defined as $t_{RP}$. The TLM phase-pair consists of `BEGIN_PRE` and `END_PRE` and its duration is equal to $t_{RP}$.

A bank cannot be precharged until at least $t_{RAS}$ time after the previous `ACT` (see Figure 4). The row cycle time can be calculated as $t_{RC} = t_{RAS} + t_{RP}$.

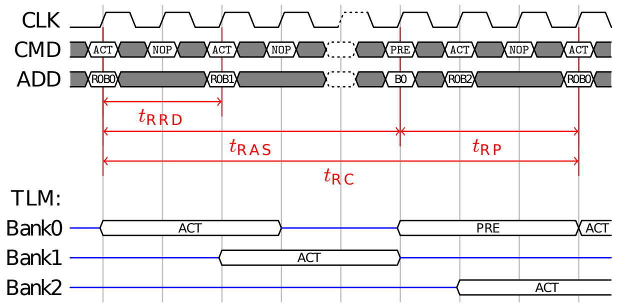

**Figure 4: Duration and dependencies of PRE**

### *Read - RD*

The read command (RD) is used to initiate a burst read access to an active row with a certain burst length. It is possible to turn on auto precharge in the configuration. If auto precharge is selected, the row being accessed will be precharged at the end of the read burst if the minimum $t_{RAS}$ timing is fullfilled (closed page mode). If auto precharge is not selected, the row will remain open for subsequent accesses (open pagemode). When no auto precharge command has been issued, data from a read burst may be concatenated by a subsequent RD command. The first data element from the new burst follows the last element of a completed burst. The new RD command should be issued $t_{BL}$ after the first RD command, where $t_{BL}$ is the burst length of the last RD before. The TLM phase-pair consists of `BEGIN_RD` and `END_RD` and its duration is equal to the read latency $t_{RL}$ plus the burst length $t_{BL}$ (see Figure 5).

If a RD transaction is being executed, a subsequent WR can be issued after the completion of this RD command. If a WR command is issued, the following RD command has to wait a minimum time of $t_{WTR}$ (write to read delay) .

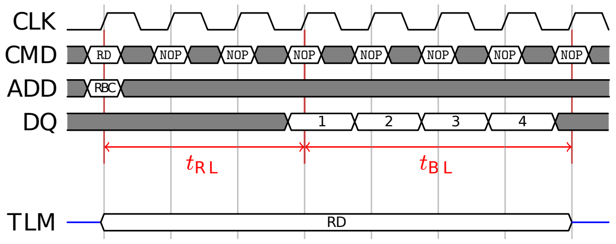

**Figure 5: Duration of the RD command**

The phases of all other DRAM commands such as Write (`WR`), Precharge-All (`PREA`), Refresh (`REFA`) etc. and their interdependencies are implemented in a similar way. An important restriction is that two phases must not begin at

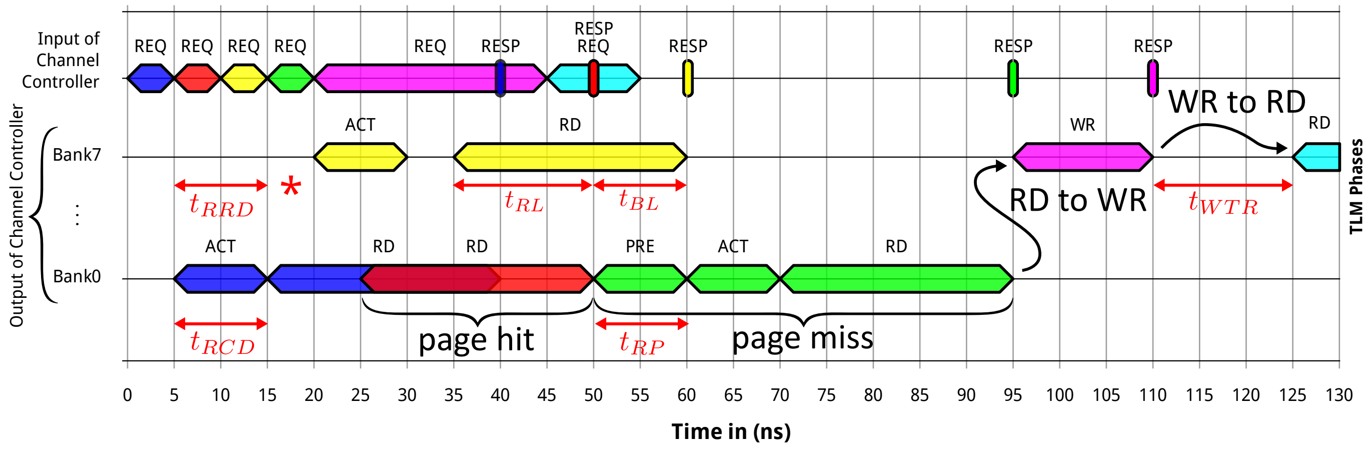

**Figure 6: Example of a typical TLM trace with DRAM related phases**

the same cycle since this would lead to a conflict on the command bus. If this case occurs the second phase must be shifted for one clock cycle.

Figure 6 shows an example of a typical trace with DRAM specific TLM phases, which are depicted per bank. The first line shows the input of the standard TLM2.0 target socket of the channel controller and the following lines the output to the DRAM. The controller of this example is able to handle a new request every clock cycle. It has a input buffer size of four, which leads to stalling in case the buffer is full.

In Figure 6 the previously discussed timing dependencies are shown, e.g. the ACT in Bank7 needs to be shifted by one clock cycle because of the scheduled RD command in Bank0 (*). The second RD command in Bank0 can start already after $t_{BL}$ of the first RD (page hit). The third RD command on Bank0 has to access another row. Therefore a PRE and an ACT command are issued in advance (page miss). The dependencies of consecutive RD and WR commands are shown at the end of the trace example.

## 3.2 Implementation

TLM transactions are sent through initiator sockets and received through target sockets. In the AT coding style the initiator calls the **nb_transport_fw** function (called *forward path*) of the target to issue a transaction. The target receives the transaction and stores it in the payload event queue (PEQ) that queues the transaction until it is ready to be executed. Then a callback function is called that processes the transaction. The target calls the **nb_transport_bw** function of the initiator (called *backward path*) to answer the request of the initiator with an **END** phase.

The controller and DRAM TLM models have to deal with the timings discussed in the last section. Figure 7 shows the basic structure of the controller backend model, that consists of standard TLM2.0 payload event queues (PEQ), a status table for the DRAM bank states and timing check functions (TC) for the DRAM commands.

If a new transaction is sent from scheduler and enters the channel controller with the **BEGIN_REQ** phase it is stored in the input PEQ. The callback function of the PEQ is called and an **END_REQ** is sent back to the scheduler. Then it is checked whether there is a page miss, bank miss or page hit according to the address and the current state of the target bank (stored in the state table) to determine the appropriate **BEGIN** phase that will be issued to the DRAM.

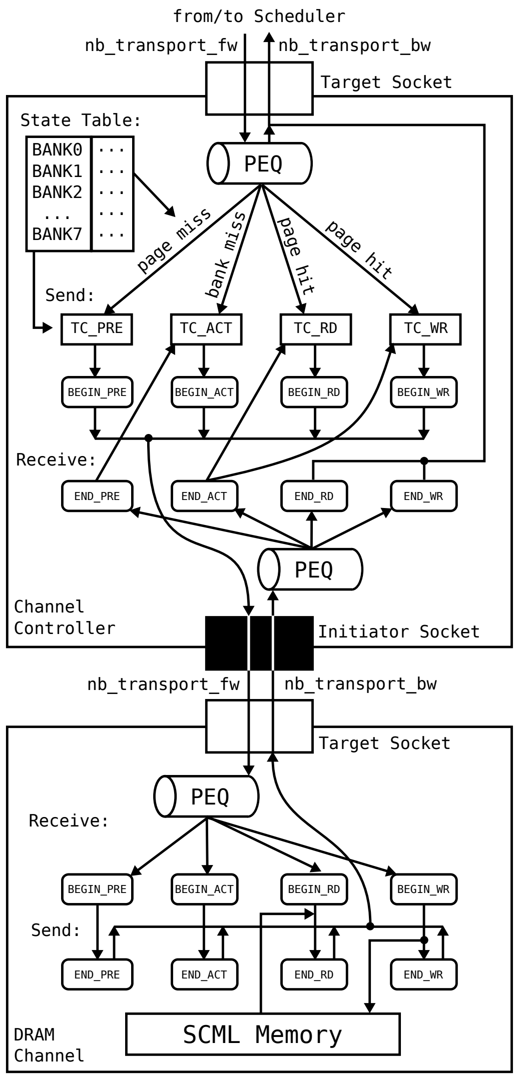

**Figure 7: Architecture of the backend TLM model**

The TC function corresponding to the selected `BEGIN` phase is called next. This function checks the timing dependencies within the target bank and across all other banks and returns a waiting delay which is passed to the `nb_transport_fw` function of the DRAM.

The transaction is passed with the determined `BEGIN` phase to the DRAM device and in case of a RD or WR the data is loaded from or stored in a SCML (SystemC Modeling Library) [23] memory that enables detailed debugging with the Synopsys tools.

Then the transaction is issued back to the controller by calling `nb_tranport_bw` with the appropriate `END` phase. In case the controller received an `END_PRE` or an `END_ACT`, a `BEGIN_ACT` or `BEGIN_RD`/`BEGIN_WR` must be sent to the DRAM after the associated timing checks. If the controller receives an `END_RD` or an `END_WR` phase, the transaction is sent back to the scheduler component with a `BEGIN_RESP` phase. The scheduler acknowledges the transaction by issuing a `END_RESP`.

The TLM abstraction does not in anyway restrict our ability to analyse DRAM power consumption. In fact by logging just the timestamps of the phases we can employ transaction based [24, 25] and CA [25] simulators to obtain power consumption estimates, as well.

## 4. EXPERIMENTS AND RESULTS

To measure the performance of the TLM backend model we compare it against a handwritten cycle accurate (CA) SystemC model. The two platforms are created in the *Synopsys Platform Architect* [19] and use the same input stimuli which are generated from the *CHStone* [26] and the *Mediabench* [27] benchmarks.

The benchmark traces are generated by means of the Simplescalar simulator with a 16KB L1 D-cache, 16KB L1 I-cache, 128KB shared L2 cache and 32-byte cache line configuration. We filtered out the L2 cache misses for instructions and data, and obtained a trace of the transactions meant for the DRAM.

The TLM model is very fast with respect to runtime. For instance the *mediabench mpeg2encode* runs 1h 41m with the CA model compared to 42s with the TLM Model giving a speedup of 145x. Similarly with the *mediabench h263decode* we achieved a speedup of 377x compared to the CA implementation as shown in Table 1, Figure 8 and Figure 9. The difference of the simulation times between the CA and TLM model are very small and this results in a overall accuracy of more than 99%.

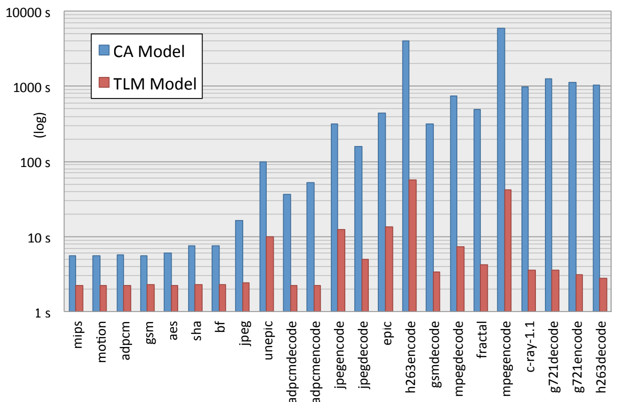

**Figure 8: Runtime of the Models**

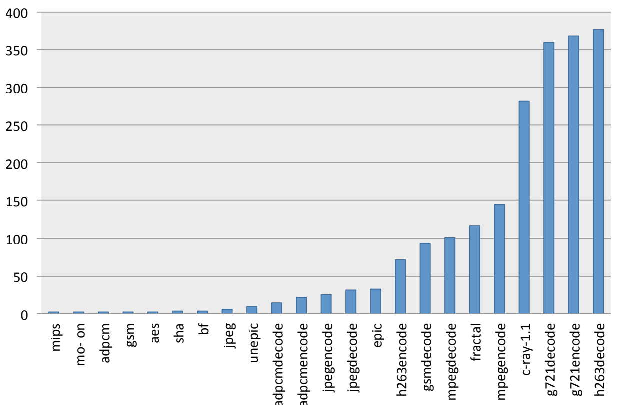

**Figure 9: Measured Speedup**

For some traces we could only achieve a speedup of 2x. The reason for this is the density of the traces that is calculated as follows:

$$\text{Trace Density} = \frac{\text{Number of Transactions}}{\text{Simulated Time}}$$

If the trace density is high, the TLM model has to simulate more events in a certain time interval and the simulation time converges to the CA implementation. As a result smaller speedup is achieved. However if the trace density is low the TLM model achieves much better performance as the CA has to compute each clock event. This dependency is shown in Figure 10 that depicts the different benchmarks from Table 1 with respect to speedup and density.

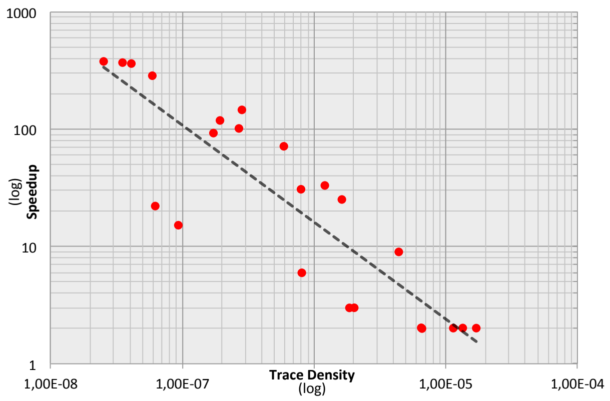

**Figure 10: Performance of the TLM Model**

In summary the results show that the simulation is speeded up in orders of magnitudes while maintaining high accuracy.

## 5. CONCLUSIONS

Three-dimensional stacked DRAMs are the future of memory technology for high performance and embedded computing. In this paper, we presented a virtual platform for the exploration and optimisation of multi-channel Wide I/O DRAM controllers. Towards this, we introduced a new DRAM specific TLM protocol for the backend part of the controller to obtain the needed accuracy to analyse the impact of different scheduling algorithms or arbitration schemes on the latency and power on system level, while improving simulation speed.

Due to the large speedup of the TLM models and the flexibility of the platform, we are able to explore alternative controller architectures in a fast and accurate manner.

| Benchmark | # Trans. | Simulated Time ($\mu s$) | Runtime CA (s) | Runtime TLM (s) | Accuracy | Speedup | Density |
|---|---|---|---|---|---|---|---|
| chstone mips | 993 | 87.348 | 5.53 | 2.25 | 98.98% | 2 | 1.1E-05 |
| chstone motion | 1144 | 67.097 | 5.55 | 2.25 | 99.17% | 2 | 1.7E-05 |
| chstone adpcm | 1211 | 188.125 | 5.75 | 2.23 | 99.65% | 2 | 6.4E-06 |
| chstone gsm | 1202 | 88.751 | 5.55 | 2.26 | 99.29% | 2 | 1.4E-05 |
| chstone aes | 1382 | 208.579 | 5.98 | 2.24 | 99.97% | 2 | 6.6E-06 |
| chstone sha | 1503 | 803.847 | 7.50 | 2.27 | 99.99% | 3 | 1.9E-06 |
| chstone bf | 1660 | 829.227 | 7.59 | 2.26 | 99.92% | 3 | 2.0E-06 |
| chstone jpeg | 3247 | 4004.316 | 16.36 | 2.39 | 99.99% | 6 | 8.1E-07 |
| mediabench unepic | 129245 | 29214.391 | 99.27 | 9.96 | 99.42% | 9 | 4.4E-06 |
| mediabench adpcmdecode | 1059 | 11416.187 | 36.06 | 2.26 | 99.94% | 15 | 9.3E-08 |
| mediabench adpcmencode | 1072 | 17370.592 | 51.73 | 2.25 | 99.96% | 22 | 6.2E-08 |
| mediabench jpegencode | 173995 | 107826.614 | 317.70 | 12.45 | 100.0% | 25 | 1.6E-06 |
| mediabench jpegdecode | 43143 | 54131.091 | 156.88 | 4.94 | 99.96% | 31 | 8.0E-07 |
| mediabench epic | 182957 | 151658.953 | 439.88 | 13.33 | 100.00% | 33 | 1.2E-06 |
| mediabench h263encode | 858099 | 1448583.227 | 4005.70 | 56.11 | 100.00% | 71 | 5.9E-07 |
| mediabench gsmdecode | 19734 | 116129.692 | 318.65 | 3.39 | 99.96% | 93 | 1.7E-07 |
| mediabench mpegdecode | 72043 | 268409.430 | 746.23 | 7.37 | 100.00% | 101 | 2.7E-07 |
| mediabench fractal | 33895 | 176513.378 | 488.85 | 4.18 | 99.99% | 117 | 1.9E-07 |
| mediabench mpegencode | 616935 | 2196276.902 | 6029.20 | 41.48 | 100.00% | 145 | 2.8E-07 |
| mediabench c-ray-1.1 | 21627 | 365540.557 | 997.03 | 3.53 | 99.99% | 282 | 5.9E-08 |
| mediabench g721decode | 19350 | 472404.358 | 1275.99 | 3.55 | 99.99% | 359 | 4.1E-08 |
| mediabench g721encode | 14655 | 418781.118 | 1131.07 | 3.07 | 99.99% | 368 | 3.5E-08 |
| mediabench h263decode | 9866 | 391013.384 | 1052.30 | 2.79 | 99.99% | 377 | 2.5E-08 |

**Table 1: Runtimes and Simulation Times for Different Benchmarks**

## REFERENCES

[1] P. Stanley-Marbell, et al. Pinned to the walls; Impact of packaging and application properties on the memory and power walls. In *proc. ISLPED 2011*, Aug. 2011.

[2] Wm. A. Wulf et al. Hitting the memory wall: implications of the obvious. *SIGARCH Comput. Archit. News*, March 1995.

[3] C. Weis, et al. An energy efficient DRAM subsystem for 3D integrated SoCs. In *proc. DATE 2012*, march 2012.

[4] G. Manil D., et al. DRAM selection and configuration for real-time mobile systems. In *proc. DATE 2012*, March 2012.

[5] C. Weis, et al. Design space exploration for 3D-stacked DRAMs. In *proc. DATE 2011*, March 2011.

[6] Cadence Design IP: Wide-I/O Controller. Technical report, Cadence Design Systems, Inc., 2012.

[7] M. Ghosh et al. Smart Refresh: An Enhanced Memory Controller Design for Reducing Energy in Conventional and 3D Die-Stacked DRAMs. In *proc. MICRO 2007*, Dec. 2007.

[8] I. Loi et al. An efficient distributed memory interface for many-core platform with 3D stacked DRAM. In *proc. DATE 2010*, March 2010.

[9] Guangfei Zhang, et al. Heterogeneous multi-channel: fine-grained DRAM control for both system performance and power efficiency. In *DAC 12*, Jun. 2012.

[10] F. Kesel. *Modellierung von digitalen Systemen mit SystemC: Von der RTL- zur Transaction-Level-Modellierung.* Oldenbourg Wissenschaftsverlag, 2012.

[11] Lukai Cai et al. Transaction level modeling: an overview. In *proc. CODES+ISSS '03*, 2003.

[12] IEEE Computer Society. *IEEE 1666: SystemC Language Reference Manual*, 2012 edition, 2011.

[13] D.C. Black, et al. *SystemC: From the Ground Up, Second Edition.* Springer, 2009.

[14] P. Rosenfeld, et al. DRAMSim2: A Cycle Accurate Memory System Simulator. *Computer Architecture Letters*, 10(1), Jan.-June 2011.

[15] Nathan Binkert, et al. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, August 2011.

[16] Nan Li, et al. Design and Implementation of an Accurate Memory Subsystem Model in SystemC. Technical report, December 2010.

[17] DesignWare TLM Library. http://www.synopsys.com/Systems/VirtualPrototyping/VPModels/Pages/DW-TLM-Library.aspx, 2012.

[18] MemMax Scheduler. http://sonicsinc.com/wp-content/uploads/2012/09/Sonics_ProductBrief_MemMax.pdf, 2012.

[19] Inc Synopsys. Synopsys Virtual Prototyping Solution. http://www.synopsys.com/Systems/VirtualPrototyping/Pages/default.aspx, 2012.

[20] Tim Kogel. Generating Workload Models from TLM-2.0-based Virtual Prototypes for Efficient Architecture Performance Analysis. http://www.nascug.org/events/13th/tlm20_workload_models.pdf, Jun. 2010.

[21] Doug Burger et al. The SimpleScalar tool set, version 2.0. *SIGARCH Comput. Archit. News*, 25(3), June 1997.

[22] Jedec Solid State Technology Association. Wide I/O Single Data Rate JESD 229, Dec. 2011.

[23] SystemC Modeling Library (SCML). http://www.synopsys.com/cgi-bin/slcw/kits/reg.cgi.

[24] Micron Technology Inc. Calculating Memory System Power for DDR3. Technical report, 2007.

[25] Karthik Chandrasekar, et al. Improved Power Modeling of DDR SDRAMs. In *proc. DSD'11*, 2011.

[26] Yuko Hara, et al. Proposal and Quantitative Analysis of the CHStone Benchmark Program Suite for Practical C-based High-level Synthesis. *JIP*, 17, 2009.

[27] Mediabench. http://euler.slu.edu/ fritts/mediabench/.