CONFIGURABLE VERSION MANAGEMENT HARDWARE TRANSACTIONAL MEMORY FOR EMBEDDED MULTIPROCESSOR FIELD-PROGRAMMABLE GATE ARRAY

JEEVAN A/L SIRKUNAN

UNIVERSITI TEKNOLOGI MALAYSIA

CONFIGURABLE VERSION MANAGEMENT HARDWARE TRANSACTIONAL MEMORY FOR EMBEDDED MULTIPROCESSOR FIELD-PROGRAMMABLE GATE ARRAY

JEEVAN A/L SIRKUNAN

A thesis submitted in fulfilment of the
requirements for the award of the degree of
Master of Engineering (Electrical)

Faculty of Electrical Engineering
Universiti Teknologi Malaysia

JULY 2015

To my supervisors, family and friends for taking care of me during my studies

ACKNOWLEDGEMENT

First and foremost, I would like to thank God for giving me the strength to complete this thesis. I would also like to express my greatest appreciation to my supervisor, Associate Professor Dr. Muhammad Nadzir Bin Marsono, for giving me the opportunity in working in an amazing field of research. With his continuous encouragement, criticism and guidance I was able to complete my research. Thanks to him, I was able to realize my full potential in academic and also other aspects of life. Besides that, I would like to thank my co-supervisor Dr. Ooi Chia Yee for commenting on my work and involving me in external project which expanded my horizon in this field.

My sincerest appreciation goes to my seniors and fellow lectures, Alireza Monemi, Tang Jia wei, Dr. Shaikh Nasir @ Nasir bin Shaikh Husin and Dr Jasmine Hau Yuan Wen for their support and technical advice. I would also like to thank my all fellow researchers Tei Yin Zhen, Lee Yee Hui, Tan Tze Hon, Loo Ling Kim, Loo Hui Ru and Lee Kher Li who had accompany and supported me through hard times during my study here.

I would also like to thank the developers of the utmthesis LaTeX project for making the thesis writing process a lot easier for me. Thanks to them, I could focus on the content of the thesis, and not waste time with formatting issues.

Finally, I would like to thank my family for always being there for me, through thick and thin. Especially my parents who never gave up believing in me. Their role in my life is something I will always nee and constantly appreciate.

Jeevan

ABSTRACT

Multiprocessor platforms have been introduced to solve the performance limitation of uni-processor platform. However, programming on a shared memory multiprocessor platform in an efficient way is difficult. The inefficiency of lock-based synchronization limits the performance of the parallel programs. Transactional memory (TM) provides a promising method in creating an abstraction layer for programmers to maximize hardware capacity of multiprocessor platform. Hardware TM (HTM) is faster compared to software TM although the performance of hardware transactional memory (HTM) is application-specific. Previous HTM implementations for embedded system were built on fixed version management which results in significant performance loss when transaction behaviour changes. In this thesis, a configurable version management HTM is proposed. The proposed version management is able to be configured to eager version management for low contention applications since it allows fast commit, or lazy version management that is suitable for applications with high contention since it can abort fast. In this work, an analytical model of the proposed hardware transactional processing time for different version management has been developed. With the analytical model, the bounds of the worst case and best case processing time can be estimated for a particular transaction size. The switching point of the performance between eager and lazy version management can also be estimated. The HTM has been prototyped and analyzed on Altera Cyclone IV platform. Based on our experiments, lazy version management is able to obtain up to 12.82% speed-up while eager version management obtains up to 37.84% speed-up on different memory request distributions for transaction sizes of 4, 8 and 16. The proposed HTM can be configured to obtain a shorter processing time for different types of applications compared to fixed version management.

ABSTRAK

Platform multipemproses telah diperkenalkan untuk meningkatkan prestasi platform unipemproses. Walau bagaimanapun, proses utuk membina pengaturcaraan cekap untuk multipemproses dengan ingatan sepunya adalah sukar. Ketidakcekapan penyegerakan berasaskan kunci menghadkan kecekapan program selari. Ingatan Transaksi (TM) mewujudkan lapisan abstrak untuk memudahkan pengaturcara membina aturcara yang cekap supaya kapasiti multipemproses dapat dimaksimumkan. Perkakasan TM (HTM) adalah lebih cepat berbanding dengan perisian TM walaupun prestasi perkakasan ingatan transaksi (HTM) adalah khusus atas satu-satu aplikasi. Pelaksanaan HTM sebelum ini untuk sistem terbenam dibina dengan pengurusan versi tetap mengakibatkan penurunan prestasi yang ketara apabila corak transaksi berubah. Dalam tesis ini, ingatan transaksi dengan pengurusan versi keboleh-konfigurasi adalah dicadangkan. Pengurusan versi bersemangat sesuai untuk aplikasi dengan kadar konflik yang rendah kerana masa yang diperlukan untuk menetapkan perubahan yang dilakukan oleh satu transaksi adalah singkat. Manakala, versi malas adalah sesuai untuk digunakan dengan aplikasi dengan kadar konflik yang tinggi kerana masa yang diperlukan untuk membatalkan perubahan yang dilakukan oleh satu transaksi adalah singkat. Model analisa berdasarkan perkakasan TM juga dibincangkan dalam tesis ini. Dengan model analitikal ini, masa maksimum dan minimum untuk menjalankan transaksi dapat dianggarkan. Titik pengalihan prestasi di antara versi malas dan versi semangat dapat dianggarkan. Ingatan transaksi ini di prototaip dan dianalisa pada platform Altera Cyclone IV. Berdasarkan eksperimen yang dijalankan, pengurusan versi malas menunjukkan peningkatan sehingga 12.82% kelajuan manakala versi pengurusan bersemangat menunjukkan peningkatan sehingga 37.84% kelajuan bagi aras konflik yang berlainan untuk saiz transaksi 4, 8 dan 16. HTM yang dicadangkan dapat dikonfigurasikan bagi mendapatkan masa pemprosesan yang lebih rendah berbanding pengurusan versi tetap.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

HTM	–	Hardware Transactional Memory
MPSoC	–	Multiprocessor System on Chip
STM	–	Software Transactional Memory

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

Multiprocessor System-on-Chip (Multiprocessor System on Chip (MPSoC)) consists of several general-purpose processors on in a single chip and shares hardware resources such as memory and input/output (I/O) pins. MPSoC offers lower latency, higher bandwidth and lower clock rate without losing throughput [1] compared to uni-processor systems. MPSoC has become a norm for servers, desktop, and embedded systems. However, programmers could not fully exploit parallelism potential of MPSoC parallel architecture as software industry still produce programs for uniprocessor. Application tasks are still handled sequentially, thus making the performance of MPSoC similar to uniprocessor for a single segmented application. In order to improve MPSoC throughput by running programs in parallel [1], memory architecture must also facilitate parallelism at atomic level.

Parallel programming model partitions tasks to parallel executables in parallel execution such that the time taken to complete the task can be made considerably low. However, maximizing parallel programming potential is not trivial, even for expert programmers [2]. Message passing and shared memory are the most common parallel programming models. Message passing models need explicit communication in which programmers are required to synchronize memory access. On the other hand, shared memory model requires blocking synchronization to maintain coherency among multiple threads. When a task acquires a lock for a specific memory segment, other tasks have to wait until that particular task has been completed and the memory segment is unlocked. Shared memory model are hardware supported, where everything occurs implicitly [1]. Software level management causes large overhead and programmers needs to know the memory segments modified. The scratch pad model is an example of methods which manage parallel access of memory. However, memory transfer need to be done beforehand, thus making it suitable for certain applications [2].

In general, shared memory MPSoC uses blocking synchronization. Blocking synchronization can be divided into fine grain lock and coarse grain lock. Fine grain locking gives better performance than coarse grain but at the cost of development time. On top of that, fine grain method requires programmers to explicitly develop applications using fine-grain locks. On the other hand, coarse grain locking is much easier to implement. However, large chuck of memory segment which are locked becomes unavailable to other threads and thus, potential parallelism could not be fully exploited. Fine grain lock requires multiple locks each for a memory segment to allow parallelism. However, this type of synchronization needs to be handled by the programmer. In short, synchronization on shared memory based on locks and mutual exclusion are difficult to scale [2].

A much simpler abstraction for MPSoC synchronization is by using transactions. A transaction is defined as a sequence of memory read and write that belong within a single thread. Transactional memory provides non-blocking wait-free synchronization on multiple threads accessing the same memory location. Its execution is atomic, isolated, durable and consistent. Each thread or processor will have to execute its task as transactions. When there is a conflict between two or more transactions, all transactions except the winner have to restart or abort. For a successful transaction, modification made by it becomes permanent and available for other transactions to modify at the end of the transaction. In the end, all these changes are updated for all transactions and become visible to the other threads or processors.

1.1 Problem statement

Currently, lock-based synchronization schemes are widely used for synchronizing MPSoC threads. The increasing application programming complexity has led on researches towards TM. Software Transactional Memories (STM) such as [3, 4] are flexible in-terms of size of transaction. However, their performance is inferior to hardware TM. The main tasks of transactional memory (TM): conflict detection, commit and abort; are done in software and cause high latency, making STM the bottle neck in MPSoC [1]. Thus, several hardware transactional memory (HTM) architectures have been proposed such as [5–10] which focus on high performance cache coherent systems.

Existing HTM architectures were implemented and tested in simulation environment to allow architectures that are more complex to be proposed without concerning about the underlying hardware implementation details. Another spectrum of HTM implementation are those works that focus on building HTM prototype on field programmable gate array (FPGA) platform. ATLAS [11] and Real Time Transactional Memory (RTTM) [12] focuses on prototyping HTM for embedded architecture. NetTM [13] HTM implementation is targeted for network applications. These HTM architectures are based on fixed configurations and their performance is highly dependent on the variant running applications.

Designing HTM for embedded system put consideration on energy consumption and simplified complexity of HTM design. EmbeddedTM [14] focuses on reducing power consumption. However, it also uses cache coherent protocols which adds significant resource overhead that is too complex for embedded applications [2]. CTM [2] has introduced a generic approach in building HTM for embedded system. In this design, HTM can be configured to either lazy or eager conflict management to suit to application demand (i.e., probability of conflict). Its architecture consists of a unified cache for all processors, eliminating the need for complex coherence protocol. However in [2], version management context was not fully exploited, where transactional memory cache inside CTM needs to update main memory one word at a time.

1.2 Objectives

The primary objective of this thesis is to prototype a hardware transactional memory for MPSoC system. The baseline architecture is an improvement on CTM [2] by embedding configurable version management mechanism. Hence, the proposed HTM version management can be configured between eager and lazy as each configurations is more suitable for different types applications. Specifically, this thesis proposes the following.

1. The first objective is to characterize the performance of different version management schemes based on the CTM architecture [2] to work with varying application behaviours (contention level). This is done through analytical modelling of HTM version management.
2. The second objective is to prototype configurable version management HTM architecture on FPGA. The proposed architecture is parametrizable and able to

initiate switching of version management at run time.

3. The third objective is to verify the proposed HTM performance. This includes comparison and analysis of the analytical model with results from simulation on hardware prototype.

1.3 Scope of Work

This thesis focuses on characterizing the effect of version management on the performance of HTM. Conflict management is kept constant using lazy conflict management to provide a fair comparison between both version managements. Different conflict management may result in different pathology to complete execution of program, thus producing varying performance [15].

HTM overflow handling mechanism is not implemented. Overflow in HTM happen when the number of transactions that need to be speculated is more then the available memory space. All transactions in the test cases in this thesis are designed to be within the bound of the memory hardware capacity.

The HTM prototype presented in this thesis uses only available memory blocks available on the FPGA device. There is no difference in access time for speculative memory or the main memory. The memory access speed up factor between the speculative memory and main memory is fixed at 1. The discussion on the effect of hierarchical memory is not included in this thesis.

There are no additional instructions implemented in for the proposed transactional memory. Processors access the proposed HTM at the instruction level. Similar with [2], the HTM is shared among different processors, where each processor has a local cache for instructions. By doing this, heterogeneous core with or without individual cache is able to use the HTM.

1.4 Contributions

The proposed HTM architecture is an improved architecture over [2]. A MPSoC platform using four Nios II processors has been developed to verify the functionality of the proposed HTM architecture against the STM library [16]. A configurable version management HTM architecture is proposed with the ability to switch its version management at run time. The proposed HTM is also parameterizable and is able to be configured based on varying application demand. An analytical model has been formulated to approximate the bounds of processing time for both eager and lazy version management. Using this analytical model, it is possible to determine the most suitable version management for a certain application.

1.5 Thesis Organization

The rest of the thesis is organized as follows.

Chapter 2 covers related works in literature and discusses important aspects of TM analysis. This chapter also explains the problems in producing efficient programs for MPSoC systems. The advantages and disadvantages of current parallel programming models are also discussed. This chapter then provides a detailed discussion on TM and existing TM architecture in hardware.

Chapter 3 provides the methodology for the work done presented this thesis. This chapter also includes the general approach in TM research, as well as the tools and platform used to model, prototyped, and verify the proposed TM. The final section in this chapter describes the method used for creating datasets for verification purposes.

Chapter 4 presents the characterization of the proposed HTM. An analytical model is proposed to characterize the bounds of HTM processing time. Based on the model, the relationship of HTM processing time and different application contention levels is analysed. The comparison between the mathematical model and the implementation model is also provided.

Chapter 5 provides an overview of the hardware TM architecture of the proposed HTM system and a detail description of the proposed FPGA SoC hardware prototyping at different abstraction levels. This chapter also discusses the design consideration to allow HTM to be deployed in real MPSoC embedded system. Besides, the HTM programming model is also shown at the end of in this chapter.

Chapter 6 summarizes the work presented in this thesis, re-stating the contributions to knowledge, and suggests directions for future research.

REFERENCES

1. Navazo, M. L. *Hardware Approaches for Transactional Memory*. M.sc. thesis. Departament d'Arquitectura de Computadors, Universitat Politecnica de Catalunya. 2008. URL `arco.e.ac.upc.edu/wiki/images/e/e3/Mlupon_msc.pdf`.
2. Kachris, C. and Kulkarni, C. Transactional memories for multi-processor FPGA platforms. *Journal of Systems Architecture*, 2011. 57(1): 160–168.
3. Shavit, N. and Touitou, D. Software transactional memory. *Proceedings of the 14th Annual ACM Symposium on Principles of Distributed Computing*. Ottowa, Ontario, Canada. 1995. 204–213.
4. Dice, D., Shalev, O. and Shavit, N. Transactional Locking II. *Proceedings of the 20th International Conference on Distributed Computing*. Berlin, Heidelberg: Springer-Verlag. 2006, DISC'06. ISBN 3-540-44624-9, 978-3-540-44624-8. 194–208. doi:10.1007/11864219_14. URL `http://dx.doi.org/10.1007/11864219_14`.
5. Hammond, L., Wong, V., Chen, M., Carlstrom, B. D., Davis, J. D., Hertzberg, B., Prabhu, M. K., Wijaya, H., Kozyrakis, C. and Olukotun, K. Transactional Memory Coherence and Consistency. *SIGARCH Comput. Archit. News*, 2004. 32(2).
6. Ananian, C. S., Asanovic, K., Kuszmaul, B. C., Leiserson, C. E. and Lie, S. Unbounded Transactional Memory. *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*. Washington, DC, USA. 2005. 316–327.
7. Yen, L. *Signatures in transactional memory systems*. Ph.d. dissertation. University of Wisconsin. 2009.
8. Shriraman, A., Dwarkadas, S. and Scott, M. L. Flexible Decoupled Transactional Memory Support. *Proceedings of the 35th Annual International Symposium on Computer Architecture*. Beijing, China. 2008. 139–150.
9. Lupon, M., Magklis, G. and González, A. A dynamically adaptable hardware transactional memory. *Proceedings of the 43rd Annual IEEE/ACM*

International Symposium on Microarchitecture. 2010. 27–38.

10. Titos-Gil, R., Negi, A., Acacio, M., Garcia, J. and Stenstrom, P. ZEBRA: Data-Centric Contention Management in Hardware Transactional Memory. *IEEE Transactions on Parallel and Distributed Systems*, 2014. 25(5): 1359–1369. ISSN 1045-9219. doi:10.1109/TPDS.2013.262.

11. Njoroge, N., Casper, J., Wee, S., Teslyar, Y., Ge, D., Kozyrakis, C. and Olukotun, K. ATLAS: A chip-multiprocessor with transactional memory support. *Proceedings of the Conference on Design, Automation and Test in Europe*. Nice, France. 2007. 3–8.

12. Schoeberl, M. and Hilber, P. Design and implementation of real-time transactional memory. *Proceedings of the 20th International Conference on Field Programmable Logic and Applications (FPL)*. Milan,Lombardy,Italy. 2010. 279–284.

13. Labrecque, M. and Steffan, J. G. NetTM: Faster and Easier Synchronization for Soft Multicores via Transactional Memory. *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. New York, NY, USA: ACM. 2011, FPGA '11. ISBN 978-1-4503-0554-9. 29–32. doi:10.1145/1950413.1950422. URL `http://doi.acm.org/10.1145/1950413.1950422`.

14. Ferri, C., Wood, S., Moreshet, T., Bahar, R. I. and Herlihy, M. Embedded-TM: Energy and complexity-effective hardware transactional memory for embedded multicore systems. *Journal of Parallel and Distributed Computing*, 2010. 70(10): 1042–1052.

15. Bobba, J., Moore, K. E., Volos, H., Yen, L., Hill, M. D., Swift, M. M. and Wood, D. A. Performance pathologies in hardware transactional memory. *Proceedings of the 34th Annual International Symposium on Computer architecture (ISCA)*. New York, NY, USA. 2007. 81–91.

16. Krizhanovsky, A. *Software Transactional Memory (STM) in GCC-4.7*. Technical report. 2014. URL `http://natsys-lab.blogspot.com/2012/05/software-transactional-memory-stm-in.html`.

17. Hennessy, J. L. and Patterson, D. A. *Computer architecture: A quantitative approach*. San Francisco, CA, USA: Elsevier. 2012.

18. Dagum, L. and Menon, R. OpenMP: An Industry-Standard API for Shared-Memory Programming. *IEEE Comput. Sci. Eng.*, 1998. 5(1): 46–55. ISSN 1070-9924. doi:10.1109/99.660313. URL `http://dx.doi.org/10.1109/99.660313`.

19. Casper, J., Oguntebi, T., Hong, S., Bronson, N. G., Kozyrakis, C. and Olukotun, K. Hardware Acceleration of Transactional Memory on Commodity Systems. *SIGPLAN Not.*, 2011. 46(3): 27–38. ISSN 0362-1340. doi:10.1145/1961296.1950372. URL `http://doi.acm.org/10.1145/1961296.1950372`.

20. Forum, M. P. I. *MPI: A Message-Passing Interface Standard Version 3.0*. Technical report. 2012. Chapter author for Collective Communication, Process Topologies, and One Sided Communications.

21. Marathe, V. J., Scherer, W. N. and Scott, M. L. Adaptive Software Transactional Memory. *Proceedings of the 19th International Conference on Distributed Computing*. Berlin, Heidelberg: Springer-Verlag. 2005, DISC'05. ISBN 3-540-29163-6, 978-3-540-29163-3. 354–368. doi:10.1007/11561927_26. URL `http://dx.doi.org/10.1007/11561927_26`.

22. Saha, B., Adl-Tabatabai, A.-R., Hudson, R. L., Minh, C. C. and Hertzberg, B. McRT-STM: A High Performance Software Transactional Memory System for a Multi-core Runtime. *Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. New York, NY, USA: ACM. 2006, PPoPP '06. ISBN 1-59593-189-9. 187–197. doi:10.1145/1122971.1123001. URL `http://doi.acm.org/10.1145/1122971.1123001`.

23. Spear, M. F., Michael, M. M. and von Praun, C. RingSTM: Scalable Transactions with a Single Atomic Instruction. *Proceedings of the Twentieth Annual Symposium on Parallelism in Algorithms and Architectures*. New York, NY, USA: ACM. 2008, SPAA '08. ISBN 978-1-59593-973-9. 275–284. doi:10.1145/1378533.1378583. URL `http://doi.acm.org/10.1145/1378533.1378583`.

24. Herlihy, M., Luchangco, V., Moir, M. and Scherer, W. N., III. Software Transactional Memory for Dynamic-sized Data Structures. *Proceedings of the Twenty-second Annual Symposium on Principles of Distributed Computing*. New York, NY, USA: ACM. 2003, PODC '03. ISBN 1-58113-708-7. 92–101. doi:10.1145/872035.872048. URL `http://doi.acm.org/10.1145/872035.872048`.

25. Herlihy, M. and Moss, J. E. B. Transactional memory: Architectural support for lock-free data structures. *SIGARCH Comput. Archit. News*, 1993. 21(2): 289–300.

26. Labrecque, M. and Steffan, J. G. The case for hardware transactional memory in software packet processing. *Proceedings of the 6th ACM/IEEE Symposium*

on Architectures for Networking and Communications Systems. La Jolla, California, USA. 2010. 37.

27. Damron, P., Fedorova, A., Lev, Y., Luchangco, V., Moir, M. and Nussbaum, D. Hybrid Transactional Memory. *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*. New York, NY, USA: ACM. 2006, ASPLOS XII. ISBN 1-59593-451-0. 336–346. doi:10.1145/1168857.1168900. URL `http://doi.acm.org/10.1145/1168857.1168900`.

28. Kumar, S., Chu, M., Hughes, C. J., Kundu, P. and Nguyen, A. Hybrid Transactional Memory. *Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. New York, NY, USA: ACM. 2006, PPoPP '06. ISBN 1-59593-189-9. 209–220. doi:10.1145/1122971.1123003. URL `http://doi.acm.org/10.1145/1122971.1123003`.

29. Lev, Y., Moir, M. and Nussbaum, D. PhTM: Phased transactional memory. *In Workshop on Transactional Computing (Transact), 2007. research.sun.com/scalable/pubs/TRANSACT2007PhTM.pdf*. 2007.

30. Shriraman, A., Marathe, V. J., Dwarkadas, S., Scott, M. L., Eisenstat, D., Heriot, C., III, W. N. S. and Spear, M. F. *Hardware Acceleration of Software Transactional Memory*. Technical report. DEPT. OF COMPUTER SCIENCE, UNIV. OF ROCHESTER. 2006.

31. Minh, C. C., Trautmann, M., Chung, J., McDonald, A., Bronson, N., Casper, J., Kozyrakis, C. and Olukotun, K. An Effective Hybrid Transactional Memory System with Strong Isolation Guarantees. *Proceedings of the 34th Annual International Symposium on Computer Architecture*. New York, NY, USA: ACM. 2007, ISCA '07. ISBN 978-1-59593-706-3. 69–80. doi:10.1145/1250662.1250673. URL `http://doi.acm.org/10.1145/1250662.1250673`.

32. Moore, K. E., Bobba, J., Moravan, M. J., Hill, M. D. and Wood, D. A. LogTM: Log-based transactional memory. *Proceedings of the 12th International Symposium on High-Performance Computer Architecture*. Austin, Texas, USA. 2006. 254–265.

33. Harris, T., Larus, J. and Rajwar, R. *Transactional Memory, 2Nd Edition*. 2nd ed. Morgan and Claypool Publishers. 2010. ISBN 1608452352, 9781608452354.

34. Magnusson, P. S., Christensson, M., Eskilson, J., Forsgren, D., Hållberg, G.,

Högberg, J., Larsson, F., Moestedt, A. and Werner, B. Simics: A Full System Simulation Platform. *Computer*, 2002. 35(2): 50–58. ISSN 0018-9162. doi: 10.1109/2.982916. URL `http://dx.doi.org/10.1109/2.982916`.

35. Martin, M. M. K., Sorin, D. J., Beckmann, B. M., Marty, M. R., Xu, M., Alameldeen, A. R., Moore, K. E., Hill, M. D. and Wood, D. A. Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset. *SIGARCH Comput. Archit. News*, 2005. 33(4): 92–99. ISSN 0163-5964. doi:10.1145/1105734.1105747. URL `http://doi.acm.org/10.1145/1105734.1105747`.

36. Pusceddu, M., Ceccolini, S., Tumeo, A., Palermo, G. and Sciuto, D. Emulating Transactional Memory on FPGA Multiprocessors. *Proceedings of the 24th International Conference on Architecture of Computing Systems*. Berlin, Heidelberg: Springer-Verlag. 2011, ARCS'11. ISBN 978-3-642-19136-7. 74–85. URL `http://dl.acm.org/citation.cfm?id=1966221.1966231`.

37. Grinberg, S. and Weiss, S. Investigation of Transactional Memory Using FPGAs. *Electrical and Electronics Engineers in Israel, 2006 IEEE 24th Convention of*. IEEE. 2006. 119–122.

38. Altera. User Guide Getting Started with Quartus II Simulation Using the ModelSim-Altera Software. 2014. URL `http://www.altera.com.my/literature/ug/ug_gs_msa_qii.pdf`.

39. Matlab. Matlab Primer. 2014. URL `http://in.mathworks.com/help/pdf_doc/matlab/getstart.pdf`.

40. System, G. O. *What is GNU?* Technical report. 2014. URL `https://www.gnu.org/`.

41. Xilinx. *LogiCORE IP Fast Simplex Link (FSL) V20 Bus (v2.11c)*. Technical report. 2010.

42. Dimitrakopoulos, G. and Kalligeros, E. Dynamic-priority Arbiter and Multiplexer Soft Macros for On-chip Networks Switches. *Proceedings of the Conference on Design, Automation and Test in Europe*. San Jose, CA, USA: EDA Consortium. 2012, DATE '12. ISBN 978-3-9810801-8-6. 542–545. URL `http://dl.acm.org/citation.cfm?id=2492708.2492843`.