International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2012): 3.358

# Implementation of Core-Lock mechanism as a Data Synchronization Method in Embedded Multi-core Systems

**Megha .S[1], Dr. C R Byrareddy[2]**

[1]M.Tech in VLSI Design and Embedded Systems, Bangalore Institute of Technology, Bangalore, Karnataka, India

[2]Associate Professor, Dept of ECE, Bangalore Institute of Technology, Bangalore, Karnataka, India

**Abstract:** ***Multi-core processors have become prevalent in the embedded systems for High-performance computations especially in the high-end digital applications. One of the major challenges in multi-core system is Data synchronization which facilitates the simultaneous execution of multiple threads in the same processor environment. Traditional methods solved the Data Synchronization issues using Lock based methods like semaphores or mutual exclusion of critical data. More advanced methods use transactional memory to achieve the same purpose. But there are advantages and disadvantages in both methods. So we propose a mechanism which exploits advantages of Traditional Lock based methods and evolving transactional memory methods. This Hybrid method will be termed as Core Locking (C-Lock) which is performance and energy efficient. C-Lock allows parallelism by detecting true conflicts and disables the clocks of the idle cores thereby minimizing the dynamic power consumption. This paper aims to implement the C-Lock manager using Verilog HDL, simulated using Cadence ncsim and synthesized using Cadence RTL compiler.***

**Keywords: Multi-core, Data Synchronization, embedded systems, energy, performance, Transaction memory (TM).**

## 1. Introduction

Innovations in semiconductor technology have led to evolution of processor architectures to enhance computation power and performance and also have managed to reduce the size of the processor chip by scaling. In accordance with Moore's law, this has resulted in chip speeds to rise and prices to drop. As the transistor components grow thinner, chip manufacturers have struggled to limit power usage and heat generation. Even performance enhancing approaches like running multiple instructions per thread have aged out. Due to this, the performance of the chip is falling short of meeting the application demands. In response, manufacturers are building chips with multiple energy-efficient processing cores instead of single powerful core [1][2].

### 1.1 Multi-core Processors

A multi-core processor is the technology that chip manufacturers are focusing on. The Multi-core chips don't necessarily run as fast as the highest performing single-core model, but they improve overall performance by handling more work in parallel, as shown in Fig1.

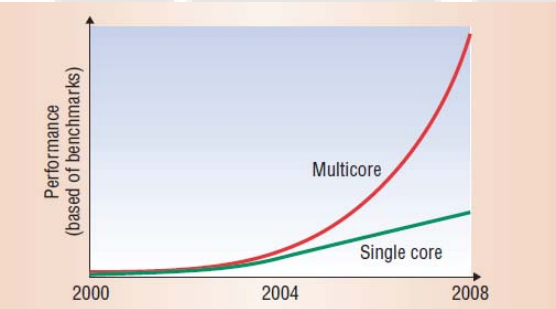


**Figure 1:** Performance of single and Multicore processors

However, adding more processing cores does not necessarily lead to a predictable gain in system performance due to the limited parallelism of real world programs. It also introduces many challenging problems in data handling and communication between processing cores and memory while multi-tasking [5][6]. Data synchronization is one of the main issues that must be addressed while dealing with any Multi-core systems.

### 1.2 Data Synchronization

Data Synchronization issues are related to Avoiding Conflicts in Resource access, Create Sequence of Operation, communicating between Processes when multiple tasks execute simultaneously. So when Data Synchronization is targeted, the objectives will be to overcome the issues like Task Co-Operation and Communication, Eliminating Competition for shared resource, Defining methods to access shared resource, Protecting Critical section Objects.

The data synchronization techniques which were originally developed for general purpose systems cannot be transferred directly into the embedded world since they do not take the nature of embedded systems in sufficient consideration: these include stringent requirements for low energy consumption as well as high performance.

### 1.3 Data Synchronization Methods

Traditional Data synchronization methods used Locking of Critical Section resource to protect shared memory areas. These methods included Semaphores and Mutual exclusions and went well when the complexity of applications was less demanding and there were no hard real time constraints to be met. With evolving complexity the methods aimed at data

synchronization also evolved to use Transactional memory approaches to provide Lock Free mechanisms. This is still evolving to meet the present day challenges. So there are methods which exploit advantages of Lock based and TM approaches to meet the application requirements. These methods use the main advantages of embedded system that applications can be implemented by Hardware architectures or Software based algorithms or a mixture of both.

## 2. Previous Works to Resolve Data Synchronization Issues

### 2.1 Lock-based Approach

1. **Speculative Lock Elision (SLE)**
   This is a hardware-based approach which elides the unnecessary lock-induced serialization from dynamic execution stream [4].
2. **Transactional Lock Removal (TLR)**
   This also uses hardware to convert lock-based critical sections transparently and dynamically into lock-free optimistic transactions [2][3].
3. **Synchronization-operation Buffer (SB)**
   This monitors the shared variable and, if it is changed, notifies the processor of the change so that the processors energy- and bandwidth consuming polling operation can be avoided. This uses Software application to control hardware resource [11].

Lock based schemes share the drawback of being overly conservative in their exploitation of parallelism. They operate at Process level and use scheduling algorithms. For this reason, a process cannot simultaneously run on two shared data elements if it has already requested a lock for one of them. To counter this problem, an identifier should be given to each shared data element rather than to a process, at the cost of increased programming complexity and increased delay in execution.

### 2.2 Lock-free or Transactional Memory (TM) Approach

TM provides sufficient programmability to the programmers by abstracting the details of the synchronization. Consequently, the programmers rather focus on the functionality. Even though TM simplifies the programming model and maximizes concurrency, transactions may suffer from interference which causes them to abort and from heavy overheads for memory accesses.

1. **Embedded-TM**
   This aims at balancing energy efficiency and simplicity in an embedded system. However, the energy efficiency of TM strongly depends on the accuracy of the speculation. However, when the speculation is wrong, it consumes non-negligible energy for the associated transaction abort and restart [12][13].
2. **Shutdown method**
   They dynamically turned off a processor by gating all its clocks, whenever any transaction running on the processor is aborted. TM has an advantage over locks in terms of energy consumption, but that this advantage largely depends on the architecture of the system, the contention level, and the conflict resolution policy [14].

### 2.3 Hybrid Approach

Hybrid approach combines the merits of both lock and TM.

1. **Adaptive locks**
   This is a hybrid method which dynamically selects TM or a mutex lock to improve performance. However, it only focuses on improving program execution time. That is, the system that allows speculative execution may cause a power consuming rollback operation. In addition, there is no power saving mechanism for the processors waiting for the execution of a critical section [15]. To summarize, the traditional lock-based schemes are inadequate from a performance perspective, while TM methods are not well designed from an energy perspective. For these reasons, it is necessary to design a data synchronization method which exploits the advantages of both methods. This paper discusses a Data Synchronization method called ***C-lock*** which is a performance and energy efficient method in Embedded Multi-core systems.

## 3. C-Lock Mechanism

C-Lock is a Hybrid approach that combines the advantages of both lock-based and lock-free methods for Data synchronization in Multi-core systems. The main aim of the C-Lock system is to exploit available parallelism by detecting true conflicts and to minimize the dynamic power consumption with clock gating for the idle cores. C-Lock detects conflicts using address ranges from different cores that wish to access the memory.

### 3.1 Concept of C-Lock mechanism

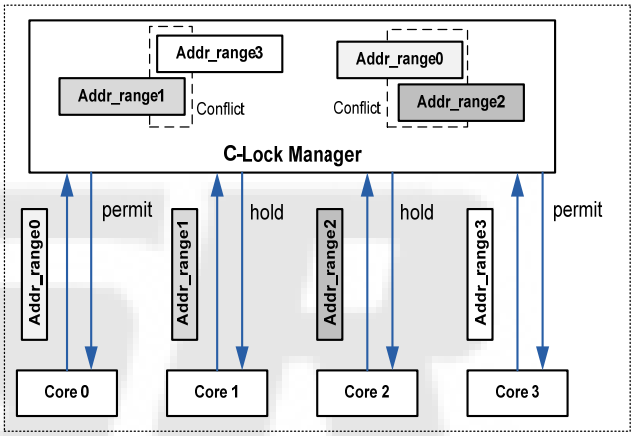


**Figure 2:** C-Lock mechanism

Fig2 shows how the address range is used for detecting true dependencies so as to decide whether to execute or hold the operation. If the cores are accessing different address regions of the memory, then the C-Lock detects no conflicts and thus all the cores can perform operation simultaneously. This is similar to TM operation. However, if the address range overlaps between different cores then the conflict exists and hence the system allows only one access at a time. Meanwhile, the cores without access permission move into the clock-gated state thereby reducing the dynamic power

consumption. In this way, C-Lock yields higher energy efficiency than TM and provides higher performance than Lock.

Fig2 shows the concept of C-Lock Mechanism. Before the execution of the critical section, every core sends the address range to be accessed; Addr range0 to Addr range3. After that, the centralized peripheral C-Lock Manager decides whether the ranges overlap or not. If there is an overlap, only one among the cores that cause conflict is permitted to run while the others are stalled with clock gating until the former ends the execution.

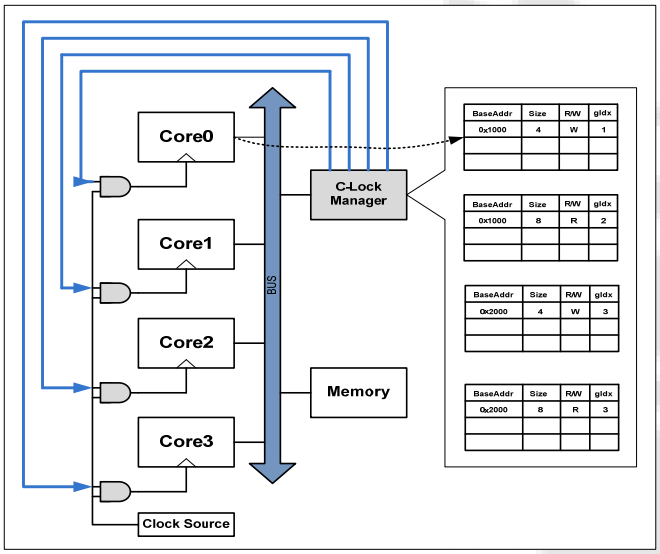


**Figure 3:** Top-level architecture of C-Lock

Fig shows the top-level architecture of C-Lock that consists of 4 cores, memory, C-Lock manager consisting of dedicated pools for each core and a bus for communication between core and memory.

### 3.2 Modification from the traditional lock scheme

**Hardware side:** C-Lock Manager is the key component of C-Lock which is in charge of detecting true conflicts among the accesses to the shared data, and controlling clock-gating of the cores.

**Software side:** Each core is in charge of setting the necessary information to C-Lock Manager, which includes base address, size, and type of the data it intends to access. When this information is set, the core is allowed to attempt its atomic operation by notifying C-Lock Manager.

### 3.3 Implementation of C-Lock Manager

In this paper, we assume that there are 4 cores in the processor and each core can record 4 items with the C-Lock Manager. Item is a storage element that contains access information for checking the conflicts. Each Item consists of following fields:

1. Base address
2. Access Size
3. Read/Write operation
4. gIdx- Global Index for conflict detection.
5. Valid field

The internal architecture of C-Lock Manager consists of 4 pools for a dedicated core, an arbiter, a global counter, 4 item buses and signals for detecting conflicts.

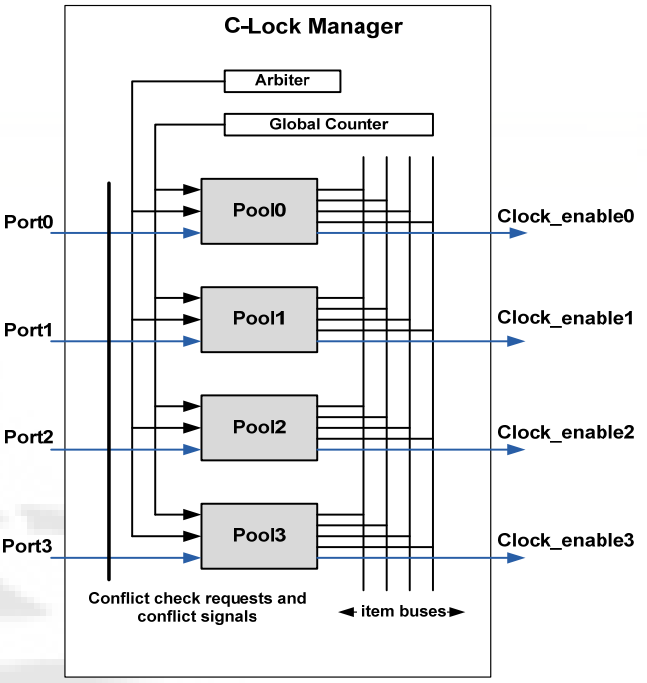


**Figure 4:** Internal architecture of C-Lock Manager

**Arbiter:**

A memory arbiter is a device used in a shared memory system to decide, for each memory cycle, which core will be allowed to access that shared memory. Arbiter can inspect the possible deadlock. When the arbiter receives the C-Lock Id from the request for an atomic operation of the Pool, it allows only one C-Lock Id to be acquired by the cores within the group.

**Global Counter:**

Global counter keeps track of the items registered by allotting a gIdx value for each item depending on the order it is registered. Once the Pool gets the grant for access, it sets the gIdx fields of the newly registered Item entries to the current global index values which is broadcasted by the global counter. At the same time, the granted Pool signals the global counter to increment the global index value.

**Pool:**

Each pool consists of registered items from the corresponding core, conflict checker and logic for clock enable and arbiter request. The pool broadcasts all the items to the item buses and requests the other pools to check for conflicts by comparing the broadcasted items with their own registered items. Immediately after, the conflict checking process is performed in the other pools. Fig5 shows the architecture of the pool.

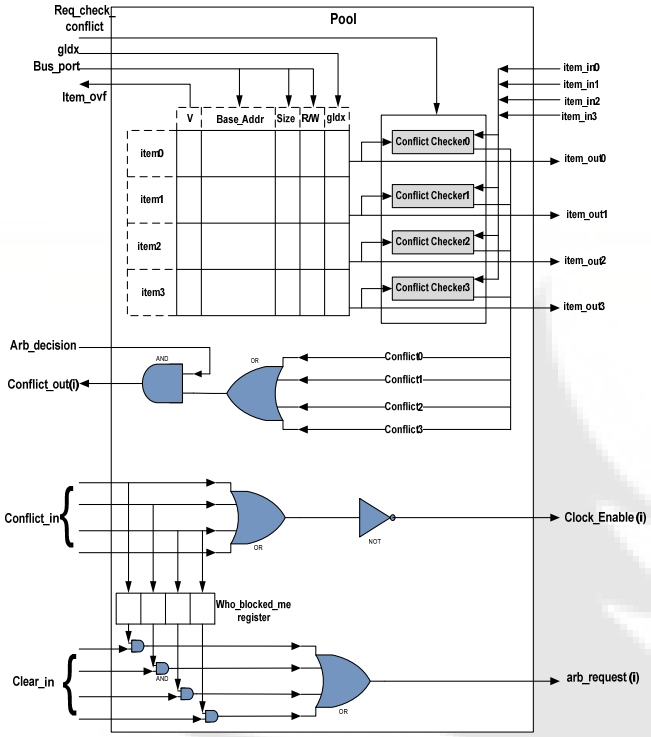


**Figure 5:** Architecture of each Pool

### 3.4 Operation of C-Lock Manager

Conflict checking is done by the conflict checker block. A conflict checker is dedicated to an Item entry and checks whether any of the broadcasted Items cause true conflicts with its own Item.

If Itemloc is the Item to which the conflict checker is dedicated (loc stands for local), and Itemrem is one of the broadcasted Items (rem stands for remote). Then, Itemloc and Itemrem have true conflicts if the following conditions are simultaneously present:

- Both Items are valid
- Their address ranges overlap
- At least one of them is a write operation
- gIdx of Itemloc is smaller than gIdx of Itemrem

The third one filter out the false dependency (Read-after-Read).The fourth condition detects possible data hazard. If the fourth condition holds, it means that the Itemrem is registered later than Itemloc and, therefore, executing Itemrem prior to Itemloc may cause data hazard in the requested memory region.

Each conflict checker performs the above operation for all the broadcasted Items and finally produces out the conflict signal by simply pairwise ORing the results. Again, by ORing all the conflict signals from the conflict checkers, the Pool finally makes the signal which indicates whether any of the broadcasted Items are in conflict with the Items in this Pool. The signal is AND gated with the arb decision signal to output the final conflict out signal.

After that, the Pool which requested conflict checks from the other Pools gathers the results by watching the conflict in signals in Fig. 5. If any of the other Pools reports conflict, it means the requested atomic access cannot be executed at this time, and therefore, the Pool disables the clock of the corresponding core.

Also, the conflict in signals is stored in the “who-blocked-me" register so that the Pool can watch the events of the blocking Pools being cleared and reattempt its access. This can effectively avoid the blocked Pools watching the activities from all the other cores. When no conflicts are reported from the other Pools, the core keeps running and executes the atomic access for the registered Items.

### 3.5 C-LOCK Algorithm

The interaction between the cores during C-Lock operation is as illustrated in the flowchart Fig6.

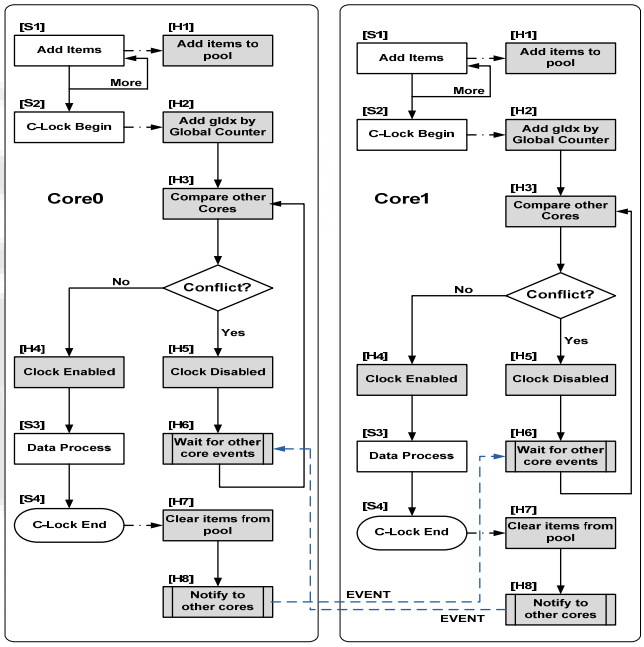


**Figure 6:** Interaction of cores in C-Lock operation

The transaction of the core with the memory is implemented using FSM as shown in Fig7.

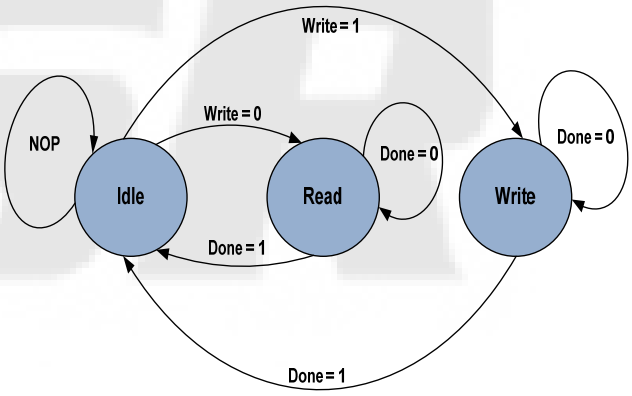


**Figure 7:** FSM for interaction of the core with the memory

Fig7 represents the 2 possible states of Transaction memory operation used to implement C-Lock manager. When there is no conflict and core is clock enabled read/write operations will be performed. Once operation is completed the core

updates required registers and the pool is cleared to make way for new items. The core asserts C-Lock manager by setting flag (Done) and accordingly true conflicts between cores are determined for next cycle. In the event of a True Conflict the core is clock gated and remains Idle till clock enabled by C-Lock manager. Thus C-Lock mechanism improves performance and conserves energy over Speculative Lock Elusion method of Lock free TM methods and Lock based methods. The results will be discussed in the coming section.

## 4. Results

### 4.1 Simulation results

We implemented C-Lock Manager with Verilog HDL, to analyze the hardware overhead of the proposed scheme. Simulation and Synthesis of the implementation were analyzed using Cadence Tool Suites using a 180nm technology library - NC Sim for Simulation and RTL Compiler for Synthesis.

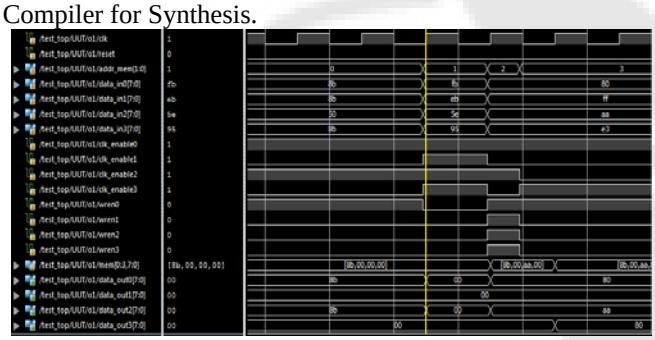

**Figure 8:** Output buffer signals

Fig8 shows output buffer signals when the read and write operations are carried out in varying (and also conflicting) address ranges for different cores.

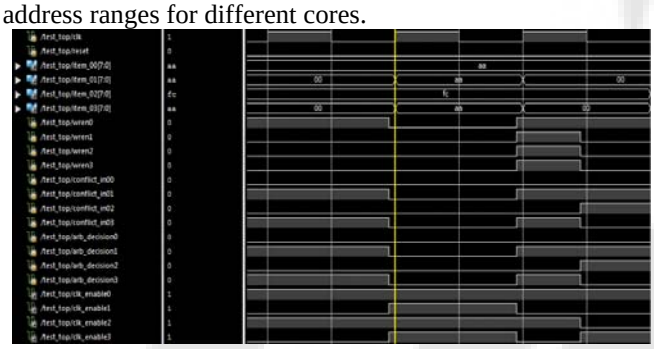

**Figure 9:** Read without conflict

Fig9 shows that read operation does not require a conflict detection and all cores have access to perform read operation even on shared address ranges.

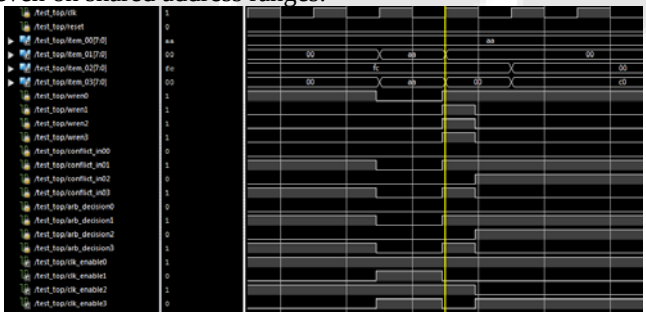

**Figure 10:**Write with conflict

In Fig10 all the cores are performing wrte operation, but C-Lock manager has identified conflict in Core1 and Core3 with respect to other cores. So Core1 and Core3 are clock gated and remain Idle till C-Lock manager Clock Enables the cores.

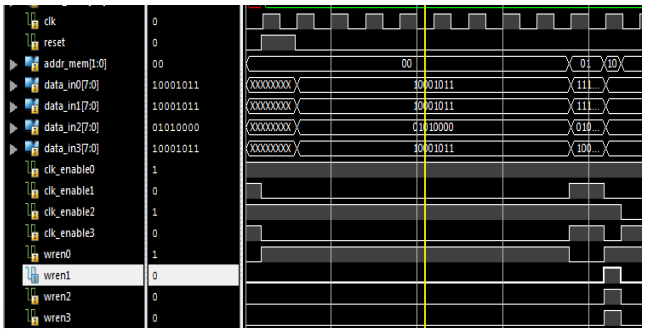


**Figure 11:** Write and read with conflict

In Fig11 Core 0 is performing a write operation and Core 2 is enabled to perform only read operation on the requested address ranges. Here no conflict is identified by C-Lock manager between Core0 and Core2.

### 4.2 Synthesis results

A synthesis of C-Lock Manager implementation yielded favorable results for evaluation purpose. We implemented two methods in C-Lock manager where in a state machine was used to model control unit to implement Transaction memory states and other was implemented without using FSM. A basic performance analysis of both methods in terms of Speed, Area and Power are tabulated in Table1. Fig 12 shows the RTL Schematic generated in Xilinx design Suite.

**Table 1:** Basic Performance evaluation

| *Design* | *Leakage Power(nW)* | *Dynamic Power (mW)* | *Cell area* |
|---|---|---|---|
| Without FSM | 26.416 | 2.24 | 7531 |
| FSM | 26.416 | 2.22 | 7531 |

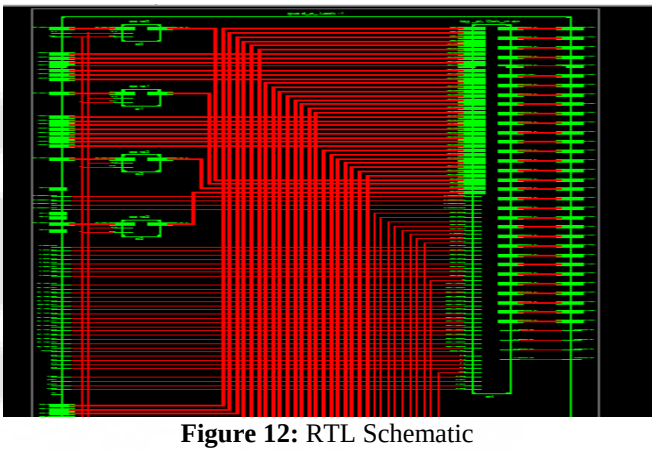

**Figure 12:** RTL Schematic

To get the actual gauge of performance enhancements using C-Lock method, processor benchmark evaluation tools can be used. The benchmark applications can be choose from the STAMP benchmark suite, the MiBench suite, thread scheduling model (TSM), and microbenchmarks for MPARM.

## 5. Conclusion

A C-Lock manager is implemented using Verilog HDL and is programmed with the objectives of using the hardware resource and software algorithms to overcome the data synchronisation issues in a multi-core multiprogramming architecture. The effectiveness of the proposed scheme is in exploiting available parallelism with low power consumption. In detail, C-Lock prevents unnecessary exclusive execution using the access address range comparison and the system does not perform a power-wasting speculative execution. Also, the clock-gating feature reduces the dynamic power of the cores that are not granted for a critical section access. The proposed scheme may require significant work from the programmer; this can be regarded as a trade-off of the improved performance including power efficiency. On the other hand, the hardware area overhead and the power and execution time overhead of the proposed approach are not significant compared to performance enhancements. The high efficiency of C-Lock relies mostly on the special hardware C-Lock Manager, with marginal support from the software.

## References

[1] D. Geer, **"Chip Makers Turn to Multicore Processors",** *Computer*, vol. 38, no. 5, pp. 11–13, 2005.
[2] M. Levy and T. Conte, **"Embedded Multicore Processors and Systems",** *Micro, IEEE*, vol. 29, no. 3, pp. 7–9, 2009.
[3] M. Herlihy and J. E. B. Moss, **"Transactional Memory: Architectural Support For Lock-free Data Structures",** in Proc. 20th Int'l Symp. on Computer Architecture (ISCA '93), 1993, pp. 289–300.
[4] R. Rajwar and J. Goodman, **"Transactional Lock-Free Execution of Lock-Based Programs",** in Proc. 10th Int'l Conf. on Architectural Support for Programming Languages and Operating systems. ACM, 2002, pp. 5–17.
[5] R. Rajwar and J. Goodman, **"Speculative Lock Elision: Enabling Highly Concurrent Multithreaded Execution",** in Proc. 34th Annual ACM/IEEE Int'l. Symp. on Microarchitecture. IEEE Computer Society, 2001, pp. 294–305.
[6] M. Monchiero, G. Palermo, C. Silvano, and O. Villa, **"Efficient Synchronization for Embedded On-Chip Multiprocessors",** Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, vol. 14, no. 10, pp. 1049–1062, 2006.
[7] J. Li, J. F. Martinez, and M. C. Huang, **"The Thrifty Barrier: Energy-Aware Synchronization in Shared-Memory Multiprocessors,"** in Proc. Int'l Symp. on IEEE High-Performance Computer Architecture, vol. 10, 2004, pp. 14–23.
[8] L. Hammond, V. Wong, M. Chen, B. Carlstrom, J. Davis, B. Hertzberg, M. Prabhu, H. Wijaya, C. Kozyrakis, and K. Olukotun, **"Transactional Memory Coherence and Consistency,"** in ACM SIGARCH Computer Architecture News, vol. 32, no. 2. IEEE Computer Society, 2004, p. 102.
[9] C. Ananian, K. Asanovic, B. Kuszmaul, C. Leiserson, and S. Lie, **"Unbounded transactional memory,"** in Proc. 11th Int'l Symp. On High-Performance Computer Architecture (HPCA '05), 2005, pp. 316–327.
[10] K. Moore, J. Bobba, M. Moravan, M. Hill, and D. Wood, **"LogTM: Log-Based Transactional Memory,"** in Proc. 12th Int'l Symp, On High-Performance Computer Architecture, 2006, pp. 254–265.
[11] M. Monchiero, G. Palermo, C. Silvano, and O. Villa, **"Power/performance hardware optimization for synchronization intensive applications in mpsocs,"** in Proc. Design, Automation and Test in Europe, vol. 1, 2006.
[12] C. Ferri, T. Moreshet, R. Bahar, L. Benini, and M. Herlihy, **"A Hardware/Software Framework for Supporting Transactional Memory in a MPSoC Environment,"** ACM SIGARCH Computer Architecture News, vol. 35, no. 1, pp. 47–54, 2007.
[13] C. Ferri, S. Wood, T. Moreshet, R. Iris Bahar, and M. Herlihy, "**Embedded-tm: Energy and complexity-effective hardware transactional memory for embedded multicore systems,"** Journal of Parallel and Distributed Computing, vol. 70, no. 10, pp. 1042–1052, 2010.
[14] S. Sanyal, S. Roy, A. Cristal, O. S. Unsal, and M. Valero, **"Clock Gate on Abort: Towards Energy-Efficient Hardware Transactional Memory,**" in Proc. IEEE Int'l Symp. on Parallel and Distributed Processing, 2009.
[15] T. Usui, R. Behrends, J. Evans, and Y. Smaragdakis, **"Adaptive Locks: Combining Transactions and Locks for Efficient Concurrency,"** Journal of Parallel and Distributed Computing, vol. 70, no. 10, pp. 1009–1023, 2010.

## Author Profile

**Megha.S** received her B.E in Electronics and communication from Visvesvaraya Technological University (VTU) in 2010. She worked at M/S Mindtree for 2 years from July 2012 - October 2014 in the field of Digital Board design. Currently, she is pursing M.Tech in VLSI and Embedded systems from Bangalore Institute of Technology, VTU. She is also working as Intern in the field of ASIC verification at LSI, Bangalore.

**Dr. C.R. ByraReddy** received B.E in Instrumentation Technology from RV College of Engineering, Bangalore University in 1991, and M.E from UVCE, Bangalore University in 1999. He hold Ph.D from Sri Venkateswar University, Tirupathi. He has both academic and industry experience in various research fields. He is currently working as Associate Professor in Dept of ECE, Bangalore Institute of Technology, Bangalore. He has published 9 research papers in National and International conferences.