International Journal of Ethics in Engineering & Management Education
Website: www.ijeee.in (ISSN: 2348-4748, Volume 1, Issue 5, May2014)

PERFORMANCE-EFFICIENT DATA SYNCHRONIZATION ARCHITECTURE FOR MULTI-CORE SYSTEMS USING C-LOCK

Aswathy Surendran
The Oxford College of Engineering, Bangalore
aswathysurendran4u@gmail.com

Abstract: **Data synchronization among multiple cores has been one of the critical issues which must be resolved in order to optimize the parallelism of Multi-core architectures. Data synchronization schemes can be classified as lock-based methods and lock-free methods. However, none of these methods consider the nature of embedded systems which have demanding and sometimes conflicting requirements not only for high performance but also for low power consumption. As an answer to these problems, here proposes C-Lock, an energy- and performance-efficient data synchronization method for Multi-core embedded systems. C-Lock achieves balanced energy- and performance-efficiency by combining the advantages of lock-based methods and transactional memory (TM) approaches; in C-Lock, the core is blocked only when true conflicts exist (advantage of TM), while avoiding roll-back operations which can cause huge overhead with regard to both performance and energy (advantage of locks). Also, in order to save more energy, C-Lock disables the clocks of the cores which are blocked for the access to the shared data until the shared data become available.**

1. INTRODUCTION

In modern computer systems multiple core systems have become prevalent, not only for high performance desktops or servers but also for many application such as mobile devices. Multi-core means single computing component with 2 or more independent actual cpu's (called cores), that read execute program instructions. The instructions are ordinary CPU instructions such as add, move data etc., but the multiple cores can run multiple instructions at the same time, increasing overall speed for programs .Multi-core can run multiple instructions simultaneously.

In order to meet the increasing demands for higher performance, increasing CPU clock frequency was one of the most obvious methods in traditional processors. However, for single cores, this is turning out to be impractical due to prohibitive power and heat dissipation requirements. This limitation made the multi-core approach a more viable and scalable solution to the performance demands of embedded systems. In fact, contemporary embedded systems, especially high-end products such as smartphones, are rapidly adopting multi-core chips at their core. Nowadays data synchronization among multi-core A system is a critical issue. It must be resolved in order to optimize the parallelism. The data synchronization issue comes into picture when two or more processors are trying to access any shared data simultaneously.

Present data synchronization methods can be classified as either lock-based or lock-free. The former includes locks, semaphores, and barriers; it blocks the accesses to the shared data from the processors which fail to acquire the permission. The latter allow all processors to access the shared data in an optimistic manner, and then perform rollback and re-execution when a conflict occurs.

But there are some drawbacks associated with these methods. Lock based methods are widely used because of their simple control mechanism, but they sacrifice much parallelism, which results in poor performance. Lock- free methods such as Transactional Memory approach perform speculative execution which might turn out to be wasteful of energy when the execution must be rolled back. In such cases, the rollback operation consumes additional energy. Here proposing a novel solution to this problem- C-Lock, an energy- and performance-efficient data synchronization method for embedded systems. C-Lock delivers TM-like parallelism in race conditions by detecting true data conflicts. The domain area here is System-on- Chip (SOC). SoC is an integrated circuit that integrates all components of a computer or other electronic system into a single chip.

2. MOTIVATIONAL EXAMPLES

2.1 Transactional Memory Approach

It is providing enough programmability to the programmers. Transactions suffer from interference and it makes to abort and from heavy overheads for memory access. In terms of energy consumption, transactional memory (TM) is better than Lock based method. But it is majorly influenced by the architecture of the system. Ferry[8] called the hardware TM as embedded TM. It is characterized by energy efficiency and simplicity. Accuracy of speculation is more affected by energy efficiency. If speculation is wrong, then non-negligible energy consumption will be the result. To overcome this

International Journal of Ethics in Engineering & Management Education
Website: www.ijeee.in (ISSN: 2348-4748, Volume 1, Issue 5, May2014)

hurdle, a method called shut down method is proposed. In this method, when any transaction running is aborted, it turns off the processor. It is done by gating all its clocks.

2.2 Lock based Approach

Speculation Lock Elision (SLE) [6] is a hardware based approach. It is a technique to remove dynamically unnecessary lock induced serialization. It enables highly concurrent execution of multiple threads. Whenever the data conflicts happen, to acquire the lock the corresponding threads are restarted.

Transactional Lock Removal also uses hardware in the conversion from lock based critical sections into lock free optimistic transactions. It makes use of time stamps for resolving conflicts. Its advantages are improved programmability, performance and high stability.

2.3 Hybrid Approach

By combining the advantages of Lock and TM, another approach can be introduced, called Hybrid Approach. One method belongs to this approach is Adaptive Locks [9]. For better performance, it dynamically selects lock method or transactional memory approach. The main focus of adaptive lock is on improving program execution time. Additional adaptive logic is needed for introducing these adaptive locks. Major drawback of this approach is, it is not having any power saving mechanism.

In summary, we can say that TM[7] Methods are not well designed from energy perspective. On the same way traditional lock schemes are inadequate from a performance perspective. So it necessitates the need of a novel approach for data synchronization which combines the advantages of all the above approaches, i.e. C-Lock (Core-Lock). But C-Lock is having a unique approach i.e. normally behaves like lock scheme for energy efficiency, but it shows a transactional behavior for checking data conflicts. It does clock-gating the stalls for power saving.

3 CORE LOCK (C- Lock)

The data synchronization issue arises when two or more processors access any shared data simultaneously. Mishandling of these conflicts results in incorrect operations and cause fatal errors. Some problems occurred are: Communication overhead, Performance overhead, Power consumption issues etc. In the burst waited stage the requesting processors are tied up sending out polling messages. Additional contention may lead to deadlock conditions that require extra mechanism for deadlock prevention which further degrade system performance. Until the request is obtained the requesting processors need to continuously place the lock requests on the system bus. In a shared memory multi-processor with spin lock synchronization, the no. of synchronization requests grows nonlinearly with no. of contending process making the system not scalable.

As an answer to these data synchronization problems, here proposing a new approach called C-Lock (Core Lock). It is a new performance -efficient data synchronization method for multi-core embedded systems. It achieves balanced energy- and performance-efficiency. The main idea of the C-Lock system is to exploit available parallelism with true conflict detection and to minimize dynamic power consumption with clock gating for the idle cores. It combines the advantages of both lock-based methods and transactional memory approaches. It delivers TM-like parallelism in race conditions by detecting true data conflicts. The detection is done by considering the type, address range, and dependency of simultaneous accesses. In those cases when true data conflicts are detected, the cores which are not given permission to access the data are immediately clock-gated in order to minimize the dynamic power consumption. Since no speculative execution and rollback are performed, C-Lock [1] results in higher energy efficiency than TM. Also, due to the immediate clock-gating of cores, C-Lock can consume less energy than lock-based methods.

FSM (Finite State Machine) logic is associated with each core. The Clock performance can be enhanced with the finite state machine logic. This Finite State Machine associated with each core will control the current and next states of cores to enable write / read transaction over shared memory. The formal model of a communicating finite state machine plays an important role in three different areas of p design: formal validation, protocol synthesis, and conformance testing. A finite-state machine, or FSM[4],[5] for short, is a model of computation based on a hypothetical machine made of one or more states. Only a single state can be active at the same time, so the machine must transition from one state to another in order to perform different actions. FSMs are commonly used to organize and represent an execution flow. Basically a FSM consists of combinational, sequential and output logic. Combinational logic is used to decide the next state of the FSM; sequential logic is used to store the current state of the FSM. The output logic is a mixture of both combinational and sequential logic. It is a mathematical model of computation used to design both computer programs and sequential logic circuits. It is conceived as an abstract machine that can be in one of a finite number of states. The machine is in only one state at a time; the state it is in at any given time is called the current state. It can change from one state to another when initiated by a triggering event or condition; this is called a transition. A particular FSM is defined by a list of its states, and the triggering condition for each transition.

The main idea of the C-Lock system is to exploit available parallelism with true conflict detection and to minimize dynamic power consumption with clock gating for the idle cores. Before the execution of the critical section, every core sends the address range to be accessed. After that, the centralized peripheral C-Lock Manager decides whether the ranges overlap or not. If there is an overlap, only one among

International Journal of Ethics in Engineering & Management Education
Website: www.ijeee.in (ISSN: 2348-4748, Volume 1, Issue 5, May2014)

the cores that cause conflict is permitted to run while the others are stalled with clockgating until the former ends the execution.

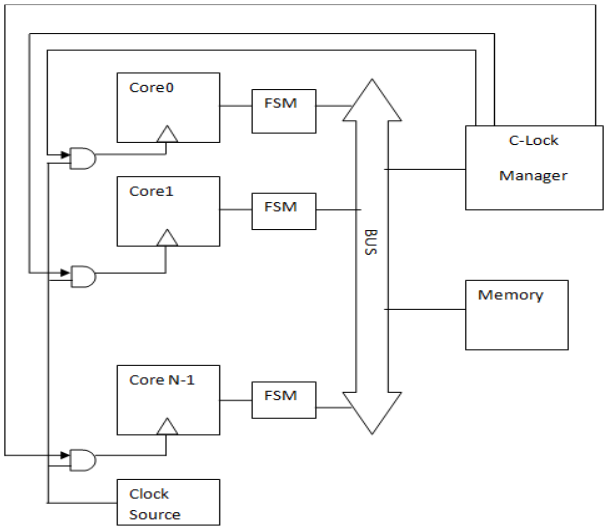

Fig. 1 Top Level Architecture

The major modification from the traditional lock schemes is that, on the hardware side, an additional peripheral called Finite State Machine (FSM) is added to each core. It helps in greater controllability over synchronization. The Clock performance can be enhanced with the finite state machine logic. FSM for each core will control the current and next states of cores to enable write / read transaction over shared memory.

There are two types of state machines.1) Mealy State Machine 2) Moore State Machine. A Moore FSM is a state machine where the outputs are only a function of the present state. A Mealy FSM is a state machine where one or more of the outputs is a function of the present state and one or more of the inputs.

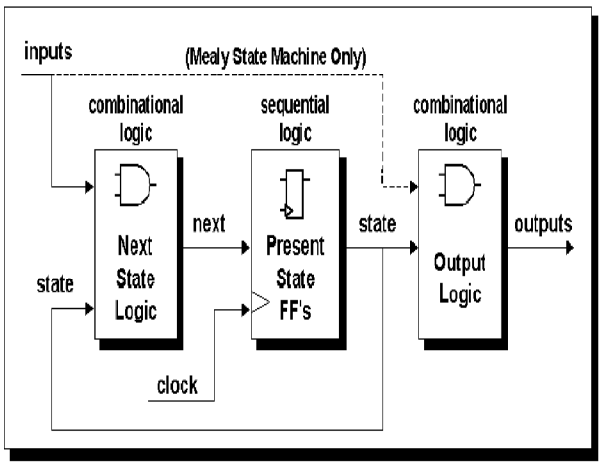

Fig. 2 Block Diagram

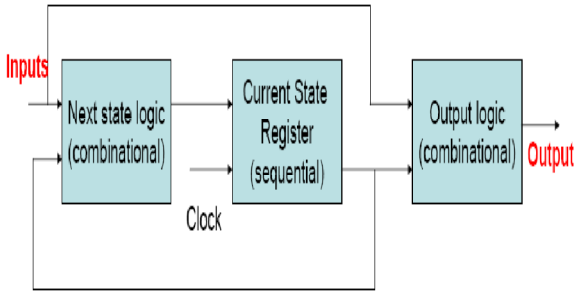

Fig. 3 Simple model of FSM

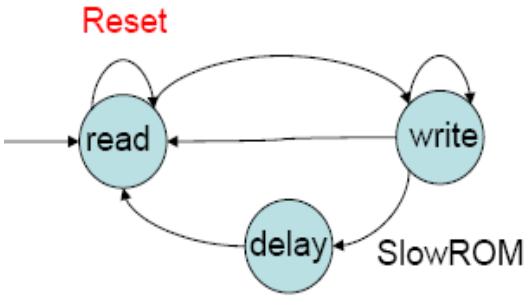

Fig. 4 State Diagram of FSM

The simple model and state diagram of FSM are given in figure. A finite state machine contains a finite number of states and produces outputs on state transitions after receiving inputs. Finite state machines are widely used to model systems in diverse areas, including sequential circuits, certain types of programs, and, more recently, communication protocols.

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. 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. In the next step, C-Lock Manager initiates the conflict detection routine and, in case of a conflict, grants permission to only one of the cores while gating the other cores which intend to access the data. Note that multiple cores can get the permissions if the accesses are not involved in any true conflict. After the core which has obtained the permission completes its atomic access, it notifies C-Lock Manager to release the permission. This command also triggers C-Lock Manager's conflict detection routine. C-Lock Manager gives permission to another core by de-asserting the corresponding clock gating signal.

Assume that there are *N* cores in the processor, and that each core can store *M* Items with C-LockManager. Item is a storage that contains information for checking true conflicts with the accesses of other cores. One Item consists of the following fields:

International Journal of Ethics in Engineering & Management Education
Website: www.ijeee.in (ISSN: 2348-4748, Volume 1, Issue 5, May2014)

- BaseAddr: base address
- Size: access size
- R/W: read/write
- gIdx : global index for conflict detection
- *V:* one bit valid field for indication of the validity of *Item*

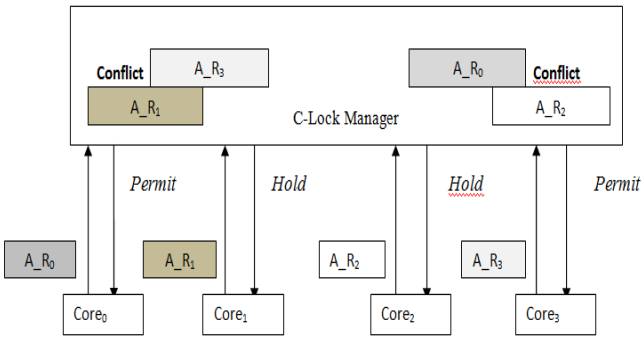

Fig. 5 Concept of C-Lock Mechanism

The internal architecture of *C-Lock Manager* is shown in Fig. It is composed of *N Pools*, *M Item busses*, a *global counter*, an *arbiter*, and a couple of signals among the *Pools* for the purpose of detecting conflicts (signals for requesting conflict check to the other *Pools*, and for responding to the requests). True conflicts occur
If the following conditions are simultaneously present: Both *Items* are valid, their address ranges overlap, At least one of them is a write operation.

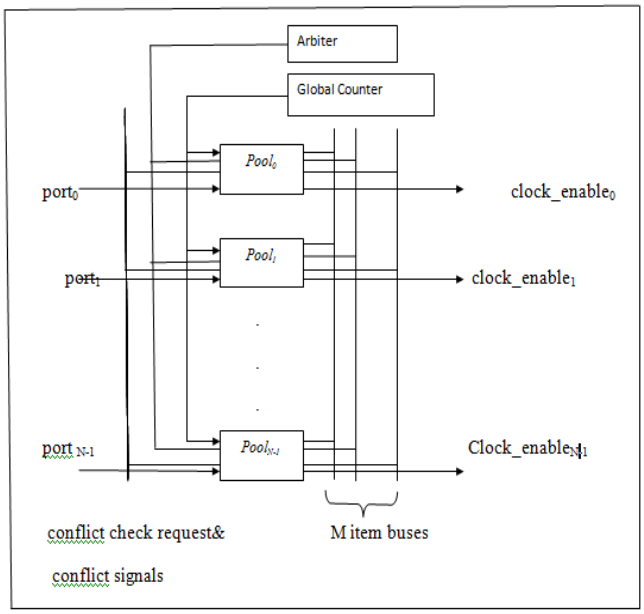

Fig. 6 C-Lock Manager Internal Architecture

4 CORE-LOCKS MANAGER- OPERATION

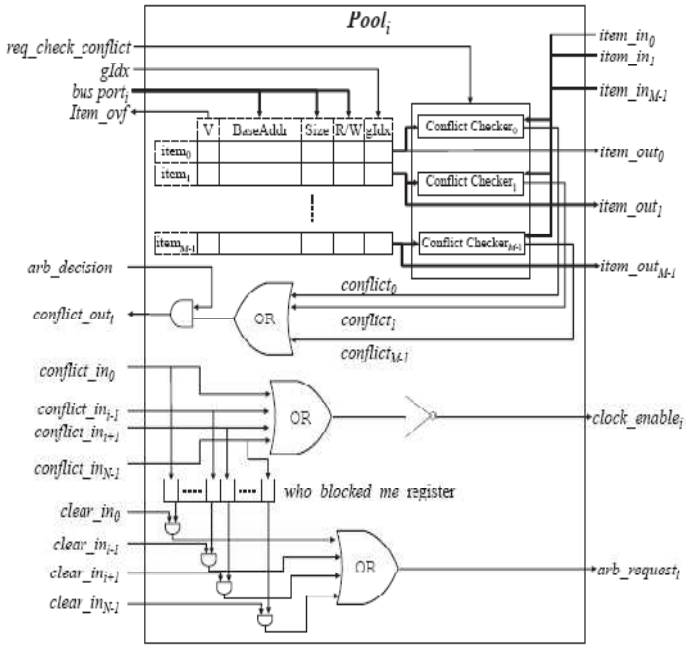

Fig.7 Pool Architecture

Pool is the main part of C-Lock Manager: It consists of *M Item* entries and conflict checker and clock-gating logics. Architecture each of Pool is shown in Fig. 6. Each Core initiates the C-Lock operation by recording the access information to the corresponding Pool .Each core can register at most *M Item*s. Here the handling logic of Pool manages the status of the entries by checking the valid fields thus put the incoming *Item* to an empty entry. Therefore, the program does not need to identify which *Item* entry it is accessing.

The access is triggered when the core sends the begin command to the corresponding *Pool* through the bus. Then, from the arbiter the Pool requests a grant to the conflict checking operation. This procedure is necessary since multiple cores can trigger their atomic accesses at the same time if C-Lock Manager is connected via multiple buses. If the *Pool* gets the grant, it sets the gIdx fields of the newly registered *Item* entries to the current global index values which is broadcasted by the global counter and the granted Pool signals the global counter to increment the global index value simultaneously. After that, the Pool broadcasts all the *M Item*s to the item buses and requests the other Pools to check for conflicts by comparing the broadcasted *Item*s and their own registered *Item*s. Immediately after, the conflict checking process is done in the other Pools. The major part of this process is done by the conflict checker. A conflict checker is dedicated to an *Item* entry and checks whether any of the *M* broadcasted *Item*s causes true conflicts with its own *Item*. True conflicts is detected when both *Item*s are valid, when their address ranges overlap and there should be at least one of them is a write operation.

International Journal of Ethics in Engineering & Management Education
Website: www.ijeee.in (ISSN: 2348-4748, Volume 1, Issue 5, May2014)

Each conflict checker performs these operation for all the broadcasted *Item*s and finally produces out the conflict signal by simply ORing the results. By ORing all the conflict signals from the conflict checkers again, the Pool finally makes the signal which indicates whether any of the broadcasted *Item*s are in conflict with the *Item*s in this Pool. The signal is ANDed with the arb_decision signal to output the final conflict_out signal.

Then, the Pool which requested conflict checks from the other Pools collects the results by noticing the conflict_in signals. If any of the other *Pool*s shows conflict, it means the requested atomic access cannot be triggered at this time, and therefore, the Pool disables the clock of the corresponding core. And, the conflict_in signals are stored in the who blocked me register. So Pool can watch the events of the blocking Pools being cleared and retry its access. This can surely avoid the blocked *Pool*s 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 *Item*s. Each *Pool* has its own *Item*s and the number of *Item*s is fixed as *M*. Therefore, if the number of requested *Item*s is larger than *M*, some addresses cannot be registered. In order to solve this problem, Pool is designed to send the Item_ovf signal to the arbiter if there is no empty space for the *Item*. Then, the arbiter sends the arb_decision signal back to the core for synchronization. When the core completes its atomic access, it asks Core-Lock Manager to clear the corresponding *Item* entries. If there is any other *Pool* which was blocked by this *Pool*, it would retry its access first by requesting the grant from the arbiter.

5. CONCLUSIONS

C-Lock: Performance-efficient data synchronization method for embedded multi-core systems. In order to minimize the performance loss due to conflict C-Lock checks the true dependencies among the cores. It is done by examining their address range, access type, and so on. C-Lock can save system energy by gating clocks of some cores which request shared data but are blocked since the data are being occupied by another core. These properties of C-Lock combine the advantages of locks and TM and offer the most efficiency. Finite State Machine associated with each core can enhance the clock performance. This will control the current and next states of cores to enable write / read transaction over shared memory.

5 REFERENCES

[1]. Seung Hun Kim, Sang Hyong Lee, Minje Jun, Byunghoon Lee, Won Woo Ro, Eui-Young Chung, Jean-Luc Gaudiot," *C-Lock : Energy Efficient Synchronization for* Embedded Multi-core Systems" IEEE Transactions on computers, VOL. X, NO. X, XXXX XXXX

[2]. Christian Stoif, Martin Schoeberl, Benito Liccardi, Jan Haase ," *Hardware Synchronization for Embedded Multi-Core Processors"* IEEE Transactions,2011

[3]. M. Monchiero, G. Palermo, C. Silvano, and O. Villa, "Efficient Synchronization for Embedded On-Chip Multiprocessors,IEEE Transactions,2010

[4]. David Lee,Mihalis Yannakakis ,AT&T Bell Laboratories Murray Hill, New Jersey,"*Principles And Methods Of Testing Finite State Machines- A Survey*"

[5]. Clifford E. Cummings, Sunburst Design, Inc."*The Fundamentals of Efficient Synthesizable Finite State Machine Design using NC-Verilog and Build Gates*" 2002

[6]. 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.

[7]. 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.

[8]. 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.

[9]. 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.