A Novel Hardware Synchronization for Multicore Embedded Systems

Merin Y.
PG student,VLSI & Embedded Systems, ECE Department
TKM Institute of Technology
Karuvelil P.O, Kollam, Kerala-691505, India
merin.ytm@gmail.com

Abstract— **All multicore technologies aim to exploit parallelism to increase performance. Data synchronization among multiple cores has been one of the critical issues which must be resolved in order to optimize parallelism in these multicore systems. Efficient improvements in synchronization overheads in terms of latency, memory bandwidth, delay and scalability of the system involve a solution in hardware rather than in software. This paper investigates optimized synchronization techniques for shared memory on-chip multicore processors targeted at embedded systems. A hardware synchronization module, Sync-Lock is used here which exploits the unique nature of embedded systems which have demanding requirements for high performance and low power consumption. Balanced energy and performance efficiency of this approach is achieved by combining the advantages of true conflict detection and clock gating. Synchronization hardware is described by Verilog HDL and simulated using ModelSim SE 6.2c.**

Keywords— *data synchronization, multicore, embedded system, clock gating.*

I. Introduction

Multicore has been around for many years in the desktop and supercomputing arenas. But it lagged in the mainstream embedded world; it is now here for embedded as well. Increasing demands for low power, performance/throughput and memory bandwidth are the driving forces to multicore architectures. Concurrency, more than one thing happening at a time is the main feature of multicore. Multicore platform will lead to bigger and faster parallel simulation that takes advantage of the larger number of inexpensive processing units. But it also opens up a number of significant challenges that haven't deal with before. The processor synchronization is a performance bottleneck. When more than one processor attempt to access any shared data simultaneously data synchronization issue arises. The success of various pieces of a program running in parallel to yield a correct result depends strongly on good synchronization between these programs.

Data synchronization prevents data from being invalidated by parallel access. Existing data synchronization methods are either lock-based or lock-free. The former includes locks, semaphores and barriers; these methods blocks access to shared data from processors which fail to acquire permission. But lock-free allow all processors to access the shared data in an optimistic manner and then perform roll back and/or re-execution when conflict occurs. Most popular lock-free approach is transactional memory(TM) [13].

The data synchronization method which was originally developed for general purpose cannot be transferred directly to embedded systems. Embedded system include stringent requirement for low energy consumption as well as high performance. Lock-based methods are widely used in embedded application because of their simple control mechanism. But they sacrifice much parallelism resulting in poor performance. In Transactional memory speculative execution turns out to be wasteful when rollback occurs.

In general, as multicore systems continue to provide increasingly more cores on a single die, hardware support for synchronization will take various steps to mitigate both complexity and performance issue. This paper investigates optimized synchronization techniques for shared memory on-chip multicore processors targeted at embedded systems. It behaves like a lock scheme for energy efficiency, but it shows transactional behavior for checking data conflicts. It delivers Transactional Memory like parallelism in race condition by detecting true conflicts. The detection is done by considering address range, type and dependency of simultaneous accesses. When true data conflict is detected only one of the cores gets access and others are clock-gated to minimize dynamic power consumption. There is no speculative execution and rollback as in Transactional Memory approach. So its energy efficient compared to TM-Approach. It utilizes the available parallelism much better than lock based approach due to true conflict detection. All these advantages are achieved by a simple hardware support, Sync-Lock.

II. Related Work

Christian Stoif in the paper "Hardware Synchronization for Embedded Multi-Core Processors," proposed Multi-Access Controller (MACtrl) consists of core-side and inter-core logic, establishing coherence and consistency for different types of shared memory by hardware means. It also support for point-to-point synchronization between the processor cores is realized implementing different hardware barriers. Multiple cores use inherent parallelism by locking shared memory more intelligently using an address-sensitive method.

Speculative Lock Elision (SLE) [11], a hardware-based approach which elides the unnecessary lock-induced serialization from dynamic execution stream and enable highly concurrent multithreaded execution. It allows non-conflicting critical sections to be executed ad committed concurrently. Misspeculation due to inter-thread data conflict is detected using existing cache mechanism and roll back is used for recovery. SLE guarantees correct execution even in the absence of precise information from software and independent of nesting levels and memory ordering. Successful speculation elision is validated and committed without acquiring lock. In the paper "Transactional Lock-Free Execution of Lock-Based Programs," he proposed Transactional Lock Removal. When data conflict occurs corresponding threads are restarted to acquire the lock in a serialized manner. Transactional Lock Removal (TLR)[9] also uses hardware to convert lock-based critical sections transparently and dynamically into lock-free optimistic transaction. It resolves data conflict based on time stamp in order to provide transactional semantics and freedom from starvation.

Monchiero proposed a hardware lock that optimizes power and performance by replacing the processors polling with hardware notification in the paper "Power/performance hardware optimization for synchronization intensive applications in mpsocs". The idea is to locally perform the synchronization operations which require the continuous polling of a shared variable, thus featuring large contention. They used a hardware block called Synchronization – operation Buffer (SB) which monitors the processors energy and bandwidth consuming polling operation can be avoided. SB is portioned into two separate components. One is devoted to manage events, named Event Buffer (EB), the other one is dedicated to spinlocks, named Lock Buffer (LB). SB queues and manages the requests issued by the processors. This improves performance and energy efficiency of data synchronization by reducing memory overhead [5].

Shavit proposed a software method for supporting flexible transactional programming of synchronization operations Software Transactional Memory (STM) in the paper "Software Transactional Memory". STM is non-blocking and can be implemented on existing methods using only a Load_Linked/Store_Conditional operation. TM provides sufficient programmability to the programmers by abstracting the details of synchronization. Programmers rather focus on functionality. Even though TM simplifies the programming model and maximizes the concurrency, transaction may suffer from interference which causes them to abort and from heavy overheads for memory accesses [12].

Takayuki in the paper "Adaptive Locks: Combining Transactions and Locks for Efficient Concurrency," proposed an adaptive locking technique that dynamically observes whether a critical section would be best executed transactionally or while holding a mutex lock. The critical new elements of our approach include the adaptivity logic and cost-benefit analysis, a low overhead implementation of statistics collection and adaptive locking in a full C compiler, and an exposition of the effects on the programming model. Adaptive locks simplify the programming model by reducing the need for fine-grained locking: with adaptive locks, the programmer can specify coarse-grained locking annotations and often achieve fine-grained locking performance due to the transactional memory mechanisms. Some other works proposed energy-aware lock methods [3].

III. SYNC-LOCK

Ideal synchronization is the elimination of speculative execution while exploiting parallelism as much as possible. Sync-Lock is such a synchronization scheme. It uses the address range for detecting true dependencies which decide whether to execute or hold the operation. If the cores are accessing different variables, Sync-lock detects no conflict. Then operation can be performed simultaneously, achieving Transactional Memory (TM) [13] like parallelism. The system will permit only one access at a time if there is a true conflict among the cores. Moreover, the cores without access permission move into clock-gated state to reduce dynamic power consumption. Consequently Sync-Lock yields higher energy efficiency than TM and provides higher performance than Lock.

A. Concept

The main idea of Sync-Lock system is to exploit available parallelism with true conflict detection and to minimize dynamic power consumption with clock-gating idle cores. Fig.1 shows the concept of locking scheme. For explaining the concept a dual core system is considered and the synchronization among the cores is handled by Sync-Lock.

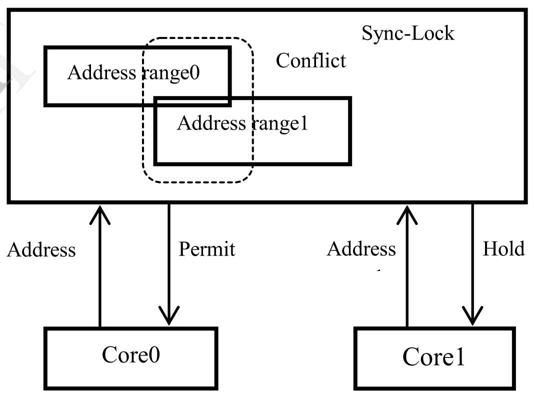

Fig. 1. Concept of Sync-Lock mechanism

Before executing the critical section, every core sends the address range to be accessed i.e., address range0 (core0) and address range1 (core1). After that, the centralized peripheral Sync-Lock decides whether address range overlaps 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. The clock gating reduces dynamic power dissipation. If there is no true dependencies among the cores simultaneous execution are possible. This approach considers low power consumption requirement of embedded system and also provide more parallelism than lock-based method. This approach is energy and performance efficient.

B. Sync-Lock Architecture

The top-level architecture of a shared memory multicore system is shown in Fig. 2. There are N number of cores connected to a shared memory. The synchronization is handled by the hardware module Sync-Lock. Sync-Lock; the hardware module is an additional peripheral and the key component of synchronization mechanism which is in charge of detecting true conflict among the accesses to shared data and controlling clock-gating of cores. Each core is in charge of setting the necessary information to Sync-Lock, which includes base address, size, and type of the data it intends to access. When the information is set, the core is allowed to attempt its atomic operation by notifying the Sync-Lock.

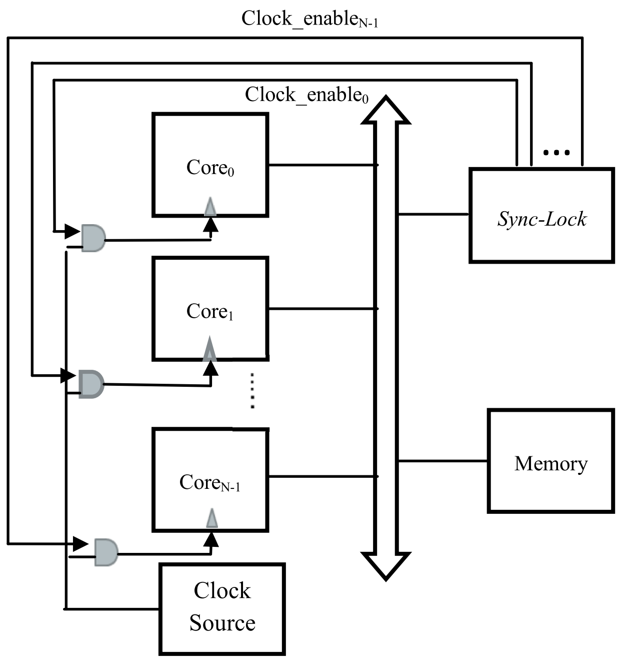

Fig. 2. Top-level architecture

Sync-Lock do conflict checking and in case of conflict, grants permission to only one of the cores while gating the other cores which tend to access the data. 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 the Sync-Lock that its atomic access is finished. So the conflict checking routine in Sync-Lock is again triggered. After conflict checking Sync-Lock gives permission to another core by de-asserting the corresponding clock gating signal. Each core in the processor can record access information with Sync-Lock. It refers to a storage that contains information for checking true conflict with the accesses of other cores.

Access information consists of the following fields:

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

gIdx is used as a time stamp. Each core can register at most M such access records. Sync-Lock manages the status of the entries by checking the valid fields thus put the incoming record to an empty entry. Sync-Lock uses a dynamic priority scheme to determine which core should get the grant to store their access records. If the core gets grant it sets gIdx of the newly registered record with a proper time stamp. After that, the Sync-Lock check for conflicts by comparing the access records of requested core and other cores records. The major part of this process is done by the conflict checking logic. The conflict checking logic checks for true conflicts among the cores' memory access. As an illustration, true conflict occurs when the following conditions are simultaneously present:

- Both access records are valid
- Their address range overlaps
- At least one of them is a write operation
- gIdx of requesting core is greater

The first two conditions are obvious, while third one filters out the false dependencies. The fourth condition detects the possible data hazard. .

Sync-Lock look for the true conflict among requested cores record and the other core records for producing clock enable signal for the requesting core. If any conflict is reported, it means requested atomic access cannot be executed at this time and therefore Sync-Lock disables the clock of the corresponding core by de-asserting clock enable signal. Also conflict information are stored in a register so that the cores can watch the events of the blocking core being cleared and reattempt its access. When no conflicts are reported from the other core, the core keeps running and executes the atomic access for the corresponding access records.

Each core has its own fixed number of access records that can be stored as M in the scheme. Therefore, if the number of records to be registered is larger than M, some addresses cannot be registered. Sync-Lock is designed to handle this problem as well. Core's operation is stalled if other cores are executing critical section.

IV. Results And Discussion

The design entry is modelled using Verilog HDL in Xilinx ISE Design Suite 13.2 and the simulation of the design is performed using Modelsim6.2c to validate the functionality of the design.

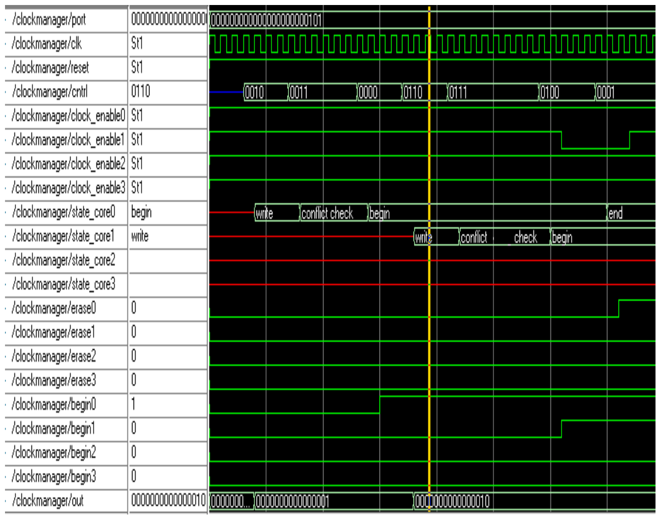

Fig. 3. Simulation result of Sync-Lock

A fully generic design of the Sync-Lock has been developed in the hardware description language Verilog HDL in order to allow easy scaling in terms of the processor cores. The goal to keep the design as compact as possible is achieved by a code optimized algorithm that allows the core to access the shared memory in case no true conflict. For this purpose access information are first recorded before accessing the critical section. This access records are checked for conflict with cores which are accessing the critical section to produce clock enable signal. Once the atomic is finished they are notified to the Sync-Lock manager so that cores that are clock gated can again check for conflict.

V. CONCLUSION

This is an energy and performance efficient data synchronization for multicore embedded systems. It can save more energy by gating the clocks of some cores which request shared data but are blocked since data are being occupied by other cores. In order to minimize the performance loss due to conflict, Sync-Lock checks the true dependencies among the cores by examining their address range, access type and so on, unlike traditional lock. These properties of this synchronization scheme combine the advantages of locks and TM .It consists of special hardware Sync-Lock. The modules of this synchronization hardware are described by Verilog HDL to allow easy scaling and they are simulated for the functionality using ModelSim 6.2c.

REFERENCES

[1] Christian Stoif, Martin Schoeberl, Benito Liccardi, Jan Haase, "Hardware Synchronization for Embedded Multi-Core Processors" IEEE International Symposium on Circuits and Systems (ISCAS),

[2] Bryon Moyer, *Real World Multicore Embedded Systems*, 1st Edition Elsevier/Newnes, 2013

[3] 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.

[4] A. Tumeo et al., "HW/SW methodologies for synchronization in FPGA multiprocessors," *in FPGA'09*, Monterey, California, USA. *IEEE* Press, 2009, pp. 265–268.

[5] 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

[6] 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.

[7] P. G. Paulin, C. Pilkington, M. Langevin, E. Bensoudane, and G. Nicolescu, "Parallel Programming Models for a Multi-Processor SoC Platform Applied to High-speed Traffic Management," *in Proc. 2nd IEEE/ACM/IFIP Int'l. Conf. on Hardware/Software Codesign and Systems Synthesis*, CODES+ISSS 2004, 2004, pp. 48–53

[8] M. Herlihy, V. Luchangco, M. Moir, and W. Scherer III, "Software Transactional Memory for Dynamic-Sized Data Structures," *in Proc. of the 22nd Symp. on Principles of Distributed Computing.*ACM, 2003, pp. 92–101.

[9] 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.

[10] Eung S. Shin, Vincent J. Mooney III and George F. Riley, " Round-robin Arbiter Design and Generation, " ISSS'02

[11] 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

[12] N. Shavit and D. Touitou, "Software Transactional Memory," *in Proc. 14th Symp. on Principles of Distributed Computing*. ACM, 1995, pp. 204–213

[13] 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.