Synchronizing Code Execution on Ultra-Low-Power Embedded Multi-Channel Signal Analysis Platforms

Ahmed Yasir Dogan*, Rubén Braojos*, Jeremy Constantin†,
Giovanni Ansaloni*, Andreas Burg† and David Atienza*
*Embedded Systems Lab. (ESL) - EPFL, Lausanne, Switzerland.
Email: {ahmed.dogan,ruben.braojoslopez,giovanni.ansaloni,david.atienza}@epfl.ch
†Telecommunications Circuits Lab. (TCL) - EPFL, Lausanne, Switzerland. Email: {jeremy.constantin,andreas.burg}@epfl.ch

***Abstract*—Embedded biosignal analysis involves a considerable amount of parallel computations, which can be exploited by employing low-voltage and ultra-low-power (ULP) parallel computing architectures. By allowing data and instruction broadcasting, single instruction multiple data (SIMD) processing paradigm enables considerable power savings and application speedup, in turn allowing for a lower voltage supply for a given workload. The state-of-the-art multi-core architectures for biosignal analysis however lack a bare, yet smart, synchronization technique among the cores, allowing lockstep execution of algorithm parts that can be performed using the SIMD, even in the presence of data-dependent execution flows. In this paper, we propose a lightweight synchronization technique to enhance an ULP multi-core processor, resulting in improved energy efficiency through lockstep SIMD execution. Our results show that the proposed improvements accomplish tangible power savings, up to 64% for an 8-core system operating at a workload of 89 MOps/s while exploiting voltage scaling.**

I. Introduction and Related Work

Energy efficiency is a fundamental aspect of portable autonomous systems for biosignal analysis, where a considerable amount of processing is performed with limited energy supplies. An effective technique to achieve computational power savings is supply voltage scaling, all the way to the sub-threshold region. In the literature, voltage scaling has been extensively analyzed, including its limitations and disadvantages [1], [2].

One of the main issues with low-voltage operation is performance degradation, which can limit the degree of achievable voltage scaling for a given processing requirement. Parallel computing using multiple cores can alleviate this issue, provided that applications can be parallelized. To this end, in [3] near threshold ultra-low-power (ULP) multi- and single-core architectures are compared in terms of power and performance ability for several multi-channel biosignal processing applications. It has been shown that the multi-core approach achieves better energy efficiency compared to the single-core approach for medium and high workloads [3].

This work was partially supported by the the Swiss Confederation under the Nano-Tera.ch NTF Project BioCS-Node. We also acknowledge the support of the Swiss NSF under the project number PP002-119052.
978-3-9815370-0-0/DATE13/©2013 EDAA

One of the key contributors to the overall power consumption is the active power dissipated by instruction memory accesses. To alleviate this issue, in [4] coordinated accesses to instruction and data memories, enabled by a smart crossbar interconnect that supports broadcasting, is proposed. However, the benefit of broadcasting (up to 40.6% active power savings [4]) relies on lockstep execution of algorithm parts that can be carried out using the single instruction multiple data (SIMD) processing paradigm. Therefore, substantial power savings can only be achieved by synchronous instruction execution, which even for many embarrassingly parallel applications is not guaranteed, due to data dependent program flow as well as data memory access conflicts, which bring the processing cores out of lockstep.

Barrier insertion techniques are widely used in parallel computing architectures to achieve synchronization [5]. Cores are synchronized at certain barrier points of execution, i.e., a core is stalled until all other cores reach the same point. While multi-core synchronization has been mostly proposed for high-performance computing [6], we propose to apply this technique to achieve reduced power consumption on ULP multi-core architectures.

In the literature, many software-only [7] and software-hardware hybrid implementations of barriers are proposed [8]. However, these techniques are rather complicated for an embedded ULP platform, where both energy efficiency and low complexity due to real-time applications, are critical. Moreover, mainstream SIMD architectures (e.g., GPUs) lack the flexibility needed for dynamically managing lockstep execution of cores during data-dependent program flows.

The main contributions of this paper are the following:

1) We introduce the usage of barrier synchronization for ULP computing architectures. We show that lockstep SIMD code execution reduces power consumption on multi-channel data processing platforms, maximizing the positive impact of instruction and data broadcasting.
2) We describe a hybrid lightweight hardware-software barrier technique to synchronize code execution among cores. The proposed solution involves a hardware synchronizer in conjunction with an instruction set extension (ISE) for the processing cores.

Our results show that by applying the proposed enhancements to a state-of-the-art processing platform with 8-core, up to 64% power savings is achieved for a biosignal analysis workload of 89 MOps/s while exploiting voltage scaling.

II. Biosignals Monitoring Applications

Biosignal analysis applications process acquisitions of single- or multiple-input biological signals, either to de-noise them or to extract their characteristic features. As it has been recently shown in [9], this signal processing can be optimized to run in real-time on typical embedded low-power microcontrollers. Biosignal applications present considerable parallelism, which can be exploited on multi-core processing platforms in conjunction with low voltage operation to achieve energy savings.

In this work we consider three reference benchmarks, widely used in Electrocardiogram (ECG) processing: the first one (MRPFLTR) performs baseline wander correction and noise suppression based on the morphological filtering [10]. The second benchmark (MRPDLN) delineates ECG signals based on multi-scale morphological derivatives [11]. The last benchmark (SQRT32) consists of a square root kernel, mostly used for multi-lead ECG combination, based on [12].

III. Target Multi-Core Processing Architecture

The target multi-core architecture (c.f. Fig. 1) used in this study supports SIMD operations to exploit data level parallelism. It consists of 8 processing cores, a shared data memory (DM, 64 kByte in total, divided into 16 banks), a shared instruction memory (IM, 96 kByte in total, divided into 8 banks) and a hardware synchronizer for enhanced lockstep execution of applications. Central data and instruction crossbars (hereafter *D-Xbar* and *I-Xbar*, respectively) interconnect the shared memories and the processing cores. In case of multiple conflicting memory access requests (occurring when a memory bank is accessed by more than one core at different memory locations), the cores are served in sequence and the waiting cores are clock gated.

Each core is a custom 16-bit reduced instruction set computing (RISC) architecture [3], providing a complete RISC instruction set including instructions for interrupt and sleep mode support. The sleep mode allows external clock gating of the entire core, until a wakeup event occurs. This new multi-core architecture improves the one in [4] (c.f. *ulpmc-bank*) with a light-weight hardware synchronizer that cooperates with the cores to coordinate IM accesses.

IV. Proposed Synchronization Technique

The substantial power savings achieved through SIMD operations highly rely on synchronous code execution among the cores. A loss of synchronization between the cores can occur mainly due to two reasons: data access conflicts and data-dependent program flow.

A data access conflict occurs when a DM bank is accessed by more than one core at different memory locations (i.e., a shared data access). In this case, the cores that have been

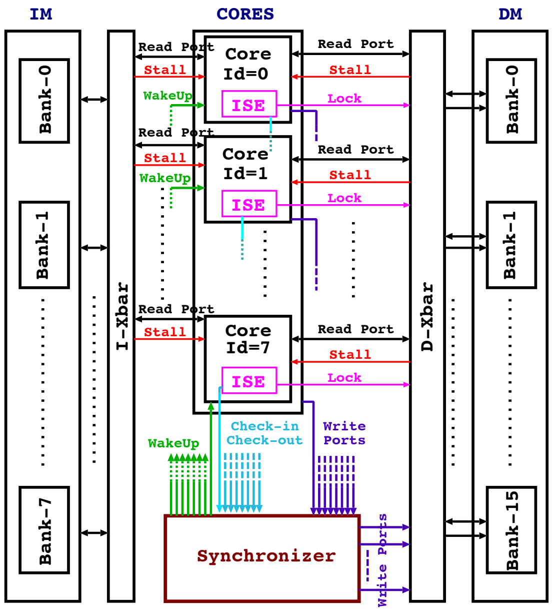


Fig. 1. Improved Multi-Core Architecture with Hardware Synchronizer

served continue code execution while the rest of the cores wait for data to be served. We propose to address this issue by enhancing the data serving policy in the *D-Xbar*. This enhancement stalls synchronous cores until all of them have been served successfully. When a data access conflict occurs, the program counter of the cores are compared to detect if the cores are synchronous.

On the other hand, many applications involve data-dependent code sections, which lead to conditional execution of different parts of the code and, consequently, synchronization is lost. To address this case, we propose to resynchronize the cores at the end of data-dependent code sections with a light-weight synchronization strategy:

1) First, the data-dependent code sections in applications are determined (c.f. Section IV-C). For instance, for a given application with a program flow as depicted in Fig. 2, A, B and C (check-in points) are the beginning of the data-dependent code sections, whereas A', B' and C' (check-out points) are the points where the corresponding data-dependent code sections end.
2) Second, for each data-dependent code section a DM position is assigned to trace the synchronization process. In these memory positions, both the 1-bit core identity flags and total number of cores (the core counter), currently running the corresponding data-dependent code section are stored. Once a core arrives to a check-in point, the corresponding DM position is modified by setting the identity flag and incrementing the core counter.
3) Third, when a core arrives to a check-out point, the core counter is decremented. The arriving core goes into sleep mode and waits for the other cores, expected to arrive at the same check-out point, to resynchronize. Once all the cores, expected to arrive at the same check-out point, reach to the check-out point, then the cores continue execution of the code in lockstep.

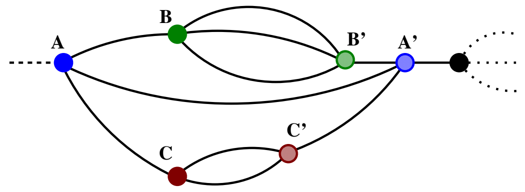


Fig. 2. An Example of Data-Dependent Code Section

The following sections detail the hardware and software support that implements this execution paradigm.

A. Hardware Synchronizer

Typically, several cores reach a check-in point together and then the cores may branch to different conditional code path. Depending on the taken conditional code sections, the cores can reach the corresponding check-out point together, separately, or some of them together while the others separately.

To check-in/check-out a core needs two clock cycles, since a memory read and then a memory write are needed. The synchronizer merges multiple check-in/check-out requests for a synchronization point and modifies the assigned memory position, accordingly. The checkpoint status is updated with the new identity flags and the core counter (incremented for check-in and decremented for check-out). Merged check-in/check-out requests are also executed in two clock cycles.

When all the expected cores reach to a check-out point, the core counter becomes zero. In this case, the synchronizer wakes up all the cores, waiting to be resynchronized (indicated by the core identity flags), and the corresponding memory word is initialized to zero.

B. ISE in the Processing Core

To support the above-described strategy, we have customized the instruction set architecture of the core by adding two new instructions (*SINC* and *SDEC*) and a core output (lock signal) to support the check-in and check-out processes. More specifically, *SINC* and *SDEC* are dedicated to the check-in and check-out processes, respectively, whereas the lock output signal is used to guarantee the atomicity of the read-modify-write operations performed during check-ins and check-outs. In addition, a specific core register (R_{sync}) is used to store the base address of the assigned DM array, holding the identity flags and core counter for each synchronization point. The details of the extensions are as follows:

a) SINC: The assembler semantic is: *SINC #literal*. The literal stands for the synchronization point index, which addresses the position of the synchronization point in the assigned DM array. The instruction reads data from the memory address, calculated by adding the literal value to the R_{sync} base address register. This read data is forwarded to the write port (without any manipulation since the corresponding modifications are performed in the synchronizer), and the core output indicating a check-in request to the synchronizer is activated (c.f. Fig 1).

b) SDEC: The assembler semantic is: *SDEC #literal*. It is similar to SINC, but the output indicates a check-out request. In addition, after requesting the check-out the core goes into sleep mode until a wake up occurs (i.e., when all expected cores reach to the check-out point).

c) Lock Output Signal: This output is activated when *SINC* and *SDEC* instructions are executed. This signal locks the memory position, accessed via the instructions, until it has been modified with the new value for serializing not synchronous memory accesses among the cores in sequential check-in/check-out processes.

C. Synchronization Points Insertion

The proposed synchronization framework only requires the resynchronization points to be defined by inserting pragma in the code. While manually implemented in this study, this instrumentation can in principle be automated during the compilation process. For a given code the check-in and check-out instructions are inserted as shown below.

```
 ....
SINC #(Synchronization Array Index X)
if (CONDITION)
 {//if operations... }
else
 {//else operations... }
SDEC #(Synchronization Array Index X)
 ....
```

Listing 1. Synchronization Points Insertion

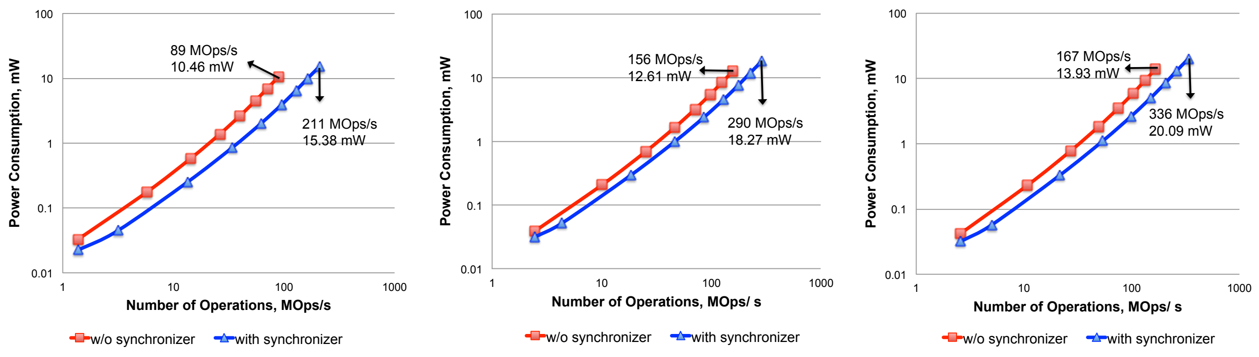
These check-in and check-out instructions are inserted on each data-dependent conditional statement (while/for loops, case/if-else statements, etc.).

V. Power and Performance Evaluation

A. Experimental Setup

To assess the proposed synchronization paradigm in terms of power and performance, we have implemented two multi-core designs, with and without the synchronization feature. Both designs are synthesized in a 90 nm low-leakage process technology. For an accurate power analysis of the designs, toggling information while running the reference benchmarks is obtained by simulating a fully routed design (including the clock tree) with back-annotated timing information. The power values at scaled voltages are calculated considering that the power decreases with the square of the supply voltage. The scaling of the operating voltages is limited to the transistor threshold voltage level, to avoid performance variability and functional failures occurring mainly at sub-threshold voltages.

The achieved minimum critical path delay at the nominal voltage (1.2 V) is 8.9 ns and 9.6 ns for the architectures without and with the synchronization feature, respectively. The targeted applications do not require such high clock frequencies, thus no vital timing issue is present. A relaxed constraint of 12 ns gives good power results for both designs with and without the synchronization feature [4], while still allowing for considerable voltage scaling.



(a) MRPFLTR benchmark (b) SQRT32 benchmark (c) MRPDLN benchmark

Fig. 3. Total Power Consumption of the ULP architectures while running different benchmarks

B. Power and Performance Results

Thanks to the resynchronization process, the stalls due to IM access conflicts are minimized, achieving a speed-up of up to 2.4x on the benchmarks. The architecture with the synchronization feature achieves between 2.5 Ops and 4.0 Ops per clock cycle, whereas the architecture without it can execute between 1.1 Ops and 2.0 Ops per clock cycle, due to the overhead incurred by stalls.

Table I reports the distribution of the average power consumptions between the different components for the multi-core designs with and without the synchronization feature. The improved architecture achieves up to 60% savings for the total number of IM bank accesses when running the ECG benchmarks. This access rate reduction leads to a significant reduction of the power consumed due to IM accesses, as seen in Table I. On the other hand, due to the synchronization overhead, the total number of DM accesses is increased by less than 10%, which leads to almost no increase in the power consumed by the DM. Furthermore, the synchronizer consumes less than 2% of the total power.

The cores in the improved architecture consume slightly more power than those of the architecture without the synchronization feature due to the ISE. This effect is balanced by savings in the interconnects, because of the reduced signal activity due to increased SIMD operations. The improved architecture achieves 2x power savings in the clock tree, since it requires lower clock frequency for a given workload. Overall, without exploiting voltage scaling, synchronization provides up to 38% dynamic power savings.

The synchronization feature is even more beneficial when voltage scaling is considered (c.f. Fig 3). Thanks to higher Ops/cycle, in fact power savings up to 64% are achieved while running the MRPFLTR benchmark at a workload of 89 MOps/s. Similar results have been observed for the other benchmarks: 56% power reduction for SQRT32 at 156 MOps/s and 55% power reduction for MRPDLN at 167 MOps/s.

TABLE I
DYNAMIC POWER DISTRIBUTION OF THE DESIGNS WHILE RUNNING REFERENCE BENCHMARKS AT 8 MOPS/S AND 1.2 V

	w/o synchronizer	with synchronizer
Total	$0.64\,\text{mW} < P < 0.94\ \text{mW}$	$0.47\,\text{mW} < P < 0.58\ \text{mW}$
Cores	0.14 mW	0.16 mW
IM	$0.20\,\text{mW} < P < 0.36\ \text{mW}$	$0.09\,\text{mW} < P < 0.15\ \text{mW}$
DM	$0.05\,\text{mW} < P < 0.08\ \text{mW}$	$0.06\,\text{mW} < P < 0.08\ \text{mW}$
D-Xbar	0.06 mW	0.05 mW
I-Xbar	0.03 mW	0.02 mW
Syncronizer	-	0.01 mW
Clock Tree	$0.09\,\text{mW} < P < 0.16\ \text{mW}$	$0.05\,\text{mW} < P < 0.08\ \text{mW}$

VI. CONCLUSION

Embedded multi-input data processing expose a high degree of parallelism, which can be exploited in multi-core architectures in conjunction with supply voltage scaling to develop an ultra-low-power processor for portable systems. One of the drawbacks in the state-of-the-art architectures is high instruction fetch power consumption, especially for applications causing asynchronous code execution among the cores. To address this issue, we have proposed a technique of code execution synchronization among the cores to maximize the rate of SIMD operations. The proposed enhancements achieve up to 64% power savings when applied to a near-threshold 8-cores platform.

REFERENCES

[1] S. Hanson *et al.*, “Exploring variability and performance in a sub-200-mV processor,” *SSC*, vol. 43, no. 4, pp. 881–891, 2008.

[2] R. Dreslinski *et al.*, “Near-threshold computing: Reclaiming Moore’s law through energy efficient integrated circuits,” *Proc. IEEE*, vol. 98, no. 2, pp. 253–266, 2010.

[3] A. Y. Dogan *et al.*, “Low-power processor architecture exploration for online biomedical signal analysis,” *IET Circuits, Devices Systems*, vol. 6, no. 5, pp. 279 –286, 2012.

[4] ——, “Multi-core architecture design for ultra-low-power wearable health monitoring systems,” in *DATE*, vol. 1, no. 1, 2012, pp. 988–994.

[5] D. E. Culler *et al.*, *Parallel Computer Architecture: A Hardware/Software Approach*. San Francisco: Morgan Kaufmann Publishers, 1999.

[6] A. Gottlieb *et al.*, “The NYU ultracomputer-designing an MIMD shared memory parallel computer,” *IEEE Trans. Comput*, vol. C-32, no. 2, pp. 175 –189, feb. 1983.

[7] C. Ferri *et al.*, “Energy-optimal synchronization primitives for single-chip multi-processors,” in *GLSVLSI*, New York, 2009, pp. 141–144.

[8] C. Stoif *et al.*, “Hardware synchronization for embedded multi-core processors,” in *ISCAS*, may 2011, pp. 2557 –2560.

[9] F. Rincon *et al.*, “Development and evaluation of multilead wavelet-based ECG delineation algorithms for embedded wireless sensor nodes,” *TITB*, vol. 15, no. 6, pp. 854–863, Nov. 2011.

[10] Y. Sun *et al.*, “ECG signal conditioning by morphological filtering,” *Computers in Biology and Medicine*, vol. 32, no. 6, pp. 465–479, 2002.

[11] Y. Sun, K. L. Chan, and S. M. Krishnan, “Characteristic wave detection in ECG signal using morphological transform,” *BMC Cardiovascular Disorders*, vol. 5, no. 1, p. 28, 2005.

[12] T. J. Rolfe, “On a fast integer square root algorithm,” *SIGNUM Newsl.*, vol. 22, no. 4, pp. 6–11, Oct. 1987.