Hardware/Software Approach for Code Synchronization in Low-Power Multi-Core Sensor Nodes

Rubén Braojos, Ahmed Dogan, Ivan Beretta, Giovanni Ansaloni *and* David Atienza
Embedded Systems Laboratory
École Polytechnique Fédérale de Lausanne, Switzerland
Email: {ruben.braojoslopez}, {ahmed.dogan}, {ivan.beretta}, {giovanni.ansaloni}, {david.atienza}@epfl.ch

***Abstract*—Latest embedded bio-signal analysis applications, targeting low-power Wireless Body Sensor Nodes (WBSNs), present conflicting requirements. On one hand, bio-signal analysis applications are continuously increasing their demand for high computing capabilities. On the other hand, long-term signal processing in WBSNs must be provided within their highly constrained energy budget. In this context, parallel processing effectively increases the power efficiency of WBSNs, but only if the execution can be properly synchronized among computing elements. To address this challenge, in this work we propose a hardware/software approach to synchronize the execution of bio-signal processing applications in multi-core WBSNs. This new approach requires little hardware resources and very few adaptations in the source code. Moreover, it provides the necessary flexibility to execute applications with an arbitrarily large degree of complexity and parallelism, enabling considerable reductions in power consumption for all multi-core WBSN execution conditions. Experimental results show that a multi-core WBSN architecture using the illustrated approach can obtain energy savings of up to 40%, with respect to an equivalent single-core architecture, when performing advanced bio-signal analysis.**

***Index Terms*—Embedded Systems, Bio-Medical Signal Processing, Wireless Sensor Nodes, Code Synchronization.**

I. Introduction and Motivation

Recent advances in embedded bio-signal analysis have changed the landscape of health monitoring applications, allowing for continuous digital signal processing (DSP) directly on low-power Wireless Body Sensor Nodes (WBSNs) [1]. In addition to acquisition and wireless transmission of sampled data, state-of-the-art WBSNs embed advanced real-time applications, able to automatically retrieve relevant diagnostic data such as the analysis of respiration or heart rhythm [2] and the detection of epileptic seizures [3].

Energy efficiency is of paramount importance for battery-supplied WBSNs, which must operate autonomously for prolonged periods of time. To minimize energy consumption, processing on these devices requires a carefully tailored computing architecture. An effective method to decrease power consumption is voltage-frequency scaling (VFS), which trades-off the voltage supply (and, consequently, energy consumption) for peak clock frequency [4] [5]. Aggressive VFS [6] can nonetheless degrade run-time performance below the requirements of the target application, so that parallel computations have to be performed to achieve the required throughput [7]. Low-power multi-core WBSNs, thus, appear as promising candidates to improve energy efficiency, exploiting the benefits of single-instructions-multiple-data (SIMD) architectures when executing code synchronously [8]. Devising synchronization techniques for multi-core platforms is often domain-specific, as trade-offs must be considered between flexibility and efficiency of implementations.

Herein, we propose a new lightweight synchronization methodology, enabling parallel execution of embedded bio-signal processing applications on multi-core WBSN platforms. Our solution stems from the observation that applications in this field [2] [9] [10] are divided in several consecutive phases. In the illustrative example of Figure 1, multiple signals are acquired in parallel and independently processed, and outputs are subsequently combined and transformed into a single data stream or set of features that are later analyzed. Similar schemes are found in most WBSNs applications [1].

In this scenario, substantial energy gains can be achieved when multiple cores execute the same phases of an application (e.g., conditioning in Figure 1) on multiple acquired inputs, as demonstrated by the authors of [11] and [8]. However, these synchronization approaches do not provide the necessary flexibility to perform parallel processing of streams

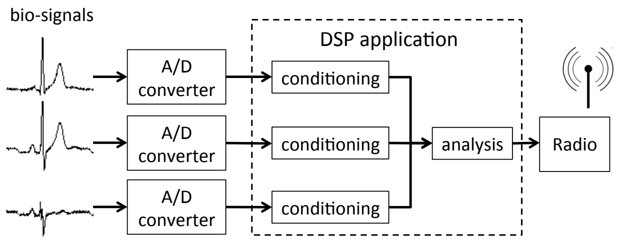

Fig. 1. Block scheme of a smart WBSN platform.

This work has been partially supported by the EC FP7 FET Phidias project (Grant agreement no. 318013.), and the ObeSense (no. 20NA21_143081) and BodyPoweredSenSE (no. 20NA21_143069) RTD projects evaluated by the Swiss NSF and funded by Nano-Tera.ch with Swiss Confederation financing.

on multiple cores, while also supporting producer-consumer relationships among cores. In this work, we generalize the concept of multi-core execution to more complex applications presenting multiple internal phases, with an arbitrarily large degree of parallelism and producer-consumer relationships. To achieve this goal, we propose a novel synchronization mechanism, allowing an efficient mapping of advanced bio-signal processing applications.

The method detailed in the paper is a hybrid hardware/software (HW/SW) solution, employing a dedicated Instruction Set Extension (ISE) and a synchronizer unit that orchestrates the run-time behavior of the system. It requires little manual effort to perform application mapping and results in small run-time overheads.

The contributions of this paper are the following:

- We propose a flexible synchronization technique for bio-signal processing applications presenting different degrees of parallelism and/or producer-consumer schemes among their internal processing phases.
- We detail the synchronization methodology and its hardware and software requirements, and show a case study of its realization on a multi-core WBSN.
- We comparatively evaluate the performance of the proposed system when executing advanced bio-signal DSP benchmarks, obtaining power savings of up to 40% with respect to a similar single-core architecture.

The remaining of this paper proceeds as follows: Section II reviews related works in the field, while Section III details the proposed synchronization method. Section IV describes the experimental set-up and Section V discusses the obtained results. Section VI concludes the paper.

II. Related work

Embedded signal processing on WBSNs reduces the amount of data to be stored or transmitted through the power-hungry radio link by reporting only the analysis results [10]. Moreover, the tight energy budget of these platforms demands careful optimization of both the executed algorithms and the underlying hardware. Indeed, dedicated processing cores have been proposed which, by integrating custom accelerators for common operations (such as FFT and Cordic engines), increase the energy efficiency of bio-signal applications [12] [13]. As opposed to our work, these studies assume an a-priori knowledge of the computationally intensive operations of each target applications.

An orthogonal approach, focus of this paper, is to minimize energy consumption by parallelizing the workload on multiple computing units. Lower clock frequency attainable by a multi-core platform with respect to an equivalent single-core implementation can in fact be exploited to aggressively scale down the supply voltage, resulting in energy gains [7]. The goal of parallelism in this context is therefore not to obtain the highest run-time performance, as is the case for general-purpose multi-core CPUs or GPUs [14], but to achieve the performance required by the target application while lowering the energy consumption.

The authors of [11] and [15] have shown that, even in the field of embedded bio-signal analysis applications, where the computational demands are rather low, parallelism leads to high energy efficiency, especially when multiple cores can execute in *lock-step*, i.e.: they are at the same point of the program flow, so that a single instruction is fetched and delivered in parallel. Indeed, a mechanism to recover lock-step execution across data-dependent branches was proposed in [8]. Nonetheless, the introduced paradigm has the limiting assumption that each core executes the same program flow on a different data stream, which restricts the range of applications that can be targeted. The solution presented herein can instead support either parallel processing on multiple data streams, successive computation stages executed on different cores or a combination of the two paradigms.

Our work is based on barrier insertion, a widely used technique for synchronization of multi-core platforms [16]. A number of implementations of this concept have been proposed, either software-based [17] or requiring dedicated hardware support [18]. Compared to [17] and [18], our method is much less resource-intensive, and therefore better suited for low-power computing architectures.

III. Synchronization methodology

In this section we detail the high-level characteristics of the system in which the proposed technique can be applied, the synchronization mechanism itself and how to application mapping is performed to obtain maximum energy savings.

A. *Hardware/Software elements*

Low-power multi-core WBSNs (e.g. [11] and [15]) typically presents a structure comprising a set of computing units interfaced to instruction and data memories (IM and DM) through interconnection networks, as depicted in Figure 2. Three properties, common in this family of systems, must be satisfied by the platform to apply the proposed synchronization method. First, IM and DM must be divided into several banks so that they can be read/written independently and powered-off if not used in order to save energy. Second, the address space of the data memory needs to be divided into shared and private sections, each core having its dedicated region. Third, to maximize the savings, the interconnection between the memory units and the processors has to allow *broadcasting*,

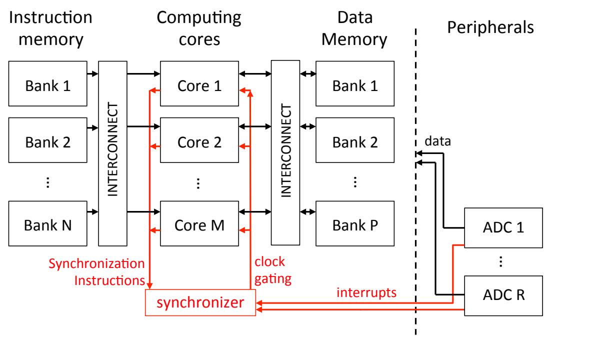

Fig. 2. Hardware architecture of a multi-core WBSN. In red, HW support for the proposed synchronization technique

i.e. multiple read requests from the same location in memory and in the same clock cycle have to be merged into a single memory access.

In this context, our proposed approach consists of a hybrid HW/SW synchronization mechanism. Hardware support for synchronization is provided by a lightweight synchronizer unit that manages the interaction among cores, ensuring lock-step execution when possible, resolving memory access conflicts and keeping track of the execution flow. The unit can clock-gate (pause) cores and resume them, according to the received interrupts and the synchronization instructions issued by the cores. Software support consists of a set of dedicated instructions (*SINC*, *SDEC* and *SNOP*), employed to synchronize code execution using reserved locations (*synchronization points*) in the shared data memory, which store information about the execution flow. Furthermore, a *SLEEP* instruction requests the synchronizer to clock-gate the issuing core until the next synchronization event happens.

B. Synchronization mechanism

Synchronization instructions modify synchronization points, which are divided in two fields: their most significant bits contain 1-bit flags corresponding to the identifiers of each core, while the least significant bits are used as an up/down counter (as illustrated in Figure 3).

The *SNOP(#lit)* instruction appends the identification flag of the issuing core to the *#lit* synchronization point, without modifying the core counter. *SINC(#lit)* also sets the core identification flag, but in addition increases by one the counter. Finally, the *SDEC(#lit)* instruction, without modifying the identification flags, decreases the counter.

These instructions are used both to manage producer-consumer relationships and to enforce lock-step execution after data-dependent branches. In the first case, consumer cores having no data to process execute a *SNOP* instruction, registering themselves in the corresponding synchronization point. The cores then go immediately to the clock-gated mode by issuing a *SLEEP* instruction. Producers, instead, use *SINC*s to register in the synchronization point when starting to compute data for the consumer cores, and *SDEC*s when data is ready (Figure 3-a). When all input data from the producers is available, the value of the counter reaches zero and the synchronizer resumes execution of all the registered cores, as indicated by the identification flags. In the common case when more than one synchronization instruction are issued on the same memory location, the synchronizer merges the requests to perform a single and consistent memory modification.

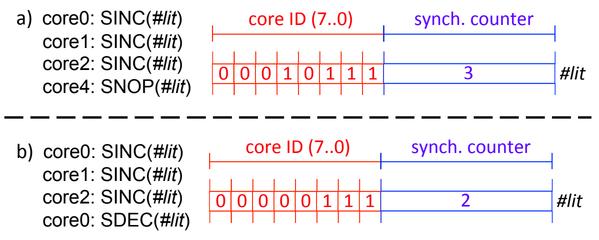

Fig. 3. Examples of synchronization points values. a) cores 0, 1 and 2 should jointly produce data for core 4; data is not yet available. b) cores 0, 1 and 2 have entered a data-dependent branch, core 0 has finished executing it.

The data producer for a core can also be an external peripheral, such as an analog-to-digital converter (ADC) sampling a bio-signal at a constant frequency and providing a data-ready interrupt that will be connected to the synchronizer. When data is not available, cores subscribe to the interrupt line through a memory-mapped register, execute a *SLEEP* instruction and remain clock-gated until the arrival of an interrupt from the registered source forwarded by the synchronizer.

As aforementioned, the synchronization instructions can also enforce lock-step execution among cores across data-dependent branches (Figure 3-b), by employing the methodology described by the authors of [8]. In this case, before entering a branch, cores execute a *SINC* instruction. When they arrive to the end of the segment of code, they issue a *SDEC* and enable clock gating with the *SLEEP* instruction. When all cores that initially entered the branch finish executing it, the core counter becomes zero, and cores are notified by the synchronizer to resume their execution in lock-step.

As a result, starting from the source code of a single-core implementation, three steps have to be performed to parallelize and execute an application using the propose synchronization method, namely:

1) **Partitioning**: applications are divided into different phases, each executing on one core. To exploit lock-step execution, application phases operating in parallel on different data streams should be assigned to different cores. As discussed in Section I, partitioning naturally follows the structure of bio-signal processing algorithms.
2) **Insertion of synchronization instructions**: *SNOP* instructions are added to consumer cores, while *SINC* and *SDEC* to producers. *SINC* and *SDEC* pairs are also inserted before and after data-dependent code segments executing on cores assigned to parallel computation streams.
3) **Mapping**: binary code of the different phases is placed in different IM banks in order to avoid access conflicts and benefit from broadcasting. Moreover, the threshold between shared and private sections in memory and the number of synchronization points must be configured.

Figure 4 graphically shows the result of applying these steps to the application introduced in Figure 1. First, the application is divided into two phases: conditioning and processing. Because conditioning is performed in three different inputs, it is assigned to three different cores that run in parallel, each of them processing one input. The processing phase is assigned to a fourth core that consumes the data produced by the first three. *SNOP* and pairs of *SINC* and *SDEC* are placed properly to manage the producer-consumer relationship and ensuring lock-step execution. In the mapping step, code dedicated for the different phases is placed in different IM banks, with cores executing the same application phase sharing the same bank.

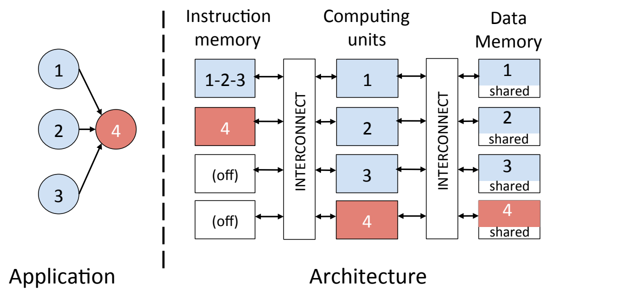

Fig. 4. Mapping the application in Figure 1 on a WBSN embedding 4 computer units, 4 IM banks and 4 DM banks.

IV. Experimental set-up

A. Hardware architecture

We considered a multi-core platform similar to the ones described in [11] and [15] employing parallel computing cores connected to multi-banked IM and DM through crossbars. In addition, the synchronizer unit proposed in Section III-A is integrated in the system and peripherals (such as ADCs) are interfaced using memory-mapped registers.

The crossbars fully connect the cores to the memories and their implementation follows the logarithmic interconnect scheme proposed in [19], which allows combinational (single-cycle) accesses from cores to memories. We modified the crossbars to allow *broadcasting* of data and instructions.

Each computing core consists of a 16-bits RISC architecture featuring a three-stages pipeline with forwarding paths. Their instruction set has been extended to support the proposed synchronization technique. To enforce the division of DM into private and shared sections, each core is equipped with a combinational Address Translation Unit (ATU) consisting of a multiplexor that appends a unique tag per core when an access to the private section is requested. This implementation interleaves the shared section of DM between all the available memory banks.

B. Target multi-core and baseline single-core systems

The target multi-core system employs 8 cores, interfaced with a 96 KByte instruction memory (32 KWords of 24 bits width) divided into 8 banks and a 64 KByte data memory (32 KWords of 16 bits width) divided into 16 banks. Crossbars are sized accordingly and a three-channels ADC unit is interfaced to the system using memory mapped registers located in shared DM and data-ready interrupt lines connected to the synchronizer, which forwards them to cores.

We considered as baseline configuration a single core connected to the same memory hierarchy as in the previous case, so that unused memory banks can be powered-off. To manage the memory interface in this system, simpler decoders can be used instead of crossbars allowing higher clock frequencies at the same voltage level.

C. Simulation framework

The HW/SW co-simulation framework developed to evaluate the proposed method is composed by the programming tool-chain (compiler, builder and linker) and the simulation environment. The former allows for the compilation of code to be loaded and executed on the platform and requires a set of building directives, which guide the automatic linking process. The latter includes a synthesizable RTL description and a System-C architectural simulator of the target platform, which encapsulates the model of the employed RISC cores developed using the LISA tool-suite from Synopsys [20].

We investigated the baseline and target architectures, executing bio-signal processing applications, at two levels of abstraction. At the lower level, post-layout RTL simulations (using a $90nm$ low-leakage process) are employed, measuring the average energy consumption of each architectural element when executing small code sections. Data gathered from RTL simulations is then used to annotate a System-C model of the system, from which application-wide energy consumption figures are extracted in different settings.

Output of the framework is then the average power consumption gathered from an extended period of simulated time (60 seconds for all the experiments in this work), which would not be possible to obtain with time- and resource-consuming RTL simulations. This aspect is of great relevance for WBSN applications, as the input bio-signals have slow dynamics (e.g., in the example of heart monitoring, the normal heart rate ranges from 60 to 100 beats-per-minute), requiring extended simulations to capture the average energy efficiency of different architectural configurations.

D. Benchmarks

We considered three highly optimized applications, from the field of embedded electrocardiogram (ECG) signal processing.

The first benchmark performs three-lead morphological filtering (3L-MF) [21], which removes unwanted components from acquired ECG signals, and operates in parallel on three different input streams (Figure 5-a). When mapped on three cores, the application does not employ producer-consumer relationships, so that synchronization primitives are only used to recover lock-step execution among cores.

Instead, in the second benchmark (Figure 5-b), a three-lead delineation application using multi-scale morphological derivatives (3L-MMD), based on [10], presents both types of synchronizations. In fact, in addition to filtering inputs, it also aggregates them and analyses the resulting combined streams to automatically detect the ECG fiducial points. Consequently, as opposed to 3L-MF, it cannot be mapped using the technique described in [8]. The application is mapped onto five cores, of which three perform filtering in parallel and two are employed to combine the signals and identify the fiducial points, respectively.

The third benchmark (RP-CLASS) uses a heartbeat classifier, operating on a single lead, to discern normal from pathological heartbeats, applying the method proposed by the authors of [22]. When an abnormal situation is detected, a three-lead delineation is activated only for the pathological heartbeat. RP-CLASS is mapped onto six cores (Figure 5-c), and showcases the ability of our proposed synchronization technique to manage both control and data flows among cores.

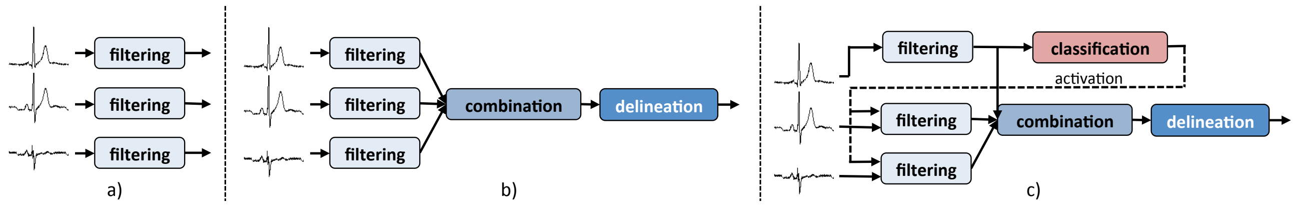

Fig. 5. Block schemes of benchmark applications: a) 3-leads morphological filtering (3L-MF), b) 3-lead filtering + delineation (3L-MMD), c) early classification of pathological beats activating 3-leads delineation (RP-CLASS).

	3L-MF		3L-MMD		RP-CLASS	
	SC	MC	SC	MC	SC	MC
Active Cores	1	3	1	5	1	6
Active IM banks	1	1	3	4	4	6
Active DM banks	3	16	3	16	11	16
IM Broadcast (%)	-	40,36	-	23,44	-	10,30
DM Broadcast (%)	-	3,74	-	2,82	-	1,07
Min. Clock (MHz)	2,3	1,0	3,4	1,0	3,3	1,0
Min. Voltage (V)	0,6	0,5	0,6	0,5	0,6	0,5
Code Overhead (%)	-	2,57	-	0,92	-	0,69
Run-tim Overhead (%)	-	1,65	-	0,96	-	0,60
Avg. Power (μW)	**53,6**	**31,8**	**79,7**	**50,3**	**80,4**	**56,9**
Saving	**40,7 %**		**36,9 %**		**29,2 %**	

TABLE I
DETAILS OF THE EXECUTIONS OF THE DIFFERENT BENCHMARKS ON THE SINGLE-CORE (SC) SYSTEM AND THE MULTI-CORE (MC) ONE EMPLOYING THE PROPOSED APPROACH

It also exemplifies a case where workload is not uniform: as abnormal heartbeats are rare, in fact, the four cores in the delineation chain are seldom activated.

Across experimental tests, we have used standard multi-lead ECG inputs. To evaluate the 3L-MF and 3L-MMD, a multi-lead signal from a healthy subject of the CSE Database [23] has been employed. For the RP-CLASS application, 20% of pathological beats were inserted, representing the average presence of abnormalities in the CSE database.

V. Experimental results

Three aspects are investigated in this section. First, the run-time requirements of the described benchmarks are analyzed while executed in the baseline and target architectures. Then, the power consumption of the building components of both systems is shown and the obtained numbers are discussed. Finally, the most complex of the evaluated benchmarks, RP-CLASS, is further employed to demonstrate the effectiveness of the proposed synchronization technique in reducing the power consumption of the multi-core system even in the case of unbalanced workload and not lock-step code execution.

A. Performance and memory footprint comparison

The evaluated benchmarks were optimized to be executed in both the single- and multi-core architectures, considering in each case the least possible amount of memory and computational requirements while meeting real-time constraints. In particular, the unused memory banks are powered-off and the system clock frequency is reduced to the minimum in order to exploit the benefits of voltage-frequency scaling (VFS). Details of the executed experiments are shown in Table I.

Three main conclusions can be drawn from these numbers. First, all the applications can run in real-time and at a lower clock frequency in the case of the multi-core architecture, which allows to perform aggressive voltage scaling. Indeed, as explained in Section II, higher demands of computing power are solved using a larger number of cores instead of increasing the system clock frequency.

Second, the single-core architecture presents lower memory requirements. In fact, the mapping of code in the IM is less constrained, whereas in the multi-core platform instructions need to be placed in different memory banks to avoid access conflicts. In addition, in order to support the division of the data memory into shared and private sections, all the banks of the multi-core platform need to be active due to the design of the ATU unit explained in Section IV-A.

Third, the introduced overhead due to the proposed methodology is very low. In the worst case (3L-MF), the inserted special instructions add-up less than 3% of the total code while, at run-time, the issued synchronization primitives represent 1,65% of the active time.

B. Single- and- multi-core energy consumption

Power consumption numbers from Table I show a considerable reduction of up to 40% when employing the proposed approach in the multi-core platform. Figure 6 presents a decomposition of the power consumption of the building components of both architectures. Moreover, it shows the power consumption of a multi-core system that does not employ the proposed synchronization approach, performing active waiting for the producer-consumer relationships.

The experiments show that the multi-core system adds a non-negligible overhead (e.g. up to 34% of the total energy in 3L-MF) due to the extra necessary components (crossbars, logic and a more complex clock tree). In addition, when the synchronization technique is not employed, the total power consumption of the multi-core platform can be lower, comparable or higher (e.g. 3L-MF, 3L-MMD and RP-CLASS respectively) than the consumption of the single-core architecture, depending on the workload balance among cores. However, if our proposed approach is used, the energy requirements are drastically reduced in all the cases, achieving important savings thanks to the benefits of VFS.

As Figure 6 shows, one of the advantages of our technique is the reduction of the program memory consumption due to instruction broadcasting. In addition, although the synchronization technique slightly increases the memory usage, the DM consumption is not incremented significantly. In fact, when the application memory footprint is large, like in RP-CLASS, the multi-core DM becomes more energy-efficient,

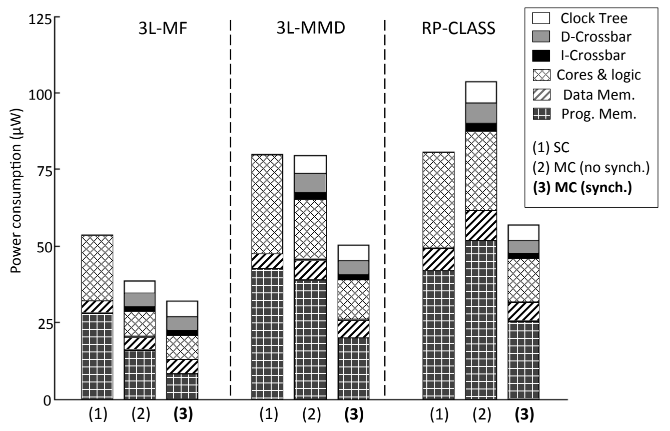

Fig. 6. Power consumption decomposition of the single-core (SC) and multi-core (MC) systems with and without the proposed synchronization approach.

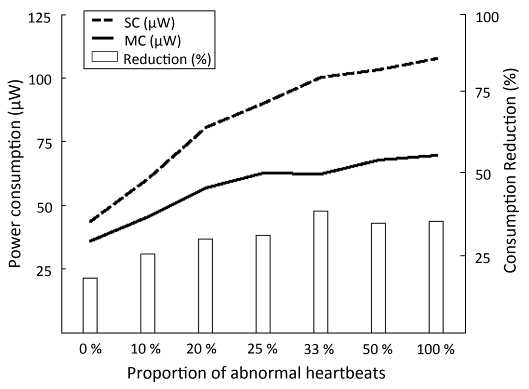

Fig. 7. Power consumption (left axis in μW) of the single-core (SC) and multi-core (MC) systems and the respective reduction (right axis, in percentage) when employing the proposed approach in the multi-core platform

since it operates at a lower voltage level and only few banks can be powered-off in the baseline system.

C. Synergies between VFS and broadcasting

The proposed synchronization methodology allows to efficiently exploit the benefits of voltage-frequency scaling and broadcasting. These two features, on their own, improve the energy efficiency of low-power multi-core systems ([4] [6] [11] [8]), and in combination lead to even greater savings. Figure 7 shows the energy consumption of the baseline and the target architectures and the percentage reduction while executing the RP-CLASS applications with different inputs, varying the amount of pathological heartbeats. For all the tests the abnormal heartbeats have been distributed uniformly.

When there are no pathological heartbeats, the analysis chain (4 cores) is never activated and no parallel computation is carried out. However, energy savings of 17% are still obtained due to voltage-frequency scaling since the workload is divided among cores in the multi-core system. In addition, when abnormalities are present, broadcasting reduces the consumption when the analysis chain is activated due to the lock-step execution of code. In that case, the benefits of both features combined allow for improvements in the energy efficiency of up to 38% in the best case.

VI. Conclusion

In this work we have proposed a HW/SW synchronization methodology to reduce the power consumption of low-power multi-core WBSNs targeting the execution of bio-medical applications. The energy efficiency is improved by exploiting instruction and data broadcasting and voltage-frequency scaling. The described methodology includes minimal hardware support and an instruction set extension while requiring little software adaptations in the source code of the applications.

Three advanced DSP applications including different levels of complexity and parallelism have been used as benchmarks for our study. Experimental results show that, by using the described synchronization technique, a state-of-the-art multi-core system can achieve considerable energy savings of up to 40% with respect to an equivalent single-core platform.

References

[1] Y. Hao *et al.*, “Wireless Body Sensor Networks for Health-Monitoring Applications,” *Physiological Measur.*, vol. 29, no. 11, p. R27, 2008.
[2] T. Berset *et al.*, “Robust Heart Rhythm Calculation and Respiration Rate Estimation in Ambulatory ECG Monitoring,” *BHI*, pp. 400–403, 2012.
[3] F. Massé *et al.*, “Miniaturized Wireless ECG Monitor for Real-Time Detection of Epileptic Seizures,” *ACM TECS*, vol. 12, no. 4, pp. 102:1–102:21, 2013.
[4] S. Hanson *et al.*, “Exploring Variability and Performance in a Sub-200-mV Processor,” *Solid-State Circuits*, vol. 43, no. 4, pp. 881–891, 2008.
[5] R. Dreslinski *et al.*, “Near-Threshold Computing: Reclaiming Moore’s Law Through Energy Efficient Integrated Circuits,” *IEEE*, vol. 98, no. 2, pp. 253–266, 2010.
[6] M. Seok *et al.*, “The Phoenix Processor: A 30pW Platform for Sensor Applications,” *VLSI Circuits*, pp. 188–189, 2008.
[7] Y. He *et al.*, “Xetal-Pro: an Ultra-Low Energy and High Throughput SIMD Processor,” *DAC*, pp. 543–548, 2010.
[8] A. Dogan *et al.*, “Synchronizing Code Execution on Ultra-Low-Power Embedded Multi-Channel Signal Analysis Platforms,” *DATE*, pp. 396–399, 2013.
[9] J. Yoo *et al.*, “An 8-Channel Scalable EEG Acquisition SoC with Fully Integrated Patient-Specific Seizure Classification and Recording Processor,” *ISSCC*, pp. 292–294, 2012.
[10] F. Rincon *et al.*, “Development and Evaluation of Multilead Wavelet-Based ECG Delineation Algorithms for Embedded Wireless Sensor Nodes,” *Info. Tech. in Biomedicine*, vol.15, no.6, pp. 854–863, 2011.
[11] M. Ashouei *et al.*, “A Voltage-Scalable Biomedical Signal Processor Running ECG Using 13pJ/cycle at 1MHz and 0.4V,” *ISSCC*, pp. 332–334, 2011.
[12] J. Kwong *et al.*, “An Energy-Efficient Biomedical Signal Processing Platform,” *Solid-State Circuits*, vol. 46, no. 7, pp. 1742–1753, 2011.
[13] S. Sridhara *et al.*, “Microwatt Embedded Processor Platform for Medical System-on-Chip Applications,” *Solid-State Circuits*, vol. 46, no. 4, pp. 721–730, 2011.
[14] J. Owens *et al.*, “GPU computing,” *IEEE*, vol. 96, no. 5, pp. 879–899, 2008.
[15] A. Dogan, *et al.*, “Multi-Core Architecture Design for Ultra-Low-Power Wearable Health Monitoring Systems,” *DATE*, pp. 988–993, 2012.
[16] D. E. Culler *et al.*, “Parallel Computer Architecture: a Hardware/Software Approach,”, *Gulf Professional Pub.*, 1999.
[17] C. Ferri *et al.*, “Energy-Optimal Synchronization Primitives for Single-Chip Multi-Processors,” *GLSVLSI*, pp. 141–144, 2009.
[18] C. Stoif *et al.*, “Hardware Synchronization for Embedded Multi-Core Processors,” *ISCAS*, pp. 2557–2560, 2011.
[19] M. Kakoee *et al.*, “A Resilient Architecture for Low Latency Communication in Shared-L1 Processor Clusters,” *DATE*, pp. 887–892, 2012.
[20] Synopsys [Online]. Available: www.synopsys.com
[21] Y. Sun *et al.*, “ECG Signal Conditioning by Morphological Filtering,” *Comp. in Biology and Medicine*, vol. 32, no. 6, pp. 465–479, 2002.
[22] R. Braojos *et al.*, “A Methodology for Embedded Classification of Heartbeats Using Random Projections,” *DATE*, pp. 899–904, 2013.
[23] J. Willems *et al.*, “Common Standards for Quantitative Electrocardiography: Goals and Main results. CSE working party.” *Methods Inf. Med.*, vol. 29, no. 4, pp. 263–271, 1990.