Development of Real-Time EEG Application on an Ultra-Low-Power DSP

Suzan Cirit[1], Julien Penders[2], Maryam Ashouei[2], Jos Hulzink[2], Michael de Nil[2], Jos Huisken[2], Ulrich G. Hofmann[3]

[1] Methodpark Software AG, Wetterkreuz 19a, 91058 Erlangen, Germany
[2] IMEC-NL/Holst Centre, High Tech Campus 31, 5656 AE Eindhoven, The Netherlands
[3] Institute for Signal Processing, University of Lübeck, Ratzeburger Allee 160, 23538 Lübeck, Germany

Corresponding author: maryam.ashouei@imec-nl.nl

Abstract— **Autonomous wireless devices for healthcare monitoring become a reality only if such devices are embedded with enough intelligent and processing capabilities to minimize the amount of data being transferred through the wireless network. Also, the embedded processing capability must be made energy efficient so that the device can operate on scavenged energy or very limited battery power. This paper reports the development of a real-time EEG (Electroencephalography) application based on DWT (Discrete Wavelet Transform) and its mapping and optimization on an Application Specific Instruction set Processor (ASIP). It shows the drastic energy reduction that can be achieved by cross-optimization of the algorithm and the ASIP architecture. Our results indicate that such cross-optimization can reduced the dynamic energy by more than 80%.**

Index Terms— **ASIP, DWT, EEG, Real-Time, Ultra-Low Power.**

I. INTRODUCTION

Recent advances in miniaturized wireless autonomous devices are expected to enable ambulatory and non-invasive monitoring of health parameters, thus increasing quality of services in healthcare systems in the coming years. There are still important technology challenges to overcome in order to achieve widespread use of wireless intelligent monitoring devices, as reported previously [1]. Some of the tasks to materialize such an intelligent wireless autonomous system include the development of low-power sensors to collect data from human body, of real-time and efficient algorithms for analysis of biomedical signals, of platforms to process the data with very low energy consumption, and of radios to transmit the analyzed data to a monitoring center. For example, current devices used for monitoring of bio-potential signals, such as Electroencephalography (EEG), Electrocardiography (ECG), and Electromyography (EMG) typically include long wires connecting sensors, attached to the patient body, to a data acquisition box. The implementation of an autonomous wireless intelligent sensor with small size, low power, embedded processing capabilities and low-cost would remove the burdens due to wires and enable monitoring of the patient in normal conditions. A first generation of such a system was previously reported, and shown in an application for sleep monitoring in a home environment [2]-[3].

Today's wireless sensor nodes spend significant of their power budget in wireless communication as shown in [4]. While power-efficient wireless communication is essential in the autonomy of WSN, the problem can be alleviated by local processing and reducing the amount of data needed to be transferred over the wireless link. However, local processing of the data is also expensive if not performed effectively. While today some processing can be performed using commercial micro-controllers, such as TI-MSP430, their computational power is limited and their power efficiency is not acceptable for autonomous WSN.

In [5], it was shown that by careful code and architecture co-optimization, significant power reduction can be achieved when running ECG R-peak detection algorithm. This paper reports the development of a real-time biomedical application analyzing EEG signal based on DWT (Discrete Wavelet Transform) and its mapping on an ASIP under-development in-house for bio-medical application domain. The code and ASIP optimizations were performed to achieve high energy efficiency.

The paper is organized as follows. Next section gives a brief introduction to the DWT-based EEG. Section III presents application mapping and optimizations and the application mapping and architectural optimization. Section IV presents the results. Section V outlines the conclusions and future work.

II. EEG SIGNAL PROCESSING

A. EEG Overview

This section introduces the basic of EEG signals and a widely used technique for processing the signals. EEG is the

Table 1. Characteristics and the origin of brain waves

Type of wave	Frequency range [Hz]	Characteristics	Source of generation
Alpha	8~12	Resting condition, sitting in a relaxed position with eyes closed	Cortical, thalamic nuclei and brain stem
Beta	12~30	Mental thought and activity (eyes open)	Inside the cortex
Gamma	>30	Cognitive functions, sensory stimuli, motoric activity	Local areas of the cortex and brain stem
Delta	2~4	Deep sleep	Sub-cortical areas
Theta	4~8	Asleep	Cortical, hippocampal, brain stem and thalamus

measurement of brain electrical activities using electrodes, which are placed on a patient's scalp following the international 10-20 system as shown in Figure 1[6]. Different frequency components exist in the measured signals. EEG waves are classified into five frequency bands (Table 1). Each frequency band is generated by different regions of the brain and indicates certain features in the patient such as his depth of sleep. The recorded EEG signals are used as input for health care monitoring and diagnosis, such as epileptic seizure detection, emotion monitoring, sleep monitoring, etc. For instance, one of the early signs of epileptic seizure is the presence of characteristic transient waveforms (spikes and sharp waves) in EEG data.

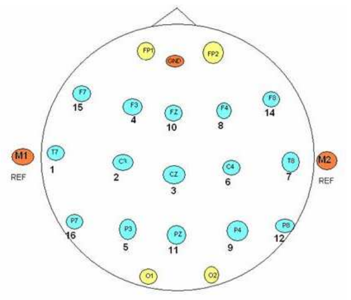

Figure 1. Electrode placement scheme on scalp

Several methods exist to extract the oscillations of a specific frequency from EEG data. Among the most popular are wavelet transform (WT), Fourier transform (FT), autoregressive model and bi-spectral analysis. Since the EEG signals are non-stationary, (discrete) wavelet transform (DWT) is widely used for EEG analysis [8]-[11]. This is because the DWT maintains both time and frequency resolution, which is essential for non-stationary signals. DWT is explained in the next subsection.

A. DWT-based EEG Analysis

This subsection provides an overview of DWT without going into the mathematical details behind it. Interested readers are referred to [8] for further in-depth explanation. DWT provides a time-frequency decomposition of the signal and is usually implemented using two FIR filters, a high-pass and a low-pass filter [15], derived from the mother wavelet. The DWT of a signal is calculated by recursively applying these filters (Figure 2). The filters' outputs are down-sampled by a factor of two. This procedure is repeated until the desired frequency band remains in the signal.

At each iteration of the recursion (i.e. at each step of the decomposition scale), two types of coefficients are left: the details (d), representing high frequencies, and the approximation (a), representing low frequencies of the input signal of the respective decomposition step. During the computation, the approximation coefficients are used as input for the next decomposition level and the details will be kept for the analysis, as illustrated in Figure 2 .

As mentioned earlier, at each decomposition scale, the signals are down sampled. This introduces time-variance and is not suitable for biomedical signal analysis (EEG, ECG). To solve the aliasing issue caused by down sampling, the un-decimated DWT was introduced. It is computed in a similar way, except that the down-sampling step is by-passed. Practically, this can be easy implemented by up-sampling the filter coefficients at each decomposition level, i.e. by inserting zeros between the filter coefficients. This is known as the "trous algorithm" [15], and is used as a starting point for our research.

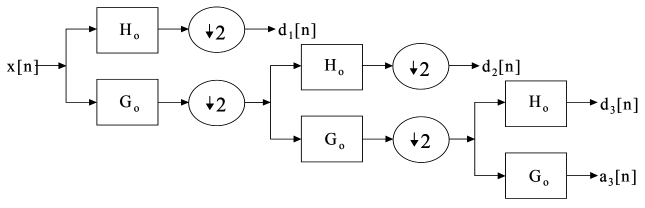

Figure 2. Recursive calculation of DWT

On-the-fly analysis of EEG signals is essential to reduce the output latency. Furthermore, in a wireless sensor node, where the processor has limited resources, it is important to limit the amount of memory space required. Therefore, we implemented an un-decimated DWT algorithm that meets the real-time and limited memory space constraints by optimization of filter coefficient up-sampling.

Table 2 shows the data buffer size, the program memory size, and the latency of the original EEG algorithm and its real-time implementation. The results were obtained on an Intel Core 2 duo 1.8 GHz using Visual Studio 2005 compiler.

	Data Buffer Size	Program Memory Size	Latency
Original Implementation	20 kB	184 kB	1022 samples
Real-Rime Implementation	4 kB	40 kB	256 samples

Table 2: Algorithm Performance

Furthermore, since resource constrained low power architectures usually are fixed-point architectures, we converted the algorithm from floating point to fixed-point format. The acquired signal from the sensor already is in 12-bit format. Therefore a conversion into fixed-point format was applied only to the FIR filter coefficients and computations. A fixed-point implementation has been done in 32-bit representation. The output of the fixed-point representation was compared with the output of the floating-point format. The error distribution of the output is shown in Figure 3. The 32-bit fixed-point implementation introduces an error range of [-5, +4] with a mean error value of 0.4899, which is negligible relative to the signal magnitude.

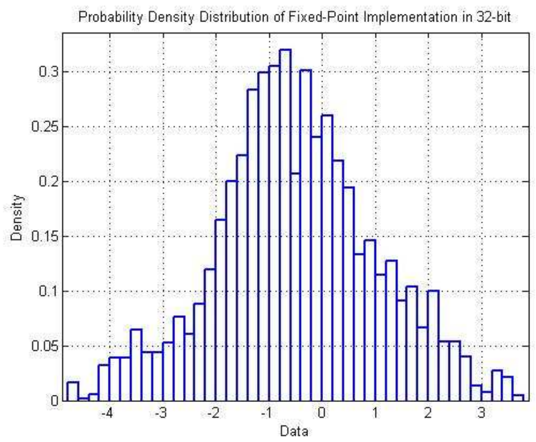

Figure 3. The output error PDF using 32-bit fixed-point implementation

The real-time implementation was verified with a use case scenario “opening and closing eyes”. Open-eye and closed-eye statues correspond to the brain wave beta (12-30 Hz) and the brain wave alpha (8-12 Hz) respectively. For the experiment, EEG was recorded at O1 and O2 (the occipital and visual locations on the scalp) locations.

The upper part of Figure 4 shows the recorded EEG data at O2 location. The opening and closing of eyes happen every 20 seconds. Alpha waves are expected in the sixth scale, whereas beta waves appear in the fifth scale. To validate this assumption a decomposition depth of six was applied to the signal. From the figure, one can see that in closed-eye status, an increase in alpha activity is present (the lower part of Figure 4).

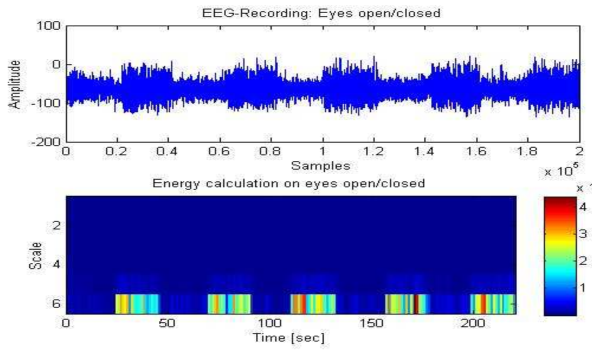

Figure 4. The raw EEG data sampled at 1kHz is shown in the upper part. The lower part shows the energy content of the decomposition at each scale.

In the following section, the code and ASIP (Application Specific Instruction set Processor (ASIP) optimizations are described.

III. Application mapping and optimizations

EEG signal processing can be performed on a range of hardware platforms including general purpose processor, DSP, ASIP, or ASIC. While general purpose processors are the most flexible in terms of programming capability, they consume substantial amount of power, which makes them unsuitable for a WSN. On the other hand, ASIC’s, while power efficient, provides little to no programmability, which makes them cost-inefficient in the presence of changing algorithm. For this reason, the Human++ activity at IMEC is focusing on ASIP designs, which provide some degree of programmability while still power efficient.

Today, there are a few commercially available design tool suits for ASIP design. One of such a tool is Silicon Hive [17], used for architectural exploration in this study.

A. ASIP Design Flow

Silicon Hive processors are described in a high level object oriented language from which a processor and a C compiler for that processor can be generated [17]. The compiled code can be run on the processor using a simulation environment. This way the bottlenecks can be identified and optimizations can be performed.

The high-level processor description allows for quick changes in the architecture and fast architectural exploration. This way, application specific processors can be designed easily.

For the optimized processor, then Silicon Hive can generate the synthesizable RTL code. The commercially available CAD tools for synthesis and place & route are then used to complete the design flow from RTL to GDSII.

This way the acquired power and timing information are more accurate as will be discussed in Section IV.

B. Code and Architectural Optimizations

The reference processor (Figure 5) is a 3 issue slot machine connected to a bus and controlled by a host, which can be a simple state machine. Each issue slot has its own local register file and separate set of functional units. Looking at the interfaces, issue slot 1 has two FIFO connections, issue slot 2 is connected to the local data memory and issue slot 3 has a master interface to the bus, which can be used for accessing external memory. The processor reads its instructions out of a 128-bit wide program memory.

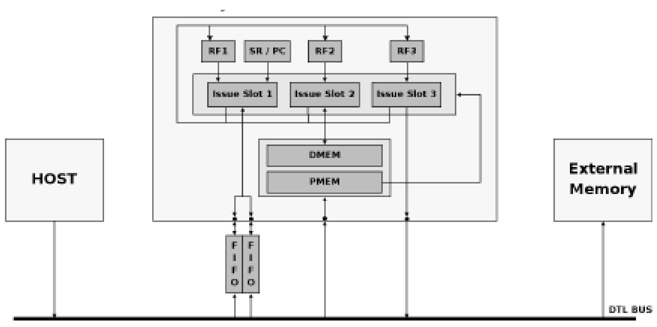

Figure 5. Reference architecture

The optimizations are performed with the objective of reducing the cycle count. The idea is that if the cycle count is reduced with minimal increase in overall architecture power consumption, the execution time (i.e. active time) is reduced and that directly translates to dynamic energy reduction. Leakage reduction techniques then can be applied during the idle time to reduce the standby leakage consumption. The rest of this section presents different optimization steps and their impacts on the cycle count. The result is summarized in Table 3.

1) Code Enhancement:

The following code enhancements were performed to reduce the cycle count.

- **Memory access reduction:** Analyzing the execution statistics, it turns out that a performance bottleneck is having only one load/store unit in issue slot 2 to communicate with the data memory. This results in many stall cycles for instructions requiring access to the data memory. The memory access should be as few as possible because of their limited availability and also their high energy per access. Therefore, the code has to be rewritten into a compact form to avoid many load operations.
- **Conditional statement:** Conditional statements such as if-else or switch-case statements should be removed when possible because they consume clock cycles for checking whether the condition is true or not. Rewriting the code into a loop kernel and removing the conditional statements gives a more compact code, reducing the cycle count.
- **Native data type:** We also remove non-native data types whenever possible. This is beneficial because whenever arithmetic operations are performed on data types shorter than integer, the compiler is obliged to insert sign-extend operations after it to make sure that the value stays within the range of the data type.

Rewriting the most prominent code segment into a loop kernel and changing the non-native data types into 32-bit integer native data type reduce the cycle count from 7494 clock cycles to 5982 clock cycles per sample, a 20% reduction.

2) Circular Buffer

There are six linear buffers in the code, each of which corresponding to a different decomposition level, *n*. In the algorithm, a sliding window principle is applied to the data such that all elements of a buffer were shifted by one to overwrite the oldest sample in the buffer and to insert the new incoming sample at the end of the buffer. Because the buffer size increases from decomposition level *n-1* to decomposition level *n* by 2^n, the required move operations increases by 2^n . On the other hand, the register files are not large enough to keep the entire buffer and it must be stored in the memory. To solve the problem, we used a circular buffer instead of a linear buffer. In terms of hardware, the circular buffer is a storage element with two ports for write and read. The one advantage of circular buffer versus shift buffer is having write- and read-pointers and using them in parallel. The other advantage comes from rotating the read- and write-pointers instead of physically moving elements of a buffer.

In practice, a circular buffer is a linear buffer with circular addressing mode. Therefore a manual bound checking for the end of the circle using modulo arithmetic is required.

After implementing the circular buffer and running the rewritten code on the processor, the cycle count was not decreased. On the contrary*, the cycle count was increased by a factor of 2.* The conclusion is that this implementation is only efficient for processors that support the circular addressing mode.

The modulo arithmetic operation can be replaced by the binary mask operation. In other words, the number to use as a mask in order to perform a modulo *2n* is *2n-1*. The following example explains the concept.

Modulo operation:	x = y % 8	x = y % 16
Equivalent binary mask operation:	x = y & 7	x = y & 15

Implementing the circular buffer using the binary mask operation reduces the cycle count from 5982 to 2352, a 60% reduction.

3) Custom Operations

Silicon Hive provides two ways of running an application on the processor. One is without any customer instruction. The complier decides which path will be taken in the processor. The other way is to force the compiler to use customer instruction. The following custom operations were used and their impacts were analyzed.

- ***Modadd:*** Such a operation can be used in the circular buffer (see III.B.2). With this operation, the the intermediate code from III.B.2) can be rewritten with the custom operation *modadd*.
- ***Mac:*** Since the loop kernel uses convolution, the computation is done using multiplication and a successive addition. This can be also optimized using a multiply accumulate unit (MAC), which is usually available on a common DSP processor.

  However, the reference processor does not support the MAC function unit. Therefore, the processor architecture needs to be modified to implement the MAC operation. To increase the parallelism, it would be more efficient to insert MAC unit into issue slot 1.

Adding the MAC function unit reduces the cycle count from to 2352 to 1713. Further modification was applied to issue slot 2 by removing the multiplication function unit. Since all multiplications in the code is done by the MAC function unit, the multiplication unit is redundant and can removed.

4) Exhaustive Scheduling and Software Pipelining

As a result of the above optimizations, the main part of the code now has an optimized loop kernel that does not have any control flow. Moreover, all iterations of the loop are independent. The computations in any given iteration do not depend on the results of the previous iterations. Thus we can apply *software pipelining*, a.k.a. loop-folding.

We can also have a better utilization of different functional units/issue slots. We use the compiler option for *exhaustive scheduling* in the Silicon Hive compiler. Applying the two mentioned techniques reduces the cycle count from 1713 to 1415.

5) Removing Global Variables

The use of global variables should be kept minimal, because global variables are stored in the memory and loading them from the memory and storing them to the memory is expensive. Whenever possible the global variables should be changed to local variables to keep them in the register files. This step reduces the cycle count from 1415 to 1396.

The next section presents the power measurement results.

Table 3. Performance optimization steps

Step	Optimization Techniques	Cycles/Sample
1	Initial Code	7494
2	Code Enhancement (reducing conditional branches)	5982
3	Employing Circular Buffer	2352
4	Using Custom Operation (mod & mac)	1713
5	Exhaustive Scheduling, software pipelining	1415
6	Code Enhancement (removing globals)	1396

IV. RESULTS

To obtain the power consumption of running the DWT-based EEG algorithm on the optimized architecture, the VHDL model of the processor was generated. Then, the VHDL design was synthesized and placed & routed using TSMC 90 nm process technology. The memory blocks were generated using a commercial memory generator.

The DWT-based EEG algorithm then was simulated on the back-annotated netlist of processor after place and route using Cadence NCSim. The value change data (VCD) information, generated during the simulation was used by Synopsys PrimeTime to get power numbers.

The reference processor runs at 100 MHz and consumes 68.7 micro-Watt/MHz dynamic energy and 100 micro-Watt of leakage. It takes 7494 cycles to process one EEG sample. Given that EEG samples arrive at 1 kHz, the duty cycle of the application is approximately 8%. The active energy/sample is approximately 515 nJ. The leakage energy/sample is 100 nJ. Since the active energy is substantially larger than leakage energy, our initial effort was concentrated on reducing the active energy by reducing the cycle counts as described in the previous Section III.B. The energy consumption per sample calculated from the cycle count at each optimization step is shown in Figure 6.

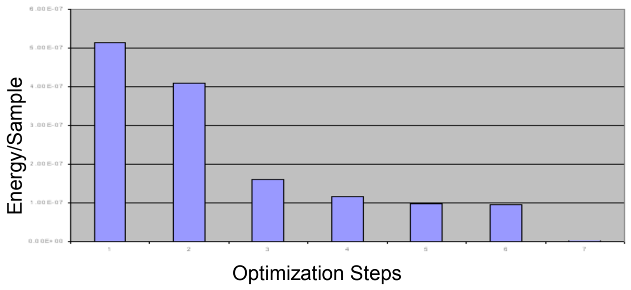

Figure 6. Energy consumption per sample in each optimization step

It shows an initially exponential drop of energy consumption during the optimization steps. After the 4th optimization step, the cycle count goes down slowly and so does the energy saving. The optimization steps reduce the active energy consumption from 515 nJ to 96 nJ. The new duty cycle of the application after optimization is about 1% now. With the reduced duty cycle, the active energy is reduced to 96 pJ, making the leakage energy also a dominant component in the overall system energy consumption. Further optimizations are required to reduce leakage. The leakage power consumption of the architecture is shown in Figure 7 [5].

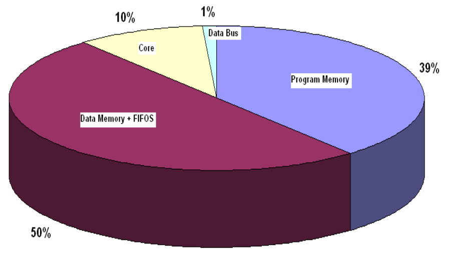

Figure 7. Leakage power break down

It can be seen that the power consumption of memory dominates over the power consumption of the logic. Memory consumes 89% of the total power. This needs to be addressed in future. One way of reducing the memory power consumption was memory width reduction. The power consumption of the program memory is 39% and decreasing the width of the VLIW processor reduces it. However, by having a 2-issue-slot VLIW machine, the cycle count goes up by 6.5% but the power consumption in the program memory goes down. This brings a gain of 30% in the energy consumption.

V. Conclusion

This paper provides a low power platform for processing EEG signal by cross-optimizing the EEG application and the processing architecture. Such processing platform can be embedded into wireless devices for health care monitoring system to perform local signal processing and to reduce the wireless data transfer, which consumes substantial energy. This work is a step toward making the wireless health care monitoring devices a reality. We were able to reduce the dynamic energy consumption (per EEG sample) from 515 nJ to 96 nJ, an 81% reduction. Currently, we are working on adapting the code and the architecture for processing multiple-lead EEG signals using vector processing.

References

[1] B. Gyselinckx, et al., "Human++: emerging technology for Body Area Networks", in Brave New Interfaces: Individual, Social and Economic Impact of the Next Generation Interfaces (Crosstalks) 2007

[2] J. Penders, et al., "Human++: from technology to emerging monitoring concepts", Proceedings of the 5th International workshop on wearable and implantable Body Sensor Network, 2008

[3] N. de Vicq, et al., "Wireless Body Area Network for Sleep Staging", in Proc. Int. Conf. on Biological Circuits and Systems, 2007

[4] J. Penders, et al, "Human++: Emerging Technology for Body Area Network", in *VLSI book*, G. De. Micheili, Ed. Springer, 2007

[5] M. de Nil, et al., "Ultra Low Power ASIP Design for Wireless Sensor Nodes", ICECS 2007

[6] www.picobay.com/projects/2006/05/controlling-video-game-with-brain.html

[7] I. Al Khatib, et al. "MPSoC ECG Biochip: A Multiprocessor System-on-Chip for Real-Time Human Heart Monitoring and Analysis", Proceeding of the 3rd conference on Computing Frontiers, 2006

[8] P. S. Addison, "The illustrated wavelet transform handbook: Introductory theory and applications in science, engineering, medicine and finance", Institute of Physics Publishing, 2002

[9] O. A. Rosso, et al, "EEG analysis using wavelet-based information tools", Journal of Neuroscience Methods, 2006

[10] J. C. Letelier and P. P. Weber, "Spike sorting based on discrete wavelet transform coefficients", Journal of Neuroscience Methods, 2000

[11] L. Qin, L. and B. He, "A wavelet-based time–frequency analysis approach for classification of motor imagery for brain–computer interface applications." J. Neural Eng., 2005

[12] A. Jensen and A.la Cour-Harbo, "Ripples in Mathematics: *Discrete Wavelet Transform*", Springer Press Book, 2001

[13] Matlab 7.1 Release 14, Wavelet Toolbox 4.2, licensed version, description available: http://www.mathworks.com/products/wavelet/description1.html

[14] C. S. Burrs, et al, "Introduction to Wavelets and Wavelet Transforms: *A Primer*". Prentice Hall Inc., 1998

[15] A. Mertins "Signal Analysis: Wavelets, Filter Banks, Time-Frequency Transforms and Applications", Wiley, English Rev. Edition, 1999

[16] D. Lee Fugal,"Wavelets: An In-Depth, Practical Approach for the Non-Mathematician", Space & Signals Technologies LLC, available: www.ConceptualWavelets.com.

[17] Silicon Hive, available: http://www.silicon-hive.com