Ultra Low Power Application Specific Instruction-Set Processor Design for a Cardiac Beat Detector Algorithm

Yahya H. Yassin, Per Gunnar Kjeldsberg
Norwegian University of Science and Technology
Trondheim, Norway
yhyassin@gmail.com

Jos Hulzink, Iñaki Romero, Jos Huisken
Holst Centre / IMEC
Eindhoven, Netherlands
jos.hulzink@imec-nl.nl

***Abstract*—High efficiency and low power consumption are among the main topics in embedded systems today. For complex applications, off-the-shelf processor cores might not provide the desired goals in terms of power consumption. By optimizing the processor for the application, one can improve the computing power by introducing special purpose hardware units.**

In this paper, we present a case study with a possible design methodology for an ultra low power application specific instruction-set processor. A cardiac beat detector algorithm based on the Continuous Wavelet Transform is implemented in the C language. This application is further optimized using several software power optimization techniques. The resulting application is mapped on a basic processor architecture provided by Target Compiler Technologies, and the processor is further optimized for ultra low power consumption by applying application specific hardware, and by using several hardware optimization techniques.

The optimized processor is compared with the unoptimized version, resulting in a 55% reduction in power consumption. The reduction in the total execution cycle count is 81%.

Power gating, and dynamic voltage and frequency scaling, are investigated for further power optimization. For a given case, the reduction in the already optimized power consumption is estimated to be 62% and 35%, respectively.

I. Introduction

In embedded systems design today, there is need for high performance, as well as low power consumption [1], [2]. In recent years, application specific instruction-set processors (ASIPs) have become popular because they simultaneously offer high performance and short design cycles. In contrast to off-the-shelf processor cores, ASIPs include dedicated functional units and machine instructions that speed up execution of the "hot spots" in a given application [1].

In this paper, an algorithm using the continuous wavelet transform modulus maxima (CWTMM) [3] for automatic heart beat detection is studied and implemented in the C language from a Matlab model. The algorithm processes 3 seconds of input data at a time, including 0.5 seconds of overlap with consecutive samples. In a real-time environment, every 2 seconds of the ECG signal must be sampled and buffered before the processing starts. The real-time requirement for this algorithm is to finish processing before the next input buffer is ready. In addition, it is desired to reduce the power dissipation optimally because the algorithm is intended to be used within ambulatory monitoring.

Before an algorithm can be mapped into a processor one has to go through several design steps, as we will show later in this paper. In our case, the algorithm is modeled in Matlab, using built-in Matlab functions which do not exist in programming languages like C. Therefore, the first step is to convert the code into embeddable software. In our project the program language C is used, because it is supported by various development tools for processor design. During the conversion, the algorithm is tested using extensive test benches with test signals from widely used biomedical databases. In order to increase efficiency, all floating point signals are converted into fixed point, and heavy computational parts of the code like square roots and divisions are optimized away. By doing so, we eliminate the need for a floating point emulator on the processor, in addition to complex logic. This results in less power consumption. After the embedded software is optimized, the memory resources are determined by profiling.

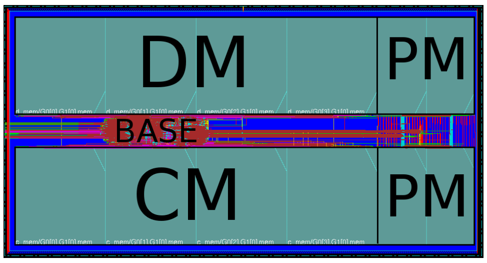

Fig. 1. The optimized processor after place and route.

After the software optimizations, the work on an energy efficient processor architecture starts. Based on the constraints and optimizations made from the embeddable software application, an ASIP is designed. This step involves architecture exploration and mapping of the software application through an iterative process. In our project, the development tools from Target Compiler Technologies are used. An existing basic processor architecture is used as a base, and that architecture is modified for the application. This architecture is shown in Figure 1, where BASE is the processor, PM is the program memory, and DM and CM are the data memories.

The main focus in this case-study paper is to show how state-of-the-art tools combined with a comprehensive design methodology has enabled us to make an ultra-low power implementation of a large application in a short time. We will therefore not go into details regarding the optimization techniques used.

II. Previous work

In this paper, a heart beat detector algorithm is implemented. The physiologic signal used to determine the heart rate is the electrocardiogram (ECG). By analyzing the ECG signal, it is possible to detect heart beats.

The heart beat detector algorithm developed by Romero et al. [3] uses the continuous wavelet transform (CWT) [4] in one of the steps. This algorithm has been tested for various shapes of input signals, and it can cope with significant signal noise [5]. In addition, the algorithm has been further optimized by IMEC-NL for use in ambulatory monitoring. Two comprehensive databases with real-life patient cardiac data are used to test this algorithm; the MIT/BIH database and a database recorded by IMEC-NL [6]. The MIT/BIH is one of the most common databases used for testing beat detection algorithms [7]. The IMEC-NL database contains a set of recorded ECG signals under different levels of activity that produce noise and motion artifacts. Usually, the parameters used to evaluate beat detection algorithms are sensitivity (Se) and positive predictivity (+P). The testing of the algorithm in a Matlab simulation environment resulted in Se = 99.68%, and +P = 99.75% on the MIT/BIH database, and Se = 99.86% and +P = 99.91% in the IMEC database, [5]. The algorithm and the databases were implemented in a Matlab environment, and optimized with built-in Matlab functions.

III. The C application

A. Methodology

Conversion from Matlab code optimized with built-in Matlab functions to a C application is not easily done without changing the algorithm. One possibility is to create a prototype version of the Matlab code, and for every built-in function, one can find the optimal algorithm with the same functionality using only for-loops, while-loops and if-statements. This can be just as time consuming as starting from scratch in C. However, the advantage of this approach is that it is easier to simulate the test signals within the same environment to verify the correctness of the prototype. Instead of starting from scratch in C, a Matlab prototype is created in this project. In this prototype, the functionality of the original Matlab code is re-written without the use of built-in Matlab functions. By making a prototype, it is possible to split the code in several parts, and convert one part at a time. Every part is then validated by interfacing each part directly into the original Matlab code.

The conversion process is divided into four steps, where the application is validated in each step. The first step consists of analyzing the original Matlab code, in addition to the initial simulation of the model. In the second step, a prototype is made, based on the considerations taken during the analysis. All built-in Matlab functions are interpreted by studying their Matlab documentation, and they are implemented using only for-loops, while-loops and if-statements. In the case where a direct conversion is very inefficient, a more optimized solution is chosen. In the third step, the Matlab prototype is manually translated into a floating point C application. After creating the first running version of the C code, the optimization process is started. This optimization step is followed by a fixed point conversion, where all floating point values are converted to integer values.

After the conversion steps, the resulting application is run through profiling in order to find possible bottlenecks and memory leaks. Further, the application is simulated using an instruction-set simulator from Target Compiler Technologies, using a 32 bit processor, in order to determine the memory usage and estimate the cycle counts for the execution of the application.

B. Software optimization results

All memory usage are optimized, and unnecessary code are removed. In addition, complex functions are optimized away. Close investigation in our case made it possible to optimize away all divisions and square roots. The resulting C application is further optimized by using lookup tables, inline functions, reducing read and write operations to global memories, and by eliminating divisions with constant values where it is possible. In addition, some loops are combined, and a special cosine function is implemented using a small lookup table. Some specific values had to be calculated outside the critical loop of the application, and for this purpose, a software division is designed. Some variables are re-used, and pre-processor macros are inserted to prepare the application for processor implementation without the use of standard GNU libraries. Finally, the C application is translated into fixed-point format. This is achieved with only a 0.5% degradation in Se and +P values when tested with the cardiac databases, which are still very good results. As a result of the optimizations the run-time memory usage of the application is reduced significantly, from 1.8 MB to only 512 KB for an ECG sample frequency of 1000 Hz. The program memory requirement is 8 KB in all cases, large enough to store the more than 3300 instruction words required for the optimized version. The application after software optimization used 27.7 million cycles for one 3-second input signal. The unoptimized version could not be run on the processor due to its lack of complex operators and floating point support.

IV. IP designer tools

Target Compiler Technologies is a leading provider of re-targetable software tools for the design, programming, and verification of Application Specific Instruction-set Processors (ASIPs), [8]. The processor is designed with a processor description language called nML, which is a hierarchical and highly structured architecture description language, used to represent ASIP designs at the abstraction level of a programmer's manual [9].

The re-targetable tool-suite from Target is called IP Designer, and it supports both the design and use of embedded processors in a heterogeneous HW/SW Co-design environment. This design environment is used for the ASIP design in our project. In this design environment, C is used to model the software application, and nML is used to model the processor. The IP Designer environment consists of six major packages; a C compiler, a linker, an assembler and disassembler, an instruction-set simulator, a hardware description language (HDL) generator, and a test program generator. These tools are re-targetable, and can be applied to different instruction-set architectures.

V. Processor design

A. Methodology

Before the application is mapped onto the platform architecture, a basic general purpose processor architecture (called BASE) from Target Compiler Technologies is converted from 16 bit to 32 bit. The second step is to simulate the application on the processor with the instruction-set simulator (ISS) from Target, and optimize the execution of the application through an iterative process. By exploiting the profiling information produced by the ISS, and by reading the assembly code produced by the C compiler, it is possible to identify the instructions in the critical loop, and optimize them by using a number of techniques suggested in [10]. The main purpose of this optimization is to reduce the cycle count of the application, in order to reduce the overall power consumption.

VHDL files of the optimized and unoptimized processor are automatically generated using the HDL generation tool. The memories had to be added to the generated VHDL files, because the HDL generation tool only generates interfaces for the memories in the design. In this project, 90 nm TSMC low power memories from Virage Logic Corporation are used. The processor and memories are interfaced to each other through a top module, before validating the design through RTL simulation with the $ncsim$ tool from $Cadence$. The design is then synthesized, and place and route is performed with $Encounter$ from $Cadence$ using a 90 nm low power TSMC process.

After the place and route, the location of the individual standard cells are known. Standard cells from a low power TSMC library are used in this project. The capacitance on the output of each standard cell is then calculated by adding both the input capacitance of the connected standard cells, and the interconnect (wire) capacitance. After the capacitance and resistance (from the layout of the design) of each node is known, and the drive strength of the cells is given, the timing of the design is calculated with $Primetime$ from $Synopsis$. The timing is calculated by adding the interconnect delay, and the delay from the standard cells (from input to output) in all paths. During the netlist simulation, several value change dump (VCD) files are created in order to extract information about the power dissipation (power numbers).

In the final step, the power dissipation information is extracted with $Primetime$. The VCD files created in the previous step are analyzed through this tool, where information related to the power consumption are reported in a text file.

Two types of simulation environments are used for validation; self-checking simulation and simulation through text files. The self-checking simulation is integrated within the main function of the application, where the results are written to a specific memory location. Similarly, a TCL script is made to read text files with new input stimuli and update the memory locations reserved for the input data for the test environment. Those text files include stimuli from the biomedical databases.

B. Hardware optimization results

Several optimizations are applied to the processor in order to reduce the execution cycle count of the application. Where it is possible, two or more instructions are combined to execute in parallel, within the same clock cycle. Additional functional units are implemented, and the data memory of the processor is divided in two smaller memories, in order to access in parallel two memories within the same clock cycle. In the main critical loop, instruction level parallelism is added with two load and one multiply-accumulate operation, reducing the main critical loop to one single clock cycle. All these optimizations resulted in a reduction of the total execution cycle count by 81%.

Based on the profiling reports generated by the IP Designer tools, the reduction of program execution cycles are shown in Figure 2, for each optimization step. The same signal is simulated using ECG sample frequencies of 1000, 500, 360 and 198 Hz, in order to illustrate how the different execution times vary with different input sizes (number of samples). In the figure, the relative improvement ratio is the same for all frequencies. $unoptproc$ is the processor before hardware optimizations (but with software application optimizations in place). $Cust\ MAC$ is the improvement after the customized MAC operation has been added. Similarly, $Par\ ld, st$ shows the optimization after the parallel load and store have been added. $Par\ MAC, ld$ is after the parallel MAC and load instruction is applied, and $Par\ load, eq/gt, sel$ shows the result after all custom instructions are created. $Ctrl\ sig\ in\ local\ reg$ is after the final optimization, where some control signals are defined locally instead of globally. The clock frequency of the processor is chosen to be 100 MHz in all of our experiments. This fits the specifications of the memory library, and also resulted in straightforward synthesis for all processor versions. The different hardware optimization steps did not influence the quality of the application, as measured with Se and +P numbers.

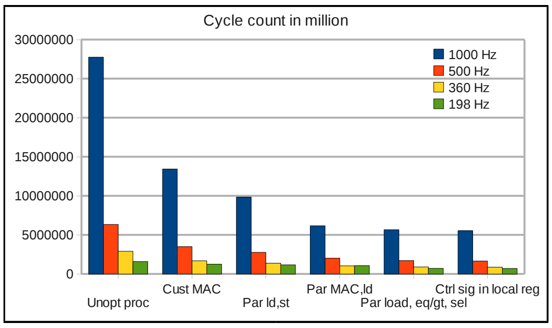

Fig. 2. Improvement in execution cycles after every optimization step.

As seen in Figure 3, the dynamic power during execution is higher for the optimized processor than for the unoptimized processor. However, the duty cycle of the processor, i.e., the length of time the processor needs to execute its task with respect to the time budget available, is reduced from 14% to 2.8%. Difference in leakage power consumed by the two processors is not shown in Figure 3. The actual difference was 13% in favor of the unoptimized version.

The total average power dissipation of the different parts of the unoptimized and fully optimized processors are shown in Figure 4. The total power consumption of the optimized version is reduced by 55%. The dynamic power dissipation is reduced by 78%, while, as mentioned, the leakage power is increased by 13%.

The optimized processor's total average power for the case of a 1000 Hz ECG sample frequency is estimated to be further

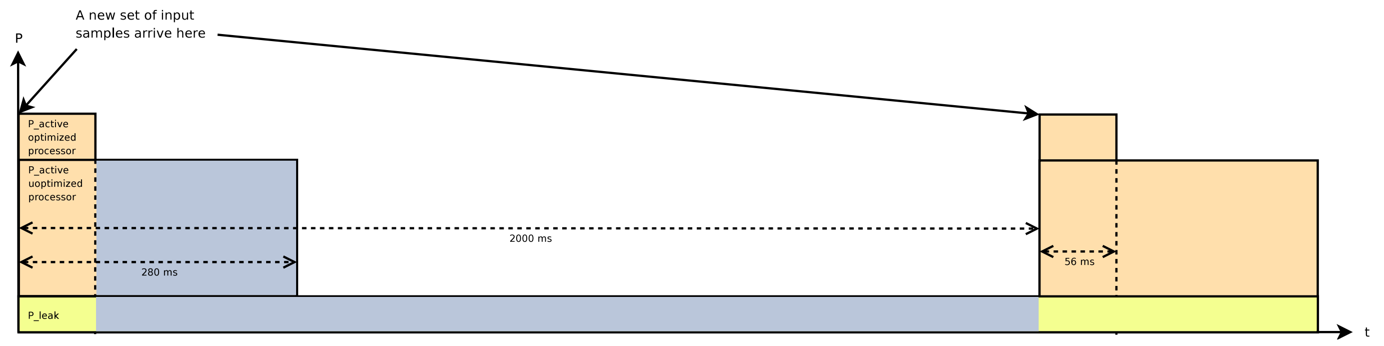
A new set of input samples arrive here
P
P_active optimized processor
P_active uoptimized processor
2000 ms
280 ms
56 ms
P_leak
t

Fig. 3. Total average power consumption of the processors with 100 MHz clock frequency and 1000 Hz ECG sample frequency.

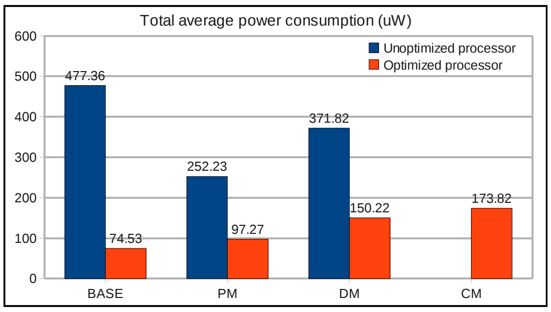

Fig. 4. Total average power dissipation of the processors compared.

reduced by 35% by applying dynamic voltage and frequency scaling. This number is found through an investigation of the supply voltage vs. delay characteristics of the transistors. For the processor to finish its work on a set of data just before new data arrives, its minimum clock frequency must be 3 MHz. The supply voltage is scaled down as much as possible while still maintaining this frequency. Alternatively, by applying power gating, one can turn off the supply voltage of the processor after it has finished processing, and turn it on again just before the next set of samples are about to arrive. The leakage power consumed when the processor is idle (in Figure 3), would then be reduced to approximately zero. This gives a further reduction in the total average power for the optimized processor of 62%.

VI. Discussions and Conclusions

In this paper we have shown how we, using state-of-the-art design tools and a comprehensive design methodology, have been able to go from an abstract Matlab description of a cardiac beat detector algorithm to an ultra low power ASIP implementation of the same application. The work was done as part of a master thesis project and went from no knowledge about the tools or application to a full implementation, including layout, in about four man-months. First, software optimizations resulted in a reduction in data memory requirements from 1.8 MB to 512 KB. The first software version that could run on a processor required 8 KB of program memory and 27.7 million cycles for a 3-second sequence of an ECG input signal sampled at 1000 Hz. After hardware optimization of the ASIP, the required cycle count was reduced by 81% to 5.3 million. The hardware optimizations also reduced the power consumption of the processor with 55%, from 1100 μW to 500 μW.

However, there are more optimizations that could be done in order to make the processor more power efficient. A change in the application that would result in less memory usage, would for example automatically reduce the memory sizes and hence the power consumption. Furthermore, with the selected clock frequency, the optimized processor is only active 2.8% of the time. This opens for use of dynamic voltage and frequency scaling or, alternatively, power gating. Estimates show that with a 90 nm low power standard cell library from TSMC, the power reduction achieved with power gating is much larger (62%) compared with that of voltage and frequency scaling (35%). With newer technologies leakage power is becoming even more important compared to dynamic power. This indicates that it is most beneficial to reduce the duty cycle of the processor as much as possible, and apply power gating when it is idle. The application is currently being re-designed with focus of reducing the data memory usage. In parallel, power gating is being applied.

VII. Acknowledgements

The work in this project started in January 2009 and ended in June the same year as a master thesis. The task was a challenge, and thanks to good development tools and support, from both IMEC-NL and Target Compiler Technologies, a complete design was finished in such a short time.

References

[1] K. Karuri et al., “Fine-grained application source code profiling for asip design,” *Proc. 42nd Design Autom. Conf., 2005*, pp. 329–334, June 2005.
[2] T. Glokler and H. Meyr, “Power reduction for asips: a case study,” *Proc. 2001 IEEE Workshop on Sig. Process. Sys.*, pp. 235–246, 2001.
[3] I. Romero Legarreta et al., “Continuous wavelet transform modulus maxima analysis of the electrocardiogram: beat characterisation and beat-to-beat measurement,” *Int. Jour. of Wavelets, Multires. and Inform. Process.*, vol. 3, no. 1, pp. 19–42, 2005.
[4] P. Schniter, “Continuous wavelet transform.” Connexions, June 9 2005. http://cnx.org/content/m10418/2.13/ on April 06, 2009.
[5] I. Romero Legarreta et al., “R-wave detection using continuous wavelet modulus maxima,” in *Comp. in Card., 2003*, pp. 565–568, Sept. 2003.
[6] I. Romero, B. Grundlehner, and J. Penders, “Robust beat detector for ambulatory cardiac monitoring.” in Proc. 31st EMBS, in Press, 2009.
[7] A. L. Goldberger et al., “PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals,” *Circulation*, vol. 101, no. 23, pp. e215–e220, 2000 (June 13).
[8] “About target.” http://retarget.com/about.php on May 16, 2009.
[9] P. Mishra and N. Dutt, *Processor Description Languages, Volume 1*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008.
[10] L. Benini and G. De Micheli, “System-level power optimization: techniques and tools,” *ACM Trans. Des. Autom. Electron. Syst.*, vol. 5, no. 2, pp. 115–192, 2000.