Ultralow power processor employing block instruction for ECG applications

Jing Qiu¹, Xiaoyan Xiang², Zhijian Chen^{1a)},
Jianyi Meng², and Yong Ding¹

¹ Institute of VLSI Design, Zhejiang University, Hangzhou 310027, China

² State Key Laboratory of ASIC & System, Fudan University,
Shanghai 201203, China

a) chenzj@vlsi.zju.edu.cn

Abstract: Conventional processors that execute a single instruction at a time are easy to implement but lack the power efficiency. This paper presents a novel hardware-software co-designed method to save power consumption for ECG applications. The software generates block instruction which is comprised of several atomic operations, reduces the instruction memory space, and merges memory operations within a block. The hardware executes instructions block by block, eliminates redundant fetching and decoding operations. The experiments indicate that the proposed design methodology can reduce the active power consumption and code size by 40% and 55% relative to CK802 (a conventional processor).

Keywords: ECG, ultralow power processor, block instruction, hardware-software co-design

Classification: Electron devices, circuits, and systems

References

- [1] F. Massé, M. V. Bussel and A. Serteyn: TECS **12** (2013) 102. DOI:10.1145/2485984.2485990
- [2] X. Zhang and Y. Lian: IEEE Trans. Biomed. Circuits Syst. **8** (2014) 834. DOI:10.1109/TBCAS.2013.2296942
- [3] P. Hamilton: IEEE Computers in Cardiology (2002) 101. DOI:10.1109/CIC.2002.1166717
- [4] N. Boichat, D. Atienza and N. Khaled: IEEE Sixth International Workshop on BSN 2009 (2009) 256. DOI:10.1109/BSN.2009.30
- [5] C-SKY Microsystems: <http://www.c-sky.com/>.
- [6] B. U. Kohler, C. Hennig and R. Orglmeister: IEEE Eng. Med. Biol. Mag. **21** (2002) 42. DOI:10.1109/51.993193
- [7] U. Banerjee: ACM TOPLAS **33** (2011) 10.
- [8] J. Stanier and D. Watson: ACM CSUR **45** (2013) 26. DOI:10.1145/2480741.2480743

1 Introduction

In the past few years, wireless body sensor networks (WBSNs) have been developed into a key mobile health monitoring technology. Electrocardiogram (ECG) nodes that record the electrical activity of the heart represent a vital WBSN technology for continuous cardiac monitoring. Since ECG nodes have significant limitation of battery capacity, ultralow energy consumption is needed heavily to extend the battery life. The processor and memory together consume more than 60% of the total system power [1], therefore a substantial amount of power can be saved by developing a more energy efficient processor and memory architecture.

The traditional approach seeks to economize processor power through architecture-level optimization such as streamlining the components and reducing the instruction set. However, excessively streamlined architecture and instruction set design can result in a substantial rise in power consumption when facing applications other than those for which the processor was specifically designed.

This paper presents a novel processor architecture. It is based on classic RISC architecture and aggregates low power oriented optimizations at both hardware and software level to customizes energy efficient block instructions for different applications. A software based post compilation tool is added to convert the original fine-grained RISC instructions into coarse-grained block instructions containing several atomic operations, pre-decode the block instruction and merge adjacent memory operations within a block. Furthermore, the compilation tool fixes the rule for allocating registers and uses a dictionary assembly method to reduce the instruction space. At the hardware level, the processor starts from execution unit in the driven of instruction memory and execute instructions block by block.

The rest of this paper is organized as follows: Section 2 analyzes the functionality of ECG applications and provides the motivation for introducing block instruction. Section 3 describes the detailed block instruction methodology. Section 4 presents the experimental results which are compared with a conventional processor. Finally, Section 5 presents the conclusions of the paper.

2 Motivations

QRS wave detection acts as a fundamental operation in nearly all ECG algorithms. It can be generally classified into two categories: the time-domain manipulation and the frequency-domain analysis [2]. To analyze the program characteristics and power consumption of QRS detection algorithms, a time-domain manipulation [3] and a frequency-domain analysis [4] were tested as two typical algorithms on a low

Fig. 1. Relative power consumption of ECG applications

power RISC processor CK802 [5]. More detailed information regarding the experimental method employed can be found in Section 5.

In conventional ECG nodes, memory power consumption represents 23% of total power consumption, while processor power consumption is approximately 43% [1]. The power consumption of each module in CK802 with these two typical algorithms is also similar. The power consumed during fetching and decoding operations is about 26% of the total processor power consumption, the power consumed during memory access is 29%, and the execution power consumption which represent the actual computational activity is about 45% (as shown in Fig. 1). Reducing redundant power during fetching and decoding operations and retaining only the actual computational power can greatly reduce the power consumption of the processor.

Table I. The summary of ECG algorithms

Author (Published year)	Background	Accuracy	Algorithms	typical functions	Reference
Hamilton & Tompkins (1986)	Real time QRS detection	99.69%	Optimized feature detection	Threshold detecting	[3]
Suppappola & Sun (1994)	Real time digital QRS detection, easy to implement on microprocessor	93.06%	Multiplication of backward differences	Nonlinear transform	[6]
Poli et al (1995)	QRS detector facing noise, artifacts, variability of ECG morphology	99.0%–99.8%	Polynomial filter and maxima detector	Polynomial function	[6]
Ruha et al (1997)	Reliable QRS detection in both ambulatory and clinical use	97.8%	Optimized filter and dual edge threshold detection	Adaptive threshold detector	[6]
Afonso et al (1999)	Multi-rate digital signal heartbeats detection	99.56%	Filter banks	Time and frequency analysis	[6]
Hamilton (2002)	Most basic ECG analysis operations	96.48%	Peak detection with multi-filter	Peak detection in time domain	[6]
Kohler et al (2003)	Very noisy ECG signal	99.57%	Zero crossing counts	Zero crossing detection	[6]
Boichat et al (2009)	Robust, efficient and reliable automated ECG analysis	99.8%	Wavelet transform	DWT	[4]

By analyzing the two QRS detection algorithms above and some typical detection algorithms discussed elsewhere [6] which are listed in Table I, three primary characteristics for ECG applications can be given as follows.

1) Algorithm stability

Based on the different measurement techniques and accuracy requirements, many QRS detection algorithms have been proposed in recent years, nevertheless, the algorithm does not require frequent updates if the application scenarios remain unchanged. The stability of the algorithm allows for an extended low power oriented compilation for reducing runtime power consumption.

2) High repetition

Detection algorithms are comprised of typical functions such as band pass filter and cosine transfer. These typical functions are always made up by repeated operations and steps. A series of instruction segments can be abstracted from these repeated operations and steps. The instruction segment contains several atomic instructions inside (as shown in Fig. 2 at the bottom of the middle). That is to say, the entire detection algorithm can be constructed from a finite number of instruction segments with different operands and sequences. As illustrated in the Fig. 2, different number stands for different instruction segments. All illustrated typical functions are formed by only 6 instruction segments.

Fig. 2. Diagram of block based approach.

3) Memory access concentration

The memory operations in ECG programs primary involve accessing constants and passing the parameters. This tends to result in a nearly continuous access to a minority number of memory addresses. Merge all the adjacent memory operations can result in a reduction of memory access power consumption.

3 Frameworks

In conventional architecture, an atomic instruction is considered to be a minimal unit for an individual operation. The atomic instruction based approach brings flexibility to a program at the cost of redundant power consumption. For example, fragmented instruction fetch operations result in frequent memory access; undue instruction detail results in bloated memory space; fine-grained instruction distribution fails to provide sufficient scope for power optimization. Considering the architectural issues discussed above, this paper presents a block based approach for designing an ultralow power ECG processor. In order to attain the objective of eliminating redundant hardware power, and reducing instruction space, the proposed design transfer classic atomic instructions into coarse-grained block instruc-

tion. It includes a post-compilation block scheduler at the software level, an execution processor and an active memory at the hardware level.

3.1 Block instruction

As mentioned above, there are many repeated instruction segments in QRS detection algorithms. We abstracted these repeated segments and constructed them as block instructions. A block instruction always contains several atomic operations inside which act as classic atomic instructions. The processor executes these block instruction as the most basic operational unit.

A block instruction can only be entered at the first atomic operation, and only the last atomic operation can cause the program to begin executing another block. There are only two end conditions of a block, that is branch operation or the block size is extended. If the block instruction terminates with a branch operation, there can be only one branch target address except the adjacent block. Therefore, the block is effectively executed as a single step. The characteristics of non-interrupt inside block instruction provides greater scope for incorporating techniques within a block to reduce redundant active power and instruction space.

Fine-grained memory operations require more power consumption due to frequent memory read/write operations and data bus switching. The absence of internal branch operations facilitates the merge of adjacent memory access in a block instruction. This replaces fine-grained access with coarse-grained memory access, therefore, reduces the power consumption of data memory operations.

The absence of interrupt makes presetting relationship of the destination registers in a block possible. We uses predefined criterion to allocate destination registers. After that, only a mark is subsequently required to record all the destination registers in a block instruction.

Since the entire program can be decomposed into a finite number of instruction segments with different sequences and operands, we used instruction block samples to record all these segments. As a result, only the sequences and operands are required to record, rather than the whole instruction information of the program. Based on this dictionary assembly approach, the block instruction can significantly reduce the instruction space and the times for processor to fetch the instruction.

3.2 Block scheduler

A block scheduler is a post-compilation optimization tool after conventional compilers. It re-optimize the atomic instructions, which includes removing redundant operations and reducing instruction spaces. The block scheduler finally translates the program into block instructions. The proposed block scheduler consists of four passes: rough blocking, register allocation, block division and optimization (As shown in Fig. 3).

Rough blocking pass parses the entire program and divides it into rough block according to branch instructions. Trace scheduling [7] and branching expansion [8] are used to extend the rough block size.

Register allocation pass reallocates registers using predefined criterion for destination registers.

Fig. 3. Schematic describing the block scheduler

Block division pass parses the entire program by the unit of rough block. The scheduler first compares the program with existing dictionary entries and picks out hit blocks. Hit block is defined as all instructions inside can be found in a dictionary entry. Then we use code-word to replace the hit blocks, the mismatch instruction in the dictionary entry is replaced with a dummy instruction. Then block scheduler generates new dictionary entries for the remainder of the program to cover the whole rough block.

When the instruction set and system architecture is determined, the information which the processor issues to execution units after the decode stage can be easily obtained. Optimization pass pre-decode instructions for these information to eliminate the decoding process. Optimization pass also merges memory operations in each block, uses code-word with its operands and sequences to replace the program and assembles the programs in a block way.

Finally, the block scheduler assembles the program into three parts: block alphabet, flow dictionary and operand dictionary (as described in Fig. 4). The block alphabet maintains all the instruction block samples with fully decoded information. A single entry of block alphabet is separated into several atoms where each atom represents a single atomic operation. In detail, an atom includes unit select which indicates the execution unit operating the instruction, sub function which indicates operand functionality, detail information which indicates the method to prepare the operands. The flow dictionary maintains program control flow by saving the code-word of block alphabet strictly according to the program flow. If the block ends with a branch operation, the flow dictionary also contains the target address. The operand dictionary maintains the source registers and destination register marks. The order of this dictionary shares a one-one correspondence with flow dictionary. As the code-word is shorter than the original instruction, instruction space is reduced by dictionary assembly method.

Fig. 4. Schematic describing block instruction

3.3 Execution processor and active memory

The atomic instruction is always handled in five steps: instruction fetch, decode, execution, memory operation, and write back. However, the fetch and decode operations are always repeated and redundant since ECG applications with operations of high repetition. The block instruction which is executed by execution processor leaps over these redundant procedures and starts from the execution units.

Fig. 5. Schematic describing block instruction hardware

The basic operations to execute a block instruction by the execution processor can be explained as follows. Firstly, the active memory generates a block instruction which awaits execution in the block board. Afterwards, the block board dispatches one single atomic operation to the corresponding execution unit by unit select signal. After execution, the block board will dispatch the next atomic operation. Since the block instruction in the block board employs fully decoded information, the processor consumes almost no decoding power. When all operations in a block instruction have been executed, the execution processor will request the memory to generate another block instruction.

The active memory behaves more intelligent than conventional memory in that it drives itself to supply block instructions for the processor. It consists of four

components (as shown in Fig. 5): block memory to store the program, data memory to stores all intermediate results during operations, memory counter to trace the program and memory engine to package the instruction.

Block memory has three sub memories: block alphabet memory to save the block alphabet, flow dictionary memory to save the flow dictionary, operand dictionary memory to save the operand dictionary.

Memory counter uses the target address recorded in the flow dictionary memory to calculate the successor block counter.

Memory engine packages block instruction for the block board. The memory engine reads the block counter from the memory counter and treats it as an address to access the flow dictionary memory and the operand dictionary memory. Using the code-word from the flow dictionary memory, the memory engine indexes the block alphabet memory and gets the detailed decoded information. It combines the decoded information and its operands to package a fully decoded block instruction, then sends it to the block board. The block board works as the interface between the processor and memory. It dispatches the block instruction to the processor, and also provides the memory with the feedback of processor execution state.

Fig. 6. Experiment platform

4 Experiments

We implemented the proposed design from CK802 [5], a RISC embedded processor designed by C-SKY Microsystems. CK802 is an ultralow power processor which employs reduced instruction set, dynamic power management and low voltage power supply. As illustrated in Fig. 6, a block scheduler was introduced after C-SKY compiler to transform the atomic instruction into block instruction. Some modules in the execution unit were rewritten to implement execution processor. We also rewrote the memory structure. The open source time domain ECG analysis software designed by PhysioNet [3] is used as typical time-domain algorithms and DWT software proposed by Nicolas Boichat [4] is used as typical frequency-domain algorithms to evaluate the effectiveness of the proposed design

methodology. The design was tested with five sequences of 30 minute data records (rec 100, 105, 108, 203 and 222) obtained from the MIT-BIH database. All experimental data comparison below are between CK802 and the new processor employing block instruction.

Fig. 7. Program compress ratio relative to the numbers of atomic operations in one block

The block scheduler assembled the traditional atomic instruction into block instruction in four passes. After rough blocking pass, the average size of raw block is 10.94. In block division pass, different number of atomic operations inside a block instruction was tested, from 2 atomic operations in one block instruction to 10 atomic operations in one block instruction (as shown in Fig. 7). The more atomic operations in one block instruction, the higher compression ratio, but also more dummy operations in one block instruction. The instruction space reduced most when there are four atomic operations in one block instruction. There were only 0.98 dummy operations in a block instruction on average (as illustrated in Fig. 8). The block scheduler abstracted seven block alphabet entries for time

Fig. 8. Typical program after block scheduler

domain algorithm and eight block alphabet entries for frequency domain algorithm (as shown in Fig. 6). In the optimization pass, 44 adjacent memory access operations were merged in time domain algorithm and 414 adjacent memory access operations were merged in frequency domain algorithm.

Table II. Algorithm symbol description

Symbol	Detailed description	Algorithm subordinate
hp	high pass filter	time domain algorithm
lp	low pass filter	time domain algorithm
mw	move window average	time domain algorithm
pd	peak detection	time domain algorithm
df	derivation filter	time domain algorithm
rr	remove redundant	frequency domain algorithm
rs	refresh suspect	frequency domain algorithm
zf	finding zero	frequency domain algorithm
md	max detection	frequency domain algorithm
mm	modulus maxima	frequency domain algorithm
hp	high pass filter	time domain algorithm

For a clear display of the experimental results, we divided each algorithm into 5 portions (as listed in Table II). Two aspects including instruction space and memory access times were tested by FPGA prototyping verification. Power consumption of the processor and memory were tested by post simulation on VCS tools with TSMC 65 nm technology low leakage standard cell library.

Due to the dictionary assembly method, the required instruction space is only 2730 bytes for time domain algorithms and 18270 bytes for frequency domain algorithms. The instruction space has been reduced to approximately 45% of the original processor on average (46.62% for time domain algorithm and 42.68% for frequency domain algorithm) (as shown in Fig. 9).

Fig. 9. Instruction space reduction relative to conventional processor

The processor employing block instruction only access the instruction memory each block instead of each atomic instruction. In other words, it accesses instruction memory at a time while conventional processor accesses memory for four times.

Instruction memory access times is reduced to 35.7% for time domain algorithm and 38.2% for frequency domain algorithm (as shown in Fig. 10).

Additionally, data memory access times is reduced since adjacent memory access operations in one single block instruction were merged together. The data memory access time is reduced to approximately 90% (91.07% for time domain algorithm and 92.45% for frequency domain algorithm) compared with before (as shown in Fig. 10).

Fig. 10. Memory access time reduction relative to conventional processor

Due to the reduction of memory access, and elimination of redundant instruction fetching and decoding operations, the dynamic power consumption to execute the ECG applications for the processor and memory is reduced from 48.34 pJ/task to 29.08 pJ/task in time domain algorithm and from 216.02 pJ/task to 141.39 pJ/task in frequency domain algorithm. The average power estimates are 60.16% and 65.45% compared to the power consumption when implemented on CK802 (as shown in Fig. 11).

Fig. 11. Dynamic power consumption relative to conventional processor

5 Conclusion

This paper presents a low power oriented optimizations at both hardware and software level. It uses a more coarse-grained view of the entire program, and introducing block instruction to reduce power consumption. The experimental

results indicate a 40% reduction in active power and 55% reduction in instruction space relative to implementation on a conventional ultralow power processor.

In the future, the proposed design methodology will be applied to different kinds of WBSN applications. Further efforts toward a hardware-algorithm collaborative methodology are also required to further reduce power consumption.

Acknowledgments

This work is supported by State Key Laboratory of ASIC & System, China.