A Power Reduction Technique with Object Code Merging for Application Specific Embedded Processors

Tohru Ishihara Hiroto Yasuura
Department of Computer Science and Communication Engineering
Graduate School of Information Science and Electrical Engineering
Kyushu University
6–1 Kasuga-koen, Kasuga-shi, Fukuoka 816-8580 Japan

Abstract— In this paper, a power reduction technique which merges frequently executed sequences of object codes into a set of single instructions is proposed. The merged sequence of object codes is restored by an instruction decompressor before decoding the object codes. The decompressor is implemented by a ROM. In many programs, only a few sequences of object codes are frequently executed. Therefore, merging these frequently executed sequences into a single instructions leads to a significant energy reduction. Our experiments with actual read only memory(ROM) modules and some benchmark program demonstrate significant energy reductions up to more than 65% at best case over a instruction memory without the object code merging.

1 Introduction

An important class of digital systems includes applications, such as video image processing and speech recognition, which are extremely memory-intensive. In such systems, a much power is consumed by memory accesses. In most of today's microprocessors, an instruction memory including a cache memory is one of the main power consumer. The on-chip caches of the 21164 DEC Alpha chip dissipate 25% of the total power of the processor. The StrongARM SA-110 processor from DEC, which specifically targets low power applications, dissipate about 27% of the power in the instruction cache[1]. Thus, utilizing low-power memory organizations can greatly reduce the overall power consumption in the system. In addition, it is a noteworthy that the power consumed by memory accesses certainly increases as memory size is increased[2, 3, 4]. Therefore, low power techniques which reduce power consumed by memory accesses become more important for prospective memory-intensive applications.

Especially for embedded systems, these low power oriented applications also require design flexibility, which results in the need for implementation on programmable hardware platform. Current semiconductor technology allows the integration of processor cores and memory modules on a single die, which enables the implementation of a system on a single chip. Consequently, the design productivity along with the traditional synthesis process has not followed the exponential growth of both applications and implementation technology. The shrinked time-to-market has made this situation worse. There is a wide consensus that only a reuse of highly optimized cores can match the demands of the pending applications and the potential ultra large scale integration. Therefore, a low power core-based system-on-chip, consisting of easily reconfigurable cores attracts much interest of all silicon vendors.

In this paper, we propose an application specific power optimization technique utilizing object code merging. Frequently executed sequences of object codes are merged into a single instruction. A decompressor is use to restore the merged object codes and implemented by a small ROM. Our technique targets a typical application specific system-on-chip, consisting of a simple processor core and reconfigurable two instruction memories : a main program memory and a decompressor. A major premise of the target system is that the programs are stored into embedded ROMs because the programs need not to be modified after system design is completed. We assume that a compiler optimizes size of the two memories and determines which sequences of object codes should be merged into a set of the single instructions so as to minimize total read energy consumption.

The rest of the paper is organized in the following way. In Section 2, we discuss the motivations for our work and present our concept to optimize the instruction memory. A novel power optimization technique with the code merging technique is proposed in section 3. Section 4 presents experimental results and discussion on the effectiveness of the approach. The paper is concluded in Section 5.

2 Motivations and Our Approach

Our memory optimization technique is based on the following two facts.

- The bigger memory size becomes, the more energy will be consumed.
- Most of parts of the program is rarely executed.

2.1 Area-Power Correlation

A curve labeled **Divided Array** in Figure 1 shows energy consumption in ROM modules whose array part is divided so as to minimize energy consumption. The basic idea of the divided array is to divide the memory in different blocks and powered-up only a subset of them for any one access[5, 6, 7, 8]. This curve is approximated using information of load capacitance of sense amplifiers, bit lines, word lines, and address decoders of actual ROMs. The 32 bits ROMs ranging in words from 64 to 4,096 are generated by Alliance CAD System Ver. 3.0 with $0.5\mu m$ double metal CMOS technology. The curve demonstrates that the energy consumption in ROMs strongly depends on the number of words. We can approximate **Divided Array** curve as (1). Experimental results shown in Section 4 are derived using (1).

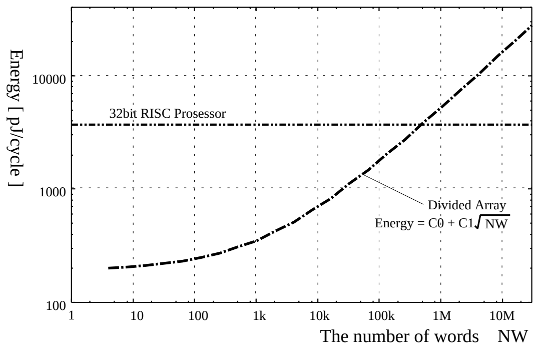

Figure 1: The number of words VS. read energy consumption

$$E_{model} = 190 + 4.8 \cdot \sqrt{NW} \quad [pJ/cycle] \qquad (1)$$

2.2 Memory Reference Locality

It is well known fact that only a few parts of programs are frequently executed in many application programs, and therefore, reducing energy for such a frequently executed instructions is effective way for energy reduction[9]. To demonstrate this fact, we measured access locality of instruction memory with three kinds of programs: Arithmetic calculator, MPEG2 decoder, and MPEG2 encoder. The measured results are shown in Fig. 2. Vertical axis of a line chart in Fig. 2 represents the number of execution cycles produced by a few basic blocks whose total size(the number of words) is only 1% of total program size. In the benchmark programs, only a few basic blocks whose total size is only 1% of total program size produce more than 50% of execution cycles. Reducing the energy dissipation of frequently accessed memory blocks is effective way to reduce total energy consumption in memory.

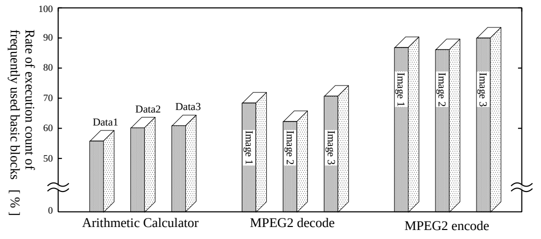

Figure 2: Memory reference locality for sample programs

2.3 Our Approach

Merging frequently executed sequences of object codes into a set of single instructions reduces energy of memory, because the number of memory access to the main program memory is extremely reduced. However, merging too many sequences into single instructions leads to an increase of energy consumption in the decompressor. Our power optimization technique finds optimal point of this trade-off where total energy consumption is minimized.

In many embedded systems, object code of the application programs need not to be modified after system design is completed. Therefore, the object code of the programs are stored into ROM. In this paper, we also target systems which assume that the object code is stored into ROM and is not modified after the system organization is fixed. The optimization technique targets systems which assume the following.

- Instruction memories are organized by two on-chip ROMs, a main program memory and an instruction decompressor.
- A compiler determines which sequences of the object codes should be merged into single instructions so as to minimize total read energy consumption.

It is also possible to develop the decompressor with the wired logic. However, developing the decompressor with the wired logic requires much more turn around time than the decompressor design with ROMs does. The most important merit of utilizing ROMs for the decompressor are design flexibility and suitability for IP-base system design. Our code merging technique is well-suited for systems employing IP cores whose internal architecture can not be modified, because our technique requires no modification to the processor architecture.

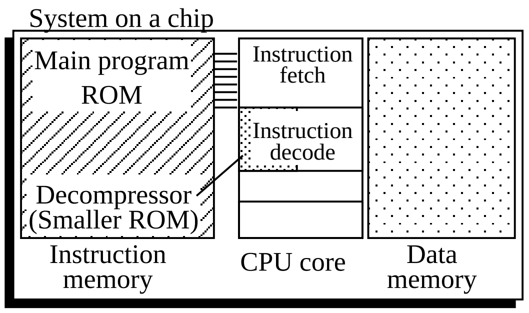

Figure 3: Power reduction considering memory reference locality

3 Memory Power Optimization with Object Code Merging

At first, we present a power optimization technique based on the object code merging. Next, we formulate a *memory optimization problem* as a 0-1 integer programming problem. The *memory optimization problem* is a problem to determine which sequences of object codes should be merged into a set of single instructions.

3.1 Optimization Flow

The procedure of our power optimization technique is described below.

1. Given the object codes of the target program.
2. Measure the execution count of each basic block using some sample data.
3. Formulate energy dissipation of ROM modules as the function of memory size. The energy dissipation model must be made considering circuit and process technologies applied to ROM modules. In this paper we use (1) as the ROM power model.
4. For a given set of basic blocks with the information of the execution count of each basic block and a given energy dissipation model of ROM modules, optimize memory organization so as to minimize average of energy consumption. We give a detailed explanation about the *memory optimization problem* in section 3.3.
5. Adjust the physical size of both the main program memory and the instruction decompressor to the minimum required area.

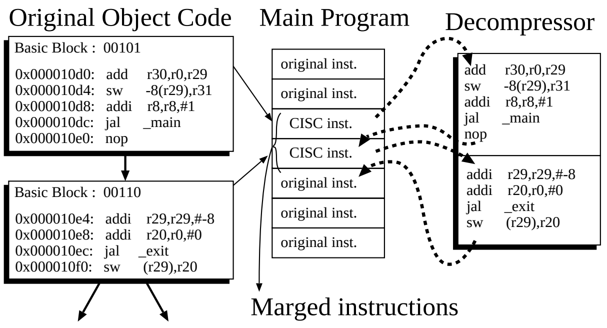

Figure 4: Power reduction based on the object code merging

3.2 Architecture

The frequently executed basic blocks are replaced with a special instruction, named **CISC instruction** and this instruction is allocated into main program memory as shown in Fig. 4. The CISC means a complex instruction set computer. When the CISC instruction is executed, the decompressor is activated and merged object codes are restored to the original object codes, otherwise, the fetched instruction is directly executed without activating the decompressor. The decompressor is also implemented by ROM circuit. The key point of this power optimization method is that only a few parts of object code are replaced with the CISC instruction at compiling phase, so as to keep energy consumption in the instruction decompressor very small. Access time for the decompressor is also much smaller than the access time for the main program memory.

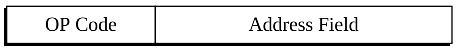

Figure 5: CISC instruction

As shown in Fig. 5, the CISC instruction consists of an operation code field and an address field. The original object codes are read out from this address of the decompressor.

Our optimization technique targets a system as shown in Fig. 6. In this system, an instruction is fetched from main program memory, at first. When the CISC instruction is fetched, main program memory is powered-down until a branch instruction is read out from the decompressor. If branch instructions are read out from the decompressor, the decompressor is powered-down and main program memory is powered-up. The instruction next to the branch instruction is always fetched from main program memory. Detailed discussion of interruption processing is our future work.

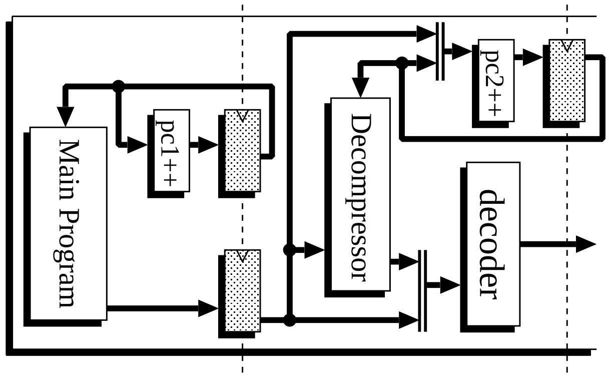

Figure 6: Memory architecture for power optimization

3.3 Problem Formulation

In this section, we present a problem formulation for the *memory optimization problem*. Firstly, we give notations used in the formulation. Next, we present a problem formulation as a 0-1 integer programming problem.

- N : The number of basic blocks appeared in a given program.
- B : A set of basic blocks
- b_i : The ith basic block. $b_i = (S_i, X_i, a_i) \in B$
- S_i : The number of instructions included in ith basic block.

- X_i : The execution count of ith basic block.
- $E(S)$: The average of read energy consumption in ROM module whose size is S bit.
- a_i : 0-1 integer variables associated with b_i. $a_i = 1$ if b_i is merged into the **CISC instruction**; otherwise, $a_i = 0$.
- $Main$: The average of read energy consumption of the main program memory.
- Dec : The average of read energy consumption of the decompressor.

$$\begin{aligned} OBJ = Main \cdot \sum_{i=0}^{N} \{S_i \cdot X_i \cdot (1 - a_i)\} \\ + Main \cdot \sum_{i=0}^{N} (X_i \cdot a_i) \\ + Dec \cdot \sum_{i=0}^{N} (S_i \cdot X_i \cdot a_i) \end{aligned} \quad (2)$$

$$Main = E\left(\sum_{i=0}^{N} \{S_i \cdot (1 - a_i)\} + \sum_{i=0}^{N} a_i\right) \quad (3)$$

$$Dec = E\left(\sum_{i=0}^{N} (S_i \cdot a_i)\right) \quad (4)$$

The *memory optimization problem* is formally defined as follows. **" For a given set of basic blocks B, find a set of a_i which minimize OBJ"**.

3.4 Algorithm

The worst case computation time to solve the *memory optimization problem* is $O(2^N)$, where N represents the number of basic blocks appeared in a program. Therefore, if a naive algorithm is applied, the problem can not be solved within feasible time. Experimental results shown in the succeeded section are derived by the greedy algorithm shown in Fig. 7. The complexity of this heuristic algorithm is $O(N^2)$.

The inputs to algorithm *Memory optimization* are a set of basic blocks B, where each basic block $b_i \in B$ is characterized by its execution count X_i, its size S_i, and its location a_i. All the a_i is set to zero at first step. This means that all the basic blocks are allocated in the main program memory. Next, the algorithm select a basic block, and relocate the basic block from main program memory to decompressor. After calculating $Cost = OBJ$, the provisionally located basic block is moved back to former location. This process is operated for each basic block. After that, the algorithm selects a basic block which minimize the *Cost*, and the selected basic block is merged into the CISC instruction. Basic blocks located in main program memory is successively merged into the CISC instruction in this manner while the *Cost* is reduced. If the *Cost* becomes not to be improved, the algorithm stops after outputting the updated set of basic block B.

```
Given: a set of basic blocks B, where each basic block
$b_i \in B$ is characterized by its execution count $X_i$,
its size $S_i$, and its location $a_i$.
Algorithm Memory optimization
  for each $b_i$
    $a_i$=0;
  end for
  $E_{min} = \infty$;
  while (the algorithm leads to reduction in $E_{min}$)
    for ($i = 0 \ldots N-1$)
      if ($a_i = 0$)
        $b_i = (S_i, X_i, 1)$ ;
        $Cost$ = OBJ ;
        if ($Cost < E_{min}$)
          $E_{min} = Cost$ ;      m = i ;
        end if
        $b_i = (S_i, X_i, 0)$ ;
      end if
    end for
    $b_m = (S_m, X_m, 1)$ ;
  end while
  Output a set of basic blocks $B$ ;
end Algorithm
```

Figure 7: Algorithm for the memory optimization

4 Experimental Results

We use five benchmark programs shown in Table 1, in this experiments. The benchmark programs are compiled by gcc-dlx compiler which is based on GNU CC Ver. 2.7.2 for DLX architecture[10]. Table 2 shows description of three kinds of sample video images used as input to the MPEG2 program. Three kinds of sample input data were also used for Arithmetic calculator, TV remote controller, Espresso : a boolean function optimizer, and Fast Fourier Translator, respectively.

The following two object code merging techniques are evaluated in this section.

- A basic block packing approach

  This approach packs a frequently executed basic

Table 1: Description of benchmark programs

Benchmark	The number of basic blocks
Arithmetic Calculator	2,103
TV Remote Controller	2,994
Espresso	11,023
Fast Fourier Translator	2,875
MPEG2 Decoder	5,361

Table 2: Description of sample data

Data	The number of frames	The size of frames
Image1	50	416x386
Image2	26	352x224
Image3	14	704x480

block into the CISC instruction. Detailed discussion will be done in succeeding subsection.

- A sequence merging approach

  This approach merges together a few basic blocks into the CISC instruction. Detailed discussion will be done in the subsection 4.2.

4.1 Power Optimization with Basic Block Packing Approach

We have evaluated our memory optimization technique by the following way.

1. Measure the number of execution count of each basic block with a sample data.
2. Determine which basic blocks are merged into the CISC instructions. Consequently, physical size of the two memories and allocation of basic blocks to the memories are decided.
3. Measure the total read energy consumption for three kinds of input data with the previously optimized memory organization.

Table 3: The size of memory optimized with the basic-block packing technique

Arithmetic Calculator			
Optimized for	Data1-1	Data1-2	Data1-3
Decompressor	271	308	272
Main program	10,682	10,645	10,679
TV Remote Controller			
Optimized for	Data2-1	Data2-2	Data2-3
Decompressor	1,067	357	1,083
Main program	14,151	14,777	14,135
Espresso			
Optimized for	Data3-1	Data3-2	Data3-3
Decompressor	1,461	1,884	2,101
Main program	60,877	60,550	60,342
Fast Fourier Translator			
Optimized for	Data4-1	Data4-2	Data4-3
Decompressor	1,630	1,483	1,562
Main program	13,701	13,842	13,764
MPEG2 decoder			
Optimized for	Image1	Image2	Image3
Decompressor	1,134	1,212	988
Main program	28,407	28,338	28,542

The optimized memory size are shown in Table 3. Each value represents the number of words of two instruction memories which are optimized for each input data. The results indicate that the size of the decompressor are from 2% to 4% of main program memory for benchmark programs, except TV remote controller and FFT(Fast Fourier Translator). It is a key point of our memory optimization technique that the size of decompressor is restrained very small. This leads to a drastic reduction in average of read energy consumption. In addition, the access time for the decompressor is also much smaller than that of the main program memory, as is shown in Fig. 8. However, in FFT, the size of the decompressor is more than 10% of the main program memory. This result in a low energy reduction rate in FFT.

The energy reduction rates are shown in Table 4. All the values appeared in Table 4 are calculated by using (5).

$$\frac{Energy\ of optimized\ memory}{Apploximated\ energy\ accouding\ to\ (1)} \times 100(\%) \qquad (5)$$

Table 4: The energy consumption of memory with the basic-block packing technique

Arithmetic Calculator			
Optimized for / Executed data	Data1-1	Data1-2	Data1-3
Data1-1	**58.45%**	59.70%	58.82%
Data1-2	60.27%	**59.72%**	60.08%
Data1-3	59.78%	59.96%	**59.69%**
TV Remote Controller			
Optimized for / Executed data	Data2-1	Data2-2	Data2-3
Data2-1	**64.84%**	67.51%	66.39%
Data2-2	63.29%	**61.57%**	62.94%
Data2-3	64.32%	66.93%	**65.83%**
Espresso			
Optimized for / Executed data	Data3-1	Data3-2	Data3-3
Data3-1	**45.55%**	57.06%	64.32%
Data3-2	54.14%	**48.14%**	52.68%
Data3-3	61.33%	51.76%	**49.18%**
Fast Fourier Translator			
Optimized for / Executed data	Data4-1	Data4-2	Data4-3
Data4-1	**65.85%**	66.12%	65.99%
Data4-2	64.83%	**64.70%**	64.71%
Data4-3	64.97%	64.89%	**64.88%**
MPEG2 decoder			
Optimized for / Executed data	Image1	Image2	Image3
Image1	**45.82%**	46.11%	46.48%
Image2	46.81%	**46.46%**	48.80%
Image3	46.42%	46.23%	**45.76%**

The second column of Table 4 represents the input data which are used for memory optimization. The leftmost row of Table 4 represents the input data which are used for evaluation of energy consumption. We have used the divided bit and word lines structure as memory models in this experiment. Therefore, read energy consumption is approximately proportional to the square root of memory size. The results show that energy reductions strongly depend on the kinds of program, but weakly depends on input data. The results show that the energy can always be the smallest when and only when the data used for the optimizations and that for the evaluations are the same from each other. An optimized memory organization with certain input data can be almost optimal for the other input data. Therefore, we can regard that the heuristic algorithm described in Fig. 7 can find good solutions which are very close to the optimal solutions.

Approximated access time in ROMs are shown in Fig. 8. A target process technology is 1.2 micron CMOS. A solid line represents the measured access time of actual 32 bits ROMs ranging from 64 words to 4096 words. A broken line is approximated from the measured values. The access time includes the time for a precharge and evaluation. This figure demonstrates that the access time of the decompressor is almost half of that of the main program memory. For example in the Espresso, the access time of the decompressor is less than one third of that of the main program memory.

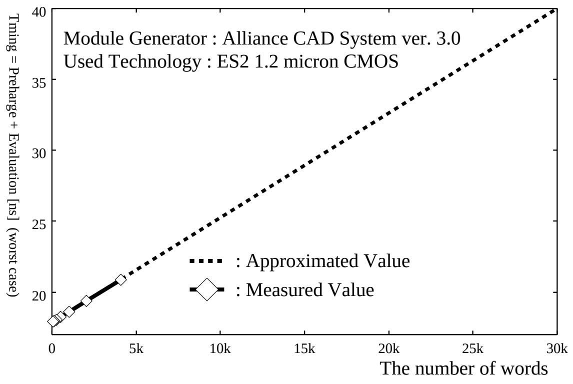

Figure 8: Approximated access time in ROMs

4.2 Power Optimization with a Sequence Merging Approach

The issue for the basic block packing approach is an increase of the energy dissipation for reading the CISC instruction from the main program memory. The CISC instructions are frequently read out from main program memory and the energy for reading the CISC instruction can not be neglected. Therefore, merging a sequence of few basic blocks into the single CISC instruction as shown in Fig.9 is more effective. The basic idea is to merge more than one basic block, which is on the frequently executed loops, into the CISC instruction. Of course, this technique needs some more jump instructions to return to the main program from the basic blocks allocated in the decompressor. These jump instructions cause a performance overhead. Evaluation of the worst case execution time considering these overheads is our future work. We applied this sequence merging technique to the benchmark programs. The environment for the experiments are the same as the experiment of the

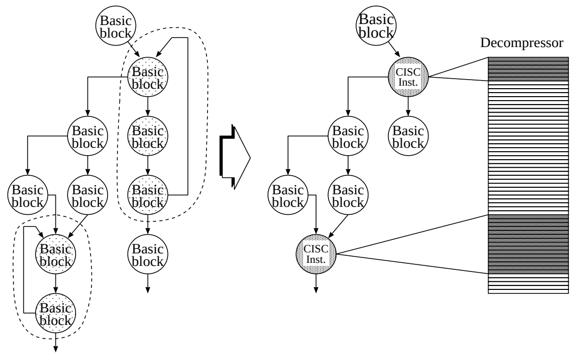

Figure 9: An example of the sequence merging approach

previous subsection. The optimized memory sizes and energy consumptions of memories with the sequence merging are shown in Table 5 and 6, respectively. We can see the tables 5 and 6 in the same manner as the tables 3 and 4, respectively.

The size of decompressor and main program memory shown in Table 5 are comparative with the size of memories which are applied the basic block packing approach. Therefore, access time of memories are almost same from the memories which are optimized by the basic block packing approach. From 7% to 10% improvements in energy consumption are achieved by the sequence merging technique over the basic block packing approach. This is because the number of executed CISC instructions are drastically reduced. In Espresso, the energy consumption is reduced by 68%.

Table 5: The size of memory optimized with the sequence merging technique

Arithmetic Calculator			
Optimized for	Data1-1	Data1-2	Data1-3
Decompressor	312	296	327
Main program	10,620	10,626	10,596
TV Remote Controller			
Optimized for	Data2-1	Data2-2	Data2-3
Decompressor	1,534	401	1,504
Main program	13,827	14,935	13,863
Espresso			
Optimized for	Data3-1	Data3-2	Data3-3
Decompressor	1,654	2,075	2,353
Main program	60,502	60,081	59,803
Fast Fourier Translator			
Optimized for	Data4-1	Data4-2	Data4-3
Decompressor	1,752	1,677	1,677
Main program	13,439	13,514	13,514
MPEG2 decoder			
Optimized for	Image1	Image2	Image3
Decompressor	1,027	1,171	1350
Main program	28,483	28,337	28,165

Table 6: The energy consumption of memory with the sequence merging technique

Arithmetic Calculator			
Optimized for / Executed data	Data1-1	Data1-2	Data1-3
Data1-1	**47.00%**	49.95%	47.65%
Data1-2	50.39%	**47.86%**	48.71%
Data1-3	49.23%	49.64%	**48.66%**
TV Remote Controller			
Optimized for / Executed data	Data2-1	Data2-2	Data2-3
Data2-1	**53.35%**	60.71%	52.98%
Data2-2	52.22%	**51.56%**	52.05%
Data2-3	52.20%	59.75%	**51.97%**
Espresso			
Optimized for / Executed data	Data3-1	Data3-2	Data3-3
Data3-1	**31.65%**	45.61%	54.19%
Data3-2	41.39%	**34.77%**	40.28%
Data3-3	47.16%	38.26%	**35.16%**
TV Remote Controller			
Optimized for / Executed data	Data4-1	Data4-2	Data4-3
Data4-1	**53.35%**	60.71%	52.98%
Data4-2	52.22%	**51.56%**	52.05%
Data4-3	52.20%	59.75%	**51.97%**
MPEG2 decoder			
Optimized for / Executed data	Image1	Image2	Image3
Image1	**40.70%**	41.32%	42.45%
Image2	41.41%	**40.35%**	42.14%
Image3	48.67%	48.15%	**42.25%**

5 Conclusion

In this paper, we have proposed an application specific power optimization technique for embedded systems utilizing the object code merging. We also have presented a 0-1 integer programming problem for the *memory optimization problem.*

Experimental results demonstrated the following. i) The energy consumption in the memories optimized with the basic-block packing technique can be less than 50% of energy in memories which are not applied the code merging, ii) 7-10% improvements in energy consumption are achieved by the sequence merging technique over the basic-block packing approach, iii) The energy reductions strongly depend on the kinds of program, but weakly depends on the kinds of input data. Therefore, optimizing memory organizations with appropriate input data can be almost optimal memory organizations for the other input data, iv) The access time of the decompressor is almost half of that of the main program memory.

Our future work will be devoted to extend the proposed optimization technique considering the data memory.

Acknowledgments

This work has been supported by Semiconductor Technology Academic Research Center (STARC) grant No. PJ-No.987, and the Grant in Aid for Scientific Research of the Ministry of Education, Science, Sports and Culture. We are grateful for their support.

References

[1] Nikolaos Bellas, and Ibrahim Hajj,. "Architectural and Compiler Support for Energy Reduction in the Memory Hierarchy of High Performance Microprocessors". In *Proc. of International Symposium on Low Power Electronics and Design*, pages 70–75, 1998.

[2] C.-L. Su and A. M. Despain. "Cache Design Tradeoffs for Power and Performance Optimization: A Case Study". In *Int'l Symp. on Low Power Design(ISLPD'95)*, pages 282–286, 1995.

[3] Y. Yoshida, B. Y. Song, H. Okuhata, T. Onoye, and I. Shirakawa . "An Object Code Compression Approach to Embedded Processors". In *Proc. of Int'l Symposium on Low Power Electronics and Design*, pages 265–268, Aug. 1997.

[4] K. Ogawa. "PASTEL: A Parametrized Memory Characterization System". In *Proc. of Design, Automation and Test in Europe*, March 1998.

[5] H. Shinohara T. Yoshihara H. Takagi S. Nagao S. Kayano M. Yoshimoto, K. Anami and T. Nakano. "A Divided Word-Line Structure in the Staticture in the Static RAM and its Application to a 64K Full CMOS RAM". *IEEE Journal of Solid-State Circuits*, pages 479–485, June 1983.

[6] M. Isobe, J. Matsunaga, T. Sakurai, T. Ohtani, K. Sawada, H. Nozawa, T. Iszuka and S. Kohyama. "A Low Power 46ns 256K bit CMOS Static RAM with Dynamic Double Word Line". *IEEE Journal of Solid State Circuits*, SC-19(5):578–585, May 1984.

[7] Edwin de Angel and Jr. Earl E. Swartslander. "Survey of Low Power Techniques for ROMs". In *Proc. of Int'l Symposium on Low Power Electronics and Design*, pages 7–11, 1997.

[8] T. Sakurai, and T. Iizuka. "Double Word Line and Bit Line Structure for VLSI RAMs –Reduction of Word Line and Bit Line Delay –". In *Extended Abstracts of the 15th Conf. on Solid State Devices and Materials*, pages 269–272, 1983.

[9] L. Benini, A. Macii, E. Macii, and M. Pancino. "Selective Instruction Compression for Memory Energy Reduction in Embedded System". In *Proc. of Int'l Symposium on Low Power Electronics and Design*, pages 206–211, Aug. 1999.

[10] J. L. Hennessy and D. A. Patterson. *"Computer Architecture: A Quantitative Approach"*. Morgan Kaufmann Publishers, Inc., 2nd edition, 1996.