Stage-Skip Pipeline: A Low Power Processor Architecture Using a Decoded Instruction Buffer

Mitsuru Hiraki, Raminder S. Bajwa*, Hirotsugu Kojima*, Douglas J. Gorny*,
Ken-ichi Nitta**, Avadhani Shridhar*, Katsuro Sasaki*, and Koichi Seki

Central Research Laboratory, Hitachi, Ltd., Kokubunji, Tokyo 185, Japan
*Semiconductor Research Laboratory, Hitachi America, Ltd., San Jose, CA 95134, USA
**Semiconductor and Integrated Circuits Division, Hitachi, Ltd., Kodaira, Tokyo 187, Japan
E-mail: hiraki@crl.hitachi.co.jp, rbajwa@hmsi.com

Abstract

This paper presents a new pipeline structure that dramatically reduces the power consumption of multimedia processors by using the commonly observed characteristic that most of the execution cycles of signal processing programs are used for loop executions. In our pipeline, the signals obtained by decoding the instructions included in a loop are temporarily stored in a small-capacity RAM that we call decoded instruction buffer (DIB), and are reused at every cycle of the loop iterations. The power saving is achieved by stopping the instruction fetch and decode stages of the processor during the loop execution except its first iteration. The result of our power analysis shows that about 40% power saving can be achieved when our pipeline structure is incorporated into a digital signal processor or RISC processor. The area of the DIB is estimated to be about 0.7mm^2 assuming triple-metal 0.5μm CMOS technology.

I. Introduction

Multimedia capabilities are being incorporated into many kinds of computer systems, such as personal computers, personal digital assistants, digital cellular phones, car navigation systems, and video games. To handle these computationally intensive applications, the required LSI performance is increasing greatly, and this is a major factor in the increased power consumption of LSIs. This is especially true for processors because more and more multimedia applications are being handled by software as the performance of various kinds of processors improves drastically [1], [2].

Thus, it is evident that low power design methodologies directed toward multimedia processors are desired. So far, low power design research has been made extensively in various areas, including digital architecture, circuit design, process/device technology, and CAD. Although many of the research results for the latter three can be directly applied to multimedia processors, those for digital architecture are of little use in this area. This is because most of the recent research into low power architectures [3]-[5] assumed a dedicated LSI designed for a specific purpose rather than a programmable processor. We have thus focused on a low power architecture for multimedia processors whose functions can be changed by programming.

In this paper, we present a new pipeline structure that dramatically reduces the power consumption of multimedia processors by using characteristics commonly observed in signal processing programs. In Section II, the power wasted in conventional pipeline structures is briefly discussed to explain our motive for this research. The concept of our low power pipeline structure is presented in Section III. Section IV presents application examples of our pipeline structure to demonstrate its power reduction capability. Some variants of our pipeline structure are discussed in Section V. Section VI summarizes the key points of this paper.

II. Problem of Conventional Pipelines

The majority of multimedia applications use digital signal processing. In processors handling signal processing, most of the execution cycles are used for executing loops because the typical core routines of these programs perform multiply-accumulate operations (given by (1) for example),

$$c = \sum_{n=1}^{N} x(n)y(n) \qquad (1)$$

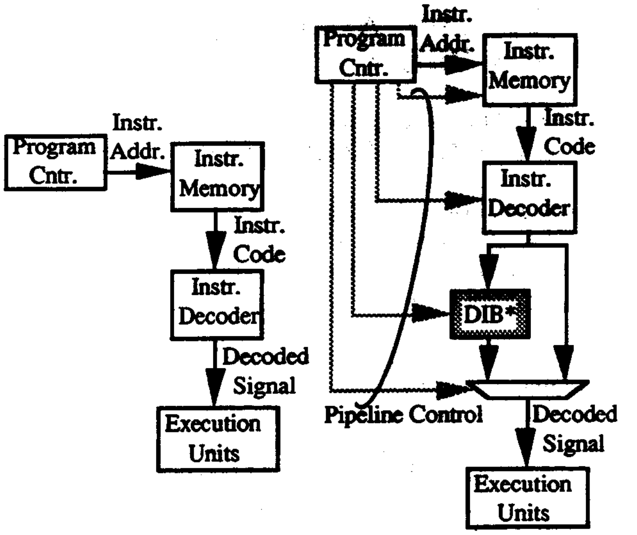
and these operations are accomplished by iterating multiplication and addition many times (N times in the case of (1)).

Processors are generally pipelined to increase throughput. Fig. 1 (a) illustrates the concept of a conventional pipeline. The pipeline is conceptually divided into the following three parts.

1) Instruction memory (or instruction fetch part): provides instruction code stored at the instruction address to which program controller points.
2) Instruction decoder: decodes the instruction code and provides the decoded signal.
3) Execution units: execute the instruction according to the decoded signal.

All three parts of a conventional pipeline are sequentially activated to execute any instruction. From the viewpoint of low power design, however, this is not desirable for loop executions, which cover most of the execution cycles of signal processing programs. This is because instruction fetching and instruction decoding waste precious power budget each time the loop is iterated after the first loop because the codes and decoded signals are the same for each iteration of the loop no matter how many times the loop is iterated.

ISLPED 1996 Monterey CA USA
0-7803-3571-8/96/$5.00©1996



(a) Conventional pipeline (b) SS pipeline** (proposed)

*DIB: Decoded Instruction Buffer
**SS: Stage-Skip

Fig. 1. Concept of conventional and proposed pipelines.

III. Concept of the Stage-Skip Pipeline

Our proposed pipeline structure dramatically reduces the power consumption of multimedia processors by using the characteristic that most execution cycles in signal processing programs are used for loop executions. The concept of our pipeline is illustrated in Fig. 1 (b). The following two elements are inserted between the instruction decoder and execution units:

- Decoded Instruction Buffer (DIB): stores decoded signals for a whole loop.
- Selector: selects the output from either the instruction decoder or DIB.

The decoded instruction signals for a loop are temporarily stored in the DIB and are reused in each iteration of the loop. The power wasted in the conventional pipeline is saved in our pipeline by stopping the instruction fetching and decoding for each loop execution. Since power consumption is reduced by virtually skipping the instruction fetching and decoding stages, we call it a stage-skip pipeline (SS pipeline).

The SS pipeline has three types of operations as shown in Fig. 2. One of these operations is dynamically selected depending on the current execution state of the processor, as follows.

i) *Outside a Loop*

When the processor is executing instructions outside a loop, the pipeline operates exactly like a conventional pipeline. The instructions are fetched, decoded, and sent to the execution units. The DIB is inactive.

ii) *1st Iteration of a Loop*

When the processor executes the first iteration of a loop, the decoded signals for each instruction of the loop are directly provided to the execution units, and at the same time, also written into the DIB. Thus, the decoded signals for the entire loop are stored in the buffer in sequence.

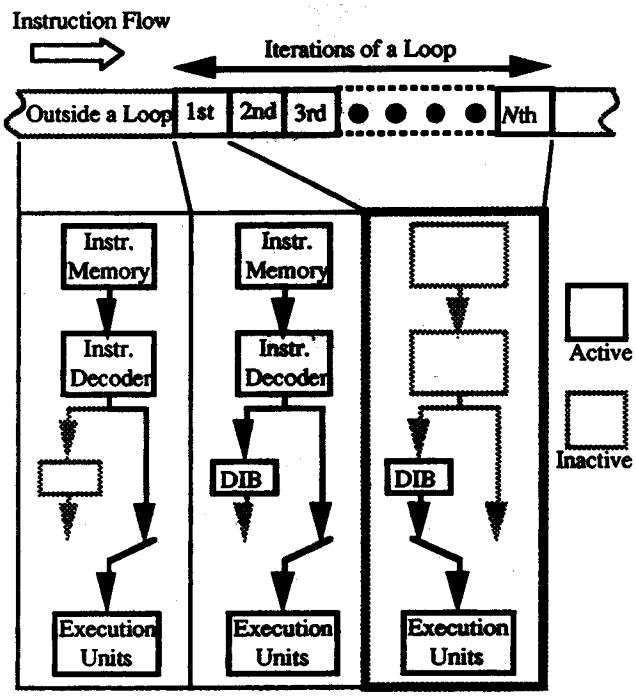


Fig. 2. Operation of SS pipeline.

iii) *2nd-to-Last Iterations of a Loop*

When the processor is executing the 2nd-to-last iterations of a loop, the DIB provides the decoded signals to the execution units in the same sequence as for the first iteration. Since fetching and decoding are not required, power is saved in the SS pipeline.

The pipeline is controlled by a program controller (denoted as Program Cntr. in Fig. 1 (b)), which knows the current execution state as follows. Suppose that the current execution state is *Outside a Loop*, where the program controller increments the instruction addresses one by one. When the processor executes a specific instruction evoking loop execution, the program controller begins to point repeatedly at the addresses for the group of instructions specified in the loop-evoking instruction. When this occurs, the program controller detects that the execution state has changed to *1st Iteration of a Loop*. The number of times for the loop to be iterated is specified in the loop-evoking instruction, and is copied to an iteration count register. Every time the loop is iterated, this register is decremented. The program controller detects that the execution state has changed to *2nd-to-Last Iterations* when the register is decremented for the first time. When the value of the register reaches zero, the program controller detects the end of loop execution and once again begins to increment the instruction addresses one by one. The program controller detects that the execution state has returned to *Outside a Loop* when this occurs.

IV. Application Examples

To demonstrate the power saving capability of the SS pipeline, the power that would be saved by using an SS

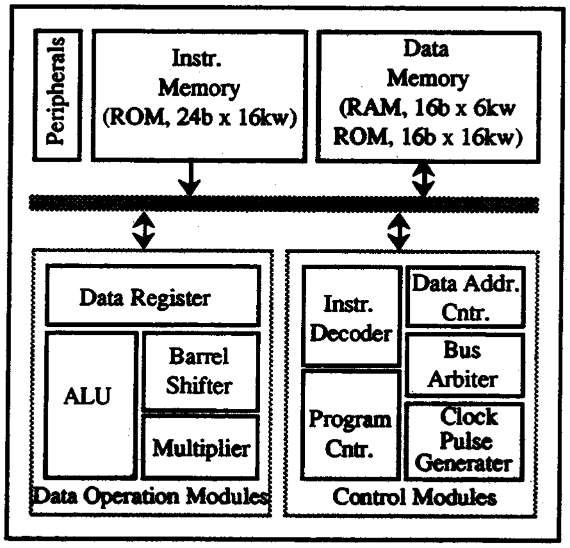


Fig. 3. DSP block diagram.

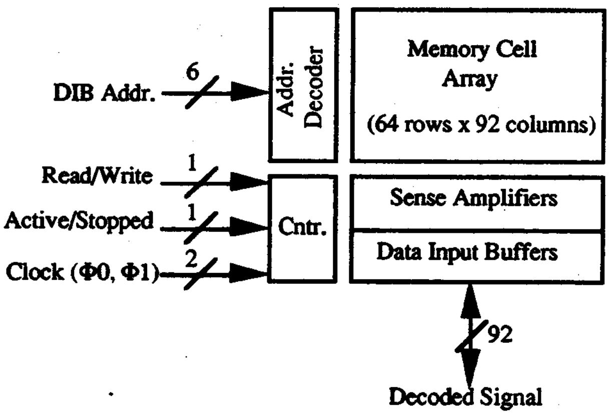


Fig. 4. Block diagram of DIB.

pipeline was analyzed for two processors, one of which is a digital signal processor (DSP) and the other is a RISC processor. First, we will briefly describe these two processors. Then, we will consider the design issues of a DIB for the processors and analyze the behavior of various signal processing programs. Finally, we will present the results of our power analysis for the processors into which an SS pipeline is incorporated.

A. Outline of the Processors

One of the processors that we analyzed is a 24-bit-instruction DSP that we experimentally designed previously. It has an instruction set applicable to a wide range of signal processing applications [6], [7], although its data length was shortened to 16 bits for use as a speech CODEC LSI. A block diagram of the DSP chip is shown in Fig. 3.

The other is a MIPS R3000-like RISC core that was experimentally implemented to perform a power analysis on it [8]. This chip includes integer datapath (register file, ALU, and shifter), next-PC module, controller, I-cache (2kB), and D-cache (2kB).

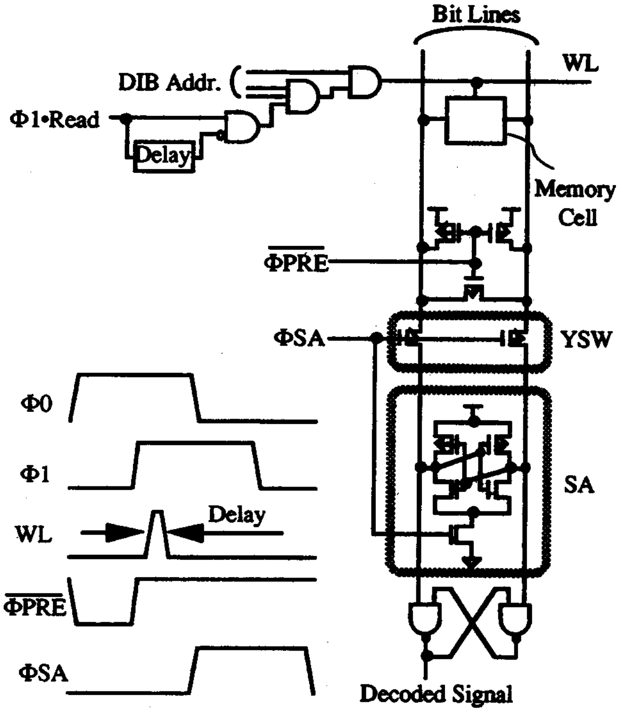


Fig. 5. Circuit diagram of DIB.

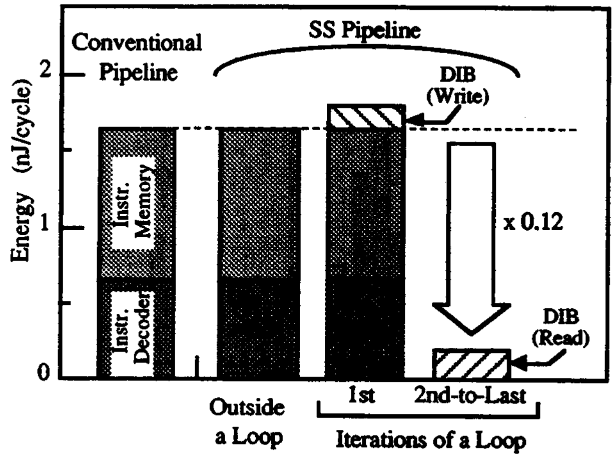


Fig. 6. Energy needed to provide decoded signals.

B. DIB implementation

We studied the design of a DIB, and estimated the area needed for the buffer and the power consumed by the buffer itself. The estimated results are described below.

A block diagram of the DIB for use in the DSP is shown in Fig. 4. To reduce the area needed for the DIB, we use an SRAM structure. The number of columns in the memory cell array (92) corresponds to the number of pins for the decoded signals in the DSP. The number of rows in the memory cell array (64) corresponds to the maximum number of instructions whose decoded signals can be stored in the buffer. The memory cell is composed of six MOS transistors. The area of the DIB is estimated to be about 0.7 mm^2, assuming 0.5µm CMOS technology with a triple-metal process. Note that this corresponds to only 0.8% of the whole chip.

Table 1. Loop size in application programs.

Program	Loop Size (words)
FIR Filter	
single-precision	1
double-precision	3
complex	4
Biquad IIR Filter	
single-precision	4
double-precision	13
LMS Adaptive Filter	
single-precision	2
double-precision	7
complex	8
All Zero Latice Filter	2
All Pole Latice Filter	3
VSELP	1 to 38
LPC coefficient computation	2 to 7
2D 8 x 8 DCT	53

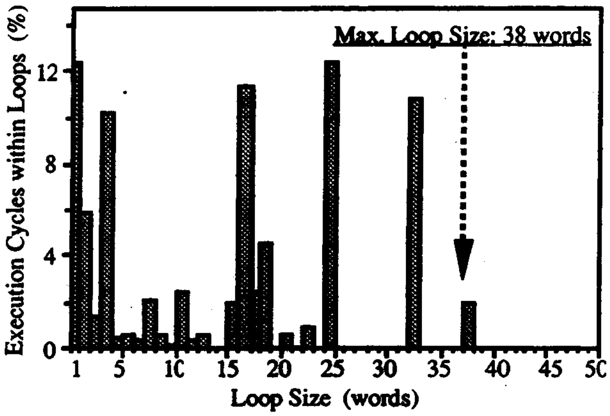


Fig. 7. Loop-size distribution for the case of executing VSELP.

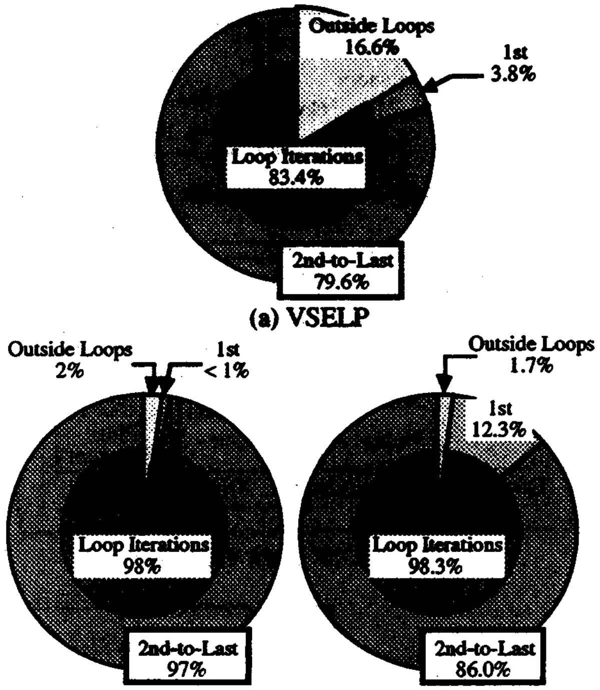
To reduce the power consumed, we used the following power reduction techniques for its read circuits (Fig. 5) [9].

i) To reduce cell current, the word line (WL) is driven by a short pulse.

ii) To reduce power dissipation in the sense amplifiers, latch-type sense amplifiers (SA) are used.

iii) To prevent power dissipation which the latch-type sense amplifiers would cause by driving the bit lines, the bit lines are isolated from the sense amplifiers by the switches (YSW) when the sense amplifiers are activated.

We estimated the power needed for both a conventional pipeline and the SS pipeline to provide decoded signals in the case of the DSP (Fig. 6). Here, SPICE simulation was used to estimate the power consumed in the instruction memory (fetching) and DIB, and the power consumed by the instruction decoder was estimated using power analysis data reported in [10]. Outside a loop, the required power is the same for both. For the 1st iteration of a loop, there is an overhead of about 10% for the SS pipeline to write to the DIB. However, in the 2nd-to-last iterations of a loop, where instruction fetching and decoding are inactive and only the DIB is activated, the required power is reduced to as little as 12% of that of the conventional pipeline. This is because the DIB uses an extremely small-capacity RAM (only 64words x 92bits), so it uses much less power than is used for fetching and decoding.



(a) VSELP

(b) LPC coefficient cpmutation

(c) 2D 8 x 8 DCT

Fig. 8. Execution cycles within and outside loops.

In the case of the R3000-like RISC core, the DIB takes up only about 2% of the die area. The power of the DIB is estimated to be about 11% of the power consumed in the controller and I-cache together. This estimation was done based on the information regarding the schematic, chip size, and power analysis of the RISC core, which are reported in [8].

C. Program Analysis

Since the SS pipeline saves power by using a small-capacity DIB to replace the instruction fetching and decoding during loop execution, the following two points are essential from the viewpoint of program characteristics.

i) For most (preferably all) cases, the number of instructions in a loop (here, it is called loop size) should be small enough for the entire loop to be stored in the DIB.

ii) Most execution cycles should be loop executions.

We verified these points by extensively analyzing signal processing programs coded for the DSP. The analysis results are described below.

Table 1 shows the loop sizes in various signal processing programs. Fig. 7 shows the loop-size distribution when executing vector sum excited linear

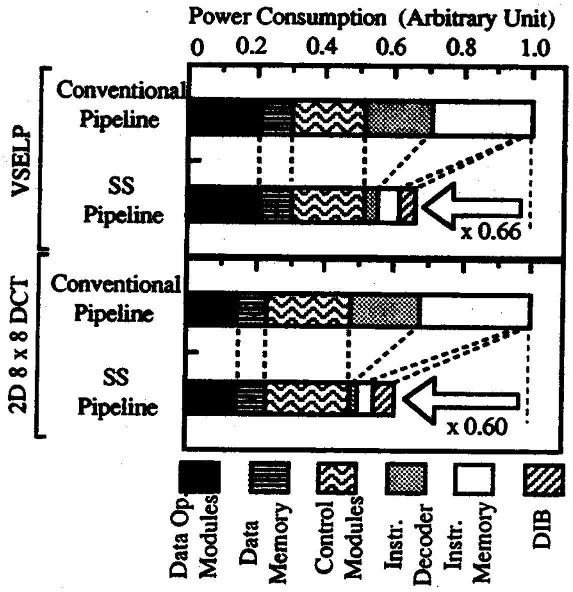


Fig. 9. Improvement in the power consumption of the DSP.

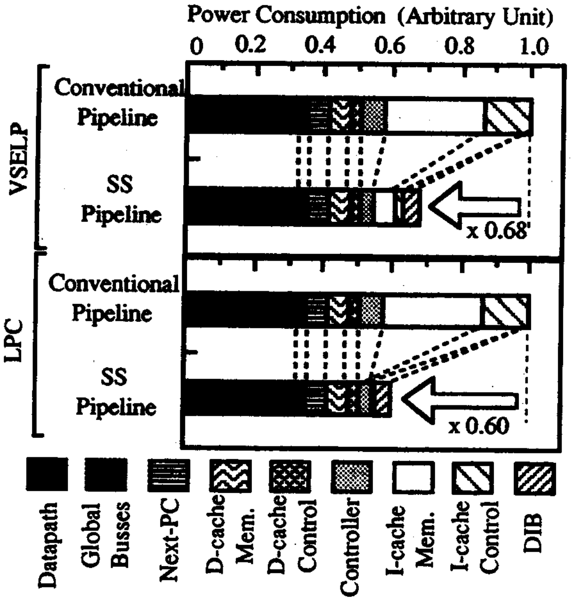
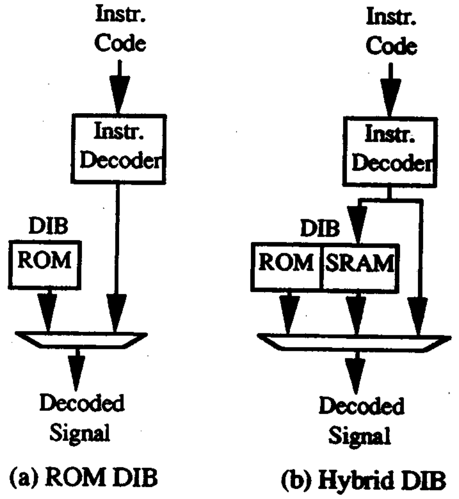


Fig. 10. Improvement in the power consumption of the R3000-like RISC core.

prediction (VSELP) speech CODEC. The largest loop size among the programs that we analyzed was 53 words, which appeared in the 2D 8x8 discrete cosine transform (DCT). The loops in any of these signal processing programs are thus small enough to fit in a 64-word DIB.

Fig. 8 shows the execution cycles within and outside loops when executing VSELP, linear prediction coding (LPC) coefficient computation, and 2D 8x8 DCT. In any of these cases, more than 80% of the execution cycles were used for loop iterations, especially for the 2nd-to-last iterations that are the key to power saving in the SS pipeline.



(a) ROM DIB (b) Hybrid DIB

Fig. 11. Block diagram of the ROM and hybrid DIB.

D. Power Analysis

Based on these studies, we estimated the power that would be saved by incorporating the SS pipeline into the DSP or RISC processor.

Fig. 9 compares the power consumption for the DSP with the SS pipeline and that for a conventional pipeline for executing VSELP and 2D 8x8 DCT (Our analysis process is described in the Appendix). The results are dramatic. The power for the SS pipeline is reduced to between 60% and 66% of that for a conventional pipeline. The power of the instruction memory and instruction decoder in the SS pipeline is greatly reduced since they are stopped during the many cycles used for loop execution. The power overhead for the DIB is so small that it is almost negligible.

Fig. 10 shows the power saving of the R3000-like RISC core with the SS pipeline for executing VSELP and LPC coefficient computation. The results are quite similar to those of the DSP. The power for the SS pipeline is reduced to between 60% and 68% of that for a conventional pipeline.

V. Variants of the DIB

In the case of embedded applications where the DSP processor has to run a small set of applications, cost is a very big constraints and so it is important to keep code size small so that memory (RAM/ROM) requirements are kept down. A variant of the DIB approach can be used to reduce code size as well as power. The numbers in Fig. 8 and inspection of the code indicate that the same code structure is repeated very often. In such cases, it would be possible to isolate these code structures and install decoded versions in an on-chip ROM which can be treated as a library to be used at run time by the application program. ROMs are approximately four times denser than SRAMs and so for the same area overhead larger storage can be used. This approach also presents the opportunity to further reduce the power consumption by reducing traffic to off-chip memories. Finally, a hybrid structure which includes a ROM and SRAM in the DIB can also be used to benefit from both code size reduction and power reduction. Fig. 11

Table 2. Consumed energy and active cycle rate of each module of the DSP.

Module, *i*	Energy, E*i* (pJ/cycle)	Active Cycle Rate, N*i*			
		VSELP		2D 8 x 8 DCT	
		conv.	SS	conv.	SS
Data Op. Modules					
ALU	300	50.25%	←	48.32%	←
Multiplier	349	39.88%	←	25.96%	←
Barrel Shifter	668	8.82%	←	0.00%	←
Data Register					
Data Op. Load/Store					
double double	664	27.05%	←	0.00%	←
double single	538	5.17%	←	1.85%	←
double none	412	3.42%	←	5.56%	←
single double	434	3.11%	←	0.00%	←
single single	308	3.67%	←	25.96%	←
single none	182	22.51%	←	33.37%	←
none double	252	10.52%	←	3.71%	←
none single	126	8.85%	←	25.96%	←
Control Modules					
Data Addr. Cntr.	445	56.71%	←	58.63%	←
Program Cntr.	192	100.00%	←	100.00%	←
Instr. Decoder	634	100.00%	20.36%	100.00%	14.02%
Bus Arbiter	70	100.00%	←	100.00%	←
Clock Pulse Generater	208	100.00%	←	100.00%	←
Data Memory					
RAM (Read)	320	76.89%	←	39.63%	←
RAM (Write)	540	12.85%	←	22.25%	←
ROM	716	0.00%	←	0.00%	←
Instr. Memory	990	100.00%	20.36%	100.00%	14.02%
DIB					
Read	196	0.00%	79.64%	0.00%	85.98%
Write	163	0.00%	3.81%	0.00%	12.28%
Peripherals	669	0.00%	←	0.00%	←

shows the block diagrams for these cases.

These variants would require careful assembly code generation and can benefit from compiler optimization techniques that can generate reusable segments of code for such a library-based approach. Several small loops may be consolidated whenever possible to reduce overheads as long as this can be done without register spilling.

VI. Conclusion

We have proposed a stage-skip pipeline (SS pipeline) to reduce the power consumption of multimedia processors. The SS pipeline saves power by stopping instruction fetching and decoding during loop execution and using a small-capacity RAM that we call decoded instruction buffer (DIB) to take over for them. This greatly reduces the power used by processors handling multimedia applications because the overwhelming majority of execution cycles in signal processing programs are used for loop execution. Our case study showed that power consumption is reduced about 40% when an SS pipeline is incorporated into a digital signal processor or RISC processor. The increase in silicon area due to the DIB is estimated to be about 0.7 mm^2, assuming that triple-metal 0.5μm CMOS technology is used.

Appendix

The overall consumed energy was assumed to be given by (2).

$$E_{Total} = \sum_i E_i \times N_i \qquad (2)$$

where E_{Total} is the overall consumed energy, E_i is the energy consumed per cycle in module *i*, and N_i is the number of active cycles for module *i*. The E_i for each module is listed in Table 2, where the energy values for the on-chip memories and DIB were extracted using SPICE and those for the rest of the modules were extracted using a gate-level power analysis tool [10]. The operating voltage is 3.3 V. The N_i for each module is also listed in Table 2 for the execution of VSELP and 2D 8x8 DCT using the DSP with a conventional pipeline and one with an SS pipeline. The N_i for the conventional pipeline can be obtained by simply counting the number of times each instruction is used because the activated modules for each cycle can be specified from the instruction being executed in that cycle. The N_i for the SS pipeline can be obtained by converting the results based on the loop execution statistics shown in Fig. 8.

Acknowledgments

We are indebted to E. Takeda for his support and encouragement, and to S. Bhattacharyya, Y. Hatano, T. Nakagawa, and A. Kiuchi for their comments, suggestions, and overall assistance in preparation of this paper.

References

[1] H. Sasaki, "Multimedia Complex on a Chip," in *ISSCC Dig. Tech. Papers*, pp. 16-19, Feb. 1996.

[2] D. Epstein, "Chromatic Raises the Multimedia Bar," *Microprocessor Report*, pp. 23-27, Oct. 23, 1995.

[3] A. Chandrakasan, A. Burstein, and R. W. Broderson, "A Low Power Chipset for Portable Multimedia Applications," in *ISSCC Dig. Tech. Papers*, pp. 82-83, Feb. 1994.

[4] D. B. Lidsky and J. M. Rabaey, "Low-Power Design of Memory Intensive Functions," in *Symp. Low Power Electronics, Dig. Tech. Papers*, pp. 16-17, Oct. 1994.

[5] B. M. Gordon, T. H. Meng, and N. Chaddha, "A 1.2mW Video-Rate 2D Color Subband Decoder," in *ISSCC Dig. Tech. Papers*, pp. 290-291, Feb. 1995.

[6] T. Baji, et al., "HX24 24-bit Fixed Point Digital Signal Processor," in *Proc. ICSPAT*, pp. 622-629, Oct. 1993.

[7] A. Kiuchi, T. Nakagawa, T. Baji, and K. Kaneko, "Digital Signal Processor Supporting Two Types of Instruction Sets," *Electronics and Communications in Japan*, Part 3, Vol. 78, No. 6, pp. 20-29, 1995.

[8] T. Burd and B. Peters, "A Power Analysis of a Microprocessor: A Study of an Implementation of the MIPS R3000 Architecture," Technical Report ERL, University of California, Berkeley, May 1994. http://infopad.eecs.berkeley.edu:80/~burd/gpp/r3000/total.html.

[9] Y. Shimazaki, et al., "An Automatic-Power-Save Cache Memory for Low-Power RISC Processors," in *Symp. Low Power Electronics Dig. Tech. Papers*, pp. 58-59, Oct. 1995.

[10] H. Kojima, D. J. Gorny, K. Nitta, and K. Sasaki, "Power Analysis of a Programmable DSP for Architecture /Program Optimization," in *Symp. Low Power Electronics Dig. Tech. Papers*, pp. 26-27, Oct. 1995.