Regular Paper

Low-Energy Design Using Datapath Width Optimization for Embedded Processor-Based Systems

Yun Cao† and Hiroto Yasuura†

This paper presents a novel system-level technique that minimizes the energy consumption of embedded processor-based systems through datapath width optimization. It is based on the idea of minimizing energy consumed by redundant bits, which are unused during execution of programs by means of optimizing the datapath width of processors. To minimize the redundant bits of variables in a given application program, the effective size of each variable is determined by variable size analysis, and Valen-C language is used to preserve the precision of computation. Analysis results of variables show that there are average 39% redundant bits in the C source program of MPEG2 video decoder. In our experiments for several real embedded applications, energy savings without performance penalty are reported and range from about 10.8% to 48.3%.

1. Introduction

Minimizing power consumption of embedded systems is a crucial task. Battery-operated portable systems demand tight constraints on energy consumption. Better low-power circuit design techniques and advances in battery technology have helped to increase battery lifetime. On the other hand, managing power dissipation at higher design levels can considerably reduce energy consumption, and thus increase battery lifetime. Energy consumption at all design levels should be considered to reduce power of the whole embedded system.

We have developed a design platform, which consists of a Valen-C retargetable compiler [2], soft-core processor (Bung-DLX)[1),3)] and a cycle-based simulator [4]. We also have done some researches on reduction of area and cost for embedded core-based systems [5),6)]. In this paper, we focus on minimizing energy dissipation and present a system-level technique for embedded processor-based systems, which minimizes energy consumption of the whole system while providing adequate performance level. In the initial design phase of our approach, we design a system with a soft-core processor, data RAMs, instruction ROMs and logic circuits. Then we analyze the effective bit width of each variable of a given application program. After that, using the results of analysis, we rewrite the application program in Valen-C language [2], in which we specify the word length of each variable satisfying accurate computation to reduce energy consumed by redundant bits in the application program. After verifying the functionality of the initial design, we modify several design parameters of the soft-core processor, including the datapath width, the number of registers and the instruction set. We can tune up the soft-core processor to minimize the energy consumption while satisfying the system performance constraints. To get first-cut estimates of energy consumption early in the design, a few component-based power estimation models are also developed, total energy is obtained by summing over all components of the system.

This paper is structured as follows: the next Section 2 gives an overview of related work. Section 3 describes our energy minimization technique by datapath width optimization. Section 4 presents our energy estimation models. Experiments and results are shown in Section 5. Finally, Section 6 concludes our work.

2. Related Work

Hardware and software techniques to reduce energy consumption have become an essential part of current system designs. Extensive researches on power optimization from circuit level to system level have been conducted in these recent years. Such techniques have particularly targeted the memory system [7)~9)] due to the prevalent use of data-dominated signal and video applications. Reference 14) focuses on exploiting cache to reduce power consumption. The work [11)] presented an architecture-oriented power minimization approach. A power and performance simulation tool that can be used to do architecture-level optimizations has been

† Kyushu University

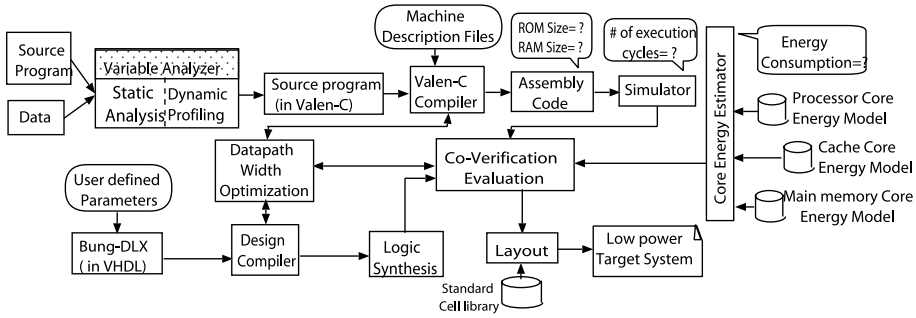


Fig. 1 An energy minimization flow using datapath width optimization.

introduced by Sato, et al. [12]. The approach [13] uses a multiple-voltage power supply to minimize system-power consumption. A framework for describing the power behavior of system-level designs was proposed by Ref. 10). The paper Ref. 15) proposed a low power hardware/software partitioning approach using a high utilization rate of the involved resources.

As far as we know, this paper first present a system-level energy minimization approach in which designers can control the width of datapath freely. The energy consumption of the whole system is drastically reduced without decline of performance by optimizing the datapath width.

3. An Energy Minimization Approach

In the design of consumer electronic systems, designers have to manage rapid increase of complexity of a target system with requirments on high-performance and low-energy comsumption under the tight constrait of short design time. Therefore, core-based solutions are proposed for embedded system design. Our approach gives designers a freedom to determine the datapath width of soft-core processor, because the datapath width of a processor has great impacts not only on power consumption and performance of the processor but also on those of memories. Optimizing datapath width for each given application is an effective approach to minimize energy consumption of the whole embedded systems.

3.1 Problem Description

We focus on reducing the energy consumption of an embedded system by tuning the datapath width of processors. We define an embedded system as a processor along with its data memory and instruction memory, all implemented on a single chip (SOC). The soft-core processor (Bung-DLX) that we used in this paper is a variable configuration RISC processor that can be synthesized to any bit-width and register file size. The instruction memory is implemented in ROM and the data memory is implemented in RAM that has a word length equal to the width of the processor datapath. The instruction word length remains invariant in this paper.

Valen-C retargetable compiler is produced by a multi-precision compiler generator that automatically emits the necessary multi-precision instructions, according to the processor datapath and the bit width specification of each variable.

The energy minimization problem is formulated as:

$$\begin{array}{ll} \mathit{minimize} & \mathit{Energy}(w) \\ \mathit{subject\ to} & \mathit{Cycle}(w) \le C_{cst} \\ & \mathit{Area}(w) \le A_{cst} \end{array}$$

Where $\mathit{Energy}(w)$, $\mathit{Cycle}(w)$ and $\mathit{Area}(w)$ are functions of the datapath width w, C_{cst} and A_{cst} are the constraints on the execution cycle and area respectively. This is a nonlinear optimization problem. The overview of our energy minimization algorithm is described in **Fig. 2**.

3.2 Our Approach

Figure 1 shows our proposed approach, which consists of the following phases:

- Phase 1: The source program of the target application, which was originally written in C or other language, is rewritten in Valen-C language, after the bit width of each variable is analyzed. For instance, if the variable x requires at most 11 bits, the programmer can write $int11\ x$; in the variable declaration of Valen-C program.
- Phase 2: Bung-DLX is customized to different soft-core processors by choosing different design parameters, such as the datapath width and the address size of the data

- Input:
  - source program: S_j
  - input data: D_{in}
  - the constraint of cycles: C_{cst}
- Variable:
  - datapath width $w_i \in W = \{w_1, w_2, \ldots w_n\}$
- Output:
  - execution cycles $c_i \in C = \{c_1, c_2, \ldots c_n\}$
  - the minimal energy consumption E_{min} when $c_k \leq C_{cst}$
  - the datapath width w_k when $E_k = E_{min}$
- Phase 1: Analysis
  - static analysis of variable size $x_i \in X = \{x_1, x_2, \ldots x_n\}$
  - compile the source program S_j
  - dynamic analysis of variable size $y_i \in Y = \{y_1, y_2, \ldots y_n\}$
- Phase 2: Soft-core processor (Bung-DLX)
  - define the design parameters
    - datapath width w_i
    - the number of registers n_i
- Phase 3: Valen-C program VS_j
  - variable declaration of bit width (x_{ie} and y_{ie})
  - compile the Valen-C source program VS_j for customized Bung-DLX at w_i
- Phase 4: Estimation
  - for $w_i \in W$
    - calculate the execution cycles c_i
    - calculate the energy consumption E_i
    - get (E_{min}, w_k) when $c_k \leq C_{cst}$
  - return (E_{min}, w_k)

Fig. 2 Pseudo code of the algorithm for energy minimization.

memory.

- Phase 3: The Valen-C source program of an application is compiled for the customized soft-core processors. Retargetable Valen-C compiler generates the assembly code from the source program. As a result, different embedded systems are generated based on different customized processors and assembly codes. At this phase, the size of both the data memory and the instruction memory of each system are estimated.
- Phase 4: The systems generated at phase 3 are evaluated. Execution cycles, memory size and energy consumption are estimated. The impact of the design parameters on the energy consumption and on the system performance is evaluated, the embedded system of the minimal energy consumption, which satisfies the design constraints, is chosen among those systems.

3.3 Variable Size Analysis

In order to optimize datapath width, the effective size of each variable in an application program needs to be analyzed. This section explains our methods to analyze effective sizes of variables in C programs. In this paper, we define *effective size* as the smallest size which can hold both maximum and minimum values of a variable. In many cases, some bits of a variable are never used during execution of a program. If a variable x of unsigned integer type whose value is in [0, 2000], i.e., between 0 and 2000, then the number of necessary bits of x is 11, because the 11-bit size is large enough to hold any value in [0, 2000].

We use two methods to analyze effective size of variables. One is dynamic analysis, which runs programs and monitors the value of each variable. Dynamic analysis is one kind of simulation-based method whose results depend on input data sets given to the programs. The other is static analysis.

For static analysis, when the maximum value of an unsigned integer variable x is n_{max}, the effective size of x, $e(x)$, is given as follows:

$$e(x) = \log_2(n_{max} + 1) \quad (1)$$

For a signed integer x with a maximum value n_{max} and a minimum value n_{min}, $e(x)$ is defined as follows:

$$e(x) = \lceil \log_2 \mathcal{N} \rceil + 1 \quad (2)$$

where

$$\mathcal{N} = \max(|n_{max}| + 1, |n_{min}|) \quad (3)$$

Static analysis is an efficient method to analyze the effective size of variables. However, in many cases when we can not predict the assigned value of a variable unless we execute the program, such as the case of unbounded loops, static analysis becomes insufficient. As a solution to this problem, we adopted dynamic analysis in our approach.

Figure 3 shows a part of the algorithm used for dynamic analysis. In dynamic analysis, we execute the program S_j with input data D_{in} and monitor the values y_i assigned to each variable n_i. We insert the monitoring function *checkbits* to the assignment statement of variables. The arguments of the monitoring function are the variable name n_i and its assigned value y_i. The monitoring function *checkbits* checks the value assigned to the variable, verifies the bit width required and then memorize it. After that, it keeps the bit width temporarily in a table. When the monitoring function checks the same variable with a different assigned value, it compares the new bit width with the bit width already memorized in the table, and keeps the bigger one in the table and so on. Thus, the required bit width y_{ie} of the

- Initial values:
  - initialize the table for int type variables,
  - initialize the table for short type variables,
  - initialize the index for tables $i = 0$
- Input:
  - the name of variable: $n_i \in N = \{n_1, n_2, \ldots, n_m\}$
  - the value of variable: $y_i \in Y = \{y_1, y_2, \ldots, y_m\}$
- Output:
  - the name of variable: $n_i \in N = \{n_1, n_2, \ldots, n_m\}$
  - the effective value of variable: $y_{ie} \in Y_{ie} = \{y_{1e}, y_{2e}, \ldots, y_{me}\}$
- Algorithm:

$int \quad checkbits(char \star name, int\ y_{ie})\ \{$
$while\ ((strcmp((char \quad \star)name_{int}[i], n_i) \neq 0)$
$and\ (i < Ntable_{int}))\quad \{$
$\quad i = i + 1;\ \}$
$if \quad (i == Ntable_{int}) \quad then \quad \{$
$\quad strcpy((char \quad \star)name_{int}[i], n_i);$
$\quad Ntable_{int} = Ntable_{int} + 1;\ \}$
$bitstable_{int}[i] \mid= y_{ie};$
$return(n_i, y_{ie});\ \}$

Fig. 3 A part of the algorithm for dynamic analysis.

variable n_i is got after executing the program.

3.4 Efficient Use of Data Memory

Since in many cases, high-level specifications are devoted to describe functionalities of target systems rather than implementation details, they often contain a lot of redundancies such as duplicated computations and never executed code. Therefore, the specifications must be optimized to remove the redundancies for energy-efficient design. Some redundancies are introduced in size of variables. For example, in C programs, a variable whose value is between 0 and 1000 is often declared as the *int* type, i.e., usually 16 or 32 bits depending on target processors, and then some upper bits make nonsense. This means that the memory has many unnecessary bits, which do not essentially contribute to the calculation of programs. Therefore redundant bits should be removed to reduce power consumption.

C language provides for three integer sizes, declared using the keywords *short*, *int* and *long*. The compiler designer determines the sizes of these integer types. In many processors, the size of *short* is 16 bits, *int* is 16 or 32 bits, *long* is 32 bits. On the other hand, in Valen-C, programmers explicitly specify the required bit width of each integer data type. Thus it becomes possible to reduce the energy of the datapath and the data memory, which is dissipated by the redundant bits. For instance, if variables x, y, and z require 12, 20 and 24 bits respectively, the programmer can write "int12 x; int20 y; int24 z;" in the variable declaration of Valen-C program. If a processor with a datapath width of 20 bits is used in the system, the total memory size will be 80 bits. Moreover, the unused bits in the data memory will be 24 bits. On the other hand, if a processor of a datapath width of 12 bits is used, the total data memory size will become only 60 bits, and the unused memory size will decrease to 4 bits. As a result, specifying the word length required for each variable and changing the datapath width have a significant role in reducing the data memory size of a system. Therefore it also affects the power consumption of the system.

3.5 Datapath Width Optimization

System designers can tune the value of the datapath width in accordance with the characteristics of target system to deliver most suited processor. Designers can reduce the datapath width until the single precision point (SPP) without performance loss [5]. SPP is the processor datapath width, which is equal to the bit width of the largest variable in a program. It is the smallest datapath width at which all instructions can remain single-precision. Designers may obtain better solutions, more power savings by shrinking the datapath less than SPP, under performance constraints. **Figure 4** shows the overview of our datapath width optimization algorithm.

3.6 Power versus Performance Tradeoff

Minimizing power consumption is not simply an altruistic activity. A device consuming less power will accrue several desirable advantages such as longer battery life for wireless devices, but somewhat less obvious advantages, such as reliability and performance. The datapath width of a processor strongly affects the power consumption of the whole system including the processor, data memories and instruction memories, it also affects the execution cycles of a given task, i.e., narrowing the datapath width less than SPP will cause the increase of execution cycles because of multiple-precision operations. For example, that an addition of 20 bit data is executed by only one instruction on a 20 bit processor is assumed, If the datapath width becomes to 10 bits, two instructions including additions of lower 10 bits and high

- **Input:**
  - assembly code of Valen-C program $VSas_j$
- **Variable:**
  - datapath width: $W_i \in W = \{W_1, W_2, \ldots, W_n\}$
- **Output:**
  - the minimal energy consumption E_{min} when $C_k \leq C_{cst}$
  - the optimal datapath width W_k when $E_k = E_{min}$
  - the cycle C_k when $E_k = E_{min}$
- **Algorithm:** $DaPO(VSas_j, E_{min}, W_k, C_k)$ {

$W_i = W_1$;
while $(W_i \neq W_n + 1)$ {
$GetC(C_i)$;
if $(C_i \leq C_{cst})$ *then* {
$E_{proc} = \sum_{j \in I} e_j \times Cycle_j$;
$e_{ROM} = 50.97 * W_i * \sqrt{N_{words}} + 1.4$;
$e_{Sr} = 24.9 * W_i * \sqrt{N_{words}} + 56$;
$e_{Sw} = 197 * W_i * \sqrt{N_{words}} + 369$;
$E_{ROM} = e_{ROM} \times \sum_{i \in I} Cycle_i$;
$E_{SRAM} = e_{Sr} \times C_{load} + e_{Sw} \times C_{store}$;
$E_{mem} = E_{ROM} + E_{SRAM}$;
$E_i = E_{proc} + E_{mem}$;
$GetMIN(E_i)$;
$GetODC(W_k, C_k)$; }
$W_i = W_i + 1$; }
return(E_{min}, W_k, C_k); }

Fig. 4 Pseudo code of the algorithm for datapath width optimization.

10 bits with carry are required. So trade-offs exist between datapath width and execution cycles. Although a processor with narrower datapath width dissipates lower power per clock cycle, the total energy for the task is not reduced always by narrowing the datapath width. Thus, for a given target system, trading off the power consumption and performance is an important work.

4. Energy Estimation Models

This section describes energy consumption models. The total energy consumption, E, is the summation of energy consumed by the processor (E_{proc}) and memories (E_{mem}).

$$E = E_{proc} + E_{mem} \quad (4)$$

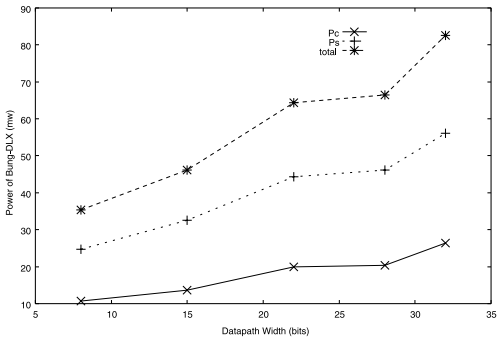
We estimated E_{proc} and E_{mem} separately, and got the energy consumption model of our soft-core processor generated by HITACH 0.5um CMOS technology and the energy consumption models of memory generated by Alliance CAD System Ver.3.0 with 0.5um double metal CMOS technology.

E_{proc} is given by

$$E_{proc} = \sum_{i \in I} e_i \times Cycle_i \quad (5)$$

where

e_i: Average energy of instruction i



Datapath Width (bit)	P_c (mw)	P_s (mw)	P_{total} (mw)	Savings (%)
32	26.39	56.15	82.54	-
28	20.33	46.15	66.48	19.46
22	19.95	44.39	64.34	22.05
15	13.62	32.54	46.16	44.08
8	10.67	24.69	35.36	57.16

Fig. 5 Power of Bung-DLX ($V_{dd} = 3.3$ V).

$Cycle_i$: The number of execution of instruction i

I: Instruction set of Bung-DLX

e_i is obtained by performing post-layout simulation of switch-level. After several simulations, we obtained the empirical energy model at several datapath widths in **Fig. 5**, where power savings are got by comparing to the power consumption of 32 bits Bung-DLX. The power dissipation in static CMOS can be divided into static, dynamic and short-circuit power. Because static power and short-circuit power are far less than dynamic power, we just focus on dynamic power, which consists of Cell Internal Power (P_c) and Net Switching power (P_s).

e_i is shown as follows:

$$e_i = \frac{1}{2} \times V_{dd}^2 \sum_{net} [C_j \times S_j + E_{c_k} \times S_k] \quad (6)$$

where

V_{dd}: Supply voltage

C_j: Load capacitance of net j

S_j: The average number of switching of net j per clock cycle

E_{c_k}: Internal power of cell k

S_k: The average number of switching of cell k per clock cycle

E_{mem} is estimated as follows:

$$E_{mem} = E_{ROM} + E_{SRAM} \quad (7)$$

$$E_{ROM} = e_{ROM} \times \sum_{i \in I} C_i$$

$$E_{SRAM} = e_{Sr} \times C_{load} + e_{Sw} \times C_{store} \quad (8)$$

where

e_{ROM}: Energy per read access to ROM

$e_{Sr}(e_{Sw})$: Energy per read (write) access to SRAM

$C_{load}(C_{store})$: The number of read (write) accesses of SRAM

The access energy of memories (e_{ROM}, e_{Sr}, e_{Sw}) is obtained from the SPICE simulation of several memories with the different configurations. As the result, we have obtained the estimation models as follows:

$$e_{ROM} = 50.97 \cdot b \cdot \sqrt{N_w} + 1.4[pJ/C] \quad (9)$$

$$e_{Sr} = 24.9 \cdot b \cdot \sqrt{N_w} + 56[pJ/C] \quad (10)$$

$$e_{Sw} = 197 \cdot b \cdot \sqrt{N_w} + 369[pJ/C] \quad (11)$$

Where b is the word width of the memory and N_w is the number of words. pJ/C means one pJ per cycle.

5. Experiments and Results

In this section we present experiments and results based on several real applications to evaluate our proposed approach. We mainly illustrate how we use our approach to minimize energy consumption of MPEG2 video decoder, a relatively large program.

In the experiments, we assumed the target system, a SOC chip, which consists a Bung-DLX processor, a ROM and a SRAM. Bung-DLX is a non-pipelined, simple RISC processor, which has several design parameters including the datapath width and the number of registers. All instructions are executed within a single machine cycle. The ROM and the SRAM are used as instruction memory and data memory respectively. These memories are generated by Alliance CAD System Ver. 2.0 with 0.5 μm double metal CMOS technology. For simplicity, we assumed that no other core is integrated in the SOC chip.

5.1 Variable Size Analysis for MPEG2

Our program is based on Mpeg2decode program from the MPEG Software Simulation Group. It is a player for MPEG-1 and MPEG2 video bitstreams. Mpeg2decode is an implementation of an ISO/IEC DIS 13818-2 decoder, whose emphasis is on correct implementation of the MPEG standard and comprehensive code structure. We rewrote it in Valen-C with about 6650 lines. The MPEG2 core consists of several function blocks such as a soft-core processor, IDCT blocks, a couple of motion estimation blocks, a motion compensation block, variable length encoding, decoding blocks and so on.

Table 1 The number and types of the variables (MPEG2 decoder).

types	Num.	types	Num.
int	384	unsigned	35
pointers	101	short	6
char	3	unsigned char	21

Table 2 Static analysis results (MPEG2 decoder).

E.Size	N.of Variables	E.Size	N. of Variables
1 bit	50	12 bits	14
2 bits	17	14 bits	46
3 bits	10	15 bits	2
4 bits	11	16 bits	39
5 bits	8	17 bits	2
6 bits	11	18 bits	2
7 bits	12	26 bits	2
8 bits	9	27 bits	4
9 bits	7	28 bits	3
10 bits	3	29 bits	3
11 bits	6	30 bits	7
Total	5,656 bits	−34%	(8,576 bits)

Table 3 Dynamic analysis results (MPEG2).

V.name	E.size	V.name	E.size
fn	5 bits	g2nc	7 bits
fl	12 bits	rbx	12 bits
sn	6 bits	rby	12 bits
nl	20 bits	rec4s1	24 bits
gb32l	20 bits	rec4s4	24 bits
gbl	20 bits	rec4cs1	24 bits
gbn	5 bits	rec4cs4	24 bits
g2ai	20 bits	rechs1	24 bits
g2asign	20 bits	rechcs1	24 bits
g2aincn	18 bits	rec4as1	24 bits
g2anc	6 bits	rec4as4	24 bits
gi	7 bits	rechas1	24 bits
gsign	20 bits	rechas2	24 bits
gincnt	6 bits	rec4acs1	24 bits
g2i	7 bits	rec4acs2	24 bits
g2sign	3 bits	rechacs1	24 bits
g2incnt	7 bits	rechacs2	24 bits
Total	1,056 bits	521 bits	−52%

We analyzed the C source program of MPEG2 video decoder and got some analysis results. The number and types of the variables in MPEG2 decoder are described in **Table 1**. The results of static variable analysis are depicted in **Table 2** (*E.Size* means effective size of variable; *N. of Variables* means the number of variables), and that of dynamic analysis are shown in **Table 3** (*V.name* means variable name). From Table 2 and Table 3, we can see that there are many redundant bits in the variables of MPEG2 decoder C source program. We got 34% reduction of bits from the static analysis and 52% from the dynamic analysis.

To verify our analysis results of variable size, we used the following model.

D. W	Supply Voltage $V_{dd} = 3.3V$				
	$E_p(J)$	$E_s(J)$	$E_r(J)$	$E_t(J)$	Sav.
32 bits	0.85	85.76	70.55	157.2	-
30 bits	0.78	76.97	70.55	148.3	5.6%
28 bits	0.68	69.01	70.55	140.2	10.8%
26 bits	0.90	124.8	72.14	197.8	−25.8%
22 bits	1.03	108.6	109.8	219.4	−39.6%

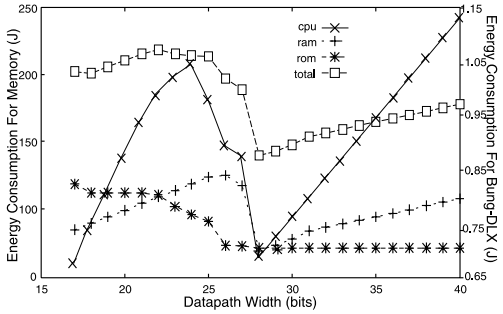


Fig. 6 Energy consumption for MPEG2 decoder.

$$PSNR = 10 \times \log_{10}[\frac{1}{E} \times 255^2][dB] \quad (12)$$

where, $PSNR$: Ratio of pick signal to noise
E: Mean-square error

Our experimental results of $PSNR$ are *infinite*, so it shows that the variables, which are assumed according to the analysis results can work exactly as that of the source program of MPEG2 video decoder. Therefore, our analysis results are verified.

5.2 Power and Performance Estimation

This section reports some experimental data concerning the use of our approach to reduce energy consumption. The cycle count is obtained by using our instruction-level simulator. The input of the simulator is the assembly code, which is generated by the retargetable Valen-C compiler. Results of energy consumption E_t (shown in **Fig. 6**) include energy of a soft-core processor (E_p), a data RAM (E_s) and an instruction ROM (E_r), where $D.W$ is datapath width. We use the energy consumption models in Section 4. Apparently, the energy consumption changes nonlinearly.

Figure 7 shows the energy consumption, execution cycles and area (gates) of MPEG2 video decoder, and we got the optimal datapath width, 28 bits for MPEG2 video decoder. **Figure 8** describes the energy savings of our benchmarks, such as Lempel-Ziv algorithm, ADPCM encoder, and MPEG2 AAC decoder and so on. No Opt. means the original datapath width of Bung-DLX (32 bits). Opt. is the datapath width where the whole system has the mini-

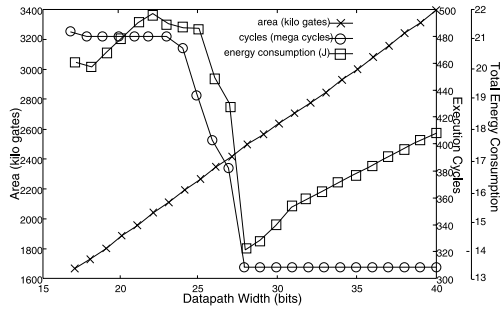


Fig. 7 Energy consumption, execution cycles and area (gates) for MPEG2 decoder.

Applications	Energy Consumption (J)		
	No Opt.	Opt.	Savings
Lempel-Ziv	0.9517	0.4919	48.3%
ADPCM	1.181	0.9187	22.8%
MPEG2	157.16	140.24	10.8%

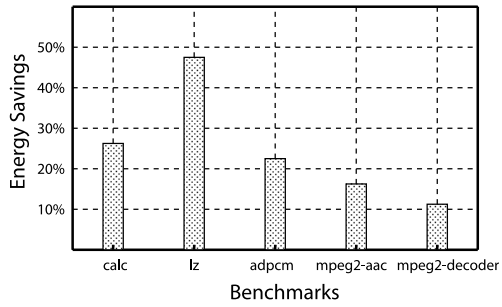


Fig. 8 Energy savings for benchmarks.

mization energy consumption without performance loss. For Lempel-Ziv algorithm, we got energy savings of 48.3% at datapath width of 15 bits, for ADPCM encoder, energy savings is 22.8% at datapath width of 19 bits and for MPEG2 video decoder, the energy savings is 10.8% at datapath width of 28 bits. For different application, the number of variables is different and the effective size of variables is also different, therefore the optimal datapath width of minimal energy is different. For a given application, our approach just tries to take advantage of the characteristics of the application to reduce the energy consumption.

6. Conclusions

In this paper, we proposed a system-level energy minimization technique through datapath width optimization, which can suit the complexity of embedded systems and stringent time-to-market constraints. We also presented a set of algorithms that minimize energy consumption in system-level. We illustrated issues and tradeoffs involved in the design. Our ex-

perimental results show that for a given application we can reduce significantly the energy consumption by datapath width optimization. We have demonstrated energy savings without performance penalty range from about 10.8% to 48.3%, which based on a number of real embedded applications. Extending parameter-tuning for low power to DSPs is our future work.

Acknowledgments This research was partly supported by the Grant-in Aid for Scientific Research (B) (2) 12558029 and VCDS project of STARC.

References

1) Yasuura, H., Tomiyama, H., Inoue, A. and Eko, F.N.: Embedded System Design Using Soft-Core Processor and Valen-C, *Journal of Information Science and Engineering*, No.14, pp.587–603 (Aug. 1998).
2) Inoue, A., Tomiyama, H., Okuma, T., Kanbara, H. and Yasuura, H.: Language and Compiler for Optimizing Datapath Width of Embedded Systems, *IEICE Trans. Fundamentals*, Vol.E81-A, No.12, pp.2595–2604 (1998).
3) Eko, F.N., Inoue, A., Tomiyama, H. and Yasuura, H.: Soft-Core Processor Architecture for Embedded System Design, *IEICE Trans. Electronics*, Vol.E81-C, No.9, pp.1416–1423 (1998).
4) Eko, E.N. and Yasuura, H.: A Cycle-Accurate Simulator Toolkit for Soft-Core Processors, *Proc. Asia Pacific Conference on cHip Design Languages* (*APCHDL '99*), pp.11–16 (Oct. 1999).
5) Shackleford, B., Yasuda, M., Okushi, E., Koizumi, H., Tomiyama, H. and Yasuura, H.: Embedded System Cost Optimization via Data Path Width Adjustment, *IEICE Trans. Information and Systems*, Vol.E80-D, No.10, pp.974–981 (1997).
6) Inoue, A., Ishihara, T. and Yasuura, H.: Flexible system lsi for embedded systems and its optimization techniques, *Journal of Design Automation for Embedded System*, Vol.5, No.2 (2000).
7) Panda, P.R., Catthoor, F., Dutt, N.D., Danckaert, K., Brockmeyer, E. and Vandercappelle, A.: Data and Memory Optimization Techniques for Embedded Systems, *ACM Trans. Design Automation of Electronic Systems*, Vol.6, No.2, pp.149–206 (2001).
8) Panda, P.R., Dutt, N.D., Catthoor, F., Vandercappelle, A., Brockmeyer, E., Kulkarni, C. and Greef, D.: Data Memory Organization and Optimizations in Application-Specific Systems, *IEEE Design & Test of Computers*, Vol.18, No.3, pp.56–68 (MAY–JUNE 2001).
9) Panda, P.R., Dutt, N.D. and Nicolau, A.: On-Chip vs. Off-Chip Memory: The Data Partitioning Problem in Embedded Processor-Based Systems, *ACM Trans. Design Automation of Electronic Systems*, Vol.5, No.3, pp.682–704 (2000).
10) Benini, L., Hodgson, R. and Siegel, P.: System-level Power Estimation and Optimization, *International Symposium* on *Low Power Electronics and Design*, pp.173–178 (Aug. 1998).
11) Landman, P. and Rabaey, J.: Architectural Power Analysis: The Dual Bit Type Method, *IEEE Trans. on VLSI Systems*, Vol.3, No.2 (1995).
12) Sato, T., Nagamatsu, M. and Tago, H.: Power and Performance Simulator: ESP and its Application for 100 MIPS/W Class RISC Design, *IEEE Proc. Symposium* on Low Power Electronics and Design, pp.46–47 (1994).
13) Hong, I., Kirovski, D., et al.: Power Optimization of Variable voltage Core-Based Systems, *IEEE Proc. 35th* Design Automation Conference (DAC '98), pp.176–181 (1998).
14) Ko, U. and Balsara, P.: Energy Optimization of Multilevel Cache Architectures for RISC and CISC Processors, *IEEE Trans. on VLSI Systems*, Vol.6, No.2, pp.299–308 (1998).
15) Henkel, J.: A Low Power Hardware/software partitioning Approach Core-Based Embedded Systems, *IEEE Proc. 36th. Design Automation Conference* (*DAC '99*), pp.122–127 (1999).

(Received September 20, 2001)
(Accepted January 16, 2002)

Yun Cao is a Ph.D. candidate in Department of Computer Science and Communication Engineering, Kyushu University, Japan. She received her B.E. degree of Electronics and Information Engineering from Huazhong University of Science and Technology, China, in 1989. Her research interests are hardware/software co-design, low power/low energy system design and system design methodology. She is a student member of IPSJ, IEEE and ACM.

Hiroto Yasuura is a professor of Department of Computer Science and Communication Engineering, Graduate School of Information Science and Electrical Engineering, and the director of System LSI Center of Kyushu University, Fukuoka, Japan. He received the B.E., M.E., and Ph.D. degrees in computer science from Kyoto University. His current interests include parallel computer architectures, hardware algorithms for VLSI, VLSI CAD, and system design methodology. He is a member of IPSJ, ACM, and IEEE.