Leakage Power Reduction Using Bitwidth Optimization

Cao, Yun
Department of Computer Science and Communication Engineering, Kyushu University

Yasuura, Hiroto
Department of Computer Science and Communication Engineering, Kyushu University

出版情報：Proc. of the 6th World Multiconference on Systemics, Cybernetics and Informatics (SCI2002), pp.36-41, 2002-07. World Multiconference on Systemics, Cybernetics and Informatics
バージョン：
権利関係：

Leakage Power Reduction Using Bitwidth Optimization

Yun Cao Hiroto Yasuura
Department of Computer Science and Communication Engineering
Kyushu University
6–1 Kasuga-koen, Kasuga-shi, Fukuoka 816-8580, Japan
{cao,yasuura}@c.csce.kyushu-u.ac.jp

ABSTRACT

Leakage power dissipation constitutes an increasing fraction of the total power in modern semiconductor technologies. Designing power efficient products will require consideration of leakage power in the earliest phases of design. This paper addresses bitwidth optimization focusing on leakage power reduction for system-level low-power design. By means of tuning the design parameter, bitwidth tailored to a given application requirements, the datapath width of processors and size of memories are optimized resulting in significant leakage power reduction besides dynamic power reduction. In our experiments for several real embedded applications, power reduction without performance penalty are reported range from about 21.5% to 66.2% of leakage power, and 14.5% to 59.2% of dynamic power.

Keywords

Bitwidth optimization, leakage power reduction, dynamic power reduction

1. INTRODUCTION

The increasing use of battery-operated portable computing and wireless communication systems makes power dissipation a major concern in modern designs [1]. Reducing power dissipation hence becomes a crucial challenge for today's software and hardware designers. Maximization of battery life is an obvious goal for these applications. Extensive researches for low power designs show that minimization of power dissipation can be considered at all design levels from circuit level to system level [2] [3] [4] [5].

In CMOS digital circuits, power dissipation consists of dynamic and static components. In circuits with a high supply voltage, a relatively high transistor threshold voltage can be used, therefore subthreshold current can be negligible. That is the common assumption for the existing techniques for average power optimization [3] [4] [5]. However, low power applications have been driving the supply voltage to become lower and lower, which requires the device threshold to be reduced so as to satisfy performance requirements. This leads to dramatic increase of leakage current due to the exponential relationship between leakage current and threshold voltage. Consequently, leakage power (static power) is no longer negligible in low voltage circuits. Two implementations of Intel's Pentium III processor manufactured on Intel's $0.18\mu m$ process are good examples. They are the Pentium III 1.0 GHz B and the Pentium III 1.13 GHz [6]. The Intel datasheet lists the maximum core power dissipation of the 1.0 GHz part at 33.0 watts and the deep sleep (i.e. leakage) power dissipation at 3.74 watts. The 1.13 GHz processor has a total power dissipation of 41.4 watts and leakage power dissipation of 5.40 watts. While the total power has increased by only 25%, the leakage power has increased by 44% and comprises 13% of the total power dissipation. The dynamic power dissipation of the processor core varies significantly depending on the workload while the leakage power dissipation is almost constant. Therefore, leakage power is even a larger percentage of the total power dissipation on average. Reducing leakage power can be especially important to battery when a system is idle for a long time, such as for mobile phones.

Figure1 shows the increase in leakage and dynamic power for Intel's past few technologies [7]. These trends indicate that leakage power will likely contribute as much to total power as dynamic power in as little as two technology generations. Therefore leakage power should be considered as important as dynamic power when making design tradeoffs, optimization techniques for leakage power is necessary. The device and circuits communities have been concerned with increasing leakage power for several generations. Paper [8] presents the various leakage modes of the MOS transistor and identify subthreshold leakage as the dominant one. Paper [1] projects dual V_t(transistor threshold voltage) design technique will be widely used to reduce leakage power.

As far as we know, prior work on power reduction at the system level has been focused almost entirely on dynamic power. In order to limit dynamic power dissipation, techniques such as clocking gating [3] and cache subbanking [9] have been employed. The goal of these techniques is to reduce the number or frequency for switching devices. Optimization of the supply voltage to minimize the power/performance ratio is also performed [5], this has the added benefit of addressing dynamic power dissipation, which is proportional to the square of the supply voltage. However, all of these techniques are hardly used to reduce leakage power. Some may claim that system level designers have no control over leakage power because of its strong dependence on technology and circuit optimization. We think that although lower level optimization more directly affects the final leakage power dissipation, awareness of the issue during the system design can result in a design better suited to later optimization.

This paper presents bitwidth optimization focusing on leakage power reduction for system level design, while providing adequate performance level. Using the technique, system designers can control the design parameters, such

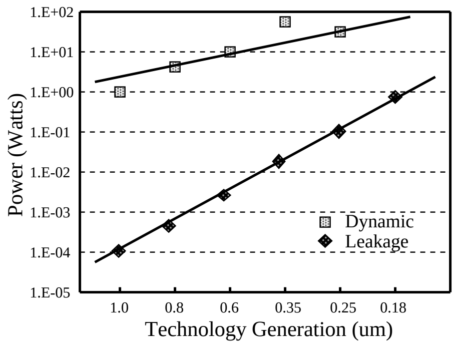


Figure 1: Trends in dynamic power and leakage power dissipation(from [7])

as the datapath width of processors freely. The power dissipation of the whole system not only dynamic power but also leakage power is drastically reduced by tuning the parameters of processors and memories tailored for the applications. To get first-cut estimates of leakage power dissipation early in the design, a few component-based estimation models are also developed.

The rest of this paper is structured as follows: the next section 2 presents leakage power reduction technique using bitwidth optimization, estimation models for leakage power dissipation are also presented in this section. Experiments are described in section 3. Finally, section 4 concludes our work.

2. LEAKAGE POWER REDUCTION

As transistors become smaller and faster, leakage power dissipation due to leakage current in the absence of any switching activity has become important, system designers will be called upon to consider it in making design decisions.

2.1 A Leakage Power Model

We have proposed bitwidth optimization for dynamic power reduction in [15]. Dynamic power dissipation is described by the familiar formula, $P_{dyn} = \frac{1}{2} \cdot C \cdot V_{cc}^2 \cdot f$, C is the capacitance of switching nodes, which is roughly proportional to the number of switching devices. V_{cc} is the supply voltage, and f is the effective operating frequency (frequency times activity factor). Our technique reduces dynamic power by reducing C and f.

This paper focuses on leakage power reduction. Leakage power dissipation is equal to the product of the supply voltage and the leakage current. To deal with the leakage power at system level, we start addressing leakage power model from a simple equation for estimation in [10],

$$P_{static} \quad = \quad V_{cc} \cdot N \cdot k_{design} \cdot \hat{I}_{leak} \tag{1}$$

where V_{cc} is the supply voltage, N is the number of transistors, k_{design} is a design dependent parameter, and $\hat{I}_{leak}$ is a technology dependent parameter. This model enables high-level reasoning about the likely leakage power demands of alternative designs. Reasonably accurate values for the factors in the equation may be obtained directly from the high-level designs. The factors in the equation also suggest opportunities for leakage power optimization, including reducing the total number of devices, partitioning the design to allow for lower supply voltages or slower, less leaky transistors, turning off unused devices, favoring certain design styles, and favoring high bandwidth over low latency. The parameters of the leakage power model of the equation may be divided into two groups. The technology parameters are derived from measurements or simulations of individual devices. They are all dependent on a host of lower-level process parameters(e.g., oxide thickness and doping profiles) in complex ways. The design dependent parameters(V_{cc}, N and k_{design}) apply to groups of devices interconnected in a specific design style. Within certain constraints, they are independent of the process technology and may be varied independently.

The model suggests different ways in which leakage power may be reduced. One obvious technique that can be employed is to reduce the total number of devices. However, finding opportunities to reduce the device count enough to impact power dissipation without decreasing performance or functionality is difficult, normal design practices eliminate obvious redundancy. At system level the number of transistors (represented by N) can often be estimated although circuit designs are not yet available. Presuming a circuit with known functionality has been designed in the past, a reasonably accurate estimate may be obtained with little effort. Design exploration to get optimized alternatives can be achieved by estimation without reaching the circuit design phase. N is only constrained by the functionality required of the circuit and the available area in which to implement it. For a given functionality, the number of transistors should be constant across generations.

In this paper, we present bitwidth optimization to reduce the number of devices(N) by tuning datapath width of processors for each application, to try to ease the problem of leakage power. Because reducing the number of devices directly reduces the switching capacitance(C), our technique also reduce dynamic power dissipation as well.

2.2 Design Platform

We have developed a design platform for embedded core-based systems, which consists of a variable configuration processor called Bung-DLX [11], a multi-precision retargetable compiler called Valen-C retargetable compiler [12], a variable size analyzer [13] and a cycle-based simulator [14].

2.2.1 A Variable Configuration Processor

A variable configuration processor called Bung-DLX, is a prototype of embedded processor, which has some design parameters. The parameters can be tuned by designers for each application, and a customized processor optimized for the application can be obtained. By using the variable configuration processor, development of embedded systems becomes easier and requires less time, further more performance and power optimization can be achieved.

Bung-DLX is designed based on DLX architecture. The number of general-purpose registers, datapath width and instruction set are parameters in Bung-DLX. Bung-DLX description is written in VHDL around 7000 lines, it inherits its properties from DLX such as addressing mode and datapath structure. The number of general-purpose registers can be changed to any value bigger than 1. This makes it possible to customize the number of general-purpose registers. For example, a resource intensive application may require more registers, on the other hand, control intensive application may require less registers. The width of datapath (also the width of registers) is scalable too, but it must be bigger than the width of special purpose register width. The datapath width of the processor need not to be 2^n (n is a positive integer number), it can be any bit. If an application, for instance, requires only at most 21-bit precision, datapath wider than 21 bits can not give any performance improvement, namely the width of datapath more than 21 bits is redundant. Bung-DLX incorporates Harvard style addressing, i.e., separation of data and program space. This makes the width of program counter(PC) and the width of memory address register(MAR) can be defined independently. In addition, the bit width of data memory and instruction memory space can be different. System designers can tune the value of these parameters in accordance with the characteristics of target system to deliver most suited processor.

2.2.2 A Multi-precision Retargetable Compiler

If processor architecture is modified, the compiler for the processor also needs to be modified. To make the modification easier, we developed a multi-precision retargetable compiler called Valen-C compiler, which uses SUIF (Stanford University Intermediate Format) library. The feature of it is that any datapath width can be applicable without 8×2^n limit.

Valen-C language(Variable Length C) has been developed [12]. It is an extended C language, by which system designers can specify the required bit length of each variable in programs explicitly. Even if system designers customize the datapath width for their application, the Valen-C programs can be reused on processors with various datapath widths. Valen-C is one solution for the problem of word-length support in C language.

The Valen-C compiler is retargetable by modifying the machine description including the datapath width, the number of registers, the instruction set, the sizes and alignments of the program and data memories, the minimum addressable size of the data memory, and so on. The Valen-C compiler preserves the precision of programs in the following manner: If a variable has a precision of n bits, the Valen-C compiler allocates the storage of not less than n bits for the variable. If an operation in a Valen-C program requires the precision of n bits, the operation is performed with the precision of not less than n bits. For example, an addition of two 13-bits variables will be calculated with a precision of 20 bits on 20-bit processors. In cases that the precision of an operation is bigger than the datapath width, the operation is performed by a certain number of machine instructions. For example, an addition with a 20-bit precision is performed by two addition instructions of lower 10 bits and upper 10 bits on a 10-bit processor.

2.2.3 A Variable Size Analyzer

Although the Valen-C language has enhanced reusability over ANSI-C, specifying bitwidth of variables is vary cumbersome and time-consuming. Therefore, we have developed techniques for variable size analysis, which automatically analyze required bitwidth of variables in C programs. Using the techniques, C programs are automatically translated into Valen-C programs.

We define *variable effective size* as the smallest size of a variable which can hold both maximum and minimum values of a variable. We use two methods to analyze effective size of variables [13]. One is static analysis, the other is simulation-based dynamic analysis.

```
Input:
  source program : $AP$
      (variables : $x_i \in X = \{x_1, x_2, ..., x_n\}$, $1 \leq i \leq n$)
  input data : $D_{in}$
  the constraint of cycles : $C_{cst}$
Variable:
  datapath width $w_i \in W$=$\{w_1$,$w_2$,...$w_n\}$
Output:
  execution cycles $c_i \in C = \{c_1$,$c_2$,...$c_n\}$
  the minimal leakage power dissipation $P_{sMin}$
            when $c_k \leq C_{cst}$
  the datapath width $w_k$ when $P_{s_k} = P_{sMin}$
Phase 1 : Variable Size Analysis
  {
   $analyzer \leftarrow (AP, X, D_{in})$
   $return(EWd = \{EWd(x_1), EWd(x_2), ..., EWd(x_n)\})$
  }
Phase 2 : Define Design Parameters for Bung-DLX
  datapath width $w_i$
  the number of registers $n_i$
Phase 3 : Valen-C program
  variable declaration of bit width $EWd(x_i)$
  compile the Valen-C source program for customized
            Bung-DLX at $w_i$
Step 4 : Bitwidth Optimization
  for $W \neq \emptyset$
  {
    estimate the execution cycles $c_i$
    estimate the leakage power dissipation $P_s$
    $P_{sMin} \leftarrow$ the minimal leakage power when $w_i = w_k$
            under $c_k \leq C_{cst}$
  }
   $return(P_{sMin}, w_k, c_k)$
```

Figure 2: Pseudo code of the algorithm for leakage power reduction

2.2.4 A Cycle-based Simulator

An instruction level simulator called Bung-DLX simulator, which consists of a controller and a datapath, is also developed. The input of the simulator is the assembly code generated by the retargetable Valen-C compiler.

The innovation of the simulator is decoupling processor internal disassembler (i.e. instruction decoder) from the simulation description. A symbolic instruction format as the input of the simulator is used. Since during architecture/instruction set exploration phase it is hardly needed to encode instructions into bitmap, which has little direct effect to simulator accuracy. The simulator achieved a slight simulation speed gain. In addition, any kind of instructions is easily formed by the simulator including those, which are difficult to be expressed in assemble-disassemble approach such as a variable length instruction, a multi operand instruction and an instruction with complex encoding.

2.3 Bitwidth Optimization

Optimizing bitwidth of datapath for each given application is an effective technique to reduce leakage power dissipation of the whole embedded systems. The leakage power reduction problem is formulated as *to minimize* $P_s(w)$*, subject to* $Cycle(w) \leq C_{cst}$. Leakage power $P_s(w)$ and cycle $Cycle(w)$ are functions of datapath width w, C_{cst} is the constraint on the execution cycle.

The overview of our leakage power reduction algorithm is described in Figure2. In the initial design phase of our approach, we design a system with a variable configuration processor(Bung-DLX), data RAMs, instruction ROMs and logic circuits. Then we analyze the effective bitwidth of each variable ($EWd(x_i)$) in a given application program (AP). After that, using the results of analysis, we rewrite the application program in Valen-C language, in which we specify the word length of each variable satisfying accurate computation to reduce leakage power consumed by redundant bits in the application program. After verifying the functionality of the initial design, we modify several design parameters of the variable configuration processor, including the datapath width w_i, the number of registers and the instruction set. We can tune up the variable configuration processor to minimize the leakage power dissipation (P_{sMin}) while satisfying the system performance constraints($c_k \leq C_{cst}$).

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 power-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 are useless. C language provides three integer sizes, declared using the keywords *short*, *int* and *long*. The compiler designer determines the sizes of these integer types. We present Valen-C language, by which programmers can explicitly specify the required bitwidth of each integer data type, so it becomes possible to reduce the leakage power of the datapath and the data memory, which is dissipated by the redundant bits. As a result, specifying the bitwidth required for each variable and changing the datapath width have a significant role in reducing the processor and the data memory size of a system. Therefore it also affects the power dissipation of the system.

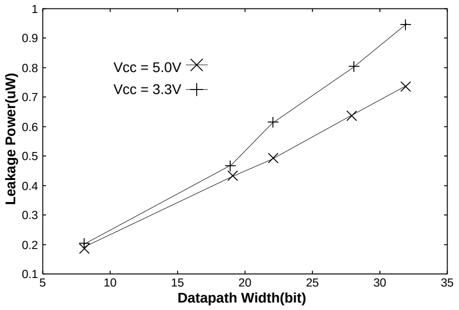
The value of the datapath width can be tuned in accordance with the characteristics of target system to deliver most suited processor. Designers can reduce the datapath width until the single precision point (SPP) without performance loss. SPP is the processor datapath width, which is equal to the bitwidth of the biggest variable in a program. It is the smallest datapath width at which all instructions can remain single-precision. The datapath width of a processor strongly affects the power dissipation 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. So trade-offs exist between datapath width and execution cycles. Therefore, for a given target system, trading off the power dissipation and performance is an important work for bitwidth optimization.

2.4 Estimation Models for Leakage Power

This section presents leakage power models to give the criteria for leakage power reduction. We assume that the target system consists a processor, a data RAM and an instruction ROM. The variable configuration processor Bung-DLX is used. The total leakage power dissipation, P_{sTot} of a system, is estimated as the summation of leakage power consumed by the processor (P_{sProc}) and memories (P_{sMem}).

$$P_{sTot} = P_{sProc} + P_{sMem} \tag{2}$$

P_{sProc} and P_{sMem} are estimated separately. We built the leakage power dissipation model of Bung-DLX generated by HITACH 0.5 μm CMOS technology and the power dissipation models of memories generated by Alliance CAD System Ver.4.0 with 0.5 μm double metal CMOS technology.



$DatapathWidth$	$V_{cc} = 5.0V$		$V_{cc} = 3.3V$	
	P_{sProc}	$Saving$	P_{sProc}	$Saving$
32bit	0.74	-	0.95	-
28bit	0.64	13.5%	0.80	15.8%
22bit	0.49	33.8%	0.61	35.8%
19bit	0.43	41.9%	0.47	61.1%
8bit	0.19	74.3%	0.20	79.0%

Figure 3: Leakage power of Bung-DLX (μw)

P_{sProc} is obtained by using Synopsys Power Compiler. The power dissipation in static CMOS circuitry can be divided into static(leakage), dynamic and short-circuit power. Here we just focus on leakage power (P_{sProc}). After several simulations, we obtained the empirical power model at several datapath widths for supply voltage V_{cc} of $5.0V$ and $3.3V$ respectively, shown in the table of Figure3, where power savings, $Saving$ are got by comparing to the leakage power dissipation of the 32bits processor.

P_{sProc} can be described as follows:

$$P_{sProc} = \sum_{\forall cells_k} P_{CellLeakage_k} \tag{3}$$

P_{sProc}: Total leakage power of a processor
$P_{CellLeakage_k}$: Leakage power of each cell k
P_{sMem} is estimated as follows:

$$P_{sMem} = P_{sROM} + P_{sSRAM} \tag{4}$$

where
P_{sMem} : Total leakage power of memory
P_{sROM} : Leakage power of ROM
P_{sSRAM} : Leakage power of SRAM

The power of P_{sROM}, P_{sSRAM} is obtained from the SPICE simulation of several memories with different configurations. As the result, we have obtained the estimation models as follows:

$$P_{sROM} = 18.1 * b * \sqrt{N_{words}} + 0.08[pw] \tag{5}$$
$$P_{sSRAM} = 231.0 * b * \sqrt{N_{words}} + 1.1[pw] \tag{6}$$

Where b is the bit width of the memory and N_{words} is the number of words.

3. EXPERIMENTS

This section reports some experimental results concerning the use of our technique to reduce power dissipation based on several real applications. We report both dynamic power and leakage power results.

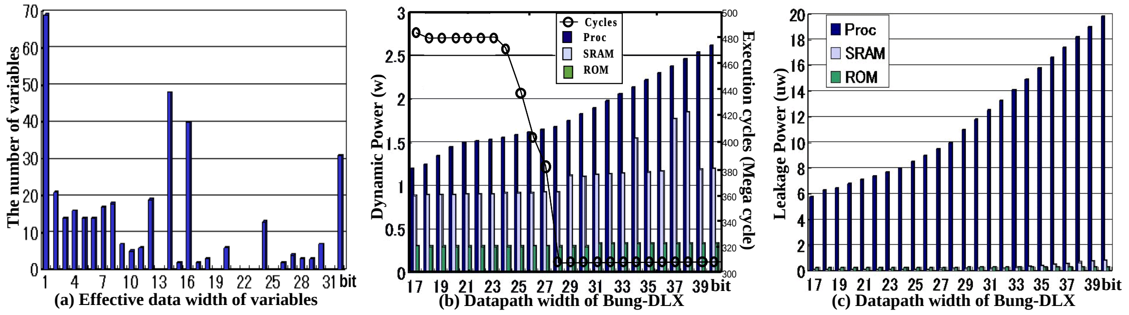


Figure 4: Results of MPEG-2 video decoder

Application	No. Opt. Dynamic Power(mw)				Opt. Dynamic Power(mw)				Savings
	P_{dProc}	P_{dSRAM}	P_{dROM}	P_{dTot}	P_{Proc}	P_{dSRAM}	P_{dROM}	P_{dTot}	
Lempel-Ziv	645.8	330.2	66.98	1.04w	208.3	184.6	31.4	424.3	59.2%
ADPCM	155.5	82.5	118.5	356.5	77.5	51.28	70.3	199.1	44.2%
Mpeg2AAC	589.3	247.6	337.23	1.17w	503.6	194.5	301.7	999.8	14.5%
Mpeg2Video	2.05w	1.16w	349.68	3.56w	1.68w	930.72	305.97	2.91w	18.3%

Application	No. Opt. Leakage Power(μw)				Opt. Leakage Power(μw)				Savings
	P_{sProc}	P_{sSRAM}	P_{sROM}	P_{sTot}	P_{sProc}	P_{sSRAM}	P_{sROM}	P_{sTot}	
Lempel-Ziv	3.8	36.7nw	59.7nw	3.90	1.28	19.8nw	24.9nw	1.32	66.2%
ADPCM	0.95	8.8nw	0.11	1.06	0.48	5.7nw	61.7nw	0.55	48.1%
Mpeg2AAC	3.1	28.3nw	0.30	3.43	2.3	21.5nw	0.26	2.59	21.5%
Mpeg2Video	13.3	0.21	0.31	13.82	10.0	0.18	0.27	10.4	24.7%

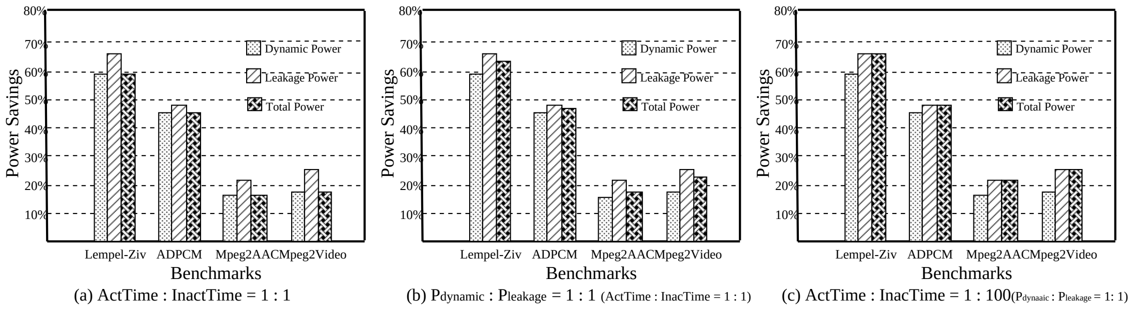


Figure 5: Power savings for benchmarks

In the experiments, we assumed the target system, a SOC chip, which consists a Bung-DLX processor, a ROM and a SRAM. The ROM and the SRAM are used as instruction memory and data memory respectively. For simplicity, we assumed that no other core is integrated in the SOC chip.

3.1 Results

Four real embedded applications are used as benchmarks, which are Lempel-Ziv algorithm, ADPCM encoder, MPEG-2 AAC audio decoder, and MPEG-2 video decoder. The cycle count is used to evaluate performance obtained by using the simulator [14]. For dynamic power estimation, the power models in [15] are used, and for leakage power estimation, models in section 2.4 are used.

Figure 4 shows the estimation results for MPEG-2 video decoder. We analyzed the C source program of MPEG-2 video decoder from the MPEG Software Simulation Group, using our developed variable size analyzer and got the variable size analysis results of effective data width depicted in Figure4(a). This figure shows that there are a lot of variables having many unused bits in MPEG-2 decoder, which originally declared as "int" type. We got average 39% reduction of bits from the variable size analysis. The dynamic power dissipation and execution cycles are described in Figure 4(b), and the estimation results of leakage power shown in Figure 4(c). From these figures, we can get the optimal datapath width, 28bits for MPEG-2 video decoder without performance loss. Optimal datapath width is the datapath width where the whole system has the minimization power dissipation without performance penalty.

Figure5 shows the results of the experiments employed our technique for the benchmarks. In the first table of Figure5 for dynamic power, column $No.Opt.DynamicPower$ including four columns, which show the results for original designs without using our technique. Column P_{dProc} shows the dynamic power dissipation of processor Bung-DLX, column P_{dSRAM} is dynamic power of SRAM, column P_{dROM} is the dynamic power of ROM, and column P_{dTot} shows the total dynamic power dissipation. The next four columns show the results using our optimization technique($Opt.$). The last column $Savings$ shows the dynamic power reduction compared to the designs without using our technique($No.Opt.$).

The results of leakage power obtained are listed in the second table of Figure 5. Column $No.Opt.LeakagePower$ show the results for original designs without using our technique. Column P_{sProc} shows the leakage power for processor Bung-DLX, column P_{sSRAM} for SRAM, column P_{sROM} for ROM, and column P_{sTot} shows the total leakage power dissipation. The next four columns show the results using our optimization technique($Opt.$). The last column $Savings$ shows the leakage power reduction compared to the designs without using our technique($No.Opt.$).

3.2 Discussion

Figure 5(a) shows the power savings of our benchmarks including for dynamic power, leakage power and total average power. The value of $Savings$ is the results by comparing our technique with original designs(datapath width of Bung-DLX is 32bits). For Lempel-Ziv algorithm, we got dynamic power saving of 59.2% and leakage power saving of 64.3% at the optimal datapath width of 15bits; for ADPCM encoder, dynamic power savings is 44.2% and leakage power saving is 47.4% at the optimal datapath width of 19bits; for MPEG-2 AAC audio decoder, the dynamic power saving is 14.5% and leakage power saving is 18.1% at the optimal datapath of 24bits and for MPEG-2 video decoder, the dynamic power savings is 18.3% and leakage power is 19.1% at optimal datapath width of 28bits. 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 power is different. For a given application, our technique just tries to take advantage of the characteristics of the application to reduce the power dissipation.

Our technique achieved drastically average power reduction up to 59.2% shown in Figure 5(a), note that this is under the assumption $ActTime : InactTime = 1 : 1$. $ActTime$ is the application execution time, which is called active time and $InactTime$ is the idle time, which is called inactive time. However with the rapid increase of leakage power dissipation, almost half the power of the chip can be from the leakage power [1]. In this case, we can get total average power saving up to 62.7% ($P_{dynamic} : P_{leakage} = 1 : 1$) shown in Figure 5(b), and further more, considering such kind of applications like mobile phone, the idle time is far more than active time (ActTime : InactTime = 1 : 100), we can get more average power saving up to 66.2% shown in Figure 5(c).

4. CONCLUSIONS

In this paper, we have proposed a system-level design technique focusing on leakage power reduction, which can suit the complexity of embedded systems and stringent time-to-market constraints. The presented bitwidth optimization technique can reduce both dynamic power and leakage power dissipation at system-level. We illustrated issues and tradeoffs involved in the design. Our experimental results show that for a given application we can reduce significantly the power dissipation by bitwidth optimization. We have demonstrated power savings without performance penalty average about 40.1% of leakage power, and 34.1% of dynamic power, which based on a number of real embedded applications.

5. ACKNOWLEDGMENTS

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

REFERENCES

[1] S. Borkar, “Low Power Design Challenges for the Decade”, Proc. of Asia South Pacific Design Automation Conference, pp. 293-296, 2001

[2] C. Svensson and D. Liu, ” ’Low Power Circuit Techniques,’ in Low Power Design Methodologies”, pp. 37-64, Kluwer Academic, Norwell, MA, 1996.

[3] V.Tiwari, D.Singh, S.Rajgopal, G.Mehta, R.Patel, F.Baez, “Reducing Power in High-Performance Microprocessors”, Proc. of the 35th Design Automation Conference, pp. 732-737, 1998.

[4] L.Benini, R.Hodgson and P.Siegel, “System-level Power Estimation and Optimization”, Proc. of International Symposium on Low Power Electronics and Design, pp. 173-178, 1998.

[5] I.Hong, D.Kirovski et al., “Power Optimization of Variable Voltage Core-Based Systems”, Proc.of the 35th Design Automation Conference, pp.176-181,1998.

[6] Intel Corporation. Pentium III Processor for the SC242 at 450 MHz to 1.13 GHz Datasheet, pp. 26-30.

[7] S. Thompson, P. Packan and M. Bohr, “CMOS Scaling: Tansistor Challenges for the 21st Century”, Intel Technology Journal, Q3, 1998.

[8] A. kshavarzi, K. Roy, and C. Hawkins, “Intrinsic Leakage in Low Power Deep Submicron CMOS ICs”, Proc. of International Test Conference, pp. 146-155, 1997.

[9] U.Ko and P.Balsara, “Energy Optimization of Multilevel Cache Architectures for RISC and CISC Processors”, IEEE Transactions on VLSI Systems, vol.6, no.2, pp. 299-308, 1998.

[10] J. Adam Butts and Gurindar S. Sohi, “A Static Power Model for Architects”, Proc. of the 33th Annual International Symposium on Microarchitecture, pp. 191-201, 2000.

[11] F. N. Eko, A. Inoue, H. Tomiyama, H. Yasuura, “Soft-Core Processor Architecture for Embedded System Design”, IEICE Trans.on Electronics, Vol.E81-C, No.9, pp. 1416-1423, 1998.

[12] A.Inoue, H.Tomiyama, T.Okuma, H.Kanbara and H.Yasuura, “Language and Compiler for Optimizing Datapath Width of Embedded Systems”, IEICE Trans. Fundamentals, Vol. E81-A, No.12, pp. 2595-2604, Dec. 1998.

[13] H. Yamashita, H. Yasuura, F. N. Eko, and Yun Cao, “Variable Size Analysis and Validation of Computation Quality”, Proc. of Workshop on High-Level Design Validation and Test, pp.95-100, 2000.

[14] E. N. Eko and H. Yasuura, “A Cycle-Accurate Simulator Toolkit for Soft-Core Processors”, Proc. of Asia Pacific Conference on CHip Design Languages, pp. 11-16, 1999.

[15] Yun Cao, Hiroto. Yasuura, “A System-level Energy Minimization Approach Using Datapath Width Optimization”, Proc. of International Symposium on Low Power Electronics and Design, 2001.