The 2004 IEEE Asia-Pacific Conference on
Circuits and Systems, December 6-9, 2004

EXPERIMENTAL EVALUATION OF HIGH-LEVEL ENERGY OPTIMIZATION BASED ON THREAD PARTITIONING

Jumpei UCHIDA[†], *Yuichiro MIYAOKA*[†], *Nozomu TOGAWA*[††,‡], *Masao YANAGISAWA*[†] *and Tatsuo OHTSUKI*[†]

[†] Dept. of Computer Science, Waseda University
[††] Dept. of Information and Media Sciences, The University of Kitakyushu
[‡] Advanced Research Institute for Science and Engineering, Waseda University

ABSTRACT

This paper presents a thread partitioning algorithm for high-level synthesis systems which generate low energy circuits. In the algorithm, we partitions a thread into two sub-threads, one of which has RF and the other does not have RF. The partitioned sub-threads need to be synchronized with each other to keep the data dependency of the original thread. Since the partitioned sub-threads have waiting time for synchronization, gated clocks can be applied to each sub-thread. We achieve 33% energy reduction when we apply our proposed algorithm to a JPEG encoder.

1. INTRODUCTION

Recently, design complexity is highly increasing. At the same time, requirements for low energy system LSIs are also increasing for the needs of cellular phones, PDAs, and mobile PCs. We should develop high speed, small area and low energy system LSIs in a short period of time. One of the solutions of these requirements is using high-level synthesis systems which are able to synthesize low energy system LSIs.

Several power reduction techniques at high-level synthesis were proposed. Gated clocks were exploited for power reduction techniques [1],[6],[9]. In [1], gated clocks were applied efficiently by reducing waiting time of logic circuits. In [6], area/delay/power estimation for low power system LSIs with gated clocks was proposed. In [9], a clock tree was generated based on the profile of switching activities at high-level. A binding technique of functional units for reducing switching activities was proposed in [4]. A register allocation and binding technique was adopted for the purpose of minimizing switching probability [2]. A scheduling technique for low power circuits was proposed in [5]. These days, several practical high-level synthesis systems were developed. In these high-level synthesis system, we can define *threads* as parallel behaving circuit blocks. In [3],[8],[10] a high-level synthesis system called Bach system has been proposed. The input language of Bach system is called BachC where we can describe threads explicitly. Each thread has waiting time for synchronization, since it has synchronous communication to keep the data dependency. The Bach system has a high-level energy reduction mechanism. In order to reduce energy consumption, the Bach system automatically applies gated clocks to all of the threads in a BachC description[7].

Assume that we have a single thread in a BachC description. If this thread is partitioned into two or more sub-threads, we can further reduce energy consumption. This is because each partitioned sub-thread must have waiting time for synchronization and thus we can apply gated clocks to each of the partitioned sub-threads. We consider that thread partitioning must be one of the most powerful energy reducing techniques.

Based on the above idea, we propose in this paper a thread partitioning algorithm for high-level synthesis systems which generate low energy circuits. First the algorithm focuses on a local register file RF in a thread. It partitions a thread into two sub-threads, one of which includes RF and the other does not include RF. Each of the partitioned two sub-threads needs to be synchronized between the two sub-threads to keep the data dependency of the original thread. Since the partitioned two sub-threads have waiting time for synchronization, gated clocks can be applied to each of the partitioned sub-threads. We can synthesize a low energy circuit with small additional area overhead compared to the original circuit. In this paper, we also present experimental evaluation of high-level energy optimization based on thread partitioning. We apply our proposed algorithm to a JPEG encoder.

This paper is organized as follows: Section 2 proposes a new thread partitioning algorithm. Section 3 shows several experimental results and evaluates effectiveness of the proposed algorithm. Section 4 gives concluding remarks.

2. THREAD PARTITIONING BASED ON LOCAL REGISTER FILE

2.1. Outline of Thread Partitioning

We propose a new thread partitioning algorithm to generate sub-threads which have waiting time. It is assumed that a circuit has large register files. The power consumption of the circuit occupies the greater portion of the power consumption by charge and discharge capacitance in the register files and the power consumption of the clock tree.

0-7803-8660-4/04/$20.00 ©2004 IEEE

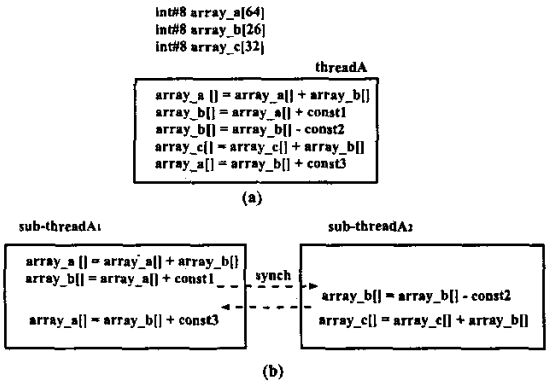

Fig. 1. A thread partitioning based on a local register. (a) The behavioral description before applying thread partitioning. (b) The behavioral description after applying thread partitioning in sub-threads.

We partition a thread based on a local register file for reducing this power consumption. When a thread is partitioned into two sub-threads, execution time of the sub-threads is almost equal to that of the thread. If we can reduce the power consumption without increasing execution time of the original circuits, we can generate low energy circuits compared to the original circuits.

First our thread partitioning algorithm focuses on a local register file RF in a thread. It partitions a thread into two sub-threads, one of which has RF and the other does not have RF. The power consumption of the outside registers of sub-threads can not be reduced by applying gated clocks to the sub-threads. Therefore the register file has to be generated in sub-threads to reduce the power consumption effectively. We partition the expressions in the thread into two categories, one of which has the variable assigned to RF and the other does not have it. By assigning the former to a sub-thread, RF can be generated in the sub-thread. When the other register files are used in the only one sub-thread, they are generated in the sub-thread. Applying gated clocks to sub-threads can reduce the power consumption of the register files generated in sub-threads.

In the Bach system, the array described in BachC language is assigned to a register file. Here we define the size of the register file as the product of the width and the number of the registers. In Figure 1, array_a[] is assigned to the 64 registers whose bit width is 8bits. We decide that the target register file RF is the largest register file in the thread. In Fig. 1, array_a[] is assigned the largest register file. Therefore the target RF is array_a[]. We partition the threadA into the sub-threadA_1 and the sub-threadA_2 based on array_a[]. Then the sub-threadA_1 has the target RF(array_a[]). The partitioned two sub-threads need to synchronize when the data dependency exists between the two sub-threads. Then the sub-threads may have waiting time. In Fig. 1, the data dependency exists between sub-threadA_1 and sub-threadA_2. Then sub-threadA_1 and A_2 have waiting time. Figure 2 shows the hardware architecture after partitioning the threadA. The array_a[] is only used in sub-threadA_1, and the array_c[] is only used in sub-threadA_2. Therefore array_a[] and array_c[] are generated in sub-threadA_1 and sub-threadA_2. Array_b[] is not included in any sub-threads, since it is used in both sub-threads.

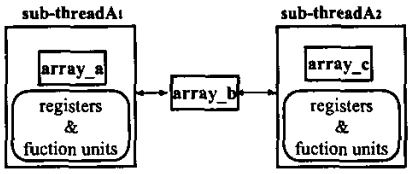

Fig. 2. The hardware architecture after partitioning the threadA.

Generally, the power reduction techniques by applying gated clocks depend on the input data. However applying gated clocks to threads does not depend on it. We partition a thread into two sub-threads. The partitioned sub-threads have waiting time, since they have synchronous communication. The waiting time does not depend on the input data.

2.2. Thread Partitioning Algorithm

In this section, we propose the thread partitioning algorithm based on the outline of the Sect. 2.1. The proposed algorithm consists of the two steps. In Step1, we partition a thread into two sub-threads based on RF. The one sub-thread has RF. In Step2, we insert the synchronous communication in the partitioned sub-threads to keep the data dependency of the original circuit.

The inputs of the proposed algorithm are the source code which is input into the high-level synthesis system and the target local register file RF. The designer decides that the target RF is the largest register file in the thread. The source code is a behavioral description and we can describe parallel behaving circuit blocks explicitly in it.

First, we find the function which is assigned a thread. We generate F1 and F2 each of which is the copy of the function (Step 1-1). F1 and F2 are executed in parallel. We define X as a set of variables which are assigned the target RF. The expressions including x($\in$ X) are executed in F1 and the expressions not including x are executed in F2(Step 1-2). Secondly, it is necessary to generate the synchronous communication to keep the data dependency between F1 and F2. Since F1 and F2 are executed in parallel, we obtain the incorrect result when the data dependency exists between F1 and F2. Therefore we insert the synchronous communication into the place where we removed the expressions in F1 and F2 (Step 2-1). In the original source code, the expressions which do not have the data dependency can be executed in parallel. In Step 2-1, we insert the synchronous communication into the sub-threads even if the sub-threads do not have the data dependency. Therefore F1 and F2 may have excessive

synchronous communication. Then we remove it(Step 2-2).

We define the data dependency between the two sub-threads. $E1_n$ and $E2_n$ denote a set of the expressions between the n-th synchronous communication and (n+1)th synchronous communication in F1 and F2 respectively. $V1_n\_l$ and $V2_n\_l$ denote a set of the variables of the left side of the expressions and $V1_n\_r$ and $V2_n\_r$ denote a set of the variables of the right side of the expressions in $E1_n$ and $E2_n$ respectively. It is assumed that an expression depends on another expression. The definitions are below,

$$
\begin{array}{lll}
{}^{\exists}v1_n\_l \in V1_n\_l, & & {}^{\exists}v2_{n+1}\_l \in V2_{n+1} \\
 & s.t. & v1_n\_l = v2_{n+1}\_l \ or \\
{}^{\exists}v2_n\_l \in V2_n\_l, & & {}^{\exists}v1_{n+1}\_l \in V1_{n+1}\_l \\
 & s.t. & v2_n\_l = v1_{n+1}\_l \ or \\
{}^{\exists}v1_n\_l \in V1_n\_l, & & {}^{\exists}v2_{n+1}\_r \in V2_{n+1}\_r \\
 & s.t. & v1_n\_l = v2_{n+1}\_r \ or \\
{}^{\exists}v2_n\_l \in V2_n\_l, & & {}^{\exists}v1_{n+1}\_r \in V1_{n+1}\_r \\
 & s.t. & v2_n\_l = v1_{n+1}\_r \ or \\
{}^{\exists}v1_n\_r \in V1_n\_r, & & {}^{\exists}v2_{n+1}\_l \in V2_{n+1}\_l \\
 & s.t. & v1_n\_r = v2_{n+1}\_l \ or \\
{}^{\exists}v2_n\_r \in V2_n\_r, & & {}^{\exists}v1_{n+1}\_l \in V1_{n+1}\_l \\
 & s.t. & v2_n\_r = v1_{n+1}\_l \ .
\end{array}
\quad (1)
$$

When Equation (1) is satisfied, the data dependency exists between F1 and F2. Then we can not remove the (n+1)th synchronous communication.

Finally, we create the function F which executes F1 and F2 in parallel. Then we can obtain the source code partitioned into the threads F1, F2 and F(Step 2-3,2-4).

Figure 3 shows the proposed algorithm.

3. EXPERIMENTAL RESULT

In this section, we verify the proposed algorithm which is shown in Fig. 3. We utilize a DCT, a Quantizer, a Huffman encoder and a JPEG encoder designed by the Bach system for verification. The JPEG encoder consists of the DCT, the Quantizer and the Huffman encoder designed by Bach system. We apply the gated clocks to the circuits adopted the proposed algorithm.

We use Synopsys Design Compiler as a logic-level synthesis tool, VDEC libraries (CMOS and 0.35μm technology) [1], and Synopsys DesignPower to estimate the power consumption in the simulation by Synopsys VSS. We use the logic-level synthesized values as the values of area and delay. Synopsys DesignPower estimates the values of power consumption with switching information which is obtained by the simulation. We decide that the input RF of the proposed algorithm are the register file of the largest size in the input source code. When there are more than one register file with the largest size, we select the register file which are more frequently used in the input source code.

[1]The libraries in this study have been developed in the chip fabrication program of VLSI Design and Education Center (VDEC), the University of Tokyo with the collaboration by Hitachi Ltd. and Dai Nippon Printing Corporation.

Inputs: the input source code of the high-level synthesis system and the local register file RF
Output: the input source code which has partitioned threads of the high-level synthesis system

Step 1-1. Find the function which is assigned to a thread and generate F1 and F2 each of which is the copy of the function.

Step 1-2. For all the expressions in F1, remove the expressions which do not include x($\in$ X) in F1. For all the expressions in F2, remove the expressions which include x($\in$ X) in F2.

Step 2-1. For F1 and F2 obtained in **Step 1-2**, insert the synchronous communication into the points where we removed expressions in F1 and F2.

Step 2-2. For all the synchronous communication inserted in the **Step 2-1**, if the expressions in $E1_n$ and $E2_n$ do not depend on the expressions in $E1_{n+1}$ and $E2_{n+1}$, remove the (n+1)th synchronous communication.

Step 2-3. Create the function F which executes F1 and F2 in parallel.

Step 2-4. Output the obtained F, F1 and F2, and exit.

Fig. 3. A thread partitioning algorithm.

Table 1 and 2 show the experiment results of the values of energy consumption, area, delay, execution time and active rate of threads before and after the proposed algorithm is applied to the circuits. The energy consumption E is given by

$$E = Clock \times T_{exe} \quad (2)$$

Where, $Clock$ is the clock cycle and T_{exe} is the execution time of the circuit. In the all circuits, We evaluate the energy consumption using the clock cycle 20MHz.

Table 1 shows the energy consumption when 1 block(the block size is 8bits $\times$ 8bits and the kind of images is 8bits binary format image) of a image is processed. The JPEG encoder has pipeline architecture. Figure 4 shows the behavior of the JPEG encoder. The *par* of Table 1 and 2 denotes the results when our proposed algorithm is applied, and The *non-par* denotes the results of the original circuit. The P of Table 1 denotes the energy consumption rate of reducing the energy consumption compared to the energy consumption of the original circuit. We partition the thread into two sub-threads. In Table 2, *active rate* denotes the rate of the time supplied clock signals to sub-threads per the execution time. The active rate of the JPEG encoder is the combination of that of each module.

Table 1. Experimental result of the energy consumption.

component	energy[μJ]		P[%]
	non-par	par	
DCT	4.19	2.72	64.92
Quantizer	2.82	2.01	71.28
Encoder	1.23	0.81	65.85
JPEG Encoder	8.24	5.54	67.23

Table 2. Synthesized result of the circuits.

component	area[mm^2]	delay[ns]	execution time[μs]	active rate[%]
DCT(non-par)	2.53	24.38	90.85	100
DCT(par)	2.68	41.01	80.25	74:40
Quantizer(non-par)	1.67	14.99	80.45	100
Quantizer(par)	1.70	33.93	103.15	19:87
Encoder(non-par)	0.96	10.69	60.90	100
Encoder(par)	0.95	28.32	53.45	15:86
JPEG Encoder(non-par)	5.17	33.93	272.55	***
JPEG Encoder(par)	5.32	41.01	309.45	***

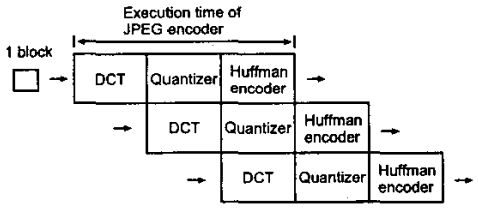

Fig. 4. Behavior of JPEG encoder.

Table 1 shows that we achieve in maximum 45% energy reduction in the modules. We achieve 33% energy reduction in the JPEG encoder. In the modules, the energy consumption is reduced effectively when the sub-threads have larger registers.

Table 2 shows our proposed algorithm obtains larger circuits than the original circuits in area. The number of expression does not increase in the behavioral description of the applications for the proposed algorithm. However the number of functional units increases by the proposed algorithm compared to the number of the original circuits, for example the functional units used in control (loop and branch) and the few functional units. The DCT and the Huffman encoder have less execution time compared to the original circuits, since they are executed in parallel. In contrast, the Quantizer has more execution time compared to the original circuits, since the synchronous communication is increased. The Huffman encoder obtained by the proposed algorithm has simple memory controllers compared to the original Huffman encoder. Therefore it has smaller area and less execution time compared to the original encoder. We achieve the most powerful energy reduction with low additional area and execution time overheard.

The experimental results show that the proposed algorithm obtains the sub-thread, which has low active rate, has the large size registers compared to the other. Consequently, the proposed algorithm obtains lower energy system LSIs.

4. CONCLUSIONS

In this paper, we proposed the thread partitioning algorithm in low energy high-level synthesis. This approach reduces the energy more efficiently with small additional area overhead when gated clocks are applied. Consequently, We achieve 33% energy reduction when we apply our proposed algorithm to a JPEG encoder. This approach enables the high-level synthesis system to synthesize low energy LSIs.

We intend to improve the decision of the target RF when the same largest size registers exist in the input source code in the future.

5. REFERENCES

[1] L. Benini and G. De Micheli, "Automatic synthesis of gated-clock finite-state machines," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 6, pp. 630–643, June 1996.

[2] J. M. Chang and M. Pedram, "Register allocation and binding for low power," in *Proc. 32nd DAC*, pp. 29–35, June 1995.

[3] T. Kambe, A. Yamada, K. Nishida, K. Okada, M. Ohnishi, "A C-based synthesis system, Bach, and its application," in *Proc. ASP-DAC2001*, pp. 151–155, 2001.

[4] A. Kumar and M. Bayoumi, "Novel formulations for low-power binding of function units in high-level synthesis," in *Proc. ICCD '99*, pp. 321–324, Oct. 1999.

[5] J. Monteiro, S. Devadas, P. Ashar, and A. Mauskar, "Scheduling techniques to enable power management," in *Proc. 33rd DAC*, pp. 349–352, June 1996.

[6] S. Noda, N. Togawa, M. Yanagisawa, and T. Ohtsuki, "High-level area/delay/power estimation for low power system VLSIs with gated clocks," *IEICE Trans. on Fundamentals*, vol. E85–A, pp. 827–834, April 2002.

[7] M. Ohnishi, R. Sakurai, K. Nishida, K. Okada, A. Yamada, T. Kambe, "A method of low power design for Bach system," in *Proc. IPSJ Design Automation Symposium 2001*, pp. 119–123, July 2001.

[8] K. Okada, A. Yamada and T. Kambe, "Hardware algorithm optimization using Bach C," *IEICE Trans. on Fundamentals*, vol. E85-A, no. 4, pp. 835–841, April 2002.

[9] G. Tellez, A. Farrahi, and M. Sarrafzadeh, "Activity driven clock design for low power circuits," in *Proc. ICCAD-95*, pp. 62–65, Nov. 1995.

[10] A. Yamada, K. Nishida, A. Kay, A. Yamada, T. Fujimoto, and T. Kambe, "A scheduling method for synchronous communication in the Bach hardware compiler," in *Proc. of ASP-DAC'99*, pp. 193–196, 1999.