A Thread Partitioning Algorithm in Low Power High-Level Synthesis

Jumpei Uchida[†] Nozomu Togawa[††,‡] Masao Yanagisawa[†] 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
3-4-1 Okubo, Shinjuku, Tokyo, 169-8555, Japan
Tel: +81-3-3209-3211(5716) Fax: +81-3-3204-4875
E-mail: uchida@yanagi.comm.waseda.ac.jp

Abstract

This paper proposes a thread partitioning algorithm in low power high-level synthesis. The algorithm is applied to high-level synthesis systems. In the systems, we can describe parallel behaving circuit blocks(threads) explicitly. First it 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 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. Then we can synthesize a low power circuit with a low area overhead, compared to the original circuit. Experimental results demonstrate effectiveness and efficiency of the algorithm.

1 Introduction

Recently, design complexity is highly increasing. At the same time, requirements for low power system VLSIs are also increasing for the needs of cellular phones, PDAs, and note PCs. We should develop high speed, small area and low power system VLSIs 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 power system VLSIs.

Several power reduction techniques at high-level synthesis were proposed. Gated clocks were exploited for power reduction techniques [1],[6],[10]. In [1], gated clocks were applied efficiently by reducing waiting time of logic circuits. In [6], area/delay/power estimation for low power system VLSIs with gated clocks was proposed. In [10], 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 high-level synthesis systems were developed. In the Bach system[3],[8], we can describe parallel behaving circuit blocks(threads) explicitly in BachC which is the input language of the Bach system. Each thread has waiting time for synchronization, since each thread has synchronous communication to keep the data dependency. The Bach system has a high-level power reduction technique. The technique is gated clocks which is applied to threads[7].

In this paper, we propose a thread partitioning algorithm in low power high-level synthesis. First the 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 partitioned two sub-threads need 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 sub-thread. Then we can synthesize a low power circuit with a low area overhead, compared to the original circuit.

This paper is organized as follows: Section 2 shows the gated clocks of the Bach system which is adopted in the proposed algorithm and the example for reducing power consumption. Section 3 proposes a thread partitioning algorithm. Section 4 shows several experimental results and evaluates effectiveness of the proposed algorithm. Section 5 gives concluding remarks.

2 Motivated example

2.1 Clock gating for threads

We apply gated clocks to the threads when threads have waiting time. We can use the gated clocks in the Bach system. We show the controller of the gated clocks in this section.

In the Bach system, threads have two types of waiting time.

(I) A thread waits until all the operations of the other threads finish.

(II) A thread waits until the handshake of synchronous communication is completed.

Figure 1(a) explicitly shows that the processes A and B are executed in parallel. The processes A and B consist of the sequential operations. The processes A and B are allocated to two threads (in Fig. 1(b)). The processes described by par construction start and finish at the same time. The synchronous communication (I) is generated at the start and the end of the threads . Suppose that the process A in the thread1 finish before the process B of the thread2 finish in Fig. 1. Then the thread1 waits until the process B of the thread2 finish. In Figure 2, the signal sy_req1 is high when all the operations of the thread1 finish. The signal sy_ok is

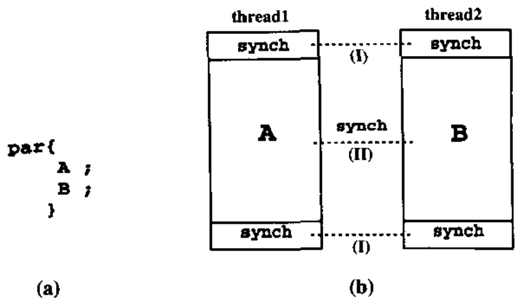


Figure 1. Synchronous communication between threads. (a) The par construction described by BachC. (b) The syncronous communication between the thread1 and the thread2.

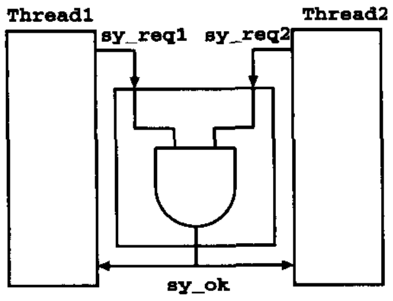


Figure 2. The circuit of the synchronous communication between threads.

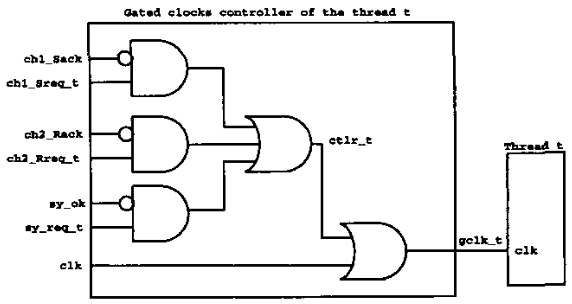


Figure 3. Gated clocks controller in the Bach system.

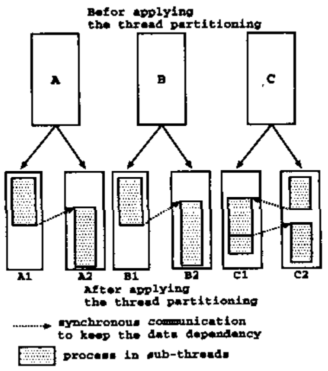

Befor applying the thread partitioning
A
B
C
A1
A2
B1
B2
C1
C2
After applying the thread partitioning
synchronous communication to keep the data dependency
process in sub-threads

Figure 4. Thread partitioning.

high when all the operations of the two threads finish. The synchronous communication (II) is need to keep the data dependency between the therads. Suppose that the data dependency exists between the processes A and B. Therefore the synchronous communication must be inserted in the thread1 and the thread2. The thread1 has waiting time, when the sender of the synchronous communication in the thread1 is ready before the receiver of the synchronous communication in the thread2 is ready.

Figure 3 shows the gate-level circuit model for applying gated clocks. Since the controller of the gated clocks consists of small logic gates, low power circuits are synthesized, with a low area overhead, compared to the original circuits to which gated clocks are not applied. In the case (I), the signal sy_req_t is high when the synchronous communication of par construction in the thread is ready, and the signal sy_ok is low until all the operations of the other threads finish. The case (II) is synchronous communication by the channel. In the case (II), the signal ch1_Sreq_t and ch2_Sreq_t are high when the synchronous communication of the channel 1 and 2 is ready, and the signal ch1_Sack_t and ch2_Rack_t are high until it is completed. Then the signal gclk_t is obtained by

$$gclk\_t <= ctrl\_t \text{ or } clk \quad (1)$$

Since the clock of each thread which waits for other threads is not supplied by applying gated clocks to the each thread, the power consumption of the circuit is reduced[7].

As deiscussed in Section 2.1, we show the example. The circuit consists of some threads. For example, the circuit consists of the threadA, the threadB, and the threadC in Figure 4. The threads are executed in parallel. Figure 5(a) shows the execution time of the threads before applying the thread partitioning. The threadC has the longest execution time than other threads and the threadB has longer execution time than the threadA. Then the threadA and the threadB have waiting time. In the case of Fig. 5(a), the power consumption of the circuit is reduced, since the clock of the each thread which waits for other threads is not supplied by applying gated clocks to the each thread.

We partition the each thread into two sub-threads in Fig. 4. For example, the threadA is partitoned into the sub-threadA1 and the sub-threadA2. Figure 5(b) shows the execution time of the threads after applying the thread partitioning algorithm. When the data dependency exists between sub-threads, the sub-threads need to be synchronized with the each other to keep the data dependency in Fig. 4. As showed in Fig. 5, they are not executed in parallel at all times. Suppose that the area overhead is low by partitioning the threads in Fig. 5(b). In Fig. 5, the waiting time of the threadA is 50ns. The threadA corresponds to the sub-threadA1 and the sub-threadA2. The average waiting time of the sub-threadA1 and the sub-threadA2 is (75+25+50)/2=75ns. In Fig. 5, the waiting time of the case (a) and the case

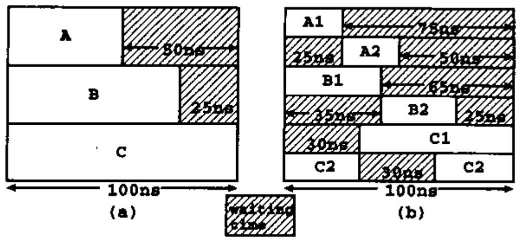


Figure 5. Execution time. (a)Before applying the thread partitioning. (b)After applying the thread partitioning.

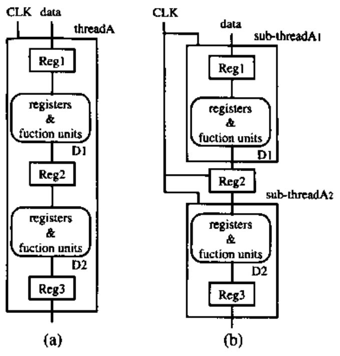

CLK
data
threadA
Regl
registers & fuction units
D1
Reg2
registers & fuction units
D2
Reg3
(a)
CLK
data
sub-threadA1
Regl
registers & fuction units
D1
Reg2
sub-threadA2
registers & fuction units
D2
Reg3
(b)

Figure 6. Example for reducing power consumption. (a) is the original circuit. (b) is the thread partitioned circuit.

(b) is

$$(a) : 50 + 25 = 75ns \quad (2)$$

$$(b) : (75 + 25 + 50 + 65 + 35 + 25 + 30 + 30)/2 = 162.5ns \quad (3)$$

$$75ns < 162.5ns \quad (4)$$

We apply gated clock to the sub-threads which wait for other sub-therads. When the total waiting time of Fig. 5(b) increases compared to that of Fig. 5(a), we can synthesize a low power circuit.

2.2 Power reduction by gated clocks and thread partitioning

We confirm that power consumption of a circuit can be reduced by partitioning a thread. Power consumption of a circuit can be classified into static power consumption and dynamic one. Static power consumption is the power consumption when the gates do not run. Dynamic power is the power consumption by the charge and the discharge of load capacitances in the gates. Generally, dynamic power consumption occupies the greater portion of the power consumption of the circuit than static power consumption. The purpose of this paper is to reduce dynamic power consumption.

Generally, dynamic power consumption P_{dyn} in a CMOS circuit is given by

$$P_{dyn} = V_{dd}{}^2 f C_{eff} = P_{CLK} + P_{gate} \quad (5)$$

Where, V_{dd} is the power supply voltage, f is the clock frequency, and C_{eff} is the effective capacitance. Let C_L be the load capacitance, and $p_{0\to1}$ be the probability of signal transition "0"→"1." Then $p_{0\to1}$ is equivalent to switching activity. Here the effective capacitance C_{eff} is obtained by $C_{eff} = C_L \times p_{0\to1}$[9]. P_{CLK} is the power consumption of the clock tree. P_{gate} is the power consumption of gates. The technique using gated clocks reduces the switching activity of registers.

Figure 6 shows an example for reducing dynamic power consumption. In Fig. 6(a), we suppose that the register file size of Regl, Reg2 and Reg3 are chosen arbitrarily, and the data path of D1 and D2 consist of registers and functional units. The execution time from Regl to Reg2 takes m clocks, and the execution time from Reg2 to Reg3 takes n clocks. In Fig. 6(a), D2 executes useless operations former m clocks, and D1 executes useless operations latter n clocks. Then D1 and D2 have useless power consumption. In Fig. 6(b), threadA is partitioned into the sub-threadA_1 and the sub-threadA_2. Since the data dependency exists between the sub-threadA_1 and the sub-threadA_2, the sub-threadA_1 and the sub-threadA_2 are not executed in parallel. Then the sub-threadA_2 has waiting time former m clocks, and the sub-threadA_1 has waiting time latter n clocks. The clock signals do not need to be supplied to the sub-thread which has waiting time. Then we can apply gated clocks in order not to supply clocks to sub-threads. Therefore we can obtain the low power circuit. The power consumption of a clock tree is directly proportional to the effective capacitance. The technique using gated clocks reduces the effective capacitance of the clock tree. Therefore the power consumption of the clock tree is reduced. By Equation (5), the power consumption $P_{Regl_{orig}}$ of Regl in the threadA and the power consumption $P_{Regl_{par}}$ of Regl in the sub-threadA_1 are

$$P_{Regl_{orig}} = V_{dd}{}^2 \cdot f \cdot C_{Regl} \cdot S_{Regl_{orig}} \quad (6)$$

$$P_{Regl_{par}} = V_{dd}{}^2 \cdot f \cdot C_{Regl} \cdot S_{Regl_{par}} \quad (7)$$

$$S_{Regl_{orig}} \geq S_{Regl_{par}} \quad (8)$$

where C_{Regl} is the load capacitance of Regl, $S_{Regl_{orig}}$ and $S_{Regl_{par}}$ are the switching activity of Regl before and after partitioning the thread respectively. The power consumption of Reg3, D1, and D2 is obtained the same as the power consumption of Regl is obtained by Equation (6), (7), (8). In Fig. 6, the gated clocks are not applied to Reg2, since Reg2 is the outside register file of the sub-threads. In Fig. 6(a) and (b), the switching activity of Reg2 is not changed before and after the thread is partitioned. Therefore the power consumption $P_{Reg2_{orig}}$ and $P_{Reg2_{par}}$ of Reg2 in Fig. 6(a) and (b) are

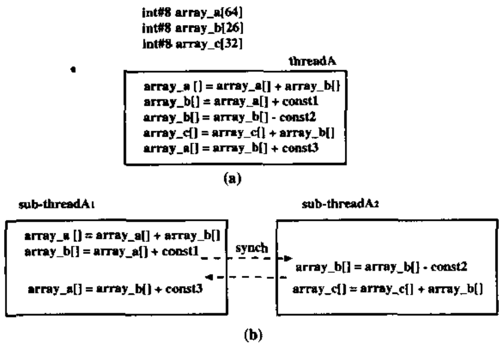
$$P_{Reg2_{orig}} = P_{Reg2_{par}} \quad (9)$$

Therefore the power consumption P_{orig} and P_{par} of the circuits in Fig. 6(a) and (b) are obtained by

$$P_{orig} = P_{Regl_{orig}} + P_{Reg2_{orig}} + P_{Reg3_{orig}} + P_{D1_{orig}} + P_{D2_{orig}} \quad (10)$$

$$P_{par} = P_{Regl_{par}} + P_{Reg2_{par}} + P_{Reg3_{par}} + P_{D1_{par}} + P_{D2_{par}} \quad (11)$$

$$P_{orig} \geq P_{par} \quad (12)$$



```
int#8 array_a[64]
int#8 array_b[26]
int#8 array_c[32]
```

```
array_a [] = array_a[] + array_b[]
array_b[] = array_a[] + const1
array_b[] = array_b[] - const2
array_c[] = array_c[] + array_b[]
array_a[] = array_b[] + const3
```

```
array_a [] = array_a[] + array_b[]
array_b[] = array_a[] + const1

array_a[] = array_b[] + const3
```

```
array_b[] = array_b[] - const2
array_c[] = array_c[] + array_b[]
```

Figure 7. 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.

The power consumption of the clock tree $P_{CLK_{orig}}$ and $P_{CLK_{par}}$ of the circuits in Fig. 6 are

$$P_{CLK_{orig}} = V_{dd}^2 \cdot f \cdot C_{orig} \quad (13)$$

$$P_{CLK_{par}} = V_{dd}^2 \cdot f \cdot C_{par} \quad (14)$$

Since the effective capacitance is reduced by applying gated clocks to the partitioned sub-threads, the effective capacitance C_{orig} and C_{par} of the clock tree in Fig. 6(a) and (b) are

$$C_{orig} \geq C_{par} \quad (15)$$

Then the power consumption of the clock tree is

$$P_{orig} \geq P_{par} \quad (16)$$

3 Thread Partitioning based on Local Register

3.1 Outline of Thread Partitioning

We propose a thread partitioning algorithm to generate sub-threads which have waiting time . Suppose 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. We partition a thread based on a local register file for reducing this power consumption.

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.

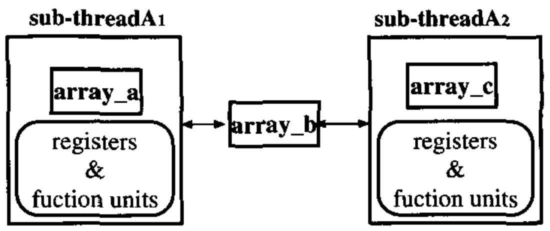


Figure 8. The hardware architecture after partitioning the threadA.

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 7, array_a[] is assigned to the 64 registers whose bit width is 8bit. We decide that the target register file RF is the largest register file in the thread. In Fig. 7, array_a[] is assigned the largest register file. Therefore 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 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. 7, the data dependency exists between sub-threadA_1 and sub-threadA_2. Then sub-threadA_1 and A_2 have waiting time. Figure 8 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 generated outside of the sub-threads, since it is used in the two sub-threads.

Generally, the power reducing 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.

3.2 Thread Partitioning Algorithm

In this section, we propose the thread partitioning algorithm based on the outline of the Sect. 3.1. 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 syncronous 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 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 RF. The expressions including x(∈ 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 dependecy 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. Suppose that an expression depends on another expression. The definitions are below,

$$\begin{aligned}
{}^{\exists}v1_n\_l &= v2_{n+1}\_l \ \ or \ \ {}^{\exists}v2_n\_l = v1_{n+1}\_l \ \ or\\
{}^{\exists}v1_n\_l &= v2_{n+1}\_r \ \ or \ \ {}^{\exists}v2_n\_l = v1_{n+1}\_r \ \ or\\
{}^{\exists}v1_n\_r &= v2_{n+1}\_l \ \ or \ \ {}^{\exists}v2_n\_r = v1_{n+1}\_l \ .\\
(v1_n\_l &\in V1_n\_l, \quad v1_{n+1}\_l \in V1_{n+1}\_l, \qquad (17)\\
v2_n\_l &\in V2_n\_l, \quad v2_{n+1}\_l \in V2_{n+1}\_l,\\
v1_n\_r &\in V1_n\_r, \quad v1_{n+1}\_r \in V1_{n+1}\_r\\
v2_n\_r &\in V2_n\_r, \quad v2_{n+1}\_r \in V2_{n+1}\_r)
\end{aligned}$$

When Equation (17) 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 9 shows the proposed algorithm.

(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 genarate 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(∈ X) in F1. For all the expressions in F2, remove the expressions which include x(∈ X) in F2.

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

Step 2-2. For all the syncronus 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 syncronous 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.

Figure 9. A thread partitioning algorithm.

4 Experimental Result

In this section, we verify the proposed algorithm which is shown in Fig. 9. We utilize a DCT, a Quantizer, a Huffman encoder and an IIR filter designed by the Bach system for verification. 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.

In table 1 and 2, we show the experiment results of the values of power consumption, area, delay, execution time and active rate of threads before and after the proposed algorithm is applied to the circuits. In Table 1, CLK net power denotes the power consumption of a clock tree, and except CLK net power denotes the power consumption of gates. P denotes the power consumption rate of reducing the power consumption compared to the power consumption of the original circuit. In Table 2, active rate denotes the rate of the time supplied clocks to sub-threads per the execution time. Table 3 shows the experimental results of the values of area and delay on the Quantizer before and after applying the proposed algorithm.

Table 1 shows that we achieve in maximum 45% and 42% power reduction at total power and CLK net power respectively. The power consumption of the clock tree is reduced more than that of the gates. Since the combinational circuits of the DCT and the Quantizer are larger than the other circuits, the power consumption of gates is reduced effectively by applying gated clocks to the sub-threads. Since the power consumption of the clock tree is directly proportional to the effective capacitance by Equation (5), the power consumption of the clock tree is reduced effectively when the sub-threads have larger registers.

[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.

Table 1. Experimental result of the power consumption.

component	CLK net power[mW]		P[%]	except CLK net power[mW]		P[%]	total power[mW]		P[%]
	non-par	par		non-par	par		non-par	par	
DCT	13.909	9.325	67.04	32.187	24.497	76.11	46.096	33.882	73.50
Quantizer	10.188	5.930	58.20	24.922	13.580	54.49	35.110	19.510	55.57
Encoder	6.605	4.857	73.54	13.537	11.990	88.57	20.142	15.165	75.29
IIR filter	0.941	0.618	65.67	2.213	2.165	97.83	3.154	2.783	88.23

Table 2. Synthesized result of the circuits

component	area[μm^2]	delay[ns]	execution time[ns]	active rate[%]
DCT(non-par)	2527393	24.38	90850	100
DCT(par)	2675176	41.01	80250	74:40
Quantizer(non-par)	1674794	14.99	80450	100
Quantizer(par)	1695054	33.93	103150	19:87
Encoder(non-par)	963727	10.69	60900	100
Encoder(par)	945421	28.32	53450	15:86
IIR filter(non-par)	167808	11.64	35200	100
IIR filter(par)	199306	26.67	40250	37:62

Table 3. Synthesized result of the Quantizer.

component	area[μm^2]	delay[ns]
Quantizer(non-par)	1674794	14.99
Quantizer(par)	1695054	33.93
sub-thread1	712611	33.93
sub-thread2	950456	33.93
outside reg(16bit*2)	18549	***
synch communication	13438	***

Table 2 shows that the sub-thread1 of the DCT has 74% active rate, and the sub-thread2 of it has 40% active rate. The one of the sub-threads, which has the lower active rate, has smaller local registers than the other sub-thread in the DCT, tne Huffman encoder and the IIR filter. However in the Quantizer, both sub-threads have the same size registers. Therefore the power consumption of the Quantizer is reduced effectively by applying gated clocks to the sub-threads which have large size registers. Table 3 shows that the sub-threads have same size registers in the Quantizer. Each sub-thread has the 64 registers whose bit width is 16bit. The outside registers of the sub-threads are smaller than the registers in the sub-threads. The area overhead of the circuits for synchronous communication is small. In the other circuits, the thread is partitioned similarly.

The proposed algorithm obtains larger circuits than the original circuits in area. The number of expression do not increase in the behavioral description of the applications for the proposed algorithm. However the number of functional units increase 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 and the IIR filter have 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. 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 power system VLSIs.

5 Conclusions

In this paper, we proposed the thread partitioning algorithm in low power high-level synthesis. This approach reduces the power more efficiently with a low area overhead when gated clocks are applied. Consequently, This approach enables the high-level synthesis system to synthesize low power VLSIs.

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

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," 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. Fundamentals, vol. E85-A, no. 4, pp. 835–841, April 2002.

[9] J. M. Rabaey, *Digital Integrated Circuits: A Design Perspective*, Prentice Hall, 1995.

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