Automated Design of Wave Pipelined Multiport Register Files

Kouji Takano†
chano@archi.is.tohoku.ac.jp

Takehito Sasaki†
take@archi.is.tohoku.ac.jp

Nobuyuki Oba‡‡
ooba@jp.ibm.com

Hiroaki Kobayashi†
koba@archi.is.tohoku.ac.jp

Tadao Nakamura†
tadao@archi.is.tohoku.ac.jp

†Graduate School of Information Sciences
Tohoku University
Sendai, Japan 980-0845
Tel: +81-22-217-7013
Fax: +81-22-216-8113

‡‡IBM Research
Tokyo Research Laboratory
IBM Japan Ltd.
Yamato, Japan 212-0001
Tel: +81-22-711-4735
Fax: +81-22-223-0964

Abstract—Recent high-performance microprocessors have two or more functional units (FUs) to exploit instruction-level parallelism. To make full use of this capability, multiport register files are generally used. However, conventional multiport register files need a considerable amount of hardware. This paper proposes a multiport register file scheme, which uses time-division multiplexing with wave pipelining in order to save the needed hardware resources. For adjusting propagation delay timings, we develop a tool which automatically inserts dummy buffers into combinatorial logic.

I. INTRODUCTION

Recent high-performance microprocessors, such as superscalar and VLIW processors, have two or more functional units (FUs) to execute multiple instructions in parallel. The FUs frequently access a register file and can even simultaneously access the same register file. If the register file does not have sufficient bandwidth to accommodate these accesses, this would cause the access conflict and result in the bottleneck in the system performance. To make full use of the execution capability of the multiple FUs, data must be quickly transferred from and to the register file. To do this, multiport register files are generally used in microprocessors [1].

Since RISC microprocessors have instruction pipeline capability, the register file is possibly accessed up to three times even in a single instruction. For example, [ADD R1, R2, R3], which is one of the typical arithmetic operations, reads data from registers 1 and 2, adds them, and writes the result to register 3. Therefore, the register file must have three ports to accommodate up to three simultaneous read and write requests issued by the FU. When a microprocessor has m FUs, up to 3m read and write requests can be issued to the register file per clock. Next-generation superscalar microprocessors will have four to eight FUs [2]. This fact implies that the register file should have 12 to 24 ports in order to make full use of the processing power of the FUs. It is not easy for a conventional register file to implement so many ports because the conventional one requires many multiplexers, three-state buffers, and a large amount of wiring; thus it would raise the hardware cost and slow down the operational speed [3].

To solve these problems, Capitanio et al. propose a partitioned register file [4]. The partitioned register file, however, requires special compiler assist, because it employs an idea on restricting the connectivity between the FUs and the register file blocks. For example, FU1 can access register file blocks 1 and 2 but cannot access register file block 3. FU2 can access register file blocks 2 and 3 but cannot access register file block 1.

To save the hardware cost for implementing a multiport register file, we have proposed a time-division pseudo-multiport register file scheme [5][6]. It uses a time-division multiplexing for providing wider data bandwidth between the FUs and the register file. The basic idea is that the register file and the multiplexer run N times as fast as the system clock to realize an N-multiport register file by using wave pipelining [7][8].

In a wave-pipelining circuit, there are two or more coherent waves propagating between latches, as shown in Fig. 1. In other words, the propagation delay through combinational circuits virtually gives the latch function. In the figure, N is three, and three waves propagate from latch 1 to latch 2.
The features of wave pipelining are as follows:

- As already stated, it improves the system throughput without increasing the number of latches.
- It reduces power dissipation, space, and delay.
- It can be applied to any digital pipeline systems.

To allow two or more waves to successively propagate between latches, wave pipelining requires the precise delay control. The delay control in the design flow consists of two steps, that is, rough and fine tunings. Rough tuning is performed by inserting dummy buffers for adjusting the propagation delay. Fine tuning changes the gate length of transistors and the output load in order to further adjust the propagation delay. It is true that the precise delay control is not an easy job for irregular-patterned digital circuits, such as a complicated combinatorial logic, but it is relatively easy for regular-patterned digital circuits, such as a register file [5][6]. In our previous work, we designed multiport register files based on conventional and proposed methods. The design of register files was first described in VHDL, and then was compiled into a gate-level design by a Mentor Graphics’ logic synthesizer. Rough tuning was applied to the gate-level design, and the number of gates for implementing them was compared.

To facilitate rough tuning, we develop a tool, named REDNET, which automatically inserts dummy buffers that adjust propagation delay timings. In this paper, we represent the algorithm of REDNET. The gate-level design tuned by using REDNET is translated into a transistor-level design by a Mentor Graphics’ technology mapping tool. With an analog simulator, we verify that coherent waves correctly propagate between latches. We evaluate the amount of required hardware resources in the number of transistors.

This paper is organized as follows. Section 2 shows the design model of the register file and a design flow using Mentor Graphics CAD and REDNET. Section 3 shows analog simulation results and estimates the hardware cost for implementing the register file. Section 4 offers some concluding remarks.

II. TIME-DIVISION PSEUDO-MULTIPORT REGISTER FILE

The time-division pseudo-multiport register file runs N times as fast as the system clock. The FU reads and writes data through a register port in a time-division manner. The wave pipeline technique is used for time-division multiplexing, which reduces the number of physical ports. If the register file runs N times as fast as the system clock, the number of ports is ideally reduced to 1/N. Since the cell layout and the data-path length are regular in the register file, wave pipelining is readily applicable to its design.

Fig. 2 shows the design model, which consists of one or more FUs, latches, a multiplexer, and a register file. The multiplexer switches the signals between the port and the FUs in a time-division manner. With the wave pipeline technique, the multiplexer and register file run N times as fast as the system clock.

Fig. 3 is a timing chart of the instruction pipeline and register accesses with respect to the system clock. For simplicity, the system is assumed to have only one FU. The register file runs three times as fast as the system clock, that is, N is three. In cycle 4, instruction 1 writes the result back to the register file. In the same cycle, instruction 3 reads two operands from the register file. During one system clock cycle, the register file serves three accesses issued by the FU.

In accordance with the model shown above, we designed the time-division pseudo-multiport register file using a high-level design language, VHDL. The source design code is compiled into a gate-level design with the
Fig. 3. Register access timings (N = 3)

AutoLogic II compiler provided in the Mentor Graphics CAD [9].

To adjust delays in the gate-level design, we develop a rough-tuning tool, named REDNET. It analyzes all the signal paths in the circuit and automatically inserts dummy buffers to minimize the signal propagation skew between latches. Fig. 4 shows the flow chart of REDNET. In the chart, \( D_{gate} \) is propagation delay of a gate, and \( D_{dummy_buffer} \) is propagation delay of a dummy buffer. \( D \) is a delay stamp that represents accumulative delay from a signal input of the circuit to the gate. The complexity of the algorithm can be represented as \( O(E) \), where \( E \) is the number of nets in a circuit.

Let us explain how the REDNET works using an example. Fig. 5(a) shows a circuit consisting of four gates. At the initial state, no net is assigned a delay stamp. The delays of gates \( G_1, G_2 \) and \( G_4 \) are assumed to be 2, and that of gate \( G_3 \) is assumed to be 1. The delay of a dummy buffer is assumed to be 2. A dummy buffer is implemented with two cascaded inverters. The REDNET starts with the output net, \( N_9 \). Gate \( G_4 \) has two inputs, nets \( N_3 \) and \( N_7 \). Net \( N_3 \) is connected to the output net of gate \( G_1 \) and net \( N_7 \) is connected to the output net of gate \( G_3 \) (Fig. 5(b)). Since neither of two input nets of gate \( G_4 \) has a delay stamp yet, the algorithm is recursively applied to gate \( G_1 \). Gate \( G_1 \) has two input nets, \( N_1 \) and \( N_2 \), both of which are inputs of the circuit, and therefore they are marked with delay stamp 0, as shown in Fig. 5(c).

Since the delay of gate \( G_1 \) is 2, the delay stamp of the output net \( N_4 \) is set to 2. Consequently, delay stamp 2 is assigned to input net \( N_2 \). However, gate \( G_4 \) has another input net \( N_7 \), which has no delay stamp yet (Fig. 5(d)). Net \( N_7 \) is connected to the output of gate \( G_3 \). Gate \( G_3 \) has an input, net \( N_6 \), which has no delay stamp yet, so the REDNET has to trace back further. Gate \( G_2 \) has two input nets, \( N_4 \) and \( N_5 \). Net \( N_5 \) is connected to the output net of gate \( G_1 \) and net \( N_4 \) is directly connected to the logic input that has delay stamp 0 (Fig. 5(e)). Since net \( N_5 \) already has delay stamp 2, one dummy buffer is inserted in the net that is directly connected to the logic input, as shown in Fig. 5(f). The delay stamp of the output of gate \( G_2 \), that is, net \( N_6 \) is now set to 4.

In the same manner, the output of gate \( G_3 \) is assigned delay stamp 5 (Fig. 5(g)). Now, the delay stamps of the input nets of gate \( G_4 \) are settled (Fig. 5(h)). Nests \( N_4 \) and \( N_7 \) have delay stamps 2 and 5, respectively. Thus, we insert one dummy buffer in net \( N_4 \), as shown in Fig. 5(i). It should be noted that the delay stamps of the gate \( G_4 \)’s input nets are not the same, that is to say, 4 and 5, even by inserting dummy buffers. It is not possible for the REDNET to adjust this skew by inserting dummy buffers. It is responsible for fine tuning to adjust the skew.

The net list including dummy buffers are then translated to the transistor-level design with the AutoCells, which is also provided in the Mentor Graphics CAD. To adjust delays in the transistor level design, we are developing a fine tuning tool, named FITNET. The FITNET is now under development. Fig. 6 summarizes the design flow.
III. Hardware resource estimation

We have designed three-, six-, nine-, and twelve-port register files based on the conventional and proposed design methods. It should be noted that we have performed rough tuning and inserts dummy buffers to the net-list level design but have not performed fine tuning.

Fig. 7 shows the number of transistors needed for implementing conventional multiport register files and time-division pseudo-multiport register files with three, six, nine, and twelve ports. In the figure, TDFMRF stands for a time-division pseudo-multiport register file generated by using the wave pipeline technique. In the case of the six-port register file, the proposed design requires less than 50% of transistors for implementing the conventional multiport register file. In the cases of nine- and twelve-port register files, the proposed design requires about 25% of transistors. It is observed that the hardware resource reduction in the number of transistors becomes more significant as the number of ports increases.

For rough tuning, dummy buffers were inserted to the combinatorial logic. However, the number of transistors used for dummy buffers is 3.2% of the total number of transistors even in the worst case. This is due to the fact that the cell layout and the data-path length are very regular in the register file and multiplexers.

The results of the experimental design show that the time-division pseudo-multiport register files can be implemented with many fewer hardware resources than conventional register files. Although the proposed design needs further study for production, it is advantageous for the speed up and cost reduction of microprocessors.

To see how the waves propagate in the circuit, we did an analog timing simulation with Mentor’s LSIM. Fig. 8 shows a result of the simulation for six consecutive read
cycles. During one system clock, the register output data changes three times.

At time 326, the first read request signal (Acc1) begins to propagate into the circuit. The register file outputs four bits of data, “1011,” and the data is latched in the port 1’s latch at time 356. At time 342, the second read request signal (Acc2) propagates in the circuit, and the register file outputs “1101,” and the data is latched in the port 2’s latch at time 372. In the same way, the third read request signal (Acc3) propagates in the circuit at time 358. The register file outputs “1111,” and the data is latched in the port 3’s latch at time 386.

As can be seen in the figure, on the basis of the wave pipelining technique, the access to port 2 (Acc2) starts at time 342 before the output data from port 1 becomes valid at time 358. In a similar manner, the access to port 3 (Acc3) starts at time 358 before the output data from port 2 becomes valid at time 372. All the output data from ports 1, 2, and 3 become valid by the rising edge of the system clock at time 388. This can be done by rough tuning, which makes all the data paths in the circuit have almost the same delay, and thus allows multiple coherent waves to propagate through the circuit. For this particular example, two coherent waves propagate through the circuit.

Since the cell layout and the data-path length are very regular in the register file and multiplexers, the register file behaves in the wave pipelining manner with only rough tuning. However, rough tuning does not adjust the rising and falling time of gates or the data-path length. For this reason, small delay skew is still accumulated through the circuit and exposed as the invalid portion of the output data. To propagate more coherent waves for improving the throughput, the circuit should be designed to keep skew as small as possible. To this end, fine tuning is needed.

Fig. 9 is a microphotograph of the three-port TDPFMR.

IV. CONCLUDING REMARKS AND FUTURE WORK

Multiport register files are essential for making full use of the processing power of instruction-level parallel execution. In this paper, we have proposed a design scheme for multiport register files. It uses time-division multiplexing with wave pipelining to make it possible that, with a small increase of hardware resources, the register file can serve simultaneous read and write requests issued by functional units. On the basis of the proposed design scheme, we have designed three-, six-, nine-, and twelve-port register files using a high-level design language, VHDL, and have estimated the operational speed and the amount of hardware. We have shown that the register files designed with the proposed scheme require many fewer hardware resources needed for a conventional register file. For precise delay control, we have developed a rough tuning tool, REDNET. For further delay control, we are developing a fine tuning tool, FITNET. We also studying production process management scheme to keep delay skew as small as possible.

REFERENCES


Fig. 8. A result of analog timing simulation


