Low power register files by eliminating redundant read

Leiou Wang\textsuperscript{1,2a}, Donghui Wang\textsuperscript{1}, Huan Ying\textsuperscript{1,2}, and Chengpeng Hao\textsuperscript{1}

\textsuperscript{1} Institute of Acoustics, Chinese Academy of Sciences, 21 North 4th Ring Road, Haidian District, Beijing, 100190, China
\textsuperscript{2} University of Chinese Academy of Sciences, No.19A Yuquan Road, Beijing, 100049, China
\textsuperscript{a} wangleiou@mail.ioa.ac.cn

Abstract: Since register files consume a large portion of power in microprocessors, therefore, this paper proposes that a power saving method in register files can be implemented by eliminating branch instruction redundant read. Experimental results from running various application programs show an average of 8.89\% reduction in dynamic switching power, with negligible area overheads.

Keywords: low power, redundant read, register files

Classification: Integrated circuits

References

1 Introduction

Register files represent a substantial portion of the power budget in modern microprocessors. For example, in the Motorolas M.CORE architecture, the register files consumes 16% of the total processor power and 42% of the data path power [1]. Azavedo et al. [2] observed that the register files power may reach 25% of the total processor power consumption when running embedded applications. So many techniques have been proposed to reduce the power consumption of register files in designing power-efficient microprocessors. However, all these techniques tend to have a more complicated structure, resulting in greater area overhead.

This paper proposes one method to reduce number of access to register files by eliminating branch instruction redundant read. Take a simple program, shown in Fig. 1, as a motivational example. The left C language code performs a sum function from 0 to 99, and the right is the corresponding assembly code. Instruction 4, 5, 6, 7 are a loop and it will be repeatedly executed 100 times. As can be seen, the second operand value of branch instruction 4 is always unchanged (i.e., loop number d3 = 100).

Therefore, if the decode stage logic detects this branch instruction redundant read, we can save this operand value into a special register. And then this source operand can be read from the special register instead of being read from the register files. As a result, the number of access to register files can be effectively reduced.

The rest of this paper is organized as follows. Section 2 provides a review of previous low power register files techniques. Section 3 presents a model to estimate register files power consumption and proposes the Eliminating Branch Instruction Redundant Read (EBIRR) method. Section 4 presents experimental results to verify the proposed method. Finally, the paper is concluded in section 5.

2 Related work

This section will review various techniques in low power register files.

In [3] Balakrishnan et al. proposed Physical Register Reuse, which eliminates all duplication of values in the physical register files. But this method requires significant micro-architectural support and is quite complicated.
In order to reduce the number of access to register files, Trivial Computations (TC) was used in [4] by Tran et al. TC are those computations whose output can be determined without performing the computation. In some cases, e.g., multiplying by zero, we do not read both operands to compute the result.

Since a significant fraction of the result values are delivered to their consumers via the bypass network and never read out from register files. In [5], Balkan et al. proposed a technique to avoid the write back of such transient values into register files. Nevertheless, this technique needs some additional conditions, e.g., there must be at most one instruction that consumes the result values.

In order to reduce register files power and complexity, in [6] Sirsi et al. focused on reducing the number of register files read ports.

In [7] Han et al. proposed a low power register file by non-full-swing technique. But this method is mainly applied to circuit-level.

An asynchronously controlled read-isolation circuit was applied to prevent unnecessary register files read access in [8]. However, this technique also requires the compiler to support.

3 EBIRR

As mentioned before, the existing low power register files approaches usually need to greatly change the micro-architecture or require the compiler to support. However, we describe a structure-level full-hardware implementation in the following discussion. Surprisingly, this method is quite simple and relies on detecting branch instructions.

This section first presents a model to estimate register files power consumption. Then we propose the detailed algorithm and depict the hardware implementation.

3.1 Power consumption model

The power consumption model can be characterized as a function of the number of access to register files. For an M-read ports and N-write ports register files, the power consumption can be expressed by the following formula:

\[ P_{rf} = P_{rd} + P_{wr} = \sum_{i=0}^{M-1} (Ren[i] \times P_{read}) + \sum_{j=0}^{N-1} (Wen[j] \times P_{write}) \]  (1)

\( P_{rd} \): The read power consumption of register files. \( P_{wr} \): The write power consumption of register files. \( Ren[i] \): the read enable number of port i. \( Wen[j] \): the write enable number of port j. \( P_{read} \): the power consumption of a read operation. \( P_{write} \): the power consumption of a write operation.

The EBIRR focuses on optimizing \( P_{rd} \) by reducing the read enable number (Ren). Let us reconsider the example shown in Fig. 1. In fact, the branch instruction 4 is a part of the loop program. Since generally 10% of the loop programs occupy 90% of the execution time [9], this suggests that a significant reduction in the Ren can be achieved by using this method.
3.2 The EBIRR algorithm

The EBIRR algorithm is shown in the Table I.

<table>
<thead>
<tr>
<th>Table I. EBIRR algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initialization</strong>: all register is reset;</td>
</tr>
<tr>
<td>if (the instruction is not a branch instruction) branch instruction identify flag = 0; /<em>the EBIRR circuit does not work</em>/;</td>
</tr>
<tr>
<td>else /<em>branch instruction identify flag = 1</em>/ compare result = compare {second operand address, AddrReg}</td>
</tr>
<tr>
<td>if (compare result is unequal)</td>
</tr>
<tr>
<td>miss = 1, hit = 0; wren@dc = 1, rden@dc = 0; /<em>in the decode stage</em>/</td>
</tr>
<tr>
<td>ren[i] = 1, Ren[i] = Ren[i] + ren[i] = Ren[i] + 1;</td>
</tr>
<tr>
<td>AddrReg = second operand address;</td>
</tr>
<tr>
<td>wren@rf = 1, rden@rf = 0; /<em>in the read register files stage</em>/</td>
</tr>
<tr>
<td>LoopCountReg = second operand value;</td>
</tr>
<tr>
<td>final output = second operand value;</td>
</tr>
<tr>
<td>else /<em>compare result is equal</em>/</td>
</tr>
<tr>
<td>miss = 0, hit = 1; wren@dc = 0, rden@dc = 1; /<em>in the decode stage</em>/</td>
</tr>
<tr>
<td>ren[i] = 0, Ren[i] = Ren[i]/<em>power savings</em>/;</td>
</tr>
<tr>
<td>wren@rf = 0, rden@rf = 1;/<em>in the read register files stage</em>/</td>
</tr>
<tr>
<td>final output = LoopCountReg;</td>
</tr>
</tbody>
</table>

In EBIRR method, when there is a branch instruction, the branch instruction identify flag (i.e., biif) is set to one. And the EBIRR circuit begins to work. At this time, if the compare result between the second operand address and AddrReg is unequal, the ren[i] is one and the AddrReg will be updated by the second operand address. In the read register files stage, the second operand value reading from register files will be saved to the LoopCountReg and be selected as the final output.

If the compare result is equal, it means that there is a hit and the second operand value of branch instructions (i.e., loop number) has been already saved in the LoopCountReg. Therefore, the ren[i] is zero and this branch instruction redundant read can be eliminated. And then the LoopCountReg will be selected as the final output in the read register files stage.

3.3 The EBIRR implementation

The EBIRR circuit is shown in the Fig. 2. The whole circuit only needs two registers, two multiplexers, two and gates and one comparator. And this circuit working process is already specified in the above subsection.

In order to further reduce the EBIRR circuit area overheads, both the AddrReg and the LoopCountReg do not need a full-width register (e.g., 32-bits). For a 32*32-bits register files, the AdderReg is just a 5-bits register. Similarly, since the loop number usually do not tends to be a very large number, e.g., in the motivational example d3 = 100, a 7-bits LoopCountReg is enough.
4 Experiment results

To evaluate the proposed method, some classic DSP benchmarks [10] are utilized. The EBIRR method is evaluated on gate-level with the power analysis tool (Prime Time PX) based on the switching activity. The gate-level netlist is generated by the synthesis tool with the technology library of SIMC 90 nm process. In addition, this low power EBIRR method is applied to the SuperV DSP, which has a $32 \times 32$-bits, 7-read ports and 4-write ports register files. In order to prove the effectiveness of this method, the TC is also accomplished in our experiment. The simulation results can be seen in Fig. 3 and Fig. 4.

Fig. 3 shows the hit rates for EBIRR, and the main observation is that using our method is sufficient to capture roughly 95% of the branch instruction redundant read.

As can be seen in Fig. 4, the EBIRR power savings in the register files varies from application to application, ranging from 7.73% to 9.76%. On the average, the power savings are 8.89% without performance loss and the additional logic is accounted for.

In terms of area and delay overheads, as shown in the Table II, the EBIRR has the lower overheads. In contrast, the TC requires adding a zero tag register and needs more control logic.
5 Conclusions

This paper introduced the notion of the branch instruction redundant read. Based on this observation, we presented EBIRR — a full-hardware micro-architectural method that eliminates redundant read and obtains power benefits compared to the traditional register files. The simulation results show an average of 8.89% reduction in dynamic switching power, with negligible area overheads.

The method proposed in this paper incurs lower area and delay overheads in comparison with the existing techniques. Therefore, it is highly feasible and efficient.

In order to further reduce power consumption, we are planning to combine the EBIRR with reducing the number of read ports in the register files.

Acknowledgments

Project supported by the Knowledge Innovation Program of Chinese Academy of Sciences.