Decomposition of Instruction Decoder for Low Power Design

Wu-An Kuo, TingTing Hwang, and Allen C.-H. Wu
Computer Science Department, Tsing Hua University, Hsin-Chu, Taiwan 30043

Abstract

Microprocessors have been used in wide-ranged applications. During the execution of instructions, instruction decoding is a major task for identifying instructions and generating control signals for data-paths. By exploiting program behaviors, we propose a novel instruction-decoding approach for power minimization. Using the proposed instruction-decoding structure, we present a partitioning method that decomposes the instruction-decoding circuit into two sub-circuits according to the execution frequencies of instructions. Using our proposed decoding structure, only one sub-circuit will be activated when executing an instruction. Experimental results have demonstrated that our proposed approach achieves on an average of 26.71% and 15.69% power reductions for the instruction decoder and the control unit, respectively.

1. Introduction

Figure 1 illustrates the profiling result of the instruction execution-frequency by executing the benchmark set of Motorola's *Powerstone* that contains a set of benchmark programs targeted to various applications. Figure 1 shows that the three instructions in the *MOV* class have been executed very frequently (22%), while some instructions have never been executed. This observation shows that active instructions occur only within a sub-set of all instructions. Intuitively, we can partition the instruction-decoding circuit into two sub-circuits, one for the three *MOV* instructions and the other for the rest of the instructions. In this case, we can turn off one instruction-decoding sub-circuit while executing the other one and hence reduce the overall power dissipation.

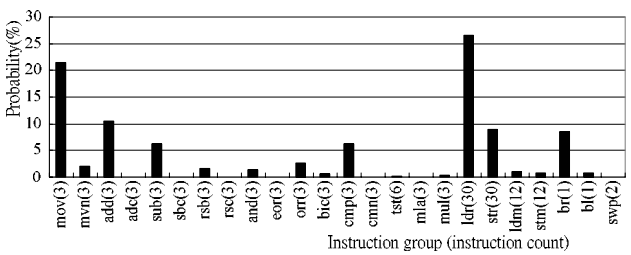

Figure 1: An instruction execution-frequency example.

2. The Decomposed-Decoder Model

In this section, we present our proposed decomposed-decoder model. In section 2.1, we describe the instruction-decoding method. In section 2.2, we present the low-power decoding architecture.

2.1. Insertion of Intermediate Code

One approach to decode instructions is to decode instructions *just in time*. That is, the control signals are decoded only when they are needed in the next pipeline stage. Instead of decoding all the control signals and propagating them along the pipeline, the instruction information is carried along. However, the instruction opcodes are not in the same position for different instruction class of processors. Take the *ARM7tdmi* instruction set as an example, we need to store 16 bits to identify the whole set. Since there are only 142 instructions, we need only 8 bits to represent all instruction types. Therefore, at decoding stage, when an instruction type is identified, an intermediate code is introduced to represent the specific instruction. Figure 2(a) shows the block diagram at the decoding stage, where an instruction is decoded by the *Instr-Decoder* (*ID*) and the output is an intermediate code. The generated intermediate code is used to generate control signals for the decoding stage and the next execution stages. The intermediate code is stored in the *Instruction State Registers* (*ISR*), which will be used to control the rest of pipelined stages.

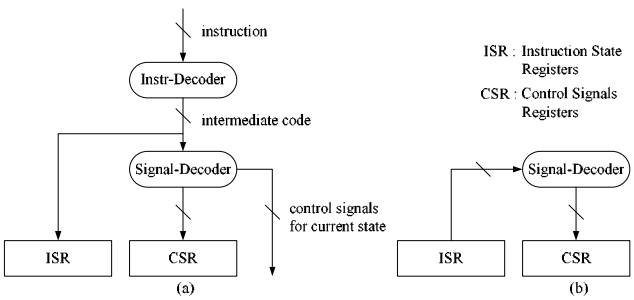

Figure 2: The block diagram of a control path (a) decoder at decoding stages (b) decoder at other execution stages.

2.2. Decomposition Model for Low Power

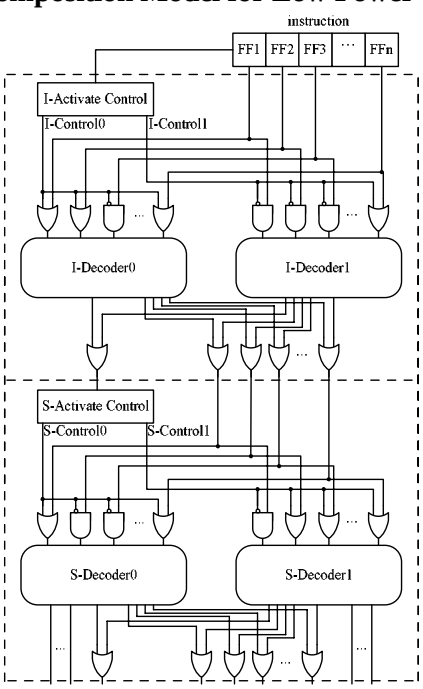

Figure 3: The decomposed-decoder model.

As discussed in Section 1, active instructions occur only within a subset of all instructions. Based on this observation, we propose a decomposed-decoding method that decomposes an instruction decoder into a number of coupled sub-decoders. Figure 3 shows that the decoder is decomposed into two coupled sub-decoders where the *Instr-Decoder* is decomposed into $I\text{-}Decoder_0$ and $I\text{-}Decoder_1$. At any moment, only one sub-decoder is active. The control logic to turn on/off sub-decoders consists of *Activate-Control*, *input AND-ORs*, and

1530-1591/04 $20.00 (c) 2004 IEEE

output ORs. *I-Activate-Control* determines which sub-decoder is activated by decoding the input instruction. There are two output signals, $I\text{-}Control_0$ and $I\text{-}Control_1$. When $I\text{-}Control_0 = 0$, $I\text{-}Decoder_0$ is on and $I\text{-}Decoder_1$ is off; when $I\text{-}Control_1 = 0$, $I\text{-}Decoder_1$ is on and $I\text{-}Decoder_0$ is off.

The next question is how to construct *AND-OR* gates in front of the sub *I-Decoders* to inhibit the propagation of inputs. It is constructed by selecting a *minterm* that is *don't care* to the sub-circuit. For example, for a four-input sub-circuit with 1101 as *don't care*, we construct *OR-OR-AND-OR* gates in front of the sub-circuit. For this input combination 1101, we assign the output of the sub-circuit all 0's. We can decompose the *Signal-Decoder* in the same way.

The questions remained to be solved are (1) how to decompose the decoder so that the instruction decoding has a high probability to be executed in a sub-circuit? (2) How to take the overhead of *I-Activate Control* for *Instr-Decoder* (decoder that determines the active sub *I-Decoders*) into consideration when performing the decomposition?

3. Decomposition of Decoder

In this section we present the decoder-decomposition method. In Section 3.1, we discuss how to decompose the *Instr-Decoder*. In Section 3.2, we present the decomposition method for the *Signal-Decoder*.

3.1. Decomposition of *Instr-Decoder*

We propose a method to use only one bit to partition all instructions. First, for the first bit of the instruction opcode, we sum up the execution frequencies of instructions with 0 and 1 value of the bit, respectively. We repeat the computations for all bits in the instruction opcode. Then, we select the bit that has the most un-even summation of instruction-execution frequency for 0 and 1 value. For example, we consider to partition instruction using two bits, bit 27 or bit 26. If bit 27 is selected to partition instructions, instructions with bit 27 = 0 are assigned to $group_0$ and the others to $group_1$. By summing each execution probability in each group, Figure 4(a) shows the result of selecting bit 27 as the partition bit. The bit 27 will be assigned to 0 and 1 for the instructions with the execution probabilities of 80% and 20%, respectively. On the other hand, if bit 26 is used, the bit will be assigned to 0 and 1 for the instructions with the execution probabilities of 90% and 10%, as shown in Figure 4(b). We can implement this instruction partitioning approach simply using an inverter in the *I-Activate Control* circuit.

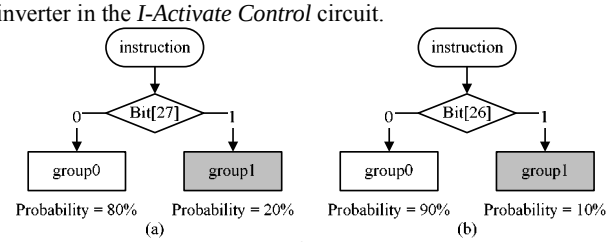

Figure 4: An example of partitioning-bit selection.

3.2. Decomposition of *Signal-Decoder*

In this section, we present the *Signal-Decoder* decomposition method that consists of two steps: (1) initial partitioning and (2) iterative improvement. In the first step, based on the instruction execution frequency, we partition instructions into two groups. Instructions with execution frequency higher than a probability are assigned to $group_0$ (highly active sub-circuit) and instructions with execution frequency lower than the probability to $group_1$.

In the second step, we iteratively move instructions between groups to improve the execution probabilities. We first define the dominance relations of two instructions. We define that instruction *I1* dominates instruction *I2*, if for all output signals O_i, O_i of *I2* is 1 if O_i of *I1* is 1. Our iterative improvement procedure performs as follows. For each instruction in $group_0$, we check whether there is an instruction in $group_1$ that can be dominated by it. If there is one, we move this instruction form $group_1$ to $group_0$.

4. Experimental Results

We compared the power dissipation using the original control circuit and our proposed control circuit targeted to *Powerstone* benchmark set. The circuits are described in *Verilog*. The final circuits were generated by the *Synopsys Design Compiler* using the *TSMC* 0.25um cell library. The area data of control circuits were reported by *Design Compiler* with *report_area*.

Table 2 shows the area comparison on the program set when selecting 0.01 as partitioning probability.. The area of partitioned circuit is the sum of the areas of activated controls, turn-on/off control logics and sub-decoders. The areas of the partitioned decoding circuits are 11.7% larger than the original decoding circuit. Let the area of the rest of control unit be α. C_Original and C_Partitioned are computed as α + D_Original and α + D_Partitioned, respectively. Our decoding structure can also improve the critical path timing of instruction decoding and control-signal generation with small area overheads.

Table 1: Area overhead of partitioned decoder (in decoding circuit control unit) compared to the original processor. (OH presents overhead.)

	D_Original	C_Original	D_Partitioned	OH	C_Partitioned	OH
Area	30387	65975	33942	11.7%	70138	6.31%

We evaluated the effectiveness of power reduction on the *Powerstone* benchmarking set using our proposed instruction decoding method. We used *Synopsys PrimePower* to compute the power dissipation (both dynamic and leakage power dissipation). These programs are simulated with our *ARM Verilog* code at the speed of 20MHz. Then the switching activities are fed into *PrimePower* to calculate power consumption. Table 3 shows the average results on the program set when selecting 0.01 as partitioning probability. The results show that our proposed instruction-decoding method achieves on an average of 26.71% in power reduction compared to the original instruction decoder and 15.69% improvement on the control unit after applying our decoding structure.

Table 2: Power comparisons on the Powerstone set. (In the table, ID denotes the instruction decoder, CU the control unit and Imp the improvement.)

Bench	Original		After partitioned			
	ID(W)	CU(W)	ID(W)	Imp(%)	CU(W)	Imp(%)
average	4.15E-4	1.03E-3	3.04E-4	26.71	8.69E-4	15.69

References

[1] Kalambur, A. and M. J. Irwin, "An Extended Addressing Mode for Low Power", Int. Symp. On Low Power Electronics and Design, 1997.

[2] Alidina, M., Monteiro, J., Devadas, S., Ghosh, A., and Papaefthymiou, M., "Precomputation-based sequential Logic Optimization for Low Power", Proceedings of ICCAD-94, pp. 74-81, 1994.

[3] R. S. Bajwa, M. Hiraki, G. Kojima, D. J. Gorny, K. Nitta, A. Shridhar, K. Seki, and K. Sasaki, "Instruction buffering to reduce power in processors for signal processing", IEEE Transactions on VLSI, vol.5 no.4, pp. 417-424, 1997.

[4] Intel, "PentiumPro Family Developer's Manual Volume 2: Programmer's Reference Manual", December 1995.