國立交通大學

資訊科學與工程研究所

碩 士 論 文

在無串擾位元變化編碼之指令匯流排中利用暫存器
重新標記以減少匯流排的耗電

Power Reduction by Register Relabeling for
Crosstalk-Toggling-Free-Coded Instruction Bus

研 究 生：林均翰

指導教授：單智君 教授

中 華 民 國 九 十 九 年 十一 月

在無串擾位元變化編碼之指令匯流排中利用暫存器重新標記以減少匯流排的耗電

Power Reduction by Register Relabeling for Crosstalk-Toggling-Free-Coded Instruction Bus

研 究 生：林均翰　　Student：Chun-Han Lin

指導教授：單智君　　Advisor：Jean Jyh-Jiun Shann

國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science and Engineering

Novermember 2010

Hsinchu, Taiwan, Republic of China

中華民國 九 十 九 年 十一 月

在無串擾位元變化之指令匯流排的耗電中利用暫存器重新標記以減少匯流排的耗電

學生：林均翰　　指導教授：單智君　博士

國立交通大學資訊科學與工程研究所碩士班

摘要

隨著製程進步至深亞微米級 (deep submicron level)，crosstalk 在深亞微米級製程之匯流排中的影響越受重視。而當兩條鄰近匯流線上的訊號轉換方向相反時，稱之為 crosstalk-toggling transitions，所帶來的影響不只是更多的耗電，也帶來更長的傳送延遲。現有相當多的研究是在同步電路設計 (synchronous circuit designs) 中，利用編碼方式使匯流排上能夠完全排除串擾位元變化轉換來達到減少傳送延遲。對於其耗電則仍保有進一步降低的機會。

這篇論文主要是研究在無串擾位元變化的 Selective Shielding 匯流排編碼方法下，利用暫存器重新標記 (register relabeling)進一步降低指令匯流排之耗電。在不需要增加額外硬體需求及沒有效能損失的前提下，我們的設計是未編碼指令匯流排平均耗電的 95.3%。此外，與 Selective Shielding 方法相較，在指令匯流排上可進一步減少 12.1%的耗電。整體而言，我們的設計可保持原 Selective Shielding 方法無串音位元變化之特性並減少其耗電。

Power Reduction by Register Relabeling for

Crosstalk-Toggling-Free-Coded Instruction Bus

Student：Chun-Han Lin Advisor：Jean Jyh-Jiun Shann

Institute of Computer Science and Engineering

National Chiao Tung University

Abstract

With process technology scale down to the deep submicron level, crosstalk effects are increasingly important considerations especially when adjacent bus lines switch in opposite directions (so called crosstalk-toggling transitions) on deep-submicron buses. Crosstalk-toggling transitions increase not only power consumption but also data transmission delays. While many bus encoding schemes have been proposed to totally avoid crosstalk-toggling transitions thus reducing data transmission delays in synchronous circuit designs, opportunities still exist to additionally reduce power consumption.

Therefore, we propose a register relabeling algorithm to further reduce the instruction bus power consumption based on the existing Selective Shielding bus encoding scheme which guarantees the encoded bus being crosstalk-toggling free. With no extra hardware requirements and performance loss, the average energy consumption of our design is 95.3% compared with an un-encoded instruction bus using 90nm technology with a 14mm bus length, and is 12.1% less than that of an instruction bus with Selective Shielding coding. In summary, our scheme preserves the crosstalk-toggling free characteristic of the Selective Shielding method and saves more energy.

致謝

由衷感謝我的指導老師 單智君教授，在老師細心的教誨與指導之下，使我學習到對於任何事物皆可進一步探索其原理、發現問題、並解決問題，進而培養其研究的能力。同時感謝實驗室的另外一位大家長 鍾崇斌教授，多次提出寶貴的建議與意見，使我受益良多。

除此之外，感謝實驗室的博士班翁綜禧學長，不論在任何方面對我提出問題並適時的給予建議；即使在我人生中的低潮，學長依然鼓勵並勉勵我繼續加油。還有，感謝實驗室的學長姐、同學及學弟妹，感謝你們，不僅是研究的好夥伴，更是一生的好朋友。

最後，感謝我的家人，對於我的栽培不餘遺力，並再次對於所有幫助過我的人，獻上最真誠的感謝。

林均翰 2010.11

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

In this chapter, we first introduce the importance low power design, and, then, discuss the sources of power consumption on bus and the effects of crosstalk-toggling transitions. The research motivation and objective are then introduced. The organization of this thesis is elaborated in the end.

1.1 Importance of Low Power Design

As the complexity of system-on-chip (SoC) design increases, power consumption is becoming one of the most important design issues especially for embedded systems due to heat reduction, cooling cost reduction, longer cell life, and etc. In addition to these problems, energy efficiency has become an important characteristic of product quality. In mobile devices such as cellular phone and other handheld devices, energy efficiency further determines the usability and acceptance of these products. Since these products are battery-powered and the required usage amounts are increasing rapidly, low power design for these systems becomes a very important research topic.

1.2 Sources of Power Consumptions on Buses

The power consumption of bit transitions on bus lines is one of the major sources to the total power consumption. The power consumption by bit transitions on bus lines comes from

charging and discharging the capacitance for data transmission. The bit transitions can be classified into self-transition and coupling-transition of capacitances [1]. As CMOS processes scale down to the deep submicron level, both self-capacitance and coupling-capacitance needs to be taken into account. Capacitance between a bus line and ground is called self-capacitance (C_s), and capacitance between adjacent bus lines is called coupling-capacitance (C_c). Both capacitances are shown in Figure 1-1.

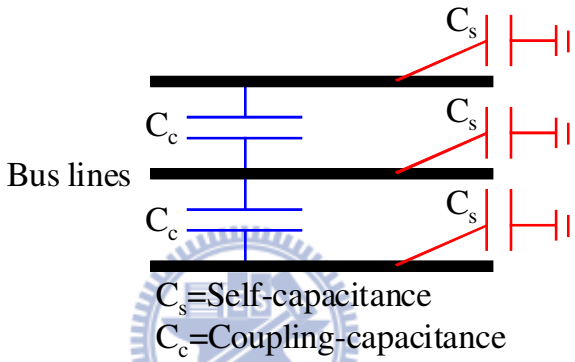
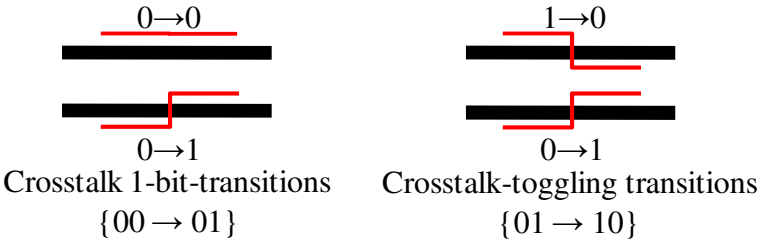


Figure 1-1：Self and coupling-capacitance for buses

Self-transitions are bit transitions on each individual bus line which make self-capacitance charging and discharging. Coupling-transitions are bit transitions between adjacent bus lines that cause a voltage level difference and thus cause coupling-capacitance charging and discharging. Coupling-transitions can be subdivided into two types, crosstalk 1-bit transitions and crosstalk-toggling transitions. Moreover, crosstalk 1-bit transitions occur in the cases when only one of the bus lines switches between adjacent bus lines, for examples {00 ⇔ 01}, {00 ⇔ 10}, {11 ⇔ 01}, {11 ⇔ 10}. Crosstalk-toggling transitions occur when both of the adjacent bus lines switch to the opposite directions, for examples {01 ⇔ 10}. And

the remaining cases, for examples {00 ⇔ 00}, {11 ⇔ 11}, {00 ⇔ 11}, do not trigger any activity on coupling-capacitance. Figure 1-2 shows the examples of a crosstalk 1-bit transition and a crosstalk-toggling transition.



Crosstalk 1-bit-transitions {00 → 01}

Crosstalk-toggling transitions {01 → 10}

Figure 1-2：Examples of a crosstalk 1-bit transition and a crosstalk-toggling transition

1.3 Effects of Crosstalk-Toggling transitions

With process technology moving toward the deep submicron level, coupling-capacitance between adjacent bus lines is becoming ever more prominent. The ratio of coupling-capacitance to self-capacitance increases as process shrinks [2]. Crosstalk-toggling transitions cause not only more power consumption but also longer data transmission delays. The data transmission delay from crosstalk-toggling transition is at least twice of that of other transitions [3]. As regards power consumption, the power consumption due to crosstalk-toggling transitions is at least four times of that of other transitions [1]. Thus, the effects of crosstalk-toggling transitions are much more serious than that of other transitions.

1.4 Research Motivation

Since the effects of crosstalk-toggling transitions are much more serious than that of

others, many bus-encoding schemes have been proposed to totally avoid the crosstalk-toggling transitions. The purpose of crosstalk-toggling-free bus encoding schemes is to reduce data transmission delay in synchronous circuit designs. However, opportunities still exist in previous crosstalk-toggling-free bus encoding schemes to reduce total power consumption on buses at the same time with crosstalk-toggling free.

The power consumption on instruction bus constitutes great portion of total power consumption on buses since a processor typically accesses instructions every instruction cycle and the bit patterns of instruction bus is less regular than that of its address bus. However, instructions are compiled at static time. There are opportunities to deal with instructions in a post-compilation phase. For example, a typical ISA exhibits regularity that the register fields are in fixed positions within the instruction encoding, and the register fields constitute a significant part of an instruction word. Choosing registers appropriately may reduce the power consumption of instruction bus [4]. Therefore, it is possible to reduce the power consumption by generating instructions which consume less power.

1.5 Research Objective and Approaches

In this thesis, instructions are handled at static time to further reduce power consumption for a crosstalk-toggling-free-coded instruction bus with no extra hardware and performance loss. This goal is achieved by exploiting the characteristics of code-words on crosstalk-toggling-free encoded bus. We figure out the power consumptions on

crosstalk-toggling-free encoded instruction bus that depend only on the number of 1s of code-words. Thus, the instructions which have less 1s after crosstalk-toggling-free bus encoding are generated. Moreover, register relabeling is used for relabel registers of instructions, and our modified register relabeling method can consider only the register number itself. Furthermore, the relabeling scope may be a smaller one that provides more opportunities to reuse register numbers with less 1s after crosstalk-toggling-free bus encoding resulting in fewer transitions. Consequently, our approaches will be suitable for crosstalk-toggling-free-coded instruction bus so as to reduce the bit transitions on instruction bus for power reduction.

1.6 Organization of This Thesis

The remaining chapters of this thesis are organized as follow. Chapter 2 introduces the source of power consumption and analytical model of delay and discusses previous related researches on crosstalk-toggling free and power reduction techniques for instruction bus. In Chapter 3, we illustrate our power reduction techniques for instruction bus. The experimental environment, simulation results and relative analysis are presented in Chapter 4. Finally, we summarize our conclusions and future works in Chapter 5.

Chapter 2 Background and Related Work

The main purpose of this chapter is to provide the necessary background for the concepts and methods presented in the following chapters. First, we will introduce the analytical model of power consumption for deep submicron buses. Then, a survey of the related approaches for crosstalk-toggling free bus encoding scheme and bus power reduction will be presented.

2.1 Analytical Model of Power Consumption

There are three major sources of power consumption in digital CMOS circuits [5]. The first one is the switching power for charging and discharging the circuit node capacitances. The second one is short-circuit power due to the direct-path short circuit current arises when both the NMOS and PMOS transistors are active simultaneously, and then conduct current directly from supply to ground. Finally, leakage power, which can arise from substrate injection and sub-threshold effects, is primarily determined by fabrication technology considerations while we will not discuss it [6].

In this thesis, we focus on reducing the switching power for charging and discharging the capacitances. The equation for the total power consumption of switch power listed as follows:

[1]

$$P_{switching} = P_s + P_c$$

$$= ST \cdot C_S \cdot V_{dd}^2 + XTTr \cdot C_C \cdot V_{dd}^2 + 4XTTg \cdot C_C \cdot V_{dd}^2$$

$$= (ST + \lambda \cdot XTTr + 4\lambda \cdot XTTg) \cdot C_S \cdot V_{dd}^2, \quad (1)$$

,where the first term, P_s, represents the switching power of self-transitions; the second term, P_c, represents the switching power of coupling-transitions that included crosstalk 1-bit transitions and crosstalk-toggling transitions where C_S is the self-capacitance, C_C is the coupling-capacitance, V_{dd} is the supply voltage, λ is equal to C_C / C_S, ST is the total number of self-transitions, $XTTr$ is the total number of crosstalk 1-bit transitions, and $XTTg$ is the total number of crosstalk-toggling transitions.

Low power design is to minimize transitions, capacitances, and V_{dd}. Once the technology process has been chosen, capacitances will be decided. From the power equation, decreasing the V_{dd} factor can be an effective way for power dissipation of switch power. However, the supply voltage is usually determined by the system and technology consideration, and decreasing V_{dd} will increase the propagation delay consequently. Finally, the remaining important factor is the transitions. Reducing the number of bit transitions per transaction may reduce the number of capacitances needed to be driven. Bus encoding is a well-known technique to encode the contents of a bus to reduce the total bit transitions. Consequently, the

equation of power dissipation cost (PDC) function can be defined as follows：

$$PDC = ST + \lambda \cdot XTTr + 4\lambda \cdot XTTg \quad (2)$$

In this thesis, minimizing the PDC function is the goal of our proposed methods. For crosstalk-toggling-free bus encoding scheme, $XTTg$ is guaranteed to be "0", and, thus, our objective is to reduce ST and $XTTr$ as many as possible for power reduction.

2.2 Previous Crosstalk-Toggling-Free Methods for Buses

Many methods have been proposed to totally avoid crosstalk-toggling transitions to reduce transition delays in synchronous circuit design. We briefly describe some of these techniques and discuss the reason why we focus on one of them.

2.2.1 Simple Shielding Technique

The simplest method to avoid crosstalk-toggling transitions is the simple shielding technique, where a shield line is inserted between every pair of adjacent bus lines [7]. The shield lines have no signal transitions, and, thus, crosstalk-toggling transitions are avoided. No encoder and decoder are required, but n-1 extra bus lines are needed for n-bit bus, and, thus, double the area used by the bus. When the bus is routed using scare top-level metal resources, double area is an unacceptable consequence [8].

2.2.2 Victor's Method

From the concept of simple shielding technique, Victor's method provides a theoretical framework to generate crosstalk-toggling free code-words [8]. As compared to $2n$-1 bus lines required by the simple shielding technique, Victor's method proofs that the lower limit on the number of required bus lines is $\lfloor log_2(f_m) \rfloor$ for a n-bit bus, where f_m is the mth Fibonacci number, for example, it requires 46 bus lines for 32-bit bus. However, it is hard to generate the crosstalk-toggling-free code-words due to the lack of generalized procedure to generate the code-words.

2.2.3 Fibonacci Coding

Fibonacci coding scheme is based on the theoretical framework of Victor's method and gives a recursive procedure to generate code-words [9]. The same number of bus lines, $\lfloor log_2(f_m) \rfloor$, is required as Victor's method for n-bit bus.

Table 2-1 shows the encoding algorithm of Fibonacci coding, and Figure 2-1 shows the examples from f_1 to f_4. In Fibonacci coding, let $a_m a_{m-1} \cdots a_2 a_1$ is an m-bit crosstalk-toggling-free Fibonacci code-word. The decimal value will be $a_m \times \text{Fib}(m) + a_{m-1} \times \text{Fib}(m-1) + \ldots + a_2 \times \text{Fib}(2) + a_1 \times \text{Fib}(1)$, where Fib($i$), $1 \leq i \leq m$, is the i^{th} Fibonacci number.

Fibonacci coding scheme gives a recursive procedure to generate code-words. However,

the larger data width, the higher gate delay of the corresponding encoder and decoder will be.

Table 2-1：Fibonacci encoding algorithm

$f_1 = \{0, 1\}$
if m is odd $f_{m+1} = \{0\ x \mid \forall\ x \in F_m\} \cup \{11\ y \mid \forall\ 1\ y \in f_m\}$ else $f_{m+1} = \{00\ x \mid \forall\ 0\ x \in f_m\} \cup \{1\ y \mid \forall\ y \in f_m\}$

f_1	f_2	f_3	f_4
1	**1 1**	**2 1 1**	**3 2 1 1**
0	0 0	0 0 0	0 0 0 0
1	0 1	0 0 1	0 0 0 1
	1 1	1 0 0	0 1 0 0
		1 0 1	0 1 0 1
		1 1 1	0 1 1 1
			1 1 0 0
			1 1 0 1
			1 1 1 1

Figure 2-1：The examples of Fibonacci encoding from f_1 to f_4

2.2.4 Selective Shielding Technique

The concept of Selective Shielding comes also from the simple shielding technique. It is shown that if the code-words may avoid adjacent 1s data pattern, the crosstalk-toggling transitions may be avoided after Transition Signaling encoding scheme. Thus, Selective Shielding guarantees that there is no adjacent 1s in coed-words, and the required bus lines are $3n/2$ for n-bit bus.

4-to-6 Selective Shielding is an extension from Selective Shielding. The concept of

4-to-6 Selective Shielding is also to avoid adjacent 1s in coed-words, and $3n/2$ bus lines are required as well for n-bit bus. The difference between Selective Shielding and 4-to-6 Selective Shielding is that 4-to-6 Selective Shielding partitions data into several fields and may encode all fields at the same time. Hence, the gate delay of encoder and decoder of 4-to-6 Selective Shielding is much less than that of Selective Shielding.

2.2.5 Comparison of Different Approaches

Table 2-2 shows the comparison of these above approaches. Considering the required bus lines and the coding delay of encoder and decoder, we choose 4-to-6 Selective Shielding in our method. Therefore, the detail description of Selective Shielding and the extension, 4-to-6 Selective Shielding, will be introduced in the next section.

Table 2-2：Comparison of different approaches

Approaches (for n-bit bus)	# of bus lines required	Delay of Encoder	Delay of Decoder
Simple Shielding	$2n$-1	—	—
Victor's Method	≒ (<) $3n/2$	N/A	N/A
Fibonacci Coding	≒ (<) $3n/2$	High	High
Selective Shielding	$3n/2$	Medium	Medium
4-to-6 SS	$3n/2$	Low	Low

2.3 Previous Researches

Two previous researches of crosstalk-toggling-free bus encoding schemes, Selective

Shielding (SS) encoding scheme [10][11] and 4-to-6 Selective Shielding (4-to-6 SS) encoding scheme [10], and one previous research for reducing the switching activities on buses, register relabeling [4], are introduced in the following subsections.

2.3.1 Selective Shielding Crosstalk-Toggling-Free Technique

The goal of Selective Shielding (SS) encoding scheme is to avoid crosstalk-toggling transitions on bus by using *n*/2 extra bus lines for *n*-bit bus [10][11]. The basic idea of SS comes from Transition Signaling (TS) encoding scheme [12]. The encoding of TS encoding scheme is to perform an XOR operation on the *n*-bit previous bus value (Bus_{t-1}) with the *n*-bit current data ($Data_t$) and transmit the result as the *n*-bit current bus value (Bus_t), that is, $Bus_t = Bus_{t-1} \oplus Data_t$. In decoding, get the current data ($Data_t$) by performing XOR operation on the previous bus value (Bus_{t-1}) with the current bus value (Bus_t), i.e., $Data_t = Bus_{t-1} \oplus Bus_t$. It is observed that only when the current data has adjacent 1s data pattern, a crosstalk-toggling transition may be generated on a TS encoded bus. The results of TS coding for all combinations of Bus_{t-1} and $Data_t$ are shown in Figure 2-2. Since crosstalk-toggling transitions occurred on adjacent bus lines, Figure 2-2 shoes only 2 bits for each of the previous bus value (Bus_{t-1}), the current bus value (Bus_t), and the current data ($Data_t$). All the possible combination of the 2-bit Bus_{t-1} are shown on the rows, all the possible combination of the 2-bit $Data_t$ are shown on the columns, and the results of the 2-bit Bus_t (= $Bus_{t-1} \oplus Data_t$.) are shown in the matrix. It is clear that only when the current data has “11” data pattern, the

previous bus value and the current bus value may cause a crosstalk-toggling transition.

Bus_{t-1} : previous bus value
$Data_t$: current data
Bus_t : current bus value

Bus_{t-1} \ $Data_t$	00	01	10	11
00	00	01	10	11
01	01	00	11	**10**
10	10	11	00	**01**
11	11	10	01	00

Bus_t

Figure 2-2：Results of TS encoding scheme for $Bus_t = Bus_{t-1} \oplus Data_t$

Therefore, if the current data do not have adjacent 1s, crosstalk-toggling transitions may be avoided on TS encoded bus. According to this basic idea, the design of SS is to make sure that there are no adjacent 1s in the code-words to make the crosstalk-toggling free after TS coding. Figure 2-3 shows the system overview of SS.

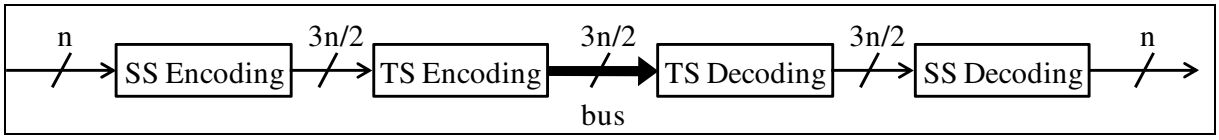


Figure 2-3：System overview of SS

The method of avoiding adjacent 1s in data is to encode each “1” to “10” rather than simple shielding technique [7] which inserts a shield bit (assume “0”) between adjacent data bits. While the data bits are all 1s, the SS encoding method will be the same as the simple shielding technique. Therefore, the number of 0s should be added depends on the number of

1s present in the data. In order to reduce the number of 1s in data to limit the number of added 0s, SS technique calculates the number of 1s in data first. If the number of 1s in the *n*-bit data is less than *n*/2, encode each "1" to "10" directly. Otherwise, invert the data first to make the number of 1s in the *n*-bit data less than *n*/2, and then encode each "1" to "10" of the converted data. After that, append an invert bit "1" at the LSB of the code-word to denote that the data have been inverted.

Note that it needs at most *n*/2 extra bus lines to encode an *n*-bit bus. In order to providing fixed length (3*n*/2-bit) of code-words, append "0s" at MSB positions if the length of a code-word is less than 3*n*/2.

In decoding, check the LSB of the code-word first. If LSB is "0", convert each "10" to "1" directly. Otherwise, it means that the data have been inverted in encoding process. Thus, cut the end-bit first and covert each "10" to "1", and then invert the converted data. After above decoding process, remove the leading bits that exceed the original data length (*n*).

The encoding and decoding algorithms of SS technique are shown in Figure 2-4, and examples of applying the algorithms are shown in Figure 2-5.

If # of 1s in the n-bit data $\leq n/2$
    Each "1" is encoded as "10".
Else
    Invert the data first.
    Each "1" is encoded as "10".
    Append an invert bit, "1" , at LSB.
Append 0s at MSB to provide a $3n/2$-bit code-word.

(a) Encoding algorithm

If the end-bit of a code-word == 1
    Cut the end-bit, covert "10" to "1."
    Invert the converted code-word.
Else
    Convert "10" to "1."
Remove leading bits that exceed the original data length(n)

(b) Decoding algorithm

Figure 2-4：SS encoding/decoding algorithm

Encoding

Data	0001 0001	1101 1000	1100 1110	1011 1111
Invert?	X	X	0011 0001	0100 0000
"1"to "10"	00 0100 0010	1010 0101 0000	001 0100 0010	0 1000 0000
If inverted, append 1	X	X	001 0100 0010 1	0 1000 0000 1
Append 0s for fixed length ($3n/2$)	00 00 0100 0010	X	X	00 0 1000 0000 1

Decoding

Code-word	0000 0100 0010	1010 0101 0000	0010 1000 0101	0001 0000 0001
If inverted, cut the end-bit	X	X	0010 1000 010	0001 0000 000
"10"to "1"	00 0001 0001	1101 1000	0011 0001	00 0100 0000
Inverted?	X	X	1100 1110	11 1011 1111
Remove the leading bit	0001 0001	X	X	1011 1111

Figure 2-5：SS encoding/decoding examples

2.3.2 4-to-6 Selective Shielding Crosstalk-Toggling-Free Technique

Since SS encoding scheme encodes the whole n-bit data at a time, its corresponding

hardware is complex and time consuming. If data are partitioned into several fields and each field is encoded individually at the same time, its corresponding hardware may be simpler and may save more transition time than SS encoding scheme. 4-to-6 selective shielding (4-to-6 SS) was proposed for these proposes. It is an extension of SS encoding scheme with smaller encoding unit and needs also *n*/2 extra bus for *n*-bit bus [10]. Figure 2-6 shows the system overview of 4-to-6 SS.

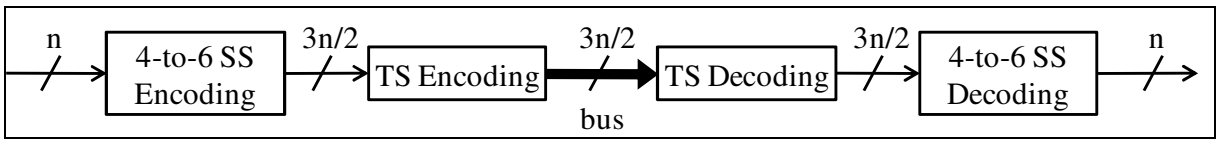


Figure 2-6：System overview of 4-to-6 SS

The 4-to-6 SS encoding scheme first apply the smallest possible encoding unit of SS to encode 2-bit data into 3-bit code-words to simplify the corresponding hardware. Figure 2-7 shows the 2-bit data and the corresponding 3-bit code-words. In the other words, it applies SS technique to *n*/2 2-bit sub-data in parallel to generate *n*/2 3-bit code-words.

data	Code-words
00	000
01	010
10	100
11	001

Figure 2-7：The 2-bit data and the relative 3-bit SS code-words

However, when data exist 11 patterns which are followed by 10 patterns, there have adjacent 1s between code-words of adjacent partitions, that is, 11 10 are encoded into 001 100. To overcome this problem, once it encounters adjacent 1s between two 3-bit code-words, it

swaps one of the adjacent 1s with other bit to avoid the two adjacent 1s. Assuming that the *i*th and the (*i*-1)th bits are the two adjacent 1s, swap the *i*th and the (*i*-3)th bits. After swapping the *i*th and the (*i*-3)th bit positions, if the (*i*-4)th bit is a "1", repeat the swapping process until there is no adjacent 1s between code-words. In the worst case, it may require $n/2 - 1$ bit-swaps and thus increases the coding delay. Figure 2-8 gives an example of the bit-swap process. From Figure 2-8, when data exist 11 patterns followed by m 10 patterns, m swaps will occur.

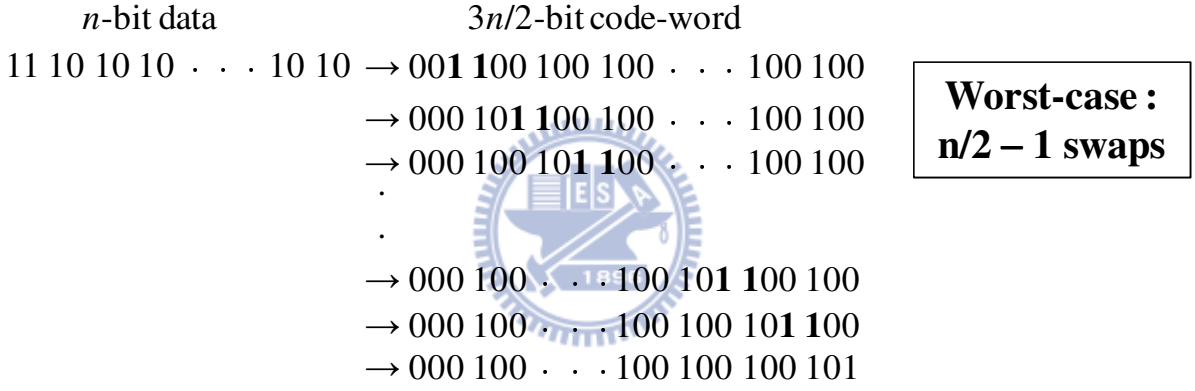


Figure 2-8：An example of the worst case of bit-swap process

In order to reducing the number of swaps from $n/2 - 1$ to 1, 4-to-6 SS consider partition data into several fields with size 4, then partition each 4-bit field into two 2-bit sub-partitions, and then apply SS to two 2-bit sub-partitions to generate a 6-bit code-word. The bit-swap process is the same as that mentioned before, i.e., swap the *i*th and the (*i*-3)th bits if the *i*th and the (*i*-1)th bits are both 1s. Under the encoding method, it is apparent that 1110 is encoded into 001100 which has adjacent 1s in its code-word and thus bit swapping intra cod-word is applied to form 000101. Moreover, 1010 is encoded into 100100 which may have repetitious

swaps inter code-words if the right-hand-side code-word ends with 1 and the same bit-swap process is performed, and thus 1010 is encoded into 010101 to avoid repetitious swaps. Table 2-3 shows the 4-bit data and the corresponding 6-bit code-words. However, when the code-words of left-hand side end with 1 and the code-words of right-hand-side start with 1, the adjacent 1s inter code-words happen. Note that code-words of right-hand-side start with 1, and the following 2nd, 3rd and 4th bits are 0s. Therefore, once it encounters adjacent 1s inter 6-bit code-words, it needs only one bit-swap process to avoid adjacent 1s. Figure 2-9 shows an example of the bit-swap process.

Table 2-3：The 4-bit data and relative 6-bit code-words

4-bit data	4-to-6 SS code-word
0000	000000
0001	000010
0010	000100
0011	000001
0100	010000
0101	010010
0110	010100
0111	010001
1000	100000
1001	100010
1010	**010101**
1011	100001
1100	001000
1101	001010
1110	**000101**
1111	001001

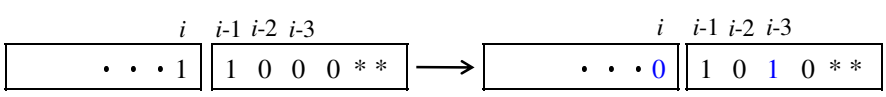


Figure 2-9：An example of the bit-swap process inter adjacent code-words

In 4-to-6 SS decoding, if the 6-bit code-words has “1010**” data pattern, it means that the bit-swap process has been applied to the code-words to avoid adjacent 1s between 6-bit code-words. Thus, it needs to swap back first, and then process 4-to-6 SS decoding. Figure 2-10 shows examples of 4-to-6 SS encoding and decoding.

Encoding

Data	0010 1010	0011 1011
4-to-6 SS code-word	000100 010101	000001 100001
If adjacent 1s, swap	X	000000 101001

Decoding

4-to-6 SS code-word	000100 010101	000000 101001
If 1010**, swap back	X	000001 100001
Data	0010 1010	0011 1011

Figure 2-10：4-to-6 SS encoding/decoding examples

2.3.3 Register Relabeling Power Saving Technique

In a typical RISC ISA, register fields are fixed within the instructions and occupy large portion in the instruction encoding. If the number of bit transitions in two register numbers in the same bit positions of two consecutive instructions is higher and the combination of the two register numbers often appears in the any two consecutive instructions, the power consumption will be larger. However, the registers of a typical RISC ISA are general purpose, and general-purpose registers are interchangeable. The basic idea of register relabeling is to

minimize the bit transitions of register fields during instruction fetches by relabeling register numbers statically [4]. Figure 2-11 shows an example of code fragment. It could achieve reduction in bit transition with no performance penalties.

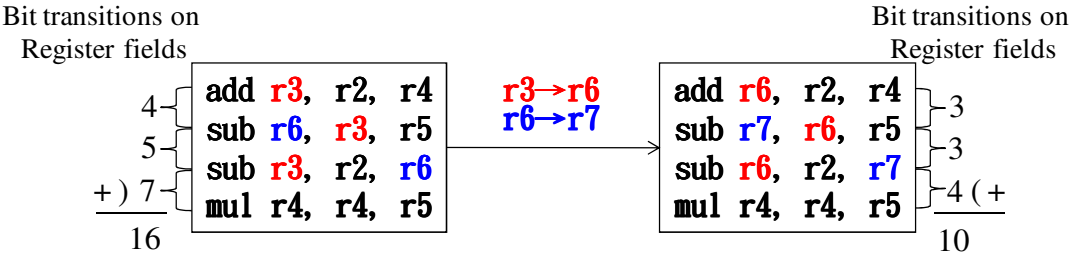


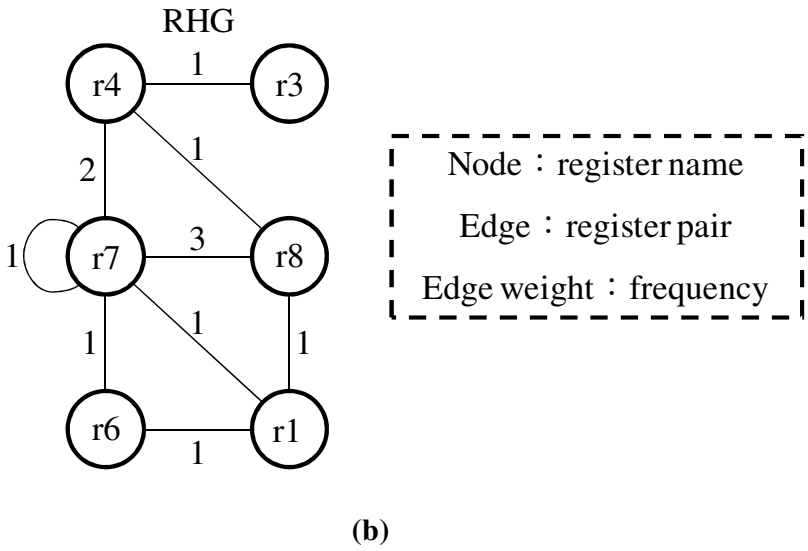
Figure 2-11：An example of code fragment

The first step of relabeling is that constructed a graph called the “Register Histogram Graph” (RHG). The RHG captures the occurrence frequency and relationship between register pairs which are two register numbers in the same bit positions of two consecutive instructions. Each RHG node represents a register. Each RHG edge represents that two register numbers compose a register pair, and the weight of each edge annotates with the frequency of register pairs. Figure 2-12 (a) shows an example of all pairs of registers appeared in a code fragment and the frequency of each pair. In Figure 2-12, assume that the architecture uses registers from register $1 to register $8. Figure 2-12 (b) is a RHG derived from Figure 2-12 (a).

Reg Pair	frequency
(r7 , r8)	3
(r4 , r7)	2
(r1 , r6)	1
(r1 , r7)	1
(r1 , r8)	1
(r3 , r4)	1
(r4 , r8)	1
(r6 , r7)	1
(r7 , r7)	1

Total bit transitions：29

(a)



(b)

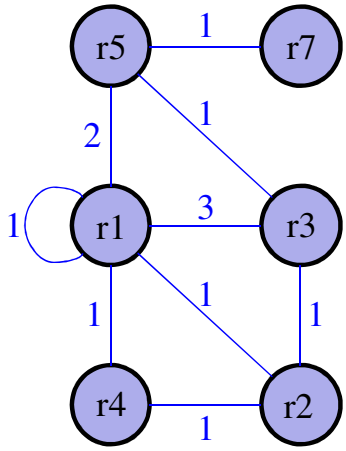
Figure 2-12：(a) Example frequency distribution of register pairs (b) RHG from (a)

The following algorithm utilizes the RHG to relabel the register numbers [13]. Figure 2-13 shows the RHG after register relabeling. In this example, start from the most frequent edge of register pair, register $7 and register $8, relabel them into a register pair, register $1 and register $3, whose hamming distance is minimized. Then, for the second most frequent edge of register pair, register $7 and register $4 , since the register $7 is assigned, relabel register $4 into register $5 so that the hamming distance to its assigned neighbor registers is minimized. The following relabeling steps are the same as above description.

Algorithm

- Iterate through the edges starting from the most frequent ones
- Rename the registers yet unassigned so that hamming distance to ***all their assigned neighbors*** in the graph ***is minimized***

(a)



Reassigned results

r7 → r1
r8 → r3
r4 → r5
r1 → r2
r6 → r4
r3 → r7

Total bit transitions from 29 to 15

(b)

Figure 2-13：(a) Register relabeling algorithm (b) RHG after register relabeling

2.3.4 Summary of Previous Researches

The SS and 4-to-6 SS are all crosstalk-toggling free bus encoding schemes and both need *n*/2 extra bus for n-bit bus. Since SS encodes the whole *n*-bit data at a time, its corresponding hardware is more complex and time consuming than that of 4-to-6 SS which partitions data into several fields and encodes all fields at the same time. We focus on 4-to-6 SS since its corresponding hardware is less time consuming and the partitioning method can be further complemented by our modified relabeling method.

Register relabeling may reduce bit transitions of register fields on a traditional instruction bus. It needs to consider the relationship between register fields of consecutive instructions. 4-to-6 SS encoding scheme brings a different situation for register relabeling such that the original register relabeling method may not be suitable on a 4-to-6 SS coded bus.

Due to the characteristics of TS encoding scheme, if data hold fewer 1s in code-words, the number of bit transitions on a TS encoded bus is lower [12]. Therefore, we may make use of the characteristics to modify register relabeling for 4-to-6 SS to produce code-words with fewer 1s to reduce the number of bit transitions on a TS encoded bus. The detail description of our design will be discussed in the next chapter.

Chapter 3 Proposed Design

This chapter will introduce our design of modified register relabeling to reduce the number of bit transitions on instruction bus. The overview of proposed design will be shown in Section 3.1. The observations of our design foundation will be presented in Section 3.2. The remaining sections will show the details of our design.

3.1 System Overview

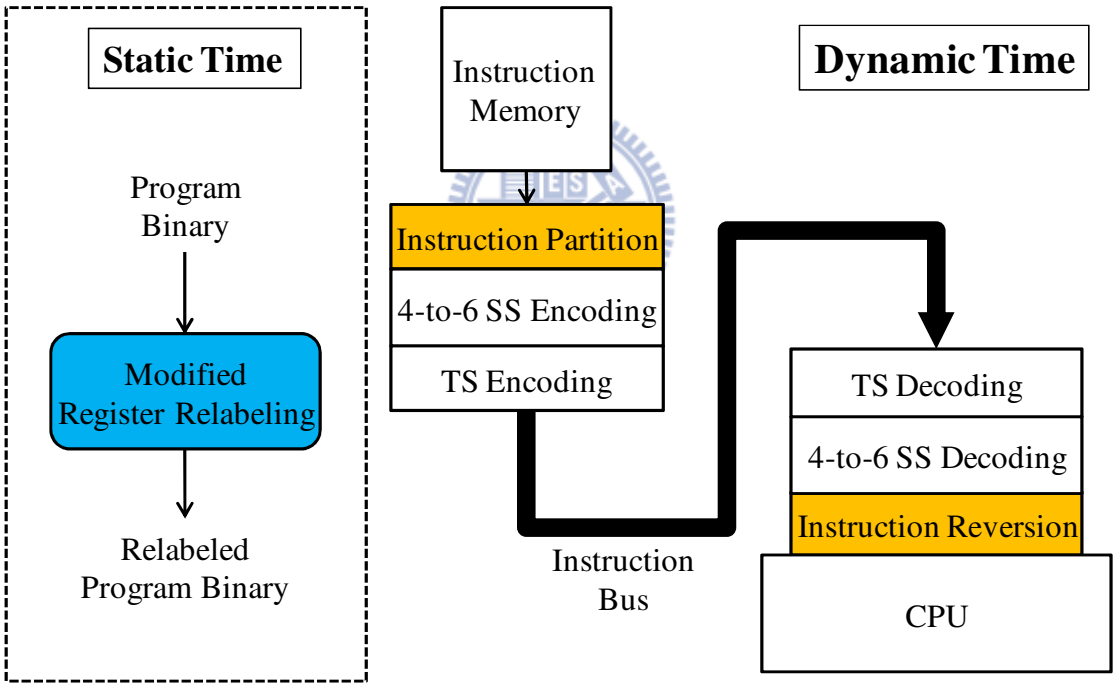


Figure 3-1：Overview of system

The system contains static-time phase and dynamic-time phase. Figure 3-1 illustrates the system overview.

Our method concentrates mainly on the register fields of instructions. Our modified register relabeling is applied to the program binary according to the partition of instructions

for 4-to-6 SS encoding scheme to produce relabeled program binary that resides in the instruction memory at static time. At dynamic time, instruction will be partitioned in Instruction Partition step after fetching from instruction memory in order to combine 4-to-6 SS encoding scheme with our modified register relabeling. After that, the coding process including data coding (4-to-6 SS Encoding/Decoding) and data transmitting (Transition Signaling Encoding/Decoding) through the instruction bus is exactly the same as that of the original 4-to-6 SS encoding scheme. The Instruction Reversion is the reverse of the Instruction Partition step.

3.2 Observations

As described in subsection 2.3.2, 4-to-6 SS encoding scheme brings a different situation for register relabeling, so that it is necessary to consider the impact. In a 4-to-6 SS encoded instruction bus, the current data is converted to 4-to-6 code-word without adjacent 1s, and then an XOR operation is performed between the previous bus value and the 4-to-6 SS code-word to get the current bus value. Figure 3-2 shows the flowchart of data processing.

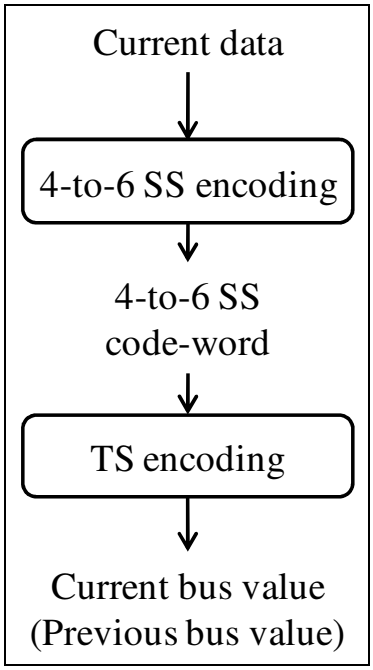


Figure 3-2：Flowchart of 4-to-6 SS data processing

Due to the characteristics of TS encoding scheme, if the inputs are a previous result and "0", the current result will be equal to the previous result; if the inputs are a previous result and "1", the current result will be equal to the inversion of the previous result. Therefore, the first observation is that the number of self-transitions between the previous bus vaule and the current bus value is equal to the number of 1s in the 4-to-6 SS code-word. Moreover, once "1" appears in a 4-to-6 SS code-word, its neighbor bits must be "0". After TS encoding scheme, the neighbor positions of a self-transtion must be no signal trantiions. Thus, the number of crosstalk 1-bit transitions between the previous bus value and the current bus value is twice as many as the number of 1s in the 4-to-6 SS code-word except the "1s" appeared in the most significant bit (MSB) and the least significant bit (LSB) bit postitions. Since the crosstalk 1-bit transitions may occur only on adjacent bus lines, each of the MSB and LSB bit positions

has only one adjacent bus line and thus may cause one crosstalk 1-bit transition at most. Figure 3-3 shows an example of 4-to-6 SS encoding scheme. From Figure 3-3, after XOR operation, the number of self-transitions between the previous bus value and the current bus value is 2 which is equal to the number of 1s, 2 "1s", in the 4-to-6 SS code-word. Furthermore, after XOR operation, the number of crosstalk 1-bit transitions between the previous bus value and the current bus value is 3 which is equal to twice of the number of 1s in the 4-to-6 SS code-word and minus the "1" appeared at LSB, $2 \times 2 - 1$.

$$
\begin{array}{ll}
0\ \ 0\ \ \cdot\cdot 0\ \ 1\ \ 0\ \ \cdot\cdot 0\ \ 1 & \text{(4-to-6 SS code-word, 2 "1s")} \\
\oplus\ a_{n-1}\ a_{n-2}\cdot\cdot a_{x+1}\ a_x\ a_{x-1}\cdot\cdot a_1\ a_0 & \text{(Previous bus value)} \\
\hline
a_{n-1}\ a_{n-2}\cdot\cdot a_{x+1}\ \overline{a_x}\ a_{x-1}\cdot\cdot a_1\ \overline{a_0} & \text{(Current bus value)}
\end{array}
$$

Figure 3-3：An example of 4-to-6 SS coding

Therefore, the power cost terms of the power dissipation cost (PDC) in Eq.(2) may be formulated as follows:

$$ST = \#\ of\ 1s\ in\ the\ 4to6\ SS\ codewords \tag{3}$$

$$XTTr = (2 \times \#\ of\ 1s\ in\ codewords) - (\#\ of\ 1s\ in\ MSB\ and\ LSB\ of\ codewords) \tag{4}$$

,where ST is the total number of self-transitions and $XTTr$ is the total number of crosstalk 1-bit transitions. As for the number of crosstalk-toggling transitions, $XTTg$, it is guaranteed to be 0 after 4-to-6 SS encoding scheme. Consequently, the power consumption depends on the number of 1s in the 4-to-6 SS code-words, and our modified register relabeling method is

built up by our observations.

3.3 Instruction Partition

The purposes of designing Instruction Partition are to preserve the chance for register relabeling on register fields and to make use of the characteristics of 4-to-6 SS encoding scheme. Firstly, for register fields, each register field is better to be fit in one partition. However, 4-to-6 SS encoding scheme requires 4-bit partitions, and, thus, each register field would be partitioned into 4-bit fields and the remaining bits of register fields will be processed with other fields. Taking the MIPS instruction set for example, its instruction format are shown in Figure 3-4 (a) [14]. The proposed partitions for all register fields of R-type and I-type, and bits on the same positions of I-type and J-type are shown in Figure 3-4 (b).

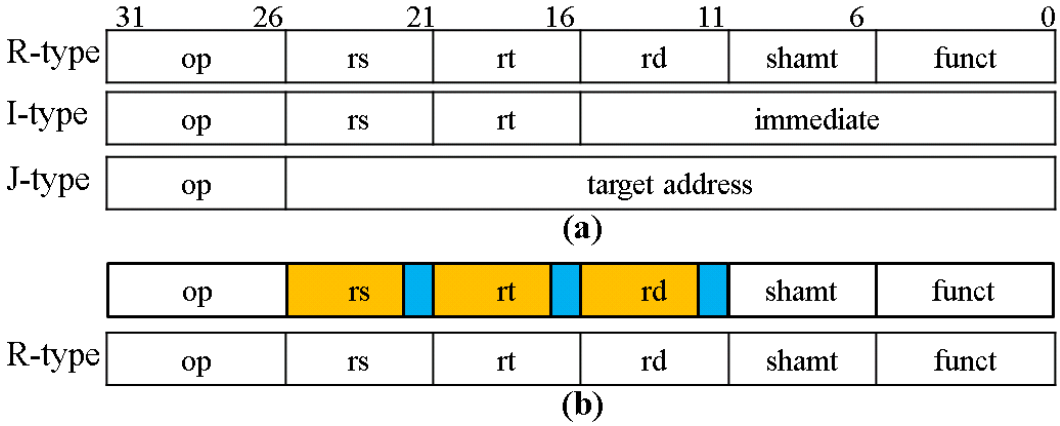


Figure 3-4：(a) MIPS instruction formats (b) Partition of register fields and bits on the same positions

Furthermore, characteristics of 4-to-6 SS encoding scheme should be considered for both the remaining bits of the register fields and other fields. From the code-words of 4-to-6 SS

encoding scheme, the bits with more 0s in the same partition will have a probability of having fewer 1s in their 4-to-6 SS code-words than that of original data, and fewer 1s in the 4-to-6 SS code-words lead to fewer transitions from our observations in Section 3.2. Table 3-1 shows the characteristics of 4-to-6 SS code-words. It is clear that if the 4-bit data have more 0s, the corresponding 6-bit code-words will have more 0s, too.

Table 3-1：4-bit data sorted by the number of 0s and their corresponding 4-to-6 SS code-words

4-bit data	4-to-6 SS code-word	# of 1s
0000	000000	0
0001	000010	1
0010	000100	1
0100	010000	1
1000	100000	1
0011	000001	1
0101	010010	2
0110	010100	2
1001	100010	2
1010	010101	3
1100	001000	1
0111	010001	2
1011	100001	2
1101	001010	2
1110	000101	2
1111	001001	2

Therefore, our approach is to sort the probabilities of 0s of the remaining bits of register fields and all other fields after register relabeling at static time, and then partition them into 4-bit fields by the sorted order. In order to avoid extra hardware overheads, we apply fixed partition only according to the statistics of a specific set of applications for the system.

3.4 Modified Register Relabeling

According to our observations, no matter what the previous bus values is, the power consumption depends only on the number of 1s in the 4-to-6 SS code-words of the current data. Therefore, rather than depending on the relation between registers as the case for original register relabeling, the power consumption caused by the 4-to-6 encoded register fields depends on the register numbers themselves only. The basic idea of our modified register relabeling is to relabel more frequently occurred registers to registers that have fewer 1s after 4-to-6 SS encoding.

In addition, we may count the number of 1s in 4-to-6 SS code-word of each register in advance to decide which register should be selected early to relabel the freuqently occurred registers. The selection order is called the relabeling register selection sequence. This relabeling register selection sequence is constructed in terms of the instruction partition on register fields. For example, according to the instruction partition on register fields as shown in Figure 3-4 (b), the leading 4 bits of each register can be classified according to the number of 1s in its corresponding 4-to-6 SS code-word. Note that there are two registers that have the same leading 4 bits with a different least significant bit (LSB). Figure 3-5 shows an example of the classification of registers. In this example, register $6 and register $7 have the same leading 4 bits, i.e., they have the same 4-to-6 SS code-word and a different LSB.

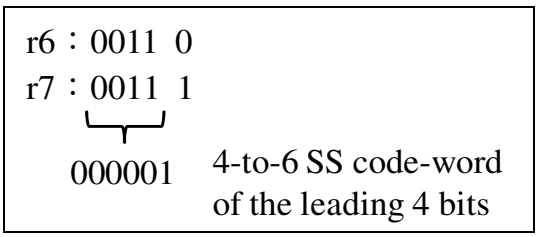


Figure 3-5：An example of classification of register

In Table 3-2, registers are classified according to the number of 1s in the 4-to-6 SS code-word of the leading 4 bits of its register number. The registers that have less number of 1s in their corresponding 4-to-6 SS code-words of the leading 4 bits of their register numbers are selected for relabeling first. Then, for two registers with the same number of 1s in the 4-to-6 SS code-words of their leading 4 bits, choose the one with LSB "0" to gain a higher probability of having less 1s in the code-word than that with LSB "1". The last column of Table 3-2 shows the selection order for register relabeling in descending priorities. The registers with the same sequence number may be chosen randomly.

Table 3-2：Relabeling register selection sequence

# of 1s in the 4-to-6 SS code-word of the leading 4-bit of register number	LSB	Register number (in decimal)	Relabeling selection sequence
0	0	r0	1
	1	r1	2
1	0	r2 r4 r6 r8 r16 r24	3
	1	r3 r5 r7 r9 r17 r25	4
2	0	r10 r12 r14 r18 r22 r26 r28 r30	5
	1	r11 r13 r15 r19 r23 r27 r29 r31	6
3	0	r20	7
	1	r21	8

Due to the constraints of an instruction set architecture (ISA), the registers may be classified into non-relabelable and relabelable. For non-relabelable regsiters, these registers sholud not be relabeled and can not be used for relabeling for the whole program.

Taking the register usage conventions of MIPS architecture for example, registers are classified in terms of their usage purposes as shown in Table 3-3 [15]. In MIPS registers, register $0 is non-relabelable since register $0 is hard wired to the value zero. Register $31 is the destination register used by instructions JAL, BLTZAL, BLTZALL, BGEZAL, and BGEZALL without being explicitly specified so that register $31 is non-relabelable, neither. The remaining registers are relabelable. Therefore, the relabeling registers selection sequence for MIPS ISA is shown in Table 3-4.

Table 3-3：MIPS registers categorization

Category	Name	Number	Use
Non-relabelable	$zero	$0	Always 0
	$ra	$31	Return address

Category	Name	Number	Use
Relabelable	$at	$1	Assembler temporary
	$k0 - $k1	$26 - $27	Kernel registers
	$gp	$28	Global pointer
	$sp	$29	Stack pointer
	$v0 - $v1	$2 - $3	Return value
	$a0 - $a3	$4 - $7	Argument registers
	$t0 - $t9	$8 - $15, $24 - $25	Temporary registers
	$s0 - $s8	$16 - $23, $30	Saved registers

Table 3-4：Relabeling register selection sequence for MIPS ISA

# of 1s in the 4-to-6 SS code-word of the leading 4-bit of register number	LSB	Register number (in decimal)	Relabeling selection sequence
0	1	r1	1
1	0	r2 r4 r6 r8 r16 r24	2
	1	r3 r5 r7 r9 r17 r25	3
2	0	r10 r12 r14 r18 r22 r26 r28 r30	4
	1	r11 r13 r15 r19 r23 r27 r29	5
3	0	r20	6
	1	r21	7

In this thesis, we propose two register relabeling methods. In register relabeling method 1, we gather the occurrence frequency of each relabelable register from a program trace, and then relabel more frequently occurred registers to registers that have fewer 1s after 4-to-6 SS encoding. In this method, each relabelable register is relabeled to a specific registers consistently for the whole program to reduce the power consumption while preserving the correctness of the program.

However, considering register usage convention and no performance degradation, there are regsiters that are used independently for each procedure. These registers in different procedures may be relabeled into the same reigster which has less 1s after 4-to-6 SS encoding to reduce the number of 1s in 4-to-6 SS code-words for more power reduction. Thus, in register relabeling method 2, there are regsiters which may be relabeled independently for each procedure, while there are other registers which are still relabeled to a specific registers

consistently for the whole program to keep the correct execution.

The details of these two register relabeling methods are described in the following subsections

3.4.1 Register Relabeling Method 1

Figure 3-6 shows the flowchart of our modified regsiter relabeling method 1. The first step of this method records the occurrence frequency of each relabelable register from a program trace. In the next step of register relabeling method 1, sort the occurrence frequencies of the relabelable regsiters. The final step is to relabel the registers by the sorted order according to the relabeling register selection sequence shown in Table 3-2. In this method, a relabelable register is relabeled to another register consistently through the porgam, that is to say, it is a program-scoped relabelabel register.

Table 3-4 shows a relabeling example according to the selection sequence in Table 3-2. In this example, the occurrence frequencies of relabelable registers are collected and sorted. Then, according to the selection sequence in Table 3-2, relabel registers into registers with less 1s after 4-to-6 SS coding.

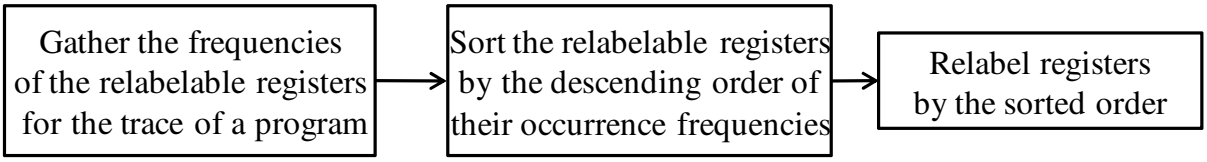


Figure 3-6：Flowchart of our modified regsiter relabeling algorithm

Table 3-5：An example of register relabeling method 1

Register before relabeling	Occurrence frequency	Register after relabeling
r8	12	r1
r6	8	r2
r1	6	r4
r4	6	r6
r3	3	r8
r12	2	r16
⋮	⋮	⋮

3.4.2 Register Relabeling Method 2

Register relabeling method 1 is program-scoped relabeling, i.e., each relabelabel register is relabeled to a specific registers consistently for the whole program. However, the relabeling scope of some registers may be relaxed to be within a procedure, i.e., some registers in different procedures may be relabeled into a same register to raise the occurrence frequencies of registers which have less 1s after 4-to-6 SS encoding. Figure 3-7 gives examples to show the different between program-scope relabeling and procedure-scope relabeling. Figure 3-7 (a) shows the rank order of each register occurrence frequency. In Figure 3-7 (b), program-scoped relabeling, after the most occurred registers $8 is relabeled into register $1, register $7 only can be relabeled into another register $2. While, In Figure 3-7 (c), procedure-scoped relabeling, if the registers of both two procedures are used independently, registers $8 and register $7 can all be relabeled into register $1 to raise the occurrence frequencies of register

$1 to gain less 1s in code-words after 4-to-6 SS encoding for more power reduction.

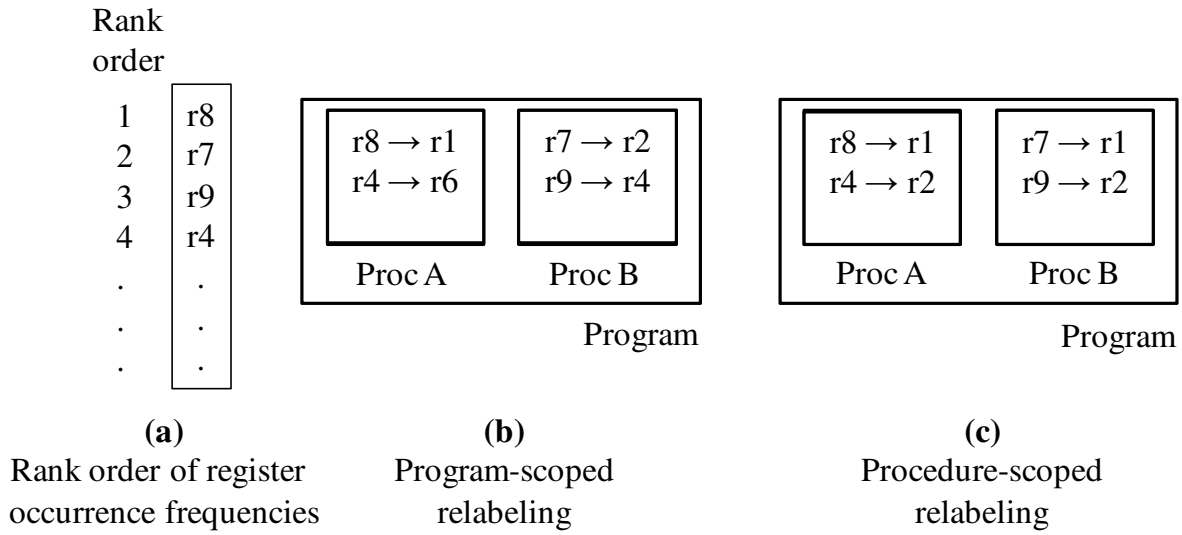


Figure 3-7：Program-scoped relabeling V.S. Procedure-scoped relabeling

Considering register usage convention and no performance degradation, some relabelabel regsiter must be relabeled consistently for the whole program to keep the correctness of program execution, while some other relabelable regsiters may be relabeled independently for each procedure. Therefore, the relabeling scopes of relabelable registers of register relabeling method 2 are classified into program-scoped and procedure-scoped.

For example, in Table 3-6, MIPS registers, temporary registers of caller-saved registers and saved registers of callee-saved registers may be further classified as procedure-scoped for register relabeling. Temporary registers are caller-saved registers in MIPS calling convention. That is, once a procedure needs to use a temporary register after procedure-call, the procedure will save the value of the temporary register before procedure-call and restore the value after procedure-call. Therefore, each procedure can use temporary registers at will. Saved registers

are callee-saved registers in MIPS calling convention. The value of saved registers must be preserved across procedures. The callee will save the values of saved registers at the procedure entry and restore at the procedure exit if it needs to use the saved registers. Therefore, each procedure can use saved registers at will after they are saved at procedure entry.

The relabeling scope of the remaining relabelable registers of MIPS is program-scope. Register $1 is reserved for assembler and thus should be relabeled for the whole program. Registers $26, $27 are reserved for kernel while they may be relabeled in application stand-alone system. The value of pointer registers, registers $28, $29 can be relabeled for the whole program since procedures recognize the same register names of pointer registers. The argument registers, register $4 - $7, which are used for arguments passing for procedures, and the return value registers, register $2 - $3 , which are used for return value from procedures, must be keep consistently across procedures; thus, they can be relabeled for the whole program.

Table 3-6：MIPS relabelabel registers with different relabeling scope

Category	Name	Number	Use	
Relabelable	$at	$1	Assembler temporary	Program-scope
	$k0 - $k1	$26 - $27	Kernel registers	
	$gp	$28	Global pointer	
	$sp	$29	Stack pointer	
	$v0 - $v1	$2 - $3	Return value	
	$a0 - $a3	$4 - $7	Argument registers	
	$t0 - $t9	$8 - $15, $24 - $25	Temporary registers	Procedure-scope
	$s0 - $s8	$16 - $23, $30	Saved registers	

In register relabeling method 2, we consider the relabeling of procedure-scoped registers and program-scoped registers together for power reduction. For program-scoped registers, the occurrence frequency of each register is collected by the same way as that mentioned in register relabeling method 1. As for the procedure-scoped registers, their occurrence frequencies have to be totaled from all procedures. Instead of summing up the frequencies of the same register numbers in different procedures, sum up the frequencies of the same rank order of occurrence frequency of register. Consequently, the relabeling of procedure-scoped registers and program-scoped registers can be together, and there are more procedure-scoped registers which may be relabeled into registers with less 1s after 4-to-6 SS encoding for more power reduction.

Therefore, the frequencies of procedure-scoped registers are gathered in each procedure separately. Then, sort the frequencies of registers in each procedure, and sum up the

frequencies of the registers with same rank in different procedures. However, in MIPS registers, saved registers should not mix with temporary registers to avoid other procedures to use the values of saved registers without saving at procedure entry and restoring at procedure exit. Hence, saved registers and temporary registers should be relabeled separately.

The steps of register relabeling method 2 are described as follows:

- Gather the occurrence frequency of each register

  For procedure-scoped registers, gather the frequencies in each procedure separately. For program-scoped registers, gather the frequencies from the whole program.

- Sorting and intermediate relabel within a procedure

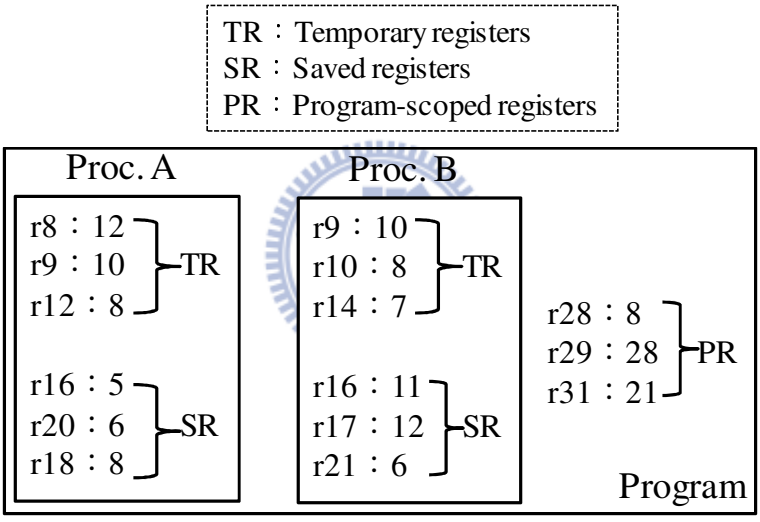
  For temporary registers, from high occurrence frequency to low occurrence frequency, relabel them to TR*n*, where *n* is the rank order according to its frequency. For saved registers, from high occurrence frequency to low occurrence frequency, relabel them to SR*n*, where *n* is the rank order according to its frequency.

- Sum up the frequencies of the same TR*n*/SR*n* of all procedures.

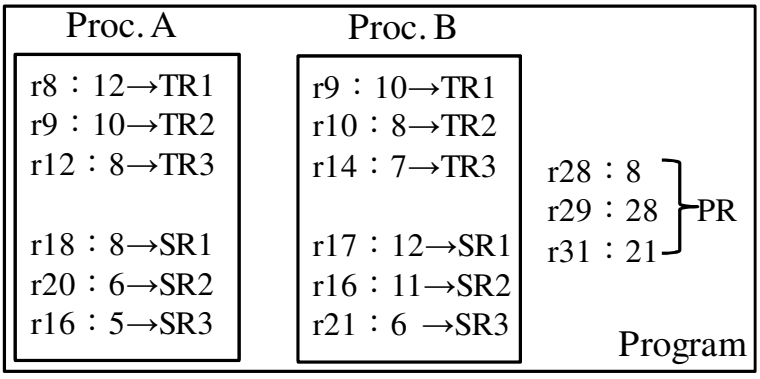
- Sort TR/SR and program-scoped registers by their occurrence frequencies, and relabel them by the sorted order according to the relabeling register selection sequence.

Figure 3-8 is an example for the relabeling steps of register relabeling method 2. Suppose that the instruction partition and the corresponding relabeling selection sequence are shown in Figure 3-4 (b) and Table 3-2, respectively. In Figure 3-8(a), a program has two

procedures, Proc. A and Proc. B, and the occurrence frequencies and categories of registers are given. Sorting and intermediate relabeling for temporary registers and saved registers separately in each procedure are shown in Figure 3-8 (b). Then, Figure 3-8 (c) shows the totals of the occurrence frequencies of the same rank of temporary registers and saved registers. In Figure 3-8 (d), sort the frequencies of TR/SR and program-scoped registers, and relabel them by the sorted order according to the register selection sequence. Finally, Figure 3-8 (e) illustrates the program after register relabeling method 2.



(a) Program before register relabeling



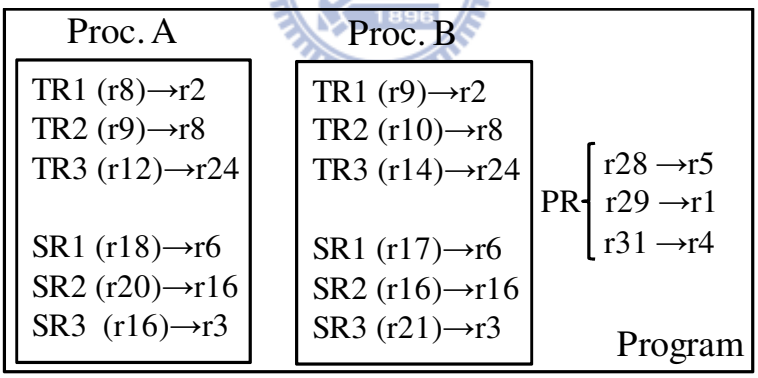
(b) Sorting and intermediate relabeling

TR1：22
TR2：18
TR3：15
SR1：20
SR2：17
SR3：11
r28：8
r29：28
r31：21

(c) Sum up the occurrence frequencies of procedure-scoped registers

r29：28 → r1
TR1：22 → r2
r31：21 → r4
SR1：20 → r6
TR2：18 → r8
SR2：17 → r16
TR3：15 → r24
SR3：11 → r3
r28：8 → r5

(d) Sort the occurrence frequencies of TR/SR and program-scoped registers, and relabel them



(e) Program after register relabeling

Figure 3-8：An example of register relabeling method 2

Chapter 4 Simulation and Analysis

Experiments have been carried out to evaluate the efficiency of our modified register relabeling method. In this chapter, benchmark programs are first introduced. Then, experimental methods which include the simulation environment, simulation method, and the simulated methods are presented. The last part of this chapter is the simulation results and the analysis of the results.

4.1 Experimental Benchmarks

We carry out experiments for six benchmark programs to evaluate the efficiency of our design in power reduction. These six DSP and numerical-computation kernels which have been heavily applied in many embedded system products with deep submicron buses that always encounter crosstalk effect are collected as benchmark programs. Table 4-1 gives a summary of these benchmark programs.

Table 4-1：Benchmark programs

Benchmark	Description
fft	Fast Fourier transform
sor	Successive over-relaxation
lu	Lower/upper triangular matrix decomposition algorithm
ej	Extrapolated Jacobi-iterative method
mmul	A matrix multiplication
tri	Tri-diagonal system solver

4.2 Experimental Methods

4.2.1 Environment

We use MIPS32 instruction set architecture in our experiments, and the experimental toolset used is the MIPS® SDE-Lite which builds the MIPS environment and generates the trace of each benchmark programs for the simulation of our method [14][16], [17]. Assume that the process technology is 90nm with the parameters based on ITRS 2004 Edition as shown in Table 4-2 [18].

Table 4-2：Device parameters for 90nm technology based on the ITRS 2004 Edition

Parameter	Value
Width	205nm
Space	205nm
Thickness	430.5nm
Height	398.5nm
Dielectric constant	3.3

The corresponding values of self-capacitance and coupling-capacitance could be taken from Predictive Technology Model (PTM) [19]. The bus length is assumed to be 14mm with 1V supply voltage. Moreover, we designed a trace driven simulator that includes 4-to-6 SS encoding scheme and our modified register relabeling to estimate the number of bit transitions on instruction bus.

4.2.2 Experimental Method

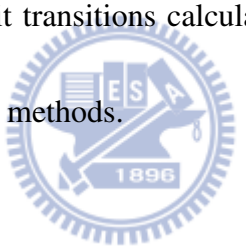
The experimental flow that includes three sub-phases of simulation method is shown in Figure 4-1. By horizontal dotted lines, Figure 4-1 can be divided into three sub-figures representing three phases where start from Code generation and statistics phase and end with Result calculation phase.

Each phase of our simulation method is described below：

- Code generation and statistics phase：This phase first generates the program execution trace for each benchmark program by adopted MIPS SDE Lite version 6.06.01 to build the MIPS ELF (executable and linkable format) image format.

Then, it scans the program execution trace to gather statistics of register usage counts for our modified register relabeling method.

- Relabeling and statistics phase：The purposes of this phase are to relabel the register names of instructions according to the register usage count statistics, and generate the relabeled program execution trace. Then, scan the relabeled program execution trace to gather statistics of the probabilities of 0s of the remaining bits of register fields and other fields for Instruction Partition step.
- Result calculation phase：The final phase includes the 4-to-6 SS encoding scheme, Instruction partition, and bit transitions calculator to evaluate the efficiency of our modified register relabeling methods.



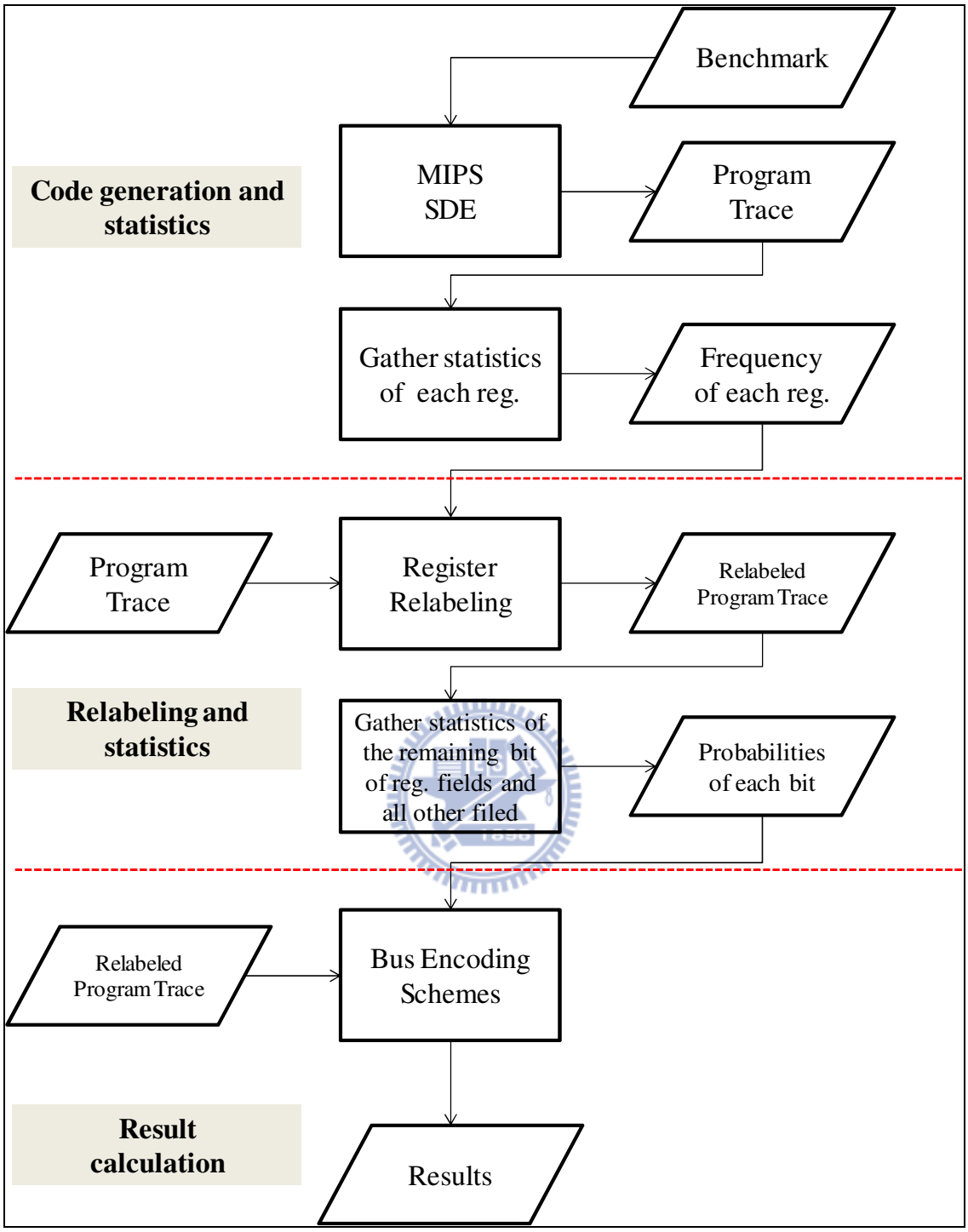


Figure 4-1：Experimental flow

4.2.3 Simulated Methods

In our simulation, we evaluate and compare the power dissipation costs and the energy consumption of the following methods:

1. Original Register Relabeling：The power reduction technique is introduced in Chapter 2 [4]. We simulate this method to compare the results between it and ours.
2. 4-to-6 Selective Shielding：This crosstalk-toggling free technique is introduced in Chapter 2 [10]. We simulate this method to compare the results between it and ours.
3. 4-to-6 Selective Shielding with original Register Relabeling：We apply original register relabeling with the 4-to-6 Selective Shielding crosstalk-toggling free technique to compare the results between this combined approach and ours.
4. 4-to-6 Selective Shielding with our modified Register Relabeling (Method 1)：This is our modified register relabeling method 1 that is applied for 4-to-6 Selective Shielding crosstalk-toggling free technique.
5. 4-to-6 Selective Shielding with our modified Register Relabeling (Method 2)：This is our modified register relabeling method 2 that is applied for 4-to-6 Selective Shielding crosstalk-toggling free technique.

4.3 Simulation Results and Analysis

The experimental results obtained from evaluating the power dissipation cost and energy consumption of the benchmark programs mentioned above are presented in this section. The energy consumption of the 4-to-6 SS encoder and decoder, and that of the bit transitions are first presented. The power dissipation cost and energy consumption of different techniques for each benchmark program are then evaluated. Finally, the effects of register usage frequency of

each benchmark program for various techniques are analyzed.

4.3.1 Hardware Overhead Analysis

For 4-to-6 Selective Shielding encoding scheme, the overhead of its encoder and decoder should be considered. From [10], the average energy consumption of the encoder and decoder of 4-to-6 Selective Shielding with 90nm TSMC technology library is 3.47 pJ for each 4-to-6 Selective Shielding and Transition Signaling encoding and decoding. The simulation environment has been described in Section 4.2.1. Thus, the corresponding values of self-capacitance and coupling-capacitance from Predictive Technology Model (PTM) [19] are 0.486 pF and 1.501 pF, respectively. According to the switching power equation introduced in Section 2.1, a single self-transition consumes 0.243 pJ, a single crosstalk 1-bit transition consumes 0.752 pJ, and a crosstalk-toggling transition consumes 3.01 pJ.

4.3.2 Energy Consumption of Different Techniques

The energy consumption by various techniques for each benchmark program is evaluated and presented in this section. There are five techniques simulated: original register relabeling (ORR), 4-to-6 Selective Shielding (4-to-6 SS), 4-to-6 SS with original register relabeling (4-to-6 SS + ORR), 4-to-6 SS with our modified register relabeling method 1 (4-to-6 SS+MRR (Method 1)), and 4-to-6 SS with our modified register relabeling method 2 (4-to-6 SS+MRR (Method 2)).

From Figure 4-2 to 4-7, the numbers of bit transitions of different types and techniques are shown for each benchmark program respectively, and Figure 4-8 shows the total number of bit transitions of different types. After 4-to-6 SS encoding scheme, the number of crosstalk-toggling transitions is guaranteed to be “0”, the number of self transitions is less than that of the un-encoded instruction bus, while the number of crosstalk 1-bit transitions is more than that of the un-encoded instruction bus.

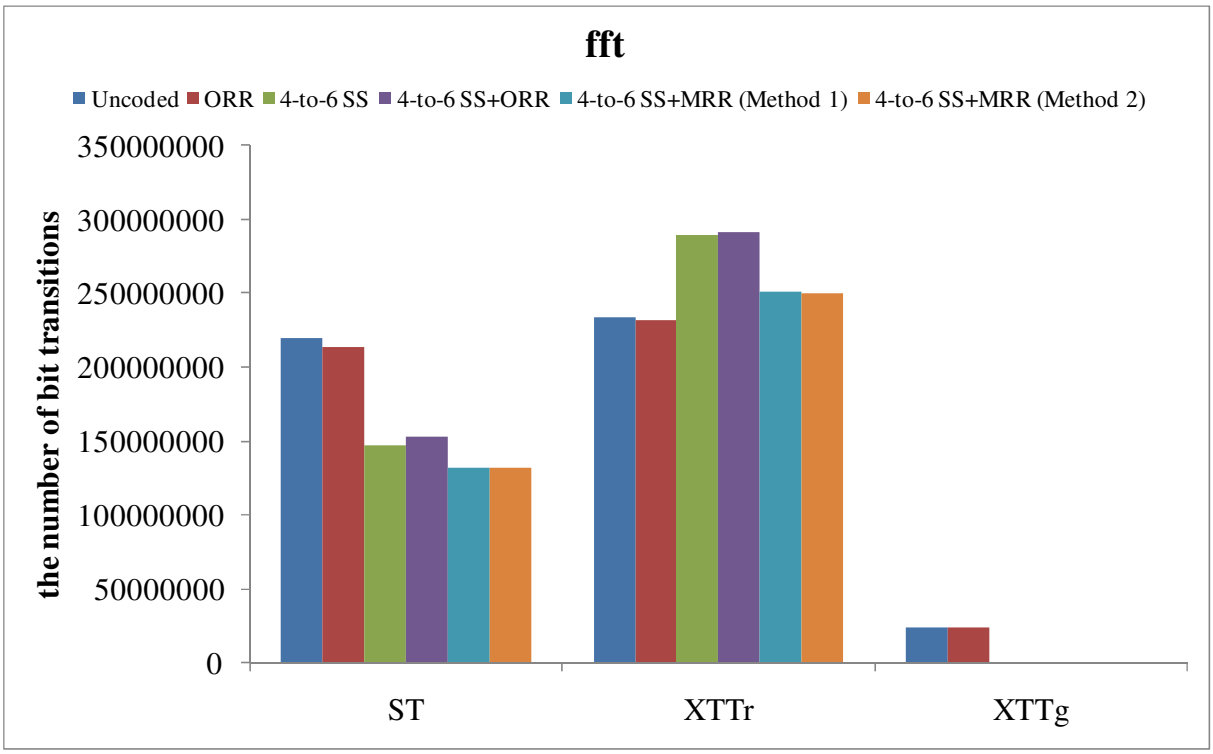


Figure 4-2：The number of bit transitions of different types and techniques (fft)

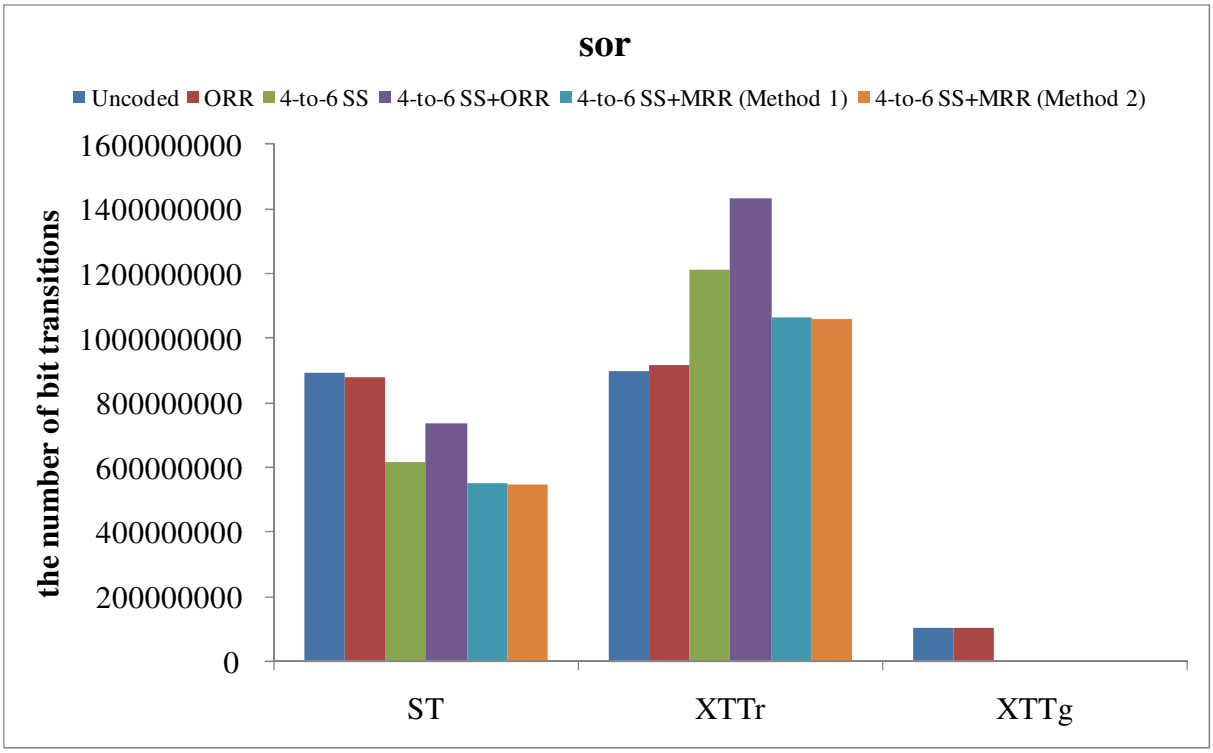


Figure 4-3：The number of bit transitions of different types and techniques (sor)

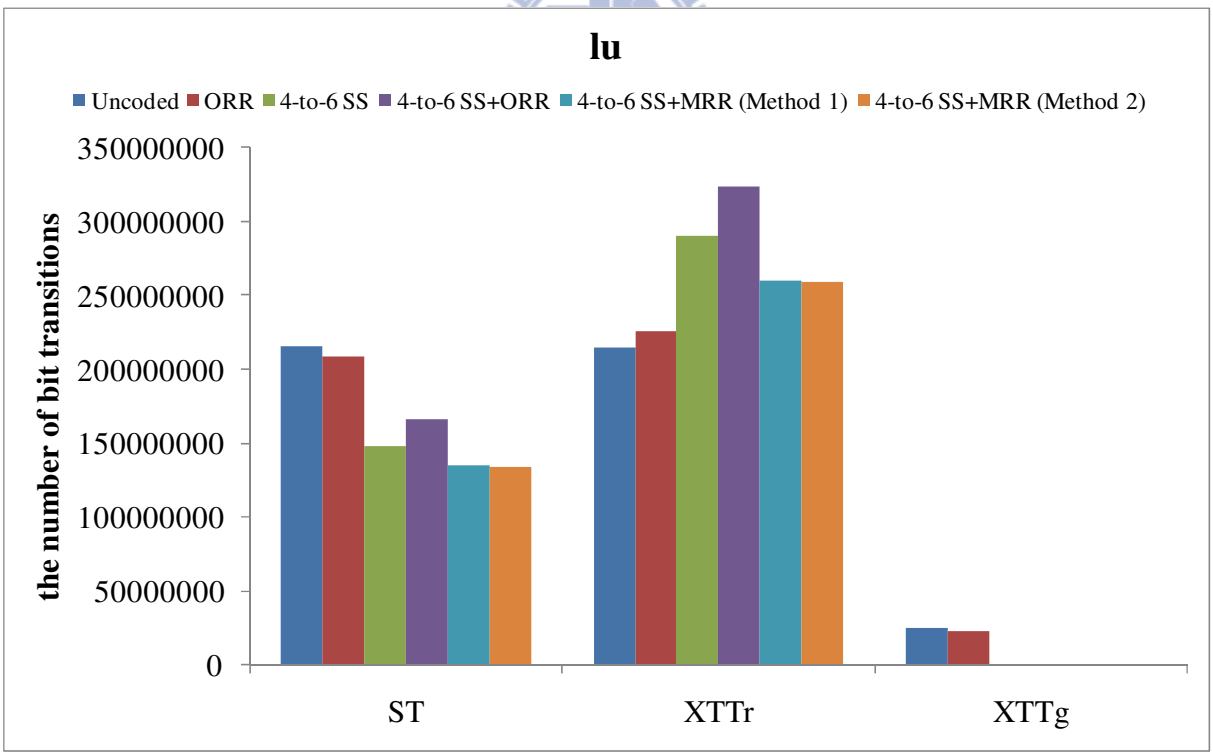


Figure 4-4：The number of bit transitions of different types and techniques (lu)

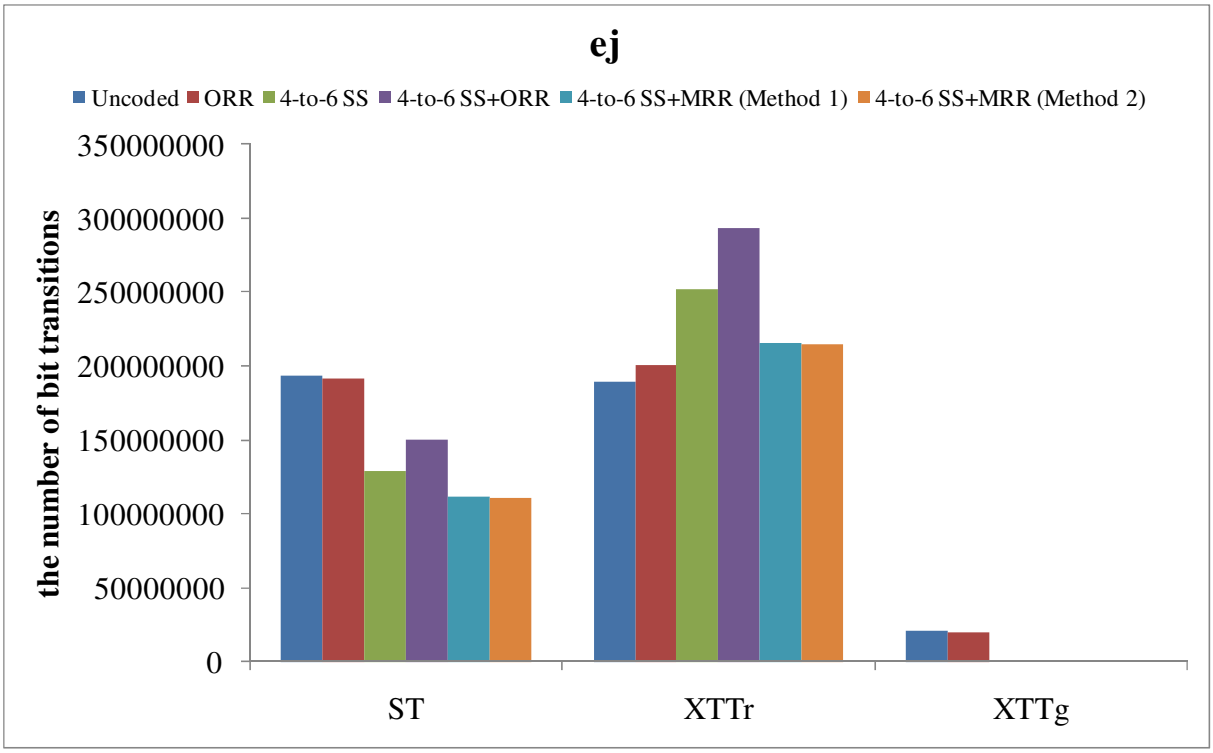


Figure 4-5：The number of bit transitions of different types and techniques (ej)

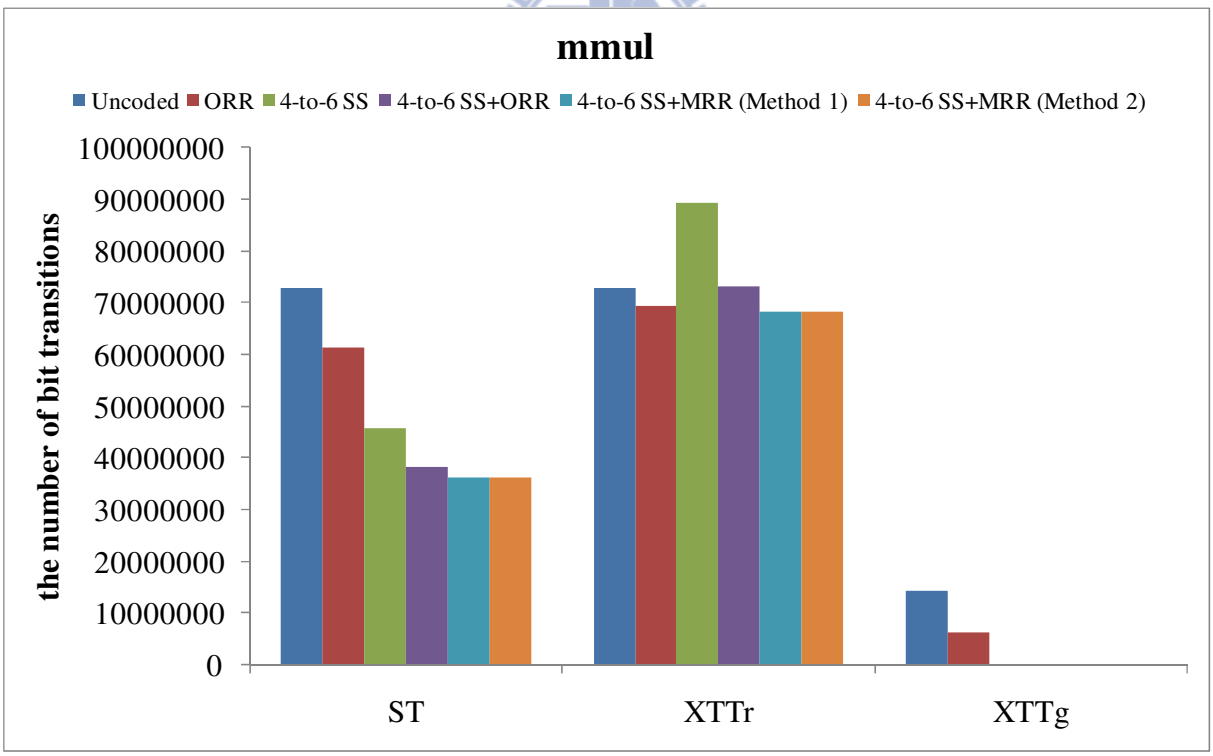


Figure 4-6：The number of bit transitions of different types and techniques (mmul)

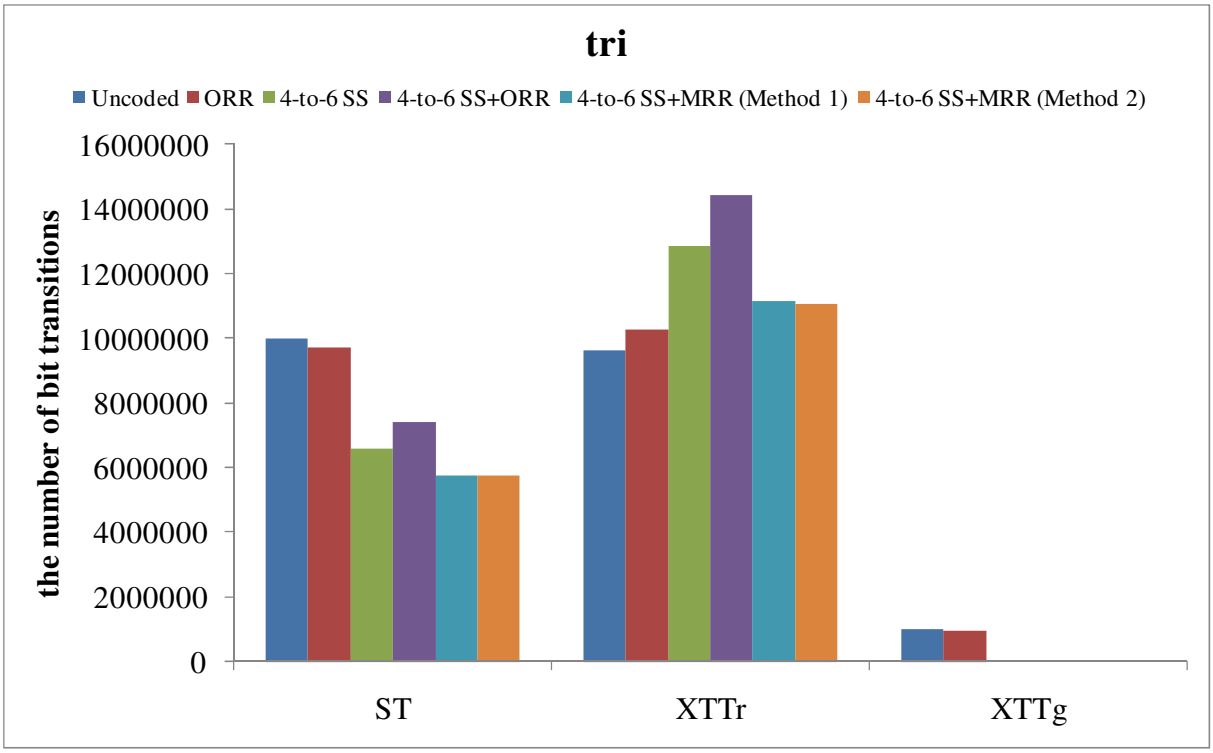


Figure 4-7：The number of bit transitions of different types and techniques (tri)

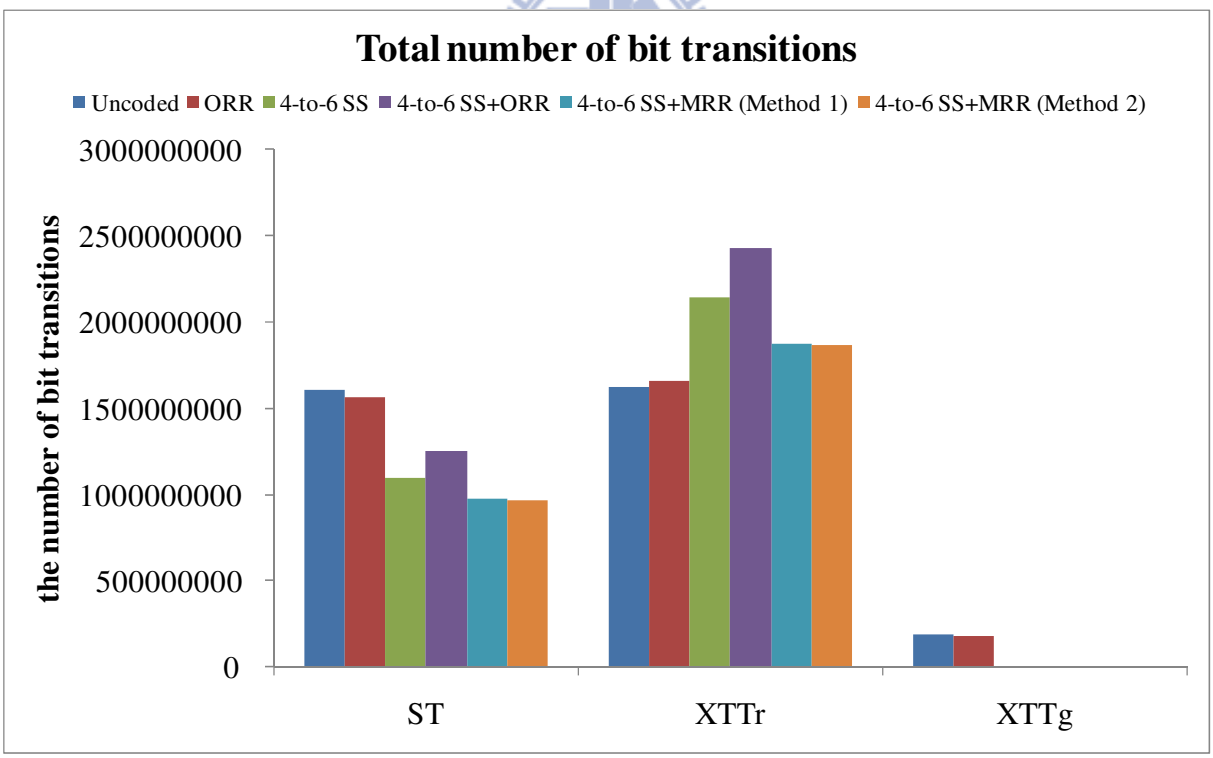


Figure 4-8：The total number of bit transitions of different types and techniques

Figure 4-9 shows the energy consumption of each benchmark program for different techniques, respectively. Note that the results are all normalized to that of the un-encoded instruction bus. Figure 4-9 shows the energy consumption of all techniques while considering the hardware overhead. Experimental results indicate that the average energy consumption of ORR is even less than that of 4-to-6 SS encoding scheme and 4-to-6 SS with original register relabeling. Recall that the 4-to-6 SS encoded bus is crosstalk-toggling free and is aimed at reducing data transmission delay. The average energy consumption of 4-to-6 SS encoding scheme is 107.4% which is even more energy consuming than un-encoded instruction bus. The average energy consumption of 4-to-6+ORR is 114.1%. It is clear that original register relabeling is not suitable for 4-to-6 SS encoding scheme. The average energy consumptions of 4-to-6 SS+MRR (Method 1) and 4-to-6 SS+MRR (Method 2) are 95.6% and 95.3%, respectively. Compared with 4-to-6 SS encoding scheme, 4-to-6 SS+MRR could save energy without extra hardware overhead and performance loss.

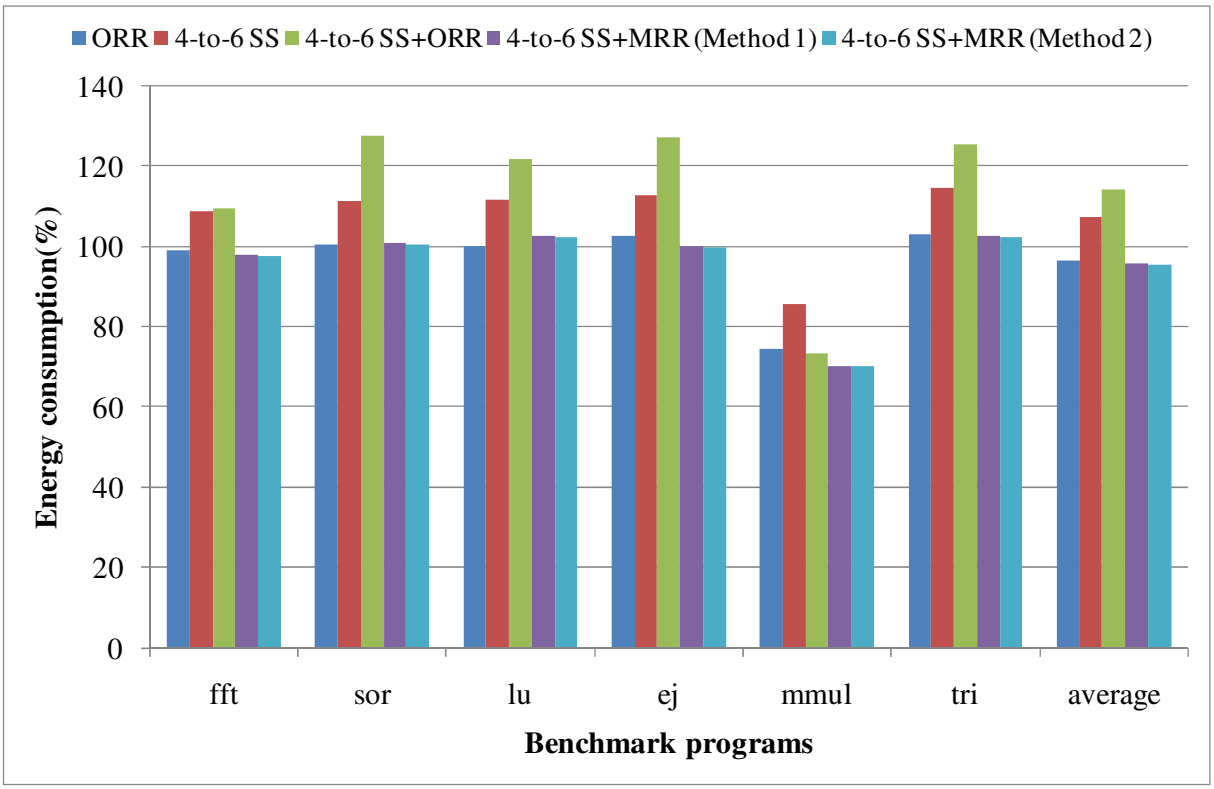


Figure 4-9：Energy consumption of different techniques

Figure 4-10 shows the average energy consumption of different bit transition types and encoder/decoder hardware of the benchmark programs for different techniques. In Figure 4-10, although the energy consumptions by bit transitions of all 4-to-6 SS are less than that of un-encoded instruction bus, the energy consumption of the encoder/decoder hardware constitutes 23.2% compared to the total energy consumed by the un-encoded instruction bus. Thus, the energy consumptions of encoder and decoder cannot be ignored.

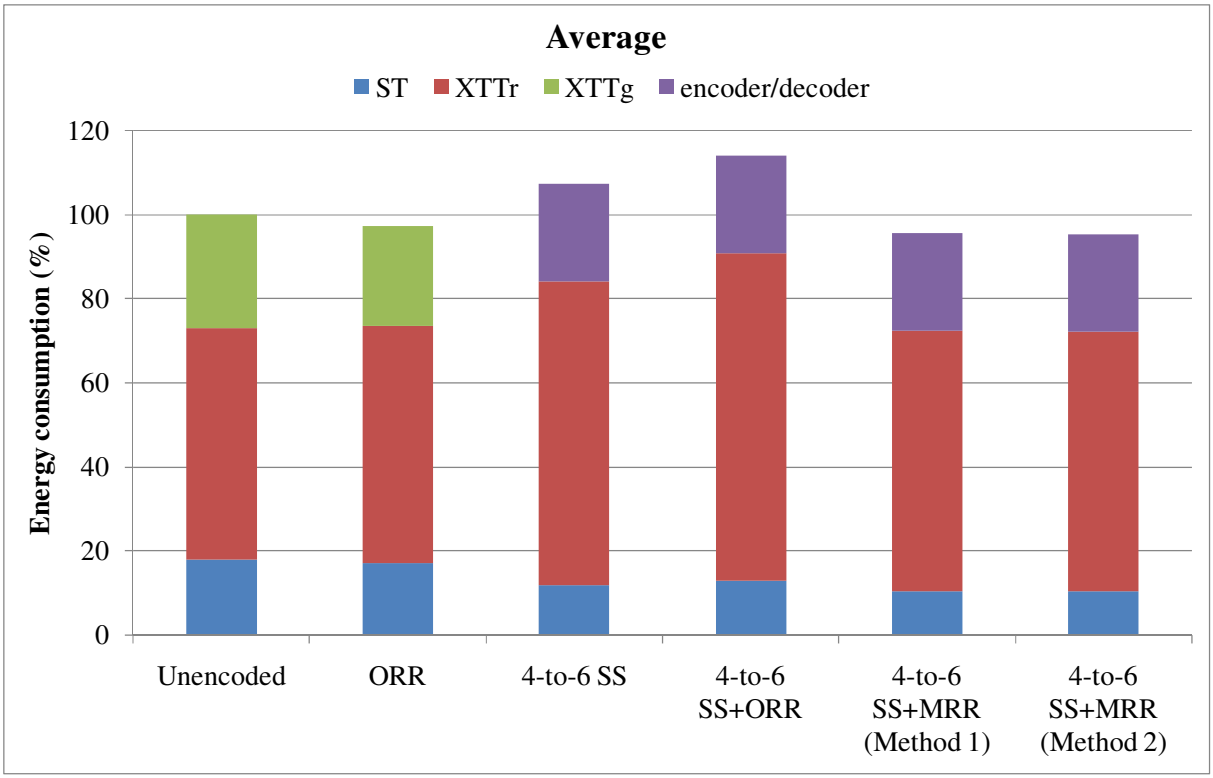


Figure 4-10：Average energy consumption of different techniques with each portion of energy consumption of transition and/or hardware

Moreover, the difference in energy consumption between 4-to-6 SS+MRR (Method 1) and 4-to-6 SS+MRR (Method 2) are quite small. The register occurrence frequencies of these two techniques deserve further discussions and will be described in the next subsection.

4.3.3 Effects of Register Occurrence Frequencies

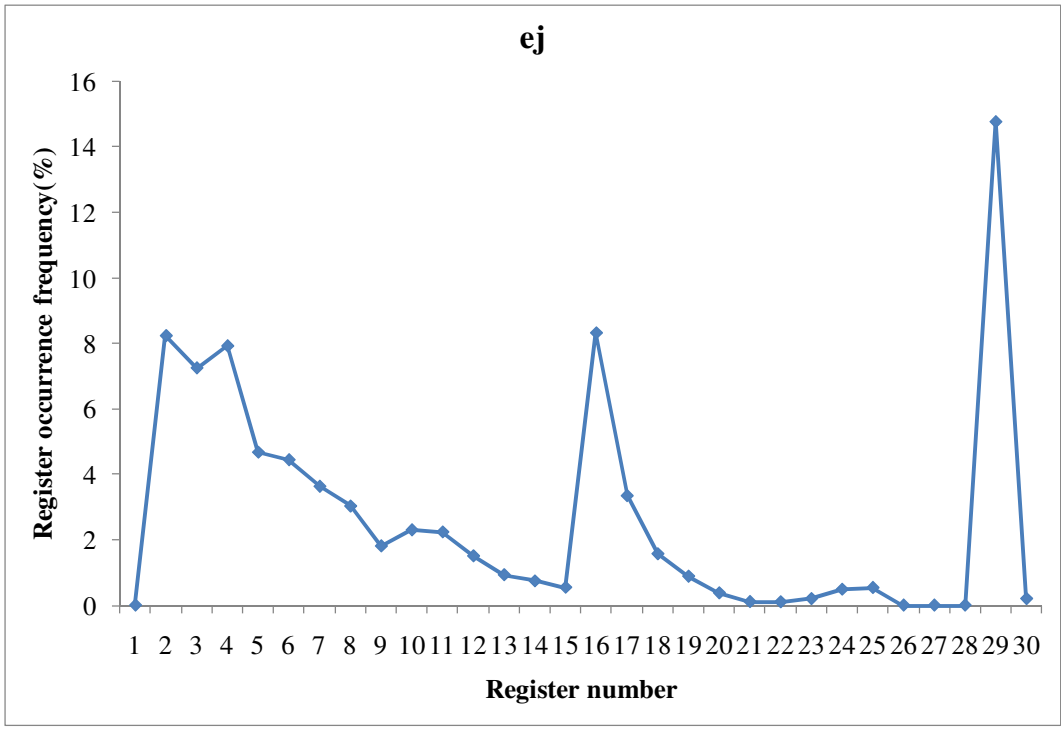
The register occurrence frequencies of different benchmark programs are collected. In the following figures, the register occurrence frequency is normalized to the total register occurrences in the trace of each benchmark program.

From Figure 4-11 to 4-16, the register occurrence frequencies and the rank order of register occurrence frequencies with both register relabeling methods of ej, lu, fft, sor, tri, and

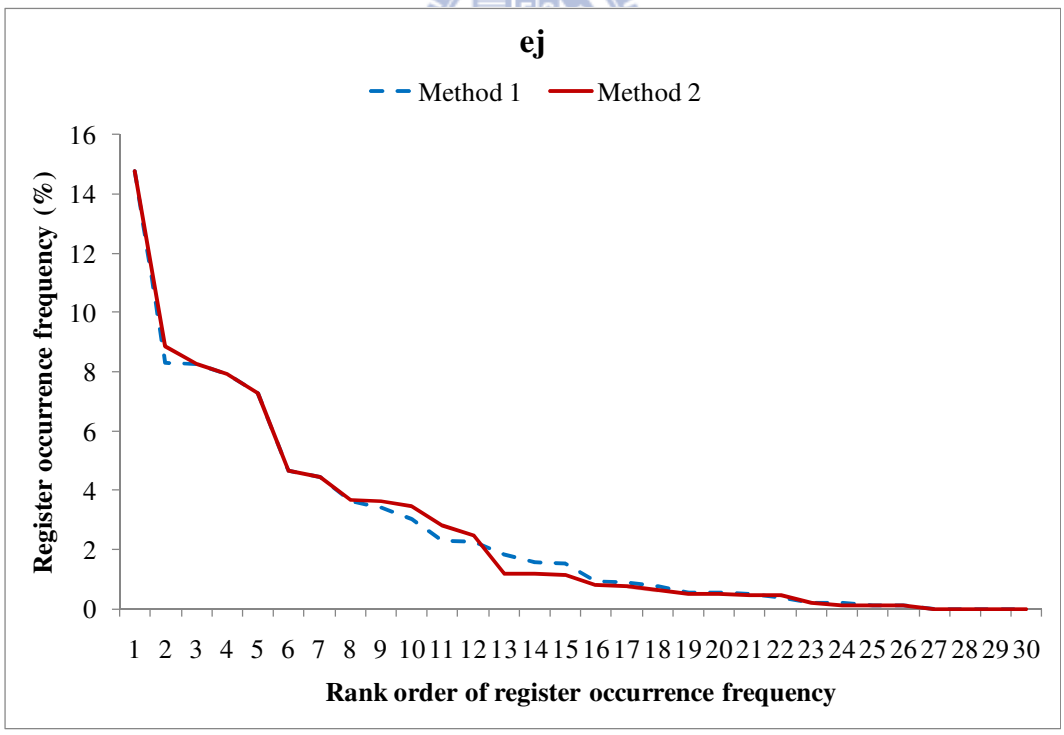
mmul programs are shown respectively, and the average values is shown in Figure 4-17.

For MIPS, in part (a) of these figures, it is clear that the register occurrences of temporary registers (register $8 - $15, $24 - $25) and saved registers (register $16 - $23, $30) are highly skewed. The reason is that MIPS compiler usually uses the smaller register numbers of a register category when they are available.

In part (b) of these figures, the register occurrence frequencies are sorted. The register occurrence frequencies shown as the blue dotted line for our register relabeling method are gathered from the whole program. The red solid line is for our register relabeling method 2. The relabeling scopes of register relabeling method 2 are program-scope and procedure-scope. The frequencies of program-scoped registers are gathered by the same way as register relabeling method 1, while the frequencies of procedure-scoped registers are gathered from each procedure separately, then, sort the frequencies in each procedure, and sum up the frequencies of the same rank registers from all procedures. However, the blue dotted line and the red solid line are very similar since the frequently occurred registers of a register category in different procedures are usually the same as the frequently occurred registers of the same category in the program. Therefore, the register occurrence frequencies of both methods are quite close and result in the small difference in energy consumption between these two methods.

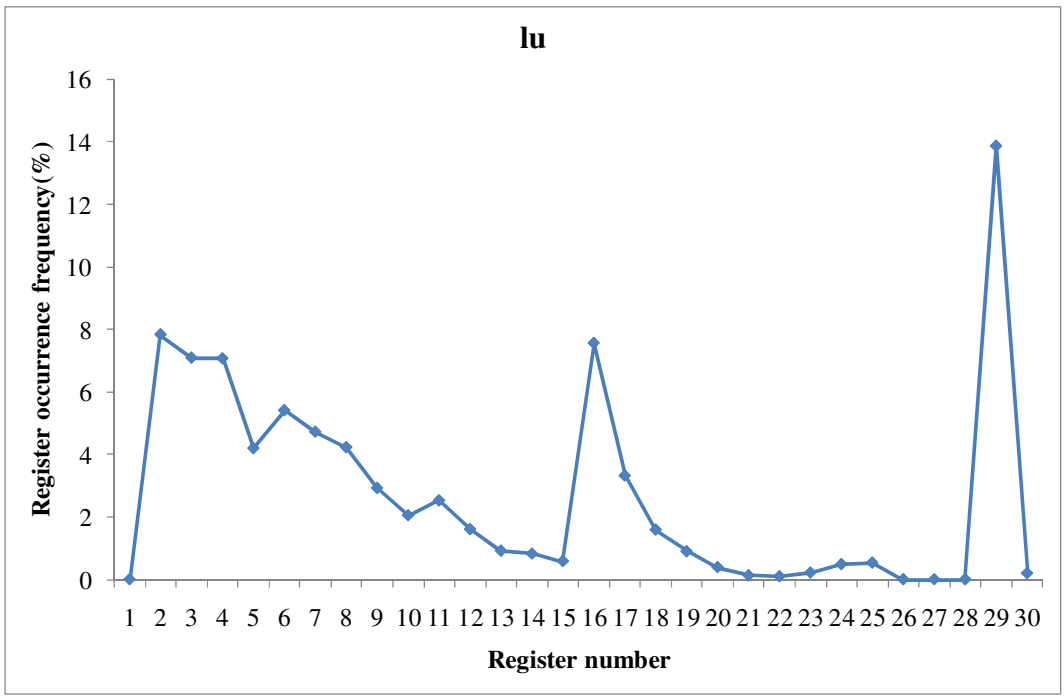


(a)

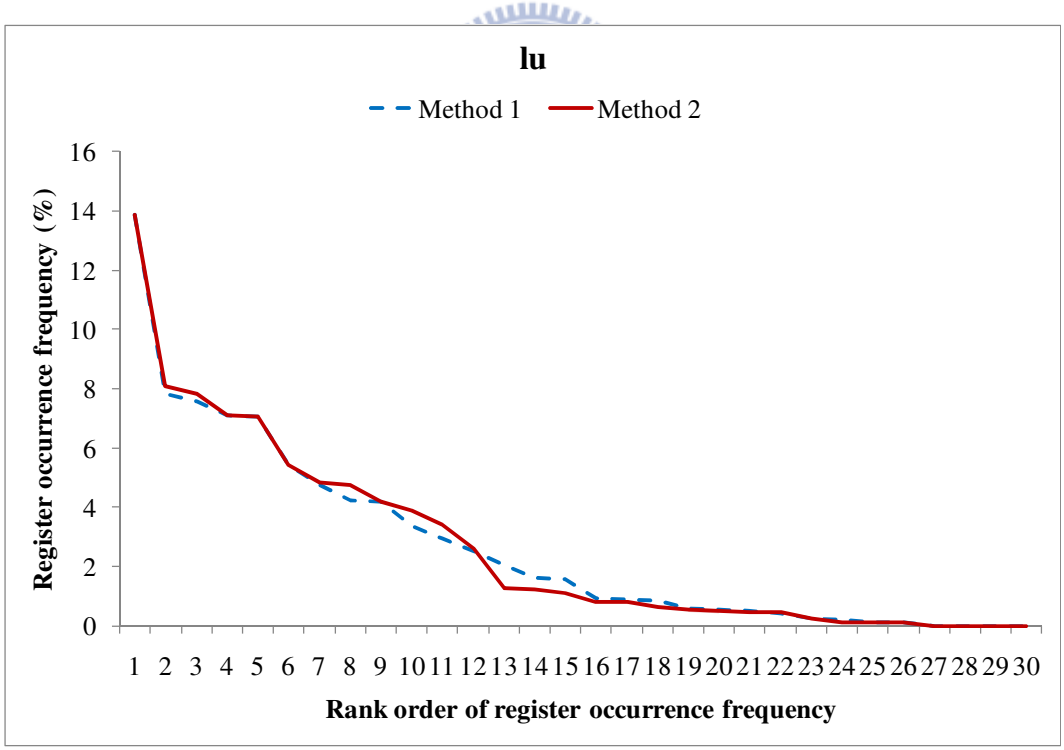


(b)

Figure 4-11：(a) Register occurrence frequency (b) Rank order of register occurrence frequency with both register relabeling methods (ej)

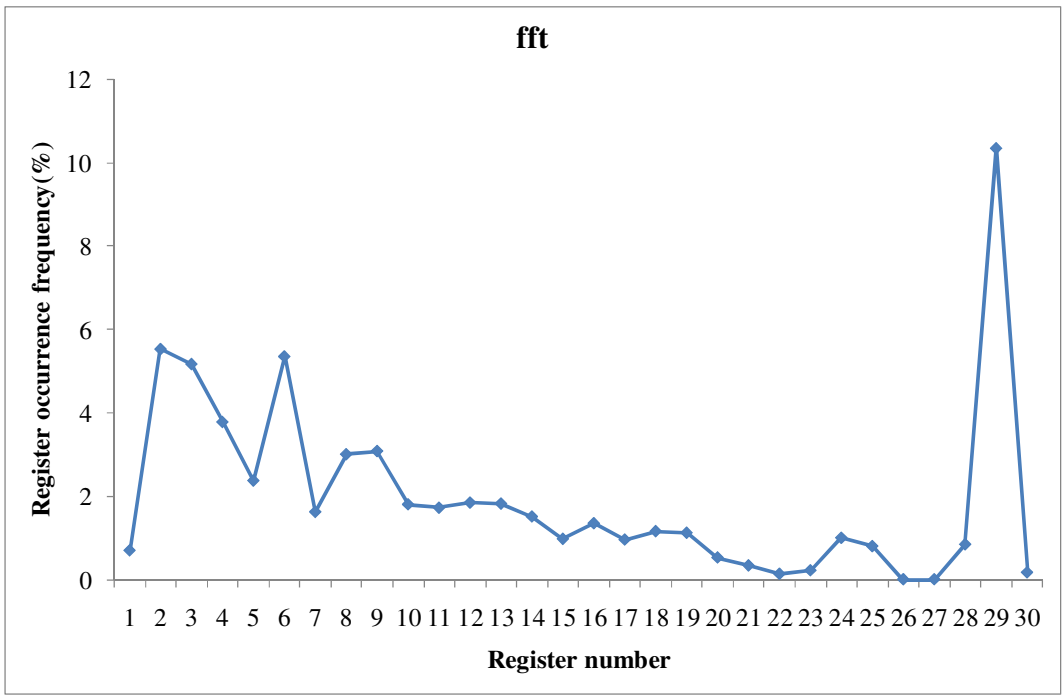


(a)

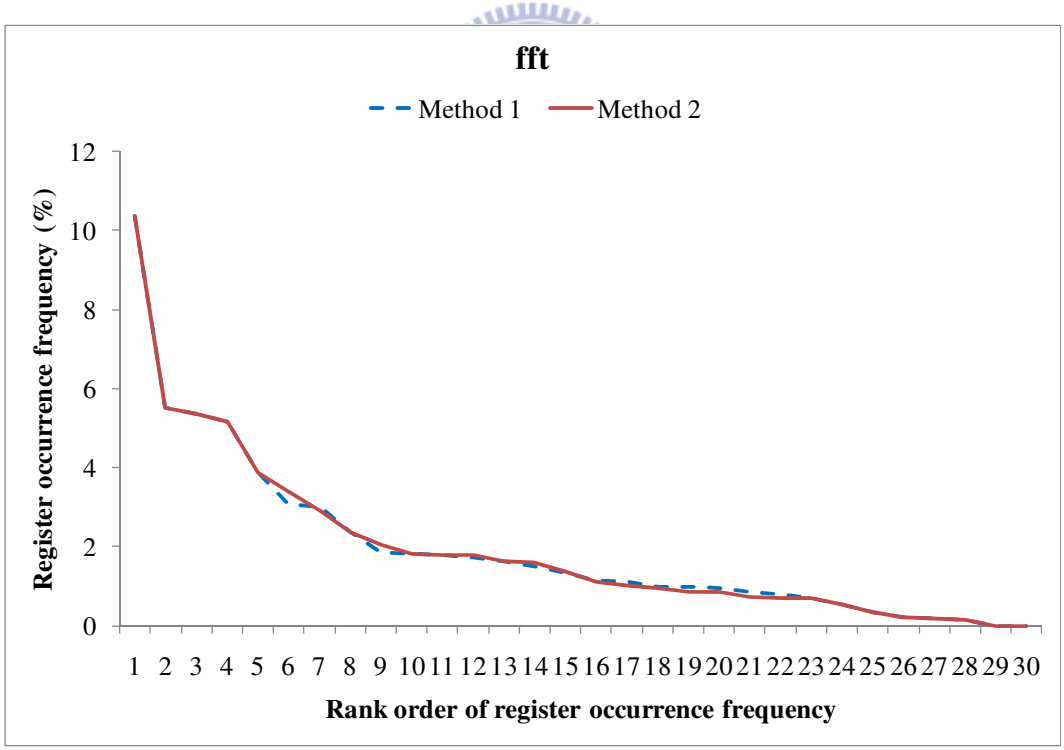


(b)

Figure 4-12：(a) Register occurrence frequency (b) Rank order of register occurrence frequency with both register relabeling methods (lu)

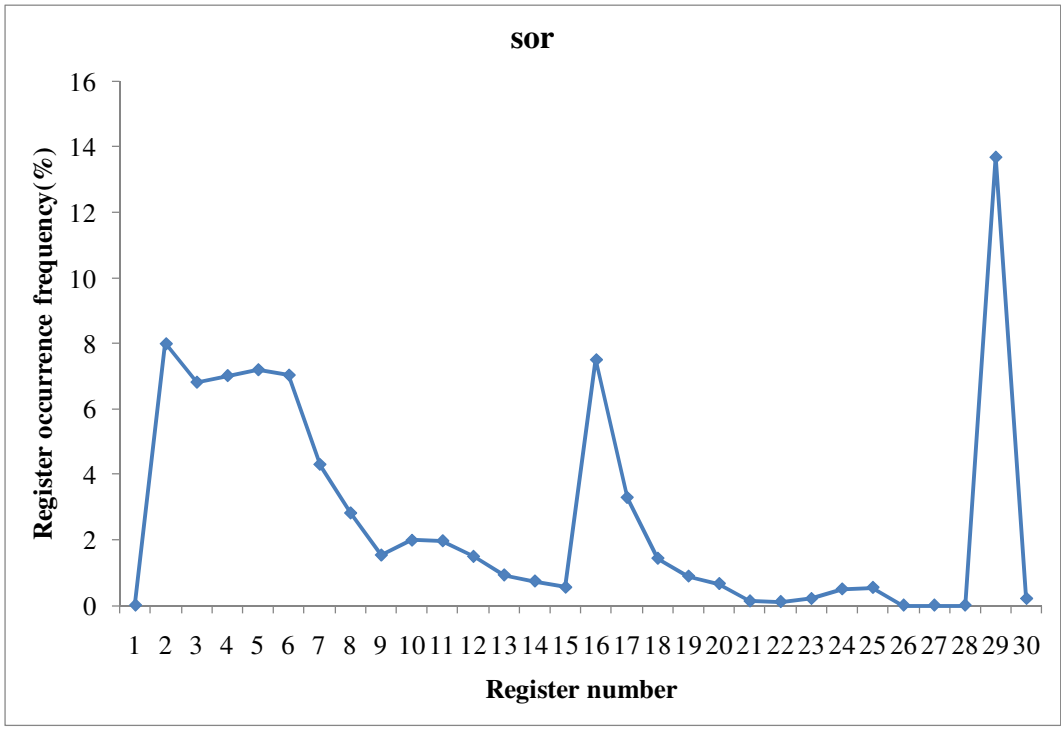


(a)

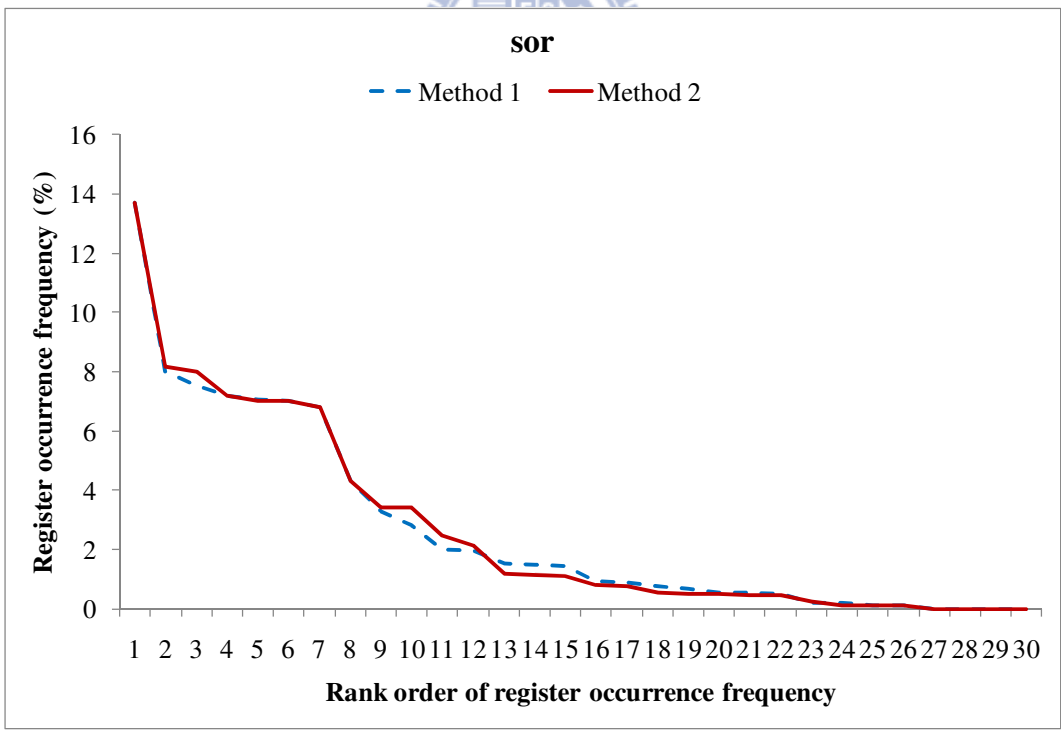


(b)

Figure 4-13：(a) Register occurrence frequency (b) Rank order of register occurrence frequency with both register relabeling methods (fft)

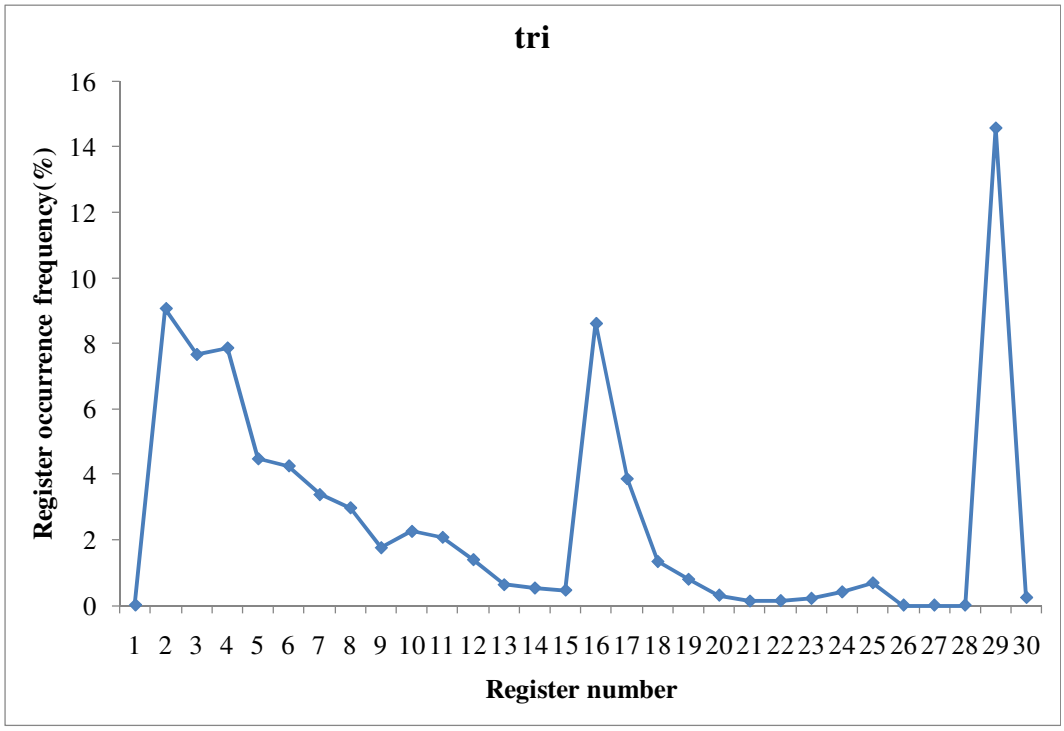


(a)

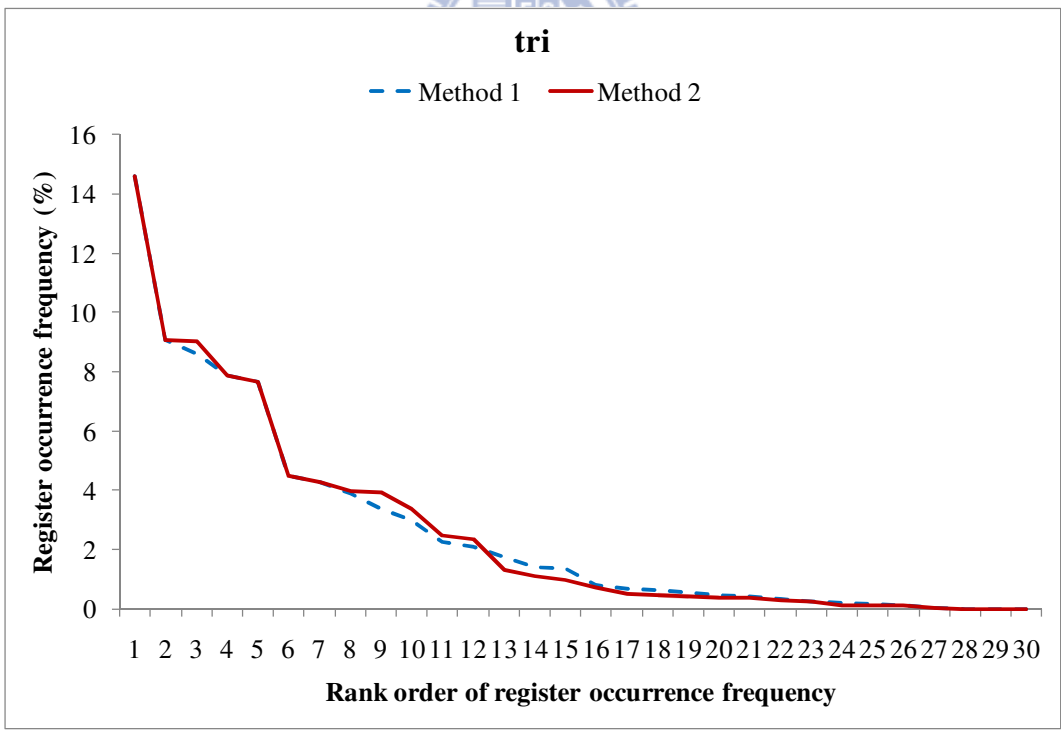


(b)

Figure 4-14： (a) Register occurrence frequency (b) Rank order of register occurrence frequency with both register relabeling methods (sor)

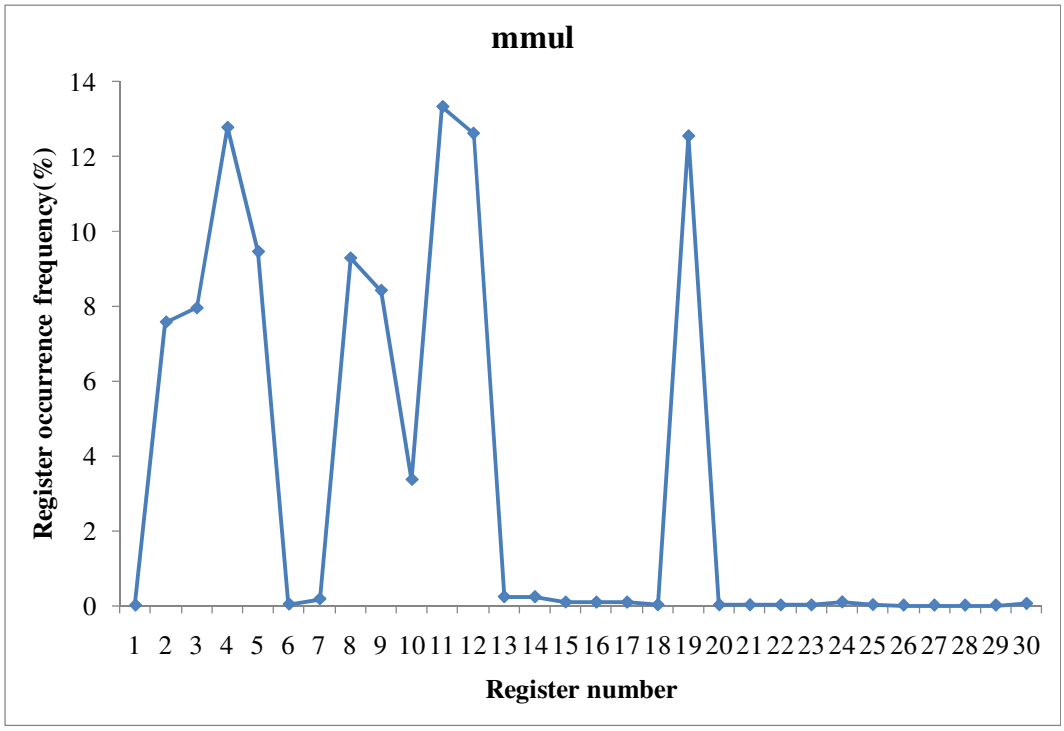


(a)

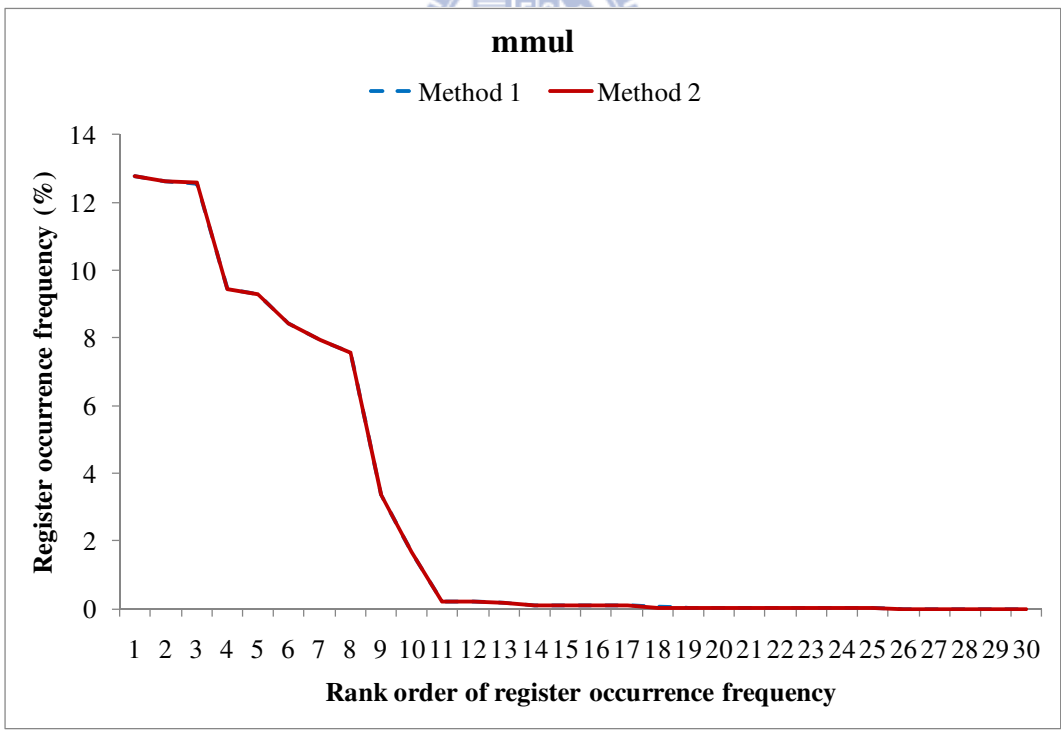


(b)

Figure 4-15： (a) Register occurrence frequency (b) Rank order of register occurrence frequency with both register relabeling methods (tri)

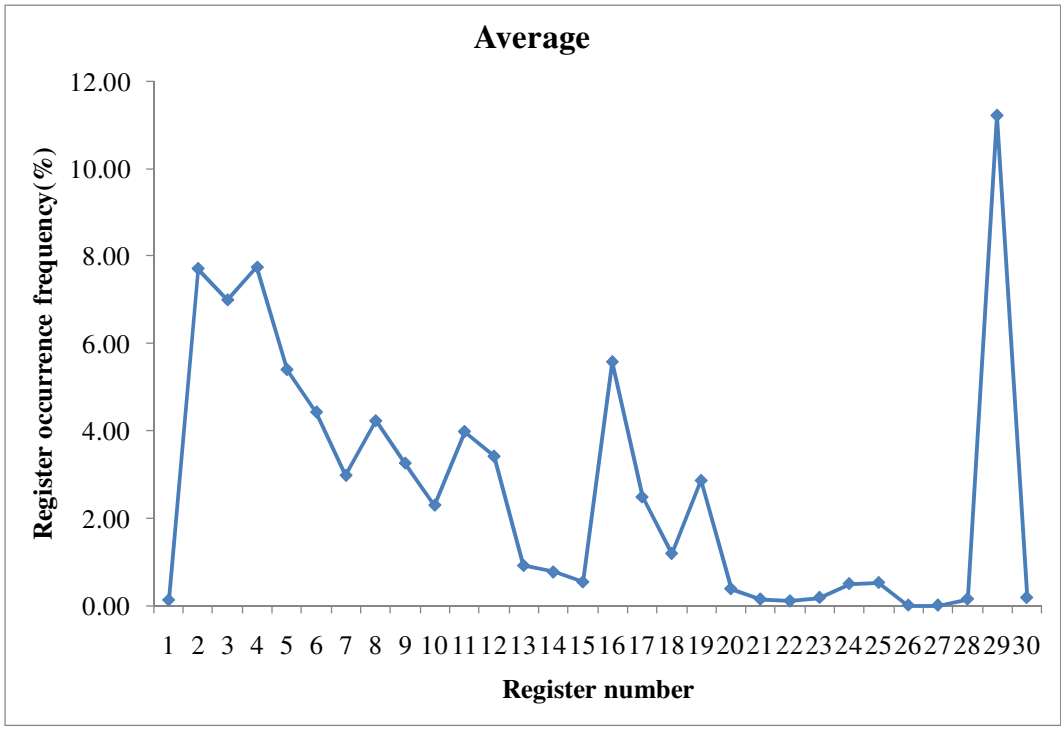


(a)

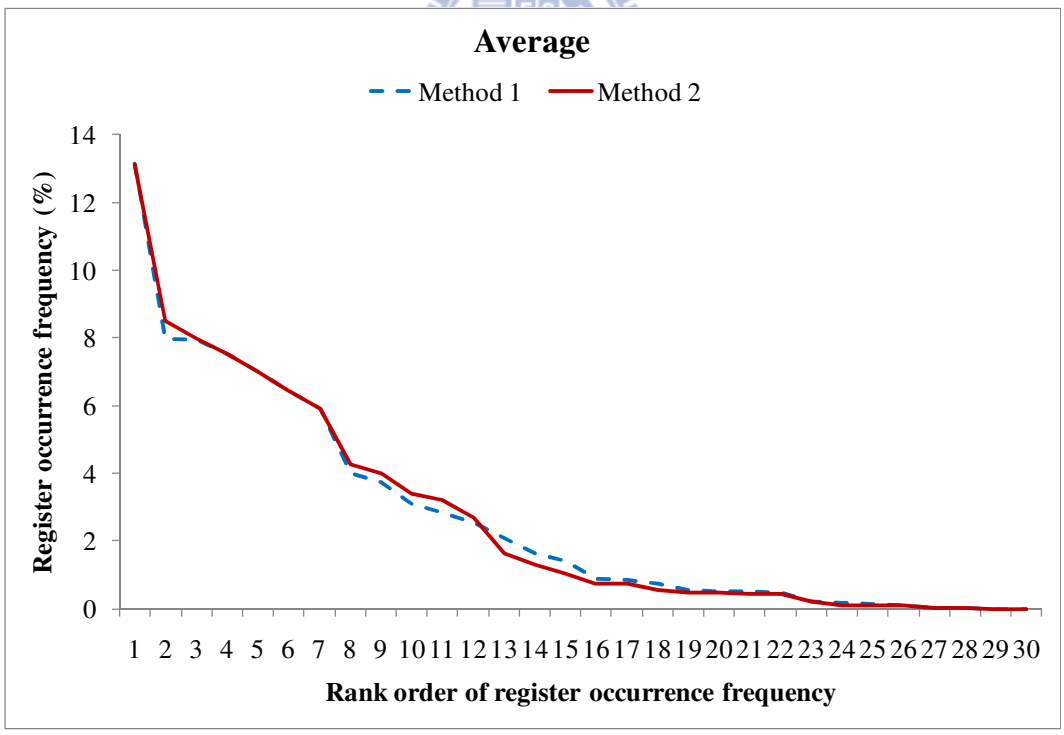


(b)

Figure 4-16：(a) Register occurrence frequency (b) Rank order of register occurrence frequency with both register relabeling methods (mmul)



(a)



(b)

Figure 4-17：(a) Register occurrence frequency (b) Rank order of register occurrence frequency with both register relabeling methods (Average)

Chapter 5 Conclusion and Future Work

While 4-to-6 SS encoding scheme results in being crosstalk-toggling free with an increase in power consumption, 4-to-6 SS encoding scheme with our modified register relabeling method is a crosstalk-toggling free and achieves a power reduction at the same time. Furthermore, we also discover that combining with 4-to-6 Selective Shielding encoding scheme might reduce the complexity of register relabeling algorithms. While the original register relabeling algorithm considers the relationship between register fields of consecutive instructions, our modified register relabeling method considers only the register number itself, which is more straight-forward than the original one.

Simulation results show that the overall average energy consumption of 4-to-6 SS encoding scheme with our modified register relabeling is 95.3%, 12.1% less than 4-to-6 SS encoding scheme and 18.8% less than 4-to-6 SS encoding scheme with original register relabeling.

There are several related researches deserving further discussions. On the one hand, associating other compiler techniques such as register allocation with our design may be able to further reduce the overall power consumption with crosstalk-toggling free. On the other hand, adjusting the scope of register relabeling to a smaller one such as extended basic-block or basic-block level might provide more opportunities to use the register with less 1s after 4-to-6 SS encoding scheme result in fewer transitions for power reduction.

Therefore, these techniques must be analyzed deeply and thus can be modified to suitable crosstalk-toggling free bus encoding schemes to further reduce power consumption.

Reference

[1] Petrov, P.; Orailoglu, A.; , "Low-power instruction bus encoding for embedded processors," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* , vol.12, no.8, pp. 812- 826, Aug. 2004.

[2] Lindkvist, T.; Lofvenberg, J.; Gustafsson, O.; , "Deep sub-micron bus invert coding," *Signal Processing Symposium, 2004. NORSIG 2004. Proceedings of the 6th Nordic* , vol., no., pp. 133- 136, 2004.

[3] Sotiriadis, P.P.; Chandrakasan, A.; , "Reducing bus delay in submicron technology using coding," *Design Automation Conference, 2001. Proceedings of the ASP-DAC 2001. Asia and South Pacific* , vol., no., pp.109-114, 2001.

[4] Mehta, R.; Owens, R.M.; Irwin, M.J.; Chen, R.; Ghosh, D.; , "Techniques for low energy software," *Low Power Electronics and Design, 1997. Proceedings., 1997 International Symposium on* , vol., no., pp. 72- 75, 18-20 Aug 1997.

[5] N. H. E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, Addison Wesley, 1994.

[6] Chandrakasan, A.P.; Brodersen, R.W.; , "Minimizing power consumption in digital CMOS circuits," *Proceedings of the IEEE* , vol.83, no.4, pp.498-523, Apr 1995.

[7] Arunachalam, R.; Acar, E.; Nassif, S.R.; , "Optimal shielding/spacing metrics for low power design," *VLSI, 2003. Proceedings. IEEE Computer Society Annual Symposium on* , vol., no., pp. 167- 172, 20-21 Feb. 2003.

[8] Victor, B.; Keutzer, K.; , "Bus encoding to prevent crosstalk delay," *Computer Aided Design, 2001. ICCAD 2001. IEEE/ACM International Conference on* , vol., no., pp.57-63, 2001.

[9] Mutyam, M.; , "Preventing crosstalk delay using Fibonacci representation," *VLSI Design,*

2004. Proceedings. 17th International Conference on , vol., no., pp. 685- 688, 2004.

[10] Mutyam, M.; , "Selective shielding technique to eliminate crosstalk transitions," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 14, no. 3, pp. 1-20, 2009.

[11] Mutyam, M.; , "Selective shielding: a crosstalk-free bus encoding technique," *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on* , vol., no., pp.618-621, 4-8 Nov. 2007.

[12] M. R. Stan and W. P. Burleson, "Limited-Weight Codes for Low Power I/O," 1994.

[13] Chin-Tzung Cheng; Wei-Hau Chiao; Jean Jyh-Jiun Shann; Chung-Ping Chung; Wen-Feng Chen; , "Low-power BIBITS encoding with register relabeling for instruction bus," *VLSI Design, Automation and Test, 2005. (VLSI-TSA-DAT). 2005 IEEE VLSI-TSA International Symposium on* , vol., no., pp. 41- 44, 27-29 April 2005.

[14] MIPS Technologies, *MIPS Architecture For Programmers Volume I-A: Introduction to the MIPS32 Architecture*. 2010.

[15] R. Britton, *MIPS Assembly Language Programming*, Prentice Hall, 2003.

[16] MIPS Technologies, *MIPS SDE 6.x Programmers' Guide*. 2007.

[17] D. Sweetman, *See MIPS Run*, Morgan Kaufmann, 1999.

[18] "ITRS Home." [Online]. Available: http://www.itrs.net/.

[19] "Predictive Technology Model (PTM)." [Online]. Available: http://ptm.asu.edu/.