

國立交通大學

資訊科學與工程研究所

碩士論文

減少傳送次數之窄匯流排編碼

Narrow Bus Encoding to Reduce Bus Transactions



研究生：鄭式勳

指導教授：鍾崇斌 教授

中華民國 九十五年 八月

減少傳送次數之窄匯流排編碼

Narrow Bus Encoding to Reduce Bus Transactions

研究生：鄭式勳

指導教授：鍾崇斌

Advisor : Chung-Ping Chung

國立交通大學

資訊科學與工程研究所



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

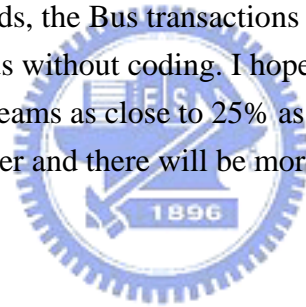
June 2006

Hsinchu, Taiwan, Republic of China

中華民國九十五年八月

Abstract

Narrow Bus is a Bus with narrower width than data width. The Bus with the same width with data width is called normal Bus. This thesis will focus on 8-bit width Bus and 32-bit width data environment. Using narrow Bus can save chip area but the Bus transactions will be raised. Most area constraint systems hope to use narrow Bus but they cannot tolerate the performance loss and extra system static energy consumption caused by additional Bus transactions. I add some functions to the transmitter and corresponding receiver to reduce Bus transactions on narrow Bus. I will adopt suitable coding methods to the proposed transmitter depending on the stream type. In instruction address stream, I use T0-C to utilize the sequential property and narrow Bus DAT to utilize the branch behavior property. In data address stream, variable stride algorithm, historical addresses algorithm and the idea of SRWEC (Separated Read/Write Encoding Contents) are used. In instruction stream, a code compression algorithm is used to reduce the instruction size. In data stream, the idea of SRWEC, the occurrence of small value and the relationship between data are utilized. The number of narrow Bus transaction is four times of the number of normal Bus transactions. In other words, the Bus transactions of normal Bus are 25% of the Bus transactions of narrow Bus without coding. I hope to reduce the number of narrow Bus transactions of streams as close to 25% as possible so that the extra Bus transactions overhead is slighter and there will be more systems are willing to use narrow Bus.



摘 要

窄匯流排是指寬度比資料寬度要小的匯流排，而和資料寬度等寬的匯流排則稱一般匯流排，這篇論文將針對使用 8-bit 寬的匯流排來傳送 32-bit 寬的資料的環境，使用窄匯流排可以節省晶片面積但是卻需要較多的傳送次數，大部分著重面積考量的系統希望能使用窄匯流排卻無法容忍效能和額外能量的損失，我新增了些功能給傳送端和接收端以減少在窄匯流排上的傳送次數，針對不同的資訊流，我將提出不同的方法，在指令位址流上，我使用 T0-C 以利用其連續的特性而使用 NBDAT 去利用其不連續的特性，在資料位址流上，我使用 variable stride algorithm、historical address algorithm 和 SRWEC (Separated Read/Write Encoding Contents) 的概念，在指令流上，我使用指令壓縮的技巧來減少傳送次數，在資料流上，我使用 SRWEC 和一些減少資料大小的概念以減少傳送次數，在我的環境下，一筆資料若不編碼將需要四次傳送，因此，我的編碼的最低傳送次數比例是 25%，我將儘可能將傳送次數減少至 25%，如此一來，將有更多系統會願意使用窄匯流排。



誌 謝

感謝鍾崇斌老師和單智君老師這兩年的指導，實驗室裡學長和同學的意見和幫助，最重要的是有家人的支持與鼓勵，由於大家的幫助，我才得以完成此篇論文，僅在此上我的謝意。

祝所有幫助過我的人，身體健康，萬事如意。
鄭式勳 2006 年 9 月 2 日



Chapter 1: Introduction	1
1.1 Narrow Bus Environment and Normal Bus Environment.....	1
1.2 Research Observation, Motivation and Objective	1
1.3 The Environment and Assumptions	2
1.4 Organization	2
Chapter 2: Background.....	2
2.1 Redundant and Effective Fragments.....	3
2.1.1 Redundant bits:.....	3
2.1.2 Effective Bits:	4
2.1.3 Effective Fragments and Redundant Fragments	4
2.2 Related Coding Methods	4
2.2.1 Asymptotic Zero-Transition Activity Encoding for Address Busses in Low-Power Microprocessor-Based Systems (T0).....	5
2.2.2 Irredundant Address Bus Encoding for Low Power (T0-C).....	6
2.2.3 Discontinuous Address Table Algorithm (DAT)	8
2.2.5 Variable Stride Algorithm.....	10
2.2.4 Idea of Separating Read and Write Streams.....	10
2.2.5 Selective Instruction Compression for Memory Energy Reduction in Embedded Systems	12
Chapter 3: Design of Proposed Coding	13
3.1 Coding Flow.....	13
3.2 VL-Encoder.....	14
3.2.1 Method One : Adding the Information of the Number of Bus Transactions at the Beginning of First Transmission	14
3.2.2 Method Two : Indicating the Codeword is Transmitted over or not at Every Bus Transaction.....	15
3.2.3 Comparison of Method One and Method Two.....	15
3.2.4 The VL-Encoder Using Method Two	17
3.3 Instruction Address Stream	17
3.3.1 Type of Redundant Bits Used by EB VL-Encoder :.....	17
3.3.2 Encoder :	17
3.3.3 Proposed Methods :.....	18
3.4 Data Address Stream	19
3.4.1 Type of Redundant Bits Used by EB VL-Encoder :.....	19
3.4.2 Encoder :	19
3.4.3 Proposed Methods :.....	23
3.5 Instruction Stream	24
3.6 Data Stream.....	25

3.6.1 Type of Redundant Bits Used by EB VL-Encoder :.....	25
Chapter 4: Statistics and Simulation Result.....	27
4.1 Statistics and Simulation Environment.....	27
4.2 Statistics and Simulation Result of Instruction Address Streams.....	27
4.2.1 Statistics	27
4.2.2 Simulation Result.....	28
4.3 Statistics and Simulation Result of Data Address Streams	32
4.3.1 Statistics	32
4.3.2 Simulation Result.....	32
4.4 Statistics and Simulation Result of Instruction Streams	35
4.4.1 Statistics	35
4.4.2 Simulation Result.....	36
4.5 Statistics and Simulation Result of Data Streams	37
4.5.1 Statistics	37
4.5.2 Simulation Result.....	37
4.6 The Simulation Results and Comments of these Streams.....	39
4.6.1 Instruction Address.....	39
4.6.2 Data Address	39
4.6.3 Instruction	40
4.6.4 Data	40
4.6.5 Summary.....	41
Chapter 5: Summary, Discussion, and Conclusion.....	41
5.1 The Effect of the Narrow Bus Encoding.....	41
5.2 The Coding Methods Effects on Bit Toggles.....	42
5.3 The Coding Methods Effects on Performance.....	42
5.4 The Other Bus Architectures	43
5.5 The coding (encoding and decoding) delay affection on memory access stage:	44
5.6 Reducing the coding delay effect on pipeline stage:.....	45
Q&A	46

Chapter 1: Introduction

In modern computer architecture design, the area, energy consumption, and performance are some main considering factors. However, these factors are conflict with each other on design in most time. In this chapter, I will introduce and compare the environment of narrow Bus and normal Bus on these factors. And then I will describe my observation, motivation and objective. On the end of this chapter is the organization of this thesis.

1.1 Narrow Bus Environment and Normal Bus Environment

The narrow Bus is a Bus with narrower width than data width. The normal Bus is a Bus with the same width with data width. In intrinsic characteristic, the narrow Bus will have less routing area and lower routing complexity than normal Bus. However, if a system uses narrow Bus with one-fourth width of data width, it needs four times Bus transactions than normal Bus to transmit data. For example, if a system uses 8-bit width Bus to transmit 32-bit width data, it needs to send 8-bit fragments four times. (This thesis will focus on 8-bit width Bus and 32-bit width data environment.) This will cause system performance loss and more static energy consumption on devices. Basing on this intrinsic characteristic difference, the time constraint systems will use normal Bus and the area constraint systems will prefer narrow Bus.

1.2 Research Observation, Motivation and Objective

There should be many area constraint systems hope to use narrow Bus to save routing area; however, they may give up because they cannot tolerate the severely performance loss and additional static energy consumption caused by extra Bus transactions.

In this thesis, I assume the environment is using 8-bit width Bus to transmit 32-bit width data. In this situation, the number of Bus transactions will rise to four times. However, there are some opportunities can be used to reduce Bus transactions. First, the sender can skip the regular portion of a codeword without sending, and inform the receiver what the regularity is. Second, when the data is regular, the sender can use a protocol to inform the receiver what the regularity is instead of sending the data.

As mentioned before, the different systems will have different constraint on area, performance and energy consumption. A system which decides to use narrow Bus should be a high area constraint system. I hope to reduce the additional Bus

transactions by tolerable area overhead of coding logic gates and extra control lines. The different systems can tolerate different area overhead, so I will propose several methods for systems to use without proposing so-called “Best Choice”. My objective is using narrow Bus encoding to reduce Bus transactions at narrow Bus environment. In this way, there should be more system willing to use narrow Bus.

1.3 The Environment and Assumptions

In this thesis, I will deal with the traffic between processor and memory. The processor and memory communicate by external Bus and there is no cache in this system. I use 8-bit width external Bus to transmit 32-bit width data (address). Figure 1-1 is the architecture of my environment

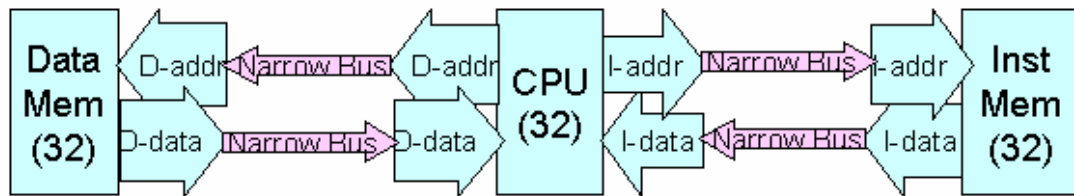


Figure 1-1: Narrow Bus Architecture

The area overhead caused by coding logic gates depends on the processing technology. The area overhead caused by extra control lines depends on the processing technology and routing length. Both of them are decided case by case. It is hard to evaluate the weighting of them. Base on my environment, I simplify the area overhead to the number of extra control lines. In other words, I assume the area overhead of coding logic gates is much slighter than extra control lines. Of course, system designers can get the area overhead according to their own system environments.

1.4 Organization

The rest of this thesis is organized as follow. Chapter 2 is the background of the characteristics of streams and some useful coding methods. In chapter 3, I will introduce my coding flow and then the proposed methods. The statistics and simulation result will be presented in chapter 4. At last, some summary, discussions, and conclusions are in chapter 5.

Chapter 2: Background

In order to reduce Bus transactions, this thesis utilize two opportunities. As mentioned in chapter 1, first opportunity is to skip the regular portion of a codeword without sending, and inform the receiver what the regularity is. Second opportunity is when

the data is regular; the sender can use a protocol to inform the receiver what the regularity is instead of sending the data. I will introduce some regularity below. The end of this chapter is some of the existing normal Bus coding methods. Some of these coding methods use protocol to reduce bit toggles and which can be used on narrow Bus to reduce Bus transactions

2.1 Redundant and Effective Fragments

I will explain some nouns I used in following chapter.

2.1.1 Redundant bits:

The general definition of redundant bits is the additional bits added to the original information data for some specific purposes. However, I use this noun here as the bits which have the regular property. There may be different regular bits types at different kinds of stream. When dealing with different streams, the default regular bits type will be different. Followings are some kinds of regular bits types.

Type 1: Repeated Bits (The Same Leading Bits with Previous Value):

This type often occurs at address stream because of the values of successive addresses are often close. When the current address is not too far from previous address, this two address values will be similar and there will be many identical leading bits. Sometimes the data stream has this property because of the relationship, too. If two successive accessed data values have relationship, these two values have chance to be close and have some identical leading bits.

Type 2: Insignificant Bits (Sign Extension):

This type often occurs at data stream because of the occurrence of small values. The small positive values will be with many leading 0's and the small negative values will be with many leading 1's.

In this thesis, I will only utilize these two kinds of redundant bits mentioned above. Depending on different systems, applications or Bus interfaces, there may be some special types of redundant bits can be utilized. Take AMBA for example, whenever the data width is smaller than Bus width, it will copy the data to other vacant Bus space. Those special types of redundant bits can be utilized case by case but this thesis won't focus on them. Figure 2-1 is some examples of upper described redundant bits types.

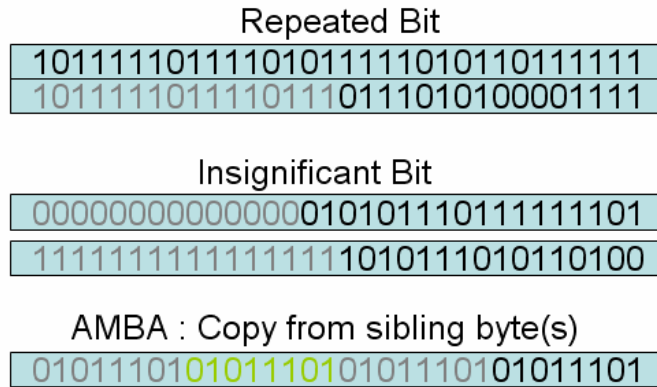


Figure 2-1: types of redundant bits

2.1.2 Effective Bits:

Effective bits are the opposition of redundant bits. In other words, a bit will be effective bit if it is not redundant bit.

2.1.3 Effective Fragments and Redundant Fragments

This thesis will focus on 8-bit width external Bus and 32-bit width data environment. The 32-bit data can be separated into several fragments. If a fragment is composed of effective bits and redundant bits, or composed only of effective bits, I call this fragment as effective fragment. If a fragment is composed of only redundant bits, it is called redundant fragment.

2.2 Related Coding Methods


Following are some of normal Bus coding methods and they have chance to be adopted on narrow Bus to reduce Bus transactions. “Asymptotic Zero-Transition Activity Encoding for Address Busses in Low-Power Microprocessor-Based Systems” (T0), “Irredundant Address Bus Encoding for Low Power” (T0-C, a modified T0), and discontinuous address table algorithm (DAT) are originally designed to reduce bit toggles on instruction address Bus. The idea of SRWEC (separating read/write encoding contents) can be used on data address stream and data stream. Variable stride algorithm is designed for data address stream. And “Selective Instruction Compression for Memory Energy Reduction in Embedded Systems” describes a code compression method which is suitable for us to adopt on narrow Bus to reduce Bus transactions.

2.2.1 Asymptotic Zero-Transition Activity Encoding for Address Busses in Low-Power Microprocessor-Based Systems (T0)

In “Asymptotic Zero-Transition Activity Encoding for Address Busses in Low-Power Microprocessor-Based Systems”, the author proposes a coding method utilizing instruction address sequential property to reduce the bit toggles. This coding method needs an extra control line. It defines the size of instruction as the stride value.

At the sender, when this instruction address is previous instruction address value adds the specific stride value, it means the instruction addresses are sequential. When the instruction addresses are sequential, it sets the extra control line as 1 and freezes the Bus to reduce bit toggles. Otherwise, it sets the extra control line as 0 and transmits the instruction address by Bus. At the receiver end, when the extra control line is 1, it doesn't care the value at Bus but adds previous instruction address by the specific stride value to get current instruction address. Only when the extra control line is 0, the receiver reads the Bus value as instruction address.

Following is the T0 encoder pseudo code.



```
b(t):real value, B(t):Bus value, S: stride value
T0 encoder pseudo algorithm:
while(1)
{
    if (b(t) == b(t-1) + S)
        B(t) = B(t-1)
        control = 1
    else
        B(t) = b(t)
        control = 0
}
```

Following is the T0 decoder pseudo code.

```
b(t):real value, B(t):Bus value, S: stride value
T0 decoder pseudo algorithm:
while(1)
{
    if (control == 1)
        b(t) = b(t-1) + S
    else
```

$b(t) = B(t)$ }

Table 2-1 is a simple example.

b(t):real value, B(t):Bus value, S: stride value = 4				
Time	Sender	Bus	Extral line	Receiver
t	b(t)	B(t)	control	b(t)
1	000000C0	000000C0	0	000000C0
2	000000C4	-	1	000000C4
3	000000C8	-	1	000000C8
4	000000CC	-	1	000000CC
5	000000C0	000000C0	0	000000C0
6	000000C4	-	1	000000C4
7	000000C8	-	1	000000C8
8	000000CC	-	1	000000CC

Table 2-1

When $b(t) = b(t-1) + S$, the control line will be set to 1 and $B(t) = B(t-1)$. When $b(t) \neq b(t-1) + S$, the control line will be set to 0 and $B(t) = b(t)$.

2.2.2 Irredundant Address Bus Encoding for Low Power (T0-C)

The T0-C is a modified method of T0. It does some changes to avoid adding the extra control line. The main idea of T0-C is that making a one-to-one and onto function. There must be existent a corresponding inverter function. Figure 2-2 is the T0-C function. When current instruction address is equal to previous instruction address adds the specific stride value, the sender sends the previous Bus value (fixes Bus value) instead. On the other hand, if the current instruction address is equal to previous Bus value, sends the previous instruction address adds the specific stride value. The other cases will be sent without any change. Because this is a one-to-one and onto function, the receiver can easily get the instruction addresses back by the inverter function.

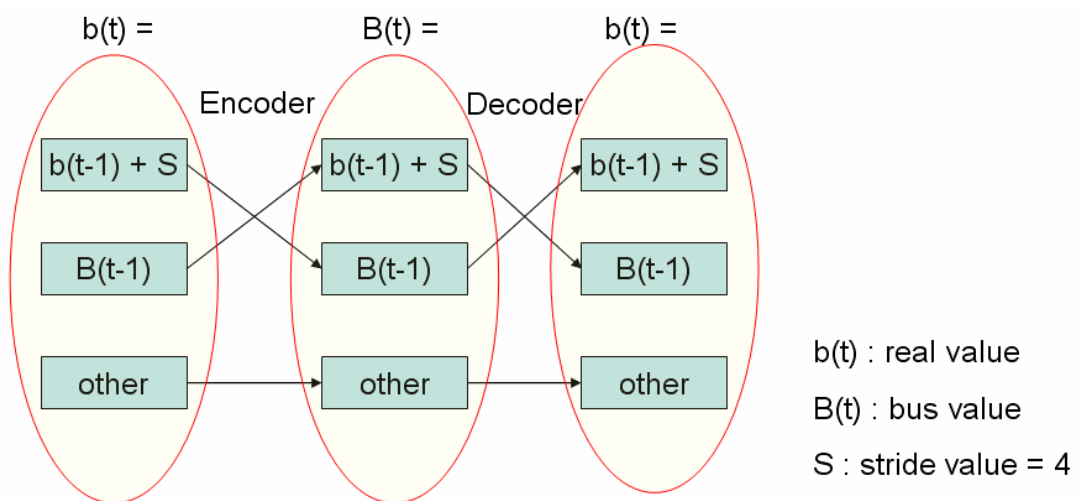



Figure 2-2: T0-C function

Following is the T0-C encoder pseudo code.

```

                                     b(t):real value, B(t):Bus value, S: stride value
T0-C encoder pseudo algorithm:
while(1)
{
  if (b(t) == b(t-1) + S)
    B(t) = B(t-1)
  else if (b(t) == B(t-1))
    B(t) = b(t-1) + S
  else
    B(t) = b(t)
}

```



Following is the T0-C decoder pseudo code.

```

                                     b(t):real value, B(t):Bus value, S: stride value
T0-C decoder pseudo algorithm:
while(1)
{
  if (B(t) == B(t-1))
    b(t) = b(t-1) + S
  else if (B(t) == b(t-1) + S)
    b(t) = B(t-1)
  else
    b(t) = B(t)
}

```

Table 2-2 is a simple example.

b(t):real value, B(t):Bus value, S: stride value = 4			
Time	Sender	Bus	Receiver
t	b(t)	B(t)	b(t)
1	000000C0	000000C0	000000C0
2	000000C4	-	000000C4
3	000000C8	-	000000C8
4	000000CC	-	000000CC
5	000000C0	000000D0	000000C0
6	000000C4	-	000000C4
7	000000C8	-	000000C8
8	000000CC	-	000000CC
9	000000F0	000000F0	000000F0

Table 2-2

2.2.3 Discontinuous Address Table Algorithm (DAT)

The instruction addresses can be classified into sequential accesses and non-sequential accesses. The sequential access property can be utilized by T0 or T0-C as mentioned above. Discontinuous address Table (DAT) is designed to utilize the non-sequential accesses part property. The non-sequential access happens when the processor executes a branch instruction. Discontinuous address table algorithm needs one extra control line and a discontinuous address table (DAT) to record the branch information.

In general case, the target addresses of taken branches are seldom changed. Discontinuous address table algorithm utilizes this property to reduce bit toggles. When a non-sequential access happens, it means that a branch instruction jumps to the specific target address. At this time, the address of the branch instruction and target instruction address will be recorded into DAT. When next time the program reaches the same source address and branch to the same target address, the sender sets the extra control line as 1 and freeze the Bus to reduce bit toggles. Otherwise, the sender sets the extra control line as 0 and sends the instruction address by Bus.

Following is the DAT encoder pseudo code.

b(t):real value, B(t):Bus value, S: stride value
DAT encoder pseudo algorithm:

```

while(1)
{
    if (b(t-1) == one of source address of DAT
    && b(t) == the corresponding target address)
        B(t) = B(t-1)
        control = 1
    else
        B(t) = b(t)
        control = 0
}

```

Following is the DAT decoder pseudo code.

b(t):real value, B(t):Bus value, S: stride value

T0 decoder pseudo algorithm:

```

while(1)
{
    if (control == 1)
        // b(t-1) is source address
        b(t) = the corresponding target address of b(t-1) in DAT
    else
        b(t) = B(t)
}

```




Table 2-3 is a simple example and Table 2-4 is corresponding DAT.

b(t):real value, B(t):Bus value, S: stride value = 4					
Time	Sender	Bus	Extral line	Receiver	DAT
T	b(t)	B(t)	control	b(t)	operation
1	00000034	00000034	0	00000034	
2	00000040	00000040	0	00000040	Insert
3	00000044	00000044	0	00000044	
4	00000030	00000030	0	00000030	Insert
5	00000034	00000034	0	00000034	
6	00000040	-	1	00000040	Found
7	00000044	00000044	0	00000044	
8	00000030	-	1	00000030	Found

Table 2-3

2 entries DAT

	Entry 1		Entry 2	
Time	Source 1	Target 1	Source 2	Target 2
T	-----	-----	-----	-----
1	-----	-----	-----	-----
2	00000034	00000040	-----	-----
3	00000034	00000040	-----	-----
4	00000044	00000030	00000034	00000040
5	00000044	00000030	00000034	00000040
6	00000044	00000030	00000034	00000040
7	00000044	00000030	00000034	00000040
8	00000044	00000030	00000034	00000040
→ DAT Queue →				
Table 2-4				

2.2.4 Idea of Separating Read and Write Streams

The data address stream includes the read data addresses and write data addresses. When executing program, these two streams will be mixed and the characteristics of each stream will be disturbed. If we deal with these two parts individually, the characteristics of them will be preserved without being disturbed. For the same reason, the data stream can utilize this idea, too.

2.2.5 Variable Stride Algorithm

There are two kinds of variable stride algorithm, one is similar to T0 and the other is similar to T0-C. However, its stride value will be changed with time. Whenever the current address is encoded at sender end, it updates the variable stride as current data address minus previous data address. At the receiver, it will update the variable stride as current data address minus previous data address right after finishing decoding. I will only introduce variable stride algorithm similar to T0-C for example below.

Following is the Variable Stride algorithm (similar to T0-C) encoder pseudo code.

b(t):real value, B(t):Bus value, S: stride value

Variable Stride encoder pseudo algorithm:

```

while(1)
{
    if (b(t) == b(t-1) + S)

```

```

    B(t) = B(t-1)
else if(b(t) == B(t-1))
    B(t) = b(t-1) + S
else
    B(t) = b(t)
S = b(t) - b(t-1)
}

```

Following is the Variable Stride algorithm (similar to T0-C) decoder pseudo code.

b(t):real value, B(t):Bus value, S: stride value

Variable Stride decoder pseudo algorithm:

```

while(1)
{
    if (B(t) = B(t-1))
        b(t) == b(t-1) + S
    else if(B(t) = b(t-1) + S)
        b(t) == B(t-1)
    else
        b(t) = B(t)
    S = b(t) - b(t-1)
}

```



Table 2-5 is a simple example.

b(t):real value, B(t):Bus value, S: variable stride value				
Time	Sender	Bus	Variable stride	Receiver
T	b(t)	B(t)	S	b(t)
1	000000C0	000000C0	-	000000C0
2	000000C4	000000C4	4	000000C4
3	000000C8	-	4	000000C8
4	000000CC	-	4	000000CC
5	000000F0	000000F0	24	000000F0
6	000000F2	000000F2	2	000000F2
7	000000F4	-	2	000000F4
8	000000F6	-	2	000000F6
9	000000F8	-	2	000000F8
10	000000FA	-	2	000000FA
11	000000F2	000000FC	-8	000000F2
12	000000F4	000000F4	2	000000F4

13	000000F6	-	2	000000F6
Table 2-5				

2.2.6 Selective Instruction Compression for Memory Energy Reduction in Embedded Systems

In “Selective Instruction Compression for Memory Energy Reduction in Embedded Systems”, the author proposes a dictionary based instruction compression. The main idea of this paper is that there are a few instructions are executed frequently. The method proposed is to gather the most frequently used 255 instructions in an Instruction Decompression Table (IDT) at static time. When the processor requesting instruction is in the IDT, the memory sends the index of the instruction in IDT to the processor. If the instruction doesn't exist in the IDT, the memory sends the preserved index “mark” (00000000) and then sends the instruction byte by byte. At the receiver, if the received byte is not the preserved index “mark” (00000000), it looks up the IDT and extracts the instruction. If the receiver receives the “mark,” it continues receives the following four bytes and then composes them to a complete instruction. Because the instruction is 32 bits and the index is 8 bits, this algorithm can reach compression effect when the instruction is in IDT.

Figure 2-3 is the proposed architecture of IDT.

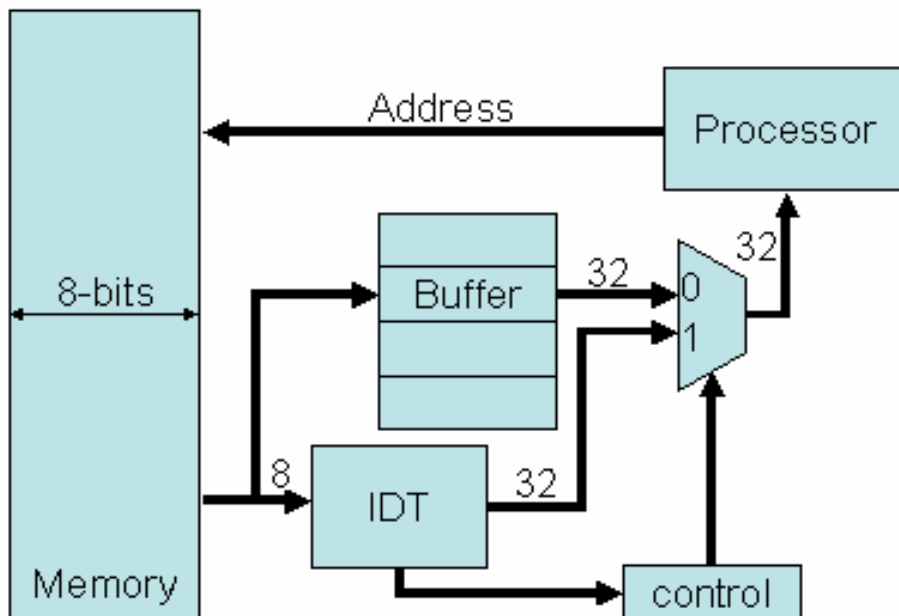


Figure 2-3: the proposed architecture of IDT

If the first byte is not “mark”, the control signal will be 1. If the first byte is “mark”,

the control signal will be 0 and the buffer will collect 4 bytes and then compose them to original instruction.

Chapter 3: Design of Proposed Coding

In the beginning of this chapter, I will introduce my coding flow. Next, I will discuss how to reduce Bus transactions. At the last of this chapter, I will describe the complete coding methods of instruction address stream, data address stream, instruction stream, and data stream.

3.1 Coding Flow

Figure 3-1 is a coding flow diagram. The purpose of the encoder is to convert the effective bits of a codeword into redundant bits. The redundant type will be different in different streams. Between encoder and narrow Bus is a variable length encoder (VL-Encoder). The VL-Encoder will separate the 32-bit codeword into several fragments, skip the redundant fragments without sending, and put the effective fragments on the Bus. The variable length decoder (VL-Decoder) receives the effective fragments of a codeword, composes them, and then fills the redundant fragment parts to become original codeword. The number of effective fragments of a codeword is variable and then the number of Bus transactions of a codeword will be variable. The VL-Decoder has to get the additional information to know the number of the Bus transactions of a codeword. Otherwise, the receiver won't know when to stop receiving data and composing the effective fragments. The VL-Encoder and VL-Decoder, encoder and decoder, will all be introduced below.

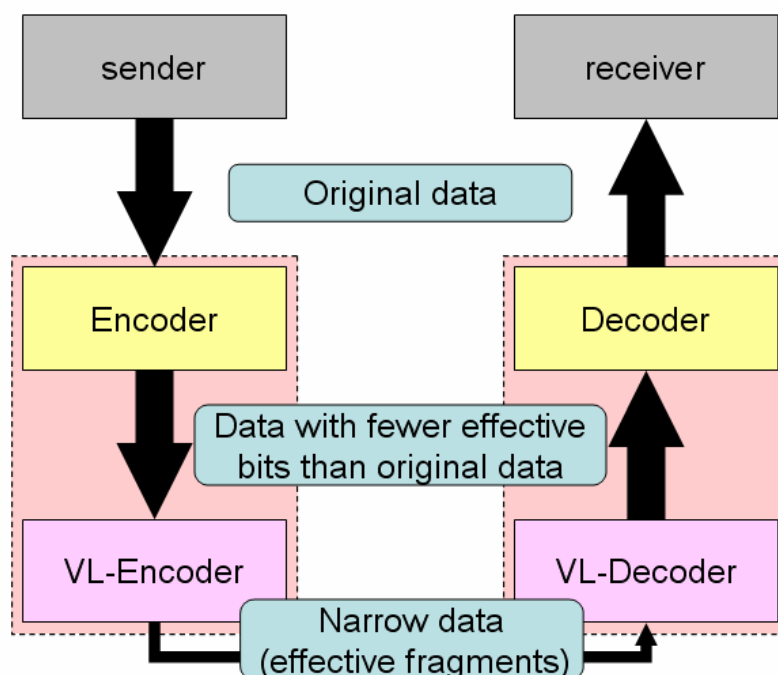


Figure 3-1: the coding flow diagram

3.2 VL-Encoder

The purpose of VL-Encoder is to reduce Bus transactions. There are two problems rising here. The first problem is what to be transmitted and what to be skipped. The second problem is how the receiver knows the number of Bus transactions of a codeword.

The first problem is what can be skipped without sending and what should be transmitted on Bus. The solution is to transmit the effective fragments and skip the redundant fragments without sending. The redundant bits type will be different on different streams. There will be default redundant type at each stream. Because the receiver knows the redundant bits type, it can retrieve the redundant fragments which are not transmitted.

The second problem is how the receiver knows the number of Bus transactions of a codeword. If the receiver end wants to know this additional information, the sender has to add this information into the codeword. There will be two main directions to achieve this purpose. The first method is to add the information of the number of Bus transactions at the beginning of first transmission. The second method is to inform the receiver whether the total fragments of a codeword are transmitted or not at every Bus transaction. Following I will introduce these two methods and do a simple comparison to select the suitable method as my VL-Encoder policy at this environment.

3.2.1 Method One : Adding the Information of the Number of Bus Transactions at the Beginning of First Transmission

This method will add the information of the number of Bus transactions at the beginning of first transmission. The extra information indicates the number of Bus transactions. This extra information must be transmitted at first transaction of a codeword; otherwise, the receiver will need much more complexity mechanism to get this information and this may require too much coding time. The receiver knows how many fragments should be received whenever the first fragment of a codeword is received. I formulize the relation between Bus width (w) and extra information bits (k) at Inequality 3-1.

$$(32+k) / w < 2^k \quad \text{Inequality 3-1}$$

Take 32-bit width data and 8-bit width Bus for example. The effective bits of data can be 0 to 32 bits. After adding k bits extra information, the effective bits will be k to $(32 + k)$ bits. The worse case of Bus transaction will be more than 4 times. In order to indicate 1 to more than 4 Bus transaction, the 'k' has to be more than 3. To sum up, if I want to transmit 32-bit codeword by 8-bit width Bus, I need 3 extra information bits. This implies that the effective bits will increase by additional 3 bits and the Bus transaction will be 1 to 5 times.

3.2.2 Method Two : Indicating the Codeword is Transmitted over or not at Every Bus Transaction

This method informs receiver end whether the total fragments of a codeword are transmitted over or not at each Bus transaction. This method is to add an extra control line called "End bit" (EB). If this Bus transaction transmits the last fragment of a codeword, the sender sets the EB value as 1. Otherwise, the sender sets EB as 0. At receiver end, the receiver keeps collecting codeword fragments until the EB is set to 1.

3.2.3 Comparison of Method One and Method Two

The method one adds the information of the number of Bus transactions at the beginning of first transmission and the method two adds an extra control line to indicate the last fragment of a codeword. In order to compare the effect of reducing Bus transactions of method one and method two in the same standard, I use 9-bit as Bus width.

In method one, I substitute the parameter 'w' by 9 in inequality 3-1. The smallest k I can get is 2. The number of Bus transactions is from 1 to 4. When the Bus transactions are n times, the sender sends $9*n$ bits. The representable effective bits except the extra two information bits are $(9*n - 2)$ bits.

In method two, the extra EB is permanent. The number of Bus transactions is from 1 to 4. When the Bus transactions are n times, the sender can send $(8 * n)$ effective bits of codeword.

The comparison of representable effective bits of these two methods is shown in Table 3-1

	The effective bit represented by method 1	The effective bit represented by method 2
The number of Bus transactions is 1	$9 - 2 = 7$	8
The number of Bus transactions is 2	$2 * 9 - 2 = 16$	$2 * 8 = 16$
The number of Bus transactions is 3	$3 * 9 - 2 = 25$	$3 * 8 = 24$
The number of Bus transactions is 4	$4 * 9 - 2 = 34 (32)$	$4 * 8 = 32$
Table 3-1		

The representable effective bits of a method will affect the number of Bus transactions of a codeword. For example, if the number of effective bits of a codeword is 8 bits, method one needs 2 Bus transactions because 1 Bus transaction can only represent 7 effective bits. The main difference of these two methods appears when the number of effective bits of a codeword is 8 bits or 25 bits. When the number of effective bits of a codeword is 8 bits, the method one needs 2 Bus transactions and method two needs 1 Bus transaction. On the other hand, when the effective bits of a codeword is 25 bits, the method one needs 3 Bus transactions and method two needs 4 Bus transactions.

I use instruction address stream and data address stream to compare the effect of these two methods. I don't use instruction stream to evaluate because I will compress instruction stream using its statistic property not its numerical property. I don't use data stream to evaluate because there may be multiple redundant bits types are utilized and it need adding other information bits into codeword. Table 3-2 is the Bus transactions ratio based on narrow Bus without coding (4 Bus transactions).

	IA	DA
Method one	27.49%	50.08%
Method two	26.81%	49.39%
Table 3-2		

The Bus transactions ratio of method one is smaller than method two when method two uses 9-bit width Bus. Therefore, we can compare the Bus transactions ratio of these methods. We can observe that the method one (9-bit Bus) is better than the

method two with 9-bit width Bus. Besides, my objective is to reduce Bus transactions but the method one with 8-bit width Bus will increase the Bus transactions in worst case. Therefore, I think the byte based method is more suitable in my environment.

3.2.4 The VL-Encoder Will Use Method Two

After the discussion and evaluation, I will use method two, EB VL-Encoder, as my VL-Encoder policy. I will add an extra control bit “EB”. When sending the last fragment of a codeword, the sender will set the EB as 1. Otherwise, the sender should set EB as 0. The bytes of a codeword will be sent from MSB to LSB. The receiver will keep collecting bytes of a codeword until the EB value is 1, compose the effective bytes, and then fill the redundant bytes to become original codeword. The Figure 3-2 shows the behavior of EB VL-Encoder.

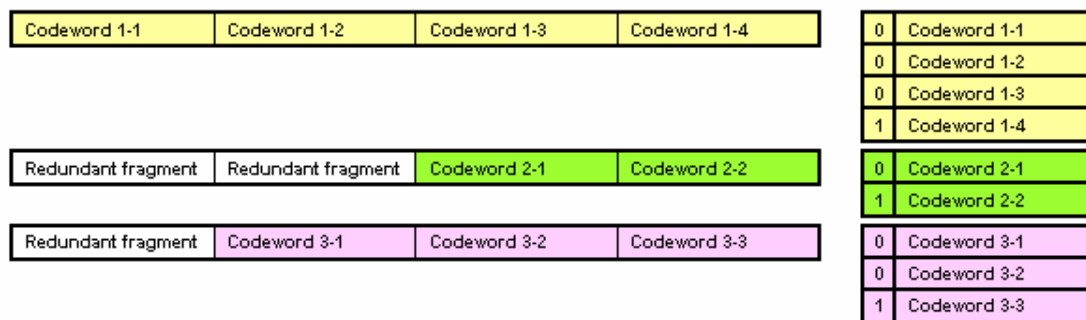


Figure 3-2: behavior of EB VL-Encoder

3.3 Instruction Address Stream

3.3.1 Type of Redundant Bits Used by EB VL-Encoder :

Due to the numerical property of instruction address stream, the instruction address stream uses repeated bits (the same leading bits) as default redundant bits type. The VL-Encoder will skip the redundant bytes of a codeword without sending.

3.3.2 Encoder :

The job of encoder is to convert the effective bits of a codeword into redundant bits and then the VL-Encoder can transmit fewer bytes. This thesis will propose several coding methods in each stream. Following I will introduce some coding methods which I will adopt to narrow instruction address Bus.

Encoder 1: T0 (1 extra control line)

T0 utilizes the sequential property of instruction address. It has been introduced in chapter 2.

Encoder 2: T0-C (no extra control line)

The same as T0, it utilizes the sequential property of instruction address and has been introduced in chapter 2.

Encoder 3: NBDAT (1 extra control line)

Discontinuous Address Table (DAT) algorithm focuses on the discontinuous address behavior of instruction addresses. It has been introduced in chapter 2. However, I will propose a modified DAT which I call it “Narrow Bus DAT” (NBDAT). This modified DAT has a filter mechanism. If a record can reduce Bus transaction, I will record it into NBDAT. Otherwise, I won’t record it into NBDAT even if the addresses are discontinuous. This will raise the utility of DAT.

3.3.3 Proposed Methods :

I will propose several combinational coding methods which have different area overhead and different Bus transactions reduction.



Proposed Method 1: (1 extra control line)

The encoder 1 (T0) can be adopted on narrow Bus independently without the assistance of the VL-Encoder. The general T0 on normal Bus will send a signal instead of sending the instruction addresses when the instruction addresses are sequential. The T0 on narrow Bus uses the same idea. When the instruction addresses are sequential, the sender sets the extra control line as 1 and freeze the Bus value. The number of Bus transaction will be 1. Otherwise, the sender sets the extra control line as 0 and transmits the instruction address byte by byte. The number of Bus transaction will be 4. I will use the number of Bus transaction of this method as base line to compare with other methods.

Proposed Method 2: (1 extra control line)

The second method is the combinational method of encoder 2 and EB VL-Encoder. The T0 can be used on narrow Bus to reduce Bus transactions independently but T0-C can't. This is because the 32-bit instruction address to the 8-bit Bus is a multiple to one function. There is no inverter function if there is no other extra coding information. Therefore, method 2 combines T0-C encoder and EB

VL-Encoder together to reduce Bus transactions. The T0-C is a 32-bit to 32-bit function which can convert the effective bits of a codeword into the redundant bits. EB VL-Encoder decomposes the output of T0-C and puts the effective fragments on narrow Bus to transmit.

Proposed Method 3: (2 extra control lines)

The third method is the combinational method of encoder 2, encoder 3 and EB VL-Encoder. The NBDAT and T0-C can be adopted on narrow Bus as encoder at the same time and get more Bus transaction reduction.

3.4 Data Address Stream

3.4.1 Type of Redundant Bits Used by EB VL-Encoder :

Because of the numerical property of data address stream is similar with instruction address stream; the data address stream uses repeated bits (the same leading bits) as default redundant bits type, too. The VL-Encoder will skip these redundant fragments of a codeword without sending.

3.4.2 Encoder :

The job of encoder is to convert the effective bits of a codeword into redundant bits and then the VL-Encoder can transmit fewer bytes. Because of the idea of separating read data address and write data address mentioned in chapter 2, the following discussion will all base on this principle without emphasizing it repeatedly. As instruction address stream, this thesis will propose several coding methods. Following I will introduce some coding methods which I will adopt to narrow Bus.

Encoder 1: Variable stride (0 or 1 extra control line)

Both type of variable stride algorithm (similar to T0 and similar to T0-C) can utilize the sequential property of data address. It has been introduced in chapter 2.

Encoder 2: 1st version historical addresses algorithm (Y extra control lines)

In order to utilize the spatial locality and temporal locality of data address stream, I propose historical addresses algorithm here. The idea of historical addresses algorithm is that the surroundings of accessed address have high chance to be used latter.

First, I define the leading first k bytes of an address as its k-byte-address. On narrow Bus environment with the VL-Encoder which I proposed, it will need more than 1 Bus transaction whenever the 3-byte-address of Bus value differ from the 3-byte-address of next data address. The idea of historical addresses algorithm is to save some k-byte-address into historical address record. Whenever a k-byte-address of transmitted value differ from the k-byte-address of Bus value but recorded in historical address record, it can utilize this record to reduce Bus transactions.

Historical addresses algorithm use an X-entries FIFO historical address record table and extra Y-bit historical address index control lines to indicate the entry of the historical address record. The relationship between X and Y is " $X = 2^Y - 1$ ". This is because the Y-bit control lines can represent 2^Y states. One state has to be preserved for "the k-byte-address is not found in historical address record." Therefore, these Y-bit control lines can indicate $2^Y - 1$ historical address indexes. The value of Y is a parameter restricted by the system area constraint. Historical addresses algorithm deals with 3 cases. I will introduce the coding with 3-byte-address case by case below.

The first case is that the 3-byte-address of transmitted codeword is the same with the 3-byte-address of Bus value. In this case, the number of effective bytes of the transmitted codeword will be 1 and this codeword needs only 1 Bus transaction. The information of extra control line is set to mean "not use record".

The second case is that the 3-byte-address of transmitted codeword differ from the 3-byte-address of Bus value and this 3-byte-address of transmitted codeword is not stored in historical address record. In this case, the number of effective bytes of the transmitted codeword will be more than 1 and this codeword needs more than 1 Bus transactions. The sender will set the information of extra control line as "not use record" and store the 3-byte-address of previous Bus value into the historical address record as update.

The third case is that the 3-byte-address of transmitted codeword differ from the 3-byte-address of Bus value but this 3-byte-address of transmitted codeword has been stored in historical address record. In this case, the sender will change the leading 3 bytes of the transmitted codeword to be the same with the leading 3 bytes of the Bus value, and set the information of extra control line as "index of the 3-byte-address in historical address record". In this way, the number of effective

bytes of transmitted codeword will be 1 and the number of Bus transaction of this codeword has been reduced to only 1.

Table 3-3 is a 1-entry 3-byte-address record example. When time is 2 the case one happens. When time is 3 the case two happens. When time is 4 the case three happens. The case three affects the number of Bus transactions. When time is 4, the number of effective bytes of the codeword reduces from 4 to 1 after encoding. This reduces the Bus transactions from 4 to 1 at the same time.

b(t):real value, B(t):Bus value,				
Time	Sender	Bus	control state	1 entry Table
T	b(t)	B(t)	control state	Historical address record
1	FFFFFFF00	<u>FFFFFFF00</u>	Not use	-----
2	FFFFFFF01	<u>FFFFFFF01</u>	Not use	-----
3	EEEEEEE00	<u>EEEEEEE00</u>	Not use	FFFFFFFF
4	FFFFFFF02	----- <u>02</u>	Index	FFFFFFFF
Table 3-3				

At algorithm 3 and algorithm 4, I will propose two modification versions of historical addresses algorithm. First modification, 2nd version, is to combine the information of EB of VL-Encoder and historical addresses indexes information together. Basing on algorithm 2, the second modification, 3rd version, is that the sender can utilize all size of historical addresses (1-byte-address, 2-byte-address and 3-byte-address) without adding other extra control lines.

Encoder 3: 2nd version historical addresses algorithm (Y extra control lines)

After simulation, I find the recording 3-byte-address will be better than recording 2-byte-address or 1-byte-address. In this 2nd version I will use 3-byte-address to describe the modification. In my observation, when the 3-byte-address is not in the historical address record, the codeword needs more than 1 Bus transactions. When the fragments are transmitted, no matter the EB is 0 or 1, the value of historical address index control lines is meaningless. The coding flow I describe at beginning of this chapter is separating the encoder and VL-Encoder into two parts. However, if I can combine the information of VL-Encoder EB and the information of encoder historical address index control lines, I can represent more number of historical address indexes by the same number of extra control lines.

Taking a system which can tolerate total Y-bit extra control lines and use 3-byte-address for example, if we don't combine the VL-Encoder EB and encoder historical address index control lines, we have to leave 1 bit for VL-Encoder EB and other (Y - 1) bits can be used to represent $2^{(Y-1)}$ states and can indicate $2^{(Y-1)} - 1$ historical address index. This is because 1 of the $2^{(Y-1)}$ state is taken to mean "the 3-byte-address is not found in historical address record". On the other hand, if we combine the VL-Encoder EB and encoder historical address index control lines, we can represent 2^Y states and can indicate $2^Y - 2$ historical address index. 1 of the 2^Y state is used to mean "this fragment is not the last fragment of this codeword", this will cause more than 1 Bus transactions and hint that the 3-byte-address of this codeword is not in historical address record with the result that the Bus transactions will be more than 1. Another 1 of the 2^Y state is used to mean "this fragment is the last fragment of this codeword but the 3-byte-address of this codeword is not in historical address record". The number of representable states of combining VL-Encoder EB with encoder historical address index control lines will be more than not combining policy as long as Y is bigger than 1.

Encoder 4: 3rd version historical addresses algorithm (Y extra control lines)

After simulation, I find the recording 3-byte-address will be better than recording 2-byte-address or 1-byte-address. However, there is an embedded hint can be used to adopt 1-byte-address, 2-byte-address, and 3-byte-address by the same historical address index.

In 2nd version of historical addresses algorithm, one state can indicate one historical address index. If a system can tolerate two (Y = 2) extra control lines, it can indicate two ($2^2 - 2 = 2$) historical address indexes. This 3rd version of historical addresses algorithm is modified and can represent two 1-byte-address records, two 2-byte-address records, and two 3-byte-address records with these two states. This is because when I find k-byte-address is utilizable, I will change the first k bytes of codeword into redundant bytes and set the historical address index. I have to transmit the other (4 - k) effective byte. When the receiver receives these (4 - k) effective byte and the historical address index, it will know which size of historical address record should be looked up by the number of Bus transactions. Table 3-4 is an example and Table 3-5 is the corresponding change of 2-entry historical address record. When the time is 3, the sender utilize the first (index = 1) historical address of 2-byte-address record. The sender sends the two effective bytes and sets the control state as "index = 1". The receiver receives the second fragment of this codeword and finds the control state is set to "index = 1", it knows that it should

look up 2-byte-address and reload the first historical address of 2-byte-address record back to codeword.

B(t):real value, B(t):Bus value				
Time	Sender	Bus	Narrow Bus	control state
	b(t)	B(t)		index
1-1	FFFFFF00	FFFFFF00	FF	Not End
1-2			FF	Not End
1-3			FF	Not End
1-4			00	End
2-1	EEEEEE00	EEEEEE00	EE	Not End
2-2			EE	Not End
2-3			EE	Not End
2-4			00	End
3-1	FFFF0000	---- 0000	00	Not End
3-2			00	Index = 1

Table 3-4

2 entry historical address record with all size						
	1-byte-address record		2-byte-address record		3-byte-address record	
	1-byte-addr ess	1-byte-addr ess	2-byte-addr ess	2-byte-addr ess	3-byte-addr ess	3-byte-addr ess
index	1	2	1	2	1	2
Time	--	--	----	----	-----	-----
1	--	--	----	----	-----	-----
2	FF	--	FFFF	----	FFFFFF	-----
3	FF	--	FFFF	----	EEEEEE	FFFFFF

Table 3-5

3.4.3 Proposed Methods :

I will propose several combinations of these coding methods and adopt these combination methods to narrow Bus. These combinations have different extra control lines and can reach different Bus transactions reduction.

Proposed Method 1: (1 extra control line)

The encoder 1 (similar to T0 type variable stride algorithm) can be adopted on narrow Bus independently without the assistance of the VL-Encoder. The behavior

is totally the same with proposed method 1 of instruction address stream.

Proposed Method 2: (1 extra control line)

The second method is to combine encoder 1 (similar to T0-C type variable stride algorithm) and EB VL-Encoder. The behavior of this method is totally the same with proposed method 2 of instruction address stream.

Proposed Method 3: (Y extra control line)

The third method is the combinational method of encoder 1 (similar to T0-C type variable stride algorithm), encoder 4 (3rd version of historical addresses algorithm) and EB VL-Encoder. The encoder 1 utilizes the sequential access property and encoder 4 utilizes the spatial and temporal locality. These two encoder can be adopted on narrow Bus at the same time and get more Bus transaction reduction.

3.5 Instruction Stream

In this stream, I adopt code compression method to reduce code size. There are two main consideration factors when we select code compression method. First consideration factor is coding time. The Bus coding (encoding and decoding) is a time critical work. The coding time delay must be very short. The second consideration factor is compression ratio. In our environment, the unit of compression size is byte due to the 8-bit Bus width. It is the same to compress a date into 8 bits and into 1 bit because they both need 1 Bus transaction.

Due to the instinct property, the narrow Bus environment has higher probability to be an embedded system environment. Therefore, we can choose a code compression method even if it has high compression ratio only in embedded system.

The “Selective Instruction Compression for Memory Energy Reduction in Embedded Systems” encodes instruction at static time. It has no encoding delay to do encoding. The decoding delay is equal to access a 256 entries direct map cache. In average of my test programs (Mibench), 98.35% instructions can be compressed into 8 bits and need only 1 Bus transactions. I think it is one of the suitable code compression methods.

After compress instruction, I discuss the methods of transmitting this compressed instruction on narrow Bus below.

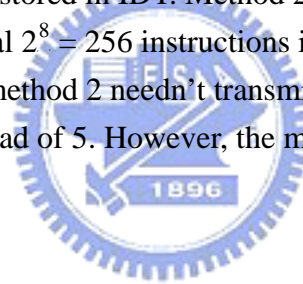
Transmission method 1: (no extra control line)

This is just transplant the code compression mechanism from memory to Bus. As mentioned at chapter 2, if the transmitted instruction is in IDT, the sender transmits the index of that instruction. If the transmitted instruction is not in IDT, it transmits the preserved index MARK “00000000” and then sends the instruction byte by byte.

Transmission method 2: (1 extra control line)

This transmission method use the extra control line EB to replace the function of the preserved index MARK “00000000”. If the instruction is in IDT, the sender transmits the index of that instruction and set the EB as 1. Otherwise, the sender sends the instruction byte by byte, sets the EB as 0 until the last Bus transaction the sender sets the EB as 1.

The differences between these two methods are the control line, transaction times, and the number of instruction stored in IDT. Method 2 needn't the preserved index “00000000” so it can save total $2^8 = 256$ instructions into IDT. When the instruction is not in IDT, the method 2 needn't transmit the preserved index and the Bus transaction will be 4 instead of 5. However, the method 2 needs an extra control line EB as assistance.



3.6 Data Stream

3.6.1 Type of Redundant Bits Used by EB VL-Encoder :

Different from instruction address stream and data address stream, the two types of redundant bits (insignificant bits and repeated bits) both can be utilized in data stream. If there is k-bit extra control lines to indicate the type of redundant bits, I have total k+1 extra control lines (including EB). Besides the not transmitted over state, I can utilize $2^{k+1}-1$ kinds of redundant bits type. In instruction address stream and data address stream, the value of k is 0, so there is only $2^{0+1}-1 = 1$ kind of default redundant bits type can be used.

When I regard the leading 0's or 1's as the redundant bits type, there are two ways to utilize it. The original behavior of sign extension is that the system decides to extend 0's or 1's depending on the value is positive or negative. If I want to skip the leading 0's and leading 1's, I have to decide where the positive or negative hint

should be recorded. It may be preserved outside the codeword or embedded in the codeword as the first bit of effective bits. For example, a data value is “11111111,11111111,11100000,10100011”. It may be seen as “-----,-----,---00000,10100011” with extra information to indicate this is a negative value, or it may be seen as “-----,-----,--100000,10100011” and using the first effective bit to hint that this is a negative value. In order to fill the correct bits at VL-Decoder, there must be enough information to indicate the redundant bits type. If I want to use the former policy, I need two information states to indicate that “the redundant bits should be filled by doing 0’s extension” and “the redundant bits should be filled by doing 1’s extension”. If I want to use the latter policy, I need only one information state to indicate that “the redundant bits should be filled by doing sign extension” because the positive or negative hint has been embedded in first bit of effective bits.

How to choose the suitable redundant bits types is the most important thing in data stream. For using insignificant bits as redundant bits, I have two possible choices as mention above. One needs two information states and the other will raise the number of effective bits by 1 but needs only one information state. For using repeated bits as redundant bits, it needs only one information states to indicate that “the redundant bits should be filled by copying the corresponding bits of previous Bus value”. These two kinds of redundant bits both may be used or not. Therefore, the possible redundant bits combinations are classified at Table 3-6. For insignificant bits, it may be not seen as redundant bits, use two information states, or use one information state to utilize. For repeated bits, it may be not seen as redundant bits or use one information state to utilize.

	Repeated	Insignificant Embedded hint	Insignificant Additional hint	Number of need states
Choice 1	0	0	0	0
Choice 2	1	0	0	1
Choice 3	0	1	0	1
Choice 4	1	1	0	2
Choice 5	0	0	2	2
Choice 6	1	0	2	3

Table 3-6

Even if no extra control line to indicate the type of redundant type, I also can decide a default redundant type. Therefore the choice 1 won’t be adopted. If there is no extra control line to indicate the type of redundant bits, I have one default

redundant bits type can use and I can use choice 2 or choice 3. If there is one extra control line to indicate the type of redundant bits, I can indicate three redundant bits types and I can use choice 4, choice 5, or choice 6. There may be multiple choices on the same extra control line constraint. I will make decision by simulation result.

Chapter 4: Statistics and Simulation Result

In this chapter, I will describe the environment of my statistics and simulation. And then are the statistics and coding simulation results of instruction address stream, data address stream, instruction stream, and data stream.

4.1 Statistics and Simulation Environment

The simulator I use is simple-scalar for ARM instruction. The simulated programs are Mibench. My environment is 8-bit width narrow Bus and 32-bit width data.

The area overheads include extra control lines, extra coding logic gates, and additional table. I assume the area overhead of external extra control lines is much severer than extra logic gates. Because of the table size must be limited; I will use different size of table to simulate and observe the effects of different size, and then suggest a “good enough” size. The main advantage of coding is to reduce Bus transactions. This will reduce Bus bit toggles at the same time. However, I won’t consider the power advantage of reducing bit toggles. I will put my attention on the Bus transactions only.

I will use Bus transactions ratio to compare the effect of coding methods. I normalize the number of Bus transactions of narrow Bus without coding as 1. In my simulation environment, the Bus transaction of narrow Bus without coding is 4; therefore the number of normal Bus transactions will be one fourth comparing to narrow Bus. The Bus transactions ratio of normal Bus will be 25%.

4.2 Statistics and Simulation Result of Instruction Address Streams

4.2.1 Statistics

In average, the ratio of sequential instruction addresses is 91% and the ratio of non sequential instruction addresses is 9%. These sequential instruction addresses can be utilized by T0 (or T0-C) and the Bus transaction will be 1. If we discuss them in more detail, there is 0.17% of total instruction address is first executed and 8.83%

of instruction address is repeatedly executed. That is, the 8.83% of instruction address can be utilized by NBDAT.

In total instruction addresses, 93.93% of instruction address with 1 effective byte, 4.82% of instruction addresses with 2 effective bytes, 1.25% of instruction address with 3 effective bytes, and almost no instruction address with 4 effective bytes. The number of effective bytes is equal to the number of Bus transactions with EB VL-Decoder. Therefore, even if no other coding, EB VL-Encoder can reach very low Bus transactions ratio in instruction address stream.

4.2.2 Simulation Result

I propose three narrow Bus coding methods in this thesis. First of them is transplanting the normal Bus T0 coding to narrow Bus. The second is that T0-C algorithm and EB VL-Encoder work together. The third method is that T0-C algorithm, NBDAT algorithm and EB VL-Encoder work together.

First, I compare the effect of DAT and NBDAT in Figure 4-1. Figure 4-1 is the Bus transactions ratio of DAT and NBDAT comparing to narrow Bus without coding. The Y-axis is the Bus transactions ratio of the coding methods comparing to narrow Bus without coding. The X-axis is the table entries of DAT and NBDAT. Because NBDAT has a filter to filter the helpless entry not to save in table, NBDAT can reach lower Bus transactions ratio than DAT. The Bus transactions ratio of 512-entry NBDAT can reach the same effect with unlimited NBDAT. I choose 16-entries NBDAT as “good enough” choice and the effect of 512-entries NBDAT as the lower bound of Bus transactions ratio of NBDAT algorithm.

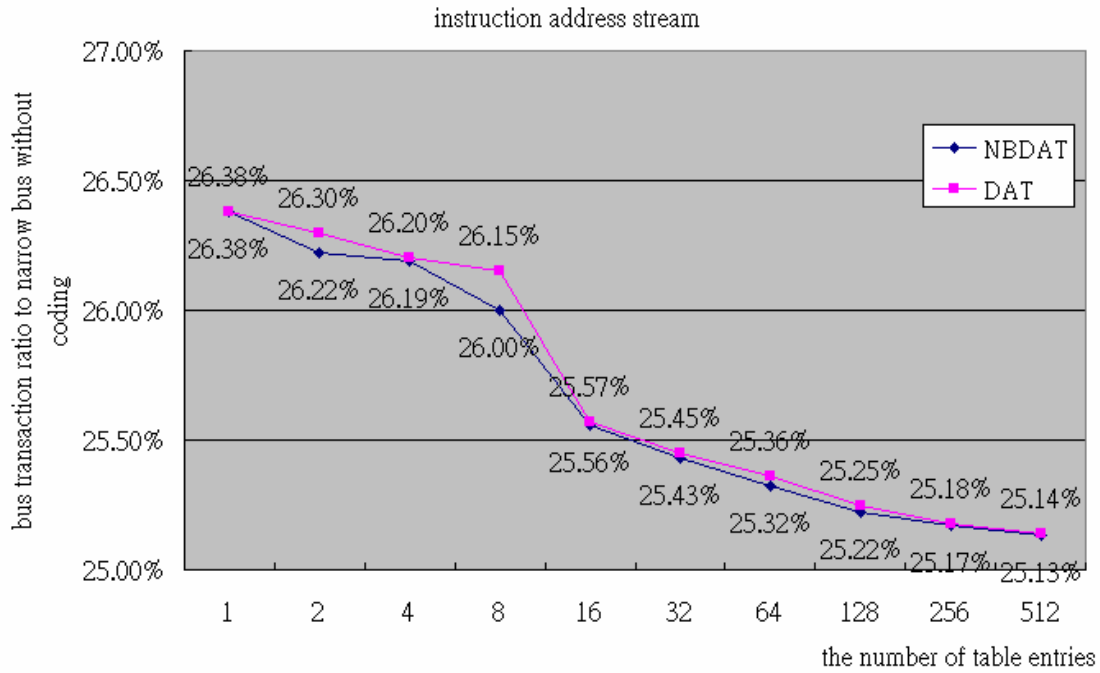


Figure 4-1: The Bus transactions ratio of DAT and NBDAT in different number of table entries

Figure 4-2 is the comparison of the proposed coding methods. The Y-axis is the Bus transactions ratio of the coding methods comparing to narrow Bus without coding. The X-axis lists the proposed coding methods. The last value is the Bus transactions ratio of normal Bus. The first method is to transplant the normal Bus T0 to narrow Bus. The second is that T0-C algorithm and EB VL-Encoder work together. The third and fourth method are that T0-C algorithm, NBDAT algorithm and EB VL-Encoder work together. The third method is with 16-entries NBDAT and the fourth method is with 512-entries (unlimited) NBDAT.

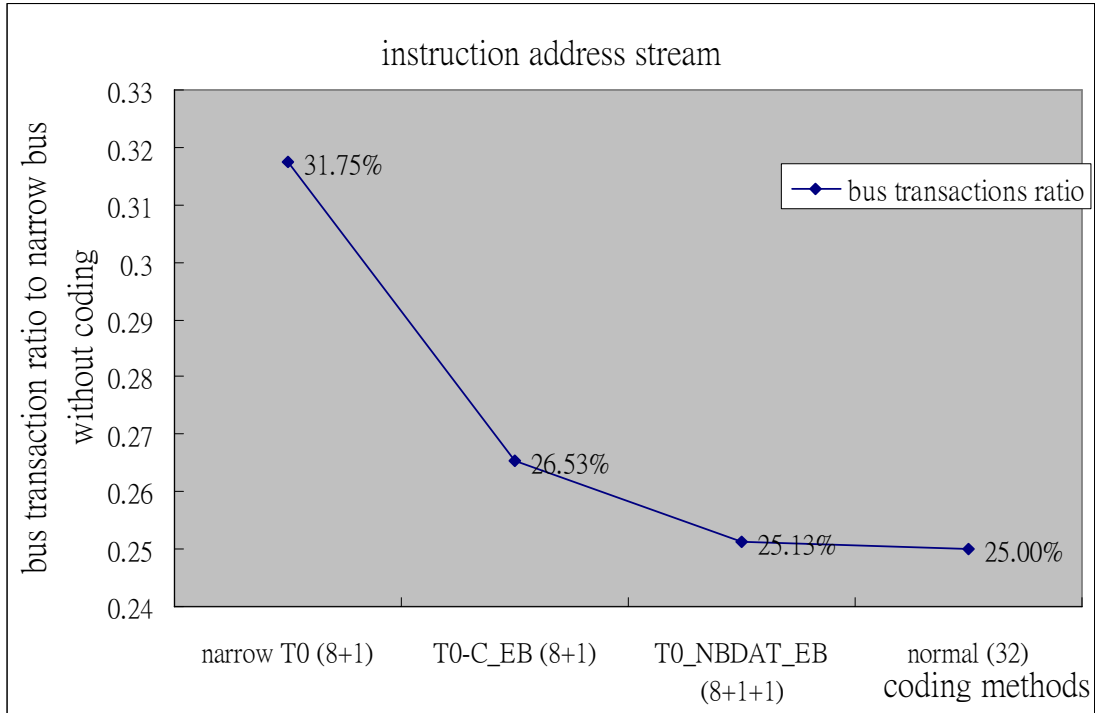


Figure 4-2: The Bus transactions ratio of proposed methods in instruction address stream.

The Table 4-1 shows the coding time delay and the area overhead (ignoring the coding logic). Because the real delay time will depend on the implementation, I only list the gate delay here.

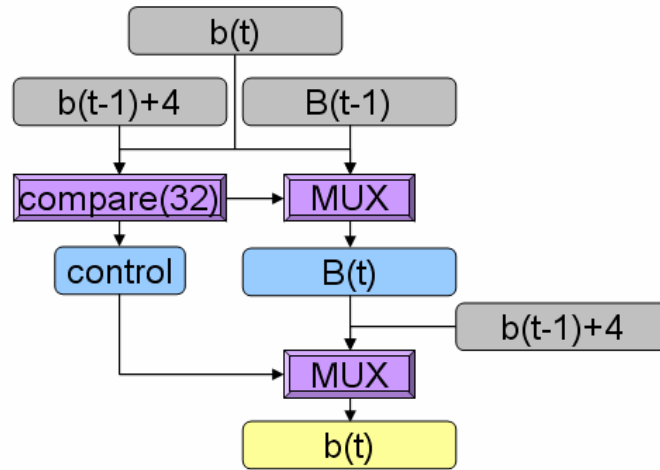
	T0	T0-C	DAT(NBDAT)	EB
Gate delay (encoder)	Comparator(32) 2-to-1 MUX	Comparator(32) 3-to-1 MUX	Comparator(32) 2-to-1 MUX	Comparator(8) 4-to-1 MUX
Gate delay (decoder)	2-to-1 MUX	Comparator(32) 3-to-1 MUX	2-to-1 MUX	
Overhead	1 control line		1 control line 2*k entry table	1 control line

Table 4-1

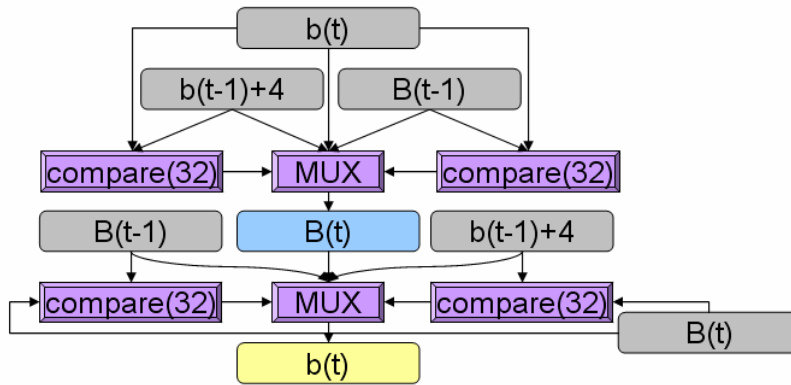
The followings are the coding logic diagram. In pipeline system, the input can be processed in previous pipeline stage. It won't cause coding delay.



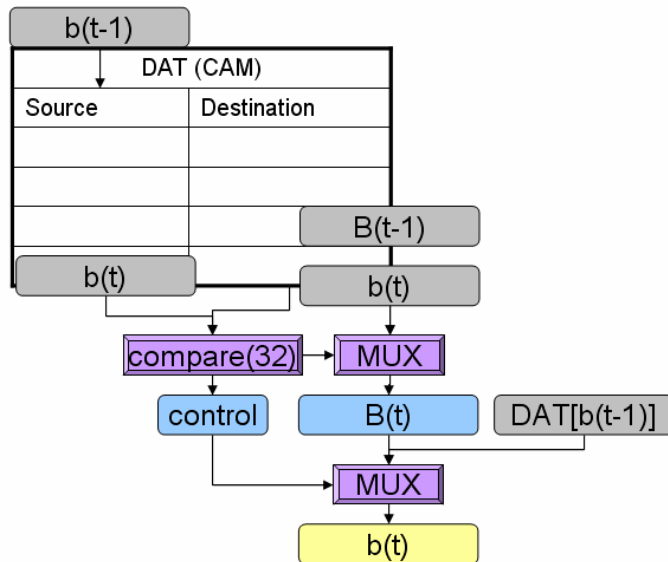
Logic 4-1: block information



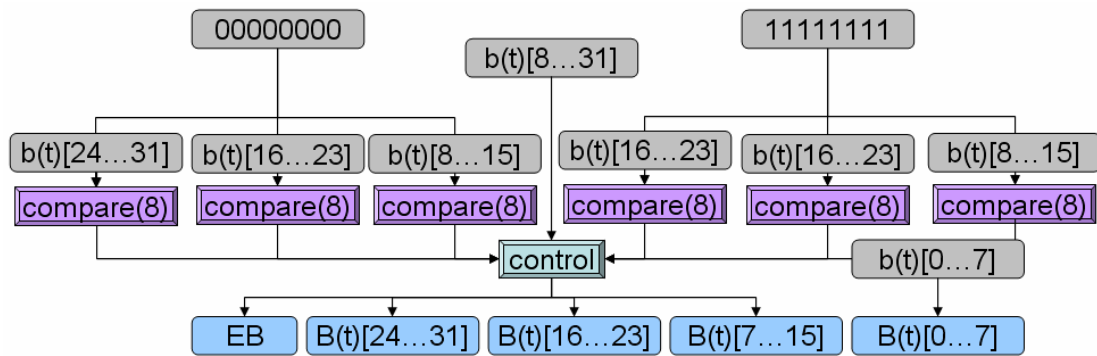
Logic 4-2: T0



Logic 4-3: T0-C



Logic 4-4: DAT



Logic 4-5: VL-Encoder (redundant type : insignificant bits)

4.3 Statistics and Simulation Result of Data Address Streams

4.3.1 Statistics

As mentioned in instruction address, I separate the data addresses into sequential and non sequential. In average, the ratio of sequential data address is 37.66% and the ratio of non sequential data address is 62.34%. These sequential data addresses can be utilize by variable stride algorithm and the Bus transaction will be 1.

In total data address, 63.85% of data address with 1 effective byte, 9.64% of data address with 2 effective bytes, 13.18% of data address with 3 effective bytes, and 13.33% of data address with 4 effective bytes. The number of effective bytes is equal to the number of Bus transactions with EB VL-Decoder.

4.3.2 Simulation Result

I propose three narrow Bus coding methods for data address in this thesis. First of them is to transplant the normal Bus variable stride algorithm (similar to T0) to narrow Bus. The second is that variable stride algorithm (similar to T0-C) and EB VL-Encoder work together. The third method is that variable stride algorithm (similar to T0-C), historical addresses algorithm and EB VL-Encoder work together.

Figure 4-3 is the Bus transactions ratio to narrow Bus without coding of historical addresses algorithm with different number of control lines. The Y-axis is the Bus transactions ratio of historical addresses algorithm comparing to narrow Bus without coding. The X-axis is the number of extra control lines used. These extra control lines include EB information therefore it begin from 2. (1 extra control line

can only indicate “this is not the last fragment of a codeword” and “this is the last fragment of a codeword”.) The Bus transactions ratio of unlimited extra control lines is about the same with 8-bit extra control lines. I choose 4-bit extra control lines (3-bit extra control lines except EB) as “good enough” choice and the effect of 8-bit extra control lines as the lower bound of Bus transactions ratio of historical addresses algorithm.

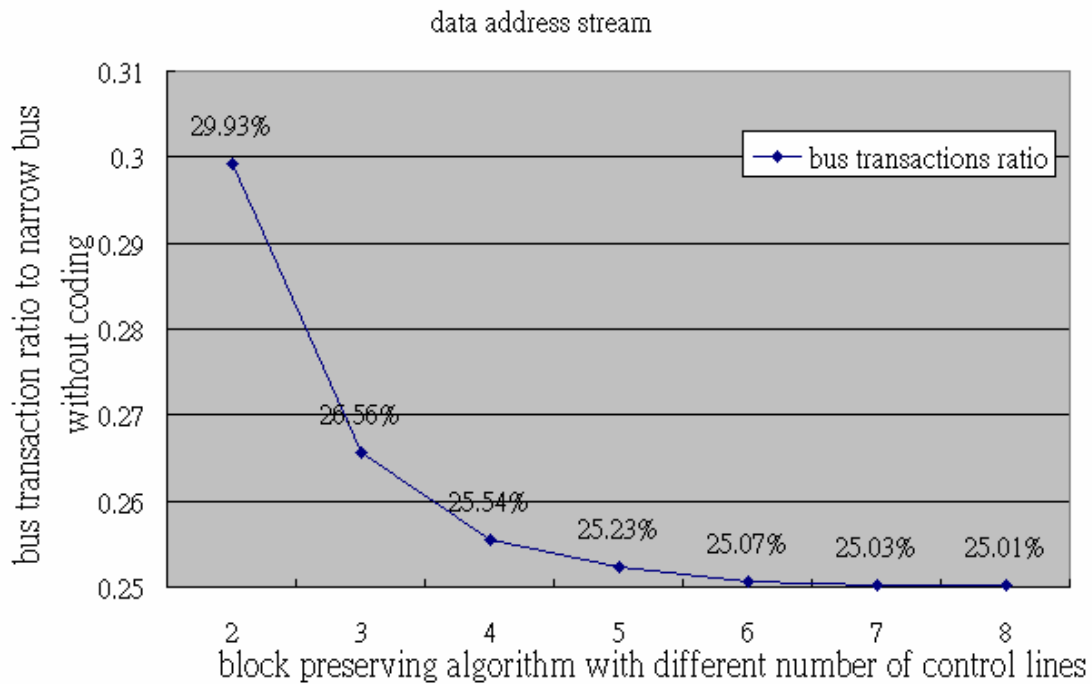


Figure 4-3: The Bus transactions ratio of different number of control lines using in historical addresses algorithm.

Figure 4-4 is the comparison of the proposed coding methods. The Y-axis is the Bus transactions ratio of the coding methods comparing to narrow Bus without coding. The X-axis lists the proposed three coding methods. The last value is the Bus transactions ratio of normal Bus. The Bus transactions ratio is 25%. The first method is to transplant the normal Bus variable stride algorithm (similar to T0) to narrow Bus. The second is that the variable stride algorithm (similar to T0-C) and EB VL-Encoder work together. The third, forth, fifth, and sixth method are that the variable stride algorithm (similar to T0-C), historical addresses algorithm and EB VL-Encoder work together. They are corresponding to 2-bit, 3-bit, 4-bit, and 8-bit (unlimited) extra control lines.

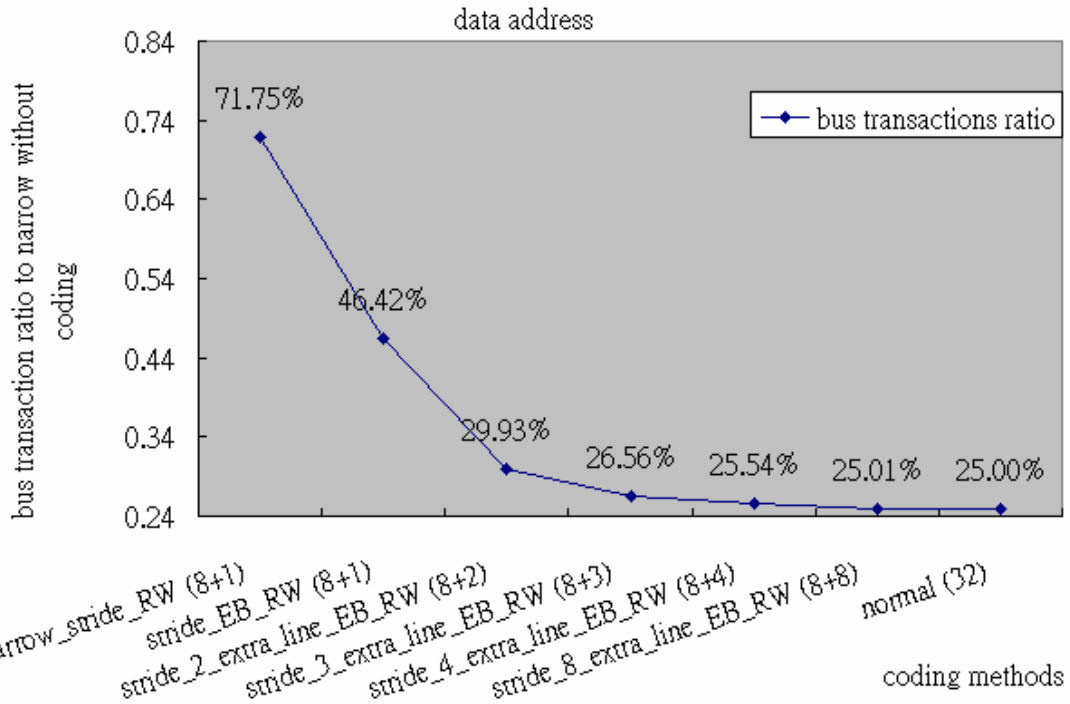


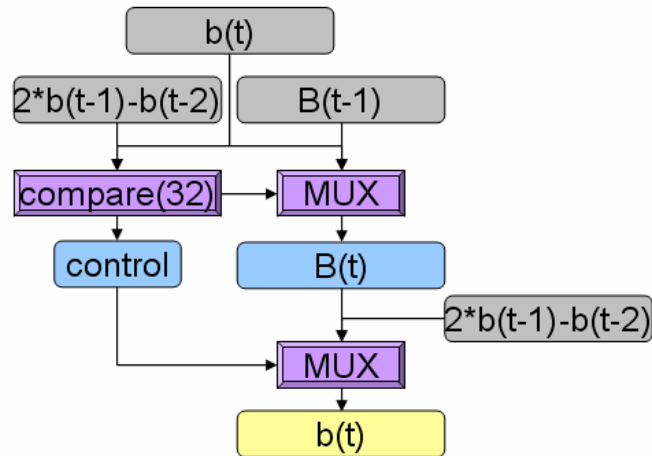
Figure 4-4: The Bus transactions ratio of proposed methods in data address stream.

The Table 4-2 shows the coding time delay represented by gate delay and the area overhead (ignoring the coding logic).

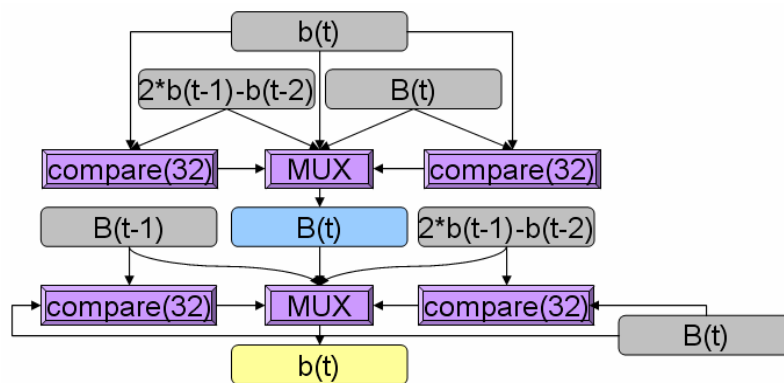
	Variable stride (similar to T0)	Variable stride (similar to T0-C)	Historical address (k extra lines)	EB
Gate delay (encoder)	Comparator(32) 2-to-1 MUX	Comparator(32) 3-to-1 MUX	Comparator(24) ($2^{k+1}-1$)-to-1 MUX	Comparator(8) 4-to-1 MUX
Gate delay (decoder)	2-to-1 MUX	Comparator(32) 3-to-1 MUX	($2^{k+1}-1$)-to-1 MUX	
Overhead	1 control line		K control line $2^{k+1}-2$ entry table	1 control line

Table 4-2

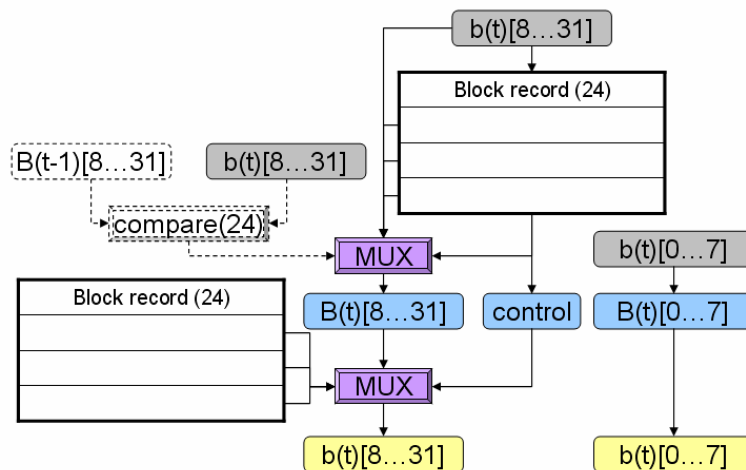
The followings are the coding logic diagram



Logic 4-6: variable stride algorithm (similar to T0)



Logic 4-7: variable stride algorithm (similar to T0-C)



Logic 4-8: historical addresses algorithm

4.4 Statistics and Simulation Result of Instruction Streams

4.4.1 Statistics

In instruction stream, the most often used 255 instructions are executed for 97.8% of the time and the most often used 256 instructions are executed for 98.34% of the time.

4.4.2 Simulation Result

In instruction stream, I compare two transmission methods. I will transplant the code compression method mentioned in chapter 2 from memory to Bus. The first transmission method is using the preserved index “00000000” to indicate that this is an index or a fragment of an instruction. The second transmission method is using VL-Encoder EB to indicate that. When transmitting an index, the EB will be set to 1. When the transmitted data is the fragments of an instruction, EB will be set to 0 except that the EB will be set to 1 at last Bus transaction. Figure 4-5 is the Bus transactions ratio to narrow Bus without coding of these two transmission ways.

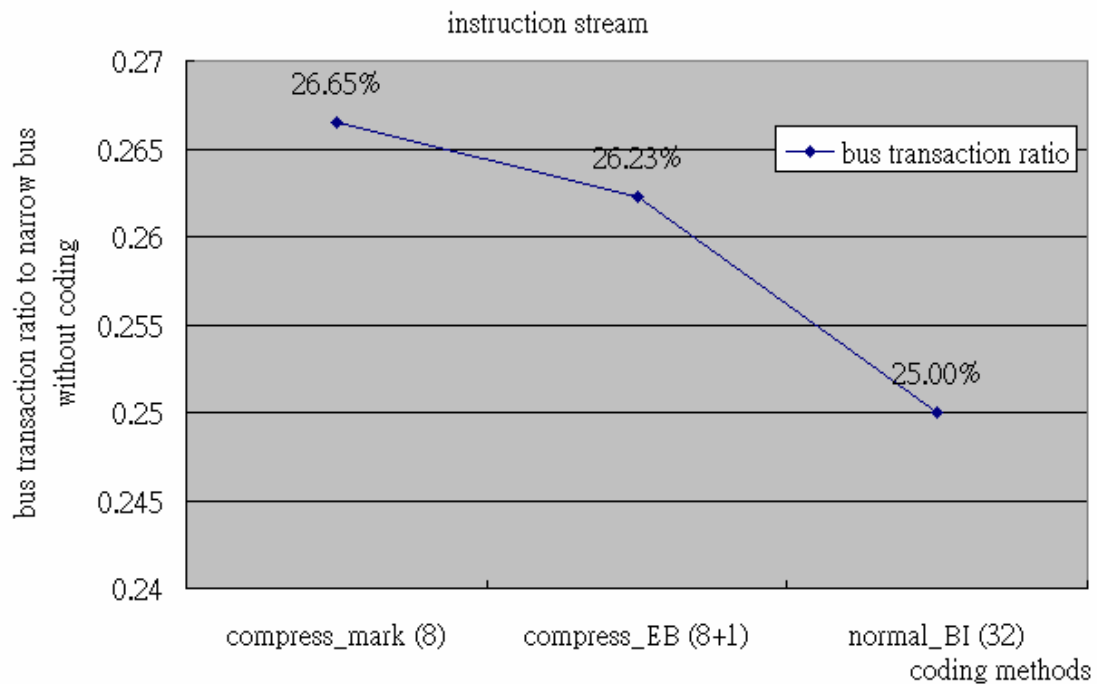


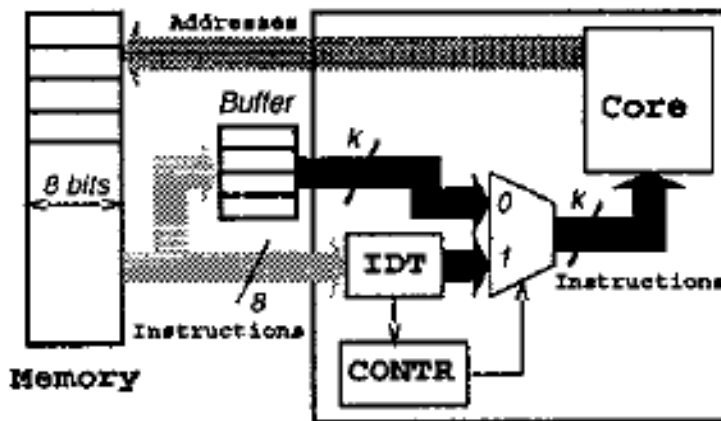
Figure 4-5 : The Bus transactions ratio of two transmission ways.

The Table 4-3 shows the coding time delay represented by gate delay and the area overhead (ignoring the coding logic).

	IDT + MARK	IDT + EB
Gate delay (encoder)	Static time	Static time
Gate delay (decoder)	255 entries Table lookup 2-to-1 MUX	256 entries Table lookup 2-to-1 MUX

Area Overhead	255 entry table	1 control line 256 entry table
Table 4-3		

The followings are the coding logic diagram



Logic 4-9: IDT, cited from “Selective Instruction Compression for Memory Energy Reduction in Embedded Systems”, L Benini, A Macii, E Macii, M Poncino, ISLPED, 1999

4.5 Statistics and Simulation Result of Data Streams

4.5.1 Statistics

In data stream, there is no encoding to the effective bits of codeword into redundant bits but the sender may utilize multiple kinds of redundant bits types. Therefore the statistics result will be very similar to simulation result. I will discuss these at next section.

4.5.2 Simulation Result

As describe in Table 3-6, I have five possible redundant type VL-Encoder policies. Figure 4-6 is the comparison of the different redundant type VL-Encoder policies. The first policy is using insignificant bits as redundant bits type and with embedded sign extension hint. The second is using repeated bits as redundant bits type. These two policies needn't extra control line except the EB. The third policy is using insignificant bits as redundant bits type and with additional sign extension hint. It needs 1-bit extra control line to provide the additional sign extension hint. The fourth is to utilize the insignificant bits as redundant bits type with embedded sign

extension hint and to utilize the repeated bits as redundant bits type together. It needs 1-bit extra control line to indicate these two types of redundant types. The fifth policy is to utilize the insignificant bits as redundant bits type with additional sign extension hint and to utilize the repeated bits as redundant bits type together. Because this policy needs 2 states to provide the sign extension hint and 1 state to indicate the redundant type is repeated bits, it needs three states (2-bit extra control lines) to indicate the redundant bits type.

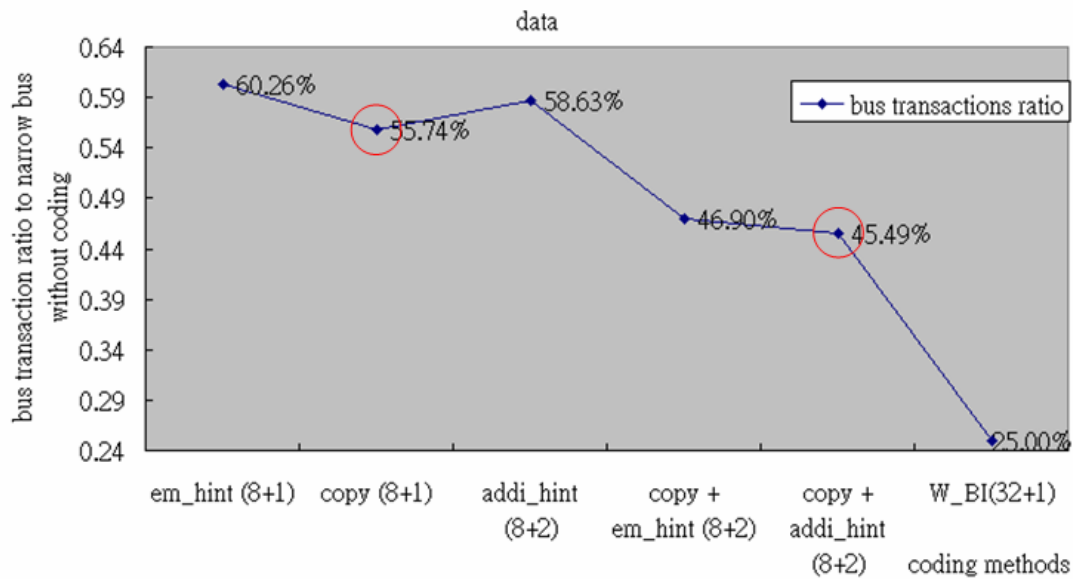


Figure 4-6: The Bus transactions ratio of five VL-Encoder policies.

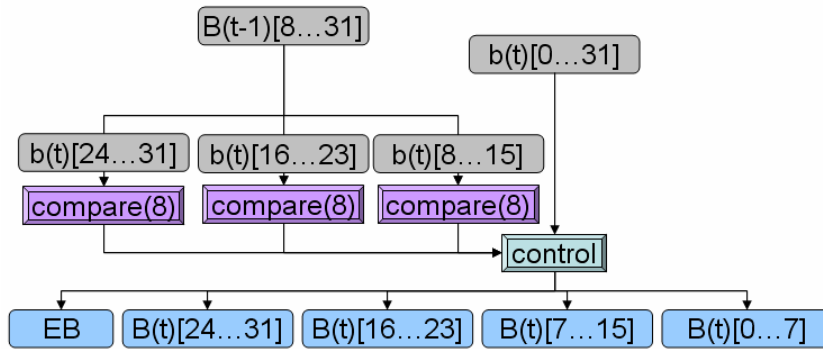
I make my decision depending on the simulation result. If I have only one extra control line (EB), I will use repeated bits as default redundant bits type. If I have two extra control lines, I will use repeated bits and insignificant bits with embedded sign extension hint as two redundant bits types. If there are three extra control lines can be used, I will use repeated bits and insignificant bits with additional sign extension hint as redundant bits types.

The Table 4-3 shows the coding time delay represented by gate delay and the area overhead (ignoring the coding logic).

	Repeated bits	Insignificant bits	Insignificant bits
Deal with	Copy	Embedded hint	Additional hint
Gate delay (encoder)	Comparator(8) 4-to-1 MUX	Comparator(8) 4-to-1 MUX	Comparator(8) 4-to-1 MUX
Area Overhead	1 state	1 state	2 states

Table 4-4

The followings are the coding logic diagram



Logic 4-10: VL-Encoder (redundant type : repeated bits)

4.6 The Simulation Results and Comments of these Streams

Table 4-5 is the simulation results summary. The red portion represents the entries which when comparing to normal Bus, the Bus transactions overhead is lower than 10%.

	Instruction Address	Data Address	Instruction	Data
0 control line			<u>26.65%</u>	
1 control line	<u>26.53%</u>	46.42%	<u>26.23%</u>	55.74%
2 control lines	<u>25.13%</u>	29.93%		45.49%
3 control lines		<u>26.56%</u>		
8 control lines		<u>25.01%</u>		

Table 4-5

4.6.1 Instruction Address

In instruction address stream, the number of Bus transactions of sequential instruction address (91%) can be reduced to 1 by T0 (or T0-C). The number of Bus transactions of repeated occurrence the same branch source and target (8.83%) can be reduced to 1 by NBDAT. I can use 8-bit width Bus and 2-bit extra control lines to utilize both these two properties and reduce the Bus transaction of the 99.83% instruction address to 1. There is only the 0.17% unpredictable first occurrence branch instruction addresses need more than 1 Bus transaction. The Bus transactions of these unpredictable instruction addresses have chance to be reduced by EB because of spatial locality.

4.6.2 Data Address

In data address stream, the sequential data address (37.66%) can be reduced to 1 by variable stride algorithm. The more extra control lines added the more Bus transactions can be reduced. When there are 8-bit width Bus with 8-bit extra control lines (total 16-bit width Bus lines), the ratio of Bus transaction is reduce to 25.01%. This is only 0.01% of Bus transactions ratio different from 32-bit width normal Bus.

The reason that we can reduce the data address Bus transactions ratio to 25.01% (better than instruction address) is mainly on test pattern. When I use Mibench to do simulation, there is only one program executed at the same time. This will cause the data being located too centralize.

The historical addresses algorithm will record the occurred 1-byte-address, 2-byte-address, and 3-byte-address. After all 3-byte-addresses in a program are recorded, all data addresses need only 1 Bus transaction. Due to test patterns and program size, 256 entries historical addresses table can record all 3-byte-addresses in my simulation program (Mibench). However, before all 3-byte-addresses in a program are recorded, some data addresses need more than 1 Bus transaction. We can reduce the Bus transaction ratio to 25.01% and there is only 0.01% to reach 25%. This 0.01% Bus transaction ratio is wasted on historical addresses table initialization.

4.6.3 Instruction

In instruction stream, I use “Selective Instruction Compression for Memory Energy Reduction in Embedded Systems” to compress code to reduce Bus transactions. Under Mibench, 98.35% instructions can be transmitted by 1 Bus transaction in average. In worst case, there are still 87.54% instructions can be saved to IDT and be transmitted by 1 Bus transaction.

4.6.4 Data

Due to the data types, the sizes of data may be byte, half word, word and so on. In normal Bus, these data are all transmitted on 32-bit Bus even if the data size small than 32 bits. On 8-bit narrow Bus environment, the proposed methods can transmit data with smaller size by fewer Bus transactions without mistake.

The general researches on discussing data size focus on insignificant bits and the data relationships. The proposed methods can transmit only the significant bytes to reduce Bus transactions. Besides, we forsake the repeated bytes without sending to utilize the data relationship.

In general, the data value is unpredictable and almost random distribution. We can hardly predict the exact value and we can only predict the range. There is no encoding can be applied on data stream to reduce the range of data value. The number of bytes which are needed to be transmitted depends on the range of data values. In other words, we can hardly use encoding to reduce the number of Bus transactions.

There may be other redundant bits type can be used such as floating point. No matter in exponent part or mantissa part, there is high chance that there is a string of 0's or 1's. But these properties are highly application specific; I won't discuss this subject in this thesis.

4.6.5 Summary

Adopting narrow Bus to system without coding will bring severely performance loss and this is hard to be tolerated in most situations. However, if we do some coding on narrow Bus system, this problem can be reduced to a very slight degree. According to the simulation and description above, we can know that we don't always need 32-bit width Bus if we can tolerate slightly performance loss and the coding delay. This will cause more systems being willing to use narrow Bus.

Chapter 5: Summary, Discussion, and Conclusion

5.1 The Effect of the Narrow Bus Encoding

The Bus width and the number of Bus transactions are inverse proportion when the Bus width is narrower than data width. The decision of Bus width should be a problem of trade-off. However, many systems which have high area constraint but they may not tolerate the severely performance loss and extra static energy consumption caused by additional Bus transactions. For the narrow Bus without coding environment discussed in this thesis, the system uses additional three times of Bus transactions to exchange the 24-bit width Bus saving is an unwise behavior. This should be the reason of why there is almost no system uses narrow Bus. As

most low power or high performance researches, we hope to use a few overhead to get high advantage. Narrow Bus encoding provides a compromise choice for system designers. It can save large external Bus area by only a few number of additional Bus transactions.

5.2 The Coding Methods Effects on Bit Toggles

In my thesis, I only discuss the number of Bus transactions. The number of Bus transactions can affect the performance and energy consumption at the same time. For a system which cares about energy consumption, we should take the bit toggles into consideration. Because of the total Bus transactions are reduced, the total bit toggles are reduced as side effect.

The side effect on bit toggle can be separated into two parts. First, some of the coding methods use a special signal to inform the receiver the transmitted data and set the Bus value as don't care. These coding methods can reduce bit toggles. Second, some of the coding methods such as code compression algorithm which doesn't care about bit toggle. This will affect the number of bit toggle but it is random effect.

The objective of this thesis is to reduce Bus transactions. The proposed methods can reduce bit toggles as side effect. However, these methods are not designed to reduce bit toggle. If a system cares about the energy consumption, it should evaluate the weights of the static energy caused by additional Bus transactions and the dynamic energy caused by bit toggles. It can adopt the power model of system and of Bus to find the most suitable method easily.

5.3 The Coding Methods Effects on Performance

The number of Bus transactions will affect the system performance. However, they are not direct proportion. When the system executes programs with pipeline, there will be some pipeline stall cycles can fill the extra Bus transactions. This will reduce the performance loss caused by extra narrow Bus transactions. Besides, one pipeline cycle may be able to fill multiple Bus transactions if there is hardware support. However, this involves the system implementation and has no relation with this thesis.

5.4 The Other Bus Architectures

In this thesis, we discuss the behavior of the four streams independently. However, there are some systems with only 2 Buses or only 1 Bus as figure 5-1 and figure 5-2.

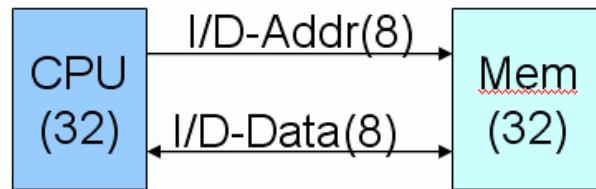


Figure 5-1 : The 2 Buses architecture

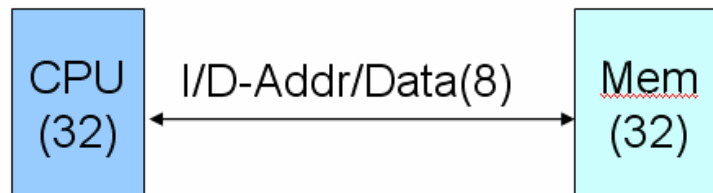


Figure 5-2 : The 1 Bus architecture

The mixed data broke the properties of individual stream. A trivial solution is to add extra control lines to separate these streams. For 2 Buses architecture in Figure 5-1, we need only 1 extra control line to separate two types of stream in a Bus. For the 1 Bus architecture systems shown in Figure 5-2, we need only 1 extra control line, too. This is because whenever an address is transmitted, there will be a data followed. The function of the extra control line is to indicate that this address is instruction address or data address.

We need different number of extra control lines to encode different streams. On the mixed data stream, the number of extra control lines depends on the stream which requires most. That is waste for other streams. This will be another consideration of mixed streams Bus architecture.

The information of different streams on a Bus can be utilized by each other. Take the instruction addresses and data addresses mixed Bus for example, the instruction address can provide the hints of data address stride value. The same instruction address means the same instruction is executed. The accessed data addresses of this instruction will be regular. The same idea can be adopted on the Bus which transmits instruction stream and data address stream on the same Bus.

5.5 The coding (encoding and decoding) delay affection on memory access stage:

stage:

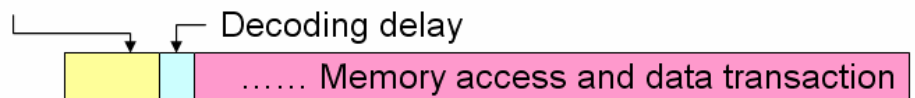
- ✧ There is enough free time intervals in memory access stage to fit coding (encoding and decoding) time.
 - The coding delay won't effect memory access stage cycle.
- ✧ There is no enough free time intervals in memory access stage to fit coding (encoding and decoding) time.

1. Increase the memory access stage cycle.
2. Using extra pipeline stage cycle to do coding.

I use address Bus transaction to illustrate. It is the same at all streams.

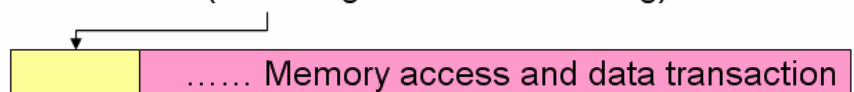
1. There is enough free time interval to fit decoding but there is no enough free time interval to fit encoding.

Address bus transaction



1. When it needs only 1 Bus transaction :
 - => it needs 1 extra pipeline stage cycle to do encoding
2. When it needs more than 1 Bus transaction :
 - => Encoding can be done at first Bus transaction and it doesn't need extra pipeline stage cycle.
2. There is no enough free time intervals to fit coding (encoding and decoding) delay.

No address bus transaction (no enough time for decoding)



1. Encoding can be done at first Bus transaction and decoding can be done at last Bus transaction. This need 1 extra pipeline stage cycle.

It is worthy to do narrow Bus encoding even if the coding delay need 1 pipeline stage cycle time.

- ✧ The worst case is that there is no enough free time intervals in memory access stage to fit both encoding and decoding time.
 - Every data needs 1 more pipeline stage cycle to do coding (encoding and decoding), it is equal to that each data need 1 more Bus

transaction.

- ✓ The Bus transaction ratio will be increased by 25%.
- ✓ The narrow Bus transaction ratio reductions are all more than 40% in all streams.
- ✓ It is worthy to do narrow Bus encoding even if the coding delay need 1 more pipeline stage cycle.

5.6 Reducing the coding delay effect on pipeline stage:

✧ DAT:

Looking up table can be done before the discontinuous address occurs.

- IA(t) : instruction address at time t
- Time = t

Look up IA(t) in source address entries of DAT

- ✓ Case 1 : IA(t) is in source address entries of DAT
Extract the destination address of IA(t) in DAT.
- ✓ Case 2 : IA(t) is not in source address entries of DAT
IA(t+1) cannot utilize DAT to reduce Bus transactions.

- Time = t + 1 (Case 1)

Comparing this destination address with IA(t+1)

- ✓ Case 1 : IA(t+1) is the same with the destination address
Sending a specific signal instead of sending IA(t+1).
- ✓ Case 2 : IA(t+1) is not the same with the destination address
IA(t+1) cannot utilize DAT to reduce Bus transactions.

Q&A

Q1:

The proposed methods are many and diverse. What is the main idea of the narrow Bus encoding? (chapter 3.1)

A:

Reduce transmitted bytes of data as long as receiver can get exact data.

✧ VL-Encoder :

- The sender can skip the leading regular bytes of data without sending, and inform the receiver what the regularity is.
 - ✓ Using additional information to inform the sender what the regularity is.
 - ✓ Using additional information to inform the sender the number of Bus transactions.

✧ VL-Encoder :

- Increasing the leading regular bytes in each 32-bits data.

Q2:

When considering the energy consumption, the bit toggles play an important role. Though the purpose of this thesis isn't on low energy, how is effect of the narrow Bus encoding on bit toggles?

A:

The proposed methods are designed without considering bit toggles.

System static energy consumption:

The narrow Bus encoding can reduce Bus transaction.

- ✧ Fewer Bus transaction => Shorter program execution time
- ✧ Shorter program execution time => Lower system static energy consumption

The bit toggles of transmitted bytes:

- ✧ Coding methods which have good effect:
 - Some of the coding methods use a specific signal to inform the receiver the transmitted data and see the Bus value as “don't care”.
=> The Bus value can be set to reduce bit toggles except the control line.
- ✧ Coding methods which have unpredictable effect:

- Some of the coding methods don't consider the bit toggles. The effect on bit toggle is unpredictable.

Q3:

The simulations are basing on the traces extracted by O1 compiler. How about the effect on the O3 or other level compiler?

A:

My research focuses on the streams characteristic. I don't consider the difference between O1 and O3 compiler.

Q4:

No matter what stream, you propose several coding methods. What method you suggest to use in each stream? (chapter 1.3)

A:

The product of Bus width and Bus transaction ratio

- ✧ When the Bus width is smaller than data width and the Bus width is the power of 2, the Bus width and the Bus transaction ratio is inverse proportion. I use the product of Bus width and Bus transaction ratio as evaluation matrix.

The commended method of each stream evaluated by this function

✧ Instruction Address

- T0-C + EB
- 1 extra control line
- I don't consider the area overhead of NBDAT table and the "2 control line method" has gotten worse ratio. If the "2 control lines method" considers the area overhead of NBDAT, the product ratio will be worse.

	Coding method	Bus transaction ratio
1 control line	<u>T0-C + EB</u>	<u>26.53%</u>
2 control lines	T0-C + NBDAT(unlimited) + EB	25.13%

✧ Data Address

- No control line variable stride algorithm + historical addresses algorithm (with 6 historical address entries) + EB
- 3 extra control lines

	Coding method	Bus transaction

		ratio
1 control line	No control line variable stride algorithm + EB	46.42%
2 control lines	No control line variable stride algorithm + historical addresses algorithm (with 2 historical address entries) + EB	29.93%
3 control lines	<u>No control line variable stride algorithm + historical addresses algorithm (with 6 historical address entries) + EB</u>	<u>26.56%</u>
...
8 control lines	No control line variable stride algorithm + historical addresses algorithm (with 254 historical address entries) + EB	25.01%

✧ Instruction

- Using “MARK” to indicate whether this instruction is in IDT or not
- 0 extra control line

	Coding method	Bus transaction ratio
0 control line	<u>Using “MARK” to indicate whether this instruction is in IDT or not</u>	<u>26.65%</u>
1 control line	Using “EB” to indicate whether this instruction is in IDT or not	26.23%

✧ Data

- utilize both insignificant bit and repeated bit + EB
- 2 extra control lines

	Coding method	Bus transaction ratio
1 control line	utilize repeated bit + EB	55.74%
2 control lines	<u>utilize both insignificant bit and repeated bit + EB</u>	<u>45.49%</u>

These methods need different number of extra control lines and get different Bus transactions reduction. There may be systems cannot tolerate the area overhead of suggest method. They can choose the suitable coding method from the tables.

Q5:

You have an assumption that the area overhead of extra coding logic gates is much slighter than the extra control lines. Is this assumption reasonable? (chapter 1.3)

A:

Coding logic gates area depends on:

✧ Processing technology

Extra control lines area depends on:

✧ Processing technology

✧ Routing length

The needed coding logic gates and extra control lines are provided. The system designer can evaluate easily.

My thesis environment focuses on the external Bus between processor and memory. The assumption that the area overhead of extra coding logic gates is much slighter than the extra control lines is reasonable.

Q6:

There are many coding methods can inform the receiver end the number of Bus transactions. Why do you use the method which adds an extra control line to indicate the number of Bus transactions? (chapter 3.2.1, 3.2.2, 3.2.3)

A:

If I want to inform the receiver end the number of Bus transactions, I have to transmit extra information. There are two main directions to reach this purpose.

Policy 1 :

✧ Using an extra control line (EB) to indicate a codeword is transmitted over or not at every Bus transaction.

Policy 2 :

✧ Adding several extra bits to indicate the number of Bus transactions.

The reason to use policy 1:

✧ In order to compare these two policies in the same standard, I use the same Bus width (including control lines) for both policies to compare the Bus transaction ratio of these two methods.

➤ The policy 1 can get lower Bus transaction ratio.

Q7:

You seem to reduce the number of Bus transaction to a very low degree. However, why do you think these ratios are low enough? (chapter 5.8)

A:

	Instruction Address	Data Address	Instruction	Data
0 control line			<u>26.65%</u>	
1 control line	<u>26.53%</u>	46.42%	<u>26.23%</u>	55.74%
2 control lines	<u>25.13%</u>	29.93%		45.49%
3 control lines		<u>26.56%</u>		
8 control lines		<u>25.01%</u>		

Instruction Address Stream:

- ✧ Sequential addresses (T0, T0-C)
The number of Bus transaction can be reduced to 1.
- ✧ Non-sequential addresses pairs (NBDAT)
 - When executing a branch instruction, it will branch from source address to target address. The source address and target address is a non-sequential addresses pair.
 - First occurrence non-sequential addresses pair:
The number of Bus transactions is not guaranteed to be 1.
 - After second occurrence non-sequential addresses pair:
The number of Bus transactions can be reduced to 1.

The Bus transactions of those first occurrence non-sequential addresses pairs can be reduced by EB, but they are not guaranteed to be 1. Only these addresses may cause more than 1 Bus transaction.

Data Address Stream:

- ✧ Sequential addresses (Variable stride algorithm)
 - The number of Bus transaction can be reduced to 1.
- ✧ Locality properties (Historical addresses --- Described in section 3.4.2 encoder 2 ~ encoder 4)
 - The historical addresses algorithm will record the occurred 1-byte-address, 2-byte-address, and 3-byte-address.

- After all 3-byte-addresses in this program are recorded; all data addresses need only 1 Bus transaction.
 - ✓ Due to test patterns and program size, 254 entries historical addresses table can record all 3-byte-addresses in my simulation program (Mibench).
- ✧ Initialization
 - Variable stride algorithm:
 - ✓ Before get right stride value, some data addresses need more than 1 Bus transaction.
 - ✓ When updating right stride value, EB can help to reduce the Bus transactions, but they are not guaranteed to be 1.
 - Historical addresses algorithm:
 - ✓ Before all 3-byte-addresses in this program are recorded, some data addresses need more than 1 Bus transaction.
 - ✓ When initializing historical addresses table, EB can help to reduce the Bus transactions, but they are not guaranteed to be 1.
 - ✓ When initializing historical addresses table, 2-byte-address and 1-byte-address recorders can help to reduce the Bus transactions, but they are not guaranteed to be 1.

The Bus transactions of those not utilized by variable stride and not recorded in 3-byte-address table addresses can be reduced by EB, but they are not guaranteed to be 1. Only these addresses may cause more than 1 Bus transactions. Besides, the table size and control lines cannot be unlimited. The effect of proposed method will be further limited.

The proposed methods can reduce the Bus transactions of almost all data addresses to 1. The situation that the address need more than 1 Bus transaction is when the table or stride value is under initialization. I consider that the proposed method can reduce the Bus transactions low enough.

Instruction Stream:

- ✧ **Lower Bus transaction ratio than other streams when using the same number of control lines:**
 - Even if there is no extra control line, the Bus transaction ratio of instruction stream is about the same with instruction address stream using 1 extra control line. And it is about the same with data address stream using 3 extra control lines.

- ✧ **Some surveyed but not suitable compression methods**
 - **VLC coding is not suitable in my environment**
The VLC coding can be used for compressing a whole program but it is not suitable to compress instructions one by one. However, the “Selective Instruction Compression for Memory Energy Reduction in Embedded Systems” is very suitable to compress instruction one by one.
 - **Compressing operands is not suitable in my environment**
Though some operands can be compressed to very small, some operands may be still very large. The Bus transactions are the ceiling of the quotient of the compressed size divided by 8. It doesn't consider this and get lower Bus transaction reduction than “Selective Instruction Compression for Memory Energy Reduction in Embedded Systems” in my environment.

- ✧ **The “Selective Instruction Compression for Memory Energy Reduction in Embedded Systems” is suitable in my environment**
 - Not all programs execute lower than 256 kinds of instructions. That is, not all instructions can be transmitted by 1 Bus transactions.
 - ✓ We can increase the table size to store more instructions.
However, this will further increase the table access delay and need more bit for indexing.
 - In average, there is only 1.65% instructions need more than 1 Bus transaction. I think it is not worthy to double the table size and increase the Bus width by 1.

As the reasons mentioned above, I think that the “Selective Instruction Compression for Memory Energy Reduction in Embedded Systems” is the most suitable compression method which I have surveyed and the Bus transaction ratio is low enough.

Data Stream:

- ✧ **Different data type:**
Due to the data types, the sizes of data may be various. In normal Bus, these data are all transmitted on 32-bit Bus even if the data size smaller than 32 bits. On 8-bit narrow Bus environment, the proposed methods can transmit data with small size by fewer Bus transactions without mistake.
- ✧ **The general researches on the data size in time critical environments**

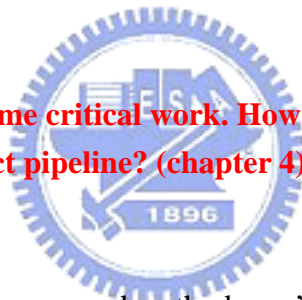
focus on two main points

- Insignificant Bits (sign extension)
- Repeated Bits (relationships between data)
- ✧ **The reason that I think that the data stream needn't encoding.**
 - The relationship between 2 data
 - ✓ Using repeated bit as redundant type is to utilize the relationship between 2 data.
 - The relationship between more than 2 data
 - ✓ It is highly program dependent. It is hard to find a common relationship in all programs.
 - Some data value will repeatedly occur
 - ✓ The probability is very low and hardly to catch the appearance.

The data stream only utilizes the most efficient properties. The other properties are unobvious and hardly to utilize. I think that it is good enough to use just insignificant bits and repeated bits as redundant bits types in data stream.

Q8:

The Bus encoding is a time critical work. How do you convince us that the proposed method won't affect pipeline? (chapter 4)



A:

I cannot convince that the proposed method won't affect pipeline. I can only propose the coding (encoding and decoding) delays of coding methods for system designer to estimate.

The coding (encoding and decoding) delays of coding method:

✧ **Instruction Address**

	T0	T0-C	DAT(NBDAT)	EB (Repeated bits)
Gate delay (encoder)	Comparator(32) 2-to-1 MUX	Comparator(32) 3-to-1 MUX	Comparator(32) 2-to-1 MUX	Comparator(8) 4-to-1 MUX
Gate delay (decoder)	2-to-1 MUX	Comparator(32) 3-to-1 MUX	2-to-1 MUX	

✧ **Data Address**

	Variable stride	Variable stride	Historical	EB (Repeated
--	-----------------	-----------------	------------	--------------

	(similar to T0)	(similar to T0-C)	address (k extra lines)	bits)
Gate delay (encoder)	Comparator(32) 2-to-1 MUX	Comparator(32) 3-to-1 MUX	Comparator(24) ($2^{k+1}-1$)-to-1 MUX	Comparator(8) 4-to-1 MUX
Gate delay (decoder)	2-to-1 MUX	Comparator(32) 3-to-1 MUX	($2^{k+1}-1$)-to-1 MUX	

✧ **Instruction**

	IDT + MARK	IDT + EB
Gate delay (encoder)	No run time delay	No run time delay
Gate delay (decoder)	255 entries direct map cache look up 2-to-1 MUX	256 entries direct map cache look up 2-to-1 MUX

✧ **Data (VL-Encoder)**

	Repeated bits	Insignificant bits	Insignificant bits
Handle Method	Copy	Embedded hint	Additional hint
Gate delay (encoder)	Comparator(8) 4-to-1 MUX	Comparator(8) 4-to-1 MUX	Comparator(8) 4-to-1 MUX
Gate delay (decoder)	No delay	No delay	No delay

94

碩士論文

減少傳送次數之窄匯流排編碼

交通大學

94

碩士論文

減少傳送次數之窄匯流排編碼

交通大學

94

碩士論文

減少傳送次數之窄匯流排編碼

交通大學

94

碩士論文

減少傳送次數之窄匯流排編碼

交通大學

94

碩士論文

減少傳送次數之窄匯流排編碼

交通大學



資資
訊訊
科學
學院
與
工程
研究
所

鄭式勳

9317580

資資
訊訊
科學
學院
與
工程
研究
所

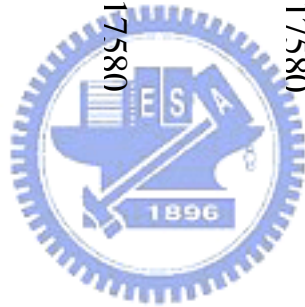
鄭式勳

9317580

資資
訊訊
科學
學院
與
工程
研究
所

鄭式勳

9317580



資資
訊訊
科學
學院
與
工程
研究
所

鄭式勳

9317580

資資
訊訊
科學
學院
與
工程
研究
所

鄭式勳

9317580