

國立交通大學

電機資訊學院 資訊學程

碩士論文

TI TMS320C55x' DSP 架構下執行模乘法運算之效率研究

A study of the performance of operating modular
multiplication based on TI TMS320C55x' DSP



研究生：薛明宏

指導教授：葉義雄 教授

中華民國 九十四年六月

TI TMS320C55x' DSP 架構下執行模乘法運算之效率研究

A study of the performance of operating modular
multiplication based on TI TMS320C55x' DSP

研究生：薛明宏

Student : Ming-Hung Hsueh

指導教授：葉義雄

Advisor : Dr. Yi-Shiung Yeh

國立交通大學

電機資訊學院 資訊學程

碩士論文

A circular logo for National Chiao Tung University, featuring a gear-like border and a central emblem with a book and a stylized 'NCTU' monogram.

A Thesis

Submitted to Degree Program of Electrical Engineering and Computer Science

College of Electrical Engineering and Computer Science

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

In

Computer Science

June 2005

Hsinchu, Taiwan, Republic of China

中華民國九十四年六月

TI TMS320C55x' DSP 架構下執行模乘法運算之效率研究

學生: 薛明宏

指導教授: 葉義雄博士

國立交通大學電機資訊學院 資訊學程 (研究所) 碩士班

摘 要

現階段由於無線通訊技術發達，許多無線移動運算平台也應運而生，例如，無線電話、個人數位助理及具無線上網功能之筆記型電腦等。由於移動計算平台與基地台溝通的媒介乃是空氣，因此加強資訊安全保護便成了移動計算平台上一個重要的課題。採用加密系統時除了安全性外，最重要的便是運算效能，由於模乘法乃現有密碼系統中最主要的計算元件，加速模乘法的運算將可直接提昇整體密碼系統之效能。本文針對適用於無線環境多功能的CPU TI TMS320C55'x DSP，進行其架構研究，發展出一套系統性的模乘法選用法則，並在研究過程中發展出一套新的模乘法，此模乘法的運算效能較Morita & Yang演算法稍佳。

A study of the performance of operating modular
multiplication based on TI TMS320C55x' DSP

Student: Ming-Hung Hsueh Advisor: Dr. Yi-Shiung Yeh

Degree Program of Electrical Engineering Computer Science
National Chiao Tung University

Abstract

Nowadays, because of the rapid progressive of wireless communication, a variety of mobile computational platforms are emerged, such as mobile phone, PDA, and notebook. The medium between mobile stations and base stations is air. As a result, to protect the information of mobile platform becomes an important issue. TI TMS320c55x' DSP is a high performance CPU, which can handle multimedia, voice compression, and voice communication, and also, has highly performance with low power consumption. Therefore, it is an ideal multipurpose CPU that can be introduce to mobile platform. In this thesis, we introduce a systemic model to estimate the performance of executing a modular multiplication on TI TMS320c55x' DSP. Moreover, we propose a new modular multiplication algorithm, which achieves higher performance than Morita & Yang's Algorithm.

致 謝

首先感謝指導教授葉義雄教授，在研究所求學階段授與我專業知識及正確的研究態度。其次感謝公司長官們的包容讓我在工作後還能有機會充實自我，更要感謝同事豐富、瑞敏及嘉耀在我求學過程中所給予的協助。

最後感謝我的家人的支持，尤其是內人莉淳這段日子辛苦負擔了大部分的家務與女兒們的教養，使我能無後顧之憂，安心完成學業。



Contents

中文摘要.....	i
英文摘要.....	ii
致謝.....	iii
List of Figures	vi
List of Tables.....	vii
Chapter 1 Introduction	1
1.1 Modular Multiplication.....	1
1.2 The Goal of This Thesis.....	2
Chapter 2 Implementation of the Basic Arithmetic of Modular Multiplication	
Algorithm on TI TMS320c55x' DSP	3
2.1 Architecture of TI TMS320c55x' DSP.....	3
2.2 Arithmetic Instructions of TI TMS320c55x' DSP.....	5
2.2.1 The Registers	5
2.2.2 Left Shift	5
2.2.3 Addition.....	6
2.2.4 Subtraction	6
2.2.5 Multiplication.....	7
2.3 The Implementation issues of Frequent Used Arithmetic of Modular	
Multiplication Algorithm on TI TMS320c55x' DSP.....	9
2.3.1 Pipeline Protection Issue.....	9
2.3.2 Arithmetic of Modular Multiplication Algorithm on TI	
TMS320c55x' DSP	11
Chapter 3 Background on Iterative Modular Multiplication	13
3.1 Single Precision Left-to-Right Convention Algorithm.....	13
3.1.1 Traditional Algorithm of Modular Multiplication.....	13
3.1.2 Blakley Algorithm[5].....	15
3.1.3 Chiou & Yang's Algorithm[2].....	17
3.2 Multi-Precision Left-to-Right Convention Algorithm.....	20
3.2.1 Leong, Tan & Tan's Algorithm [8].....	20
3.2.2 b-ary number system.....	22
3.2.3 Morita & Yang's Algorithm[6].....	22
Chapter 4 A New Iterative Modular Multiplication Algorithm	25
4.1 The New Modular Multiplication Algorithm	25
4.1.1 Procedure	25
4.1.2 Correctness.....	27
4.2 Implement Issues	30
4.2.1 Calculate $P' \leftarrow P' - q \times D$ with two-digit q	30

4.2.2 Calculating $b^{n+1} \pmod{D}$	31
4.2.3 Complete Algorithm.....	33
4.3 Special Case	34
4.4 Complexity.....	35
4.4.1 General Purpose CPU	35
4.4.2 TI MS3200C 55x' DSP	35
4.4.3 Storage Complexity	35
4.6 Compare with Other Algorithms	36
Chapter 5 Conclusion.....	37
REFERENCES	38



List of Figures

Figure 2-1 The CPU Diagram[13]	3
Figure 2-2 The Data Computation Unit Diagram[13]	4
Figure 2-3 Left Shift Instructions[11]	6
Figure 2-4 Addition Instructions[11]	6
Figure 2-5 Subtraction Instructions[11]	7
Figure 2-6 Multiplication Instructions (1) [11]	8
Figure 2-7 Multiplication Instructions (2) [11]	8
Figure 2-8 1024-Bit addition code with RAW hazard	9
Figure 2-9 1024-Bit addition code with less RAW hazard	11



List of Tables

Table 2-1 The clock cycles consumption of frequent used arithmetic of modular multiplication algorithm.....	11
Table 2-2 The weighted clock cycles consumption of frequent used arithmetic of modular multiplication algorithm on TI TMS320c55’	12
Table 3-1 The time complexity of Traditional Algorithm.....	14
Table 3-2 The time complexity of Blakely’s Algorithm	15
Table 3-3 The time and storage complexity of Chiou & Yang’s Algorithm.....	19
Table 3-4 The time complexity of Leong, Tan & Tan’s Algorithm.....	21
Table 3-5 The time complexity of Morita & Yang’s Algorithm.....	24
Table 4-1 The computational effort of Proposed Algorithm.....	35
Table 4-2 Comparing with other iterative algorithm by weighted clock cycles consumption.....	36



Chapter 1 Introduction

1.1 Modular Multiplication

Diffie & Hellman[15] proposed the public key concepts in 1976. Since then, a lot of famous elegant public key crypto-systems had been proposed, such as ElGamal [14], in which the security is based on the difficulty of discrete logarithm problem, and RSA[9], in which the security is based on the large prime problem. Those cryptosystems open a new era of cryptology. Nowadays, crypto-system can provide not only confidential of information but also authenticity, integrity, and nonrepudiation. The progression of public key cryptosystems provides variety usages, for example, sharing key between strangers, making e-transaction reliably, and authenticating the issuer of information.

The core operation of some important public key cryptosystems, such as RSA, ELGamal, and DSA[4] is modular exponentiation. Executing a modular exponentiation takes the longest time running a public key encryption or decryption. As a result, improving the performance of executing modular exponentiation becomes a key point.

There are two main schemes to achieve higher performance of executing a modular exponentiation. First, reduce the amount of modular multiplications, such as Knuth's[3] M-ary approach and Chiou's Parallel scheme[1]. Second, improve the performance of executing a modular multiplication, such as [2][5][6].

1.2 The Goal of This Thesis

Nowadays, because of the rapid progressive of wireless communication, a variety of mobile computational platforms are emerged, such as mobile phone, PDA, and notebook. The medium between mobile stations and base stations is air. As a result, to protect information of mobile platform becomes an important issue.

TI TMS320c55x' DSP is a high performance CPU. It can handle multimedia, voice compression, voice communication, and highly performance with low power consumption. Therefore, it is a multipurpose CPU that can be introduced to mobile platform.

Our research is to introduce a systemic model to estimate the performance of executing the modular multiplications on TI TMS320c55x' DSP. Moreover, we propose a new modular multiplication algorithm which achieves higher performance than Morita & Yang's Algorithm.

Chapter 2 Implementation of the Basic Arithmetic of Modular Multiplication Algorithm on TI TMS320c55x' DSP

MS320C55x' DSP takes only one clock cycle to execute an addition, a subtraction, or a multiplication instruction. Therefore, we need a new method to evaluation a modular multiplication algorithm for the DSP.

In this chapter, we will briefly introduce the architecture of TI TMS320c55x' DSP in Section 2.1 and Section 2.2. Then, we discuss the implementation issues of some frequent used arithmetic of modular multiplication algorithms in Section 2.3.

2.1 Architecture of TI TMS320c55x' DSP

The architecture of TI TMS320c55x' DSP is shown as Fig. 2-1. The pipeline of TI TMS320c55x' DSP has four stages. There are,

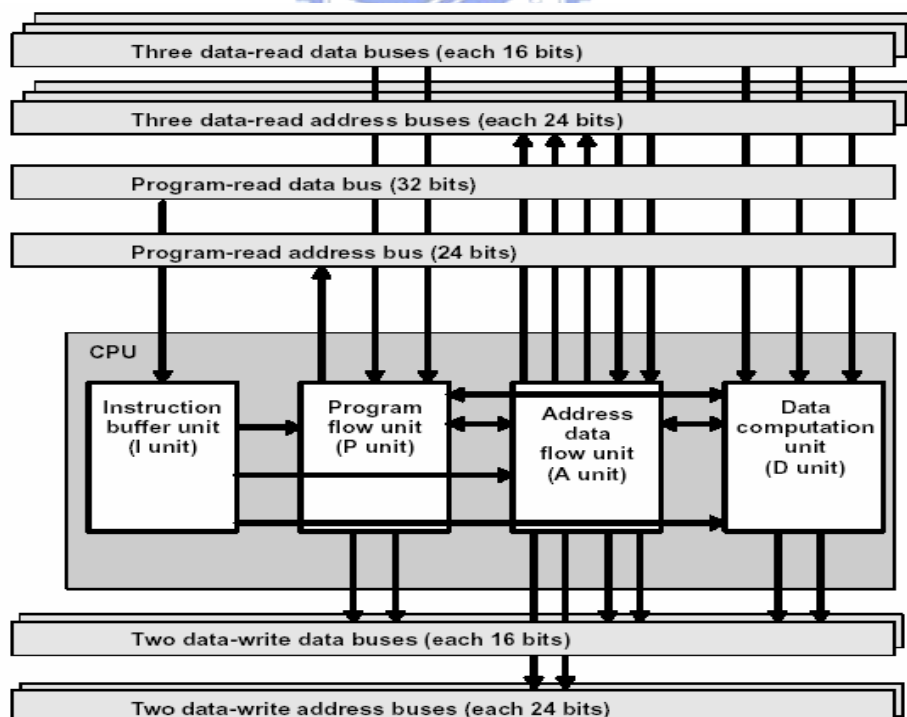


Figure 2-1 The CPU Diagram[13]

1. Instruction buffer unit : Machine codes of the instruction set of TI TMS320c55x' DSP are variety length. As a result, an instruction buffer is needed to smoothen the executing of programs.
2. Program flow unit : This unit handles branch, conditional operation and pipeline protection. Pipeline protection is an automatic mechanism to avoid read-write hazard[7].
3. Address data flow unit : This unit controls data addresses for data reading and writing. TI TMS320c55x' provides three data read buses and two data write buses.
4. Data computation unit : This unit performs arithmetic and logic operations. There are two 17-bit by 17-bit MAC, a 40-bit ALU and a Shifter shown in Fig, 2-2.

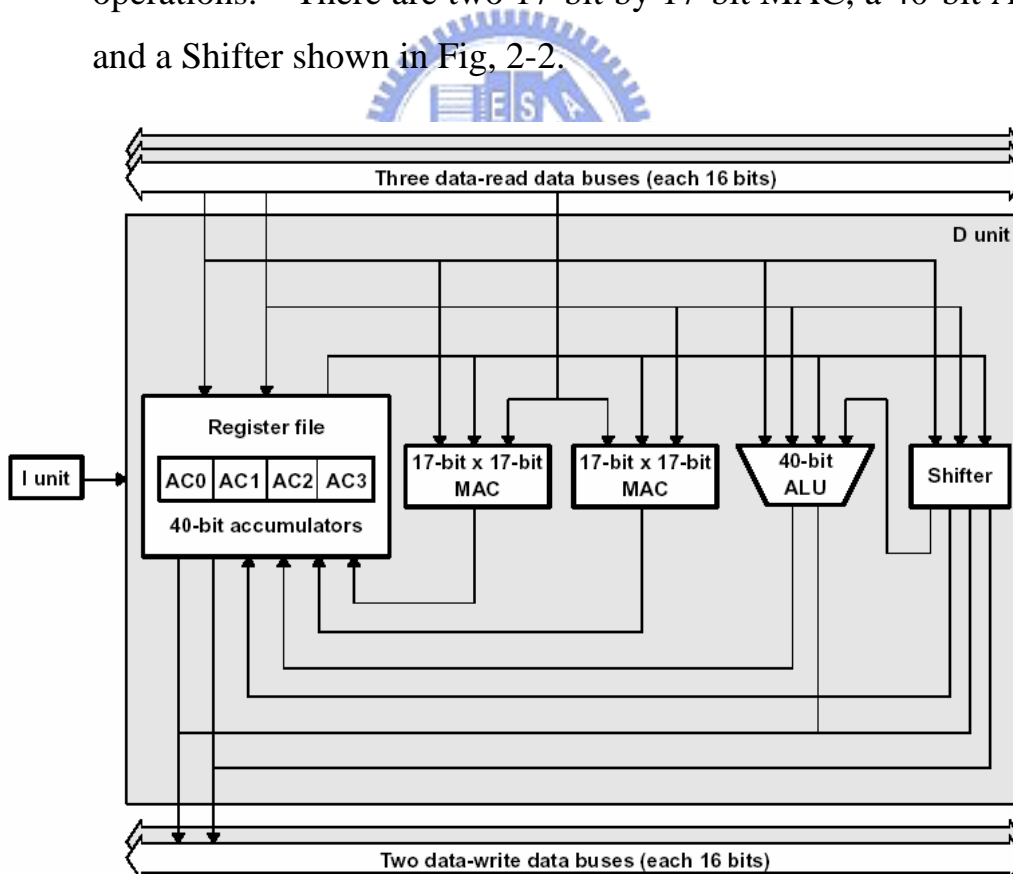


Figure 2-2 The Data Computation Unit Diagram[13]

2.2 Arithmetic Instructions of TI TMS320c55x' DSP

2.2.1 The Registers

1. There are four 40-bit Accumulators AC0, AC1, AC2, AC3. Accumulators are used to execute arithmetic and logic operations. The highest 8-bit of the accumulator is used for sign-extended purpose. As a result, the accumulators can perform 16-bit and 32-bit operations.

2. There are eight 24-bit auxiliary registers XAR0~XAR7. The auxiliary registers can be used to execute arithmetic and logic operations. The other purpose of auxiliary registers is data addressing. Only the lower 16-bit of auxiliary registers, also called as AR0~AR7, can be used to perform arithmetic and logic operations.

3. There are four 16-bit temporary registers T0, T1, T2, and T3. Temporary registers are used for data addressing and to execute arithmetic and logic operations.

The symbols of the instruction set of TI TMS320c55x' DSP are denoted as:

dst, src represent ACn, ARn, and Tn

Smem, Xmem, Ymem means the value of an address

k4, k8, k16 means 4-bit, 8-bit, and 16-bit constants

2.2.2 Left Shift

The left-shift instructions of TI TMS320c55x'DSP with each left-shift instruction taking only one clock cycle are shown in Fig. 2-3. Note that only some of the left-shift instructions can be

performed a 32-bit left-shift.

No.	Syntax	Parallel Enable Bit	Size	Cycles	Pipeline
[1]	SFTL dst, #1	Yes	2	1	X
[2]	SFTL dst, #-1	Yes	2	1	X
[3]	SFTL ACx, Tx[, ACy]	Yes	2	1	X
[4]	SFTL ACx, #SHIFTW[, ACy]	Yes	3	1	X

Figure 2-3 Left Shift Instructions[11]

2.2.3 Addition

The addition instructions of TI TMS320c55x'DSP with each addition instruction taking only one clock cycle are shown in Fig. 2-4. Note that only some of the addition instructions can be performed a 32-bit addition.



No.	Syntax	Parallel Enable Bit	Size	Cycles	Pipeline
[1]	ADD [src.] dst	Yes	2	1	X
[2]	ADD k4, dst	Yes	2	1	X
[3]	ADD K16, [src.] dst	No	4	1	X
[4]	ADD Smem, [src.] dst	No	3	1	X
[5]	ADD ACx << Tx	Yes	2	1	X
[6]	ADD ACx << #SHIFTW, ACy	Yes	3	1	X
[7]	ADD K16 << #16, [ACx.] ACy	No	4	1	X
[8]	ADD K16 << #SHFT, [ACx.] ACy	No	4	1	X
[9]	ADD Smem << Tx, [ACx.] ACy	No	3	1	X
[10]	ADD Smem << #16, [ACx.] ACy	No	3	1	X
[11]	ADD [uns(]Smem[)], CARRY, [ACx.] ACy	No	3	1	X
[12]	ADD [uns(]Smem[)], [ACx.] ACy	No	3	1	X
[13]	ADD [uns(]Smem[)] << #SHIFTW, [ACx.] ACy	No	4	1	X
[14]	ADD dbl(Lmem), [ACx.] ACy	No	3	1	X
[15]	ADD Xmem, Ymem, ACx	No	3	1	X
[16]	ADD K16, Smem	No	4	1	X
[17]	ADDV [ACx.] ACy	Yes	2	1	X

Figure 2-4 Addition Instructions[11]

2.2.4 Subtraction

The subtraction instructions of TI TMS320c55x'DSP with each

subtraction instruction taking only one clock cycle are shown in Fig. 2-5. Note that only some of the subtraction instructions can be performed a 32-bit subtraction.

No.	Syntax	Parallel Enable Bit	Size	Cycles	Pipeline
[1]	SUB [src,] dst	Yes	2	1	X
[2]	SUB k4, dst	Yes	2	1	X
[3]	SUB K16, [src,] dst	No	4	1	X
[4]	SUB Smem, [src,] dst	No	3	1	X
[5]	SUB src, Smem, dst	No	3	1	X
[6]	SUB ACx << Tx, ACy	Yes	2	1	X
[7]	SUB ACx << #SHIFTW, ACy	Yes	3	1	X
[8]	SUB K16 << #16, [ACx,] ACy	No	4	1	X
[9]	SUB (K16 << #SHFT, [ACx,] ACy	No	4	1	X
[10]	SUB Smem << Tx, [ACx,] ACy	No	3	1	X
[11]	SUB Smem << #16, [ACx,] ACy	No	3	1	X
[12]	SUB ACx, Smem << #16, ACy	No	3	1	X
[13]	SUB [uns(]Smem[]), BORROW, [ACx,] ACy	No	3	1	X
[14]	SUB [uns(]Smem[]), [ACx,] ACy	No	3	1	X
[15]	SUB [uns(]Smem[]) << #SHIFTW, [ACx,] ACy	No	4	1	X
[16]	SUB dbl(Lmem), [ACx,] ACy	No	3	1	X
[17]	SUB ACx, dbl(Lmem) ACy	No	3	1	X
[18]	SUB Xmem, Ymem, ACx	No	3	1	X



Figure 2-5 Subtraction Instructions[11]

2.2.5 Multiplication

The multiplication instructions of TI TMS320c55x'DSP with each multiplication instruction taking only one clock cycle are shown in Fig. 2-6 and Fig. 2-7. Note that only 16-bit by 16-bit multiplication can be performed.

No.	Syntax	Parallel			
		Enable Bit	Size	Cycles	Pipeline
[1]	SQR[R] [ACx.] ACy	Yes	2	1	X
[2]	MPY[R] [ACx.] ACy	Yes	2	1	X
[3]	MPY[R] Tx, [ACx.] ACy	Yes	2	1	X
[4]	MPYK[R] K8, [ACx.] ACy	Yes	3	1	X
[5]	MPYK[R] K16, [ACx.] ACy	No	4	1	X
[6]	MPYM[R] [T3 =]Smem, Cmem, ACx	No	3	1	X
[7]	SQRM[R] [T3 =]Smem, ACx	No	3	1	X
[8]	MPYM[R] [T3 =]Smem, [ACx.] ACy	No	3	1	X
[9]	MPYMK[R] [T3 =]Smem, K8, ACx	No	4	1	X
[10]	MPYM[R][40] [T3 =][uns()Xmem[]], [uns()Ymem[]], ACx	No	4	1	X
[11]	MPYM[R][U] [T3 =]Smem, Tx, ACx	No	3	1	X

Figure 2-6 Multiplication Instructions (1) [11]

No.	Syntax	Parallel			
		Enable Bit	Size	Cycles	Pipeline
[1]	SQA[R] [ACx.] ACy	Yes	2	1	X
[2]	MAC[R] ACx, Tx, ACy[, ACy]	Yes	2	1	X
[3]	MAC[R] ACy, Tx, ACx, ACy	Yes	2	1	X
[4]	MACK[R] Tx, K8, [ACx.] ACy	Yes	3	1	X
[5]	MACK[R] Tx, K16, [ACx.] ACy	No	4	1	X
[6]	MACM[R] [T3 =]Smem, Cmem, ACx	No	3	1	X
[7]	MACM[R]Z [T3 =]Smem, Cmem, ACx	No	3	1	X
[8]	SQAM[R] [T3 =]Smem, [ACx.] ACy	No	3	1	X
[9]	MACM[R] [T3 =]Smem, [ACx.] ACy	No	3	1	X
[10]	MACM[R] [T3 =]Smem, Tx, [ACx.] ACy	No	3	1	X
[11]	MACMK[R] [T3 =]Smem, K8, [ACx.] ACy	No	4	1	X
[12]	MACM[R][40] [T3 =][uns()Xmem[]], [uns()Ymem[]], [ACx.] ACy	No	4	1	X
[13]	MACM[R][40] [T3 =][uns()Xmem[]], [uns()Ymem[]], ACx >> #16 [, ACy]	No	4	1	X



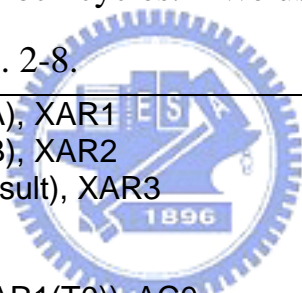
Figure 2-7 Multiplication Instructions (2) [11]

2.3 The Implementation issues of Frequent Used Arithmetic of Modular Multiplication Algorithm on TI TMS320c55x' DSP

The implementations in this chapter are also referred to section 5.1 and section 5.4 in *TMS320C55x DSP Programmer's Guide*[12]. The fixed-point arithmetic is in section 5.1 and the division is in section 5.4.

2.3.1 Pipeline Protection Issue

TI TMS320c55x' DSP provides an automatic pipeline protection mechanism to prevent read-write hazard. However, it also causes the extra clock cycles. We use a 1024-Bit addition as an example shown in Fig. 2-8.



```
01   AMOV #(VarA), XAR1
02   AMOV #(VarB), XAR2
03   AMOV #(AResult), XAR3
04   MOV #62, T0
05   MOV #64, T1
06   MOV40 dbl(*AR1(T0)), AC0
07   ADD dbl(*AR2(T0)), AC0
08   MOV AC0,dbl(*AR3(T1))
09   SFTS AC0, #-32
10   || MOV #30, BRC0
11   RPTBLocal Adder_Loop
12   SUB #2, T0
13   MOV40 dbl(*AR1(T0)), AC1
14   SUB #2, T1
15   ADD #1, T0
16   ADD uns(*AR2(T0)),AC1
17   ADD AC1, AC0
18   SUB #1, T0
19   ADD uns(*AR2(T0))<< #16, AC0
20   MOV AC0, dbl(*AR3(T1))
21 Adder_Loop:
22   SFTS AC0, #-32
23   SUB #1, T1
24   MOV AC0, *AR3(T1)
```

Figure 2-8 1024-Bit addition code with RAW hazard

The above program contains some RAW hazards. The Adder_loop is from the line 12 through the line 22. The loop uses 10 instructions. As we learn from Section 2.2, there must be only 10 clock cycles needed for each loop. However, 20 clock cycles are consumed for each loop. We discover that the extra clock cycles are affected by the lines 12, 13, 15, 16, 18, and 19. Let's analyze the codes of the lines 12 to 13.

```
12    SUB #2, T0
13    MOV40 dbl(*AR1(T0)), AC1
```

The result of T0 at the line 12 will be used by the line 13 immediately. As a result, it causes a RAW hazard[7]. Therefore, the automatic pipeline protection mechanism of TMS320c55x' inserts the four NOPs, i.e. the four bubbles into the pipeline to assure the correct value of T0 will be used by the line 13. The extra NOPs cause the extra clock cycles. At this case, running a 1024-bit addition costs about 650 clock cycles. It's almost 6 times of executing a 1024-bit by 16-bit multiplication.

We rewrite the codes in Fig. 2-9. The adder_loop is from the line 12 through the line 22. The new loop uses 8 instructions and costs 10 clock cycles. To finish, a 1024-bit addition it costs about 350 clock cycles.

01	AMOV #(VarA), XAR1
02	AMOV #(VarB), XAR2
03	AMOV #(AResult), XAR3
04	ADD #2, AR3
05	MOV #62, T0
06	MOV #63, T1
07	MOV40 dbl(*AR1(T0)), AC0
08	ADD dbl(*AR2(T0)), AC0
09	MOV AC0,dbl(*AR3(T0))

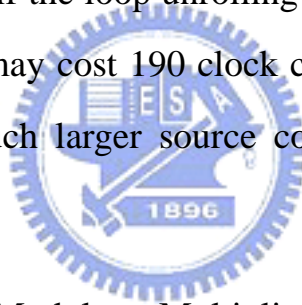
```

10    SFTS AC0, #-32
11    || MOV #30, BRC0
12    RPTBLocal Adder_Loop
13    SUB #2, T0
14    SUB #2, T1
15    MOV40 dbl(*AR1(T0)), AC1
16    ADD uns(*AR2(T1)),AC1
17    ADD AC1, AC0
18    ADD uns(*AR2(T0))<< #16, AC0
19    MOV AC0, dbl(*AR3(T0))
20 Adder_Loop:
21    SFTS AC0, #-32
22    SUB #1, T0
23    MOV AC0, *AR3(T0)

```

Figure 2-9 1024-Bit addition code with less RAW hazard

It spent 11 clock cycles running a 64-bit addition in *TMS320C55x DSP Programmer's Guide*[12] costs only 11 clock cycles. As a result, if the loop unrolling skill[7] is adopted, running a 1024-Bit addition may cost 190 clock cycles only. However, this approach will be much larger source code and make source code difficult to maintain.



2.3.2 Arithmetic of Modular Multiplication Algorithm on TI TMS320c55x' DSP

According to our implementation and taking of pipeline protection, the Clock Cycles Consumption of Frequent Used Arithmetic of Modular Multiplication is shown in Table. 2-1.

Table 2-1 The clock cycles consumption of frequent used arithmetic of modular multiplication algorithm

Operation	Clock cycles	Loop Unrolling
1024 bits shift	110	
1024 bits + 1024 bits	360	190
1024 bits - 1024 bits	300	130
1024 bits × 16 bits	140	
32 bits ÷ 16 bits	50	
1024 bits comparison	15~650	

We use the 1024-bit by 16-bit multiplication as the baseline and weight other arithmetic for evaluation. The weighted clock cycles consumption is shown in Table 2-2. Table 2-2 will be used to evaluate modular multiplication algorithms in chapter 4.

Table 2-2 The weighted clock cycles consumption of frequent used arithmetic of modular multiplication algorithm on TI TMS320c55'

Operation	Real	Unrolling
1024 bits shift	.75	
1024 bits + 1024 bits	2.15	0.95
1024 bits - 1024 bits	2.15	0.95
1024 bits \times 16 bits	1	
32 bits \div 16 bits	0.5	
1024 bits comparison	0~6	



Chapter 3 Background on Iterative Modular Multiplication

In this chapter, we will go through some well know iterative modular multiplication algorithm. Our goal is to re-evaluate those algorithms computational complexity in general purpose CPU and TI TMS320c55x' DSP. We try to make a guideline for choosing a properly modular-multiplication algorithm for TI TMS320c55x' DSP.

3.1 Single Precision Left-to-Right Convention Algorithm

Single precision means “perform a modular multiplication bit by bit”. Left-to- right convention means “perform a modular multiplication from the most significant bit to the less significant bit”.

All the single precision left-to-right convention algorithms are suitable for hardware implementation.

3.1.1 Traditional Algorithm of Modular Multiplication

Algorithm Modula_Multiplication (X, Y, D)

//Input: X, Y and D are the n-bit positive integers

//Output: $P = X \times Y \bmod D$, P is an n-bit positive integer

```
1: P ← 0;
2: for j := n-1 downto 0 do
3:   begin
4:     P ← 2 × P;
5:     if ( P ≥ D ) then
6:       P ← P - D;
7:     if( yj = 1 ) then
8:       begin
9:         P ← P + X;
10:        if ( P ≥ D ) then
11:          P ← P - D;
```

```

12:   end;
13: end;
14: return(P);

```

Time Complexity

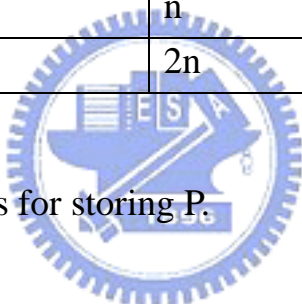
This algorithm is suitable for hardware because left shift costs nothing in hardware. Assume that P has 50% possibility larger than D. As a result, this algorithm needs n times of n-bit shift, n times of n-bit addition, and n times of n-bit subtraction.

Table 3-1 The time complexity of Traditional Algorithm

Operation	Times
n-bit Left Shift	n
n-bit Addition	n
n-bit subtraction	n
n-bit comparison	2n

Storage Complexity

We need $n+1$ bits for storing P.



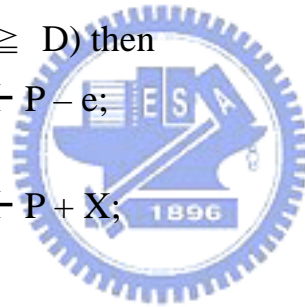
3.1.2 Blakley Algorithm[5]

Algorithm Blakley(X, Y, D)

//Input: X, Y and D are the n-bit positive integers

//Output: $P = X \times Y \bmod D$, P is an n+1-bit positive integer

```
1: P ← 0;
2: e ← D - X;
3: for j := n-1 downto 0 do
4:   begin
5:     if ( P ≥ D ) then
6:       P ← 2 × ( P - D );
7:     else
8:       P ← 2 × P;
9:     if( yj = 1 ) then
10:      if ( P ≥ D ) then
11:        P ← P - e;
12:      else
13:        P ← P + X;
14:   end;
15: if ( P ≥ D ) then
16:   P ← P - D;
17: return(P);
```



Blakley Algorithm introduced a pre-computed integer e. As a result, it costs less than the traditional modular multiplication algorithm.

Time Complexity

Assume that P has 50% possibility larger than D. As a result, this algorithm needs n times of n-bit shift, $1/2n$ times of n-bit addition, and n times of n-bit subtraction.

Table 3-2 The time complexity of Blakely's Algorithm

Operation	Times
n-bit Left Shift	n
n-bit Addition	$1/2 n$
n-bit subtraction	n
n-bit comparison	$2n$

Storage Complexity

The lengths of P and e are $n+1$ bits and n bits, respectively. Note that in process are need $n+1$ bits for the length of P, but we only output n bits for the length of P.



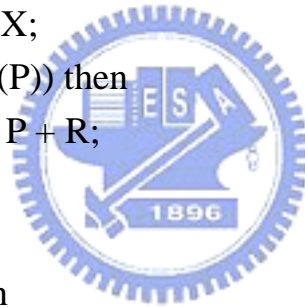
3.1.3 Chiou & Yang's Algorithm[2]

Algorithm Chiou_Yang1(X, Y, D)

//Input: X, Y and D are the n-bit positive integers

//Output: $P = X \times Y \bmod D$, P is an n+1-bit positive integer

```
1:  P ← 0;
2:  R ←  $2^n \bmod D$ ;
3:  for j := n-1 downto 0 do
4:  begin
5:      P ←  $2 \times P$ ;
6:      if (carry(P)) then
7:          P ← P + R;
8:      if(  $y_j = 1$  ) then
9:          begin
10:             P ← P + X;
11:             if (carry(P)) then
12:                 P ← P + R;
13:          end;
14:  end;
15: if (  $P \geq D$  ) then
16:     P ← P - D;
17: return(P);
```



Algorithm Chiou_Yang 2(X, Y, D)

//Input: X, Y and D are the n-bit positive integers

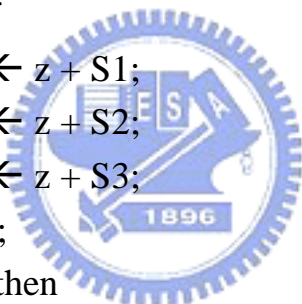
//Output: $P = X \times Y \bmod D$, P is an n-bit positive integer

```
1:  P ← 0;
2:  C ← 0;
3:  R1 ←  $2^n \bmod D$ ;
4:  R2 ←  $2 \times 2^n \bmod D$ ;
5:  R3 ←  $3 \times 2^n \bmod D$ ;
6:  T1 ←  $(2^n + X) \bmod D$ ;
7:  T2 ←  $(2 \times 2^n + X) \bmod D$ ;
```

```

8:  T3 ← (3 × 2n + X) mod D;
9:  for j := n-1 downto 0 do
10: begin
11:   P ← 2 × P;
12:   if (carry(P)) then
13:     c ← c + 1;
14:   if( yj = 1 ) then
15:     case c of
16:       0: z ← z + X;
17:       1: z ← z + T1;
18:       2: z ← z + T2;
19:       3: z ← z + T3;
20:     end case;
21:   else
22:     case c of
23:       1: z ← z + S1;
24:       2: z ← z + S2;
25:       3: z ← z + S3;
26:     end case;
27:   if (carry(P)) then
28:     c ← 2;
29:   else
30:     c ← 0;
31: end;
32: if ( P ≥ D ) then
33:   P ← P - D;
34: return(P);

```



Chiou & Yang's Algorithm 1 uses Carry(P 2ⁿ) and pre-computed table to perform the modular-multiplication. The time complexity for the algorithm is n times “n-bit left shift” and 2/3n times “n-bit addition”. It also requires 2n+1 bits storage space for P and R.

Later Chiou & Yang proposed another algorithm, called Algorithm 2, to improve the performance. The major difference between them is that Algorithm 2 using six-element pre-computed table instead of using one-element in Algorithm 1. In algorithm 2 it requires $7n+2$ bits storage space for P and pre-computed table.

Time and Storage Complexity

The comparison between two Algorithms are shown in table 3-3.

Table 3-3 The time and storage complexity of Chiou & Yang's Algorithm

Operation	Algorithm 1	Algorithm 2
n-bit Left Shift	n	n
n-bit Addition	$\frac{3}{2}n$	n
Space required(bits)	$2n+1$	$7n+2$

3.2 Multi-Precision Left-to-Right Convention Algorithm

Multi-precision means “perform modular multiplication block by block”. Left-to-right convention means “perform modular multiplication from the most significant block to the less significant block”. All the multi-precision algorithms are more suitable for software implementation.

3.2.1 Leong, Tan & Tan’s Algorithm [8]

Algorithm Leong_Tan_Tan(X, Y, D)

// $P = X \times Y \pmod{D}$, where $0 \leq X, Y < D$

// $D = 2^{n-1} + \sum_{i=0}^{n-2} d_i 2^i$, $X = \sum_{i=0}^{n-1} x_i 2^i$

// $Y = \sum_{i=0}^{n-1} y_i 2^i$, $P = \sum_{i=0}^{n-1} p_i 2^i$

// w is the bit amount of a digit

- 1: $e[0] \leftarrow 2^n - D$;
- 2: *for* ($i = 1, i \leq w, i \leftarrow i + 1$)
- 3: **begin**
- 4: $e[i] \leftarrow (e[i - 1] \times 2)$;
- 5: *if* $e[i] \geq D$ *then*
- 6: $e[i] \leftarrow (e[i] + e[0]) \pmod{2^n}$;
- 7: **end**;
- 8: $P \leftarrow 0$;
- 9: *for* ($i = \left\lceil \frac{n}{w} \right\rceil, i > 0, i \leftarrow i - 1$)
- 10: **begin**
- 11: $P \leftarrow X \times y_{i w - 1} y_{i w - 2} \dots y_{i w - w} + P \times 2^w$
- 12: *for* ($j = 0, j \leq w, j \leftarrow j + 1$)
- 13: **begin**
- 14: *if* p_{j+n} *then*

```

15:      begin
16:           $P \leftarrow P - 2^{j+n} + e[j]$ ;
17:          if  $p_n = 1$  then  $P \leftarrow P - 2^n + e[0]$ ;
18:      end;
19:  end;
20: end;
21: if  $P \geq D$  then  $P \leftarrow (P + e[0]) \bmod D$ ;
22: return(P);

```

Leong, Tan & Tan's Algorithm is suitable for software implementing. Even though they claim that the algorithm is a multi-precision algorithm, but it looks like a single precision. The algorithm does not reduce the time complexity significantly.

Time Complexity

$$\text{Let } n' = \left\lceil \frac{n}{w} \right\rceil.$$



The time spent on operating a multiplication is much larger than operating an addition. Then measuring the time complexity, we always neglect the addition part. Then, Leong, Tan & Tan's Algorithm needs only $2n'^2$ multiplications.

For a realistic estimate, the operation of Leong, Tan & Tan's Algorithm list in Table 3-6.

Table 3-4 The time complexity of Leong, Tan & Tan's Algorithm

Operation	Times
n-bit By w-bit Multiplication	n'
n-bit Addition	$(2w+1)n'$
n-bit Subtraction	$2wn'$

Storage Complexity

Leong, Tan & Tan's Algorithm needs $(w^2+2w)n'+w$ bits.

3.2.2 b-ary number system

Let b be a positive integer and $B = \{0, 1, \dots, b-1\}$

We denote Z, Q, R be the set of integers, the set of rational numbers, and the set of real numbers, respectively.

Any number r in Q can be represent by $r = \sum_{i=d}^s r_i b^i$ for each r_i in B .

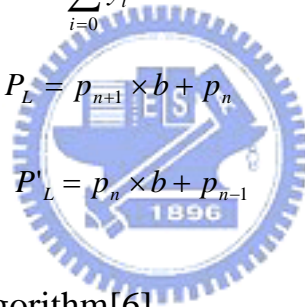
We have $d \geq 0$ if $r \in Z$; $d < 0$ if $r \in Q - Z$. Usually, we take $-\infty < d, s < \infty$.

Let $D = \sum_{i=0}^{n-1} d_i b^i$ and $\frac{b}{2} \leq d_{n-1} \leq b-1$

Let $X = \sum_{i=0}^{n-1} x_i b^i$ and $Y = \sum_{i=0}^{n-1} y_i b^i$

Let $P = \sum_{i=0}^{n+1} p_i b^i$ and $P_L = p_{n+1} \times b + p_n$

Let $P' = \sum_{i=0}^n p_i b^i$ and $P'_L = p_n \times b + p_{n-1}$



3.2.3 Morita & Yang's Algorithm[6]

Algorithm Morita_Yang(X, Y, D)

//Input: X, Y and D are n digits, where $0 \leq X, Y < D$

//Output: $P = X \times Y \text{ mod } D$. P is $n+2$ digits

- 1: $P \leftarrow 0$;
- 2: for $j := n-1$ downto 0 do
- 3: begin
- 4: $P \leftarrow b \times P + X \times y_j$;
- 5: $q \leftarrow \left\lfloor \frac{P_L}{d_{n-1} + 1} \right\rfloor$;
- 6: $P \leftarrow P - b \times q \times D$;
- 7: $q \leftarrow \left\lfloor \frac{P_L}{d_{n-1} + 1} \right\rfloor$;

```

8:     if (q=1) then
9:          $P \leftarrow P - b \times D;$ 
10:    if(q=2) then
11:         $P \leftarrow P - b \times (2 \times D);$ 
12:    if( $P'_L \geq (b - 1) \times (d_{n-1}+1)$ ) then
13:         $P \leftarrow P - (b - 1) \times D;$ 
14:    end;
15:    if ( $P \geq D$ ) then
16:    begin
17:         $q \leftarrow \left\lfloor \frac{P_L}{d_{n-1} + 1} \right\rfloor;$ 
18:         $P \leftarrow P - q \times D;$ 
19:        While ( $P \geq D$ ) do
20:             $P \leftarrow P - D;$ 
21:    end;
22:    return(P);

```

Morita & Yang's Algorithm is suitable for implementing in software for a low storage and computational power CPU. The advantages of the algorithm are:

- i) Using an approximation value of q. It can lower the execution time.
- ii) Using lookahead determinate approach. It can restrict the value in the small range, then it can guess an estimation value shortly.

Actually, the above two condition can also lower the running time and shorter the storage space.

Time Complexity

We should notice that n presents the number of b-ary digit.

The time spent on operating a multiplication is much larger than operating an addition. Then measuring the time complexity, we

always neglects the addition part. Then, Morita & Yang's Algorithm needs $2n^2 + 2n$ multiplications.

For a realistic estimate, the operations of Morita & Yang's Algorithm list in Table 3-5.

Table 3-5 The time complexity of Morita & Yang's Algorithm

Operation	Times
n-bit By w-bit Multiplication	n
2w-bit By w-bit Division	2n
n-bit Addition	n
n-bit Subtraction	2n

Storage Complexity

Morita & Yang's Algorithm needs $n+2$ digits.



Chapter 4 A New Iterative Modular Multiplication Algorithm

We combine both of the algorithms described in the previous chapter to derive a new iterative modular multiplication algorithm. Our algorithm need less pre-computed table space than Leong, Tan & Tan's Algorithm and achieve the same performance like Morita & Yang's Algorithm. Furthermore, our proposed algorithm is better than Morita & Yang's Algorithm.

4.1 The New Modular Multiplication Algorithm

The proposed algorithm is given in Section 4.1.1 and its correctness is shown in Section 4.1.2

4.1.1 Procedure

Algorithm New_Modular_Multiplication(X, Y, D)

//Input: X, Y and D are n digits, where $0 \leq X, Y < D$

//Output: $P = X \times Y \bmod D$. P is n digits.

- 1: $e \leftarrow \text{Pre-computation}(D)$;
- 2: $\hat{P} \leftarrow \text{Approximate}(X, Y, D, e)$;
- 3: *while* ($\hat{P} \geq D$) *do* // The length of \hat{P} is n+2 digits.
- 4: $\hat{P} \leftarrow \hat{P} - D$;
- 5: $P \leftarrow \hat{P}$;
- 6: *return*(P);

Algorithm Pre-computation(D)

//Input: D is n digits

// Output: $e[i] = i \times b^{n+1} \bmod D, 1 \leq i \leq 6$

- 1: $e[1] \leftarrow b^{n+1} \bmod D$;

```

2:  for( $i = 2, i \leq 6, i \leftarrow i + 1$ )
3:  begin
4:       $e[i] \leftarrow (e[i - 1] + e[1]);$ 
5:      if  $e[i] \geq D$  then  $e[i] \leftarrow e[i] - D;$ 
6:  end;
7:  return(e);

```

Algorithm Approximation(X, Y, D, e)

// Input: X, Y and D are n digits, where $0 \leq X, Y < D$

// Output: $P = X \times Y \bmod D$. P is n+2 digits

```

1:   $P \leftarrow 0;$ 
2:  for( $i = n - 1, i > 0, i \leftarrow i - 1$ )
3:  begin
4:       $P \leftarrow X \times y_i + b \times P;$ 
5:       $q \leftarrow \left\lfloor \frac{P'_L}{d_{n-1} + 1} \right\rfloor$ 
6:       $P' \leftarrow P' - q \times D;$ 
7:      if  $p_{n+1} > 0$  then  $P \leftarrow P' + e[p_{n+1}];$ 
8:  end;
9:  return(P);

```

The Algorithm Pre-computation calculates six pre-computed elements, i.e. b^{n+1} , $2 \times b^{n+1}$, $3 \times b^{n+1}$, $4 \times b^{n+1}$, $5 \times b^{n+1}$, and $6 \times b^{n+1} \bmod D$. The pre-computed elements store in e.

The Algorithm Approximation uses approximation method and pre-computed table to calculate $P = X \times Y \bmod D$. The

approximation method uses $\left\lfloor \frac{P'_L}{d_{n-1} + 1} \right\rfloor$ instead of $\left\lfloor \frac{P}{D} \right\rfloor$ to calculate

quotient q. The approximation method reduces great amount of computational time. Certainly, the approximate quotient is not

equal to the real quotient. The difference in value between the approximate value and the real value of quotient causes that P may be greater than D . We use the pre-computed table to limit P to $6D$.

However, the output of Algorithm Approximation, i.e. \hat{P} , may still be larger than D . The line 3 and 4 of Algorithm New_Modular_Multiplication truncate the \hat{P} to $\hat{P} < D$.

4.1.2 Correctness

$$\text{Let } \varepsilon = \left\lfloor \frac{P'_L}{d_{n-1} + 1} \right\rfloor$$

$$\text{Let } t = \frac{\sum_{i=0}^{n-2} p_i b^i}{b^{n-1}}, \text{ we have } 0 \leq t < 1.$$

$$\text{we have } P' = (P'_L + t) \times b^{n-1},$$

$$\text{and } (P'_L + 1) \times b^{n-1} > P' > P'_L \times b^{n-1} \text{ from } 0 \leq t < 1$$

$$\text{Let } a = \frac{\sum_{i=0}^{n-2} d_i b^i}{b^{n-1}}, \text{ we have } 0 \leq a < 1.$$

$$\text{It implies } D = (d_{n-1} + a) \times b^{n-1} \text{ and } (d_{n-1} + 1) \times b^{n-1} > D \geq d_{n-1} \times b^{n-1}$$

For $b, c, M \in \mathbb{Z}$

$$b \pmod{M} + c \equiv (b + c) \pmod{M}$$

$$\text{We consider } P \pmod{D} = (p_{n+1} \times b^{n+1} + P') \pmod{D}$$

$$\equiv p_{n+1} \times b^{n+1} \pmod{D} + P' \pmod{D}$$

We compute $p_{n+1} \pmod{D}$ by using table look-up and

compute $P' \bmod D$ by using approximation method.

$$\text{Theorem 1 : } 0 \leq P' - \left\lfloor \frac{P'_L}{d_{n-1} + 1} \right\rfloor \times D < 5D$$

Proof:

$$\text{From } P' \geq P'_L \times b^{n-1} \quad \text{and} \quad (d_{n-1} + 1) \times b^{n-1} > D$$

$$\text{We have } \frac{P'}{D} \geq \frac{P'_L \times b^{n-1}}{(d_{n-1} + 1) \times b^{n-1}} \geq \left\lfloor \frac{P'_L \times b^{n-1}}{(d_{n-1} + 1) \times b^{n-1}} \right\rfloor = \varepsilon$$

$$\text{It implies } P' - \varepsilon \times D \geq 0 \dots\dots\dots (1)$$

$$\text{We have } (P'_L + 1) \times b^{n-1} > D$$

$$\Rightarrow \frac{(P'_L + 1)}{d_{n-1} + a} > \frac{P'}{D} \quad \text{from } D = (d_{n-1} + a) \times b^{n-1}$$

$$\Rightarrow \frac{(P'_L + 1)}{d_{n-1} + a} - \left\lfloor \frac{P'_L}{d_{n-1} + 1} \right\rfloor > \frac{P'}{D} - \left\lfloor \frac{P'_L}{d_{n-1} + 1} \right\rfloor \dots\dots\dots (2)$$

$$\text{Max}_a \left\{ \frac{(P'_L + 1)}{d_{n-1} + a} - \frac{P'_L}{d_{n-1} + 1} \right\} = \frac{P'_L + 1}{d_{n-1} + a} - \frac{P'_L}{d_{n-1} + 1}$$

$$\text{Max}_{\frac{b}{2} \leq d_{n-1} < b-1; 0 \leq P'_L < b^2-1} \left\{ \frac{P'_L + 1}{d_{n-1} + a} - \frac{P'_L}{d_{n-1} + 1} \right\} = 4 \times \frac{b^2 + \frac{b}{2}}{b^2 + 2b} = 4 - \frac{6}{b+2}$$

$$\text{We know } 0 < \frac{6}{b+2} < 1, \quad \text{if } b \geq 4$$

$$\text{Then } 3 < 4 - \frac{6}{b+2} < 4$$

$$\text{Thus } 4 > \frac{(P'_L + 1)}{d_{n-1} + a} - \frac{P'_L}{d_{n-1} + 1}$$

$$\Rightarrow 5 > \frac{(P'_L + 1)}{d_{n-1} + a} - \left\lfloor \frac{P'_L}{d_{n-1} + 1} \right\rfloor \dots\dots\dots (3)$$

From (2) and (3)

$$\text{We have } \frac{P'}{D} - \left\lfloor \frac{P'_L}{d_{n-1} + 1} \right\rfloor < 5$$

Thus, $P' - \left\lfloor \frac{P'_L}{d_{n-1} + 1} \right\rfloor \times D < 5D$ (4)

From (1) and (4)

We have $0 \leq P' - \left\lfloor \frac{P'_L}{d_{n-1} + 1} \right\rfloor \times D < 5D$

In Algorithm Approximation we need six elements for our pre-computed table.



4.2 Implement Issues

The proposed algorithm is easy to be implemented. However, there are two conditions must to be handle. First, the quotient q is some times greater than one digit in the Algorithm Approximation. Then, there are no other instructions to compute $b^{n+1} \bmod D$ directly in the Algorithm Pre-computation. We discuss the two issues at Section 4.2.1 and Section 4.2.2, respectively.

4.2.1 Calculate $P' \leftarrow P' - q \times D$ with two-digit q

Lemma 1: Let $q_1 \times b + q_0 = \left\lfloor \frac{P'_L}{(d_{n-1} + 1)} \right\rfloor$. Then we get $q_1 \leq 1$.

Proof:

$$q_1 \times b + q_0 = \left\lfloor \frac{P'_L}{(d_{n-1} + 1)} \right\rfloor$$

$$\Rightarrow q_1 = \left\lfloor \frac{p_n}{(d_{n-1} + 1)} \right\rfloor \dots \dots \dots (5)$$

We knew $P'_L = p_n \times b + p_{n-1}$, $0 \leq P'_L \leq b^2 - 1$ and $\frac{b}{2} \leq d_{n-1} \leq b - 1$

It implies $0 \leq p_n \leq b - 1$

$$\text{Max}_{0 \leq p_n \leq b-1; \frac{b}{2} \leq d_{n-1} \leq b-1} \left\{ \frac{p_n}{d_{n-1} + 1} \right\} = 2 \times \frac{b-1}{b+2} \dots \dots \dots (6)$$

$$0 \leq \frac{b-1}{b+2} < 1, \text{ for all } b \geq 1$$

From (5), we have q_1 is an integer.

$$\text{From (6), we have } 2 \times \frac{b-1}{b+2} \geq q_1$$

Thus, $q_1 \leq 1$

Base on the LEMMA, we modify the Algorithm Approximation to Algorithm Approximation_M.

Algorithm Approximation_M(X, Y, D, e)

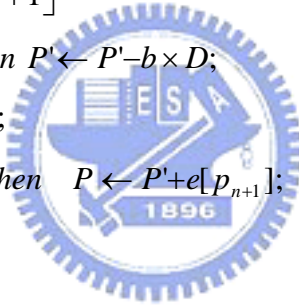
// Input: X, Y and D are n digits, where $0 \leq X, Y < D$

// Output: P= X \times Y mod D. P is n+2 digits

```

1:  P  $\leftarrow$  0;
2:  for(i = n - 1, i > 0, i  $\leftarrow$  i - 1)
3:  begin
4:    P  $\leftarrow$  X  $\times$  yi + b  $\times$  P;
5:    q1 || q0  $\leftarrow$   $\left\lfloor \frac{P'_L}{d_{n-1} + 1} \right\rfloor$ 
6:    if q1 > 0 then P'  $\leftarrow$  P' - b  $\times$  D;
7:    P'  $\leftarrow$  P' - q0  $\times$  D;
8:    if pn+1 > 0 then P  $\leftarrow$  P' + e[pn+1];
9:  end;
10: return(P);

```



4.2.2 Calculating $b^{n+1} \bmod D$

There are no other instruction to compute $b^{n+1} \bmod D$ directly. We use the approximation method to calculate $b^{n+1} \bmod D$.

Algorithm e1(D)

//Input:D is n digits

// Output: e[1] = $b^{n+1} \bmod D$

```

1: e[1]  $\leftarrow$   $b^{n+1}$ ;

```



```

2:  $q \leftarrow \left\lfloor \frac{e[1]_{n+1} \| e[1]_n \| e[1]_{n-1}}{d_{n-1} + 1} \right\rfloor$ ;
3:  $e[1] \leftarrow e[1] - q \times D$ ;
4: while ( $e[1] \geq D$ ) do
5:    $e[1] \leftarrow e[1] - D$ ;
6: return( $e[1]$ );

```

The line 2 of Algorithm e1 executes a three-digit dividend divided by one-digit divisor. We know that the first digit of $e[1]$ is 1 and $\frac{b}{2} \leq d_{n-1} \leq b-1$. Therefore, the expression $e[1]-b \times D$ operation will make the first digit of $e[1]$ equal to 0. It means that only two-digit dividend divided by one-digit divisor is needed. Then, referred to Lemma 1, we modify the Algorithm e1 to Algorithm e1_M.



```

Algorithm e1_M(D)
//Input:D is n digits
// Output:  $e[1] = b^{n+1} \bmod D$ 

```

```

1:  $e[1] \leftarrow b^{n+1}$ ;
2:  $e[1] \leftarrow e[1] - b \times D$ ;
3:  $q_1 \| q_0 \leftarrow \left\lfloor \frac{e[1]_n \| e[1]_{n-1}}{d_{n-1} + 1} \right\rfloor$ ;
4: if  $q_1 > 0$  then  $e[1] \leftarrow e[1] - b \times D$ ;
5:  $e[1] \leftarrow e[1] - q_0 \times D$ ;
6: while ( $e[1] \geq D$ ) do
7:    $e[1] \leftarrow e[1] - D$ ;
8: return( $e[1]$ );

```

Using Algorithm e1_M, we can modify Algorithm Pre-computation to Algorithm Pre-computation_M.

Algorithm Pre-computation_M(D)

//Input: D is n digits

// Output: $e[i] = i \times b^{n+1} \pmod{D}$, $1 \leq i \leq 6$

```
1: e[1] ← e1_M(D);
2: for(i = 2, i ≤ 6, i ← i + 1)
3: begin
4:     e[i] ← (e[i - 1] + e[1]);
5:     if e[i] ≥ D then e[i] ← e[i] - D;
6: end;
7: return(e);
```

4.2.3 Complete Algorithm

Consider implement issues, we modify the Algorithm New_Modular_Multiplication to New_Modular_Multiplication_M.

Algorithm New_Modular_Multiplication_M(X, Y, D)

//Input: X, Y and D are n digits, where $0 \leq X, Y < D$

//Output: $P = X \times Y \pmod{D}$. P is n digits

```
1: e ← Pre-computation_M(D);
2:  $\hat{P} \leftarrow$  Approximate_M(X, Y, D, e);
3: while ( $\hat{P} \geq D$ ) do // The length of  $\hat{P}$  is n+2 digits.
4:      $\hat{P} \leftarrow \hat{P} - D$ ;
5: P ←  $\hat{P}$ ;
6: return(P);
```

4.3 Special Case

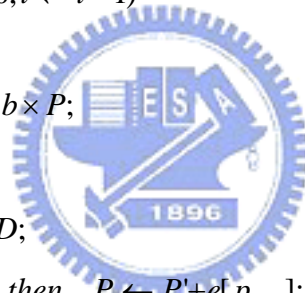
If $d_{n-1} = b-1$ is used, $d_{n-1} + 1 = b$. As a result, no more division operation is needed and no more q_i exist, nether. In this condition, the Algorithm Approximation_M become Algorithm Approximation_SPC.

Algorithm Approximation_SPC(X, Y, D, e)

// Input: X, Y and D are n digits, where $0 \leq X, Y < D$

// Output: P= X \times Y mod D. P is n+2 digits

```
1:  P  $\leftarrow$  0;
2:  for(i = n-1, i > 0, i  $\leftarrow$  i-1)
3:  begin
4:    P  $\leftarrow$  X  $\times$  yi + b  $\times$  P;
5:    q  $\leftarrow$  pn;
6:    P'  $\leftarrow$  P' - q0  $\times$  D;
7:    if pn+1 > 0 then P  $\leftarrow$  P' + e[pn+1];
8:  end;
9:  return(P);
```



4.4 Complexity

We discuss our new algorithm in general purpose CPU and the TI MS3200 55x' DSP, respectively.

4.4.1 General Purpose CPU

Usually executing a multiplication is much longer than executing a addition. Then, we only consider time complexity of running the multiplications. The other assumption is that a 2-digit dividend divided by 1-digit divisor need about the same clock cycles as a multiplication operation. The loop repeats n times and each loop performs $2n+1$ times multiplication. Totally, the new algorithm will take $2n^2+n$ times of multiplication. In the special case, our algorithm needs only $2n^2$ times of multiplication.

4.4.2 TI MS3200C 55x' DSP

Based on Fig. 4-9 the modified of Proposed Algorithm, the total computational effort is list in Table 4-1.

Table 4-1 The computational effort of Proposed Algorithm

Operation	Worst Case	In Average
n-digit subtraction	$2n$	$3/2n$
2- digit by 1- digit division	n	n
n- digit by 1- digit multiplication	$2n$	$2n$
n- digit addition	$2n$	$3/2n$

4.4.3 Storage Complexity

The new proposed algorithm required $7n+2$ digits of b-ary numbers.

4.6 Compare with Other Algorithms

Base on table 2-2, and the estimated clock cycles consumption results of chapter 3, we derive table 4-2. In this table, we assume n is 1024, w is 16.

Table 4-2 Comparing with other iterative algorithm by weighted clock cycles consumption

Modular Multiplication Algorithm	ideal	real
Traditional Algorithm	8.650 n	11.050n
Blakely's Algorithm	8.175 n	9.975n
Chiou & Yang's Algorithm 1	2.175 n	3.975n
Chiou & Yang's Algorithm 2	1.700 n	2.900n
Morita & Yang's Algorithm	0.366 n	0.590n
Leong, Tan & Tan's Algorithm	3.921 n	8.797n
New Proposed Algorithm (best case)	0.273 n	0.694n
New Proposed Algorithm (average case)	0.334 n	0.559n
New Proposed Algorithm (worst case)	0.394 n	0.461n

In average, our proposed algorithm will take the same time as Morita & Yang's Algorithm. In special case, new proposed algorithm will be better than Morita & Yang's Algorithm.

Chapter 5 Conclusion

In this thesis, we study the architecture of TI TMS320c55x' DSP and its instruction set. We show that only consider multiplication as the main computational complexity is not enough. And also, we derive a evaluation model to estimate the computational complexity of modular modulation algorithm for TI TMS320c55x' DSP.

Moreover, we proposed a new iterative modular multiplication algorithm. The new proposed algorithm can achieve the same performance as Morita & Yang's Algorithm in average, and have better performance in certain cases.

Future work

Our modular multiplication evaluation model is based on TI TMS320c55x' DSP, now. In the feature we could extend the evaluation model to other new proposed TI DSP family.



REFERENCES

- [1] C.W. Chiou, "Parallel Implementation of the RSA Public-Key Cryptosystem" Intern. J. Computer Math., vol. 48, pp135-155,1993.
- [2] C.W. Chiou & T.C. Yang, "Iterative modular multiplication algorithm without magnitude comparison" Electronics Letters, vol. 30, no.24, pp2017-2018, 1994.
- [3] D.E. Knuth, "The Art of Computer Programming" Addition-Welsey, 2nd, 1981.
- [4] "FIPS PUB 186-2: Digital Signature Standard (DSS)", NIST, 2000.
- [5] G.R Blakley, "A computer Algorithm for Calculating the Product AB Modulo M" IEEE Transaction On Computer, vol. c-32. no. 5, pp497-500, 1983.
- [6] H. Morita & C.H. Yang, "A Modular-Multiplication Algorithm Using Lookahead Determination" IEICE Trans. Fundamentals, vol.E76-A, no.1, 1993.
- [7] J.L. Hennessy & D.A. Patterson "Computer Architecture—A Quantitative Approach" Morgan-Kaufmann, 2nd, 1995.
- [8] P.C. Leong, E.C. Tan & P.C. Tan, "An Iterative Modular Multiplication Algorithm", Computers and Mathematics with Applications vol.44, pp175-180, 2002.
- [9] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signature and public-key cryptosystems", Commun. of ACM, vol.21, no.2, pp.120-126, 1978.
- [10] S.Y. Yan, "Number Theory for Computing", Springer, 2000.
- [11] "SPRU374 TMS320C55x DSP Mnemonic Instruction Set Reference Guide", Texas Instruments, 2001.
- [12] "SPRU376 TMS320C55x DSP Programmer's Guide", Texas Instruments, 2000.

- [13] “SPRU393 TMS320C55x Technical Overview”, Texas Instruments, 2000.
- [14] T. ElGamal, “A public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms” IEEE Trans. On Info. Theory, vol.31, pp469-472, 1985.
- [15] W. Diffie & M. Hellman, “New Directions in Cryptography” IEEE Trans. On Info. Theory, vol.22, no.6, pp637-647,1976.

