HYBRID ANALYSIS OF TMVP FOR MODULAR POLYNOMIAL MULTIPLICATION IN CRYPTOGRAPHY

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF APPLIED MATHEMATICS
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

GİRAY EFE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
CRYPTOGRAPHY

FEBRUARY 2022

Approval of the thesis:

HYBRID ANALYSIS OF TMVP FOR MODULAR POLYNOMIAL MULTIPLICATION IN CRYPTOGRAPHY

submitted by **GİRAY EFE** in partial fulfillment of the requirements for the degree of **Master of Science in Cryptography Department, Middle East Technical University** by,

Prof. Dr. A. Sevtap Selçuk Kestel _____
Dean, Graduate School of **Applied Mathematics**

Prof. Dr. Ferruh Özbudak _____
Head of Department, **Cryptography**

Assoc. Prof. Dr. Murat Cenk _____
Supervisor, **Cryptography, METU, IAM**

Examining Committee Members:

Prof. Dr. Ferruh Özbudak
Department of Mathematics, METU _____

Assoc. Prof. Dr. Murat Cenk
Cryptography, IAM, METU _____

Assoc. Prof. Dr. Fatih Sulak
Department of Mathematics, Atılım University _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: GİRAY EFE

Signature :

ABSTRACT

HYBRID ANALYSIS OF TMVP FOR MODULAR POLYNOMIAL MULTIPLICATION IN CRYPTOGRAPHY

EFE, Giray

M.S., Department of Cryptography

Supervisor : Assoc. Prof. Dr. Murat Cenk

February 2022, 119 pages

Polynomial multiplication on the quotient ring $\mathbb{Z}[x]/\langle x^n \pm 1\rangle$ is one of the most fundamental, general-purpose operations frequently used in cryptographic algorithms. Therefore, a possible improvement over a multiplication algorithm directly affects the performance of algorithms used in a cryptographic application. Well-known multiplication algorithms such as School-book, Karatsuba, and Toom-Cook are dominant choices against NTT in small and ordinary input sizes. On the other hand, how these approaches are implemented under the quotient ring of polynomials, $\mathbb{Z}[x]/\langle x^n \pm 1\rangle$, matters. Instead of applying the reduction procedure as the final stage, using Toeplitz Matrix Product (i.e., TMVP) is a clever way to realize the modular multiplication more efficiently. Furthermore, the hybrid use of these algorithms yields more efficient results than the static choice of any single algorithm. For this purpose, we derive and analyze various constructions of multiplication and share the best possible sequences under different circumstances, and show that TMVP is a decent choice instead of classical modular polynomial multiplication approaches in cryptographic applications.

Keywords: Cryptography, Modular Polynomial Multiplication, Hybrid Multiplication,TMVP, 2-way Split Multiplication, 3-way Split Multiplication, Karatsuba Multiplication, Toom-Cook Multiplication, Arithmetic Complexity Analysis, Delay Complexity Analysis, Cost Metrics, Hybrid Cost Calculation, Most Efficient Hybrid Cost Results

ÖZ

KRİPTOGRAFİDE KULLANILAN MODÜLER POLİNOM ÇARPMALARI İÇİN TMVP'NİN HİBRİT ANALİZİ

EFE, Giray
Yüksek Lisans, Kriptografi Bölümü
Tez Yöneticisi : Doç. Dr. Murat Cenk

Şubat 2022, 119 sayfa

$\mathbb{Z}[x]/\langle x^n \pm 1\rangle$ bölüm halkasındaki polinom çarpımı, kriptografik algoritmalarda sıklıkla kullanılan en temel, genel amaçlı işlemlerden biridir. Bu nedenle, bir çarpma algoritmasında olası bir iyileştirme, bir kriptografik uygulamada kullanılan algoritmaların performansını doğrudan etkiler. Schoolbook, Karatsuba ve Toom-Cook gibi iyi bilinen çarpma algoritmaları, küçük ve sıradan girdi boyutlarında NTT'ye karşı baskın seçeneklerdir. Öte yandan, bu yaklaşımların polinomların bölüm halkası $\mathbb{Z}[x]/\langle x^n \pm 1\rangle$ altında nasıl uygulandığı önemlidir. Son aşama olarak indirgeme prosedürünü uygulamak yerine Toeplitz Matris Vektör Çarpmasını (yani TMVÇ) kullanmak, modüler çarpmayı daha verimli bir şekilde gerçekleştirmenin akıllıca bir yoludur. Ayrıca, bu algoritmaların hibrit kullanımı, herhangi bir tek algoritmanın statik seçiminden daha verimli sonuçlar verir. Bu amaçla, çeşitli çarpma yapılarını türetiyor, analiz ediyor ve farklı koşullar altında mümkün olan en iyi dizileri paylaşıyoruz. Kriptografik uygulamalarda TMVÇ'nin klasik modüler polinom çarpma yaklaşımlarına göre daha iyi bir seçim olduğunu gösteriyoruz.

Anahtar Kelimeler: Kriptografi, Modüler Polinom Çarpması, Hibrid Çarpma, TMVÇ, 2 Yönlü Bölünmüş Çarpma, 3 Yönlü Bölünmüş Çarpma, Karatsuba Çarpması, Toom-Cook Çarpması, Aritmetik Karmaşıklık Analizi, Gecikme Karmaşıklık Analizi, Tutar Metriği, Hibrid Maliyet Hesaplaması

To My Family

ACKNOWLEDGMENTS

I would like to express my great appreciation to my thesis supervisor Assoc. Prof. Dr. Murat Cenk for his patient guidance, enthusiastic encouragement, and valuable advice during the development and preparation of this thesis. His willingness to give his time and share his experiences has brightened my path. I would also like to thank jury members of the thesis, Prof. Dr. Ferruh Özbudak, and Assoc. Prof Dr. Fatih Sulak for their meaningful inputs.

I would like to express my gratitude to TÜBİTAK for their financial support during my studies. I am also grateful to my first company Meteksan Savunma and my colleagues for their permanent support. Finally, my sincere thanks go to ASELSAN and the Cryptographic Software and Digital Design department for their warm welcome and understanding during the thesis.

I wish to extend my special thanks to Irem Keskinkurt Paksoy for her invaluable support, endless motivation, and excellent guidance.

Last but not least, I would like to thank my family members Sami Efe, Semra Efe, and Seval Nihan Açıkgöz, who have always been in my heart.

TABLE OF CONTENTS

APPENDICES

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

TMVP	Toeplitz Matrix Vector Product
MPM	Modular Polynomial Multiplication
$M(n)$	Arithmetic Complexity
$D(n)$	Delay Complexity
M	Number of Multiplication Blocks
A_s	Number of Single Word Addition Blocks
A_d	Number of Double Word Addition Blocks
D_m	Delay (Or Deepness) Introduced by a Multiplication
D_s	Delay (Or Deepness) Introduced by a Single Word Addition
D_d	Delay (Or Deepness) Introduced by a Double Word Addition
TMVP-SB-ITR	Toeplitz Matrix Vector Product, Schoolbook (Naive) Approach by Iteratively
TMVP-SB-REC	Toeplitz Matrix Vector Product, Schoolbook (Naive) Approach by Recursion
TMVP-STD-2	Toeplitz Matrix Vector Product, Standard 2 Way Split Approach
TMVP-STD-3	Toeplitz Matrix Vector Product, Standard 3 Way Split Approach with 6 Multiplication
TMVP-3	Toeplitz Matrix Vector Product, 3 Way Split Approach with 5 Multiplication
TMVP-4	Toeplitz Matrix Vector Product, 4 Way Split Approach with 7 Multiplication
POLY-SB-ITR	Schoolbook Polynomial Multiplication by Iteratively
POLY-SB-REC	Schoolbook Polynomial Multiplication by Recursion
POLY-SB-2	Schoolbook 2 Way Polynomial Multiplication
POLY-KB-2	Karatsuba 2 Way Polynomial Multiplication
POLY-RKB-2	Refined Karatsuba 2 Way Polynomial Multiplication
POLY-OKB-2	Optimized Karatsuba 2 Way Polynomial Multiplication
POLY-SB-3	Schoolbook 3 Way Polynomial Multiplication
POLY-KB-3	Karatsuba 3 Way Polynomial Multiplication
POLY-OKB-2	Optimized Karatsuba 3 Way Polynomial Multiplication

CHAPTER 1

INTRODUCTION

Multiplication is one of the most fundamental operations used in cryptographic algorithms. Primarily, public key cryptography consists of various multiplication operations with huge sizes. Therefore, it is a well-known fact that an improvement over a multiplication algorithm directly affects the performance of algorithms used in public-key cryptography.

Recently, many post-quantum cryptographic algorithms for various purposes such as confidentiality, authentication, integrity, and so on continue to be developed. Independent of their development purpose, efficiency is a key factor, and there are many performance and security trade-offs. In the case of efficient implementation, choice of target cost metric (time, space, or both of them) and environment specifications (constrained MCU architecture, reconfigurable hardware) become crucial. Therefore, one should understand and optimize the way of multiplication implemented in a target cryptographic application depending on the needs accordingly.

In the presence of any multiplication, the type of its operands and how to multiply them differs. For example, a multiplication might only consist of integer operands, or it may have a matrix to be processed with a vector whose elements do not even need to be an integer, but another algebraic structure like a polynomial. Despite this extensive variation, some of these multiplication structures are similar and might be convertible to each other in some circumstances. Despite the similarity in representation, the realization of these multiplications might differ and provide some advantages in certain circumstances. TMVP is one of those alternatives in the case of modular polynomial multiplication.

1.1 Previous Work

Before sharing the studies regarding TMVP, let us first share the ordinary modular polynomial multiplication over finite fields. Multipliers proposed in [16], [29], [20], [22], [30] before 2000's were having quadratic space complexity, and those later studied in [13], [33], [28], [9], [21], [32] proceed an improvement in space cost having sub-quadratic complexity class before 2010. Regardless of their efficiency, the modular reduction step is at the end of their multi-

plication. Towards the end of 2010, Fan and Hasan proposed a new architecture in [12], what they call "Fan-Hasan Multiplier"; exploiting the equivalence of TMVP with modular polynomial multiplication embedding the reduction stage throughout the overall process. They concluded that this approach has various advantages in space and delay complexity compared to those ordinary modular polynomial multiplications. Hasan et al. improved the space complexity of their architecture by another approach called 'Block Recombination' in [14]. After 2010, there have been many further studies of TMVP. Hasan and Christophe in [2] worked through the generic splitting of TMVP over binary fields and analyzed the efficiency of this approach in terms of space and delay complexities. Paksoy and Cenk expanded this splitting approach by introducing new constructions for efficient TMVP. They exploited Toom-Cook multiplication approach [10] to get more efficient construction of generic split methodologies in TMVP. They derived arithmetic (space) complexities of their constructions and demonstrated the applicability by a MCU implementation on ARM-Cortex M4, which advances the performance for a Saber implementation given in [18]. Taşkın and Cenk examine the suitable choice of prime fields to boost the performance of TMVP constructions in [31]. They utilize TMVP as a building block in finite field multiplication to provide an efficient construction basis for Elliptic Curve Cryptography.

All these studies above encouraged us to investigate further possible opportunities of TMVP for cryptographic purposes. Alternatively, there are different approaches in the literature examining the equivalence of polynomial multiplication over $\mathbb{Z}[x]/\langle x^n \pm 1\rangle$ in terms of cyclic convolutions. The reader may also want to refer to [4] to gain a deeper understanding of this equivalence. Furthermore, Akleylek et al. exploit this equivalence to realize more efficient results in MCU implementation in [1]. On the other hand, TMVP does cover not only cyclic convolutions but also promise functional representation, easing in finding out new split constructions.

1.2 Our contribution

In this thesis, we show that TMVP is a good candidate for reconfigurable hardwares, and it has decent advantages over ordinary modular polynomial multiplications. For this purpose, we first demonstrate arithmetic and delay complexities of several TMVP constructions compiled in Paksoy's work in [24]. Furthermore, we extend the work of Ilter [17] by also sharing delay complexity results in addition to existing arithmetic costs of standard polynomial multiplication methodologies. Throughout the derivations, we use stick diagrams for arithmetic complexity and hardware schemes, containing addition and multiplication logic blocks, for delay complexity to clarify the derivation process. Finally, we introduce our hybrid analysis approach and the algorithm we used to provide a large set of hybrid multiplication cost results of both TMVP and MPM for various input sizes under different cost metric weight configurations.

1.3 Notation, Assumptions and Domain of Computation

We work with the field of integers $\mathbb{Z}$. We use very similar notion used in [7]. We classify our elements in terms of size as either single or double word, regardless of the type (either MPM or TMVP) they belong to. We assume that the default size is a single word for inputs of a multiplication algorithm.

$M(n)$ refers to the number of arithmetic operation during the multiplication of either two polynomial of degree $n-1$ or TMVP with size $(n \times n) \times (n \times 1)$. We decompose the total number of arithmetic operations into two groups: number of multiplications $M_{\bigotimes}(n)$, and additions $M_{\bigoplus}(n)$. We further express the cost metrics: number of multiplication, single and double word addition. (M, A_s, A_d respectively)

$D(n)$ represents the delay complexity for a hardware implementation of TMVP or MPM construction. We define cost metrics as 'Single Word Addition', 'Double Word Addition' and 'Multiplication' represented by D_s, D_d and D_m. We also use $D_{\bigotimes}(n)$ and $D_{\bigoplus}(n)$ as the decomposition of overall delay complexity to represent the delay contribution from additions and multiplication, respectively.

We ignore scalar multiplication and shifting operations throughout the paper in order to provide a fair comparison with previous works we have focused ([24], [17]).

1.4 Organization of Thesis

In Chapter 2, we recall Toeplitz matrices and how it is related to modular polynomial multiplication. In addition to these, we clarify our cost metrics and types of complexities we have focused on more detailedly. We share its similarities and differences with those previously done in the literature. In Chapter 3, we derive arithmetic and delay complexities of various TMVP constructions in [24] by providing hardware schemes as the combination of addition and multiplication logic blocks. Chapter 4 further derives arithmetic and delay complexities of polynomial multiplications compiled in [17]. We introduce our hybrid analysis approach in Chapter 5. Finally, we compare and analyze results in Chapter 6 and conclude the study in Chapter 7 with a brief summary of these results by encouraging readers with further possible studies, extensions, or improvements over this thesis.

CHAPTER 2

PRELIMINARY

This chapter provides comprehensive background information for the thesis. We start the chapter by introducing Toeplitz Matrices. We continue with expressing the computational complexity definitions and cost metrics that we use throughout the thesis. Finally, we show the equivalence between polynomial multiplication over $\mathbb{Z}[x]/\langle x^n \pm 1\rangle$ and TMVP.

2.1 Brief Introduction to Toeplitz Matrices

Definition 2.1.1. *A toeplitz matrix A of size $n \times n$ is a matrix whose elements $A_{i,j}$ with row index i and column index j satisfies $A_{i,j} = a_{i-j}$ where a_{i-j} s are arbitrary constant values satisfying $1 - n \leq i - j \leq n - 1$*

Above condition can be depicted into more convenient view as follows.

$$A_{n\times n} = \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & & a_{1,n} \\ a_{2,1} & a_{2,2} & & & \vdots \\ \vdots & & \ddots & & \\ & & & & a_{n-1,n} \\ a_{n,1} & \dots & & a_{n,n-1} & a_{n,n} \end{pmatrix} = \begin{pmatrix} a_0 & a_{-1} & a_{-2} & \dots & a_{1-n} \\ a_1 & a_0 & a_{-1} & & \vdots \\ a_2 & a_1 & a_0 & \ddots & a_{-2} \\ \vdots & & \ddots & a_0 & a_{-1} \\ a_{n-1} & \dots & a_2 & a_1 & a_0 \end{pmatrix}$$

Following mathematical inferences can trivially be extracted from the definition of toeplitz matrices.

Lemma 2.1.1. *Addition of two Toeplitz Matrix yields again a Toeplitz Matrix.*

Proof. Let A and B are be Toeplitz Matrices of size $n \times n$. From the definition of the toeplitz matrices, elements of such matrices with row index i and column index j satisfy the following

conditions.

$$A_{i,j} = a_{i-j}$$
$$B_{i,j} = b_{i-j}$$

Elements of matrices, $a_k = a_{i-j}$ and $b_k = b_{i-j}$ are arbitrary constants, and for the size $n \times n$, $1 - n \leq k = i - j \leq n - 1$. Now, without loss of generality, we can define another set of elements s_k such that

$$s_k = a_k + b_k$$

When we insert $1 - n \leq k = i - j \leq n - 1$ instead of k in the above equation, we get

$$s_{i-j} = a_{i-j} + b_{i-j} \qquad \text{where } 1 - n \leq i - j \leq n - 1$$

From the above equation, we see that the set of elements s_k constructs another toeplitz matrix.

□

Lemma 2.1.2. *Scalar product of a Toeplitz Matrix yields again a Toeplitz Matrix.*

Proof. Let A be Toeplitz matrix with elements $A_{i,j} = a_{i-j}$ where the index of elements a_k satisfy $1 - n \leq k = i - j \leq n - 1$. Without loss of generality we can define another set of constants $b_k = ca_k$. So if we make a scalar product of matrix $A_{i,j}$ by a arbitrary scalar constant c , we would get such elements $b_{i-j} = ca_{i-j}$ where $1 - n \leq k = i - j \leq n - 1$; and eventually these elements b_{i-j} with $1 - n \leq k = i - j \leq n - 1$ constructs another Toeplitz matrix of size $n \times n$. □

Lemma 2.1.3. *A Toeplitz Matrix of size $n \times n$ can be represented by $2n - 1$ elements.*

Proof. A toeplitz matrix A of size $n \times n$ can be decomposed into the summation of upper triangle matrix A_{upper}, lower triangle matrix A_{lower}, and its diagonal matrix $A_{diagonal}$ as follows

$$A_{n\times n} = A_{diagonal} + A_{upper} + A_{lower}$$

$$A_{n\times n} = \begin{pmatrix} a_0 & & & \\ & a_0 & & \mathbf{0} \\ & & a_0 & \\ & \mathbf{0} & & a_0 \end{pmatrix} + \begin{pmatrix} 0 & a_1' & a_2' & \dots & a_{n-1}' \\ & 0 & a_1' & & \vdots \\ & & 0 & \ddots & a_2' \\ & \mathbf{0} & & 0 & a_1' \\ & & & & 0 \end{pmatrix} + \begin{pmatrix} 0 & & & & \\ a_1 & 0 & & & \mathbf{0} \\ a_2 & a_1 & 0 & & \\ \vdots & & \ddots & 0 & \\ a_{n-1} & \dots & a_2 & a_1 & 0 \end{pmatrix}$$

$$A_{n\times n} = \begin{pmatrix} a_0 & a_1^{'} & a_2^{'} & \dots & a_{n-1}^{'} \\ a_1 & a_0 & a_1^{'} & & \vdots \\ a_2 & a_1 & a_0 & \ddots & a_2^{'} \\ \vdots & & \ddots & a_0 & a_1^{'} \\ a_{n-1} & \dots & a_2 & a_1 & a_0 \end{pmatrix}$$

Let us denote the number of non-zero elements in a matrix A as $cu(A)$. In the above composition, a diagonal matrix contains only one non-zero element a_0 that is $cu(A_{diagonal}) = 1$ meaning that the Toeplitz Matrix contains only one distinct element at its diagonal. Furthermore, it is trivial to see that $cu(A_{upper}) = cu(A_{lower}) = n - 1$ indicating the number of non-zero elements in the triangular matrices are equal and number of such elements are $n-1$. Finally, the total number of such distinct elements in the Toeplitz Matrix equals to the number of total elements in each of its decomposed matrices.

$$cu(A_{n\times n}) = cu(A_{diagonal}) + cu(A_{upper}) + cu(A_{lower}) \tag{2.1}$$

$$cu(A_{n\times n}) = 1 + (n-1) + (n-1) = 2n - 1 \tag{2.2}$$

□

Proposition 2.1. *Addition of two Toeplitz Matrix of size $n \times n$ requires $2n - 1$ addition at most.*

Proof. From Lemma 2.1.1, we know that addition of two Toeplitz matrices yields a toeplitz matrices and the number of elements used in the addition of two matrices are $2n - 1$ from Lemma 2.1.3. Combining these facts leads us $2n-1$ addition operation at most in summation of two toeplitz matrices. □

2.2 Computational Complexity Definitions

In computer science, the measure of an algorithm's efficiency is referred to as the algorithm's complexity. On the other hand, the target domain or efficiency type may vary. The most common classification of the complexity is to state the time and space as the efficiency factors of the algorithms. These are called 'time complexity' and 'space complexity'. On the other hand, it is possible to define new types of complexities to express different types of costs for various systems and circumstances. For example, one could express the efficiency of power consumption of an algorithm in a target machine as power complexity.

In the case of reconfigurable hardware implementation, Hasan and others in [15] state the 'space' complexity as the number of gates used in the digital circuitry during various multiplication architectures. They use the term 'delay' and 'time' complexity interchangeably as the time required to propagate the signals to find out the result. In their convention, the number of primitive operation 'XOR' or 'AND' is directly proportional to the number of gates

used in hardware implementation, which they call 'space' complexity since it is a sort of resource used in the scheme.

We follow the same previously mentioned convention with a slight difference. We re-factor the name of 'space' as 'arithmetic' in order to express the number of arithmetic operations used. We prefer using the word 'delay' rather than 'time' to express the cost of propagation time for the hardware implementation. Our motivation regarding these name choices is to prevent confusion. One possible confusion would be when the target implementation environment is resource-constrained software rather than hardware. Such implementations without any parallel computation, all instructions are executed sequentially, waiting for each other to be executed, completely different from hardware implementation where one can achieve purely parallel computation over the logic gates. Therefore, the number of arithmetic operations in such software implementation starts to refer to the time elapsed (i.e., time complexity). Besides, space complexity implies the memory used through the software implementation. For all these reasons, we call arithmetic complexity as the number of arithmetic operations regardless of the environment, whether it is reconfigurable hardware or software. On the other hand, we define and refer to delay complexity as the time or deepness required to propagate the signals in case of hardware implementation.

2.2.1 Arithmetic Complexity

We represent $M(n)$ as the total number of arithmetic operations during the multiplication of either two polynomial of degree $n-1$ or TMVP with size $(n \times n) \times (n \times 1)$.

We define 'Arithmetic Cost Metric' as the original number of operation count for any arithmetic operation during the multiplication. Whereas various arithmetic cost metrics might affect the total number of arithmetic operations, the thesis restricts cost metrics to multiplication, single and double word addition. In other words, these cost metrics are our building blocks to count the total number of operations during the multiplication. Abbreviation for these metrics are given below in Table 2.1

Table 2.1: Arithmetic Cost Metric and Corresponding Abbreviations Table

Arithmetic Cost Metric	**Abbreviation**
Original Number of Multiplication	M
Original Number of Single Addition	A_s
Original Number of Double Addition	A_d

We define weight for each cost metric. Denoting any cost metric c_i, and the corresponding weight w_i; Equation 2.3 evaluates the total number of operations $M(n)$ in below.

$$M(n) = \sum_i c_i w_i \tag{2.3}$$

Weightings for cost metrics enrich the comparison facility for diverse multiplication platforms that might perform differently under the same operation configuration. On the other hand, Chapters 3 and 4 regarding complexity derivation does not concern with weights of cost metrics. The inclusion of weightings will be within the hybrid analysis results computed through the cost functions. (See Chapters 5, 6)

Cost function $M(n)$ used to represent the overall arithmetic complexity during a Toeplitz matrix-vector product (i.e., TMVP) can further be decomposed into multiplication cost $M_{\bigotimes}(n)$, and addition cost $M_{\bigoplus}(n)$ for the sake of ease in the analysis. These are costs to express the number of multiplication and additions during a TMVP of size $n \times n$ respectively. Initial conditions are determined when the size $n = 1$, which is nothing but a scalar multiplication, for which we assume a single multiplication cost $1M$ and zero addition cost $0A$ (neither single nor double addition). Below decomposition and initial conditions are going to be valid for all types of TMVP approaches unless otherwise is stated.

$$M(n) = M_{\bigotimes}(n) + M_{\bigoplus}(n)$$

subjected to the initial conditions

$$\begin{aligned} M_{\bigotimes}(1) &= 1M \\ M_{\bigoplus}(1) &= 0A \\ M(1) = M_{\bigotimes}(1) + M_{\bigoplus}(1) \\ M(1) &= 1M + 0A \\ M(1) &= 1M \end{aligned}$$

2.2.2 Delay Complexity

We define delay complexity $D(n)$ as the time required to compute a multiplication operation in a hardware implementation, at which limitless number logic gates or units might operate simultaneously in ideal. There is certainly an upper limit for the number of parallel logic gates that can feed by a driving current of the circuit. In our thesis, we are not concerned with this limit and assume that there is no limit.

From the digital electronic point of view, our delay complexity (or cost) is a measure of the sum of 'gate delay' (or 'propagation delay') and 'transmission delay'. Gate (or propagation) delay is defined as the elapsed time of propagation of a change in the input signal through the output of the logic gate. On the other hand, transmission delay measures the required time to transmit the information in the wire.

In our study, we restrict our cost metrics to multiplication, single and double word addition. In

other words, these cost metrics are our building blocks to count the total number of operations during the multiplication. Abbreviations for these metrics are given below in Table 2.2.

Table 2.2: Delay Cost Metrics and Corresponding Abbreviations Table

Delay Cost Metric	**Abbreviation**
Delay of Multiplication	D_m
Delay of Single Addition	D_s
Delay of Double Addition	D_d

Hasan and others state their cost metrics as 'Delay of XOR' and 'Delay of AND' in their work [15] since their target domain is binary fields. For further clarification, let us demonstrate their cost metrics and abbreviation throughout their work.

Table 2.3: Delay Cost Metric used in [15] and Corresponding Abbreviations Table

Delay Cost Metric	**Abbreviation**
Delay of AND	D_A
Delay of XOR	D_X

Now, let us demonstrate the logic diagram of a full-adder circuit and calculate the delay complexity. Nevertheless, before starting, let us tell about what is a full-adder circuit first. A full-adder circuit is a digital circuit with three inputs and two outputs. Two of the inputs are bits to be added, and one remaining input is carry bit which is considered a sign about another previous adding operation result. This bit is also added to the sum of inputs. One signal corresponds to the summation (XOR) result in the output, and the other remaining output pin belongs to the carry bit. A carry bit is set to 1 if the summation result exceeds 2.

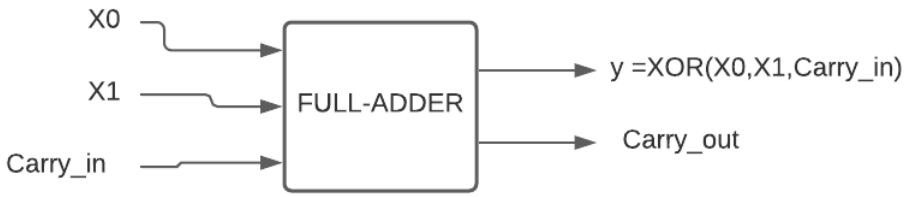

Figure 2.1: Full Adder Block

Table 2.4: Delay Cost Metric used in [15] and Corresponding Abbreviations Table

X0	X1	Carry_in	y	Carry_out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

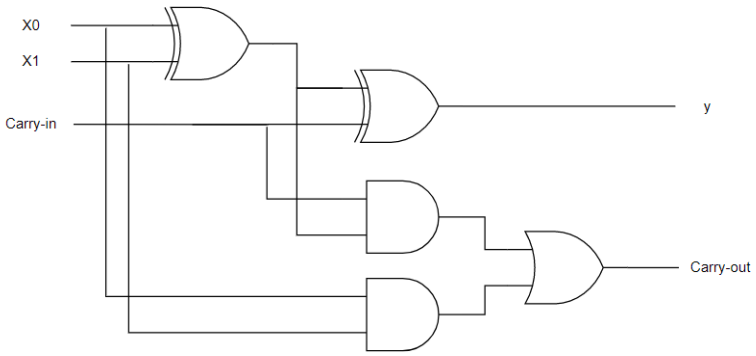

Figure 2.2: Full Adder Block

Before analyzing the delay, let us first extend the aforementioned Table 2.2 in order to able to express 'OR' gate as well.

Table 2.5: Extended Version of Delay Cost Metric used in [15]

Delay Cost Metric	Abbreviation
Delay of AND	D_A
Delay of XOR	D_X
Delay of OR	D_O

From Figure 2.2, it is clear that there are two signal path for each output 'y' and 'Carry_out'. Let us denote delays for each path as $D_y(n)$ and $D_{Carry\_out}(n)$. There are two simple facts used to calculate delay complexities.

Lemma 2.2.1. *Delays are summed up if the corresponding gates are cascaded to each other.*

Proof. In cascaded logic gates, stable or steady-state output of the leading gate requires 'waiting' the preceding gate's output. This is why the delays are summed up. □

Lemma 2.2.2. *Let there be multiple inputs affecting the result of interested output signal. Final delay cost is the maximum one among those different signal paths.*

Proof. Every input signal affecting the output might encounter a different type and number of gates. The stable or steady state result of the output signal is achieved after the longest lasting signal reached to the output. This is why the maximum delay cost is the final delay cost of the logic scheme. □

2 XOR gates to compute y. Recall Lemma 2.2.1.

$$D_y(n) = 2D_X$$

1 XOR - 1 AND - 1 OR to compute carry out. Recall Lemma 2.2.1.

$$D_{Carry\_out}(n) = D_X + D_A + D_O$$

Overall delay should be the maximum of possible delay. Recall Lemma 2.2.2.

$$D_{FULL-ADDER}(n) = max\{D_y(n), D_{Carry\_out}(n)\} \tag{2.4}$$

Without loss of generality, we can assume the cost metrics are quite close and equivalent to each other.

$$D_X \approx D_A \approx D_O$$
$$D_X + D_A + D_O > 2D_X \implies D_{Carry\_out}(n) > D_y(n)$$
$$D_{FULL-ADDER}(n) = max\{D_y(n), D_{Carry\_out}(n)\} = D_{Carry\_out}(n)$$
$$D_{FULL-ADDER}(n) = D_X + D_A + D_O \tag{2.5}$$

In our case, elementary logic units are not primitive gates, but rather a combination of them representing either 'integer multiplication' or 'integer addition' since our interest is a field of integer rather than a binary field. In other words, our cost metrics are our black boxes from which we built up a multiplication architecture. Therefore, being able to assign different weights for our delay cost metrics 'integer multiplication' and 'integer addition' is important. In below, block diagrams of 'integer addition' and 'integer multiplication' are given.

From Figure 2.3, each full adder block waits for the previous carry bit, and propagates it through the final 'carry_out'. From the delay cost convention used in [15], this logic scheme would yield 16 times the original delay of the full adder block, which was given in 2.5. On the other hand, we will use this logic block of 'word adder' as our smallest primitive block and define its delay cost as a single addition delay D_s.

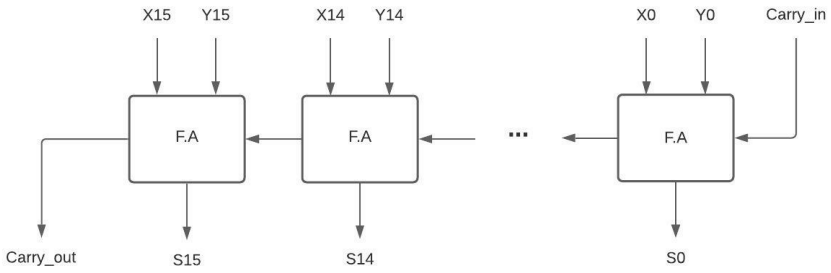

Figure 2.3: 16-bit-adder

In the case of integer multiplication implementation in hardware, there are several schemes in the literature realizing the well-known multiplication methods such as 'Karatsuba' etc. For example, Arish and Sharma in [3] implements a matrix multiplication in FPGA by utilizing the Strassen algorithm. Regardless of those implementations, we rather choose arbitrarily one of them as a black-box and call it a multiplication block whose delay is denoted by D_m.

2.3 TMVP and Modular Polynomial Multiplication Relation

Polynomial multiplication over quotient rings is a fundamental operation used quite frequently in lattice-based systems in PQC. When the quotient ring is of the form $\mathbb{Z}[x]/\langle x^n \pm 1\rangle$, the polynomial multiplication problem can be converted to TMVP. To illustrate, consider the multiplication of two polynomials $a(x)$ and $b(x)$ over $\mathbb{Z}[x]/\langle x^3 - 1\rangle$. This can be simply done by replacing $x^3 = 1$ after the regular multiplication, as is demonstrated below.

$$a(x) = a_0 + a_1 x + a_2 x^2$$
$$b(x) = b_0 + b_1 x + b_2 x^2$$
$$c(x) = c_0 + c_1 x + c_2 x^2 \equiv a(x)b(x) \mod x^3 - 1$$
$$a(x)b(x) = a_0 b_0 + x(a_1 b_0 + a_0 b_1) + x^2(a_0 b_2 + a_1 b_1 + a_2 b_0) + x^3(a_1 b_2 + a_2 b_1) + x^4(a_2 b_2)$$
$$c(x) = a_0 b_0 + a_1 b_2 + a_2 b_1 + x(a_1 b_0 + a_0 b_1 + a_2 b_2) + x^2(a_0 b_2 + a_1 b_1 + a_2 b_0)$$
$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} a_0 & a_2 & a_1 \\ a_1 & a_0 & a_2 \\ a_2 & a_1 & a_0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix}$$

We can generalize this equivalence as below.

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-1} \end{pmatrix} = \begin{pmatrix} a_0 & a_{n-1} & a_{n-2} & \dots & a_1 \\ a_1 & a_0 & a_{n-1} & & \vdots \\ a_2 & a_1 & a_0 & \ddots & a_{n-2} \\ \vdots & & \ddots & a_0 & a_{n-1} \\ a_{n-1} & \dots & a_2 & a_1 & a_0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{n-1} \end{pmatrix} \equiv a(x)b(x) \mod x^n - 1$$

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-1} \end{pmatrix} = \begin{pmatrix} a_0 & -a_{n-1} & -a_{n-2} & \dots & -a_1 \\ a_1 & a_0 & -a_{n-1} & & \vdots \\ a_2 & a_1 & a_0 & \ddots & -a_{n-2} \\ \vdots & & \ddots & a_0 & -a_{n-1} \\ a_{n-1} & \dots & a_2 & a_1 & a_0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{n-1} \end{pmatrix} \equiv a(x)b(x) \mod x^n + 1$$

CHAPTER 3

THE DERIVATION OF ARITHMETIC AND DELAY COMPLEXITIES FOR TMVP CONSTRUCTIONS

This chapter derives arithmetic and delay complexities of various algorithms presented in [15, 24, 34] for calculating the product P_n of an $n \times n$ Toeplitz matrix T_n by an $n \times 1$ vector V_n in the form of recursive cost functions. Without loss of generality, while computing the delay complexity of a k-split formula, we assume that the dimension n is a multiple of k.

3.1 Schoolbook Iterative (TMVP-SB-ITR)

Schoolbook approach is the naive one in which a direct multiplication of row vectors of the square matrix and the column vector is multiplied. Let us denote the multiplication as follows

$$P_n = T_n V_n$$

3.1.1 Arithmetic Complexity

$$\text{Dot Product of Each Row, Column Vector} \rightarrow nM + (n-1)A_d$$

Since there are n such row for matrix size of nxn, total arithmetic cost can be expressed as

$$M(n) = n^2 M + n(n-1)A_d \tag{3.1}$$

3.1.2 Delay Complexity

In Figure 3.1, construction starts with a multiplication stage, each of which are done simultaneously. Each orange dashed line in the first stage separates distinct rows of the Toeplitz matrix to be multiplied with a vector. Overall delay contributed by multiplications is simply

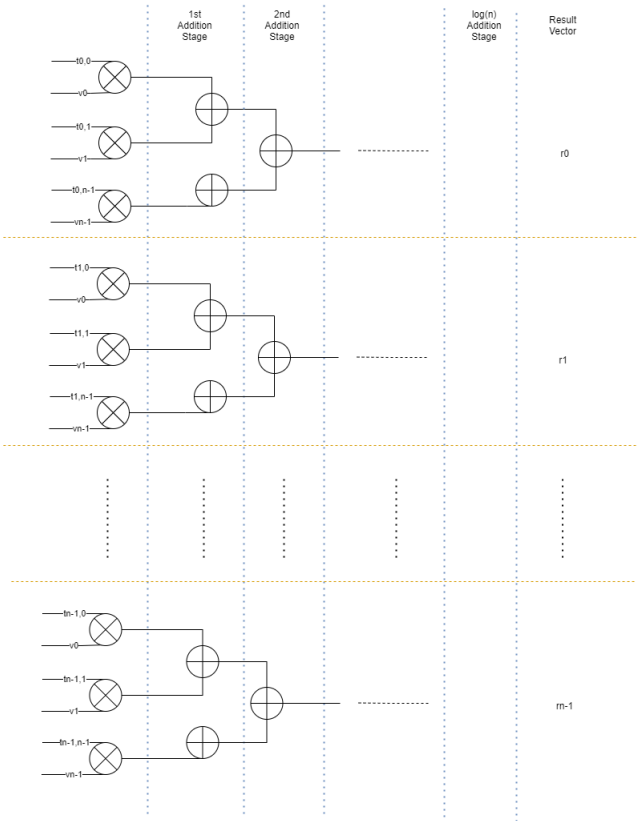

Figure 3.1: TMVP-SB Hardware Implementation Block Scheme - No Recursion

equal to the delay of single multiplication.

$$D_{\otimes}(n) = D_m \tag{3.2}$$

Addition contribution to the delay is again trivial. There are n double elements in the result vector; each of which consist of n elements to be summed, and none of which do not need to wait each other. Those elements can be added in $\lceil log_2(n) \rceil$ cascaded double addition blocks if they are grouped into 2 at each step of addition chain. The scenario is illustrated in Figure 3.1.

$$D_{\oplus}(n) = \lceil log_2(n) \rceil D_d \tag{3.3}$$

Final, delay complexity could be explained as follow.

$$\begin{aligned} D(n) &= D_{\otimes}(n) + D_{\oplus}(n) \\ D(n) &= D_m + \lceil log_2(n) \rceil D_d \end{aligned} \tag{3.4}$$

3.2 Schoolbook Recursive (TMVP-SB-REC)

Same naive approach given in previous Section 3.1 can be implemented as recursively. We start with splitting the matrix into 4 equal sub-Toeplitz matrices as follow. Construction is given as below in Equation 3.5.

$$P_n = T_n V_n = \begin{pmatrix} T_1 & T_0 \\ T_2 & T_1 \end{pmatrix} \begin{pmatrix} V_0 \\ V_1 \end{pmatrix} = \begin{pmatrix} T_1 V_0 + T_0 V_1 \\ T_2 V_0 + T_1 V_1 \end{pmatrix} \tag{3.5}$$

3.2.1 Arithmetic Complexity

As it is seen from Equation 3.5, there are 4 TMVP instances to be called recursively, each of which has the half size of the original problem. Result of these instances are a vector of size $n/2$ including double word elements, and there are two such vector to be summed in every recursion step. This leads to the following arithmetic cost function.

$$M(n) = 4M(n/2) + nA_d \qquad \text{where } M(1) = 1M \tag{3.6}$$

Note that the solution of Equation 3.6 is the same as given in Equation 3.1. This briefly shows us that arithmetic complexity is independent of the way it is implemented. Algorithm can be either recursive or iteratively handled, both of which gives the same arithmetic complexity result.

3.2.2 Delay Complexity

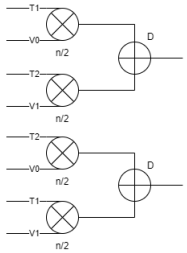

Figure 3.2: TMVP-SB Hardware Implementation Block Scheme - Recursive Approach

$$\begin{aligned} D_{\bigoplus}(n) &= D_{\bigoplus}(n/2) + D_d & D_{\bigoplus}(1) &= 0 \\ D_{\bigotimes}(n) &= D_{\bigotimes}(n/2) & D_{\bigotimes}(1) &= D_m \\ D(n) &= D_{\bigotimes}(n) + D_{\bigoplus}(n) & D(1) &= D_m \\ D(n) &= D(n/2) + D_d & D(1) &= D_m = 0 \end{aligned} \tag{3.7}$$

Note that ‘Schoolbook’ (or naive) approaches did not exploit any property of being TMVP. Complexity of these constructions are equivalent to any matrix vector product. On the other hand, following constructions during the chapter exploits the advantage of being TMVP, which brings us several advantages in terms of arithmetic and delay complexities.

3.3 Two way formula (TMVP-STD-2)

Two way method proposes a division of the matrix into 4 sub-matrices of equal size to be proceed a recursive TMVP. Product substitutions are taken from [24], and demonstrated as follows.

$$\begin{pmatrix} P_1 \\ P_2 \end{pmatrix} = \begin{pmatrix} T_1 & T_0 \\ T_2 & T_1 \end{pmatrix} \begin{pmatrix} V_0 \\ V_1 \end{pmatrix} = \begin{pmatrix} M_1 + M_2 \\ M_1 - M_3 \end{pmatrix} \tag{3.8}$$

If we define 3 multiplication of size $n/2 \times n/2$ as in [24] such that

$$M_1 = T_0(V_0 + V_1) \tag{3.9}$$
$$M_2 = (T_0 - T_1)V_1 \tag{3.10}$$
$$M_3 = (T_1 - T_2)V_0 \tag{3.11}$$

3.3.1 Arithmetic Complexity

Two way method proposes a division of the matrix into 4 sub-matrices of equal size to be proceed a recursive TMVP. Product substitutions are taken from [24], and demonstrated as follows.

$$M_{nxn} = \left(\begin{array}{c|c} (T_1)_{(n/2)\times(n/2)} & (T_0)_{(n/2)\times(n/2)} \\ \hline (T_2)_{(n/2)\times(n/2)} & (T_1)_{(n/2)\times(n/2)} \end{array} \right)$$

$$V_{nx1} = \left(\begin{array}{c} (V_0)_{(n/2)\times 1} \\ \hline (V_1)_{(n/2)\times 1} \end{array} \right)$$

$$P_{n\times 1} = \left(\begin{array}{c} (P_1)_{(n/2)\times(n/2)} \\ \hline (P_2)_{(n/2)\times(n/2)} \end{array} \right)$$

$$P = MV$$

If we define 3 multiplication of size $n/2 \times n/2$ as in [24] such that

$$M_1 = T_0(V_0 + V_1) \tag{3.12}$$
$$M_2 = (T_0 - T_1)V_1 \tag{3.13}$$
$$M_3 = (T_1 - T_2)V_0 \tag{3.14}$$

Then, we can rewrite the multiplication result as

$$P = \begin{pmatrix} P_1 \\ P_2 \end{pmatrix} = \begin{pmatrix} M_1 + M_2 \\ M_1 - M_3 \end{pmatrix} \tag{3.15}$$

Coming back to our substitutions given in Equations from 3.12 to 3.14. There seems to be 3 matrix vector multiplication, in each of which a toeplitz matrix is present. (See Lemma 2.1.2.). Recursive TMVP subjected to the multiplication cost can be represented as follow.

$$M_{\bigotimes}(n) = 3M_{\bigotimes}(n/2) \qquad \text{where } M_{\bigotimes}(1) = 1 \tag{3.16}$$

Now, we need to count additions. From 3.13 to 3.14, there are two matrix addition, and in 3.12 we have a vector addition. Also there are two more vector addition, which is seen in 3.15. Before we start to write addition costs, for the sake of simplicity; let us divide all addition costs as follows.

$$\begin{aligned} M_{\bigoplus,v1}(n) &\leftarrow \text{addition cost of } V_0 + V_1 \\ M_{\bigoplus,v2}(n) &\leftarrow \text{addition cost of } M_1 + M_2 \\ M_{\bigoplus,v3}(n) &\leftarrow \text{addition cost of } M_1 - M_3 \\ M_{\bigoplus,t1}(n) &\leftarrow \text{addition cost of } T_0 - T_1 \\ M_{\bigoplus,t2}(n) &\leftarrow \text{addition cost of } T_1 - T_2 \end{aligned}$$

In addition to the above cost definitions, we need to define an extra cost which represents the same elements already exist both in the term $T_0 - T_1$ and $T_1 - T_2$. These same elements enables us further reducing the number of total addition during the TMVP. So let us represent this common cost coming from those same elements as $M_{\bigoplus,tCommon}(n)$

$$M_{\bigoplus,tCommon}(n) \leftarrow \text{common addition cost of } T_0 - T_1 \text{ and } T_1 - T_2$$

Finally, we can write resultant addition cost in terms of predefined partial addition costs as below.

$$\begin{aligned} M_{\bigoplus}(n) = M_{\bigoplus,v1}(n) + M_{\bigoplus,v2}(n) + M_{\bigoplus,v3}(n) + \\ M_{\bigoplus,t1}(n) + M_{\bigoplus,t2}(n) - M_{\bigoplus,tCommon}(n) \end{aligned} \tag{3.17}$$

For vector of sizes $n/2 \times 1$, addition cost is trivial and $n/2$.

$$M_{\bigoplus,v1}(n) = M_{\bigoplus,v2}(n) = M_{\bigoplus,v3}(n) = \frac{n}{2} \tag{3.18}$$

From Lemma 2.1.3, there are $2n-1$ elements identifying a toeplitz matrix of size $n \times n$. Hence for out partitioned matrices T_0, T_1, T_2 with sizes of $n/2 \times n/2$, number of such elements are $2(n/2) - 1 = n - 1$. Hence, we can write $M_{\bigoplus,t1}(n)$ and $M_{\bigoplus,t2}(n)$ as follow.

$$M_{\bigoplus,t1}(n) = M_{\bigoplus,t2}(n) = n - 1 \tag{3.19}$$

However, it is not trivial to compute $M_{\bigoplus,tCommon}(n)$. To make things simple and for the sake of visualization, let us expand $T_0 - T_1$ and $T_1 - T_2$ when the partitioned toeplitz matrix $T_{partitioned}$ has a size of $4x4$ without loss of generality.

$$T_{partitioned} = \left(\begin{array}{c|c} (T_1) & (T_0) \\ \hline (T_2) & (T_1) \end{array}\right) = \begin{pmatrix} a_0 & a_1^{'} & a_2^{'} & a_3^{'} \\ a_1 & a_0 & a_1^{'} & a_2^{'} \\ a_2 & a_1 & a_0 & a_1^{'} \\ a_3 & a_2 & a_1 & a_0 \end{pmatrix} \tag{3.20}$$

$$T_0 - T_1 = \begin{pmatrix} a_2^{'} - a_0 & a_3^{'} - a_1^{'} \\ a_1^{'} - a_1 & a_2^{'} - a_0 \end{pmatrix} \tag{3.21}$$

$$T_1 - T_2 = \begin{pmatrix} a_0 - a_2 & a_1^{'} - a_1 \\ a_1 - a_3 & a_0 - a_2 \end{pmatrix} \tag{3.22}$$

Now under let us focus the common term $a_1^{'} - a_1$ seen as lower triangle at Equation 3.21, and upper triangle at Equation 3.22. Cost of this term is common for both $T_0 - T_1$ and $T_1 - T_2$, and this cost must be excluded from the overall addition cost as it had also been stated in Equation 3.17 accordingly. Recalling our partition sizes, the number of unique elements in $T_0 - T_1$ and $T_1 - T_2$ are $n - 1$ for each, which was also demonstrated in Equation 3.19. Number of diagonal element is 1 for all toeplitz matrices and all remaining elements are contained in either upper triangle or lower triangle, both of which have equal elements. For this reason, we can rewrite the decomposition Equation 2.1 given in Lemma 2.1.3 for our case as follow.

$$\text{\# of elements in any triangle} = \frac{\text{\# of total elements - 1}}{2} \tag{3.23}$$

Therefore, our common cost can be written as

$$M_{\bigoplus,tCommon}(n) = \frac{(n-1)-1}{2} = \frac{n}{2} - 1 \tag{3.24}$$

Finally, we are ready to combine all sub addition costs given in Equations 3.18, 3.19 and 3.24, and insert them into Equation 3.17 in order to get the final addition cost.

$$\begin{aligned} M_{\bigoplus}(n) = M_{\bigoplus,v1}(n) + M_{\bigoplus,v2}(n) + M_{\bigoplus,v3}(n) + \\ M_{\bigoplus,t1}(n) + M_{\bigoplus,t2}(n) - M_{\bigoplus,tCommon}(n) \end{aligned}$$

$$M_{\oplus}(n) = \frac{n}{2} + \frac{n}{2} + \frac{n}{2} + (n-1) + (n-1) - (\frac{n}{2} - 1)$$

$$\begin{aligned} M_{\oplus}(n) &= 3n - 1 && \text{where } M_{\oplus}(1) = 0 \\ M_{\oplus}(n) &= (2n-1)A_s + nA_d && \text{where } M_{\oplus}(1) = 0 \end{aligned} \tag{3.25}$$

Once we have the cost of addition given in Equation 3.25 and multiplication in Equation 3.16, we can sum them side by side to get the overall TMVP cost as follow.

$$M(n) = M_{\otimes}(n) + M_{\oplus}(n)$$

subjected to the following initial condition

$$\begin{aligned} M(1) &= M_{\otimes}(1) + M_{\oplus}(1) \\ M(1) &= 1M + 0A = 1M \end{aligned}$$

Our final cost function becomes as follows.

$$M(n) = 3M(n/2) + (2n-1)A_s + nA_d \qquad \text{where } M(1) = 1M \tag{3.26}$$

3.3.2 Delay Complexity

Hardware schematic of the construction is given below in Figure 3.3.

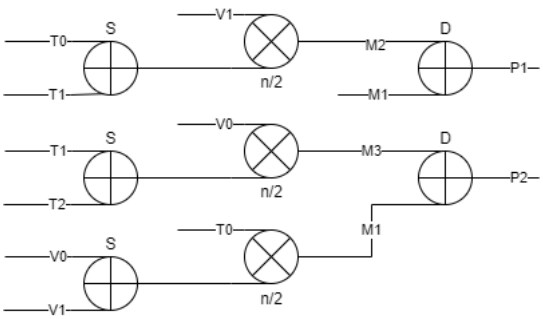

Figure 3.3: TMVP-STD-2 Hardware Implementation Block Scheme

$$\begin{aligned} D_{\oplus}(n) &= D_s + D_{\oplus}(n/2) + D_d & D_{\oplus}(1) &= 0 \\ D_{\otimes}(n) &= D_{\otimes}(n/2) & D_{\otimes}(1) &= D_m \\ D(n) &= D_{\otimes}(n) + D_{\oplus}(n) & D(1) &= D_m \\ D(n) &= D(n/2) + D_s + D_d & D(1) &= D_{\otimes} \end{aligned} \tag{3.27}$$

3.4 Three way formula with six multiplications (TMVP-STD-3)

Similar approach with 'TMVP-STD-2' is applied here, but this time the division factor is 3.

$$P_n = T_n V_n = \begin{pmatrix} T_2 & T_1 & T_0 \\ T_3 & T_2 & T_1 \\ T_4 & T_3 & T_2 \end{pmatrix} \begin{pmatrix} V_1 \\ V_2 \\ V_3 \end{pmatrix} = \begin{pmatrix} P_1 + P_4 + P_5 \\ P_2 - P_4 + P_6 \\ P_3 - P_5 - P_6 \end{pmatrix} \tag{3.28}$$

$$P_1 = (T_0 + T_1 + T_2)V_2 \tag{3.29}$$
$$P_2 = (T_1 + T_2 + T_3)V_1 \tag{3.30}$$
$$P_3 = (T_2 + T_3 + T_4)V_0 \tag{3.31}$$
$$P_4 = T_1(V_1 - V_2) \tag{3.32}$$
$$P_5 = T_2(V_0 - V_2) \tag{3.33}$$
$$P_6 = T_3(V_0 - V_1) \tag{3.34}$$

3.4.1 Arithmetic Complexity

Note that each T_i matrices are also Toeplitz in the above matrix partition, and each has size $n/3 \times n/3$. Each partition vector V_i has size $n/3 \times 1$. One might ask whether every matrix is suitable for the above partition construction. The answer is yes if the size is a multiple of 3. If the size is not a multiple of 3, consider the following two options. The first option is to ignore and skip this approach for sizes not multiple of 3, which we refer to as 'No Padding' case in Chapter 5. The other option is to utilize a padding operation to make the size of multiple of 3, which we call 'Padding Allowed' in Chapter 5. The methodology of padding and how it preserves the form of the Toeplitz matrix is out of scope for this thesis. On the other hand, if such padding construction is somehow given, integrating it through the hybrid analysis stage will be explained in Chapter 5. For now, let us assume that the matrix size is a multiple of 3. If the following substitutions are used as in [24], there yields 6 sub TMVP arises.

$$P_1 = (T_0 + T_1 + T_2)V_2 \tag{3.35}$$
$$P_2 = (T_1 + T_2 + T_3)V_1 \tag{3.36}$$
$$P_3 = (T_2 + T_3 + T_4)V_0 \tag{3.37}$$
$$P_4 = T_1(V_1 + V_2) \tag{3.38}$$
$$P_5 = T_2(V_0 + V_2) \tag{3.39}$$
$$P_6 = T_3(V_0 + V_1) \tag{3.40}$$

Using these substitution allows us to construct below TMVP

$$\begin{pmatrix} T_2 & T_1 & T_0 \\ T_3 & T_2 & T_1 \\ T_4 & T_3 & T_2 \end{pmatrix} \begin{pmatrix} V_1 \\ V_2 \\ V_3 \end{pmatrix} = \begin{pmatrix} P_1 + P_4 + P_5 \\ P_2 + P_4 + P_6 \\ P_3 + P_5 + P_6 \end{pmatrix} \tag{3.41}$$

with the following multiplication cost.

$$M_{\bigotimes}(n) = 6M_{\bigotimes}(n/3) \qquad \text{where } M_{\bigotimes}(1) = 1 \tag{3.42}$$

In order to count the additions, we can separate them as matrix and vector additions for the sake of simplicity. Let us consider matrix additions first. During the addition of Toeplitz matrices, many overlapping terms must be ignored in the cost calculation. However, determining

these overlapping terms might not be easy, especially in the case of several matrices. For this reason, we are going to use a vector approach in the representation of Toeplitz matrices. From Lemma 2.1.3, we know that $2n-1$ elements are enough to identify a Toeplitz matrix of size $n \times n$. Hence any Toeplitz matrix T_i can be represented as a concatenation of vectors as demonstrated below.

$$T_i = \begin{pmatrix} t_{(i+1)n-1} & T_{iU} \\ T_{iL} & t_{(i+1)n-1} \end{pmatrix}$$

$$T_i = (T_{iU}||t_{(i+1)n-1}||T_{iL}) \tag{3.43}$$

In the above representation t_{n-1} is a single element which corresponds to the diagonal element of the Toeplitz matrix T_i of size $n \times n$. Other vectors T_{iU} and T_{iL} represents upper and lower triangle of the matrix respectively, both of which have same number of elements satisfying $|T_{iU}| = |T_{iL}| = n-1$. Note that elements of these vectors contain the minimum number of elements required to define a triangle of Toeplitz and do not contain repetitive elements at the same diagonals. For our case, partitioned Toeplitz matrices have a size of $n/3 \times n/3$, so we replace n by $n/3$ during cost calculations or counting the number of elements in a vector. Before we are going to make our representation more useful for our case, we introduce the following definition.

Definition 3.4.1. *Two Toeplitz matrices are called adjacent if a set consisting of one of the Toeplitz's upper triangle elements equals the other set, including the other Toeplitz's lower triangle elements.*

From the above Definition 3.4.1, one could trivially see that partitioned Toeplitz matrices which are adjacent to each other are adjacent Toeplitz matrices. The below example represents two adjacent Toeplitz matrices of size 3×3.

$$T_1 = \begin{pmatrix} t_5 & t_4 & t_3 \\ t_6 & t_5 & t_4 \\ t_7 & t_6 & t_5 \end{pmatrix} T_0 = \begin{pmatrix} t_2 & t_1 & t_0 \\ t_3 & t_2 & t_1 \\ t_4 & t_3 & t_2 \end{pmatrix}$$

If we denote lower and upper triangles of a toeplitz with L and U index-suffixes. We can see the equality of sets as below.

$$\{T_{1U}\} = \{T_{0L}\} = \{t_3, t_4\}$$

We can extend the above example for any toeplitz matrices as below where two consecutive toeplitz matrices T_i are adjacent.

$$\{T_{(i+1)U}\} = \{T_{iL}\} \tag{3.44}$$

Therefore, we can construct more flexible and equivalent representation of Equation 3.43 for a toeplitz matrix T_i of size $n \times n$ by means of Equation 3.44 as below.

$$T_i = (\widetilde{T}_i || t_{(i+1)n-1} || \widetilde{T}_{i+1}) \qquad \text{where} \qquad \{\widetilde{T}_i\} = \{t_{in}, \ldots, t_{(i+1)n-2}\} \tag{3.45}$$

Returning back to our substitution Equations starting from 3.35 to 3.40, we can finally identify partitioned matrices by utilizing 3.45 as follow. Note that the size is replaced by $n/3$ since the partitioned matrices have the original size divided by 3.

$$\begin{aligned}
T_0 &= (\widetilde{T}_0 || t_{\frac{n}{3}-1} || \widetilde{T}_1) = (\widetilde{T}_0 || \widetilde{t}_{d0} || \widetilde{T}_1) \\
T_1 &= (\widetilde{T}_1 || t_{\frac{2n}{3}-1} || \widetilde{T}_2) = (\widetilde{T}_1 || \widetilde{t}_{d1} || \widetilde{T}_2) \\
T_2 &= (\widetilde{T}_2 || t_{\frac{3n}{3}-1} || \widetilde{T}_3) = (\widetilde{T}_2 || \widetilde{t}_{d2} || \widetilde{T}_3) \\
T_3 &= (\widetilde{T}_3 || t_{\frac{4n}{3}-1} || \widetilde{T}_4) = (\widetilde{T}_3 || \widetilde{t}_{d3} || \widetilde{T}_4) \\
T_4 &= (\widetilde{T}_4 || t_{\frac{5n}{3}-1} || \widetilde{T}_5) = (\widetilde{T}_4 || \widetilde{t}_{d4} || \widetilde{T}_5)
\end{aligned}$$

Now, let us write the toeplitz summations

$$\begin{aligned}
T_0 + T_1 + T_2 &= (\widetilde{T}_0 + \widetilde{T}_1 + \widetilde{T}_2 || \widetilde{t}_{d0} + \widetilde{t}_{d1} + \widetilde{t}_{d2} || \widetilde{T}_1 + \widetilde{T}_2 + \widetilde{T}_3) \\
T_1 + T_2 + T_3 &= (\widetilde{T}_1 + \widetilde{T}_2 + \widetilde{T}_3 || \widetilde{t}_{d1} + \widetilde{t}_{d2} + \widetilde{t}_{d3} || \widetilde{T}_2 + \widetilde{T}_3 + \widetilde{T}_4) \\
T_2 + T_3 + T_4 &= (\widetilde{T}_2 + \widetilde{T}_3 + \widetilde{T}_4 || \widetilde{t}_{d2} + \widetilde{t}_{d3} + \widetilde{t}_{d4} || \widetilde{T}_3 + \widetilde{T}_4 + \widetilde{T}_5)
\end{aligned}$$

Note that $\widetilde{T}_i$ vectors do not have overlapping elements and have same number of element, $\frac{n}{3} - 1$; which also gives us the summation cost of any two of them. In order to get above summations, one can construct the below summation sequence and corresponding summation cost without loss of generality.

$$M_{\bigoplus,\widetilde{T}_X}(n) \leftarrow \text{addition cost of } \widetilde{T}_X \leftarrow \widetilde{T}_1 + \widetilde{T}_2$$
$$M_{\bigoplus,\widetilde{T}_Y}(n) \leftarrow \text{addition cost of } \widetilde{T}_Y \leftarrow T_0 + \widetilde{T}_X$$
$$M_{\bigoplus,\widetilde{T}_Z}(n) \leftarrow \text{addition cost of } \widetilde{T}_Z \leftarrow \widetilde{T}_X + \widetilde{T}_3$$
$$M_{\bigoplus,\widetilde{T}_A}(n) \leftarrow \text{addition cost of } \widetilde{T}_A \leftarrow \widetilde{T}_3 + \widetilde{T}_4$$
$$M_{\bigoplus,\widetilde{T}_B}(n) \leftarrow \text{addition cost of } \widetilde{T}_B \leftarrow \widetilde{T}_2 + \widetilde{T}_A$$
$$M_{\bigoplus,\widetilde{T}_C}(n) \leftarrow \text{addition cost of } \widetilde{T}_C \leftarrow \widetilde{T}_A + \widetilde{T}_5$$
$$M_{\bigoplus,t2}(n) \leftarrow \text{addition cost of } X + T_A$$
$$M_{\bigoplus,t3}(n) \leftarrow \text{addition cost of } Y + T_4$$

Therefore addition cost of triangle sides of toeplitz matrices would be calculated like below Equation 3.46.

$$M_{\bigoplus,T_{triangle}}(n) \leftarrow M_{\bigoplus,\widetilde{T}_X}(n) + M_{\bigoplus,\widetilde{T}_Y}(n)$$
$$+M_{\bigoplus,\widetilde{T}_Z}(n) + M_{\bigoplus,\widetilde{T}_A}(n) + M_{\bigoplus,\widetilde{T}_B}(n) + M_{\bigoplus,\widetilde{T}_C}(n)$$

$$M_{\bigoplus,T_{triangle}}(n) = 6(\frac{n}{3} - 1) = 2n - 6 \tag{3.46}$$

Remaining component of the summation cost of toeplitz matrices consists of individual diagonal elements (i.e single element), for which we had denoted t_i.

$$M_{\bigoplus,\widetilde{t}_X}(n) \leftarrow \text{addition cost of } \widetilde{t}_X \leftarrow \widetilde{t}_{d1} + \widetilde{t}_{d2}$$
$$M_{\bigoplus,\widetilde{t}_Y}(n) \leftarrow \text{addition cost of } \widetilde{t}_Y \leftarrow \widetilde{t}_{d0} + \widetilde{t}_X$$
$$M_{\bigoplus,\widetilde{t}_Z}(n) \leftarrow \text{addition cost of } \widetilde{t}_Z \leftarrow \widetilde{t}_X + \widetilde{t}_{d3}$$
$$M_{\bigoplus,\widetilde{t}_A}(n) \leftarrow \text{addition cost of } \widetilde{t}_A \leftarrow \widetilde{t}_{d2} + \widetilde{t}_{d3}$$
$$M_{\bigoplus,\widetilde{t}_B}(n) \leftarrow \text{addition cost of } \widetilde{t}_B \leftarrow \widetilde{t}_A + \widetilde{t}_{d4}$$

$$M_{\bigoplus,t_{diagonal}}(n) \leftarrow M_{\bigoplus,\widetilde{t}_X}(n) + M_{\bigoplus,\widetilde{t}_Y}(n) + M_{\bigoplus,\widetilde{t}_Z}(n) + M_{\bigoplus,\widetilde{t}_A}(n) + M_{\bigoplus,\widetilde{t}_B}(n)$$

$$M_{\bigoplus,t_{diagonal}}(n) = 5 \tag{3.47}$$

If we denote the overall summation cost coming from the toeplitz additions as $M_{\bigoplus,T}(n)$, we can find it by summing the costs coming from triangle elements in Equation 3.46, and diagonal elements from Equation 3.47.

$$M_{\bigoplus,T}(n) = M_{\bigoplus,T_{triangle}}(n) + M_{\bigoplus,t_{diagonal}}(n)$$

$$M_{\bigoplus,T}(n) = 2n - 1 \tag{3.48}$$

Final contribution to the addition cost comes from the addition of vectors. See Equations from 3.38 to 3.40. There comes 3 vector addition of size $\frac{n}{3} \times 1$ coming from Equations 3.38 to 3.40. Additionally, there comes 6 more which can be seen in 3.41. There is no common or overlapping terms which lead us to write down the overall 9 vector summation costs as demonstrated in below.

$$M_{\bigoplus,V}(n) = 9(n/3) = 3n \tag{3.49}$$

Overall addition cost is the one which is summation of toeplitz matrix addition and vector addition costs, represented in Equations 3.48 and 3.49 respectively.

$$M_{\bigoplus}(n) = M_{\bigoplus,T}(n) + M_{\bigoplus,V}(n) = 5n - 1 \tag{3.50}$$

We can classify the addition cost with respect to the cost metrics single and double additions denoted A_s and A_d respectively (See Section 2.2). Recall the assumption that we made in Section 1.3 such that each element of matrix and vectors are assumed to be as 'single' word. This assumption results in additions of type 'single' coming from toeplitz additions given by Equation 3.48 and vector additions from Equation 3.38 to 3.40. On the other hand, the additions of substitution terms $P_i s$ in Equation 3.41 results in additions of type 'double' since each of term is actually a multiplication of two 'single' word. Briefly, we can refactor the summation cost given in 3.50 in terms of single and double word addition cost metrics as below.

$$M_{\bigoplus}(n) = (3n - 1)A_s + (2n)A_d \tag{3.51}$$

Finally, we can combine multiplication cost given by Equation 3.42, and addition cost from Equation 3.51; which results in the following overall multiplication cost.

$$M(n) = 6M(n/3) + (3n - 1)A_s + (2n)A_d \qquad \text{where } M(1) = 1M \tag{3.52}$$

3.4.2 Delay Complexity

Figure 3.4 summarizes the construction whose substitutions are presented from Equation 3.35 to 3.40.

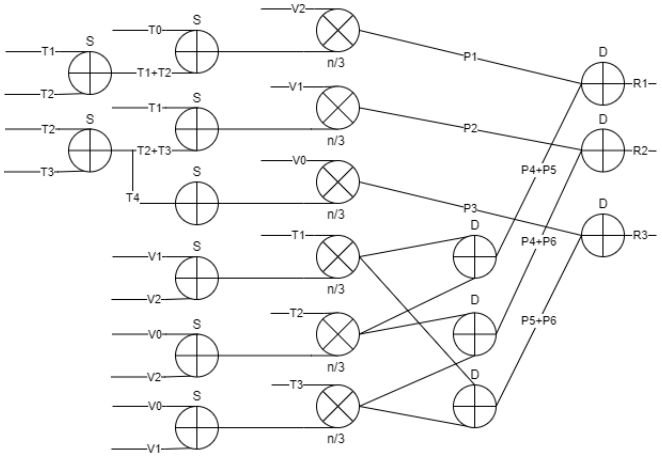

Figure 3.4: TMVP-STD-3 Hardware Implementation Block Scheme

$$
\begin{aligned}
D_{\oplus}(n) &= D_s + D_{\oplus}(n/3) + 2D_d & \qquad D_{\oplus}(1) &= 0 \\
D_{\otimes}(n) &= D_{\otimes}(n/3) & \qquad D_{\otimes}(1) &= D_m \\
D(n) &= D_{\otimes}(n) + D_{\oplus}(n) & \qquad D(1) &= D_m \\
D(n) &= D(n/3) + D_s + 2D_d & \qquad D(1) &= D_m
\end{aligned} \tag{3.53}
$$

3.5 Three way formula with five multiplications (TMVP-3)

Starting from the multiplication of two polynomials with three indeterminate, utilizing Toom-Cook algorithm steps, evaluation and interpolation and a sequence of equation refactoring; Paksoy in [24], derives the following TMVP scheme, for which we are going to express cost functions and find out arithmetic complexity.

$$
\begin{pmatrix} t_2 & t_1 & t_0 \\ t_3 & t_2 & t_1 \\ t_4 & t_3 & t_2 \end{pmatrix}
\begin{pmatrix} v_0 \\ v_1 \\ v_2 \end{pmatrix}
=
\begin{pmatrix} P_1 + P_2 + 4P_3 + P_4 \\ P_1 - P_2 - 2P_3 \\ P_0 + P_1 + P_2 + P_3 \end{pmatrix} \tag{3.54}
$$

where the following substitutions are given

$$P_0 = \frac{1}{2}(2t_4 + t_3 - 2t_2 - t_1)v_0 \tag{3.55}$$

$$P_1 = \frac{1}{6}(2t_3 + 3t_2 + t_1)(v_0 + v_1 + v_2) \tag{3.56}$$

$$P_2 = \frac{1}{2}(-2t_3 + t_2 + t_1)(v_0 - v_1 + v_2) \tag{3.57}$$

$$P_3 = \frac{1}{6}(t_3 - t_1)(v_0 - 2v_1 + 4v_2) \tag{3.58}$$

$$P_4 = \frac{1}{2}(-2t_3 - t_2 + 2t_1 + t_0)(v_2) \tag{3.59}$$

3.5.1 Arithmetic Complexity

Let us first express the multiplication cost, which is trivially seen by the above construction at which 5 multiplication corresponding to 5 substitution Equations (from P_0 to P_4), each of which has the original size divided by 3.

$$M_{\bigotimes}(n) = 5M_{\bigotimes}(n/3) \qquad \text{where } M_{\bigotimes}(1) = 1 \tag{3.60}$$

In order to count additions, we can group them as vector and toeplitz matrix additions. Let us count toeplitz additions, each of which are 'single' word additions under the assumption that we made in Section 1.3

$$\begin{aligned} M_{\bigoplus,t0}(n) &\leftarrow \text{addition cost of } 2t_4 - 2t_2 + t_3 - t_1 \\ M_{\bigoplus,t1}(n) &\leftarrow \text{addition cost of } 2t_3 + 3t_2 + t_1 \\ M_{\bigoplus,t2}(n) &\leftarrow \text{addition cost of } -2t_3 + t_2 + t_1 \\ M_{\bigoplus,t3}(n) &\leftarrow \text{addition cost of } t_1 - t_3 \\ M_{\bigoplus,t4}(n) &\leftarrow \text{addition cost of } -2t_3 + 2t_1 - t_2 + t_0 \end{aligned}$$

To provide the consistency with the work [24], scalar multiplications will not be included during the cost calculation.

$$2t_3 + 3t_2 + t_1 = (t_1 + t_2) + 2(t_2 + t_3) \leftarrow M_{\bigoplus,t1}(n) = (n-1) + (\frac{2n}{3} - 1) \tag{3.61}$$

In the above cost calculation 3.61, we realize that t_1, t_2, t_3 are 'adjacent' toeplitz matrices (see the definition 3.4.1). Addition of t_1, t_2 and t_2, t_3 results in some common overlapping terms. Number of such terms are equal one of the triangle terms (either lower or upper). So for the size $n/3 \times n/3$, there are such $\frac{n}{3} - 1$ terms which are needed to be excluded at the final cost calculation. This is why $n - 1 = 2(\frac{2n}{3} - 1) - (\frac{n}{3} - 1)$ operations are enough to get $t_1 + t_2$ and $t_2 + t_3$. Note that having these terms, and ignoring the scalar multiplication; summation of these two terms requires $2(\frac{n}{3}) - 1$, which is nothing but the number of distinct elements in a $n/3 \times n/3$ toeplitz matrix. (See Lemma 2.1.3.)

$t_2 + t_1$ were computed in $M_{\bigoplus,t1}(n)$, 1 single addition is enough

$$-2t_3 + (t_2 + t_1) \leftarrow M_{\bigoplus,t2}(n) = \frac{2n}{3} - 1 \tag{3.62}$$

$$t_1 - t_3 \leftarrow M_{\bigoplus,t3}(n) = \frac{2n}{3} - 1 \tag{3.63}$$

$t_1 - t_3$ were computed in $M_{\bigoplus,t3}(n)$, 2 single additions are enough

$$-2t_3 + 2t_1 - t_2 + t_0 \leftarrow M_{\bigoplus,t4}(n) = 2(\frac{2n}{3} - 1) \tag{3.64}$$

$t_1 - t_3$ were computed in $M_{\bigoplus,t3}(n)$, 2 single additions are enough

$$2t_4 - 2t_2 + t_3 - t_1 \leftarrow M_{\bigoplus,t0}(n) = 2(\frac{2n}{3} - 1) \tag{3.65}$$

For overall toeplitz addition, we sum up from Equations 3.62 to 3.65, we would get the following.

$$M_{\bigoplus,t}(n) = M_{\bigoplus,t0}(n) + M_{\bigoplus,t1}(n) + M_{\bigoplus,t2}(n) + M_{\bigoplus,t3}(n) + M_{\bigoplus,t4}(n)$$
$$M_{\bigoplus,t}(n) = 7(\frac{2n}{3} - 1) + (n - 1) = (\frac{17n}{3} - 8)A_s \tag{3.66}$$

Now let us continue counting vector additions. In below, we denote vector additions seen in substitution terms, which will yield again single additions.

$$M_{\bigoplus,vs0}(n) \leftarrow \text{addition cost of } v_0 + v_2$$
$$M_{\bigoplus,vs1}(n) \leftarrow \text{addition cost of } v_0 + v_1 + v_2$$
$$M_{\bigoplus,vs2}(n) \leftarrow \text{addition cost of } v_0 - v_1 + v_2$$
$$M_{\bigoplus,vs3}(n) \leftarrow \text{addition cost of } v_0 - 2v_1 + 4v_2$$

In above, $v_0 + v_2$ are common to all vector additions except $M_{\bigoplus,vs3}(n)$, and it is enough to calculate it once and reuse it in $M_{\bigoplus,vs1}(n)$ and $M_{\bigoplus,vs2}(n)$. Then the following costs are trivial.

$$M_{\bigoplus,vs0}(n) = \frac{n}{3} \tag{3.67}$$
$$M_{\bigoplus,vs1}(n) = \frac{n}{3} \tag{3.68}$$
$$M_{\bigoplus,vs2}(n) = \frac{n}{3} \tag{3.69}$$
$$M_{\bigoplus,vs3}(n) = \frac{2n}{3} \tag{3.70}$$

Therefore the single additions coming from vectors can be summarized as follow.

$$M_{\bigoplus,vs}(n) = M_{\bigoplus,vs0}(n) + M_{\bigoplus,vs1}(n) + M_{\bigoplus,vs2}(n) + M_{\bigoplus,vs3}(n)$$
$$M_{\bigoplus,vs}(n) = \frac{5n}{3}A_s \tag{3.71}$$

Final contribution to the vector addition comes from the summation of P_i s seen in the result vector, which is given in the construction 3.76. Let us denote the costs as below. Note that these these terms are 'double' in contrast to the above vector additions because they are appeared as the multiplication of single terms in toeplitz matrix and vector.

$$M_{\bigoplus,vd0}(n) \leftarrow \text{addition cost of } P_1 + P_2 + 4P_3 + P_4$$
$$M_{\bigoplus,vd1}(n) \leftarrow \text{addition cost of } P_1 - P_2 - 2P_3$$
$$M_{\bigoplus,vd2}(n) \leftarrow \text{addition cost of } P_0 + P_1 + P_2 + P_3$$

In above vector additions, there are total of 8 vector additions, one of which $P_1 + P_2$ are used twice. Therefore we can demonstrate the 7 vector addition cost as below.

$$M_{\bigoplus,vd}(n) = M_{\bigoplus,vd0}(n) + M_{\bigoplus,vd1}(n) + M_{\bigoplus,vd2}(n)$$
$$M_{\bigoplus,vd}(n) = \frac{7n}{3}A_d \tag{3.72}$$

We can finally bring together Equations 3.71, 3.72 and 3.66 for addition cost.

$$M_{\bigoplus}(n) = M_{\bigoplus,vs}(n) + M_{\bigoplus,vd}(n) + M_{\bigoplus,t}(n)$$
$$M_{\bigoplus}(n) = \frac{5n}{3}A_s + \frac{7n}{3}A_d + (\frac{17n}{3} - 8)A_s$$
$$M_{\bigoplus}(n) = (\frac{22n}{3} - 8)A_s + \frac{7n}{3}A_d \tag{3.73}$$

Combining the overall addition cost given by Equation 3.73 and multiplication cost given in Equation 3.60 yields the following overall cost function of the 'TMVP-TOOM-3' construction.

$$M(n) = M_{\bigoplus}(n) + M_{\bigotimes}(n)$$
$$M(n) = 5M(n/3) + (\frac{22n}{3} - 8)A_s + \frac{7n}{3}A_d \tag{3.74}$$

3.5.2 Delay Complexity

In this construction, the maximum delay path starts with the single addition of 4 terms which comes from the addition of partitioned Toeplitz matrices. After the usual recursive multiplication of the TMVP instance, there seem to be four more terms to be exposed to double addition which comes from the vector addition of substituted terms P_i s. As it has already been stated in Section 3.1, there occurs $\lceil log_2(n) \rceil$ cascaded stage of addition block for n number of terms to be summed. Visualization of this preposition was also given in Figure 3.1. Therefore, we can immediately express the number of the cascaded stage as $\lceil log_2(4) \rceil = 2$ for both types of addition (i.e., single and double).

$$\begin{aligned} D_{\bigoplus}(n) &= 2D_s + D_{\bigoplus}(n/3) + 2D_d & D_{\bigoplus}(1) &= 0 \\ D_{\bigotimes}(n) &= D_{\bigotimes}(n/3) & D_{\bigotimes}(1) &= D_m \\ D(n) &= D_{\bigotimes}(n) + D_{\bigoplus}(n) & D(1) &= D_m \\ D(n) &= D(n/3) + 2D_s + 2D_d & D(1) &= D_{\bigotimes} \end{aligned} \tag{3.75}$$

3.6 Four way fomula (TMVP-4)

Similar to the previous construction 'TMVP-TOOM-3', Paksoy in [24] finds out substitutions having the division size 4.

$$\begin{pmatrix} t_3 & t_2 & t_1 & t_0 \\ t_4 & t_3 & t_2 & t_1 \\ t_5 & t_4 & t_3 & t_2 \\ t_6 & t_5 & t_4 & t_3 \end{pmatrix} \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{pmatrix} = \begin{pmatrix} P_1 - P_2 + 8P_3 - 8P_4 + 27P_5 + P_6 \\ P_1 + P_2 + 4P_3 + 4P_4 + 9P_5 \\ P_1 - P_2 + 2P_3 - 2P_4 + 3P_5 \\ P_0 + P_1 + P_2 + P_3 + P_4 + P_5 \end{pmatrix} \tag{3.76}$$

where the following substitutions are given

$$P_0 = \frac{1}{12}(12t_6 - 4t_5 - 15t_4 + 5t_3 + 3t_2 - t_1)v_0 \tag{3.77}$$

$$P_1 = \frac{1}{12}(12t_5 + 8t_4 - 7t_3 - 2t_2 + t_1)(v_0 + v_1 + v_2 + v_3) \tag{3.78}$$

$$P_2 = \frac{1}{24}(-12t_5 + 16t_4 - t_3 - 4t_2 + t_1)(v_0 - v_1 + v_2 - v_3) \tag{3.79}$$

$$P_3 = \frac{1}{24}(-6t_5 - t_4 + 7t_3 + t_2 - t_1)(v_0 + 2v_1 + 4v_2 + 8v_3) \tag{3.80}$$

$$P_4 = \frac{1}{120}(6t_5 - 5t_4 - 5t_3 + 5t_2 - t_1)(v_0 - 2v_1 + 4v_2 - 8v_3) \tag{3.81}$$

$$P_5 = \frac{1}{120}(4t_5 - 5t_3 + t_1)(v_0 + 3v_1 + 9v_2 + 27v_3) \tag{3.82}$$

$$P_6 = (-12t_5 + 4t_4 + 15t_3 - 5t_2 - 3t_1 + t_0)(v_3) \tag{3.83}$$

3.6.1 Arithmetic Complexity

Using the same approach given in previous Section 3.5 and ignoring the scalar multiplications, we can conclude the arithmetic complexity as follow.

$$M(n) = 7M(n/4) + (\frac{65n}{4} - 27)A_s + \frac{7n}{2}A_d \tag{3.84}$$

3.6.2 Delay Complexity

After six cascaded single addition term initiating the recursive multiplication block, six more double addition terms are required to compute the final result. Therefore, there are $\lceil log_2(6) \rceil = 3$ cascaded stage of addition for both types of addition (i.e., single and double).

$$
\begin{aligned}
D_{\bigoplus}(n) &= 3D_s + D_{\bigoplus}(n/4) + 3D_d & D_{\bigoplus}(1) &= 0 \\
D_{\bigotimes}(n) &= D_{\bigotimes}(n/4) & D_{\bigotimes}(1) &= D_m \\
D(n) &= D_{\bigotimes}(n) + D_{\bigoplus}(n) & D(1) &= D_m \\
D(n) &= D(n/4) + 3D_s + 3D_d & D(1) &= D_m
\end{aligned} \tag{3.85}
$$

3.7 Unbalanced (TMVP-UNB)

This construction provides a TMVP, in which a matrix of size nxn is divided partitions of arbitrary size of $k < n$ in such a way that partitioned matrix namely M contains four sub-matrices, two of which preserves the form of Toeplitz, denoted by T_1 and T_2. Other two matrices have no special form unless otherwise stated, and they are denoted by S_1 and S_2. Note that these matrices do not have to be in square form. Size and partitions of the product have been summarized as below.

$$
M_{nxn} = \left(\begin{array}{c|c} (T_1)_{kxk} & (S_1)_{kx(n-k)} \\ \hline (S_2)_{(n-k)xk} & (T_2)_{(n-k)x(n-k)} \end{array} \right)
$$

$$
V_{nx1} = \left(\begin{array}{c} (V_1)_{kx1} \\ \hline (V_2)_{(n-k)x1} \end{array} \right)
$$

$$
P_{nx1} = \left(\begin{array}{c} (P_1)_{kx1} \\ \hline (P_2)_{(n-k)x1} \end{array} \right)
$$

$$
P = MV
$$

$$
\left(\begin{array}{c} P_1 \\ \hline P_2 \end{array} \right) = \left(\begin{array}{c|c} T_1 & S_1 \\ \hline S_2 & T_2 \end{array} \right) \left(\begin{array}{c} V_1 \\ \hline V_2 \end{array} \right) = \left(\begin{array}{c} T_1V_1 + S_1V_2 \\ \hline S_2V_1 + T_2V_2 \end{array} \right)
$$

In this construction, four invocations of the multiplication blocks exist. Two of these invocations are regular TMVP with sizes k and $n - k$ where the original multiplication problem size is $n \times n$ and the partition $k < n$. These multiplication blocks can call any other suitable TMVP construction which minimizes the target cost (delay, space or delay times space, and so on). On the other hand, the matrix need not be in Toeplitz form, nor even symmetric for the other two multiplication.

3.7.1 Arithmetic Complexity

Let us calculate the number of operations in P_1 without any optimization in the S_1V_2 product.

$$\begin{aligned} T_1V_1 &\rightarrow M(k) \\ S_1V_2 &\rightarrow k(n-k)M + k(n-k-1)A_d \\ T_1V_1 + S_1V_2 &\rightarrow kA_d \end{aligned}$$

Once we calculate P_1, it is easy to derive for P_2 by just making the following transformation.

$$k \rightarrow n-k$$

This allows us to re-factor the above arithmetic costs for P_2 as follows

$$\begin{aligned} T_2V_2 &\rightarrow M(n-k) \\ S_2V_1 &\rightarrow (n-k)kM + (n-k)(k-1)A_d \\ T_2V_2 + S_2V_1 &\rightarrow (n-k)A_d \end{aligned}$$

Summing up costs for P_1 and P_2, we end up with the cost function $M(n,k)$ in terms of matrix size n and the inner partition size k as follows.

$$M(n,k) = M(k) + M(n-k) + 2(n-k)kM + 2(n-k)k(A_d) \tag{3.86}$$

We describe a cost function as symmetric if it satisfy the following condition

$$M(n,k) = M(n,n-k) \tag{3.87}$$

Note that cost function of TMVP-UNB given in Equation 3.86 is symmetric which in turn it is sufficient to check partition sizes for $k \leq n/2$. We are going to use this fact to our advantage during the simulation stage.

3.7.2 Delay Complexity

Let us denote delays for each multiplication block by its operands. Then, the maximum of D_{T1V1},D_{S1V2},D_{S2V1},D_{T2V2}, is the delay corresponding the overall delay through the final

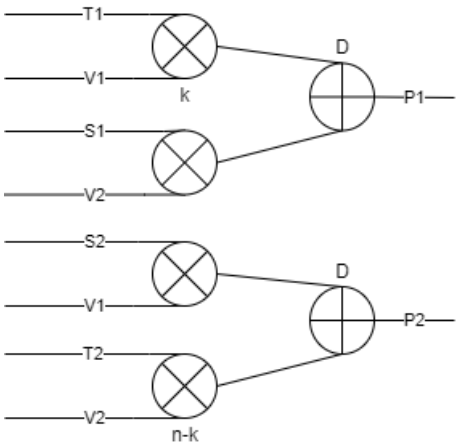

Figure 3.5: TMVP-LT Hardware Implementation Block Scheme

double addition blocks by Lemma 2.2.2. This delay is summed up by a final double addition block delay.

$$D(n) = max\{D_{T1V1}, D_{S1V2}, D_{S2V1}, D_{T2V2}\} + D_d \tag{3.88}$$
$$D_{T1V1} = D(k)$$
$$D_{T2V2} = D(n-k)$$

$$D(n) = max\{D(k), D(n-k), D_{TMVP-SB}(max\{k, n-k\})\} + D_d \quad \text{where} \quad k \leq \frac{n}{2} \tag{3.89}$$

3.8 Last Term (TMVP-LT)

Last term method methodology was applied in another article in the context of polynomial multiplication. (See [8]) This section applies the same approach in the scope of TMVP and demonstrates the complexity result. Notice that the last term approach is a special case of the previous section, i.e., unbalanced TMVP, where the partition size $k = n - 1$ or $k = 1$ where the n is square matrix size.

$$M_{nxn} = \left(\begin{array}{c|c} (T_1)_{(n-1)x(n-1)} & (S_1)_{(n-1)x1} \\ \hline (S_2)_{1x(n-1)} & (T_2)_{1x1} \end{array} \right)$$

$$V_{nx1} = \left(\begin{array}{c} (V_1)_{(n-1)x1} \\ \hline (V_2)_{1x1} \end{array} \right)$$

$$P_{nx1} = \left(\begin{array}{c} (P_1)_{kx1} \\ \hline (P_2)_{(n-k)x1} \end{array} \right)$$

$$P = MV$$

3.8.1 Arithmetic Complexity

We obtain arithmetic cost function by simply setting $k = 1$ into the Equation 3.86 where the initial condition is taken as $M(1) = 1M$.

$$M_{TMVP-LT}(n) = M_{TMVP-UNB}(n, 1)$$
$$M_{TMVP-LT}(n) = M(1) + M(n-1) + 2(n-1)M + 2(n-1)A_d \quad \text{where } M(1) = 1M$$

$$M_{TMVP-LT}(n) = M(n-1) + (2n-1)M + 2(n-1)A_d \quad \text{where } M(1) = 1M \quad (3.90)$$

3.8.2 Delay Complexity

This approach is a special case of 'Unbalanced TMVP' where the partition size k is 1. Therefore the block scheme is same as Figure 3.5 with $k = 1$. We can simply refactor the Equation 3.91 as below.

$$D(n) = max\{D(1), D(n-1), D_{TMVP-SB}(max\{1, n-1\})\} + D_d$$
$$D(n) = max\{D_m, D(n-1), D_{TMVP-SB}(n-1)\} + D_d \quad (3.91)$$

CHAPTER 4

THE DERIVATION OF ARITHMETIC AND DELAY COMPLEXITIES FOR POLYNOMIAL MULTIPLICATION CONSTRUCTIONS

This chapter derives arithmetic and delay complexities for various polynomial multiplication methods in recursive cost functions. Ilter, in his thesis, [17], has already derived and revisited the arithmetic complexities of various polynomial multiplication approaches. In this chapter, we are going to extend his work in terms of delay complexities. Convention used throughout the chapter is the same as the previous Chapter 3 at which we have derived both arithmetic and delay complexities of various TMVP schemes. Besides, we are going to use the stick diagram approach used in [17]. Furthermore, we are going to demonstrate hardware block schemes in order to justify our delay complexity analysis of polynomial multiplications. Throughout the derivations, we ignore the final reduction stage. We include the cost of the reduction stage for modular polynomial multiplication in the final result chapter (See Chapter 6), where we make a comparison of all algorithms in terms of various target cost metrics.

4.1 Schoolbook Iterative (POLY-SB-ITR)

The schoolbook approach is the naive one in which a direct multiplication of two polynomials arises. This section analyzes the case where there is no recursion during the multiplication. In contrast to the arithmetic complexity, how we multiply using schoolbook alters the cost of delay. Let $a(x)$ and $b(x)$ be polynomials of degree $n-1$, and $c(x)$ be the multiplication result.

$$a(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$$
$$b(x) = b_0 + b_1 x + \ldots + b_{n-1} x^{n-1}$$
$$c(x) = a(x)b(x) = c_0 + c_1 x + \ldots + c_{2n-2} x^{2n-2}$$

4.1.1 Arithmetic Complexity

Number of multiplication is trivial. For each element a_i in $a(x)$ is multiplied n times with the elements of polynomial $b(x)$ of size n (or vice versa). Since there are n more such elements in $a(x)$ to be multiplied with elements of $b(x)$, we have $n^2 = n.n$ multiplication operation, which we can summarize as below.

$$M_{\otimes}(n) = n^2 M \tag{4.1}$$

Examining the result polynomial $c(x)$, we see that each element c_i consists of i summation of double word where $i \leq n-1$ and $2n-2-i$ elements for $n-1 < i < 2n-1$. We can visualize the case as below.

$$\begin{array}{rlr} c_0 = a_0 b_0 & \text{(No Summation)} & c_{2n-2} = a_{n-1} b_{n-1} \\ c_1 = a_0 b_1 + a_1 b_0 & \text{(1 Summation)} & c_{2n-3} = a_{n-1} b_{n-2} + a_{n-2} b_{n-1} \\ \vdots & & \vdots \\ c_{n-2} = a_0 b_{n-2} + \ldots + a_{n-2} b_0 & (n-2 \text{ Summation}) & c_n = a_1 b_{n-1} + \ldots + a_{n-1} b_1 \end{array}$$

Maximum number of term occur at the middle term where $i = n-1$.

$$c_{n-1} = a_0 b_{n-1} + \ldots + a_{n-1} b_0 \qquad (n-1 \text{ Summation})$$

Now, we are ready to express number of additions as follow.

$$\begin{aligned} M_{\oplus}(n) &= [2(1 + 2 + \ldots + (n-2)) + (n-1)] A_d \\ M_{\oplus}(n) &= [(n-2)(n-1) + (n-1)] A_d \\ M_{\oplus}(n) &= (n-1)^2 A_d \end{aligned} \tag{4.2}$$

We get the final arithmetic complexity by combining Equation 4.1 and 4.2.

$$\begin{aligned} M(n) &= M_{\otimes}(n) + M_{\oplus}(n) \\ M(n) &= n^2 M + (n-1)^2 A_d \end{aligned} \tag{4.3}$$

4.1.2 Delay Complexity

Maximum number of term to be summed occurs at the middle term of the result polynomial $c(x)$.

$$c_{n-1} = a_0 b_{n-1} + a_1 b_{n-2} + \ldots + a_{n-1} b_0$$

The term c_{n-1} seen in Equation 4.4 consist of n elements which can be computed $\lceil log_2(n) \rceil$ cascaded stages if they are grouped by 2 in each time until the end of the addition process. (See Figure 3.1). Before summation stage, computation of $a_i b_j$ must be handled. Since there

is not any dependent, consecutive multiplication; all of them can be computed in a single parallel stage.

$$\begin{aligned} D_{\bigotimes}(n) &= D_m \\ D_{\bigoplus}(n) &= \lceil log_2(n) \rceil D_d \\ D(n) &= D_{\bigotimes}(n) + D_{\bigoplus}(n) \\ D(n) &= D_m + \lceil log_2(n) \rceil D_d \end{aligned} \tag{4.4}$$

4.2 Schoolbook 2 Way (POLY-SB-2)

This approach realizes the naive multiplication in a recursive manner by dividing the polynomials into two smaller ones and then calling the original problem with smaller input sizes. Although the arithmetic complexity representing the number of elementary arithmetic operations remains the same, delay complexity differs. This is why the section introduces a new extension to the schoolbook approach. Let us multiply $a(x)\ b(x)$, but before divide them into A_0, A_1, B_0, B_1 as follow.

$$\begin{gathered} a(x) = A_0 + A_1 x^{n/2} \\ b(x) = B_0 + B_1 x^{n/2} \\ A_0 = a_0 + a_1 x + \ldots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1} \\ A_1 = a_{\frac{n}{2}} + a_{\frac{n}{2}+1} x + \ldots + a_{n-1} x^{\frac{n}{2}-1} \\ B_0 = b_0 + b_1 x + \ldots + b_{\frac{n}{2}-1} x^{\frac{n}{2}-1} \\ B_1 = b_{\frac{n}{2}} + b_{\frac{n}{2}+1} x + \ldots + b_{n-1} x^{\frac{n}{2}-1} \\ c(x) = a(x)b(x) = A_0B_0 + (A_0B_1 + A_1B_0)x^{n/2} + A_1B_1x^n \end{gathered}$$

In this configuration, overlapping terms are inevitable, and overhead during the summation of overlapping terms brings an additional delay term. If we denote $C_0 = A_0B_0$, $C_1 = A_0B_1 + A_1B_0$ and $C_2 = A_1B_1$, we can see the overlapping terms from the stick diagram given in Figure 4.1.

4.2.1 Arithmetic Complexity

We have total of $2(\frac{n}{2} - 1) + n - 1 = 2n - 3$ addition together with 4 multiplication call of half size of the original problem.

$$M(n) = 4M(n/2) + (2n-3)A_d \qquad\qquad M(1) = 1M \tag{4.5}$$

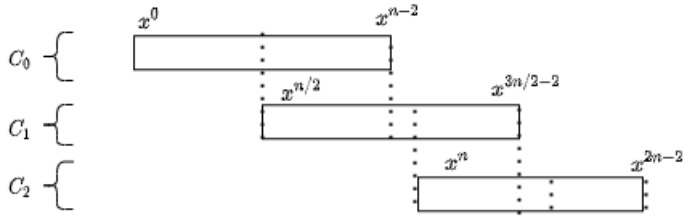

Figure 4.1: 2-Way Polynomial Multiplication Stick Diagram

Note that the arithmetic cost function in recursive form which is given in Equation 4.5 has the solution given in Equation 4.3 under the initial condition $M(1) = 1M$. This shows us that there is not any difference in terms of arithmetic complexity under different way of implementation of the same naive approach. However, this is not the case in the delay complexity.

4.2.2 Delay Complexity

Hardware implementation block scheme is presented in Figure 4.2.

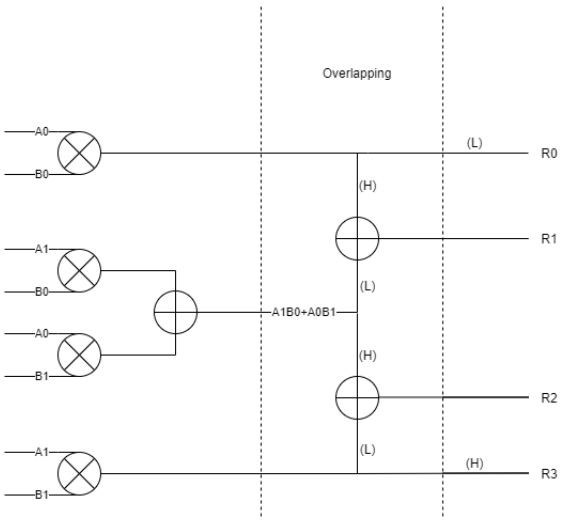

Figure 4.2: POLY-SB-2 Hardware Implementation Block Scheme

Figure 4.2 yields the following delay complexities.

$$\begin{aligned} D_{\otimes}(n) &= D_{\otimes}(n/2) & D_{\otimes}(1) &= D_m \\ D_{\oplus}(n) &= D_{\oplus}(n/2) + 2D_d & D_{\oplus}(1) &= 0 \\ D(n) &= D_{\otimes}(n) + D_{\oplus}(n) & & \\ D(n) &= D(n/2) + 2D_d & D(1) &= D_m \end{aligned} \tag{4.6}$$

In contrast to the arithmetic complexity, we see that the delay complexity of iterative approach is more efficient than the recursive approach. Even if they were be the same, recursive approaches have some disadvantages in reconfigurable hardware implementations having synchronous logic (i.e. synchronized by a predefined clock signal). See [26]. The reason stems

from the fact that recursive implementations are restricted to the frequency of the clock signal. Each iteration of the recursive implementation must wait for the next clock edge of interest. On the other hand, we do not care about implementation details of a specific environment but rather provide a general insight and analysis approach for efficient multiplication of various construction. Therefore, all remaining constructions are analyzed by a recursive approach regardless of those concerns in a real implementation. Reader is free to implement them by iterative, recursive or combination of both; depending on the desired specification.

4.3 Karatsuba 2 Way (POLY-KB-2)

This famous approach is found by Karatsuba decreasing the number of multiplication count (See [19]). The methodology can be extended to various forms of multiplication such as integer, polynomial, and even matrix-vector product, which we had shown in previous sections in the context of TMVP. The construction is given below.

$$a(x) = A_0 + A_1 x^{n/2}$$
$$b(x) = B_0 + B_1 x^{n/2}$$
$$c(x) = a(x)b(x) = A_0B_0 + [(A_0 + A_1)(B_0 + B_1) - A_0B_0 - A_1B_1]x^{n/2} + A_1B_1x^n$$

4.4 Arithmetic Complexity

Karatsuba in [19] achieves decreasing the number of multiplication to three instead of four in every recursive iteration of the problem. We have n addition of single word A_s, half of which comes from $A_0 + A_1$, and the remaining half comes from the summation of $B_0 + B_1$. Finally, we have $3n - 4$ double word additions A_d during the construction which gives the following arithmetic complexity.

$$M(n) = 3M(n/2) + (3n - 4)A_d + nA_s \qquad M(1) = 1M \tag{4.7}$$

4.5 Delay Complexity

Denoting $C_0 = A_0B_0$, $C_1 = (A_0 + A_1)(B_0 + B_1) - A_0B_0 - A_1B_1$ and $C_2 = A_1B_1$, stick diagram given in Figure 4.1 demonstrates the overlapping terms. Figure 4.3 depicts the delay computation as follow.

Until the overlapping stage, there are two possible ways from which we need to choose the one having the bigger delay. (See Lemma 2.2.2.) Examining the block scheme given in Figure 4.3, branch starting with multiplication first, introduces an addition delay of the double term

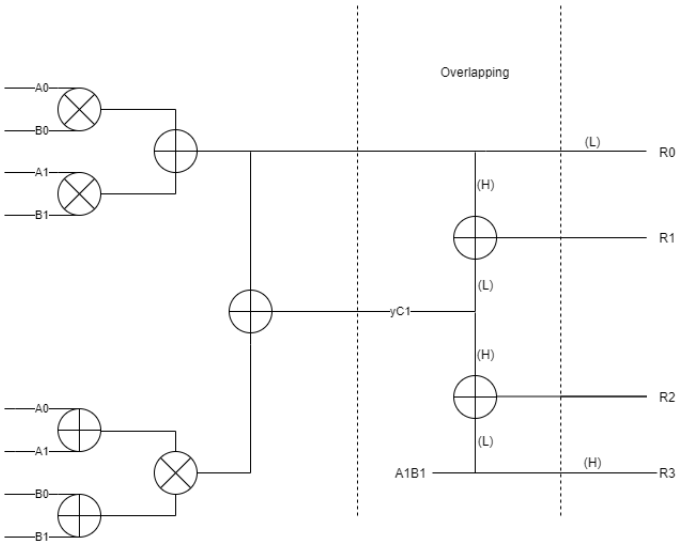

Figure 4.3: POLY-KB-2 Hardware Implementation Block Scheme

since its inputs are the result of two single term products. On the other hand, the below branch starts with a single summation delay, then leads to the multiplication block. Since the delay of the double addition process is not less than a single addition, we consider the delay coming from the upper branch. We see that the additional cascaded summation gate just before the overlapping stage. The final stage is the overlapping stage contributing a final delay of double addition, which we can express the overall delay cost as shown below.

$$\begin{aligned} D_{\otimes}(n) &= D_{\otimes}(n/2) & D_{\otimes}(1) &= D_m \\ D_{\oplus}(n) &= D_{\oplus}(n/2) + 3D_d & D_{\oplus}(1) &= 0 \\ D(n) &= D_{\otimes}(n) + D_{\oplus}(n) & & \\ D(n) &= D(n/2) + 3D_d & D(1) &= D_m \end{aligned} \tag{4.8}$$

4.6 Refined Karatsuba 2 Way (POLY-RKB-2)

This approach is given in [5], which is a variation of the original karatsuba algorithm decreasing the number of arithmetic operations.

$$\begin{aligned} a(x) &= A_0 + A_1 x^{n/2} \\ b(x) &= B_0 + B_1 x^{n/2} \\ c(x) = a(x)b(x) &= (x^{n/2} - 1)(x^{n/2} P_3 - P_1) + P_2 x^{n/2} \end{aligned}$$

where the substitutions for P_1, P_2 and P_3 as follow,

$$P_1 = A_0 B_0$$
$$P_2 = (A_0 + A_1)(B_0 + B_1)$$
$$P_3 = A_1 B_1$$

Stick diagram showing the overlapping terms is given in Figure 4.4.

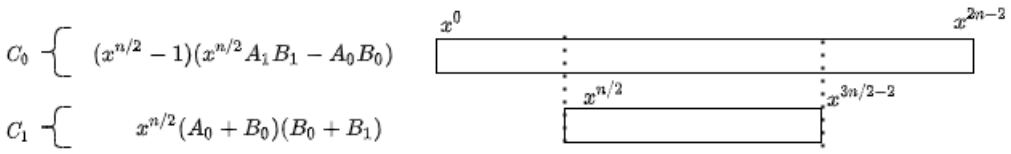

Figure 4.4: POLY-RKB2 Stick Diagram

4.6.1 Arithmetic Complexity

Examining the 4.4, we have $\frac{n}{2} - 1 + (n-1) = \frac{3n}{2} - 2$ addition of double word A_d during the construction of upper stick $C_0 = (x^{n/2} - 1)(x^{n/2}A_1B_1 - A_0B_0)$. Final overlapping summation of C_0 and C_1 brings $n-1$ addition of double A_d more. Including the n addition of single A_s coming from $A_0 + A_1$, $B_0 + B_1$; we get the following arithmetic complexity.

$$M(n) = 3M(n/2) + (2.5n - 3)A_d + nA_s \qquad M(1) = 1M \tag{4.9}$$

4.6.2 Delay Complexity

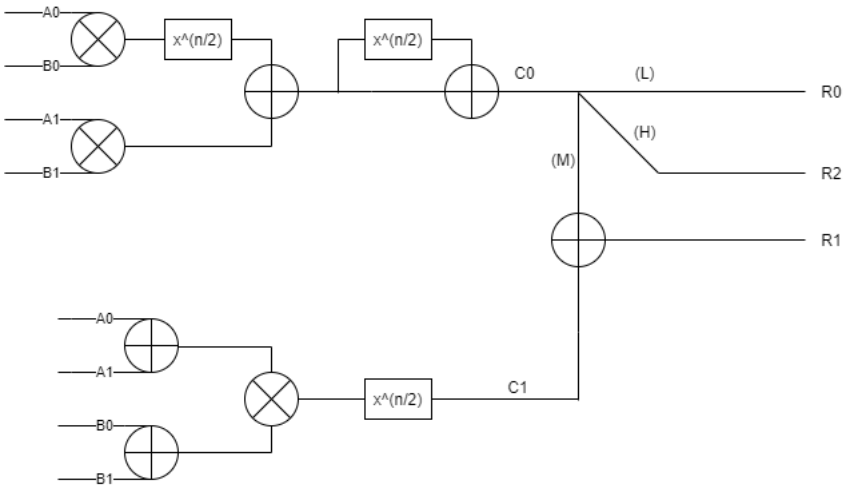

Figure 4.5: POLY-RKB-2 Hardware Implementation Block Scheme

In Figure 4.5, shifts are also demonstrated in the construction, however we are going to omit them during the cost calculation.

$$D(n) = D(n/2) + 3D_d \qquad D(1) = D_{\otimes} \tag{4.10}$$

4.7 Optimized Karatsuba 2 Way (POLY-OKB-2)

Another variation of the Karatsuba algorithm is constructed in [23]. The constructions summary is given below.

$$\begin{gathered} a(x) = A_0 + A_1 x^{n/2} \\ b(x) = B_0 + B_1 x^{n/2} \\ c(x) = a(x)b(x) = P_{1L} + x^{n/2}(P_{1H} + P_{2L} - P_{1L} - P_{3L}) \\ +x^{n}(P_{2H} - P_{1H} - P_{3H} + P_{3L}) + x^{3n/2}P_{3H} \end{gathered}$$

where the substitutions for P_1, P_2 and P_3 are given as follow,

$$\begin{gathered} P_1 = A_0 B_0 \\ P_2 = (A_0 + A_1)(B_0 + B_1) \\ P_3 = A_1 B_1 \end{gathered}$$

Note that sub-index notation L and H correspond to lower and higher degree decomposition of a given substitution. Stick diagram and hardware block scheme are given in Figures 4.6, 4.7 respectively.

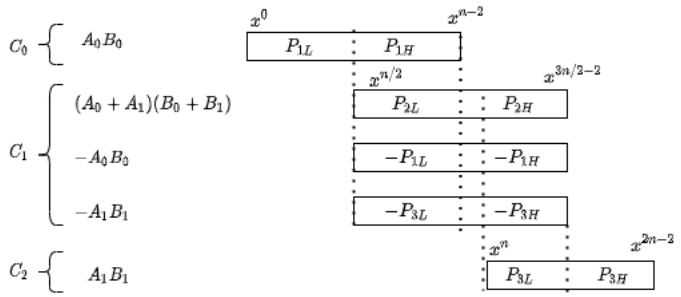

Figure 4.6: POLY-OKB2 Stick Diagram

Substitutions used in stick diagram are as follow.

$$\begin{gathered} P_1 = A_0 B_0 \\ P_2 = (A_0 + A_1)(B_0 + B_1) \\ P_3 = A_1 B_1 \end{gathered}$$

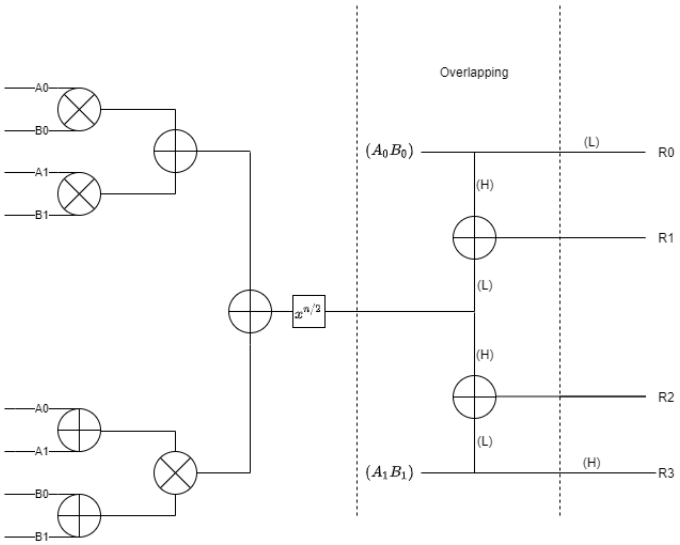

Figure 4.7: POLY-OKB-2 Hardware Implementation Block Scheme

4.7.1 Arithmetic Complexity

Examining the substitutions, and the overlapping parts from Figure 4.6, we conclude that there is not any change in terms of the number of arithmetic operations compared to the 'Refined Karatsuba' construction given in 4.6.

$$M(n) = 3M(n/2) + (2.5n - 3)A_d + nA_s \qquad M(1) = 1M \tag{4.11}$$

4.7.2 Delay Complexity

Examining the Figure 4.7 results in the following delay cost.

$$D(n) = D(n/2) + 3D_d \qquad D(1) = D_m \tag{4.12}$$

4.8 Schoolbook 3 Way (POLY-SB-3)

This approach recursively realizes the naive multiplication by dividing the polynomials into three smaller ones and calling the original problem with smaller input sizes. Let us multiply $a(x)$ $b(x)$, but before divide them into $A_0, A_1, A_2, B_0, B_1, B_2$ as follow. Note that we are going to use the same notation for the rest of all 3-way multiplication algorithms.

$$a(x) = A_0 + A_1x^{n/3} + A_2x^{2n/3}$$
$$b(x) = B_0 + B_1x^{n/3} + B_2x^{2n/3}$$
$$c(x) = a(x)b(x)$$

$$A_0 = a_0 + a_1x + \ldots + a_{\frac{n}{3}-1}x^{\frac{n}{3}-1}$$
$$A_1 = a_{\frac{n}{3}} + a_{\frac{n}{3}+1}x + \ldots + a_{\frac{2n}{3}-1}x^{\frac{n}{3}-1}$$
$$A_2 = a_{\frac{2n}{3}} + a_{\frac{2n}{3}+1}x + \ldots + a_{\frac{3n}{3}-1}x^{\frac{n}{3}-1}$$
$$B_0 = b_0 + b_1x + \ldots + b_{\frac{n}{3}-1}x^{\frac{n}{3}-1}$$
$$B_1 = b_{\frac{n}{3}} + b_{\frac{n}{3}+1}x + \ldots + b_{\frac{2n}{3}-1}x^{\frac{n}{3}-1}$$
$$B_2 = b_{\frac{2n}{3}} + b_{\frac{2n}{3}+1}x + \ldots + b_{\frac{3n}{3}-1}x^{\frac{n}{3}-1}$$
$$c(x) = A_0B_0 + (A_0B_1 + A_1B_0)x^{\frac{n}{3}} + (A_0B_2 + A_1B_1 + A_2B_0)x^{\frac{2n}{3}}$$
$$+(A_1B_2 + A_2B_1)x^{\frac{3n}{3}} + A_2B_2x^{\frac{4n}{3}}$$

Denoting $C_0 = A_0B_0$, $C_1 = A_0B_1 + A_1B_0$, $C_2 = A_0B_2 + A_1B_1 + A_2B_0$, $C_3 = A_1B_2 + A_2B_1$ and $C_4 = A_2B_2$, we can see the overlapping terms from the stick diagram given in Figure 4.8. Moreover, hardware implementation block scheme is presented in Figure 4.9.

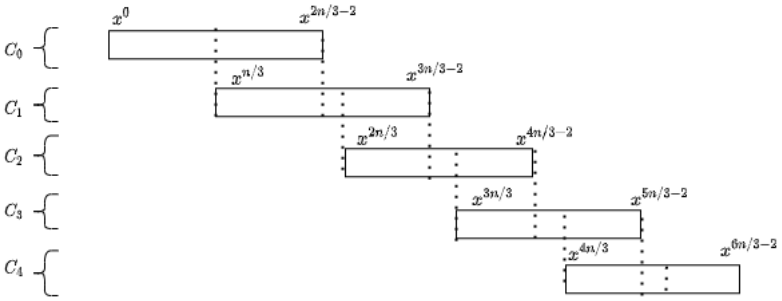

Figure 4.8: 3-Way Polynomial Multiplication Stick Diagram

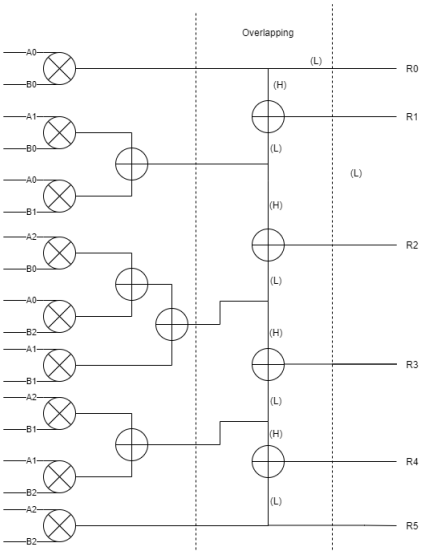

Figure 4.9: POLY-SB-3 Hardware Implementation Block Scheme

4.8.1 Arithmetic Complexity

Recalling our substitutions given by C_i, we see that the number of addition is $2 - |i-2|$ where $0 \leq i \leq 4$, at each of which $2n/3 - 1$ elements are added; which can be seen also from Figure 4.8. We can write those additions from substitutions C_is as below

$$M_{\bigoplus,C_i}(n) = \sum_{i=0}^{4}(2-|i-2|)(\frac{2n}{3}-1)$$
$$M_{\bigoplus,C_i}(n) = 4(\frac{2n}{3}-1)A_d \tag{4.13}$$

Counting the overlaps from Figure 4.8, for five substitution terms C_i, there are four overlapping terms, each of which consists of $\frac{n}{3} - 1$ elements. We can represent the number of additions coming from the overlapping terms as $M_{\bigoplus,overlap}(n)$.

$$M_{\bigoplus,overlap}(n) = 4(\frac{n}{3}-1))A_d \tag{4.14}$$

Combining Equations 4.13, 4.14 gives the overall number of additions. Note that problem calls itself with 9 smaller instances of size divided by 3, and all terms are double word.

$$\begin{aligned} M_{\bigoplus}(n) = 9M_{\bigoplus}(n/3) + M_{\bigoplus,C_i}(n) + M_{\bigoplus,overlap}(n) \quad & M_{\bigoplus}(1) = 0 \\ M_{\bigotimes}(n) = 9M_{\bigotimes}(n/3) \quad & M_{\bigotimes}(1) = 1 \\ M(n) = M_{\bigoplus}(n) + M_{\bigotimes}(n) \quad & M(1) = M_{\bigoplus}(1) + M_{\bigotimes}(1) \\ M(n) = 9M(n/3) + (4n-8)A_d \quad & M(1) = 1 \end{aligned} \tag{4.15}$$

4.8.2 Delay Complexity

Examining below block scheme given in Figure 4.9 leads to the following delay cost results.

$$D(n) = D(n/2) + 3D_d \qquad D(1) = D_m \tag{4.16}$$

4.9 Karatsuba 3 Way (POLY-KB-3)

This approach adapts the Karatsuba approach, where the split size is three rather than two. In below, we give the following substitutions C_1,C_2, and C_3 for the sake of simplicity in the hardware block representation.

$$C_1 = (A_0 + A_1)(B_0 + B_1) - A_0B_0 - A_1B_1$$
$$C_2 = (A_0 + A_2)(B_0 + B_2) - A_0B_0 - A_2B_2 + A_1B_1$$
$$C_3 = (A_1 + A_2)(B_1 + B_2) - A_1B_1 - A_2B_2$$

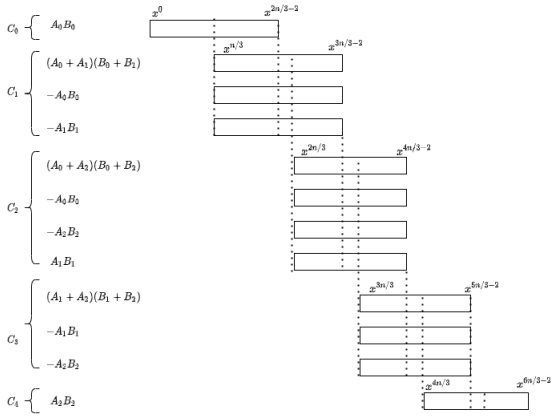

Figure 4.10: POLY-KB-3 Stick Diagram

4.9.1 Arithmetic Complexity

We can count the number of double word additions coming from the construction as C_is.

$$M_{\bigoplus,C_i}(n) = 7(\frac{2n}{3} - 1)A_d \tag{4.17}$$

We can count the overlaps between C_i terms as below.

$$M_{\bigoplus,overlap}(n) = 4(\frac{n}{3} - 1)A_d \tag{4.18}$$

There are six additions of single word, three of which comes from $A_i + A_j$ and the other three ones from $B_i + B_j$. Each summation consists of $n/3$ elements since the size of splitted polynomials A_i, B_j are $n/3$. If we denote those additions $M_{\bigoplus,single}(n)$, we get the following.

$$M_{\bigoplus,single}(n) = 4(\frac{n}{3} - 1)A_d \tag{4.19}$$

Finally, we can combine all Equations 4.17, 4.18 and 4.19 to find out the final arithmetic complexity result as below. Note that there are 6 multiplications called in every recursion step instead of 9, which is decent improvement compared to naive schoolbook split constructions.

$$M(n) = 6M(\frac{n}{3}) + (6n - 11)A_d + 2nA_s \qquad M(1) = 1 \tag{4.20}$$

4.9.2 Delay Complexity

Figure 4.10 summarizes overlapping terms. Until this overlapping stage, we already compute C_1,C_2, and C_3 in the construction stage, which is seen at the left side of the Figure 4.11. We have the maximum delay path contributed by two double and one single addition stage, to which a final delay of one double addition stage is added.

$$D(n) = D(n/3) + 3D_d + D_s \qquad D(1) = D_m \tag{4.21}$$

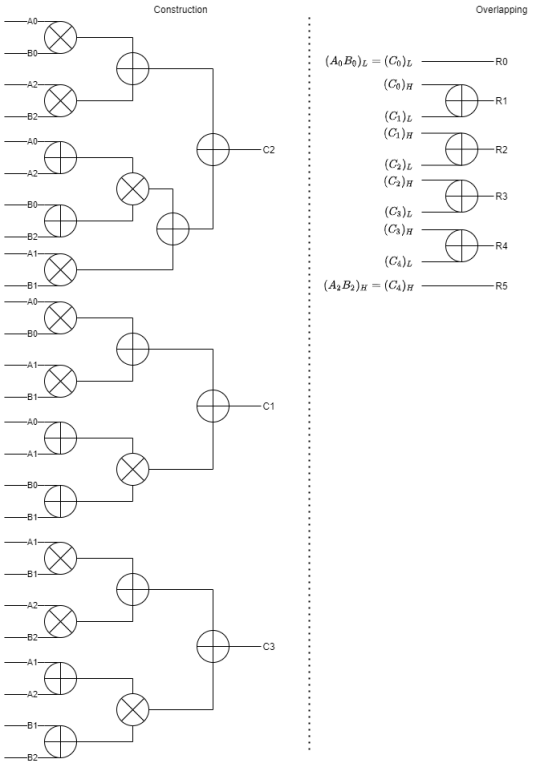

Figure 4.11: POLY-KB-3 Hardware Implementation Block Scheme

4.10 Optimized Karatsuba 3 Way (POLY-OKB-3)

Ilter in [17] express another variation of Karatsuba 3-Way multiplication in order to reduce the arithmetic cost further. Consider the following substitutions.

$$P_1 = A_0B_0 \tag{4.22}$$
$$P_2 = (A_0 + A_1)(B_0 + B_1) \tag{4.23}$$
$$P_3 = A_1B_1 \tag{4.24}$$
$$P_4 = (A_0 + A_2)(B_0 + B_2) \tag{4.25}$$
$$P_5 = A_2B_2 \tag{4.26}$$
$$P_6 = (A_1 + A_2)(B_1 + B_2) \tag{4.27}$$

$$a(x) = A_0 + A_1x^{n/2}$$
$$b(x) = B_0 + B_1x^{n/2}$$
$$c(x) = a(x)b(x) = P_{1L} + x^{n/3}(P_{1H} + P_{2L} - P_{1L} - P_{3L}) \tag{4.28}$$
$$+x^{2n/3}(P_{2H} - P_{1H} - P_{3H} + P_{4L} - P_{1L} - P_{5L} + P_{3L}) \tag{4.29}$$
$$+x^{3n/3}(P_{4H} - P_{1H} - P_{5H} + P_{3H} + P_{6L} - P_{5L} - P_{3L}) \tag{4.30}$$
$$+x^{4n/3}(P_{6H} - P_{3H} - P_{5H} + P_{5L} + x^{5n/3}P_{5H} \tag{4.31}$$

4.10.1 Arithmetic Complexity

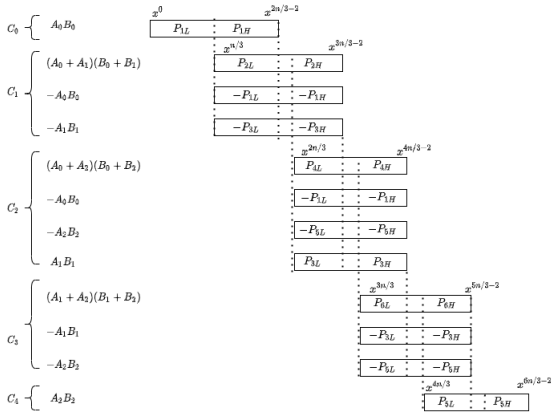

Figure 4.12: POLY-OKB-3 Stick Diagram

Although the stick diagram given in Figure 4.12 seems identical to the previous one given in Figure 4.10, the current one 4.12 representing the "Optimized Karatsuba 3" (i.e. POLY-OKB-3) includes substitutions denoted by either P_{iH} or P_{iL} which corresponds to higher and lower decomposition of substitutions given in Equations from 4.22 to 4.27. The main motivation of these substitution is to decrease the number of arithmetic operations while preserving the same result. Thanks to these substitutions, Ilter in [17] achieves to create some twin additions to be discarded during the construction. When we count the arithmetic operations, we see $6(\frac{n}{3}) = 2n$ addition of single word A_s coming from those additions of type $A_i + A_j$, $B_i + B_j$, seen between the Equations from 4.22 to 4.27. Furthermore, there are $\frac{16n}{3} - 9$ double word additions during the construction which leads to the overall arithmetic complexity as follow.

$$M(n) = 6M(\frac{n}{3}) + (\frac{16n}{3} - 9)A_d + 2nA_s \qquad M(1) = 1 \tag{4.32}$$

4.10.2 Delay Complexity

The stick diagram given below in Figure 4.12 shows not only the overlapping stage but also the overall multiplication scheme summary. During the substitution stage, which is described by the Equations from 4.22 to 4.27, there is one addition of single. At the overlapping stage, we see that the maximum number of block segments is 7, each consisting of double addition. Those 7 overlapping additions can be handled via $\lceil log_2(7) \rceil = 3$ cascaded stage of double addition.

$$D(n) = D(n/3) + 3D_d + D_s \qquad D(1) = D_m \tag{4.33}$$

CHAPTER 5

HYBRID MULTIPLICATION APPROACH

Throughout the previous chapters, we have seen that there are many possible constructions for realizing polynomial multiplication in $\mathbb{Z}[x]/\langle x^n \pm 1\rangle$. Regardless of the methodology, whether it is a TMVP or not, it has been shown that hybrid use of these constructions yields efficient multiplications employing less arithmetic operation than the one at which a single multiplication algorithm is preferred for all sizes. (See [25],[27],[11],[6]). This chapter starts explaining the hybrid approach and illustrates how it outperforms the static multiplication approaches by a simple example. Afterward, we share the details of our hybrid approach algorithm and explain how we generate results for finding the best possible sequence under some specific constraints.

5.1 Illustration of Hybrid Approach against Static Multiplication

Let us consider that a single algorithm is used for multiplication for a specific size. In that case, one should choose an algorithm for a given input size and divide the size by the split size of chosen algorithm permanently until a non-divisible size is obtained. When this is the case, one should pad the size to a divisible one and continue the process until the multiplication is done. On the other hand, this is not the case for hybrid approach, in which one can choose different split sizes during the multiplication process. Let us give the following examples for clarification. In below, Example 5.1.1 calculates the number of total arithmetic operations for static choice of 'TMVP-STD-2' (See Section 3.3). Example 5.1.2 does the same task for static choice of 'TMVP-SB'. (See Section 3.2). The final Example 5.1.3 shows how the hybrid approach supersede and outperforms the static usage of algorithms by combining the 'TMVP-STD-2' and 'TMVP-SB'.

Example 5.1.1. *Let us assume that we implement a TMVP, and we want to see how the hybrid approach affects the number of arithmetic operations. For this purpose, we choose an input size $n = 10$ and want to try the classical approach at which 'TMVP-STD-2' as a single choice of algorithm. As it was previously stated, we should permanently call 'TMVP-STD-2' and make the zero-padding when the size is not suitable.*

$$
\begin{aligned}
M(n) &= 3M(n/2) + (2n-1)A_s + nA_d && \text{TMVP-STD-2} \\
M(10) &= 3M(5) + 19A_s + 10A_d && \\
M(5) &\rightarrow M(6) && \text{zero-padding} \\
M(6) &= 3M(3) + 11A_s + 6A_d && \\
M(3) &\rightarrow M(4) && \text{zero-padding} \\
M(4) &= 3M(2) + 7A_s + 4A_d && \\
M(2) &= 3M(1) + 3A_s + 2A_d && \text{zero-padding} \\
M(1) &= 1M && \text{Initial condition}
\end{aligned}
$$

Below shows the overall number of arithmetic operation.

$$M_{static-tmvp-std-2}(10) = 81M + 196A_s + 118A_s$$

If we assign equal 'weight' to each of cost metrics M, A_s, A_d, we get the following total number of operation.

$$M_{static-tmvp-std-2}(10) = 395 \qquad M = A_s = A_d$$

Example 5.1.2. *Let us repeat the static approach with the algorithm choice 'TMVP-SB'.*

Note that schoolbook construction does not need any padding operation in case of iterative implementation, and its arithmetic complexity is given by $M_{TMVP-SB}(n) = n^2M + n(n-1)A_d$. On the other hand, a recursive version of the schoolbook approach needs some zero padding depending on the split size of the implementation; the result would not change. See Section 3.2.1 for the recursive approach of split size is 2.

$$
\begin{aligned}
M_{static-tmvp-sb}(10) &= 100M + 90A_d \\
M_{static-tmvp-sb}(10) &= 190 \qquad M = A_s = A_d
\end{aligned}
$$

Example 5.1.3. *Finally, we are ready to show hybrid approach by combining 'TMVP-SB' and 'TMVP-STD-2'*

If we start multiplication 'TMVP-STD-2', and continue with 'TMVP-SB', we get the following number of arithmetic operations.

$$
\begin{aligned}
M(10) &= 3M(5) + 19A_s + 10A_d && \text{TMVP-STD-2} \\
M(5) &= 25M + 20A_d && \text{TMVP-SB} \\
M_{hybrid}(10) &= 164 && M = A_d = A_s
\end{aligned}
$$

It is certain that hybrid approach outperforms the static usage of algorithms in terms of number of arithmetic operations during the multiplication process.

5.2 Hybrid Analysis and Finding the Best Algorithm Sequence

As the input size of a multiplication increases, the number of possible algorithm choices in the hybrid approach increases. Furthermore, the best possible sequence optimizing the target cost might change under different 'weightings' of cost metrics. In the previous Section 5.1, we have assumed equal weightings for each cost metric to optimize 'Arithmetic' cost. We did not even try different cost metric weightings and did not talk about any other type to optimize, such as 'Delay Cost'.

In our thesis, we make the hybrid analysis approach as generic as possible. We take a set of multiplication algorithms in which there are both arithmetic and delay cost functions for each algorithm defined. We set cost weightings to each set of cost metric (See Tables 2.1 and 2.2) for the chosen target cost type. Finally, before starting analysis, we decide if zero-padding is allowed or not among the algorithms we compare together with a predefined initial condition for those recursive cost functions. Input definitions are summarized in below Table 5.1.

Table 5.1: Input Definitions for Hybrid Analysis Approach

Input Definition	**Possible Values**
Algorithm Container	TMVP [24], Modular Polynomial Multiplications [17]
Target Cost to Optimize	Arithmetic, Delay, Arithmetic x Delay
Cost Metric Set (Weightings)	(M, A_d, A_s) for Arithmetic Cost (D_m, D_d, D_s) for Delay Cost
Padding Choice	Padding Allowed, No-Padding
Initial Condition	$M(1) = M$ for Arithmetic Cost $D(1) = D_m$ for Delay Cost

We describe our hybrid analysis approach from an algorithmic point of view as stated in Algorithm 1. Shortly, we calculate all possible algorithm costs for each size i and save the minimum target cost until we reach the original input size of the algorithm. Depending on the choice of 'Padding Choice' given in Table 5.1, we either ignore and skip algorithms whose split size does not divide the current size of the algorithm i (See steps 7-11 in Algorithm 1), or include them by temporarily increasing the size (i.e., zero padding, See steps 12-22 in Algorithm 1) to try them.

Algorithm 1 Finding the Best Sequence, Hybrid Analysis

Require: Table 5.1, $n \leftarrow$ input size

Ensure: $bestPossibleAlgorithmList \leftarrow$ Best Possible Algorithm Sequence Optimizing the Target Cost

1: $i \leftarrow 2$
2: $minCostAlgorithm \leftarrow algorithmContainer[0]$
3: $minCost \leftarrow minCostAlgorithm.TargetCostFunction(i)$
4: **while** $i \leq n$ **do**
5: **for each** $currentAlgorithm$ **in** $algorithmContainer$ **do**
6: $currentCost \leftarrow currentAlgorithm.TargetCostFunction(i)$
7: **if** $paddingOption == NoPadding$ **then**
8: (Check if the size i is not divisible by $splitSize$ of $currentAlgorithm$)
9: **if** $splitSize \nmid i$ **then**
10: Update $currentAlgorithm$, Continue Loop (Go to line: 5)
11: **end if**
12: **else if** $paddingOption == PaddingAllowed$ **then**
13: (Check if the size i is not divisible by $splitSize$ of $currentAlgorithm$)
14: **if** $splitSize \nmid i$ **then**
15: (Find the suitable size)
16: $paddedSize \leftarrow i$
17: **while** $splitSize \nmid paddedSize$ **do**
18: $paddedSize \leftarrow paddedSize + 1$
19: **end while**
20: $currentCost \leftarrow currentAlgorithm.TargetCostFunction(paddedSize)$
21: **end if**
22: **end if**
23: **if** $currentCost < minCost$ **then**
24: $minCostAlgorithm \leftarrow currentAlgorithm$
25: $minCost \leftarrow currentCost$
26: $bestPossibleAlgorithmList.Append(minCostAlgorithm)$
27: Print($minCost$)
28: **end if**
29: **end for**
30: $i \leftarrow i + 1$
31: **end while**
32: output $bestPossibleAlgorithmList$

5.3 Software Tool for Hybrid Analysis

We have designed a software tool using C# to realize, visualize and report the task of finding the best algorithm sequence under the constraints given in Table 5.1. This tool realizes the Algorithm 1 which we proposed in the previous Section 5.2. We share a partial set of results in Chapter 6, and an extended version at Appendix A.

CHAPTER 6

RESULTS, COMPARISON AND ANALYSIS

6.1 Arithmetic and Delay Complexity Summaries

Before sharing the hybrid iteration results, let us summarize all delay complexities of TMVP together with its arithmetic costs shared by [24].

Table 6.1: Delay Cost Functions of various TMVP constructions derived from [24]

Algorithm Name	Delay Cost Function
TMVP-SB-ITR	$D(n) = D(n/2) + D_d$
TMVP-SB-REC	$D(n) = D(n/2) + D_d$
TMVP-STD-2	$D(n) = D(n/2) + D_s + D_d$
TMVP-STD-3	$D(n) = D(n/3) + D_s + 2D_d$
TMVP-3	$D(n) = D(n/3) + 2D_s + 2D_d$
TMVP-4	$D(n) = D(n/4) + 3D_s + 3D_d$
TMVP-UNB	$max\{D(k), D(n-k), D_{TMVP-SB}(max\{k, n-k\})\} + D_d$
TMVP-LT	$max\{D_m, D(n-1), D_{TMVP-SB}(n-1)\} + D_d$

Table 6.2: Arithmetic Cost Functions of various TMVP constructions derived from [24]

Algorithm Name	Arithmetic (Space) Cost Function
TMVP-SB-ITR	$M(n) = (n^2 - n)A_d + n^2M$
TMVP-SB-REC	$M(n) = (n^2 - n)A_d + n^2M$
TMVP-STD-2	$M(n) = 3M(n/2) + (2n-1)A_s + nA_d$
TMVP-STD-3	$M(n) = 6M(n/3) + (3n-1)A_s + (2n)A_d$
TMVP-3	$M(n) = 5M(n/3) + (\frac{22n}{3} - 8)A_s + \frac{7n}{3}A_d$
TMVP-4	$M(n) = 5M(n/4) + (\frac{65n}{4} - 27)A_s + \frac{7n}{2}A_d$
TMVP-UNB	$M(n,k) = M(n-k) + M(k) + 2k(n-k)A_d + 2k(n-k)M$
TMVP-LT	$M(n) = M(n-1) + (2n-2)A_d + (2n-1)M$

In order to make a comparison with polynomial multiplication constructions given in [17], we share cost functions in the thesis.

Table 6.3: Recursive Delay and Arithmetic Cost Functions of Polynomial Multiplication Constructions in [17]

Algorithm Name	Delay Cost Function	Arithmetic (Space) Cost Function
POLY-SB-ITR	$D(n) = D(n/2) + D_d$	$M(n) = n^2M + (n-1)^2A_d$
POLY-SB-2	$D(n) = D(n/2) + 2D_d$	$M(n) = 4M(n/2) + (2n-3)A_d$
POLY-KB-2	$D(n) = D(n/2) + 3D_d$	$M(n) = 3M(n/2) + (3n-4)A_d + nA_s$
POLY-RKB-2	$D(n) = D(n/2) + 3D_d$	$M(n) = 3M(n/2) + (2.5n-3)A_d + nA_s$
POLY-OKB-2	$D(n) = D(n/2) + 3D_d$	$M(n) = 3M(n/2) + (2.5n-3)A_d + nA_s$
POLY-SB-3	$D(n) = D(n/3) + 3D_d$	$M(n) = 9M(n/3) + (4n-8)A_d$
POLY-KB-3	$D(n) = D(n/3) + 3D_d + D_s$	$M(n) = 6M(n/3) + (6n-11)A_d + 2nA_s$
POLY-OKB-3	$D(n) = D(n/3) + 3D_d + D_s$	$M(n) = 6M(n/3) + (\frac{16}{3}n-9)A_d + 2nA_s$

Note that the costs shown in Table 6.3 is calculated without any modular reduction, which would bring an overhead to the existing cost. Recall that our quotient polynomial is $x^n \pm 1$ which is equivalently replace the terms higher than x^n with $x^n = \pm 1$. This yields an n double addition indicated by nA_d in arithmetic complexity. In the case of hardware implementation, this would yield a single additional stage of double summation blocks, denoted by D_d. Therefore, we can update the Table 6.3 constructed from [17] as showed in Table 6.4 according to the modular polynomial multiplication scenario where the aforementioned reduction step is an overhead.

Table 6.4: Recursive Delay and Arithmetic Cost Functions of Modular Polynomial Multiplications adapted by [17]

Algorithm Name	Delay Cost Function	Arithmetic (Space) Cost Function
POLY-SB-ITR	$D(n) = D(n/2) + 2D_d$	$M(n) = n^2M + (n^2-n+1)A_d$
POLY-SB-2	$D(n) = D(n/2) + 3D_d$	$M(n) = 4M(n/2) + (3n-3)A_d$
POLY-KB-2	$D(n) = D(n/2) + 4D_d$	$M(n) = 3M(n/2) + (4n-4)A_d + nA_s$
POLY-RKB-2	$D(n) = D(n/2) + 4D_d$	$M(n) = 3M(n/2) + (3.5n-3)A_d + nA_s$
POLY-OKB-2	$D(n) = D(n/2) + 4D_d$	$M(n) = 3M(n/2) + (3.5n-3)A_d + nA_s$
POLY-SB-3	$D(n) = D(n/3) + 4D_d$	$M(n) = 9M(n/3) + (5n-8)A_d$
POLY-KB-3	$D(n) = D(n/3) + 4D_d + D_s$	$M(n) = 6M(n/3) + (7n-11)A_d + 2nA_s$
POLY-OKB-3	$D(n) = D(n/3) + 4D_d + D_s$	$M(n) = 6M(n/3) + (\frac{19}{3}n-9)A_d + 2nA_s$

6.2 Hybrid Multiplication Results

In this section, we share the results of the hybrid multiplication approach for both modular polynomial multiplication and TMVP. We scan all target costs given in Table 5.1 with the choice of padding allowed and equal-cost metric weights. We use 'MPM' as abbreviation for 'Modular Polynomial Multiplication' (i.e, Polynomial Multiplication over $\mathbb{Z}[x]/\langle x^n \pm 1\rangle$) ; and 'TMVP' stands for 'Toeplitz Matrix Vector Product'.

Table 6.5: Hybrid Multiplication Lookup Table

Table Index	Alg. Container	Optimized Cost	Space Weightings	Delay Weightings	Padding Choice
Table 6.6	MPM	Space	$M = A_s = A_d$	$D_m = D_s = D_d$	Allowed
Table 6.7	MPM	Delay	$M = A_s = A_d$	$D_m = D_s = D_d$	Allowed
Table 6.8	MPM	Space x Delay	$M = A_s = A_d$	$D_m = D_s = D_d$	Allowed
Table 6.9	TMVP	Space	$M = A_s = A_d$	$D_m = D_s = D_d$	Allowed
Table 6.10	TMVP	Delay	$M = A_s = A_d$	$D_m = D_s = D_d$	Allowed
Table 6.11	TMVP	Space x Delay	$M = A_s = A_d$	$D_m = D_s = D_d$	Allowed

Table 6.6: Polynomial Multiplication Space Optimization Result (Padding Allowed, $M = A_s = A_d$, $D_m = D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	1	0	0	1	1	0	0	1	1	Initial Condition
2	4	0	3	7	1	0	2	3	21	SB
3	9	0	7	16	1	0	4	5	80	SB
4	16	0	13	29	1	0	4	5	145	SB
5	25	0	21	46	1	0	6	7	322	SB
6	36	0	31	67	1	0	6	7	469	SB
7	49	0	43	92	1	0	6	7	644	SB
8	48	8	64	120	1	0	8	9	1080	RKB2
9	81	0	73	154	1	0	8	9	1386	SB
10	75	10	95	180	1	0	10	11	1980	RKB2
11	121	0	111	232	1	0	8	9	2088	SB
12	108	12	132	252	1	0	10	11	2772	RKB2
13	169	0	157	326	1	0	8	9	2934	SB
14	147	14	175	336	1	0	10	11	3696	RKB2
15	150	30	212	392	1	1	10	12	4704	O3W
16	144	40	245	429	1	0	12	13	5577	RKB2
17	243	18	279	540	1	0	12	13	7020	RKB2
18	243	18	279	540	1	0	12	13	7020	RKB2
19	225	50	352	627	1	0	14	15	9405	RKB2
20	225	50	352	627	1	0	14	15	9405	RKB2
21	294	42	382	718	1	1	10	12	8616	O3W
22	363	22	407	792	1	0	12	13	10296	RKB2
23	324	60	477	861	1	0	14	15	12915	RKB2
24	324	60	477	861	1	0	14	15	12915	RKB2
25	507	26	559	1092	1	0	12	13	14196	RKB2
26	507	26	559	1092	1	0	12	13	14196	RKB2
27	441	70	620	1131	1	0	14	15	16965	RKB2
28	441	70	620	1131	1	0	14	15	16965	RKB2
29	450	120	738	1308	1	1	14	16	20928	RKB2
30	450	120	738	1308	1	1	14	16	20928	RKB2
31	432	152	844	1428	1	0	16	17	24276	RKB2
32	432	152	844	1428	1	0	16	17	24276	RKB2

Table 6.6 - continued from previous page

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
33	726	66	866	1658	1	1	12	14	23212	O3W
34	729	88	953	1770	1	0	16	17	30090	RKB2
35	729	90	960	1779	1	0	16	17	30243	RKB2
36	729	90	960	1779	1	0	16	17	30243	RKB2
37	675	188	1186	2049	1	0	18	19	38931	RKB2
38	675	188	1186	2049	1	0	18	19	38931	RKB2
39	675	190	1193	2058	1	0	18	19	39102	RKB2
40	675	190	1193	2058	1	0	18	19	39102	RKB2
41	882	168	1290	2340	1	1	14	16	37440	RKB2
42	882	168	1290	2340	1	1	14	16	37440	RKB2
43	1089	110	1372	2571	1	0	16	17	43707	RKB2
44	1089	110	1372	2571	1	0	16	17	43707	RKB2
45	900	270	1548	2718	1	2	14	17	46206	O3W
46	972	226	1589	2787	1	0	18	19	52953	RKB2
47	972	228	1596	2796	1	0	18	19	53124	RKB2
48	972	228	1596	2796	1	0	18	19	53124	RKB2
49	1521	128	1849	3498	1	0	16	17	59466	RKB2
50	1521	128	1849	3498	1	0	16	17	59466	RKB2
51	1521	130	1856	3507	1	0	16	17	59619	RKB2
52	1521	130	1856	3507	1	0	16	17	59619	RKB2
53	1323	264	2046	3633	1	0	18	19	69027	RKB2
54	1323	264	2046	3633	1	0	18	19	69027	RKB2
55	1323	266	2053	3642	1	0	18	19	69198	RKB2
56	1323	266	2053	3642	1	0	18	19	69198	RKB2
57	1350	418	2414	4182	1	1	18	20	83640	RKB2
58	1350	418	2414	4182	1	1	18	20	83640	RKB2
59	1350	420	2421	4191	1	1	18	20	83820	RKB2
60	1350	420	2421	4191	1	1	18	20	83820	RKB2
61	1296	518	2746	4560	1	0	20	21	95760	RKB2
62	1296	518	2746	4560	1	0	20	21	95760	RKB2
63	1296	520	2753	4569	1	0	20	21	95949	RKB2
64	1296	520	2753	4569	1	0	20	21	95949	RKB2
65	2178	264	2826	5268	1	1	16	18	94824	RKB2
66	2178	264	2826	5268	1	1	16	18	94824	RKB2
67	2187	332	3094	5613	1	0	20	21	117873	RKB2
68	2187	332	3094	5613	1	0	20	21	117873	RKB2
69	2187	340	3122	5649	1	0	20	21	118629	RKB2
70	2187	340	3122	5649	1	0	20	21	118629	RKB2
71	2187	342	3129	5658	1	0	20	21	118818	RKB2
72	2187	342	3129	5658	1	0	20	21	118818	RKB2

Table 6.6 - continued from previous page

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
73	2025	638	3814	6477	1	0	22	23	148971	RKB2
74	2025	638	3814	6477	1	0	22	23	148971	RKB2
75	2025	640	3821	6486	1	0	22	23	149178	RKB2
76	2025	640	3821	6486	1	0	22	23	149178	RKB2
77	2025	648	3849	6522	1	0	22	23	150006	RKB2
78	2025	648	3849	6522	1	0	22	23	150006	RKB2
79	2025	650	3856	6531	1	0	22	23	150213	RKB2
80	2025	650	3856	6531	1	0	22	23	150213	RKB2
81	2646	586	4154	7386	1	1	18	20	147720	RKB2
82	2646	586	4154	7386	1	1	18	20	147720	RKB2
83	2646	588	4161	7395	1	1	18	20	147900	RKB2
84	2646	588	4161	7395	1	1	18	20	147900	RKB2
85	3267	416	4414	8097	1	0	20	21	170037	RKB2
86	3267	416	4414	8097	1	0	20	21	170037	RKB2
87	3267	418	4421	8106	1	0	20	21	170226	RKB2
88	3267	418	4421	8106	1	0	20	21	170226	RKB2
89	2700	900	4956	8556	1	2	18	21	179676	RKB2
90	2700	900	4956	8556	1	2	18	21	179676	RKB2
91	2916	770	5086	8772	1	0	22	23	201756	RKB2
92	2916	770	5086	8772	1	0	22	23	201756	RKB2
93	2916	778	5114	8808	1	0	22	23	202584	RKB2
94	2916	778	5114	8808	1	0	22	23	202584	RKB2
95	2916	780	5121	8817	1	0	22	23	202791	RKB2
96	2916	780	5121	8817	1	0	22	23	202791	RKB2
97	4356	594	5814	10764	1	2	16	19	204516	O3W
98	4356	594	5814	10764	1	2	16	19	204516	O3W
99	4356	594	5814	10764	1	2	16	19	204516	O3W
100	4563	484	5894	10941	1	0	20	21	229761	RKB2
101	4563	492	5922	10977	1	0	20	21	230517	RKB2
102	4563	492	5922	10977	1	0	20	21	230517	RKB2
103	4563	494	5929	10986	1	0	20	21	230706	RKB2
104	4563	494	5929	10986	1	0	20	21	230706	RKB2
105	3969	898	6506	11373	1	0	22	23	261579	RKB2
106	3969	898	6506	11373	1	0	22	23	261579	RKB2
107	3969	900	6513	11382	1	0	22	23	261786	RKB2
108	3969	900	6513	11382	1	0	22	23	261786	RKB2
109	3969	908	6541	11418	1	0	22	23	262614	RKB2
110	3969	908	6541	11418	1	0	22	23	262614	RKB2
111	3969	910	6548	11427	1	0	22	23	262821	RKB2
112	3969	910	6548	11427	1	0	22	23	262821	RKB2

Table 6.6 - continued from previous page

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
113	4050	1368	7638	13056	1	1	22	24	313344	RKB2
114	4050	1368	7638	13056	1	1	22	24	313344	RKB2
115	4050	1370	7645	13065	1	1	22	24	313560	RKB2
116	4050	1370	7645	13065	1	1	22	24	313560	RKB2
117	4050	1378	7673	13101	1	1	22	24	314424	RKB2
118	4050	1378	7673	13101	1	1	22	24	314424	RKB2
119	4050	1380	7680	13110	1	1	22	24	314640	RKB2
120	4050	1380	7680	13110	1	1	22	24	314640	RKB2
121	3888	1676	8662	14226	1	0	24	25	355650	RKB2
122	3888	1676	8662	14226	1	0	24	25	355650	RKB2
123	3888	1678	8669	14235	1	0	24	25	355875	RKB2
124	3888	1678	8669	14235	1	0	24	25	355875	RKB2
125	3888	1686	8697	14271	1	0	24	25	356775	RKB2
126	3888	1686	8697	14271	1	0	24	25	356775	RKB2
127	3888	1688	8704	14280	1	0	24	25	357000	RKB2
128	3888	1688	8704	14280	1	0	24	25	357000	RKB2

Table 6.7: Polynomial Multiplication Delay Optimization Result (Padding Allowed, $M = A_s = A_d$, $D_m = D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	1	0	0	1	1	0	0	1	1	Initial Condition
2	4	0	3	7	1	0	2	3	21	SB
3	9	0	7	16	1	0	4	5	80	SB
4	16	0	13	29	1	0	4	5	145	SB
5	25	0	21	46	1	0	6	7	322	SB
6	36	0	31	67	1	0	6	7	469	SB
7	49	0	43	92	1	0	6	7	644	SB
8	64	0	57	121	1	0	6	7	847	SB
9	81	0	73	154	1	0	8	9	1386	SB
10	100	0	91	191	1	0	8	9	1719	SB
11	121	0	111	232	1	0	8	9	2088	SB
12	144	0	133	277	1	0	8	9	2493	SB
13	169	0	157	326	1	0	8	9	2934	SB
14	196	0	183	379	1	0	8	9	3411	SB
15	225	0	211	436	1	0	8	9	3924	SB
16	256	0	241	497	1	0	8	9	4473	SB
17	289	0	273	562	1	0	10	11	6182	SB
18	324	0	307	631	1	0	10	11	6941	SB
19	361	0	343	704	1	0	10	11	7744	SB
20	400	0	381	781	1	0	10	11	8591	SB

Table 6.7 - continued from previous page

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
21	441	0	421	862	1	0	10	11	9482	SB
22	484	0	463	947	1	0	10	11	10417	SB
23	529	0	507	1036	1	0	10	11	11396	SB
24	576	0	553	1129	1	0	10	11	12419	SB
25	625	0	601	1226	1	0	10	11	13486	SB
26	676	0	651	1327	1	0	10	11	14597	SB
27	729	0	703	1432	1	0	10	11	15752	SB
28	784	0	757	1541	1	0	10	11	16951	SB
29	841	0	813	1654	1	0	10	11	18194	SB
30	900	0	871	1771	1	0	10	11	19481	SB
31	961	0	931	1892	1	0	10	11	20812	SB
32	1024	0	993	2017	1	0	10	11	22187	SB
33	1089	0	1057	2146	1	0	12	13	27898	SB
34	1156	0	1123	2279	1	0	12	13	29627	SB
35	1225	0	1191	2416	1	0	12	13	31408	SB
36	1296	0	1261	2557	1	0	12	13	33241	SB
37	1369	0	1333	2702	1	0	12	13	35126	SB
38	1444	0	1407	2851	1	0	12	13	37063	SB
39	1521	0	1483	3004	1	0	12	13	39052	SB
40	1600	0	1561	3161	1	0	12	13	41093	SB
41	1681	0	1641	3322	1	0	12	13	43186	SB
42	1764	0	1723	3487	1	0	12	13	45331	SB
43	1849	0	1807	3656	1	0	12	13	47528	SB
44	1936	0	1893	3829	1	0	12	13	49777	SB
45	2025	0	1981	4006	1	0	12	13	52078	SB
46	2116	0	2071	4187	1	0	12	13	54431	SB
47	2209	0	2163	4372	1	0	12	13	56836	SB
48	2304	0	2257	4561	1	0	12	13	59293	SB
49	2401	0	2353	4754	1	0	12	13	61802	SB
50	2500	0	2451	4951	1	0	12	13	64363	SB
51	2601	0	2551	5152	1	0	12	13	66976	SB
52	2704	0	2653	5357	1	0	12	13	69641	SB
53	2809	0	2757	5566	1	0	12	13	72358	SB
54	2916	0	2863	5779	1	0	12	13	75127	SB
55	3025	0	2971	5996	1	0	12	13	77948	SB
56	3136	0	3081	6217	1	0	12	13	80821	SB
57	3249	0	3193	6442	1	0	12	13	83746	SB
58	3364	0	3307	6671	1	0	12	13	86723	SB
59	3481	0	3423	6904	1	0	12	13	89752	SB
60	3600	0	3541	7141	1	0	12	13	92833	SB

Table 6.7 - continued from previous page

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
61	3721	0	3661	7382	1	0	12	13	95966	SB
62	3844	0	3783	7627	1	0	12	13	99151	SB
63	3969	0	3907	7876	1	0	12	13	102388	SB
64	4096	0	4033	8129	1	0	12	13	105677	SB
65	4225	0	4161	8386	1	0	14	15	125790	SB
66	4356	0	4291	8647	1	0	14	15	129705	SB
67	4489	0	4423	8912	1	0	14	15	133680	SB
68	4624	0	4557	9181	1	0	14	15	137715	SB
69	4761	0	4693	9454	1	0	14	15	141810	SB
70	4900	0	4831	9731	1	0	14	15	145965	SB
71	5041	0	4971	10012	1	0	14	15	150180	SB
72	5184	0	5113	10297	1	0	14	15	154455	SB
73	5329	0	5257	10586	1	0	14	15	158790	SB
74	5476	0	5403	10879	1	0	14	15	163185	SB
75	5625	0	5551	11176	1	0	14	15	167640	SB
76	5776	0	5701	11477	1	0	14	15	172155	SB
77	5929	0	5853	11782	1	0	14	15	176730	SB
78	6084	0	6007	12091	1	0	14	15	181365	SB
79	6241	0	6163	12404	1	0	14	15	186060	SB
80	6400	0	6321	12721	1	0	14	15	190815	SB
81	6561	0	6481	13042	1	0	14	15	195630	SB
82	6724	0	6643	13367	1	0	14	15	200505	SB
83	6889	0	6807	13696	1	0	14	15	205440	SB
84	7056	0	6973	14029	1	0	14	15	210435	SB
85	7225	0	7141	14366	1	0	14	15	215490	SB
86	7396	0	7311	14707	1	0	14	15	220605	SB
87	7569	0	7483	15052	1	0	14	15	225780	SB
88	7744	0	7657	15401	1	0	14	15	231015	SB
89	7921	0	7833	15754	1	0	14	15	236310	SB
90	8100	0	8011	16111	1	0	14	15	241665	SB
91	8281	0	8191	16472	1	0	14	15	247080	SB
92	8464	0	8373	16837	1	0	14	15	252555	SB
93	8649	0	8557	17206	1	0	14	15	258090	SB
94	8836	0	8743	17579	1	0	14	15	263685	SB
95	9025	0	8931	17956	1	0	14	15	269340	SB
96	9216	0	9121	18337	1	0	14	15	275055	SB
97	9409	0	9313	18722	1	0	14	15	280830	SB
98	9604	0	9507	19111	1	0	14	15	286665	SB
99	9801	0	9703	19504	1	0	14	15	292560	SB
100	10000	0	9901	19901	1	0	14	15	298515	SB

Table 6.7 - continued from previous page

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
101	10201	0	10101	20302	1	0	14	15	304530	SB
102	10404	0	10303	20707	1	0	14	15	310605	SB
103	10609	0	10507	21116	1	0	14	15	316740	SB
104	10816	0	10713	21529	1	0	14	15	322935	SB
105	11025	0	10921	21946	1	0	14	15	329190	SB
106	11236	0	11131	22367	1	0	14	15	335505	SB
107	11449	0	11343	22792	1	0	14	15	341880	SB
108	11664	0	11557	23221	1	0	14	15	348315	SB
109	11881	0	11773	23654	1	0	14	15	354810	SB
110	12100	0	11991	24091	1	0	14	15	361365	SB
111	12321	0	12211	24532	1	0	14	15	367980	SB
112	12544	0	12433	24977	1	0	14	15	374655	SB
113	12769	0	12657	25426	1	0	14	15	381390	SB
114	12996	0	12883	25879	1	0	14	15	388185	SB
115	13225	0	13111	26336	1	0	14	15	395040	SB
116	13456	0	13341	26797	1	0	14	15	401955	SB
117	13689	0	13573	27262	1	0	14	15	408930	SB
118	13924	0	13807	27731	1	0	14	15	415965	SB
119	14161	0	14043	28204	1	0	14	15	423060	SB
120	14400	0	14281	28681	1	0	14	15	430215	SB
121	14641	0	14521	29162	1	0	14	15	437430	SB
122	14884	0	14763	29647	1	0	14	15	444705	SB
123	15129	0	15007	30136	1	0	14	15	452040	SB
124	15376	0	15253	30629	1	0	14	15	459435	SB
125	15625	0	15501	31126	1	0	14	15	466890	SB
126	15876	0	15751	31627	1	0	14	15	474405	SB
127	16129	0	16003	32132	1	0	14	15	481980	SB
128	16384	0	16257	32641	1	0	14	15	489615	SB

Table 6.8: Polynomial Multiplication Space x Delay Optimization Result (Padding Allowed, $M = A_s = A_d$, $D_m = D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	1	0	0	1	1	0	0	1	1	Initial Condition
2	4	0	3	7	1	0	2	3	21	SB
3	9	0	7	16	1	0	4	5	80	SB
4	16	0	13	29	1	0	4	5	145	SB
5	25	0	21	46	1	0	6	7	322	SB
6	36	0	31	67	1	0	6	7	469	SB
7	49	0	43	92	1	0	6	7	644	SB
8	64	0	57	121	1	0	6	7	847	SB

Table 6.8 - continued from previous page

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
9	81	0	73	154	1	0	8	9	1386	SB
10	100	0	91	191	1	0	8	9	1719	SB
11	121	0	111	232	1	0	8	9	2088	SB
12	144	0	133	277	1	0	8	9	2493	SB
13	169	0	157	326	1	0	8	9	2934	SB
14	196	0	183	379	1	0	8	9	3411	SB
15	225	0	211	436	1	0	8	9	3924	SB
16	256	0	241	497	1	0	8	9	4473	SB
17	289	0	273	562	1	0	10	11	6182	SB
18	216	36	291	543	1	1	10	12	6516	O3W
19	361	0	343	704	1	0	10	11	7744	SB
20	300	20	340	660	1	0	12	13	8580	RKB2
21	294	42	382	718	1	1	10	12	8616	O3W
22	363	22	407	792	1	0	12	13	10296	RKB2
23	384	48	485	917	1	1	10	12	11004	O3W
24	384	48	485	917	1	1	10	12	11004	O3W
25	625	0	601	1226	1	0	10	11	13486	SB
26	507	26	559	1092	1	0	12	13	14196	RKB2
27	729	0	703	1432	1	0	10	11	15752	SB
28	588	28	644	1260	1	0	12	13	16380	RKB2
29	841	0	813	1654	1	0	10	11	18194	SB
30	675	30	735	1440	1	0	12	13	18720	RKB2
31	961	0	931	1892	1	0	10	11	20812	SB
32	768	32	832	1632	1	0	12	13	21216	RKB2
33	726	66	866	1658	1	1	12	14	23212	O3W
34	864	72	1017	1953	1	1	12	14	27342	O3W
35	864	72	1017	1953	1	1	12	14	27342	O3W
36	864	72	1017	1953	1	1	12	14	27342	O3W
37	1014	78	1180	2272	1	1	12	14	31808	O3W
38	1014	78	1180	2272	1	1	12	14	31808	O3W
39	1014	78	1180	2272	1	1	12	14	31808	O3W
40	1176	84	1355	2615	1	1	12	14	36610	O3W
41	1176	84	1355	2615	1	1	12	14	36610	O3W
42	1176	84	1355	2615	1	1	12	14	36610	O3W
43	1350	90	1542	2982	1	1	12	14	41748	O3W
44	1350	90	1542	2982	1	1	12	14	41748	O3W
45	1350	90	1542	2982	1	1	12	14	41748	O3W
46	1536	96	1741	3373	1	1	12	14	47222	O3W
47	1536	96	1741	3373	1	1	12	14	47222	O3W
48	1536	96	1741	3373	1	1	12	14	47222	O3W

Table 6.8 - continued from previous page

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
49	1875	50	1975	3900	1	0	14	15	58500	RKB2
50	1875	50	1975	3900	1	0	14	15	58500	RKB2
51	1521	130	1856	3507	1	0	16	17	59619	RKB2
52	1521	130	1856	3507	1	0	16	17	59619	RKB2
53	1296	324	2079	3699	1	2	14	17	62883	O3W
54	1296	324	2079	3699	1	2	14	17	62883	O3W
55	1764	140	2125	4029	1	0	16	17	68493	RKB2
56	1764	140	2125	4029	1	0	16	17	68493	RKB2
57	2166	114	2410	4690	1	1	14	16	75040	O3W
58	2523	58	2639	5220	1	0	14	15	78300	RKB2
59	2025	150	2412	4587	1	0	16	17	77979	RKB2
60	2025	150	2412	4587	1	0	16	17	77979	RKB2
61	1764	378	2682	4824	1	2	14	17	82008	O3W
62	1764	378	2682	4824	1	2	14	17	82008	O3W
63	1764	378	2682	4824	1	2	14	17	82008	O3W
64	2304	160	2717	5181	1	0	16	17	88077	RKB2
65	2178	264	2826	5268	1	1	16	18	94824	RKB2
66	2178	264	2826	5268	1	1	16	18	94824	RKB2
67	2304	426	3338	6068	1	2	14	17	103156	O3W
68	2304	426	3338	6068	1	2	14	17	103156	O3W
69	2304	426	3338	6068	1	2	14	17	103156	O3W
70	2304	432	3357	6093	1	2	14	17	103581	O3W
71	2304	432	3357	6093	1	2	14	17	103581	O3W
72	2304	432	3357	6093	1	2	14	17	103581	O3W
73	3750	150	4072	7972	1	1	14	16	127552	O3W
74	3750	150	4072	7972	1	1	14	16	127552	O3W
75	3750	150	4072	7972	1	1	14	16	127552	O3W
76	3042	310	3803	7155	1	1	16	18	128790	RKB2
77	3042	312	3810	7164	1	1	16	18	128952	RKB2
78	3042	312	3810	7164	1	1	16	18	128952	RKB2
79	3528	332	4342	8202	1	1	16	18	147636	RKB2
80	3528	332	4342	8202	1	1	16	18	147636	RKB2
81	3528	334	4349	8211	1	1	16	18	147798	RKB2
82	3528	334	4349	8211	1	1	16	18	147798	RKB2
83	3528	336	4356	8220	1	1	16	18	147960	RKB2
84	3528	336	4356	8220	1	1	16	18	147960	RKB2
85	4050	356	4924	9330	1	1	16	18	167940	RKB2
86	4050	356	4924	9330	1	1	16	18	167940	RKB2
87	4050	358	4931	9339	1	1	16	18	168102	RKB2
88	4050	358	4931	9339	1	1	16	18	168102	RKB2

Table 6.8 - continued from previous page

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
89	4050	360	4938	9348	1	1	16	18	168264	RKB2
90	4050	360	4938	9348	1	1	16	18	168264	RKB2
91	4608	380	5542	10530	1	1	16	18	189540	RKB2
92	4608	380	5542	10530	1	1	16	18	189540	RKB2
93	4608	382	5549	10539	1	1	16	18	189702	RKB2
94	4608	382	5549	10539	1	1	16	18	189702	RKB2
95	4608	384	5556	10548	1	1	16	18	189864	RKB2
96	4608	384	5556	10548	1	1	16	18	189864	RKB2
97	4356	594	5814	10764	1	2	16	19	204516	O3W
98	4356	594	5814	10764	1	2	16	19	204516	O3W
99	4356	594	5814	10764	1	2	16	19	204516	O3W
100	5625	250	6272	12147	1	0	18	19	230793	RKB2
101	4563	492	5922	10977	1	0	20	21	230517	RKB2
102	4563	492	5922	10977	1	0	20	21	230517	RKB2
103	4563	494	5929	10986	1	0	20	21	230706	RKB2
104	4563	494	5929	10986	1	0	20	21	230706	RKB2
105	5184	642	6758	12584	1	2	16	19	239096	O3W
106	5184	648	6777	12609	1	2	16	19	239571	O3W
107	5184	648	6777	12609	1	2	16	19	239571	O3W
108	5184	648	6777	12609	1	2	16	19	239571	O3W
109	5292	530	6757	12579	1	0	20	21	264159	RKB2
110	5292	530	6757	12579	1	0	20	21	264159	RKB2
111	5292	532	6764	12588	1	0	20	21	264348	RKB2
112	5292	532	6764	12588	1	0	20	21	264348	RKB2
113	6084	696	7793	14573	1	2	16	19	276887	O3W
114	6084	696	7793	14573	1	2	16	19	276887	O3W
115	6084	702	7812	14598	1	2	16	19	277362	O3W
116	6084	702	7812	14598	1	2	16	19	277362	O3W
117	6084	702	7812	14598	1	2	16	19	277362	O3W
118	6075	568	7646	14289	1	0	20	21	300069	RKB2
119	6075	570	7653	14298	1	0	20	21	300258	RKB2
120	6075	570	7653	14298	1	0	20	21	300258	RKB2
121	5292	1256	8470	15018	1	2	18	21	315378	RKB2
122	5292	1256	8470	15018	1	2	18	21	315378	RKB2
123	5292	1258	8477	15027	1	2	18	21	315567	RKB2
124	5292	1258	8477	15027	1	2	18	21	315567	RKB2
125	5292	1260	8484	15036	1	2	18	21	315756	RKB2
126	5292	1260	8484	15036	1	2	18	21	315756	RKB2
127	6912	608	8596	16116	1	0	20	21	338436	RKB2
128	6912	608	8596	16116	1	0	20	21	338436	RKB2

Table 6.9: TMVP Space Optimization Result (Padding Allowed, $M = A_s = A_d$, $D_m = D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	1	0	0	1	1	0	0	1	1	Initial Condition
2	4	0	2	6	1	0	1	2	12	TMVP-SB
3	9	0	6	15	1	0	2	3	45	TMVP-SB
4	16	0	12	28	1	0	2	3	84	TMVP-SB
5	25	0	20	45	1	0	3	4	180	TMVP-SB
6	27	11	24	62	1	1	3	5	310	TMVP-STD-2
7	40	11	36	87	1	1	4	6	522	TMVP-LT
8	48	15	44	107	1	1	3	5	535	TMVP-STD-2
9	54	26	54	134	1	1	4	6	804	TMVP-STD-3
10	75	19	70	164	1	1	4	6	984	TMVP-STD-2
11	96	19	90	205	1	1	5	7	1435	TMVP-LT
12	81	56	84	221	1	2	4	7	1547	TMVP-STD-2
13	106	56	108	270	1	2	5	8	2160	TMVP-LT
14	120	60	122	302	1	2	5	8	2416	TMVP-STD-2
15	150	44	150	344	1	1	5	7	2408	TMVP-STD-3
16	144	76	148	368	1	2	4	7	2576	TMVP-STD-2
17	177	76	180	433	1	2	5	8	3464	TMVP-LT
18	162	113	180	455	1	2	5	8	3640	TMVP-STD-2
19	199	113	216	528	1	2	6	9	4752	TMVP-LT
20	225	96	230	551	1	2	5	8	4408	TMVP-STD-2
21	240	128	258	626	1	2	6	9	5634	TMVP-STD-3
22	288	100	292	680	1	2	6	9	6120	TMVP-STD-2
23	243	215	276	734	1	3	5	9	6606	TMVP-STD-2
24	243	215	276	734	1	3	5	9	6606	TMVP-STD-2
25	292	215	324	831	1	3	6	10	8310	TMVP-LT
26	318	219	350	887	1	3	6	10	8870	TMVP-STD-2
27	270	320	333	923	1	3	6	10	9230	TMVP-3
28	360	235	394	989	1	3	6	10	9890	TMVP-STD-2
29	375	307	420	1102	1	3	6	10	11020	TMVP-3
30	375	307	420	1102	1	3	6	10	11020	TMVP-3
31	432	291	476	1199	1	3	5	9	10791	TMVP-STD-2
32	432	291	476	1199	1	3	5	9	10791	TMVP-STD-2
33	497	291	540	1328	1	3	6	10	13280	TMVP-LT
34	531	295	574	1400	1	3	6	10	14000	TMVP-STD-2
35	405	536	504	1445	1	4	6	11	15895	TMVP-3
36	405	536	504	1445	1	4	6	11	15895	TMVP-3
37	478	536	576	1590	1	4	7	12	19080	TMVP-LT
38	597	414	686	1697	1	3	7	11	18667	TMVP-STD-2
39	530	558	631	1719	1	4	7	12	20628	TMVP-3

Table 6.9 - continued from previous page

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
40	675	367	730	1772	1	3	6	10	17720	TMVP-STD-2
41	600	600	708	1908	1	4	7	12	22896	TMVP-3
42	600	600	708	1908	1	4	7	12	22896	TMVP-3
43	685	600	792	2077	1	4	8	13	27001	TMVP-LT
44	750	542	855	2147	1	3	7	11	23617	TMVP-3
45	750	542	855	2147	1	3	7	11	23617	TMVP-3
46	720	724	852	2296	1	4	6	11	25256	TMVP-3
47	720	724	852	2296	1	4	6	11	25256	TMVP-3
48	720	724	852	2296	1	4	6	11	25256	TMVP-3
49	817	724	948	2489	1	4	7	12	29868	TMVP-LT
50	876	744	1022	2642	1	4	7	12	31704	TMVP-STD-2
51	885	746	1019	2650	1	4	7	12	31800	TMVP-3
52	810	953	1026	2789	1	4	7	12	33468	TMVP-3
53	810	953	1026	2789	1	4	7	12	33468	TMVP-3
54	810	953	1026	2789	1	4	7	12	33468	TMVP-3
55	919	953	1134	3006	1	4	8	13	39078	TMVP-LT
56	1080	816	1238	3134	1	4	7	12	37608	TMVP-STD-2
57	995	975	1213	3183	1	4	8	13	41379	TMVP-3
58	1125	912	1290	3327	1	4	7	12	39924	TMVP-3
59	1125	912	1290	3327	1	4	7	12	39924	TMVP-3
60	1125	912	1290	3327	1	4	7	12	39924	TMVP-3
61	1246	912	1410	3568	1	4	8	13	46384	TMVP-LT
62	1200	1094	1437	3731	1	4	8	13	48503	TMVP-3
63	1200	1094	1437	3731	1	4	8	13	48503	TMVP-3
64	1296	1000	1492	3788	1	4	6	11	41668	TMVP-STD-2
65	1440	976	1614	4030	1	4	8	13	52390	TMVP-3
66	1440	976	1614	4030	1	4	8	13	52390	TMVP-3
67	1573	976	1746	4295	1	4	9	14	60130	TMVP-LT
68	1215	1573	1541	4329	1	5	7	13	56277	TMVP-3
69	1215	1573	1541	4329	1	5	7	13	56277	TMVP-3
70	1215	1595	1548	4358	1	5	7	13	56654	TMVP-3
71	1215	1595	1548	4358	1	5	7	13	56654	TMVP-3
72	1215	1595	1548	4358	1	5	7	13	56654	TMVP-3
73	1360	1595	1692	4647	1	5	8	14	65058	TMVP-LT
74	1460	1617	1795	4872	1	5	8	14	68208	TMVP-3
75	1460	1617	1795	4872	1	5	8	14	68208	TMVP-3
76	1393	1999	1778	5170	1	5	9	15	77550	TMVP-4
77	1590	1659	1932	5181	1	5	8	14	72534	TMVP-3
78	1590	1659	1932	5181	1	5	8	14	72534	TMVP-3
79	1350	2186	1854	5390	1	5	8	14	75460	TMVP-3

Table 6.9 - continued from previous page

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
80	1350	2186	1854	5390	1	5	8	14	75460	TMVP-3
81	1350	2186	1854	5390	1	5	8	14	75460	TMVP-3
82	1513	2186	2016	5715	1	5	9	15	85725	TMVP-LT
83	1800	1783	2166	5749	1	5	8	14	80486	TMVP-3
84	1800	1783	2166	5749	1	5	8	14	80486	TMVP-3
85	1969	1783	2334	6086	1	5	9	15	91290	TMVP-LT
86	1875	2165	2303	6343	1	5	8	14	88802	TMVP-3
87	1875	2165	2303	6343	1	5	8	14	88802	TMVP-3
88	1875	2187	2310	6372	1	5	8	14	89208	TMVP-3
89	1875	2187	2310	6372	1	5	8	14	89208	TMVP-3
90	1875	2187	2310	6372	1	5	8	14	89208	TMVP-3
91	2056	2187	2490	6733	1	5	9	15	100995	TMVP-LT
92	2160	2129	2597	6886	1	5	7	13	89518	TMVP-3
93	2160	2129	2597	6886	1	5	7	13	89518	TMVP-3
94	2160	2151	2604	6915	1	5	7	13	89895	TMVP-3
95	2160	2151	2604	6915	1	5	7	13	89895	TMVP-3
96	2160	2151	2604	6915	1	5	7	13	89895	TMVP-3
97	2353	2151	2796	7300	1	5	8	14	102200	TMVP-LT
98	2485	2173	2931	7589	1	5	8	14	106246	TMVP-3
99	2485	2173	2931	7589	1	5	8	14	106246	TMVP-3
100	2044	3103	2618	7765	1	6	9	16	124240	TMVP-4
101	2655	2215	3108	7978	1	5	8	14	111692	TMVP-3
102	2655	2215	3108	7978	1	5	8	14	111692	TMVP-3
103	2025	3442	2765	8232	1	6	8	15	123480	TMVP-3
104	2025	3442	2765	8232	1	6	8	15	123480	TMVP-3
105	2025	3442	2765	8232	1	6	8	15	123480	TMVP-3
106	2025	3464	2772	8261	1	6	8	15	123915	TMVP-3
107	2025	3464	2772	8261	1	6	8	15	123915	TMVP-3
108	2025	3464	2772	8261	1	6	8	15	123915	TMVP-3
109	2242	3464	2988	8694	1	6	9	16	139104	TMVP-LT
110	2390	3486	3139	9015	1	6	9	16	144240	TMVP-3
111	2390	3486	3139	9015	1	6	9	16	144240	TMVP-3
112	2520	3438	3150	9108	1	6	9	16	145728	TMVP-4
113	2745	3438	3374	9557	1	6	10	17	162469	TMVP-LT
114	2985	2898	3696	9579	1	5	9	15	143685	TMVP-3
115	2650	3640	3428	9718	1	6	9	16	155488	TMVP-3
116	2650	3640	3428	9718	1	6	9	16	155488	TMVP-3
117	2650	3640	3428	9718	1	6	9	16	155488	TMVP-3
118	3375	2707	3930	10012	1	5	8	14	140168	TMVP-3
119	3375	2707	3930	10012	1	5	8	14	140168	TMVP-3

Table 6.9 - continued from previous page

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
120	3375	2707	3930	10012	1	5	8	14	140168	TMVP-3
121	3616	2707	4170	10493	1	5	9	15	157395	TMVP-LT
122	3000	3894	3827	10721	1	6	9	16	171536	TMVP-3
123	3000	3894	3827	10721	1	6	9	16	171536	TMVP-3
124	3000	3916	3834	10750	1	6	9	16	172000	TMVP-3
125	3000	3916	3834	10750	1	6	9	16	172000	TMVP-3
126	3000	3916	3834	10750	1	6	9	16	172000	TMVP-3
127	3024	4090	3780	10894	1	6	8	15	163410	TMVP-4
128	3024	4090	3780	10894	1	6	8	15	163410	TMVP-4

Table 6.10: TMVP Delay Optimization Result (Padding Allowed, $M = A_s = A_d$, $D_m = D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	1	0	0	1	1	0	0	1	1	Initial Condition
2	4	0	2	6	1	0	1	2	12	TMVP-SB
3	9	0	6	15	1	0	2	3	45	TMVP-SB
4	16	0	12	28	1	0	2	3	84	TMVP-SB
5	25	0	20	45	1	0	3	4	180	TMVP-SB
6	36	0	30	66	1	0	3	4	264	TMVP-SB
7	49	0	42	91	1	0	3	4	364	TMVP-SB
8	64	0	56	120	1	0	3	4	480	TMVP-SB
9	81	0	72	153	1	0	4	5	765	TMVP-SB
10	100	0	90	190	1	0	4	5	950	TMVP-SB
11	121	0	110	231	1	0	4	5	1155	TMVP-SB
12	144	0	132	276	1	0	4	5	1380	TMVP-SB
13	169	0	156	325	1	0	4	5	1625	TMVP-SB
14	196	0	182	378	1	0	4	5	1890	TMVP-SB
15	225	0	210	435	1	0	4	5	2175	TMVP-SB
16	256	0	240	496	1	0	4	5	2480	TMVP-SB
17	289	0	272	561	1	0	5	6	3366	TMVP-SB
18	324	0	306	630	1	0	5	6	3780	TMVP-SB
19	361	0	342	703	1	0	5	6	4218	TMVP-SB
20	400	0	380	780	1	0	5	6	4680	TMVP-SB
21	441	0	420	861	1	0	5	6	5166	TMVP-SB
22	484	0	462	946	1	0	5	6	5676	TMVP-SB
23	529	0	506	1035	1	0	5	6	6210	TMVP-SB
24	576	0	552	1128	1	0	5	6	6768	TMVP-SB
25	625	0	600	1225	1	0	5	6	7350	TMVP-SB
26	676	0	650	1326	1	0	5	6	7956	TMVP-SB
27	729	0	702	1431	1	0	5	6	8586	TMVP-SB

Table 6.10 - continued from previous page

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
28	784	0	756	1540	1	0	5	6	9240	TMVP-SB
29	841	0	812	1653	1	0	5	6	9918	TMVP-SB
30	900	0	870	1770	1	0	5	6	10620	TMVP-SB
31	961	0	930	1891	1	0	5	6	11346	TMVP-SB
32	1024	0	992	2016	1	0	5	6	12096	TMVP-SB
33	1089	0	1056	2145	1	0	6	7	15015	TMVP-SB
34	1156	0	1122	2278	1	0	6	7	15946	TMVP-SB
35	1225	0	1190	2415	1	0	6	7	16905	TMVP-SB
36	1296	0	1260	2556	1	0	6	7	17892	TMVP-SB
37	1369	0	1332	2701	1	0	6	7	18907	TMVP-SB
38	1444	0	1406	2850	1	0	6	7	19950	TMVP-SB
39	1521	0	1482	3003	1	0	6	7	21021	TMVP-SB
40	1600	0	1560	3160	1	0	6	7	22120	TMVP-SB
41	1681	0	1640	3321	1	0	6	7	23247	TMVP-SB
42	1764	0	1722	3486	1	0	6	7	24402	TMVP-SB
43	1849	0	1806	3655	1	0	6	7	25585	TMVP-SB
44	1936	0	1892	3828	1	0	6	7	26796	TMVP-SB
45	2025	0	1980	4005	1	0	6	7	28035	TMVP-SB
46	2116	0	2070	4186	1	0	6	7	29302	TMVP-SB
47	2209	0	2162	4371	1	0	6	7	30597	TMVP-SB
48	2304	0	2256	4560	1	0	6	7	31920	TMVP-SB
49	2401	0	2352	4753	1	0	6	7	33271	TMVP-SB
50	2500	0	2450	4950	1	0	6	7	34650	TMVP-SB
51	2601	0	2550	5151	1	0	6	7	36057	TMVP-SB
52	2704	0	2652	5356	1	0	6	7	37492	TMVP-SB
53	2809	0	2756	5565	1	0	6	7	38955	TMVP-SB
54	2916	0	2862	5778	1	0	6	7	40446	TMVP-SB
55	3025	0	2970	5995	1	0	6	7	41965	TMVP-SB
56	3136	0	3080	6216	1	0	6	7	43512	TMVP-SB
57	3249	0	3192	6441	1	0	6	7	45087	TMVP-SB
58	3364	0	3306	6670	1	0	6	7	46690	TMVP-SB
59	3481	0	3422	6903	1	0	6	7	48321	TMVP-SB
60	3600	0	3540	7140	1	0	6	7	49980	TMVP-SB
61	3721	0	3660	7381	1	0	6	7	51667	TMVP-SB
62	3844	0	3782	7626	1	0	6	7	53382	TMVP-SB
63	3969	0	3906	7875	1	0	6	7	55125	TMVP-SB
64	4096	0	4032	8128	1	0	6	7	56896	TMVP-SB
65	4225	0	4160	8385	1	0	7	8	67080	TMVP-SB
66	4356	0	4290	8646	1	0	7	8	69168	TMVP-SB
67	4489	0	4422	8911	1	0	7	8	71288	TMVP-SB

Table 6.10 - continued from previous page

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
68	4624	0	4556	9180	1	0	7	8	73440	TMVP-SB
69	4761	0	4692	9453	1	0	7	8	75624	TMVP-SB
70	4900	0	4830	9730	1	0	7	8	77840	TMVP-SB
71	5041	0	4970	10011	1	0	7	8	80088	TMVP-SB
72	5184	0	5112	10296	1	0	7	8	82368	TMVP-SB
73	5329	0	5256	10585	1	0	7	8	84680	TMVP-SB
74	5476	0	5402	10878	1	0	7	8	87024	TMVP-SB
75	5625	0	5550	11175	1	0	7	8	89400	TMVP-SB
76	5776	0	5700	11476	1	0	7	8	91808	TMVP-SB
77	5929	0	5852	11781	1	0	7	8	94248	TMVP-SB
78	6084	0	6006	12090	1	0	7	8	96720	TMVP-SB
79	6241	0	6162	12403	1	0	7	8	99224	TMVP-SB
80	6400	0	6320	12720	1	0	7	8	101760	TMVP-SB
81	6561	0	6480	13041	1	0	7	8	104328	TMVP-SB
82	6724	0	6642	13366	1	0	7	8	106928	TMVP-SB
83	6889	0	6806	13695	1	0	7	8	109560	TMVP-SB
84	7056	0	6972	14028	1	0	7	8	112224	TMVP-SB
85	7225	0	7140	14365	1	0	7	8	114920	TMVP-SB
86	7396	0	7310	14706	1	0	7	8	117648	TMVP-SB
87	7569	0	7482	15051	1	0	7	8	120408	TMVP-SB
88	7744	0	7656	15400	1	0	7	8	123200	TMVP-SB
89	7921	0	7832	15753	1	0	7	8	126024	TMVP-SB
90	8100	0	8010	16110	1	0	7	8	128880	TMVP-SB
91	8281	0	8190	16471	1	0	7	8	131768	TMVP-SB
92	8464	0	8372	16836	1	0	7	8	134688	TMVP-SB
93	8649	0	8556	17205	1	0	7	8	137640	TMVP-SB
94	8836	0	8742	17578	1	0	7	8	140624	TMVP-SB
95	9025	0	8930	17955	1	0	7	8	143640	TMVP-SB
96	9216	0	9120	18336	1	0	7	8	146688	TMVP-SB
97	9409	0	9312	18721	1	0	7	8	149768	TMVP-SB
98	9604	0	9506	19110	1	0	7	8	152880	TMVP-SB
99	9801	0	9702	19503	1	0	7	8	156024	TMVP-SB
100	10000	0	9900	19900	1	0	7	8	159200	TMVP-SB
101	10201	0	10100	20301	1	0	7	8	162408	TMVP-SB
102	10404	0	10302	20706	1	0	7	8	165648	TMVP-SB
103	10609	0	10506	21115	1	0	7	8	168920	TMVP-SB
104	10816	0	10712	21528	1	0	7	8	172224	TMVP-SB
105	11025	0	10920	21945	1	0	7	8	175560	TMVP-SB
106	11236	0	11130	22366	1	0	7	8	178928	TMVP-SB
107	11449	0	11342	22791	1	0	7	8	182328	TMVP-SB

Table 6.10 - continued from previous page

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
108	11664	0	11556	23220	1	0	7	8	185760	TMVP-SB
109	11881	0	11772	23653	1	0	7	8	189224	TMVP-SB
110	12100	0	11990	24090	1	0	7	8	192720	TMVP-SB
111	12321	0	12210	24531	1	0	7	8	196248	TMVP-SB
112	12544	0	12432	24976	1	0	7	8	199808	TMVP-SB
113	12769	0	12656	25425	1	0	7	8	203400	TMVP-SB
114	12996	0	12882	25878	1	0	7	8	207024	TMVP-SB
115	13225	0	13110	26335	1	0	7	8	210680	TMVP-SB
116	13456	0	13340	26796	1	0	7	8	214368	TMVP-SB
117	13689	0	13572	27261	1	0	7	8	218088	TMVP-SB
118	13924	0	13806	27730	1	0	7	8	221840	TMVP-SB
119	14161	0	14042	28203	1	0	7	8	225624	TMVP-SB
120	14400	0	14280	28680	1	0	7	8	229440	TMVP-SB
121	14641	0	14520	29161	1	0	7	8	233288	TMVP-SB
122	14884	0	14762	29646	1	0	7	8	237168	TMVP-SB
123	15129	0	15006	30135	1	0	7	8	241080	TMVP-SB
124	15376	0	15252	30628	1	0	7	8	245024	TMVP-SB
125	15625	0	15500	31125	1	0	7	8	249000	TMVP-SB
126	15876	0	15750	31626	1	0	7	8	253008	TMVP-SB
127	16129	0	16002	32131	1	0	7	8	257048	TMVP-SB
128	16384	0	16256	32640	1	0	7	8	261120	TMVP-SB

Table 6.11: TMVP Space x Delay Optimization Result (Padding Allowed, $M = A_s = A_d$, $D_m = D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	1	0	0	1	1	0	0	1	1	Initial Condition
2	4	0	2	6	1	0	1	2	12	TMVP-SB
3	9	0	6	15	1	0	2	3	45	TMVP-SB
4	16	0	12	28	1	0	2	3	84	TMVP-SB
5	25	0	20	45	1	0	3	4	180	TMVP-SB
6	36	0	30	66	1	0	3	4	264	TMVP-SB
7	49	0	42	91	1	0	3	4	364	TMVP-SB
8	64	0	56	120	1	0	3	4	480	TMVP-SB
9	81	0	72	153	1	0	4	5	765	TMVP-SB
10	100	0	90	190	1	0	4	5	950	TMVP-SB
11	121	0	110	231	1	0	4	5	1155	TMVP-SB
12	96	35	96	227	1	1	4	6	1362	TMVP-STD-3
13	169	0	156	325	1	0	4	5	1625	TMVP-SB
14	147	27	140	314	1	1	4	6	1884	TMVP-STD-2
15	225	0	210	435	1	0	4	5	2175	TMVP-SB

Table 6.11 - continued from previous page

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
16	192	31	184	407	1	1	4	6	2442	TMVP-STD-2
17	225	31	216	472	1	1	5	7	3304	TMVP-LT
18	216	53	216	485	1	1	5	7	3395	TMVP-STD-3
19	361	0	342	703	1	0	5	6	4218	TMVP-SB
20	300	39	290	629	1	1	5	7	4403	TMVP-STD-2
21	294	62	294	650	1	1	5	7	4550	TMVP-STD-3
22	363	43	352	758	1	1	5	7	5306	TMVP-STD-2
23	384	71	384	839	1	1	5	7	5873	TMVP-STD-3
24	384	71	384	839	1	1	5	7	5873	TMVP-STD-3
25	625	0	600	1225	1	0	5	6	7350	TMVP-SB
26	507	51	494	1052	1	1	5	7	7364	TMVP-STD-2
27	441	136	448	1025	1	2	5	8	8200	TMVP-STD-2
28	441	136	448	1025	1	2	5	8	8200	TMVP-STD-2
29	675	59	660	1394	1	1	5	7	9758	TMVP-STD-2
30	675	59	660	1394	1	1	5	7	9758	TMVP-STD-2
31	576	156	584	1316	1	2	5	8	10528	TMVP-STD-2
32	576	156	584	1316	1	2	5	8	10528	TMVP-STD-2
33	726	98	726	1550	1	1	6	8	12400	TMVP-STD-3
34	675	160	682	1517	1	2	6	9	13653	TMVP-STD-2
35	576	317	648	1541	1	2	6	9	13869	TMVP-STD-3
36	576	317	648	1541	1	2	6	9	13869	TMVP-STD-3
37	649	317	720	1686	1	2	7	10	16860	TMVP-LT
38	1014	116	1014	2144	1	1	6	8	17152	TMVP-STD-3
39	1014	116	1014	2144	1	1	6	8	17152	TMVP-STD-3
40	900	196	910	2006	1	2	6	9	18054	TMVP-STD-2
41	882	269	924	2075	1	2	6	9	18675	TMVP-STD-2
42	882	269	924	2075	1	2	6	9	18675	TMVP-STD-2
43	1089	216	1100	2405	1	2	6	9	21645	TMVP-STD-2
44	1089	216	1100	2405	1	2	6	9	21645	TMVP-STD-2
45	1350	134	1350	2834	1	1	6	8	22672	TMVP-STD-3
46	1152	304	1198	2654	1	2	6	9	23886	TMVP-STD-2
47	1152	308	1200	2660	1	2	6	9	23940	TMVP-STD-2
48	1152	308	1200	2660	1	2	6	9	23940	TMVP-STD-2
49	1249	308	1296	2853	1	2	7	10	28530	TMVP-LT
50	1875	99	1850	3824	1	1	6	8	30592	TMVP-STD-2
51	1521	256	1534	3311	1	2	6	9	29799	TMVP-STD-2
52	1521	256	1534	3311	1	2	6	9	29799	TMVP-STD-2
53	1296	479	1404	3179	1	2	7	10	31790	TMVP-STD-3
54	1296	479	1404	3179	1	2	7	10	31790	TMVP-STD-3
55	1323	519	1400	3242	1	3	6	10	32420	TMVP-STD-2

Table 6.11 - continued from previous page

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
56	1323	519	1400	3242	1	3	6	10	32420	TMVP-STD-2
57	1436	519	1512	3467	1	3	7	11	38137	TMVP-LT
58	2025	292	2038	4355	1	2	6	9	39195	TMVP-STD-2
59	2025	296	2040	4361	1	2	6	9	39249	TMVP-STD-2
60	2025	296	2040	4361	1	2	6	9	39249	TMVP-STD-2
61	1728	591	1814	4133	1	3	6	10	41330	TMVP-STD-2
62	1728	591	1814	4133	1	3	6	10	41330	TMVP-STD-2
63	1728	595	1816	4139	1	3	6	10	41390	TMVP-STD-2
64	1728	595	1816	4139	1	3	6	10	41390	TMVP-STD-2
65	1857	595	1944	4396	1	3	7	11	48356	TMVP-LT
66	2178	425	2244	4847	1	2	7	10	48470	TMVP-STD-2
67	2025	615	2114	4754	1	3	7	11	52294	TMVP-STD-2
68	2025	615	2114	4754	1	3	7	11	52294	TMVP-STD-2
69	1728	1090	2014	4832	1	3	7	11	53152	TMVP-STD-2
70	1728	1090	2014	4832	1	3	7	11	53152	TMVP-STD-2
71	1728	1094	2016	4838	1	3	7	11	53218	TMVP-STD-2
72	1728	1094	2016	4838	1	3	7	11	53218	TMVP-STD-2
73	1873	1094	2160	5127	1	3	8	12	61524	TMVP-LT
74	1947	1098	2234	5279	1	3	8	12	63348	TMVP-STD-2
75	3042	499	3118	6659	1	2	7	10	66590	TMVP-STD-2
76	2535	819	2652	6006	1	3	7	11	66066	TMVP-3
77	2535	819	2652	6006	1	3	7	11	66066	TMVP-3
78	2535	819	2652	6006	1	3	7	11	66066	TMVP-3
79	2700	747	2810	6257	1	3	7	11	68827	TMVP-STD-2
80	2700	747	2810	6257	1	3	7	11	68827	TMVP-STD-2
81	2205	1266	2429	5900	1	4	7	12	70800	TMVP-3
82	2205	1288	2436	5929	1	4	7	12	71148	TMVP-3
83	2205	1288	2436	5929	1	4	7	12	71148	TMVP-3
84	2205	1288	2436	5929	1	4	7	12	71148	TMVP-3
85	2374	1288	2604	6266	1	4	8	13	81458	TMVP-LT
86	3267	819	3386	7472	1	3	7	11	82192	TMVP-STD-2
87	3267	823	3388	7478	1	3	7	11	82258	TMVP-STD-2
88	3267	823	3388	7478	1	3	7	11	82258	TMVP-STD-2
89	3375	947	3510	7832	1	3	7	11	86152	TMVP-3
90	3375	947	3510	7832	1	3	7	11	86152	TMVP-3
91	2880	1454	3137	7471	1	4	7	12	89652	TMVP-3
92	2880	1454	3137	7471	1	4	7	12	89652	TMVP-3
93	2880	1454	3137	7471	1	4	7	12	89652	TMVP-3
94	2880	1476	3144	7500	1	4	7	12	90000	TMVP-3
95	2880	1476	3144	7500	1	4	7	12	90000	TMVP-3

Table 6.11 - continued from previous page

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
96	2880	1476	3144	7500	1	4	7	12	90000	TMVP-3
97	3073	1476	3336	7885	1	4	8	13	102505	TMVP-LT
98	3630	1208	3861	8699	1	3	8	12	104388	TMVP-3
99	3630	1208	3861	8699	1	3	8	12	104388	TMVP-3
100	3375	1540	3648	8563	1	4	8	13	111319	TMVP-3
101	3375	1540	3648	8563	1	4	8	13	111319	TMVP-3
102	3375	1540	3648	8563	1	4	8	13	111319	TMVP-3
103	4563	975	4706	10244	1	3	7	11	112684	TMVP-STD-2
104	4563	975	4706	10244	1	3	7	11	112684	TMVP-STD-2
105	2880	2347	3485	8712	1	4	8	13	113256	TMVP-3
106	2880	2369	3492	8741	1	4	8	13	113633	TMVP-3
107	2880	2369	3492	8741	1	4	8	13	113633	TMVP-3
108	2880	2369	3492	8741	1	4	8	13	113633	TMVP-3
109	3969	1776	4310	10055	1	4	7	12	120660	TMVP-STD-2
110	3969	1776	4310	10055	1	4	7	12	120660	TMVP-STD-2
111	3969	1780	4312	10061	1	4	7	12	120732	TMVP-STD-2
112	3969	1780	4312	10061	1	4	7	12	120732	TMVP-STD-2
113	4194	1780	4536	10510	1	4	8	13	136630	TMVP-LT
114	4308	1784	4650	10742	1	4	8	13	139646	TMVP-STD-2
115	5070	1430	5343	11843	1	3	8	12	142116	TMVP-3
116	5070	1430	5343	11843	1	3	8	12	142116	TMVP-3
117	5070	1430	5343	11843	1	3	8	12	142116	TMVP-3
118	4500	1852	4830	11182	1	4	8	13	145366	TMVP-3
119	4500	1852	4830	11182	1	4	8	13	145366	TMVP-3
120	4500	1852	4830	11182	1	4	8	13	145366	TMVP-3
121	4410	2239	4907	11556	1	4	8	13	150228	TMVP-3
122	4410	2239	4907	11556	1	4	8	13	150228	TMVP-3
123	4410	2239	4907	11556	1	4	8	13	150228	TMVP-3
124	4410	2261	4914	11585	1	4	8	13	150605	TMVP-3
125	4410	2261	4914	11585	1	4	8	13	150605	TMVP-3
126	4410	2261	4914	11585	1	4	8	13	150605	TMVP-3
127	5184	2040	5576	12800	1	4	7	12	153600	TMVP-STD-2
128	5184	2040	5576	12800	1	4	7	12	153600	TMVP-STD-2

We share our extended hybrid iteration results in the appendix. Although some slight changes in hybrid sequences occur, their primary attitude does not alter significantly from the asymptotic complexity point of view as expected. Bigger split size constructions start to dominate algorithm choice in larger sizes. For instance, while the 'TMVP-3' is a better choice until the size around 200, 'TMVP-4' starts to dominate the choice of algorithm for bigger sizes.

Below Table 6.12 is a look-up table for extended hybrid results of TMVP given in the Appendix A

Table 6.12: Extended Hybrid Multiplication Lookup Table

Table Index	Alg. Container	Optimized Cost	Space Weightings	Delay Weightings	Padding Choice
Table A.1	TMVP	Space	$M = A_s = A_d$	$D_m = D_s = D_d$	Not Allowed
Table A.2	TMVP	Space	$M = 2A_s = 2A_d$	$D_m = D_s = D_d$	Not Allowed
Table A.3	TMVP	Space	$M = 4A_s = 4A_d$	$D_m = D_s = D_d$	Not Allowed
Table A.4	TMVP	Space	$M = 4A_s = 2A_d$	$D_m = D_s = D_d$	Not Allowed
Table A.5	TMVP	Space	$M = A_s = A_d$	$2D_m = 2D_s = D_d$	Not Allowed
Table A.6	TMVP	Space	$M = 2A_s = 2A_d$	$2D_m = 2D_s = D_d$	Not Allowed
Table A.7	TMVP	Space	$M = 4A_s = 4A_d$	$2D_m = 2D_s = D_d$	Not Allowed
Table A.8	TMVP	Space	$M = 4A_s = 2A_d$	$2D_m = 2D_s = D_d$	Not Allowed
Table A.9	TMVP	Space x Delay	$M = A_s = A_d$	$D_m = D_s = D_d$	Not Allowed
Table A.10	TMVP	Space x Delay	$M = 2A_s = 2A_d$	$D_m = D_s = D_d$	Not Allowed
Table A.11	TMVP	Space x Delay	$M = 4A_s = 4A_d$	$D_m = D_s = D_d$	Not Allowed
Table A.12	TMVP	Space x Delay	$M = 4A_s = 2A_d$	$D_m = D_s = D_d$	Not Allowed
Table A.13	TMVP	Space x Delay	$M = A_s = A_d$	$2D_m = 2D_s = D_d$	Not Allowed
Table A.14	TMVP	Space x Delay	$M = 2A_s = 2A_d$	$2D_m = 2D_s = D_d$	Not Allowed
Table A.15	TMVP	Space x Delay	$M = 4A_s = 4A_d$	$2D_m = 2D_s = D_d$	Not Allowed
Table A.16	TMVP	Space x Delay	$M = 4A_s = 2A_d$	$2D_m = 2D_s = D_d$	Not Allowed
Table A.17	TMVP	Space	$M = A_s = A_d$	$D_m = D_s = D_d$	Allowed
Table A.18	TMVP	Space	$M = 2A_s = 2A_d$	$D_m = D_s = D_d$	Allowed
Table A.19	TMVP	Space	$M = 4A_s = 4A_d$	$D_m = D_s = D_d$	Allowed
Table A.20	TMVP	Space	$M = 4A_s = 2A_d$	$D_m = D_s = D_d$	Allowed
Table A.21	TMVP	Space	$M = A_s = A_d$	$2D_m = 2D_s = D_d$	Allowed
Table A.22	TMVP	Space	$M = 2A_s = 2A_d$	$2D_m = 2D_s = D_d$	Allowed
Table A.23	TMVP	Space	$M = 4A_s = 4A_d$	$2D_m = 2D_s = D_d$	Allowed
Table A.24	TMVP	Space	$M = 4A_s = 2A_d$	$2D_m = 2D_s = D_d$	Allowed
Table A.25	TMVP	Space x Delay	$M = A_s = A_d$	$D_m = D_s = D_d$	Allowed
Table A.26	TMVP	Space x Delay	$M = 2A_s = 2A_d$	$D_m = D_s = D_d$	Allowed
Table A.27	TMVP	Space x Delay	$M = 4A_s = 4A_d$	$D_m = D_s = D_d$	Allowed
Table A.28	TMVP	Space x Delay	$M = 4A_s = 2A_d$	$D_m = D_s = D_d$	Allowed
Table A.29	TMVP	Space x Delay	$M = A_s = A_d$	$2D_m = 2D_s = D_d$	Allowed
Table A.30	TMVP	Space x Delay	$M = 2A_s = 2A_d$	$2D_m = 2D_s = D_d$	Allowed
Table A.31	TMVP	Space x Delay	$M = 4A_s = 4A_d$	$2D_m = 2D_s = D_d$	Allowed
Table A.32	TMVP	Space x Delay	$M = 4A_s = 2A_d$	$2D_m = 2D_s = D_d$	Allowed

6.3 Comparison of Results and Analysis

From Tables 6.3 and 6.4, it is clear that schoolbook constructions for both TMVP and modular polynomial multiplication is the best among others in terms of delay complexity. Though,

there is a slight difference. While the way the schoolbook method is implemented does not affect the theoretical delay complexity in the case of TMVP, this is not the case for polynomial multiplication, and the iterative approach gives better performance than the recursive version. Even if they were theoretically the same in TMVP, reconfigurable implementation results differ as Guneysu states in [26] because any recursion time step is restricted to the synchronous circuit clock, which is not a problem in the case of an iterative approach. On the other hand, we do not concert about how it is implemented but focus on the best hybrid sequence of a given set of multiplication algorithms.

When we consider arithmetic complexities, the first obvious observation, regardless of the type of construction, is the reverse proportionality with delay complexities. The advantage of the naive schoolbook approach disappears in arithmetic complexity and even becomes the worst for large enough input size n. This result is not surprising since any algorithm has a well-known trade-off between time and space complexity. Those two and three-way split constructions become more efficient than the schoolbook for both TMVP and modular polynomial multiplications.

We see that TMVP arithmetic and delay costs are more effective than modular polynomial multiplication costs among the same split sizes. Furthermore, our hybrid iteration results show that the best hybrid TMVP construction sequence always has better efficiency than the best hybrid modular polynomial construction for all sizes regardless of the cost metric weightings and the choice of padding. This briefly shows that TMVP is a good candidate in place of modular polynomial multiplication.

CHAPTER 7

CONCLUSION

In this thesis, we have seen that the hybrid approach is useful for TMVP constructions. We extended Paksoy's work [24] by analyzing delay complexities as well. We revisited Ilter's work in [17], and reconsidered arithmetic complexities in case of a modular multiplication scenario, and we derived delay complexities accordingly. Finally, we compared them in the previous Chapter 6, and saw that TMVP construction is a decent candidate compared to modular polynomial multiplication algorithms compiled in [17]. Therefore, we concluded that TMVP is worthwhile, especially in reconfigurable hardware implementations. Moreover, we have introduced our hybrid analysis approach and seen how it is vital in optimizing a given target cost under different cost-weight settings and padding choices.

Throughout the thesis, our assumptions and target operations determining our cost functions were restricted in order to provide a fair comparison with previous works [17],[24]. In real-life implementations, there might be many other concerns that need to be considered. For this purpose, one further possible study could extend our cost metrics by means of including shifting costs. Moreover, our cost weightings are comfortably chosen in our hybrid analysis results. Another work could enlighten the optimum way of determining weightings for target cost metrics by analyzing an actual MCU or reconfigurable hardware implementation. For example, the cost of single multiplication is quite close to the cost of addition in some architectures. However, this might not be the case for some other architectures. Systematic choice of these weightings for a target platform would be more beneficial to find out more efficient hybrid sequences of algorithms. We freely chose these weights in our thesis and shared corresponding iteration results.

REFERENCES

[1] S. Akleylek, E. Alkım, and Z. Y. Tok, Sparse polynomial multiplication for lattice-based cryptography with small complexity, The Journal of Supercomputing, 72(2), pp. 438–450, 2016.

[2] M. Anwar Hasan and C. Negre, Multiway splitting method for toeplitz matrix vector product, IEEE Transactions on Computers, 62(7), pp. 1467–1471, 2013.

[3] S. Arish and R. Sharma, Run-time-reconfigurable multi-precision floating-point matrix multiplier intellectual property core on FPGA, Circuits, Systems, and Signal Processing, 36(3), pp. 998–1026, 2017.

[4] D. V. Bailey, D. Coffin, A. Elbirt, J. H. Silverman, and A. D. Woodbury, Ntru in constrained devices, in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 262–272, Springer, 2001.

[5] D. J. Bernstein, Batch binary edwards, in *Annual International Cryptology Conference*, pp. 317–336, Springer, 2009.

[6] D. J. Bernstein, Batch binary edwards, in *Annual International Cryptology Conference*, pp. 317–336, Springer, 2009.

[7] M. Cenk and M. A. Hasan, Some new results on binary polynomial multiplication, Journal of Cryptographic Engineering, 5(4), pp. 289–303, 2015.

[8] M. Cenk et al., Efficient big integer multiplication in cryptography, International Journal of Information Security Science, 6(4), pp. 70–78, 2017.

[9] A. E. Cohen and K. K. Parhi, Implementation of scalable elliptic curve cryptosystem crypto-accelerators for gf (2/sup m/), in *Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers, 2004.*, volume 1, pp. 471–477, IEEE, 2004.

[10] S. A. Cook and S. O. Aanderaa, On the minimum computation time of functions, Transactions of the American Mathematical Society, 142, pp. 291–314, 1969.

[11] Z. Dyka, P. Langendoerfer, and F. Vater, Combining multiplication methods with optimized processing sequence for polynomial multiplier in gf (2 k), in *Western European Workshop on Research in Cryptology*, pp. 137–150, Springer, 2011.

[12] H. Fan and M. A. Hasan, A new approach to subquadratic space complexity parallel multipliers for extended binary fields, IEEE Transactions on Computers, 56(2), pp. 224–233, 2007.

[13] C. Grabbe, M. Bednara, J. Teich, J. von zur Gathen, and J. Shokrollahi, Fpga designs of parallel high performance gf (2/sup 233/) multipliers [cryptographic applications], in *Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS'03.*, volume 2, pp. II–II, IEEE, 2003.

[14] M. A. Hasan, N. Meloni, A. H. Namin, and C. Negre, Block recombination approach for subquadratic space complexity binary field multiplication based on toeplitz matrix-vector product, IEEE Transactions on Computers, 61(2), pp. 151–163, 2010.

[15] M. A. Hasan, N. Meloni, A. H. Namin, and C. Negre, Block Recombination Approach for Subquadratic Space Complexity Binary Field Multiplication Based on Toeplitz Matrix-Vector Producta, IEEE Transactions on Computers, 61(2), pp. 151–163, 2012.

[16] M. A. Hasan, M. Wang, and V. K. Bhargava, Modular construction of low complexity parallel multipliers for a class of finite fields gf(2ˆm), IEEE Transactions on Computers, 41(8), pp. 962–971, 1992.

[17] M. B. İlter, *An Analysis on efficient polynomial multiplication algorithms for cryptographic purposes*, Ph.D. thesis, Middle East Technical University, 2016.

[18] M. J. Kannwischer, J. Rijneveld, and P. Schwabe, Faster multiplication on cortex-m4 to speed up nist pqc candidates, in *International Conference on Applied Cryptography and Network Security*, pp. 281–301, Springer, 2019.

[19] A. Karatsuba, Multiplication of multidigit numbers on automata, in *Soviet physics doklady*, volume 7, pp. 595–596, 1963.

[20] C. K. Koc and B. Sunar, Low-complexity bit-parallel canonical and normal basis multipliers for a class of finite fields, IEEE Transactions on Computers, 47(3), pp. 353–356, 1998.

[21] M. Machhout, M. Zeghid, B. Bouallegue, R. Tourki, et al., Efficient hardware architecture of recursive karatsuba-ofman multiplier, in *2008 3rd International Conference on Design and Technology of Integrated Systems in Nanoscale Era*, pp. 1–6, IEEE, 2008.

[22] E. D. Mastrovito, Vlsi designs for multiplication over finite fields gf (2 m), in *International Conference on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes*, pp. 297–309, Springer, 1988.

[23] C. Paar, A new architecture for a parallel finite field multiplier with low complexity based on composite fields, IEEE Transactions on Computers, 45(7), pp. 856–861, 1996.

[24] I. K. Paksoy and M. Cenk, TMVP-based Multiplication for Polynomial Quotient Rings and Application to Saber on ARM Cortex-M4., IACR Cryptol. ePrint Arch., 2020, p. 1302, 2020.

[25] S. Peter and P. Langendorfer, An efficient polynomial multiplier in gf (2m) and its application to ecc designs, in *2007 Design, Automation & Test in Europe Conference & Exhibition*, pp. 1–6, IEEE, 2007.

[26] T. Pöppelmann and T. Güneysu, Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware, in *International conference on cryptology and information security in Latin America*, pp. 139–158, Springer, 2012.

[27] F. Rodríguez-Henríquez, On fully parallel karatsuba multipliers for gf (2ˆ m), in *Proc. International Conference on Computer Science and Technology-CST 2003, May*, Acta Press, 2003.

[28] F. Rodríguez-Henríquez, N. A. Saqib, A. D. Pérez, and C. K. Koc, *Cryptographic algorithms on reconfigurable hardware*, Springer Science & Business Media, 2007.

[29] E. Savaš, A. F. Tenca, and C. K. Koç, A scalable and unified multiplier architecture for finite fields gf (p) and gf (2 m), in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 277–292, Springer, 2000.

[30] B. Sunar and C. K. Koc, Mastrovito multiplier for all trinomials, IEEE Transactions on Computers, 48(5), pp. 522–527, 1999.

[31] H. K. Taşkın and M. Cenk, Tmvp-friendly primes for efficient elliptic curve cryptography, in *2020 International Conference on Information Security and Cryptology (ISCTURKEY)*, pp. 80–87, IEEE, 2020.

[32] J. von zur Gathen and J. Shokrollahi, Efficient fpga-based karatsuba multipliers for polynomials over gf2, in *International Workshop on Selected Areas in Cryptography*, pp. 359–369, Springer, 2005.

[33] A. Weimerskirch and C. Paar, Generalizations of the karatsuba algorithm for efficient implementations., IACR Cryptol. ePrint Arch., 2006, p. 224, 2006.

[34] S. Winograd, *Arithmetic Complexity of Computations*, Society For Industrial & Applied Mathematics, U.S., 1980.

APPENDIX A

TMVP HYBRID ANALYSIS RESULTS

Table A.1: TMVP Space Optimization Result (No Padding, $M = A_s = A_d$, $D_m = D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	1	0	0	1	1	0	0	1	1	Initial Condition
2	4	0	2	6	1	0	1	2	12	TMVP-SB
3	9	0	6	15	1	0	2	3	45	TMVP-SB
4	16	0	12	28	1	0	2	3	84	TMVP-SB
5	25	0	20	45	1	0	3	4	180	TMVP-SB
6	27	11	24	62	1	1	3	5	310	TMVP-STD-2
7	40	11	36	87	1	1	4	6	522	TMVP-LT
8	48	15	44	107	1	1	3	5	535	TMVP-STD-2
16	144	76	148	368	1	2	4	7	2576	TMVP-STD-2
31	436	307	480	1223	1	3	7	11	13453	TMVP-LT
32	432	291	476	1199	1	3	5	9	10791	TMVP-STD-2
33	497	291	540	1328	1	3	6	10	13280	TMVP-LT
44	864	387	920	2171	1	3	7	11	23881	TMVP-STD-2
45	750	542	855	2147	1	3	7	11	23617	TMVP-3
48	720	724	852	2296	1	4	6	11	25256	TMVP-3
49	817	724	948	2489	1	4	7	12	29868	TMVP-LT
63	1200	1094	1437	3731	1	4	8	13	48503	TMVP-3
64	1296	1000	1492	3788	1	4	6	11	41668	TMVP-STD-2
65	1425	1000	1620	4045	1	4	7	12	48540	TMVP-LT
67	1573	976	1746	4295	1	4	9	14	60130	TMVP-LT
77	1546	1999	1930	5475	1	5	10	16	87600	TMVP-LT
91	2056	2187	2490	6733	1	5	9	15	100995	TMVP-LT
96	2160	2151	2604	6915	1	5	7	13	89895	TMVP-3
99	2485	2173	2931	7589	1	5	8	14	106246	TMVP-3
104	2226	3196	2814	8236	1	6	9	16	131776	TMVP-4
121	3616	2707	4170	10493	1	5	9	15	157395	TMVP-LT
127	3253	3916	4086	11255	1	6	10	17	191335	TMVP-LT
127	3253	3916	4086	11255	1	6	10	17	191335	TMVP-LT
128	3024	4090	3780	10894	1	6	8	15	163410	TMVP-4

Table A.1 - continued from previous page

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
129	3281	4090	4036	11407	1	6	9	16	182512	TMVP-LT
137	3990	4248	4766	13004	1	6	10	17	221068	TMVP-LT
143	5238	3736	6080	15054	1	5	13	19	286026	TMVP-LT
169	4537	6903	5880	17320	1	7	11	19	329080	TMVP-LT
188	6538	6822	7917	21277	1	6	12	19	404263	TMVP-4
189	6000	6848	7626	20474	1	6	10	17	348058	TMVP-3
192	5040	8161	6636	19837	1	7	9	17	337229	TMVP-4
197	6112	8226	7714	22052	1	7	11	19	418988	TMVP-LT
256	9072	11133	11340	31545	1	7	9	17	536265	TMVP-4

Table A.2: TMVP Space Optimization Result (No Padding, $M = 2A_s = 2A_d$, $D_m = D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	2	0	0	2	1	0	0	1	2	Initial Condition
2	8	0	2	10	1	0	1	2	20	TMVP-SB
3	18	0	6	24	1	0	2	3	72	TMVP-SB
4	24	7	10	41	1	1	2	4	164	TMVP-STD-2
5	42	7	18	67	1	1	3	5	335	TMVP-LT
6	54	11	24	89	1	1	3	5	445	TMVP-STD-2
7	80	11	36	127	1	1	4	6	762	TMVP-LT
8	72	36	38	146	1	2	3	6	876	TMVP-STD-2
16	216	139	130	485	1	3	4	8	3880	TMVP-STD-2
31	752	412	450	1614	1	4	7	12	19368	TMVP-LT
32	648	480	422	1550	1	4	5	10	15500	TMVP-STD-2
33	778	480	486	1744	1	4	6	11	19184	TMVP-LT
44	1176	968	742	2886	1	5	8	14	40404	TMVP-4
45	1050	1007	730	2787	1	5	7	13	36231	TMVP-3
48	1080	1039	762	2881	1	5	6	12	34572	TMVP-3
49	1274	1039	858	3171	1	5	7	13	41223	TMVP-LT
63	2000	1459	1292	4751	1	5	8	14	66514	TMVP-3
64	1512	1986	1134	4632	1	6	7	14	64848	TMVP-4
65	1770	1986	1262	5018	1	6	8	15	75270	TMVP-LT
67	2786	1291	1656	5733	1	5	9	15	85995	TMVP-LT
77	2714	2461	1804	6979	1	6	10	17	118643	TMVP-LT
91	3512	2712	2340	8564	1	6	9	16	137024	TMVP-LT
96	2520	3969	2058	8547	1	7	8	16	136752	TMVP-4
99	3890	3118	2661	9669	1	6	8	15	145035	TMVP-3
104	4452	3196	2814	10462	1	6	9	16	167392	TMVP-4
121	4892	4807	3390	13089	1	7	10	18	235602	TMVP-LT
127	6506	3916	4086	14508	1	6	10	17	246636	TMVP-LT

Table A.2 - continued from previous page

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
127	6506	3916	4086	14508	1	6	10	17	246636	TMVP-LT
128	4536	5413	3402	13351	1	7	8	16	213616	TMVP-4
129	5050	5413	3658	14121	1	7	9	17	240057	TMVP-LT
137	6468	5571	4388	16427	1	7	10	18	295686	TMVP-LT
143	8586	5636	5452	19674	1	7	13	21	413154	TMVP-LT
169	9074	6903	5880	21857	1	7	11	19	415283	TMVP-LT
188	9926	10077	7042	27045	1	8	12	21	567945	TMVP-4
189	10000	8673	6901	25574	1	7	10	18	460332	TMVP-3
192	7560	10366	6006	23932	1	8	9	18	430776	TMVP-4
197	9704	10431	7084	27219	1	8	11	20	544380	TMVP-LT
256	10584	18035	8834	37453	1	9	10	20	749060	TMVP-4

Table A.3: TMVP Space Optimization Result (No Padding, $M = 4A_s = 4A_d$, $D_m = D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	4	0	0	4	1	0	0	1	4	Initial Condition
2	12	3	2	17	1	1	1	3	51	TMVP-STD-2
3	24	8	6	38	1	1	2	4	152	TMVP-STD-3
4	36	16	10	62	1	2	2	5	310	TMVP-STD-2
5	72	16	18	106	1	2	3	6	636	TMVP-LT
6	72	35	24	131	1	2	3	6	786	TMVP-STD-2
7	124	35	36	195	1	2	4	7	1365	TMVP-LT
8	108	63	38	209	1	3	3	7	1463	TMVP-STD-2
16	324	220	130	674	1	4	4	9	6066	TMVP-STD-2
31	1324	547	450	2321	1	5	7	13	30173	TMVP-LT
32	756	934	378	2068	1	6	6	13	26884	TMVP-4
33	1016	934	442	2392	1	6	7	14	33488	TMVP-LT
44	2100	1157	742	3999	1	6	8	15	59985	TMVP-4
45	1800	1232	730	3762	1	6	7	14	52668	TMVP-3
48	1620	1444	762	3826	1	6	6	13	49738	TMVP-3
49	2008	1444	858	4310	1	6	7	14	60340	TMVP-LT
63	3100	2059	1292	6451	1	6	8	15	96765	TMVP-3
64	2268	2553	1134	5955	1	7	7	15	89325	TMVP-4
65	2784	2553	1262	6599	1	7	8	16	105584	TMVP-LT
67	3840	2553	1524	7917	1	7	10	18	142506	TMVP-LT
77	4168	3301	1804	9273	1	7	10	18	166914	TMVP-LT
91	6124	3387	2340	11851	1	7	9	17	201467	TMVP-LT
96	3780	4914	2058	10752	1	8	8	17	182784	TMVP-4
99	5080	5388	2441	12909	1	8	9	18	232362	TMVP-3
104	5880	5380	2688	13948	1	8	9	18	251064	TMVP-4

Table A.3 - continued from previous page

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
121	8524	5752	3390	17666	1	8	10	19	335654	TMVP-LT
127	10312	5716	4086	20114	1	7	10	18	362052	TMVP-LT
127	10312	5716	4086	20114	1	7	10	18	362052	TMVP-LT
128	5292	8591	3094	16977	1	9	9	19	322563	TMVP-4
129	6320	8591	3350	18261	1	9	10	20	365220	TMVP-LT
137	10668	7272	4388	22328	1	8	10	19	424232	TMVP-LT
143	14952	7186	5455	27593	1	8	13	22	607046	TMVP-LT
169	14368	9423	5880	29671	1	8	11	20	593420	TMVP-LT
188	17752	11652	7042	36446	1	9	12	22	801812	TMVP-4
189	15500	11673	6901	34074	1	8	10	19	647406	TMVP-3
192	11340	13201	6006	30547	1	9	9	19	580393	TMVP-4
197	15628	13266	7084	35978	1	9	11	21	755538	TMVP-LT
256	15876	22004	8834	46714	1	10	10	21	980994	TMVP-4

Table A.4: TMVP Space Optimization Result (No Padding, $M = 4A_s = 2A_d$, $D_m = D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	4	0	0	4	1	0	0	1	4	Initial Condition
2	12	3	4	19	1	1	1	3	57	TMVP-STD-2
3	24	8	12	44	1	1	2	4	176	TMVP-STD-3
4	36	16	20	72	1	2	2	5	360	TMVP-STD-2
5	72	16	36	124	1	2	3	6	744	TMVP-LT
6	72	35	48	155	1	2	3	6	930	TMVP-STD-2
7	124	35	72	231	1	2	4	7	1617	TMVP-LT
8	108	63	76	247	1	3	3	7	1729	TMVP-STD-2
16	324	220	260	804	1	4	4	9	7236	TMVP-STD-2
31	1324	547	900	2771	1	5	7	13	36023	TMVP-LT
32	756	934	756	2446	1	6	6	13	31798	TMVP-4
33	1016	934	884	2834	1	6	7	14	39676	TMVP-LT
44	2100	1157	1484	4741	1	6	8	15	71115	TMVP-4
45	1800	1232	1460	4492	1	6	7	14	62888	TMVP-3
48	1260	1873	1428	4561	1	7	7	15	68415	TMVP-4
49	1648	1873	1620	5141	1	7	8	16	82256	TMVP-LT
63	3100	2059	2584	7743	1	6	8	15	116145	TMVP-3
64	2268	2553	2268	7089	1	7	7	15	106335	TMVP-4
65	2784	2553	2524	7861	1	7	8	16	125776	TMVP-LT
67	3840	2553	3048	9441	1	7	10	18	169938	TMVP-LT
77	4168	3301	3608	11077	1	7	10	18	199386	TMVP-LT
91	6124	3387	4680	14191	1	7	9	17	241247	TMVP-LT
96	3780	4914	4116	12810	1	8	8	17	217770	TMVP-4

Table A.4 - continued from previous page

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
99	5080	5388	4882	15350	1	8	9	18	276300	TMVP-3
104	5880	5380	5376	16636	1	8	9	18	299448	TMVP-4
121	8524	5752	6780	21056	1	8	10	19	400064	TMVP-LT
127	10312	5716	8172	24200	1	7	10	18	435600	TMVP-LT
127	10312	5716	8172	24200	1	7	10	18	435600	TMVP-LT
128	5292	8591	6188	20071	1	9	9	19	381349	TMVP-4
129	6320	8591	6700	21611	1	9	10	20	432220	TMVP-LT
137	10668	7272	8776	26716	1	8	10	19	507604	TMVP-LT
143	14952	7186	10910	33048	1	8	13	22	727056	TMVP-LT
169	14368	9423	11760	35551	1	8	11	20	711020	TMVP-LT
188	17752	11652	14084	43488	1	9	12	22	956736	TMVP-4
189	15500	11673	13802	40975	1	8	10	19	778525	TMVP-3
192	8820	16204	11340	36364	1	10	10	21	763644	TMVP-4
197	13108	16269	13496	42873	1	10	12	23	986079	TMVP-LT
256	15876	22004	17668	55548	1	10	10	21	1166508	TMVP-4

Table A.5: TMVP Space Optimization Result (No Padding, $M = A_s = A_d$, $2D_m = 2D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	1	0	0	1	1	0	0	1	1	Initial Condition
2	4	0	2	6	1	0	2	3	18	TMVP-SB
3	9	0	6	15	1	0	4	5	75	TMVP-SB
4	16	0	12	28	1	0	4	5	140	TMVP-SB
5	25	0	20	45	1	0	6	7	315	TMVP-SB
6	27	11	24	62	1	1	6	8	496	TMVP-STD-2
7	40	11	36	87	1	1	8	10	870	TMVP-LT
8	48	15	44	107	1	1	6	8	856	TMVP-STD-2
16	144	76	148	368	1	2	8	11	4048	TMVP-STD-2
31	436	307	480	1223	1	3	14	18	22014	TMVP-LT
32	432	291	476	1199	1	3	10	14	16786	TMVP-STD-2
33	497	291	540	1328	1	3	12	16	21248	TMVP-LT
44	864	387	920	2171	1	3	14	18	39078	TMVP-STD-2
45	750	542	855	2147	1	3	14	18	38646	TMVP-3
48	720	724	852	2296	1	4	12	17	39032	TMVP-3
49	817	724	948	2489	1	4	14	19	47291	TMVP-LT
63	1200	1094	1437	3731	1	4	16	21	78351	TMVP-3
64	1296	1000	1492	3788	1	4	12	17	64396	TMVP-STD-2
65	1425	1000	1620	4045	1	4	14	19	76855	TMVP-LT
67	1573	976	1746	4295	1	4	18	23	98785	TMVP-LT
77	1546	1999	1930	5475	1	5	20	26	142350	TMVP-LT

Table A.5 - continued from previous page

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
91	2056	2187	2490	6733	1	5	18	24	161592	TMVP-LT
96	2160	2151	2604	6915	1	5	14	20	138300	TMVP-3
99	2485	2173	2931	7589	1	5	16	22	166958	TMVP-3
104	2226	3196	2814	8236	1	6	18	25	205900	TMVP-4
121	3616	2707	4170	10493	1	5	18	24	251832	TMVP-LT
127	3253	3916	4086	11255	1	6	20	27	303885	TMVP-LT
127	3253	3916	4086	11255	1	6	20	27	303885	TMVP-LT
128	3024	4090	3780	10894	1	6	16	23	250562	TMVP-4
129	3281	4090	4036	11407	1	6	18	25	285175	TMVP-LT
137	3990	4248	4766	13004	1	6	20	27	351108	TMVP-LT
143	5238	3736	6080	15054	1	5	26	32	481728	TMVP-LT
169	4537	6903	5880	17320	1	7	22	30	519600	TMVP-LT
188	6538	6822	7917	21277	1	6	24	31	659587	TMVP-4
189	6000	6848	7626	20474	1	6	20	27	552798	TMVP-3
192	5040	8161	6636	19837	1	7	18	26	515762	TMVP-4
197	6112	8226	7714	22052	1	7	22	30	661560	TMVP-LT
256	9072	11133	11340	31545	1	7	18	26	820170	TMVP-4

Table A.6: TMVP Space Optimization Result (No Padding, $M = 2A_s = 2A_d$, $2D_m = 2D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	2	0	0	2	1	0	0	1	2	Initial Condition
2	8	0	2	10	1	0	2	3	30	TMVP-SB
3	18	0	6	24	1	0	4	5	120	TMVP-SB
4	24	7	10	41	1	1	4	6	246	TMVP-STD-2
5	42	7	18	67	1	1	6	8	536	TMVP-LT
6	54	11	24	89	1	1	6	8	712	TMVP-STD-2
7	80	11	36	127	1	1	8	10	1270	TMVP-LT
8	72	36	38	146	1	2	6	9	1314	TMVP-STD-2
16	216	139	130	485	1	3	8	12	5820	TMVP-STD-2
31	752	412	450	1614	1	4	14	19	30666	TMVP-LT
32	648	480	422	1550	1	4	10	15	23250	TMVP-STD-2
33	778	480	486	1744	1	4	12	17	29648	TMVP-LT
44	1176	968	742	2886	1	5	16	22	63492	TMVP-4
45	1050	1007	730	2787	1	5	14	20	55740	TMVP-3
48	1080	1039	762	2881	1	5	12	18	51858	TMVP-3
49	1274	1039	858	3171	1	5	14	20	63420	TMVP-LT
63	2000	1459	1292	4751	1	5	16	22	104522	TMVP-3
64	1512	1986	1134	4632	1	6	14	21	97272	TMVP-4
65	1770	1986	1262	5018	1	6	16	23	115414	TMVP-LT

Table A.6 - continued from previous page

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
67	2786	1291	1656	5733	1	5	18	24	137592	TMVP-LT
77	2714	2461	1804	6979	1	6	20	27	188433	TMVP-LT
91	3512	2712	2340	8564	1	6	18	25	214100	TMVP-LT
96	2520	3969	2058	8547	1	7	16	24	205128	TMVP-4
99	3890	3118	2661	9669	1	6	16	23	222387	TMVP-3
104	4452	3196	2814	10462	1	6	18	25	261550	TMVP-4
121	4892	4807	3390	13089	1	7	20	28	366492	TMVP-LT
127	6506	3916	4086	14508	1	6	20	27	391716	TMVP-LT
127	6506	3916	4086	14508	1	6	20	27	391716	TMVP-LT
128	4536	5413	3402	13351	1	7	16	24	320424	TMVP-4
129	5050	5413	3658	14121	1	7	18	26	367146	TMVP-LT
137	6468	5571	4388	16427	1	7	20	28	459956	TMVP-LT
143	8586	5636	5452	19674	1	7	26	34	668916	TMVP-LT
169	9074	6903	5880	21857	1	7	22	30	655710	TMVP-LT
188	9926	10077	7042	27045	1	8	24	33	892485	TMVP-4
189	10000	8673	6901	25574	1	7	20	28	716072	TMVP-3
192	7560	10366	6006	23932	1	8	18	27	646164	TMVP-4
197	9704	10431	7084	27219	1	8	22	31	843789	TMVP-LT
256	10584	18035	8834	37453	1	9	20	30	1123590	TMVP-4

Table A.7: TMVP Space Optimization Result (No Padding, $M = 4A_s = 4A_d$, $2D_m = 2D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	4	0	0	4	1	0	0	1	4	Initial Condition
2	12	3	2	17	1	1	2	4	68	TMVP-STD-2
3	24	8	6	38	1	1	4	6	228	TMVP-STD-3
4	36	16	10	62	1	2	4	7	434	TMVP-STD-2
5	72	16	18	106	1	2	6	9	954	TMVP-LT
6	72	35	24	131	1	2	6	9	1179	TMVP-STD-2
7	124	35	36	195	1	2	8	11	2145	TMVP-LT
8	108	63	38	209	1	3	6	10	2090	TMVP-STD-2
16	324	220	130	674	1	4	8	13	8762	TMVP-STD-2
31	1324	547	450	2321	1	5	14	20	46420	TMVP-LT
32	756	934	378	2068	1	6	12	19	39292	TMVP-4
33	1016	934	442	2392	1	6	14	21	50232	TMVP-LT
44	2100	1157	742	3999	1	6	16	23	91977	TMVP-4
45	1800	1232	730	3762	1	6	14	21	79002	TMVP-3
48	1620	1444	762	3826	1	6	12	19	72694	TMVP-3
49	2008	1444	858	4310	1	6	14	21	90510	TMVP-LT
63	3100	2059	1292	6451	1	6	16	23	148373	TMVP-3

Table A.7 - continued from previous page

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
64	2268	2553	1134	5955	1	7	14	22	131010	TMVP-4
65	2784	2553	1262	6599	1	7	16	24	158376	TMVP-LT
67	3840	2553	1524	7917	1	7	20	28	221676	TMVP-LT
77	4168	3301	1804	9273	1	7	20	28	259644	TMVP-LT
91	6124	3387	2340	11851	1	7	18	26	308126	TMVP-LT
96	3780	4914	2058	10752	1	8	16	25	268800	TMVP-4
99	5080	5388	2441	12909	1	8	18	27	348543	TMVP-3
104	5880	5380	2688	13948	1	8	18	27	376596	TMVP-4
121	8524	5752	3390	17666	1	8	20	29	512314	TMVP-LT
127	10312	5716	4086	20114	1	7	20	28	563192	TMVP-LT
127	10312	5716	4086	20114	1	7	20	28	563192	TMVP-LT
128	5292	8591	3094	16977	1	9	18	28	475356	TMVP-4
129	6320	8591	3350	18261	1	9	20	30	547830	TMVP-LT
137	10668	7272	4388	22328	1	8	20	29	647512	TMVP-LT
143	14952	7186	5455	27593	1	8	26	35	965755	TMVP-LT
169	14368	9423	5880	29671	1	8	22	31	919801	TMVP-LT
188	17752	11652	7042	36446	1	9	24	34	1239164	TMVP-4
189	15500	11673	6901	34074	1	8	20	29	988146	TMVP-3
192	11340	13201	6006	30547	1	9	18	28	855316	TMVP-4
197	15628	13266	7084	35978	1	9	22	32	1151296	TMVP-LT
256	15876	22004	8834	46714	1	10	20	31	1448134	TMVP-4

Table A.8: TMVP Space Optimization Result (No Padding, $M = 4A_s = 2A_d$, $2D_m = 2D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	4	0	0	4	1	0	0	1	4	Initial Condition
2	12	3	4	19	1	1	2	4	76	TMVP-STD-2
3	24	8	12	44	1	1	4	6	264	TMVP-STD-3
4	36	16	20	72	1	2	4	7	504	TMVP-STD-2
5	72	16	36	124	1	2	6	9	1116	TMVP-LT
6	72	35	48	155	1	2	6	9	1395	TMVP-STD-2
7	124	35	72	231	1	2	8	11	2541	TMVP-LT
8	108	63	76	247	1	3	6	10	2470	TMVP-STD-2
16	324	220	260	804	1	4	8	13	10452	TMVP-STD-2
31	1324	547	900	2771	1	5	14	20	55420	TMVP-LT
32	756	934	756	2446	1	6	12	19	46474	TMVP-4
33	1016	934	884	2834	1	6	14	21	59514	TMVP-LT
44	2100	1157	1484	4741	1	6	16	23	109043	TMVP-4
45	1800	1232	1460	4492	1	6	14	21	94332	TMVP-3
48	1260	1873	1428	4561	1	7	14	22	100342	TMVP-4

Table A.8 - continued from previous page

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
49	1648	1873	1620	5141	1	7	16	24	123384	TMVP-LT
63	3100	2059	2584	7743	1	6	16	23	178089	TMVP-3
64	2268	2553	2268	7089	1	7	14	22	155958	TMVP-4
65	2784	2553	2524	7861	1	7	16	24	188664	TMVP-LT
67	3840	2553	3048	9441	1	7	20	28	264348	TMVP-LT
77	4168	3301	3608	11077	1	7	20	28	310156	TMVP-LT
91	6124	3387	4680	14191	1	7	18	26	368966	TMVP-LT
96	3780	4914	4116	12810	1	8	16	25	320250	TMVP-4
99	5080	5388	4882	15350	1	8	18	27	414450	TMVP-3
104	5880	5380	5376	16636	1	8	18	27	449172	TMVP-4
121	8524	5752	6780	21056	1	8	20	29	610624	TMVP-LT
127	10312	5716	8172	24200	1	7	20	28	677600	TMVP-LT
127	10312	5716	8172	24200	1	7	20	28	677600	TMVP-LT
128	5292	8591	6188	20071	1	9	18	28	561988	TMVP-4
129	6320	8591	6700	21611	1	9	20	30	648330	TMVP-LT
137	10668	7272	8776	26716	1	8	20	29	774764	TMVP-LT
143	14952	7186	10910	33048	1	8	26	35	1156680	TMVP-LT
169	14368	9423	11760	35551	1	8	22	31	1102081	TMVP-LT
188	17752	11652	14084	43488	1	9	24	34	1478592	TMVP-4
189	15500	11673	13802	40975	1	8	20	29	1188275	TMVP-3
192	8820	16204	11340	36364	1	10	20	31	1127284	TMVP-4
197	13108	16269	13496	42873	1	10	24	35	1500555	TMVP-LT
256	15876	22004	17668	55548	1	10	20	31	1721988	TMVP-4

Table A.9: TMVP Space x Delay Optimization Result (No Padding, $M = A_s = A_d$, $D_m = D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	1	0	0	1	1	0	0	1	1	Initial Condition
2	4	0	2	6	1	0	1	2	12	TMVP-SB
3	9	0	6	15	1	0	2	3	45	TMVP-LT
4	16	0	12	28	1	0	2	3	84	TMVP-SB
5	25	0	20	45	1	0	3	4	180	TMVP-LT
6	36	0	30	66	1	0	3	4	264	TMVP-SB
7	49	0	42	91	1	0	4	5	455	TMVP-LT
8	64	0	56	120	1	0	3	4	480	TMVP-SB
16	192	31	184	407	1	1	4	6	2442	TMVP-STD-2
31	511	191	540	1242	1	2	7	10	12420	TMVP-LT
32	576	156	584	1316	1	2	5	8	10528	TMVP-STD-2
33	641	156	648	1445	1	2	6	9	13005	TMVP-LT
44	1089	216	1100	2405	1	2	7	10	24050	TMVP-STD-2

Table A.9 - continued from previous page

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
45	900	398	990	2288	1	2	7	10	22880	TMVP-STD-3
48	1152	308	1200	2660	1	2	6	9	23940	TMVP-STD-2
49	1249	308	1296	2853	1	2	7	10	28530	TMVP-LT
63	1470	764	1617	3851	1	3	8	12	46212	TMVP-3
64	1728	595	1816	4139	1	3	6	10	41390	TMVP-STD-2
65	1857	595	1944	4396	1	3	7	11	48356	TMVP-LT
67	2056	599	2142	4797	1	3	8	12	57564	TMVP-LT
77	2430	853	2610	5893	1	3	9	13	76609	TMVP-LT
91	2431	1607	2790	6828	1	4	9	14	95592	TMVP-LT
96	2880	1476	3144	7500	1	4	7	12	90000	TMVP-3
99	3205	1498	3471	8174	1	4	8	13	106262	TMVP-3
104	3267	1920	3734	8921	1	4	8	13	115973	TMVP-STD-2
121	4291	2315	4860	11466	1	4	9	14	160524	TMVP-LT
127	4663	2261	5166	12090	1	4	10	15	181350	TMVP-LT
127	4663	2261	5166	12090	1	4	10	15	181350	TMVP-LT
128	5184	2040	5576	12800	1	4	7	12	153600	TMVP-STD-2
129	5441	2040	5832	13313	1	4	8	13	173069	TMVP-LT
137	6348	2116	6750	15214	1	4	9	14	212996	TMVP-LT
143	7404	2140	7806	17350	1	4	11	16	277600	TMVP-LT
169	6952	3819	7728	18499	1	5	10	16	295984	TMVP-LT
188	8226	5262	9497	22985	1	5	11	17	390745	TMVP-STD-2
189	7350	5198	8526	21074	1	5	10	16	337184	TMVP-3
192	8640	4375	9528	22543	1	5	8	14	315602	TMVP-3
197	11634	3748	12546	27928	1	4	10	15	418920	TMVP-LT
256	12096	8298	13608	34002	1	6	9	16	544032	TMVP-4

Table A.10: TMVP Space x Delay Optimization Result (No Padding, $M = 2A_s = 2A_d$, $D_m = D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	2	0	0	2	1	0	0	1	2	Initial Condition
2	8	0	2	10	1	0	1	2	20	TMVP-SB
3	18	0	6	24	1	0	2	3	72	TMVP-LT
4	32	0	12	44	1	0	2	3	132	TMVP-SB
5	50	0	20	70	1	0	3	4	280	TMVP-LT
6	72	0	30	102	1	0	3	4	408	TMVP-SB
7	98	0	42	140	1	0	4	5	700	TMVP-LT
8	128	0	56	184	1	0	3	4	736	TMVP-SB
16	384	31	184	599	1	1	4	6	3594	TMVP-STD-2
31	1022	191	540	1753	1	2	7	10	17530	TMVP-LT
32	1152	156	584	1892	1	2	5	8	15136	TMVP-STD-2

Table A.10 - continued from previous page

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
33	1282	156	648	2086	1	2	6	9	18774	TMVP-LT
44	1728	387	920	3035	1	3	7	11	33385	TMVP-STD-2
45	1500	542	855	2897	1	3	7	11	31867	TMVP-3
48	1728	551	984	3263	1	3	6	10	32630	TMVP-STD-2
49	1922	551	1080	3553	1	3	7	11	39083	TMVP-LT
63	2940	764	1617	5321	1	3	8	12	63852	TMVP-3
64	3456	595	1816	5867	1	3	6	10	58670	TMVP-STD-2
65	3714	595	1944	6253	1	3	7	11	68783	TMVP-LT
67	4112	599	2142	6853	1	3	8	12	82236	TMVP-LT
77	4860	853	2610	8323	1	3	9	13	108199	TMVP-LT
91	4862	1607	2790	9259	1	4	9	14	129626	TMVP-LT
96	5184	1844	3048	10076	1	4	7	12	120912	TMVP-STD-2
99	6410	1498	3471	11379	1	4	8	13	147927	TMVP-3
104	6534	1920	3734	12188	1	4	8	13	158444	TMVP-STD-2
121	7232	2707	4170	14109	1	5	9	15	211635	TMVP-LT
127	9326	2261	5166	16753	1	4	10	15	251295	TMVP-LT
127	9326	2261	5166	16753	1	4	10	15	251295	TMVP-LT
128	10368	2040	5576	17984	1	4	7	12	215808	TMVP-STD-2
129	10882	2040	5832	18754	1	4	8	13	243802	TMVP-LT
137	8588	3692	5132	17412	1	5	11	17	296004	TMVP-LT
143	10476	3736	6080	20292	1	5	13	19	385548	TMVP-LT
169	13904	3819	7728	25451	1	5	10	16	407216	TMVP-LT
188	13076	6822	7917	27815	1	6	12	19	528485	TMVP-4
189	14700	5198	8526	28424	1	5	10	16	454784	TMVP-3
192	12096	6950	7560	26606	1	6	9	16	425696	TMVP-4
197	14240	7015	8638	29893	1	6	11	18	538074	TMVP-LT
256	24192	8298	13608	46098	1	6	9	16	737568	TMVP-4

Table A.11: TMVP Space x Delay Optimization Result (No Padding, $M = 4A_s = 4A_d$, $D_m = D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	4	0	0	4	1	0	0	1	4	Initial Condition
2	16	0	2	18	1	0	1	2	36	TMVP-SB
3	36	0	6	42	1	0	2	3	126	TMVP-LT
4	64	0	12	76	1	0	2	3	228	TMVP-SB
5	100	0	20	120	1	0	3	4	480	TMVP-LT
6	96	17	24	137	1	1	3	5	685	TMVP-STD-3
7	148	17	36	201	1	1	4	6	1206	TMVP-LT
8	256	0	56	312	1	0	3	4	1248	TMVP-SB
16	768	31	184	983	1	1	4	6	5898	TMVP-STD-2

Table A.11 - continued from previous page

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
31	2044	191	540	2775	1	2	7	10	27750	TMVP-LT
32	2304	156	584	3044	1	2	5	8	24352	TMVP-STD-2
33	2564	156	648	3368	1	2	6	9	30312	TMVP-LT
44	3456	387	920	4763	1	3	7	11	52393	TMVP-STD-2
45	3000	542	855	4397	1	3	7	11	48367	TMVP-3
48	3456	551	984	4991	1	3	6	10	49910	TMVP-STD-2
49	3844	551	1080	5475	1	3	7	11	60225	TMVP-LT
63	4440	1274	1437	7151	1	4	8	13	92963	TMVP-3
64	6912	595	1816	9323	1	3	6	10	93230	TMVP-STD-2
65	7428	595	1944	9967	1	3	7	11	109637	TMVP-LT
67	6292	976	1746	9014	1	4	9	14	126196	TMVP-LT
77	7128	1771	2286	11185	1	4	9	14	156590	TMVP-LT
91	9724	1607	2790	14121	1	4	9	14	197694	TMVP-LT
96	10368	1844	3048	15260	1	4	7	12	183120	TMVP-STD-2
99	12820	1498	3471	17789	1	4	8	13	231257	TMVP-3
104	10164	2755	3066	15985	1	5	9	15	239775	TMVP-4
121	14464	2707	4170	21341	1	5	9	15	320115	TMVP-LT
127	12112	4366	4086	20564	1	6	10	17	349588	TMVP-LT
127	12112	4366	4086	20564	1	6	10	17	349588	TMVP-LT
128	16128	3145	4536	23809	1	5	8	14	333326	TMVP-4
129	12800	4388	4261	21449	1	6	10	17	364633	TMVP-3
137	17176	3692	5132	26000	1	5	11	17	442000	TMVP-LT
143	20952	3736	6080	30768	1	5	13	19	584592	TMVP-LT
169	16888	7533	5880	30301	1	7	11	19	575719	TMVP-LT
188	26152	6822	7917	40891	1	6	12	19	776929	TMVP-4
189	22200	7748	7626	37574	1	6	10	17	638758	TMVP-3
192	24192	6950	7560	38702	1	6	9	16	619232	TMVP-4
197	28480	7015	8638	44133	1	6	11	18	794394	TMVP-LT
256	48384	8298	13608	70290	1	6	9	16	1124640	TMVP-4

Table A.12: TMVP Space x Delay Optimization Result (No Padding, $M = 4A_s = 2A_d$, $D_m = D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	4	0	0	4	1	0	0	1	4	Initial Condition
2	16	0	4	20	1	0	1	2	40	TMVP-SB
3	36	0	12	48	1	0	2	3	144	TMVP-LT
4	64	0	24	88	1	0	2	3	264	TMVP-SB
5	100	0	40	140	1	0	3	4	560	TMVP-LT
6	96	17	48	161	1	1	3	5	805	TMVP-STD-3
7	148	17	72	237	1	1	4	6	1422	TMVP-LT

Table A.12 - continued from previous page

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
8	256	0	112	368	1	0	3	4	1472	TMVP-SB
16	768	31	368	1167	1	1	4	6	7002	TMVP-STD-2
31	1744	307	960	3011	1	3	7	11	33121	TMVP-LT
32	2304	156	1168	3628	1	2	5	8	29024	TMVP-STD-2
33	2564	156	1296	4016	1	2	6	9	36144	TMVP-LT
44	3456	387	1840	5683	1	3	7	11	62513	TMVP-STD-2
45	3000	542	1710	5252	1	3	7	11	57772	TMVP-3
48	3456	551	1968	5975	1	3	6	10	59750	TMVP-STD-2
49	3844	551	2160	6555	1	3	7	11	72105	TMVP-LT
63	4440	1274	2874	8588	1	4	8	13	111644	TMVP-3
64	6912	595	3632	11139	1	3	6	10	111390	TMVP-STD-2
65	7428	595	3888	11911	1	3	7	11	131021	TMVP-LT
67	6292	976	3492	10760	1	4	9	14	150640	TMVP-LT
77	7128	1771	4572	13471	1	4	9	14	188594	TMVP-LT
91	8224	2187	4980	15391	1	5	9	15	230865	TMVP-LT
96	10368	1844	6096	18308	1	4	7	12	219696	TMVP-STD-2
99	12820	1498	6942	21260	1	4	8	13	276380	TMVP-3
104	10164	2755	6132	19051	1	5	9	15	285765	TMVP-4
121	14464	2707	8340	25511	1	5	9	15	382665	TMVP-LT
127	12112	4366	8172	24650	1	6	10	17	419050	TMVP-LT
127	12112	4366	8172	24650	1	6	10	17	419050	TMVP-LT
128	16128	3145	9072	28345	1	5	8	14	396830	TMVP-4
129	12800	4388	8522	25710	1	6	10	17	437070	TMVP-3
137	17176	3692	10264	31132	1	5	11	17	529244	TMVP-LT
143	20952	3736	12160	36848	1	5	13	19	700112	TMVP-LT
169	16888	7533	11760	36181	1	7	11	19	687439	TMVP-LT
188	26152	6822	15834	48808	1	6	12	19	927352	TMVP-4
189	22200	7748	15252	45200	1	6	10	17	768400	TMVP-3
192	24192	6950	15120	46262	1	6	9	16	740192	TMVP-4
197	28480	7015	17276	52771	1	6	11	18	949878	TMVP-LT
256	48384	8298	27216	83898	1	6	9	16	1342368	TMVP-4

Table A.13: TMVP Space x Delay Optimization Result (No Padding, $M = A_s = A_d$, $2D_m = 2D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	1	0	0	1	1	0	0	1	1	Initial Condition
2	4	0	2	6	1	0	2	3	18	TMVP-SB
3	9	0	6	15	1	0	4	5	75	TMVP-LT
4	16	0	12	28	1	0	4	5	140	TMVP-SB
5	25	0	20	45	1	0	6	7	315	TMVP-LT

Table A.13 - continued from previous page

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
6	36	0	30	66	1	0	6	7	462	TMVP-SB
7	49	0	42	91	1	0	8	9	819	TMVP-LT
8	64	0	56	120	1	0	6	7	840	TMVP-SB
16	192	31	184	407	1	1	8	10	4070	TMVP-STD-2
31	511	191	540	1242	1	2	14	17	21114	TMVP-LT
32	576	156	584	1316	1	2	10	13	17108	TMVP-STD-2
33	641	156	648	1445	1	2	12	15	21675	TMVP-LT
44	864	387	920	2171	1	3	14	18	39078	TMVP-STD-2
45	750	542	855	2147	1	3	14	18	38646	TMVP-3
48	864	551	984	2399	1	3	12	16	38384	TMVP-STD-2
49	961	551	1080	2592	1	3	14	18	46656	TMVP-LT
63	1470	764	1617	3851	1	3	16	20	77020	TMVP-3
64	1728	595	1816	4139	1	3	12	16	66224	TMVP-STD-2
65	1857	595	1944	4396	1	3	14	18	79128	TMVP-LT
67	2056	599	2142	4797	1	3	16	20	95940	TMVP-LT
77	2430	853	2610	5893	1	3	18	22	129646	TMVP-LT
91	2431	1607	2790	6828	1	4	18	23	157044	TMVP-LT
96	2592	1844	3048	7484	1	4	14	19	142196	TMVP-STD-2
99	3205	1498	3471	8174	1	4	16	21	171654	TMVP-3
104	3267	1920	3734	8921	1	4	16	21	187341	TMVP-STD-2
121	3616	2707	4170	10493	1	5	18	24	251832	TMVP-LT
127	4663	2261	5166	12090	1	4	20	25	302250	TMVP-LT
127	4663	2261	5166	12090	1	4	20	25	302250	TMVP-LT
128	5184	2040	5576	12800	1	4	14	19	243200	TMVP-STD-2
129	5441	2040	5832	13313	1	4	16	21	279573	TMVP-LT
137	6348	2116	6750	15214	1	4	18	23	349922	TMVP-LT
143	7404	2140	7806	17350	1	4	22	27	468450	TMVP-LT
169	6952	3819	7728	18499	1	5	20	26	480974	TMVP-LT
188	8226	5262	9497	22985	1	5	22	28	643580	TMVP-STD-2
189	7350	5198	8526	21074	1	5	20	26	547924	TMVP-3
192	8640	4375	9528	22543	1	5	16	22	495946	TMVP-3
197	7120	7015	8638	22773	1	6	22	29	660417	TMVP-LT
256	12096	8298	13608	34002	1	6	18	25	850050	TMVP-4

Table A.14: TMVP Space x Delay Optimization Result (No Padding, $M = 2A_s = 2A_d$, $2D_m = 2D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	2	0	0	2	1	0	0	1	2	Initial Condition
2	8	0	2	10	1	0	2	3	30	TMVP-SB
3	18	0	6	24	1	0	4	5	120	TMVP-LT

Table A.14 - continued from previous page

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
4	32	0	12	44	1	0	4	5	220	TMVP-SB
5	50	0	20	70	1	0	6	7	490	TMVP-LT
6	54	11	24	89	1	1	6	8	712	TMVP-STD-2
7	80	11	36	127	1	1	8	10	1270	TMVP-LT
8	96	15	44	155	1	1	6	8	1240	TMVP-STD-2
16	288	76	148	512	1	2	8	11	5632	TMVP-STD-2
31	1022	191	540	1753	1	2	14	17	29801	TMVP-LT
32	864	291	476	1631	1	3	10	14	22834	TMVP-STD-2
33	994	291	540	1825	1	3	12	16	29200	TMVP-LT
44	1728	387	920	3035	1	3	14	18	54630	TMVP-STD-2
45	1500	542	855	2897	1	3	14	18	52146	TMVP-3
48	1440	724	852	3016	1	4	12	17	51272	TMVP-3
49	1634	724	948	3306	1	4	14	19	62814	TMVP-LT
63	2400	1094	1437	4931	1	4	16	21	103551	TMVP-3
64	2592	1000	1492	5084	1	4	12	17	86428	TMVP-STD-2
65	2850	1000	1620	5470	1	4	14	19	103930	TMVP-LT
67	3248	1004	1818	6070	1	4	16	21	127470	TMVP-LT
77	3888	1393	2286	7567	1	4	18	23	174041	TMVP-LT
91	4862	1607	2790	9259	1	4	18	23	212957	TMVP-LT
96	4320	2151	2604	9075	1	5	14	20	181500	TMVP-3
99	4970	2173	2931	10074	1	5	16	22	221628	TMVP-3
104	6534	1920	3734	12188	1	4	16	21	255948	TMVP-STD-2
121	7232	2707	4170	14109	1	5	18	24	338616	TMVP-LT
127	6506	3916	4086	14508	1	6	20	27	391716	TMVP-LT
127	6506	3916	4086	14508	1	6	20	27	391716	TMVP-LT
128	7776	3255	4604	15635	1	5	14	20	312700	TMVP-STD-2
129	8290	3255	4860	16405	1	5	16	22	360910	TMVP-LT
137	7980	4248	4766	16994	1	6	20	27	458838	TMVP-LT
143	12216	3355	6834	22405	1	5	22	28	627340	TMVP-LT
169	11474	5304	6918	23696	1	6	20	27	639792	TMVP-LT
188	13076	6822	7917	27815	1	6	24	31	862265	TMVP-4
189	12000	6848	7626	26474	1	6	20	27	714798	TMVP-3
192	12960	6400	7908	27268	1	6	16	23	627164	TMVP-3
197	12224	8226	7714	28164	1	7	22	30	844920	TMVP-LT
256	18144	11133	11340	40617	1	7	18	26	1056042	TMVP-4

Table A.15: TMVP Space x Delay Optimization Result (No Padding, $M = 4A_s = 4A_d$, $2D_m = 2D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	4	0	0	4	1	0	0	1	4	Initial Condition

Table A.15 - continued from previous page

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
2	16	0	2	18	1	0	2	3	54	TMVP-SB
3	36	0	6	42	1	0	4	5	210	TMVP-LT
4	64	0	12	76	1	0	4	5	380	TMVP-SB
5	100	0	20	120	1	0	6	7	840	TMVP-LT
6	96	17	24	137	1	1	6	8	1096	TMVP-STD-3
7	148	17	36	201	1	1	8	10	2010	TMVP-LT
8	192	15	44	251	1	1	6	8	2008	TMVP-STD-2
16	576	76	148	800	1	2	8	11	8800	TMVP-STD-2
31	1744	307	480	2531	1	3	14	18	45558	TMVP-LT
32	1728	291	476	2495	1	3	10	14	34930	TMVP-STD-2
33	1988	291	540	2819	1	3	12	16	45104	TMVP-LT
44	3456	387	920	4763	1	3	14	18	85734	TMVP-STD-2
45	3000	542	855	4397	1	3	14	18	79146	TMVP-3
48	2592	902	876	4370	1	4	12	17	74290	TMVP-STD-2
49	2980	902	972	4854	1	4	14	19	92226	TMVP-LT
63	3700	1609	1292	6601	1	5	16	22	145222	TMVP-3
64	5184	1000	1492	7676	1	4	12	17	130492	TMVP-STD-2
65	5700	1000	1620	8320	1	4	14	19	158080	TMVP-LT
67	6496	1004	1818	9318	1	4	16	21	195678	TMVP-LT
77	7128	1771	2286	11185	1	4	18	23	257255	TMVP-LT
91	8224	2187	2490	12901	1	5	18	24	309624	TMVP-LT
96	8640	2151	2604	13395	1	5	14	20	267900	TMVP-3
99	9940	2173	2931	15044	1	5	16	22	330968	TMVP-3
104	8148	3574	2814	14536	1	6	18	25	363400	TMVP-4
121	14464	2707	4170	21341	1	5	18	24	512184	TMVP-LT
127	12112	4366	4086	20564	1	6	20	27	555228	TMVP-LT
127	12112	4366	4086	20564	1	6	20	27	555228	TMVP-LT
128	12096	4090	3780	19966	1	6	16	23	459218	TMVP-4
129	13124	4090	4036	21250	1	6	18	25	531250	TMVP-LT
137	15960	4248	4766	24974	1	6	20	27	674298	TMVP-LT
143	24432	3355	6834	34621	1	5	22	28	969388	TMVP-LT
169	16888	7533	5880	30301	1	7	22	30	909030	TMVP-LT
188	26152	6822	7917	40891	1	6	24	31	1267621	TMVP-4
189	18500	9423	6901	34824	1	7	20	28	975072	TMVP-3
192	18144	9407	6804	34355	1	7	18	26	893230	TMVP-4
197	22432	9472	7882	39786	1	7	22	30	1193580	TMVP-LT
256	36288	11133	11340	58761	1	7	18	26	1527786	TMVP-4

Table A.16: TMVP Space x Delay Optimization Result (No Padding, $M = 4A_s = 2A_d$, $2D_m = 2D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	4	0	0	4	1	0	0	1	4	Initial Condition
2	16	0	4	20	1	0	2	3	60	TMVP-SB
3	36	0	12	48	1	0	4	5	240	TMVP-LT
4	64	0	24	88	1	0	4	5	440	TMVP-SB
5	100	0	40	140	1	0	6	7	980	TMVP-LT
6	96	17	48	161	1	1	6	8	1288	TMVP-STD-3
7	148	17	72	237	1	1	8	10	2370	TMVP-LT
8	192	15	88	295	1	1	6	8	2360	TMVP-STD-2
16	576	76	296	948	1	2	8	11	10428	TMVP-STD-2
31	1744	307	960	3011	1	3	14	18	54198	TMVP-LT
32	1728	291	952	2971	1	3	10	14	41594	TMVP-STD-2
33	1988	291	1080	3359	1	3	12	16	53744	TMVP-LT
44	3456	387	1840	5683	1	3	14	18	102294	TMVP-STD-2
45	3000	542	1710	5252	1	3	14	18	94536	TMVP-3
48	2592	902	1752	5246	1	4	12	17	89182	TMVP-STD-2
49	2980	902	1944	5826	1	4	14	19	110694	TMVP-LT
63	3700	1609	2584	7893	1	5	16	22	173646	TMVP-3
64	5184	1000	2984	9168	1	4	12	17	155856	TMVP-STD-2
65	5700	1000	3240	9940	1	4	14	19	188860	TMVP-LT
67	6496	1004	3636	11136	1	4	16	21	233856	TMVP-LT
77	5008	2671	3608	11287	1	6	20	27	304749	TMVP-LT
91	8224	2187	4980	15391	1	5	18	24	369384	TMVP-LT
96	8640	2151	5208	15999	1	5	14	20	319980	TMVP-3
99	9940	2173	5862	17975	1	5	16	22	395450	TMVP-3
104	8148	3574	5628	17350	1	6	18	25	433750	TMVP-4
121	14464	2707	8340	25511	1	5	18	24	612264	TMVP-LT
127	12112	4366	8172	24650	1	6	20	27	665550	TMVP-LT
127	12112	4366	8172	24650	1	6	20	27	665550	TMVP-LT
128	12096	4090	7560	23746	1	6	16	23	546158	TMVP-4
129	13124	4090	8072	25286	1	6	18	25	632150	TMVP-LT
137	15960	4248	9532	29740	1	6	20	27	802980	TMVP-LT
143	24432	3355	13668	41455	1	5	22	28	1160740	TMVP-LT
169	16888	7533	11760	36181	1	7	22	30	1085430	TMVP-LT
188	26152	6822	15834	48808	1	6	24	31	1513048	TMVP-4
189	18500	9423	13802	41725	1	7	20	28	1168300	TMVP-3
192	18144	9407	13608	41159	1	7	18	26	1070134	TMVP-4
197	22432	9472	15764	47668	1	7	22	30	1430040	TMVP-LT
256	36288	11133	22680	70101	1	7	18	26	1822626	TMVP-4

Table A.17: TMVP Space Optimization Result (Padding Allowed, $M = A_s = A_d$, $D_m = D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	1	0	0	1	1	0	0	1	1	Initial Condition
2	4	0	2	6	1	0	1	2	12	TMVP-SB
3	9	0	6	15	1	0	2	3	45	TMVP-SB
4	16	0	12	28	1	0	2	3	84	TMVP-SB
5	25	0	20	45	1	0	3	4	180	TMVP-SB
6	27	11	24	62	1	1	3	5	310	TMVP-STD-2
7	40	11	36	87	1	1	4	6	522	TMVP-LT
8	48	15	44	107	1	1	3	5	535	TMVP-STD-2
16	144	76	148	368	1	2	4	7	2576	TMVP-STD-2
31	432	291	476	1199	1	3	5	9	10791	TMVP-STD-2
32	432	291	476	1199	1	3	5	9	10791	TMVP-STD-2
33	497	291	540	1328	1	3	6	10	13280	TMVP-LT
44	750	542	855	2147	1	3	7	11	23617	TMVP-3
45	750	542	855	2147	1	3	7	11	23617	TMVP-3
48	720	724	852	2296	1	4	6	11	25256	TMVP-3
49	817	724	948	2489	1	4	7	12	29868	TMVP-LT
63	1200	1094	1437	3731	1	4	8	13	48503	TMVP-3
64	1296	1000	1492	3788	1	4	6	11	41668	TMVP-STD-2
65	1440	976	1614	4030	1	4	8	13	52390	TMVP-3
67	1573	976	1746	4295	1	4	9	14	60130	TMVP-LT
77	1590	1659	1932	5181	1	5	8	14	72534	TMVP-3
91	2056	2187	2490	6733	1	5	9	15	100995	TMVP-LT
96	2160	2151	2604	6915	1	5	7	13	89895	TMVP-3
99	2485	2173	2931	7589	1	5	8	14	106246	TMVP-3
104	2025	3442	2765	8232	1	6	8	15	123480	TMVP-3
121	3616	2707	4170	10493	1	5	9	15	157395	TMVP-LT
127	3024	4090	3780	10894	1	6	8	15	163410	TMVP-4
127	3024	4090	3780	10894	1	6	8	15	163410	TMVP-4
128	3024	4090	3780	10894	1	6	8	15	163410	TMVP-4
129	3281	4090	4036	11407	1	6	9	16	182512	TMVP-LT
137	3600	4624	4582	12806	1	6	8	15	192090	TMVP-3
143	3600	4668	4596	12864	1	6	8	15	192960	TMVP-3
169	4537	6903	5880	17320	1	7	11	19	329080	TMVP-LT
188	5040	8096	6622	19758	1	7	9	17	335886	TMVP-4
189	5040	8161	6636	19837	1	7	9	17	337229	TMVP-4
192	5040	8161	6636	19837	1	7	9	17	337229	TMVP-4
197	6112	8226	7714	22052	1	7	11	19	418988	TMVP-LT
256	9072	11133	11340	31545	1	7	9	17	536265	TMVP-4

Table A.18: TMVP Space Optimization Result (Padding Allowed, $M = 2A_s = 2A_d$, $D_m = D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	2	0	0	2	1	0	0	1	2	Initial Condition
2	8	0	2	10	1	0	1	2	20	TMVP-SB
3	18	0	6	24	1	0	2	3	72	TMVP-SB
4	24	7	10	41	1	1	2	4	164	TMVP-STD-2
5	42	7	18	67	1	1	3	5	335	TMVP-LT
6	54	11	24	89	1	1	3	5	445	TMVP-STD-2
7	80	11	36	127	1	1	4	6	762	TMVP-LT
8	72	36	38	146	1	2	3	6	876	TMVP-STD-2
16	216	139	130	485	1	3	4	8	3880	TMVP-STD-2
31	648	480	422	1550	1	4	5	10	15500	TMVP-STD-2
32	648	480	422	1550	1	4	5	10	15500	TMVP-STD-2
33	778	480	486	1744	1	4	6	11	19184	TMVP-LT
44	1050	1007	730	2787	1	5	7	13	36231	TMVP-3
45	1050	1007	730	2787	1	5	7	13	36231	TMVP-3
48	1080	1039	762	2881	1	5	6	12	34572	TMVP-3
49	1274	1039	858	3171	1	5	7	13	41223	TMVP-LT
63	1512	1986	1134	4632	1	6	7	14	64848	TMVP-4
64	1512	1986	1134	4632	1	6	7	14	64848	TMVP-4
65	1770	1986	1262	5018	1	6	8	15	75270	TMVP-LT
67	1974	2051	1372	5397	1	6	8	15	80955	TMVP-4
77	2700	2164	1847	6711	1	5	8	14	93954	TMVP-3
91	2520	3904	2044	8468	1	7	8	16	135488	TMVP-4
96	2520	3969	2058	8547	1	7	8	16	136752	TMVP-4
99	3206	4034	2408	9648	1	7	9	17	164016	TMVP-4
104	4050	3442	2765	10257	1	6	8	15	153855	TMVP-3
121	4892	4807	3390	13089	1	7	10	18	235602	TMVP-LT
127	4536	5413	3402	13351	1	7	8	16	213616	TMVP-4
127	4536	5413	3402	13351	1	7	8	16	213616	TMVP-4
128	4536	5413	3402	13351	1	7	8	16	213616	TMVP-4
129	5050	5413	3658	14121	1	7	9	17	240057	TMVP-LT
137	5670	6000	4018	15688	1	7	9	17	266696	TMVP-4
143	5670	6065	4032	15767	1	7	9	17	268039	TMVP-4
169	9074	6903	5880	21857	1	7	11	19	415283	TMVP-LT
188	7560	10301	5992	23853	1	8	9	18	429354	TMVP-4
189	7560	10366	6006	23932	1	8	9	18	430776	TMVP-4
192	7560	10366	6006	23932	1	8	9	18	430776	TMVP-4
197	9704	10431	7084	27219	1	8	11	20	544380	TMVP-LT
256	10584	18035	8834	37453	1	9	10	20	749060	TMVP-4

Table A.19: TMVP Space Optimization Result (Padding Allowed, $M = 4A_s = 4A_d$, $D_m = D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	4	0	0	4	1	0	0	1	4	Initial Condition
2	12	3	2	17	1	1	1	3	51	TMVP-STD-2
3	24	8	6	38	1	1	2	4	152	TMVP-STD-3
4	36	16	10	62	1	2	2	5	310	TMVP-STD-2
5	72	16	18	106	1	2	3	6	636	TMVP-LT
6	72	35	24	131	1	2	3	6	786	TMVP-STD-2
7	124	35	36	195	1	2	4	7	1365	TMVP-LT
8	108	63	38	209	1	3	3	7	1463	TMVP-STD-2
16	324	220	130	674	1	4	4	9	6066	TMVP-STD-2
31	756	934	378	2068	1	6	6	13	26884	TMVP-4
32	756	934	378	2068	1	6	6	13	26884	TMVP-4
33	1016	934	442	2392	1	6	7	14	33488	TMVP-LT
44	1800	1232	730	3762	1	6	7	14	52668	TMVP-3
45	1800	1232	730	3762	1	6	7	14	52668	TMVP-3
48	1620	1444	762	3826	1	6	6	13	49738	TMVP-3
49	2008	1444	858	4310	1	6	7	14	60340	TMVP-LT
63	2268	2553	1134	5955	1	7	7	15	89325	TMVP-4
64	2268	2553	1134	5955	1	7	7	15	89325	TMVP-4
65	2784	2553	1262	6599	1	7	8	16	105584	TMVP-LT
67	2700	2913	1391	7004	1	7	7	15	105060	TMVP-3
77	3000	3964	1772	8736	1	7	8	16	139776	TMVP-3
91	3780	4849	2044	10673	1	8	8	17	181441	TMVP-4
96	3780	4914	2058	10752	1	8	8	17	182784	TMVP-4
99	5152	4979	2408	12539	1	8	9	18	225702	TMVP-4
104	4500	6042	2615	13157	1	8	8	17	223669	TMVP-3
121	5292	8526	3080	16898	1	9	9	19	321062	TMVP-4
127	5292	8591	3094	16977	1	9	9	19	322563	TMVP-4
127	5292	8591	3094	16977	1	9	9	19	322563	TMVP-4
128	5292	8591	3094	16977	1	9	9	19	322563	TMVP-4
129	6320	8591	3350	18261	1	9	10	20	365220	TMVP-LT
137	6300	9640	3808	19748	1	9	9	19	375212	TMVP-4
143	6300	9705	3822	19827	1	9	9	19	376713	TMVP-4
169	12700	10771	6014	29485	1	8	10	19	560215	TMVP-3
188	11340	13136	5992	30468	1	9	9	19	578892	TMVP-4
189	11340	13201	6006	30547	1	9	9	19	580393	TMVP-4
192	11340	13201	6006	30547	1	9	9	19	580393	TMVP-4
197	15628	13266	7084	35978	1	9	11	21	755538	TMVP-LT
256	15876	22004	8834	46714	1	10	10	21	980994	TMVP-4

Table A.20: TMVP Space Optimization Result (Padding Allowed, $M = 4A_s = 2A_d$, $D_m = D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	4	0	0	4	1	0	0	1	4	Initial Condition
2	12	3	4	19	1	1	1	3	57	TMVP-STD-2
3	24	8	12	44	1	1	2	4	176	TMVP-STD-3
4	36	16	20	72	1	2	2	5	360	TMVP-STD-2
5	72	16	36	124	1	2	3	6	744	TMVP-LT
6	72	35	48	155	1	2	3	6	930	TMVP-STD-2
7	124	35	72	231	1	2	4	7	1617	TMVP-LT
8	108	63	76	247	1	3	3	7	1729	TMVP-STD-2
16	324	220	260	804	1	4	4	9	7236	TMVP-STD-2
31	756	934	756	2446	1	6	6	13	31798	TMVP-4
32	756	934	756	2446	1	6	6	13	31798	TMVP-4
33	1016	934	884	2834	1	6	7	14	39676	TMVP-LT
44	1260	1808	1400	4468	1	7	7	15	67020	TMVP-4
45	1800	1232	1460	4492	1	6	7	14	62888	TMVP-3
48	1260	1873	1428	4561	1	7	7	15	68415	TMVP-4
49	1648	1873	1620	5141	1	7	8	16	82256	TMVP-LT
63	2268	2553	2268	7089	1	7	7	15	106335	TMVP-4
64	2268	2553	2268	7089	1	7	7	15	106335	TMVP-4
65	2784	2553	2524	7861	1	7	8	16	125776	TMVP-LT
67	2700	2913	2782	8395	1	7	7	15	125925	TMVP-3
77	3000	3964	3544	10508	1	7	8	16	168128	TMVP-3
91	3780	4849	4088	12717	1	8	8	17	216189	TMVP-4
96	3780	4914	4116	12810	1	8	8	17	217770	TMVP-4
99	5152	4979	4816	14947	1	8	9	18	269046	TMVP-4
104	4500	6042	5230	15772	1	8	8	17	268124	TMVP-3
121	5292	8526	6160	19978	1	9	9	19	379582	TMVP-4
127	5292	8591	6188	20071	1	9	9	19	381349	TMVP-4
127	5292	8591	6188	20071	1	9	9	19	381349	TMVP-4
128	5292	8591	6188	20071	1	9	9	19	381349	TMVP-4
129	6320	8591	6700	21611	1	9	10	20	432220	TMVP-LT
137	6300	9640	7616	23556	1	9	9	19	447564	TMVP-4
143	6300	9705	7644	23649	1	9	9	19	449331	TMVP-4
169	8820	15424	11004	35248	1	10	10	21	740208	TMVP-4
188	8820	16139	11312	36271	1	10	10	21	761691	TMVP-4
189	8820	16204	11340	36364	1	10	10	21	763644	TMVP-4
192	8820	16204	11340	36364	1	10	10	21	763644	TMVP-4
197	13108	16269	13496	42873	1	10	12	23	986079	TMVP-LT
256	15876	22004	17668	55548	1	10	10	21	1166508	TMVP-4

Table A.21: TMVP Space Optimization Result (Padding Allowed, $M = A_s = A_d$, $2D_m = 2D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	1	0	0	1	1	0	0	1	1	Initial Condition
2	4	0	2	6	1	0	2	3	18	TMVP-SB
3	9	0	6	15	1	0	4	5	75	TMVP-SB
4	16	0	12	28	1	0	4	5	140	TMVP-SB
5	25	0	20	45	1	0	6	7	315	TMVP-SB
6	27	11	24	62	1	1	6	8	496	TMVP-STD-2
7	40	11	36	87	1	1	8	10	870	TMVP-LT
8	48	15	44	107	1	1	6	8	856	TMVP-STD-2
16	144	76	148	368	1	2	8	11	4048	TMVP-STD-2
31	432	291	476	1199	1	3	10	14	16786	TMVP-STD-2
32	432	291	476	1199	1	3	10	14	16786	TMVP-STD-2
33	497	291	540	1328	1	3	12	16	21248	TMVP-LT
44	750	542	855	2147	1	3	14	18	38646	TMVP-3
45	750	542	855	2147	1	3	14	18	38646	TMVP-3
48	720	724	852	2296	1	4	12	17	39032	TMVP-3
49	817	724	948	2489	1	4	14	19	47291	TMVP-LT
63	1200	1094	1437	3731	1	4	16	21	78351	TMVP-3
64	1296	1000	1492	3788	1	4	12	17	64396	TMVP-STD-2
65	1440	976	1614	4030	1	4	16	21	84630	TMVP-3
67	1573	976	1746	4295	1	4	18	23	98785	TMVP-LT
77	1590	1659	1932	5181	1	5	16	22	113982	TMVP-3
91	2056	2187	2490	6733	1	5	18	24	161592	TMVP-LT
96	2160	2151	2604	6915	1	5	14	20	138300	TMVP-3
99	2485	2173	2931	7589	1	5	16	22	166958	TMVP-3
104	2025	3442	2765	8232	1	6	16	23	189336	TMVP-3
121	3616	2707	4170	10493	1	5	18	24	251832	TMVP-LT
127	3024	4090	3780	10894	1	6	16	23	250562	TMVP-4
127	3024	4090	3780	10894	1	6	16	23	250562	TMVP-4
128	3024	4090	3780	10894	1	6	16	23	250562	TMVP-4
129	3281	4090	4036	11407	1	6	18	25	285175	TMVP-LT
137	3600	4624	4582	12806	1	6	16	23	294538	TMVP-3
143	3600	4668	4596	12864	1	6	16	23	295872	TMVP-3
169	4537	6903	5880	17320	1	7	22	30	519600	TMVP-LT
188	5040	8096	6622	19758	1	7	18	26	513708	TMVP-4
189	5040	8161	6636	19837	1	7	18	26	515762	TMVP-4
192	5040	8161	6636	19837	1	7	18	26	515762	TMVP-4
197	6112	8226	7714	22052	1	7	22	30	661560	TMVP-LT
256	9072	11133	11340	31545	1	7	18	26	820170	TMVP-4

Table A.22: TMVP Space Optimization Result (Padding Allowed, $M = 2A_s = 2A_d$, $2D_m = 2D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	2	0	0	2	1	0	0	1	2	Initial Condition
2	8	0	2	10	1	0	2	3	30	TMVP-SB
3	18	0	6	24	1	0	4	5	120	TMVP-SB
4	24	7	10	41	1	1	4	6	246	TMVP-STD-2
5	42	7	18	67	1	1	6	8	536	TMVP-LT
6	54	11	24	89	1	1	6	8	712	TMVP-STD-2
7	80	11	36	127	1	1	8	10	1270	TMVP-LT
8	72	36	38	146	1	2	6	9	1314	TMVP-STD-2
16	216	139	130	485	1	3	8	12	5820	TMVP-STD-2
31	648	480	422	1550	1	4	10	15	23250	TMVP-STD-2
32	648	480	422	1550	1	4	10	15	23250	TMVP-STD-2
33	778	480	486	1744	1	4	12	17	29648	TMVP-LT
44	1050	1007	730	2787	1	5	14	20	55740	TMVP-3
45	1050	1007	730	2787	1	5	14	20	55740	TMVP-3
48	1080	1039	762	2881	1	5	12	18	51858	TMVP-3
49	1274	1039	858	3171	1	5	14	20	63420	TMVP-LT
63	1512	1986	1134	4632	1	6	14	21	97272	TMVP-4
64	1512	1986	1134	4632	1	6	14	21	97272	TMVP-4
65	1770	1986	1262	5018	1	6	16	23	115414	TMVP-LT
67	1974	2051	1372	5397	1	6	16	23	124131	TMVP-4
77	2700	2164	1847	6711	1	5	16	22	147642	TMVP-3
91	2520	3904	2044	8468	1	7	16	24	203232	TMVP-4
96	2520	3969	2058	8547	1	7	16	24	205128	TMVP-4
99	3206	4034	2408	9648	1	7	18	26	250848	TMVP-4
104	4050	3442	2765	10257	1	6	16	23	235911	TMVP-3
121	4892	4807	3390	13089	1	7	20	28	366492	TMVP-LT
127	4536	5413	3402	13351	1	7	16	24	320424	TMVP-4
127	4536	5413	3402	13351	1	7	16	24	320424	TMVP-4
128	4536	5413	3402	13351	1	7	16	24	320424	TMVP-4
129	5050	5413	3658	14121	1	7	18	26	367146	TMVP-LT
137	5670	6000	4018	15688	1	7	18	26	407888	TMVP-4
143	5670	6065	4032	15767	1	7	18	26	409942	TMVP-4
169	9074	6903	5880	21857	1	7	22	30	655710	TMVP-LT
188	7560	10301	5992	23853	1	8	18	27	644031	TMVP-4
189	7560	10366	6006	23932	1	8	18	27	646164	TMVP-4
192	7560	10366	6006	23932	1	8	18	27	646164	TMVP-4
197	9704	10431	7084	27219	1	8	22	31	843789	TMVP-LT
256	10584	18035	8834	37453	1	9	20	30	1123590	TMVP-4

Table A.23: TMVP Space Optimization Result (Padding Allowed, $M = 4A_s = 4A_d$, $2D_m = 2D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	4	0	0	4	1	0	0	1	4	Initial Condition
2	12	3	2	17	1	1	2	4	68	TMVP-STD-2
3	24	8	6	38	1	1	4	6	228	TMVP-STD-3
4	36	16	10	62	1	2	4	7	434	TMVP-STD-2
5	72	16	18	106	1	2	6	9	954	TMVP-LT
6	72	35	24	131	1	2	6	9	1179	TMVP-STD-2
7	124	35	36	195	1	2	8	11	2145	TMVP-LT
8	108	63	38	209	1	3	6	10	2090	TMVP-STD-2
16	324	220	130	674	1	4	8	13	8762	TMVP-STD-2
31	756	934	378	2068	1	6	12	19	39292	TMVP-4
32	756	934	378	2068	1	6	12	19	39292	TMVP-4
33	1016	934	442	2392	1	6	14	21	50232	TMVP-LT
44	1800	1232	730	3762	1	6	14	21	79002	TMVP-3
45	1800	1232	730	3762	1	6	14	21	79002	TMVP-3
48	1620	1444	762	3826	1	6	12	19	72694	TMVP-3
49	2008	1444	858	4310	1	6	14	21	90510	TMVP-LT
63	2268	2553	1134	5955	1	7	14	22	131010	TMVP-4
64	2268	2553	1134	5955	1	7	14	22	131010	TMVP-4
65	2784	2553	1262	6599	1	7	16	24	158376	TMVP-LT
67	2700	2913	1391	7004	1	7	14	22	154088	TMVP-3
77	3000	3964	1772	8736	1	7	16	24	209664	TMVP-3
91	3780	4849	2044	10673	1	8	16	25	266825	TMVP-4
96	3780	4914	2058	10752	1	8	16	25	268800	TMVP-4
99	5152	4979	2408	12539	1	8	18	27	338553	TMVP-4
104	4500	6042	2615	13157	1	8	16	25	328925	TMVP-3
121	5292	8526	3080	16898	1	9	18	28	473144	TMVP-4
127	5292	8591	3094	16977	1	9	18	28	475356	TMVP-4
127	5292	8591	3094	16977	1	9	18	28	475356	TMVP-4
128	5292	8591	3094	16977	1	9	18	28	475356	TMVP-4
129	6320	8591	3350	18261	1	9	20	30	547830	TMVP-LT
137	6300	9640	3808	19748	1	9	18	28	552944	TMVP-4
143	6300	9705	3822	19827	1	9	18	28	555156	TMVP-4
169	12700	10771	6014	29485	1	8	20	29	855065	TMVP-3
188	11340	13136	5992	30468	1	9	18	28	853104	TMVP-4
189	11340	13201	6006	30547	1	9	18	28	855316	TMVP-4
192	11340	13201	6006	30547	1	9	18	28	855316	TMVP-4
197	15628	13266	7084	35978	1	9	22	32	1151296	TMVP-LT
256	15876	22004	8834	46714	1	10	20	31	1448134	TMVP-4

Table A.24: TMVP Space Optimization Result (Padding Allowed, $M = 4A_s = 2A_d$, $2D_m = 2D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	4	0	0	4	1	0	0	1	4	Initial Condition
2	12	3	4	19	1	1	2	4	76	TMVP-STD-2
3	24	8	12	44	1	1	4	6	264	TMVP-STD-3
4	36	16	20	72	1	2	4	7	504	TMVP-STD-2
5	72	16	36	124	1	2	6	9	1116	TMVP-LT
6	72	35	48	155	1	2	6	9	1395	TMVP-STD-2
7	124	35	72	231	1	2	8	11	2541	TMVP-LT
8	108	63	76	247	1	3	6	10	2470	TMVP-STD-2
16	324	220	260	804	1	4	8	13	10452	TMVP-STD-2
31	756	934	756	2446	1	6	12	19	46474	TMVP-4
32	756	934	756	2446	1	6	12	19	46474	TMVP-4
33	1016	934	884	2834	1	6	14	21	59514	TMVP-LT
44	1260	1808	1400	4468	1	7	14	22	98296	TMVP-4
45	1800	1232	1460	4492	1	6	14	21	94332	TMVP-3
48	1260	1873	1428	4561	1	7	14	22	100342	TMVP-4
49	1648	1873	1620	5141	1	7	16	24	123384	TMVP-LT
63	2268	2553	2268	7089	1	7	14	22	155958	TMVP-4
64	2268	2553	2268	7089	1	7	14	22	155958	TMVP-4
65	2784	2553	2524	7861	1	7	16	24	188664	TMVP-LT
67	2700	2913	2782	8395	1	7	14	22	184690	TMVP-3
77	3000	3964	3544	10508	1	7	16	24	252192	TMVP-3
91	3780	4849	4088	12717	1	8	16	25	317925	TMVP-4
96	3780	4914	4116	12810	1	8	16	25	320250	TMVP-4
99	5152	4979	4816	14947	1	8	18	27	403569	TMVP-4
104	4500	6042	5230	15772	1	8	16	25	394300	TMVP-3
121	5292	8526	6160	19978	1	9	18	28	559384	TMVP-4
127	5292	8591	6188	20071	1	9	18	28	561988	TMVP-4
127	5292	8591	6188	20071	1	9	18	28	561988	TMVP-4
128	5292	8591	6188	20071	1	9	18	28	561988	TMVP-4
129	6320	8591	6700	21611	1	9	20	30	648330	TMVP-LT
137	6300	9640	7616	23556	1	9	18	28	659568	TMVP-4
143	6300	9705	7644	23649	1	9	18	28	662172	TMVP-4
169	8820	15424	11004	35248	1	10	20	31	1092688	TMVP-4
188	8820	16139	11312	36271	1	10	20	31	1124401	TMVP-4
189	8820	16204	11340	36364	1	10	20	31	1127284	TMVP-4
192	8820	16204	11340	36364	1	10	20	31	1127284	TMVP-4
197	13108	16269	13496	42873	1	10	24	35	1500555	TMVP-LT
256	15876	22004	17668	55548	1	10	20	31	1721988	TMVP-4

Table A.25: TMVP Space x Delay Optimization Result (Padding Allowed, $M = A_s = A_d$, $D_m = D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	1	0	0	1	1	0	0	1	1	Initial Condition
2	4	0	2	6	1	0	1	2	12	TMVP-SB
3	9	0	6	15	1	0	2	3	45	TMVP-SB
4	16	0	12	28	1	0	2	3	84	TMVP-SB
5	25	0	20	45	1	0	3	4	180	TMVP-SB
6	36	0	30	66	1	0	3	4	264	TMVP-SB
7	49	0	42	91	1	0	3	4	364	TMVP-SB
8	64	0	56	120	1	0	3	4	480	TMVP-SB
16	192	31	184	407	1	1	4	6	2442	TMVP-STD-2
31	576	156	584	1316	1	2	5	8	10528	TMVP-STD-2
32	576	156	584	1316	1	2	5	8	10528	TMVP-STD-2
33	726	98	726	1550	1	1	6	8	12400	TMVP-STD-3
44	1089	216	1100	2405	1	2	6	9	21645	TMVP-STD-2
45	1350	134	1350	2834	1	1	6	8	22672	TMVP-STD-3
48	1152	308	1200	2660	1	2	6	9	23940	TMVP-STD-2
49	1249	308	1296	2853	1	2	7	10	28530	TMVP-LT
63	1728	595	1816	4139	1	3	6	10	41390	TMVP-STD-2
64	1728	595	1816	4139	1	3	6	10	41390	TMVP-STD-2
65	1857	595	1944	4396	1	3	7	11	48356	TMVP-LT
67	2025	615	2114	4754	1	3	7	11	52294	TMVP-STD-2
77	2535	819	2652	6006	1	3	7	11	66066	TMVP-3
91	2880	1454	3137	7471	1	4	7	12	89652	TMVP-3
96	2880	1476	3144	7500	1	4	7	12	90000	TMVP-3
99	3630	1208	3861	8699	1	3	8	12	104388	TMVP-3
104	4563	975	4706	10244	1	3	7	11	112684	TMVP-STD-2
121	4410	2239	4907	11556	1	4	8	13	150228	TMVP-3
127	5184	2040	5576	12800	1	4	7	12	153600	TMVP-STD-2
127	5184	2040	5576	12800	1	4	7	12	153600	TMVP-STD-2
128	5184	2040	5576	12800	1	4	7	12	153600	TMVP-STD-2
129	5445	2018	5801	13264	1	4	8	13	172432	TMVP-3
137	5760	2524	6312	14596	1	4	8	13	189748	TMVP-3
143	5760	2588	6336	14684	1	4	8	13	190892	TMVP-3
169	6952	3819	7728	18499	1	5	9	15	277485	TMVP-LT
188	8640	4353	9521	22514	1	5	8	14	315196	TMVP-3
189	8640	4353	9521	22514	1	5	8	14	315196	TMVP-3
192	8640	4375	9528	22543	1	5	8	14	315602	TMVP-3
197	10890	3569	11682	26141	1	4	9	14	365974	TMVP-3
256	12096	8298	13608	34002	1	6	9	16	544032	TMVP-4

Table A.26: TMVP Space x Delay Optimization Result (Padding Allowed, $M = 2A_s = 2A_d$, $D_m = D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	2	0	0	2	1	0	0	1	2	Initial Condition
2	8	0	2	10	1	0	1	2	20	TMVP-SB
3	18	0	6	24	1	0	2	3	72	TMVP-SB
4	32	0	12	44	1	0	2	3	132	TMVP-SB
5	50	0	20	70	1	0	3	4	280	TMVP-SB
6	72	0	30	102	1	0	3	4	408	TMVP-SB
7	98	0	42	140	1	0	3	4	560	TMVP-SB
8	128	0	56	184	1	0	3	4	736	TMVP-SB
16	384	31	184	599	1	1	4	6	3594	TMVP-STD-2
31	1152	156	584	1892	1	2	5	8	15136	TMVP-STD-2
32	1152	156	584	1892	1	2	5	8	15136	TMVP-STD-2
33	1452	98	726	2276	1	1	6	8	18208	TMVP-STD-3
44	2178	216	1100	3494	1	2	6	9	31446	TMVP-STD-2
45	1728	547	982	3257	1	3	6	10	32570	TMVP-STD-2
48	1728	551	984	3263	1	3	6	10	32630	TMVP-STD-2
49	1922	551	1080	3553	1	3	7	11	39083	TMVP-LT
63	2940	764	1617	5321	1	3	7	11	58531	TMVP-3
64	3456	595	1816	5867	1	3	6	10	58670	TMVP-STD-2
65	3630	691	1914	6235	1	3	7	11	68585	TMVP-3
67	2880	1258	1721	5859	1	4	7	12	70308	TMVP-3
77	4050	1256	2268	7574	1	4	7	12	90888	TMVP-STD-2
91	5184	1824	3038	10046	1	4	7	12	120552	TMVP-STD-2
96	5184	1844	3048	10076	1	4	7	12	120912	TMVP-STD-2
99	7260	1208	3861	12329	1	3	8	12	147948	TMVP-3
104	5760	2347	3485	11592	1	4	8	13	150696	TMVP-3
121	8820	2239	4907	15966	1	4	8	13	207558	TMVP-3
127	10368	2040	5576	17984	1	4	7	12	215808	TMVP-STD-2
127	10368	2040	5576	17984	1	4	7	12	215808	TMVP-STD-2
128	10368	2040	5576	17984	1	4	7	12	215808	TMVP-STD-2
129	10890	2018	5801	18709	1	4	8	13	243217	TMVP-3
137	8640	3739	5232	17611	1	5	8	14	246554	TMVP-3
143	8640	3803	5256	17699	1	5	8	14	247786	TMVP-3
169	11250	5696	6814	23760	1	6	9	16	380160	TMVP-3
188	12096	6885	7546	26527	1	6	9	16	424432	TMVP-4
189	12096	6950	7560	26606	1	6	9	16	425696	TMVP-4
192	12096	6950	7560	26606	1	6	9	16	425696	TMVP-4
197	18150	4899	10032	33081	1	5	9	15	496215	TMVP-3
256	24192	8298	13608	46098	1	6	9	16	737568	TMVP-4

Table A.27: TMVP Space x Delay Optimization Result (Padding Allowed, $M = 4A_s = 4A_d$, $D_m = D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	4	0	0	4	1	0	0	1	4	Initial Condition
2	16	0	2	18	1	0	1	2	36	TMVP-SB
3	36	0	6	42	1	0	2	3	126	TMVP-SB
4	64	0	12	76	1	0	2	3	228	TMVP-SB
5	100	0	20	120	1	0	3	4	480	TMVP-SB
6	96	17	24	137	1	1	3	5	685	TMVP-STD-3
7	196	0	42	238	1	0	3	4	952	TMVP-SB
8	256	0	56	312	1	0	3	4	1248	TMVP-SB
16	768	31	184	983	1	1	4	6	5898	TMVP-STD-2
31	2304	156	584	3044	1	2	5	8	24352	TMVP-STD-2
32	2304	156	584	3044	1	2	5	8	24352	TMVP-STD-2
33	1728	532	574	2834	1	3	6	10	28340	TMVP-STD-2
44	3456	531	974	4961	1	3	6	10	49610	TMVP-STD-2
45	3456	547	982	4985	1	3	6	10	49850	TMVP-STD-2
48	3456	551	984	4991	1	3	6	10	49910	TMVP-STD-2
49	3844	551	1080	5475	1	3	7	11	60225	TMVP-LT
63	5880	764	1617	8261	1	3	7	11	90871	TMVP-3
64	6912	595	1816	9323	1	3	6	10	93230	TMVP-STD-2
65	5760	1216	1704	8680	1	4	7	12	104160	TMVP-3
67	5184	1731	1790	8705	1	4	7	12	104460	TMVP-STD-2
77	6480	1744	2072	10296	1	4	8	13	133848	TMVP-3
91	10368	1824	3038	15230	1	4	7	12	182760	TMVP-STD-2
96	10368	1844	3048	15260	1	4	7	12	183120	TMVP-STD-2
99	8640	3378	3101	15119	1	5	8	14	211666	TMVP-3
104	8640	3442	3125	15207	1	5	8	14	212898	TMVP-3
121	14700	3069	4277	22046	1	5	8	14	308644	TMVP-3
127	16128	3145	4536	23809	1	5	8	14	333326	TMVP-4
127	16128	3145	4536	23809	1	5	8	14	333326	TMVP-4
128	16128	3145	4536	23809	1	5	8	14	333326	TMVP-4
129	12096	5842	4480	22418	1	6	9	16	358688	TMVP-4
137	12096	6000	4522	22618	1	6	9	16	361888	TMVP-4
143	12096	6065	4536	22697	1	6	9	16	363152	TMVP-4
169	22500	5696	6814	35010	1	6	9	16	560160	TMVP-3
188	24192	6885	7546	38623	1	6	9	16	617968	TMVP-4
189	24192	6950	7560	38702	1	6	9	16	619232	TMVP-4
192	24192	6950	7560	38702	1	6	9	16	619232	TMVP-4
197	20160	11210	7833	39203	1	7	10	18	705654	TMVP-4
256	37500	12709	12117	62326	1	7	10	18	1121868	TMVP-3

Table A.28: TMVP Space x Delay Optimization Result (Padding Allowed, $M = 4A_s = 2A_d$, $D_m = D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	4	0	0	4	1	0	0	1	4	Initial Condition
2	16	0	4	20	1	0	1	2	40	TMVP-SB
3	36	0	12	48	1	0	2	3	144	TMVP-SB
4	64	0	24	88	1	0	2	3	264	TMVP-SB
5	100	0	40	140	1	0	3	4	560	TMVP-SB
6	96	17	48	161	1	1	3	5	805	TMVP-STD-3
7	196	0	84	280	1	0	3	4	1120	TMVP-SB
8	256	0	112	368	1	0	3	4	1472	TMVP-SB
16	768	31	368	1167	1	1	4	6	7002	TMVP-STD-2
31	2304	156	1168	3628	1	2	5	8	29024	TMVP-STD-2
32	2304	156	1168	3628	1	2	5	8	29024	TMVP-STD-2
33	1728	532	1148	3408	1	3	6	10	34080	TMVP-STD-2
44	3456	531	1948	5935	1	3	6	10	59350	TMVP-STD-2
45	3456	547	1964	5967	1	3	6	10	59670	TMVP-STD-2
48	3456	551	1968	5975	1	3	6	10	59750	TMVP-STD-2
49	3844	551	2160	6555	1	3	7	11	72105	TMVP-LT
63	5880	764	3234	9878	1	3	7	11	108658	TMVP-3
64	6912	595	3632	11139	1	3	6	10	111390	TMVP-STD-2
65	5760	1216	3408	10384	1	4	7	12	124608	TMVP-3
67	5760	1258	3442	10460	1	4	7	12	125520	TMVP-3
77	6480	1744	4144	12368	1	4	8	13	160784	TMVP-3
91	8064	2532	5012	15608	1	5	8	14	218512	TMVP-4
96	10368	1844	6096	18308	1	4	7	12	219696	TMVP-STD-2
99	8640	3378	6202	18220	1	5	8	14	255080	TMVP-3
104	8640	3442	6250	18332	1	5	8	14	256648	TMVP-3
121	14700	3069	8554	26323	1	5	8	14	368522	TMVP-3
127	16128	3145	9072	28345	1	5	8	14	396830	TMVP-4
127	16128	3145	9072	28345	1	5	8	14	396830	TMVP-4
128	16128	3145	9072	28345	1	5	8	14	396830	TMVP-4
129	12096	5842	8960	26898	1	6	9	16	430368	TMVP-4
137	12096	6000	9044	27140	1	6	9	16	434240	TMVP-4
143	12096	6065	9072	27233	1	6	9	16	435728	TMVP-4
169	22500	5696	13628	41824	1	6	9	16	669184	TMVP-3
188	24192	6885	15092	46169	1	6	9	16	738704	TMVP-4
189	24192	6950	15120	46262	1	6	9	16	740192	TMVP-4
192	24192	6950	15120	46262	1	6	9	16	740192	TMVP-4
197	20160	11210	15666	47036	1	7	10	18	846648	TMVP-4
256	37500	12709	24234	74443	1	7	10	18	1339974	TMVP-3

Table A.29: TMVP Space x Delay Optimization Result (Padding Allowed, $M = A_s = A_d$, $2D_m = 2D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	1	0	0	1	1	0	0	1	1	Initial Condition
2	4	0	2	6	1	0	2	3	18	TMVP-SB
3	9	0	6	15	1	0	4	5	75	TMVP-SB
4	16	0	12	28	1	0	4	5	140	TMVP-SB
5	25	0	20	45	1	0	6	7	315	TMVP-SB
6	36	0	30	66	1	0	6	7	462	TMVP-SB
7	49	0	42	91	1	0	6	7	637	TMVP-SB
8	64	0	56	120	1	0	6	7	840	TMVP-SB
16	192	31	184	407	1	1	8	10	4070	TMVP-STD-2
31	576	156	584	1316	1	2	10	13	17108	TMVP-STD-2
32	576	156	584	1316	1	2	10	13	17108	TMVP-STD-2
33	641	156	648	1445	1	2	12	15	21675	TMVP-LT
44	1089	216	1100	2405	1	2	12	15	36075	TMVP-STD-2
45	864	547	982	2393	1	3	12	16	38288	TMVP-STD-2
48	864	551	984	2399	1	3	12	16	38384	TMVP-STD-2
49	961	551	1080	2592	1	3	14	18	46656	TMVP-LT
63	1728	595	1816	4139	1	3	12	16	66224	TMVP-STD-2
64	1728	595	1816	4139	1	3	12	16	66224	TMVP-STD-2
65	1857	595	1944	4396	1	3	14	18	79128	TMVP-LT
67	1440	1258	1721	4419	1	4	14	19	83961	TMVP-3
77	2025	1256	2268	5549	1	4	14	19	105431	TMVP-STD-2
91	2592	1824	3038	7454	1	4	14	19	141626	TMVP-STD-2
96	2592	1844	3048	7484	1	4	14	19	142196	TMVP-STD-2
99	3205	1498	3471	8174	1	4	16	21	171654	TMVP-3
104	2880	2347	3485	8712	1	4	16	21	182952	TMVP-3
121	5184	2016	5564	12764	1	4	14	19	242516	TMVP-STD-2
127	5184	2040	5576	12800	1	4	14	19	243200	TMVP-STD-2
127	5184	2040	5576	12800	1	4	14	19	243200	TMVP-STD-2
128	5184	2040	5576	12800	1	4	14	19	243200	TMVP-STD-2
129	5445	2018	5801	13264	1	4	16	21	278544	TMVP-3
137	4320	3739	5232	13291	1	5	16	22	292402	TMVP-3
143	4320	3803	5256	13379	1	5	16	22	294338	TMVP-3
169	6952	3819	7728	18499	1	5	18	24	443976	TMVP-LT
188	8640	4353	9521	22514	1	5	16	22	495308	TMVP-3
189	8640	4353	9521	22514	1	5	16	22	495308	TMVP-3
192	8640	4375	9528	22543	1	5	16	22	495946	TMVP-3
197	9075	4899	10032	24006	1	5	18	24	576144	TMVP-3
256	12096	8298	13608	34002	1	6	18	25	850050	TMVP-4

Table A.30: TMVP Space x Delay Optimization Result (Padding Allowed, $M = 2A_s = 2A_d$, $2D_m = 2D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	2	0	0	2	1	0	0	1	2	Initial Condition
2	8	0	2	10	1	0	2	3	30	TMVP-SB
3	18	0	6	24	1	0	4	5	120	TMVP-SB
4	32	0	12	44	1	0	4	5	220	TMVP-SB
5	50	0	20	70	1	0	6	7	490	TMVP-SB
6	54	11	24	89	1	1	6	8	712	TMVP-STD-2
7	98	0	42	140	1	0	6	7	980	TMVP-SB
8	96	15	44	155	1	1	6	8	1240	TMVP-STD-2
16	288	76	148	512	1	2	8	11	5632	TMVP-STD-2
31	864	291	476	1631	1	3	10	14	22834	TMVP-STD-2
32	864	291	476	1631	1	3	10	14	22834	TMVP-STD-2
33	994	291	540	1825	1	3	12	16	29200	TMVP-LT
44	1440	702	845	2987	1	4	12	17	50779	TMVP-3
45	1440	702	845	2987	1	4	12	17	50779	TMVP-3
48	1440	724	852	3016	1	4	12	17	51272	TMVP-3
49	1634	724	948	3306	1	4	14	19	62814	TMVP-LT
63	2592	1000	1492	5084	1	4	12	17	86428	TMVP-STD-2
64	2592	1000	1492	5084	1	4	12	17	86428	TMVP-STD-2
65	2850	1000	1620	5470	1	4	14	19	103930	TMVP-LT
67	2880	1258	1721	5859	1	4	14	19	111321	TMVP-3
77	4050	1256	2268	7574	1	4	14	19	143906	TMVP-STD-2
91	4320	2129	2597	9046	1	5	14	20	180920	TMVP-3
96	4320	2151	2604	9075	1	5	14	20	181500	TMVP-3
99	4970	2173	2931	10074	1	5	16	22	221628	TMVP-3
104	4800	2917	3065	10782	1	5	16	22	237204	TMVP-3
121	7776	3231	4592	15599	1	5	14	20	311980	TMVP-STD-2
127	7776	3255	4604	15635	1	5	14	20	312700	TMVP-STD-2
127	7776	3255	4604	15635	1	5	14	20	312700	TMVP-STD-2
128	7776	3255	4604	15635	1	5	14	20	312700	TMVP-STD-2
129	8290	3255	4860	16405	1	5	16	22	360910	TMVP-LT
137	7200	4624	4582	16406	1	6	16	23	377338	TMVP-3
143	7200	4668	4596	16464	1	6	16	23	378672	TMVP-3
169	11250	5696	6814	23760	1	6	18	25	594000	TMVP-3
188	12960	6378	7901	27239	1	6	16	23	626497	TMVP-3
189	12960	6378	7901	27239	1	6	16	23	626497	TMVP-3
192	12960	6400	7908	27268	1	6	16	23	627164	TMVP-3
197	14910	6464	8892	30266	1	6	18	25	756650	TMVP-3
256	18144	11133	11340	40617	1	7	18	26	1056042	TMVP-4

Table A.31: TMVP Space x Delay Optimization Result (Padding Allowed, $M = 4A_s = 4A_d$, $2D_m = 2D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	4	0	0	4	1	0	0	1	4	Initial Condition
2	16	0	2	18	1	0	2	3	54	TMVP-SB
3	36	0	6	42	1	0	4	5	210	TMVP-SB
4	64	0	12	76	1	0	4	5	380	TMVP-SB
5	100	0	20	120	1	0	6	7	840	TMVP-SB
6	96	17	24	137	1	1	6	8	1096	TMVP-STD-3
7	196	0	42	238	1	0	6	7	1666	TMVP-SB
8	192	15	44	251	1	1	6	8	2008	TMVP-STD-2
16	576	76	148	800	1	2	8	11	8800	TMVP-STD-2
31	1728	291	476	2495	1	3	10	14	34930	TMVP-STD-2
32	1728	291	476	2495	1	3	10	14	34930	TMVP-STD-2
33	1440	604	497	2541	1	4	12	17	43197	TMVP-3
44	2592	882	866	4340	1	4	12	17	73780	TMVP-STD-2
45	2592	898	874	4364	1	4	12	17	74188	TMVP-STD-2
48	2592	902	876	4370	1	4	12	17	74290	TMVP-STD-2
49	2980	902	972	4854	1	4	14	19	92226	TMVP-LT
63	5184	1000	1492	7676	1	4	12	17	130492	TMVP-STD-2
64	5184	1000	1492	7676	1	4	12	17	130492	TMVP-STD-2
65	4320	1801	1524	7645	1	5	14	20	152900	TMVP-3
67	4320	1843	1541	7704	1	5	14	20	154080	TMVP-3
77	5400	2164	1847	9411	1	5	16	22	207042	TMVP-3
91	8640	2129	2597	13366	1	5	14	20	267320	TMVP-3
96	8640	2151	2604	13395	1	5	14	20	267900	TMVP-3
99	7200	3738	2716	13654	1	6	16	23	314042	TMVP-3
104	7200	3892	2765	13857	1	6	16	23	318711	TMVP-3
121	12096	4025	3766	19887	1	6	16	23	457401	TMVP-4
127	12096	4090	3780	19966	1	6	16	23	459218	TMVP-4
127	12096	4090	3780	19966	1	6	16	23	459218	TMVP-4
128	12096	4090	3780	19966	1	6	16	23	459218	TMVP-4
129	12960	5348	4631	22939	1	6	16	23	527597	TMVP-3
137	12960	5494	4692	23146	1	6	16	23	532358	TMVP-3
143	12960	5558	4716	23234	1	6	16	23	534382	TMVP-3
169	22500	5696	6814	35010	1	6	18	25	875250	TMVP-3
188	18144	9342	6790	34276	1	7	18	26	891176	TMVP-4
189	18144	9407	6804	34355	1	7	18	26	893230	TMVP-4
192	18144	9407	6804	34355	1	7	18	26	893230	TMVP-4
197	21600	10449	8082	40131	1	7	18	26	1043406	TMVP-3
256	36288	11133	11340	58761	1	7	18	26	1527786	TMVP-4

Table A.32: TMVP Space x Delay Optimization Result (Padding Allowed, $M = 4A_s = 2A_d$, $2D_m = 2D_s = D_d$)

n	M	A_s	A_d	**Space**	D_m	D_s	D_d	**Delay**	**Space x Delay**	**Algorithm**
1	4	0	0	4	1	0	0	1	4	Initial Condition
2	16	0	4	20	1	0	2	3	60	TMVP-SB
3	36	0	12	48	1	0	4	5	240	TMVP-SB
4	64	0	24	88	1	0	4	5	440	TMVP-SB
5	100	0	40	140	1	0	6	7	980	TMVP-SB
6	96	17	48	161	1	1	6	8	1288	TMVP-STD-3
7	196	0	84	280	1	0	6	7	1960	TMVP-SB
8	192	15	88	295	1	1	6	8	2360	TMVP-STD-2
16	576	76	296	948	1	2	8	11	10428	TMVP-STD-2
31	1728	291	952	2971	1	3	10	14	41594	TMVP-STD-2
32	1728	291	952	2971	1	3	10	14	41594	TMVP-STD-2
33	1440	604	994	3038	1	4	12	17	51646	TMVP-3
44	2592	882	1732	5206	1	4	12	17	88502	TMVP-STD-2
45	2592	898	1748	5238	1	4	12	17	89046	TMVP-STD-2
48	2592	902	1752	5246	1	4	12	17	89182	TMVP-STD-2
49	2980	902	1944	5826	1	4	14	19	110694	TMVP-LT
63	5184	1000	2984	9168	1	4	12	17	155856	TMVP-STD-2
64	5184	1000	2984	9168	1	4	12	17	155856	TMVP-STD-2
65	4320	1801	3048	9169	1	5	14	20	183380	TMVP-3
67	4320	1843	3082	9245	1	5	14	20	184900	TMVP-3
77	5400	2164	3694	11258	1	5	16	22	247676	TMVP-3
91	8640	2129	5194	15963	1	5	14	20	319260	TMVP-3
96	8640	2151	5208	15999	1	5	14	20	319980	TMVP-3
99	7200	3738	5432	16370	1	6	16	23	376510	TMVP-3
104	7200	3892	5530	16622	1	6	16	23	382306	TMVP-3
121	12096	4025	7532	23653	1	6	16	23	544019	TMVP-4
127	12096	4090	7560	23746	1	6	16	23	546158	TMVP-4
127	12096	4090	7560	23746	1	6	16	23	546158	TMVP-4
128	12096	4090	7560	23746	1	6	16	23	546158	TMVP-4
129	10080	6346	7882	24308	1	7	18	26	632008	TMVP-4
137	12960	5494	9384	27838	1	6	16	23	640274	TMVP-3
143	12960	5558	9432	27950	1	6	16	23	642850	TMVP-3
169	22500	5696	13628	41824	1	6	18	25	1045600	TMVP-3
188	18144	9342	13580	41066	1	7	18	26	1067716	TMVP-4
189	18144	9407	13608	41159	1	7	18	26	1070134	TMVP-4
192	18144	9407	13608	41159	1	7	18	26	1070134	TMVP-4
197	21600	10449	16164	48213	1	7	18	26	1253538	TMVP-3
256	36288	11133	22680	70101	1	7	18	26	1822626	TMVP-4