2011

Signal Processing Algorithms and Their Applications

Xuebin Wu
Lehigh University

EFFICIENT SIGNAL PROCESSING ALGORITHMS FOR DIGITAL COMMUNICATION SYSTEMS

by

Xuebin Wu

Presented to the Graduate and Research Committee

of Lehigh University

in Candidacy for the Degree of

Doctor of Philosophy

in

Electrical Engineering

Lehigh University

September 2011

Approved and recommended for acceptance as a dissertation in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Date

Prof. Zhiyuan Yan
(Dissertation Advisor)

Accepted Date

Committee Members:

Prof. Zhiyuan Yan
(Committee Chair)

Prof. Meghanad D. Wagh

Prof. Shalinee Kishore

Dr. Ying Wang
Qualcom Inc.

Acknowledgments

Pursuing Ph.D degree could be a tedious and lonely process, and I could hardly finish it without the personal, intellectual, and financial support from numerous people. Therefore, my heartily acknowledgment goes to my advisor and committee members, all of my friends and colleagues, and last but not least, my parents.

First, I would like to express my most sincere gratitude to my esteemed advisor, Prof. Zhiyuan Yan, for offering me this Ph.D student position, for his expert guidance and valuable advices, and for his continuous support throughout my research work. It is my honor to pursue my Ph.D degree under his supervision and guidance, and the research directions he pointed to me are fruitful, leading to many important results and forming the essence of this dissertation. Prof. Yan also recommended me for an internship position, which is an invaluable experience to me and has profoundly changed my life. I am very fortunate to have him as my advisor, my mentor, and my friend.

I would like to to thank Prof. Meghanad D. Wagh, Prof. Shalinee Kishore, and Dr. Ying Wang for serving in my committee and spending their precious time on examining my work. Prof. Meghanad D. Wagh and Prof. Shalinee Kishore both offer interesting and useful courses, and I am grateful that I have the honor to be one of their students. Moreover, I also have the honor to collaborate with Prof. Wagh and Dr. Wang. I am deeply grateful that Prof. Wagh shared his efficient cyclic convolution algorithm with me, and Dr. Wang helped me improve my manuscripts from all aspects. I will never forget their efforts.

I am deeply indebted to my Master's advisor, Prof. Yonglin Xue at Digital TV R&D Center in Tsinghua University, who gave me many opportunities to learn and grow and supported me while I was applying to study abroad. I would also like to thank my friends at Tsinghua University, Haigang Zhang, Liuhe Lu, Xin Zhou, Kang Zhao, Jin Huang, Yue Wang, Yan Zhang, and Chen Chen. In particular, I would like to thank Yue Wang for accompanying me spiritually in the past years. She made my life full of sunshine.

My thanks also go to my friends and colleagues in my lab, Yongmei Dai, Ning Chen, Maximilien Gadouleau, Feng Shi, Hongmei Xie, and Chenrong Xiong. I learned a lot from many interesting discussions with them. In particular, I would like to thank Yongmei and Ning for their recommendations and help while I was seeking career opportunities. My thanks are also extended to my friends at Lehigh University, Xiaochen Zhang, Dan Li, Xiaoxiao Ma, Hanzhu Ni, Xingjian Zhang, and Pu Zhao. I am really grateful for their help over the past years. They make my life easier and more colorful.

Finally, my deepest thanks go to my dear parents for their unconditional love and endless support. It is they who taught me how to read and write when I was young. It is they who supported me and brought me up. It is they who encouraged me to pursue my Ph.D degree abroad. I cannot imagine how happy they would be after knowing that I have earned my Ph.D degree.

Contents

List of Tables

List of Figures

Abstract

Efficient signal processing algorithms are essential in various communication systems, such as on-chip interconnect buses, wireless communication systems, magnetic recording channels, and cryptosystems. These efficient algorithms either have low computational complexities or have modular structures that are favorable in hardware implementations. In this dissertation, we investigate several efficient signal processing algorithms in different areas. These areas include crosstalk avoidance codes, multiple-input and multiple-output (MIMO) communication systems, and discrete Fourier transforms (DFTs) over finite fields.

Crosstalk avoidance codes (CACs) are of great interest since they are promising in combating the crosstalk delay problem of interconnect on-chip buses in deep sub-micron technology. As feature shrinks, the gate delay is reduced while the interconnect delay increases due to increasingly more severe crosstalk between adjacent wires. CACs are a promising technology that can reduce the crosstalk delay effectively. However, CACs require extra logic circuits as encoders and decoders (CODECs), but CODECs of the CACs are so complex, prohibiting the usage of CACs in practice. The areas and the delays of our CAC CODECs based on numeral systems all increase quadratically with the width of bus. Furthermore, to increase the code rates of CACs, we propose two dimensional CACs (TDCACs). We investigate the properties of TDCACs with and without memory, respectively.

We also investigate efficient algorithms in MIMO detection. We apply several novel ordering schemes to K-Best detectors, and show that they can improve the reliability of K-Best detecting results. The hardware implementations of our ordering schemes show that they only incur a small overhead in comparison with traditional ordering schemes, and they can achieve a high throughput with a pipelined architecture. Furthermore, we also propose a list MIMO detection algorithm using the memory-constrained tree search strategy. This algorithm offers a flexible balance of computational complexity and memory requirement, which can be tuned by the number of available memory units.

DFTs over finite fields find wide applications in various communication systems, and cyclotomic fast Fourier transforms (CFFTs) can reduce their multiplicative complexities greatly. As DFTs over non-characteristic-2 fields are taken into consideration in modern communication systems, we generalize CFFTs to arbitrary finite fields by devising efficient algorithms for cyclic convolutions over arbitrary finite fields. We also analyze the computational complexities of CFFTs in theory, and our results confirm the advantages of CFFTs. To further reduce the computational complexity of DFTs, we propose composite cyclotomic Fourier transforms, which integrate CFFTs and the idea of the prime-factor algorithm as well as the Cooley-Tukey algorithm.

Chapter 1

Introduction

Efficient signal processing algorithms are of great importance in various modern communication systems, such as on-chip global buses, wireless communication systems, magnetic and optical recording systems, optical transmission systems, cryptosystems, etc. These algorithms are efficient because they either have low computational complexities or have modular structures that are favorable in hardware implementations. In this dissertation, we investigate and propose several efficient signal processing algorithms in different areas. In particular, these areas include the crosstalk avoidance codes (CACs), multiple-input and multiple-output (MIMO) communication systems, and discrete Fourier transforms (DFTs) over finite fields.

In this chapter, we first explain our motivations of our research in Sec. 1.1, and then present our main contributions in this dissertation as well as the organization of this dissertation in Sec. 1.2.

1.1 Motivations

1.1.1 Crosstalk Avoidance Codes

The technology of integrated circuits has dramatically changed our lives. As the feature shrinks, more gates can be placed on a single chip, and the speed of the chip increases due to the reduced gate delay. However, the speed of the interconnect bus slows down due to the shrinking process. Because of the coupling capacitance between the adjacent wires, the transitions on one wire may affect other wires of the same bus. Due to the shrinking process, the ratio between the coupling capacitance and the loading capacitance between a single wire and ground is increasing, and the crosstalk will significantly delay the transmission of the signals on the bus. Therefore, the crosstalk delay has become a bottleneck in deep sub-micron system-on-chip designs [1].

Crosstalk avoidance codes [2–4] are promising compared with other technologies, such as shielding, pre-charging, and using repeaters, because they are technology independent, require less bus area and power, and can make trade-offs between area consumption and worst-case delay. However, they require extra logic for the encoders and decoders (CODECs), and the complexity of the LUT-based CODEC increases exponentially with the bus width, which prevents the usage of CACs in practice. Inspired by the idea in [5] and [6], we propose a numeral system based CODEC for general CACs, and design efficient CODECs for several families of CACs.

The CACs proposed previously are all sets of one dimensional column vectors in the spatial domain. Although they require less bus area than shielding, we can further reduce the area consumptions by two dimensional CACs (TDCACs), which transmit a group of vectors where the crosstalk has been reduced between adjacent vectors. The construction and properties of such codes are discussed in this dissertation.

1.1.2 List MIMO detection and Ordering Schemes

Multiple-input and multiple-output (MIMO) communication systems have been attractive in recent years because they can provide high data rate transmission and improve link reliability. Though the detection of MIMO systems, if implemented in a brute-force way, has an exponential complexity with the constellation size and the transmit antennas number, the complexity will be significantly reduced if a tree search algorithm is used. Different tree search strategies have different computational complexities and memory requirements. For example, the sphere decoding (SD) algorithm [7–11] which employs the depth-first strategy has a fixed memory requirement, but it has a very high computational complexity in the worst case, which will affect the throughput of the MIMO detector. In contrast, the stack algorithm which employs the best-first strategy has been shown to have the smallest computational complexity [12], but it requires much more memory in the worst case, which increases the cost of the MIMO detector. The memory-constrained tree search (MCTS) strategy, proposed in [13], takes into account the available memory sizes and dynamically alters between the best-first and depth-first strategies. It requires a fixed size of memory, and its computational complexity decreases when the memory size increases. It has a smaller computational complexity than the SD algorithm even with a smaller size of memory. When the computational complexity of the MCTS detector approaches that of the stack algorithm, it only requires a fraction of the memory required by the stack algorithm.

One of the issues regarding the MIMO detection is the soft information generation. For coded MIMO systems, properly designed soft-decision MIMO detection and soft-input and soft-output channel decoder can achieve near-capacity performance on a multiple-antenna channel [14]. Therefore, soft information generation is of great significance. The soft information is generated based on a candidate list created by the list sphere decoding (LSD) detector [14], list sequential (LISS) detector [15], or other list creation algorithms. The LSD and LISS detectors are extensions of the SD algorithm and the stack algorithm, respectively. Similar to the relation between the SD algorithm and the stack algorithm, the LSD algorithm has a fixed and small memory requirement, but it has a high computational complexity in the worst case, while the LISS algorithm has the smallest computational complexity but a large memory requirement. Therefore, we need to find a trade-off between the memory requirement and the computational complexity.

One of the key steps of the tree search based MIMO detection algorithm is decomposing the channel matrix $\mathbf{H} = \mathbf{QR}$, where $\mathbf{Q}$ is a unitary matrix and $\mathbf{R}$ is an upper triangular matrix. The column order of $\mathbf{H}$ can affect the computational complexity of the tree search algorithm. In [13], several novel ordering schemes are proposed, which are shown to reduce the computational complexity of the SD, stack, and MCTS algorithms. Obviously, the ordering schemes cannot affect the detection error rates of the maximum likelihood (ML) detectors, but for the other types of tree search based MIMO detectors, e.g., the K-Best detector, it is possible that different ordering schemes have different impact on the detector error rate.

In this dissertation, the MCTS tree search strategy is extended to the list MIMO detection algorithms, and a list MCTS (LMCTS) algorithm is proposed. The advantages of the MCTS algorithm still hold for the LMCTS algorithm. It requires a fixed size of memory, and the computational complexity is tunable through memory size. We also apply our novel ordering schemes to the K-Best MIMO detector, which employs breadth-first strategy and has a fixed computational complexity. The simulation results show that some of our ordering schemes can improve the reliability of the K-Best detecting results.

1.1.3 Discrete Fourier Transforms over Finite Fields

Discrete Fourier transforms (DFTs) over finite fields are of great significance in communication and storage systems. For example, Reed-Solomon (RS) codes are an important family of error control codes in current communication and storage systems, and the syndrome-based decoders of RS codes can be implemented efficiently with DFTs over finite fields. Very long RS codes over large fields, e.g., RS codes over GF(2^{12}) with up to thousands of symbols, are being considered in the systems requiring a very low error rate, such as magnetic storage systems [16] and high speed optical transmission systems [17]. However, the complexities of the RS decoders prohibit their usages in practice. Therefore reducing the complexities of DFTs over finite fields is always very important, especially in the RS code decoder implementations. Furthermore, DFTs over non-characteristic-2 finite fields, e.g., GF(3^m), are also considered in modern error control codes [18] and cryptosystems [19], therefore low complexity DFT implementations over arbitrary finite fields are needed in practice.

The recently proposed cyclotomic fast Fourier transforms (CFFTs) [20, 21] are promising due to their very low multiplicative complexities. However, they are proposed for finite fields GF(2^m). The key challenge in generalizing CFFTs to arbitrary finite fields is the efficient algorithm for cyclic convolutions over arbitrary finite fields, which is the key to the multiplicative complexity reduction for CFFTs. Furthermore, CFFTs are attractive due to their low overall computational complexities, which is demonstrated for DFTs with short and moderate lengths (see, e.g., [22]), but their computational complexities in theory have never been studied to the best of our knowledge.

The CFFTs have a disadvantage of high additive complexities. Their additive complexities increase much faster than their multiplicative complexities, and will dominate the computational complexities of long-DFTs. Though we can use some preprocessing techniques, such as the common subexpression elimination (CSE), to reduce the additive complexities, the complexity of the preporcessing makes it impossible to derive efficient algorithm for very long DFTs, such as 2047-point DFT over GF(2^{11}) and 4095-point DFT over GF(2^{12}). Therefore, we need to improve the CFFTs by making trade-offs between the additive and multiplicative complexities.

In this dissertation, all the aforementioned problems regarding the CFFTs are addressed to some extent. We devise an efficient algorithm for arbitrary length cyclic convolutions over arbitrary finite fields so as to generalize the CFFTs. The computational complexities of CFFTs are theoretically analyzed, and the bounds on their additive and multiplicative complexities are given. To further reduce the computational complexities for long DFTs, such as 2047-point DFTs over GF(2^{11}) and 4095-point DFTs over GF(2^{12}), we propose a composite cyclotomic Fourier transform (CCFT) that combines the idea of prime-factor algorithm [23] and Cooley-Tukey algorithm [24] as well as the idea of CFFT. When the length of DFT is prime, the CCFT reduces to CFFT, and they have the same complexity. When the length of DFT is composite, the CCFT decomposes the DFT into short sub-DFTs, and the DFT result is constructed from the sub-DFT results. Each sub-DFT is implemented by the CFFT to reduce the multiplicative complexity. Since the lengths of the sub-DFTs are usually much shorter than the original DFT length, it is much easier to use preprocessing technologies to reduce their additive complexities.

1.2 Contributions and Organization

This dissertation has the following contributions and is organized as follows.

- In Chapter 2, we first propose a generic CODEC based on numeral systems for CACs, and then based on this generic CODEC, we find the numeral systems for each family of CACs. Finally, we are able to encode the one lambda code (OLC), forbidden pattern code (FPC), forbidden transition code (FTC), and a subset of forbidden overlapping code (FOC). We also implement our CODECs. The hardware implementation results show that our CODECs have quadratic complexities with respect to the bus width. The delay and power consumption of our CODECs also grow quadratically with the bus width.

- In Chapter 3, we propose the idea of two dimensional CAC (TDCAC). The TDCAC has a higher code rate than the previously proposed one dimensional codes, such as FTC and FPC. The construction of the TDCACs with memory and memoryless TDCACs is discussed. The maximum code rates for certain

TDCACs are computed in this dissertation, and we show by an example that the TDCACs can be efficiently implemented.

- In Chapter 4, we apply the novel ordering schemes in [13] to the K-Best detectors. The simulation results show that some of them can improve the reliability of the K-Best detectors. Therefore, for the same detection error rate requirement, we can use a smaller K in the K-Best detector by using our ordering schemes, which will reduce both the hardware and computational complexities. We also implement two of our ordering schemes in hardware. Compared with previous designs, they only incur a small overhead. High throughput can be achieved by pipelining our implementation, the architecture of which is shown in this dissertation.

- In Chapter 5, we integrate the list MIMO detection algorithms with the MCTS strategy, resulting in the LMCTS algorithm. Our simulation results show that the LMCTS algorithm has a tunable computational complexity through the memory size. It has a smaller computational complexity than the SD algorithm, and requires only a fraction of the memory required by the stack algorithm when they have almost the same complexity. We also implement the LMCTS algorithm in hardware. Though it has a poor throughput compared with the list detection implementations in the literature, it can be greatly improved by carefully parallelizing the circuit and making trade-off between the computational complexity and error rate performance.

- In Chapter 6, we generalize the CFFTs to arbitrary finite fields by first proposing an algorithm for arbitrary size Toeplitz matrix vector product (TMVP) and then proposing an efficient algorithm for p-point cyclic convolutions over arbitrary finite fields. We first reduce the p-point cyclic convolution to an $(p-1) \times (p-1)$ TMVP, and then use our TMVP algorithm to construct the cyclic convolution algorithm. With the multi-dimensional technologies in [25], we devise efficient algorithms for arbitrary length cyclic convolutions and then the CFFTs is generalized to arbitrary finite fields. Furthermore, we also analyze the computational complexity of the CFFTs in theory. Our results show that the multiplicative complexity of an n-point CFFT is on the order of $O(n(\log_p n)^{\log_2 \frac{3}{2}})$, and the additive complexity on the order of $O(n^2/(\log_p n)^{\log_2 \frac{8}{3}})$. The CFFTs have the smallest multiplicative complexities, but their asymptotically sub-optimal additive complexities render them asymptotically inefficient for long-DFTs.

- In Chapter 7, we propose the composite cyclotomic Fourier transform (CCFT), which combines the ideas of the prime-factor algorithm and the Cooley-Tukey algorithm as well as the CFFT. Compared with the direct implementation of CCFT, this algorithm significantly reduces the overall computational complexities for long DFTs with composite lengths. We derive the complexities of all the possible DFTs over GF(2^l) with $4 \leq l \leq 12$. It is the first efficient algorithm for 4095-point DFT over GF(2^{12}) to the best of our knowledge. Furthermore, the regular structure of this algorithm is suitable for hardware implementations. It is easy to reuse modules to save chip area or parallelize the circuit to achieve a high throughput.

Chapter 2

Efficient CAC CODEC Desgins Based on Numeral Systems

2.1 Introduction

Deep sub-micron system-on-chip designs suffer from the delay of global buses, which **increases** while the gate delay **decreases** with the shrinking feature size. The delay of the i-th wire of an m-bit bus is given by [26]

$$T_i = \begin{cases} \tau_0[(1+\lambda)\Delta_1^2 - \lambda\Delta_1\Delta_2], & i = 1, \\ \tau_0[(1+2\lambda)\Delta_i^2 - \lambda\Delta_i(\Delta_{i-1}+\Delta_{i+1})], & i \neq 1, m, \\ \tau_0[(1+\lambda)\Delta_m^2 - \lambda\Delta_m\Delta_{m-1}], & i = m, \end{cases} \tag{2.1}$$

where λ is the ratio of the coupling capacitance between adjacent wires and the loading capacitance between the i-th wire and the ground, τ_0 is the delay of a transition on a single wire, and Δ_i equals 1 for $0 \to 1$ transition, -1 for $1 \to 0$

transition, or 0 for no transition on the i-th wire. As the feature size shrinks, the ratio λ increases, and the crosstalk delay may be several times more than the delay of a single wire and thus dominates the delay of a bus. The crosstalk delay has become a bottleneck in deep sub-micron system-on-chip designs. This problem is so significant that global wiring scaling issues have been identified as Grand Challenges in recent International Technology Roadmap for Semiconductors (ITRS) [1].

Since the crosstalk delay is the major part of the delay, different solutions have been proposed to reduce it, e.g., skewing the timing of signals on the bus [27], bus interleaving, pre-charging, or using repeaters. These solutions have varying degrees of success. Unfortunately, these solutions are often technology-dependent, power consuming, or susceptible to process variation. A technology-independent solution to this problem is *shielding*, which cuts the worst case crosstalk delay by half, but it nearly doubles the wiring area; hence it is unattractive since the routing resource on a chip is scarce.

Crosstalk avoidance codes (CACs) (see, for example, [2,3,28–31]) have emerged as an elegant and promising solution. It not only is technology-independent, but also reduces crosstalk delay while requiring less area and power due to extra wires than shielding. Different codes can be used to make a trade-off between area consumption and worst case delay. For example, a worst-case delay of $(1+2\lambda)\tau_0$ can be achieved through two families of crosstalk avoidance codes: *forbidden transition codes* (FTCs) [2,28] and *forbidden pattern codes* (FPCs) [3]. Furthermore, *one lambda codes* (OLCs) and *forbidden overlap codes* (FOCs) are proposed to reduce the maximum delay to $(1+\lambda)\tau_0$ and $(1+3\lambda)\tau_0$, respectively [31]. Joint coding schemes, which address the crosstalk delay problem as well as the reliability and/or power consumption problem of global buses, have also been considered [30, 31].

Although most CACs in the literature require less area and power overhead due to wires than shielding, extra logic circuits have to be implemented at both ends of the bus as encoders and decoders (CODECs). Unfortunately, most CODEC designs in the literature have very high complexities, rendering CACs-based solutions impractical for wide buses. For example, the CODEC in [4] has an **exponential** complexity with respect to the size of the bus. Researchers have made a lot of effort in finding an efficient way to implement the CODEC of CACs, leading to solutions such as partial coding [31]. In partial coding, a bus is first broken into sub-buses, which are encoded by using CACs with smaller sizes; then a shielding wire is inserted between each pair of adjacent sub-buses to avoid transition patterns with long crosstalk delay. *Forbidden transition overlapping codes* (FTOCs) and *forbidden pattern overlapping codes* (FPOCs) [30] combine partial coding with FTCs and FPCs, respectively. At the expense of a lower code rate and hence larger area and power consumption for the bus, partial coding reduces the complexities of CODECs by keeping the numbers of wires in sub-buses small.

Recently, CODECs based on a Fibonacci-based numeral system (FNS) have effectively solved the complexity problem for FPCs and FTCs [5, 6]. Two FPC CODEC designs are proposed based on an FNS [6], and both CODECs have **quadratic** complexities with the size of the bus. One CODEC in [6] is suboptimal due to its potentially lower code rate, but has a simpler CODEC; the other CODEC in [6] is optimal in its code rate, but requires a more complex circuit. In [5], the FNS is used to encode FTCs. All CODECs in [5, 6] have quadratic complexities.

In this chapter, we generalize the idea in [5, 6] and establish a generic framework for the CODEC design of **all** classes of CACs based on binary mixed-radix numeral systems. Using this framework, we propose CODECs for OLCs and FPCs with optimal code rates as well as CODECs for FOCs with near-optimal code rates. Our implementation results show that all our CAC CODECs have area complexity and delay that increase quadratically with the number of wires. Our main contributions are as follows:

- In Sec. 2.3, generalizing the idea in [5, 6], we propose a generic encoding algorithm for CACs based on numeral systems.
- In Sec. 2.4, we define a **modified** Fibonacci numeral system, and propose an FPC CODEC based on it. Our FPC CODEC achieves the same code rate as the optimal FPC CODEC in [6] and has a simple circuit, similar to the near-optimal FPC CODEC in [6], integrating the advantages of the two FPC CODECs in [6].
- In Sec. 2.5, we define a numeral system for OLC CODECs, and propose an OLC encoding algorithm based on this numeral system. Our CODEC also has a quadratic complexity, which are novel to the best of our knowledge.
- In Sec. 2.6, we first prove that we cannot use the generic CAC encoding algorithm based on numeral systems to encode to the **whole** codebook of an FOC with maximal size. Then we propose an encoding algorithm based on a numeral system that encodes to a **subset** of an FOC with maximal size. For small m, the code rate loss of our suboptimal encoder is small. Our FOC CODECs are also novel to the best of our knowledge.

- In Sec. 2.7, we present implementation results that show our CODECs have area complexities, delays and power consumptions increasing quadratically with m, the number of wires in a bus. We also discuss scenarios where our efficient CODECs are most relevant.

We remark that CODECs for FTCs are not considered in our work for two reasons. First, the FTC CODECs based on the FNS in [5] are already optimal. Second, for a fixed number of wires, since FPCs have a larger codebook size than FTCs, our FPC CODECs with maximal code rates render the investigation of FTC CODECs unnecessary. However, the FTC CODECs proposed in [5] can still be viewed as a special case of our generic CAC CODEC framework.

Our work is inspired by and generalizes the CODECs based on the FNS in [6] and [5]. Our generalization is in two aspects. First, the work in [6] and [5] are for FPCs and FTCs, respectively, whereas our generic CODEC is applicable to FPCs and FTCs as well as other classes of CACs — OLCs and FOCs. Second, the works in [6] and [5] always assume a binary FNS, whereas we consider **all** binary numeral systems in general. In the case of FPCs, this generalization results in either a simpler CODEC or higher code rates (i.e., fewer additional wires for the buses). In the case of OLCs and FOCs, the generalization allows us to further simplify their CODECs.

Compared with partial coding, our efficient CODECs offer a different approach to reducing the area, power, and delay of CAC CODECs. Our efficient CODECs have area, power, and delay that increase quadratically with the number of wires, and achieve the maximal code rates and hence minimize the area and power overheads due to additional wires. On the other hand, CACs based on partial coding introduce smaller area, power, and delay of CODECs since the sub-buses considered are small, but they need more additional wires due to their smaller effective code rates, and thus more area and power overheads due to additional wires. When we integrate these two approaches together (cf. [6, Fig. 9]), our efficient CODECs amplify the benefits of partial coding in two ways. First, our efficient CODECs can be used for the sub-buses for partial coding. Second, our efficient CODECs will allow partial coding schemes to use sub-buses with more wires, leading to fewer shielding wires and hence less area and power overheads. Thus, our efficient CODECs, used with partial coding, help to find the balance between delay and area/power overheads so as to minimize the area and power overheads while satisfying the speed requirement for the bus.

We remark that, as pointed out in [32], the effectiveness of CACs is affected by other factors, such as the synchronization of the switching of all the wires in a bus. Although the effectiveness of CACs is certainly a critical issue to their application, it is beyond the scope of our work. The thrust of our work is to devise efficient CODECs for all classes of CACs, and our efficient CODECs **improve** the effectiveness of CACs in all cases.

2.2 Crosstalk Avoidance Codes

By (2.1), the worst case delay of a global bus is given by $(1 + 4\lambda)\tau_0$. The delay may be reduced by avoiding the transition patterns with a long delay, and hence additional wires are required for an encoded bus, which are the price of the lower crosstalk delay. To measure this redundancy, the *rate* of a CAC is defined to be the ratio between the number of data bits and the number of wires. Based on this idea, different CACs have been proposed, and they have different code rates and can reduce the crosstalk delay to different levels.

2.2.1 $(1+\lambda)$ Codes

The $(1+\lambda)$ codes can achieve a worst case delay of $(1+\lambda)\tau_0$. OLCs, studied in [4], are a kind of $(1+\lambda)$ codes. In an OLC, no adjacent wires can transition in opposite directions when transitioning from one codeword to another. Thus the transition patterns $01 \rightarrow 10$ and $10 \rightarrow 01$ are avoided. Consider a boundary between two adjacent wires. If in all codewords, there are only 00, 01, and 11 across this boundary, it is referred to as a 01-type boundary. Otherwise if only 00, 10, and 11 appear in this boundary, it is referred to as a 10-type boundary. All of the boundaries in an OLC are either 01-type or 10-type. It was proved in [4] that the OLC codebook with maximal size satisfies the following two conditions: (1) The codebook has alternating 01- and 10-type boundaries, and (2) The bit patterns 010, 101, 1001, and 0110 cannot appear in any of the codewords. The maximal cardinality of an m-bit OLC codebook, g_m, satisfies following recursion relation [30]:

$$g_m = g_{m-1} + g_{m-5} \text{ for } m \geq 6 \tag{2.2}$$

with initial conditions $g_1 = 2$, $g_2 = 3$, $g_3 = 4$, $g_4 = 5$, and $g_5 = 7$. We are not aware of efficient OLC CODEC designs in the literature.

2.2.2 $(1+2\lambda)$ Codes

The $(1+2\lambda)$ codes can achieve a worst case delay of $(1+2\lambda)\tau_0$. FTCs [2] and FPCs [3] are two families of $(1+2\lambda)$ codes, and they are the CACs that have been studied mostly. In an FTC, the transition patterns $01 \rightarrow 10$ and $10 \rightarrow 01$ are avoided, which is called *forbidden transition* (FT) condition. The size of the largest codebook of an m-bit FTC is given by F_{m+2} [2], where $\{F_n\}$ is a Fibonacci sequence satisfying $F_n = F_{n-1} + F_{n-2}$ for $n \geq 3$, and $F_1 = F_2 = 1$. In an FPC, the bit patterns 010 and 101 are avoided, which is called *forbidden pattern* (FP) condition. The codebook size of an m-bit FPC is given by $2F_{m+1}$ [3], slightly greater than that of an m-bit FTC. Since the number of codewords needed is a power of two, an m-bit FPC leads to a higher rate than an m-bit FTC when there exists an l such that $F_{m+2} < 2^l \leq 2F_{m+1}$.

An efficient FTC CODEC was proposed in [5] based on the FNS, where a binary string $d_m d_{m-1} \cdots d_1$ represents $v = \sum_{i=1}^{m} d_i F_i$. The CODEC in [5] can encode all of the FTC codewords with a quadratic complexity and thus it is optimal. A similar CODEC based on the FNS is designed for FPCs in [6]. However, the number of integers that can be represented by the FNS is given by $1 + \sum_{i=1}^{m} F_i = F_{m+2}$, which is less than $2F_{m+1}$, the maximal cardinality of an m-bit FPC codebook. Thus we can only encode a subset of the FPC codebook based on the FNS directly. This CODEC is referred to as a near-optimal FPC CODEC in [6]. An optimal FPC CODEC design proposed in [6] solves this problem by employing extra logic to encode the integers that cannot be encoded by the FNS. However, the extra logic makes the CODEC more complex.

2.2.3 $(1 + 3\lambda)$ Codes

The $(1 + 3\lambda)$ codes can achieve a worst case delay of $(1 + 3\lambda)\tau_0$. FOCs are a kind of $(1 + 3\lambda)$ CACs [31]. We say a 3 bit pattern $b_1b_2b_3$ appears *around* a bit d_i if $d_{i+1}d_id_{i-1} = b_1b_2b_3$. The FOC codebook satisfies the following constraint: the codebook cannot have both 010 and 101 appearing around any bit position. The maximal size of an m-bit FOC is given by T_m, where

$$T_m = T_{m-1} + T_{m-2} + T_{m-3} \text{ for } m \geq 4 \tag{2.3}$$

and $T_1 = 2$, $T_2 = 4$, and $T_3 = 7$ [31]. We are not aware of efficient CODEC designs for FOCs in the literature.

2.3 Generic CAC CODEC Designs Based on Numeral Systems

2.3.1 Introduction to Numeral Systems

A numeral system is a linguistic system and mathematical notation for representing numbers of a given set by symbols in a consistent manner [33]. The most commonly used numeral systems are positional numeral systems [33], where given a positive natural number b, a string $(a_m \cdots a_2a_1)_b$ represents a number $\sum_{i=1}^{m} a_ib^{i-1}$. For example, the binary and decimal numeral systems use powers of two and powers of ten, respectively, as bases. A *binary mixed-radix numeral system* is that given a basis set of non-negative numbers $\{\gamma_m, \cdots, \gamma_2, \gamma_1\}$, a binary string $(d_m \cdots d_2d_1)$ represents a number $\sum_{i=1}^{m} d_i\gamma_i$. In our research, we focus on binary mixed-radix numeral systems henceforth.

A numeral system is *complete* if any integer $u \in [0, \sum_{i=1}^{m} \gamma_i]$ can be represented by at least one binary string $d_m d_{m-1} \cdots d_1$. To determine whether a numeral system is complete, we have the following lemma:

Lemma 2.1. *Given a basis set of non-negative numbers $\{\gamma_i\}_{i=1}^{m}$, suppose there is a permutation of these numbers such that for all $2 \leq k \leq m$, $\gamma_k \leq 1 + \sum_{i=1}^{k-1} \gamma_i$, and $\gamma_1 = 1$, then the numeral system defined by $\{\gamma_k\}_{k=1}^{m}$ is complete.*

Proof. Without loss of generality, suppose $\{\gamma_i\}_{i=1}^{m}$ has already been ordered with the property stated in Lemma 2.1. When $m = 1$, this numeral system is complete because the two numbers 0 and 1 can be represented by strings $d_1 = 0$ and $d_1 = 1$ respectively. Suppose this lemma holds for $m = n$. When $m = n+1$, $\{\gamma_i\}_{i=1}^{n}$ defines a complete numeral system, and all the number less than or equal to $\sum_{i=1}^{n} \gamma_i$ can be represented by at least a bit string $d_n d_{n-1} \ldots d_1$. Consider an integer $u \leq \sum_{i=1}^{n+1} \gamma_i$. If $u \leq \sum_{i=1}^{n} \gamma_i$, let $d_{n+1} = 0$, and $d_n d_{n-1} \cdots d_1$ be the representation of u in the numeral system defined by $\{\gamma_i\}_{i=1}^{n}$. If $1 + \sum_{i=1}^{n} \gamma_i \leq u \leq \sum_{i=1}^{n+1} \gamma_i$, $v = u - \gamma_{n+1} \leq \sum_{i=1}^{n} \gamma_i$ and $v \geq (1 + \sum_{i=1}^{n} \gamma_i) - \gamma_{n+1} \geq 0$. Thus $v \in [0, \sum_{i=1}^{n} \gamma_i]$ can be represented by a binary string $d_n d_{n-1} \cdots d_1$, and let $d_{n+1} = 1$, we have an $(n+1)$-bit binary string representation of u. Thus all the number less than or equal to $\sum_{i=1}^{n+1} \gamma_i$ can be represented by an $(n+1)$-bit binary string. $\square$

Note that if $\gamma_i > 0$ for $1 \leq i \leq m$ and $\gamma_j \geq \gamma_i$ for $1 \leq i \leq j \leq m$, then the numeral system defined by $\{\gamma_k\}_{k=1}^{m}$ is complete if and only if for all $2 \leq k \leq m$, $\gamma_k \leq 1 + \sum_{i=1}^{k-1} \gamma_i$, and $\gamma_1 = 1$. Nevertheless, Lemma 2.1 is enough for our CODEC design.

2.3.2 Generic CAC Encoding Algorithm

Suppose we want to transmit a k-bit data message over a bus with m ($m \geq k$) wires in one clock cycle. These k bits are first encoded into an m-bit CAC codeword so that the transition patterns with long crosstalk delays are avoided. The m-bit CAC codeword is then transmitted over the bus and received by the decoder. Then the k-bit message is recovered at the decoder.

The idea of numeral system based CAC CODEC is that the k-bit data message can be viewed as an integer v such that $0 \leq v \leq 2^k - 1$ in the binary numeral system, and the goal of encoding algorithm is to convert v into an m-bit binary string, which represents v under another numeral system and has no transition pattern with long crosstalk delay. Since the encoded codeword contains only 0 and 1 and the numeral system needs to be complete, we have to use a binary mixed-radix numeral system.

Consider an m-bit CAC codebook $\mathcal{C}(m)$ with size $|\mathcal{C}(m)|$. If a numeral system $\{\gamma_k\}_{k=1}^{m}$ is used to encode $\mathcal{C}$, we consider an encoder, which is essentially a mapping f from all integers in $[0, |\mathcal{C}(m)| - 1]$ to $\mathcal{C}$ with the following properties:

- All the codewords can be mapped from an integer in $[0, |\mathcal{C}(m)| - 1]$, which implies that f is surjective;
- Different codewords represent different integers under the mapping.

Therefore f is a bijection from integers in $[0, |\mathcal{C}(m)| - 1]$ to $\mathcal{C}$. Then we have the following lemma for f:

Lemma 2.2. *$f^{-1}(00\cdots0) = 0$, and $f^{-1}(11\cdots1) = |\mathcal{C}(m)| - 1$ when $\gamma_i \geq 0$ for $1 \leq i \leq m$.*

Proof. Since the codewords $00\cdots0$ and $11\cdots1$ exist in all CACs, they are mapped from the integers that can be represented by the numeral system respectively. $f^{-1}(00\cdots0) = \sum_{i=1}^{m} 0 \cdot \gamma_i = 0$. Obviously, $f^{-1}(11\cdots1) = \sum_{i=1}^{m} \gamma_i$ is the largest integer that can be represented by this numeral system since the bases of the numeral system are non-negative, thus it is $|\mathcal{C}(m)| - 1$. □

Note that we assume that f is a bijection from the integers in $[0, |\mathcal{C}(m)| - 1]$ to $\mathcal{C}(m)$. Of course, one may find another set of $|\mathcal{C}(m)|$ integers to construct this bijection, but this makes no difference for FTCs, FPCs, and OLCs. However, for FOCs with maximal sizes, we cannot establish such bijection f based on numeral system, which is proved in Proposition 2.12. It is still not clear that if there is a set of T_m integers to help us derive an FPC CODEC with a quadratic complexity.

We propose a generic CAC encoding algorithm based on a numeral system in Alg. 1 below. In Alg. 1, $\{\gamma_1, \gamma_2, \cdots, \gamma_m\}$ is the basis set of the encoding numeral system, $\{\alpha_i\}_{i=1}^{m}$, $\{\beta_i\}_{i=1}^{m}$, and Θ are some constants depending on the CACs. $d_m d_{m-1} \cdots d_1$ is the output of the encoding algorithm; also it is a codeword in the CAC. It is easy to see that the data message is recovered by computing $v = \sum_{i=1}^{m} d_i \gamma_i$.

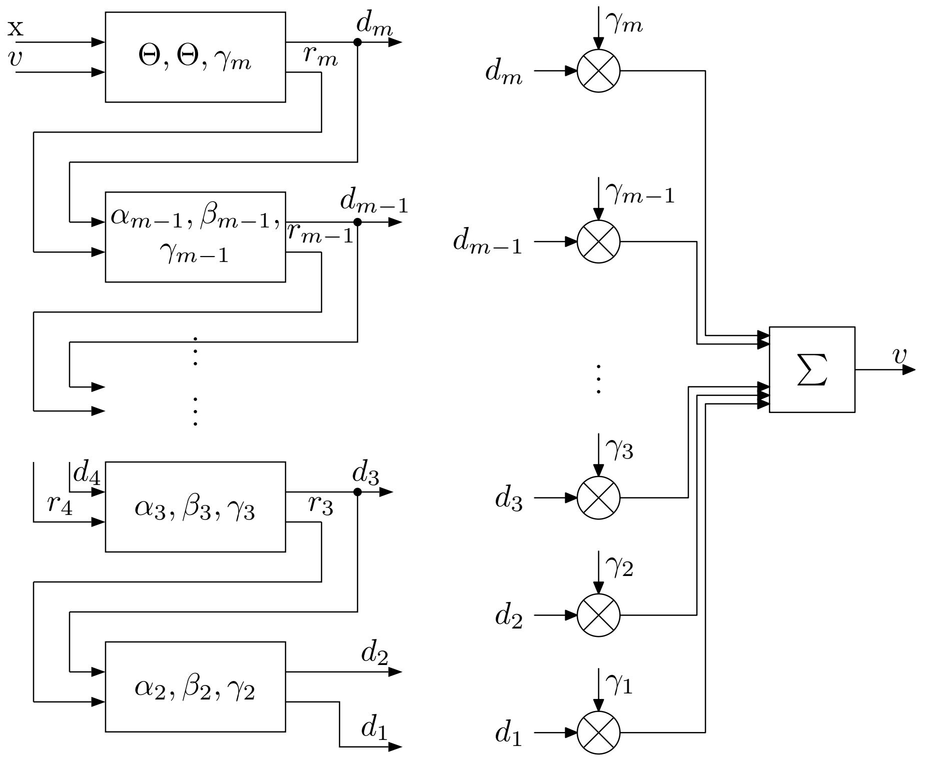
The CODEC for a CAC based on Alg. 1 is shown in Fig. 2.1. The encoder consists of $m - 1$ processing elements, and all processing elements have the same circuit, shown in Fig. 2.2. The top processing element is slightly different from the others in that $\alpha_m = \beta_m = \Theta$, which renders the input d_{m+1} inconsequential (the input d_{m+1} to the top processing element is don't care in Fig. 2.1). Each processing

Algorithm 1 Generic CAC encoding algorithm

```
Input: code length $m$, integer $v$ ($0 \le v \le \sum_{i=1}^{m} \gamma_i$).
 1: for $k = m$ downto 2 do
 2:    if $k = m$ then
 3:       if $v \ge \Theta$ then
 4:          $d_m = 1$;
 5:       else
 6:          $d_m = 0$;
 7:       end if
 8:       $r_m = v - d_m \cdot \gamma_m$;
 9:    else
10:       if $r_{k+1} \ge \alpha_k$ then
11:          $d_k = 1$;
12:       else if $r_{k+1} < \beta_k$ then
13:          $d_k = 0$;
14:       else
15:          $d_k = d_{k+1}$;
16:       end if
17:       $r_k = r_{k+1} - d_k \cdot \gamma_k$;
18:    end if
19: end for
20: $d_1 = r_2$;
21: Output: $d_m d_{m-1} \cdots d_1$.
```

element consists of two comparators, one subtractor, and one multiplexer. Each processing element has three parameters α_k, β_k, and γ_k, two inputs d_{k+1} and r_{k+1}, and two outputs d_k and r_k.

We observe the similarities between Alg. 1 and [6, Alg. 2] for FPCs, between the generic CODEC in Fig. 2.1 and the FPC CODEC shown in [6, Figure 3], as well as between the processing element in Fig. 2.2 and that in [6, Figure 4]. Despite these similarities, there are some key differences. The CODEC in [6, Alg. 2 and Figure 3] is for a near-optimal FPC, and the constants used therein are from the FNS. By finding proper numeral systems (not necessarily Fibonacci) $\{\gamma_k\}$ and constants Θ,


x
v
Θ, Θ, γ_m
d_m
r_m
$\alpha_{m-1}, \beta_{m-1}, \gamma_{m-1}$
d_{m-1}
r_{m-1}
d_4
r_4
$\alpha_3, \beta_3, \gamma_3$
d_3
r_3
$\alpha_2, \beta_2, \gamma_2$
d_2
d_1
γ_m
γ_{m-1}
γ_3
γ_2
γ_1
Σ
v

(a) encoder (b) decoder

Figure 2.1: The generic CODEC of an m-bit CAC based on Alg. 1 (note the similarity to the CODEC shown in [6, Figure 3]).

$\{\alpha_k\}$, and $\{\beta_k\}$, our generic CODEC can be used for all classes of CACs — OLCs, FPCs, and FOCs — with different worst-case crosstalk delay. Furthermore, the processing element in our generic CODEC can be further simplified for OLCs and FOCs, which will be shown in Sec.s 2.5 and 2.6.

Since Alg. 1 is generic and applicable to all CACs, Alg. 1 includes the encoding algorithms based on the FNS in [5, 6] as special cases. For example, the FNS based FTC encoding algorithm (cf. [5, Alg. 1]) is a special case of Alg. 1, where $\Theta = F_{2\lfloor \frac{m}{2} \rfloor + 1}$, $\alpha_k = \beta_k = F_{2\lfloor \frac{k}{2} \rfloor + 1}$, and $\gamma_k = F_k$. Furthermore, the near-optimal FPC encoding algorithm (cf. [6, Alg. 2]) is also a special case of Alg. 1, where $\Theta = F_{m+1}$,

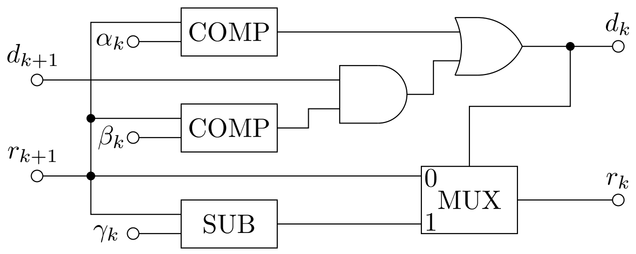


Figure 2.2: The processing element of the encoder in Fig. 2.1 (note the similarity to [6, Figure 4]).

$\gamma_k = F_k$, $\alpha_k = F_{k+1}$, and $\beta_k = F_k$.

We remark that our CODEC in Fig. 2.1 has a quadratic complexity with m. This can be verified in two ways. First, we note that each parameter as well as each input for one processing element has at most m bits. Since the comparators, subtractor, and multiplexer in Fig. 2.2 all can be implemented with linear complexity with m, each processing element also has a linear complexity and hence the encoder has a quadratic complexity. The decoder is an m-input adder with each input an m-bit number, and hence it also has a quadratic complexity. Thus our CODEC in Fig. 2.1 has a quadratic complexity. Second, this is confirmed by implementation results, presented in Sec. 2.7, of our CODECs based on this generic CODEC.

We finally observe that our CODEC in Fig. 2.1 has a simple and regular circuit, and it can achieve high throughput via pipelining if desired.

2.4 The FPC CODEC Design

2.4.1 Numeral Systems for FPC CODECs

Let $\{F_k\}$ be a Fibonacci sequence. We have the following theorem:

Theorem 2.3. *When $m \geq 2$, the set $\{p_k\}_{k=1}^{m}$ defines a complete numeral system, where $p_m = F_{m+1}$ and $p_k = F_k$ for $1 \leq k \leq m-1$.*

Proof. We have a property of Fibonacci sequence: $1 + \sum_{i=1}^{n} F_i = F_{n+2} > F_{n+1}$ [5]. Thus the numeral system defined by $\{p_k\}_{k=1}^{m}$ is complete. □

In the following, the numeral system defined by $\{p_k\}_{k=1}^{m}$ is referred to as a *modified Fibonacci numeral system* (MFNS).

2.4.2 The FPC CODEC Design

With the help of the MFNS, the FPC CODEC can be designed as a special case of our general CAC CODEC by choosing

$$\gamma_k = p_k,\ \Theta = F_{m-1},\ \alpha_k = F_{k+1},\ \beta_k = F_k. \tag{2.4}$$

We will show in Theorem 2.4 that this algorithm is correct and optimal that each integer in $[0, 2F_{m+1} - 1]$ can be encoded into one codeword in the FPC codebook. The decoding algorithm is given by the formula $v = \sum_{i=1}^{m} d_i p_i$.

Theorem 2.4. *The output of Alg. 1 with the constants specified in (2.4) is an FPC codeword.*

Proof. As in [6], the correctness of the FPC encoding algorithm can be proved by showing that if the partially generated output vector $d_m \cdots d_{k+1} d_k$ after the k-th ($k \leq m-2$) stage has no forbidden pattern, adding the output of the $(k-1)$-th stage, d_{k-1}, will not introduce a forbidden pattern. For $k < m-2$, the proof is the same as the proof in [6] since our encoding algorithm is the same as [6, Alg. 2], and thus it suffices to prove the case when $k = m - 2$.

If $d_m = 0$, we know $r_m = v < F_{m+1}$. If $d_{m-1} = 0$, no forbidden pattern will be generated after adding d_{m-2}, the output of the $(m-2)$-th stage. If $d_{m-1} = 1$, $r_m \geq F_m$, and then $r_{m-1} = r_m - F_{m-1} \geq F_m - F_{m-1} = F_{m-2}$, implying that $d_{m-2} = d_{m-1} = 1$.

If $d_m = 1$, we know $F_{m+1} \leq v \leq 2F_{m+1} - 1$, and $0 \leq r_m \leq F_{m+1} - 1$. If $d_{m-1} = 1$, no forbidden pattern will be generated regardless of d_{m-2}. If $d_{m-1} = 0$, then $r_{m-1} = r_m < F_{m-1}$, implying $d_{m-2} = 0$. $\square$

As a special case of our generic CAC CODEC, the circuitry of our FPC CODEC design has a quadratic complexity. In comparison to the near-optimal FPC encoder in [6, Fig. 3], our FPC encoder uses a different formula to generate the most significant bit and our decoder in multiplies d_m with F_{m+1} as opposed to F_m in [6, Fig. 3]. Therefore our CODEC design has a larger codebook with the same complexity of the near optimal FPC encoder, resulting in a slightly higher code rate when there exists l such that $2F_{m+1} \geq 2^l \geq F_{m+2}$. Compared with the optimal FPC CODEC in [6] (cf. [6, (21) and (22)]), our CODEC achieves the same code rate, and has a slightly simpler circuit, thus achieving a shorter latency and requiring a smaller area. As for the CODECs in [6], high throughput can be achieved for our CODEC through pipelining if necessary.

2.5 The OLC CODEC Design

2.5.1 Numeral Systems for OLC CODECs

Based on the property of the sequence $\{g_i\}_{i=1}^{\infty}$ defined in (2.2), we have the following theorem:

Theorem 2.5. *For $m \geq 4$, let $\{f_i\}_{i=1}^{m}$ be defined by $f_m = g_{m-3}$, $f_i = g_{i-1} - g_{i-2}$ for $3 \leq i \leq m-1$, $f_2 = 0$, and $f_1 = 1$. Then $\{f_i\}_{i=1}^{j}$ defines a complete numeral system for $j \leq m$.*

Proof. It is easy to see that all the entries in the sequence $\{g_k\}_{i=1}^{\infty}$ are **positive** and **strictly increasing**. Since $g_k = g_{k-1} + g_{k-5}$, we can see that $g_{k-1} < g_k < 2g_{k-1}$. Since $\sum_{i=1}^{m-1} f_i = g_{m-2} - g_1 + 1 = g_{m-2} - 1$, $f_m = g_{m-3} \leq g_{m-2} - 1 = \sum_{i=1}^{m-1} f_i$. If $2 < k < m$, $f_k = g_{k-1} - g_{k-2} \leq g_{k-2} - 1 = \sum_{i=1}^{k-1} f_i$. Since $f_2 = 0 < f_1 = 1$, $\{f_k\}_{k=1}^{j}$ defines a complete numeral system for $j \leq m$ by Lemma 2.1. □

2.5.2 The OLC CODEC Design

The maximal cardinality of an m-bit OLC codebook is given by g_m as defined in (2.2) [30]. The numeral system defined by $\{f_i\}_{i=1}^{m}$ can be used to encode an m-bit OLC. Construct a sequence $\{G_n\}$ such that $G_n = G_{n-1} + G_{n-5}$ for $n \geq 6$ with initial conditions $G_1 = G_3 = G_4 = G_5 = 1$ and $G_2 = 0$. By inspection, we can easily verify that $G_n = g_{n-6}$ for $n > 6$, which means that $\{G_n\}$ is an extension of the sequence $\{g_n\}$. We also note that $f_n = G_n$ when $n < m$, and $f_m = G_{m+3}$. Therefore, by setting the constants in Alg. 1 as follows,

$$\gamma_k = f_k,\ \Theta = G_{2\lfloor \frac{m}{2} \rfloor + 4},$$
$$\alpha_k = \begin{cases} G_{2l+2} & k = 2l-1, \\ \infty & k = 2l, \end{cases} \tag{2.5}$$
$$\beta_k = \begin{cases} 0 & k = 2l-1, \\ G_{2l+2} & k = 2l, \end{cases}$$

the numeral system defined by $\{f_i\}_{i=1}^{m}$ can be used to encode the OLC by Alg. 1. The decoding algorithm is given by the formulae $v = \sum_{i=1}^{m} d_i f_i$.

We first establish four technical results, all of which are instrumental in proving the correctness of our OLC encoding algorithm.

Lemma 2.6. *$G_n = G_{n-2} + G_{n-3}$ for $n \geq 4$.*

Proof. For $4 \leq n \leq 8$, Lemma 2.6 can be shown by inspection. Suppose it holds for all n up to $k \geq 8$. When $n = k+1$,

$$\begin{aligned} G_{k+1} &= G_k + G_{k-4} \\ &= G_{k-2} + G_{k-3} + G_{k-6} + G_{k-7} \\ &= G_{k-1} + G_{k-2}. \end{aligned}$$

Thus the lemma holds for all $n \geq 4$. □

Since $\{G_n\}$ is an extension of sequence $\{g_n\}$, Lemma 2.6 implies that

Lemma 2.7. *$g_n = g_{n-2} + g_{n-3}$ for $n \geq 4$.*

Lemma 2.7 gives a **novel** recursive relation of the sequence $\{g_n\}$. Note that $\{g_n\}$ is the maximal cardinality of an m-bit OLC codebook [30].

Lemma 2.8. $1 + \sum_{i=1}^{2n-1} G_i = G_{2n+4}$ *for* $n \geq 1$.

Proof. If $n = 1$, then $G_1 = 1$, $G_6 = g_1 = 2$, and thus $1 + G_1 = G_6$. Suppose that Lemma 2.8 holds for $n \leq k$. When $n = k + 1$,

$$\begin{aligned} 1 + \sum_{i=1}^{2k+1} G_i &= 1 + \sum_{i=1}^{2k-1} G_i + G_{2k+1} + G_{2k} \\ &= G_{2k+4} + G_{2k+3} \\ &= G_{2k+6} \end{aligned}$$

by Lemma 2.6. Thus this lemma holds for all $n \geq 1$. □

To show that the output of the OLC encoding algorithm is an OLC codeword, we need to show that it is a binary string first. From the algorithm, when $k \geq 2$, d_k is either 0 or 1. Then it suffices to show that d_1 can only be either 0 or 1. To prove this, we need Lemma 2.9.

Lemma 2.9. *When the constants of Alg. 1 are set as specified in (2.5) ,* $0 \leq r_k \leq \sum_{i=1}^{k-1} f_i$ *for each integer* $k \in [2, m]$.

Proof. We prove this property by induction on k from m to 2. First we are going to show that $r_k \leq \sum_{i=1}^{k-1} f_i$. Since the input integer $v \leq g_m - 1 = \sum_{i=1}^{m} f_i$ by Theorem 2.5, if $d_m = 1$, $r_m = v - f_m \leq \sum_{i=1}^{m-1} f_i$, and if $d_m = 0$, $r_m = v \leq G_{2\lfloor \frac{m}{2} \rfloor + 4} - 1 = \sum_{i=1}^{2\lfloor \frac{m}{2} \rfloor - 1} G_i \leq \sum_{i=1}^{m-1} f_i$ by Lemma 2.8. For $2 \leq k < m$, suppose that $r_{k+1} \leq \sum_{i=1}^{k} f_i$. It is obvious $r_k \leq \sum_{i=1}^{k-1} f_i$ if d_k is encoded as 1. Therefore it suffices to consider the case when d_k is encoded as 0. If $k = 2l + 1$, then $d_{2l+1} = 0$ implies that $r_{2l+1} = r_{2l+2} \leq G_{2l+4} - 1 = \sum_{i=0}^{2l-1} f_i < \sum_{i=0}^{2l} f_i$. If $k = 2l$, then $d_{2l} = 0$ implies that either $r_{2l} = r_{2l+1} \leq G_{2l+2} - 1 = \sum_{i=1}^{2l-3} f_i < \sum_{i=1}^{2l-1} f_i$, or $d_{2l} = d_{2l+1} = 0$. In the latter case, $d_{2l+1} = 0$ implies that $r_{2l+1} \leq \sum_{i=0}^{2l-1} f_i$, thus $r_{2l} = r_{2l+1} \leq \sum_{i=0}^{2l-1} f_i$.

Next, we are going to show that $r_k \geq 0$. If $d_m = 0$, then $r_m = v \geq 0$. If $d_m = 1$, then $r_m = v - f_m \geq G_{2\lfloor \frac{m}{2} \rfloor + 4} - G_{m+3}$. If m is even, $r_m \geq G_{m+4} - G_{m+3} = G_{m-1} \geq 0$, and if m is odd, then $r_m \geq G_{m+3} - G_{m+3} = 0$. Suppose at step $k+1$, $r_{k+1} \geq 0$. Then at step k, if $d_k = 0$, then $r_k = r_{k+1} \geq 0$. Suppose $d_k = 1$ and k is even. Therefore $r_{k+1} \geq G_{k+2}$, and $r_k = r_{k+1} - f_k \geq G_{k+2} - G_k = G_{k-1} \geq 0$. If $d_k = 1$ and k is odd, there are two possible cases: either $d_k = 1$ because $r_{k+1} \geq G_{k+3}$ or $d_k = d_{k+1} = 1$. In the first case, $r_k = r_{k+1} - f_k \geq G_{k+3} - G_k = G_{k+1} \geq 0$. In the latter case, since $k+1$ is even, by the above argument, $d_{k+1} = 1$ implies that $r_{k+1} \geq G_k = f_k$, and $r_k = r_{k+1} - f_k \geq 0$. Then we have $r_k \geq 0$ at each step. $\square$

By Lemma 2.9, $0 \leq r_2 \leq f_1 = 1$, therefore d_1 is either 0 or 1. From our OLC encoding algorithm, it is also easy to see that $v = \sum_{i=1}^{m} d_i f_i$. Thus the output of our OLC encoding algorithm cannot be identical binary string for different integers.

Theorem 2.10. *The output of Alg. 1 is an OLC codeword by setting the constants as specified in (2.5).*

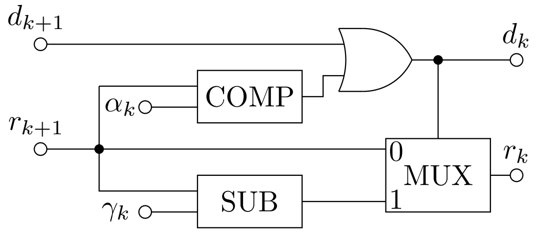
Proof. First, let us prove that the output of our OLC encoding algorithm has alternating 01- and 10-type boundaries. At the $2l$-th stage, the output d_{2l} is 1 only if $d_{2l+1} = 1$. Thus the 01 pattern is avoided at the boundary $d_{2l+1}d_{2l}$, and this is a 10-type boundary. At the $(2l-1)$-th stage, the output d_{2l-1} is 0 only if $d_{2l} = 0$. Thus the 10 pattern is avoided at the boundary $d_{2l}d_{2l-1}$, and this is a 01 boundary. Thus the output $d_m d_{m-1} \cdots d_1$ has alternating 01- and 10-type boundaries.

Next, let us show that the bit patterns 101 and 010 are avoided in the output of the OLC encoding algorithm. From the constraint of alternating boundary type, only $d_{2l+1}d_{2l}$ can have an output 10. Thus $r_{2l} = r_{2l+1} \leq \sum_{i=1}^{2l-3} G_i < G_{2l+2}$ by Lemma 2.8, and d_{2l-1} will be zero as well; hence 101 is avoided in the output vector. Similarly, bit pattern 01 appears only in $d_{2l+2}d_{2l+1}$. Thus $r_{2l+1} = r_{2l+2} - f_{2l+1} \geq G_{2l+4} - G_{2l+1} = G_{2l+2}$. Thus d_{2l} is the same as d_{2l+1}, and hence 010 is avoided in the output vector.

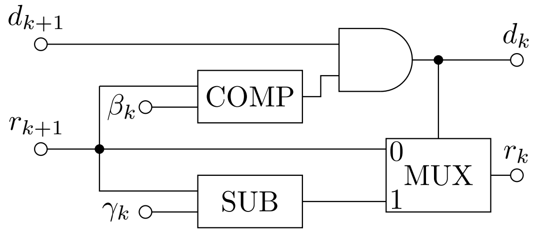
The bit patterns 1001 and 0110 violate the alternating boundary type constraint, and thus they cannot appear in the output vector. Thus the output vector is an OLC codeword. $\square$

Since the largest codebook size of an m-bit OLC is g_m, and we can get g_m different codewords satisfying the constraints of OLC codewords by the OLC encoding algorithm, the algorithm gives a bijection from integers in $[0, g_m - 1]$ to the m-bit OLC codebook, implying that the algorithm is optimal.

Our numeral system based OLC CODEC has a quadratic complexity since it is a special case of Alg. 1, and thus the same structure of the general CODEC in Fig. 2.1 can be applied to implement OLC CODEC. Note that $\alpha_{2l} = \infty$ and $\beta_{2l-1} = 0$, all the processing element except the top one can be simplified to have only one comparator instead of two in the general case, which is shown in Fig.2.3. The decoder of our OLC CODEC is implemented by the formula $v = \sum_{i=1}^{m} d_i f_i$, which is an m-input adder. We can also design a high throughput CODEC by pipelining if desired.



(a) processing element circuit when k is odd.



(b) processing element circuit when k is even.

Figure 2.3: The OLC CODEC processing element.

2.6 The FOC CODEC Design

In Sec. 2.6.1, we first show that unfortunately, it is impossible to find constants $\{\alpha_k\}$, $\{\beta_k\}$, $\{\gamma_k\}$, and Θ, so that our generic CAC encoding algorithm can be used to encode all the codewords in the largest codebook of an m-bit FOC when the code length $m \geq 4$. Instead, we propose an FOC which is a subset of the largest FOC codebook that can be encoded with Alg. 1 in Sec. 2.6.2.

2.6.1 Numeral Systems for FOC CODECs

We will show in Proposition 2.12 that when $m \geq 4$, no numeral system can establish a bijection from the integers $[0, T_m - 1]$ to the FOC codebook $\mathcal{C}_{\text{FOC}}(m)$ with the maximal size T_m. Before proving Proposition 2.12, we are going to prove a lemma.

Lemma 2.11. *If a numeral system $\{\gamma_i\}_{i=1}^m$ establishes a bijection $f : S_m \mapsto \mathcal{C}_{\text{FOC}}(m)$ where S_m is the set of integers in $[0, T_m - 1]$, and for a codeword $\mathbf{c} = (c_1, c_2, \cdots, c_m)$, $f(\sum_{i=1}^m c_i\gamma_i) = \mathbf{c}$, then*

(a) $\gamma_i \neq \gamma_j$ for $1 \leq i < j \leq m$;

(b) $\gamma_i > 0$ for all $1 \leq i \leq m$;

(c) either $\gamma_1 = 1$ and $\gamma_m = 2$, or $\gamma_1 = 2$ and $\gamma_m = 1$ if $m \geq 2$;

(d) if $m \geq 4$, $\gamma_i \neq 3$ and $\gamma_i \neq 4$, for all $1 \leq i \leq m$.

Proof. There are two types of m-bit FOCs that both achieve the maximal $(1 + 3\lambda)$ codebook size. One type avoids 101 pattern around even bit position, and avoids 010 around odd bit position; the other type avoids 010 pattern around even bit position, and avoids 101 pattern around odd bit position. We will prove Lemma 2.11 by considering only the first type of FOCs. A similar argument holds for the second type of FOCs.

Part (a). There is a codeword $\mathbf{c}_1$ with alternating ones and zeros with $d_1 = 0$ in $\mathcal{C}_{\text{FOC}}(m)$. We can obtain another codeword $\mathbf{c}_2$ in $\mathcal{C}_{\text{FOC}}(m)$ by flipping a bit d_{2p} from 1 to 0 and a bit d_{2q+1} from 0 to 1, where $1 \leq 2p, 2q+1 \leq m$. Then we have $\gamma_{2p} \neq \gamma_{2q+1}$, otherwise we will have $f^{-1}(\mathbf{c}_2) = f^{-1}(\mathbf{c}_1) - \gamma_{2p} + \gamma_{2q+1} = f^{-1}(\mathbf{c}_1)$, which contradicts that f is a bijection. Consider two codewords $\mathbf{c}_3$ and $\mathbf{c}_4$ with all zeros except a single one at d_{2p} and d_{2q}, respectively. $f^{-1}(\mathbf{c}_3) \neq f^{-1}(\mathbf{c}_4)$ implies $\gamma_{2p} \neq \gamma_{2q}$. Consider two codewords $\mathbf{c}_5$ and $\mathbf{c}_6$ with all ones except a single zero at d_{2p+1} and d_{2q+1}, respectively. $f^{-1}(\mathbf{c}_5) \neq f^{-1}(\mathbf{c}_6)$ implies $T_m - 1 - \gamma_{2p+1} \neq T_m - 1 - \gamma_{2q+1}$, and hence $\gamma_{2p+1} \neq \gamma_{2q+1}$. Summarizing all these results, we have $\gamma_i \neq \gamma_j$ for $1 \leq i \neq j \leq m$.

Part (b). Suppose $\gamma_i = 0$. If i is even, the all zeros codeword and the codeword with all zeros except a single 1 at d_i coexist in $\mathcal{C}_{\text{FOC}}(m)$, and they represent the same integer 0. If i is odd, then the all ones codeword and the codeword with all ones except a single 0 at d_i coexist in $\mathcal{C}_{\text{FOC}}(m)$, and they represent the same integer $T_m - 1$. In either case, f is no longer a bijection. Therefore γ_i is positive for $1 \leq i \leq m$.

Part (c). To make our numeral system defined by $\{\gamma_i\}_{i=1}^m$ complete, there is one and only one element $\gamma_i = 1$ for some $1 \leq i \leq m$ by Lemma 2.11(a). By Lemma 2.11(b), there is only one way to map the integer 1 to an FOC codeword, which is setting all bits to 0 except $d_i = 1$. Similarly setting all bits to 1 except $d_i = 0$ is the only way to represent the integer $T_m - 2$. Thus i can not be a value other than 1 or m since otherwise we will have both 010 and 101 centered around d_i. Since we cannot have two elements in the base set equal to 1 by Lemma 2.11(a), we need a 2 appeared in the base set to represent the integer 2. A similar argument shows that if $\gamma_i = 2$, then i is either 1 or m.

Part (d). If $m \geq 4$, then the following 8 codewords $000\cdots000$, $000\cdots001$, $100\cdots000$, $100\cdots001$, $011\cdots110$, $011\cdots111$, $111\cdots110$, and $111\cdots111$ are all in $\mathcal{C}_{\text{FOC}}(m)$. They are mapped from integers in $[0, 3]$ and $[T_m-4, T_m-1]$ by Lemmas 2.2 and 2.11(c). If $\gamma_i = 3$, then i has to be an integer lies in $[2, m-1]$ because γ_1 and γ_m have to take the values 1 and 2 by Lemma 2.11(c). Thus either the codeword with all zeros except a single 1 at d_i or the codeword with all ones except a single 0 at d_i is in $\mathcal{C}_{\text{FOC}}(m)$, which will be mapped to the same integer that can be represented by one of the aforementioned eight codewords, and f is no longer a bijection. Therefore $\gamma_i \neq 3$ for all $1 \leq i \leq m$. If $\gamma_i = 4$, by Lemma 2.11(b), we have to use the codeword with all zeros except $d_i = 1$ to represent the integer 4, and the codeword with all ones except $d_i = 0$ to represent the integer $T_m - 5$ at the same time. Since i can be neither 1 nor m, both bit patterns 010 and 101 appear around d_i in $\mathcal{C}_{\text{FOC}}(m)$, and the FOC constraint is violated. Therefore $\gamma_i \neq 4$ for all $1 \leq i \leq m$. □

Theorem 2.12. *For $m \geq 4$, no numeral system can establish bijection from S_m to an FOC codebook.*

Proof. By Lemma 2.11(d), we cannot map the integer 4 to an FOC codeword when $m \geq 4$. Thus we cannot find a numeral system to establish a bijection from $[0, T_m - 1]$ to an FOC codebook. □

We acknowledge that our proof relies on $S_m = \{0, 1, \cdots, m\}$, which is required by our generic encoding algorithm. In general, one may find a set of T_m distinct integers that can be mapped from S_m efficiently and can be expressed by the codewords in $\mathcal{C}_{\text{FOC}}(m)$ under some numeral systems. However, it is not clear now how these integers can be designed and encoded into $\mathcal{C}_{\text{FOC}}(m)$ while achieving a quadratic complexity.

2.6.2 The Sub-optimal FOC CODEC Design

Consider a set $\mathcal{C}'_{\text{FOC}}(m)$ of m-bit codewords derived by avoiding bit pattern 10 on the boundaries $d_{2l+1}d_{2l}$. It is easy to see that $\mathcal{C}'_{\text{FOC}}(m)$ is actually a subset of $\mathcal{C}_{\text{FOC}}(m)$, because it does not have 010 on $d_{2l+2}d_{2l+1}d_{2l}$, and does not have 101 on $d_{2l+1}d_{2l}d_{2l-1}$.

Let $H_m = |\mathcal{C}'_{\text{FOC}}(m)|$. We can construct the codebook $\mathcal{C}'_{\text{FOC}}(m+1)$ from $\mathcal{C}'_{\text{FOC}}(m)$. Let x_m and y_m denote the number of codewords starting with 1 and 0 respectively. When $m = 1$, no constraint can be applied on the codebook, therefore $H_1 = 2$, and $x_1 = y_1 = 1$. If $m = 2l - 1$, since no restriction is applied on the boundary $d_{2l}d_{2l-1}$, d_{2l} can be either 1 or 0. Thus $H_{2l} = 2H_{2l-1}$ and $x_{2l} = y_{2l} = H_{2l-1}$. If $m = 2l$ is even, since 10 is avoided from the boundary $d_{2l+1}d_{2l}$, $x_{2l+1} = x_{2l}$ and $y_{2l+1} = x_{2l} + y_{2l}$, and $H_{2l+1} = 2x_{2l} + y_{2l} = 1.5H_{2l}$ since $x_{2l} = y_{2l} = 0.5H_{2l}$. Thus the codebook size of this subset is given by $4 \times 3^{\frac{m-2}{2}}$ when m is even, and $2 \times 3^{\frac{m-1}{2}}$ when m is odd. We note that $\lfloor \log_2 H_m \rfloor$ increases with m, and the code rate approaches a limit $\frac{1}{2}\log_2 3 = 0.7925$ when m grows large. This limit is higher than the limit for m-bit FPCs (0.6943), but is lower than that of m-bit FOCs (0.8791). The code rates of $\mathcal{C}_{\text{FOC}}(m)$ and $\mathcal{C}'_{\text{FOC}}(m)$ are compared in Fig. 2.4. From the comparison, if m is small, the difference between the code rates of $\mathcal{C}_{\text{FOC}}(m)$ and $\mathcal{C}'_{\text{FOC}}(m)$ is quite small, and the area overhead of $\mathcal{C}'_{\text{FOC}}(m)$ is close to that of $\mathcal{C}_{\text{FOC}}(m)$.

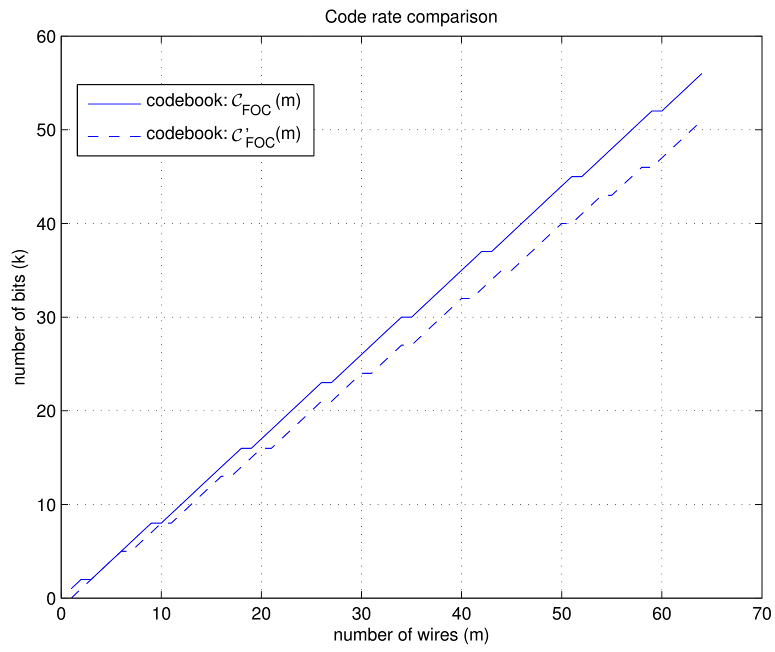


Figure 2.4: The comparison of the code rates of $\mathcal{C}_{\text{FOC}}(m)$ and $\mathcal{C}'_{\text{FOC}}(m)$.

Let $\{h_i\}_{i=1}^{\infty}$ be the following sequence: $h_{2l} = 3h_{2l-2}$ and $h_{2l-1} = h_{2l-2}$ for $l > 1$ with inial conditions $h_1 = 1$ and $h2 = 2$. This sequence has the following property

Lemma 2.13. *$h_{2l} = 1 + \sum_{i=1}^{2l-1} h_i$ for all $l > 0$.*

Proof. $1+\sum_{i=1}^{2l-1} h_i = 2+\sum_{i=2}^{2l-1} h_i = 2+2\sum_{i=1}^{l-1} h_{2i}$. $\{h_{2i}\}_{i=1}^{\infty}$ is a geometric sequence by its definition. Therefore $2 + 2\sum_{i=1}^{l-1} h_{2i} = 2 \times 3^{l-1} = h_{2l}$. □

Then we have the following proposition:

Table 2.1: The constants used in Alg. 1 for different CACs.

CAC	Θ	α_k	β_k	γ_k
OLC	$G_{2\lfloor \frac{m}{2} \rfloor+4}$	G_{2l+2} $k=2l-1$ ∞ $k=2l$	0 $k=2l-1$ G_{2l+2} $k=2l$	f_k
FTC	$F_{2\lfloor \frac{k}{2} \rfloor+1}$	$F_{2\lfloor \frac{k}{2} \rfloor+1}$	$F_{2\lfloor \frac{k}{2} \rfloor+1}$	F_k
FPC	F_{m+1}	F_{k+1}	F_k	F_{m+1} $k=m$ F_k $1 \le k < m$
FOC	$\sum_{i=2\lfloor \frac{m}{2} \rfloor}^{m} h_i$	$\sum_{i=2\lfloor \frac{k}{2} \rfloor}^{k} h_i$	$\sum_{i=2\lfloor \frac{k}{2} \rfloor}^{k} h_i$	h_k

Theorem 2.14. *The numeral system defined by the basis set $\{h_i\}_{i=1}^m$ is complete.*

Proof. If $k > 1$ is odd, then $h_k = h_{k-1} < \sum_{i=1}^{k-1} h_i + 1$. If $k = 2l$ is even, then $h_k = 1 + \sum_{i=1}^{k-1} h_i$ by Lemma 2.13. Thus the theorem follows by Lemma 2.1. $\square$

Lemma 2.15. *$H_m = 1 + \sum_{i=1}^{m} h_i$, for all $m \geq 0$.*

It is easy to prove this lemma by induction, and hence the proof is omitted.

We can encode $\mathcal{C}'_{\mathrm{FOC}}(m)$ with Alg. 1 by choosing

$$\gamma_i = h_i,\ \Theta = \sum_{i=2\lfloor \frac{m}{2} \rfloor}^{m} h_i,\ \alpha_k = \beta_k = \sum_{i=2\lfloor \frac{k}{2} \rfloor}^{k} h_i. \tag{2.6}$$

Similar with what we did in proving the correctness of OLC encoder algorithm, we first need to show that the output of Alg. 1 with the constants specified in (2.6 is a binary string, which is a result of Lemma 2.16.

Lemma 2.16. *For all $k \in [2, m]$, $0 \leq r_k \leq \sum_{i=1}^{k-1} h_i$ in the FOC encoding algorithm.*

Proof. We prove this lemma by induction on k from m to 2. Let us first show $r_m \leq \sum_{i=1}^{m-1} h_i$. When $m = 2l$, if d_{2l} is encoded as 0, then $r_{2l} = v \leq h_{2l}-1 \leq \sum_{i=1}^{2l-1} h_i$

by Lemma 2.13; if d_{2l} is encoded as 1, then $r_{2l} = v - h_{2l} \leq H_m - h_{2l} - 1 = \sum_{i=1}^{2l-1} h_i$ by Lemma 2.15. When $m = 2l + 1$, if d_m is encoded as 0, then $r_{2l+1} = v \leq h_{2l} + h_{2l+1} - 1 = 2h_{2l} - 1 = \sum_{i=1}^{2l} h_i$ by Lemma 2.13. Suppose at step $k + 1$, $r_{k+1} \leq \sum_{i=1}^{k} h_i$. If d_k is encoded as 1, then $r_k = r_{k+1} - h_k \leq \sum_{i=1}^{k} h_i - h_k = \sum_{i=1}^{k-1} h_i$. Therefore we need to consider only the case where d_k is encoded as 0. If $d_k = 0$ and k is even, then $r_k < h_k$ which implies $r_k \leq h_k - 1 = \sum_{i=1}^{k-1} h_i$ by Lemma 2.13. When k is odd, $h_k = h_{k-1}$, therefore if $d_k = 0$, $r_k < h_{k-1} + h_k$ implies $r_k \leq 2h_{k-1} - 1 = \sum_{i=1}^{k-2} h_i + h_{k-1} = \sum_{i=1}^{k-1} h_i$. Thus $r_k \leq \sum_{i=1}^{k-1} h_i$ is proved.

Now we are going to show that $r_k \geq 0$ by induction from m to 2. When $k = m$, if $d_m = 0$, then $r_m = v \geq 0$. If $d_m = 1$, then $r_m = v - h_m \geq h_m - h_m = 0$. Suppose at step $k + 1$, $r_{k+1} \geq 0$. At step k, if $d_k = 0$, then $r_k = r_{k+1} \geq 0$. If $d_k = 1$, then $r_{k+1} \geq \sum_{i=2\lfloor \frac{k}{2} \rfloor}^{k} h_k \geq h_k$, hence $r_k = r_{k+1} - h_k \geq 0$. $\square$

Theorem 2.17. *The output of FOC encoding algorithm satisfies the constraints of $\mathcal{C}'_{\mathrm{FOC}}(m)$.*

Proof. We prove it by induction on k from m to 2. Suppose the previous output $d_m d_{m-1} \cdots d_k$ satisfies the constraints of $\mathcal{C}'_{\mathrm{FOC}}(m)$, which is that $d_{2l+1}d_{2l} \neq 10$. When $k = m$ and $m = 2l$, there is no constraint on d_{2l-1}. If $m = 2l + 1$, there are two possible cases: if $d_{2l+1} = 0$, there is also no constraint on d_{2l}; if $d_{2l+1} = 1$, then $v \geq h_{2l} + h_{2l+1}$, and $r_m = v - h_{2l+1} \geq h_{2l}$, implying that $d_{m-1} = 1$. Therefore, the bit pattern 10 is avoided from $d_m d_{m-1}$ when m is odd. If $k = 2l + 1$, and d_{2l+1} is encoded as 1, then $r_{2l+1} = r_{2l+2} - h_{2l+1} \geq \alpha_{2l+1} - h_{2l+1} = \gamma_{2l} = \alpha_{2l}$. Thus $d_{k-1} = d_{2l}$ has to be encoded as 1, and the bit pattern 10 is avoided at $d_{2l+1}d_{2l}$. Since there is no constraint on $d_{2l}d_{2l-1}$, the output is a codeword in $\mathcal{C}'_{\mathrm{FOC}}(m)$. $\square$

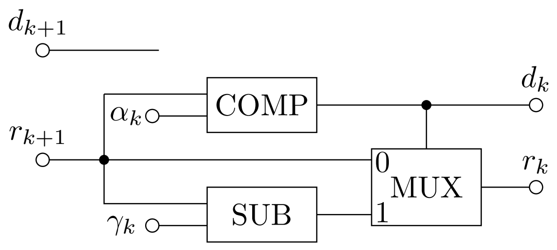


Figure 2.5: The FOC CODEC processing element. Although neither output depends on d_{k+1}, it is shown here to be consistent with Fig. 2.2

The decoding algorithm is given by the formula $v = \sum_{i=1}^{m} d_i h_i$. Since different integers generate different codewords, the FOC encoding algorithm gives a bijection from the integers in $[0, H_m - 1]$ to the codebook $\mathcal{C}'_{\text{FOC}}(m)$.

As a special case of Alg. 1, the same CODEC structure in Fig. 2.1 can be used as the FOC CODEC after replacing the constants specified in (2.6). Because $\alpha_k = \beta_k$, as shown in Fig.2.5, in each processing element one comparator suffices and the input d_{k+1} is inconsequential. The quadratic complexity of our FOC CODEC follows by the complexity of the general CAC CODEC. The throughput of this CODEC can be increased via pipelining if desired.

2.7 Implementation Results and Discussion

To quantify the delay as well as area and power overheads introduced by our CAC CODECs, we implement our OLC, FPC, and FOC CODECs based on numeral systems without pipelining. Our CODECs are simulated on Modelsim [34] and synthesized by Cadence Encounter RTL Compiler [35] with an OKSU FreePDK 45 nm process [36]. The figures for power consumption of our CODECs are derived by the power analysis tool in Encounter. To measure the CODEC power consumption, we assume the clock rate is 100 MHz and set the input switching rate to be 0.5 for all CODECs. The clock rate is selected merely for the purpose of demonstrations, and are inconsequential to our conclusions below. In practice, the clock rate should be determined by bus delays as well as CODEC delays. Our CODEC delays are not likely to be the bottleneck of achievable clock rates for two reasons. First, the delays of our CODECs can be easily improved by pipelining or partitioning the bus. Second, since the technology trend indicates increasing bus delays and decreasing gate delays, the bus delays will be more likely to be the bottleneck.

The implementation results are shown in Figs. 2.6–2.8. Fig. 2.6 shows the delay introduced by our CODECs, while Figs. 2.7 show CODEC complexities in terms equivalent gate count. The result of area consumption includes the cell area only. Fig. 2.8 shows the power consumption of our CODECs, including the leakage power and the estimated internal and switching power. Our simulation results show that the delay and the area complexity as well as the power consumption of our CODECs increase quadratically with the bus width. The gate counts and delay of our CODECs can be estimated with the following quadratic functions,

$$GE_{OLC}(m) = 13.44m^2 - 64.53 + 200.1, \tag{2.7}$$

$$GE_{FPC}(m) = 11.14m^2 - 45.25m + 131.6, \tag{2.8}$$

$$GE_{FOC}(m) = 7.648m^2 - 40.72m + 112, \tag{2.9}$$

$$D_{OLC}(m) = 0.0550m^2 + 0.2008m - 0.0485\,\text{ns}, \tag{2.10}$$

$$D_{FPC}(m) = 0.0358m^2 + 0.8152m - 3.096\,\text{ns}, \tag{2.11}$$

$$D_{FOC}(m) = 0.0250m^2 + 0.1476m - 0.0412\,\text{ns}. \tag{2.12}$$

We observe that our implementation results of FPC CODECs differ from those in [6] (cf. [6, Fig. 7]), possibly due to different technology and design tools. Despite this difference, our CODECs also have quadratic complexities, which is consistent with the conclusion in [5, 6]. We also remark that the quadratic functions can be used

Fig. 2.9 compares the area consumptions of the CAC CODECs based on look-up table (LUT) and those based on numeral systems. Although they have the same codebook mapping, the synthesis results are different. From Fig. 2.9, we can see that when the bus input width is small (less than 8 for OLC and FPC, and less than 9 for FOC), the LUT-based CODECs take up smaller area than our CODECs. However, when the bus input width increases, the area consumption of the LUT-based CODEC increases exponentially and they become infeasible. In contrast, the area consumption of our numeral system based CODECs increases quadratically with the bus input width, and therefore our CODECs remain practical even when the LUT-based CODECs become infeasible.

Based on our implementation results above, our CODECs are applicable to the buses with severe crosstalk delay and moderate bus width. Note that the delay as well as area and power overheads of our CODECs increase quadratically with the bus width. Since CACs trade additional area and power for reduced delay, the delays of CAC CODECs need to be kept small so as not to offset the delay improvement by CAC encoding. Hence the number of wires cannot be too large, say $m < m_0$. Furthermore, the area and power overheads of our CODECs also increase with the bus width; when the bus width is too large, it may be better to divide the bus into sub-buses with shielding or duplication wires between sub-buses. Moreover, as observed above, when the bus width is very small, LUT-based CODECs have smaller area and power overheads. Finally, the technology trends indicate that as the CMOS technology scales, the gate delay decreases while the interconnect delay increases. Therefore, the delay of our CODECs will become relatively small as opposed to crosstalk delay, and will be more relevant in the future.

To find and achieve the appropriate balance between delay, area, power, and latency, other techniques, such as partial coding and pipelining, can be used together with our efficient CODECs. Although the quadratic functions in (2.7)–(2.12) are approximate and may vary with technology, they can be used to find the proper tradeoff with m being the key parameter. Both partial coding and pipelining are useful tools to achieve the appropriate balance. First, our CODECs have regular circuitry and can be pipelined to have a smaller delay (and hence a higher clock rate) at the expense of a larger area and longer latency. A bus with a high clock rate (and thus throughput) and long latency may be useful in certain scenarios. However, pipelining of CODECs transforms the bus into a multi-cycle bus; thus, whether CACs are worthwhile should be evaluated by taking this into account. Second, our CODECs can be integrated with the partial coding technologies (cf. [6, Fig. 9]). Since our CODECs have quadratic complexity, the delay is reduced to approximately $\frac{1}{n^2}$ of the original delay and the total area is reduced to roughly $\frac{1}{n}$, by breaking the bus into n sub-buses. The price of this approach is that it needs extra shielding or duplication wires inserted between the sub-buses. In summary, when integrated with other techniques, our efficient CODECs help to find the balance between the delay and area/power overheads so as to minimize the overheads while satisfying the speed requirement for the bus when integrated with other techniques.

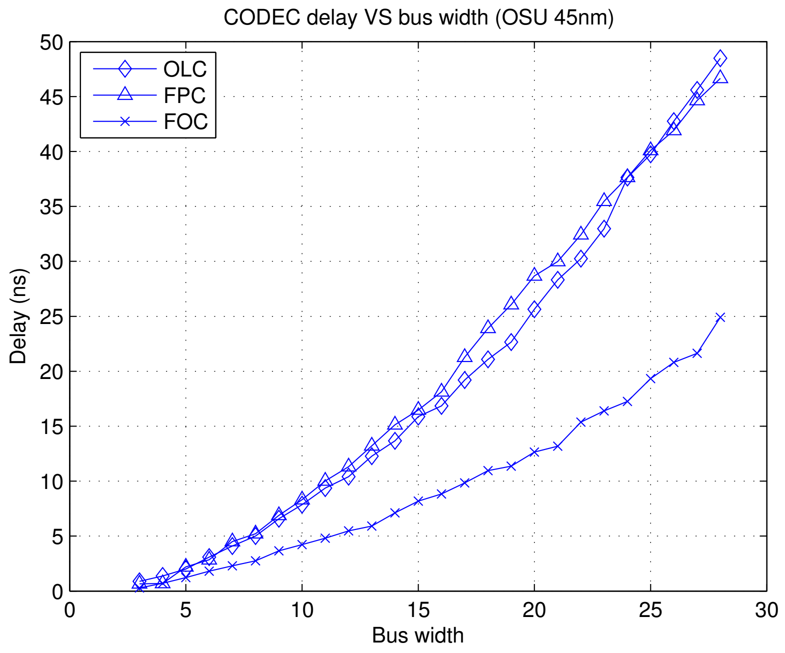


Figure 2.6: Delay of numeral system based CAC CODECs

2.8 Summary

In this chapter, we establish a framework for the CAC CODEC design based on numeral systems, and devise efficient CODECs for OLCs, FPCs, and FOCs by choosing appropriate numeral systems and constants. The results are summarized in Tab. 2.1. Implementation results show that our CODECs all have area and delay that increase quadratically with the bus width. Used together with partial coding, our efficient CODECs help make CACs a viable option in combating crosstalk delay, which is a bottleneck in deep sub-micron system-on-chip designs.

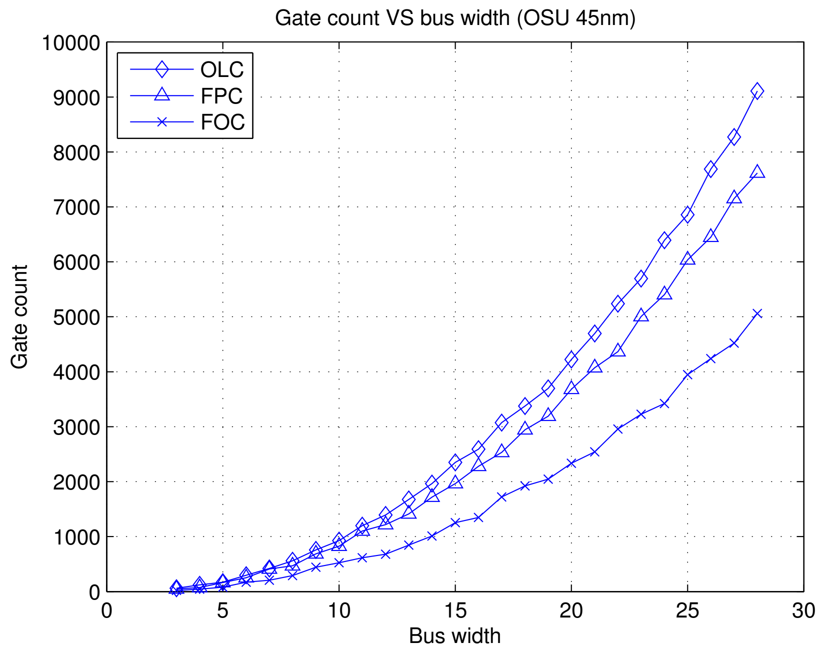


Figure 2.7: Gate count of numeral system based CAC CODECs

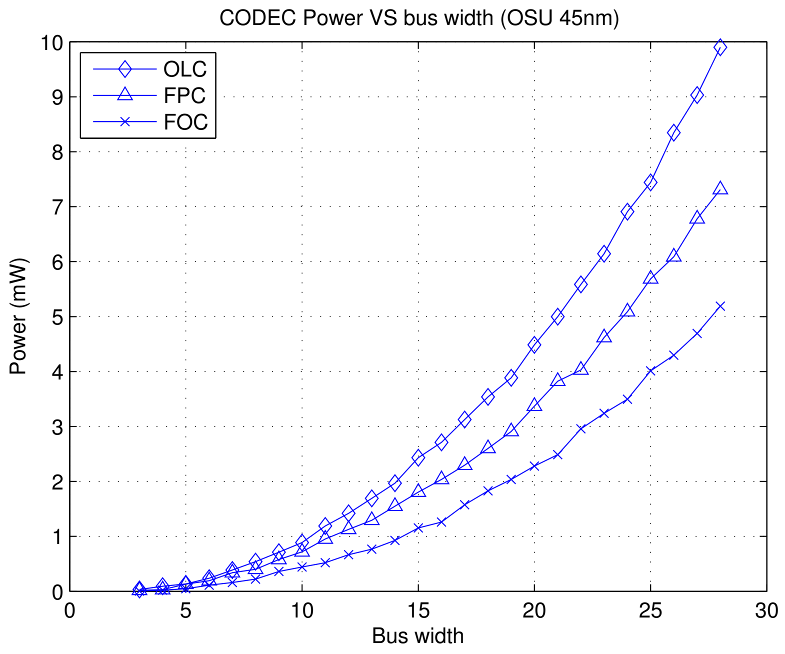


Figure 2.8: Power consumption of numeral system based CAC CODECs

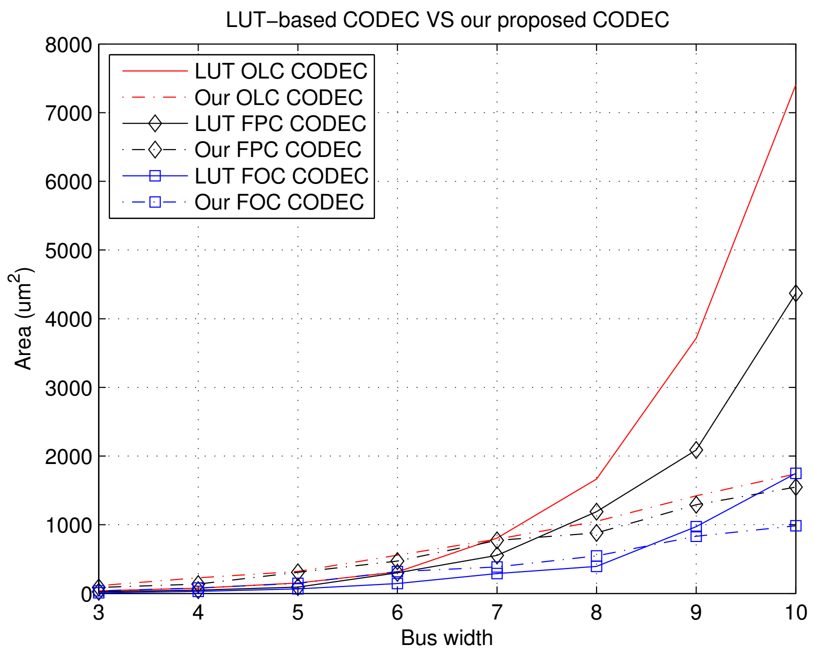


Figure 2.9: Comparison between the LUT-based CAC CODECs and our CAC CODECs with the same codebook mapping.

Chapter 3

Two-Dimensional Crosstalk Avoidance Codes

We have introduced in Chapter 2 that different crosstalk avoidance codes (CACs) are proposed to alleviate the problem of the increasing crosstalk delay on the interconnect bus in the deep sub-micron technologies. These CACs, such as *One lambda codes* (OLCs) [4], *forbidden transition codes* (FTCs) [2], *forbidden patterns codes* (FPCs) [3], and *forbidden overlap codes* (FOCs) [31], are all one dimensional since both encoding and decoding are restricted in the **spatial** domain.

In this chapter, we propose two-dimensional crosstalk avoidance codes (TDCACs). In TDCACs, encoding and decoding occur in **both spatial and temporal** domains. The main advantage of TDCACs is **higher code rates**, which implies **less area and power overhead** to reduce crosstalk delay. The rates of crosstalk avoidance codes are due to the restriction on the transitions over the wires to reduce

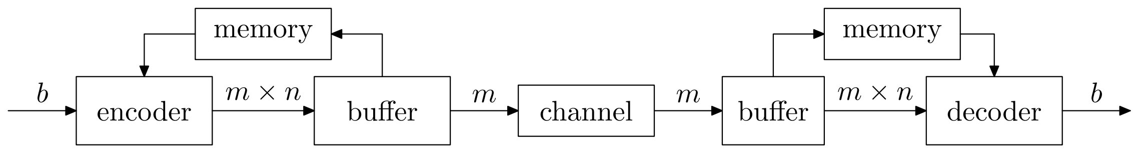


Figure 3.1: Channel model of our TDCAC.

crosstalk delay. Since TDCACs have an additional temporal dimension, the restriction on the transitions does not have so severe an impact on their code rates as traditional one dimensional CACs. Indeed, our results (details shown below) show that TDCACs result in higher code rates than their one-dimensional counterparts. However, two potential drawbacks for TDCACs are encoding/decoding complexity and latency. As later shown in an example, the encoding and decoding complexities are low for properly designed TDCACs. Furthermore, partial coding can be used to reduce encoding and decoding complexities. As regard to latency, it is important to emphasize that in many systems the latency of global buses is much less important than throughput. Even one-dimensional crosstalk avoidance codes incur latency penalty up to two clock cycles (cf. [28, Figs. 7.2 and 7.4]). Longer latency may be needed when the decoding cannot begin until the **whole** code matrices are received. However, when the TDCACs are memoryless and systematic, the decoding may start after only **part** of the code matrices are received. Furthermore, our results show that even TDCACs with a small n can achieve higher code rates without incurring severe latency penalty. Thus, properly designed TDCACs represent a promising solution to the crosstalk delay problem for global buses.

This chapter aims to establish a theoretical framework for TDCACs. In this sense, the results in this chapter parallel those in [2]. In Sec. 3.1, we first consider TDCACs with memory. Specifically, we first determine the maximum code rates for TDCACs with memory for $n \leq 15$; we also conjecture the maximum code rates for TDCACs with memory for $n > 15$. In Sec. 3.2, we present our graph model for TDCACs without memory. We also show an example of 3×3 TDCACs including its encoding and decoding circuits in Sec. 3.3. In Sec. 3.4, the code rates of some TDCACs are compared with one-dimensional CAC.

3.1 Two Dimensional Codes with Memory

3.1.1 System Model

Let us consider TDCACs that consist of $m \times n$ code matrices, where m is the width of the bus and n is the number of consecutive clock cycles. The channel model of our TDCACs is shown in Figure 3.1. First, b data bits are encoded into an $m \times n$ code matrix, and the code matrix is placed in the encoder's buffer, waiting to be transmitted. The encoding may also depend on the previous encoded code matrix, which is saved in the memory. Each $m \times n$ code matrix is then transmitted over the m-bit bus in n clock cycles, and the received vectors are saved in the decoder's buffer. Finally, the decoder recovers the b data bits using each received code matrix and possibly the previous code matrix saved in the memory. In the worst case scenario, both the encoder's and decoder's buffers need to hold one code matrix, and hence the latency is at most $2n + 1$ clock cycles. However, if the decoder can start when only part of the code matrix is received, the latency is shorter and the buffer is smaller. Although the latency of the bus increases, the throughput of the bus is the same as that for the bus using one-dimensional CAC. The rates of such TDCACs are defined to be $\frac{b}{mn}$, which again measures the efficiency of the codes.

The TDCACs shown in Figure 3.1 are called codes *with memory* since their encoding and decoding operations depend on the code matrices stored in the memory. When the encoding and decoding operations of TDCACs do not rely on the code matrices stored in the memory, such TDCACs are referred to as *memoryless* TDCACs.

3.1.2 Minimization Problem to Determine Codebook Size

The goal of our $m \times n$ TDCACs is to reduce the worst-case delay in (2.1) from $(1+4\lambda)\tau_0$ to $(1+2\lambda)\tau_0$. Based on (2.1), this translates into a crosstalk avoidance condition on the transitions of all the wires. The i-th column vector in the code matrix is the vector transmitted on the bus at the i-th clock cycle. The crosstalk avoidance condition is then represented by a $2^m \times 2^m$ transfer matrix $\mathbf{M}(m)$. The entry M_{ij} in $\mathbf{M}(m)$ is 1 if the transition from the vector representing i to the vector representing j satisfies the crosstalk avoidance condition, and is 0 if the transition does not satisfy the condition. Suppose the last column vector of a code matrix is $\mathbf{v} = (v_0, v_1, \cdots, v_{m-1})$, representing an integer $v = \sum_{i=0}^{m-1} 2^i v_i$. The number of the valid code matrices that can transition from such a code matrix ending with $\mathbf{v}$ is given by $K_n(\mathbf{v}, m) = \mathbf{e}_v^t \mathbf{M}^n(m) \mathbf{1}_m$, where $\mathbf{1}_m$ is a 2^m-dimensional column vector with all entries equal to one, and $\mathbf{e}_v$ is a 2^m-dimensional column vector with the entry at position v equal to one and all the other entries are zero.

As in [2, 28], we assume that all vectors can appear in all code matrices and no further restriction (such as pruning) is put on the codebook, the maximum number of code matrices allowed in the codebook is limited by the minimum number of the possible code matrices transitioning from an arbitrary vector $\mathbf{v}$. That is, the maximum codebook size, denoted by $|\mathcal{C}(m,n)|$, is given by

$$|\mathcal{C}(m,n)| = \min_{0 \le v \le 2^m - 1} K_n(\mathbf{v}, m). \tag{3.1}$$

It is actually quite difficult to solve the minimization problem in (3.1) when $\mathbf{M}(m)$ reflects all valid transitions. Thus, we focus on the FT condition as the crosstalk avoidance condition henceforth in this section. In this case, $\mathbf{M}(m)$ can be constructed from $\mathbf{M}(m-1)$. Suppose the vector on the bus before transition is denoted by $\mathbf{v}^a$ and the vector after transition is $\mathbf{v}^b$. If $v^a_{m-1} = v^b_{m-1}$, the transition $\mathbf{v}^a \to \mathbf{v}^b$ is valid if and only if the transition of the remaining $m-1$ bits are valid. If $v^a_{m-1} \ne v^b_{m-1}$, the transition $\mathbf{v}^a \to \mathbf{v}^b$ is valid if and only if the other $m-1$ bits make a valid transition and the transition of $v^a_{m-2} \to v^b_{m-2}$ can not transition in an opposite direction with v_{m-1}. Since the valid transitions of $(m-1)$-bit vectors are given by $\mathbf{M}(m-1)$,

$$\mathbf{M}(m) = \begin{bmatrix} \mathbf{M}(m-1)\,\mathbf{M}(m-1) \\ \mathbf{M}(m-1)\,\mathbf{M}(m-1) \end{bmatrix} \odot \begin{bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} \\ \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \end{bmatrix},$$

where $\odot$ means an entry-wise matrix multiplication, and $\mathbf{1}$ and $\mathbf{0}$ are $2^{m-2} \times 2^{m-2}$ matrices with all entries equal to 1 and 0, respectively. The initial value $\mathbf{M}(1)$ is a 2×2 matrix with all entries equal to 1.

3.1.3 Reformulation of the Minimization Problem

We first reformulate the minimization problem in (3.1) for $n = 2$ when the FT condition is employed to avoid crosstalk delay. For a code matrix ending with an m-bit vector $\mathbf{v}$, the set of the code matrices it can transition to is divided into 4 sets according to the activity of the first bit in the two clock cycles: (unchanged, unchanged), (unchanged, changed), (changed, unchanged), and (changed, changed). We keep track of the numbers of the code matrices in these 4 sets by a vector $\mathbf{y}_2(\mathbf{v}, m) = [y_{uu}(m)\, y_{uc}(m)\, y_{cu}(m)\, y_{cc}(m)]^T$ respectively. These numbers are related to $\mathbf{y}_2(\mathbf{v}, m-1)$. By looking at the transition pattern of the first two bits, we can see that if $v_{m-1} \neq v_{m-2}$, $\mathbf{y}_2(\mathbf{v}, m) = \mathbf{D}_2\mathbf{y}_2(\mathbf{v}, m-1)$, and if $v_{m-1} = v_{m-2}$, $\mathbf{y}_2(\mathbf{v}, m) = \mathbf{S}_2\mathbf{y}_2(\mathbf{v}, m-1)$, where

$$\mathbf{D}_2 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix} \text{ and } \mathbf{S}_2 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}.$$

The total number of the code matrices that can transition from a matrix ending with vector $\mathbf{v}$ $K_2(\mathbf{v}, m)$ is simply the sum of the numbers of code matrices in these four cases, and is given by $K_2(\mathbf{v}, m) = [1\,1\,1\,1]\mathbf{y}_2(\mathbf{v}, m)$. Thus $K_2(\mathbf{v}, m) = \mathbf{1}_2^T\mathbf{P}_2^{(m-1)}\mathbf{P}_2^{(m-2)} \cdots \mathbf{P}_2^{(0)}\mathbf{y}_2(0)$, where $\mathbf{P}_2^{(0)} = \mathbf{D}_2$ and $\mathbf{P}_2^{(i)}$ is equal to $\mathbf{D}_2$ if $v_i \neq v_{i-1}$ and $\mathbf{S}_2$ if $v_i = v_{i-1}$. The initial value $\mathbf{y}_2(0) = [1\,0\,0\,0]^T$.

We now extend our reformulation of the minimization problem when $n = 2$ to the general case. The key of the extension is to generalize $\mathbf{D}_2$ and $\mathbf{S}_2$. Suppose the previous $m \times n$ code matrix ends with an m-dimensional vector $\mathbf{v} = \{v_0, v_1, \cdots, v_{m-1}\}$.

We still keep track of the number of the code matrices which have the same transition patterns of new row in n clock cycles, and these numbers form a 2^n-dimensional vector $\mathbf{y}_n(\mathbf{v}, m)$. Similar as $n = 2$, we can construct two $2^n \times 2^n$ matrices $\mathbf{D}_n$ and $\mathbf{S}_n$ thus when $\mathbf{v}_{m-1} \neq \mathbf{v}_{m-2}$, $\mathbf{y}_n(\mathbf{v}, m) = \mathbf{D}_n \mathbf{y}_n(\mathbf{v}, m-1)$, and when $\mathbf{v}_{m-1} = \mathbf{v}_{m-2}$, $\mathbf{y}_n(\mathbf{v}, m) = \mathbf{S}_n \mathbf{y}_n(\mathbf{v}, m-1)$. The matrix $\mathbf{D}_n$ and $\mathbf{S}_n$ can both be constructed recursively from $\mathbf{D}_{n-1}$ and $\mathbf{S}_{n-1}$. Suppose we are going to add a new row to each $m \times n$ code matrix to get an $(m+1) \times n$ TDCAC and the previous vector $\mathbf{v}_m \neq \mathbf{v}_{m-1}$. Consider the two first entries of the new row and the row adjacent to the new one. If these two entries have different transition pattern, they are the same now, and the rest of the transition patterns in these two rows can be described by $\mathbf{S}_{n-1}$. If these two entries stays the same, they are still different, and the rest of the transition patterns can be described by $\mathbf{D}_{n-1}$. If these two entries are both changed, the FT condition is violated, and the rest of the transition patterns can be described by an $2^{n-1} \times 2^{n-1}$ zero matrix. Thus we have $\mathbf{D}_n = \begin{bmatrix} \mathbf{D}_{n-1} & \mathbf{S}_{n-1} \\ \mathbf{S}_{n-1} & \mathbf{0} \end{bmatrix}$. By a similar argument, we have $\mathbf{S}_n = \begin{bmatrix} \mathbf{S}_{n-1} & \mathbf{D}_{n-1} \\ \mathbf{D}_{n-1} & \mathbf{S}_{n-1} \end{bmatrix}$. The initial value is $\mathbf{D}_0 = \mathbf{S}_0 = 1$.

Now consider a code matrix ending with a vector $\mathbf{v} = (v_0, v_1, \cdots, v_{m-1})^T$, and we will determine $K_n(\mathbf{v}, m)$, the number of the code matrices that can transition from $\mathbf{v}$. Consider the first k bits of the vector $\mathbf{v}$. The set of the $k \times n$ code matrices that can transition from the first k bits of $\mathbf{v}$ can be partitioned into 2^n sets according to the activity of the last bit in this vector, v_{k-1}. The numbers of the code matrices in these sets form a vector $\mathbf{y}_n(\mathbf{v}, k)$. From the analysis and the definition of $\mathbf{D}_n$ and

$\mathbf{S}_n$, we know that $\mathbf{y}_n(\mathbf{v}, k)$ can be calculated from $\mathbf{y}_n(\mathbf{v}, k-1)$:

$$\mathbf{y}_n(\mathbf{v}, k) = \begin{cases} \mathbf{D}_n \mathbf{y}_n(\mathbf{v}, k-1), & v_k \neq v_{k-1}, \\ \mathbf{S}_n \mathbf{y}_n(\mathbf{v}, k-1), & v_k = v_{k-1}. \end{cases}$$

Thus $K_n(\mathbf{v}, m)$ is given by

$$\begin{aligned} K_n(\mathbf{v}, m) &= \mathbf{1}^T \mathbf{y}_n(\mathbf{v}, m) \\ &= \mathbf{1}^T \mathbf{P}_n^{(m-1)} \mathbf{P}_n^{(m-2)} \cdots \mathbf{P}_n^{(0)} \mathbf{y}_n(0), \end{aligned}$$

where $\mathbf{y}_n(0)$ is a 2^n-dimensional column vector with the first entry 1 and all the other entries 0, $\mathbf{P}_n^{(0)} = \mathbf{D}_n$, and $\mathbf{P}_n^{(i)}$ is a matrix defined in the following way,

$$\mathbf{P}_n^{(i)} = \begin{cases} \mathbf{D}_n, & v_i \neq v_{i-1}, \\ \mathbf{S}_n, & v_i = v_{i-1}, \end{cases} \quad \text{for } 1 \leq i \leq m-1.$$

Since one-dimensional CAC are special cases of TDCACs with $n = 1$, many results for one-dimensional CAC (such as those in [28]) can be readily obtained from our results for TDCACs. For example, the aforementioned result that the degree of class 1 codewords is Fibonacci sequence [28] is a special case with $n = 1$ of our results. The matrix $\mathbf{D}_1$ is given by $\mathbf{D}_1 = \left[\begin{smallmatrix} 1 & 1 \\ 1 & 0 \end{smallmatrix}\right]$. The matrix $\mathbf{D}_1$ satisfies $\mathbf{D}_1^2 = \mathbf{I} + \mathbf{D}_1$. Thus $K_1(m)$ admits a recursive relation: $K_1(m) = K_1(m-1) + K_1(m-2)$, which is exactly the recurrence relation of Fibonacci number. Our results also lead to $K_1(1) = 2$ and $K_1(2) = 3$, same as those in [28].

Another example is the result in [28] that the degree of any vector is equal to $d_{\{n_1, n_2, \cdots, n_c\}} = \prod_{i=1}^{c} d_{n_i}$, where $d_{n_i} = F(n_i + 2)$, and n_i is the length of the section with alternating 0 and 1. Since all the adjacent bits in the section with alternating 0 and 1 are different, the expression of $K_1(\mathbf{v}, m)$ is given by

$$K_1(\mathbf{v}, m) = \mathbf{1}_1^T \mathbf{D}_1^{n_1-1} \mathbf{S}_1 \mathbf{D}_1^{n_2-1} \mathbf{S}_1 \cdots \mathbf{D}_1^{n_c} \mathbf{y}_1(0). \tag{3.2}$$

Plugging $\mathbf{S}_1 = \mathbf{D}_1 \mathbf{y}_1(0) \mathbf{1}_1^T$ in (3.2), we get

$$\begin{aligned} K_1(\mathbf{v}, m) &= \mathbf{1}_1^T \mathbf{D}_1^{n_1} \mathbf{y}_1(0) \mathbf{1}_1^T \mathbf{D}_1^{n_2} \mathbf{y}_1(0) \cdots \mathbf{1}_1^T \mathbf{D}_1^{n_c} \mathbf{y}_1(0) \\ &= \prod_{i=1}^{c} K_1(n_i) = \prod_{i=1}^{c} F(n_i + 2), \end{aligned}$$

which is exactly the same result as in [28].

3.1.4 The Solution to the Minimization Problem

In this section, we will derive the solution for the minimization problem in (3.1) for n up to 15. Before presenting our result, we need to establish two lemmas first. For the simplicity of our expression, we call a matrix or a vector *non-negative* if all of its elements are non-negative. For example, from the construction of the $\mathbf{S}_n$ and $\mathbf{D}_n$, we can see that both $\mathbf{S}_n$ and $\mathbf{D}_n$ are binary and so they are non-negative.

Lemma 3.1. *$\mathbf{S}_n^2 - \mathbf{D}_n^2$ is a non-negative matrix.*

Proof. We are going to show this lemma by induction. When $n = 0$, $\mathbf{S}_0^2 - \mathbf{D}_0^2 = 0$ is non-negative. Suppose the lemma holds for n up to k. When $n = k+1$,

$$\begin{aligned}
&\mathbf{S}_{k+1}^2 - \mathbf{D}_{k+1}^2 \\
&= \begin{bmatrix} \mathbf{S}_k^2 + \mathbf{D}_k^2 & \mathbf{S}_k\mathbf{D}_k + \mathbf{D}_k\mathbf{S}_k \\ \mathbf{S}_k\mathbf{D}_k + \mathbf{D}_k\mathbf{S}_k & \mathbf{S}_k^2 + \mathbf{D}_k^2 \end{bmatrix} - \begin{bmatrix} \mathbf{S}_k^2 + \mathbf{D}_k^2 & \mathbf{D}_k\mathbf{S}_k \\ \mathbf{S}_k\mathbf{D}_k & \mathbf{S}_k^2 \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{0}_k & \mathbf{S}_k\mathbf{D}_k \\ \mathbf{D}_k\mathbf{S}_k & \mathbf{D}_k^2 \end{bmatrix},
\end{aligned}$$

which is also a non-negative matrix, because both $\mathbf{S}_k$ and $\mathbf{D}_k$ are non-negative and so are their products. □

Lemma 3.2. *When $n \leq 15$, the vector $\mathbf{1}_n^T\mathbf{D}_n^k(\mathbf{S}_n - \mathbf{D}_n)\mathbf{D}_n$ is a vector with all entries non-negative for all $k \geq 0$.*

Proof. We can check that the matrix $\mathbf{D}_n^7(\mathbf{S}_n - \mathbf{D}_n)\mathbf{D}_n$ is a non-negative one for all $n \leq 15$. Therefore, when $k > 7$, $\mathbf{1}_n^T\mathbf{D}_n^k(\mathbf{S}_n - \mathbf{D}_n)\mathbf{D}_n = \mathbf{1}_n^T\mathbf{D}_n^{k-7} \cdot \mathbf{D}_n^7(\mathbf{S}_n - \mathbf{D}_n)\mathbf{D}_n$ is a product between a non-negative vector and a non-negative matrix, and the result is a non-negative vector. When $k < 7$, the lemma can be verified by inspection on each n and each k. □

Theorem 3.3. *When $n \leq 15$, given an positive integer m, the function $K_n(\mathbf{v}, m)$ is minimized by a vector with alternating zero and one, i.e., $\mathbf{P}_n^{(i)} = \mathbf{D}_n$ for $1 \leq i \leq m-1$.*

Proof. Given an arbitrary m-dimensional vector $\mathbf{v}$, we first check if it consists of the pattern $v_{i-1}v_iv_{i+1} = 000$ or 111, for $1 \leq i \leq m-2$. If such pattern exists, we flip v_i, resulting in a new vector $\mathbf{v}'$. Therefore, in the expression of $K_n(\mathbf{v}, m)$,

$\mathbf{P}_n^{(i)} = \mathbf{P}_n^{(i+1)} = \mathbf{S}_n$, and in that of $K_n(\mathbf{v}', m)$, $\mathbf{P}_n^{(i)} = \mathbf{P}_n^{(i+1)} = \mathbf{D}_n$. By Lemma 3.1, $K_n(\mathbf{v}', m) < K_n(\mathbf{v}, m)$. After we remove all such patterns, we get a new vector $\mathbf{u}$, and $K_n(\mathbf{u}, m) < K_n(\mathbf{v}, m)$.

After removing all such patterns, we check if the new vector $\mathbf{u}$ consists of pattern $u_{i-1} = u_i$ from $i = m-1$ to 1. If such pattern exists, we flip all the bits u_j with $j < i$, resulting in a new vector $\mathbf{u}'$. In the expression of $K_n(\mathbf{u}, m)$, $\mathbf{P}_n^{(m-1)} = \mathbf{P}_n^{(m-2)} = \cdots = \mathbf{P}_n^{(i+1)} = \mathbf{D}_n$, $\mathbf{P}_n^{(i)} = \mathbf{S}_n$, and since there are no three consecutive bits identical, $\mathbf{P}_n^{(i-1)} = \mathbf{D}_n$. In the expression of $K_n(\mathbf{u}', m)$, we have $\mathbf{P}_n^{(m-1)} = \mathbf{P}_n^{(m-2)} = \cdots = \mathbf{P}_n^{(i-1)} = \mathbf{D}_n$. Therefore

$$K_n(\mathbf{u}, m) - K_n(\mathbf{u}', m) = \mathbf{1}_n^T \mathbf{D}_n^{m-1-i} (\mathbf{S}_n - \mathbf{D}_n) \mathbf{D}_n \prod_{j=i-2}^{1} \mathbf{P}_n^{(j)} \mathbf{1}_n \geq 0,$$

since $\mathbf{1}_n^T \mathbf{D}_n^{m-1-i} (\mathbf{S}_n - \mathbf{D}_n) \mathbf{D}_n$ is a non-negative vector by Lemma 3.2, and it is also obvious that $\prod_{j=i-2}^{1} \mathbf{P}_n^{(j)} \mathbf{1}_n$ is non-negative. After removing all such patterns, we get a vector $\mathbf{c}$ with no two consecutive bits identical, i.e., a vector with alternating zero and one. Thus we have shown that $K_n(\mathbf{c}, m) \leq K_n(\mathbf{v}, m)$, where $\mathbf{v}$ is an arbitrary m-dimensional binary vector. □

We conjecture that the Theorem 3.3 can be generalized to arbitrary positive integer n, and we restate it in the following conjecture:

Conjecture 3.4. *Given m, the function $K_n(\mathbf{v}, m)$ is minimized by a vector with alternating zero and one, i.e., $\mathbf{P}_n^{(i)} = \mathbf{D}_n$ for $1 \leq i \leq m-1$.*

The proof of this conjecture relies on generalizing the Lemma 3.2 to arbitrary positive integer n. Now Lemma 3.2 is actually shown by direct computation. When

$n > 15$, $\mathbf{S}_n$ and $\mathbf{D}_n$ are so big that direct computation is impractical. The structure of $\mathbf{S}_n$ and $\mathbf{D}_n$ need to be explored to generalize Lemma 3.2 to arbitrary n and to prove Conjecture 3.4 in the end.

3.2 Memoryless Two Dimensional Codes

The TDCACs we discussed above are with memory, i.e., the encoding and decoding of a code matrix depend on its previous code matrices. The existence of memory increases the complexity of the encoder and decoder, and thus in this section we will consider memoryless TDCACs. As in the case $n = 1$, memoryless TDCACs should consist of a set of code matrices so that the transition between any two code matrices in this set has a crosstalk delay at most $(1 + 2\lambda)\tau_0$. Let us denote the first and last column vectors in an $m \times n$ code matrix as $\mathbf{v}_0$ and $\mathbf{v}_{n-1}$, respectively. We will now establish a graph model for memoryless TDCACs. Let V_1 be a set containing 2^m nodes, and each node denotes a possible vector $\mathbf{v}_0$. Similar, V_2 is a set representing all possible vectors $\mathbf{v}_{n-1}$. If the transition from $\mathbf{v}_{n-1}$ to $\mathbf{v}_0$ is valid, there is an edge connecting the nodes $\mathbf{v}_0$ in V_1 and $\mathbf{v}_{n-1}$ in V_2, and the weight of the edge is set to the number of all valid code matrices starting with $\mathbf{v}_0$ and ending with $\mathbf{v}_{n-1}$, which is given by $\mathbf{e}_{v_0}\mathbf{M}^{n-1}(m)\mathbf{e}_{v_{n-1}}$. When the transition from $\mathbf{v}_{n-1}$ to $\mathbf{v}_0$ is invalid, there is no edge between these two nodes. Thus $B = (V_1, V_2, E)$ is a bipartite graph, where E is the set of edges. In graph theory, a complete bipartite graph is referred to as *biclique* [37]. The problem of finding the largest codebook of memoryless codebook is then equivalent with the problem of finding the biclique with maximum weight in B.

Although it is hard to find the optimal memoryless TDCACs, we can design suboptimal memoryless TDCACs by ensuring the first and last vectors of the code matrices to satisfy the FP condition. The transition between the vectors in a code matrix can satisfy either the FT condition or a complete valid transition condition [26], which forbids the transitions $010 \leftrightarrow 001$, $010 \leftrightarrow 100$, $101 \leftrightarrow 110$, $101 \leftrightarrow 011$, and $010 \leftrightarrow 101$.

If the transitions between the vectors in a code matrix satisfy the FT condition or the complete valid transition condition, we can use transfer matrix to calculate the size of the codebook of memoryless TDCACs. Let $\mathbf{M}(m)$ denote the transfer matrix of the crosstalk avoidance condition, and a $2^m \times 1$ vector $\mathbf{x}$ denote the FPC codewords, where $\mathbf{x}_i$ is 1 when there is no 010 and 101 pattern exist in the binary representation of i and otherwise 0. Then the size of the codebook of $m \times n$ memoryless TDCAC is

$$|\mathcal{C}| = \mathbf{x}^T \mathbf{M}^{n-1}(m)\mathbf{x}.$$

3.3 Example: 3×3 TDCAC

From our simulation, the highest code rate for the 3×3 memoryless matrix code is given by $\frac{7}{9}$ if the methodology presented in Sec. 3.2 is used, so we can encode seven data bits to one code matrix. The seven data bits, denoted as b_0, b_1, b_2, $\cdots$, b_6, are encoded to three column vectors $[d_0\, d_1\, d_2]^T$, $[d_3\, d_4\, d_5]^T$, and $[d_6\, d_7\, d_8]^T$ as follows. First, divide these seven bits into three groups: $\{b_0, b_1\}$, $\{b_2, b_3, b_4\}$, and $\{b_5, b_6\}$, and encode them to the three column vectors respectively. The encoding of the middle column is simply $[d_3\, d_4\, d_5] = [b_2\, b_3\, b_4]$, and the first and last columns are encoded so

Table 3.1: Encoding rule for a 3×3 CAC. "$\times$" means don't care, which can be used to simplify the encoding circuit. The code rate is $\frac{7}{9}$.

$b_0 b_1$ or $b_5 b_6$	$d_0\, d_1\, d_2$ or $d_6\, d_7\, d_8$		
	$b_2\, b_3\, b_4 = 010$	$b_2\, b_3\, b_4 = 101$	others
00	000	000	000
01	011	001	$0{\times}1$
10	110	100	$1{\times}0$
11	111	111	111

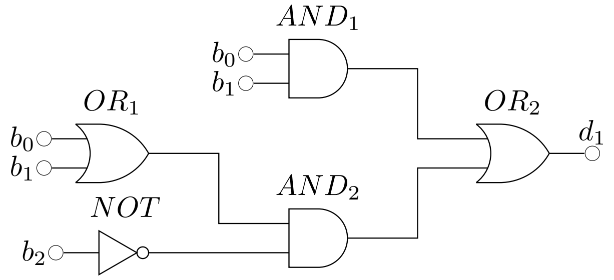


Figure 3.2: Encoder circuit for a 3×3 CAC.

as to satisfy both the FP and FT conditions. The encoding rule for the first and last columns is listed in Tab. 3.1. To simplify encoding and decoding, the encoding of $\{b_0, b_1\}$ and $\{b_5, b_6\}$ are both systematic, and only the wire in the middle needs to be encoded. For example, we can use $d_1 = b_0 b_1 + b_1 \bar{b}_2 + b_0 \bar{b}_2$. Implementing this encoder will use 2 *AND* gate, 2 *OR* gate and 1 *NOT* gate, and the circuit of the encoder is shown in Fig. 3.2.

Since the encoding is systematic, it is very easy to decode. For the received vectors $[d_0\, d_1\, d_2]^T$, $[d_3\, d_4\, d_5]^T$, and $[d_6\, d_7\, d_8]^T$, the decoding result is given by (d_0, d_2, d_3, d_4, d_5, d_6, d_8). No extra logic is needed.

Note that the code rate of this 3×3 TDCAC is $\frac{7}{9}$, as opposed to $\frac{2}{3}$ if using FTC coding scheme. Also the encoding and decoding complexities are low when properly designed.

Table 3.2: The code rates for $m \times n$ TDCACs with memory, satisfying the FT condition.

	$m=3$	$m=4$	$m=5$	$m=6$	$m=7$	$m=8$	$m=9$	$m=\infty$
$n=1$	0.6667	0.7500	0.6000	0.6667	0.7143	0.6250	0.6667	0.6942
$n=2$	0.8333	0.7500	0.8000	0.7500	0.7857	0.7500	0.7778	0.7768
$n=3$	0.7778	0.8333	0.8000	0.7778	0.8095	0.7917	0.8148	0.7998
$n=4$	0.8333	0.8125	0.8000	0.8333	0.8214	0.8125	0.8056	0.8119
$n=5$	0.8667	0.8500	0.8400	0.8333	0.8286	0.8250	0.8222	0.8192
$n=6$	0.8333	0.8333	0.8333	0.8333	0.8333	0.8333	0.8333	0.8240
$n=\infty$	0.8941	0.8826	0.8757	0.8712	0.8679	0.8654	0.8635	

3.4 Result and Discussion

To compare with FTCs and FPCs, the code rates of our TDCACs are defined to be the maximum number of data bits that can be transmitted over a wire in one clock cycle, which is given by

$$R = \frac{\lfloor \log_2 |\mathcal{C}| \rfloor}{mn} = \frac{\lfloor \log_2 K_n(m) \rfloor}{mn}, \tag{3.3}$$

where $|\mathcal{C}|$ denotes the codebook size. Note that the definition of the code rate also provides a measure of redundant bus usage and hence is equivalent with the code rates of one dimensional CACs. For TDCACs with memory, the simulation results as well as the asymptotic values of the code rate when m or n approaches infinity are shown in Tab. 3.2. From Tab. 3.2 we can see that the code rate decreases with m but increases with n. We observe that the code rate increase rapidly when n increases from 1 to 4. Thus, it suffices to consider TDCACs with small n.

The code rates of TDCACs are compared with those of one-dimensional CAC in Tab. 3.3. Our TDCACs have higher code rates than FTCs and FPCs.

We don't impose any restriction on the selection of code matrices. Thus the

Table 3.3: The code rates can be achieved by different coding schemes. Code 1 is a memoryless code using the FT condition to reduce crosstalk delay, code 2 is a memoryless code using complete valid transition, and code 3 is a code with memory using the FT condition to reduce crosstalk delay.

m	FTC	FPC	TDCAC with $n = 4$		
			Code 1	Code 2	Code 3
3	0.6667	0.6667	0.8333	0.8133	0.8333
4	0.7500	0.7500	0.8125	0.8125	0.8125
5	0.6000	0.8000	0.8000	0.8500	0.8000
6	0.6667	0.6667	0.7917	0.8333	0.8333
7	0.7143	0.7143	0.7857	0.8214	0.8214
8	0.6250	0.7500	0.7813	0.8125	0.8125
9	0.6667	0.6667	0.7778	0.8056	0.8056
10	0.7000	0.7000	0.7750	0.8000	0.8250

maximum codebook size is bounded by the code matrices that have the minimum number of possible code matrix to transition to. If these code matrices are removed from the codebook (this technique is called pruning [2]), it is likely to achieve a larger codebook size. Pruning is not considered in our research. TDCACs with pruning will be the topic of our future work.

Chapter 4

Efficient Ordering Schemes for High-Throughput MIMO Detectors

4.1 Introduction

The multiple-input multiple-output (MIMO) systems for wireless communication systems have attracted a lot of attention because of their high data rate transmission and improved link reliability. The maximum diversity gains of MIMO systems can be achieved by the exhaustive search based maximum likelihood (ML) detection, which has an exponential complexity with respect to the order of the modulation and the number of transmit antennas and hence is infeasible when high order modulation and many transmit antennas are employed. Numerous low-complexity ML or near-ML detection algorithms [7–12, 15, 38–42] have been proposed. Among these algorithms, the sphere detection (SD) algorithm [7–11], the stack algorithm [42], the K-Best algorithm [43, 44], and the memory-constraint tree-search (MCTS) [45] algorithm are able to achieve the exact ML or near ML performance while maintaining an affordable computational and hardware complexity for a wide range of signal-to-noise ratios (SNRs), constellation sizes, and numbers of antennas.

The SD, stack, and MCTS algorithms formulate the ML detection as a tree search problem, and they employ the depth-first, best-first, and MCTS strategy [45], respectively. The SD algorithm has a fixed memory requirement, but its worst case computational complexity — in terms of the number of the visited nodes — is significantly higher than its average computational complexity. In contrast, the stack algorithm has the smallest computational complexity among all the tree search strategies [12]. However, its worst case memory requirements are significantly larger than its average memory requirement. The MCTS algorithm requires a fixed size of memory. When the memory size is sufficient, it uses the best-first strategy and behaves like the stack algorithm; otherwise, it uses the depth-first strategy and behaves like the SD algorithm. Because of its mixed behavior, its computational complexity lies between the SD and the stack algorithm, even in the worst case.

The K-Best algorithm is based on a breadth-first tree search strategy. It is not an ML detection algorithm and suffers performance loss, especially when K is small. However, the K-Best algorithm is very suitable for hardware implementation, since it has a fixed memory requirement and computational complexity and can be easily pipelined to achieve high throughput due to its feed-forward structure.

The order of the columns in the channel matrix plays a significant role in the performance of MIMO detectors. For example, the computational complexities, in terms of nodes visited, of the SD, MCTS, and stack algorithms depend on the column order. Thus, several ordering schemes have been proposed for MIMO detectors in the literature. The vertical Bell Labs layered space-time (V-BLAST) ordering scheme reduces the average computational complexity of the SD algorithm [46], albeit not the worst case computational complexity [47]. The V-BLAST ordering scheme itself has a high computational complexity, and hence an efficient ordering scheme based on sorted QR decomposition (QRD) is proposed [48], which has a lower computational complexity. Ordering schemes that take into account both the channel matrix and the received signal (hence noise) have also been proposed [41,49]. The geometrically-inspired ordering scheme proposed in [41] leads to significant reduction of both average and worst case computational complexities for the SD algorithm in comparison to the V-BLAST ordering scheme. However, the scheme in [41] is proposed for a real-valued signal model, which is not suitable for hardware implementation [50, 51]. It also has a higher computational complexity than the sorted QRD ordering scheme and cannot be embedded in the QRD. Recently, we have proposed several new ordering schemes [13, 52], which have the following properties: taking into account both the channel matrix and the received signal; reducing both the average and worst-case computational complexities of the SD, MCTS, and stack algorithms [13, 52]; some being easily embedded into QRD.

In this chapter, we extend our previous work by investigating two problems that are important to high-throughput MIMO detectors. Since the K-Best algorithm is suitable for high-throughput MIMO detectors and hardware implementations, we first study how our ordering schemes affect the K-Best algorithm and compare them with previously proposed ordering schemes. Since our ordering schemes are more complex than some previously proposed ordering schemes, we are also interested in evaluating the incurred area and throughput penalty. To this end, the main contributions in this chapter are as follows:

- We apply our ordering algorithms to the K-Best algorithm. Our ordering schemes lead to better detection error rate for the K-Best detector than other ordering schemes, especially when K is small. Thus, with a given detection error rate, our ordering schemes either lead to SNR gains, or enable the usage of an even smaller K.

- We implement in hardware two of these ordering schemes, which can be easily embedded into the QR decomposition. Givens based QRD algorithms are adopted due to their numerical stability when fixed point representations are used. Our hardware implementation results show that our novel ordering schemes incur negligible overheads, when compared with other ordering schemes, and are particularly suitable for high-throughput implementations.

The remainder of the chapter is organized as follows. In Sec. 4.2, we briefly describe our system model and review the K-Best algorithm and various ordering schemes. In Sec. 4.3, our ordering schemes are applied to the K-Best algorithm and their effects are evaluated. The VLSI implementations of our ordering algorithms are described in Sec. 4.4. Finally, this chapter concludes in Sec. 4.5.

4.2 Background Review

4.2.1 System Model

Consider a MIMO system with N_r receive and N_t transmit antennas and assume that the channel is flat fading. The received signal vector $\mathbf{r} = (r_1, r_2, \cdots, r_{N_r})^T$ can be expressed as

$$\mathbf{r} = \mathbf{Hs} + \mathbf{n},$$

where r_i is the complex signal received at the i-th receive antenna. $\mathbf{H}$ is an $N_r \times N_t$ matrix characterizing the channel which is known perfectly at the receiver. The entry $H_{i,j}$ is a complex zero-mean unit-variance Gaussian random variable that represents the path gain from the j-th transmit antenna to the i-th receive antenna. The transmit signal vector is given by $\mathbf{s} = (s_1, s_2, \cdots, s_{N_t})^T \in \Omega^{N_t}$, where the signal s_j transmitted from the j-th antenna is chosen from a complex constellation point set Ω with $|\Omega| = 2^{M_c}$. Therefore each constellation point can be mapped into M_c bits. For the noise vector $\mathbf{n} = (n_1, n_2, \cdots, n_{N_r})^T$, n_i is circular symmetric complex additive white Gaussian noise (CSAWGN).

The ML MIMO detection problem can be formulated as

$$\mathbf{s}_{\mathrm{ML}} = \arg \min_{\mathbf{s} \in \Omega^{N_t}} ||\mathbf{r} - \mathbf{Hs}||^2.$$

An exhaustive search among all $2^{M_c N_t}$ possible candidates in Ω^{N_t} can solve this problem when M_c and N_t are relatively small. However, this strategy has an exponential complexity and it is impractical when high order modulation and many transmit antennas are employed. Assuming $N_r \geq N_t$ and $rank(\mathbf{H}) = N_t$, $\mathbf{H}$ can be decomposed as $\mathbf{H} = \mathbf{QR}$, where $\mathbf{Q}$ is an $N_r \times N_r$ unitary matrix satisfying $\mathbf{Q}^H\mathbf{Q} = \mathbf{I}$ (the superscription $(\cdot)^H$ denotes conjugate transpose) and $\mathbf{R}$ is an $N_r \times N_t$ upper triangular matrix with diagonal elements $R_{i,i} > 0$. Let the unconstrained zero-forcing (ZF) estimate $\hat{\mathbf{s}}$ be given by

$$\hat{\mathbf{s}} = (\mathbf{H}^H\mathbf{H})^{-1}\mathbf{H}^H\mathbf{r} = (\hat{s}_1, \hat{s}_2, \cdots, \hat{s}_{N_t})^T, \tag{4.1}$$

and the received vector $\mathbf{r}$ is then transformed to $\mathbf{y}$, where

$$\mathbf{y} = \mathbf{Q}^H\mathbf{r} = \mathbf{R}\hat{\mathbf{s}} = (y_1, y_2, \cdots, y_{N_r})^T. \tag{4.2}$$

Then the ML MIMO detection problem can be reformulated as

$$\begin{aligned} \mathbf{s}_{\text{ML}} &= \arg\min_{\mathbf{s}\in\Omega^{N_t}} ||\mathbf{y} - \mathbf{Rs}||^2 \\ &= \arg\min_{\mathbf{s}\in\Omega^{N_t}} ||\mathbf{R}(\hat{\mathbf{s}} - \mathbf{s})||^2 \\ &= \arg\min_{\mathbf{s}\in\Omega^{N_t}} \sum_{i=1}^{N_t} \Big| \sum_{j=i}^{N_t} R_{i,j}(\hat{s}_j - s_j) \Big|^2 \\ &= \arg\min_{\mathbf{s}\in\Omega^{N_t}} \sum_{i=1}^{N_t} \Big| y_i - \sum_{j=i}^{N_t} R_{i,j}s_j \Big|^2. \end{aligned} \tag{4.3}$$

Based on this formulation, the ML detection problem can be solved by a tree search algorithm. Consider an $(N_t + 1)$-level tree. Each node at the k-th level of this tree corresponds to a unique partial candidate vector $\mathbf{s}^k = (s_k, s_{k+1}, \cdots, s_{N_t})^T$ when $k \le N_t$, and an empty vector when $k = N_t + 1$. Each node at the lowest level ($k = 1$), called a leaf node, corresponds to a complete signal vector. Every node at the k-th level when $k \le N_t$ is associated with a metric $w(\mathbf{s}^k) = \sum_{i=k}^{N_t} m(\mathbf{s}^i)$, where

$m(\mathbf{s}^i) = |y_i - \sum_{j=i}^{N_t} R_{i,j} s_j|^2$, and the root node has a metric $w(\mathbf{s}^{N_t+1}) = 0$. The leaf node(s) with the smallest metric corresponds to the ML estimate, which can be found efficiently with proper tree search algorithms and pruning techniques, e.g., the SD algorithm, the stack algorithm, and the MCTS algorithm.

4.2.2 The K-Best Algorithm

Instead of the depth- or best-first strategy, the K-Best algorithm uses the breadth-first strategy to search the tree and to try to find the ML estimate. A strict breadth-first tree search algorithm without compromising on the ML performance expands all the nodes in each level, and hence it has the same computational complexity with the exhaustive search, which is exponential and becomes infeasible when higher order modulation and many transmit antennas are employed. Given a predefined number K and denote the number of nodes in the l-th level by K_l, the K-Best algorithm only expands $\min\{K_l, K\}$ nodes that have the smallest metrics among all the nodes in the l-th level of the tree [43, 44], and the computational complexity can be significantly reduced. The K-Best algorithm has a constant computational complexity (it visits no more than KN_t nodes) and has an easily pipelined architecture [44]. Therefore it is easy to achieve high throughput. It is an attractive solution of the MIMO detectors in practice.

However, the leaf node found by the K-Best algorithm may not be the ML estimate, and hence the K-Best algorithm suffers from a performance degradation in terms of detection error rate. The performance of the K-Best algorithm is tunable through the parameter K. When a larger K is used, the detection error rate degradation becomes smaller while the computational complexity is increased.

4.2.3 Ordering Schemes

The column ordering of the channel matrix is an important factor which affects the computational complexity of the tree search based MIMO detection algorithms. For example, when the initial radius of the SD algorithm is set to infinity, the first point that the SD algorithm finds, often called the Babai point, is also called the ZF interference cancelation (ZF-IC) estimate [10]. Different column ordering affects the distance between the Babai point and the ML estimate, and hence the number of nodes falling inside the sphere [11]. Therefore the column ordering of the channel matrix can affect the computational complexity of the SD algorithm. Thus, the tree search is usually preceded by a preprocessing phase, which consists of ordering and QRD. To reduce the complexity of the preprocessing phase, the ordering needs to be carried out with a low computational complexity. For the same reason, it is desirable that the ordering scheme can be embedded in the QRD process.

The V-BLAST ordering scheme proposed in [46] maximizes $\min_{1\leq i\leq N_t} R_{i,i}$ by maximizing $R_{i,i}$ for i from N_t to 1, and requires $O(N_t^2/2)$ QRDs of permutations of $\mathbf{H}$ [48]. The sorted QRD ordering algorithm [48] simplifies the V-BLAST ordering by minimizing $R_{i,i}$ for i from 1 to N_t, and it can be readily embedded into the QRD process. The H-norm ordering scheme [11] orders the columns of the channel matrix according to the column norm $||\mathbf{h}_i||$, where $\mathbf{h}_i$ is the i-th column of the channel matrix, and hence it has the lowest computational complexity because it only needs to calculate N_t column norms. These three ordering schemes consider the channel matrix only. In [49], an ordering scheme is proposed to detect the symbols by an increasing order of $\min_{s_i\in\Omega}|s_i - \hat{s}_i|^2$, where $\hat{s}_i$ is the i-th element of the unconstrained ZF estimate.

Our new ordering schemes proposed in [52] are based on a study of the data collected from worst-case scenarios in the simulation of the SD algorithm and aim to reduce the computational complexities in these scenarios. It is observed that the SD algorithm has a much higher computational complexity when the channel is ill-conditioned and the symbol with the largest unconstrained ZF estimate is detected last. In this case, the Babai point is far from the ML estimate, and hardly any nodes can be pruned from the tree. The simulation results show that if the symbol with the largest unconstrained ZF estimate is detected first, the computational complexity can be significantly reduced. However, the symbol with the largest unconstrained ZF estimate usually corresponds to the weakest signal, and detecting the weakest signal first will increase the computational complexity in the well-conditioned channel. Therefore the benefit from reducing the computational complexity in the ill-conditioned channel is mitigated or even eliminated by the increase of the complexity in the well-conditioned channel, and the average computational complexity may be significantly increased. We need to make a computational complexity balance between the well-conditioned and ill-conditioned channels.

Our first ordering scheme proposed in [13, 52], referred to as the balanced sorted QRD (BSQRD) algorithm, sorts the columns of the channel matrix according to the product between the diagonal element of the upper-triangular matrix and the absolute value of the unconstrained ZF estimate, i.e., $R_{i,i}|\hat{s}_i|$. When the unconstrained ZF estimate $\hat{\mathbf{s}}$ is not available, our simplified BSQRD (SBSQRD) ordering scheme orders the channel matrix columns according to the inner product of the channel matrix columns and the received vector, i.e., $|\mathbf{h}_i^H \mathbf{r}|$. Those two ordering schemes

Algorithm 2 Modified Gram-Schmidt based SBSQRD algorithm

```
Input: channel matrix $\mathbf{H}$, received signal vector $\mathbf{r}$
 1: $\mathbf{R} = \mathbf{0}$, $\mathbf{Q} = \mathbf{H}$, $\mathbf{p} = (1, 2, \cdots, N_t)$
 2: for $i = 1$ to $N_t$ do
 3:     $p_i = \arg\min_{i \le j \le N_t} (|\mathbf{q}_j^H \mathbf{r}|)$
 4:     exchange columns $i$ and $p_i$ in $\mathbf{Q}$, $\mathbf{R}$, $\mathbf{p}$, and $\mathbf{r}^T$
 5:     $R_{i,i} = ||\mathbf{q}_i||$
 6:     $\mathbf{q}_i = \mathbf{q}_i / R_{i,i}$
 7:     for $j = i + 1$ to $N_t$ do
 8:         $R_{i,j} = \mathbf{q}_i^H \cdot \mathbf{q}_j$
 9:         $\mathbf{q}_j = \mathbf{q}_j - R_{i,j} \cdot \mathbf{q}_i$
10:     end for
11: end for
12: Output: upper-triangular matrix $\mathbf{R}$, order vector $\mathbf{p}$
```

can both be embedded in a QRD algorithm. In [13], they are embedded in a modified Gram-Schmidt (MGS) based QRD algorithm in particular. The MGS based SBSQRD algorithm is shown in Alg. 2, which is very similar to the MGS based BSQRD algorithm shown in [13]. In Alg. 2, $\mathbf{q}_i$ is the i-th column of the matrix $\mathbf{Q}$.

The idea of V-BLAST can also be applied to our ordering schemes, i.e., maximizing $\min_{1 \le i \le N_t} R_{i,i}|\hat{s}_i|$ and $\min_{1 \le i \le N_t} |\mathbf{q}_i^H \mathbf{r}|$, respectively [13]. However, the V-BLAST versions of the BSQRD and SBSQRD have a much higher complexity than the original BSQRD and SBSQRD, and they cannot be embedded in QRD. In [13], it is shown that our ordering schemes significantly reduce the computational complexities of the SD, MCTS, and stack algorithms.

4.3 K-Best Detection Error Rate Improvement

It is easy to show that the ordering of the channel matrix columns affects the performance of the K-Best algorithm. Consider the extreme case of $K = 1$ for the K-Best

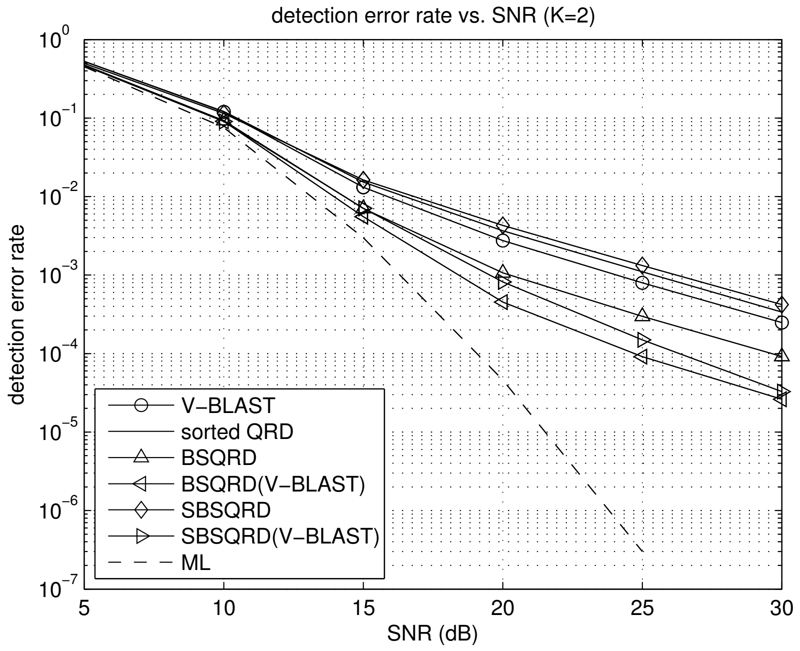


Figure 4.1: The detection error rates of the K-Best detector under different ordering schemes when $K = 2$ for a 4×4 QPSK modulated MIMO system.

algorithm. In this case, the detection process of the K-Best algorithm corresponds to that of the SD algorithm until finding the first leaf node, and the K-Best algorithm produces the aforementioned Babai point in Sec. 4.2.3. As the ordering of the channel matrix columns affects the distance between the Babai point and the ML estimate, it also affects the detection error probability of the K-Best detector.

In order to evaluate the detection error rate improvement of our ordering schemes to the K-Best algorithm and to compare them to other ordering schemes, we numerically obtain the detection error rates of the K-Best detector with different ordering schemes for MIMO systems with various numbers of transmit and receive antennas and constellation sizes.

Figs. 4.1–4.3 compare the detection error rates of the K-Best algorithm for $K =$

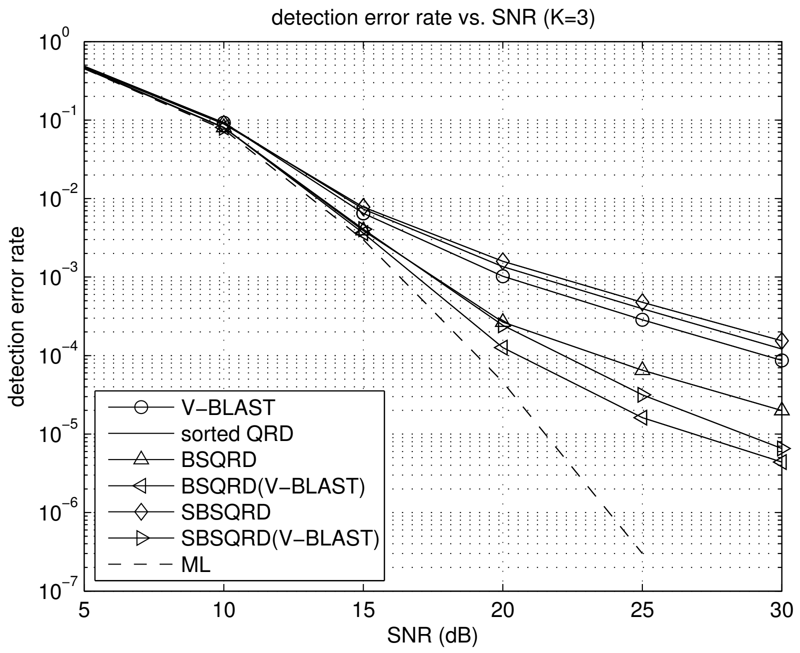


Figure 4.2: The detection error rates of the K-Best detector under different ordering schemes when $K = 3$ for a 4×4 QPSK modulated MIMO system.

2, 3, and 4, respectively, with different ordering schemes for a QPSK modulated 4×4 MIMO system. The detection error rates of the ML detector are also shown in the these figures as a benchmark. Due to the small constellation size and small numbers of transmit and receive antennas, small values of K are selected to illustrate the impact of our ordering schemes. For a larger constellation size and/or larger numbers of transmit and receive antennas, larger K's can be used as well. As shown in Fig. 4.3, when $K = 4$, the detection error rates of the K-Best detector with all different ordering schemes approach that of the ML detector. Thus, for $K \geq 4$, our ordering schemes achieve no gains since the K-Best detector achieves near-ML performance. As shown in Figs. 4.1 and 4.2, when $K = 2$ and $K = 3$, the detection error rates depend on the ordering scheme. We now compare our BSQRD and

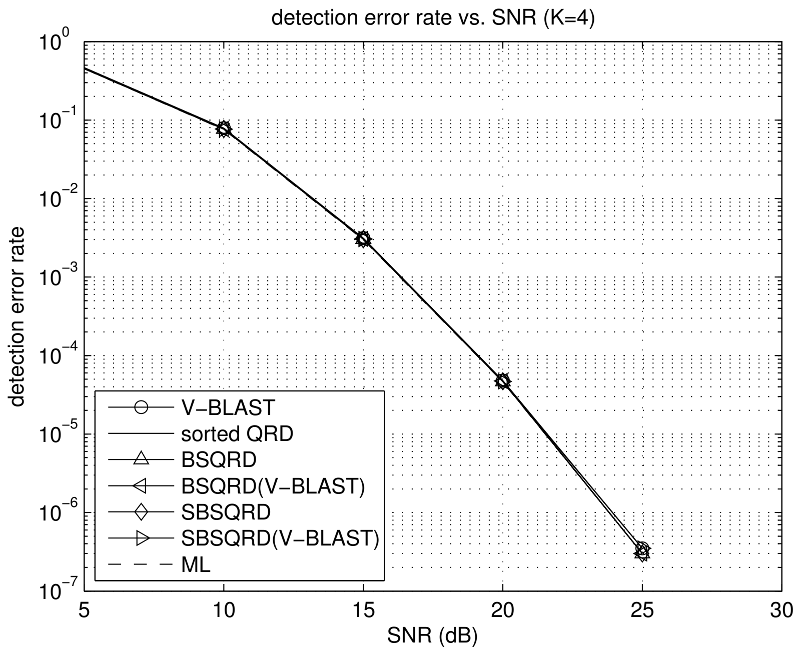


Figure 4.3: The detection error rates of the K-Best detector under different ordering schemes when $K = 4$ for a 4×4 QPSK modulated MIMO system.

SBSQRD ordering schemes with the sorted QRD ordering algorithm since they have similar complexities, and similarly compare the V-BLAST versions of our BSQRD and SBSQRD ordering schemes with the V-BLAST ordering scheme. For $K = 2$, when compared with the sorted QRD algorithm, our SBSQRD ordering scheme is slightly worse, while our BSQRD has a gain of roughly 5 dB at the detection error rate of 10^{-3}. The V-BLAST versions of our BSQRD and SBSQRD ordering schemes also have gains of roughly 5 dB at the detection error rate of 10^{-3}, when compared with the V-BLAST ordering scheme. The comparisons for $K = 3$ are similar. When compared with the sorted QRD algorithm, our SBSQRD ordering scheme is slightly worse, while our BSQRD ordering scheme has a gain of roughly 4.5 dB at the detection error rate of 10^{-3}. When compared with the V-BLAST ordering scheme, the V-BLAST versions of our BSQRD and SBSQRD ordering schemes have gains of at least 2.5 dB at the detection error rate of 10^{-3}. We remark that the curves for our BSQRD ordering scheme and two V-BLAST versions have steeper slope than their respective counterparts and possibly lower error floors. Thus, the SNR gains are greater for a smaller detection error rate.

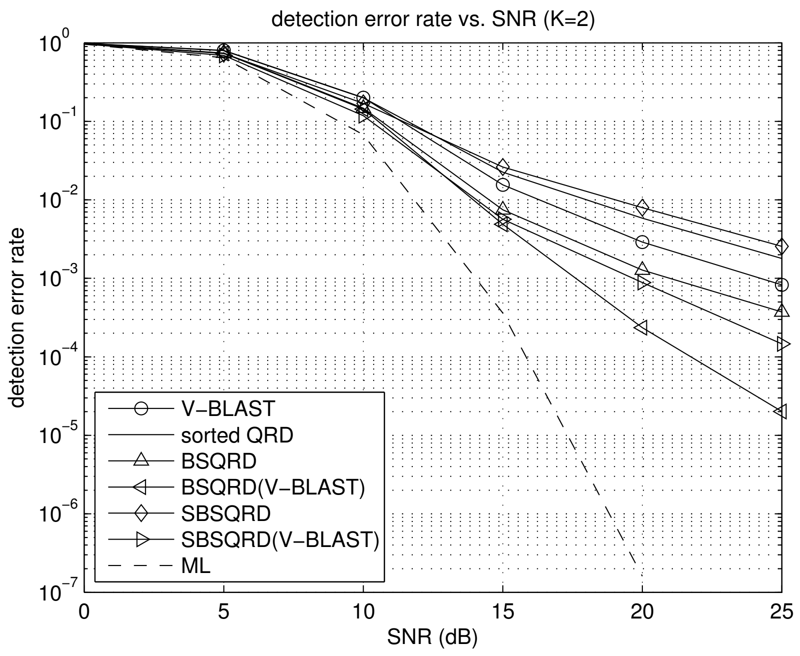


Figure 4.4: The detection error rates of the K-Best detector under different ordering schemes when $K = 2$ for a 8×8 QPSK modulated MIMO system.

Figs. 4.4 and 4.5 compare the detection error rates of the K-Best algorithm for $K = 2$ and 4, respectively, with different ordering schemes for a QPSK modulated 8×8 MIMO system. The comparisons show a similar trend. For $K = 2$ (Fig. 4.4), our SBSQRD algorithm is slightly worse than the sorted QRD algorithm, while our other ordering schemes achieve even greater gains over their counterparts than in Fig. 4.1. Due to greater numbers of antennas, our BSQRD ordering scheme and two

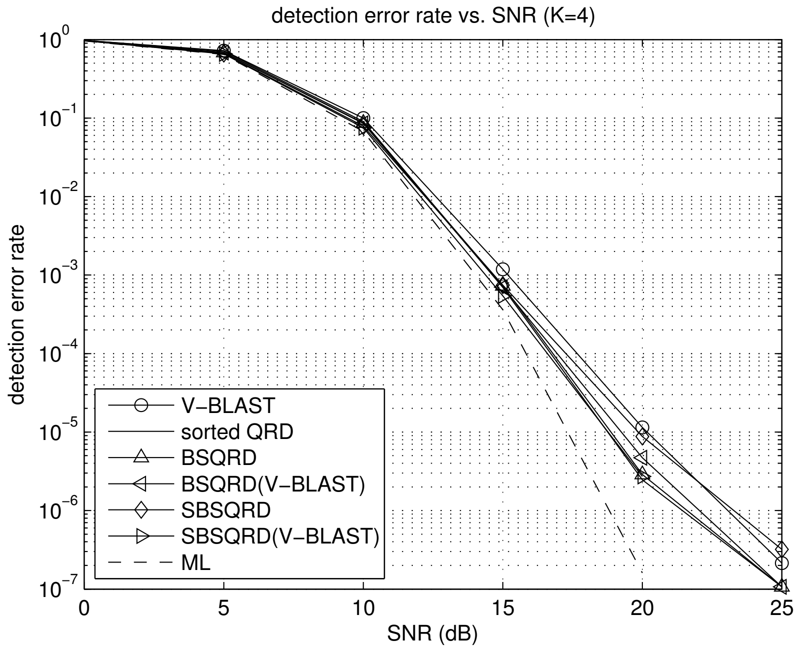


Figure 4.5: The detection error rates of the K-Best detector under different ordering schemes when $K = 4$ for a 8×8 QPSK modulated MIMO system.

V-BLAST versions show small gains even when $K = 4$.

Figs. 4.6 and 4.7 compare the detection error rates of the K-Best algorithm for $K = 6$ and 12, respectively, with different ordering schemes for a 16QAM modulated 4×4 MIMO system. Due to the greater signal constellation, even at $K = 6$, our BSQRD ordering scheme and two V-BLAST versions show modest gains versus their respective counterparts at the detection error rate of 10^{-3}. For $K \geq 12$, the K-Best detector achieves near-ML performance regardless of ordering schemes.

We have also obtained numerical results for other settings (for example, 64QAM modulated MIMO systems), and the conclusions are similar. Due to limited space, these additional numerical results are omitted.

Our SBSQRD ordering scheme always has slightly worse performance than the sorted QRD algorithm. To reduce complexity, our SBSQRD approximates our BSQRD ordering scheme by using $\mathbf{q}_i^H\mathbf{r}$ to approximate $||\mathbf{q}_i|| \cdot |\hat{s}_i|$, and it seems that this approximation leads to worse performance of the K-Best detector. On the other hand, our BSQRD ordering scheme and two V-BLAST versions all achieve SNR gains versus their respective counterparts. Furthermore, when K is fixed, these SNR gains increase as the constellation size or the numbers of antennas increase. This is because the performance loss by the K-Best detector is greater when the tree is wider or has more levels, and good ordering schemes help in minimizing the loss. Finally, for a given MIMO system, these SNR gains decrease as K increases, and become negligible when K is such that the K-Best detector approaches the ML detector.

In summary, our BSQRD ordering scheme and two V-BLAST versions improve the performance of the K-Best detector. As shown above, they improve the detection error rate of the K-Best detector, thereby leading to improved overall system performance. For a fixed detection error rate, our BSQRD ordering scheme and two V-BLAST versions either lead to SNR gains, as shown above, or enable the usage of a smaller K. For example, when a detection error rate of 10^{-3} is required, the substitution of our BSQRD ordering scheme for the V-BLAST algorithm would allow us to use $K = 2$ instead of 3 in a QPSK modulated 4×4 MIMO system. Of course, a smaller K in turn leads to advantages, such as smaller areas, in hardware implementations.

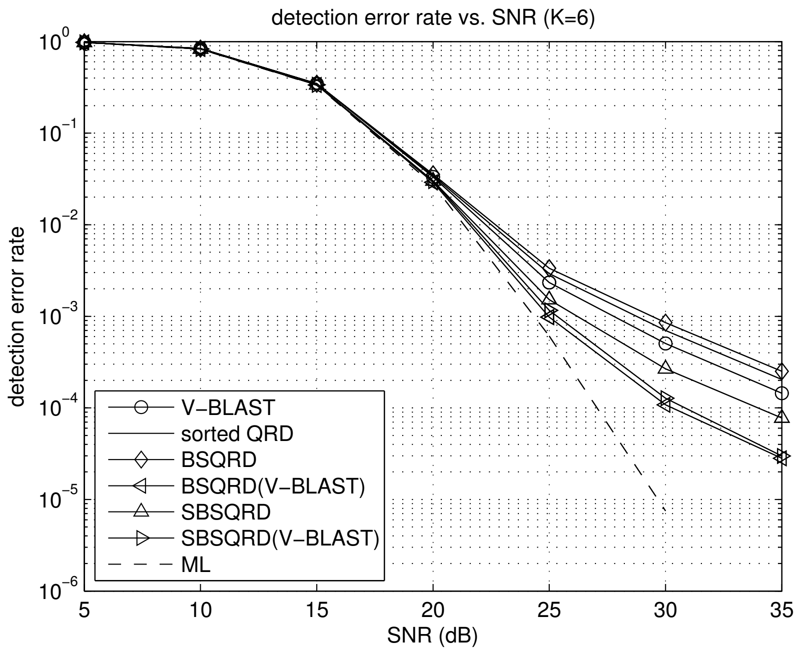


Figure 4.6: The detection error rates of the K-Best detector under different ordering schemes when $K = 6$ for a 4×4 16QAM modulated MIMO system.

4.4 Hardware Implementation of Our Ordering Schemes

In order to evaluate the overheads incurred by our ordering schemes and to verify their feasibility for high-throughput hardware implementations, the BSQRD and SB-SQRD ordering schemes are implemented in hardware. These two ordering schemes are selected because of their relative simplicity and good performance. Both can be embedded into a QRD algorithm in order to reduce the complexity and delay of the preprocessing stage. Both are simpler than their respective V-BLAST versions, while achieving good performance. The corresponding V-BLAST version ordering schemes have much higher complexities, and they cannot be embedded into the

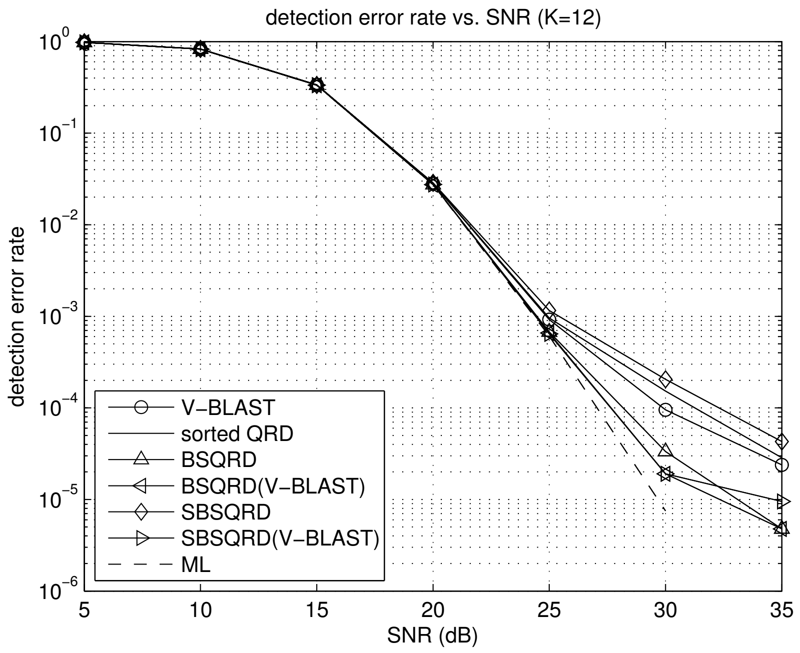


Figure 4.7: The detection error rates of the K-Best detector under different ordering schemes when $K = 12$ for a 4×4 16QAM modulated MIMO system.

QRD algorithm. We list the complexities of the BSQRD and SBSQRD ordering schemes as well as their V-BLAST versions in Tab. 4.1. As shown in Sec. 4.3, our BSQRD ordering scheme improves the error performance of the K-Best detector, and also reduces the computational complexities of the SD, stack, and MCTS algorithms [13, 52]. Despite its relative poor performance with the K-Best detector, our SBSQRD ordering scheme is still considered here because it has an even lower complexity than the BSQRD and reduces the computational complexities of the SD, stack, and MCTS algorithm, although it incurs small detection performance loss to the K-Best algorithm.

Table 4.1: Complexities of the ordering schemes in terms of the number of multiplications.

Ordering scheme	Complexity
BSQRD	$O(N_t^2 N_r)$
SBSQRD	$O(N_t^2 N_r)$
V-BLAST BSQRD	$O(N_t^3(2N_r + N_t))$
V-BLAST SBSQRD	$O(N_t^3(2N_r + N_t))$

4.4.1 Givens Rotation Based BSQRD and SBSQRD Ordering Schemes and Fixed Point Representations

Our BSQRD and SBSQRD ordering schemes proposed in [52] are embedded into a MGS based QRD procedure. Both have performed well in our numerical simulations thus far since we use the floating point numbers with double precision, for which numerical errors are negligible. However, fixed point numbers are more suitable for VLSI implementations, and hence both quantization errors and numerical stability must be taken into account now. To this end, we simulate the detection error rate performance of the SD algorithm with double precision floating numbers. However, the preprocessing is done with our BSQRD ordering schemes using **different** settings, and the simulation results are shown in Fig. 4.8. The QRD algorithm in settings 1 and 2 in Fig. 4.8 uses floating point numbers with double precision, and in settings 3 and 4 uses a fixed point representation, where a number is represented by 13 bits, including one sign bit and 8 fractional bits. Therefore, a 13-bit integer d represents a rational number $d/2^8 \in [-16, 16)$. The fixed point toolbox in MATLAB is used in our simulations.

We first evaluate the effect of quantization errors by comparing settings 1 and 2. In both settings, the double precision floating point representation are used for

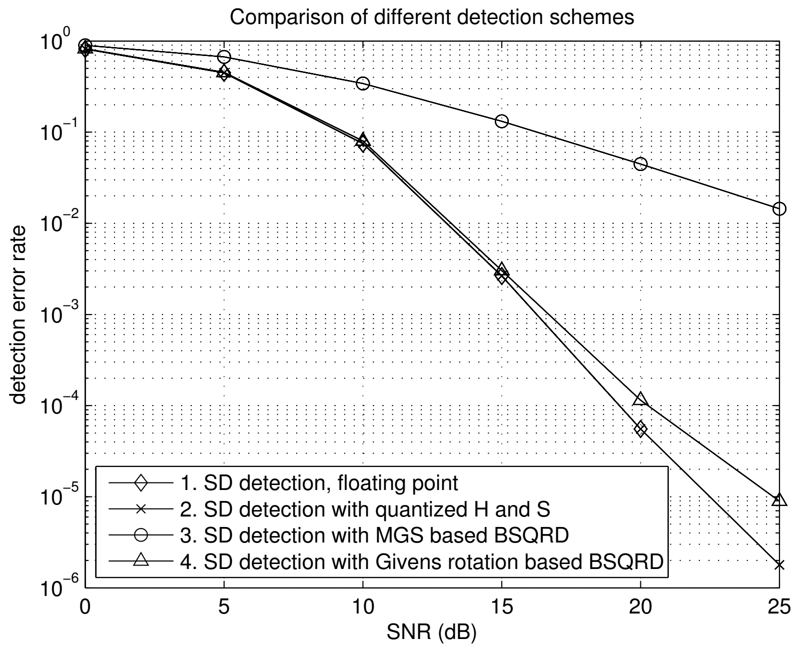


Figure 4.8: Detection error rates comparison between floating point BSQRD, floating point BSQRD with quantized inputs, fixed point MGS based BSQRD, and fixed point Givens rotation based BSQRD. The detection algorithm for all these curves is the SD algorithm using double precision floating numbers.

our BSQRD ordering scheme. The two settings differ in their inputs. In setting 1, the channel matrix $\mathbf{H}$ and the unconstrained ZF estimate $\hat{\mathbf{s}}$ are both represented in double precision floating point and hence the SD algorithm is an ML detector. In setting 2, the channel matrix $\mathbf{H}$ and the unconstrained ZF estimate $\hat{\mathbf{s}}$ are both quantized using the 13-bit fixed point number representation. In Fig. 4.8, the two curves corresponding to these two settings are on top of each other. This shows that the 13-bit fixed point number representation is precise enough to represent $\mathbf{H}$ and $\hat{\mathbf{s}}$, and the incurred quantization errors are inconsequential.

We then consider numerical stability when a fixed point number representation is used. As observed in [53], the MGS based QRD algorithm is not numerically stable due to the division operations it involves, and hence the performance of the MGS based QRD with a fixed point number representation is far from that with a double precision representation. Division is not stable since when the absolute value of the divisor is very small, a small error in the divisor will cause a large error in the quotient. Furthermore, dividing a small number usually results in a large quotient, and the range of the fixed point numbers has to be designed large enough in order to avoid overflow, which will increase the hardware complexity. Our simulation results confirm this observation. When the BSQRD procedure is carried out with a fixed point number representation based on the MGS algorithm, the detection error rate curve deviates greatly from the ML performance, as shown by setting 3 in Fig. 4.8.

Due to its numerical instability, the sorted QRD based on the MGS algorithm [48, 53] is costly in hardware implementations. For example, the preprocessing module in [54] uses floating number representation directly. In [53], to mitigate the impact of the MGS numerical instability, the word length is designed to have 20 bits and each column $\mathbf{h}_i$ of the matrix is associated with a column exponent e_i. The number saved in the matrix is thus given by $H_{i,j} \cdot 2^{e_i}$. Due to this compromise between the floating point number and fixed point number representations, the MGS based QRD becomes more stable with only a small detection error rate degradation at the expense of more complex logic. However, this scheme requires more area and has a long latency and a low throughput.

Instead, the Givens rotation based QRD algorithm is considered since it is numerically more stable than that based on MGS algorithm (see, e.g., [55]). A Givens rotation is represented by an $N \times N$ matrix $\mathbf{G}(i, j, \theta)$, which is an identity matrix with the substitutions $G_{i,i} = G_{j,j} = \cos\theta$, $G_{i,j} = \sin\theta$, $G_{j,i} = -\sin\theta$ [56]. Note that $\mathbf{G}(i,j,\theta)$ is a unitary matrix. Given a matrix $\mathbf{A}$ with $A_{i,k}=a$ and $A_{j,k}=b$, if we want to introduce a zero element at position (j,k), we can simply multiply $\mathbf{A}$ from the left with the matrix $\mathbf{G}(i,j,\theta)$, where $\theta=\arctan\left(\frac{b}{a}\right)$ or $\arctan\left(\frac{b}{a}\right)+\pi$ if $a\neq 0$, and $\theta=\pm\frac{\pi}{2}$ if $a=0$. Finding the appropriate Givens rotation $\mathbf{G}(i,j,\theta)$ and applying it to a matrix can both be done at the same time with the coordinate rotation digital computer (CORDIC) algorithm [57]. Given a two-dimensional vector $(x,y)^T$, a vector mode CORDIC algorithm can rotate it to a vector $(r,0)^T$ where $r=\kappa\sqrt{x^2+y^2}$ and the scaling factor $\kappa=1.64676$ is a constant. Thus, after a Givens rotation by the CORDIC algorithm, it is necessary to re-scale the result with $\frac{1}{\kappa}$. The CORDIC algorithm is easy to be implemented with only addition and shift operations. Therefore, applying a sequence of Givens rotations can transform a matrix $\mathbf{A}$ into an upper triangular matrix in a numerically stable way. The details of the VLSI architecture of Givens rotation based QRD algorithms by the CORDIC algorithm can be found in [55].

The SBSQRD algorithm based on Givens rotation is restated in Alg. 3. The Givens rotation based BSQRD algorithm is the same with Alg. 3 except for the following changes: (1) the input is the channel matrix $\mathbf{H}$ and the unconstrained ZF estimate $\hat{\mathbf{s}}$; (2) line 3 should be changed to $x_i=||\mathbf{h_i}||^2||\hat{\mathbf{s}}||^2$; (3) the i-th and p_i-th elements of $\hat{\mathbf{s}}$ should also be exchanged when executing line 7; and (4) line 17 should be changed to $x_k=x_k-|h_{i,k}|^2\cdot|\hat{s}_k|^2$. Note that our algorithm does not produce the unitary matrix $\mathbf{Q}$, which is unnecessary if the unconstrained ZF estimate is computed ahead of the ordering module since $\mathbf{y}=\mathbf{R}\hat{\mathbf{s}}$ by (4.2). However, $\mathbf{Q}$ can be derived by performing to an $N_r\times N_r$ identity matrix the same row multiplications and Givens rotations on $\mathbf{H}$ at the expense of extra logic and registers.

Algorithm 3 Givens rotation based SBSQRD algorithm

```
Input: channel matrix $\mathbf{H}$, received signal vector $\mathbf{r}$
 1: $\mathbf{p} = (1, 2, \cdots, N_t)$
 2: for $i = 1$ to $N_t$ do
 3:     $x_i = \mathbf{h}_i^T \cdot \mathbf{r}$
 4: end for
 5: for $i = 1$ to $N_t$ do
 6:     $p_i = \arg \min_{i \leq k \leq N_t} |x_k|^2$
 7:     exchange columns $i$ and $p_i$ in both $\mathbf{H}$ and $\mathbf{p}$
 8:     for $k = i$ to $N_r$ do
 9:         Using CORDIC algorithm to find out $\phi_k$ such that $H_{k,i}e^{-j\phi_k} = |H_{k,i}|$
10:         for $l = i$ to $N_t$ do
11:             $H_{k,l} = H_{k,l}e^{-j\phi_k}$
12:         end for
13:     end for
14:     Compute a sequence of Givens rotation $\Theta_u$ such that the rows $i+1, i+2, \cdots, N_r$ in columns $\mathbf{h}_i$ become zero.
15:     $\mathbf{H} = (\prod \Theta_u)\mathbf{H}$
16:     for $k = i + 1$ to $N_t$ do
17:         $x_k = x_k - H_{i,k}x_i$
18:     end for
19: end for
20: $\mathbf{R} = \mathbf{H}$
21: Output: upper-triangular matrix $\mathbf{R}$, order vector $\mathbf{p}$
```

The Givens rotation based BSQRD is much more numerically stable than the MGS based BSQRD. As a result, as shown in Fig. 4.8, setting 4 which corresponds to the Givens rotation based BSQRD has a much smaller detection error rate than that for the MGS based BSQRD, and it is quite close to the ML detector. This also shows that the aforementioned 13-bit fixed point number representation is accurate enough for the Givens rotation based BSQRD algorithm. As the numerical stability issue only affects the QRD algorithm where our ordering schemes are embedded, a similar conclusion holds for the Givens rotation based SBSQRD algorithm.

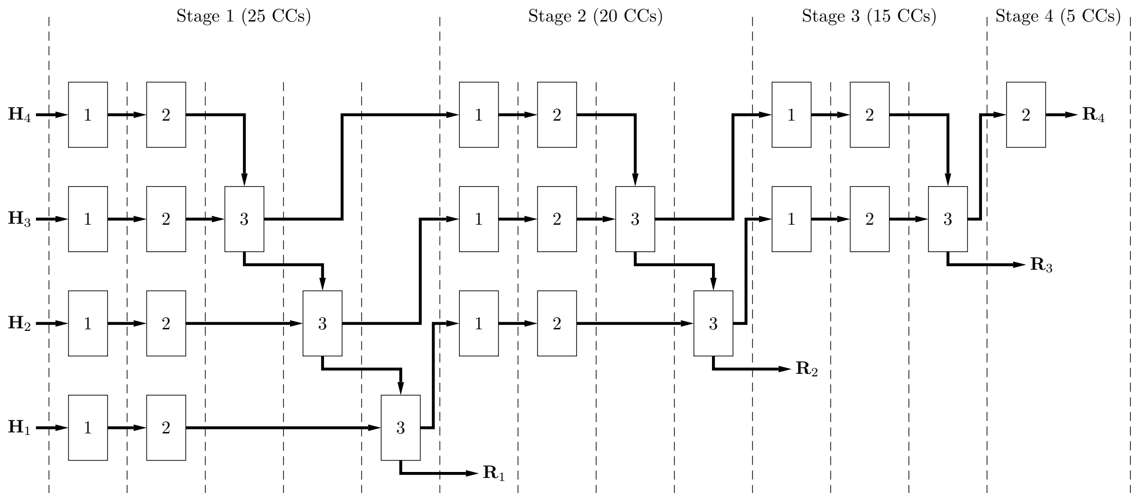


Figure 4.9: A pipelined architecture for QR decomposition of a 4×4 complex matrix (CC stands for clock cycle). All dashed lines represent pipeline registers. Each stage are further decomposed into sub-stages, and each sub-stage takes 5 CCs. $\mathbf{R}_i$ and $\mathbf{H}_i$ denotes the i-th row of $\mathbf{R}$ and $\mathbf{H}$, respectively.

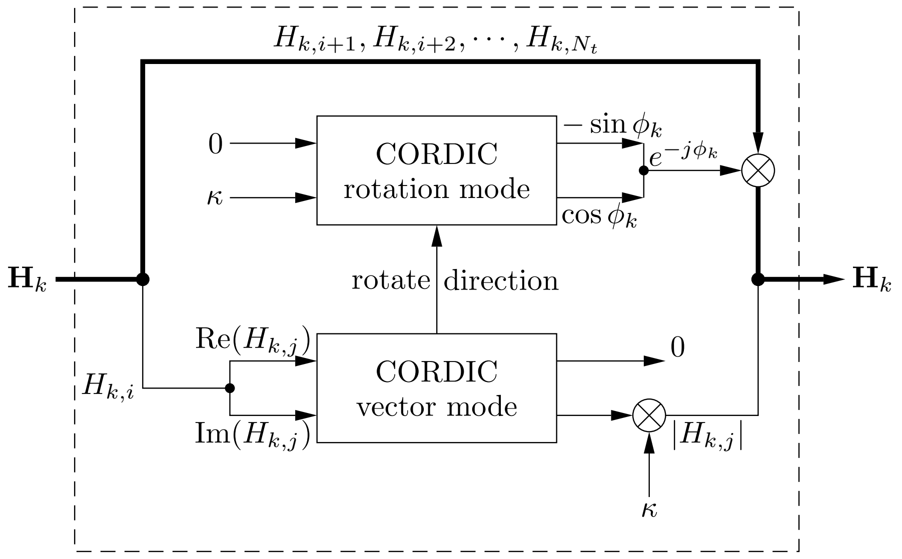


Figure 4.10: The circuitry for type-2 PEs at the i-th stage. The inputs are the elements of the j-th row of $\mathbf{H}$. The description of the vector mode and rotation mode CORDIC algorithm can be found in literature, e.g., [58]. Note that $H_{k,i}$ is a real number at the output.

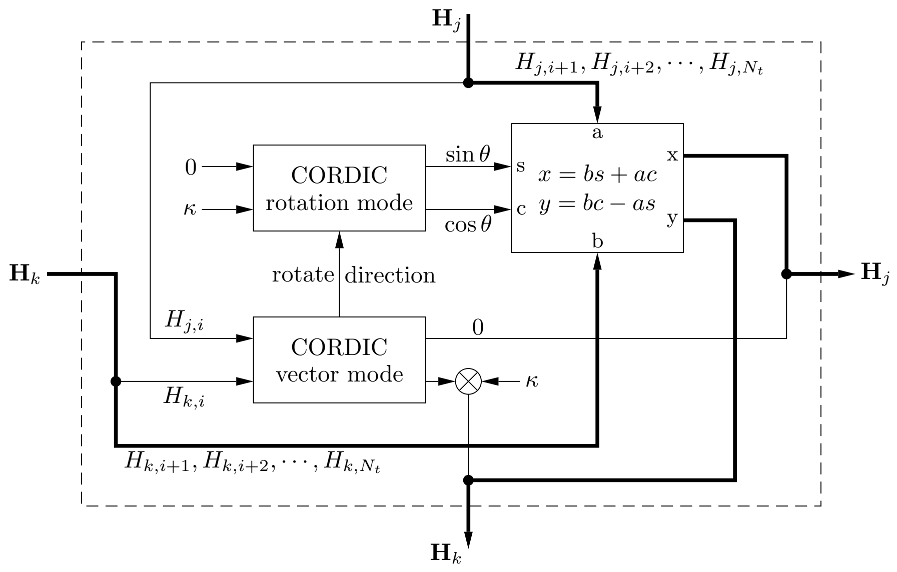


Figure 4.11: The circuitry for type-3 PEs at the i-th stage. The inputs are the j- and k-th rows of $\mathbf{H}$. The box on the upper right computes the linear combinations of the j- and k-th rows of $\mathbf{H}$ with the equations on the box.

4.4.2 Pipelined Architectures for High Throughput

MIMO communication systems nowadays often have a very high data throughput (measured by data bits per second). For example, in IEEE 802.11n [59], the maximum data rate is 600 Mbit/s with a 64QAM modulated 4×4 MIMO system. Therefore, the preprocessing stage should be designed to achieve high throughput. However, the data throughput of the preprocessing stage depends on the processing speed of the ordering scheme and sometimes the channel condition. For example, the sorted QRD needs to process each channel instantiation instead of each received signal vector. Hence, in VLSI implementations of the sorted QRD in [53] and [55] , the speed is measured in the number of channel matrices processed per second. The drawback of this metric is that the data throughput it translates into depends on the channel condition. Under a slow fading channel, the speed in [53] and [55] — roughly

1.56 and 2.08 million channel matrices per second, respectively — may represent a much higher throughput. But for a fast fading channel, the data throughputs of these implementations are much lower, and can become a bottleneck. In contrast, as remarked in [52], our BSQRD or SBSQRD ordering schemes are performed for each received signal vector, since both ordering schemes depend not only on the channel matrix but also on the received signal. For our ordering schemes, the number of channel matrices processed per second is proportional to the data throughput, regardless of the channel condition.

To achieve high throughput, a pipelined architecture for our ordering schemes is proposed, and Fig. 4.9 illustrates the pipelined architecture for a 4×4 MIMO system. In this pipelined architecture, the QRD process, in which our ordering schemes are embedded, is divided into N_t stages. Stage i corresponds to the i-th column, and consists of three types of processing elements (PEs), denoted by type-1, 2, and 3, respectively in Fig. 4.9. We are going to explain this figure by embedding Alg. 3 into it. The type-1 PEs correspond to lines 2-4 or lines 16-18 as well as lines 6 and 7 in Alg. 3. The type-1 PEs initialize or update the vector $X = (x_1, x_2, \cdots, x_{N_t})$ in Alg. 3, find the minimum, and exchange the columns of $\mathbf{H}$ and $\mathbf{p}$. Corresponding to lines 8-13 in Alg. 3, the type-2 PEs rotate the elements of the rows in $\mathbf{H}$ simultaneously to make the elements $H_{j,i} = |H_{j,i}|$ for all $j \geq i$ in the i-th stage, and the structure for these PEs are shown in Fig. 4.10. The type-3 PEs in the i-th stage correspond to lines 14 and 15, and perform on adjacent rows of $\mathbf{H}$ to introduce 0 in rows $i+1, i+2, \cdots N_r$ of $\mathbf{h}_i$, and the structure for these PEs are shown in Fig. 4.11. The structures of the vector mode and rotation mode CORDIC modules in Fig. 4.10 and 4.11 can be found in literature, e.g. [58], and hence are omitted. Note that in the last stage, since only one column is left, the type-1 PE is not needed. Furthermore, when $N_r = N_t$, no row in the last column needs to be set to zero, and hence we do not need the type-3 PE in the final stage. When $N_r = N_t$, type-3 PEs are still required to introduce 0 elements to rows $N_t + 1, N_t + 2, \cdots, N_r$ of $\mathbf{h}_i$. Hence, in Fig. 4.9, only the type-2 PE is needed in the last stage. Since we have explained the difference between the Givens rotation based BSQRD algorithm and Alg. 3, it is easy to implement the BSQRD ordering scheme in the architecture shown in Fig. 4.9.

To balance the work load and achieve high throughput, each stage is further divided into sub-stages, and pipeline registers are inserted between all sub-stages. In stage i, the type-1 PEs and the type-2 PEs occupies a sub-stage together, respectively, and each type-3 PEs requires one sub-stages (this is determined by the longest delay which can be seen from Fig. 4.9) to introduce 0 to $N_r - i$ elements in the i-th column, hence the i-th stage requires $N_r - i + 2$. To improve the throughput, each sub-stage is assigned five clock cycles in our implementation. This pipelined architecture can process one matrix every five clock cycles. Furthermore, assuming $N_r = N_t$, there are $(N_t - 1)\,\frac{4+N_t}{2} + 1$ sub-stages, and hence the latency is $5\left[(N_t - 1)\,\frac{4+N_t}{2} + 1\right]$ clock cycles. For example, the pipelined architecture for a 4×4 MIMO system in Fig. 4.9 has a latency of 65 clock cycles.

Table 4.2: Comparison of hardware implementations of the sorted QRD ordering scheme and our BSQRD and SBSQRD ordering schemes.

	non-pipelined					pipelined	
	BSQRD[1]	SBSQRD[1]	[55][1]	[55]	[53]	BSQRD[2]	SBSQRD [2]
Technology (nm)	45	45	45	250	250	45	45
Word length: fractional length	13:10	13:10	13:10	13:10	20:17	13:8	13:8
Core Area (mm^2)	0.146	0.150	0.134	0.785	0.997	0.845	1.017
Equivalent Gate Count	77.8k	80.0k	71.2k	48.7k	61.8k	450k	542k
f_{clk} (MHz)	250	250	250	166	162	300	300
Latency (CCs)	80	80	80	80	104	65	65
Speed (million matrices/s)[3].	13	3.13	3.13	2.08	1.56	60	60
Data throughput (Mbit/s)[4]	75	75	–	–	–	1440	1440

[1]All three implementations use the same structure in [55] and are re-synthesized with our cell libraries and our synthesis tools.
[2]These two implementations use pipelined QRD structure shown in Fig. 4.9.
[3]The speed is measured by the number of matrices that can be processed in one second.
[4]The throughput is the highest data rate that can be achieved for a 64 QAM modulated 4×4 MIMO system.

4.4.3 Implementation Results and Comparison

It is often difficult to compare hardware implementation results due to various factors, such as cell libraries, software, and quantization schemes. Our hardware implementations serve two purposes. First, we would like to evaluate the overheads incurred by our ordering schemes. Second, we would like to verify the suitability of our ordering schemes for high data throughput applications. To this end, both non-pipelined and pipelined architectures of the BSQRD and SBSQRD ordering schemes are implemented in VHDL and synthesized in RTL compiler with a 45 nm library from Oklahoma State University [36] for a 4×4 MIMO system. The synthesis results are summarized in Tab. 4.2.

To evaluate the overheads incurred by our ordering schemes, we implement our BSQRD and SBSQRD ordering schemes as well as the sorted QRD algorithm based on the non-pipelined architecture in [55], using the distributed source code from [60], and compare them in the leftmost three columns of Tab. 4.2. Note that the **only difference** among these three implementations is their underlying ordering schemes. Compared with the sorted QRD algorithm, our ordering schemes achieve the same speed with roughly 10% greater gate count. Thus, the overheads incurred by our ordering schemes are negligible. We note that our ordering schemes reduce computational complexities of the SD, stack, and MCTS algorithms, and allow smaller K values for the K-Best algorithm as shown above. Since the complexity of the preprocessing is much smaller than that of the tree search, we conjecture that our ordering schemes would reduce the overall complexities of MIMO systems.

For easy reference, the implementation technology results in [53, 55] which are based on non-pipelined architectures and using 250 nm technology are also listed in

Tab. 4.2 because both implementations are dealing with the 4×4 MIMO system, the same system as our exemplary system. Since they are based on 250 nm cell libraries and/or different quantization schemes, their results are not directly comparable to our results. The implementation in [54] is designed for 8×8 MIMO systems, hence it is not comparable to ours.

Assuming $N_r \geq N_t$ and no channel coding is employed, the data throughput and speed of our ordering schemes have the following relation: data throughput = speed $\times N_t M_c$. For easy comparison, both metrics are provided in Tab. 4.2. Assuming a constellation of 64QAM, the data throughput of our non-pipelined architectures of our ordering schemes is at most 75 Mbit/s, which is much lower than the maximal data throughput of 600 Mbit/s specified in 802.11n. Thus, we also implement the pipelined architectures of our BSQRD and SBSQRD ordering schemes shown in Fig. 4.9, and present their results in the rightmost two columns of Tab. 4.2. Both implementations are for 4×4 MIMO systems, and are optimized at the same clock rate of 300 MHz. Since our pipelined architectures process one channel matrix in every five clock cycles, their processing speed is 60 million matrices per second. For a 64QAM modulated MIMO system, the data throughput of our implementations is 1.44 Gbit/s, much higher than the maximal data throughput specified in 802.11n. Unfortunately, the price of high speed and data throughput is greater gate counts due to pipeline registers. Finally, trade-offs can be easily made between hardware complexity and throughput. For applications that do not require very high throughput, the gate counts can be reduced by reducing the number of pipeline stages, or even using the non-pipelined architectures.

4.5 Summary

In this chapter, we apply the novel ordering schemes proposed in our previous work [13, 52] to the K-Best detector, and show that the performance of the K-Best detector is improved by our BSQRD ordering scheme and the V-BLAST versions of our BSQRD and SBSQRD ordering schemes. These ordering schemes lead to smaller detection error rates, reduce the required SNR, or allow even smaller values for K. The SBSQRD ordering scheme does not seem to improve the performance of the K-Best detector. We also implement both non-pipelined and pipelined architectures for our BSQRD and SBSQRD ordering schemes. Our implementation results show that our BSQRD and SBSQRD ordering schemes incur roughly 10% overheads in hardware implementations, and that their implementations can achieve high throughput at the expense of greater gate counts. Finally, trade-offs can be easily made between hardware complexity and throughput.

Chapter 5

List Based Soft-Decision MIMO Detection by the MCTS Algorithm

5.1 Introduction

In the previous chapter, we focus on the effects of different ordering schemes on the detection error rate of a K-Best detector. In this chapter, we are going to explore the computational complexity and memory requirement of MIMO detection algorithm. The maximum diversity gain of a MIMO communication system can be achieved by the maximum likelihood (ML) detection, but it has an exponential complexity if exhaustive search strategy is employed. Converting the ML detection problem to a tree search problem can significantly reduce the computational complexity. Many tree search detection algorithms have been proposed, such as the sphere decoding (SD) algorithm [10, 11, 61] based on depth-first strategy, the stack algorithm [42] based on best-first strategy, the K-Best algorithm [43] based on breadth-first strategy, and the memory constraint tree search (MCTS) algorithm [45]. Since the K-Best algorithm is not guaranteed to find the ML estimate and hence results in BER performance degradation, we do not consider it further in this chapter. The other three tree search algorithms all lead to the ML estimate.

The aforementioned ML tree search algorithms all have advantages and disadvantages in hardware implementations. Two key metrics are the computational complexity, which is usually measured by the number of visited nodes [50], and the size of required memory, which is used to store temporary data so as to facilitate the tree search. The computational complexity of a tree search algorithm determines the throughput, area, and power of its hardware implementation. The computational complexity and memory requirement of a tree search algorithm vary, and both the average and the worst-case figures are important to hardware implementation. For example, the worst-case computational complexity determines the lowest instantaneous throughput and the worst-case memory requirement determines the memory size of hardware implementations. While the memory requirement of the SD algorithm is fixed and linear with the product of tree depth, it visits many nodes in the worst case, resulting in a high average computational complexity. On the other hand, the stack algorithm visits the fewest nodes among all ML tree search algorithms [12], but the memory requirement of the stack algorithm in the worst case is much higher than the average requirement. Since the hardware implementation has to meet the memory requirement in the worst case, the stack algorithm is infeasible when higher order modulation and more transmit antennas are employed. The

MCTS algorithm fills the gap between the SD and stack algorithms. It requires a fixed but tunable memory, and its computational complexity depends on its memory size. When the memory is set at the minimum, the MCTS algorithm visits slightly fewer nodes than the SD algorithm while requiring slightly less memory. When the memory of the MCTS algorithm increases, its average number of visited nodes decreases and approaches that of the stack algorithm while requiring much less memory than the latter.

The result of the MIMO detection algorithm is the constellation point with the maximum likelihood, which can be viewed as a hard decision detector. However, For coded MIMO systems, the generation of soft information by soft-decision MIMO detection is very important. Properly designed soft-decision MIMO detection and soft-input and soft-output channel decoder can achieve near-capacity on a multiple-antenna channel [14]. List based soft-decision MIMO detection is an efficient way to generate reliable soft information. The soft information is generated based on a candidate list created by the list sphere decoding (LSD) detector [14], list sequential (LISS) detector [15], or other list creation algorithms. The LSD and LISS detectors are extensions of the SD algorithm and the stack algorithm, respectively. The main contribution of this chapter is a list-based soft-decision MIMO detector based on the MCTS algorithm. Our list MCTS (LMCTS) algorithm achieves a better tradeoff between memory requirement and computational complexity, and this tradeoff can be tuned through the number of the available memory units. When its memory is set at the minimum, our LMCTS algorithm achieves smaller computational complexity than the LSD algorithm while requiring slightly less memory. When the memory of our LMCTS algorithm increases, its computational complexity decreases and approaches that of the LISS detector. Given its fixed memory requirement and low computational complexity, our LMCTS algorithm is suitable for hardware implementations.

The rest of this chapter is organized as follows. Sec. 5.2 briefly describes the MCTS algorithm and the list based soft information generation for MIMO communication systems. Sec. 5.3 proposes our LMCTS algorithm and analyzes its memory requirement. Our simulation and implementation results are presented in Sec. 5.4.

5.2 Background Review

5.2.1 MCTS Algorithm

It has been shown in Chapter 4 that we can reformulate the ML MIMO detection problem into a tree search problem. While the SD algorithm based on the depth-first tree search strategy [10, 11, 61] has a fixed memory requirement and a high computational complexity, the stack algorithm based on the best-first strategy [42] has the smallest computational complexity but requires a lot of memory in the worst case. The MCTS algorithm [45] fills the gap between the SD and stack algorithm by alternating between the best-first and depth-first strategies. It determines at each iteration which node should be visited based on the number of the memory units currently available, and it always visits the node with the smallest metric while not exceeding the memory requirement. Thus it behaves like the stack algorithm when there is sufficient memory, and like the SD algorithm when the available memory is limited.

In one iteration, if the algorithm wants to visit a node at level k, $2 \leq k \leq N_t$, the number of the currently available memory units must satisfy $N \geq (k-2)(|\Omega|-1)$ [45]. Thus given the current available units number N, the maximum tree level in which a node can be visited is given by $\tilde{k} = \min\left\{\lfloor\frac{N}{|\Omega|-1}\rfloor + 2, N_t\right\}$. Therefore the node to be visited is the node $\mathbf{s}^k$ with the smallest metric in the memory and $k \leq \tilde{k}$. As shown by theoretical analysis and simulation results in [45], the MCTS algorithm provides a balance between the computational complexity and the memory requirement, and the balance can be easily tuned by the memory size for the MCTS algorithm.

5.2.2 List Based Soft Decision Detection

In a coded MIMO system based on the turbo principle [62], the transmitted signal $\mathbf{s}$ is mapped from a binary vector $\mathbf{x} = (x_1, x_2, \cdots, x_k)^T \in \mathcal{X}$, where $x_i \in \{1, -1\}$, and $\mathcal{X} = \{1, -1\}^k$, i.e., $\mathbf{s} = \mathit{map}(\mathbf{x})$. Then the max-log approximation of the extrinsic likelihood value of a bit x_i is given by [14]

$$\begin{aligned} L_E(x_i|\mathbf{r}) \approx &\frac{1}{2}\max_{\mathbf{x}\in\mathcal{X}_{i,+1}}\left\{-\frac{1}{\sigma^2}||\mathbf{y}-\mathbf{H}\mathbf{s}||^2 + \mathbf{x}_{[i]}\mathbf{L}_{A,[i]}\right\} \\ &- \frac{1}{2}\max_{\mathbf{x}\in\mathcal{X}_{i,-1}}\left\{\frac{1}{\sigma^2}||\mathbf{y}-\mathbf{H}\mathbf{s}||^2 + \mathbf{x}_{[i]}\mathbf{L}_{A,[i]}\right\}, \end{aligned} \tag{5.1}$$

where $\mathcal{X}_{i,+1}$ and $\mathcal{X}_{i,-1}$ are the sets of the vectors with $x_i = 1$ and $x_i = -1$, respectively. $\mathbf{x}_{[i]}$ is the sub-vector of $\mathbf{x}$ consisting of all the elements in $\mathbf{x}$ but x_i, and $\mathbf{L}_{A,[i]}$ is a vector consisting of the corresponding *a priori* likelihood ratios of the elements in $\mathbf{x}_{[i]}$. However, when higher order modulation and more transmit and receive antennas are used, the cardinalities of $\mathcal{X}_{i,+1}$ and $\mathcal{X}_{i,-1}$ become very large and calculating $L_E(x_i|\mathbf{r})$ become computationally intensive. Thus we may choose a subset $\mathcal{L} \subset \mathcal{X}$ which contains the maximizers in (5.1) with high probability, and the approximate extrinsic values is thus given by

$$\begin{aligned} L_E(x_i|\mathbf{r}) \approx \frac{1}{2} \max_{\mathbf{x}\in\mathcal{L}\cap\mathcal{X}_{i,+1}} \left\{ -\frac{1}{\sigma^2}||\mathbf{y}-\mathbf{Hs}||^2 + \mathbf{x}_{[i]}\mathbf{L}_{A,[i]} \right\} \\ - \frac{1}{2} \max_{\mathbf{x}\in\mathcal{L}\cap\mathcal{X}_{i,-1}} \left\{ \frac{1}{\sigma^2}||\mathbf{y}-\mathbf{Hs}||^2 + \mathbf{x}_{[i]}\mathbf{L}_{A,[i]} \right\}. \end{aligned} \tag{5.2}$$

The size of the candidate list $\mathcal{L}$ is relatively small compared with the size of $\mathcal{X}$ while maintaining good performance.

The candidate list $\mathcal{L}$ contains the ML estimate and the other $|\mathcal{L}| - 1$ ML candidates with the smallest metric $||\mathbf{r} - \mathbf{Hs}||^2$ among all possible transmitted signals. By formulating the ML detection problem as a tree search problem, $\mathcal{L}$ can be found by using tree search techniques discussed below.

5.2.3 The LSD Algorithm and the LISS Algorithm

The LSD algorithm uses the depth-first strategy to search the tree and create the candidate list $\mathcal{L}$. An initial radius C_0 is set at the beginning, and only those nodes with metric $w(\mathbf{s}^k) < C_0$ are visited. A leaf node within the sphere may be found by solving $w(\mathbf{s}^k) < C_0$ successively from N_t to 1. If $\mathcal{L}$ is full, remove the node with the largest metric from $\mathcal{L}$, and add the new leaf node into it. The radius C_0 is reduced to the largest metric of the nodes in $\mathcal{L}$, and the tree is pruned. If $\mathcal{L}$ is not full, add the leaf node to $\mathcal{L}$ directly. This process is repeated until there are no leaf nodes within the sphere. If the initial radius C_0 is too small, the list $\mathcal{L}$ may be not full or even empty. In this case, we increase C_0 and then restart the tree search. Similar to the SD algorithm, the LSD algorithm has fixed memory requirement but it also has a higher average complexity than the LISS algorithm.

The LISS algorithm uses the best-first strategy to search the tree and create the candidate list. At the beginning, the candidate list is set to be empty, and the root node is added into the stack with metric 0. In each iteration, the node with the smallest metric in the stack is removed from the stack. If this removed node is a leaf node, then add this node to the candidate list; otherwise add its children to the stack with their metrics. The algorithm ends when we have $|\mathcal{L}|$ nodes in the list, and $|\mathcal{L}|$ is a predefined number set at the beginning of the algorithm. The LISS algorithm has smaller computational complexity than the LSD algorithm, but its variable memory requirement is a significant drawback. As opposed to the LSD algorithm, the LISS algorithm proposed in [15] has two additional features to overcome this drawback. First, in each iteration, the stack is sorted with the node metrics in ascending order, and the nodes with indexes exceeding a predefined number $L_{\max}$ are discarded. Note that this modified search no longer guarantees to produce the candidates with the smallest metrics, and the ML estimate may not even exist in $\mathcal{L}$. Second, the reliability of the soft output is improved by augmenting the stack entries, and the BER performance is thus improved [15].

5.3 The List MCTS Algorithm

5.3.1 Generic Algorithm for List Generation

The key difference between the LSD algorithm and the LISS algorithm is the strategy they use to decide which node should be visited. The LSD algorithm uses the depth-first strategy while the LISS algorithm uses the best-first strategy, and the performances of these two algorithms are determined by these strategies. Hence we can summarize the LSD and LISS algorithms in a generic framework, which is embedded in the previous works [14,15,62] and stated explicitly in Alg. 4. The input of this algorithm is the receive vector $\mathbf{r}$, the upper triangular matrix $\mathbf{R}$ derived from the QR decomposition of the channel matrix $\mathbf{H}$, the candidate list size L. We also keep track of T, the number of non-leaf nodes saved in the memory which has been determined to be within the sphere of radius C_0. At the end of the algorithm, a candidate list $\mathcal{L}$ consisting of L ML candidates as well as their metric is returned.

Algorithm 4 Generic List Detection Algorithm

```
Input: $\mathbf{r}$, $\mathbf{R}$, $L$
 1: Set the initial radius $C_0$, list length $l = 0$, the size of the tree $T = 0$.
 2: Add the root node $\mathbf{s}^{N_t+1}$ to the memory. Now the tree in the memory has one
    node $\mathbf{s}^{N_t+1}$. Set $T = T + 1$.
 3: while $T > 0$ do
 4:     choose a node $\tilde{\mathbf{s}}^k$ to visit according to the tree search strategy.
 5:     Find the $K$ children of $\tilde{\mathbf{s}}^k$ within the sphere of radius $C_0$ and remove $\mathbf{s}^k$ from
        the memory. Set $T = T - 1$.
 6:     if $k > 2$ then
 7:         Add the $K$ children of $\tilde{\mathbf{s}}^k$ to the memory. Set $T = T + K$.
 8:     else
 9:         if $l + K \leq L$ then
10:             Add the $K$ children of $\tilde{\mathbf{s}}^k$ to the candidate list, and set $l = l + K$.
11:         else
12:             Set $l = L$. Find $L$ nodes with the smallest metric out of the $K$ children
                and the nodes in the candidate list. Update the candidate list with these
                $L$ nodes and there weights.
13:         end if
14:         if $l = L$ and $\max\{w_i\}_{i=1}^{L} < C_0$ then
15:             set $C_0 = \max\{w_i\}_{i=1}^{L}$, prune the tree and update the memory as well as
                the tree size $T$.
16:         end if
17:     end if
18: end while
19: Output: $\mathcal{L} = \{\tilde{\mathbf{s}}_i\}_{i=1}^{L}$, $\{w_i\}_{i=1}^{L}$.
```

Our LMCTS algorithm is derived by using the tree search strategy in the MCTS algorithm, i.e., replacing line 4 of Alg. 4 by $\tilde{\mathbf{s}}^k = \text{MCTSS}(\{\mathbf{s}\}, N)$, where MCTSS is the memory constraint tree search strategy generalized from [45] and stated in Alg. 5. Note that $\{\mathbf{s}\}$ is the current nodes of the tree in the memory, and N is the total memory available for the tree search. How to calculated N is shown in Sec. 5.3.2.

Algorithm 5 Memory Constraint Tree Search Strategy [45]

```
Input: $\{\mathbf{s}\}$, $N$.
1: Suppose the nodes in the memory span $D$ distinct levels: $2 \le k_1, k_2, \cdots, k_D \le N_t$. Find the best nodes $\tilde{\mathbf{s}}^{k_j}$ of all nodes in the $k_i$-th level according their weights.
2: Sort these best nodes in increasing order, assuming the sorted result is $w(\tilde{\mathbf{s}}^{k_1}) \le w(\tilde{\mathbf{s}}^{k_2}) \le \cdots \le w(\tilde{\mathbf{s}}^{k_D})$ without loss of generality.
3: Find the maximum tree level $\tilde{k} = \min\left\{\lfloor \frac{N}{|\Omega|-1} \rfloor + 2, N_t\right\}$
4: Let $\tilde{\mathbf{s}}^{k_j}$, $1 \le j \le D$, be the first nodes in the sorted best nodes that satisfies $k_j \le \tilde{k}$
5: Output: $\tilde{\mathbf{s}}^k$
```

5.3.2 Minimum Memory Requirement of LMCTS

The memory used by the LMCTS algorithm can be separated into two parts: the candidate list memory containing M_1 memory units and the tree search memory containing M_2 memory units. Clearly the first part should have $M_1 \ge L$ memory units. It is proved in [45] that the minimum memory required to extend a tree from the k-th level to level 2 is $(k-2)(|\Omega|-1)$, and hence it requires at least $(N_t-1)(|\Omega|-1)$ memory units to extend the tree from the root node to level 2. Therefore, we need $M_2 \ge (N_t-1)(|\Omega|-1)+1$ memory units to save the tree nodes to be visited. Thus with $(N_t-1)(|\Omega|-1)+1+L$ memory units, our LMCTS algorithm can create the candidate list successfully.

However, note that in the MCTS algorithm, one memory unit is used to save the best leaf node that has been found. Since this leaf node can be saved in the candidate list memory instead in our LMCTS algorithm, one memory unit can be saved from the tree search memory. We can use the following strategy: when the candidate list is empty, use the tree search memory and one memory unit from the candidate list memory to search the tree, and in line 4 the number of available memory units N is set to M_2+1-T; when the candidate list is non-empty, use the tree search memory only, and N is thus set to M_2-T. Therefore, when the total memory units number $M \geq (N_t-1)(|\Omega|-1)+L$, our LMCTS algorithm can find all L ML candidates. Note that when $L=1$, which means we only want to find the ML estimate, our LMCTS algorithm degenerates to the MCTS algorithm, and they have the same memory requirements.

5.4 Simulation and Implementation Results

5.4.1 Simulation Results

In this section, we compare the computational complexities as well as the memory requirements of the LSD, the LISS, and the LMCTS algorithms. As we are interested in the performance of ML candidate list generation, we implement a simplified LISS algorithm without the two features mentioned in Sec. 5.2.3. Therefore the memory requirement of the LISS algorithm is given by the worst case in our simulation. Since all these three algorithms generate the same candidate list, they have the same BER performance, and hence their BER performances are omitted. All these algorithms are implemented in the framework of Alg. 4. Our simulation results are based on the system model described in Sec. 4.2 and obtained over 2×10^5 channel realizations for each SNR value. We use the sorted QRD ordering [48] to reorder the columns of $\mathbf{H}$ to speed up the tree search convergence.

Our simulation results are summarized in Tables 5.1. N_c is the number of visited nodes, i.e., the number of the executions of line 4 in Alg. 4, and M is the number of memory units used in the algorithm. We use N_c as a measure of the computational complexities of these algorithms, which provides a good estimate of the throughput of the MIMO detector in hardware implementation [50]. We simulate the LSD, LISS, and LMCTS algorithms with different memory configuration for 4×4 QPSK, 4×4 16QAM, and 8×8 QPSK modulated MIMO systems. Note that the maximum computational complexity and memory requirement are only those observed among 2×10^5 channel realizations. Since both the LSD and LMCTS algorithms have fixed memory requirements, the maximum and average memory usages are equal for the LSD and LMCTS algorithms in the same simulation setting. The memory size required by the LSD algorithm is given by $(N_t - 1)|\Omega| + L$. Since the performance of our LMCTS algorithm can be tuned through the memory constraint, we provide two or three memory sizes of the LMCTS algorithm in each simulation setting. One of the memory sizes is the minimum size, $(N_t - 1)(|\Omega| - 1) + L$, of our LMCTS algorithm, and the others are larger than the minimum. Since we wish to compare the performance of finding the ML candidates of the received vector, we set the size of the memory used by the simplified LISS algorithm to be large enough to avoid discarding nodes. The initial radius C_0 is set large enough for all three algorithms to avoid adjusting the initial radius of the search sphere.

Table 5.1: Complexities and memory requirement of different list detection algorithms for various MIMO systems.

Modulation		4×4 QPSK, $L = 8$				4×4 16QAM, $L = 96$				8×8 QPSK, $L = 32$			
SNR		10dB		20dB		5dB		15dB		5dB		15dB	
		N_c	M	N_c	M	N_c	M	N_c	M	N_c	M	N_c	M
LSD	Max	85	20	85	20	3726	144	2555	144	9577	60	3790	60
	Avg	22.70	20	20.75	20	228.39	144	157.69	144	499.92	60	282.86	60
LMCTS	Max	82	17	75	17	3639	141	2372	141	9427	53	3751	53
	Avg	20.76	17	19.56	17	201.39	141	135.58	141	474.73	53	269.61	53
	Max	77	23	71	23	3411	191	2056	191	7002	95	1661	95
	Avg	19.52	23	19.10	23	160.55	191	114.11	191	324.49	95	221.32	95
	Max	-	-	-	-	3347	255	1804	255	6517	151	1606	151
	Avg					153.28	255	112.57	255	308.78	151	219.20	151
LISS	Max	77	67	71	67	3156	4171	1716	2431	6138	4591	1482	1315
	Avg	19.50	25.51	19.08	25.53	150.44	393.10	112.22	308.30	305.92	350.00	218.97	250.71

We now compare the performance of our LMCTS algorithm with those of the LSD and simplified LISS algorithms. Our LMCTS algorithm has smaller average and maximum complexities than the LSD algorithm even if its memory size is set to the minimum requirement, which is slightly smaller than the memory requirement of the LSD algorithm. For example, for a 4×4 16QAM modulated system, when the SNR is 5dB and L=96, our LMCTS algorithm visits 201.39 nodes with 141 memory units on average, and the LSD algorithm visits 228.39 nodes with 144 memory units on average. Our LMCTS algorithm visits more nodes than the simplified LISS algorithm, but as the memory size increases, both the average and maximum numbers of the visited nodes of our LMCTS algorithm decrease and approach those of the simplified LISS algorithm. For example, for a 4×4 16QAM modulated system, when the SNR is 15dB and L=96, when the memory size of our LMCTS algorithm increases to 255, our LMCTS algorithm needs to visit 153.28 nodes on average, only about 2% more than that of the simplified LISS algorithm. Note that the memory requirement for the LISS algorithm is bounded by its worst case, it needs more than 4171 memory units, more than 16 times of the memory size of our LMCTS algorithm. Therefore our LMCTS algorithm and the simplified LISS algorithm have similar computational complexity, the memory required by the LMCTS is only a **fraction** of that for the simplified LISS algorithm in the worst case. From our simulation results, our LMCTS algorithm achieves a flexible tradeoff between the memory requirement and the computational complexity, and hence is more suitable for hardware implementations than the LSD and the LISS algorithms.

5.4.2 Implementation

To further justify the feasibility of our LMCTS algorithm, we implement the LMCTS algorithm in VHDL. Our design is divided into three modules. The first one is the metric calculation unit. The children node metrics of the visited nodes are calculated in the same architecture shown in [63], and then the K children with metrics less than C_0 are sent to the second module, memory unit. In this module, the input children nodes are first added to the tree search memory or the candidate list according to their levels as described in Alg. 4. Then the tree search memory is traversed to find the nodes with the smallest metric in each level and prune those with metrics larger than the radius C_0, which may be updated when new leaf nodes are found. The nodes with the smallest metric in each level and the number of available memory units are then sent to the third module, selection unit. The node with the smallest metric and without violating the memory constraint is selected and sent to the metric calculation unit, and a new iteration starts.

We synthesize our design for a 4×4 16QAM modulated MIMO system with RTL compiler and OKSU 45 nm technology [36] as a demonstration of our algorithm. The available tree search memory unit is set to 160 and the list length is 5. The cell area is 0.355 mm^2, and the equivalent gate count is 189123. Our detector can achieve a maximum clock speed of 116.6 MHz. In the worst case of an iteration, we have to sort the metrics of the founded nodes and traverse the tree search memory. To reduce the complexity of sorting, we can first sort the metrics of the leaf nodes just found, and then merge this sorted list with the previous ordered candidate list to find the L smallest candidates. As our design sorts the list and traverses the memory sequentially, an iteration needs at most $|\Omega|^2/2 + L + M_2$ clock

Table 5.2: Comparison of implementations of list MIMO detectors.

	Ours	[64]	[65]
Technology	45 nm	0.25 μm	0.25 μm
MIMO system	4×4	4×4	4×4
Modulation	16QAM	16QAM	16QAM
Equivalent gate count	190 k	205 k	95 k
Clock rate (MHz)	116.6	67.8	170
Throughput @ 20dB (Mvec/s)	0.0128	0.425	5

cycles, where $|\Omega|$ is the constellation size, and M_2 is the tree search memory size. Therefore our implementation can output the decoding result of a received vector in about $N_c(|\Omega|^2/2 + L + M_2)$ clock cycles. Note that N_c is decreased when we increase M_2, there is an optimal size of M_2 that achieves the largest throughput. The simulation shows that our LMCTS detector has a throughput of 0.0128 Mvec/s when SNR=20dB.

We summarize our results and other results from literature in Tab. 5.2. Comparing with [64], their detector is highly parallelized while our LMCTS detector needs many cycles to process one node since our design works in a sequential way, and thus our throughput is much lower. However, our throughput may be greatly increased by parallelizing our LMCTS detector or using a number of LMCTS detectors working together. For example, sorting the leaf nodes and traversing the tree search memory can both be done in one clock cycle with extra logic, as what has been done in [64]. In [65], the LSD algorithm is simplified. While processing one node per clock cycle, the detector only add the best leaf node of one node at level 2, and all of its sibling nodes are discard, even if they have a smaller metric than the nodes in the list. With the simplification, the clock speed is increased and area is reduced. However, the ML property of the list no longer holds. Our design can also adopt this simplification if needed.

5.5 Summary

We propose a list-based soft-decision MIMO detection algorithm to find a list of ML candidates based on the MCTS strategy. Our simulation results show that our LMCTS algorithm achieves a flexible balance by tuning its available memory size between the memory requirement and the computational complexity. The feasibility of our LMCTS algorithm is shown by our implementation results. Since our demonstration design works sequentially, it does not have an attractive throughput. However, with more sophisticated technique (e.g. [63, 64]) and an expense of more area consumption and proper BER performance degradation, our LMCTS algorithm has a potentiality to achieve a high throughput with the flexible balance between memory requirement and computational complexity.

Chapter 6

On Algorithms and Complexities of Cyclotomic Fast Fourier Transforms over Arbitrary Finite Fields

6.1 Introduction

Discrete Fourier transforms (DFTs) over finite fields [25] have been widely used in cryptography and error control codes. However, direct implementation of an n-point DFT requires $O(n^2)$ multiplications and additions, and this complexity becomes prohibitive for very large n. Recently, Reed-Solomon codes over GF(2^{12}) with thousands of symbols are considered for hard drive [16] and tape storage [17] as well as optical communication systems [66] to achieve a lower error rate; the syndrome based decoders (e.g., [67]) of such codes require DFTs of lengths up to 4095 over GF(2^{12}). Elliptic curve cryptosystems involve multiplications over finite fields up to GF(2^{571}) [68, 69], which can also be implemented as polynomial multiplications and hence be implemented by DFTs via the convolution theorem [25]. Therefore efficient DFT algorithms are required in practice. Recently proposed cyclotomic fast Fourier transforms (CFFTs) [20, 21, 70] have attracted a lot of attention due to their low multiplicative complexities.

One open problem for CFFTs is that all of the existing CFFTs [20, 21, 70] are for characteristic-2 finite fields, while it is necessary to generalize CFFTs to arbitrary finite fields. This is because recently non-characteristic-2 fields such as GF(3^m) have been considered in modern error control codes [18] and cryptosystems [19]. The low multiplicative complexities of CFFTs are primarily due to the efficient short cyclic convolutions used to construct CFFTs. Thus, a key challenge in generalizing CFFTs to arbitrary finite fields is efficient cyclic convolution algorithms over any finite field. To the best of our knowledge, there is no efficient general algorithm for cyclic convolutions over arbitrary finite fields other than the implementation via the convolution theorem [25]. Unfortunately, this approach is not efficient for short cyclic convolutions and hence not suitable for CFFTs.

Another issue regarding CFFTs is that their computational complexities have not been carefully examined. Although their advantages in both multiplicative and overall complexities have been demonstrated for DFTs with short and moderate lengths (see, for example, [22]), it is unknown whether these advantages hold for very long DFTs. This question becomes particularly important, as applications require DFTs of increasingly longer lengths. Furthermore, it is also interesting to compare the complexities of CFFTs to other existing DFT algorithms [71–74]. Such comparison will help system designers to select appropriate long-DFT algorithms with minimal complexities.

In this chapter, we address both issues mentioned above. To this end, our main contributions are as follows:

- We propose an efficient bilinear algorithm to compute Toeplitz matrix vector products (TMVPs). It works on all finite fields as well as the real and complex fields, and has a smaller computational complexity than existing TMVP algorithms, such as that in [75]. This TMVP algorithm not only enables us to devise efficient algorithms for cyclic convolutions and CFFTs over arbitrary finite fields, but also is instrumental in our analysis of the computational complexities of CFFTs. Moreover, TMVPs are important in themselves due to their many applications, such as low-complexity finite field multiplications [75].

- We propose an efficient algorithm for cyclic convolutions with prime lengths over arbitrary finite fields. The cyclic convolutions are first reformulated as the product of a Toeplitz matrix and a vector at the expense of only one extra multiplication, and then we can use the efficient TMVP algorithm we propose to derive the cyclic convolution results. The algorithm for cyclic convolutions with arbitrary lengths can be derived through multidimensional technology. This efficient cyclic convolution algorithm enables us to extend CFFTs to arbitrary fields with low computational complexities.

- We derive the bounds on both the multiplicative and additive complexities of CFFTs over arbitrary finite fields. The multiplicative complexities of CFFTs are lower than all known algorithms, but their additive complexities are much higher, rendering them asymptotically suboptimal. However, CFFTs are still of practical value for DFTs with up to thousands of symbols.

The rest of this chapter is organized as follows. We first propose an efficient algorithm for TMVPs in Sec. 6.2. Then we propose an algorithm for cyclic convolutions and CFFTs over an arbitrary finite field in Sec. 6.3. The multiplicative and additive complexities of CFFTs over arbitrary finite fields are derived in Sec. 6.4 and this chapter concludes in Sec. 6.5.

6.2 An Efficient Algorithm for Toeplitz Matrix Vector Product

An $n \times n$ matrix $\mathbf{T}$ is called a Toeplitz matrix when each of its diagonals is constant, that is, $T_{i,j} = t_{i-j}$. Hence $\mathbf{T}$ is also represented by a corresponding vector $\mathbf{t} = (t_{-n+1}, t_{-n+2}, \cdots, t_{n-1})^T$. The product between $\mathbf{T}$ and an n-dimensional vector $\mathbf{v}$,

$$\mathbf{u} = \mathbf{T}\mathbf{v} = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{n-1} \end{bmatrix} = \begin{bmatrix} t_0 & t_{-1} & \cdots & t_{-n+1} \\ t_1 & t_0 & \cdots & t_{-n+2} \\ \vdots & \vdots & \ddots & \vdots \\ t_{n-1} & t_{n-2} & \cdots & t_0 \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_{n-1} \end{bmatrix}, \tag{6.1}$$

is referred to as an $n \times n$ Toeplitz matrix vector product (TMVP).

6.2.1 Background and Motivation

Efficient algorithms for cyclic convolutions of short lengths are essential to achieving CFFTs of low multiplicative complexities. We will show in Sec. 6.3 that cyclic convolutions over finite fields can be formulated as TMVPs. Hence efficient TMVPs are important to our cyclic convolutions and CFFTs over finite fields in Sec. 6.3 and to the complexity analysis of CFFTs in Sec. 6.4. We present our efficient algorithm for TMVPs separately in this section for two reasons. First, while cyclic convolutions and CFFTs considered in Secs. 6.3 and 6.4 are over finite fields, our algorithm for TMVPs applies to finite fields as well as the real and complex fields. Second and more importantly, efficient algorithms for TMVPs are important in themselves due to other applications of TMVPs besides their relation to cyclic convolutions. For instance, efficient algorithms for TMVPs are used to devise low-complexity finite field multiplications [75].

Efficient algorithms for TMVPs are often derived using multidimensional technologies, which decompose long TMVPs into short TMVPs with efficient algorithms. If $n = n_1 n_2$, an $n \times n$ TMVP can be decomposed into $n_1 \times n_1$ and $n_2 \times n_2$ TMVPs, and n_1 and n_2 can be further decomposed. When n is not a composite number, both $n+1$ and $n-1$ are composite and hence two ad-hoc techniques are often used (see, for example, [75]): 1) one can obtain an $(n+1)\times(n+1)$ TMVP by padding a zero at the end of the vector and extending the $n \times n$ Toeplitz matrix to an $(n+1)\times(n+1)$ one by setting $T_{0,n} = T_{n,0} = 0$, and then apply the multidimensional techniques; 2) one can first obtain an $(n-1)\times(n-1)$ TMVP by ignoring the first row and the last column of the $n \times n$ Toeplitz matrix as well as the last element of the vector, and then apply the multidimensional techniques to $(n-1)\times(n-1)$ TMVP and account for the ignored parts separately. In practice, the most efficient algorithm is selected among these ad-hoc approaches.

In this section, however, based on a systematic approach, we propose a bilinear algorithm of $n \times n$ TMVPs that is fixed for any given n. Its computational complexity can be expressed explicitly, which is useful for our complexity analysis of CFFTs later. Furthermore, our algorithm achieves a smaller computational complexity than existing TMVP algorithms to the best of our knowledge.

6.2.2 A Bilinear TMVP Algorithm

We compute the $n \times n$ TMVP in (6.1) by a bilinear algorithm, i.e., $\mathbf{u} = \mathbf{E}^{(n)}(\mathbf{G}^{(n)}\mathbf{t} \cdot \mathbf{H}^{(n)}\mathbf{v})$, where $\cdot$ denotes an entry-wise multiplication between two vectors, $\mathbf{E}^{(n)}$ is an $n \times M(n)$ matrix, $\mathbf{G}^{(n)}$ is an $M(n) \times (2n-1)$ one, and $\mathbf{H}^{(n)}$ is an $M(n) \times n$ one. $M(n)$ denotes the number of columns of $\mathbf{E}^{(n)}$, which is the same as the number of rows of $\mathbf{G}^{(n)}$ and $\mathbf{H}^{(n)}$. We will show that $M(n)$ is in fact the number of multiplications required by our bilinear algorithm later.

We will first construct the matrices $\mathbf{E}^{(n)}$, $\mathbf{G}^{(n)}$, and $\mathbf{H}^{(n)}$ for the bilinear algorithm inductively, then show that our bilinear algorithm indeed computes the $n \times n$ TMVP in (6.1), and finally derive the additive and multiplicative complexities of our bilinear algorithm.

The 1×1 TMVP is just a multiplication between t_0 and v_0, and hence $\mathbf{E}^{(1)} = \mathbf{G}^{(1)} = \mathbf{H}^{(1)} = 1$.

Assume that the matrices $\mathbf{E}^{(k)}$, $\mathbf{G}^{(k)}$, and $\mathbf{H}^{(k)}$ are known. When $n = 2k$, we partition the Toeplitz matrix $\mathbf{T}$ into four $k \times k$ matrices, and both $\mathbf{v}$ and $\mathbf{u}$ into two

k-dimensional vectors as

$$\begin{bmatrix} \mathbf{u}_0 \\ \mathbf{u}_1 \end{bmatrix} = \begin{bmatrix} \mathbf{T}_0 & \mathbf{T}_{-1} \\ \mathbf{T}_1 & \mathbf{T}_0 \end{bmatrix} \begin{bmatrix} \mathbf{v}_0 \\ \mathbf{v}_1 \end{bmatrix}, \tag{6.2}$$

then we can construct the bilinear algorithm with matrices

$$\begin{aligned} \mathbf{E}^{(2k)} &= \begin{bmatrix} \mathbf{E}^{(k)} & \mathbf{E}^{(k)} & \mathbf{0}_{k\times M(k)} \\ \mathbf{E}^{(k)} & \mathbf{0}_{k\times M(k)} & \mathbf{E}^{(k)} \end{bmatrix}, \\ \mathbf{G}^{(2k)} &= \begin{bmatrix} \mathbf{G}^{(k)}\mathbf{P}_0^{(k)} \\ \mathbf{G}^{(k)}(\mathbf{P}_{-1}^{(k)} - \mathbf{P}_0^{(k)}) \\ \mathbf{G}^{(k)}(\mathbf{P}_1^{(k)} - \mathbf{P}_0^{(k)}) \end{bmatrix}, \mathbf{H}^{(2k)} = \begin{bmatrix} \mathbf{H}^{(k)} & \mathbf{H}^{(k)} \\ \mathbf{0}_{M(k)\times k} & \mathbf{H}^{(k)} \\ \mathbf{H}^{(k)} & \mathbf{0}_{M(k)\times k} \end{bmatrix}, \end{aligned} \tag{6.3}$$

respectively, where $\mathbf{0}_{s\times t}$ is an $s\times t$ zero matrix, and the matrices $\mathbf{P}_{-1}^{(k)}$, $\mathbf{P}_0^{(k)}$, and $\mathbf{P}_1^{(k)}$ are all $(2k-1)\times(4k-1)$ ones, given by $[\mathbf{I}_{2k-1}\ \mathbf{0}_{(2k-1)\times 2k}]$, $[\mathbf{0}_{(2k-1)\times k}\ \mathbf{I}_{2k-1}\ \mathbf{0}_{(2k-1)\times k}]$, and $[\mathbf{0}_{(2k-1)\times 2k}\ \mathbf{I}_{2k-1}]$, respectively.

When $n = 2k+1$, we partition the TMVP in a symmetric way as

$$\begin{bmatrix} \mathbf{u}_0 \\ u_k \\ \mathbf{u}_1 \end{bmatrix} = \left[\begin{array}{c:c:c} \mathbf{T}_0 & \mathbf{t}_{-k}^{-1} & \mathbf{T}_{-1} \\ \hdashline (\mathbf{t}_k^1)^T & t_0 & (\mathbf{t}_{-1}^{-k})^T \\ \hdashline \mathbf{T}_1 & \mathbf{t}_1^k & \mathbf{T}_0 \end{array}\right] \begin{bmatrix} \mathbf{v}_0 \\ v_k \\ \mathbf{v}_1 \end{bmatrix}, \tag{6.4}$$

where $\mathbf{t}_a^b$ denotes the vector $(t_a, t_{a+1}, \cdots, t_b)^T$ if $a \le b$, or $(t_a, t_{a-1}, \cdots, t_b)^T$ if $a > b$.

Accordingly, our bilinear algorithm uses

$$\mathbf{E}^{(2k+1)} = \left[\begin{array}{c:c:c:c} \mathbf{E}^{(k)} & \mathbf{E}^{(k)} & & \mathbf{0}_{k\times M(k)} \\ \mathbf{0}_{1\times M(k)} & \mathbf{0}_{1\times M(k)} & \mathbf{D}_{2k+1} & \mathbf{0}_{1\times M(k)} \\ \mathbf{E}^{(k)} & \mathbf{0}_{k\times M(k)} & & \mathbf{E}^{(k)} \end{array}\right],$$

$$\mathbf{G}^{(2k+1)} = \left[\begin{array}{ccc} \multicolumn{3}{c}{\mathbf{G}^{(k)}\mathbf{Q}_0^{(k)}} \\ \hdashline \multicolumn{3}{c}{\mathbf{G}^{(k)}(\mathbf{Q}_{-1}^{(k)} - \mathbf{Q}_0^{(k)}) + \mathbf{g}_0^{(k)}} \\ \hdashline \mathbf{0}_{(2k+1)\times k} & \mathbf{D}_{2k+1} & \mathbf{0}_{(2k+1)\times k} \\ \hdashline \multicolumn{3}{c}{\mathbf{G}^{(k)}(\mathbf{Q}_1^{(k)} - \mathbf{Q}_0^{(k)}) + \mathbf{g}_1^{(k)}} \end{array}\right], \tag{6.5}$$

$$\mathbf{H}^{(2k+1)} = \left[\begin{array}{ccc} \mathbf{H}^{(k)} & \mathbf{0}_{M(k)\times 1} & \mathbf{H}^{(k)} \\ \hdashline \mathbf{0}_{M(k)\times k} & \mathbf{0}_{M(k)\times 1} & \mathbf{H}^{(k)} \\ \hdashline \multicolumn{3}{c}{\mathbf{D}_{2k+1}^T\mathbf{Y}_{2k+1}} \\ \hdashline \mathbf{H}^{(k)} & \mathbf{0}_{M(k)\times 1} & \mathbf{0}_{M(k)\times k} \end{array}\right],$$

where the matrices $\mathbf{Q}_{-1}^{(k)}$, $\mathbf{Q}_1^{(k)}$, and $\mathbf{Q}_0^{(k)}$ are all $(2k-1)\times(4k+1)$ ones and given by $[\mathbf{I}_{2k-1}\ \mathbf{0}_{(2k-1)\times(2k+2)}]$, $[\mathbf{0}_{(2k-1)\times(2k+2)}\ \mathbf{I}_{2k-1}]$, and $[\mathbf{0}_{(2k-1)\times(k+1)}\ \mathbf{I}_{2k-1}\ \mathbf{0}_{(2k-1)\times(k+1)}]$, respectively. The matrix $\mathbf{D}_{2k+1}$ is a $(2k+1)\times(2k+1)$ matrix given by

$$\mathbf{D}_{2k+1} = \begin{bmatrix} -\mathbf{I}_k & \mathbf{0}_{k\times 1} & \mathbf{0}_{k\times k} \\ \mathbf{1}_{1\times k} & 1 & \mathbf{1}_{1\times k} \\ \mathbf{0}_{k\times k} & \mathbf{0}_{k\times 1} & -\mathbf{I}_k \end{bmatrix}, \tag{6.6}$$

where $\mathbf{1}_{s\times t}$ is an $s\times t$ matrix with all entries one. The matrices $\mathbf{g}_0^{(k)}$ and $\mathbf{g}_1^{(k)}$ are both $M(k)\times(4k+1)$ matrices, and both depend on $\mathbf{H}^{(k)}$: if row i of $\mathbf{H}^{(k)}$ has **only one** non-zero element at column j, then the element at row i of column $2k-j-1$ in $\mathbf{g}_0^{(k)}$ and the element at row i of column $3k-j$ in $\mathbf{g}_1^{(k)}$ are set to one; all the other elements in $\mathbf{g}_0^{(k)}$ and $\mathbf{g}_1^{(k)}$ are zeros. Since $0 \le i \le M(k)-1$ and $0 \le j \le k-1$, the non-zero columns in $\mathbf{g}_0^{(k)}$ and $\mathbf{g}_1^{(k)}$ are from k to $2k-1$ and from $2k+1$ to $3k$, respectively. The matrix $\mathbf{Y}_s$ is an $s \times s$ anti-diagonal matrix with ones on its anti-diagonal.

Before showing that the matrices we construct in (6.3) and (6.5) induce a bilinear algorithm to compute the $n \times n$ TMVP in (6.1), we first establish some technical results about the matrices introduced in (6.3) and (6.5).

Lemma 6.1. $\mathbf{E}^{(n)} = (\mathbf{H}^{(n)}\mathbf{Y}_n)^T$*, and both* $\mathbf{E}^{(n)}$ *and* $\mathbf{H}^{(n)}$ *consist of only* 0*,* 1*, and* -1*.*

Proof. When $n = 1$, $\mathbf{E}^{(1)} = \mathbf{H}^{(1)} = \mathbf{Y}_1 = 1$, and the lemma is satisfied. Suppose the lemma holds for n up to k.

When $n = 2k$, note that we can partition $\mathbf{Y}_{2k}$ into

$$\mathbf{Y}_{2k} = \begin{bmatrix} \mathbf{0}_{k\times k} & \mathbf{Y}_k \\ \mathbf{Y}_k & \mathbf{0}_{k\times k} \end{bmatrix},$$

we have

$$\begin{aligned}(\mathbf{H}^{(2k)}\mathbf{Y}_{2k})^T &= \begin{bmatrix} (\mathbf{H}^{(k)}\mathbf{Y}_k)^T & (\mathbf{H}^{(k)}\mathbf{Y}_k)^T & \mathbf{0}_{k\times M(k)} \\ (\mathbf{H}^{(k)}\mathbf{Y}_k)^T & \mathbf{0}_{k\times M(k)} & (\mathbf{H}^{(k)}\mathbf{Y}_k)^T \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{E}^{(k)} & \mathbf{E}^{(k)} & \mathbf{0}_{k\times M(k)} \\ \mathbf{E}^{(k)} & \mathbf{0}_{k\times M(k)} & \mathbf{E}^{(k)} \end{bmatrix} = \mathbf{E}^{(2k)},\end{aligned}$$

and by the construction, $\mathbf{E}^{(2k)}$ and $\mathbf{H}^{(2k)}$ contain only 0, 1, and -1.

When $n = 2k+1$, We can partition the matrix $\mathbf{Y}_{2k+1}$ into

$$\mathbf{Y}_{2k+1} = \begin{bmatrix} \mathbf{0}_{k\times k} & \mathbf{0}_{k\times 1} & \mathbf{Y}_k \\ \mathbf{0}_{1\times k} & 1 & \mathbf{0}_{1\times k} \\ \mathbf{Y}_k & \mathbf{0}_{k\times 1} & \mathbf{0}_{k\times k} \end{bmatrix},$$

and the product of $(\mathbf{H}^{2k+1}\mathbf{Y}_{2k+1})^T$ can be computed as

$$\begin{aligned}
&\left[\begin{array}{c:c:c:c} (\mathbf{H}^{(k)}\mathbf{Y}_k)^T & (\mathbf{H}^{(k)}\mathbf{Y}_k)^T & & \mathbf{0}_{k\times M(k)} \\ \mathbf{0}_{1\times M(k)} & \mathbf{0}_{1\times M(k)} & (\mathbf{D}_{2k+1}^T\mathbf{Y}_{2k+1}^2)^T & \mathbf{0}_{1\times M(k)} \\ (\mathbf{H}^{(k)}\mathbf{Y}_k)^T & \mathbf{0}_{k\times M(k)} & & (\mathbf{H}^{(k)}\mathbf{Y}_k)^T \end{array}\right] \\
=&\left[\begin{array}{c:c:c:c} \mathbf{E}^{(k)} & \mathbf{E}^{(k)} & & \mathbf{0}_{k\times M(k)} \\ \mathbf{0}_{1\times M(k)} & \mathbf{0}_{1\times M(k)} & \mathbf{D}_{2k+1} & \mathbf{0}_{1\times M(k)} \\ \mathbf{E}^{(k)} & \mathbf{0}_{k\times M(k)} & & \mathbf{E}^{(k)} \end{array}\right] \\
=&\mathbf{E}^{(2k+1)}.
\end{aligned}$$

Since the matrix $\mathbf{D}_{2k+1}$ contains only 0, 1, and -1, $\mathbf{E}^{(2k+1)}$ and $\mathbf{H}^{(2k+1)}$ also contain only these three numbers. □

Lemma 6.2. $\mathbf{H}^{(n)}$ *does not contain any zero rows, and the rows of* $\mathbf{H}^{(n)}$ *that contain only one non-zero element form an anti-diagonal matrix* $\mathbf{Y}_n$.

Proof. We are going to prove this lemma by induction. When $n = 1$, $\mathbf{H}^{(1)} = 1$, and the lemma holds. Suppose the lemma holds for n up to k, which means removing the rows of $\mathbf{H}^{(k)}$ with more than one non-zero element results in an anti-diagonal matrix $\mathbf{Y}_k$.

When $n = 2k$, $\mathbf{H}^{(2k)}$ in (6.3) has no zero rows, and removing the rows containing more than one non-zero element results in a matrix given by

$$\begin{bmatrix} \mathbf{0}_{k\times k} & \mathbf{Y}_k \\ \mathbf{Y}_k & \mathbf{0}_{k\times k} \end{bmatrix} = \mathbf{Y}_{2k}.$$

When $n = 2k+1$, in the block $\mathbf{D}_{2k+1}^T \mathbf{Y}_{2k+1}$ in $\mathbf{H}^{(2k+1)}$ only row k contains one nonzero element, and there is no zero rows. Therefore, removing the rows with more than one non-zero element results in a matrix given by

$$\begin{bmatrix} \mathbf{0}_{k\times k} & \mathbf{0}_{k\times 1} & \mathbf{Y}_k \\ \mathbf{0}_{1\times k} & 1 & \mathbf{0}_{1\times k} \\ \mathbf{Y}_k & \mathbf{0}_{k\times 1} & \mathbf{0}_{k\times k} \end{bmatrix} = \mathbf{Y}_{2k+1}.$$

□

Corollary 6.3. $\mathbf{E}^{(n)}$ *contains no zero columns, and removing all the columns containing more than one non-zero element in* $\mathbf{E}^{(n)}$ *results in the identity matrix* $\mathbf{I}_n$*.*

Proof. By Lemma 6.1, the number of the non-zero elements in a column in $\mathbf{E}^{(n)}$ is the same with the number of the non-zero elements in the corresponding row in $\mathbf{H}^{(n)}$, and removing columns containing more than one non-zero element results in a matrix $(\mathbf{Y}_n\mathbf{Y}_n)^T = \mathbf{I}_n$. □

Corollary 6.4. *Removing both the zero rows and zero columns of either* $\mathbf{g}_0^{(k)}$ *or* $\mathbf{g}_1^{(k)}$ *in* (6.5) *results in* $\mathbf{I}_k$*.*

Proof. We prove this corollary for $\mathbf{g}_0^{(k)}$ first. The zero rows in $\mathbf{g}_0^{(k)}$ correspond to the rows in $\mathbf{H}^{(k)}$ containing more than one non-zero element. Removing these rows from $\mathbf{g}_0^{(k)}$ and $\mathbf{H}^{(k)}$ result in a $k \times (4k-1)$ matrix $\mathbf{g}_0^{(k)'}$ and $\mathbf{Y}_k$, respectively. By the construction of $\mathbf{g}_0^{(k)}$, we know that in row j of $\mathbf{g}_0^{(k)'}$, the element at position $k+j$ is one, and other elements are zeros. Therefore, the non-zero columns are from k to $2k-1$, and removing the zero columns results in an identity matrix $\mathbf{I}_k$. The corollary also holds for $\mathbf{g}_1^{(k)}$, which can be shown in a similar way. □

Proposition 6.5. *The bilinear algorithm induced by* $\mathbf{E}^{(n)}$*,* $\mathbf{G}^{(n)}$*, and* $\mathbf{H}^{(n)}$ *in* (6.3) *and* (6.5) *computes the* $n \times n$ *TMVP in* (6.1).

Proof. We are going to prove this proposition by induction. When $n = 1$, $\mathbf{E}^{(1)} = \mathbf{H}^{(1)} = \mathbf{G}^{(1)} = 1$, and the proposition holds. Assume that the proposition holds for n up to k. When $n = 2k$, we can check by inspection that the bilinear algorithm given by $\mathbf{E}^{(2k)}$, $\mathbf{G}^{(2k)}$, and $\mathbf{H}^{(2k)}$ in (6.3) is actually the well-known two way split method in [76],

$$\begin{aligned}\mathbf{u}_0 &= \mathbf{T}_0(\mathbf{v}_0+\mathbf{v}_1) + (\mathbf{T}_{-1}-\mathbf{T}_0)\mathbf{v}_1,\\ \mathbf{u}_1 &= \mathbf{T}_0(\mathbf{v}_0+\mathbf{v}_1) + (\mathbf{T}_1-\mathbf{T}_0)\mathbf{v}_0,\end{aligned} \tag{6.7}$$

which computes the $2k \times 2k$ TMVP.

When $n = 2k+1$, note the structure of $\mathbf{D}_{2k+1}$ in (6.6) and that $\mathbf{Q}_0^{(k)}\mathbf{t}$ and $\mathbf{Q}_{-1}^{(k)}\mathbf{t}$ correspond to $\mathbf{T}_0$ and $\mathbf{T}_{-1}$, respectively, the vector $\mathbf{u}_0$ as partitioned in (6.4) can be computed as

$$\begin{aligned}\mathbf{u}_0 = {} & \mathbf{T}_0(\mathbf{v}_0+\mathbf{v}_1) + (\mathbf{T}_{-1}-\mathbf{T}_0)\mathbf{v}_1\\ & + \mathbf{E}^{(k)}(\mathbf{g}_0^{(k)}\mathbf{t} \cdot \mathbf{H}^{(k)}\mathbf{v}_1) + \mathbf{t}_{-k}^{-1} \cdot (v_k \mathbf{1}_{k\times 1} - \mathbf{Y}_k\mathbf{v}_1).\end{aligned}$$

Since the zero rows of $\mathbf{g}_0^{(k)}$ correspond to the rows with more than one non-zero element in $\mathbf{H}^{(k)}$ as well as the columns with more than one non-zero element in $\mathbf{E}^{(k)}$ and since the non-zero columns of $\mathbf{g}_0^{(k)}$ are from column k to column $2k-1$, we have

$$\mathbf{E}^{(k)}(\mathbf{g}_0^{(k)}\mathbf{t} \cdot \mathbf{H}^{(k)}\mathbf{v}_1) = \mathbf{t}_{-k}^{-1} \cdot \mathbf{Y}_k \mathbf{v}_1,$$

and then we have $\mathbf{u}_0 = \mathbf{T}_0\mathbf{v}_0 + v_k\mathbf{t}_{-k}^{-1} + \mathbf{T}_{-1}\mathbf{v}_\mathbf{1}$. Similarly we have $\mathbf{u}_1 = \mathbf{T}_1\mathbf{v}_0 + \mathbf{T}_0\mathbf{v}_1 + v_k\mathbf{t}_1^k$. Note that

$$u_k = v_k \sum_{i=-k}^{k} t_i - \sum_{i=-k}^{k} t_i(v_k - v_{k-i}) = \sum_{i=-k}^{k} t_i v_{k-i},$$

hence the output of the bilinear algorithm is the TMVP. □

6.2.3 Complexity Calculation

We now analyze the additive and multiplicative complexities of our bilinear algorithm constructed above. By Lemma 6.1, the matrices $\mathbf{E}^{(n)}$ and $\mathbf{H}^{(n)}$ consist of only 0, 1, and -1, and hence a multiplication between such a matrix and any vector requires only additions and subtractions. Since the Toeplitz matrix $\mathbf{T}$ and its corresponding vector $\mathbf{t}$ are known in advance in our application, $\mathbf{G}^{(n)}\mathbf{t}$ is pre-computed. Hence, the computational complexity required by $\mathbf{G}^{(n)}\mathbf{t}$ is one-time and is not counted in the computational complexity of our bilinear algorithm. Thus, all multiplications required by our bilinear algorithm are attributed to the entry-wise multiplication between vectors, $\mathbf{G}^{(n)}\mathbf{t} \cdot \mathbf{H}^{(n)}\mathbf{v}$. That is, $M(n)$, the number of rows in $\mathbf{H}^{(n)}$, is also the number of multiplications required by our bilinear algorithm. We denote the additive complexities of the bilinear algorithm for an $n \times n$ TMVP constructed by Eqs. (6.3) and (6.5) as $A(n)$, which is defined as the numbers of additions and subtractions required by the algorithm. Our explicit construction leads to recursive relations for $M(n)$ and $A(n)$, as shown below.

Proposition 6.6. *For an $n \times n$ TMVP, the multiplicative complexity of our bilinear algorithm $M(n)$ satisfies*

$$M(n) = \begin{cases} 3M(k), & n = 2k, \\ 3M(k) + 2k + 1, & n = 2k + 1, \end{cases} \tag{6.8}$$

with initial condition $M(1) = 1$, and the additive complexity $A(n)$ satisfies $A(n) = 3M(n) - 3n$.

Proof. The recursive relation (6.8) is derived by counting the rows of $\mathbf{H}^{(2k)}$ and $\mathbf{H}^{(2k+1)}$ in (6.3) and (6.5), respectively.

The matrices $\mathbf{E}^{(n)}$ and $\mathbf{H}^{(n)}$ consist of only 0, 1, and -1 by Lemma 6.1, which implies the multiplication between such a matrix and a vector requires only additions and subtractions. Assume that $\mathbf{w} = \mathbf{G}^{(n)}\mathbf{t} \cdot \mathbf{H}^{(n)}\mathbf{v}$, we denote the additive complexities of $\mathbf{H}^{(n)}\mathbf{v}$ and $\mathbf{E}^{(n)}\mathbf{w}$ as $A_1(n)$ and $A_2(n)$, respectively.

First, let us show that $A_1(n) = M(n) - n$ by induction. When $n = 1$, no addition is needed, and hence $A_1(n) = M(1) - 1 = 0$. Suppose it holds for n up to t. If $t = 2k - 1$, then when $n = t + 1 = 2k$, by (6.2) and (6.3), $\mathbf{H}^{(2k)}\mathbf{v} = [(\mathbf{H}^{(k)}(\mathbf{v}_0+\mathbf{v}_1))^T \; (\mathbf{H}^{(k)}\mathbf{v}_0)^T \; (\mathbf{H}^{(k)}\mathbf{v}_1)^T]^T$. We first compute a k-dimensional vector addition $\mathbf{v}_0 + \mathbf{v}_1$, and then multiply $\mathbf{H}^{(k)}$ with $\mathbf{v}_0$, $\mathbf{v}_1$, and $\mathbf{v}_0 + \mathbf{v}_1$, respectively. Hence we have $A_1(2k) = 3A_1(k) + k = 3M(k) - 2k = M(2k) - 2k$. If $t = 2k$, then when $n = t + 1 = 2k + 1$, by (6.4) and (6.5), $\mathbf{H}^{(2k+1)}\mathbf{v} =$

$[(\mathbf{H}^{(k)}(\mathbf{v}_0+\mathbf{v}_1))^T\,(\mathbf{H}^{(k)}\mathbf{v}_1)^T\,(\mathbf{D}_{2k+1}^T\mathbf{Y}_{2k+1}\mathbf{v})^T\,(\mathbf{H}^{(k)}\mathbf{v}_0)^T]^T$. Considering the structure of $\mathbf{D}_{2k+1}$ in (6.6), we have $A_1(2k+1) = 3A_1(k)+3k = 3M(k) = M(2k+1)-(2k+1)$. Then we have $A_1(n) = M(n) - n$.

Next, let us show that $A_2(n) = 2M(n)-2n$. when $n = 1$, $A_2(n) = 2M(1)-2 = 0$. Suppose it holds for n up to t. If $t = 2k-1$, then when $n = t+1 = 2k$, we can partition $\mathbf{w}$ into $[\mathbf{w}_0^T\,\mathbf{w}_1^T\,\mathbf{w}_2^T]^T$ with proper sizes, and $\mathbf{E}^{(2k)}\mathbf{w} = [(\mathbf{E}^{(k)}\mathbf{w}_0 + \mathbf{E}^{(k)}\mathbf{w}_1)^T\,(\mathbf{E}^{(k)}\mathbf{w}_0+\mathbf{E}^{(k)}\mathbf{w}_2)^T]^T$. We first compute $\mathbf{E}^{(k)}\mathbf{w}_i$ for $i = 0, 1$, and 2, and then with two additional k-dimensional vector additions, we can derive the product $\mathbf{u}$. Therefore, $A_2(2k) = 3A_2(k) + 2k = 6M(k) - 4k = 2M(2k) - 4k$. If $t = 2k$, then when $n = t+1 = 2k+1$, we can partition $\mathbf{w}$ into $[\mathbf{w}_0^T\,\mathbf{w}_1^T\,\mathbf{w}_2^T\,\mathbf{w}_3^T]^T$, and we have

$$\begin{aligned}
\mathbf{u}_0 &= \mathbf{E}^{(k)}\mathbf{w}_0 + \mathbf{E}^{(k)}\mathbf{w}_1 - [\mathbf{I}_k\ \mathbf{0}_{k\times(k+1)}]\mathbf{w}_2, \\
u_k &= \mathbf{1}_{1\times(2k+1)}\mathbf{w}_2, \\
\mathbf{u}_1 &= \mathbf{E}^{(k)}\mathbf{w}_0 + \mathbf{E}^{(k)}\mathbf{w}_3 - [\mathbf{0}_{k\times(k+1)}\ \mathbf{I}_k]\mathbf{w}_2,
\end{aligned} \tag{6.9}$$

and hence $A_2(2k+1) = 3A_2(k) + 6k = 2M(2k+1) - 4k - 2$.

Since $\mathbf{G}^{(n)}\mathbf{t}$ can be pre-computed, we have $A(n) = A_1(n) + A_2(n) = 3M(n) - 3n$. □

Alternatively, when computing $\mathbf{H}^{(k)}(\mathbf{v}_1+\mathbf{v}_2)$, one can first compute $\mathbf{H}^{(k)}\mathbf{v}_1$ and $\mathbf{H}^{(k)}\mathbf{v}_2$ first, and then sum up them together. This strategy will lead to a different recursion for the additive complexities of $A_1(n)$, given by

$$A_1'(n) = \begin{cases} 2A_1'(k) + M(k), & n = 2k, \\ 2A_1'(k) + M(k) + 2k, & n = 2k+1, \end{cases}$$

with $A_1'(1) = 0$. It is easy to show by induction that $A_1'(n) = A_1(n) = M(n) - n$. Similarly, when computing $\mathbf{E}^{(k)}(\mathbf{w}_0 + \mathbf{w}_1)$ and $\mathbf{E}^{(k)}(\mathbf{w}_0 + \mathbf{w}_2)$, one can first compute $\mathbf{w}_0 + \mathbf{w}_1$ and $\mathbf{w}_0 + \mathbf{w}_3$, and then multiply $\mathbf{E}^{(k)}$ with both sum vectors. This approach will also leads to the same additive complexity with $A_2(n)$.

6.2.4 Comparison with Other TMVP Algorithms

We now compare our bilinear algorithm with its counterpart proposed in [75]. We note that our algorithm is bilinear and the algorithm in [75] is constructed recursively and not presented in a bilinear form. When $n = 2k$, both our bilinear algorithm and that in [75] are based on (6.2). When $n = 2k + 1$, the algorithm in [75] first ignores the first row and the last column of the Toeplitz matrix $\mathbf{T}$ as well as the last element of the vector $\mathbf{v}$, and then computes the $2k \times 2k$ TMVP by (6.7), which requires three $k \times k$ TMVPs. The ignored parts are accounted for separately. That is, the algorithm in [75] is based on

$$\begin{bmatrix} u_0 \\ \mathbf{u}_0 \\ \mathbf{u}_1 \end{bmatrix} = \left[\begin{array}{c:c:c} (\mathbf{t}_0^{-k+1})^T & (\mathbf{t}_{-k}^{-2k+1})^T & t_{-2k} \\ \hdashline \mathbf{T}_0 & \mathbf{T}_{-1} & \mathbf{t}_{-2k+1}^{-k} \\ \hdashline \mathbf{T}_1 & \mathbf{T}_0 & \mathbf{t}_{-k+1}^{0} \end{array}\right] \begin{bmatrix} \mathbf{v}_0 \\ \mathbf{v}_1 \\ v_{2k} \end{bmatrix}. \tag{6.10}$$

Denoting the additive and multiplicative complexities of the algorithm in [75] as $A'(n)$ and $M'(n)$, respectively, we have

$$A'(n) = \begin{cases} 3A'(k) + 3k, & n = 2k, \\ 3A'(k) + 7k, & n = 2k+1, \end{cases}$$

and

$$M'(n) = \begin{cases} 3M'(k), & n = 2k, \\ 3M'(k) + 4k + 1, & n = 2k + 1, \end{cases}$$

with initial conditions $A'(1) = 0$ and $M'(1) = 1$. Compared with the algorithm in [75], when n is odd, the recursive relation of $A(n)$ saves $2k$ multiplications and uses $2k$ more additions; when n is even, $A(n)$ has the same recursive relation. Since $A'(1) = A(1)$ and $M'(1) = M(1)$, one can show that $A'(n) + M'(n) = A(n) + M(n)$ for any n. When n is a power of two, both algorithms have the same computational complexities. When n is not a power of two, our algorithm requires fewer multiplications while needing more additions, which implies that, as long as a multiplication is more complex than an addition, our algorithm has a smaller computational complexity than its counterpart in [75]. In Tab. 6.1, we compare the computational complexities of both the recursive algorithm in [75] and our bilinear algorithm for $n \times n$ TMVPs up to $n = 16$. Although the additive and multiplicative complexities of both algorithms are asymptotically on the order $O(n^{\log_2 3})$ for an $n \times n$ TMVP, the reduced complexities of our algorithm for small TMVPs have a significant impact on CFFTs. This is because the multiplicative complexities of CFFTs depends on efficient cyclic convolutions of small lengths, which will be reformulated as short TMVPs in Sec. 6.3.

6.2.5 Remarks

Our bilinear algorithm includes several well-known efficient (and possibly optimal) TMVP algorithms as special cases. These include 2×2 and 3×3 TMVPs [25,76] as well as the 5×5 TMVP algorithm used in deriving our 11-point cyclic convolutions in

Table 6.1: Comparison of computational complexities of our algorithm and its counterpart in [75] for $n \times n$ TMVPs.

n	[75]		ours		n	[75]		ours	
	$A'(n)$	$M'(n)$	$A(n)$	$M(n)$		$A'(n)$	$M'(n)$	$A(n)$	$M(n)$
1	0	1	0	1	9	73	44	81	36
2	3	3	3	3	10	84	54	96	42
3	7	8	9	6	11	104	75	126	53
4	15	9	15	9	12	108	72	126	54
5	23	18	27	14	13	132	97	162	67
6	30	24	36	18	14	147	111	183	75
7	42	37	54	25	15	175	140	225	90
8	57	27	57	27	16	195	81	195	81

Appendix A, which is the most efficient one to the best of our knowledge. However, $A(n)$ and $M(n)$ are not necessarily the lowest computational complexities for $n \times n$ TMVP. For example, in Tab. 6.1, $M(15) = 90$ and $A(15) = 177$, while $M(16) = 81$ and $A(16) = 195$. As described above, one can extend a 15×15 TMVP to a 16×16 TMVP by padding zeros. To devise an efficient algorithm for any TMVP, it is necessary to explore different approaches described above, including our bilinear algorithm. Our analysis of $A(n)$ assumes a rather straightforward implementation, and $A(n)$ may be further reduced by sophisticated additive complexity reduction tools such as the common subexpression elimination algorithm in [22].

Our bilinear algorithm is designed for TMVPs over any finite field as well as the real and complex fields. Hence, it is possible that the computational complexities of our bilinear algorithm can be reduced by taking advantage of further properties of any particular field.

6.3 CFFTs over Arbitrary Finite Fields

Cyclotomic fast Fourier transforms (CFFTs) considered in the literature [20–22, 67, 70] are for GF(2^m) only. In this section, we generalize CFFTs to any finite field GF(p^m), where p is a prime and m is a positive integer. First, we will formulate CFFTs over any finite field in Sec. 6.3.1. Although our formulation is straightforward and not significantly different from that in previous works [20, 21, 70], we formally present it here so that the content in this chapter will be self-contained. The key ingredient of CFFTs over any finite field is efficient cyclic convolutions over finite fields. Cyclic convolution algorithms over arbitrary finite fields have not been investigated in detail in previous works [20, 21, 70]. We propose an efficient cyclic convolution algorithm for any finite field in Sec. 6.3.2, which in turn enables us to construct CFFTs over any finite field.

6.3.1 CFFTs over Arbitrary Finite Fields

Given a vector $\mathbf{f} = (f_0, f_1, \cdots, f_{n-1})^T$ over GF(p^m), we define its *polynomial representation* by $f(x) = \sum_{i=0}^{n-1} f_i x^i$. The Fourier transform of the vector $\mathbf{f}$ is the collection of elements

$$F_j = f(\alpha^j) = \sum_{i=0}^{n-1} f_i \alpha^{ij}, \tag{6.11}$$

where $0 \le j \le n-1$, and $\alpha \in$ GF(p^m) is an element of order n, i.e. α is an n-th primitive root of one. Therefore n has to divide $p^m - 1$ so that such an element exists in GF(p^m).

A linearized polynomial over GF(p^m) is a polynomial of the form $L(x) = \sum_i l_i x^{p^i}$, where $l_i \in$ GF(p^m). Linearized polynomials are so named because for a linearized polynomial $L(x)$ over $\mathrm{GF}(p^m)$, β_1 and β_2 in an extension field $\mathbb{K}$ of $\mathrm{GF}(p^m)$, and $\lambda_1, \lambda_2 \in \mathrm{GF}(p)$, we have $L(\lambda_1\beta_1 + \lambda_2\beta_2) = \lambda_1 L(\beta_1) + \lambda_2 L(\beta_2)$. Thus, if the elements $\beta_0, \beta_1, \cdots, \beta_{m-1}$ form a basis of $\mathrm{GF}(p^m)$, and $a = \sum_{i=0}^{m-1} a_i\beta_i$, $a_i \in \mathrm{GF}(p)$, then $L(a) = \sum_{i=0}^{m-1} a_i L(\beta_i)$.

Suppose the set of integers $\{0, 1, \cdots, n-1\}$ can be partitioned into k cyclotomic cosets modulo n over $\mathrm{GF}(p)$:

$$\begin{aligned}&\{0\},\ \{s_1, ps_1, \cdots, p^{m_1-1}s_1\},\ \{s_2, ps_2, \cdots, p^{m_2-1}s_2\},\ \cdots\\ &\{s_{k-1}, ps_{k-1}, \cdots, p^{m_{k-1}-1}s_{k-1}\},\end{aligned}$$

where $s_i = s_i p^{m_i} \pmod{n}$ and m_i is the size of the i-th cyclotomic coset. Then any polynomial $f(x) = \sum_{i=0}^{n-1} f_i x^i$ with $f_i \in \mathrm{GF}(p^m)$ can be decomposed as $f(x) = \sum_{i=0}^{k-1} L_i(x^{s_i})$, where $L_i(y) = \sum_{j=0}^{m_i-1} f_{s_i p^j \bmod n}\, y^{p^j}$. Note that for $s_0 = 0$, the term f_0 can be written as the value of the polynomial $L_0(y) = f_0 y$ at $y = x^0 = 1$.

Example: the cyclotomic cosets of $\{0, 1, \cdots, 12\}$ modulo 13 with respect to three are given by $\{0\}$, $\{1, 3, 9\}$, $\{2, 6, 5\}$, $\{4, 12, 10\}$, $\{7, 8, 11\}$. Hence the polynomial $f(x) = \sum_{i=0}^{12} f_i x^i$, $f_i \in \mathrm{GF}(3^3)$ can be decomposed as

$$\begin{aligned}f(x) &= L_0(x^0) + L_1(x) + L_2(x^2) + L_3(x^4) + L_4(x^7),\\ L_0(y) &= f_0 y,\\ L_1(y) &= f_1 y + f_3 y^3 + f_9 y^9,\\ L_2(y) &= f_2 y + f_6 y^3 + f_5 y^9,\\ L_3(y) &= f_4 y + f_{12} y^3 + f_{10} y^9,\\ L_4(y) &= f_7 y + f_8 y^3 + f_{11} y^9.\end{aligned}$$

According to the decomposition, we can write $f(\alpha^j) = \sum_{i=0}^{k-1} L_i(\alpha^{js_i})$. All the elements $(\alpha^{s_i})^j \in \mathrm{GF}(p^{m_i})$ can be represented by a normal basis $\{\gamma_i^{p^0}, \gamma_i^{p^1}, \cdots, \gamma_i^{p^{m_i-1}}\}$ of $\mathrm{GF}(p^{m_i})$ (the existence of such basis in any finite field is guaranteed by the normal basis theorem [77]). That is, $\alpha^{js_i} = \sum_{s=0}^{m_i-1} a_{i,j,s}\gamma_i^{p^s}$, where $a_{i,j,s} \in \mathrm{GF}(p)$. Hence

$$\begin{aligned} f(\alpha^j) &= \sum_{i=0}^{k-1}\sum_{s=0}^{m_i-1} a_{i,j,s} L_i(\gamma_i^{p^s}) \\ &= \sum_{i=0}^{k-1}\sum_{s=0}^{m_i-1} a_{i,j,s}\left(\sum_{t=0}^{m_i-1} \gamma_i^{p^{s+t \bmod m_i}} f_{s_i p^t \bmod n}\right). \end{aligned}$$

This expression can be rewritten in the matrix form as

$$\mathbf{F} = \mathbf{A}\mathbf{L}\mathbf{f}' \tag{6.12}$$

where $\mathbf{f}_i = (f_{s_i}, f_{ps_i} \cdots, f_{p^{m_i-1}s_i})$ and $\mathbf{f}' = (f_0, \mathbf{f}_1, \mathbf{f}_2, \cdots, \mathbf{f}_{k-1})^T$ is just a permutation of $\mathbf{f}$ according to the cyclotomic cosets. The matrix $\mathbf{A}$ consists of the elements $a_{i,j,s} \in \mathrm{GF}(p)$. The matrix $\mathbf{L}$ is block diagonal given by $\mathrm{diag}(\mathbf{L}_0, \mathbf{L}_1, \cdots, \mathbf{L}_{k-1})$, where

$$\mathbf{L}_i = \begin{bmatrix} \gamma_i^{p^0} & \gamma_i^{p^1} & \cdots & \gamma_i^{p^{m_i-1}} \\ \gamma_i^{p^1} & \gamma_i^{p^2} & \cdots & \gamma_i^{p^0} \\ \vdots & \vdots & \ddots & \vdots \\ \gamma_i^{p^{m_i-1}} & \gamma_i^{p^0} & \cdots & \gamma_i^{p^{m_i-2}} \end{bmatrix}$$

is a cyclic matrix. The multiplication between $\mathbf{L}_i$ and the vector $\mathbf{f}_i^T$ can be formulated as a cyclic convolution between $\mathbf{b}_i = (\gamma_i^{p^0}, \gamma_i^{p^{m_i-1}}, \cdots, \gamma_i^{p^1})^T$ and $\mathbf{f}_i$. Hence with efficient cyclic convolution algorithms (see, e.g., [25]), the multiplicative complexities of CFFTs can be greatly reduced. Therefore, for CFFTs over arbitrary finite fields, efficient cyclic convolution algorithms over arbitrary finite fields, especially those for short cyclic convolutions are needed.

6.3.2 A Cyclic Convolution Algorithm over GF(p^m)

Efficient cyclic convolution algorithms over arbitrary finite fields play an important role in the multiplicative complexity reduction of CFFTs. Unfortunately, cyclic convolution algorithms over arbitrary finite fields have not been investigated in detail in previous works on CFFTs [20, 21, 70]. Herein we propose an efficient algorithm for cyclic convolutions over arbitrary finite fields.

Consider an n-point cyclic convolution over GF(p^m), where p is prime and m is a positive integer. If $n = n_1 n_2$, we can use the multidimensional technologies (see, e.g., [25]) to construct an n-point cyclic convolution from n_1- and n_2-point cyclic convolutions. Thus, henceforth we consider only the cases where n is a prime number.

For an n-dimensional vector $\mathbf{x} = (x_0, x_1, \cdots, x_{n-1})^T$ over GF(p^m), where n is any prime integer, we consider its polynomial representation $x(w) = \sum_{i=0}^{n-1} x_i w^i$. Assuming that the n-point cyclic convolution of two vectors $\mathbf{x}$ and $\mathbf{y}$ is $\mathbf{z}$, all of which are n-dimensional vectors over GF(p^m), their polynomial representations are related by

$$z(w) = x(w)y(w) \pmod{w^n - 1}. \tag{6.13}$$

Note that $w^n - 1 = (w-1)(w^{n-1} + w^{n-2} + \cdots + 1)$, and $w-1$ and $w^{n-1} + w^{n-2} + \cdots + 1$ are co-prime in GF(p^m) when $n \neq p$. Hence by the Chinese remainder theorem [77],

$z(w)$ can be uniquely determined by Z_0 and $Z'(w) = \sum_{i=0}^{n-2} z'_i w^i$, where

$$\begin{aligned} Z_0 &= z(w) \pmod{w-1}, \\ Z'(w) &= z(w) \pmod{w^{n-1} + w^{n-2} + \cdots + 1}. \end{aligned} \tag{6.14}$$

It is easy to see that $Z_0 = \sum_{i=0}^{n-1} z_i$, $Z'_i = z_i - z_{n-1}$ for $0 \leq i \leq n-2$, and the vector $\mathbf{Z} = (Z_0, Z'_0, Z'_1, \cdots, Z'_{n-2})^T = (Z_0, \mathbf{Z}'^T)^T$ can be derived by multiplying the vector $\mathbf{z}$ with an $n \times n$ matrix:

$$\mathbf{Z} = \mathbf{B}\mathbf{z} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ & & & -1 \\ & \mathbf{I}_{n-1} & & \vdots \\ & & & -1 \end{bmatrix} \mathbf{z}.$$

Representing $\mathbf{x}$ and $\mathbf{y}$ in the same fashion, it is easy to see that $Z_0 = X_0 Y_0$, and

$$Z'(w) = X'(w)Y'(w) \pmod{w^{n-1} + w^{n-2} + \cdots + 1}. \tag{6.15}$$

Therefore, to compute the n-point cyclic convolution of $\mathbf{x}$ and $\mathbf{y}$, we first compute $\mathbf{X} = \mathbf{B}\mathbf{x}$ and $\mathbf{Y} = \mathbf{B}\mathbf{y}$, then compute $\mathbf{Z}$ from $\mathbf{X}$ and $\mathbf{Y}$, and finally obtain $\mathbf{z} = \mathbf{B}^{-1}\mathbf{Z}$.

From (6.15), the polynomial product can be computed as

$$\begin{aligned} X'(w)Y'(w) = \sum_{k=0}^{n-2} \sum_{j=0}^{n-2} (Y'_{k-j} + Y'_{k-j+n} + Y'_{n-1-j}) X'_j w^k \\ \pmod{w^{n-1} + w^{n-2} + \cdots + 1}, \end{aligned} \tag{6.16}$$

and hence the vector $\mathbf{Z}'$ can be computed through a matrix product $\mathbf{Z}' = \mathbf{M}\mathbf{X}'$, where the elements of matrix $\mathbf{M}$ are

$$M_{k,j} = Y'_{k-j} + Y'_{k-j+n} - Y'_{n-1-j}. \tag{6.17}$$

Note that in (6.16) and (6.17), Y'_i is zero outside its valid range, i.e., $Y'_i = 0$ if $i < 0$ or $i > n-2$.

We can check that $\mathbf{B}$ is invertible, and $\mathbf{B}^{-1}$ is given by

$$\mathbf{B}^{-1} = n^{-1} \begin{bmatrix} 1 & \mathbf{A}_1 \\ \mathbf{A}_2 & \mathbf{A}_3 \end{bmatrix},$$

where $n^{-1} \in \mathrm{GF}(p)$ is an integer such that $n^{-1}n = 1 \pmod{p}$, the $(n-1)$-dimensional row vector $\mathbf{A}_1 = (n-1,\ -1, -1, \cdots, -1)$, the $(n-1)$-dimensional column vector $\mathbf{A}_2 = (1, 1, \cdots, 1)^T$, and the $(n-1) \times (n-1)$ matrix $\mathbf{A}_3$ has $n-1$ on the first upper diagonal and -1 everywhere else.

Now consider $\mathbf{z} = (z_0, \mathbf{z}'^T)^T$ as the product of $\mathbf{B}^{-1}$ and $\mathbf{Z}$:

$$\mathbf{z} = \begin{bmatrix} z_0 \\ \mathbf{z}' \end{bmatrix} = \mathbf{B}^{-1} \begin{bmatrix} Z_0 \\ \mathbf{Z}' \end{bmatrix} = n^{-1} \begin{bmatrix} 1 & \mathbf{A}_1 \\ \mathbf{A}_2 & \mathbf{A}_3 \end{bmatrix} \begin{bmatrix} Z_0 \\ \mathbf{Z}' \end{bmatrix}.$$

Note that $\mathbf{A}_1$ and $\mathbf{A}_3$ are related by $\mathbf{A}_1 = -(1, 1, \ldots, 1)\,\mathbf{A}_3$. This implies that the sum of the components of $\mathbf{A}_3\mathbf{Z}'$ gives $\mathbf{A}_1\mathbf{Z}'$. Furthermore, $\mathbf{A}_2$ contains only ones. Thus the computation of z_0 and $\mathbf{z}'$ reduces to

$$\begin{aligned} z_0 &= n^{-1}(Z_0 - \mathbf{1}_{n-1}^T(\mathbf{A}_3\mathbf{Z}')), \\ \mathbf{z}' &= n^{-1}([Z_0, Z_0, \ldots, Z_0]^{\mathrm{T}} + \mathbf{A}_3\mathbf{Z}'), \end{aligned} \tag{6.18}$$

where $\mathbf{1}_{n-1}$ is an $(n-1)$-dimensional all-one column vector. Eq. (6.18) shows that multiplying a vector with $\mathbf{B}^{-1}$ needs only an evaluation of $\mathbf{A}_3\mathbf{Z}'$. Though the final results have to be derived by multiplying n^{-1}, whence an operand of the cyclic convolution is a constant vector, multiplying n^{-1} can be done in the pre-computation by scaling the constant operand.

Since $\mathbf{Z}' = \mathbf{M}\mathbf{X}'$, one needs to compute $\mathbf{R}\mathbf{X}'$ where the $(n-1)\times(n-1)$ matrix $\mathbf{R} = \mathbf{A}_3\mathbf{M}$. We now show that $\mathbf{R}$ is a Toeplitz matrix. From the structure of $\mathbf{A}_3$, we have

$$R_{i,j} = M_{i+1,j} - n^{-1}\sum_{k=0}^{n-2} M_{k,j}. \tag{6.19}$$

Using appropriate ranges for the three terms of the right hand side of (6.17), we get

$$\sum_{k=0}^{n-2} M_{k,j} = -nY'_{n-1-j} + \sum_{s=0}^{n-2} Y'_s. \tag{6.20}$$

Finally, combining (6.17), (6.19) and (6.20) leads to

$$R_{i,j} = Y'_{i-j+1} + Y'_{i-j+n+1} - n^{-1}\sum_{s=0}^{n-2} Y'_s. \tag{6.21}$$

Since $R_{i,j}$ is a function of only $i-j$, $\mathbf{R}$ is a Toeplitz matrix. Recall that Y'_i is zero if its index is outside the valid range of 0 to $n-2$. Thus in (6.21), at most one of the first two terms is valid for any combination of i and j.

With this method, we reformulate the n-point cyclic convolution into the product between a Toeplitz matrix $\mathbf{R}$ and a vector $\mathbf{X}'$. Direct implementation of $\mathbf{R}\mathbf{X}'$ requires $(n-1)^2$ multiplications over GF(p^m), but we can reduce its multiplicative complexity since $\mathbf{R}$ is a Toeplitz matrix. For any odd prime $n > 3$, $n-1$ is composite and $\mathbf{RX}'$ can be obtained by using multidimensional technologies from Toeplitz matrix vector products of smaller sizes [75, 76, 78–80], using our efficient TMVP algorithm in Sec. 6.2.

When $n = p$, $w^p - 1 = (w-1)^p$, and Z_0 and $Z'(w)$ cannot determine $Z(w)$, hence the above method is not valid in such cases. We formulate the cyclic convolution as the multiplication between a cyclic matrix and a vector, i.e.,

$$\mathbf{z} = \begin{bmatrix} x_0 & x_{p-1} & x_{p-2} & \cdots & x_1 \\ x_1 & x_0 & x_{p-1} & \cdots & x_2 \\ x_2 & x_1 & x_0 & \cdots & x_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{p-1} & x_{p-2} & x_{p-3} & \cdots & x_0 \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{p-1} \end{bmatrix}. \tag{6.22}$$

It is easy to see that the cyclic matrix is also a Toeplitz matrix, and hence we can evaluate this matrix vector product using our TMVP algorithm in Section 6.2.

In summary, an n-point (n is prime) cyclic convolution over any finite field is reformulated as either an $(n-1) \times (n-1)$ TMVP or an $n \times n$ TMVP. Based on our efficient TMVP algorithm in Sec. 6.2, both the multiplicative and additive complexities of an n-point (n is prime) cyclic convolution over any finite field is on the order of $O(n^{\log_2 3})$ asymptotically. However, when used for CFFTs, a small reduction in the computational complexities of cyclic convolutions yields a large reduction of the total complexities of CFFTs. The merit of our reformulation above is in that we can take advantage of multidimensional technologies at the expense of only one extra multiplication when the cyclic convolution length is prime and not equal to the characteristic of the field.

We remark that the cyclic convolution algorithm based on our efficient bilinear TMVP is also a bilinear one. That is, over $\mathrm{GF}(p^m)$, given a prime n and $n \neq p$, the cyclic convolution of two n-dimensional vectors $\mathbf{x}$ and $\mathbf{y}$ can be computed by $n^{-1}\mathbf{Q}^{(n)}(\mathbf{P}^{(n)}\mathbf{x} \cdot \mathbf{R}^{(n)}\mathbf{y})$, where $\mathbf{Q}^{(n)}$, $\mathbf{P}^{(n)}$, and $\mathbf{R}^{(n)}$ are matrices containing only -1, 0, and 1, and multiplying with such matrices only requires additions and subtractions. Since this algorithm is bilinear, whence an operand is a constant, it can be pre-scaled with n^{-1}, and hence multiplying n^{-1} does not account for any multiplication. Furthermore, in characteristic-2 fields, which are most commonly used in practice, $n^{-1} = 1$ and hence this term can be neglected.

Example: 11-point cyclic convolution algorithm does not exist in the literature to the best of our knowledge. As this algorithm is instrumental in deriving CFFTs over $\mathrm{GF}(2^{11})$, we propose an 11-point cyclic convolution algorithm in Appendix A, which is designed based on the reformulation introduced in this section. Note that in $\mathrm{GF}(2)$, $-1 = 1$ and hence the matrices $\mathbf{Q}^{(11)}$, $\mathbf{P}^{(11)}$, and $\mathbf{R}^{(11)}$ are all binary.

6.4 Computational Complexities of CFFTs

The advantages of CFFTs in their multiplicative and total computational complexities have been established by numerical comparison with other algorithms so far [20–22]. The goal of our research is to theoretically show that they do have the smallest multiplicative complexities among all known algorithms, and to determine the optimality of the total computational complexity of CFFTs.

In this section, we focus on CFFTs of lengths $n = p^m - 1$ over $\mathrm{GF}(p^m)$. As shown in Sec 6.3.1, a CFFT computes the DFT of an n-dimensional vector $\mathbf{f}$ by

$\mathbf{ALf}'$, where the matrix $\mathbf{A}$ is over GF(p). We assume that multiplying a vector with $\mathbf{A}$ can be done by additions, since $ab = \sum_{i=1}^{a} b$ for $a \in \mathrm{GF}(p)$ and $b \in \mathrm{GF}(p^m)$. The vector $\mathbf{v} = \mathbf{Lf}'$ is computed via k cyclic convolutions, with $\mathbf{L}_i\mathbf{f}'_i$ being an m_i-point cyclic convolution. Thus the multiplicative complexity of a CFFT is contributed by the k convolutions, and the additive complexity is due to both the convolutions and computing $\mathbf{Av}$.

6.4.1 Multiplicative Complexities of CFFTs

We first introduce some notations instrumental to our derivations. We partition the set $\{0, 1, \cdots, n-1\}$ into k cyclotomic cosets modulo n with respect to p, denoted by $C_0, C_1, \cdots, C_{k-1}$. We assume the size of C_i is given by m_i, and s_i is its representative. We know that m_i divides m. We divide the cosets C_i's into d groups, denoted by $G_0, G_1, \cdots, G_{d-1}$, so that all C_i's in the same group G_j are of the same size g_j. The size of G_j is given by $|G_j|$.

The convolutions are the only source of the multiplicative complexity of an n-point CFFT. It is a well-known result that an m_i-point cyclic convolution has a multiplicative and additive complexities on the order of $O(m_i^{\log_2 3})$ [76], which implies that there exists a constant c independent with m_i such that its multiplicative complexity is less than $cm_i^{\log_2 3}$. Therefore, the total multiplicative complexity of an n-point CFFT is less than $c\sum_{i=0}^{k-1} m_i^{\log_2 3}$. We have that the size of the cosets in G_j divides m, i.e., $g_j|m$, and also we have $d \leq m$. Since $\log_2 3 > 1$, we have $\frac{m}{g_j}(g_j)^{\log_2 3} = m(g_j)^{\log_2 \frac{3}{2}} \leq m(m)^{\log_2 \frac{3}{2}} = m^{\log_2 3}$. Then when $m \geq 4$, we can show that $d \leq m \leq (2^m - 1)/m \leq (p^m - 1)/m$, and hence the total multiplicative complexity satisfies

$$\begin{aligned}
&c\sum_{i=0}^{k-1} m_i^{\log_2 3} = c\sum_{j=0}^{d-1} |G_j| g_j^{\log_2 3}\\
=&c\sum_{j=0}^{d-1} \lfloor \frac{|G_j| g_j}{m} \rfloor \frac{m}{g_j} g_j^{\log_2 3} + c\sum_{j=0}^{d-1} (|G_j| \bmod m/g_j) g_j^{\log_2 3}\\
\le& c\lfloor \frac{p^m-1}{m} \rfloor m^{\log_2 3} + cdm^{\log_2 3}\\
\le& 2c\frac{p^m-1}{m} m^{\log_2 3}.
\end{aligned}$$

The first inequality is because $\frac{m}{g_j}(g_j)^{\log_2 3} \le m^{\log_2 3}$, and the second one is because $d \le (p^m-1)/m$ when $m \ge 4$. We are considering the asymptotic complexity, and hence we do not need to consider the cases $m < 4$. The total multiplicative complexity of an n-point CFFT is thus $O(n(\log_p n)^{\log_2 \frac{3}{2}})$ since $m = \log_p(n+1)$.

6.4.2 Additive Complexities of CFFTs

The additive complexity of an n-point CFFT over GF(p^m) has two sources, the convolutions and computing $\mathbf{Av}$. The total additive complexity contributed by the convolutions is given by $O(n(\log_p n)^{\log_2 \frac{3}{2}})$, the same as the CFFT multiplicative complexity since the additive and multiplicative complexities of a cyclic convolution are on the same order. Even though $\mathbf{A}$ is a matrix over GF(p), the multiplication between $\mathbf{A}$ and a vector can still be implemented with only additions. A multiplication between an element in GF(p) and one in GF(p^m) can be done via at most $p-2$ additions. Computing $\mathbf{Av}$ is the primary source of the additive complexity and we will derive the additive complexity of $\mathbf{Av}$.

Multiplication between an Arbitrary Matrix over GF(p) and a Vector

Consider the multiplication between an arbitrary $M \times M$ matrix over GF(p) and an M-dimensional vector $\mathbf{x}$. Implementing directly, it requires $O(pM^2)$ additions. When $p = 2$, the Four-Russian algorithm [81] can be used to reduce the additive complexity. We are going to generalize the Four-Russian algorithm to GF(p).

Let $s = \lfloor \log_p M \rfloor$. If s does not divide M, we need to pad at most $s - 1$ zero columns to $\mathbf{M}$, and the new matrix is of size $M \times M'$. Since $M \leq M' < M + s < 2M$, M and M' are on the same order, and we can assume that s divides M without loss of generality. We first partition $\mathbf{M}$ as

$$\mathbf{M} = \begin{bmatrix} \mathbf{M}_0 & \mathbf{M}_1 & \cdots & \mathbf{M}_{\frac{M}{s}-1} \end{bmatrix},$$

where $\mathbf{M}_i$ is an $M \times s$ matrix. To compute the product $\mathbf{Mx}$, we first partition the vector $\mathbf{x}$ into appropriate sizes, and then $\mathbf{Mx} = \sum_{i=0}^{\frac{M}{s}-1} \mathbf{M}_i \mathbf{x}_i$. We can compute the product between the $M \times s$ matrix $\mathbf{M}_i$ and the vector $\mathbf{x}_i$ by first computing all the p^s possible linear combinations of the elements in the s-dimensional sub-vector $\mathbf{x}_i$. Then each element in $\mathbf{M}_i \mathbf{x}_i$ can be looked up in these combinations. All p^s combinations can be done by Alg. 6, where $\mathcal{C} + x \stackrel{\text{def}}{=} \{c + x : c \in \mathcal{C}\}$. The k-th round requires $(p-1)p^k$ additions, and the total additive complexity of this algorithm is $(p-1)\sum_{k=0}^{s-1} p^k = p^s - 1 \leq M$. Therefore, computing all $\mathbf{M}_i \mathbf{x}_i$ requires at most $\frac{M}{s}(p^s - 1) \leq M^2/s$ additions.

After evaluating all the products $\mathbf{M}_i \mathbf{x}_i$, the final result can be derived by at most another M^2/s additions. Since $s = \lfloor \log_p M \rfloor$, the additive complexity for computing $\mathbf{Mx}$ is $O(M^2/\log_p M)$.

Algorithm 6 Compute all combinations of s elements with coefficients up to $p-1$

```
Input: s numbers $x_0, x_1, \cdots, x_{s-1}$
1: Initialize: $\mathcal{C} = 0$.
2: for $k = 0$ to $s - 1$ do
3:     $\mathcal{C} = \mathcal{C} \cup (\mathcal{C} + x_k) \cup \cdots \cup (\mathcal{C} + (p-1)x_k)$
4: end for
5: Output: All combinations of $x_0, x_1, \cdots, x_{s-1}$ with coefficients from GF($p$).
```

Additive Complexity of Av

The result we derived above assumes that the matrix operand is arbitrary and does not consider its structure. To further reduce the additive complexity of $\mathbf{Av}$, we need to explore the inner structure of the $n \times n$ matrix $\mathbf{A}$.

Consider an n-point CFFT over GF(p^m) with $n = p^m - 1$. Let α be a primitive element in GF(p^m). As shown in Sec. 6.3.1, we can partition $\mathbf{A}$ into $1 \times k$ blocks. Each block $\mathbf{A}_i$ is of size $(p^m - 1) \times m_i$, and its row j is the representation of α^{js_i} under a normal basis in the subfield GF(p^{m_i}). In other words, the columns of $\mathbf{A}$ correspond to the k cyclotomic cosets of $\{0, 1, \cdots, n-1\}$ modulo n with respect to p. We further reformulate $\mathbf{Av}$ in the following **three** steps.

1. We first reorder the rows of the matrix $\mathbf{A}$ according to these cyclotomic cosets. We then obtain a new matrix $\mathbf{A}'$ after the permutation, and partition it into $k \times k$ blocks. Each block $\mathbf{A}'_{ij}$ is an $m_i \times m_j$ matrix, and its row t is the representation of $\alpha^{p^t s_i s_j}$ under a normal basis in GF(p^{m_j}). Because of the property of normal bases, row t ($1 \le t \le m_i - 1$) is just a cyclic shift of its previous row, and therefore $\mathbf{A}'_{ij}$ is a cyclic matrix [21]. Then we partition the vector $\mathbf{v}$ into k sub-vectors accordingly, i.e., each sub-vector $\mathbf{v}_i$ has m_i elements. The product $\mathbf{Av}$ is then recovered by reordering the elements in the

product $\mathbf{A}'\mathbf{v}$.

2. The next step is extending all the $m_i \times m_j$ blocks of $\mathbf{A}'$ to $m \times m$ ones while maintaining the cyclic property. Since m_i and m_j both divide m, this can be done by first partitioning an $m \times m$ matrix into $\frac{m}{m_i} \times \frac{m}{m_j}$ blocks of size $m_i \times m_j$, and then setting each block to $\mathbf{A}'_{ij}$. Obviously, the $m \times m$ matrix is still a cyclic matrix. Then we obtain a $km \times km$ matrix $\mathbf{A}''$ after extending all the blocks to $m \times m$ ones in this way. Each sub-vector $\mathbf{v}_i$ should also be extended to an m-dimensional one by padding zeros in the end, resulting in a km-dimensional vector $\mathbf{v}''$. The product $\mathbf{Av}$ can be recovered from $\mathbf{A}''\mathbf{v}''$ by discarding the elements corresponding to the extended rows.

3. The last step is constructing a matrix $\mathbf{B}$ and a vector $\mathbf{u}$ from $\mathbf{A}''$ and $\mathbf{v}''$, respectively, according to the following rules:

$$B_{i_2k+i_1,j_2k+j_1} = A''_{i_1m+i_2,j_1m+j_2}, \quad u_{i_2k+i_1} = v''_{i_1m+i_2},$$

where for $0 \le i_1, j_1 < k$ and $0 \le i_2, j_2 < m$, $A''_{i,j}$ and $B_{i,j}$ are the elements in row i and column j in $\mathbf{A}''$ and $\mathbf{B}$, respectively, and u_i and v''_i are the elements at position i in the vector $\mathbf{u}$ and $\mathbf{v}''$, respectively. It is easy to check that $\mathbf{B}$ and $\mathbf{u}$ are just permutations of $\mathbf{A}''$ and $\mathbf{v}''$, respectively, and they are reordered in the same way so that the product $\mathbf{A}''\mathbf{v}''$ can be recovered from $\mathbf{Bu}$ by only reordering. Furthermore, since the matrix $\mathbf{A}''$ contains $k \times k$ blocks of cyclic matrices, each of which is of size $m \times m$, the matrix $\mathbf{B}$ is a block-cyclic matrix with $m \times m$ blocks of size $k \times k$, i.e., the block row t $(1 \le t \le m-1)$ is a cyclic shift of its previous block row.

The computational complexity of $\mathbf{Bu}$ serves as an upper bound of that of $\mathbf{Av}$ because $\mathbf{Av}$ can be obtained from $\mathbf{Bu}$ without any additional computation. Since the matrix $\mathbf{B}$ is block-cyclic, we can compute $\mathbf{Bu}$ with the bilinear algorithm of m-point block cyclic convolution, which requires three kinds of operations: the multiplication between a $k \times k$ matrix and a k-dimensional vector, the addition between two $k \times k$ matrices, and the addition between two k-dimensional vectors. Each operation have to be performed $O(m^{\log_2 3})$ times [25, 76]. Since the matrix $\mathbf{B}$ is fixed, all the additions between $k \times k$ matrices do not contribute to the additive complexity because they can be pre-computed. Since those blocks are over GF(p), we use Alg. 6 to compute the $k \times k$ matrix vector product. Since each k-dimensional vector addition requires k additions, the total computational complexity is given by

$$O(m^{\log_2 3}\frac{k^2}{\log_2 k}) + O(m^{\log_2 3}k) = O(m^{\log_2 3}\frac{k^2}{\log_2 k}).$$

We need to find out the lower and upper bounds of k to further simplify the additive complexity bound. Let us prove a lemma before giving the bounds.

Lemma 6.7. *For each group G_j of the cyclotomic cosets of $\{0, 1, \cdots, p^m-2\}$ modulo $p^m - 1$ with respect to p, $|G_j| \le (p^{g_j} - 1)/g_j$.*

Proof. Let α be a primitive element in GF(p^m). Each nonzero element in GF(p^m) can be represented as α^t. There is at least one normal basis in GF(p^m) by normal basis theorem [77]. Given a normal basis $\{\gamma^{p^0}, \gamma^{p^1}, \cdots, \gamma^{p^{m-1}}\}$ in GF(p^m), every element in GF(p^m) can be represented by an m-dimensional vector over GF(p). That is, if $\alpha^t = \sum_{i=0}^{m-1} b_i \gamma^{p^j}$, then the vector $(b_{m-1}, b_{m-2}, \cdots, b_0)$ is the vector representation of α^j.

The normal basis representation of α^{pt} is just a cyclic shift of that of α^t. Therefore, if $t \in C_i$ which is in group G_j, the vector representation of α^t repeats itself after g_j shifts since $t = p^{g_j}t \pmod{p^m - 1}$. If $g_j < m$, then $g_j | m$, and we can partition the vector representation of α^j into m/g_j blocks of lengths g_i. All of these blocks are identical, otherwise the vector cannot repeat itself after g_j cyclic shifts. Hence there are at most $(p^{g_j} - 1)/g_j$ cyclotomic cosets with size g_j. □

Lemma 6.8. *Let k be the number of the cyclotomic cosets of $\{0, 1, \cdots, p^m - 2\}$ modulo $p^m - 1$ with respect to p. When $m \geq 9$, $p^m - 1 < km < 2(p^m - 1)$.*

Proof. The lower bound of km is because m is the maximum size of cyclotomic cosets. To prove the upper bound of km, we first assume that the group G_0 contains the cosets with a size of m, and that other groups contain the cosets with sizes less than m. Then we have

$$\begin{aligned} km &= |G_0|m + \sum_{j=1}^{d-1} |G_j|m \\ &\leq (p^m - 1) + m \sum_{m_i=1, m_i|m}^{\lfloor \frac{m}{2} \rfloor} \frac{p^{m_i} - 1}{m_i} \\ &\leq (p^m - 1) + m \sum_{m_i=1}^{\lfloor \frac{m}{2} \rfloor} p^{m_i} \\ &= (p^m - 1) + m(p^{\lfloor \frac{m}{2} \rfloor + 1} - 1) \\ &\leq (p^m - 1) + m(p^{\frac{m+2}{2}} - 1). \end{aligned} \tag{6.23}$$

The first inequality is given by Lemma 6.7. When $m \geq 9$, $(p^m - 1)/(p^{\frac{m+2}{2}} - 1) \geq p^{\frac{m-2}{2}} - 1 \geq 2^{\frac{m-2}{2}} - 1 \geq m$, therefore, $m(p^{\frac{m+2}{2}} - 1) \leq p^m - 1$ and then we have $km \leq 2(p^m - 1)$. □

Since we are considering the asymptotic computational complexity of an n-point CFFT over GF(p^m), and there are only a finite number of m's that do not satisfy the upper bound given by Lemma 6.8, we do not need to consider them. Furthermore, Lemma 6.8 holds for all $m \geq 1$ when $p = 2$.

The total additive complexity of computing $\mathbf{Bu}$ is $O(m^{\log_2 3} k^2 / \log_p k)$, hence there exists a constant c independent of m and k such that computing $\mathbf{Bu}$ requires less than $c m^{\log_2 3} k^2 / \log_p k$ additions. By Lemma 6.8, we have

$$\begin{aligned} c m^{\log_2 3} \frac{k^2}{\log_p k} &\leq c m^{\log_2 3} \frac{4(p^m-1)^2}{m^2 \left[\log_p(p^m-1) - \log_p m\right]} \\ &\leq c m^{\log_2 3} \frac{4(p^m-1)^2}{m^2 (m + \log_p \frac{1-p^{-m}}{m})}. \end{aligned} \tag{6.24}$$

Since $\lim_{m\to\infty} \frac{1}{m} \log_p \frac{1-p^{-m}}{m} = 0$, there exists a number N such that when $m > N$, $\log_p \frac{1-p^{-m}}{m} \geq -\frac{m}{2}$. Substituting this result to (6.24), we have

$$c m^{\log_2 3} \frac{k^2}{\log_p k} \leq c m^{\log_2 3} \frac{8(p^m-1)^2}{m^3} = 8c \frac{(p^m-1)^2}{m^{\log_2 \frac{8}{3}}}.$$

We do not need to consider the cases $m \leq N$ since our focus is on the asymptotic additive complexity of $\mathbf{Bu}$.

When $n = p^m - 1$, the additive complexity of computing $\mathbf{Bu}$ is upper bounded by $O(n^2/(\log_p n)^{\log_2 \frac{8}{3}})$, a complexity lower than that of Alg. 6. The additive complexity of $\mathbf{Av}$ is on the same order since it can be obtained from $\mathbf{Bu}$ without additional computation. Since this complexity is higher than the additive complexity contributed by the convolutions, the total additive complexity of CFFT is also bounded by $O(n^2/(\log_p n)^{\log_2 \frac{8}{3}})$.

6.4.3 Discussions

Applying our results to GF(2^m), i.e., $p = 2$, we have that for an n-point CFFT over GF(2^m) with $n = 2^m - 1$, the multiplicative complexity is $O(n(\log_2 n)^{\log_2 \frac{3}{2}})$ and the additive complexity is $O(n^2/(\log_2 n)^{\log_2 \frac{8}{3}})$. To evaluate the tightness of these asymptotic bounds, we compare them with the actual computational complexities of CFFTs in [22]. We scale our bounds to match the actual complexities when $n = 1023$ in Fig. 6.1. Our bounds on both the additive and multiplicative complexities are

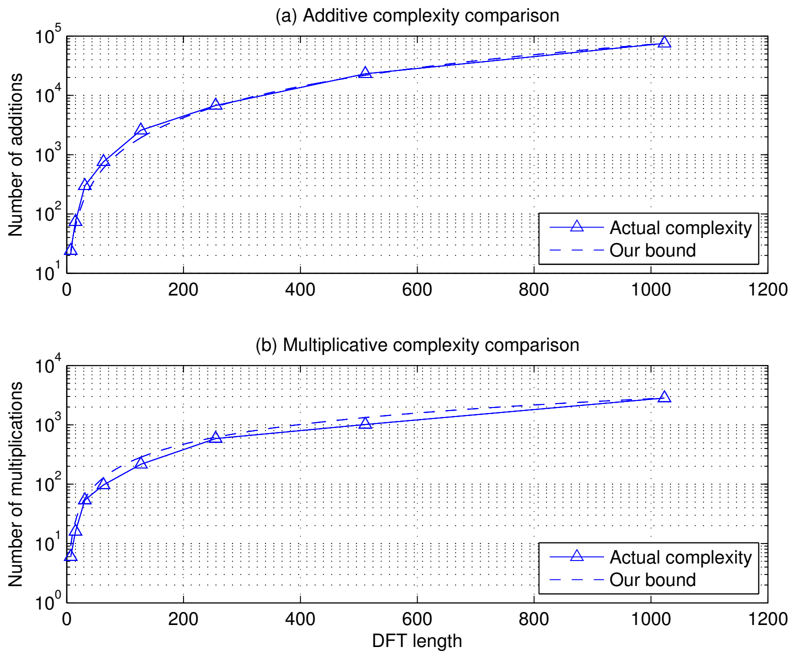


Figure 6.1: Comparison of the actual complexities and our bounds for n-point CFFTs over GF(2^m) with $n = 2^m - 1$.

rather tight, which is shown in Fig. 6.1. The dashed curves corresponding to the theoretical bounds almost coincide with the solid curves corresponding to the actual complexities, which implies the actual additive complexities increase in a similar order to $O(n^2/(\log_2 n)^{\log_2 \frac{8}{3}})$, and the multiplicative complexities increase in a similar order to $O(n(\log_2 n)^{\log_2 \frac{3}{2}})$. Since we have scaled our bounds to match the actual complexities at $n = 1023$, it is not necessary that our bounds are strictly larger than the actual complexities.

In the following we are going to compare the asymptotic bounds of our results and other DFT algorithms in the literature. Here we still consider the case $p = 2$ in our results. A fast algorithm is proposed in [71] for DFTs in GF(2^m) with arbitrary integer m, and its multiplicative and additive complexities are both of $O(n(\log_2 n)^2)$. Another algorithm with the complexities on the same order are proposed [82], respectively. In [83], an FFT algorithm with $O(n \log_2 n)$ multiplicative complexity and $O(n(\log_2 n)^2)$ additive complexity is proposed. When m is a power of two, more efficient algorithms are proposed in [71,73,74,84], and [83]. The most efficient one proposed in [83] has a multiplicative complexity of $O(n \log_2 n)$ and an additive complexity of $O(n \log_2 n \log_2 \log_2 n)$. Cantor's algorithm [72] is more general and it works for arbitrary algebras rather than finite fields. When DFT with a length of $n = s^r$ is well-defined in the underlying algebra, Cantor's algorithm has both multiplicative and additive complexities of $rn(s-1)$.

We compare the asymptotic computational complexities of CFFTs and other existing algorithms in Tab. 6.2, applying them to the DFTs with length $2^m - 1$ over GF(2^m). The total complexity is defined to be a weighted sum of the additive and multiplicative complexities. We assume that one multiplication has the same complexity as $(2m-1)$ additions over the same field. This assumption comes from both the hardware and software considerations [22,85]. Since we focus on (2^m-1)-point DFT, $m = \log_2(n+1)$ and $2m-1$ is on the order of $O(\log_2 n)$.

Table 6.2: Asymptotic complexities of $(2^m - 1)$-point DFT algorithms and their respective restrictions. All logarithms are base two.

Algorithm	Restriction		Complexities		
	Fields	Lengths (n)	multiplicative	additive	total
Mateer [74], Gao [83]	GF(2^m), $m = 2^K$	$2^m - 1$	$O(n \log n)$	$O(n \log n \log \log n)$	$O(n(\log n)^2)$
Cantor [84], Wang [71]	GF(2^m), $m = 2^k$	$2^m - 1$	$O(n \log n)$	$O(n(\log n)^{1.585})$	$O(n(\log n)^2)$
Gao [73]	GF(2^m), $m = 2^k$	$2^m - 1$	$O(n \log n \log \log n)$	$O(n \log n \log \log n)$	$O(n(\log n)^2 \log \log n)$
CFFTs	GF(2^m), m arbitrary	$2^m - 1$	$O(n(\log n)^{\log_2 \frac{3}{2}})$	$O(n^2/(\log n)^{\log_2 \frac{8}{3}})$	$O(n^2/(\log n)^{\log_2 \frac{8}{3}})$
Gao [83]	GF(2^m), m arbitrary	$2^m - 1$	$O(n \log n)$	$O(n(\log n)^2)$	$O(n(\log n)^2$
Wang [71], Cantor [84], Gathen [82]	GF(2^m), m arbitrary	$2^m - 1$	$O(n(\log n)^2)$	$O(n(\log n)^2)$	$O(n(\log n)^3)$
Cantor [72]	GF(2^m), m arbitrary	$2^m - 1$	$O(n^2)$	$O(n^2)$	$O(n^2 \log n)$

From Tab. 6.2, CFFTs have the smallest multiplicative complexities among all algorithms. and hence our results confirm the advantage of CFFTs in low multiplicative complexity. However, due to their high additive complexities, the additive and total complexities of CFFTs are asymptotically suboptimal. On the other hand, CFFTs and the fast DFT algorithm in [71] have no additional assumptions, while the other three algorithms all have additional constraints. Cantor's algorithm requires $n = 2^m - 1 = s^r$, but due to Mihăilescu's Theorem [86], the only way to satisfy this conditions is $n = n^1$. When $r = 1$, Cantor's algorithm has a quadratic additive and multiplicative complexities, and has no computational advantage. The other algorithms work only in the field GF(2^m) with $m = 2^K$.

Even though CFFTs have a suboptimal asymptotic total complexity, they remain very significant because they have the smallest total complexities for most practical lengths up to thousands of symbols over GF(2^m) with $m \leq 12$ [22]. The **only exception** is that for 255-point DFT over GF(2^8), Mateer's algorithm has the lowest total computational complexity, roughly 4% smaller than a 255-point CFFT.

6.5 Conclusion

CFFTs are of great importance due to their very low multiplicative complexities and small total complexities for DFTs with practical lengths. In this chapter, we generalize CFFTs from characteristic-2 fields to arbitrary finite fields, and analyze their computational complexities. To this end, we first propose an efficient algorithm for TMVPs that is instrumental in CFFTs. We then propose an efficient cyclic convolution algorithm over arbitrary finite fields. Since efficient cyclic convolution algorithm is the key to reducing the multiplicative complexities of CFFTs, our cyclic convolution algorithms for arbitrary finite fields enable us to generalize CFFTs to arbitrary finite fields. Finally, we analyze the computational complexities of CFFTs over arbitrary finite fields. Our results confirm that CFFTs have the smallest multiplicative complexities of all known algorithms, but they are asymptotically suboptimal due to their relatively high additive complexities. Nonetheless, they are still of great practical value because for DFTs of practical lengths, since they have the smallest total computational complexity in most cases.

Chapter 7

Composite Cyclotomic Fourier Transform with Reduced Additive Complexity

7.1 Introduction

In Chapter 6, we introduce a cyclic convolution algorithm which is the key to the CFFT implementation, and investigate the additive and multiplicative complexities of CFFTs in theory. However, our results show that the high additive complexity renders CFFT computationally inefficient for very long DFTs, e.g., 2047-point DFTs over GF(2^{11}) and 4095-point DFTs over GF(2^{12}). As Reed-Solomon codes [87, 88] over GF(2^{12}) with thousands of symbols have already been considered for hard drive [16] and tape storage [17] as well as optical communication systems [66] to achieve better error performance, and their syndrome based decoders require DFTs of lengths up to 4095 over GF(2^{12}). Therefore, efficient DFT algorithm is needed in practice. In addition to complexity, a modular structure is desirable for efficient hardware implementations of DFTs.

In the literature, fast Fourier transforms (FFTs) based on the prime-factor algorithm [23] and the Cooley-Tukey algorithm [24] have been proposed for DFTs over complex field. When FFTs based on the prime-factor algorithm are adapted to DFTs over finite fields [89], they still have high multiplicative complexities. In contrast, CFFTs [20,21] are promising since they have significantly lower multiplicative complexities while their high additive complexities are a drawback. In this chapter, we try to integrate CFFT with the prime-factor algorithm and the Cooley-Tukey algorithm, and our contributions are as follows.

- Due to the high additive complexities of CFFTs, we propose composite cyclotomic Fourier transforms (CCFTs). When the length n of a DFT is factored, i.e., $n = n_1 \times n_2$, CCFT uses n_1- and n_2-point CFFTs as sub-DFTs via the prime-factor and Cooley-Tukey algorithms. Thus, CFFTs are simply a special case of our CCFTs, corresponding to the trivial factorization, i.e., $n = 1 \times n$. This generalization reduces overall complexities in three ways. First, this divide-and-conquer strategy itself leads to lower complexities. Second, the moderate lengths of the sub-DFTs enable us to apply complexity-reducing techniques such as the CSE algorithm in [22] more effectively. Third, when the length n admits different factorizations, the one with the lowest complexity is selected. In the end, while an n-point CCFT may have a higher multiplicative complexity than an n-point CFFT, the former achieves a lower overall complexity for long DFTs because of its significantly lower additive complexity.

Moreover, when n is composite, an n-point CCFT has a modular structure, which is suitable for efficient hardware implementations. Our CCFTs provide a systematic approach to designing long DFTs with low complexity.

- Our efficient algorithms for cyclic convolutions devised in Sec. 6.3.2 allow us to obtain longer DFTs over larger fields. For example, we propose CFFTs over GF(2^{11}), which are unavailable in the literature heretofore partially due to the lack of efficient 11-point cyclic convolution algorithms. Our 2047-point DFTs over GF(2^{11}) and 4095-point DFTs over GF(2^{12}) are also first efficient DFTs of such lengths to the best of our knowledge, and they are promising for emerging communication systems.

Our work extends and improves previous works [20, 22] on CFFTs over finite fields of characteristic-2 in several ways. First, previously proposed CFFTs focus on $(2^m - 1)$-point CFFTs over GF(2^m) for $m \leq 10$. In contrast, our CCFTs allow us to derive long DFTs with low complexity over larger fields. Our approach can be applied to any finite field, but we present CCFTs over GF(2^{11}) and GF(2^{12}) due to their significance in applications. Furthermore, our work investigates n-point CFFTs over GF(2^m) for any n that divides 2^m-1, i.e., $n|2^m-1$. Second, our CCFTs achieve lower overall complexities than **all** previously proposed FFTs for moderate to long lengths, and the improvement significantly increases as the length grows.

The rest of this chapter is organized as follows. Sec. 7.2 briefly reviews the necessary background of this chapter, such as the prime-factor algorithm, the Cooley-Tukey algorithm, and the CSE algorithm. We then use an 11-point cyclic convolution algorithm devised in Sec. 6.3.2 to construct a 2047-point CFFT over GF(2^{11}) in Sec. 7.4, and discuss the difficulties of implementing very long CFFTs. We also propose our CCFTs and compare their complexities with previously proposed FFTs in Sec. 7.4. Concluding remarks are provided in Sec. 7.5.

7.2 Background

7.2.1 Common Subexpression Elimination

Given an $N \times M$ binary matrix $\mathbf{M}$ and an M-dimensional vector $\mathbf{x}$ over a field $\mathbb{F}$. The matrix vector multiplication $\mathbf{Mx}$ can be done by additions over $\mathbb{F}$ only, the number of which is denoted by $\mathcal{C}(\mathbf{M})$ since the complexity is determined by $\mathbf{M}$, when $\mathbf{x}$ is arbitrary. The problem of determining the minimal number of additions, denoted by $\mathcal{C}_{\text{opt}}(\mathbf{M})$, has been shown to be NP-complete [90].

Instead, different common subexpression elimination algorithms (see, e.g., [81, 91, 92]) have been proposed to reduce $\mathcal{C}(\mathbf{M})$. The CSE algorithm proposed in [22] takes advantage of the *differential savings* and *recursive savings*, and can greatly reduce the number of additions in calculating $\mathbf{Mx}$, although the reduced additive complexity, denoted by $\mathcal{C}_{\text{CSE}}(\mathbf{M})$, is not guaranteed to be the minimum. Like other CSE algorithms, the CSE algorithm in [22] is randomized, and the reduction results of different runs are not necessarily the same. Therefore in practice, a better result can be obtained by first running the CSE algorithm many times and then selecting the smallest number of additions. The CSE algorithm in [22] greatly reduces the additive and overall complexities of CFFTs with length up to 1023, but it is much more difficult to reduce the additive complexity of longer CFFTs. This is because though the CSE algorithm in [22] has a polynomial complexity (it is shown that its complexity is $O(N^4 + N^3M^3)$), the runtime and memory requirements become prohibitive when M and N are very large, which occurs for long CFFTs.

7.2.2 Prime-Factor and Cooley-Tukey Algorithms

We have reviewed DFT over finite field in Sec. 6.3.1. Both the prime-factor algorithm and the Cooley-Tukey algorithm first decompose an n-point DFT into shorter sub-DFTs, and then construct the n-point DFT from the sub-DFTs [25]. The prime-factor algorithm requires that the length n has at least two co-prime factors, i.e., there exist two co-prime numbers n_1 and n_2 such that $n = n_1 n_2$. For an integer $i \in \{0, 1, \cdots, n-1\}$, there is a unique integer pair (i_1, i_2) such that $0 \leq i_1 \leq n_1 - 1$, $0 \leq i_2 \leq n_2 - 1$, and $i = i_1 n_2 + i_2 n_1 \pmod{n}$, since n_1 and n_2 are co-prime. For any integer $j \in \{0, 1, \cdots, n-1\}$, let $j_1 = j \pmod{n_1}$, $j_2 = j \pmod{n_2}$, where $0 \leq j_1 \leq n_1 - 1$ and $0 \leq j_2 \leq n_2 - 1$. By the Chinese remainder theorem, (j_1, j_2) uniquely determines j, and j can be represented by $j = j_1 n_2^{-1} n_2 + j_2 n_1^{-1} n_1 \pmod{n}$, where $n_2^{-1} n_2 = 1 \pmod{n_1}$ and $n_1^{-1} n_1 = 1 \pmod{n_2}$. Substituting the above representation of i and j in (6.11), we get $\alpha^{ij} = (\alpha^{n_2})^{i_1 j_1} (\alpha^{n_1})^{i_2 j_2}$, where α^{n_2} and α^{n_1} are the n_1-th root and the n_2-th root of 1, respectively. Therefore, (6.11) becomes

$$F_j = \underbrace{\sum_{i_1=0}^{n_1-1} \Big(\overbrace{\sum_{i_2=0}^{n_2-1} f_{i_1 n_2 + i_2 n_1} \alpha^{n_1 i_2 j_2}}^{n_2-\text{point DFT}} \Big) \alpha^{n_2 i_1 j_1}}_{n_1-\text{point DFT}} . \tag{7.1}$$

In this way, the n-point DFT is obtained by using n_1- and n_2-point sub-DFTs. The n-point DFT result is derived by first carrying out n_1 n_2-point DFTs and n_2 n_1-point DFTs, and then combining the results according to the representation of j. The prime-factor algorithm can also be applied to n_1- and n_2-point DFTs if they have co-prime factors.

The Cooley-Tukey algorithm has a different decomposition strategy from the prime-factor algorithm. Let $n = n_1 n_2$, where n_1 and n_2 do not have to be co-prime. Let $i = i_1 + i_2 n_1$, where $0 \le i_1 \le n_1 - 1$ and $0 \le i_2 \le n_2 - 1$, and $j = j_1 n_2 + j_2$, where $0 \le j_1 \le n_1 - 1$ and $0 \le j_2 \le n_2 - 1$. Then (6.11) becomes

$$F_j = \underbrace{\sum_{i_1=0}^{n_1-1} \Big(\overbrace{\sum_{i_2=0}^{n_2-1} f_{i_1+i_2 n_1} \alpha^{n_1 i_2 j_2}}^{n_2-\text{point DFT}} \Big) \alpha^{i_1 j_2} \alpha^{n_2 i_1 j_1}}_{n_1-\text{point DFT}} . \tag{7.2}$$

In this way, the Cooley-Tukey algorithm also decomposes the n-point DFT into n_1- and n_2-point DFTs. However, compared with (7.1), (7.2) has an extra term $\alpha^{i_1 j_2}$, which is called a *twiddle factor* and incurs additional multiplicative complexity. The Cooley-Tukey algorithm can be used for arbitrary non-prime length n, including the prime powers to which case the prime-factor algorithm cannot be applied. The Cooley-Tukey algorithm is very suitable if n has a lot of small factors: for example, a 2^m-point DFT by the Cooley-Tukey algorithm requires $O(m \cdot 2^m)$ multiplications.

7.3 Long Cyclotomic Fourier Transforms

7.3.1 2047-point CFFT over GF(2^{11})

The 11-point cyclic convolution proposed in Appendix A is the key to CFFTs over GF(2^{11}). A direct implementation of a 2047-point CFFT with this cyclic convolution algorithm requires 7812 multiplications and 2130248 additions. The prohibitively high additive complexity is dominated by the multiplication between the 2047×2047

binary matrix $\mathbf{A}$ (see Sec. 6.3.1) and a 2047-dimensional vector, which requires 2095280 additions. Unfortunately, if we use the CSE algorithm in [22] to reduce its additive complexity, the time complexity of the CSE algorithm itself is too high. It may need months to finish.

Due to the high time complexity of the CSE algorithm in [22], we have tried a simplified CSE algorithm with limited success. In the original CSE algorithm in [22], only one of the patterns with the greatest recursive savings is selected and removed in each round of iterations. Instead of selecting only one pattern, our simplified CSE algorithm has a reduced time complexity as it removes multiple patterns at one time. The simplified CSE algorithm with a reduced time complexity allows us to reduce the additive complexity for the 2047-point CFFT to 529720 additions, about one fourth of that for the direct implementation. Despite this improvement, the effectiveness of this simplified CSE algorithm is rather limited.

7.3.2 Difficulty with Long CFFTs

Consider an n-point CFFT over GF(2^m). Let $C_0, C_1, \cdots, C_{k-1}$ be the k cyclotomic cosets modulo n over GF(2), and $|C_i| = m_i$. Suppose an m_i-point cyclic convolution can be done with $\mathcal{M}(m_i)$ multiplications, and hence directly implementing the n-point DFT with CFFT requires $\sum_{i=0}^{k-1} \mathcal{M}(m_i)$ multiplications and $\mathcal{C}(\mathbf{AQ}) + \mathcal{C}(\mathbf{P})$ additions, where $\mathcal{C}(\cdot)$ denotes the number of additions we need to evaluate the product of a binary matrix and a vector. The multiplicative complexity can be further reduced because we can pre-compute the vector $\mathbf{c}$ in (6.12) and some of its elements may be unitary. Then the CSE algorithm can be applied to the matrices $\mathbf{AQ}$ and $\mathbf{P}$ to reduce $\mathcal{C}(\mathbf{AQ})$ and $\mathcal{C}(\mathbf{P})$ to $\mathcal{C}_{\text{CSE}}(\mathbf{AQ})$ and $\mathcal{C}_{\text{CSE}}(\mathbf{P})$, respectively. Since $\mathbf{P} = \operatorname{diag}(\mathbf{P}_0, \mathbf{P}_1, \cdots, \mathbf{P}_{k-1})$ is a block diagonal matrix, we have $\mathcal{C}_{\text{CSE}}(\mathbf{P}) = \sum_{i=0}^{k-1} \mathcal{C}_{\text{CSE}}(\mathbf{P}_i)$. That is, we can reduce the additive complexity of each $\mathbf{P}_i$ to get a better result of $\mathcal{C}(\mathbf{P})$. Since the size of $\mathbf{P}_i$ is much smaller than that of $\mathbf{P}$, it allows us to run the CSE algorithm many times to achieve a smaller additive complexity. However, the matrix $\mathbf{AQ}$ is not a block diagonal matrix, and therefore we have to apply the CSE algorithm directly to $\mathbf{AQ}$. When the size of $\mathbf{AQ}$ is large, the CSE algorithm in [22] requires a lot of time and memory and hence it is impractical for extremely long DFTs. As mentioned above, it would take months for the CSE algorithm in [22] to reduce the additive complexity of 2047-point CFFT over GF(2^{11}), let alone 4095-point CFFTs over GF(2^{12}). The prohibitively high time complexity of the CSE algorithm in [22] and the limited effectiveness of the simplified CSE algorithm motivate our composite cyclotomic Fourier transforms.

7.4 Composite Cyclotomic Fourier Transforms

7.4.1 Composite Cyclotomic Fourier Transforms

Instead of simplifying the CSE algorithm or designing other low complexity optimization algorithms, we propose composite cyclotomic Fourier transforms (CCFTs) by first decomposing a long DFT into shorter sub-DFTs, via the prime-factor or the Cooley-Tukey algorithm, and then implementing the sub-DFTs by CFFTs. Note that both the decompositions require only that α is a primitive n-th root of 1, hence they can be extended to finite fields easily. When n is prime, our CCFTs reduce to CFFTs. When n is composite, we first decompose DFT into shorter sub-DFTs, and then combine the sub-DFT results according to (7.1) or (7.2). The shorter sub-DFTs are implemented by CFFTs to reduce their multiplicative complexities, and then we use the CSE algorithm in [22] to reduce their additive complexities. Finally, when n has multiple factors, the factorization can be carried out recursively.

Suppose the length of DFT is composite, i.e., $n = n_1 n_2$. Either the prime-factor algorithm or the Cooley-Tukey algorithm can be used to decompose the n-point DFT into sub-DFTs when n_1 and n_2 are co-prime. When n_1 and n_2 are not co-prime, only the Cooley-Tukey algorithm can be used. It is easy to show that if n_1 and n_2 are co-prime, the prime-factor and the Cooley-Tukey algorithms lead to the same additive complexity for CCFTs, but the latter results in a higher multiplicative complexity due to the twiddle factors. Hence the prime-factor algorithm is better than the Cooley-Tukey algorithm in this case, and the Cooley-Tukey algorithm is used only if the prime-factor algorithm cannot be applied.

We denote the multiplicative and additive complexities of an n-point DFT by $\mathcal{K}^{\text{mult}}(n)$ and $\mathcal{K}^{\text{add}}(n)$, respectively, and the algorithm used to implement this DFT is specified in the subscription of $\mathcal{K}$. Assuming that $n = \prod_{i=1}^{s} n_i$ and the total number of non-unitary twiddle factors required by the Cooley-Tukey algorithm decompositions is denoted by T, the complexity of this decomposition is given by

$$\mathcal{K}^{\text{add}}_{\text{CCFT}}(n) = \sum_{i=1}^{s} \frac{n}{n_i} \mathcal{K}^{\text{add}}_{\text{CFFT}}(n_i), \tag{7.3}$$

$$\mathcal{K}^{\text{mult}}_{\text{CCFT}}(n) = \sum_{i=1}^{s} \frac{n}{n_i} \mathcal{K}^{\text{mult}}_{\text{CFFT}}(n_i) + T. \tag{7.4}$$

For $n | 2^m - 1$, $4 \leq m \leq 12$, there is at most one pair of n_i's that are not co-prime in the decomposition of n, say n_1 and n_2, without loss of generality. In this case,

$T = \frac{n}{n_1 n_2}(n_1 - 1)(n_2 - 1)$. If all the elements in the decomposition of n are co-prime to each other, then $T = 0$.

The decomposition allows our CCFTs to achieve lower complexities for several reasons. First, this divide-and-conquer strategy is used in many fast Fourier transforms. If we assume CFFTs have quadratic additive complexities with their length n when directly implemented, the CCFT decomposition reduces the additive complexity from $O(n^2)$ to $O(n \sum_{i=1}^{s} n_i)$. Second, the lengths of the sub-DFTs are much shorter, which enables us to apply several powerful but complicated techniques to reduce the complexities of the sub-DFTs. For example, it takes much less time and memory to apply the CSE algorithm in [22] to the sub-DFTs, and thus we can run it multiple times to get a better reduction result. Third, when the length of the DFT admits different factorizations (for example, $2^6 - 1 = 63 = 3 \times 21 = 9 \times 7$), we choose the decomposition(s) with the lowest complexity.

7.4.2 Complexity Reduction

We reduce the additive and the overall complexities of our CCFTs in three steps. First, we reduce the complexities of short cyclic convolutions. Second, we use these short cyclic convolutions to construct CFFTs of moderate lengths. Third, we use CFFTs of moderate lengths as sub-DFTs to construct our CCFTs.

Complexity reduction of short cyclic convolutions

Efficient short cyclic convolution algorithms, such as the p-point reformulations we propose in Sec. 6.3.2, are the keys to the multiplicative complexity reduction of

CFFTs and our CCFTs. Suppose an L-point cyclic convolution $\mathbf{b}^{(L)} \otimes \mathbf{a}^{(L)}$ is calculated with the bilinear form $\mathbf{Q}^{(L)}(\mathbf{R}^{(L)}\mathbf{b}^{(L)} \cdot \mathbf{P}^{(L)}\mathbf{a}^{(L)})$. Since $\mathbf{b}^{(L)}$ is the normal basis in our CCFTs, $\mathbf{R}^{(L)}\mathbf{b}^{(L)}$ can be precomputed to reduce the multiplicative complexity. We apply the CSE algorithm in [22] to reduce the additive complexities in the multiplication with binary matrices $\mathbf{Q}^{(L)}$ and $\mathbf{P}^{(L)}$. The complexity reduction results $\mathcal{C}_{\text{CSE}}(\mathbf{Q}^{(L)})$, $\mathcal{C}_{\text{CSE}}(\mathbf{P}^{(L)})$, the total additive complexity $\mathcal{C}_{\text{CSE}}(\mathbf{Q}^{(L)}) + \mathcal{C}_{\text{CSE}}(\mathbf{P}^{(L)})$, and the multiplicative complexities are listed in Table 7.1. Note that the complexity of the 11-point cyclic convolution is derived from the algorithm in Appendix A.

Table 7.1: Complexities of short cyclic convolutions over GF(2^m).

L	mult.	additive complexities		
		$\mathcal{C}_{\text{CSE}}(\mathbf{Q}^{(L)})$	$\mathcal{C}_{\text{CSE}}(\mathbf{P}^{(L)})$	total
2	1	2	1	3
3	3	5	4	9
4	5	9	4	13
5	9	16	10	26
6	10	21	11	32
7	12	24	23	47
8	19	35	16	51
9	18	40	31	71
10	28	52	31	83
11	42	76	44	120
12	32	53	34	87

Additive complexity reduction of CFFTs with moderate lengths

Blocks of CFFTs with moderate lengths are used to build our CCFTs. Their moderate lengths allow us to use multiple techniques to reduce their additive complexities.

- First, for any CFFT, we run the CSE algorithm in [22] multiple times and then choose the best results.

- Second, for each CFFT in (6.12), we may reduce $\mathcal{C}(\mathbf{AQ})$ together as a whole, or reduce $\mathcal{C}(\mathbf{A})$ and $\mathcal{C}(\mathbf{Q})$ separately. Since $(\mathbf{AQ})\mathbf{v} = \mathbf{A}(\mathbf{Qv})$, $\mathcal{C}_{\text{opt}}(\mathbf{AQ}) \leq \mathcal{C}_{\text{opt}}(\mathbf{A})+\mathcal{C}_{\text{opt}}(\mathbf{Q})$. However, this property may not hold for the CSE algorithm because it may not find the optimal solutions. Furthermore, we may benefit from reducing $\mathcal{C}(\mathbf{A})$ and $\mathcal{C}(\mathbf{Q})$ separately for the following reasons. First, $\mathbf{Q}$ has a block diagonal structure, which is similar as $\mathbf{P}$, and we can find a better reduction result for $\mathcal{C}(\mathbf{Q})$. Second, $\mathbf{AQ}$ has much more columns than $\mathbf{A}$, and hence the CSE algorithm requires less memory and time to reduce $\mathbf{A}$ than to reduce $\mathbf{AQ}$.

- Third, there is flexibility in terms of normal bases used to construct the matrix $\mathbf{A}$ in (6.12), and this flexibility can be used to further reduce the additive complexity of any CFFT. For each cyclotomic coset, a normal basis is needed. A normal basis is not unique in finite fields, and any normal basis can be used in the construction of the matrix $\mathbf{A}$, leading to the same multiplicative complexity. But different normal bases result in different $\mathbf{A}$s and hence different additive complexities due to $\mathbf{A}$. There are several options regarding the normal basis. One can simply choose a fixed normal basis for all cyclotomic cosets of the same size as in [22]. A more ideal option is to enumerate all possible normal bases and their corresponding $\mathbf{A}$s and to select the smallest additive complexity. However, when the underlying field is large, the number of possible normal bases is very large, and hence it becomes infeasible to enumerate all possible constructions. Thus, we use a compromise of these two options: for each cyclotomic coset we choose a normal basis at random and the combination of random normal bases leads to $\mathbf{A}$; we minimize the complexity over as many combinations as complexity permits. We refer to this as a random normal basis option.

We emphasize that all three techniques require multiple runs of the CSE algorithm. Since the time and memory requirements of the CSE algorithm grow with the length of DFT, the moderate length of the sub-DFTs is the key enabler of these techniques. Though the CSE algorithm may be costly for moderate length CFFTs, it is a one-time task.

For any $n \leq 320$ so that $n|2^m - 1$ $(4 \leq m \leq 12)$, the multiplicative and additive complexities of the n-point CFFT are shown in Table 7.2. Table 7.2 shows four different schemes to reduce the additive complexity for CFFTs. Schemes A and B both use the fixed normal basis option in the construction of the matrix $\mathbf{A}$, while schemes C and D are based on the random normal basis option. Schemes A and C reduce $\mathcal{C}(\mathbf{A})$ and $\mathcal{C}(\mathbf{Q})$ separately, while schemes B and D reduce $\mathcal{C}(\mathbf{AQ})$ as a whole. For smaller CFFTs, we typically minimize the complexity over hundreds of combinations of normal bases, and fewer combinations for longer CFFTs. In Table 7.2, the smallest additive complexities are in a boldface font. We observe that the random normal basis option offers further additive complexity reduction in most of the cases. However, since the fixed normal basis is not necessarily one of the combinations, in some cases the fixed normal basis option outperforms the random normal basis option. Also, sometimes applying the CSE to $\mathbf{AQ}$ together as a whole leads to lower complexity, and in some cases it is better to apply the CSE to $\mathbf{A}$ and $\mathbf{Q}$ separately.

Table 7.2: The complexities of the CFFTs whose lengths are less than 320 and are factors of $2^m - 1$ for $1 \leq m \leq 12$.

n	l	mult.	additive complexities			
			A	B	C	D
3	2	1	**6**	**6**	**6**	**6**
5	4	5	20	**16**	20	**16**
7	3	6	31	**24**	31	**24**
9	6	11	51	**48**	51	**48**
11	10	28	109	102	102	**84**
13	12	32	125	100	110	**91**
15	4	16	87	**74**	87	**74**
17	8	38	153	163	**151**	153
21	6	27	167	179	**147**	153
23	11	84	335	407	**323**	357
31	5	54	354	**299**	335	350
33	10	85	413	440	**404**	434
35	12	75	406	303	358	**299**
39	12	97	502	425	472	**391**
45	12	90	481	415	498	**414**
51	8	115	**641**	755	676	739
63	6	97	798	**759**	806	1031
65	12	165	1092	**901**	1114	915
73	9	144	1498	1567	**1447**	1526
85	8	195	1601	1816	**1589**	1810
89	11	336	**2085**	4326	2247	3973
91	12	230	1668	1431	1596	**1421**
93	10	223	1772	1939	**1736**	1788
105	12	234	1762	1481	1776	**1333**
117	12	299	2304	2028	2366	**1947**
195	12	496	4900	4230	4942	**4166**
273	12	699	8064	**7217**	8082	7223
315	12	752	8965	**8032**	9899	8099

Construction of CCFTs using moderate-length CFFTs as sub-DFTs

We use the CFFTs with moderate lengths in Table 7.2 as sub-DFTs to construct our CCFTs. With (7.3) and (7.4), the computational complexities of our CCFTs over GF(2^m) ($4 \leq m \leq 12$) with non-prime lengths can be calculated. The results are summarized in Table 7.3, where the factorizations in parentheses are not co-prime and the Cooley-Tukey algorithm is used in these cases. We have tried all the decompositions with lengths smaller than 320, and the decompositions with the smallest overall complexities are listed in Table 7.3. Note that for each sub-DFT, the scheme with the smallest additive complexity listed in Table 7.2 is used in the CCFT implementation to reduce the total additive complexity. We also note that all DFT lengths in Table 7.3 are composite. The prime lengths are omitted because in these cases, a CCFT reduces to a CFFT, which can be found in Table 7.2.

Since some lengths of the DFTs have different decompositions, it is possible that one decomposition has a smaller additive complexity but a larger multiplicative complexity than another one. Therefore, we need a metric to compare the overall complexities between different decompositions. We follow [22] and assume that the complexity of a multiplication over GF(2^m) is $2m-1$ times of that of an addition over the same field, and the total complexity of a DFT is a weighted sum of the additive and multiplicative complexities, i.e., total $= (2m-1) \times \text{mult} + \text{add}$. This assumption is based on both the software and hardware implementation considerations [22]. Table 7.3 lists the decompositions with the smallest overall complexities.

Table 7.3 provide complexities of all n-point DFTs over GF(2^m) when $n|2^m - 1$ and $4 \leq m \leq 12$. Note that the decomposition corresponding to $1 \times n$ is merely the n-point CFFT over GF(2^m). We have used the simplified CSE algorithm described in Sec. 7.3.1 to reduce the complexity of the 2047-point CFFTs over GF(2^{11}), and applied the CSE algorithm in [22] to the other CFFTs. Thus, we have expanded the results of [22], where only the ($2^m - 1$)-point CFFTs over GF(2^m) were given. We also observe that for some short lengths (see, for example, n = 15, 33, or 65), the

Table 7.3: The smallest complexity of our n-point CCFTs over GF(2^m) for composite n and $n|2^m - 1$ for $4 \leq m \leq 12$ (we assume the sub-DFTs are shorter than 320).

l	n	Decomposition	mult.	add.	total
4	15	1×15	16	74	186
6	9	(3×3)	10	36	146
	21	3×7	25	114	389
	63	$(3 \times 3) \times 7$	124	468	1832
8	51	1×51	115	641	2366
	85	1×85	195	1590	4515
	255	3×85	670	5277	15327
9	511	7×73	1446	11881	36463
10	33	1×33	85	404	2019
	93	3×31	193	1083	4750
	341	1×341	922	15184	32702
	1023	33×31	4417	22391	106314
11	2047	23×89	15204	76702	395986
12	35	5×7	65	232	1727
	39	1×39	97	391	2622
	45	(3×15)	91	312	2405
	65	1×65	165	902	4697
	91	1×93	230	1421	6711
	105	7×15	202	878	5524
	117	1×117	299	1947	8824
	195	3×65	560	3093	15973
	273	3×91	781	4809	22772
	315	5×63	800	4803	23203
	455	7×65	1545	7867	43402
	585	5×117	2080	11607	59447
	819	7×117	2795	16437	80722
	1365	7×195	4642	33842	140608
	4095	65×63	16700	106098	490198

n-point CFFTs lead to the lowest complexity for the n-point CCFTs. For the DFTs longer than 320, i.e., 511-point CFFTs over GF(2^9), 341-point CFFTs over GF(2^{10}), and 455-, 585-, 819- and 1365-point CFFTs over GF(2^{12}), the time complexity of the CSE algorithm in [22] is still considerable, and we cannot minimize their complexities using schemes A, B, C, and D, and hence they are not listed in Table 7.2.

Table 7.4: Comparison of the complexities of our n-point CCFTs with FFTs available in the literature.

n	Field	Wang and Zhu [71]			Trung et al. [89]			CFFT					CCFT		
		mult.	add.	total	mult.	add.	total	mult.	w/o CSE		w/ CSE [22]		mult.	add.	total
									add.	total	add.	total			
15	GF(2^4)	41	97	384	–	–	–	16	201	313	74	**186**	20	78	218
63	GF(2^6)	801	801	9612	–	–	–	97	2527	3594	759	**1826**	124	468	1832
255	GF(2^8)	1665	5377	30352	1135	3887	20902	586	34783	43573	6736	15526	670	5277	**15327**
511	GF(2^9)	13313	13313	239634	6516	17506	128278	1014	141710	158948	23130	40368	1446	11881	**36463**
1023	GF(2^{10})	32257	32257	645140	5915	30547	142932	2827	536093	589806	75360	129073	4417	22391	**106314**
2047	GF(2^{11})	78601	78601	1689622	–	–	–	7812	2130248	2294300	–	–	15204	76702	**395986**
4095	GF(2^{12})	180225	180225	4325400	–	–	–	10832	8434414	8683550	–	–	16700	106098	**490198**

Though the twiddle factors in the Cooley-Tukey decomposition incur extra multiplicative complexity, Table 7.3 show that the Cooley-Tukey decomposition reduces the total complexity of our CCFTs in some cases (the decompositions in parentheses). For example, a 9-point CFFT requires 11 multiplications and 48 additions, and a 3×3 CCFT based on the Cooley-Tukey decomposition requires 10 multiplications and 36 additions. Despite the twiddle factors, the CCFT based on the Cooley-Tukey decomposition have lower multiplicative and additive complexities, because we can take advantage of the low complexity of the 3-point DFT.

7.4.3 Complexity Comparison and Analysis

We compare the complexities of our CCFTs with those of previously proposed FFTs in the literature in Table 7.4. For each length, the lowest total complexity is in boldface font. In Table 7.4, our CCFTs achieve the lowest complexities for $n \geq 255$. Although the algorithm in [71] is proved asymptotically fast, the complexities of our CCFTs are only a fraction of those in [71], and the advantage grows as the length increases. Although the FFTs in [89] are also based on the prime-factor algorithm, our CCFTs achieve lower complexities for two reasons. First, since our CCFTs use CFFTs as the sub-DFTs, the multiplicative complexities of our CCFTs are greatly reduced compared with the FFTs in [89]. For example, the multiplicative complexity of our 511-point CCFT is only one fourth of the prime-factor algorithm in [89]. Furthermore, using the powerful CSE algorithm in [22], the additive complexities of our CCFTs are also greatly reduced. Compared with the CFFTs, our CCFTs have somewhat higher multiplicative complexities, but this is more than made up by their reduced additive complexities. The additive complexities of our CCFTs are only a small fraction of those of CFFTs when directly implemented. Compared with the CFFTs with reduced additive complexities in [22], our CCFTs still have much smaller additive complexities due to their decomposition structure for $n \geq 63$. For example, the additive complexities of our CCFT is only about half of that of the CFFT for $n = 511$, and one third for $n = 1023$. Due to the significant reduction of the additive complexities, the total complexities of our CCFTs with $n \geq 255$ are lower than those of CFFTs. In comparison to CFFTs, the improvement by our CCFTs also grows as the length increases. For the prime-length DFTs, such as the 31-point DFT over GF(2^5), 127-point DFT over GF(2^7), and 8191-point DFT over GF(2^{13}), our CCFTs reduce to the CFFTs, and they have the same complexities.

In the end, we remark that our CCFT is built from shorter DFTs, and short DFTs can be used as modules in the hardware design. We may either pipeline them to achieve a high speed, or reuse them to save the chip area. This modular structure is favorable to hardware implementations.

7.5 Summary

We propose a novel composite cyclotomic Fourier transform algorithm that leads to lower complexities through decomposing long DFTs into shorter ones using the prime-factor or Cooley-Tukey algorithm. Our CCFTs over GF(2^m) ($8 \leq m \leq 12$) have lower complexities than previously known FFTs over finite fields. They also have a modular structure, which is desirable in hardware implementations.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

In this dissertation, we investigate efficient signal processing algorithms in several areas. These algorithms are divided into three categories, CACs for on-chip global buses in deep sub-micron technology, MIMO detection, and efficient DFTs over finite fields. We design efficient algorithms that are favorable in hardware implementations because of their low computational complexity and modular structure. We provide simulation and hardware implementation results to demonstrate the advantages of our algorithms. We also provide theoretical complexity analysis for most of these algorithms. Our work was published in the following conference and journal papers [85, 93–101]. We briefly summarize our main contributions in this dissertation as follows.

In Chapter 2, we generalize the CODEC designed in [5] and [6] and propose a generic CAC CODEC framework based on binary mix-radix numeral system. Based on this framework, we devise efficient CODECs for OLCs, FPCs, and FOCs by choosing appropriate numeral systems and constants. Our CODECs have areas and delays increasing quadratically with the bus width in theory, which is also shown by the hardware implementation results. Our efficient CODECs can also be integrated with the partial coding technology, making CACs a viable option in combating crosstalk delay.

In Chapter 3, we propose the TDCAC technology, whose encoding and decoding are not only done in the spatial domain, but also in the temporal domain. We investigate the TDCACs with and without memory, respectively, which reduces the crosstalk delay in the worst case to $(1 + 2\lambda)\tau_0$. We also derive the codebook sizes of the TDCACs with memory when $n \leq 15$, and propose a conjecture for the cases when $n > 15$. The simulation results show that both the TDCACs with and without memory have higher code rates than the FTCs and FPCs. The CODEC of the TDCACs can be simple and efficient if designed properly, which is shown by an example, a 3×3 TDCAC with a code rate of $\frac{7}{9}$.

In Chapter 4, we apply some novel ordering schemes to the K-Best detector, and show that the detection error rate of the K-Best detector is improved by the BSQRD ordering scheme as well as the V-BLAST versions of the BSQRD and SB-SQRD ordering schemes. As the reliability of the K-Best detector is improved, these ordering schemes lead to smaller detection error rates, reduce the SNR required, or allow even a smaller value for K to reduce the hardware complexity. Our hardware implementations show that our BSQRD and SBSQRD algorithms incur roughly 10% overhead in comparison with the hardware implementation of the sorted QRD algorithm, and they can achieve high throughput with our pipelined architecture at the expense of greater gate counts.

In Chapter 5, we propose a list-based soft-decision MIMO detection algorithm to find a list of ML candidates based on the MCTS strategy [45]. Our simulation results show that our LMCTS algorithm achieves a flexible balance between the memory requirement and the computational complexity, which can be tuned by changing its available memory size. Our hardware implementation of the LMCTS algorithm shows its feasibility. Although our implementation has a low throughput due to its sequential architecture, it has a potentiality to achieve a high throughput with the flexible balance between memory requirement and computational complexity .

In Chapter 6, we first propose an efficient algorithm for TMVPs that is instrumental in CFFTs, and then propose an efficient cyclic convolution algorithm over arbitrary finite fields. Our cyclic convolution algorithm is the key to reducing the multiplicative complexities of the CFFTs, and hence it enables us to generalize the CFFTs from characteristic-2 fields to arbitrary finite fields. Then we analyze the additive and multiplicative complexities of the CFFTs over arbitrary finite fields in theory. Our results confirm that the CFFTs have the smallest multiplicative complexities of all known algorithms, but they are asymptotically suboptimal because of their relatively high additive complexities. Nonetheless, they are of great value because for DFTs of practical lengths, they still have the smallest total computational complexity in most cases.

In Chapter 7, we propose a novel composite cyclotomic Fourier transform algorithm leading to lower overall computational complexities. This algorithm integrates the ideas of the prime-factor algorithm and the Cooley-Tukey algorithm as well as the idea of CFFTs. With efficient cyclic convolution algorithms, our CFFTs over

GF(2^l) ($8 \leq l \leq 12$) have lower complexities than previously known FFTs over finite fields. The modular structure of our CCFTs is also favorable in hardware implementations.

8.2 Future Work

For future work, the following point may be worthy to be looked into:

- The CAC CODECs we design in Chapter 2 and the TDCACs we propose in Chapter 3 are all based on the delay model (2.1) in [26]. However, this model has several drawbacks. First of all, its accuracy is limited, which can be seen from simulations. Secondly, it assumes that only the transitions on the two adjacent wires affect the transition delay on the middle wire, which is no longer valid in the modern deep sub-micron technology. Moreover, it does not consider the crosstalk caused by mutual inductance between wires, which could be a significant problem as feature shrinks. Therefore, we can derive more accurate delay models by considering more adjacent wires and mutual inductance. We already have some preliminary results in [102]. In this paper, we derive a delay model by considering the capacitive crosstalk from four adjacent wires, and then classify the 5-bit transition patterns. Based on this novel transition pattern classification, we can design new families of CACs, and their CODECs should be designed accordingly.

- The CAC CODECs we design in Chapter 2 are based on mixed binary numeral systems. We find those numeral systems used in different families of CAC CODECs in an *ad hoc* way. We are not sure if we can find other numeral systems to encode and decode the novel CAC families in [102], and it is also of great interest to find the necessary and sufficient condition for the existence of such numeral systems. Furthermore, as the CACs are all non-linear codes [31], this approach may also lead to efficient CODEC for other non-linear codes, e.g., the modulation codes in [103].

- The TDCACs we design in Chapter 3 only reduce the transition delay to $(1+2\lambda)\tau_0$, and other transition delay reduction targets need to be looked into. As we analyze in Chapter 3, the codebook size of TDCACs with memory is limited by the vectors with the smallest number of code matrices these vectors could transition to. If we could remove such vectors, we may derive a larger codebook, and this technology is called *pruning*. The pruning technology can effectively increase the codebook size of the one dimensional CAC [28], but its effects to the TDCACs are unknown, which should also be examined in future work. Furthermore, although we can show that the CODECs of TDCACs can be efficiently implemented by a small example, the efficient CODECs of TDCACs are unknown in general. This is another point that is worthy to be investigated.

- The hardware implementation of the LMCTS algorithm we design in Chapter 5 has a much lower throughput than the other list detection implementations in literature, although we are using a more advanced 45 nm process. This is mainly because our implementation has a sequential architecture and makes no trade-off between the detection error rate performance and the throughput. A more practical LMCTS detector for MIMO communication systems should be implemented to demonstrate the advantages of the LMCTS algorithm.

- Although the additive complexities of CFFTs can be effectively reduced by the CSE algorithm in [22], the reduction results always have an irregular structure and therefore incur some difficulty in hardware implementation, especially for very long CFFTs. The analysis of the additive complexities of the CFFTs in Chapter 6 actually suggests a novel method to reduce the additive complexity of the CFFTs. Due to its modular structure, this approach may lead to an efficient implementation of the additive networks in the CFFTs. However, hardware implementation is needed to demonstrate this advantage.

- The CCFTs we propose in Chapter 7 have the smallest computational complexities for DFTs over GF(2^m) with $8 \leq m \leq 12$. Their modular structures are also favorable in the hardware implementation. We should also demonstrate this advantage by hardware implementation. Furthermore, as DFTs are important in the communication systems, we can apply our CCFT to these systems, e.g., a syndrome based decoder for RS code over GF(2^{12}).

Bibliography

[1] [Online]. Available: http://www.itrs.net/Links/2005ITRS/Home2005.htm

[2] B. Victor and K. Keutzer, "Bus encoding to prevent crosstalk delay," in *IEEE /ACM International Conference on Computer Aided Design*, 2001, pp. 57–63.

[3] C. Duan and A. Tirumala, "Analysis and avoidance of cross-talk in on-chip buses," in *Proceedings of the Ninth Symposium on High Performance Interconnects*, 2001, p. 133.

[4] S. R. Sridhara, "Communication inspired design of on-chip buses," Ph.D. dissertation, Dept. of Electrical and Computer Engineering, University of Illinois, Urbana, IL, Jan. 2006.

[5] C. Duan, C. Zhu, and S. P. Khatri, "Forbidden transition free crosstalk avoidance CODEC design," in *Proc. 45th ACM/IEEE Design Automation Conf.*, 2008, pp. 986–991.

[6] C. Duan, V. H. C. Calle, and S. P. Khatri, "Efficient on-chip crosstalk avoidance CODEC design," *IEEE Trans. VLSI Syst.*, vol. 17, no. 4, pp. 551–560, April 2009.

[7] E. Viterbo and J. Boutros, "A universal lattice code decoder for fading channels," *IEEE Trans. Inf. Theory*, vol. 45, no. 5, pp. 1639–1642, Jul. 1999.

[8] O. Damen, A. Chkeif, and J.-C. Belfiore, "Lattice code decoder for space-time codes," *IEEE Commun. Lett.*, vol. 4, no. 5, pp. 161–163, May 2000.

[9] A. M. Chan and I. Lee, "A new reduced-complexity sphere decoder for multiple antenna systems," in *Proc. IEEE International Conference on Communications*, vol. 1, Apr. 28–May 2, 2002, pp. 460–464.

[10] E. Agrell, T. Eriksson, A. Vardy, and K. Zeger, "Closest point search in lattices," *IEEE Trans. Inf. Theory*, vol. 48, no. 8, pp. 2201–2214, Aug. 2002.

[11] M. O. Damen, H. El Gamal, and G. Caire, "On maximum-likelihood detection and the search for the closest lattice point," *IEEE Trans. Inf. Theory*, vol. 49, no. 10, pp. 2389–2402, Oct. 2003.

[12] W. Xu, Y. Wang, Z. Zhou, and J. Wang, "A computationally efficient exact ML sphere decoder," in *Proc. IEEE Global Telecommunications Conference*, vol. 4, Nov. 29–Dec. 3, 2004, pp. 2594–2598.

[13] Y. Dai and Z. Yan, "Memory-constrained tree search detection and new ordering schemes," *IEEE J. Sel. Topics Signal Process.*, vol. 3, no. 6, pp. 1026 – 1037, December 2009.

[14] B. M. Hochwald and S. ten Brink, "Achieving near-capacity on a multiple-antenna channel," *IEEE Trans. Commun.*, vol. 51, no. 3, pp. 389–399, March 2003.

[15] S. Baro, J. Hagenauer, and M. Witzke, "Iterative detection of MIMO transmission using a list-sequential (LISS) detector," in *Proc. IEEE International Conference on Communications*, vol. 4, May 11–15, 2003, pp. 2653–2657.

[16] "Hard disk drive long data sector white paper," April 20 2007. [Online]. Available: http://www.idema.org/

[17] Y. Han, W. E. Ryan, and R. Wesel, "Dual-mode decoding of product codes with application to tape storage," in *Proc. IEEE Global Telecommunications Conference*, vol. 3, Nov. 28–Dec. 2, 2005, pp. 1255–1260.

[18] G. Van Meerbergen, M. Moonen, and H. De Man, "Reed-Solomon codes implementing a coded single-carrier with cyclic prefix scheme," *IEEE Trans. Commun.*, vol. 57, no. 4, pp. 1031–1038, 2009.

[19] T. Kerins, E. M. Popovici, and W. P. Marnane, "Fully paramaterisable Galois field arithmetic processor over GF(3^m) suitable for elliptic curve cryptography," in *Proc. 24th Int Microelectronics Conf*, vol. 2, 2004, pp. 739–742.

[20] P. V. Trifonov and S. V. Fedorenko, "A method for fast computation of the Fourier transform over a finite field," *Probl. Inf. Transm.*, vol. 39, no. 3, pp. 231–238, 2003.

[21] S. V. Fedorenko, "A method for computation of the discrete Fourier transform over a finite fields," *Probl. Inf. Transm.*, vol. 42, pp. 139–151, 2006.

[22] N. Chen and Z. Yan, "Cyclotomic FFTs with reduced additive complexities based on a novel common subexpression elimination algorithm," *IEEE Trans. Signal Process.*, vol. 57, no. 3, pp. 1010–1020, Mar. 2009.

[23] I. J. Good, “The interaction algorithm and practical Fourier analysis,” *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 20, no. 2, pp. 361–372, 1958. [Online]. Available: http://www.jstor.org/stable/2983896

[24] J. W. Cooley and J. W. Tukey, “An algorithm for the machine calculation of complex Fourier series,” *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965.

[25] R. E. Blahut, *Fast Algorithms for Digital Signal Processing.* Addison-Wesley, 1985.

[26] P. P. Sotiriadis, “Interconnect modeling and optimization in deep sub-micron technologies,” Ph.D. dissertation, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 2002.

[27] K. Hirose and H. Yasuura, “A bus delay reduction technique considering crosstalk,” in *Proc. Design Automation and Test in Europe Conference and Exhibition*, 2000, pp. 441–445.

[28] B. Victor, “Bus encoding to prevent crosstalk delay,” Master’s thesis, Dept. of Electrical Engineering and Computer Science, University of California, Berkeley, May 2001.

[29] P. P. Sotiriadis and A. Chandrakasan, “Reducing bus delay in submicron technology using coding,” in *Proceedings of the 2001 Conference on Asia South Pacific design automation*, 2001, pp. 109–114.

[30] S. R. Sridhara, A. Ahmed, and N. R. Shanbhag, "Area and energy-efficient crosstalk avoidance codes for on-chip buses," in *IEEE International Conference on Computer Design: VLSI in Computers and Processors*, 2004, pp. 12–17.

[31] S. R. Sridhara and N. R. Shanbhag, "Coding for reliable on-chip buses: A class of fundamental bounds and practical codes," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 5, pp. 977–982, May 2007.

[32] C. Duan and S. P. Khatri, "Exploiting crosstalk to speed up on-chip buses," in *Proc. Design, Automation and Test in Europe Conference and Exhibition*, vol. 2, 16–20 Feb. 2004, pp. 778–783.

[33] D. Knuth, *The Art of Computer Programming*, 3rd ed. Addison-Wesley, 1997, vol. 2.

[34] [Online]. Available: http://www.model.com/

[35] [Online]. Available: http://www.cadence.com/eu/Pages/rtl_compiler.aspx

[36] [Online]. Available: http://vcag.ecen.okstate.edu/projects/scells/

[37] R. Peeters, "The maximum-edge biclique problem is NP-complete," 2000. [Online]. Available: citeseer.ist.psu.edu/peeters00maximum.html

[38] K. J. Kim and R. A. Iltis, "Joint detection and channel estimation algorithms for QS-CDMA signals over time-varying channels," *IEEE Trans. Commun.*, vol. 50, no. 5, pp. 845–855, May 2002.

[39] Z. Lei, Y. Dai, and S. Sun, "A low complexity near ML V-BLAST algorithms," in *Proc. VTC-2005-Fall Vehicular Technology Conference*, vol. 2, Sep. 25–28, 2005, pp. 942–946.

[40] D. Seethaler, H. Artes, and F. Hlawatsch, "Dynamic nulling-and-canceling for efficient near-ML decoding of MIMO systems," *IEEE Trans. Signal Process.*, vol. 54, no. 12, pp. 4741–4752, Dec. 2006.

[41] K. Su and I. J. Wassell, "A new ordering for efficient sphere decoding," in *Proc. IEEE International Conference on Communications*, vol. 3, May 16–20, 2005, pp. 1906–1910.

[42] A. D. Murugan, H. El Gamal, M. O. Damen, and G. Caire, "A unified framework for tree search decoding: rediscovering the sequential decoder," *IEEE Trans. Inf. Theory*, vol. 52, no. 3, pp. 933–953, Mar. 2006.

[43] K. Wong, C. Tsui, R. S. Cheng, and W. Mow, "A VLSI architecture of a K-Best lattice decoding algorithm for MIMO channels," in *Proc. IEEE International Symposium on Circuits and Systems*, vol. 3, May 26–29, 2002, pp. III–273–I276.

[44] Z. Guo and P. Nilsson, "Algorithm and implementation of the K-Best sphere decoding for MIMO detection," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 3, pp. 491–503, Mar. 2006.

[45] Y. Dai and Z. Yan, "Memory-constrained ML-optimal tree search detection," in *Proc. 42nd Annual Conference on Information Sciences and Systems*, Mar. 19–21, 2008, pp. 1037–1041.

[46] P. W. Wolniansky, G. J. Foschini, G. D. Golden, and R. A. Valenzuela, "V-BLAST: an architecture for realizing very high data rates over the rich-scattering wireless channels," in *Proc. URSI International Symposium on Signals, Systems, and Electronics*, Sep. 29–Oct. 2, 1998, pp. 295–300.

[47] K. Su, "Efficient maximum likelihood detection for communication over multiple input multiple output channels," Ph.D. dissertation, University of Cambridge, 2005.

[48] D. Wubben, R. Bohnke, J. Rinas, V. Kuhn, and K. D. Kammeyer, "Efficient algorithm for decoding layered space-time codes," *Electronics Letters*, vol. 37, no. 22, pp. 1348–1350, Oct. 25, 2001.

[49] A. Wiesel, X. Mestre, A. Pages, and J. R. Fonollosa, "Efficient implementation of sphere demodulation," in *Proc. 4th IEEE Workshop on Signal Processing Advances in Wireless Communications*, Jun. 15–18, 2003, pp. 36–40.

[50] A. Burg, M. Borgmann, M. Wenk, M. Zellweger, W. Fichtner, and H. Bolcskei, "VLSI implementation of MIMO detection using the sphere decoding algorithms," *IEEE J. Solid-State Circuits*, vol. 40, no. 7, pp. 1566–1577, Jul. 2005.

[51] X. Huang, C. Liang, and J. Ma, "System architecture and implementation of MIMO sphere decoders on FPGA," *IEEE Trans. VLSI Syst.*, vol. 16, no. 2, pp. 188–197, Feb. 2008.

[52] Y. Dai and Z. Yan, "Efficient ordering schemes for sphere decoder," in *Proc. IEEE Workshop on Signal Processing Systems*, Oct. 8–10, 2008, pp. 146–151.

[53] P. Luethi, C. Studer, S. Duetsch, E. Zgraggen, H. Kaeslin, N. Felber, and W. Fichtner, "Gram-Schmidt-based QR decomposition for MIMO detection: VLSI implementation and comparison," in *Proc. IEEE Asia Pacific Conference on Circuits and Systems*, Nov. 2008, pp. 830–833.

[54] G. Knagge, M. Bickerstaff, B. Ninness, S. R. Weller, and G. Woodward, "A VLSI 8×8 MIMO near-ML decoder engine," in *Proc. IEEE Workshop on Signal Processing Systems Design and Implementation*, Oct. 2–4, 2006, pp. 387–392.

[55] P. Luethi, A. Burg, S. Haene, D. Perels, N. Felber, and W. Fichtner, "VLSI implementation of a high-speed iterative sorted MMSE QR decomposition," in *Proc. IEEE International Symposium on Circuits and Systems*, May 27–30, 2007, pp. 1421–1424.

[56] G. H. Golub and C. F. Van Loan, *Matrix computations.* Baltimore, MD, USA: Johns Hopkins University Press, 1996.

[57] J. E. Volder, "The CORDIC trigonometric computing technique," *IEEE Trans. Electron. Comput.*, no. 3, pp. 330–334, Sep. 1959.

[58] P. K. Meher, J. Valls, T.-B. Juang, S. K., and K. Maharatna, "50 years of CORDIC: Algorithms, architectures, and applications," *IEEE Trans. Circuits Syst. I*, vol. 56, no. 9, pp. 1893 – 1907, September 2009.

[59] *IEEE 802.11n-2009, Amendment 5: Enhancements for Higher Throughput*, IEEE Std., Oct. 29 2009.

[60] [Online]. Available: http://www.ist-mascot.org/

[61] J. Jalden and B. Ottersten, "On the complexity of sphere decoding in digital communications," *IEEE Trans. Signal Process.*, vol. 53, no. 4, pp. 1474–1484, Apr. 2005.

[62] J. Hagenauer and C. Kuhn, "The list-sequential (LISS) algorithm and its application," *IEEE Trans. Commun.*, vol. 55, no. 5, pp. 918–928, May 2007.

[63] M. Wenk, M. Zellweger, A. Burg, N. Felber, and W. Fichtner, "K-Best MIMO detection VLSI architectures achieving up to 424 Mbps," in *Proc. IEEE International Symposium on Circuits and Systems*, 2006, pp. 4pp.–1154.

[64] J. Lee and S.-C. Park, "Area efficient pipelined VLSI implementation of list sphere decoder," in *Proc. Asia-Pacific Conference on Communications*, Aug. 2006, pp. 1–5.

[65] M. Wenk, A. Burg, M. Zellweger, C. Studer, and W. Fichtner, "VLSI implementation of the list sphere algorithm," in *Proc. 24th Norchip Conference*, Nov. 2006, pp. 107–110.

[66] T. Buerner, R. Dohmen, A. Zottmann, M. Saeger, and A. J. van Wijngaarden, "On a high-speed Reed-Solomon CODEC architecture for 43 Gb/s optical transmission systems," in *Proc. 24th International Conference on Microelectronics*, vol. 2, May 16–19, 2004, pp. 743–746.

[67] N. Chen and Z. Yan, "Reduced-complexity Reed-Solomon decoders based on cyclotomic FFTs," *IEEE Signal Process. Lett.*, vol. 16, no. 4, pp. 279–282, 2009.

[68] *Recommended Elliptic Curves for Federal Government Use*, National Institute of Standards and Technology Std.

[69] *SEC 2: Recommended Elliptic Curve Domain Parameters*, Standard for Efficient Cryptography Group Std.

[70] E. Costa, S. V. Fedorenko, and P. V. Trifonov, "On computing the syndrome polynomial in Reed-Solomon decoder," *European Transactions on Telecommunications*, vol. 15, no. 4, pp. 337–342, 2004. [Online]. Available: http://dx.doi.org/10.1002/ett.982

[71] Y. Wang and X. Zhu, "A fast algorithm for the Fourier transform over finite fields and its VLSI implementation," *IEEE Journal on Selected Areas in Communications*, vol. 6, no. 3, pp. 572–577, Apr. 1988.

[72] D. G. Cantor and E. Kaltofen, "On fast multiplication of polynomials over arbitrary algebras," *Acta Informatica*, vol. 28, pp. 693–701, 1991.

[73] S. Gao, "Clemson university mathematical sciences 985 course notes," Fall 2001.

[74] T. Mateer, "Fast Fourier transform algorithms with applications," Ph.D. dissertation, Clemson University, 2008.

[75] H. Fan and M. A. Hasan, "A new approach to subquadratic space complexity parallel multipliers for extended binary fields," *IEEE Trans. Comput.*, vol. 56, no. 2, pp. 224–233, 2007.

[76] S. Winograd, *Arithmetic Complexity of Computations*. SIAM, 1980.

[77] R. Lidl and H. Niederreiter, *Finite Fields (Encyclopedia of Mathematics and its Applications)*. Cambridge University Press, October 1996.

[78] J. C. Allwright, "Real factorisation of noncyclic-convolution operators with application to fast convolution," *Electronics Letters*, vol. 7, no. 24, pp. 718–719, 1971.

[79] D. A. Pitassi, "Fast convolution using the Walsh transforms," *IEEE Trans. Electromagn. Compat.*, no. 3, pp. 130–133, 1971.

[80] R. Agarwal and C. Burrus, "Fast one-dimensional digital convolution by multidimensional techniques," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 22, no. 1, pp. 1–10, 1974.

[81] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Reading, MA: Addison-Wesley, 1974.

[82] J. von zur Gathen and J. Gerhard, "Arithmetic and factorization of polynomials over F_2," in *PROC. ISSAC 96*. ACM PRESS, 1996.

[83] S. Gao and T. Mateer, "Additive fast Fourier transforms over finite fields," *IEEE Trans. Inf. Theory*, vol. 56, no. 12, pp. 6265–6272, 2010.

[84] D. G. Cantor, "On arithmetical algorithms over finite fields," *Journal of Combinatorial Theory, Series A*, vol. 50, no. 2, pp. 285 – 300, 1989. [Online]. Available: http://www.sciencedirect.com/science/article/B6WHS-4D7CXSX-6N/2/629d9278eee2594f48a6a884ba7c52c9

[85] X. Wu, Y. Dai, and Z. Yan, "List based soft-decision MIMO detection by the MCTS algorithm," in *Proc. IEEE International Symposium on Circuits and Systems*, 2010, pp. 3537–3540.

[86] P. Mihăilescu, "Primary cyclotomic units and a proof of Catalan's conjecture," *Journal für die reine und angewandte Mathematik*, vol. 2004, pp. 167 – 195, 2004.

[87] R. E. Blahut, *Theory and Practice of Error Control Codes.* Addison-Wesley, 1984.

[88] S. B. Wicker, *Error Control Systems for Digital Communications and Storage.* Upper Saddle River, NJ: Prentice Hall, 1995.

[89] T. K. Truong, P. D. Chen, L. J. Wang, Y. Chang, and I. S. Reed, "Fast, prime factor, discrete Fourier transform algorithms over $GF(2^m)$ for $8 \leq m \leq 10$," *Inf. Sci.*, vol. 176, no. 1, pp. 1–26, 2006.

[90] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences).* W. H. Freeman, January 1979.

[91] P. Trifonov, "Matrix-vector multiplication via erasure decoding," in *Proceedings of XI International Symposium on Problems of Redundancy in Information and Control Systems*, Jul. 2007.

[92] O. Gustafsson and M. Olofsson, "Complexity reduction of constant matrix computations over the binary field," in *WAIFI '07: Proceedings of the 1st*

international workshop on Arithmetic of Finite Fields. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 103–115.

[93] X. Wu, Z. Yan, and Y. Xie, "Two-dimensional crosstalk avoidance codes," in *Proc. IEEE Workshop Signal Processing Systems*, 2008, pp. 106–111.

[94] X. Wu and Z. Yan, "CAC CODEC designs based on numeral systems," in *Proc. IEEE Workshop Signal Processing Systems*, 2009, pp. 191–196.

[95] ——, "Efficient CODEC designs for crosstalk avoidance codes based on numeral systems," *IEEE Trans. VLSI Syst.*, vol. 19, no. 4, pp. 548–558, 2011.

[96] X. Wu, Y. Dai, Z. Yan, and Y. Wang, "Improving the reliability of the K-Best algorithm for MIMO detection with ordering," in *Proc. 19th Annual Wireless and Optical Communications Conf.*, 2010, pp. 1–5.

[97] X. Wu, Y. Dai, Y. Wang, and Z. Yan, "Efficient ordering schemes for high-throughput MIMO detectors," *Journal of Signal Processing Systems*, vol. 64, no. 1, pp. 61–74, 2011.

[98] X. Wu, Z. Yan, N. Chen, and M. Wagh, "Prime factor cyclotomic Fourier transforms with reduced complexity over finite fields," in *Proc. IEEE Workshop Signal Processing Systems*, 2010, pp. 450–455.

[99] X. Wu, M. Wagh, N. Chen, Y. Wang, and Z. Yan, "Composite cyclotomic Fourier transforms with reduced complexities," *IEEE Trans. Signal Process.*, vol. 59, no. 5, pp. 2136–2145, 2011.

[100] X. Wu and Z. Yan, “Computational complexity of cyclotomic fast Fourier transforms over characteristic-2 fields,” in *Proc. IEEE Workshop Signal Processing Systems*, 2011, accepted.

[101] X. Wu, Y. Wang, and Y. Zhiyuan, “On algorithms and complexities of cyclotomic fast Fourier transforms over arbitrary finite fields,” submitted to IEEE Tranaction on Signal Processing, 2011.

[102] F. Shi, X. Wu, and Z. Yan, “Crosstalk avoidance codes design through a novel pattern classification,” in *Proc. IEEE Workshop Signal Processing Systems*, 2011, accepted.

[103] M. Blaum and K. Lakovic, “Variable span Fibonacci modulation codes with limited byte error propagation,” in *Proc. IEEE Int. Symp. Information Theory*, 2007, pp. 2541–2544.

Appendix A

11-point Cyclic Convolution algorithms over GF(2^m)

The cyclic convolution of two 11-dimensional vectors $\mathbf{x}$ and $\mathbf{y}$ can be computed as $\mathbf{Q}^{(11)}(\mathbf{P}^{(11)}\mathbf{x} \cdot \mathbf{R}^{(11)}\mathbf{y})$, where $\cdot$ denotes an entry-wise multiplication between two vectors, and the matrices $\mathbf{Q}^{(11)}$, $\mathbf{P}^{(11)}$, and $\mathbf{R}^{(11)}$ are give by

$$\mathbf{Q}^{(11)} = \begin{bmatrix} 1000000000000001111100000000011111000000000 \\ 1000010000101110000100001011100000000000000 \\ 1000100010010110001000100101100000000000000 \\ 1001000101011000010001010110000000000000000 \\ 1010001001100010100010011000100000000000000 \\ 1100001110000011000011100000100000000000000 \\ 1000010000101110000000000000000001000010111 \\ 1000100010010110000000000000000010001001011 \\ 1001000101011000000000000000000100010101100 \\ 1010001001100010000000000000001000100110001 \\ 1100001110000010000000000000001000011100000 1 \end{bmatrix},$$

$$(\mathbf{P}^{(11)})^T = \begin{bmatrix} 11000001011010111111000011000011110000011000 \\ 10000011111111011111101010011111111101010011 \\ 10000101100110111111000110000011111001100000 \\ 10001100010110111101010000000111111010000010 \\ 10011111010110111011100000001011111100000100 \\ 10111101010110110111000000100111111000011000 \\ 11111101010110101111000011000111111101010011 \\ 11111101010110111111110101001111110001100000 \\ 11111001010110111111001100000111010100000000 \\ 11110001010111111111010000010110111000000010 \\ 11100001010100111111100000100101110000000100 \end{bmatrix},$$

and

$$(\mathbf{R}^{(11)})^T = \begin{bmatrix} 11000011100000100000000000000010000111000001 \\ 10100010011000100000000000000001000100110001 \\ 10010001010110000000000000000000100010101100 \\ 10001000100101100000000000000000010001001011 \\ 10000100000101110000000000000000001000010111 \\ 11000011100000110000111000000100000000000000 \\ 10100010011000101000100110001000000000000000 \\ 10010001010110000100010101100000000000000000 \\ 10001000100101100010001001011000000000000000 \\ 10000100000101110000100000101110000000000000 \\ 10000000000000011111000000000011111000000000 \end{bmatrix}.$$

Vita

Xuebin Wu received his B.E. and M.E. degrees in electrical engineering from Tsinghua University, Beijing, China in 2004 and 2007, respectively. He is currently pursuing the Ph.D degree in electrical engineering at Lehigh University, Bethlehem, Pennsylvania.

His research interest lies in channel coding technologies and VLSI architecture and circuit design for digital signal processing and communication systems.