2015

SIGNAL PROCESSING TECHNIQUES AND APPLICATIONS

Feng Shi
Lehigh University

SIGNAL PROCESSING TECHNIQUES AND APPLICATIONS

by

Feng Shi

Presented to the Graduate and Research Committee

of Lehigh University

in Candidacy for the Degree of

Doctor of Philosophy

in

Electrical Engineering

Lehigh University

January 2015

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. Tiffany Jing Li

Dr. Viswanath Annampedu
Avago Technologies

Acknowledgments

Foremost, I would like to express my sincere gratitude to my advisor Prof. Zhiyuan Yan for the continuous support of my Ph.D study and research, for his patience and expertise. There were numerous times when I was lost in details and made no progress. He can always guide me to work out the details with his expertise and valuable advices. More importantly, he showed me how to look the work in a higher level and more intriguing perspective. I am obliged to him for his endless support and encouragement. He is also my mentor and gave me a lot of valuable advices in seeking career opportunities. He recommended me for an internship, from which I gained invaluable industry experience. It is really lucky for me to pursue my Ph.D degree under his guidance. I could not imagine having a better advisor and mentor for my Ph.D study.

Besides my advisor, I would like to thank my co-advisor Prof. Wagh for his guidance in my later Ph.D research. His valuable expertise and insightful comments were very beneficial and inspired me in the new area. I would like also to thank Prof. Tiffany Jing Li and Dr. Viswanath Annampedu for serving on my Ph.D committee and spending their precious time for examming my work. I am grateful for their encouragement, insightful comments and inspiring questions.

Many sincere thanks to my labmates and friends in Lehigh University, Ning Chen, Xuebin Wu, Hongmei Xie, Chenrong Xiong, and Jun Lin. We shared a lot of memorable time and exciting discussions. In particular, I would like to thank Xuebin Wu for helping me and my wife in finding an apartment and shopping grocery every week in my first year. I am also grateful for many discussions and comments we had on our collaborated work. Many thanks also go to my friends at Lehigh: Chen Chen, Xingjian Zhang, Yang Liu, et al.

Finally, I would like to thank my parents for their endless support and unconditional love. They provided everything financially and spiritually for me to get better educations. I am proud of them and owe them this Ph.D degree. I am indebted to my wife for everything. She sacrificed her best years accompanying me at Lehigh. Without her love, I could not imagine how far I have gone.

Contents

List of Tables

List of Figures

Abstract

As the technologies scaling down, more transistors can be fabricated into the same area, which enables the integration of many components into the same substrate, referred to as system-on-chip (SoC). The components on SoC are connected by on-chip global interconnects. It has been shown in the recent International Technology Roadmap of Semiconductors (ITRS) that when scaling down, gate delay decreases, but global interconnect delay increases due to crosstalk. The interconnect delay has become a bottleneck of the overall system performance. Many techniques have been proposed to address crosstalk, such as shielding, buffer insertion, and crosstalk avoidance codes (CACs). The CAC is a promising technique due to its good crosstalk reduction, less power consumption and lower area. In this dissertation, I will present analytical delay models for on-chip interconnects with improved accuracy. This enables us to have a more accurate control of delays for transition patterns and lead to a more efficient CAC, whose worst-case delay is 30-40% smaller than the best of previously proposed CACs. As the clock frequency approaches multi-gigahertz, the parasitic inductance of on-chip interconnects has become significant and its detrimental effects, including increased delay, voltage overshoots and undershoots, and increased crosstalk noise, cannot be ignored. We introduce new CACs to address

both capacitive and inductive couplings simultaneously.

Quantum computers are more powerful in solving some NP problems than the classical computers. However, quantum computers suffer greatly from unwanted interactions with environment. Quantum error correction codes (QECCs) are needed to protect quantum information against noise and decoherence. Given their good error-correcting performance, it is desirable to adapt existing iterative decoding algorithms of LDPC codes to obtain LDPC-based QECCs. Several QECCs based on nonbinary LDPC codes have been proposed with a much better error-correcting performance than existing quantum codes over a qubit channel. In this dissertation, I will present stabilizer codes based on nonbinary QC-LDPC codes for qubit channels. The results will confirm the observation that QECCs based on nonbinary LDPC codes appear to achieve better performance than QECCs based on binary LDPC codes.

As the technologies scaling down further to nanoscale, CMOS devices suffer greatly from the quantum mechanical effects. Some emerging nano devices, such as resonant tunneling diodes (RTDs), quantum cellular automata (QCA), and single electron transistors (SETs), have no such issues and are promising candidates to replace the traditional CMOS devices. Threshold gate, which can implement complex Boolean functions within a single gate, can be easily realized with these devices. Several applications dealing with real-valued signals have already been realized using nanotechnology based threshold gates. Unfortunately, the applications using finite fields, such as error correcting coding and cryptography, have not been realized using nanotechnology. The main obstacle is that they require a great number of exclusive-ORs (XORs), which cannot be realized in a single threshold gate.

Besides, the fan-in of a threshold gate in RTD nanotechnology needs to be bounded for both reliability and performance purpose. In this dissertation, I will present a majority-class threshold architecture of XORs with bounded fan-in, and compare it with a Boolean-class architecture. I will show an application of the proposed XORs for the finite field multiplications. The analysis results will show that the majority class outperforms the Boolean class architectures in terms of hardware complexity and latency. I will also introduce a sort-and-search algorithm, which can be used for implementations of any symmetric functions. Since XOR is a special symmetric function, it can be implemented via the sort-and-search algorithm. To leverage the power of multi-input threshold functions, I generalize the previously proposed sort-and-search algorithm from a fan-in of two to arbitrary fan-ins, and propose an architecture of multi-input XORs with bounded fan-ins.

Chapter 1

Introduction

In many communication systems, such as on-chip interconnects and quantum systems, interferences from the system and environment often aggravate the performance and lead to functional issues. Techniques, such as crosstalk avoidance coding and error correction coding, have been proposed to address these issues. In nanotechnologies, conventional implementations of Boolean operations are quite different from CMOS technology and new techniques are needed for efficient implementations. In this proposal, we investigate and propose such signal processing techniques to address issues in these systems.

In the following, we first give a brief introduction and show the motivations of our work. Then, we present the main results of our work.

1.1 Motivations

1.1.1 Delay modeling for On-Chip Interconnects

With the process technologies scaling down into deep submicrometer, coupling capacitance between adjacent wires becomes more significant and increases the crosstalk delays greatly. Recent International Technology Roadmap of Semiconductors (ITRS) [1] shows that gate delay **decreases** with scaling while global wire delay **increases**. The crosstalk delay becomes a major part of the total delay, and greatly affects the overall system performance.

To evaluate and alleviate crosstalk delays, various delay models of interconnects have been proposed recently (see, for example, [2–7]), most of which are based on numerical approaches and offer little insight (see, e.g., [2–5]). Although these numerical models can have high accuracy, they have several drawbacks, such as bulky lookup tables, dependence on technology, poor portability, and high complexity.

In contrast, analytical delay models (see, e.g., [6, 7]) depend on few technology parameters and have very low computational complexities. The model in [6] has much higher accuracy. However, it is not conductive to alleviate the crosstalk. One widely used analytical delay model, proposed by Sotiriadis *et al.* [7], illustrates the connection between delays of coupled interconnects and transition patterns and appears to be the most comparable previous delay model. However, the model in [7] has limited accuracy. To improve accuracy, we focus on closed-form expressions of the signals on the bus based on a distributed RC model, and approximate the wire delays by evaluating these closed-form expressions.

1.1.2 Crosstalk Avoidance Codes

The analytical model proposed by Sotiriadis *et al.* [7], a widely used delay model, gives upper bounds on the delay of all wires on a bus. In this model, the delay of the k-th wire depends on the transition patterns of at most three wires, $k-1$, k, and $k+1$ only. From [7], the delay of the k-th wire ($k \in \{1, 2, \cdots, m\}$) of an m-bit bus is given by

$$T_k = \begin{cases} \tau_0[(1+\lambda)\Delta_1^2 - \lambda\Delta_1\Delta_2], & k = 1 \\ \tau_0[(1+2\lambda)\Delta_k^2 - \lambda\Delta_k(\Delta_{k-1} + \Delta_{k+1})], & k \neq 1, m \\ \tau_0[(1+\lambda)\Delta_m^2 - \lambda\Delta_m\Delta_{m-1}], & k = m, \end{cases} \tag{1.1}$$

According to this model, there are five classes of transition patterns, denoted by iC for $i = 0, 1, 2, 3, 4$, each of which has a delay $(1+i\lambda)\tau_0$. This classification enables one to limit the worst-case delay over a bus by restricting the patterns transmitted on the bus. That is, by avoiding all transition patterns in iC for $i > i_0$, one can achieve a worst-case delay of $(1+i_0\lambda)\tau_0$ over the bus. Based on this basic principle, crosstalk avoidance codes (CACs) of different worst-case delays have been proposed recently (see, for example, [8–10]). For example, forbidden overlap codes (FOCs), forbidden transition codes (FTCs), forbidden pattern codes (FPCs), and one lambda codes (OLCs) achieve a worst-case delay of $(1+3\lambda)\tau_0$, $(1+2\lambda)\tau_0$, $(1+2\lambda)\tau_0$, and $(1+\lambda)\tau_0$, respectively. In theory, a worst-case delay of τ_0 can be achieved by assigning two protection wires to each data wire [9].

The classification of transition patterns based on the model in [7] has two drawbacks. First, the model in [7] has limited accuracy because of its dependence on only three wires. That is, the model overestimates the delays of patterns in $1C$

through $4C$, while it underestimates the delays of patterns in $0C$. For this reason, the scheme with a worst-case delay of τ_0 is invalid since its actual delay is much greater. Second, the actual delay ranges in some classes overlap with others in their adjacent classes.

This, plus the overestimation of delays for $1C$ through $4C$, implies that the delays of existing CACs are not tightly controlled. These drawbacks motivate us to include more wires and to classify the transition patterns without overlapping delay ranges.

1.1.3 Quantum Error Correction

Quantum computers are more efficient than classical computers for some computational problems, such as factoring a large number and searching an unknown space for an element satisfying a known property [11]. However, quantum information, represented by quantum bits or qubits, suffers greatly from unwanted interactions with the outside world. Thus, quantum error correction codes (QECCs) are needed to protect quantum information against noise and decoherence [11].

Many QECCs have been proposed in the literature by importing classical error correction codes, such as low-density parity-check (LDPC) codes, convolutional codes, Turbo codes, and polar codes (see, for example, [12–21]). Among them, QECCs based on LDPC codes (see, for example, [12,13,16,17]) are important, since they can be decoded by adapting existing iterative decoding algorithms. As classical LDPC codes have asymptotically good performance for a wide class of noisy channels when decoded by the belief propagation algorithm [22], well-designed quantum LDPC codes also show good performance [16,17,23]. While most quantum LDPC

codes are based on binary LDPC codes, recently several QECCs based on nonbinary LDPC codes have been proposed in [23] with a much better error-correcting performance than existing quantum codes over a qubit channel.

Since stabilizer codes based on nonbinary LDPC codes have not been studied, motivated by the success of adopting nonbinary QC-LDPC codes in CSS codes in [23], we investigate stabilizer codes based on nonbinary QC-LDPC codes for qubit channels.

1.1.4 Efficient Threshold Architecture

According to the International Technology Roadmap of Semiconductors (ITRS) [1], the conventional CMOS technology has great challenge in further scaling. Although new materials and device structures can keep the CMOS scaling for the next several years, the CMOS scaling would reach the fundamental limits eventually. Some emerging nanotechnology, such as resonant tunneling diodes (RTDs), quantum cellular automata (QCA), and single electron transistors (SETs), have nanoscale structure and are promising candidates to replace the CMOS technology [24]. These new nanotechnology devices promise to have smaller feature size, higher speed and lower power consumption. Even at system level design they present two advantages. Firstly they easily realize threshold gates (see Fig. 7.11). Threshold gates are often more powerful than Boolean gates, and can implement complex Boolean functions with a single gate [25]. Thus the hardware complexity of larger systems implemented using nanotechnology tends to be a lot smaller. Secondly, the outputs of the threshold gates built with nanotechnology are self-latched. This provides a natural way of pipelining these systems in most signal processing applications.

Several applications dealing with real-valued signals have already been realized using nanotechnology based threshold gates [26–29]. However, there is an equally important class of signal processing applications using finite fields, such as error correcting coding and cryptography [30]. Unfortunately, the applications using finite fields have not been realized using nanotechnology.

The main obstacle for the nanotechnology based implementations of applications of finite fields of characteristic two, denoted as GF(2^m), is that they require exclusive-ORs (XORs) to realize all arithmetic operations over GF(2^m). Unlike most conventional Boolean gates such as AND, OR, NOT, NAND, and NOR, XOR cannot be realized as a single threshold gate. Thus the translation of a finite field architecture to nanotechnology merely by replacing a conventional gate with an appropriate combination of threshold gates becomes overly complex. Efficient implementations based on threshold logic are desired.

1.1.5 Multiway Sorting

Merging-based sorting networks are an important family of sorting networks. One popular 2-way merging algorithm called odd-even merging [31] merges two sorted lists (odd and even lists) into one sorted list. Most merge sorting networks are based on 2-way or multi-way merging algorithms using 2-sorters as basic building blocks. An alternative is to use n-sorters, instead of 2-sorters, as the basic building blocks so as to greatly reduce the number of sorters as well as the latency. This is also motivated by efficient threshold implementations of n-sorters due to the powerful computing capability of threshold logic.

1.2 Main results

Motivated by the above ideas, we have proposed several techniques and methods to address these issues. The main results are given by

1. **Delay modeling:** In chapter 2, we propose analytical delay models for coupled interconnects with improved accuracy. Based on a distributed RC model, we first derive closed-form expressions of the signals on the bus. Then we approximate the wire delays by evaluating these closed-form expressions. Our delay models differ from the model in [7] in two aspects. First, we use direct evaluations other than the Elmore delay in the model in [7] to approximate the delays. Second, we consider either three wires or five wires in our delay models for improved accuracy. Thus, our models achieves improved accuracy than the model in [7]. Since our delay models use the same classification as the model in [7], they also maintain the simplicity of the model in [7]. Hence, it is easy and conducive to use our delay models for the CAC designs. Also, our five-wire model can be applied to buses of any number of wires. Our extensive simulation results show that our delay models have improved accuracy than the model in [7].

2. **Crosstalk avoidance codes:** In chapter 3, we propose new CACs for RC-coupled on-chip interconnects. First, we partition all transition patterns with respect to the delays on the middle wire of a 5-wire bus. By grouping these patterns according to their evaluated delays, we have a finer classification of patterns without overlapping delays between adjacent classes. This enables us to have a more accurate control of delays for transition patterns, and CACs

designed based on our classification will be more effective. Then, we provide a method to design CACs with our classification. To illustrate this method, we present a new CAC based on our classification, which achieves a worst-case delay that is 30%–40% smaller than that of OLCs.

In chapter 4, we propose novel CACs accounting for both the capacitive and inductive couplings. The capacitive crosstalk is reduced by restricting opposite transitions in adjacent wires. Since the inductive coupling is a long-range effect, more neighboring wires are considered for inductive crosstalk. The reduction of inductive coupling is achieved by restricting same transitions in neighboring wires. We also propose CODEC design for our codes based on binary mixed-radix numeral systems. The complexity and delay of our CODECs are quadratically increasing with the size of the bus.

3. **Quantum error correction:** In chapter 5, we propose quasi-cyclic (QC) LDPC stabilizer codes over a qubit depolarizing channel. The construction of our QC-LDPC stabilizer codes is reduced to the construction of nonbinary QC-LDPC codes over GF(2^m) satisfying the zero SIP condition, and the decoding of our QC-LDPC stabilizer codes is based on that of the nonbinary QC-LDPC codes. First, we derive conditions for nonbinary QC-LDPC codes over GF(2^m) in order to satisfy the zero SIP condition and to eliminate the cycles of girth four, which usually lead to poor decoding performance by iterative decoding algorithms for LDPC codes. We have constructed two QC-LDPC stabilizer codes, and simulation results show that they outperform their counterparts in [16, 17]. This seems to confirm the observation [23] that QECCs based on nonbinary LDPC codes appear to achieve better performance than QECCs

based on binary LDPC codes.

4. **Threshold architecture:** In chapter 6, we propose efficient threshold architectures for exclusive-ORs with bounded fan-in. The main results are two classes of threshold architectures with bounded fan-ins of an n-input XOR. The first, called the *Boolean class*, expresses the XOR in a two-stage NAND circuit implemented through threshold gates. The second, referred to as the *majority class*, also has a two-level implementation and uses only generalized majority gates in the first level. Since one can implement an n-input XOR as a tree of two-input XORs, each of which can be expressed based on other Boolean gates and implemented by their threshold gates, we refer to this approach as *direct conversion* and use it as a basis for comparison. It turns out the architectures obtained by direct conversion are the same as Boolean class architectures with $B = 3$. Hence, our Boolean class architectures provide a variety of tradeoffs between hardware and time complexities beyond the direct conversion architectures. Our analysis results also show that the majority class performs better than the Boolean class as well as the architectures by direct conversion in both the hardware and time complexity, because the majority class takes better advantage of the more powerful nature of threshold gates.

5. **Multiway Sorting Network:** In chapter 7, we propose a new multiway merging algorithm with n-sorters as basic blocks. This merging algorithm merges n sorted lists of m values each in $1 + \lceil m/2 \rceil$ steps, where $n \leq m$. A sorting algorithm based on the proposed merging algorithm is also introduced. Our sorting networks of N inputs have an order $O(N \log^2 N)$ of basic sorters,

which is asymptotically the same with previously proposed multiway sorting algorithms. In the wide range of N, our algorithm performs better than other sorting algorithms. For $N \leq 1.46 \times 10^4$, our algorithm has up to 46% fewer sorters. For a more accurate comparison, we show a binary sorting network in threshold logic, where the basic sorter size scales linearly with the number of inputs, and compare the number of gates for sorting N inputs. For $N \leq 2 \times 10^4$, there are up to 39% fewer gates for a binary sorting network in threshold logic.

Chapter 2

Delay Models for On-Chip Interconnects

2.1 Introduction

As the process technologies scale into deep submicron region, crosstalk delay is becoming increasingly severe, especially for global on-chip buses. To cope with this problem, accurate delay models of coupled interconnects are needed. In particular, delay models based on analytical approaches are desirable, because they not only are largely transparent to technology, but also explicitly establish the connections between delays of coupled interconnects and transition patterns, thereby enabling crosstalk alleviating techniques such as crosstalk avoidance codes (CACs). Unfortunately, existing analytical delay models, such as the widely cited model [7], have limited accuracy and do not account for possibly asynchronous switching instants of wires.

The delay of the k-th wire ($k \in \{1, 2, \cdots, m\}$) of an m-bit bus is rewritten in the following [7], ,

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

where λ is the ratio of the coupling capacitance between adjacent wires and the loading capacitance, τ_0 is the intrinsic delay of a transition on a single wire, and Δ_k is 1 for $0 \rightarrow 1$ transition, -1 for $1 \rightarrow 0$ transition, or 0 for no transition on the k-th wire. We observe that in this model, the delay of the k-th wire depends on the transition patterns of wires $k-1$, k, and $k+1$ only. As shown in Eq. (3.1), all possible values of T_k are given by $(1+i\lambda)\tau_0$ for $i \in \{0, 1, 2, 3, 4\}$. Thus, all transition patterns on wires $k-1$, k, and $k+1$ can be divided into five classes iC for $i \in \{0, 1, 2, 3, 4\}$ according to their corresponding i (this classification was also used in [8]). By limiting transition patterns over the bus, the worst delay can be reduced. Various crosstalk avoidance codes (CACs) (see, for example, [8, 10, 32, 33]) have been proposed based on this model.

Unfortunately, the model in [7] has limited accuracy for the following reasons. To achieve simplicity, only three wires are considered in the derivation of the model. In a bus with more than three wires, the simulated wire delay for $0C$ transition patterns is much larger than τ_0, the delay of $0C$ given by (3.1). For example, the scheme to achieve a delay of τ_0 in [9] would be ineffective. Furthermore, the Elmore delay, which tends to overestimate the delay [34], is used in the derivation. This is also verified by our simulation results.

In the following, we propose analytical delay models for coupled interconnects with improved accuracy. Based on a distributed RC model, we first derive closed-form expressions of the signals on the bus. Then we approximate the wire delays by evaluating these closed-form expressions. Our delay models differ from the model in [7] in two aspects. First, we use direct evaluations other than the Elmore delay in the model in [7] to approximate the delays. Second, we consider either three wires or five wires in our delay models for improved accuracy. Thus, our models achieves improved accuracy than the model in [7]. Since our delay models use the same classification as the model in [7], they also maintain the simplicity of the model in [7]. Hence, it is easy and conducive to use our delay models for the CAC designs. Also, our five-wire model can be applied to buses of any number of wires. Our extensive simulation results show that our delay models have improved accuracy than the model in [7].

2.2 DELAY MODEL

2.2.1 System model

The on-chip buses are often approximated by the distributed RC model [35]. Our delay models do not consider the effects of inductance for two reasons. First, it is difficult to derive a closed-form expression of the signals on the bus based on the RLC model. More importantly, according to the criteria in [36], the inductance effects are negligible for buses with length in some range. This conclusion was also confirmed by other works: the 16b, 32Gb/s, 5mm-long bus and 8b, 16Gb/s, 10mm-long bus in [37] show that the distributed RC model is still accurate to characterize

these high-speed long interconnects from 5mm to 10mm. So we use distributed RC model in our derivation for delay models.

In the following, our models do not account for the source resistance and load capacitance. However, they can be readily modified to account for both using the techniques in [38]. In general, source resistance and load capacitance tend to increase the delay. Since the crosstalk delay on the bus is the major part of the whole delay, the delays introduced by other parts are ignored. For this reason, no buffer is used. We assume that ideal step signals are applied on the bus directly.

According to [5], the closed-form expressions of the signals on the bus via a distributed RC model are sums of infinite terms. It was shown in [38] that sums of the two most significant terms provide a very close approximation of signals on the bus. This technique is crucial for the evaluation of the closed-form expressions.

The distributed RC model of an m-wire bus is shown in Fig. 2.1, where $V_i(x,t)$ denotes the transient signal at a position x along wire i for $i \in \{1, 2, \cdots, m\}$, r and c denote the resistance and capacitance per unit length, respectively. Also, λc denotes the coupling capacitance per unit length between two adjacent wires. In this work, we focus on a uniformly distributed bus and hence assume the parameters r, c, and λ are the same for all wires.

Our models are based on the 50% delay, which is defined as the time difference between the respective instants when the input signal and corresponding output signal cross 50% of the supply voltage V_{dd}. In the following, we focus on worst-case patterns leading to the largest 50% delay of the middle wire(s). For some transition patterns, the delay of the middle wire(s) is the greatest among all wires. For other transition patterns, other wires may have a greater delay, but the worst delays of

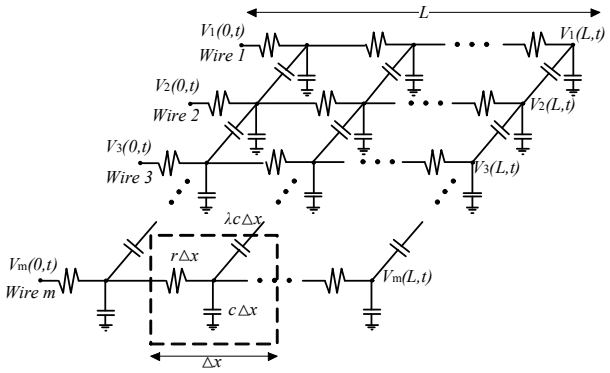


Figure 2.1: A distributed RC model of an m-wire bus.

all wires within the same class are close. Hence, our model can also be applied to other wires so as to approximate their delays with high accuracy. For simplicity, we assume m is odd, and hence wire $\frac{m+1}{2}$ is the middle wire. We use T_m^{iC} to denote the **worst** delay of the middle wire (wire $\frac{m+1}{2}$) of an m-wire bus for all iC patterns.

We first investigate the case $m = 3$ and then extend our results to the case $m = 5$. There are two reasons for studying the three-wire model. First and foremost, the three-wire model is the foundation of the derivation of our five-wire model. Second, our three-wire model shows higher accuracy than our five-wire model for buses with only three wires, which are used in partial coding schemes (see, e.g., [8, 10, 32]).

2.2.2 Three-wire model

Based on the same technique in [38], the differential equations characterizing a three-wire bus with length L are given by:

$$\frac{\partial^2}{\partial x^2}V(x,t) = RC\frac{\partial}{\partial t}V(x,t), \tag{2.2}$$

where $V(x,t) = [V_1(x,t)\ V_2(x,t)\ V_3(x,t)]^T$ and $V_i(x,t)$ denotes the voltage of wire i at distance x $(0 \leq x \leq L)$ at time t for $i = 1,2,3$, $R = \begin{bmatrix} r & 0 & 0 \\ 0 & r & 0 \\ 0 & 0 & r \end{bmatrix}$, and $C = \begin{bmatrix} c+\lambda c & -\lambda c & 0 \\ -\lambda c & c+2\lambda c & -\lambda c \\ 0 & -\lambda c & c+\lambda c \end{bmatrix}$.

The three eigenvalues of C are given by $p_1 = c$, $p_2 = (1+\lambda)c$, and $p_3 = (1+3\lambda)c$, and their respective eigenvectors e_i's are $[1\ 1\ 1]^T$, $[1\ 0\ -1]^T$, and $[-1\ 2\ -1]^T$. Hence, Eq. (3.2) is transformed to

$$\frac{\partial^2}{\partial x^2} U_i(x,t) = r p_i \frac{\partial}{\partial t} U_i(x,t) \text{ for } i = 1,2,3, \tag{2.3}$$

where $U_i(x,t) = V^T(x,t)e_i$ for $i = 1,2,3$. So $U_1(x,t) = V_1(x,t) + V_2(x,t) + V_3(x,t)$, $U_2(x,t) = V_1(x,t) - V_3(x,t)$, and $U_3(x,t) = 2V_2(x,t) - V_1(x,t) - V_3(x,t)$.

Applying Laplace transform on Eq. (2.3), we have

$$\frac{\partial^2}{\partial x^2} U_i(x,s) = r p_i [s U_i(x,s) - U_i(x,0)] \text{ for } i = 1,2,3. \tag{2.4}$$

Using appropriate initial conditions, we solve Eq. (2.4) for $U_i(x,t)$ and obtain $V_2(L,t) = \frac{1}{3}[U_1(L,t) + U_3(L,t)]$. By solving $V_2(L,t) = 0.5V_{dd}$, we can approximate the 50% delay of a three-wire bus for different transition patterns.

The expressions of wire 2 are given by $V_2(L,t) \doteq 1 - a_1 e^{-\frac{t}{\tau}} - a_2 e^{-\frac{t}{(1+3\lambda)\tau}}$, where a_i, $i = 1,2$ are constant coefficients, and $\tau = \frac{8}{\pi^2}\tau_0$. We use "↑" to denote a transition from 0 to the supply voltage V_{dd} (normalized to 1), "-" no transition, and "↓" a transition from V_{dd} to 0. We first identify the worst-case patterns in all classes through simulations, which are shown in Tab. 2.1. The expressions of wire 2 and the approximate delays of all classes are also shown in Tab. 2.1, respectively.

iC	Worst-case patterns	$V_2(L,t)$		Delay
		a_1	a_2	
$0C$	↑↑↑	$\frac{4}{\pi}$	0	$\left(\ln \frac{8}{\pi}\right) \tau$
$1C$	-↑↑	$\frac{8}{3\pi}$	$\frac{4}{3\pi}$	$\left(\ln \frac{16}{\pi}\right) \tau$
$2C$	-↑-	$\frac{4}{3\pi}$	$\frac{8}{3\pi}$	$\left(\ln \frac{16}{3\pi}\right) (1+3\lambda)\tau$
$3C$	-↑ ↓	0	$\frac{4}{\pi}$	$\left(\ln \frac{8}{\pi}\right) (1+3\lambda)\tau$
$4C$	↓↑↓	$-\frac{4}{3\pi}$	$\frac{16}{3\pi}$	$\left(\ln \frac{32}{3\pi}\right) (1+3\lambda)\tau$

Table 2.1: Analytical three-wire model ($V_2(L,t) \doteq 1 - a_1 e^{-\frac{t}{\tau}} - a_2 e^{-\frac{t}{(1+3\lambda)\tau}}$, $\tau_0 = \frac{rcL^2}{2}$, and $\tau = \frac{8}{\pi^2}\tau_0$).

iC	Worst-case patterns	Decomposition
$0C$	↓↑↑↑↓	(↓-↑-↓)+(-↑- - -)+(- - -↑-)
$1C$	↓-↑↑↓	(↓-↑-↓)+(- - -↑-)
$2C$	↓-↑-↓	(↓-↑-↓)
$3C$	↑-↑↓↑	(↑↑↑↑↑)+ 2(- - -↓-) + (-↓- - -)
$4C$	↑↓↑↓↑	(↑↑↑↑↑)+ 2(-↓- - -)+ 2(- - -↓-)

Table 2.2: Decomposition of worst-case patterns in the five-wire model.

2.2.3 Five-wire model

To further improve the accuracy of delay, we include two extra adjacent wires and consider the influences of all five wires to approximate the delays. There are three kinds of transition: ↑, -, and ↓ for each wire. Thus, for such a five-wire bus, there are 3^5 transition patterns. To maintain the simplicity of our models, we still categorize them into five classes ($iC,\ i \in \{0, 1, 2, 3, 4\}$) based on the transition patterns of middle three wires (wires 2, 3, and 4). Hence, there are nine different transition patterns for each pattern of the same class.

Since the bus is a linear system, any pattern could be decomposed into a combination of patterns with single transition. Then the expression of the middle wire equals the sums of expressions of all individual wires on the middle wire. However, this would lead to complicated expressions, which are not easy to solve. We propose to group these individual wires to form some special patterns, reducible transition patterns (RTPs) and single transition patterns (STPs), which are easy to analyze.

An RTP is defined as a transition pattern in the five-wire model which can be reduced to a transition pattern in the three-wire model. {↑↑↑↑↑, ↓↓↓↓↓, ↓-↑-↓, ↑-↓-↑} is the set of RTPs for the five-wire model. For the transition ↑↑↑↑↑ (similarly for ↓↓↓↓↓), the expression of wire 3 is approximated by $V_3(L,t) \doteq 1-\frac{4}{\pi}e^{-\frac{t}{\tau}}$. For the transition ↓-↑-↓ (similarly for ↑-↓-↑), it can be converted into a three-wire pattern ↓↑↓, where the coupling capacitor between wire 1 (or 5) and wire 3 is $\frac{\lambda}{2}$ per unit length. The expression of wire 3 is approximated by $V_3(L,t) \doteq 1+\frac{4}{3\pi}e^{-\frac{t}{\tau}}-\frac{16}{3\pi}e^{-\frac{t}{(1+\frac{3}{2}\lambda)\tau}}$, and the delay is approximated by $\ln(\frac{16}{3\pi})(1+\frac{3}{2}\lambda)\tau$.

An STP is defined to be a transition pattern with transitions on only one wire in the five-wire model. For our five-wire model, we focus on the set of STPs with

iC	Worst-case patterns	$V_3(L,t)$			Delay
		a_1	a_2	a_3	
$0C$	↓↑↑↑↓	0	$\frac{16}{3\pi}$	$-\frac{8}{3\pi}$	$0.165(1+3\lambda)\tau$
$1C$	↓-↑↑↓	0	$\frac{16}{3\pi}$	$-\frac{4}{3\pi}$	$0.310(1+3\lambda)\tau$
$2C$	↓-↑-↓	$-\frac{4}{3\pi}$	$\frac{16}{3\pi}$	0	$\left(\ln\frac{32}{3\pi}\right)(1+\frac{3}{2}\lambda)\tau$
$3C$	↑-↑↓↑	0	0	$\frac{4}{\pi}$	$\left(\ln\frac{8}{\pi}\right)(1+3\lambda)\tau$
$4C$	↑↓↑↓↑	$-\frac{4}{3\pi}$	0	$\frac{16}{3\pi}$	$\left(\ln\frac{32}{3\pi}\right)(1+3\lambda)\tau$

Table 2.3: Analytical five-wire model ($V_3(L,t) \doteq 1-a_1e^{-\frac{t}{\tau}}-a_2e^{-\frac{t}{(1+\frac{3}{2}\lambda)\tau}}-a_3e^{-\frac{t}{(1+3\lambda)\tau}}$, $\tau_0=\frac{rcL^2}{2}$, and $\tau=\frac{8}{\pi^2}\tau_0$).

iC	Worst-case patterns	Sim.	Our three-wire model			[7]		
		T_d (ps)	T_3^{iC}	(ps)	$\frac{\lvert T_3^{iC}-T_d\rvert}{T_d}$	T_2 in (3.1)	(ps)	$\frac{\lvert T_2-T_d\rvert}{T_d}$
$0C$	↑↑↑	2.87	$\left(\ln\frac{8}{\pi}\right)\tau$	2.84	1.05%	τ_0	3.75	30.66%
$1C$	↑↑-	4.99	$\left(\ln\frac{16}{\pi}\right)\tau$	4.94	1.00%	$(1+\lambda)\tau_0$	41.25	726.65%
$2C$	-↑-	49.70	$\left(\ln\frac{16}{3\pi}\right)(1+3\lambda)\tau$	49.87	0.34%	$(1+2\lambda)\tau_0$	78.75	58.45%
$3C$	↓↑-	88.61	$\left(\ln\frac{8}{\pi}\right)(1+3\lambda)\tau$	88.08	0.60%	$(1+3\lambda)\tau_0$	116.25	31.19%
$4C$	↓↑↓	115.89	$\left(\ln\frac{32}{3\pi}\right)(1+3\lambda)\tau$	115.18	0.61%	$(1+4\lambda)\tau_0$	153.75	32.67%

Table 2.4: Comparison of simulated delays, our three-wire model, and the model in [7] ($\tau_0=3.75$ps, $\tau=\frac{8}{\pi^2}\tau_0$, and $\lambda=10$)

transitions on wire 2 or 4, {-↑- - -, -↓- - -, - - -↑-, - - -↓-}.

The expressions of wire 3 can be approximated by our three-wire model. Let $V_j^i(x,t)$ denote the signal on wire j due to coupling from wire i. For example, by ignoring coupling from wires 4 and 5 in -↑- - -, the output of wire 3 is approximated by $V_3^2(L,t) \doteq -\frac{4}{3\pi}e^{-\frac{t}{\tau}}+\frac{4}{3\pi}e^{-\frac{t}{(1+3\lambda)\tau}}$, which is obtained by our three-wire model.

We propose the following approaches to derive the delay of the five-wire bus.

(1) We first **decompose** the worst-case pattern in each class into a combination of an RTP and STP(s).

(2) Then we combine the expressions of the RTP and STP(s) for the middle wire based on our three-wire model.

iC	Worst-case patterns	Sim.	Our five-wire model			[7]		
		T_d (ps)	T_5^{iC}	(ps)	$\frac{\lvert T_5^{iC}-T_d\rvert}{T_d}$	T_3 in (3.1)	(ps)	$\frac{\lvert T_3-T_d\rvert}{T_d}$
$0C$	↓↑↑↑↓	23.30	$0.165(1+3\lambda)\tau$	19.18	17.68%	τ_0	3.75	83.91%
$1C$	↓-↑↑↓	37.42	$0.310(1+3\lambda)\tau$	34.99	6.49%	$(1+\lambda)\tau_0$	41.25	10.24%
$2C$	↓-↑-↓	56.58	$\left(\ln\frac{32}{3\pi}\right)\left(1+\frac{3}{2}\lambda\right)\tau$	59.46	5.09%	$(1+2\lambda)\tau_0$	78.75	39.18%
$3C$	↑-↑↓↑	88.55	$\left(\ln\frac{8}{\pi}\right)(1+3\lambda)\tau$	88.08	0.53%	$(1+3\lambda)\tau_0$	116.25	31.28%
$4C$	↑↓↑↓↑	127.29	$\left(\ln\frac{32}{3\pi}\right)(1+3\lambda)\tau$	115.18	9.51%	$(1+4\lambda)\tau_0$	153.75	20.79%

Table 2.5: Comparison of simulated delays, our five-wire model, and the model in [7] ($\tau_0 = 3.75$ps, $\tau = \frac{8}{\pi^2}\tau_0$, and $\lambda = 10$).

(3) Finally, we evaluate the expression of the middle wire to approximate its delay.

Since the performance is limited by the worst delay in each class, we only need to approximate the delays of the worst-case patterns in all classes. For classes $0C$-$4C$, we use simulations to identify the worst-case patterns, which are given by ↓↑↑↑↓, ↓-↑↑↓, ↓-↑-↓, ↑-↑↓↑, and ↑↓↑↓↑, respectively (assuming the middle wire has an upward transition). With RTPs and STPs, we decompose the worst-case pattern in each class as shown in Tab. 2.2.

The expressions of wire 3 are given by $V_3(L,t) \doteq 1 - a_1 e^{-\frac{t}{\tau}} - a_2 e^{-\frac{t}{(1+\frac{3}{2}\lambda)\tau}} - a_3 e^{-\frac{t}{(1+3\lambda)\tau}}$, where a_i, $i = 1, 2, 3$ are constant coefficients. For all worst-case patterns in a five-wire bus, the expressions of wire 3 and the approximate delays are shown in Tab. 2.3, respectively.

2.3 PERFORMANCE EVALUATION

To evaluate the performance of our models and compare them with the model in [7], we consider following three scenarios. First, we focus on three-wire and five-wire buses, where our models are originally derived. This scenario can also be applied

to partial coding schemes (see, e.g., [8, 10, 32]), where a wide bus is divided into sub-buses with a few wires. Then we consider buses with more than five wires and run extensive simulations with an odd number of wires (up to 33 wires). For brevity, only simulation results for a 17-wire bus are presented. In the first two scenarios, we only focus on the worst delays of middle wires. In the third scenario, we assume the transition patterns are limited to three families of CACs and consider the worst delays for all wires of a 17-wire bus.

The simulation results are obtained by HSPICE. The coupling factor λ depends on the layer for routing the interconnect, the layer for the ground, the width for each wire, and the space between adjacent wires. We adopt a 0.1μm process and route the global interconnects in the top metal layer. The bulk capacitance is considered from top metal layer to the substrate, with $\lambda = 10$. For the 0.1μm process, the parasitic parameters are given by [39], and the parameter $\tau_0 = \frac{rcL^2}{2}$ for a 5mm long bus is approximately 3.75ps. Though this process is somewhat outdated, we have also tried other process technologies with different values for λ and τ_0 such as 45nm technology [40]. For all process technologies, our delay models can be easily adapted and show better accuracy than the model in [7].

2.3.1 Three-wire and five-wire buses

For a three-wire bus, we compare the simulated delays with the delays given by our model and the model in [7] for all classes in Tab. 3.2, where T_d denotes the simulated worst delay of wire 2, T_3^{iC} the approximate delay for iC pattern by our three-wire model, and T_2 by the model in [7]. The error percentages of our model and the model in [7] for each class are also included in Tab. 3.2. For all five classes

iC	Worst-case patterns with respect to our assumptions	Sim.	Our model		[7]	
		T_d	T_5^{iC}	$\frac{\lvert T_5^{iC}-T_d\rvert}{T_d}$	T_9	$\frac{\lvert T_9-T_d\rvert}{T_d}$
$0C$	↑↑↑↑↓↓↓ (↑↑↑) ↓↓↓↑↑↑↑	25.07	19.18	23.49%	3.75	85.04%
$1C$	↑↑↑↑↓↓ (↑↑ -) ↓↓↑↑↑↑↑	39.13	34.99	10.58%	41.25	5.42%
$2C$	↓↓↑↑↑↑↓ (- ↑ -) ↓↑↑↑↑↓↓	65.93	59.46	9.81%	78.75	19.44%
$3C$	↓↓↓↑↑↑↑ (↓↑ -) ↑↑↑↑↓↓↓	95.39	88.08	7.66%	116.25	21.87%
$4C$	↑↓↓↓↑↑↑ (↓↑↓) ↑↑↑↓↓↓↑	130.43	115.18	11.69%	153.75	17.88%

Table 2.6: Comparison of simulated delays and delays given by our five-wire model and the model in [7] for wire 9 in a 17-wire bus ($\tau_0 = 3.75$ps, $\tau = \frac{8}{\pi^2}\tau_0$, and $\lambda = 10$). All delays are in ps.

of transition patterns in a three-wire bus, the **maximum** and **minimum** errors by our model are 1.05% and 0.34%, respectively, as opposed to 726.65% and 30.66% by the model in [7], respectively. As shown in Tab. 3.2, our three-wire model is much more accurate than the model in [7] for all patterns in a three-wire bus. We remark that the delay of a $1C$ pattern by our model, $\left(\ln \frac{16}{\pi}\right)\tau$, does not depend on λ.

For a five-wire bus, we compare the simulated delays with the delays given by our five-wire model and the model in [7] for all classes in Tab. 3.6, where T_d denotes the simulated worst delay of wire 3 for all iC patterns, T_5^{iC} the approximate delay for iC pattern by our five-wire model, and T_3 by the model in [7]. For a five-wire bus, the **maximum** and **minimum** errors by our model are 17.68% and 0.53%, respectively, in comparison to 83.91% and 10.24% by the model in [7], respectively. As shown in Tab. 3.6, our five-wire model is more accurate than the model in [7] for all patterns in a five-wire bus. Particularly, we observe that the worst delay for the $0C$ patterns are much larger than that given by the model in [7]. In [9], the author proposed a scheme to achieve a delay of τ_0 by surrounding each data wire with two identical wires. According to our model, this scheme is ineffective, because the worst delay could be as large as $0.165(1 + 3\lambda)\tau_0$.

2.3.2 17-wire bus

We next compare our five-wire model with the model in [7] for a 17-wire bus. We still focus on the middle wire (wire 9) in the 17-wire bus and identify the worst-case patterns in all classes through simulations. The transition patterns are categorized by the transitions of the middle three wires (wires 8, 9, and 10). Since there are 3^{14} transition patterns in each class, it is infeasible to search all transitions to identify the pattern with the longest delay. For any two wires symmetric to wire 9 (wire i and wire 18-i, $i \in \{1, 2, \cdots, 8\}$), there are nine possible patterns, $\uparrow\uparrow$, $\downarrow\downarrow$, - -, $\uparrow$-, -$\uparrow$, $\downarrow$-, -$\downarrow$, $\uparrow\downarrow$, and $\downarrow\uparrow$. If the transitions on the two symmetric wires are in opposite direction, we assume the influences of the two transitions will cancel out as a result of symmetry. For other patterns, we assume $\uparrow\uparrow$ has greater influence than $\uparrow$- or -$\uparrow$. Similarly, $\downarrow\downarrow$ has greater influence than $\downarrow$- or -$\downarrow$. Based on the discussion above, we assume the longest delay happens when two symmetric wires have either $\uparrow\uparrow$ or $\downarrow\downarrow$ transitions. So there are only $2^7 = 128$ patterns left to check in each class.

To find the worst-case patterns, we search all possible symmetric transition patterns in each class. The worst-case patterns are listed in the second column of Tab. 2.6, where the pattern on wires 8, 9, and 10 is shown in the parenthesis. We compare the simulated worst delays with the delays given by our five-wire model and the model in [7] for all classes in Tab. 2.6, where T_d denotes the simulated worst delay of wire 9 for all iC patterns, T_5^{iC} the approximate delay for iC pattern by our five-wire model, and T_9 by the model in [7]. For all five classes, the **maximum** and **minimum** errors by our model are 23.49% and 7.66%, respectively, as opposed to 85.04% and 5.42% by the model in [7], respectively. For all classes except $1C$, our five-wire model outperforms the model in [7]. the model in [7] also shows a large

error percentage for $0C$. We have also tried other buses with odd number of wires up to 33. Based on the simulation results, we conjecture that our five-wire model would be more accurate than the model in [7] for buses with any number of wires.

	Delays		
	OLC	FPC	FOC
[7]	41.25	78.75	116.25
T_5^{iC}	34.99	59.46	88.08
wire i			
1	**33.96**	64.31	63.00
2	21.59	62.52	95.06
3	32.37	63.73	94.90
4	32.40	62.14	96.56
5	32.16	63.40	94.30
6	32.49	64.63	96.99
7	32.55	**65.00**	93.69
8	32.50	62.26	93.31
9	33.18	60.74	94.93
10	33.27	62.21	95.94
11	31.92	61.10	94.74
12	32.07	61.55	94.33
13	32.02	63.31	96.56
14	32.97	60.22	**97.24**
15	32.83	64.70	92.01
16	21.29	63.25	95.29
17	33.61	63.52	63.83

Table 2.7: Comparison of simulated delays and delays given by our five-wire model and [7] for all wire in a 17-wire bus ($\tau_0 = 3.75$ps, $\tau = \frac{8}{\pi^2}\tau_0$, and $\lambda = 10$). All delays are in ps.

2.3.3 Performance of CACs

In the simulation results above, we only focus on the middle wire of a 17-wire bus. Now we evaluate the delays on all wires of a 17-wire bus for three families of CACs [8, 10, 32]: *one Lambda codes* (OLCs), *forbidden pattern codes* (FPCs), and

forbidden overlap codes (FOCs). Based on our five-wire model, the worst delays of aforementioned CACs are given by $0.310(1+3\lambda)\tau$, $\left(\ln \frac{32}{3\pi}\right)\left(1+\frac{3}{2}\lambda\right)\tau$, and $\left(\ln \frac{8}{\pi}\right)(1+3\lambda)\tau$, respectively. Based on the model in [7], the worst delays of aforementioned CACs are given by $(1+\lambda)\tau_0$, $(1+2\lambda)\tau_0$, and $(1+3\lambda)\tau_0$, respectively. For each CAC, 1000 codewords from the code book are randomly chosen and transmitted over a 17-wire bus consecutively, thus forming 999 transitions. We compare the simulated worst delays of each wire and the delays given by our five-wire mode and the model in [7], respectively, in Tab. 2.7. For the OLC, the FPC, and the FOC in a 17-wire bus, the largest delays are emphasized in boldface. As shown in Tab. 2.7, our five-wire model is more accurate than the model in [7] for all three families of CACs. Also, in [41], a new CAC with a smaller worst delay than an OLC has been proposed based on our five-wire model and a new classification of transition patterns. It shows that our five-wire model enables the design of more effective CACs.

2.4 Summary

In this chapter, we propose improved analytical delay models for coupled interconnects, based on the distributed RC model. First the closed-form expressions of the signals on three-wire and five-wire buses are derived, which are also motivated by partial coding schemes. Then delays corresponding to different patterns are approximated by evaluating the closed-form expressions. The simulation results show that our models have better accuracy than that in [7]. Although our models are based on three-wire and five-wire buses, they can also be employed for a bus with more than five wires. Our simulation results also show that our five-wire model could still approximate delays better than the model in [7] for a general bus.

Chapter 3

Improved Crosstalk Avoidance Codes Based On A Novel Pattern Classification

3.1 Introduction

Recent International Technology Roadmap of Semiconductors (ITRS) [1] has shown a troubling trend: while gate delay **decreases** with scaling, global wire delay **increases**. This is because with the process technologies scaling down into deep submicrometer (DSM), the crosstalk delay becomes dominant in global wire delay due to the increasing coupling capacitance between adjacent wires. Hence, the crosstalk delay has become a serious bottleneck of the overall system performance.

The analytical model proposed by Sotiriadis *et al.* [7, 42], a widely used delay model, gives upper bounds on the delay of all wires on a bus. According to [7, 42],

the delay of the k-th wire ($k \in \{1, 2, \cdots, m\}$) of an m-bit bus is given by

$$T_k = \begin{cases} \tau_0[(1+\lambda)\Delta_1^2 - \lambda\Delta_1\Delta_2], & k = 1 \\ \tau_0[(1+2\lambda)\Delta_k^2 - \lambda\Delta_k(\Delta_{k-1} + \Delta_{k+1})], & k \neq 1, m \\ \tau_0[(1+\lambda)\Delta_m^2 - \lambda\Delta_m\Delta_{m-1}], & k = m, \end{cases} \tag{3.1}$$

where λ is the ratio of the coupling capacitance between adjacent wires and the ground capacitance, τ_0 is the propagation delay of a wire free of crosstalk, and Δ_k is 1 for $0 \rightarrow 1$ transition, -1 for $1 \rightarrow 0$ transition, or 0 for no transition on the k-th wire. In this model, the delay of the k-th wire depends on the transition patterns of at most three wires, $k-1$, k, and $k+1$ only. The transition patterns over these three wires can be classified based on Eq. (3.1) into five classes, denoted by Di for $i = 0, 1, 2, 3, 4$, and the patterns in Di have a worst-case delay $(1+i\lambda)\tau_0$. This classification enables one to limit the worst-case delay over a bus by restricting the patterns transmitted on the bus. That is, by avoiding all transition patterns in Di for $i > i_0$, one can achieve a worst-case delay of $(1+i_0\lambda)\tau_0$ over the bus. Based on this principle, crosstalk avoidance codes (CACs) of different worst-case delays have been proposed (see, for example, [8–10]). For example, forbidden overlap codes (FOCs), forbidden transition codes (FTCs), forbidden pattern codes (FPCs), and one lambda codes (OLCs) achieve a worst-case delay of $(1+3\lambda)\tau_0$, $(1+2\lambda)\tau_0$, $(1+2\lambda)\tau_0$, and $(1+\lambda)\tau_0$, respectively. Based on Eq. (3.1), a worst-case delay of τ_0 can be achieved by assigning two protection wires to each data wire [9]. Other types of CACs, such as those with equalization [43] or two-dimensional CACs [44], have been proposed in the literature. For CACs, since the area and power consumption of their encoder/decoder (CODECs) are all overheads, the complexities of the CODECs are

important to the effectiveness of CACs. Thus, efficient CODECs have been proposed for CACs [45–47].

The classification of transition patterns based on the model in [7, 42] has two drawbacks. First, the model in [7,42] has limited accuracy because of its dependence on only three wires: the model overestimates the delays of patterns in $D1$ through $D4$, while it underestimates the delays of patterns in $D0$. For this reason, the scheme with a worst-case delay of τ_0 in [9] is invalid since its actual delay is much greater. Second, the actual delay ranges in some classes overlap with others. This, plus the overestimation of delays for $D1$ through $D4$, implies that the delays of existing CACs are not tightly controlled. These drawbacks motivate us to include more wires and to classify the transition patterns without overlapping delay ranges.

In [48], we have proposed a new analytical five-wire delay model. Two extra neighboring wires are included in the delay model [48], and the delay of the middle wire of five neighboring wires is determined by the transition patterns on all five wires. This five-wire model has better accuracy than the model in [7, 42] for Di for $i = 0, 1, 2, 3, 4$ [48]. This work confirms that using more wires leads to improved accuracy.

There are two main contributions in this chapter:

- First, we approximate the crosstalk delay in a five-wire model and propose a new classification of transition patterns.
- Second, we propose a family of CACs based on our classification.

The work in this chapter is different from previous works, including our previous works, in several aspects:

- First, although the delay approximation in this chapter is also based on a five-wire model, it is different from that in our previous work [48]. The delay approximation in this chapter is carried out by extending the approach in [38] from a three-wire model to a five-wire one.

- Second, our classification of transition patters is different from that in [7, 42] (based on Eq. (3.1)), in two aspects. First, our classification has seven classes as opposed to five based on Eq. (3.1). Second, while the delays of some classes overlap for the classification based on Eq. (3.1), all classes in our classification have non-overlapping delays. These two key differences allow us to have a more accurate control of delays for transition patterns.

- Our new family of CACs is also different from previously proposed CACs, all of which are based on the classification in [7, 42] (based on Eq. (3.1)). While some codes in this new family are shown to be the same as existing CACs, OLCs, FPCs, and FOCs, this family also includes new codes that achieve smaller worst-case delays and improved throughputs than OLCs, which have the smallest worst-case delays among all existing CACs.

The rest of the chapter is organized as follows. In Section 3.2, we first propose our classification and compare it with that in [7, 42]. We then present our new family of CACs in Section 3.3 and compare their performance with existing CACs in Section 4.5. Some concluding remarks are provided in Section 7.6.

3.2 INTERCONNECT DELAYS AND CLASSIFICATION

3.2.1 Interconnect Modeling

Since the functionality and performance in DSM technology are greatly affected by the parasitics, distributed RC models are widely employed to analyze on-chip interconnects. In this chapter, we consider the distributed RC model of five wires shown in Fig. 4.1, where $V_i(x,t)$ denotes the transient signal at time t and position x $(0 \leq x \leq L)$ over wire i for $i \in \{1,2,3,4,5\}$, r and c denote the resistance and ground capacitance per unit length, respectively. Also, λc denotes the coupling capacitance per unit length between two adjacent wires. The value of λ depends on many factors, such as the metal layer in which we route the bus, the wire width, the spacing between adjacent wires, and the distance to the ground layer. We consider a uniformly distributed bus with the same parameters r, c, and λ for all the wires.

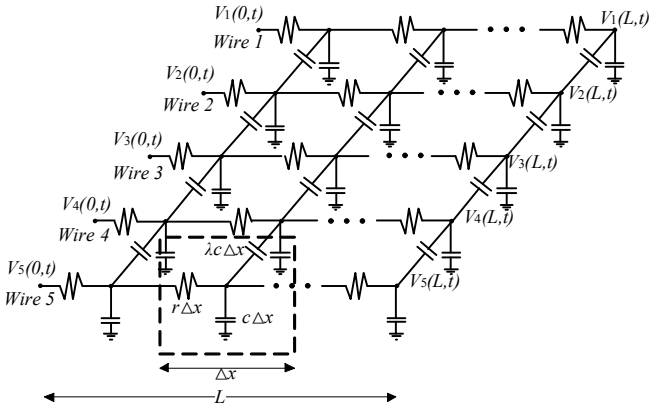


Figure 3.1: A distributed RC model for five wires.

3.2.2 Derivation of Closed-form Expressions

When determining the delay of a wire, the model in [7,42] considers only the effects of either **one** or **two** neighboring wires (cf. Eq. (3.1)). To address the drawbacks of the model in [7, 42] described above, additional neighboring wires need to be accounted for. In our delay derivation below, whenever possible we consider **four** neighboring wires of a wire, two neighboring wires on each side, to determine its delay. To approximate the delay of a side wire (wires 1, 2, $n-1$ or n) of an n-wire bus, **three** neighboring wires are considered. This is because the side wires are affected by fewer neighboring wires. This scheme is similar to the model in [7, 42] and appears to work well. We focus on the 50% delay, which is defined as the time required for the unit step response to reach 50% of its final value.

In [38], the crosstalk of two coupled lines was described by partial differential equations (PDEs), and a technique for decoupling these highly coupled PDEs was introduced by using eigenvalues and corresponding eigenvectors. In our work, we extend this approach from a three-wire model to a five-wire one. Specifically, we first use the technique in [38] to decouple the PDEs that describe the crosstalk of four coupled wires, then solve these independent PDEs for closed-form expressions, and finally approximate the delays of each wire.

The PDEs characterizing five wires with length L are given by:

$$\frac{\partial^2}{\partial x^2}\mathbf{V}(x,t) = \mathbf{RC}\frac{\partial}{\partial t}\mathbf{V}(x,t), \tag{3.2}$$

where $\mathbf{R} = \text{diag}\{r\ r\ r\ r\ r\}$, $\mathbf{V}(x,t) = [V_1(x,t)\ V_2(x,t)\ V_3(x,t)\ V_4(x,t)\ V_5(x,t)]^T$,

and

$$\mathbf{C} = c \begin{bmatrix} 1+\lambda & -\lambda & 0 & 0 & 0 \\ -\lambda & 1+2\lambda & -\lambda & 0 & 0 \\ 0 & -\lambda & 1+2\lambda & -\lambda & 0 \\ 0 & 0 & -\lambda & 1+2\lambda & -\lambda \\ 0 & 0 & 0 & -\lambda & 1+\lambda \end{bmatrix}.$$

The eigenvalues of $\mathbf{C}/c$ are given by $p_1 = 1$, $p_2 = 1 + \frac{5+\sqrt{5}}{2}\lambda$, $p_3 = 1 + \frac{5-\sqrt{5}}{2}\lambda$, $p_4 = 1 + \frac{3+\sqrt{5}}{2}\lambda$, and $p_5 = 1 + \frac{3-\sqrt{5}}{2}\lambda$. Their corresponding eigenvectors $\mathbf{e}_i$'s are given by $\mathbf{e}_1 = [1\ 1\ 1\ 1\ 1]^T$, $\mathbf{e}_2 = [\frac{\sqrt{5}-1}{4}\ -\frac{1+\sqrt{5}}{4}\ 1\ -\frac{1+\sqrt{5}}{4}\ \frac{\sqrt{5}-1}{4}]^T$, $\mathbf{e}_3 = [\frac{-\sqrt{5}+1}{4}\ \frac{\sqrt{5}-1}{4}\ 1\ \frac{\sqrt{5}-1}{4}\ -\frac{\sqrt{5}+1}{4}]^T$, $\mathbf{e}_4 = [-1\ \frac{\sqrt{5}+1}{2}\ 0\ -\frac{\sqrt{5}+1}{2}\ 1]^T$, and $\mathbf{e}_5 = [-1\ -\frac{\sqrt{5}-1}{2}\ 0\ \frac{\sqrt{5}-1}{2}\ 1]^T$, respectively.

With a technique for decoupling partial differential equations similar to [38], Eq. (3.2) is transformed into

$$\frac{\partial^2}{\partial x^2} U_i(x,t) = rcp_i \frac{\partial}{\partial t} U_i(x,t), \text{ for } i = 1,2,3,4,5,$$

where $U_i(x,t) = \mathbf{V}^T(x,t)\mathbf{e}_i$ denotes the transformed signals. This decoupled PDEs are independent of each other. Each $U_i(x,t)$ describes a single wire with a modified capacitance cp_i. The solution to $U_i(L,t)$ is given by a series of the form $U_i(L,t) = V_{dd} + \sum_{k=0}^{\infty} r_k e^{-\frac{t}{s_k \tau}}$. As shown in [38], a single-exponent approximation $V_{dd}(1 + r_0 e^{-\frac{t}{s_0 \tau}})$ is enough for $t/\tau > 0.1$, where r_0 and s_0 are the coefficients of the most significant term.

For different transitions, we solve Eq. (3.2.2) for $U_i(x,t)$ and obtain $V_3(L,t) = \frac{1}{5}[U_1(L,t) + 2U_2(L,t) + 2U_3(L,t)]$, which is given by a sum of a constant and three exponent terms, $V_{dd}(1 - c_0 e^{-\frac{t}{a_0 \tau}} - c_1 e^{-\frac{t}{a_1 \tau}} - c_2 e^{-\frac{t}{a_2 \tau}})$. Then the 50% delay of wire 3 can be evaluated by solving $V_3(L,t) = 0.5V_{dd}$.

For side wires, PDEs characterizing four wires with length L are given by:

$$\frac{\partial^2}{\partial x^2} \mathbf{V}(x,t) = \mathbf{RC} \frac{\partial}{\partial t} \mathbf{V}(x,t),$$

Table 3.1: Subclassification of patterns by signal expressions on wire 3 in a five-wire bus.

Subclass k	Patterns	Subclass k	Patterns
1	↑↑↑↑↑	13	- -↑- -, ↑-↑-↓, ↓-↑-↑ -↑↑↓-, ↑↑↑↓↓, ↓↑↑↓↑ -↓↑↑-, ↑↓↑↑↓, ↓↓↑↑↑
2	-↑↑↑↑, ↑↑↑↑-		
3	↑-↑↑↑, ↑↑↑-↑		
4	-↑↑↑-, ↓↑↑↑↑, ↑↑↑↑↓	14	- -↑-↓, ↓-↑- -, -↑↑↓↓, ↓↑↑↓-, -↓↑↑↓, ↓↓↑↑-
5	- -↑↑↑, ↑↑↑- -, -↑↑-↑, ↑-↑↑-	15	↓-↑-↓, ↓↑↑↓↓, ↓↓↑↑↓
		16	↓↓↑-↓, ↓-↑↓↓
6	↑-↑-↑, ↑↑↑↓↑, ↑↓↑↑↑	17	- -↑↓↓, ↓↓↑- -, -↓↑-↓, ↓-↑↓-
7	-↑↑↑↓, ↓↑↑↑-	18	- -↑↓-, -↓↑- -, ↑-↑↓↓, ↑↓↑-↓, ↓-↑↓↑, ↓↓↑-↑
8	- -↑↑-, -↑↑- -, ↓-↑↑↑, ↓↑↑-↑, ↑-↑↑↓, ↑↑↑-↓	19	- -↑↓↑, ↑↓↑- -, -↓↑-↑, ↑-↑↓-
9	↓↑↑↑↓	20	↑-↑↓↑, ↑↓↑-↑
10	- -↑↑↓, ↓↑↑- -, -↑↑-↓, ↓-↑↑-	21	↓↓↑↓↓
		22	↓↓↑↓-, -↓↑↓↓
11	↓-↑↑↓, ↓↑↑-↓	23	-↓↑↓-, ↑↓↑↓↓, ↓↓↑↓↑
12	- -↑-↑, ↑-↑- -, -↑↑↓↑, ↑↑↑↓-, -↓↑↑↑, ↑↓↑↑-	24	↑↓↑↓-, -↓↑↓↑
		25	↑↓↑↓↑

where $\mathbf{R} = \text{diag}\{r\ r\ r\ r\}$, $\mathbf{V}(x,t) = [V_1(x,t)\ V_2(x,t)\ V_3(x,t)\ V_4(x,t)]^T$, and $\mathbf{C} = c\begin{bmatrix} 1+\lambda & -\lambda & 0 & 0 \\ -\lambda & 1+2\lambda & -\lambda & 0 \\ 0 & -\lambda & 1+2\lambda & -\lambda \\ 0 & 0 & -\lambda & 1+\lambda \end{bmatrix}$.

The eigenvalues of $\mathbf{C}/c$ are given by $p_1 = 1$, $p_2 = 1 + (2 - \sqrt{2})\lambda$, $p_3 = 1 + 2\lambda$, and $p_4 = 1 + (2 + \sqrt{2})\lambda$. Their corresponding eigenvectors $\mathbf{e}_i$'s are given by $\mathbf{e}_1 = [1\ \ 1\ \ 1\ \ 1]^T$, $\mathbf{e}_2 = [-1\ \ (1 - \sqrt{2})\ \ -(1 - \sqrt{2})\ \ 1]^T$, $\mathbf{e}_3 = [1\ \ -1\ \ -1\ \ 1]^T$, and $\mathbf{e}_4 = [-1\ (1 + \sqrt{2})\ \ -(1 + \sqrt{2})\ 1]^T$, respectively.

By decoupling the PDEs for side wires, we have

$$\frac{\partial^2}{\partial x^2} U_i(x,t) = rcp_i \frac{\partial}{\partial t} U_i(x,t), \text{ for } i = 1, 2, 3, 4,$$

Table 3.2: Closed-form expressions for the output signals on wire 3 ($V_3(L,t) = V_{dd}(1 - c_0 e^{-\frac{t}{a_0\tau}} - c_1 e^{-\frac{t}{a_1\tau}} - c_2 e^{-\frac{t}{a_2\tau}})$ in a five-wire bus with evaluated and simulated 50% delays ($\tau_0 = 1.42$ ps, $\tau = \frac{8}{\pi^2}\tau_0$, $\lambda = 12.24$, $a_0 = 1$, $a_1 = 1 + \frac{5-\sqrt{5}}{2}\lambda$, and $a_2 = 1 + \frac{5+\sqrt{5}}{2}\lambda$ for all classes).

Ci	Subclass k	Coeffs. of $V_3(L,t)$			Eva. (ps)	Sim. (ps)
		c_0	c_1	c_2		
0	1	$\frac{4}{\pi}$	0	0	1.08	1.18
	2	$\frac{16}{5\pi}$	$\frac{2(1+\sqrt{5})}{5\pi}$	$\frac{2(1-\sqrt{5})}{5\pi}$	**1.41**	**1.50**
	3	$\frac{16}{5\pi}$	$\frac{2(1-\sqrt{5})}{5\pi}$	$\frac{2(1+\sqrt{5})}{5\pi}$	**1.41**	**1.50**
1	4	$\frac{12}{5\pi}$	$\frac{4(1+\sqrt{5})}{5\pi}$	$\frac{4(1-\sqrt{5})}{5\pi}$	**2.35**	2.40
	5	$\frac{12}{5\pi}$	$\frac{4}{5\pi}$	$\frac{4}{5\pi}$	**2.35**	2.40
	6	$\frac{12}{5\pi}$	$\frac{4(1-\sqrt{5})}{5\pi}$	$\frac{4(1+\sqrt{5})}{5\pi}$	**2.35**	**2.45**
2	7	$\frac{8}{5\pi}$	$\frac{6(1+\sqrt{5})}{5\pi}$	$\frac{6(1-\sqrt{5})}{5\pi}$	6.17	6.84
	8	$\frac{8}{5\pi}$	$\frac{2(3+\sqrt{5})}{5\pi}$	$\frac{2(3-\sqrt{5})}{5\pi}$	9.62	9.21
	9	$\frac{4}{5\pi}$	$\frac{8(1+\sqrt{5})}{5\pi}$	$\frac{8(1-\sqrt{5})}{5\pi}$	**9.90**	**10.70**
3	10	$\frac{4}{5\pi}$	$\frac{4(2+\sqrt{5})}{5\pi}$	$\frac{4(2-\sqrt{5})}{5\pi}$	14.07	14.22
	11	0	$\frac{2(5+3\sqrt{5})}{5\pi}$	$\frac{2(5-3\sqrt{5})}{5\pi}$	16.91	17.18
	12	$\frac{8}{5\pi}$	$\frac{2(3-\sqrt{5})}{5\pi}$	$\frac{2(3+\sqrt{5})}{5\pi}$	**19.24**	**18.47**
4	13	$\frac{4}{5\pi}$	$\frac{8}{5\pi}$	$\frac{8}{5\pi}$	22.67	22.60
	14	0	$\frac{2(5+\sqrt{5})}{5\pi}$	$\frac{2(5-\sqrt{5})}{5\pi}$	24.58	24.68
	15	$-\frac{4}{5\pi}$	$\frac{4(3+\sqrt{5})}{5\pi}$	$\frac{4(3-\sqrt{5})}{5\pi}$	**25.84**	**26.03**
5	16	$-\frac{8}{5\pi}$	$\frac{2(7+\sqrt{5})}{5\pi}$	$\frac{2(7-\sqrt{5})}{5\pi}$	36.63	36.91
	17	$-\frac{4}{5\pi}$	$\frac{12}{5\pi}$	$\frac{12}{5\pi}$	37.24	37.52
	18	0	$\frac{2(5-\sqrt{5})}{5\pi}$	$\frac{2(5+\sqrt{5})}{5\pi}$	38.07	38.35
	19	$\frac{4}{5\pi}$	$\frac{4(2-\sqrt{5})}{5\pi}$	$\frac{4(2+\sqrt{5})}{5\pi}$	39.22	39.47
	20	$\frac{8}{5\pi}$	$\frac{6(1-\sqrt{5})}{5\pi}$	$\frac{6(1+\sqrt{5})}{5\pi}$	**40.87**	**41.11**
6	21	$-\frac{12}{5\pi}$	$\frac{16}{5\pi}$	$\frac{16}{5\pi}$	48.43	48.85
	22	$-\frac{8}{5\pi}$	$\frac{2(7-\sqrt{5})}{5\pi}$	$\frac{2(7+\sqrt{5})}{5\pi}$	50.43	50.86
	23	$-\frac{4}{5\pi}$	$\frac{4(3-\sqrt{5})}{5\pi}$	$\frac{4(3+\sqrt{5})}{5\pi}$	52.78	53.25
	24	0	$\frac{4(5-3\sqrt{5})}{5\pi}$	$\frac{4(5+3\sqrt{5})}{5\pi}$	55.48	55.97
	25	$\frac{4}{5\pi}$	$\frac{8(1-\sqrt{5})}{5\pi}$	$\frac{8(1+\sqrt{5})}{5\pi}$	**58.52**	**59.04**

The expressions of wires 1 and 2 are given by $V_1(L,t) = \frac{1}{4}U_1(L,t) - \frac{2+\sqrt{2}}{8}U_2(L,t) + \frac{1}{4}U_3(L,t) - \frac{2-\sqrt{2}}{8}U_4(L,t)$ and $V_2(L,t) = \frac{1}{4}U_1(L,t) - \frac{\sqrt{2}}{8}U_2(L,t) - \frac{1}{4}U_3(L,t) + \frac{\sqrt{2}}{8}U_4(L,t)$, respectively. Then the 50% delays of wires 1 and 2 can be evaluated by solving $V_i(L,t) = 0.5V_{dd}$ for $i = 1, 2$.

3.2.3 Pattern Classification

First, we consider the classification of transition patterns over five wires with respect to the delay of the middle wire (wire 3). In this chapter, we use "↑" to denote a transition from 0 to the supply voltage V_{dd} (normalized to 1), "-" no transition, and "↓" a transition from V_{dd} to 0. We first focus on patterns with a ↑ transition on wire 3 in a five-wire bus and derive $V_3(L,t)$ for each pattern as described in Sec. 3.2.2. There are $3^4 = 81$ different transition patterns, which can be partitioned into 25 subclasses as shown in Tab. 3.1 according to the expressions of the output signals on wire 3: All transition patterns in each subclass have the same expression $V_3(L,t)$. The coefficients for all 25 subclasses are shown in columns 3-5 of Tab. 3.2. Then the expressions $V_3(L,t)$ of all patterns in the 25 subclasses are evaluated for their 50% delays. By grouping subclasses with close delays into one class, we can divide the 81 transition patterns into seven classes Ci for $i = 0, 1, \cdots, 6$ shown in Tab. 3.2. For all 25 subclasses, evaluated and simulated delays are provided in columns 6 and 7 of Tab. 3.2, respectively. For all seven classes, the difference between evaluated delay and simulated delay in Tab. 3.2 is small.

All evaluations and simulations are based on a freePDK 45nm CMOS technology with 10 metal layers [49]. We assume that the top two metal layers, layers 9 and 10, are used for routing global interconnects, and that metal layer 8 is used as the ground layer. An interconnect model in [50] is used for parasitic extraction. For a 5mm bus in the top metal layer, the key parasitics, resistance, ground capacitance, and coupling capacitance, are given by $R = 68.75\Omega$, $C_{gnd} = 41.32fF$, and $C_{couple} = 505.68fF$, respectively. The bus is modeled by a distributed RC model as shown in Fig. 4.1 with 100 segments. The two important parameters used in our delay

approximation are $\tau_0 = 0.5RC_{gnd} = 1.42$ps and $\lambda = C_{couple}/C_{gnd} = 12.24$. Since the crosstalk delay on the bus constitutes a major part of the whole delay, the delays introduced by buffers are ignored. We assume that ideal step signals are applied on the bus directly. The closed-form expressions are evaluated for 50% delays via MATLAB and the simulation is done by HSPICE.

From Tab. 3.2, it can be easily verified that $C5$ and $C6$ are the same as $D3$ and $D4$ in [7, 42], respectively. That is, the middle three wires of the transition patterns in $C5$ ($C6$, respectively) constitute $D3$ ($D4$, respectively). The transition patterns in $D0$, $D1$, and $D2$ are divided into five classes $C0$—$C4$ in our classification with following relations, $C4 \subset D2$, $C3 \subset D1 \cup D2$, $C2 \subset D0 \cup D1$, $C1 \subset D0 \cup D1 \cup D2$, and $C0 \subset D0 \cup D1$.

Note that the coefficients c_i for $i = 0, 1, 2$ of the expression of wire 3 are independent of technology and determined by different patterns. For a given pattern, the coefficients c_i are fixed and the delay is a function of τ_0 and λ. Since the ratio t/τ_0 appears in the exponent term, varying τ_0 would scale delays in all classes. Thus, the classification does not depend on τ_0. The coupling factor λ could affect the delay differently. In the following, we verify our classification for technology with different coupling factor, $\lambda = 1, 2, \cdots, 13$, and show the results in Fig. 3.2. Different classes are denoted by different line styles. Each class contains multiple lines, which represents a subclass. Patterns in each subclass have the same delay. For $\lambda \geq 3$, the ranges of delays in all classes do not overlap. Also, the delay in each subclass increases linearly with λ. This implies that our classification is valid provided that the coupling factor λ is at least 3.

Then, we consider the classification of transition patterns over four wires with

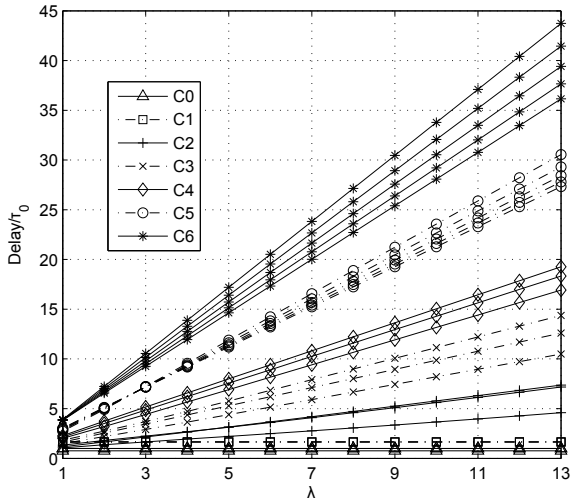


Figure 3.2: Delays of the middle wire for all patterns with respect to λ in a five-wire bus ($\tau_0 = 1.42$ps).

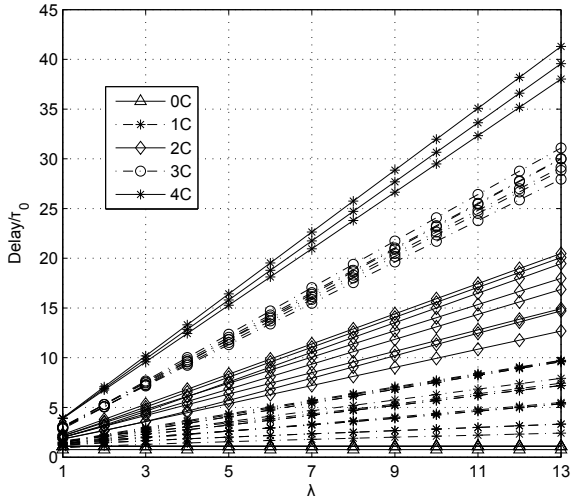


Figure 3.3: Delays of side wires for all patterns with respect to λ in a four-wire bus ($\tau_0 = 1.42$ps).

respect to the delays of the side wires. We classify patterns by considering the worst-case delays of wires 1 and 2, respectively. Note that the classification with respect

to the delays of wires 4 and 5 would be the same by symmetry. We first focus on patterns with a $\uparrow$ transition on wire 2 in a four-wire bus. There are $3^3 = 27$ different transition patterns. As described in Sec. 3.2.2, we first derive the expressions $V_2(L,t)$ of these 27 patterns shown in Tab. 3.3. By evaluating these patterns for their 50% delays, we group patterns with close delays into one class, and form five classes jC for $j = 0, 1, 2, 3, 4$ as shown in Tab. 3.3. Then, we focus on patterns with a $\uparrow$ transition on wire 1. There are $3^3 = 27$ different transition patterns. As described in Sec. 3.2.2, we first derive the expressions $V_1(L,t)$ of these 27 patterns shown in Tab. 3.4. By evaluating these patterns for their 50% delays, we group patterns with close delays into one class, and form three classes jC for $j = 0, 1, 2$ as shown in Tab. 3.4. When both wires 1 and 2 have transitions, the delay on wire 2 is larger than that of wire 1, which can be verified from Tabs. 3.3 and 3.4. In this case, we focus on the delay of wire 2. When only wire 1 has transition, we focus on the delay of wire 1. The difference between evaluated delay and simulated delay is small as shown in Tabs. 3.3 and 3.4 with one exception (the pattern $\uparrow\uparrow\downarrow\uparrow$ in $1C$ in Tab. 3.3), which doesn't change our classification.

From Tabs. 3.3 and 3.4, the classes $3C$ and $4C$ of our classification are exactly the same as $D3$ and $D4$ in [7,42], respectively. The class $1C$ and $2C$ of our classification are subsets of $D1$ and $D2$ in [7,42], respectively. The class $0C$ is a subset of $D0 \cup D1$ in [7,42].

Similar to the classification of middle wires, we conclude that the classification on side wires does not depend on τ_0. To verify our classification for technology with different coupling effects, we consider coupling factor $\lambda = 1, 2, \cdots, 13$, and show the results in Fig. 3.3. Each class contains multiple lines, each of which represents a

Table 3.3: Closed-form expressions for the output signals on wire 2 ($V_2(L,t) = V_{dd}(1-c_0e^{-\frac{t}{a_0\tau}}-c_1e^{-\frac{t}{a_1\tau}}-c_2e^{-\frac{t}{a_2\tau}}-c_3e^{-\frac{t}{a_3\tau}})$) in a four-wire bus with evaluated and simulated 50% delays ($\tau_0 = 1.42$ ps, $\tau = \frac{8}{\pi^2}\tau_0$, $\lambda = 12.24$, $a_0 = 1$, $a_1 = 1+(2-\sqrt{2})\lambda$, $a_2 = 1+2\lambda$, and $a_3 = 1+(2+\sqrt{2})\lambda$ for all classes).

jC	Pattern	Coeffs of $V_2(L,t)$				Eva. (ps)	Sim. (ps)
		c_0	c_1	c_2	c_3		
0	↑↑↑↑	$\frac{4}{\pi}$	0	0	0	1.08	1.18
	↑↑↑-	$\frac{3}{\pi}$	$\frac{\sqrt{2}}{2\pi}$	$\frac{1}{\pi}$	$-\frac{\sqrt{2}}{2\pi}$	**1.55**	1.61
	↑↑-↑	$\frac{3}{\pi}$	$\frac{2-\sqrt{2}}{2\pi}$	$-\frac{1}{\pi}$	$-\frac{2+\sqrt{2}}{2\pi}$	**1.55**	1.62
	-↑↑↑	$\frac{3}{\pi}$	$-\frac{\sqrt{2}}{2\pi}$	$\frac{1}{\pi}$	$\frac{\sqrt{2}}{2\pi}$	**1.55**	**1.64**
1	↑↑↑↓	$\frac{2}{\pi}$	$\frac{\sqrt{2}}{\pi}$	$\frac{2}{\pi}$	$-\frac{\sqrt{2}}{\pi}$	3.33	3.22
	↑↑- -	$\frac{2}{\pi}$	$\frac{1}{\pi}$	0	$\frac{1}{\pi}$	4.54	3.48
	-↑↑-	$\frac{2}{\pi}$	0	$\frac{2}{\pi}$	0	7.21	5.15
	↑↑-↓	$\frac{1}{\pi}$	$\frac{2+\sqrt{2}}{2\pi}$	$\frac{1}{\pi}$	$\frac{2-\sqrt{2}}{2\pi}$	9.70	9.38
	↑↑↓↑	$\frac{2}{\pi}$	0	$\frac{2-\sqrt{2}}{2\pi}$	$-\frac{2}{\pi}$	9.98	3.92
	-↑↑↓	$\frac{1}{\pi}$	$\frac{\sqrt{2}}{2\pi}$	$\frac{3}{\pi}$	$\frac{-\sqrt{2}}{2\pi}$	**12.89**	**13.03**
2	↑↑↓-	$\frac{1}{\pi}$	$\frac{4-\sqrt{2}}{2\pi}$	$-\frac{1}{\pi}$	$\frac{4+\sqrt{2}}{2\pi}$	17.02	16.05
	-↑-↑	$\frac{2}{\pi}$	$\frac{1-\sqrt{2}}{\pi}$	0	$\frac{1+\sqrt{2}}{\pi}$	19.67	18.79
	↑↑↓↓	0	$\frac{2}{\pi}$	0	$\frac{2}{\pi}$	20.05	19.85
	-↑- -	$\frac{1}{\pi}$	$\frac{2-\sqrt{2}}{2\pi}$	$\frac{1}{\pi}$	$\frac{2+\sqrt{2}}{2\pi}$	22.59	22.48
	-↑-↓	0	$\frac{1}{\pi}$	$\frac{2}{\pi}$	$\frac{1}{\pi}$	24.12	24.22
	↓↑↑↑	$\frac{2}{\pi}$	$-\frac{\sqrt{2}}{\pi}$	$\frac{2}{\pi}$	$\frac{\sqrt{2}}{\pi}$	26.02	26.06
	↓↑↑-	$\frac{1}{\pi}$	$-\frac{\sqrt{2}}{2\pi}$	$\frac{3}{\pi}$	$\frac{\sqrt{2}}{2\pi}$	26.89	27.06
	↓↑↑↓	0	0	$\frac{4}{\pi}$	0	**27.45**	**27.68**
3	-↑↓↓	$-\frac{1}{\pi}$	$\frac{4-\sqrt{2}}{2\pi}$	$\frac{1}{\pi}$	$\frac{4+\sqrt{2}}{2\pi}$	37.44	37.74
	-↑↓-	0	$\frac{2-\sqrt{2}}{\pi}$	0	$\frac{2+\sqrt{2}}{\pi}$	38.61	38.89
	↓↑-↓	$-\frac{1}{\pi}$	$\frac{2-\sqrt{2}}{2\pi}$	$\frac{3}{\pi}$	$\frac{2+\sqrt{2}}{2\pi}$	39.06	39.40
	-↑↓↑	$\frac{1}{\pi}$	$\frac{4-\sqrt{2}}{2\pi}$	$-\frac{1}{\pi}$	$\frac{4+\sqrt{2}}{2\pi}$	40.12	40.39
	↓↑- -	0	$\frac{1-\sqrt{2}}{\pi}$	$\frac{2}{\pi}$	$\frac{1+\sqrt{2}}{\pi}$	40.21	40.55
	↓↑-↑	$\frac{1}{\pi}$	$\frac{2-3\sqrt{2}}{2\pi}$	$\frac{1}{\pi}$	$\frac{2+3\sqrt{2}}{2\pi}$	**41.63**	**41.98**
4	↓↑↓↓	$-\frac{2}{\pi}$	$\frac{2-\sqrt{2}}{\pi}$	$\frac{2}{\pi}$	$\frac{2+\sqrt{2}}{\pi}$	50.92	51.36
	↓↑↓-	$-\frac{1}{\pi}$	$\frac{4-3\sqrt{2}}{2\pi}$	$\frac{1}{\pi}$	$\frac{4+3\sqrt{2}}{2\pi}$	52.99	53.44
	↓↑↓↑	0	$\frac{2-2\sqrt{2}}{\pi}$	0	$\frac{2+2\sqrt{2}}{\pi}$	**55.28**	**55.79**

pattern in Tabs. 3.3 and 3.4. For $\lambda \geq 1$, the ranges of delays in all classes do not overlap. Also, the delay in each subclass increases linearly with λ. This implies that our classification on side wires is valid provided that the coupling factor λ is at least 1.

Table 3.4: Closed-form expressions for the output signals on wire 1 ($V_1(L,t) = V_{dd}(1-c_0e^{-\frac{t}{a_0\tau}}-c_1e^{-\frac{t}{a_1\tau}}-c_2e^{-\frac{t}{a_2\tau}}-c_3e^{-\frac{t}{a_3\tau}})$) in a four-wire bus with evaluated and simulated 50% delays ($\tau_0 = 1.42$ ps, $\tau = \frac{8}{\pi^2}\tau_0$, $\lambda = 12.24$, $a_0 = 1$, $a_1 = 1+(2-\sqrt{2})\lambda$, $a_2 = 1+2\lambda$, and $a_3 = 1+(2+\sqrt{2})\lambda$ for all classes).

jC	Pattern	Coeffs of $V_1(L,t)$				Eva. (ps)	Sim. (ps)
		c_0	c_1	c_2	c_3		
0	↑↑↑↑	$\frac{4}{\pi}$	0	0	0	1.08	1.18
	↑↑↑-	$\frac{3}{\pi}$	$-\frac{2+\sqrt{2}}{2\pi}$	$-\frac{1}{\pi}$	$\frac{2-\sqrt{2}}{2\pi}$	**1.55**	1.59
	↑↑-↑	$\frac{3}{\pi}$	$\frac{\sqrt{2}}{2\pi}$	$\frac{1}{\pi}$	$-\frac{\sqrt{2}}{2\pi}$	**1.55**	1.61
	↑-↑↑	$\frac{3}{\pi}$	$-\frac{\sqrt{2}}{2\pi}$	$\frac{1}{\pi}$	$\frac{\sqrt{2}}{2\pi}$	**1.55**	**1.64**
1	↑↑↑↓	$\frac{2}{\pi}$	$\frac{2+\sqrt{2}}{\pi}$	$-\frac{2}{\pi}$	$\frac{2-\sqrt{2}}{\pi}$	2.50	2.70
	↑↑- -	$\frac{2}{\pi}$	$\frac{1+\sqrt{2}}{\pi}$	0	$\frac{1-\sqrt{2}}{\pi}$	2.83	2.90
	↑↑↓↑	$\frac{2}{\pi}$	$\frac{\sqrt{2}}{\pi}$	$\frac{2}{\pi}$	$-\frac{\sqrt{2}}{\pi}$	3.33	3.20
	↑↑-↓	$\frac{1}{\pi}$	$\frac{4+3\sqrt{2}}{2\pi}$	$-\frac{1}{\pi}$	$\frac{4-3\sqrt{2}}{2\pi}$	4.65	4.99
	↑-↑-	$\frac{2}{\pi}$	$\frac{1}{2\pi}$	0	$\frac{1}{2\pi}$	4.54	3.49
	↑↑↓-	$\frac{1}{\pi}$	$\frac{2+3\sqrt{2}}{2\pi}$	$\frac{1}{\pi}$	$\frac{2-3\sqrt{2}}{2\pi}$	5.53	5.88
	↑↑↓↓	0	$\frac{2+2\sqrt{2}}{\pi}$	0	$\frac{2-2\sqrt{2}}{\pi}$	7.03	7.39
	↑- -↑	$\frac{2}{\pi}$	0	$\frac{2}{\pi}$	0	7.21	5.15
	↑-↑↓	$\frac{1}{\pi}$	$\frac{4+\sqrt{2}}{2\pi}$	$-\frac{1}{\pi}$	$\frac{4-\sqrt{2}}{2\pi}$	7.41	6.89
	↑- - -	$\frac{1}{\pi}$	$\frac{2+\sqrt{2}}{2\pi}$	$\frac{1}{\pi}$	$\frac{2-\sqrt{2}}{2\pi}$	9.70	9.35
	↑- -↓	0	$\frac{2+\sqrt{2}}{\pi}$	0	$\frac{2-\sqrt{2}}{\pi}$	10.68	10.54
	↑-↓↑	$\frac{1}{\pi}$	$\frac{\sqrt{2}}{2\pi}$	$\frac{3}{\pi}$	$\frac{-\sqrt{2}}{2\pi}$	12.89	13.03
	↑-↓-	0	$\frac{2+2\sqrt{2}}{2\pi}$	$\frac{2}{\pi}$	$\frac{2-2\sqrt{2}}{2\pi}$	13.03	13.14
	↑-↓↓	$-\frac{1}{\pi}$	$\frac{4+3\sqrt{2}}{2\pi}$	$\frac{1}{\pi}$	$\frac{4-3\sqrt{2}}{2\pi}$	**13.11**	**13.21**
2	↑↓↑↓	0	$\frac{2}{\pi}$	0	$\frac{2}{\pi}$	20.05	19.85
	↑↓-↓	$-\frac{1}{\pi}$	$\frac{4+\sqrt{2}}{2\pi}$	$\frac{1}{\pi}$	$\frac{4-\sqrt{2}}{2\pi}$	21.86	21.91
	↑↓↑-	$\frac{1}{\pi}$	$\frac{2-\sqrt{2}}{2\pi}$	$\frac{1}{\pi}$	$\frac{2+\sqrt{2}}{2\pi}$	22.59	22.48
	↑↓↓↓	$-\frac{2}{\pi}$	$\frac{2+\sqrt{2}}{\pi}$	$\frac{2}{\pi}$	$\frac{2-\sqrt{2}}{\pi}$	23.10	23.23
	↑↓- -	0	$\frac{1}{\pi}$	$\frac{2}{\pi}$	$\frac{1}{\pi}$	24.12	24.22
	↑↓↓-	$-\frac{1}{\pi}$	$\frac{2+\sqrt{2}}{2\pi}$	$\frac{3}{\pi}$	$\frac{2-\sqrt{2}}{2\pi}$	25.10	25.30
	↑↓↑↑	$\frac{2}{\pi}$	$\frac{-\sqrt{2}}{\pi}$	$\frac{2}{\pi}$	$\frac{\sqrt{2}}{\pi}$	26.02	26.06
	↑↓-↑	$\frac{1}{\pi}$	$-\frac{\sqrt{2}}{2\pi}$	$\frac{3}{\pi}$	$\frac{\sqrt{2}}{2\pi}$	26.89	27.06
	↑↓↓↑	0	0	$\frac{4}{\pi}$	0	**27.45**	**27.68**

In addition to being a finer classification, the new classification has no overlapping delays among different classes. Fig. 3.4 compares the simulated delays of different classes based on the classification in [7, 42] and our new classification. In Fig. 3.4, the grey bars identify the minimum and maximum simulated delays in every class. Note that only two extremes are important, and not all delay values in

the grey bars are achievable by some transition patterns. In Fig. 3.4(a), the thick line segments denote the upper bounds for delay of each class based on Eq. (3.1). The upper bounds by the model in [7, 42] overestimate the delays of $D1$ through $D4$ and underestimate the delay of $D0$. As shown in Fig. 3.4(a), the actual delays in $D0$, $D1$, and $D2$ overlap with each other. Some patterns with smaller delays have potential to transmit information at a higher speed, but are categorized into a class with a larger delay bound. Thus, the classification by the model in [7, 42] does not result in effective crosstalk avoidance codes. In contrast, the delays of different classes in our new classification do not overlap as shown in Fig. 3.4(b), 4(c), and 4(d). By classifying patterns this way, we have a more accurate control of delays for transition patterns.

3.3 NEW MEMORYLESS CROSSTALK AVOIDANCE CODES

3.3.1 Previous CAC Design

CACs reduce the crosstalk delay for on-chip global interconnects by encoding a k-bit data word $(x_1x_2\cdots x_k)$ into an n-bit $(n > k)$ codeword $(c_1c_2\cdots c_n)$. Two kinds of CACs, CACs with memory and memoryless CACs, have been investigated in the literature [51]. CACs with memory need to store all codebooks corresponding to different codewords $(c_1c_2\cdots c_n)$, since the encoding depends on the data word $(x_1x_2\cdots x_k)$ as well as the preceding codeword. In contrast, memoryless CACs

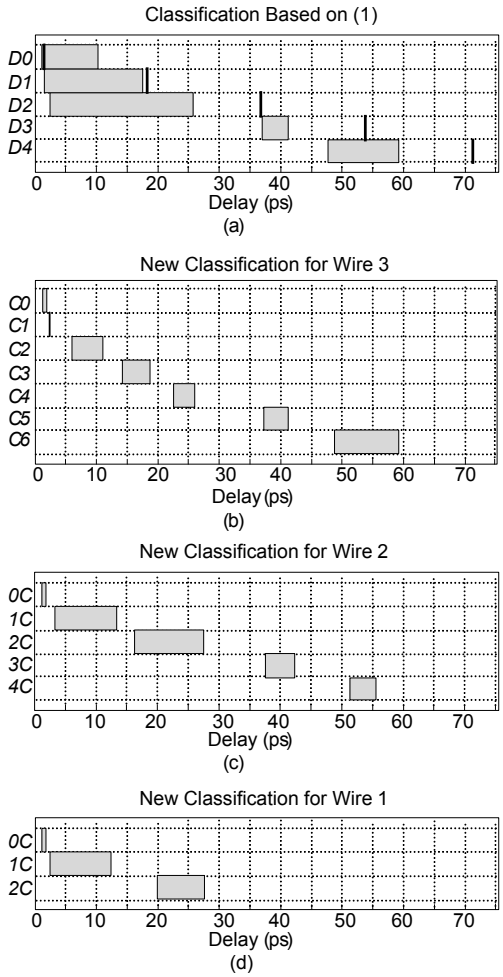


Figure 3.4: Simulated delays of different classes of transition patterns using (a) Classification based on (3.1); (b) Classification with respect to the delay of the middle wire in a five-wire bus; (c) Classification with respect to the delay of wire 2 in a four-wire bus; (d) Classification with respect to the delay of wire 1 in a four-wire bus ($\lambda = 12.24$ and $\tau_0 = 1.42$ps).

require a single codebook to generate codewords for transmission, because the encoding depends on the data word only. Hence, memoryless CACs are simpler to implement than CACs with memory. We focus on memoryless CACs in this chapter.

The codebook of a memoryless CAC satisfies the property that each codeword

must be able to transition to every other codeword in the codebook with a delay less than the requirement. Most memoryless CACs in the literature are based on the model in [7, 42]. The key idea is to eliminate undesirable patterns for transmission. Existing memoryless CACs include OLCs, FPCs, FTCs, and FOCs [8–10, 32], which achieve a worst-case delay of $(1 + \lambda)\tau_0$, $(1 + 2\lambda)\tau_0$, $(1 + 2\lambda)\tau_0$, and $(1 + 3\lambda)\tau_0$, respectively. As mentioned above, the scheme that was proposed to achieve a worst-case delay of τ_0 is invalid since the model in [7, 42] underestimates the delays for $0C$. Thus, OLCs achieve the smallest worst-case delay $(1 + \lambda)\tau_0$ among existing CACs.

There exist several methods to obtain a memoryless codebook based on pattern pruning, transition pruning, or recursive construction. The pattern pruning technique is quite straightforward, and gives a codebook with a smaller worst-case delay by eliminating some patterns. For example, FOCs cannot have both 010 and 101 patterns around any bit position, and FPCs are free of 010 and 101 patterns [32]. The transition pruning technique [10] is based on graph theory. This method first builds a transition graph with all possible codewords as nodes and all valid transitions as edges, and then finds a maximum clique. A clique is defined as a subgraph where every pair of nodes are connected with an edge. A maximum clique is defined as a clique of the largest possible size in a given graph. Since every pair of nodes is connected, a maximum clique in this graph constitutes a memoryless codebook with the largest size. The codebook generation method is based on exhaustive search. Although it is easy to get a maximum clique from a transition graph with a small n, the complexity increases rapidly with n. This is because the number of edges in an n-bit transition graph is upper bounded by $2^{n-1}(2^n - 1)$, which increases exponentially with n. In fact, it is an NP problem to find a maximum clique for given

constraints [52]. The recursive technique constructs an $(n+1)$-bit codebook from an n-bit codebook [8,9]. Since for a small n, a largest codebook can be obtained easily via the second method, a codebook for an n-wire bus can be constructed recursively.

3.3.2 CAC Design with New Classification

Since our classification of patterns is different from that in [7,42], the CAC designs should be reconsidered with our new classification. In the following, we first introduce a recursive method for codebook construction under different constraints, and then derive the size of codebooks.

In our work, we use the recursive method to obtain a memoryless codebook for the following two reasons. First, it is complex to apply the pattern pruning technique, since our new classification is based on transitions over five wires, and it is not clear which patterns have larger worst-case delays and should be removed. Second, it is hard to find a maximum clique for a transition graph with a large n. In our method, we first start with a 5-bit codebook, obtained by searching for maximum cliques in a five-wire bus, and then build an $(n+1)$-bit codebook by appending '0' and '1' to codewords of an n-bit codebook while satisfying delay constraints.

Our new classifications partition patterns over five adjacent wires into seven classes, $C0$ to $C6$, and patterns over four adjacent wires into five classes, $0C$ to $4C$. Similar to the CAC design based on the model in [7,42], the new classifications are conducive to the design of CACs by eliminating undesirable transition patterns with large worst-case delays.

To get valid 5-bit codebooks, we first assume the allowed patterns are from $C0$ to Ci for $i = 0, 1, \cdots, 6$ in our classification for middle wires. Then, for the side

wires, we assume patterns are from $0C$ to jC based on the classification for side wires. Under these two assumptions, there are many configurations of constraints, which are referred as (Ci, jC), where $i \in \{0, 1, \cdots, 6\}$ and $j \in \{0, 1, \cdots, 4\}$.

Since the worst-case delay of a bus is determined by the largest delays among all wires, for an n-bit ($n \geq 5$) bus under (Ci, jC) we require that the worst-case delays on middle wires and side wires are close enough. By our classifications, we find $0C$ is close to $C0$, $1C$ close to $C2$ and $C3$, $2C$ close to $C4$, $3C$ close to $C5$, and $4C$ close to $C6$. Hence, among all configurations of constraints (Ci, jC), we only focus on $(C0, 0C)$, $(C2, 1C)$, $(C3, 1C)$, $(C4, 2C)$, $(C5, 3C)$, and $(C6, 4C)$. When $n \leq 4$, the constraint Ci cannot be enforced. Hence, the constraint (Ci, jC) reduces to jC. The constraint $(C0, 0C)$ appears to be too restrictive, and hence we do not investigate it in this chapter. The last configuration $(C6, 4C)$ is trivial, since it allows arbitrary transitions.

Algorithm 1 Codebook design under (Ci, jC)

Input: C_5^0, C_5^1, n;
  Initialize: $k = 5$, $C_5 = C_5^0$, $s = 1$;
  while $k \leq n - 1$ **do**
    for $\forall \mathbf{c}_k = (c_1 c_2 \cdots c_k) \in C(k)$ **do**
      if $(c_{k-3} c_{k-2} c_{k-1} c_k 0) \in C_5^s$ **then**
        append 0 to $\mathbf{c}_k$ and add the new codeword to $C(k+1)$;
      else if $(c_{k-3} c_{k-2} c_{k-1} c_k 1) \in C_5^s$ **then**
        append 1 to $\mathbf{c}_k$ and add the new codeword to $C(k+1)$;
      end if
    end for
    $s = 1 - s$;
    $k = k + 1$;
  end while
  Output: $C(n)$.

In the following, we propose a scheme for finding an n-bit codebook $C_{(Ci,jC)}(n)$.

For simplicity, we denote $C_{(Ci,jC)}(n)$ as $C(n)$ when there is no ambiguity about the constraint. First, for a five-wire bus under constraint (Ci, jC), a pattern transition graph is obtained. We search the graph for the largest 5-bit codebooks. One or two 5-bit codebooks of maximum sizes exist for each constraint in Tab. 3.5, where we denote an n-bit binary codeword $(c_1c_2\cdots c_n)$ as a decimal number $\sum_{i=1}^{n} c_i 2^{n-i}$ for simplicity. In [10], a bit boundary in a set of codewords is said to be 01-type if only codewords with 00, 01, and 11 are allowed across that boundary, and a bit boundary is said to be 10-type when only codewords with 00, 10, and 11 are allowed across that boundary. It is shown that the largest clique for a given constraint has alternating boundary types. Thus, there are two largest cliques. Similarly, from Tab. 3.5, we conjecture that the largest codebooks have alternating constraints, C_5^0 and C_5^1, for every five consecutive wires. For constraint $(C4, 2C)$, only one maximum 5-bit codebook exists. We assume C_5^1 is the same as C_5^0 for constraint $(C4, 2C)$. Since we have two types of constraints, two largest codebooks for each constraint can be obtained, except for $(C4, 2C)$, where the two codebooks are the same. Then we apply Alg. 1 to obtain $C(n)$. In the initialization, we pick a 5-bit codebook $C_5 = C_5^0$. Then, the algorithm recursively appends one bit to the codewords in the codebook in each iteration. For $\mathbf{c}_k = (c_1c_2\cdots c_k)$, the appended bit x needs to satisfy that the last five bits $(c_{k-3}c_{k-2}c_{k-1}c_kx)$ form a codeword in C_5^s, which alternates between C_5^0 and C_5^1. If we pick the other 5-bit codebook $C_5 = C_5^1$, we would obtain another codebook.

The recursive construction allows us to derive the size of the codebooks. Let $\mathbf{V}_{(Ci,jC)}$ be an all-one m-dimensional row vector $(m = |C_5^0|)$ under constraint (Ci, jC). Let $\mathbf{c}_k^s$ be a k-bit codeword with last five consecutive bits $(c_{k-4}c_{k-3}c_{k-2}c_{k-1}c_k) \in C_5^s$

Table 3.5: Largest 5-bit codebook(s) under constraint (Ci, jC).

Constraint	C_5^0	C_5^1
$(C5, 3C)$	{0, 1, 2, 3, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16, 17, 18, 19, 24, 25, 26, 27, 28, 30, 31}	{0, 1, 3, 4, 5, 6, 7, 12, 13, 14, 15, 16, 17, 19, 20, 21, 22, 23, 24, 25, 28, 29, 30, 31}
$(C4, 2C)$	{0, 1, 3, 6, 7, 12, 14, 15, 16, 17, 19, 24, 25, 28, 30, 31}	
$(C3, 1C)$	{0, 3, 14, 15, 24, 30, 31}	{0, 1, 7, 16, 17, 28, 31}
$(C2, 1C)$	{0, 3, 15, 24, 30, 31}	{0, 1, 7, 16, 28, 31}

for $s = 0$ or 1. If a 0 or 1 can be appended to $\mathbf{c}_k^s$ to form a $(k+1)$-bit codeword whose last five bits $(c_{k-3}c_{k-2}c_{k-1}c_kc_{k+1}) \in C_5^{1-s}$, such an expansion is called a valid expansion. Otherwise, it is called an invalid expansion. An expansion matrix is denoted as a $m \times m$ matrix $\mathbf{D}^s_{(Ci,jC)}$, where $\mathbf{D}^s_{(Ci,jC)}(i,j) = 0$ denotes an invalid expansion and $\mathbf{D}^s_{(Ci,jC)}(i,j) = 1$ a valid expansion from the i-th codeword in C_5^s to the j-th codeword in C_5^{1-s} under constraint (Ci, jC). Each row of $\mathbf{D}^s_{(Ci,jC)}$ has at most two ones, since each k-bit codeword can be appended to form at most two $(k+1)$-bit codewords whose last five bits satisfy the appropriate constraints. Let $\mathbf{Y}$ be an $m \times m$ anti-diagonal matrix with all ones. Due to symmetry between C_5^0 and C_5^1, $\mathbf{D}^0$ and $\mathbf{D}^1$ satisfy $\mathbf{D}^1_{(Ci,jC)} = \mathbf{Y}\mathbf{D}^0_{(Ci,jC)}\mathbf{Y}$. Define $\mathbf{D}_{(Ci,jC)} = \mathbf{D}^0_{(Ci,jC)}\mathbf{Y} = \mathbf{Y}\mathbf{D}^1_{(Ci,jC)}$. We denote $\mathbf{V}_{(Ci,jC)}$ and $\mathbf{D}_{(Ci,jC)}$ as $\mathbf{V}$ and $\mathbf{D}$, respectively, when there is no ambiguity about the constraint. For example, the expansion matrices corresponding to

constraints $(C3, 1C)$, $(C4, 2C)$, and $(C5, 3C)$ are given by

$$\mathbf{D}_{(C3,1C)} = \begin{bmatrix} 0&0&0&0&0&1&1 \\ 0&0&0&0&1&0&0 \\ 0&1&0&0&0&0&0 \\ 1&0&0&0&0&0&0 \\ 0&0&1&1&0&0&0 \\ 0&1&0&0&0&0&0 \\ 1&0&0&0&0&0&0 \end{bmatrix}, \mathbf{D}_{(C4,2C)} = \begin{bmatrix} 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0 \\ 0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \end{bmatrix},$$

$$\mathbf{D}_{(C5,3C)} = \begin{bmatrix} 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \end{bmatrix}.$$

Then, for $n \geq 5$, the number of codewords in an n-bit bus is equal to counting the valid transitions and is given by

$$\begin{aligned} |C(n)| &= \mathbf{V}\mathbf{D^0}\mathbf{D^1}\cdots\mathbf{V}^T \\ &= \begin{cases} \mathbf{V}(\mathbf{D}^0\mathbf{Y}\mathbf{Y}\mathbf{D}^1)^{\frac{n-5}{2}}\mathbf{V}^T & \text{if } n \text{ is odd;} \\ \mathbf{V}(\mathbf{D}^0\mathbf{Y}\mathbf{Y}\mathbf{D}^1)^{\frac{n-6}{2}}\mathbf{D}^0\mathbf{Y}\mathbf{Y}\mathbf{V}^T & \text{if } n \text{ is even;} \end{cases} \\ &= \mathbf{V}\mathbf{D}^{n-5}\mathbf{Y}\mathbf{V}^T. \end{aligned} \tag{3.3}$$

In the following, we first focus on constraints $(C3, 1C)$, $(C4, 2C)$, and $(C5, 3C)$. The codes based on these constraints are shown to have the same codebooks as OLCs, FPCs, and FOCs, respectively. Then, we consider constraint $(C2, 1C)$, which

would lead to codes with a smaller delay at the expense of a lower code rate. Several lemmas and theorems about the aforementioned codebooks and their sizes have been established below. All the proofs are straightforward, and hence omitted for conciseness. See the extended manuscript [53] of this work for more details.

3.3.3 Codes Under $(C3, 1C)$

The one Lambda codes have a worst-case delay $(1 + \lambda)\tau$. According to [32], the worst-case delay $(1 + \lambda)\tau$ can only be achieved **if and only if** the transitions $\uparrow\downarrow \times$, -$\uparrow$-, and $\uparrow$-$\uparrow$ plus their symmetric and complement versions (e.g. $\uparrow\downarrow \times$ and $\times \downarrow\uparrow$ are symmetric, and -$\downarrow$- is the complement of -$\uparrow$-) are avoided, where $\uparrow$, $\downarrow$, $\times$, and - denote 0$\rightarrow$1, 1$\rightarrow$0, don't care, and no transition, respectively. The first constraint of avoiding $\uparrow\downarrow \times$ ensures that a transition between any two codewords does not cause opposite transition on any wire. This condition is referred as a forbidden-transition (FT) condition. The second constraint of avoiding -$\uparrow$- ensures that 2C patterns are removed. This constraint ensures two adjacent bit boundaries cannot both be 01-type or 10-type, and is referred as a forbidden adjacent boundary pattern (FABP) condition [32]. The last two forbidden patterns give the constraint that no patterns 010 and 101 appear in the codeword, which is referred as a forbidden-pattern (FP) condition [32]. Codes satisfying these **necessary and sufficient** conditions are called one Lambda codes (OLCs). We denote the largest OLC codebook size for an n-bit bus as G_n, and G_n is given by

$$G_n = G_{n-1} + G_{n-5} \tag{3.4}$$

with initial conditions $G_1 = 2, G_2 = 3, G_3 = 4, G_4 = 5$, and $G_5 = 7$ [54].

With our classification, we explore codes under constraint $(C3, 1C)$. From Tab. 3.5, the two largest 5-bit codebooks are given by C_5^0={0, 3, 14, 15, 24, 30, 31} and C_5^1={0, 1, 7, 16, 17, 28, 31}. An n-bit codebook $C(n)$ can be obtained via Alg. 1. The number of codewords is given by

$$|C(n)| = \mathbf{V}\mathbf{D}_{(C3,1C)}^{n-5}\mathbf{V}^T \text{ for } n \geq 5, \tag{3.5}$$

where $\mathbf{V}$ is a seven-dimensional all one vector and $\mathbf{D}_{(C3,1C)}$ is a 7×7 expansion matrix. We further establish that the largest codebook sizes under constraint $(C3, 1C)$ satisfy the recursion:

Lemma 3.3.1. *For $n \geq 8$, $|C_{(C3,1C)}(n)|$ is given by a recursion $|C_{(C3,1C)}(n)| = |C_{(C3,1C)}(n-2)| + |C_{(C3,1C)}(n-3)|$, with initial conditions $|C_{(C3,1C)}(n)|$ =7, 9, 12, for n =5, 6, 7, respectively.*

In fact, we can further relate these codes with OLCs by the following:

Theorem 3.3.1. *The codes under $(C3, 1C)$ have the same codebooks as OLCs. Hence, $G_n = |C_{(C3,1C)}(n)|$.*

Theorem 3.3.1 implies that the codes under constraint $(C3, 1C)$ are equivalent to the class of OLC codes.

3.3.4 Codes Under $(C4, 2C)$

The $(1+2\lambda)$ codes have a worst-case delay of $(1+2\lambda)\tau$. No necessary and sufficient condition is known for a code to be a $(1 + 2\lambda)$ code. Two sufficient conditions FT

and FP are found, which lead to two families of $(1 + 2\lambda)$ codes, FTC and FPC, respectively. The size of an FTC codebook for an n-wire bus is given by F_{n+2}, where F_n is the Fibonacci sequence that satisfies $F_{n+2} = F_{n+1} + F_n$ and has initial conditions $F_1 = F_2 = 1$ [10]. The FPCs for an n-wire bus have a larger codebook size $2F_{n+1}$ [8].

With our classification, we explore codes under constraint $(C4, 2C)$. From Tab. 3.5, only one largest 5-bit codebook is found C_5^0={0, 1, 3, 6, 7, 12, 14, 15, 16, 17, 19, 24, 25, 28, 30, 31}. An n-bit codebook $C(n)$ can be obtained via Alg. 1 by setting $C_5^1 = C_5^0$. The number of codewords is given by

$$|C(n)| = \mathbf{V}\mathbf{D}_{(C4,2C)}^{n-5}\mathbf{V}^T \text{ for } n \geq 5 \tag{3.6}$$

where $\mathbf{V}$ is a 16-dimensional all one vector and $\mathbf{D}_{(C4,2C)}$ is a 16×16 expansion matrix. We further establish that the largest codebook sizes under constraint $(C4, 2C)$ satisfy the recursion:

Lemma 3.3.2. *For $n \geq 9$, $|C_{(C4,2C)}(n)|$ can be simplified as recursion $|C_{(C4,2C)}(n)| = 2|C_{(C4,2C)}(n-1)| - |C_{(C4,2C)}(n-2)| + |C_{(C4,2C)}(n-4)|$, with boundary conditions $|C_{(C4,2C)}(n)|$ =16, 26, 42, 68, for n =5, 6, 7, 8, respectively.*

Again, we can relate these codes to existing CACs by the following:

Theorem 3.3.2. *The codes under $(C4, 2C)$ have the same codebooks as FPCs. Hence, $2F_{n+1} = |C_{(C4,2C)}(n)|$.*

Since FPCs and our codes under $(C4, 2C)$ can be obtained by excluding $D3$ plus $D4$ patterns and $C5$ plus $C6$ patterns, respectively, Theorem 3.3.2 is not surprising

given that $C5$ and $C6$ are the same as $D3$ and $D4$, respectively. Theorem 3.3.2 implies that results in the literature regarding FPCs are also applicable to codes under constraint $(C4, 2C)$.

3.3.5 Codes Under $(C5, 3C)$

The $(1+3\lambda)$ codes have a worst-case delay of $(1+3\lambda)\tau$, which can be achieved **if and only if** $\downarrow\uparrow\downarrow$ and $\uparrow\downarrow\uparrow$ are avoided. So the **necessary and sufficient** condition for the $(1+3\lambda)$ codes is that the codebook cannot have both 010 and 101 appearing centered around any bit position, which is referred as a forbidden-overlap (FO) condition. Codes satisfying the FO condition are called FOCs. It is shown that the largest FOC codebook for an n-bit bus is given by T_{n+2}, where $T_n = T_{n-1} + T_{n-2} + T_{n-3}$ is the tribonacci number sequence with initial conditions $T_1 = 1$, $T_2 = 1$, and $T_3 = 2$ [32].

With our classification, we explore codes under constraint $(C5, 3C)$. Two largest 5-bit codebooks C_5^0={0, 1, 2, 3, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16, 17, 18, 19, 24, 25, 26, 27, 28, 30, 31} and C_5^1={0, 1, 3, 4, 5, 6, 7, 12, 13, 14, 15, 16, 17, 19, 20, 21, 22, 23, 24, 25, 28, 29, 30, 31} are found. Via Alg. 1, an n-bit codebook $C(n)$ can be obtained. The number of codewords is given by

$$|C(n)| = \mathbf{V}\mathbf{D}_{(C5,3C)}^{n-5}\mathbf{V}^T \text{ for } n \geq 5, \tag{3.7}$$

where $\mathbf{V}$ is a 24-dimensional all one vector and $\mathbf{D}_{(C5,3C)}$ is a 24×24 expansion matrix.

We further establish that the largest codebook sizes under constraint $(C5, 3C)$

satisfy the recursion:

Lemma 3.3.3. *For $n \geq 8$, $|C_{(C5,3C)}(n)|$ can be simplified as recursion $|C_{(C5,3C)}(n)| = |C_{(C5,3C)}(n-1)| - |C_{(C5,3C)}(n-2)| + |C_{(C5,3C)}(n-3)|$, with boundary conditions $|C_{(C5,3C)}(n)|$ =24,44,81, for n =5, 6, 7, respectively.*

Again we can relate these codes to existing CACs by the following:

Theorem 3.3.3. *The codes under $(C5, 3C)$ have the same codebooks as FOCs. Hence, $T_{n+2} = |C_{(C5,3C)}(n)|$.*

Theorem 3.3.3 is not surprising, since FOCs and our codes under $(C5, 3C)$ can be obtained by excluding $D4$ and $C6$ patterns, respectively, and $D4$ and $C6$ have been shown to be the same. Theorem 3.3.3 implies that results in the literature regarding FOCs are also applicable to codes under constraint $(C5, 3C)$.

3.3.6 Codes Under $(C2, 1C)$

With our classification, we explore codes under constraint $(C2, 1C)$. From Tab. 3.5, the two largest 5-bit codebooks are given by C_5^0={00000, 00011, 01111, 11000, 11110, 11111} and C_5^1={00000, 00001, 00111, 10000, 11100, 11111}. An n-bit codebook $C(n)$ can be obtained via Alg. 1. The number of codewords is given by

$$|C(n)| = \mathbf{V}\mathbf{D}^{n-5}\mathbf{V}^T \text{ for } n \geq 5, \tag{3.8}$$

where $\mathbf{V}$ is a six-dimensional all one vector and $\mathbf{D} = \begin{bmatrix} 0&0&0&0&1&1 \\ 0&0&0&1&0&0 \\ 1&0&0&0&0&0 \\ 0&0&1&0&0&0 \\ 0&1&0&0&0&0 \\ 1&0&0&0&0&0 \end{bmatrix}$.

We further establish that the largest codebook sizes under constraint $(C2, 1C)$ satisfy the recursion:

Lemma 3.3.4. *For $n \geq 10$, $|C_{(C3,1C)}(n)|$ can be simplified as recursion $|C_{(C2,1C)}(n)| = |C_{(C2,1C)}(n-2)| + |C_{(C2,1C)}(n-5)|$, with initial conditions $|C_{(C2,1C)}(n)|$ =6, 7, 9, 11, 14, for n =5, 6, 7, 8, 9, respectively.*

Lemma 3.3.5. *The codebook under $(C2, 1C)$ is a subset of OLC.*

3.3.7 Pruned Codes Under $(C2, 1C)$

For $(C2, 1C)$, the restriction on the side wires is more relaxed than that on the middle wires, which results in larger worst-case delays for the side wires. Hence, we prune the CACs under constraint $(C2, 1C)$ by removing codewords with larger delays on the side wires in order to achieve a smaller worst-case delay. Since the pruned codes have a smaller delay than OLCs, we call these pruned CACs improved one Lambda codes (IOLCs). We obtain IOLCs by first finding an n-bit codebook via Alg. 1 as in Sec. 3.3.6, and then pruning the codebook with Alg. 2. To prune the codebook $C(n)$, we search for maximum subsets of C_5^i $(i = 0, 1)$ with smaller delays on the side wires. For C_5^0, two maximum subsets $C_5^{0,0}$={0, 3, 15, 30, 31} and $C_5^{0,1}$={0, 15, 24, 30, 31} are found with smaller worst-case delays on wires 1 and 2 and wires 4 and 5, respectively. For C_5^1, a maximum subset $C_5^{1,1}$={0, 1, 7, 16, 31} is found with smaller worst-case delays on wires 4 and 5. Finally, a valid n-bit codebook is obtained with the leftmost five bits belonging to $C_5^{0,0}$, and the rightmost five bits belonging to $C_5^{0,1}$ or $C_5^{1,1}$ depending on whether n is odd or even.

The pruning algorithm for CACs under $(C2, 1C)$ on an n-bit bus is shown in Alg. 2. By pruning all codewords $\mathbf{c}_n$ in $C(n)$, the algorithm removes codewords with larger delay on side wires. With Alg. 2, we get an n-bit IOLC under constraint

Algorithm 2 Pruning CACs under $(C2, 1C)$

Input: $C_5^{0,0}$, $C_5^{0,1}$, $C_5^{1,1}$, $C(n)$;
  if n is odd **then**
    $i = 1$;
  else
    $i = 0$;
  end if
  for $\forall \mathbf{c}_n = (c_1 c_2 \cdots c_n) \in C(n)$ **do**
    if $(c_1 c_2 c_3 c_4 c_5) \notin C_5^{0,0}$ or $(c_{n-4} c_{n-3} c_{n-2} c_{n-1} c_n) \notin C_5^{1-i,1}$ **then**
      eliminate $\mathbf{c}_n$ from $C(n)$;
    end if
  end for
  Output: $C(n)$.

$(C2, 1C)$, and its size is given by

$$|C_{IOLC}(n)| = \mathbf{W}_1 \mathbf{D}^{n-5} \mathbf{Y} \mathbf{W}_2^T \text{ for } n \geq 5, \tag{3.9}$$

where $\mathbf{W}_1 = [1\ 1\ 1\ 0\ 1\ 1]$, $\mathbf{W}_2 = [1\ 0\ 1\ 1\ 1\ 1]$, and $\mathbf{D}$ is the same as that in Eq. (3.8). Note that $\mathbf{W}_1$ and $\mathbf{W}_2$ are used instead of $\mathbf{V}$, because of the pruning of valid patterns on side wires.

We further establish that the largest codebook sizes of IOLCs satisfy the recursion:

Lemma 3.3.6. *For $n \geq 10$, $|C_{IOLC}(n)|$ can be simplified as recursion $|C_{IOLC}(n)| = |C_{IOLC}(n-2)| + |C_{IOLC}(n-5)|$, with initial conditions $|C_{IOLC}(n)|$ =4, 5, 7, 8, 11, for n =5, 6, 7, 8, 9, respectively.*

Lemma 3.3.7. *The IOLC codebook is a subset of OLC.*

3.4 Performance Evaluation

In this section, we evaluate the performance of CACs based on our classification with extensive simulations, and compare them with existing CACs. Each CAC has two key performance metrics: delay and rate. The delay of a CAC is the worst-case delay when the codewords from the CAC are transmitted over the bus. Codebook size and code rate are often used to measure the overhead of CACs. The codebook size of a CAC is simply the number of codewords. Suppose a CAC of size M is transmitted over an n-bit bus, then its rate is defined as $\frac{\lfloor \log_2 M \rfloor}{n}$. A CAC of rate k/n implies that $n-k$ extra wires are used in addition to k data wires so as to reduce the crosstalk delay. Hence, the code rate measures the area and power overhead of CACs: the higher the rate, the smaller the overhead. Obviously, there is a tradeoff between the code rate and delay of a CAC: typically a lower rate code is needed to achieve a smaller delay. To measure the overall effects of both rate and delay, we also define the throughput of a CAC as the ratio of code rate and delay. The assumptions for this definition are: (1) the clock rate of the bus is determined by the inverse of the worst-case delay; (2) the throughput of the bus is linearly proportional to k, the number of data wires.

Since codes under $(C3, 1C)$, $(C4, 2C)$, and $(C5, 3C)$ have exactly the same codebooks as OLCs, FPCs, and FOCs, their delay, rate, and throughput are also the same. Under constraint $(C2, 1C)$, we propose two kinds of codes, unpruned codes and pruned codes (IOLCs). In the following, we compare their performance with OLCs in [9] with extensive simulations.

To compare the worst-case delay of our IOLCs, unpruned $(C2, 1C)$ codes, and OLCs, we simulate two buses, a 10-bit bus and a 16-bit bus, with all transitions

between any two codewords in their codebooks and obtain the worst-case delays of each wire. The simulation environment has been explained in Sec. 3.2.3. Both buses have a length of 5mm, and $\tau_0 = 1.42$ps and $\lambda = 12.24$. For a 10-bit bus, the worst-case delays of our IOLC, unpruned $(C2, 1C)$ code, and an OLC are given by 10.14ps, 13.50ps, and 14.84ps, respectively. The worst-case delay of our IOLC and unpruned $(C2, 1C)$ code are 31.67% and 9.03% smaller than that of the OLC, respectively. For a 16-bit bus, the worst-case delays of our IOLC, unpruned $(C2, 1C)$ code, and an OLC are given by 10.40ps, 13.92ps, and 16.11ps, respectively. The worst-case delay of our IOLC and unpruned $(C2, 1C)$ code are 35.44% and 13.59% smaller than that of the OLC, respectively. See the extended manuscript [53] of this work for additional information.

For all simulations, our IOLCs have better delay performance than OLCs. Although both IOLCs and unpruned $(C2, 1C)$ codes have almost the same code rate and better delay performance than OLCs, the delay performance of IOLCs is much better than the unpruned $(C2, 1C)$ codes. With a more advanced technology where the coupling effect is significant, the improvement of our IOLCs is bigger.

The comparisons of the codebook size between our IOLCs, unpruned $(C2, 1C)$ codes, and OLCs [9] and the throughput gain with respect to OLCs are shown in Tab. 3.6. The throughput gain of our CACs with respect to OLCs is given by the ratio between the throughput of our CACs and the throughput of OLCs. The codebook sizes of the three codes are close. In all cases, the difference of the number of bits between our IOLCs and unpruned $(C2, 1C)$ codes is within 1 bit. The difference of the number of bits between our IOLCs and OLCs [9] is within 2 bits for $n \leq 16$. In respect to throughput, our IOLCs always have a greater throughput

than OLCs, and their throughput gain ranges from 1.02 to 1.55 for an n-wire bus ($5 \leq n \leq 16$). The unpruned $(C2, 1C)$ codes have better throughput in some cases than OLCs, and the throughput gain ranges from 0.78 to 1.10 for an n-wire bus ($5 \leq n \leq 16$). When unpruned $(C2, 1C)$ codes have a lower throughput than OLCs, IOLCs can be used.

Our IOLCs and unpruned $(C2, 1C)$ codes provide additional options for the tradeoff between code rate and code delay. In addition to achieving higher throughputs, the new CACs are also appropriate for interconnects where the delay is of top priority.

Table 3.6: Comparison of codebook size and throughput of IOLC, unpruned $(C2, 1C)$ code (UC), and OLC [9] ($\lambda = 12.24$ and $\tau_0 = 1.42$ps).

# of wires	# of words			# of bits			Throughput Gain	
	IOLC	UC	[9]	IOLC	UC	[9]	IOLC	UC
5	4	6	7	2	2	2	**1.55**	**1.10**
6	5	7	9	2	2	3	1.07	0.78
7	7	9	12	2	3	3	1.02	1.14
8	8	11	16	3	3	4	1.12	0.84
9	11	14	21	3	3	4	1.10	0.84
10	12	17	28	3	4	4	1.10	**1.10**
11	16	21	37	4	4	5	1.18	0.88
12	18	26	49	4	4	5	1.19	0.89
13	23	32	65	4	5	6	1.03	0.96
14	27	40	86	4	5	6	1.02	0.95
15	34	49	114	5	5	6	1.27	0.95
16	41	61	151	5	5	7	1.11	0.83

It has been shown that the encoding and decoding of OLCs, FPCs, and FOCs have quadratic complexity based on numeral systems [47]. Since codes under $(C3, 1C)$, $(C4, 2C)$, and $(C5, 3C)$ have exactly the same codebooks as OLCs, FPCs, and FOCs,

their CODECs also have quadratic complexity. Also, it is expected that the encoding and decoding of our IOLCs and unpruned $(C2, 1C)$ codes have a quadratic complexity, since the codebooks of our IOLCs and unpruned $(C2, 1C)$ codes are proper subsets of OLCs.

We remark that the simulation results in Sections 3.2.3 and 4.5 are all based on a 45nm CMOS technology. We have also run the same set of simulations based on a 0.1-μm technology (omitted for brevity). Between the two sets of simulation results, the main conclusions of the manuscript and the key features of our proposed classification and CACs remain the same. For instance, the delays of the patterns in different classes do not overlap, regardless of the technology. Also, the proposed CACs based on the new classification are also the same. This actually demonstrates that our approach to delay classification and CACs is applicable to a wide variety of technology. This is because in our approach, the dependency of the crosstalk delay on the technology is represented by the two parameters, the propagation delay τ_0 of a wire free of crosstalk and the coupling factor λ. Since our analytical approach to the classification and CACs treats these two parameters as variables, our approach can be easily adapted to a wide variety of technology.

3.5 SUMMARY

In this chapter, we propose a new classification of transition patterns. The new classification has finer classes and the delays do not overlap among different classes. Hence the new classification is conducive to the design of CACs. To illustrate this, we design a family of CACs with different constraints. Some codes of the family

are the same as existing codes, OLCs, FPCs, and FOCs. We also propose two new CACs with a smaller worst-case delay and better throughput than OLCs. Since our analytical approach to the classification and CACs treats the technology-dependent parameters as variables, our approach can be easily adapted to a wide variety of technology.

Chapter 4

Crosstalk avoidance codes for RLC On-Chip Interconnects

4.1 INTRODUCTION

Recent International Technology Roadmap of Semiconductors (ITRS) [1] has shown a troubling trend: while gate delay **decreases** with scaling, global wire delay **increases**. This is because with the process technologies scaling down, the crosstalk delay becomes more prominent due to the increasing capacitive and inductive couplings among all wires. At low clock frequency, the inductive coupling can be ignored and only the capacitive coupling determines the propagation delays. Many approaches (see, e.g. [8, 10, 32, 33, 44, 55–57]) have been proposed to alleviate the capacitive coupling. As the clock frequency approaches multi-gigahertz, the parasitic inductance of on-chip interconnects has become significant and its detrimental effects, including increased delay, voltage overshoots and undershoots, and increased

crosstalk noise [58–60], cannot be ignored. Hence, when the process technologies scaling down into deep submicrometer (DSM) and the clock frequency approaching multi-gigahertz range, the crosstalk delay and noise due to the capacitive and inductive coupling become the performance bottleneck in many high-performance VLSI designs, especially for global on-chip buses. It is imperative for designers to devise new techniques to address both capacitive and inductive couplings simultaneously.

Many approaches have been proposed to reduce the crosstalk delays due to the capacitive coupling, such as shielding, repeater insertion, and bus encoding [8, 10, 32, 33, 44, 55–57]. Among these approaches, the shielding scheme is the simplest one, but it requires a large area overhead. The repeater insertion scheme prevents simultaneously opposite switching between adjacent wires by introducing intentional time skewing. But it is hungry for power consumption. The bus encoding scheme, referred to as crosstalk avoidance coding (CAC) [8, 10, 32, 33, 44, 56, 57], is a promising technique for its effective delay reductions and low power consumptions compared with other techniques. Hence, in this work, we focus on this coding scheme for crosstalk reduction. However, the previously proposed CACs are based on distributed RC model and only consider neighboring two wires for crosstalk. When the inductance effect is significant, more neighboring wires should be considered for crosstalk due to the long-range effect of inductive coupling. It has been shown that the worst case switching pattern with the largest delay for the RLC-coupled interconnects is quite different from the RC-coupled interconnects [59, 60]. The growing inductive coupling renders the previously proposed approaches inefficient in delay reduction. In addition, signal noise like overshoots and undershoots are not accounted for by these previously proposed techniques. Hence, it is necessary to develop other coding

schemes to reduce the crosstalk delay and noise due to both capacitive and inductive couplings.

The inductive coupling is greatly dependent on the switching patterns on on-chip interconnects. It is important to find the patterns incurring larger delays and noises. In this chapter, we use "↑" to denote a transition from 0 to the supply voltage V_{dd} (normalized to 1), "-" no transition, and "↓" a transition from V_{dd} to 0. In [59], a worst case pattern considering capacitive and inductive coupling is given by ↑↓↑↓↑ (↑ and ↓ denote up and down transitions, respectively), where immediate neighbors switch oppositely and higher order neighbors switch in the same direction. In [60], the authors show that the worst case switching pattern would change from ↑↓↑↓↑ to ↑↑↑↑↑ when inductance coupling dominates. A bus invert scheme is also proposed to reduce the inductance effects by inverting the input data when the number of wires switching in the same direction is more than half of the number of wires [60]. Hence, patterns with more than half of wires switching in the same direction are eliminated.

The bus inver scheme in [60] is the first coding scheme in the literature to address the on-chip inductive coupling. However, there are two disadvantages of this scheme. First, the capacitive coupling is ignored for the crosstalk delay. The worst case pattern is only based on the largest inductive coupling, which increases linearly with length. However, the capacitive coupling is a quadratic function of length and cannot be ignored for long wires of global on-chip buses. Second, the classification of patterns for RLC modeled bus is too simple, since only one worst case pattern is considered for inductive coupling reduction. Other patterns with slightly less inductive coupling would compromise the coding scheme. For instance, for a 5-bit

pattern ↑↑↑↑- (- denotes no transition), the inductive coupling is also large for the middle wire and this pattern should also be avoided for better inductive coupling reduction.

Addressing these disadvantages for the scheme in [59,60], in this work we propose a new coding scheme. There are two main contributions in this chapter:

- First, we define a parameter to quantify the significance of inductive effects and propose a new classification of patterns based on a combined of two constraints for capacitive and inductive couplings, respectively.

- Second, we proposed new CACs based on our classification and design architectures of encoders and decoders (CODECs) based on a revised numeral system.

Our approach allows us to fine tune the patterns for different combination of capacitive and inductive couplings. Note that there are two extreme scenarios. If capacitive coupling dominates, our classification would reduce to the classification for RC-coupled interconnects [56]. If inductive coupling dominates, our classification would only consider inductance effects.

The rest of the chapter is organized as follows. In Section II, we first present adverse inductance effects and then define a parameter to quantify the significance of inductance effects. We then propose new CACs for RLC-coupled interconnects based on our classification of patterns in Section III and their CODEC designs based on a revised binary mixed-radix numeral system in Section IV. In Section 4.5, we compare their performance in terms of worst case delays and peak noises. Some concluding remarks are provided in Section 7.6.

4.2 CAPACITANCE AND INDUCTANCE EFFECTS

4.2.1 Interconnect Model

With the scaling of technologies and the clock frequency approaching multi-gigahertz, the inductance is becoming significant and impacts the signals on the bus greatly. Inductive coupling can cause adverse effects, such as crosstalk delay, signal overshoots and undershoots, and switching noise, which can lead to serious signal integrity issues [60]. In addition, the worst-case patterns due to the inductive coupling are quite different from those due to capacitive coupling [60], making previously proposed coding schemes ineffective. Hence, in today's high performance circuit design, the inductance effects cannot be neglected. A transition from an RC interconnect model to an RLC model is necessary.

A distributed RLC model of a five-wire bus is shown in Fig. 4.1, where $V_i(x,t)$ denotes the transient signal at time t and position x $(0 \leq x \leq L)$ over wire i for $i \in \{1,2,3,4,5\}$, r, l, and c denote the resistance, inductance, and capacitance per unit length, respectively. λ is the ratio of the coupling capacitance between two adjacent wires over the wire capacitance. $l_{i,j}$ denotes the coupling inductance per unit length between wires i and j. The values of λ and $l_{i,j}$ depend on many factors, such as the metal layer in which we route the bus, the wire width, the spacing between adjacent wires, and the distance to the ground layer. We consider a uniformly distributed bus with the same parameters r, l, c, and λ for all the wires.

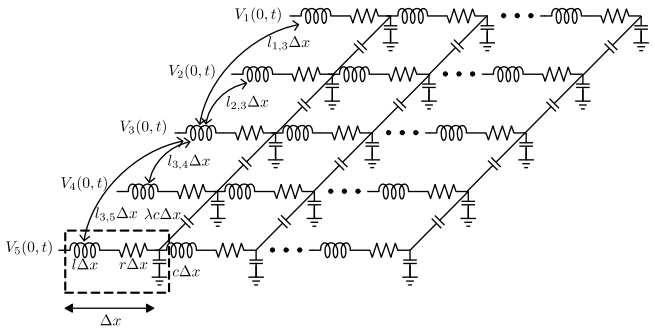


Figure 4.1: A distributed RLC model for five wires.

4.2.2 Crosstalk Delay

For a single line, many approaches have been proposed to analyze and characterize the delay and noise [38,61]. For a single distributed RC line, a closed-form expression of time delay is derived in [38] and given by $\tau = 0.693 R_{\text{tr}} cL + 0.377 rcL^2$, where R_{drv} is the driver resistance and L is the interconnect length. In [61], inductance is included to derive time delay of a single line in the following two scenarios:

- If $(R/Z_0) \leq \ln[4Z_0/(R_{\text{tr}} + Z_0)]$ AND $R_{\text{tr}} < 3Z_0$, the delay is $\tau == t_f = L\sqrt{lc}$, where $Z_0 = \sqrt{l/c}$ is the lossless characteristic impedance, $R = rL$ is the resistance of each wire, and $t_f = L\sqrt{lc}$ is the time of flight of the signals across the whole interconnects;

- If $(R/Z_0) \geq 2\ln[4Z_0/(R_{\text{tr}} + Z_0)]$ OR $R_{\text{tr}} > 3Z_0$, the time delay is $\tau = 0.693 R_{\text{tr}} cL + 0.377 rcL^2$, the same as that of a distributed RC line.

The first case occurs when the inductance becomes significant, since the two inequalities can be easily satisfied for large Z_0. In this case, the 50% time delay is approximated as the time flight $t_f = L\sqrt{lc}$. The second case is for small inductance effects and the delay is the same as that of an RC-modeled bus.

For a multi-wire bus, the on-chip RLC interconnects are characterized by telegrapher's equations given by [5]

$$\begin{cases} \frac{\partial}{\partial x}\mathbf{V}(x,t) = -\mathbf{L}\frac{\partial}{\partial t}\mathbf{I}(x,t) - \mathbf{R}\mathbf{I}(x,t), \\ \frac{\partial}{\partial x}\mathbf{I}(x,t) = -\mathbf{C}\frac{\partial}{\partial t}\mathbf{V}(x,t), \end{cases} \tag{4.1}$$

where $\mathbf{V}(x,t)$ and $\mathbf{I}(x,t)$ denote the voltage and current vectors of the interconnects, respectively, and $\mathbf{R} = [r_{i,j}]$, $\mathbf{L} = [l_{i,j}]$, and $\mathbf{C} = [c_{i,j}]$ are the resistance, inductance, and capacitance matrices, respectively. $\mathbf{R} = R\mathbf{I}$ is a diagonal matrix. Since only wire capacitance $c_{i,i}$ and coupling capacitance between adjacent wires $c_{i,i+1}$ are considered, we have $c_{i,j} = 0$ for $i \neq i-1, i, i+1$. Hence, $\mathbf{C} = [c_{i,j}]$ is a tri-diagonal matrix. $\mathbf{L}$ is a dense matrix, since inductance effect is long-rang effect. Eq. (4.1) can be simplified as

$$\frac{\partial^2}{\partial x^2}\mathbf{V}(x,t) = \mathbf{LC}\frac{\partial^2}{\partial t^2}\mathbf{V}(x,t) + \mathbf{RC}\frac{\partial}{\partial t}\mathbf{V}(x,t). \tag{4.2}$$

It is known that the PDEs for a distributed RC interconnect can be decoupled to isolated equations by diagonalizing the coupling matrix $\mathbf{C}$ [5,38]. It has been shown that capacitance and inductance matrices of a bus with ideal return path satisfy [5]

$$\mathbf{LC} = \frac{1}{\nu^2}[I],$$

where ν is the speed of an electromagnetic wave in a given dielectric material and $[I]$ is the identify matrix. Hence, Eq.(4.1) is simplified as

$$\frac{\partial^2}{\partial x^2}\mathbf{V}(x,t) = \frac{1}{\nu^2}\frac{\partial^2}{\partial t^2}\mathbf{V}(x,t) + \mathbf{RC}\frac{\partial}{\partial t}\mathbf{V}(x,t). \tag{4.3}$$

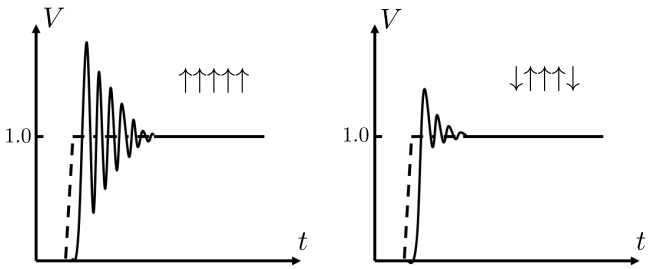


Figure 4.2: Ringing on wire 3 of a five-wire bus for ↑↑↑↑↑ and ↓↑↑↑↓.

Then, Eq. (4.3) can be decoupled using the same technique in [5, 38] for a distributed RC interconnect. The conclusion for single wire can be used for estimating the 50% delays and noises of all wires in a multi-wire bus. Since the product of $\mathbf{LC} = \frac{1}{\nu^2}[I]$ is a constant matrix, patterns with larger capacitive couplings have smaller inductive couplings, and vice versa. This has been verified in [60] by finding the best and worst case patterns considering inductive couplings. The worst pattern with the largest ring has all wires switching simultaneously in the same direction, and the worst pattern with the largest delay has immediate neighbors switching oppositely [60].

4.2.3 Interconnect Ring

Another adverse inductance effect is severe ringing of on-chip interconnect with growing inductance. The ringing is more severe for patterns with many wires switching in the same direction due to larger inductive couplings. The time delay has been approximated as the time of flight when inductance is significant [61]. However, this is **only true** when overshoots or undershoots are not crossing 50% V_{dd} multiple times. When the inductance is significant, the ring decays slowly and multiple undershoots (overshoots) may go below (above) 50% V_{dd} for a rising (falling) step

signal. Glitches would appear at the receiver end. A larger delay is required to get a stable result. In this case, the 50% delay is obtained based on the last crossing of 50% V_{dd}.

In the following, we show that the significance of ringing depends on the transition activity. Since the mutual inductance decays slowly, the inductance effect is long-range effect. All high order neighbors would contribute to the crosstalk. For this reason, we include two more wires and focus on wire 3 of a 5-wire bus in Fig. 4.1. The total capacitance C_t and inductance L_t of wire 3 satisfy $C_t L_t = \frac{1}{\nu^2}$. For transition $\uparrow\uparrow\uparrow\uparrow\uparrow$, C_t gets its smallest value, since there is no capacitive coupling. However, L_t gets its maximum value and the inductive coupling is significant. The resulted ring decays slowly as shown in Fig. 4.2. Similarly, for transition $\downarrow\uparrow\uparrow\uparrow\downarrow$, C_t increases and L_t decreases. The inductive coupling decreases and the ring decays quickly as shown in Fig. 4.2.

The significance of ringing can also be explained by a parameter ζ introduced in [62], where a closed-form delay model is derived for a single RLC wire as a function of parameter ζ. The parameter ζ is given by

$$\zeta = \frac{R_t}{2\sqrt{L_t/C_t}} \cdot \frac{R_T + C_T + R_T C_T + 0.5}{\sqrt{1 + C_T}},$$

where $C_T = \frac{C_L}{C_t}$ and $R_T = \frac{R_{tr}}{R_t}$. For a small ζ, the ringing is significant and the 50% delay is large due to multiple crossing of 50% V_{dd} [62]. For a large ζ, the ringing is weak and the 50% delay is obtained based on the first crossing of 50% V_{dd}. For the two transition patterns, $\uparrow\uparrow\uparrow\uparrow\uparrow$ and $\downarrow\uparrow\uparrow\uparrow\downarrow$, the former has a smaller ζ than that of the latter, since $\uparrow\uparrow\uparrow\uparrow\uparrow$ has larger L_t and smaller C_t. Hence, the ringing of pattern

↑↑↑↑↑ is more significant as shown Fig. 4.2.

4.3 CAC design

4.3.1 Previous CAC Design

CACs are first proposed to reduce the crosstalk delay for on-chip global interconnects. A k-bit data word $(x_k x_{k-1} \cdots x_1)$ is encoded into an m-bit $(m > k)$ codeword $(c_m c_{m-1} \cdots c_1)$. Two kinds of CACs, CACs with memory and memoryless CACs, have been investigated in the literature [51]. CACs with memory need to store all codebooks corresponding to different codewords $(c_m c_{m-1} \cdots c_1)$, since the encoding depends on the data word $(x_k x_{k-1} \cdots x_1)$ as well as the preceding codeword. In contrast, memoryless CACs require a single codebook to generate codewords for transmission, because the encoding depends on the data word only. Hence, memoryless CACs are much simpler to implement than CACs with memory. We focus on memoryless CACs in this chapter.

There exist several methods to obtain a memoryless codebook based on pattern pruning, transition pruning, or recursive construction. The pattern pruning technique is quite straightforward, and gives a codebook with a smaller worst-case delay by eliminating some patterns. The transition pruning technique [10] is based on graph theory. This method first builds a transition graph with all possible codewords as nodes and all valid transitions as edges, and then finds a maximum clique. A clique is defined as a subgraph where every pair of nodes are connected with an edge. A maximum clique is defined as a clique of the largest possible size in a given graph. Since every pair of nodes is connected, a maximum clique in this graph

constitutes a memoryless codebook with the largest size. The codebook generation method is based on exhaustive search. Although it is easy to get a maximum clique from a transition graph with a small m, the complexity increases rapidly with m. This is because the number of edges in an m-bit transition graph is upper bounded by $2^{m-1}(2^m - 1)$, which increases exponentially with m. In fact, it is an NP problem to find a maximum clique for given constraints [52]. The recursive technique constructs an $(m+1)$-bit codebook from an m-bit codebook [8,9]. Since for a small m, a largest codebook can be obtained easily via the second method, a codebook for an m-wire bus can be constructed recursively.

Previously proposed CACs (see, for example, [8, 10, 32, 33]) are not efficient if the inductance effects are significant. Adverse inductance effects, such as voltage overshoots and undershoots, and switching noise, would change the worst case switching pattern and also lead to serious signal integrity issues [60]. Hence, other coding scheme is needed to account for these adverse effects. The key idea of previous CACs is to eliminate transition patterns incurring larger delays. To account for inductance effects, we first find patterns with larger inductive couplings. Then, using a similar idea, we extend CACs to account for inductance effects by eliminating those patterns with larger inductive couplings.

4.3.2 Classification

In the following, we consider the classification of transition patterns with respect to total inductance of the middle wire (wire k). To quantify the inductive coupling,

we introduce a parameter

$$W_k = \left| \sum_{i=k-\lfloor \Delta/2 \rfloor}^{i=k+\lfloor \Delta/2 \rfloor} w_i \right| \quad \text{for wire } k, \tag{4.4}$$

where Δ is the number of neighboring wires considered for mutual inductance and $w_i = -1, 0, 1$ corresponds to $\downarrow$, $-$, and $\uparrow$ on wire i, respectively. Note that $w_i = 1$ or -1 denotes the largest inductive coupling. Since the mutual inductance decays slowly, more neighbors would contribute to the crosstalk. Instead of choosing two adjacent wires for capacitive coupling, we choose two more adjacent wires ($\Delta = 4$) for inductive coupling. The first reason of choosing $\Delta = 4$ is that the classification of transitions would be easy, since we have a reasonable number of transition patterns to classify. For instance for $\Delta = 4$, there are a total of $3^5 = 243$ transition patterns compared with $3^7 = 2187$ for $\Delta = 6$. The other reason is that our CAC design is based on a recursive coding scheme as explained in Sec. III-C, which would help to restrict inductive coupling on wires beyond the chosen neighboring wires.

We first focus on a five-wire bus for transition pattern classification. There are $3^4 = 81$ different transition patterns with $\uparrow$ transition on wire 3, which can be partitioned into 6 classes as shown in Table 4.1. For patterns with a $\downarrow$ transition on wire 3, a similar classification can be obtained by inverting all patterns in Table 4.1. For patterns with no transition on wire 3, a classification is shown in Table 4.2.

From Table 4.1, we note that the worst case pattern with respect to inductive coupling is $\uparrow\uparrow\uparrow\uparrow\uparrow$, which is the best case pattern in terms of capacitive coupling. By choosing those with $|W_3| \leq k_w$ in Tables 4.1 and 4.2 where $k_w = \{0, \cdots, 4\}$, we can reduce the worst case inductive couplings. The smaller k_w is, the larger reduction

Table 4.1: Classification of patterns with respect to $W_3 = |\sum_{i=1}^{5} w_i|$.

W_3	Patterns with ↑ on wire 3
0	↓-↑- -, -↓↑- -, - -↑↓-, - -↑-↓, ↑-↑↓↓, ↑↓↑-↓, ↑↓↑↓-, -↑↑↓↓ ↓↑↑-↓, ↓↑↑↓-, -↓↑↑↓, ↓-↑↑↓, ↓↓↑↑-, -↓↑↓↑, ↓-↑↓↑, ↓↓↑-↑
1	- -↑- -, ↑↓↑- -, ↑-↑↓-, ↑-↑-↓, ↓↑↑- -, -↑↑↓-, -↑↑-↓, ↓-↑↑- -↓↑↑-, - -↑↑↓, ↓-↑-↑, -↓↑-↑, - -↑↓↑, ↑↑↑↓↓, ↓↓↑↑↑, ↑↓↑↑↓ ↑↓↑↓↑, ↓↑↑↑↓, ↓↑↑↓↑, ↓↓↑- -, - -↑↓↓, ↓-↑-↓, ↓-↑↓-, -↓↑-↓ -↓↑↓-, ↑↓↑↓↓, ↓↑↑↓↓, ↓↓↑↑↓, ↓↓↑↓↑
2	↑-↑- -, -↑↑- -, - -↑↑-, - -↑-↑, ↑↑↑↓-, ↑↑↑-↓, ↑↓↑↑-, ↑-↑↑↓ ↑↓↑-↑, ↑-↑↓↑, ↓↑↑↑-, -↑↑↑↓, ↓↑↑-↑, -↑↑↓↑, ↓-↑↑↑, -↓↑↑↑ -↓↑↓↓, ↓-↑↓↓, ↓↓↑-↓, ↓↓↑↓-
3	↑↑↑- -, - -↑↑↑, ↑-↑-↑, ↑-↑↑-, -↑↑↑-, -↑↑-↑ ↓↑↑↑↑, ↑↓↑↑↑, ↑↑↑↓↑, ↑↑↑↑↓, ↓↓↑↓↓
4	↑↑↑↑-, ↑↑↑-↑, ↑-↑↑↑, -↑↑↑↑
5	↑↑↑↑↑

Table 4.2: Classification of patterns with respect to $W_3 = |\sum_{i=1}^{5} w_i|$.

W_3	Patterns with - on wire 3
0	- - - - -, ↑↓- - -, ↑- -↓-, ↑- - -↓, -↑-↓-, -↑- -↓, - - -↑↓, - - -↓↑ -↓-↑-, -↓- -↑, ↓↑- - -, ↓- -↑-, ↓- - -↑, ↑↑-↓↓, ↓↓-↑↑, ↑↓-↑↓ ↑↓-↓↑, ↓↑-↑↓, ↓↑-↓↑
1	↑- - - -, -↑- - -, - - -↑-, - - - -↑, ↓- - - -, -↓- - -, - - -↓-, - - - -↓ ↑↑-↓-, ↑↑- -↓, ↑- -↑↓, ↑- -↓↑, ↑↓-↑-, ↑↓- -↑, -↑-↑↓, -↑-↓↑ -↓-↑↑, ↓↑-↑-, ↓↑- -↑, ↓- -↑↑, ↑- -↓↓, ↑↓- -↓, ↑↓-↓-, -↑-↓↓ -↓-↑↓, -↓-↓↑, ↓↑- -↓, ↓↑-↓-, ↓- -↑↓, ↓- -↓↑, ↓↓-↑-, ↓↓- -↑
2	↑↑- - -, - - -↑↑, ↑- -↑-, ↑- - -↑, -↑- -↑, -↑-↑-, ↓↓- - -, - - -↓↓ ↓- -↓-, ↓- - -↓, -↓- -↓, -↓-↓-, ↑↑-↑↓, ↑↑-↓↑, ↑↓-↑↑, ↓↑-↑↑ ↓↓-↓↑, ↓↓-↑↓, ↓↑-↓↓, ↑↓-↓↓
3	↑↑-↑-, ↑↑- -↑, ↑- -↑↑, -↑-↑↑, ↓↓-↓-, ↓↓- -↓, ↓- -↓↓, -↓-↓↓
4	↑↑-↑↑, ↓↓-↓↓

of inductive coupling.

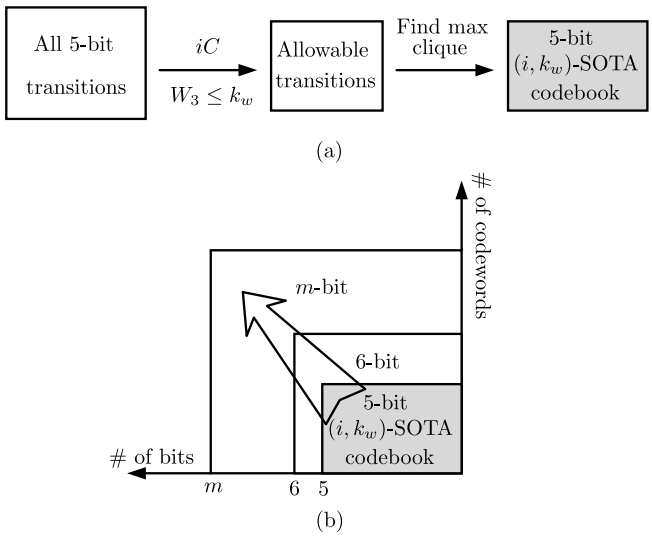


Figure 4.3: (a) Find a 5-bit (i, k_w)-SOTA codebook; (b) Construct an m-bit (i, k_w)-SOTA codebook recursively.

4.3.3 New CAC Design

New CACs accounting for both capacitive and inductive couplings are desired. Opposite transitions on adjacent wires lead to large capacitive couplings, and same transitions on adjacent wires lead to large inductive couplings. Hence, to reduce reliability issue as well as delay issue due to capacitive and inductive coupling effects, we need to avoid those patterns with most same and opposite switchings as shown in Tables 4.1 and 4.2. In the following, with consideration of both the capacitive and inductive couplings, we propose a **S**ame and **O**pposite **T**ransitions **A**voidance (**SOTA**) coding scheme. The reduction of the capacitive coupling is achieved by avoiding iC patterns in [42] for $i = \{1, 2, 3, 4\}$. The reduction of the inductive coupling is achieved by eliminating patterns in Tables 4.1 and 4.2 for $W_3 > k_w$ ($k_w = \{0, 1, 2, 3, 4\}$). Such codes are referred to as (i, k_w)-SOTA codes.

In this chapter, we use the recursive scheme to find an (i, k_w)-SOTA codebook for

an m-bit bus. First, we focus on wire 3 in a five-bit window and find a 5-bit (i, k_w)-SOTA codebook. The procedure is illustrated in Fig. 4.3(a) with two steps. The first step is to obtain all allowable transitions by applying the two constraints, iC and $W_3 \leq k_w$. This can be done by applying the two constraints sequentially. Here, we first pick all transitions satisfying $W_3 \leq k_w$ in Tables 4.1 and 4.2, and remove those having $(i+1)C, \cdots, 4C$ patterns. The second step is to find a maximum clique of nodes from the list of all allowable transitions. A 5-bit (i, k_w)-SOTA codebook is given by such a maximum clique of nodes. To obtain a 6-bit (i, k_w)-SOTA codebook, we obtain a 6-bit candidate codebook by appending 0's and 1's to the left of all 5-bit codewords. Then, we check if the left 5-bit pattern satisfies the two constraint and remove those 6-bit codewords violating any of the two constraints from the 6-bit candidate codebook. After doing all the appending and checking operations, we obtain a 6-bit (i, k_w)-SOTA codebook. Similarly, an m-bit (i, k_w)-SOTA codebook can be obtained recursively as shown in Fig. 4.3(b).

4.3.4 $(2, 1)$-SOTA codes

For a worst capacitive coupling $2C$ and a worst inductive coupling $W_3 \leq 1$, a list of allowable transitions in Fig. 4.3 can be obtained by removing $3C$ and $4C$ patterns in Tables 4.1 and 4.2. Using MATLAB, we find a maximum clique given by {00011, 00110, 00111, 01100, 01110, 10001, 10011, 11000, 11001, 11100}, which is a 5-bit $(2, 1)$-SOTA codebook. Let $C(m)$ be the set of m-bit $(2, 1)$-SOTA codewords and $\mathbf{c}(m) = c_m c_{m-1} \cdots c_1$ be a codeword in $C(m)$. An m-bit $(2, 1)$-SOTA codebook can be generated recursively in the following algorithm, where $\cdot$ denotes the concatenation operation.

Algorithm 3 $(2,1)$-SOTA codeword generation.

Input: $C(5) = \{00011,\ 00110,\ 00111,\ 01100,\ 01110,\ 10001,\ 10011,\ 11000,\ 11001,\ 11100\}$; $k = 5$;
  while $k \leq m-1$ **do**
    for $\forall \mathbf{c}(k) \in C(k)$ **do**
      if $c_k c_{k-1} c_{k-2} c_{k-3} = 0001$ **then**
        add $1 \cdot \mathbf{c}(k)$ to $C(k+1)$;
      else if $c_k c_{k-1} c_{k-2} c_{k-3} = 0011$ **then**
        add $0 \cdot \mathbf{c}(k)$ and $1 \cdot \mathbf{c}(k)$ to $C(k+1)$;
      else if $c_k c_{k-1} c_{k-2} c_{k-3} = 0110$ **then**
        add $0 \cdot \mathbf{c}(k)$ to $C(k+1)$;
      else if $c_k c_{k-1} c_{k-2} c_{k-3} = 0111$ **then**
        add $0 \cdot \mathbf{c}(k)$ to $C(k+1)$;
      else if $c_k c_{k-1} c_{k-2} c_{k-3} = 1000$ **then**
        add $1 \cdot \mathbf{c}(k)$ to $C(k+1)$;
      else if $c_k c_{k-1} c_{k-2} c_{k-3} = 1001$ **then**
        add $1 \cdot \mathbf{c}(k)$ to $C(k+1)$;
      else if $c_k c_{k-1} c_{k-2} c_{k-3} = 1100$ **then**
        add $0 \cdot \mathbf{c}(k)$ and $1 \cdot \mathbf{c}(k)$ to $C(k+1)$;
      else if $c_k c_{k-1} c_{k-2} c_{k-3} = 1110$ **then**
        add $0 \cdot \mathbf{c}(k)$ to $C(k+1)$;
      end if
    end for
    $k = k + 1$;
  end while
  Output: $C(m)$;

The m-bit codebook generated by Alg. 3 is a subset of m-bit FPC codebook. Hence, no 010 or 101 patterns are allowed in all codewords. The following lemma shows a necessary and sufficient condition for a 5-bit codebook to be a $(2,1)$-SOTA codebook.

Lemma 4.3.1. *An m-bit ($m \geq 5$) codebook is a $(2,1)$-SOTA codebook if and only if all m-bit codewords avoid 010, 101, 0000, and 1111 patterns.*

Proof. It is easy to see that Alg. 3 does not introduce 010, 101, 0000, and 1111 patterns. Hence, it is equivalent to prove that a 5-bit codebook is a $(2,1)$-SOTA codebook if and only if all 5-bit codewords avoid 010, 101, 0000, and 1111 patterns.

We first prove the necessity. A 5-bit $(2,1)$-SOTA codebook is given by {00011, 00110, 00111, 01100, 01110, 10001, 10011, 11000, 11001, 11100}. It is observed that no 010, 101, 0000, and 1111 patterns appear in any of these codewords.

To prove its sufficiency, we eliminate those codewords with 010, 101, 0000, and 1111 pattern from all 32 5-bit codewords. The refined codebook is given by {00011, 00110, 00111, 01100, 01110, 10001, 10011, 11000, 11001, 11100}, which is the same as the 5-bit $(2,1)$-SOTA codebook. □

Let C_m be the size of codebook $C(m)$. We further establish that the largest $(2,1)$-SOTA codebook size satisfies the recursion:

Lemma 4.3.2. *For $m \geq 8$, C_m is given by a recursion $C_m = C_{m-2} + C_{m-3}$, with initial conditions C_m = 10, 14, 18 for m =5, 6, 7, respectively.*

Proof. $\forall \mathbf{c}(m) \in C(m)$, define C_m^d as the number of codewords satisfying $c_m = c_{m-1}$ and $c_{m-1} \neq c_{m-2}$. Define C_m^t and C_m^f as the numbers of codewords satisfying

$c_m = c_{m-1} = c_{m-2}$ and $c_m \neq c_{m-1}$, respectively. Hence, $C_m = C_m^d + C_m^t + C_m^f$. For $m = 5$, we have $C_5^d = 4$, $C_5^t = 2$, $C_5^f = 4$, and $C_5 = 10$.

For $m > 5$, according to Alg. 3, we have

$$C_m^d = C_{m-1}^f,$$

$$C_m^t = C_{m-1}^d,$$

$$C_m^f = C_{m-1}^t + C_{m-1}^d,$$

$$C_m = C_{m-1}^t + 2C_{m-1}^d + C_{m-1}^f.$$

For $m = 6$, $C_6^d = 4$, $C_6^t = 4$, $C_6^f = 6$, and $C_6 = 14$. For $m = 7$, $C_7^d = 6$, $C_7^t = 4$, $C_7^f = 8$, and $C_7 = 18$.

Since $C_m = C_m^d + C_m^t + C_m^f$, we also have $C_{m-1} = C_{m-1}^d + C_{m-1}^t + C_{m-1}^f = C_m^d + C_m^f$. Hence, for $m \geq 8$,

$$\begin{aligned} C_m &= C_{m-1}^t + 2C_{m-1}^d + C_{m-1}^f \\ &= (C_{m-1}^d + C_{m-1}^f) + (C_{m-1}^t + C_{m-1}^d) \\ &= C_{m-2} + (C_{m-2}^d + C_{m-2}^f) \\ &= C_{m-2} + C_{m-3}. \end{aligned}$$

□

4.4 CODEC design

A numeral system is a mathematical notation for representing numbers of a given set by symbols in a consistent manner [63]. A binary mixed-radix numeral system represents a number as $\sum_{i=1}^{m} = d_i f_i$, where $(d_m, \cdots, d_2 d_1)$ is a binary string and $\{f_m, \cdots, f_2, f_1\}$ is a basis set of non-negative numbers. If any integer $u \in [0, \sum_{i=1}^{m} f_i]$ can be represented by at least one binary string $d_m \cdots d_2 d_1$, the numeral system is complete. In [47], a generic CAC encoding algorithm is proposed based on a binary mixed-radix numeral system. Since all-zero and all-one codewords are forbidden for RLC-coupled interconnects, the representable range starting from a nonzero value P is $u \in [P, P + \sum_{i=1}^{m} f_i]$. The revised encoding algorithm is shown in Alg. 4, where $\{f_i\}_{i=1}^{m}$ denotes the basis set of the encoding numeral system, v $(0 \leq v \leq \sum_{i=1}^{m} f_i)$ is data message, P is a non-zero integer, $\{\alpha_i\}_{i=1}^{m}$, $\{\alpha_i\}_{i=1}^{m}$, and Θ are some constants to be determined for different CACs, and $d_m d_{m-1} \cdots d_1$ is the encoded codeword. The data message v is first added by P. The decoding is straightforward by computing $\sum_{i=1}^{m} d_i f_i - P$. The CODEC based on Alg. 4 is shown in Fig. 4.4. The encoder has $m - 1$ same processing elements as shown in Fig. 4.4(c) and one additional adder for input v. One of the inputs to the top processing element denotes a don't care and is connected to the ground. Each processing element has two inputs, d_{k+1} and r_{k+1}, and two outputs, d_k and r_k, and is consisted of two comparators, one adder, one multiplexer, one AND, and one OR.

Algorithm 4 Generic CAC encoding algorithm.

Input: code length m; data message v; non-zero integer P;
  Initialize $v = v + P$;
  for $k = m$ downto 2 **do**
    if $k = m$ **then**
      if $v \geq \Theta$ **then**
        $d_m = 1$;
      else
        $d_m = 0$;
      end if
      $r_m = v - d_m \cdot f_m$;
    else
      if $r_{k+1} \geq \alpha_k$ **then**
        $d_k = 1$;
      else if $r_{k+1} < \beta_k$ **then**
        $d_k = 0$;
      else
        $d_k = d_{k+1}$;
      end if
      $r_k = r_{k+1} - d_k \cdot f_k$;
    end if
  end for
  $d_1 = r_2$;
  Output: $d_m d_{m-1} \cdots d_1$.

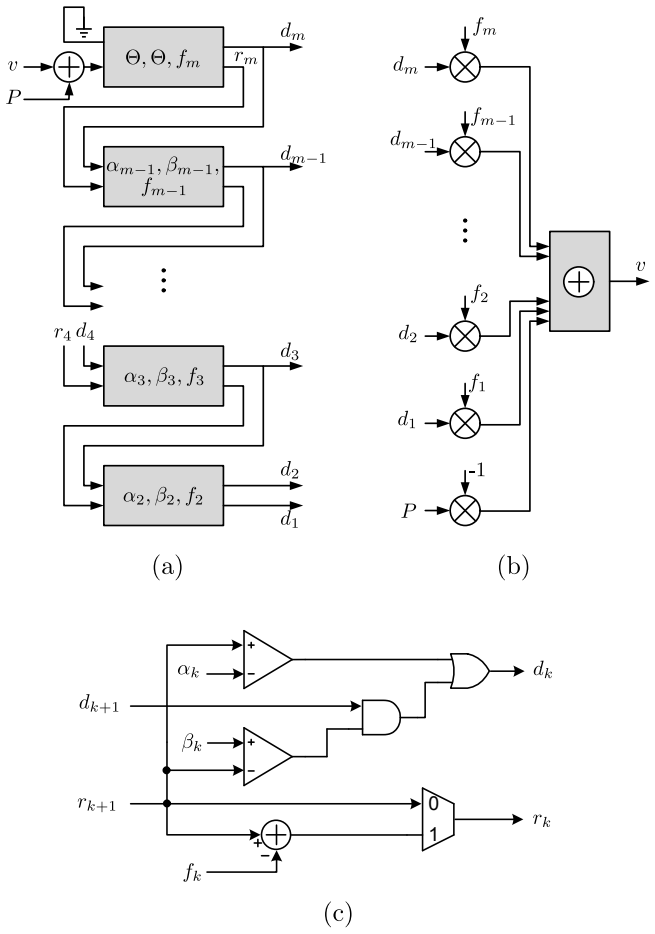


Figure 4.4: CODEC for an m-bit CAC via Alg. 4. (a) Encoder; (b) Decoder; (c) Processing element in (a).

4.4.1 $(2,1)$-SOTA codes

Assume the basis $\{f_i\}_{i=1}^m$ is positive and non decreasing. We define the min and max codewords of $(2,1)$-SOTA as

$$Min_n = \begin{cases} (00011)_k, & n = 5k, \\ (00011)_k \cdot 0, & n = 5k+1, \\ (00011)_k \cdot 00, & n = 5k+2, \\ (00011)_k \cdot 000, & n = 5k+3, \\ (00011)_k \cdot 0001, & n = 5k+4, \end{cases} \tag{4.5}$$

and

$$Max_n = \begin{cases} (11100)_k, & n = 5k, \\ (11100)_k \cdot 1, & n = 5k+1, \\ (11100)_k \cdot 11, & n = 5k+2, \\ (11100)_k \cdot 111, & n = 5k+3, \\ (11100)_k \cdot 1110, & n = 5k+4, \end{cases} , \tag{4.6}$$

where $n \leq m$ and $(00011)_k$ denotes k repetition of 00011. For $n = 1, 2, 3, 4$, let Min_n be 0, 00, 000, 0001, respectively, and Max_n be 1, 11, 111, 1110, respectively.

Define $g(\mathbf{c}_m) = \sum_{i=1}^m c_i f_i$ as the weight function based on a basis $\{f_i\}_{i=1}^m$. A basis $\{f_i\}_{i=1}^m$, $f_1 = 1$, $f_2 = 1$, $f_3 = 2$, $f_i = g(Max_{i-1}) - g(Min_{i-2}) - f_{i-1} + 1$ for $4 \leq i \leq m-1$, and $f_m = g(Max_{m-1}) - g(Min_{m-1}) + 1$, defines a complete system.

With the basis set $\{f_i\}_{i=1}^m$, the $(2,1)$-SOTA CODEC can be designed by choosing

$$\begin{gathered} \gamma_k = f_k, \Theta = g(Min_{m-1}) + f_m, \\ \alpha_k = g(Max_{k-1}) + 1, \beta_k = g(Min_{k-1}) + f_k. \end{gathered} \tag{4.7}$$

To show the correctness of the encoding algorithm in Alg. 4 for $(2,1)$-SOTA codes, we need following lemmas.

Lemma 4.4.1. *$f_n = f_{n-2} + f_{n-3}$ for $6 \leq n \leq m-1$.*

Proof. We prove this property by induction on n from 6 to $m-1$. For $n = 6$, $f_6 = g(Max_5) - g(Min_4) - f_5 + 1 = 5 = 3 + 2 = f_4 + f_3$. Suppose for $n \leq i$ $(i \geq 6)$, $f_n = f_{n-2} + f_{n-3}$. When $n = i+1$, $f_n = f_{i+1} = g(Max_i) - g(Min_{i-1}) - f_i + 1$. It is equivalent to prove that

$$g(Max_i) + 1 = g(Min_{i-1}) + f_i + f_{i-1} + f_{i-2}. \tag{4.8}$$

If $i = 5k+1$ $(k \geq 1)$, $Max_i = 111 \cdot (00111)_{k-1} \cdot 001$ and $Min_{i-1} = 000 \cdot (01100)_{k-1} \cdot 011$. Then, in Eq. (4.8), $LHS = f_i + f_{i-1} + f_{i-2} + \sum_{j=1}^{k-1}(f_{i-5j} + f_{i-5j-1} + f_{i-5j-2}) + f_1 + 1$. $RHS = f_i + f_{i-1} + f_{i-2} + \sum_{j=1}^{k-1}(f_{i-5j+1} + f_{i-5j}) + f_2 + f_1$. According to our assumption, $f_{i-5j+1} = f_{i-5j-1} + f_{i-5j-2}$. Since $f_2 = 1$, we have $LHS = RHS$.

If $i = 5k+2$ $(k \geq 1)$, $Max_i = 111 \cdot (00111)_{k-1} \cdot 0011$ and $Min_{i-1} = 000 \cdot (01100)_{k-1} \cdot 0110$. Then, in Eq. (4.8), $LHS = f_i + f_{i-1} + f_{i-2} + \sum_{j=1}^{k-1}(f_{i-5j} + f_{i-5j-1} + f_{i-5j-2}) + f_2 + f_1 + 1$. $RHS = f_i + f_{i-1} + f_{i-2} + \sum_{j=1}^{k-1}(f_{i-5j+1} + f_{i-5j}) + f_3 + f_2$. Since $f_{i-5j+1} = f_{i-5j-1} + f_{i-5j-2}$ and $f_3 = f_1 + 1 = 2$, $LHS = RHS$.

If $i = 5k+3$ $(k \geq 1)$, $Max_i = 111 \cdot (00111)_{k-1} \cdot 00111$ and $Min_{i-1} = 000 \cdot (01100)_{k-1} \cdot 01100$. Then, in Eq. (4.8), $LHS = f_i + f_{i-1} + f_{i-2} + \sum_{j=1}^{k-1}(f_{i-5j} + f_{i-5j-1} + f_{i-5j-2}) + f_3 + f_2 + f_1 + 1$. $RHS = f_i + f_{i-1} + f_{i-2} + \sum_{j=1}^{k-1}(f_{i-5j+1} + f_{i-5j}) + f_4 + f_3$. Since $f_{i-5j+1} = f_{i-5j-1} + f_{i-5j-2}$ and $f_4 = f_2 + f_1 + 1 = 3$, $LHS = RHS$.

If $i = 5k+4$ $(k \geq 1)$, $Max_i = 111 \cdot (00111)_{k-1} \cdot 001110$ and $Min_{i-1} = 000 \cdot (01100)_{k-1} \cdot 011000$. Then, in Eq. (4.8), $LHS = f_i + f_{i-1} + f_{i-2} + \sum_{j=1}^{k-1}(f_{i-5j} + f_{i-5j-1} +$

$f_{i-5j-2}) + f_4 + f_3 + f_2 + 1$. $RHS = f_i + f_{i-1} + f_{i-2} + \sum_{j=1}^{k-1}(f_{i-5j+1} + f_{i-5j}) + f_5 + f_4$. Since $f_{i-5j+1} = f_{i-5j-1} + f_{i-5j-2}$ and $f_5 = f_3 + f_2 + 1 = 6$, $LHS = RHS$.

If $i = 5k + 5$ $(k \geq 1)$, $Max_i = 111 \cdot (00111)_{k-1} \cdot 0011100$ and $Min_{i-1} = 000 \cdot (01100)_{k-1} \cdot 0110001$. Then, in Eq. (4.8), $LHS = f_i + f_{i-1} + f_{i-2} + \sum_{j=1}^{k-1}(f_{i-5j} + f_{i-5j-1} + f_{i-5j-2}) + f_5 + f_4 + f_3 + 1$. $RHS = f_i + f_{i-1} + f_{i-2} + \sum_{j=1}^{k-1}(f_{i-5j+1} + f_{i-5j}) + f_6 + f_5 + f_1$. Since $f_{i-5j+1} = f_{i-5j-1} + f_{i-5j-2}$ and $f_6 + f_1 = f_4 + f_3 + 1 = 6$, $LHS = RHS$. □

Lemma 4.4.2. *$\alpha_{n-1} = \beta_n$ for $3 \leq n \leq m - 1$.*

Proof. For $n = 3$, $\alpha_2 = g(Max_1) + 1 = 2$ and $\beta_3 = g(Min_2) + f_3 = 2$. We have $\alpha_2 = \beta_3$. For $n = 5k - 2$ $(k \geq 1)$, $Max_{n-2} = (00111)_{k-1} \cdot 001$ and $1 \cdot Min_{n-1} = (10001)_{k-1} \cdot 100$. $LHS = \sum_{j=1}^{k-1}(f_{n-5j+3} + f_{n-5j+2} + f_{n-5j+1}) + f_1$. $RHS = \sum_{j=1}^{k-1}(f_{n-5j+5} + f_{n-5j+1}) + f_3$. Since $f_{n-5j+5} = f_{n-5j+3} + f_{n-5j+2}$ and $f_3 = f_1 + 1$, $LHS = RHS$. For $n = 5k-1, 5k, 5k+1, 5k+2$ $(k \geq 1)$, the proof is similar to that in the proof of Lemma 4.4.1. Hence, for $3 \leq n \leq m - 1$, we have $\alpha_{n-1} = \beta_n$. □

Lemma 4.4.3. *$g(Max_n) = g(Max_{n-3}) + f_n + f_{n-1}$ for $4 \leq n \leq m - 1$.*

Proof. For $n = 4$, $LHS = g(1110) = 5$ and $RHS = g(1) + f_4 + f_3 = 5$. We have $LHS = RHS$. For $n = 5k-1$ $(k \geq 1)$, $Max_n = 11 \cdot (10011)_{k-1} \cdot 10$ and $11 \cdot Max_{n-3} = 11 \cdot (01110)_{k-1} \cdot 01$. $LHS = f_n + f_{n-1} + \sum_{j=1}^{k-1}(f_{n-5j+3} + f_{n-5j} + f_{n-5j-1}) + f_2$. $RHS = f_n + f_{n-1} + \sum_{j=1}^{k-1}(f_{n-5j+2} + f_{n-5j+1} + f_{n-5j}) + f_1$. Since $f_{n-5j+3} = f_{n-5j+1} + f_{n-5j}$, $f_{n-5j}f_{n-5j-1} = f_{n-5j+2}$, and $f_1 = f_2$, we have $LHS = RHS$. For $n = 5k, 5k + 1, 5k + 2, 5k + 3$ $(k \geq 1)$, the proof is similar to that in the proof of Lemma 4.4.1. Hence, for $4 \leq n \leq m - 1$, we have $g(Max_n) = g(Max_{n-3}) + f_n + f_{n-1}$. □

Lemma 4.4.4. *$g(Min_n) = g(Max_{n-4}) + 1$ for $5 \leq n \leq m$.*

Proof. For $n = 5$, $LHS = g(00011) = 2$ and $RHS = g(1) + 1 = 2$. We have $LHS = RHS$. For $n = 5k$ $(k \geq 1)$, $Min_n = 00 \cdot (01100)_{k-1} \cdot 011$ and $Max_{n-4} = 00 \cdot (00111)_{k-1} \cdot 001$. $LHS = \sum_{j=1}^{k-1}(f_{n-5j+2} + f_{n-5j+1}) + f_2 + f_1$. $RHS = \sum_{j=1}^{k-1}(f_{n-5j+1} + f_{n-5j} + f_{n-5j-1}) + f_1 + 1$. Since $f_{n-5j+2} = f_{n-5j} + f_{n-5j-1}$ and $f_2 = f_1 + 1$, we have $LHS = RHS$. For $n = 5k+1, 5k+2, 5k+3, 5k+4$ $(k \geq 1)$, the proof is similar to that in the proof of Lemma 4.4.1. Hence, for $5 \leq n \leq m$, we have $g(Min_n) = g(Max_{n-4}) + 1$. □

The following theorem shows the correctness of the encoding algorithm for $(2, 1)$-SOTA codes.

Theorem 4.4.1. *The output of encoding algorithm in Alg. 4 with constants specified in Eq. (4.7) is a* $(2, 1)$*-SOTA codebook.*

Proof. According to Lemma 4.3.1, the correctness of the encoding algorithm can be proved by showing that 010, 101, 0000, and 1111 are forbidden patterns.

If $d_k = 1$ and $d_{k-1} = 0$, we have $r_k < \beta_{k-1} = g(Min_{k-2}) + f_{k-1}$. Hence, $r_{k-1} = r_k < \beta_{k-1} = \alpha_{k-2}$ (Lemma 4.4.2), implying that $d_{k-2} = 0$. Hence, 101 is forbidden.

If $d_k = 0$ and $d_{k-1} = 1$, we have $r_k \geq \alpha_{k-1} = g(Max_{k-2}) + 1$. Hence, $r_{k-1} = r_k - f_{k-1} \geq \alpha_k - f_{k-1} = (g(Max_{k-2} + 1) - (g(Max_{k-2}) - g(Min_{k-3}) - f_{k-2} + 1) = g(Min_{k-3}) + f_{k-2} = \beta_{k-2}$, implying that $d_{k-2} = 1$. Hence, 010 is forbidden.

If $d_k = d_{k-1} = d_{k-2} = 0$, we have $r_{k-2} = r_{k-1} = r_k \geq g(Min_k) = g(Max_{k-4}) + 1 = \alpha_{k-3}$ (Lemma 4.4.4), implying that $d_{k-3} = 1$. Hence, 0000 pattern is forbidden.

If $d_k = d_{k-1} = d_{k-2} = 1$, we have $r_{k-2} \leq g(Max_k) - f_k - f_{k-1} - f_{k-2} = g(Max_k) - f_k - f_{k-1} - g(Max_{k-3}) + \beta_{k-3} - 1 = \beta_{k-3} - 1$ (Lemma 4.4.3), implying

that $d_{k-3} = 0$. Hence, 1111 pattern is forbidden. □

4.5 Performance

In this section, we evaluate the performance of our new CACs for RLC-coupled interconnects with respect to worst case delays, peak noises, and rates. The worst-case delay of a CAC is the largest delay among all wires when the codewords from the CAC are transmitted over the bus. The peak noise of a CAC the maximum of overshoots and undershoots, which are normalized to supply voltage V_{dd}. The code rate of a CAC over an m-bit bus is defined by $\frac{\lfloor \log_2 C_m \rfloor}{m}$, where C_m is the codebook size. The code rate measures the redundance of a CAC. A rate k/n implies that additional $n - k$ bits are needed for a k-bit data to reduce the crosstalk.

The codebook size and code rate of our (2,1)-SOTA codes are summarized in Table 4.3. For m-bit bus ($5 \leq m \leq 32$), the code rate ranges between 0.41 and 0.60. The best code rate 0.6 is achieved for $m = 5$. When m approaches inf, the asymptotic code rate is given by 0.406, the same as that of OLCs [8].

All the simulation results in this chapter are obtained from HSPICE based on a 45nm technology with 10 metal layers [49]. We focus on global buses in the top metal layer 10 with substrate as the ground. The bus parameters are obtained by structure 1 in [50]. All wires are uniformly distributed with a length $L = 5$ mm, width $w = 0.8\mu$m, spacing $s = 0.8\mu$m, thickness $t = 2\mu$m, and height to ground $h = 9.3\mu$m. The bus parameters, unit length resistance, inductance, capacitance and coupling capacitance, are obtained by a 2D extraction tool, Raphael from Synopsys, for on-chip multi-level interconnect structures. We assume $R_S = 50\ \Omega$ and $C_L = 100$

Table 4.3: Code rates of our (2,1)-SOTA codes for an m-bit bus ($m = 5, \cdots, 32$).

m-bit	# of words	Rate	m-bit	# of words	Rate
5	10	3/5	19	530	9/19
6	14	3/6	20	702	9/20
7	18	4/7	21	930	9/21
8	24	4/8	22	1232	10/22
9	32	5/9	23	1632	10/23
10	42	5/10	24	2162	11/24
11	56	5/11	25	2864	11/25
12	74	6/12	26	3794	11/26
13	98	6/13	27	5026	12/27
14	130	7/14	28	6658	12/28
15	172	7/15	29	8820	13/29
16	228	7/16	30	11684	13/30
17	302	8/17	31	15478	13/31
18	400	8/18	32	20504	14/32

fF for simulations. To show the reduction of capacitive and inductive couplings, we also simulate interconnects without coding. For the same information bits, the scheme without coding uses less wires than our CAC scheme. Assume the scheme without coding uses equal width and spacing, we find the value of width and spacing of the scheme without coding for the same area used by our CAC scheme.

The simulation results of delays and noises are shown in Tables 4.4 and 4.5, respectively. As shown in Table 4.4, our (2,1)-SOTA codes can significantly reduce the worst case delays except for a 3-wire bus. This is because the inductive coupling is only from neighboring two wires for a 3-wire bus. For larger bus, the ring due to the increasing inductive coupling would cross the threshold multiple times for scheme without coding, leading to larger delays. For our CAC scheme, the ring is significantly reduced and the delay is determined by capacitive coupling. For $k \geq 4$,

Table 4.4: Reduction of worst case delays via our (2,1)-SOTA coding scheme over no coding scheme (NC).

NC		Ours		Reduction
k	Delay (ps)	n	Delay (ps)	
3	80.84	5	95.90	-18.63%
4	140.30	7	105.77	24.61%
5	164.39	9	106.02	35.51%
6	164.30	12	112.37	31.61%
7	169.53	14	107.06	36.85%

Table 4.5: Reduction of worst case noise via our (2,1)-SOTA coding scheme over no coding scheme (NC).

NC		Ours		Reduction
k	Noise (ps)	n	Noise (ps)	
3	0.63	5	0.38	39.68%
4	0.64	7	0.39	39.06%
5	0.65	9	0.34	47.69%
6	0.65	12	0.41	36.92%
7	0.66	14	0.38	42.42%

the reduction of worst case delay is at least about 24% compared with the scheme without coding. With regard to the peak nose, the reduction of our CAC scheme is at least 37%. Hence, our proposed CAC scheme can reduce both crosstalk delay and noise due to the capacitance and inductance effects.

4.6 CONCLUSIONS

In this chapter, we propose a new family of CACs accounting for both the capacitive and inductive couplings. The capacitive crosstalk is reduced by restricting opposite

transitions in adjacent wires and the inductive coupling is reduced by restricting same transitions in neighboring wires. CODECs based on a revised binary mixed-radix numeral system are also proposed. Simulation results show that our codes can significantly reduce the worst case delay and peak noise simultaneously. The complexity and delay of our CODECs are quadratically increasing with the size of the bus.

Chapter 5

Quasi-Cyclic Low-Density Parity-Check Stabilizer Codes

5.1 Introduction

Quantum computers are more efficient than classical computers for some computational problems, such as factoring a large number and searching an unknown space for an element satisfying a known property [11]. However, quantum information, represented by quantum bits or qubits, suffers greatly from unwanted interactions with the outside world. Thus, quantum error correction codes (QECCs) are needed to protect quantum information against noise and decoherence [11].

Many QECCs have been proposed in the literature by importing classical error correction codes, such as low-density parity-check (LDPC) codes, convolutional codes, Turbo codes, and polar codes (see, for example, [12–21]). Among them, QECCs based on LDPC codes (see, for example, [12,13,16,17]) are important, since

they can be decoded by adapting existing iterative decoding algorithms. As classical LDPC codes have asymptotically good performance for a wide class of noisy channels when decoded by the belief propagation algorithm [22], well-designed quantum LDPC codes also show good performance [16, 17, 23]. While most quantum LDPC codes are based on binary LDPC codes, recently several QECCs based on nonbinary LDPC codes have been proposed in [23] with a much better error-correcting performance than existing quantum codes over a qubit channel.

Most existing QECCs belong to two related classes. In [64], Gottesman proposed the theory of stabilizer codes, which allows us to construct QECCs based on classical error correction codes by satisfying a zero symplectic inner product (SIP) condition (also called the general stabilizer formalism). A subclass of stabilizer codes, known as CSS codes [65,66], enables us to construct QECCs by using classical error correction codes that satisfy the dual-containing condition (referred to as the CSS formalism sometimes). Since the dual-containing condition is a special case of the zero SIP condition, CSS codes are a subclass of stabilizer codes. Since the dual-containing condition is much easier to satisfy than the zero SIP condition, CSS codes have attracted a lot of attention. However, the error correction capability of CSS codes is limited [16, 17] in comparison to stabilizer codes. For example in a binary quantum system, to correct one qubit error, a CSS code takes seven qubits to encode one qubit, while a general stabilizer code needs only five qubits [67]. Most QECCs mentioned above are CSS codes. Tan *et al.* [16, 17] proposed several systematic constructions of binary quasi-cyclic low-density parity-check (QC-LDPC) based on the general stabilizer formalism, and their codes are the first LDPC stabilizer codes to the best of our knowledge.

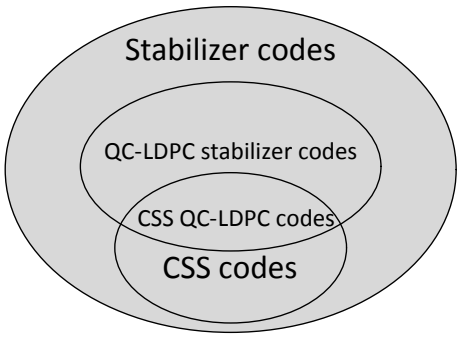


Figure 5.1: Classification of stabilizer codes.

Since stabilizer codes based on nonbinary LDPC codes have not been studied, motivated by the success of adopting nonbinary QC-LDPC codes in CSS codes in [23], in this chapter we investigate stabilizer codes based on nonbinary QC-LDPC codes, referred to as QC-LDPC stabilizer codes henceforth, for qubit channels. As in [16,17], we consider LDPC codes with a quasi-cyclic (QC) structure, which makes it easier to satisfy the zero SIP condition. The relationship of stabilizer codes, CSS codes, and QC-LDPC stabilizer codes is shown in Fig. 5.1. Our QC-LDPC stabilizer codes are a subclass of stabilizer codes, while the CSS QC-LDPC codes, including those proposed in [23], are a special case of the QC-LDPC stabilizer codes.

Our main contributions are:

- The construction of our QC-LDPC stabilizer codes is reduced to the construction of nonbinary QC-LDPC codes over GF(2^m) satisfying the zero SIP condition, and the decoding of our QC-LDPC stabilizer codes is based on that of the nonbinary QC-LDPC codes. First, we derive conditions for nonbinary QC-LDPC codes over GF(2^m) in order to satisfy the zero SIP condition and to eliminate the cycles of girth four, which usually lead to poor decoding performance by iterative decoding algorithms for LDPC codes.

- We have constructed two QC-LDPC stabilizer codes, and simulation results show that they outperform their counterparts in [16,17]. This seems to confirm the observation [23] that QECCs based on nonbinary LDPC codes appear to achieve better performance than QECCs based on binary LDPC codes.

Our work is different from recent works in [16, 17, 23]. Our QC-LDPC stabilizer codes are constructed through nonbinary codes and are decoded by a nonbinary sum-product algorithm, whereas Tan *et al.* [16, 17] focus on QC-LDPC stabilizer codes based on binary LDPC codes and decoded by a binary sum-product algorithm. Our codes also outperform those in [16, 17]. As mentioned above, the QC-LDPC codes in [23] are CSS codes, whereas our codes herein are stabilizer codes. The two stabilizer codes constructed herein have worse performance than those in [23]. However, we emphasize that the CSS codes in [23] build on extensive research on CSS codes and are the results of various optimizations in [23]. In contrast, our two codes are the first QC-LDPC stabilizer codes based on nonbinary codes. As explained above, CSS codes are a subclass of stabilizer codes, and hence stabilizer codes promise better error performance. Hence, we plan to further improve our QC-LDPC stabilizer codes in our future work.

5.2 Preliminary

In this section, we present basic concepts and notions of stabilizer codes. More details on the theory of stabilizer codes can be found in [64].

Quantum noise can be modeled in several ways. Among them, the depolarizing channel is often used to characterized a worst scenario channel, where three types

of errors, bit flip error X, phase flip error Z, and bit-and-phase flip error Y, occur independently and equal likely on each qubit [11]. For a depolarizing channel with a total flip probability f on each qubit, X, Z, and Y occur with probability $f/3$. Since a Y error is equivalent to the combination of an X error and a Z error, the marginal probability of X (Z) error is given by $2f/3$.

Stabilizer codes can be represented by a compact quaternary form with I, X, Y, Z corresponding to $0, 1, \omega, \omega^2$ over GF(4), where ω is a primitive element in GF(4) [68]. It is more convenient to denote stabilizer codes by an expanded parity check matrix over GF(2). For an $[[n, k]]_2$ stabilizer code, the $n-k$ stabilizer generators can be described as the juxtaposition of a pair of $(n-k) \times n$ matrices, $\mathbf{H} = (\mathbf{C}|\mathbf{D})$, where each row in $\mathbf{H}$ corresponds to a unique stabilizer generator and each pair of columns correspond to a qubit [16, 17]. Each "1" entry in $\mathbf{C}$ and $\mathbf{D}$ corresponds to an X and a Z operator, respectively, and each "0" entry corresponds to an I operator. For a qubit channel, such a matrix of size $(n-k) \times 2n$ over GF(2) defines a binary stabilizer code. For example, $\mathbf{H} = \begin{pmatrix} I & X & I & X \\ X & I & I & I \\ I & Z & I & Z \end{pmatrix}$ can be represented by an expanded parity check matrix $\mathbf{H} = \left(\begin{array}{cccc|cccc} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{array}\right)$.

A necessary and sufficient condition for a matrix to represent a stabilizer code is given by

Theorem 5.2.1 (Zero symplectic inner product condition [16,17])**.** *An $(n-k) \times 2n$ matrix $\mathbf{H} = (\mathbf{C}|\mathbf{D})$ is a parity check matrix of a stabilizer code if and only if $\mathbf{H}$ satisfies*

$$\mathbf{C}\mathbf{D}^T + \mathbf{D}\mathbf{C}^T = \mathbf{0}, \tag{5.1}$$

where T denotes the matrix transpose and $\mathbf{0}$ denotes an $(n-k) \times (n-k)$ zero matrix.

Many existing stabilizer codes are based on the CSS formalism, which makes use of classical dual-containing codes for the design of QECCs. Let $\mathbf{H}_C$ and $\mathbf{H}_D$ be two parity check matrices corresponding to two classical code C and D, respectively. If $D^{\perp} \subset C$ (the dual code $D^{\perp}$ of D is a subset of C), then $\mathbf{H}_C\mathbf{H}_D^T = \mathbf{0}$, which is referred to as the dual-containing condition. The following matrix defines a stabilizer code [65, 66]:

$$\mathbf{H} = \left(\begin{array}{c|c} \mathbf{H}_C & \mathbf{0} \\ \mathbf{0} & \mathbf{H}_D \end{array}\right).$$

If $\mathbf{H}_C = \mathbf{H}_D$, the dual-containing condition reduces to $\mathbf{H}_C\mathbf{H}_C^T = \mathbf{0}$. Code C (D) is called a weakly self-dual code. A stabilizer matrix is given by

$$\mathbf{H} = \left(\begin{array}{c|c} \mathbf{H}_C & \mathbf{0} \\ \mathbf{0} & \mathbf{H}_C \end{array}\right).$$

It can be easily verified that CSS codes satisfy the zero SIP condition in Eq. (5.1). CSS codes are a special family of stabilizer codes.

Recently, nonbinary LDPC codes have been used for the construction of binary CSS codes through a ring homomorphism [23]. A ring homomorphism, $A : \mathrm{GF}(2^m) \rightarrow \mathrm{GF}(2)^{m\times m}$ with its images homomorphic to $\mathrm{GF}(2^m)$ by matrix addition and multiplication operations, is given in [23]. Let α be a primitive element of $\mathrm{GF}(2^m)$. The minimal polynomial of α is $\pi(x) = \sum_{i=0}^{m-1} \pi_i x^i + x^m$. Such a mapping is given by $A(\alpha^i) := A(\alpha)^i \in \mathrm{GF}(2)^{m\times m}, \forall \alpha^i \in \mathrm{GF}(2^m)$, with $A(0) = \mathbf{0}$ and

$$A(\alpha) := \begin{pmatrix} 0 & 0 & \cdots & 0 & \pi_0 \\ 1 & 0 & \cdots & 0 & \pi_1 \\ 0 & 1 & \cdots & 0 & \pi_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & \pi_{m-1} \end{pmatrix}.$$

By the ring homomorphism, two nonbinary LDPC codes satisfying the zero SIP condition can be mapped to a binary stabilizer code. For example, for $\mathbf{C}, \mathbf{D} \in \mathrm{GF}(2^m)^{M\times N}$, $\mathbf{C}^A, \mathbf{D}^A \in \mathrm{GF}(2)^{mM\times mN}$ are obtained by replacing all entries in $\mathbf{C}$ and $\mathbf{D}$ with their images under the mapping A. If $\mathbf{C}\mathbf{D}^T + \mathbf{D}\mathbf{C}^T = \mathbf{0}$, then $\mathbf{C}^A(\mathbf{D}^A)^T + \mathbf{D}^A(\mathbf{C}^A)^T = \mathbf{0}$. Hence, it is shown that $\mathbf{H} = (\mathbf{C}|\mathbf{D})$ satisfying Eq. (5.1) over $\mathrm{GF}(2^m)$ defines a binary stabilizer code with $(\mathbf{C}^A|\mathbf{D}^A)$.

Quantum LDPC codes can be decoded by a belief propagation decoding algorithm similar to that of classical LDPC codes [12,69]. For CSS codes with a parity check matrix $\mathbf{H} = \left(\begin{smallmatrix} \mathbf{H}_C & \mathbf{0} \\ \mathbf{0} & \mathbf{H}_D \end{smallmatrix}\right)$, X and Z errors can be corrected by decoding $\mathbf{C}$ and $\mathbf{D}$, respectively. For general stabilizer codes with a parity check matrix $\mathbf{H} = (\mathbf{C}|\mathbf{D})$, it has been shown that the decoding is equivalent to a syndrome version of sum-product algorithm on the Tanner graph of $[\mathbf{C}, \mathbf{D}]$, which is obtained by merging corresponding checks of $\mathbf{C}$ and $\mathbf{D}$ [16,17].

5.3 QC-LDPC Stabilizer Codes

In this section, we propose two constructions of QC-LDPC stabilizer codes for a qubit channel. This is achieved by constructing nonbinary QC-LDPC codes over finite fields of characteristic two satisfying Eq. (5.1). This is because the state of a

qubit in most quantum systems is binary and nonbinary codes over GF(2^m) can be easily connected to a binary stabilizer code in a qubit channel. Also, a nonbinary LDPC code satisfying Eq. (5.1) over GF(2^m) defines a binary stabilizer code, which can be decoded by a sum-product algorithm for nonbinary LDPC codes. The rest of the work is to find good nonbinary QC-LDPC codes with parity check matrices satisfying Eq. (5.1) over GF(2^m).

We focus on nonbinary codes over GF(2^m) with column weight two only, since the nonbinary LDPC codes with column weight two over GF(2^m) are empirically known as the best performing codes for $2^m \geq 64$ [70]. Several approaches to the construction of QC-LDPC codes have been proposed based on finite geometry, arrays and array dispersions, and finite fields [71–73]. The key idea is first constructing a base matrix over some finite field satisfying a certain constraint, and then replacing the elements in the base matrix by binary or nonbinary cyclic matrices to obtain parity check matrices of QC-LDPC codes.

We propose the following method to obtain a nonbinary parity check matrix over GF(2^m) via a pair of base matrices over GF(2). We first construct two base quasi-cyclic parity check matrices $\mathbf{C}^b$ and $\mathbf{D}^b$ satisfying Eq. (5.1) with column weight $J = 2$ and row weight L. Both matrices consist of $2 \times L$ block matrices, which are shifted identity matrices of size $P \times P$. Then, we use the pair of base parity check matrices $\mathbf{H}^b = (\mathbf{C}^b|\mathbf{D}^b)$ and replace each one in $\mathbf{C}^b$ and $\mathbf{D}^b$ with a nonzero element in GF(2^m). By solving a set of linear equations over $\mathbb{Z}_{2^m-1}$ satisfying Eq. (5.1), we obtain two $2P \times LP$ nonbinary parity check matrices $\mathbf{C}$ and $\mathbf{D}$, which form a parity check matrix $\mathbf{H} = (\mathbf{C}|\mathbf{D})$ over GF(2^m). The code length is given by LP symbols and the number of information symbols is approximated by $LP - 2P$. Hence, the

quantum code rate is lower bounded by $R_Q = 1 - 2/L$.

5.3.1 Base parity check matrix

Our nonbinary quantum codes are obtained from their base. Hence, the performance of the nonbinary codes is affected by the parameters of the base matrices. There are three parameters to consider, row weight, minimum distance, and girth, when designing such base matrices. At the error-floor region, small minimum distance leads to poor decoding performance. If the regular (J, L) LDPC code is a CSS code, the minimum distance is upper-bounded by the row weight L due to the dual-containing condition and sparsity of the parity check matrix [23]. To have a large minimum distance, the row weight of the parity check matrix should be chosen large. At the waterfall region, the sum-product decoding performance degrades with increasing row weight L [74]. So the row weight L should not be too large. It is also known that cycles of girth four in the Tanner graph degrade the SP decoding performance [23]. The cycles of girth four can be classified into two groups, critical cycles of girth four and non-critical cycles of girth four [16]. The critical cycles of girth four are present in both the compact quaternary form and the expanded form and the non-critical cycles of girth four present only in the compact form. For example, $\mathbf{H}_1 = \begin{pmatrix} I & X & I & X \\ Z & I & I & I \\ I & X & I & X \end{pmatrix}$ contains a critical cycle of girth four, and $\mathbf{H}_2 = \begin{pmatrix} I & X & I & X \\ Z & I & I & I \\ I & Z & I & Z \end{pmatrix}$ contains only a non-critical cycle of girth four. In our work, we consider only the cycles of girth four in the expanded parity check matrix $\mathbf{H} = (\mathbf{C}|\mathbf{D})$, since our decoding scheme is based on the Tanner graph corresponding to the expanded parity check matrix. It is desired to reduce or avoid critical cycles of girth four.

In the following, we first introduce the base matrices used in the quasi-cyclic

structure of our proposed stabilizer codes. Then, we introduce a juxtaposition technique to construct longer codes.

Definition [Binary cyclic matrices] Let $\mathbf{I}$ be a $P \times P$ identity matrix. A binary cyclic matrix $\mathbf{I}(1) \in \{0,1\}^{P\times P}$ is obtained by cyclicly shifting each row of $\mathbf{I}$ to the right by one position:

$$\mathbf{I}(1) := \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & \cdots & 0 \end{pmatrix}.$$

We define $\mathbf{I}(0) := \mathbf{I}$ and $\mathbf{I}(i) := \mathbf{I}(1)^i$, where i is the offset and $0 < i < P$. We have $\mathbf{I}(a)\mathbf{I}(b) = \mathbf{I}(a+b)$ and $\mathbf{I}^T(a) = \mathbf{I}(-a)$.

To obtain longer codes with different code rate, we juxtapose shorter codes as follows.

Definition [Juxtaposition of Matrices] For a set of matrices $\mathbf{C}_1, \mathbf{C}_2, \cdots, \mathbf{C}_L$ and $\mathbf{D}_1, \mathbf{D}_2, \cdots, \mathbf{D}_L$ with the same number of rows, we juxtapose corresponding pairs horizontally as $\mathbf{H} = (\mathbf{C}_1\mathbf{C}_2\cdots\mathbf{C}_L|\mathbf{D}_1\mathbf{D}_2\cdots\mathbf{D}_L)$.

It is shown that juxtaposition preserves the zero SIP condition in the following lemma.

Lemma 5.3.1 ([16,17]). *Let $\mathbf{H}_i = (\mathbf{C}_i|\mathbf{D}_i)$ for $1 \le i \le L$. If $\mathbf{H}_i$ satisfies the zero SIP condition in Eq. (5.1), the juxtaposed matrix $\mathbf{H} = (\mathbf{C}_1\mathbf{C}_2\cdots\mathbf{C}_L|\mathbf{D}_1\mathbf{D}_2\cdots\mathbf{D}_L)$ also satisfies the zero SIP condition.*

According to Lemma 5.3.1, we first construct shorter codes with a pair of matrices $\mathbf{C}$ and $\mathbf{D}$ satisfying the zero SIP condition. In the following, we propose a

construction of binary QC stabilizer codes free of cycles of girth four.

Let $\mathbf{H} = (\mathbf{C}|\mathbf{D})$, where $\mathbf{C} = \begin{pmatrix} \mathbf{I}(c_{1,1}) & \mathbf{I}(c_{1,2}) \\ \mathbf{I}(c_{2,1}) & \mathbf{I}(c_{2,2}) \end{pmatrix}$ and $\mathbf{D} = \begin{pmatrix} \mathbf{I}(d_{1,1}) & \mathbf{I}(d_{1,2}) \\ \mathbf{I}(d_{2,1}) & \mathbf{I}(d_{2,2}) \end{pmatrix}$.

Theorem 5.3.1. *A sufficient condition for a binary QC-LDPC stabilizer code with* $\mathbf{H} = (\mathbf{C}|\mathbf{D})$ *satisfying the zero SIP condition is given by*

$$\begin{cases} c_{1,1} - d_{1,1} = d_{1,2} - c_{1,2} \\ c_{2,1} - d_{1,1} = d_{2,1} - c_{1,1} \quad (mod\ P). \\ c_{2,2} - d_{1,2} = d_{2,2} - c_{1,2} \end{cases} \tag{5.2}$$

Proof. $\mathbf{CD}^T + \mathbf{DC}^T = \begin{pmatrix} \mathbf{I}(c_{1,1}) & \mathbf{I}(c_{1,2}) \\ \mathbf{I}(c_{2,1}) & \mathbf{I}(c_{2,2}) \end{pmatrix} \begin{pmatrix} \mathbf{I}(-d_{1,1}) & \mathbf{I}(-d_{2,1}) \\ \mathbf{I}(-d_{1,2}) & \mathbf{I}(-d_{2,2}) \end{pmatrix} + \begin{pmatrix} \mathbf{I}(d_{1,1}) & \mathbf{I}(d_{1,2}) \\ \mathbf{I}(d_{2,1}) & \mathbf{I}(d_{2,2}) \end{pmatrix} \begin{pmatrix} \mathbf{I}(-c_{1,1}) & \mathbf{I}(-c_{2,1}) \\ \mathbf{I}(-c_{1,2}) & \mathbf{I}(-c_{2,2}) \end{pmatrix} =$
$\begin{bmatrix} \mathbf{I}(c_{1,1}-d_{1,1})+\mathbf{I}(c_{1,2}-d_{1,2})+\mathbf{I}(d_{1,1}-c_{1,1})+\mathbf{I}(d_{1,2}-c_{1,2}) & \mathbf{I}(c_{1,1}-d_{2,1})+\mathbf{I}(c_{1,2}-d_{2,2})+\mathbf{I}(d_{1,1}-c_{2,1})+\mathbf{I}(d_{1,2}-c_{2,2}) \\ \mathbf{I}(c_{2,1}-d_{1,1})+\mathbf{I}(c_{2,2}-d_{1,2})+\mathbf{I}(d_{2,1}-c_{1,1})+\mathbf{I}(d_{2,2}-c_{1,2}) & \mathbf{I}(c_{2,1}-d_{2,1})+\mathbf{I}(c_{2,2}-d_{2,2})+\mathbf{I}(d_{2,1}-c_{2,1})+\mathbf{I}(d_{2,2}-c_{2,2}) \end{bmatrix}$
$= \mathbf{0}$. Hence, $\mathbf{H}$ satisfies the zero SIP condition. □

Example: Given parameters $J = 2$, $L = 2$, and $P = 15$, a parity check matrix of a (2,2) QC stabilizer code is given by $\left(\begin{array}{cc|cc} \mathbf{I}(7) & \mathbf{I}(5) & \mathbf{I}(13) & \mathbf{I}(1) \\ \mathbf{I}(5) & \mathbf{I}(7) & \mathbf{I}(1) & \mathbf{I}(13) \end{array}\right)$.

To construct longer codes, we juxtapose additional pairs of codes satisfying the condition in Eq. (5.1). The problem of the juxtaposition is that cycles of girth four can be introduced, if the offset parameters are not carefully chosen. For example, for $P = 15$, an expanded parity matrix of a $(2, 4)$ code satisfying Eq. (5.2) is given by $\left(\begin{array}{cccc|cccc} \mathbf{I}(7) & \mathbf{I}(5) & \mathbf{I}(8) & \mathbf{I}(6) & \mathbf{I}(3) & \mathbf{I}(9) & \mathbf{I}(12) & \mathbf{I}(2) \\ \mathbf{I}(5) & \mathbf{I}(7) & \mathbf{I}(6) & \mathbf{I}(8) & \mathbf{I}(9) & \mathbf{I}(3) & \mathbf{I}(2) & \mathbf{I}(12) \end{array}\right)$, where the four cyclic matrices with italic offsets introduce cycles of girth four since $7 - 5 = 8 - 6$ (mod 15).

Let $\mathcal{H} = (h_{j,l})$ denote a matrix containing the offset information of $\mathbf{H} = (\mathbf{I}(h_{j,l}))$, where $1 \leq j \leq J$ and $1 \leq l \leq L$. The following theorem gives a necessary and sufficient condition to avoid cycles of girth four for a QC-LDPC code with $\mathcal{H} = (h_{j,l})$.

Theorem 5.3.2 ([75])**.** *A QC-LDPC code with* $\mathcal{H} = (h_{j,l})$ *has no cycles of girth*

four if and only if $h_{j_1,l_1} - h_{j_2,l_1} \neq h_{j_1,l_2} - h_{j_2,l_2}(\ mod\ P)$ *for* $1 \leq j_1 < j_2 \leq J$ *and* $1 \leq l_1 < l_2 \leq L$, *where* P *is the size of cyclic matrices* $\mathbf{I}(h_{j,l})$.

Example: Given parameters $J = 2$, $L = 4$, and $P = 15$, a parity check matrix of a $(2, 4)$ QC stabilizer code is given by $\left(\begin{array}{cccc|cccc} \mathbf{I}(7) & \mathbf{I}(5) & \mathbf{I}(8) & \mathbf{I}(5) & \mathbf{I}(3) & \mathbf{I}(9) & \mathbf{I}(12) & \mathbf{I}(1) \\ \mathbf{I}(5) & \mathbf{I}(7) & \mathbf{I}(5) & \mathbf{I}(8) & \mathbf{I}(9) & \mathbf{I}(3) & \mathbf{I}(1) & \mathbf{I}(12) \end{array}\right)$, where no cycle of girth four exists.

5.3.2 QC-LDPC stabilizer codes with no cycles of girth four

The parity check matrix of nonbinary QC-LDPC codes can be obtained from a pair of base matrices based on single-weight shifted identity matrices. This is achieved by replacing the ones in its binary image with nonzero elements in GF(2^m) such that the nonbinary matrix $\mathbf{H}$ satisfies the zero SIP condition in Eq. (5.1) and defines a binary stabilizer code.

Let $\mathbf{H}^b = (\mathbf{C}^b|\mathbf{D}^b)$ be a parity check matrix of a $(2, 2)$ binary $[[N, K]]_2$ code, where $\mathbf{C}^b = (\mathbf{I}(c_{i,j}))_{2P\times 2P}$ ($\mathbf{D}^b = (\mathbf{I}(d_{i,j}))_{2P\times 2P}$) for $i, j = 1, 2$. A sufficient condition for zero SIP is given in Eq. (5.2).

Let α be a primitive element in GF(2^m). Suppose each block $\mathbf{I}(c_{i,j})$ ($\mathbf{I}(d_{i,j})$, respectively) of $\mathbf{C}^b$ ($\mathbf{D}^b$, respectively) is replaced with $\mathbf{X}_{ij}(c_{i,j}) = \mathrm{diag}(\alpha^{x_{i,j,1}}, \cdots, \alpha^{x_{i,j,P}}) \cdot \mathbf{I}(c_{i,j})$ ($\mathbf{Y}_{ij}(d_{i,j}) = \mathrm{diag}(\alpha^{y_{i,j,1}}, \cdots, \alpha^{y_{i,j,P}}) \cdot \mathbf{I}(d_{i,j})$, respectively) for $i, j = 1, 2$. For simplicity, we denote $\mathbf{X}_{ij}(c_{i,j})$ and $\mathbf{Y}_{ij}(c_{i,j})$ as $\mathbf{X}_{ij}$ and $\mathbf{Y}_{ij}$, respectively, when there is no ambiguity about the offsets $c_{i,j}$ and $d_{i,j}$. Since each component block matrix $\mathbf{X}_{ij}$ ($\mathbf{Y}_{ij}$, respectively) has P nonzeros $\alpha^{x_{i,j,l}}$ ($\alpha^{y_{i,j,l}}$, respectively) over GF(2^m), there are a total of $8P$ unknown exponents $x_{i,j,l}$'s and $y_{i,j,l}$'s to determine. After the replacement, the parity check matrix is given by $\mathbf{H} = (\mathbf{C}|\mathbf{D}) = \left(\begin{array}{cc|cc} \mathbf{X}_{11} & \mathbf{X}_{12} & \mathbf{Y}_{11} & \mathbf{Y}_{12} \\ \mathbf{X}_{21} & \mathbf{X}_{22} & \mathbf{Y}_{21} & \mathbf{Y}_{22} \end{array}\right)$.

$$\mathbf{C}\mathbf{D}^T + \mathbf{D}\mathbf{C}^T = \begin{pmatrix} \mathbf{X}_{11}\mathbf{Y}_{11}^T + \mathbf{X}_{12}\mathbf{Y}_{12}^T + \mathbf{Y}_{11}\mathbf{X}_{11}^T + \mathbf{Y}_{12}\mathbf{X}_{12}^T & \mathbf{X}_{11}\mathbf{Y}_{21}^T + \mathbf{X}_{12}\mathbf{Y}_{22}^T + \mathbf{Y}_{11}\mathbf{X}_{21}^T + \mathbf{Y}_{12}\mathbf{X}_{22}^T \\ \mathbf{X}_{21}\mathbf{Y}_{11}^T + \mathbf{X}_{22}\mathbf{Y}_{12}^T + \mathbf{Y}_{21}\mathbf{X}_{11}^T + \mathbf{Y}_{22}\mathbf{X}_{12}^T & \mathbf{X}_{21}\mathbf{Y}_{21}^T + \mathbf{X}_{22}\mathbf{Y}_{22}^T + \mathbf{Y}_{21}\mathbf{X}_{21}^T + \mathbf{Y}_{22}\mathbf{X}_{22}^T \end{pmatrix}. \tag{5.3}$$

$$\begin{aligned} & \mathbf{C}^R(\mathbf{D}^R)^T + \mathbf{D}^R(\mathbf{C}^R)^T \\ = & \begin{pmatrix} \mathbf{X}_{11} & \mathbf{X}_{12}\mathbf{R}^T \\ \mathbf{R}\mathbf{X}_{21} & \mathbf{R}\mathbf{X}_{22}\mathbf{R}^T \end{pmatrix} \begin{pmatrix} \mathbf{Y}_{11} & \mathbf{Y}_{12}\mathbf{R}^T \\ \mathbf{R}\mathbf{Y}_{21} & \mathbf{R}\mathbf{Y}_{22}\mathbf{R}^T \end{pmatrix}^T + \begin{pmatrix} \mathbf{Y}_{11} & \mathbf{Y}_{12}\mathbf{R}^T \\ \mathbf{R}\mathbf{Y}_{21} & \mathbf{R}\mathbf{Y}_{22}\mathbf{R}^T \end{pmatrix} \begin{pmatrix} \mathbf{X}_{11} & \mathbf{X}_{12}\mathbf{R}^T \\ \mathbf{R}\mathbf{X}_{21} & \mathbf{R}\mathbf{X}_{22}\mathbf{R}^T \end{pmatrix}^T \\ = & \begin{pmatrix} (\mathbf{X}_{11}\mathbf{Y}_{11}^T + \mathbf{X}_{12}\mathbf{Y}_{12}^T) + (\mathbf{X}_{11}\mathbf{Y}_{11}^T + \mathbf{X}_{12}\mathbf{Y}_{12}^T)^T & [(\mathbf{X}_{11}\mathbf{Y}_{21}^T + \mathbf{X}_{12}\mathbf{Y}_{22}^T) + (\mathbf{X}_{21}\mathbf{Y}_{11}^T + \mathbf{X}_{22}\mathbf{Y}_{12}^T)^T]\mathbf{R}^T \\ \mathbf{R}[(\mathbf{X}_{21}\mathbf{Y}_{11}^T + \mathbf{X}_{22}\mathbf{Y}_{12}^T) + (\mathbf{X}_{11}\mathbf{Y}_{21}^T + \mathbf{X}_{12}\mathbf{Y}_{22}^T)^T] & \mathbf{R}[(\mathbf{X}_{21}\mathbf{Y}_{21}^T + \mathbf{X}_{22}\mathbf{Y}_{22}^T) + (\mathbf{X}_{21}\mathbf{Y}_{21}^T + \mathbf{X}_{22}\mathbf{Y}_{22}^T)^T]\mathbf{R}^T \end{pmatrix}. \end{aligned} \tag{5.4}$$

The symplectic inner product is shown in Eq. (5.3). Due to the quasi-cyclic structure, each of the four block matrices in Eq. (5.3) would introduce P linear equations of exponents $x_{i,j,l}$'s and $y_{i,j,l}$'s for $i, j = 1, 2$ and there are a total of $4P$ equations. Since the number of equations is smaller than the number of variables, we can always find a set of solutions satisfying the zero SIP condition. By picking randomly from the solutions, we obtain a parity check matrix $\mathbf{H} = (\mathbf{C}|\mathbf{D})$ over GF(2^m). Then, we can use juxtaposition to obtain codes with different rates.

5.3.3 QC-LDPC stabilizer codes with rotation

In the following, we use the rotation operation similar to that in [16,17] to increase the randomness.

Definition [General rotation operation]: A binary square matrix $\mathbf{R}$ is called a general rotational matrix if $\mathbf{R}^T = \mathbf{R}^{-1}$. We only focus on sparse matrix $\mathbf{R}$, since dense matrix could increase the density of sparse parity check matrix. Permutation matrix is a special rotational matrix and is used in our work for rotation operations.

The general rotation operation Π on a square matrix $\mathbf{X}$ is given by

$$\Pi\{\mathbf{X}\} = \mathbf{R}\mathbf{X}^T,$$
$$\Pi^k\{\mathbf{X}\} = \Pi\{\Pi^{k-1}\{\mathbf{X}\}\}, \quad k = 2, 3, \cdots.$$

Let $\mathbf{H} = \left(\begin{smallmatrix} \mathbf{X}_{11} & \mathbf{X}_{12} \\ \mathbf{X}_{21} & \mathbf{X}_{22} \end{smallmatrix} \middle| \begin{smallmatrix} \mathbf{Y}_{11} & \mathbf{Y}_{12} \\ \mathbf{Y}_{21} & \mathbf{Y}_{22} \end{smallmatrix}\right)$ be a parity check matrix obtained in Sec. 5.3.2. We apply rotation operation on $\mathbf{H}$ and obtain $\mathbf{H}^R$ as follows: $\mathbf{H}^R = (\mathbf{C}^R|\mathbf{D}^R) = \left(\begin{smallmatrix} \mathbf{X}_{11} & (\Pi\{\mathbf{X}_{12}\})^T \\ \Pi\{\mathbf{X}_{21}^T\} & \Pi^2\{\mathbf{X}_{22}\} \end{smallmatrix} \middle| \begin{smallmatrix} \mathbf{Y}_{11} & (\Pi\{\mathbf{Y}_{12}\})^T \\ \Pi\{\mathbf{Y}_{21}^T\} & \Pi^2\{\mathbf{Y}_{22}\} \end{smallmatrix}\right) = \left(\begin{smallmatrix} \mathbf{X}_{11} & \mathbf{X}_{12}\mathbf{R}^T \\ \mathbf{R}\mathbf{X}_{21} & \mathbf{R}\mathbf{X}_{22}\mathbf{R}^T \end{smallmatrix} \middle| \begin{smallmatrix} \mathbf{Y}_{11} & \mathbf{Y}_{12}\mathbf{R}^T \\ \mathbf{R}\mathbf{Y}_{21} & \mathbf{R}\mathbf{Y}_{22}\mathbf{R}^T \end{smallmatrix}\right)$. Then, the symplectic inner product is shown in Eq. (5.4). When $\mathbf{X}_{11}\mathbf{Y}_{11}^T + \mathbf{X}_{12}\mathbf{Y}_{12}^T = \mathbf{0}$, $\mathbf{X}_{11}\mathbf{Y}_{21}^T + \mathbf{X}_{12}\mathbf{Y}_{22}^T = \mathbf{0}$, $\mathbf{X}_{21}\mathbf{Y}_{11}^T + \mathbf{X}_{22}\mathbf{Y}_{12}^T = \mathbf{0}$, and $\mathbf{X}_{21}\mathbf{Y}_{21}^T + \mathbf{X}_{22}\mathbf{Y}_{22}^T = \mathbf{0}$, $\mathbf{H}^R$ satisfies the zero SIP condition. Note that $\mathbf{X}_{ij}$ and $\mathbf{Y}_{ij}$ are single-weight cyclic matrices, each set of the four sets of equations above would introduce P linear equations of unknown exponents $x_{i,j,l}$'s and $y_{i,j,l}$'s. We obtain a total of $4P$ linear equations. Since each component block matrix of $\mathbf{C}^R$ and $\mathbf{D}^R$ has P nonzeros, there are a total of $8P$ unknown exponents $x_{i,j,l}$'s and $y_{i,j,l}$'s to determine. Thus, we can always find a set of solutions satisfying the zero SIP condition. By picking randomly from the set of solutions, we obtain a parity check matrix $\mathbf{H}^R = (\mathbf{C}^R|\mathbf{D}^R)$ over GF(2^m). A higher rate code can be obtained by juxtaposition.

5.4 Performance Evaluation

In this section, we evaluate our QC-LDPC stabilizer codes in a qubit depolarizing channel with a total flip probability f, where bit-flip error X, phase-flip error Z, and bit-and-phase flip error Y occur independently with probability $f/3$ [11]. Our binary QC-LDPC stabilizer codes are constructed through nonbinary QC-LDPC

codes satisfying Eq. (5.1). The decoding is based on the nonbinary QC-LDPC codes with an expanded parity matrix $\mathbf{H} = (\mathbf{C}|\mathbf{D})$ over GF(2^m). Hence, we use a sum-product algorithm powered by FFT over finite field GF(2^m) [76] and simulate the frame error rate (FER) of our codes. In [21], Dutton *et al.* proposed quantum polar codes with good performance for the depolarizing channel. However, at medium length, the codes proposed by Kasai *et al.* in [23] outperform the quantum polar codes [21]. Hence, we compare our code with the best CSS QC-LDPC code in [23]. We also include the best binary QC-LDPC stabilizer code in [16,17] for comparison, since both our codes and the codes in [16, 17] are based on the general stabilizer formalism.

Using the approach described in Sec. 5.3.2, we first construct a $(2, 4)$ nonbinary QC-LDPC code (referred to as code 1) over GF(2^8) without cycles of girth four. It has a code rate 1/2 and length of 520 symbols, which is equivalent to 2080 qubits. A parity check matrix of code 1 is given by

$$\mathbf{H} = \left(\begin{array}{cccc|cccc} \mathbf{X}_{11}(1) & \mathbf{X}_{12}(64) & \mathbf{X}_{13}(7) & \mathbf{X}_{14}(58) & \mathbf{Y}_{11}(5) & \mathbf{Y}_{12}(60) & \mathbf{Y}_{13}(13) & \mathbf{Y}_{14}(52) \\ \mathbf{X}_{21}(64) & \mathbf{X}_{22}(1) & \mathbf{X}_{23}(58) & \mathbf{X}_{24}(7) & \mathbf{Y}_{21}(60) & \mathbf{Y}_{22}(5) & \mathbf{Y}_{23}(52) & \mathbf{Y}_{24}(13) \end{array} \right),$$

where $\mathbf{X}_{i,j}(c_{i,j})$ and $\mathbf{Y}_{i,j}(d_{i,j})$ are cyclic shifted matrix over GF(2^8) with a size of $P = 65$. Based on the approach described in Sec. 5.3.3, we obtain another $(2, 4)$ nonbinary QC-LDPC code (referred to as code 2) over GF(2^8), which has the same offsets, code rate, and length as code 1. A parity check matrix of code 2 is given by $\mathbf{H}^R = \left(\begin{array}{cccc|cccc} \mathbf{X}_{11}(1) & \mathbf{X}_{12}(64)\mathbf{R}^T & \mathbf{X}_{13}(7) & \mathbf{X}_{14}(58)\mathbf{R}^T & \mathbf{Y}_{11}(5) & \mathbf{Y}_{12}(60)\mathbf{R}^T & \mathbf{Y}_{13}(13) & \mathbf{Y}_{14}(52)\mathbf{R}^T \\ \mathbf{R}\mathbf{X}_{21}(64) & \mathbf{R}\mathbf{X}_{22}(1)\mathbf{R}^T & \mathbf{R}\mathbf{X}_{23}(58) & \mathbf{R}\mathbf{X}_{24}(7)\mathbf{R}^T & \mathbf{R}\mathbf{Y}_{21}(60) & \mathbf{R}\mathbf{Y}_{22}(5)\mathbf{R}^T & \mathbf{R}\mathbf{Y}_{23}(52) & \mathbf{R}\mathbf{Y}_{24}(13)\mathbf{R}^T \end{array} \right)$. The two parity check matrices of codes 1 and 2 with column weight 2, row weight 8, and size 130×520 are plotted in Fig. 5.2, where each dot denotes a nonzero elements.

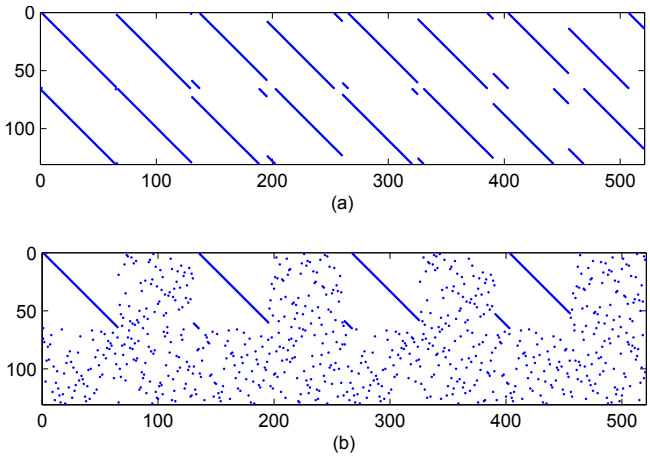


Figure 5.2: Parity check matrices of (a) Code 1 (b) Code 2.

We compare our codes with the KHIS (Kasai-Hagiwara-Imai-Sakaniwa) codes proposed in [23], which are the best known CSS QC-LDPC codes. The KHIS code for comparison has a code rate 1/2 and length of 2624 qubits. We also compare our codes with code B in [16, 17], which has a code rate 1/2 and length of 2068 qubits. The FER performances of our codes, code B in [16, 17], and KHIS code in [23] are shown in Fig. 5.3. Our code 2 has better FER performance than code 1 up to 10^{-3}. It shows that the rotation operation can improve the FER performance by introducing randomness, which can reduce the number of smallest cycles. Though both our codes and code B in [16,17] are for a qubit channel, our codes outperform code B by using a nonbinary LDPC decoding algorithm. We conclude that QC-LDPC stabilizer codes via a nonbinary decoding algorithm have better FER performance than the binary QC-LDPC stabilizer codes via a binary decoding algorithm. The performance of our code 1 is not as good as the KHIS code in [23]. This is because our construction removes only the cycles of girth four. In contrast, the method in constructing KHIS codes ensures that cycles of girth up to $2L$ are eliminated, where L is the row weight.

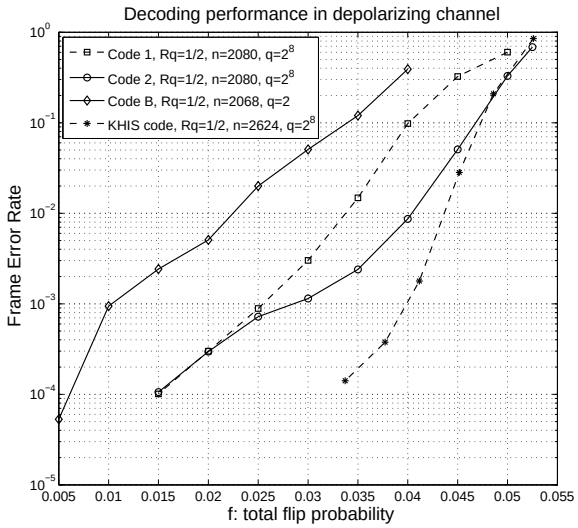


Figure 5.3: The block error probability of our codes, code B in [16, 17], and KHIS code in [23].

By introducing randomness, which can reduce the number of girth six cycles, the performance of our code 2 is better than code 1. Our codes do not perform well as the KHIS code. We remark that this comparison is somewhat misleading and to our disadvantage definitely, because the KHIS code in [23] is the result of significant optimization efforts on CSS codes, whereas our code 1 is obtained without much optimization. With more work on optimizations of our stabilizer codes, it is possible to find better QC-LDPC stabilizer codes than the CSS QC-LDPC codes, which is shown to be a special case of the QC-LDPC stabilizer codes.

5.5 Summary

Many quantum error correction codes in the literature are based on the CSS formalism, which uses classical dual-containing codes as component codes. The drawback

of CSS codes is the restriction on the code structure, which leads to a lower rate code compared with non-CSS codes. In this work, we focus on the constructions of quantum codes based on the general stabilizer formalism and propose two constructions of QC-LDPC stabilizer codes decoded by a nonbinary sum-product algorithm. Our simulation results show that our nonbinary quantum QC-LDPC codes outperform their binary counterparts. Though the performance of our codes are not good as the best CSS QC-LDPC code in [23], it possibly leads to better codes than CSS QC-LDPC codes by further reducing the number of smallest cycles. We plan to search for better codes in our future work.

Chapter 6

Efficient Threshold Architectures with Bounded Fan-Ins for Finite Field Operations

6.1 Introduction

According to the International Technology Roadmap of Semiconductors (ITRS) [1], the conventional CMOS technology has great challenges in further scaling. Although new materials and device structures can keep the CMOS scaling for the next ten years, it would reach fundamental limits during 2020–2025 [1]. After that, it would be difficult to operate any MOS-based transistor structure using classical physics. With smaller feature sizes, higher speeds, and lower power consumption, some emerging nanotechnology devices such as resonant tunneling diodes (RTDs), quantum cellular automata (QCA) and single electron transistors (SETs)

are promising candidates to replace the CMOS devices. At the system level, they have two distinct advantages over their CMOS counterparts. Firstly they can easily realize threshold gates (see Fig. 6.2). Since threshold gates can implement complex Boolean functions with single gates [25], the area of larger systems implemented using nanotechnology tends to be a lot smaller. Secondly, the outputs of the threshold gates built with nanotechnology are self-latched. This provides a natural way of pipelining these systems in most signal processing applications.

Several applications dealing with real valued signals have already been realized based on threshold gates, such as parallel adders via RTDs [27–29], comparison via QCAs [77], pattern matching for nanotechnology [78], parity via neural networks [79], multiplication via neural networks [80–82], division via neural networks [83]. However, an important class of signal processing applications, including error correcting coding and cryptography which use characteristic-2 fields (denoted by GF(2^m)) [30, 84], have yet been realized using nanotechnology.

The main obstacle to the nanotechnology implementations of applications over finite field GF(2^m) is that all arithmetic operations, addition, multiplication [30] and inversion [85–89], require exclusive-ORs (XORs). Unlike most conventional Boolean primitives such as AND, OR, NOT, NAND and NOR, XOR is not a threshold function and cannot be realized as a single threshold gate [90]. While two-input XORs have been the focus in CMOS technology, multi-input XORs are better suited to finite field applications, which typically employ a large number of XOR operations. Further, multi-input XORs allow us to better exploit the power of the threshold gates. Previously proposed threshold logic gate (TLG) implementation of n-input XORs have linear ($O(n)$) [91, 92] or sublinear ($O(\sqrt{n})$) [81] number of threshold

gates. However, these TLG implementations of multi-input XORs require threshold gates with unbounded fan-ins. Bounded fan-in is critical to both reliability and performance of nanotechnology architectures. The reliability of a threshold gate in nanotechnology decreases sharply as the fan-in grows [93]. In addition, threshold gates with a large fan-in tend to have slower switching speeds [94]. Previous theoretical results on XOR implementation with threshold logic ignored the practically important fan-in bounds [91, 92], and hence they are not readily applicable when fan-in is constrained.

One straightforward method of applying fan-in constraint to a multi-input XOR is to decompose each threshold gate of the XOR in [81,91,92] into gates with smaller fan-ins. Many approaches for decomposing threshold functions or symmetric functions have been proposed in [95–99]. In [95], the decomposition of an arbitrary Boolean function of n inputs considers only a fan-in of 2 and has a complexity of $O(2^n/n)$. In [96–98], a divide and conquer algorithm for fan-in reduction was proposed for a majority function of n inputs with a complexity of quasi-polynomial in n, $O(n^{\log n}/B^{\log B})$, where B is the maximum fan-in. In [99], the proposed decomposition of a majority function of n inputs with a fan-in B reduced the complexity to quadratic in n, $O(n^2/B)$. For XORs with sublinear complexity in [81], $O(\sqrt{n})$ threshold gates are needed. The total complexity of an XOR with a bounded fan-in is on the order of $O(n^{2.5})$ using the decomposition in [99].

Alternately one could use generic threshold synthesis approaches with arbitrary fan-ins proposed for $\mathbf{F}_{n,m}$ (a class of functions of n inputs with m groups of ones in their truth table) [100, 101] or symmetric Boolean functions [98], since XORs are both a class of $\mathbf{F}_{n,m}$ functions and symmetric functions. However, we show that

when applied to XORs, these approaches result in much higher complexities than our approach, and hence are ineffective. For instance, the number of groups of ones in the truth table of an n-input XOR increases exponentially and the total complexity via the $\mathbf{F}_{n,m}$ approach with a bounded fan-in B is on the order of $O(n2^n/B)$. The total number of gates required via the sort-and-search approach in [98] is on the order of $O(n \log^2 n)$. For all previous approaches based on decomposition and synthesis, the complexity of n-input XORs via the sort-and-search [98] is much smaller than other approaches. However, the sort-and-search algorithm in [98] assumed a fan-in of two. Due to the regular structure of XORs, one could decompose a large XOR into a tree of smaller XORs with bounded fan-ins. In our work, we present tree implementations of XORs by using TLG implementation of multi-input XORs in [79, 91, 92] as primitives. We treat the fan-in as a parameter for the architecture of multi-input XORs, which satisfies the fan-in requirement by design. Regardless of the fan-in requirement, the complexity of n-input XORs of our design is linear with n.

In this chapter we aim to address the class of architectures over finite fields GF(2^m). In particular, we study the multiplication architectures using threshold gates with a given fan-in bound. The work in this chapter presents two main results. The first main result of the manuscript is TLG implementation of multi-input XORs with bounded fan-ins. To leverage the power of multi-input threshold function, we first generalize the sort-and-search algorithm in [98] from fan-in of two to arbitrary fan-ins, and propose an architecture of multi-input XORs with finite fan-ins. We use the XORs in [81,91,92] as primitives and propose two classes of tree implementations of multi-input XORs with finite fan-ins and compare them with the XORs via the

sort-and-search algorithm.

The other main contribution of our manuscript is multiplication over finite fields. Using our proposed multi-input XORs, we then develop two efficient threshold architectures for multiplication in GF(2^m) with a given fan-in bound. Many other GF(2^m) architectures such as those for division and inversion are based on multiplications [102]. Yet, to the best of our knowledge, this is the first work on the implementation of characteristic-2 multiplication in threshold logic. We investigate two types of bit-parallel multiplications, the polynomial basis multiplication [103, 104] and the Massey-Omura (MO) multiplication [105]. We propose efficient implementations of both of these using multi-input XORs and obtain analytical expressions for the gate area and the latency of our designs. These are compared with the architectures synthesized by approaches in [25, 106]. While the synthesis approach in [2,Theorem 12.2.1.2] is chosen for its simplicity, the work in [106] is the first comprehensive synthesis methodology and provides a tool for general multilevel threshold logic design. Our results show that our custom-designed multipliers outperform those synthesized via generic approaches in [25, 106].

The rest of the chapter is organized as following. In Sec. 7.2, we introduce threshold logic and show a typical threshold gate implementation using resonant tunneling diodes (RTDs). Sec. 6.3 generalizes the sort-and-search approach to arbitrary fan-ins and presents a threshold architecture for multi-input XORs. Sec. 6.4 presents our tree architectures for multi-input XORs and evaluates their performance. Sec. 6.5 provides the new efficient threshold implementations of polynomial basis and normal basis multiplications over GF(2^m). This section also evaluates the gate area, number of interconnects and latency of these designs and compares them

to prior work. Finally Sec. 6.6 presents the conclusions of this work.

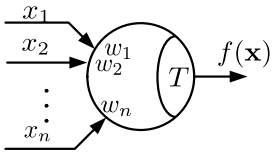


Figure 6.1: Threshold gate realizing $f(\mathbf{x})$ for n inputs, $x_1, x_2, \cdots, x_n$, with corresponding weights $w_1, w_2, \cdots, w_n$ and a threshold T.

6.2 Background

6.2.1 Boolean function

A Boolean function is a function with the mapping $f : \{0,1\}^n \to \{0,1\}$, where n is the number of inputs. Denote the n-input Boolean function by B_n. Boolean functions B_2, such as 2-input AND, OR, and XOR, play an important role in CMOS circuit design. For $x \in \{0,1\}$, the negation of x is denoted by $\neg x$ or $\overline{x}$. Let $x^1 = x$ and $x^0 = \overline{x}$. For $x, y \in \{0,1\}$, the logical conjunction $x \wedge y$ is 1 if and only if $x = y = 1$ and the logical disjunction $x \vee y$ is 1 if and only if $x = 1$ or $y = 1$ [107]. For $\mathbf{x} = (x_1, \cdots, x_n)$, define the minterm m_a for $\mathbf{a} = (a(1), \cdots, a(n)) \in \{0,1\}^n$ by $m_a(x) = x_1^{a(1)} \wedge \cdots \wedge x_n^{a(n)}$. Similarly, define the maxterm s_a for $\mathbf{a} = (a(1), \cdots, a(n)) \in \{0,1\}^n$ by $s_a(x) = x_1^{\neg a(1)} \vee \cdots \vee x_n^{\neg a(n)}$ [107].

An arbitrary n-input Boolean function can be expressed by $f(x) = \bigvee_{a \in f^{-1}(1)} m_a(x) = \bigwedge_{b \in f^{-1}(0)} s_b(x)$, where the first and second representations are called disjunctive and conjunctive normal form (DNF and CNF), respectively [107]. $\{\wedge, \vee, \neg\}$ is called a complete basis [107]. For Boolean functions, we use $\cdot, +, \bar{}$ for $\wedge, \vee, \neg$, respectively, and omit $\cdot$ when there is no ambiguity.

6.2.2 Symmetric function

A Boolean function is said to be symmetric if its output is invariant under any permutation of its input bits. Let f be a symmetric function of n variables, we have

$$f(x_1, x_2, \cdots, x_n) = f(x_{\sigma(1)}, x_{\sigma(2)}, \cdots, x_{\sigma(n)})$$

for all permutations σ of $\{1, \cdots, n\}$. Some Boolean functions, such as n-input AND, OR, NAND, NOR, and XOR, are all symmetric functions. Since any Boolean function has a disjunctive norm form, a symmetric function can be constructed in two levels. The first level is to compute all conjunctions (products) of literals. The second level is to compute the disjunction (sum) of all terms obtained in the first level.

Symmetric functions can also be represented by partially defined Boolean functions if the inputs are sorted first. A partially defined Boolean function has a mapping $f : \{0,1\}^n \rightarrow \{0,1,?\}$, where "?" can be either 0 or 1. Let $\langle x'_1, x'_2, \cdots, x'_n \rangle$ be the sorted sequence of $(x_1, x_2, \cdots, x_n)$. $f_p(x'_1, x'_2, \cdots, x'_n)$ denotes a partially defined Boolean function with inputs only from $n+1$ ordered binary sequences, $(0 \cdots 00)$, $(0 \cdots 01)$, $(0 \cdots 11)$, $\cdots$, $(1 \cdots 11)$. We refer to $f_p(x'_1, x'_2, \cdots, x'_n)$ as a searching function of the symmetric function $f(x_1, x_2, \cdots, x_n)$.

The cost of a polynomial $f(x_1, x_2, \cdots, x_n) = \sum_{i=1}^{m} f_i(x_1, x_2, \cdots, x_n)$ is defined as the sum of costs of all products $f_i(x_1, x_2, \cdots, x_n)$, of which each has a cost equal to the number of its literals. For example, $f(x_1, x_2, x_3) = x_1x_2 + x_1x_3 + x_2x_3$ has a cost of six. A polynomial f_{min} is a minimal polynomial for f, if f_{min} computes f and no other polynomial computing f has a smaller cost than f_{min}. An implicant of f is

a product term f_m satisfying $f_m^{-1}(1) \subseteq f^{-1}(\{1, ?\})$ and $f_m^{-1} \not\subseteq f^{-1}(?)$. An implicant f_p of f is called a prime implicant if no proper sub-term of f_p is an implicant of f. According to Thm. 1.1 in [107], minimal polynomials for f consist only of prime implicants. By computing all prime implicants of f, we can obtain the minimal polynomial for $f(x_1', \cdots, x_n')$.

6.2.3 Threshold logic

A threshold function f with n inputs ($n \geq 1$), $x_1, x_2, \cdots, x_n$, is a Boolean function whose output is determined by [25]

$$f(x_1, x_2, \cdots, x_n) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i x_i \geq T \\ 0 & \text{otherwise,} \end{cases} \tag{6.1}$$

where w_i is called the *weight* of x_i and T the *threshold*. In this chapter we denote this threshold function as $[x_1, x_2, \cdots, x_n; w_1, w_2, \cdots, w_n; T]$, and for simplicity sometimes denote it as $f = [\mathbf{x}; \mathbf{w}; T]$, where $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ and $\mathbf{w} = (w_1, w_2, \cdots, w_n)$.

For the Boolean functions NOT and n-input AND, OR, NAND, and NOR, each corresponds to a **single** threshold function: $[x; -1; 0]$ is the NOT gate, $[\mathbf{x}; 1, 1, \cdots, 1; n]$ and $[\mathbf{x}; 1, 1, \cdots, 1; 1]$ are n-input AND and OR, respectively, $[\mathbf{x}; -1, -1, \cdots, -1; 0]$ and $[\mathbf{x}; -1, -1, \cdots, -1; 1-n]$ equal n-input NAND and NOR, respectively. Unfortunately, an XOR cannot be expressed as a single threshold function.

Certain threshold functions are of particular interest. An n-input threshold function with unit weights and a threshold $\lfloor \frac{n}{2} \rfloor + 1$ is called a majority function. A threshold function with all unit weights but an arbitrary threshold is called a

generalized majority function. Henceforth we denote a generalized n-input majority gate with a threshold k by t_k^n.

The physical entity realizing a threshold function is called a threshold gate. Fig. 7.11 shows a threshold gate realizing (7.5).

6.2.4 RTD Implementation of TG

RTD is a promising (see, e.g., [27]) nanotechnology to replace CMOS. Hence, in this work we focus on RTD implementations of threshold gates. RTD is a diode with resonant tunneling structure. It has a negative differential resistance, i.e., if one increases voltage across it, the current through it initially increases and then drops down to zero again after reaching a certain peak. If two RTDs are tied in series and voltage across them is swept from low to high, then at the end, the RTD with the higher peak current bears all the applied voltage. The peak current depends on the area of the RTD. By replacing each of these RTDs by multiple RTDs in parallel, and selectively adding them into the circuit with input variables, one can change the effective area of the top and bottom RTDs. The voltage at the junction of the two sets of RTDs is thus decided by the comparison of the two sets of areas. This structure, in fact, implements a threshold gate. A typical threshold gate built with RTDs is shown in Fig. 6.2, where a *load* RTD and a *driver* RTD are needed for every output. Let a Boolean function of n variables be realized as a network of k threshold functions $f_i(\mathbf{x}) = [\mathbf{x}^i; \mathbf{w}^i; T_i]$ with $\mathbf{x}^i = (x_1^i, \cdots, x_{n_i}^i)$ and $\mathbf{w}^i = (w_1^i, \cdots, w_{n_i}^i)$ for $i = 1, \cdots, k$, where n_i denotes the number of variables involved in the i-th threshold function. The total gate area of the implementation, composed of areas of all the RTD gates, including the *load* and the *driver* RTDs, is

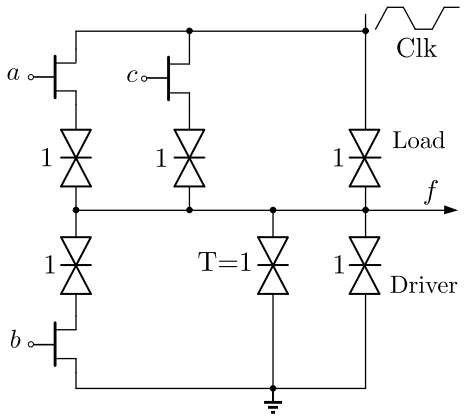


Figure 6.2: RTD implementation of a threshold gate computing $f = a\overline{b} + \overline{b}c + ac = [a, b, c; 1, -1, 1; 1]$. The numerical value next to each RTD is its area.

given by $A(n) = 2k + \sum_{i=1}^{k}(\sum_{j=1}^{n_i} |w_j^i| + |T_i|)$.

The fan-in of a threshold gate in RTD nanotechnology needs to be bounded for both reliability and performance. First, the reliability of an RTD threshold gate decreases sharply with the fan-in due to noise, fluctuation of supply voltage, and manufacture deviations [108]. Second, the switching speed of an RTD implementation of a threshold gate depends on the radio of load to drive peak currents [94]: The closer the ratio is to one, the slower the RTDs switch. Since a gate with more inputs is more likely to have a ratio closer to one than a small gate, it also suggests that the fan-in of an RTD threshold gate be bounded. A maximum fan-in of seven inputs was suggested in [27] for RTDs.

6.3 XOR via Sort-and-Search

In this section, we first propose a sort-and-search algorithm with n-input ($n \geq 2$) threshold gates for implementing any symmetric function. Then, we show an

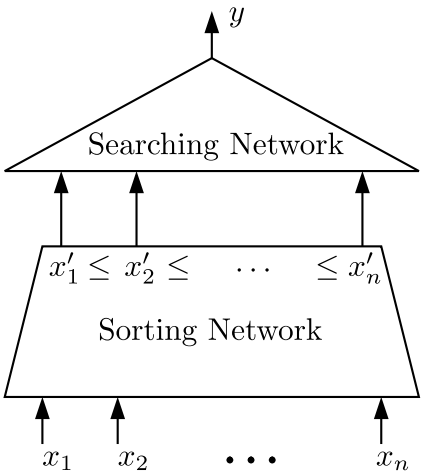


Figure 6.3: Computing a symmetric function via a sort-and-search algorithm [98].

implementation of XOR via the proposed algorithm.

6.3.1 Sort-and-search

Any Boolean function has a two-level implementation. The first level is to compute all product terms and the second level to compute the disjunction of all product terms. For symmetric functions, the evaluation is reduced to comparing the sum of the binary input variables with some constants [98]. A sort-and-search algorithm was proposed in [98] for realizing symmetric functions. It first sorts the inputs and then detects the position in the sorted list switching from zero to one. This sort-and-search algorithm contains two blocks: a sorting block and a searching block as shown in Fig. 6.3. The searching block can be easily implemented as a tree of gates as in [98]. The sorting networks have more complex implementation. Many efficient sorting networks exist in the literature, such as the Batcher's odd-even sort [31] and the bitonic merge sort [109], which use a 2-sorter as the basic block. Binary sorters can be easily implemented in threshold logic. In [98], a 2-by-2 comparator

(2-sorter) was implemented by two threshold gates as shown in Fig. 7.12(a). We use the Knuth diagram in [110] for easy representation of the sorting networks, where switching elements are denoted by connections on a a set of wires. A symbol for a two-sorter is shown in Fig. 7.12(b).

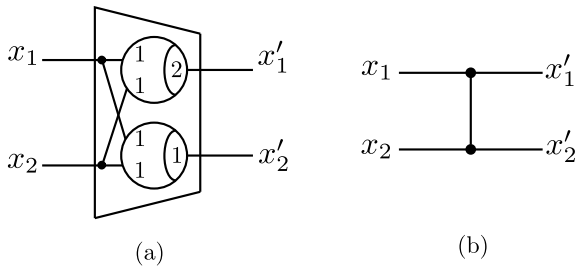


Figure 6.4: Sorters implemented in threshold logic (a) 2-sorter threshold implementation; (b) 2-sorter symbol.

Since XOR is a special symmetric function, it can be implemented via the sort-and-search algorithm in [98]. For instance, a 4-input XOR is shown in Fig. 6.5(a). The corresponding threshold logic implementation is shown in Fig. 6.5(b) with 2-input threshold gates as the basic blocks. The weights and thresholds for all 2-sorters are omitted for simplicity and can be obtained by reviewing Fig. 7.12. The sorting network is simply a 4-input sorting network, where the inputs and outputs are denoted by (x_1, x_2, x_3, x_4) and $\langle x'_1, x'_2, x'_3, x'_4 \rangle$, respectively. For the searching network a truth table with sorted inputs is shown in Table 6.1, where the sorted list x'_1, x'_2, x'_3, x'_4 has only five possible cases and the output y is only true if the inputs have an odd number of 1's. The searching network is composed of two ANDs and one OR as shown in Fig. 6.5(a).

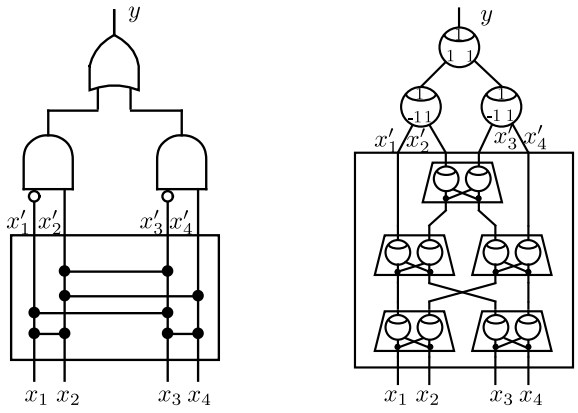


Figure 6.5: A 4-input XOR implemented via the sort-and-search algorithm in [98].

Table 6.1: Truth table for the searching network of a 4-input parity function.

x'_1	x'_2	x'_3	x'_4	y
0	0	0	0	0
0	0	0	1	1
0	0	1	1	0
0	1	1	1	1
1	1	1	1	0

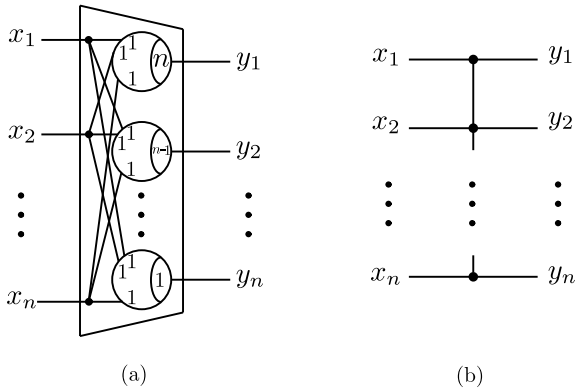


Figure 6.6: Sorters implemented in threshold logic (a) n-sorter threshold implementation; (b) n-sorter symbol.

6.3.2 Generalized sort-and-search

Any symmetric function can be realized by the sort-and-search algorithm in [98]. However, the previously proposed sort-and-search algorithm [98] is based on gates with a fan-in of two. In this work, we generalize the sort-and-search algorithm with fan-in of two in [98] to arbitrary fan-ins. Similar to the basic block, 2-sorter, used in [98], we use n-sorters as the basic blocks. The TLG implementation of n-sorter is shown in Fig. 6.6(a), where n threshold gates are required. A symbol for n-sorter is shown in Fig. 6.6(b). As shown in Fig. 6.6(a), the number of gates of an n-sorter scales linearly with the number of inputs n. For practical concerns, such as reliability, some limit on the fan-in of the basic sorter is assumed. Many sorting networks with n-sorters as building blocks have been proposed in the literature. The multiway merge sort in [111] and the multiway bitonic sort in [109] use n-sorters in part of the sorting network and 2-sorters required for some other parts. Sorting networks with n-sorters ($n \geq 2$) as the basic blocks were proposed in [112, 113]. It has been shown that using larger sorters can reduce the number of gates greatly. In this work, we generalize the sort-and-search algorithm with fan-in of two in [98] to arbitrary fan-ins via our proposed multiway merge sort algorithm in [114].

The searching network is to implement a partially defined function with sorted binary inputs. As explained in Sec. 6.3.1, the truth table for an n-input partially defined function contains $n+1$ output entries corresponding to $n+1$ sorted binary sequences. We denote the $(n+1)$-entry output by $v_f = (v_f(0), v_f(1), \cdots, v_f(n))$. Suppose there are k groups of 1's in v_f, denoted by pairs (b_j, e_j) for $1 \leq j \leq k$, where b_j and e_j are the beginning and ending positions of the j-th group with $0 \leq b_j \leq e_j \leq n$ and $e_{j-1} + 1 < b_j$. We assume $x'_0 = 0$ and $x'_{n+1} = 1$ to deal with

groups of 1's with boundaries on the left and right most positions of v^f. We have the following lemma.

Lemma 6.3.1. *For a partially defined Boolean function f_p of sorted binary inputs $(x'_1, \cdots, x'_n)$, suppose there are k groups of 1's in the $(n+1)$-entry output v_f, denoted by pairs (b_j, e_j) $(1 \leq j \leq k)$. Then, the minimal polynomial is given by $f_{min} = \sum_{j=1}^{k} \overline{x'_{n-e_j}} x'_{n+1-b_j}$ assuming $x'_0 = 0$ and $x'_{n+1} = 1$.*

Proof. We first prove that $\sum_{j=1}^{k} \overline{x'_{n-e_j}} x'_{n+1-b_j}$ is a polynomial for f_{min}. Since $m_j = \overline{x'_{n-e_j}} x'_{n+1-b_j}$ outputs 1 for all inputs corresponding to the j-th group of 1's, m_j is an implicant of f_{min}. For each group of 1's, there is an implicant m_j for f_{min}. Hence, $\sum_{j=1}^{k} m_j$ is a polynomial of f_{min}.

Then we show that $\sum_{j=1}^{k} \overline{x'_{n-e_j}} x'_{n+1-b_j}$ is the minimal polynomial of f_{min}. It is equivalent to prove that all implicants are prime implicants. For $b_j = 1$ or $e_j = n$, the implicant m_j contains only one literal and is already a prime implicant. For $b_j \neq 1$ and $e_j \neq n$, each implicant $m_j = \overline{x'_{n-e_j}} x'_{n+1-b_j}$ has two proper sub-terms, $m_{j_1} = \overline{x'_{n-e_j}}$ and $m_{j_2} = x'_{n+1-b_j}$. Since $m_{j_1}^{-1}(1)$ contains all ordered sequences with $x'_{n-e_j} = 0$, at least one input is not in $f_{min}^{-1}(\{1, ?\})$. Hence, $m_{j_1}^{-1}(1) \not\subseteq f_{min}^{-1}(\{1, ?\})$ for $1 \leq j \leq n$. Similarly, $m_{j_2}^{-1}(1) \not\subseteq f_{min}^{-1}(\{1, ?\})$ for $1 \leq j \leq n$. Thus, all implicants m_j's are prime implicants of f_{min} and $\sum_{j=1}^{k} m_j$ is the minimal polynomial of f_{min}. □

According to Lemma 6.3.1, the searching network is to implement the minimal polynomial $f_{min} = \sum_{j=1}^{k} \overline{x'_{n-e_j}} x'_{n+1-b_j}$. If the fan-in is not constrained, the searching network is simply a two-level tree, with the first level computing all prime implicants of f_{min} and the second level combining all terms as a k-input OR gate. If the fan-in is no more than B, the searching network can be decomposed as a B-ary tree of at most

$\lceil \log_B(2\lfloor \frac{n}{2} \rfloor + 1) \rceil$ levels. The following approaches are used for the decomposition.

If B is even, we apply the following approach.

1. Staring from $j = k$, adjacent $B/2$ implicants m_j's are grouped to form a B-input threshold gate with weight 1 for x'_{n+1-b_j}, -1 for x'_{n-e_j}, and a threshold of 1. If less than $B/2$ implicants are left for small j, combine the left implicants as a smaller threshold gate using the same rule for denoting weights and threshold.

2. Since only one output from the first level is true, outputs from the first level are combined as a B-ary tree of OR gates.

If B is odd, we apply the following approach.

1. Starting from $j = k$, adjacent B literals in m_j's are grouped to form a B-input threshold gate with alternating 1 and -1 as weights starting from the largest literal in the group and a threshold of 1. If less than B literals are left for small j, combine the left literals as a smaller threshold gate using the same rule for denoting weights and threshold.

2. It can be easily shown that outputs from the first level are still an ordered sequence. Repeat step 1) until only one gate is needed for all inputs from last level.

The above proposed sort-and-search algorithm works for any symmetric functions. Since the XOR function is symmetric, we can implement XORs via the generalized sort-and-search algorithm. For an n-input XOR, we first sort the inputs $x_1, x_2, \cdots, x_n$ and obtain $x'_1 \leq x'_2 \leq \cdots \leq x'_n$. According to Lemma 6.3.1,

Table 6.2: A searching function y of ordered binary sequences $(x'_1, x'_2, \cdots, x'_9)$.

x'_1	x'_2	x'_3	x'_4	x'_5	x'_6	x'_7	x'_8	x'_9	y
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	1	1	0
0	0	0	0	0	0	1	1	1	1
0	0	0	0	0	1	1	1	1	0
0	0	0	0	1	1	1	1	1	1
0	0	0	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	1
0	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1

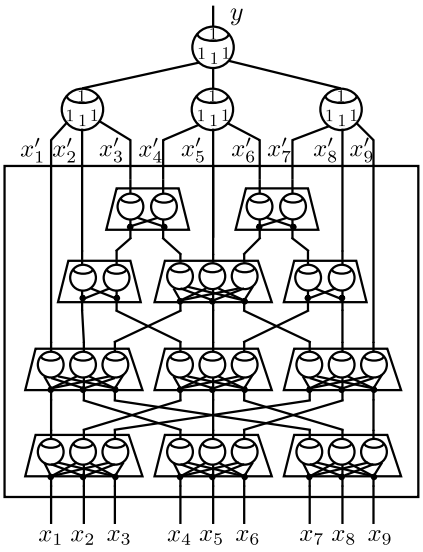


Figure 6.7: A 9-input XOR implemented via the generalized sort-and-search algorithm.

the searching function is $y = x'_1 + \overline{x'_2}x'_3 + \overline{x'_4}x'_5 + \cdots + \overline{x'_{n-1}}x'_n$ for odd n, and $y = \overline{x'_1}x'_2 + \overline{x'_3}x'_4 + \cdots + \overline{x'_{n-1}}x'_n$ for even n. For a 9-input XOR, the partially defined truth table of a searching network y with ordered binary inputs $(x'_1, x'_2, \cdots, x'_9)$ is shown in Table 6.2. According to Lemma 6.3.1, $y = x'_1 + \overline{x'_2}x'_3 + \overline{x'_4}x'_5 + \overline{x'_6}x'_7 + \overline{x'_8}x'_9$.

The whole network for the 9-input XOR is shown in Fig. 6.7, and consists of 33 threshold gates in 6 levels. The weights and thresholds of 2-sorters and 3-sorters in Fig. 6.7 are omitted for simplicity, and can be obtained by reviewing Fig. 6.6.

6.3.3 Analysis of gate area

In the following, we first derive the total number of gates. For $n = B^p$, the latency is given by

$$L_{sort}(B^p) = p + \lceil \tfrac{B}{2} \rceil \times \tfrac{p(p-1)}{2}. \tag{6.2}$$

The total number of gates for the sorting network is given by

$$G_{sort}(B^p) = B^p \cdot L_{sort}(B^p) - G(B^p), \tag{6.3}$$

where $G(B^p) = (p-1)B^{p-2}\frac{B^2+6B-5}{4} + \frac{((p-2)B^{p-1}-(p-1)B^{p-2}+1)(B+5)}{4(B-1)} + \frac{(p-1)(p-2)}{2}B^{p-1}$ for $B \neq 2$ and $G(B^p) = (p^2-p+4)2^{p-1} - 2$ for $B = 2$. The total number of gates for the searching network is given by

$$G_{search}(B^p) = \frac{B^p - 1}{B-1}. \tag{6.4}$$

Hence, the total number of gates for implementing n-XOR is given by

$$G_{s\&s}(n) = \frac{n-1}{B-1} + n \cdot L_{sort}(n) - G(B^p). \tag{6.5}$$

For a fixed fan-in bound B, the area of each gate is bounded by a constant. Hence, the total gate area of an n-input XOR via the generalized sort-and-search approach

is on the order of $O(n \log^2 n)$.

6.4 Tree Implementation of XOR

In Sec. 6.3, we show that the gate area of n-input XORs based on the generalized sort-and-search approach is on the order of $O(n \log^2 n)$. Next, we propose tree implementations of n-input XORs with linear gate area. First, we present an implementation of an n-input XOR as a tree of two-input XORs, each of which is expressed based on other Boolean gates and implemented by their threshold gates. We refer to this approach as *direct conversion* and use it as a basis for comparison. Then, we present a traditional manner of implementing an n-input XOR, and then propose new designs. The former, called the *Boolean class*, expresses the XOR in a two-level NAND circuit implemented through threshold gates. The latter, referred to as the *majority class*, has a two-level implementation and uses only generalized majority gates in the first level. Finally, we analyze the gate area, number of interconnects, and latency of an n-input XOR via these approaches.

6.4.1 Direct conversion

Although an XOR cannot be expressed as a single threshold function, one can first express an XOR based on other Boolean gates and then use their threshold logic implementations. We refer to this approach as direct conversion.

One can implement an n-input XOR through a binary tree of two-input XORs. A two-input XOR $s = a \oplus b$ can be expressed as $s = a\bar{b} + \bar{a}b$, $s = \overline{\overline{\bar{a} + \bar{b}} + \overline{a + b}}$, or $s = \overline{\overline{a\bar{b}}\ \overline{\bar{a}b}}$, and implemented as shown in Fig. 6.8. Among the three implementations in

Fig. 6.8, the NAND-type implementation has the smallest gate area and is therefore chosen in our implementation.

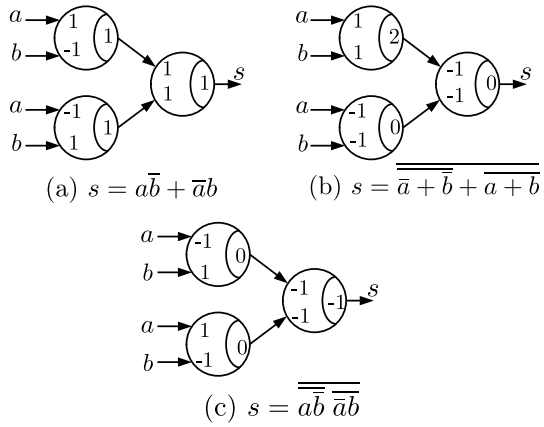


Figure 6.8: Threshold implementation of two-input XOR gate (a) SOP-type (b) NOR-type and (c) NAND-type. All three use threshold gates with a fan-in of only two.

6.4.2 XOR with a small number of inputs

We consider two classes of architectures without considering the maximum fan-in, which is valid when the number of inputs to an XOR is sufficiently small.

Boolean-class implementations

We can use Boolean algebra to express an n-input XOR based on two levels of NANDs. Since NANDs can be implemented as threshold gates, this provides a two-level implementation, called the *Boolean-class* implementation.

Let s denote the XOR of x_1 x_2, ..., x_n. One can express s in a sum-of-product (SOP) form. The NAND implementation of such a form is obtained by using a NAND to combine the literals (a literal is a variable or its inverse) in each product

term and combining the outputs of these NANDs with another NAND. For example, a three-input XOR $s = a \oplus b \oplus c$ can be expressed as $s = \overline{\overline{\bar{a}bc}\ \overline{a\bar{b}c}\ \overline{ab\bar{c}}\ \overline{\bar{a}\bar{b}\bar{c}}}$.

Note that there are 2^{n-1} product terms in the SOP expression of an n-input XOR. Since the last threshold gate has inputs from each of these terms, this implementation is possible only when the maximum fan-in B satisfies $B \geq 2^{n-1}$.

Majority-class implementations

A two-level implementation of n-input XORs is proposed in [91] as shown in Fig. 6.9. All the gates in the first level are generalized majority gates with thresholds from 1 to n. A single gate is in the second level with alternating weights 1 and -1 corresponding to odd and even gates, respectively, in the first level. The gate in the second level combines the outputs from the first level, and compare with a threshold of 1 to compute the n-input XOR. All the threshold gates have the same fan-in n. Thus this two-level implementation in [91] is useful only when $B \geq n$. Another two-level implementation of n-input XORs is proposed in [92] as shown in Fig. 6.10. For an n-input XOR $s = x_1 \oplus x_2 \oplus \cdots \oplus x_n$, let the generalized majority functions t_i^n's with even threshold i's be the intermediate variables. Then, $s = [x_1, x_2, \cdots, x_n, t_2^n, t_4^n, \cdots, t_{2\lfloor \frac{n}{2} \rfloor}^n; 1, 1, \cdots, 1, -2, -2, \cdots, -2; 1]$, which can be implemented in two levels. The first level is to compute how many pairs of ones there are in the inputs. The second level subtracts all pairs of ones from n. The result is either one (odd number of ones) or zero (even number of ones). The threshold gate of the second level has the maximum fan-in amongst all the gates and it equals $\lfloor 3n/2 \rfloor$. Thus this two-level implementation in [92] is useful only when $B \geq \lfloor 3n/2 \rfloor$. The above two two-level implementations require linear number of gates. In [81],

the number of gates for an n-input XOR is reduced to $2\sqrt{n}+O(1)$ in a three-level implementation. There are $\lfloor\sqrt{n}\rfloor$ gates in the first level, $2l$ ($2l \leq \lceil n/\lfloor\sqrt{n}\rfloor\rceil$) gates in the second level, and 1 gate in the third level, where l is an integer. The threshold gates of the second level has the maximum fan-in amongst all the gates and it equals $n+2l$. Thus the three-level implementation in [81] is useful only when $B \geq n+2l$.

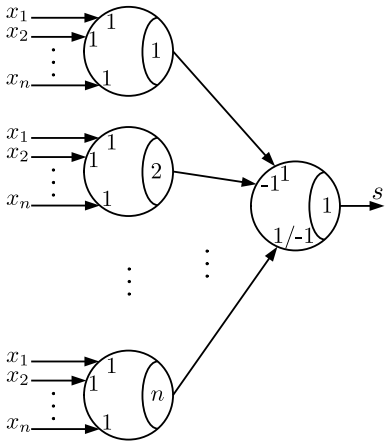


Figure 6.9: Threshold gate implementation of $s = x_1 \oplus x_2 \cdots \oplus x_n$ via [91].

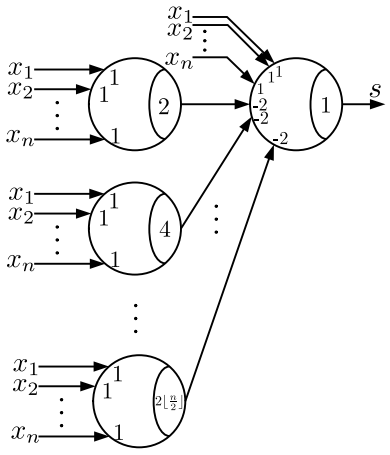


Figure 6.10: Threshold gate implementation of $s = x_1 \oplus x_2 \cdots \oplus x_n$ via [92].

6.4.3 XOR with a large number of inputs

When the number of inputs for an XOR exceeds the maximum fan-in, the implementations in Sec. 6.4.2 cannot be used directly. Instead, an n-input XOR is realized using a tree of B'-input XOR gates as shown in Fig. 6.11. For given B and n, the height of the tree l is minimized while satisfying $n \leq B'^{l}$, and B' is maximized such that no gate in the tree exceeds the maximum fan-in B.

If the Boolean-class implementation is used for the B'-input XORs, then from the results in Sec. 6.4.2, $B \geq 2^{B'-1}$, which gives $B' = 1 + \lfloor \log_2 B \rfloor$.

Alternatively, we can use the threshold gate implementations of XORs via [81, 91, 92] as basic B'-input XORs. For [91], $B = B'$. For [92], $B \geq \lfloor 3B'/2 \rfloor$, which gives $B' = \lfloor (2B+1)/3 \rfloor$. For [81], $B \geq B' + 2l$, where $2l \leq \lceil B'/\lfloor \sqrt{B'} \rfloor \rceil$. In Table 6.3, we show the gate area ($\mathcal{G}$), number of interconnects ($\mathcal{I}$), and latency ($\mathcal{L}$) of XORs via [81, 91, 92] for fan-ins $B = 3, 4, 5, 6, 7$. For the same B in Table 6.3, the XOR via [92] has the smallest gate area, number of interconnects, and latency. In the following, we refer to XORs via [92] as majority-class XORs and use them as primitives for our tree implementation.

6.4.4 Complexity of multi-input XOR

We now investigate the gate area, number of interconnects, and latency of various designs presented in Secs. 6.4.2 and 6.4.3. We treat the gate area, number of interconnects, and latency of an XOR as a function of two parameters: the number of inputs n and the fan-in bound B. When the fan-in bound is not violated, the XOR is simply implemented in a two-level structure. When the fan-in bound is violated, the XOR is decomposed into a tree of smaller XORs such that all smaller

Table 6.3: Comparison of XORs via [81, 91, 92] for fan-ins $B = 3, 4, 5, 6, 7$.

B	B'	Impl.	$\mathcal{G}$	$\mathcal{I}$	$\mathcal{L}$
3	3	Muroga [91]	27	12	2
	2	Minnick [92]	13	5	2
	2	Siu [81]	29	12	3
4	4	Muroga [91]	41	20	2
	3	Minnick [92]	15	7	2
	2	Siu [81]	29	12	3
5	5	Muroga [91]	58	30	2
	3	Minnick [92]	15	7	2
	3	Siu [81]	44	20	3
6	6	Muroga [91]	78	42	2
	4	Minnick [92]	29	14	2
	4	Siu [81]	48	24	3
7	7	Muroga [91]	101	56	2
	5	Minnick [92]	32	17	2
	5	Siu [81]	94	44	3

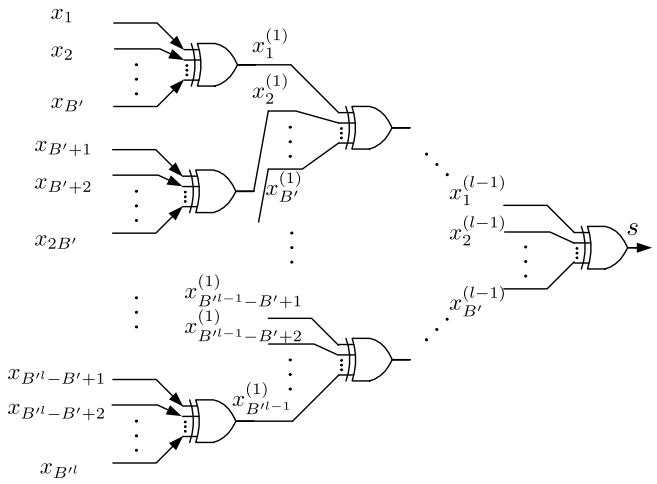


Figure 6.11: n-input XOR realized as a tree with a height of l using B'-input XORs as basic units, where $s = x_1 \oplus x_2 \oplus \cdots \oplus x_n$ for $n = B'^l$.

XORs can be implemented without violating the fan-in bound. For mathematical convenience, assume that $n = B'^l$ for some l, where B' is dependent on the fan-in

bound B. That is, the tree is complete with height $l = \log_{B'} n$. Thus the implementation involves $N_B = (B'^l - 1)/(B' - 1)$ B'-input XORs. For a tree of k threshold gates, the total gate area is simply the sum of all smaller XORs, which is given by $A_{\text{XOR}}(n, B) = 2k + \sum_{i=1}^{k}(\sum_{j=1}^{n_i} |w_j^i| + |T_i|)$. The number of interconnects is given by $I_{\text{XOR}}(n, B) = \sum_{i=1}^{k} n_i$. The latency $L_{\text{XOR}}(n, B)$ of the XOR is given by identifying the critical path from the inputs to the output.

For Boolean-class implementations, the gate area, number of interconnects, and latency are given by

$$A_{\text{XOR}}^{\text{BC}}(n, B) = \begin{cases} 3(n+2)2^{n-2} + 1, & n \leq B' \\ \frac{(n-1)(3(\lfloor \log_2 B \rfloor + 3)2^{\lfloor \log_2 B \rfloor - 1} + 1)}{\lfloor \log_2 B \rfloor}, & n > B' \end{cases}$$

$$I_{\text{XOR}}^{\text{BC}}(n, B) = \begin{cases} (n+1)2^{n-1}, & n \leq B' \\ \frac{(n-1)((\lfloor \log_2 B \rfloor + 2)2^{\lfloor \log_2 B \rfloor})}{\lfloor \log_2 B \rfloor}, & n > B' \end{cases}$$

$$L_{\text{XOR}}^{\text{BC}}(n, B) = \begin{cases} 2, & n \leq B' \\ 2\lceil \log_{\lfloor 1 + \log_2 B \rfloor} n \rceil, & n > B' \end{cases}$$

respectively, where $B' = 1 + \lfloor \log_2 B \rfloor$, which determines the fan-in violation condition for the Boolean-class implementation.

For majority-class implementations, the gate area, number of interconnects, and latency are given by

$$A_{\text{XOR}}^{\text{MC}}(n, B) = \begin{cases} \lfloor n/2 \rfloor \lfloor (3n + 10)/2 \rfloor + n + 3, & n \leq B' \\ \frac{(n-1)(\lfloor \frac{B}{3} \rfloor^2 + \lfloor \frac{2B+16}{3} \rfloor \lfloor \frac{B}{3} \rfloor + \lfloor \frac{2B+10}{3} \rfloor)}{\lfloor \frac{2B-2}{3} \rfloor}, & n > B' \end{cases}$$

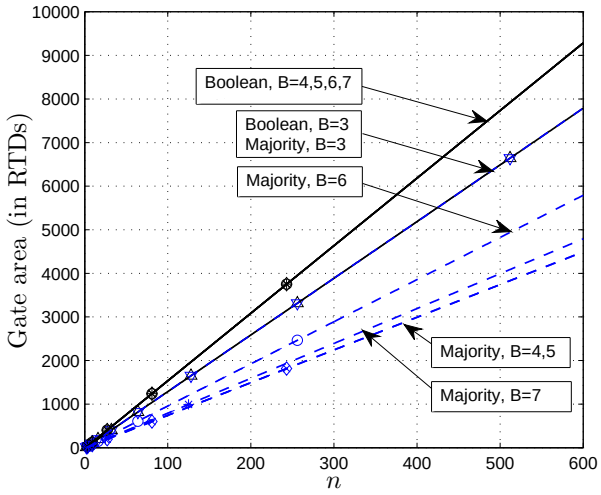


Figure 6.12: Gate area of n-input XOR using threshold gate with fan-in bound B.

$$I_{\text{XOR}}^{\text{MC}}(n,B) = \begin{cases} (n+1)\lfloor n/2 \rfloor + n, & n \leq B' \\ \frac{(n-1)(\lfloor \frac{2B+4}{3} \rfloor \lfloor \frac{B}{3} \rfloor + \lfloor \frac{2B+1}{3} \rfloor)}{\lfloor \frac{2B-2}{3} \rfloor}, & n > B' \end{cases}$$

$$L_{\text{XOR}}^{\text{MC}}(n,B) = \begin{cases} 2, & n \leq B' \\ 2\lceil \log_{\lfloor (2B+1)/3 \rfloor} n \rceil, & n > B' \end{cases}$$

respectively, where $B' = \lfloor (2B+1)/3 \rfloor$, which determines the fan-in violation condition for the majority-class implementation.

The gate area, number of interconnects, and latency of the three classes of XORs are compared in Figs. 6.12, 6.13, and 6.14, respectively, for $B = 3, 4, \cdots, 7$. Though the closed-form expressions of the gate area, number of gates, and latency for tree implementations are derived for complete tree with $n = B''$, we assume the expressions are valid for all n. According to [115], the operand size n is as large as 521. Hence, all the curves in Figs. 6.12, 6.13, and 6.14 are depicted for n up to 600. In Figs. 6.12, 6.13, and 6.14, all the legends correspond to the discrete values $n = B''$

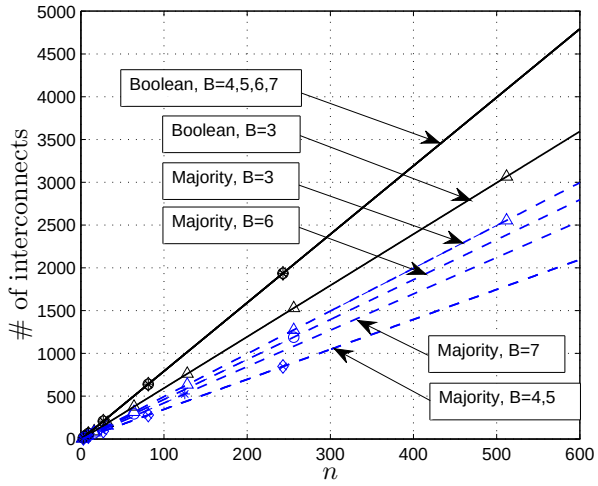


Figure 6.13: Number of interconnects of n-input XOR using threshold gate with fan-in bound B.

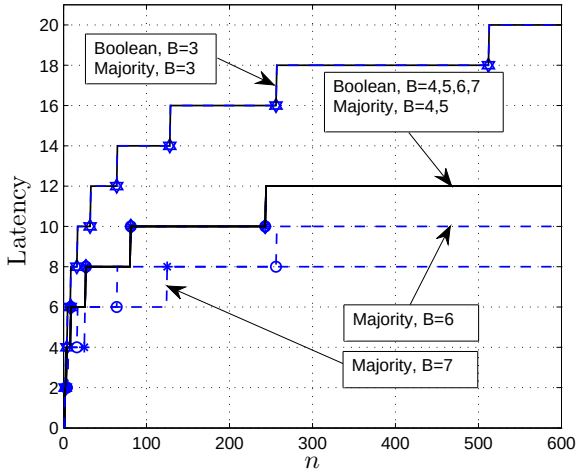


Figure 6.14: Latency of n-input XOR using threshold gate with fan-in bound B.

for $l = 1, 2, \cdots$. For $n \neq B^l$, a multi-input XOR is realized as a partial tree of B'-input XORs. Assume the gate area and number of interconnects of the partial tree of B'-input XORs increases linearly with n. All the curves corresponding to the gate area and number of interconnects are straight lines. In contrast, the curves

corresponding to the latency are step functions, which is due to the ceiling function. All three classes of XORs have linear complexity. The direct conversion has the same gate area, number of interconnects, and latency as the Boolean class with $B = 3$, which implies that the Boolean class includes the direct conversion as a special case and hence provides more tradeoffs between the gate area, number of interconnects, and latency. From Fig. 6.12, both the Boolean-class and majority-class XORs have the same gate area when $B = 3$, the majority-class XOR is more area efficient than the Boolean-class XOR when $B = 4, 5, 6, 7$. From Fig. 6.13, the number of interconnects of the majority-class XOR is smaller than that of the Boolean-class XOR for any B. For example, when $B = 7$, the gate area and number of interconnects of Boolean-class XOR are about twice of those of majority-class XOR with the same number of inputs. For any given B, the latency of the majority-class XOR is smaller than that of the Boolean-class XOR.

It is observed that the optimum fan-in with respect to the gate area is $B = 3$ for Boolean-class, and $B = 4, 5$ for majority-class XORs. For a large maximum fan-in B, the overall gate area and number of interconnects are not the smallest, though the tree is composed of fewer gates, each of which is more powerful dealing with multiple inputs. It implies the majority-class implementation of a multi-input XOR is more efficient in terms of gate area and number of interconnects when $B = 4, 5$, even if a greater fan-in is available. The optimum fan-in can also be explained through the expressions of $A_{\text{XOR}}(n, B)$. For an n-input XOR implemented as a tree, there is a tradeoff between the number of B'-input XORs, N_B, and the gate area of each XOR. A small B leads to a smaller B'-input XOR but a larger N_B. There exists an optimum value B for a fixed n such that the overall gate area is minimized.

When the cost is the main concern, we search all B's for the minimum gate area of $A^{\mathrm{BC}}_{\mathrm{XOR}}(n, B)$ and $A^{\mathrm{MC}}_{\mathrm{XOR}}(n, B)$. The following lemma gives the optimal fan-ins for Boolean-class and majority-class XORs implemented as a tree.

Lemma 6.4.1. *For a given n ($n \geq 3$), among all values of B such that $B \leq 2^{n-1}$ and $B \leq \lfloor \frac{3n}{2} \rfloor$ for Boolean class and majority class, respectively, $B = 3$ and B=4 (or 5) minimize the gate area of the Boolean-class and majority-class implementations of an n-input XOR, respectively.*

Proof. For the Boolean-class implementation, the total gate area of an n-input XOR with the maximum fan-in B is given by $A^{BC}_{\mathrm{XOR}}(n, B) = (n-1) \cdot [3(\lfloor \log_2 B \rfloor + 3)2^{\lfloor \log_2 B \rfloor - 1} + 1] \ / \ \lfloor \log_2 B \rfloor$. For a given n ($n \geq 3$), $A^{BC}_{\mathrm{XOR}}(n, B)$ is a piece-wire function of B. For all $B \leq 2^{n-1}$, $A^{BC}_{\mathrm{XOR}}(n, B)$ is minimized when $B = 3$.

Similarly, for the majority-class implementation of an n-input XOR, the gate area is given by $A^{MC}_{\mathrm{XOR}}(n, B) = (n-1) \cdot (\lfloor \frac{B}{3} \rfloor^2 + \lfloor \frac{2B+16}{3} \rfloor \lfloor \frac{B}{3} \rfloor + \lfloor \frac{2B+10}{3} \rfloor) \ / \ \lfloor \frac{2B-2}{3} \rfloor$. By scanning $B \leq \lfloor \frac{3n}{2} \rfloor$, $A^{MC}_{\mathrm{XOR}}(n, B)$ is minimized when B=4 (or 5). □

6.5 Multiplication over GF(2^m): Threshold Implementation

In CMOS technology, many characteristic-2 field multiplication structures have been proposed using different basis representations of field elements in the literature. Most of them are based on the polynomial basis and normal basis [104]. In the following, we propose threshold architectures of polynomial basis and normal basis multiplications over GF(2^m) using multi-input XORs. Our implementation is based

on the RTD technology, which needs a four-phase clocking scheme. Hence, buffers are inserted to the circuit wherever needed. In our implementation, each output is synthesized as an independent network of threshold gates. No sharing of gates with other outputs is considered. We analyze the gate area, number of interconnects, and latency of our implementations.

6.5.1 Polynomial basis multiplication over GF(2^m)

Polynomial basis is widely used for representing finite field elements. There are two classes of implementations: bit-serial and bit-parallel [103]. The former can be modified to obtain a systolic structure, which has a highly regular structure and less interconnect complexity. However, the structure is not suitable for implementation in the new nanotechnology due to the complex clocking scheme. In contrast, the bit-parallel multiplication can be constructed in a cascaded network, which is suitable for the new nanotechnology.

Let an irreducible polynomial $P(x) = p_0 + p_1 x + \cdots + p_{m-1}x^{m-1} + x^m$ be the generator polynomial of the field GF(2^m). Let $A(x)$ and $B(x)$ be two field elements in GF(2^m) and $C(x)$ be their product modulo $P(x)$. Then,

$$\begin{aligned} C(x) &= A(x)B(x) \bmod P(x) \\ &= (A(x)b_0 \bmod P(x)) + (A(x)xb_1 \bmod P(x)) \\ &\quad + \cdots + (A(x)x^{m-1}b_{m-1} \bmod P(x)) \end{aligned} \tag{6.6}$$

Representing $C(x)$ and $B(x)$ as vectors, the multiplication can be rewritten in matrix

form as $\mathbf{C} = \mathbf{ZB}$

$$\begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{m-1} \end{pmatrix} = \begin{pmatrix} z_{0,0} & z_{0,1} & \cdots & z_{0,m-1} \\ z_{1,0} & z_{1,1} & \cdots & z_{1,m-1} \\ \vdots & \vdots & \ddots & \vdots \\ z_{m-1,0} & z_{m-1,1} & \cdots & z_{m-1,m-1} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{m-1} \end{pmatrix}, \tag{6.7}$$

where $\mathbf{Z}$ is called a product matrix. The i-th column of $\mathbf{Z}$ is obtained by $A(x)x^i \mod P(x)$.

The matrix-vector multiplication in (6.7) requires m^2 two-input AND gates and m m-input XOR gates. The complexity of computing $\mathbf{Z}$ depends on the selected generator polynomial $P(x)$. The choice of the generator polynomials may reduce the arithmetic complexity over GF(2^m). Trinomials, pentanomials, equally-spaced polynomials (ESPs), and all-one polynomials (AOPs) are usually considered for selecting generator polynomials [104]. It has been shown that roughly one half of characteristic-2 fields GF(2^m) for $2 \leq m \leq 1000$ has a trinomial generator [103]. For fields without trinomial generator, a prime pentanomial exists with very high probability [103].

6.5.2 Implementation of polynomial basis multiplication using multi-input XORs

Based on polynomial basis, an m-bit multiplication $C(x) = A(x)B(x) \pmod{P(x)}$ over GF(2^m) is implemented as $\mathbf{C} = \mathbf{ZB}$, where $\mathbf{C} = (c_0, c_1, \cdots, c_{m-1})^T$ and $\mathbf{B} = (b_0, b_1, \cdots, b_{m-1})^T$. It needs two steps: product matrix $\mathbf{Z}$ computation and vector

multiplication. For a generator polynomial $P(x) = x^m + x + 1$, the product matrix $\mathbf{Z}$ is given by

$$\mathbf{Z} = \begin{pmatrix} a_0 & a_{m-1} & a_{m-2} & \cdots & a_2 & a_1 \\ a_1 & a_0+a_{m-1} & a_{m-1}+a_{m-2} & \cdots & a_3+a_2 & a_2+a_1 \\ a_2 & a_1 & a_0+a_{m-1} & \cdots & a_4+a_3 & a_3+a_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m-1} & a_{m-2} & a_{m-3} & \cdots & a_1 & a_0+a_{m-1} \end{pmatrix},$$

where column i is given by the coefficients of $A(x)x^i$ mod $P(x)$ for $i = 0, 1, \cdots, m-1$. Hence, row 0 of $\mathbf{Z}$ is given by $(a_0, a_{m-1}, a_{m-2}, \cdots, a_1)$ and row i for $i = 1, 2, \cdots, m-1$ given by $(a_i, a_{i-1}, \cdots, a_1, a_0 + a_{m-1}, a_{m-1} + a_{m-2}, \cdots, a_{i+1} + a_i)$, where all indices are modulo m. An m-bit polynomial basis multiplication contains m independent blocks in parallel, each of which computes c_i for $i = 0, 1, \cdots, m-1$ simultaneously. The multiplication structure for computing $C(x) = A(x)B(x)$ mod $P(x)$ is shown in Fig. 6.15(a), where the block u_i for computing c_i is shown in Fig. 6.15(b). The generation of row i of $\mathbf{Z}$, given by $A(x)x^i$ mod $P(x)$, is shown in Fig. 6.15(c) and requires $m - i$ $(i = 1, \cdots, m-1)$ two-input XORs. Each block u_i requires m two-input ANDs, $m - i$ two-input XORs, and one m-input XOR. Hence, the total numbers of two-input ANDs, two-input XORs, and m-input XORs for an m-bit multiplication are given by m^2, $\frac{m(m-1)}{2}$, and m, respectively.

6.5.3 Normal basis multiplication over GF(2^m)

Similar to the implementations of polynomial basis multiplications, there are two classes of implementations of normal basis multiplications: the bit-serial and the bit-parallel [103]. The latter can be easily obtained by putting multiple identical blocks in parallel, each of which is the same as that in the former, with cyclic shifted versions of inputs. Bit-parallel implementations achieve much higher throughput at

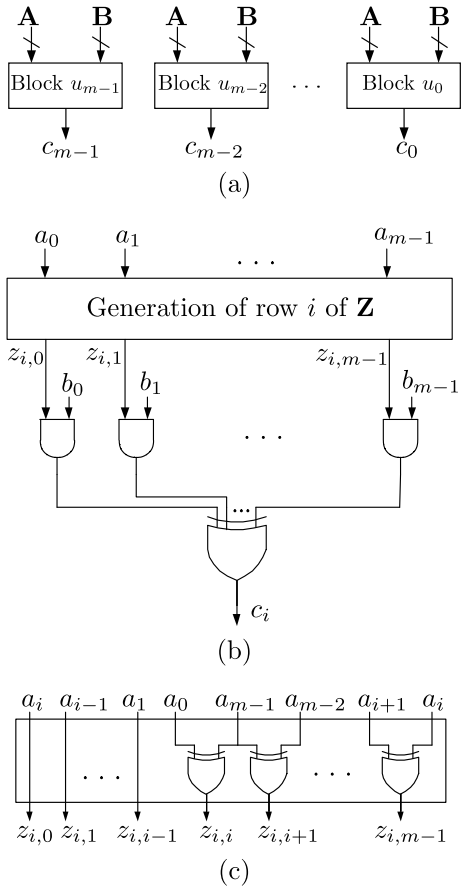


Figure 6.15: Implementation of polynomial basis multiplication (a) Structure for PB multiplication $C(x) = A(x)B(x) \bmod P(x)$; (b) Block u_i for computing c_i; (c) Generation of row i of $\mathbf{Z}$.

the expense of larger gate area.

Some characteristic-2 field operations in normal basis can be implemented efficiently. For instance, the squaring of an element in GF(2^m) is simply given by a cyclic shift. With this property, multiplications in normal basis can be implemented via an algorithm given by Massey and Omura in [105]. The implemented multiplications are referred to as Massey-Omura (MO) multiplications.

Suppose $\beta \in \mathrm{GF}(2^m)$ so that $\{\beta, \beta^2, \cdots, \beta^{2^{m-1}}\}$ forms a normal basis of GF(2^m).

Let $A' = a'_0\beta + a'_1\beta^2 + \cdots a'_{m-1}\beta^{2^{m-1}}$ be any element in GF(2^m). Denote the vector form of A' by $\mathbf{A}' = (a'_0, a'_1, \cdots, a'_{m-1})^T$. Then, $\mathbf{A}'^2$ is a right cyclic shift of $\mathbf{A}'$, $(a'_{m-1}, a'_0, \cdots, a'_{m-2})^T$. Let B' be any element in GF(2^m) with vector form $\mathbf{B}' = (b'_0, b'_1, \cdots, b'_{m-1})^T$, and $C' = A'B'$ with vector form $\mathbf{C}' = (c'_0, c'_1, \cdots, c'_{m-1})^T$ the product with respect to the same normal basis. Then, the last coefficient c'_{m-1} is some function y of coefficients of $\mathbf{A}'$ and $\mathbf{B}'$, $c'_{m-1} = y(\mathbf{A}', \mathbf{B}')$. The i-th coefficient of C' is given by $c'_i = y(\mathbf{A}'^{(m-1-i)}, \mathbf{B}'^{(m-1-i)})$, where $\mathbf{A}'^{(j)} = (a'_{m-j}, a'_{m-j+1}, \cdots, a'_{m-1}, a'_0, a'_1, \cdots, a'_{m-j-1})^T$ denotes the j-fold right cyclic shift of $\mathbf{A}'$. The y function is implemented in the matrix form, $y(\mathbf{A}', \mathbf{B}') = \mathbf{A}'^T\mathbf{H}\mathbf{B}'$, where $\mathbf{H} = [h_{ij}]_{m\times m}$ is a binary matrix.

For an irreducible polynomial $P(x) = 1+x+x^2+x^3+x^4$, let β be one of its roots. The set $\{1, \beta, \beta^2, \beta^3\}$ forms a polynomial basis of GF(2^4). It can be shown that the set $\{\beta, \beta^2, \beta^4, \beta^8\}$ is linearly independent and forms a normal basis of GF(2^4). The two bases are related in the following form:

$$\begin{pmatrix} 1 \\ \beta \\ \beta^2 \\ \beta^3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \beta \\ \beta^2 \\ \beta^4 \\ \beta^8 \end{pmatrix}. \tag{6.8}$$

Then $C' = A'B' = \sum_{i=0}^{3}\sum_{j=0}^{3} a'_i b'_j \beta^{2^i+2^j}$. Let $\beta^{2^i+2^j} = \sum_{l=0}^{3} \lambda_l(i,j)\beta^{2^l}$, where $\lambda_l(i,j)$ denotes the coefficient of β^{2^l} corresponding to the normal basis representation of $\beta^{2^i+2^j}$. Then, we have $c'_l = \sum_{i=0}^{3}\sum_{j=0}^{3} \lambda_l(i,j)\ a'_i b'_j$. Hence, $c_3 = y(\mathbf{A}', \mathbf{B}') = \mathbf{A}'^T\mathbf{H}\mathbf{B}' = \begin{pmatrix} a'_0 & a'_1 & a'_2 & a'_3 \end{pmatrix} \begin{pmatrix} 0&1&1&0 \\ 1&0&0&1 \\ 1&0&1&0 \\ 0&1&0&0 \end{pmatrix} \begin{pmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \end{pmatrix} = a'_0b'_1 + a'_0b'_2 + a'_1b'_0 + a'_1b'_3 + a'_2b'_0 + a'_2b'_2 + a'_3b'_1$.

Assume $\mathbf{H}$ contains N_m 1's and $h_{i_k j_k} = 1$ for $k = 1, 2, \cdots, N_m$. The y function implements additions of N_m terms, $\sum_{k=1}^{N_m} a'_{i_k} b'_{j_k}$ over GF(2). Each block y requires N_m two-input ANDs and one N_m-input XOR. For an m-bit bit-parallel multiplication, there are m identical blocks of y in parallel, and the total numbers of two-input ANDs and N_m-input XORs are given by mN_m and m, respectively. If the sharing of the same terms $a'_i b'_j$ among different blocks is allowed, the total number of two-input ANDs is at most m^2, since there are at most m^2 different terms $a'_i b'_j$ for $0 \leq i, j \leq m-1$. The gate area is determined by block y, which depends on the choice of normal basis. It has been shown [116] that $N_m \geq 2m-1$, and a normal basis that achieves the equality is said to be optimal. About 23% of the fields GF(2^m) for $2 \leq m \leq 1200$ have an optimal normal basis [116]. In the above example, $N_m = 2 \times 4 - 1 = 7$, which means that $\{\beta, \beta^2, \beta^4, \beta^8\}$ is an optimal normal basis.

6.5.4 Implementation of normal basis multiplication using multi-input XORs

The normal basis multiplication algorithm is first given by Massey and Omura in [105]. An m-bit MO multiplication contains m blocks in parallel, each of which implements a function y with cyclic shifted versions of inputs. The m-bit MO multiplication $C' = A'B'$ over GF(2^m) computes all c'_i for $i = 0, 1, \cdots, m-1$ simultaneously. The computation of each $c'_i = y(\mathbf{A}', \mathbf{B}')$ needs N_m two-input ANDs and one N_m-input XOR. The multiplication structure for computing $\mathbf{C}'$ is shown in Fig. 6.16(a). Each block y is implemented as shown in Fig. 6.16(b).

For simplicity, we focus on characteristic-2 fields with an optimal normal basis.

Hence, the number of 1's in $\mathbf{H}$ is given by $N_m = 2m - 1$. If the same term $a'_i b'_j$ among different blocks is shared, large gate area is needed for routing the global interconnects. Thus, we assume that no term $a'_i b'_j$ is shared among different blocks. Each block needs $(2m-1)$ two-input ANDs and one $(2m-1)$-input XOR. The total numbers of two-input ANDs and $(2m-1)$-input XORs for an m-bit multiplication are given by $m(2m-1)$ and m, respectively.

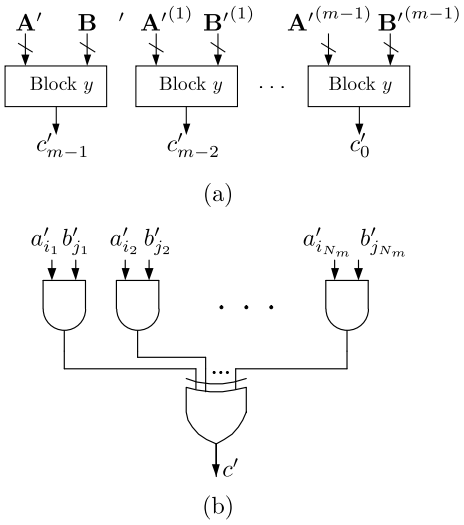


Figure 6.16: Implementation of normal basis multiplication (a) Structure of MO multiplication; (b) Block y for computing c'.

6.5.5 Complexity of multiplication implementations in nanotechnology technology

Since majority-class XOR has smaller gate area complexity than Boolean-class XOR, we use majority-class XORs for implementations of multipliers. The two-input AND can be realized as a single threshold gate with a threshold function given by $f_{2\text{AND}} =$

$$\begin{aligned} A^{\mathrm{MC}}_{\mathrm{PB,Mul}}(m,B) &= m^2 A_{\mathrm{2AND}} + \tfrac{m(m-1)}{2} A^{\mathrm{MC}}_{\mathrm{XOR}}(2,B) + m A^{\mathrm{MC}}_{\mathrm{XOR}}(m,B) \\ &= \begin{cases} m\lfloor \frac{m}{2} \rfloor \lfloor \frac{3m+10}{2} \rfloor + 13.5m^2 - 3.5m, & m \le B' \\ \frac{\lfloor \frac{B}{3} \rfloor^2 + \lfloor \frac{2B+16}{3} \rfloor \cdot \lfloor \frac{B}{3} \rfloor + \lfloor \frac{2B+10}{3} \rfloor}{\lfloor \frac{2B-2}{3} \rfloor}(m^2-m) + 12.5m^2 - 6.5m, & m > B' \end{cases} \end{aligned} \tag{6.9}$$

$$\begin{aligned} I^{\mathrm{MC}}_{\mathrm{PB,Mul}}(m,B) &= m^2 I_{\mathrm{2AND}} + \tfrac{m(m-1)}{2} I^{\mathrm{MC}}_{\mathrm{XOR}}(2,B) + m I^{\mathrm{MC}}_{\mathrm{XOR}}(m,B) \\ &= \begin{cases} (m^2+m)\lfloor \frac{m}{2} \rfloor + 5.5m^2 - 2.5m, & m \le B' \\ \frac{\lfloor \frac{2B+4}{3} \rfloor \lfloor \frac{B}{3} \rfloor + \lfloor \frac{2B+1}{3} \rfloor}{\lfloor \frac{2B-2}{3} \rfloor}(m^2-m) + 4.5m^2 - 2.5m, & m > B' \end{cases} \end{aligned} \tag{6.10}$$

$$\begin{aligned} L^{\mathrm{MC}}_{\mathrm{PB,Mul}}(m,B) &= L_{\mathrm{2AND}} + L^{\mathrm{MC}}_{\mathrm{XOR}}(2,B) + L^{\mathrm{MC}}_{\mathrm{XOR}}(m,B) \\ &= \begin{cases} 5, & m \le B' \\ 3 + 2\lceil \log_{\lfloor \frac{2B+1}{3} \rfloor} m \rceil, & m > B' \end{cases} \end{aligned} \tag{6.11}$$

$[x_1, x_2; 1, 1; 2]$. Denote the gate area, number of interconnects, and latency of a two-input AND by A_{2AND}, I_{2AND}, and L_{2AND}, respectively. Then $A_{\mathrm{2AND}} = 6$, $I_{\mathrm{2AND}} = 2$, and $L_{\mathrm{2AND}} = 1$.

Denote the gate area, number of interconnects, and latency of an m-bit majority-class polynomial basis multiplication by $A^{\mathrm{MC}}_{\mathrm{PB,Mul}}(m,B)$, $I^{\mathrm{MC}}_{\mathrm{PB,Mul}}(m,B)$, and $L^{\mathrm{MC}}_{\mathrm{PB,Mul}}(m,B)$, respectively, where PB denotes polynomial basis. According to Sec. 6.4.4, the gate area, number of interconnects, and latency of an m-input majority-class polynomial basis multiplication are given in Eqs. (6.9)-(6.11), where $B' = \lfloor (2B+1)/3 \rfloor$ determines the fan-in violation condition.

Denote the gate area, number of interconnects, and latency of an m-bit majority-class normal basis multiplication by $A^{\mathrm{MC}}_{\mathrm{NB,Mul}}(m,B)$, $I^{\mathrm{MC}}_{\mathrm{NB,Mul}}(m,B)$, and $L^{\mathrm{MC}}_{\mathrm{NB,Mul}}(m,B)$, respectively, where NB denotes normal basis. According to Sec. 6.4.4, the gate area, number of interconnects, and latency of an m-input majority-class polynomial basis multiplication are given in Eqs. (6.12)-(6.14), where $B' = \lfloor (2B+1)/3 \rfloor$ determines the fan-in violation condition.

$$\begin{aligned} A^{\mathrm{MC}}_{\mathrm{NB,Mul}}(m,B) &= m(2m-1)A_{\mathrm{2AND}} + mA^{\mathrm{MC}}_{\mathrm{XOR}}(2m-1,B) \\ &= \begin{cases} 3m^3+14m^2-7m, & m \le (B'+1)/2 \\ \frac{\lfloor \frac{B}{3} \rfloor^2 + \lfloor \frac{2B+16}{3} \rfloor \cdot \lfloor \frac{B}{3} \rfloor + \lfloor \frac{2B+10}{3} \rfloor}{\lfloor \frac{2B-2}{3} \rfloor}(2m^2-2m) & \\ +12m^2-6m, & m > (B'+1)/2 \end{cases} \end{aligned} \tag{6.12}$$

$$\begin{aligned} I^{\mathrm{MC}}_{\mathrm{NB,Mul}}(m,B) &= m(2m-1)I_{\mathrm{2AND}} + mI^{\mathrm{MC}}_{\mathrm{XOR}}(2m-1,B) \\ &= \begin{cases} 2m^3+4m^2-3m, & m \le (B'+1)/2 \\ \frac{\lfloor \frac{2B+4}{3} \rfloor \lfloor \frac{B}{3} \rfloor + \lfloor \frac{2B+1}{3} \rfloor}{\lfloor \frac{2B-2}{3} \rfloor}(2m^2-2m) + 4m^2 - 2m, & m > (B'+1)/2 \end{cases} \end{aligned} \tag{6.13}$$

$$\begin{aligned} L^{\mathrm{MC}}_{\mathrm{NB,Mul}}(m,B) &= L_{\mathrm{2AND}} + L^{\mathrm{MC}}_{\mathrm{XOR}}(2m-1,B) \\ &= \begin{cases} 3, & m \le (B'+1)/2 \\ 1+2\lceil \log_{\lfloor \frac{2B+1}{3} \rfloor}(2m-1) \rceil, & m > (B'+1)/2 \end{cases} \end{aligned} \tag{6.14}$$

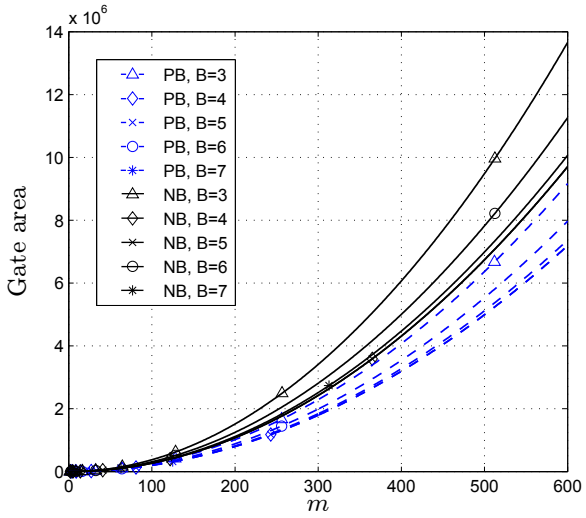


Figure 6.17: Gate area for polynomial basis and normal basis multiplications over GF(2^m).

The gate area, number of interconnects, and latency of both PB and NB implementations are illustrated for comparison with respect to different maximum fan-in B, respectively, in Figs. 6.17, 6.18, and 6.19. Though the closed-form expressions in Eqs. (6.9)-(6.14) are derived for some discrete values $m = B'^l$ for $l = 1, 2, \cdots$,

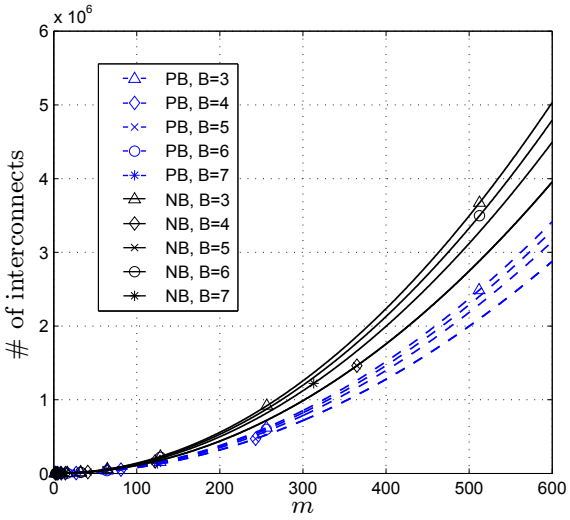


Figure 6.18: Number of interconnects for polynomial basis and normal basis multiplications over GF(2^m).

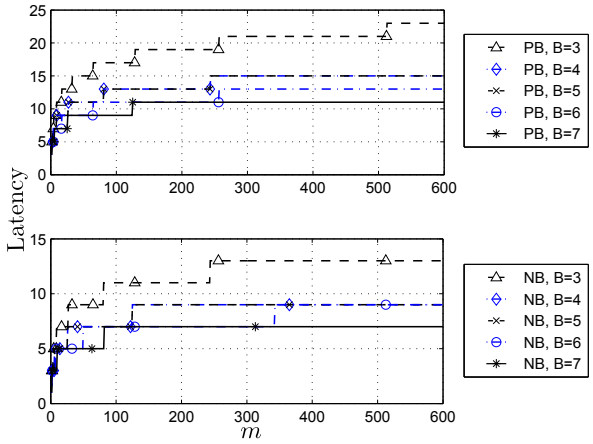


Figure 6.19: Latency for polynomial basis and normal basis multiplications over GF(2^m).

we assume the expressions are valid for all m with $B \leq \lfloor \frac{3m}{2} \rfloor$ or $B \leq 3m-2$. In Figs. 6.17, 6.18, and 6.19, all the legends correspond to the discrete values $m = B^{rl}$

for $l = 1, 2, \cdots$. For $m \neq B'^l$, a multi-input XOR used in various implementations is realized as a partial tree of B'-input XORs. Assume the gate area and number of interconnects of the partial tree of B'-input XORs increase linearly with m. Since the gate area and number of interconnects of various implementations are dominated by the multi-input XORs, all the curves corresponding to the hardware complexity are smooth. In contrast, the curves corresponding to the latency are step functions, since the latency is a ceiling function of m. Both polynomial basis and normal basis implementations require multi-input XORs. Though polynomial basis implementation needs additional $(m^2 - m)/2$ two-input XORs for computing $\mathbf{Z}$, it has smaller complexity than the normal basis implementation. This is because the gate area and number of interconnects are dominated by the multi-input XORs, and polynomial basis implementation requires m m-input XORs, compared with m $(2m-1)$-input XORs. This is observed in Figs. 6.17 and 6.18, where the complexity of normal basis implementation is about 1.4 times of that of polynomial basis implementation. From Figs. 6.17 and 6.18, the gate area and number of interconnects with respect to $B = 4, 5$ are the smallest for both polynomial basis and normal basis implementations, since the complexities of both implementations are dominated by multi-input XORs, which has the smallest complexities when $B = 4, 5$ for majority class, as explained in Sec. 6.4.4. From Fig. 6.19, the polynomial basis multiplication has a smaller latency than the normal basis multiplication for the same B.

Though the proposed implementations have similar structures to the bit-parallel multiplications in CMOS technology [103], there are two main differences. First, unlike CMOS technology, where a gate with $n > 2$ inputs will be implemented as a tree of two-input gates for smaller area [103], our implementation uses majority-class

Table 6.4: Comparison of our complexities of polynomial basis multipliers over GF(2^m) with those of [25] and [106] with or without preprocessing. NP denotes no preprocessing, and B and A denote preprocessing by running *script.Boolean* and *script.algebraic*, respectively.

m	Impl.	$\mathcal{G}$	$\mathcal{B}$	$\mathcal{A}$	$\mathcal{I}$	$\mathcal{L}$
3	[106], NP	524	104	628	275	5
	[106], B	256	628	884	266	10
	[106], A	242	324	566	183	9
	[25]	515	128	643	263	4
	Ours	138	116	254	83	5
4	[106], NP	4900	1676	5066	1915	6
	[106], B	739	2704	3443	1015	16
	[106], A	714	1004	1718	549	11
	[25]	1916	784	2700	1036	5
	Ours	286	264	550	176	7
5	[106], NP	17240	8732	25972	9715	7
	[106], B	1688	2944	4632	1458	12
	[106], A	1618	2928	4546	1380	11
	[25]	7329	3688	11017	4093	6
	Ours	485	388	873	282	7
6	[106], NP	82199	15084	97283	38926	7
	[106], B	3132	7320	10452	3153	15
	[106], A	3340	7968	11308	3321	16
	[25]	28794	15720	44514	16314	7
	Ours	669	560	1229	401	7

multi-input XORs, which significantly reduce the gate area, number of interconnects, and latency. Second, the operation of RTD nanotechnology needs a four-phase clocking scheme. The output is self-latched and the operation is suited for pipelining by constructing a cascaded network of threshold gates.

Table 6.5: Comparison of our complexities of normal basis multipliers over GF(2^m) with those of [25] and [106] with or without preprocessing. NP denotes no preprocessing, and B and A denote preprocessing by running *script.Boolean* and *script.algebraic*, respectively.

m	Impl.	$\mathcal{G}$	$\mathcal{B}$	$\mathcal{A}$	$\mathcal{I}$	$\mathcal{L}$
3	[106], NP	735	68	803	273	5
	[106], B	385	1128	1513	464	13
	[106], A	330	372	702	232	8
	[25]	831	264	1095	435	6
	Ours	213	84	297	102	5
4	[106], NP	4223	2464	6687	2428	6
	[106], B	1395	9816	11211	3099	27
	[106], A	1250	1804	3054	960	10
	[25]	3892	1824	5716	2148	8
	Ours	348	176	524	184	5
5	[106], NP	20975	9856	30831	11440	7
	[106], B	3960	8152	12112	3816	12
	[106], A	3540	6992	10532	3154	13
	[25]	18425	9880	28305	10405	10
	Ours	570	240	810	290	5
6	[106], NP	101117	14192	115309	46721	7
	[106], B	9155	22040	31195	9466	15
	[106], A	8105	19228	27333	7993	17
	[25]	86766	48624	135390	49398	12
	Ours	912	432	1344	468	7

6.5.6 Comparison with existing approaches

We focus on m-bit polynomial basis and normal basis multiplications and compare the gate area, number of interconnects, and latency of our implementations with those obtained via generic synthesis approaches in [25, 106]. The implementations in [106] are obtained with and without running preprocessing. Two preprocessed Boolean networks are obtained by running two scripts, *script.Boolean*

and *script.algebraic*, respectively. The fan-in $B = 5$ is chosen for our implementation, since the gate area of our implementation is minimized. For implementations in [106], we assume the fan-in $B = 5$. For implementations in [25], the fan-in $B = 3$ is chosen by necessity, since only symmetric three-input majority gates are used. For $m = 3, 4, 5, 6$, the complexities of our proposed multipliers and those synthesized via the approaches in [25, 106] are shown in Tables 6.4 and 6.5 for polynomial basis and normal basis multiplications, respectively, where the gate area $\mathcal{G}$, buffer area $\mathcal{B}$, total gate area $\mathcal{A}$, number of interconnects $\mathcal{I}$, and latency $\mathcal{L}$ are shown in columns 3 through 7, respectively. Buffers are listed separately in Tables 6.4 and 6.5. Each buffer consumes four RTDs and one interconnect. The total gate area is the sum of gate area and buffer area. Due to efficient implementation of multi-input XORs, our implementation requires smaller area than those in [25, 106], as well as fewer interconnects. For a 6-bit multiplier, the total gate area and number of interconnects of our implementations are about one order of magnitude smaller than the best results given in [25, 106]. For a larger multiplier, the complexity saving would be even greater. The advantage of our implementation becomes more significant as m grows. According to the application of elliptical curve in cryptography, m can be as as large as 512 [115]. For $m = 512$, our custom-designed implementation requires much smaller area than others. For normal basis multiplier, the latency of ours is always better than the that in [25]. For polynomial basis multiplier, the latency of ours is slightly greater than the that in [25] for $m = 3, 4, 5$. However, the latency of ours increases logarithmically with m as shown in Eq. 6.11, compared with that in [25], which increases linearly with m. Hence, for a large multiplier, the latency of ours is much smaller than that in [25]. For both polynomial basis and normal basis

multiplications, the implementations without preprocessing in [106] have roughly the same latency as ours, but require much larger gate area. The implementations with preprocessing in [106] have larger latency than ours, as well as larger gate area. Our analysis results show that our implementations of multipliers performs better than existing approaches in [25,106] in terms of overall area complexity and latency.

6.6 Conclusion

This chapter has provided the designs of multipliers over GF(2^m) using threshold gates with bounded fan-in that are suitable for nanotechnology implementation. Fan-in of nanotechnology gates influences their reliability and speed. Thus our designs allow the trade-off between complexity, reliability and speed. A comparison of our implementations of various multiplication architectures shows that they use less gate area and fewer interconnects than those obtained by the approaches available in the literature [25, 106].

Since most designs of GF(2^n) architectures use a large number of XOR gates, we have first focused on efficient designs of multi-input XORs using threshold gates. We have shown that the implementations based on generalized majority gates have smaller area and latency as compared with those that use Boolean algebra approaches. Other architectures over GF(2^m) will also benefit from use of multi-input XORs developed here.

Chapter 7

An Enhanced Multiway Sorting Network Based on n-Sorters

7.1 Introduction

Sorting is one important operation in data processing, and hence its efficiency greatly affects the overall performance of a wide variety of applications [31, 110]. Sorting networks can achieve high throughput rates by performing operations simultaneously. These parallel sorting networks have attracted attention of researchers due to increasing hardware speed and decreasing hardware cost. One of the most popular sorting algorithm is called merge-sort algorithm, which performs the sorting in two steps [31]. First, it divides the input **list** (a sequence of values) into multiple **sublists** (a smaller sequence of values) and sorts each sublist simultaneously. Then, the sorted sublists are merged as a single sorted list. The sorting process of sublists can then be decomposed recursively into the sorting and merging of even smaller

sublists, which are then merged as a single sorted list. Hence, the merging operation is the key procedure for the decomposition-based sorting approach. One popular 2-way merging algorithm called odd-even merging [31] merges two sorted lists (odd and even lists) into one sorted list. In [117], a modulo merge sorting was introduced as a generalization of the odd-even merge by dividing the two sorted input lists into multiple sublists with a modulo not limited to 2. Another popular 2-way merging algorithm is bitonic merging algorithm [109]. Two sorted lists are first arranged as a bitonic list, which is then converted to obtain a sorted list. These 2-way merging algorithms employ 2-way merge procedure recursively and have a capability of sorting N values in $O(\log^2 N)$ stages [31]. In [118], a sorting network, named AKS sorting network, with $O(\log N)$ stages was proposed. However, there is a very large constant in the depth expression, which makes it impractical. Recently, a modular design of high-throughput low-latency sorting units are proposed in [119]. However, the basic building block in these 2-way merging algorithm is a 2-sorter, which is simply a 2×2 switching element or comparator as shown in Fig. 7.1(a).

Instead of using 2-sorters, n-sorters can be used as basic building blocks. This was first proposed as a generalization of the Batcher's odd-even merging algorithm [111]. It was also motivated by the use of n-sorters, which sort n ($n \geq 2$) values in unit time [112, 120]. Since large sorters are used as basic building blocks, the number of sorters as well as the latency is expected to be reduced greatly. An n-way merging algorithm was first proposed by Lee and Batcher [111], where n is not restricted to 2. A version of the bitonic n-way merging algorithm was proposed by Nakatani *et al.* [121, 122]. However, the combining operation in the n-way merging algorithms still use 2-sorters as basic building blocks. Leighton proposed an

algorithm for sorting r lists of c values each, represented as an $r \times c$ matrix [123]. This algorithm is a generalization of the odd-even merge-sort and named columnsort, since it merges all sorted columns to obtain a single sorted list in row order. In the original columnsort, no specific operation was provided for sorting columns and no recursive construction of sorting network was provided. In [112], a modified columnsort algorithm was proposed with sorting networks constructed from n-sorters ($n \geq 2$) [124]. However, a 2-way merge is still used for the merging process. In [113], an n-way merging algorithm, named SS-Mk, based on the modified columnsort was proposed with n-sorters as basic building blocks, where n is prime. For n sorted lists of m values each, the idea is to sort the $m \times n$ values first in each row and then in slope lines with decreasing slope rates. An improved version of the SS-Mk merge sort, called ISS-Mk, was provided in [125], where n can be any integer. We compare our sorting scheme with the SS-Mk but not the ISS-Mk, because for our interested ranges of N, the ISS-Mk requires larger latency due to a large constant.

In this chapter, we propose an n-way merging algorithm, which generalizes the odd-even merge by using n-sorters as basic building blocks, where n (≥ 2) is prime. Based on this merging algorithm, we also propose a sorting algorithm. For $N = n^p$ input values, $p + \lceil n/2 \rceil \times \frac{p(p-1)}{2}$ stages are needed. The complexity of the sorting network is evaluated by the total number of n-sorters. The closed-form expression for the number of sorters is also derived.

Instead of 2-sorters, n-sorters ($n > 2$) are used as basic blocks in this chapter. This is because larger sorters have some efficient implementation. For example, for binary sorting in threshold logic, the area of an n-sorter scales linearly with the number of inputs n, while the latency stays as a constant. Hence, a smaller number

of sorters and latency of the whole sorting network can be achieved. However, we cannot use arbitrary large sorters as basic blocks, since larger sorters are more complex and difficult to be implemented. Hence, the benefit of using a larger block diminishes with increasing n. We assume that the size of basic sorter $n \leq 20$ and 10 when evaluating the number of sorters and latency. Our algorithm works for any upper bound on n, and one can plug any upper bound on n into our algorithm. Asymptotically, the number of sorters required by our sorting algorithm is on the same order of $O(N \log^2 N)$ as the SS-Mk [113] for sorting N inputs. Our sorting algorithm requires fewer sorters than the SS-Mk in [113] in wide ranges of N. For instance, for $n \leq 20$, when $N \leq 1.46 \times 10^4$, our algorithm requires up to 46% fewer sorters than the SS-Mk. When $1.46 \times 10^4 < N \leq 1.3 \times 10^5$, our algorithm has fewer sorters for some segments of N's. When $N > 1.3 \times 10^5$, our algorithm needs more sorters.

The work in this chapter is different from previous works [111, 113, 125] in the following aspects:

- While the multiway merge [111] uses 2-sorters in the combining network, our proposed n-way merging algorithm uses n-sorters as basic building blocks. By using larger sorters ($n > 2$), the number of sorters as well as the latency would be reduced greatly.

- The merge-based sorting algorithms in [113, 125] are based on the modified columnsort [124], which merges sorted columns as a single sorted list in row order. Our n-way merge sorting algorithm is a direct generalization of the multiway merge sorting in [111].

- We analyze the performance of our approach by deriving the closed-form expressions of the latency and the number of sorters. We also derive the closed-form expression of the number of sorters for the SS-Mk [113], since it was not provided in [113]. Then we present extensive comparisons between the latency and the number of sorters required by our approach and the SS-Mk [113].

- Finally, we show an implementation of a binary sorting network in threshold logic. With an implementation of a large sorter in threshold logic, we compare the performance of sorting networks in terms of the number of gates.

The rest of the chapter is organized as following. In Sec. 7.2, we briefly review the background of sorting networks. In Sec. 7.3, we propose a multiway merging algorithm with n-sorters as basic blocks. In Sec. 7.4, we introduce a multiway sorting algorithm based on the proposed merging algorithm, and show extensive results for the comparison of our sorting algorithm and previous works. In Sec. 7.5, we focus on a binary sorting network, where basic sorters are implemented by threshold logic and have complexity linear with the input size, and measure the complexity in terms of number of gates. Finally Sec. 7.6 presents the conclusion of this chapter.

7.2 Background

A sorting network is a feedforward network, which gives a sorted list for unsorted inputs. It is composed of two items: **switching elements (or comparators)** and **wires**. The depth of a comparator is defined to be the longest length from the inputs of the sorting network to that comparator's outputs. The **latency** of the sorting network is the maximum depth of all comparators. The network is

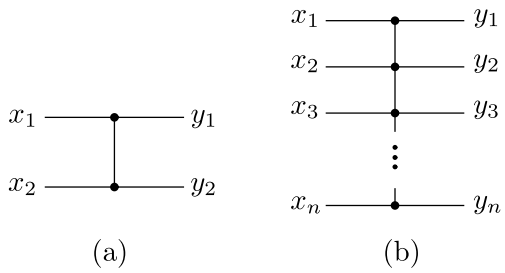


Figure 7.1: (a) 2-sorter ($y_1 \leq y_2$); (b) n-sorter ($y_1 \leq y_2 \leq \cdots \leq y_n$).

oblivious in the sense that the time and location of input and output are fixed ahead of time and not dependent on the values [31]. We use the Knuth diagram in [110] for easy representation of the sorting networks, where switching elements are denoted by connections on a a set of wires. The inputs enter at one side and sorted values are output at the other side, and what remains is how to arrange the switching elements. The sorting network is measured in two aspects, latency (number of stages) and complexity (number of sorters). The basic building block used by the odd-even merge [31] is a 2-by-2 comparator (compare-exchange element). It receives two inputs and outputs the minimum and maximum in an ordered way. The symbol for a 2-sorter is shown in Fig. 7.1(a), where x_i and y_i for $i = 1, 2$ are input and output, respectively. Similarly, an n-sorter is a device sorting n values in unit time. The symbol for an n-sorter is shown in Fig. 7.1(b), where x_i and y_i for $i = 1, 2, \cdots, n$ are input and output, respectively, and the output satisfies $y_1 \leq y_2 \leq \cdots \leq y_n$. In this chapter, we denote the sorted values $y_1 \leq y_2 \leq \cdots \leq y_n$ by $\langle y_1, y_2, \cdots, y_n \rangle$ and use n-sorters as basic blocks for sorting.

Merging-based sorting networks are an important family of sorting networks, where the merging operation is the key. There are two classes of merging algorithms, the odd-even merging [31] and the bitonic merging [109]. The former is an efficient

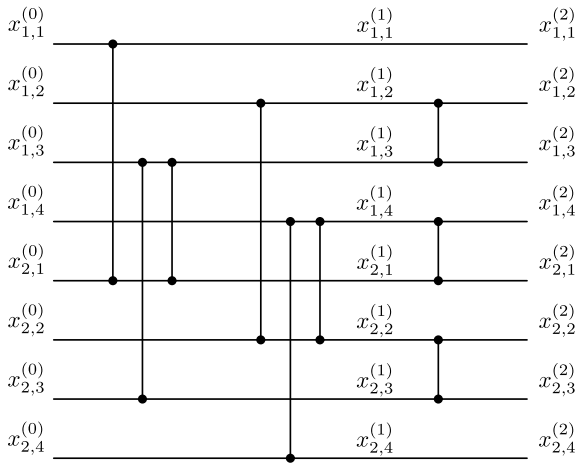


Figure 7.2: The odd-even merge of two sorted lists of 4 values each using 2-sorters.

sorting technique based on the divide-and-conquer approach, which decomposes the inputs into two sublists (odd and even), sorts each sublist, and then merges two sorted lists into one. Further decomposition and merging operations are applied on the sublists. An example of odd-even merging network using 2-sorters is shown in Fig. 7.2, where two sorted lists, $\langle x_{1,1}^{(0)}, \cdots, x_{1,4}^{(0)} \rangle$ and $\langle x_{2,1}^{(0)}, \cdots, x_{2,4}^{(0)} \rangle$, are merged as a single list $\langle x_{1,1}^{(2)}, \cdots, x_{1,4}^{(2)}, x_{2,1}^{(2)}, \cdots, x_{2,4}^{(2)} \rangle$ in two stages.

Instead of merging two lists, multiple sorted lists can be merged as a single sorted list simultaneously. An **n-way merger** ($n \geq 2$) of size m is a network merging n sorted lists of size m (m values) each into a single sorted list in multiple stages. This was first proposed as a generalization of the Batcher's odd-even merging algorithm. It is also motivated by the use of n-sorters, which sort n ($n \geq 2$) values in unit time [112,120]. Since large sorters are used as basic building blocks, the number of sorters as well as the latency is expected to be reduced greatly. Many multiway merging algorithms exist in the literature [111–113, 121–127]. The algorithms in [126, 127] implement multiway merge using 2-sorters. In [111], a generalization of Batcher's

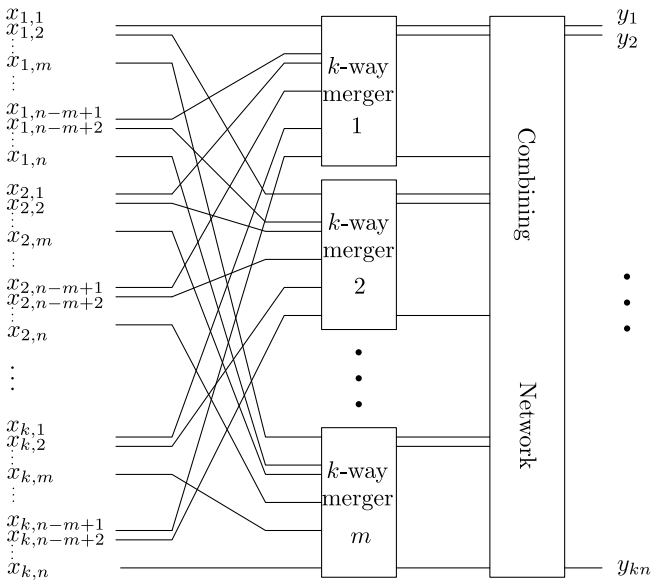


Figure 7.3: Iterative construction rule for the n-way merger [111].

odd-even is introduced as shown in Fig. 7.3, where an n-way merger of n lists of size ud is decomposed into d n-way mergers of n sublists of size u plus a combining network. Each of the small n-way mergers is further decomposed similarly. However, the combining network in the merging network in Fig. 7.3 still uses 2-sorters as basic blocks. In [123], Leighton proposed a columnsort algorithm, which showed how to sort an $m \times n$ matrix denoting the n sorted lists of m values each. A modification of Leighton's columnsort algorithm was given in [112]. In [113, 125], merging networks with n-sorters as basic blocks are introduced based on the modified Leighton's columnsort algorithm.

In this chapter, we focus on multiway merge sort with binary values as inputs. Our merge sort also works for arbitrary values, which is justified by the following theorem.

Theorem 7.2.1 (Zero-one principle [31]). *If a network with n input lines sorts all*

2^n lists of 0s and 1s into nondecreasing order, it will sort any arbitrary list of n values into nondecreasing order.

7.3 Multiway Merging

In the following, we propose an n-way merging algorithm with n-sorters as basic building blocks as shown in Alg. 5. We consider a sorting network, where all iterations of Alg. 5 are simultaneously instantiated (loop unrolling). We refer to the instantiation of iteration i of Alg. 5 as stage i of the sorting network. The sorters in the last for loop in Alg. 5 consist of the last stage. Let the n sorted input lists be $\langle x_{j,1}^{(0)}, x_{j,2}^{(0)}, \cdots x_{j,m}^{(0)} \rangle$ for $j = 1, \cdots, n$. Denote the values of j-th list after stage k by $(x_{j,1}^{(k)}, x_{j,2}^{(k)}, \cdots, x_{j,m}^{(k)})$. After $T = 1 + \lceil \frac{m}{2} \rceil$ stages, all input lists are sorted as a single list, $\langle x_{1,1}^{(T)}, x_{1,2}^{(T)}, \cdots, x_{1,m}^{(T)} \rangle, \langle x_{2,1}^{(T)}, x_{2,2}^{(T)}, \cdots, x_{2,m}^{(T)} \rangle, \cdots, \langle x_{n,1}^{(T)}, x_{n,2}^{(T)}, \cdots, x_{n,m}^{(T)} \rangle$.

For convenience of describing and proving our algorithm, we introduce some notations and definitions. Denote the number of zeros in the j-th list after stage i as $r_j^{(i)}$, where $i = 1, 2, \cdots, \lceil \frac{m}{2} \rceil + 1$ and $j = 1, \cdots, n$. A sorter is called a **k-spaced sorter** if its adjacent inputs span k other wires and each connection of the same sorter comes from different lists of m wires, where $0 \leq k \leq m - 1$. For simplicity, we arrange the sorters in the order of their first connections in each stage. Denote $\{1, 2, \cdots, m\}$ as $\mathbb{Z}_m$. Two k-spaced sorters are said to be **adjacent** if they connect adjacent two wires, $x_{j,k}^{(i)}$ and $x_{j,k+1}^{(i)}$, respectively, for some $j \in \mathbb{Z}_m$ and $k \in \mathbb{Z}_{m-1}$. Then, our n-way merging Alg. 5 can be intuitively understood as flooding lists with zeros in descending order. The correctness of Alg. 5 can be shown by first proving the following lemmas. See the appendix for the proofs of the following lemmas and

Algorithm 5 Algorithm for n-way merging network.

Input: n sorted lists $\langle x_{j,1}^{(0)}, x_{j,2}^{(0)}, \cdots x_{j,m}^{(0)} \rangle$ for $j = 1, \cdots, n$;
  $i = 1$;
  while $i \leq \lceil \frac{m}{2} \rceil$ **do**
    for $j = 1$ to $n - 1$ **do**
      Apply $(m - i)$-spaced sorters between lists j and $j + 1$;
    end for
    Merge all $(m - i)$-spaced sorters;
    Update n sorted lists $\langle x_{j,1}^{(i)}, x_{j,2}^{(i)}, \cdots x_{j,m}^{(i)} \rangle$ for $j = 1, \cdots, n$;
    $i = i + 1$;
  end while
  for $j = 1$ to $n - 1$ **do**
    Apply $(m - 1)$-sorters on $m - 1$ adjacent lines with first half, $x_{j,m-k}^{(i-1)}$, from list j and second half, $x_{j+1,k}^{(i-1)}$, from list $j + 1$, where $k = 1, \cdots, \frac{m-1}{2}$;
  end for
  Output: Sorted lists.

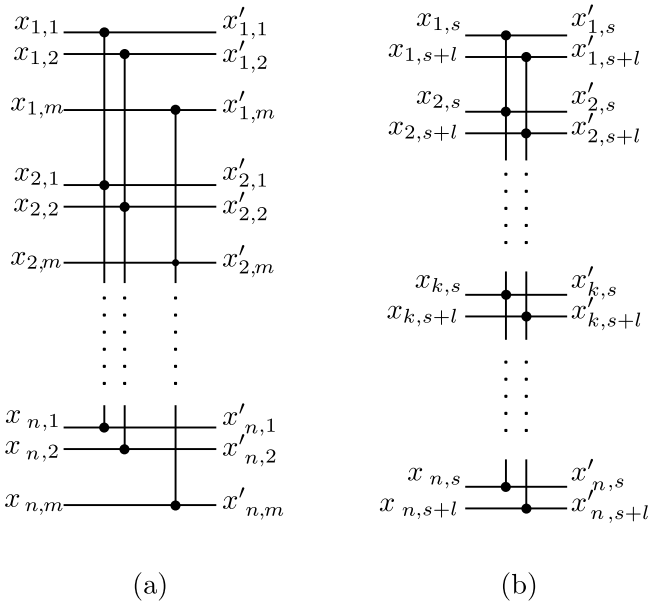


Figure 7.4: The network for n sorted lists of m wires.

theorems.

Lemma 7.3.1. *Apply* $(m-1)$*-spaced sorters to* n *lists of* m *values,* $\langle x_{j,1}, x_{j,2}, \cdots, x_{j,m} \rangle$,

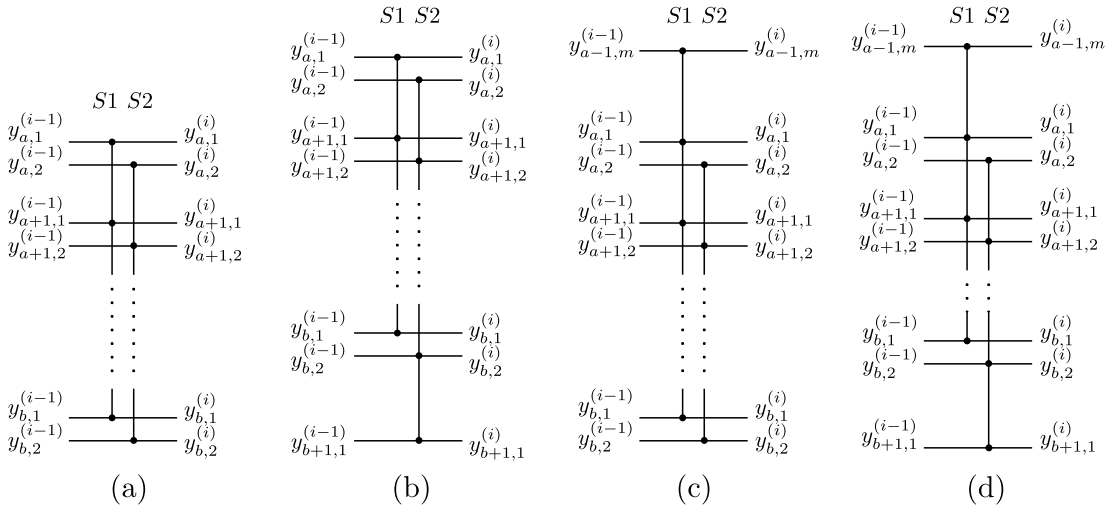


Figure 7.5: Adjacent two sorters $S1$ and $S2$ in each stage of Alg. 5 can be classified into four four cases. (a) Case I ($\Delta = \frac{v-w}{b-a}$); (b) Case II ($\Delta = \frac{v-1}{b-a+1}$); (c) Case III ($\Delta = \frac{m-w+1}{b-a+1}$); (d) Case IV ($\Delta = \frac{m}{b-a+2}$).

for $j = 1, \cdots, n$. *The outputs of each list are still sorted,* $\langle x'_{j,1}, x'_{j,2}, \cdots, x'_{j,m} \rangle$, *for* $j = 1, 2, \cdots, n$.

For n sorted lists of m values, there are m $(m-1)$-spaced sorters as illustrated in Fig. 7.4(a). The proof of the lemma can be reduced to showing that any two wires $s, s+l \in \mathbb{Z}_m$ of each list connected by the s- and $(s+l)$-th sorters are sorted. The simplified network is shown in Fig. 7.4(b). Without lose of generality, we can choose $l = 1$.

Lemma 7.3.2. *In each stage of Alg. 5, there are at most four cases of adjacent two sorters as shown in Fig. 7.5. If* m *is prime, case IV is impossible.*

We first show that the first connections of adjacent two sorters, $S1$ and $S2$, belong to either the same list or adjacent two lists. The same relation is true for the last connections of $S1$ and $S2$. This gives us a total of four cases as shown in

Fig. 7.5, where $b \geq a+1$ for Fig. 7.5(a)-(c), and $b \geq a$ for Fig. 7.5(d) such that $S1$ and $S2$ have a size of at least two.

The following theorem proves the correctness of Alg. 5.

Theorem 7.3.1. *For a prime m in Alg. 5, all lists are self-sorted after every stage. In particular, all lists are sorted after the final stage.*

The theorem can be proved by induction on i.

In Alg. 5, the latency increases linearly with $\lceil \frac{m}{2} \rceil$. When m is large, the latency is also very large. By further decomposing m into a product of small factors, we can reduce the latency significantly. In the following, we propose Alg. 6 for merging n lists of m values, where $m = n^{p-1}$ for $p \geq 2$. When m is not a power of n, we can use a larger network of $m' = n^{p'} > m$ inputs. For any q in stage i ($2 \leq i \leq p-1$), denote the number of zeros in each new formed list after stage i as $r_{j,q}^{(i)}$, where $j = 1, \cdots, n^i$. Assume two dummy lists with $r_{0,q}^{(i)} = n$ and $r_{n^i+1,q}^{(i)} = 0$ are appended to the two ends of n^i lists. The correctness of Alg. 6 can be shown by first proving the following lemma.

Lemma 7.3.3. *In Alg. 6, the new lists in stage i with respect to q are self-sorted. The numbers of zeros of all new lists after stage i are non-increasing,*

$$r_{j,q}^{(i)} \geq r_{j+1,q}^{(i)} \quad \textit{for} \quad j = 1, \cdots, n^i - 1,$$

where $i = 2, \cdots, p-1$ and $q = 1, \cdots, n^{p-1-i}$. Furthermore, there are at most n consecutive lists that have between 1 and $n-1$ zeros,

$$r_{s,q}^{(i)} = n > r_{s+1,q}^{(i)} \geq \cdots \geq r_{s+l,q}^{(i)} > 0 = r_{s+l+1,q}^{(i)} \quad \textit{for} \quad l \leq n,$$

Algorithm 6 Algorithm for combining n lists of $m = n^{p-1}$ values.

Input: n sorted lists $\langle x_{j,1}^{(0)}, x_{j,2}^{(0)}, \cdots x_{j,m}^{(0)} \rangle$ for $j = 1, \cdots, n$ and $m = n^{p-1}$;
$i = 1$;
for $q = 1$ to n^{p-2} **do**
Apply Alg. 5 on $\langle x_{j,q}^{(0)}, x_{j,n^{p-2}+q}^{(0)}, x_{j,2n^{p-2}+q}^{(0)}, \cdots, x_{j,(n-1)n^{p-2}+q}^{(0)} \rangle$ for $j = 1, \cdots, n$ and obtain a single sorted list $\langle x_{1,q}^{(1)}, x_{1,n^{p-2}+q}^{(1)}, \cdots x_{1,(n-1)n^{p-2}+q}^{(1)}, x_{2,q}^{(1)}, x_{2,n^{p-2}+q}^{(1)}, \cdots, x_{2,(n-1)n^{p-2}+q}^{(1)}, \cdots, x_{n,q}^{(1)}, x_{n,n^{p-2}+q}^{(1)}, \cdots, x_{n,(n-1)n^{p-2}+q}^{(1)} \rangle$;
end for
for $i = 2$ to $p-1$ **do**
for $q = 1$ to n^{p-1-i} **do**
Group n adjacent values of $\langle x_{j,q}^{(i-1)}, x_{j,n^{p-i-1}+q}^{(i-1)}, x_{j,2n^{p-i-1}+q}^{(i-1)}, \cdots x_{j,(n-1)n^{p-i-1}+q}^{(i-1)} \rangle$ for $j = 1, \cdots, n$ and denote the new lists as $\langle x_{j,q}^{(i-1)}, x_{j,n^{p-i-1}+q}^{(i-1)}, \cdots x_{j,(n-1)n^{p-i-1}+q}^{(i-1)} \rangle$ for $j = 1, \cdots, n^i$;
for $k = 2$ to $\lceil \frac{n}{2} \rceil$ **do**
Apply $(n-k)$-spaced sorters between lists j and $j+1$;
end for
Apply $(n-1)$-sorters between lists j and $j+1$ for $j = 1, \cdots, n^i - 1$;
Obtain a single sorted list $\langle x_{1,q}^{(i)}, x_{1,n^{p-i-1}+q}^{(i)}, \cdots, x_{1,(n-1)n^{p-i-1}+q}^{(i)}, x_{2,q}^{(i)}, x_{2,n^{p-i-1}+q}^{(i)}, \cdots, x_{2,(n-1)n^{p-i-1}+q}^{(i)}, \cdots, x_{n^i,q}^{(i)}, x_{n^i,n^{p-i-1}+q}^{(i)}, \cdots, x_{n^i,(n-1)n^{p-i-1}+q}^{(i)} \rangle$;
end for
end for
Output: Sorted list.

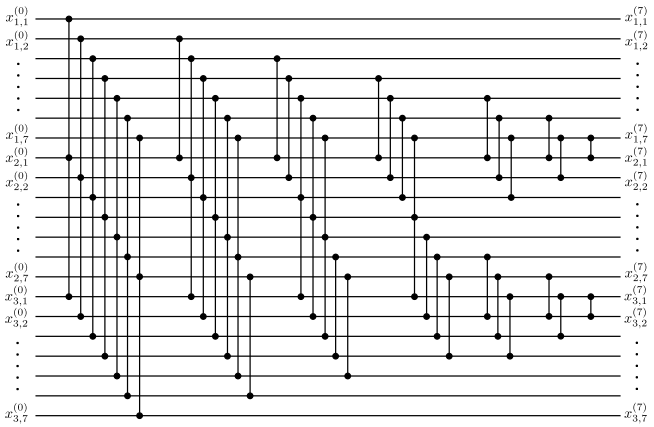


Figure 7.6: A 3-way merging network of $N = 3 \times 7$ inputs implemented via 7 stages.

where $s \geq 0$ *and* $s + l \leq n^i$.

See Sec. A.4 for the proof.

The following theorem proves the correctness of Alg. 6.

Theorem 7.3.2. *Alg. 6 combines* n *sorted lists of* $m = n^{p-1}$ *values as a single sorted list.*

In Alg. 6, the latency is reduced to $1 + (p-1)\lceil \frac{n}{2} \rceil$ for n sorted lists of $m = n^{p-1}$ values.

In the following, we show two examples for comparison of the two algorithms. First, a 3-way merging network of $N = 3 \times 7$ inputs via Alg. 5 is shown in Fig. 7.6. Then, a 3-way merging network of $N = 3 \times 9$ inputs via Alg. 6 is shown in Fig. 7.7. Though there are more inputs in Fig. 7.7 than that in Fig. 7.6, the latency of Alg. 6 is smaller due to recursive decomposition. The numbers of sorters in Figs. 7.6 and 7.7 are given by 40 and 41, respectively. For six more inputs, it requires only one more sorter in Fig. 7.7. Hence, Alg. 6 can be more efficient than Alg. 5 for a large m.

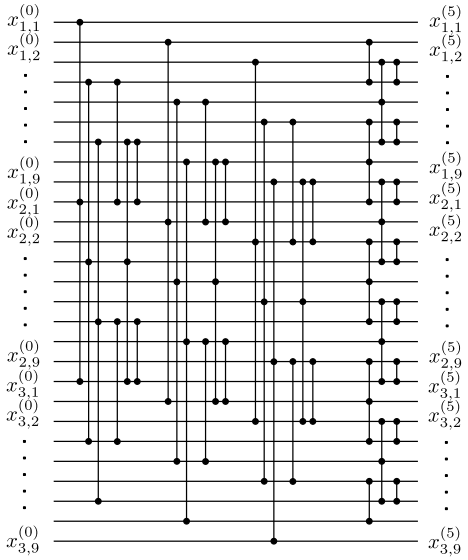


Figure 7.7: A 3-way merging network of $N = 3 \times 9$ inputs implemented via 5 stages.

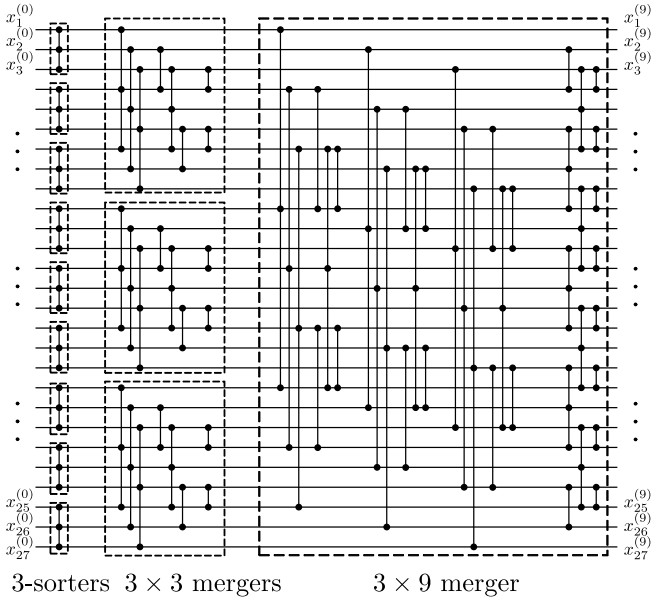


Figure 7.8: A 3-way sorting network of $N = 3^3$ inputs implemented via 9 stages.

7.4 Multiway Sorting

In this section, we first focus on how to construct sorting networks with n-sorters using the multiway merging algorithm in Sec. 7.3. Then, we analyze the latency and the number of sorters of the proposed sorting networks by deriving the closed-form expressions. We compare them with previously proposed SS-Mk in [113] but not the ISS-Mk [125], because for our interested ranges of N, the ISS-Mk requires larger latency due to a large constant.

7.4.1 Multiway sorting algorithm

Based on the multiway merging algorithm in Sec. 7.3, we proposed a parallel sorting algorithm using a divide-and-conquer method. The idea is to first decompose large list of inputs into smaller sublists, then sort each sublist, and finally merge them into one sorted list. The sorting of each sublist is done by further decomposition. For instance, for $N = n^p$ inputs, we first divide the n^p inputs into n lists of n^{p-1} values. Then we sort each of these n lists and combine them with Alg. 6. The sorting operation of each of the n lists is done by dividing the n^{p-1} inputs into n smaller lists of n^{p-2} values. We repeat the above operations until that each of n smaller lists contains only n values, which can be sorted by a single n-sorter. The detailed procedures are shown in Alg. 7.

For example, a 3-way sorting network of $N = 3^3$ inputs is shown in Fig. 7.8. The first stage contains 9 3-sorters. The second stage contains 3 three-way mergers with a depth of 3. The last stage contains a three-way merger with a depth of 5. The total depth is given by 9.

Algorithm 7 Algorithm for sorting $N = n^p$ values.

Input: $N = n^p$ values, $x_1^{(0)}, x_2^{(0)}, \cdots, x_{n^p}^{(0)}$;
Partition the $N = n^p$ values as n^{p-1} lists of n values each, $(x_{j,1}^{(0)}, x_{j,2}^{(0)}, \cdots, x_{j,n}^{(0)})$ for $j = 1, \cdots, n^{p-1}$;
Apply one n-sorter on each of n^{p-1} lists and obtain $\langle x_{j,1}^{(1)}, x_{j,2}^{(1)}, \cdots, x_{j,n}^{(1)} \rangle$ for $j = 1, \cdots, n^{p-1}$;
for $i = 2$ to p **do**
  for $j = 1$ to n^{p-i} **do**
    Apply Alg. 5 on $\langle x_{(j-1)n+k,1}^{(i-1)}, x_{(j-1)n+k,2}^{(i-1)}, \cdots, x_{(j-1)n+k,n^{i-1}}^{(i-1)} \rangle$ for $k = 1, \cdots, n$, and obtain a single sorted list $\langle x_{j,1}^{(i)}, x_{j,2}^{(i)}, \cdots x_{j,n^i}^{(i)} \rangle$;
  end for
end for
Output: Sorted list.

7.4.2 Latency analysis

First, we focus on the latency for sorting N values. The latency is defined as the number of basic sorters in the longest paths from the inputs to the sorted output. In Alg. 7, there are p iterations. In iteration i, there are n^i merging networks, each of which is to merge n sorted lists of n^{p-i} values. For iteration i, the latency is given by $L_{our}(n, n^{i-1}) = 1 + (i-1)\lceil \frac{n}{2} \rceil$. For a sorting network of $N = n^p$ values via Alg. 7, by summing up the latencies of all levels, we obtain the total latency

$$\begin{aligned} L_{our}(n^p) &= \sum_{i=1}^{p} L_{our}(n, n^{i-1}) \\ &= p + \lceil \tfrac{n}{2} \rceil \times \tfrac{p(p-1)}{2}. \end{aligned} \tag{7.1}$$

The closed-form expression of latency for the SS-Mk given in [113] is

$$L_{SS-Mk}(n^p) = 1 + (p-1)n + \frac{(p-1)(p-2)}{2} \lceil \log_2 n \rceil. \tag{7.2}$$

We compare our latency for sorting $N = n^p$ values with that for the SS-Mk

in [113]. From Eqs. (7.1) and (7.2), for $N = n^p$ inputs, p should be as small as possible to obtain small latencies. In Table 7.1, we compare the latencies of Eqs. (7.1) and (7.2) for small p ($p = 2, 3, 4$). It is easily seen that our implementation has a smaller latency than the SS-Mk in [113] for a prime greater than 3. It is also observed that $L_{our}(2^p) = L_{SS-Mk}(2^p) = p(p+1)/2$ for $n = 2$, which is the same as the odd-even merge sort in [31].

Table 7.1: Comparison of latencies of sorting networks of $N = n^p$ inputs via the SS-Mk in [113] and our implementation.

	$p = 2$	$p = 3$	$p = 4$
[113]	$1 + n$	$1 + 2n + \lceil \log_2 n \rceil$	$1 + 3n + 3\lceil \log_2 n \rceil$
Ours	$2 + \lceil \frac{n}{2} \rceil$	$3 + 3\lceil \frac{n}{2} \rceil$	$4 + 6\lceil \frac{n}{2} \rceil$

7.4.3 Analysis of the number of sorters

In the following, we compare the number of sorters of our algorithms with the SS-Mk in [113]. Since the distribution of sorters for an arbitrary sorting network of N inputs is not known, we assume that any m-sorter ($m < n$) has the same delay and area as the basic n-sorter and count the number of sorters. We first derive the closed-form expression of the number of sorters for sorting N values via our Alg. 7. Since the expression of the number of sorters for the SS-Mk was not provided in [113], we also derive the corresponding closed-form expression and compare it with our algorithm. The whole sorting network is constructed recursively by merging small sorted lists into a larger sorted list. We first derive the number of sorters of a merging network

of n lists of n^{p-i} values, which is given by

$$S_{our}(n, n^{p-i}) = (p-i) \cdot M^*_{n^{p-i}} + \frac{n^{p-i}-1}{n-1} \cdot C^*_n + n^{p-i},$$

where $M^*_{n^{p-i}} = \left(1 + \frac{\lceil n/2 \rceil (\lceil n/2 \rceil - 1)}{2}\right) n^{p-i}$ and $C^*_n = (\lceil n/2 \rceil - 1)n - \frac{3\lceil n/2 \rceil (\lceil n/2 \rceil - 1)}{2} - 1$. By summing up the numbers of sorters of all mergers in all stages, we obtain the total number of sorters, which is given by

$$\begin{aligned} T_{our}(n^p) &= \textstyle\sum_{i=1}^{p-1} n^{i-1} \cdot S_{our}(n, n^{p-i}) + n^{p-1} \\ &= \tfrac{p(p-1)}{2} \cdot M^*_{n^{p-1}} + \left[\tfrac{(p-1)n^{p-1}}{n-1} - \tfrac{n^{p-1}-1}{(n-1)^2}\right] \\ &\quad \cdot C^*_n + pn^{p-1}, \end{aligned} \tag{7.3}$$

As $N \to \infty$, $T_{our}(n^p)$ is on the order of $O(A_1 \frac{N \log N (\log N - \log n)}{(\log n)^2/n} + A_2 \frac{N(\log N - \log n)}{\log n} + A_3 \frac{N \log N}{n \log n})$. Similarly for the SS-Mk in [113], the number of sorters of the merging network of n lists of n^{p-i} values each is given by

$$S_{SS-Mk}(n, n^{p-i}) = M^\dagger_{n^{p-i}} + K^\dagger_{n,n^{p-i}} + C^\dagger_n,$$

where

$$M^\dagger_{n^{p-i}} = \Big(\frac{(n+1-\lceil n/2 \rceil)(n-\lceil n/2 \rceil)}{2} + \frac{(\lceil n/2 \rceil + 1)(\lceil n/2 \rceil - 2)}{2} + 2\Big) n^{p-i},$$

$$K^\dagger_{n,n^{p-i}} = \lceil \log_2 n^{p-1-i} \rceil n^{p-i} + (n-3)2^{\lceil \log_2 n^{p-1-i} \rceil + 1},$$

and

$$C_n^{\dagger} = (\lceil n/2 \rceil - 2)n - \frac{3(\lceil n/2+1 \rceil)(\lceil n/2 \rceil - 2)}{2} - \frac{(n+1-\lceil n/2 \rceil)(n-\lceil n/2 \rceil)}{2} - (n-3).$$

The total number of sorters of the sorting network via the SS-Mk in [113] is given by

$$\begin{aligned} T_{SS-Mk}(n^p) &= \textstyle\sum_{i=1}^{p-1} n^{i-1} \cdot S_{SS-Mk}(n, n^{p-i}) + n^{p-1} \\ &= (p-1) \cdot M_{n^{p-1}}^{\dagger} + \tfrac{n^{p-1}-1}{n-1} \cdot C_n^{\dagger} + n^{p-1} \\ &\quad + n^{p-1} \textstyle\sum_{i=1}^{p-2} \lceil i \log_2 n \rceil \\ &\quad + \textstyle\sum_{i=1}^{p-1} n^{i-1}(n-3) 2^{\lceil (p-1-i)\log_2 n \rceil + 1}, \end{aligned} \tag{7.4}$$

As $N \to \infty$, $T_{SS-Mk}(n^p)$ is on the order of $O(B_1 \frac{N(\log N - \log n)}{(\log n)/n} + B_2 \frac{N \log N(\log N - \log n)}{n \log n} + B_3 \frac{N(\log N - \log n)}{n \log n} + B_4 \frac{N}{n})$.

According to the big-O expressions of $T_{our}(n^p)$ and $T_{SS-Mk}(n^p)$, when n is bounded, the asymptotic bounds on the number of sorters required by both our Alg. 7 and the SS-Mk in [113] are given by $O(N \log^2 N)$, which is also the asymptotical bound for the odd-even and bitonic sorting algorithms [31, 109]. When N is fixed and n increases, the first term of the big-O expressions of $T_{our}(n^p)$ and $T_{SS-Mk}(n^p)$ decreases first, then increases, and decreases to zero when $n \to N$. While other terms decrease monotonically with n. Hence, if n is not constrained, the minimum value of $T_{our}(n^p)$ and $T_{SS-Mk}(n^p)$ is one when $n = N$, meaning a single N-sorter is used.

7.4.4 Comparison of the number of sorters

According to the analysis of both our Alg. 7 and the SS-Mk in [113], the number of sorters for sorting $N = n^p$ inputs can be reduced by using a larger basic sorter. However, a very large basic sorter is not feasible due to some practical concerns, such as fan-in and cost. In this chapter, we assume that the basic sorter size is limited. For a given N, we take the total number of sorters in Eqs. (7.3) and (7.4) as a function of p with $n = N^{1/p} \leq n_b$, where n_b is the upper bound of the basic sorter size. When N is not a power of a prime, we append redundant inputs of 0's and get a larger N' such that N' is a power of a prime. Hence, we have $n' = N'^{1/p} = \lceil \lceil N^{1/p} \rceil \rceil$, where $\lceil \lceil x \rceil \rceil$ denotes the smallest prime larger than or equal to x. There exists an optimal p such that the total number of sorters is the minimum. We search for the optimal p's for our Alg. 7 and the SS-Mk [113] using MATLAB. By plugging the optimal p's into Eqs. (7.3) and (7.4), we obtain the total number of sorters for sorting networks of N inputs.

We compare the number of sorters for sorting networks via the Batcher's odd-even algorithm [31], our Alg. 7, and the SS-Mk [113] for wide ranges of N. The results are show in Fig. 7.9. The numbers of sorters are illustrated by staircase curves, because we use a larger sorting network for N not being a power of prime. From Fig. 7.9, the Batcher's odd-even algorithm using 2-sorters always requires more sorters than both our Alg. 7 and the SS-Mk in [113]. For both our Alg. 7 and the SS-Mk [113], the number of sorters is smaller for a larger n_b, meaning that using larger basic sorters reduces the number of sorters. For the comparison of the number of sorters required by our Alg. 7 and the SS-Mk [113], there are three scenarios with respect to three ranges of N. We first focus on $n_b = 10$. For $N \leq 6.25 \times 10^2$, our

Alg. 7 has fewer or the same number of sorters than the SS-Mk as shown in Fig. 7.9. For some segments in $6.25 \times 10^2 < N \leq 3.13 \times 10^3$, our Alg. 7 has fewer sorters than the SS-Mk. For $N > 3.13 \times 10^3$, the SS-Mk in [113] needs fewer sorters. For $n_b = 20$, we have similar results. For $N \leq 1.46 \times 10^4$, our Alg. 7 has fewer or the same number of sorters than the SS-Mk as shown in Fig. 7.9. For some segments in $1.46 \times 10^4 < N < 1.3 \times 10^5$, our Alg. 7 has fewer sorters than the SS-Mk. For $N > 1.3 \times 10^5$, the SS-Mk in [113] needs fewer sorters.

Similarly, we compare the latency of the Batcher's odd-even algorithm, our Alg. 7, and the SS-Mk in [113]. The latencies are obtained by plugging the corresponding optimal p's into Eqs. (7.1) and (7.2) and shown in Fig. 7.10 for $N \leq 2 \times 10^4$. From Fig. 7.10, the Batcher's odd-even algorithm using 2-sorters has the largest latency. For both our Alg. 7 and the SS-Mk [113], the latency can be reduced by having a larger n_b. The latency of our Alg. 7 is not greater than the SS-Mk for $N \leq 2 \times 10^4$ for both $n_b = 10$ and $n_b = 20$ as shown in Fig. 7.10. This is because our Alg. 7 tends to use large sorters, leading to less stages of sorters. We note that the latency goes up and down for some N in Fig. 7.10. This is because of the switching from a smaller basic sorter to a larger one to reduce the number of sorters.

To some researchers' interest, we also compare the number of sorters for N being a power of two. The results are shown in Table 7.2, where columns two and three show the numbers of sorters for the SS-Mk and our Alg. 7, respectively, and column five shows the reduction by our Alg. 7 compared with the SS-Mk [113]. For our Alg. 7, there are up to 46% fewer sorters than the SS-Mk in [113] for $N = 2^i$, for $i = 4, 5, \cdots, 16$. It is also observed that a greater reduction is obtained for small p, meaning our approach is more efficient for networks with larger sorters as basic

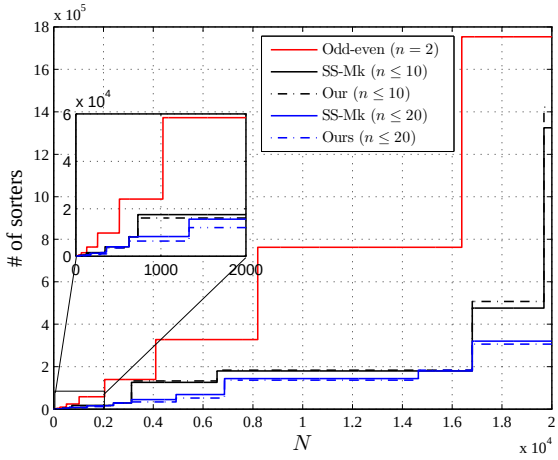


Figure 7.9: Comparison of the number of sorters ($n \leq 10$ and $n \leq 20$) for sorting N inputs via the SS-Mk in [113] and our Alg. 7.

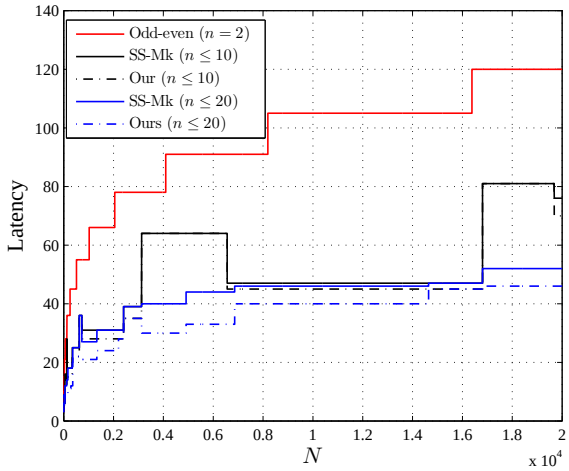


Figure 7.10: Comparison of the latency for sorting N inputs with $n \leq 10$ and $n \leq 20$ via the SS-Mk in [113] and our Alg. 7.

blocks.

Table 7.2: Comparison of the number of sorters for sorting $N = 2^k$ inputs ($1 \leq k \leq 16$) with $n \leq 20$ via the SS-Mk in [113] and our Alg. 7.

N	SS-Mk	Ours	Rd. (%)
2	1	1	0.0
4	5	5	0.0
8	11	11	0.0
16	38	30	21.05
32	95	65	31.58
64	347	207	40.35
128	566	326	42.40
256	1250	690	44.80
512	3952	3500	11.44
1024	8287	6378	23.04
2048	15595	12039	22.80
4096	44652	33891	24.10
8192	143762	136574	5.00
16384	179631	183143	-1.96
32768	1176250	1134692	3.53
65536	1176250	1134692	3.53

7.5 Application in Threshold Logic

In Sec. 7.4.4, we assume all basic sorters in the sorting network are the same and measure the complexity by the number of sorters, since the distribution of sorters is unknown. This would overestimate the total complexity. In this section, we focus on the threshold logic and measure the complexity by the number of threshold gates. In the following, we first briefly introduce the threshold logic, which is very powerful for computing complex functions, such as parity function, addition, multiplication, and sorting, with significantly reduced number of gates. Then, we present an implementation of a large sorter in threshold logic. Last, we compare the complexity

of sorting networks in terms of the number of gates. This is a very narrow application in the sense that sorters are implemented by threshold logic and the inputs are binary values.

7.5.1 Threshold logic

A threshold function [25] f with n inputs ($n \geq 1$), $x_1, x_2, \cdots, x_n$, is a Boolean function whose output is determined by

$$f(x_1, x_2, \cdots, x_n) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i x_i \geq T \\ 0 & \text{otherwise,} \end{cases} \tag{7.5}$$

where w_i is called the *weight* of x_i and T the *threshold*. In this chapter we denote this threshold function as $[x_1, x_2, \cdots, x_n; w_1, w_2, \cdots, w_n; T]$, and for simplicity sometimes denote it as $f = [\mathbf{x}; \mathbf{w}; T]$, where $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ and $\mathbf{w} = (w_1, w_2, \cdots, w_n)$. The physical entity realizing a threshold function is called a threshold gate, which can be realized with CMOS or nano technology. Fig. 7.11 shows the symbol of a threshold gate realizing (7.5).

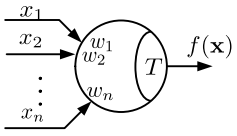
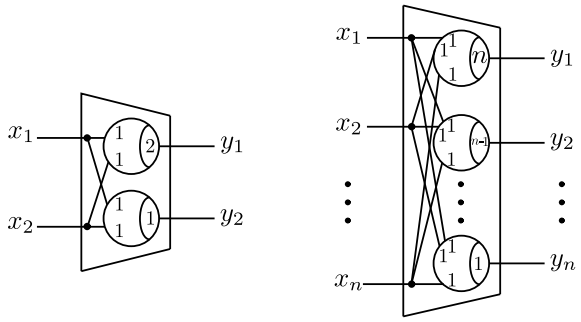


Figure 7.11: Threshold gate realizing $f(\mathbf{x})$ for n inputs, $x_1, x_2, \cdots, x_n$, with corresponding weights $\omega_1, \omega_2, \cdots, \omega_n$ and a threshold T.

7.5.2 n-sorter

Binary sorters can be easily implemented in threshold logic. In [98], a 2-by-2 comparator (2-sorter) was implemented by two threshold gates as shown in Fig. 7.12(a). Similarly, we introduce a threshold logic implementation of an n-sorter as shown in Fig. 7.12(b), where n threshold gates are required. As shown in Fig. 7.12, the number of gates of an n-sorter scales linearly with the number of inputs n. Hence, large sorters are preferred to be used as basic blocks. However, larger sorters are more complex and expensive to be implemented. For practical concerns, such as fan-in and cost, some limit on the size of basic sorters is assumed.



(a) (b)

Figure 7.12: Sorters implemented in threshold logic (a) 2-sorter; (b) n-sorter.

7.5.3 Analysis of number of gates

In the following, we assume all gates are the same and derive the total number of gates. The sorting network of N inputs is composed of multiple stages, of which each partially sorts N values. Not all values in each stage participate the comparison-and-switch operation. A simple way to count the gates is to insert buffer gates in each stage to store values without involving any sorting operation. Buffer insertion

is also needed for implementation of threshold logic in some nanotechnology, where synchronization is required for correction operation. Hence, each stage contains N gates and the total number of gates is obtained by multiplying N to the latency. Note that N does not have to be a power of n. Hence, the total number of gates of our Alg. 7 and the SS-Mk [113] are simply given by

$$Q_{our}(N) = N \cdot L_{our}(N), \tag{7.6}$$

and

$$Q_{SS-Mk}(N) = N \cdot L_{SS-Mk}(N). \tag{7.7}$$

If n is bounded, the total numbers of gates in Eqs. (7.6) and (7.7) have an order of $O(N \log^2 N)$, which is the same as the order for the numbers of sorters via our Alg. 7 and the SS-Mk in [113] in Sec. 7.4.3.

To derive the accurate number of gates, we first derive the number of buffers added for Eqs. (7.6) and (7.7). When N is a power of prime, the number of buffers for sorting $N = n^p$ values via our Alg. 7 and the SS-Mk [113] can be easily obtained due to a regular structure. For our Alg. 7, the number of buffers is given by $G_{our}(N) = (p-1)n^{p-2}\frac{n^2+6n-5}{4} + \frac{((p-2)n^{p-1}-(p-1)n^{p-2}+1)(n+5)}{4(n-1)} + \frac{(p-1)(p-2)}{2}n^{p-1}$ for $n \neq 2$ and $G(n^p) = (p^2-p+4)2^{p-1}-2$ for $n = 2$. Similarly, we derive the number of buffers for the SS-Mk in [113], which is given by $G_{SS-Mk}(N) = 2\sum_{i=2}^{p}(2^{\lceil (i-2)\log_2 n \rceil + 1} - 1)n^{p-i} + \frac{(n^{p-1}-1)(n^2-5)}{2(n-1)} + \frac{(p-1)(n-1)^2 n^{p-1}}{4}$ for $n \neq 2$ and $G(n^p) = (p^2-p+4)2^{p-1}-2$ for $n = 2$. By subtracting the number of buffers from Eqs. (7.6) and (7.7), we obtain the total

numbers of gates for our algorithm and the SS-Mk as shown in the following,

$$R_{our}(n^p) = n^p \cdot L_{our}(n^p) - G_{our}(n^p), \tag{7.8}$$

and

$$R_{SS-Mk}(n^p) = n^p \cdot L_{SS-Mk}(n^p) - G_{SS-Mk}(n^p). \tag{7.9}$$

Though it would overestimate the total number of gates by adding buffers. However, the asymptotic gate counts are not affected, since both $G_{our}(n^p)$ and $G_{SS-Mk}(n^p)$ have the same order of $O(N \log^2 N)$.

7.5.4 Comparison of the number of gates

In the following, we first compare the number of gates with consideration of buffers. Using the same idea as in Sec. 7.4.3, we search for the optimal p's of Eqs. (7.6) and (7.7) using MATLAB. For $n \leq 10$ and $n \leq 20$, the numbers of gates of the SS-Mk and our two implementations are illustrated in Fig. 7.13. We also plot the odd-even sorting for comparison. The curves in Fig. 7.13 are segmented linear lines. This can be explained by Eqs. (7.6) and (7.7), which are functions of N and latency. From Fig. 7.13, the Batcher's odd-even algorithm using 2-sorters has more gates than both our algorithm and the SS-Mk in [113]. For both our Alg. 7 and the SS-Mk [113], the number of gates is smaller with a larger n_b, meaning that using larger basic sorters reduces the number of gates. For the comparison of the number of gates required by our Alg. 7 and the SS-Mk [113], there are also three scenarios with respect to three ranges of N. We first focus on $n_b = 10$. For $N \leq 1.68 \times 10^4$, our Alg. 7 has fewer or the same number of gates than the SS-Mk as shown in Fig. 7.13. For

$1.68 \times 10^4 < N \leq 1.17 \times 10^5$, our Alg. 7 has the same number of gates as the SS-Mk. For $N > 1.17 \times 10^5$, the SS-Mk in [113] needs fewer gates. For $n_b = 20$, we have similar results. For $N \leq 3.71 \times 10^5$, our Alg. 7 has fewer or the same number of gates than the SS-Mk. For some segments in $3.71 \times 10^5 < N \leq 2.47 \times 10^6$, our Alg. 7 has fewer gates than the SS-Mk. For $N > 2.47 \times 10^6$, the SS-Mk in [113] needs fewer gates.

Similarly, we compare the latency of our sorting algorithm with the SS-Mk in [113]. The latencies are obtained by plugging the corresponding optimal p's into Eqs. (7.6) and (7.7) and shown in Fig. 7.14 for $N \leq 2 \times 10^4$. Note that the minimization of the number of gates is essentially to minimize the latency, since each N is fixed in Eqs. (7.6) and (7.7). Fig. 7.14 also shows the minimal latencies of the Batcher's odd-even algorithm. All the latencies are illustrated by staircase curves. From Fig. 7.13, the Batcher's odd-even algorithm using 2-sorters has the largest latency. For both our Alg. 7 and the SS-Mk [113], the latency can be reduced by having a larger n_b. The latency of our Alg. 7 is not greater than the SS-Mk for $N \leq 2 \times 10^4$ for both $n_b = 10$ and $n_b = 20$ as shown in Fig. 7.14. This is because our Alg. 7 tends to use large basic sorters, leading to less stages.

We also compare the number of gates with buffers for N being a power of two. The numbers of gates are minimized by varying p according to Eqs. (7.6) and (7.7) for our algorithm and the SS-Mk [113]. Note the optimal p's are different from those in Sec. 7.4.4. The results are shown in Table 7.3, where columns two to four show the numbers of gates for the SS-Mk, our Alg. 7, and the reduction of our Alg. 7, respectively, with $n \leq 20$, and columns five to seven show those with $n \leq 10$. For $n \leq 10$ and $n \leq 20$, there are up to 25% and 39% fewer gates, respectively, than

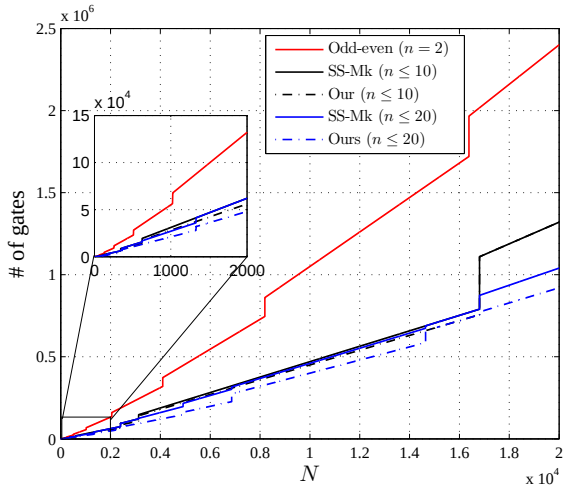


Figure 7.13: Comparison of the number of gates ($n \leq 10$ and $n \leq 20$) for sorting N inputs via the SS-Mk in [113] and our Alg. 7.

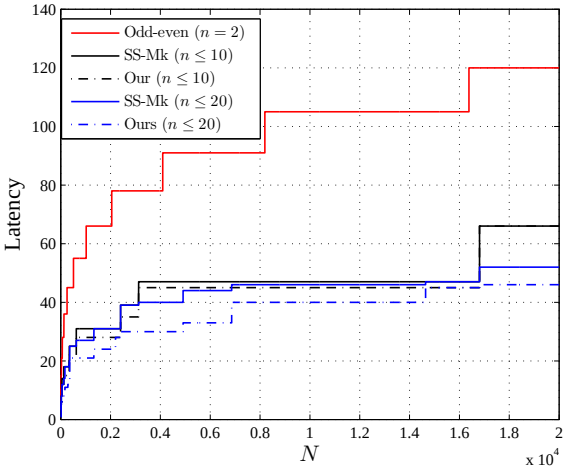


Figure 7.14: Comparison of the latency ($n \leq 10$ and $n \leq 20$) for sorting N inputs via the SS-Mk in [113] and our Alg. 7.

the SS-Mk in [113] for $N = 2^i$ with $i = 1, 5, \cdots, 16$. It is observed that fewer and the same number of gates are needed for $n \leq 20$ than for $n \leq 10$ for all $N = 2^i$ with $i = 1, 2, \cdots, 16$. The reduction percentage of $n \leq 20$ is also greater than or equal to that of $n \leq 10$ for all $N = 2^i$ with $i = 1, 2, \cdots, 16$ but $N = 16$. This means our

Table 7.3: Comparison of the number of gates with buffers for sorting $N = 2^k$ inputs ($1 \leq k \leq 16$) with $n \leq 20$ via the SS-Mk in [113] and our Alg. 7.

N	$n \leq 20$			$n \leq 10$		
	SS-Mk	Ours	Rd. (%)	SS-Mk	Ours	Rd. (%)
2	1	1	0.00	1	1	0.00
2^2	4	4	0.00	4	4	0.00
2^3	8	8	0.00	32	32	0.00
2^4	16	16	0.00	96	80	16.67
2^5	256	192	25.00	256	192	25.00
2^6	768	512	33.33	896	768	14.29
2^7	1792	1152	35.71	2304	1920	16.67
2^8	4608	2816	38.89	4608	3840	16.67
2^9	12800	11264	12.00	12800	11264	12.00
2^{10}	27648	21504	22.22	31744	28672	9.68
2^{11}	63488	49152	22.58	63488	57344	9.68
2^{12}	163840	122880	25.00	192512	184320	4.26
2^{13}	376832	327680	13.04	385024	368640	4.26
2^{14}	770048	737280	4.26	770048	737280	4.26
2^{15}	2162688	1900544	12.12	2162688	2162688	0.00
2^{16}	4325376	3801088	12.12	4325376	4325376	0.00

sorting network takes better advantage of larger basic sorters.

For N being a power of prime, we compare the number of gates without buffers according to Eqs. (7.8) and (7.9). For $N \leq 3 \times 10^4$, we search for the same N's for our Alg. 7 and the SS-Mk with the minimum number of gates. The results are shown in Tables 7.4 and 7.5 for $n \leq 10$ and $n \leq 20$, respectively, where columns three and four show the numbers of gates for the SS-Mk and our Alg. 7, and column five shows the reduction of our Alg. 7. For all N's except for $N = 7^5$, our Alg. 7 has no more gates than the SS-Mk in [113]. There are up to 13% and 23% fewer gates than the SS-Mk in [113] for $n \leq 10$ and $n \leq 20$, respectively. This means our sorting

network takes better advantage of larger basic sorters. We also remark that using a larger sorter size n may reduce the number of gates for sorting $N = n^p$ inputs. For all common N's for $n \leq 10$ in Table 7.4 and $n \leq 20$ in Table 7.5, the same number of gates is needed, since the same sorter size n is used. For all remaining N's except for $N = 3^9$ in Table 7.4, there is a corresponding larger N's in Table 7.5 with fewer gates. For $N = 3^9 = 19683$ in Table 7.4 and $N = 13^4 = 28561$ in Table 7.5, the latter has about 1% more gates than the former, but accounts for 45% more inputs.

7.6 Conclusion

In this chapter, we proposed a new merging algorithm based on n-sorters for parallel sorting networks, where n is prime. Based on the n-way merging, we also proposed a merge sorting algorithm. Our sorting algorithm is a direct generalization of odd-even merge sort with n-sorters as basic blocks. By using larger sorters ($2 \leq n \leq 20$), the number of sorters as well as the latency is reduced greatly. In comparison with other multiway sorting networks in [113], our implementation has a smaller latency and fewer sorters for wide ranges of $N \leq 1.46 \times 10^4$. We also showed an application of sorting networks implemented by linearly scaling sorters in threshold logic and have a similar conclusion that the number of gates can be greatly reduced by using larger sorters.

Table 7.4: Comparison of the number of gates without buffers for sorting $N = n^p$ inputs for $n \leq 10$ via the SS-Mk in [113] and our Alg. 7.

N	n^p	$n \leq 10$		
		SS-Mk	Ours	Rd. (%)
2	2	2	2	0.00
3	3	3	3	0.00
5	5	5	5	0.00
7	7	7	7	0.00
9	3^2	29	29	0.00
25	5^2	118	110	6.78
27	3^3	197	188	4.57
49	7^2	305	269	11.80
81	3^4	1067	998	6.47
125	5^3	1450	1315	9.31
128	2^7	2942	2942	0.00
343	7^3	5072	4728	6.78
625	5^4	13489	12140	10.00
729	3^6	22801	20411	10.48
1024	2^{10}	48126	48126	0.00
2401	7^4	63354	62254	1.74
3125	5^5	108175	97265	10.09
4096	2^{12}	278526	278526	0.00
6561	3^8	377375	330236	12.49
8192	2^{13}	655358	655358	0.00
16807	7^5	688713	704693	-2.32
19683	3^9	1443791	1259711	12.75

Table 7.5: Comparison of the number of gates without buffers for sorting $N = n^p$ inputs for $n \leq 20$ via the SS-Mk in [113] and our Alg. 7.

N	n^p	$n \leq 20$		
		SS-Mk	Ours	Rd. (%)
2	2	2	2	0.00
3	3	3	3	0.00
5	5	5	5	0.00
7	7	7	7	0.00
11	11	11	11	0.00
13	13	13	13	0.00
17	17	17	17	0.00
19	19	19	19	0.00
25	5^2	118	110	6.78
27	3^3	197	188	4.57
49	7^2	305	269	11.80
121	11^2	1117	917	17.91
125	5^3	1450	1315	9.31
169	13^2	1814	1454	19.85
289	17^2	3970	3074	22.57
361	19^2	5501	4205	23.56
625	5^4	13489	12140	10.00
729	3^6	22801	20411	10.48
1331	11^3	29107	26668	8.38
2197	13^3	54703	50763	7.20
2401	7^4	63354	62254	1.74
3125	5^5	108175	97265	10.09
4913	17^3	156812	143443	8.53
6859	19^3	239590	221052	7.74
14641	11^4	564513	562214	0.41
16807	7^5	688713	704693	-2.32
28561	13^4	1230724	1271788	-3.34

Chapter 8

Conclusion and Future Work

As the technologies scaling down, many issues arise and need to be accounted for, such as delay issue in on-chip interconnect, noise and decoherence in quantum computation, and quantum effects with nano technologies. To address these issues, efficient signal processing techniques or new design approaches are imperative. In this dissertation, we propose several efficient processing techniques and approaches in the following area: delay modeling, crosstalk avoidance coding, quantum error correction, and threshold logic design.

The delay issue in on-chip interconnect is motivated by the fact that gate delay decreases with scaling, but global interconnect delay increases due to crosstalk. We proposed analytical delay models for on-chip interconnects with improved accuracy, leading to a new CAC with worst-case delay 30-40% smaller than the best known in the literature.

Quantum error correction codes (QECCs) are needed to protect quantum information against noise and decoherence. Given the good error-correcting performance

and existing iterative decoding algorithms of classic LDPCs, it is desirable to obtain LDPC-based QECCs. Several QECCs based on nonbinary LDPC codes have been proposed with a much better error-correcting performance than existing quantum codes over a qubit channel. We proposed stabilizer codes based on the nonbinary QC-LDPC codes for qubit channels. Results show that QECCs based on the nonbinary LDPC codes achieve better performance than that based on binary LDPC codes.

Finally, threshold logic designs in nano technologies are investigated. Nano devices, such as resonant tunneling diodes (RTDs), quantum cellular automata (QCA), and single electron transistors (SETs), are fit for the threshold logic, which is different from the widely used Boolean logic in CMOS technology. Boolean gates, such as AND, OR, NOT, NAND, NOR, and XOR, are used there as basic building blocks. Besides, the fan-in of a threshold gate in RTD nanotechnology needs to be bounded for both reliability and performance purposes. We first focus on the implementations of symmetric functions in threshold logic, as AND, OR, NAND, NOR, and XOR are all special cases of symmetric functions. Furthermore, any Boolean function can be treated as a symmetric function by replicating its inputs. We propose an improved sort-and-search algorithm to implement any symmetric function in threshold logic. Both sorting and searching networks in our proposed algorithm use multi-input threshold gates. Since XOR cannot be realized in a single threshold gate, we also proposed a majority-class threshold tree architecture for XORs with bounded fan-in, and compare it with a Boolean-class architecture and the sort-and-search approach therein. Analytical results show that the majority class outperforms other architectures in terms of both hardware complexity and latency.

For the future work, following directions can be investigated:

- As the clock frequency approaches multi-gigahertz, the parasitic inductance of on-chip interconnects has become significant and its detrimental effects, including increased delay, voltage overshoots and undershoots, and increased crosstalk noise [58–60], cannot be ignored. Hence, with the process technologies scaling down into deep submicrometer (DSM) and the clock frequency approaching multi-gigahertz range, the crosstalk delay and noise due to the capacitive and inductive coupling become the performance bottleneck in many high-performance VLSI designs, especially for global on-chip buses. It is imperative for designers to devise new techniques to address both capacitive and inductive couplings simultaneously.

- The proposed QECCs in this dissertation have a column weight of two. Quasi-cyclic QECCs with column weights more than two provide more powerful error correction capability and are worth to be investigated.

- It has been shown that some complex Boolean functions can be realized with a single threshold gate. However, efficient identification of the threshold function for a given problem is not fully investigated. The identification by reformulating existing Boolean expressions for a given problem is worth to be investigated.

- In this dissertation, we show how to construct multi-input XORs using our proposed sort-and-search algorithm and use them as building blocks for finite field multiplications. However, some arithmetic operations might be decomposed into small blocks, which can be implemented by symmetric functions

more efficiently. Since our proposed sort-and-search algorithm is applicable for any symmetric Boolean function, such decomposition using blocks based on our sort-and-search algorithm is worth to be investigated.

Bibliography

[1] [Online], "International technology roadmap for semiconductors," available at http://www.itrs.net/Links/2011ITRS/Home2011.htm.

[2] R. Kay and L. Pileggi, "PRIMO: Probability interpretation of moments for delay calculation," *Proceedings of the 35th annual Design Automation Conference*, vol. 35, pp. 463–468, 1998.

[3] C. J. Alpert, A. Devgan, and C. V. Kashyap, "RC delay metrics for performance optimization," *IEEE Transactions on Computer Aided Design Integrated Circuits System*, vol. 20, no. 5, pp. 571–582, 2001.

[4] F. Liu, C. Kashyap, and C. J. Alpert, "A delay metric for RC circuits based on the weibull distribution," *Proc. IEEE/ACM International Conference on Computer-Aided Design*, pp. 620–624, 2002.

[5] J. A. Davis and J. D. Meindl, "Compact distributed RLC interconnect models-part ii: Coupled line transient expressions and peak crosstalk in multilevel networks," *IEEE Transactions on Electron Devices*, vol. 47, no. 11, pp. 2078–2087, November 2000.

[6] S. Roy and A. Dounavis, "Closed-form delay and crosstalk models for RLC on-chip interconnects using a matrix rational approximation," *IEEE Transactions on Computer Aided Design Integrated Circuits System*, vol. 28, no. 10, pp. 1481–1492, October 2009.

[7] P. P. Sotiriadis and A. Chandrakasan, "Reducing bus delay in submicron technology using coding," *Proceedings of the Asia and South Pacific Design Automation Conference*, pp. 109–114, February 2001.

[8] C. Duan, A. Tirumala, and S. Khatri, "Analysis and avoidance of cross-talk in on-chip buses," *The Ninth Symposium on High Performance Interconnects (HOTI '01)*, pp. 133–138, August 2001.

[9] C. Duan and S. Khatri, "Exploiting crosstalk to speed up on-chip buses," *Proceedings of the Conference on Design Automation and Test in Europe*, vol. 2, pp. 778–783, February 2004.

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

[11] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge, 2000.

[12] D. J. C. MacKay, G. Mitchison, and P. L. McFadden, "Sparse-graph codes for quantum error correction," *Proc. IEEE Int. Symp. on Information Theory*, vol. 50, no. 10, pp. 2315–2330, 2004.

[13] M. S. Postol, "A proposed quantum low density parity check code," *quant-ph/0108131*, 2001.

[14] G. D. Forney, M. Grassl, and S. Guha, "Convolutional and tail-biting quantum error-correcting codes," *IEEE Trans. Info. Theory*, vol. 53, no. 3, pp. 865–880, 2007.

[15] D. Poulin, J. Tillich, and H. Ollivier, "Quantum serial turbo codes," *IEEE Trans. Info. Theory*, vol. 55, no. 6, pp. 2776–2798, 2009.

[16] P. Tan, "Exploring error-correction technology in source coding and quantum communications," Ph.D. Dissertation, Lehigh University, 2009.

[17] P. Tan and J. Li, "Efficient quantum stabilizer codes: LDPC and LDPC-convolutional constructions," *IEEE Trans. Info. Theory*, vol. 56, no. 1, pp. 476–491, January 2010.

[18] M. M. Wilde and M. Hsieh, "Entanglement boosts quantum turbo codes," *Proc. IEEE Int. Symp. on Information Theory*, pp. 445–449, July 2011.

[19] M. M. Wilde and S. Guha, "Polar codes for degradable quantum channels," eprint ArXiv: quant-ph/1109.5346v2 2011.

[20] M. M. Wilde and J. M. Renes, "Quantum polar codes for arbitrary channels," eprint ArXiv: quant-ph/1201.2906v2 2012.

[21] Z. Dutton, S. Guha, and M. M. Wilde, "Performance of polar codes for quantum and private classical communication," eprint ArXiv: quant-ph/1205.5980v1 2012.

[22] M. C. Davey and D. J. C. MacKay, "Low-density parity check codes over GF(q)," *IEEE Commun. Lett.*, vol. 2, no. 6, pp. 165–167, June 1998.

[23] K. Kasai, M. Hagiwara, H. Imai, and K. Sakaniwa, "Quantum error correction beyond the bounded distance decoding limit," *IEEE Trans. Info. Theory*, vol. 58, no. 2, pp. 1223–1230, February 2012.

[24] D. Goldhaber-Gordon, M. S. Montemerlo, J. C. Love, G. J. Opiteck, and J. C. Ellenbogen, "Overview of nanoelectronic devices," *Proceedings of the IEEE*, vol. 85, no. 4, pp. 521–540, April 1997.

[25] S. Muroga, *Threshold Logic and Its Applications*. New York: WILEY-INTERSCIENCE, 1971.

[26] V. Annampedu and M. D. Wagh, "Reconfigurable approximate pattern matching architectures for nanotechnology," *Microelectronics*, vol. 38, pp. 430–438, 2007.

[27] C. Pacha, U. Auer, C. Burwick, P. Glosekotter, A. Brennemann, W. Prost, F. Tegude, and K. F. Goser, "Threshold logic circuit design of parallel adders using resonant tunneling devices," *IEEE Trans. VLSI Systems*, vol. 8, no. 5, pp. 558–572, October 2000.

[28] C. Pacha and K. Goser, "Design of arithmetic circuits using resonant tunneling diodes and threshold logic," in *Proc. of the 2nd Workshop on Innovative Circuits and Systems for Nanoelectronics*, Delft, NL, Sep. 1997, pp. 83–93.

[29] Y. Sun and M. D. Wagh, "A fan-in bounded low delay adder for nanotechnology," in *Proc. of 2010 NanoTech Conf., vol. 2*, Anaheim, CA, July 2010, pp. 83–86.

[30] B. Sunar and C. L. Koc, "Mastrovito multiplier for all trinomials," *IEEE Trans. Computers*, vol. 48, no. 5, pp. 522–527, May 1999.

[31] K. E. Batcher, "Sorting networks and their applications," in *Proc. The Spring Joint Computer Conference*. ACM, 1968, pp. 307–314.

[32] S. Sridhara, A. Ahmed, and N. Shanbhag, "Coding for reliable on-chip buses: A class of fundamental bounds and practical codes," *IEEE Transactions on Computer Aided Design Integrated Circuits System*, vol. 26, no. 5, pp. 977–982, May 2007.

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

[34] R. Gupta, B. Tutuianu, and L. T. Pileggi, "The Elmore delay as a bound for RC trees with generalized input signals," *IEEE Transactions on CAD*, vol. 16, no. 1, pp. 95–104, January 1997.

[35] J. M. Rabaey, *Digital integrated circuits*. Prentice-Hall, 1996.

[36] Y. I. Ismail, E. G. Friedman, and J. L. Neves, "Figures of merit to characterize the importance of on-chip inductance," *IEEE Trans. VLSI Systems*, vol. 7, no. 4, pp. 440–449, December 1999.

[37] L. Zhang, J. M. Wilson, R. Bashirullah, L. Luo, J. Xu, and P. D. Franzon, "A 32-Gb/s on-chip bus with driver pre-emphasis signaling," *IEEE Trans. VLSI Systems*, vol. 17, no. 9, pp. 1267–1274, September 2009.

[38] T. Sakurai, "Closed-form expressions for interconnection delay, coupling, and crosstalk in VLSI's," *IEEE Transactions on Electron Devices*, vol. 40, no. 1, pp. 118–124, January 1993.

[39] S. Khatri, R. Brayton, and A. Sangiovanni-Vincentelli, *Crosstalk noise immune VLSI design using regular layout fabrics.* Kluwer Academic Publishers, 2001.

[40] [Online], "FreePDK45," available at http://www.eda.ncsu.edu/wiki/FreePDK.

[41] F. Shi, X. Wu, and Z. Yan, "Improved crosstalk avoidance codes based on a novel pattern classification," *IEEE Workshop on Signal Processing Systems (SiPS)*, pp. 245–250, October 2011.

[42] P. P. Sotiriadis, "Interconnect modeling and optimization in deep sub-micron technologies," Ph.D. Dissertation, Massachusetts Institute of Technology, 2002.

[43] S. Sridhara, G. Balamurugan, and N. Shanbhag, "Joint equalization and coding for on-chip bus communication," *IEEE Trans. VLSI Systems*, vol. 16, no. 3, pp. 314–318, March 2008.

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

[45] C. Duan, C. Zhu, and S. P. Khatri, "Forbidden transition free crosstalk avoidance codec design," *Proceedings of annual Design Automation Conference*, pp. 986–991, 2008.

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

[47] X. Wu and Z. Yan, "Efficient CODEC designs for crosstalk avoidance codes based on numeral systems," *IEEE Trans. VLSI Systems*, vol. 19, no. 4, pp. 548–558, April 2011.

[48] F. Shi, X. Wu, and Z. Yan, "Improved analytical delay models for coupled interconnects," *in Proc. IEEE Workshop on Signal Processing Systems (SiPS)*, pp. 134–139, October 2011.

[49] [Online], "PDK for the 45nm technology," available at http://www.eda.ncsu.edu/wiki/FreePDK.

[50] ——, "Predictive technology model (PTM)," available at http://http://ptm.asu.edu.

[51] B. Victor, "Bus encoding to prevent crosstalk delay," Master Thesis, University of California, Berkeley, 2001.

[52] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

[53] F. Shi, X. Wu, and Z. Yan, "New crosstalk avoidance codes based on a novel pattern classification," available at http://arxiv.org/abs/1209.2672.

[54] S. R. Sridhara, A. Ahmed, and N. R. Shanbhag, "Area and energy efficient crosstalk avoidance codes for on-chip buses," *in Proc. Int. Conference on Computer Design*, pp. 12–17, 2004.

[55] K. Hirose and H. Yasuura, "A bus delay reduction technique considering crosstalk," in *Design, Automation and Test in Europe Conference and Exhibition 2000. Proceedings.* IEEE, 2000, pp. 441–445.

[56] F. Shi, X. Wu, and Z. Yan, "New crosstalk avoidance codes based on a novel pattern classification," *IEEE Trans. VLSI Systems*, vol. 21, no. 10, pp. 1892–1902, 2013.

[57] ——, "Improved analytical delay models for rc-coupled interconnects," *IEEE Trans. VLSI Systems*, vol. 22, no. 7, pp. 1639–1644, 2014.

[58] Y. I. Ismail, "On-chip inductance cons and pros," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 10, no. 6, pp. 685–694, 2002.

[59] Y. Cao, X. Huang, N. H. Chang, S. Lin, O. S. Nakagawa, W. Xie, D. Sylvester, and C. Hu, "Effective on-chip inductance modeling for multiple signal lines and application to repeater insertion," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 10, no. 6, pp. 799–805, 2002.

[60] S. Tu, Y. Chang, and J. Jou, "RLC coupling-aware simulation and on-chip bus encoding for delay reduction," *IEEE Transactions on Computer Aided Design Integrated Circuits System*, vol. 25, no. 10, pp. 2258–2264, 2006.

[61] J. A. Davis and J. D. Meindl, "Compact distributed RLC interconnect models part i: Single line transient, time delay, and overshoot expressions," *IEEE Transactions on Electron Devices*, vol. 47, no. 11, pp. 2068–2077, November 2000.

[62] Y. I. Ismail and E. G. Friedman, "Effects of inductance on the propagation delay and repeater insertion in vlsi circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 2, pp. 195–206, 2000.

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

[64] D. Gottesman, "Stabilizer codes and quantum error correction," Ph.D. Dissertation, California Institute of Technology, 1997.

[65] A. R. Calderbank and P. W. Shor, "Good quantum error-correcting codes exist," *Phys. Rev. A*, vol. 54, no. 2, pp. 1098–1105, August 1996.

[66] A. M. Steane, "Multiple particle interference and quantum error correction," vol. 452, pp. 2551–2577, 1996.

[67] ——, "Error-correcting codes in quantum theory," *Phys. Rev. Lett*, vol. 77, pp. 793–797, June 1996.

[68] A. R. Calderbank, E. M. Rains, P. W. Shor, and N. J. A. Sloane, "Quantum error correction via codes over GF(4)," *Proceedings Of the Annual Allerton Conference On Communication Control And Computing*, vol. 34, pp. 662–672, 1996.

[69] D. Poulin and Y. Chung, "On the iterative decoding of sparse quantum codes," *Quantum Info. and Comp.*, vol. 8, no. 10, pp. 987–1000, November 2008.

[70] C. Poulliat, M. Fossorier, and D. Declercq, "Design of regular $(2,d_c)$-LDPC codes over GF(q) using their binary images," *IEEE Trans. Computers*, vol. 56, no. 10, pp. 1626–1635, October 2008.

[71] L. Zeng, L. Lan, Y. Y. Tai, B. Zhou, S. Lin, and K. Abdel-Ghaffar, "Construction of nonbinary cyclic, quasi-cyclic and regular LDPC codes: A finite geometry approach," *IEEE Trans. Computers*, vol. 56, no. 3, pp. 378–387, March 2008.

[72] S. Song, B. Zhou, S. Lin, and K. Abdel-Ghaffar, "A unified approach to the construction of binary and nonbinary quasi-cyclic LDPC codes based on finite fields," *IEEE Trans. Computers*, vol. 57, no. 1, pp. 84–93, January 2009.

[73] B. Zhou, J. Kang, S. Song, S. Lin, and K. Abdel-Ghaffar, "Construction of non-binary quasi-cyclic LDPC codes by arrays and array dispersions," *IEEE Trans. Computers*, vol. 57, no. 6, pp. 1652–1662, June 2009.

[74] M. Hagiwara, K. Kasai, H. Imai, and K. Sakaniwa, "Spatially coupled quasi-cyclic quantum LDPC codes," *Proc. IEEE Int. Symp. on Information Theory*, pp. 638–642, July 2011.

[75] M. Fossorier and D. Declercq, "Quasi-cyclic low-density parity-check codes from circulant permutation matrices," *IEEE Trans. Info. Theory*, vol. 50, no. 8, pp. 1788–1793, 2004.

[76] [Online], "An LDPC decoding algorithm based on number-theoretic transform," available at http://ivms.stanford.edu/~varodayan/multilevel/index.html.

[77] M. D. Wagh, Y. Sun, and V. Annampedu, "Implementation of comparison function using quantum-dot cellular automata," in *Proc. NanoTech2008*, vol. 3, June 2008, pp. 76–79.

[78] V. Annampedu and M. D. Wagh, "Reconfigurable approximate pattern matching architectures for nanotechnology," *Microelectronics Journal*, vol. 38, pp. 430–438, 2007.

[79] K.-Y. Siu, V. P. Roychowdhury, and T. Kailath, "Depth-size tradeoffs for neural computation," *Computers, IEEE Transactions on*, vol. 40, no. 12, pp. 1402–1412, 1991.

[80] K.-Y. Siu and J. Bruck, "Neural computation of arithmetic functions," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1669–1675, 1990.

[81] K.-Y. Siu, V. Roychowdhury, and T. Kailath, "Circuit complexity for neural computation," in *Signals, Systems and Computers, 1991. 1991 Conference Record of the Twenty-Fifth Asilomar Conference on.* IEEE, 1991, pp. 487–490.

[82] K.-Y. Siu and V. P. Roychowdhury, "On optimal depth threshold circuits for multiplication and related problems," *SIAM Journal on Discrete Mathematics*, vol. 7, no. 2, pp. 284–292, 1994.

[83] K.-Y. Siu, J. Bruck, T. Kailath, and T. Hofmeister, "Depth efficient neural networks for division and related problems," *Information Theory, IEEE Transactions on*, vol. 39, no. 3, pp. 946–956, 1993.

[84] R. Lidl and H. Niederreiter, *Introduction to Finite Fields and Their Applications*. Cambridge University Press, 1994.

[85] Z. Yan and D. Sarwate, "Reduced-complexity pipelined architectures for finite field inversions," in *Proc. 2006 IEEE Workshop on Signal Processing Systems (SiPS06)*, October 2006, pp. 56–61.

[86] Z. Yan, "Digit-serial systolic architectures for inversions over GF(2^m)," in *Proc. 2006 IEEE Workshop on Signal Processing Systems (SiPS06)*, October 2006, pp. 77–82.

[87] Z. Yan, D. Sarwate, and Z. Liu, "Hardware-efficient systolic architectures for inversion in GF(2^m)," in *IEE Proc. on Information Security*, vol. 152, no. 1, October 2005, pp. 31–45.

[88] ——, "High-speed systolic architectures for finite field inversion," *Integration: the VLSI Journal*, vol. 38, no. 3, pp. 383–398, January 2005.

[89] Z. Yan and D. V. Sarwate, "New systolic architectures for inversion and division in GF(2^m)," *IEEE Transactions on Computers*, vol. 52, no. 11, pp. 1515–1520, November 2003.

[90] P. M. Lewis and C. L. Coates, *Threshold logic*. Wiley, 1967.

[91] S. Muroga, "The principle of majority decision logical elements and the complexity of their circuits." in *IFIP Congress*, 1959, pp. 400–406.

[92] R. C. Minnick, "Linear-input logic," *IRE Trans. Electronic Computers*, vol. EC-10, no. 1, pp. 6–16, March 1961.

[93] V. Annampedu and M. D. Wagh, "Building multi-input RTD circuits under reliability constraints," in *Proc. the 2nd IEEE Int. Workshop on Defect and Fault Tolerant Nanoscale Architectures (NANOARCH 2006)*, June 2006, pp. 45–52.

[94] K. Maezawa, "Analysis of switching time of monostable-bistable transition logic elements based on simple model calculation," *Jpn. J. Appl. Phys.*, vol. 34, no. 2B, pp. 1213–1217, February 1995.

[95] B. G. Horne and D. R. Hush, "On the node complexity of neural networks," *Neural Networks*, vol. 7, no. 9, pp. 1413–1426, 1994.

[96] V. Beiu, J. Peperstraete, and R. Lauwereins, "Algorithms for fan-in reduction," in *Les réseaux neuro-mimétiques et leurs applications. Journées internationales*, 1992, pp. 589–600.

[97] ——, "Simpler neural networks by fan-in reduction," *Proceedings of the International Joint Conference on Neural Networks IJCNN'92,*, pp. 204–209, 1992.

[98] ——, "Enhanced threshold gate fan-in reduction algorithms," in *Proceedings of the third international conference on Young computer scientists.* Tsinghua University Press, 1993, pp. 339–342.

[99] V. Annampedu and M. D. Wagh, "Decomposition of threshold functions into bounded fan-in threshold functions," *Information and Computation*, vol. 227, pp. 84–101, 2013.

[100] N. Red'kin, "Synthesis of threshold circuits for certain classes of boolean functions," *Cybernetics and Systems Analysis*, vol. 6, no. 5, pp. 540–544, 1970.

[101] V. Beiu, "Constant fan-in digital neural networks are vlsi-optimal," in *Mathematics of Neural Networks*. Springer, 1997, pp. 89–94.

[102] J.-H. Guo and C.-L. Wang, "Systolic array implementation of euclid's algorithm for inversion and division in GF(2^m)," *Computers, IEEE Transactions on*, vol. 47, no. 10, pp. 1161–1167, 1998.

[103] E. D. Mastrovito, "VLSI architectures for computations in Galois field," Ph.D. Dissertation, Linkoping University, 1991.

[104] J. Deschamps, J. L. Imana, and G. D. Sutter, *Hardware Implementation of Finite-Field Arithmetic*. The McGraw-Hill Companies, 2009.

[105] J. L. Massey and J. K. Omura, "Computational method and apparatus for finite field arithmetic," *US Patent No. 4,587,627, to OMNET Assoc., Sunnyvale CA, Washington, D.C.: Patent and Trademark Office*, 1986.

[106] R. Zhang, P. Gupta, L. Zhong, and N. K. Jha, "Threshold network synthesis and optimization and its application to nanotechnologies," *IEEE Transactions on Computer Aided Design Integreated Circuits System*, vol. 24, no. 1, pp. 107–118, January 2005.

[107] I. Wegener *et al.*, "The complexity of boolean functions," 1987.

[108] W. Prost, U. Auer, F. J. Tegude, C. Pacha, K. F. Goser, G. Janssen, and T. van der Roer, "Manufacturability and robust design of nanoelectronic logic circuits based on resonant tunnelling diodes," *Int. J. Circ. Theor. App*, vol. 28, pp. 537–552, 2000.

[109] K. E. Batcher, "On bitonic sorting networks," in *Proc. Internation Conference on Parallel Processing (ICPP)*, 1990, pp. 376–379.

[110] D. E. Knuth, "The Art of Computer Programming. Sorting and Searching, vol. III," 1973.

[111] D.-L. Lee and K. E. Batcher, "A multiway merge sorting network," *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 2, pp. 211–215, 1995.

[112] B. Parker and I. Parberry, "Constructing sorting networks from k-sorters," *Information Processing Letters*, vol. 33, no. 3, pp. 157–162, 1989.

[113] Q. Gao and Z. Liu, "Sloping-and-shaking," *Science in China Series E: Technological Sciences*, vol. 40, no. 3, pp. 225–234, 1997.

[114] F. Shi, Z. Yan, and M. D. Wagh, "An enhanced multiway sorting network based on n-sorters," *Available at*, [Online]. Available at http://arxiv.org/abs/1407.0961.

[115] [Online], "Sec 2: Recommended elliptic curve domain parameters," available at http://http://www.secg.org.

[116] R. C. Mullin, I. M. Onyszchuk, S. A. Vanstone, and R. M. Wilson, "Optimal normal bases in GF(p^n)," *Discrete Appl. Math*, vol. 22, no. 2, pp. 149–161, 1988/89.

[117] K. J. Liszka and K. E. Batcher, "A modulo merge sorting network," in *Proc. Fourth Symposium on the Frontiers of Massively Parallel Computation, 1992.* IEEE, 1992, pp. 164–169.

[118] M. Ajtai, J. Komlós, and E. Szemerédi, "An $o(n \log n)$ sorting network," in *Proc. The fifteenth annual ACM symposium on Theory of Computing.* ACM, 1983, pp. 1–9.

[119] A. Farmahini-Farahani, H. J. Duwe, M. J. Schulte, and K. Compton, "Modular design of high-throughput, low-latency sorting units," *IEEE Transactions on Computers*, vol. 62, no. 7, pp. 1389–1402, 2013.

[120] R. Beigel and J. Gill, "Sorting n objects with a k-sorter," *IEEE Transactions on Computers*, vol. 39, no. 5, pp. 714–716, 1990.

[121] T. Nakatani, S.-T. Huang, B. W. Arden, and S. K. Tripathi, "k-way bitonic sort," *IEEE Transactions on Computers*, vol. 38, no. 2, pp. 283–288, 1989.

[122] D. Lee and K. E. Batcher, "On sorting multiple bitonic sequences," in *Proc. International Conference on Parallel Processing (ICPP 1994).*, vol. 1, 1994, pp. 121–125.

[123] T. Leighton, "Tight bounds on the complexity of parallel sorting," in *Proc. The sixteenth annual ACM symposium on Theory of Computing.* ACM, 1984, pp. 71–80.

[124] K. J. Liszka and K. E. Batcher, "A generalized bitonic sorting network," in *Proc. International Conference on Parallel Processing (ICPP 1993)*, vol. 1, 1993, pp. 105–108.

[125] L. Zhao, Z. Liu, and Q. Gao, "An efficient multiway merging algorithm," *Science in China Series E: Technological Sciences*, vol. 41, no. 5, pp. 543–551, 1998.

[126] R. Drysdale, III and F. H. Young, "Improved divide sort merge sorting networks," *SIAM Journal on Computing*, vol. 4, no. 3, pp. 264–270, 1975.

[127] D. C. Van Voorhis, "An economical construction for sorting networks," in *Proceedings of the May 6-10, 1974, national computer conference and exposition.* ACM, 1974, pp. 921–927.

Appendix A

Proofs for Sorting Algorithms

A.1 Proof of Lemma 6.3.1

Proof. The proof of the lemma can be reduced to showing that for $l > 0$ any two wires $s, s+l \in \mathbb{Z}_m$ of each list are sorted as shown in Fig. 7.4(b). We prove the lemma by contradiction. The inputs satisfy $x_{j,s} \leq x_{j,s+l}$ for $j \in \mathbb{Z}_n$ and $s, s+l \in \mathbb{Z}_m$. Suppose there exist $k \in \mathbb{Z}_n$ and $s, s+l \in \mathbb{Z}_m$ such that $x'_{k,s} > x'_{k,s+l}$. Since the sorter for $x_{k,s}$ $k \in \mathbb{Z}_m$ acts as a permutation of the index k, we denote such permutation of the sorter connecting wire s as $f : \{1, \cdots, n\} \rightarrow \{1, \cdots, n\}$. Because f is bijection, an inverse f^{-1} exists. Then we have $x_{f^{-1}(t),s+l} \geq^a x_{f^{-1}(t),s} = x'_{t,s} \geq x'_{k,s} > x'_{k,s+l}$ for $k \leq t \leq n$, where the "$\geq^a$" is because the inputs are sorted and the "$=$" is due to the permutation. There are $n-k+1$ inputs of $x_{f^{-1}(t),s+l}$ satisfying $x_{f^{-1}(t),s+l} > x'_{k,s+l}$. However, at most $n-k$ outputs satisfy $x'_{t,s+l} > x'_{k,s+l}$ for $t \in \{k+1, k+2, \cdots, n\}$, resulting in a contradiction. Hence, all lists are self sorted after applying n-sorters. □

A.2 Proof of Lemma 6.3.2

Proof. First, we show that the first connections of adjacent two sorters belong to either the same list or adjacent two lists. Let (j, t_1) and $(j + l, t_2)$ be the first connections of adjacent two sorters $S1$ and $S2$, respectively, where (j, t) denotes wire t in list j. If $l > 1$, the connection of $S1$ in list $j + l - 1$ should be wire m; otherwise, $S2$ would have a valid connection in list $j + l$. For lists j to $j + l - 2$, only wires m in each list are connected by $S1$, since wire m can be connected to the preceding list only by a $(m - 1)$-spaced sorter. Hence, $S1$ is the last $(m - 1)$-spaced sorter in stage 1 and $S2$ does not exist. Similarly, we can show that the last connections of adjacent two sorters $S1$ and $S2$ belong to either the same list or adjacent two lists. This gives us a total of four cases as shown in Fig. 7.5, where $b \geq a + 1$ for Fig. 7.5(a)-(c), and $b \geq a$ for Fig. 7.5(d) such that $S1$ and $S2$ have a size of at least two.

If m is prime, no adjacent two sorters belong to case IV, which is equivalent to showing that m is a composite number if case IV in Fig. 7.5 exists. Assume two adjacent sorters $S1$ and $S2$ belong to case IV. Let the first connection of $S1$ be (j, m) and the last connection of $S2$ be $(j + p, 1)$. The last connection of $S1$ satisfies $(k + 1)p \equiv 0 \mod m$. We have $m \mid (k + 1)p$. Since case IV is not possible in the first stage, we have $p < m$. Since two adjacent sorters connect two adjacent wires in at least one list, we have $p > 1$. If $k = 0$, $S1$ would connect the last and first wires of adjacent lists, respectively, in which case $S2$ does not exist. We have $1 < k + 1 < m$. So m should have a proper factor dividing $k + 1$ or p. Hence, m is a composite number. □

A.3 Proof of Theorem 6.3.1

Proof. The theorem can be proved by induction on i. In stage 1, m-sorters are applied on corresponding wires of all m lists. According to Lemma 7.3.1, the outputs of each list are sorted. Assume any two adjacent wires s and $s+1$ in list j are sorted after stage $i-1$, $x_{j,s}^{(i-1)} \leq x_{j,s+1}^{(i-1)}$ for $1 \leq j \leq n$ and $1 \leq s \leq m-1$. We will show that $x_{j,s}^{(i)} \leq x_{j,s+1}^{(i)}$ for $1 \leq j \leq n$ and $1 \leq s \leq m-1$.

According to Lemma 7.3.2, for a prime m, there are three cases of two adjacent sorters $S1$ and $S2$ as shown in Fig. 7.5(a)-(c).

1. For case I, let $y_{j,1}^{(i-1)}$ and $y_{j,2}^{(i-1)}$ be the two adjacent wires in list j connected by adjacent two sorters in stage $i-1$ for $a \leq j \leq b$. According to Lemma 7.3.1 ($n = 2$), the outputs of each list are sorted.

2. For case II, there is an additional single wire $y_{b+1}^{(i-1)}$ connected by $S2$. If $y_{b+1,1}^{(i-1)} = 1$, we have $y_{b+1,1}^{(i)} = 1$. The last connection of $S2$ can be removed without changing the order of others in $S2$. $S1$ and the revised $S2$ reduce to case I and the outputs are sorted according to Lemma 7.3.1. If $y_{b+1,1}^{(i-1)} = 0$, we have $y_{b,1}^{(i-1)} = 0$. This is because they are connected by the same sorter in stage $i-1$. Then, we have $y_{a,1}^{(i)} = y_{a,2}^{(i)} = 0$, which are sorted outputs in list a. Remove $y_{b+1,1}^{(i-1)}, y_{b,1}^{(i-1)}, y_{a,1}^{(i)}$, and $y_{a,2}^{(i)}$, the remaining of $S1$ and $S2$ reduce to a smaller configuration of case II. With recursively applying the above approach, $S1$ and $S2$ either reduce to a smaller case I or a single wire, both of which gives sorted outputs.

3. For case III, there is an additional single wire $y_{a-1,m}^{(i-1)}$ connected by the first sorter. Similarly, the two sorters can be reduced to either a case I or a smaller

configuration of case III and the outputs of two adjacent wires in each list are sorted.

Assume all lists are self-sorted after stage $i-1$, we have $x_{j,1}^{(i-1)} \leq \cdots \leq x_{j,m}^{(i-1)}$ for $1 \leq j \leq n$. For stage $1 \leq i \leq \lceil \frac{m}{2} \rceil$, all wires in lists $j = 2, \cdots, n-1$ have connections with some sorters. We have $x_{j,k}^{(i)} \leq x_{j,k}^{(i)}$ for $j = 2, \cdots, n-1$ and $k = 1, \cdots, m-1$. Hence, lists $j = 2, \cdots, n-1$ are self-sorted after stage i. For list 1, $x_{1,i-1}^{(i-1)} \leq x_{2,1}^{(i-1)}$ and $x_{1,i-1}^{(i-1)} \leq x_{1,i}^{(i-1)}$, we have $x_{1,i-1}^{(i)} \leq x_{1,i}^{(i)}$. We have $\langle x_{1,1}^{(i)}, x_{1,2}^{(i)}, \cdots, x_{1,i-1}^{(i)} \rangle$, since list 1 is self-sorted after stage $i-1$ and $x_{1,k}^{(i-1)} = x_{1,k}^{(i)}$ for $k = 1, \cdots, i-1$. We also have $x_{1,i}^{(i)}, x_{1,i+1}^{(i)}, \cdots, x_{1,m}^{(i)} \rangle$. Hence, list 1 is self-sorted after stage i, $x_{1,1}^{(i)}, x_{1,i+1}^{(i)}, \cdots, x_{1,m}^{(i)} \rangle$. Due to symmetry, list n is also self-sorted after stage i, $x_{m,1}^{(i)}, x_{m,i+1}^{(i)}, \cdots, x_{n,m}^{(i)} \rangle$.

To prove that the outputs of n sorted lists $\langle x_{j,1}^{(\lceil \frac{m}{2} \rceil)}, \cdots, x_{j,m}^{(\lceil \frac{m}{2} \rceil)} \rangle$ for $j = 1, \cdots, n$ after stage $\lceil \frac{m}{2} \rceil$ are combined as a single sorted list in stage $\lceil \frac{m}{2} + 1 \rceil$, we need to show that $x_{j,\frac{m+1}{2}}^{(\lceil \frac{m}{2} \rceil + 1)} \leq x_{j,\frac{m+1}{2}+1}^{(\lceil \frac{m}{2} \rceil + 1)}$ for $j = 1, \cdots, n-1$ and $x_{j,\frac{m+1}{2}-1}^{(\lceil \frac{m}{2} \rceil + 1)} \leq x_{j,\frac{m+1}{2}}^{(\lceil \frac{m}{2} \rceil + 1)}$ for $j = 2, \cdots, n$. Since $x_{j,\frac{m+1}{2}}^{(\lceil \frac{m}{2} \rceil)} \leq x_{j,\frac{m+1}{2}+1}^{(\lceil \frac{m}{2} \rceil)}$ and $x_{j,\frac{m+1}{2}}^{(\lceil \frac{m}{2} \rceil)} \leq x_{j+1,1}^{(\lceil \frac{m}{2} \rceil)}$, we have $x_{j,\frac{m+1}{2}}^{(\lceil \frac{m}{2} \rceil + 1)} \leq x_{j,\frac{m+1}{2}+1}^{(\lceil \frac{m}{2} \rceil + 1)}$ for $j = 1, \cdots, n-1$. Similarly, we have $x_{j,\frac{m+1}{2}-1}^{(\lceil \frac{m}{2} \rceil + 1)} \leq x_{j,\frac{m+1}{2}}^{(\lceil \frac{m}{2} \rceil + 1)}$ for $j = 2, \cdots, n$ □

A.4 Proof of Lemma 6.3.3

Proof. In stage $i-1$, there are n^{i-1} sorted lists of n values with respect to each q ($q = 1, \cdots, n^{p-i}$). Since the outputs of each merging network are sorted after stage $i-1$, we can replace each merging network by an n^i-sorter. According to Lemma 7.3.1, the outputs of each new formed list after stage i are sorted, $x_{j,q}^{(i)} \leq x_{j,n^{p-i-1}+q}^{(i)} \leq x_{j,(n-1)n^{p-i-1}+q}^{(i)}$ for $j = 1, \cdots, n^i$. Since the corresponding wires in the new lists are connected by the same n^i-sorter in stage $i-1$, we have $x_{j,q}^{(i)} \leq x_{j+1,q}^{(i)}$

for $j = 1, \cdots, n^i - 1$. Hence, $r^{(i)}_{j,q} \geq r^{(i)}_{j+1,q}$ for $j = 1, \cdots, n^i - 1$.

For $r^{(i)}_{s,q} = n > r^{(i)}_{s+1,q} \geq \cdots \geq r^{(i)}_{s+l,q} > 0 = r^{(i)}_{s+l+1,q}$ for $l \leq n$, it is equivalent to prove that $x^{(i)}_{j+n,q} = 1$ if $x^{(i)}_{j,(n-1)n^{p-i-1}+q} = 1$ for $j \in \{1, \cdots, n^i - n\}$. For any $q \in \{1, \cdots, n^{p-1-i}\}$ in stage i, there are n^i lists of n values. Suppose $x^{(i)}_{j,(n-1)n^{p-i-1}+q} = 0$ for $j \leq s$ and $x^{(i)}_{s+1,(n-1)n^{p-i-1}+q} = 1$. If t $(t \leq s)$ zeros of $x^{(i)}_{j,(n-1)n^{p-i-1}+q}$ are from the same list of the original n sorted lists, there are at most $t + 1$ zeros of $x^{(i)}_{j,q}$ from that same list. Since $x^{(i)}_{j,(n-1)n^{p-i-1}+q} = 0$ for $j \leq s$ are from at most n original lists, there are at most $s + n$ zeros in $x^{(i)}_{j,q}$, implying that $x^{(i)}_{s+n,q} = 1$. Hence, $x^{(i)}_{j+n,q} = 1$ if $x^{(i)}_{j,(n-1)n^{p-i-1}+q} = 1$ for $j \in \{1, \cdots, n^i - n\}$. □

A.5 Proof of Theorem 6.3.2

Proof. In stage 1, all outputs with respect to the operation of the same Alg. 5 are sorted. For any $q \in \{1, \cdots, n^{p-1-i}\}$ in stage i, according to Lemma 7.3.3, at most n consecutive lists are not full of zeros. All preceding lists are all-zero lists and all following lists are all-one lists. Hence, the combining network in stage i is to sort n lists of n values, which is reduced to Alg. 5. In stage $p - 1$, we have $q = 1$ and the single sorted list, $\langle x^{(i)}_{1,q}, x^{(i)}_{1,n^{p-i-1}+q}, \cdots, x^{(i)}_{1,(n-1)n^{p-i-1}+q}, x^{(i)}_{2,q}, x^{(i)}_{2,n^{p-i-1}+q}, \cdots, x^{(i)}_{2,(n-1)n^{p-i-1}+q}, \cdots, x^{(i)}_{n^i,q}, x^{(i)}_{n^i,n^{p-i-1}+q}, \cdots, x^{(i)}_{n^i,(n-1)n^{p-i-1}+q} \rangle$, contains n^p values, implying all inputs are sorted as a single list. □

Vita

Feng Shi received his B.E. and M.E. degrees from electrical engineering and microelectronics institute of Tsinghua University, Beijing, China in 2006 and 2009, respectively. He is currently pursuing the Ph.D degree in electrical and computer engineering at Lehigh University, Bethlehem, Pennsylvania.

His research interest lies in delay modeling and crosstalk avoidance coding for current CMOS technologies, and threshold architectures for finite field operations in nanotechnology.