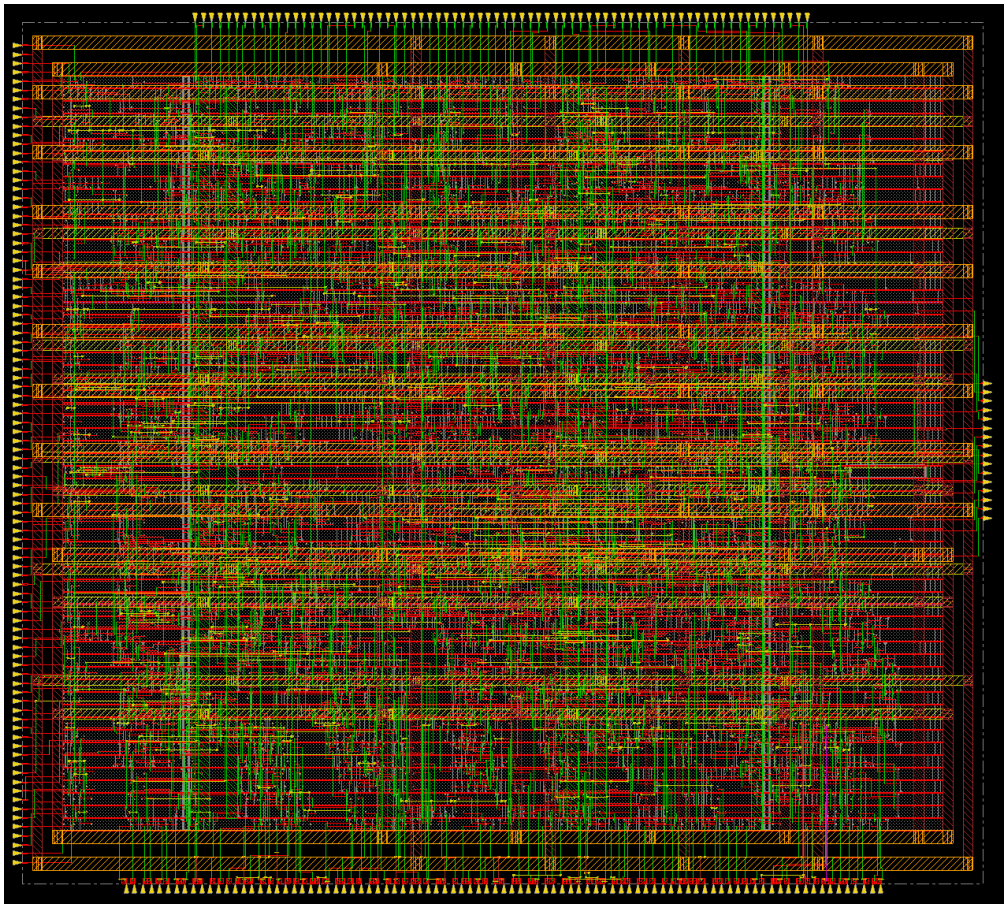


Sloppy Selection Networks for Fast Energy-Efficient Decoding

Master's thesis in Embedded Electronic System Design

HARSHIT KUMAR VENKATESH

Department of Microtechnology and Nanoscience
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026

Master's thesis 2026

Sloppy Selection Networks for Fast Energy-Efficient Decoding

HARSHIT KUMAR VENKATESH

Department of Microtechnology and Nanoscience
Chalmers University of Technology
Gothenburg, Sweden 2026

Sloppy Selection Networks
for Fast Energy-Efficient Decoding
HARSHIT KUMAR VENKATESH

Supervisor: Lars Svensson, Department of Microtechnology and Nanoscience
Examiner: Per Larsson Edefors, Department of Microtechnology and Nanoscience

Master's Thesis 2026
Department of Microtechnology and Nanoscience
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Layout of the proposed sloppy selection network that selects two smallest elements out of thirty two, that has four SBT networks.

Typeset in LaTeX
Gothenburg, Sweden 2026

Sloppy Selection Networks
for Fast Energy-Efficient Decoding
HARSHIT KUMAR VENKATESH
Department of Microtechnology and Nanoscience
Chalmers University of Technology

Abstract

This thesis investigates the possibility of reducing the impact of hardware constraints of decoder implementations employing modern soft-decision decoding schemes such as ORBGRAND, GRAND, and Chase-class algorithms for FEC. Sorting networks constitute a major hardware bottleneck in the implementation of such decoders due to their area, power, and delay requirements. To address this problem, sloppy selection is explored as an alternative to conventional sorting, and a generalized architecture for implementing sloppy selection networks is proposed.

The proposed architecture provides a configurable trade-off between selection "sloppiness" and hardware complexity. The proposed sloppy selection networks demonstrate a slight degradation in decoding performance while showcasing significant reductions in impact on hardware constraints, achieving up to 96% reduction in area and power consumption, along with up to 80% reduction in delay compared to conventional sorting-based implementations. Furthermore, an MSB-first comparator architecture was introduced to further optimize the comparator logic within the sloppy selection networks. The proposed comparator achieved an additional reduction of approximately 70% in delay and nearly 90% reduction in EDP, while incurring only a modest area increase of approximately 25%.

The obtained results strongly indicate the viability of the proposed sloppy selection networks for practical decoder implementations. If comparable decoding performance can be achieved relative to implementations using accurate sorting networks, the proposed architectures could have a significant impact on the design of 5G and 6G decoder hardware.

Keywords: Forward Error Correction, Decoding, Sloppy Selection, GRAND, Chase, Logic Optimization, Binary Trees, Selection Networks, Reliability, Hardware Considerations

Acknowledgements

I would like to express my sincere gratitude to my academic supervisor, Lars Svensson, and my industrial advisor, Christoffer Fougstedt, whose guidance and support were instrumental in advancing this research. I would also like to thank my examiner, Per Larsson-Edefors, for his valuable support throughout this work.

Furthermore, I am deeply grateful to my family for making this education possible. Finally, I would like to thank all my friends for their encouragement and support throughout the course of this research.

Harshit Kumar Venkatesh, Gothenburg, June 2026

Contents

1

Introduction

Handling noise in telecommunications systems is a fundamental challenge that directly impacts system performance. In some decoding algorithms, this issue is addressed by identifying the least reliable bits in a received packet. Some prominent examples include the GRAND [1], ORBGRAND [2] and Chase [3, 4, 5] classes of algorithms. This identification is commonly implemented using sorting or selection networks, which can incur significant hardware complexity. In fact, the sorting network forms the bottleneck in terms of power and area [6].

This work aims to improve the hardware efficiency of such decoders by exploiting the fact that reliability information is inherently noisy. By performing sloppy, sorting or selection, it may be possible to significantly reduce hardware complexity with only a limited impact on decoding performance.

The effects of sloppy sorting have been previously explored on non binary decoders [7], where the input size of the sorter is reduced by considering only a part of the soft information provided by the decoder, ultimately making the selection sloppy. However, we hypothesize that this approach remains inefficient, and that substantially greater gains can be achieved by employing sloppy selection networks that do not reduce input constraints and sloppily select but still maintain good enough performance. Further there are no such hardware implementations found for binary decoders that implement decoding algorithms like Chase or ORBGRAND.

1.1 Related Work

There have been attempts to reduce hardware complexity while maintaining acceptable error-correction performance on non binary decoders [7]. This idea can be directly applied to widely used decoding algorithms that decode binary codes, such as GRAND [1], ORBGRAND [2], and Chase class of algorithms [3, 4, 5], offering a promising path toward improved hardware efficiency without significantly degrading decoding accuracy.

Despite this potential, we are not aware of any prior work that has explored sloppy selection for binary decoding algorithms as a means to reduce hardware requirements. Some studies have examined the hardware implications of selection networks [8], which can serve as a useful reference for designing approximate selection networks aimed at lowering hardware costs in decoding.

Previous efforts have also focused on pruning and parallelizing sorting networks to reduce computational complexity and increase throughput [9, 10, 11]. However, some

of these approaches still produce fully sorted outputs, even though strict ordering is often unnecessary for decoding. Additionally, work on standardized pruning of sorting networks [12] provides relevant insights that can further support the development of approximate selection networks in this context.

1.2 Purpose and Goal

The primary goal of this thesis is to propose a general architecture to implement sloppy selection in hardware. This network can be used in a decoder to correct noisy codes. This thesis will also quantitatively evaluate the resulting trade-offs between hardware requirements and error-correction performance.

At the circuit level, the hardware evaluation focuses on logic depth, circuit area, throughput, and power consumption, listed in order of decreasing significance. At the decoder level, the evaluation considers total hardware complexity, power consumption, and output bit error rate (BER) as a function of the Signal-to-Noise Ratio (SNR) of the received signal, throughput of the decoder.

The investigation aims to determine whether sloppy selection networks can provide tangible benefits in reducing decoding energy consumption per selection, decoder area, and network depth, thereby enabling higher operating frequencies, while maintaining sufficiently good decoding performance for decoders that utilize selection networks, and how these benefits depend on the operating SNR. The key challenge lies in selecting an appropriate baseline selection (or) sorting network that is parallel enough to be implemented on hardware and progressively approximating this network to reduce hardware cost while maintaining acceptable error-correction performance. Successfully addressing this challenge would make such architectures suitable for integration into practical telecommunications systems.

1.3 Thesis Outline

This report begins with a technical background chapter (Chapter 2) that introduces sorting algorithms and related techniques, emphasizing their operational principles and hardware implementations. Building on this foundation, the chapter further examines the use of subsets or approximations of sorting networks for selection, introduces binary trees, and discusses the role and significance of selection in decoding, along with a short review of existing decoder architectures that use these networks.

The methods chapter (Chapter 3) then formalizes the concept of sloppiness in selection networks and outlines the methodologies used to evaluate and construct the proposed designs. Building directly on these methods, the design chapter (Chapter 4) presents two key contributions; a novel compare-and-swap operator and a novel architecture for constructing sloppy selection networks.

The results chapter (Chapter 5) provides a critical evaluation of the performance and limitations of the proposed architecture. Finally, the report concludes with a summary of the key findings and contributions.

2

Technical Background

This chapter presents a concise introduction to sorting, selection, and binary tree networks, which form the foundation for the design of the proposed architecture. Furthermore, it examines the pivotal role of selection in contemporary decoding algorithms.

2.1 Sorting Networks

To understand sorting networks, we begin by defining sorting in the context of numerical data. Sorting is the process of arranging a set of numbers in either ascending or descending order [13]. Although at first glance sorting may appear simple and straightforward, it becomes computationally intensive when performed systematically, especially as the number of elements increases. In practice, sorting must be carried out in a structured and deterministic manner to ensure correctness. Additionally, sorting inherently requires memory, which becomes more apparent when considering how humans perform sorting tasks.

Over the years, numerous algorithms have been developed to perform sorting systematically. Prominent examples of comparison-based algorithms include Bubble Sort [13], Merge Sort [14, 15], and Bitonic Sort [14, 15]. There also exists a separate class of sorting algorithms that do not rely on direct comparison of variables, such as Bucket Sort and Radix Sort [13]. However, this class of algorithms is beyond the scope of this work. Therefore, the discussion that follows focuses exclusively on comparison-based sorting algorithms.

Different sorting algorithms exhibit distinct characteristics. For example, Bubble Sort is highly sequential and relatively slow, whereas Bitonic Sort is among the fastest and most parallelizable algorithms [16, 17]. Despite these differences, all comparison-based algorithms are described based on a fundamental operation that compares two elements and conditionally swaps them. This operation is referred to as the Compare-and-Swap (CaS) operator.

A CaS operator consists of a comparator that evaluates two input values and generates control signals for two multiplexers (MUXs). One multiplexer selects the larger of the two inputs, while the other selects the smaller one. In this way, the outputs are always ordered according to the desired sorting direction. The topology of a typical CaS unit is illustrated in Figure 2.1.

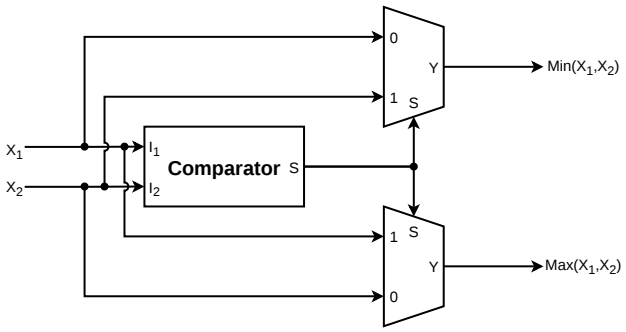


Figure 2.1: Illustration of the topology of a CaS unit (figure inspired from [18])

Now that the fundamental building block has been established, sorting networks can be constructed in digital hardware. One of the simplest implementations is based on the Bubble Sort algorithm. In this approach, the input array is repeatedly traversed, comparing each element with its immediate neighbor. If the comparison condition is violated, the elements are swapped. The condition depends on the desired ordering: for ascending order, a swap occurs when the preceding element is greater than the following element, while for descending order the condition is reversed. Multiple passes are performed over the array until a complete pass occurs with no swaps, which defines the termination condition [13].

From a digital logic perspective, this algorithm can be implemented as a sequential circuit using a single CaS operator, a First-In-First-Out (FIFO) shift register, and a clock signal to repeatedly process the data. Figure 2.2 illustrates such an implementation. A key drawback of this approach is its sequential nature, which limits performance and makes it unsuitable for time-critical applications.

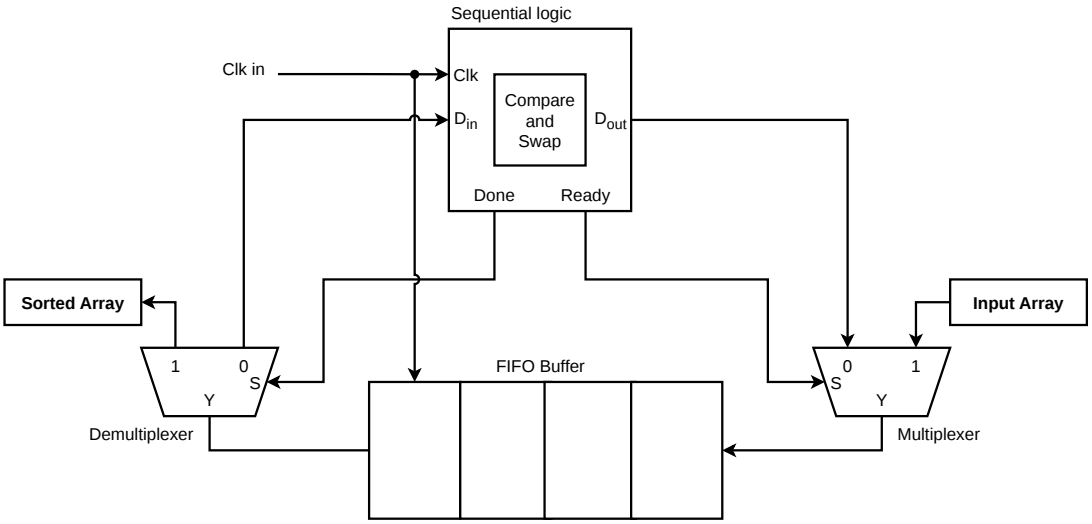


Figure 2.2: Simplified digital implementation of Bubble Sort. The input is loaded into the FIFO when the 'Ready' signal is high. Once loaded, the system performs iterative compare-and-swap operations until no swaps occur in a full pass, at which point the 'Done' signal is asserted.

It is worth noting that while Bubble Sort can be implemented in a parallel, combinational manner using multiple CaS units, such an approach requires a significant increase in hardware resources. Figure 2.3 shows such a parallel implementation of bubble sort.

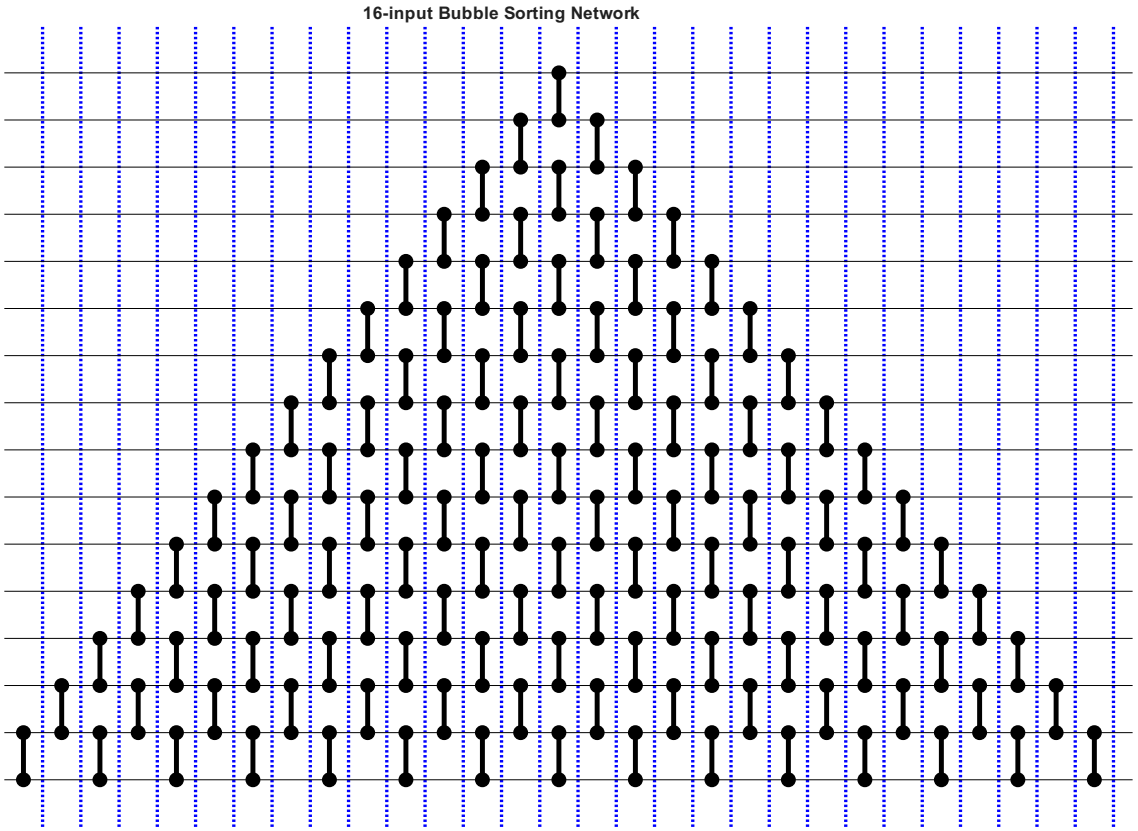


Figure 2.3: Schematic of a 16 input bubble sorting network. Each dumbbell indicates a CaS operator and each stage is separated by a dotted blue line.The inputs are on the left end and the sorted outputs are on the right end.

To address these limitations, algorithms inherently designed for parallelization can be employed to build faster sorting networks. One such algorithm is Bitonic Sort. This approach divides the input array into smaller subsets, which are sorted independently to form alternating ascending and descending sequences, referred to as bitonic sequences. These sequences are then merged in a structured manner, and the process is repeated iteratively until the final two bitonic sequences are merged to produce a fully sorted array [15, 14].

As illustrated in Figure 2.4, a bitonic sorting network consists of multiple stages of CaS operators arranged to enable parallel comparisons. Each stage performs a specific set of compare-and-swap operations, progressively enforcing global ordering.

Bitonic sorters achieve high performance because the bitonic sorting algorithm is inherently designed for parallel execution. However, this advantage comes at the cost of increased area and power consumption [6]. This trade-off is also reflected in how the hardware scales with the number of inputs. The area complexity of a

Bubble Sort implementation increases linearly with the number of elements, whereas the area complexity of a Bitonic sorting network grows on the order of $N \log_2(N)$ [6]. Similarly, the timing complexity of a Bubble Sort scales as $O(N)$, while that of a Bitonic sorter scales as $O(\log_2^2(N))$ [13], where N denotes the number of elements being sorted. Consequently, although Bubble Sort requires less area and power, it exhibits longer sorting latency. In contrast, the Bitonic sorter consumes significantly more area and power but achieves substantially lower sorting time.

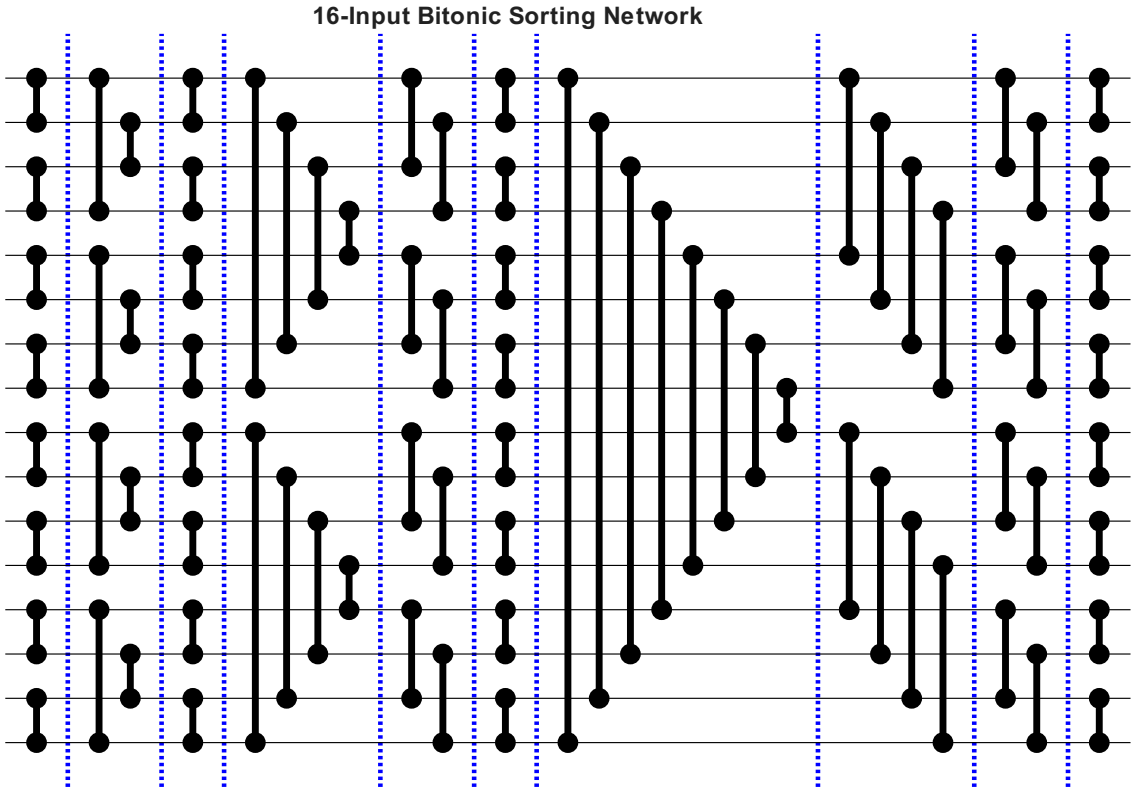


Figure 2.4: Schematic of a 16 input bitonic sorting network. Each dumbbell indicates a CaS operator and each stage is represented by a dotted blue line.The inputs are on the left end and the sorted outputs are on the right end.

There are also other parallel sorting algorithms, such as Odd-Even Merge Sort, which exhibit different design constraints compared to Bitonic Sort [8]. These trade-offs between different implementations further motivate the architectural choices explored in Chapter 4.

2.2 Selection Networks

Selection networks can be regarded as a simplified class of sorting networks. Rather than producing a fully ordered sequence, a selection network addresses a related problem: selecting the top k values from a set of N input elements [8].

A selection network that extracts the two largest elements from eight inputs may be denoted as a top$(2, 8)$ network. More generally, a network that selects the k largest

elements from N inputs is denoted as a top(k, N) network. Also, it can be noted that the same network that gives us top k out of N also can give us the smallest k out of N just by swapping the comparison sign on the comparator of the CaS, and this can be denoted as a bottom(k, N) network.

The principal distinction between sorting and selection networks lies in the output requirement. In a sorting network, all output elements must be fully ordered, whereas in a selection network, only the set of top k elements is required, and their relative ordering is not significant [8]. This relaxation enables more efficient implementations in terms of both hardware complexity and computational depth.

Any N-input sorting network can be used as a top(k, N) selection network; however, this approach performs many unnecessary comparisons. A more cost-effective top(k, N) selection network can be obtained by pruning CaS operators that do not affect the identification of the top k elements, while preserving the network's functionality [8]. This pruning process typically begins at the stages closest to the output and proceeds progressively towards the stages nearer to the inputs. In the case of the bitonic sorting network shown in Figure 2.4, this corresponds to pruning from right to left. Figure 2.5 illustrates the pruning of a 16-input bitonic sorting network to obtain a top$(4, 16)$ selection network.

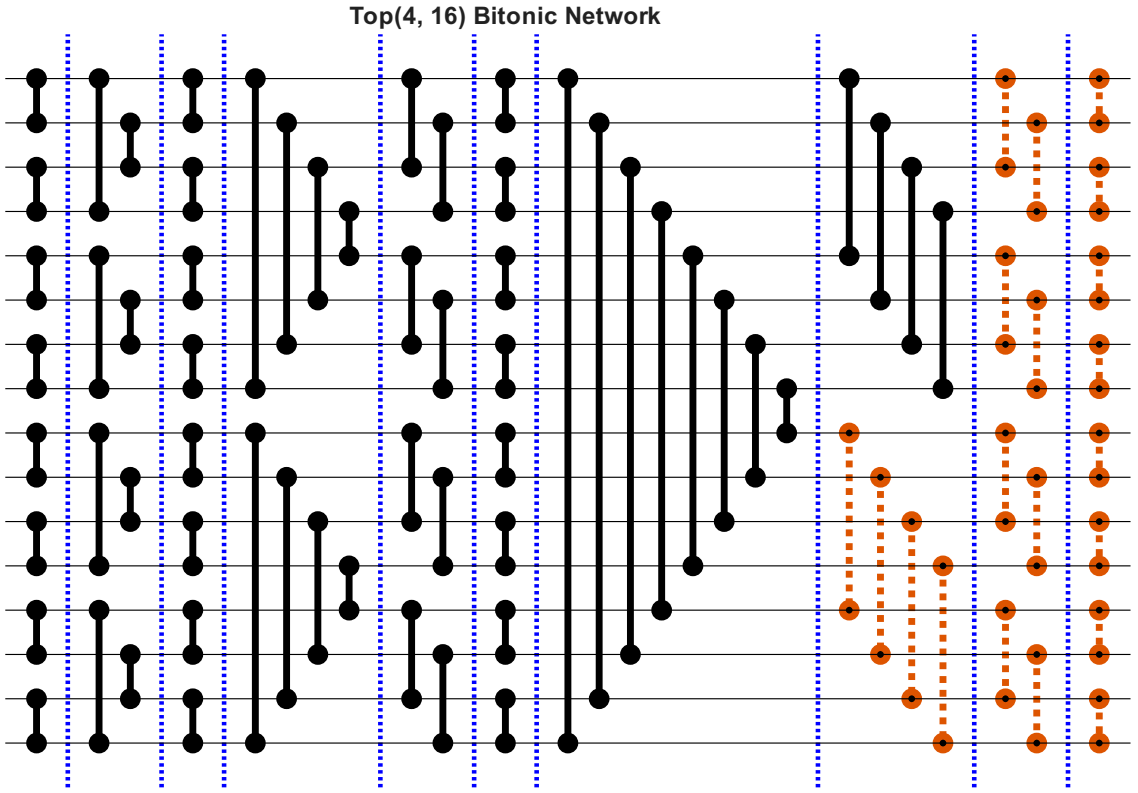


Figure 2.5: Schematic of a *top*$(4, 16)$ bitonic selection network. The highlighted red elements indicate pruned CaS operators (figure inspired from [8]).

It can be observed that selection networks are significantly less complex than their sorting network counterparts, both in terms of circuit area and network depth. These reductions can directly translate to lower power consumption and reduced delay,

respectively [8]. Consequently, selection networks form a key component of the proposed architecture and constitute one of its two primary building blocks.

2.3 Binary Tree Networks

A binary tree is a hierarchical data structure in which each parent node is connected to at most two child nodes [19]. In this work, we consider a specific class of binary trees in which every parent node has exactly two children; such trees can be referred to as full binary trees. Figure 2.6 illustrates a four-level full binary tree in which all possible child nodes are present. Similarly, Figure 2.7 shows how such a binary tree structure can be constructed using CaS operators.

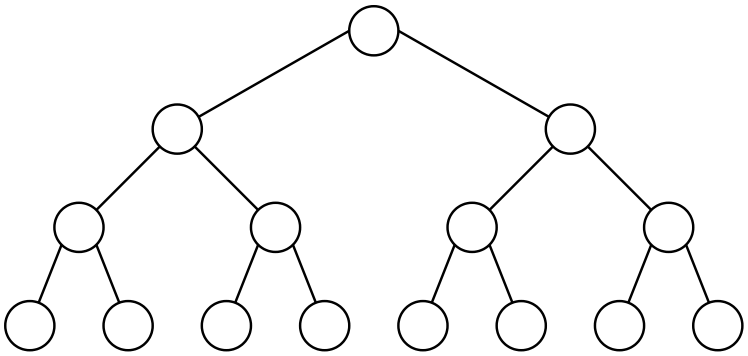


Figure 2.6: Schematic of a four level binary tree with all child nodes possible

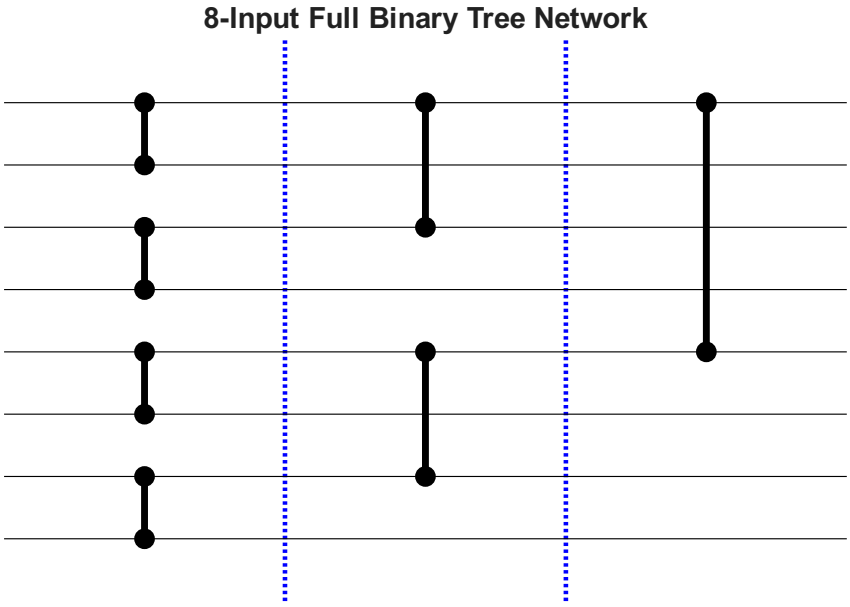


Figure 2.7: An eight input full binary tree visualised using CaS operators

Full binary tree structures can be utilized to implement minimum or maximum finder networks, enabling fast and highly parallel determination of the global minimum or maximum from a set of input values [20, 21]. In such implementations, each node of

the binary tree is replaced by a Compare-and-Select (CaSe) operator. Equivalently, the CaS operators shown in Figure 2.7 may be replaced with CaSe operators to realize a minimum or maximum finder network.

A CaSe operator can be derived from a Compare-and-Swap (CaS) operator by removing one of the multiplexers. Figure 2.8 illustrates the topology of a CaSe unit. Depending on the comparator operation, the resulting binary tree network can function either as a global minimum finder or a global maximum finder. If the comparator performs a greater-than-or-equal-to operation, the multiplexer selects the larger of the two inputs, and the CaSe operator outputs only the maximum value, resulting in a global maximum finder. Conversely, if the comparator performs a less-than-or-equal-to operation, the operator outputs the smaller of the two inputs, thereby forming a global minimum finder.

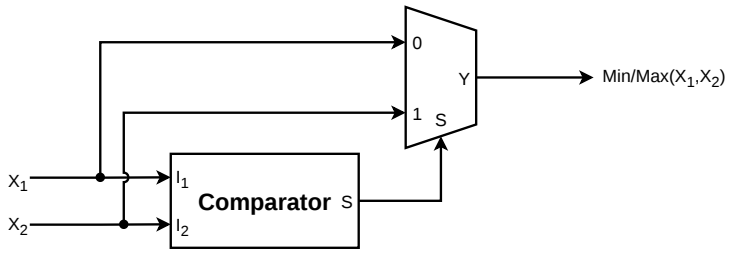


Figure 2.8: Illustration of the topology of a CaSe unit (figure inspired from [18])

Figure 2.9 illustrates the operation of a full binary tree configured as an 8-input minimum finder, where the global minimum is obtained through successive comparisons across the tree structure.

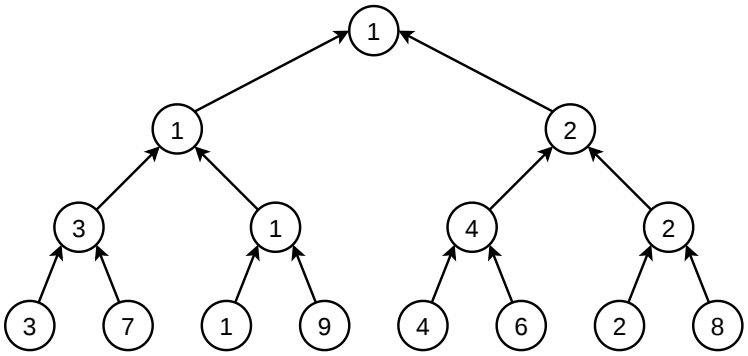


Figure 2.9: Illustration of a 8-input binary tree minimum finder. Each pair of arrows that leads to a single node represents a single CaSe unit. Following that, in this picture we have seven CaSe units

These binary tree networks exhibit an area complexity of $O(N)$ and a time complexity of $O(\log_2(N))$ [21], where N denotes the number of inputs. Notably, for a given N, these structures are both smaller and faster than their bitonic sorting network counterparts.

Consequently, these binary tree structures constitute the second key component of the proposed architecture.

2.4 Number Representation in a digital system

In digital circuits, numbers are represented using binary digits (bits), where each bit can take either a value of 1 or 0. Different number representations are used to satisfy various requirements, including computational efficiency, memory optimization, numerical precision, hardware implementation efficiency, and reliable data processing. In this section, we focus on the representation of integers, specifically unsigned integers, since they will be used throughout the design presented in this work.

An unsigned integer is a binary number representation used to store non-negative integer values in digital systems. In this representation, all available bits contribute to the magnitude of the number, and no bit is reserved for indicating a sign. Consequently, unsigned integers can represent only zero and positive integer values.

An n-bit unsigned integer consists of n binary digits, indexed from the least significant bit (LSB) to the most significant bit (MSB). The LSB, located at position $i = 0$, has a weight of 2^0, while the MSB, located at position $i = n - 1$, has the highest weight of 2^{n-1}. Unlike signed number representations, the MSB in an unsigned integer contributes to the magnitude of the number rather than indicating its sign.

For an n-bit unsigned integer, the numerical value is determined by the weighted sum of its binary digits:

$$N = \sum_{i=0}^{n-1} b_i 2^i \tag{2.1}$$

where b_i represents the binary digit (0 or 1) at position i.

Since all bits are used to represent the magnitude of the number, the range of values that can be represented by an n-bit unsigned integer is given by:

$$0 \leq N \leq 2^n - 1. \tag{2.2}$$

For example, an 8-bit unsigned integer can represent values from 0 (all bits equal to 0) to 255 (all bits equal to 1). The binary number 10110110_2 corresponds to the decimal value 182, which can be computed as follows:

$$(1\times2^7)+(0\times2^6)+(1\times2^5)+(1\times2^4)+(0\times2^3)+(1\times2^2)+(1\times2^1)+(0\times2^0) = 182. \tag{2.3}$$

3

Method

This section presents the methodologies employed to develop and evaluate the concepts and circuits investigated in this work. The discussion begins with an evaluation of the selection networks, followed by a description of the netlist generation in Hardware Description Language (HDL) and their subsequent synthesis into digital logic. Finally, the approaches used for estimating power consumption and area are detailed.

3.1 Evaluating the Selection Networks

The selection networks were evaluated in two phases. In the first phase, a functional equivalent of the selection network was implemented in MATLAB. In the second phase, the synthesized digital logic was verified using the same test vectors generated by the MATLAB model.

Two MATLAB functions were developed. One mimicked the proposed sloppy selection network architecture discussed in Chapter 4, and the other represented an ideal selection network. These functions were used to perform Monte Carlo simulations with random input sets generated as described in Section 3.1.1. Each input set was denoted as I. For each simulation, the ideal network produced an output set E, while the proposed network produced an output set A. Sloppiness was calculated for each iteration as described in Section 4.2, and the average sloppiness across all iterations was then used to compare different networks following the proposed architecture.

In addition to sloppiness, the Monte Carlo simulations also evaluated whether the selection networks correctly identified all bit flips caused by channel noise. After propagating the input vectors through the networks, the least reliable candidates were identified and compared to the actual flipped bits in the noisy demodulated bitstream. If all flipped bits were captured by the network, the test was designated as a pass; otherwise, it was considered a failure. This comparison was performed by matching the indices of the flipped bits with the indices of the least reliable candidates identified by the network.

A total of 10^6 Monte Carlo iterations were performed for each simulation. This number was chosen because MATLAB could efficiently handle this scale while still providing results within a reasonable time. It also ensured that statistical variations were low and provided a reliable estimate of the network's performance.

Finally, the input and output vectors from the simulations were logged into separate

test vector files. These files were later used for functional and post-synthesis verification of the digital logic. All implemented selection networks were tested using the stored input vectors to confirm that they produced identical output vectors. This ensured that functionality was preserved throughout the design and implementation pipeline.

3.1.1 Input Vector Generation using BPSK

The input vectors for the Monte Carlo simulation were generated using a Binary Phase Shift Keying (BPSK) modulation [22] simulation implemented in MATLAB. In the context of channel coding, BPSK is also commonly referred to as Binary Input Additive White Gaussian Noise (BI-AWGN) channel. BPSK was chosen because it represents the simplest modulation scheme and can be extended to more complex schemes for generating input vectors for the selection networks. In addition, in the previously mentioned decoding algorithms, a reliability parameter derived from a complex modulation scheme was passed through a sorting network to identify the least reliable bits.

Random binary sequences were generated and modulated using the BPSK scheme. The modulated symbols were transmitted through a simulated channel that added white Gaussian noise, resulting in a noisy signal with an SNR of 6 dB. This SNR value was selected based on empirical observations, as it provided a suitable spread of points in the constellation diagram for analysis. The recieved signal was subsequently demodulated. This process mimicked a real communication system, where transmitted bits could be flipped due to channel noise. Figure 3.1 illustrates a block diagram of the BPSK simulation.

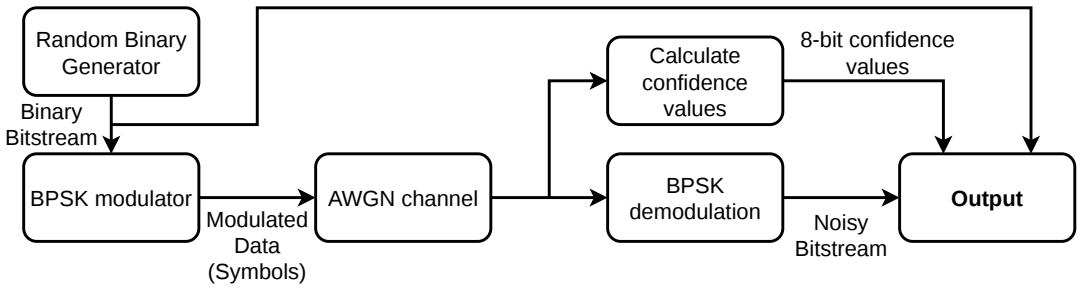


Figure 3.1: Illustration of the implemented BPSK function in MATLAB for generating input vectors of the Monte Carlo simulation. AWGN refers to Additive White Gaussian Noise.

Confidence values for each received symbol were computed by calculating the distance of the noisy modulated point to the midpoint between the ideal locations of the two BPSK symbols along the in-phase axis. Figure 3.2 illustrates this measurement process. These confidence values indicate the likelihood that a received symbol matches the originally transmitted symbol. In some cases, the computed distance may exceed the theoretical maximum value of 1. To address this, all values were clipped to the range between 0 and 1, since any confidence value above 1 corresponds

to a symbol that has a very high probability of being correct for a sufficiently high SNR.

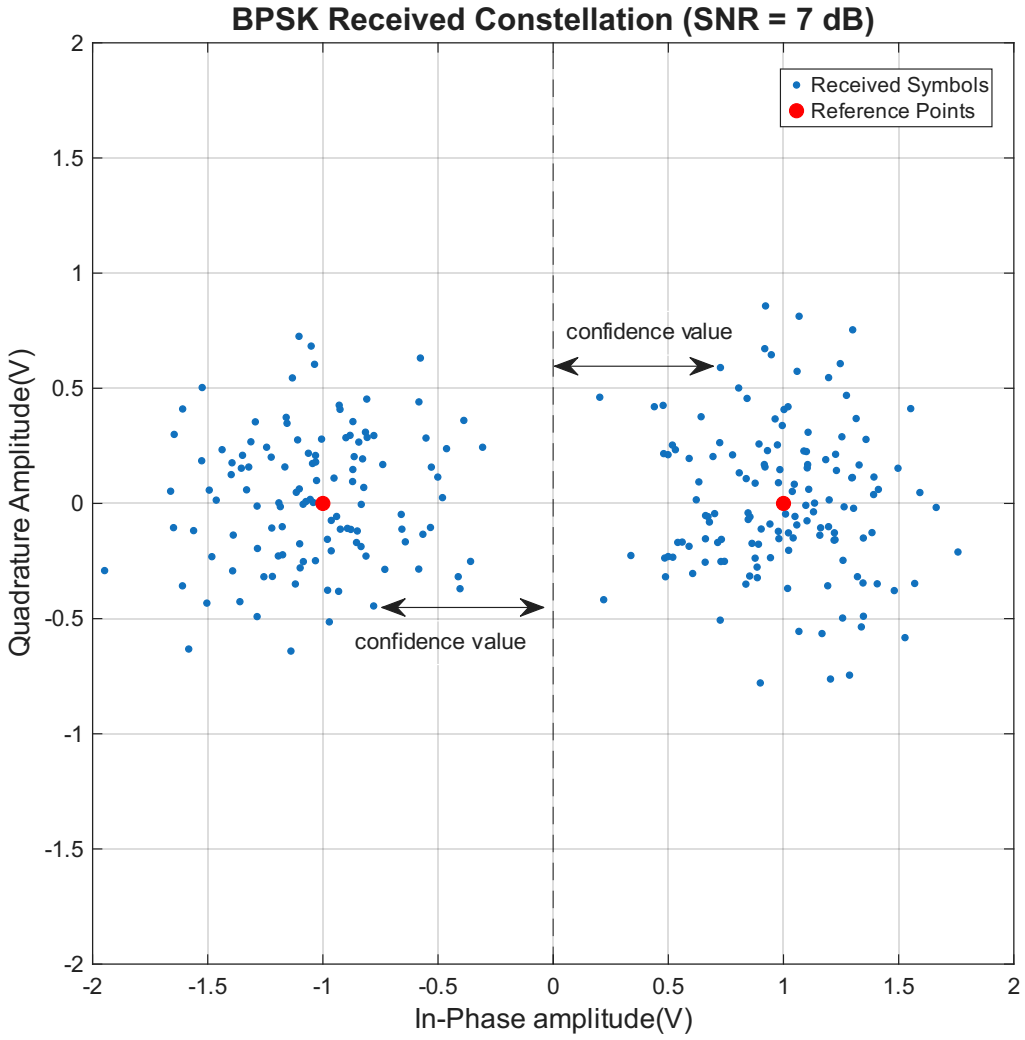


Figure 3.2: Illustration of calculating geometric distance of the modulated points for computing confidence values from a BPSK simulation.

The clipped confidence values were then converted into 8-bit unsigned integers ranging from 0 to 255, where a value of 255 represented 1, which corresponded to a symbol that was almost certainly correct (provided, we have sufficiently high SNR). These 8-bit integers served as the input vectors for the selection networks. This method of measuring confidence values based on the geometry of the constellation was scalable to higher-order modulation schemes, enabling the generation of input vectors for more complex modulation formats. The choice of an 8-bit width aligned with the data width of the final digital hardware, and the rationale for using 8 bits is further discussed in Chapter 4.

3.2 Generating HDL Scripts

The HDL netlists were generated using a MATLAB-based compiler script. This script primarily consist of file-generation routines that produc the required HDL files for synthesis. It accepts numerical parameters defining the architecture and automatically generates all possible network configurations for the specified input and output sizes. The use of such a compiler script was motivated by improved maintainability and the fact that the HDL definition remained fully reproducible for any set of input parameters.

The generated HDL code was structured without conditional statements, which simplified interpretation by synthesis tools and reduced synthesis time. Each generated network was stored in a separate folder within a specified parent directory, significantly improving the organization and manageability of synthesis and analysis tasks.

Verilog [23] was chosen as the hardware description language for all implementations.

3.3 The Digital Design Pipeline

The digital logic was targeted to be implemented on an Application-Specific Integrated Circuit (ASIC) using ASAP7 libraries [24], as justified in Chapter 4. The Cadence suite of tools was used to support this design pipeline. Cadence Xcelium was used for both functional and post-synthesis verification, while Cadence Genus was used to synthesize the HDL designs into gate-level netlists.

The digital design pipeline began with functional verification of the HDL and concluded with the extraction of gate statistics, area, and power metrics. The entire pipeline was automated using shell and Tool Command Language (TCL) scripts, which were generated by the same MATLAB-based compiler script described in Section 3.2. As a result, both the HDL and automation scripts were generated with minimal manual effort, significantly simplifying and accelerating the design workflow. Figure 3.3 illustrates the various stages of this pipeline. This process was repeated for every network that was generated and analyzed.

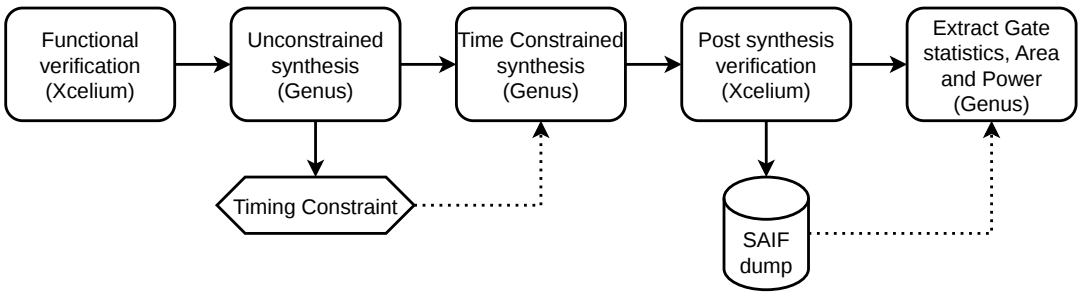


Figure 3.3: Illustration of the digital design pipeline implemented using the Cadence tool suite for design, analysis, and metric extraction.

Functional Verification: In this stage, the HDL design was verified using the test

vectors generated from MATLAB, as described in Section 3.1, to ensure correctness before synthesis.

Unconstrained Synthesis: This stage performed the initial conversion of the functional HDL description into a gate-level netlist. The synthesis was carried out with minimal optimization to preserve the original structure of the design, providing a reliable baseline for circuit delay. This was achieved by setting both the generic synthesis and mapping effort levels to "Low" in Genus.

Time-Constrained Synthesis: In this stage, the design was resynthesized under a timing constraint derived from the baseline delay obtained in the unconstrained synthesis stage. The generic synthesis and mapping efforts were set to "Medium" to allow the tool to explore simpler implementations of the logic, followed by a dedicated optimization step. The optimization effort was set to "High" to aggressively refine the design and meet the specified timing constraint.

Post-Synthesis Verification: Here, the generated gate-level netlist was verified using the same methodology as in the functional verification stage. In addition, the simulator was configured to generate a Switching Activity Interchange Format (SAIF) file, which captures the switching activity of the entire design and was later used for power estimation.

Extraction of Gate Statistics, Area, and Power: In the final stage, gate statistics, area, and power metrics were extracted using Genus. Power estimation required switching activity data, which was imported from the SAIF file generated in the previous stage. The SAIF format was preferred over the Value Change Dump (VCD) format because VCD files for large designs can grow to impractically large sizes. This was observed when attempting to generate a VCD file for one of the larger designs based on the proposed network, where the file size grew on the order of terabytes, resulting in a significant file input and output bottleneck in the pipeline.

To further accelerate the overall workflow, a top-level automation script was developed to spawn parallel jobs, enabling the digital design pipeline to be executed simultaneously for multiple networks. This significantly reduced the total time required to synthesize the circuits and extract the corresponding performance metrics.

4

Design

This chapter provides a description of the definition of a new metric for quantifying sloppiness, the proposed architecture, and the associated design decisions along with their underlying rationale. The chapter is organized into four main sections. The first covers the design space, the second introduces the sloppiness metric, the third describes the MSB-first Compare-and-Swap (MFCaS) unit, which serves as the primary building block of the architecture, and the fourth presents the architecture itself.

4.1 Design Space

Before detailing the proposed architecture, it is necessary to define the design space of the problem. This is guided by current trends in channel coding for telecommunications systems. The input and output sizes for the selection networks are defined as

$$N = \{32, 64, 128, 256, 512, 1024\} \tag{4.1}$$

$$k = \{2, 4, 8, 16\} \tag{4.2}$$

These choices are motivated by practical constraints in modern channel coding schemes, where the maximum block length is typically 1024 bits [25]. Consequently, a decoder may need to process up to 1024 reliability values simultaneously. Similarly, a decoder may need to look at up to 16 possible candidates for error correction [26].

The data type selected for representing reliability values is an unsigned integer with a fixed width of 8 bits. This choice is based on empirical observations (Refer Appendix A.1) from simulations conducted across multiple network configurations. In these experiments, the data width was varied from 2 to 12 bits, and the resulting outputs were analyzed in terms of their ability to identify the least reliable bits. It was observed that performance saturates beyond 6 bits, indicating diminishing returns for higher precision. Therefore, an 8-bit representation is selected as a conservative and efficient design choice.

The digital logic is chosen to be implemented using an ASIC flow rather than an Field Programmable Gate Array (FPGA). Although FPGA prototyping can accelerate the design process, it does not accurately reflect the area, power, and

timing characteristics of a full-scale implementation. Therefore, an ASIC flow is preferred to obtain a realistic assessment of the design's size and performance. The ASAP7 predictive Process Design Kit (Referred to as the ASAP7 PDK) [24] will be used for synthesizing ASIC-implementable logic. This is because ASAP7 PDK has a rich set of gate libraries.

4.2 Sloppiness of Selection Processes

To the best of our knowledge, there is no commonly used sloppiness metric that explicitly characterizes deviations from optimal behavior in selection processes. We therefore propose a metric, denoted as σ, to quantify this effect, which we refer to as sloppiness.

To define sloppiness, we first introduce three sets. The input set, denoted by I, contains all the candidate elements from which the selection is made. The expected selection set, denoted by E, contains the elements that should ideally be selected according to the selection criteria. The actual selection set, denoted by A, contains the elements returned by a given implementation of the selection process (e.g., a selection network). By definition, both E and A are subsets of I,

$$A \subseteq I, \quad E \subseteq I. \tag{4.3}$$

Sloppiness is defined as the proportion of elements in A that are not present in E:

$$\sigma = \frac{|A \setminus E|}{|A|} \tag{4.4}$$

From this definition, it is obvious that sloppiness is a real number between 0 and 1:

$$\sigma \in [0, 1] = \{x \in \mathbb{R} \mid 0 \leq x \leq 1\} \tag{4.5}$$

To illustrate this concept, consider the following example. Let the input set be

$$I = \{12, 3, 7, 19, 5, 14, 8, 21, 1, 17, 6, 10, 2, 15, 9, 11\} \tag{4.6}$$

and suppose that the selection criterion is to choose the four largest numbers from I. The expected selection set is then,

$$E = \{15, 17, 19, 21\} \tag{4.7}$$

Now, assume a given implementation returns the actual selection set:

$$A = \{14, 19, 17, 6\} \tag{4.8}$$

The elements in A that are not in E are:

$$A \setminus E = \{14, 6\} \tag{4.9}$$

Using the sloppiness formula (4.4), we compute:

$$\sigma = \frac{|A \setminus E|}{|A|} = \frac{2}{4} = 0.5 \tag{4.10}$$

This value indicates that half of the selected elements deviate from the expected selection (σ does not have any units).

In practice, σ provides a quantitative measure of how "sloppy" a selection process is. The selection networks described in Section 2.2 aim to choose the k largest or smallest elements from a given set. These networks perform perfect selection, and therefore, their sloppiness is 0. However, this metric allows us to evaluate future or alternative selection implementations that may not be perfect.

4.3 MSB-first Compare and Swap (MFCaS)

CaS and CaSe units form the fundamental building blocks for sorting, selection, and minimum finder networks, as discussed in Chapter 2. However, the comparator logic within these units is typically more complex than the multiplexer logic [27, 28], and can therefore become a limiting factor in terms of area and power efficiency.

When a unsigned integer comparator is synthesized from RTL, synthesis tools commonly implement it as a subtractor, where the two inputs are subtracted, and the MSB of the result is used to determine their relative magnitude. This information from the MSB then drives the control signals of the multiplexers. Consequently, the comparator often dominates the overall cost of CaS and CaSe units.

To address this limitation, the MSB-first Compare and Swap unit is introduced [29]. This consists of identical single-bit cells connected in a chain, analogous to a Ripple Carry Adder (RCA). Each cell receives one bit from each of the two input words, along with two state bits propagated from the preceding, more significant cell. Based on these inputs, the cell produces four outputs, two data bits (corresponding to the higher and lower values, respectively) and two state bits that are passed to the next, less significant cell.

Data propagates from MSB to LSB, allowing the unit to progressively determine and propagate the relative ordering of the input words. This structure enables efficient comparison and conditional swapping of multi-bit words using simple, uniform logic cells, resulting in reduced area and improved average comparison speed compared to a traditional comparator.

In a similar manner, a MSB First Compare-and-Select (MFCaSe) unit can be derived from the MFCaS by trimming it, analogous to the approach described in Section 2.3. From this point onward, unless otherwise specified, the term CaS refers to MFCaS, and CaSe refers to MFCaSe.

4.4 Proposed Sloppy Selection Network Architecture

In Chapter 2, accurate selection networks were discussed in detail. However, in decoder applications, perfect selection is not strictly necessary. This is because the soft information used to identify unreliable bits is inherently noisy. Therefore, we hypothesize that a sloppy selection network can still achieve comparable performance, on average, to an accurate selection network in identifying unreliable bits.

To construct such a network, we still rely on accurate selection structures, but only for smaller input sizes. The proposed sloppy selection architecture is based on a binary tree minimum finder network, described in Section 2.3, which is modified to approximately select the smallest k out of N inputs.

We define a sloppy binary-tree-based selection network that selects the k smallest values from N inputs as SB-bottom(k, N), and its counterpart for selecting the largest values as SB-top(k, N). The SB-bottom(k, N) network is constructed by slicing a binary tree minimum finder at an intermediate level determined by k and N. We refer to this structure as a Sliced Binary Tree (SBT) network.

Figure 4.1 illustrates an example of an SB-bottom$(4, 16)$ network obtained by slicing a 16-input minimum finder tree. Similarly, the SB-top(k, N) network can be realized by slicing a binary tree maximum finder.

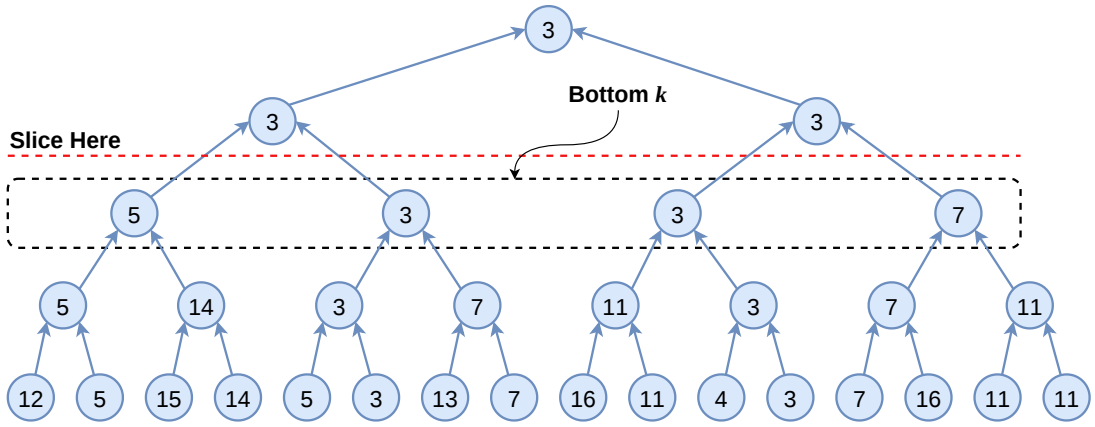


Figure 4.1: Illustration of a SB-bottom$(4, 16)$ SBT network constructed by slicing a 16-input binary tree minimum finder. The network sloppily selects 4 smallest values from 16 inputs. Each pair of arrows leading to a node represents an MFCaSe unit.

The motivation for this design arises from observing intermediate levels of the binary tree. At the level where the number of nodes equals k, the probability of correctly capturing majority of the k smallest or biggest elements remain high. By using a binary tree based approach, we effectively discard half of the candidates at each level, significantly reducing the number of comparisons. The total number of CaS and CaSe units required to construct such SBT network (hereafter referred to as the network size) can be obtained as follows(See Appendix A.2),

$$S_{SBT} = N - k \tag{4.11}$$

Furthermore, the SBT network only uses CaSe units, leading to additional hardware savings. For example, as seen by comparing Figures 4.1 and 2.5, an accurate bitonic selection network requires 60 CaS units, whereas the proposed SBT network requires only 12 CaSe units to perform a similar task, albeit sloppily.

To build further intuition, consider the input set,

$$I = \{12, 5, 15, 14, 5, 3, 13, 7, 16, 11, 4, 3, 7, 16, 11, 11\} \tag{4.12}$$

The accurate bitonic selection network (Section 2.2) produces,

$$A_{\text{Bitonic}} = \{5, 5, 3, 3\} \tag{4.13}$$

In contrast, the SBT-based sloppy selection network produces,

$$A_{\text{SBT}} = \{5, 3, 3, 7\} \tag{4.14}$$

The corresponding sloppiness metrics are,

$$\sigma_{\text{Bitonic}} = \frac{0}{4} = 0 \tag{4.15}$$

$$\sigma_{\text{SBT}} = \frac{1}{4} = 0.25 \tag{4.16}$$

This example demonstrates that approximately 80% of the hardware can be eliminated at the cost of a 25% increase in sloppiness for this input instance. Additionally, the circuit depth is reduced by roughly a factor of four, resulting in significantly faster operation or timing slack that may be used for power savings. It should be noted that sloppiness depends on the input distribution; therefore, average-case analysis is necessary for a comprehensive evaluation.

Despite these advantages, SBT networks exhibit poor performance in the presence of duplicate values. This limitation arises because the network typically discards duplicate entries, resulting in outputs that primarily consist of distinct values. In telecommunications applications, inputs are often modeled using a Gaussian distribution, which increases the likelihood of repeated values. This effect is further amplified in our case, since the data width is fixed at 8 bits, limiting the number of distinct values to 256. As the input size N increases, the probability of duplicates grows significantly due to this limited value range, becoming particularly pronounced when $N > 256$.

To mitigate this issue, we propose a hybrid architecture to perform sloppy selection. The input is partitioned into multiple segments, each processed by an SBT network. The outputs of these SBT blocks are then fed into a smaller accurate selection network. Depending on the input size, this final stage is implemented using either a Halflife selection network [8] or an odd-even selection network [8]. Specifically, a

Halflife network is used for input sizes greater than or equal to 16, while an odd-even network is used for smaller inputs. This is because the Halflife algorithm did not support input sizes below 16 and the next best algorithm was odd-even selection.

We define this hybrid selection architecture for selecting the k smallest values from N inputs as S-bottom(k, N), and its counterpart for selecting the largest values as S-top(k, N).

Figure 4.2 illustrates the proposed architecture, which combines SBT networks with an accurate selection stage. The N inputs are divided into Q equal partitions, each processed by an SB-bottom$(k, \frac{N}{Q})$ (or SB-top$(k, \frac{N}{Q})$) network. The outputs of all Q SBT blocks—each producing k candidates—are then combined and processed by a final accurate selection network of size bottom(k, Qk) (or top(k, Qk)).

The architecture enforces consistency in the selection type throughout. For an S-bottom(k, N) network, all SBT blocks are SB-bottom$(k, \frac{N}{Q})$, and the final stage is a bottom(k, Qk) network. Similarly, for an S-top(k, N) network, all components operate in top-selection mode. Mixing top and bottom selection within the same architecture is not allowed.

Since $Qk \ll N$ for small k, the accurate selection stage operates on a significantly reduced input size, leading to substantial savings in hardware compared to a fully accurate selection network.

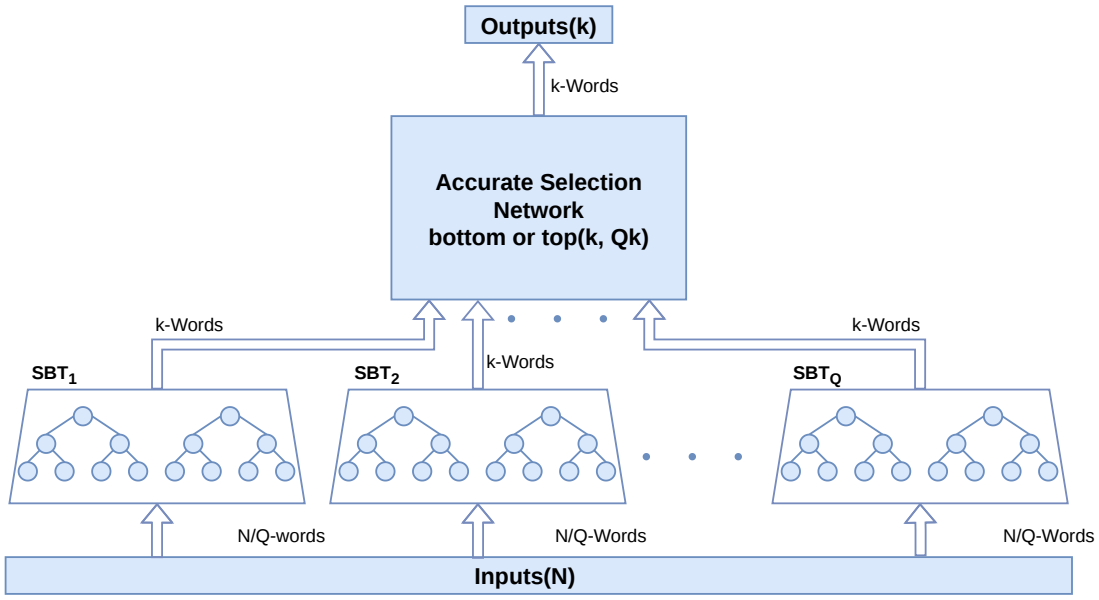


Figure 4.2: Illustration of the proposed hybrid architecture for sloppy selection of the k largest or smallest values from N inputs.

The size of the proposed network is dominated by the choice of accurate selection network.

When implemented using a Halflife accurate selection network, the size is given by

$$S_{\text{proposed}} = (N - Qk) + \left(\frac{N}{k} - 1\right)(2S_k + k)\,, \tag{4.17}$$

where S_k denotes the size of a k-input half life sorting network.

Alternatively, when an odd-even accurate selection network is used, the size becomes

$$S_{\text{proposed}} = (N - Qk) + N\left(\frac{\log^2(k) + 3\log(k) + 4}{4}\right) - k\log(k) - 1. \tag{4.18}$$

An important advantage of this architecture is that the level of sloppiness can be tuned by adjusting the number of partitions Q. Increasing Q results in smaller SBT blocks and a larger accurate selection stage, thereby reducing sloppiness at the cost of increased hardware. Conversely, reducing Q increases sloppiness while improving hardware efficiency. This tunability allows the design to be adapted to different application requirements.

It should be noted that this architecture assumes N and k are powers of two, with $N \gg k$, which aligns with the target design space.

5

Results

This chapter presents the performance evaluation of the proposed architecture for a sloppy selection network. The design performs sloppy selection, and its behavior is analysed from multiple perspectives. The chapter begins with a system-level analysis of the proposed networks, followed by an examination of how the resulting architecture translates into digital hardware requirements.

Unless otherwise specified, references to a sloppy selection network apply to both "S-bottom" and "S-top" networks. Additionally, the terms "input size" and "number of inputs" are used interchangeably throughout this chapter, as are "output size" and "number of outputs". All graphs presented in this chapter use logarithmic axes unless explicitly stated otherwise.

5.1 Performance of the Proposed Networks

To evaluate the performance of the proposed sloppy selection network architecture, the network is modeled as a system in MATLAB and compared against conventional Accurate Selection Networks (ASN) used in current decoder implementations. First, the probability of correctly identifying faulty bits (pass probability) is computed for both the accurate selection network and the proposed sloppy selection network.

Next, the difference between the pass probabilities of the accurate and sloppy selection networks is calculated and visualized using a 3D mesh plot. Section 3.1 discusses the evaluation methodology in detail. Figure 5.1 illustrates the resulting mesh plot, where each surface corresponds to a specific number of SBTs in the network. The plot demonstrates how closely the sloppy selection network matches the performance of the accurate selection network. Smaller differences indicate more similar behavior between the two networks and therefore represent better performance.

Figure 5.1 shows that, when moving from larger input sizes toward smaller ones, the difference in pass probability initially increases with increasing output size for a fixed number of inputs. This indicates that the sloppy selection network deviates further from the accurate selection network as the number of outputs increases. However, this trend gradually changes as the number of inputs decreases, causing the peak of each surface to shift toward smaller output counts.

This shift in the maxima can be observed more clearly by flattening one of the surfaces into a 2D plot. Figure 5.2 illustrates one such flattened surface that represents networks with 2 SBTs. All such plots are available in appendix A.3. The

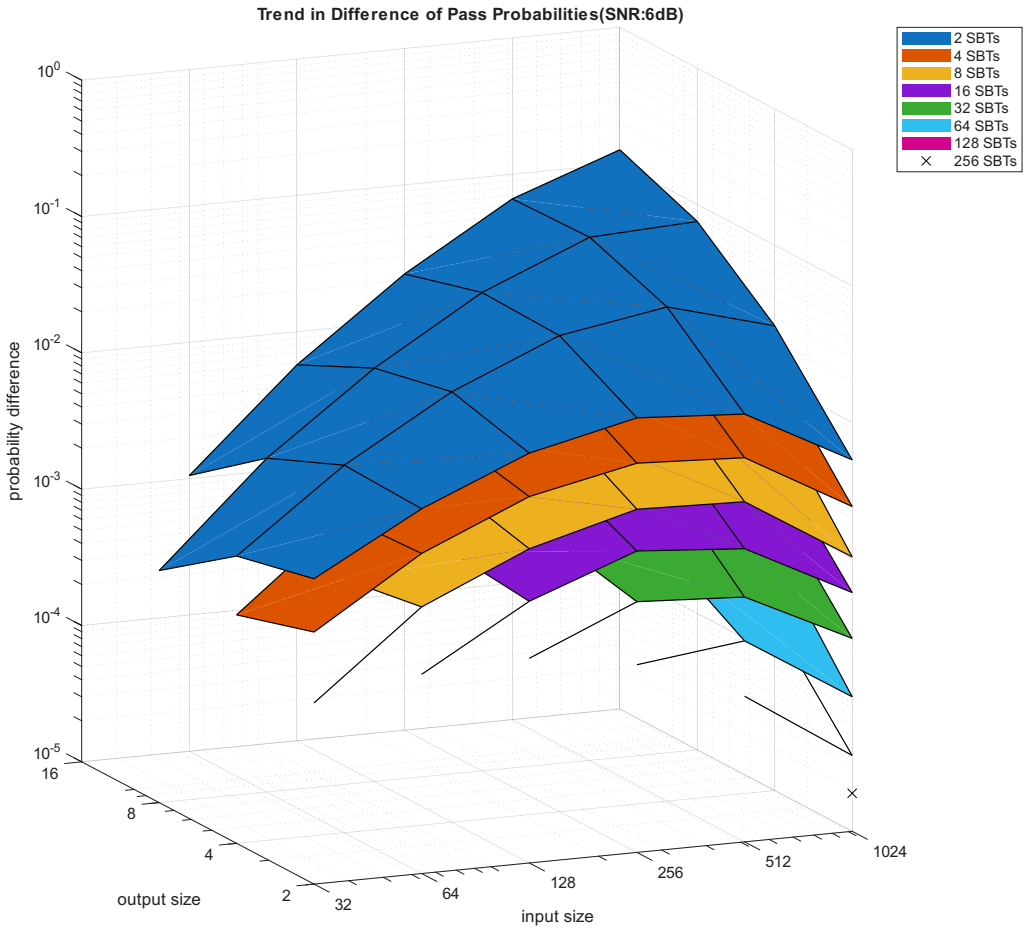


Figure 5.1: The difference in pass probabilities between accurate and sloppy selection counterparts as function of input and output sizes(lower values are better). Each surface represents a different number of SBTs. This trend is plotted at an SNR of 6 dB

figure shows that the peak of the curve shifts toward lower output counts as the number of inputs decreases. This behavior can be explained by the architecture of the sloppy selection network itself. As the number of outputs increases, the number of inputs to the accurate selection network within the sloppy architecture also increases. Consequently, the sloppiness initially decreases because the accurate selection stage has a larger pool of candidates to select from. However, for sufficiently large input sizes, the SBTs become progressively less efficient at selecting the k smallest values from the N/Q inputs. Therefore, for larger input sizes, the overall sloppiness of the selection network becomes dominated by the behavior of the binary trees.

Further observations from Figure 5.1 reveal that lower surfaces exhibit greater curvature. This behavior can be attributed to the increasing number of SBTs associated with these surfaces, which amplifies the sloppiness introduced by the SBT stages. Additionally, apart from the curvature, the spacing between adjacent surfaces remains approximately uniform. This behavior can be visualized more clearly by examining 2D slices of the surfaces along either the output-size or input-size axes.

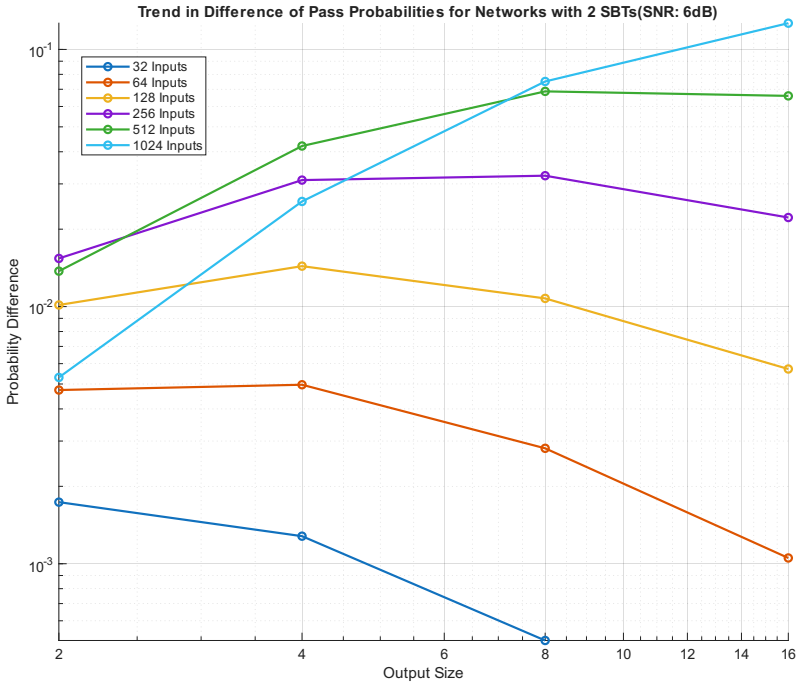


Figure 5.2: The difference in pass probabilities between accurate and sloppy selection counterparts as functions of the output sizes for a given number of SBTs in the network (lower values are better). Each line represents a different number of inputs. This trend is plotted at an SNR of 6 dB

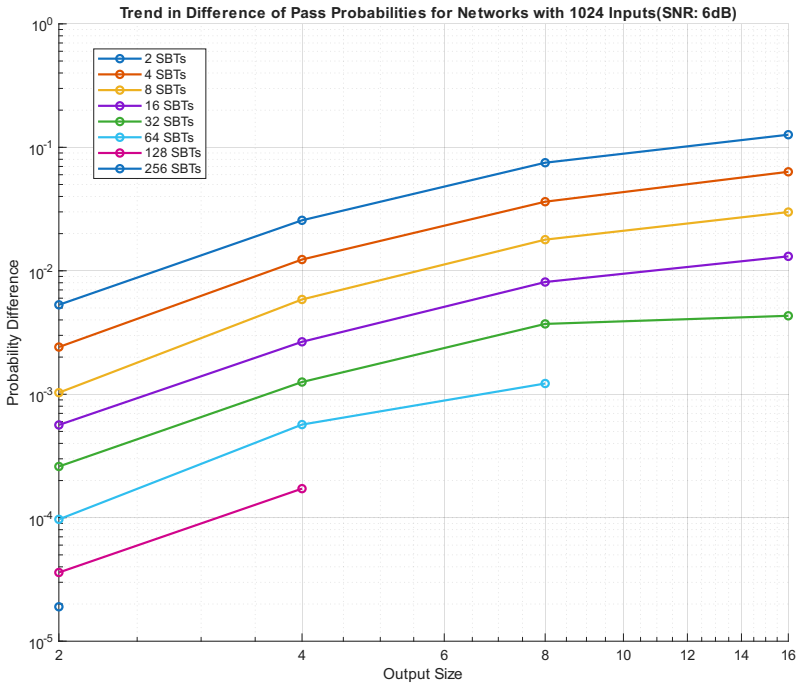


Figure 5.3: The difference in pass probabilities between accurate and sloppy selection counterparts as a function of the output sizes for 1024 input words in the network (lower values are better). Each line represents a different number of SBTs. This trend is plotted at an SNR of 6 dB

Figure 5.3 illustrates one such slice along the output-size axis for 1024 inputs. Similar slices for other input sizes are provided in Appendix A.4. The nearly uniform spacing between the curves suggests a logarithmic improvement in the performance of the sloppy selection network as the number of SBTs increases. This behavior can be explained by the scaling properties of the accurate selection network within the proposed architecture. Each curve corresponds to a number of SBTs that scale as a power of two. Consequently, as the number of SBTs increases logarithmically, the input size of the accurate selection network within the proposed architecture also increases logarithmically. This provides a progressively larger candidate pool for the final selection stage, thereby increasing the likelihood of selecting the correct candidates.

Further, Figure 5.1 is compared against the variation of average sloppiness σ for different combinations of input and output sizes. Figure 5.4 illustrates this comparison. It can be observed that the surfaces in the average sloppiness plot are nearly flat. This suggests that the sloppiness metric σ only provides a coarse estimate of the behavior of the sloppy selection network. Consequently, additional refinements to the metric are required for it to more accurately predict the selection performance of sloppy selection networks.

Additionally, when closely observed it can be seen that the levels of the sloppiness surfaces approximately align with the peaks of the surfaces in the "probability difference" trend. This relationship can be visualized more clearly by comparing the peak values of the curved "probability difference" surfaces with the average value of each corresponding "sloppiness" surface, where the average value roughly represents the overall level of the surface.

Table 5.1 presents these values side by side for comparison. From the table, it can be seen that the sloppiness metric σ initially slightly overestimates the peak probability differences. However, this estimation error progressively increases as the number of SBTs grows.

Further, it needs to be noted that while all this is true, it is also true that in a decoder, even if we do accurate selection there is a certain probability that the actually erroneous bit can be missed. This probability decreases with the number of selected elements k.

Table 5.1: Comparison between the peak values of the "probability difference" surfaces and the corresponding "sloppiness" surface levels for different numbers of SBTs

No. of SBTs	Peak Probability Difference	Sloppiness Surface Level
2	0.1265	0.1368
4	0.0633	0.0653
8	0.0299	0.0302
16	0.0131	0.0140
32	0.0043	0.0062
64	0.0012	0.0027
128	0.0002	0.0011
256	0.00002	0.0004

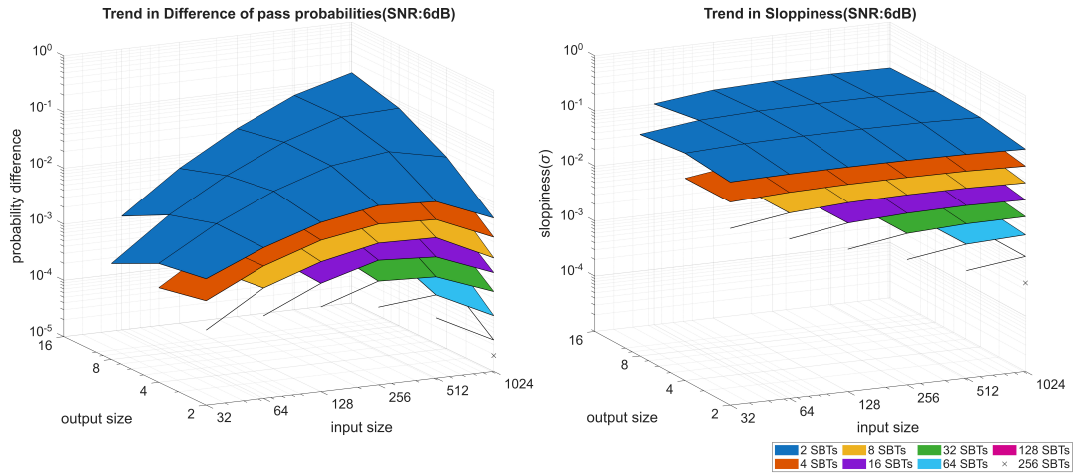


Figure 5.4: Comparison between the trends in pass probability difference (left) and average sloppiness (right). These trends are plotted at an SNR of 6 dB.

5.2 Hardware Analysis

Having analyzed the performance characteristics of the proposed sloppy selection networks, the hardware implications of the architecture are now investigated. This section first examines the circuit area of the proposed networks, followed by an analysis of the achieved reduction in circuit size compared to conventional bitonic sorting networks commonly used in decoder implementations. Next, the circuit delay characteristics are analyzed along with the corresponding reduction in network depth. This is followed by a power analysis, including evaluations of the energy-delay product (EDP) and energy per selection (EPS). Finally, the improvements obtained using the proposed compare-and-swap architecture are discussed.

All networks were synthesized under timing constraints using Cadence Genus. Since multiple implementations of the same network may be generated depending on the applied timing constraint, a final timing constraint equal to 70% of the delay obtained from unconstrained synthesis was used for all networks. Consequently, the synthesized networks are not necessarily optimized for minimum delay, but instead represent practical high-performance implementations that may realistically be adopted in hardware designs.

Furthermore, all results discussed in this section are based solely on post-synthesis analysis and do not include place-and-route effects. This is because final area and timing characteristics after layout depend on several additional factors, including the size and shape of the placement region, the number of available routing layers, and clock tree distribution. In addition, no architectural optimizations such as pipelining were applied. This is because such optimizations are highly application-dependent and can significantly alter area, delay, and energy characteristics, making direct comparisons less consistent across different design points.

5.2.1 Circuit Area

Circuit area is evaluated using the "total area" parameter reported by Cadence Genus after logic synthesis. This metric includes both active cell area and an estimation of interconnect area. However, the area trend alone is difficult to interpret analytically. To facilitate analysis, a secondary metric termed "network cost" is introduced. Network cost is defined as the number of CaS or CaSe units required to construct the network, since these form the fundamental building blocks of the proposed architecture. For simplicity, both CaS and CaSe units are assigned equal weight, although a CaSe unit is slightly smaller than a CaS unit. This is because, the network cost is intended only to provide a rough estimate of the network area.

Figure 5.5 compares the trend in network cost with the synthesized circuit area for different combinations of input and output sizes. Each surface corresponds to a different number of SBTs, as discussed previously.

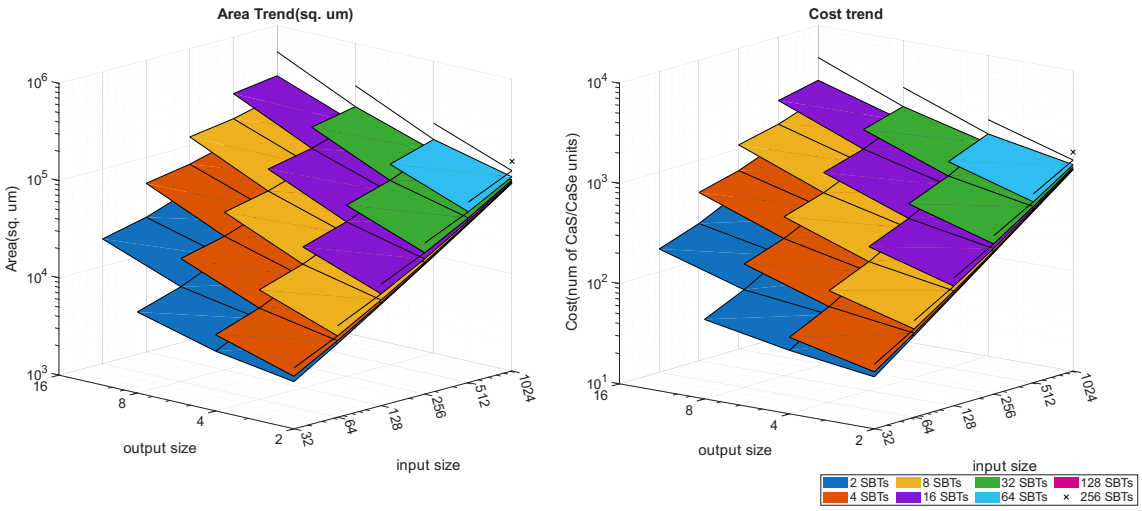


Figure 5.5: Comparison between the trends in area of the synthesized networks (left) and estimated network cost (right). The area trend is plotted for networks synthesized with a timing constraint equal to 70% of the unconstrained delay.

From Figure 5.5, it can be observed that the estimated network cost surfaces closely follow the trends obtained from the synthesized circuit area. This agreement indicates that network cost serves as a useful analytical approximation for estimating circuit area. In contrast, [8] employed a similar estimation methodology but reported significant discrepancies between the estimated and synthesized area results. A detailed discussion of the possible causes of these discrepancies is provided in Section 6.4.

For further analysis, the surfaces are examined using slices taken along fixed output sizes. Such slices provide clearer insight into the scaling behavior of the network compared to observing the full three-dimensional surfaces directly. In addition, the network cost metric can be decomposed into contributions from different stages of the architecture, whereas performing a similar decomposition using synthesized area data is considerably more difficult.

Figure 5.6 shows a slice of the network cost surface for networks with an output size of 8. Corresponding plots for other output sizes are provided in Appendix A.6. A

linear scale is used in this figure to improve visualization and interpretation of the individual trends.

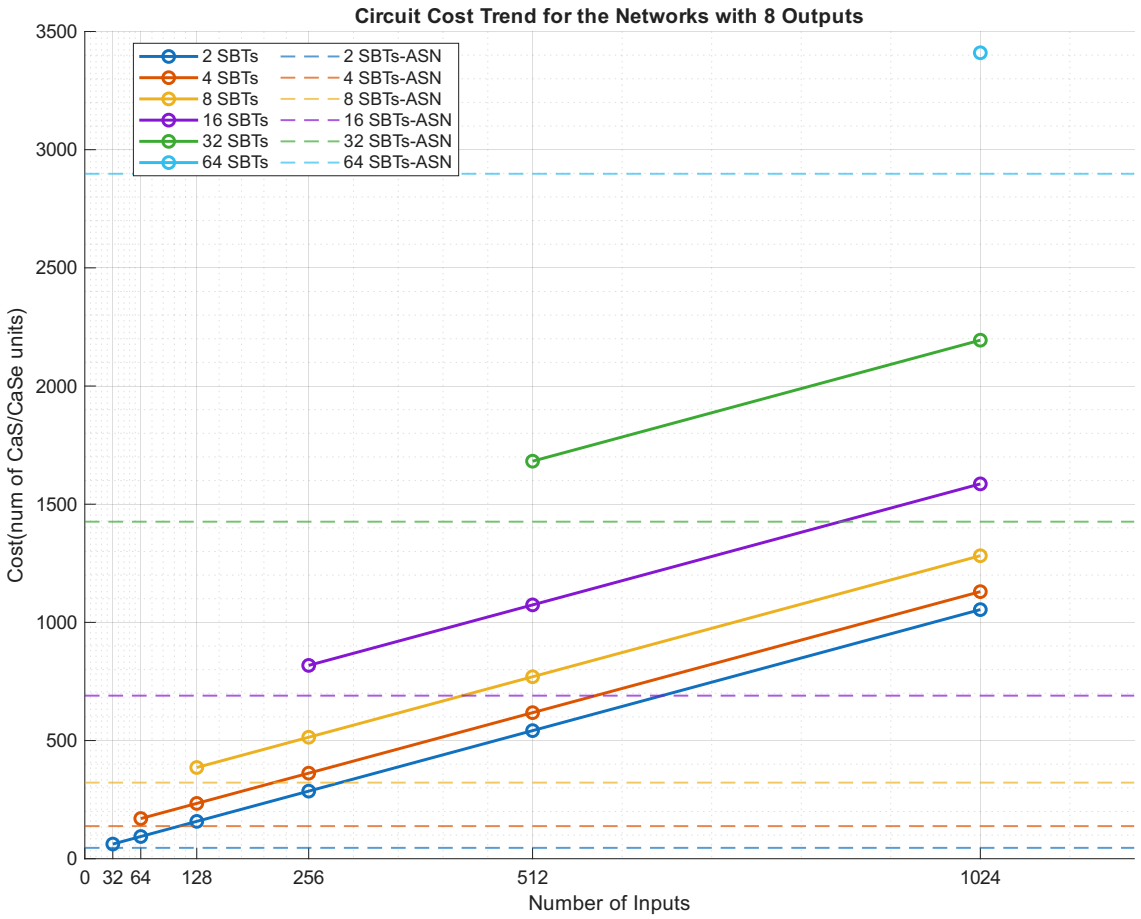


Figure 5.6: Circuit cost trend for all synthesized networks with 8 outputs. The dotted lines represent the cost of the ASN, while the solid lines represent the total network cost.

From Figure 5.6, it can be observed that the size of the ASN depends only on the number of SBTs present in the architecture. This behavior is expected because, for a fixed output size, the input and output dimensions of the ASN are fully determined by the architecture. Specifically, the number of inputs to the ASN depends only on the output size and the number of SBTs. Consequently, the variation in the total network cost with increasing input size is primarily determined by the growth of the SBT structures.

The contribution of the SBT stages to the overall network cost was obtained by subtracting the cost of the ASN from the total network cost. Figure 5.7 compares these extracted values against the analytical estimate obtained from the first term of Equation 4.11, namely $N - Qk$.

The markers in Figure 5.7 represent the SBT cost extracted from the synthesized network data, while the dotted lines correspond to the analytical estimate. Although the analytical expression is linear, the plotted curves appear nonlinear because the

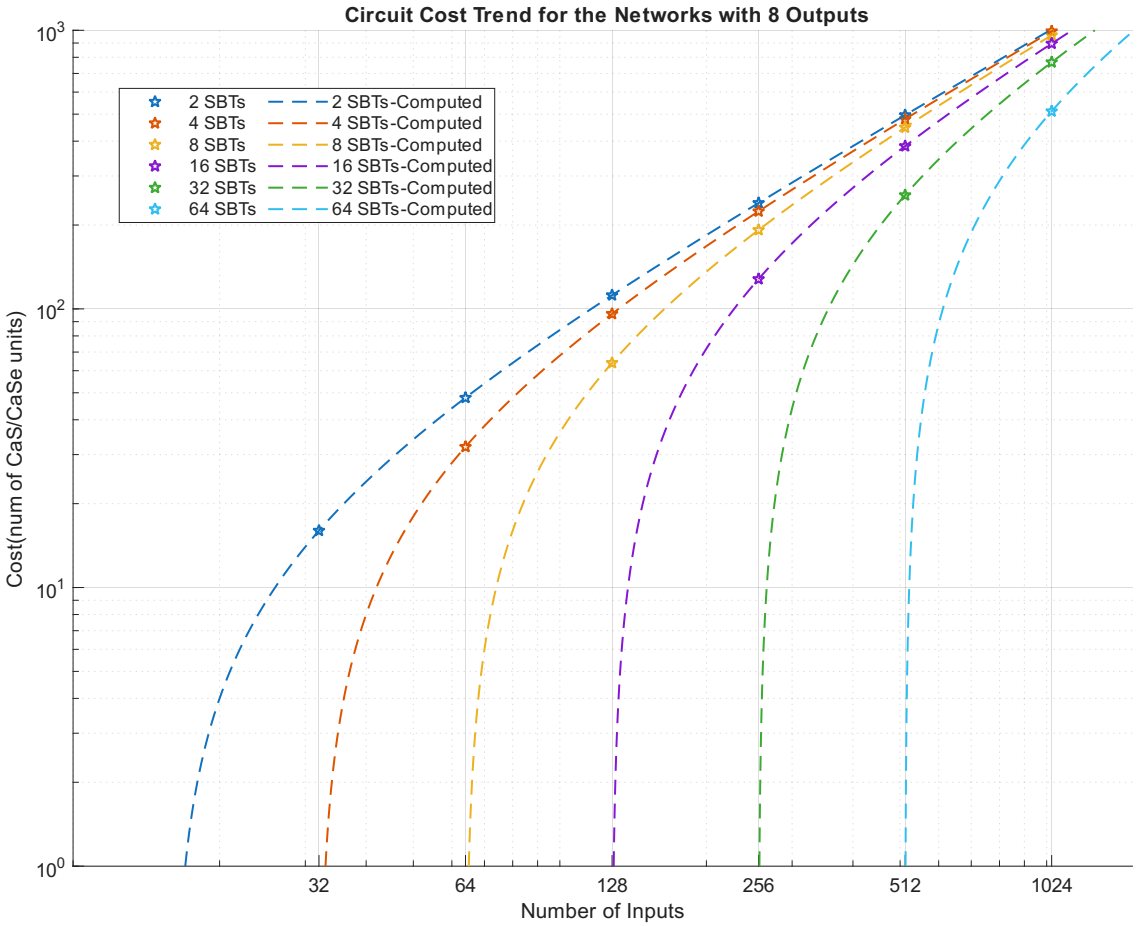


Figure 5.7: Cost trend for the SBT stages of all synthesized networks with 8 outputs. The dotted lines are computed using the expression $N - Qk$, while the markers represent the SBT cost extracted from Figure 5.6.

data is represented on logarithmic axes. Furthermore, the apparent convergence of the curves at larger input sizes is a consequence of logarithmic compression, where evenly spaced linear trends appear progressively closer together at higher values.

5.2.2 Reduction in Circuit Area

The circuit area reduction of the sloppy selection networks is estimated rather than directly calculated, since synthesizing all corresponding bitonic counterparts and estimating their exact area is highly time- and resource-intensive. The size (or cost) of the networks is therefore estimated in terms of the number of CaS or CaSe units. This metric was selected because it closely matches the trend observed in the actual synthesized area of the networks, as discussed in Section 5.2.1.

Figure 5.8 illustrates the reduction in network size achieved by the proposed sloppy selection architecture compared to its bitonic sorting counterparts. From Figure 5.8, it can be observed that the proposed sloppy selection networks are typically at most one quarter the size of their corresponding bitonic sorting networks. The largest reduction observed is approximately 96%, achieved for S-bottom(2,1024) or

S-top(2,1024) networks when compared against a 1024-input bitonic sorting network. This represents a substantial reduction in circuit size.

To better understand the origin of this reduction, the 3D surfaces shown in Figure 5.8 are sliced along the input-size axis and compared with the corresponding network-cost curves. Figure 5.9 illustrates one such slice for the case of 8 outputs with the corresponding network-cost curves. Similar slices (without the network-cost curves) for other output sizes are provided in Appendix A.7.

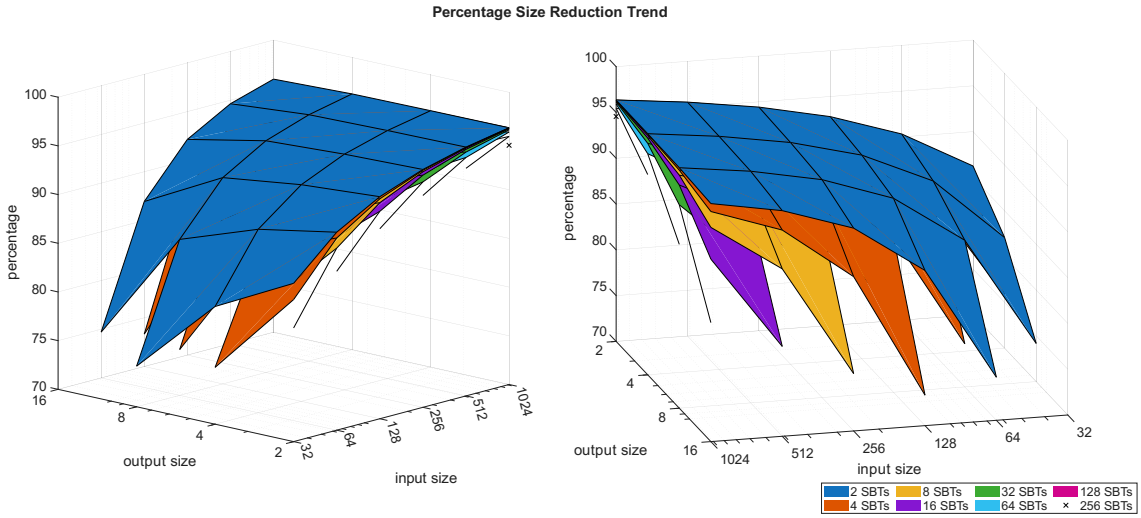


Figure 5.8: Size reduction in terms of percentage, compared to the equivalent bitonic sorting network for different combinations of input and output sizes.

From Figure 5.9, it can be observed that the overall profile of the size-reduction curves is consistent with the changing gap between the cost of the sloppy selection network and the cost of its equivalent bitonic sorting network.

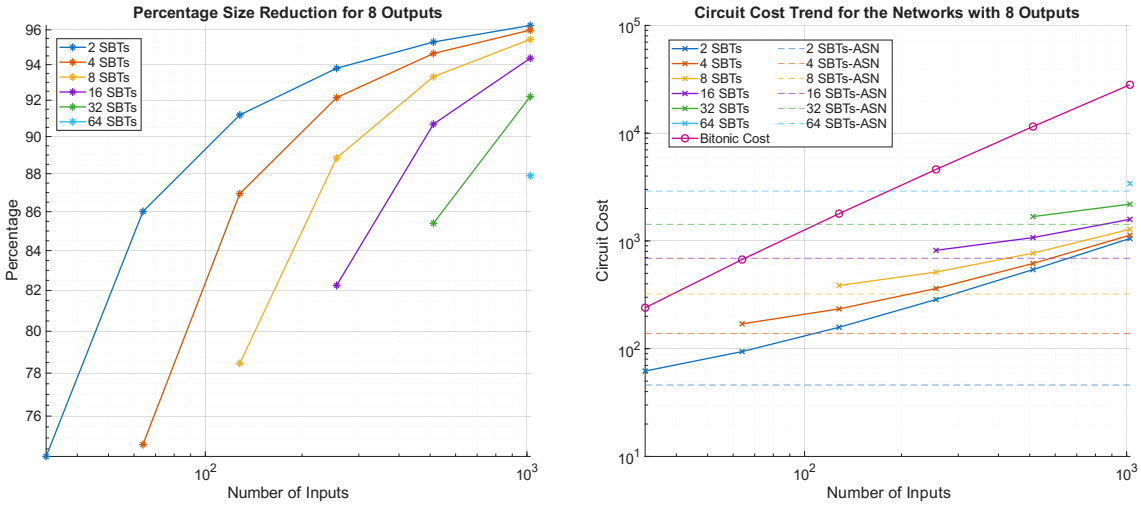


Figure 5.9: Size reduction curves for all 8 output sloppy selection networks and the corresponding cost split up. The dotted lines on the cost split up plot on the right represents the size of the accurate selection network stage present in the proposed networks, while the solid lines represent the total cost of the network

5.2.3 Circuit Delay

The delay characteristics of the synthesized networks are now evaluated. The plotted delays correspond to networks synthesized using a timing constraint equal to 70% of the unconstrained delay. Each surface in the analysis represents a different number of SBTs.

Although the delay surfaces provide an overview of the delay behavior, extracting detailed insights directly from them is difficult. Therefore, the logic depth of the networks was also analyzed, since circuit delay is strongly correlated with the number of logic levels present in the design. Figure 5.10 compares the estimated logic depth with the delays obtained from synthesis for different combinations of input and output sizes.

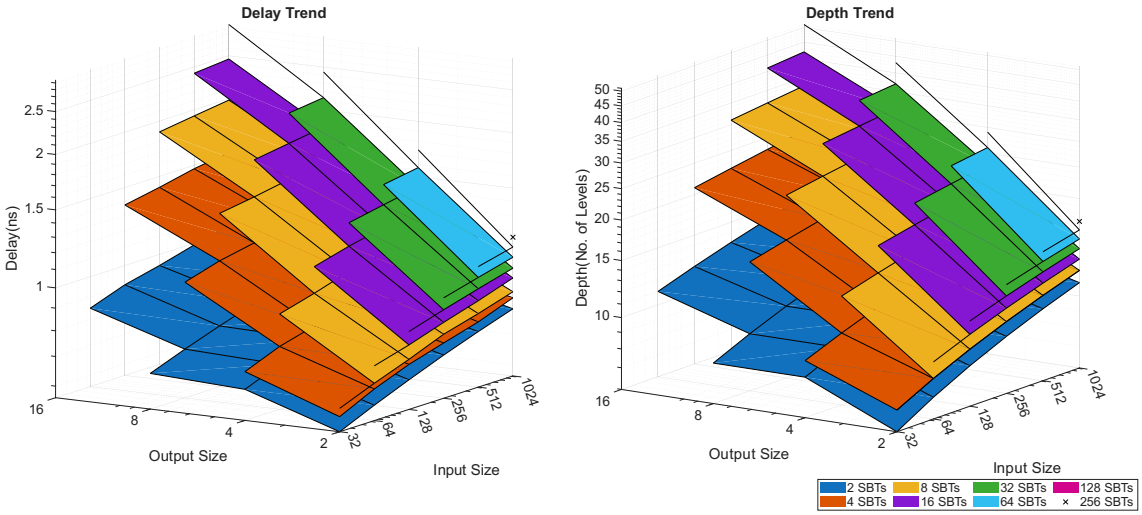


Figure 5.10: Comparison between the trends in delay of the synthesized networks (left) and the estimated logic depth of the synthesized networks (right). The trends are plotted for networks synthesized with a timing constraint equal to 70% of the unconstrained delay.

From Figure 5.10, it can be observed that the estimated logic depth closely follows the synthesized delay behavior. This indicates that decomposing the logic depth can indirectly reveal the dominant contributors to the overall circuit delay. In contrast, [8] employed a similar estimation methodology but reported significant discrepancies between the estimated and synthesized delay results. A detailed discussion of the possible causes of these discrepancies is provided in Section 6.4.

For clearer interpretation, the three-dimensional depth surfaces are reduced to sets of two-dimensional curves. Figure 5.11 illustrates one such projection performed along the input-size axis for networks containing 8 SBTs. The corresponding depth decomposition is also shown alongside the projected surface.

From Figure 5.11, it can be observed that the depth contribution of the SBTs decreases as the number of outputs increases. This occurs because the binary tree is truncated earlier as the number of outputs increases, resulting in a shallower SBT structure and a larger number of intermediate outputs. Conversely, the depth

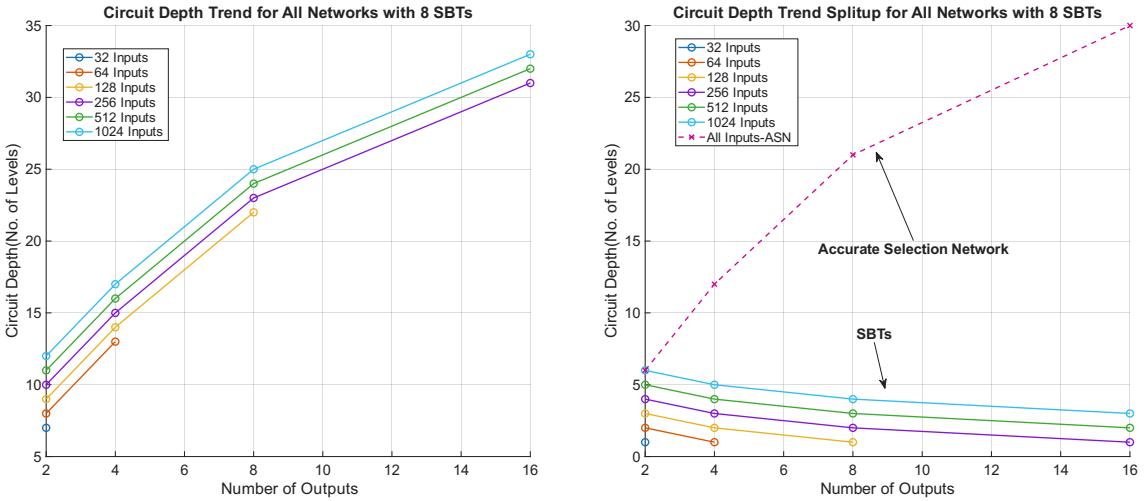


Figure 5.11: Circuit depth surface projected along the input-size axis for all synthesized networks with 8 SBTs (left), along with the corresponding depth decomposition (right).

contribution of the ASN increases with output size because the ASN must process and rearrange a larger number of candidate values, requiring additional logic levels.

Networks containing 8 or more SBTs generally follow this behavior. However, networks with fewer than 8 SBTs exhibit slightly different characteristics, where the depth curves show a dip or saturation region before eventually following the same growth behavior. This effect results from the use of a mixture of odd-even and Halflife ASNs, as described in Chapter 4.

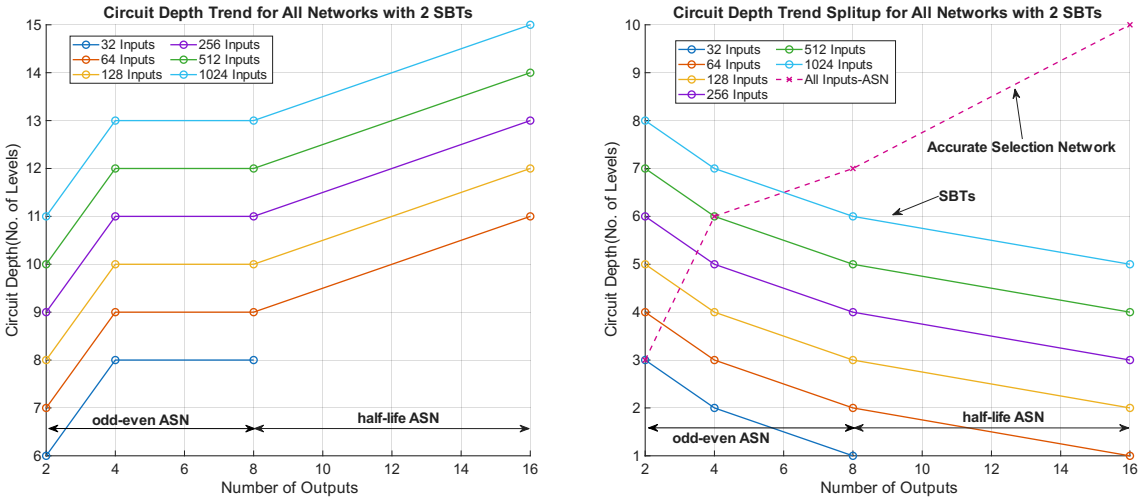


Figure 5.12: Circuit depth surface projected along the input-size axis for all synthesized networks with 2 SBTs (left), along with the corresponding depth decomposition (right).

Figure 5.12 illustrates this behavior for networks containing 2 SBTs. Furthermore, from Figures 5.11 and 5.12, it can be observed that the ASN forms the dominant

contribution to the overall circuit delay, whereas the delay contribution of the SBT stages is comparatively smaller.

5.2.4 Reduction in Logic Depth

The logic depth reduction of the proposed sloppy selection networks is estimated rather than directly calculated similar to how the reduction in circuit sizes were estimated, since synthesizing all corresponding bitonic counterparts and measuring their exact delays would be highly time and resource intensive.

Figure 5.13 illustrates the reduction in logic depth achieved by the proposed sloppy selection architecture compared to its bitonic sorting counterparts. Further it can be observed that while the lowest depth reduction of approximately 7% was achieved for a S-top(16,1024) or S-bottom(16,1024) network with 32 SBTs which is the largest network synthesized based on the proposed sloppy selection architecture, a maximum of 80% reduction was observed for a S-top(2,1024) or S-bottom(2,1024) network with 2 SBTs when compared against a 1024-input bitonic sorting network. Further a kink can be observed on the surfaces representing networks that have less than 8 SBTs. This kink is due to the usage of both odd-even and Halflife ASNs while constructing these sloppy selection networks.

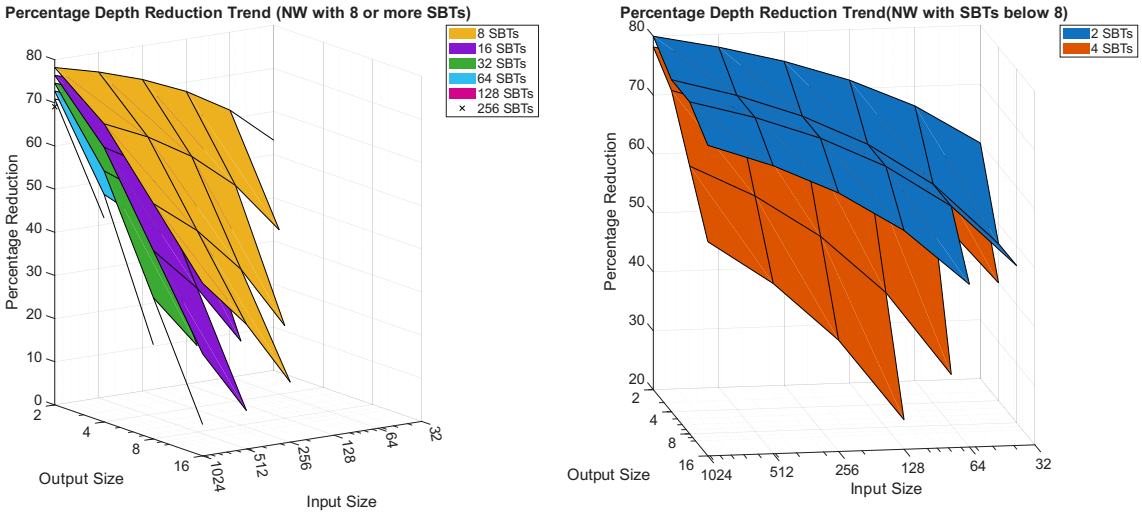


Figure 5.13: Logic depth reduction in terms of percentage, compared to the equivalent bitonic sorting network for different combinations of input and output sizes.

To better understand the origin of this reduction, the 3D surfaces shown in Figure 5.13 are sliced along the input-size axis and compared with the corresponding network-cost curves. Figure illustrates one such slice for the case of 8 outputs with the corresponding network-cost curves. Similar slices (Without the depth splitup curves) for other output sizes are provided in Appendix A.4.

From Figure 5.14, it can be observed that the overall profile of the depth-reduction curves is consistent with the changing gap between the logic depth of the sloppy selection networks and the cost of its equivalent bitonic sorting networks.

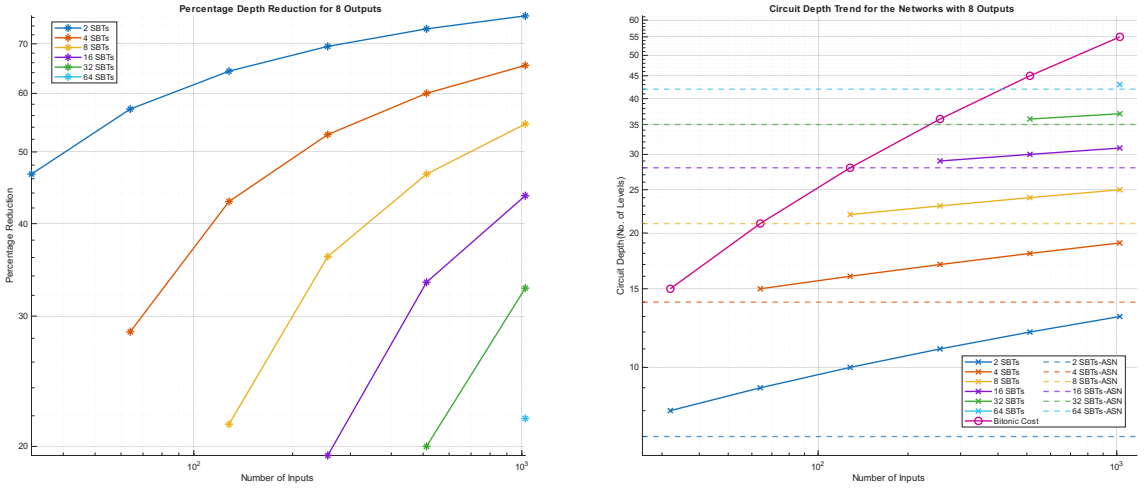


Figure 5.14: Depth reduction curves for all 8 output sloppy selection networks and the corresponding depth split up. The dotted lines on the depth split up plot on the right represents the depth of the accurate selection network stage present in the proposed networks, while the solid lines represent the total depth of the network

5.2.5 Power and Energy

This section analyzes the power and energy characteristics of the proposed networks. The dynamic power consumption closely follows the area trends observed in Figure 5.5. However, power alone does not fully capture the performance characteristics of the architecture, as it is strongly coupled with circuit delay. Therefore, EDP is used as a combined metric to evaluate overall performance.

Figure 5.15 shows the EDP trends for the proposed sloppy selection networks, where each surface corresponds to a different number of SBTs.

It is important to note that lower EDP values indicate more efficient designs, corresponding to lower energy consumption and reduced delay. From Figure 5.15, it can be observed that the most efficient configuration corresponds to the S-top(2,32) and S-bottom(2,32) with 2 SBTs which has the least amount of area and logic depth, whereas the least performant configuration correspond to S-top(16,1024) and S-bottom(16,1024) with 32 SBTs which has the largest area and the most depth. In general, architectures with fewer logic elements and lower depth achieve better EDP performance.

While EDP captures combined energy and delay efficiency, it does not fully reflect the energy cost per selected output. To address this, the EPS metric is introduced, defined as the ratio of energy to the number of outputs (i.e., number of selected elements). Figure 5.16 shows the EPS trends for the proposed networks.

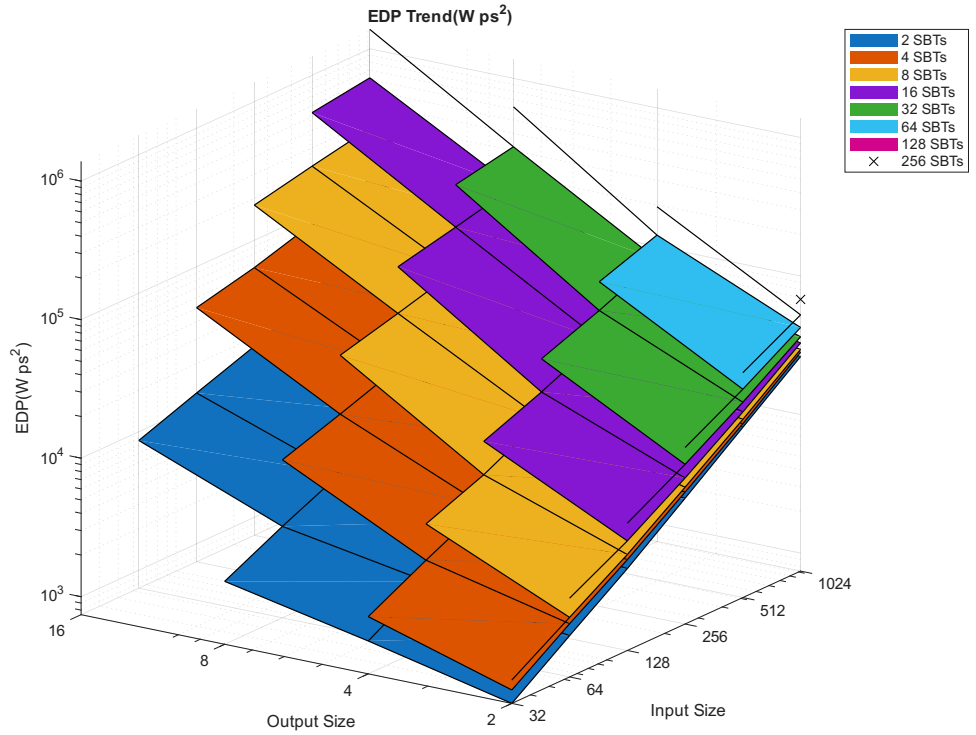


Figure 5.15: Trend in Energy-Delay Product for the sloppy selection networks based on the proposed architecture for different combinations of input and output sizes.

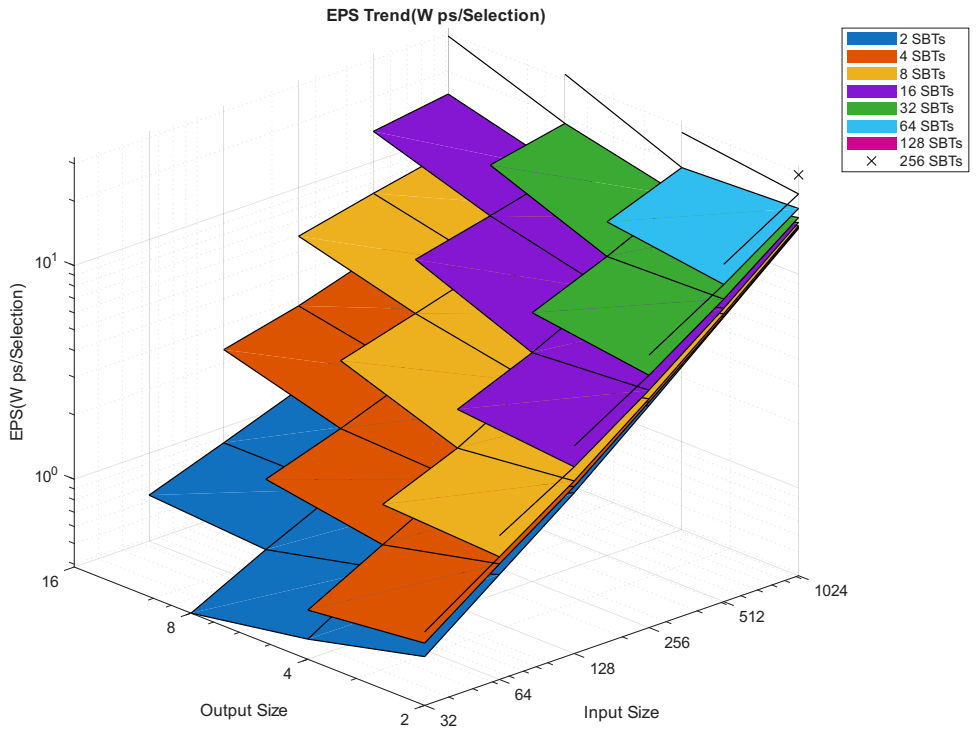


Figure 5.16: Trend in energy per selection for sloppy selection networks based on the proposed architecture for different combinations of input and output sizes.

From Figure 5.16, it can be observed that networks selecting 8 elements out of 32 inputs achieve the lowest energy per selection. This indicates that, even if such configurations are not globally optimal in terms of EDP, they provide the most energy-efficient selection operation on a per-output basis. In this context, configurations such as S-top(8,32) or S-bottom(8,32) may represent a favorable trade-off depending on energy constraints.

Finally, it should be noted that both EDP and EPS can be influenced by additional architectural optimizations. Delay optimization typically increases area and energy, which may degrade EPS, while area optimization can reduce EPS. It should also be noted that optimizations may affect EDP in either direction depending on the resulting delay and area impact. Therefore, the final efficiency depends strongly on the specific optimization strategy applied.

5.2.6 Reduction in Power

Power reduction estimations are more complicated than area or depth reduction estimations as it is time and resource intensive to synthesize and simulate all possible bitonic sorting networks to obtain power. But, since the power consumption is directly related to the area of the network as discussed earlier, it is safe to assume that the shape of the power reduction surfaces would roughly follow the area reduction surfaces shown in Figure 5.8.

5.2.7 Impact of MFCaS and MFCaSe Units

This section discusses the impact of the proposed MFCaS and MFCaSe comparison operators on the performance of the implemented networks. The evaluation is carried out by comparing area, delay, and EDP across different accurate selection networks. Since the only difference between CaS and CaSe units is a single 2:1 multiplexer, the results obtained for CaS units provide a good estimate for CaSe performance. This simplification significantly reduces additional simulation time and resource usage. Therefore, only CaS units are analyzed in the following sections, and conclusions for CaSe units are inferred directly from these results.

To perform this analysis, existing ASNs used in the proposed architecture are synthesized using both traditional CaS and MFCaS units. A timing constraint corresponding to 70% of the unconstrained delay is applied consistently across all designs. The resulting implementations are then compared side by side. Table 5.2 presents the delay comparison for the synthesized circuits. The table includes the largest network implemented using the Halflife architecture and the smallest network based on the odd-even architecture. This selection is motivated by the fact that odd-even selection networks are more efficient than Halflife networks at smaller input sizes [8].

Type	Input Size	Output Size	Trad.CaS /MFCaS	Delay(ns)	% Reduction(↓) or Increase(↑)
Halflife	1024	16	Trad.CaS	14.574	76.56% ↓
Halflife	1024	16	MFCaS	3.416	
odd-even	32	2	Trad.CaS	1.852	63.07% ↓
odd-even	32	2	MFCaS	0.701	

Table 5.2: Comparison of Halflife and odd-even ASNs in terms of delay. The reported percentage change indicates the variation observed when replacing traditional CaS units with MFCaS units.

From Table 5.2, it can be observed that MFCaS units achieve a substantial reduction in delay compared to traditional CaS units. However, this improvement is not reflected in the area results. As shown in Table 5.3, the use of MFCaS units leads to a modest increase in area across both architectures. This outcome is somewhat unexpected and suggests that further investigation is required to better understand the trade-off introduced by the proposed design. Moreover, the trend observed in the power results differs significantly from that of the area results. Given the close relationship between area and power in digital circuits, this discrepancy is also unexpected and warrants further analysis.

Type	Input Size	Output Size	Trad.CaS /MFCaS	Area(μm^2)	% Reduction(↓) or Increase(↑)
Halflife	1024	16	Trad.CaS	7.769×10^5	27.15% ↑
Halflife	report	16	MFCaS	10.664×10^5	
odd-even	32	2	Trad.CaS	4.848×10^3	27.04% ↑
odd-even	32	2	MFCaS	6.159×10^3	

Table 5.3: Comparison of Halflife and odd-even ASNs in terms of silicon area. The reported percentage change indicates the variation observed when replacing traditional CaS units with MFCaS units.

A similar comparison is conducted for the EDP to evaluate overall circuit performance. As shown in Table 5.4, MFCaS units provide a significant improvement over traditional CaS units, demonstrating a strong reduction in EDP across both network types.

Type	Input Size	Output Size	Trad.CaS /MFCaS	EDP ($W\,ps^2$)	% Reduction(↓) or Increase(↑)
Halflife	1024	16	Trad.CaS	9.826×10^7	97.15% ↓
Halflife	1024	16	MFCaS	2.796×10^6	
odd-even	32	2	Trad.CaS	9.415×10^3	83.7% ↓
odd-even	32	2	MFCaS	1.534×10^3	

Table 5.4: Comparison of Halflife and odd-even ASNs in terms of EDP. The reported percentage change indicates the variation observed when replacing traditional CaS units with MFCaS units.

Based on these observations, it can be concluded that the use of MFCaSe units would yield delay and EDP reductions comparable to those achieved with MFCaS units, along with a similarly modest increase in area. Considering the proposed selection network as a whole, which incorporates both MFCaS and MFCaSe units, the ASNs employing MFCaS units dominate in terms of delay performance, while the area overhead is distributed relatively evenly between MFCaS and MFCaSe components, as observed in the previous results. Overall, it is hypothesised based on the results that the proposed selection network can achieve approximately a 70% reduction in delay, a modest area increase of around 27%, and an EDP reduction of about 90%. These results demonstrate that the proposed MFCaS and MFCaSe architectures significantly outperform the conventional CaS and CaSe implementations.

5.3 Impact of SNR on Selection Performance

So far, we have analyzed the system- and circuit-level performance of the proposed circuits. In this section, we investigate how these performance characteristics vary with SNR. This brings us one step closer to understanding the impact of sloppy selection networks on decoding performance. To this end, we compute the pass probabilities described in Section 5.1 over an SNR range from 2 dB to 40 dB. For each SNR value, pass probabilities computed across different (N, k) configurations are averaged to obtain a single representative point, thereby providing a more robust system-level metric that is not biased toward any specific configuration. simply put, all points in Figure 5.1 are computed for a given SNR and averaged to provide a single point. This is repeated for different SNR values.

Figure 5.17 shows the resulting average pass probability as a function of SNR. As expected, the average pass probability increases with improving SNR, since the number of channel-induced errors decreases. Furthermore, the average pass probability serves as a direct indicator of the decoder's error-correction capability, making it a useful high-level performance metric.

From Figure 5.17, it can be observed that the selection performance of the sloppy selection network closely follows that of the ASN. This indicates that there is no significant performance degradation due to the introduced sloppiness, demonstrating that sloppy selection is a viable design choice for decoder implementation to tradeoff between hardware constraints and error correction performance.

To further quantify this behavior, the difference between the average pass probabilities of the accurate and sloppy selection networks(i.e. the gap between the blue and red line from figure 5.17) is computed and plotted in Figure 5.18. This plot illustrates how closely the sloppy selection network matches the performance of the accurate selection network across different SNR values. A noticeable peak occurs at 6 dB, indicating that the performance gap is largest at this SNR. Beyond this point, the gap decreases as SNR increases, showing that the sloppy selection network becomes increasingly similar in performance to the accurate selection network at higher SNR regimes.

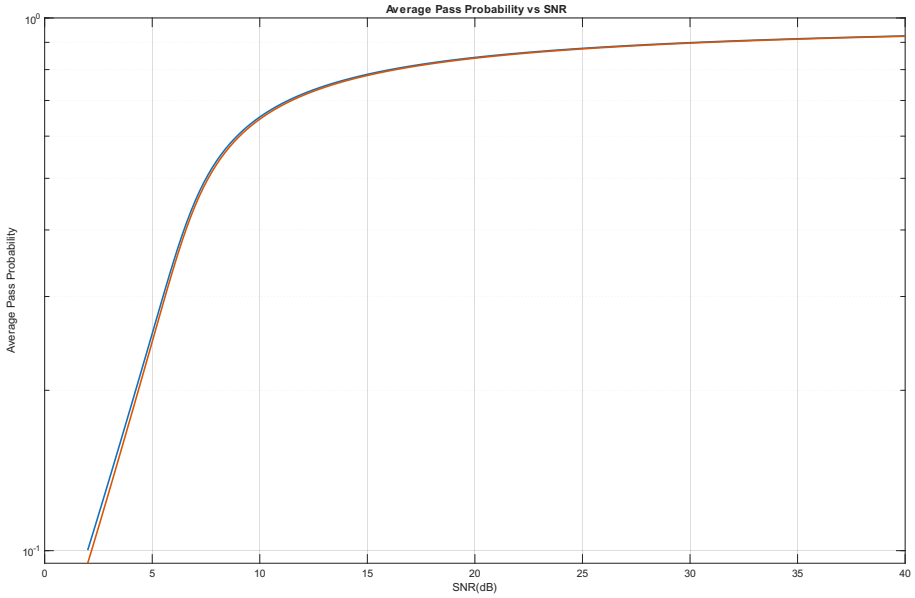


Figure 5.17: Average pass probability trend with respect to different SNR values of the demodulated signal(higher is better)

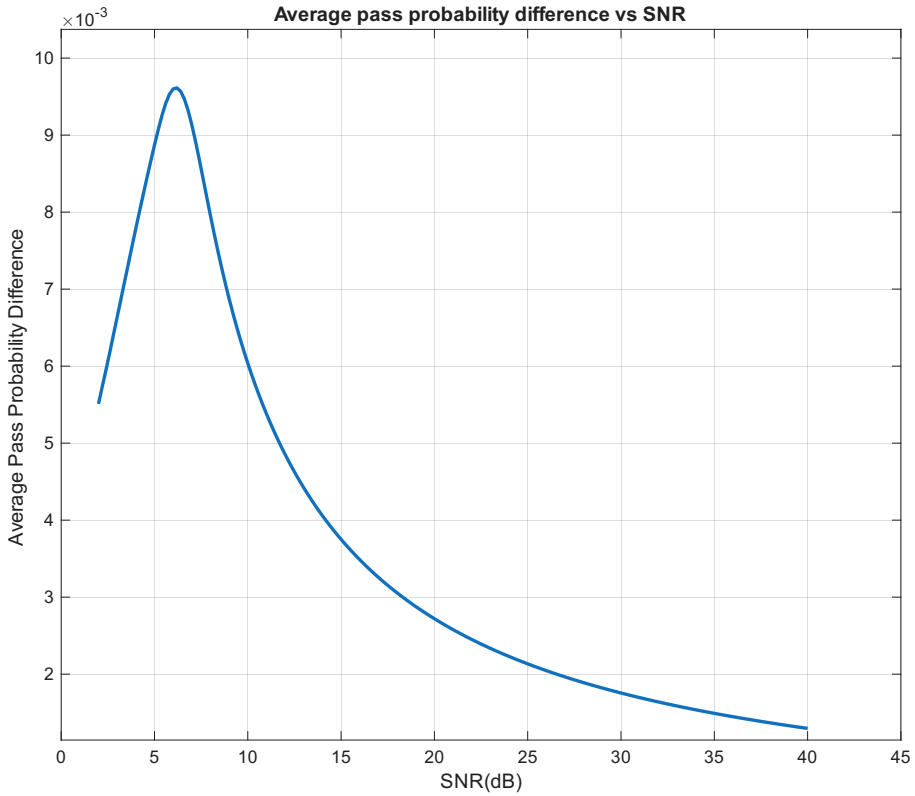


Figure 5.18: Difference between the average pass probabilities of an accurate and sloppy selection network(lower is better)

Finally, the average sloppiness ($\bar{\sigma}$) of the selected sets is computed and plotted as a function of SNR. As discussed in Section 5.1, this metric provides a coarse

indication of selection performance. Figure 5.19 compares the trends of $\bar{\sigma}$ and the average pass probability difference across different SNR values. The figure shows that the sloppiness curve broadly follows the difference in selection performance of the selection networks. While the correspondence is only approximate, these results indicate that the sloppiness metric is a promising basis for characterizing selection performance and may, with further refinement, serve as a predictor of network performance.

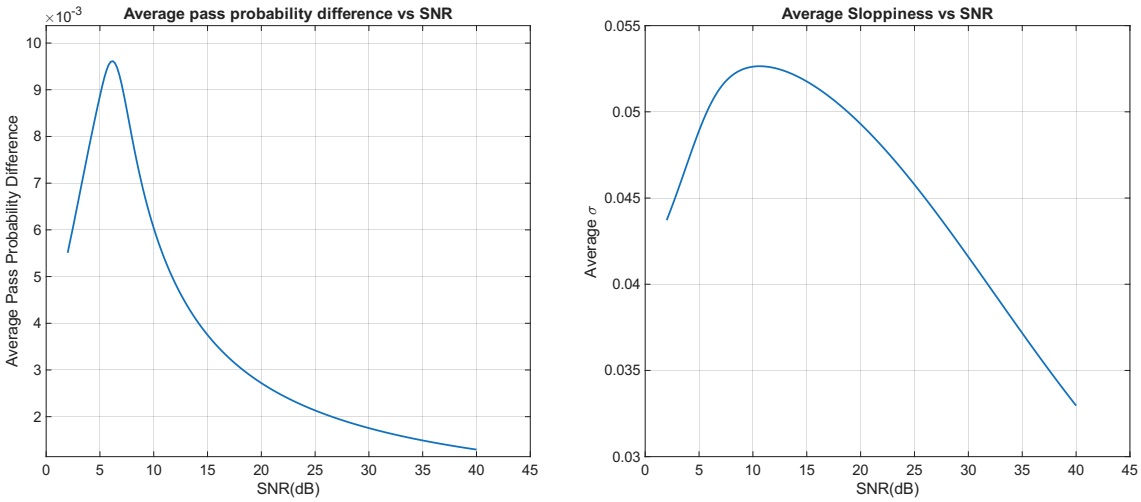


Figure 5.19: comparison between the trends in average sloppiness and average pass probability difference with respect to the SNR (lower is better).

6

Discussion

In this chapter we will discuss about how to improve upon the current status of this research, potential improvements to the existing proposed architecture, discrepancies seen in results compared to the work of Kenneth et.al [8] and finally the societal, ethical and ecological impact due to this research

6.1 Formal Analysis of Selection Performance

The proposed networks were developed primarily based on experimental observations. Although attempts were made to formalize the selection performance of the proposed architecture, as discussed in Chapter 5, the current approach is still unable to accurately predict network performance. At present, there is no formal proof that precisely characterizes the selection performance of the proposed network architecture. Therefore, developing such a proof is essential for achieving a deeper theoretical understanding of the network behavior.

Initial attempts were made toward this objective; however, due to limitations in both time and expertise, the work could not be completed within the scope of this research. As a result, this remains an important direction for future work. The following outlines the preliminary approach that was considered.

First, the selection performance of a SBT must be characterized. Consider a minimum-finder binary tree that selects the minimum value from an input set I. Let P_M denote the probability that, at a given level of the binary tree, the nodes at that level contain the M smallest values from the input set I, where M is equal to the number of nodes present at that level. Let the set of these M elements be denoted as P.

At the first level of the tree, this probability is equal to 1, since the sets P and I are identical. Similarly, at the final level, the probability is also equal to 1 because only a single node remains, corresponding to the minimum value of the entire input set I. However, a general method for computing this probability for the intermediate levels of the tree has not yet been derived.

If the probability P_M can be determined for all levels of the binary tree, then it becomes possible to evaluate the selection performance of an SBT based on the level at which the tree is sliced. This would provide a direct metric for the selection performance of an SBT that selects k elements from N/Q inputs. The

same methodology can also be extended to maximum-finder binary trees and the corresponding maximum-finder SBTs.

Once the selection performance of the SBTs is formally characterized, the overall analysis of the proposed architecture becomes significantly more straightforward. Since the probability of an ASN correctly selecting the k minimum values from its inputs is always equal to 1, the remaining challenge is to determine how the probabilities associated with the individual SBTs accumulate in the proposed the network. By analyzing this accumulated probability of identifying the $Q \cdot k$ smallest values, together with the known behavior of the ASN, it should then be possible to formally derive the probability that the proposed network correctly selects the k smallest or largest values from a total of N inputs.

6.2 Potential Sliced Binary Tree Improvements

Current SBTs employ only MFCaSe units, which compare two values at a time. However, alternative modified tree topologies discussed by Bilgiday Yuce et al. [21] are potentially faster and more efficient than the binary tree structures currently used. These architectures could be explored to further reduce both area and delay by utilizing array-based comparator topologies capable of comparing more than two elements simultaneously while directly generating the minimum value.

Additionally, SBT variants incorporating 4-input array-based comparators, similar to those used in the heterogeneous tree structures discussed by Bilgiday Yuce et al., could be investigated to study the trade-offs between timing and area efficiency. Beyond this, sliced quad-tree structures may also be considered as alternatives to SBTs in order to further reduce delay at the cost of increased area. Furthermore, several specialized heterogeneous tree-based approaches could be explored for sloppy selection of values from the input set, potentially enabling additional improvements in hardware efficiency.

6.3 Exploring impact on decoding performance

While the proposed sloppy selection networks perform well in isolation, their standalone performance may not fully reflect their effectiveness in practical decoding scenarios. This is because even accurate selection based on the soft information provided by the decoder may still fail to correctly identify the erroneous bit in certain cases. Therefore, it is important to evaluate the performance of the sloppy selection network within an actual decoding framework. Based on the observations so far, we hypothesize that decoders implementing the proposed sloppy selection network can achieve significant reductions in area, power, and delay requirements while incurring only a modest degradation in decoding performance.

6.4 Discrepancies in Area and Delay Results

It was shown in Section 5.2 that the synthesized area and delay closely matched their corresponding theoretical estimates, namely the number of CaS/CaSe units and the number of network levels, respectively. This correspondence enabled the theoretical estimates to be decomposed into their constituent components, facilitating a more detailed analysis of the synthesized area and delay.

In contrast, [8] employed a similar methodology to estimate the area and depth of their ASNs, but reported significant discrepancies between the estimated and synthesized results for both area and delay. One possible explanation for this difference lies in the synthesis methodology. Kenneth et al. synthesized their networks by optimizing for EDP [29], whereas the networks in this work were synthesized under a fixed timing constraint. Additionally, the discrepancy may be influenced by differences in the standard-cell libraries used during synthesis. Kenneth et al. employed a different technology library, which may not provide the same level of optimization opportunities as the ASAP7 library used in this work.

6.5 Societal, Ethical and Ecological Aspects

This work focuses on optimizing power consumption and silicon area. Improvements in both metrics can indirectly reduce the global carbon footprint if the proposed design is widely adopted. Lower power consumption directly translates to reduced energy drawn from the power grid. Additionally, a smaller silicon area generally implies a reduced number of transistors, which in turn lowers the amount of raw materials and energy required during chip fabrication. These reductions further decrease the carbon footprint of the chip over its entire product lifecycle.

Furthermore, if the proposed techniques are implemented in decoders for next-generation mobile networks, they may also improve latency, reliability, and throughput. Such improvements enable more efficient mobile networks at the system level. Increased decoding efficiency can reduce the number of retransmissions and allow a larger number of users to be supported per network, leading to additional reductions in overall power consumption. Collectively, these system-level gains contribute to a reduced carbon footprint of mobile communication infrastructure, while also providing society with faster and more reliable mobile connectivity. As a critical communication infrastructure, such improvements can positively impact quality of life by enhancing access to digital services, connectivity, and information.

7

Conclusion

This thesis investigated the possibility of reducing the hardware constraints of decoder circuits implementing modern decoding schemes that utilize soft information for FEC. Prominent examples of such decoding algorithms include GRAND [1], ORBGRAND [2], and the Chase class of algorithms [3, 4, 5]. The work specifically focused on the sorting circuitry used within these decoders, as sorting networks often form a major hardware bottleneck in terms of area, power consumption, and delay [6]. These sorting networks are typically responsible for ordering the reliability information generated by the decoder, which is then used for error detection and correction.

To address this problem, accurate selection networks were initially explored and were found to provide moderate reductions in hardware complexity. However, it was hypothesized that greater improvements could be achieved through the use of sloppy selection networks, which avoid fully sorting all reliability values. This hypothesis was motivated by the observation that the reliability information produced by soft-decision decoding algorithms is inherently noisy. Consequently, even perfect sorting does not guarantee correct identification and correction of all erroneous bits. Initial computer simulations demonstrated that sloppy selection achieved performance comparable to accurate selection networks, particularly at sufficiently high SNR conditions.

The average difference in pass probabilities between accurate and sloppy selection networks was found to be on the order of 10^{-3}, indicating that sloppy selection networks are a viable alternative for reducing hardware complexity. To better characterize the degree of sloppiness introduced by these networks, a "sloppiness" metric was proposed. Although this metric provided a rough estimate of selection performance, additional work is required to further refine and validate it.

Following this, several data structures, sorting algorithms, selection algorithms, and their corresponding hardware implementations were explored. Based on these investigations, a generalized architecture for constructing sloppy selection networks was proposed. The architecture consisted of cascoded SBTs cascaded with a single ASN. This framework provided a configurable trade-off between sloppiness and hardware constraints such as area, power, and delay. In addition, new comparator architectures, namely the MFCaS and MFCaSe units, were introduced to further reduce comparator complexity [29].

All possible combinations of the proposed architectures incorporating MFCaS and MFCaSe units were synthesized for predefined input and output word counts. Their

hardware characteristics were then compared against equivalent bitonic sorting network implementations to quantify improvements in area, power consumption, and delay.

The synthesized sloppy selection circuits occupied approximately one quarter of the area required by equivalent bitonic sorting networks, with maximum area reductions reaching nearly 96%. This reduction is highly significant given the relatively small degradation in selection performance observed earlier. Furthermore, reductions of up to 80% in network depth were achieved. Considering that no additional optimization techniques such as pipelining were employed to further reduce the critical path delay, these improvements are particularly noteworthy.

In terms of power consumption, it was hypothesized that the reduction trend would closely follow the observed area reductions, since circuit area is generally correlated with power consumption. The performance of the proposed sloppy selection circuits was also evaluated using EDP and EPS metrics. The circuit exhibiting the lowest EDP was found to be a configuration selecting 2 elements from 32 inputs using two SBT stages. In contrast, the network selecting 8 elements from 32 inputs using two SBTs achieved the lowest EPS, indicating superior energy efficiency per selection operation.

The impact of the proposed MSB-first comparator logic was also investigated. Based on the observed results, it was estimated that the proposed MFCaS and MFCaSe units could achieve approximately 70% reduction in delay and nearly 90% reduction in EDP, while incurring only a modest area increase of approximately 25% compared to traditional CaS and CaSe-based sloppy selection networks. These results demonstrate that substantial improvements in delay and energy efficiency were achieved with relatively small area overhead.

Overall, the results presented in this thesis demonstrate that sloppy selection networks can significantly reduce the hardware complexity associated with reliability selection in modern soft-decision decoding architectures. This makes reliability selection substantially more area- and energy-efficient compared to conventional sorting-based approaches.

Although the presented results strongly indicate the viability of sloppy selection networks, further investigation is required to evaluate their performance within complete practical decoder implementations. If comparable decoding performance can be maintained relative to accurate selection networks, the proposed architectures could have a significant impact on the design of future 5G and 6G decoder hardware. Future work may also focus on refining existing SBT structures, developing improved sloppiness metrics, and exploring new algorithms and architectures for efficient approximate selection.

Bibliography

[1] R. Hadavian, X. Huang, D. Truhachev, K. El-Sankary, H. Ebrahimzad, H. Najafi, Y. Ge, and A. Zokaei, "Ordered reliability direct error pattern testing decoding algorithm," *IEEE Transactions on Communications*, vol. 73, no. 11, pp. 9987–10 000, 2025.

[2] K. R. Duffy, W. An, and M. Médard, "Ordered reliability bits guessing random additive noise decoding," *IEEE Transactions on Signal Processing*, vol. 70, pp. 4528–4542, 2022.

[3] D. Chase, "Class of algorithms for decoding block codes with channel measurement information," *IEEE Transactions on Information Theory*, vol. 18, no. 1, pp. 170–182, 1972.

[4] R. Pyndiah, "Near-optimum decoding of product codes: block turbo codes," *IEEE Transactions on Communications*, vol. 46, no. 8, pp. 1003–1010, 1998.

[5] S. Hirst, B. Honary, and G. Markarian, "Fast Chase algorithm with an application in turbo decoding," *IEEE Transactions on Communications*, vol. 49, no. 10, pp. 1693–1699, 2001.

[6] A. Riaz, A. Yasar, F. Ercan, W. An, J. Ngo, K. Galligan, M. Médard, K. R. Duffy, and R. Tugce Yazicigil, "A sub-0.8-pJ/bit universal soft-detection decoder using ORBGRAND," *IEEE Journal of Solid-State Circuits*, vol. 60, no. 7, pp. 2645–2659, 2025.

[7] D. Chytas and V. Paliouras, "Approximate sorting check node processing in non-binary LDPC decoders," in *2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2020, pp. 1–4.

[8] K. Peter, L. Svensson, C. Fougstedt, and P. Larsson-Edefors, "Hardware considerations for selection networks," in *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)*, 2019, pp. 40–45.

[9] S. M. Abbas, T. Tonnellier, F. Ercan, M. Jalaleddine, and W. J. Gross, "High-throughput and energy-efficient VLSI architecture for ordered reliability bits grand," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 6, pp. 681–693, 2022.

[10] C. Condo, "A fixed latency ORBGRAND decoder architecture with LUT-aided error-pattern scheduling," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 5, pp. 2203–2211, 2022.

[11] H. Ji, Y. Shen, W. Song, Z. Zhang, X. You, and C. Zhang, "Hardware implementation for belief propagation flip decoding of polar codes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 3, pp. 1330–1341, 2021.

[12] J. Kim, S. Han, D. Kam, B. Y. Kong, and Y. Lee, "A design framework for cost-efficient sorters with arbitrary input/output constraints," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 71, no. 12, pp. 5410–5419, 2024.

[13] D. E. Knuth, *The Art of Computer Programming: Sorting and Searching, volume 3.* Addison-Wesley Professional, 1998.

[14] K. E. Batcher, "Means for merging data," Feb. 18 1969, uS Patent 3,428,946.

[15] K. E. Batcher, "Sorting networks and their applications," in *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, ser. AFIPS '68 (Spring). New York, NY, USA: Association for Computing Machinery, 1968, p. 307–314. [Online]. Available: https://doi.org/10.1145/1468075.1468121

[16] H. Peters, O. Schulz-Hildebrandt, and N. Luttenberger, "A novel sorting algorithm for many-core architectures based on adaptive bitonic sort," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, 2012, pp. 227–237.

[17] H. Peters, O. Schulz-Hildebrandt, and N. Luttenberger, "Fast in-place, comparison-based sorting with CUDA: a study with bitonic sort," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 7, pp. 681–693, 2011. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.1686

[18] T.-T. Nguyen-Ly and K.-A. Nguyen, "An efficient sorting algorithm and its hardware architecture for general applications," *Circuits, Systems, and Signal Processing*, pp. 1–17, 2025.

[19] D. E. Knuth, *The Art of Computer Programming: Fundamental Algorithms, Volume 1.* Addison-Wesley Professional, 1997, ch. 2.3, pp. 308–423.

[20] D. Hendry, "Comparator trees for winner-take-all circuits," *Neurocomputing*, vol. 62, pp. 389–403, 2004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231204002796

[21] B. Yuce, H. F. Ugurdag, S. Gören, and G. Dündar, "Fast and efficient circuit topologies for finding the maximum of n k-bit numbers," *IEEE Transactions on Computers*, vol. 63, no. 8, pp. 1868–1881, 2014.

[22] J. G. Proakis, *Digital signal processing: principles, algorithms, and applications, 4/E.* Pearson Education India, 2007.

[23] "IEEE standard verilog hardware description language," *IEEE Std 1364-2001*, pp. 1–792, 2001.

[24] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "ASAP7: A 7-nm finFET predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S002626921630026X

[25] Z. B. Kaykac Egilmez, L. Xiang, R. G. Maunder, and L. Hanzo, "The development, operation and performance of the 5G] Polar Codes, year=2020, volume=22, number=1, pages=96-122, keywords=Polar codes;3GPP;5G mobile communication;Parity check codes;Decoding;Uplink;5G;polar coding;forward error correction;new radio;5G wireless system standardization, doi=10.1109/COMST.2019.2960746," *IEEE Communications Surveys Tutorials*.

[26] C. Fougstedt, private communication.

[27] D. M. Harris and S. L. Harris, "Chapter 2 - combinational logic design," in *Digital Design and Computer Architecture*, D. M. Harris and S. L. Harris, Eds. Burlington: Morgan Kaufmann, 2007, pp. 51–100. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780123704979500032

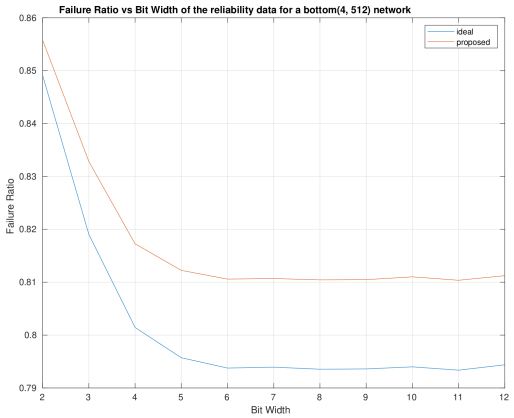
[28] D. M. Harris and S. L. Harris, "Chapter 5 - digital building blocks," in *Digital Design and Computer Architecture*, D. M. Harris and S. L. Harris, Eds. Burlington: Morgan Kaufmann, 2007, pp. 233–286. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780123704979500068

[29] L. Svensson, private communication.

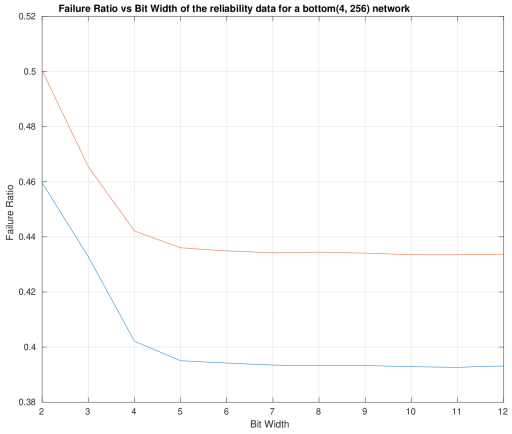
A

Appendix 1

A.1 Failure Ratio vs Data Width of the reliability data



(a) Failure Ratio vs Data Width (Bitwidth) of a bottom(4, 512) and S-bottom(4, 512) networks



(b) Failure Ratio vs Data Width (Bitwidth) of bottom(4, 256) and S-bottom(4, 256) networks

Figure A.1

A.2 Proof for the Size of SBT networks

The number of nodes n at any level l of a N-input binary reduction tree can be given as

$$n = \frac{N}{2^l} \tag{A.1}$$

Then the total number of nodes S in a binary reduction tree upto level l can be written as

$$S = \sum_{i=0}^{l} \frac{N}{2^i} \tag{A.2}$$

Equation A.2 is a finite geometric series of ratio $\frac{1}{2}$. any geometric series of a initial term a and a common ratio r can be reduced as follows,

$$\sum_{i=0}^{n} ar^i = \frac{a(1-r^{n+1})}{1-r} \tag{A.3}$$

applying this rule to equation A.2 gives us,

$$S = \sum_{i=0}^{l} \frac{N}{2^i} = \frac{N(1-(\frac{1}{2})^{l+1})}{1-\frac{1}{2}} \tag{A.4}$$

This can then be simplified as follows,

$$S = \frac{N(1-(\frac{1}{2})^{l+1})}{1-\frac{1}{2}} \tag{A.5}$$

$$= 2N - \frac{2N}{2^{(l+1)}} \tag{A.6}$$

$$= 2N - \frac{2N}{2^l \cdot 2} \tag{A.7}$$

$$\implies S = 2N - \frac{N}{2^l} \tag{A.8}$$

Substituting equation A.1 into A.8, we get,

$$S = 2N - n \tag{A.9}$$

equation A.9 gives us the number of nodes for a N-Input binary reduction tree upto level l which has n nodes. For a N input SBT network the topmost level contains k nodes and the base nodes will not be replaced by a CaSe unit, applying these implications to equation A.9 we get,

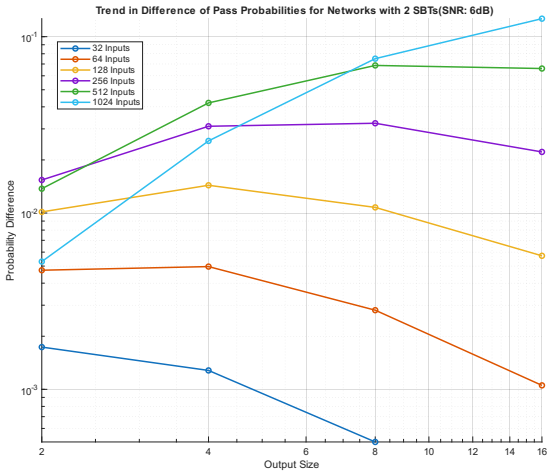
$$S_{SBT} = 2N - k - N = N - k \tag{A.10}$$

Thus the size of a SBT network that selects k elements out of N can be given as

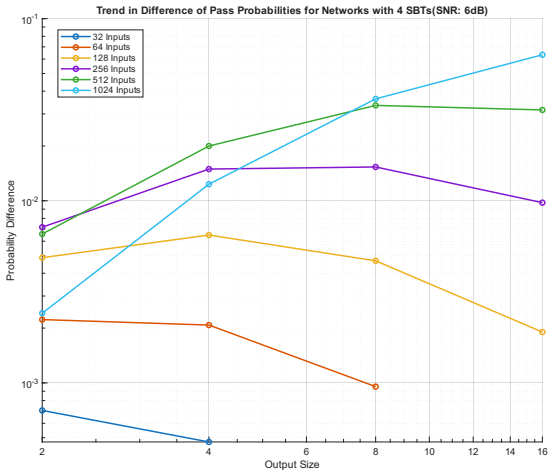
$$\boxed{S_{SBT} = N - k}$$

A.3 Trend in Difference of Pass Probabilities for Different number of SBTs

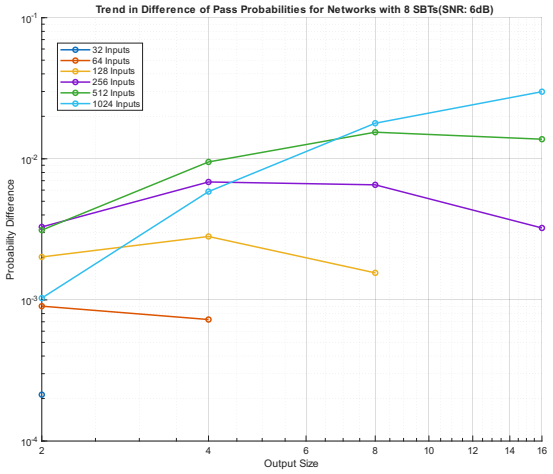
This section shows how the difference in pass probabilities between accurate and sloppy selection varies with output size for a given number of SBTs (lower is better). Each line represents a different number of inputs. The trends are plotted at an SNR of 6 dB. Results for 128 and 256 SBTs are omitted, as they can be inferred from the 3D plot.



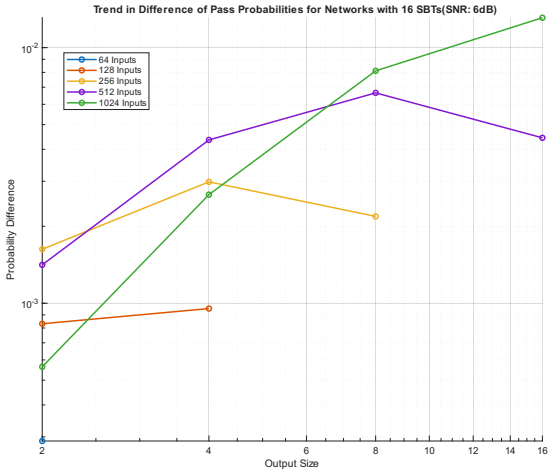
(a)



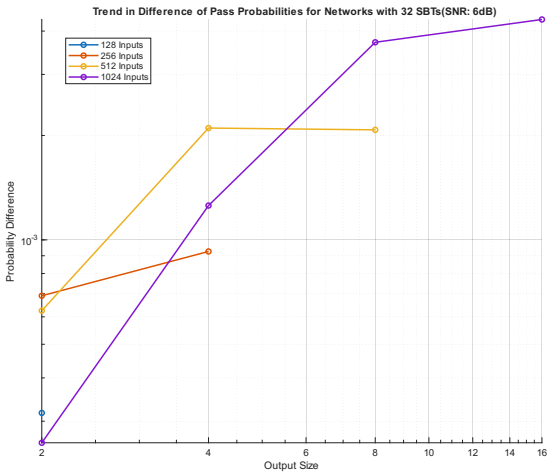
(b)



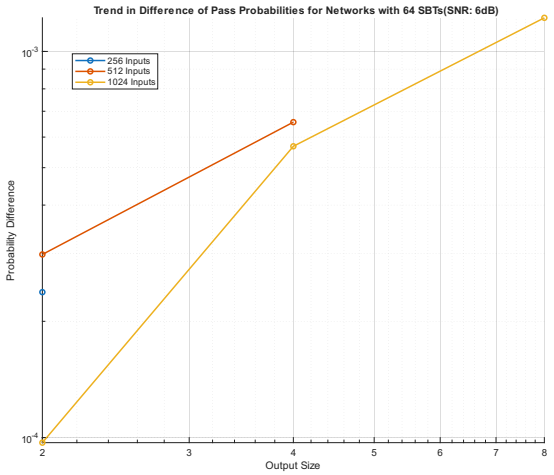
(c)



(d)



(e)

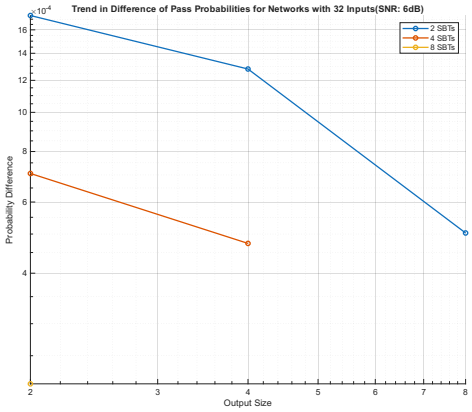


(f)

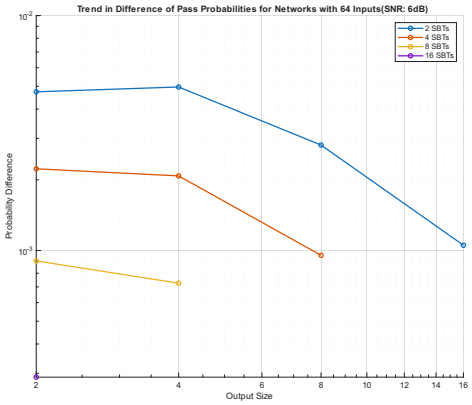
Figure A.2

A.4 Difference of Pass Probabilities for Different Input sizes

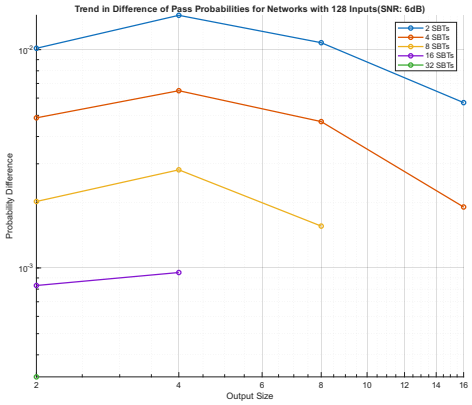
This section shows how the difference in pass probabilities between accurate and sloppy selection varies with output size for a given number of input words in the network (lower values are better). Each line corresponds to a different number of SBTs. The trends are plotted at an SNR of 6 dB.



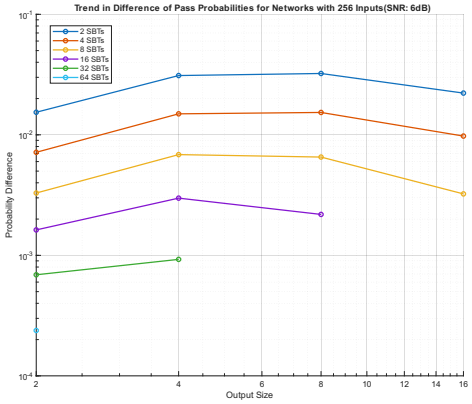
(a)



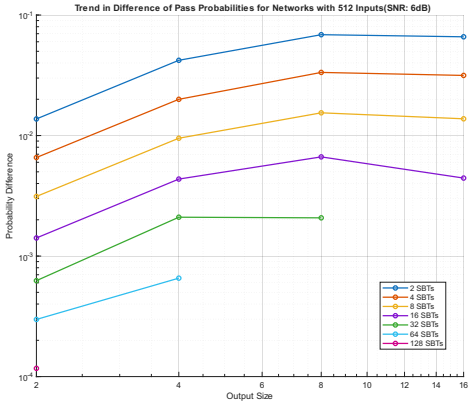
(b)



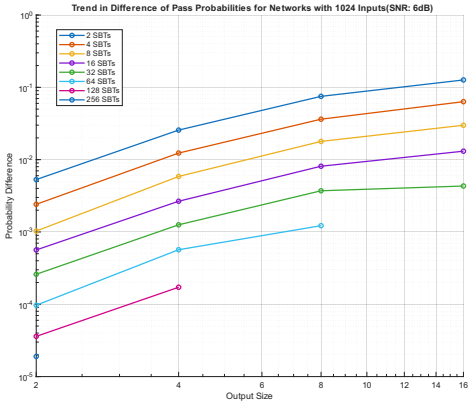
(c)



(d)



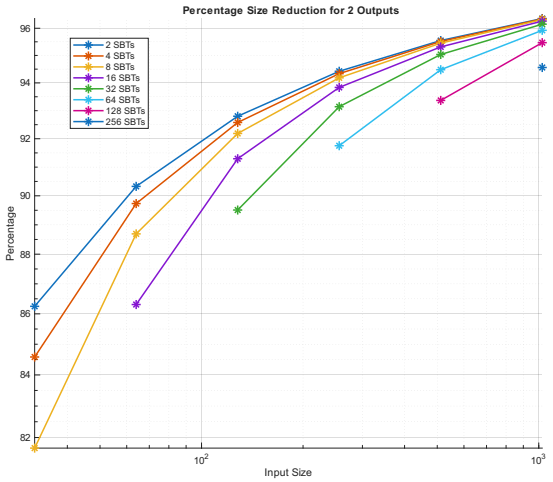
(e)



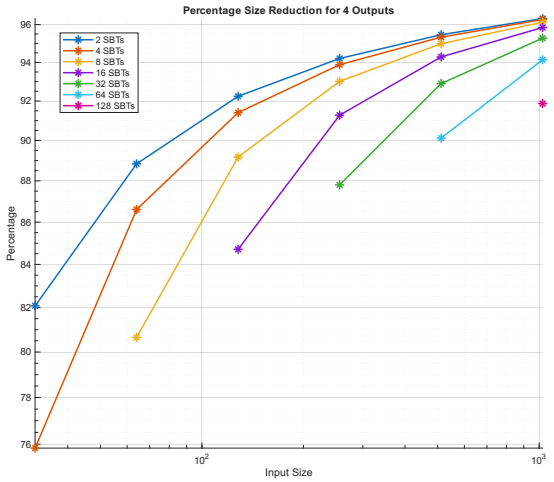
(f)

A.5 Size Reduction of Sloppy Selection Networks

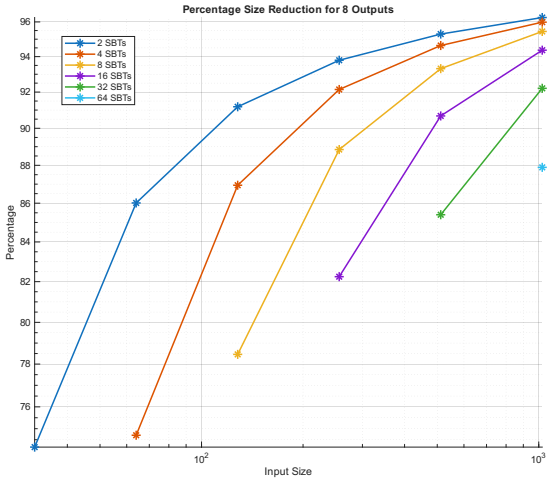
This section shows individual slices of the size reduction 3D surface plot sliced along the axis of input-size. Each slice is done along a disctinct output-size and each curve represents a certain number of SBTs in the network.



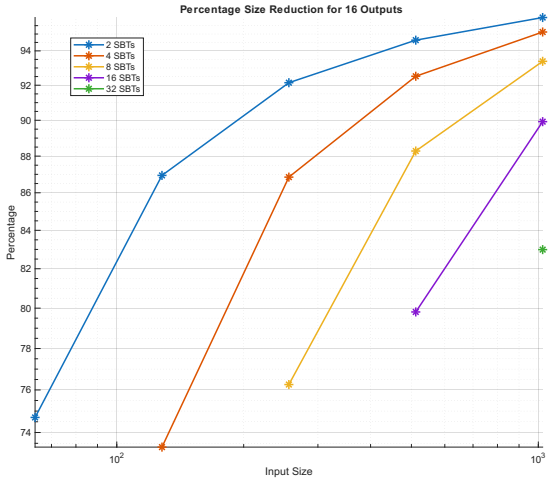
(g)



(h)



(i)

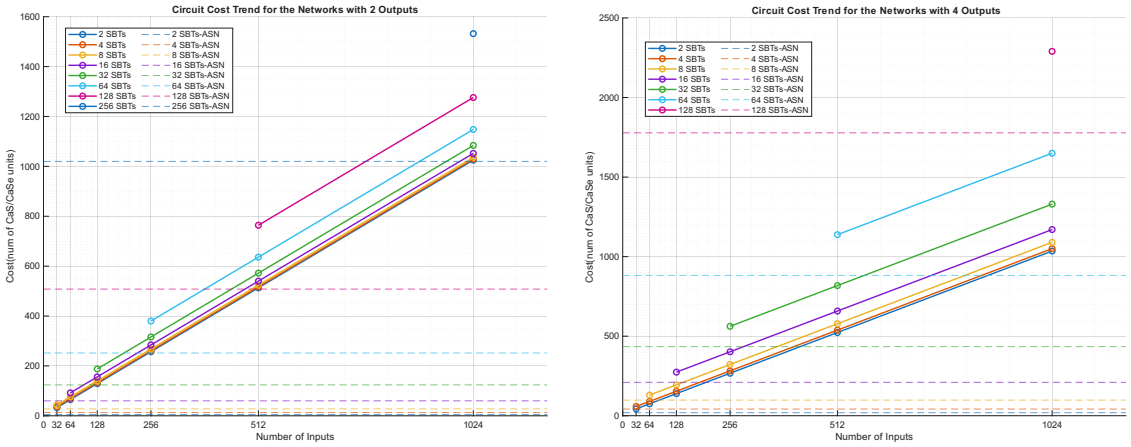


(j)

Figure A.3

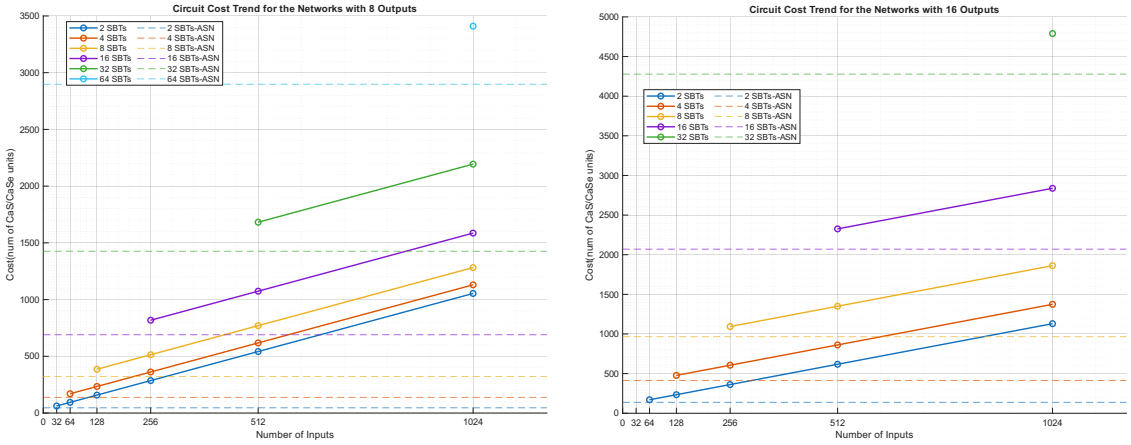
A.6 Circuit Cost Split-up of Sloppy Selection Networks

This section shows theslices of circuit cost trend surfaces for all the synthesized networks sliced along the input size axis. The dotted lines represent the size of the accurate selection network stage present in the proposed networks, while the solid lines represent the total cost of the network(Note: this is a linear-linear graph)



(a)

(b)



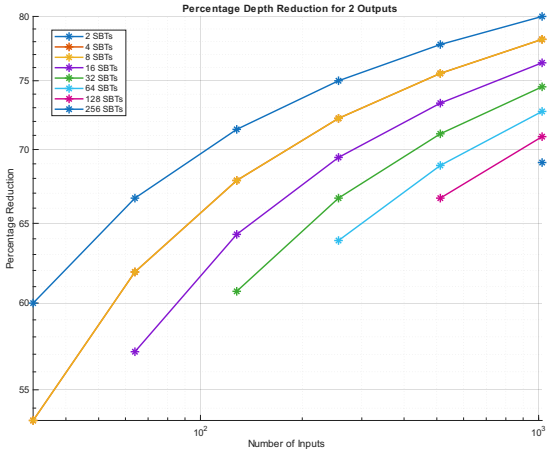
(c)

(d)

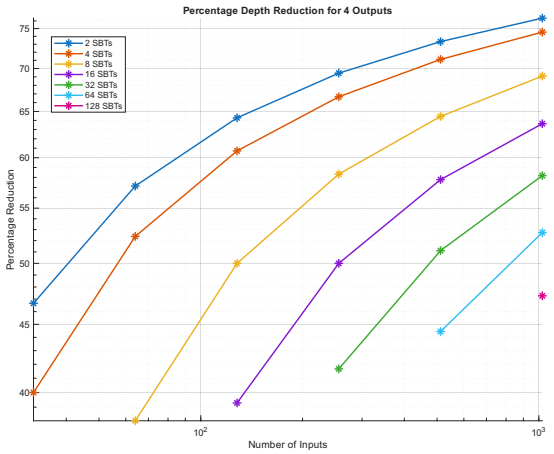
Figure A.4

A.7 Logic Depth Reduction of Sloppy Selection Networks

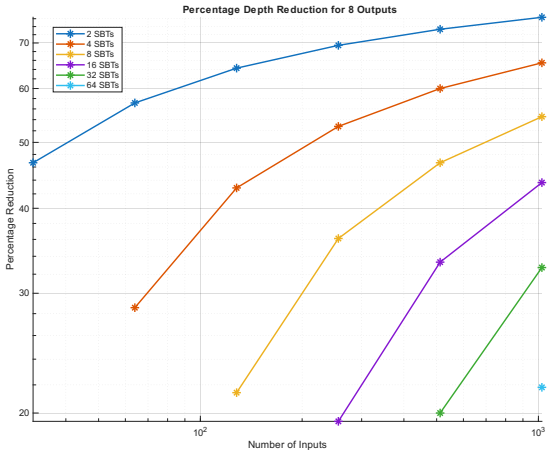
This section shows individual slices of the depth reduction 3D surface plot sliced along the axis of input-size. Each slice is done along a disctinct output-size and each curve represents a certain number of SBTs in the network.



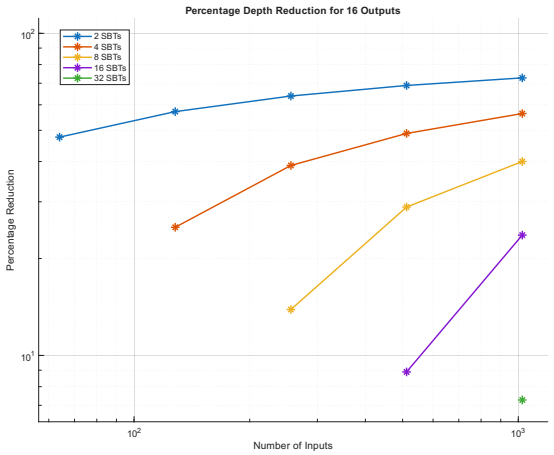
(a)



(b)



(c)



(d)

Figure A.5