Managing High-Speed Order Books Using Hybrid Binary-Linear Search Data Structures in FPGAs

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science in
in
Computer Science and Engineering by Research

by

Vaibhav Kashera
2019111003
vaibhav.kashera@research.iiit.ac.in

International Institute of Information Technology, Hyderabad
(Deemed to be University)
Hyderabad - 500 032, INDIA

June 2024

International Institute of Information Technology Hyderabad
Hyderabad, India

CERTIFICATE

This is to certify that work presented in this thesis proposal titled ***Managing High-Speed Order Books Using Hybrid Binary-Linear Search Data Structures in FPGAs*** by *Vaibhav Kashera* has been carried out under my supervision and is not submitted elsewhere for a degree.

_____ _____

Date Advisor: Dr. Suresh Purini

To my parents for always believing in me.

Acknowledgments

This thesis reflects not just my academic efforts but also the support and belief my advisor, Dr. Suresh Purini, and my family have placed in me. I am truly thankful for the guidance and motivation provided by Dr. Suresh Purini throughout my undergraduate studies. His commitment to high-quality research has greatly influenced me. The valuable feedback he offered has been crucial in improving my work and helping me grow. I also want to thank Mr. Abhishek Banerjee, CEO of Lotusdew Wealth, for giving me and my team the chance to take on this project. His technical support and insights from his own experience in trading have helped us understand the complex issues we wanted to solve. A big thanks to my friend and co-author, Siddhant, for all his contributions and for being there through the different challenges of this project. Sharing this journey with him has been a big part of our success. I'm grateful for my friends who made college life so enjoyable. Life at college would not have been the same without you all.

Most importantly, I owe everything to my parents. They've always believed in me and supported me through everything. Their love and encouragement have kept me going, even in tough times. I also want to thank my sister for always having faith in me, even when I doubted myself. As I stand at this significant milestone, I look back with immense gratitude towards everyone who has been a part of my journey. The lessons learned, the friendships forged, and the challenges overcome have all contributed to my personal and academic growth. Your belief in me has been my greatest motivation, and for that, I am endlessly thankful. To all of you who have contributed in ways big and small to my journey, thank you for being part of my story.

Abstract

The popularity of High-Frequency Trading (HFT) or algorithmic trading has surged in the last ten years, largely due to the exponential increase in computing power. Despite the evolution and optimization of software solutions tailored for HFT, challenges persist, notably due to inefficiencies associated with network stack overheads and the separation between kernel and user spaces. A pivotal element in any HFT framework is the construction of the order book, which serves as a critical snapshot of the market, underpinning trading strategies and executions.

In this thesis, we introduce an innovative, simple linear data structure designed for efficient order book tracking, coupled with a hybrid binary-linear search algorithm. This algorithm is specifically engineered to dynamically maintain the top bid and ask offers, which is crucial for reflecting the market depth on Field-Programmable Gate Arrays (FPGAs). This methodology is premised on the understanding that a significant portion of trading activities concentrates at the top levels of the bid and ask prices. Through design, analysis and experimentation, we demonstrate that our simple approach is scalable and practical, outperforming previous methods.

Contents

Chapter Page

1 Introduction . . . 1
1.1 Motivation . . . 1
1.2 What are FPGAs, and where do they step in? . . . 2
1.3 Summary of Contributions . . . 3
1.4 Thesis Organization . . . 3
1.5 Literature Review . . . 4

2 FPGAs and High Frequency Trading . . . 6
2.1 Reconfigurable Computing and FPGAs . . . 6
2.1.1 Field-Programmable Gate Arrays . . . 7
2.2 Real-World Applications of FPGAs . . . 10
2.2.1 In Cloud and Data centres . . . 10
2.2.2 In the Internet of Things . . . 11
2.2.3 Other Areas . . . 11
2.3 Market Mechanics . . . 12
2.3.1 Order Book . . . 12
2.3.2 Algorithmic trading . . . 13
2.3.3 High Frequency Trading . . . 14
2.3.4 Importance of Low Latency in HFT . . . 14
2.4 Role of FPGAs in High Frequency Trading . . . 15
2.4.1 Trading Infrastructure . . . 15
2.4.2 Network Stack . . . 16
2.4.3 Advantages of FPGAs . . . 16

3 Building Low Latency Order Books . . . 18
3.1 Market Feed . . . 18
3.2 The National Stock Exchange of India . . . 19
3.3 Architectural Motivation . . . 20
3.4 Architecture . . . 23
3.4.1 Ethernet, MAC and UDP/IP . . . 24
3.4.2 Line Arbitrator and Feed Arbitrator . . . 24
3.4.3 Order Book . . . 24
3.5 Hardware Data Structures in the order book . . . 25
3.5.1 orderBook . . . 26
3.5.2 topCache and orderBookMask . . . 26

3.5.3 allOrders . . . 27
3.6 Update Algorithms . . . 27
3.6.1 operationNew . . . 28
3.6.2 operationDelete . . . 28
3.6.3 operationModify . . . 29
3.6.4 operationTrade . . . 29
3.7 Hardware Friendly Order Book Update . . . 30

4 Experimental Setup & Results . . . 33
4.1 Experimental Setup . . . 33
4.2 Results . . . 34
4.2.1 Area Utilization . . . 34
4.2.2 Latency . . . 35

5 Conclusion and Future Work . . . 37

Bibliography . . . 39

List of Figures

Figure		Page
1.1	Network Packet Processing in General Purpose CPUs	2
2.1	An overview of the underlying structure of Programmable Logic (PL).	7
2.2	An FPGA Configurable Logic Block (CLB).	8
2.3	An FPGA Dual Port BRAM.	9
2.4	An overview of the trading infrastructure in the co-location centre	15
3.1	The figures in this analysis depict the hit ratio with respect to the chunk index based on the historical data of a ticker.	22
3.2	A high level overview of the system architecture	23
3.3	Hybrid Binary-Linear Search	31
4.1	Experimental Setup	33

List of Tables

Table Page

2.1 Spatial-Temporal Distribution of Computation Implementations. 6
2.2 A sample order book. 13
2.3 Order Book before trade (left), order Book after trade (right). 13
2.4 Comparison of Features in HFT and AT 14

3.1 Structure of an order message sent by NSE. 19
3.2 Structure of a trade message sent by NSE. 19
3.3 Extended Top Cache 27
3.4 Sample Orderbook for Ask 32

4.1 FPGA Resource Utilization for 2 tickers. 35
4.2 FPGA Latency results. 35

Abbreviations

AT	Algorithmic Trading
BRAM	Block Random Access Memory
CLB	Configurable Logic Blocks
CPU	Central Processing Unit
DSP	Digital Signal Processing
FF	Flip Flops
FPGA	Field Programmable Gate Arrays
GPU	Graphics Processing Unit
HBM	High Bandwidth Memory
HFT	High Frequency Trading
HLS	High Level Synthesis
NIC	Network Interface Card
NSE	National Stock Exchange
OS	Operating System
PCAP	Packet Capture
SFP	Small Form-factor Pluggable
SRAM	Static Random Access Memory
UDP	User Datagram Protocol
URAM	Ultra Random Access Memory

Chapter 1

Introduction

The stock market is an avenue where investors trade in shares, bonds, and derivatives. This trading is facilitated by stock exchanges. The term stock exchange refers to a centralised location where shares of a publicly traded company are bought and sold. There are numerous stock exchanges around the world. Some of the largest exchanges are the New York Stock Exchange (NYSE), the NASDAQ, and the Tokyo Stock Exchange (JPX). Other well-known stock exchanges include the London Stock Exchange Ltd (LSE), the National Stock Exchange of India (NSE) and the Bombay Stock Exchange (BSE).

1.1 Motivation

The purchase and sale of stocks listed on the exchanges are facilitated by stockbrokers/brokerage firms who act as middlemen, placing orders on behalf of the customer. Hence, stock trading means buying and selling shares in companies to try to make money on price changes. In contrast to past methods of over-the-counter stock trading, trading has become extremely convenient over the Internet today as one can trade over a large range of devices, from personal computers to handheld smartphones. This process, also known as electronic trading [1], opens up the avenue for Algorithmic Trading (AT) [2], which refers to the use of computer programs to automate one or more stages of the trading process. These stages include pre-trade analysis, trade recommendations, and order execution. Each of these stages can be conducted by a human, fully by an algorithm or by humans and algorithms.

To enable algorithmic trading, traders need to be updated on the latest activity of a stock. This information is provided by the order book, which is an electronic list of buy and sell orders for a security in an exchange, organized by price. The order book is constantly updated as new orders are placed and cancelled. By analyzing the order book, traders can get a real-time view of the supply and demand for a security. This information can be used to make trading decisions, such as when to place an order or how much to pay for a security.

In this era of ever-growing computational power, stock markets have witnessed the rise of High Frequency Trading (HFT). HFT is a subset of all algorithmic trading and is characterized by the use of extraordinarily high-speed and sophisticated computer programs for generating, routing, and executing

orders [3]. In traditional trading with human interaction on trading floors, traders could gain an edge by being faster and louder. With algorithms negotiating on prices nowadays, those physical advantages are no longer needed. Nevertheless, in markets trading at high speed, the capability to receive data and submit orders at the lowest latency is essential [4]. To enable this, HFTs are also characterized by the use of co-location services and individual data feeds offered by exchanges and others to minimize network and other types of latencies [3]. Instead of sending the complete orderbook to the brokers, the exchanges regulate its reconstruction by sending orders as User Datagram Protocol (UDP) packets. This saves time, as the complete order book does not need to be transmitted to the broker.

1.2 What are FPGAs, and where do they step in?

Field Programmable Gate Arrays (FPGA) are semiconductor components structured using a matrix of Configurable Logic Blocks (CLB) interconnected via programmable pathways. These fall under the category of programmable logic devices, also known as programmable hardware. FPGAs offer the capability to be reprogrammed according to specific application or functional needs even after the manufacturing process. This sets them apart from Application Specific Integrated Circuits (ASIC), which are tailored for specific design purposes through custom manufacturing. FPGAs rely on Static Random Access Memory (SRAM)-based configurations, allowing for virtually limitless reconfigurations.

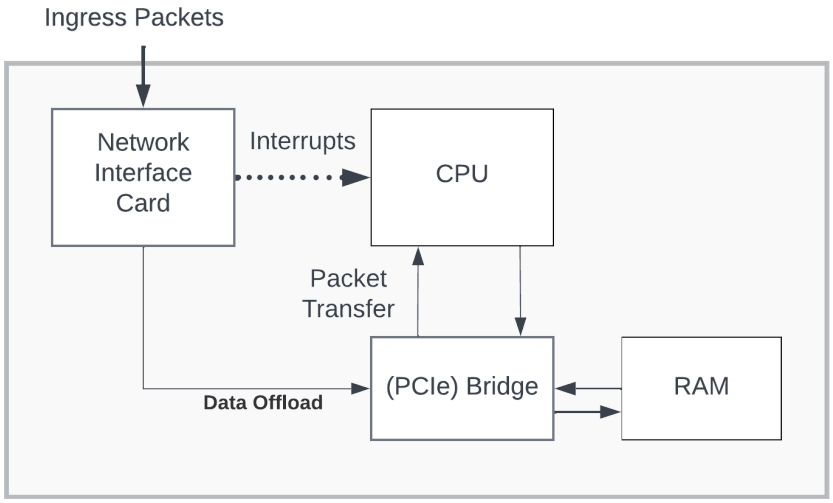


Figure 1.1: Network Packet Processing in General Purpose CPUs

For a general-purpose Central Processing Unit (CPU), a network packet arrives at the Network Interface Card (NIC), where it is stored in the NIC's memory. The NIC then sends the packet to the Operating System (OS) kernel, which processes it through the different layers of the network stack. Once the packet has been processed, it is moved from kernel space to user space memory, where the application can access it. However, this introduces latency due to the two memory copies required. In

addition, there is computational delay associated with packet processing, as the OS kernel must parse the packet header and perform other tasks. This routine is depicted in Figure 1.1.

FPGAs offer a different approach, where the received input network packet can be parsed in hardware with low latency in just a few clock cycles. This allows the parsed packet to be directly passed to the order book processing module(kernel), eliminating the need for additional memory copies. This advantage holds true even for outgoing orders, making FPGA-based systems superior to CPU-based systems, particularly in High-Frequency Trading.

1.3 Summary of Contributions

The main contributions of this thesis are presented in Chapter 3 and are explained briefly here:

1. This work presents a low-latency hardware architecture designed to manage the order book on FPGAs. Our work is based on the Indian National Stock Exchange (NSE).

2. We propose the use of individual data structures to capture the market depth of a stock for quick update and access while maintaining the complete order book separately. We also propose the use of High Bandwidth Memory (HBM) to store and maintain a mapping of the order IDs to price and quantity, to facilitate modification, cancellations and trades.

3. We test out our experimental setup in a simulated but close to real-time market conditions using recorded Packet Capture (PCAP)s for replay, at a high frequency of messages through a 10 Gigabit Fiber Cable.

4. We are able to surpass the current implementations marginally, simultaneously allowing more complex operations and providing the latest information after each order, in addition to developing and testing the application for a new exchange, NSE.

1.4 Thesis Organization

This thesis is organized into the following chapters:

- Chapter 1 presents an introduction to the problem statement and a summary of contributions. Further, a literature review of the pre-existing work is covered here.

- Chapter 2 discusses reconfigurable computing, including FPGA architecture, components, and applications. The chapter also explores market mechanics essential for understanding the order book, Algorithmic Trading (AT), and High-Frequency Trading (HFT). Finally, it establishes the groundwork for the use of FPGAs in HFT.

- Chapter 3 details the hardware architecture of the proposed solution, including the problem statement, hardware data structures, and kernel functions.
- Chapter 4 presents the hardware evaluation of the proposed architecture. This chapter presents the performance metrics of our architecture, such as latency and hardware resource usage.
- Chapter 5 concludes this thesis with a summary of contributions and the scope of future work.

1.5 Literature Review

In the past ten years, there has been considerable effort in both academic research and industry towards optimizing network packet processing for low latency [5, 6]. Given the time-critical nature of high-frequency trading, extensive research has been conducted to enhance the efficiency of each component implemented on FPGAs. This includes optimization efforts for network packet parsing and financial protocol parsing, as well as the architecture of the core systems responsible for order book regeneration and the order execution engine.

Leber et al. [5] delve into the application of FPGA technology for the acceleration of High-Frequency Trading (HFT), presenting a system that significantly reduces latency in market data processing. They developed a microcode engine for flexible protocol support and demonstrated a fourfold reduction in latency compared to software-based approaches. Their work specifically addresses the implementation challenges and solutions associated with integrating hardware decoding of UDP and the FAST protocol, thereby not only demonstrating a substantial decrease in processing time compared to software solutions but also laying the groundwork for future advancements in electronic trading technologies by leveraging the unique capabilities of FPGAs.

Lockwood et al. [7] presents a comprehensive exploration of FPGA-based acceleration for High-Frequency Trading (HFT), highlighting the significant latency reduction achievable compared to traditional software solutions. The paper introduces an FPGA IP library designed to facilitate the rapid development and verification of financial applications, demonstrating a tangible example where FPGA deployment sustains a 10Gb/s Ethernet line rate with a mere 1µs end-to-end latency. This represents a substantial improvement, up to two orders of magnitude lower than software implementations, underscoring the critical advantage of FPGA technology in the competitive landscape of HFT.

In their work, Boutros et al.(2017) [8] delve into the optimization of FPGA-based HFT systems using High-Level Synthesis (HLS), with a special focus on innovative data structure advancements. They introduce a top-down heap-based approach for managing market data, specifically tailored to enhance order book updates' efficiency and accuracy. This strategy significantly bolsters the system's capacity to process and respond to market changes swiftly, providing a competitive edge in the fast-paced realm of HFT. Their research streamlines the development process by enabling the implementation of complex trading algorithms in High Level Synthesis (HLS) C++ and opens new avenues for trading strategy experimentation and deployment.

In their work, He et al.(2017) [9] explore the utilization of reconfigurable platforms, specifically Field Programmable Gate Arrays (FPGAs), to enhance the efficiency of Order Book Update (OBU) algorithms essential in financial exchanges. The paper introduces a novel FPGA-based solution that significantly outperforms CPU-based systems in processing speed and latency reduction. The authors' contributions include the development of a fixed tick data structure for improved hardware mapping, a customized cache for the top five levels of the order book to reduce latency, and a hardware-friendly OBU algorithm. Their experimental results demonstrate a processing capability of millions of messages per second at high throughput with low latency.

In this work, our main focus is improving order book construction and maintenance speed and accuracy while adapting Xilinx's packet processing and parsing modules for NSE. Recent and noteworthy attempts at order book construction include the works of He et al. [9] and Boutrous et al. [10]. In their work, He et al. [9] proposed a hardware-friendly clipped tree structure to maintain the order book in BRAM, with add, delete, and trade routines having a time complexity of $\log n$, where n represents the number of price levels. In our proposed approach, the add operation always takes constant time ($O(1)$). The delete and trade operations can also be performed in constant time if they do not impact the entries in the order book cache that maintains bid and ask offers related to market depth. If such entries are affected, we use a hybrid algorithm combining linear and binary search to promptly update the order book cache. In Section 3.3, we provide data analysis that demonstrates why our approach leads to consistently low latency. Furthermore, as we need to support modify orders in NSE, we utilize a simple array data structure in HBM memory to maintain the global order book table, which differs from He et al.'s approach.

Boutrous et al. [10] presented a heap-based data structure in their work on maintaining the order book. The extraction routines of the min-heap (or max-heap) have a time complexity of $O(\log n)$, while trade and delete operations may take linear time. To address this issue, they propose storing the delete orders in a separate heap structure and applying them lazily when the top of the order book heap matches the delete heap structure. However, a heap only provides information on the maximum or minimum value, and to obtain information on the top of the sorted order book that corresponds to market depth, multiple heap calls need to be made repeatedly. This not only substantially increases latency but also leads to race conditions resulting from concurrent accesses to the data structure.

In contrast to previous approaches, we employ a straightforward array data structure to represent the order book, where each entry corresponds to a price tick. Some entries may be valid while others are not, and we indicate their status using a separate bit array. Rather than using chaining to locate the next valid entry, we conduct a linear search of order book segments, with binary search being used on the bitmask for searching within each segment. Our empirical observations demonstrate that since most of the activity occurs at the top of the order book, a valid entry can usually be found within the top 2 to 3 segments. We provide supporting evidence for this claim through data analysis in Section 3.3. The simplicity of our approach renders it practical, scalable, and robust against subtle bugs that may arise due to the high volume and velocity of incoming transactions leading to concurrent accesses.

Chapter 2

FPGAs and High Frequency Trading

2.1 Reconfigurable Computing and FPGAs

Reconfigurable computing [11, 12] has gained significance as a novel framework for executing computations. It fuses the after-production programmability of processors with the spatial computational approach predominantly utilized in hardware designs. This outcome alters the established distinctions between "hardware" and "software", offering the potential for enhanced computational power and density within a programmable medium.

Traditionally, computations were implemented using distinct approaches: hardware (e.g., custom VLSI, ASICs, gate-arrays) and software executed on processors (e.g., DSPs, microcontrollers, embedded or general-purpose microprocessors). However, Field-Programmable Gate Arrays (FPGAs) have introduced an alternative that amalgamates traditional hardware and software features. Systems based on FPGAs have demonstrated remarkable performance, often surpassing processor-based alternatives by a factor of 100 and providing 10 to 100 times better performance per unit of silicon area [11].

	When is Computation defined?		
Computations Distributed Across?		Pre-Fabrication	Post-Fabrication
	Space	ASIC/Gate-Array	Reconfigurable
	Time	-	↓ Processors

Table 2.1: Spatial-Temporal Distribution of Computation Implementations.

The adoption of FPGAs in computing has paved the way for a broader category of computer architectures known as reconfigurable computing architectures. These architectures are distinguished by two principal features: the ability to be customized for specific tasks after manufacturing and the extensive utilization of spatially optimized computation in their operations. The primary advantage of FPGAs,

and reconfigurable devices in a broader sense, lies in their capacity to usher in a category of devices that can be configured after fabrication. These devices facilitate spatial computations, marking a novel organizational milestone in this domain. [11]

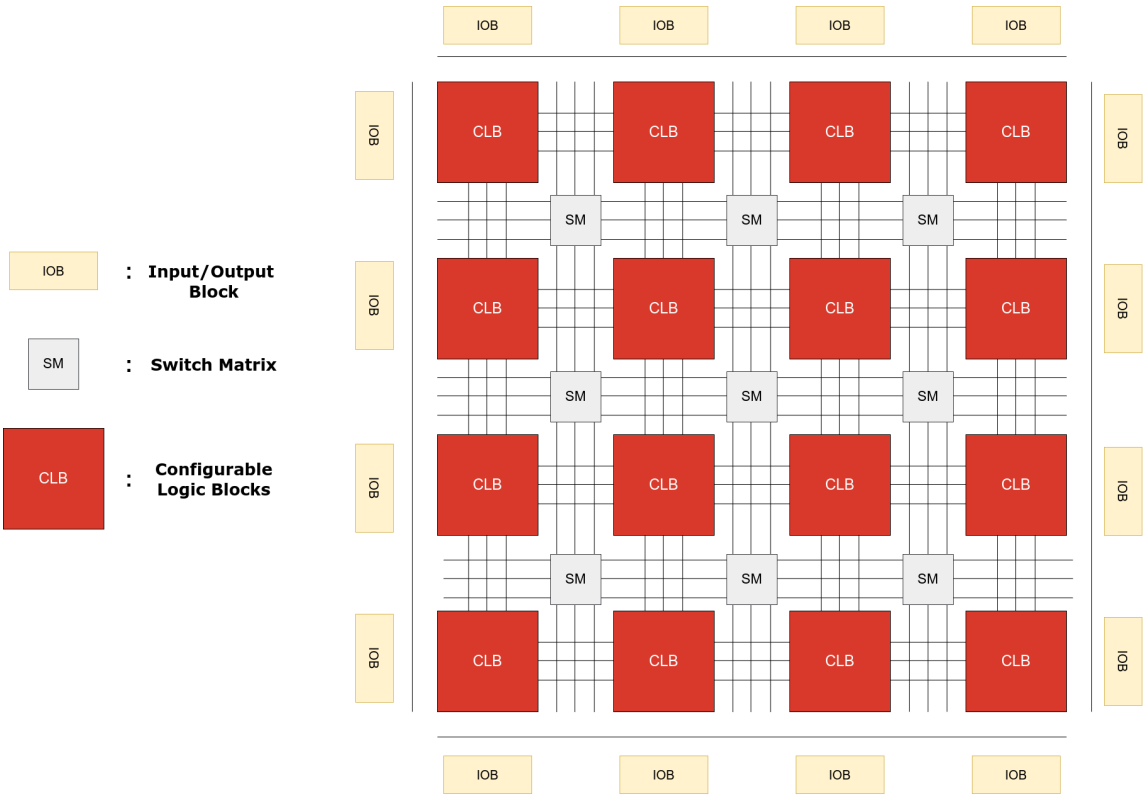


Figure 2.1: An overview of the underlying structure of Programmable Logic (PL).

2.1.1 Field-Programmable Gate Arrays

FPGAs, or Field-Programmable Gate Arrays, are custom computing platforms designed for dedicated hardware architectures characterized by their power efficiency, parallel processing capabilities, and reprogrammability. FPGAs are highly versatile and complex programmable logic devices. In terms of power efficiency, they outperform Graphics Processing Unit (GPU)s, and in speed, they surpass CPUs, giving them a competitive edge. Their reprogrammability is a distinct advantage over ASICs, enabling quicker deployment and allowing for post-deployment updates to accommodate algorithmic and hardware architecture improvements. FPGAs are equipped with programmable logic elements and interconnectivity, devoid of a Program Counter for instruction fetching. Figure 2.1 depicts the programmable logic region of an FPGA.

Originally conceived as a hybrid between Programmable Array Logic (PALs) and Mask Programmable Gate Arrays (MPGAs), FPGAs offer full electrical programmability, leading to cost-effective customiza-

tion for multiple application circuits and facilitating the implementation of intricate computations on a single chip. An FPGA consists of the following components:

1. **CLB**: A CLB is a fundamental building block within a FPGA. FPGAs typically contain multiple CLBs, which are interconnected through routing resources. CLBs serve the purpose of implementing both combinational and sequential logic in FPGA designs. They are highly versatile and allow for customization to perform a wide range of tasks. CLBs consist of three essential components:

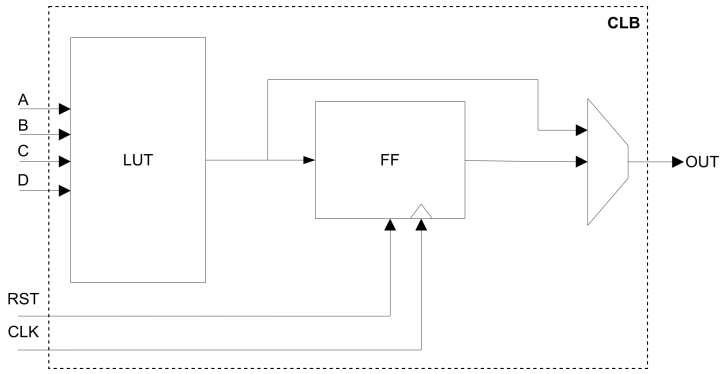


Figure 2.2: An FPGA Configurable Logic Block (CLB).

   (a) **Flip-Flops**: Flip Flops (FF) are single-bit storage elements available within a CLB. They are used to store and retain binary data between clock cycles, which is essential for sequential logic operations in FPGA circuits.

   (b) **Look-Up Tables (LUTs)**: LUTs are a critical component of CLBs that store precomputed outputs for all possible input combinations, making them a crucial resource for implementing logic operations. Each LUT comprises SRAM cells connected to a multiplexer, with the multiplexer's select lines representing the LUT's inputs. During FPGA configuration, the SRAM cells are programmed with the truth table's output, called the LUT-Mask. This versatile resource allows LUTs to function as Read-Only Memory (ROM), Random Access Memory (RAM), logic functions with N inputs, and even as shift registers, enabling FPGAs to efficiently execute a wide range of digital logic operations while offering rapid, precomputed output retrieval.

   (c) **Multiplexers**: Multiplexers are circuits within CLBs that are responsible for selecting one of multiple inputs and routing it to the output based on the signals received from select lines. They play a key role in managing signal routing within the CLB and the broader FPGA.

2. **Programmable Interconnects**: Programmable interconnects are integral components in Field-Programmable Gate Arrays (FPGAs), comprising an extensive network of wires, switches, and multiplexers. They form the fundamental framework for establishing customized digital circuits,

enabling the interconnection of configurable logic elements to facilitate signal routing and data flow. Within this programmable interconnect framework, a subset known as the switch matrix consists of essential routing components responsible for governing signal paths and allowing users to configure connections and manipulate logic routes. Together, both of them combine to create a flexible hardware infrastructure in FPGAs, enable specific design requirements, and support a wide array of applications.

3. **I/O Blocks**: Input/Output Blocks (IOBs) in FPGAs are dedicated components that enable the interfacing between the FPGA's programmable logic and external circuits. These IOBs are crucial in handling various I/O standards, ensuring signal integrity, and managing clock distribution within the FPGA. Located at the periphery of the device, IOBs efficiently manage bidirectional data transfer, voltage-level matching, and timing requirements.

4. **Block Random Access Memory (BRAM)**: BRAM is a fundamental component of Field-Programmable Gate Arrays (FPGAs), serving as on-chip memory modules used for data storage and manipulation. BRAMs are exceptionally useful in FPGA designs due to their ability to efficiently transfer data between various clock domains, enabling DMA-based communication with external processors, supporting peer-to-peer data exchange among FPGA targets, and storing large datasets far more efficiently than equivalent memory constructed from lookup tables.

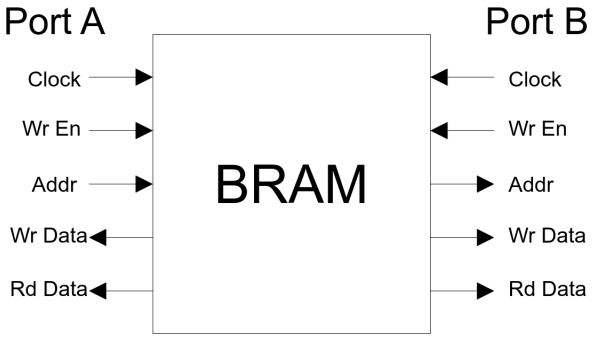


Figure 2.3: An FPGA Dual Port BRAM.

In Xilinx FPGAs, BRAMs come in 18k or 36k-bit variants, and their dual-port nature allows for simultaneous, parallel access to different memory locations, making them invaluable for diverse data management and memory access tasks within FPGA-based applications. Additionally, BRAM can be configured to operate as either RAM or ROM, allowing for flexible data manipulation or read-only functionality as required by the FPGA application.

5. **Ultra Random Access Memory (URAM)**: UltraRAM, exclusive to select Xilinx FPGA families such as Virtex UltraScale+ and Versal, serves as a dedicated and high-capacity memory resource designed to address demanding on-chip memory needs. With its notable advantages, including an expansive memory capacity and high bandwidth that enable efficient storage and rapid data processing, UltraRAM empowers FPGA designers to tackle challenging tasks.

6. **DSP Slices**: Digital Signal Processing (DSP) slices are specialized hardware blocks optimized for high-speed digital signal processing tasks such as filtering, FFT, and mathematical operations. These dedicated components, comprising efficient multipliers, adders, accumulators, and more, accelerate tasks requiring rapid computation and precise data manipulation within the FPGA architecture. DSP slices are a key component in FPGA cards like the Xilinx Virtex UltraScale+ and Intel Arria 10, enhancing their capabilities in applications demanding real-time applications.

2.2 Real-World Applications of FPGAs

FPGAs, or Field-Programmable Gate Arrays, are versatile tools used in a wide range of practical applications, including audio and video processing, cryptography, signal processing, image processing, random number generation, and algorithm implementation.

2.2.1 In Cloud and Data centres

FPGAs are increasingly integral to the datacenter strategies of major cloud providers due to their potential for higher performance and energy efficiency. This potential arises from the ability to use custom datatypes, optimize memory hierarchies and connectivity, eliminate instructions, employ very deep pipelining, harness abundant parallelism, and operate at lower clock frequencies [13].

Big data analytics, driven by the ever-increasing volume and velocity of data, has become a critical component of modern data-driven businesses. To tackle the growing demands for performance in this domain, the integration of Field-Programmable Gate Arrays (FPGAs) has emerged as a game-changing solution. FPGAs offer several unique advantages that can revolutionize big data processing in cloud environments. FPGAs are also instrumental in cloud platforms like Amazon's Elastic Compute Cloud (EC2) F1 instances [14, 15], which power artificial intelligence and big data algorithms.

One of the key strengths of FPGAs in cloud environments is their ability to specialize FPGA accelerators for specific applications without the need for hardware replacement. This adaptability allows users to tailor the logic mapped onto the FPGA fabric, effectively creating custom dataflow computers. In doing so, there is a reduced requirement to shuttle data back and forth through the memory hierarchy, significantly enhancing overall system efficiency. Even in cases where FPGA-accelerated designs offer only marginal improvements in terms of throughput, they tend to be highly power-efficient due to reduced data movement and the ability to operate at relatively low clock frequencies [13].

For efficient FPGA deployment, programmable overlays strike a balance between performance and compilation time. These predefined FPGA designs contain compute elements optimized for specific applications. Notable examples include the Catapult project [16] and machine learning accelerator overlays, like the Deep Learning Processing Unit (DPU) in Xilinx's Vitis toolset [17].

2.2.2 In the Internet of Things

Over recent years, FPGA-based System-on-Chip (SoC) design [14, 18], leading to the creation of Field-Programmable SoCs (FPSoCs), has seen remarkable growth. Vendors have crafted dedicated FPSoC architectures to enhance the seamless integration of soft or embedded processors with hardware peripherals and distributed logic. The combination of a standalone processor and FPGA architecture on a single silicon chip, has greatly boosted research in FPGA applications. This integration significantly cuts production costs and circuit board space. Moreover, the absence of a bus for communication between the processor and FPGA not only slashes power consumption but also minimizes communication delay, ultimately boosting bandwidth.

As a supporting processing edge device, FPGAs have been integrated with Raspberry Pi [19] to execute image processing tasks. The Raspberry Pi triggers certain GPIO pins that communicate with the FPGA for processing. Once image processing is complete, the FPGA outputs the result to a computer monitor via VGA. Users access the system through the Raspberry Pi's IP address, turning it into an IoT application. This separation of FPGA processing from IoT devices and web services optimizes performance and enables various IoT devices to interface with the FPGA using communication standards like USB.

2.2.3 Other Areas

In the realm of genetics research, as the volume of DNA data expands exponentially, the adoption of distributed and parallel computing methodologies has become indispensable. These techniques involve breaking down complex computational tasks into smaller, more manageable parts, and distributing them across multiple machines or processors, which not only accelerates data analysis but also optimizes resource utilization.

The integration of FPGAs in bioinformatics showcases their remarkable potential in accelerating critical genetic analysis algorithms. One piece of research focuses on addressing the challenge of resource-intensive DNA sequence analysis on mobile platforms by designing and implementing a low-power real-time FPGA hardware accelerator for basecalling in nanopore-based DNA measurements, achieving over 100 times faster processing speed and a significant energy efficiency improvement compared to CPU-only basecalling [20]. In another research endeavour, an FPGA-based approach is explored to accelerate the pair-HMMs forward algorithm for DNA sequence analysis. The implementation on the Convey supercomputing platform demonstrates a remarkable speed improvement of up to 67 times compared to software-only execution, underscoring the effectiveness of FPGA-based computation in this context [21]. In another study, researchers delve into the field of bioinformatics to address the computationally intensive task of short-read alignment. Their innovative approach introduces FPGA-based acceleration to significantly reduce the execution time of aligning millions of short DNA reads to a reference genome, offering a promising solution to this longstanding challenge [22].

In space technology, the James Webb Space Telescope (JWST) leverages FPGAs [23] for flexible sensor interfaces and dynamic image processing, enhancing celestial observation. With its multitude of micro-shuttles, the Near Infrared Spectrograph in the JWST enables detailed study of distant galaxies.

2.3 Market Mechanics

To completely understand HFT, getting acquainted with certain terminology regarding the stock market is important. The *bid price* in a stock market represents the purchase price for buyers, while the *ask price* signifies the sale price for sellers. Orders are the cornerstone of stock trading. They are explicit instructions given by traders, indicating their intention to buy or sell a particular stock. These instructions encompass details, including the stock's identity, the quantity of shares involved, and the desired price at which the trade should occur. Market orders represent the simplest form of trading instruction. They dictate an immediate purchase or sale of a stock at the best-prevailing market price. Limit Orders, in contrast, enable traders to stipulate a specific price at which they are willing to buy or sell a stock. For buyers, it specifies the highest price they are willing to pay; for sellers, it signifies the minimum acceptable price.

Bid orders come into play when traders intend to purchase a stock. By placing a bid order, they express the highest price they are willing to pay for the shares. Ask orders, conversely, are relevant when traders wish to sell a stock. Through an ask order, they state the lowest price at which they are willing to part with their shares. The market spread signifies the gap between the highest bid price and the lowest ask price in the market. A narrower spread typically indicates higher market liquidity, while a wider spread may indicate lower liquidity. A trade occurs when a buyer's bid matches a seller's ask. Consequently, the ownership of shares is transferred from the seller (the ask) to the buyer (the bid).

2.3.1 Order Book

An order book is a list of all the orders that are used to describe all buy and sell orders for a specific security or financial instrument. Fundamentally, it is represented as an electronic list of buy and sell orders for any stock/security in an exchange, organized by price. Table 2.2 represents a sample order book for a stock on the NSE. The left half represents the *bid* orders, and the right half represents the *ask* orders, both accumulated by price. Bid orders are sorted in descending order by price, and ask orders are also sorted in descending order by price. The order book is structured this way because the market spread is the smallest between the highest bid and the lowest ask prices. Consequently, trades are executed at the top of the table accordingly when the market spread equals zero.

Depending on the liquidity of the stock, the exchange can receive hundreds to thousands of orders for a stock every second. Consequently, there are rapid updates to the order book as a result of entry of new orders as well as modification and cancellation of existing orders. Hence, it is less than ideal for an exchange to send the complete order book to the registered brokers due to network latency con-

Order Book Snapshot					
Bid	Orders	Qty	Ask	Orders	Qty
210.45	1	100	210.55	2	188
210.40	1	4	210.60	1	5
210.35	2	301	210.65	1	100
210.30	3	74	210.75	1	5
210.20	11	2871	210.80	1	5

Table 2.2: A sample order book.

straints. Rather than transmitting the entire order book, exchanges usually dispatch order information in a chronological stream of network packets, ensuring swift data transmission. The orders are transmitted via UDP packets following packet formats prescribed by each exchange. The UDP packets are then parsed at the broker systems, which allows them to re-construct a copy of the order book present at the exchange.

2.3.2 Algorithmic trading

Algorithmic Trading (AT) [2, 24] refers to any form of trading using sophisticated algorithms (programmed systems) to automate all or some part of the trade cycle. AT usually involves learning, dynamic planning, reasoning, and decision-making. The key stages in AT are the pre-trade analysis, signal generation, trade execution, post-trade analysis, risk management, and asset allocation. Pre-trade analysis involves evaluating asset features to identify potential trades by analyzing market data or financial news. This analysis is used to generate recommendations and subsequently signal when and what to trade.

Order Book XYZ Inc					
Bid	Orders	Qty	Ask	Orders	Qty
210.55	1	100	210.55	2	188
210.40	1	4	210.60	1	5
210.35	2	301	210.65	1	100
210.30	3	74	210.70	1	5
210.20	11	2871	210.80	1	5

Order Book XYZ Inc					
Bid	Orders	Qty	Ask	Orders	Qty
210.40	1	4	210.55	1	88
210.35	2	301	210.60	1	5
210.30	3	74	210.65	1	100
210.20	11	2871	210.70	1	5
210.15	5	21	210.80	1	5

Table 2.3: Order Book before trade (left), order Book after trade (right).

Pre-trade analysis mainly comprises three main categories of techniques. *Fundamental Analysis* involves a detailed examination of various factors that could influence an asset's price in order to determine its fair value or potential future price movements. *Technical analysis* focuses on predicting future price movements by analyzing historical price data and related trading information, operating on the assumption that market prices already incorporate all relevant information and aiming to identify and exploit price patterns. *Quantitative analysis* treats asset prices as random and employs mathematical and

Common in HFT and AT	Specific for AT excl. HFT	Specific for HFT
1.Automated order submission 2.Automated order management 3.Pre-architected trading decisions	1.Agent trading 2.Longer holding periods 3.Goals to achieve particular benchmarks	1.Very High Number of Orders 2.Very low margins per trade 3.Extremely rapid order placing and cancellation 4.Focus on high liquid instruments 5.Very short holding periods 6.Use of co-location services

Table 2.4: Comparison of Features in HFT and AT

statistical methods to develop models that describe this randomness, playing a crucial role in portfolio theory, derivatives pricing, and risk management within the financial industry.

During real-time trade signal generation, technical analysis plays a significant role as the state of the market changes continuously. This change is reflected in the order book, which reflects the changes in the trading activity for a stock. Hence, rapid re-construction of the order book is vital to swift trade signal generation. It is also worth noting that multiple AT systems can react to the same state of the order book. Therefore, faster AT systems pose a significant advantage, as they can send in new orders which can alter the order book before the other AT systems place their orders. Hence, orderbook reconstruction becomes a time-critical process for AT systems.

2.3.3 High Frequency Trading

High Frequency Trading (HFT) [3,4] is a newer phenomenon in the AT landscape, which shares a lot of common properties with it but is also characterized by distinct features of its own. The U.S. Securities and Exchange Commission, in its 2010 Concept Release on Equity Market Structure, identifies High Frequency Traders as professional traders who operate on their own behalf and participate in strategies that result in a high volume of trades each day. According to the SEC, several key features distinguish HFTs, including the employment of advanced, high-speed computer algorithms for order generation, routing, and execution; leveraging co-location services and proprietary data feeds to reduce various latencies; extremely brief durations for opening and closing positions; a tendency to submit a large number of orders that are quickly canceled; and striving to end the trading day with minimal, if any, open positions, thereby avoiding significant overnight risks.

2.3.4 Importance of Low Latency in HFT

HFT traders identify and benefit from the transient inefficiencies of the market. This not only enables them to make profits but makes the market more efficient. Due to the extremely competitive nature of HFTs, only the fastest HFT traders can capitalize on an attractive order and make profitable trades when it pops up on the limit order book. The need for such a competitive edge motivates HFT firms to invest in cutting-edge hardware and fine-tune their algorithms to reduce latency. The essence of success in

high-frequency trading lies in being lightning-fast. Since light and electrical signals have a finite speed, HFT firms strategically position their computers as close to trading venue servers as possible.

Several exchanges offer HFT firms the option to rent space in their data centres, known as co-location [3] services, allowing them to position their hardware closest to the trading servers. While this practice raises concerns about fair market competition and equal market access, thankfully, the regulatory framework governing these services is meticulously designed to address and minimize these issues, ensuring a level playing field for all market participants.

2.4 Role of FPGAs in High Frequency Trading

To understand the benefits of using an FPGA for HFT, it is essential to understand the different components of the trading infrastructure with respect to HFT.

2.4.1 Trading Infrastructure

Figure 2.4 provides an overview of the trading infrastructure in an exchange where multiple HFT firms place their proprietary trading servers through allotted access switches.

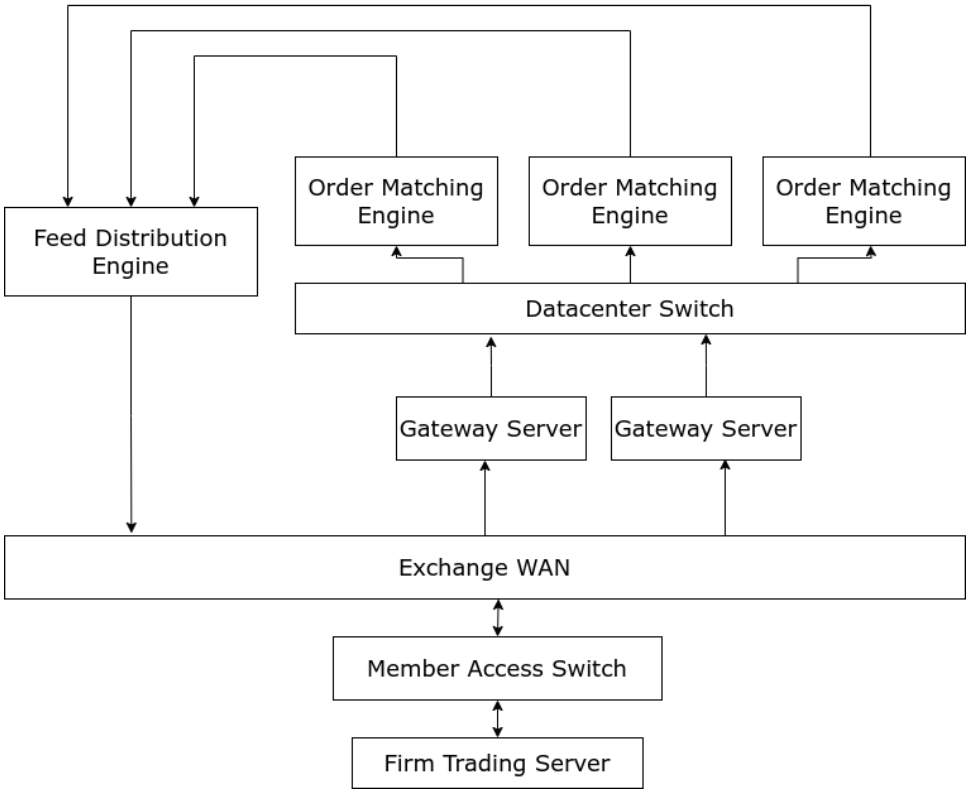


Figure 2.4: An overview of the trading infrastructure in the co-location centre

Within this system, the order-matching engine initiates trading opportunities that drive the trading process. These opportunities are relayed to the feed distribution engine. Subsequently, the feed distribution engine disseminates this update to all the clients. These clients, functioning as firm trading servers, assess the opportunity and promptly respond with an order. The order is subsequently relayed to the order matching engine through the gateway, which in turn proceeds to match the initial arriving order with the created opportunity, ultimately resulting in the execution of a trade. This intricate sequence forms the basis of how information flows and how orders are efficiently processed within this system.

2.4.2 Network Stack

Today, many high-frequency trading (HFT) firms still use software-based (CPU) servers located in data centres. While these systems are easier to manage and allow for quicker updates to trading algorithms compared to FPGA-based HFT systems, they suffer from higher latency, mainly due to network card and operating system routines.

In contrast to FPGA-based trading platforms, software-based platforms involve the reception of network traffic by a network adapter. Relevant data is then moved to memory, and the CPU is interrupted to handle the application processing. Once the processing is done, the data is sent back to the network adapter and transmitted over the network. This interruption-driven software setup, along with unpredictable PCIe transfers and cache misses, results in higher and less predictable network latencies in software implementations.

2.4.3 Advantages of FPGAs

Conversely, FPGA-based implementations facilitate the direct routing of incoming data through a tailored, application-specific processing pathway, utilizing the Ethernet Physical Layer (PHY) and the Media Access Controller (MAC). Notably, important data within a packet can be extracted before receiving the entire packet. This means the data is instantly accessible, right within the same clock cycle as when it arrives, which is quite different from how software-based systems work (where the whole packet typically waits in the network stack before it can be processed).

Importantly, there's no variability in how quickly we can extract and process incoming data. We can accurately predict when any piece of data will be available down to a clock cycle. As a result, FPGAs naturally achieve significantly lower latencies with very little variation. Furthermore, in modern FPGA setups, incoming market data can be processed entirely within the FPGA itself, without the need to send it back to the host CPU.

When an application runs on an FPGA, it functions differently from how compiled instructions work on CPUs and GPUs. Instead, the FPGA guides data through custom-designed pipelines that align with the software's intended operations. This close alignment between the dataflow pipeline hardware and the software eliminates control-related overhead, leading to enhanced performance and efficiency. In the case of CPUs, instruction stages can be processed in a pipelined fashion, with new instructions

commencing execution every clock cycle. In contrast, FPGAs employ pipelined operations, allowing different streams of instructions operating on different data to begin execution in each clock cycle. Therefore, multiple pipelines can be instantiated in parallel when data dependencies do not exist.

In contrast to ASICs(Application-Specific Integrated Circuit), FPGAs offer the performance benefits of custom hardware while retaining programmability. This is particularly significant in the context of today's rapidly evolving financial markets, where new regulations and trading algorithms continually emerge. ASICs become infeasible in such environments since they lack the flexibility to adapt to these changes. Trading algorithms cannot be easily improved or circuit issues fixed in the field with ASICs, making FPGAs a more adaptable choice for modern trading systems.

Chapter 3

Building Low Latency Order Books

Building on the previous chapter's exposition of the order book as an electronic list of buy and sell orders for securities, this chapter introduces the concept of market feed, specifically within the National Stock Exchange of India, to address the problem statement outlined. We then proceed to detail the main contributions of this paper, including the architectural motivations behind high-frequency trading (HFT) systems, an in-depth examination of the HFT system's architecture, and a nuanced discussion on the design of the order book. The chapter concludes with a comprehensive explanation of our Hybrid Binary-Linear search algorithm, illustrated with an example for clarity.

3.1 Market Feed

In today's high-frequency trading (HFT) landscape, millions of bid and ask orders are generated, cancelled, or modified every second. The primary responsibility of any exchange is to manage information related to these orders, settle matching bid and ask orders, and disseminate trade information to all traders and brokers. One approach to achieving this is by transmitting the updated order book for each security after every trade and order. However, given the vast number of stocks and messages involved, this method proves inefficient for information dissemination in exchanges.

Instead, market participants gain access to outstanding orders through a real-time data stream, either compressed or uncompressed, known as feeds. These feeds convey pricing information for stocks and are multicast to market participants through standardized protocols, typically transmitted over UDP on Ethernet [5]. Due to the multitude of instruments, the feed is divided into multiple data streams. Consequently, traders receive numerous high-data-rate UDP/IP market data streams over the network from trading venues, forming a crucial aspect of real-time information access.

Exchanges globally use various protocols for market data distribution. FIX is widely adopted for standardized real-time data transmission, while FAST extends FIX for high-speed streaming. Simple Binary Encoding (SBE) is chosen for efficient binary message handling. Multicast UDP/IP is commonly employed for simultaneous data delivery to multiple users. These protocols collectively enable seamless and efficient connectivity for traders worldwide.

3.2 The National Stock Exchange of India

The National Stock Exchange of India Limited (NSE), headquartered in Mumbai, is a prominent stock exchange in India. It is the world's largest derivatives exchange in terms of the number of contracts traded as well as it ranks fourth in cash equities based on the number of trades conducted during the calendar year 2021. NSE facilitates electronic trading through its National Exchange for Automated Trading (NEAT) platform.

NSE uses it's own MTBT Feed, which is the Tick By Tick data feed of NSE, providing real-time information about orders and trades. The feed comprises a series of sequenced variable-length messages. NSE MTBT Feed data is transmitted to clients using a connection-less UDP Multicast. NSE streams information for cash markets in two main types of packets:

Order Message		
Field	Data Type	Value
Global Header	STREAM_HEADER	length, sequence, stream
Message Type	CHAR	'N' or 'M' or 'X'
Timestamp	LONG	Numeric
Order ID	DOUBLE	Numeric
Token	INT	Numeric
Order Type	CHAR	'B' - Buy; 'S' - Sell
Price	INT	Price of the order in paise
Quantity	INT	Numeric

Table 3.1: Structure of an order message sent by NSE.

1. *Order*: This packet contains information for an order. It can be a new ('N'), delete ('X') or modify('M'). In the order book, a new order adds a specified quantity at a specific price level, while a delete order removes a previous order, and a modify order updates the price and quantity of an existing order.

Trade Message		
Field	Data Type	Value
Global Header	STREAM_HEADER	length, sequence, stream
Message Type	CHAR	'T'
Timestamp	LONG	Numeric
Buy Order ID	DOUBLE	Numeric
Sell Order ID	DOUBLE	Numeric
Token	INT	Numeric
Trade Price	INT	Price of the order in paise
Trade Quantity	INT	Numeric

Table 3.2: Structure of a trade message sent by NSE.

2. *Trade*: A trade refers to the sale and purchase of assets and securities between two consensual sides, i.e. when the quantities at bid side meet the ask side, a trade happens. Fundamentally, trade can be understood as concurrent delete operations on the bid and ask side of the order book. However, the quantity of stock removed by a trade message can be less than the quantity of the order in contrast to a delete message which removes the quantity corresponding to a complete order at a given price.

Table 3.1 illustrates the fields of an "order" message from NSE, encompassing details like order type, price, and quantity. Table 3.2 outlines the structure of a trade message transmitted by the exchange. Both order messages are preceded by a *stream header*, incorporating message length, stream ID, and sequence number. It is important to note that the exchange doesn't transmit price and quantity for cancellation and modification orders. In such cases, the broker/trader needs to reference the latest information related to that order ID. Keeping a structured mapping of order IDs to their respective prices and quantities is a must for carrying out these operations effectively. It's crucial to keep that order ID info updated by updating it with all incoming operations associated with it. This way, we ensure that the order book is always consistent and that all future operations are based on the right and updated information.

3.3 Architectural Motivation

The bid side of the order book comprises the top bids, while the sell side contains the best offers. A critical observation from analyzing trading activities across multiple stocks is that the majority of transactions occur at the top of the order book, specifically within the top 5 to 20 price levels. This phenomenon underscores the importance of these levels in capturing the bulk of market activity, thereby suggesting a targeted focus for trading strategies and algorithms. Given this market dynamic, in our work, we propose a novel approach to order book architecture. We argue that, although maintaining a complete order book is necessary for market integrity and correctness, concentrating on the top price levels significantly enhances the efficiency and scalability of trading operations. This approach offers a substantial improvement over methods that rely on the full order book by simplifying the complexity of trading algorithms to operate within constant time and reducing the variables involved in manually derived strategies. This streamlined focus caters to most trading activity and paves the way for a low-latency hardware architecture optimized for high-frequency trading environments. We substantiate our claims through an in-depth analysis of market activity for multiple stocks, examining how trading patterns vary across different price levels. The findings, presented in Figure 3.1, reveal a pronounced concentration of trades at the top of the order book, validating our premise for a simplified yet effective architectural design.

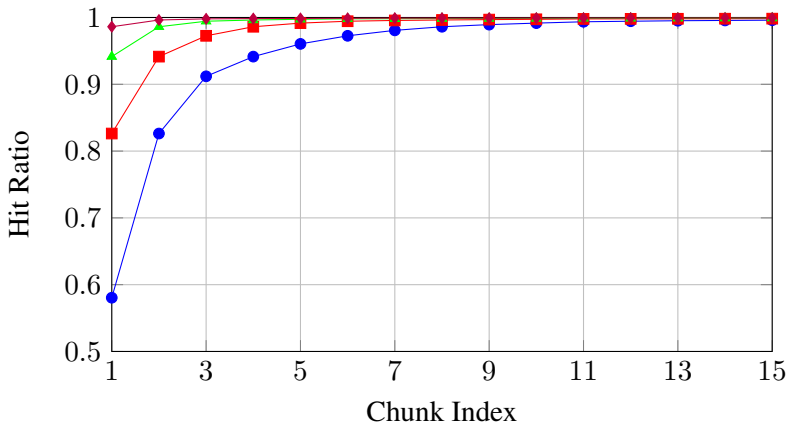
In the operation of an order book, any changes—be it through trades, deletions, or modifications—result in adjustments to the quantities listed at specific price levels. This dynamic nature of the order book necessitates a continuous update mechanism to accurately reflect the current market conditions. For instance, when a trade is executed, or an order is modified or deleted, the quantity associated with the

respective price level is accordingly altered. A unique scenario arises if the affected order is the only one at its price level; the removal or modification of this order reduces the quantity to zero, effectively eliminating that price level from the order book. In trading, any order book only shows price levels that have orders, or in other words, price levels with a non-zero quantity. We refer to these price levels as "valid'.

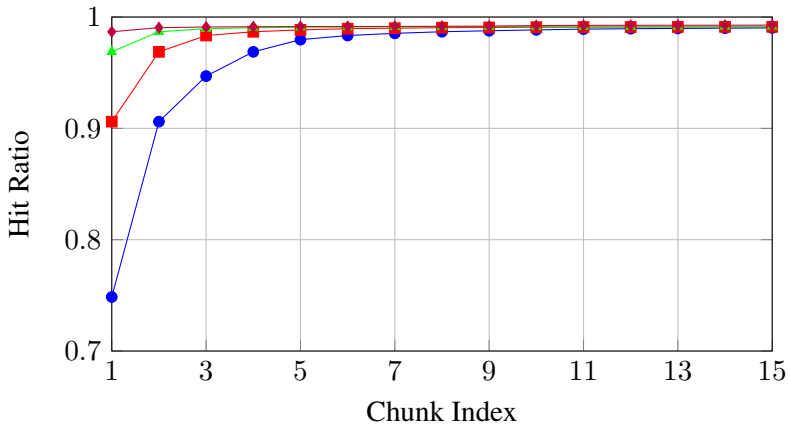
If an invalidation happens at the top of the order book, it's essential to quickly find the next highest (for sell) or lowest (for buy) price level that still has orders—this is the next valid price level. The speed at which we can find this next valid level depends on how close it is to the level that just changed. The mechanism for identifying and promoting the next valid price level is crucial for maintaining the order book's functionality, especially in fast-paced trading environments. This process ensures that the order book remains reliable for traders, reflecting the most up-to-date pricing information and available quantities for efficient market operations.

In this work, we introduce a novel method for navigating the order book to efficiently find the next valid price level following an invalidation. Our approach combines elements of both binary and linear search strategies into a hybrid binary-linear search algorithm. This method is designed to swiftly identify the subsequent valid entry in the order book, which is crucial for maintaining accurate and up-to-date market information. We introduce the concept of a *chunk* to facilitate a clear understanding of our approach. In essence, a chunk represents a cluster of price levels grouped together for the purpose of optimization. For instance, consider an order book with 128 price entries, where valid entries are marked as '1' and invalid ones as '0'. Traditionally, these could be represented as 128 individual integers in an array. Our method, however, aggregates these entries into clusters, or chunks, of bits. Using 4-bit integers to represent every four successive price levels, we reduce the need to manage 128 integers to just 32 unsigned integers. Similarly, employing 8-bit chunks would only require 16 unsigned integers. This n-bit association, which we refer to as a chunk, significantly streamlines the search process.

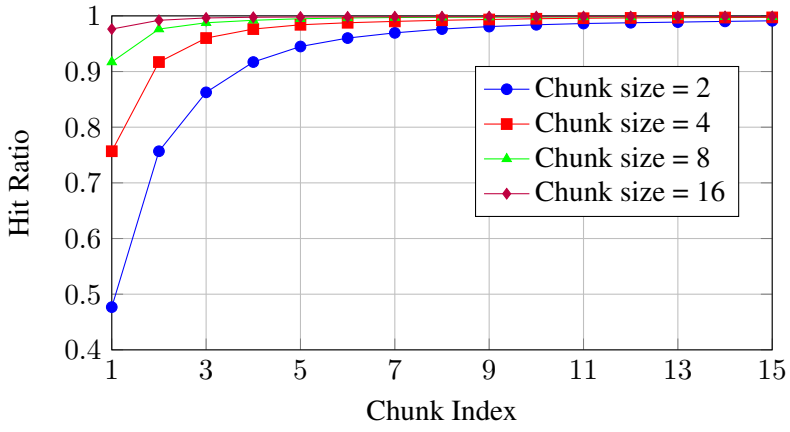
Employing this chunk-based structure, our search for the next valid price level becomes highly efficient. For example, when working with 4-bit chunks, to cover 16 price levels, we only need to linearly examine 4 integers to find a chunk that contains at least one valid entry. Once a chunk with a potentially valid entry is identified, we apply a modified binary search within that chunk. This method allows us to pinpoint the nearest valid price level quickly. Figure 3.1 illustrates the hit ratios for different stocks at various price levels. To illustrate the effectiveness of our method, we analyze hit ratios—the probability of finding the next valid entry within a specified number of chunks—for various stocks at different price levels. For instance, in the case of the stock CANFINHOME, with a chunk size of 2, we observe an 83 percent hit ratio at chunk index 2. This means that 83 percent of the time, the next valid entry is located within the next two chunks. Moreover, increasing the chunk size to 16 results in a 99 percent hit ratio within the first chunk. This efficiency enables us to locate the next valid entry in 99 percent of operations by performing a binary search within the chunk in just 4 clock cycles. These findings underpin the architectural and hardware data structure decisions for our order book design.



(a) CANFINHOME, Trades at Rs. 520



(b) CIPLA, Trades at Rs. 900



(c) BAJAJFINANCE, Trades at Rs. 5500

Figure 3.1: The figures in this analysis depict the hit ratio with respect to the chunk index based on the historical data of a ticker.

The determination of chunk size (or the hybrid binary-linear search factor) for any given ticker depends on the inherent characteristics of a stock and its trading activity. Key factors include, but are not limited to, the price gap between consecutive price levels and historical volatility, which enable us to approximate the ideal chunk size for a ticker. The base price of the ticker is also an integral component of this heuristic, as less expensive stocks typically have smaller price gaps, while more expensive stocks exhibit larger gaps. A typical estimate would be to select a chunk size such that the hit ratio is greater than 80 per cent, ensuring that the first valid price is obtained in the next chunk 80 per cent of the time.

The chunk size for a randomly chosen stock does not require intraday revision or frequent adjustments. Rather, periodic reevaluation is sufficient to ensure the chunk size accurately reflects the relevant stock information. This periodic re-determination of chunk size helps maintain the heuristic's relevance in light of various corporate actions, such as stock splits or bonus share issues, that may affect a particular stock.

3.4 Architecture

The NSE sends market feed as a stream of UDP packets. Now, before we can process the data in the packet to modify the orderbook, it has to be processed by the network and arbitration kernels within the FPGA. Figure 3.2 depicts a kernel level diagram of the system architecture. The UDP feed is transmitted through the Small Form-factor Pluggable (SFP)+ port on the Alveo U50 accelerator card, where it is received by the Ethernet kernel and the UDP/IP kernel. The packet is then passed frame-by-frame to the line arbitrator kernel to keep a check on the in-order arrival of UDP packets using the sequence number of the packet. The packets are further passed through the feed arbitrator kernel which ensures that packets corresponding to only desired tickers reach the orderbook kernel. The kernels are inter-connected using AXI-4 streaming interface.

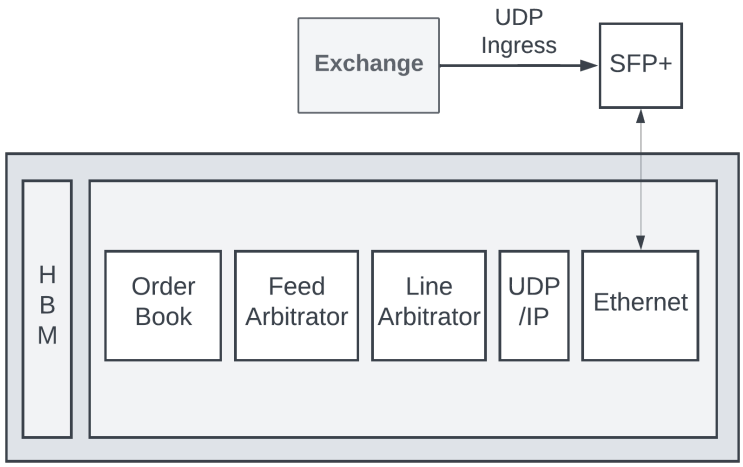


Figure 3.2: A high level overview of the system architecture

3.4.1 Ethernet, MAC and UDP/IP

In the network stack within FPGAs, the handling of data packets is structured through multiple layers, each with a distinct role. Initially, packets encounter the Physical Layer (PHY), responsible for the initial reception and conversion of data from its physical form into a digital format. Following this, the Media Access Control (MAC) Layer takes over, where the packets are subjected to error checking, address verification, and other control processes. Lastly, these packets are processed through the Ethernet Layer (ETH), which manages the Ethernet protocol standards, including framing and preparing the data payload for subsequent higher-level processing. The UDP/IP kernel enables the functionality to receives the market feed through the User Datagram Protocol (UDP).

This structured approach ensures efficient and accurate data transmission, essential in high-performance computing and networking applications like HFT. Also, as mentioned earlier, the data within a packet can be extracted and is accessible within the next clock cycle as when it arrives, reducing variability and increasing the predictability of the availability of any piece of data.

3.4.2 Line Arbitrator and Feed Arbitrator

The UDP/IP Kernel is responsible for streaming data to the Line Arbitrator Kernel, which manages out-of-order packet reception. While UDP multicast facilitates the delivery of a large volume of market feed at low latencies, its inherent unreliability may lead to packet loss or out-of-order issues. To mitigate these challenges, traders often choose to receive two lines of market feed. The Line Arbitrator Kernel is then employed to ensure the proper sequencing of packets for subsequent processing.

In this system architecture, the Feed Arbitrator Kernel plays a crucial role in processing the UDP feed. Its primary function is to extract all the data related to a set of tickers and selectively stream this information to the Order Book Kernel for consumption. Acting as a filter, the Feed Arbitrator Kernel ensures that only orders for stocks of interest are forwarded to maintain an organized order book.

3.4.3 Order Book

The Order Book Kernel acts as the core component within our system architecture, tasked with the real-time processing of the market feed. It diligently maintains the order book data structures, specifically focusing on selected tickers. The kernel also ensures the store of all orders of a ticker (price and quantity information) by organizing orders into a structured mapping, identified by their orderID.

The maintenance of the order book serves two primary purposes. Firstly, it enables the transfer of this data to the host memory, which allows for manual intervention and the execution of orders by humans. Secondly, it provides the necessary data to an order entry kernel, which is responsible for generating new trade orders, leveraging simplified Algorithmic Trading (AT) strategies that can be developed on our platform. It's important to note that both traders and AT systems often require access only to the top layers of the order book — typically the top 5 to 10 levels, and sometimes up to 20 for highly liquid stocks. This information is critical for making informed trading decisions. We utilize a specialized

hardware data structure designed to efficiently manage and provide access to these top-level order book details. A more detailed discussion of this hardware data structure and its implementation is provided in the following section for better understanding.

3.5 Hardware Data Structures in the order book

In high-frequency trading (HFT) systems, the organization and retrieval of order book data are paramount for achieving minimal latency. Traditionally, software-based HFT systems utilize binary search trees, such as AVL trees or Red-Black trees, to store the order book. These data structures enable the maintenance of a sorted order book and allow for constant-time market depth access. However, they introduce a trade-off: while they ensure data is well-organized, the operations on these trees—such as adding, deleting, or modifying orders—require logarithmic time to complete. This logarithmic complexity is manageable for order books with a relatively small number of price levels but becomes a bottleneck for instruments characterized by a high volume of price levels, where speed is of the essence.

In contrast to traditional CPU-based systems, Field-Programmable Gate Arrays (FPGAs) offer a distinct advantage due to their inherent design characteristics. FPGAs are configured during the design phase, creating a static setup that lacks the capability for dynamic memory allocation at runtime. This feature positions FPGAs as uniquely suited for HFT applications, as they leverage hardware-level parallelism and efficiency. FPGAs allocate resources such as logic elements and memory blocks during design, avoiding the complexities associated with software memory management and abstraction.

Given the static nature of FPGA configurations, an alternative approach to storing order book data involves the use of fixed-size arrays, with each element corresponding to a specific price level determined by the market's tick size. This arrangement, while straightforward, poses challenges in efficiently updating and accessing market-depth information, as it necessitates maintaining a sorted structure or devising a custom cache mechanism to achieve optimal performance. Our proposed solution addresses these challenges by implementing a custom cache structure tailored for constant-time complexity ($O(1)$), for adding entries to the order book. Similarly, operations to delete, modify, or execute trades are also optimized to operate in almost constant time, ensuring rapid processing, which is crucial for HFT systems. In our approach, we not only streamline transaction execution but also support tracking critical market parameters, such as best-ask-volume and best-bid-volume, which are indispensable for automated trading strategies.

Specifically, for markets like the Indian stock market, where the minimum tick size is 0.05 INR, we have developed a data structure based on fixed tick sizes. This section of the thesis delineates the data structures we have devised, demonstrating their applicability and efficiency on both the bid and ask sides of the order book.

3.5.1 orderBook

The orderBook is the primary data structure representing the accumulated quantity at a given price for any ticker. It is stored in the URAM and contains the price and accumulated quantity at all the price levels. Since we have a fixed tick price level, we use an array of structs. With the tick size set at 0.05 INR and exchange-provided prices in paise, a linear hash function is employed to map each valid price to an index in the array. Consequently, each array index corresponds to a specific price level, and the structure at that level stores the price, quantity, participants, and traded volume. An index in the orderBook array signifying a quantity greater than zero denotes a valid price level, whereas an index reflecting a quantity of zero is considered non-valid.

Given that each entry occupies 12 bytes and the system accommodates 20 distinct price levels per INR, approximately $\frac{2^{20}}{12*20} \approx 4370$ price levels can be stored within 1 MB of memory, facilitating a price fluctuation scope of INR 218. Utilizing a U50 Field-Programmable Gate Array (FPGA) with 28 MB of on-chip RAM for our analyses, it is feasible to accommodate up to $4370 * 28 = 122360$ price levels within the order book, translating to a potential price variation span of $218 * 28 = 6104$. In adherence to regulatory mandates that limit the daily price variability for a stock, a single FPGA can efficiently manage multiple securities, dependent on the initial price of these stocks. The linear hashing function, pivotal for mapping prices to their respective array indices, accounts for both the initial stock price and the maximal permissible price fluctuation within a trading session.

3.5.2 topCache and orderBookMask

The typical market depth monitored for most trading instruments extends to 5 levels. To facilitate constant-time access to these levels, we structure the order book as an array of structures, mirroring the configuration of the orderBook itself. Direct linear traversal of the orderBook to identify the top 5 valid price levels, however, may lead us through numerous invalid price levels. In the ideal scenario, the first five levels would all be valid, yet this situation still requires us to expend 5 cycles for each new order processed. To circumvent this inefficiency, we advocate for the continuous maintenance of a sorted array of structures, referred to as topCache. It is important to note that any incoming order or trade, being mapped to a specific price, is capable of impacting at most a single level within the topCache, a situation that can be efficiently managed with a single pass of a hardware-friendly sorting routine.

Further enhancing our system, we introduce a bitmask array named orderBookMask, mapped to the size of the order book for any given stock. This bitmask signifies the validity of entries at corresponding price levels within the order book: a bit is set to '1' to indicate a valid entry (where $quantity > 0$) and to '0' for entries lacking quantity. For ease of management and understanding, this bitmask is segmented into chunks, with each chunk composed of multiple bits. In practice, the bitmask array is implemented as an array of arbitrary precision integers (ap_uint), with each bit representing the validity of the price level in the orderBook. This setup is crucial for the operation of the OBReduction algorithm, as detailed in Section 3.7.

3.5.3 allOrders

Due to the nature of the NSE orders, it is necessary for us to keep a record of past orders to facilitate trade, delete, and modify requests. With the daily volume of unique orders potentially escalating into the millions, the capacity of on-chip memory proves inadequate for this purpose. Consequently, we resort to leveraging High Bandwidth Memory (HBM) for the archival of order data. Although a hash-map-based strategy might initially appear optimal, it falls short in terms of efficiency within the HBM framework, which favours burst read operations. Instead, we apply a straightforward hashing technique to associate order IDs with their corresponding locations in the HBM. Our evaluations indicate that the daily influx of unique orders varies between one hundred million to one hundred and fifty million. Given the availability of eight gigabytes of HBM, our storage infrastructure is well-equipped to handle more than double the peak volume of orders identified through our analysis.

3.6 Update Algorithms

In this section, we describe the algorithms used for the four types of orders enumerated in Section 3.2. While each operation requires updating the orderBook and allOrders data structures, the topCache is only updated if an order makes a change corresponding to the top k levels maintained in the topCache. Generally, we follow the principle that changes to the order book are made prior to any changes in the topCache because a change in the order book does not necessarily modify the topCache. The algorithms are explained for ask messages, assuming a market depth of 5, for simplicity. They are applicable in a similar way to bid messages, with the only difference being the manner in which the bid prices are sorted compared to ask prices. Table 3.3 is a sample topCache used to illustrate each update algorithm listed in this section. There is also information corresponding to the two following valid price levels from the order book to help illustrate the delete, modify, and trade operations.

Order Book Snapshot					
Bid	Orders	Qty	Ask	Orders	Qty
210.55	1	100	210.55	2	188
210.40	1	4	210.60	1	5
210.35	2	301	210.65	1	100
210.30	3	74	210.75	1	5
210.20	11	2871	210.80	1	5
210.15	1	65	210.95	1	15
210.05	2	10	211.05	1	26

Table 3.3: Extended Top Cache

3.6.1 operationNew

A new order is the simplest order message, which instructs that the given quantity is added for the given price level. In the operationNew routine, the orderBook is updated by verifying if an order already exists at the specified price. If it does, the quantity is added to the existing quantity. Otherwise, the quantity is set, and the bit corresponding to the price index is set to 1 in the bitmask. We also update the order id mapping in the allOrders. To update the topCache, we compare the encountered price with the fifth level of the topCache. If the price is greater or equal, we incorporate the quantity. Otherwise, we replace the fifth level with the total quantity at the current price in the order book, followed by sorting.

To illustrate this algorithm, consider a new ask order at 210.70 with quantity 50. The algorithm first checks the order book to update the price and quantity for this level. Given that the topCache lacks an entry at 210.70, this implies a zero quantity available in the order book at this price. After the order book update, the algorithm also updates the order ID to price and quantity mapping within the HBM. Subsequently, the topCache is updated. The algorithm compares the last price in the topCache with the new order price. In this instance, 210.70 is less than 210.80, prompting a replacement of the topCache price level with 210.70. Finally, a hardware-optimized sorting routine maintains the topCache in sorted order.

3.6.2 operationDelete

A delete or a cancellation order is intended to remove the quantity of shares that were earlier bid or asked at a given price. Once the order has been cancelled, that orderId cannot be updated in the future. In the operationDelete routine, we first extract the price and quantity from allOrders, and then we remove the entry of the given orderId from allOrders. Then we subtract the quantity at the corresponding price level in the orderBook. If the total quantity becomes zero, we update the bitmask array to indicate no quantity at that price, i.e. the price is no longer valid at that instance. If that price is present in the topCache, and the quantity does not get reduced to zero we simply update the modified quantity at that price. However, if the price gets reduced to zero, that level needs to be cleared and replaced by the successive price levels from the orderBook. We determine the successive valid price using the OBReduction algorithm on the orderBookMask array. We then substitute the cleared level by the obtained price, followed by sorting.

To illustrate this algorithm, consider a delete order at 210.65 for a quantity of 100. The algorithm first checks the order book to update the price and quantity at this level. In the given example, there is only a single order at the given price, so it gets cleared. After updating the order book, the algorithm also updates the order ID to the price and quantity mapping within the HBM. Subsequently, the topCache is updated. The algorithm compares the last price in the topCache with the new order price. In this instance, 210.65 is less than 210.80, indicating there is a valid price level of 210.65 in the topCache that needs to be cleared. We use the hybrid binary-linear search algorithm to determine the next valid price

in the order book (here 210.95). Now, we replace the price level at 210.65 with the information for the price level at 210.95 from the order book, followed by sorting.

3.6.3 operationModify

The purpose of a modify order is to modify the price and quantity of a given order(referenced by orderId), and substitute it by the supplied price and quantity. It is to be noted that the incoming packet only contains the information about the new price and quantity, hence the older price and quantity needs to be refered from allOrders, so as to update the other data structures accordingly.

In the operationModify routine, we extract the price and quantity from allOrders using the orderId and then update it with the new price and quantity received in the order message. Then we remove the older price and quantity from the orderBook and topCache, and we update the new information subsequently. Logically, this is a combination of individual delete (operationDelete) and add (operationNew) operations on the respective data structures. However, certain modify operations which require only a change of quantity at a given price level are executed directly by updating the data structures without the need for reduction of any sort. In other cases, we do a simplified combination of delete and add, eliminating duplicate memory access and computation.

To illustrate this algorithm, consider a modify order at 210.65 for a quantity of 100, for which the new price is 210.60 for a quantity of 100. The algorithm first retrieves the existing price and quantity mapping at 210.65 from the HBM and updates it with the new information. The algorithm then checks the order book to update the price and quantity at this level. In the given example, there is only a single order at the given price, so it gets cleared. Now, since there is an existing order at 210.60, the quantity is aggregated at the existing price. Subsequently, the topCache is updated. Since the modify order has to be treated as a delete followed by an add operation, we first follow the steps to remove the level at 210.60, similar to the operationDelete in Section 3.6.2, followed by sorting. Then, we add the 100 quantity at 210.65, similar to operationNew in Section 3.6.1. Due to an in-place update at 210.65, we do not need to perform a sort operation after adding a new quantity.

3.6.4 operationTrade

A trade, unlike other orders is generated by the exchange not the market participants. A trade happens when a bid matches an ask. Quantity of any bid order is allowed to match partially with the quantity of an ask order, and vice-versa. Another important indicator associated with trade is traded volume, which is the quantity of stock already traded at the given price.

In the operationTrade routine, trades can show correspondence to delete operations done on both the bid and ask sides, provided the required criteria are met. In contrast to the delete order, a trade can partially clear an order. For example, if we assume that there is one bid and ask order each at Rs 199.05, assuming the bid order is for 100 quantity, and the ask order is for 50 quantity, the ask order gets completely cleared, whereas the bid order gets updated with 50 quantity remaining. Hence, the

operationTrade routine is implemented as adapted delete operations on both the bid and ask side of the orderBook. We keep the data structures for bid and ask orders isolated so that operations on both sides can be implemented in parallel on the FPGA.

To illustrate this algorithm, consider a trade order at 210.55 on both the ask and bid sides for a quantity of 100. The algorithm first checks the order book to update the price and quantity at this level. In the given example, there is only a single bid order at the given price, and its quantity equals the order quantity, so it gets cleared. On the ask side, there are two orders adding up to 188 units. Let us assume that it is split into 100 and 88, and the trade contained the order ID with 100 quantity, so it gets cleared. We update both the ask and bid sides of the order book in parallel. After the order book update, the algorithm also updates the order ID to price and quantity mapping within the HBM on both the bid and the ask sides. Subsequently, the topCache is updated. On the ask side, the algorithm compares the last price in the topCache with the ask trade price. In this instance, 210.55 is less than 210.80, indicating there is a valid price level of 210.55 in the topCache that needs to be updated. However, since the level wasn't cleared, the hybrid binary-linear search algorithm is not required. Instead, we just update the information at 210.55 with the latest information from the order book. On the bid side, the algorithm compares the last price (210.20) in the topCache with the bid trade price (210.55). Since the bid side is sorted in descending order, when the trade price is greater than the last bid price, the topCache needs to be updated. In this case, we start from the top of the table since most trades happen at the top of the table. Then, the price level at 210.55 is replaced with the next valid price (210.15) determined using the hybrid binary-linear search, followed by a sort routine.

3.7 Hardware Friendly Order Book Update

In the previous section, we elaborated on the routines corresponding to the add, delete, modify, and trade operations. Except for the add operation, all of them require a reduction algorithm. We explain the reduction algorithm in this section.

For delete orders, we are only concerned with a single price level corresponding to the orderID. In case of a modify operation, we remove the quantity at a particular price level and add the quantity to the specified new price level. In certain cases, a modify order only represents a change in quantity, so the modification happens only at one price level. In the case of trades, we subtract the given quantities at both the bid and ask prices. However, since the bid and ask data structures are independent, trade can be treated as a delete order, with the only distinction being that a trade can remove partial quantity for the given orderID, whereas delete always removes all quantity at the specified price.

Except for the modify order, all these operations can affect only one price level in the topCache, on either side (bid/ask) and invalidate it at worst. Even for the modify order, it is implemented as a combination of a delete and add operation. This implies that only one price level can be invalidated for any order or trade. Hence, the reduction algorithm needs to be deployed once at most, to determine the next price level in succession.

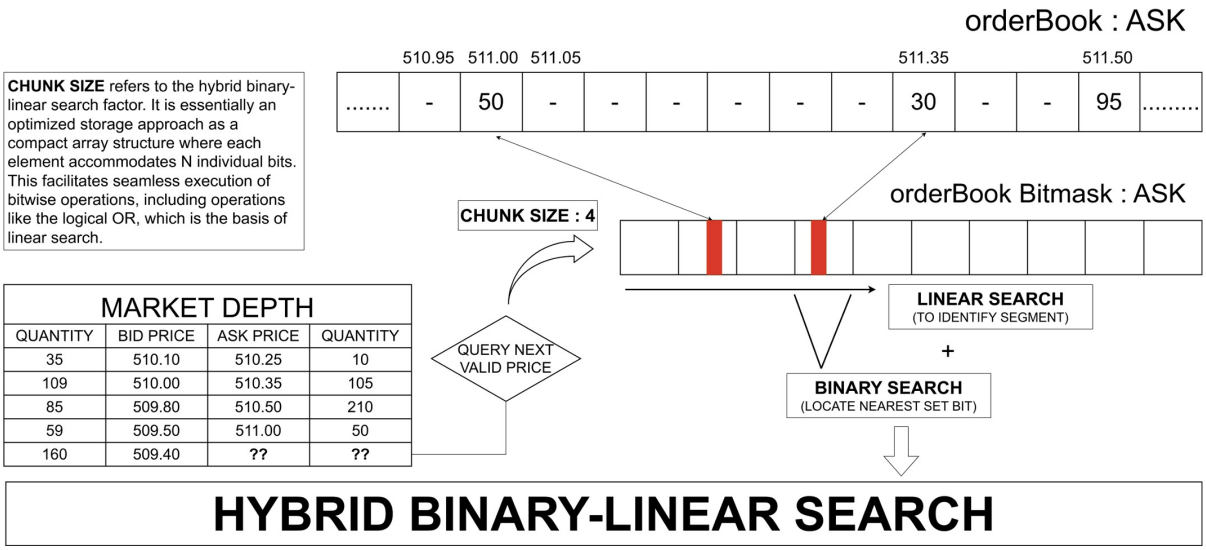


Figure 3.3: Hybrid Binary-Linear Search

Figure 3.3 is a graphic representation of how the OBReduction or the Hybrid Binary-Linear search algorithm works to maintain the topCache. The reduction algorithm, also called the OBReduction algorithm, helps determine the next valid price level in succession when a current price level is invalidated in the topCache. Fundamentally, determining the next valid price level requires a linear search as we need not only a valid price level, but also the first one in succession. However, the time complexity of linear search renders it unsuitable for this use case.

In our approach, we use a combination of binary and linear search to tackle this challenge. Instead of using the orderBook, referred to in Section 3.5.1 to iterate through individual price levels and determine the next valid price, we use the orderBookMask, referred to in Section 3.5.2 to determine the next valid price. The orderBookMask is nothing but a restructured version of the orderBook specifically to aid the OBReduction algorithm. Each element of the orderBookMask(also called a chunk) represents an unsigned integer made up of bits corresponding to successive x price levels, where x is known as the chunk size, which is the number of successive price levels represented in the chunk.

Building upon the approach outlined in our work, we delve into a practical example to elucidate the workings of our hybrid binary-linear search algorithm, particularly focusing on how it efficiently navigates the order book to identify the next valid price level. This example is based on a sample order book structure, depicted in Table 3.4, with the initial price level set at Rs. 19.05, or 1905 paise. Assuming a chunk size of 2, the order book mask (orderBookMask) segments are represented as 2-bit unsigned integers. In this scenario, the binary representation of the first two elements would be **11** and **01**, respectively. Here, a '1' denotes a valid price level within that chunk, signifying the presence of orders at that price. Should we opt for a larger chunk size, say 4, the first element's binary value would then be **1101**, encapsulating more price levels within a single chunk. It's imperative to note that a chunk holding all '0's signifies no valid price levels within that segment, reducing its integer value to zero. The

selection of the chunk size is a strategic decision informed by an analysis of the stock's trading activity on the previous day, as detailed in Section 3.3.

Price	Quantity	Count	Volume
19.05	50	1	500.00
19.10	30	20	375.00
19.20	20	3	315.00
19.25	10	11	200.00
19.35	15	2	200.00

Table 3.4: Sample Orderbook for Ask

To pinpoint the next valid price following an invalidation, our search begins with a linear traversal through the chunk array until we encounter a chunk with a non-zero value. This indicates the presence of at least one valid price level within that chunk. Upon identifying such a chunk, we deploy a modified bitwise binary search technique to locate the nearest set bit within it. The combination of the chunk's position in the array and the position of the identified '1'(set) bit within the chunk enables us to map back accurately to the corresponding valid price level in the order book. Through this example, we explain how, by judiciously determining the chunk size and employing our hybrid binary-linear search method, we can swiftly and effectively locate the next valid price level in the order book within a minimal number of cycles.

Chapter 4

Experimental Setup & Results

4.1 Experimental Setup

Our experimental configuration was devised to closely mimic the real-world conditions of stock market data transmission, specifically designed for performance evaluation in high-frequency trading scenarios. The primary component of our setup is the Xilinx® Alveo™ U50 accelerator card, which is known for its high throughput and low latency capabilities, making it an ideal choice for processing financial market data.

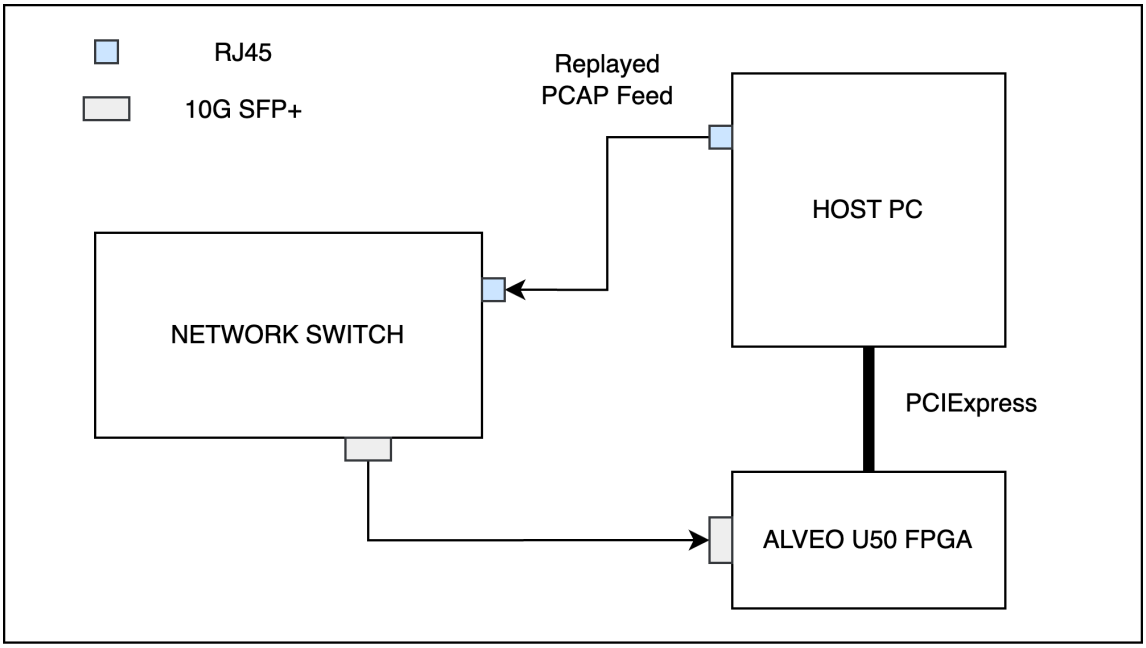


Figure 4.1: Experimental Setup

To simulate the real market environment, our approach involved the use of recorded Packet Capture (PCAP) files. The Alveo U50 card has a QSFP+ port, on which we connect a QSFP+ to four SFP+

breakout cables; each of the cables can be connected to one stream of multicast data feed. The core of our simulated environment was a network setup configured to replicate the conditions under which the FPGA on the Alveo U50 card would receive data in a live trading scenario. This setup, demonstrated in Figure 4.1, included a switch to which we fed the recorded PCAP files. The switch was configured to burst the packets onto an RJ45 port, which then forwarded the data to an SFP+ port. This arrangement was crucial for ensuring that the FPGA received the feed through the SFP+ port, simulating the actual operational setup of receiving live market data.

Since the number of stocks is very large, the market feed is divided into four data streams, and one can subscribe to the one containing data for the tickers of interest. Recognizing the limitations of the FPGA in terms of network features and advanced controller capabilities, we also tested our setup, configuring the switch in our existing setup to subscribe to the particular stream and receive feed corresponding to the tickers in that stream. By doing so, we could precisely control the data being forwarded to the FPGA, ensuring it received only the relevant market stream through the SFP+ port on the switch. This method allowed us to simulate a more dynamic and realistic trading environment where the FPGA would need to process specific data streams selectively.

4.2 Results

In this part of our study, we present the experimental findings, specifically looking at latency and how efficiently the card uses its available space. For our experiments, we maintained an order book that tracked two different securities, each with 50,000 price levels on both bid and ask sides. The setup is flexible, allowing for adjustments in the number of securities based on the ticker's characteristics and the trading platform's requirements. Additionally, the quantity of price levels designated for each security is dynamic and determined by the daily price fluctuation limits of the security, set by the exchange, one of the factors for which is the current price of the security. Thus, the selection and number of securities managed on the card can be tailored according to specific requirements.

4.2.1 Area Utilization

The U50 card employs the Xilinx Stacked Silicon Interconnect (SSI) technology, which splits the card into two Super Logic Regions (SLRs) with equal distribution of resources. We allocate the order book kernel to SLR0, which is closer to the High Bandwidth Memory (HBM), to reduce the need for data transfers across SLRs. The other kernels are placed in SLR1. Table 4.1 shows the resource usage in SLR0, revealing that around 75% of the Look-Up Tables (LUTs) and 99% of the Digital Signal Processors (DSPs) are still available, providing ample capacity for the development of complex automated trading systems. The URAM utilization is 62.5%, as the fixed tick order book is extensively partitioned by HLS for optimal throughput.

Logic and Block Utilization	
Site Type	SLR0 Utilization %
LUTs	25.54
Registers	18.17
BRAM	20.61
URAM	62.50
DSPs	1.39

Table 4.1: FPGA Resource Utilization for 2 tickers.

4.2.2 Latency

To test the latency of the orderbook subsystem, we utilize a kernel function operating in a free-running manner that increments counts per cycle. This allows us to measure the latency of order book update functions, considering the absence of a standardized intra-kernel latency measurement method. The U50 Accelerator card operates at a frequency of 300 MHz for its synthesized architecture. Through our testing, we observed an average latency of 105 cycles, translating to 350 nanoseconds, and identified a worst-case latency of 150 cycles, or 500 nanoseconds. This highest latency scenario is primarily the result of particular modification operations that necessitate updates to the topCache, such as the removal or addition of entries, and the execution of the OBReduction process. Previous works have not addressed such operations requiring the complexity of both add and delete operations in a single order. The latency results, including the network stack, are presented in Table 4.2.

FPGA Timing			
System	Frequency	Cycles	Latency
HFT system	300 Mhz	200	660 ns
OrderBook subsystem	300 Mhz	105	350 ns

Table 4.2: FPGA Latency results.

Since the functional requirements of NSE differ significantly from the exchanges present in previous contributions, it is not straightforward to compare the benchmarks of our work with previous contributions. Additionally, the operations required for the NSE, particularly the modify operation, are more complex than those of other exchanges on which previous works are based and also involve an extra data structure stored and accessed in the HBM. Although latency is not significantly lower than those of the previous works, it is important to note that we can maintain an order book of size 10 times that of Boutrous et al. [10]. Although the heap-based structure allows the order book to be sorted at all times, extracting the top levels after each insertion/update is an expensive operation. Moreover, in this work, delete orders are stored in a separate heap structure, which can severely affect worst-case latency due to

the potential accumulation of multiple delete orders and lead to potential race conditions. In contrast, our approach processes each order immediately, maintaining consistent average latency throughout the trading session and being scalable against order volume. In the work by He et al. [9], the latency measurement is reported separately for the add and publish (retrieve the market feed) routines. In our work, we update the topCache after every message, as well as include that latency in each function rather than recognising it as a separate process, ensuring updated order book data is utilized by the automated trading system to place orders.

Chapter 5

Conclusion and Future Work

In summary, this work introduces an FPGA-based framework designed for the efficient reconstruction of the order book in the Indian NSE market. Our approach distinguishes itself by ensuring the retention of all order data throughout the trading day, ensuring the correctness of the order book throughout. This necessity is effectively addressed by leveraging HBM memory with a direct hash function for straightforward and efficient data storage. The URAM is utilized to house detailed order book information, indicating the quantity available at each price level. Additionally, our strategy includes the use of register memory, or 'topCache', to dynamically keep track of the top k bids and asks, in line with the market depth parameter.

A key feature of this work is the implementation of a hybrid linear-binary search algorithm, which stands out for its simplicity and operational efficiency, optimizing both space and processing time. Unlike other systems that depend on periodic refreshes of their topCache, our solution allows for immediate updates on a per-order basis, ensuring a swift reflection of the market's current status. Furthermore, our framework has undergone extensive testing for correctness and performance in a close-to-production environment through the streaming of UDP packets. By offering a solution that simplifies and streamlines the management of order book data while ensuring real-time accuracy, we try to contribute to the infrastructure supporting high-frequency trading. We have made an attempt to advance order book update algorithms, with the goal of improving the efficiency and reliability of financial market operations.

We have also attempted to implement the order entry kernel for NSE, which allows us to connect to the exchange via TCP and allow us to place orders. Future developments include utilizing machine learning to devise simple and fast trading strategies, with the goal of integrating them directly into hardware to achieve quicker trading operations. Additionally, we can explore the automation of chunk size selection in search algorithms using market trends and historical data. Possible enhancements to the FPGA-based platform could involve integrating on-chip and High-Bandwidth Memory (HBM) architectures for efficient management of a higher number of order books. These advancements aim to optimize trading systems for improved performance.

List of Related Publications

- Vaibhav Kashera, Siddhant Jain, Abhishek Banerjee and Suresh Purini, **"Building Low-Latency Order Books with Hybrid Binary-Linear Search Data Structures on FPGAs"**, in proceedings of *2023 33rd International Conference on Field-Programmable Logic and Applications (FPL)*.

Bibliography

[1] H. R. Stoll, "Electronic trading in stock markets," *Journal of Economic Perspectives*, vol. 20, no. 1, pp. 153–174, March 2006. [Online]. Available: https://www.aeaweb.org/articles?id=10.1257/089533006776526067

[2] G. Nuti, M. Mirghaemi, P. Treleaven, and C. Yingsaeree, "Algorithmic trading," *Computer*, vol. 44, no. 11, pp. 61–69, 2011.

[3] C. M. Jones, "What do we know about high-frequency trading?" *Columbia Business School Research Paper*, no. 13-11, 2013.

[4] P. Gomber and M. Haferkorn, "High frequency trading," in *Encyclopedia of Information Science and Technology, Third Edition*. IGI Global, 2015, pp. 1–9.

[5] C. Leber, B. Geib, and H. Litz, "High frequency trading acceleration using fpgas," in *2011 21st International Conference on Field Programmable Logic and Applications*. IEEE, 2011, pp. 317–322.

[6] H. Jia, Y. Huan, C. Ding, Y. Yan, J. Cui, J. Wang, C. Cai, L. Xu, Z. Zou, and L. Zheng, "A domain-specific accelerator for ultralow latency market data distribution system," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 4, pp. 5465–5475, 2023.

[7] J. W. Lockwood, A. Gupte, N. Mehta, M. Blott, T. English, and K. Vissers, "A low-latency library in fpga hardware for high-frequency trading (hft)," in *2012 IEEE 20th annual symposium on high-performance interconnects*. IEEE, 2012, pp. 9–16.

[8] A. Boutros, B. Grady, M. Abbas, and P. Chow, "Build fast, trade fast: Fpga-based high-frequency trading using high-level synthesis," in *2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE, 2017, pp. 1–6.

[9] C. He, H. Fu, W. Luk, W. Li, and G. Yang, "Exploring the potential of reconfigurable platforms for order book update," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017, pp. 1–8.

[10] A. Boutros, B. Grady, M. Abbas, and P. Chow, "Build fast, trade fast: Fpga-based high-frequency trading using high-level synthesis," in *2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, 2017, pp. 1–6.

[11] A. DeHon and J. Wawrzynek, "Reconfigurable computing: what, why, and implications for design automation," in *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, 1999, pp. 610–615.

[12] K. Compton and S. Hauck, "Reconfigurable computing: a survey of systems and software," *ACM Computing Surveys (csuR)*, vol. 34, no. 2, pp. 171–210, 2002.

[13] J. Hoozemans, J. Peltenburg, F. Nonnemacher, A. Hadnagy, Z. Al-Ars, and H. P. Hofstee, "Fpga acceleration for big data analytics: Challenges and opportunities," *IEEE Circuits and Systems Magazine*, vol. 21, no. 2, pp. 30–47, 2021.

[14] J. Choi, R. Lian, Z. Li, A. Canis, and J. Anderson, "Accelerating memcached on aws cloud fpgas," in *Proceedings of the 9th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*, 2018, pp. 1–8.

[15] Amazon Web Services. Amazon ec2 f1 instances. https://aws.amazon.com/ec2/instance-types/f1/.

[16] Microsoft. Project catapult. https://www.microsoft.com/enus/research/project/project-catapult/.

[17] V. Kathail, "Xilinx vitis unified software platform," in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 173–174. [Online]. Available: https://doi.org/10.1145/3373087.3375887

[18] M. Elnawawy, A. Farhan, A. A. Nabulsi, A. Al-Ali, and A. Sagahyroon, "Role of fpga in internet of things applications," in *2019 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*, 2019, pp. 1–6.

[19] A. Rupani, P. Whig, G. Sujediya, and P. Vyas, "A robust technique for image processing based on interfacing of raspberry-pi and fpga using iot," in *2017 International Conference on Computer, Communications and Electronics (Comptelix)*, 2017, pp. 350–353.

[20] Z. Wu, K. Hammad, E. Ghafar-Zadeh, and S. Magierowski, "Fpga-accelerated 3rd generation dna sequencing," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 14, no. 1, pp. 65–74, 2020.

[21] S. Ren, V.-M. Sima, and Z. Al-Ars, "Fpga acceleration of the pair-hmms forward algorithm for dna sequence analysis," in *2015 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, 2015, pp. 1465–1470.

[22] E. B. Fernandez, W. A. Najjar, S. Lonardi, and J. Villarreal, "Multithreaded fpga acceleration of dna sequence mapping," in *2012 IEEE Conference on High Performance Extreme Computing*, 2012, pp. 1–6.

[23] M. J. Daniel and K. Kumar, "Recent trends and improvisations in fpga," *IOSR Journal of Electrical and Electronics Engineering (IOSR-JEEE)*, vol. 12, no. 3, pp. 2278–1676, 2017.

[24] P. Treleaven, M. Galas, and V. Lalchand, "Algorithmic trading review," *Commun. ACM*, vol. 56, no. 11, p. 76–85, nov 2013. [Online]. Available: https://doi.org/10.1145/2500117