Research on Hardware Systems for Data Analytics Based on Bitmap Index

NGUYEN XUAN THUAN

Department of Engineering Science

The University of Electro-Communications

A dissertation submitted for the degree of

Doctor of Engineering

September 2017

Research on Hardware Systems for Data Analytics Based on Bitmap Index

APPROVED

Prof. Cong-Kha PHAM, Chairman

Prof. Koichiro ISHIBASHI

Prof. Yoshinao MIZUGAKI

Prof. Yoshiyasu UENO

Assoc. Prof. Tomokazu SHIGA

Date Approved by Chairman _____

This page intentionally left blank.

I would like to dedicate this dissertation to my entire family who encouraged me to pursue a higher education.

This page intentionally left blank.

Acknowledgements

First of all, I would like to express my deep respect and appreciation to my advisors, Prof. Cong-Kha Pham and Prof. Koichiro Ishibashi, for their mentorship throughout the research process. I would also like to express my sincere appreciation to the committee members, including Prof. Yoshinao Mizugaki, Prof. Yoshiyasu Ueno, and Assoc. Prof. Tomokazu Shiga, for their critical questions and valuable comments on this dissertation.

Second, the completion of this dissertation is due largely to the support of my friends in the PHAM laboratory. I would like to thank Dr. Duc-Hung Le for his sincere assistance. I also thank Trong-Thuc Hoang for the countless hours of brainstorming and helping. I also thank all of my Vietnamese friends, Japanese friends, as well as other international friends for keeping my spirits up.

Third, I would like to thank Mr. Katsumi Inoue, president of Advanced Original Technologies Co., Ltd (AOT) and Mr. Osamu Shimojo, president of Nippon Computer Dynamics Co., Ltd (NCD), as well as all members of the Information Recognition Technology (IRT) Section of NCD for their cooperation. The research was also supported by VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Synopsys, Inc., Cadence Design Systems, Inc., and Mentor Graphics, Inc.

Fourth, I would like to express my gratitude to the University of Electro-Communications and the Ministry of Education, Culture, Sports, Science, and Technology for providing me with an excellent opportunity to experience my study and life in Japan.

Last but not least, I am grateful to my entire family, especially my late father Xuan-Vu Nguyen, my mother Thi-Nhu-Nguyen Pham, my aunt Thi-Xuan-Loc Nguyen, and my little sister Thi-Xuan-Mai Nguyen, for offering their love and sharing their wisdom with me throughout my life. I always pay tribute to you, my beloved father.

This page intentionally left blank.

Abstract

Recent years have witnessed a massive growth of global data generated from web services, social media networks, and science experiments, as well as the "tsunami" of Internet-of-Things devices. According to a Cisco forecast, total data center traffic is projected to hit 15.3 zettabytes (ZB) by the end of 2020. Gaining insight into a vast amount of data is highly important because valuable data are the driving force for business decisions and processes, as well as scientists' exploration and discovery.

To facilitate analytics, data are usually indexed in advance. Depending on the workloads, such as online transaction processing (OLTP) workloads and online analytics processing (OLAP) workloads, several indexing frameworks have been proposed. Specifically, B^+-tree and hash are two common indexing methods in OLTP, where the number of querying and updating processes are nearly similar. Unlike OLTP, OLAP concentrates on querying in a huge historical storage, where updating processes are irregular. Most queries in OLAP are also highly complex and involve aggregations, while the execution time is often limited. To address these challenges, a bitmap index (BI) was proposed and has been proven as a promising candidate for OLAP-like workloads.

A BI is a bit-level matrix, whose number of rows and columns are the length and cardinality of the datasets, respectively. With a BI, answering multi-dimensional queries becomes a series of bitwise operators, e.g. AND, OR, XOR, and NOT, on bit columns. As a result, a BI has proven profitable for solving complex queries in large enterprise databases and scientific databases. More significantly, because of

the usage of low-hardware logical operators, a BI appears to be suitable for advanced parallel-processing platforms, such as multi-core CPUs, graphics processing units (GPUs), field-programmable logic arrays (FPGAs), and application-specific integrated circuits (ASIC).

Modern FPGAs and ASICs have become increasingly important in data analytics because they can confront both data-intensive and computing-intensive tasks effectively. Furthermore, FPGAs and ASICs can provide higher energy efficiency, compared to CPUs and GPUs. As a result, since 2010, Microsoft has been working on the so-called Catapult project, where FPGAs were integrated into datacenter servers to accelerate their search engine as well as AI applications. In 2016, Oracle for the first time introduced SPARC S7 and M7 processors that are used for accelerating the OLTP databases. Nonetheless, a study on the feasibility of BI-based analytics systems using FPGAs and ASICs has not yet been developed.

This dissertation, therefore, focuses on implementing the data analytics systems, in both FPGAs and ASICs, using BI. The advantages of the proposed systems include scalability, low data input/output cost, high processing throughput, and high energy efficiency. Three main modules are proposed: (1) a BI creator that indexes the given records by a list of keys and outputs the BI vectors to the external memory; (2) a BI-based query processor that employs the given BI vectors to answer users' queries and outputs the results to the external memory; and (3) an BI encoder that returns the positions of one-bits of bitmap results to the external memory. Six hardware systems based on those three modules are implemented in an FPGA in advance for functional verification and then partially in two ASICs—180-nm bulk complementary metal-oxide-semiconductor (CMOS) and 65-nm Silicon-On-Thin-Buried-Oxide (SOTB) CMOS technology—for physical design verification. Based on the experimental results, these proposed systems outperform other CPU-based and GPU-based designs, especially in terms of energy efficiency.

Contents

List of Abbreviations

ALM	Adaptive Logic Module
ALU	Arithmetic Logic Unit
ASIC	Application Specific Integrated Circuit
BI	Bitmap Index
BIC	Bitmap Index Creator
BIE	Bitmap Index Encoder
BIQP	Bitmap-Index-Based Query Processor
CAM	Content-Addressable Memory
CPU	Central Processing Unit
DBMS	Database Management System
DDR3	Double Data Rate 3
DMA	Direct Memory Access
DPM	Dual-Port Memory
EB	Exabyte
FPGA	Field-Programmable Gate Array
GB	Gigabyte

GPU	Graphics Processing Unit
I/O	Input/Output
IoT	Internet-of-Things
LE	Logic Element
MPE	Multi-match Priority Encoder
PCI	Peripheral Component Interconnect
PE	Priority Encoder
RAM	Random-Access Memory
SDRAM	Synchronous Dynamic Random-Access Memory
SOTB	Silicon On Thin Buried Oxide

List of Figures

This page intentionally left blank.

List of Tables

Chapter 1

Introduction

The purpose of this chapter is to establish a general understanding of data analytics and the general approaches to a high-performance analytics system. Three index/query methods and three platforms for data analytics are briefly mentioned. The motivation and key contributions of this research are then stated. Finally, the layout of the dissertation is presented at the end.

1.1 What Is Data Analytics?

It is well-known that the amount of global data is exponentially increasing because of the rapid expansion of scientific observations and experiments, social networking services, mobile devices, and the Internet-of-Things (IoT). For example, Facebook, which was launched in 2004, is now the third most popular website on the Internet with 1.86 billion daily active users (as of February 2017). Corresponding to this large number of users, an enormous amount of data, including a vast stockpile of new messages, photos, videos, and more recently, 360-degree video, is generated every day. The total data size of Facebook, therefore, currently reaches several exabytes (1 EB = 10^9 GB).

Analyzing a huge amount of data to discover hidden patterns, unknown correlations, or valuable information for business decisions and scientific experiments is undoubtedly a crucial and time-consuming task. For example, powerful data analytics can precisely predict the emerging market trends so that businesses can

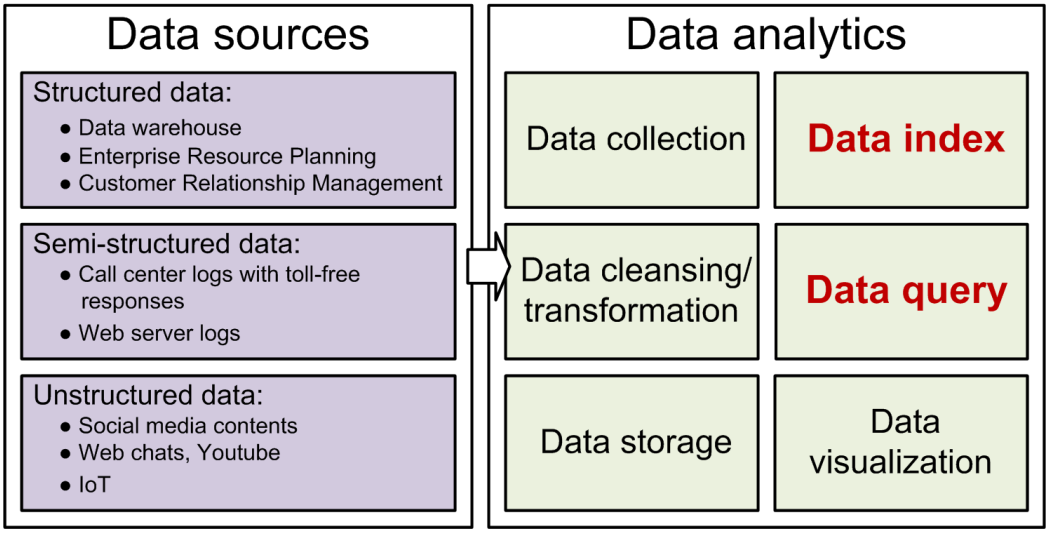


Figure 1.1: The process of data analytics.

respond more quickly and gain a competitive edge over their rivals. However, the input data are mostly highly complex because they encompass a mix of structured, semi-structured, and unstructured data, such as social media contents, text from customer emails and survey responses, and more recently, data captured by sensors connected to the IoT [1].

As a result of this complexity, the process of data analytics is composed of many stages, as illustrated in Figure 1.1. To begin, data from different sources are collected, cleansed, and transformed into a defined format, so that they can then be stored and loaded into an analytics system, such as a Hadoop cluster, a NoSQL database, or a data warehouse. The data index and data query stage are performed whenever users want to query information in the database. These are considered to be the two most important stages in an analytics system. Data indexing is completed in advance to greatly reduce the execution time of the data query stage. Finally, the query results are visualized in a proper way to aid business executives or end users in their decision making. Machine learning algorithms can also be applied with those data to construct an analytical model, from which several alarms can be automatically triggered, whenever something unexpected is uncovered.

Most studies on the efficient analytics system focus on data index and query

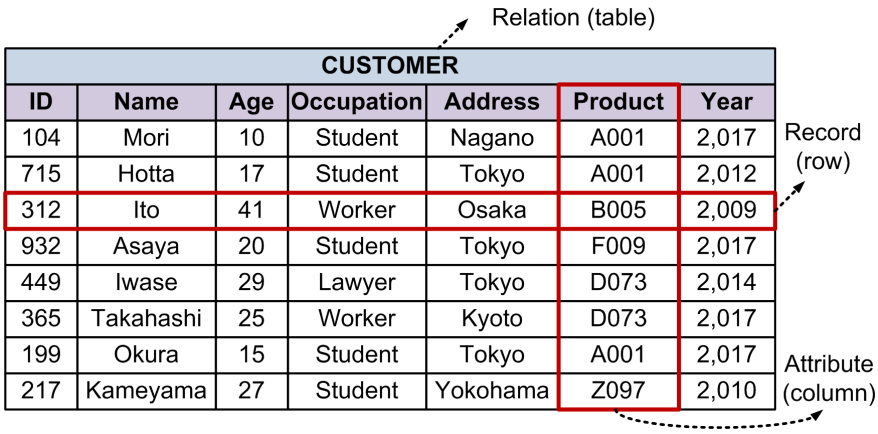


Figure 1.2: The CUSTOMER relation.

based on tree, hash, and bitmap structure, as well as their implementation in the central processing units (CPUs), graphics processing units (GPUs), and field-programmable gate arrays (FPGAs). Section 1.2 and 1.3 give some common understanding of those studies.

1.2 Approach to Data Analytics: Index and Query

Efficient approaches to index and query are indispensable to constructing a high-performance data-analytics system. Specifically, an efficient index method aims to look any data up in the database as fast as possible, while an efficient query method aims to use the index results to answer all queries as fast as possible. The concept of indexes in a database management system (DBMS) is similar to book catalogs in a library or even like an index in a book. In this system, a list of (*key*, *pointer*) tuples is organized in such a way that, given a specific key K_i or a range of keys $\{K_i - K_j\}$, the index search will return the corresponding pointers, from which the corresponding records can be accessed. Tree index, hash index, and bitmap index (BI) are the most widely used approaches in current DBMSs.

The following contents will sequentially describe each approach, with and without index, using a CUSTOMER *relation*, also called table, as an example. CUSTOMER is comprised of eight *records*, also called rows, each of which stores seven *attributes*, or the information of each record, as seen in Figure 1.2. The

example query is given as *"find all customers whose ages are between 10 and 19, is living in Tokyo, and bought product A001"*.

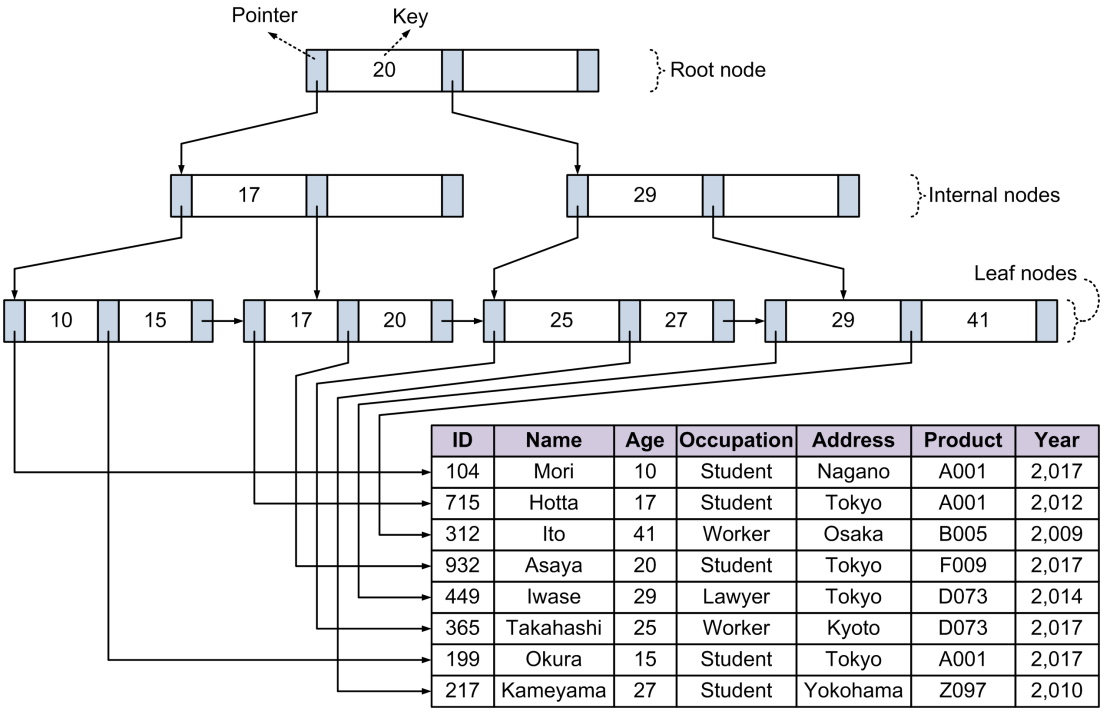
1.2.1 Full Table Scan

Full table scan, or sequential scan, is a scan made on a database where each row of the table under the scan is read in a sequential order and the columns encountered are checked for the validity of a condition. In order to solve the example query above, we check each record from the beginning to the end. In every check, we have to confirm whether the values of *Age* is between 10 and 19, *Address* is "Tokyo", and *Product* is "A001". If all of them are matched, that customer satisfies the example query. A full table scan is usually the slowest method of scanning a table, due to the large number of input/output (I/O) reads required from the disk which consists of multiple seeks as well as costly disk to memory transfers.

1.2.2 Tree Index and Query

A tree index stores a list of (key, pointer) tuples in a tree-like structure [2, 3]. B^+-tree is the most widely used tree index. To solve a query above, one option is to create a B^+-tree of attribute *Age* of CUSTOMER, as shown in Figure 1.3. This tree contains one root node, two internal nodes, and four leaf nodes. The "B" in B^+-tree stands for "balanced", because every path from the root to any leaf of the tree is of the same length. A typical node contains several pairs of keys and pointers. The key values within a node are kept in sorted order, where the left-most node and right-most node contain the smallest value and largest value, respectively. The pointer values within the leaf nodes also point to the corresponding rows of CUSTOMER, or the record location in a database.

Suppose that we wish to find a record with *Age* of 15, or the search key value $K_i = 15$. The function intuitively starts at the root of the tree and traverses down the tree until it reaches a leaf node that would contain the specified key, if it exists in the tree. From the root, the first internal node is selected because $15 < 20$. Likewise, from the first internal node, the first leaf node is selected because $15 < 17$. DBMS will then perform a sequential search in this node to find the



ID	Name	Age	Occupation	Address	Product	Year
104	Mori	10	Student	Nagano	A001	2,017
715	Hotta	17	Student	Tokyo	A001	2,012
312	Ito	41	Worker	Osaka	B005	2,009
932	Asaya	20	Student	Tokyo	F009	2,017
449	Iwase	29	Lawyer	Tokyo	D073	2,014
365	Takahashi	25	Worker	Kyoto	D073	2,017
199	Okura	15	Student	Tokyo	A001	2,017
217	Kameyama	27	Student	Yokohama	Z097	2,010

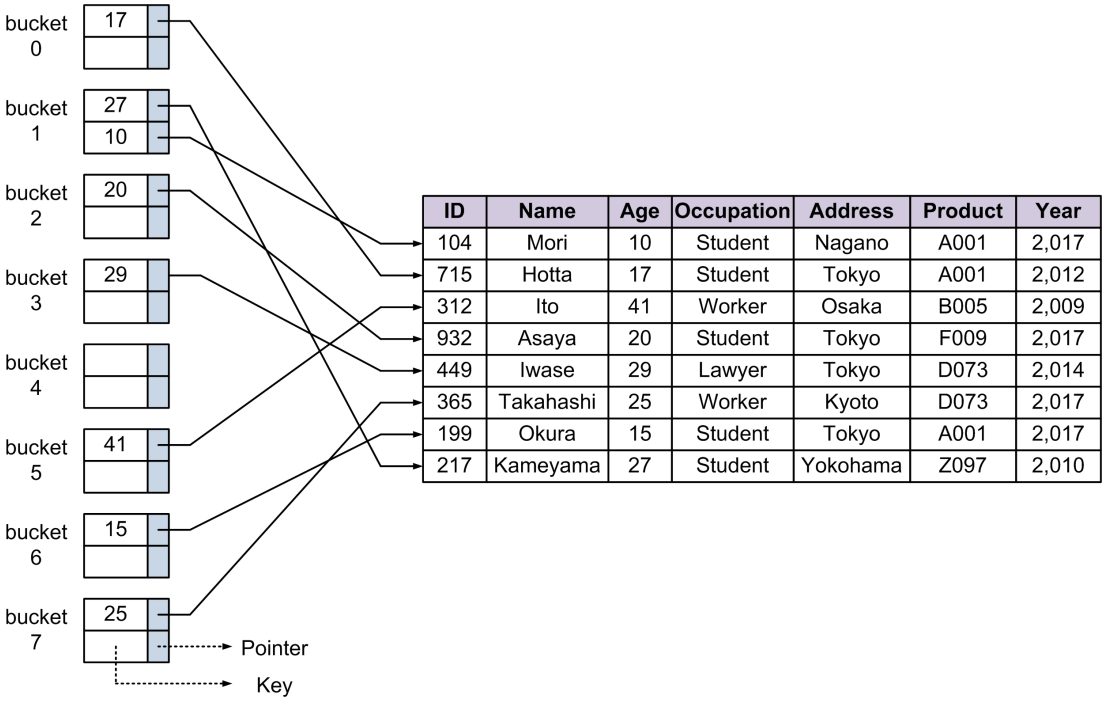
Figure 1.3: The B$^+$-tree index on key *Age* of CUSTOMER relation.

key value $K_i = 15$, from which the corresponding record can be located. If there is no key value $K_i = 15$ in the leaf node, i.e. no record exits in the relation, null will be returned to indicate failure.

B$^+$-tree can also be used to find all records with search key values in a specified range $\{K_i - K_j\}$. Such queries are called range queries. For example, to find all customer records with age in a specified range above, i.e. $\{10 - 19\}$, we first look for the leaf that contains a search key value $K_i = 10$. We then step through records with the stopping condition being that $K_j > 19$. Furthermore, to solve the example query above, in each found record, we have to check whether the values of *Address* and *Product* are "Tokyo" and "A001", respectively. If all of them are also matched, that customer satisfies the example query.

1.2.3 Hash Index and Query

The hash-based approach allows us to avoid accessing an index structure to locate records [4]. In hashing, the term *bucket* denotes a unit of storage that can point



ID	Name	Age	Occupation	Address	Product	Year
104	Mori	10	Student	Nagano	A001	2,017
715	Hotta	17	Student	Tokyo	A001	2,012
312	Ito	41	Worker	Osaka	B005	2,009
932	Asaya	20	Student	Tokyo	F009	2,017
449	Iwase	29	Lawyer	Tokyo	D073	2,014
365	Takahashi	25	Worker	Kyoto	D073	2,017
199	Okura	15	Student	Tokyo	A001	2,017
217	Kameyama	27	Student	Yokohama	Z097	2,010

Figure 1.4: The hash index on key *Age* of CUSTOMER relation.

to one or more records. A hash function is applied on a search key to identify a bucket, and then store the key and its associated pointers in the bucket.

Suppose that K denotes a set of key values, B denotes a set of all bucket addresses, and h denotes a hash function from K to B, i.e. $B = h(K)$. To insert a record with a search key K_i, we first compute $h(K_i)$, which gives the address of the bucket B_i for that record. The record i is then stored in that bucket B_i. To perform a lookup on a key value K_i, we simply compute $h(K_i)$, then access the bucket with that address. Suppose that two keys, K_i and K_j, have the same hash value, i.e. $h(K_i) = h(K_j)$. If we perform a lookup on K_i, the bucket $h(K_i)$ points to both record with key K_i and record with key K_j. Hence, the key values in the bucket must be verified to obtain the correct record.

An example of a hash index on *Age* that consists of eight buckets is depicted in Figure 1.4. The hash function h computes the sum of all digits of every *Age* modulo eight. Take an example of $Age = 29$, or the search key value $K_i = 29$, bucket 3 is used to store record with this K_i, because the sum of all digits is 11 and 11 modulo $8 = 3$. In the case of $K_i = 27$ and $K_j = 10$ that shares the same bucket,

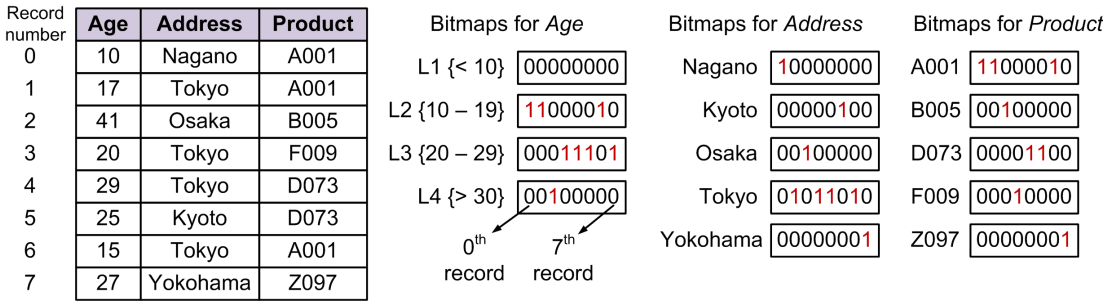
post-processing must be performed to obtain the correct record. Although each search key has only one associated pointer in this example, multiple pointers can be associated with each key in the real applications. To solve the example query above, we find all records with *Age* from 10 to 19, and in every match, we then check whether the values of the *Address* attribute and the *Product* attribute are "Tokyo" and "A001", respectively. If all of them are also matched, that customer satisfies the example query.

1.2.4 Bitmap Index and Query

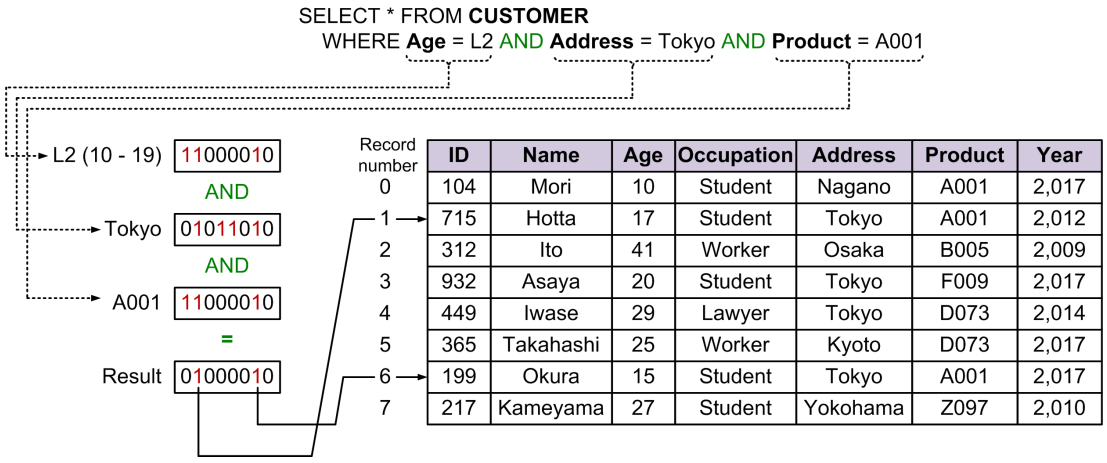
BI is a specialized type of index designed for easy querying on multiple keys, although each BI is built on a single key. The term BI was originally described by Wong et al. [5] in 1985 and later popularized by O'Neil et al. [6] in 1997. As a result of solving multi-key queries effectively, BI is widely applied in large enterprise databases and scientific databases.

A BI is simply an array of bits. Figure 1.5(a) illustrates all BIs of three attributes *Age*, *Address*, and *Product*. To begin, all records in a relation must be numbered sequentially. Attribute *Age* is split into four smaller ranges, from L1 to L4, to simplify the data analysis. Four bitmaps corresponding to all ranges is then created. It is noted that the studies on selecting optimum range, such as encoding and binning, can be found in detail in [7, 8]. Because *Age* of record 0^{th} record and 1^{th} record is within a range of $\{10 - 19\}$, the first two bits of L2 turn into one. Likewise, because there is no *Age* smaller than 10, all bits of L1 become zero. Attributes *Address* and *Product* take five values corresponding to the location name and product, respectively.

Figure 1.5(b) indicates how powerfully BI can cope with the multi-key query given above, i.e. *"find all customers whose ages are between 10 and 19, is living in Tokyo, and bought product A001"*. To answer this query, we fetch the bitmaps for *Age* value L2, *Address* value "Tokyo", and *Product* value "A001", and then perform an intersection, or bitwise logical AND, of three bitmaps. In other words, we compute a new bitmap where i^{th} bit has a value one, if the i^{th} bit of the three bitmaps are both one, and has a value zero otherwise. In the example in Figure 1.5(b), the intersection of three bitmaps *Age* = L2 (11000010), *Address* = "Tokyo"



(a) The bitmaps for *Age*, *Address*, and *Product*.



(b) The BI-based query processing.

Figure 1.5: The BIs on multiple keys of CUSTOMER relation.

(01011010), and *Product* = "A001" (11000010) gives the bitmap 01000010. Based on the value of bitmap result, i.e. the 1^{st}-bit and 6^{th}-bit are ones, the 1^{st} and 6^{th} record contain the result for the query.

The summary of the three approaches regarding data index and query is shown in Table 1.1. Depending on the specific applications, a suitable index method is selected. For example, in the e-commerce databases where millions of products are bought (inserted to the store) and sold (removed from the store) every single minute, using B^+-tree index or hash index can obtain a high benefit. However, in the OLAP-like workload, most queries are highly complex and involve aggregations, not to mention the limited execution time. Therefore, BI is considered as a prominent solution [9].

	Advantages	Disadvantages
B^+-tree index	– Good performance for lookup due to the balance property – Range queries support	– Memory access and parallel processing – Multi-key queries unsupported
Hash index	– Very good performance for lookup, due to the elimination of index structure access	– Bucket overflows – Range queries and multi-key queries unsupported
Bitmap index	– Very good performance for lookup, due to the bitwise logical operations usage – Range queries and multi-key queries support – Efficient memory access and parallel processing capability	– Strategy of selecting optimum range [7, 8]

Table 1.1: The summary of index and query approaches.

1.3 Approach to Data Analytics: Platforms

Efficient platforms for data index and query are indispensable for a high-performance analytics system. CPUs, GPUs, and FPGAs are the most commonly used platforms applied in such a system. The following contents will briefly describe their architectures and advantages.

1.3.1 CPU-Based Platform

Moore's Law has dominated the computer industry since it was firstly defined in 1965 by Gordon E. Moore. According to Moore's Law, the number of transistors on a CPU chip is predicted to double every two years, which has generally meant that the chip's frequency will also increase. Since 2005, more parallelism via multiple cores, with multiple levels of cache memory and arithmetic logic units (ALUs) per chip, has remarkably enhanced the computing capability, while keeping the power consumption steady. Because of these benefits, multi-core CPUs, as shown in Figure 1.6, are currently exploited to solve compute-intensive problems, instead of using the single-core high-frequency ones.

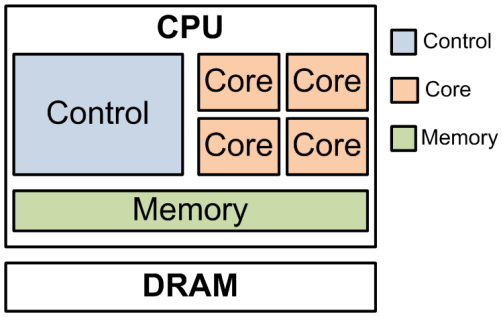


Figure 1.6: The illustration of CPU architecture.

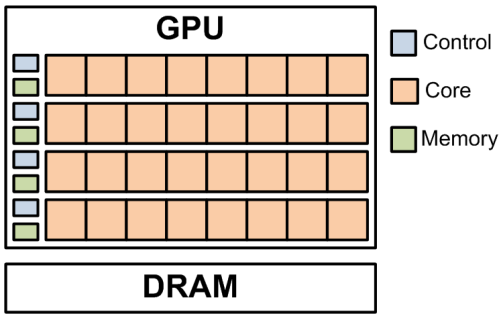


Figure 1.7: The illustration of GPU architecture.

Besides a stand-alone computer, a distributed system containing a large number of connecting computers has also been designed to cope with the heavy computing workloads of big data analytics. For example, in the worldwide Large Hadron Collider (LHC) computing grid project [12], measurement data are stored, distributed, and analyzed by more than 170 computing centers in 42 countries linked by the high-speed network infrastructure. Another example is the NERSC Edison supercomputer that contains 5,586 computer nodes, each of which is comprised of a Intel Ivy Bridge 24-core 2.4-GHz CPU and 64-GB DDR3 SDRAM [13]. In other words, distributed computing is currently indispensable to resolving the problem of big data.

1.3.2 GPU-Based Platform

GPU is a specialized microprocessor that was first designed for rapidly manipulating intensive graphics processing tasks for output to a display device. Its architecture is similar to that of a CPU, as depicted in Figure 1.7. Modern GPUs

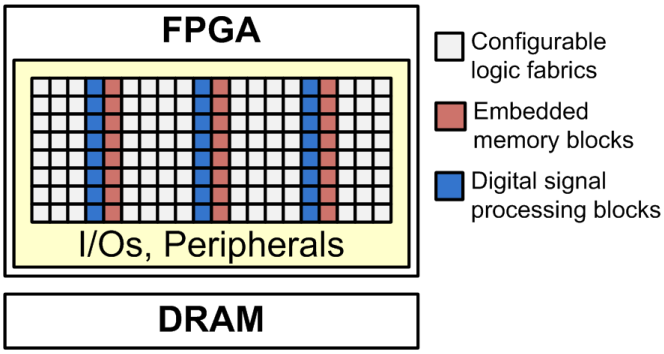


Figure 1.8: The illustration of FPGA architecture.

have taken the multi-core trend to the extreme, by integrating thousands of cores into a single chip. For example, the Geforce GTX 1080 contains 2,560 cores with a base clock of 1.6 GHz [14]. Moreover, the large amounts of integrated ALU resources inside all cores makes GPUs more efficient than CPUs for algorithms, where the processing of large blocks of data is done in parallel. The use of GPUs for accelerated computing has already been explored and applied in many advanced applications.

1.3.3 FPGA-Based Platform

FPGA is a reconfigurable chip that allows users to rewire it to implement a specific functionality by using a hardware description language (HDL) language, such as Verilog or VHDL. Its architecture is completely different from a CPU and a GPU, as illustrated in Figure 1.8. Modern FPGAs contain not only hundred thousands of configurable logic fabrics, thousands of digital signal processing blocks and embedded memory blocks, but also a wide variety of high-speed I/Os and peripherals, such as DDR3 SDRAM controllers and PCI Express 3.0 [15]. For those reasons, users can exploit FPGAs to construct various massive parallel processing systems for compute-intensive tasks, namely big data analytics and artificial intelligence [16, 17, 18, 19].

In addition to FPGA, application specific integrated circuits (ASIC) are also designed for increasing the computing efficiency, such as an Oracle SPARC M7 processor [20]. ASIC is a custom designed integrated circuit used for a particular application, rather than intended for general-purpose application. The architec-

	Advantages	Disadvantages
CPU	– Software implementation, wide range of tools for programming/debugging – No special interface is required	– Sequential processing, non-deterministic execution time due to the variation of many dependencies – High I/O cost – High power consumption
GPU	– Software implementation, some tools for programming/debugging – Parallel processing, fast execution time – No special interface is required	– Medium I/O cost – Very high power consumption
FPGA	– Massive parallel and pipeline processing, extremely fast execution time – Low data I/O cost – Low power consumption	– Hardware implementation (circuit-level), limited tools for programming/debugging – Special interface to high-level software is required

Table 1.2: The summary of platform approaches.

ture of an ASIC is similar in function to an FPGA, except for the ability of reconfiguration. The program is actually wired onto a piece of silicon and is packaged into a chip. For instance, at 90-nm technology, the ASIC designs can operate as fast as threefold to fourfold, whereas the dynamic power consumption ratio is approximately 14 fold as small as the FPGA designs [21]. However, because of high startup costs, ASICs are only considered when large volumes are needed.

The summary of the three platform approaches regarding performance and implementation is shown in Table 1.2. Despite the difficulty in implementation, FPGA is capable of delivering higher performance, while consuming much lower power than CPU and GPU [22].

1.4 Motivation and Key Contributions

Based on the background above, there are two major findings:

- BI allows multi-key queries to be effectively answered by using simple bit-wise logical operations. BI also supports parallel processing and efficient memory access, thereby reducing both computation and I/O cost. Figure 1.9 shows the performance of a BI-based data-analytics software called Fast-bit, compared to a DBMS using B-tree. It is noted that B-tree is similar to B^+-tree, except that not only leaf nodes but also root node and internal nodes can point to the records. Various experiments in a high-energy physics dataset proved that the BI-based Fastbit could operate at least 8× faster than a B-tree-based DBMS.

- FPGAs and ASICs are considered to be prominent candidates for an analytics system because they can not only achieve higher computing efficiency but also consume less energy than both CPUs and GPUs. Figure 1.10 depicts an example of the performance/power advantage of FPGA in certain classes of server applications, as compared to a CPU and GPU [23]. The authors compared the search algorithm acceleration capabilities of a Xilinx Kintex-7 FPGA, an Intel Xeon E5620 2.4-GHz 16-core CPU, and a NVIDIA K10 GPU across a range of parallel-thread scenarios. The authors found that a mid-range Kintex-7 FPGA could process around 3.5× and 2.5× faster than CPU and GPU, respectively. More significantly, FPGA consumed less than 20 W, which is about 4× and 22× lower than that of the 80-W CPU and 225-W GPU, respectively.

Although many studies exploited the parallel processing capability of CPU [32, 33, 34, 35] and GPU [39, 40] for fast BI-based index and query tasks, research on the feasibility of a BI-based analytics system using FPGAs and ASICs has not been developed yet. This dissertation, therefore, aims to bridge this gap by originally proposing such system in an FPGA, and later in an ASIC for power reduction.

Three components named bitmap index creator (BIC), bitmap-index-based query processor (BIQP), and bitmap index encoder (BIE) are first proposed, as

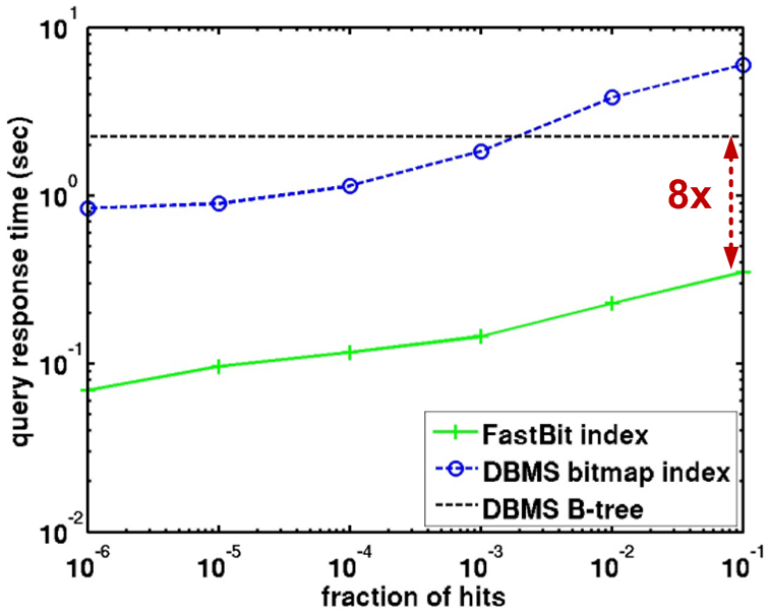


Figure 1.9: The performance of BI-based Fastbit versus a DBMS [10, 11].

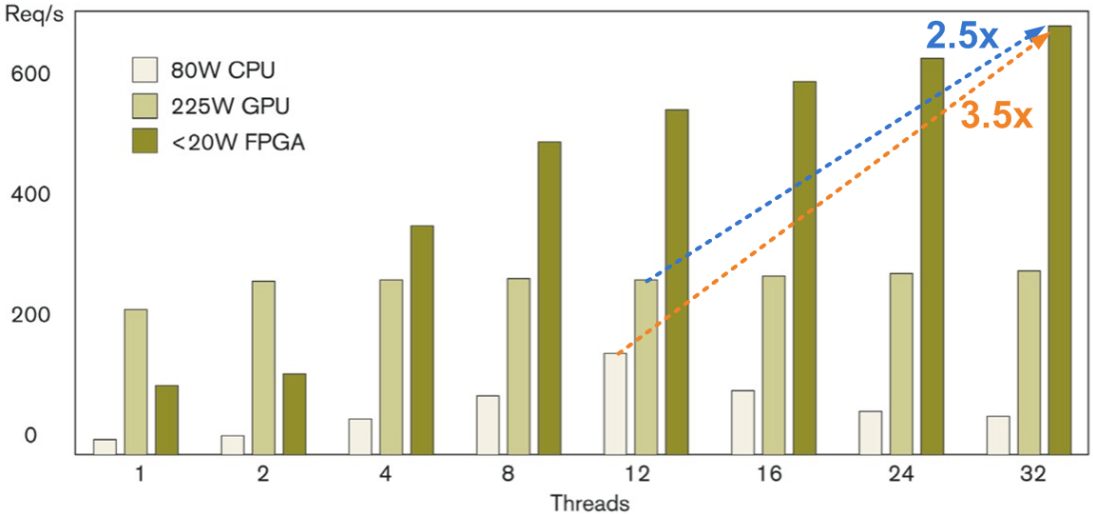
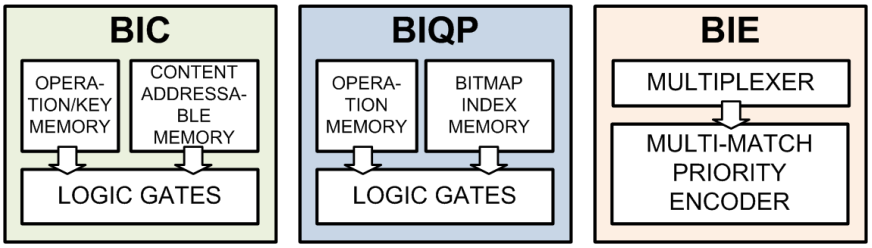


Figure 1.10: The performance/power benefit of predictive search algorithms on a FPGA versus on a CPU and GPU [23, 24].



(a) Bitmap index creator. (b) Bitmap-index-based query processor. (c) Bitmap index encoder.

Figure 1.11: The architecture of BIC, BIQP, and BIE.

shown in Figure 1.11. The overview of each component is summarized as follows:

- **BIC**: indexes input data by given keys. It is composed of an operation/key memory, a content-addressable memory, and a set of logic gates used for range indexing, as illustrated in Figure 1.11(a). Two different BICs are designed to be able to index 8-bit and 16-bit input words. With each given key, BIC produces a N-bit BI vector, where N corresponds to the number of input words. Both of them are fully operational at 100 MHz in an Arria V FPGA and surpassed other CPU-based and GPU-based designs. Furthermore, a set of equations is introduced to be able to quickly predict the indexing throughput.

- **BIQP**: answers the given queries by using the given BI vectors. It consists of an operation memory, a BI memory, and a set of logic gates used for query solving, as depicted in Figure 1.11(b). The outcome of BIQP is a N-bit bitmap result, whose length is the same as the input BI vectors. BIQP is also fully operational at 100 MHz in an Arria V FPGA and surpassed other CPU-based designs. A set of equations is also discussed to be able to quickly estimate the query time. The simulation and measurement results of BIQP in 180-nm bulk CMOS and 65-nm Silicon On Thin Buried Oxide (SOTB) CMOS [25, 26, 27, 28] also confirm the ASIC feasibility.

- **BIE**: encodes the bitmap results into matching positions. For example, if the result is 11010100_2, the output will become zero, one, three, and

five, which corresponds to the position of all bit ones. BIE is mainly built by a special multi-match priority encoder and a multiplexer, as illustrated in Figure 1.11(c). This encoder employs a so-called one-dimensional-array to two-dimensional-array conversion and a look-ahead signal to cope with large-sized input data effectively. The experimental results on various FPGAs and an 180-nm bulk CMOS proves the advantage of this work over other state-of-the-art studies.

Each of three components can function independently or can be put together to form an index creation system, query processing system, and data-analytics system, as shown in Figure 1.12. Those systems are verified in an Intel Arria V development kit, which includes a mid-range Arria V System-on-Chip FPGA and high-speed external DDR3 memory. The overview of three systems is summarized as follows:

- **BI creation system**: employs a BIC to index a large amount of data of each relation in parallel, as shown in Figure 1.12(a). Two systems are then proposed: (1) BI creation system without an encoder: BIE is not connected to BIC, so all generated BIs are saved directly to the external memory for further processing by either BIQP or other DBMSs; (2) BI creation system with an encoder: BIE is connected to BIC, so only the matching positions of BI vectors, or encoded values, are stored in the memory.

- **BI-based query processing system**: necessary BI vectors are sent to BIQP for query processing, as shown in Figure 1.12(b). Those indexes can be generated in advance, either by BIC or other DBMSs. Two systems are then proposed: (1) Query processing system without an encoder: BIE is not connected to BIQP, so all bitmap results are saved directly to the external memory for further processing; (2) Query processing system with an encoder: BIE is connected to BIQP, so only matching positions of the bitmap results, or encoded results, are stored in the memory.

- **BI-based analytics system**: index and query are performed sequentially by BIC and BIQP, respectively, as shown in Figure 1.12(c). Two systems are then proposed: (1) BI-based analytics system without an encoder: BIE

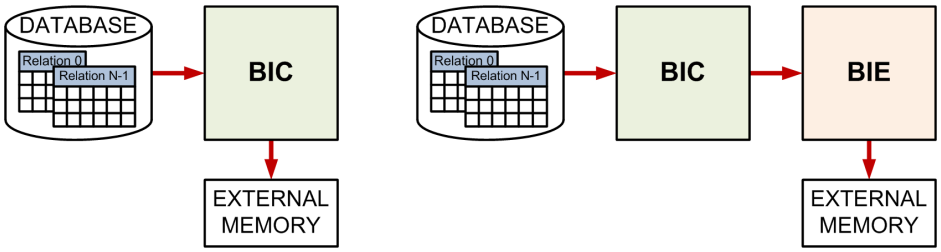
is not connected to BIQP, so all query results are saved directly to the external memory for further processing; (2) BI-based analytics system with an encoder: BIE is connected to BIQP, so only matching positions of the query results are stored in the memory.

The architecture of BIC, BIQP, and BIE will meet the criteria of scalability and parallel processing capability. Meanwhile, the performance of BIC, BIQP, and BIE will be evaluated by data I/O cost, processing throughput, and energy efficiency.

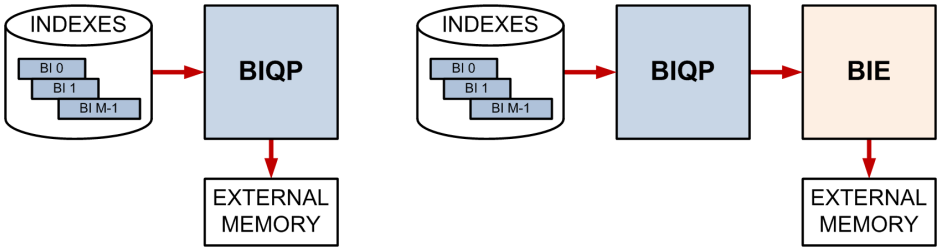
- **Scalability**: is defined as the ability of the system to handle a growing workload in a capable manner or its ability to be enlarged to accommodate that growth. In other words, scalability is considered to be the ability to add more hardware—scale up or scale out—to improve the capacity and performance of a system. For example, two BICs that can index 65,536 8-bit words and 32,768 16-bit words are made by simply changing the memory size and the number of logic gates. Likewise, BIQP depth and width are also adjusted by using additional memory blocks and logic circuits. Finally, BIE architecture can deal with a wide range of input data effectively, from 32 bits to 4 Kbits.

- **Parallel processing**: is defined as the ability of the system to perform several tasks in parallel, thereby helping to improve the entire processing throughput as well as reduce the whole latency. For example, all three components can process data in parallel at the speed of clock cycle: BIC indexes as many as 32,768 16-bit words at the rate of one key/32,768 words/cycle; BIQP processes each 32-Kbit BI in parallel at the rate of one operation/32-Kbit BI/cycle; BIE detects and encodes the matching positions in a 4-Kbit bitmap data at the rate of one match/4-Kbit data/cycle.

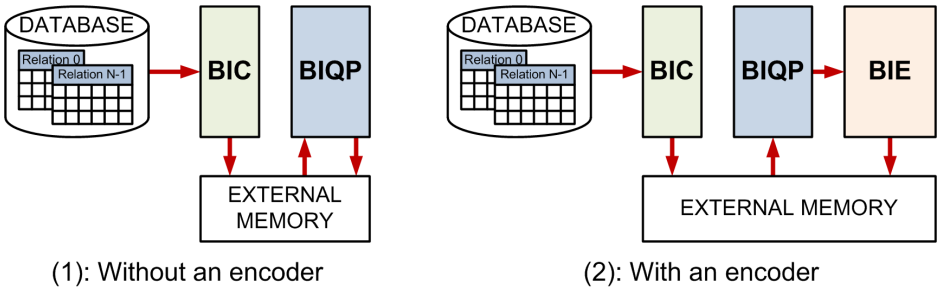
- **Data I/O cost**: is defined as the rate at which data is transferred from and to a peripheral device. Data I/O cost can be viewed as the rate at which the data is read and written to the memory (or disk) or the data transfer rate between the nodes in a cluster. For example, both BIC and BIQP can access the data and operations directly from the external memory



(a) The demonstration of a BI creation system.



(b) The demonstration of a BI-based query processing system.



(c) The demonstration of a BI-based analytics system.

Figure 1.12: The utilization of BIC, BIQP, and BIE.

using direct memory access mechanism. Therefore, the I/O performance is likely to reach as high as the memory bandwidth. In other words, both components can fully exploit the 25.6-Gbps DDR3 memory integrated in the FPGA development board.

- **Processing throughput**: is defined as the number of data that can be processed in a time unit. Throughput is particularly important in the systems whose results must be strictly answered within certain time constraints. The throughputs of all three components at 100-MHz operating frequency are listed as follows. BIC indexes 16-bit data in parallel at the throughput of 1.48 GB/s; BIQP processes 32-Kbit BI vectors in parallel at the throughput of 2.45 GB/s; BIE encodes 2,048-bit data in parallel at the maximum and minimum throughput of 66.34 Gb/s and 95.23 Mb/s, respectively.

- **Energy efficiency**: is defined as the ability to deliver the same processing throughput, but consume less power. For instance, the conventional processors like CPUs consume a large amount of energy and cannot be optimized to suit the target applications. On the other hand, GPUs are programmable, but consume an even higher amount of energy. FPGAs offer a middle ground among the platforms with high energy efficiency without sacrificing the throughput of the application. Therefore, BIC, BIQP, and BIE implemented in an FPGA will be proven to achieve better energy efficiency than those implemented in CPUs and GPUs.

1.5 Dissertation Layout

The dissertation is divided into seven chapters, whose layout is depicted in Figure 1.13.

- Chapter 2 presents the literature review of data-analytics systems and encoding methods.

- Chapter 3 describes in detail the architecture of a BIC that is mainly based on an enhanced content-addressable memory and a set of logic gates.

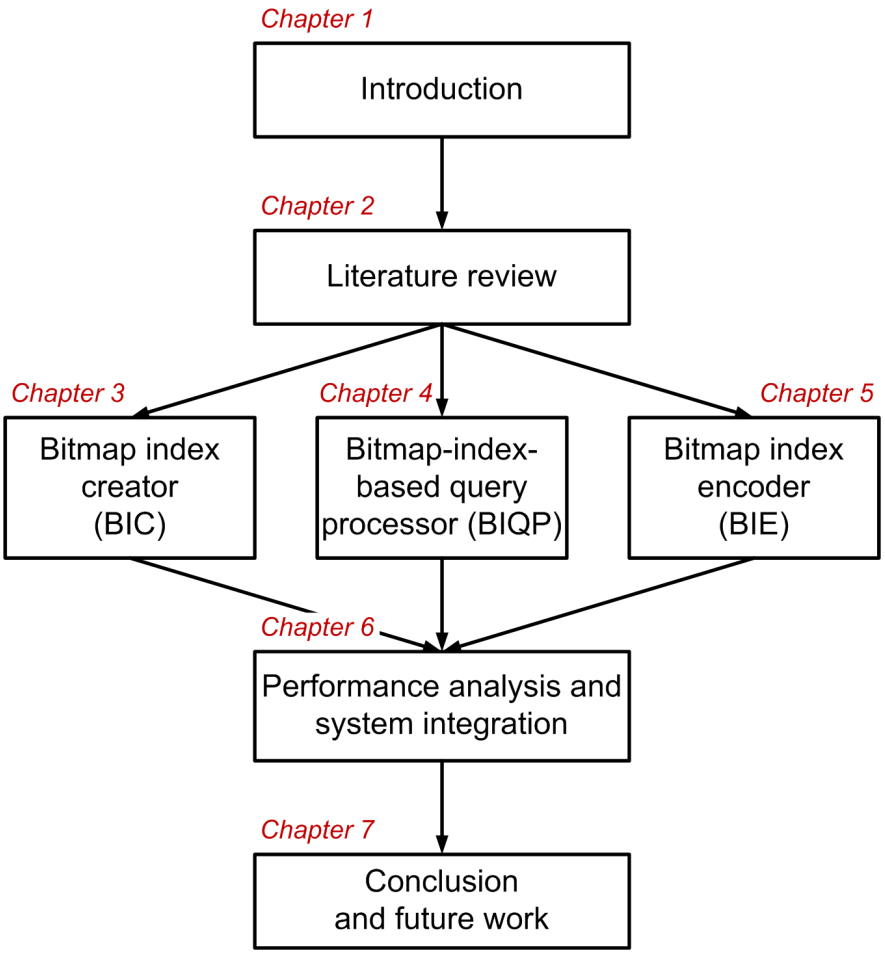


Figure 1.13: The dissertation layout.

- Chapter 4 focuses on the architecture of a BIQP that is primarily based on a cascade BI memory and a set of logic gates.
- Chapter 5 provides a detailed description of a BIE using the proposed 1D-array to 2D-array conversion method and a look-ahead signal.
- Chapter 6 evaluates the performance of each component and their integration in a data-analytics system.
- Chapter 7 summarizes the primary results and identifies a list of open topics for the future research.

Chapter 2

Literature Review

The purpose of this chapter is to review the motivation, methodologies, and achievements of several recent works. The first section focuses on CPU-based, GPU-based, and FPGA-based approaches to the data index and query process. The last section focuses on FPGA-based approaches to bitmap encoding, which are mainly based on a priority encoder and a multi-match priority encoder.

2.1 Approaches to Index and Query

Most studies on efficient data analytics focus on tree index, hash index, and BI, as well as their implementations in the CPUs, GPUs, and FPGAs. Figure 2.1 illustrates some recent works, whose descriptions are shown below.

2.1.1 CPU-Based Approaches

Several CPU-based data analytics systems were reviewed as described below.

[1]: Z. Li et al. [29] proposed an efficient index called IR-Tree for geographic document search, such as *"Boston's hotels and bars reviews"* query. The proposed search algorithm can filter both spatial and textual information, perform relevance computation, and rank the results. All functions were written by C++ and implemented in Microsoft Windows Server 2003 operating on a Intel Xeon 2.0-GHz CPU with 8-GB memory. Compared to two previous works, i.e. KR*-tree and Hybrid$_R$, the search time of IR-tree in two real document sets was

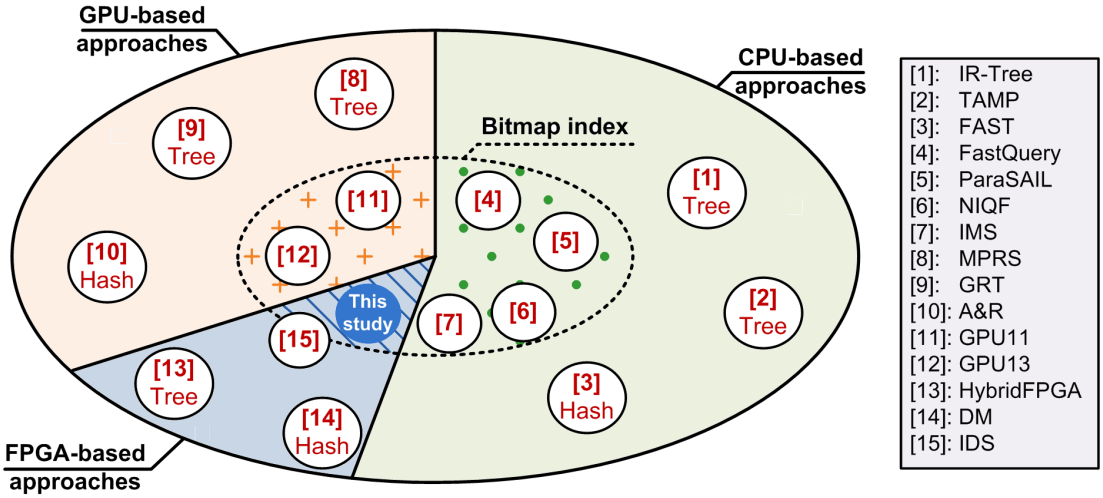


Figure 2.1: The summary of approaches to data analytics.

approximately 20% and 50% faster, respectively.

[2]: H. Wang et al. [30] presented a new framework that could effectively process queries in a big data warehouse. The framework is called TAMP, which stands for transform, aggregate, merge, and post-process. It was then deployed on Hadoop distributed on a cluster of 14 Linux computer nodes, each of which contains a Intel Core2Duo 1.87-GHz CPU with 2-GB memory, to evaluate the scalability and performance. The Star Schema Benchmark proved that the performance of Hadoop-based TAMP was 8× and 13× as fast as that of the original Hadoop and HadoopDB, respectively.

[3]: Y. Hua et al. [31] introduced a novel near-real-time semantic search called FAST for analyzing massive datasets. This methodology exploited the correlation property within and among datasets using improved correlation-aware hashing and flat-structured addressing to significantly reduce the query latency, while incurring an acceptably small loss of accuracy. The proposed design was written by C in the Linux environment. All experiments were then conducted in a cluster of 256 computer nodes, each of which contained 32-core CPU and 64-GB memory. The 200-TB real datasets including 60 million images were collected from the cloud. Compared to the previous work, FAST was more than 300× faster, whereas the accuracy was only 0.0033% lower.

[4]: J. Chou et al. [32] proposed a novel BI-based software framework called

FastQuery that indexes and queries massive datasets using modern supercomputing platforms. Due to the BI usage, data indexing and query processing procedures could be effectively distributed in a large numbers of many-core processors. All of the experiments were then conducted in a Hopper supercomputer, which consists of 6,500 computer nodes. In each node, an AMD 12-core 2.1-GHz CPU and 2-GB memory were employed. The results showed that the indexing throughput of a 50-TB dataset reached as high as 6.1 GB/s and 15.7 GB/s when using 2,304 cores inside 192 CPUs and 11,520 inside 960 CPUs, respectively. Since the power consumption of each CPU is 80 W, 192 CPUs and 960 CPUs usage required 15,360 W and 76,800 W, respectively. As a result, the corresponding energy efficiencies were 2,477 J/GB and 4,891 J/GB. With the calculated BIs, FastQuery was able to process all queries in 12 seconds using 2,880 cores. The overall throughput and power of index and query tasks, therefore, were similar to that of the index task itself.

[5]: T. Zhong et al. [33] presented a BI creation method called ParaSAIL that focused on maximizing the speedup from parallel execution by avoiding write-write and write-read cacheline conflicts among CPUs. As a result, ParaSAIL is claimed to be well-suited to systems with high numbers of cores, such as mainstream multi-core processors and many-core coprocessors (Co-CPU). All experiments were conducted in two Intel Xeon E5-2680 2.7-GHz 8-core CPU and in a Intel Xeon Phi 60-core 1-GHz Co-CPU card. An evaluation using data from an oceanographic dataset showed that ParaSAIL indexed 108 MB/s and 473 MB/s using CPU and Co-CPU, respectively. Due to the power consumption of 130 W and 225 W, the corresponding energy efficiencies of CPU and Co-CPU were 1,232 J/GB and 487 J/GB, respectively.

[6]: Y. Liu et al. [34] employed NIQF—a word-aligned hybrid bitmaps framework—to index and query massive data collected from the New Vacuum Solar Telescope. In addition to the framework, several simple and useful application programming interfaces were also designed for data manipulation and system integration. All experiments were then conducted in a Intel E7200 2.53-GHz CPU with 4-GB memory. Using the 1.6-GB synthetic data, NIQF could process all multi-conditional AND queries at a throughput of 107.5 MB/s, which surpassed MySQL operations. Since the CPU power consumption is 65 W, the

corresponding energy efficiency was 619 J/GB.

[7]: C. H.-Te et al. [35] presented an in-memory query system (IMS) that combined bitmap indexing, spatial data layout reorganization, distributed shared memory, and location-aware parallel execution. The real datasets were obtained from a 750-GB plasma physics simulation. All experiments were conducted in a Edison supercomputer containing 5,576 compute nodes. In each node, two Intel Ivy Bridge 12-core 2.4-GHz CPUs and 2-GB memory are utilized. The indexing tasks employed 1,250 cores and 20,000 cores to generate 150-GB indexes at the throughput of 28 GB/s and 510 GB/s, respectively. Since each CPU consumes 115 W, the corresponding energy efficiencies were 213 J/GB and 188 J/GB, respectively. The query throughput on 150-GB indexes reached 10.1 GB/s and 42.5 GB/s using 5,000 cores and 20,000 cores, or the energy efficiencies reached 2,368 J/GB and 2,254 J/GB, respectively.

2.1.2 GPU-Based Approaches

Several GPU-based data analytics systems were reviewed as described below.

[8]: J. Kim et al. [36] proposed a novel traversal algorithm called Massively Parallel Restart Scanning (MPRS) for multi-dimentional queries on the GPU. This algorithm effectively addressed two primary drawbacks of R-tree indexing structure including irregular memory access patterns and recursive back-tracking function calls. All experiments were then conducted on a NVIDIA Tesla M2090 GPU and an AMD 8-core 2.0-GHz CPU with 64-GB memory. The real datasets were obtained from the national climate data center. Using the GPU, MPRS could process 56,000 queries per second, which reached 364× higher throughput than the multi-threaded R-tree implemented in the CPU.

[9]: M. Alam et al. [37] introduced a GPU-based Adaptive Radix Tree (GRT) for high-performance data search in an in-memory database. GRT, together with the state-of-the-art Adaptive Radix Tree (ART) and Fast Architecture Sensitive Tree (FAST), were conducted in a NVIDIA Tesla K80 GPU for the performance comparison. The sparse keys and dense keys were correspondingly extracted from the music and book datasets. For the lookup throughput of the sparse keys, GRT was 10.6× and 2.1× higher than ART and FAST, respectively. Moreover, when

testing with the dense keys, GRT throughput was 8.3× and 1.6× higher than ART and FAST, respectively.

[10]: H. Pirk et al. [38] provided a generic strategy for efficient CPU/GPU cooperation. This strategy calculates an approximate result based on lossy compressed, GPU-resident data and refines the results using residuals on the CPU. To achieve this goal, a set of approximate and refine (A&R) operators using the hash technique was designed. All experiments were then conducted in two NVIDIA Geforce GTX 680 GPUs. Compared to the standard MonetDB implemented in a Intel Xeon E5-2650 16-core 2.0-GHz CPU, the GPU-based A&R could achieve up to 8× performance improvement, even for datasets larger than the available GPU memory.

[11]: W. Andrzejewski et al. [39] utilized GPU to enhance the BI compression algorithm called Position List Word Aligned Hybrid (PLWAH). Although the BI size is relatively smaller than other indexing methods, it becomes too large for wide domains. A BI compression algorithm, such as PLWAH, therefore, is proposed to reduce not only the BI size but also the I/O transfer time. To parallelize compressing and decompressing steps of PLWAH as well as to perform bitwise operations on the compressed bitmaps, GPU was exploited instead of CPU. All experiments were then conducted in a NVIDIA Geforce 285 GTX GPU and a Intel Core i7 2.8-GHz CPU for the performance comparison. GPU-based PLWAH were approximately 17.5× and 6.1× faster than CPU-based PLWAH in the BI compression and decompression task, respectively. The experimental results also proved that the query throughput on the two 120-MB BIs reached 0.1 GB/s and 0.65 GB/s. Because the power consumption of CPU and GPU are 95 W and 204 W, the corresponding energy efficiencies were 950 J/GB and 313 J/GB, respectively.

[12]: F. Fusco et al. [40] presented the algorithms for building compressed BI in real time on GPUs, which targets the multi-10-Gbps network traffic recorders. Word Aligned Hybrid (WAH) and PLWAH algorithms were applied to compress the BIs. To evaluate the performance, the authors used a Intel Core i7-2600K 3.4-GHz CPU and a NVIDIA Geforce GTX 670 GPU. Regarding the indexing throughput, GPU-based WAH reached a 20× speedup over CPU-based WAH. Furthermore, GPU-based PLWAH could achieve the indexing throughput of up to

336 MB/s, which was well suited for multi-10-Gbps packet indexing requirement. Due to the power consumption of 170 W, the corresponding energy efficiency was 518 J/GB, respectively.

2.1.3 FPGA-Based Approaches

Several FPGA-based data analytics systems were reviewed as described below.

[13]: D. Heinrich et al. [41] proposed a hybrid B^+-tree index structure in a semantic web database system. In the proposed structure, the lower levels of the B^+-tree, especially the leaves where the values are stored, are located on the host system, while the root and the most of the upper levels with the interior nodes are stored on the FPGA. The search in the upper levels of the hybrid index was processed in parallel by applying an FPGA. All experiments were conducted in a Xilinx Virtex-6 FPGA and a Intel Xeon E5-1600 3.0-GHz CPU for the performance comparison. Depending on the configurations, the computation time of the FPGA-based system was up to 2.3× faster than that of the CPU-based system.

[14]: K. Agarwal et al. [42] introduced a high performance and scalable hardware accelerator of dictionary matching on very large dictionaries for text analytics applications. This accelerator employed a novel hashing-based approach in order to process a string token per clock cycle. The experimental results on an Altera Stratix IV FPGA proved that the proposed design could process typical document streams at a processing rate of approximately 12 Gbps, while simultaneously allowing support for large dictionary sizes containing up to 100K patterns. As a result, the information extraction workload could be significantly accelerated.

[15]: D.-H. Le et al. [43] presented a content-addressable memory based information detection system (IDS) in an FPGA for fast, exact, and approximate image matching on two-dimensional data. The proposed system can be potentially applied to fast image matching with various search patterns, without using search principles. Because the content-addressable memory was constructed by dual-port memories, IDS architecture was partially similar to BIC. Nonetheless, IDS did not effectively solve the problem of loading data to a content-addressable

	Platforms	Throughput (GB/s)		Energy (J/GB)	
		Index	Query	Index	Query
[4] FastQuery [32]	11,520 cores/960 CPUs	15.66	–	4,891	–
	2,304 cores/192 CPUs	6.20 *(sum)*		2,477 *(sum)*	
[5] ParaSAIL [33]	8 cores/1 CPU	0.105	–	1,232	–
	60 cores/1 Co-CPU	0.46	–	487	–
[6] NIQF [34]	4 cores/1 CPU	0.105 *(sum)*		619 *(sum)*	
[7] IMS [35]	20,000 cores/834 CPUs	510	–	188	–
	20,000 cores/834 CPUs	–	42.5	–	2,254
[11] GPU11 [39]	4 cores/1 CPU	–	0.10	–	950
	240 cores/1 GPU	–	0.65	–	313
[12] GPU13 [40]	1,344 cores/1 GPU	0.33	–	518	–

Table 2.1: The summary of state-of-the-art works.

memory and encoding the bitmap result. IDS was also not suitable for multi-dimensional analytics targets. The experimental results on an Stratix III showed that querying a 32×32 gray image in a 128×128 gray image required 0.122 ms at frequency of 50 MHz.

In summary, several recent studies on information detection using a content-addressable memory and data analytics using tree index, hash index, or BI that implemented in CPU, GPU, or FPGA have been surveyed. Although CPU-based data analytics approaches are very common, there has been an increasing number of GPU-based and FPGA-based approaches recently, because of their benefit of parallel processing [36, 38, 39, 40, 41]. For BI, there is a lack of study of a BI-based analytics system in FPGA so far. Therefore, the achievements of the BI-based analytics systems in CPUs and GPUs, as summarized in Table 2.1, will be considered as the state-of-the-art works in this dissertation. The quotient of power consumption (J/s) and processing throughput (GB/s), which is called as the energy efficiency, is used to evaluate the performance. The small value of energy efficiency of a certain system means that this system delivers high throughput but only consumes low power. In other words, if a system possesses high value of energy efficiency, such system possibly requires high power in order to deliver the certain throughput.

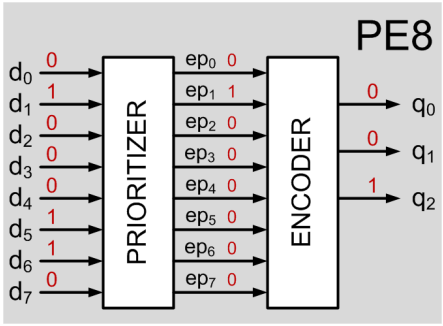


Figure 2.2: The block diagram of PE8.

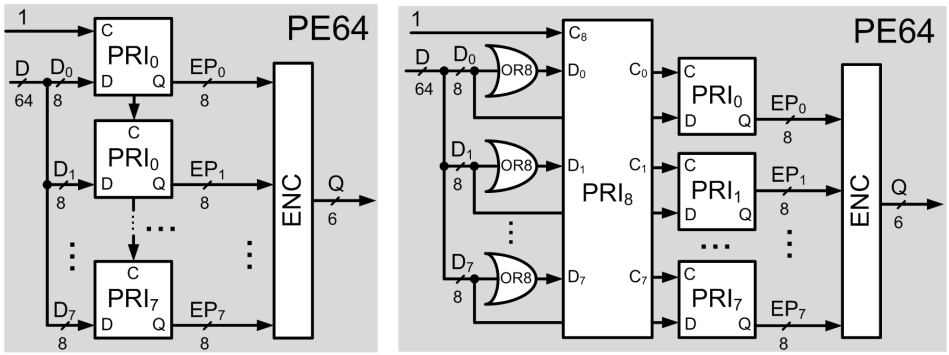
2.2 Approaches to Bitmap Index Encoder

One prominent solution for the BIE is to employ a multi-match priority encoder. A multi-match priority encoder is a special encoder that can output all positions of all match bits inside the bitmap input. It is usually composed of two primary components, including a preprocessing module and a priority encoder. A priority encoder is used to resolve the highest priority match and output the matching location into binary format, from which corresponding data inside memory are retrieved accurately.

2.2.1 FPGA-based Approaches

Most previous works considered a priority encoder as the combination of two primary modules, which are called PRIORITIZER (PRI) and ENCODER (ENC). Figure 2.2 depicts an 8-bit priority encoder (PE8) with the input d of 0100110. Since d_1 is the highest priority bit, the output ep and q would become 01000000 and 001, respectively.

In order to handle larger input data, several PE8s are connected together. Figure 2.3(a) illustrates a conventional architecture of a 64-bit priority encoder (PE64) containing a set of 8-bit PRIs and 64-bit ENC. PRI_0 to PRI_7 are connected in serial, and each of them is controlled by the input enable C from the former PRI. Initially, 64-bit input data is split into eight 8-bit groups. Each PRI resolves the highest priority bit of each group, while ENC outputs a matching location into binary format. For instance, if D_0 is 01001110, EP_0 and Q become 0100000



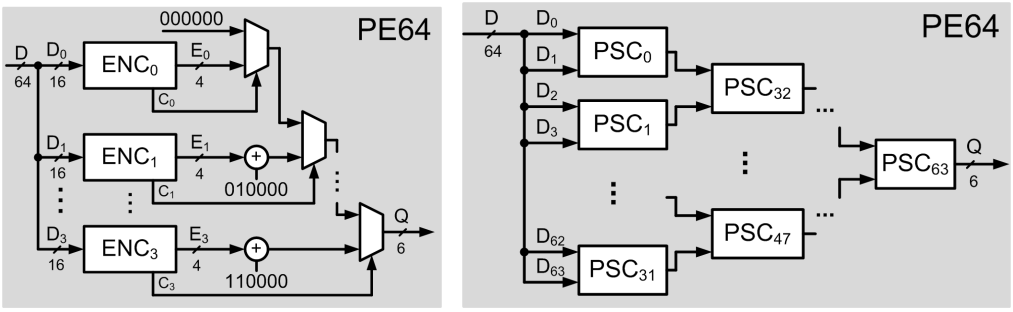
(a) Based on serial architecture. (b) Based on parallel architecture.

Figure 2.3: The block diagram of PE64s.

and 000010, respectively. Because all PRI modules are connected in series, the worst latency of PE64 is about eight times as high as that of one PRI.

To reduce such latency, C.-H. Huang et al. [44] presented multi-level lookahead and multi-level folding techniques. By remapping all control signals, the performance was improved up to ten times. However, this mapping strategy became increasingly complicated as priority encoder size went up. Figure 2.3(b) depicts a parallel priority look-ahead architecture, which was initially introduced by C. Kun et al. [45] and then was applied in ternary content-addressable memory [46]. With this architecture, PRI_0 to PRI_7 can return their priority matches in parallel due to the control signal provided by PRI_8. Despite decreasing the latency, the resource utilization increases because of the additional PRI_8 and logic gates. Another improvement from D. Balobas et al. [47] exploited a new design of a 4-bit priority encoder (PE4) and a static-dynamic parallel priority lookahead architecture to boost the performance of PE64. However, the architectures of large-sized priority encoders were not mentioned. Furthermore, S. Abdel-Hafeez et al. [48] presented a special prefix scheme for priority encoders whose size rises to 256-bit. Nevertheless, the performance declines sharply with increased priority encoder size.

Figure 2.4(a) shows the architecture of a PE64 based on four one-hot encoders, which was designed by D.-H. Le et al. [49]. Each ENC converts a corresponding 16-bit group into 4-bit position and a control signal C decides whether the results are passed to the next multiplexers. If it is assumed that the priority encoder



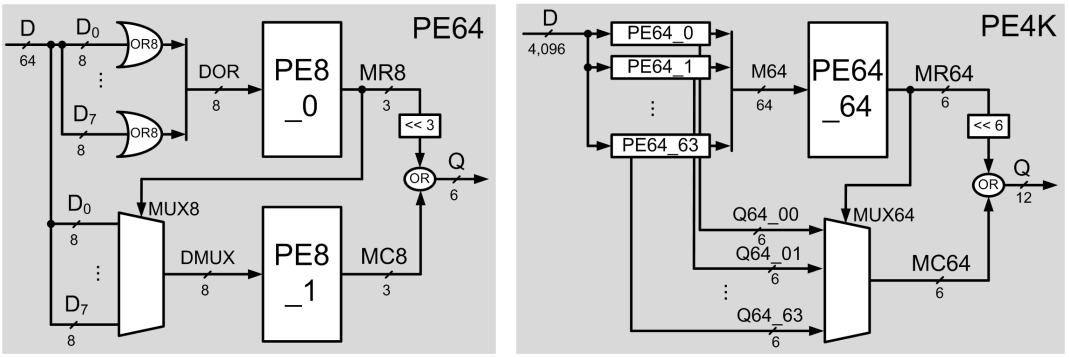
(a) Based on the one-hot encoders. (b) Based on the comparison and sort circuits.

Figure 2.4: The block diagram of PE64s.

size is 2,048 bits, then up to 128 ENCs connected in series would be required. A 2,048-bit multi-match priority encoder based on this priority encoder was also found in [49].

Figure 2.4(b) depicts another approach proposed by S. K. Maurya et al. [50], where a set of comparator and sort circuits are deployed to check each pair of bits of input data, so the highest priority bit is selected. If the priority encoder size is 2,048 bits, as many as 2,047 comparator and sort circuits connecting in 11 pipeline stages are demanded. In other words, those architectures are likely to result in large-scale resource consumption.

A novel architecture of an L-bit priority encoder using the one-dimensional-array to two-dimensional-array (1D-to-2D) conversion method was originally proposed in [51]. Figure 2.5(a) illustrates this method, where L-bit input data is converted into a $M \times N$-bit matrix, where M and N are the numbers of columns and rows, respectively. All bits of row status are obtained by performing the bitwise OR to all bits in the corresponding row. Subsequently, an N-bit priority encoder finds the highest priority bit i (row index) in the N-bit row status, and an M-bit priority encoder seeks the highest priority bit j (column index) in this row i. The matching position k of an 1D-array input is retrieved as $k = i \times M + j$. More significantly, if M is a power of two, the multiplier and adder are simply replaced by the fixed wirings that function as left-shift and OR operators. Similarly, a large-sized priority encoder such as a 4,096-bit priority encoder (PE4K) was built by 64 PE64s connecting in parallel and one central PE64, as depicted



(a) PE64. (b) PE4K.

Figure 2.5: The block diagram of PE64 and PE4K using 1D-to-2D conversion.

	Platform	(FREQ×L) (Gbps)			Resource (LE/Gbps)		
		8-bit	64-bit	2,048-bit	8-bit	64-bit	2,048-bit
[46]	Stratix FPGA	1.9	3.5	–	17.5	59.5	–
[49]	Cyclone IV FPGA	–	–	102.4	–	–	191.2
[51]	Stratix FPGA	2.9	6.1	–	7.9	34.9	–
	Cyclone IV FPGA	–	–	145.4	–	–	30.7

Table 2.2: The summary of state-of-the-art works.

in Figure 2.5(b). The detailed of this methodology can be found in Chapter 5.

In summary, several recent studies on priority encoder and multi-match priority encoder in FPGAs have been surveyed. These encoders are the primary component of a BIE. The achievements of related studies are summarized in Table 2.1 and will be considered as the state-of-the-art works in this dissertation. Suppose that FREQ and L are the operating frequency and the number of bits of MPE, respectively. The resource efficiency is the quotient of (FREQ×L) and logic elements (LEs) utilization of corresponding designs. The smaller the resource efficiency is, the more energy efficient a design becomes.

Chapter 3

Bitmap Index Creator

The purpose of this chapter is to propose a scalable hardware architecture of a BI creation accelerator that can index a large number of data in parallel. The output of this design is a set of BI vectors, which are used to answer the multi-dimensional queries by a query processor in the subsequent stage.

3.1 Introduction

In order to accelerate the query process, data are commonly indexed in advance. Because this dissertation addresses the problems of multi-dimensional queries, which are mainly used in analytics workloads, a BI is employed, instead of a tree index or a hash index. Furthermore, FPGA-based approach is targeted, instead of a CPU-based or a GPU-based approach, due to its computing efficiency. This chapter, therefore, focuses on a scalable hardware architecture of a BI creation accelerator, also called BI creator (BIC). The contributions are listed as follows.

- This chapter points out the relationship between the random-access-memory-based content-addressable memory (RAM-based CAM) and the BI. Using the concept of RAM-based CAM, a methodology to index data in parallel is then proposed.
- This chapter presents an enhanced cascade architecture of RAM-based CAM that significantly reduces both the reset time and load time of the

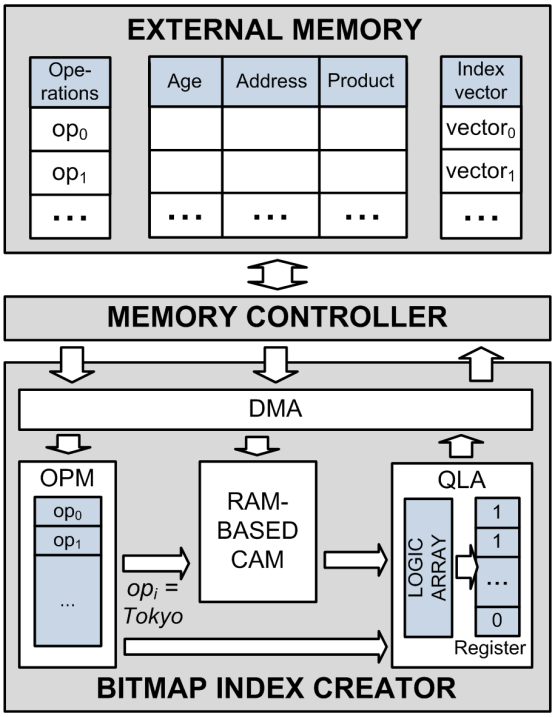


Figure 3.1: The block diagram of BIC.

original RAM-based CAM. Specifically, the reset time and load time go down 32×, if the external memory bus width becomes 256 bits.

- This chapter proposes a full design of BIC that can directly access the external memory to lower the I/O cost and produce the range indexes of up to 65,536 words in parallel. Moreover, BIC size is simply adjusted by adding or removing corresponding logic and memory resources. The content of this chapter was partially presented in some previous works [52, 53, 54, 55].

The remainder of this chapter is organized as follows. Section 3.2 describes in detail the hardware architecture of BIC. Section 3.3 concludes this chapter. The performance analysis of BIC is discussed in detail in Chapter 6.

3.2 Architecture

3.2.1 Overview

Figure 3.1 illustrates the block diagram of a BIC used to index three given attributes (columns), namely *Age*, *Address*, and *Product*, of a relation (table). All of the operations/keys are extracted from the queries and initially stored in an external memory together with all attributes. BIC is composed of four modules operating in parallel, namely direct memory access (DMA) module, RAM-based CAM module, operation memory (OPM) module, and query logic array (QLA) module.

To begin, all operations/keys are transferred to the OPM. Subsequently, CAM continuously receives the values of attribute *Age* until it becomes full. CAM then starts using the operations and keys from OPM to produce the correspondent BI vectors. Each separate vector is dispatched to QLA in turn, where an array of logic gates and an internal register are employed to calculate the range index. Lastly, the register values or range indexes are orderly stored in the external memory. This process repeats until all values of the current attribute are indexed. The next two attributes also follow the same process.

3.2.2 Direct Memory Access (DMA) Module

The three-channel DMA allows BIC to directly access to DDR3 through a memory controller. The current used FPGA provides two high-throughput memory controllers so that users can exploit the huge space of external memory. Because we are using DDR3 and its theoretical bandwidth is 25.6 Gbps, we configure the bus data width of 256 bits wide and operating frequency of 100 MHz. The DMA is properly designed so that BIC can input and output data at the rate of DDR3 theoretical bandwidth.

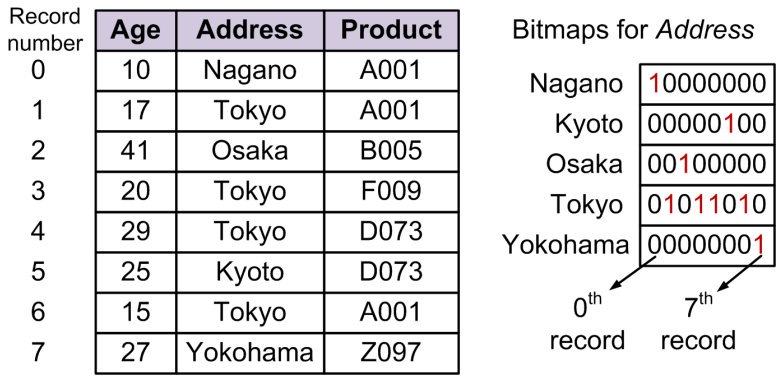


Figure 3.2: Bitmap index of an attribute *Address*.

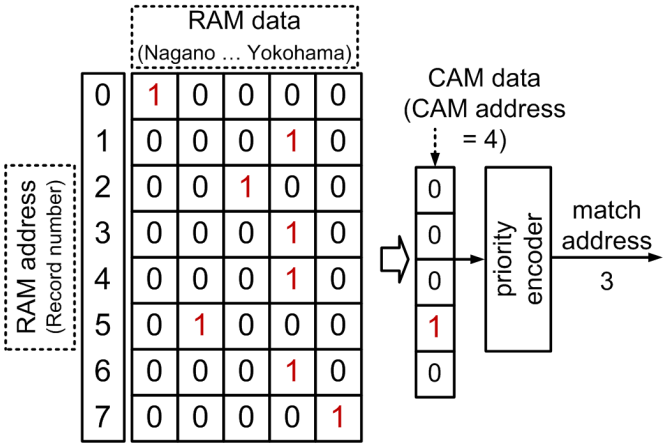


Figure 3.3: The block diagram of RAM-based CAM of *Address*.

3.2.3 Random-Access-Memory-Based Content-Addressable Memory (RAM-Based CAM) Module

3.2.3.1 Relationship between Bitmap Index and RAM-Based CAM

As mentioned earlier in Section 1.2.4, BI of an attribute *Address* is a bit-level matrix, as shown in Figure 3.2. The number of rows is four (the cardinality of attribute values) and the number of columns is eight (the number of records). The row j and column i of BI turns into one if $Address_i{=}j$ and vice versa. In addition, row j of the BI is also called a BI vector of value i of attribute *Address*.

CAM is a special type of computer memory that is applied in various search-intensive applications, such as multimedia processing, data analytics, and data

mining [56]. In contrast to RAM, each input and output of CAM are the content of data and address of matching data, respectively. Depending on different applications, either binary CAM or ternary CAM is used. The former only supports storage and searching binary bits (zero or one) while the latter allows a third matching state, so-called "don't care", in its storage. From now on, as only binary CAM is employed to construct BI, the term CAM represents binary CAM.

Although modern FPGAs provide a large number of embedded RAM blocks, dedicated registers, and lookup tables, they exclude dedicated CAM blocks presumably because of their disadvantages of area and power. Instead, FPGA vendors propose the methodology to construct a scalable CAM from the available embedded RAM blocks of FPGA by performing several mapping techniques to input data and address [57, 58]. Using those techniques, a RAM-based CAM of attribute *Address* is built as in Figure 3.3. Each row in the RAM represents one possible mapping of the input data bits to the CAM contents. Concretely, the value of each cell is set to one if the data is stored at that address, and vice versa.

Intuitively, RAM-based CAM is a transpose of the BI. In fact, RAM value at address j is equivalent to a BI vector j, or column j of BI matrix depicted in Figure 3.2. For this reason, the concepts from RAM-based CAM is borrowed to construct a BIC. To keep it short, the term CAM refers RAM-based CAM from now on.

3.2.3.2 Simplified Cascade Architecture of CAM

Although the architecture of embedded memory integrated in an FPGA varies with the FPGA vendors, a CAM is usually built from the dual-port memory (DPM). Concretely, the Intel Arria V FPGA exploits a DPM, where port A is an 8,192×1-bit memory and port B is 256×32-bit memory, to construct a 32×8-bit CAM, also named as a CAM unit (CU). One CAM bit, therefore, costs 32 RAM bits. We select these settings because 32×8-bit CAM is the most efficient CAM primitive that can be built from an M10K—a basic memory block unit of Arria V FPGA.

Figure 3.4(a) describes the architecture of a CU, whose input data *data* and addresses *addr* enter at port A while search data (*key*) and matching address

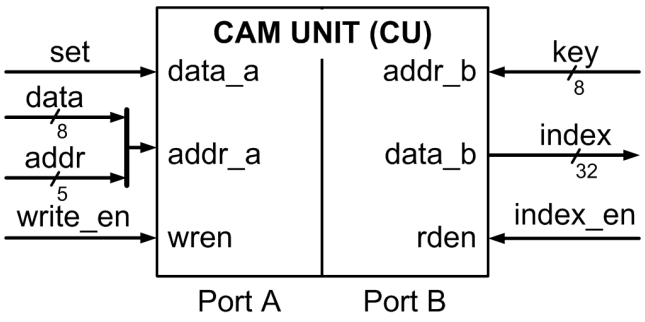
(*index*) leave at port B. Firstly, input data combines with input address as ((*data* << 5) OR *addr*, << is a shift left operation) to form the address of port A *addr_a*. Secondly, *set* is used as *data_a* of port A. If *set* turns into one, input data is indexed; otherwise, the available index is cleared. Because CAM must be fully reset before receiving new inputs, it takes two clock cycles for writing one value.

Several CUs are connected to increase the CAM size. An (32×N)×8-bit CAM is shown in Figure 3.4(b), where input data are put into each CU in turn from CU_0 to CU_{N-1}. As soon as CU_0 is full, CU_1 starts receiving data. The process repeats until all CUs are filled up. Another illustration of an 32×(8×M)-bit CAM is seen in Figure 3.4(c). Unlike the previous architecture, input data are divided into M segments and each of them enters each correspondent CU simultaneously. The output can be seen as 32 M-bit groups and each connects to an M-bit AND gate. This is because a match here is defined as all group bits matching the read input on the same address. With this configuration, high-cardinality values can be handled rapidly because of an array of AND gates, which obviously costs much time in software implementation. Additionally, a scalable (32×N)×(8×M)-bit CAM is obtained by combining two architectures above.

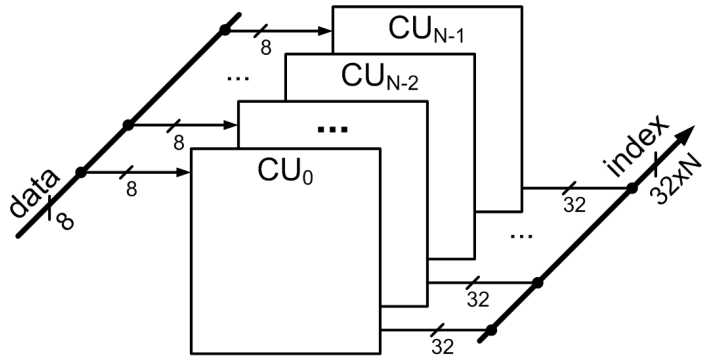
With the rich logic elements and M10K memory blocks, an Arria V FPGA can afford as large as 65,536×8-bit CAM (CAM64K8) or 32,768×16-bit CAM (CAM32K16). As a result of 8-bit word and 16-bit word, CAM64K8 and CAM32K16 can support cardinality of 256 and 65,536, respectively. Besides, write operations cost two clock cycles/word, whereas read operations cost one clock cycle/BI vector. A drawback of this architecture is the proportional increase in loading time and CAM size, e.g. 65,536×2 clock cycles is required to fill up CAM64K8. To reduce such waste time, we propose a new mapping way so that as much data can enter CAM in every clock cycle.

3.2.3.3 Enhanced Cascade Architecture of CAM

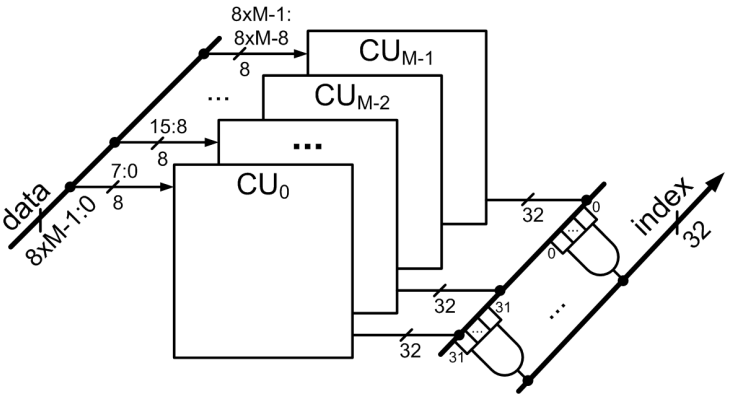
As mentioned earlier, the data bus between DMA and CAM is 256 bits wide. If each value of the attribute is eight bits long, DMA can transfer $\frac{256}{8} = 32$ values to CAM64K8 simultaneously in every cycle. Accordingly, CAM64K8 is capable



(a) 32×8-bit CAM.



(b) (32×N)×8-bit CAM.



(c) 32×(8×M)-bit CAM.

Figure 3.4: The block diagram of cascade CAM.

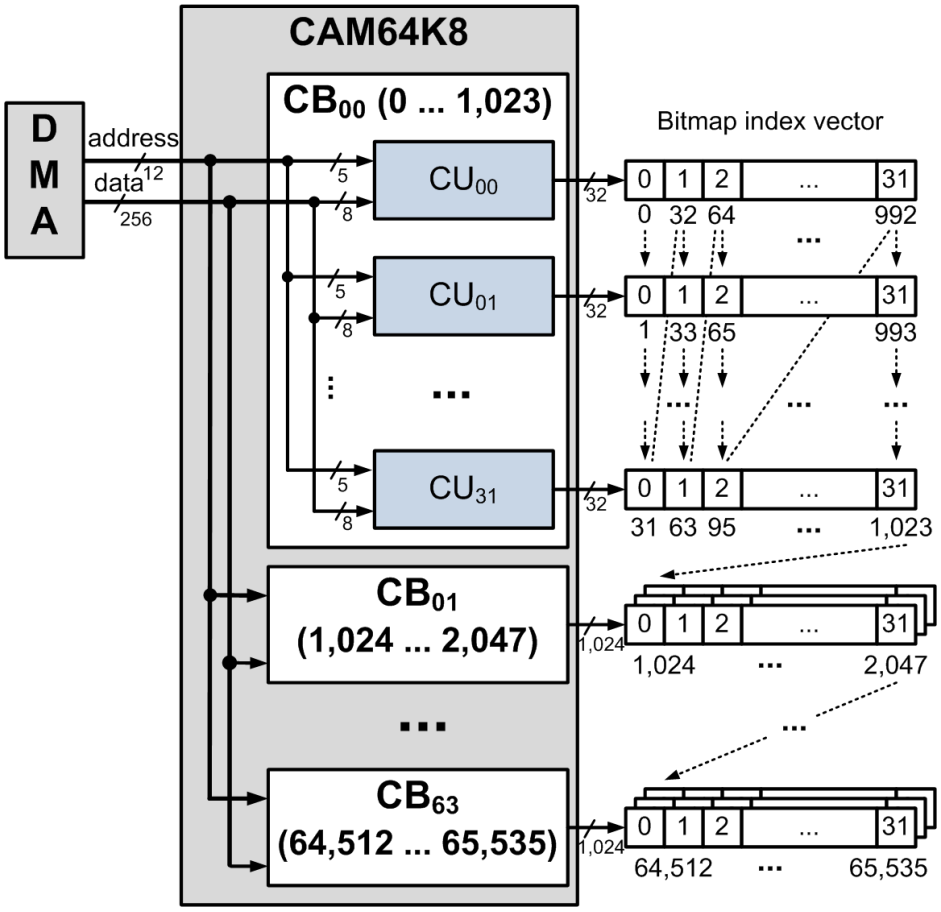


Figure 3.5: The enhanced architecture of a 65,536×8-bit CAM.

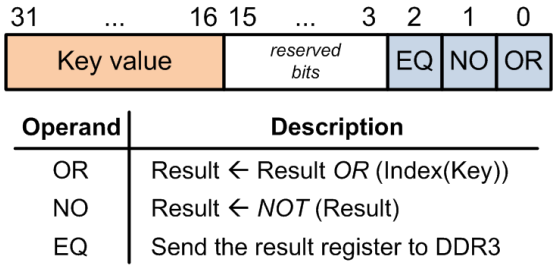
of loading all values within $\frac{65,536}{32}$ = 2,048 cycles. The achievement is done by an enhanced architecture of CAM64K8, where all inputs and outputs of CUs are grouped in a specific order. Figure 3.5 illustrates the architecture, where CAM64K8 is formed by 64 CU blocks (CBs) and each block consists of 32 CUs. The number of CUs in each CB is the quotient of data bus width and value size. The outputs are also arranged by a specific order. All zeroth bits of CB_{00} is formed by the first bit of BI vectors of {CU_{00}, ..., CU_{31}}. Likewise, all first bits of CB_{00} match the second bits of BI vectors of {CU_{00}, ..., CU_{31}} and so on. The 1-Kbit index vector of CB_{00} is obtained by those 32 small index vectors. As a result of this strategy, CAM64K8 can receive many characters simultaneously, while the order of its 64-Kbit BI vector is unchanged.

The loading process is summarized as follows. To begin, the first 32 values are put into the first 32 consecutive CUs of CB_{00} concurrently. As soon as CB_{00} is full, the incoming values are sent to the next group of CB_{01}. This process continues until all data are properly stored. Accordingly, the loading time is reduced up to 32 times as compared to the traditional structure. However, similar to simplified architecture, CAM64K8 has to be cleared before receiving new value. Hence, the write operation costs two clock cycles for loading 32 values to CAM64K8, or 2,048 clock cycles for filling entire CAM64K8. Additionally, the number of CUs in each group varies with the width of the data bus and the value size. For instance, in the case of CAM32K16, each CB only contains 16 16-bit CUs.

3.2.4 Operation Memory (OPM) Module

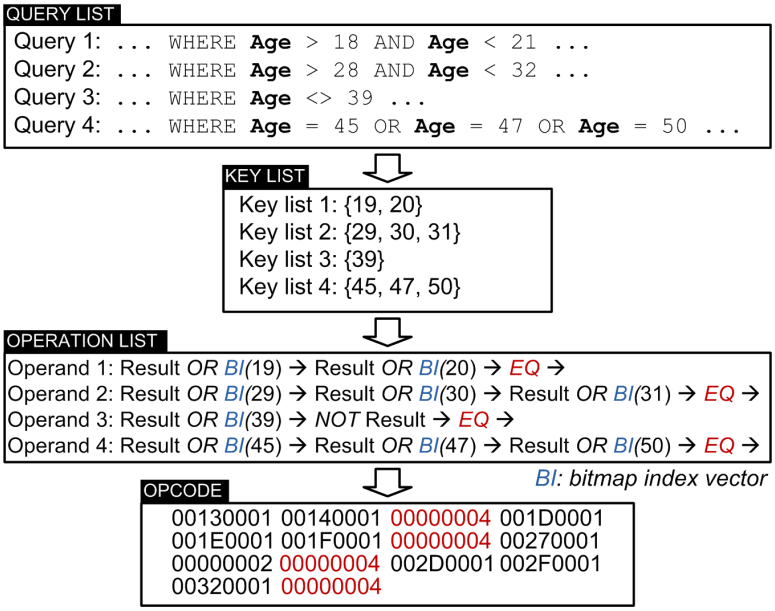
OPM stores the operations/keys extracted from the user's query. It is built from embedded RAM blocks and can contain as many as 2,048 32-bit operations, or OPM's size is 64 Kbit. Larger OPM is also easily constructed by adding more RAM blocks. Each operation is composed of two parts, as seen in Figure 3.6(a). The first part is a 16-bit key value that supports the highest cardinality of 65,536. The second part is a 3-bit instruction value that supports three operations, namely OR, NO, and EQ. Additional operations can be inserted easily by using the reserved bits. Except for OR and NO are logical operations, EQ is only asserted whenever we transfer BI vector to the external memory. Due to the data width of 256 bits, up to eight 32-bit operations are loaded into OPM at every clock cycle.

Figure 3.6(b) gives an example of translation from a list of four queries to the correspondent operations/keys. Due to the light workload, the translation is performed in advance, such as by a computer, and the final binary configuration will be transferred to the external memory. Concretely, four key lists are obtained from those queries promptly. Afterward, each key list is combined with a set of relating operations, i.e. OR, NO, or EQ. Finally, all binary opcodes are copied to the external memory. Taking the first operation as an example, the index vector of key = 19 is combined with the result by OR operation. After two executions, the range index vector of the first query is sent out by the EQ operation. It is noted that the result is automatically cleared as soon as BIC is powered up or



Operand	Description
OR	Result ← Result *OR* (Index(Key))
NO	Result ← *NOT* (Result)
EQ	Send the result register to DDR3

(a) The word structure of OPM.



(b) The example of query-to-operation/key conversion.

Figure 3.6: The illustration of OPM.

the *EQ* operation is completed.

3.2.5 Query Logic Array (QLA) Module

QLA is the most compute-intensive module where the range indexes are calculated. Figure 3.7 depicts the scalable architecture of QLA collaborating with CAM64K8 that contains a 64-Kbit result register and an array of the set of logic gates, including an inverter gate, an OR gate, and a multiplexer. Each bit of CAM output is connected with each logic set numbered from zero to 65,535. Additionally, the output of each logic set enters the result register for temporary

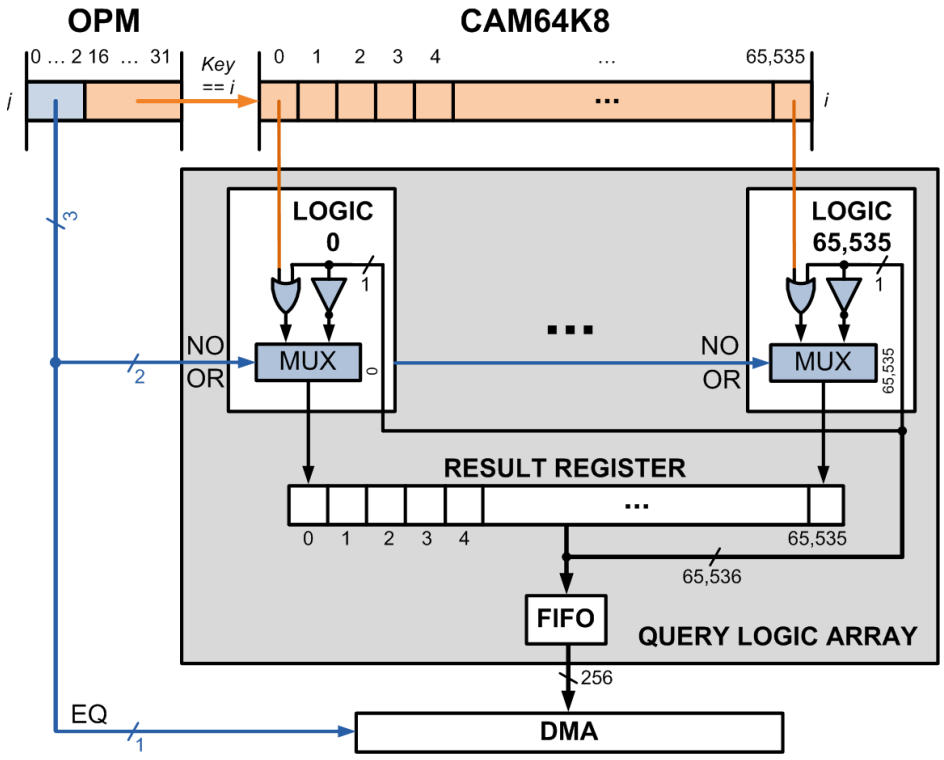


Figure 3.7: The block diagram of QLA.

storage. Taking OPM(i) as an example, the 16-bit key value selects the proper BI vector in CAM64K8, while the 3-bit operation configures the multiplexers and DMA. As a result of the simplicity of logic sets, each logical operation can be solved within one clock cycle. However, the execution time of the *EQ* operation varies with the register size. For example, in the case of CAM64K8, it takes at least 256 cycles to transfer a 64-Kbit result to the external memory, whereas in the case of CAM32K16, the size of the register and logic gates are halved. Thus, only 128 cycles are needed.

The main advantage of QLA is that the range index of many values can be indexed at the latency of the clock cycle. For instance, we can index as many as 65,536 values by the key list of {19, 20} only within two clock cycles or 20 ns in the case of 100-MHz operating frequency. Moreover, FIFO is used to enhance the parallelism between the indexing and transfer process. If *EQ* is asserted, the range index result enters FIFO. When the result register is fully stored in

FIFO, the next indexing process is started immediately. Unless FIFO is empty, DMA transfers data from FIFO to external memory. As a result, the indexing and transfer process can be operated in parallel to save the whole indexing time. Through those achievements, BIC is likely to be far better than that implemented in software.

3.3 Summary

This chapter has proposed two scalable high-performance BIC64K8 and BIC32K16, which can index as many as 65,536 8-bit words and 32,768 16-bit words in parallel, respectively. Each BIC is constructed by three primary modules, i.e. a RAM-based CAM, an OPM, and a QLA. An enhanced architecture of RAM-based CAM is introduced to significantly improve the loading time. The performance analysis of BIC will be described in detail in Chapter 6.

Chapter 4

Bitmap-Index-Based Query Processor

The purpose of this chapter is to propose a scalable hardware architecture of a bitmap-index-based query processor that can answer multi-dimensional queries by given BI vectors in parallel. The output of this design is the bitmap vectors, where all one-bits point to the addresses of matching records.

4.1 Introduction

In the analytics-like workloads, most queries are highly complex and involve aggregations. Because BI copes with such queries effectively, research on BI has gained extensive interest. In the previous chapter, we have introduced a BIC that can index up to 64-KB data in parallel. In this chapter, a BI-based query processor (BIQP) that can deliver the query results from the given BI vectors is proposed. The contributions are listed as follows.

- This chapter presents an enhanced cascade architecture of a BI memory that stores all necessary BIs for query response. This BI memory stores up to 512 32-Kbit BI vectors. When being requested, each 32-Kbit vector is employed for bitwise logical operation calculation.
- This chapter proposes a full design of BIQP that can directly access the external memory to lower the I/O cost and produce the query results of up

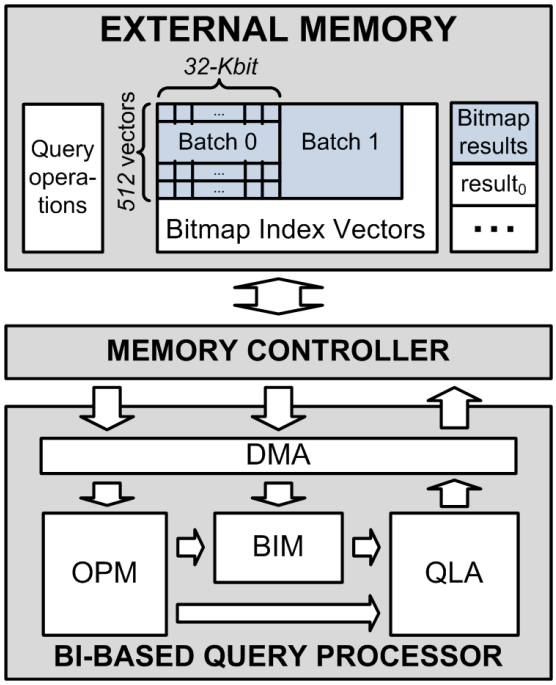


Figure 4.1: The block diagram of BIQP.

to 512 32-Kbit BI vectors. Additionally, BIQP size is simply adjusted by adding or removing correspondent logic and memory resources. The content of this chapter was partially presented in some previous works [59, 60, 61].

The remainder of this chapter is organized as follows. Section 4.2 describes the hardware architecture of BIQP in detail. Section 4.3 concludes this study. The performance analysis of BIQP is discussed in detail in Chapter 6.

4.2 Architecture

4.2.1 Overview

Figure 4.1 illustrates the block diagram of a BIQP used to response to multi-dimensional queries. BIQP is a combination of four main modules including a direct memory access (DMA) module, a BI memory (BIM) module, an operation memory (OPM) module, and a query logic array (QLA) module. The external memory stores all of the operations that are extracted from users' queries, all

BI vectors previously produced by BIC or other DBMS, and all query results delivered by BIQP. Furthermore, to fit in BIM, BI vectors are distributed into a set of batches, each of which can contain up to 512 32-Kbit vectors, depending on the number of dimensions in a query.

To begin, all operations are transferred to the OPM. Subsequently, the first batch of BI vectors is copied into BIM. QLA then uses OPM and BIM to produce the query results and return them to the external memory. Afterwards, the second batch is copied into BIM to begin a new process. This procedure repeats until the final batch is completely executed.

4.2.2 Direct Memory Access (DMA) Module

The custom designed DMA guarantees the high-efficiency transfer between BIQP and the memory controller provided by Intel FPGA. Since the theoretical bandwidth of DDR3 memory is 25.6 Gbps, we set the data width of 256 bits and operating frequency of 100 MHz for the memory controller, in order to fully exploit this bandwidth. Inside the DMA, three channels are also designed to allow BIQP to access the memory independently.

4.2.3 Bitmap Index Memory (BIM) Module

BIM temporarily stores a $M \times N$-bit BIs received from the DDR3 memory. The basic component of BIM is a dual-port memory (DPM), whose architecture is shown in Figure 4.2(a). Roughly speaking, DPM is a special type of RAM that supports two distinct ports—port A and port B. At port A, data can be read and written using the data output qa, data input da, address aa and write enable wa. If wa turns into one, DPM[aa] = da; otherwise, qa = DPM[aa]. Likewise, at port B, qb, db, ab and wb are used to control the read and write processes. Two ports can manage the access requests simultaneously. Since Arria V SX FPGA deploys M10K as a basic memory block, every DPM is configured as a 512×16-bit RAM to gain the best resource utilization.

As mentioned earlier, DMA is connected with the memory controller by 256-bit bus width. For this reason, 16 DPMs are connected in parallel to form a so-called BIM unit (BIMU), as depicted in Figure 4.2(b). The BIMU receives

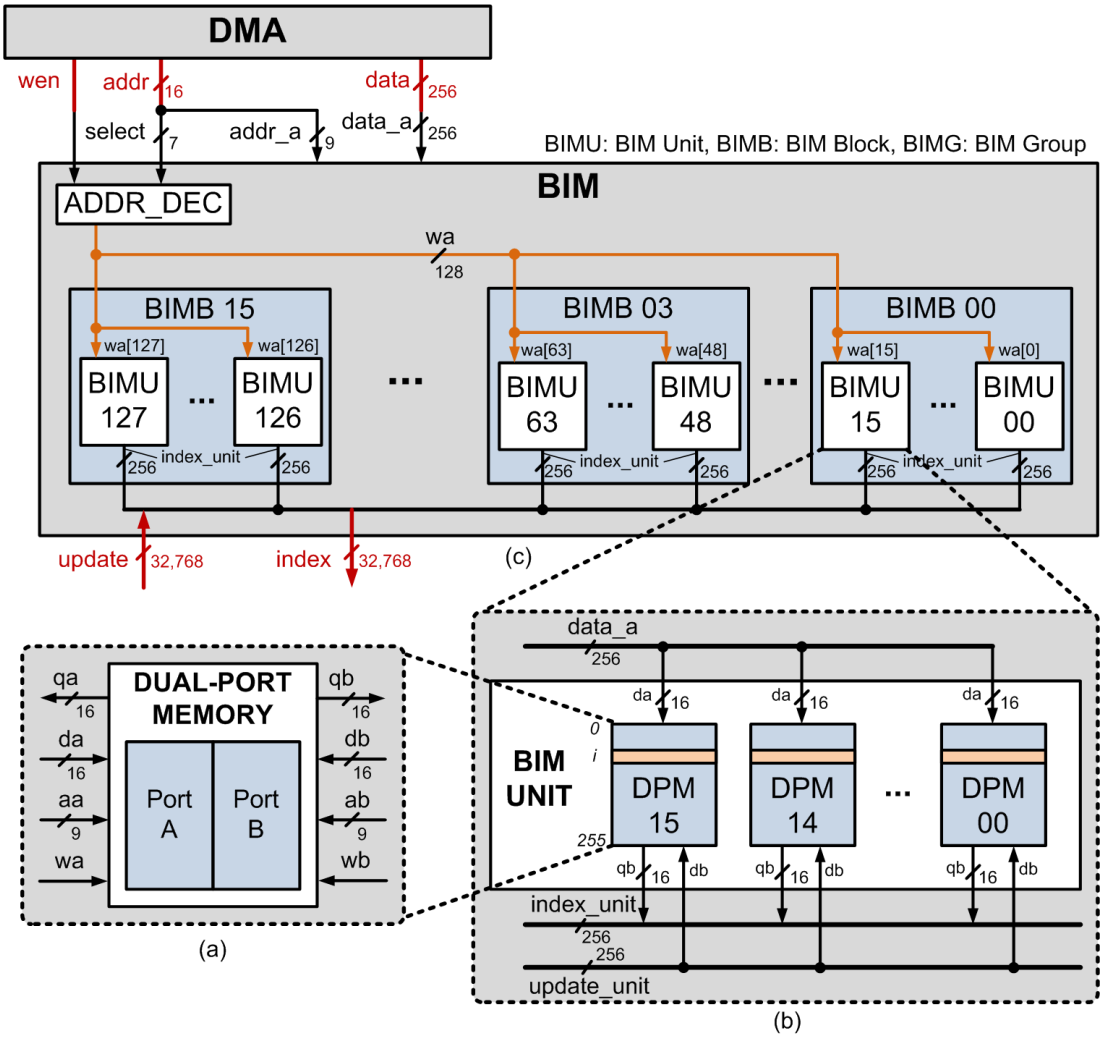


Figure 4.2: The block diagram of BIM.

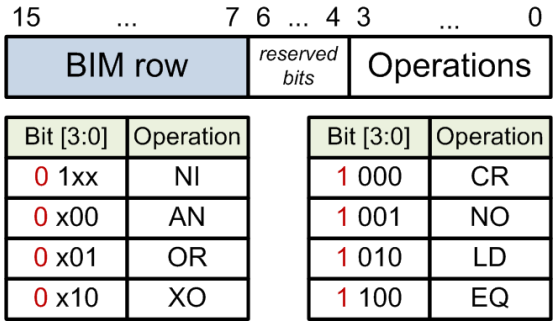
Table 4.1: The loading process of BIM.

Cycle	*wen*	*addr*[15:0]		*wa*[127:0]
		addr_a[8:0]	*select*[6:0]	
1	1	0 0000 0000	000 0000	$wa[0] = 1$
2	1	0 0000 0000	000 0001	$wa[1] = 1$
...	...	...	...	...
127	1	0 0000 0000	111 1111	$wa[127] = 1$
128	1	0 0000 0001	000 0000	$wa[0] = 1$
...	...	...	...	...
65,536	1	1 1111 1111	111 1111	$wa[127] = 1$

256-bit inputs *data_a* from DMA and delivers 256-bit outputs *index_unit* to QLA. Since data enter BIMU in parallel, its loading throughput presumably reaches as high as that of DMA—25.6 Gbps. Hence, every BIMU can be filled up within 512 clock cycles corresponding to the DPM size. In addition to *index_unit*, BIMU can also receive a 256-bit *update_unit* from QLA.

Figure 4.2(c) illustrates the scalable architecture of a 512×32-Kbit BIM constructed with eight BIM blocks (BIMB) that employs 16 BIMUs inside of it. As a result, the number of DPMs, BIMUs, and BIMBs are 2,048, 64 and eight, respectively. The address signal *addr* is divided into *addr_a* and *select*. Both *addr_a* and data signal *data_a* are connected to all of the individual BIMUs. The address decoder ADDR_DEC combines write enable signal *wen* and *select* to allow a certain BIMU to receive new *data_a*. Concretely, at the beginning, BIMU00[0] saves *data_a* at the first cycle. BIMU01[0] then saves *data_a* at the second cycle, and after 125 clock cycles, BIMU127[0] can store *data_a*. Hence, each row requires 128 clock cycles to be filled up. Following this, BIMU00[1] updates *data_a* at the next cycle. This process repeats until N_b rows of BIMU are filled properly. At that moment, each row of BIMU stores each correspondent BI vector of the current batch. The loading process above is briefly summarized in Table 4.1. Apparently, it takes 65,536 clock cycles to fully fill BIM.

The advantage of BIM is that every 32-Kbit BIM row can be requested at each clock cycle. Accordingly, BIQP is capable of handling one operand per 32,768 indexes in every single cycle, which potentially enhances the total query processing throughput.



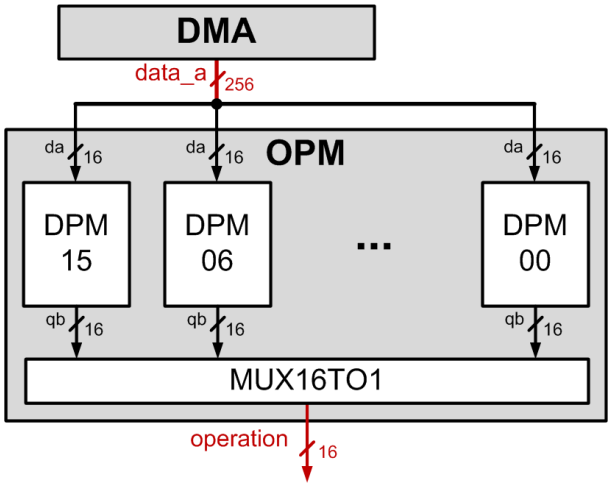
Bit [3:0]	Operation
0 1xx	NI
0 x00	AN
0 x01	OR
0 x10	XO

Bit [3:0]	Operation
1 000	CR
1 001	NO
1 010	LD
1 100	EQ

(a) The structure of an operation

Operation	Description
NI	variable ← *NOT* (variable)
XO	variable ← *XOR* (Result, variable)
OR	variable ← *OR* (Result, variable)
AN	variable ← *AND* (Result, variable)
NO	variable ← *NOT* (variable)
CR	Result ← 0
LD	BIM row ← Result
EQ	DDR3 ← Result

(b) The description of all operations.



(c) The block diagram of OPM.

Figure 4.3: The description of OPM.

4.2.4 Operation Memory (OPM) Module

OPM is built by the FPGA embedded memories, which store the operations extracted from the users' queries. Each 16-bit operation is composed of two main parts: BIM row and operation content, as shown in Figure 4.3(a). The first part– BIM row–has nine bits to manage current 512-row BIM. However, this part can expand up to 4,096 because of three reserved bits. The second part has three bits to support nine distinct operations. Those operations are divided into two groups, i.e. $\{NI, AN, OR, XO\}$ and $\{CR, NO, LD, EQ\}$, which are encoded by the most significant bit. The exact operation is then selected by the last three bits.

Figure 4.3(b) gives a brief description of each operation. Generally speaking, XO, OR, AN, and NI/NO indicate the bitwise logical operators XOR, OR, AND, and NOT, respectively, between input data and the result register (RR). The difference between NI and NO will be clearly stated in Section 4.2.5. In addition to those five logical operations, CR, LD, and EQ are used to control QLA and DMA. First, RR is cleared by setting CR to one. Second, RR is copied to a specific row of BIM by turning LD into one. Third, RR is sent to memory by setting EQ to one.

Figure 4.3(c) depicts the general block diagram of OPM. Similar to BIM, OPM is constructed by a set of 512×16-bit DPMs. However, the loading process of OPM is completely different from that of BIM. Concretely, 256-bit input data enters to every row of all DPMs simultaneously, or 16 operations are loaded at every clock cycle. On the other hand, each operation is retrieved in turn by MUX16TO1. The order to this process is from the first row of DPM00 to the first row of DPM15, then from the second row of DPM00 to the second row of DPM15, and so on. The process repeats until all operations are pushed out properly. Due to this architecture, OPM is capable of containing up to $\frac{512\times16}{2} =$ 4,096 operations.

Figure 4.4 gives an example of query-to-operation conversion. Since four attributes, namely *Product*, *Year*, *Occupation*, and *Address*, are involved in this query, four correspondent BI vectors are generated by BIC in advance. Those BI vectors may be stored in several batches, if their lengths are larger than BIM

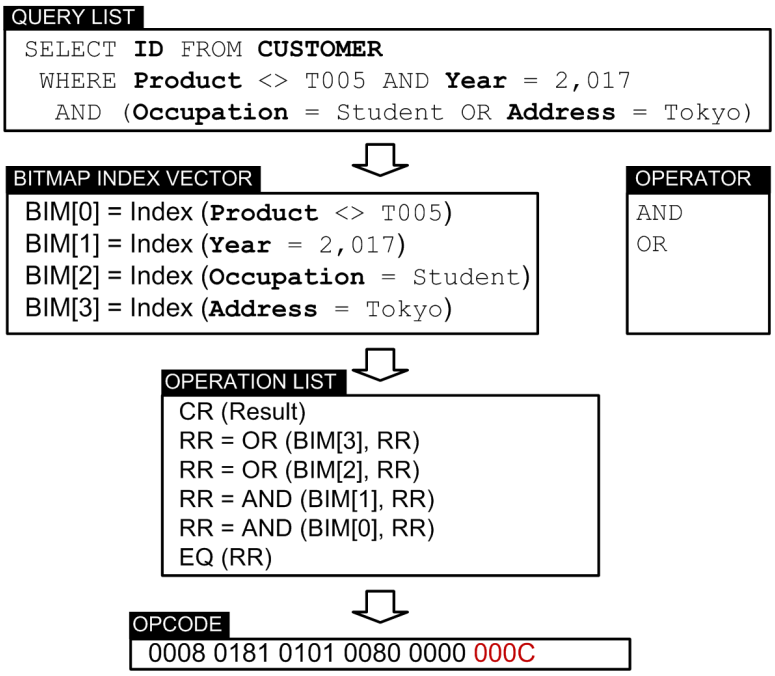


Figure 4.4: The example of query-to-operation conversion.

length. Each batch is then dispatched to four available rows inside BIM, e.g. from the zeroth row to the third row. Subsequently, six operations are executed. First, RR is cleared by $CR = 1$. Second, bitwise AND and OR operations are performed between four correspondent BIM rows and RR. Finally, RR is returned to the DDR3 memory by $EQ = 1$. Due to the light workload, the query-to-operation conversion is conducted by computer software.

4.2.5 Query Logic Array (QLA) Module

QLA is the most important module of BIQP, where the queries will be executed. Figure 4.5 depicts the scalable architecture of QLA collaborating with BIM that contains 32,768 sets of query logic and a 32-Kbit RR. Moreover, the output of each set is connected with each bit of the RR. Hence, the number of query logic sets is proportional to the length of RR or BIM row. For instance, in case the BIM row only has 16 Kbits, 16,384 query sets and 16-Kbit RR are designed instead. Each query set is composed of two NOT gates, one AND gate, one OR gate, one XOR gate and three multiplexers. As mentioned above, although both

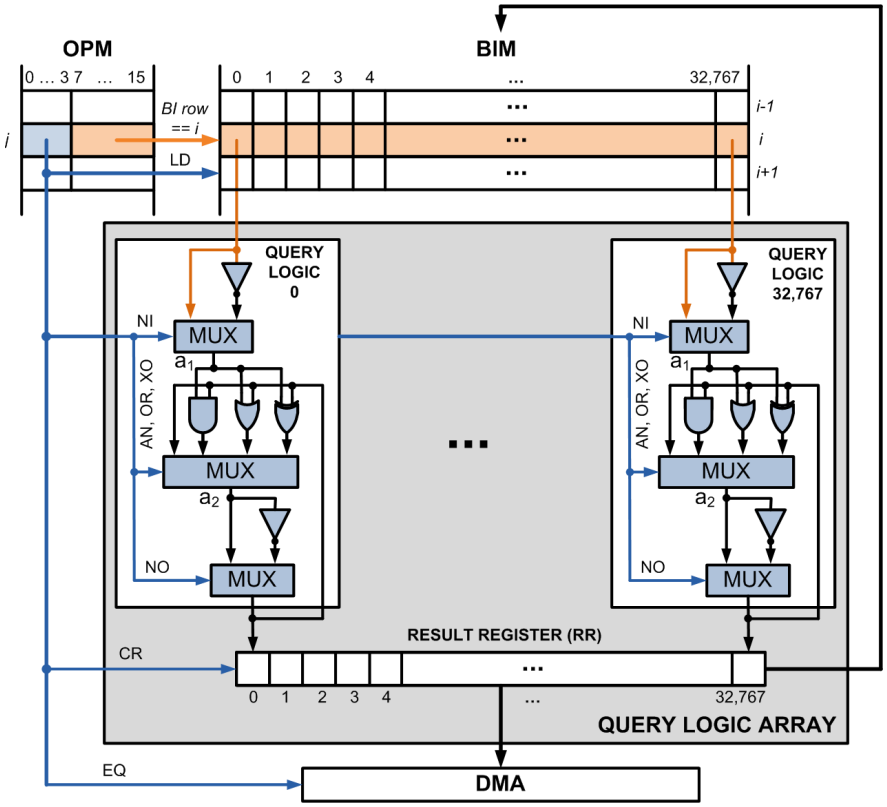


Figure 4.5: The block diagram of QLA module.

NI and NO perform the bitwise NOT operators, their functions are completely difference. In fact, $NI = 1$ reverses the BI vector coming from BIM, whereas $NO = 1$ reverses either the intermediate value $a2$ or RR.

Taking OPM_j as an example, the last nine bits select a correspondent BIM's row, while the first three bits control the multiplexers and DMA. As a result of the simplicity of logic sets, each logical operation can be solved within one clock cycle. Besides, it takes at least 128 clock cycles to transfer a 32-Kbit result to the external memory because of the 256-bit data width.

Furthermore, $CR = 1$ will reset RR instantly. LD is connected to write enable wen of BIM to allow RR to be copied into BIM. More significantly, CR and LD are combined to be able to clear the entire BIM. To achieve it, RR is first reseted by CR. Second, RR is transfered to BIM sequentially–from the first row to the last row. With this method, BIM are cleared properly after 512 clock cycles.

The main advantages of QLA is that as many as 32-Kbit BI vector can be

processed concurrently at the latency of the clock cycle. Moreover, the hardware parallelism allows QLA and BIM to operate independently. Concretely, when all operations are fully executed, a new batch enters to BIM in parallel with the result transfer from QLA to DMA. Therefore, the entire processing time is significantly reduced.

4.3 Summary

This chapter has proposed a BIQP that can process the queries using given BI vectors. The BIQP is constructed by three primary modules, i.e. a BIM, an OPM, and a QLA. The BIM architecture allows as high as 32-Kbit BI vector to be calculated in parallel in each clock cycle. The performance analysis of BIQP will be clearly described in detail in Chapter 6.

Chapter 5

Bitmap Index Encoder

The purpose of this chapter is to propose a scalable hardware architecture of a bitmap index encoder accelerator that can output all matching bits of input bitmap data in parallel. Those matching bits are then used to locate the corresponding match records in a database.

5.1 Introduction

In previous chapter, a BI-based query processor that delivers the bitmap results, where one-bits indicate the location of matching records in a database, has been presented. Encoding those one-bits without severely affecting the whole analytics time also plays an important role in the design of a BI-based analytics systems. This chapter, therefore, focuses on a scalable hardware architecture of a bitmap index encoder (BIE) that is mainly constructed on a multi-match priority encoder (MPE). The contributions are listed as follows.

- This chapter introduces an efficient architecture of a priority encoder (PE) based on a novel approach called a one-dimensional-array to two-dimensional-array (1D-to-2D) conversion method. This method converts an L-bit input data into an $M \times N$-bit matrix, from which an N-bit PE and an M-bit PE are employed to obtain the highest priority bit of L-bit input data.
- This chapter presents an efficient architecture of a MPE built from a PE and a preprocessing logic circuit. After PE detects a matching bit, this bit

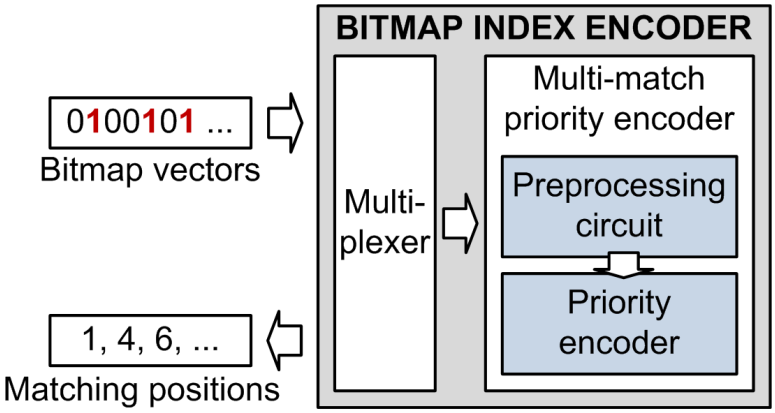


Figure 5.1: The block diagram of BIE.

will cleared by the preprocessing circuit so that the next priority bit can be look up in the next clock cycle.

- This chapter proposes a full BIE design that receives the 32-Kbit bitmap results and outputs the matching bits. BIE is composed of a MPE and a multiplexer that controls which input segment can enter into a MPE at the specific time. Moreover, BIE size is simply adjusted by adding or removing correspondent logic resources. The content of this chapter was partially presented in our previous works [51, 63, 64].

The remainder of this chapter is organized as follows. Section 5.2 clearly describes a hardware architecture of large-sized PEs. Section 5.3 presents the conclusion. The performance analysis of BIE is discussed in detail in Chapter 6.

5.2 Architecture

5.2.1 Overview

Figure 5.1 illustrates the block diagram of a BIE that is used to encode a bitmap vector. As mentioned earlier, BIE consists of a multiplexer and a MPE, which is made by a preprocessing circuit and a PE. To begin, the multiplexer selects the which segment of input that enters into MPE, in case the input size exceeds the MPE size. Subsequently, MPE outputs each encoded position at the rate of

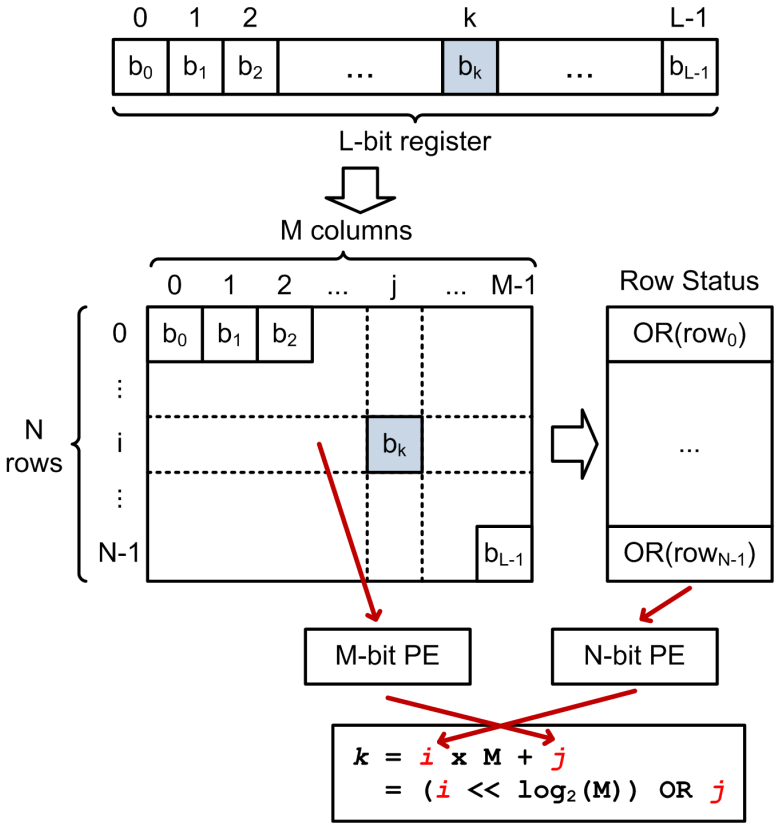


Figure 5.2: The conversion from L-bit input to $M \times N$-bit input.

one clock cycle per match. Such performance is achieved by using an efficient PE based on an original 1D-to-2D conversion method.

Figure 5.2 illustrates this method, where L-bit input data is converted into a $M \times N$-bit matrix, where M and N are the numbers of columns and rows, respectively. All bits of row status are obtained by performing the bitwise OR to all bits in the corresponding row. Subsequently, an N-bit PE finds the highest priority bit i (row index) in the N-bit row status, and an M-bit PE seeks the highest priority bit j (column index) in this row i. The matching position k of an 1D-array input is retrieved as $k = i \times M + j$. More significantly, if M is a power of two, the multiplier and adder are simply replaced by the fixed wirings that function as left-shift and OR operators.

Taking an example of PE64, only two small-sized PEs, e.g. $(M, N) = (2, 32)$, $(32, 2)$, $(8, 8)$, $(4, 16)$, and $(16, 4)$, are required to attain the highest priority

position of a PE64. Furthermore, a scalable structure of a large-sized PE can be simply developed in a similar vein by using the results of PE64s. Finally, MPE is formed by combining PRE with PE. The next section will present a detailed systematic approach to the scalable high-performance large-sized PEs. We also propose a methodology to select an optimum pair of (M, N) because the correct choice of (M, N) also plays an important role in constructing high-performance PEs.

5.2.2 Priority Encoder (PE) Module

Figure 5.3(a) describes the truth table and optimized boolean expression of a PE4. Apparently, this circuit only employs basic operations, i.e. AND, OR, and NOT, which is processed very quickly by hardware logic units. Similarly, the expressions of a PE8 and 16-bit PE (PE16) are correspondingly given in Figure 5.3(b) and Figure 5.3(c). We observe that the complexity of expressions increases drastically, as PE size varies from 4-bit to 16-bit, which possibly causes an implementation of a 32-bit PE to become impracticable. Thus, only PE4, PE8, and PE16 are employed to construct large-sized PEs. Concretely, at L of 64-bit, we examine (M, N) as (8, 8), (4, 16), and (16, 4).

Figure 5.4(a) shows PE64 formed by two PE8s connecting in a series, namely $\text{PE64}_{(8n)}$. To begin with, the input data D is separated into eight 8-bit signals that are put into eight 8-bit OR gates (OR8s) in order together with the 8-to-1 multiplexer (MUX8N). The output of MUX8N, called $DMUX$, is determined by $MR8$ - the position of the highest priority bit of DOR. Following this, $MC8$, the location of the highest priority bit of $DMUX$, is obtained. The output Q is derived from the bitwise OR between $MC8$ and $MR8$ that was shifted left by three bits. Additionally, if D contains any bit one, M turns into one.

Because $\text{PE64}_{(8n)}$ follows the formula stated in Figure 5.2, the longest delay of $\text{PE64}_{(8n)}$ is approximately the sum of four individual components' delay. In fact, MUX8N has to wait until $MR8$ is ready before allocating a proper column index to $DMUX$. To reduce this delay, we employ DOR as a look-ahead signal, which is illustrated in Figure 5.4(b). As can be easily seen, DOR cuts the longest data path, from the input of PE8_0 to the output Q, in two shorter paths operating

TRUTH TABLE

D_3	D_2	D_1	D_0	Q_1	Q_0
0	0	0	1	0	0
0	0	1	x	0	1
0	1	x	x	1	0
1	x	x	x	1	1

x: Don't care

$$Q_0 = \sum(D_1, D_3) = \overline{D_2}.D_1 + D_3$$
$$Q_1 = \sum(D_2, D_3) = D_2 + D_3$$

(a) PE4.

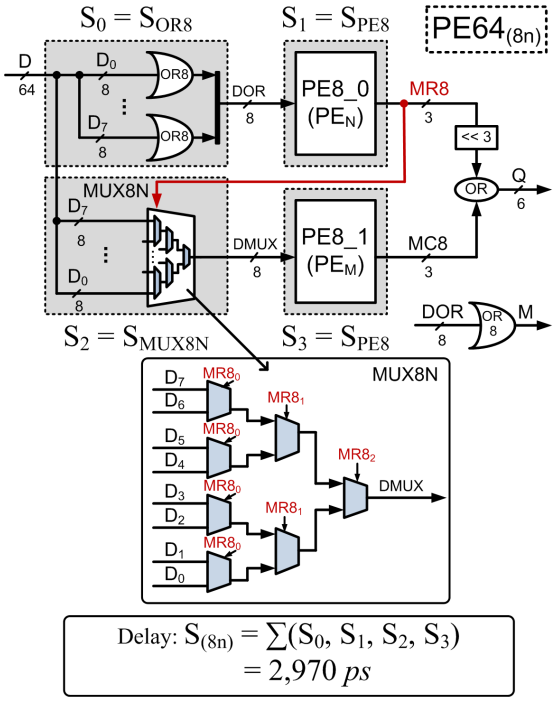
$$Q_0 = \sum(D_1, D_3, D_5, D_7) = \overline{D_6}.(\overline{D_4}.\overline{D_2}.D_1 + \overline{D_4}.D_3 + D_5) + D_7$$
$$Q_1 = \sum(D_2, D_3, D_6, D_7) = \overline{D_5}.\overline{D_4}.(D_2 + D_3) + D_6 + D_7$$
$$Q_2 = \sum(D_4, D_5, D_6, D_7) = D_4 + D_5 + D_6 + D_7$$

(b) PE8.

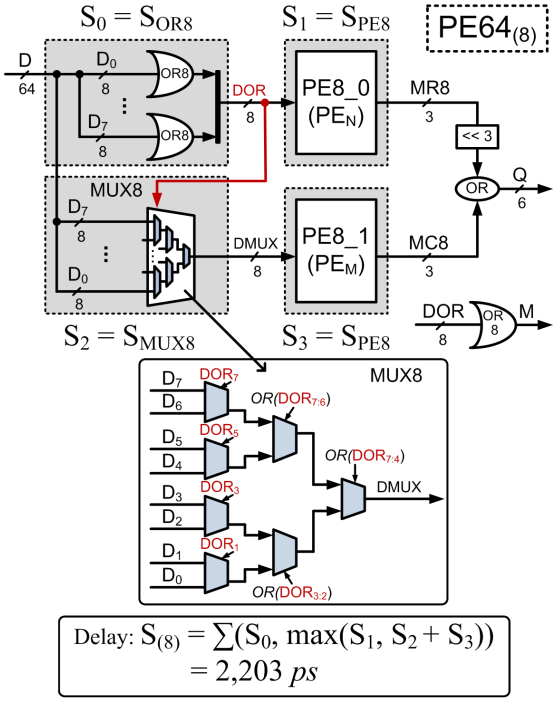
$$\begin{aligned}
Q_0 &= \sum(D_1, D_3, D_5, D_7, D_9, D_{11}, D_{13}, D_{15}) \\
&= \overline{D_{14}}.\overline{D_{13}}.\overline{D_{12}}.(\overline{D_{11}}.\overline{D_{10}}.\overline{D_9}.\overline{D_8}.(\overline{D_7}.\overline{D_6}.\overline{D_5}.\overline{D_4}. \\
&\quad (\overline{D_2}.D_1 + D_3) + (\overline{D_6}.D_5 + D_7)) + \overline{D_{10}}.D_9 + D_{11}) + \\
&\quad \overline{D_{14}}.D_{13} + D_{15} \\
Q_1 &= \sum(D_2, D_3, D_6, D_7, D_{10}, D_{11}, D_{14}, D_{15}) \\
&= (\overline{D_{13}}.\overline{D_{12}}.(\overline{D_{11}}.\overline{D_{10}}.\overline{D_9}.\overline{D_8}.(\overline{D_7}.\overline{D_6}.\overline{D_5}.\overline{D_4}. \\
&\quad (D_2 + D_3) + D_6 + D_7) + D_{10} + D_{11})) + D_{14} + D_{15} \\
Q_2 &= \sum(D_4, D_5, D_6, D_7, D_{12}, D_{13}, D_{14}, D_{15}) \\
&= (\overline{D_{11}}.\overline{D_{10}}.\overline{D_9}.\overline{D_8}.((D_7 + D_6) + (\overline{D_7}.\overline{D_6}.(D_5 + D_4)))) \\
&\quad + D_{12} + D_{13} + D_{14} + D_{15} \\
Q_3 &= \sum(D_8, D_9, D_{10}, D_{11}, D_{12}, D_{13}, D_{14}, D_{15}) \\
&= D_8 + D_9 + D_{10} + D_{11} + D_{12} + D_{13} + D_{14} + D_{15}
\end{aligned}$$

(c) PE16.

Figure 5.3: The truth table and Boolean expression.

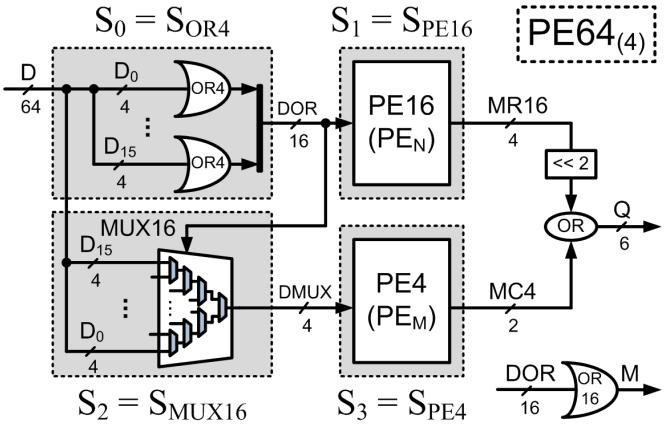


(a) Without look-ahead signal.

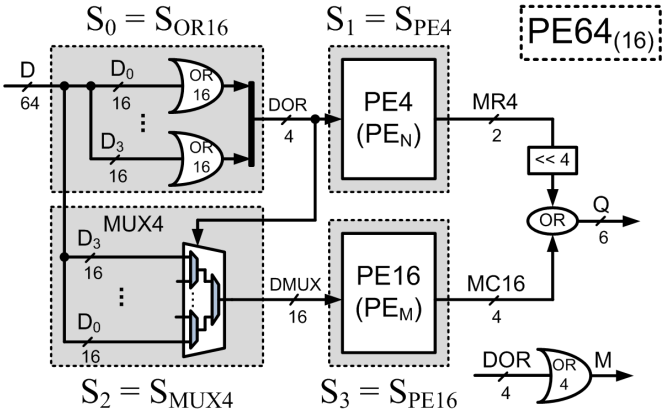


(b) With look-ahead signal.

Figure 5.4: The architecture of PE64.



(a) $(M, N) = (4, 16)$.



(b) $(M, N) = (16, 4)$.

Figure 5.5: The architecture of PE64.

in parallel. Therefore, the entire latency of $\text{PE64}_{(8)}$ is likely to be lowered, as compared to that of $\text{PE64}_{(8n)}$. Moreover, the select signals inside MUX8 reassigned because of the difference in the number of bits between $MR8$ and DOR. The resource utilization of $\text{PE64}_{(8)}$, therefore, increases because MUX8 requires several additional OR gates.

To quickly estimate PE performance, we synthesize all OR gates, PEs, and multiplexers to observe the path delay (in terms of ps), from the input to the output of each circuit. The synthesis tool is configured to generate the gate-level logic under an aggressive timing constraint. Table 5.1 summarizes the synthesized results in 180-nm CMOS technology. Suppose that S_0, S_1, S_2, and S_3

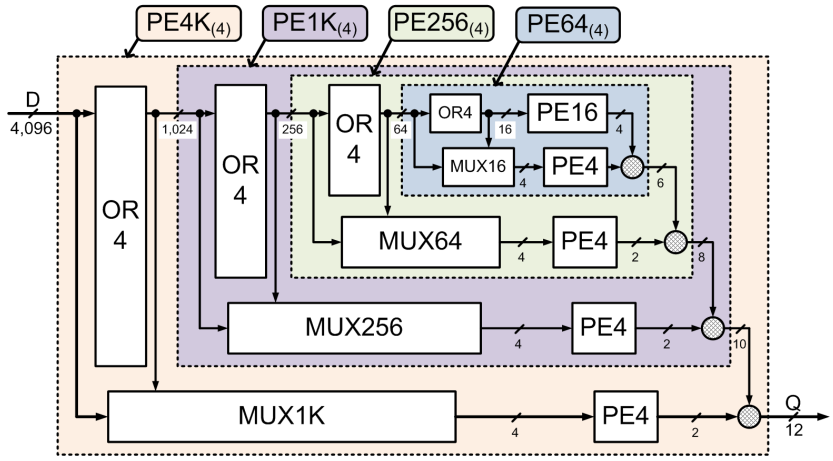


Figure 5.6: The scalable architecture of $PE4K_{(4)}$.

are the path delays of four primary circuits in $PE64_{(8n)}$ and $PE64_{(8)}$. As seen in Figure 5.4(a), without a look-ahead signal, the delay of $PE64_{(8n)}$ is $S_{(8n)} = \Sigma(S_0, S_1, S_2, S_3) = 2{,}970$ *ps*. On the other hand, the latency $PE64_{(8)}$ is lessened as $S_{(8)} = \Sigma(S_0, \max(S_1, S_2 + S_3)) = 2{,}203$ *ps*. The preliminary analysis suggests that the look-ahead signal enhances the circuit performance.

As briefly mentioned before, in case of PE64, there are three possible pairs of (M, N), i.e. (8, 8), (16, 4), and (4, 16). The architecture of PE with (M, N) of (4, 16) and (16, 4), so-called $PE64_{(4)}$ and $PE64_{(16)}$, are defined in Figure 5.5(a) and Figure 5.5(b), respectively. It is noted that PE_N and PE_M also represent the top PE and bottom PE. In both architectures, the highest priority bit of input data D is discovered in a similar manner with $PE64_{(8)}$, except for the different use of OR gates, multiplexers, and the organization of PE_N and PE_M. Using the preliminary analysis above, the path delay of $PE64_{(4)}$ is $S_{(4)} = 2{,}086$ *ps*, whereas that of $PE64_{(16)}$ is $S_{(16)} = 2{,}444$ *ps*. Altogether, the performance of four alternative PE64s are sorted as $PE64_{(4)} > PE64_{(8)} > PE64_{(16)} > PE64_{(8n)}$. In other words, if PE4 is used to generate the column index ($M = 4$), the overall performance is likely to be optimized.

This preliminary analysis also implies the scalable architecture of a large-sized PE such as $PE4K_{(4)}$ that can be developed by PE4, PE16, $PE64_{(4)}$, 256-bit PE ($PE256_{(4)}$), and 1,024-bit PE ($PE1K_{(4)}$), as seen in Figure 5.6. Initially, the 4,096-bit input is considered as a 1,024×4-bit array. Subsequently, $PE1K_{(4)}$ and

Table 5.1: The number of logic stages.

Circuit	Delay (ps)
OR4	336
OR8	399
OR16	550

Circuit	Delay (ps)
PE4	680
PE8	780
PE16	980

Circuit	Delay (ps)
16-bit MUX4	914
8-bit MUX8N	1,011
8-bit MUX8	1,024
4-bit MUX16	1,070

PE4 are employed to calculate the corresponding indexes of row and column. Similarly, inside $\text{PE1K}_{(4)}$, the 1,024-bit is converted into a 256×4-bit array for the next processing from $\text{PE256}_{(4)}$ and PE4. Dividing the input repeats until PE_N is either PE16 or PE8. Finally, the highest priority bit is achieved from all PE outputs, based on the formula described in Figure 5.2.

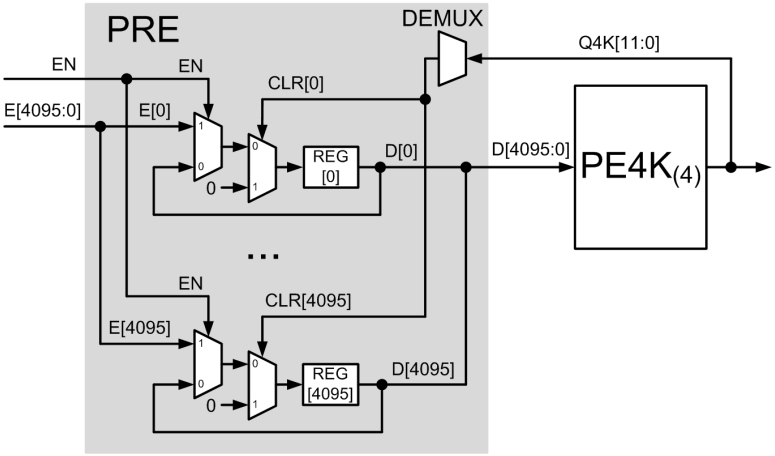
5.2.3 Multi-match Priority Encoder (MPE) Module

Figure 5.7(a) depicts a 4,096-bit MPE (MPE4K) built up from a $\text{PE4K}_{(4)}$ and a PRE circuit containing a set of multiplexers and register arrays (REG). In the beginning, EN is set to one and input data E is kept in REG. In the subsequent cycles, D - the output of REG, is sent to $\text{PE4K}_{(4)}$. Upon receiving the matching position $Q4K$, the demultiplexer (DEMUX) converts this value into a 4096-bit clear signal CLR. If $CLR[i]$ is equal to one, REG$[i]$ is set to zero instantly, so that $\text{PE4K}_{(4)}$ will look for the next priority bit in the following cycles. This procedure repeats until REG turns into zeros completely.

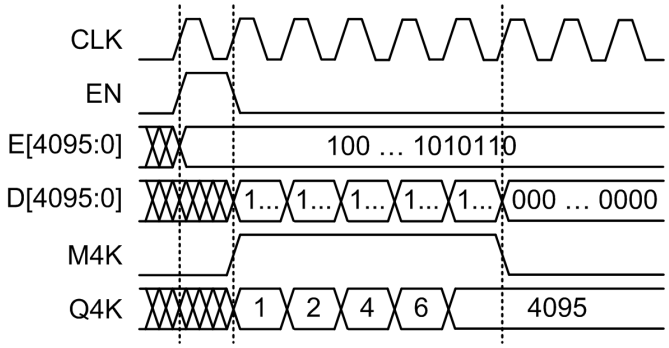
If E contains five matching bits, the simulation waveform of MPE4K is shown in Figure 5.7(b). At the first clock, EN is asserted, and D captures the value of E. During the next five clocks, $M4K$ becomes one and each matching position, i.e. $\{1, 2, 4, 6, \text{and } 4,095\}$, is returned in turn. Afterwards, $M4K$ changes to zero, which means that all matches are captured completely.

5.2.4 Bitmap Index Encoder (BIE) Module

Figure 5.8(a) illustrates a BIE using an MPE4K to find all the matching bits of a 32-Kbit bitmap input. This bitmap can be given by either a BIQP mentioned in Chapter 4 or a software DBMS. To begin, the 32-Kbit bitmap is divided into eight 4-Kbit segments, and each of them is put into MPE4K sequentially, as depicted



(a) The block diagram of MPE4K.



(b) The example waveform of MPE4K.

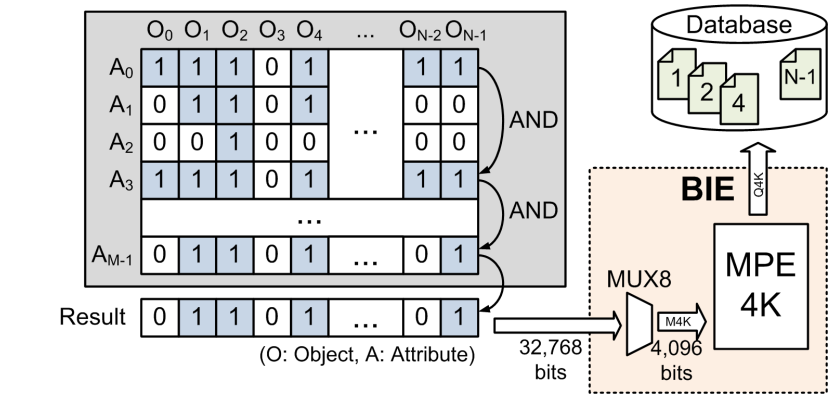
Figure 5.7: The illustration of MPE4K.

in Figure 5.8(b). With each segment processing, BIE requires one initial cycle t_{int} to load a new 4-Kbit segment and K cycles t_{seg} to obtain K matching bits in this segment. Moreover, it takes two additional cycles t_{wait} to be ready for next 4-Kbit segment. Equation (5.1) shows the encoding time in clock cycles in the worst case t_{worst}—all bits are ones, and in the best case t_{best}—all bits are zeros. The worst and best latencies are 32,790 cycles and 22 cycles, respectively.

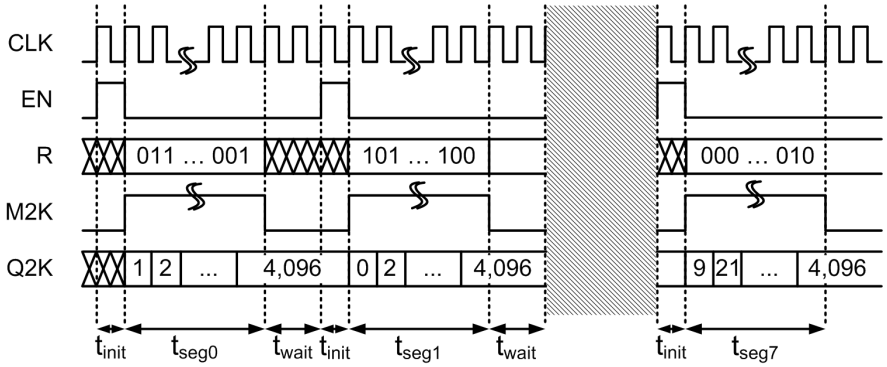
$$t = (\frac{32,768}{4,096}) \times (t_{int} + t_{seg}) + (\frac{32,768}{4,096} - 1) \times t_{wait} \quad (cycles) \tag{5.1}$$

$$t_{worst} = 8 \times (1 + 4,096) + 7 \times 2 = 32,790 \quad (cycles)$$

$$t_{best} = 8 \times (1 + 0) + 7 \times 2 = 22 \quad (cycles)$$



(a) The block diagram of BIE.



(b) The example waveform of BIE.

Figure 5.8: The illustration of BIE.

5.3 Summary

This chapter has proposed a method to develop a scalable high-performance BIE from a PE and a MPE. The BIE is mainly constructed by a PE which is enhanced by an original 1D-to-2D conversion method. The performance analysis of BIE will be clearly described in detail in Chapter 6.

Chapter 6

Performance Analysis and System Integration

The purpose of this chapter is to evaluate the performance of a bitmap index creator, a bitmap-index-based query processor, a bitmap index encoder, and their combinations in an FPGA and an ASIC. The performance analysis primarily focuses on data I/O cost, processing throughput, and energy efficiency. A bitmap-index-based analytics system based on those components is then proposed. By using those dedicated hardware accelerators, the system performance is proven to outperform other CPU-based and GPU-based approaches.

6.1 Introduction

The performance of a BIC, a BIQP, and a BIE are evaluated by data I/O cost, processing throughput, and energy consumption. First, data I/O cost is viewed as the rate at which the data are read and written to the external memory. Low data I/O cost allows user designs to be able to almost exploit the full external memory bandwidth. Second, processing throughput is defined as the amount of data that can be processed within a given time unit. High throughput achievement is generally the result of hardware parallelism. Third, energy efficiency is defined as a ratio of power consumption and processing throughput. Low energy consumption is widely recognized as one of the current primary targets of big

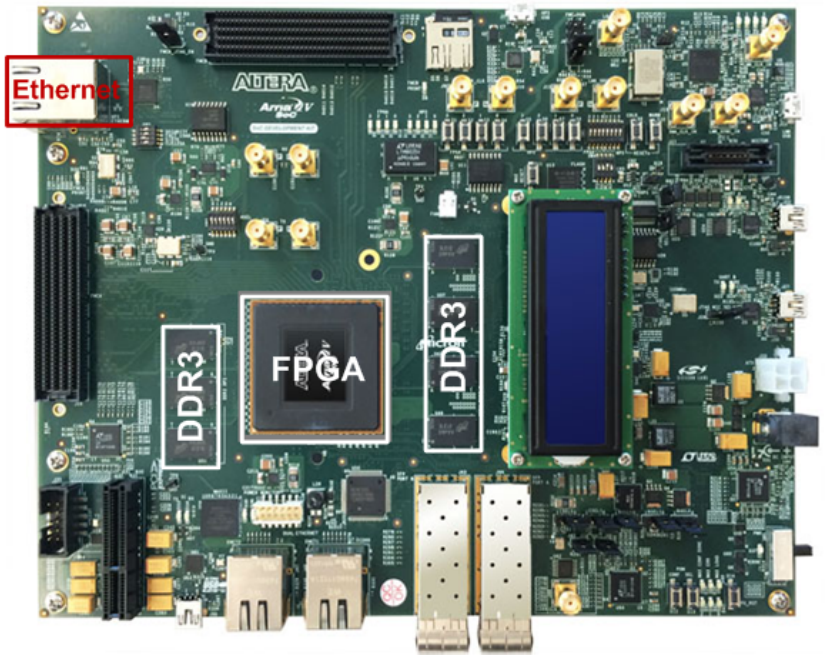


Figure 6.1: The Intel Arria V development kit [65].

data and analytics systems.

The BIC, BIQP, BIE, and their corresponding systems are implemented in an Intel Arria V development kit [65]. This kit contains a mid-range Intel Arria V 5ASTFD5K3F40I3 FPGA, a 1-Gbps Ethernet port, and two separate 1-GB 533-MHz DDR3 external memories, as shown in Figure 6.1. This FPGA is employed because it consists of not only the FPGA portion, but also an ARM-based System-on-Chip portion [66] that allows users to control all of the I/O peripherals above effectively. Because the theoretical bandwidth of DDR3 is 25.6 Gbps, all components need to manage the 256-bit I/O interfaces as well as operate at a *minimum 100 MHz* to avoid the timing violation. In other words, the operating frequency of BIC, BIQP, and BIE must exceed 100 MHz in slow-corner timing analysis.

After being verified in an FPGA, the combination of BIQP and BIE are implemented in the 180-nm bulk CMOS and 65-nm SOTB CMOS process using the Synopsys and Cadence design tools. The layout verification steps including design rule check, layout versus schematic, antenna rule check, and formal verification are strictly followed. Both post-place-and-route simulation using SPICE

simulators and chip measurement results indicate the feasibility of ASIC implementation.

The remainder of this chapter is organized as follows. Sections 6.2, 6.3, 6.4 evaluate the performance in an Arria V FPGA of a BI creation system based on BIC, a BI-based query processing system based on BIQP, and a BI-based analytics system based on both BIC and BIQP, respectively. Section 6.5 presents the experimental results in an Arria V FPGA of BIE. Sections 6.6, 6.7, 6.8 discusses the FPGA results when connecting BIE to the BI creation system, the BI-based query processing system, and the BI-based analytics system, respectively. Moreover, the implementation of query processing system with BIE in both 180-nm bulk CMOS and 65-nm SOTB CMOS technology is also mentioned in Section 6.7. Finally, Section 6.9 concludes this chapter.

6.2 Bitmap Index Creation System Without An Encoder

6.2.1 Experimental Setup

In Chapter 3, we have proposed a BIC32K16 that can index 32,768 16-bit data using a three-channel direct memory access (DMA) module, an operation memory (OPM) module, a content addressable memory (CAM) module, and a query logic array (QLA) module. In this chapter, BIC32K16 is implemented in an Arria V FPGA and its performance is evaluated by the data I/O cost, indexing throughput, and energy efficiency in comparison with other CPU-based and GPU-based designs. The notation that will be used is defined in Table 6.1.

Figure 6.2 illustrates the block diagram of hardware system testing. Each experiment follows three main steps: (1) all test operations/keys and data are transferred from a host computer and temporarily stored in DDR3; (2) BIC32K16's DMA copies all data and all operations/keys from DDR3 to CAM and OPM, respectively. Subsequently, BIC32K16 indexes data by given operations/keys and sends all generated BI vectors to DDR3; (3) those vectors will be returned to the host computer for verification. Because (1) is for initialization and (3) is for

Table 6.1: The notation of proposed design.

Notation	Description
N_c	The number of words in CAM: N_c = 32,768
S_c	The size of each word: S_c = 2 bytes
N_k	The number of operations in OPM: maximum N_k = 2,048
S_k	The size of each operation: S_k = 32 bits
B	The number of batches
h	The number of generated BI vectors
w	The data bus width: w = 256 bits

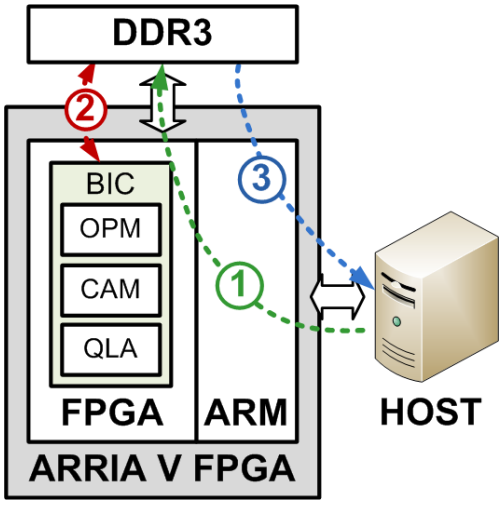


Figure 6.2: The block diagram of hardware system testing.

verification, the indexing time only counts (2), whose execution time is calculated by several internal counters integrated into BIC32K16.

Table 6.2 describes a group of five synthesis data sets, called DB, used to verify BIC32K16. Since the CAM size is 64 KB (32,768×16 bits = 512 Kbits = 64 KB), we assign the size of each batch to 64 KB length. The number of batches B in each test data varies from one to 8,192, which is equivalent to 64-KB and 512-MB data in total. Likewise, Table 6.3 shows the synthesis key sets, called KB. Each set may contain a few to several hundred operations. The number of operations N_k and the number of generated BI vectors h are extracted from a list of given queries, which is demonstrated in Section 3.2.

Table 6.2: The synthesis data sets of BIC32K16.

Test data	# of batches (B)	Size (KB)
DB1	1	64
DB2	4	256
DB3	128	2,048
DB4	1,024	65,536
DB5	8,192	524,288

Table 6.3: The synthesis operation/key sets of BIC32K16.

Test operations/keys	# of operations/keys (N_k)	# of BIs (h)
KB1	2	1
KB2	5	1
KB3	129	1
KB4	14	4
KB5	300	11
KB6	528	16

6.2.2 Experimental Results

6.2.2.1 Hardware Utilization

Table 6.4 clearly states the hardware consumption of BIC32K16 synthesized by Quartus II 16.0 design software. Adaptive logic modules (ALMs)—a basic building block of Arria V FPGA—together with memory bits are used to evaluate the resource utilization. Each ALM is composed of one 8-input combinational look-up table with four dedicated registers. Because one CAM bit costs 32 embedded memory bits, as large as 16-Mbit memory is required to build an entire 64-KB CAM. The other modules, such as OPM and QLA, utilize the remaining 0.14-Mbit memory. The operating frequency of BIC32K16 is approximately 129.5 MHz, which surpasses the required frequency of 100 MHz of mentioned in Section 6.1.

Table 6.4: The hardware utilization of BIC32K16.

Device	Arria V 5ASTFD5K3F40I3
CAM size	32,768×16-bit
ALMs	43,868 (25%)
– Lookup tables	76,361
– Registers	43,101
Memory (Mbits)	16.14 (73%)
Frequency (MHz)	129.5

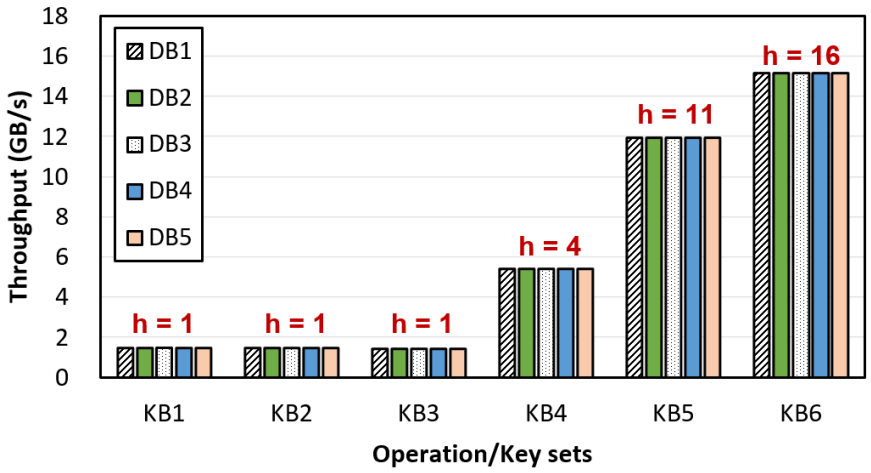


Figure 6.3: The indexing throughput.

6.2.2.2 Indexing Throughput

BIC32K16 produces each 32-Kbit BI vector of a 64-KB data set at every clock cycle. Figure 6.3 illustrates the measurement indexing throughput THR_{meas} of each index vector of BIC32K16 at 100-MHz frequency. THR_{meas} is defined in Equation (6.1), where $(N_c \times S_c) = 64$ KB is the CAM size, h is the number of generated BI vectors, and T_{meas} is the measurement index time directly obtained from the counters inside BIC32K16. As seen in Figure 6.3, at each operation/key set, the achieved throughput is almost unchanged, regardless of the variety of data size from DB1 (64 KB) to DB5 (512 MB).

Furthermore, THR_{meas} is more affected by h than N_k. Taking DB1 as an example, despite the sharp rises in the number of operations between KB3 ($N_k = 129$, $h = 1$) and KB1 ($N_k = 2$, $h = 1$), the difference of THR_{meas} changes as little

Table 6.5: The execution time of each BIC module.

Time (clock cycle)	Description
$t_{OPM} = \frac{N_k \times S_k}{w}$	Time to load operation/key sets to OPM
$t_{CAM} = (\frac{N_c \times S_c}{w}) \times 2$	Time to reset CAM and load new batch data to CAM
$t_{QLA} = N_k$	Time to process all operations/keys
$t_{OUT} = (\frac{N_c}{w}) \times h$	Time to output all BI vectors
$T_{theo} = t_{OPM} + (t_{CAM} + t_{QLA} + t_{OUT}) \times B$	Total theoretical index time

as 2.9%. However, in the case of KB4 ($N_k = 14$, $h = 4$), where four index vectors are created, THR_{meas} increases around 3.6× as compared to that of KB1. This is because at DB1 and KB1, only one BI vector is generated within $T_{meas} = 44.4$ μs. Thus, the corresponding THR_{meas} is around 1.48 GB/s. However, at KB4, four index vectors are produced within $T_{meas} = 48.3$ μs, so the corresponding THR_{meas} is 1.36 GB/s for 4 vectors, or approximately 5.44 GB/s for one vector. Similarly, when h becomes 16, the THR_{meas} reaches as high as 15.04 GB/s.

$$THR_{meas} = (\frac{1}{T_{meas}}) \times (N_c \times S_c) \times h \ \ \text{(GB/s)} \tag{6.1}$$

6.2.2.3 Data I/O Cost

To evaluate the I/O cost whenever BIC32K16 accesses DDR3 memory, the execution time of all individual modules of BIC32K16 is built up, as shown in Table 6.5. Specifically, t_{OPM} is the time to load all test operations/keys from DDR3 to OPM, t_{CAM} is the time to reset and load all test data from DDR3 to CAM, t_{QLA} is the time to process all operations, t_{OUT} is the time to return the BI vectors to DDR3, and T_{theo} is the theoretical index time. Because we have to eliminate the old values in CAM before loading new data, t_{CAM} is the sum of reset and load time. Moreover, the reset and load time are identical, thereby t_{CAM} is twice the load time.

Figure 6.4 illustrates the share of input time ($t_{CAM} + t_{OPM}$), process time

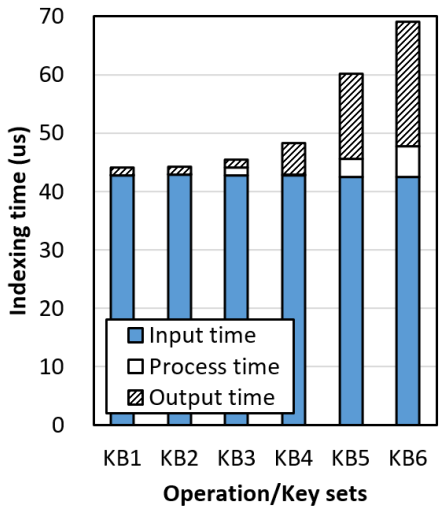


Figure 6.4: The timing distribution of each state at DB1.

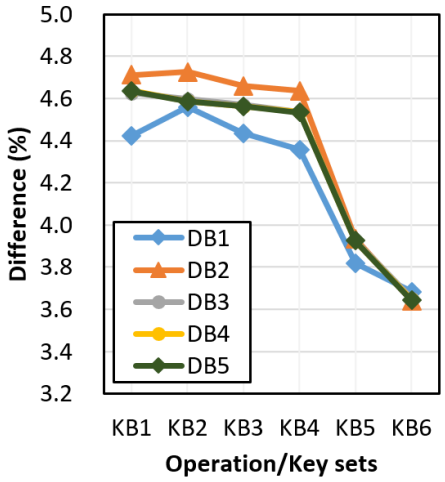


Figure 6.5: The difference between theoretical and measurement index time.

t_{QLA}, and output time t_{OUT} in the theoretical index time T_{theo} when test data is DB1 and test operations/keys vary from KB1 to KB6. It is easy to see that the input time and output time dominate the whole T_{theo}. Additionally, the I/O time relies on the system bus width w, or the DDR3 memory bandwidth, whereas t_{QLA} only depends on the FPGA frequency itself. In other words, the index performance of BIC32K16 depends largely on the DDR3 memory access time.

Figure 6.5 shows the difference of measurement index time T_{meas} and theoretical index time T_{theo} at various test data and operations/keys. In fact, T_{meas} is slightly (3.5% to 4.7%) larger than T_{theo}. This occurs because access to DDR3 requires several redundant cycles. In fact, DDR3 is constructed with several memory banks, each containing many columns and rows. To read or write data in DDR3, the memory controller firstly opens a certain row in a particular bank. The entire row of the memory array is then transferred into the corresponding row buffer. Upon completion, a column access command is performed to read or write data from or to the row buffer. Lastly, the row buffer must be written back to the memory array by a precharge command so that this bank is available for a subsequent row activation. The difference of 4.7% between T_{meas} and T_{theo} suggests that BIC32K16 achieves low data I/O cost [67, 68, 69, 70]. More significantly, using the formula of T_{theo}, we can relatively estimate T_{meas} with the maximum error of 4.7%.

6.2.2.4 Energy Efficiency

Figure 6.6 shows the comparison of energy efficiency between BIC32K16 and four platforms, including FastQuery [32], ParaSAIL [33], IMS [35], and GPU13 [40], which were implemented in CPU and GPU, respectively. The energy efficiency (J/GB) is calculated as a quotient of power dissipation (J/s) and measured index throughput (GB/s). The small value of energy efficiency of a certain system means that this system delivers high throughput but only consumes low power. In other words, if one system possesses higher value of energy efficiency than another, such system definitely requires high power in order to deliver the same throughput. The power dissipation parameters of CPU and GPU are extracted

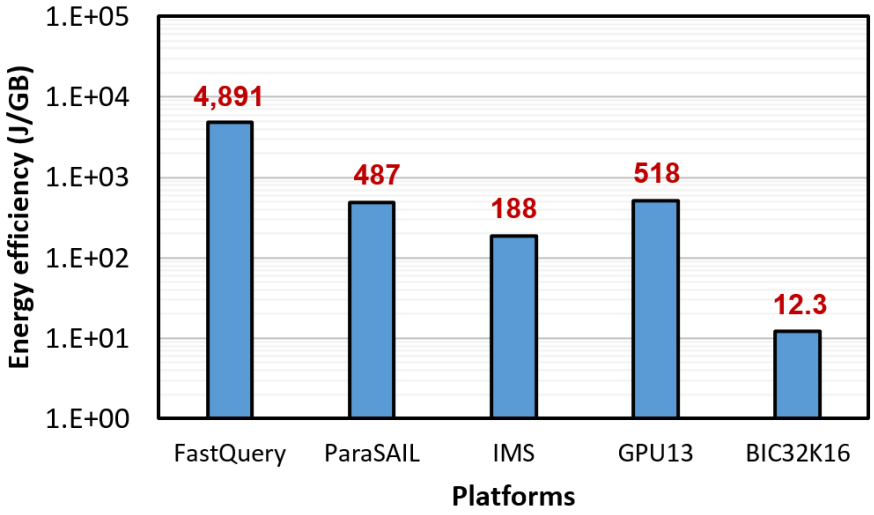


Figure 6.6: The comparison of energy efficiency.

from the product specifications provided by corresponding vendors, while the power dissipation of BIC32K16 was estimated by the PowerPlay Power Analyzer tool. The hardware specification of each platform is summarized in Table 6.6, while their details can be found in Section 2.1. As seen in Figure 6.6, FPGA-based BIC32K16 requires 15.3× and 43.1× lower energy than IMS and GPU13, respectively, to index 1-GB data.

Table 6.6: The platform description.

Platform	Description
FastQuery [32]	Hardware: 11,520 cores/960 AMD CPUs Throughput: 15.7 GB/s Power: 80 W/1 CPU or 76.8 KW/960 CPUs
ParaSAIL [33]	Hardware: 60 cores/1 Intel Co-CPU Throughput: 0.46 GB/s Power: 225 W/1 Co-CPU
IMS [35]	Hardware: 20,000 cores/834 Intel CPUs Throughput: 510 GB/s Power: 115 W/1 CPU or 95.9 KW/834 CPUs
GPU13 [40]	Hardware: 1,344 cores/ 1 NVIDIA GPU Throughput: 0.33 GB/s Power: 170 W/1 GPU
BIC32K16	Hardware: 1 Arria V FPGA Throughput: 1.48 GB/s Power: 18.2 W/1 FPGA

6.3 Bitmap-Index-Based Query Processing System Without An Encoder

6.3.1 Experimental Setup

In Chapter 4, we presented a BIQP that can answer all queries by using 32-Kbit BI vectors received from either BIC32K16 or other software DBMSs. BIQP is also composed of a three-channel DMA module, an OPM module, a bitmap index memory (BIM) module, and a QLA module. In this chapter, BIQP is implemented in an Arria V FPGA and its performance is evaluated by the data I/O cost, query processing throughput, and energy efficiency, as compared to other CPU-based and GPU-based designs. Table 6.7 summarizes the notation that is now used.

The block diagram of hardware system testing is illustrated in Figure 6.7. Each experiment follows three main steps: (1) all test operations and BI vectors are transferred from a host computer to DDR3; (2) BIQP's DMA copies all BI vectors and all operations from DDR3 to BIM and OPM, respectively. Upon

Table 6.7: The notation of proposed design.

Notation	Description
N_b	The number of BI vectors in each batch, maximum $N_b = 512$
S_b	The size of each BI vector, $S_b = 32{,}768$ bits
N_q	The number of operations inside OPM, maximum $N_q = 4{,}096$
S_q	The size of each operation, $S_q = 16$ bits
B	The number of batches
h	The number of query results, $h = 1$
w	The data bus width, $w = 256$ bits

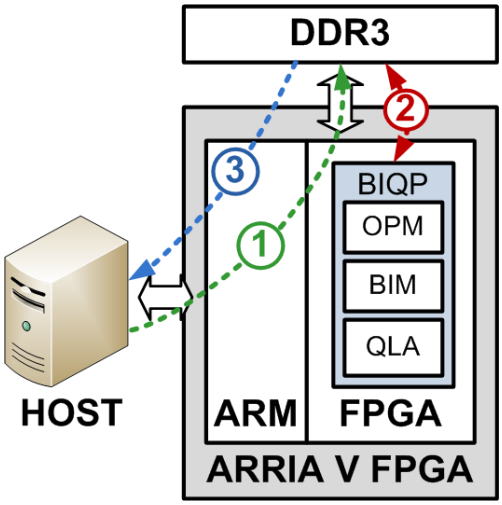


Figure 6.7: The block diagram of hardware system testing.

completing, BIQP processes all operations and returns all results to DDR3; (3) the bitmap results are sent back to the host computer for verification. Because (1) is for initialization and (3) is for verification, the processing time only counts (2) that is calculated by several internal counters integrated into BIQP.

To prepare for the verification, seven experiments including synthesis BI vectors and operation sets are created, as stated in Table 6.8. The experiments are numbered from EXP1 to EXP7. The number of batches (B), the number of BI vectors in each batch (N_b), and the number of operations (N_q), vary according to the particular test case. At the maximum setting, BIQP processes 1,822 operation with 512-MB indexes. It is reminded that N_q and h are extracted from a list of given queries, which is demonstrated in Section 4.2.

Table 6.8: The test cases of BIQP.

Test case	# of operations (N_q)	# of BIs/ batch (N_b)	# of batches (B)	Batch size
EXP1	6	4	1, 4, 16, 32, 128, 256	64 KB - 16 MB
EXP2	29	16		
EXP3	74	256		
EXP4	314	512	1, 4, 16, 32, 128, 256	2 MB - 512 MB
EXP5	633	512		
EXP6	1,186	512		
EXP7	1,822	512		

Table 6.9: The hardware utilization of BIQP.

Device	Arria V 5ASTFD5K3F40I3
BIM size	512×32,768-bit
ALMs	51,744 (29%)
– Lookup tables	91,273
– Registers	69,602
Memory (Mbits)	17.1 (80%)
Frequency (MHz)	109.3

6.3.2 Experimental Results

6.3.2.1 Hardware Utilization

Table 6.9 clearly states the hardware utilization of BIQP synthesized by Quartus II 16.0 design software. The synthesis results show that BIQP utilizes approximately 29% of ALM and 80% of the memory bits. Concretely, 16-Mbit memory are needed to construct a 32,768×512-bit BIM, whereas the remaining 1.1-Mbit memory are employed by other modules, such as OPM and QLA. The timing analysis also suggested that BIQP is capable of operating at 109.3 MHz, which is adequate for the minimum 100 MHz required by the hardware testing system.

6.3.2.2 Query Processing Throughput

Figure 6.8 illustrates the measurement processing throughput (THR_{meas}) of BIQP, including memory access time, at 100-MHz operating frequency. THR_{meas} is cal-

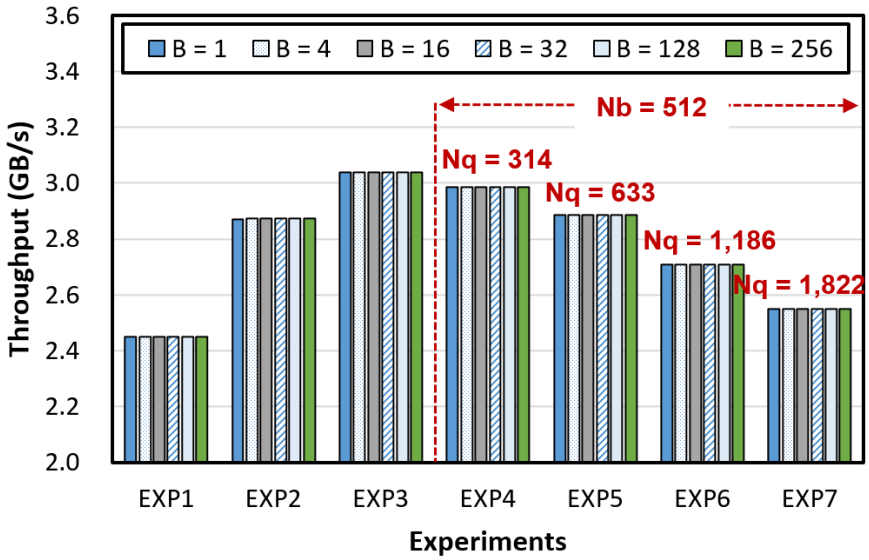


Figure 6.8: The query processing throughput.

culated as the number of BI vectors that can be solved in every second, as illustrated in Equation (6.2). The maximum number of BI vectors/batch is 512, or the batch size can reach up to 2 MB. Roughly speaking, THR_{meas} mainly depends on the total size of incoming BI vectors and the number of operations N_q. For example, from EXP1 to EXP3, because N_b varies from 4 to 256, THR_{meas} correspondingly increases. However, from EXP4 to EXP7, N_b is unchanged and THR_{meas} gradually decreases due to the increase of N_q—from 314 to 1,822 operations. Depending on the specific experiments, different throughputs can be achieved, i.e. the minimum and maximum throughputs in our experiments are 2.45 GB/s and 3.04 GB/s, respectively.

$$THR_{meas} = (\frac{1}{T_{meas}}) \times (N_b \times S_b) \quad \text{(GB/s)} \tag{6.2}$$

6.3.2.3 Data I/O Cost

Table 6.10 shows the theoretical execution time, in clock cycles, of all the individual modules of BIQP. Suppose that t_{OPM} is the time to load all test operations from DDR3 to OPM, t_{BIM} is the time to load all BI vectors from DDR3 to BIM, t_{QLA} is the time to process operations, t_{OUT} is the time to return the results to

Table 6.10: The execution time of each BIQP module.

Time (clock cycle)	Description
$t_{OPM} = \frac{N_q \times S_q}{w}$	Time to load test operations to OPM
$t_{BIM} = \frac{N_b \times S_b}{w}$	Time to load a test batch to BIM
$t_{QLA} = N_q$	Time to process N_q operations
$t_{OUT} = \frac{S_b}{w}$	Time to output all bitmap results
$T_{theo} = t_{OPM} + (t_{BIM} + t_{QLA} + t_{OUT}) \times B$	Total theoretical query processing time

DDR3, and T_{theo} is the theoretical processing time. The definition of all parameters, e.g. N_q and S_q, can be found in Table 6.7. Apparently, t_{OPM}, t_{BIM}, and t_{OUT} depend solely on the system bus width w, or the bandwidth of the DDR3 memory, whereas t_{QLA} only depends on the FPGA frequency itself.

Figure 6.9 illustrates the share of input time ($t_{BIM} + t_{OPM}$), process time t_{QLA}, and output time t_{OUT} in T_{theo} in different experiments when $B = 1$. It is easy to see that the input time and output time dominate the entire T_{theo}. Furthermore, t_{QLA} linearly increases only according to N_q. Nonetheless, that time plays an insignificant role in T_{theo}. Reducing the I/O time, therefore, has a beneficial effect on the overall processing time.

Figure 6.10 shows the increase of measurement time T_{meas}, from 3.2% to 3.4%, as compared to T_{theo} for various test data and keys. The reason is that access to DDR3 requires several redundant cycles. However, the slight difference between both execution times proves that BIQP achieves low data I/O cost, as compared to other works [67, 68, 69, 70]. Furthermore, using the formula of T_{theo}, we can relatively estimate T_{meas} with the maximum error of 3.4%.

6.3.2.4 Energy Efficiency

Figure 6.11 draws the comparison of energy efficiency between BIQP and two platforms, including CPU-based IMS [35] and GPU-based GPU11 [39]. It is reminded that a small value of energy efficiency of a certain system means that such system

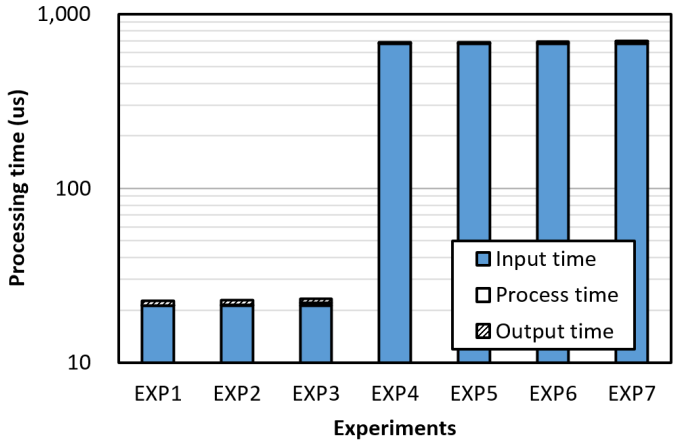


Figure 6.9: The timing distribution of each state $B = 1$.

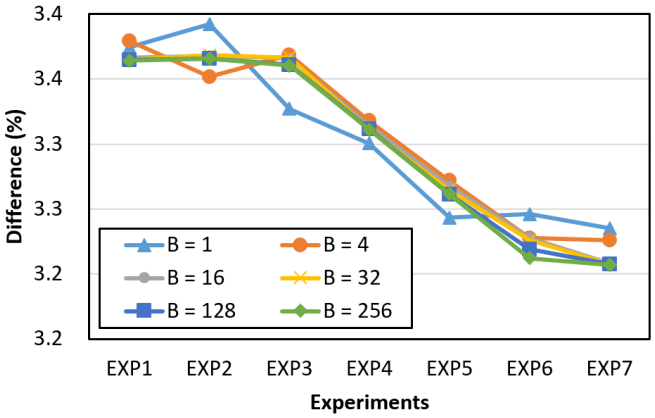


Figure 6.10: The difference between theoretical and measurement time of BIQP.

delivers high throughput but only consumes low power. The power dissipation parameters of CPU and GPU were extracted from the product specifications provided by corresponding vendors, while the power dissipation of BIC32K16 was estimated by the PowerPlay Power Analyzer tool. The hardware specification of each platform is summarized in Table 6.11, while their details can be found in Section 2.1. As seen in Figure 6.11, an FPGA-based BIQP consumes 223× and 31× less energy than CPU-based IMS and GPU-based GPU11, respectively, to process 1-GB BI vectors.

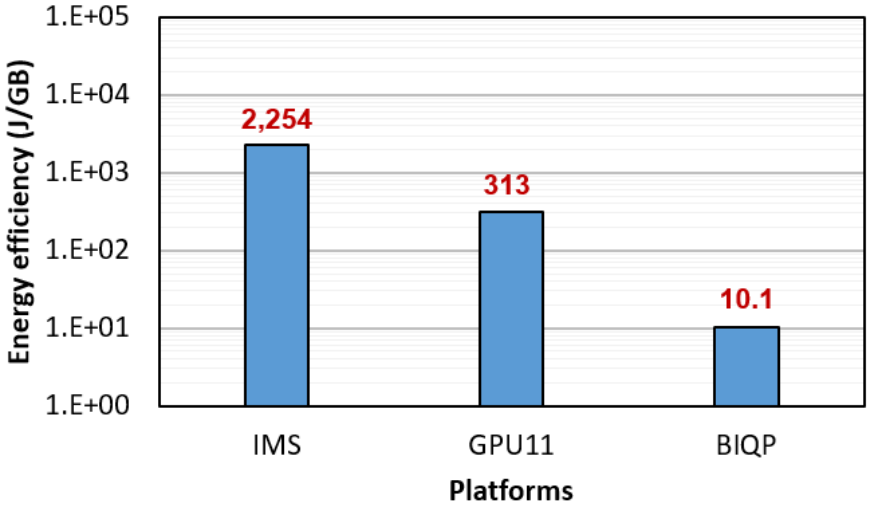


Figure 6.11: The comparison of energy efficiency.

Table 6.11: The platform description.

Platform	Description
IMS [35]	Hardware: 20,000 cores/834 Intel CPUs Throughput: 42.5 GB/s Power: 115 W/1 CPU or 95.9 KW/834 CPUs
GPU11 [39]	Hardware: 240 cores/ 1 NVIDIA GPU Throughput: 0.65 GB/s Power: 204 W/1 GPU
BIQP	Hardware: 1 Arria V FPGA Throughput: 2.45 GB/s Power: 25.2 W/1 FPGA

6.4 Bitmap-Index-Based Analytics System Without An Encoder

6.4.1 Experimental Setup

Figure 6.12 proposes a BI-based data-analytics system built from a BIC32K16 and a BIQP. The system can be implemented in two separate Arria V SoC development boards, where the first one indexes data given by a host computer and the second one processes the queries with those generated BI vectors. It then also returns the results to the host computer. In other words, an indexing system

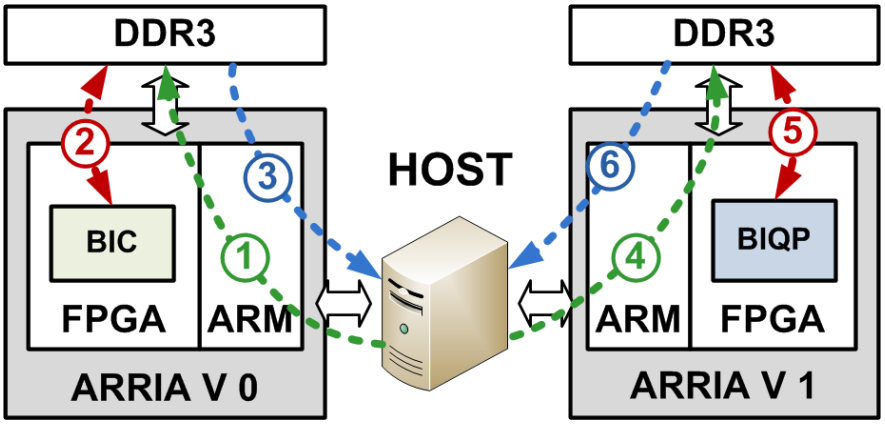


Figure 6.12: The BI-based data-analytics system.

using BIC32K16 is deployed on the first board, whereas a query processing system using BIQP is deployed on the second board. The bus width and operating system of both designs are 256 bits and 100 MHz, respectively. The procedure of each subsystem is summarized as follows. Because (1), (3), (4), and (6) can be improved according to communication means, such as 10-Gbps Ethernet or PCI-Express, the total analytics time is only counted (2) and (5).

- In the first board: (1) all test operations/keys and test data are transferred from a host computer and temporarily stored in DDR3; (2) BIC32K16 loads all test keys and data to its OPM and CAM using the DMA mechanism; BIC32K16 generates the BI vectors and then stores all of them in DDR3 memory; (3) all BI vectors are sent to a host computer for storage.

- In the second board: (4) all test operations and BI vectors are transferred from a host computer and temporarily stored in DDR3; (5) BIQP loads all test operations and indexes to its OPM and BIM using the DMA mechanism; BIQP processes the queries and returns the results to DDR3 memory; (6) all bitmap results are sent to a host computer for storage.

In order to evaluate the index and query performance of a BI-based analytics system, a synthesis dataset includes four 32,768×16-bit attributes and five random operations/keys are generated for indexing. Four generated BI vectors, corresponding to four attributes, are then put into BIQP for answering the query with six given operations.

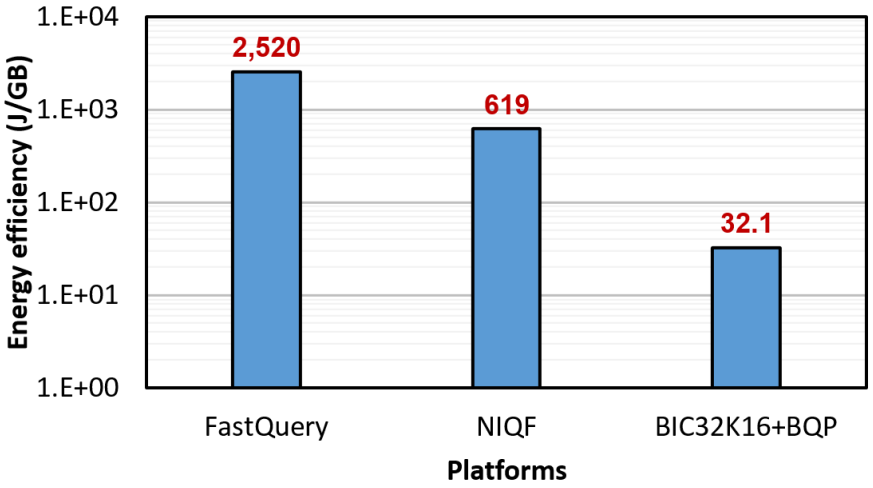


Figure 6.13: The comparison of energy efficiency.

6.4.2 Experimental Results

Suppose that T_I, T_Q, and T_A are the index time, the query process time, and the total analytics time, respectively. The experimental results show that $T_I = 177.5$ μs and $T_Q = 2.7$ μs. Therefore, the total analytics time $T_A = T_I + T_Q = 180.2$ μs. It is easy to see that T_I mainly contributes to T_A, since indexing is the most expensive task in the analytics system. The analytics throughput is defined as the total data analyzed in a time unit, and is approximately 1.35 GB/s. Figure 6.13 draws the comparison of energy efficiency between the proposed analytics system and two other CPU-based platforms. The platform description of each one is summarized in Table 6.12 and their details can be found in Section 2.1. Apparently, the FPGA-based design attains lower energy consumption, 77× and 19× compared to FastQuery and NIQF, respectively.

Table 6.12: The platform description.

Platform	Description
FastQuery [32]	Hardware: 2,304 cores/192 Intel CPUs Throughput: 6.2 GB/s Power: 80 W/1 CPU or 15.36 KW/834 CPUs
NIQF [34]	Hardware: 240 cores/ 1 NVIDIA GPU Throughput: 0.105 GB/s Power: 65 W/1 GPU
BIC32K16 and BIQP	Hardware: 2 Arria V FPGAs Throughput: 1.35 GB/s Power: 43.4 W/2 FPGAs

6.5 Bitmap Index Encoder

6.5.1 Experimental Setup

The performance of BIE is evaluated by the performance of both PE and MPE in an ASIC and an FPGA, respectively. Specifically, various PEs whose sizes L range from 4-bit to 4-Kbit were implemented in 180-nm CMOS technology for the evaluation of resource utilization and operating frequency. Corresponding MPEs were then implemented in an Intel Arria V FPGA to measure the encoding throughput, as well as prepare for the integration in a BI-based analytics system.

6.5.2 Experimental Results

6.5.2.1 Priority Encoder

The post-place-and-route simulation results at 1.8 V of various PEs can be seen in Table 6.13. The three main findings based on these results are listed below:

• Firstly, 1D-to-2D conversion usage evidently improves the deterioration of performance at large PE sizes. In fact, assume that DEC_L is the percentage decrease of operating frequency (FREQ) between PE_L and $\mathrm{PE}_{L/2}$. It is easy to see the major difference between DEC_8 and DEC_{16}, whose circuits are directly built from the truth tables. On the contrary, from $\mathrm{DEC}_{64_{(4)}}$ to $\mathrm{DEC}_{4K_{(4)}}$, the mean value is approximately 11%, whenever the PE size is doubled.

• Secondly, the look-ahead signal usage contributes to FREQ enhancement.

Using as an example of $PE64_{(8n)}$ and $PE64_{(8)}$, the FREQ of the latter increases approximately 5.2%. The improvement is not as much as the preliminary analysis, because in the real implementation, we applied flat-design synthesis in each PE instead of hierarchy-design. The difference between the two types of designs is that in hierarchy-design, each hierarchical unit is placed separately, and then the units are combined to form the circuit. By contrast, in flat-design, the borders between some or all logical hierarchies are dissolved, and the flattened logic is placed together. Flattening a design allows a significantly better optimization of performance and power during placement and subsequent design steps. In the preliminary analysis, we assumed that PE was synthesized in hierarchy-design, so that the total delay of PE64 was the sum of all the individual component delays. However, in the real implementation, we configured the synthesis tool in flat-design (with an aggressive timing constraint) in order to achieve the optimum results. For those reasons, the performance between the preliminary analysis and those in Table II is different.

• Thirdly, the organization of PE_N and PE_M clearly affects the outcome of a large-sized PE. For example, $PE64_{(4)}$ achieves the highest FREQ, while $PE64_{(16)}$ obtains the lowest FREQ, which is identical to the preliminary analysis above. Therefore, only large-sized PEs with $M = 4$ are compared with other previous works.

In comparison with RefA [48], which was simulated in 150-nm CMOS technology, the current designs gradually become better when PE sizes vary from 32-bit to 256-bit. As seen in Figure 6.14(a), FREQ of $PE32_{(4)}$ is only 1.3 times as high as that of RefA [48], whereas at a PE size of 256-bit, the difference of FREQ increases to 4.7 times. Moreover, according to Figure 6.14(b), the transistor count of $PE32_{(4)}$ and $PE256_{(4)}$ are only 0.94 times and 0.73 times, as compared to those in RefA [48].

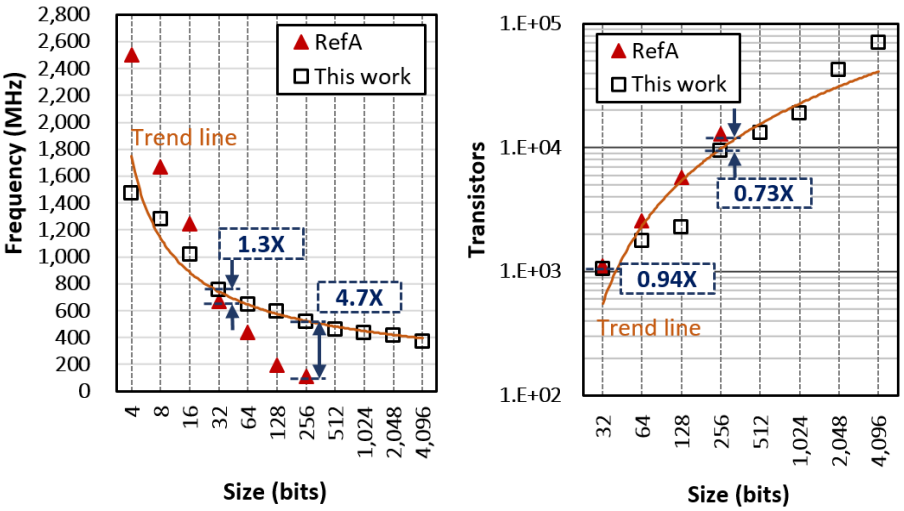
In addition, Figure 6.15 depicts the comparison of FREQ and transistor count between two works in 180-nm CMOS technology, when PE sizes vary from 64-bit to 2,048-bit. Because the architecture of PE4, PE8, and PE16 are identical in both works, their FREQs and transistor count are unchanged. In RefB [51], PE64 shares the same architecture with $PE64_{(8n)}$, where $(M, N) = (8, 8)$ is a non-optimal configuration and the look-ahead signal is unused. In a similar vein,

Table 6.13: The simulation results of proposed PEs.

Design	PE_N / PE_M	FREQ (MHz)	DEC_L (%)	Transistors
PE4	–/–	1,470	–	48
PE8	–/–	1,282	-12.7	202
$PE16_{(4)}$	PE4 / PE4	909	–	406
PE16	–/–	1,020	-20.4	396
$PE32_{(8)}$	PE4 / PE8	645	–	626
$PE32_{(4)}$	PE8 / PE4	757	-25.2	1,046
$PE64_{(16)}$	PE4 / PE16	520	–	1,702
$PE64_{(8n)}$	PE8 / PE8	526	–	2,128
$PE64_{(8)}$	PE8 / PE8	555	–	2,464
$PE64_{(4)}$	PE16 / PE4	649	-14.2	1,768
$PE128_{(16)}$	PE8 / PE16	480	–	3,804
$PE128_{(8)}$	PE16 / PE8	510	–	3,536
$PE128_{(4)}$	$PE32_{(4)}$ / PE4	595	-8.3	2,292
$PE256_{(4)}$	$PE64_{(4)}$ / PE4	520	-12.6	9,448
$PE512_{(4)}$	$PE128_{(4)}$ / PE4	462	-11.1	13,256
$PE1K_{(4)}$	$PE256_{(4)}$ / PE4	434	-6.1	18,960
$PE2K_{(4)}$	$PE512_{(4)}$ / PE4	416	-4.1	42,722
$PE4K_{(4)}$	$PE1K_{(4)}$ / PE4	370	-11.1	70,502

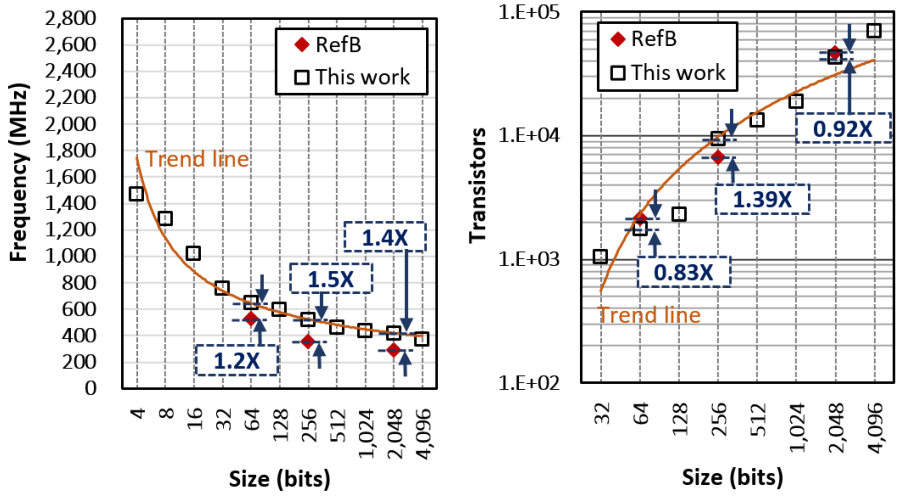
PE256 is constructed by two PE16s. PE2K, however, is formed by 32 PE64s operating in parallel, together with one central PE32.

As seen in Figure 6.15(a), the FREQs of $PE64_{(4)}$, $PE256_{(4)}$, and $PE2K_{(4)}$ are 1.2 times, 1.5 times, and 1.4 times as high as those in RefB [51]. When it comes to logic utilization, $PE64_{(4)}$ and $PE2K_{(4)}$ cost fewer transistors than PE64 and PE2K, respectively, as depicted in Figure 6.15(b). However, the power consumption of $PE64_{(4)}$ and $PE2K_{(4)}$ are 20.6% and 29.9% as high as those in RefB [51]. Nevertheless, the resource and power consumption will be considered for future work, since this work mainly concentrates on the high-performance architecture.



(a) Size vs frequency. (b) Size vs transistor count.

Figure 6.14: The comparison with RefA [48].



(a) Size vs frequency. (b) Size vs transistor count.

Figure 6.15: The comparison with RefB [51].

Table 6.14: The comparison of several MPEs.

Design	RefC	RefB	MPE8	RefC	RefB	MPE64	RefD	RefB	MPE2K
Device	Stratix FPGA			Stratix FPGA			Cyclone IV FPGA		
L (bits)	8			64			2,048		
LEs	33	23	23	208	213	237	19,579	4,464	5,354
FREQ (MHz)	236	369	369	55	95	113.4	50	71	83.9
FREQ×L (Gbps)	1.9	2.9	2.9	3.5	6.1	7.3	102.4	145.4	171.8

6.5.2.2 Multi-match Priority Encoder

Table 6.14 shows three experiments of MPE, between our designs and three previous ones–RefB [51], RefC [46], RefD [49]. Since RefB, RefC, and RefD were verified in an Altera Stratix EP1S10F780C6 and Cyclone IV EP4CE115F29C7, we synthesized our 8-bit MPE (MPE8), 64-bit MPE (MPE64), and 2,048-bit MPE (MPE2K) in the same FPGA devices, in order to draw a fair comparison. All of them are based on PE8, $PE64_{(4)}$, and $PE2K_{(4)}$, respectively. The results of logic elements (LEs) and FREQ are also correspondingly estimated by Altera Quartus II 13.0 and TimeQuest Timing Analyzer.

As seen in Table 6.14, our designs considerably surpass the others in terms of (FREQ×L), where L is the number of bits of MPE. Apparently, MPE8, MPE64, and MPE2K produce (FREQ×L) of 1.5 times (2.9/1.9), 1.7 times (6.1/3.5), and 2.5 times (252.9/102.4) as high as that of RefC and RefD, respectively. In terms of resource allocation, although RefC did not include the ENC circuit, they still consume as much as LEs in comparison with MPE8 and MPE64, respectively. RefD also requires the number of LEs to be 3.6 times larger than MPE2K, which is likely to cause severe problems of resource utilization and power consumption, since N keeps increasing. In comparison with our previous work RefB, which excluded the utilization of a look-ahead signal and optimum (M, N) selection, (FREQ×L) of MPE64 and MPE2K are 1.19 times (7.3/6.1) and 1.18 times (171.8/145.4) as high as those of RefB.

The resource efficiency of this work and three others are illustrated in Figure 6.16. The resource efficiency (LE/Gbps) is calculated as the quotient of the resource consumption (LE) and the parameter of (FREQ×L) (Gbps). The smaller

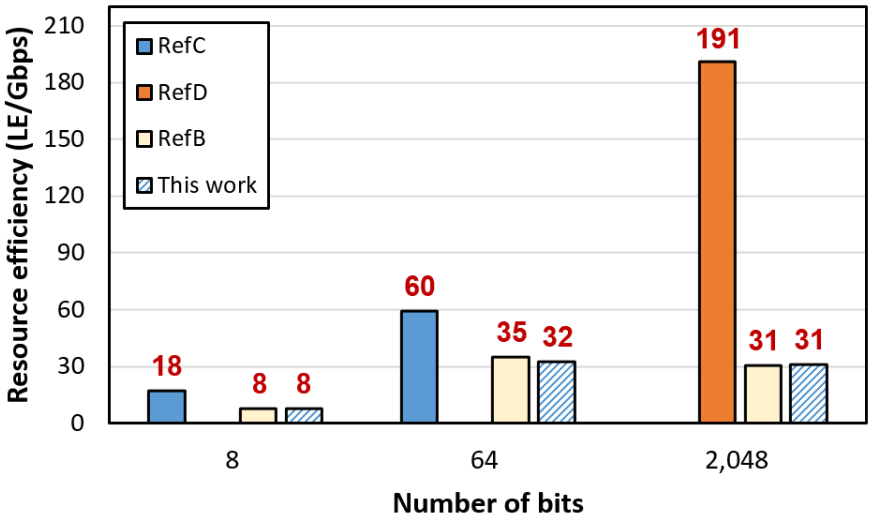


Figure 6.16: The comparison of resource efficiency.

the resource efficiency is, the lower the energy efficiency becomes. As seen in Figure 6.16, the proposed MPE requires fewer resources to encode the same bit data in comparison with other state-of-the-art works.

MPE2K is used to construct 32-Kbit BIE. According to the analysis in Section 5.2, BIE requires one initial cycle t_{int} to get the 2-Kbit segment and K cycles t_{seg} to obtain K matching bits in this segment. Moreover, it takes two additional cycles t_{wait} to receive a new 2-Kbit segment. Assume that T_{ENC} is the execution time of BIE. Because T_{ENC} only relies on the FPGA clock cycle itself, the measurement time is identical to the theoretical time. As a result, the best and worst encoding time are $t_{ENCb} = 46$ and $t_{ENCw} = 32{,}814$ clock cycles, respectively, as calculated in Equation (6.3). In other words, the maximum and minimum throughputs at 100-MHz operating frequency become 66.34 Gbps and 95.23 Mbps, respectively.

$$t_{ENC} = (\frac{32,768}{2,048}) \times (t_{int} + t_{seg}) + (\frac{32,768}{2,048} - 1) \times t_{wait} \quad (cycles) \tag{6.3}$$

$$t_{ENCw} = 16 \times (1 + 2,048) + 15 \times 2 = 32,814 \quad (cycles)$$

$$t_{ENCb} = 16 \times (1 + 0) + 15 \times 2 = 46 \quad (cycles)$$

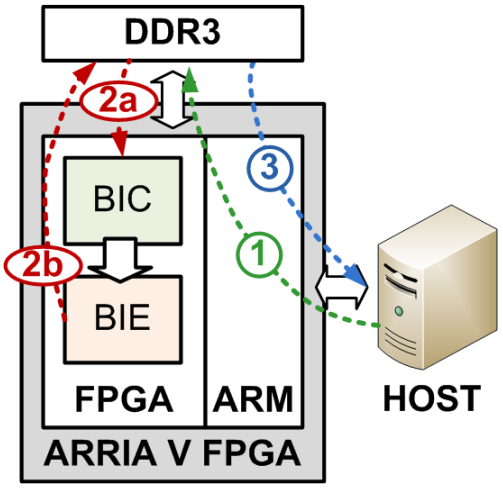


Figure 6.17: The block diagram of hardware system testing.

6.6 Bitmap Index Creation System With An Encoder

6.6.1 Experimental Setup

BIE is connected to BIC32K16 to output the matching positions of the BI vectors, as shown in Figure 6.17. The primary component of this BIE is a 2,048-bit multi-match priority encoder (MPE2K), whose detailed architecture is described in Section 5.2. Because the length of BI vector is larger than 2,048 bits, a multiplexer is added to divide the BI vectors into 16 2-Kbit segments. Each segment is encoded by BIE in turn, then all encoded results are returned to DDR3 memory using DMA mechanism for low I/O cost. An Arria V FPGA is employed to verify the performance of the combination of BIC32K16 and BIE.

6.6.2 Experimental Results

Table 6.15 draws the comparison of hardware consumption between a combination of BIC32K16 and BIE and a CAM-based information detection system (IDS) [43]. It is easy to see that in this combination system, the ALM utilization increases around 6% and the operating frequency is reduced around 8%, compared to those of the indexing system mentioned in Section 6.2. Nonetheless, this achieved

Table 6.15: The comparison of hardware utilization.

	IDS [43]	BIC32K16 + BIE
Device	Cyclone IV EP4CE115F29C7	Arria V 5ASTFD5K3F40I3
CAM size	2,048×8-bit	32,768×16-bit
ALMs	—	55,288 (31%)
– Lookup tables	77,205	87,940
– Registers	31,094	43,822
Memory (Mbits)	3.00	16.14
Frequency (MHz)	50	120.1

Table 6.16: The execution time of BI creation system with BIE.

Time (clock cycle)	Description
$t_{OUT} = t_{ENC} \times h$	Time to output all encoded results
$T_{theo} = t_{OPM} + (t_{CAM} + t_{QLA} + t_{OUT}) \times B$	Total theoretical index time

frequency is still adequate for the current 100-MHz testing system. In comparison with IDS, despite the difference of FPGAs, we could design a CAM that was 32× larger, but only employed an additional 14% of lookup tables and 40% of registers, respectively. This is due to the enhanced structure of the BIC32K16 component, especially the QLA module, as well as the 1D-to-2D-conversion-based BIE component.

Because BIE returns the encoded positions at the rate of one match per clock cycle, the time to output all matches is proportional to the number of matches in a BI vector. Suppose that t_{OUT} is the encoding time, t_{OUT} is a product of h and t_{ENC} that are mentioned in Equation (6.3) of Section 6.5. Following this, the best and worst encoding time become (46×h) and (32,814×h), respectively. The output time t_{OUT} and theoretical index time T_{theo}, in clock cycles, are summarized in Table 6.16.

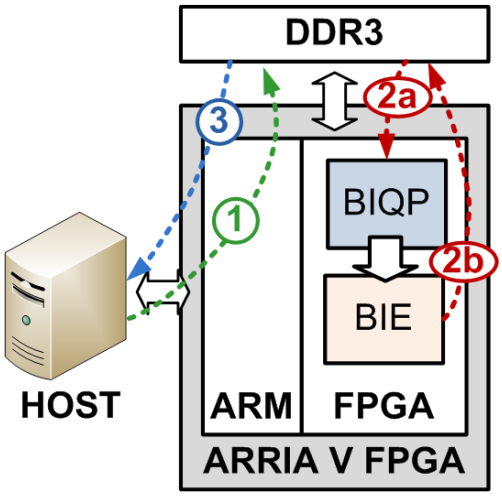


Figure 6.18: The block diagram of hardware system testing.

6.7 Bitmap-Index-Based Query Processing System With An Encoder

6.7.1 Experimental Setup

The combination of BIQP and BIE have been successfully implemented in an Arria V FPGA and then in two ASICs using 180-nm bulk CMOS and 65-nm SOTB CMOS technology. A 2-Kbit BIE is connected to BIQP to output the matching positions of the query results, as shown in Figure 6.18. The detailed architecture of this BIE is described in Section 5.2. Because of the 32-Kbit query result, a multiplexer is added to select each of 16 2-Kbit segments in the result. All encoded results are then returned to DDR3 memory, using the DMA mechanism for low I/O cost.

The ASIC implementation followed the digital circuit design flow using the Synopsys and Cadence design tools. Since BIQP contains up to 512 32-Kbit BI vectors, or a BIM size of 16 Mbits, ASIC implementation of such a design is very challenging. For this reason, only two small-sized BQPs, containing a 256×16-bit BIM and 256×32-bit BIM, are implemented in 180-nm bulk CMOS and 65-nm SOTB CMOS technologies, respectively. It is noted that the BIM size in 65-nm CMOS is double of that in 180-nm CMOS. The number of operations/keys

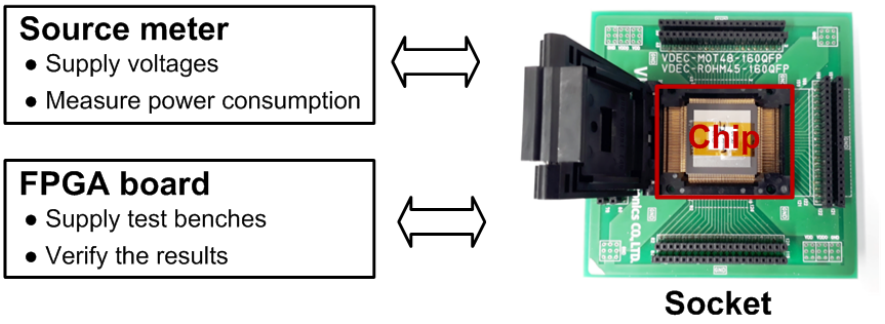


Figure 6.19: The illustration of the chip testing method.

Table 6.17: The hardware utilization.

Device	Arria V 5ASTFD5K3F40I3
ALMs	62,527 (35%)
– Lookup tables	102,856
– Registers	70,322
Memory (Mbits)	17.1 (80%)
Frequency (MHz)	106.8

of both designs are also reduced to 32, instead of 4,096 as implemented in an FPGA.

Figure 6.19 describes our testing method of the chip using a source meter and an FPGA board. First, a source meter is employed to supply the necessary voltages to the chip. The active current and power consumption can also be observed in the source meter. Second, an FPGA board provides the chip with several test benches that contains synthesis operations/keys and BI vectors. The results returned from the chip are directly compared with the expected results initialized in the FPGA. If any mismatch occurs, an alarm will be triggered instantly. Moreover, the operating frequency is adjusted by the FPGA's phase-lock loop to determine the maximum value. Each experiment was repeated ten million times to guarantee the stability.

6.7.2 Experimental Results

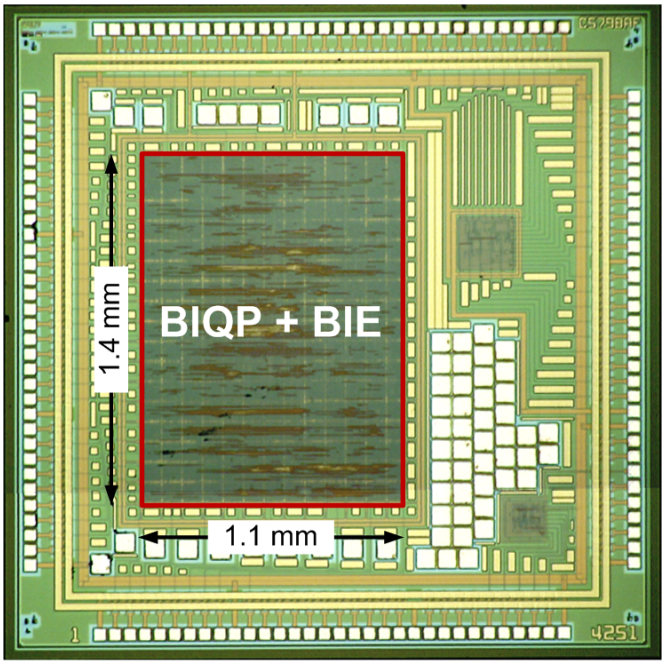
Table 6.17 draws the hardware utilization of the combination of BIQP and BIE. In this combination, the number of ALMs increases approximately 6% and the

Table 6.18: The execution time of BI-based query processing system with BIE.

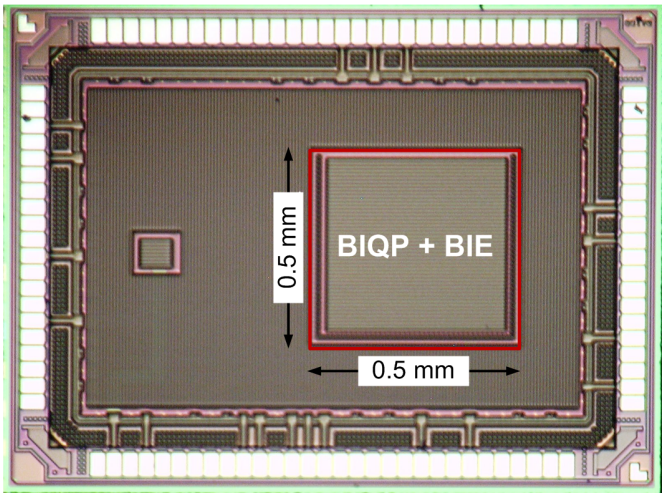
Time (clock cycle)	Description
$t_{OUT} = t_{ENC} \times h$	Time to output all encoded results
$T_{theo} = t_{OPM} + (t_{BIM} + t_{QLA} + t_{OUT}) \times B$	Total theoretical processing time

operating frequency is slightly reduced by 3%, compared to those of a BIQP-based query processing presented in Section 6.3. Nonetheless, the achieved frequency of this combination is still adequate for the current 100-MHz testing system. Table 6.18 shows the theoretical execution time, in the clock cycles, of t_{OUT} and T_{theo}, where t_{OUT} is the encoding time of BIE and T_{theo} is the total theoretical processing time. Because BIE returns the encoded positions at the rate of one match per clock cycle, the time to output all matches is proportional to the number of matches in a BI vector. Suppose that t_{OUT} is the encoding time, since $h = 1$, t_{OUT} is equal to t_{ENC} that are mentioned in Equation (6.3) of Section 6.5. Specifically, the best and worst processing time are 46 and 32,814 clock cycles, respectively.

Two proof-of-concept chips of the combination of BIQP and BIE are shown in Figure 6.20 and the their measurement results are stated in Table 6.19. The 180-nm chip was fully operational at 40 MHz and consumed 26.14 mW with a supply voltage of 1.8 V. The 65-nm chip that contains the double-sized-BIM was fully operational at 45 MHz and consumed only 8.29 mW with a supply voltage of 1.2 V. More significantly, the performance of the 65-nm chip can be improved by applying appropriate bias voltages.



(a) In 180-nm bulk CMOS technology.



(b) In 65-nm SOTB CMOS technology.

Figure 6.20: The proof-of-concept chips.

Table 6.19: The measurement results.

	180-nm bulk CMOS	65-nm SOTB CMOS
Data width w (bits)	16	8
BIM size	16×256-bit	32×256-bit
OPM size	32×16-bit	32×16-bit
# of transistors ($\times 10^3$)	345	654
Area (mm^2)	1.4×1.1	0.5×0.5
Supply voltage (V)	1.8	1.2
Frequency (MHz)	40	45
Power (mW)	26.14	8.29

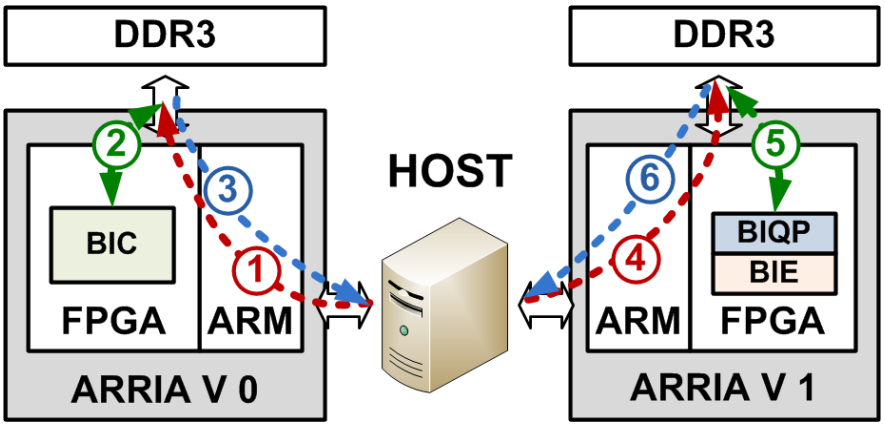


Figure 6.21: The block diagram of hardware system testing.

6.8 Bitmap-Index-Based Analytics System With An Encoder

As mentioned earlier in Section 6.4, a BI-based data-analytics system is built from a BIC32K16 and a BIQP. To output the matching positions of the result register, BIE is connected to BIQP. Figure 6.21 depicts the architecture of this system. Each bitmap resulting from BIQP is encoded by BIE, before being returned to DDR3 memory using the DMA mechanism. Because the operating frequency of a BIC32K16 and a combination of BIQP and BIE are 129.5 MHz and 106.8 MHz, respectively, those systems can be well-suited to the 100-MHz requirement of the testing system. The performance of each system, therefore, is similar to those of BIC32K16 and the combination of BIQP and BIE.

6.9 Summary

This chapter has evaluated the performance including data I/O cost, processing throughput, and energy efficiency of BIC, BIQP, and BIE in an Intel Arria V FPGA. A BI-based analytics system is then proposed by combining a 32,768×16-bit BIC with a BIQP. The combination of BIC with BIE and BIQP with BIE are also verified in an Arria V FPGA and in two ASICs using a 180-nm bulk CMOS and a 65-nm SOTB CMOS process. More significantly, the experimental results prove that the FPGA-based design always requires low energy consumption, compared to other CPU-based and GPU-based designs.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

As big data continues to develop, the ability to process large amounts of data and distill valuable information from it has become increasingly important for both enterprise businesses as well as research scientists. Traditional computing platforms, mainly based on CPUs, can no longer catch up with the massive growth of data. Hybrid computing platforms that combine CPUs with either GPUs or FPGAs to boost the parallel processing, has attracted great interest. Despite being extremely powerful, GPUs consume a considerable amount of power. Therefore, its computing efficiency is much lower than that of FPGAs. As a result of this drawback, FPGAs have become more popular in many huge data centers, such as those of IBM [71] and Microsoft [72].

Data indexing plays a major role in data analytics because an efficient indexing method is capable of minimizing the entire query time. Depending on the database system, data model, and workloads, different indexing methods are employed, such as B^+-tree and hash that are the two most common approaches in OLTP databases. However, in OLAP databases, most queries are highly complex and involve aggregations, in addition to the limited execution time. Therefore, BI is applied instead. BI has attracted a variety of studies, because it can solve those expensive queries effectively by using only basic logical operators, such as AND, OR, and NOT. Moreover, BI is amenable to efficient in-memory and distributed processing [32, 35].

7.1.1 Achievements

This dissertation originally proposes a BI-based data analytics system, which is implemented in an FPGA and ASIC. First, BI is confirmed to be perfect for hardware implementation because it supports parallel processing and only requires basic bitwise logic operators to answer the complex queries. Second, three components – BIC, BIQP, and BIE – are designed from scratch to index data, process queries, and encode bitmap values, respectively. Each component can operate independently or combine to form a BI creation system, a BI-based query processing system, and a BI-based data-analytics system. In the analytics system, BIC mainly constructed by a RAM-based CAM indexes all data received from the host computer. All of the BI vectors are then sent to BIQP to resolving the queries. Finally, the BIQP outputs are encoded by BIE before returning to the host computer.

The architectures of the proposed designs meet the criteria of scalability and parallel processing capability, while their performance is evaluated by data I/O cost, processing throughput, and energy efficiency.

- **Scalability**: the size of BIC, BIQP, and BIE can be easily adjusted by adding or removing correspondent logic circuits. For example, BIC64K8 and BIC32K16, which index 65,536 8-bit words and 32,768 16-bit words, are created by slightly modifying the CAM and QLA size. Likewise, BIQP depth and width are varied by inserting additional memory blocks and logic circuits into BIM and QLA, respectively. The 1D-array to 2D-array conversion is also used to construct a BIE that can deal with different input size effectively.

- **Parallel processing**: BIC, BIQP, and BIE can process data in parallel at the speed of a clock cycle. For example, BIC64K8 and BIC32K16 can index as many as 65,536 8-bit words or 32,768 16-bit words simultaneously at every cycle, BIQP can process 32-Kbit BI vectors at the rate of one operation/cycle, and BIE can detect and encode the matching positions in a bitmap result at the rate of one match/cycle.

- **Data I/O cost**: BIC, BIQP, and BIE can retrieve the test data and test

operations directly from the external memory using the DMA mechanism. Therefore, the I/O cost is likely to reach as high as the memory bandwidth. In fact, the experimental results in an Intel Arria SoC FPGA kit show that more than 95% of the DDR3 memory bandwidth are utilized. More significantly, those components can also work easier with higher bus width systems, e.g. 512 bits, by adjusting the DMA configuration.

- **Processing throughput**: BIC, BIQP, and BIE are fully operational at 100 MHz. At this frequency, BIC indexes 16-bit data in parallel at the throughput of 1.48 GB/s, BIQP processes 32-Kbit BI vectors in parallel at the throughput of 2.45 GB/s, and BIE encodes 2,048-bit data in parallel at the maximum and minimum throughput of 66.34 Gb/s and 95.23 Mb/s, respectively.

- **Energy efficiency**: BIC, BIQP, and BIE requires less energy to process the same data unit, compared to CPU-based and GPU-based designs. For example, BIC32K16 requires 15.3× and 43.1 × lower energy than IMS and GPU13 to index 1-GB data, respectively. BIQP consumes 223× and 31× less energy than CPU-based IMS and GPU-based GPU11, respectively, to process 1-GB BI vectors.

The simulation and measurement results of the combination of BIQP and BIE in the 180-nm bulk CMOS and 65-nm SOTB CMOS process have proven their capability for ASIC implementation. The 180-nm chip was fully operational at 40 MHz and consumed 26.14 mW with a supply voltage of 1.8 V. The 65-nm chip, containing the double-sized-BIM, was fully operational at 45 MHz and consumed only 8.29 mW with a supply voltage of 1.2 V. More significantly, the performance of the 65-nm chip can be improved by applying appropriate bias voltages [25].

7.1.2 Limitations

In this dissertation, the proposed hardware systems still contain several limitations as shown below.

- In an Arria V FPGA, the embedded memory blocks are used to construct RAM-based CAM and BI memory, which are the primary modules of BIC

and BIQP, respectively. As much as 20-Mbit memory is employed in BIC and BIQP. Although those memory blocks are simply used in an FPGA, their implementation in an ASIC is very challenging. This is because those blocks are built from the flip-flop-based registers, instead of the memory cells themselves. A one-bit register requires much more transistors than a one-bit memory cell, thereby significantly increasing the circuit size as well as power consumption. As a consequence, only a small portion of the BI memory was verified in both the 180-nm bulk CMOS and the 65-nm SOTB CMOS process.

- Data from the FPGA development kit are transferred from a host computer through a 1-Gbps Ethernet, which is likely to become a bottleneck in a real data analytics system. Fortunately, current FPGAs support many other high-speed communication means, such as a 10/100-Gbps fiber optical transceiver, 32-Gbps PCI-Express 3.0, and 6-Gbps SATA. The utilization of those means can reduce the transfer time between a host computer and an FPGA kit. However, since this research mainly focuses on the efficient hardware architectures, the study on effective communication will be left in future work.

7.2 Future Work

Although the performance of each component outperforms other designs in multi-core CPUs and GPUs, the impact of an entire system when being integrated into a DBMS has still not been studied. For this reason, future work will focus on: (1) examining a suitable open DBMS that supports the OLAP workload and column-oriented model, (2) studying the highly efficient means of communication between the data analytics system and the host computer (i.e. employ PCI Express bus or 10/100-Gbps fiber optical connection), and (3) developing necessary software libraries and frameworks, so that open DBMS can fully exploit this system.

Furthermore, in this dissertation, BIC only generates raw BI vectors. In some cases, BI vectors must be stored in the external memory or hard disk for future use. To save the I/O cost, compression is almost inevitable. Therefore, future work will focus on: (1) reviewing all BI compression algorithms [73], such as BBC

(1995), WAH (2006), PLWAH (2010), CONCISE (2010), and SPLWAH (2015), (2) studying the feasibility of implementing those compression algorithms in hardware (a so-called custom instruction set framework, first introduced in 2016, is able to speed up the compression process in WAH, PLWAH, and COMPAX [74]), and (3) proposing a hardware-friendly algorithm that balances the compression ratio with resource utilization and operating frequency.

The capability of solving aggregation queries is a strong advantage of BI over other indexing methods. The iceberg query, which was originally introduced in 1988 by F. Min et al. [75], is a special type of aggregation query that computes aggregate values above a user-provided threshold. The number of above threshold results is often very small (the tip of an iceberg), relative to a large amount of input data (the iceberg). Nonetheless, the results often carry critical and valuable business insights. As a result, future research will focus on: (1) hardware-based frequent item counting and (2) the iceberg query with the help of BI and the frequent item counting circuit.

This page intentionally left blank.

Bibliography

[1] "Cisco Global Cloud Index: Forecast and Methodology 2015–2020." [Online]. Available: http://www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.pdf. [Accessed: 15-Feb-2017].

[2] H. Zhang, G. Chen, B. Ooi, K. Tan, and C. Ooi, "In-memory big data management and processing: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 7, pp. 1920–1948, 2016.

[3] X.-T. Nguyen, S.-C. Haw, S. Subramaniam, and C.-K. Pham, "Dynamic Node Labeling Schemes for XML Updates," in *Proceedings of the 6th International Conference On Computing & Informatics (ICOCI)*, pp. 505–510, 2017.

[4] J. Wang, W. Liu, S. Kumar, and S.-F. Chang, "Learning to Hash for Indexing Big Data–A Survey," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 34–57, 2016.

[5] H. Wong, H.-F. Liu, F. Olken, D. Rotem, and L. Wong, "Bit transposed files," in *Proceedings of the 11th International Conference on Very Large Data Bases*, pp. 448–457, 1985.

[6] P. O'Neil and D. Quass, "Improved query performance with variant indexes," *ACM SIGMOD Record*, vol. 26, no. 2, pp. 38–49, 1997.

[7] R. R. Sinha and M. Winslett, "Multi-resolution bitmap indexes for scientific data," *ACM Transactions on Database Systems*, vol. 32, no. 3, pp. 16–39, 2007.

[8] K. Wu, A. Shoshani, and K. Stockinger, "Analyses of multi-level and multi-component compressed bitmap indexes," *ACM Transactions on Database Systems*, vol. 35, no. 1, pp. 1–52, 2010.

[9] K. Stockinger and K. Wu, "Bitmap Indices for Data Warehouses," in *Data Warehouses and OLAP: Concepts, Architectures and Solutions*, pp. 157–178, 2007.

[10] K. Wu, E. J. Otoo, and A. Shoshani, "Optimizing bitmap indices with efficient compression," *ACM Transactions on Database Systems*, vol. 31, no. 1, pp. 1–38, 2006.

[11] "FastBit: An Efficient Compressed Bitmap Index Technology." [Online]. Available: https://sdm.lbl.gov/fastbit/. [Accessed: 30-June-2017].

[12] "The Worldwide LHC Computing Grid (WLCG)." [Online]. Available: http://wlcg.web.cern.ch/. [Accessed: 15-Mar-2017].

[13] "Edison." [Online]. Available: http://www.nersc.gov/users/computational-systems/edison/. [Accessed: 01-Mar-2017].

[14] "Nvidia Geforce GTX 1080." [Online]. Available: https://www.nvidia.com/en-us/geforce/products/10series/geforce-gtx-1080/. [Accessed: 01-Mar-2017].

[15] "Intel Stratix V FPGAs." [Online]. Available: https://www.altera.com/products/fpga/stratix-series/stratix-v/overview.html. [Accessed: 05-Mar-2017].

[16] M. Rene, T. Jens, and A. Gustavo, "Data processing on FPGAs," *Proceedings of the VLDB Endowment*, vol. 2, iss. 1, pp. 910–912, 2009.

[17] A. Dollas. "Big Data Processing with FPGA Supercomputers: Opportunities and Challenges," in *Proceedings of IEEE Computer Society Annual Symposium on VLSI*, pp. 474–479, 2014.

[18] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman,

S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao, and D. Burger, "A reconfigurable fabric for accelerating large-scale datacenter services," in *Proceedings of ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pp. 13–24, 2014.

[19] B. Sukhwani, M. Thoennes, H. Min, P. Dube, B. Brezzo, S. Asaad, and D. Dillenberger, "A Hardware/Software Approach for Database Query Acceleration with FPGAs," *International Journal of Parallel Programming*, vol. 43, no. 6, pp. 1129–1159, 2015.

[20] Oracle, "Software in Silicon," 2016. [Online]. Available: https://community.oracle.com/docs/DOC-932216. [Accessed: 01-Feb-2017].

[21] I. Kuon and J. Rose, "Measuring the Gap Between FPGAs and ASICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, 2007.

[22] D. Singh and C. K. Reddy, "A survey on platforms for big data analytics," *Journal of Big Data*, vol. 2, no. 1, pp. 1–20, 2014.

[23] O. Jian, L. Shiding, Q. Wei, W. Yong, Y. Bo, and J. Song, "SDA: Software-defined accelerator for large-scale DNN systems," in *Proceedings of IEEE Hot Chips 26 Symposium (HCS)*, pp. 1–23, 2014.

[24] "With SDAccel, Xilinx Embraces OpenCL." [Online]. Available: https://www.bdti.com/InsideDSP/2015/01/22/Xilinx. [Accessed: 30-June-2017].

[25] R. Tsuchiya, M. Horiuchi, S. Kimura, M. Yamaoka, T. Kawahara, S. Maegawa, T. Ipposhi, Y. Ohji, and H. Matsuoka, "Silicon on thin BOX: a new paradigm of the CMOSFET for low-power and high-performance application featuring wide-range back-bias control," in *Proceedings of IEEE International Electron Devices Meeting (IEDM) Technical Digest*, pp. 631–634, 2014.

[26] T. Ishigaki, R. Tsuchiya, Y. Morita, N. Sugii, and S. Kimura, "Ultralow-power LSI Technology with Silicon on Thin Buried Oxide (SOTB) CMOS-FET," in *Solid State Circuits Technologies, Chapter 7*, pp. 146–156, 2010.

[27] D.-H. Le, N. Sugii, S. Kamohara, X.-T. Nguyen, K. Ishibashi, and C.-K. Pham, "Design of a Low-power Fixed-point 16-bit Digital Signal Processor Using 65nm SOTB Process," in *The International Conference on Integrated Circuit Design and Technology (ICICDT)*, pp. 1–4, 2015.

[28] K. Ishibashi, N. Sugii, S. Kamohara, K. Usami, H. Amano, K. Kobayashi, and C.-K. Pham, "A Perpetuum Mobile 32bit CPU on 65nm SOTB CMOS Technology with Reverse-Body-Bias Assisted Sleep Mode," *IEICE Transactions on Electronics*, vol. E98–C, no. 7, pp. 536–543, 2015.

[29] Z. Li, K. C. K. Lee, B. Zheng, W.-C. Lee, D. Lee, and X. Wang, "IR-Tree: An Efficient Index for Geographic Document Search," *IEEE Transactions on Knowledge Data Engineering*, vol. 23, no. 4, pp. 585–599, 2011.

[30] H. Wang, X. Qin, X. Zhou, F. Li, Z. Qin, Q. Zhu, and S. Wang, "Efficient query processing framework for big data warehouse: an almost join-free approach," *Frontiers of Computer Science*, vol. 9, no. 2, pp. 224–236, 2015.

[31] Y. Hua, H. Jiang, and D. Feng, "Real-Time Semantic Search Using Approximate Methodology for Large-Scale Storage Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 4, pp. 1212–1225, 2016.

[32] J. Chou, R. D. Ryne, M. Howison, B. Austin, K. Wu, J. Qiang, E. W. Bethel, A. Shoshani, and O. Rübel, "Parallel index and query for large scale data analysis," in *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 1–11, 2011.

[33] T. Zhong, K. A. Doshi, and G. Deng, "ParaSAIL: Bitmap Indexing Using Many Cores," in *Proceedings of International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, pp. 233–240, 2014.

[34] Y. B. Liu, H. M. Dai, F. Wang, W. Dai, and H. Deng, "Efficient Indexing and Querying of Massive Astronomical Data Using Compressed Word-Aligned Hybrid Bitmap," in *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 1567–1572, 2015.

[35] C. Hsuan-Te, J. Chou, V. Vishwanath, and W. Kesheng, "In-memory Query System for Scientific Datasets," in *Proceedings of IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 362–371, 2015.

[36] J. Kim, W.-K. Jeong, and B. Nam, "Exploiting Massive Parallelism for Indexing Multi-Dimensional Datasets on the GPU," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 8, pp. 2258–2271, 2015.

[37] M. Alam, S. B. Yoginath, and K. S. Perumalla, "Performance of Point and Range Queries for In-memory Databases Using Radix Trees on GPUs," in *Proceedings of IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pp. 1493–1500, 2016.

[38] H. Pirk, S. Manegold, and M. Kersten, "Waste not... Efficient co-processing of relational data," in *Proceedings of IEEE 30th International Conference on Data Engineering (ICDE)*, pp. 508–519, 2014.

[39] W. Andrzejewski and R. Wrembel, "GPU-PLWAH: GPU-based implementation of the PLWAH algorithm for compressing bitmaps," *Control and Cybernetics*, vol. 40, no. 3, pp. 627–650, 2011.

[40] F. Fusco, M. Vlachos, X. Dimitropoulos, and L. Deri, "Indexing million of packets per second using GPUs," in *Proceedings of the Conference on Internet Measurement Conference (IMC)*, pp. 327–332, 2013.

[41] D. Heinrich, S. Werner, M. Stelzner, C. Blochwitz, T. Pionteck, and S. Groppe, "Hybrid FPGA approach for a B+ tree in a Semantic Web database system," in *Proceedings of 10th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, pp. 1–8, 2015.

[42] K. Agarwal and R. Polig, "A high-speed and large-scale dictionary matching engine for information extraction systems," in *IEEE 24th International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, pp. 59–66, 2013.

[43] D.-H. Le, K. Inoue, M. Sowa, and C.-K. Pham, "An FPGA-Based Information Detection Hardware System Employing Multi-Match Content Addressable Memory," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E95.A, no. 10, pp. 1708–1717, 2012.

[44] C.-H. Huang, J.-S. Wang, and Y.-C. Huang, "Design of high-performance CMOS priority encoders and incrementer/decrementers using multilevel lookahead and multilevel folding techniques," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 1, pp. 63–76, 2002.

[45] C. Kun, S. Quan, and A. Mason, "A power-optimized 64-bit priority encoder utilizing parallel priority look-ahead," in *Proceedings of IEEE International Symposium on Circuits Systems*, vol. 2, pp. II-–753-–6, 2004.

[46] M. Faezipour and M. Nourani, "Wire-Speed TCAM-Based Architectures for Multimatch Packet Classification," *IEEE Transactions on Computers*, vol. 58, no. 1, pp. 5–17, 2009.

[47] D. Balobas and N. Konofaos, "Low-power, high-performance 64-bit CMOS priority encoder using static-dynamic parallel architecture," in *Proceedings of IEEE International Conference on Modern Circuits and Systems Technologies (MOCAST)*, pp. 1–4, 2016.

[48] S. Abdel-Hafeez and S. Harb, "A VLSI High-Performance Priority Encoder Using Standard CMOS Library," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 53, no. 8, pp. 597–601, 2006.

[49] D.-H. Le, T.-B.-T. Cao, K. Inoue, C.-K. Pham, "A CAM-based Information Detection Hardware System for Fast Image Matching on FPGA," *IEICE Transactions on Electronics*, vol. E97-C, no. 1, pp. 65–76, 2014.

[50] S. K. Maurya and L. T. Clark, "A Dynamic Longest Prefix Matching Content Addressable Memory for IP Routing," *IEEE Transactions on Very Large Scale Integrated (VLSI) Systems*, vol. 19, no. 6, pp. 963–972, 2011.

[51] X.-T. Nguyen, H.-T. Nguyen, and C.-K. Pham, "An FPGA approach for high-performance multi-match priority encoder," *IEICE Electronics Express*, vol. 13, no. 13, pp. 1–9, July 2016.

[52] X.-T. Nguyen, H.-T. Nguyen, and C.-K. Pham, "An FPGA approach for fast bitmap indexing," *IEICE Electronics Express*, vol. 13, no. 4, pp. 1–9, 2016.

[53] X.-T. Nguyen, H.-T. Nguyen, and C.-K. Pham, "A High-Throughput and Low-Power Design for Bitmap Indexing on 65-nm SOTB CMOS Process," in *Proceedings of The International Conference on IC Design and Technology (ICICDT)*, pp. 1–4, 2016.

[54] X.-T. Nguyen, H.-T. Nguyen, and C.-K. Pham, "A Bit-Level Matrix Transpose for Bitmap-Index-Based Data Analytics," in *Proceedings of The 6th International Conference on Communications and Electronics (ICCE)*, pp. 217–220, 2016.

[55] X.-T. Nguyen, H.-T. Nguyen, and C.-K. Pham, K. Inoue, O. Shimojo, and C.-K. Pham, "Highly Parallel Bitmap-Index-Based Regular Expression Matching For Text Analytics," in *Proceedings of The IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2667–2670, 2017.

[56] R. Karam, R. Puri, S. Ghosh, and S. Bhunia, "Emerging Trends in Design and Applications of Memory-Based Computing and Content-Addressable Memories," *Proceedings of the IEEE*, vol. 103, no. 8, pp. 1311–1330, 2015.

[57] "Implementing High-Speed Fundamentals CAM in Altera," in *Altera Application Note 119*, pp. 1–50, 2001.

[58] L. Kyle. "Parameterizable Content-Addressable Memory," in *Xilinx Application Note 1151*, pp. 1–31, 2011.

[59] X.-T. Nguyen, H.-T. Nguyen, T.-T. Hoang, K. Inoue, O. Shimojo, T. Murayama, K. Tominaga, and C.-K. Pham, "An Efficient FPGA-Based Database Processor for Fast Database Analytics," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1758–1761, 2016.

[60] X.-T. Nguyen, H.-T. Nguyen, and C.-K. Pham, "A High-Performance Bitmap-Index-Based Query Processor on 65-nm SOTB CMOS Process," in *Proceedings of The IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*, pp. 1–2, 2016.

[61] X.-T. Nguyen, H.-T. Nguyen, and C.-K. Pham, "A 180-nm CMOS Bitmap-Index-Based Query Processor for Fast Data Analytics," in *Proceedings of The International Conference on Recent Advances on Signal Processing, Telecommunications & Computing (SigTelCom)*, pp. 155–157, 2017. (*invited paper*)

[62] S.-W. Huang and Y.-J. Chang, "A full parallel priority encoder design used in comparator," in *Proceedings of IEEE International Midwest Symposium on Circuits Systems*, pp. 877–880, 2010.

[63] X.-T. Nguyen, H.-T. Nguyen, and C.-K. Pham, "A High-Throughput Multi-Match Priority Encoder for Data Retrieval on 65-nm SOTB CMOS Process," in *Proceedings of The IEEE Region 10 Conference (TENCON)*, pp. 2392–2395, Singapore, November 22–25, 2016.

[64] X.-T. Nguyen, H.-T. Nguyen, and C.-K. Pham, "A Scalable High-Performance Priority Encoder Using 1D-array to 2D-array Conversion," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2017. (*in press*)

[65] "Arria V SoC Development Kit." [Online]. Available: https://www.altera.com/products/boards_and_kits/dev-kits/altera/kit-arria-v-soc.html. [Accessed: 18-April-2017].

[66] "Arria V Hard Processor System Technical Reference Manual." [Online]. Available: https://www.altera.com/literature/hb/arria-v/av_5v4.pdf. [Accessed: 20-April-2017].

[67] Altera, "Optimizing the Controller," 2013.

[68] T. Hussain, O. Palomar, O. Unsal, A. Cristal, E. Ayguade, and M. Valero, "Advanced Pattern based Memory Controller for FPGA based HPC applications," in *Proceedings of International Conference on High Performance Computing & Simulation (HPCS)*, pp. 287–294, 2014.

[69] X.-T. Nguyen and C.-K. Pham, "An Efficient Multi-port Memory Controller for Multimedia Applications," in *Proceedings of the 20th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 12–13, 2015.

[70] X.-T. Nguyen, H.-T. Nguyen, and C.-K. Pham, "Parallel Pipelining Configurable Multi-port Memory Controller For Multimedia Applications," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2908–2911, 2015.

[71] IBM, "White paper: The IBM Netezza data warehouse appliance architecture," 2011.

[72] "Project Catapult." [Online]. Available: https://www.microsoft.com/en-us/research/project/project-catapult/. [Accessed: 16-Mar-2017].

[73] Z. Chen, Y. Wen, J. Cao, W. Zheng, J. Chang, Y. Wu, G. Ma, M. Hakmaoui, and G. Peng, "A survey of bitmap index compression algorithms for big data," *Tsinghua Science and Technology*, vol. 20, no. 1, pp. 100–115, 2015.

[74] S. Haas, T. Karnagel, O. Arnold, E. Laux, B. Schlegel, G. Fettweis, and W. Lehner, "HW/SW-database-codesign for compressed bitmap index processing," in *Proceedings of IEEE 27th Int. Conf. Application-specific Systems, Architectures and Processors (ASAP)*, pp. 50–57, 2016.

[75] F. Min, S. Narayanan, G.-M. Hector, M. Rajeev, and D. U. Jeffrey, "Computing Iceberg Queries Efficiently," in *Proceedings of the 24rd International Conference on Very Large Data Bases*, pp. 299–310, 1998.

Appendix A

Full-Chip Layout Micrograph

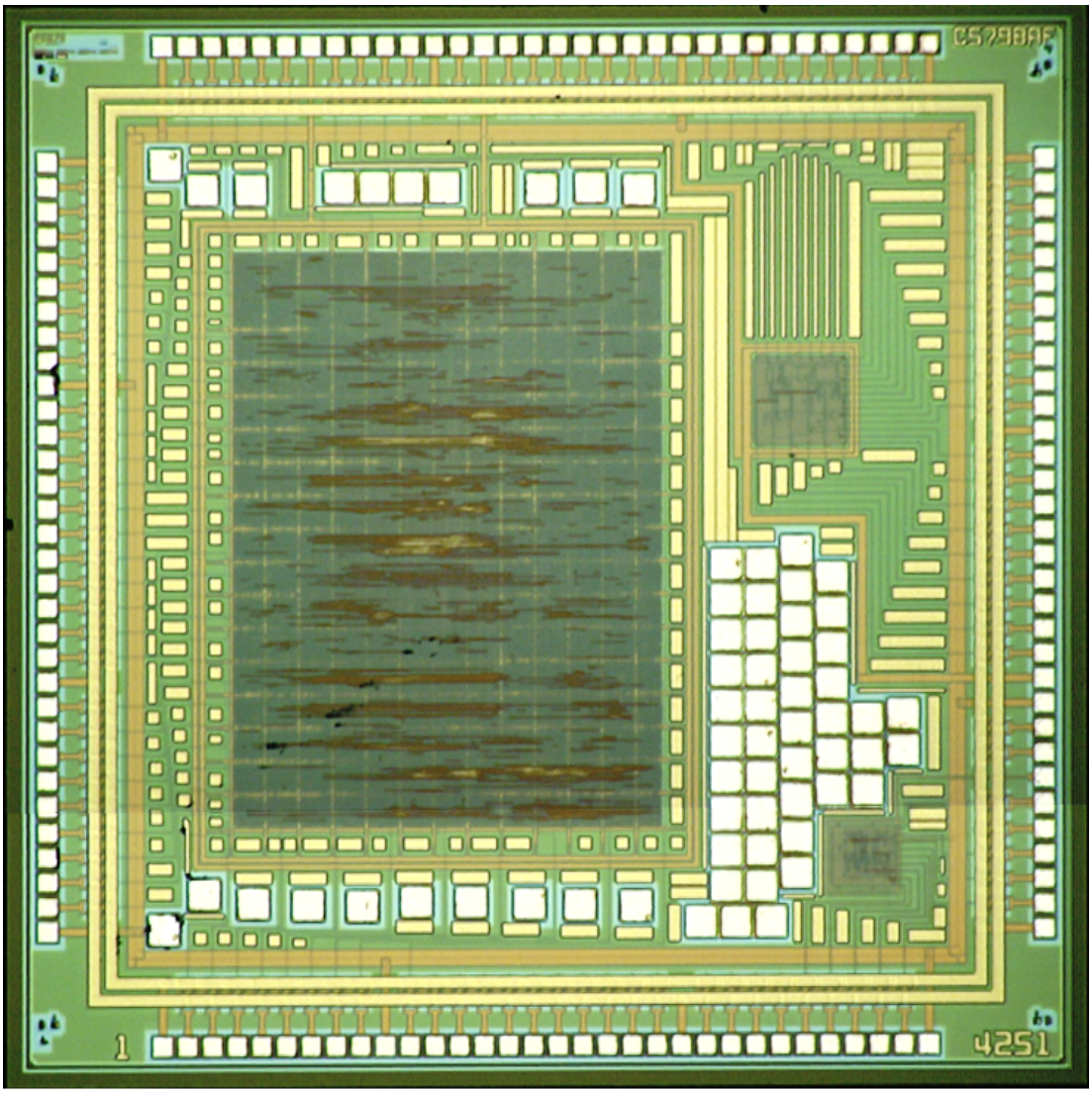


Figure A.1: The layout of the combination of BIQP and BIE in 180-nm bulk CMOS technology.

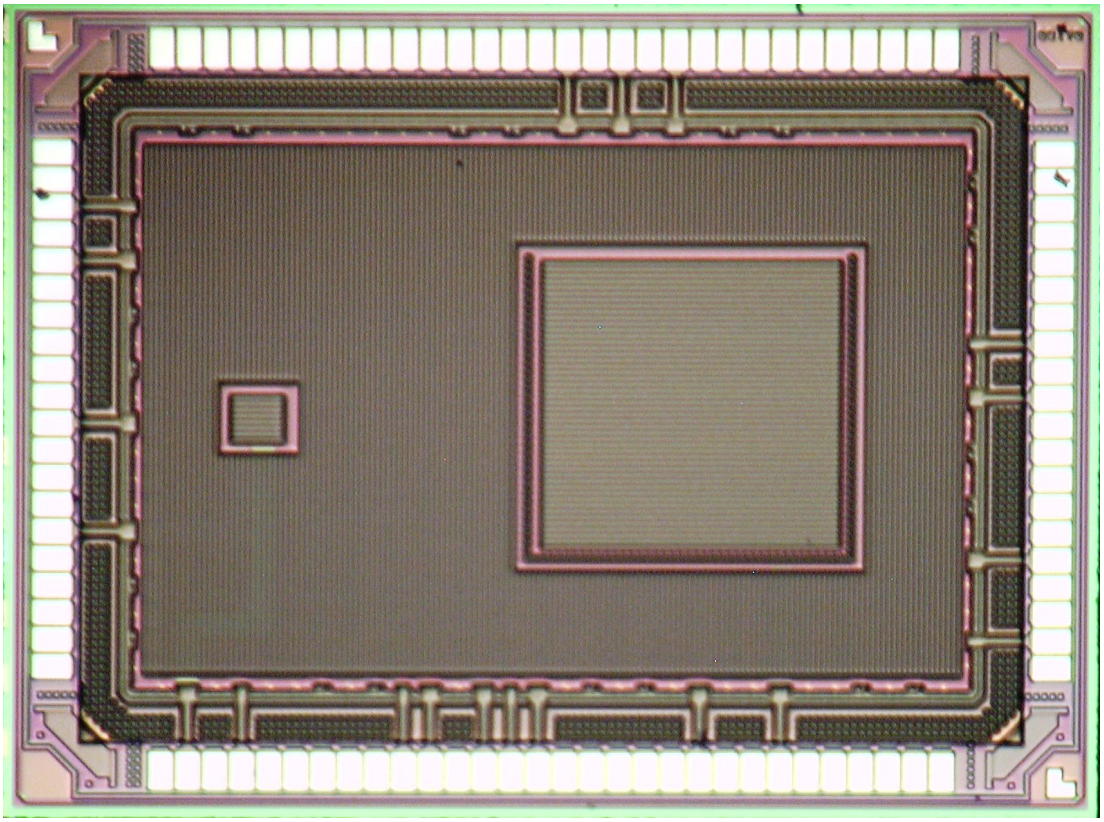


Figure A.2: The layout of the combination of BIQP and BIE in 65-nm SOTB CMOS technology.

This page intentionally left blank.

Appendix B

List of Publications

B.1 Journal Papers

[1] Xuan-Thuan Nguyen, Hong-Thu Nguyen, and Cong-Kha Pham, "An FPGA approach for fast bitmap indexing," *IEICE Electronics Express*, vol. 13, no. 4, pp. 1–9, February 2016.

[2] Xuan-Thuan Nguyen, Hong-Thu Nguyen, and Cong-Kha Pham, "An FPGA approach for high-performance multi-match priority encoder," *IEICE Electronics Express*, vol. 13, no. 13, pp. 1–9, July 2016.

[3] Xuan-Thuan Nguyen, Hong-Thu Nguyen, and Cong-Kha Pham, "A Scalable High-Performance Priority Encoder Using 1D-array to 2D-array Conversion," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2017. (*in press*)

B.2 International Conference Presentations

[1] Xuan-Thuan Nguyen and Cong-Kha Pham, "An Efficient Multi-port Memory Controller for Multimedia Applications," *The 20th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 12–13, Chiba, Japan, January 19–22, 2015.

[2] Xuan-Thuan Nguyen, Hong-Thu Nguyen, and Cong-Kha Pham, "Parallel Pipelining Configurable Multi-port Memory Controller For Multimedia Ap-

plications," *The IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2908–2911, Lisbon, Portugal, May 24–27, 2015.

[3] Duc-Hung Le, Nobuyuki Sugii, Shiro Kamohara, Xuan-Thuan Nguyen, Koichiro Ishibashi, and Cong-Kha Pham, "Design of a Low-power Fixed-point 16-bit Digital Signal Processor Using 65nm SOTB Process," *The International Conference on Integrated Circuit Design and Technology (ICICDT)*, pp. 1–4, Leuven, Belgium, June 1–3, 2015.

[4] Xuan-Thuan Nguyen, Hong-Thu Nguyen, and Cong-Kha Pham, "A Reliable Protocol For Multimedia Transmission Over Wireless Sensor Networks," *The 11th Conference on PhD Research in Microelectronics and Electronics (PRIME)*, pp. 302–305, Glasgow, Scotland, June 29–July 2, 2015.

[5] Xuan-Thuan Nguyen, Hong-Thu Nguyen, and Cong-Kha Pham, "SAR: A Self-Adaptive and Reliable Protocol for Wireless Multimedia Sensor Networks," *The 7th International Conference on Ubiquitous and Future Networks (ICUFN)*, pp. 760–765, Sapporo, Japan, July 7–10, 2015.

[6] Xuan-Thuan Nguyen, Hong-Thu Nguyen, Trong-Thuc Hoang, Katsumi Inoue, Osamu Shimojo, Toshio Murayama, Kenji Tominaga, and Cong-Kha Pham, "An Efficient FPGA-Based Database Processor for Fast Database Analytics," *The IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1758–1761, Montreal, Canada, May 22–25, 2016.

[7] Xuan-Thuan Nguyen, Hong-Thu Nguyen, and Cong-Kha Pham, "A High-Throughput and Low-Power Design for Bitmap Indexing on 65-nm SOTB CMOS Process," *The International Conference on IC Design and Technology (ICICDT)*, pp. 1–4, Ho Chi Minh, Vietnam, June 27–29, 2016.

[8] Xuan-Thuan Nguyen, Hong-Thu Nguyen, and Cong-Kha Pham, "A Bit-Level Matrix Transpose for Bitmap-Index-Based Data Analytics," *The 6th International Conference on Communications and Electronics (ICCE)*, pp. 217–220, Ha Long, Vietnam, July 27–29, 2016.

[9] Xuan-Thuan Nguyen, Hong-Thu Nguyen, and Cong-Kha Pham, "A High-Performance Bitmap-Index-Based Query Processor on 65-nm SOTB CMOS

Process," *The IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*, pp. 1–2, San Francisco, USA, Oct 10–13, 2016.

[10] Xuan-Thuan Nguyen, Hong-Thu Nguyen, and Cong-Kha Pham, "A High-Throughput Multi-Match Priority Encoder for Data Retrieval on 65-nm SOTB CMOS Process," *The IEEE Region 10 Conference (TENCON)*, pp. 2392–2395, Singapore, November 22–25, 2016.

[11] Xuan-Thuan Nguyen, Hong-Thu Nguyen, and Cong-Kha Pham, "A 180-nm CMOS Bitmap-Index-Based Query Processor for Fast Data Analytics," *The International Conference on Recent Advances on Signal Processing, Telecommunications & Computing (SigTelCom)*, pp. 155–157, Da Nang, Vietnam, January 9–11, 2017. (*invited paper*)

[12] Xuan-Thuan Nguyen, Su-Cheng Haw, Samini Subramaniam, and Cong-Kha Pham, "Dynamic Node Labeling Schemes for XML Updates," *The 6th International Conference On Computing & Informatics (ICOCI)*, pp. 505–510, Selangor Darul Ehsan, Malaysia, April 25–27, 2017.

[13] Xuan-Thuan Nguyen, Hong-Thu Nguyen, Katsumi Inoue, Osamu Shimojo, and Cong-Kha Pham, "Highly Parallel Bitmap-Index-Based Regular Expression Matching For Text Analytics," *The IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2667–2670, Baltimore, USA, May 28–31, 2017.

This page intentionally left blank.

Author Biography

Xuan-Thuan NGUYEN received the B.Sc. degree in Electronics and Telecommunications (first class honor) in 2010, and M.Sc. degree in Microelectronics and VLSI design in 2013, both from the University of Science, Ho Chi Minh City (HCMUS), Vietnam. From 2010 to 2013, he was a research and teaching assistant at the Faculty of Electronics and Telecommunications, HCMUS. Since 2013, he has been on the Department of Engineering Science, the University of Electro-Communications, Tokyo, where he is currently a PhD student.

His research interests include hardware acceleration in data analytics and embedded systems. He is a member of IEEE and IEICE.

THE END.