DIGITAL ARCHITECTURE OF SORTING ALGORITHMS BASED ON FPGA

A DISSERTATION

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE AWARD OF THE DEGREE
OF
MASTER OF TECHNOLOGY (2018-2020)
IN
VLSI DESIGN & EMBEDDED SYSTEM

Submitted by:

GAURI SETHI

2K18/VLS/04

Under the supervision of

Ms. Kriti Suneja

Department of Electronics & Communication Engineering
Delhi Technological University
(Formerly Delhi College of Engineering)
Bawana Road , Delhi -110042

AUGUST 2020

Department of Electronics & Communication Engineering
Delhi Technological University
(Formerly Delhi College of Engineering)
Bawana Road , Delhi-110042

CANDIDATE'S DECLARATION

I, GAURI SETHI, Roll no 2K18/VLS/04, a student of Mtech VLSI DESIGN & EMBEDDED SYSTEM, hereby declare that the project Dissertation titled "Digital Architecture of Sorting Algorithms based on FPGA" which is submitted by me to the Department of Electronics and Communication Engineering, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the degree of Master of Technology, is original and not copied from any source without proper citation. The work has not previously formed the basis for the award of any Degree, Diploma Associateship, Fellowship or other similar title or recognition.

Place: Delhi

Date: August, 2020

GAURI SETHI
(2K18/VLS/04)

Department of Electronics & Communication Engineering
Delhi Technological University
(Formerly Delhi College of Engineering)
Bawana Road , Delhi-110042

CERTIFICATE

I hereby certify that the Project Dissertation titled "DIGITAL ARCHITECTURE OF SORTING ALGORITHMS BASED ON FPGA" which is submitted by GAURI SETHI, 2K18/VLS/04 of VLSI DESIGN , Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the degree of Master of Technology is a bonafide record of the project work carried out by the student under my supervision . To the best of my knowledge the work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place: Delhi **Ms. KRITI SUNEJA**

Date: August, 2020 **SUPERVISOR**

ABSTRACT

Due to the burgeoning demand of programmable logic density having fast speed, high density and based on hardware description language (HDL), the engineers are being empowered to implement within Field Programmable Gate Array (FPGA) high performance digital functionality and various complex circuits. Sorting algorithms have marked an epoch in the life of computer engineers and the advancements related to these will only help in adding tranquility to their lives while they are maneuvering large amounts of data at a time.

This research encompasses the sorting algorithms covering all of their facets from the history of the algorithms up to their implementation in the software and also giving the details about the research that has been carried out comrade in the hardware domain. I have implemented 6 sorting algorithms in Verilog language i.e. Bubble sort, Merge sort, Insertion sort, Selection sort, Radix sort and Count sort. Bubble sort has the easiest hardware implementation as evident in the analysis carried whereas count sort is limited by the largest number present in the array. All of these have been compared on the basis of the three most important metrics which are always considered in the design and implementation in VLSI field i.e. area, timing and power.

Target device used for obtaining synthesis results is ZYNQ – 7000 FPGA, which is increasingly becoming popular among the FPGA engineers due to its advanced features that make it stand out among all boards in the presence of an ARM cortex A9 chip which is the main reason for its usage as a system on chip (SOC), having an integrated support for PCI Express also helps it to

persuade its dominance over other FPGAs known to us.

For simulations and synthesis, VIVADO 2019.1 has been used. The output waveforms of all the six sorting algorithms have been plotted. They have further been analyzed in terms of hardware utilization (number of slices, which is comprised of Look Up Tables or LUTs and flip flops) , timing (delay in ns) and power consumption (in mW).

ACKNOWLEDGEMENT

A wise man once said “A journey of a thousand miles begins with a single step”. So I would like to express my special thanks of gratitude to all the individuals who have been patient with me whenever I have fumbled at any step. Also this is a long journey that has took over an year to reach its beautiful destination and I could only gather the courage to follow this road map because of the support anchored by few people in this journey at all times.

First of all I feel indebted to the Almighty God for giving me the strength and the knowledge to have been able to explore this topic and for always enlightening me with the correct path in life. Next to them are my parents without whose support I would not be able to accomplish this task and saying “Thanks” is a very small word for that. I feel obliged in availing this occasion to have been able to express my deep sense of appreciation to **Prof. N.S. Raghava**, HOD of the Electronics and Communication Engineering Department, Delhi Technological University for bestowing me with the opportunity to work on this project, motivating me and guiding me through the numerous thick and thin I encountered during my research.

I would also like to express my deepest gratitude to my mentor, philosopher and supervisor **Ms. Kriti Suneja** whose valuable guidance has armed me with the results I have obtained and has acted as the golden lamp I was empowered with in this path of darkness. Her dedication, keen interest, and her timely suggestions filled with kindness which have always proven to be quite beneficial for this project have helped me in achieving the desired results and in turn helped me in realizing my potential. She has always supported me and respected my ideologies with full enthusiasm and zest that have enabled me to complete this project. I am grateful to have been able to complete my Master of Technology thesis work under her supervision and to learn a lot of new things for enhancing my knowledge and skills within this limited time frame.

I also feel indebted to the constant support catered to me by the entire staff of the VLSI Laboratory, Delhi Technological University. I also consider this as the suitable occasion to appreciate the co-operation showed by my friends and colleagues to all my queries.

GAURI SETHI
VLSI DESIGN & EMBEDDED SYSTEM
4th Semester
Delhi Technological University
(Formerly Delhi College of Engineering)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

1.	**MIO**	- Multiplexed Input Output
2.	**FPGA**	- Field-Programmable Gate Array
3.	**LUT**	- Look-Up-Table
4.	**IOB**	- Input-Output-Block
5.	**DCM**	- Digital Clock Manager
6.	**IC**	- Integrated Circuit
7.	**SOC**	- System – On – Chip
8.	**DAC**	- Divide and Conquer
9.	**CPU**	- Central Processing Unit
10.	**GPU**	- Graphics Processing Unit
11.	**Gbps**	- Giga bits per second
12.	**PS**	- Processing System
13.	**PL**	- Programmable Logic
14.	**ASIC**	- Application Specific Integrated Circuit
15.	**IOP**	- Input Output Peripherals
16.	**BW**	- Bandwidth
17.	**Swre**	- Software
18.	**APU**	- Application Processing System
19.	**ADC**	- Analog to Digital Converter

20. AXI	-	Advances eXtensible Interface
21. HSTL	-	High speed transreceiver logic
22. ns	-	Nanosecond
23. FF	-	Flip Flop
24. MW	-	Milli Watt
25. GPIOB	**-**	General purpose Input output buffer
26. CAN	**-**	Controller Area Network
27. DMA	-	Direct Memory Access
28. RAM	**-**	Random Access Memory
29. ROM	-	Read – only – Memory
30. KB	-	Kilo byte
31. GHz	-	Giga Hertz
32. ARM	-	Advanced RISC Machines
33. PCIe	-	Peripheral Component Interconnect Express
34. FIFO	-	First-in-First Out
35. SPI	-	Serial Peripheral Interconnect
36. VFPU	-	Vector Floating Point Unit
37. MAC	-	Media access control address
38. USB	-	Universal Serial Bus
39. OTG	-	USB -on –the- go
40. UART	-	Universal asynchronous receiver transmitter

41.	**PHY**	- Physical Layer
42.	**JTAG**	- Joint test action group
43.	**PLL**	- Phased Locked Loop
44.	**PCAP**	- Processor configuration access port
45.	**DDR**	- Double data rate
46.	**SDRAM**	- Synchronous Dynamic Random Access Memory
47.	**CMT**	- Clock Management Tiles
48.	**MCMM**	- Mixed-Mode Clock Manager
49.	**AES**	- Advanced Encryption standard
50.	**CLB**	- Configurable Logic Block

CHAPTER 1

INTRODUCTION

In recent times, the field of digital systems has undergone a drastic change with respect to its automization and functionality. Circuits are advancing towards nano - metre design processes, thereby reducing their physical size while growing in speed. Hence, large no of devices i.e. millions of transistors are now implemented on the IC today. As we are aware of the fact that the economic growth of any country is driven by the amount of oil reserves of that country , similarly today’s digital economy is affected by the amount of data present. Hence, Mathematician Clive Humby in 2006 coined the famous quote “Data is the new oil” , which though recently has been highlighted after an Economist published in 2017 a report titled “ The world’s most valuable resource is no longer oil , but data”. There are many companies which are banking on this concept and the much talked about pioneering algorithms of Machine Learning is also based on this manipulation of data wherein we can decode the hidden pattern in behavior by analyzing large volumes of data [1].

Now for efficient arrangement of this data so that it can be accessed easily & result in fast computations we are familiar with have certain sorting algorithms that are frequently used. These algorithms help us in sorting our data systematically and they can be classified according to the mechanisms these work on, which thereby have their merits & demerits accordingly depending on the applications they are being used in. The sorting mechanisms usually determines the speed & performance of the system, hence designing an algorithm that can process data fast and efficiently is the need of the hour. When working with data objected for research purpose, sorting is an often used technique that makes it easier to comprehend the story the data is telling when we have used it for visualizing data in a form we are familiar with. Without doubt, sorting is one of the most fundamental algorithmic problem that was noticed from the pre medieval period of computing. In fact, most of the computer science research was centered on finding the best way to sort a set of data [2]. Having knowledge about sorting algorithms is like learning keys of a musician, quite fascinating and challenging at the same time.

Performance of any sorting algorithm depends on various factors like no. of inputs, format of data, value of inputs, nature of machine etc. Sorting has lots of applications like database search, management, research operations, signal processing, scientific computing, robotics & artificial intelligence among others. The sorting algorithms when implemented on software on a large number of data inputs leads to an increase in their execution time and complexity. As a result various implementations have been carried out to improve their performance by tactfully using the advantage of the parallelism of multicore processors [3]. In recent years, designers have scrutinized the objective of designing hardware accelerators using field programmable gate arrays (FPGAs) [4]–[8]. They have a lot of distinctive features like parallel processing, high bandwidth, ease of programming, low latency among others.

1.2 HISTORY & CLASSIFICATION OF SORTING ALGORITHMS

The simplest algorithms of all was analyzed as early as in 1956. There exists an underlying principle of Ω (n log n) comparison around which the nucleus of comparison based algorithms revolves however the latter ones like counting sort can perform much better [9].

We can classify sorting algorithms as specified below:

- Computational complexity (worst, average and best behavior): This can be carried out in terms of the size of the list (n). Good behavior of the Algorithms is $O(n \log n)$ for typical serial sorting, while bad behavior is $O(n^2)$ and with parallel sort in $O(\log^2 n)$, Ideal behavior for a serial sort is $O(n)$, but in the average case it is neither feasible nor possible . O (log n) can be conveniently specified in the optimal parallel sorting cases. At least $\Omega(n \log n)$ comparisons for majority no. of inputs are required for comparison-based sorting algorithms require [10].
- We also have to calculate computational complexity of swaps (for "in-place" algorithms).
- Memory usage : In general a few sorting algorithms are "in-place", that strictly needs only O(1) memory beyond the items being sorted; sometimes additional memory of O(log(n)) is considered "in-place".
- Recursion: These algorithms can be either recursive or non-recursive, while some may be both (e.g., merge sort). There are countable algorithms that have properties of being both recursive & non-recursive.
- Stability: Those algorithm which are able to sustain the relative order of records with equal keys (i.e., values) are termed as stable sorting algorithms
- Comparison sort: It examines the data with a comparison operator by

simply comparing two elements and then analyzing the result obtained.

- Serial or parallel mode of operation : Often we come across serial algorithms only and seldom we face any parallel algorithms in our day-to-day computer data processing applications.
- Adaptability: Whether the running time of the computations are affected by the pre-sortedness of the input or not is a deciding factor that is employed while deciding the adaptability of any algorithm.

1.2.1 Stability

The repeated elements are sorted in the same order in which they appear in the input in case of stable sort algorithms. Only some part of the data is examined when we have the task of determining the sort order when we are given the task of sorting some kind of data. For example, this can be depicted easily by considering a simple card sorting i.e. while the suits can be ignored, the cards need to be arranged by their rank. This increases the probability of multi-varied accurately sorted variety of the same data in the original list. The stable sorting algorithms works according to the following procedure described here i.e it chooses one of these : if two items on comparison have equal value, like the two 5 cards as shown in fig 1.1 then we will preserve their relative order, so that if one card appeared before the other in the input, then the same pattern will be observed in the output also [11] .

The data values on which sorting operation is being performed can be depicted as a record or sequence of values, but the fragment of the data that is exclusively being called for sorting is referred to as the key. In the card example, the key is the rank while the cards are denoted as a record (rank, suit).

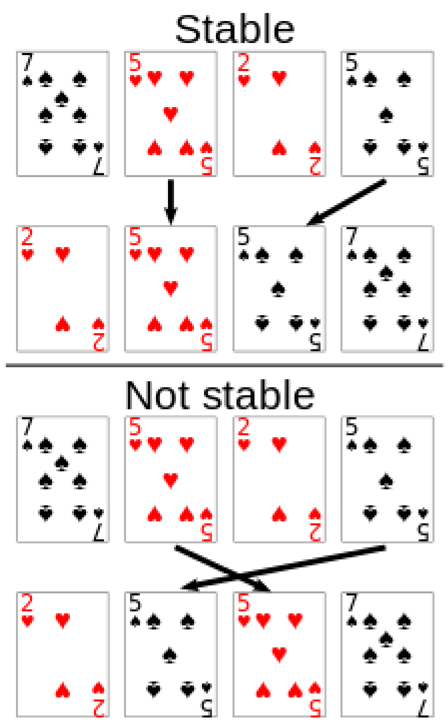
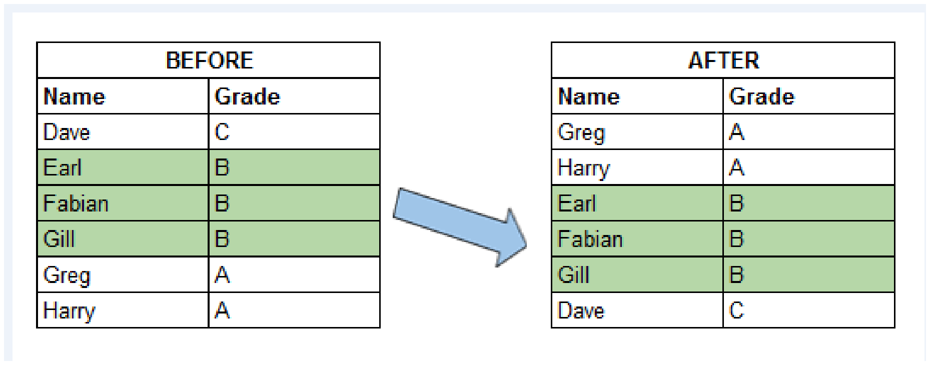


Fig 1.1 Playing cards example [11]

When two elements that have equal value are indistinguishable, such as for e.g. integers, or more briefly, stability is not questionable when the entire element is the key in any data. Stability cannot become a major concern when all keys differ from one another too [12].The sorting algorithms that are stable include: Tim Sort , Merge Sort , Bubble Sort , Count Sort , Quad Sort , Library Sort, Odd – even Sort while Heap Sort , Selection Sort, Shell Sort , Quick Sort , Tree Sort , Cycle Sort is the list of the unstable sorting algorithms.

Unstable sorting algorithms can also be made stable by specially implementing them. One method of achieving this is to artificially outstretch the key comparison, by making use of the original order of the entries in the input list as a tie-breaker to decide the differentiation between two objects with otherwise equal keys [9]. We may require additional time and space by remembering this order.

One application for stable sorting algorithm is shown in fig 1.2 wherein the grades obtained by the student in a class exam have been displayed randomly [13]. When the teacher uses stable sorting algorithm on the result, the grades are arranged according to a rule that is governed below. The order in which students Earl , Fabian are Gill are arranged in the input list is retained after arranged data is observed as output since it is a stable sorting algorithm.



BEFORE	
Name	**Grade**
Dave	C
Earl	B
Fabian	B
Gill	B
Greg	A
Harry	A

AFTER	
Name	**Grade**
Greg	A
Harry	A
Earl	B
Fabian	B
Gill	B
Dave	C

Fig 1.2 Stable Algorithm on grades [13]

1.3 TYPES OF SORTING PROCESS

There are two sorting procedures available while implementing algorithms i.e. internal & external.

1.3.1 Internal Sorting

It can be described as any sorting procedure which works when the data to be sorted is in small amount only as the entire data is stored in the main memory of the computer [14]. But when we have to sort large volumes of data, it may be necessary to hold on a block of data in memory at a time, as it won't be possible to accommodate all at once. We require smaller storage media like hard disk for keeping rest of the data. But as a consequence of the repeated access for any operation i.e. reading or writing data, it becomes time-consuming. Some common sorting algorithms are :

1. Insertion Sort
2. Bubble Sort
3. Selection Sort
4. Heap Sort
5. Radix Sort
6. Quick Sort

No access of external memory is required for execution of sorting program. When the input size is small, it is then only used.

1.3.2 External Sorting

In this case the data that is to be arranged is present in large volumes and hence stored in memory present outside the system memory like disks [9]. While sorting, it will be pulled out in large chunks from memory and hence we get the collected data all available to us systematically.

1.4 MOTIVATION

Sorting is an integral part of every computer application. Usage of correct algorithm for a large amount of data can result in saving large amount of power and time. The execution of entire system is decided upon by the performance of the sorting algorithms used. Sorting in C language had been implemented 2-3 decades ago, now we are implementing sorting algorithms using Verilog language, which is a user friendly language used for designing of digital systems. Another important deciding factor is that how much data can be sorted by an algorithm implementation keeping an optimal balance between hardware utilization, power and processing speed. Several algorithms to undertake the sorting process are Selection sort, Merge sort, Insertion sort, Heap sort, Radix sort, and Bubble sort. To sort large number of data we can make use of Heap sort, Radix sort, and Merge sort, that are proven to be quite powerful [15]. Meanwhile the Selection sort, Count sort , Insertion sort and Bubble sort are powerful for varied forms of data. Analyzing the sorting algorithms on FPGA like ZYNQ -7000 can give informative details about LUTs, FF and help us in analyzing their performance hence forth , which can then be later incorporated in System on Chip designs for integrating numerous applications.

1.5 ORGANIZATION OF REPORT

This dissertation is divided into 7 chapters. The 1st chapter gives an overview of the sorting techniques adopted since a long time . All the research that has been carried out centric on algorithms has been stated in 2nd chapter. While the six types of sorting algorithms that I have implemented on hardware have been formulated in chapter 3. Chapter 4 gives us a deep insight into the FPGA board used in this thesis. The design methodology has been explained in chapter 5 while results, and conclusion are in chapter 6 and 7 respectively. Finally all the sources of references that have helped in carrying out the thesis have been mentioned as per format defined whereas the research paper has been cited in the appendix.

CHAPTER 2

LITERATURE REVIEW

From decades we have been studying about sorting algorithms and scientists are involved in their research; to improve their implementation and hence increase the speed of their computation. Since 2009 E. Mansour et al [16] have addressed the aspect of software optimization in this paper wherein to meet the underlying user requirements while conserving maximum energy so that the software must be able to adapt itself . By maintaining an optimal balance between energy consumption and processing speed they have found out that for reducing energy consumption in a battery operated device like mobile phone, Insertion sort is the best choice for this purpose. B. Englert et al in [8] proposed the pipelined and optimal multiway Merge sort implementation as a basic block in simplified 2D array with reconfigurable pipelined bus system. Similarly another architecture for sorting exercise is depicted for Low power by P Y Chen in [6] which uses a pointer –like design wherein power dissipation is reduced by minimizing slew. This design of hardware accelerator is implemented in 90nm technology in which automatic Routing & Placement step is done by Synopsys IC.

Many researchers have focused on software implementations of many sorting algorithms as discussed in [17, 18] wherein comparison between Bubble sort & Selection sort and Merge & Quick sort were implemented respectively. Sort is an algorithm which is used for arranging all the elements in an ordered manner and in this paper, their performance is compared relative to time & space complexity. Irfan Ali et al in [18] also describes the memory usage of these algorithms by stating that the internal sorting algorithms only consume primary memory for their usage while external algorithms makes use of both.

This research is also focused on the computational methods of both algorithms wherein Merge sort is based on Divide and Conquer (DAC) methodology. Similarly in Quick sort the list is divided into two unequal units where this works recursively and these are implemented in Java programming language. A lower cost bubble sort is implemented in [17] and the focus of research was also on finding a similar low cost selection sort. We also find that the order of selection sort is nearly least than bubble sort.

Kazim Ali in [10] has shown a comparative study of various algorithms like Bubble sort, Selection sort, Radix sort, Insertion sort , Merge sort, Quick sort , Count sort and Bucket sort. The results of this analysis point to the fact that quick sort is a very stable algorithm and is the best choice on which the user can count on. We can prefer Count sort , Bucket sort or any linear algorithm when we have to arrange data like grades of students . An experimental study carried out by You Yang in [19] suggests that for small records we can either prefer insertion or selection sort while for large records Quick sort or our overall performing algorithm Merge sort is preferred. While in worst case scenario merge sort takes comparable time to quick sort. Importance of using multicore parallel processing over sequential processing for bubble sort & linear search is shown by K Sujatha in [3].
Efficient usage of CPU is carried out in multicore utilization where code is working on more than one core of a single CPU chip while being simulated. Another comparison between numerous algorithms have been carried out in [14] where quick sort scores the maximum in terms of performance .We come to know about the fact that efficiency of any sorting algorithm depends on its usage as shown by [20] where a review analysis of sorting algorithms is carried out. Bitonic sort was implemented by the latest technique in [12] on hardware in a special cubic tree network where in the timing was greatly optimized.

A. Srivastava et al [4] way back in 2005 came face to face with the fact that to accelerate sorting we employ parallel Bitonic sorting networks for hardware implementation. Also the well- known and widely used merge sort algorithm due to lack of parallelism in final stage suffers from low throughput resulting in trivial memory usage & low latency. A hybrid design was proposed wherein streaming permutation network (SPN) was used to realize all interconnection patterns or final stages in merge sort are replaced with ‘folded’ Bitonic merge networks and synthesized on Xilinx Virtex -7 FPGA board. This gives us very good throughput i.e. close to 10 Gbps and lower latency design. On the same route , S. Mashimo et al in [7] have proposed a high performance hardware merge sorter whose performance is compared to CPU & GPU. We have become aware of the truth that performance of the sorter is directly proportional to frequency, hence frequency drops with no of outputs that can be output every cycle. This criterion need to be carefully taken care of while implementing merge sort.

Another hardware sorter for the merge sort was proposed by W. Song in [5]. This implementation covered the parallel hardware design by utilizing 32 port parallel mere-tree which merges sequences at the rate of 32 numbers per cycle on Virtex-7 FPGA board. This implementation helped us in reducing the sorting time by around 160 times compared with sequential sorters. A novel method of discarding where the number of total registers used on Stratix III Altera FPGA were greatly used by F. A. Alquaied et al in [21] in which a new data item is inserted by using a constant clock rate mechanism . This synthesis and implementation was done in VHDL language for the RADAR applications.

Though it gave us energy efficient solution but the data dependency of the partition algorithm in software cannot be accounted here. Another hardware sorting unit having the potential to sort large amounts of data i.e. GB’s of data in linear time complexity was put into practice on Virtex-5 in research carried out in [22]. Analysis predicted to the usage of a FIFO- based Merge sorter & tree-based merge sorter at a very low cost . This optimized sorter demonstrated that almost half the FPGA resources can be saved by using partial run-time configuration. A throughput of 2GBps for problems related to FPGA memory was achievable by this method in 2011. High performance parallel Merge sort & Radix sort for many core GPUs are studied by N Satish et al in [23]. It makes use of the fact that binary representation of keys is directly manipulated by radix sort and only a comparison function of keys is required in merge sort. Keys are referred to the data elements that need to be sorted .This method is highly efficient and the radix sort is 4 times faster than the GPU sort , being 23% faster on average than the CPU routines and is depicted on several NVIDIA GPU’s like GTX 280 , 8800 Ultra among others.

The earlier implementations of sorting algorithms on FPGA were carried out by many researchers. One of them is implemented in [15] by A. Lipu on Spartan -6 FPGA by using the Xilinx-ISE software . This implementation was shown to be 10 times more rapid when compared with the sequential implementation for 20 data inputs and the hardware accelerator showed much better performance when tested on same constraints as that of the general purpose processor after studying sorting algorithms for several decades.

As we know when the delay of sorting unit decreases, its speed will increase exponentially. Magesh et al have tried to exploit this in [2] by proposing a scheme of sorting which divides the execution stage into 3 segments by using register delay. Combinational circuit with pipelined registers are used giving 64ns delay and a utilization of 179 slices area on Spartan – 6 whose clock speed is 400 MHz. The sorting units was composed of the compare –swap unit , comparator and two multiplexers to select the outputs . Efficient parallel implementation of the bubble sort and bitonic sort are synthesized in Verilog HDL language using the Xilinx – ISE software and using 6 clock cycles to obtain the output.

Y. B. Jmaa et al [24] recently in 2019 have implemented numerous sorting algorithms on ZYNQ 7000 FPGA i.e Bubble sort , Insertion sort , Selection sort, Quick sort , Heap sort , Shell Sort , Merge Sort and Tim Sort . They have analyzed the results for different data length and have concluded that Selection Sort gives us the fastest computation results for less than 64 bits. The research was focused in suggesting the usage of the sorting algorithms extensively for embedded system field like the avionics system. My aim was to simulate the behavior of elements for intelligent transportation system. Even High level synthesis was carried out and the design was optimized by using pipeline and loop unroll pragma whose unroll factor was 2. Whenever big applications where sorting of large no of elements is to be carried out then using a FPGA for sorting algorithms is a very attractive and cheap on pocket solution which can be extensively used now-a- days.

CHAPTER – 3

SORTING ALGORITHMS

The term sorting has been into practice since the word searching was quickly noticed by humans .Basically whatever things we have in our daily life ,we have to arrange them in a particular order so that they can be accessed quickly and also not look like a pile of shabbily placed things; for example roll numbers in a merit list, contact numbers in a telephone and books in a bookshelf so that the required book can be picked up quickly and help us in reducing our time for searching them; are among few of the examples that we encounter on a daily basis [17]. So basically sorting can be ‘‘termed as any process that makes searching easier by arranging the data in a particular sequence’’.

Now if we're discussing about the example of arranging cards in a deck, our first step would be to first check each & every card and then put the card in the particular deck of that color and so on [13]. But if you are going to follow this iterative procedure, then our computer will consume a very long time for sorting the data that is present in its memory, which is quite time-consuming and exhaustive. So our computers make use of the sorting algorithms to arrange the data in a particular sequence so that the power is optimized and it doesn't take us longer durations of time. In few algorithms the numbers to be searched are placed distant and their variability is also deeply affected by the sorting procedures we employ. From the times, the computer has come into existence, scientists are in the interminable research and appetence of a sorting algorithm that can work efficiently in terms of area, power and time. On the same pretext my research focuses on 6 sorting algorithms that have been implemented in hardware but have varied results for the area, delay and power efficiency. As tabulated in [19], merge sort is a potential candidate that scores well in all of these three domains, hardware results are no different from the behavior shown in the software.

The two criterion on which the designer or the scientist has to focus on while deciding on the usage of sorting algorithms are specified as :

1) Time taken to sort the data

2) Memory space required to do so

Though sorting algorithms can also get categorized depending upon their space, time complexity and on the area requirements.

When we have the task of sorting any type of data arranged in list, array etc. then some time is taken when we try to juxtapose one element to another element on the list and after that only we can swap or exchange these elements [14]. This type of sorting is called comparison based sorting. Algorithm and property for every sorting algorithm known to us varies a lot because every algorithm takes varied time and disparate memory allocation technique depending upon the data bits / words we are sorting etc.

3.1 BUBBLE SORT

It is one of the easiest algorithm to implement, requires very less time in developing and its realization but has a weak performance when a lot of large numbers are to be sorted. The basic principle utilized in bubble sort is to compare two consecutive numbers from leftmost to rightmost and swap these numbers if the underlying condition is met [15]. It is one of the classical algorithms which is now rarely used and is registered in the bygone lanes of the computer science student textbooks. Its basic procedure in layman terms can be expressed as, to repeatedly iterate through the entire list and then after comparing the two items at a time, we would be swapping them if the items present in the list are in the wrong order [16]. The worst - case time complexity is O(n^2) and the memory complexity is depicted by O (n).Therefore for smaller number of inputs bubble sort is the easiest to derive benefits from ,by using parallelization techniques it enacts as a feasible solution even for mid - range numbers . This performs M comparison and switching events in the first round when the input size is M and its running time is O ($n \log n$). The procedure that is adopted while arranging numbers is depicted in fig 3.1. Initially we have an unsorted array of numbers [5, 1 , 4 , 2 ,8].

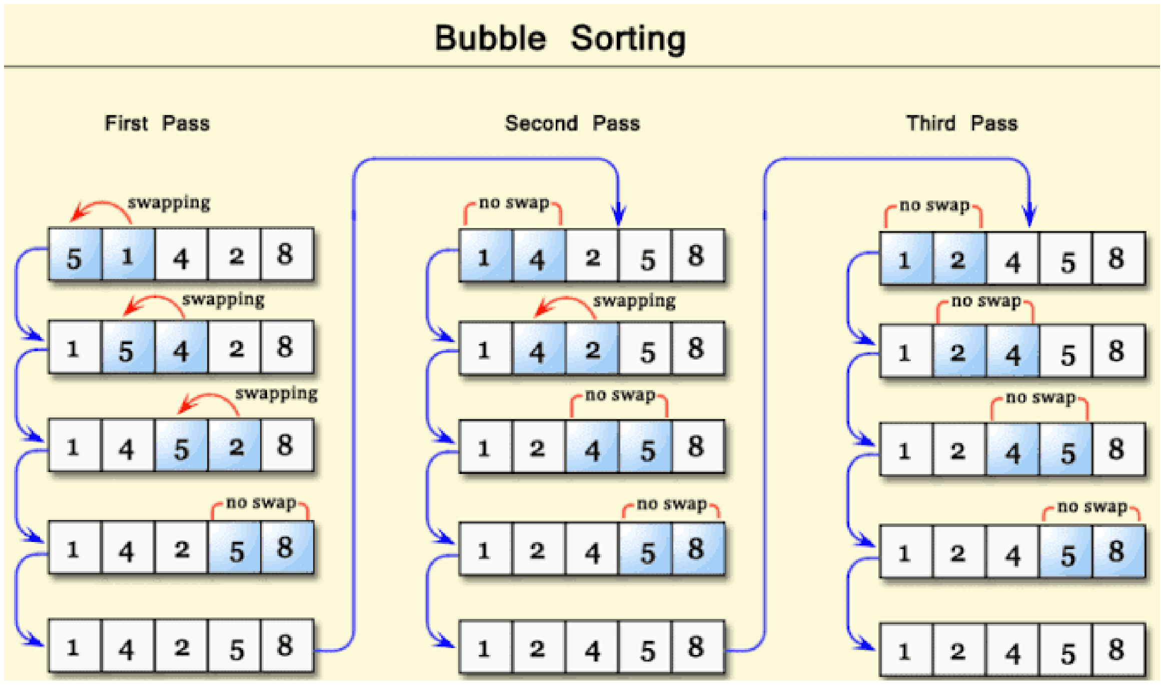


Fig 3.1 Bubble Sort example [15]

3.2 MERGE SORT

It is a comparison based sorting algorithm based on the basic “divide – and – conquer rule” which is a paradigm based on multi-branched recursion [25]. It belongs to the class of comparison based sorting algorithms, its implementations result in stable sort i.e. the equal elements have the same order in the input and in output. Its working is based on the age old tradition of Divide and Rule that was adopted by Britishers when they came to India. Its three basic steps are :

1. **Divide:** Dissolving the big problem into various atomic sub- problems; facilitating a deeper understanding of the data elements.

2. **Conquer:** Conquer the sub-problems by calling them recursively, they get solved.

3. **Combine:** Solve all the sub problems, hence find the problem solution swiftly.

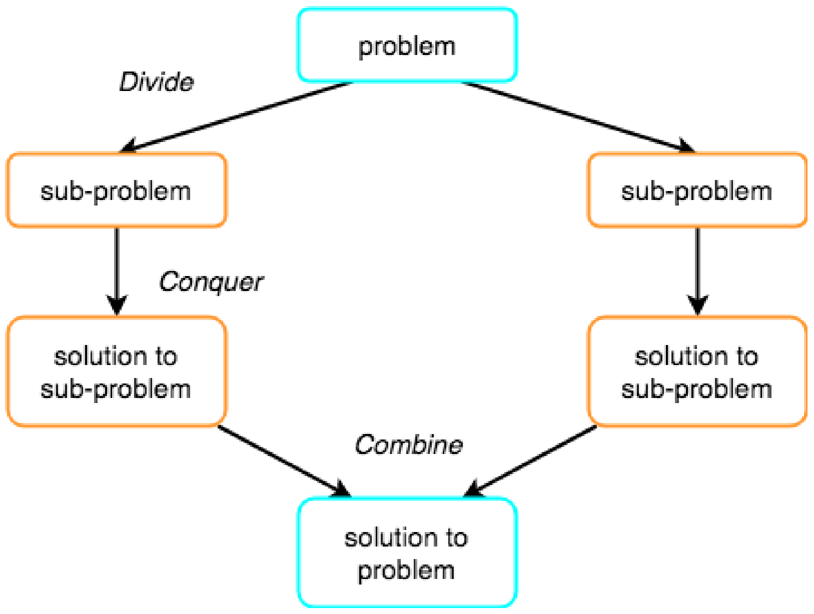


Fig 3.2 Procedure of Merge Sort Algorithm [19]

The basic procedure of Merge sort algorithm referenced from [19] is shown in fig 3.2. Robert Greene once said , “ Defeat them in detail : Look at the parts and determine how to control the individual parts, create dissension and leverage it”. The same rule was followed by John Von Neumann while inventing this popular algorithm. It can be explained as follows: At first the algorithm will begin with splitting the list/array of data into two smaller units, after that the comparison of the elements will be carried out with the adjacent list and then the two pieces or the units of data are sorted and set recursively thereafter, consequently it will merge and sort all other elements of the list and hence the list gets sorted at quite a fast pace [18]. Theoretically, merge sort will perform repeated operations to split the disordered list into n elements sub-units and thereby comparing each and every element of the list with the single element observed as sorted. Its processing speed is very fast and is therefore quite efficient for a large amount of data. It gives a comparison of each element index, chooses smallest element, separates it out in two arrays and finally the two sorted arrays are merged to give us the final result [24]. It’s time complexity is denoted as: O ($n \log n$). Whenever the worst case scenario appears, the merge sort works as the most competitive alternative to vote for and it’s execution capabilities have been shown in research carried in [19] too.

3.3 Divide & Conquer (DAC) Algorithm

This is depicted below [25] as used in software and it should be used only when similar sub-problems are not evaluated several times continuously. Also the example of sorting algorithm referenced from [15] has illustrated the step – by – step procedure used for arranging the numbers [9, 10, 6, 4, 5, 7, 19, 11] in fig 3.3. This strategy is employed in numerous applications like Closest Pair of Points, Strassen's Multiplication, Karatsuba Algorithm, Cooley-Tukey Algorithm, Quick Sort, Binary Search etc. Also it’s most recognizable advantage is that, it enhances the programmer with skills that can be utilized for solving difficult mathematical problems like the Tower of Hanoi very easily.

```
DAC(a, i, j)
{
   if(small(a, i, j))
     return(Solution(a, i, j))
   else
     m = divide(a, i, j)              // f1(n)
     b = DAC(a, i, mid)               // T(n/2)
     c = DAC(a, mid+1, j)             // T(n/2)
     d = combine(b, c)                // f2(n)
  return(d)
}
```

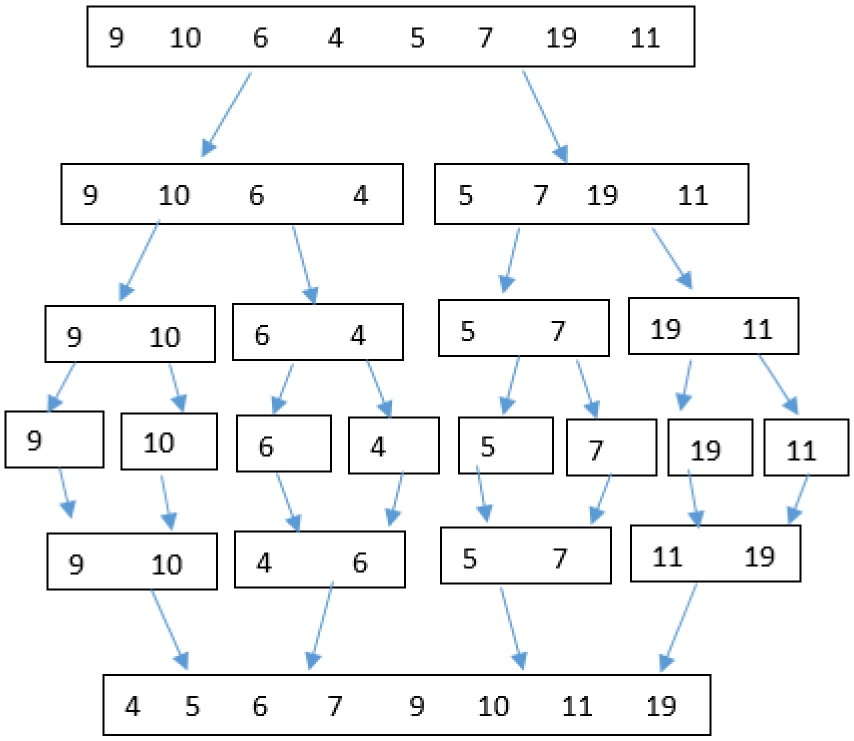


Fig 3.3 Example of Merge Sort [18]

3.3 INSERTION SORT

It is an ingenious algorithm belonging to the family of comparison based sorting algorithms and is a particular example of an incremental algorithm that is; it builds the sorted sequence by considering one number at a time i.e slowly and steadily. In the sorting procedures, we can identify all the data that are placed from positions 1 to N and they are later on inserted into the proper position after the comparison of the individual elements is done . Also n-1 passes are required for sorting these data that will later define the sorting time. We can even briefly explain this algorithm by considering the example of the card player who is arranging the cards being dealt to him. The player picks up the card and then insert them into the required position, also at each and every step we place the item onto its specified place [14].

We need (N-1) comparisons (at most) to insert the last element while sorting is carried out, so lets calculate the time complexity as follows :

We will calculate the number of comparisons of an array of N elements:
0 comparisons are required to insert the first element
1 comparison is required to insert the second element
2 comparisons are required to insert the third element
...
We therefore require (N-1) comparisons (at most) to insert the last element in the array

Summing it all we get,
1 + 2 + 3 + ... + (N-1) = O (n^2).

Hence O (n^2) is the time complexity depicted by Insertion Sort Algorithm. Insertion Sort can further be explained by considering a simple example referenced from [20].

Initially we have an unsorted list of numbers [85, 12, 59, 45, 72, 51] , the step by step procedure of arranging them is depicted below in fig 3.4

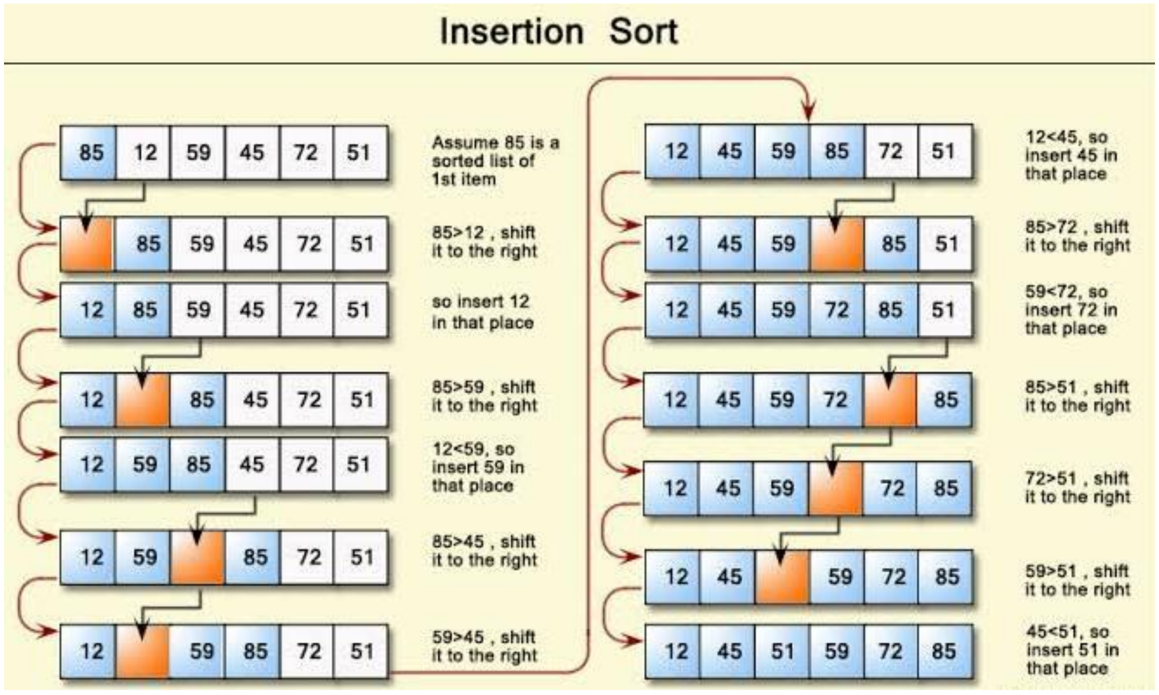

Insertion Sort
Assume 85 is a sorted list of 1st item
85>12 , shift it to the right
so insert 12 in that place
85>59 , shift it to the right
12<59, so insert 59 in that place
85>45 , shift it to the right
59>45 , shift it to the right
12<45, so insert 45 in that place
85>72 , shift it to the right
59<72, so insert 72 in that place
85>51 , shift it to the right
72>51 , shift it to the right
59>51 , shift it to the right
45<51, so insert 51 in that place

Fig 3.4 Insertion Sort example [20]

3.4 COUNT SORT

Used for sorting the range of numbers from 1 to k where k is the smallest value. The basic notion behind this algorithm is to determine the rank of each elements and the rank of the element is the number of elements which are less than or equal to that number [10]. Once that has been determined, we will copy the numbers to the final array .The counting sort algorithm takes O (n + k) time for its processing and hence that is termed as its time complexity.

It is asymptotically faster than several comparison based algorithms like merge sort, quick sort etc. It only works better when the restricted inputs are used in array. When the range of potential values is large, then a lot of space is required i.e. depicted by its space complexity given as O (n) [28].

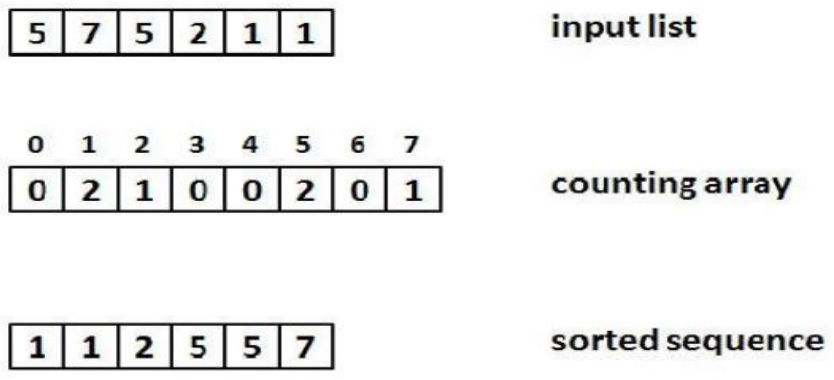


Fig 3.5 Count Sort example [28]

3.5 RADIX SORT

It is a non-comparative type sorting algorithm i.e. comparisons are avoided by creating & distributing elements into buckets according to their radix. Even though it is not an in – place algorithm, it is stable in nature [3]. The sorting procedure is:

- Initially sort the most significant digit i.e. MSB bit, then the next most significant digit and continue this procedure
- Finally when you have sorted all the bits sort the least significant digit.

Count sort is a linear time sorting algorithm and therefore it cannot be used for elements that have a very big range like 1 to N^2 , as its worst case is even terrible when compared with majority of the comparison based sorting algorithm ; hence it is seldom used . Therefore to sort an array in real-time we make use of Radix sort, i.e. for sorting elements ranging from 1 to n^2. It can be explained with the help of fig 3.6 referenced from work done in [23]. The numbers used in this example are [326 , 453, 608 , 835, 751, 435, 704, 690].

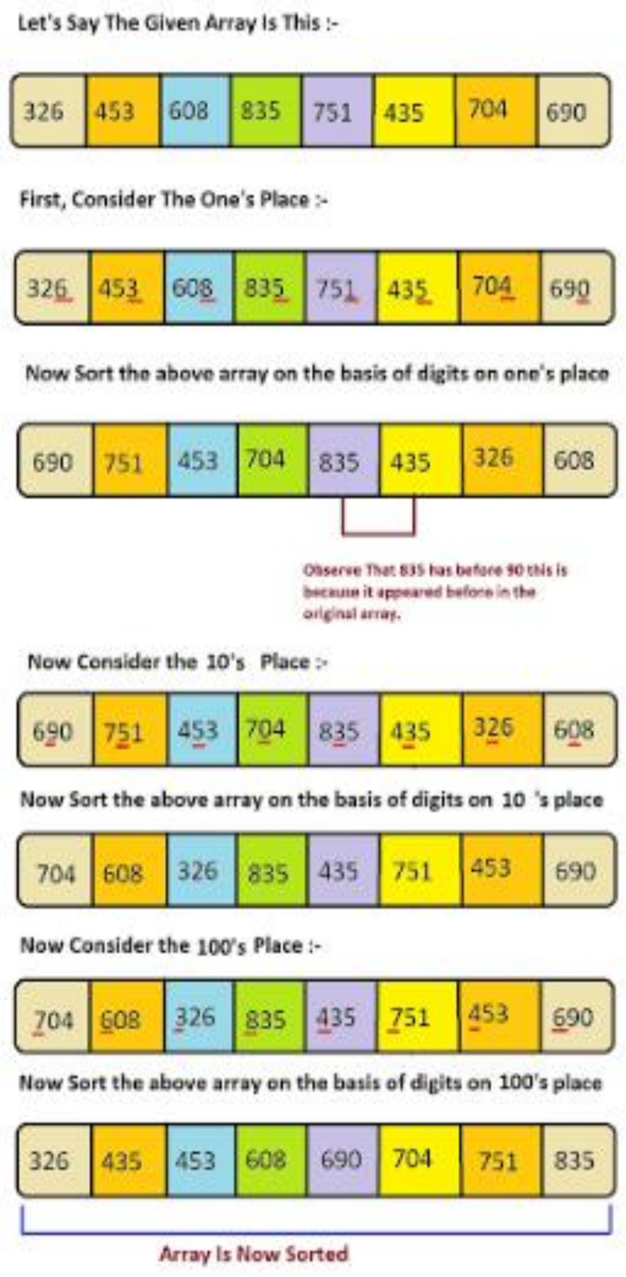

Let's Say The Given Array Is This :-
326 453 608 835 751 435 704 690
First, Consider The One's Place :-
326 453 608 835 751 435 704 690
Now Sort the above array on the basis of digits on one's place
690 751 453 704 835 435 326 608
Observe That 835 has before 90 this is because it appeared before in the original array.
Now Consider the 10's Place :-
690 751 453 704 835 435 326 608
Now Sort the above array on the basis of digits on 10 's place
704 608 326 835 435 751 453 690
Now Consider the 100's Place :-
704 608 326 835 435 751 453 690
Now Sort the above array on the basis of digits on 100's place
326 435 453 608 690 704 751 835
Array Is Now Sorted

Fig 3.6 Radix Sort example [23]

3.6 SELECTION SORT

Though it cannot be implemented as a stable algorithm but still it is similar to the bubble sort algorithm, in the manner that it can be used efficiently for small number of elements however, is inefficient when the task of sorting a large number of elements arises; as its complexity is quite high i.e. it has a time complexity of O (n^2) $[24]$. Its name is derived from the fact that it works by selecting a minimum of elements in each step of the sort and the most important step is to pick the minimum value at index 0. It is an in- place sorting algorithm and therefore is a simple algorithm that can be used for analysis of data. It basically selects from an unsorted list the smallest element in each iteration and marks it in the unsorted list as the first element i.e. at the MSB position of the list / array of input numbers.

The working procedure can be explained as follows :

- The initial step begins from 0 index position of the array, i.e. the first element; we find the smallest element in the array, and replace it with the element in the first position.
- After that we move on to the element present on the second position, and then look within the subarray for the presence of smallest element, starting from index 1, till the last index.
- Third step involves replacing the element at the **second** position in the original array, or we can say the MSB position bit of the subarray, with the second smallest element.
- Above step is repeated, until the entire array is sorted [10].

The step – by – step procedure of the data elements [29, 72, 98, 13, 87, 66, 52, 51, 36] to be sorted is represented in fig 3.7

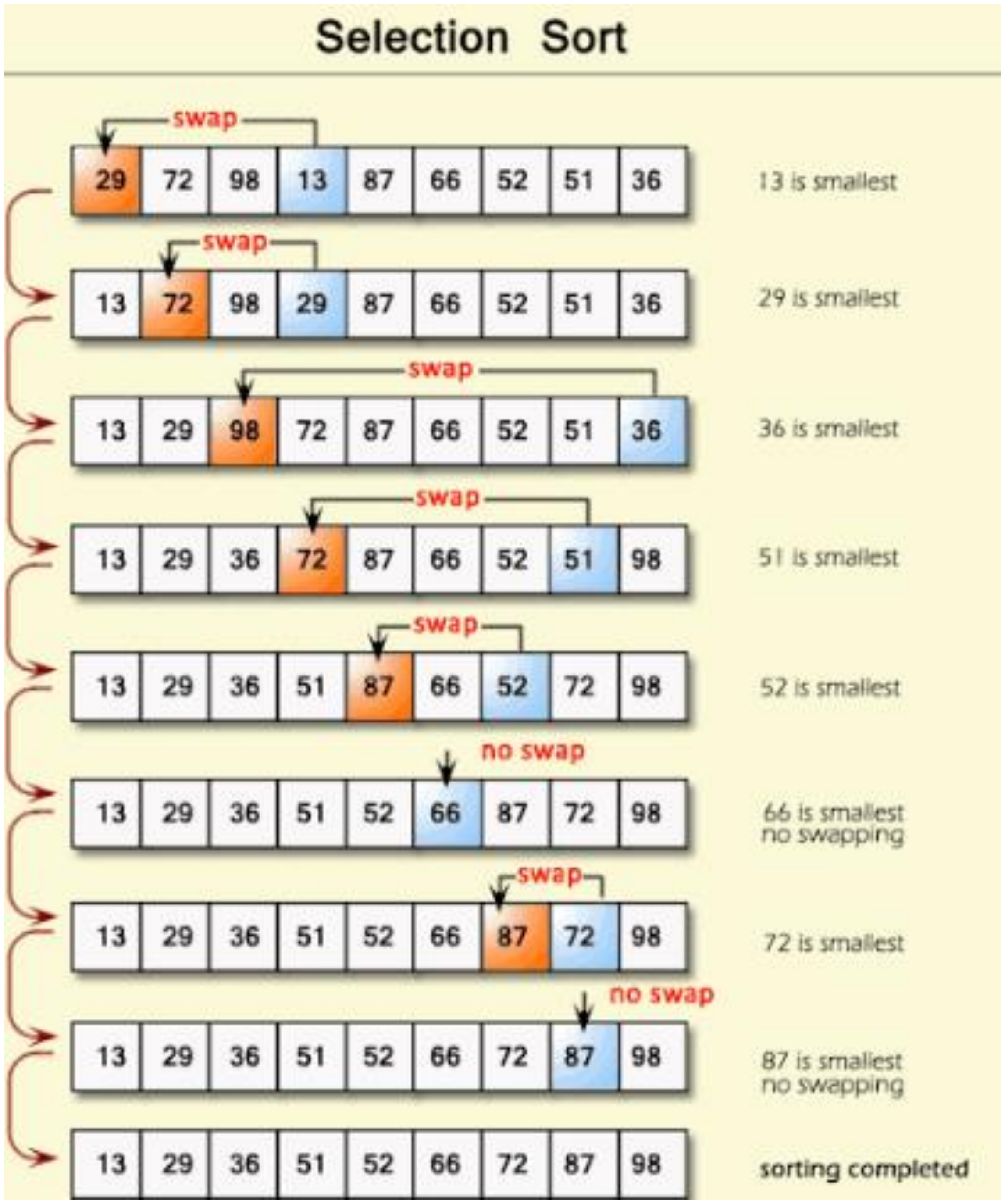


Fig 3.7 Selection Sort example [10]

CHAPTER – 4

ZYNQ 7000 FPGA FAMILY

This FPGA family has a System - On -Chip (SOC) type architecture that encapsulates a dual core ARM Cortex A9 microprocessor chip and 28nm Programmable Logic in a single device. The A9 chip is also the heart of the Processing System (PS) and makes this FPGA the ideal choice to be used as an SOC. It also includes an on-chip boot Read only Memory (ROM), external memory interfaces of 16 to 32 bits and a few peripheral interconnect interfaces. While providing solutions to ASIC and system-on-chip users a fully programmable alternative, it facilitates a flexible platform to launch new solution. This acts as a highly integrated and our optimized alternative idle for computationally extensive and performance demanding applications. This FPGA architecture's primary focus is on automative applications and its members consists of these Z-7010 Z-7020 Z-7030 devices [29].

The ZYNQ 7000 is optimised for maximum design flexibility and performance per watt. To enable highly differentiated designs the dual core arm cortex A9 processor integrated with 7 series Programmable Logic (up to 6.6 M logic cell and 12.5 GB per second trans- receiver) for a wide range of embedded applications. These SOCs are cost optimized entry point with ARM single Core processor acquaintance with 28nm Artix 7 based Programmable Logic. The system on chips are an ideal candidate for a numerous applications in the motor control field and vision engineering. The Programmable Logic embedded in these are connected to a system with over 3,000 interconnects providing 100 GB/s of I/O bandwidth, beyond of a multi-chip solution. This family contains up to 10 devices to be chosen from that may be single or dual core, hence allowing a scalable platform for the consumer. FPGAs are the ideal candidates for implementing sorting algorithms due to their unparalleled features like parallelism, low latency , high bandwidth , faster processing speed , in-built processor core availability among others [24]. This family provides the flexibility and scalability of an FPGA while facilitating the performance and ease that is typically associated with ASIC and ASSPs.

The Zynq-7000 SoC devices are able to provide numerous applications including:

- Automotive driver assistance : lane departure , blind spot detection
- Wireless applications, Reliable ethernet
- Embedded prototyping
- IP and smart camera
- LTE radio and baseband
- Medical diagnostics and imaging
- Software acceleration for DSP functionality
- Time domain reflectometer

The ZYNQ conveniently maps the s/wre and custom logic in the PL and PS respectively and enables realization of unique and differentiated system functions. Its major blocks are:

- Processing System (PS)
- Application processor unit (APU)
- Memory interfaces
- I/O peripherals (IOP)
- Interconnect

- Programmable Logic (PL)

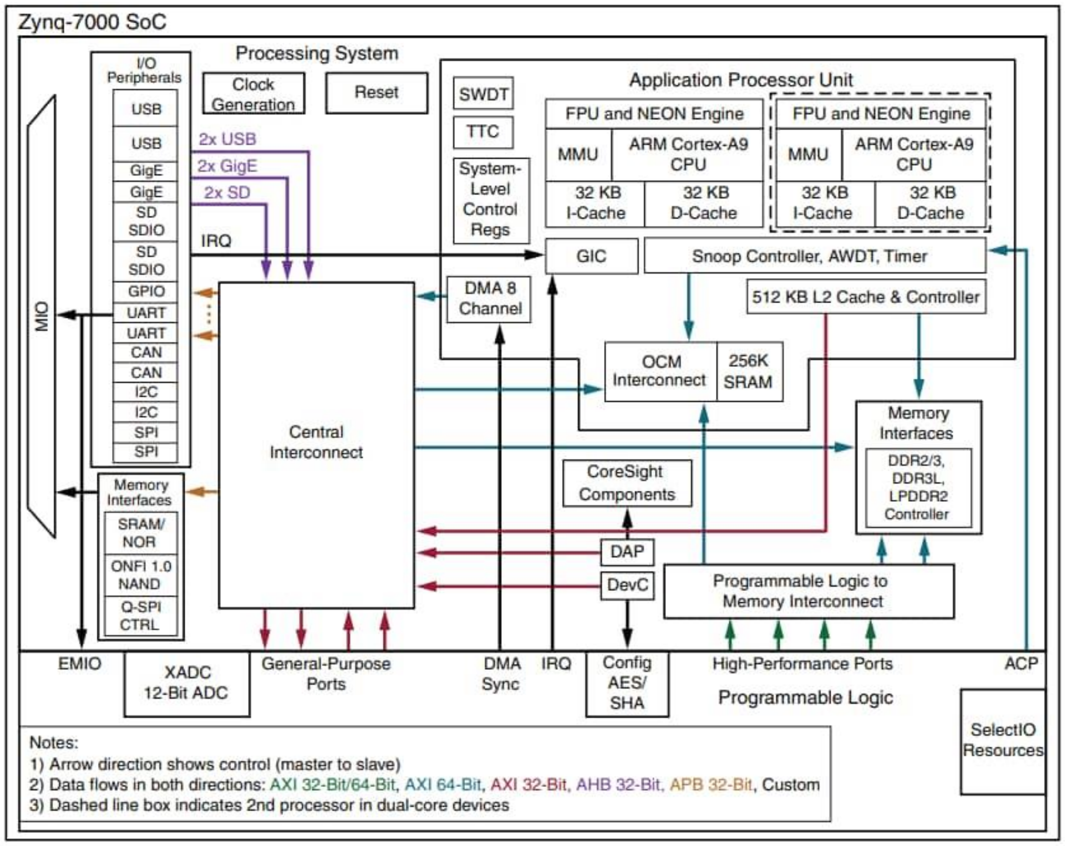


Fig 4.1 Architecture of ZYNQ FPGA chip [29]

4.1 HISTORY

It is an all programmable system on chip that is embodied by two hard processors, ADC block Programmable Logic (PL) and a lot more components embedded in one Silicon chip only. Before the innovation of the ZYNQ came into practice, the processes were coupled with a FPGA which made the communication between the Processing System & the Programmable Logic quite intricate and its layout difficult for the engineers to understand. The advanced extensible interface (AXI) standard is used as a means of interfacing across different elements present on the ZYNQ architecture which thereby accounts for the high bandwidth and low latency present in connections. A soft core processor such as Microblaze was being used by the users before the ARM processor was implemented inside the ZYNQ device; as its heart. The upper hand provided by the Microblaze till date, is the flexibility of the processor instances within a design. On the other hand ZYNQ delivers significant performance enhancement with the encompassing of the hard processor in the ZYNQ [30]. Also the cost to market and the physical size gets reduced by simplifying the system to a single chip.

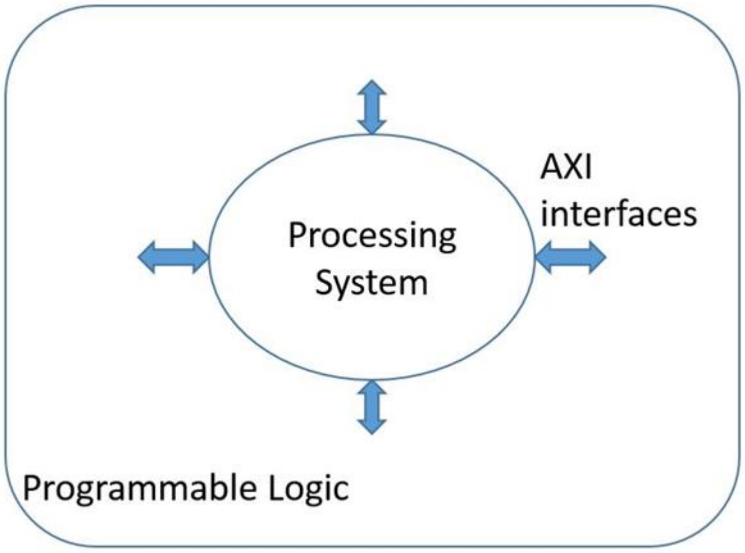


Figure 4.2 ZYNQ overall view [30]

4.2 ZYNQ DESIGN FLOW

This design flow has some steps that are recurrent to a regular FPGA. We start the design cycle by first defining the requirements and specifications of the system, next the different tasks are assigned to implement in either the PL or PS which is called task partitioning. Because the overall performance of the system will depend on the task or function being assigned for implementation, so this stage is most important in the technology node be it hardware or software. Next step is the testing of the hardware and software development. We need to now identify the functional blocks related to the Programmable Logic in order to attain the design characteristics and also to congregate them as IPs and for facilitating the connections between all of these IPs, all the steps are hence governed with respect to the functionality of the Programmable Logic (PL). The software activity includes to run on the PS, the code that is developed [30]. To wrap the design, system integration and testing is needed. Figure 4.3 gives design cycle briefly.

4.3 PROCESSING SYSTEM UNIT

It comprises of four major blocks which are the Application Processing Unit (APU), the memory interfacing, interconnect and the input output (I/O) peripherals.

4.3.1 Application Processing Unit (APU)

Two Cortex A9 processor units are present in it along with NEON Unit, Memory management unit, floating point unit ,L1 caches. Additionally L2 cache and Snoop controls are also present in it.

- NEON : the implementation of the single instruction multiple data in the ARM processor is provided by this unit that acts as a catalyst to the DSP and the media algorithms
- FPU: the floating point unit operations are managed by this unit
- Level 1 Cache : storing the instructions and the data separately we have a data and instruction cache
- MMU: the virtual memory gets translated to the physical memory address by this unit.

- Snoop control unit (SCU) : its main task is to create interfaces among the processors 4 –way set associative L1 and L2 cache.
- L2 cache: to check the current updated value of a variable , cache is shared between two processors

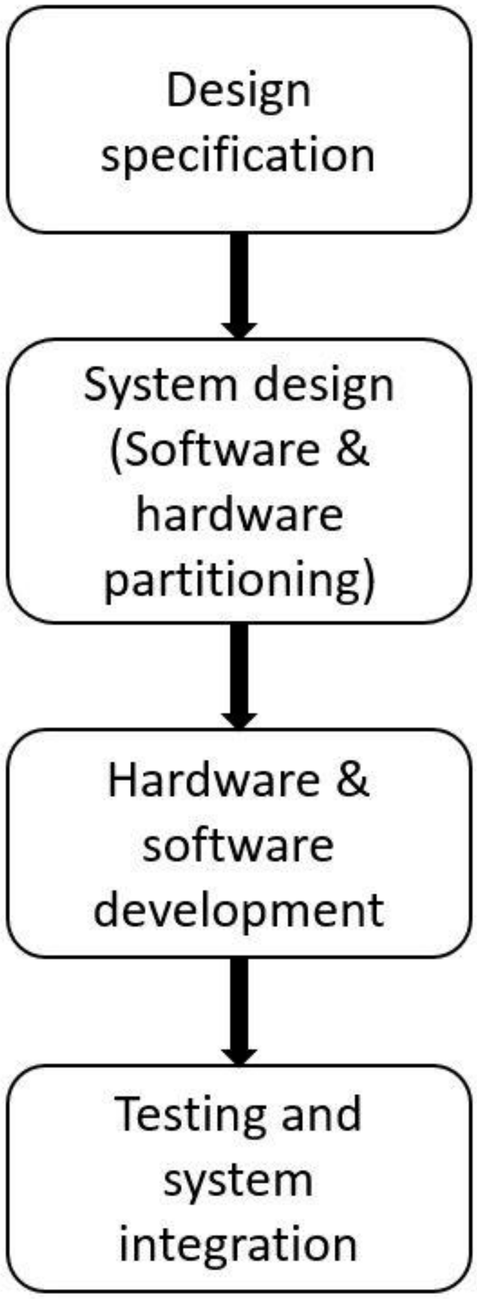

Design specification
System design (Software & hardware partitioning)
Hardware & software development
Testing and system integration

Fig 4.3 ZYNQ Design Flow Steps [30]

4.3.2 General Interrupt Controller (GIC)

Consists of two main components :

- Three watch dog timers (WDT) (one per CPU and one system WDT)
- Two triple timers/counters (TTC)

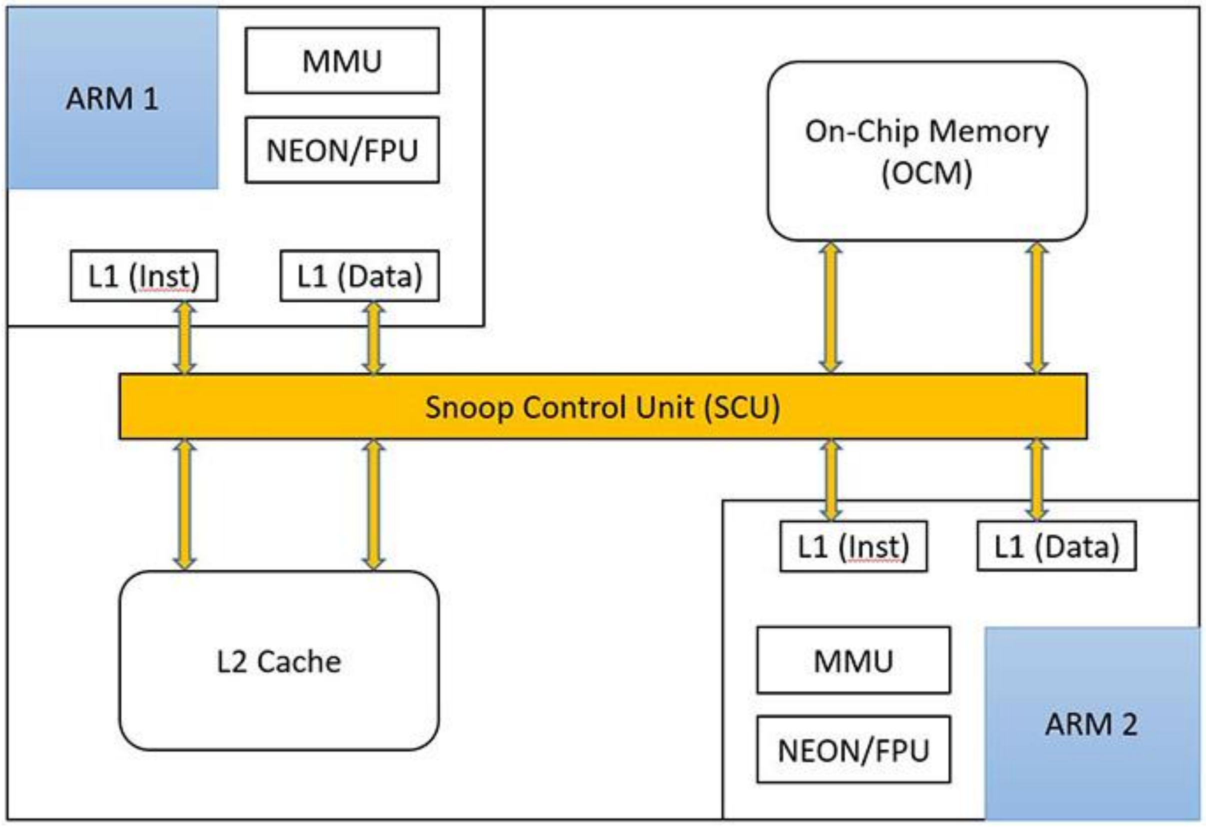


Figure 4.4 Application Processing Unit Structure [29]

4.3.3 Dynamic Memory Interfaces

Multi-protocol memory controller can be configured to provide 16 bit or 32 bit byte accesses using a single rank configuration of 8 bit ,16 bit or 32 bit to a 1 GB address space [29]. It incorporates it's own set of dedicated IOS and hence speed of up to 1333 MB per second for the DDR 3 is supported. 4 AXI slave ports are featured for this purpose namely as :

- via the L2 cache controller : a 64-bit port is dedicated for low latency
- for the PL access: two 64- bit ports are designated
- all other AXI Masters share: one 64-bit port via the central interconnect

4.3.4 Static Memory Interfaces

It supports static external memory such as :

- 8 bit data bus upto 64-MB
- 8 bit parallel NOR flash up to 64- MB
- NAND flash support with 1-bit ECC

4.3.5 I/O Peripherals (IOP)

The data communication peripherals are present within the IOP unit. Its features are;

- Ethernet MAC peripherals : Tri Mode
- Supports an external PHY interface
- High speed and full speed mode in host, devices by presence of two USB 2.0 OTG peripherals having 12 endpoints
- Makes use of the 32 bit AHB slave and AHB DMA master interfaces
- Two full CAN (Controller Area Network) bus interface controllers that has automotive applications
- Three peripheral chip select signals accompanied by 2 full duplex SPI ports
- Two UARTS

- Up to 118 GPIO bits

4.3.6 Interconnect

A multilayered ARM AMBA AXI interconnect is used to connect the APU, memory interface unit and the IOP to each other and also to the PL .This interconnection supports multiple simultaneous transactions of the master and slave ; it’s a non - blocking type. And is therefore designed with latency sensitive masters which have the shortest path to the memory and the bandwidth critical Masters having highest throughput with the slaves through which they have to communicate [29]. The traffic generated by the CPU, DMA controller can be regulated by means of a block known as the quality of service (QoS) present in the interconnect .

4.3.7 PS External Interfaces

They cannot be assigned as PL Pins and are hence designed specifically for the purpose of interfacing They interfaces are encompassed by :

- Clock, reset, boot mode, and voltage reference
- 32-bit or 16-bit DDR2/DDR3/DDR3L/LPDDR2 memories

4.4 MIO (MULTIPLEXED INPUT – OUTPUT) OVERVIEW

There are up to 54 MIO present for multiplexing access to PS pins that can be used by the static memory interface and the PS interfaces, which can be mapped with the different peripheral pins at all steps of interfacing framework [29]. The signal routing of the MIO block has been shown below in fig 4.5. If greater than 54 are required then we need to route this through the PL to the input output associated I/O with the PL and are therefore referred to as the extendable multiplex input output (EMIO). MIO_PIN_[53:0] configuration registers located in the SCLR registers set is controlled by the signal routed through the MIO block [29] . We can program any one of the total pins present on the MIO pins to the reference clock of an external CAN controller . A typical module block diagram of EMIO to PL has been shown in fig 4.6.

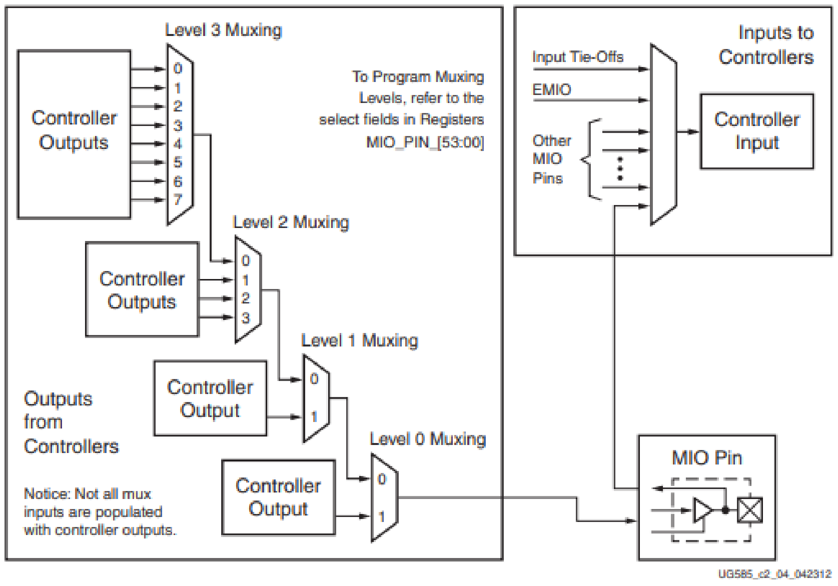


Fig 4.5 MIO Signal Routing [29]

4.5 PROGRAMMABLE LOGIC STRUCTURE

It is comprised of configurable logic blocks (CLBs) which are housed by slices like any other FPGA we are familiar with. Any slice contains a combination of 8 flip flops + 4 LUTs and is accompanied by a switch matrix, also there are DSP slices and Block RAMs as well. Figure 4.7 shows the structure of the Programmable Logic. Its main components are:

- Slice: It is embodied by resources for implementing the combinatorial and the sequential circuits of the design.

- Look-up-table (LUT) : For implementing a logic function of inputs upto 6 or more RAM and ROM shift registers are used.

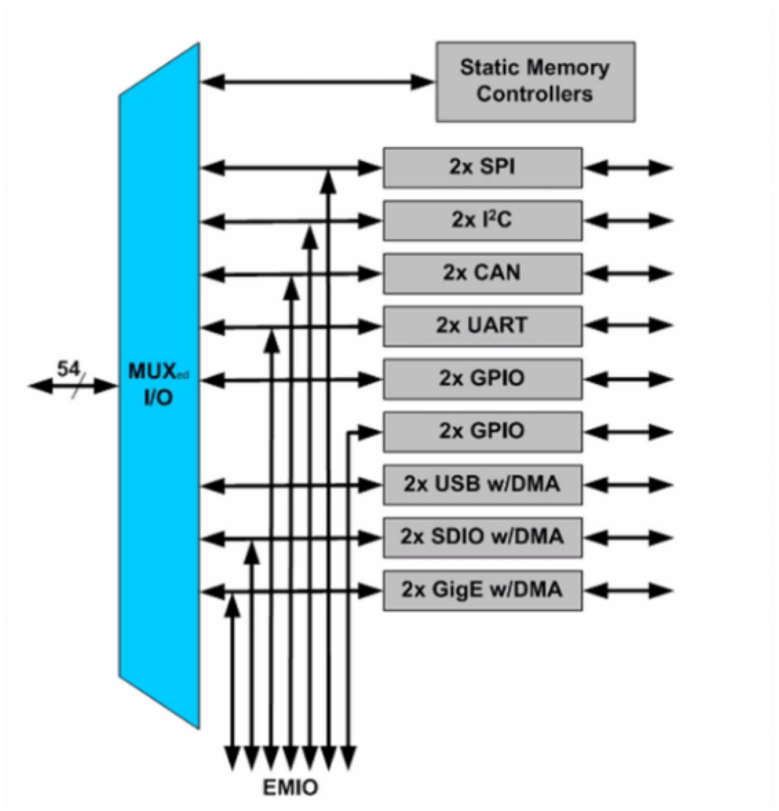


Fig 4.6 Module block diagram of EMIO to PL [30]

- Flip Flop (FF) : Usually employed for implementation of 1 bit register with a reset functionality.
- Switch Matrix : The connections among different parts present within the combinational logic blocks as well as with other CLB and other parts of the programming logic structure.

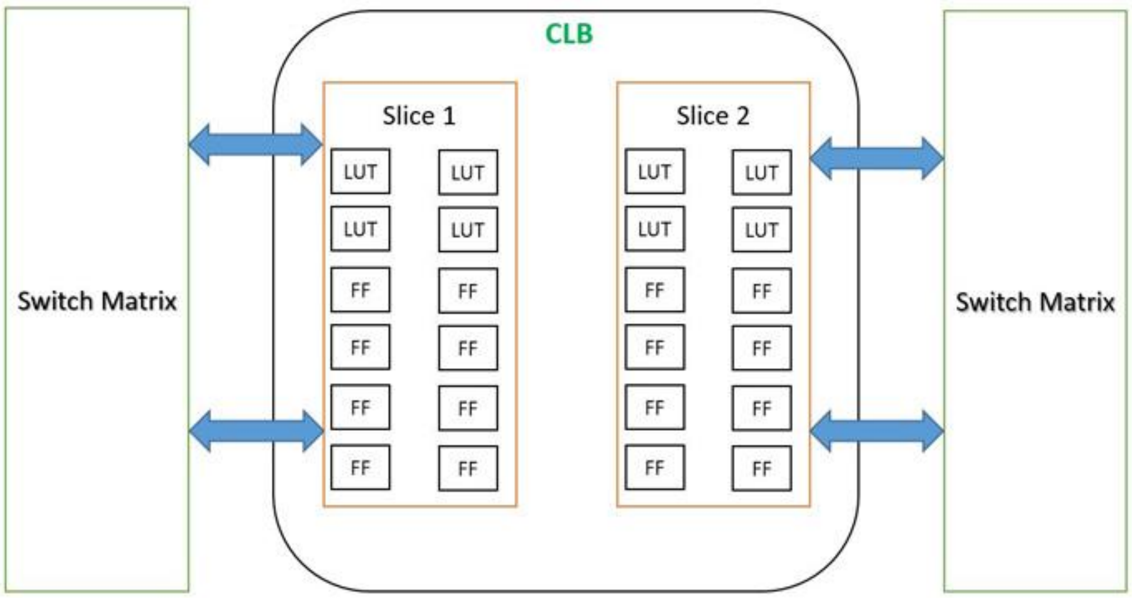


Figure 4.7 Structure of the PL [30]

4.6 DISTINCTIVE FEATURES

The key features of the ZYNQ 7000 are:

•1 GHz CPU Frequency

•GPIO has four 32-bit banks

•All programmable SOC

• Level 1 cache : 32 KB each

•Level 2 Cache : 512 KB

•Upto 118 GPIO bits

•Over 85k logic cells

•256 KB on chip memory (RAM) with byte parity support

•ARM v7 architecture with Trust – Zone security

•100 Gb/s of I/O bandwidth

•8 channel DMA ; 4 channel dedicated specifically to PL

• 8 LUTs + 16 FF per CLB

•Two 12 bit ADCs (XADC)

•36 KB Block RAM

•25 bit pre-adder, 18 x 25 signed multiply

•1.2 V to 3.3 V I/O capability

•ARM Cortex A9 Microprocessor chip

•PCIe supports upto 8 lanes, Gen2 speed

•4 AXI ports : configurable as 32 or 64- bit interfaces

•1KB deep FIFO ; 32 word buffer for read acceptance

•16 Interrupts available

•Upto 220 transreceivers for enhanced capability

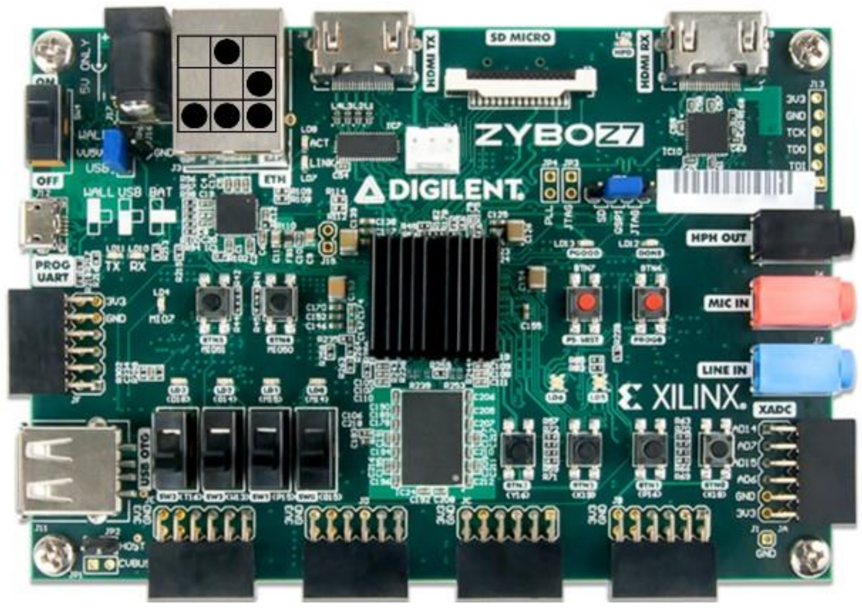


Figure 4.8 ZYNQ 7000 FPGA board [31]

•1-bit , 2 bit , quad SPI , or two-quad SPI

•Single & Double Precision Vector Floating Point Unit (VFPU)

•Scatter-gather DMA capability

•Two USB 2.0 OTG peripherals

•8-bit PHY external interface

•1 Mb/s high Speed UART's

•XADC , JTAG interfaces

•Processor configuration access port (PCAP) for facilitating chip security

• PS boot image authentication

•8 Clock Management Tiles (CMT)

• CMT has Mixed – Mode Clock Manager (MMCM) & PLL

•Memory controllers like USB, Gigabit, Ethernet, SD-SDIO

•AES & SHA 256b Decryption present for secure boot

CHAPTER 5

DESIGN METHODOLOGY

While implementing all the algorithms in Verilog Hardware language we have considered five 16 bit vector inputs denoted as [15:0] in1,in2, in3,in4 ,in5 ; corresponding to which we get five 16 bit outputs as [15:0] out1, out2,out3,out4,out5 . A 16 bit array of [1:5] size is also used to store the inputs. Five data registers each of 16 bits denoted as data1, data2, data3, data4, data5 have also been used while implementing all the algorithms. The input values are sent to the data registers sequentially using blocking assignment statements.

The procedure carried out in these algorithms have been formulated in flowcharts that have been depicted in below figures. This shows the algorithm that I have followed while studying their behavior on hardware using Verilog HDL language. Five inputs of 16 bits have been denoted as in 1 , in 2 , in 3, in4 , in5 [15:0]

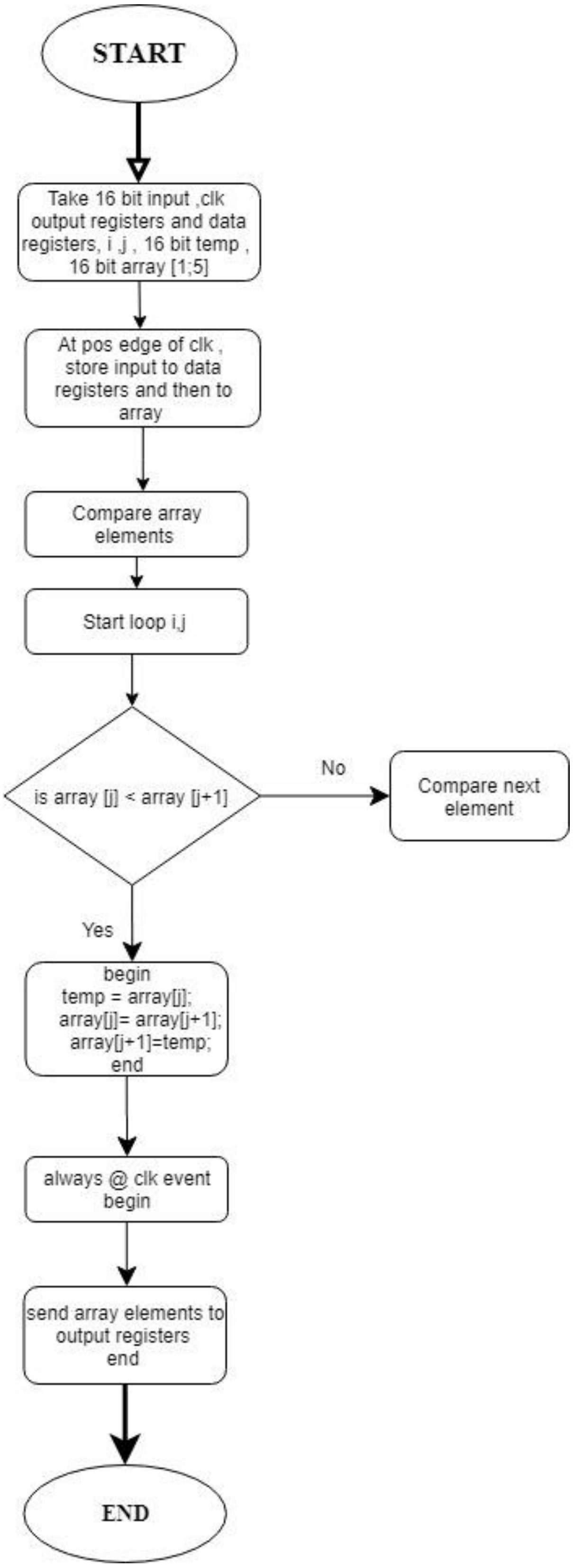

START
Take 16 bit input ,clk output registers and data registers, i ,j , 16 bit temp , 16 bit array [1;5]
At pos edge of clk , store input to data registers and then to array
Compare array elements
Start loop i,j
is array [j] < array [j+1]
No
Compare next element
Yes
begin
temp = array[j];
array[j]= array[j+1];
array[j+1]=temp;
end
always @ clk event begin
send array elements to output registers end
END

Fig 5.1 Bubble Sort Flowchart

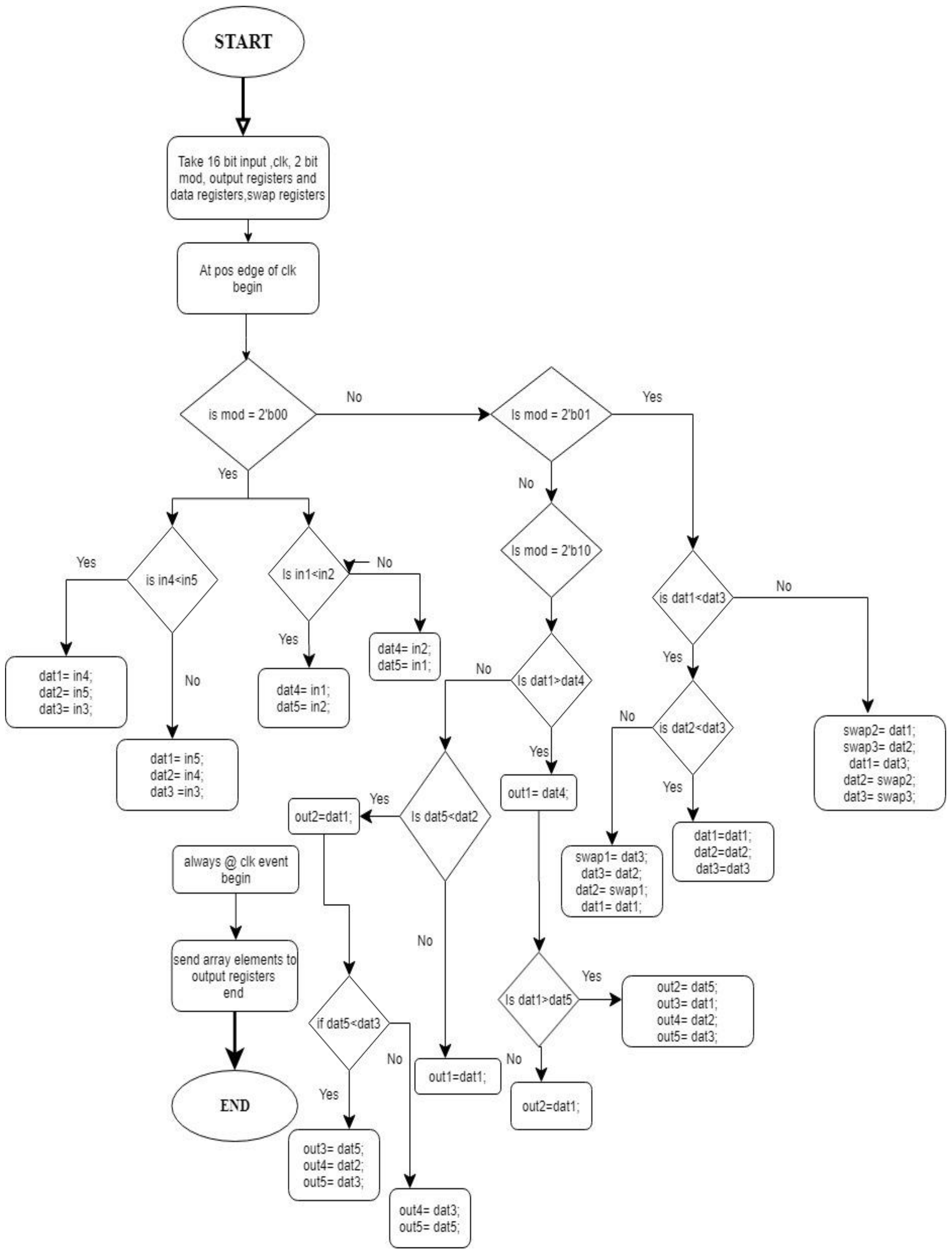

START
Take 16 bit input ,clk, 2 bit mod, output registers and data registers,swap registers
At pos edge of clk begin
is mod = 2'b00
No
Yes
Is mod = 2'b01
Yes
No
Is mod = 2'b10
is in4<in5
Yes
No
dat1= in4; dat2= in5; dat3= in3;
dat1= in5; dat2= in4; dat3 =in3;
Is in1<in2
No
Yes
dat4= in1; dat5= in2;
dat4= in2; dat5= in1;
is dat1<dat3
No
Yes
swap2= dat1; swap3= dat2; dat1= dat3; dat2= swap2; dat3= swap3;
is dat2<dat3
No
Yes
swap1= dat3; dat3= dat2; dat2= swap1; dat1= dat1;
dat1=dat1; dat2=dat2; dat3=dat3
Is dat1>dat4
No
Yes
out1= dat4;
Is dat5<dat2
Yes
No
out2=dat1;
out1=dat1;
if dat5<dat3
Yes
No
out3= dat5; out4= dat2; out5= dat3;
out4= dat3; out5= dat5;
Is dat1>dat5
Yes
No
out2= dat5; out3= dat1; out4= dat2; out5= dat3;
out2=dat1;
always @ clk event begin
send array elements to output registers end
END

Fig 5.2 Merge Sort Flowchart

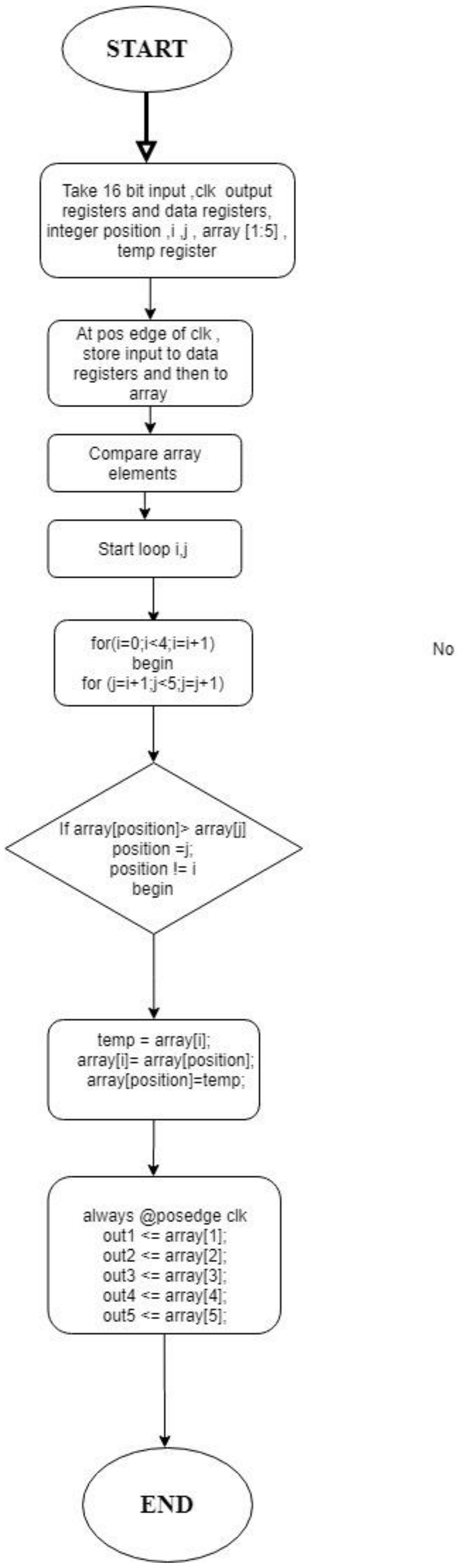

START
Take 16 bit input ,clk output registers and data registers, integer position ,i ,j , array [1:5] , temp register
At pos edge of clk , store input to data registers and then to array
Compare array elements
Start loop i,j
for(i=0;i<4;i=i+1) begin for (j=i+1;j<5;j=j+1)
No
If array[position]> array[j] position =j; position != i begin
temp = array[i]; array[i]= array[position]; array[position]=temp;
always @posedge clk out1 <= array[1]; out2 <= array[2]; out3 <= array[3]; out4 <= array[4]; out5 <= array[5];
END

Fig 5.3 Insertion Sort Flowchart

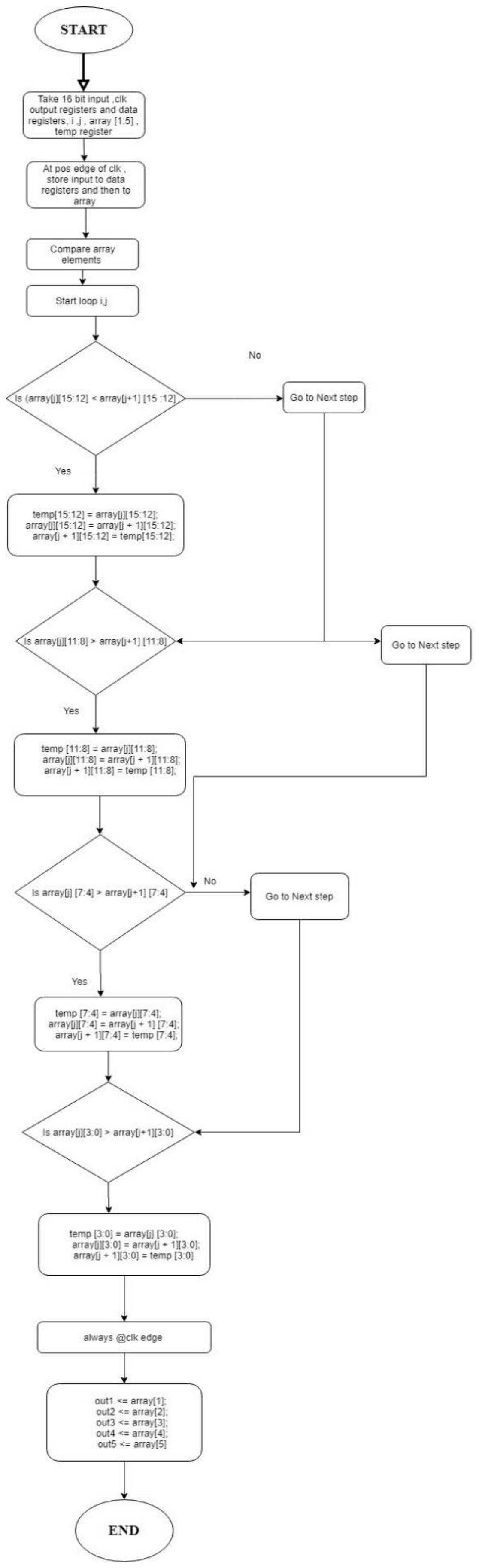

START
Take 16 bit input ,clk output registers and data registers, i ,j , array [1:5] , temp register
At pos edge of clk , store input to data registers and then to array
Compare array elements
Start loop i,j
Is (array[j][15:12] < array[j+1] [15 :12]
No
Go to Next step
Yes
temp[15:12] = array[j][15:12];
array[j][15:12] = array[j + 1][15:12];
array[j + 1][15:12] = temp[15:12];
Is array[j][11:8] > array[j+1] [11:8]
Go to Next step
Yes
temp [11:8] = array[j][11:8];
array[j][11:8] = array[j + 1][11:8];
array[j + 1][11:8] = temp [11:8];
Is array[j] [7:4] > array[j+1] [7:4]
No
Go to Next step
Yes
temp [7:4] = array[j][7:4];
array[j][7:4] = array[j + 1] [7:4];
array[j + 1][7:4] = temp [7:4];
Is array[j][3:0] > array[j+1][3:0]
temp [3:0] = array[j] [3:0];
array[j][3:0] = array[j + 1][3:0];
array[j + 1][3:0] = temp [3:0]
always @clk edge
out1 <= array[1];
out2 <= array[2];
out3 <= array[3];
out4 <= array[4];
out5 <= array[5]
END

Fig 5.4 Radix Sort Flow

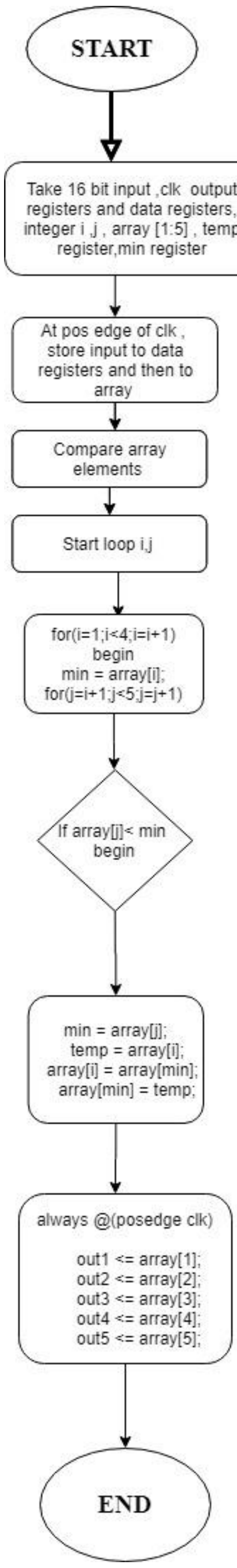

START
Take 16 bit input ,clk output registers and data registers, integer i ,j , array [1:5] , temp register,min register
At pos edge of clk , store input to data registers and then to array
Compare array elements
Start loop i,j
for(i=1;i<4;i=i+1) begin min = array[i]; for(j=i+1;j<5;j=j+1)
If array[j]< min begin
min = array[j]; temp = array[i]; array[i] = array[min]; array[min] = temp;
always @(posedge clk) out1 <= array[1]; out2 <= array[2]; out3 <= array[3]; out4 <= array[4]; out5 <= array[5];
END

Fig 5.5 Selection Sort Flowchart

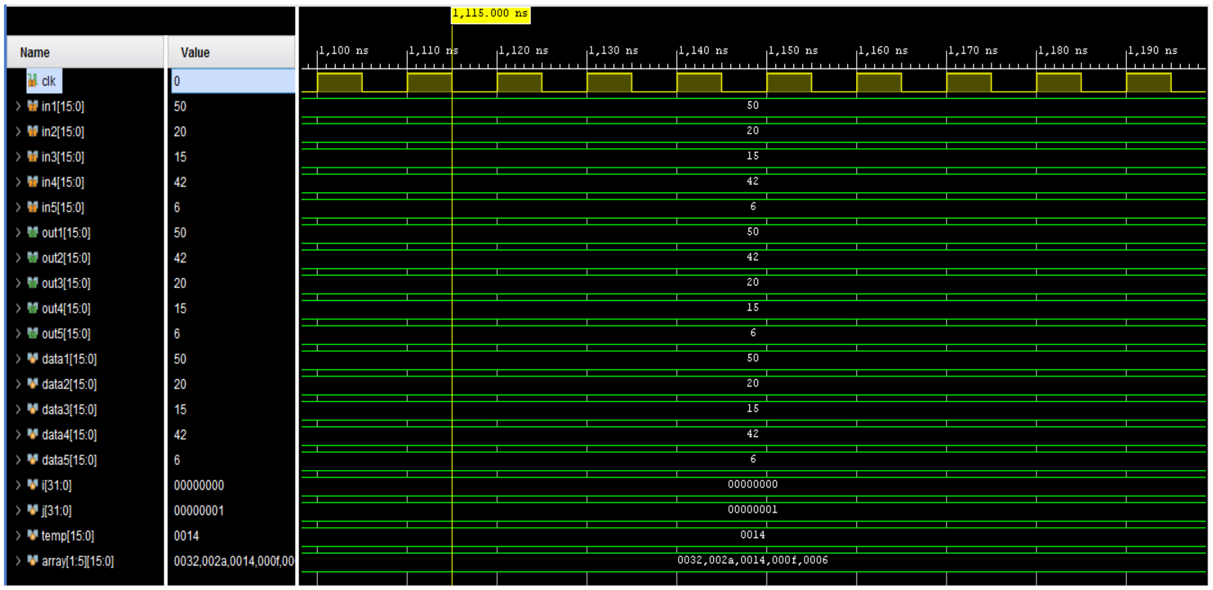
CHAPTER 6

RESULTS

The sorting algorithms have been simulated and synthesized on the VIVADO 2019.1 software and have been implemented on ZYNQ 7000 FPGA board. The simulation waveforms observed have been shown below. We have taken the following numbers as input :

In1 = 50	In2 = 20
In3 = 15	In4 = 42
In5 = 06	

6.1 BUBBLE SORT



Name	Value
clk	0
in1[15:0]	50
in2[15:0]	20
in3[15:0]	15
in4[15:0]	42
in5[15:0]	6
out1[15:0]	50
out2[15:0]	42
out3[15:0]	20
out4[15:0]	15
out5[15:0]	6
data1[15:0]	50
data2[15:0]	20
data3[15:0]	15
data4[15:0]	42
data5[15:0]	6
i[31:0]	00000000
j[31:0]	00000001
temp[15:0]	0014
array[1:5][15:0]	0032,002a,0014,000f,00

Fig 6.1 Bubble Sort Simulation Waveform

6.2 MERGE SORT

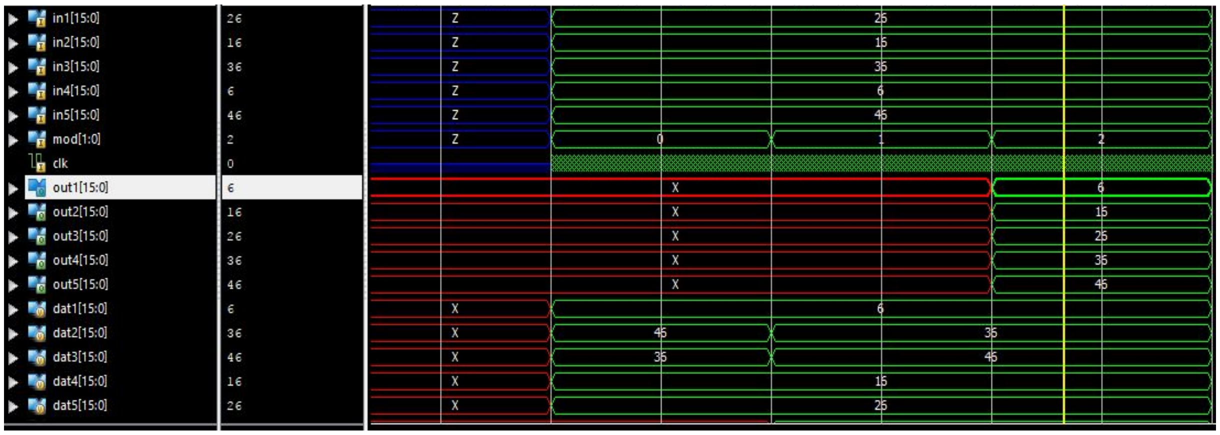


Fig 6.2 Merge Sort Simulation Waveform

6.3 INSERTION SORT

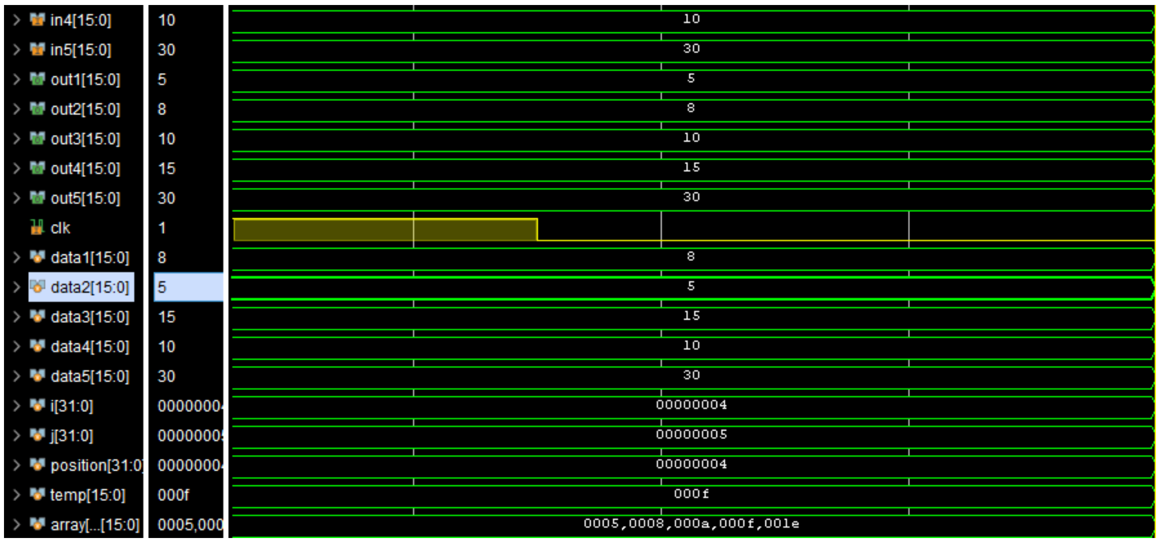
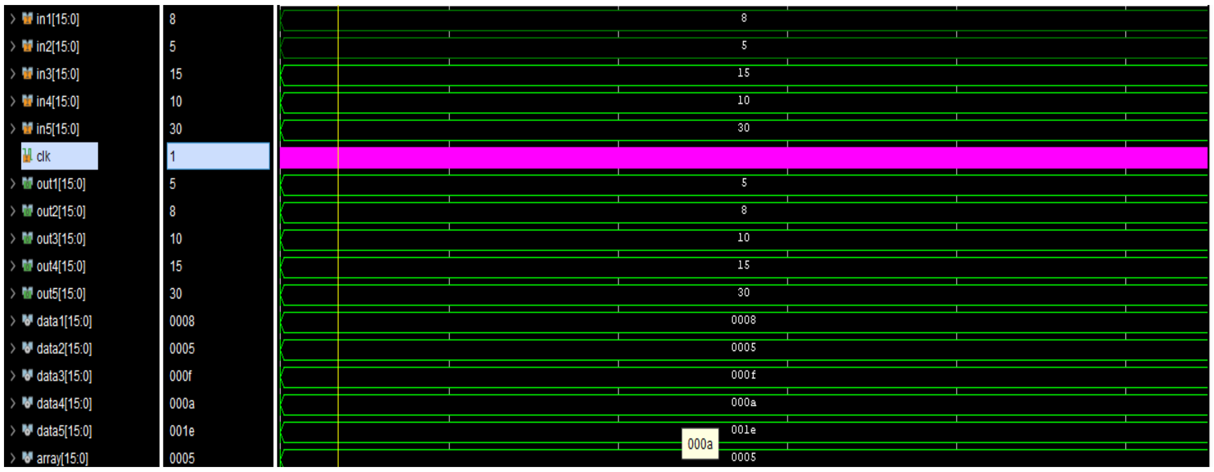


Fig 6.3 Insertion Sort Simulation Waveform

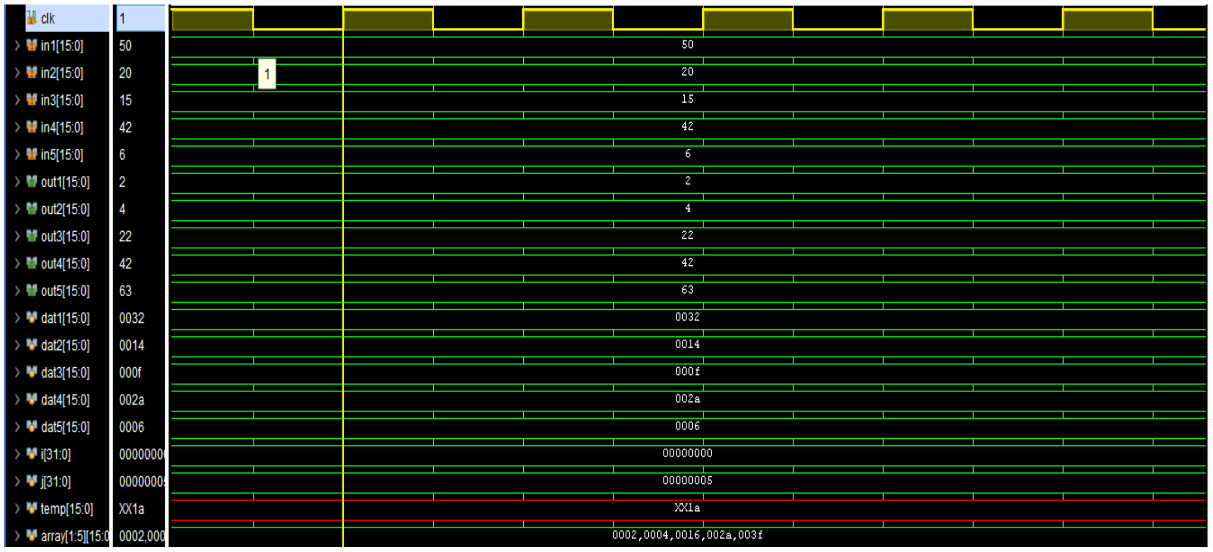
6.4 SELECTION SORT



```
in1[15:0]    8      8
in2[15:0]    5      5
in3[15:0]    15     15
in4[15:0]    10     10
in5[15:0]    30     30
clk          1
out1[15:0]   5      5
out2[15:0]   8      8
out3[15:0]   10     10
out4[15:0]   15     15
out5[15:0]   30     30
data1[15:0]  0008   0008
data2[15:0]  0005   0005
data3[15:0]  000f   000f
data4[15:0]  000a   000a
data5[15:0]  001e   001e
array[15:0]  0005   0005
```

Fig 6.4 Selection Sort Simulation Waveform

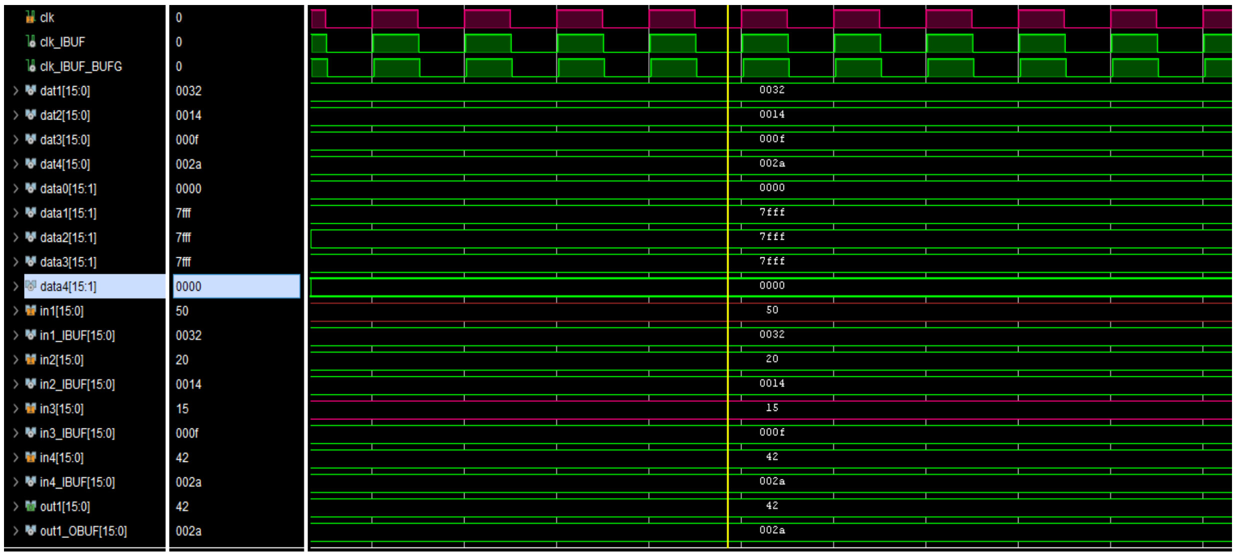
6.5 RADIX SORT



```
clk              1
in1[15:0]        50         50
in2[15:0]        20         20
in3[15:0]        15         15
in4[15:0]        42         42
in5[15:0]        6          6
out1[15:0]       2          2
out2[15:0]       4          4
out3[15:0]       22         22
out4[15:0]       42         42
out5[15:0]       63         63
dat1[15:0]       0032       0032
dat2[15:0]       0014       0014
dat3[15:0]       000f       000f
dat4[15:0]       002a       002a
dat5[15:0]       0006       0006
i[31:0]          0000000    00000000
j[31:0]          0000000    00000005
temp[15:0]       XX1a       XX1a
array[1:5][15:0  0002,000   0002,0004,0016,002a,003f
```

Fig 6.5 Radix Sort Simulation Waveform

6.6 COUNT SORT



clk	0
clk_IBUF	0
clk_IBUF_BUFG	0
dat1[15:0]	0032
dat2[15:0]	0014
dat3[15:0]	000f
dat4[15:0]	002a
data0[15:1]	0000
data1[15:1]	7fff
data2[15:1]	7fff
data3[15:1]	7fff
data4[15:1]	0000
in1[15:0]	50
in1_IBUF[15:0]	0032
in2[15:0]	20
in2_IBUF[15:0]	0014
in3[15:0]	15
in3_IBUF[15:0]	000f
in4[15:0]	42
in4_IBUF[15:0]	002a
out1[15:0]	42
out1_OBUF[15:0]	002a

Fig 6.6 Count Sort Simulation Waveform

6.7 PERFORMANCE ANALYSIS OF ALGORITHMS IN TERMS OF AREA , POWER AND TIMING

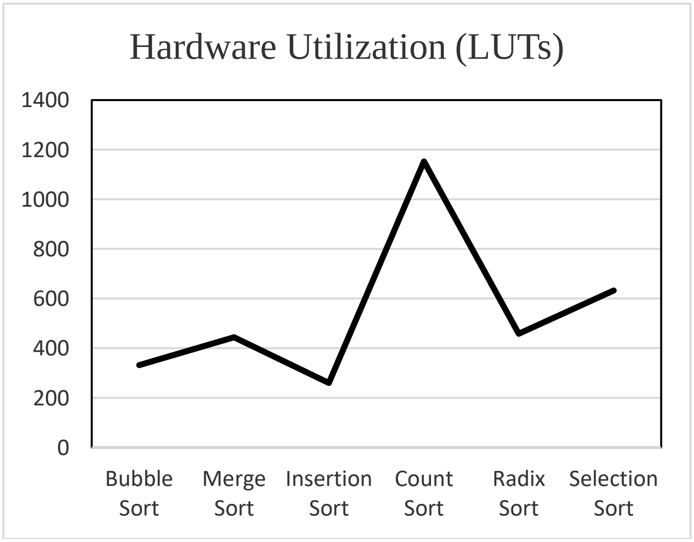


Fig 6.7 Hardware Utilization of Sorting Algorithms

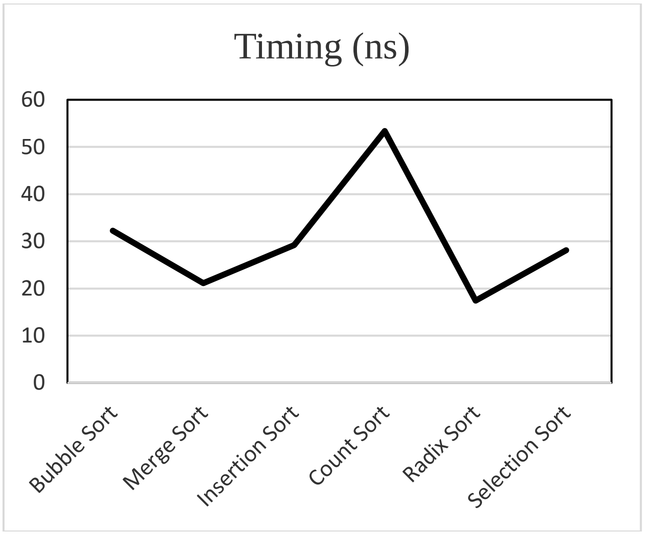


Fig 6.8 Representing delay in execution of sorting algorithms

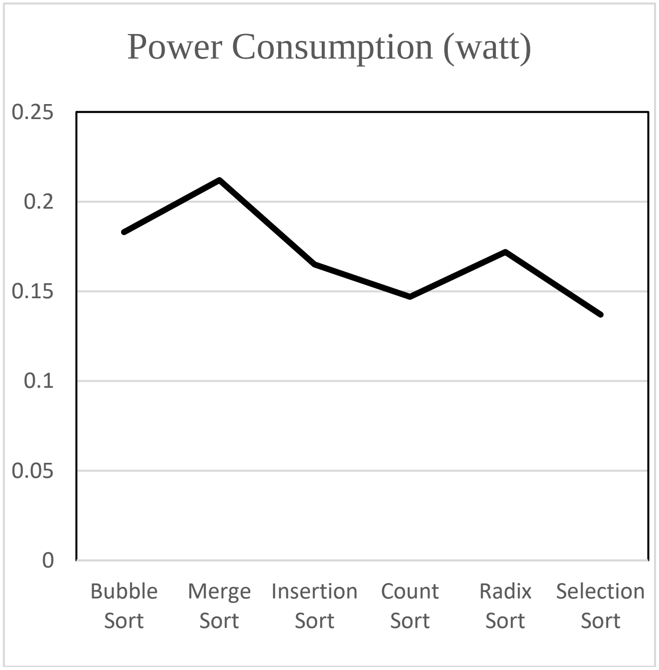


Fig 6.9 Power consumption

From the above three graphs it is evident that In terms of area, insertion sort scores the best and in terms of timing Radix Sort is the best choice while selection sort is the most power efficient algorithm. So while deciding the algorithm there is a tradeoff between area, power & timing.

CHAPTER 7

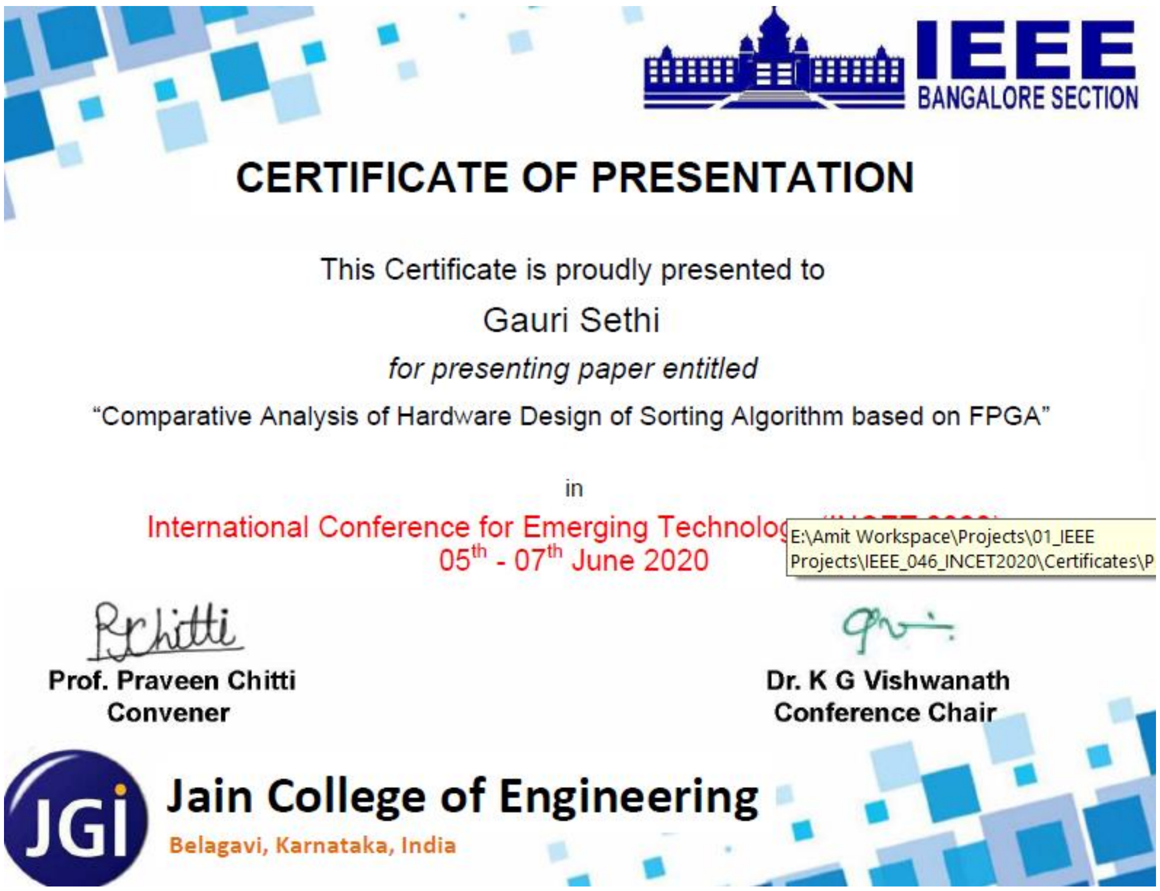
CONCLUSION & FUTURE SCOPE

All the algorithms have been implemented in VIVADO 2019.1 , written in Verilog HDL language . We have arrived at the conclusion from table 7.1 that while Insertion sort algorithms has the least hardware utilization or LUT consumed , Bubble sort being the simplest algorithm also has less amount of hardware utilization. Radix sort gives us the result in the least time and is considered to be the fastest while merge sort also does computation at a comparable rate only. Count sort on the other hand, being dependent on the value of the largest element on the array, hence is the slowest of all. Also Merge Sort is an overall good algorithm which can be preferred for varied amount of applications. I will be implementing an application of sorting algorithm like an algorithm of Machine Learning for proving its credibility in hardware oriented examples

Table 7.1 Analysis of Sorting Algorithms

Sorting Algorithm	LUT	Flip Flops	Total Delay (ns)	Net Delay (ns)	Power (Watt)
Bubble Sort	332	160	32.213	25.579	0.183
Merge Sort	444	160	21.107	18.003	0.212
Insertion Sort	260	160	29.209	23.435	0.165
Count Sort	1153	144	53.374	43.020	0.147
Radix Sort	458	160	17.417	15.031	0.172
Selection Sort	633	160	28.090	22.601	0.137

APPENDIX



IEEE
BANGALORE SECTION

CERTIFICATE OF PRESENTATION

This Certificate is proudly presented to

Gauri Sethi

for presenting paper entitled

"Comparative Analysis of Hardware Design of Sorting Algorithm based on FPGA"

in

International Conference for Emerging Technolog
05th - 07th June 2020

E:\Amit Workspace\Projects\01_IEEE
Projects\IEEE_046_INCET2020\Certificates\P

Prof. Praveen Chitti
Convener

Dr. K G Vishwanath
Conference Chair

Jain College of Engineering
Belagavi, Karnataka, India

My paper Id was : 539

REFERENCES

1. S. Jacobs and C. P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350

2. V. Magesh, S. Megavarnan, A. Pragadish and S. Saravanan ,“FPGA Implementation of Sorting Algorithm”, International Journal for Technological Research in Engineering, vol. 5, 2018

3. K. Sujatha, P. V. N. Rao, A. A. Rao, V. G. Sastry, V. Praneeta and R. K. Bharat, “ Multicore parallel processing concepts for effective sorting and searching” , International Conference Signal Processing Communication Engineering System, pp. 162–166, 2015

4. A. Srivastava, R. Chen, V. K. Prasanna, and C. Chelmis, “A hybrid design for high performance large-scale sorting on FPGA”, International Conference on Reconfigurable Computing FPGA’s, pp.1–6, 2015

5. W. Song, D. Koch, M. Luján and J. Garside, “Parallel hardware merge sorter”, Annual International Symposium Field-Program Custom Computing Machines (FCCM), pp. 95–102, 2016

6. S. H. Lin, P. Y. Chen and Y. N. Lin, “Hardware design of low-power high-throughput sorting unit” , Transaction Computing, vol. 66, pp. 1383–1395, 2017

7. S. Mashimo, T. V. Chu and K. Kise, “High-performance hardware merge sorter”, International Symp. Field-Program. Custom Computation, pp. 1–8, 2017.

8. B. Englert, M. He, X. Wu and Q. U. Zheng , “Optimal sorting algorithms for a simplified 2D array with reconfigurable pipelined bus system” ,IEEE Trans. Parallel Distribution System , vol. 21, pp. 303– 312, 2010

9. www.en.wikipedia.org/sorting

10. Kazim , “A comparative study of well known sorting algorithms”, International Journal of Advanced Research in Computer Science, vol. 8, 2017 www.researchgate.net/sortingalgorithm

11. www.stackoverflow.com

12. S. W. Al-Haj Baddar and B. A. Mahafzah, “Bitonic sort on a chained cubic tree interconnection network,” J. Parallel Distrib. Comput., vol. 74, no. 1, pp. 1744–1761, Jan. 2014

13. www.studytonight.com/data-structures/introduction-to-sorting

14. Pandey, K. Kumar, R .K. Bunkar, and K. K. Raghuvanshi. "A Comparative Study of Different Types of comparison Based Sorting Algorithms in Data Structure." International Journal of Advanced Research in Computer Science and Software Engineering Volume 4, Issue 2, February 2014

15. Lipu, A. Rahman, R. Amin, N. I. Mondal and A. I. Mamun, “Exploiting parallelism for faster implementation of Bubble sort algorithm using FPGA”, International Conference on Electrical, Computer & Telecommunication Engineering (ICECTE), pp. 1-4, 2016

16. Bunse, Christian, H. Höpfner, S. R. Choudhury, and E. Mansour, "Choosing the" Best Sorting Algorithm for Optimal Energy Consumption", In ICSOFT (2), pp. 199-206. 2009

17. Edjlal, Ramin, A Edjlal, and T. Moradi, "A sort implementation comparing with bubble sort and selection sort", 3rd International Conference on Computer Research and Development, vol. 4, pp. 380-381. IEEE, 2011

18. I. Ali , H. N. Lashari, “Performance comparison between merge and quick sort algorithm in data structures”, International Journal of Advanced Computer Science and Applications, January 2018

19. Y. Yang, P. Yu and Y.Gan, Y, “Experimental study on the five sort algorithms”, Second International Conference on Mechanic Automation and Control Engineering, pp. 1314-1317, 2011

20. Kocher and N. Aggarwal, “Analysis and Review of Sorting Algorithms”, International Journal of Scientific Engineering and Research (IJSER) ,vol. 2 ,2014

21. F. A. Alquaied, A. I. Almudaifer, M. A. AlShaya, “A Novel High-Speed Parallel Sorting Algorithm Based on FPGA”, Saudi International Electronics, Communications & Photonics Conference (SIECPC) , 2013

22. Norollah, Amin, D. Derafshi, H. Beitollahi, and M. Fazeli. "RTHS: A Low-Cost High-Performance Real-Time Hardware Sorter, Using a Multidimensional Sorting Algorithm." IEEE Transactions on Very Large Scale Integration (VLSI) Systems , May (2019).

23. N. Satish, M. Harris , M. Garland, “Designing Efficient Sorting Algorithms for Manycore GPUs”, IEEE International Symposium on Parallel & Distributed Processing , 2009

24. Y. B. Jmaa, R. B. Atitallah, D. Duvivier and M.B. Jemaa, “A comparative study of sorting algorithm with FPGA Acceleration by High Level Synthesis”, IEEE conference on computation systems, vol.23, pp. 213-230. ISSN 1405-5546, 2019

25. www.geeksforgeeks.com/sorting

26. www.interviewcake.com/concept/java/counting-sort

27. D. Koch , J. Torresen , “FPGA Sort: A High Performance Sorting Architecture Exploiting Run-time Reconfiguration on FPGAs for Large Problem Sorting”, In Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP), page 299

28. www.researchgate.net/sortingalgorithm

29. Xilinx Datasheet: www.xilinx.com/zynq7000-Pkg-Pinout

30. www.aldec.com/en/company/blog/144--introduction-to-ZYNQ-architecture

31. www.store.digilentic.com