

SYSTEM ON FABRICS UTILISING DISTRIBUTED COMPUTING

by

Partheepan Kandaswamy

A Doctoral Thesis submitted in partial fulfilment of the
requirements for the award of Doctor of Philosophy of
Loughborough University

June 2018

© *by Partheepan Kandaswamy*

To my parents
Shanthi and Kandaswamy

ABSTRACT

The main vision of wearable computing is to make electronic systems an important part of everyday clothing in the future which will serve as intelligent personal assistants. Wearable devices have the potential to be wearable computers and not mere input/output devices for the human body. The present thesis focuses on introducing a new wearable computing paradigm, where the processing elements are closely coupled with the sensors that are distributed using Instruction Systolic Array (ISA) architecture.

The thesis describes a novel, multiple sensor, multiple processor system architecture prototype based on the Instruction Systolic Array paradigm for distributed computing on fabrics. The thesis introduces new programming model to implement the distributed computer on fabrics. The implementation of the concept has been validated using parallel algorithms.

A real-time shape sensing and reconstruction application has been implemented on this architecture and has demonstrated a physical design for a wearable system based on the ISA concept constructed from off-the-shelf microcontrollers and sensors. Results demonstrate that the real time application executes on the prototype ISA implementation thus confirming the viability of the proposed architecture for fabric-resident computing devices.

ACKNOWLEDGEMENTS

I would like to thank my supervisors Dr. James Flint and Dr. Vassilios Chouliaras for their continuous support, guidance, exceptional advice and shared knowledge throughout my PhD. I would also thank Dr. David Mulvaney for his shared knowledge during my PhD.

I would like to acknowledge Mr. Peter Godfrey from Wolfson School Electronic workshop for his support in building the prototype board.

I would like to thank Mr. Ben Clark for his shared knowledge and support throughout my research and also for his patience in reading my manuscript.

Finally, I would like to thank my parents for their encouragement, support and funding me throughout my University education.

CONTENTS

LIST OF ABBREVIATIONS AND SYMBOLS	viii
LIST OF FIGURES	x
LIST OF TABLES	xii
CHAPTER 1: INTRODUCTION	1
1.1 Area of Research	2
1.1.1 Distributed Computing	2
1.1.2 Distributed Sensor Networks	3
1.1.3 Wearable Electronics	4
1.1.4 Smart Fabrics	6
1.2 Research Aim	9
1.3 Objectives	9
1.4 Novel contribution of the thesis	10
1.5 Thesis Outline	10
References	12
CHAPTER 2: A NOVEL PARALLEL DISTRIBUTED ARCHITECTURE.....	14
2.1 Introduction to Multiple sensors, Multiple Processor Systems.....	14
2.1.1 Comparison between the concepts.....	17
2.2 Classifications of Parallel Computer Architectures	19
2.2.1 Flynn’s Taxonomy	19
2.2.2 Duncan’s classification	21
2.2.3 VLSI processor arrays	23
2.2.4 Conclusion	24
2.3 Systolic Array.....	25
2.3.1 Features of systolic arrays.....	26
2.3.2 Types of systolic array structures	28
2.4 The Instruction Systolic Array	31
2.4.1 Principles of ISA.....	31

2.4.2	ISA Architecture	33
2.4.3	Programming and Execution of ISA.....	35
2.4.4	Applications of ISA	36
2.5	Adaptation to ISA	37
2.6	Systola 1024	37
2.7	Conclusion.....	38
	References.....	39
CHAPTER 3: IMPLEMENTATION OF INSTRUCTION SYSTOLIC ARRAY FOR SMART FABRICS		41
3.1	A novel architecture for on-fabric parallel processing.....	41
3.2	Implementation of novel architecture	44
3.2.1	Candidates for bus systems.....	44
3.2.2	Serial bus protocols.....	46
3.3	Details of the Inter-Integrated Circuit (I ² C) Bus.....	48
3.3.1	Bus Signals	50
3.4	Prototype Design.....	52
3.5	Selection of Microcontroller for the Processing Element	55
3.6	Power and programming interface for the array	57
3.7	Conclusion.....	60
	References.....	61
CHAPTER 4: PROGRAMMING AND VALIDATION OF THE INSTRUCTION SYSTOLIC ARRAY		62
4.1	Programming the Instruction Systolic Array	62
4.2	Merge Algorithm Validation.....	65
4.2.1	Algorithm.....	65
4.2.2	Program.....	66
4.2.3	Numerical example	67
4.2.4	Result from the processor array	71
4.3	Matrix Multiplication Validation	72
4.3.1	Algorithm.....	72
4.3.2	Program.....	76
4.3.3	Numerical example	77
4.3.4	Result from the processor array	81

4.4	Conclusion.....	81
	References.....	83
CHAPTER 5: SHAPE RECONSTRUCTION USING INSTRUCTION SYSTOLIC ARRAY		84
5.1	Introduction.....	84
5.2	Background.....	85
5.2.1	Shape Reconstruction algorithm.....	85
5.2.2	Shape Reconstruction from sensor orientation data.....	87
5.3	Experimental Setup.....	92
5.4	Programming the shape reconstruction algorithm using Instruction systolic array	93
5.5	Experimental Results.....	101
5.6	Conclusion.....	105
	References.....	106
CHAPTER 6: CONCLUSION AND FUTURE WORK.....		107
6.1	Contribution of this thesis.....	107
6.2	Suggestions for future research.....	108
6.2.1	Computational performance.....	108
6.2.2	Scalability.....	108
6.2.3	Programming techniques.....	108
6.2.4	Designing.....	109
6.2.5	Applications.....	109
6.3	Summary.....	109
	References.....	111
LIST OF PUBLICATIONS.....		A
APPENDIX.....		1

LIST OF ABBREVIATIONS AND SYMBOLS

ABBREVIATIONS

Abbreviation	Expansion
2D	Two Dimension
3D	Three Dimension
ACK	Acknowledgement
ASIC	Application Specific Integrated Circuit
C	Control Unit
CAN	Controller Area Network
Cm	Centimeter
DIP	Digital Image Processing
I	Instruction
I ² C	Inter Integrated Circuit
ISA	Instruction Systolic Array
MEMS	Micro Electro Mechanical Systems
MIMD	Multiple Instruction Multiple Data
MISD	Multiple Instruction Single Data
Ms	Milliseconds
P	Processing Element
PCB	Printed Circuit Board
R/W	Read/Write
S	Sensor
SCL	Serial Clock Line
SDA	Serial Data Line
SIMD	Single Instruction Multiple Data
SISD	Single Instruction Single Data
SPI	Serial Peripheral Interface
SQS	Surface Quality Scanner
UART	Universal Asynchronous Receiver/ Transmitter
USB	Universal Serial Bus
VLSI	Very Large Scale Integration

SYMBOLS

Symbol	Denotes
E_g	Earth gravity field vector
E_m	Earth magnetic field vector
M_e	Global Earth reference matrix
M_s	Sensor measurement matrix
R	Rotational matrix
R_p	Pull-up Resistance (Ω)
S_g	Sensor gravity field vector
S_m	Sensor magnetic field vector
V_{dd}	Supply Voltage (V)

LIST OF FIGURES

Figure 1.1: Partitioning of a wearable system from a technological point of view	5
Figure 1.2: Circuit incorporated in a textile with wire grid [1.15]	6
Figure 1.3: Smart fabric in healthcare [1.15].....	8
Figure 1.4: Muscle Activating Smart suit [1.15]	8
Figure 1.5: Networked Jacket [1.15]	9
Figure 2.1: Concept 1 showing Control unit C and Sensors $S_{n,m}$	15
Figure 2.2: Concept 2 where IN shows an interconnection such as bus.....	15
Figure 2.3: Concept 3 showing the inclusion of individual processing elements P	16
Figure 2.4: Concept 4 showing communication between neighbouring P's	17
Figure 2.5: Flynn's taxonomy of computer architectures: a) SISD, b) SIMD, c) MISD, and d) MIMD (C: Control unit, P: Processor, M: Memory, I N: Interconnection Network (Bus))	21
Figure 2.6: Duncan's taxonomy of parallel computer architectures.....	22
Figure 2.7: General systolic organization	27
Figure 2.8: Linear systolic array	28
Figure 2.9: Orthogonal systolic array	29
Figure 2.10: Hexagonal systolic array	29
Figure 2.11: Triangular systolic array.....	30
Figure 2.12: Execution of an ISA instruction	32
Figure 2.13: Instruction cycle	33
Figure 2.14: Execution of an ISA diagonal (I - Instruction, S - Selector bit, + - Execution).....	34
Figure 2.15: Execution of ISA program	36
Figure 2.16: Systola 1024 from [2.21].....	37
Figure 3.1: System concept.....	42
Figure 3.2: General concept of a sensor system with integrated processing elements for human body applications	43

Figure 3.3: Different methods for transfer of information.....	45
Figure 3.4: Typical I2C bus	49
Figure 3.5: Basic Mechanism in I2C from NXP Semiconductors adapted from [3.7]...	51
Figure 3.6: Processor array showing grid arrangement	53
Figure 3.7: Detail of I2C bus connections	54
Figure 3.8: Microchip PIC16F1829	55
Figure 3.9: 32-bit ARM Cortex-M0+ LPC824 microcontroller mounted on NXP LPC824-MAX board	55
Figure 3.10: Processor Array with peripherals	56
Figure 3.11: I2C connection between two microcontrollers with sensor	57
Figure 3.12: Prototype board	57
Figure 3.13: Programming board (switching circuit)	58
Figure 3.14: Schematic for the switching circuit	59
Figure 3.15: Application for programming the microcontrollers	60
Figure 4.1: Working of the ISA program	64
Figure 4.2: ISA program for merge algorithm.....	66
Figure 4.3: Performance analysis for P(2,1) and P(3,3)	72
Figure 4.4: ISA Program for Matrix Multiplication	76
Figure 4.5: Performance analysis for P(2,1) and P(3,3)	81
Figure 5.1: Surface segment structure. Each segment consists of center C and four direction vectors N , E , S and W [5.1]	88
Figure 5.2: Structure of control point connections. $C[i; j]$ - reference point. (a) Single reference row is obtained, then all other points are calculated with column method. (b) Single reference column is obtained, then all other points are calculated with row method adapted from [5.1].....	89
Figure 5.3: Step wise implementation of Shape Reconstruction Application	91
Figure 5.4: LSM303DLHC mounted on Adafruit board	92
Figure 5.5: Sensors embedded with fabric.....	93
Figure 5.6: ISA firmware for shape reconstruction application	95
Figure 5.7: Fabric wrapped on a cylindrical object	101
Figure 5.8: Reconstructed shape of the object	102
Figure 5.9: Fabric placed on the object.....	102

Figure 5.10: : Reconstructed shape of the object.....	103
Figure 5.11: Fabric placed on the object.....	103
Figure 5.12: Reconstructed shape of the object.....	104
Figure 5.13: Performance analysis for P(2,1) and P(3,3)	105

LIST OF TABLES

Table 2.1: Comparison between concepts	17
Table 3.1: Difference between serial and parallel communication	45
Table 3.2: Comparison of different bus system	46
Table 4.1: Instruction symbol definition.....	67

CHAPTER 1: INTRODUCTION

IN today's technological era, wearable electronics has become a crucial part of day to day activities. There has been a lot of development in the field of wearable electronics due to continuous quest of innovation by industrial and academic researchers. In earlier days, communication, electronics, and computing devices used were mainly non-portable because of their large size and complexity. Next introduced were smaller and lighter portable devices along with integration of some additional functions. Due to continuous improvements, now we have multi-purpose micro devices which can be embedded into wearables and are better in terms of many criteria such as communication, weight, energy management, durability, comfort and size [1.1].

In application-oriented research, the concept of wearable computing is a fast-growing area. Wearable technology can be used in various sectors like healthcare, military applications, gaming, sports, music and emergency services [1.2]. Wearable electronics can take the form of a discrete device such as a watch or arm band or it may be integrated into clothing opening an entirely new field of applications. As wearable devices increase in the level of complexity and become more integrated the opportunities to integrate more sophisticated functionality also increase [1.3]. NASA 3D printed space [1.4] fabric could potentially be used for large antennas and other deployable devices, because the material is foldable and its shape can change quickly. The fabrics could also eventually be used to shield a spacecraft from meteorites, for astronaut spacesuits, or for capturing objects on the surface of another planet. Currently, this development is in the early stages but it is easy to see how electronics may need to be incorporated.

The remarkable progress in miniaturization of microelectronics and progress in the invention of new materials have made it possible to integrate the functionality into clothing [1.5]. The main vision of wearable computing is to make electronic systems an

important part of everyday clothing in the future which will serve as intelligent personal assistants. Wearable devices have the potential to be wearable computers and not mere input/output devices for the human body. The present thesis focuses on introducing a new wearable computing paradigm which can improve the performance of a highly human-integrated computer.

As a result of remarkable innovations in embedded systems over a period of last thirty years, the value of microprocessors and communication technology have reduced significantly in terms of cost in real terms. Due to this, distributed computer systems have become a feasible substitute for uni-processor and centralised systems in various application areas of embedded systems.

The research challenge is to address the problems of low bandwidth sensors in wearable electronics. One of the solutions to high bandwidth sensor is the use of parallelism.

1.1 Area of Research

This thesis will focus on a distributed computing platform for wearable electronics. A brief introduction to the mainly used technologies in the current thesis is discussed in the following sub-sections.

1.1.1 Distributed Computing

A distributed computing system is a collection of processor-memory pairs connected by a communications subnet and logically integrated into varying degrees by a distributed operating system or distributed database system[1.6]. The communications subnet may be a widely geographically dispersed collection of communication processors or a local area network. The widespread use of distributed computer systems is due to the price-performance revolution in microelectronics the development of cost effective and efficient communication subnets (which is itself due to the merging of data communications and computer communications), the development of resource sharing software, and the increased user demands for communication, economical sharing of resources, and productivity[1.5]. A distributed computing system potentially provides significant advantages, including performance, reliability, resource sharing, and extensibility[1.6].

The study of distributed computing has grown to include a large range of applications[1.7],[1.8]. However, at the core of all the efforts to exploit the potential power of distributed computation are issues related to the management and allocation of system resources relative to the computational load of the system. One measure of the usefulness of a general-purpose distributed computing system is the system's ability to provide a level of performance corresponding with the degree of multiplicity of resources present in the system. This is particularly true of attempts to construct large general-purpose multiprocessors[1.7].

An interesting area for research which is increasingly getting noticed is decentralized processing [1.9]. As compared with centralised processing approach, the main advantage it provides is increased robustness. The entire system would never fail resulting from the malfunctioning of processors or sensors or other components. Nodes can be more flexible in distributed networks because nodes need not be reinitialized when nodes are introduced, moved and removed from the network for new topology [1.10].

There are also potentials of avoiding the fusion of a multitude of sensor data at once and adding more units would have potential of cost saving because mostly same design only needs to be duplicated. These are other benefits of processing the data in a distributed manner [1.11].

1.1.2 Distributed Sensor Networks

In detection applications, distribution of a large amount of simple sensing devices is increasingly getting more interest, mainly inspired from its perception in biological systems [1.11]. Focus on fusion of sensor signals instead of strong analysis algorithms, and a scheme to distribute sensors, results in new paradigm. Especially in wearable computing, where sensor data continuously changes, and clothing provides an ideal supporting structure for simple sensors [1.11].

The justification for using sensors in a wearable computing architecture ranges from use in intelligence augmentation to automating tasks depending on particular features of the environment. Regardless of whether these applications would be sought after by a large

community, one trend that can be observed is that sensors are gradually becoming part of mobile and wearable devices [1.11].

Wearable computers are no exception to this concept either, since large surfaces of clothing are an ideal supporting platform for a multitude of sensors, provided they are miniaturized so that they do not obstruct the wearer. This size constraint often means that the quality of the sensor itself is compromised as well, which leads to the concept of many simple sensors [1.11].

1.1.3 Wearable Electronics

Wearable Electronics is a new technological concept that integrates electronics with clothing and opens up a whole array of well designed, multi efficient and wearable electro textiles which can sense and monitor various functions of the body, can transfer data, can offer individual environment control and are able to provide communication facilities along with various other major applications[1.3]. The potential of wearable electronics is widespread when looking at so many innovatory advancements that are happening at an extraordinary rate in many fields of science and technology. These developments have the capability to change the world and they will very rapidly pervade into commercial products[1.12], [1.13]. Expert high-quality clothing will be available to make it possible to observe the important life signs of new born babies, clothing that can record the routine of an athlete's muscles and technique efficient clothing that can call even a rescue team for victims of accidents that occur due to bad weather conditions and there are limited options for help[1.13].

As described by I.Loacher [1.3], system-on-textile is the equipped clothing that combines electrical functions with apparel and at the same time maintains the wearing comfort. Another name for this is Smart Fabrics. The main aim is not to mix large electronic devices into clothing but rather small and committed electrical devices, for e.g. sensors along with their signal conditioning components taking the comfort of clothing into consideration. The sensors can be placed into positions where they can accomplish their sensing task in best possible way by integrating them directly into clothing such as accelerometers at joints. In contrast to this, chips that are having hundreds of pads and relatively high power dissipation, for e.g. high-speed microcontrollers are favourably placed into stiff enclosures such as belt buckles and

accessories. By keeping them there, circuits take advantage of the properties of Printed Circuit Board (PCB) technology like high-density wiring, multilayer and precisely controlled impedances. Fig 1.1 shows the partitioning of a wearable system from a technological point of view.

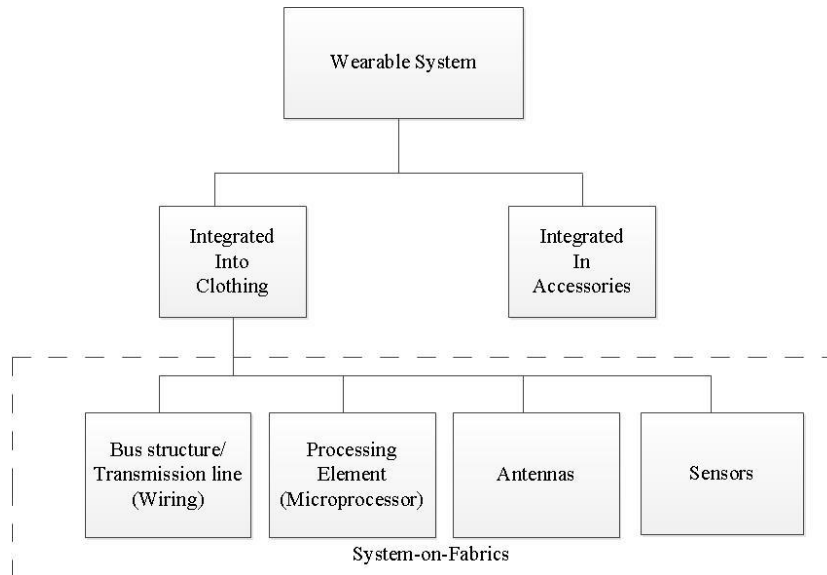


Figure 1.1: Partitioning of a wearable system from a technological point of view

The fabrics containing electronics as well as interconnections integrally woven into them are called as Electronic Textiles or e-textiles [1.14]. Electronic textiles provide physical flexibility and typical size which is hard to obtain from other existing electronic manufacturing techniques. As the electronic components and interconnections are woven into fabric, they are less visible and there are less chances of getting tangled in objects nearby. One important feature of E-textiles is their easy adaptation to any particular application requiring fast changes in computational and sensing requirements making them attractive for power management and context awareness. The vision of wearable computing is to make the electronic systems an important part of everyday clothing in the future. Although, these electronic devices should meet certain criteria to be wearable. The main feature of wearable systems will be their capability to identify the activity and the behavioural status of the person using them and the situations and environment around and then to further utilize this information to adapt the functionality and systems configuration [1.14].

There are different ways to produce electrically conductive fabrics. A technique is to incorporate conductive yarns directly into a textile structure, for instance, through weaving [1.14]. However, the incorporation of conductive yarns in a textile structure is complex and rarely a uniform process as the electrically conductive fabric has to be soft in touch or comfortable to wear rather than rigid and hard. Fig 1.2 shows an approach to incorporate circuits in a textile with wire grid [1.15].

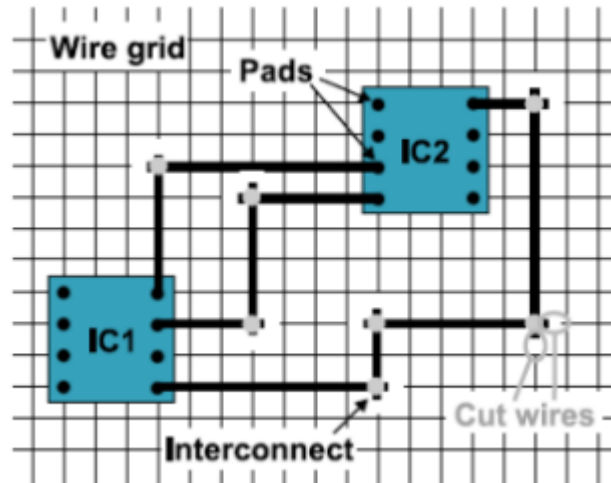


Figure 1.2: Circuit incorporated in a textile with wire grid [1.15]

1.1.4 Smart Fabrics

Electronics and Clothing were considered to be two different sectors of industries till now but now they are working together to produce some integrated and new innovative products[1.16], [1.17].

From Lymeris and Paradiso [1.18], since last 10-15 years, considerable advancements in the terms of data processing, miniaturization, functionality, seamless integration, comfort and communication have made Wearable Technology and integrated systems as well established fields. The textile industry is also increasingly interested by the potential for new value-added clothing products such as smart clothing and functionalised apparel and this is also driving the development of wearable systems.

In [1.17] Smart Fabrics are considered as the integrated systems into textiles and includes sensors, a power source, actuators and computing, forming a complete package for an interactive communication network. This type of smart systems can only be imagined by combining the innovative advances in fields like fibre and polymer research, microelectronics, embedded systems, advanced material processing,

telecommunication, signal processing and nanotechnologies. The most common platform to integrate smart materials in the form of fibres is textile. In textiles, by combining the chemical surfaces processes, the properties of the materials can be improved efficiently and also the structure of fabrics permits to exercise redundant sensor configurations.

One of the advantages of wearable application is that the smart fabrics provide a natural interface with the body considering comfort clothing with the help of precise and reproductive positioning of the sensors [1.18]. Bearing in mind comfort, the sensors are covered within the layers of fabric such as fibre optic or sometimes the fabric itself is used as a sensor or a distributed network of sensors.

Fabric computing includes designing a computing fabric which contains interconnected nodes but when observed from some distance, it seems like a fabric [1.19]. The two key components of fabrics are nodes and links. Nodes are processor(s), peripherals and memory whereas links can be described as the functional interconnection between nodes. Mainly it indicates towards a merged high-performance computing system that contains parallel processing functions, storage and networking linked with each other via high bandwidth interconnects.

Smart textiles or smart fabrics refer to clothing having integral electronics and interconnections woven into the fabrics itself [1.15]. This arrangement provides physical flexibility which is not attainable with other electronic manufacturing techniques. The electronic components and interconnections have low visibility and are less prone of getting tangled as they are embedded and woven with fabric [1.15]. The vision is to make smart textiles a part of day to day clothing. The main features of smart textiles include their ability to identify the activities around them as well as of their owner automatically and then to use the collected information to adjust functionality [1.15].

Medicine is a major area which has benefitted immensely in the applications developed from the combination of smart textiles and wearable computers in the form of Telemedicine. Fig 1.3 shows the overview of the use of smart fabrics and wearable computers in healthcare [1.15].

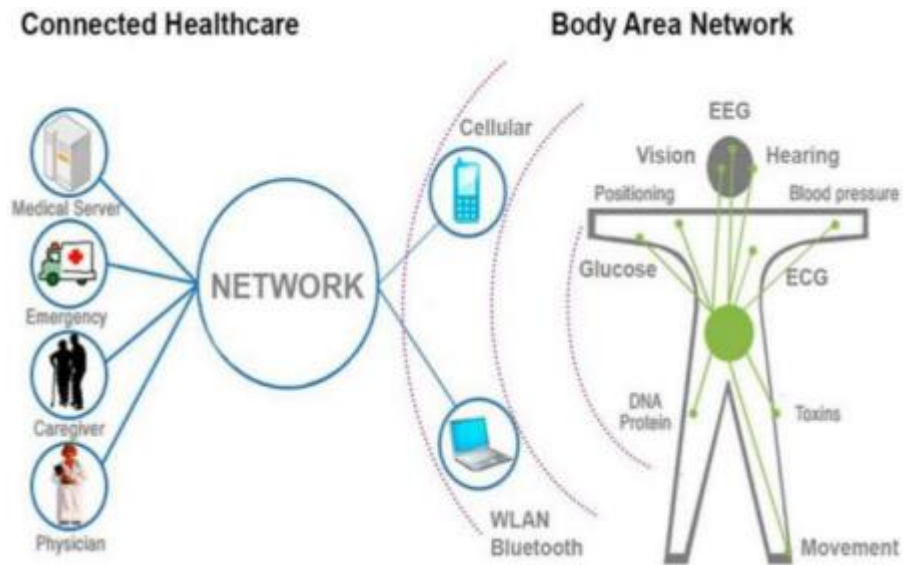


Figure 1.3: Smart fabric in healthcare [1.15]

In sports generally, important monitoring functions such as body temperature, heart rate, breathing, and other physiological parameters such as number of steps taken and total distance travelled can be achieved using smart devices embedded on sport clothing. Smart textiles in sports also help in protection against injury of athletes. Fig 1.4 shows an athlete wearing muscle activation smart suit [1.15].



Figure 1.4: Muscle Activating Smart suit [1.15]

The jacket shown in Fig 1.5 helps in the tracking of the location of the wearer using a GPS and project the map onto a flexible display screen on the sleeve of the jacket. It also displays the moods of the wearer via colour changes and signs [1.15].



Figure 1.5: Networked Jacket [1.15]

The Ohio State University researchers under the guidance of John Volakis have taken the next step toward the design of functional textiles clothes that gather, store, or transmit digital information [1.21]. This technology can result in lots of applications with further developments like sports equipment that monitors athletes performance, even a flexible fabric cap that senses activity in the brain, workout clothes that monitor your fitness level, a bandage that tells your doctor how well the tissue beneath it is healing, shirts that act as antennas for your smart phone or tablet [1.21].

1.2 Research Aim

The overall aim of this work is to advance the field of sensor networks by embedding parallel processing concepts. The application that the thesis will address is in human monitoring.

1.3 Objectives

The specific objectives of this thesis are:

- To propose a new sensor networking paradigm that exploits processor level parallelism and introduces the concept of on-fabric computation.
- To validate the method and produce parallel program that can be used on the sensor network array.
- To produce a physical demonstrator for a specific measurement scenario that has relevance to human monitoring.

1.4 Novel contribution of the thesis

- To propose a new concept for distributed on-fabric processing.
- To implement a parallel computing architecture optimised for fabric mounting.
- To apply the architecture to a physical demonstrator containing an array of computing nodes.
- To present a set of measurements obtained from a physical demonstrator.

1.5 Thesis Outline

Chapter 2: A Novel Parallel Distributed Architecture

The purpose of the chapter is to consider the concepts for attaching sensors to processing elements. This chapter will review the state of the art in parallel computer architectures and will identify a suitable architecture for a wearable computer system. The chapter also considers alternative architectures and how they interconnect with the physical local sensors.

Chapter 3: Implementation of Instruction Systolic Array for Smart Fabrics

An implementation of a prototype design of the novel architecture proposed in chapter 2 is given. The chapter also explains the challenges of implementing the design using commercial off-the-shelf components. The prototype has been designed using the concept of the Instruction Systolic Array. This chapter also discusses the bus systems and an off-the-shelf microcontroller that has been used to implement the prototyped concept.

Chapter 4: Programming and validation of Instruction Systolic Array

This chapter of the thesis describes the programming of the instructing systolic array and implementing the instruction systolic array on an array of off-the-shelf microcontrollers. To illustrate some of the basic definitions of the previous chapter, parallel algorithm examples are presented.

Chapter 5: Shape Reconstruction Application using Instruction Systolic Array

This chapter introduces a 2D mesh architecture prototype based on the Instruction systolic array paradigm for distributed computing on fabrics. A real-time shape sensing and reconstruction application executing on ISA architecture and demonstrates a physical design for a wearable system based on the ISA concept constructed from off-the-shelf microcontrollers and sensors.

Chapter 6: Conclusion

This chapter summarizes the contributions of the thesis and discusses the future work that can be conducted.

References

- [1.1] S. Lam Po Tang, "Recent developments in flexible wearable electronics for monitoring applications," *Transactions of the Institute of Measurement and Control*, vol. 29, no. 3-4, pp. 283-300, July 2016
- [1.2] *Wearable Devices* [Online]. [Accessed: 12 August 2017]. Available from: <http://www.wearabledevices.com/what-is-a-wearable-device>
- [1.3] J. McCann and D. Bryson, "Smart cloths and wearable technology", First Edition, pp.1, Woodhead Publishing Limited, 2009
- [1.4] NASA [Online]. [Accessed: 22 August 2017]. Available from: <https://www.nasa.gov/feature/jpl/space-fabric-links-fashion-and-engineering>
- [1.5] I. Locher, "Technologies for system-on-textile integration," Doctoral Thesis (PhD), Swiss Federal Institute of Technology, 2006
- [1.6] J. A. Stankovic, "A Perspective on Distributed Computer Systems," *IEEE Transactions on Computers*, vol. C-33, no. 12, pp. 1102-1115, December 1984
- [1.7] T. L. Casavant and J. G. Kuhl, "A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems," *IEEE Transactions on Software Engineering*, vol. 14, no. 2, pp. 141-154, February 1988
- [1.8] A. Burns and A. Wellings, "Concurrency of Ada", Second Edition, pp. 1, Cambridge University press, 1999
- [1.9] A. Cerpa and D. Estrin, "Ascent: Adaptive Self-Configuring Sensor Network Topologies", UCLA Computer Science Department Technical Report UCLA/CSD-TR-01-0009, May 2001
- [1.10] A. Lim, "Distributed Services for Information Dissemination in Self-Organizing Sensor Networks", Special Issue on Distributed Sensor Networks for Real-Time Systems with Adaptive Reconfiguration, *Journal of Franklin Institute*, Elsevier Science Publisher, Vol. 338, pp. 707-727, 2001
- [1.11] K. Van Laerhoven, A. Schmidt, H. Gellersen, "Multi-sensor context aware clothing", *Proc. 6th Int. Symp. Wearable Computers*, pp. 49-56, 2002
- [1.12] D. Trossen and D. Pavel, "Sensor networks, wearable computing, and healthcare applications," *Pervasive Computing*, IEEE, vol. 6, no. 2, pp. 58–61, 2007

-
- [1.13] B. Burchard, S. Jung, A. Ullsperger, and W. D. Hartmann, “Devices, software, their applications and requirements for wearable electronics,” ICCE, pp. 224–225, 2001
- [1.14] M. Stoppa and A. Chiolerio, "Wearable Electronics and Smart Textiles: A Critical Review", *Sensors* 2014, Vol. 14, 11957-11992, 2014
- [1.15] Yinka-Banjo Chika, and Salau Abiola Adekunle, “Smart Fabrics Wearable Technology”, *International Journal of Engineering Technologies and Management Research*, Vol. 4, pp. 78-98, 2017
- [1.16] S. I. Woolley, J. W. Cross, S. Ro, R. Foster, G. Reynolds, C. Baber, H. Bristow, and A. Schwirtz, “Forms of wearable computer,” in *IEE Eurowearable, IET*, pp. 47-52, 2003
- [1.17] K. V. Laerhoven, A. Schmidt, and H-W. Gellersen, “Multi-Sensor Context Aware Clothing,” in *ISWC, IEEE Computer Society*, pp.49-56, 2002
- [1.18] C. L. Cathey, J. D. Bakos, and D. A. Buell, “A reconfigurable distributed computing fabric exploiting multilevel parallelism,” in *FCCM, IEEE Computer Society*, pp. 121–130, 2006
- [1.19] A. Lymberis and R. Paradiso, “Smart Fabrics and Interactive Textile Enabling Wearable Personal Applications: R&D State of the Art and Future Challenges,” in *30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pp. 5270–5273, 2008
- [1.20] Y. Yu, C-L. Hui, T-M. Choi, and R. Au, “Intelligent Fabric Hand Prediction System with Fuzzy Neural Network,” *IEEE Transactions on Systems*, vol. 40, no. 6, pp. 619–629, 2010
- [1.21] Computers in your cloths a milestone for wearable electronics [Online]. [Accessed: 20 February 2018]. Available from: <https://news.osu.edu/news/2016/04/13/computers-in-your-clothes-a-milestone-for-wearable-electronics>

CHAPTER 2: A NOVEL PARALLEL DISTRIBUTED ARCHITECTURE

THE purpose of the chapter is to consider a series of possible concepts for attaching sensors to processing elements. This chapter will review the state of the art in parallel computer architectures and will identify a suitable architecture for a wearable computer system. The chapter also considers alternative architectures and how they interconnect with the physical local sensors.

2.1 Introduction to Multiple sensors, Multiple Processor Systems

The classification of parallel computer systems is usually based on their constituent hardware components. Once sensors are introduced into the parallel system there are a number of possible options for attaching them to the individual Processing elements.

Suppose that we have a rectangular sensor matrix of N by M sensors, each capturing analogue data with an upper-frequency f and we wish to continuously process data, producing a result. The application area is assumed to require processing of data from multiple sensors. An example of this is contained in a later chapter.

In Concept 1 shown in Fig 2.1 it can be seen that the single Control unit, C , which processes all the sensor data needs to process samples at a rate of $2.N.M.f$. That processing may be assisted by specialist hardware on particular processors but ultimately the control unit must handle this and perform its calculations at an appropriate speed.

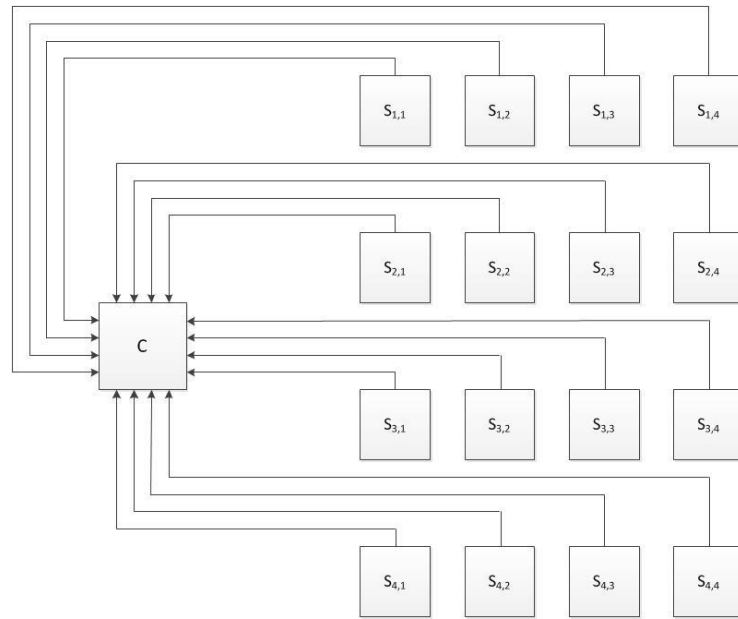


Figure 2.1: Concept 1 showing Control unit C and Sensors $S_{n,m}$

Concept 2 shown in Fig 2.2 is similar in terms of performance, however, although the wiring may well be more convenient it uses a shared bus system which may bring additional implementation cost and complexity. The interconnection IN shown in Fig 2.2 could be a bus communication used for the purpose to transfer data.

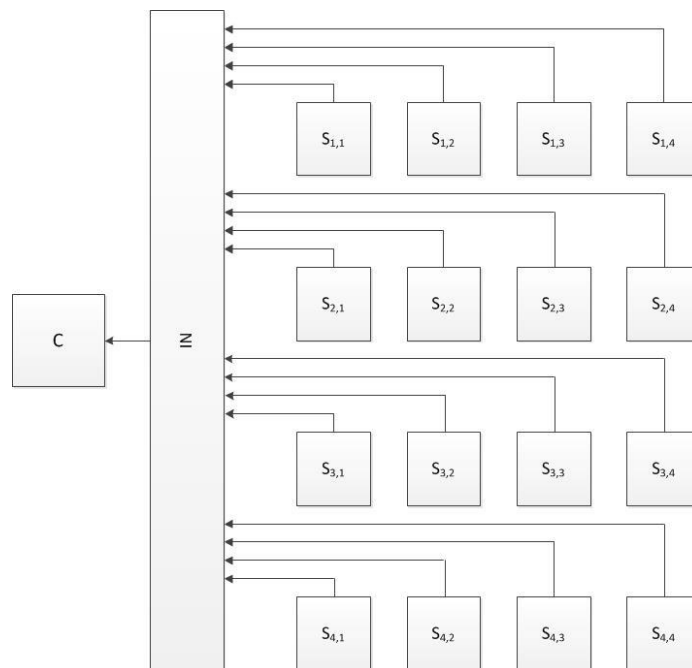


Figure 2.2: Concept 2 where IN shows an interconnection such as bus

Concept 3 shown in Fig 2.3 has a control unit and many processing elements. All the processing elements are connected to the control unit. The sensors are attached to the processing elements using their own individual buses. Here the processing elements are required to process samples at $2f$ samples/second and after preprocessing may be subsequently passed to the control unit. However, this offers a limited advantage if the purpose is to process data which involves fusing information from adjacent sensors.

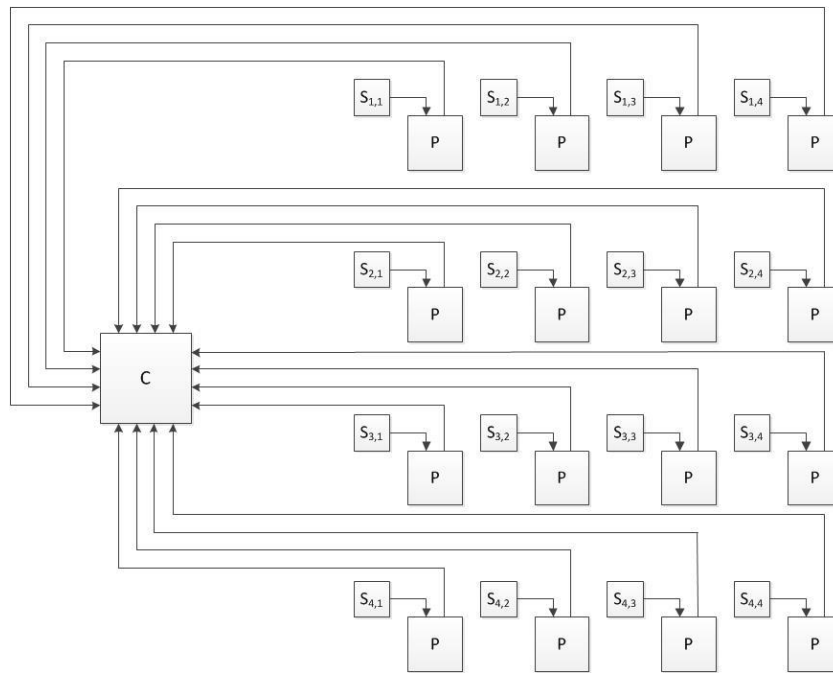


Figure 2.3: Concept 3 showing the inclusion of individual processing elements P

Concept 4 shown in Fig 2.4 has many processing elements. Each processing element is physically connected to the neighbouring processing elements. Every processing element is attached to its own sensors using an individual bus. The processing can be carried out locally at each processing element. Alternatively, the whole network of processing elements and sensors can be thought of as a form a distributed computer unit. This concept has inherent advantages as it means that co-located sensor data can be processed locally and independently by the distributed processors. Selected pre-processed data can also be communicated reducing bandwidth. It is worth emphasising that this is different to a conventional parallel concept because the processing elements are physically spaced out to coincide with their local sensors. Indeed, it may be possible

for the processing elements and sensors to be manufactured as one single integrated circuit. Each one of these integrated units would still be connected by physical bus wires which may be constructed using conductive thread or printed conductive wires on the fabric.

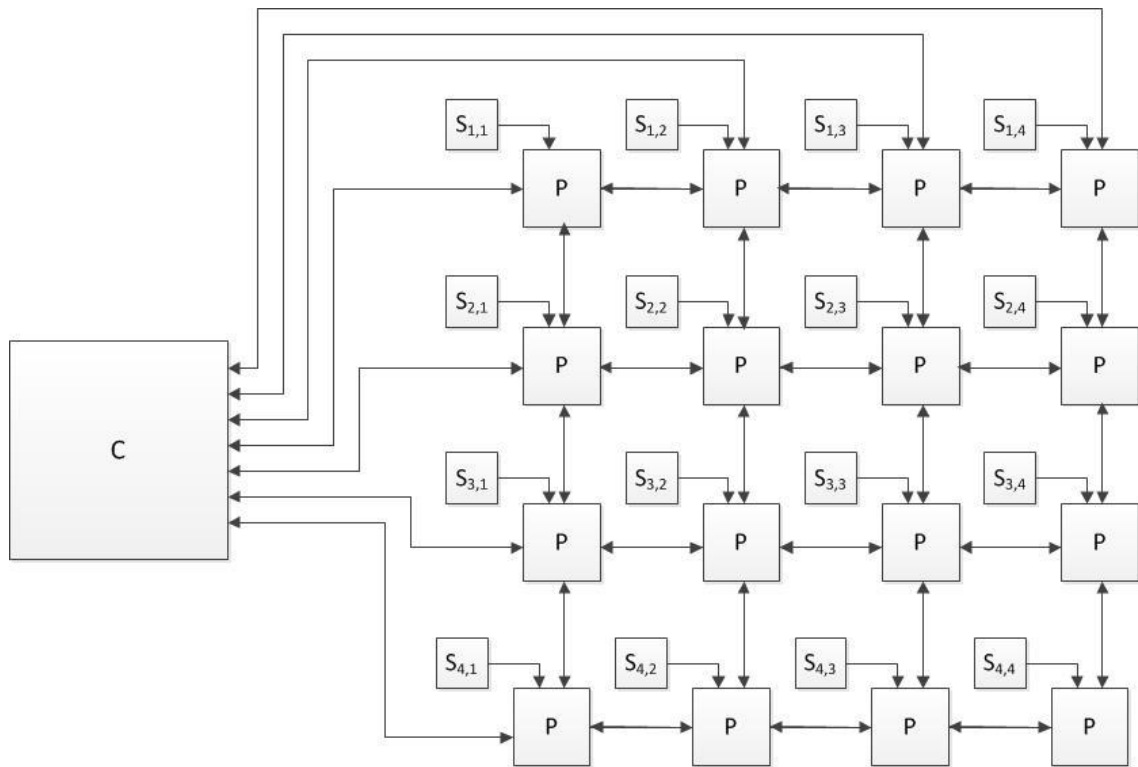


Figure 2.4: Concept 4 showing communication between neighbouring P's

2.1.1 Comparison between the concepts

The advantages and disadvantages of all four concepts are listed in the table below:

Table 2.1: Comparison between concepts

Concept	Advantages	Disadvantages
1	<ul style="list-style-type: none"> • Simple architecture. • Independent bus connection and no requirement for complex bus protocol. 	<ul style="list-style-type: none"> • Single control unit handling all the data. • Physical wiring for all sensors which returns to the single control

		unit.
2	<ul style="list-style-type: none"> • Fewer physical connections. 	<ul style="list-style-type: none"> • The bus can only be occupied by a single sensor at any one time. • Bus protocol required and sensor addressing must be implemented.
3	<ul style="list-style-type: none"> • Some pre-processing may be done at the processing elements. 	<ul style="list-style-type: none"> • More processing elements required. • Depending on the application, it may not be better than concept 1 or 2, where the application requires less sampling.
4	<ul style="list-style-type: none"> • May be able to exploit parallel processing paradigm to achieve improved performance. • Scalability may be achievable without reducing computing speed. • Buses are between adjacent processing elements and are not all routed back to the control unit. 	<ul style="list-style-type: none"> • Programmer's model is very complex. • Requires selection of suitable parallel processing concept and strategy for the control unit.

The Concept 4 looks promising as the architecture is distributed and has the potential to have the better performance compared to other concepts. It also has the benefit of

processing the data locally because it will resolve the high bandwidth problem and is not reported in the current literature. For example we can implement an FFT and then just export very small amount of data. This thesis takes the challenge of developing the concept and designing and implementing a wearable system based on this concept. The next section considers parallel architectures which may be suitable for such a system.

2.2 Classifications of Parallel Computer Architectures

Based on major methodologies that were created in the 1960s and 1970s, a wide range of computer architectures have been invented with huge development in VLSI technology over last 30 years. With expanding number of computer architectures, the classification of the architectures should be done efficiently. The classification should be done in such a way that it distinguishes the structures with considerable differences and meantime also discloses the similarities between noticeably divergent designs [2.1].

Various definitions have been proposed for a range of parallel architectures. Many authors have worked on the classification of computer architectures. The most widely accepted classifications among all are Flynn's taxonomy [2.2] which is based on instruction and data stream. One of the disadvantages of Flynn's classification is it does not clearly differentiate between various multiprocessor architectures. Some of these disadvantages from Flynn's classification have been resolved in Duncan's taxonomy [2.3]. These two taxonomies [2.2], [2.3] showing different points of view of parallel architectures have been briefly explained in the next sections.

2.2.1 Flynn's Taxonomy

Flynn's taxonomy, which is one of the earliest classification systems for parallel computers, was developed by Michael J. Flynn in 1966. This classification has been used as a tool in designing modern processors and their functionalities. Flynn mainly used two criteria for the classification of programs and computers, first being whether they were working using a single set or multiple set of instructions and second was whether or not those instructions were using a single set or multiple sets of data [2.1].

2.2.1.1 Flynn's classification

Based on the presence of either single or multiple streams of instructions and data, four groups according to Flynn's taxonomy are SISD, SIMD, MISD and MIMD. Flynn's classification is briefly described below:

- SISD (Single Instruction Single Data); which mainly describes serial computers.
- SIMD (Single Instruction Multiple Data); which works with multiple processors executing the same instruction simultaneously on different data.
- MISD (Multiple Instruction Single Data); which works with multiple processors executing different instructions to a single data stream. This is more uncommon architecture.
- MIMD (Multiple Instruction Multiple Data); which works with multiple processors simultaneously executing multiple instructions on multiple data.

These four categories along with their architectural differences are shown in Fig. 2.5. The major representatives of SISD category are single processor computers. The next one is SIMD category, which includes vector computers as well as array computers. It is also known as synchronous parallelism. MISD is an uncommon category which is even referred as non-existent by various authors. Bräunl [2.4] classified pipeline computers under this category. The last one is MIMD category which includes multi processor distributed computer systems. It is also known as asynchronous parallelism, which is opposite to SIMD.

Flynn's taxonomy provides useful information for characterising computer architectures. Many structures have been found that do not clearly show any of these characteristics and hence do not fit in any of these four groups. So, Flynn's classification became inadequate when it comes to the classification of many modern computers like pipelined processors, systolic arrays, etc. [2.3].

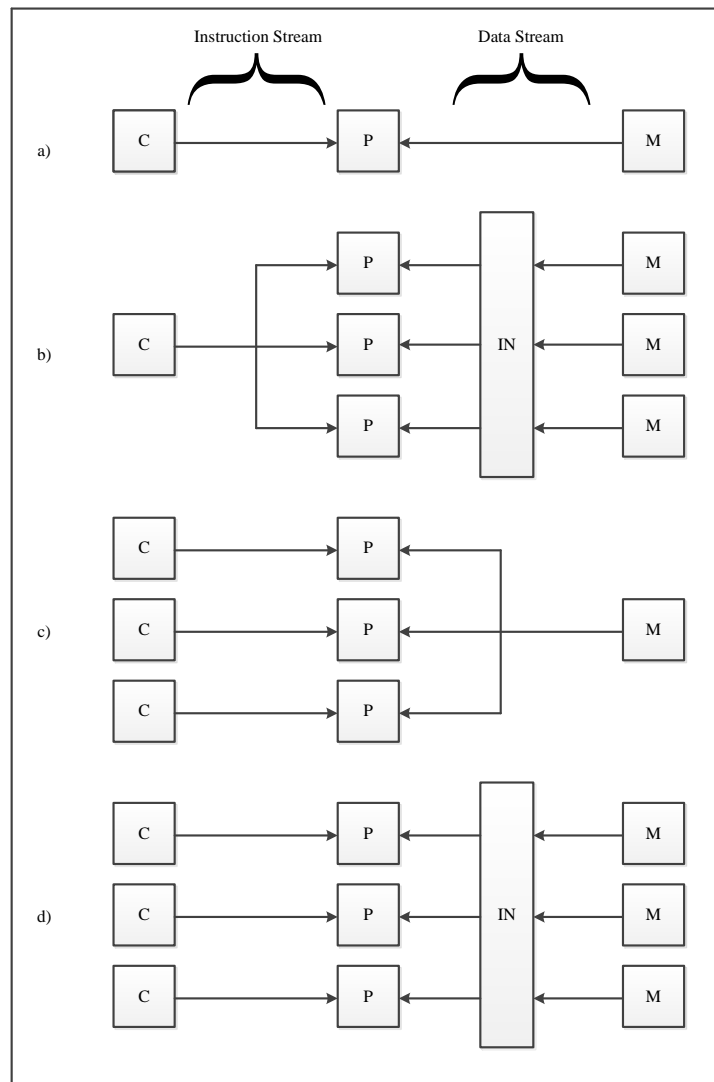


Figure 2.5: Flynn's taxonomy of computer architectures: a) SISD, b) SIMD, c) MISD, and d) MIMD (C: Control unit, P: Processor, M: Memory, I N: Interconnection Network (Bus))

2.2.2 Duncan's classification

The latest architecture innovations were positioned in a broader framework of parallel architectures by Duncan's taxonomy. According to Duncan, the classification should satisfy the following important points [2.3]:

- It should maintain the elements of Flynn's classification based on instruction and data streams;
- It should exclude the architectures which incorporate just a low-level parallel mechanism which has become a general feature of modern computers;

- It should include pipelined vector processors and other architectures which intuitively looks as parallel architectures but hard to properly classify under Flynn's taxonomy.

If the above conditions are satisfied, a parallel architecture can be described as a high level, the explicit framework used to develop parallel programming solutions with the help of multiple processors that work together through simultaneous execution to solve the problems. The processors can either be simple or complex.

The classification of processor structures according to Duncan's classification is shown in Fig. 2.6.

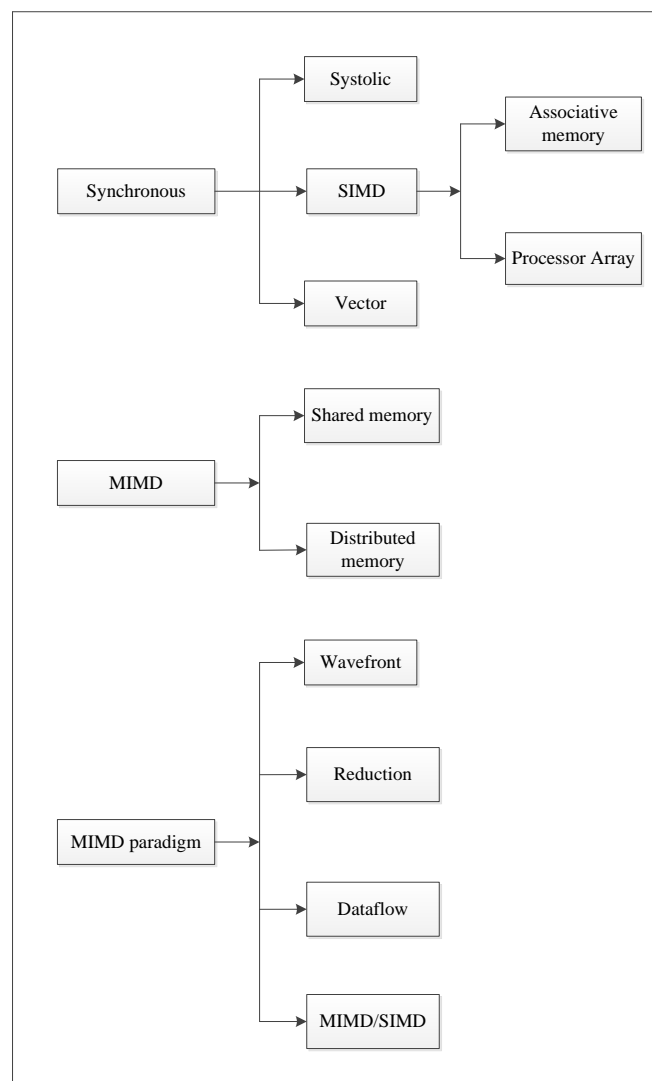


Figure 2.6: Duncan's taxonomy of parallel computer architectures

2.2.3 VLSI processor arrays

Most of the architectures are termed as Very-Large-Scale Integration (VLSI) processor arrays. The data is pipelined through the processors simultaneously with processing in systolic arrays and wavefront arrays. Wavefront arrays use data driven potential, whereas systolic arrays utilise local instructions synchronised globally. Both SIMD and MIMD utilise global data and control instead of using pipelined data. It permits broadcasting from a memory and a control unit. The main features of four computer structures are explained briefly in the segment below.

2.2.3.1 SIMD architectures

Normally, the SIMD architectures utilise a central control unit, multiple processors and an interconnection network, which establishes processor-to-processor or processor-to-memory communications. The central control unit broadcasts a single instruction to all processors. The processors, in turn, execute the instruction on local data. The main function of the interconnection network is to communicate the instruction results calculated at one processor to another processor to be used as operands in a subsequent instruction.

2.2.3.2 MIMD architectures

MIMD architectures use multiple processors which execute independent instruction stream utilising local data. These kinds of architectures are capable of supporting parallel solutions, in which processors are required to function in a largely autonomous manner. MIMD architectures are asynchronous computers that are mainly characterised by decentralised hardware control. The software processes executed on MIMD architectures are typically synchronised by either passing messages via an interconnection network or by accessing data stored in shared memory. High-level parallelism is supported by MIMD computers at sub program and task level.

2.2.3.3 Systolic architectures

Kung and Leisserson [2.8] were the first to introduce systolic architectures in 1978. Systolic arrays are typically defined as high-performance, special-purpose VLSI computer systems. They are appropriate for specific application requirements which

require a balance of intensive computations along with demanding input/output bandwidths. Systolic architectures also called as systolic arrays are organised as networks that contain a large number of identical, locally connected Elementary processing elements. Data in systolic arrays is pulsed from memory through processing elements before returning to memory in a rhythmic fashion. The system is synchronised using a global clock and explicit timing delays. For a diverse range of special purpose systems, modular processors united by regular and local interconnections act as basic building blocks. The performance requirements of special-purpose systems are handled using systolic arrays by achieving considerable parallel computations and by avoiding input/output and memory bandwidth restrictions.

2.2.3.4 Wavefront array architectures

Systolic data pipelining and asynchronous data flow execution paradigm, both are combined in wavefront array processors. Wavefront array and systolic architectures, both are designated by modular processors and regular, local interconnection networks. However, in wavefront array architectures, the global clock and explicit time delays used for synchronising systolic data pipelining are replaced with asynchronous handshaking to be used as the mechanism for coordinating inter-processor data movements. So, when a processor is finished doing its computations and wants to pass the data to its successor processor, it sends the data when successor signals that it is ready. An acknowledgement is sent by successor after receiving the data. The computational wavefronts pass smoothly through the array without intersecting using the handshaking mechanism because the processors of the array behave as a wave propagating mechanism. In this way, the correct timing of systolic architectures is replaced by correct sequencing of computations.

2.2.4 Conclusion

After evaluating all the available parallel architectures, the systolic architecture has been chosen as being suitable implementing Concept 4 chosen from the previous section. The systolic mode of parallel processing has gained a tremendous interest due to the elegant exploitation of data parallelism inherent in computationally demanding algorithms from different fields of research. In order to explain a little more about how this can be

applied to a smart fabric system, the fundamental theory behind the systolic arrays will be presented. Research into systolic arrays has been dormant for some years however there is no prior work using these arrays in the physically distributed wearable system. The application that has been chosen to be implemented was human body monitoring thus we need a distributed architecture to implement such an application. There appears to be some potential merit in using systolic array design to implement Concept 4 where a sensor is closely coupled with the processing element.

2.3 Systolic Array

The term systolic array in the computer science was introduced in 1978 by Kung et al. [2.8]. Conventionally, a systolic array is made up of a large number of similar processing elements interconnected in an array. The interconnections are local, which means each processing element can communicate only with a limited number of neighbouring processing elements. There are two types of systolic arrays, data systolic array and instruction systolic array.

In data systolic array, the data moves at a constant velocity passing from one processing element to the next processing element. Every processing element performs computations, in this way contributing to the overall processing that is required to be done by the array. Data systolic array is generally called as systolic array.

In contrast to the data systolic array, an instruction systolic array (ISA) is a grid-connected network of very simple computation units (processing elements), which is characterized by the instructions being pumped from a corner in a systolic manner.

Systolic arrays are synchronous systems. The exchange of data between directly communicating processing elements is synchronised using a global clock. The data can only be exchanged at the tick of the global clock. In between two consecutive clock ticks, each processing element performs computation on the data which it has received upon the last tick and then generates the data which is to be sent to neighbouring processing elements at the next clock tick. The processing element is also capable of holding data stored in the local memory of the processing element.

2.3.1 Features of systolic arrays

Different authors have given different definitions for systolic arrays. A well-known definition according to Kung and Leiserson [2.8] is:

“A systolic system is a network of processors which rhythmically compute and pass data through the system.”

A more reliable definition of systolic arrays is presented in terms of bullet points below. A systolic array can be defined as a computing system having the following characteristics [2.4]:

- **Network:** It is a computing network having a number of processing elements or cells with interconnections.
- **Rhythm:** The data is computed and passed throughout the network in a rhythmic and repetitive manner.
- **Regularity:** The interconnections between the processing elements are consistent and regular. The numbers of interconnections for processing elements does not depend on the size of the problem because the numbers of interconnections between the processing elements are almost the same for any size of array.
- **Synchrony:** The execution of instructions and the communication data is synchronised using a global clock.
- **Locality:** The interconnections are local, which means that only neighbouring processing elements can communicate directly with each other.
- **Modularity:** The network may contain one or more types of processing elements. The systolic array can typically be decomposed into different parts with one processor type, in case there is more than one type of processors.
- **Extensibility:** The computing network has the feature of being extended indefinitely.
- **Pipelineability:** All data is transferred using pipelining, which means that at least one delay element (register) is present between each two directly connected combinatorial processing elements.

- **Boundary:** Only processing elements in the network which are at the boundary can communicate with the outside world.

To summarise the characteristics discussed above, it can be seen that a large number of processing elements operate in parallel on different parts of the computational problem. Data enters into the systolic array through the boundary. Once the data enters into the systolic array, it can be used many times before it is output to the outside world. Typically, various data streams flow through the array at constant velocities while interacting with each other in the course of this movement. Meanwhile, processing elements execute one and the same function in a repeated manner. The systolic array does not transfer the intermediate results to the control unit. The control unit and the systolic array carry out the exchange of only the initial data and the final results [2.1].

A systolic array is a form of parallel computing method in which the processors are interconnected to each other in the form of a matrix and typically called as cells [2.9]. Each processing element has a special feature that it is capable of storing and computing data independently of other processing elements and eventually processing the data. It can share the information swiftly with its neighbouring processing elements. The major advantage of systolic arrays is that the data can flow in multiple directions. Fig 2.7 shows the general systolic array organisation. In systolic arrays, the input/output rate between the processing elements is generally very high, making them suitable for intensive parallel operations [2.10].

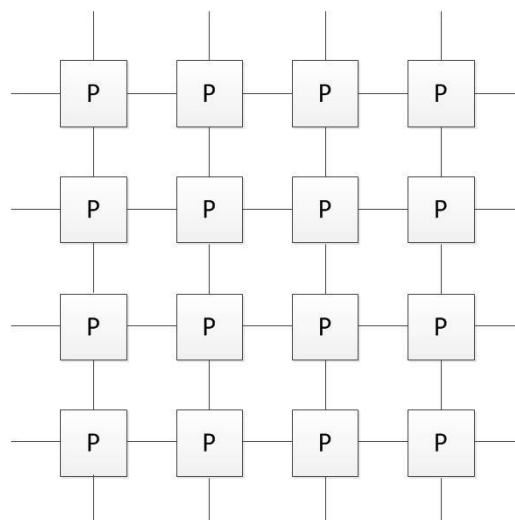


Figure 2.7: General systolic organization

2.3.2 Types of systolic array structures

This section of the chapter discusses the four different types of systolic arrays structures and their applications which are Linear systolic array, Orthogonal systolic array, Hexagonal systolic array and Triangular systolic array.

2.3.2.1 Linear systolic array

The processing elements are organised in one dimension in case of a linear systolic array as shown in Fig 2.8. The processing elements have interconnections only with their nearest neighbours. Linear systolic arrays distinguish themselves in terms of a number of data flows along with their relative velocities. One-dimensional convolution (FIR filtering) is one of the representatives of linear systolic arrays [2.1].

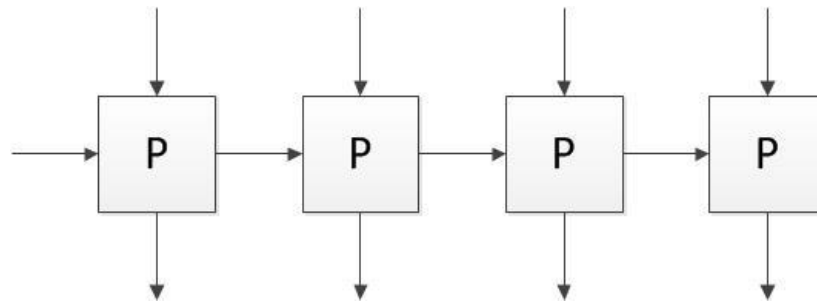


Figure 2.8: Linear systolic array

2.3.2.2 Orthogonal systolic array

The processing elements are organised in a two-dimensional grid in an orthogonal systolic array as shown in Fig 2.9. Each processing element, in this case, is interconnected to its nearest neighbours in all four directions to the north, east, south and west. The orthogonal systolic arrays differ relative to the number and direction of data flow as well as the number of delay elements organised in them. One of the possible mappings of the matrix multiplication algorithm is the most general representation of this array [2.1].

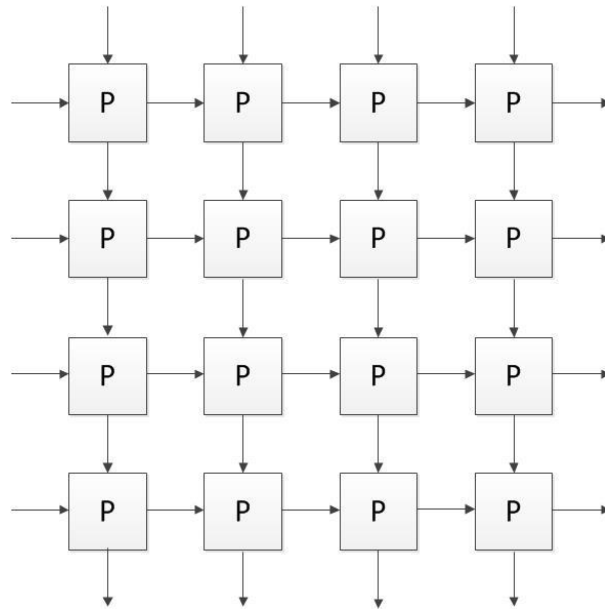


Figure 2.9: Orthogonal systolic array

2.3.2.3 Hexagonal systolic array

The processing elements are organised in a two-dimensional grid in a hexagonal systolic array as shown in Fig 2.10. The processing elements are connected with their nearest neighbours on six sides where inter-connections have a hexagonal symmetry [2.1].

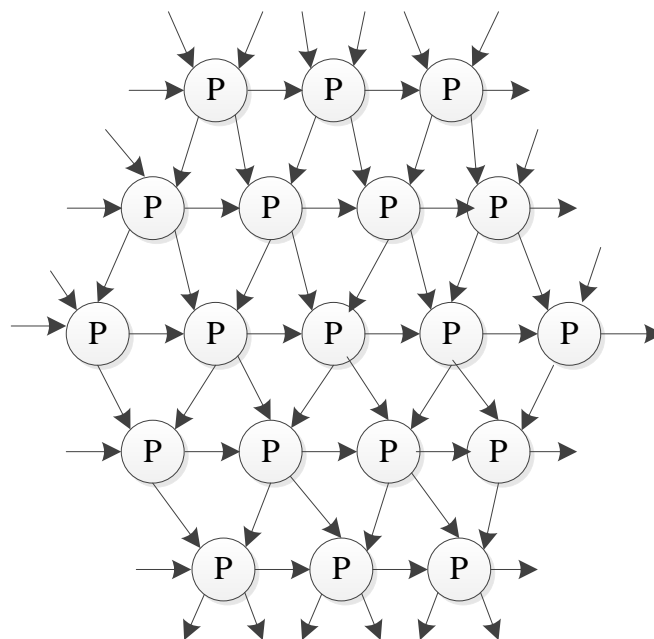


Figure 2.10: Hexagonal systolic array

2.3.2.4 Triangular systolic array

The processing elements are organised in a triangular form in a triangular systolic array as shown in Fig 2.11. It is a two-dimensional systolic array. Mostly, this form is used in different algorithms from linear algebra. Particularly, it is more important in Gaussian elimination and other decomposition algorithms [2.1].

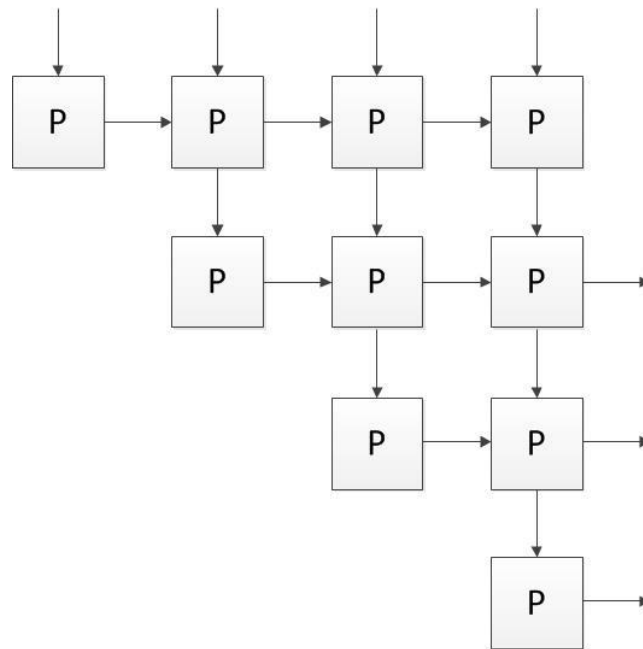


Figure 2.11: Triangular systolic array

Among various types of systolic array structures, the orthogonal systolic array is assumed as its structure fits body-worn fabrics the best. The orthogonal systolic array has been chosen as the best for wearable applications because of evenly distributed processing elements in the rows and columns which benefits in the diagonal flow of instructions along the array and the array could have a simpler instruction set. Also, the underlying parallel computer model is instruction systolic array, an architectural concept suited for implementing a system with high bandwidth and with architectural benefits for wearable.

2.4 The Instruction Systolic Array

Instruction Systolic Array (ISA) is broadly used in VLSI for execution purposes as an architectural concept [2.11], [2.12]. ISA can be viewed as more flexible and advanced from the properties below and are considered chiefly as special purpose architectures.

The important properties of ISA are:

- local communication for data and control flow,
- modularity and scalability
- local data handling
- mapping is logical

In ISA, rather than data, instructions are pumped in a systolic way through a processor array which makes it different from standard systolic arrays [2.11], [2.13]. This particular arrangement helps in executing different algorithms on the same processor array. Also, the instruction stream and the stream of selector bit both get combined. Due to this, subsets of processing elements can have a very flexible addressing. The fundamental model of a parallel computer can be seen as a mesh connected $n \times n$ -array identical processors. The processors are capable of executing instructions from a small instruction set. The processor array is synchronized by a global clock, and each instruction is supposed to take the same time for its execution.

2.4.1 Principles of ISA

The instructions for the ISA are inputted from the upper left corner of processor array as shown in Fig. 2.12, instruction flow in horizontal and vertical directions through the array step by step [2.12]. This process makes it sure that during each clock cycle, the same instruction is available for execution within every diagonal of the array.

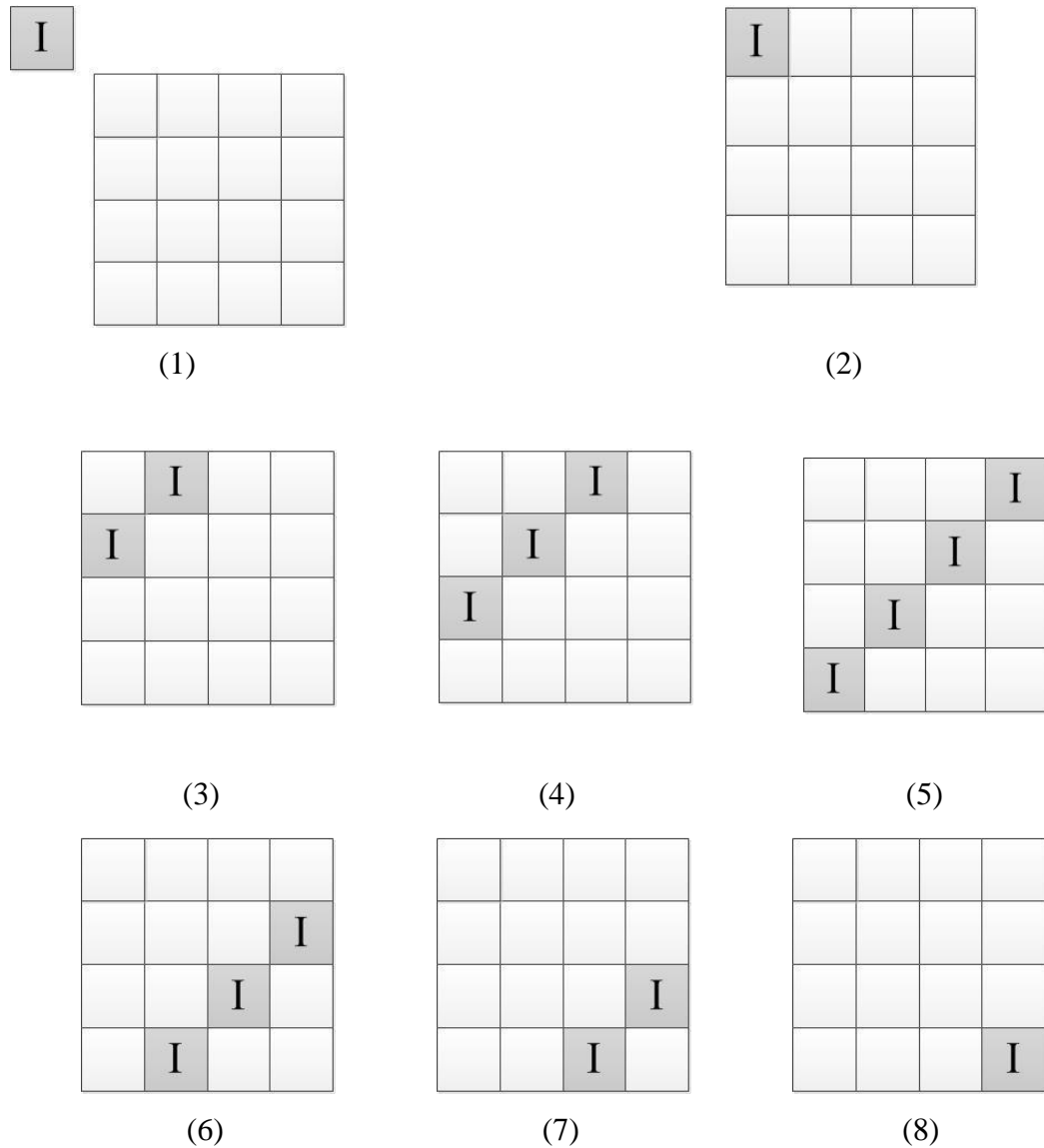


Figure 2.12: Execution of an ISA instruction

Each processor has some data registers that also includes a designated communication register C. Communication process between two processors, A and B take place in following way:

In [2.12] the concept of data transfer between the processors is explained as for example, a data item is to be sent from processor A to B, first A writes the data item into its own communication register. In the next instruction, B reads the contents from the communication register of A. Each processor is allowed only to write data to its own communication register, but it is allowed to read data from the communication registers of its four direct neighbouring processors. Two or more processors can read the data from same communication register at the same time. To avoid confusion between

read/write processes, it is arranged so that reading from a register is carried on during the first half of the execution of instruction and writing on a register is carried on during the second half as shown in Fig 2.13.

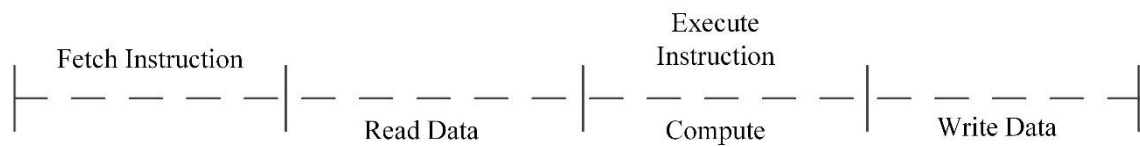


Figure 2.13: Instruction cycle

The main feature of ISA is that throughout the array, it provides a rhythmic flow of instructions [2.11]. The basic architecture of an ISA is a mesh-connected array of processing elements, and every processing element is capable of executing instructions from a fixed instruction set. The execution of a large variety of algorithms can take place on same ISA. In an ISA, along with the instruction stream, an orthogonal stream of control bits is also used. The execution step for any instruction in processing element takes place only when the selector bit at that processing element is 1. Due to the use of selector bits in execution, the array processor architecture tends to be very flexible. Instructions and selector bits are used for controlling processing elements.

The processors are provided with instructions and selector bits from outside the array. Instructions are input one by one from the upper left processor, and then they move in diagonal wave fronts throughout the array [2.14].

2.4.2 ISA Architecture

The flow of instructions is generally from top to bottom (north to south) of the array. On the other hand, the selector bit flows from left to right (west to east) of the array. To carry out the instructions at that particular processing element, the selector bits must be 1. Fig 2.14 shows the execution of ISA diagonal.

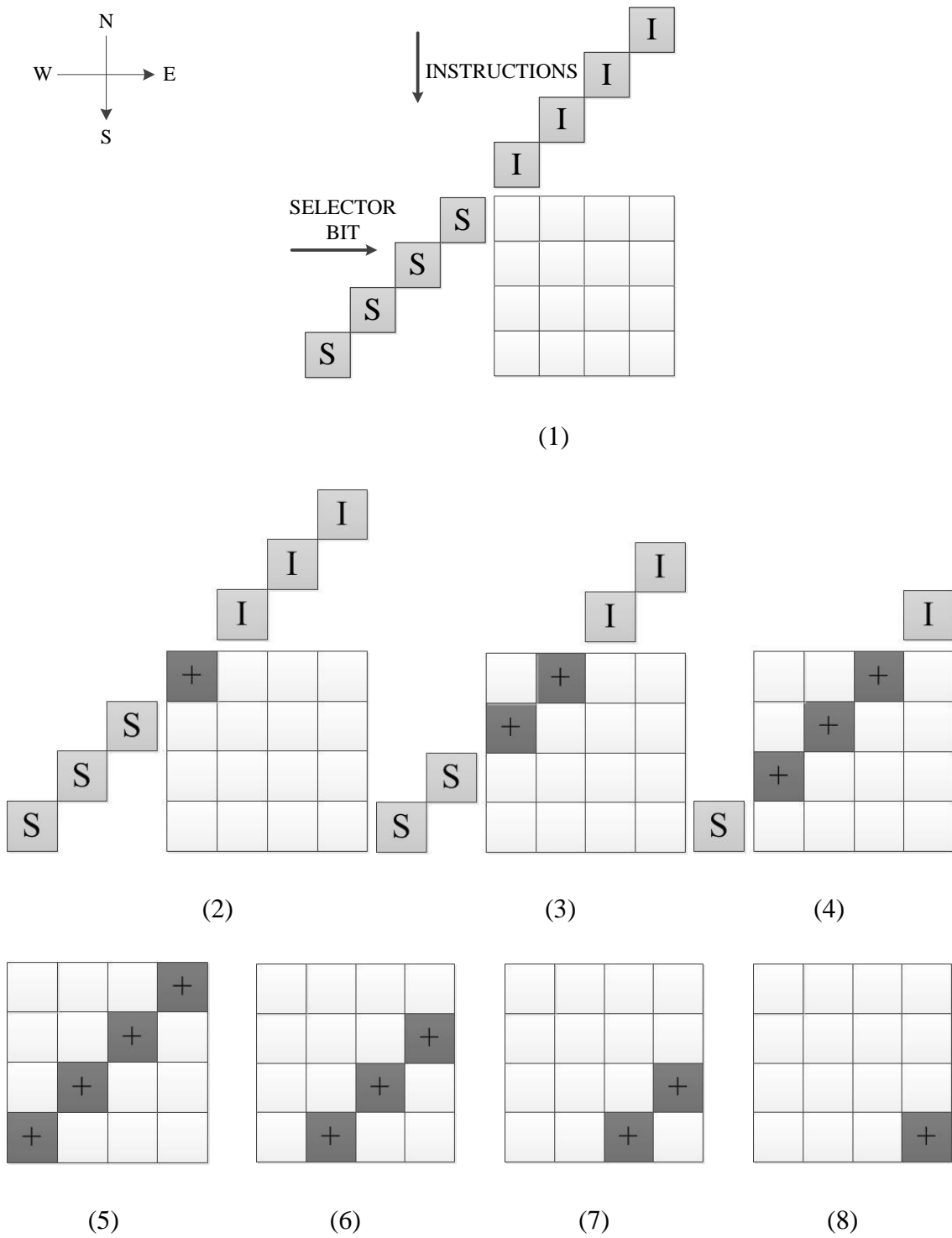


Figure 2.14: Execution of an ISA diagonal (I - Instruction, S - Selector bit, + - Execution).

The ISA can be thought as more of a pipelined SIMD array. It is still possible to perform broadcast and ring shift operations with a minimum number of instructions even though there are no global wires or wrap-around connections [2.14].

2.4.3 Programming and Execution of ISA

Laisa is a Pascal-like programming language used for ISA programs. It supports control structures like conditional statements and loops as well as procedures [2.9]. Basic machine code for the ISA is implemented in LAISA using brackets:

Elementary statements in Laisa are of the form

<instruction; selector>

Instructions can be register assignments of the form

<set source-register, destination-register>

or arithmetical or logical operations of the form

<instructioncode source-register1, source-register2, destination-register>

Registers can be any of the data registers or the communication register C, the communication registers of the western, northern, eastern or southern neighbour CW, CN, CE, CS, respectively [2.16].

Data is input or output to the processor array is finished via the open-ended processor links present at the boundary of the array [2.9]. The ISA is supposed to be embedded into an environment which is proficient enough to:

- supply ISA with instructions and selectors,
- supply ISA input data and to store its output data.

The key concept is that there should be a communication in between the processors in the form of an array with short interconnections and without the use of any global wires. By using the concept of pipelined execution of instructions in the processor, increases efficiency of the array [2.15].

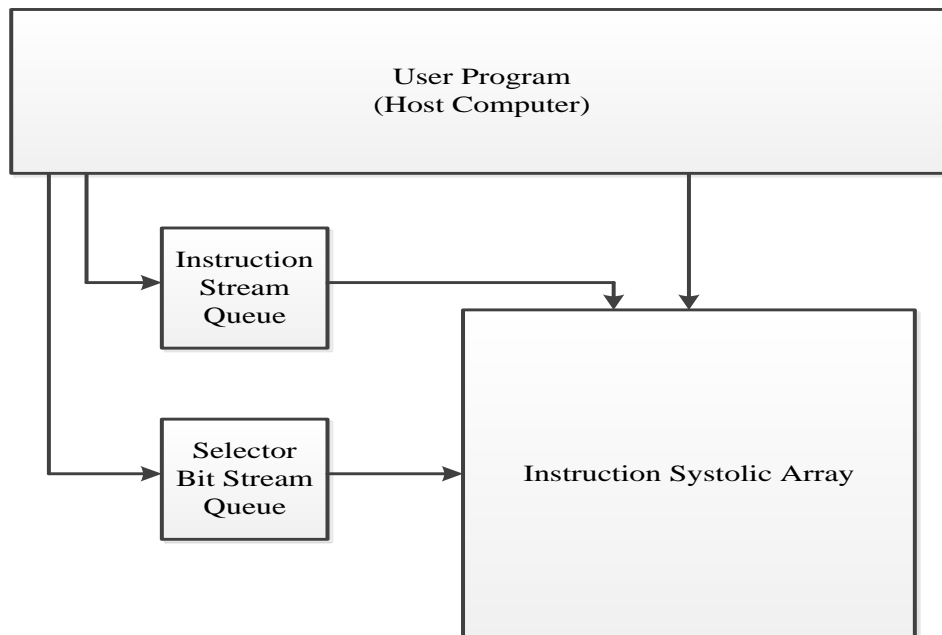


Figure 2.15: Execution of ISA program

The controller receives its instruction queue and selector bits which are loaded before the execution of an application. The ISA block consists of individual processing elements. The ISA program is loaded into each processing elements on the Instruction systolic array direct from the host computer. The ISA gets its instructions from the ISA program memory. It is also loaded before the execution of the desired application programs. The execution of the programs is started by the flow of instruction and selector bit stream, as indicated in Fig. 2.15.

2.4.4 Applications of ISA

The main applications of Instruction Systolic Array are as follows [2.16]:

- Solving problems regarding linear equations in Digital Image Processing (DIP)
- Computer Graphics
- Cryptography

To summarize, following properties sums up the advantages of ISA architecture [2.16]:

- Broad applicability
- Only local communications for control and data flow purposes
- Fast and parallel computations

- Scalability and modularity

2.5 Adaptation to ISA

Schmidt et al. [2.13] and Sim et al. [2.17] have adapted a different method from the conventional instruction systolic array. To improve the performance of their application they proposed modifying the way in which selector bits are sent from both top and left (north and west). The north will have both instruction and selector bit entering the array. In the present thesis, a similar approach has been taken into consideration. This has the advantages on the performance, simplification of instruction and data loading into the array. The details of this will be explained in chapter 4.

2.6 Systola 1024

The first commercial parallel computer based on the ISA architecture [2.18]-[2.20] is Systola 1024 which is shown in Fig. 2.16. The ISA has been integrated for standard personal computers on a low-cost add-on board. A strict co-processor concept has to be followed to operate using this board. By executing corresponding parallel programs on the Systola 1024, the sequential programs can be accelerated by replacing computationally intensive procedures.



Figure 2.16: Systola 1024 from [2.21]

One of the real time applications where ISA is used is in optical surface inspection of coated surfaces. Special measuring methods were needed for this application, which enables quick scanning of large surfaces and avoiding the direct contact to the surface at the same time. For such an application, optical methods combined with digital image processing provide a satisfactory solution. For applications mainly in the sector of machine vision and fast vision, systems provide the required computing power by utilising special image processing hardware or high-power workstations. The major

disadvantage of these systems is the involvement of large budget. The instant outcome is cutting the quality control out of economic reasons, which is the end quality control is generally carried out by human visual inspection.

A low-cost alternative to large budget solutions is developed by ISATEC and is termed as the Surface Quality Scanner (SQS 1024) [2.22]. The combination of a standard personal computer, the Systola 1024 board and low-cost video data acquisition boards, offers to provide a solution for quality control at a competitive price and performance ratio. The Systola 1024 board is used as hardware base for the technology.

2.7 Conclusion

The chapter has reviewed a range of parallel architectures which are known. The application in question has specific requirements which are somewhat unusual because there is a desire to colocate the sensors and processing elements for the purpose of reducing wiring complexity. A significant theoretical advantage of the ISA is that data is local to the processor and as such a common limitation of the ISA, namely transfer of data onto the array is circumvented. There are clearly other mechanisms for improving performance, however other architectures do not have this inherent advantage. The following chapters will make the assumption that this architecture will be used and consider the implementation, programming and performance of such a computer.

References

- [2.1] M. Zajc, "Systolic Parallel Processing," Algorithms and architectures of multimedia systems, Lecture Notes, University of Ljubljana [Online]. [Accessed: 15 October 2016]. Available from:<http://ldos.fe.uni-lj.si>
- [2.2] M. J. Flynn, "Very High Speed Computing Systems," Proc. of the IEEE, vol. 54, no. 12, pp. 1901-1909, December 1966
- [2.3] R. Duncan, "A Survey of Parallel Computer Architectures", IEEE Computer, pp. 5-16, February 1990
- [2.4] T. Bräunl, "Parallel Programming: an Introduction," Prentice Hall, 1993
- [2.5] N. Petkov, "Systolic Parallel Processing," North-Holland, 1993
- [2.6] S. Y. Kung, "VLSI Array Processors," Prentice Hall, 1988
- [2.7] E. V. Krishnamurthy, "Parallel Processing: Principles and Practise," Addison-Wesley, 1989
- [2.8] H. T. Kung, C. E. Leiserson "Systolic Arrays for (VLSI)," Technical Report CMU-CS- 79-103, Carnegie Mellon University, 1978
- [2.9] Hans Werner Lang "Instruction Systolic Array" [Online]. [Accessed: 15 October 2016]. Available from: <http://www.inf.fh-lensburg.de/lang/papers/isa/isa1.htm>
- [2.10] T. J. Fountain, "The Design of Highly-Parallel Image Processing Systems Using Nanoelectronic Devices," IEEE, pp. 210-219, 1997
- [2.11] M. Kunde, H.-W. Lang, M. Schimmler, H. Schmeck, and H. Schröder, "The instruction systolic array and its relation to other models of parallel computers," Elsevier Parallel Computing, vol. 7, no. 1, pp. 25–39, 1988
- [2.12] H. W. Lang, "The instruction systolic array - a parallel architecture for VLSI," Integration, vol. 4, pp. 65–74, 1986, doi:10.1016/0167-9260(86)90038-6
- [2.13] B. Schmidt, M. Schimmler and H. Schroder, "Morphological Hough Transform on the Instruction Systolic Array," In Practice, 1997
- [2.14] R. Hughey and D. P. Lopresti, "Architecture of a Programmable Systolic Array," pp. 41-49, 1988, doi:10.1109/ARRAYS.1988.18043
- [2.15] K. T. Johnson, A. R. Hurson, and B. Shirazi, "General-Purpose Systolic Arrays," Computer, vol. 26, no. 11, pp. 20–31, 1993
- [2.16] B. Schmidt, "Techniques for Algorithm the Instruction Design on Systolic Array," Doctoral Thesis (PhD), Loughborough University, 1999

-
- [2.17] L. C. Sim, G. Leedham and H. Schroder, "Performance Evaluation of Instruction Systolic Array Processors," Seventh International Conference on Control, Automation, Robotics and Vision, pp.910-913, 2002
- [2.18] ISATEC Software and Hardware GmbH, "The ISATEC Parallel Computer Systola 1024," Users Guide version 3.0, 1998
- [2.19] H-W. Lang, R. MaaB and M. Schimmler, "Implementation of a 1024-Processor Array Computer as an Add-On Board for Personal Computers," In Proceedings of HPCN, LNCS 797, Springer, pp. 487-488, 1994
- [2.20] M. Schimmler, "Instruction Systolic Arrays - Experiences with a First Implementation," In Proceedings of ISCA, 1992
- [2.21] Development of a low-cost hybrid massively parallel computing system [Online]. [Accessed : 24 July 2017]. Available from: <http://goanna.cs.rmit.edu.au/~heiko/Projects/Hybrid%20computing.htm>
- [2.22] W. Kolbe, "ISATEC Surface Quality Scannern SQS," Journal of Oberfldchentechnologie, vol. 3, 1997

CHAPTER 3:

IMPLEMENTATION OF INSTRUCTION SYSTOLIC ARRAY FOR SMART FABRICS

THIS chapter presents a more detailed explanation of the novel architecture proposed in the previous chapter. This chapter also explains and addresses the challenges of implementing the design using commercial off-the-shelf components. Taking the theory of the instruction systolic array, a prototype design is proposed. This chapter also discusses some candidate bus systems that can form the interconnects between processing elements and how off-the-shelf microcontrollers can be selected to produce a viable functional prototype.

3.1 A novel architecture for on-fabric parallel processing

In the subject of this thesis, a distributed wearable system is of interest. This can be mapped onto the ISA concept as shown in Fig. 3.1. The processing elements are connected to their neighbouring processing elements. Each processing element is closely coupled to different sensors. The northern boundary of the array is connected to the instruction stream flow controller which stores the array of instructions that needs to be passed to the processing elements. The western boundary of the array is connected to selector bit flow controller which stores the array of selector bits that needs to be passed to the processing elements. The processing elements and the sensors are closely coupled, which means both are co-located so that they can have local data flow and processing can be done locally. The processing elements and sensors are scalable where the array for the processing elements and the sensors connected to the processing elements can be increased or decreased.

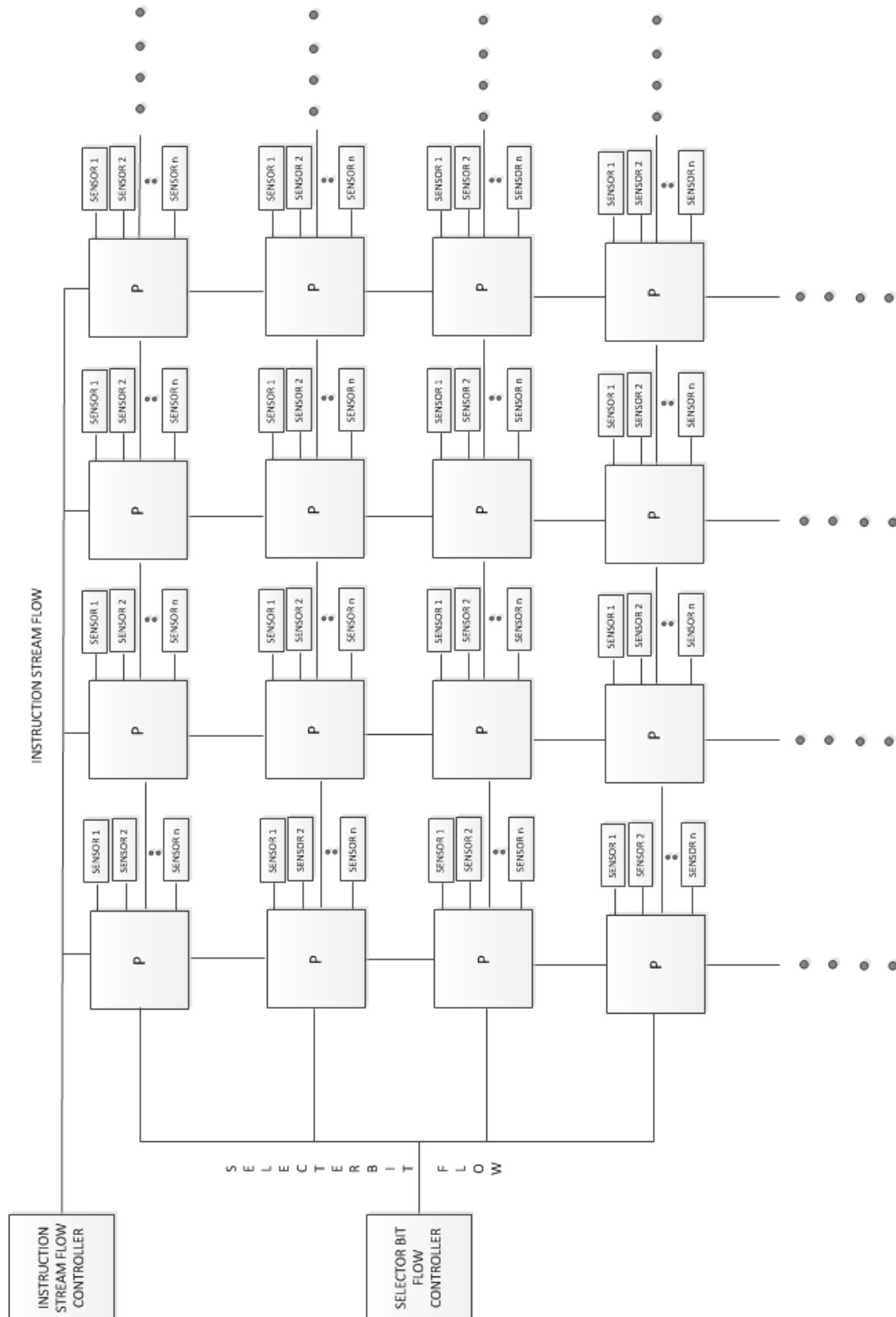


Figure 3.1: System concept

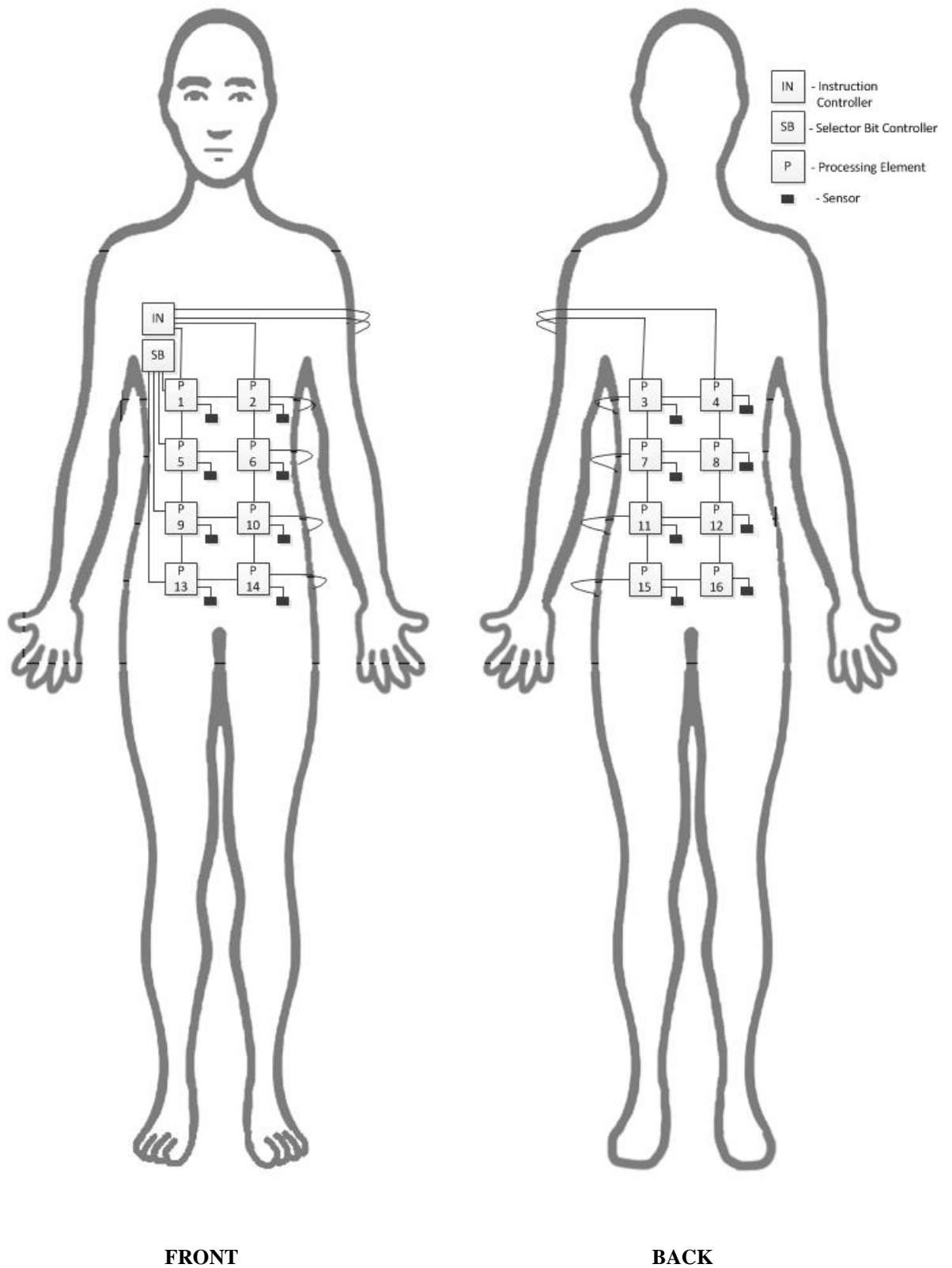


Figure 3.2: General concept of a sensor system with integrated processing elements for human body applications

As discussed earlier in Chapter 1, the ISA concept can be implemented for body sensing applications as shown in Fig 3.2. In the figure, the processing elements are distributed along the front and back side of the fabric worn on the human body. Each side is distributed with 8 processing elements and is closely coupled with their respective sensors.

3.2 Implementation of novel architecture

An important objective of this thesis is to test the concept proposed by implementing in a real application. The performance of such a system would be best optimised by custom Application Specific Integrated Circuit (ASIC) design. However, within the scope of the work, this is not realistic and as a consequence, certain compromises need to be made. It is assumed that off-the-shelf microcontrollers will be used as the processing elements which implement standard buses and protocols. It is expected that the system would prove the concept and reveal the properties of such a device.

The purpose of the implementation is to explore the merits and pitfalls of such a system, however, there is no expectation that performance will be fully optimised at this stage. In order to make use of off-the-shelf components such as sensors, conventional bus architectures for communication between elements has been assumed. Here the candidates for the bus are considered.

3.2.1 Candidates for bus systems

A reliable distributed embedded system can be achieved through a fast and efficient communication. The exact interconnections between the processing elements, sensors, instruction flow and selector bit flow using the bus are explained using Fig. 3.3 [3.1].

The most usual method of transmitting data in between two computers or between a computer and a peripheral device is serial communication. Serial communication transmits data to a receiver sequentially, one bit at a time, over a single communication line.

This transfer of information can be in different ways:

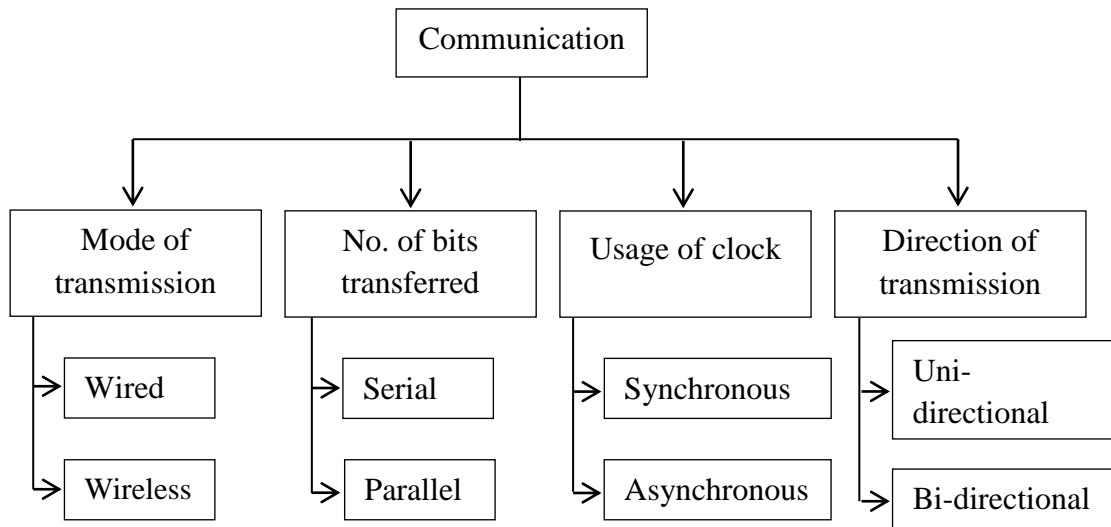


Figure 3.3: Different methods for transfer of information

The main advantage of serial communication is its low pin counts. Serial communication can be carried out by using just one input/output pin, while for parallel communication eight or more pins are required. There are so many common embedded system peripherals that support serial interfaces, like Liquid Crystal Displays (LCDs), temperature sensors, analog-to-digital and digital-to-analog converters [3.2].

Table 3.1: Difference between serial and parallel communication

SERIAL COMMUNICATION	PARALLEL COMMUNICATION
A serial port sends and receives data, one bit at a time over one wire.	A parallel port sends and receives data eight bits at a time over eight separate wires or lines.
Only a few wires are required for transmission and reception.	The setup looks bulkier because of the number of individual wires.
Serial communication is slower than parallel communication given the same signal frequency.	A parallel communication device sends and receives the same amount of data simultaneously, thus making it faster. In parallel you are transferring many bits at the same time, whereas in serial sends doing one bit at a time.
It is simpler and can be used over longer distances.	Can be used for shorter distance.

In comparison with parallel communication, serial communication has various advantages such as:

- It needs fewer interconnecting cables and therefore requires less space.
- Many peripheral devices and integrated circuits have serial interfaces.
- Clock skew between different channels is not a problem.
- There are fewer conductors as compared to that of parallel communication cables, therefore cross talk is not a big problem.
- It is comparatively cheaper to implement.

3.2.2 Serial bus protocols

There are various different protocols, with each one of them having its own interface requirements. Bus interface encodes the commands or the state of an input/output to digital information which is then transferred through the cable. The most commonly used standards in communication can be listed as [3.3]:

- 1) **UART** (Universal Asynchronous Receiver/Transmitter)
- 2) **I²C** (Inter Integrated circuit)
- 3) **SPI** (Serial Peripheral Interface)
- 4) **CAN** (Controller Area Network)
- 5) **USB** (Universal Serial Bus)

The relative advantages of these bus protocols, when applied to the proposed system, are listed in table 3.3 as shown below [3.4]:

Table 3.2: Comparison of different bus system

Protocols	Advantages	Disadvantages
UART	<ul style="list-style-type: none"> • Asynchronous serial communication. • Full duplex communication. 	<ul style="list-style-type: none"> • It is used for communication between equipments as an external bus. • Only two devices can be connected to the bus.
I ² C	<ul style="list-style-type: none"> • On PCB type bus between chips. 	<ul style="list-style-type: none"> • Can work only in half duplex mode.

	<ul style="list-style-type: none"> • Master and slave share a common clock. • Flexible data transmission rates. • Size of the address used for the slaves 7-bit, 8-bit and 10-bit support 127, 255, and 1023 devices respectively. 	<ul style="list-style-type: none"> • Requires pull-up resistors which can limit clock speed. • Imposes protocol overhead that reduces throughput.
SPI	<ul style="list-style-type: none"> • On chip or on-PCB type bus. • Synchronous serial communication. • Can work in full duplex mode. 	<ul style="list-style-type: none"> • Requires more pins on an IC package than I2C. • Can be used only for short distance communication.
CAN	<ul style="list-style-type: none"> • CAN bus is a vehicle bus standard designed for communication within a vehicle without a host computer. • Highly secured and priority based protocol. 	<ul style="list-style-type: none"> • Half Duplex as data cannot be sent and received simultaneously. • It is used for communication between equipments in automotive.
USB	<ul style="list-style-type: none"> • Supports up to 127 devices. • Plug and play. • Higher speed up to 12Mbps. 	<ul style="list-style-type: none"> • Significant hardware overhead • It is used for communication between equipments. • Not designed for simple buses.

There are clearly a number of bus protocols that could be used, and these can be commonly found on microcontrollers. Some of these come with on-chip hardware support and readily accessible from the software suite for the microcontroller. Of the ones available, both SPI and I²C are common, though of course there may be limits to the number of available buses on a single microcontroller. Owing to the large number of physical interconnections that are likely to be necessary to wire a suitable-sized sensor array a compromise has to be taken by selecting I²C bus and sharing these bus for both

the ISA inter element connections and sensors. Suitable sensors such as accelerometers are readily available with I²C.

3.3 Details of the Inter-Integrated Circuit (I²C) Bus

Typically, an embedded system contains one or more microcontrollers along with other peripheral devices such as, input/output expanders, sensors, memories, converters, matrix switches, LCD drivers [3.4]. The effort is to minimize the system complexity and the cost of connecting all those devices together. The main design requirement of the system is to make the slower devices capable of communicating with the system without slowing down the faster devices. A serial bus is required to satisfy these essentials. A bus meaning the detailed description for the formats, connections, addresses, procedures and protocols which mainly explains the rules on the bus. Serial data connections are preferred because they require just one or two signal wires as compared to a parallel bus, which needs at least eight data lines plus control signals. For any given communication channel, the best connection can be chosen based on the speed, number of hardware connections required and the distance between nodes.

From the Microchip manual [3.5] the Inter-Integrated Circuit (I²C) bus is explained as it mainly designed for short-range communication between chips within the same system by utilizing a software addressing system. It functions like a simplified local area network and needs just two wires. A simple bi-directional 2-wire bus is developed by Philips Semiconductors (now known as NXP Semiconductors) for an efficient inter-integrated circuit control. All the devices that are compatible with I²C bus integrate an on-chip interface which entitles them to communicate with each other through the I²C bus. Many interfacing problems faced while designing digital control circuits are solved by using this design concept. Typical I²C bus is shown in Fig 3.4.

The basic bus terminology is explained from[3.6],

- **Transmitter** The device that transmits data on to the bus.
- **Receiver** The device that receives data from the bus.
- **Master** The device from which the clock originates, starts communication, sending I²C commands and halting communication.
- **Slave** The device that ‘listens’ to the bus and is addressed by the master.
- **Multi-master** I²C can have more than one master and each can send commands.

- **Arbitration** The process that determines which master has control of the bus.
- **Synchronization** Process whereby the clocks of two or more devices are synchronised.

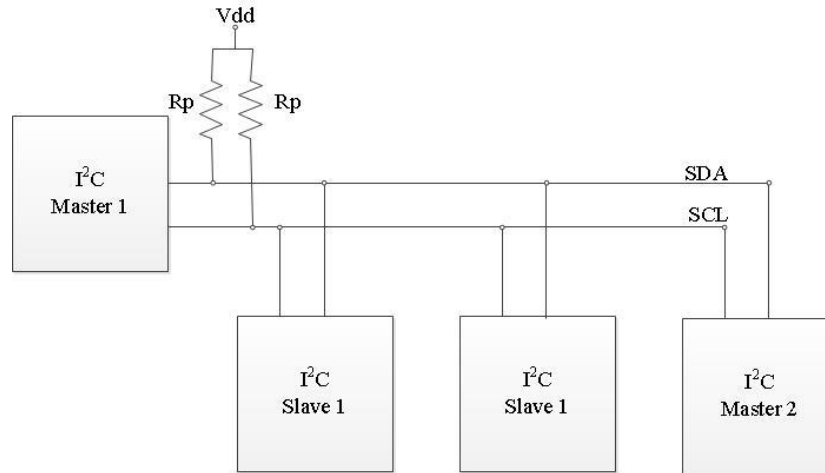


Figure 3.4: Typical I²C bus

The working process of I²C is explained in the NXP Semiconductors specification [3.7]. I²C works on synchronous communication. It is a bi-directional protocol which permits a master device to initialise communication with a slave device. Both these devices exchange data with each other which is then implemented by an “Acknowledge” system. The Acknowledge (ACK) system is considered as one of the important characteristics of an I²C system. It permits the data to be sent in one direction from one device to another device through the I²C bus. That device will ACK to signal that the data was received. As a peripheral can acknowledge data, there is an uncertainty regarding whether the data reached the peripheral. The data must be timed very precisely, however, RS232 and other asynchronous protocols do not utilise a clock pulse. As I²C is having a clock signal, the clock can vary without interrupting the data. The changes in clock rate will simply change the data rate.

The I²C is based on the principle of Master-Slave protocol, the master device controls the Serial Clock Line (SCL) and initialises the data transfers as well. This line orders the timing of all the transfers taking place through I²C bus. Slave devices are capable of manipulating this line but they can only make the line low, which means that item on the bus is not able to deal with more incoming data. When the line is forced to be low, more data is impossible to clock into any device. This situation is termed as “Clock

Stretching". As already mentioned, no data will be shifted unless the clock is manipulated. Same clock line SCL controls all the slaves. On I²C bus, the data can flow in either direction, but master device controls the data when it flows. There are a number of conditions of I²C bus. These conditions specify the events of starting, stopping, acknowledging a transfer among others [3.7].

Every device that is connected to the bus is software addressable by using a unique address. All the times, simple master/slave relationships are present, where masters can function like master-transmitters or master-receivers. It is true multi-master bus having features such as arbitration to prevent data corruption and collision detection, in case two or more masters initialize data transfer simultaneously. Bi-directional, 8-bit oriented, serial data transfers can be made at:

- up to 100 kbps in the Standard-mode
- up to 400 kbps in the Fast-mode
- up to 1 Mbps in Fast-mode Plus
- up to 3.4 Mbps in the High-speed mode

3.3.1 Bus Signals

I²C is a serial interface which utilises two signals to exchange data serially with other device. The signals used are [3.7]:

- **SDA:** This signal is called as Serial Data. Any data transferred from one device to another goes on this line.
- **SCL:** This signal is called as Serial Clock Line signal. This signal is initiated by the master device, which controls when the data is sent and when it is read. This signal can be forced to low making the data impossible to clock.

There are just two possibilities for electric states of I²C lines, which are *drive low* and *float high*. The concept of pull up resistor is really important in the functioning of I²C. I²C operates by having a pull-up resistor on the line and devices are only capable of pulling the line low to transmit the data. The line will be in state *float high* if none of the devices are pulling it. The line would be floating to an unknown state in case no pull up resistors are used.

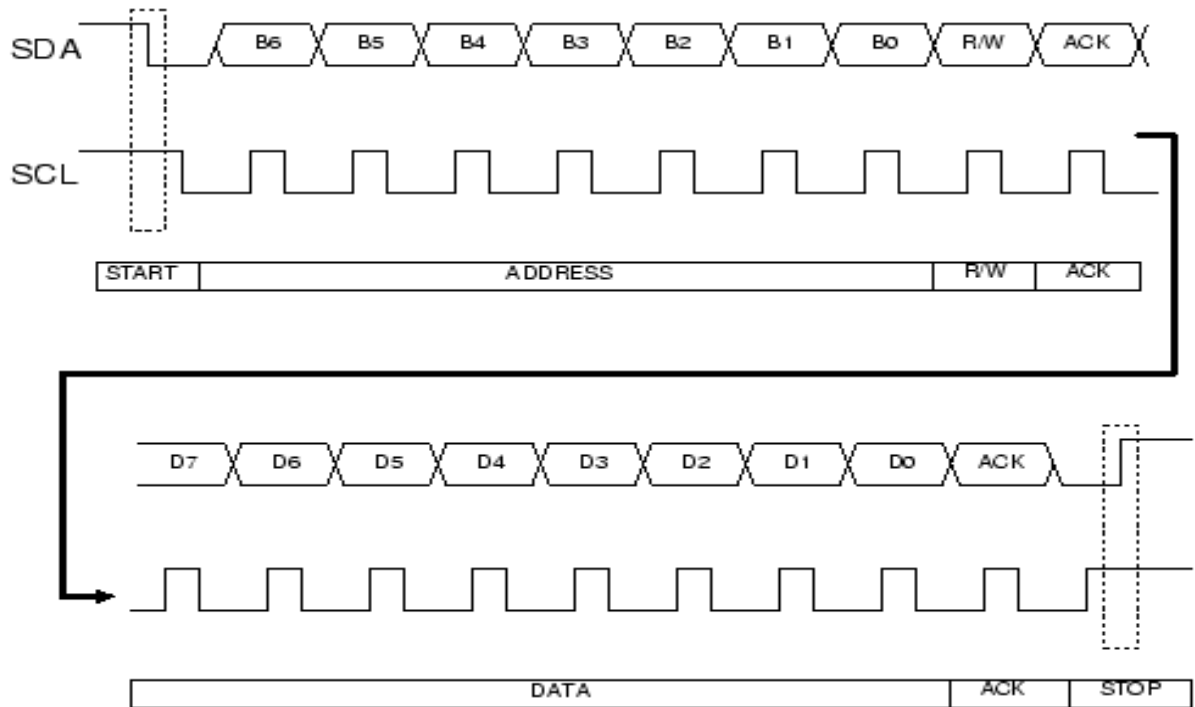


Figure 3.5: Basic Mechanism in I²C from NXP Semiconductors adapted from [3.7]

1. Data transfer is initiated with a start bit signalled by SDA being pulled low while SCL stays high.
2. SCL is pulled low, and SDA sets the first data bit level while keeping SCL low.
3. The data are received when SCL rises for the first bit. For a bit to be valid, SDA must not change between a rising edge of SCL and the subsequent falling edge.
4. This process repeats, SDA transitioning while SCL is low, and the data being read while SCL is high.
5. A stop bit is signalled when SCL rises, followed by SDA rising.

In order to avoid false marker detection, there is a minimum delay between the SCL falling edge and changing SDA, and between changing SDA and the SCL rising edge as shown in Fig 3.5.

There might be a disagreement if one device is trying to drive the line high while the other device is trying to drive it low. This disagreement might result in damaging either or both devices operating on the line. To avoid this situation, the pull-up-drive low system is used that regulates which device has control of the bus. If other devices want

to use the bus at the same time, this system indicates that the bus is busy. This device will figure out that bus is already driven low and is used by other device currently.

Thus, the working of the I²C bus and their signals and communication has been explained in this section. The next section will explain the I²C bus connection in the proposed architecture.

3.4 Prototype Design

The concept is based on the instruction systolic array which consists of an array of Processing elements connected with the different peripheral components preferably sensors. The data has to be shared by other processing elements and sensors through serial data communication. A suitable communication channel, which is I²C bus communication, has been selected for the hardware connections.

The processing elements are connected in a mesh-like structure in Fig. 3.6 which is globally interfaced to all the other devices including sensors through the I²C bus. Separate processors are allocated for global input of instructions and selector bits apart from the array of processing elements. The whole model is designed in such a way that each processing element will have four I²C protocols i.e. two of them (west and north) acts as slave and other two (east and south) acts as master. Each master will be connected to the adjacent slaves and makes sure that all the slave addresses are addressed by it. The sensor's data are communicated and transferred only by the masters as the clock and data initiation processes are controlled by the master. Each processing element is connected to its own sensors as shown in Fig. 3.7. Each processing element combined with its own sensors is termed as a unit cell.

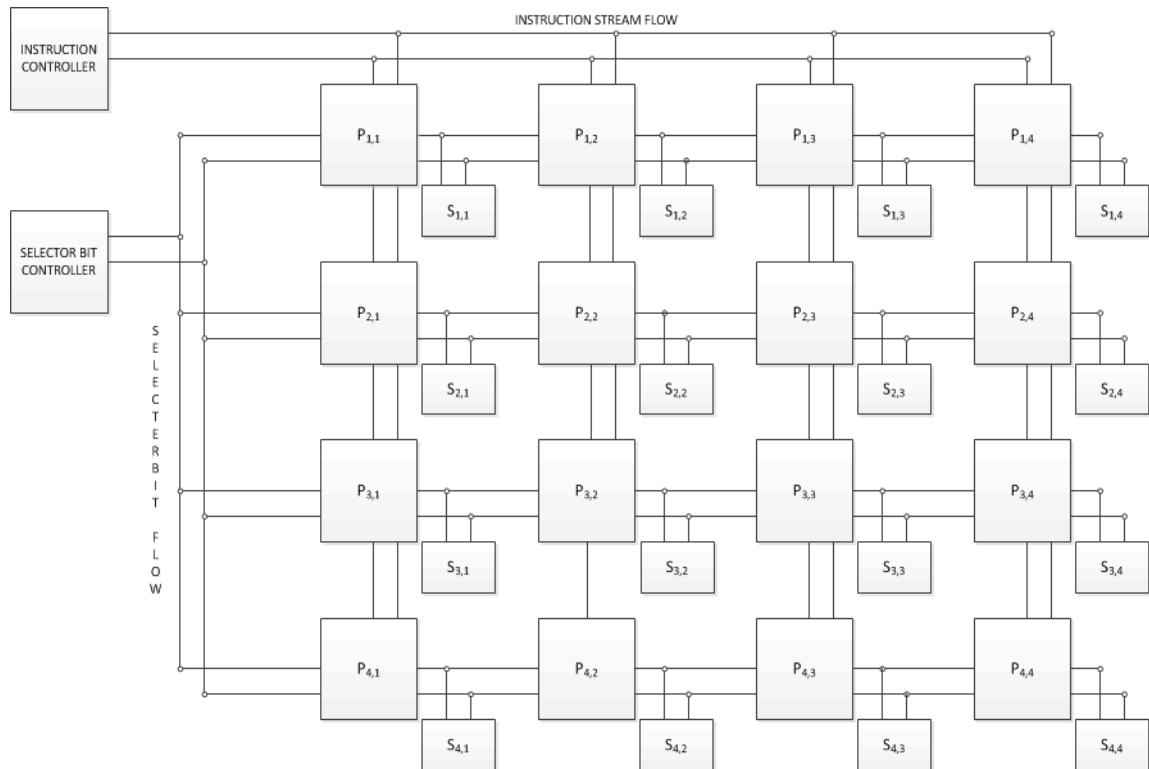


Figure 3.6: Processor array showing grid arrangement

According to the ISA concept, instructions and selector bits need to be propagated through the chain of processors according to the clock. It is also understood that all the processors connected at the extreme left and top (west and north) are only meant for getting the inputs from the instruction and selector bit controllers. Different operations can be performed on the desired data where the operations are decided by the instructions which are propagating through processors in a systematic manner and specific data on which the instructions have to be performed are decided by the selector bits. The instruction flow will be from top to bottom (north to south) of the array. The selector bits flow from left to right (west to east) of the array row and the selector bits must be 1, so that particular instruction is carried out at that particular processing element.

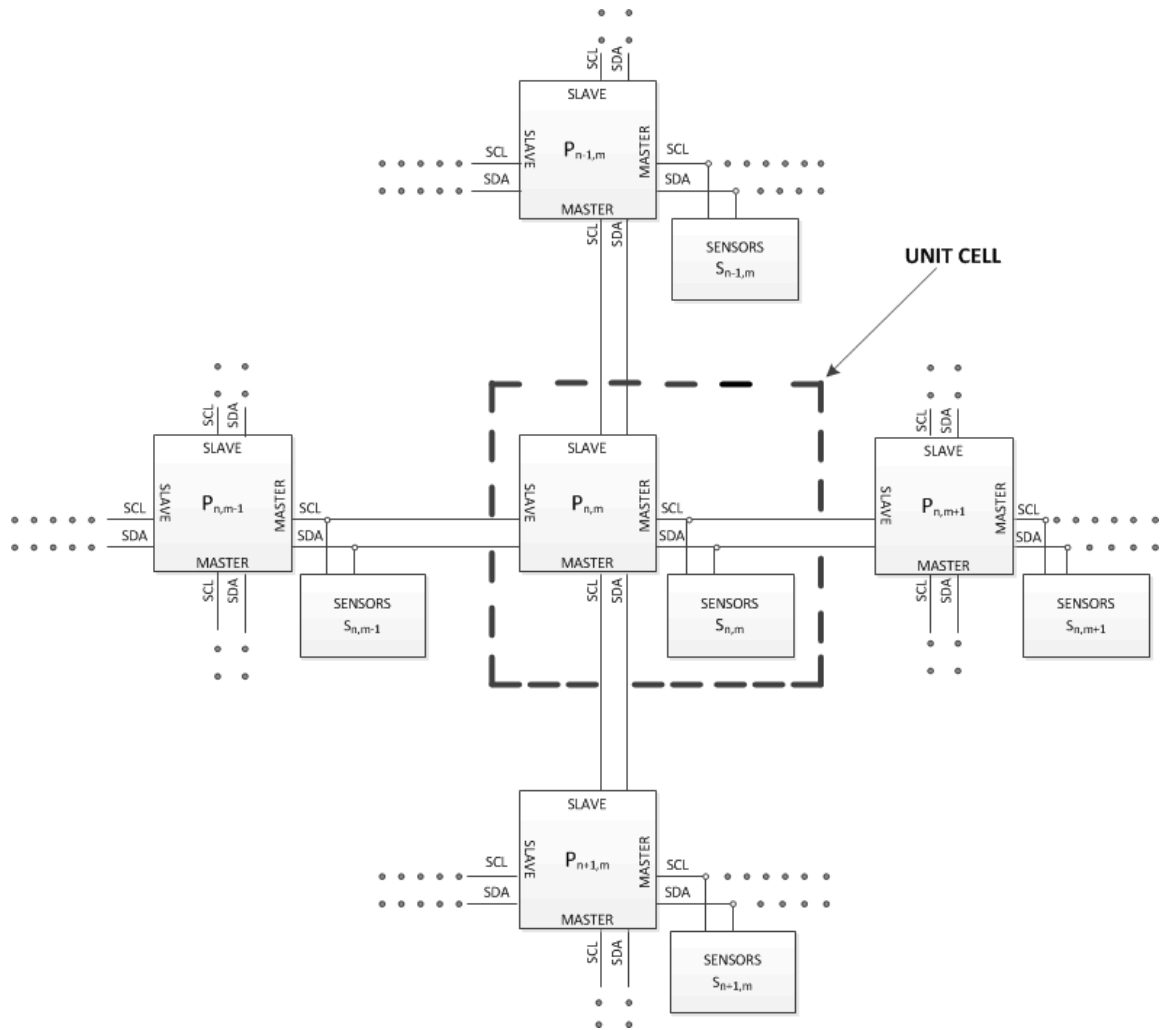


Figure 3.7: Detail of I²C bus connections

Each cycle is divided into three stages, they are fetch, execute and write. This mechanism is explained below,

- All processors start in a default state listening on slave ports (or filled with NOPs, ideally).
- 1-byte instruction is written to north boundary slaves at the same time 1-bit selector bit (as part of control bit) written to west boundary slaves.
- Instruction written to north slaves and sensor values are read from the east port.
- Then the communication takes place in the following order between North-South, South-North, East-West and West-East.

Then the execution of the instruction takes place in the processing element.

3.5 Selection of Microcontroller for the Processing Element

The ISA is implemented by using commercially available microcontrollers. Number of I²C interface buses were taken into consideration while choosing the microcontrollers for implementing ISA. The initial choice to implement the ISA concept was on a Microchip PIC16F1829 microcontroller shown in Fig 3.8. PIC16F1829 is a 20-pin microcontroller with two I²C bus interfaces. The idea was to implement the ISA concept using two available I²C bus interfaces and bit banging two more I²C interfaces. Challenges had been faced during the implementation of the ISA concept where there were timing issues with the software modified pins.



Figure 3.8: Microchip PIC16F1829

Due to the implementation challenges on PIC16F1829, research went to explore other microcontrollers with more I²C interfaces. Thus, the processing elements that have been chosen for implementing ISA concept are 32-bit ARM Cortex-M0+ LPC824 microcontrollers. Fig 3.9 shows ARM Cortex-M0+ LPC824 microcontroller mounted on NXP LPC824-MAX board. The reason for choosing LPC824 microcontroller is it includes four I²C bus interfaces. One I²C supports Fast-mode Plus with 1 Mbit/s data rates on two true open-drain pins and listen mode. Three I²Cs support data rates up to 400 kbps on standard digital pins [3.8].

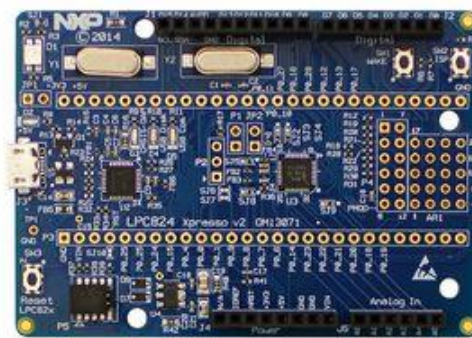


Figure 3.9: 32-bit ARM Cortex-M0+ LPC824 microcontroller mounted on NXP LPC824-MAX board

The LPC824 microcontrollers are mounted on LPC824-MAX board which is developed by NXP to enable evaluation and prototyping with the LPC824 microcontrollers. The array of microcontrollers connected using ISA concept with a peripheral device (sensor) and their I²C connections are shown in Fig. 3.10.

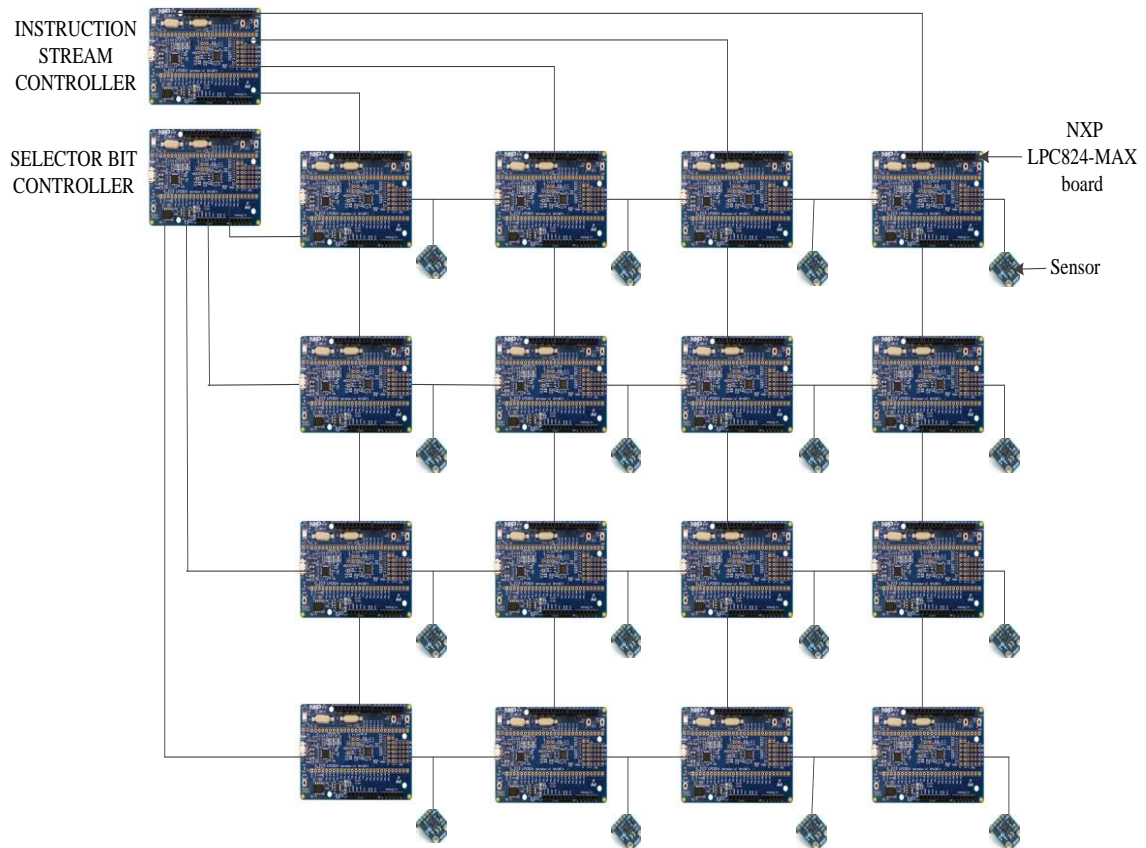


Figure 3.10: Processor Array with peripherals

Fig. 3.11 shows the connections between two microcontrollers with four I²C buses connected between them. The sensors are attached to the east port of both the microcontrollers. The working of all four I²C buses available on the microcontroller has been verified by sending an instruction and receiving the data from the sensor through the serial port. I²C0, I²C1, I²C2, I²C3 represents all the four I²C bus connected between the two microcontrollers.

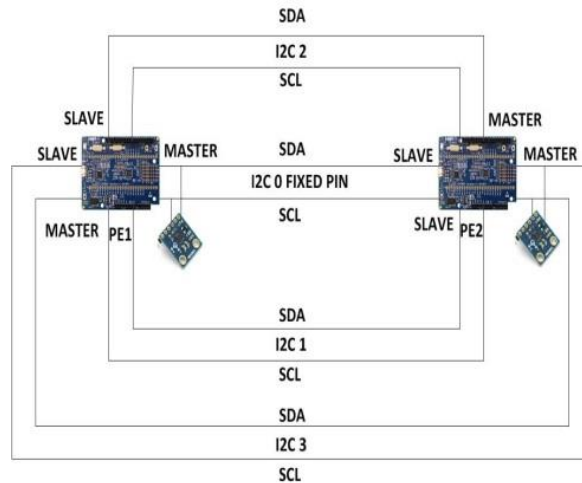


Figure 3.11: I²C connection between two microcontrollers with sensor

3.6 Power and programming interface for the array

The prototype board has been designed with 16 microcontrollers distributed in a 4 x 4 array and 2 microcontrollers as instruction and selector bit flow controllers as shown in Fig.3.12. The wires between the microcontrollers are the I²C buses. The microcontrollers are powered by the USB cable running from the hubs.

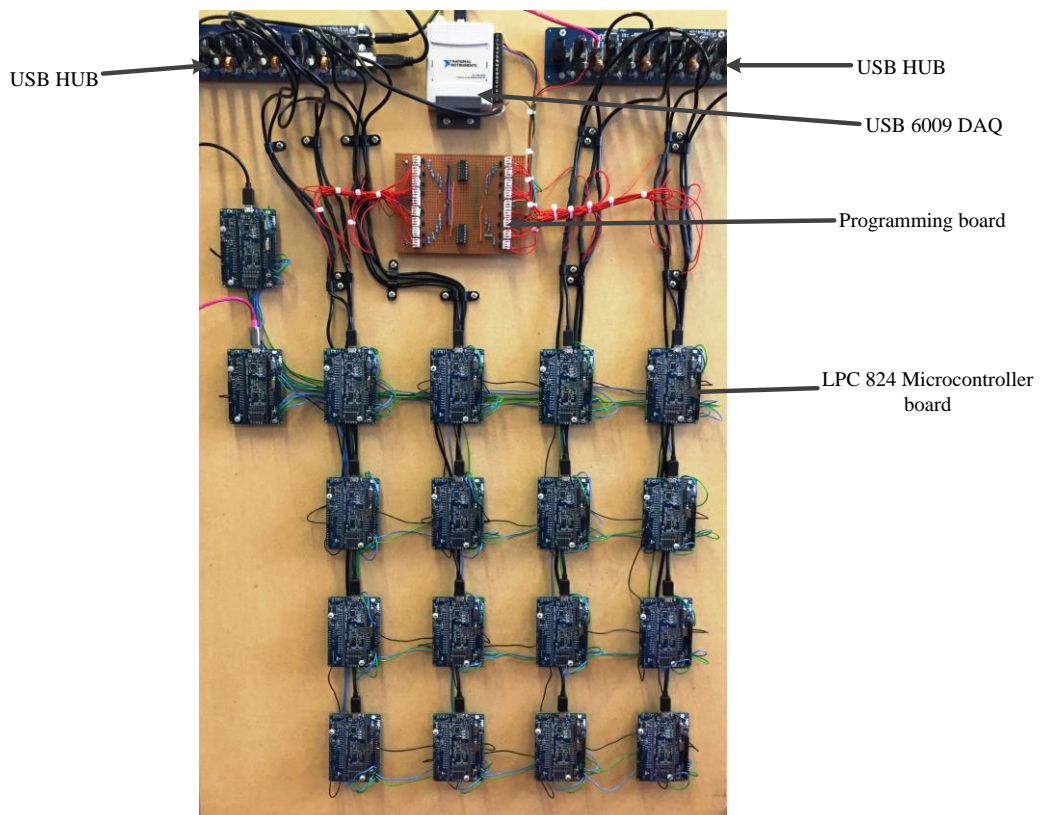


Figure 3.12: Prototype board

A secondary unit can be seen on top of the array of microcontrollers in Fig.3.12, which is developed for deploying the firmware to the array of microcontrollers without manually switching between them. The secondary circuit is also used for powering all the microcontrollers from the USB hub and also used as a serial interface with the host computer in case to extract the output data. The secondary unit consists of two 10 port USB hub, a NI USB 6009 DAQ, a programming board which acts as a switching circuit. The power for the microcontrollers is extracted from the USB hub. The programming board consists of two HCT164 8-Bit Parallel-Out Serial Shift Register, 16 ZTX551 PNP Silicon Planar Medium Power Transistor, 16 1K Resistor and 18 wire to board connectors. All these are soldered according to the schematic shown in Fig. 3.14 on a strip board as shown in Fig 3.13.

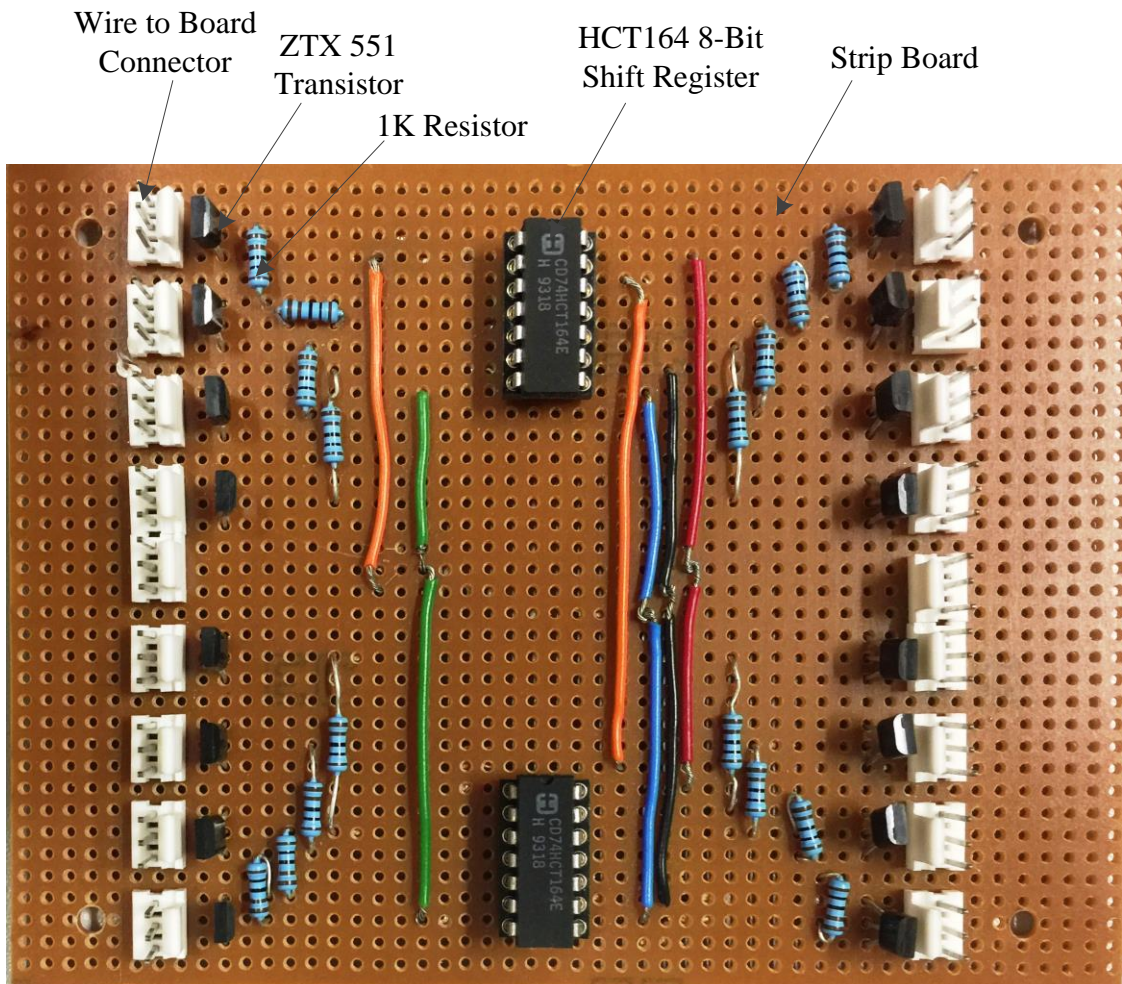


Figure 3.13: Programming board (switching circuit)

Three digital lines are taken from the NI USB 6009 DAQ and connected to the shift registers as data, memory clear and clock. The power from the micro USB cable of each microcontroller is passed through the switching circuit before reaching the microcontroller directly from the USB hub so that the switching circuit turns on the power of each microcontroller in an order to program them one at a time.

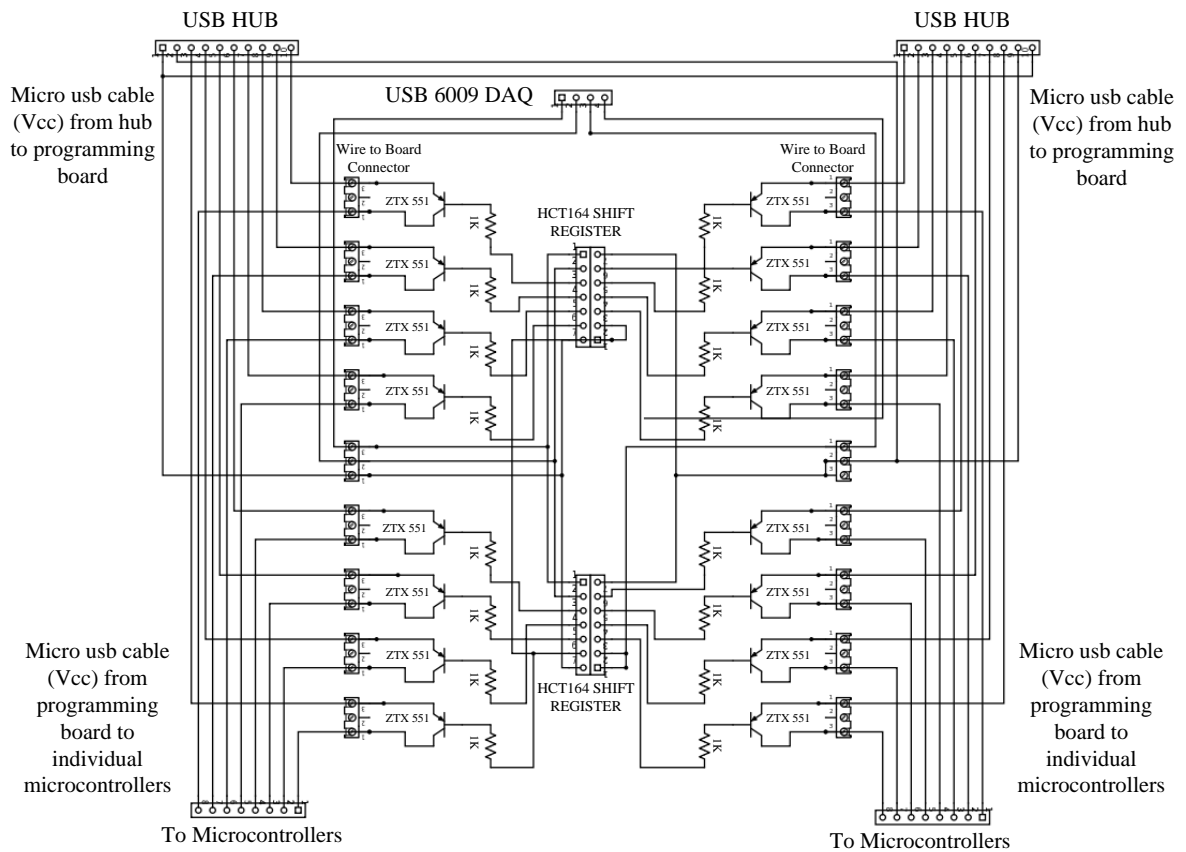


Figure 3.14: Schematic for the switching circuit

An application as seen in Fig. 3.15 has been developed using LabVIEW to control the programming of the microcontrollers. Total number of microcontrollers that needs to be programmed can be set in the application. The digital lines can also be selected from the application. Once the number of microcontrollers and the digital lines are selected the location of the object file needs to be included in the command line so that the object file can be programmed on to the microcontrollers. If an error occurs while programming the controllers the *processor failed* light will turn on and the process will

be ended. The status of the microcontrollers can be viewed in the *processor report* section.

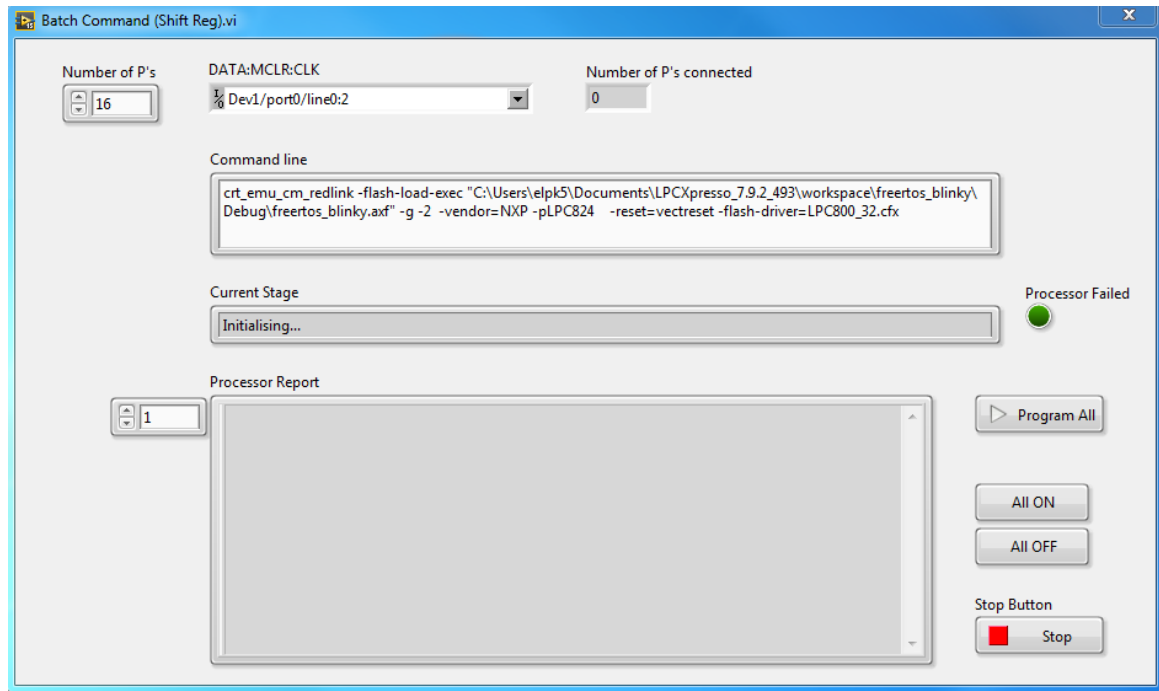


Figure 3.15: Application for programming the microcontrollers

3.7 Conclusion

In this chapter, a prototype architecture based on the concept has been described, along with a method and circuit that allows programs to be developed. A few compromises have been made during the implementing of the concept such as opting a particular bus system, selecting a microcontroller for the processing element, and sharing one of the processing devices interconnects with the peripheral devices (sensors). It can be expected that these will have an impact on the performance. However, there is a lot to research in terms of programming and realising such a device that needs to be done prior to optimising in the form of custom ASICs. As this has been previously stated, the purpose of the thesis is not to fully optimise but to explore the architecture and to study the programmer's model.

References

- [3.1] C. Patil, "Development of a Simple Serial Communication Protocol for Microcontrollers (SSCPM)," International Journal of Scientific and Research Publications, vol. 1, no. 1, pp. 1–6, 2011
- [3.2] L. K. John, "Bus Architectures," In: Z.Navabi, D. R. Kaeli ed. Computer Science and Engineering, Eolss Publishers Co. Ltd., pp. 109-130, 2009
- [3.3] A. Kumar, A. Kumar and D. Jha, "Serial communication for Embedded Systems," Technical Report [Online]. [Accessed : 25 October 2016]. Available from: <http://www.embedded.com/>
- [3.4] S. C. Sankhyan, "Design & Implementation of I2C Master Controller Interfaced With RAM Using VHDL," International Journal of Engineering Research and Applications," vol. 4, no. 7, pp. 67–70, 2014
- [3.5] Microchip, "dsPIC33 / PIC24 Family Reference Manual," [Online]. [Accessed : 24 July 2017]. Available from: <http://ww1.microchip.com/downloads/en/DeviceDoc/70000600d.pdf>
- [3.6] I2C Info – I2C Bus, Interface and Protocol, "I²C Bus Specification," [Online]. [Accessed : 02 March 2018]. Available from <http://i2c.info/i2c-bus-specification>
- [3.7] N.X P.Semiconductors, "UM10204 I2C-bus specification and user manual," [Online]. [Accessed : 24 July 2017]. Available from:<https://www.nxp.com/docs/en/user-guide/UM10204.pdf>
- [3.8] N.X P.Semiconductors, "LPC82x Product data sheet," [Online]. [Accessed : 24 July 2017]. Available from: <http://www.nxp.com/docs/en/data-sheet/LPC82X.pdf>

CHAPTER 4:

PROGRAMMING AND VALIDATION OF THE INSTRUCTION SYSTOLIC ARRAY

THIS chapter of the thesis describes the programming of the instructing systolic array and implementing the instruction systolic array on an array of off-the-shelf microcontrollers. To illustrate some of the basic definitions of the previous chapter, simple parallel algorithms are validated in this chapter.

4.1 Programming the Instruction Systolic Array

In ISA, a sequence of instructions and selector bits are pumped through an array of processing elements which can efficiently execute instructions and selector bits. An ISA is capable of executing a large variety of parallel algorithms, even if every processing element can execute only a few different instructions (see section 4.2 and 4.3). To program the processing elements for executing parallel algorithms, the operations of instruction and selector bit cycles need to be efficient.

The ISA application is programmed on to the chosen ARM Cortex-M0+ LPC824 microcontrollers. The Instruction and Selector bit controller holds the sequence of instruction and selector bits that will be passed to the microcontroller array. All the microcontrollers in the implemented array itself share a common firmware which is deployed using the method described in the previous chapter. The sending of instruction and selector bits are disabled at the end boundary microcontrollers. It is challenging to represent a program designed for an ISA in a conventional way and so a special notational scheme is helpful to understand the operation [4.1].

Instruction Controller:

1. Initialise system clock & port pins as required.
2. Initialise all 4 I²C port as master.
3. Send Instructions every tick.

4. Stop if all the instructions are completed.

Selector bit Controller:

1. Initialise system clock & port pins as required.
2. Initialise all 4 I²C port as master.
3. Send Selector bits every tick.
4. Stop if all the selector bits are completed.

Processing Element Controller:

1. Initialise system clock & port pins as required.
2. Initialise 2 I²C port as Slave (North & West).
3. Initialise 2 I²C port as Master (South & East).
4. The processor waits for frame bytes to be received from North and West Slave ports.
5. The instruction byte and the selector bit are received through north and west ports separately.
6. After receiving both the instruction and the selector bit they are decoded and executed by an interrupt service routine.
7. Once the execution is complete the instruction and the selector bit is then forwarded to the neighbours through south and east port.

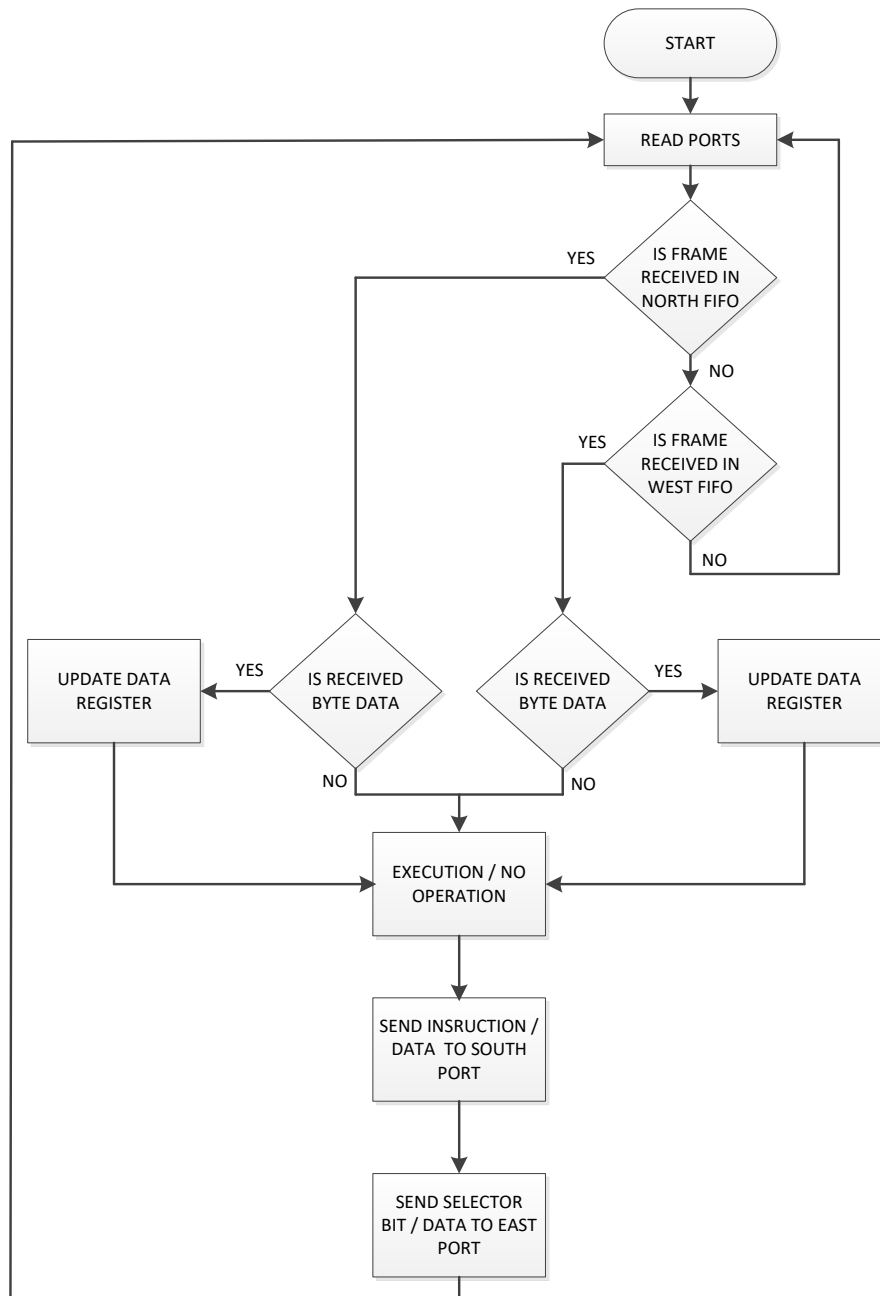


Figure 4.1: Working of the ISA program

The instruction and selector bit microcontrollers are programmed with a sequence of instructions and selector bits. Once all the microcontrollers are flashed with the firmware, all the processing element microcontrollers are initialised, all the processing element microcontrollers will be in their default mode which is *listening*. The instruction and selector bits are passed to the processing element; the latter is in the *listening* mode waiting for the frame bytes to be received from the north and west slave I²C ports. Once both the instruction and selector bit are received they are then decoded

and executed through an interrupt service routine. After the execution, the instruction and the selector bit are then forwarded to the neighbours through the south and east master I²C ports. Once the frame has been received in the corresponding FIFO register, the frame is then decoded to find whether the frame contains instruction and selector bit or data. If the received frame is data, the data register is updated else if the received frame is instruction and selector bit, depending on the instruction and selector bit it will either execute the instruction or no operation will occur. After this process, the instruction followed by the selector bit will be sent in a sequence through the south and east master ports to the neighbouring microcontrollers. Fig 4.1 shows the working of ISA program.

The next section of the thesis is to validate the concept of the implemented instruction systolic array using two simple parallel algorithms. Two well-known parallel algorithms: merge algorithm and matrix multiplication have been run to validate the instruction systolic array.

4.2 Merge Algorithm Validation

The Merge algorithm was first proposed by Kunde et al. [4.1]. The merge algorithm is comparatively simple sorting algorithm used in parallel computing. It was initially developed for use on parallel processors with local interconnections. It starts operating by comparing all indexed pairs of neighbouring elements in the array. If any pair is in wrong order, that is the first is larger than the second, the elements of the pair get switched. The above step is repeated continuously until all the elements in the array are sorted. In case of parallel processors, this process takes place simultaneously in all the processing elements depending on instruction on the particular processing element.

4.2.1 Algorithm

An instruction systolic array implementation of merge algorithm through parallel algorithm is illustrated below from [4.1],

Step 1: Sort all columns of the 4×4 array by odd-even-transposition sort.

Step 2: Sort all rows of the 4×4 array by odd-even-transposition sort.

The ISA program for the merge algorithm is presented in Fig. 4.2. The figure shows the set of instructions and selector bits that will flow through the array. In Fig. 4.2, the instruction and the selector bit part of the program are represented in parallelogram shape made up of their respective instruction and selector bits diagonals. Diagonals 1 to 6 correspond to step 1 and diagonals 7 to 12 to step 2 of the merge algorithm. A set of no operation instructions is flushed through the array before and after the instruction and selector bit diagonal. The merge algorithm is scalable the number of instructions and selector bits will be increased according to the increase in the size of the array [4.2].

4.2.2 Program

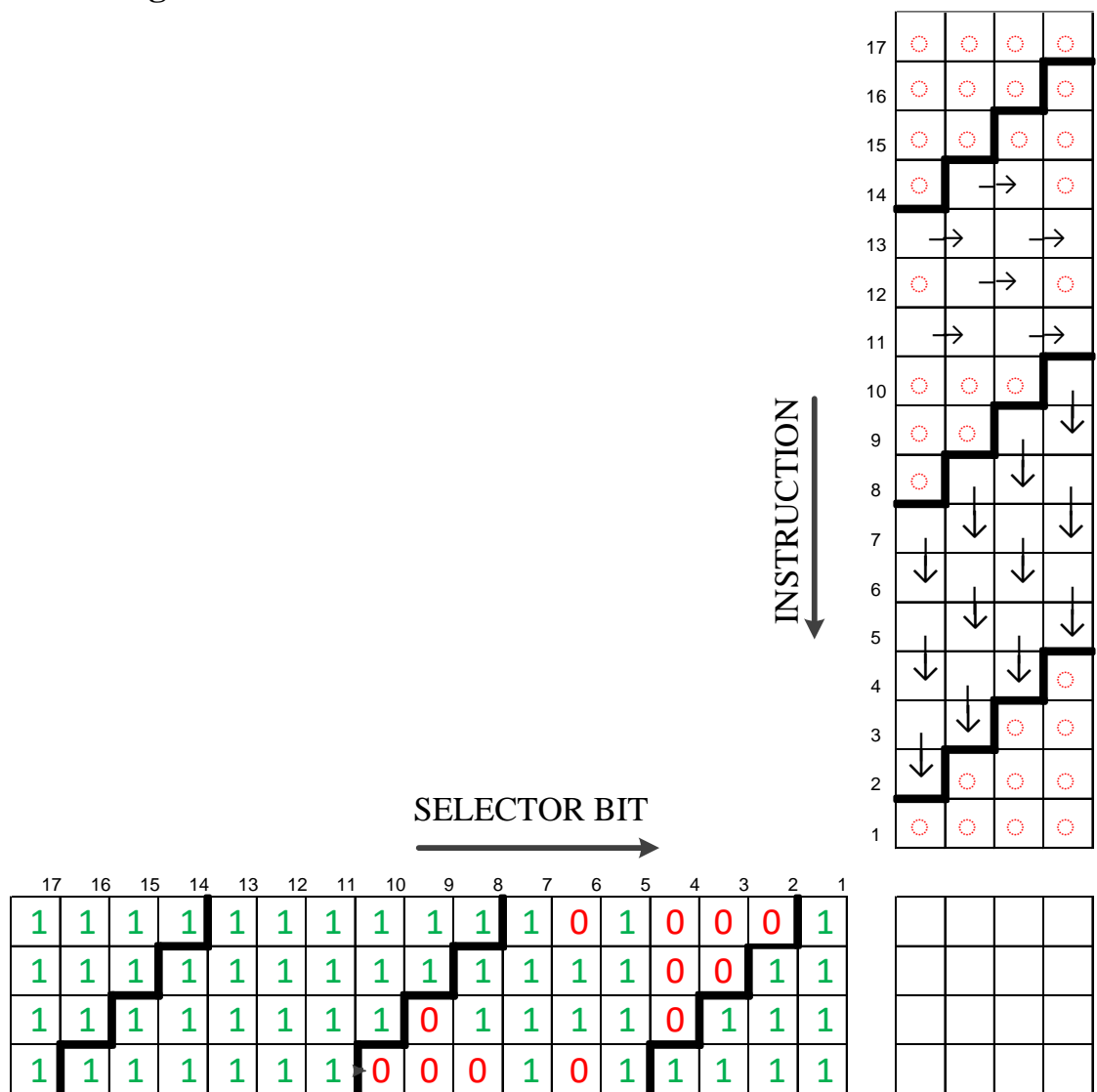







Figure 4.2: ISA program for merge algorithm

The meaning of the instruction symbols on Fig. 4.1 are illustrated below in the table,

Table 4.1: Instruction symbol definition

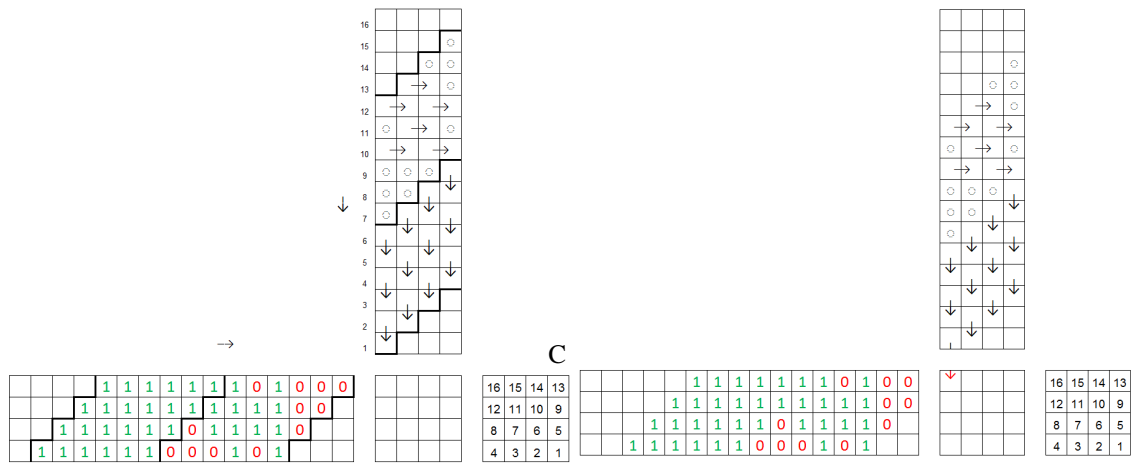
Symbol	Op-Code	Definition
	$C : -\min (C, C_{\text{lower}})$	If $C > C_{\text{lower}}$ then the content of C and C_{lower} are exchanged.
	$C : -\max (C, C_{\text{upper}})$	If $C < C_{\text{upper}}$ then the content of C and C_{upper} are exchanged.
	$C : -\min (C, C_{\text{right}})$	If $C > C_{\text{right}}$ then the content of C and C_{right} are exchanged.
	$C : -\max (C, C_{\text{left}})$	If $C < C_{\text{left}}$ then the content of C and C_{left} are exchanged.
	No Operation	No operation will occur

where C is the communication register

4.2.3 Numerical example

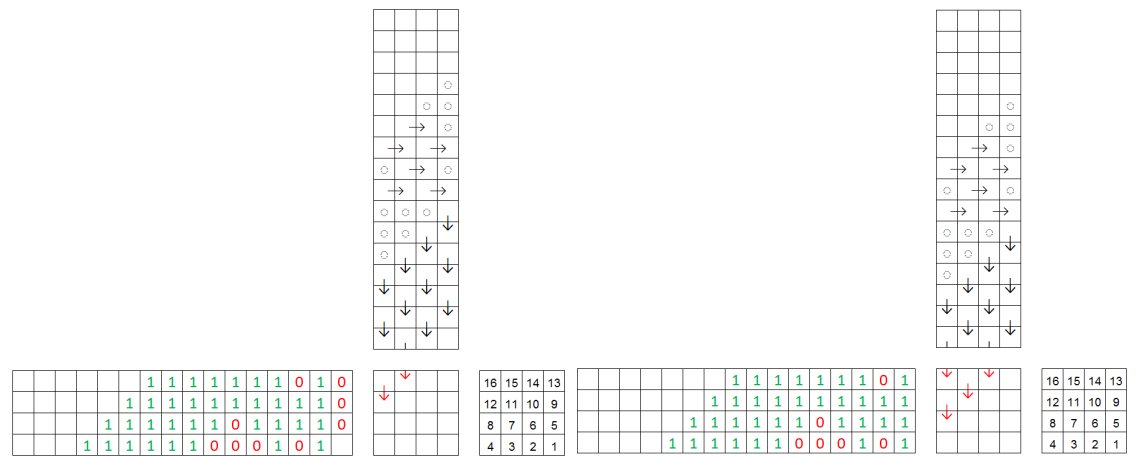
The example below shows the step by step executions of the instructions along the array. The instructions in green show the execution of the instruction on a particular microcontroller and the instruction in red represents no operation. The contents of the communication register C for each processing element is also shown along the execution of the instruction. The contents of C are shown after the instruction has been executed. Matrix X is the initial contents of the array before the execution of the instructions and Matrix Y is the contents of the array after the instructions are executed.

$$X = \begin{bmatrix} 16 & 15 & 14 & 13 \\ 12 & 11 & 10 & 9 \\ 8 & 7 & 6 & 5 \\ 4 & 3 & 2 & 1 \end{bmatrix} \quad Y = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$



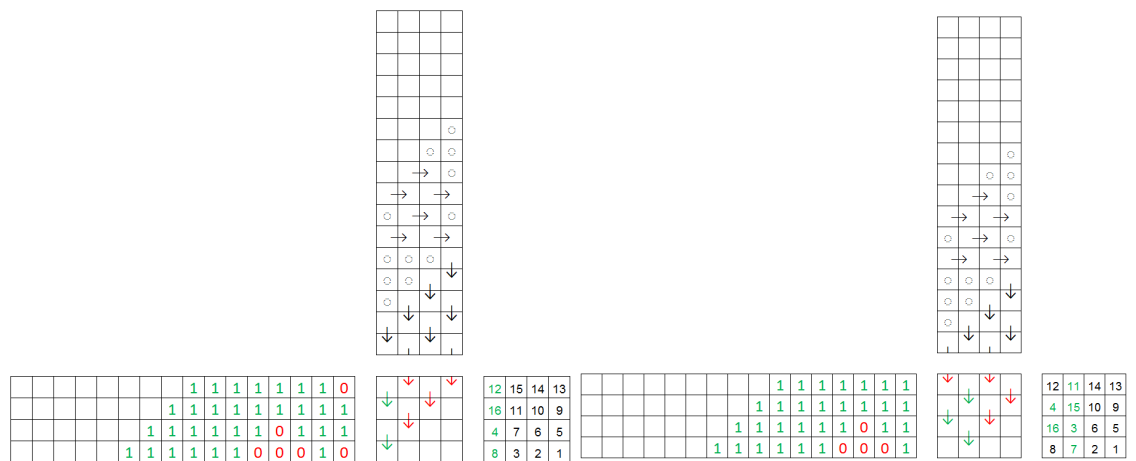
(1)

(2)



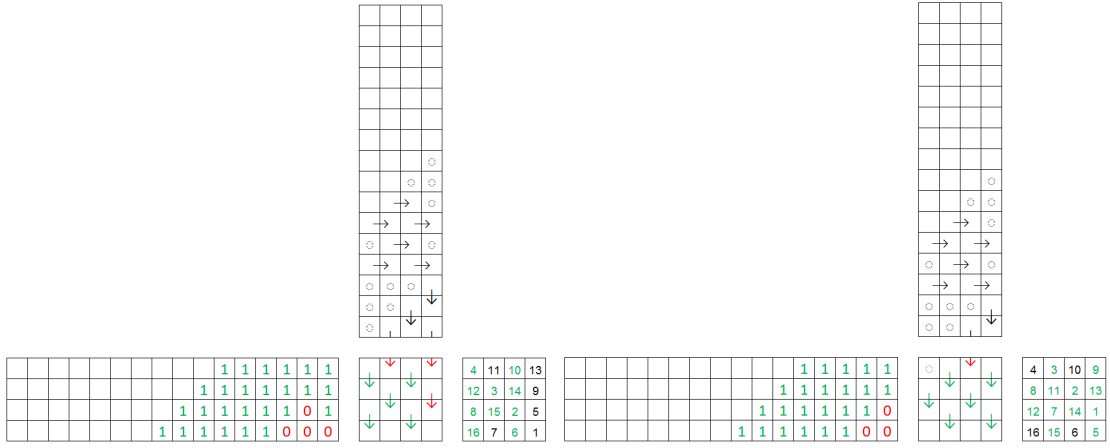
(3)

(4)



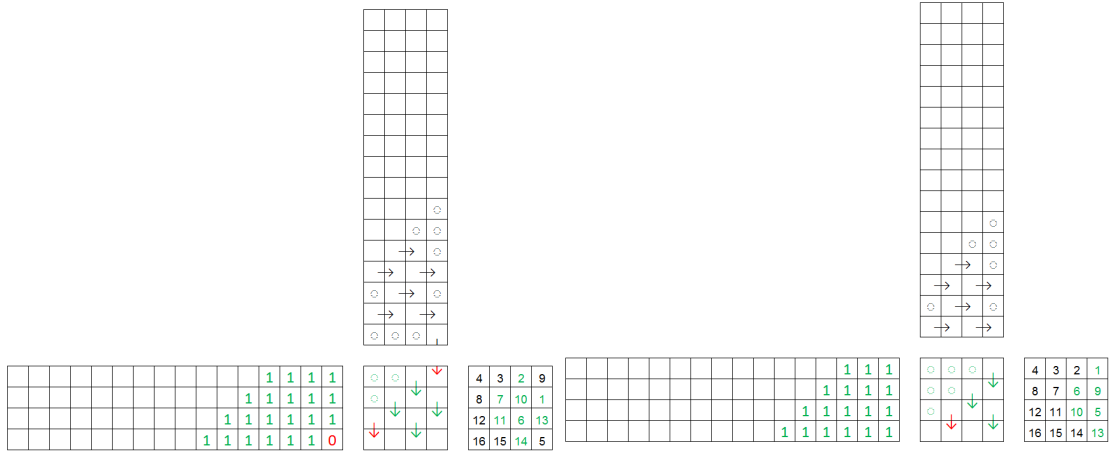
(5)

(6)



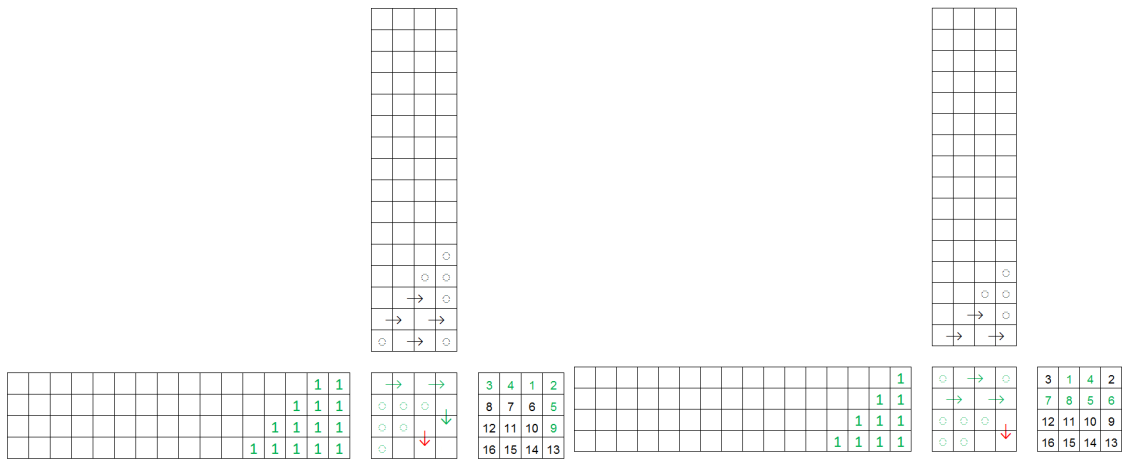
(7)

(8)



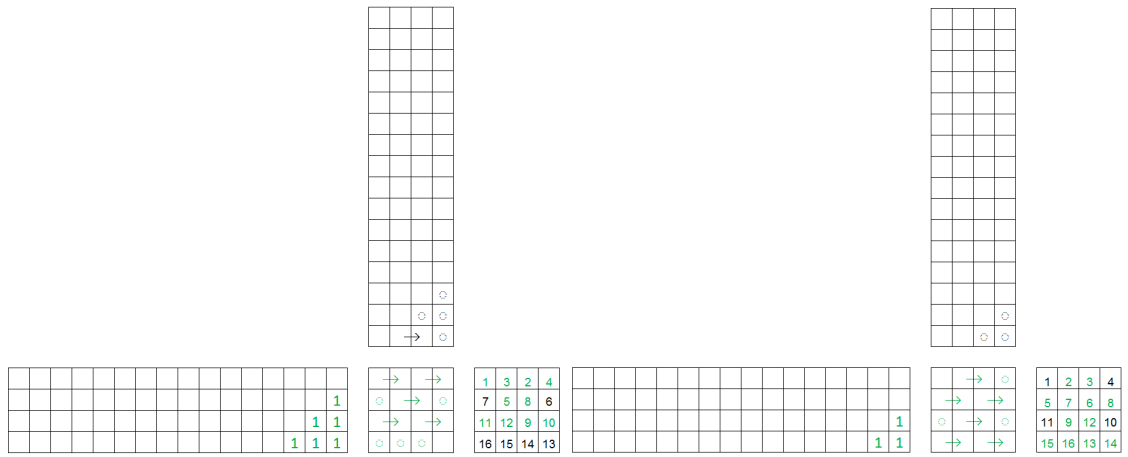
(9)

(10)



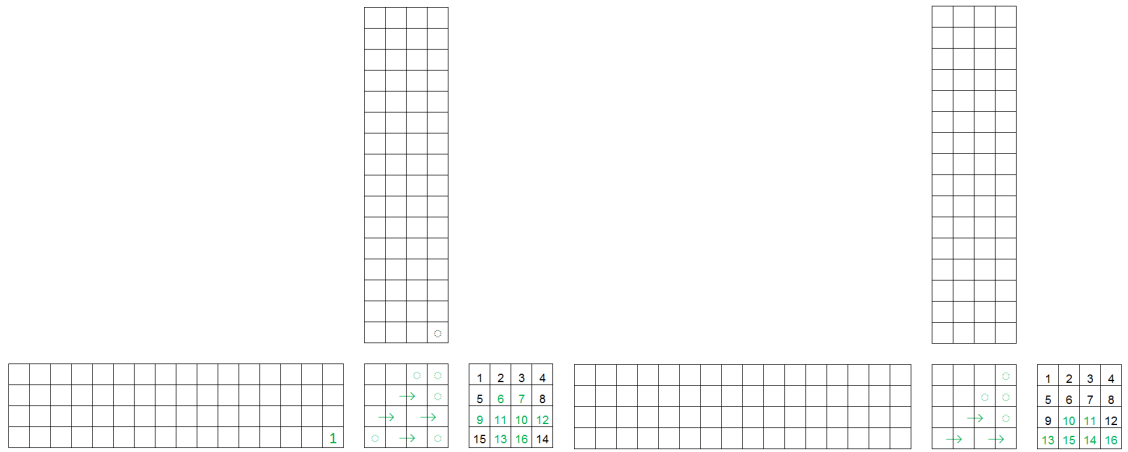
(11)

(12)



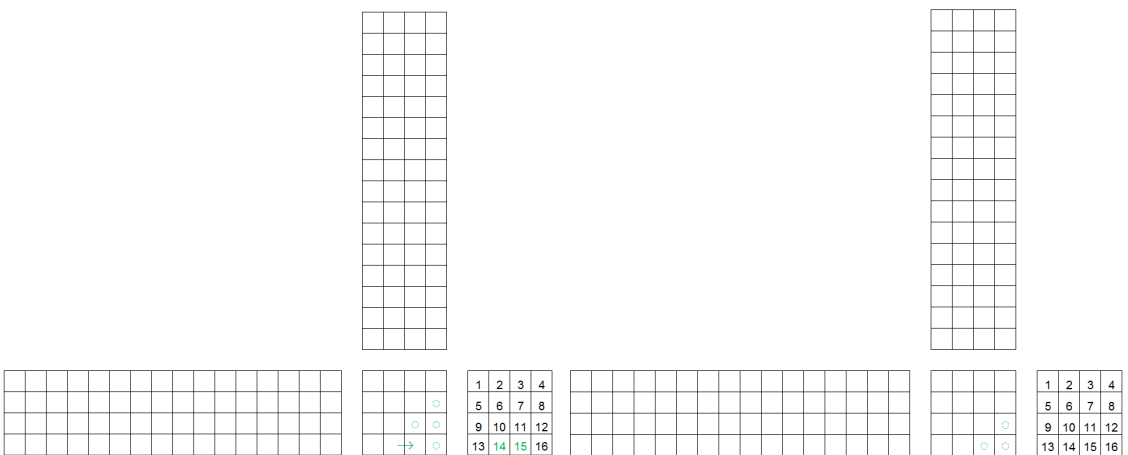
(13)

(14)



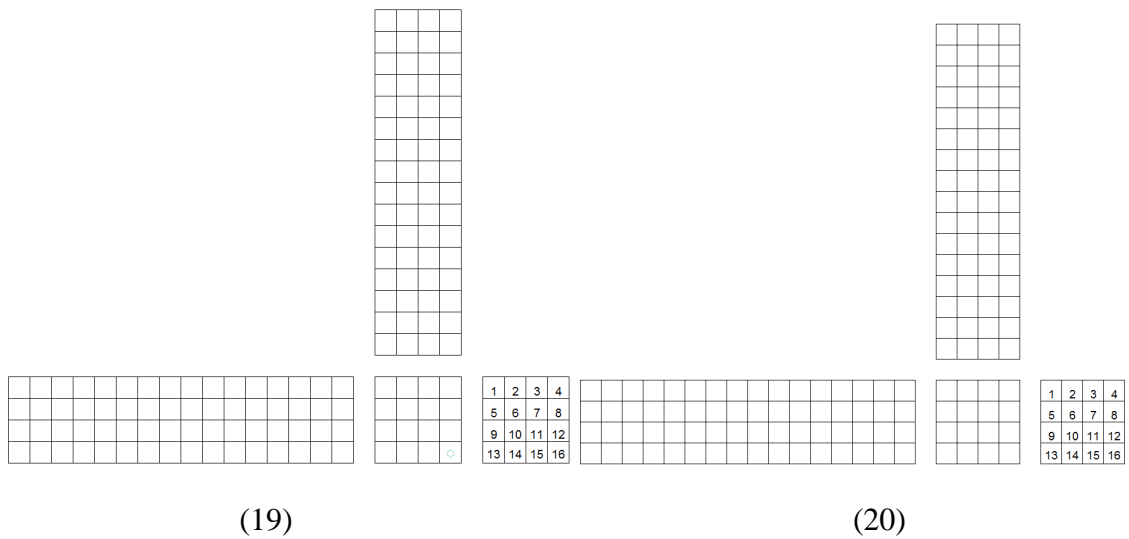
(15)

(16)



(17)

(18)



4.2.4 Result from the processor array

The above numerical example program was run on the array of microcontrollers. Fig. 4.3 shows the execution of the ISA program as the program runs. It indicates the time difference between the instruction received and the selector bit sent between the processing elements. Serial interface was used to individual controllers to interrogate the result. The processing element P(2,1) has more no operation instruction than the P(3,3) thus P(2,1) takes 29.95ms and P(3,3) takes 32.45ms to execute all the instructions. Results demonstrate that the application executes in 32.45ms on the prototype ISA implementation. This result is reasonably acceptable for some applications such as human movement measurement which tends to work at a low sampling rate. However it should be noted that there is a significant latency in this experimental setup which has not been optimised out. The processing element themselves are microcontrollers programmed in a high level language and a custom design would clearly be able to obtain very much better performance. Nevertheless, the merge algorithm application has been successfully implemented and validated on the prototype ISA using off-the-shelf microcontrollers.

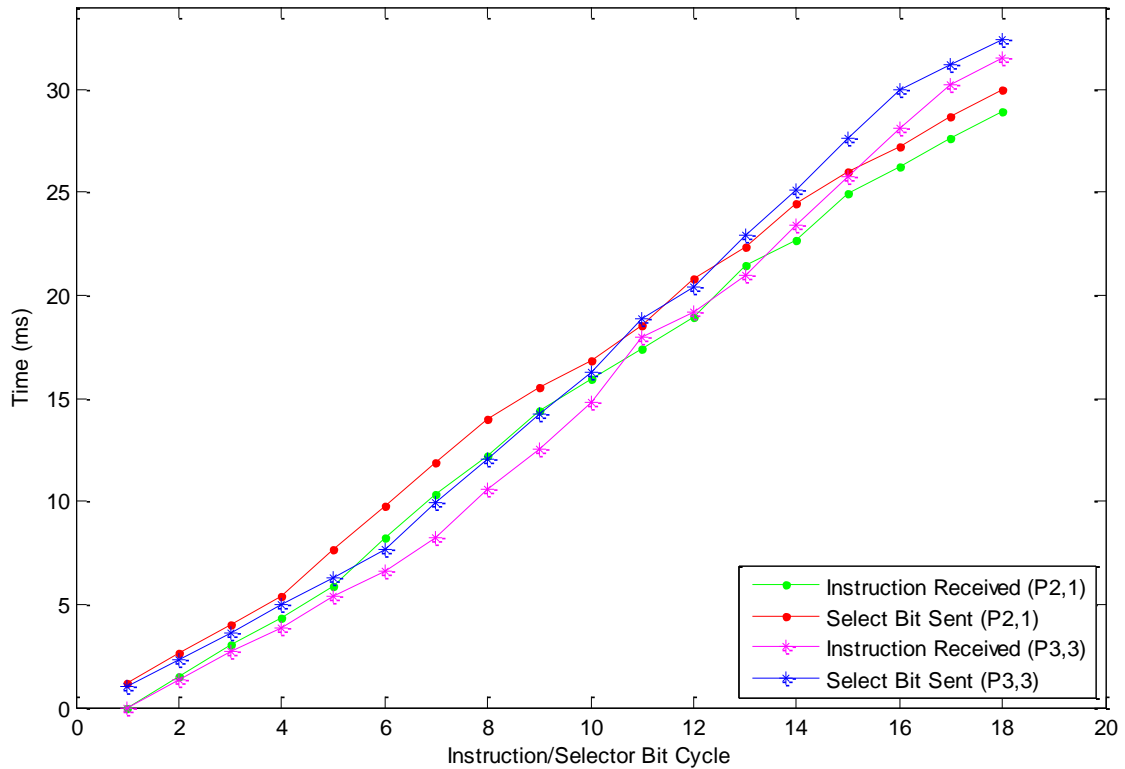


Figure 4.3: Performance analysis for P(2,1) and P(3,3)

4.3 Matrix Multiplication Validation

In numerical algebra, matrix multiplication plays a vital role because the product is calculated in various stages of many technical problems and almost in all numerical algorithms. Matrix multiplication is very standard calculation and goes well with parallel implementation. Matrix multiplication is suitable for instruction systolic array concept because of its design and nearest neighbour communication. In matrix multiplication algorithm, a network of processing elements is used to calculate rhythmically and pass the data through the system using instruction systolic array.

4.3.1 Algorithm

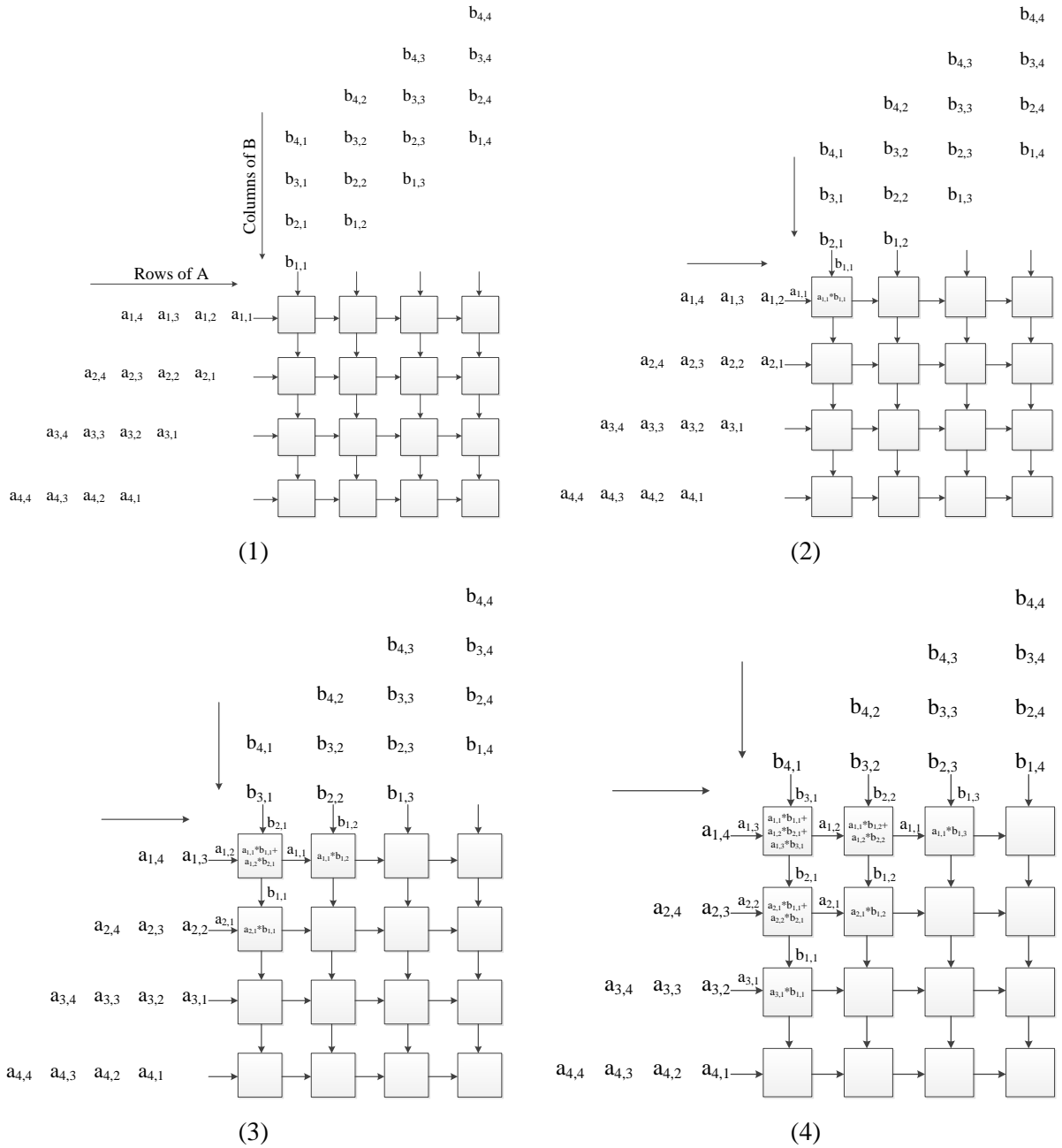
The standard algorithm for matrix multiplication is as follows from [4.3],

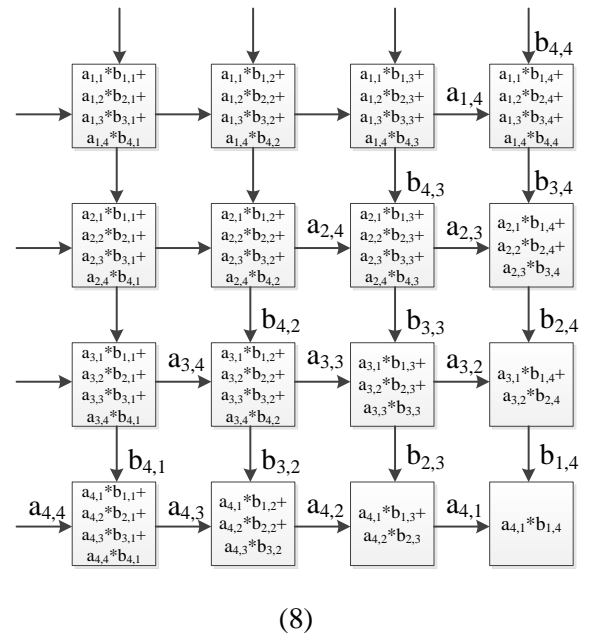
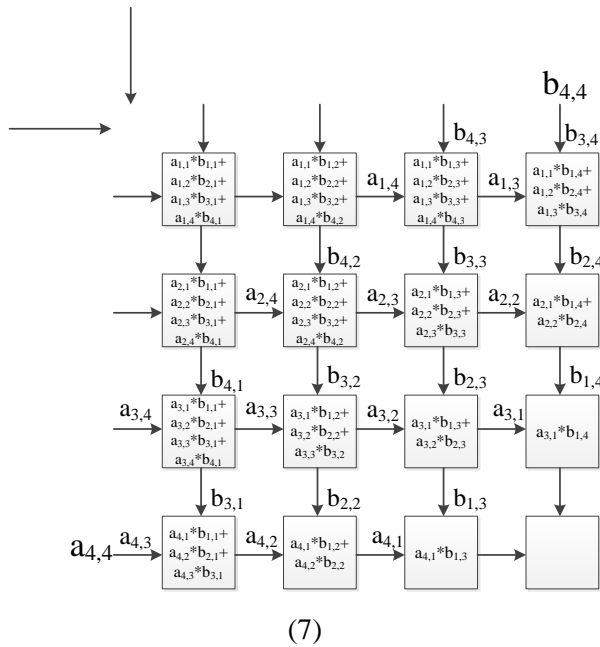
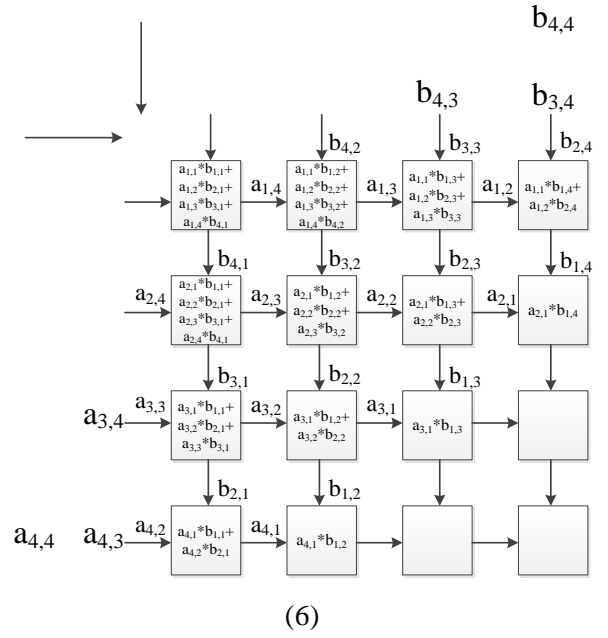
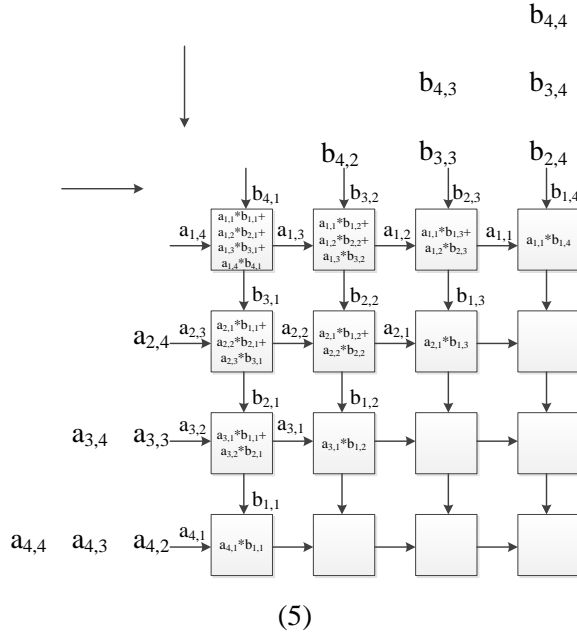
Step1: Each processing element accumulates one element of the product.

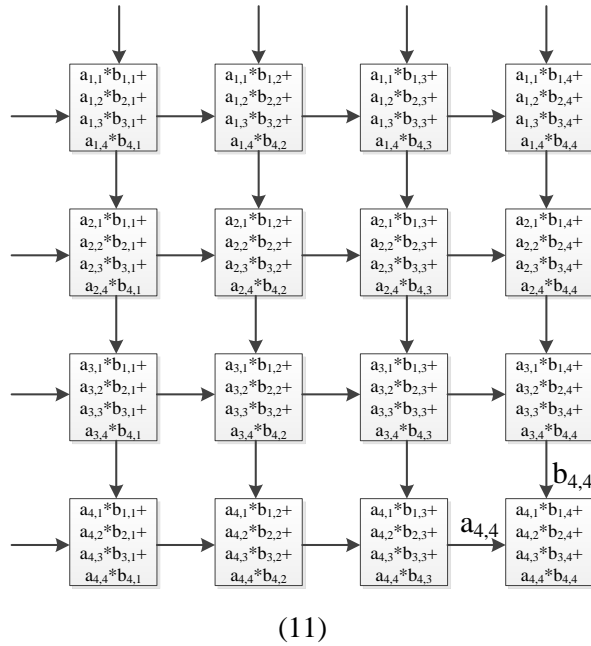
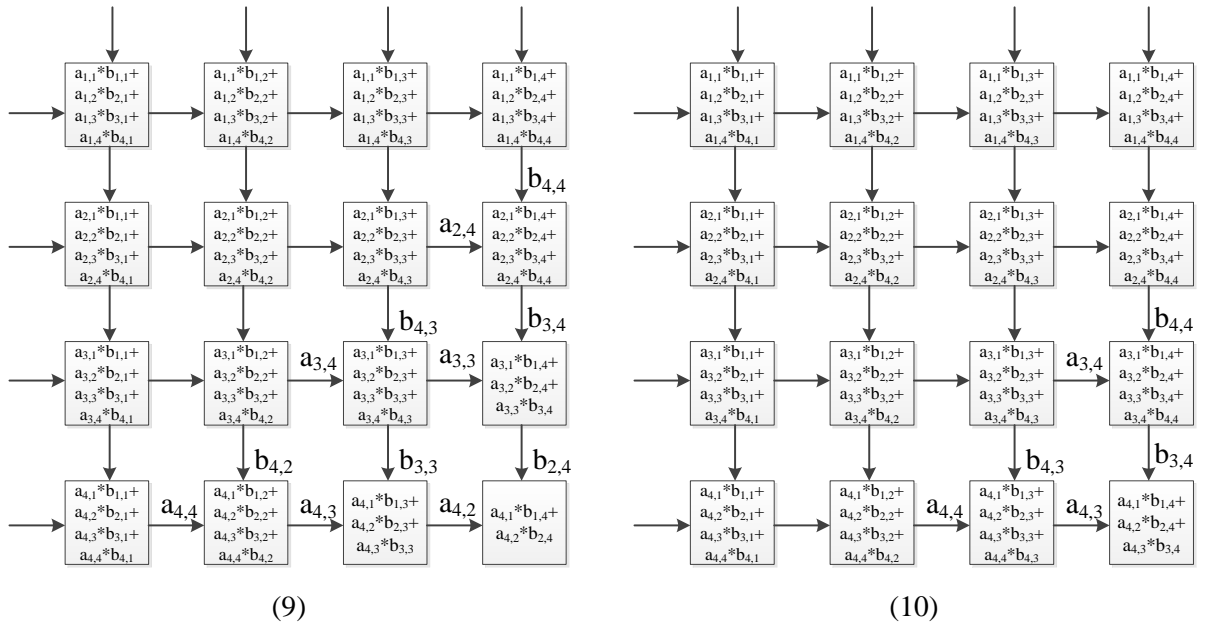
Step 2: This product is summed with the next element of product and accumulated in the processing element.

Step 3: After all the row and column instructions and selector bits are executed we get a 4×4 matrix result of Matrix A and Matrix B.

The following program shows the first iteration of an ISA program for multiplication of two $n \times n$ matrices. The first column of matrix A is input at the left border of the array, the first row of matrix B is input at the upper border.







The ISA program for matrix multiplication of two 4×4 matrices is presented in Fig. 4.4. The program shows the set of instruction and selector bits that will flow through the array. In Fig. 4.4, the instruction and selector bit part of the program are represented in a parallelogram shape made up of their corresponding instruction and selector bit

diagonals. Matrix A and B are the input matrices. As discussed in the second chapter, in this experiment the input matrices (data) are loaded into the processing elements through both instruction and selector bit array to reduce the number of execution cycle. A set of no operation instructions is flushed through the array before and after the instruction and selector bit diagonal. The matrix multiplication is scalable the number of instructions and selector bits will be increased according to the increase in the size of the array [4.2].

4.3.2 Program

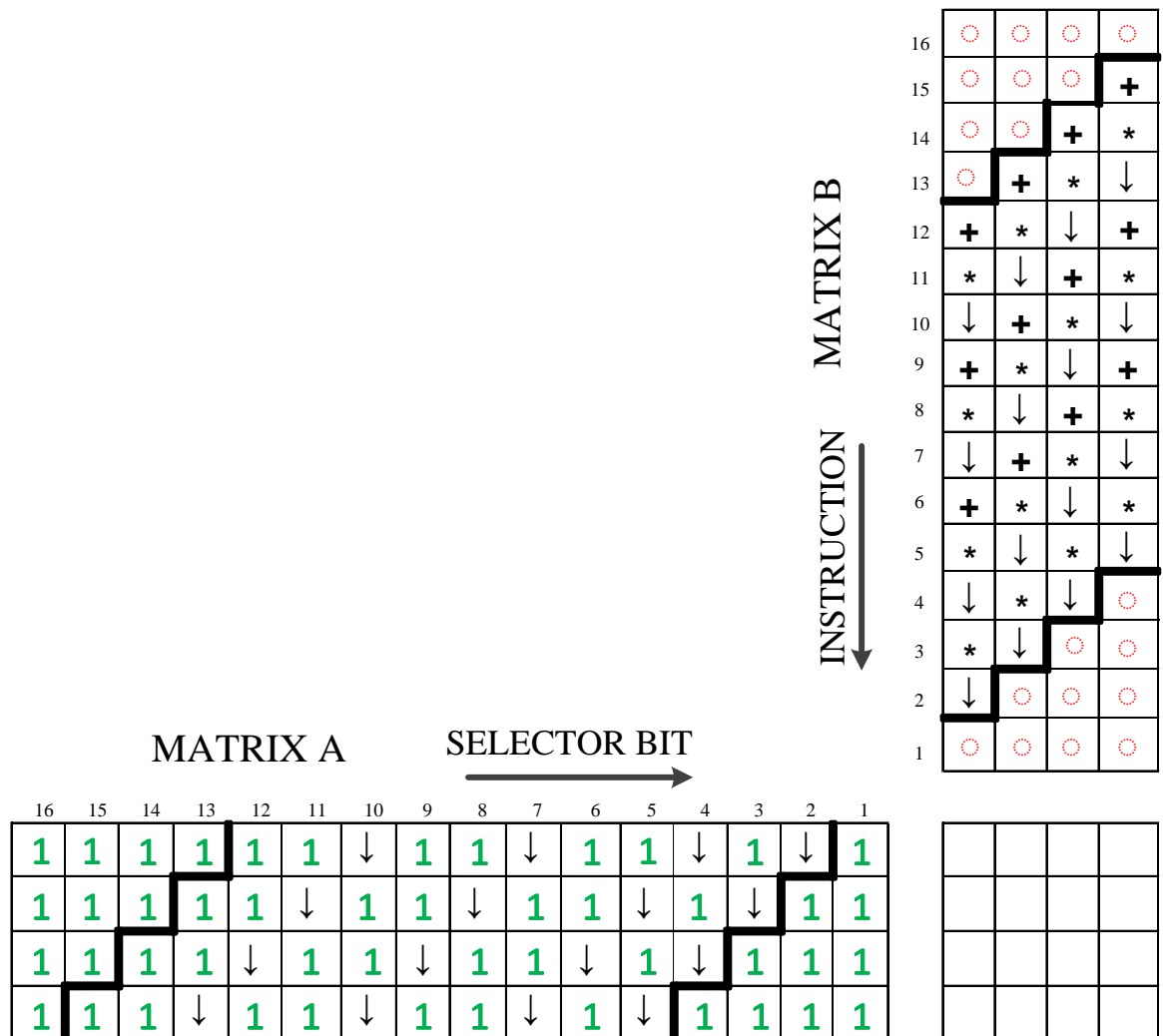





Figure 4.4: ISA Program for Matrix Multiplication

The meaning of the instruction symbols on Fig. 4.2 are illustrated below,

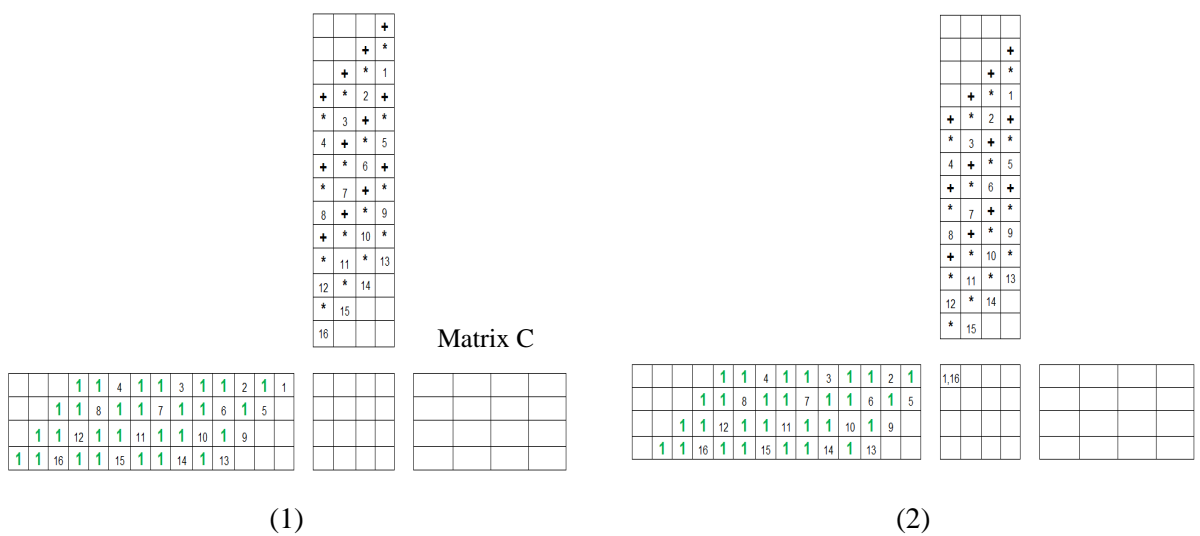
-  : Load Matrix
-  : Sum with most recent value
-  : Multiply with most recent value

4.3.3 Numerical example

When the input Matrix A and Matrix B is multiplied the resultant output Matrix C is obtained.

$$\begin{matrix} & \mathbf{Matrix\ A} & & \mathbf{Matrix\ B} & & \mathbf{Matrix\ C} \\ & \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} & \times & \begin{bmatrix} 16 & 15 & 14 & 13 \\ 12 & 11 & 10 & 9 \\ 8 & 7 & 6 & 5 \\ 4 & 3 & 2 & 1 \end{bmatrix} & = & \begin{bmatrix} 80 & 70 & 60 & 50 \\ 240 & 214 & 188 & 162 \\ 400 & 358 & 316 & 274 \\ 560 & 502 & 444 & 386 \end{bmatrix} \end{matrix}$$

The example below shows the step by step executions of the instructions along the array. The contents of Matrix C are shown after the instruction has been executed.



			+
		+	*
		+	* 1
	+	*	2 +
	*	3 +	*
4	+	*	5
+	*	6 +	+
*	7 +	*	
8	+	*	9
+	*	10 +	*
*	11 +	*	13
12	*	14	

Matrix C

			+
		+	*
		+	* 1
	+	*	2 +
	*	3 +	*
4	+	*	5
+	*	6 +	+
*	7 +	*	
8	+	*	9
+	*	10 +	*
*	11 +	*	13

				1	1	4	1	1	3	1	1	2
			1	1	8	1	1	7	1	1	6	1
		1	1	12	1	1	1	1	10	1	9	
	1	1	16	1	1	15	1	1	14	1	13	

*	1,15		
5,16			

16			
----	--	--	--

				1	1	4	1	1	3	1	1
			1	1	8	1	1	7	1	1	6
		1	1	12	1	1	1	1	10	1	9
	1	1	16	1	1	15	1	1	14	1	13

2,12	*	1,14	
*	5,15		
9,16			

16	15		
80			

(3)

(4)

			+
		+	*
		+	* 1
	+	*	2 +
	*	3 +	*
4	+	*	5
+	*	6 +	+
*	7 +	*	
8	+	*	9
+	*	10 +	*

			+
		+	*
		+	* 1
	+	*	2 +
	*	3 +	*
4	+	*	5
+	*	6 +	+
*	7 +	*	
8	+	*	9

				1	1	4	1	1	3	1	1
			1	1	8	1	1	7	1	1	6
		1	1	12	1	1	1	1	10	1	9
	1	1	16	1	1	15	1	1	14	1	13

*	2,11	*	1,13
6,12	*	5,14	
*	9,15		
13,16			

16,24	15	14	
80	75		
144			

				1	1	4	1	1	3	1	1
			1	1	8	1	1	7	1	1	6
		1	1	12	1	1	1	1	10	1	9
	1	1	16	1	1	15	1	1	14	1	13

+	*	2,10	*
*	6,11	*	5,13
10,12	*	9,14	
*	13,15		

40	15,22	14	13
80,72	75	70	
144	135		

(5)

(6)

			+
		+	*
		+	* 1
	+	*	2 +
	*	3 +	*
4	+	*	5
+	*	6 +	+
*	7 +	*	

			+
		+	*
		+	* 1
	+	*	2 +
	*	3 +	*
4	+	*	5
+	*	6 +	+

				1	1	4	1	1	1	1
			1	1	8	1	1	7	1	1
		1	1	12	1	1	1	1	1	1
	1	1	16	1	1	15	1	1	1	1

3,8	+	*	2,9
+	*	8,10	*
*	10,11	*	9,13
14,12	*	13,14	

40	37	14,20	13
152	75,66	70	65
144,120	135	126	
208	195		

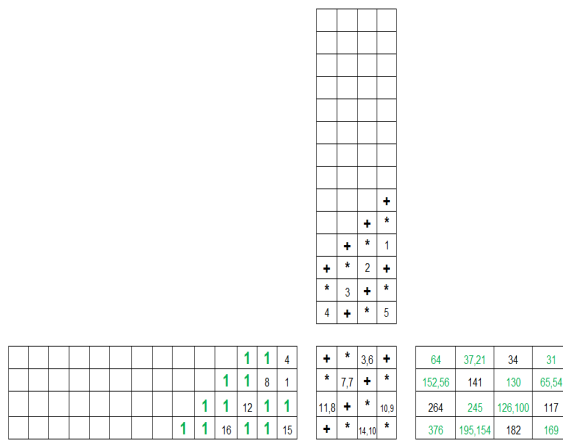
				1	1	4	1	1	1	1
			1	1	8	1	1	7	1	1
		1	1	12	1	1	1	1	1	1
	1	1	16	1	1	15	1	1	1	1

*	3,7	+	*
7,8	+	*	6,9
+	*	10,10	*
*	14,11	*	13,13

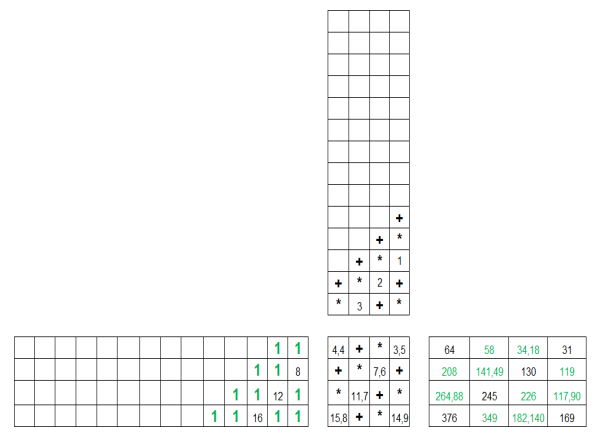
40,24	37	34	13,18
152	141	70,60	65
264	135,110	126	117
208,168	195	182	

(7)

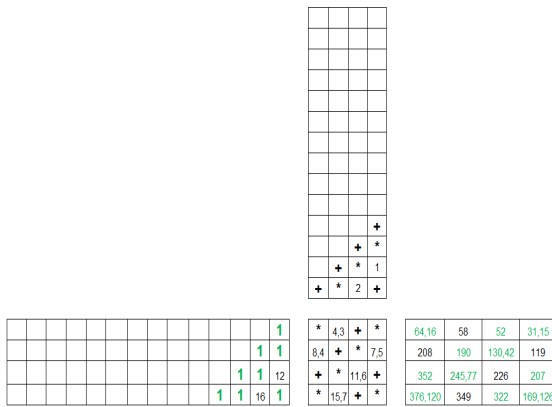
(8)



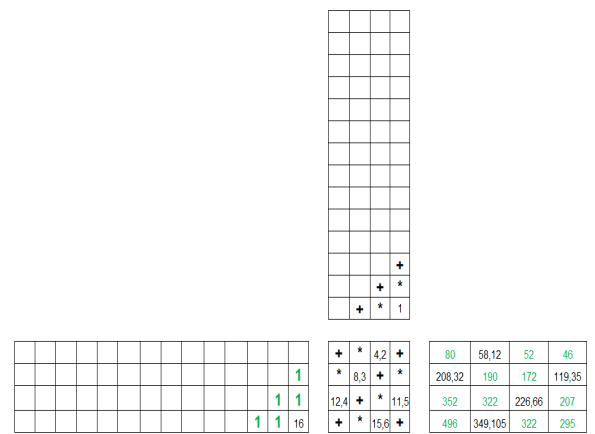
(9)



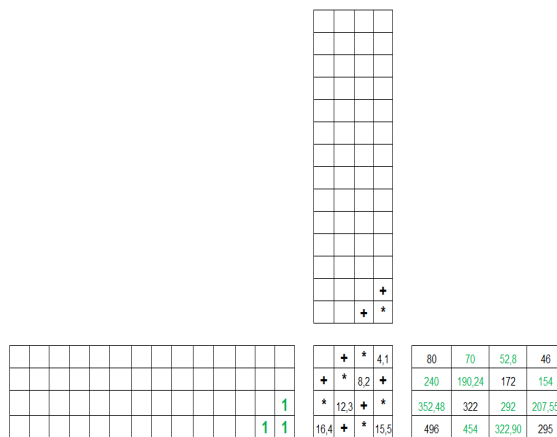
(10)



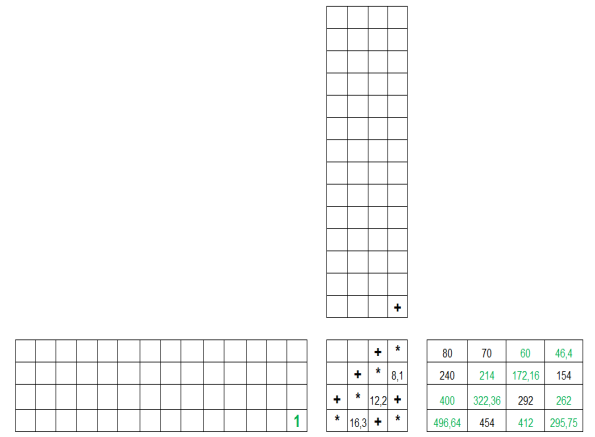
(11)



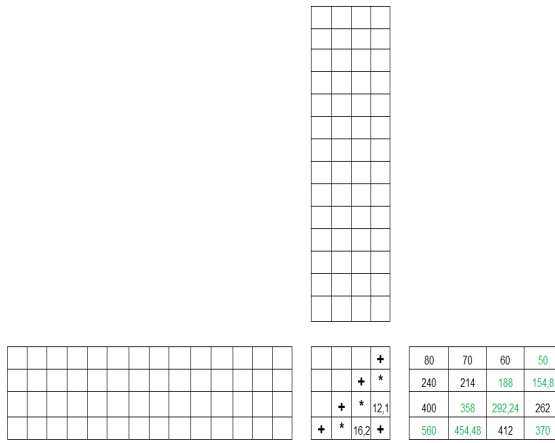
(12)



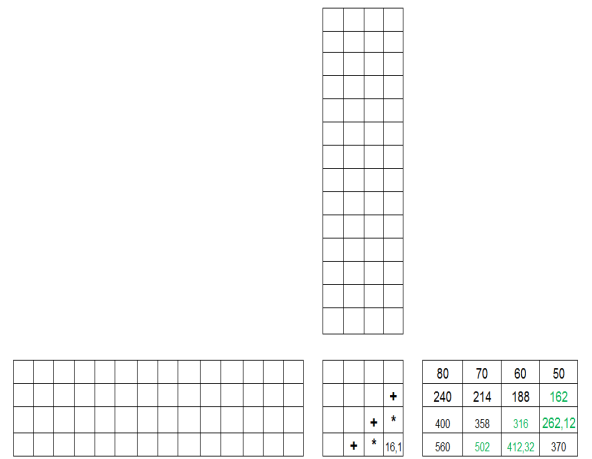
(13)



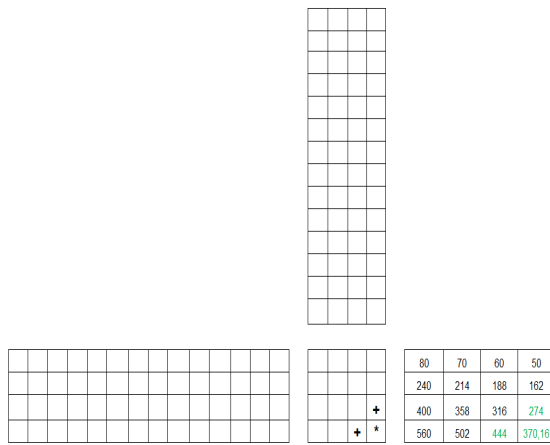
(14)



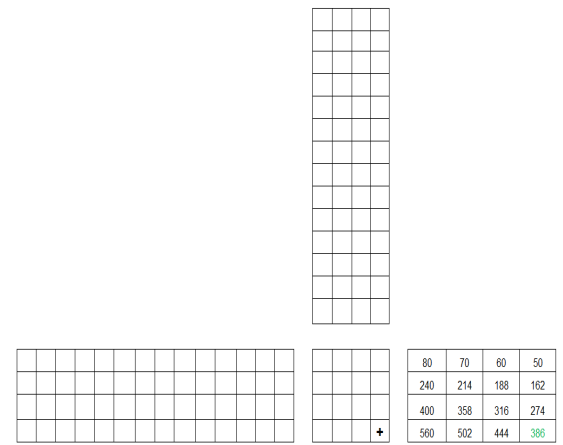
(15)



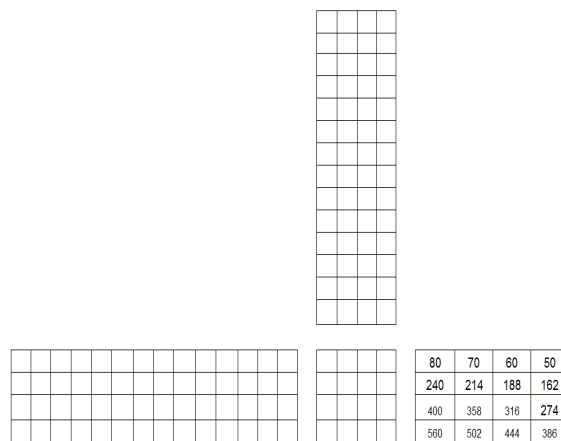
(16)



(17)



(18)



(19)

4.3.4 Result from the processor array

Fig. 4.5 shows the execution of the ISA program as the program runs. It indicates the time difference between the instruction received and the selector bit sent between the processing elements. The processing elements P(2,1) and P(3,3) has the same number of no operation instruction and thus they have completed the execution of all the instruction in the same time. Results demonstrate that the application executes in 30.95ms on our prototype ISA implementation. The processing element themselves are microcontrollers programmed in a high level language and a custom design would clearly be able to obtain very much better performance. Nevertheless, the matrix multiplication has been successfully implemented and validated on the prototype ISA using off-the-shelf microcontrollers.

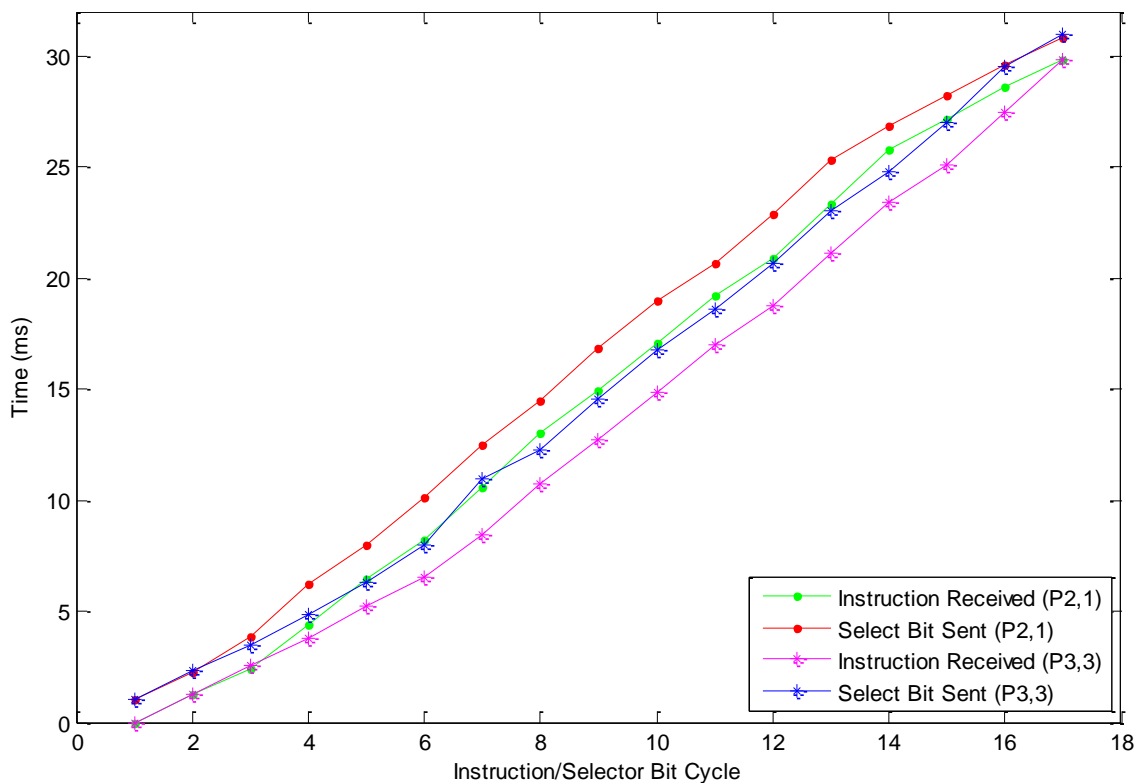


Figure 4.5: Performance analysis for P(2,1) and P(3,3)

4.4 Conclusion

The instruction systolic array has been successfully implemented on an array of off-the-shelf microcontrollers. Simple parallel algorithms have been validated using instruction

systolic array. The next chapter will use the same conventional method of programming to implement the instruction systolic array on a fabric and using a representative example of an application.

References

- [4.1] M. Kunde, H.-W. Lang, M. Schimmler, H. Schmeck, and H. Schröder, “The instruction systolic array and its relation to other models of parallel computers,” *Elsevier Parallel Computing*, vol. 7, no. 1, pp. 25–39, 1988
- [4.2] H. W. Lang, “The instruction systolic array - a parallel architecture for VLSI,” *Integration*, vol. 4, pp. 65–74, 1986, doi:10.1016/0167-9260(86)90038-6
- [4.3] B. Schmidt, “Techniques for Algorithm the Instruction Design on Systolic Array,” *Doctoral Thesis (PhD)*, Loughborough University, 1999
- [4.4] M. Kunde, H.-W. Lang, M. Schimmler, H. Schmeck, and H. Schröder, “The instruction systolic array and its relation to other models of parallel computers,” *Elsevier Parallel Computing*, vol. 7, no. 1, pp. 25–39, 1988
- [4.5] B. Schmidt, M. Schimmler and H. Schroder, “Morphological Hough Transform on the Instruction Systolic Array,” *In Practice*, 1997
- [4.6] R. Hughey and D. P. Lopresti, “Architecture of a Programmable Systolic Array,” pp. 41-49, 1988, doi:10.1109/ARRAYS.1988.18043

CHAPTER 5: SHAPE RECONSTRUCTION USING INSTRUCTION SYSTOLIC ARRAY

THIS chapter introduces a 2D mesh architecture prototype based on the Instruction systolic array paradigm for distributed computing on fabrics. A real-time shape sensing and reconstruction application executing on ISA architecture and demonstrates a physical design for a wearable system based on the ISA concept constructed from off-the-shelf microcontrollers and sensors.

5.1 Introduction

In the literature, few studies have been made to measure 3D shapes of an object using sensors wrapped around or mounted on the object itself [5.1]. One of the potential applications of shape sensing and reconstruction is the human posture sensing. Other application also includes 3D modelling of an object and wearable motion capture [5.2]. This data can be valuable in shape sensing applications such as real-time human posture and movement monitoring as well as shape feedback of flexible devices. A method is designed for applications in new emerging fields, such as smart textile and flexible electronics, where it can be used to obtain wearers posture or shape of the device [5.2]. The shape of an object can be determined by acquiring an object's 3D geometric properties. Real time measurements of the object provide continuous deformations of the shape of the object. Therefore, shape sensing applications use such data to reconstruct the shape of an object. The fabric conforms reasonably well to the human body, particularly in sports where fitted garments are common. This measurement of the fabric can give a fairly accurate idea of the shape of the human body that it is worn on.

Low-cost miniature sensors using MEMS (Micro Electro Mechanical Systems) technologies have become increasingly common in recent years. These sensors are integrated into fabrics to obtain the local data which helps in getting global shape characteristics. In order to generate a 3D model of an object, two reference directions

are required. One of them is gravity measured using the accelerometer and other one is earth's magnetic field measured using magnetic sensor. Three-axis accelerometer and magnet sensor grid is used to generate shape reconstruction of the object [5.1].

The accelerometer and magnetic sensors provide only two vector observations, which are the minimum for full orientation determination, no minimization problem can be defined [5.1]. Therefore, Hermanis et al. [5.1] proposes a triad based shape reconstruction algorithm three axis accelerometer and magnetic sensor grid.

5.2 Background

Based on Hermanis et al. in the shape reconstruction algorithm, the sensor nodes are embedded into the fabric to measure local orientation data. The shape reconstruction algorithm from Hermanis et al. along with the instruction systolic array for global shape reconstruction from local orientation measurements ensures fast computations for shape reconstruction utilizing data from a number of sensors. To implement the shape reconstruction algorithm with ISA concept, the peripheral devices acceleration and magnetic sensors are arranged in a regular grid along the fabric and each sensor is connected to their respective microcontrollers. The following subsections explain the method and equations proposed by Hermanis et al. which are used to estimate the orientation shape of the object. The same method will then be used later in this thesis with ISA to reconstruct the shape of the object.

5.2.1 Shape Reconstruction algorithm

To calculate the orientation of an object, various algorithms are proposed. Any problem related to calculating the orientation is normally termed as Wahba's problem [5.3]. To get a solution, consider Rotational matrix (R) by minimization of following expression [5.4]:

$$\sum_{k=1}^K \|v_k^* - Rv_k\|^2 \quad (5.1)$$

where $\{v_1, v_2, \dots, v_k\}$ and $\{v_1^*, v_2^*, \dots, v_k^*\}$ are sets of K vector observations respectively in object frame and general reference frame. Thus to calculate orientation estimation of an object two triads are formed from the unit vectors, one of the triad is

formed from general reference frame and the other triad is formed from the sensor reference frame through the sensor measurements. The triads of the earth reference frame and the sensor reference frame are constructed from the earth gravity field vector E_g , magnetic field vector E_m , sensor measurement of gravity field vector S_g and sensor measurement of magnetic field vector S_m .

$$e_1 = E_g \quad (5.2)$$

$$e_2 = \frac{E_g \times E_m}{|E_g \times E_m|} \quad (5.3)$$

$$e_3 = e_1 \times e_2 \quad (5.4)$$

$$s_1 = S_g \quad (5.5)$$

$$s_2 = \frac{S_g \times S_m}{|S_g \times S_m|} \quad (5.6)$$

$$s_3 = s_1 \times s_2 \quad (5.7)$$

These triads are then used to form a matrix for global Earth reference, represented as M_e

$$M_e = [e_1 e_2 e_3] \quad (5.8)$$

and matrix for sensor measurements, represented as M_s

$$M_s = [s_1 s_2 s_3] \quad (5.9)$$

The rotation matrix R is then calculated by sensor orientation relative to the global reference frame and is calculated using the formula,

$$R = M_e M_s^T \quad (5.10)$$

Now, surface segment orientation relative to initial position can be calculated using the rotation matrix R.

5.2.2 Shape Reconstruction from sensor orientation data

As shown in Fig. 5.1, acceleration and magnetic sensor nodes are arranged along the surface in form of a regular grid. The model of the surface is divided into n rigid segments, where n is the total number of sensors used and is represented as

$$n = i \cdot j$$

i and j denote row and column of sensor location in the grid. The segment structure corresponds to sensor grid structure. Each segment is defined by segment center point $C[i, j]$ and four direction vectors, represented as $\vec{N}[i, j]$, $\vec{E}[i, j]$, $\vec{S}[i, j]$ and $\vec{W}[i, j]$. The surface geometry is described using the segment center points, which are surface control points. In the beginning, all segments are aligned with global reference system by assigning some base direction vector values like:

$$\begin{aligned}\vec{N}_b &= \left[0; 0; \frac{L_1}{2}\right] \\ \vec{E}_b &= \left[\frac{L_2}{2}; 0; 0\right]\end{aligned}\tag{5.11}$$

$$\begin{aligned}\vec{S}_b &= \left[0; 0; -\frac{L_1}{2}\right] = -\vec{N}_b \\ \vec{W}_b &= \left[-\frac{L_2}{2}; 0; 0\right] = -\vec{E}_b\end{aligned}\tag{5.12}$$

where L_1 and L_2 is the distance between sensors across in the array. The structure of surface model is shown in Fig. 5.2. The base direction vectors of each segment are calculated by including segment direction vectors. The segment orientation is calculated using the following expression:

$$\vec{N}[i, j] = R_{ij}\vec{N}_b$$

$$\vec{E}[i, j] = R_{ij} \vec{E}_b \quad (5.13)$$

All other direction vectors can be calculated using formulas opposite to equation (5.13):

$$\begin{aligned} \vec{S}[i, j] &= -\vec{N}[i, j] \\ \vec{W}[i, j] &= -\vec{E}[i, j] \end{aligned} \quad (5.14)$$

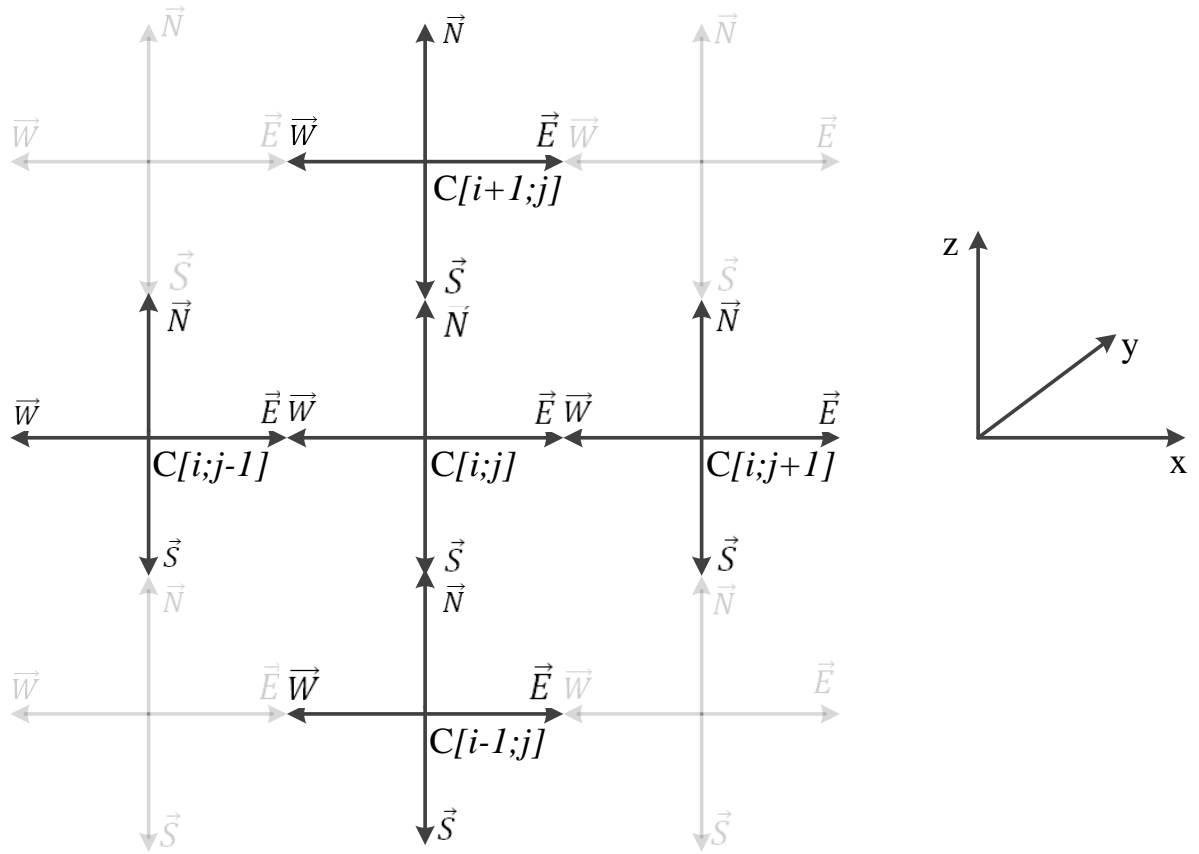


Figure 5.1: Surface segment structure. Each segment consists of center C and four direction vectors \vec{N} , \vec{E} , \vec{S} and \vec{W} [5.1]

If a single control point location is known, then all other control point on the same segment row or column can be calculated by adding and subtracting the corresponding segment direction vectors as can be seen in the Fig. 5.1. Any arbitrary sensor in i_{ref} row

and j_{ref} column can be assumed as reference by assigning some constant value to $C[i_{ref}; j_{ref}]$

Control points on the reference column can be calculated from the following expression:

$$C[i; j_{ref}] = C[i_{ref}; j_{ref}] + \sum_{k=i}^{i_{ref}-1} (-\vec{N}[k, j_{ref}] + \vec{S}[k+1, j_{ref}])$$

if($i < i_{ref}$) (5.15)

Similarly control points on the reference row ($i = i_{ref}$) can be calculated as:

$$C[i_{ref}; j] = C[i_{ref}; j_{ref}] + \sum_{k=j_{ref}}^{j-1} (\vec{E}[i_{ref}, k] - \vec{W}[i_{ref}, k+1])$$

if($j > j_{ref}$) (5.16)

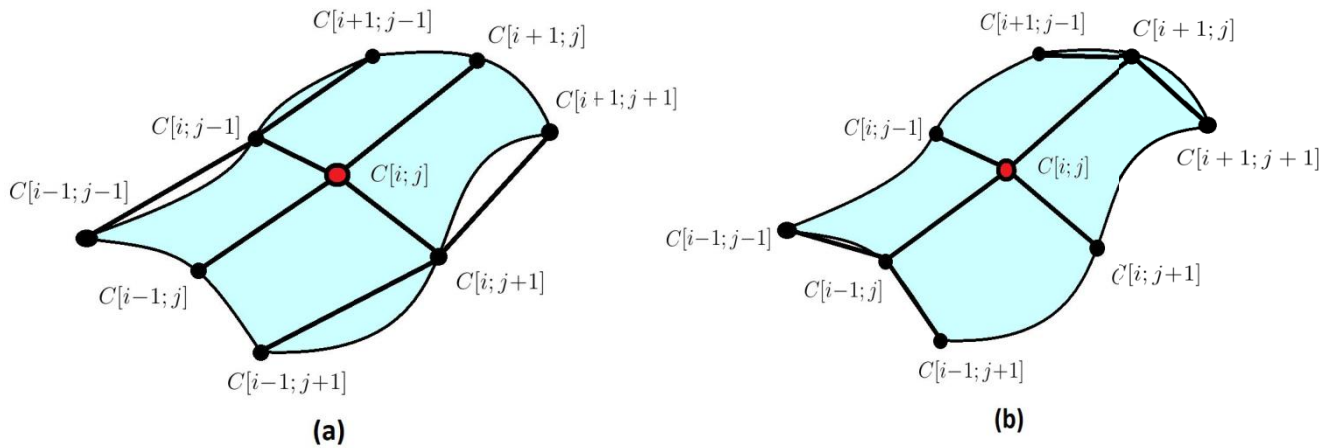


Figure 5.2: Structure of control point connections. $C[i; j]$ - reference point. (a) Single reference row is obtained, then all other points are calculated with column method. (b) Single reference column is obtained, then all other points are calculated with row method adapted from [5.1]

Once the first row and column are calculated, one control point from each row and column will be known in the grid. Using this known control point as a reference in either row or column the unknown control points can be calculated. As per theory, both the ways should give the same result but there are chances for the results to change because of the chosen connection path for the calculation of the control points. The control point recovery uses the bilateral process as explained below to avoid this problem [5.1].

First of all, as shown in Fig. 5.2(a), each segment centre coordinate is calculated from the reference by finding one reference row with equations (5.18) and (5.19) and then connecting other segment direction vectors long ways using (5.16) and (5.17). In the same way, all control points are obtained again by obtaining one reference column with equations (5.16) and (5.17) and then connecting segments across using (5.18) and (5.19) as per structure is shown in Fig. 5.2(b). Finally, results from both cases are averaged and the control points are calculated.

The Step wise implementation of Shape Reconstruction Application is explained in the flowchart below.

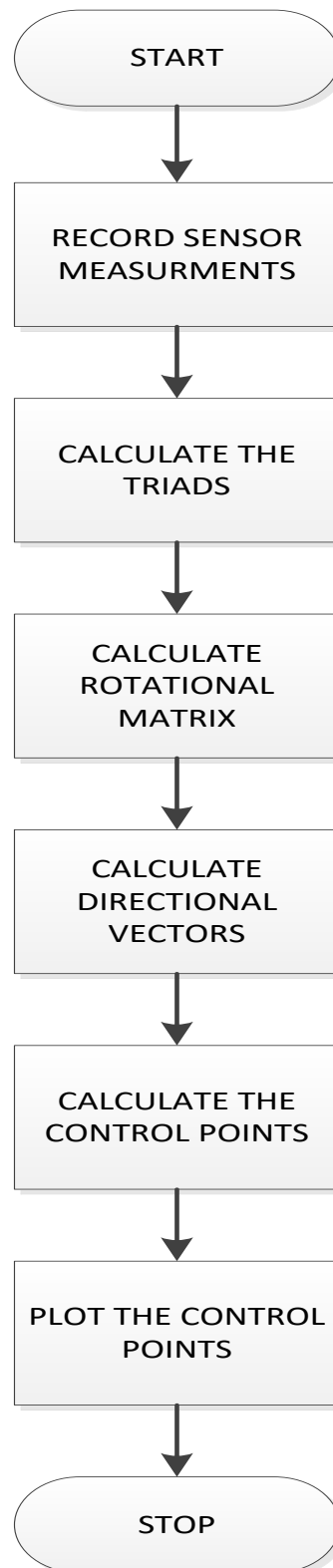


Figure 5.3: Step wise implementation of Shape Reconstruction Application

5.3 Experimental Setup

The concept prototype was designed to demonstrate and to confirm the viability of the proposed architecture for fabric-resident computing devices.

To implement the surface reconstruction application using ISA, a sensor network with 16 sensors was stitched into a 35cm × 35cm fabric swatch. Both inter-node communication and sensor connection in the prototype was achieved via I²C buses provided by the microcontroller. The sensors were of type LSM303DLHC acceleration/magnetic sensor [5.5] as shown in Fig.5.4. The LSM303DLHC is used for orientation estimation. The microcontroller serves as the interface between the sensor node and the host computer as all the computations take place locally in the microcontrollers. Each microcontroller is assigned a unique ID to identify its position in the grid and calculate the control points. Once the microcontroller receives the ID, it starts to receive the orientation data from the sensor. The orientation data is then averaged and stored for calculation of directional vectors. These directional vectors are shared between neighbouring microcontrollers for the calculation of control points. Once these control points are calculated for each sensor, they are sent to the host computer via serial port for 3D visualisation of the sensed object. The process of ISA computing the control points and the host drawing the visualisation continues indefinitely.



Figure 5.4: LSM303DLHC mounted on Adafruit board

A network of 16 sensor nodes was experimentally tested. Sensors were arranged in 4×4 grid formation and sewed on the layer of fabric with mutual distances 8.5 cm between each other as shown in Fig. 5.5.

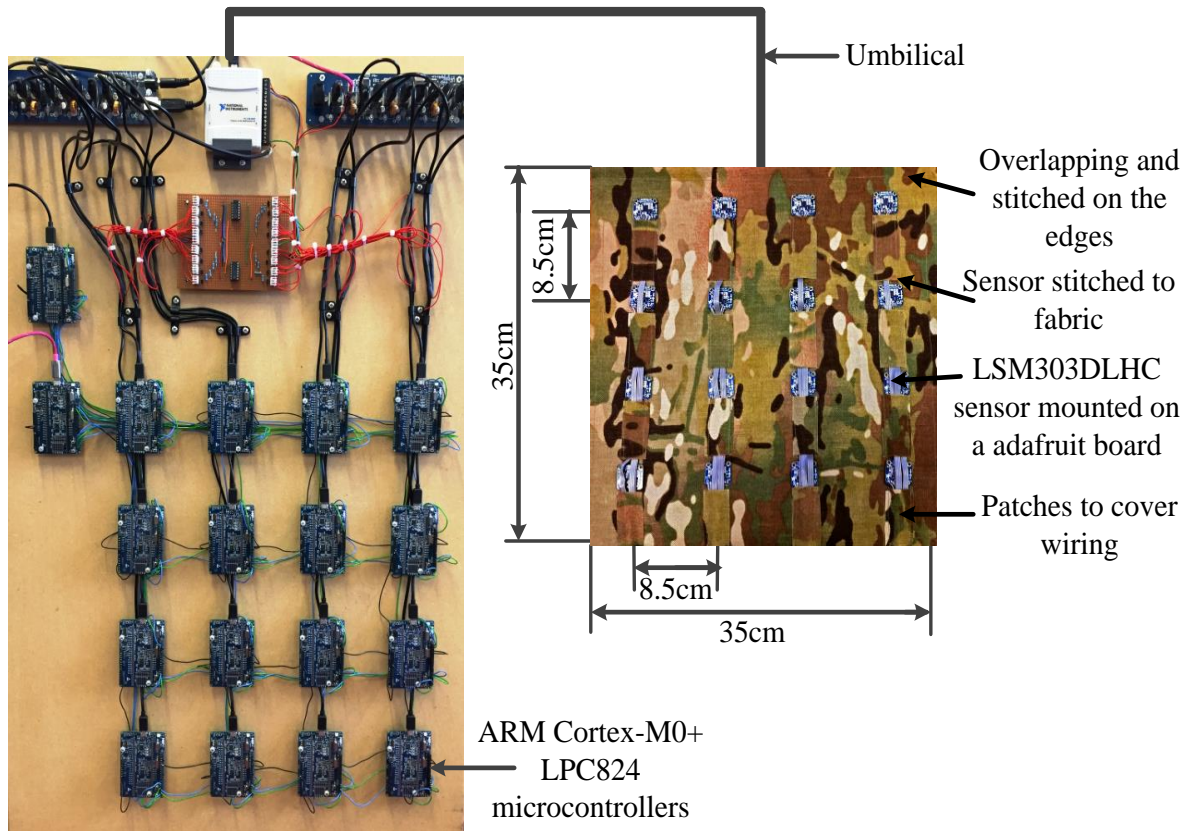


Figure 5.5: Sensors embedded with fabric.

5.4 Programming the shape reconstruction algorithm using Instruction systolic array

The ISA firmware for shape reconstruction application is explained in Fig. 5.5. The instruction and selector bits are passed to the processing element; the latter is in the *listening* mode waiting for the frame bytes to be received from the north and west slave I²C ports. Once both the instruction and selector bit are received they are then decoded and executed through an interrupt service routine. After the execution, the instruction and the selector bit are then forwarded to the neighbours through the south and east master I²C ports.

The steps below explain ISA firmware,

Step 1: The control points are calculated from the shape reconstruction algorithm which will define the surface geometry.

Step 2: From the segment structure it can be deduced that if a single control point location is known, then any other control point on the same segment row or column can be calculated by adding or subtracting the corresponding segment direction vectors.

Step 3: The calculated control points are then sent to the host computer for visualisation in 3D defining the shape of the sensed object.

The instruction and selector bit of the ISA firmware for shape reconstruction application (Fig. 5.7) can be seen as parallelogram shaped consisting of instructions and selector bits respectively. This diagonal of instruction and their corresponding selector bit is used for implementing the shape reconstruction application. A set of no operation instructions is flushed through the array before and after the instruction and selector bit diagonal.

The Directional vector \vec{D} as shown in the instruction set is used to calculate the directional vector from the equation 5.13 and 5.14. Once the directional vector is calculated the directional vector of each processing elements is shared with their neighbours to calculate the control points. The sharing of the directional vector is done by using swapping instructions as shown on merge algorithm in the previous chapter. Once all the directional vectors are shared with their neighbours the control points are then calculated by the instruction Σ . The instruction Σ implements the equation 5.15 and 5.16 and calculates the control points. Once the control points are calculated they are then sent to the host computer using the Tx instruction.

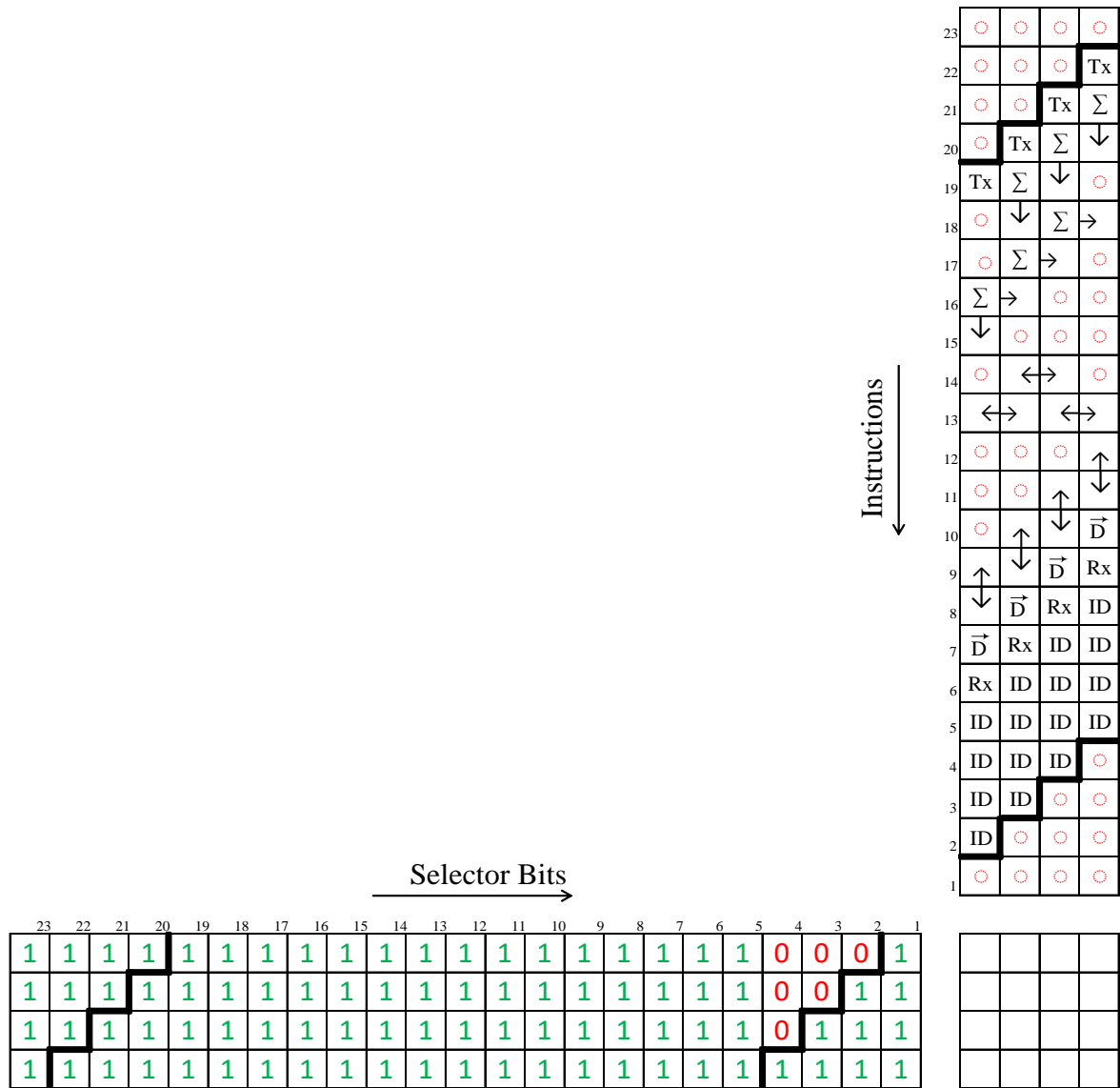


Figure 5.6: ISA firmware for shape reconstruction application

The meaning of the instruction symbols on Fig. 5.6 are illustrated as follows,

- | |
|----|
| ID |
|----|

 : Assign Identification number to controller
- | |
|----|
| Rx |
|----|

 : Receive data from sensor
- | |
|-----------|
| \vec{D} |
|-----------|

 : Calculation of Directional vector
- | |
|--------|
| ↑
↓ |
|--------|

 : Swap data between North-South ports
- | |
|--------|
| ←
→ |
|--------|

 : Swap data between East-West ports
- | |
|---|
| → |
|---|

 : Send data to east port
- | |
|---|
| ↓ |
|---|

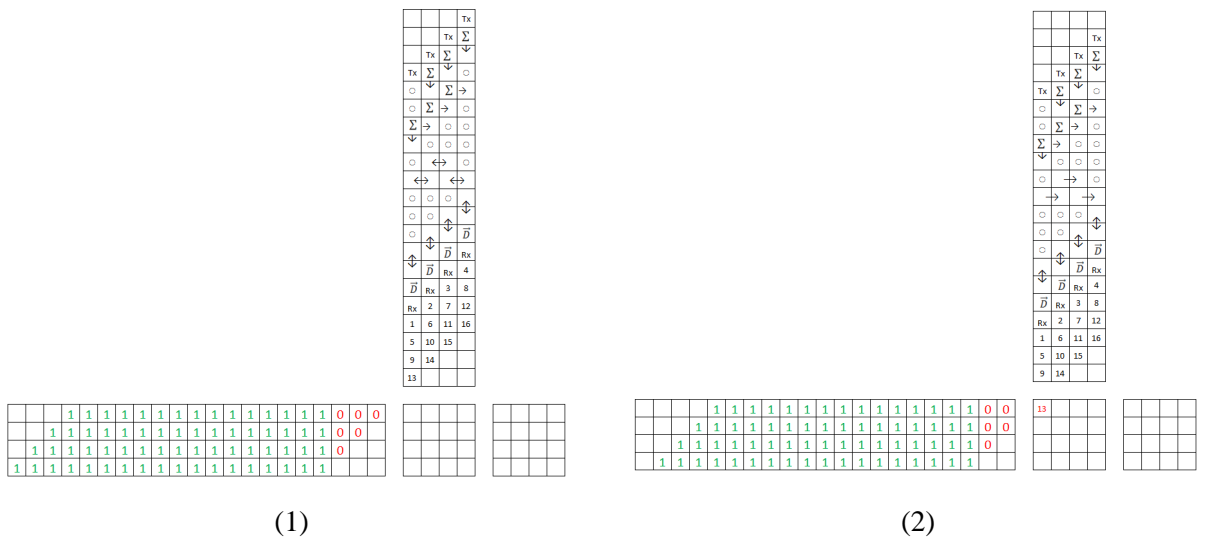
 : Send data to west port
- | |
|----------|
| Σ |
|----------|

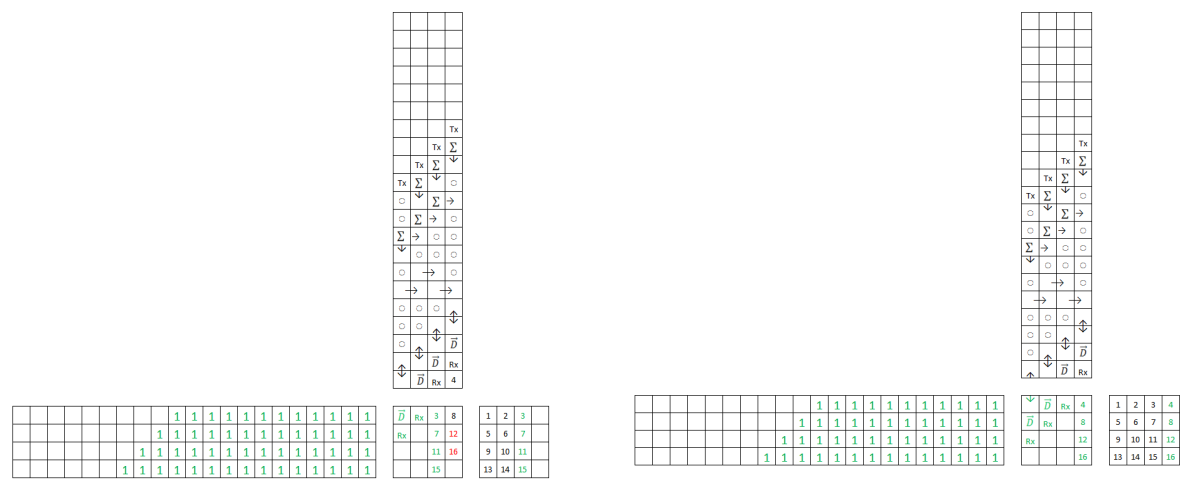
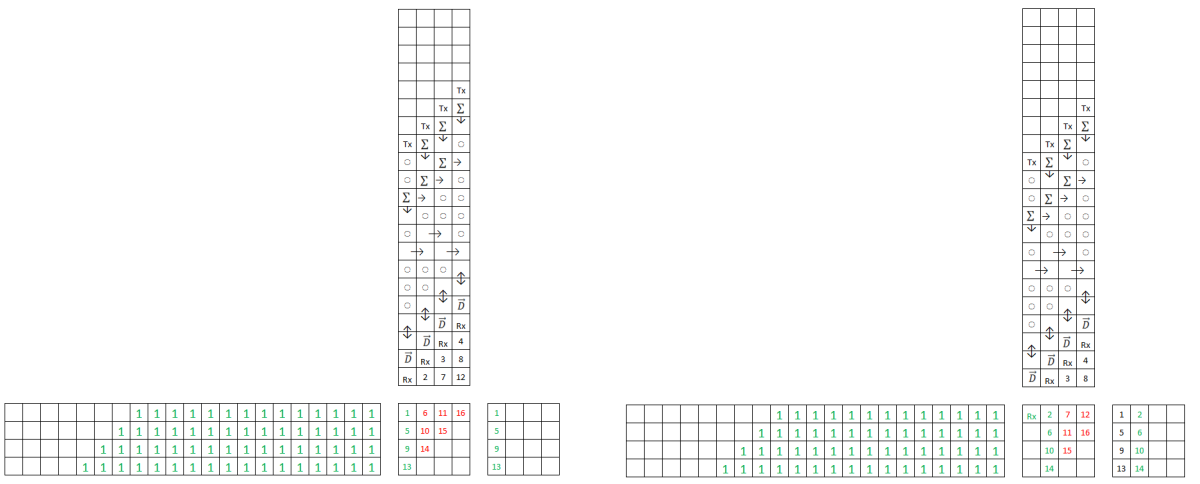
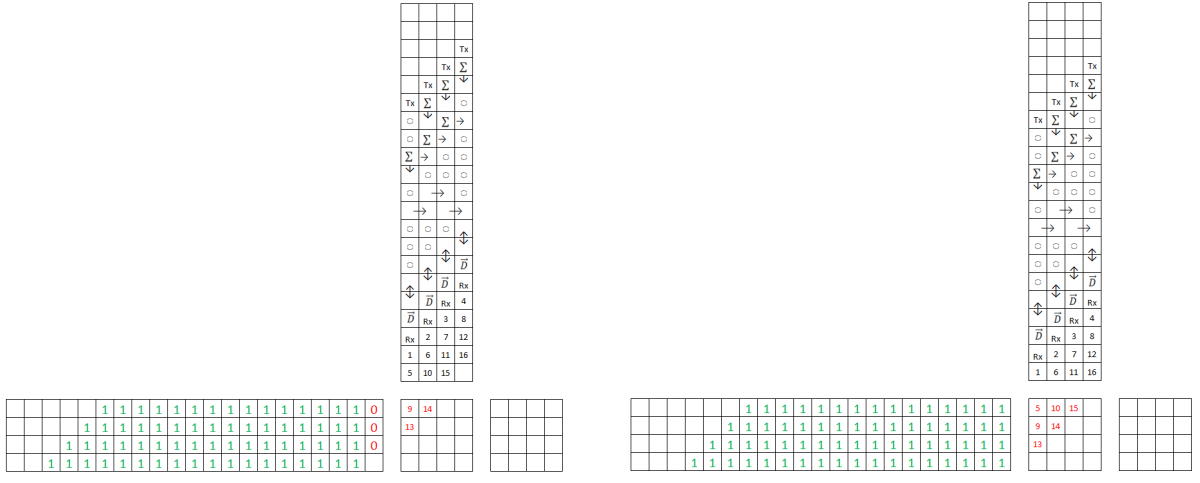
 : Calculate control points
- | |
|----|
| Tx |
|----|

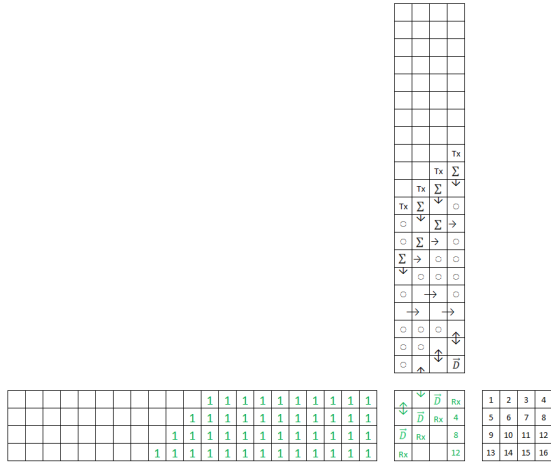
 : Send control points to host computer
- | |
|---|
| ○ |
|---|

 : No Operation

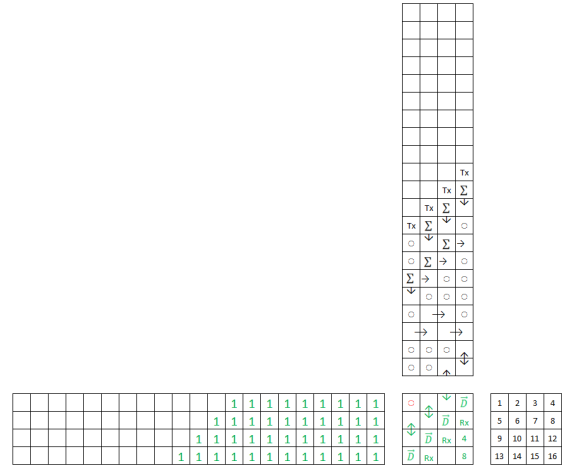
The execution of the ISA program for the Shape Reconstruction application using ISA is as follows. The execution shows only the ISA program diagonal of instructions and selector bits, it does not include the no operation that flows before and after the ISA program diagonal.



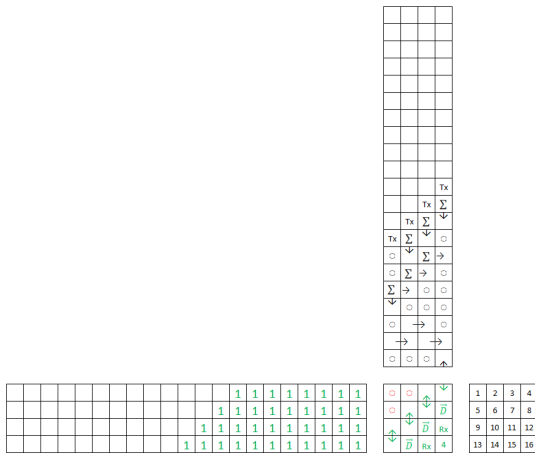




(9)



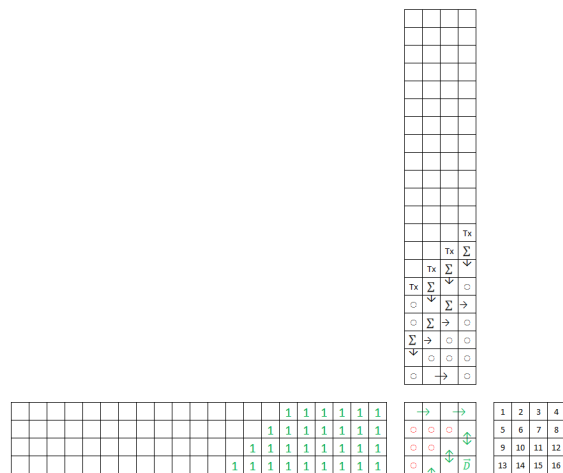
(10)



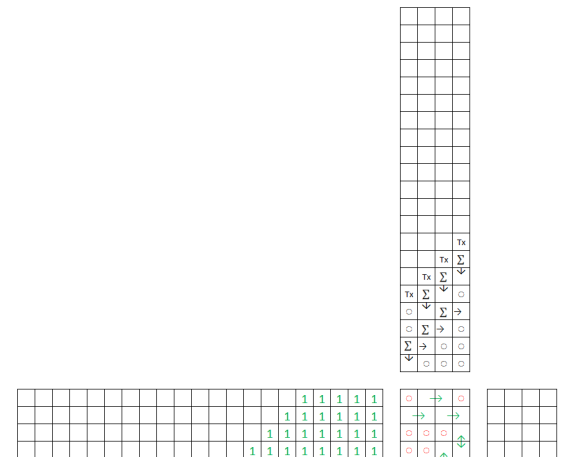
(11)



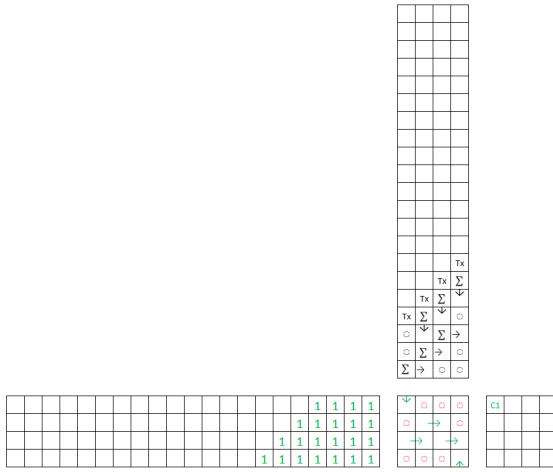
(12)



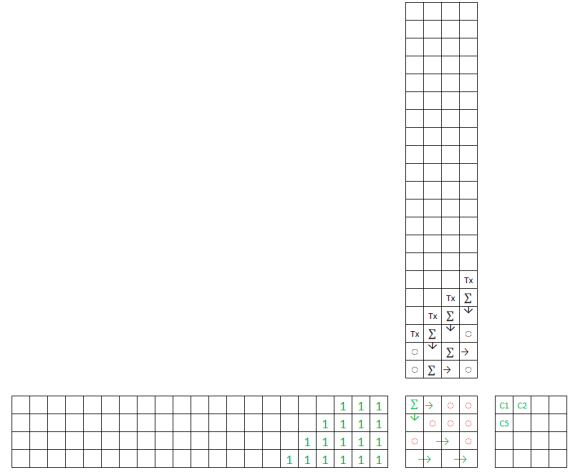
(13)



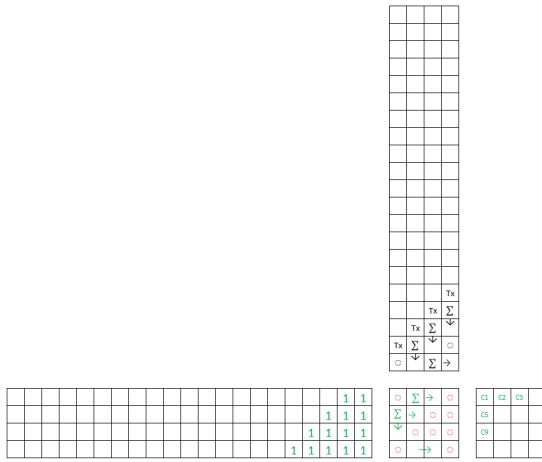
(14)



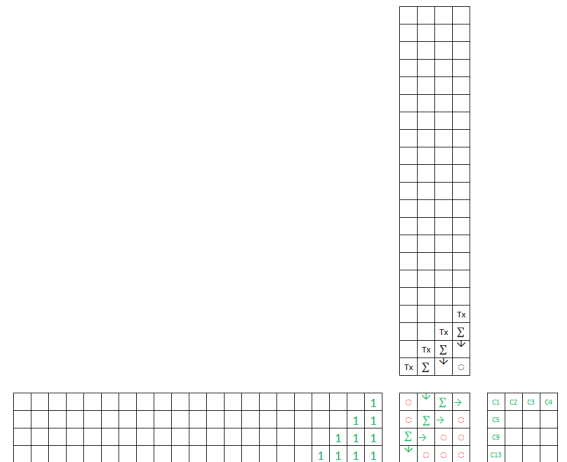
(15)



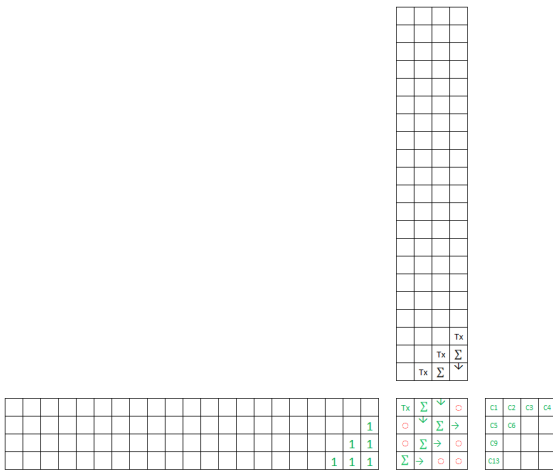
(16)



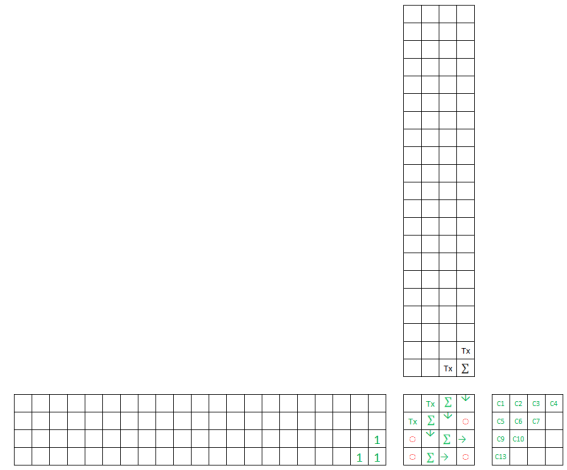
(17)



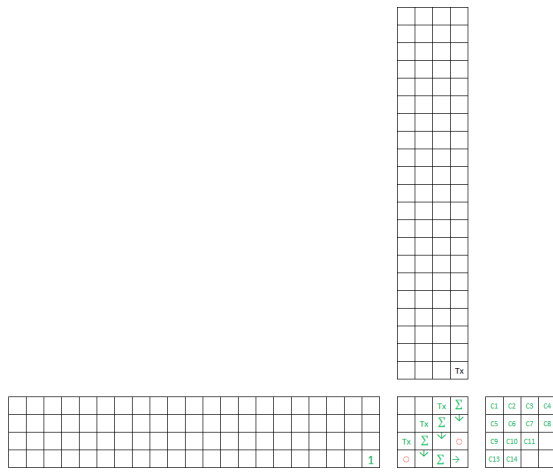
(18)



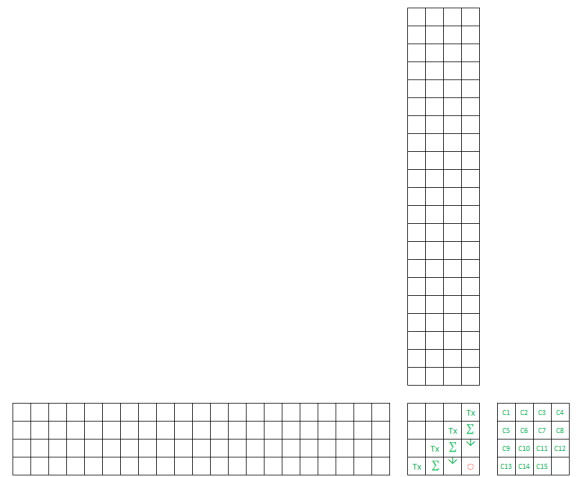
(19)



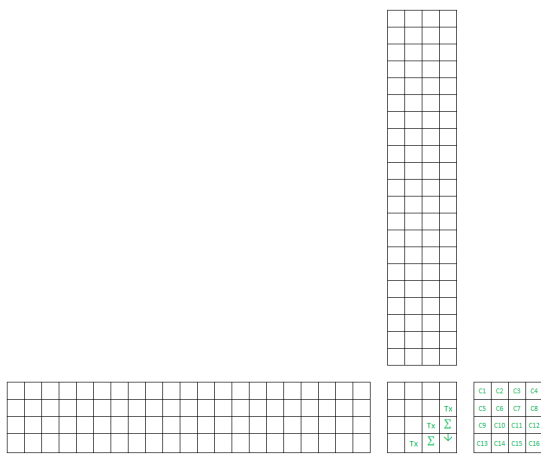
(20)



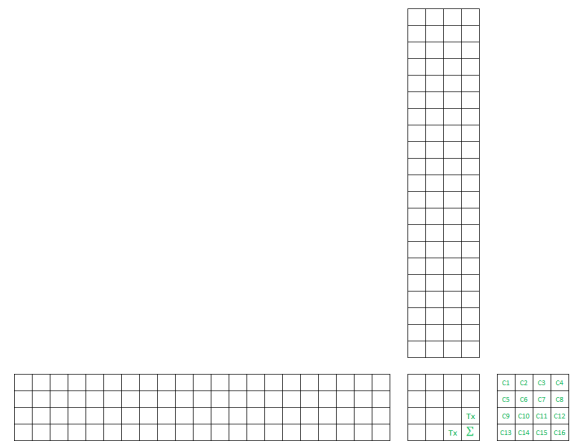
(21)



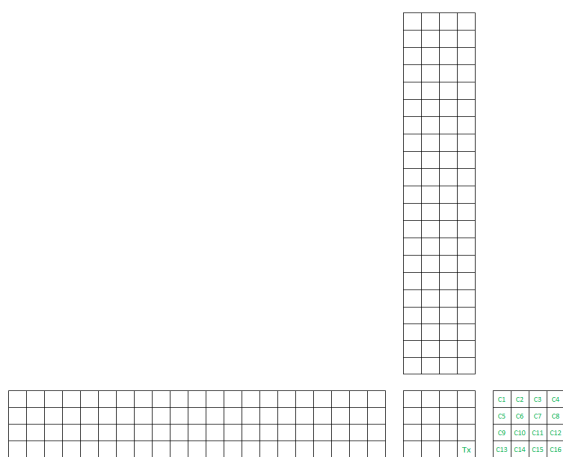
(22)



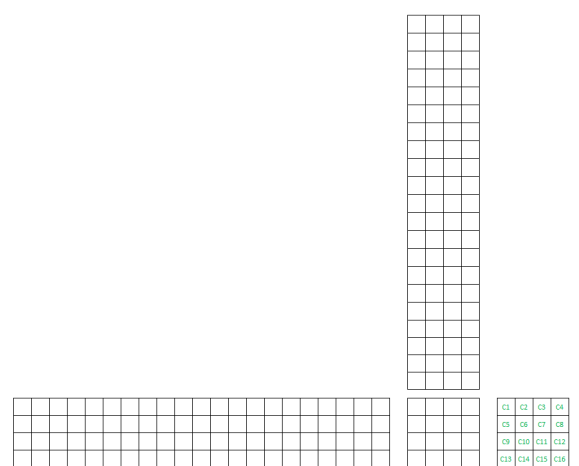
(23)



(24)



(25)



(26)

The program was run on the array continuously run on the array. The last instruction is to send the control points to the host computer through serial port. Each processing element sends their calculated control points to the host computer. The received control points are then visualised as a 3D surface in the host computer.

5.5 Experimental Results

To evaluate the accuracy of proposed shape sensing method, a number of experiments were conducted by wrapping the fabric onto different objects. The first experiment involved wrapping the fabric around a cylindrical object with a diameter 15cm and height 35cm, which was resting on one of its end faces on a horizontal table and then reconstructing its shape. The fabric swatch wrapped around the object is shown in Fig.5.7.



Figure 5.7: Fabric wrapped on a cylindrical object

The reconstructed image of the cylindrical object is shown in Fig.5.8. The X, Y and Z axis represents the calculated distance between the sensors in cm.

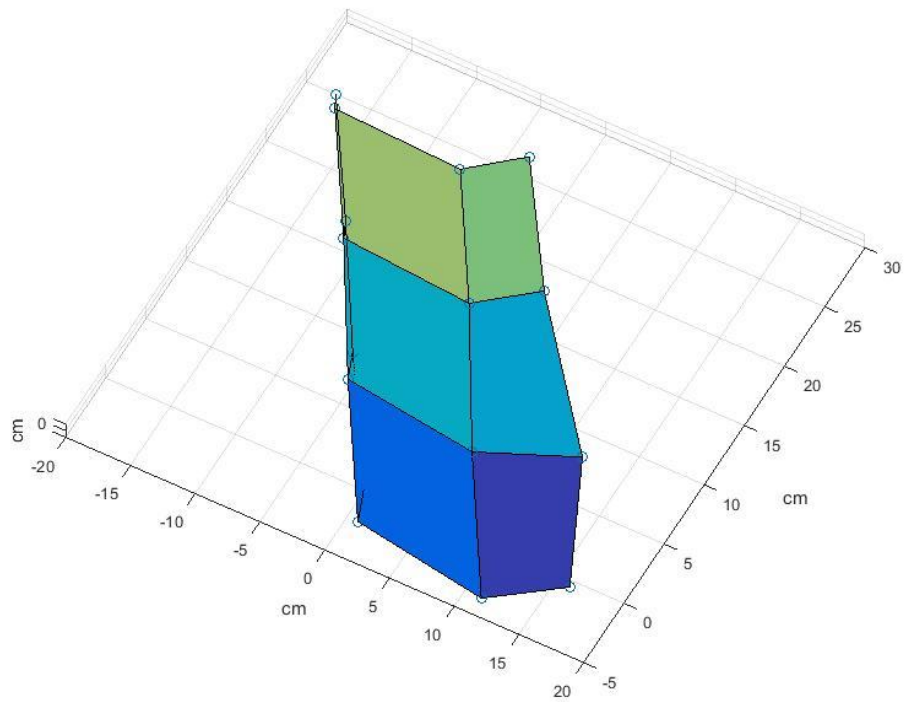


Figure 5.8: Reconstructed shape of the object

The second experiment involved placing the fabric on a ball with a diameter 65cm and then reconstructing the shape. The fabric swatch placed on the object is shown in Fig.5.9 and the reconstructed shape of is shown in Fig.5.10.



Figure 5.9: Fabric placed on the object

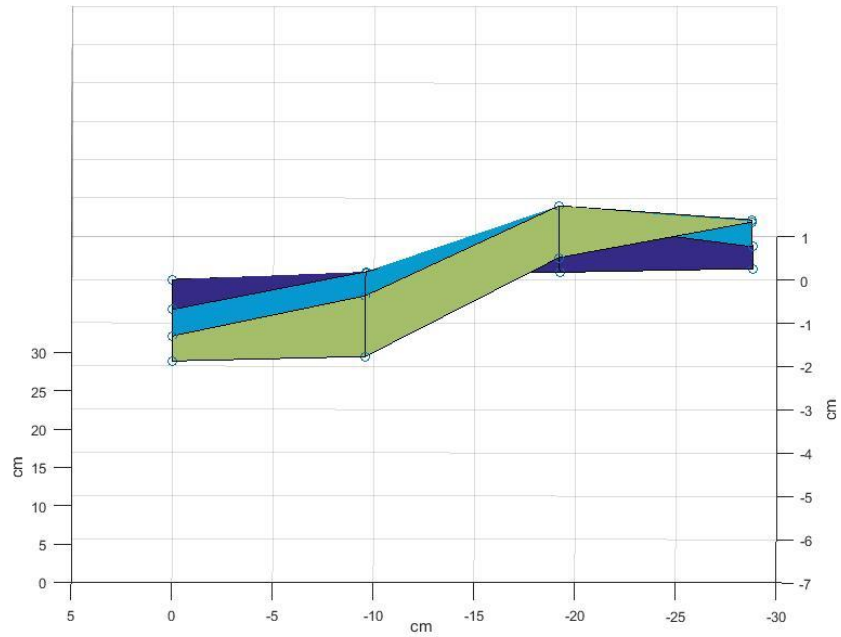


Figure 5.10: Reconstructed shape of the object

The third experiment involved placing the fabric on a perpendicular file with a length 24cm and width 32cm and then reconstructing the shape. The fabric swatch placed on the object is shown in Fig.5.11 and the reconstructed shape of is shown in Fig.5.12.

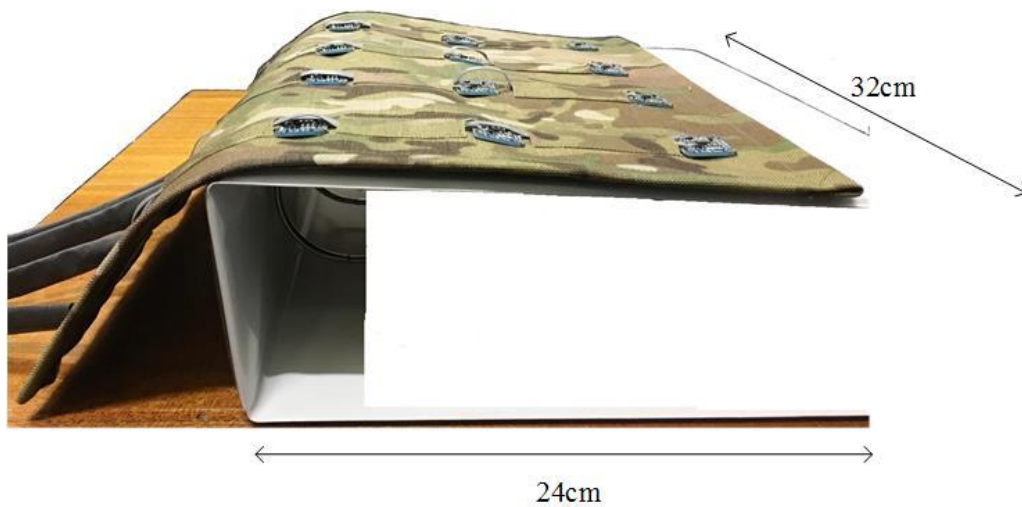


Figure 5.11: Fabric placed on the object

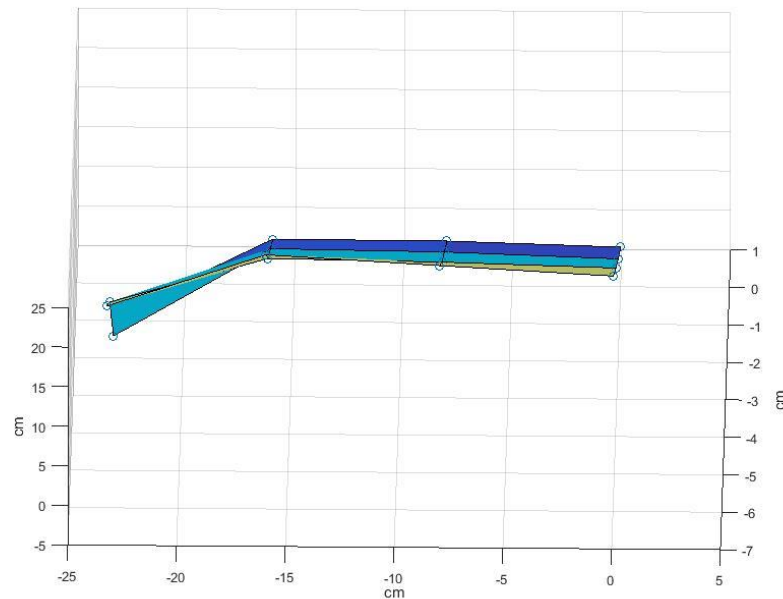


Figure 5.12: Reconstructed shape of the object

The reconstructed shape represents minor deviation from the sensor location mainly on the boundary sensors. The variation is about 0.2-0.4 cm. The variation could have been caused due to sensor noise and sensor mechanical mounting errors. The sensor noise introduces errors in Earth gravity and magnetic field vector component measurement. Sensor mechanical mounting errors include orientation errors, which introduce misalignment of sensor reference frame and placement errors, which introduces differences in inter sensor distances leading to orientation measurement in incorrect place on the curve.

Fig. 5.13 represents the execution milestone of the program as the program runs. It indicates the time difference between the instruction received and the selector bit sent between the processing elements.

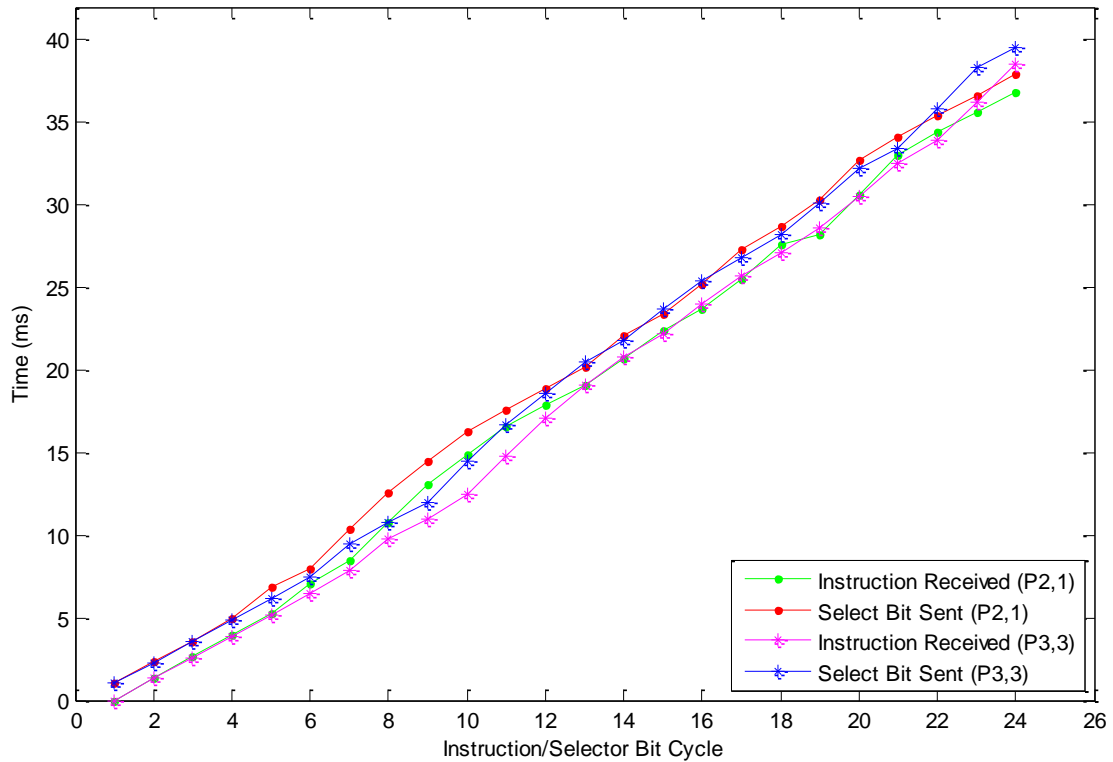


Figure 5.13: Performance analysis for P(2,1) and P(3,3)

A few instructions take longer to execute because of their implementation complexity. For example the seventh instruction on P(2,1) and the tenth instruction on P(3,3) which is a sensor read and takes an average of 1.55 millisecond to carry out, average and store in the register for further computation. In the current implementation, shared buses are being used through polling. Therefore delays occur through the communication. In a custom design, sensors could be more closely coupled to the processing element and the implementation can be carried out concurrently with the ISA processing function.

5.6 Conclusion

The wearable shape reconstruction application has been successfully implemented using our proposed concept of ISA architecture constructed out of off-the-shelf microcontrollers and sensors. Results demonstrate the application executes in 39.55ms on the prototype ISA implementation thus confirming the viability of the proposed architecture for fabric-resident computing devices.

References

- [5.1] A. Hermanis, R. Cacurs, M. Greitans, “Acceleration and magnetic sensor network for shape sensing,” *IEEE Sensors*, vol. 16, no. 5, pp. 1271–1280, March 2016.
- [5.2] T. Hoshi and H. Shinoda, “3D Shape capture sheet on gravity and geomagnetic sensing,” In Proc. 24th sensor symposium, pp. 423-427, 2007
- [5.3] G. Wahba, “A least squares estimate of satellite attitude,” *SIAM Rev.*, vol. 7, no. 3, pp. 409, 1965
- [5.4] M. D. Shuster and S. D. Oh, “Three-axis attitude determination from vector observations,” *J. Guid. Control, Dyn.*, vol. 4, no. 1, pp. 70–77, 1981
- [5.5] Afdafruit, “LSM303 Accelerometer + Compass Breakout,” [Online]. [Accessed : 24 July 2017]. Available from: <https://learn.adafruit.com/lsm303-accelerometer-slash-compass-breakout/overview>
- [5.6] A. Hermanis and K. Nesenbergs, “Grid shaped accelerometer network for surface shape recognition,” in Proc. 13th Biennial Baltic Electron. Conf. (BEC), pp. 203–206, 2012
- [5.7] T. Hoshi, S. Ozaki, and H. Shinoda, “Three-dimensional shape capture sheet using distributed triaxial accelerometers,” in Proc. 4th Int. Conf. Netw. Sens. Syst. (INSS), pp. 207–212, 2007
- [5.8] P. Mittendorfer and G. Cheng, “3D surface reconstruction for robotic body parts with artificial skins,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS), pp. 4505–4510, 2012
- [5.9] M. Huard, N. Sprynski, N. Szafran, and L. Biard, “Reconstruction of quasi developable surfaces from ribbon curves,” *Numer. Algorithms*, vol. 63, no. 3, pp. 483–506, 2013

CHAPTER 6:

CONCLUSION AND FUTURE WORK

THE aim of this thesis was to propose and implement a novel distributed computer which could be used for wearables. This chapter summarizes the contributions of the thesis and discusses the future work that can be conducted.

6.1 Contribution of this thesis

The aim of the research was to harness parallel processing across a large number of simple cores with the objective of improving the performance when compared to a serial system.

The main contributions of the thesis are:

- A new sensor networking paradigm that exploits processor level parallelism has been implemented and also has introduced the concept of on-fabric computation.
- Validated the method and produced parallel program that is used on the sensor network array.
- Produced a physical demonstrator for a specific measurement scenario that has relevance to human monitoring application.
- The architecture has been applied to a physical demonstrator containing an array of computing nodes.
- Set of measurements obtained from a physical demonstrator has been presented.

The thesis has proposed a completely new concept of an on-fabric Instruction Systolic Array. Different parallel architectures have been reviewed and the Instruction Systolic Array is a relatively under researched architecture that has meant in this particular application.

A number of compromises have been made during the implementation of the concept such as opting a particular bus system, selecting a microcontroller for the processing element, using same bus connecting for peripheral devices. It can be expected that these

will have a substantial impact on the performance. But still some advantages in implementing such as to test the functionality of the device and to prove the concept and experiment with the programmer's model which is significantly different to any known computer.

The wearable shape reconstruction application has been successfully implemented using the proposed concept of ISA architecture constructed out of off-the-shelf microcontrollers and sensors. Results confirm the viability of the proposed architecture for fabric-resident computing devices.

6.2 Suggestions for future research

The thesis suggests a number of possibilities for future research.

6.2.1 Computational performance

The next step in the research would be implementing the whole prototype system on an ASIC. While implementing the concept on ASIC the processors will be closely coupled with the sensors. When the sensors are closely coupled to the processing elements there will not be a need for the shared bus system and thus will result in better performance. Thus there will also not be a need of the umbilical as shown in Fig.5.4.

6.2.2 Scalability

One of the advantages of ISA is that it is scalable. The processing elements and sensors are scalable where the array for the processing elements and the sensors connected to the processing elements can be increased or decreased. The future research can build on into scalability because final integration of the concept is to have very large arrays that can self-process.

6.2.3 Programming techniques

This research has shown a conventional way of programming the ISA. More research can be conducted in future in the development of a full programmer's model and a full featured instruction set. Future research can also concentrate on efficient development environment and high-level programming language for the ISA which is inherently difficult to program.

6.2.4 Designing

A prototype is typically a working model of a design that demonstrates a device's appearance and functionality which has been implemented. The next stage is to have a custom design which would clearly be able to obtain a much better performance. Once the custom design is built the manufacturing could be done in big volumes. Manufacturing the product in large scale could be outsourced. Handing off prototyping to an outsourced firm can save precious time and money in the development process, as design and knowledge transfers can be streamlined.

6.2.5 Applications

There are several wearable applications that can be explored using the concept of Instruction Systolic Array. Human body sensing and human posture sensing applications are more commonly used applications in wearables.

- **Medicine:** Vital signs monitoring, body chemistry monitoring, stroke rehabilitation, blood pressure measurement.
- **Military:** Vital signs monitoring, performance monitoring, physical condition, position and orientation monitoring, radiation monitoring, monitoring of harmful gasses, wearable communications devices, camouflage, smart clothing with response to the environment. Active Camouflage is the concept of including actuators and optical devices closely coupled to Processing elements which can be used in Military applications.
- **Sports:** Performance monitoring and vital signs monitoring of the athletes and players during the sporting events helps in monitoring their health and improve their performance.

6.3 Summary

The thesis has discussed the rationale, design, implementation and benchmarking of a new concept for on-fabric sensor networks, prototyped with off-the-shelf microcontrollers. A physical prototype device has been demonstrated containing 16 computing nodes. The concept has been validated using several programming examples. The parallel architecture has been demonstrated using on-fabric application.

It is envisaged that such a system would be implemented using VLSI technology and custom ASICs which would substantially improve the performance. The future work can address the scalability of the architecture in line with the thesis vision to extend to large arrays and new applications. Several wearable applications in the field of medicine, military and sports can also be explored using the concepts and methodologies developed during this research. There can also be focus on extending the supported instructions, optimizing the communication medium and allowing for more concurrency, at node level, between computation and communication.

References

- [6.1] B. Burchard, S. Jung, A. Ullsperger, and W. D. Hartmann, “Devices, software, their applications and requirements for wearable electronics,” ICCE, pp. 224–225, 2001
- [6.2] S. I. Woolley, J. W. Cross, S. Ro, R. Foster, G. Reynolds, C. Baber, H. Bristow, and A. Schwartz, “Forms of wearable computer,” in IEE Eurowearable, IET, pp. 47-52, 2003
- [6.3] K. V. Laerhoven, A. Schmidt, and H-W. Gellersen, “Multi-Sensor Context Aware Clothing,” in ISWC, IEEE Computer Society, pp.49-56, 2002
- [6.4] C. L. Cathey, J. D. Bakos, and D. A. Buell, “A reconfigurable distributed computing fabric exploiting multilevel parallelism,”inFCCM, IEEE Computer Society, pp. 121–130, 2006

LIST OF PUBLICATIONS

1. P. Kandaswamy, J. Flint, V. Chouliaras, "Shape Reconstruction using Instruction Systolic Array," IEEE Sensors 2017, pp. 364-366, 2017.
2. P. Kandaswamy, J. Flint, V. Chouliaras, "System on Fabrics Architecture using Distributed Computing," IEEE Sensors Journal, Peer-reviewed and Accepted, Preprint (Early Access Available Online), May 2018.

APPENDIX

The computational model code for the shape reconstruction algorithm is as follows. This is a common code and can be configured for different roles. The file `global_defines.h` which has been included is used to configure the code type.

```
/*
*****
* Include header files
*****
*****/
#include "board.h"
#include "global_defines.h" // this file sets the code configuration
                           type
#include "chip.h"
#include "patterns.h"
#include "fifo.h"
#include "string.h"
#include "algo.h"

/*
*****
* Private types/enumerations/variables
*****
*****/
#define M_TX_BUFF_SIZE          750
#define M_RX_BUFF_SIZE          750
#define NO_OF_SENSOR_READ      4
/*
*****
* Public types/enumerations/variables
*****
*****/
typedef struct
{
    uint8_t frame_type;
    union
    {
        uint8_t data;
        uint8_t instruction;
        uint8_t selector_bit;
    } data;
    float C[3][1]; //control register
    float N[3][1]; //North communication register
    float E[3][1]; //East communication register
    float W[3][1]; //West communication register
}
```

```

        float S[3][1];    //South communication register
    } FRAME;
    static I2CM_XFER_T i2cm0Xfer;
    static I2CM_XFER_T i2cm1Xfer;
    #if(CONTROLLER != CONTROLLER_PROCESS_ELEMENTS)
    static I2CM_XFER_T i2cm2Xfer;
    static I2CM_XFER_T i2cm3Xfer;
    #else
    #ifndef LAST_SOUTH_CONTROLLER
    FRAME north_prev;
    #endif
    #ifndef LAST_EAST_CONTROLLER
    FRAME west_prev;
    #endif
    floatnorth_C[3] = {0, 0, 0};
    floatnorth_N[3] = {0, 0, 0};
    floatnorth_E[3] = {0, 0, 0};
    floatnorth_W[3] = {0, 0, 0};
    floatnorth_S[3] = {0, 0, 0};
    floatwest_C[3] = {0, 0, 0};
    floatwest_N[3] = {0, 0, 0};
    floatwest_E[3] = {0, 0, 0};
    floatwest_W[3] = {0, 0, 0};
    floatwest_S[3] = {0, 0, 0};
    static uint8_t tx_buff[M_TX_BUFF_SIZE];
    static uint8_t rx_buff[M_TX_BUFF_SIZE];
    #endif

    static uint8_t slave1_no_of_bytes_received;
    static uint8_t slave2_no_of_bytes_received;
    static volatile bool delay_completed = false;
    static volatile uint32_t delay_counter = 0;
    static FIFO<uint8_t, 750>fifo_north;
    static FIFO<uint8_t, 750>fifo_west;
    static uint32_t time_in_10ms = 0;
    /*****
    *****
    * Private functions
    *****
    *****/
    static void processSlave1TransferStart(uint8_t addr);
    static uint8_t processSlave1TransferSend(uint8_t *data);
    static uint8_t processSlave1TransferRecv(uint8_t data);
    static void processSlave1TransferDone(void);

    static void processSlave2TransferStart(uint8_t addr);
    static uint8_t processSlave2TransferSend(uint8_t *data);
    static uint8_t processSlave2TransferRecv(uint8_t data);
    static void processSlave2TransferDone(void);

    const static I2CS_XFER_T i2cs1Callbacks =
    {
        &processSlave1TransferStart,
        &processSlave1TransferSend,
        &processSlave1TransferRecv,
        &processSlave1TransferDone
    };
    const static I2CS_XFER_T i2cs2Callbacks =

```

```

{
    &processSlave2TransferStart,
    &processSlave2TransferSend,
    &processSlave2TransferRecv,
    &processSlave2TransferDone
};

/*****
*****
* Public functions
*****
*****/
extern "C" void IOCON_Init();
extern "C" void InputMux_Init();
extern "C" void SwitchMatrix_Init();

/* Handler for slave start callback */
static void processSlave1TransferStart(uint8_t addr)
{
    slave1_no_of_bytes_received = 0;
}
/* Handler for slave send callback */
static uint8_t processSlave1TransferSend(uint8_t *data)
{
    return 1;          // return a non zero to indicate there is data
}

/* Handler for slave receive callback */
static uint8_t processSlave1TransferRecv(uint8_t data)
{
    fifo_north = data;
    return 0;
}

/* Handler for slave transfer complete callback */
static void processSlave1TransferDone(void)
{
    /* Nothing needs to be done here */
}

/* Handler for slave start callback */
static void processSlave2TransferStart(uint8_t addr)
{
    slave2_no_of_bytes_received = 0;
}
/* Handler for slave send callback */
static uint8_t processSlave2TransferSend(uint8_t *data)
{
    return 1;
}

/* Handler for slave receive callback */
static uint8_t processSlave2TransferRecv(uint8_t data)
{
    fifo_west = data;
    return 0;
}

```



```
/* Handler for slave transfer complete callback */
static void processSlave2TransferDone(void)
{
    /* Nothing needs to be done here */
}

/* Function to wait for I2CM transfer completion */
static void WaitForI2cXferComplete(I2CM_XFER_T *xferRecPtr)
{
    /* Test for still transferring data */
    while (xferRecPtr->status == I2CM_STATUS_BUSY)
    {
        /* Sleep until next interrupt */
        __WFI();
    }
}

/* Function to setup and execute I2C transfer request */
static void SetupXferRecAndExecute(LPC_I2C_T * i2c_unit,
uint8_t devAddr, uint8_t *txBuffPtr, uint16_t txSize, uint8_t
*rxBuffPtr, uint16_t rxSize)
{
    I2CM_XFER_T* xfer = 0;
#if(CONTROLLER == CONTROLLER_INSTRUCTION)
    if(i2c_unit == I2C_COL1_MASTER)
    {
        xfer = &i2cm0Xfer;
    }
    else if(i2c_unit == I2C_COL2_MASTER)
    {
        xfer = &i2cm1Xfer;
    }
    else if(i2c_unit == I2C_COL3_MASTER)
    {
        xfer = &i2cm2Xfer;
    }
    else if(i2c_unit == I2C_COL4_MASTER)
    {
        xfer = &i2cm3Xfer;
    }
#elif(CONTROLLER == CONTROLLER_SELECTOR_BIT)
    if(i2c_unit == I2C_ROW1_MASTER)
    {
        xfer = &i2cm0Xfer;
    }
    else if(i2c_unit == I2C_ROW2_MASTER)
    {
        xfer = &i2cm1Xfer;
    }
    else if(i2c_unit == I2C_ROW3_MASTER)
    {
        xfer = &i2cm2Xfer;
    }
    else if(i2c_unit == I2C_ROW4_MASTER)
    {
        xfer = &i2cm3Xfer;
    }
#elif(CONTROLLER == CONTROLLER_PROCESS_ELEMENTS)
    if(i2c_unit == I2C_SOUTH_MASTER)
```

```

    {
        xfer = &i2cm0Xfer;
    }
    else if(i2c_unit == I2C_EAST_MASTER)
    {
        xfer = &i2cm1Xfer;
    }
#endif

    if(xfer != 0)
    {
        /* Setup I2C transfer record */
        xfer->slaveAddr = devAddr;
        xfer->status = 0;
        xfer->txSz = txSize;
        xfer->rxSz = rxSize;
        xfer->txBuff = txBuffPtr;
        xfer->rxBuff = rxBuffPtr;

        Chip_I2CM_Xfer(i2c_unit, xfer);
        /* Enable Master Interrupts */
        Chip_I2C_EnableInt(i2c_unit, I2C_INTENSET_MSTPENDING |
I2C_INTENSET_MSTRARBLOSS | I2C_INTENSET_MSTSTSTPERR);
        /* Wait for transfer completion */
        WaitForI2cXferComplete(xfer);
        /* Clear all Interrupts */
        Chip_I2C_ClearInt(i2c_unit, I2C_INTENSET_MSTPENDING |
I2C_INTENSET_MSTRARBLOSS | I2C_INTENSET_MSTSTSTPERR);
    }
}

void i2c_master_init(LPC_I2C_T * i2c_unit)
{
    /* Enable I2C clock and reset I2C peripheral */
    Chip_I2C_Init(i2c_unit);

    /* Setup clock rate for I2C */
    Chip_I2C_SetClockDiv(i2c_unit, SystemCoreClock / I2C_SPEED);

    /* Setup I2CM transfer rate */
    Chip_I2CM_SetBusSpeed(i2c_unit, I2C_SPEED);

    /* Enable Master Mode */
    Chip_I2CM_Enable(i2c_unit);
}

#if(CONTROLLER == CONTROLLER_PROCESS_ELEMENTS)
/* Setup I2C */
static void i2c_slave_init(LPC_I2C_T * i2c_unit)
{
    /* Enable I2C clock and reset I2C peripheral */
    Chip_I2C_Init(i2c_unit);

    /* Setup clock rate for I2C */
    Chip_I2C_SetClockDiv(i2c_unit, SystemCoreClock / I2C_SPEED);

    /* Setup I2CM transfer rate */
    Chip_I2CM_SetBusSpeed(i2c_unit, I2C_SPEED);
}

```

```

    /* Enable I2C master interface */
    Chip_I2CM_Enable(i2c_unit);

    /* Some common I2C init was performed in setupI2CMaster(), so it
    doesn't need to be done again for the slave setup. */

    /* Emulated EEPROM 0 is on slave index 0 */
    Chip_I2CS_SetSlaveAddr(i2c_unit, 0, I2C_SLAVE_ADDR);
    /* Disable Qualifier for Slave Address 0 */
    Chip_I2CS_SetSlaveQual0(i2c_unit, false, 0);
    /* Enable Slave Address 0 */
    Chip_I2CS_EnableSlaveAddr(i2c_unit, 0);

    /* Clear interrupt status and enable slave interrupts */
    Chip_I2CS_ClearStatus(i2c_unit, I2C_STAT_SLVDESEL);
    Chip_I2C_EnableInt(i2c_unit, I2C_INTENSET_SLVPENDING |
I2C_INTENSET_SLVDESEL);

    /* Enable I2C slave interface */
    Chip_I2CS_Enable(i2c_unit);
}

extern "C" void I2C_NORTH_SLAVE_IRQHandler(void)
{
    uint32_t state = Chip_I2C_GetPendingInt(I2C_NORTH_SLAVE);

    /* Error handling */
    if (state & (I2C_INTSTAT_MSTRARBLOSS | I2C_INTSTAT_MSTSTSTPERR))
    {
        Chip_I2CM_ClearStatus(I2C_NORTH_SLAVE,
                                I2C_STAT_MSTRARBLOSS |
I2C_STAT_MSTSTSTPERR);
    }

    /* I2C slave related interrupt */
    while (state & (I2C_INTENSET_SLVPENDING |
I2C_INTENSET_SLVDESEL))
    {
        Chip_I2CS_XferHandler(I2C_NORTH_SLAVE, &i2cs1Callbacks);

        /* Update state */
        state = Chip_I2C_GetPendingInt(I2C_NORTH_SLAVE);
    }
}

extern "C" void I2C_WEST_SLAVE_IRQHandler(void)
{
    uint32_t state = Chip_I2C_GetPendingInt(I2C_WEST_SLAVE);

    /* Error handling */
    if (state & (I2C_INTSTAT_MSTRARBLOSS | I2C_INTSTAT_MSTSTSTPERR))
    {
        Chip_I2CM_ClearStatus(I2C_WEST_SLAVE,
                                I2C_STAT_MSTRARBLOSS |
I2C_STAT_MSTSTSTPERR);
    }

    /* I2C slave related interrupt */

```

```
        while (state & (I2C_INTENSET_SLVPENDING |
I2C_INTENSET_SLVDESEL))
        {
            Chip_I2CS_XferHandler(I2C_WEST_SLAVE, &i2cs2Callbacks);

            /* Update state */
            state = Chip_I2C_GetPendingInt(I2C_WEST_SLAVE);
        }
    }

extern "C" void I2C_SOUTH_MASTER_IRQHandler(void)
{
    /* Call I2CM ISR function with the I2C device and transfer rec
*/
    Chip_I2CM_XferHandler(I2C_SOUTH_MASTER, &i2cm0Xfer);
}

extern "C" void I2C_EAST_MASTER_IRQHandler(void)
{
    /* Call I2CM ISR function with the I2C device and transfer rec
*/
    Chip_I2CM_XferHandler(I2C_EAST_MASTER, &i2cm1Xfer);
}

#endif

/**
 * Handle I2C interrupt by calling I2CM interrupt transfer handler
 * @return Nothing
 */
#if(CONTROLLER == CONTROLLER_INSTRUCTION)
extern "C" void I2C_COL1_MASTER_IRQHandler(void)
{
    /* Call I2CM ISR function with the I2C device and transfer rec
*/
    Chip_I2CM_XferHandler(I2C_COL1_MASTER, &i2cm0Xfer);
}

extern "C" void I2C_COL2_MASTER_IRQHandler(void)
{
    /* Call I2CM ISR function with the I2C device and transfer rec
*/
    Chip_I2CM_XferHandler(I2C_COL2_MASTER, &i2cm1Xfer);
}

extern "C" void I2C_COL3_MASTER_IRQHandler(void)
{
    /* Call I2CM ISR function with the I2C device and transfer rec
*/
    Chip_I2CM_XferHandler(I2C_COL3_MASTER, &i2cm2Xfer);
}

extern "C" void I2C_COL4_MASTER_IRQHandler(void)
{
    /* Call I2CM ISR function with the I2C device and transfer rec
*/
    Chip_I2CM_XferHandler(I2C_COL4_MASTER, &i2cm3Xfer);
}
#endif
```

```
#if(CONTROLLER == CONTROLLER_SELECTOR_BIT)
extern "C" void I2C_ROW1_MASTER_IRQHandler(void)
{
    /* Call I2CM ISR function with the I2C device and transfer rec
    */
    Chip_I2CM_XferHandler(I2C_ROW1_MASTER, &i2cm0Xfer);
}

extern "C" void I2C_ROW2_MASTER_IRQHandler(void)
{
    /* Call I2CM ISR function with the I2C device and transfer rec
    */
    Chip_I2CM_XferHandler(I2C_ROW2_MASTER, &i2cm1Xfer);
}

extern "C" void I2C_ROW3_MASTER_IRQHandler(void)
{
    /* Call I2CM ISR function with the I2C device and transfer rec
    */
    Chip_I2CM_XferHandler(I2C_ROW3_MASTER, &i2cm2Xfer);
}

extern "C" void I2C_ROW4_MASTER_IRQHandler(void)
{
    /* Call I2CM ISR function with the I2C device and transfer rec
    */
    Chip_I2CM_XferHandler(I2C_ROW4_MASTER, &i2cm3Xfer);
}
#endif

/**
 * Handle interrupt from SysTick timer
 * return Nothing
 */
extern "C" void SysTick_Handler(void)
{
    time_in_10ms++;
    if(delay_counter > 0)
    {
        delay_counter--;
        if(delay_counter == 0)
            delay_completed = true;
    }
}

void delay(uint32_t delay_in_10ms)
{
    if(delay_in_10ms != 0)
    {
        delay_completed = false;
        delay_counter = delay_in_10ms;
        while(delay_completed == false);
    }
}

#if(CONTROLLER == CONTROLLER_PROCESS_ELEMENTS)
uint8_t lsm303dlhc_read_reg(uint8_t dev_address, uint8_t reg_address)
```

```

{
    tx_buff[0] = reg_address;
    SetupXferRecAndExecute(I2C_SOUTH_MASTER, dev_address, tx_buff,
1, rx_buff, 1);
    return rx_buff[0];
}

void lsm303dlhc_write_reg(uint8_t dev_address, uint8_t reg_address,
uint8_t data)
{
    tx_buff[0] = reg_address;
    tx_buff[1] = data;
    SetupXferRecAndExecute(I2C_SOUTH_MASTER, dev_address, tx_buff,
2, 0, 0);
}

void lsm303dlhc_accel_read_xyz(int16_t& x, int16_t& y, int16_t& z)
{
    tx_buff[0] = LSM303DLHC_ACCEL_OUT_X_L_A | 0x80;          //
continuous read
    SetupXferRecAndExecute(I2C_SOUTH_MASTER,
LSM303DLHC_ACCEL_SLAVE_ADDR, tx_buff, 1, rx_buff, 6);
    x = (rx_buff[0] | (rx_buff[1] << 8));
    y = (rx_buff[2] | (rx_buff[3] << 8));
    z = (rx_buff[4] | (rx_buff[5] << 8));
}

void lsm303dlhc_gyro_read_xyz(int16_t& x, int16_t& y, int16_t& z)
{
    tx_buff[0] = LSM303DLHC_GYRO_OUT_X_L_M | 0x80;          //
continuous read
    SetupXferRecAndExecute(I2C_SOUTH_MASTER,
LSM303DLHC_GYRO_SLAVE_ADDR, tx_buff, 1, rx_buff, 6);
    x = (rx_buff[0] | (rx_buff[1] << 8));
    y = (rx_buff[2] | (rx_buff[3] << 8));
    z = (rx_buff[4] | (rx_buff[5] << 8));
}

void calculate_control_point(uint8_t id)
{
    float two[3] = {2, 2, 2};
    switch(id)
    {
        case 1:
            C[0] = 0;
            C[1] = 0;
            C[2] = 0;
            break;
        case 2:
        case 3:
        case 4:
            algo_matrix_add_3x1_and_3x1(west_C, west_E, C);
            algo_matrix_sub_3x1_and_3x1(C, W, C);
            break;
        case 5:
        case 9:
        case 13:

```

```

        algo_matrix_add_3x1_and_3x1(N, north_S, C);
        algo_matrix_sub_3x1_and_3x1(north_C, C, C);
        break;
    case 6:
    case 7:
    case 8:
    case 10:
    case 11:
    case 12:
    case 14:
    case 15:
    case 16:

        algo_matrix_add_3x1_and_3x1(west_C, west_E, C);
        algo_matrix_add_3x1_and_3x1(C, north_C, C);
        algo_matrix_add_3x1_and_3x1(C, north_S, C);
        algo_matrix_sub_3x1_and_3x1(C, W, C);
        algo_matrix_sub_3x1_and_3x1(C, N, C);
        algo_matrix_sub_3x1_and_3x1(C, two, C);
        break;
    }
}
#endif

/**
 *   main routine
 *   return Function should not exit.
 */
int main(void)
{
    volatile uint32_t *vt;
    uint32_t cpu_id;
#ifdef CONTROLLER == CONTROLLER_PROCESS_ELEMENTS
    uint8_t controller_id = 0;
    FRAME north_curr;
    FRAME west_curr;
    int16_t x, y, z;
    int16_t sum_x, sum_y, sum_z;
#else
    FRAME tx_frame;
#endif

    SystemCoreClockUpdate();
    Board_Init();

    IOCON_Init();
    InputMux_Init();
    SwitchMatrix_Init();

    // Set 10ms tick
    SysTick_Config(Chip_Clock_GetSystemClockRate() / 100);

    /* Display system information */
    __disable_irq();
#ifdef CONTROLLER == CONTROLLER_PROCESS_ELEMENTS
#ifdef LAST_SOUTH_CONTROLLER
#ifdef LAST_EAST_CONTROLLER
    printf("Process Element Controller LSE\n");
#else

```

```
        printf("Process Element Controller LS\n");
    #endif
    #else
    #ifdef LAST_EAST_CONTROLLER
        printf("Process Element Controller LE\n");
    #else
        printf("Process Element Controller\n");
    #endif
    #endif
    fflush(stdout);
    #elif(CONTROLLER == CONTROLLER_INSTRUCTION)
        printf("Instruction Controller\n");
        fflush(stdout);
    #elif(CONTROLLER == CONTROLLER_SELECTOR_BIT)
        printf("Selector Bit Controller\n");
        fflush(stdout);
    #endif
    printf("System Clock: %luMHz\n", SystemCoreClock / 1000000);
    fflush(stdout);
    printf("Device ID: 0x%04lX\n", Chip_SYSTCL_GetDeviceID());
    fflush(stdout);
    vt = &(SCB->VTOR);
    cpu_id = SCB->CPUID;
    printf("VTOR Address: 0x%08lX\n", (uint32_t ) vt);
    fflush(stdout);
    printf("CPU ID: 0x%08lX\n", (uint32_t ) cpu_id);
    fflush(stdout);
    printf(VERSION_STRING);
    __enable_irq();
    printf("time in 10ms tick = %u", (unsigned int)time_in_10ms);

    // Enable pullups for all
    // I2C 0
    //     Chip_IOCON_PinSetMode(LPC_IOCON, IOCON_PIO11,
PIN_MODE_PULLUP); // SDA // there is no pullup available in
PIO11 & 10
    //     Chip_IOCON_PinSetMode(LPC_IOCON, IOCON_PIO10,
PIN_MODE_PULLUP); // SCL
    Chip_IOCON_PinSetI2CMode(LPC_IOCON, IOCON_PIO11,
PIN_I2CMODE_STDFAST);
    Chip_IOCON_PinSetI2CMode(LPC_IOCON, IOCON_PIO10,
PIN_I2CMODE_STDFAST);

    // I2C 3
    Chip_IOCON_PinSetMode(LPC_IOCON, IOCON_PIO19, PIN_MODE_PULLUP);
    // SDA
    Chip_IOCON_PinSetMode(LPC_IOCON, IOCON_PIO12, PIN_MODE_PULLUP);
    // SCL
    Chip_IOCON_PinSetOpenDrainMode(LPC_IOCON, IOCON_PIO19, true);
    Chip_IOCON_PinSetOpenDrainMode(LPC_IOCON, IOCON_PIO12, true);

    // I2C 1
    Chip_IOCON_PinSetMode(LPC_IOCON, IOCON_PIO18, PIN_MODE_PULLUP);
    // SDA
    Chip_IOCON_PinSetMode(LPC_IOCON, IOCON_PIO28, PIN_MODE_PULLUP);
    // SCL
    Chip_IOCON_PinSetOpenDrainMode(LPC_IOCON, IOCON_PIO18, true);
    Chip_IOCON_PinSetOpenDrainMode(LPC_IOCON, IOCON_PIO28, true);
```



```

// I2C 2
Chip_IOCON_PinSetMode(LPC_IOCON, IOCON_PIO0, PIN_MODE_PULLUP);
// SDA
Chip_IOCON_PinSetMode(LPC_IOCON, IOCON_PIO4, PIN_MODE_PULLUP);
// SCL
Chip_IOCON_PinSetOpenDrainMode(LPC_IOCON, IOCON_PIO0, true);
Chip_IOCON_PinSetOpenDrainMode(LPC_IOCON, IOCON_PIO4, true);

#if(CONTROLLER == CONTROLLER_PROCESS_ELEMENTS)
// Init I2C Masters
i2c_master_init(I2C_SOUTH_MASTER);
i2c_master_init(I2C_EAST_MASTER);
// Init I2C Slave
i2c_slave_init(I2C_NORTH_SLAVE);
i2c_slave_init(I2C_WEST_SLAVE);
#else
i2c_master_init(LPC_I2C0);
i2c_master_init(LPC_I2C1);
i2c_master_init(LPC_I2C2);
i2c_master_init(LPC_I2C3);
#endif

/* Enable the interrupt for the I2C */
NVIC_SetPriority(I2C0_IRQn, 31);
NVIC_SetPriority(I2C1_IRQn, 31);
NVIC_SetPriority(I2C2_IRQn, 31);
NVIC_SetPriority(I2C3_IRQn, 31);
NVIC_EnableIRQ(I2C0_IRQn);
NVIC_EnableIRQ(I2C1_IRQn);
NVIC_EnableIRQ(I2C2_IRQn);
NVIC_EnableIRQ(I2C3_IRQn);

// LED init
GREEN_LED_OFF();
BLUE_LED_OFF();

#if(CONTROLLER == CONTROLLER_INSTRUCTION)
memset((void*)&tx_frame, 0, sizeof(tx_frame));
for(uint8_t i = 0; i<rows_of_array(vertical_pattern); i++)
{
    GREEN_LED_ON();
    tx_frame.frame_type = vertical_pattern_type[i][0];
    tx_frame.data.data = vertical_pattern[i][0];
    SetupXferRecAndExecute(I2C_COL1_MASTER, I2C_SLAVE_ADDR,
(uint8_t*)&tx_frame, sizeof(tx_frame), 0, 0);
    printf("Instruction %u ", i + 1);
    fflush(stdout);
    printf("sent to COL1\n");
    fflush(stdout);

    tx_frame.frame_type = vertical_pattern_type[i][1];
    tx_frame.data.data = vertical_pattern[i][1];
    SetupXferRecAndExecute(I2C_COL2_MASTER, I2C_SLAVE_ADDR,
(uint8_t*)&tx_frame, sizeof(tx_frame), 0, 0);
    printf("Instruction %u ", i + 1);
    fflush(stdout);
    printf("sent to COL2\n");
    fflush(stdout);
}

```

```
        tx_frame.frame_type = vertical_pattern_type[i][2];
        tx_frame.data.data = vertical_pattern[i][2];
        SetupXferRecAndExecute(I2C_COL3_MASTER, I2C_SLAVE_ADDR,
(uint8_t*)&tx_frame, sizeof(tx_frame), 0, 0);
        printf("Instruction %u ", i + 1);
        fflush(stdout);
        printf("sent to COL3\n");
        fflush(stdout);

        tx_frame.frame_type = vertical_pattern_type[i][3];
        tx_frame.data.data = vertical_pattern[i][3];
        SetupXferRecAndExecute(I2C_COL4_MASTER, I2C_SLAVE_ADDR,
(uint8_t*)&tx_frame, sizeof(tx_frame), 0, 0);
        printf("Instruction %u ", i + 1);
        fflush(stdout);
        printf("sent to COL4\n");
        fflush(stdout);

        GREEN_LED_OFF();
        delay(PATTERN_DELAY_IN_10ms);
    }
    GREEN_LED_OFF();
    while(1);
#elif (CONTROLLER == CONTROLLER_SELECTOR_BIT)
    memset((void*)&tx_frame, 0, sizeof(tx_frame));
    for(uint8_t i = 0; i < rows_of_array(horizontal_pattern); i++)
    {
        BLUE_LED_ON();
        tx_frame.frame_type = horizontal_pattern_type[i][0];
        tx_frame.data.data = horizontal_pattern[i][0];
        SetupXferRecAndExecute(I2C_ROW1_MASTER, I2C_SLAVE_ADDR,
(uint8_t*)&tx_frame, sizeof(tx_frame), 0, 0);
        printf("Selector Bit %u %u sent to ROW1\n", i + 1,
tx_frame.data);
        fflush(stdout);

        tx_frame.frame_type = horizontal_pattern_type[i][1];
        tx_frame.data.data = horizontal_pattern[i][1];
        SetupXferRecAndExecute(I2C_ROW2_MASTER, I2C_SLAVE_ADDR,
(uint8_t*)&tx_frame, sizeof(tx_frame), 0, 0);
        printf("Selector Bit %u %u sent to ROW2\n", i + 1,
tx_frame.data);
        fflush(stdout);

        tx_frame.frame_type = horizontal_pattern_type[i][2];
        tx_frame.data.data = horizontal_pattern[i][2];
        SetupXferRecAndExecute(I2C_ROW3_MASTER, I2C_SLAVE_ADDR,
(uint8_t*)&tx_frame, sizeof(tx_frame), 0, 0);
        printf("Selector Bit %u %u sent to ROW3\n", i + 1,
tx_frame.data);
        fflush(stdout);

        tx_frame.frame_type = horizontal_pattern_type[i][3];
        tx_frame.data.data = horizontal_pattern[i][3];
        SetupXferRecAndExecute(I2C_ROW4_MASTER, I2C_SLAVE_ADDR,
(uint8_t*)&tx_frame, sizeof(tx_frame), 0, 0);
        printf("Selector Bit %u %u sent to ROW4\n", i + 1,
tx_frame.data);
        fflush(stdout);
    }
}
```

```

        BLUE_LED_OFF();
        delay(PATTERN_DELAY_IN_10ms);
    }
    BLUE_LED_OFF();
    while(1);
#elif(CONTROLLER == CONTROLLER_PROCESS_ELEMENTS)

    // configure lsm303dlhc accel
    lsm303dlhc_write_reg(LSM303DLHC_ACCEL_SLAVE_ADDR,
LSM303DLHC_ACCEL_CTRL_REG1_A, 0x27); // Data rate 10Hz
    // Normal mode (not low power)
    // X, Y, Z enabled

    // configure lsm303dlhc gyro
    lsm303dlhc_write_reg(LSM303DLHC_GYRO_SLAVE_ADDR,
LSM303DLHC_GYRO_CRA_REG_M, 0x08); // Data rate 3Hz,
Temperature sensor disabled.
    lsm303dlhc_write_reg(LSM303DLHC_GYRO_SLAVE_ADDR,
LSM303DLHC_GYRO_MR_REG_M, 0x00); // Continuous conversion mode

    // read identification
    printf("\nID1 should be 0x48, actual = 0x%x",
lsm303dlhc_read_reg(LSM303DLHC_GYRO_SLAVE_ADDR,
LSM303DLHC_GYRO_IRA_REG_M));
    printf("\nID2 should be 0x34, actual = 0x%x",
lsm303dlhc_read_reg(LSM303DLHC_GYRO_SLAVE_ADDR,
LSM303DLHC_GYRO_IRB_REG_M));
    printf("\nID3 should be 0x33, actual = 0x%x",
lsm303dlhc_read_reg(LSM303DLHC_GYRO_SLAVE_ADDR,
LSM303DLHC_GYRO_IRC_REG_M));

#ifdef LAST_SOUTH_CONTROLLER
    memset((void*)&north_prev, 0, sizeof(north_prev));
#endif
#ifdef LAST_EAST_CONTROLLER
    memset((void*)&west_prev, 0, sizeof(west_prev));
#endif
while(1)
{
    uint8_t selector_bit = 0;
    uint8_t instruction = 0;
    printf("wait I... \n");
    fflush(stdout);
    while(fifo_north.get_no_of_data_in_fifo() <sizeof(FRAME));
        // wait till at least the complete frame is received
    printf("wait s... \n");
    fflush(stdout);
    while(fifo_west.get_no_of_data_in_fifo() <sizeof(FRAME));
        // wait till at least the complete frame is received

    // copy bytes
    for(uint8_t i = 0; i<sizeof(FRAME); i++)
    {
        ((uint8_t*)&north_curr)[i] = fifo_north;
        ((uint8_t*)&west_curr)[i] = fifo_west;
    }

    // process north

```

```

printf("n.frame_type = %d ", north_curr.frame_type);
fflush(stdout);
printf("n.data = %d ", north_curr.data.data);
fflush(stdout);
switch(north_curr.frame_type)
{
    case FRAME_TYPE_DATA:
        controller_id = north_curr.data.data;
        // 2nd byte is data
        instruction = INST_NO_OPERATION;
        printf("controller_id = %u ", controller_id);
        fflush(stdout);
        break;
    case FRAME_TYPE_INSTRUCTION:
        instruction = north_curr.data.instruction;
        // 2nd byte is instruction
        memcpy(north_C, north_curr.C, sizeof(C));
        memcpy(north_N, north_curr.N, sizeof(N));
        memcpy(north_E, north_curr.E, sizeof(E));
        memcpy(north_W, north_curr.W, sizeof(W));
        memcpy(north_S, north_curr.S, sizeof(S));
        break;
    default:
        instruction = 0;
        break;
}

// process west
printf("w.frame_type = %d ", west_curr.frame_type);
fflush(stdout);
printf("w.data = %d ", west_curr.data.selector_bit);
fflush(stdout);
switch(west_curr.frame_type)
{
    case FRAME_TYPE_SELECTOR_BIT:
        selector_bit = west_curr.data.selector_bit;
        // 2nd byte is selector bit
        memcpy(west_C, west_curr.C, sizeof(C));
        memcpy(west_N, west_curr.N, sizeof(N));
        memcpy(west_E, west_curr.E, sizeof(E));
        memcpy(west_W, west_curr.W, sizeof(W));
        memcpy(west_S, west_curr.S, sizeof(S));
        break;
    default:
        selector_bit = 0;
        break;
}

if(selector_bit > 0)
{
    switch (instruction)
    {
        case INST_SENSOR_READ:
            sum_x = sum_y = sum_z = 0;
            for(uint8_t i = 0; i < NO_OF_SENSOR_READ;
i++)
            {
                lsm303dlhc_accel_read_xyz(x, y,
z);

```

```

        sum_x += x;
        sum_y += y;
        sum_z += z;
    }
    sum_x /= NO_OF_SENSOR_READ;
    sum_y /= NO_OF_SENSOR_READ;
    sum_z /= NO_OF_SENSOR_READ;
    sg[0] = sum_x;
    sg[1] = sum_y;
    sg[2] = sum_z;
    printf("ac_x = %d, ac_y = %d ac_z = %d
", sum_x, sum_y, sum_z);

    sum_x = sum_y = sum_z = 0;
    for(uint8_t i = 0; i < NO_OF_SENSOR_READ;
i++)
    {
        lsm303dlhc_gyro_read_xyz(x, y, z);
        sum_x += x;
        sum_y += y;
        sum_z += z;
    }
    sum_x /= NO_OF_SENSOR_READ;
    sum_y /= NO_OF_SENSOR_READ;
    sum_z /= NO_OF_SENSOR_READ;
    sm[0] = sum_x;
    sm[1] = sum_y;
    sm[2] = sum_z;
    printf("\ngy_x = %d, gy_y = %d gy_z = %d
", sum_x, sum_y, sum_z);

    fflush(stdout);
    break;
case INST_CALC_DIRECTIONAL_VECTOR:
    // calculate eg norm
    eg_norm[0] = eg[controller_id][0];
    eg_norm[1] = eg[controller_id][1];
    eg_norm[2] = eg[controller_id][2];
    algo_norm_3x1(eg_norm);

    // calculate em norm
    em_norm[0] = em[controller_id][0];
    em_norm[1] = em[controller_id][1];
    em_norm[2] = em[controller_id][2];
    algo_norm_3x1(em_norm);

    // calculate eg x em
    algo_cross_3x1(eg_norm, em_norm,
eg_x_em_norm);

    // calculate norm(eg x em)
    algo_norm_3x1(eg_x_em_norm);

    // calculate e1
    e1[0] = eg[controller_id][0];
    e1[1] = eg[controller_id][1];
    e1[2] = eg[controller_id][2];

    // calculate e2
    e2[0] = eg_x_em_norm[0];
    e2[1] = eg_x_em_norm[1];

```

```

e2[2] = eg_x_em_norm[2];

// calculate e3
algo_cross_3x1(e1, e2, e3);

// Copy to Me
Me[0][0] = e1[0];
Me[0][1] = e1[2];
Me[0][2] = e1[3];
Me[1][0] = e2[0];
Me[1][1] = e2[2];
Me[1][2] = e2[3];
Me[2][0] = e3[0];
Me[2][1] = e3[2];
Me[2][2] = e3[3];

// Calculate C
algo_norm_3x1(sg);
algo_norm_3x1(sm);
algo_cross_3x1(sg, sm, sg_x_sm);
algo_norm_3x1(sg_x_sm);
// s1=Sg; already satisfied
// s2 = sg_x_sm; already satisfied
//algo_cross_3x1(s1, s2, s3);
memcpy(s1, sg, sizeof(s1));
memcpy(s2, sg_x_sm, sizeof(s2));
algo_cross_3x1(s1, s2, s3);

//this is already done...
//Me=[e1 e2 e3]; //

will be 3x3 matrix

//Ms=[s1 s2 s3]; //

will be 3x3 matrix

Ms[0][0] = s1[0];
Ms[0][1] = s1[2];
Ms[0][2] = s1[3];
Ms[1][0] = s2[0];
Ms[1][1] = s2[2];
Ms[1][2] = s2[3];
Ms[2][0] = s3[0];
Ms[2][1] = s3[2];
Ms[2][2] = s3[3];
algo_transpose_3x3(Ms);
algo_matrix_mul_3x3_and_3x3(Me, Ms, R);

// calculate directional vectors
algo_matrix_mul_3x3_and_3x1(R, Nb, N);
algo_matrix_mul_3x3_and_3x1(R, Eb, E);
algo_matrix_mul_3x3_and_3x1(R, Sb, S);
algo_matrix_mul_3x3_and_3x1(R, Wb, W);
printf("directional vector calculation

done");

fflush(stdout);
break;
case INST_DOWN_TAIL:
printf("INST_DOWN_TAIL");
break;
case INST_DOWN_HEAD:

```

```

        printf("INST_DOWN_HEAD");
        break;
    case INST_RIGHT_TAIL:
        printf("INST_RIGHT_TAIL");
        break;
    case INST_RIGHT_HEAD:
        printf("INST_RIGHT_HEAD");
        break;
    case INST_CALC_CONTROL_POINT:
        calculate_control_point(controller_id);
        printf("INST_CALC_CONTROL_POINT");
        break;
    case INST_TRANSFER:
        printf("C[0]*100 = %d, C[1]*100 = %d
C[2]*100 = %d ", (int)(C[0] * 100), (int)(C[1] * 100), (int)(C[2] *
100));
        break;
    }
}
#endif
// send the prev data register & instruction through south
master
    SetupXferRecAndExecute(I2C_SOUTH_MASTER, I2C_SLAVE_ADDR,
(uint8_t*)&north_prev, sizeof(north_prev), 0, 0);
    memcpy(north_curr.C, C, sizeof(C));
    memcpy(north_curr.N, N, sizeof(N));
    memcpy(north_curr.E, E, sizeof(E));
    memcpy(north_curr.W, W, sizeof(W));
    memcpy(north_curr.S, S, sizeof(S));
    memcpy((void*)&north_prev, (void*)&north_curr,
sizeof(north_curr));
#endif

#ifdef LAST_EAST_CONTROLLER
// send the prev data register & selector bit through east
master
    SetupXferRecAndExecute(I2C_EAST_MASTER, I2C_SLAVE_ADDR,
(uint8_t*)&west_prev, sizeof(west_prev), 0, 0);
    memcpy(west_curr.C, C, sizeof(C));
    memcpy(west_curr.N, N, sizeof(N));
    memcpy(west_curr.E, E, sizeof(E));
    memcpy(west_curr.W, W, sizeof(W));
    memcpy(west_curr.S, S, sizeof(S));
    memcpy((void*)&west_prev, (void*)&west_curr,
sizeof(west_curr));
#endif
    printf("\n");
}
#endif
return 0 ;
}

```

```

/*****
*****
* global defines header file
*****
*****/

#include "chip.h"

#define VERSION_STRING
    "ShapeRecon Version - 1.13\n"

#define CONTROLLER_INSTRUCTION          1
#define CONTROLLER_SELECTOR_BIT        2
#define CONTROLLER_PROCESS_ELEMENTS    3

// Configure this controller
// #define CONTROLLER
//     CONTROLLER_INSTRUCTION
// #define CONTROLLER
//     CONTROLLER_SELECTOR_BIT
#define CONTROLLER
    CONTROLLER_PROCESS_ELEMENTS

#define DEBUG_UART
    LPC_USART1

#if (CONTROLLER == CONTROLLER_PROCESS_ELEMENTS)
// #define LAST_SOUTH_CONTROLLER          1
//     // comment any one of these if its last
controller
// #define LAST_EAST_CONTROLLER          1

#define I2C_SOUTH_MASTER
    LPC_I2C0
#define I2C_EAST_MASTER
    LPC_I2C3
#define I2C_NORTH_SLAVE
    LPC_I2C2
#define I2C_WEST_SLAVE
    LPC_I2C1

#define I2C_SOUTH_MASTER_IRQHandler
    I2C0_IRQHandler
#define I2C_EAST_MASTER_IRQHandler
    I2C3_IRQHandler
#define I2C_NORTH_SLAVE_IRQHandler
    I2C2_IRQHandler
#define I2C_WEST_SLAVE_IRQHandler
    I2C1_IRQHandler
#elif (CONTROLLER == CONTROLLER_INSTRUCTION)
#define I2C_COL1_MASTER
    LPC_I2C2
#define I2C_COL2_MASTER
    LPC_I2C3
#define I2C_COL3_MASTER
    LPC_I2C0

```



```
#define I2C_COL4_MASTER
    LPC_I2C1

#define I2C_COL1_MASTER_IRQHandler
    I2C2_IRQHandler
#define I2C_COL2_MASTER_IRQHandler
    I2C3_IRQHandler
#define I2C_COL3_MASTER_IRQHandler
    I2C0_IRQHandler
#define I2C_COL4_MASTER_IRQHandler
    I2C1_IRQHandler
#elif(CONTROLLER == CONTROLLER_SELECTOR_BIT)
#define I2C_ROW1_MASTER
    LPC_I2C2
#define I2C_ROW2_MASTER
    LPC_I2C3
#define I2C_ROW3_MASTER
    LPC_I2C0
#define I2C_ROW4_MASTER
    LPC_I2C1

#define I2C_ROW1_MASTER_IRQHandler
    I2C2_IRQHandler
#define I2C_ROW2_MASTER_IRQHandler
    I2C3_IRQHandler
#define I2C_ROW3_MASTER_IRQHandler
    I2C0_IRQHandler
#define I2C_ROW4_MASTER_IRQHandler
    I2C1_IRQHandler
#endif

#define I2C_SLAVE_ADDR                                0x55
#define LSM303DLHC_ACCEL_SLAVE_ADDR                  0b0011001
#define LSM303DLHC_GYRO_SLAVE_ADDR                   0b0011110

#define SPEED_100KHZ                                  100000
#define SPEED_400KHZ                                  400000

#define I2C_SPEED
    SPEED_400KHZ

#define MAIN_OSC_CRYSTAL
    12000000
#define RTC_OSC_CRYSTAL
    32768

#define PIN_GREEN_LED                                  16
#define PIN_BLUE_LED                                  27
// #define PIN_RED_LED
    12

// #define RED_LED_OFF()
    Chip_GPIO_SetPinOutHigh(LPC_GPIO_PORT, 0, PIN_RED_LED)
// #define RED_LED_ON()
    Chip_GPIO_SetPinOutLow(LPC_GPIO_PORT, 0, PIN_RED_LED)
// #define RED_LED_TOGGLE()
    Chip_GPIO_SetPinToggle(LPC_GPIO_PORT, 0, PIN_RED_LED)
#define GREEN_LED_OFF()
    Chip_GPIO_SetPinOutHigh(LPC_GPIO_PORT, 0, PIN_GREEN_LED)
```

```

#define GREEN_LED_ON()
    Chip_GPIO_SetPinOutLow(LPC_GPIO_PORT, 0, PIN_GREEN_LED)
#define GREEN_LED_TOGGLE()
    Chip_GPIO_SetPinToggle(LPC_GPIO_PORT, 0, PIN_GREEN_LED)
#define BLUE_LED_OFF()
    Chip_GPIO_SetPinOutHigh(LPC_GPIO_PORT, 0, PIN_BLUE_LED)
#define BLUE_LED_ON()
    Chip_GPIO_SetPinOutLow(LPC_GPIO_PORT, 0, PIN_BLUE_LED)
#define BLUE_LED_TOGGLE()
    Chip_GPIO_SetPinToggle(LPC_GPIO_PORT, 0, PIN_BLUE_LED)

//LSM303DLHC registers
#define LSM303DLHC_ACCEL_CTRL_REG1_A          0x20
#define LSM303DLHC_ACCEL_CTRL_REG2_A          0x21
#define LSM303DLHC_ACCEL_CTRL_REG3_A          0x22
#define LSM303DLHC_ACCEL_CTRL_REG4_A          0x23
#define LSM303DLHC_ACCEL_CTRL_REG5_A          0x24
#define LSM303DLHC_ACCEL_CTRL_REG6_A          0x25
#define LSM303DLHC_ACCEL_REFERENCE_A          0x26
#define LSM303DLHC_ACCEL_STATUS_REG_A         0x27
#define LSM303DLHC_ACCEL_OUT_X_L_A            0x28
#define LSM303DLHC_ACCEL_OUT_X_H_A            0x29
#define LSM303DLHC_ACCEL_OUT_Y_L_A            0x2A
#define LSM303DLHC_ACCEL_OUT_Y_H_A            0x2B
#define LSM303DLHC_ACCEL_OUT_Z_L_A            0x2C
#define LSM303DLHC_ACCEL_OUT_Z_H_A            0x2D
#define LSM303DLHC_ACCEL_FIFO_CTRL_REG_A      0x2E
#define LSM303DLHC_ACCEL_FIFO_SRC_REG_A       0x2F
#define LSM303DLHC_ACCEL_INT1_CFG_A           0x30
#define LSM303DLHC_ACCEL_INT1_SRC_A           0x31
#define LSM303DLHC_ACCEL_INT1_THS_A           0x32
#define LSM303DLHC_ACCEL_INT1_DURATION_A      0x33
#define LSM303DLHC_ACCEL_INT2_CFG_A           0x34
#define LSM303DLHC_ACCEL_INT2_SRC_A           0x35
#define LSM303DLHC_ACCEL_INT2_THS_A           0x36
#define LSM303DLHC_ACCEL_INT2_DURATION_A      0x37
#define LSM303DLHC_ACCEL_CLICK_CFG_A          0x38
#define LSM303DLHC_ACCEL_CLICK_SRC_A          0x39
#define LSM303DLHC_ACCEL_CLICK_THS_A          0x3A

#define LSM303DLHC_GYRO_CRA_REG_M              0x00
#define LSM303DLHC_GYRO_CRB_REG_M              0x01
#define LSM303DLHC_GYRO_MR_REG_M              0x02
#define LSM303DLHC_GYRO_OUT_X_H_M              0x03
#define LSM303DLHC_GYRO_OUT_X_L_M              0x04
#define LSM303DLHC_GYRO_OUT_Z_H_M              0x05
#define LSM303DLHC_GYRO_OUT_Z_L_M              0x06
#define LSM303DLHC_GYRO_OUT_Y_H_M              0x07
#define LSM303DLHC_GYRO_OUT_Y_L_M              0x08
#define LSM303DLHC_GYRO_SR_REG_M              0x09
#define LSM303DLHC_GYRO_IRA_REG_M              0x0A
#define LSM303DLHC_GYRO_IRB_REG_M              0x0B
#define LSM303DLHC_GYRO_IRC_REG_M              0x0C
#define LSM303DLHC_GYRO_TEMP_OUT_H_M           0x31
#define LSM303DLHC_GYRO_TEMP_OUT_L_M           0x32

#define rows_of_array(name) \

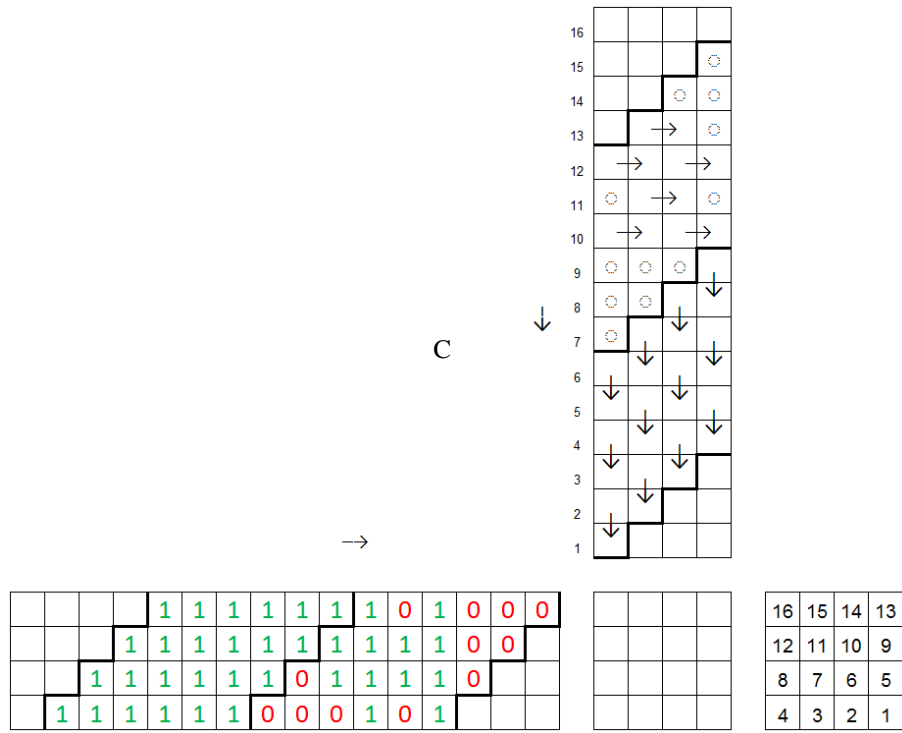
```

```
(sizeof(name ) / sizeof(name[0][0]) / columns_of_array(name))
#define columns_of_array(name) \
    (sizeof(name[0]) / sizeof(name[0][0]))

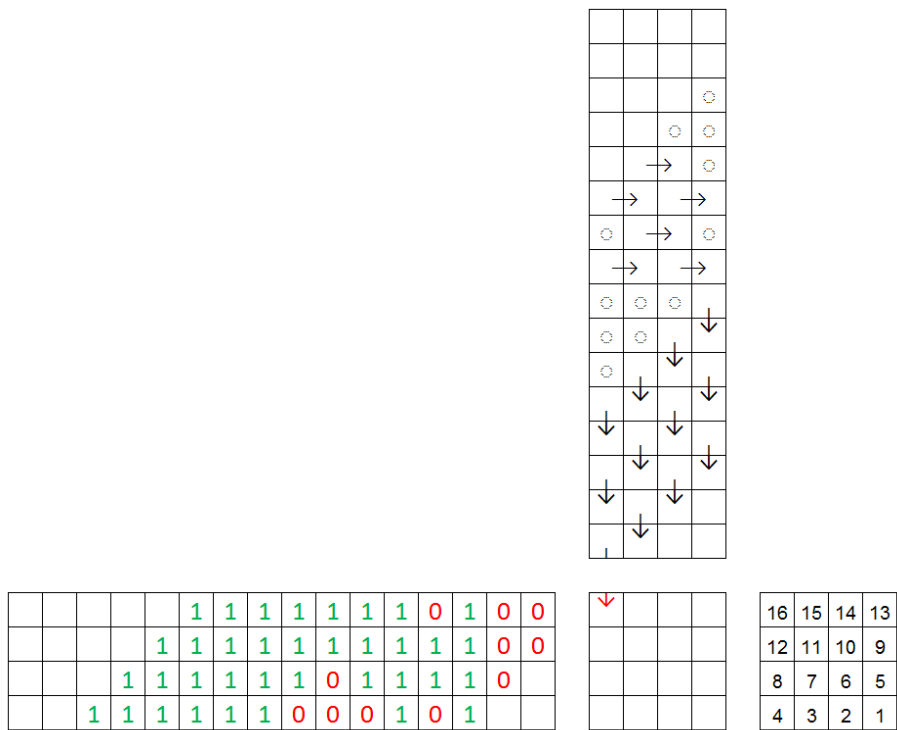
#ifdef GLOBALS
#define EXT
#else
#define EXT extern
#endif

EXT const uint32_t OscRateIn
#ifdef GLOBALS
= MAIN_OSC_CRYSTAL
#endif
;
EXT const uint32_t RTCOscRateIn
#ifdef GLOBALS
= RTC_OSC_CRYSTAL
#endif
;
```

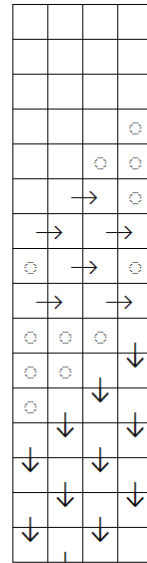
Implementation of Merge Algorithm



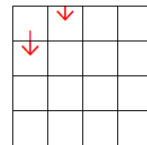
(1)



(2)

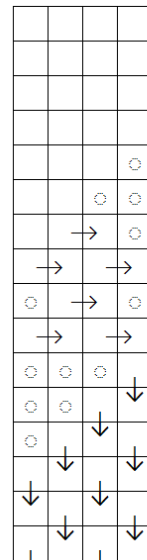


					1	1	1	1	1	1	1	1	0	1	0
					1	1	1	1	1	1	1	1	1	1	0
					1	1	1	1	1	1	0	1	1	1	0
					1	1	1	1	1	1	0	0	0	1	0

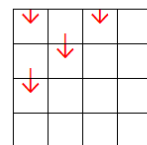


16	15	14	13
12	11	10	9
8	7	6	5
4	3	2	1

(3)

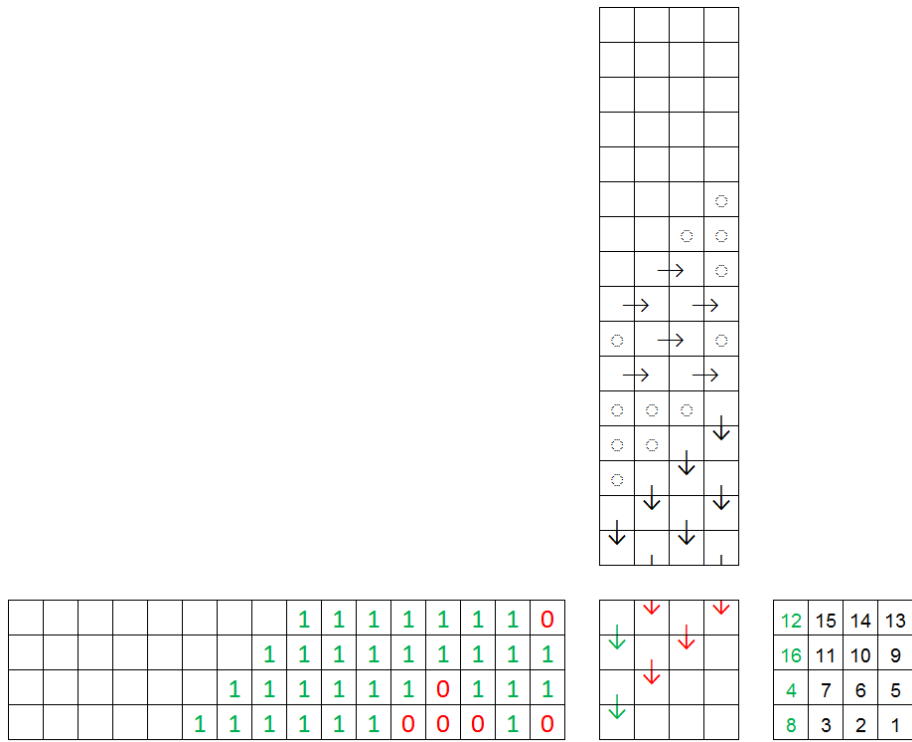


					1	1	1	1	1	1	1	1	0	1	1
					1	1	1	1	1	1	1	1	1	1	1
					1	1	1	1	1	1	0	1	1	1	1
					1	1	1	1	1	1	0	0	0	1	0

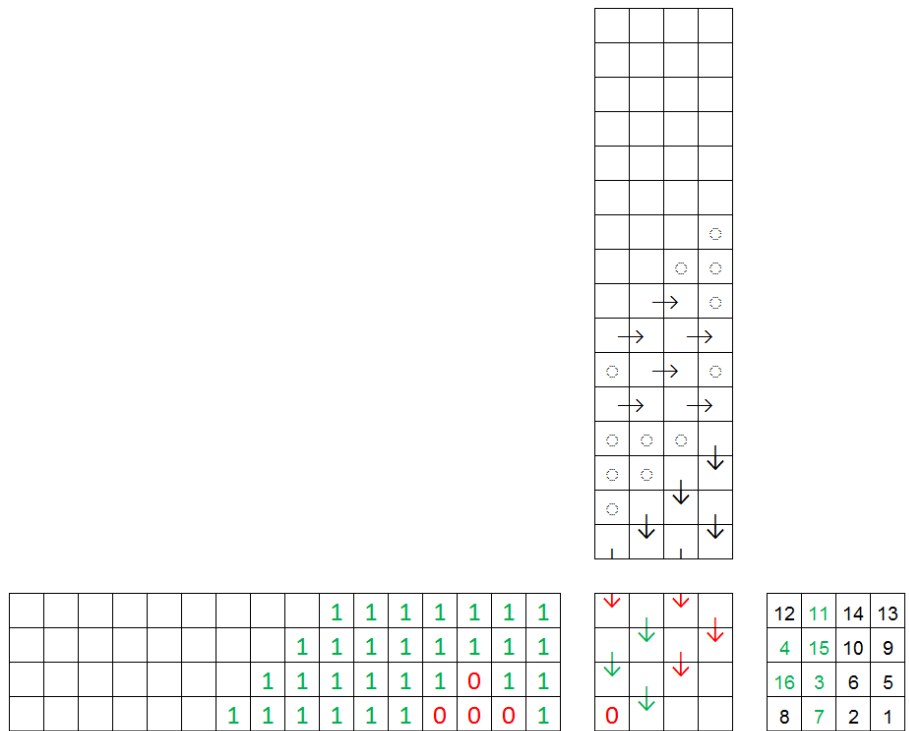


16	15	14	13
12	11	10	9
8	7	6	5
4	3	2	1

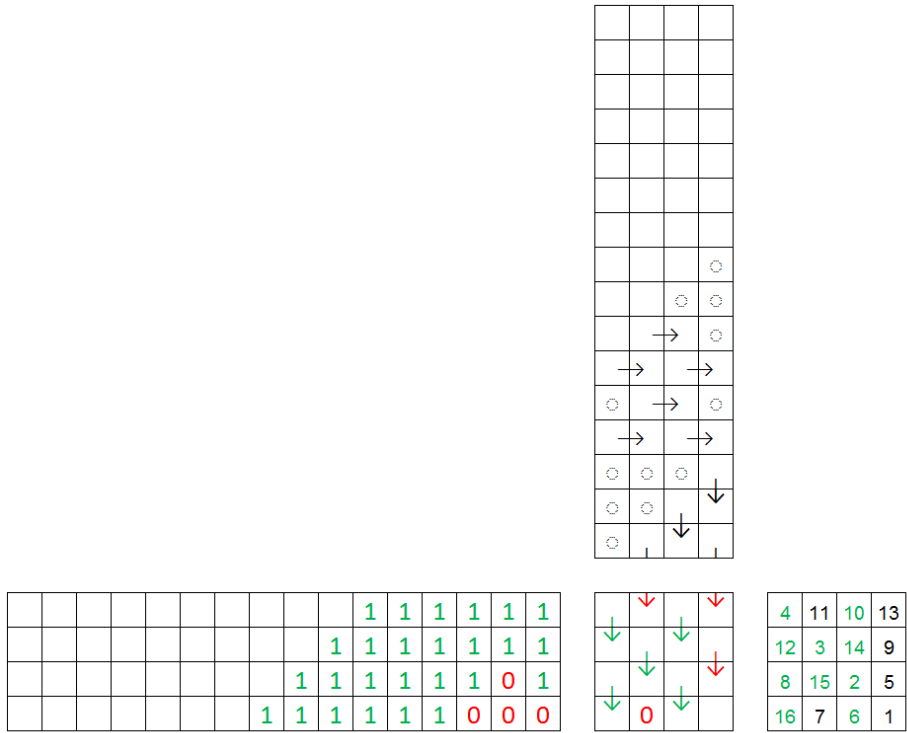
(4)



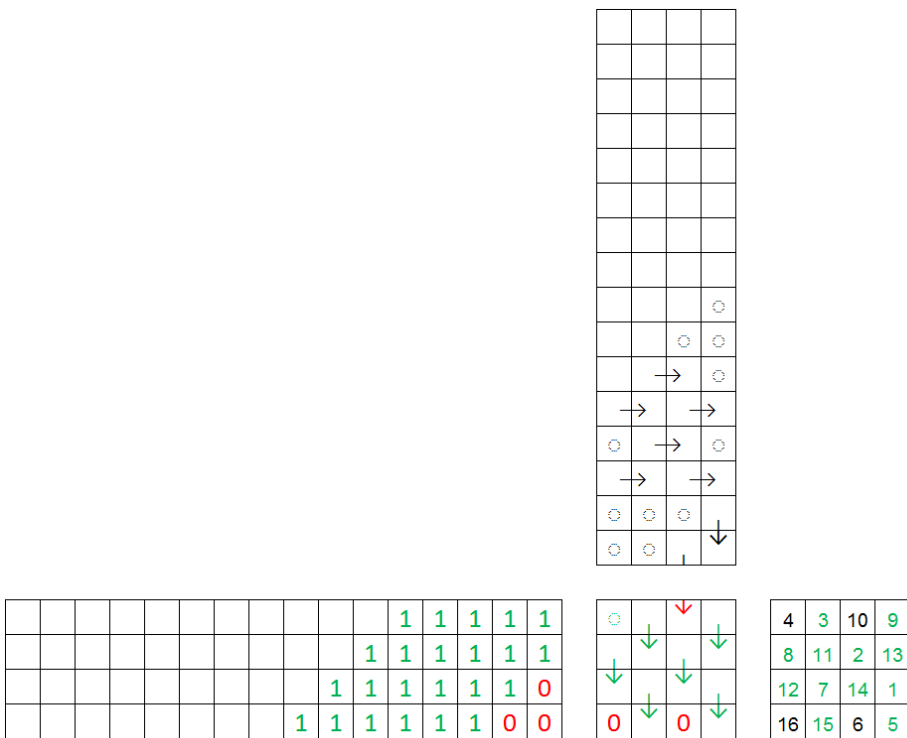
(5)



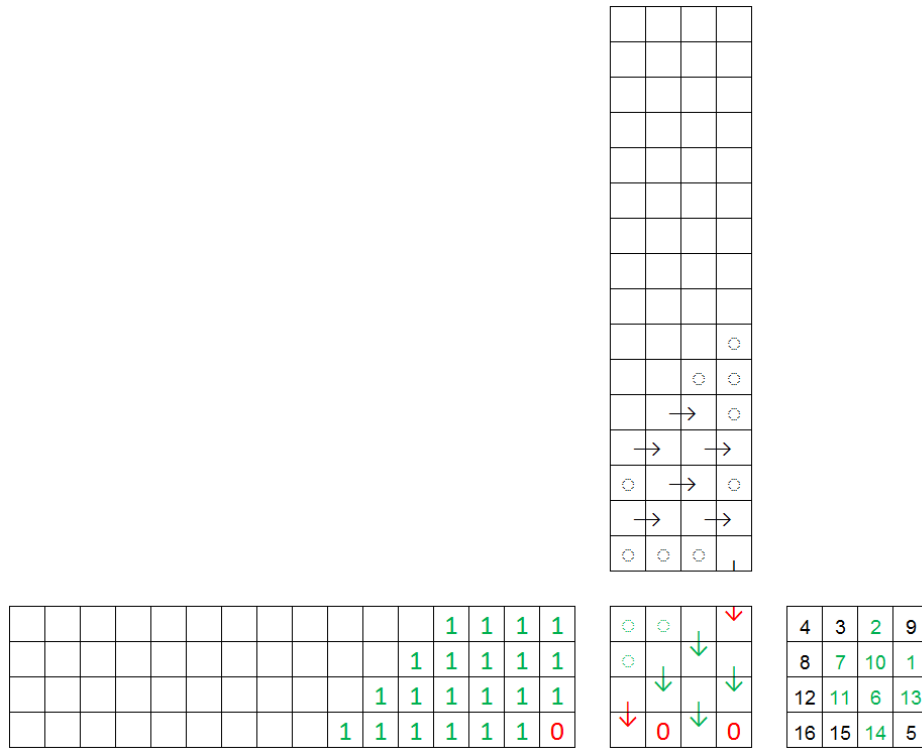
(6)



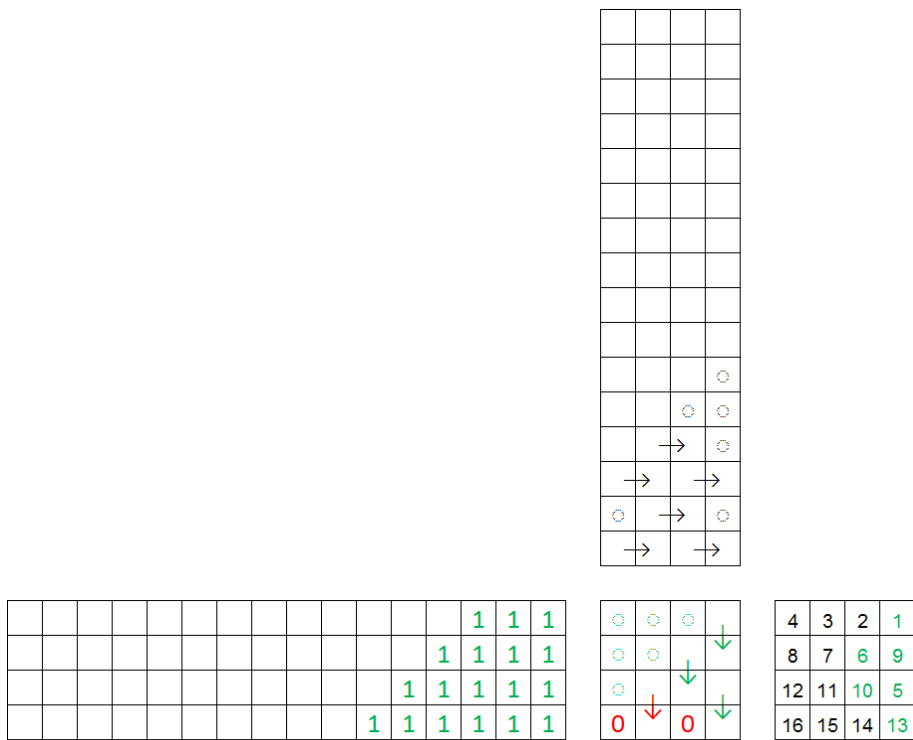
(7)



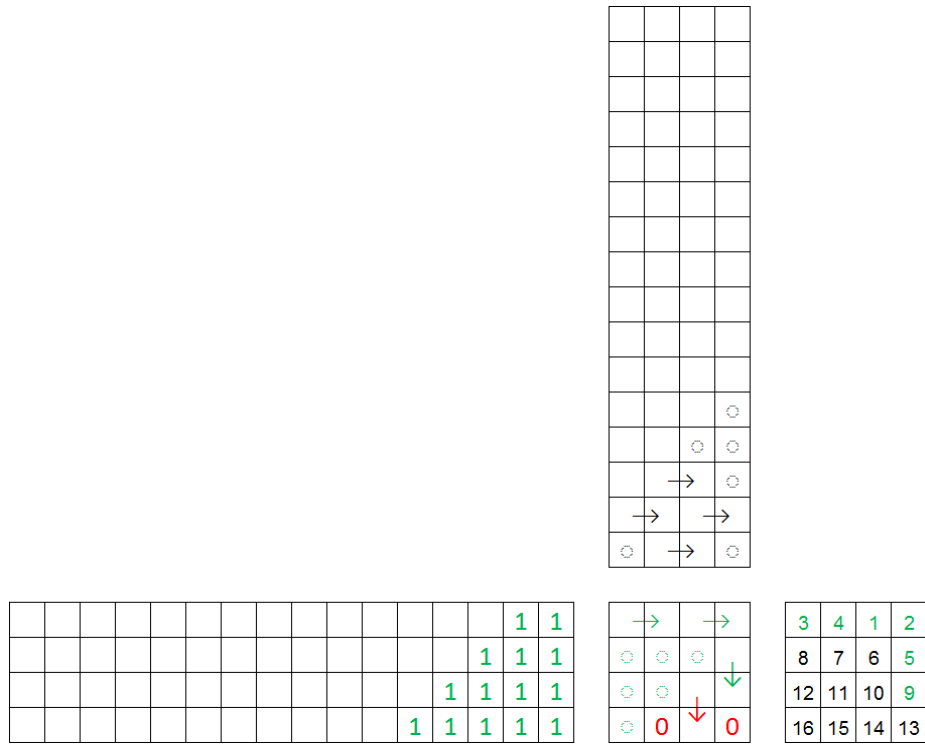
(8)



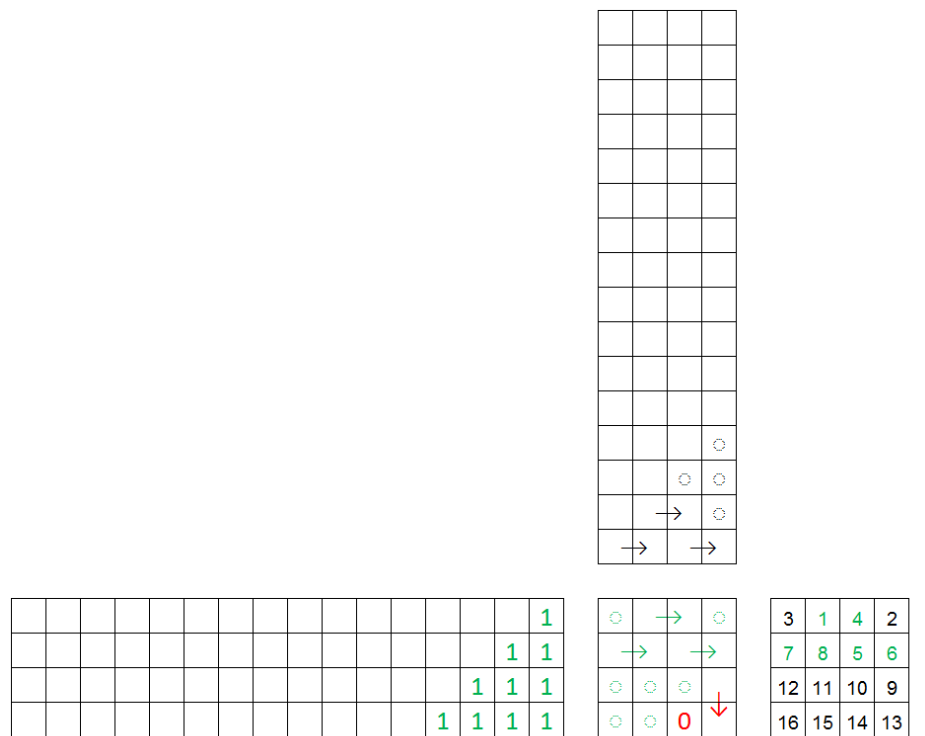
(9)



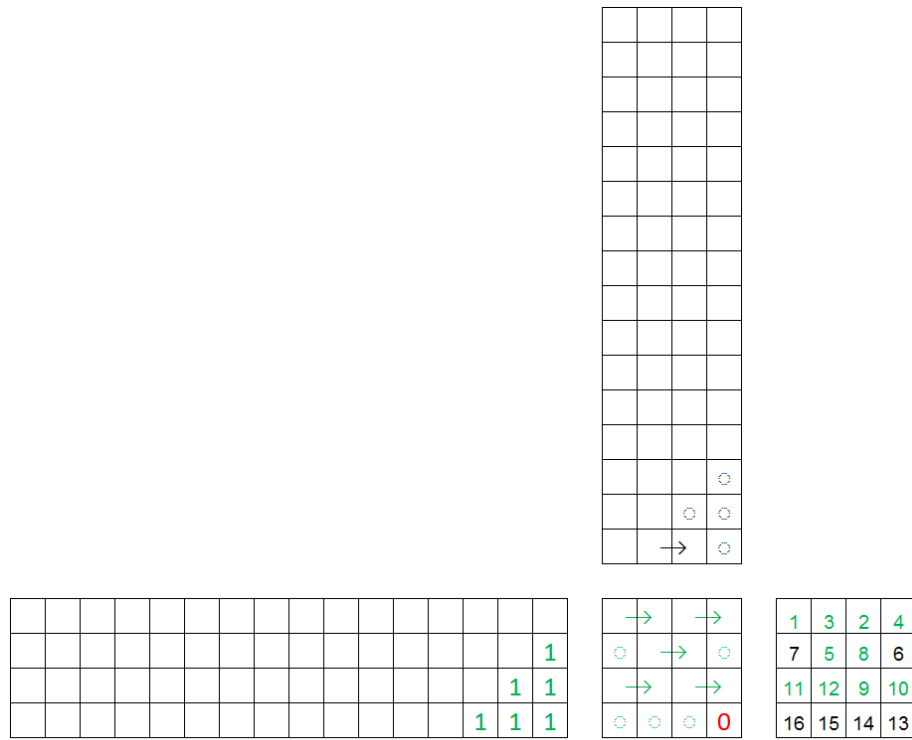
(10)



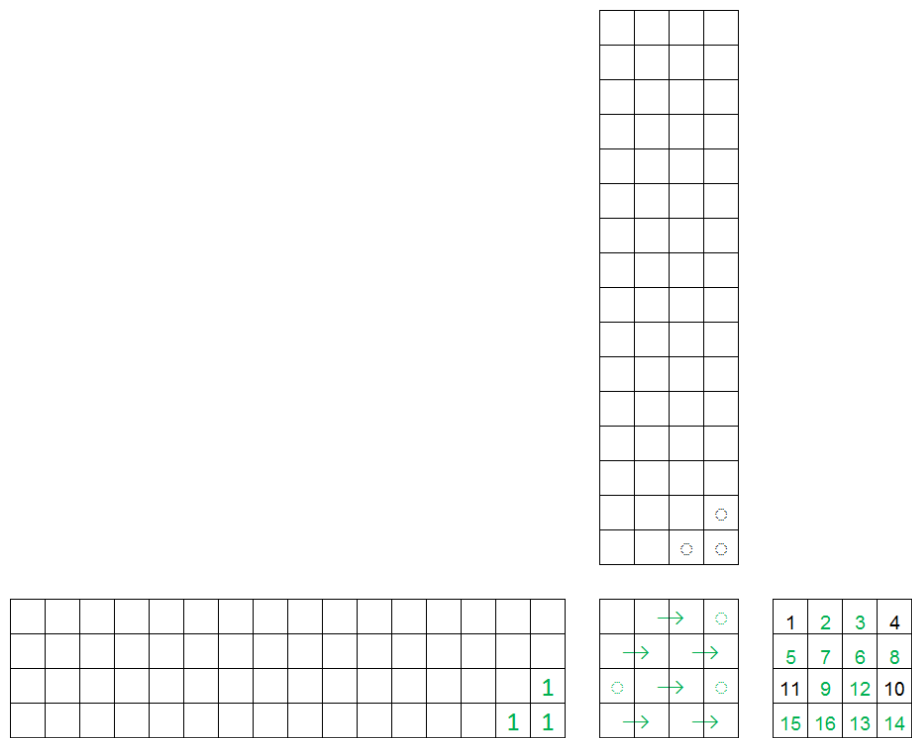
(11)



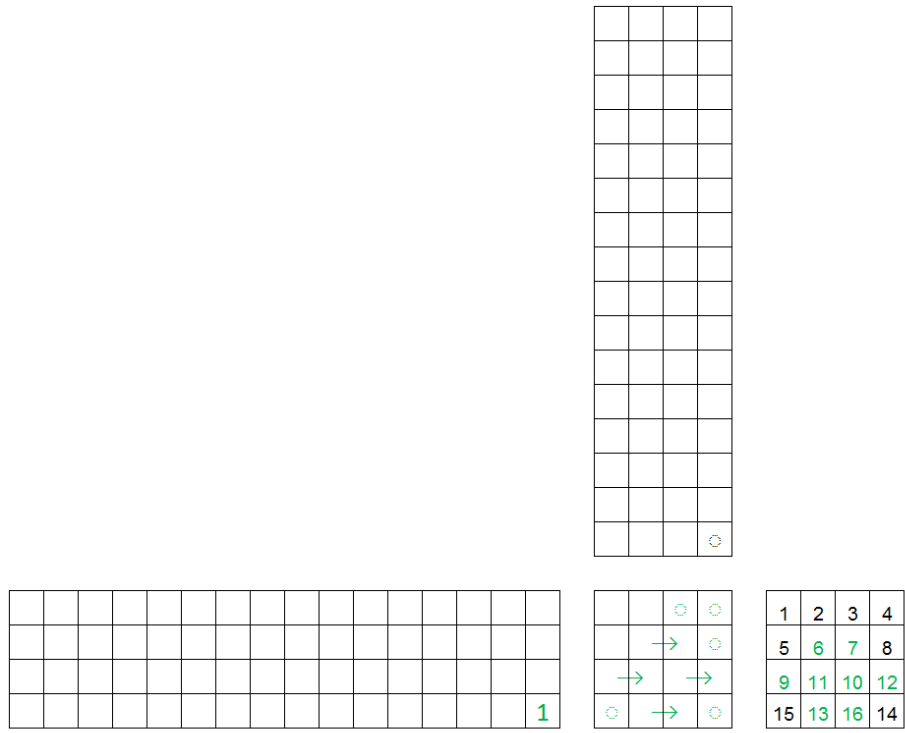
(12)



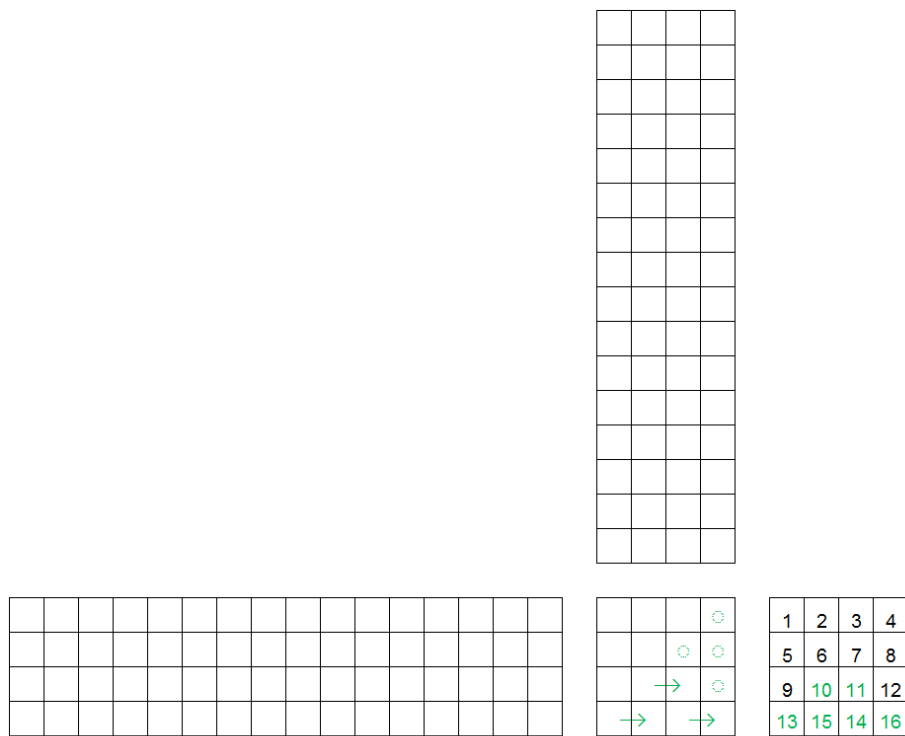
(13)



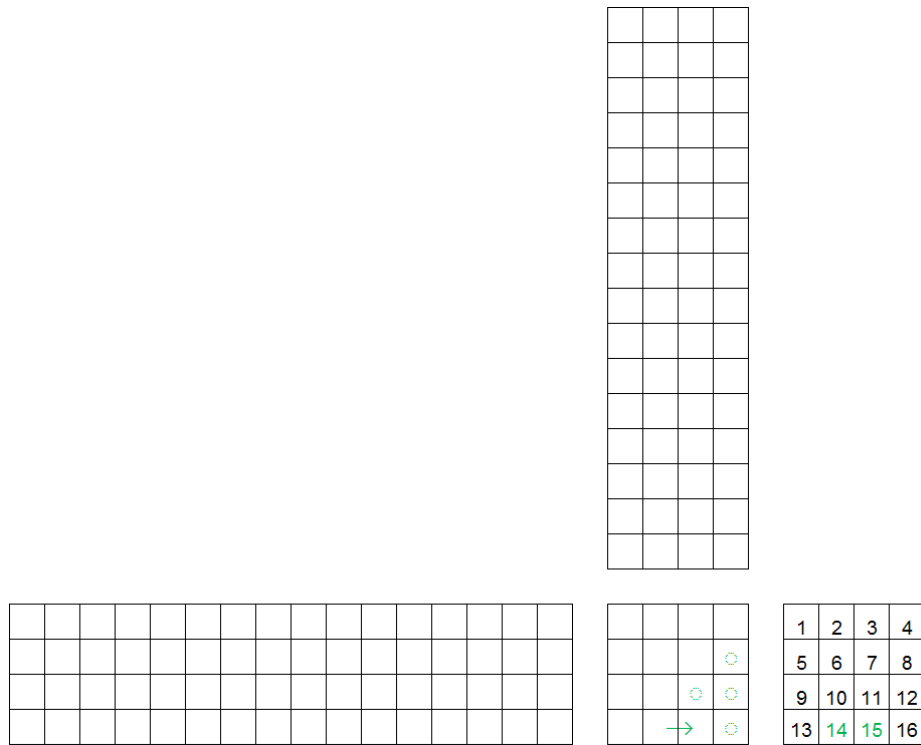
(14)



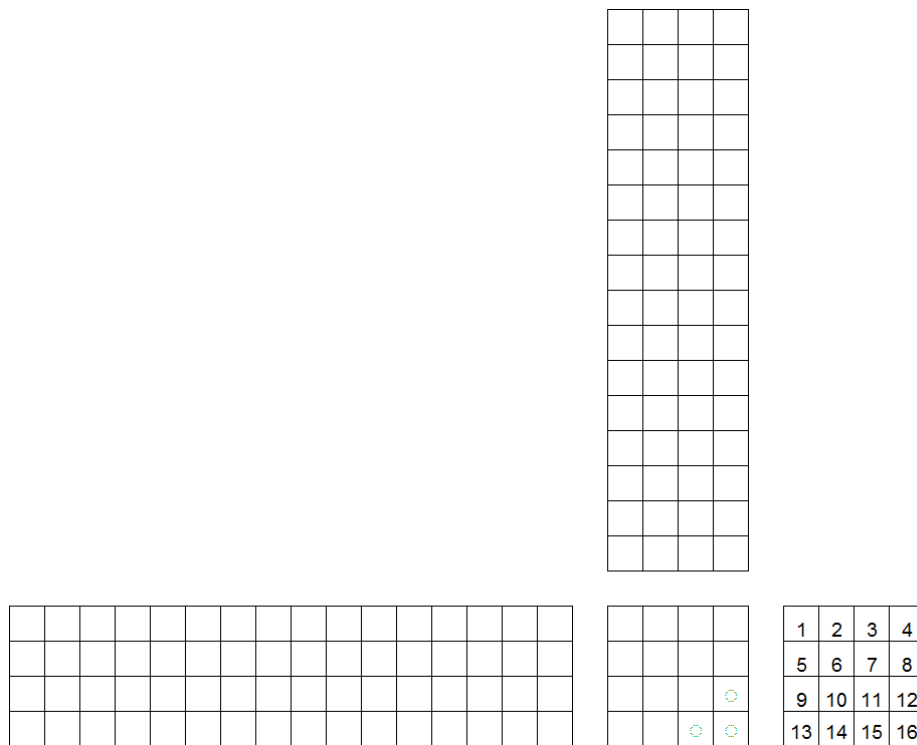
(15)



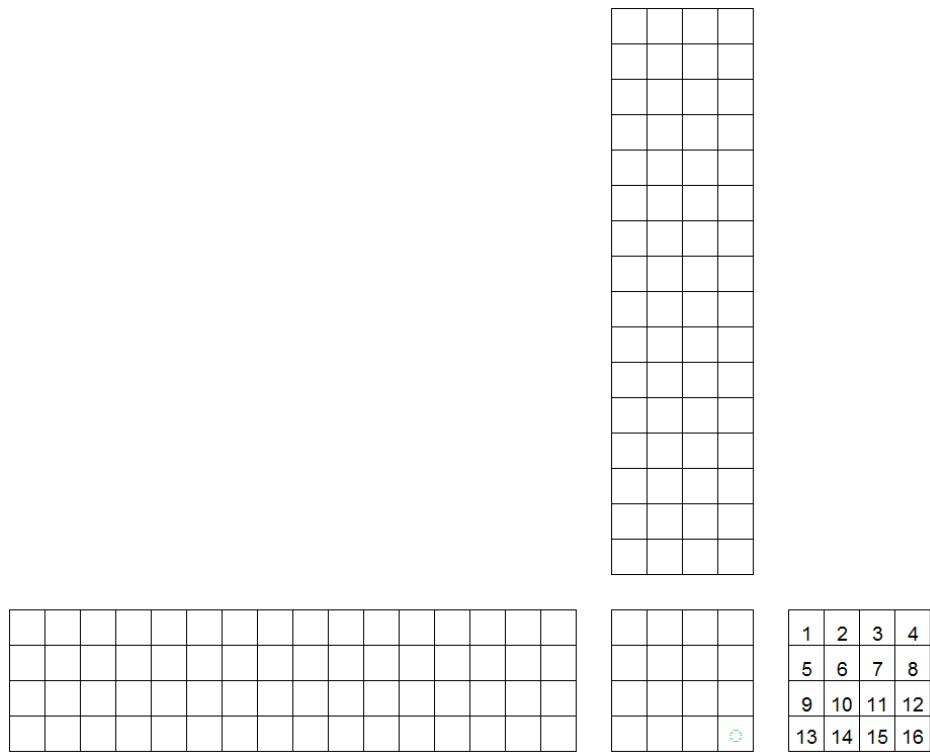
(16)



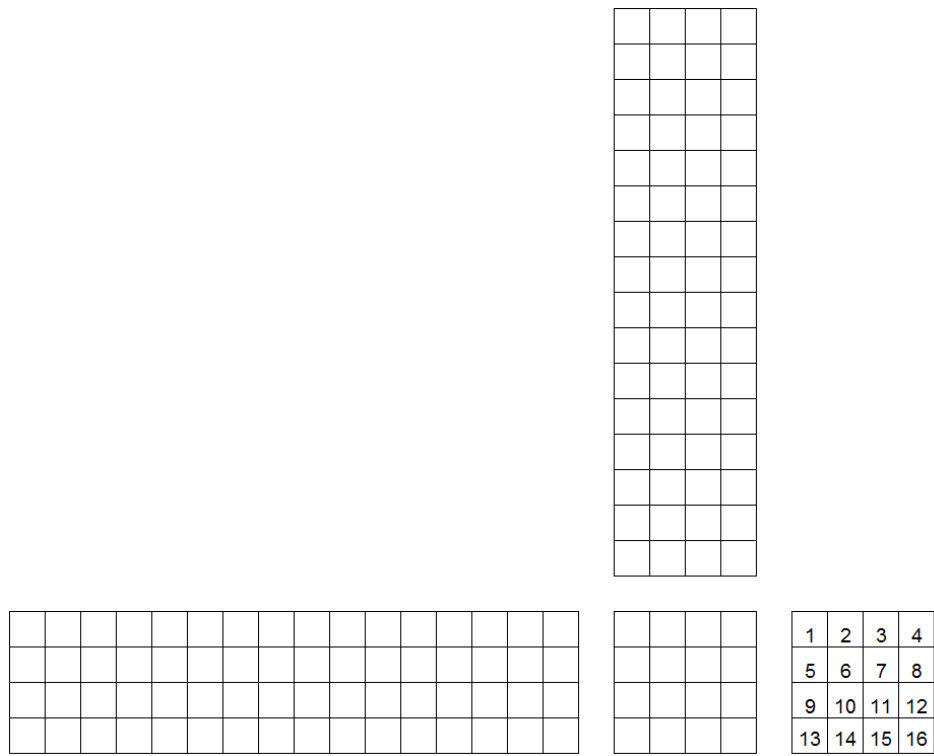
(17)



(18)

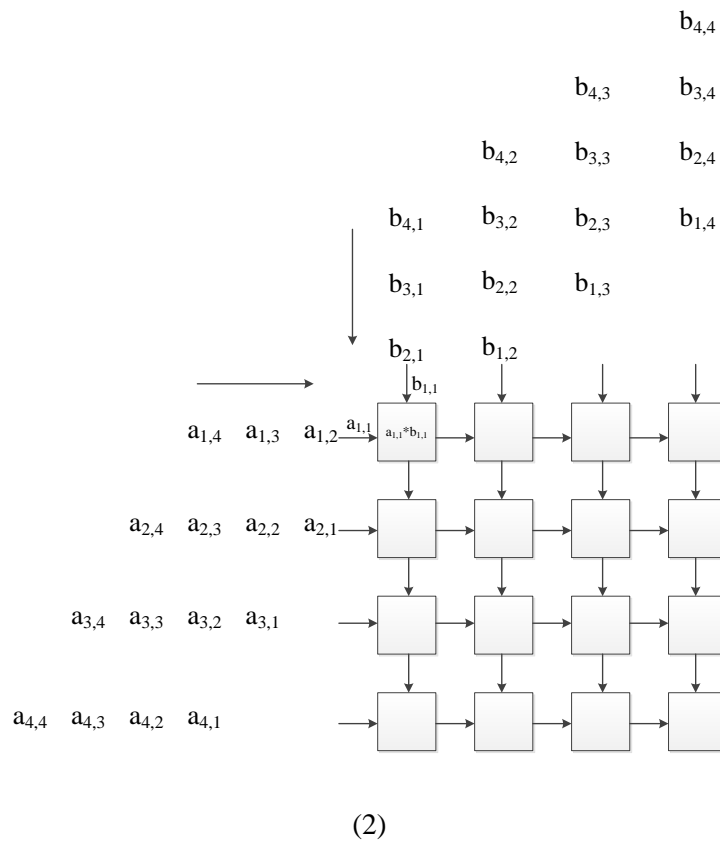
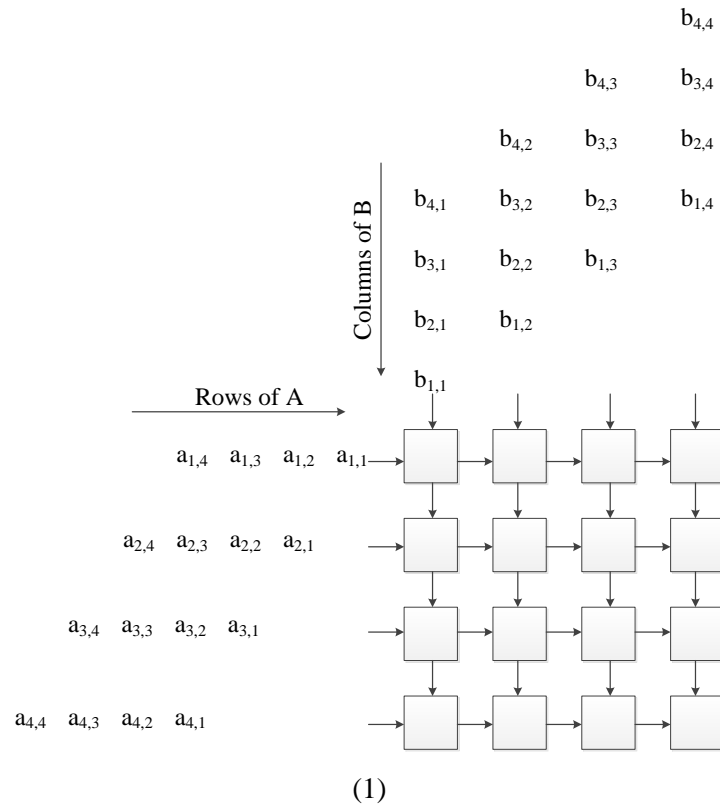


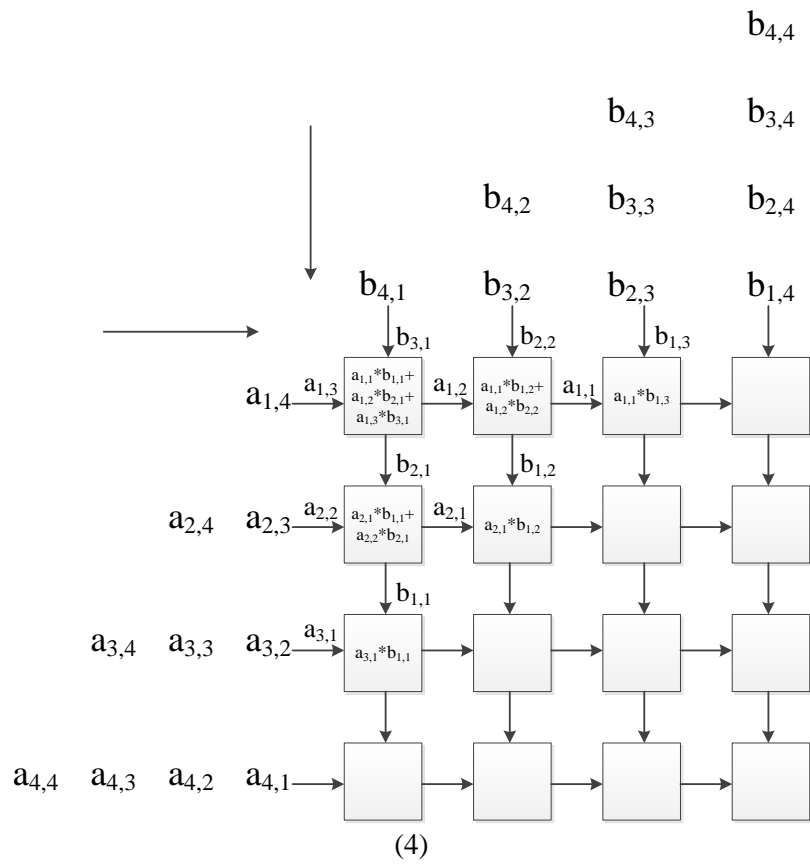
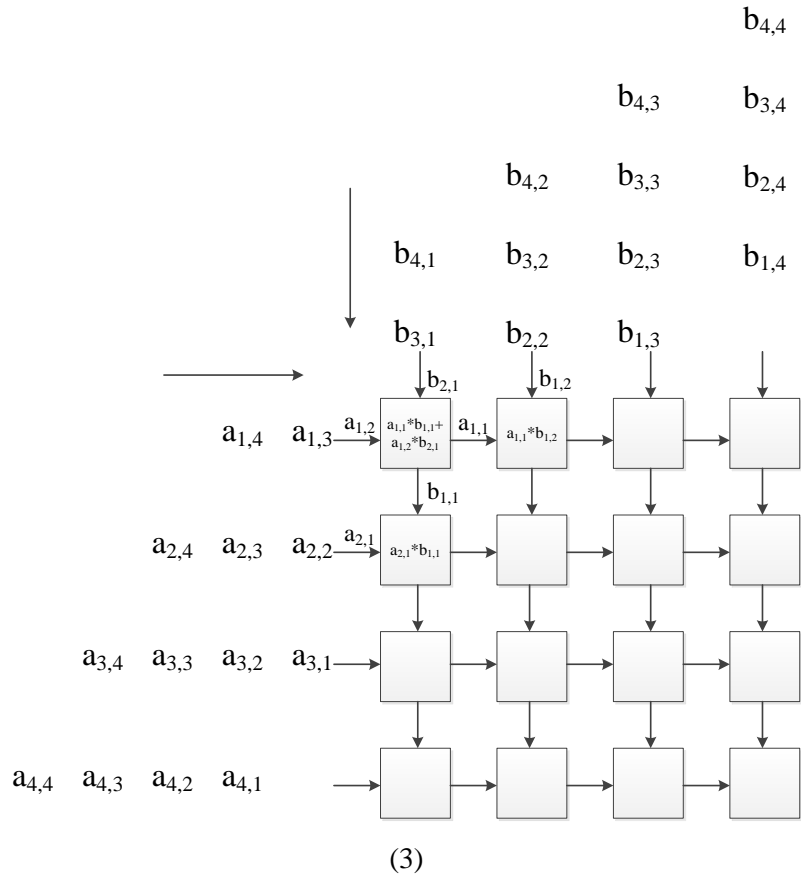
(19)

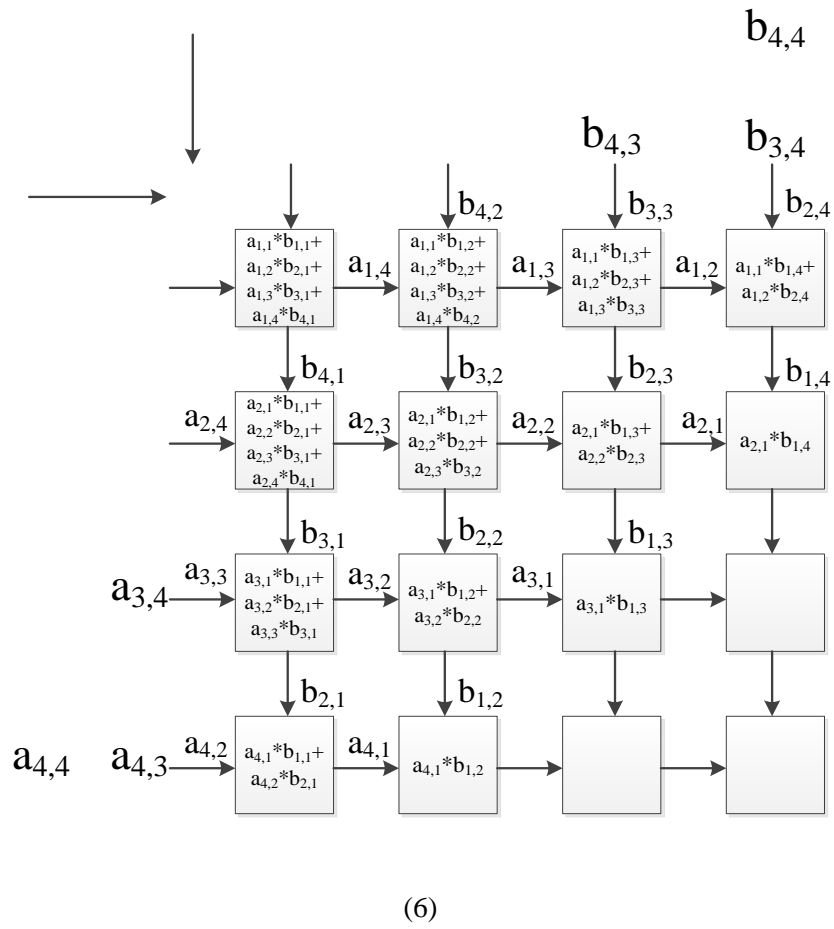
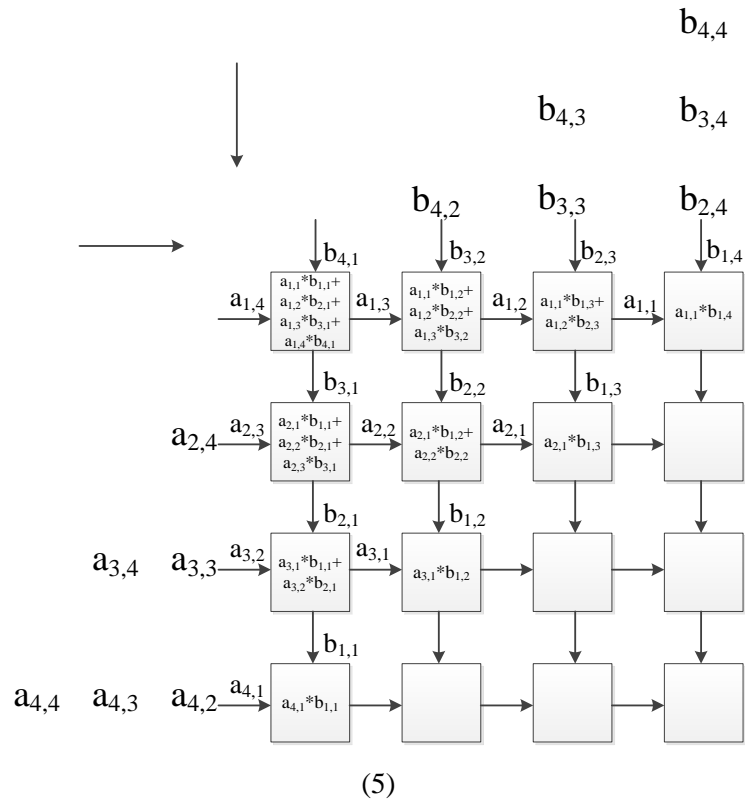


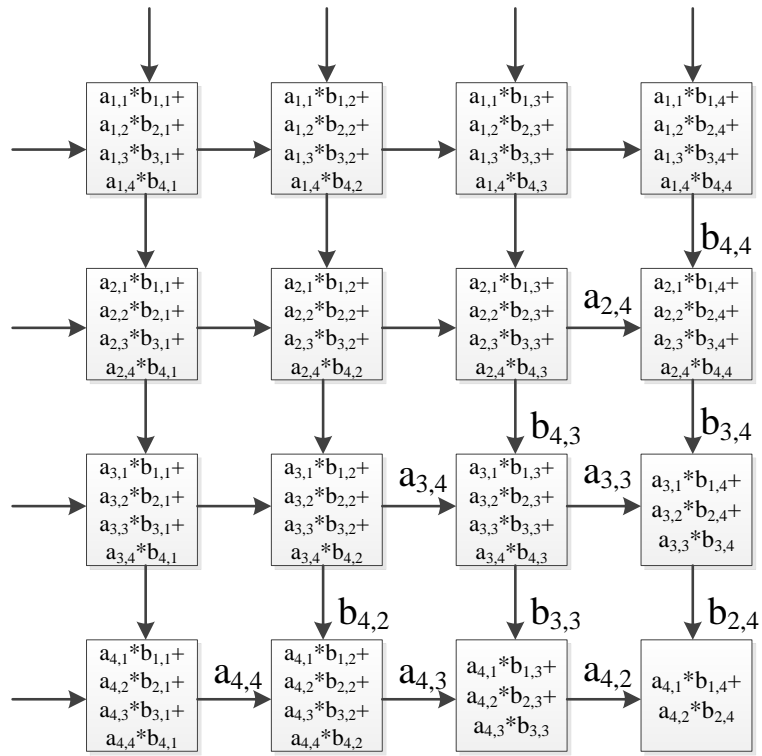
(20)

Matrix Multiplication Algorithm

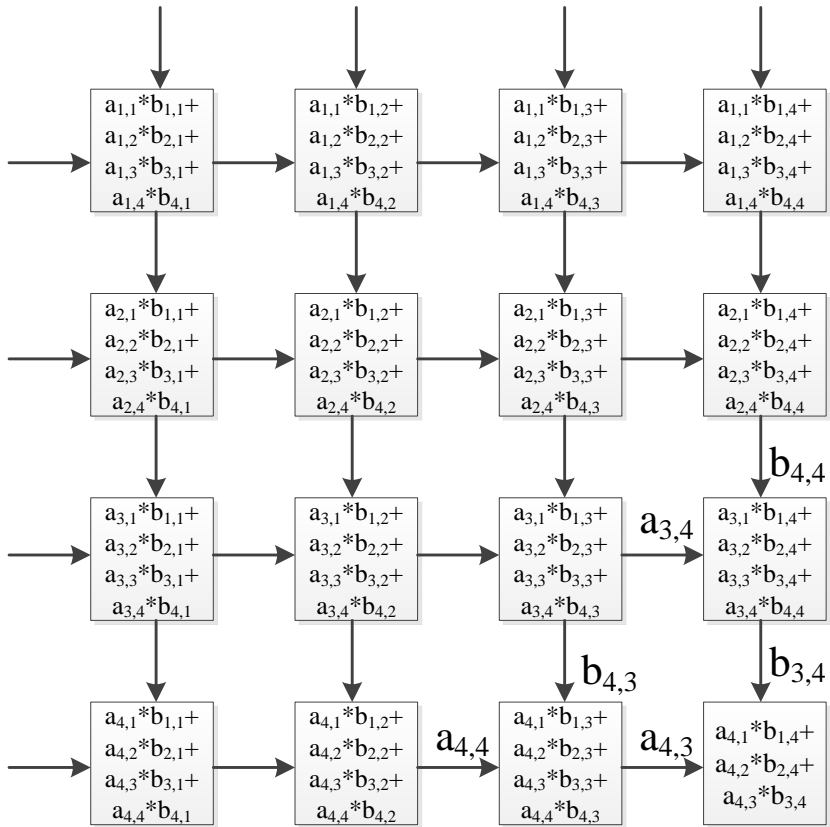




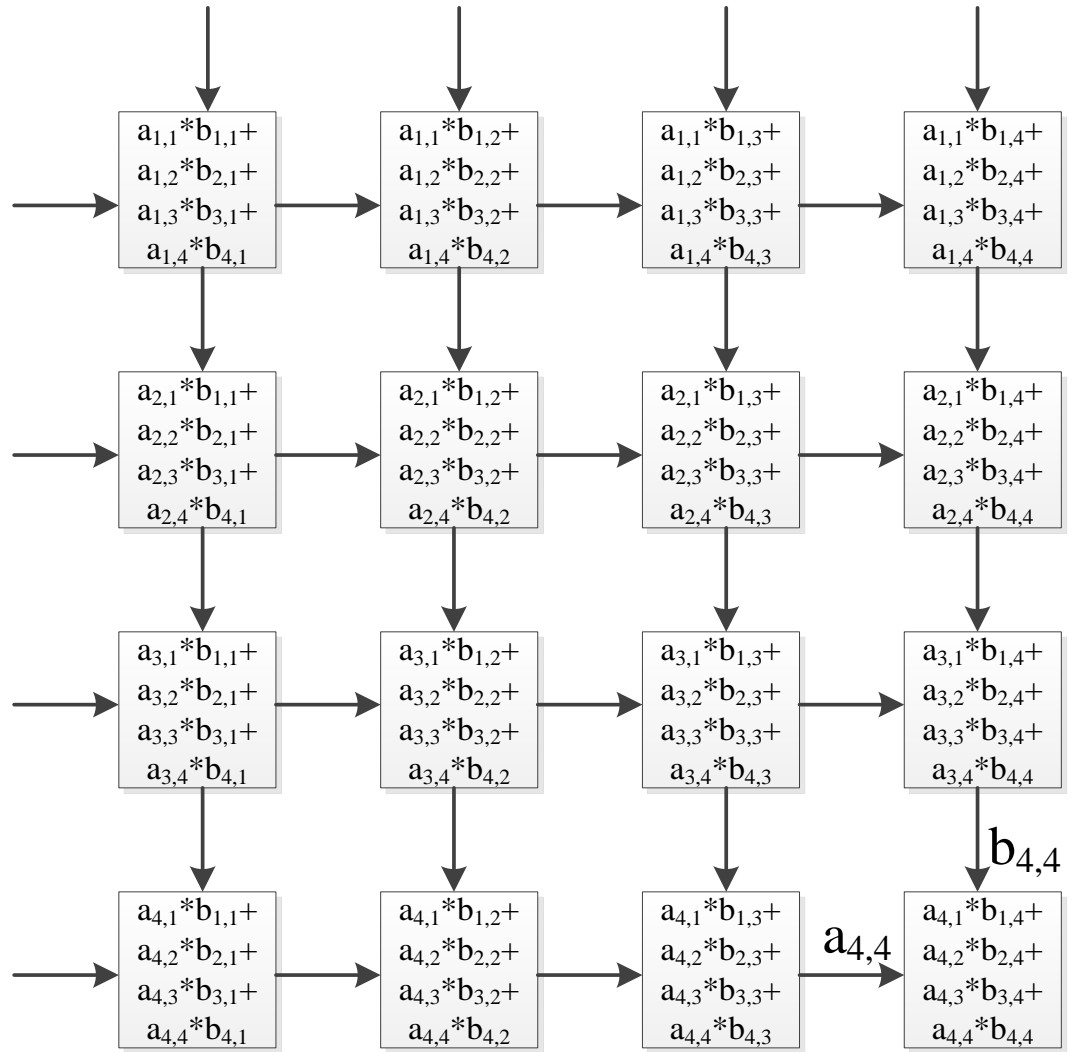




(9)



(10)



(11)

Matrix Multiplication Implementation

			+
		+	*
	+	*	1
+	*	2	+
*	3	+	*
4	+	*	5
+	*	6	+
*	7	+	*
8	+	*	9
+	*	10	*
*	11	*	13
12	*	14	
*	15		
16			

		1	1	4	1	1	3	1	1	2	1	1
		1	1	8	1	1	7	1	1	6	1	5
		1	1	12	1	1	11	1	1	10	1	9
1	1	16	1	1	15	1	1	14	1	13		

(1)

			+
		+	*
	+	*	1
+	*	2	+
*	3	+	*
4	+	*	5
+	*	6	+
*	7	+	*
8	+	*	9
+	*	10	*
*	11	*	13
12	*	14	
*	15		

		1	1	4	1	1	3	1	1	2	1	1
		1	1	8	1	1	7	1	1	6	1	5
		1	1	12	1	1	11	1	1	10	1	9
1	1	16	1	1	15	1	1	14	1	13		

1,16			

(2)

			+
		+	*
	+	*	1
+	*	2	+
*	3	+	*
4	+	*	5
+	*	6	+
*	7	+	*
8	+	*	9
+	*	10	*
*	11	*	13
12	*	14	

				1	1	4	1	1	3	1	1	2
			1	1	8	1	1	7	1	1	6	1
		1	1	12	1	1	11	1	1	10	1	9
	1	1	16	1	1	15	1	1	14	1	13	

*	1,15		
5,16			

16			

(3)

			+
		+	*
	+	*	1
+	*	2	+
*	3	+	*
4	+	*	5
+	*	6	+
*	7	+	*
8	+	*	9
+	*	10	*
*	11	*	13

				1	1	4	1	1	3	1	1	
			1	1	8	1	1	7	1	1	6	
		1	1	12	1	1	11	1	1	10	1	
	1	1	16	1	1	15	1	1	14	1	13	

2,12	*	1,14	
*	5,15		
9,16			

16	15		
80			

(4)

			+
		+	*
	+	*	1
+	*	2	+
*	3	+	*
4	+	*	5
+	*	6	+
*	7	+	*
8	+	*	9
+	*	10	*

						1	1	4	1	1	3	1
						1	1	8	1	1	7	1
						1	1	12	1	1	11	1
						1	1	16	1	1	15	1

*	2,11	*	1,13
6,12	*	5,14	
*	9,15		
13,16			

16,24	15	14	
80	75		
144			

(5)

			+
		+	*
	+	*	1
+	*	2	+
*	3	+	*
4	+	*	5
+	*	6	+
*	7	+	*
8	+	*	9

						1	1	4	1	1	3	
						1	1	8	1	1	7	1
						1	1	12	1	1	11	1
						1	1	16	1	1	15	1

+	*	2,10	*
*	6,11	*	5,13
10,12	*	9,14	
*	13,15		

40	15,22	14	13
80,72	75	70	
144	135		
208			

(6)

			+
		+	*
	+	*	1
+	*	2	+
*	3	+	*
4	+	*	5
+	*	6	+
*	7	+	*

									1	1	4	1	1
								1	1	8	1	1	7
							1	1	12	1	1	11	1
					1	1	16	1	1	15	1	1	1

3,8	+	*	2,9
+	*	6,10	*
*	10,11	*	9,13
14,12	*	13,14	

40	37	14,20	13
152	75,66	70	65
144,120	135	126	
208	195		

(7)

			+
		+	*
	+	*	1
+	*	2	+
*	3	+	*
4	+	*	5
+	*	6	+

										1	1	4	1
								1	1	8	1	1	7
							1	1	12	1	1	11	1
					1	1	16	1	1	15	1	1	1

*	3,7	+	*
7,8	+	*	6,9
+	*	10,10	*
*	14,11	*	13,13

40,24	37	34	13,18
152	141	70,60	65
264	135,110	126	117
208,168	195	182	

(8)

Shape Reconstruction Algorithm Implementation

				Tx
			Tx	Σ
	Tx	Σ	\downarrow	
Tx	Σ	\downarrow		\circ
\circ	\downarrow	Σ	\rightarrow	
\circ	Σ	\rightarrow		\circ
Σ	\rightarrow		\circ	\circ
\downarrow		\circ	\circ	\circ
\circ	\leftrightarrow			\circ
\leftrightarrow		\leftrightarrow		
\circ	\circ	\circ		\updownarrow
\circ	\circ	\updownarrow		\bar{D}
\circ	\updownarrow	\bar{D}	Rx	
\updownarrow	\bar{D}	Rx	4	
\bar{D}	Rx	3	8	
Rx	2	7	12	
1	6	11	16	
5	10	15		
9	14			
13				

			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0			
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1				

(1)

				Tx
			Tx	Σ
	Tx	Σ	\downarrow	
Tx	Σ	\downarrow		\circ
\circ	\downarrow	Σ	\rightarrow	
\circ	Σ	\rightarrow		\circ
Σ	\rightarrow		\circ	\circ
\downarrow		\circ	\circ	\circ
\circ	\rightarrow			\circ
\rightarrow		\rightarrow		
\circ	\circ	\circ		\updownarrow
\circ	\circ	\updownarrow		\bar{D}
\circ	\updownarrow	\bar{D}	Rx	
\updownarrow	\bar{D}	Rx	4	
\bar{D}	Rx	3	8	
Rx	2	7	12	
1	6	11	16	
5	10	15		
9	14			

			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	
		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0		
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0				
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1					

13			

(2)

			Tx
		Tx	Σ
	Tx	Σ	\downarrow
Tx	Σ	\downarrow	\circ
\circ	Σ	\rightarrow	\circ
\circ	Σ	\rightarrow	\circ
Σ	\rightarrow	\circ	\circ
\downarrow	\circ	\circ	\circ
\circ	\rightarrow	\circ	
\rightarrow		\rightarrow	
\circ	\circ	\circ	\updownarrow
\circ	\circ	\updownarrow	\bar{D}
\circ	\updownarrow	\bar{D}	Rx
\updownarrow	\bar{D}	Rx	4
\bar{D}	Rx	3	8
Rx	2	7	12
1	6	11	16
5	10	15	

				1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
				1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
				1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
				1	1	1	1	1	1	1	1	1	1	1	1	1	1	

	9	14	
13			

(3)

			Tx
		Tx	Σ
	Tx	Σ	\downarrow
Tx	Σ	\downarrow	\circ
\circ	Σ	\rightarrow	\circ
\circ	Σ	\rightarrow	\circ
Σ	\rightarrow	\circ	\circ
\downarrow	\circ	\circ	\circ
\circ	\rightarrow	\circ	
\rightarrow		\rightarrow	
\circ	\circ	\circ	\updownarrow
\circ	\circ	\updownarrow	\bar{D}
\circ	\updownarrow	\bar{D}	Rx
\updownarrow	\bar{D}	Rx	4
\bar{D}	Rx	3	8
Rx	2	7	12
1	6	11	16

				1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
				1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
				1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
				1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	5	10	15
9	14		
13			

(4)

			Tx
		Tx	Σ
	Tx	Σ	\downarrow
Tx	Σ	\downarrow	\circ
\circ	\downarrow	Σ	\rightarrow
\circ	Σ	\rightarrow	\circ
Σ	\rightarrow	\circ	\circ
\downarrow	\circ	\circ	\circ
\circ	\rightarrow	\circ	
\rightarrow	\rightarrow		
\circ	\circ	\circ	\updownarrow
\circ	\circ	\updownarrow	\bar{D}
\circ	\updownarrow	\bar{D}	Rx
\updownarrow	\bar{D}	Rx	4
\bar{D}	Rx	3	8
Rx	2	7	12

						1	1	1	1	1	1	1	1	1	1	1	1	1	1
						1	1	1	1	1	1	1	1	1	1	1	1	1	1
						1	1	1	1	1	1	1	1	1	1	1	1	1	1
						1	1	1	1	1	1	1	1	1	1	1	1	1	1

1	6	11	16
5	10	15	
9	14		
13			

1			
5			
9			
13			

(5)

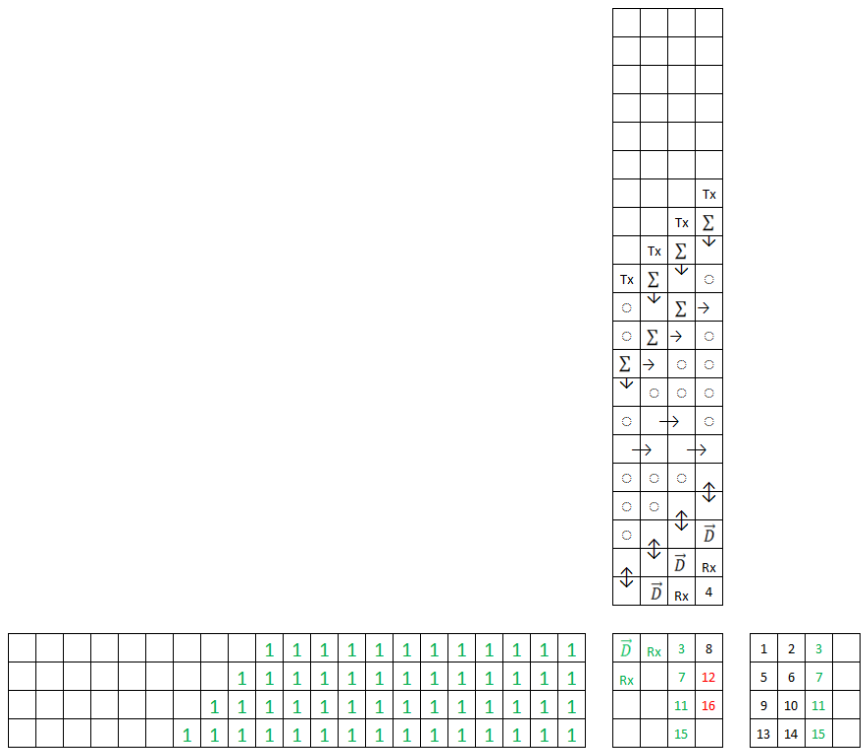
			Tx
		Tx	Σ
	Tx	Σ	\downarrow
Tx	Σ	\downarrow	\circ
\circ	\downarrow	Σ	\rightarrow
\circ	Σ	\rightarrow	\circ
Σ	\rightarrow	\circ	\circ
\downarrow	\circ	\circ	\circ
\circ	\rightarrow	\circ	
\rightarrow	\rightarrow		
\circ	\circ	\circ	\updownarrow
\circ	\circ	\updownarrow	\bar{D}
\circ	\updownarrow	\bar{D}	Rx
\updownarrow	\bar{D}	Rx	4
\bar{D}	Rx	3	8

						1	1	1	1	1	1	1	1	1	1	1	1	1	1
						1	1	1	1	1	1	1	1	1	1	1	1	1	1
						1	1	1	1	1	1	1	1	1	1	1	1	1	1
						1	1	1	1	1	1	1	1	1	1	1	1	1	1

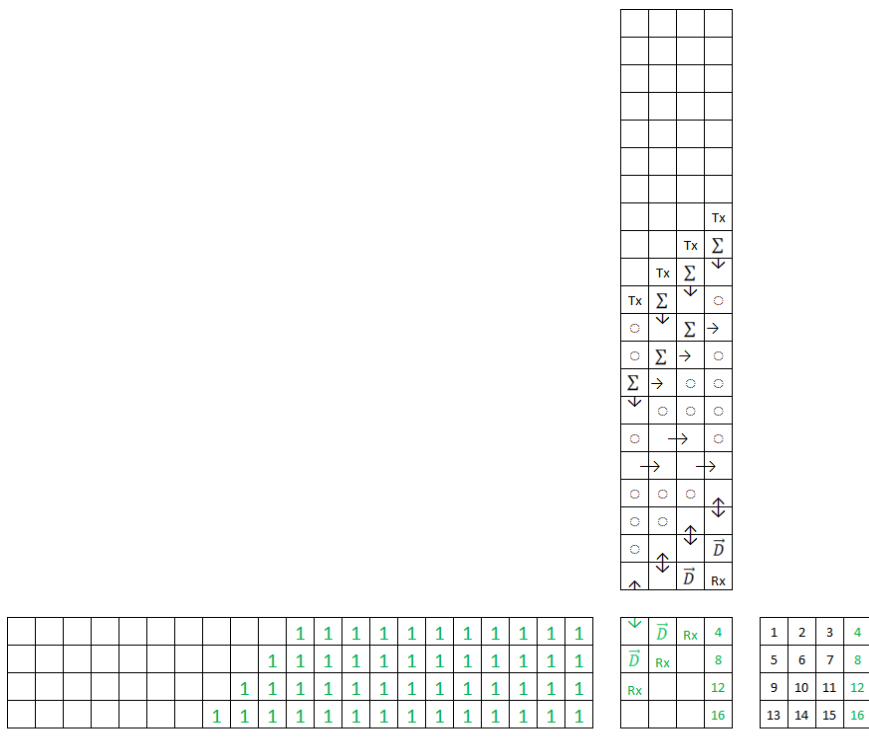
Rx	2	7	12
	6	11	16
	10	15	
	14		

1	2		
5	6		
9	10		
13	14		

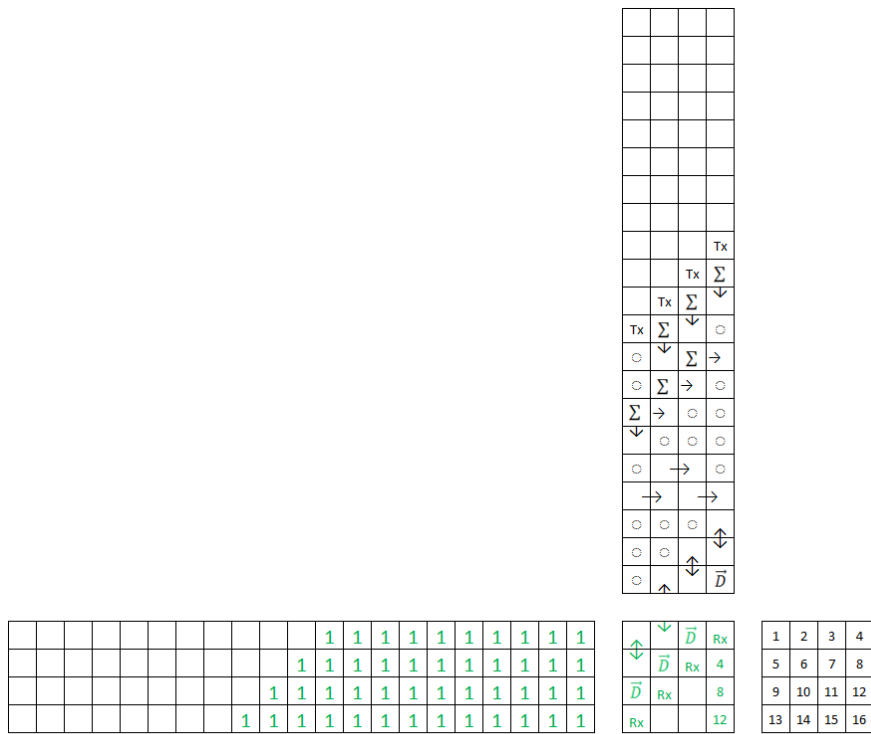
(6)



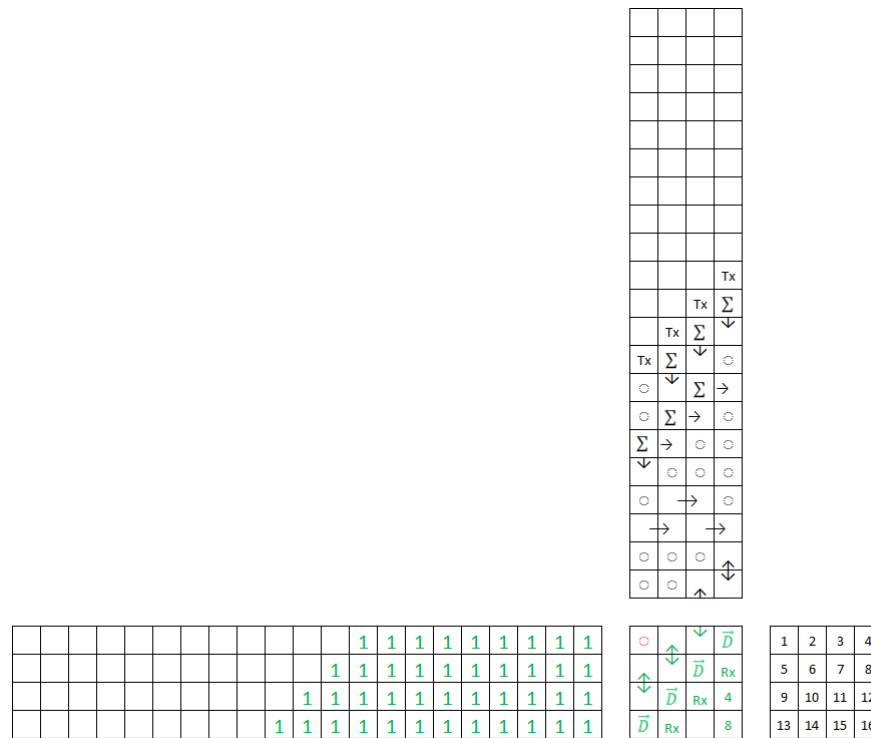
(7)



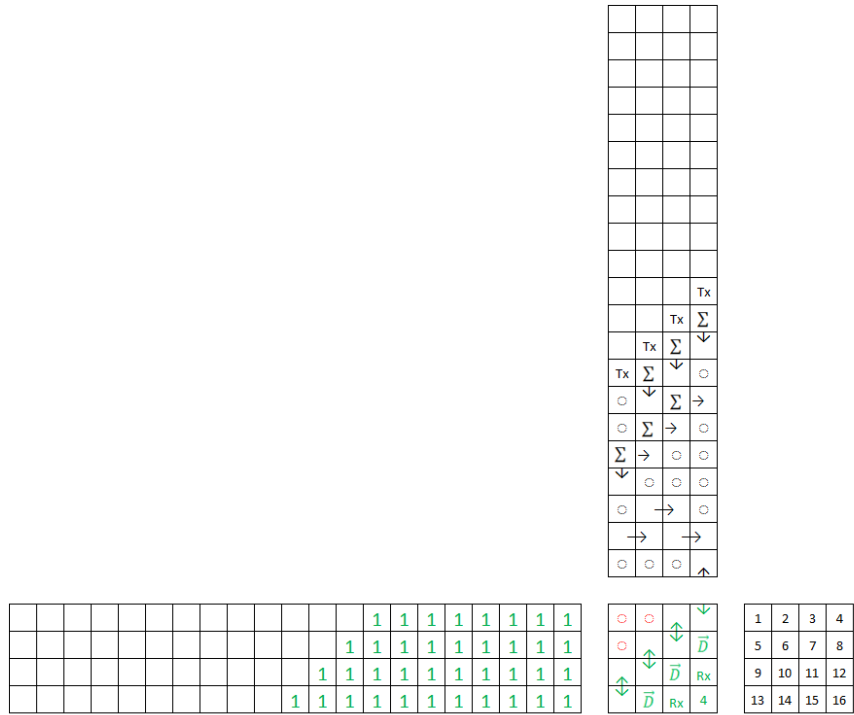
(8)



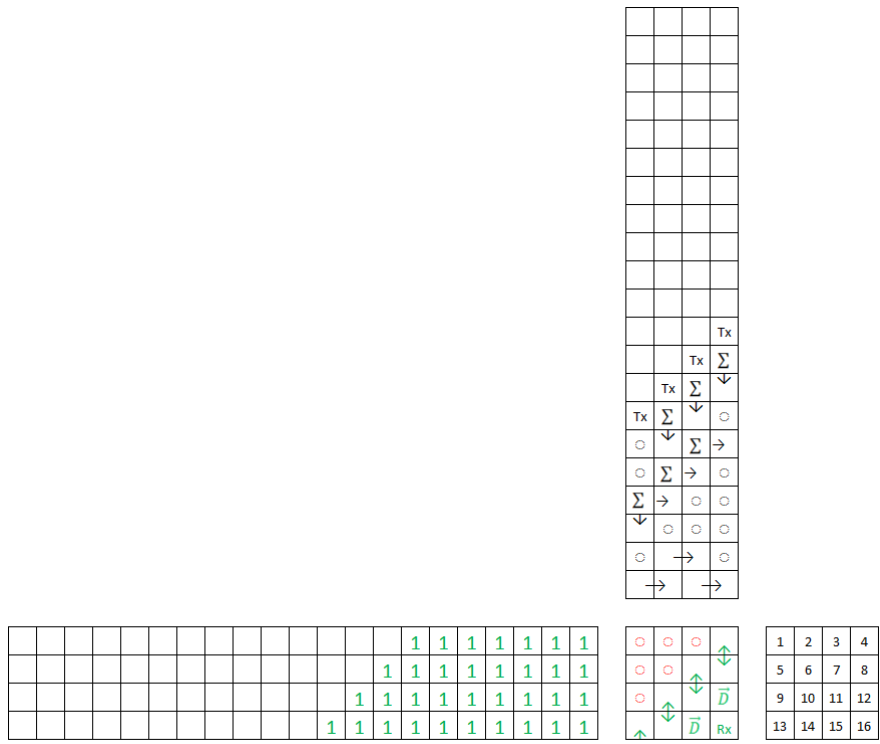
(9)



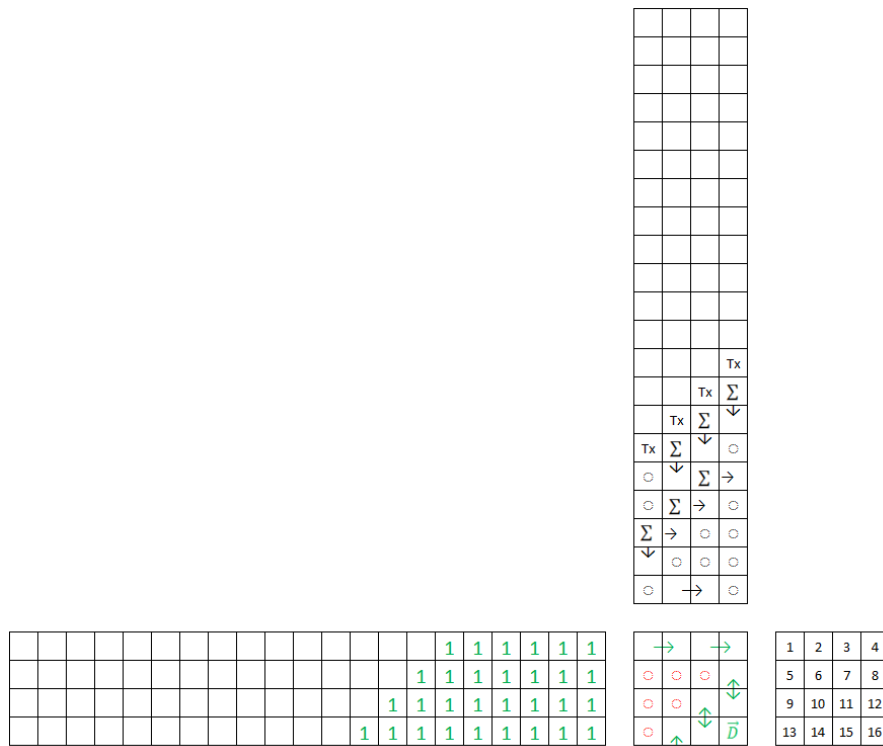
(10)



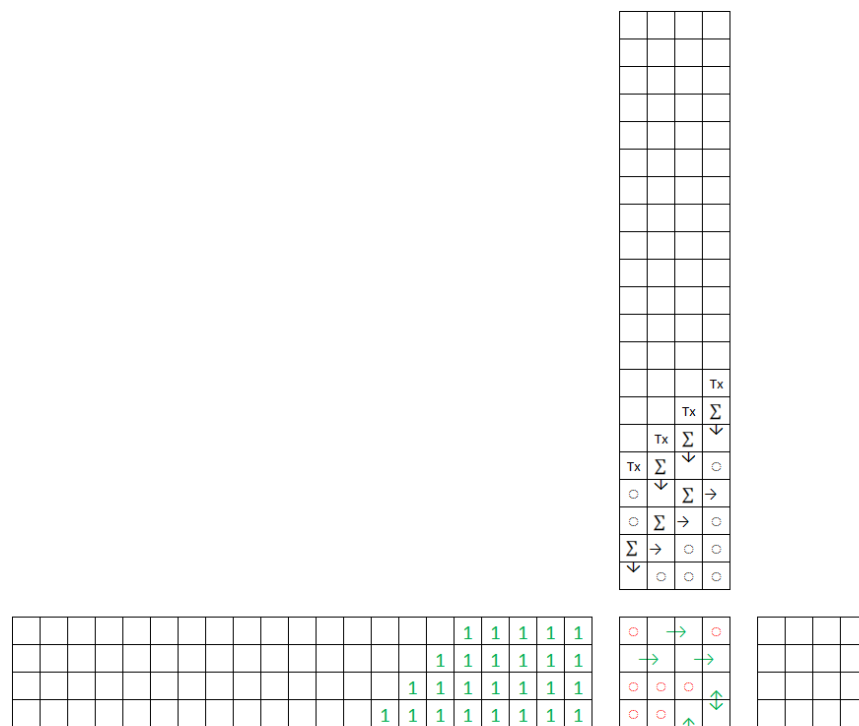
(11)



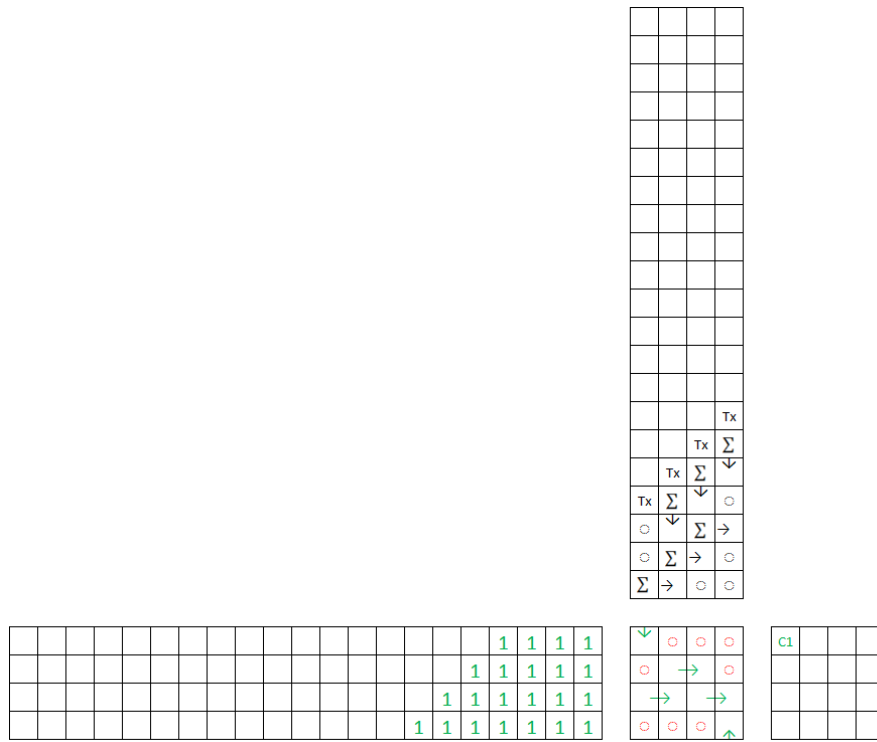
(12)



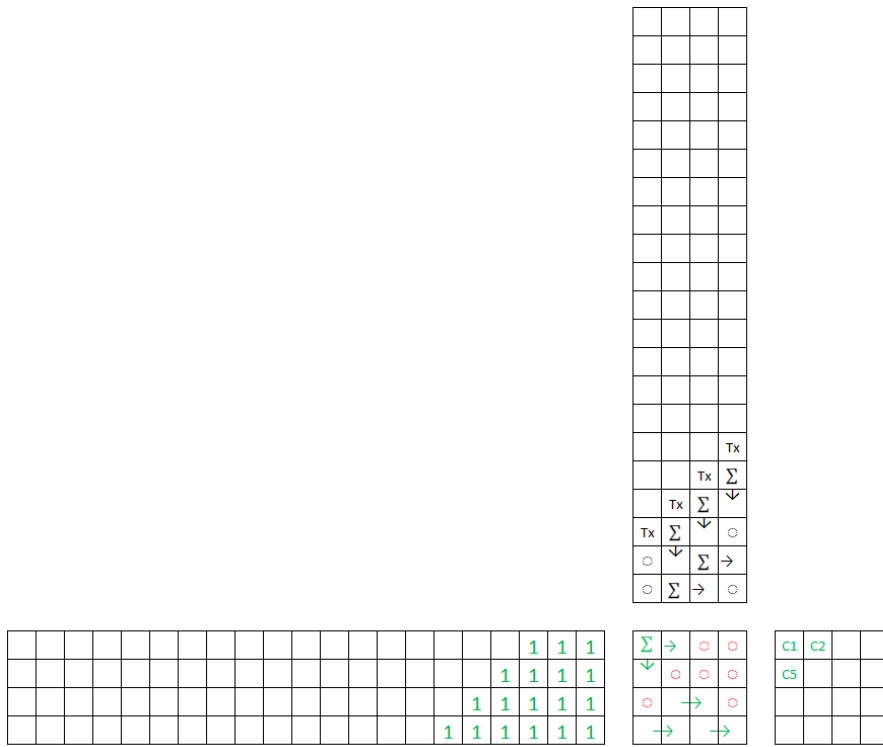
(13)



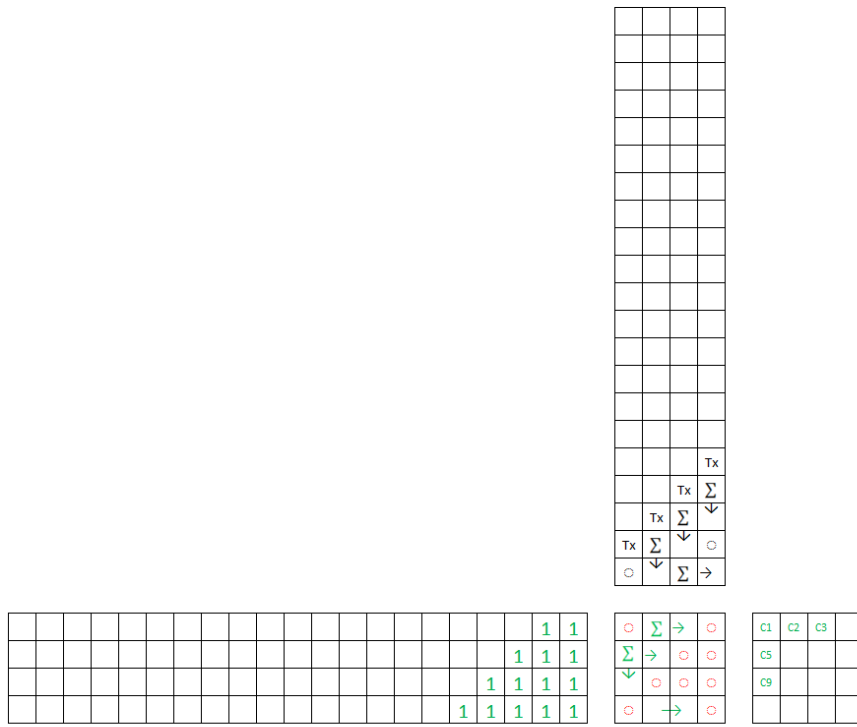
(14)



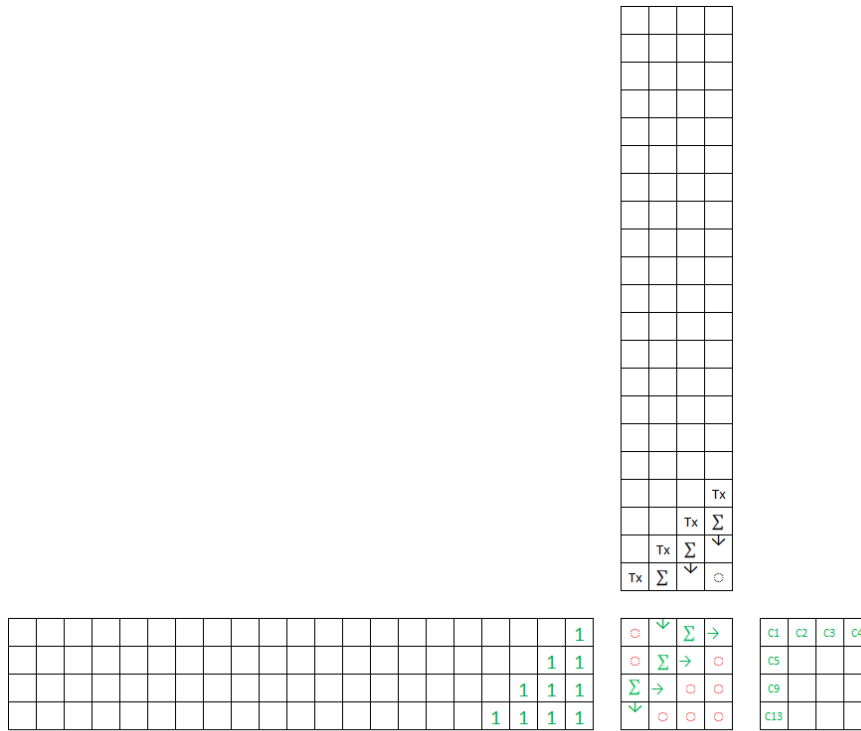
(15)



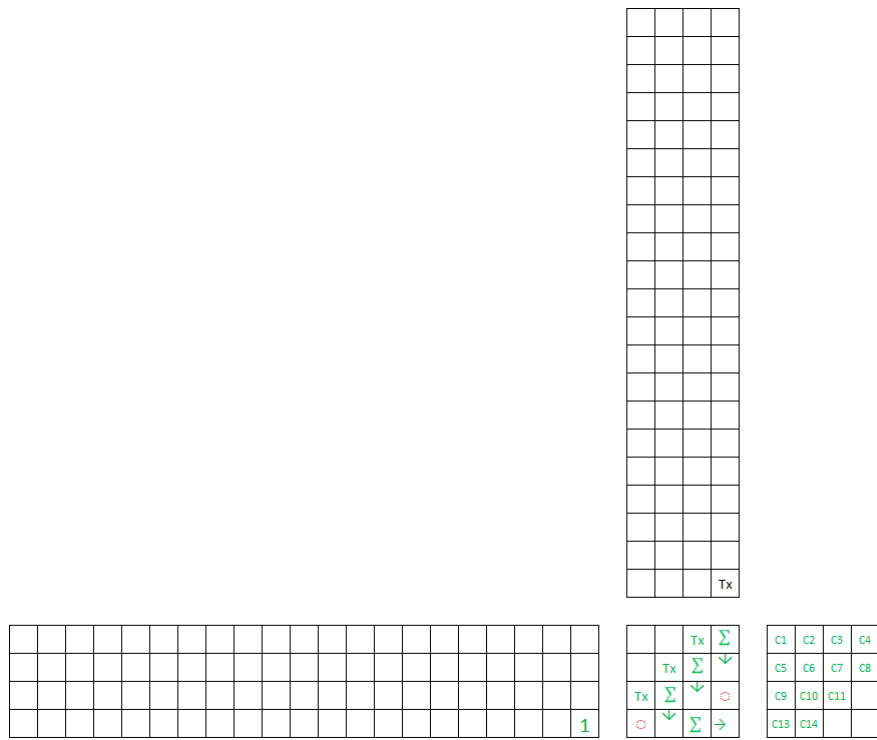
(16)



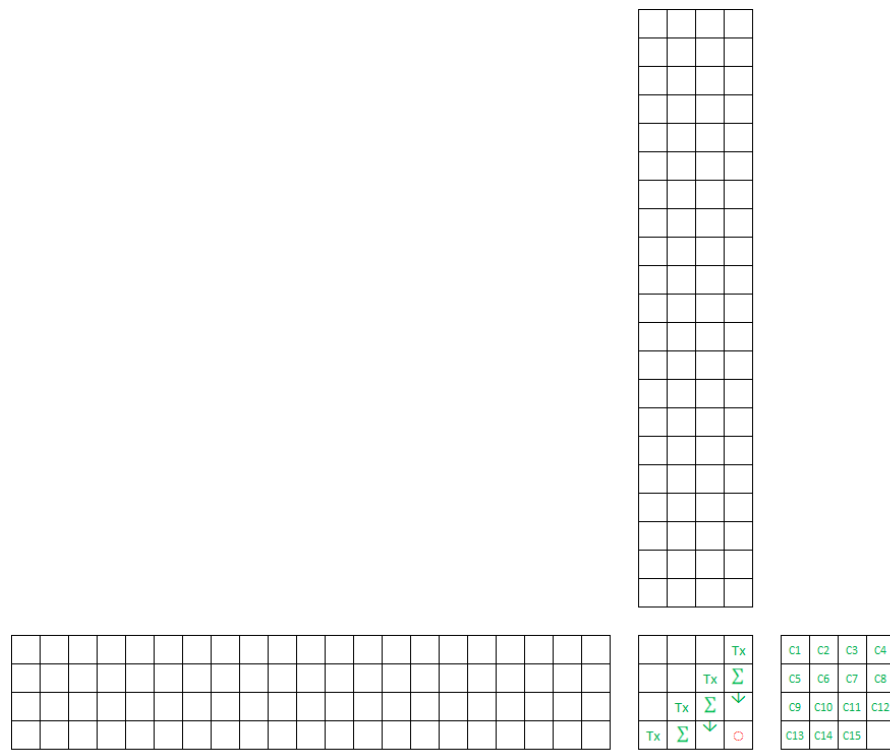
(17)



(18)



(21)



(22)

