Microcontroller based Platform Management System and FPGA Accelerator for SDN Switches

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

Master of Technology

in

Computer Technology

at

Department of Computer Science & Engineering

Jadavpur University, Kolkata

by

Bappaditya Sinha

Registration Number- 145303 of 2018-19

Class Roll Number- 001810504024

Examination Roll Number- M6TCT22002

Under the Guidance of

Prof. Chandan Mazumdar

Department of Computer Science and Engineering

Faculty of Engineering and Technology

Jadavpur University, Kolkata- 700032, India

August, 2022

JADAVPUR UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY

Certificate of Recommendation

This is to certify that the thesis entitled **"Microcontroller based Platform Management System and FPGA Accelerator for SDN Switches"** has been satisfactorily carried out by **Bappaditya Sinha (University Registration Number: - 145303 of 2018-19, Class Roll Number: - 001810504024, Examination Roll Number: - M6TCT22002)**. It is a bona-fide record of work carried out under my guidance and supervision and be accepted in partial fulfillment of the requirement for the degree of **Master of Technology in Computer Technology** from the **Department of Computer Science & Engineering, Faculty of Engineering and Technology, Jadavpur University**, Kolkata - 700032.

Prof. Chandan Mazumdar
(Thesis Supervisor)

Department of Computer Science and Engineering

Jadavpur University, Kolkata, West Bengal, India - 700032

Prof. (Dr.) Nandini Mukhopadhyay

HOD, Department of Computer Science and Engineering

Jadavpur University, Kolkata, West Bengal, India - 700032

Prof. Chandan Mazumdar

Dean, Faculty of Engineering and Technology

Jadavpur University, Kolkata, West Bengal, India – 700032

JADAVPUR UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY

Certificate of Approval

This is to certify that the thesis entitled **"Microcontroller based Platform Management System and FPGA Accelerator for SDN Switches"** is a bona-fide record of work carried out by **Bappaditya Sinha (University Registration Number:- 145303 of 2018-19, Class Roll Number: - 001810504024, Examination Roll Number:- M6TCT22002)** in partial fulfillment of the requirements for the award of the degree of **Master of Technology in Computer Technology** from the **Department of Computer Science & Engineering, Faculty of Engineering and Technology, Jadavpur University**, Kolkata - 700032, during the period of July 2021 to August 2022. It is understood that by this approval, the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion withdrawn therein, but approve the thesis only for the purpose for which it has been submitted.

_____

Signature of Examiner

Date :-

Name :-

_____

Signature of Supervisor

Date :-

Name :-

JADAVPUR UNIVERSITY

188, Raja S.C. Mallick Rd, Kolkata - 700032

Declaration of Authorship

I, hereby declare that the work described in this thesis titled **"Microcontroller based Platform Management System and FPGA Accelerator for SDN Switches"** is entirely my own and in accordance to the requirements of my Master of Technology in Computer technology studies.

All information in this document have been obtained and presented in accordance with academic rules and ethical conduct.

I also declare that, as required by these rules and conduct, I have fully cited and referenced all materials that are not original to this work

_____

Date :- 20/08/2022

Name :- BAPPADITYA SINHA

Registration Number :- 145303 of 2018-19

Class Roll Number :- 001810504024

Examination Roll Number :- M6TCT22002

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompanies the successful completion of this task would be incomplete without the mention of the people who made it possible. Their constant guidance and encouragement crowned my effort with success.

It is a great pleasure to express my sincerest thanks to my project supervisor Prof. Chandan Mazumdar, Professor, Department of Computer Science and Engineering, Faculty of Engineering and Technology, Jadavpur University, for his encouragement, valuable suggestion, and constant support during the course of thisproject.

I would like to thank all the professors of the Department of Computer Science and Engineering, Jadavpur University, Kolkata for the guidance they provided methroughout the duration of the Master of Computer Science Engineering course.

A special note of thanks goes to Prof. (Dr.) Nandini Mukhopadhyay, Head, Department of Computer Science and Engineering, Jadavpur University.

I am also thankful to Prof. (Dr.) Saswati Mazumdar, Head, Department of Electrical Engineering, Faculty of Engineering and Technology, Jadavpur University, for her encouragement during any kind of situation.

I am also indebted to Shibabrata Mukherjee, Subrata Sarkar, Basudeb Das, Anirban Sengupta, Agniswar Chakroborty, Srijita Basu, and the research scholars of CDC JU lab for their seamless cooperation and help in completion of this project. I am thankful to my fellow classmates, my juniors and my family for constant help and support.

Date:

Place:

Bappaditya Sinha

M.Tech in Computer technology,

Registration Number: - 145303 of 2018-19

Class Roll Number: - 001810504024

Examination Roll Number: - M6TCT22002

ABSTRACT

OpenFlow based SDN, is currently implemented in various networking devices and software, providing high-performance and granular traffic control across multiple vendors network devices. OpenFlow, as the first standard interface designed specifically for SDN, has gained popularity with both academic researchers and industry, as a framework for both networking research and implementation. OpenFlow technology separates the Control Plane from the Data Path and this allows the network managers to develop their own algorithms to control data flows and packets. Several vendors have already added OpenFlow to their features such as HP Labs, Cisco researchers, NEC, etc. Currently, OpenFlow Switch is already implemented on several different platforms e.g., in software (Linux, Open-WRT) and hardware (NetFPGA). More and more researchers are implementing the switch on FPGA-based platform, because it is flexible, fast and reprogrammable. However, there are a limited number of studies about the performance of the OpenFlow switch, which motivates this project. In order to execute the research on OpenFlow performance, the simulation model of OpenFlow system is implemented in this project. The main objective of this project has two aspects. On one hand, it is meant to implement OpenFlow system (switch and controller) using a hardware language on FPGA-based platform. On the other hand, it is also focuses on the measurement of different performance metrics of the OpenFlow switch, especially the service time (switch and controller). More specifically, both the data plane and control plane are to be implemented on FPGA-based platform. It is designed in VHDL language by VIVADO design tools. FPGA-platform is ZYNQ 7000 SOC(Zedboard) type from Xilinx. It is observed from the results that the service time and the sojourn time both have almost linear increase with the increase in payload size.

Commercially available traditional hardware management system is made for a particular switch system and its hardware is marked for a particular defined task. But the Programmable Hardware Management System that has been developed as a part of this work is more flexible, sophisticated and easily reconfigurable. In this paper, the features and system description of the developed system has been described which is more advantageous than traditional commercially available switches. This system management platform can be fitted, reconfigured and reprogrammed universally in any type of system.

CONTENT

List of Figures

List of Tables

INTRODUCTION

1.Introduction

Switches, routers, and many other networking devices are the key networking components used in our daily life. The increasing demands of improvement in the networking capabilities and performance have driven the network engineers to innovate new methodologies and develop different perspectives for enhancing and improving the network architecture. A new development perspective to continue the network technology evolution has been introduced in the form of Software-Defined Networking (SDN) [1]. The Ethernet, which is the dominant networking technology in Local Area Networks (LAN), is in crisis. Developed initially, decades ago, the prime function of the Ethernet was to serve as a flexible and decentralized network. Since then, Ethernet has progressed to serve expanded networks with high capacity, yet leaving the architectural security with little attention. Furthermore, popular emerging networking technologies such as network virtualization use extended Ethernet segments. Due to the lack of routing in Ethernet, such technologies suffer from inefficiency in transmission. Otherwise, network security has been improving rapidly over the past years. As new protocols and proprietary technologies are developed to secure communication between and within the hosts and other network devices, many vulnerabilities are still present with no current resolution. Experts face challenges as their solutions are constrained by the concept of how the legacy network operates. The legacy network is generally constructed with distributed intelligence. This appears as the nodes of the network handle their behavior according to the adjacent node. This directs security experts to resolve the security issues in restricted manner. By choice, security issues would be easier to tackle if the control over the whole network would be given to the security entity. The main unique feature which makes SDN popular is its ability to make a network programmable. The new emerging technology has the potential to revolutionize the networking industry in many ways. Nevertheless, security aspects have to be managed and evaluated in order to provide reliable networks. Therefore, research of SDN capabilities to overcome legacy Ethernet security issues is needed. This thesis aims to answer the question of how to improve Ethernet LAN security utilizing SDN properties, and the challenges that can be expected in the process. The network security is discussed in a broad manner, but the main focus is set on the Ethernet LAN. Existing SDN security solutions are reviewed and security issues prevailing in the Ethernet networks are discussed from the SDN point of view. In addition, a proposal for security enforcing SDN solution is introduced and evaluated. Yet, the further objective of this thesis is to open up a door for discussion and innovation in the research community to confront security challenges in SDN. The network is

in complete submission to the controller which becomes a threat when it is compromised. As long as the access to the controller is restricted, SDN can bring many advantages for network security improvement. This thesis contributes to the further SDN security development and research, by providing possible SDN solutions to overcome legacy Ethernet security threats and by presenting an example of a secure SDN network.

Network infrastructure has become critical in the Internet and enterprise network. However, with the explosion of mobile devices and the rise of cloud services with a limited available bandwidth, network architecture has become complex. This results into the fact that current network capacity can not match users' requirements. Networking technologies exert limitations such as complexity, inconsistent policies, inability to scale and vendor dependence, which can not satisfy high requirements of network architecture in enterprises, homes and educational institutes [2]. At the same time, changing traffic patterns, "IT consumerization", the rise of cloud services and bandwidth limitation trigger the need of new network architecture [3]. Moreover, some network vendors are unhappy that researchers run experiments or test new protocols in their Internet environment, because it may lower or interrupt production traffic. Thus, the network innovation is needed to satisfy more users' requirements and also to optimize the current network.

Recently, Software-Defined Networking (SDN) created by Open Networking Foundation (ONF) attracted many academic researchers and vendors. ONF, a non-profit organization, is responsible to control and publish the different OpenFlow specifications and give the trademark license "OpenFlow Switching" to companies that adopt this standard. OpenFlow technology separates the control plane from the data path and this allows network managers to develop their own algorithms to control data flows and packets, resulting in more efficient network management, more flexibility in response to demands and faster innovation [2]. Furthermore, it implements the control logic on an external controller (typically an external PC) and this controller is responsible for deciding the actions that the switch must perform. The communication between the controller and the data path is made on the network itself, using the protocol that provides OpenFlow (OpenFlow Protocol) via Secure Socket Layer (SSL) [2]. Now the researchers are not required to wait for new features to be released from equipment vendors and they can develop new intelligent services rapidly and independently in multiple-vendor environment [4]. Thus, OpenFlow has gained popularity with both academic researchers and industry due to its various advantages (decouple data and controller path, and routing intelligence).

Actually, OpenFlow switch has already been implemented on several different platforms (Linux, OpenWRT and NetFPGA). There are many studies about the OpenFlow switch implementation on FPGA-based platform[5]. FPGA-based switches are open-source hardware, which are faster than software-based switch. Besides, FPGA hardware is reprogrammable so that researchers can develop their own OpenFlow switches. And OpenFlow switch prototypes on FPGA-based platform can forward packets at 1-10Gbps. Thus, more and more people care about the OpenFlow switch implementation on FPGA-based platform and try to improve the OpenFlow switch. However, only a few works on the performance analysis of the OpenFlow switch are done. The implementation of OpenFlow Switch on NetFPGA has high latency to insert a new flow into OpenFlow Switch, which is still a bottleneck [5]. However, the implementation of Distributed Multimedia Plays (DMP) network nodes indicates lower latency and scalability features on FPGA based platform [6].

Therefore, this project is motivated to implement simulation model of OpenFlow system (data plane and control plane) on FPGA-based platform and also measure the performance metrics of the OpenFlow switch. However, our switch is designed only for research purpose and not for the market. It is observed from the results that the service time and the sojourn time both have almost linear increase with the increase in payload size. Moreover, the results indicate that the switch takes only 2 clock cycles to respond to the writing request of the controller, which decreases the communication time between the switch and the controller.

The data packets are forwarded from one device to another device by the 'Network Switch'. The other functionality of the network switch is interconnecting each other between the Network devices of the network system. Different companies are working on it, and they are making a hardware management system. These hardware management systems are different for different types of systems. But this is a distributed era and so different changes are required according to different applications. Therefore, hardware management systems also require changes according to this application. Different companies provide some solution, like SDN's controller, switches components etc. Repeated changes over this management system is problematic and it is not possible to change the system from scratch. Moreover, our growing technology wants a sophisticated and flexible system where the requirements are permitted easily according to users, that is a reprogrammable, reconfigurable, reassignable universal system. The present developed hardware management system is a system which continuously checks the status of the switch and continuously provides feedback on its present state. E.g. A power supply unit is used to supply the power to a switch unit, if any fault has occurred in the power supply unit, this developed system is acknowledged to the user and necessary precaution

is taken by the user and it also switches the power line to the secondary power supply immediately. As another example, some thermal sensors are incorporated into the system to continuously monitor the temperature condition of the system. If the desired temperature range is exceeded, the system may start to malfunction. To avoid such a situation, a fan is incorporated to cool down the system for a particular increased temperature of the system. Another way, it can be said that if the temperature is exceeded abnormally, then the system shuts down automatically as a result the whole system is saved from severe damage. Therefore, this developed management system can be incorporated into any switching system universally.

1.1 Objective

The main objective of this thesis has two aspects. Number one is implementation of 'Platform Management system for SDN Switches' and another one is to propose 'FPGA to accelerate SDN'.

- In traditional platform management system, it cannot be easily customizable or reconfigurable according to user requirements. But the developed platform management system is easily flexible, reconfigurable and programmable as per user requirements.
- This developed platform management system provides a fast power supply switching mechanism.
- As per user requirement, more number of Cooling Fan, Temperature Sensor can be customized.
- Another important objective of this developed platform management system is to monitor the Switch Parameter continuously (Temperature, Power Supply condition etc) and as per the present condition, it has been provided feedback to the user.
- In this thesis, another important objective is to propose an FPGA-based mechanism accelerating and securing SDN software switches, namely, FAS.
- In this part the main objective is, to propose how to implement the OpenFlow Switch in FPGA Board.
- All the design parameters regarding this design has been proposed in this section also.

1.2 Structure of the Thesis

The rest of the document is organized as follows: Chapter 2 describes the concept of SDN and OpenFlow, as well as their working principle. Chapter 3 discusses about a model SDN switch, basically a switching software and switch fabric. Chapter 4 describes the concept of platform management System and its features. Chapter 5 describes the Hardware implementation of Platform Management System for SDN switches. Chapter 6 describes Software implementation of Platform Management System for SDN switches. Chapter 7 describes the FPGA to Accelerate SDN and the basically how to implement OpenFlow in an FPGA based SDN Switch. Chapter 8 describes the overall conclusions drawn on the subject appear and it also includes future scope of the work. The List of Publications as an outcome of this Dissertation, references cited in the work and appendix are placed at the end.

1.3. Literature survey

Many OpenFlow switches have already been implemented in different platforms such as Linux (software), OpenWRT (software) and NetFPGA (hardware). Hardware/ commercial switches (e.g, HP ProCurve, NEC IP8800) and software/Test switches (NetFPGA switch, OpenWRT) have been released and used in real network environments. Switches are open sources and can be found on the website so that everyone can download for using or modifying. This section briefly introduces some related work about current OpenFlow switch implementations. [7] describes the Type-0 OpenFlow switch implementation on the NetFPGA platform. NetFPGA used in [5] is a Gigabit rate networking hardware, consisting of a PCI card with an FPGA, memory, and four 1-Gig Ethernet ports. This implementation could hold more than 32,000 exact-match flow entries running across four ports and the exact-match flow table can be expanded to more than 65000 entries. The implementation enhances flexibility and reusability of hardware. The performance results claim that it takes 11 μs to insert a new flow into switch due to the PCI bus bottleneck [6]. At the same time, the bottleneck of their OpenFlow switch is that it has 240 bits flow header entry currently along with the actions, which can be aggravated when packet re-injected from controller into switch using the same communication channel (PCI bus). Additionally, OpenFlow switch can provide many functionalities at lower logic gate cost in comparison with IPv4 Router, the NIC and the learning Ethernet switch [5].

While another OpenFlow switch implemented on NetFPGA can hold more than 100000 flow entries and it is also capable of running at line-rate across the four NetFPGA ports [7]. In the software plane, OpenFlow reference software implementation is extended by using a new _FA

data structure to create the rules instead of hash function, which is more advanced than the switch implemented on the NetFPGA. This switch provides a flexible packet forwarding architecture based on regular expression [7]. Besides, it also enables the standard-compliant OpenFlow switching, which can be easily reconfigured through its control plane to support other kinds of applications [7]. Furthermore, the performance analysis is also done in this article. The results of performance analysis indicate that the switch is able to process all traffic data even in the case of a Gigabit link saturated with minimum-sized packets [7].

Because low-level Verilog RTL severely limits the portability of OpenFlow switch, the switch in [8] is implemented with Bluespec System Verilog (BSV) which is a high-level HDL, and addresses the challenges of its flexibility and portability. The design comprises of approximately 2400 lines of BSV code. This switch meets the OpenFlow 1.0 specification and achieves a line rate of 10 Gbps, which is highly modular and parameterized, and makes use of latency-insensitivity, split-transaction interfaces and isolated platform-specific features [8]. In this article, the OpenFlow Switch is also ported into NetFPGA-10G, the ML605 (Xilinx) and DE4 (Altera). The exact match flow tables of this switch is implemented on both Block RAM and DRAM. It is found that it has lower pipeline latency of 19 cycles for a packet to go from ingress to egress when implementing exact flow tables on Block RAM[8]. Furthermore, the switch is implemented in two configurations, one is in an FPGA communication with controller via PCIe or the serial link, another is in an FPGA-based MIPS64 softcore. It is found that the switch responds to controller requests in less cycles used with the PCIe than serial link [8].
The related works about OpenFlow switch implementation have already mentioned above and most of OpenFlow switches are implemented on the NetFPGA. Except these related works, there is limited number of studies on performance analysis of the OpenFlow switch.

In order to improve the OpenFlow switching performance, the mechanisms are introduced in [9] and [10]. An architectural design is proposed to improve the lookup performance of OpenFlow switching in Linux by using a standard commodity network interface card. The results in [9] show a packet switching throughput increase of up to 25 percent compared to the throughput of regular software-based OpenFlow switching [9]. Instead, the results in [10] show a 20 percent reduction using network processor-based acceleration cards to perform OpenFlow switching [9]. However, only one paper studies the performance measures of OpenFlow [10]. It is concluded that the OpenFlow implementation in Linux systems can offer very good performance and it shows good fairness capability in dealing with multiple flows [11].

Furthermore, it is also found from the results that large forwarding tables are generated due to L2 switching [11]. [12] also studies the performance measurements of not only OpenFlow switch but also OpenFlow controller. In [12], a performance model of an OpenFlow system is provided, which is based on the results from queuing theory and is verified by simulations and measurement experiments with a real OpenFlow switch and controller. The results in this article show that the sojourn time mainly depends on the processing speed of the OpenFlow controller [12].

Moreover, it indicates that lower is the coefficient of variation when the probability of new flows arriving at the switch is higher, but longer is the sojourn time [12]. Thus, it can be seen from the description above that OpenFlow-SDN has already appealed to some attentions in both researchers and vendors. At the same time, the increasing number of researchers gradually has implemented their own OpenFlow switch on FPGA-based platform. The OpenFlow network implementation described in this thesis is a little different from the related work.

SDN & OPENFLOW

2.1. Software-Defined Networking

Software-Defined Networking (SDN) concept was first time introduced in 2010 [1] as the new networking paradigm which aims to ease the control and the management of a computer network environment. SDN can be explained as an architectural principle where the networks control and the management are centralized and decoupled from data plane, thus making the network programmable. To quote the definition paraphrased from the Hot SDN '12 Solicitation: Software-Defined Networking (SDN) is a refactoring of the relationship between network devices and the software that controls them. Traditionally, the data and the control planes in the Ethernet networking devices (and most of the communication principles) have been tied together. This means, the prevailing operating system and its features with the provided hardware are implemented in a single device. Therefore, network devices, such as switches, routers, firewalls, etc., are built with the intelligence of handling traffic relative to the adjacent devices. This makes the intelligence distributed and scattered in the network. In addition, most of the network devices are Command Line Interface (CLI) based and configuration is done separately per device, making configuration slow and prone to errors. This prevents the networking industry of responding quickly to feature requests or innovate new management abilities [13].

The data plane has a sensible layering model which is known by the name Open Systems Interconnection model (OSI model) [14]. It is well known model in the networking industries and academies. It is standardized by the International Organization for Standardization (ISO). The OSI model enables network applications and services to isolate the data operations to a single layer and provide interfaces between layers. This has enforced developers to develop and improve the operations without concerning the other layers. This type of layering models are used in many other fields (e.g. operating system) and it has provided simplicity to understand the overall view and the interactions. As a result, we can witness increase in the development and research in these fields. As it seems, similar layering model is essentially needed for networks control and management plane, which was not available. This creates the need to invent a new networking architecture, SDN. SDN architecture decouples control from data plane and provides it a new layering model. Open Networking Foundation (ONF) [15] is a non-profit industry consortium which has taken the lead in standardizing critical elements for SDN architecture. One of these standards is OpenFlow, which will be covered in the next section. ONF has defined the following layering scheme for SDN architecture.

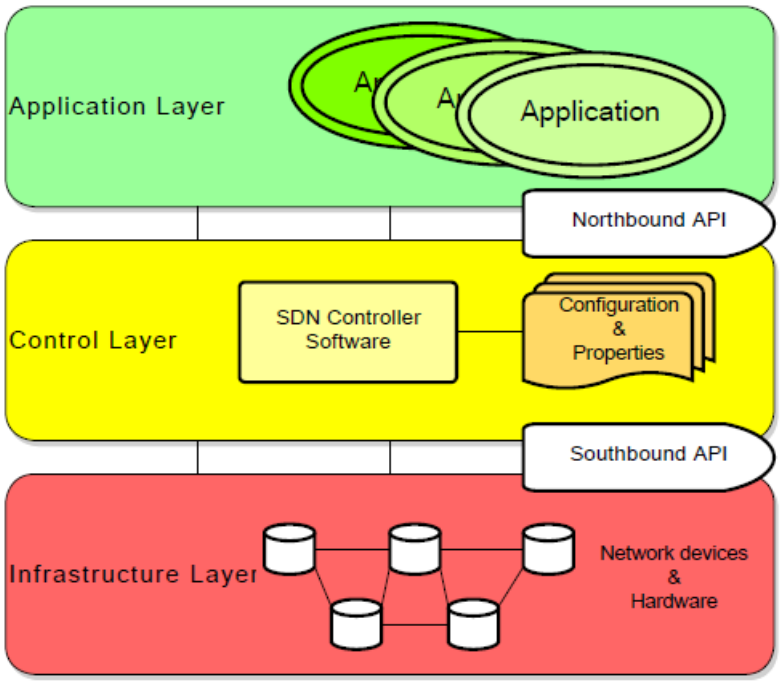

Figure 1: The SDN stack

As seen from the Figure 1, SDN architecture is divided into three layers: application layer, control layer, and infrastructure layer. This architecture and arrangement of control and management, provides the possibility to centralize the state of the network and the intelligence into one part of the network. This, enhances the property of network programmability, the network industry can start to innovate and enable differentiation in the developing process. Furthermore, programmability accelerates creativity and introduction of new network features and services. With centralization, SDN simplifies provisioning while optimizing performance and granularity of the policy management. Therefore, SDN can make networks become more scalable, flexible and proactive. SDN architecture stack abstracts and decouples hardware from software, control plane from forwarding plane, and physical from logical configuration. Infrastructure layer is the layer where all the hardware exists and are connected physically. On these hardware devices runs a software which provides a control data plane interface (Southbound API) which is used to communicate with the upper level: Control layer.

Control layer is the most important layer in the architecture. There is a controller which talks to all the network devices in the infrastructure and keeps track of the topology. While exchanging information of the network state with upper layer applications (through

(Northbound API), the controller translates their commands to the network devices to have respective and desired network behaviour.

Application layer is the layer where all the features, services and policies are defined. Applications request the information of network devices and the topology in order to to act upon it. These applications can create features end-to-end and make big picture decisions according to the changes in the network. When the network topology, feature, or policy requirements changes, applications have the control to change dynamically the network behaviour from one single point.

Between these layers, there are Application Programming Interfaces (APIs) which provide the essential communication tools between the layers. The Northbound API is provided by the controller and the applications have to manage their communication to the controller through it. At the time of writing this thesis, there was no standards for this interface, yet many SDN controllers were settling down with REST API [13]. Other popular APIs are C++, JAVA and Python. The Southbound API is the communication between the controller and the network devices. In the following section, a protocol providing this feature shall be further discussed.

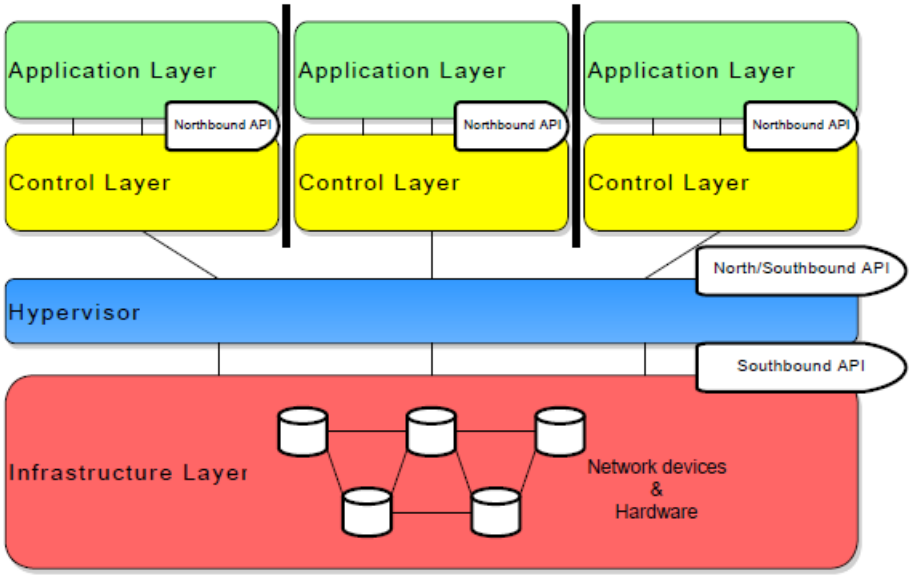

Figure 2: The SDN stack with Hypervisor

The SDN layer model can be adjusted to fit and satisfy a data center scheme. Where virtualization may need logical network segment slicing. In this case an extra layer is added in the model (see Figure 2) and the controller of controllers, also called the Hypervisor, can provide logical slicing (while physically network is untouched). In other words, the Hypervisor

allows every individual controller to control only their own hosts (physical or virtual) on the network without affecting other parts of the network.

SDN brings new challenges in networking technology and in this thesis the focus is set on the network security. A programmable network provides full control of a network. Thus, bringing more capabilities to handle security in the network, whether in a local area network (LAN) or in the core. Similarly, new unexplored threats against a SDN network can be unveiled or discovered. Nevertheless, SDN has gained rapidly its reputation and is the biggest hype word in networking business.

2.2. Architecture for SDN

A logical view of SDN architecture can be seen in Figure 3. The infrastructure layer refers to the data plane where all hardware lies. The control layer refers to the control plane where SDN control intelligence lies, and the application layer includes all other applications handled by the network. The infrastructure and control layer are connected via control data plane interface such as OpenFlow protocol, whereas the application layer is connected to the control layer via application programming interfaces (APIs). With the help of SDN, a vendor independent control from a single logical point can be obtained, and a network administrator can shape traffic from a centralized control console rather than going through individual switches. It also simplifies network devices, since devices do not need to understand and process numerous standard protocols but only handle instructions from the controller. In SDN architecture, APIs are used for implementing common network services which are customized to meet business demands such as routing, access control, bandwidth management, energy management, quality of service etc [15].

The limitations of traditional networking technologies make it harder to determine where security devices such as firewalls should be deployed in the network. SDN can overcome the classic problem by implementing a central firewall in the network, and thereby network administrators can route all traffic through a central firewall. This approach facilitates easier and centralized management for security firewall policies, real-time capture and analysis of traffic for intrusion detection and prevention. However, on the other side a central firewall poses a single point of failure in the network [13].

The nodes at control layer are called as controllers, and they send information such as routing, switching, priority etc to the data plane nodes associated with them. After receiving the information from control node, the networking devices in the data plane update their forwarding table according to the information received from the control plane. The control

nodes can further be architecturally classified into centralized and distributed mode [17], which can be seen in Figure 3 and 4 respectively.

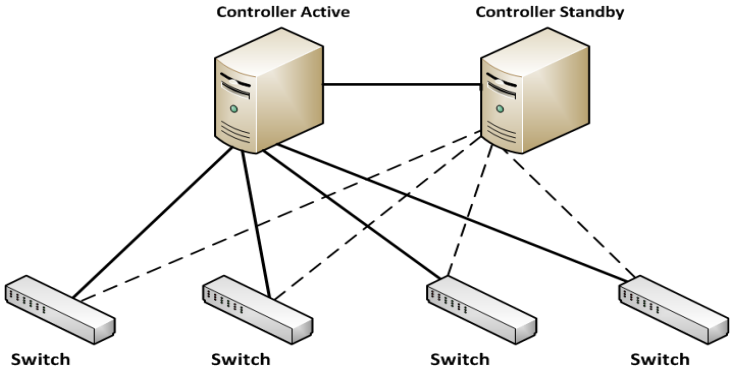

Figure 3: Centralized architecture for SDN

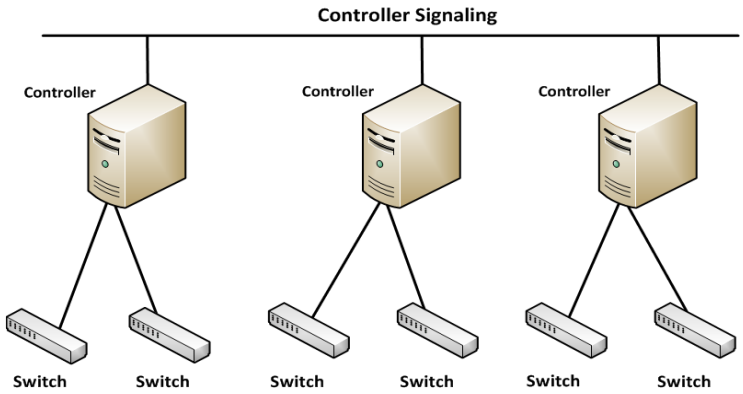

Figure 4: Distributed architecture for SDN

In centralized mode, a single central control node sends switching, routing and other information to all networking hardware in the network. Meanwhile, in distributed mode, there are plenty of control nodes associated with certain networking hardware that send information to them [16]. The centralized mode possesses a risk of single point of failure; therefore load-balancing and redundancy mechanisms are often applied in centralized approach deployment.

2.3. Information About SDN Switch

A OpenFlow/SDN switch, when it receives a packet, that it does not have a flow for (Match + exit port) will contact a SDN controller (Server) and ask what must it do with this packet. The controller can then download a flow to the switch, possibly including some packet manipulation. Once the flow is downloaded to the switch it will switch similar packets at wire-speed.

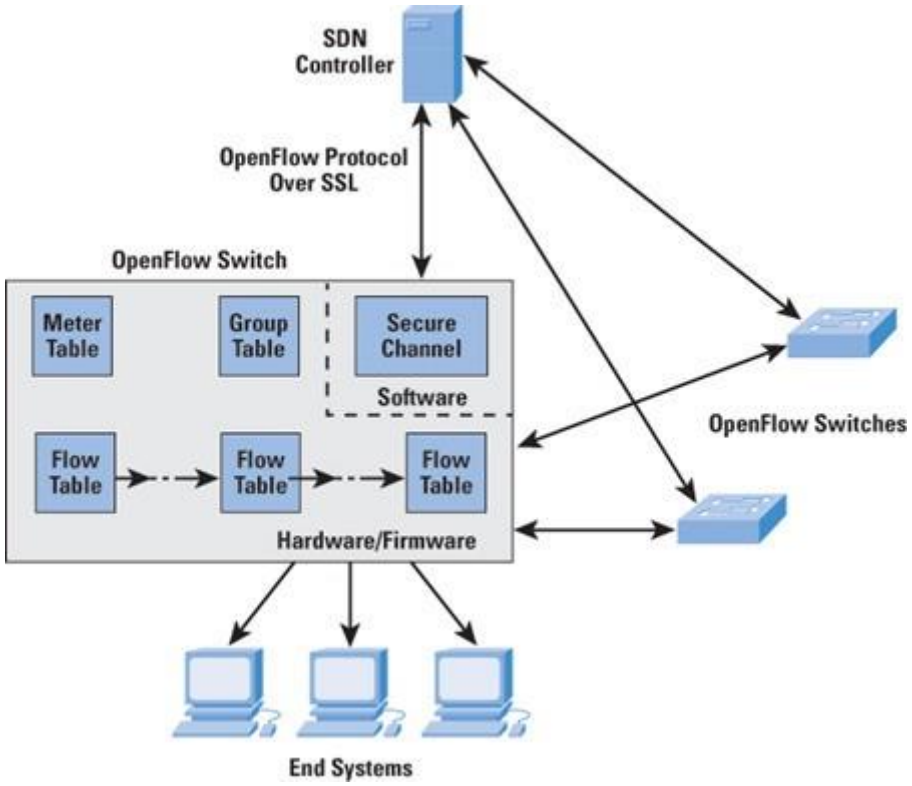

Figure 5: System overview

Having a central server that knows the network layout and can make all the switching decisions and build the paths gives us new capabilities. The advantages of the SDN switches are listed in the below:

1.The SDN controller could route non critical/bulk traffic on longer routes that are not fully utilized.

2.The SDN controller could send the initial couple of packets to a firewall, and once the firewall is happy/accepts the flow, the SDN controller can bypass the firewall thus removing load from the FW and allowing multi-gigabit data centers to be fire-walled.

3.The SDN controller can easily implement load-balancing also at high data rates by just directing different flows to different hosts, only doing the set-up of the initial flow's.

4.Traffic can be isolated without the need for vlan's, the SDN controller can just refuse certain connections.

5.Setup a network TAP/Sniffer easily for any port or even specific traffic by programming the network to send a duplicate stream to a network monitoring device.

6.It allows for the development of new services and ideas all in software on the SDN controller. OpenFlow-Actions.

2.4. OpenFlow

As mentioned in the previous section, the communication between network devices and the controller is handled through the southbound API of the controller. As the most dominant networking technology is the Ethernet, the first standard for SDN was created to manage the controlling of Ethernet switches. OpenFlow [2] is a standardized (by ONF) protocol for SDN supported networks to handle the communication between Ethernet switches and the SDN controller. OpenFlow was derived from SANE [17] and Ethane [16], which were one of the first projects to decouple control and data plane. OpenFlow shortly started to become more popular and as an open standard, it developed quickly to support more and more functionalities.

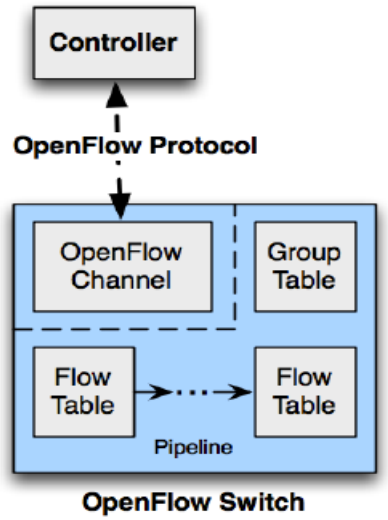

Figure 6: The three main parts of an OpenFlow switch

A switch which supports OpenFlow, initially consist of three essential parts (see Figure 6): flow and group table(s), OpenFlow channel, and OpenFlow protocol. As OpenFlow is relatively young and it continues on developing, new functionalities and improvements change OpenFlow's properties to suite current network demands. Therefore, in this section, only the essential parts of OpenFlow will be explained and other improvements and version differences are explained later on according to referral need. Using OpenFlow protocol, a controller can learn Ethernet switches' hardware details, connectivity status, and the network topology; while commanding its forwarding table behaviour by giving flow rules.

Flow and group tables define switches behaviour with data flow coming from different interfaces (physical, virtual). The table consists of a set of rules, where the flow of the communication data is defined. The Switch reacts upon every flow according to the rules, called flow rules. An OpenFlow switch has one or more flow tables and a group table for frame lookups and forwarding. A flow rule consists of three fields (see Figure 4):

2.4.1. Rule: a header to match with the frames of the flows. There are several supported Ethernet headers in OpenFlow specification, but as OpenFlow is made to be extensible, custom headers can be additionally defined. The switch merely performs a bit mask match. Therefore, OpenFlow switch is open for innovative non-IP traffic.

2.4.2. Action: as a rule is matched with traffic, the action which should be performed for it has to be defined. The actions are also open for extensions, but some basic actions are already provided in the specification. Such as, forwarding to one or more ports, forward to the controller, drop the frame, and modify frame fields. The only requirement for adding customized actions is that the data path must have flexibility while providing high performance and low cost.

2.5. OpenFlow Switch

The theory of OpenFlow switch is introduced briefly here. As shown in Figure 2.1, OpenFlow switch mainly consists of three parts: OpenFlow table, OpenFlow secure channel and OpenFlow protocol [20]. Packets are forwarded based on flow tables and controller can modify these flow tables via secure channel using OpenFlow protocol. The flow tables consist of a set of flow entries and each flow entry is associated with actions [20]. When OpenFlow switch receives a packet, it looks up the flow table (comparing received packet header with entries of the flow tables). If the packet header matches the flow table, associated actions are executed. According to the OpenFlow specification [20], actions include packet forwarding, packet

modification and addition, removing packet header, dropping packet etc. On the other hand, if the packet doesn't match, it is transmitted to controller and the controller builds a new flow table. More information about the OpenFlow switch is explained in the OpenFlow Standard [3]. The details of OpenFlow components are described in the following section.

MODEL SDN SWITCH

3.1. Switch management software

A switch is a crucial component of a network that links various devices, including computers, wireless access points, printers, servers, and others. Connected devices can exchange data and communicate with one another thanks to a switch. By transmitting a network packet just to the one or more devices it is meant for, switches control the flow of data across a network. With the ability to identify every networked device connected to a switch by its network address, the switch can direct traffic, enhancing the network's security and effectiveness. Monitoring tools can detect issues before they become serious. There are several warning indicators that suggest a problem is developing. By swiftly resolving those issues, significant issues are avoided and a network outage is avoided.

3.2. Types of switch monitoring software

There are four types of switch monitoring software:

- Switch port monitor – Records the activities of each active port on a switch
- Network traffic analyzer – Samples passing data frames and compiles traffic statistics
- Network performance monitor – Checks on the statuses of switches
- Network configuration manager – Controls the settings of switches

The switches in your network contain every element that might enhance or reduce network transmission rates. Switches therefore control the rate at which data can move throughout your network.

3.2.1. Switch Port Monitoring

If network is experiencing performance issues, one or more of switches may be overloaded. Initially, switches evenly share their capacity among all of their ports. A port that is functioning can tell which ports are using and which are empty. Then, it reallocates capacity and allocates more memory to high throughput ports than to low throughput ports. "Autosensing" is the name given to this procedure. However, if traffic patterns alter, things might quickly turn bad. The switch will automatically raise the port's capacity if the use of a new application causes more traffic to emerge on that port. However, other ports must no longer have such capability. Another area of the network that was previously operating without issue may now experience issues. Up until a crisis, capacity problems might ping through from one switch to another. Switch port

monitoring prevents this issue and offers resources to control each port's traffic capacity.

3.2.2. Network traffic analyzer

Another method for guaranteeing that traffic moves through the network at top speed is to use network traffic analyzers. As previously stated, the true problem with switch capacity is traffic capacity. As a result, traffic flow analyzers gather throughput data from the switch, just like switch port managers. NetFlow monitors, so named after the Cisco switch statistics collection protocol, are another term for network traffic analyzers. Other proprietary protocols include AppFlow, J-Flow, IPFIX, and Netstream from Huawei and Juniper Networks, respectively (Citrix). sFlow is the primary open standard. You should make sure that the network traffic analyzer you select can communicate with the traffic flow analysis protocols that all of your switches employ. Over time, NetFlow traffic analyzers create a picture of activity on each switch. They display capacity on each switch port and real-time traffic statistics for each cable segment. Historical data reveals patterns in demand variation over time. Additionally, it can pinpoint which applications and, by extension, which protocols, are driving the most traffic. Implementing traffic shaping strategies like prioritising and queuing is aided by traffic analysis technologies.

3.2.3. Network performance monitor

Network performance monitors make up the majority of switch monitoring software that you may come across. These monitoring systems take advantage of the Simple Network Management Protocol (SNMP). An SNMP agent is included with every switch. The SNMP agent fills out a form and gathers status data from the switch that hosts it. However, it won't make that form (known as a MIB) available unless an SNMP manager asks for it. SNMP managers are network performance monitors. The SNMP agent will immediately send out a status report in the event of a significant issue on a monitored switch. A trap is what this is. SNMP Trap messages are interpreted as alerts by network performance monitors. The monitoring console will display any alerts and they can also be sent through email or SMS.

3.2.4. Network configuration manager

Switches functionality can be changed by adjusting their settings. Unfortunately, some switch configurations make it simpler for hackers to browse the system covertly. We refer to these setup flaws as "vulnerabilities." If you want to tighten up security on your system, you can subscribe to a vulnerability screening service that will highlight improvements you need to do. The configuration of switches is one topic covered by these evaluations. An image of each switch type's configuration is kept on file by a configuration manager. The identical configuration will then be applied to all identical-make and similar-model switches in your network. The switches' settings are periodically checked by the configuration monitor. The configuration manager applies the saved image again to the affected switch, restoring its approved settings, if it detects a change.

3.3. A Model Switch Management software

ManageEngine A complete network switch monitoring tool is OpManager. Switch performance, health, and availability may all be tracked using OpManager[21]. Switches in your network are automatically found and added to a unique switch map by OpManager's switch monitoring feature. Additionally, every switch port is located and organized logically on the map. Operators can see the status and availability of switch ports by using the switch monitoring feature of OpManager. When a switch port or the switch goes down, OpManager immediately notifies operators and keeps track of the situation. As a best practice for Switch monitoring, operators can configure OpManager to only monitor key ports, which stops the generation of pointless alarms. OpManager also provides visibility into the status of the spanning tree, displaying which ports are forwarded and which ones are blocked.

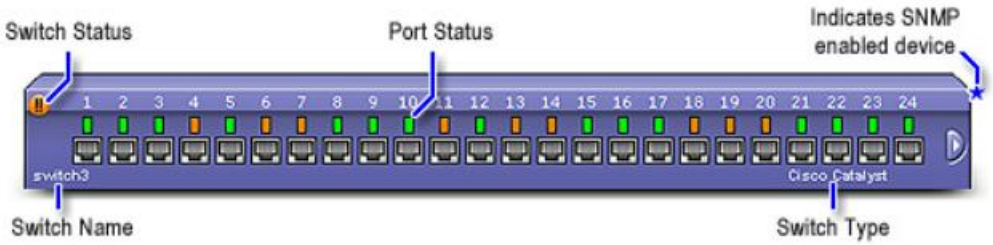

Figure 7: OpManager Switch Monitoring Software Features

3.4. Switch and Switch Ports availability monitoring

OpManager can create business views (maps) to graphically visualize entire LAN. OpManager can automatically send alerts when a link goes down. OpManager's reporting functionality also provides with a detailed availability report of Switches and Switch Ports. It can be use these reports to ensure that your SLAs are being met.

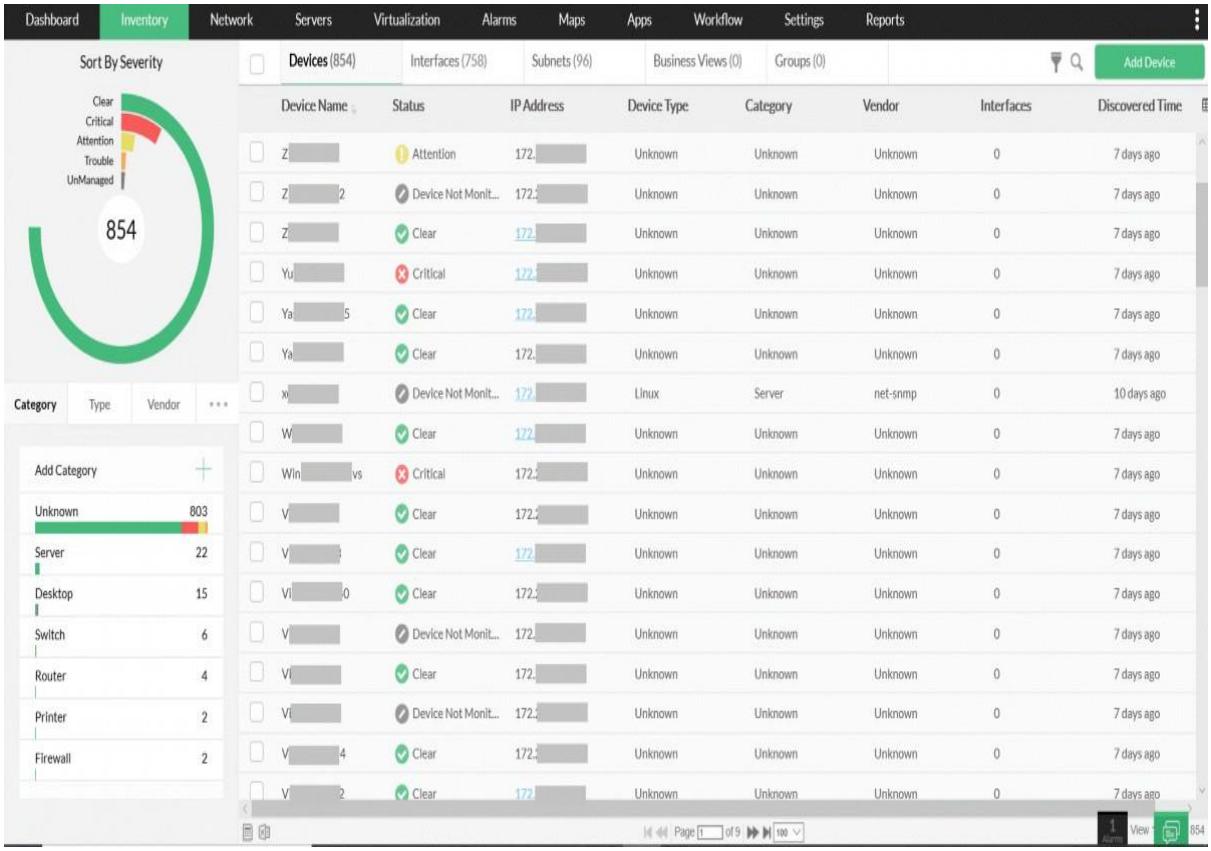

Figure 8: Switch and Switch Ports availability monitoring

3.5. Port-wise Traffic Monitoring

monitor and troubleshoot Switch Ports using OpManager to check for traffic, utilization, faults, and Service Level Agreement (SLA) compliance. OpManager aids in the identification of the top talkers on the LAN by providing precise information on port traffic and utilization.

- Monitor Port utilization and traffic with threshold alerts.
- Detect potential Broadcast Storms and pro–actively prevent the same.
- Identify highly utilized and under-utilized ports.
- Get alerted when a port start discarding packets

3.5.1. Switch Monitoring Tools

Real Time Switch Monitoring Tools such as Switch Port Mapper and STP Tool are bundled with OpManager.

3.5.2. Switch Port Mapper

An effective tool that is integrated into OpManager is the Switch Port Mapper. You can use it to rapidly obtain a list of the devices plugged into the switch ports.

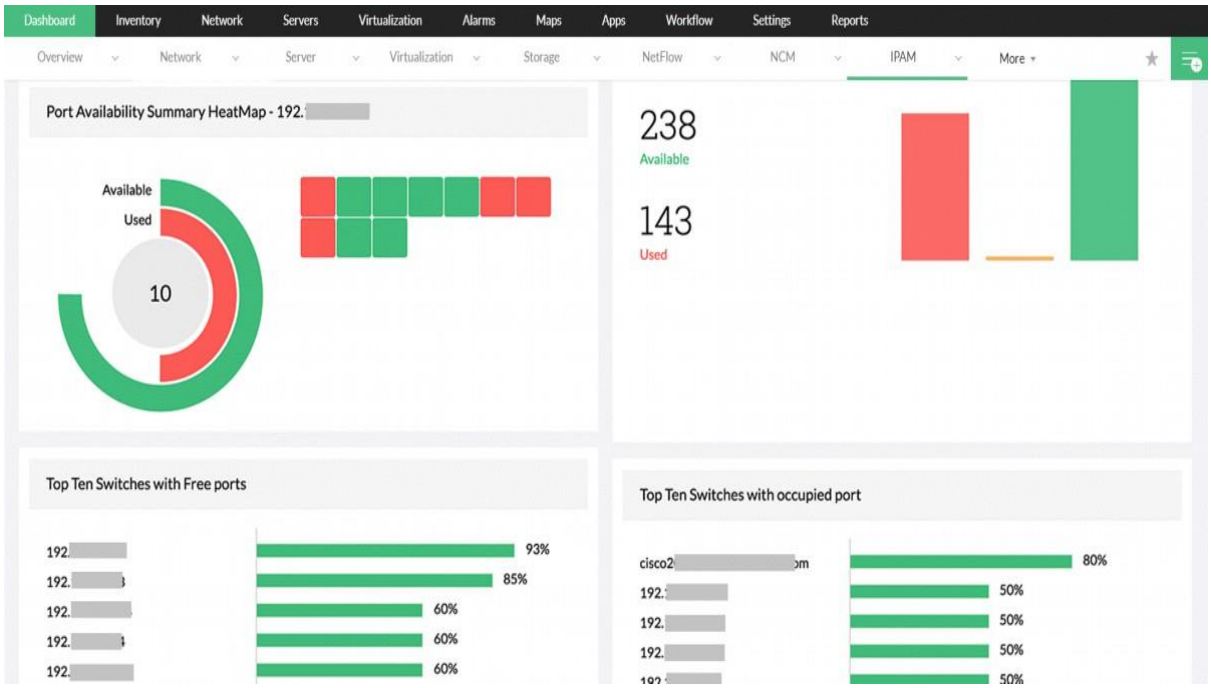

Figure 9: Switch port Mapper

3.5.3. STP Tool:

Using the STP Tool, it may inspect the specifics of the Spanning Tree Protocol for each port. This provides important details on each port's spanning tree state, including which ports are blocking and which ports are forwarding, among other things.

3.6 Switch Fabric

3.6.1. Overview:

- Redundant field replaceable power supply and fan units
- Support for “Front to Back” or “Back to Front” air flow direction
- Can be easily re-configured at each startup.
- Can be easily re-programmed as per requirement for new switch.

- Fast power supply switching.
- Cooling fans can be used in breathing mode or with/without tacho feedback or with/without PWM control or any combination of these modes.

3.6.2. Physical Overview

3.6.2.1 Dimension

Table I: Dimension of Switch Fabric

	Inches	Millimetres
Length	17.42	442.46
Width	1.73	43.95
Height	9.7	246.38

3.6.2.2. Top view:

The top view of Platform management system for switches PCBs

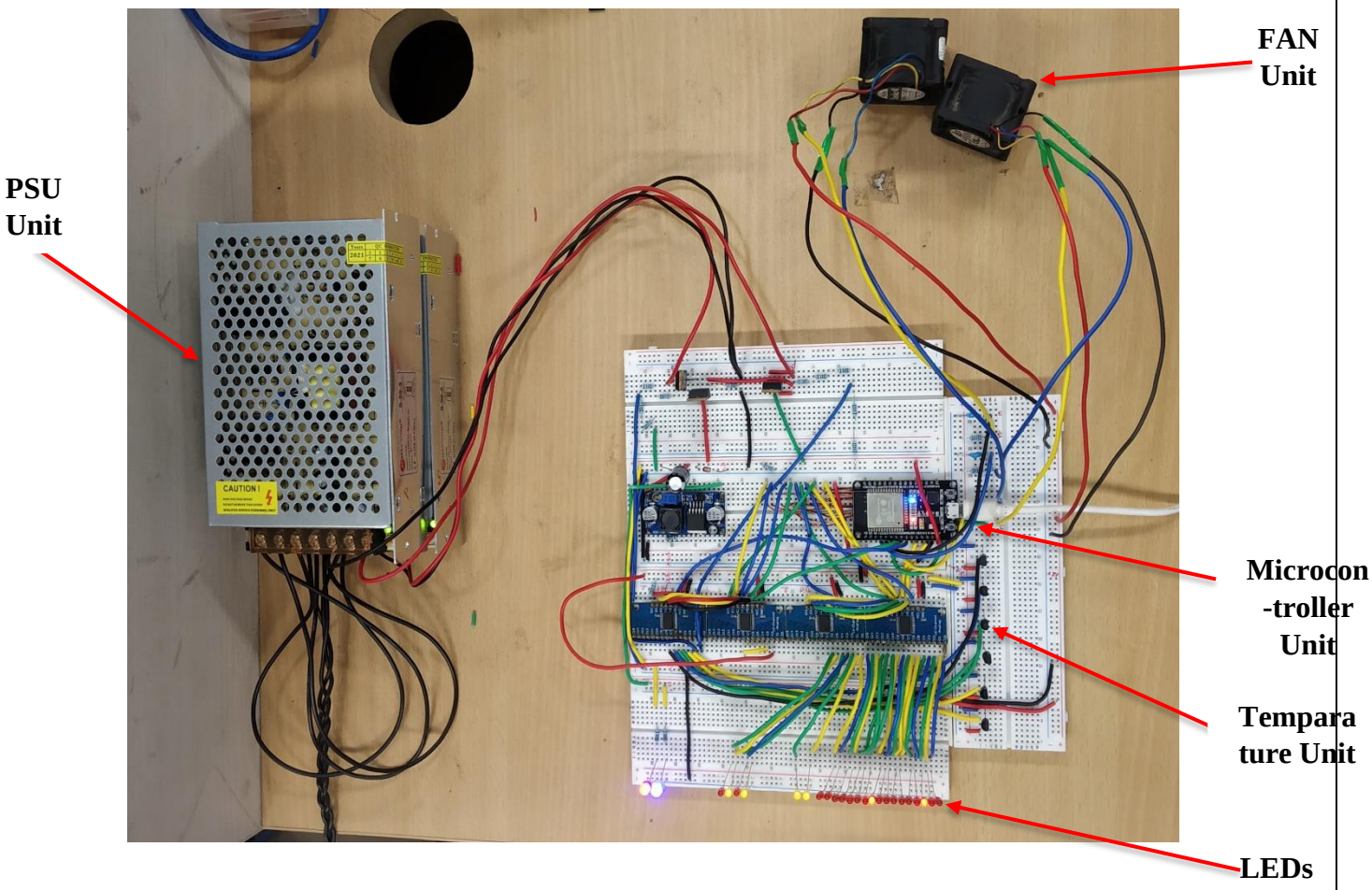

Figure 10: Top View of Platform management System for switches

3.6.3. Front Panel LED Definitions

Table II: Front Panel LED Definitions

LED Name	Description	State
PSU1	Led to indicate status of Power Supply 1	Yellow - Normal Red - Fault Off – No Power
PSU2	Led to indicate status of Power Supply 2	Yellow - Normal Red - Fault Off – No Power
FAN	LED to indicate the status of the system fans	Yellow – All fans operational Off – fan fault
Switching	LED to indicate the status of 5V or 12V	Red-5v Blue-12v
Port LED	LED to indicate the status Port	On Red & Yellow - port has link Off – No link

PLATFORM MANAGEMENT SYSTEM

4.1. Overview

The data packets are forwarded from one device to another device by the 'Network Switch'. The other functionality of the network switch is interconnecting each other between the Network devices of the network system. Different companies are working on it, and they are making a hardware management system, these hardware management systems are different for different types of systems. But this time is a distributed era, so, different changes are required according to different applications, therefore, hardware management systems also require changes according to this application. Different companies provide some solution, like SDN's controller, then switches components etc. Repeated changes over this management system is problematic and it is not possible to change the system from scratch. Moreover, our growing technology wants a sophisticated and flexible system where the requirements are permitted easily according to users, that is a reprogrammable, reconfigurable, reassignable universal system.

4.2. Features:

There are several advantageous features of this developed system are summarized below:

1. Traditional hardware management is not easily customized or easily reconfigurable because it is developed according to user requirements. But this developed system is easily configurable according to the user requirements.
2. The main feature of this system is that the reprogramming can be done according to the requirement of the new hardware ignoring its complexity.
3. This system can provide a fast power supply switching mechanism to the hardware.
4. Cooling fans are incorporated into the system with various modes: breathing mode, with/without tacho feedback mode, with/without pwm speed control mode. If the system temperature is gradually increased, the tacho sensor feedback is calculated and pwm signal given to the fan, as a result, the fan speed is increased, and the system is cooled down promptly.
5. A Pulse Width Modulation (PWM) signal is incorporated to this system for easier control of speed of fan which is used for cooling purposes. Several LEDs are used to indicate the system performance, different light intensity of LED indicates the different conditions of the system, this control is done by the PWM technique.
6. It is a more reliable and cost-effective system than other traditional hardware systems.

The aforesaid features are not reconfigured or re-prommable in the traditional switches, but in the developed hardware management system, the aforesaid features are easily reconfigured or reprogrammable.

HARDWARE PART OF PLATFORM MANAGEMENT SYSTEM

5.1 Overview-

Hardware is the backbone of the platform management system. Without hardware, the platform management system is not existing. The whole hardware management system has been done in a breadboard and this work has been accomplished very successfully. This sophisticated hardware system has different parts and subparts which are described below. For the reductant of the complicity of the whole hardware system, the components are and the part of the system are separated and all the necessary parts are given individually.

The point no 1 indicates the two-power supply, and point no. 2 indicates the combination of MOSFET and schottky diode. Basically, point no. 2 is used to select the power supply. Power has been taken from the primary power supply, when the primary power supply is failed, then the point no. 2 activates the secondary power supply. Point no 3 indicates a Microcontroller unit which is named as ESP32 [23].

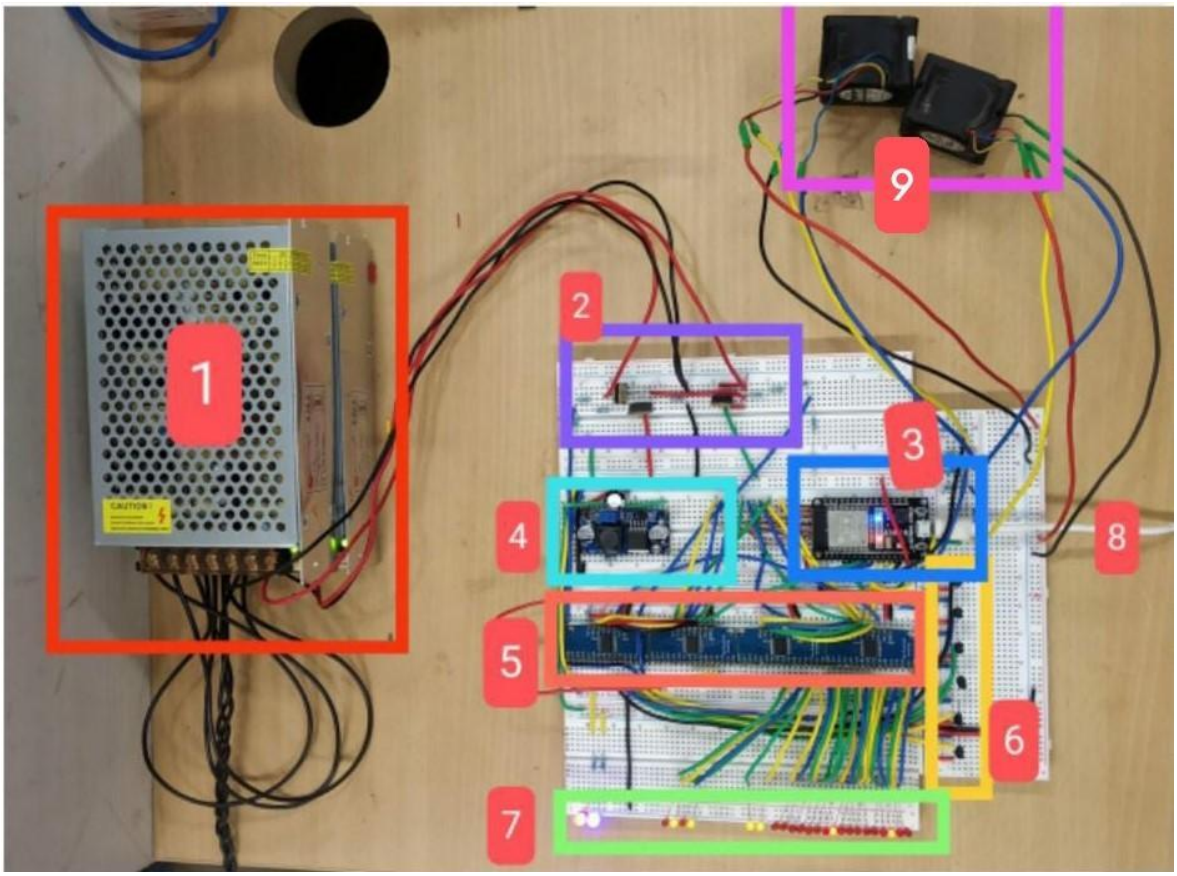

Figure 11: Circuit Design in breadboard

Point no. 4 denotes the boost converter which is used the step up the voltage from 5 V to 12 V. Mainly, 12 V is used for the fan operation. Point no. 5 indicates the multiplexer unit or multiplexer array. In this system, 4 16-bit MUX arrays have been created. Pairs of

multiplexers are used as a unit. So the final one 32-bit multiplexer unit is used as input pins and another 32-bit multiplexer unit is used as output pin as the GPIO pin numbers are less in ESP32 microcontroller. The ESP32 Microcontroller unit has less number of GPIO pins and processing power is high, therefore, extra microcontroller is not required for this purpose. In previously said that, 4 Multiplexer unit is used as a input output pin and in this pin are used to define a signal or another way, it has been said that this is defined as a signal input or output. Point no. 6 indicates the temperature unit array, the maximum no. of this array is being 6 and minimum is being 0. Point no. 7 denotes the output LED array. The main functionality of this array is that it provides the physical notation of the present system status. Like Fan condition, LED light condition, port temperature condition, power supply condition etc. Point no. 8 denotes the serial communication wire which is connected over the UART. Finally, point no 9 denotes the two fan units. If anyone wants to use less or more fans into the system, it can be done just wiring more or less fans.

5.2 Hardware Design:

- **Power Supply Unit:**

An electrical device known as a power supply provides electricity to an electrical load. A power supply's primary function is to transform electrical current from a source into the proper voltage, current, and frequency needed to drive a load. Because of this, power supplies are sometimes known as electric power converters. While some power supplies are integrated into the load appliances they power, others are independent standalone pieces of equipment. In this Platform management System two 5V 10A - 50W Metal Power Supply used.

Figure 12: 5V 10A - 50W Power Supply

Key Specifications:-

- Output Voltage-5V
- Type-Enclosed Type
- Approx. Wattage -50W
- Mechanical Form -Enclosed
- Input Voltage-100-240V auto-select
- No load power consumption <0.2W for 35W/50W; <0.3W for 75W/100W

In this system two power Supply is being used. Because if one power supply is being stopped another power supply will 'ON' automatically and network administrator will understand this and will repair the first one. The second power supply is basically used for backup system.

Circuit Digram:

Figure 13 : shows Two power supply are being sandwich to each other if one power supply failed another fails turns on the secondary power supply.

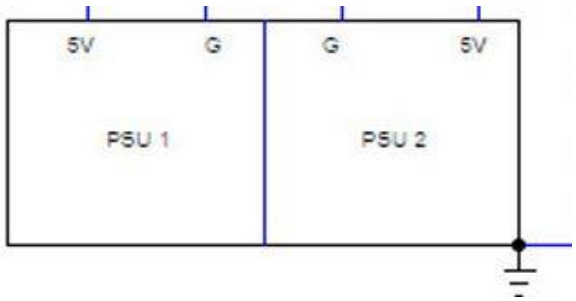

Figure 13: Two power Supply

➢ Power Supply Section Unit:

The MOSFET and schottky diode combination is shown in point. Basically, the power supply is chosen using point. When the primary power source fails then turns on the secondary power supply, drawing power from the primary power supply. The MOSFET (Metal Oxide Semiconductor Field Effect Transistor) transistor is a semiconductor component that is frequently used in electrical devices for switching and signal amplification. Due to its small size, a MOSFET is either a core or an integrated circuit that is designed and manufactured on a single chip. The area of switching in electronics has changed as a result of the invention of the MOSFET device. Let's begin with a thorough explanation of this idea. While N-channel MOSFET is used to switch the negative supply to the motor for backward direction, P-channel MOSFET is used to switch the positive supply to the motor for forward direction. And another side Similar to other junction diodes, the Schottky Diode is a type of semiconductor diode that can be utilised in a number of waves shaping, switching, and rectification applications. The key benefit is that a Schottky Diode's forward voltage drop is significantly lower than the 0.7 volts of a typical silicon pn-junction diode. In this System one N-channel MOSFET (IRFZ44N), one P-channel MOSFET(IRF4905) and one MOSPEC basically it is Schottky barrier rectifier diodes (PBYR2545CT).

Figure 14: refers P-channel MOSFET.

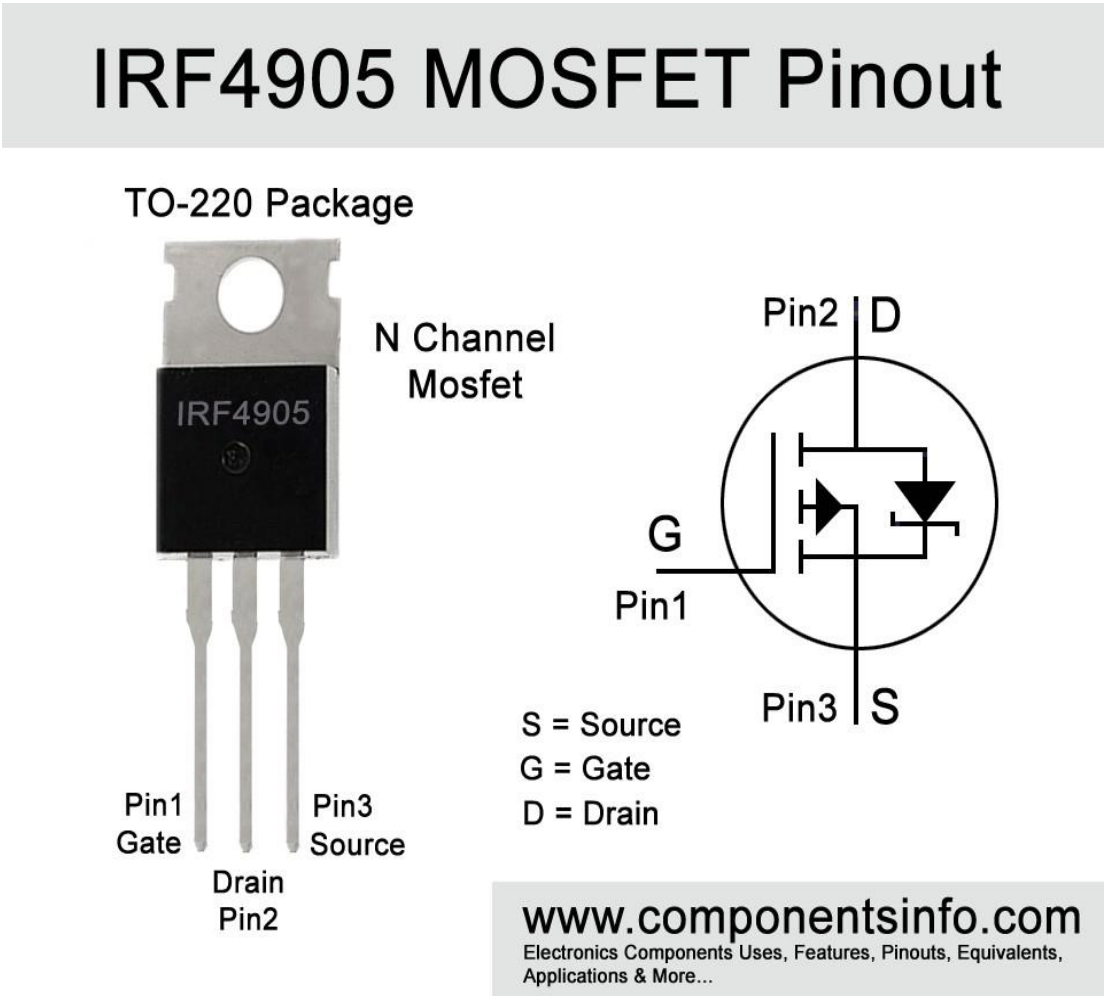

Figure 14: P-channel MOSFET

- **Features / Technical Specifications:**
  - Package Type: TO-220
  - Transistor Type: N Channel
  - Max Voltage Applied From Drain to Source: -55V
  - Max Gate to Source Voltage Should Be: ±20V
  - Max Continues Drain Current is : –74A
  - Max Pulsed Drain Current is: –260A
  - Max Power Dissipation is: 200W
  - Minimum Voltage Required to Conduct: -2V to -4V
  - Max Storage & Operating temperature Should Be: -55 to +175 Celsius

Figure 15 : refers N-channel MOSFET

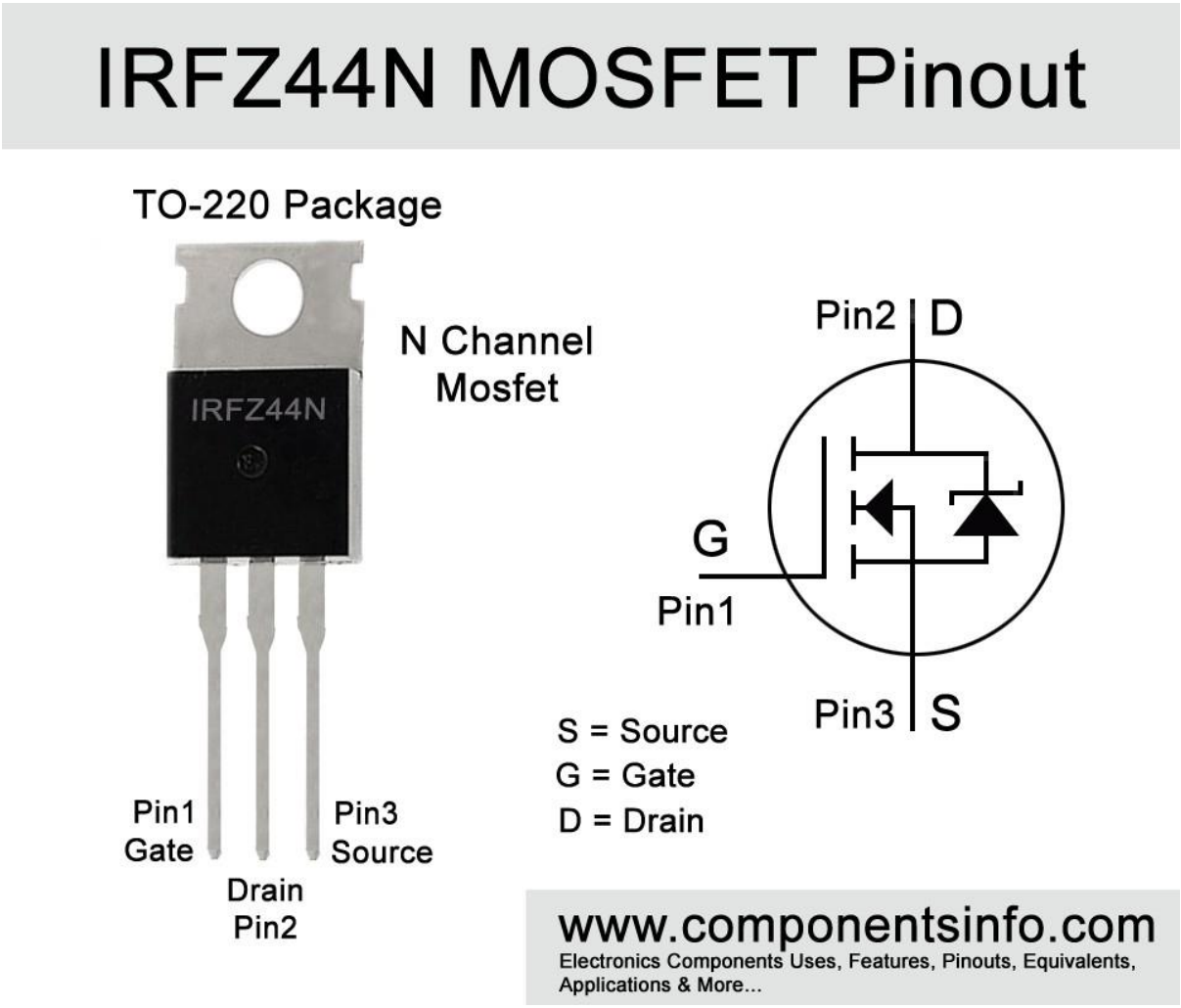

Figure 15: N-channel MOSFET

- **Features / Technical Specifications:**
  - Package Type: **TO-220**
  - Transistor Type: **N Channel**
  - Max Voltage Applied From Drain to Source: **55V**
  - Max Gate to Source Voltage Should Be: ±**20V**
  - Max Continues Drain Current is : **49A**
  - Max Pulsed Drain Current is: **160A**
  - Max Power Dissipation is: **94W**
  - Minimum Voltage Required to Conduct: **2V to 4V**
  - Max Storage & Operating temperature Should Be: **-55 to +170 Centigrade**

Figure 16 : refers Schottky barrier rectifier diodes

Figure 16: Schottky barrier rectifier diodes

- **Futures:**
  - Low forward volt drops Fast switching
  - Reverse surge capability
  - High thermal cycling performance
  - Low thermal resistance PBYR25 45CT, PBYR2545CTB series SYMBOL QUICK REFERENCE DATA VR = 40 V/ 45 V a1 1 k 2 a2 3 IO(AV) = 30 A VF ≤ 0.

❖ Circuit Digram:

Figure 17 : shows Power Supply Section Unit.

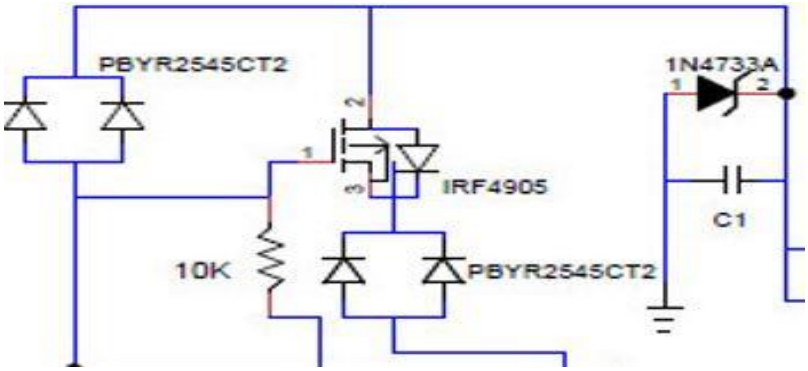

Figure 17: Power Supply Section Unit

➢ Microcontroller Unit:

An embedded system's microcontroller is a small integrated circuit that controls a single process. On a single chip, a typical microcontroller has a CPU, memory, and input/output (I/O) peripherals. Sometimes referred to as an embedded controller or microcontroller unit (MCU), microcontrollers are found in vehicles, robots, office machines, medical devices, mobile radio transceivers, vending machines and home appliances, among other devices. They are essentially straightforward mini-personal computers (PCs) without a complicated front-end operating system that are used to operate minor aspects of larger components (OS).

This system ESP32 is being used for input output purpose, feedback purpose, Temperature reading purpose etc.. ESP32 is a low-cost System on Chip (SoC) Microcontroller from Espressif Systems, the developers of the famous ESP8266 SoC. It is a successor to ESP8266 SoC and comes in both single-core and dual-core variations of the Tensilica's 32-bit Xtensa LX6 Microprocessor with integrated Wi-Fi and Bluetooth.

❖ Specifications of ESP32

ESP32 has a lot more features than ESP8266 and it is difficult to include all the specifications in this Getting Started with ESP32 guide.

- Single or Dual-Core 32-bit LX6 Microprocessor with clock frequency up to 240 MHz.
- 520 KB of SRAM, 448 KB of ROM and 16 KB of RTC SRAM.
- Supports 802.11 b/g/n Wi-Fi connectivity with speeds up to 150 Mbps.
- Support for both Classic Bluetooth v4.2 and BLE specifications.
- 34 Programmable GPIOs.
- Up to 18 channels of 12-bit SAR ADC and 2 channels of 8-bit DAC
- Serial Connectivity include 4 x SPI, 2 x I^2C, 2 x I^2S, 3 x UART.
- Ethernet MAC for physical LAN Communication (requires external PHY).
- 1 Host controller for SD/SDIO/MMC and 1 Slave controller for SDIO/SPI.
- Motor PWM and up to 16-channels of LED PWM.
- Secure Boot and Flash Encryption.
- Cryptographic Hardware Acceleration for AES, Hash (SHA-2), RSA, ECC and RNG.

Figure 18 : Shows ESP32 microcontroller

Figure 18: ESP32 microcontroller

Figure 19: Layout of ESP32 microcontroller

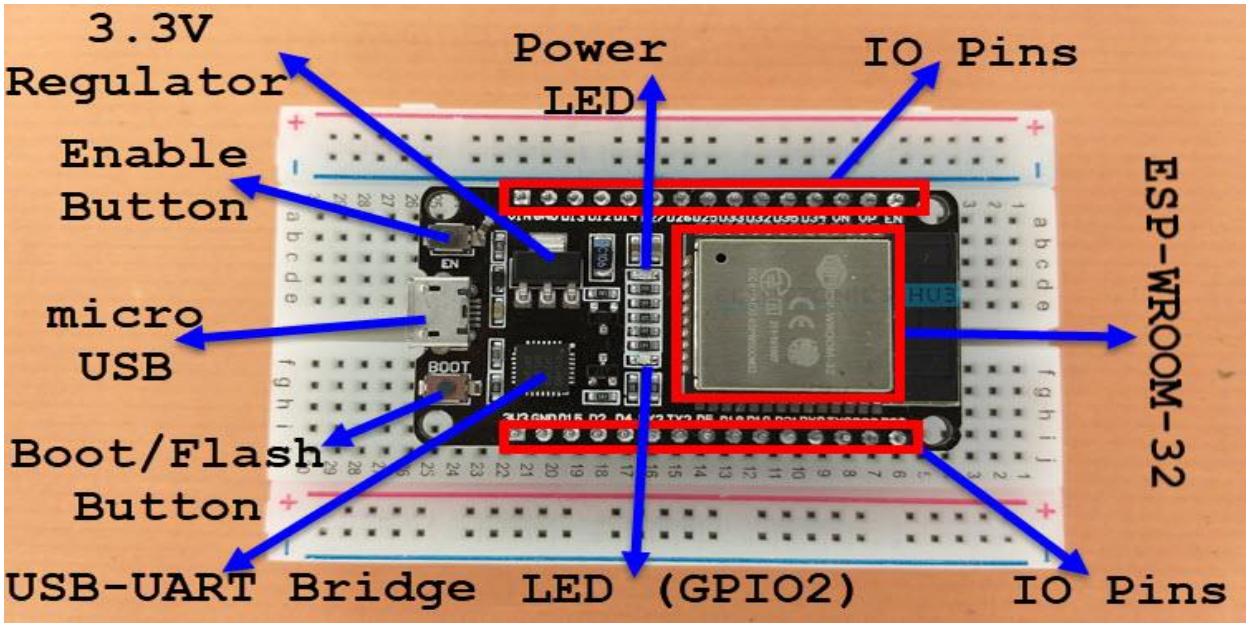

Figure 19: Layout

the ESP32 Board consists of the following:

- ESP-WROOM-32 Module
- Two rows of IO Pins (with 15 pins on each side)
- CP2012 USB – UART Bridge IC
- micro–USB Connector (for power and programming)
- AMS1117 3.3V Regulator IC
- Enable Button (for Reset)
- Boot Button (for flashing)
- Power LED (Red)
- User LED (Blue – connected to GPIO2)
- Some passive components

❖ Pinout of ESP32 Board

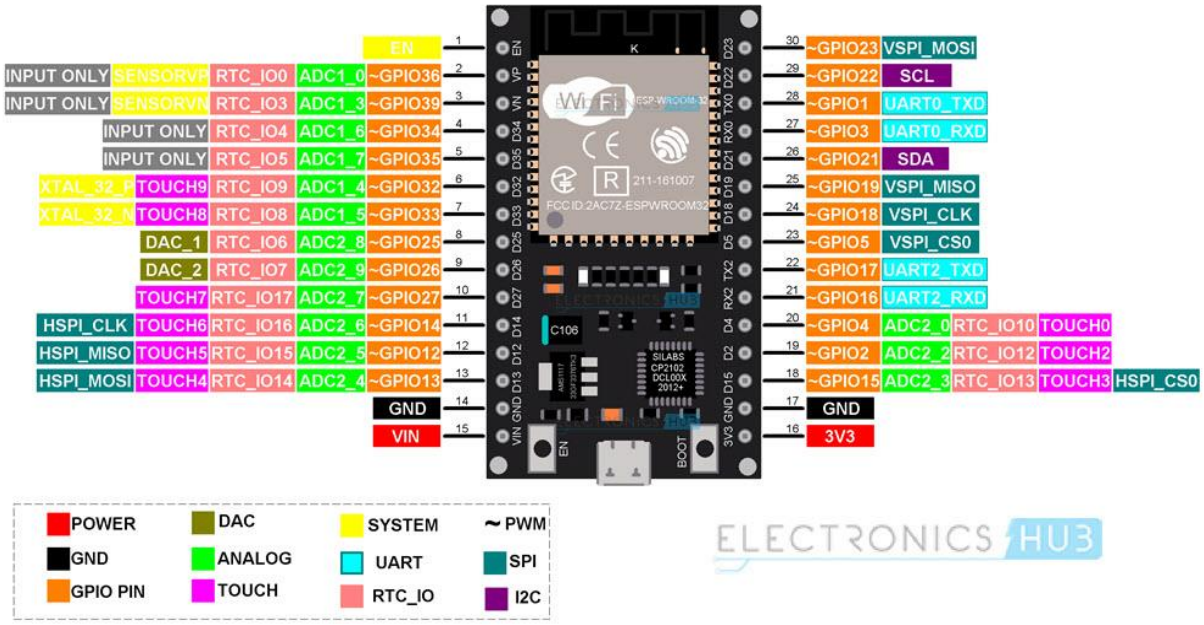

Figure 20: Pinout

➢ Boost Converter Unit:

A boost converter (step-up converter) is a DC-to-DC power converter that steps up voltage (while stepping down current) from its input (supply) to its output (load).

Here 'XL6009E1' is being used. XL6009E1 is a high-performance boost module with 4A switching current. Adopting XL6009E1 chip with high frequency switching technology as the core chip, superior performance. Ultra-wide input voltage 3V ~ 32V, the best operating voltage range of 5 ~ 32V. Ultra-wide output voltage 5V ~ 40V.

The boost converter which is used the step up the voltage from 5 V to 12 V. Mainly, 12 V is used for the fan operation. Figure: refers XL6009E1 DC-DC Adjustable Step-up Power Boost Voltage Converter Module.

Figure 21: XL6009E1 Step-up Power Boost Voltage Converter Module

Circuit Diagram:

Figure 22: refers Circuit diagram of Boost Converter

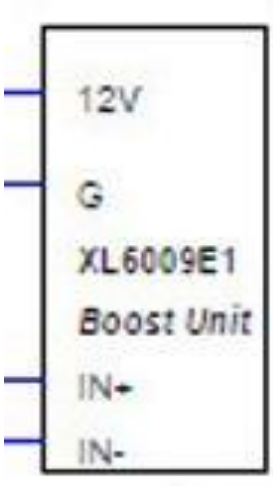

Figure 22:Circuit diagram of Boost Converter

➢ Multiplexer Unit:

The multiplexer unit or multiplexer array is being discussed in this system. A multiplexer is a sort of combinational circuit that accepts numerous data inputs but only produces one output. A combinational circuit known as a demultiplexer accepts just one input but routes it through several outputs. From parallel to serial conversion is carried out via a multiplexer. Here '74HC4067' is being used for Multiplexer and De-multiplexer unit. Four 16-bit MUX arrays have been made for this system. Multiplexers are utilised in pairs. Due to the fact that the ESP32 microcontroller's GPIO pin numbers are smaller, the final 32-bit multiplexer unit is utilised as an input pin and a second 32-bit multiplexer unit is used as an output pin. A separate microcontroller is not necessary for this task because the ESP32 Microcontroller unit has a fast-processing speed and few GPIO ports. This pin is utilised to define a signal or, to put it another way, it has been mentioned that this is defined as a signal input or output in the previously stated 4 Multiplexer unit.

Details:

This is a breakout board that mounts a 74HC4067 which is a 16-channel Analog multiplexer/demultiplexer that can route both analog and digital signals in both directions.
1 of 16 input channels can be routed to 1 output or 1 input can be routed to 1 of 16 output channels. It can handle analog signals such as from analog sensors or a bank of potentiometers or it can handle digital signals such as from switches, digital sensors or even serial communications. The 74HC4067 can operate over the range of 2 to 6V, so it is compatible with both 3.3 and 5V logic.
Four address lines (S0-S3) select one of the 16 channels and connects it to the input/output pin (SIG). It uses binary addressing, so address 0000 is Channel 0, address 1111 is Channel 15. When a channel is ON, it has a resistance of about 70 ohms which allows signals to flow both ways. With a 5V power supply, we measured about 60 ohms. Maximum current is 25mA through any of the channels.
There is an enable pin (EN) that is active LOW and defaults to that value. When LOW, the device enables the channel selected by the address lines. If EN is pulled HIGH, all channels are disabled.

On the electronics module there are 2 rows of header solder points. These can be used to connect wires to or header pins depending on the application.

- 1×8 Header Location
- SIG = Signal Input / Output. This will usually connect to an analog input or digital I/O on the microcontroller
- S3 = Binary Address Bit 3. The address bits (3…0) connect to 4 digital output pins on the microcontroller to select channel.
- S2 = Binary Address Bit 2
- S1 = Binary Address Bit 1
- S0 = Binary Address Bit 0
- EN = Enable. Internally pulled LOW to enable by default. Can be pulled HIGH to disable all channels.
- VCC = 2 to 6V. Usually connected to 5V or 3.3V to match the microcontroller power.
- GND = Ground, must be common with the microcontroller.
- 1×16 Header Location
- C15 = Channel 15
- C0 = Channel 0

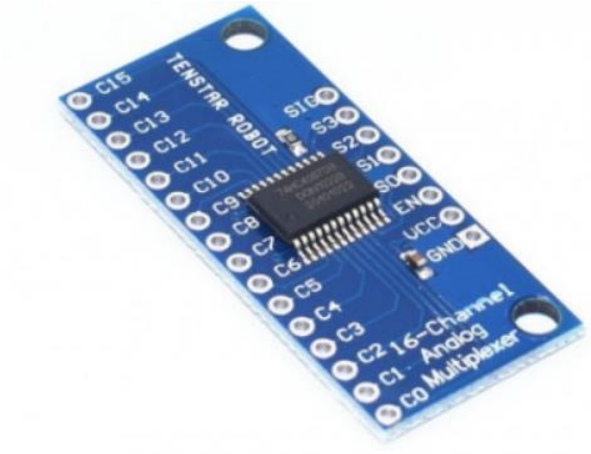

Figure 23: 16-Channel Analog Digital Multiplexer Module

Circuit diagram:

Figure 24: refers Circuit Diagram of Multiplexer Unit

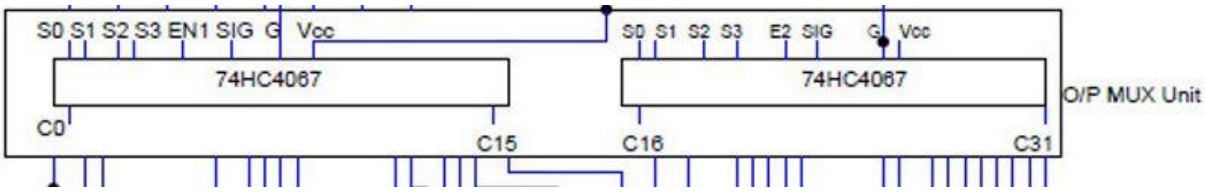

Figure 24: Circuit Diagram of Multiplexer Unit

- ## Temperature Unit Array:

A temperature sensor is an electronic device that measures the temperature of its environment and converts the input data into electronic data to record, monitor, or signal temperature changes.

In this System LM35 is being used for temperature sensor purpose. LM35 is a temperature measuring device having an analog output voltage proportional to the temperature. It provides output voltage in Centigrade (Celsius). It does not require any external calibration circuitry. The sensitivity of LM35 is 10 mV/degree Celsius. As temperature increases, output voltage also increases.

Features:

- LM35 is a temperature measuring device having an analog output voltage proportional to the temperature.
- It provides output voltage in Centigrade (Celsius). It does not require any external calibration circuitry.
- The sensitivity of LM35 is 10 mV/degree Celsius. As temperature increases, output voltage also increases.

  E.g. 250 mV means 25°C.

- It is a 3-terminal sensor used to measure surrounding temperature ranging from -55 °C to 150 °C.
- LM35 gives temperature output which is more precise than thermistor output.

Pin Details:

Figure 25: refers LM35 pin diagram.

VCC: Supply Voltage (4V – 30V)

Out: It gives analog output voltage which is proportional to the temperature (in degree Celsius).

GND: Ground

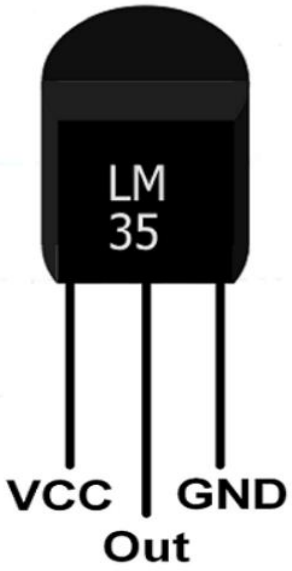

Figure 25: LM35

Circuit Diagram:

Figure: indicates the temperature unit array, the maximum number of this array is being 6 and minimum is being 0.

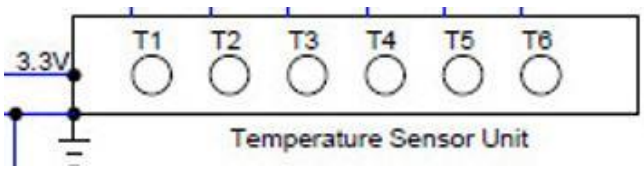

Figure 26: Temperature Sensor unit

➢ Output LED array

output LED array, main functionality of this array is that it provides the physical notation of the present system status. Like Fan condition, LED light condition, port temperature condition, power supply condition etc. here 5 mm different type color LEDs used for different purposes.

Table III: LED and its indication

Name	State
PSU1	Yellow - Normal Red - Fault Off – No Power
PSU2	Yellow - Normal Red - Fault Off – No Power
FAN	Yellow – All fans operational Off – fan fault
Switching	Red-5v Blue-12v
Port LED	On Red & Yellow - port has link Off – No link

Circuit Diagram:

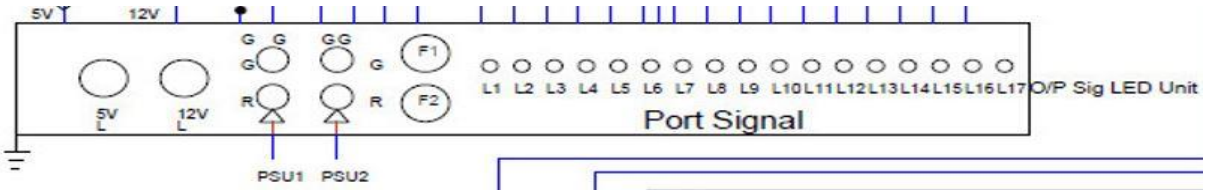

Figure 27: Output LED array

➢ Serial Communication:

Serial communication is the process of progressively delivering data, one bit at a time, through a communication channel or computer bus, in telecommunication and data transmission. This contrasts with parallel communication, when multiple bits are sent via a link with many parallel channels as a single unit.

In this system UART (Universal Asynchronous Receiver/Transmitter) serial communication is being used to give communication instruction to the ESP32 microcontroller. In software section it will describe briefly.

➢ Fan Unit:

Intel created a standard for connecting fans using 4 wires. The main goal of developing a new standard is to make it possible to measure revolution at low fan speeds and to precisely manage revolution at all speed ranges.

In this system two 4 wire fan assembled. If anyone wants to use less or more fans into the system, it can be done just wiring more or less fans.

Signal description

Connector pinout

Pin	Function	Wire color
1	GND	Black
2	12V	Yellow
3	Sense (tach.)	Green
4	Control (PWM)	Blue

Figure 28: connection pinout

Signal **GND** is ground and **12V** is voltage supply for fan.

Signal **Sense (tachometer)** provides two pulses per revolution of fan. Output is opened collector and main board must have pull-up resistor to 12V.

Signal **Control (PWM)** is input for PWM pulses. Base frequency is 25kHz and it is acceptable from 21kHz to 28kHz. Input has TTL level and includes pull-up resistor to 5V or to 3.3V in new constructions. Signal is not inverted and 100% PWM means maximal revolutions of fan. Motherboard has open-collector type output. This construction guarantee, that with disconnected PWM signal will runs fan with maximum revolutions.

Figure 29: Demo 4 wire fan

5.3 Circuit diagram of Hardware Platform Management System

The circuit diagram of the Hardware Management System has been shown in Figure:

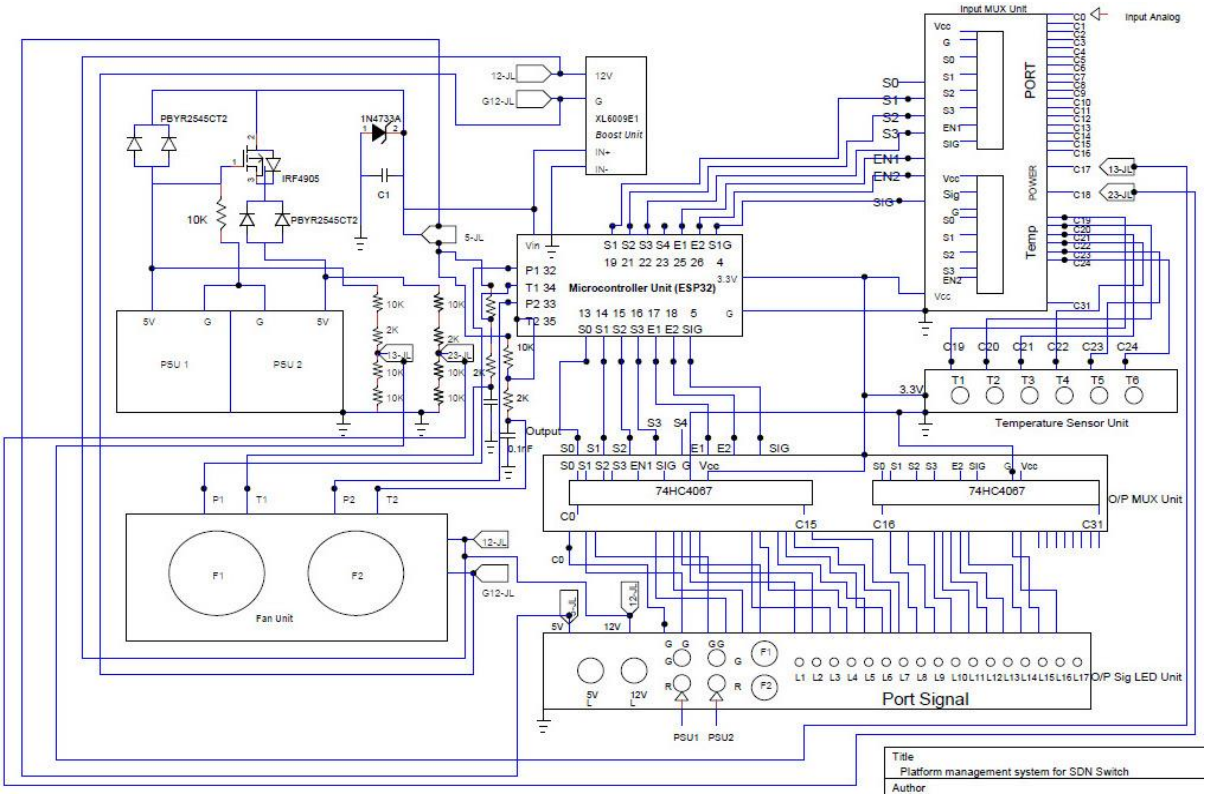

Figure 30: Circuit Diagram of the hardware platform management system

In this system, a microcontroller is used namely ESP32. The ESP32 Microcontroller unit has less number of GPIO pins and processing power is high, therefore, extra microcontroller is not required for this purpose. There are 32 channels of analog inputs which are connected with different sensors. According to requirements, like where the analog signal is required or sometimes analog to digital signal is required, so, these functions are accomplished by the multiplexer handler function. Basically, the analog multiplexer handler function stores the analog value in 'input buffer array' and according to requirement, when digital value is needed, the function converts analog value into digital value.

'Input buffer array' is an important part of this circuit which is in a time interval like 200ms to 500 ms. To depend upon the present condition of the 'input buffer array', this task involves doing some calculations, like temperature related calculation, fan speed related calculation, power supply related calculation etc. After doing this job, it creates two different buffer arrays. The 'temperature buffer array' is used to store the calculated

temperature value of all the temperature sensors which are used in the system. The other array is the ‘output buffer array’. All the other values like present power supply state, port state, fan’s running state etc are stored in ‘output buffer array’.

Another important part of this circuit is to check the present state of the fan speed, temperature value. Several numbers of fans are incorporated in the system and temperature produced from the system is sensed and then this value is compared with the present speed of the fans. Then it will take the decision whether the fan speed is increased or decreased and this status is stored in the ‘fan speed buffer array’. If the situation is raised that the fan speed is not required to increase, it is remaining to its original speed. three arrays are required to execute this particular task namely ‘temperature buffer array’, ‘output buffer array’, ‘fan speed buffer array’. By taking required values from this three-buffer array a JSON file is created which is transferred to the parent hardware system. All time, a listener is incorporated to the serial communication task, the main functionality of the listener is whether the parent hardware requires any response from the system. If any requirement is raised in the parent system, it creates the JSON file mentioning the requirements and transfers it to the hardware management system. When the request is not made by the parent system, then the present status of the hardware is sent in a particular time interval mentioned in the configuration file. If an emergency or exception occurs then the hardware management system sends an instant message to the parent hardware and shut-down the system if required.

Different present conditions like port condition, fan condition, temperature condition, power supply condition have been checked and shown to the user through a LED panel. The output LED arrays are controlled by the output task. Output task is mainly worked with the help of ‘multiplexer handler function’. At first, the Multiplexer Handler Function reads the output buffer array and finally it predicts the LED channel and sends a signal to that channel. According to signal conditions, the LEDs are being turned on and off so fast that the human eye cannot detect the blinks. The eye sees those LEDs at ON state or OFF state.

SOFTWARE PART OF PLATFORM MANAGEMENT SYSTEM

6.1 Overview:

Software is an important part of Platform management system. The bridging between software and hardware part is necessary to accomplish the whole job successfully. Similar to hardware system, the software part has different parts which is shown in below figure. The software system has mainly five individual tasks which are not dependent on each other. All the five tasks are discussed in below:

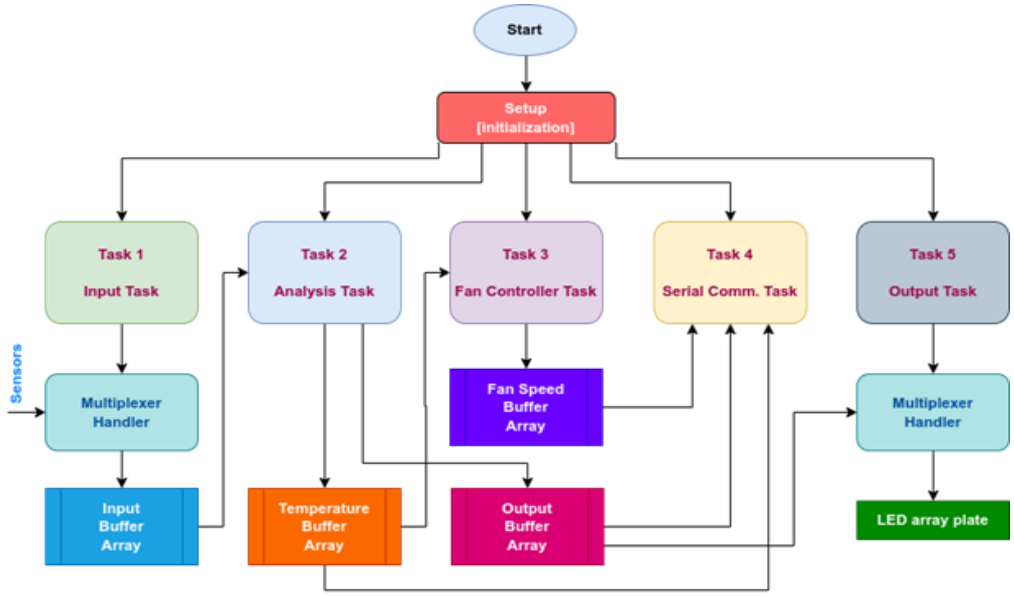

Figure 31: Platform management System flow diagram

Input Task (Task no.1)- The main function of this task is handling the multiplexer unit which provides the broad range of gpio facility. There are 32 channels of analog inputs which are connected with different sensors. According to requirements, like where the analog signal is required or sometimes analog to digital signal is required, so, this functions are accomplished by the multiplexer handler function. Figure: refers snapshot of Input Task.

```
void inputTask(void *pvParameters)
{
  (void)pvParameters;

  while (true)
  {
    for (int i = 0; i < 32; i++)
    {
      digitalWrite(IS_, (i / 16));
      digitalWrite(IS__, !(i / 16));
      input_mux.channel(i % 16);

      delay(5);

      input_buff_array[i] = analogRead(INPUT_PIN);

      delay(5);
    }

    digitalWrite(__LED_PIN, !(digitalRead(__LED_PIN)));
  }
}
```

Figure 32: Input Task

Basically, the analog multiplexer handler function stores the analog value in 'input buffer array' and according to requirement, when digital value is needed, the function converts analog value into digital value.

Analysis task (Task no.2)- This task checks the 'input buffer array' in a time interval like 200ms to 500 ms. To depend upon the present condition of the 'input buffer array', this task involves doing some calculations, like temperature related calculation, fan speed related calculation, power supply related calculation etc. After doing this job, it creates two different buffer arrays. The 'temperature buffer array' is used to store the calculated temperature value of all the temperature sensors which are used in the system. The other array is the 'output buffer array'. All the other values like present power supply state, port state, fan's running state etc. are stored in 'output buffer array'. Therefore, the main function of the analysis task is to take the value from the 'input buffer array' and create two different arrays of calculated value.

Fan controller Task (Task 3)- Its main function is to check the present state of the fan speed, temperature value which are incorporated in the system. After that, it predicts the new fan speed value, then this value is stored in the 'fan speed buffer array'. The various

number of fans has its individual fan speed value. If the fan is not incorporated in the system, then the value of the fan speed in the buffer is denoted as zero, which means the fan mode is not activated/ fan is off/ fan unavailable.

```
static volatile int rpm_counter_1 = 0;
void IRAM_ATTR fan1_rpm_counter()
{
  rpm_counter_1++;
}

static volatile int rpm_counter_2 = 0;
void IRAM_ATTR fan2_rpm_counter()
{
  rpm_counter_2++;
}
```

Figure 33: Reading Tacho signal from fan & converting to RPM

Several numbers of fans are incorporated in the system and temperature produced from the system is sensed and then this value is compared with the present speed of the fans. Then it will take the decision whether the fan speed is increased or decreased and this status is stored in the 'fan speed buffer array'. If the situation is arised that the fan speed is not required to increase, it is remaining to its original speed.

Serial communication task (Task 4)- Mainly, three arrays are required to execute this particular task namely 'temperature buffer array', 'output buffer array', 'fan speed buffer array'. By taking required values from this three buffer array a JSON file is created which is transferred to the parent hardware system. All time, a listener is incorporated to the serial communication task, the main functionality of the listener is whether the parent hardware requires any response from the system. If any requirement is raised in the parent system, it creates the JSON file mentioning the requirements and transfers it to the hardware management system. Figure: shows snapshot of serial communication task.

```
void serialCommTask(void *pvParameters)
{
  (void)pvParameters;
  while (true)
  {
    runSerial();
    vTaskDelay(10 / portTICK_PERIOD_MS);
  }

}
```

Figure 34: Serial Communication Task

When the request is not made by the parent system, then the present status of the hardware is sent in a particular time interval mentioned in the configuration file. If an emergency or exception occurs, then the hardware management system sends an instant message to the parent hardware and shut-down the system if required.

Output Task (Task-5)- Different present conditions like port condition, fan condition, temperature condition, power supply condition have been checked and shown to the user through a LED panel. The output LED arrays are controlled by the output task. Output task is mainly worked with the help of 'multiplexer handler function'. At first, the Multiplexer Handler Function reads the output buffer array and finally it predicts the LED channel and sends a signal to that channel. According to signal conditions, the LEDs are being turned on and off so fast that the human eye cannot detect the blinks. The eye sees those LEDs at ON state or OFF state.

6.2 Pin Configuration:

Output Maps

C0 - C16(C0') ==> Port activity LED pin

C17 - C18 ==> Fan activity LED pin

C19 - C22 ==> Power Supply activity LED pin

Input Maps

C0 - C16 ==> Port sensor

C17 - C18 ==> Power supply sensor

C19 - C24 ==> Temperature sensor

6.3 Software Design-

The software has been implemented in ESP32 [23] microcontrollers. The programme has been written depending upon the FreeRTOS [22] platform. Basically, FreeRTOS is a real time operating system and it is used in advanced microcontroller systems. The programme has been written in C++ language. In this system, LM35 [25][26] temperature sensors are used. The temperature value of the LM35 is calculated using this formula,

```
double LM35::getTemp()
{
    double value = analogRead(this->_analogPin);

    return getTemp(value);
}

double LM35::getTemp(uint16_t av)
{
    int ref_voltage = 3300; // in mV
    int reso_steps = 2048;
    double mVoltage = ((double)av / reso_steps) * ref_voltage;
    double tempC = mVoltage * 0.1;  // Temp in C = mV / 10 @LM35
    return tempC;
}
```

Figure 35 :Calculation of temperature value for the sensor LM35

temperature in C = output mV/10.

Where mV is

mV = (Analog Read Value / total_steps)*Ref_voltage, here tatal steps are 2048.

Again, Ref_voltage = 3.3 V = 3300 mV

An interrupt [24] handler is used on the software side to collect the fan speed values. Suppose a fan is running a particular r.p.m, the value of the r.p.m is determined by this interrupt handler. The interrupt handlers are used to count the tacho feedback for a particular time interval. For assigning the fan speed, a 8 bit PWM signal is used. The temperature is synchronized with fan speed and this synchronized speed has been calculated and assigned as a 0-255 value. When the parent hardware is woken up, a configuration file (JSON file) is sent by the parent hardware to the ESP32 microcontroller. In this JSON file, all the necessary initializing parameters are specified. For example, for a

particular temperature value is reached in a time which is used as system alert or a temperature is reached to an extreme point, then it is trying to shut down the system.

Port sensor is either attached with hardware management system or it is operated via parent hardware. In this case, in the config file, the parent system specifies the type of port sensing. System also defines the default read interval, write interval. Sync interval used for the synchronizing with power unit, temperature unit and how many numbers of those units are used in the system. The information about the inability of the PWM or tacho fan is known from this sync interval. If breathing mode is enabled, then the fan pumps air as like breathing. In this way, different configurations have been done by the parent hardware, and according to the parent hardware configuration, the hardware management system continues working properly.

6.4. Observation:

The target is that as much as number of fans are incorporated in the system and the fans are temperature dependent. The fans are PWM or tacho feedback controlled, that means the running speed of the fan is determined by the aforesaid techniques. In this system different types of fan are incorporated like two wired, three wired and four wired. When two wired fan is incorporated in the system, it is working as a breathing mode of the system and working properly. Similarly, a three wire fan is also incorporated in the system, it is controlled by the PWM signal and it has been worked properly. Again, a four wire fan is also attached to the system, this fan is also controlled by the PWM signal and the fan speed is measured by the tacho signal.

```
void updateTacho()
{
  if(millis() - last_tacho_read_millis >= TACHO_UPDATE_INTERVAL)
  {
    detachInterrupt(digitalPinToInterrupt(FAN_1_TACH_PIN));
    detachInterrupt(digitalPinToInterrupt(FAN_2_TACH_PIN));

    fan_speed_buff_array[0] = rpm_counter_1 * ((60.0/2) * (1000.0/TACHO_UPDATE_INTERVAL)
    fan_speed_buff_array[1] = rpm_counter_2 * ((60.0/2) * (1000.0/TACHO_UPDATE_INTERVAL)

    rpm_counter_1 = 0;
    rpm_counter_2 = 0;

    last_tacho_read_millis = millis();

    attachInterrupt(digitalPinToInterrupt(FAN_1_TACH_PIN), fan1_rpm_counter, FALLING);
    attachInterrupt(digitalPinToInterrupt(FAN_2_TACH_PIN), fan2_rpm_counter, FALLING);
  }

}
```

Figure 36: Counting is done by interrupt handler

Similarly, the temperature sensor is attached to this hardware management platform which senses the ambient temperature as well as read the system temperature and it works perfectly. According to the cooling management system, this temperature is worked properly.

Suppose a power supply is off condition or a fault occurs in the supply, the secondary power supply is on automatically and the operation is very fast. For this fast operation a current spike is generated in the system which is very troublesome for the system. This spike is suppressed by a properly designed filter, the filter is designed by properly choosing capacitor and Zener diode and it is also working properly.

```
if(temperature_buff_array[0]>=alert_temp || temperature_buff_array[1]>=alert_temp)
{
  emergency_handler();
}

uint16_t fan_1_pwm_val = map(temperature_buff_array[0],0,byte(alert_temp*0.7),0,255);
uint16_t fan_2_pwm_val = map(temperature_buff_array[1],0,byte(alert_temp*0.7),0,255);

(fan_1_pwm_val<255)? ledcWrite(FAN_1_CH, fan_1_pwm_val) : ledcWrite(FAN_1_CH, 255);
(fan_2_pwm_val<255)? ledcWrite(FAN_2_CH, fan_2_pwm_val) : ledcWrite(FAN_2_CH, 255);
```

Figure 37: Fan speed controlling is done by 8-bit PWM signal

According to the configuration file (JSON), the developed hardware management system is communicated with parent hardware properly. JSON is a free and open-source file format and data exchange format that employs text that can be read by humans to store and send data objects made up of arrays and attribute-value pairs. It is a widely used data format for electronic data exchange, notably between servers and online applications.

Here different port sensors are assigned for the switch, which are used as a physical port sensor and it reads the input and shown as the output. If the input has been taken from the parent hardware, the port is activated automatically and works properly without overheating.

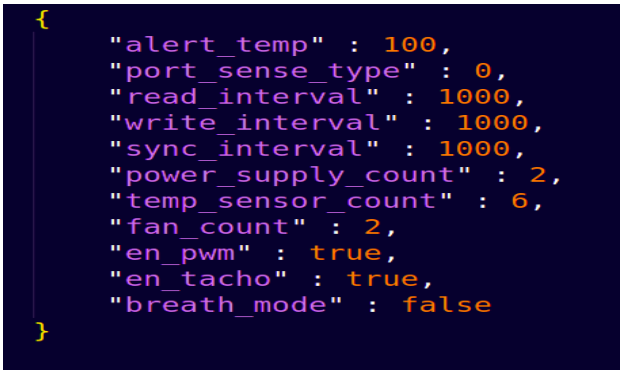

Figure 38: Configuration File(JSON)

FPGA TO ACCELERATE SDN

7.1. OVERVIEW

The data plane must offer methods for deploying new network protocols, header formats, and functions while keeping forwarding traffic as quickly as possible, even while SDN offers some hope for rapid prototyping and deployment. The most popular standard SDN southbound interface, OpenFlow, provides a vendor-neutral access to switching elements. Thus, OpenFlow SDN switches are the majority of SDN switches. The SDN data plane can be implemented in hardware or software, respectively. Some SDN hardware switches made by HP, NEC, and Arista suppliers make use of specialised ASIC switching chips. Although the ASIC method can provide excellent forwarding performance, new protocols and functions can scarcely be added. The flexibility can be provided by programmable hardware switches [27, 28] but at the expense of a significant number of hardware resources, such as pricy TCAMs for flow entries. The most adaptable switches for supporting new network services, protocols, and functions are SDN software switches [29-32]. Software SDN switches provide enough of memory resources for adapting flow rules, in contrast to the TCAMs in hardware SDN switches [33]. They have thus been widely used as first-hop virtual switches in data centres and are the first choices for SDN researchers working in labs. However, the strict performance and line-speed security requirements of the majority of current networks are difficult for a completely software-based strategy to meet. We attempt to utilise the advantages of FPGA (Field Programmable Gate Array) in packet processing as its capacity and processing power continue to increase. It is interesting that Microsoft has developed an FPGA fabric that is connected to each server in order to speed up extensive data centre services with specialised function logic [34]. FPGA is a great option for SDN software switches because of its high performance and adaptable reconfigurability. To enable the offloading of time-consuming software functional modules and the installation of real-time security modules in SDN switch processing path, we offer FAS (FPGA-Accelerated SDN Software Switch). While maintaining the adaptability of software switches, the approach may effectively address the performance gap. By implementing bump-in-the-wire security modules in FPGA, it is also possible to improve the security feature of software switches. The following is a summary of the paper's contributions.

(1) We do a thorough analysis of the present implementation models of SDN switches in academia and industry and group them into various categories.

(2) We do a thorough analysis of the present implementation models of SDN switches in academia and industry and group them into various categories.

(3) We create a FAS mechanism to offload operations in the SDN software switch's forwarding path, such as packet buffer management, packet parsing, and some packet action executions, to FPGA hardware.
(4) On Zedboard, an FPGA-based network processing platform, we implement the FAS prototype and compare its performance to that of the original SDN software switches on a common multicore platform.

7.2. Background and Related Work

The evolution of the OpenFlow protocol, which poses a challenge to the design of SDN switches, is briefly discussed in the opening paragraphs of this section.

7.2.1. Growth of OpenFlow: Among the SDN southbound interfaces, OpenFlow is the first and most widely used standard as specified by the Open Network Foundation (ONF). Many commercial switches, including HP, NEC, Arista, and Pica 8, as well as the list of supporters, are enthusiastic. The OpenFlow standard has been upgraded to version 1.5 in 2015 and has gotten more complex from the first version (v1.0) issued by ONF in December 2009 [3]. OpenFlow's fundamental idea has not altered, despite the addition of numerous features. Instead of handling and forwarding traffic based on individual packets, OpenFlow switches handle traffic based on flows. A single flow table containing flow rules that may match packets on 12 header fields, including MAC addresses, IP addresses, and TCP/UDP port numbers, makes up the data plane abstraction in the initial iteration. The OpenFlow switch has a pipeline of flow tables in version 1.5, and each flow rule has match fields (41 fields in the packet header), instructions (such as drop, flood, forward, or send the packet to the controller), a set of counters (to track the number of bytes and packets), a priority (to distinguish between rules with overlapping patterns), timeouts (flow rules' expiration times), cookies (opaque data values selected by the controller), and flags (to alter the way flow entries are managed). An OpenFlow switch locates the highest-priority matching rule when it receives a packet, executes the related operations, and increases the counters. As the OpenFlow specification has developed, Open-Flow has gained more features and capabilities. As a result, OpenFlow processing is becoming more intricate and is expanding rapidly with no sign of slowing. It presents difficulties for the use of SDN switches.

7.2.2. SDN switch implementation models: An SDN switch typically has four parts, as specified by the OpenFlow specification: an OpenFlow Channel (OFCh), an OpenFlow

Forwarding pipeline (OFFw), and a physical port (Port). Moreover, Flow Cache (FCa), a widely utilised technology, is added to SDN switches to speed up the multiple-tuple categorization process in OFFw [35]. The following are descriptions of the four modules.
Port. It serves as the interface between the switch and the network and is in charge of receiving and delivering packets.

FCa. It serves as OFFw's fast forwarding path and caches entries that have recently matched in OFFw. Bypassing OFFw, FCa can increase the forwarding rate since network traffic is sufficiently local to provide a high cache hit rate with a modest cache size.

OFFw. Each item in the flow table that it keeps lists a set of packet fields to match as well as the appropriate action to take (for example, forwarding, dropping, and altering header).

OFCh. The packet is transmitted to the controller through OFCh if a switch cannot discover a match in OFFw. The controller sends OpenFlow messages to the necessary switches after determining how to route the new flow. Following the resolution of those messages, OFCh creates flow rules and instals them into the flow table. Additionally, OFCh is in charge of transmitting states from the controller to the switch.

Switches for SDN have been put into use on many systems (e.g., general CPUs, fixed function switch ASICs, reconfigurable hardware, NPUs, and FPGAs). Hardware-based switches and software-based switches are the two categories into which the platforms can be separated. The various SDN switch implementation models are depicted in Figure 1. Hardware components like ASICs and FPGAs are used to implement FCa or OFFw in hardware-based switches. For instance, Stanford's NetFPGA platform is used by Naous et al. to create an OpenFlow switch [36]. Some businesses, like Pica 8, provide generic OpenFlow switches built on ASIC switch chips. The Flow Cache hardware offloading paradigm serves as the foundation for these hardware switches. In the OpenFlow pipeline, RMT [27] and FM6000 [28] provide reconfigurable match tables that adhere to the hardware forwarding concept. Although the SDN hardware switches can offer adequate forwarding capacity, they are expensive and rigid. The need to reconsider software switching has arisen as a result of recent improvements in multicore processing capability. The SDN software switches are drawing more attention because of their great flexibility and quick development cycle. Commodity off-the-shelf (COST) PCs or servers with multiple Network Interface Cards (NICs) and multicore processors

are frequently used by software switches. These devices run general-purpose operating systems (such as Linux). The general-purpose OS offers a favourable setting for innovation and research. The common DN software switches referred to the software forwarding paradigm are OpenFlow reference switch (kept by ONF) [29], OFSoft Switch (maintained by CPqD) [30], and OpenFlow Click (maintained by Stanford) [31]. Open vSwitch [38] (specifically, OVS, maintained by VMware) is an SDN software switch that is a part of the user space-kernel collaboration concept. The specific details of the aforementioned SDN switching platforms.

7.3. Problem Description and Analysis

7.3.1. SDN Software Switches with OpenFlow Forwarding:

We concentrate on the user space-and-kernel cooperation (UKC) paradigm as depicted in Figure 39 because the OVS is the most sophisticated and extensively used SDN software switch. The UKC model's forwarding mechanism is depicted in Figure 39, and its chronology is shown in Figure 40. To keep things simple, we first go over the process using Network Interface Cards with a single queue in a multicore architecture (NICs). The case involving numerous queue NICs will be covered. Without loss of generality, we assume that NET RX and NET TX of hardware interrupts in NIC i are both served by a fixed CPU core i. Cores i and j are denoted as Ci and Cj. If i equals j, packets are sent and received through the same NIC and processed by the same CPU core. When a packet arrives at NIC i, this packet is attached to a descriptor in the NIC i's receiving (namely, RX) queue. The descriptor indicates the memory locations to store the incoming packets via Direct Memory Access (DMA) transfer. There are two data structures in the network stack of Linux Kernel. Data buff of 2 KB size is to hold the packet itself. The other data structure is sk buff, which carries packet information metadata (e.g., pointer of packet data, MAC header, IP header, and packet states) used by TCP/IP protocol stack. The size of sk buff is about 250 bytes. The sk buff has a pointer to data buff, and it makes up a Skb with the data buff.

Table IV: SDN switching platforms.

Names	Supporting OF version	Languages/hardware type	Application scenarios	Reference model
NetFPGA-based OpenFlow switch	1.0	FPGA	Research	Figure 1(c)
NP-based OpenFlow switch	1.0	Network processor	Research, enterprise	Figure 1(c)
Commodity OpenFlow switch	1.0	ASIC	Research, enterprise, data center	Figure 1(c)
RMT	All	ASIC	Research, enterprise	Figure 1(d)
FM6000	All	ASIC	Research, enterprise	Figure 1(d)
OpenFlow reference switch	All	C	Research	Figure 1(a)
OFSoftSwitch	After 1.3	C	Research	Figure 1(a)
OpenFlow Click	All	C++	Research	Figure 1(a)
Open vSwitch	All	C & kernel C	Research, virtualized data center	Figure 1(b)

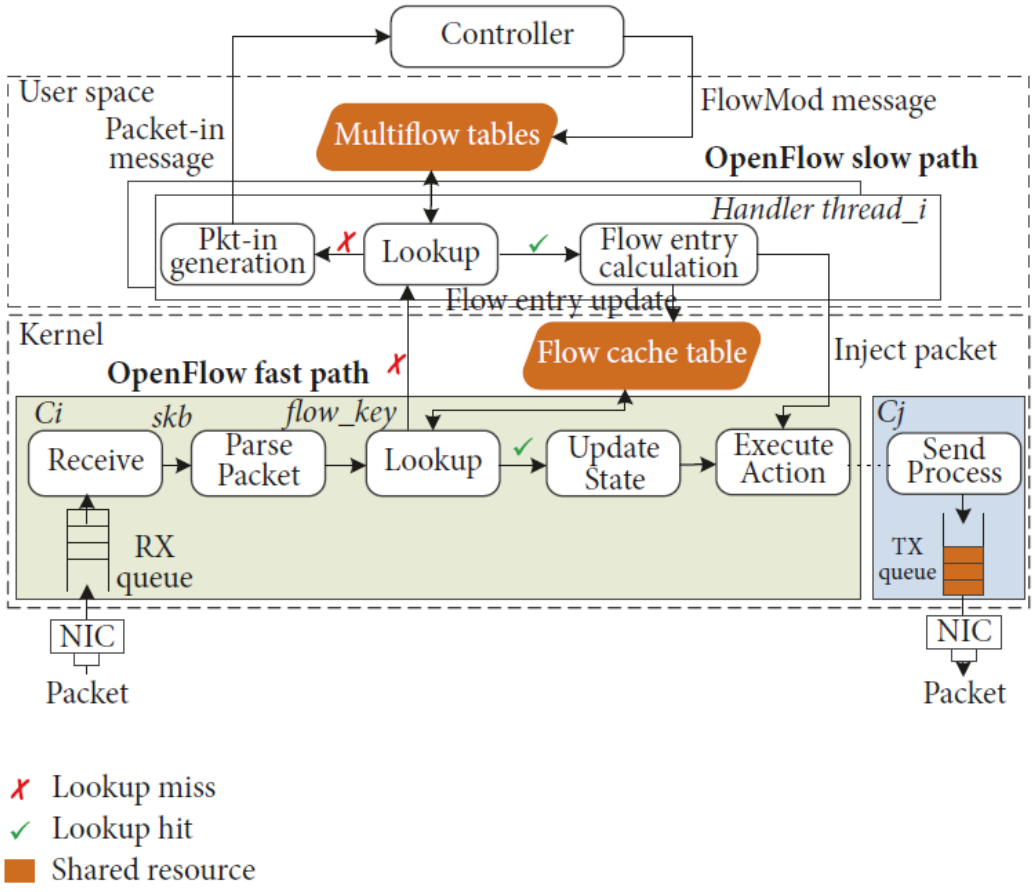

Figure 39: Forwarding process of UKC model

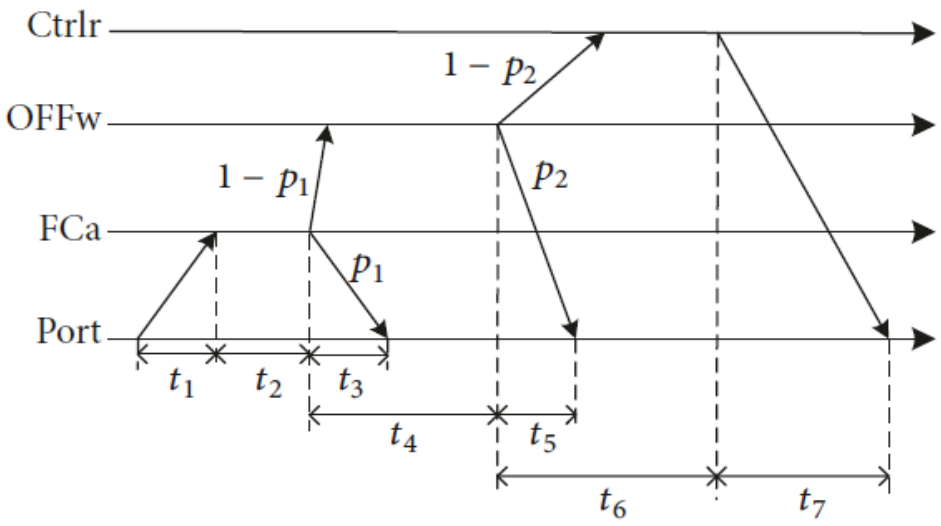

Figure 40: Timeline of UKC forwarding

After hardware interrupt of packet incoming is served, the Software Interrupts (SoftIRQs) are scheduled afterwards to accomplish the subsequent packet processing. The SoftIRQs are also bound to the specific processors. In OVS, the handler function of SoftIRQ of Ci parses the packet by extracting all related match fields from Skb and stores them to flow key. The flow key is used to look up the Flow Cache, which is shared among all processors in the kernel. If the Flow Cache contains the flow key value, the Flow Cache will return instructions for processing the packet. Then it updates corresponding states and executes actions to the packet. If the packet is to be delivered by NIC j, it will be placed in the sending (namely, TX) queue of the NIC j. The sending process will be called in the function of NET TX SoftIRQ of Cj. If the Flow Cache lookup fails, the packet will be transmitted to Linux's user space using the kernel and user space communication method, such as netlink. The OpenFlow software switch configuration will result in the creation of several handler threads in user space. These threads are in charge of handling kernel-generated mismatched packets. The handler threads search the multi flow tables that are shared. The handler threads will compute new Flow Cache entries and update them in the Flow Cache if the packets match the tables. The handler will also inject the packet once more into the kernel so that the appropriate actions can be taken with it. A Packet-in message will be created and delivered to the controller if this is not the case. The controller reacts by sending a Flow Mod message that instructs the multi flow tables to install the appropriate flow rule for the packet.

We assess the processing overhead in SDN software switches based on the examination of the forwarding process. To specify the procedure's time windows, we make a few notes. t1 refers to the amount of time that has passed between the packet's arrival at the NIC and the cache lookup. The lookup time for the flow cache is t2. t3 is the length of time between the beginning of matching Flow Cache and the conclusion of packet transmission. The cache hit rate is designated as p1 in the lookup. t4 is the time taken to search through user space's multi flow tables from beginning to end. T5 is the length of time from the start of matching multiple flow tables to the conclusion of packet transmission. We refer to the multi flow tables' hit rate as p2. From the moment the controller is triggered until the moment the rule is installed, there is a six-times delay (t6). Finally, t7 is the duration used to deliver the packet. In light of the aforementioned notations, the evaluated total processing time T is expressed as follows:

$$T = t1 + t2 + p1t3 + (1 - p1)\, p2\, (t4 + t5) + (1 - p1)\, (1 - p2)\, (t6 + t7) \quad (1)$$

The Flow Cache can achieve a high hit rate because of the high proximity of network traffic and the proactive flow rule configuration. The overall cache hit rate of Open vSwitch was 97.7% (i.e., the value of p1) in a genuine commercial multitenant data centre, as confirmed by Pfaff et al. [43]. Consequently, it is possible to roughly compute the entire process time T of a packet in SDN software switches as follows:

$$T = t1 + t2 + t3. \quad (2)$$

The OVS's fast path is made up of these three periods. The Receive, Parse Packet, Lookup, Update State, Execute Action, and Send Process functional modules make up the OpenFlow fast path, which is shown in Figure 39. We use Open vSwitch as a software switch on a common PC to analyse the overhead for each functional module. To generate the input traffic that is being sent via the PC's two 1 GbE NICs, we use the Iperf [38] utility. Next, we count the number of CPU cycles used by each function in Open vSwitch's fast path using O profile [39]. The experimental findings after classifying all functions into the aforementioned six functional groups. As we can see, the main bottleneck in the fast approach is the Lookup module. Up to 44% of the overall processing time is used by the other modules.

7.4. Bottlenecks in SDN Software Switch. Packet I/O, software forwarding, and Open-Flow classification are the three key bottlenecks in SDN software switches. IO on packets. Packet I/O is a crucial stage in software packet forwarding, as opposed to CPU-intensive processes. CPU cycles are heavily used when sending and receiving packets. According to [40], the two main overheads in packet I/O—buffer allocation and release—represent up to 54% of the overall overhead. To speed up packet I/O, researchers have suggested a number of

optimization strategies, including the Skb recycling queue, memory mapping, batch processing, affinity, and software prefetching. Data Plane Development Kit (DPDK), a high-performance packet processing architecture for x86 platforms, is created by Intel by merging some of the aforementioned methodologies [41]. However, the DPDK framework requires a certain CPU and NIC, which are not common. Netmap is a brand-new framework created by Rizzo for quick packet I/O in general-purpose operating systems [42]. Netmap, however, runs in the OS's user space by default. When it is used with kernel-based SDN software switches, the performance suffers.

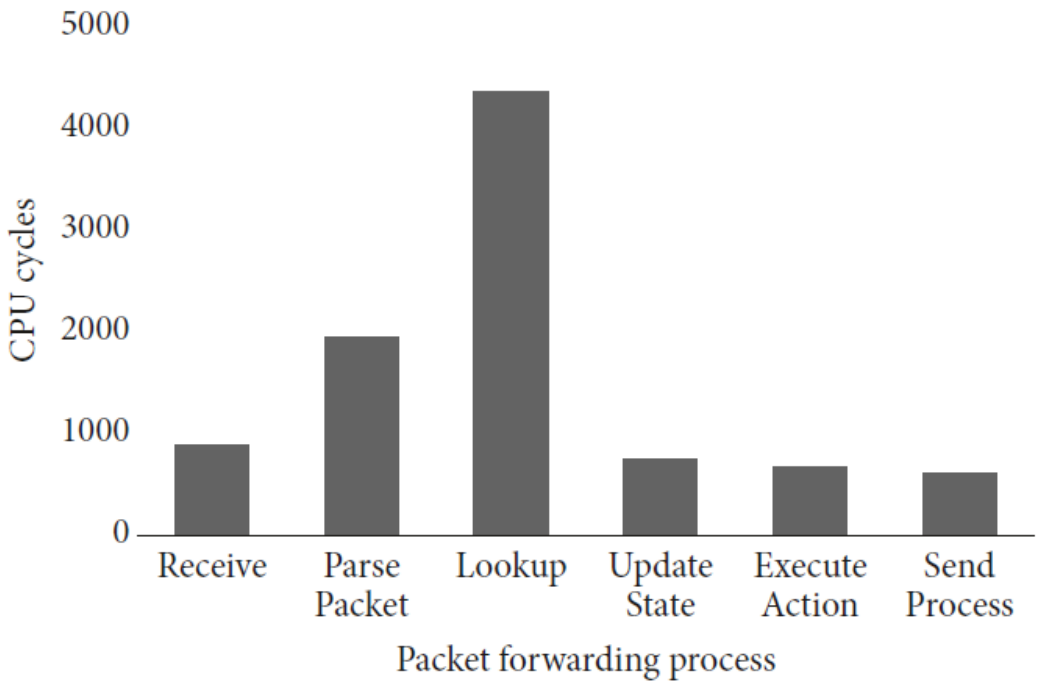

Figure 41: Packet forwarding process overhead breakdown. The processing for packets in Execute Action module only contains the forwarding operation.

Software Forwarding. Contention for shared resources, including as caches, NIC queues, and flow tables, while several packet-forwarding threads are active at once is a serious issue [43]. It is suggested to use NIC-based core affinity to take use of the multicore architecture's parallel packet processing capacity in order to increase the processing performance of SMP Linux. The defined core is responsible for handling both software and hardware interrupts of the NIC by maintaining the affinity relation between the two. It causes fewer cache misses as a result, increasing packet processing execution efficiency. However, the NIC-based core affinity is ineffective for packet forwarding. Han et al. propose queue-based core affinity [44] because

the receiving and sending of packets are typically handled by distinct cores, which results in the issues of mutex exclusion and cache coherence. Because each receiving and sending queue in a NIC corresponds to a specific CPU core, there is no longer any lock contention or cache bouncing because of shared data structures. Although the NICs can hardly be scaled, they must handle multiple queues and Receive Side Scaling (RSS). Classification of OpenFlow. The OpenFlow categorization in the kernel is made up of the tasks of packet processing, Flow Cache lookup, counter updating, and action execution, as shown in Figure 39. On general-purpose computers, packet categorization takes a long time and gets worse over time. OpenFlow 1.5 specifies that during packet classification, 41 packet fields shall be parsed, extracted, and searched up. Because of this, OpenFlow categorization is very difficult and has poor performance. In order to speed up OpenFlow categorization, Putnam et al. created a three-layer cache architecture (microflow cache, megaflow cache, and OpenFlow pipeline) [40]. Additionally, some packet processing operations, such as packet encapsulation and decapsulation, need expensive software. In addition, integrating complicated security features will significantly reduce the performance of SDN software switches. The majority of processing operations required for security tasks are CPU-intensive and stateful. For SDN software switches, the best way to increase security capacity without sacrificing performance should be looked into.

7.5. FAS Mechanism

FAS's structural layout. We use programmable hardware, FPGA, to accelerate computation in order to address the aforementioned bottlenecks of the current SDN software switches. The OpenFlow software fast path's time-consuming functional modules are intended to be offloaded to FPGA using the FAS mechanism. Three components make up the FAS mechanism in FPGA, as shown in Figure 4. The Linux kernel's self-described buffer management module is employed to offload various time-consuming packet receiving and sending operations. The purpose of the Metadata Generate module is to offload packet parsing. After getting a packet and its actions from the Metadata Resolve module, Execute Action is used to carry out some actions.

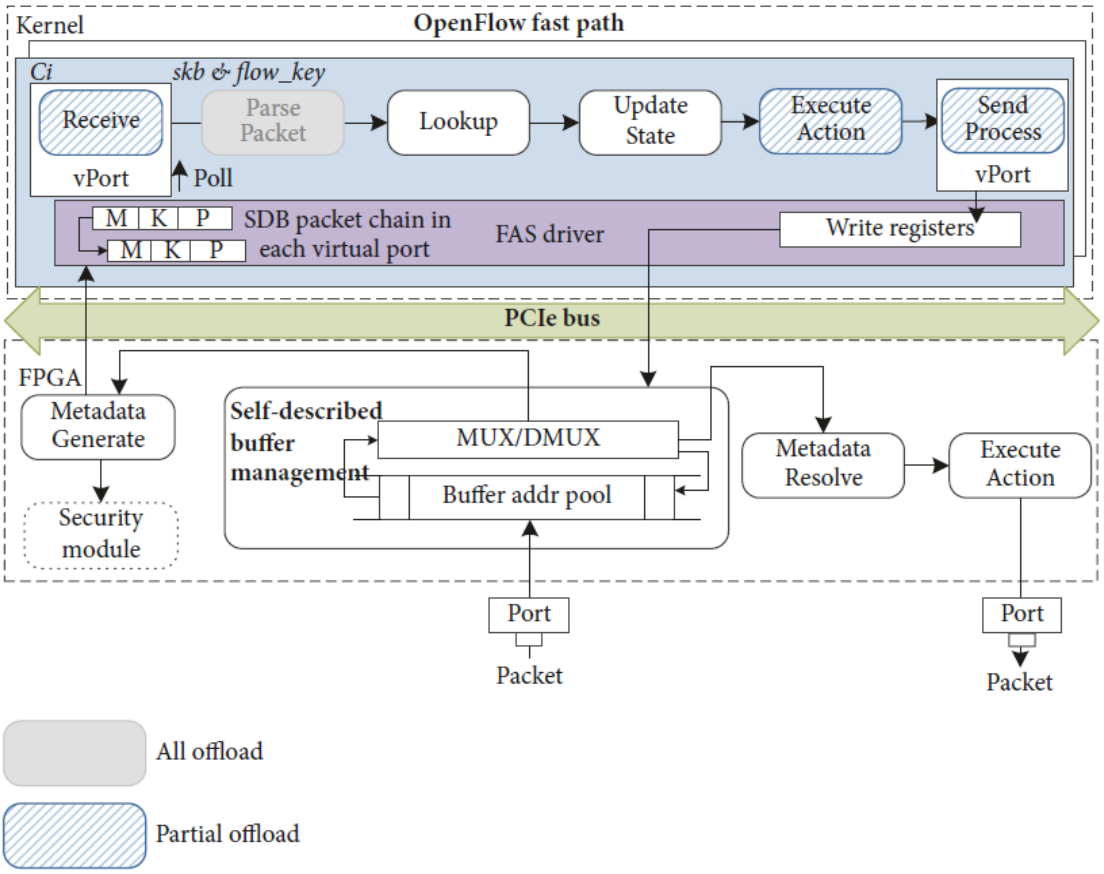

Figure 42: The framework of FAS mechanism.

When software sends metadata for packet processing, the Metadata Resolve module receives it, resolves it, and alerts the Execute Action module. Offloading of packet buffer management. During packet I/O, managing Skb for each packet is a crucial task. It includes the processes of converting the raw packet to the Skb, initialising the Skb, and allocating and releasing the Skb. The process of packet I/O takes up the majority of CPU cycles. Our research team has previously suggested Self-Described Buffer (SDB) management in hardware to reduce the expense of packet buffer management in software [45]. Packet and its information from the initial segregated data structure Skb are combined into a subsequent stored packet buffer in SDB (we call it the SDB packet buffer). During the initial phase, the software pre-allocates fixed size space in main memory for SDB packet buffers. In order to support dynamic allocation and recycling of circular addresses for SDB packet buffers during packet I/O, this enables the SDB hardware. Software's overhead for managing packet buffers is therefore removed. SDB is used by FAS to reduce the packet I/O bottleneck. Offload for packet parsing. By parsing Skb, the OpenFlow fast path generates flow keys by extracting matched fields.

Additionally, as seen in Figure 41, the parsing process is the second most expensive component. Thus, we delegate the task of packet parsing to hardware and store the parsing outcome in the packet's metadata. As seen in Figure 43, the parsing procedure in OpenFlow forwarding is rather simple. It is simple to construct in hardware using FPGA. Offloading of action execution. According to hardware capabilities, the FPGA's Execute Action module performs partial operations for packets. The forwarding action to the appropriate port is the basic procedure. During the transmitting phase, the hardware will read the packet with metadata through DMA and resolve the packet address recorded in the FPGA registers. The software specifies the information that contains the action parameters. The packet will be enqueued to the specified queue of the relevant port in the FPGA, for instance, if the queue action in the metadata is resolved by the module of Metadata Resolve. Complex and time-consuming software packet processes, like packet field rewriting and encapsulation/decapsulation, can be removed with the offload of Execution Action.

Table V: Comparisons of acceleration mechanisms for SDN software switches

Names	Method	Acceleration object	Complexity	Network virtualization	HW state manage. cost
DPDK-based OF switching	Intel CPU/NIC	Packet I/O	Low	Support	No
Netmap-based OF switching	Netmap lib	Packet I/O	Low	Support	No
Flow Director	Driver modification	OF flow cache	Medium	Not support	High
SSDP	Commodity switching chip with TCAM	OF forwarding path	High	Not support	High
FAS	FPGA	Time-consuming functional modules in OF fast path	Medium	Support	Low

Interface for Security Function Extension. We have suggested a clear module interface to facilitate the introduction of hardware security functionalities. The control command is encoded into the information that comes before the packet data on the downstream and upstream interfaces of security modules, which are FIFO-like interfaces. In the event that the module interface definition is accurate, the security module can be quickly integrated into the FPGA processing pipeline.

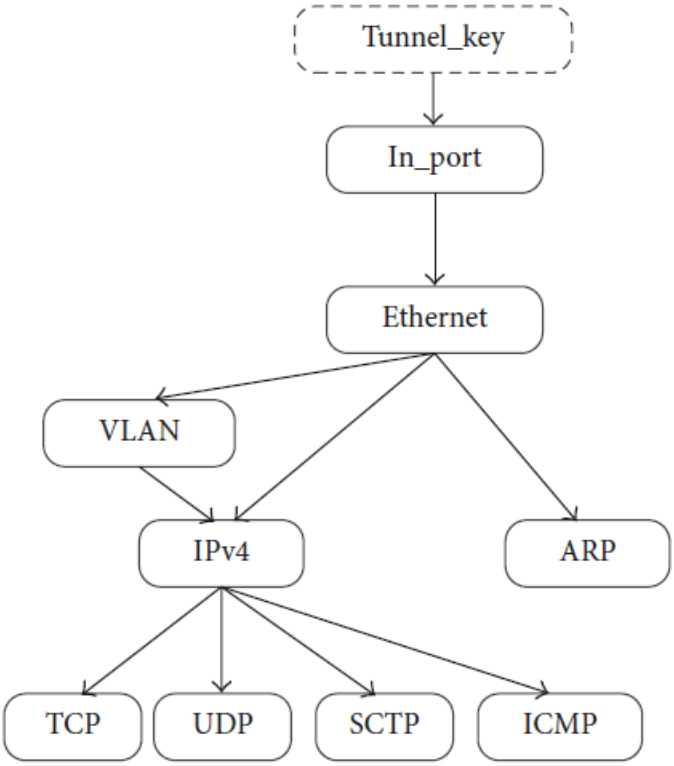

Figure 43: A parse graph example for OpenFlow

Comparisons between FAS and Existing Mechanisms.

For the purpose of enhancing OpenFlow software switching performance, numerous approaches have been proposed. In Table V, we present contemporary mechanisms and highlight how they differ from FAS. High-performance I/O frameworks for OpenFlow software switching are provided by DPDK and netmap. It should be noted that DPDK requires particular CPUs and NICs (such as the Intel 82599), whereas netmap can be used with any NIC [46]. The packet classification hardware on the NIC is suggested by Flow Director to be used as an OpenFlow fast forwarding path [47]. However, it is relatively expensive to maintain the Flow Cache entries in the NIC driver. SSDP suggests utilising TCAM and other common switching chips to improve slow software forwarding paths [48]. Microflows in the switch chip and microflows in the CPU make up the two data planes that make up the common OpenFlow forwarding path in SSDP. It is impossible to maintain state consistency between the two data planes.

7.6. Proposed Implementation

This section describes a design and preliminary implementation of FAS on SoC, and the Zedboard, which is an FPGA based network processing platform with multicore CPU. It can be implemented in ONetSwitch20. In this section 4 port FPGA based SDN Switch is being discussed. It is been developed with the help of FPGA(Zedboard) FPGA Mezzanine connector(FMC).

7.7. Overview Of Proposed Implementation

A fully programmable Open Network Innovation Platform is ONetSwitch20. ONetSwitch20 uses the Zedboard as the motherboard and the Xilinx XC7Z020 Zynq AP SoC as the processor. Additionally, it extends four Gigabit Ethernet connections via the MeshSr FMC4GE daughter board. As ONetSwitch20 customises the common platform for mother card to a network system node with potent performance on network processes, it also enhances reference designs for network interfaces and network applications.

7.8. Key Features

Table VI: ONetSwitch20 Key Features

General	
Core Silicon	XC7Z020-1CLG484
Power Supply	DC 12V
Programming Source	QSPI Flash / JTAG / SD Card
Processing System	
Processor	Dual ARM Cortex-A9@800MHz
Cache	L1: 32KB Instruction + 32KB Data per processor; L2: 512KB; OCM: 256KB
DRAM	DDR3 512MBytes
Flash	Quad SPI Flash 256Mb
DMA	8 channel (4 for Programmable Logic)
Ethernet Port	1x 1000BASE-T
Peripheral	USB / USB-UART / USB JTAG / SD Card
Programmable Logic	
Programmable Logic Equivalent	85K Logic Cells, Logic Cells, Artix-7FPGA, Approximate ASIC Gates 1.3M
PS to PL Interconnect	AMBA AXI4 interconnect, maximum 100Gbps
Ethernet Port	4x GE RJ45 10/100/1000M Ethernet, mounted at FMC card
Peripheral	I2S ADC / HDMI / VGA / OLED
User I/O	User LEDs/Pushbuttons/DIP Switch

7.9. Board Block Diagram

7.9.1. Board-to-Board Connection

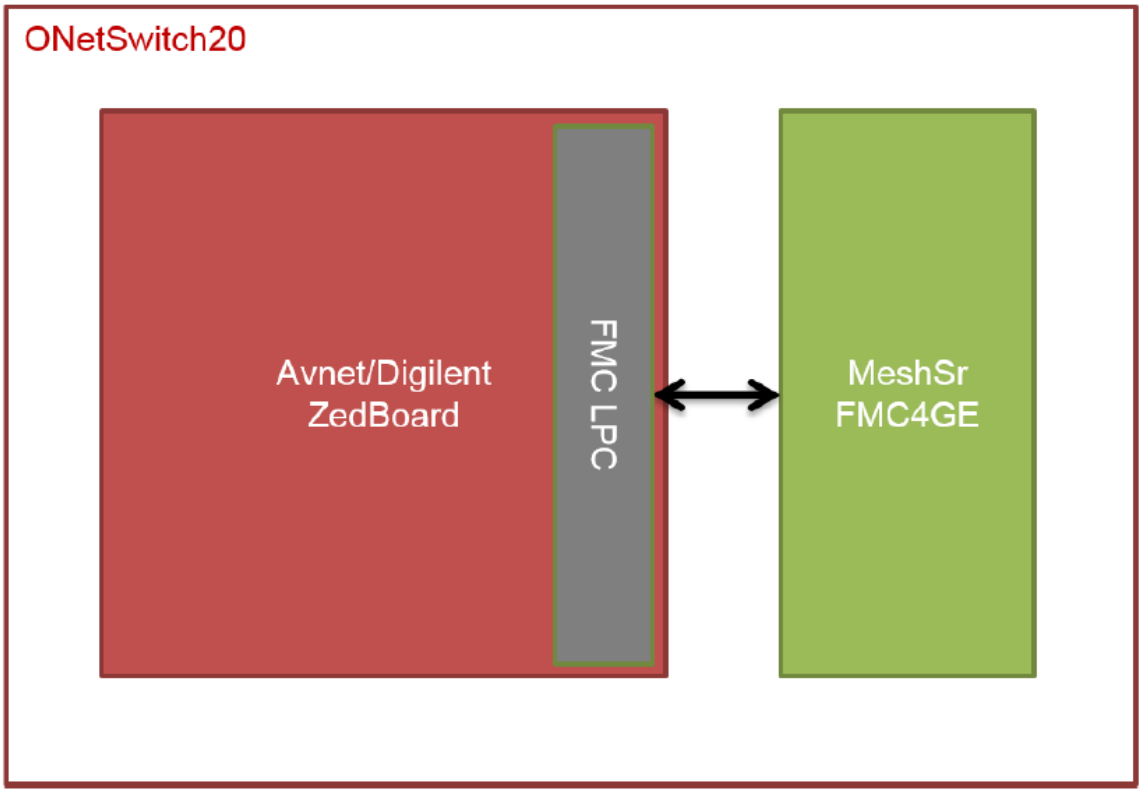

Figure 44: ONetSwitch20 Board-to-Board Block Diagram

7.9.2. ONetSwitch20 Block Diagram

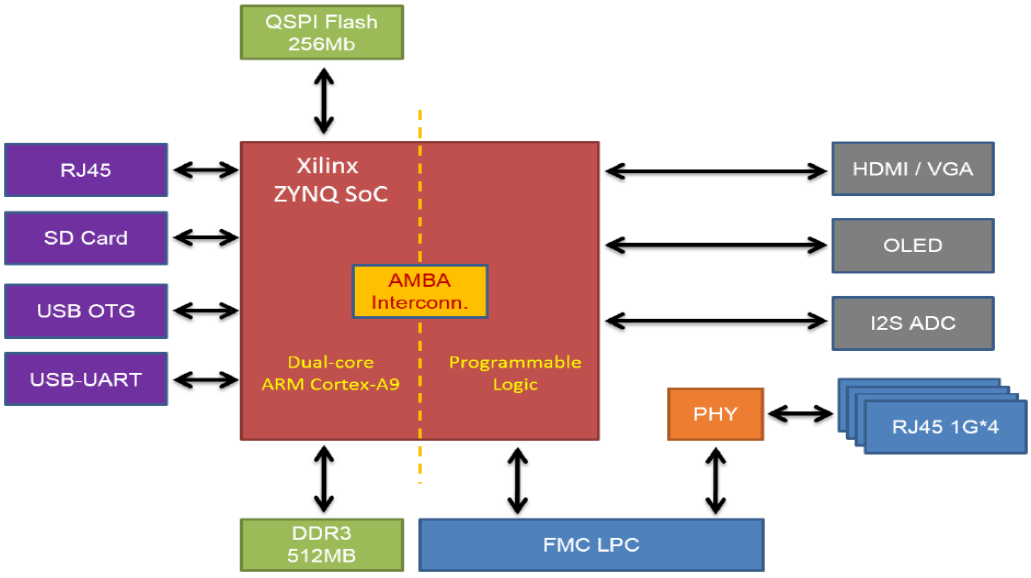

Figure 45: ONetSwitch20 Feature Block Diagram

7.10. Bank Description

7.10.1. Main Board Description

With the exception of 57 pins located at Banks 34 and 35, which are occupied by FMC mezzanine cards, ONetSwitch20 maintains the functionality of Zedboard. For further details on BANK and Pin Assignments, please consult the Zedboard Hardware User Guide.

Table VII: Zynq Bank Assignments

Bank	I/O Power supply	Feature
500(MIO 0)	3.3V	QSPI, PMOD
501(MIO 1)	1.8V	USB-UART, Ethernet RJ45, USB-OTG, SD
502(DDR)	1.5V	DDR3
13(High Range)	3.3V	OLED, PMODs, I2S ADC
33(High Range)	3.3V	HDMI, VGA
34(High Range)	2.5V(Adjustable)	FMC, Push button
35(High Range)	2.5V(Adjustable)	FMC, User switch

7.11. X4GE Ethernet FMC

The XC7Z020 Zynq AP SoC on ZedBoard can be connected using the FMC Connectivity mezzanine card's four Ethernet ports, which provide 10/100/1000BASE-T standard PHY functionalities. ONetSwitch20uses the Broadcom BCM5464SRPHY (RGMII to copper) for x4 Ethernet communications. The FMC card with LPC connector interfaces the BCM5464SR to PL of Zynq APSoC. The PHY connects to MIO Bank34(MDIO, port #0, port#1) and Bank35(RST, port #3, port#4).

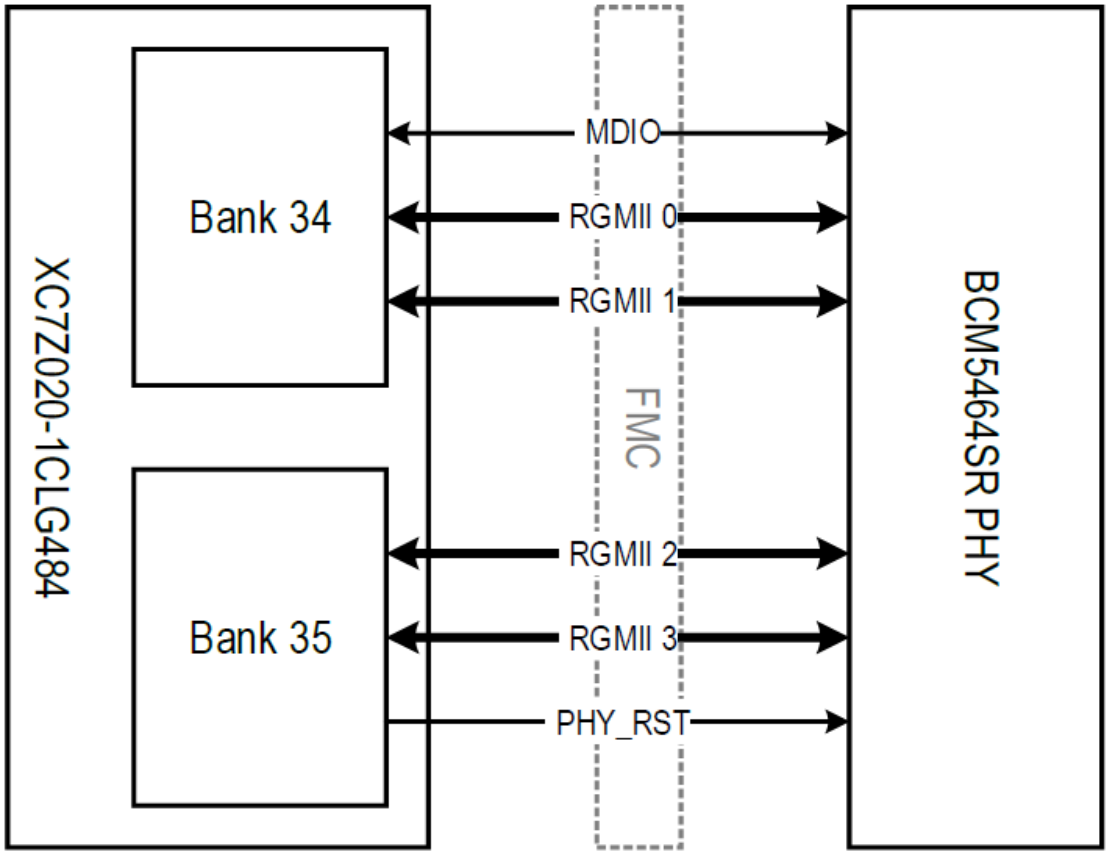

Figure 46: Quad GE PHY connections

7.11.1. Feature Descriptions

7.11.1.1 Main Board Features

The ONetSwitch20 is populated with the Xilinx Zynq XC7Z020-1CLG484AP SoC. The XC7Z020 AP SoC consists of an integrated processing system (PS) a programmable logic (PL), on a single die. Information for the Main Board specification can be found at the ZedBoard Hardware User Guide.

Table VIII: Main Board Features

Feature	Connection	Component part number	Detailed information from ZedBoard Hardware User Guide
DDR3	PS	MT41K128M16JT-125	Power supply, Connections,延迟
QSPI	PS	S25FL256SAGMFI00	Mode setting, Pin assignment
SD	PS	2041021-1	Connections, Pin assignment
USB-OTG	PS	TUSB1210|1981584-1	Reference Clock, Pin assignment
USB-UART	PS	CY7C64225|1981584-1	Connections, Pin assignment
USB-JTAG	PS	Digilent JTAG SMT1	Connections
Ethernet	PS	88E1518|1840750-7	Mode setting, Connections, Pin assignment
HDMI	PL	ADV7511KSTZ	Mode setting, Pin assignment 时序
VGA	PL	4-1734682-2	Pin assignment
I2S ADC	PL	ADAU1761BCPZ	Mode setting, Connections, Pin assignment
OLED	PL	UG-2832HSWEG04	Connections
Push button	PS/PL	N/A	Pin assignment
User switch	PL	N/A	Pin assignment
LED	PS/PL	N/A	Pin assignment

7.11.1.2. X4 GE Ethernet Ports

One Broadcom BCM5464SR Ethernet transceiver (PHY) is included for use with a Belfuse 0826-1X4T-23-F RJ 45 connector with built-in magnetics to connect to network connections. Four triple speed 10/100/1000BASE T Ethernet transceivers with RGMII interfaces make up the BCM5464SR transceiver. A MDIO bus could be used to programme the PHY to function properly. Please see the BCM5464SR datasheet for additional information.

ü Mode setting: INTF_SEL[3:0]=’0001’,RGMII to Copper, 2.5V OVDD

ü PHY address setting: PHYA_REV=0,PHYA[4:0]=’00000’,start addressis'00', increase

ü MDIO setting: MDIO_SEL[1:0]=’00’, MDIO/MDC[1] access all 4

ü Speed Select: ANEN=1,F1000=1,SPD0=0,Auto-negotiate advertise: 10/100/1000BASE-T

For more detailed information about PL HDL coding, please refer to Xilinx LogiCORE IP AXI Ethernet related documents on the Xilinx website.

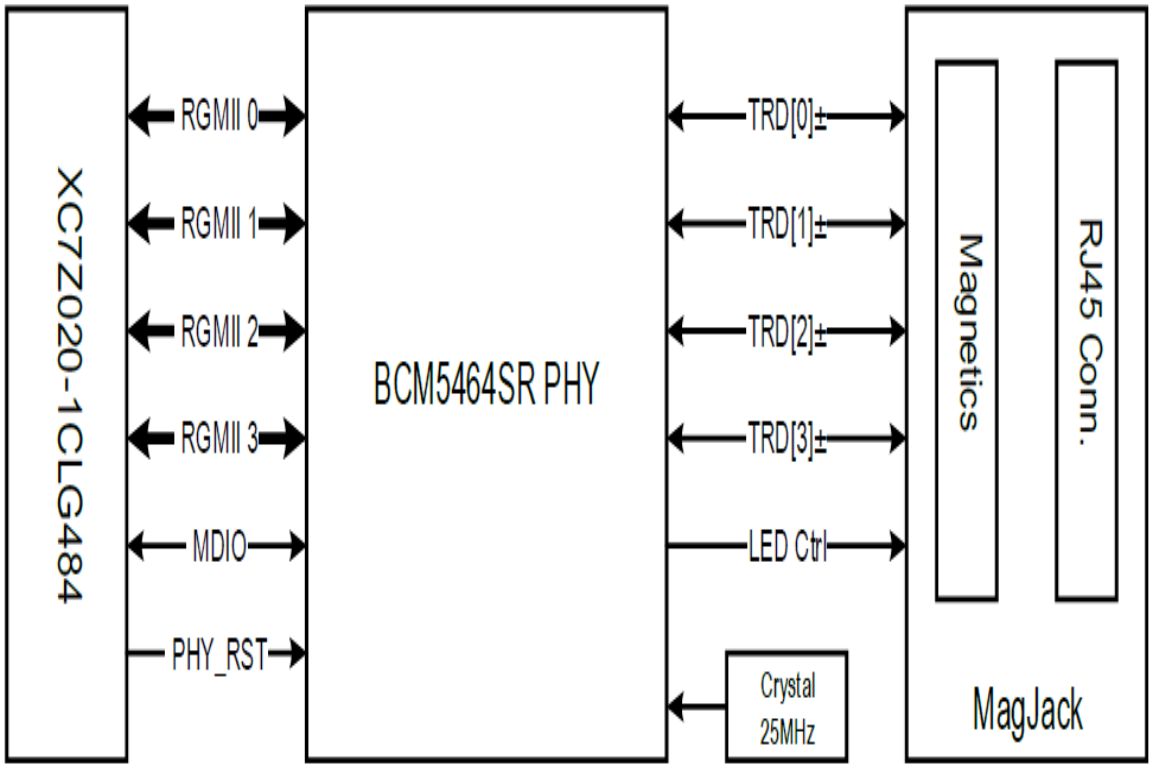

Figure 47: X4 Ethernet Block Diagram

The connections from the XC7Z020AP SoC to BCM5464SRare listed in Table 4. The FMC4GEcard output a 125MHz reference clock to main board (L18/L19). The trace lengths must be considered for RGMII layout.The main board trace lengths was informed at ZedBoard user guide, and the FMC card side are listed in Table 4.

Table IX: X4GE Ethernet Connections

Grp.	Pin	BANK	LOC	FMC	Net	Etch Len. Mils
RGMII0	rgmii_0_txc	34	J17	LA15_N	PHY_0_GTXCLK	1692.747
	rgmii_0_rxc	34	M19	LA00_P_CC	PHY_0_RXC	1716.171
	rgmii_0_rd[0]	34	J18	LA05_P	PHY_0_RXD_0	1687.121
	rgmii_0_rd[1]	34	N18	LA11_N	PHY_0_RXD_1	1708.124
	rgmii_0_rd[2]	34	P21	LA12_N	PHY_0_RXD_2	1758.829
	rgmii_0_rd[3]	34	P20	LA12_P	PHY_0_RXD_3	1685.59
	rgmii_0_rx_ctl	34	K18	LA05_N	PHY_0_RX_DV	1772.593
	rgmii_0_td[0]	34	J22	LA08_N	PHY_0_TXD_0	1702.916
	rgmii_0_td[1]	34	T17	LA07_N	PHY_0_TXD_1	1697.979
	rgmii_0_td[2]	34	N17	LA11_P	PHY_0_TXD_2	1733.734
	rgmii_0_td[3]	34	L21	LA06_P	PHY_0_TXD_3	1848.894
	rgmii_0_tx_ctl	34	J16	LA15_P	PHY_0_TX_EN	1690.039
RGMII1	rgmii_1_txc	34	R21	LA09_N	PHY_1_GTXCLK	2602.528
	rgmii_1_rxc	34	N19	LA01_P_CC	PHY_1_RXC	2431.953
	rgmii_1_rd[0]	34	T16	LA07_P	PHY_1_RXD_0	2541.806
	rgmii_1_rd[1]	34	M22	LA04_N	PHY_1_RXD_1	2432.294
	rgmii_1_rd[2]	34	P17	LA02_P	PHY_1_RXD_2	2547.459
	rgmii_1_rd[3]	34	L17	LA13_P	PHY_1_RXD_3	2451.657
	rgmii_1_rx_ctl	34	R20	LA09_P	PHY_1_RX_DV	2441.992
	rgmii_1_td[0]	34	N22	LA03_P	PHY_1_TXD_0	2459.521
	rgmii_1_td[1]	34	P18	LA02_N	PHY_1_TXD_1	2450.547
	rgmii_1_td[2]	34	M20	LA00_N_CC	PHY_1_TXD_2	2448.238
	rgmii_1_td[3]	34	T19	LA10_N	PHY_1_TXD_3	2527.951
	rgmii_1_tx_ctl	34	R19	LA10_P	PHY_1_TX_EN	2432.416
RGMII2	rgmii_2_txc	35	C22	LA25_N	PHY_2_GTXCLK	2663.201

	rgmii_2_rxc	35	B19	LA17_P_CC	PHY_2_RXC	2795.669
	rgmii_2_rd[0]	35	E15	LA23_P	PHY_2_RXD_0	2729.28
	rgmii_2_rd[1]	35	C20	LA18_N_CC	PHY_2_RXD_1	2782.084
	rgmii_2_rd[2]	35	G15	LA19_P	PHY_2_RXD_2	2706.063
	rgmii_2_rd[3]	35	E21	LA27_P	PHY_2_RXD_3	2724.659
	rgmii_2_rx_ctl	35	F18	LA26_P	PHY_2_RX_DV	2672.365
	rgmii_2_td[0]	35	G21	LA20_N	PHY_2_TXD_0	2722.673
	rgmii_2_td[1]	35	A17	LA28_N	PHY_2_TXD_1	2714.414
	rgmii_2_td[2]	35	A18	LA24_P	PHY_2_TXD_2	2722.488
	rgmii_2_td[3]	35	D21	LA27_N	PHY_2_TXD_3	2687.964
	rgmii_2_tx_ctl	35	E18	LA26_N	PHY_2_TX_EN	2717.697
RGMII3	rgmii_3_txc	35	D22	LA25_P	PHY_3_GTXCLK	2650.733
	rgmii_3_rxc	35	D20	LA18_P_CC	PHY_3_RXC	2771.734
	rgmii_3_rd[0]	35	C15	LA30_P	PHY_3_RXD_0	2748.336
	rgmii_3_rd[1]	35	A16	LA28_P	PHY_3_RXD_1	2714.475
	rgmii_3_rd[2]	35	C18	LA29_N	PHY_3_RXD_2	2689.041
	rgmii_3_rd[3]	35	C17	LA29_P	PHY_3_RXD_3	2717.528
	rgmii_3_rx_ctl	35	A19	LA24_N	PHY_3_RX_DV	2768.749
	rgmii_3_td[0]	35	F19	LA22_N	PHY_3_TXD_0	2639.867
	rgmii_3_td[1]	35	E20	LA21_N	PHY_3_TXD_1	2641.387
	rgmii_3_td[2]	35	G16	LA19_N	PHY_3_TXD_2	2685.051
	rgmii_3_td[3]	35	G19	LA22_P	PHY_3_TXD_3	2633.939
	rgmii_3_tx_ctl	35	E19	LA21_P	PHY_3_TX_EN	2648.097
RST	phy_rst_n	35	B17	LA31_N	PHY_CFG_RESET_N	3522.908
MDIO	mdio_mdc	34	J21	LA08_P	PHY_MDC	1637.892
	mdio_io	34	P22	LA03_N	PHY_MDIO	1727.127
INT	phy_int[0]	34	N20	LA01_N_CC	PHY_0_INT	1995.55
	phy_int[1]	34	M21	LA04_P	PHY_1_INT	1776.663
	phy_int[2]	35	B20	LA17_N_CC	PHY_2_INT	1937.523
	phy_int[3]	35	G20	LA20_P	PHY_3_INT	2071.132
CLK	fmc_clk125m_p	34	L18	CLK0_M2C_P	CLK_LVDS_PHY_125M_P	603.848
	fmc_clk125m_n	34	L19	CLK0_M2C_N	CLK_LVDS_PHY_125M_N	603.848

7.12. Board Setting

7.12.1. Jumper Settings

For more detailed information about jumper setting, please refer to ZedBoard Hardware User Guide. To make the 2.5V powered PHY on FMCx4 GE mezzanine card work appropriately, JP18must be setting to 2.5V mode.

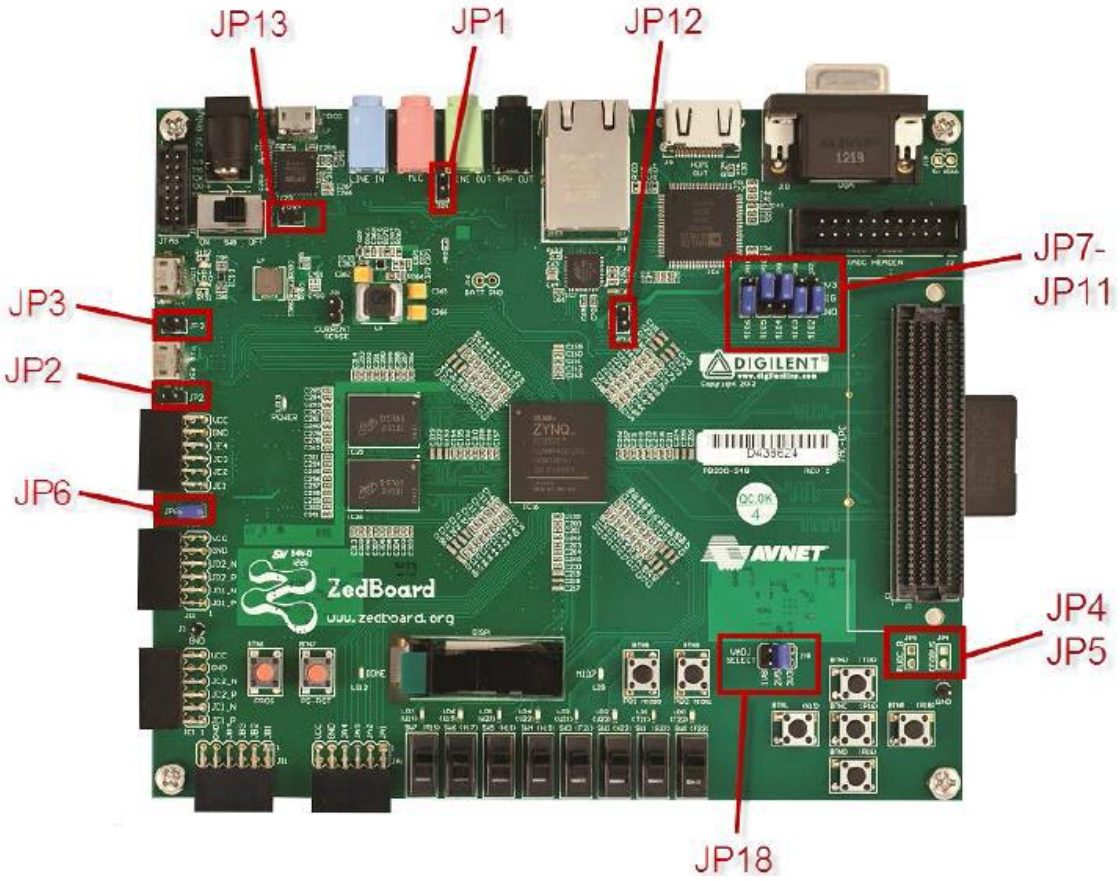

Figure 48: Main Board (ZedBoard) Jumper Locations

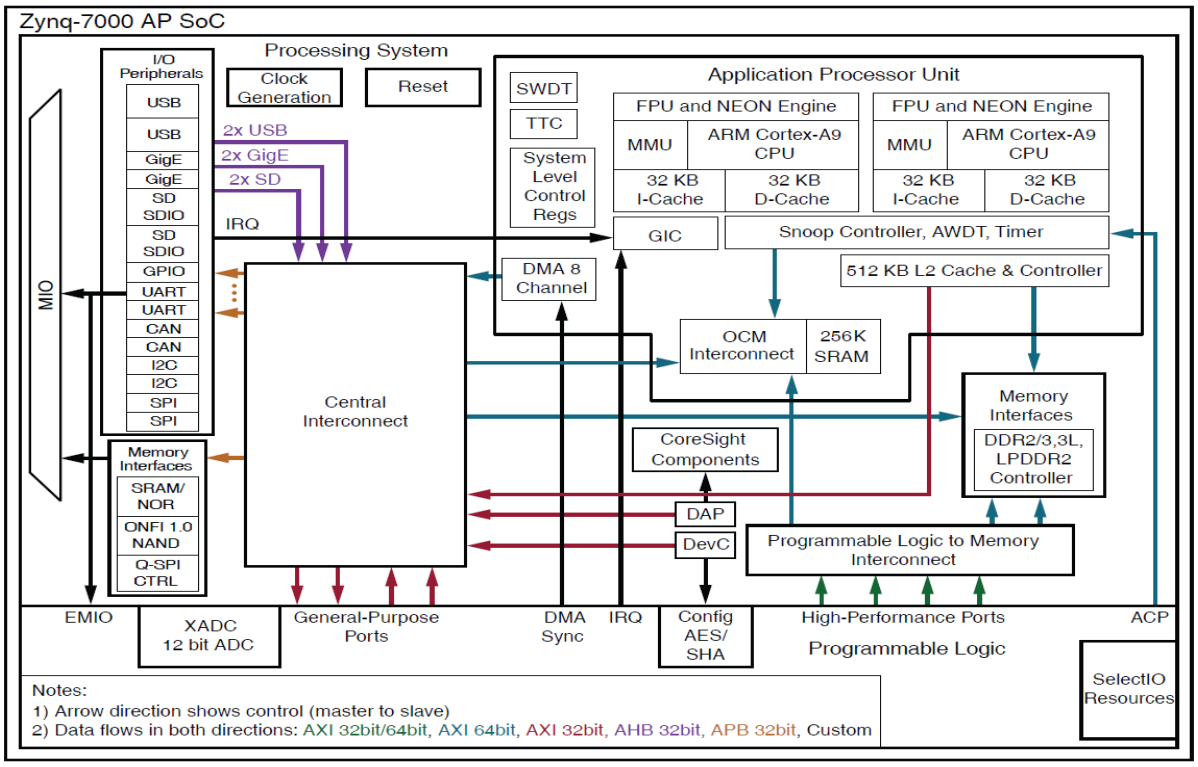

Figure 49. zynq-overview

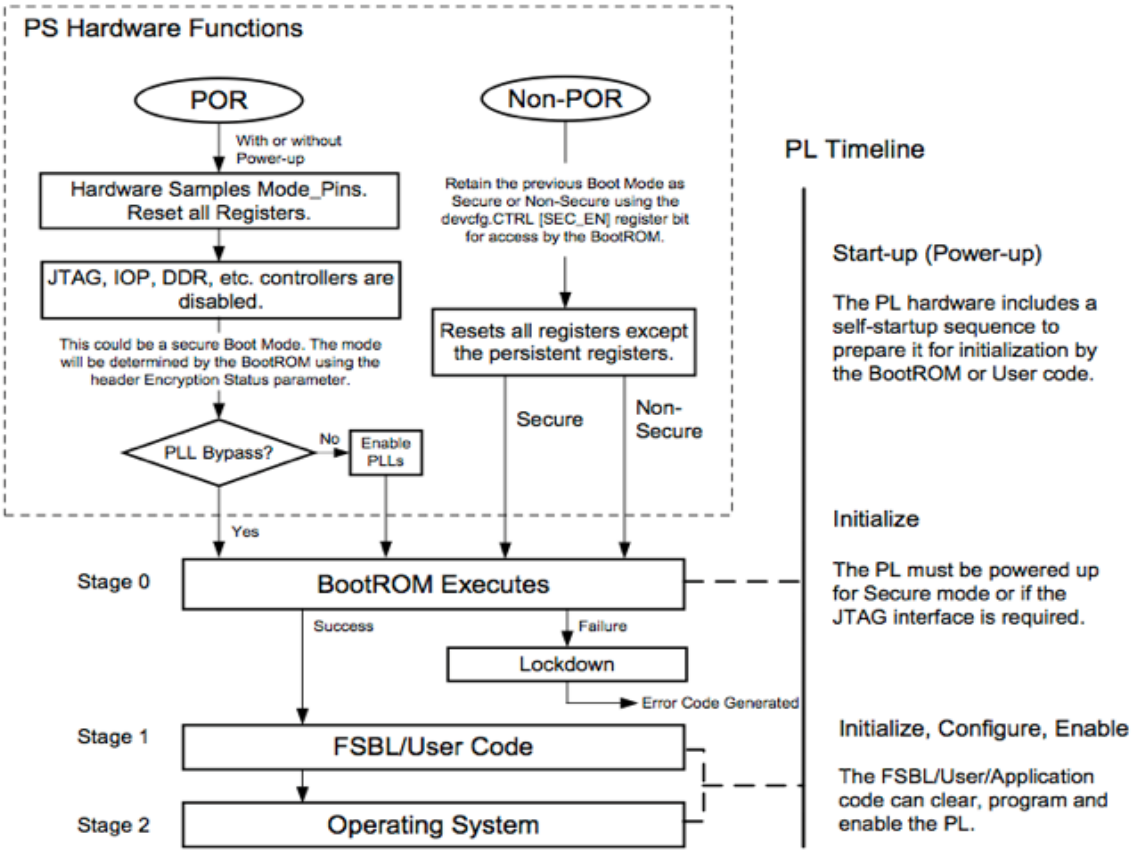

Figure 50: zynq- bootsystem

7.12.2. Build the System

Here is the Zynq design flow

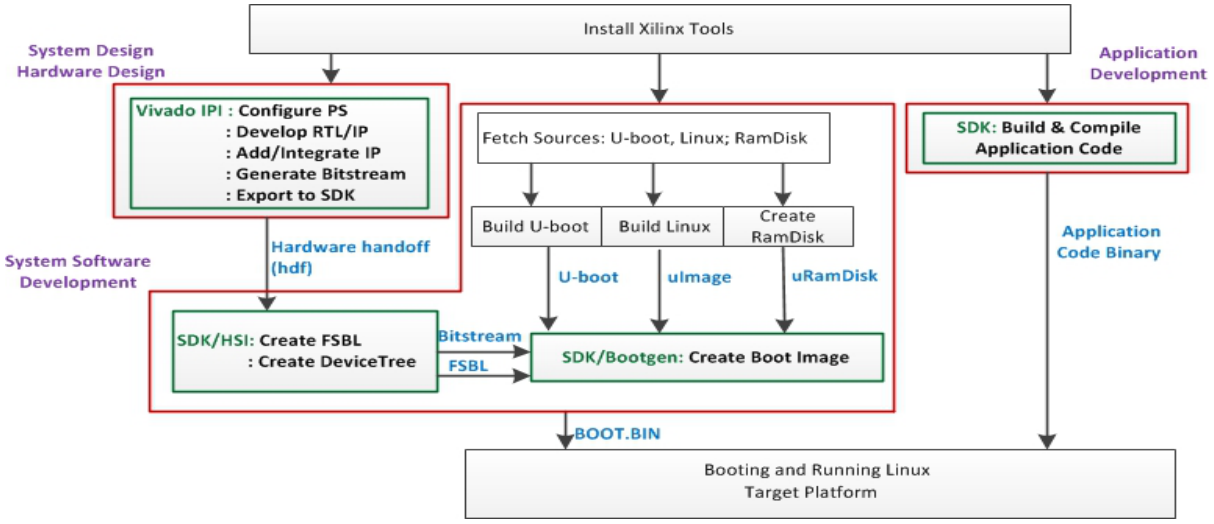

Figure 51: zynq-design-flow

7.12.3. Zynq Hardware Design

The Zynq hardware design generates the system.bit, defining the ARM CPU settings in Zynq PS and the FPGA bitstream in Zynq PL.

- Create IP cores and hardware design
- Configure the processor
- Add and integrate IP cores
- Generate the bitstream
- Export the hardware design to SDK

The entire process is implemented in Xilinx Vivado tool.

7.12.4. Zynq Software Design

The Zynq software design introduces several steps to build up an entire embedded system, from boot loader to Linux kernel, including

- First stage boot loader, FSBL
- Second stage boot loader, SSBL, using u-boot
- Root file system, rootfs
- Device tree
- Linux kernel

7.12.5. FMC Connection

This design uses JTAG UART instead of USB UART and to use this project JTAG is required.

Connect 4 Port Gigabit Ethernet FMC module to the FMC connector of Proteus.

Connect ethernet cables to the ports of FMCETHX001 module, and the other ends to the ports of a Gigabit switch.

Ethernet port of the host PC must also be connected to the Gigabit switch.

Connect JTAG and power up Proteus.

Go to- 1. Control Panel-->Select “Network and Sharing Centre” option--> “Network and Internet”

--> Select “Change adapter settings”. -->Right click on Ethernet-->click properties and select “IPv4” --> Change the IPv4 address -->say 192.168.5.100, and configure gateway as 192.168.5.1.

Make sure that the host PC supports 1000Mbps (1 Gbps).

7.12.6. FPGA Side:

In the "Bitstreams" folder, edit the batch file "download.bat" to specify the correct location of "xsct.bat" in your Vivado.

Now, double-click "download.bat" file. Observe the output displayed in the PuTTY window that pops up.

Now, telnet the ports at their respective IP addresses as shown in the terminal. That is, to test port A, open a telnet session with IP Address 192.168.5.10 at port 7.

Ethernet FMC Port	IP Address	Telnet Port
Port A	192.168.5.10	7
Port B	192.168.5.11	8
Port C	192.168.5.12	9
Port D	192.168.5.13	10

Using the project**:** Open the project(.xpr) file and Open the Block Design. Then Generate the Bitstream. After the Bitstream is generated successfully, the .bit

file will be created in the "Proteus_Gigabit_Ethernet.runs-->impl_1" folder.

In Xilinx SDK, program the FPGA on Proteus with a simple bootloop program by selecting “Program FPGA” option from “Xilinx” menu.

Connect FMCETHX001 module and change the network connections settings as mentioned in quickstart.

Now, right click on the .elf file in the Project Explorer and select “Launch on Hardware” (NOTE: ELF file needs to be

downloaded to Proteus via JTAG after programming .bit file). Observe the output displayed in SDK console. You can telnet the ports as described in quickstart.

CONCLUSION & FUTURE WORK

8. CONCLUSION AND FUTURE WORK

The SDN switches are the fundamental infrastructure to supply flexible control of flows. SDN software switches running on commodity multicore platforms are widely deployed due to their upgradability, programmability, and low cost. However, the forwarding performance as well as security capacity provided by general-purpose SDN software switch platform is usually not satisfied. The case becomes even worse for OpenFlow forwarding. In this thesis, the details of SDN, OpenFlow system, a model SDN switch, platform management system and FPGA to Accelerate SDN model has been described.

The flow entry composer module, the flow table controller module, the action processor module, and the controller policy module are the four key components of the OpenFlow switch that are incorporated in our design, because the switch is intended for research purposes only and not for commercial use. The FAS mechanism, which is FPGA-based and speeds up and secures SDN software switches, is also proposed in this thesis. The SDN software switch's time-consuming and real-time security modules can be offloaded using the FAS framework, and it also uses some optimization approaches to address the performance bottleneck between the software and the FPGA hardware. Some experimental findings demonstrate that FAS outperforms and makes good use of FPGA resources commodity platforms with a forwarding rate for tiny packets that is about 44% greater. By enabling the integration of bump-in-the-wire security modules in FPGA, FAS can be used to improve the security of SDN software switches.

The work also presents a flexible Platform management system. It is a very cost-effective solution which is easily re-configurable and re-programmable. It has the ability and potential to change perception of the world. If used and implemented correctly, it might be on the verge of a smarter, faster and a more efficient technological domain.

It is crucial to emphasise that the aforementioned conclusions are specific to the study described in this master's thesis and cannot be applied generally. Because the OpenFlow switch implementation has some limitations, this may inspire further research. Firstly, there is just one Ethernet port on the FPGA platform that was utilised to create the OpenFlow switch. Secondly, not all of the OpenFlow switch's features have been fully implemented, nor have all of the controller's features. For instance, in the OpenFlow switch component, just the forwarding action is executed, and in the controller policy module, only the writing of the flow entry is planned. Finally, the performance metrics are measured under simulation test environment by generating the packets on the board and not under real-time internet environment. According

to the limitation discussed above, there are lots of potental future work dimensions. The performance metrics (e.g, switch service time, sojourn time and controller service time) of the OpenFlow switch can be measured under the real-life internet environment in the future, and more performance metrics can be measured such as the lost rate, etc. Also, OpenFlow switch and controller can be implemented on the FPGA-platform with more Ethernet ports.

List of Publications as an outcome of this Dissertation

Paper title: 'Flexible, Programmable Hardware Management System for Switches'

Authors name:

1.Sabuj Biswas

2.Bappaditya Sinha

3.Chandan Mazumdar

Conference name: Calcon 2022, IEEE Kolkata section

Date of Submission: 30.06.2022

Status: Under review

REFERENCES

[1] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan,Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: a distributed control platform for large-scale production networks," in Proceedings of the 9th USENIX conference on Operating systems design and implementation, OSDI'10, (Berkeley, CA, USA), pp. 1–6, USENIX Association, 2010.

[2] O. Fundation, "Software-Defined Networking: The New Norm of Networks," 2012. https://www.opennetworking.org/images/stories/downloads/sdn-resources/ white-papers/wp-sdn-newnorm.pdf.

[3] FeiHu,QiHao, KeBao, "A Survey on Software-Defined Network (SDN) and OpenFlow: Form Concept to Implementation," IEEE Communications Surveys & Tutorials, 2013, unpublished.

[4] H. Hata, "A study of requirements for sdn switch platform," in Intelligent Signal Processing and Communications Systems (ISPACS), 2013 International Symposium on, pp. 79–84, Nov 2013.

[5] J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, and N. McKeown, "Implementing an openflow switch on the netfpga platform," in Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ANCS '08, (New York, NY, USA), pp. 1–9, ACM, 2008. http://doi.acm.org/10.1145/1477942.1477944.

[6] M. Wielgosz, M. Panggabean, J. Wang, and L. A. Rønningen, "An fpgabased platform for a network atchitecture with delay guarantee," Journal of Circuits, Systems and Computers, vol. 22, no. 06, p. 1350045, 2013. http://www.worldscientific.com/doi/abs/10.1142/S021812661350045X.

[7] G. Antichi, A. Di Pietro, S. Giordano, G. Procissi, and D. Ficara, "Design and development of an openflow compliant smart gigabit switch," in Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE, pp. 1–5, Dec 2011.

[8] A. Khan and N. Dave, "Enabling hardware exploration in software-defined networking: A flexible, portable openflow switch," in Field-Programmable Custom Computing Machines (FCCM), 2013 IEEE 21st Annual International Symposium on, pp. 145–148, April 2013.

[9] V. Tanyingyong, M. Hidell, and P. Sjödin, "Improving pc-based openflow switching

performance," in Proceedings of the 6th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ANCS '10, (New York, NY, USA), pp. 13:1–13:2, ACM, 2010. http://doi.acm.org/10.1145/1872007.1872023, doi = 10.1145/1872007.1872023.

[10] Y. Luo, P. Cascon, E. Murray, and J. Ortega, "Accelerating openflow switching with network processors," in Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ANCS '09, (New York, NY, USA), pp. 70–71, ACM, 2009. http://doi.acm.org/10.1145/1882486.1882504.

[11] A. Bianco, R. Birke, L. Giraudo, and M. Palacin, "Openflow switching: Data plane performance," in Communications (ICC), 2010 IEEE International Conference on, pp. 1–5, May 2010.

[12] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, "Modeling and performance evaluation of an openflow architecture," in Proceedings of the 23rd International Teletraffic Congress, ITC '11, pp. 1–7, International Teletraffic Congress, 2011. http://dl.acm.org/citation.cfm?id=2043468.2043470.

[13] R. Chua, "Juniper's New SDN Strategy and Contrail's Starring Role," Posted on SDN central: Jan. 15, 2013, http://www.sdncentral.com/sdn-blog/junipernew- sdn-strategy-contrail-role/2013/01/ Retrieved: May 28, 2014.

[14] H. Zimmermann, "OSI Reference Model–The ISO Model of Architecture for Open Systems Interconnection," Communications, IEEE Transactions on, vol. 28, pp. 425–432, Apr 1980.

[15] ONF, "Open Networking Foundation." https://www.opennetworking.org/ Retrieved May 28, 2014.

[16] H. Zimmermann, "OSI Reference Model–The ISO Model of Architecture for Open Systems Interconnection," Communications, IEEE Transactions on, vol. 28, pp. 425–432, Apr 1980.

[17]. Ferro, G. OpenFlow and Software Defined Networking ipspace webinar. [www]. [Cited 13.12.12]. Available at: http://demo.ipspace.net/get/OpenFlow

[18] IBM, "Software defined networking," IBM Systems and TechnologyThought Leadership White Paper, Octorber 2012. http://ict.unimap.edu.my/images/doc/SDN%20IBM%20WhitePaper.pdf.

[19] NEC, OpenFlow Feature Guide (IP8800/S3640), May 2010. http://support.necam.com/kbtools/sdocs.cfm?id=fcbdcb3e-45fa-4ec4-9311-215bd9ab9f81.

[20] ONF, "OpenFlow Switch Specification," Dec. 2011. http://goo.gl/tKo6r.

[21] https://www.manageengine.com/network-monitoring/switch monitoring.html?utm_source=Comparitech&utm_medium=Website-cpc&utm_campaign=OPM-SwitchMonSoftware

[22] FreeRTOS for ESP32. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos.html

Last accessed on: 20-05-2022.

[23] Espressif esp32 datasheet. https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

Last accessed on: 20-05-2022

[24] Interrupt handling in esp32. https://lastminuteengineers.com/handling-esp32-gpio-interrupts-tutorial/

Last accessed on: 22-05-2022

[25] Work on temperature sensor LM35 with esp32. https://esp32io.com/tutorials/esp32-lm35-temperature-sensor

Last accessed on: 26-05-2022

[26] LM35 temperature sensor datasheet. https://www.ti.com/lit/ds/symlink/lm35.pdf

Last accessed on: 24-05-2022

[27] P. Bosshart, G. Gibb, H.-S. Kim et al., "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," in Proceedings of the ACM SIGCOMM 2013 Conference onApplications, Technologies,Architectures, and Protocols for Computer Communication (SIGCOMM '13), pp. 99–110, August 2013.

[28] R. Ozdag, Intel Ethernet Switch FM6000 Series Software Defined Networking, August, Intel Corporation, 2012.

[29] The OpenFlow Consortium, *Openflow Switching Reference System*, January 2011, http://www.openflowswitch.org.

[30] "OFSoftSwitch13," http://cpqd.github.com/ofsoftswitch13.

[31] Y. Mundada, R. Sherwood, and N. Feamster, "An OpenFlow switch element for Click , in Symposium on Click Modular Router," 2009.

[32] *Open vSwitch – An Open Virtual Switch*, September 2014, http://www.openvswitch.org.

[33] Z. Cai, Z. Wang, K. Zheng, and J. Cao, "A Distributed TCAM coprocessor architecture for integrated longest prefix matching, policy filtering, and content filtering," IEEE Transactions on Computers, vol. 62, no. 3, pp. 417–427, 2013.

[34] A. Putnam, A. M. Caulfield, E. S. Chung et al., "A reconfigurable fabric for accelerating large-scale datacenter services," in Proceedings of the ACM/IEEE 41st International

Symposium on Computer Architecture (ISCA ’14), pp. 13–24, IEEE, Minneapolis, Minn, USA, June 2014.

[35] N. Shelly, E. J. Jackson, T. Koponen, N. McKeown, and J. Rajahalme, “Flow caching for high entropy packet fields,” in Proceedings of the 3rd ACM SIGCOMM 2014 Workshop on Hot Topics in Software Defined Networking (HotSDN ’14), pp. 151– 156, USA, August 2014.

[36] J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, and N. McKeown, “Implementing an OpenFlow switch on the NetFPGA platform,” in Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS ’08), pp. 1–9, USA, November 2008.

[37] B. Pfaff, J. Pettit, T. Koponen et al., “The design and implementation of open vSwitch,” in Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI ’15), pp. 117–130, USA, May 2015.

[38] A. Tirumala, F. Qin, J. Dugan et al., iPerf: TCP/UDP Bandwidth Measurement Tool, 2008.

[39] OProfile, http://oprofile.sourceforge.net.

[40] G. Liao, X. Znu, and L. Bnuyan, “A new server I/O architecture for high speed networks,” in Proceedings of the 17th International Symposium on High-Performance Computer Architecture (HPCA ’11), pp. 255–265, USA, February 2011.

[41] Intel, “Intel data plane development kit (intel DPDK),” in Programmer’s Guide, October 2013.

[42] L. Rizzo, “Netmap: A novel framework for fast packet I/O,” in Proceedings of the USENIX Annual Technical Conference, pp. 101–112, 2012.

[43] K. Subramanian, L. D’Antoni, and A. Akella, “Genesis: Synthesizing forwarding tables in multi-tenant networks,” in Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL ’17), pp. 572–585, France, January 2017.

[44] S. Han, K. Jang, K. Park, and S. Moon, “PacketShader: A GPUaccelerated software router,” in Proceedings of the 7th International Conference on Autonomic Computing (SIGCOMM ’10), pp. 195–206, India, September 2010.

[45] L. Tang, J. Yan, Z. Sun, T. Li, and M. Zhang, “Towards high performance packet processing on commodity multi-cores: current issues and future directions,” Science China Information Sciences, pp. 1–16, 2015.

[46] L. Rizzo,M. Carbone, and G. Catalli, “Transparent acceleration of software packet forwarding using netmap,” in Proceedings of the IEEE Conference on Computer Communications (INFOCOM ’12), pp. 2471–2479, USA, March 2012.

[47] V. Tanyingyong, M. Hidell, and P. Sjodin, “Using hardware classification to improve PC based OpenFlow switching,” in Proceedings of the 2011 IEEE 12th International Conference on High Performance Switching and Routing (HPSR ’11), pp. 215–221, Spain, July 2011.
[48] R. Narayanan, S. Kotha, G. Lin et al., “Macroflows and microflows: Enabling rapid network innovation through a split SDN data plane,” in Proceedings of the 1st European Workshop on Software Defined Networks (EWSDN ’12), pp. 79–84, Germany, October 2012.

APPENDIX

BILL OF MATERIAL OF PLATFORM MANAGEMENT SYSTEM

Sl no	Item	Legend	Part No./ Value	Quantity	Remark
1.	ESP32 Microcontroller	E1	211-161007	1	Microcontroller
2.	Boost Converter	B1	XL6009E1	1	-
3.	MOSFET	M1	IRFZ44N	1	N-channel MOSFET
4.	MOSFET	M2	IRF4905	1	P-channel MOSFET
5.	MOSPEC	M3	PBYR2545CT	2	Schottky barrier rectifier diodes
6.	Capacitor	C1	100µF/6.3V	1	Electrolytic Capacitor (Radial)
7.	Multiplexer	M1,M2,M3,M4	CD74HC4067	4	16 Channel Analog Digital Multiplexer
8.	Zener Diode	Z1	1N4733A	1	Forward bias it allows current, and in reverse bias it blocks current
9.	Temparature Sensor	T1	LM35	6	8-lead surface-mount
10.	Port sensor	P1	ACS712	1	Current Sensor Module
11.	Resistor	R1	10kΩ, 0.25W	9	-
12.	Registor	R2	1kΩ, 0.25W	7	-
13.	LED	L1	5mm	20	Red
14.	LED	L2	5mm	4	yellow
15.	LED	L3	5mm	1	Blue
16.	Fan	F1,F2	12V,0.41A	2	-
17.	Power supply	P1,P2	5V,10A	2	-
18.	Jumper Wire	J1,J2,J3,J4,J5	-	72	-