STUDY OF OPEN SOURCE WIFI USING FPGA

A THESIS

Submitted by

T.RAKESH VARMA [N170349]

J.BANSY SUBBU [N171141]

S.VENKATESWARA PRASAD [N170248]

K.TATAJI [N170273]

for the award of the degree

of

B.Tech in Electronics and Communication Engineering

under the supervision/guidance of **Mr.BHARATH GANJI,**

Assistant Professor, Department of ECE,RGUKT-NUZVID,ANDHRA PRADESH

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES
NUZVID CAMPUS

ANDHRA PRADESH-521 202

APRIL 2023

Dedication

This thesis is wholeheartedly dedicated to our guide **Mr. Bharath Ganji** and our beloved parents who have been our source of inspiration and gave us strength when we thought of giving up, who continuously provide their moral, spiritual, emotional and financial support.

To our friends and classmates who share their works of advice and encouragement to finish this thesis.

And lastly, we dedicated this book to Almighty God, thank you for the guidance, strength, power of mind, protection, skills and for giving me a healthy life. All of these, We offer to you.

THESIS CERTIFICATE

This is to certify that the thesis entitled **"STUDY OF OPEN SOURCE WIFI USING FPGA"** submitted by **T.RAKESH VARMA, J.BANSY SUBBU, S.VENKATESWARA PRASAD, K.TATAJI** to the Department of Electronics and Communication Engineering, Rajiv Gandhi University of Knowledge Technologies - Nuzvid campus, AP for the award of the degree of **B.Tech in ECE** is a bonafide record of work carried out by them under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Mr.Bharath Ganji
Project Guide
Assistant Professor
Department of ECE
RGUKT Nuzvid campus
Nuzvid - 521202

Mr. Shyam Peraka
Head of the Department
Department of ECE
RGUKT Nuzvid campus
Nuzvid - 521202

Date:

ACKNOWLEDGEMENTS

We would like to express our profound gratitude and deep regards to our guide **Mr.Bharath Ganji** for his exemplary guidance, monitoring and constant encouragement throughout the course of this thesis. We are extremely grateful for the confidence bestowed in us and entrusting our project entitled “OPEN WIFI”. At this juncture I feel deeply honored in expressing our sincere thanks to him for making the resources and providing valuable insights leading to the successful completion of my project.

We would also like to extend my thanks to the Head of the Department,**Mr.SHYAM PERAKA** sir, for their support and encouragement throughout the project. Their trust in our abilities and commitment to providing a conducive learning environment has been a significant motivation for me.

We would like to thank RGUKT Nuzvid Director, faculty and staff for their valuable suggestions and discussions. Last but not least we thank almighty and we place a deep sense of gratitude to our family members and our friends who have been constant source of information during the preparation of this project work.

T.RAKESH VARMA[N170349]

J.BANSY SUBBU [N171141]

S.VENKATESWARA PRASAD[N170248]

K.TATAJI[N170273]

ABSTRACT

WiFi is a technology that enables devices to connect to the internet or other networks wirelessly. WiFi technology has seen tremendous growth over the past few decades, and has become an essential part of modern-day communication and networking.WiFi research focuses on improving the speed, range, security, and reliability of wireless networks, as well as developing new WiFi-enabled devices and applications. In this regard, we choose the open source project OPEN WiFi to study the Wi-Fi implementation.

OpenWiFi is an open-source project hosted on GitHub that provides a hardware-agnostic WiFi stack for software-defined radios (SDRs). It aims to enable experimentation and research on wireless communications by allowing researchers and developers to implement custom WiFi protocols and algorithms on off-the-shelf SDRs. The project provides a framework for implementing PHY and MAC layers of WiFi protocols and supports a variety of SDR platforms, including USRP, LimeSDR, and PlutoSDR. OpenWiFi is written in C++ and runs on Linux, and it includes a set of tools and utilities for configuring and testing WiFi networks.

In this thesis, we reviews the Open wifi project implementation and studied the internal blocks of this project.We studied the standards of IEEE 802.11 and its various formats.

Keywords: IEEE 802.11 , IEEE 802.11b, DSSS, OPEN WiFi,SDR,OPEN OFDM.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

1.1. INTRODUCTION

Wi-Fi (Wireless Fidelity) is a wireless communication technology that allows electronic devices to connect and communicate with each other without the need for physical cables. It enables devices such as smartphones, laptops, and tablets to access the internet, transfer data, and communicate with other devices wirelessly.Wi-Fi technology is based on radio waves, which are transmitted over the airwaves to create a wireless network. The technology utilizes a wireless access point or router to provide a network connection that can be used by multiple devices simultaneously.

The popularity of Wi-Fi has grown significantly in recent years due to its convenience, flexibility, and ease of use. It has become an essential part of our daily lives, providing wireless connectivity in homes, offices, public spaces, and other areas.Wi-Fi technology has evolved over the years to provide higher data transfer speeds and improved network reliability. The latest Wi-Fi standard is Wi-Fi 6, which provides faster speeds, increased network capacity, and improved security features.

Overall, Wi-Fi technology has revolutionized the way we access and use the internet, providing a flexible, convenient, and reliable wireless connection that has transformed the way we live, work, and communicate.

The OpenWiFi project is an open-source initiative aimed at providing a full-stack, standards-compliant, and modular Wi-Fi design and implementation, running on commodity hardware.OpenWiFi is designed to run on a variety of hardware platforms, including software-defined radios (SDRs) and field-programmable gate arrays (FPGAs), among others. The project aims to provide a flexible and open platform for experimenting with new Wi-Fi technologies and protocols.

One of the key features of OpenWiFi is its support for the latest Wi-Fi standards, including 802.11ax (Wi-Fi 6) and 802.11ay (Wi-Fi 6E). The project also includes a variety of software components, such as drivers, firmware, and protocol stacks, that can be easily customized and extended.

Open-source Wi-Fi development, which Qualcomm Technologies has long supported, has been a driving force and an enabler of Wi-Fi research, performance enhancement, and new services for over a decade.

The OpenWiFi project has also continued to attract contributions from a growing community of developers, researchers, and enthusiasts. In early 2021, the OpenWiFi project released version 2.0 of its software, which included support for the latest Wi-Fi 6E standard. This update also included significant improvements to the project's codebase, making it more modular, efficient, and easier to use.

1.2. OBJECTIVE OF THE WORK

1. To understand the Wi-Fi communication system.
2. To implement the standard Wi-Fi protocol i.e., IEEE 802.11b
3. To study various other protocols of standard IEEE 802.11.
4. To study the open source project “Open Wi-Fi” and its implementation.

1.3. SCOPE OF THE THESIS

From this thesis we attempted to understand the communication system behind the Wi-Fi. We collected the information about standard IEEE 802.11 regarding WiFi architecture and Specifications of wireless LANs. We researched and analyzed the basic modulations and framing used in IEEE 802.11b. We studied open source project Open WiFi and analyzed how it implemented.

This OpenWiFi project provides a full-stack, modular, and standards-compliant Wi-Fi design and implementation that can run on commodity hardware. The project aims to provide a flexible and open platform for experimenting with new Wi-Fi technologies and protocols.

The thesis organization is as follows:

(I) Implemented the standard WiFi protocol i.e.,IEEE 802.11b

(II) Studied the open source project “Open WiFi” and how it implemented.

CHAPTER 2

IEEE 802.11 STANDARDS

2.1. INTRODUCTION

Wi-Fi is a technology that allows many electronic devices to exchange data or connect to the internet wirelessly using radio waves. The Wi-Fi Alliance defines Wi-Fi devices as any "Wireless Local Area Network (WLAN) products that are based on the Institute of Electrical and Electronics Engineers' (IEEE) 802.11 standards".The key advantage of IEEE 802.11 devices is that they allow less-expensive deployment of Local Area Networks (LANs).

IEEE 802.11 defines the physical and media access control (MAC) layer protocols for wireless communication over the 2.4 GHz and 5 GHz frequency bands.The IEEE 802.11 standards specify the requirements for wireless LANs and provide the basis for interoperability between different manufacturers' products. The standards are continuously updated and revised to keep up with the latest advancements in technology and to address new challenges in wireless networking.

The IEEE 802.11 standards specify the physical layer and media access control layer protocols for wireless networking, including the modulation scheme, encoding, and transmission protocols. The standards also define the security mechanisms, such as WEP, WPA, and WPA2, used to protect wireless networks from unauthorized access.

2.2. EVOLUTION OF WiFi

Wi-Fi technology has evolved significantly since it was first introduced in 1997. Here are some of the key milestones in the evolution of Wi-Fi as shown in the table 2.1:

IEEE 802.11 PHY Standards						
Release Date	Standard	Frequency Band (GHz)	Bandwidth (MHz)	Modulation	Advanced Antenna Technologies	Maximum Data Rate
1997	802.11	2.4 GHz	20 MHz	DSSS, FHSS	N/A	2 Mbits/s
1999	802.11b	2.4 GHz	20 MHz	DSSS	N/A	11 Mbits/s
1999	802.11a	5 GHz	20 MHz	OFDM	N/A	54 Mbits/s
2003	802.11g	2.4 GHz	20 MHz	DSSS, OFDM	N/A	542 Mbits/s
2009	802.11n	2.4 GHz, 5 GHz	20 MHz, 40 MHz	OFDM	MIMO, up to 4 spatial streams	600 Mbits/s
2013	802.11ac	5 GHz	40 MHz, 80 MHz, 160 MHz	OFDM	MIMO, MU-MIMO, up to 8 spatial streams	6.93 Gbits/s

Table 2.1: IEEE 802.11 PHY Standards [5]

1) **802.11**: In 1997, the IEEE introduced the first wireless LAN standard, 802.11, which supported data rates of up to 2 Mbps.

2) **802.11b**: In 1999, the IEEE introduced the 802.11b standard, which supported data rates of up to 11 Mbps using the 2.4 GHz frequency band.

3) **802.11a**: Also in 1999, the IEEE introduced the 802.11a standard, which supported data rates of up to 54 Mbps using the 5 GHz frequency band.

4) **802.11g**: In 2003, the IEEE introduced the 802.11g standard, which supported data rates of up to 54 Mbps using the 2.4 GHz frequency band.

5) **802.11n**: In 2009, the IEEE introduced the 802.11n standard, which supported data rates of up to 600 Mbps using both the 2.4 GHz and 5 GHz frequency bands.

6) **802.11ac**: In 2013, the IEEE introduced the 802.11ac standard, which supported data rates of up to 7 Gbps using the 5 GHz frequency band.

7) **802.11ax**: In 2019, the IEEE introduced the 802.11ax standard, also known as Wi-Fi 6, which supported data rates of up to 9.6 Gbps using both the 2.4 GHz and 5 GHz frequency bands.

In addition to faster data rates, each new generation of Wi-Fi technology has also introduced improvements in range, reliability, and energy efficiency. For example, the 802.11n standard introduced multiple-input, multiple-output (MIMO) technology, which uses multiple antennas to improve signal quality and range. The 802.11ax standard introduced orthogonal frequency-division multiple access (OFDMA), which improves network efficiency and enables more devices to connect simultaneously.

2.3. IEEE 802.11 STANDARD AND FORMATS

The 802.11 family is a series of over-the-air modulation techniques that share the same basic

protocol.It is a set of medium access control (MAC) and physical layer (PHY) specifications for implementing Wireless Local Area Network (WLAN) communication.These standards provide the basis for wireless network products using the Wi-Fi brand.

(1) **IEEE 802.11 b**:

IEEE 802.11b is a wireless networking standard that operates in the 2.4 GHz frequency band, and it was introduced in 1999. It is the first widely adopted wireless networking standard and supports data transfer rates of up to 11 Mbps.

802.11b uses the Direct Sequence Spread Spectrum (DSSS) modulation scheme to transmit data over the airwaves. This technology allows for multiple devices to communicate with each other over the same channel simultaneously, making it possible for multiple users to share a network connection.

One of the main advantages of 802.11b is its compatibility with a wide range of devices,and its relatively long range.However, one disadvantage of 802.11b is that its use of the 2.4 GHz frequency band can lead to interference with other devices that use the same bandits maximum data transfer rate of 11 Mbps is significantly slower than newer Wi-Fi standards.

(2) **IEEE 802.11 a:**

IEEE 802.11a is a Wi-Fi wireless networking standard that operates in the 5 GHz frequency band. It was introduced in 1999 and supports maximum data transfer rates of up to 54 Mbps.

802.11a uses Orthogonal Frequency Division Multiplexing (OFDM) as its modulation scheme. This allows for higher data rates and better performance in areas with interference, compared to the earlier IEEE 802.11b standard which uses the slower Direct Sequence Spread Spectrum (DSSS) modulation scheme.

One advantage of 802.11a is that the 5 GHz frequency range is less crowded than the 2.4 GHz range used by earlier Wi-Fi standards, which can lead to less interference and more reliable connections. However, the shorter wavelength of 5 GHz signals also means that they have a shorter range and are more easily blocked by obstacles such as walls

.802.11a is not compatible with the earlier 802.11b standard, as they use different frequency bands and modulation schemes. However, most modern Wi-Fi devices support both 802.11a and 802.11b/g/n/ac standards, allowing them to connect to a wider range of networks.

(3) IEEE 802.11g:

IEEE 802.11g is a wireless networking standard that operates in the 2.4 GHz frequency band, and it was introduced in 2003. It is an improvement over the earlier 802.11b standard, offering higher data transfer rates of up to 54 Mbps.

802.11g uses Orthogonal Frequency Division Multiplexing (OFDM) as its modulation scheme, similar to the 802.11a standard. This allows for higher data rates and better performance in areas with interference, compared to the earlier 802.11b standard which uses the slower Direct Sequence Spread Spectrum (DSSS) modulation scheme.

One of the main advantages of 802.11g is its compatibility with the older 802.11b standard, as they both operate in the 2.4 GHz frequency band. This means that devices that support 802.11g can connect to 802.11b networks, and vice versa. Another advantage is its longer range and better penetration through walls and other obstacles compared to 802.11a.

However, one disadvantage of 802.11g is that its use of the 2.4 GHz frequency band can lead to interference with other devices that use the same band, such as microwaves and cordless phones. Additionally, its maximum data transfer rate of 54 Mbps is significantly slower than newer Wi-Fi standards such as 802.11n and 802.11ac.

(4) IEEE 802.11n:

IEEE 802.11n is a wireless networking standard that operates in both the 2.4 GHz and 5 GHz frequency bands. It was introduced in 2009 and offers improved data transfer rates and reliability compared to earlier Wi-Fi standards.

802.11n uses Multiple-Input Multiple-Output (MIMO) technology to transmit multiple streams of data simultaneously, resulting in higher data transfer rates of up to 600 Mbps. It also uses channel bonding, which allows multiple channels to be used together to increase bandwidth and reduce interference.

One of the main advantages of 802.11n is its improved range and reliability, as it is less susceptible to interference from other devices and can penetrate walls and other obstacles better than earlier Wi-Fi standards. Additionally, its backwards compatibility with earlier Wi-Fi standards means that it can be used in networks with a mix of older and newer devices.

Overall, IEEE 802.11n represents a significant improvement over earlier Wi-Fi standards, offering higher data transfer rates, better range, and improved reliability. It is widely used in modern Wi-Fi networks, although it has since been superseded by newer and faster Wi-Fi standards such as 802.11ac and 802.11ax.

(5) IEEE 802.11ac:

IEEE 802.11ac is a wireless networking standard that operates in the 5 GHz frequency band and was introduced in 2013. It is an improvement over the earlier 802.11n standard, offering even higher data transfer rates and improved performance.

802.11ac uses a wider channel bandwidth of up to 160 MHz, compared to 40 MHz in 802.11n, which allows for much higher data transfer rates of up to several gigabits per second. It also uses Multi-User MIMO (MU-MIMO) technology, which allows for multiple devices to simultaneously transmit and receive data on different spatial streams, improving network efficiency and reducing latency.

One of the main advantages of 802.11ac is its significantly higher data transfer rates, which make it suitable for bandwidth-intensive applications such as streaming high-quality video or online gaming. It also offers improved range and reliability compared to earlier Wi-Fi standards, thanks to its use of advanced beamforming technology.

IEEE 802.11ac represents a significant improvement over earlier Wi-Fi standards, offering much higher data transfer rates, improved range and reliability, and better network efficiency.

2.4. IEEE 802.11 PROTOCOL ARCHITECTURE [4]

The Standard 802.11 covers protocols and operation of wireless networks. It only deals with the two lowest layers of the OSI reference model, the physical layer and the Data Link layer (or Media Access Control layer) which are shown in fig 2.1. The goal is for all the 802.11 series of standards to be backward compatible and to be compatible at the Medium Access Control (MAC) or Data Link layer. Each of the 802.11 standards should therefore only differ in physical layer (PHY) characteristics.

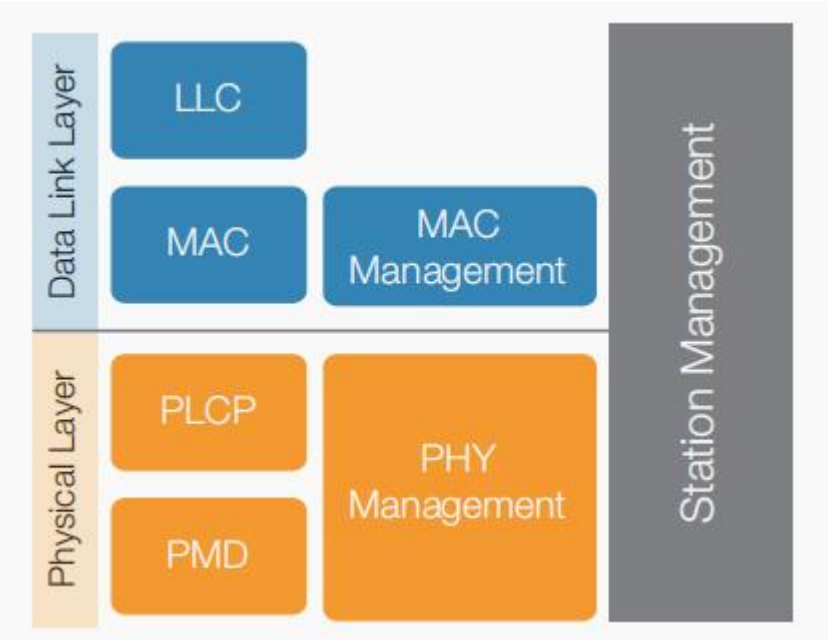

fig 2.1:The 802.11 standards focus on the Data Link and Physical Layers of the OSI reference model. [5]

The 802.11 protocol architecture is based on a layered approach, which includes the following layers:

(1) **Medium Access Control (MAC) layer:**:The MAC layer provides the functional and procedural means to transfer data between network entities and to detect and possibly correct errors that may occur in the physical layer. It provides access to contention based and contention free traffic on different kinds of physical layers.

In the MAC layer the responsibilities are divided into the MAC sub-layer and the MAC management sub-layer. The MAC sub-layer defines access mechanisms and packet formats. The MAC management sub-layer defines power management, security and roaming services.

(2) **Logical Link Control (LLC) layer**: This layer provides an interface between the MAC layer and the higher layer protocols. It is responsible for framing, flow control, and error recovery.

(3) **Physical layer (PHY):**The Physical Layer defines the electrical and physical specifications for devices. In particular, it defines the relationship between a device and a transmission medium. The major functions and services performed by the physical

layer are the following:

i) Establishment and termination of a connection to a communications medium.

ii)Participation in the process where the communication resources are effectively shared among multiple users. For example, contention resolution and flow control.

iii) Modulation or conversion between the representation of digital data in user equipment and the corresponding signals transmitted over a communications channel. These are signals operating over the physical cabling (such as copper and optical fiber) or over a radio link.

The Physical layer is divided into three sub layers.

1. The **Physical Layer Convergence Procedure** (PLCP) acts as an adaption layer. The PLCP is responsible for the Clear Channel Assessment (CCA) mode and building packets for different physical layer technologies.

2. The **Physical Medium Dependent** (PMD) layer specifies modulation and coding techniques.

3. The **PHY management laye**r takes care of the management issues like channel tuning.

The Station Management sub-layer is responsible for coordination of interactions between the MAC and PHY layers.

2.5. IEEE 802.11b OVERVIEW AND PACKET FORMAT

IEEE 802.11b is a wireless networking standard that operates in the 2.4 GHz frequency band and provides a maximum data rate of 11 Mbps.In IEEE 802.11b data is encoded using DSSS (Direct Sequence Spread Spectrum) technology. DSSS works by taking a data stream of zeros and ones and modulating it with a second pattern, the chipping sequence. In 802.11, that sequence is known as the Barker code, which is an 11 bits sequence (10110111000) that has certain mathematical properties making it ideal for modulating radio waves.

The basic data stream is XOR'd with the Barker code to generate a series of data objects called chips. Each bit is "encoded" by the 11bits Barker code, and each group of 11 chips encodes one bit of data. IEEE 802.11b uses 64 CCK (Complementary Code Keying) chipping sequences to achieve 11 Mbps.

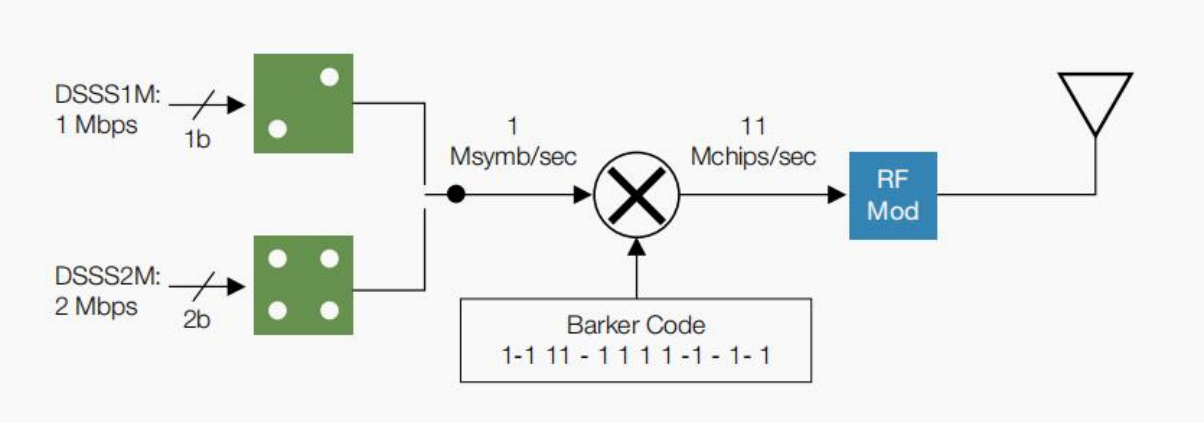

fig 2.2: BPSK/QPSK modulation techniques used in the 802.11b standard. [5]

The wireless radio generates a 2.4 GHz carrier wave (2.4 to 2.483 GHz) and modulates that wave using a variety of techniques.As shown in the fig 2.2, for 1 Mbps transmission, BPSK (Binary Phase Shift Keying) is used (one phase shift for each bit). To accomplish 2 Mbps transmission, QPSK (Quadrature Phase Shift Keying) is used. QPSK uses four rotations (0, 90, 180 and 270 degrees) to encode 2 bits of information in the same space as BPSK encodes 1.

Physical Layer Frame Structure:

The IEEE 802.11b physical layer frame structure is divided into three parts: the preamble, header, and data as shown in the fig 2.3.

fig 2.3: Each PHY packet contains a preamble, header and payload data [5]

(i) Preamble: The Preamble allows the receiver to obtain time and frequency synchronization and estimate channel characteristics for equalization. It is a bit sequence that receivers watch for to lock onto the rest of the transmission.

(ii) Header : The Header provides information about the packet configuration, such as format, data rates,data transmission, such as the length of the data, the data rate, and the type of modulation used etc.

(iii) **Data :** Payload Data contains the user's payload data being transported.The length and format of the data payload depend on the type of packet being transmitted.

IEEE 802.11b Format: Refer to fig 2.4 and table 2.2 for IEEE 802.11b packet format.

802.11b Packet Format (refer to figure below)	
Preamble and Header	Long (required in original) Short (optional in "b")
Preamble	Always uses DSS1M Long Preamble contains 144 bits (128 scrambled 1's + 16 SFD marker bits) Short Preamble contains 72 bits (56 scrambled 0's + 16 SFD marker bits)
Header	Long Header uses DSSS1M, Short Header uses DSSS2M Contains 48 bits indicating configuration: SIGNAL (8 bits): indicates payload data rate (1, 2, 5.5 or 11 Mbps) SERVICE (8 bits): additional HR configuration bits LENGTH (16 bits): length of data Payload in microseconds CRC (16 bits): Protects header data contents
Payload	Payload data modulated with DSSS1M, DSSS2M, CCK5.5M, or CCK11M

Table 2.2: IEEE 802.11b packet format [5]

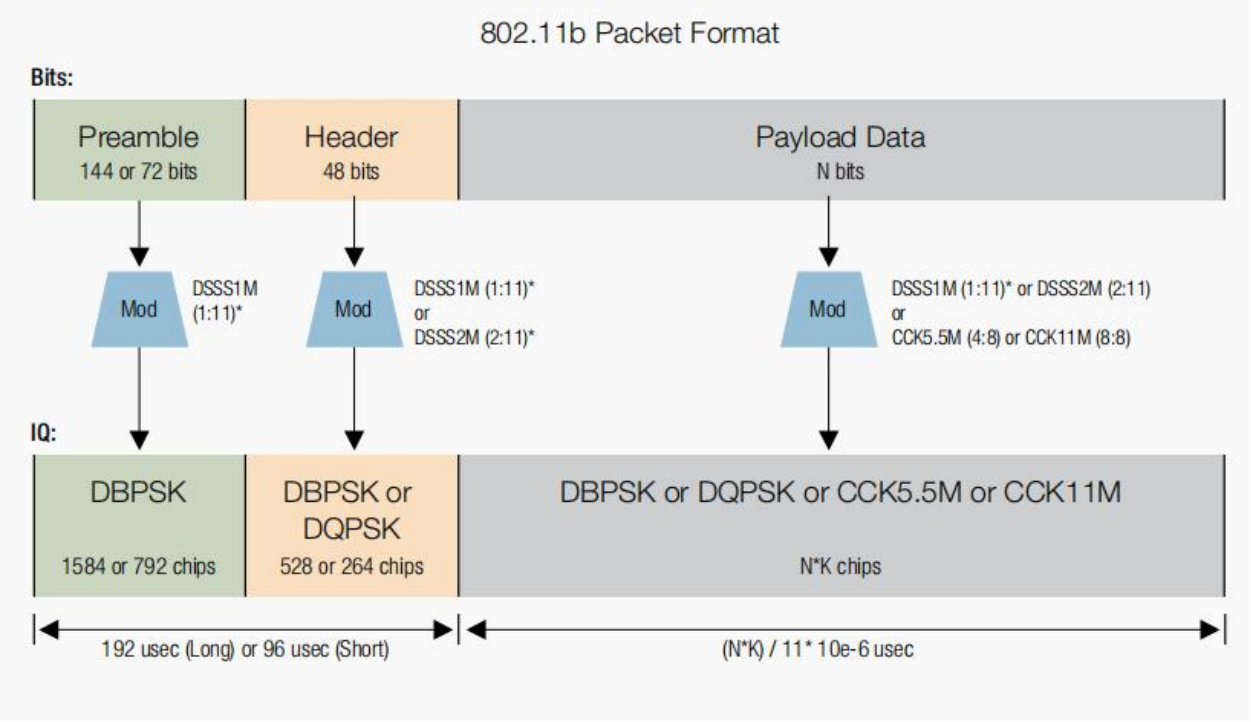

fig 2.4: IEEE 802.11b Format and its modulation [5]

CHAPTER 3

OPEN WiFi

3.1. INTRODUCTION:

OpenWiFi is an open-source project focused on developing hardware and software for WiFi communication using software-defined radio (SDR) platforms. The project's goal is to create an open and accessible platform that can be used to develop and test new WiFi protocols, as well as to provide a framework for wireless research and education.

The OpenWiFi software project provides a complete WiFi stack, including the physical layer (PHY), medium access control (MAC), and other higher-layer protocols such as TCP/IP. The software is designed to run on SDR platforms such as the Xilinx Zynq UltraScale+ RFSoC, the Analog Devices AD9361, and the Lime Microsystems LMS7002M.

3.2. OPEN WiFi HARDWARE ARCHITECTURE

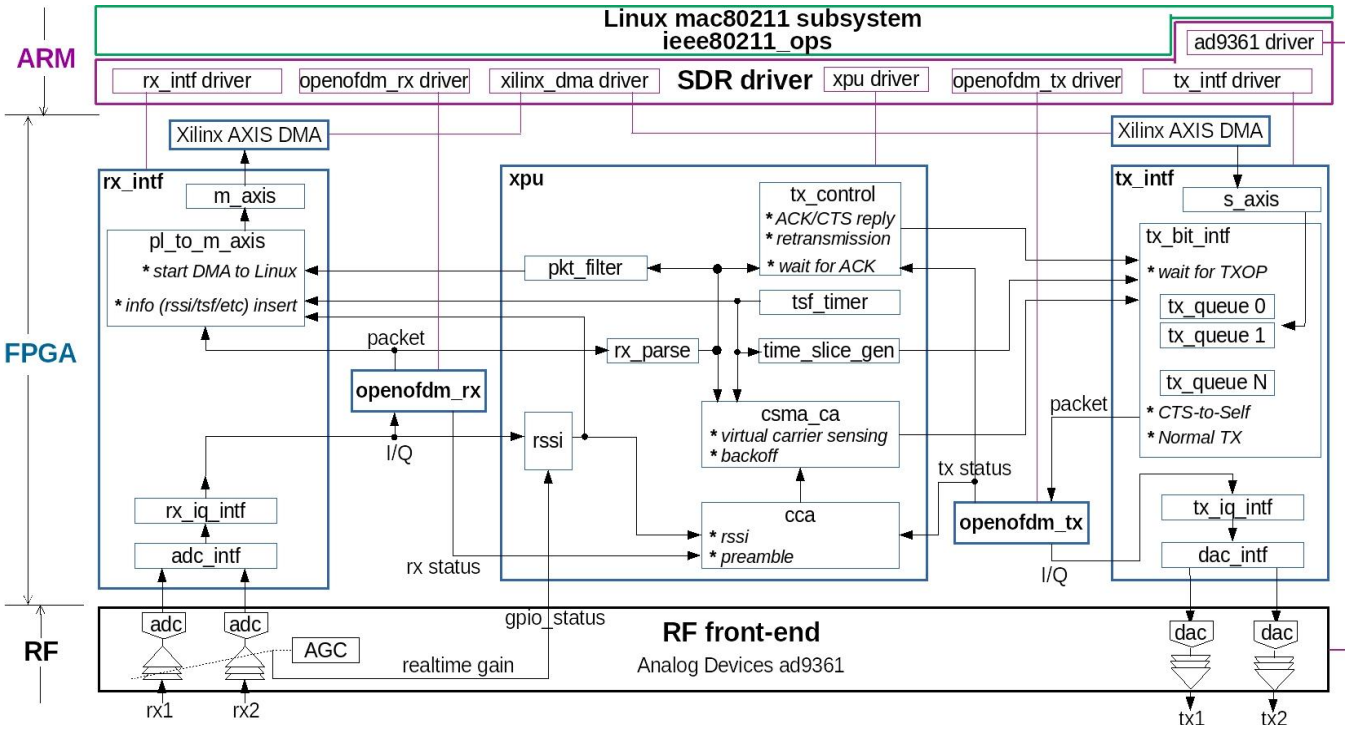

Fig 3.1 : OPEN WiFi HARDWARE ARCHITECTURE [1]

FPGA:

FPGA (Field-Programmable Gate Array) is an integrated circuit that can be programmed to perform specific logic functions. Unlike traditional ASICs (Application-Specific Integrated Circuits), FPGAs can be reprogrammed and reconfigured after manufacturing. This makes them highly flexible and adaptable, and they are widely used in a variety of applications, including digital signal processing, data center acceleration, and aerospace and defense systems.

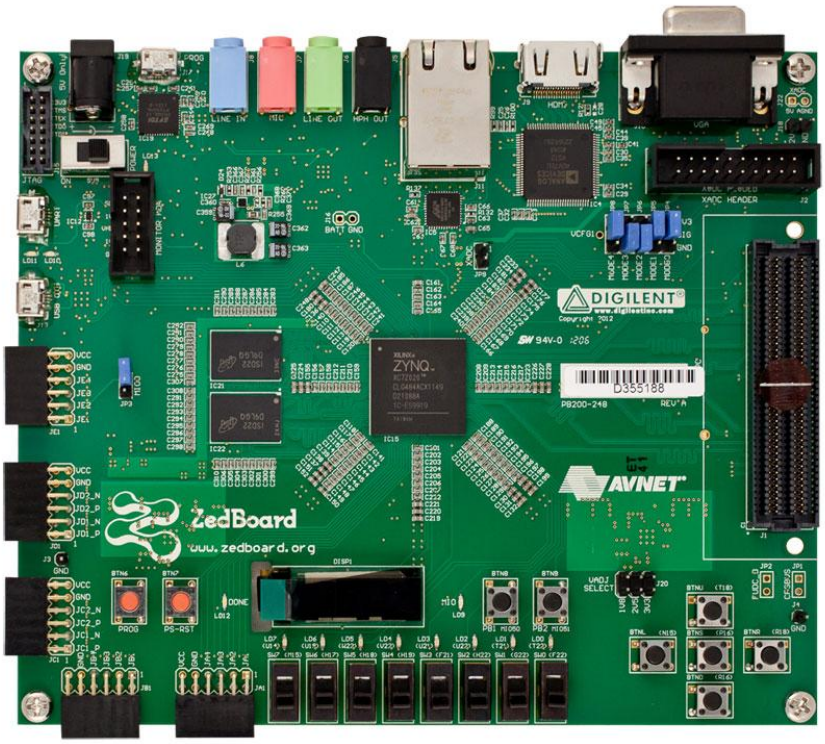

Fig 3.2. zed board

The Zynq family is a series of System-on-Chip (SoC) devices designed by Xilinx, a leading FPGA manufacturer. The Zynq family combines a dual-core ARM Cortex-A9 processor with FPGA logic on a single chip, providing a powerful and flexible platform for a wide range of applications. Zynq devices also feature a variety of interfaces, including high-speed serial links, gigabit Ethernet, and USB, as well as integrated memory controllers and programmable I/O. Zynq devices are commonly used in embedded systems, automotive and industrial control, and video processing applications.

Openofdm_tx:

The OpenOFDM transmitter is a hardware implementation of the Orthogonal Frequency Division Multiplexing (OFDM) modulation technique used in wireless communication systems. OFDM is a multi-carrier modulation technique that divides the available bandwidth into multiple subcarriers, each carrying a portion of the total data stream. This enables higher data rates and better spectral efficiency compared to traditional single-carrier modulation schemes.

The OpenOFDM transmitter consists of various signal processing modules that perform the following functions:

(1) Data framing and encapsulation: The input data is framed into packets and encapsulated with the appropriate header information required for OFDM transmission.

(2) Cyclic prefix insertion: A cyclic prefix is added to each OFDM symbol to mitigate inter-symbol interference caused by the wireless channel.

(3) Modulation: The input data is mapped onto the subcarriers using a modulation scheme such as QAM (Quadrature Amplitude Modulation).

(4) IFFT: An Inverse Fast Fourier Transform (IFFT) is applied to the modulated subcarriers to generate the time-domain OFDM symbols.

(5) RF signal generation: The OFDM symbols are upconverted to the carrier frequency and amplified to generate the RF signal that is transmitted over the wireless channel.

The OpenOFDM transmitter is a key component of the OpenWiFi system and provides a high-speed wireless data transmission capability using the OFDM modulation technique.The transmitted data can be received by a receiver implemented in the OpenWiFi system or by a compatible receiver in another wireless communication system.

tx_intf:

The "tx_intf" block in the above figure represents the interface between the OpenOFDM transmitter IP core and the rest of the system. It is responsible for interfacing the OpenOFDM transmitter with the higher-level logic of the system.

The "tx_intf" block acts as a bridge between the OpenOFDM transmitter and the system-level logic, providing a standardized interface that can be used by other system-level blocks

or modules. It is responsible for receiving input data and control signals from the system-level logic and providing them to the OpenOFDM transmitter IP core.

The "tx_intf" block provides the following functionalities:

(1) Data input: The "tx_intf" block receives input data from the system-level logic and provides it to the OpenOFDM transmitter IP core for processing. The input data may be in the form of a serial data stream or a parallel data bus, depending on the interface requirements of the OpenOFDM transmitter IP core.

(2) Control signals: The "tx_intf" block receives control signals from the system-level logic and provides them to the OpenOFDM transmitter IP core. These control signals may include frame start and end signals, reset signals, and other control signals required for proper operation of the OpenOFDM transmitter.

(3) Data output: The "tx_intf" block provides the processed data from the OpenOFDM transmitter IP core to the system-level logic. This data may be in the form of a serial data stream or a parallel data bus, depending on the interface requirements of the system-level logic.

(4) Status signals: The "tx_intf" block provides status signals to the system-level logic indicating the status of the OpenOFDM transmitter IP core. These status signals may include signals indicating the transmitter is ready to transmit, busy transmitting, or any error conditions.

The "tx_intf" module uses the input control signals to generate control signals for the OpenOFDM transmitter IP core. It also interfaces with the input data bus to provide data to the OpenOFDM transmitter and interfaces with the output data bus to provide processed data to the system-level logic.The "tx_intf" module also generates the "busy" signal to indicate to the system-level logic that the OpenOFDM transmitter is currently processing data. This signal is used to coordinate the flow of data between the system-level logic and the OpenOFDM transmitter.

Overall, the "tx_intf" block plays a critical role in the OpenWiFi system by providing a standardized interface between the OpenOFDM transmitter IP core and the system-level logic.

XPU:

The XPU (Crossbar Processing Unit) block in the OpenWiFi project is responsible for routing and switching data between different IP blocks in the design. Specifically, the XPU block provides a crossbar switch architecture that allows multiple data sources to access multiple data destinations, while ensuring efficient data transfer and minimal latency.

The XPU block consists of several internal blocks, including the crossbar switch, arbiter, and control logic. The crossbar switch is the core component of the XPU block, and is responsible for routing data between different IP blocks based on their source and destination addresses. The arbiter is responsible for managing access to the crossbar switch, and ensures that multiple data sources do not try to access the same destination at the same time. Finally, the control logic is responsible for configuring and controlling the operation of the XPU block.

In the OpenWiFi design, the XPU block is connected to several IP blocks, including the TX/RX Interface (tx_intf/rx_intf), TX/RX MAC (tx_mac/rx_mac), and the PHY (phy_intf). The XPU block receives data from these IP blocks through different input ports, and uses the crossbar switch to route the data to the appropriate output port based on the destination address. The XPU block also manages access to the crossbar switch using the arbiter, which ensures that multiple data sources do not try to access the same destination at the same time.

The XPU (Crossbar Processing Unit) block in the OpenWiFi has several internal blocks that serve different functions.

(1) tx_control: This block is responsible for controlling the transmission of data from the TX MAC (Media Access Control) to the PHY (Physical Layer) for transmission over the air. It sets various control signals, such as the transmit enable and the modulation type, based on the configuration of the system.

(2) pxt_control: This block is similar to tx_control, but it is responsible for controlling the reception of data from the PHY to the RX MAC. It sets various control signals based on the configuration of the system, such as the receive enable and the demodulation type.

(3) rssi: This block stands for Received Signal Strength Indicator, and is responsible for measuring the strength of the received signal. It takes the received signal as input and produces a signal indicating the strength of the signal.

(4) tsf_timer: This block stands for Timing Synchronization Function Timer, and is responsible for maintaining accurate timing synchronization between the different nodes in the wireless network. It uses a time-stamping mechanism to keep track of the time of transmission and reception of packets.

(5) time_slice_gen: This block is responsible for generating time slices for the wireless network, which are used to coordinate the transmission and reception of data between different nodes. It generates the time slices based on the current state of the network and the configuration of the system.

(6) csma_ca: This block stands for Carrier Sense Multiple Access/Collision Avoidance, and is responsible for managing the access to the wireless medium to avoid collisions between different nodes. It uses a carrier sense mechanism to detect the presence of other nodes on the channel, and an exponential backoff mechanism to manage access to the channel.

(7) cca: This block stands for Clear Channel Assessment, and is responsible for detecting the presence of other nodes on the channel. It monitors the channel for a certain amount of time to determine if the channel is clear, and produces a signal indicating the result of the assessment.

(8) rx_parse: This block is responsible for parsing the received data packet to extract important information, such as the source and destination addresses, the packet length, and the data payload. It uses various algorithms to parse the packet efficiently and accurately.

Overall, these blocks serve different functions within the XPU block and are critical components in the OpenWiFi design, as they enable efficient and reliable wireless communication between different nodes in the network.

Openofdm_rx:

The OFDM_RX (Orthogonal Frequency Division Multiplexing Receiver) block in the OpenWiFi project is responsible for receiving and demodulating OFDM signals transmitted over the air. It has several internal blocks that serve different functions. Here is a brief overview of the functions of the blocks in the OFDM_RX block:

(i) RF: This block stands for Radio Frequency, and is responsible for receiving the wireless signal over the air. It uses an antenna to receive the signal and amplifies it before passing it on to the next block.

(ii) BB: This block stands for Baseband, and is responsible for processing the received signal at baseband frequency. It removes any carrier frequency and converts the signal to the baseband. It also performs channel equalization and timing synchronization to improve the quality of the signal.
(iii) FFT: This block stands for Fast Fourier Transform, and is responsible for converting the signal from the time domain to the frequency domain. It splits the signal into its individual subcarriers, which correspond to different frequencies, and calculates the amplitude and phase of each subcarrier.
(iv) Demap: This block is responsible for demapping the received signal from its modulation format, such as QPSK or 16-QAM, back into binary data.
(v) Decoder: This block is responsible for decoding the binary data and reconstructing the original transmitted data. It uses error correction techniques, such as convolutional coding and Viterbi decoding, to correct any errors that may have occurred during transmission.

The OFDM_RX block in the OpenOFDM project is implemented in Verilog and consists of several modules that perform different functions. Here is an overview of the different modules in the OFDM_RX block:

- rx_controller: This module is the top-level module for the OFDM_RX block. It is responsible for configuring and coordinating the operation of the other modules in the block.
- rx_preamble: This module is responsible for detecting and synchronizing with the preamble of the incoming OFDM signal. It performs carrier frequency offset estimation and timing synchronization to ensure that the received signal is properly aligned.
- rx_synchronization: This module is responsible for performing synchronization with the incoming OFDM signal. It includes a frequency offset estimation module and a timing recovery module.
- rx_framer: This module is responsible for framing the received data into packets. It includes a module for detecting the start and end of each packet, as well as a CRC checking module for error detection.

Overall, the OFDM_RX block in the OpenOFDM project provides a complete implementation of the OFDM physical layer, including all of the necessary modules for receiving and demodulating OFDM signals. This block is essential for the reliable operation

of the OpenWiFi wireless network, as it enables nodes to communicate with each other over the air.

rx_intf:

The rx_intf block is a part of the OpenWiFi hardware architecture and is responsible for receiving data from the physical layer of the wireless communication system. It interfaces with the receiver digital signal processing (DSP) block and extracts the received bits and channel information from the received signal.

The rx_intf block consists of several submodules, each responsible for a specific task in the reception process. These submodules include rx_cfo_est for estimating the carrier frequency offset, rx_fsync for synchronizing the receiver with the frame start marker, rx_sfd for detecting the start of the packet, rx_preamble for detecting the preamble of the packet, rx_frame_control for extracting the frame control information, rx_mac_payload for extracting the MAC payload, and rx_crc for calculating the cyclic redundancy check.

These submodules enable the rx_intf block to extract useful information from the received wireless signal, which can then be used by higher-layer protocols to interpret and act upon the data.

In the rx_intf block of the OpenWiFi hardware architecture, there are several interfaces used for communication between various submodules. These interfaces include m_axis, pl_to_m_axis, rx_iq_intf, and adc_intf.

(i) **m_axis interface**: This is a commonly used interface in digital signal processing blocks and stands for "streaming master interface with variable data width and byte enable". It is used for streaming data between modules in a digital signal processing chain. In the rx_intf block, the m_axis interface is used to stream data from the rx_mac_payload submodule to higher-layer protocol modules.
(ii) **pl_to_m_axis interface**: This is a custom interface used to transfer data from the processor logic (PL) to the m_axis interface. It includes signals for the data, byte enable, valid, and ready signals.

(iii) **rx_iq_intf interface**: This interface is responsible for receiving the in-phase (I) and quadrature-phase (Q) components of the received signal from the receiver DSP block. It includes signals for the I and Q data, as well as signals for the data valid and data ready signals.
(iv) **adc_intf interface**: This interface is used to interface with the analog-to-digital converter (ADC) that converts the received analog signal to a digital signal. It includes signals for the ADC data, as well as control signals for the ADC clock and reset.

Together, these interfaces enable the rx_intf block to communicate with other modules and submodules in the OpenWiFi hardware architecture, allowing for the efficient processing of received wireless signals.

XILINX AXIS DMA:

The Xilinx AXIS DMA (Direct Memory Access) block in the given image is a component of the OpenWiFi hardware architecture that is responsible for efficiently transferring data between the processor system (PS) and programmable logic (PL) of a Xilinx FPGA.

The AXIS DMA block is designed to support the AXI4-Stream protocol, which is used for streaming data between modules in a digital signal processing chain. It includes two main components: a data mover that transfers data between the PS and PL and a DMA engine that handles data transfers from the PL to the PS.

The AXIS DMA block interfaces with other components in the OpenWiFi hardware architecture, such as the **tx_intf** and **rx_intf** blocks, to efficiently transfer data between the PL and PS. It also includes features such as scatter-gather DMA, which allows for the efficient transfer of large amounts of data in a single operation.

Overall, the Xilinx AXIS DMA block plays an important role in the OpenWiFi hardware architecture, enabling efficient data transfer between the PS and PL and supporting the high-speed data processing required in wireless communication systems. [7]

SDR DRIVER:

The SDR driver in the OpenWiFi hardware architecture, as shown in the given image, is responsible for providing software control and access to the OpenWiFi system. It provides a software interface that allows the system to be controlled and configured by higher-level software, such as a wireless communication protocol stack.

The SDR driver communicates with the OpenWiFi hardware architecture through a low-level interface, such as a PCIe or USB connection. It is typically implemented as a kernel driver or library that is loaded into the operating system and provides an API for controlling and configuring the OpenWiFi system.

The SDR driver provides a number of functions and features, including:

(i) Device initialization and configuration: The driver is responsible for initializing the OpenWiFi hardware and configuring its various components, such as the ADC and DAC interfaces, the RF front-end, and the digital signal processing modules.

(ii) Data transfer and synchronization: The driver provides functions for transferring data between the OpenWiFi hardware and the host system, ensuring proper synchronization and buffering of data.

(iii) Control and monitoring: The driver allows for the control and monitoring of various aspects of the OpenWiFi system, including the RF power levels, modulation schemes, and other system parameters.

(iv) Debugging and testing: The driver provides tools and features for debugging and testing the OpenWiFi system, including the ability to capture and analyze data in real-time.

The SDR driver plays a critical role in enabling the control and configuration of the OpenWiFi hardware architecture, allowing for the development and deployment of custom wireless communication systems.

In the OpenWiFi hardware architecture, the SDR driver provides access to several other drivers and modules that are used to control and configure the various components of the system. These drivers and modules include:

(i) **rx_intf driver**: This driver provides access to the rx_intf block, which is responsible for interfacing with the RF front-end and performing the baseband processing required for receiving wireless signals.

(ii) **openofdm_rx driver**: This driver provides support for the OpenOFDM wireless communication protocol stack, which is used for demodulating and decoding wireless signals received by the OpenWiFi system.

(iii) **xilinx_dma driver**: This driver provides access to the Xilinx AXI DMA (Direct Memory Access) controller, which is used for high-speed data transfer between the FPGA and the host system.

(iv) **xpu drive**r: This driver provides access to the Xilinx Processing System Unit (PSU), which is responsible for managing the processing and communication functions of the OpenWiFi system.

(v) **openofdm_tx driver**: This driver provides support for the OpenOFDM wireless communication protocol stack, which is used for modulating and encoding wireless signals transmitted by the OpenWiFi system.

(vi) **tx_intf driver**: This driver provides access to the tx_intf block, which is responsible for interfacing with the RF front-end and performing the baseband processing required for transmitting wireless signals.

(vii) **ad9361 driver**: This driver provides support for the AD9361 RF transceiver, which is used as the RF front-end in the OpenWiFi system.

These drivers and modules work together to provide the necessary control and configuration of the OpenWiFi hardware architecture, allowing for the development and deployment of custom wireless communication systems.The xilinx_dma driver, in particular, provides high-speed data transfer capabilities between the FPGA and the host system, which is crucial for real-time wireless communication applications. The driver is implemented using Xilinx's AXI DMA IP core, which is a configurable and flexible DMA engine that can be customized to meet the specific needs of the application. The driver provides an API for configuring and controlling the AXI DMA engine, allowing for efficient and reliable data transfer between the FPGA and the host system.

LINUX MAC 802.11 SUBSYSTEM:

The Linux MAC 802.11 subsystem is a part of the Linux kernel that provides support for wireless networking protocols, including the IEEE 802.11 family of standards. This subsystem includes a set of interfaces and data structures that allow for the implementation of various wireless networking features, such as scanning, association, authentication, and transmission/reception of wireless frames.

The ieee80211_ops structure is a key component of the Linux MAC 802.11 subsystem. It is a set of function pointers that define the interface between the kernel and the wireless device driver. These function pointers provide the necessary hooks for the kernel to communicate with the driver and configure various aspects of the wireless hardware, such as setting the channel, modulation scheme, power level, and other parameters.

In the OpenWiFi project, the ieee80211_ops structure is used to implement the necessary functions for controlling the OpenWiFi hardware architecture. These functions include configuring the wireless hardware, initiating transmission and reception of wireless frames, and performing other low-level operations required for wireless communication.

Overall, the **ieee80211_ops** structure serves as a bridge between the Linux kernel and the OpenWiFi hardware, allowing for the development of custom wireless communication systems using the Linux operating system. [6]

3.3. KERNEL DEVELOPMENT

Kernel:

The kernel is the essential foundation of a computer's operating system (OS). It is the core that provides basic services for all other parts of the OS. It is the main layer between the OS and underlying computer hardware, and it helps with tasks such as process and memory management, file systems, device control and networking.

During normal system startup, a computer's basic input/output system, or BIOS, completes a hardware bootstrap or initialization. It then runs a bootloader which loads the kernel from a storage device -- such as a hard drive -- into a protected memory space. Once the kernel is loaded into computer memory, the BIOS transfers control to the kernel. It then loads other OS components to complete the system startup and make control available to users through a desktop or other user interface.

Kernel Layout:-

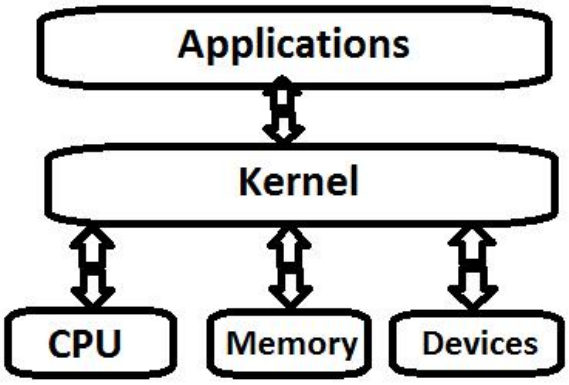

Fig 3.3 Kernel Layout

Tasks By Kernel:-

1)Loading and managing less-critical OS components, such as device drivers;

2)Organizing and managing threads and the various processes spawned by running applications;

3)Scheduling which applications can access and use the kernel, and supervising that use when the scheduled time occurs;

4)Deciding which nonprotected user memory space each application process uses;

5)Handling conflicts and errors in memory allocation and management;

6)Managing and optimizing hardware resources and dependencies, such as central processing unit (CPU) and cache use, file system operation and network transport mechanisms;

7)Managing and accessing input/output devices such as keyboards, mice, disk drives, USB ports, network adapters and displays; and

8)Handling device and application system calls using various mechanisms such as hardware interrupts or device drivers.

Types of Kernel:-

Kernels fall into three architectures: monolithic, microkernel and hybrid. The main difference between these types is the number of address spaces they support.

1) A microkernel delegates user processes and services and kernel services in different address spaces.

2) A monolithic kernel implements services in the same address space.

3) A hybrid kernel, such as the Microsoft Windows NT and Apple XNU kernels, attempts to combine the behaviors and benefits of microkernel and monolithic kernel architectures.

The Linux kernel is a monolithic kernel.These main groups include a system call interface, process management, network stack, memory management, virtual file system and device drivers.

Monolithic Kernels:-

Monolithic kernels are larger than microkernels, because they house both kernel and user services in the same address space. Monolithic kernels use a faster system call communication protocol than microkernels to execute processes between the hardware and software. They are less flexible than microkernels and require more work; admins must reconstruct the entire kernel to support a new service.

Monolithic kernels pose a greater security risk to systems than microkernels because, if a service fails, then the entire system shuts down. Monolithic kernels also don't require as much source code as a microkernel, which means they are less susceptible to bugs and need less debugging**.**

Need of Kernel:

Kernel mode refers to the processor mode that enables software to have full and unrestricted access to the system and its resources. The OS kernel and kernel drivers, such as the file system driver, are loaded into protected memory space and operate in this highly privileged kernel mode.

As User mode applications are less privileged and cannot access system resources directly. Instead, an application running in user mode must make system calls to the kernel to access system resources. The kernel then acts as a manager, scheduler and gatekeeper for those resources and works to prevent conflicting resource requests.[10]

Process Scheduling and Management:-

Scheduling and **management** are central to the kernel's operation. Computer hardware can only do one thing at a time. However, a computer's OS components and applications can spawn dozens and even hundreds of processes that the computer must host. It's impossible for all of those processes to use the computer's hardware -- such as a memory address or CPU instruction pipeline -- at the same time. The kernel is the central manager of these processes. It knows which hardware resources are available and which processes need them. It then allocates time for each process to use those resources.

Process Scheduling is an important activity performed by the process manager of the respective operating system.

Scheduling in Linux deals with the removal of the current process from the CPU and selecting another process for execution.

Process Scheduler chooses a process to be executed.

Process scheduler also decides for how long the chosen process is to be executed.

In LINUX operating system, we have mainly two types of processes namely - Real-time Process and Normal Process. Let us learn more about them in detail.

Realtime Process

Real-time processes are processes that cannot be delayed in any situation. Real-time processes are referred to as urgent processes.

There are mainly two types of real-time processes in LINUX namely:

SCHED_FIFO

SCHED_RR.

A real-time process will try to seize all the other working processes having lesser priority.

SCHED_FIFO

FIFO in SCHED_FIFO means First In First Out. Hence, the SCHED_FIFO policy schedules the processes according to the arrival time of the process.

SCHED_RR

RR in SCHED_RR means Round Robin. The SCHED_RR policy schedules the processes by giving them a fixed amount of time for execution. This fixed time is known as time quantum.

Normal Processes:-

Normal Processes are the opposite of real-time processes. Normal processes will execute or stop according to the time assigned by the process scheduler. Hence, a normal process can suffer some delay if the CPU is busy executing other high-priority processes.

Normal (SCHED_NORMAL or SCHED_OTHER)

SCHED_NORMAL / SCHED_OTHER is the default or standard scheduling policy used in the LINUX operating system. A time-sharing mechanism is used in the normal policy. A time-sharing mechanism means assigning some specific amount of time to a process for its execution. Normal policy deals with all the threads of processes that do not need any real-time mechanism.

Batch (SCHED_BATCH)

As the name suggests, the SCHED_BATCH policy is used for executing a batch of processes. This policy is somewhat similar to the Normal policy. SCHED_BATCH policy deals with the non-interactive processes that are useful in optimizing the CPU throughput time. SCHED_BATCH scheduling policy is used for a group of processes having priority: 0.

Idle (SCHED_IDLE)

SCHED_IDLE policy deals with the processes having extremely Low Priority. Low priority tasks are the tasks that are executed when there are absolutely no tasks to be executed. SHED_IDLE policy is designed for the lowest priority tasks of the operating systems.

Process Management:-

A process is a program (object code stored on some media) in the midst of execution. Processes are, however, more than just the executing program code (often called the text

section in Unix).They also include a set of resources such as open files and pending signals, internal kernel data, processor state, a memory address space with one or more memory mappings, one or more threads of execution, and a data section containing global variables. Processes, in effect, are the living result of running program code.The kernel needs to manage all these details efficiently and transparently.

Flow Chart Of Process States:-

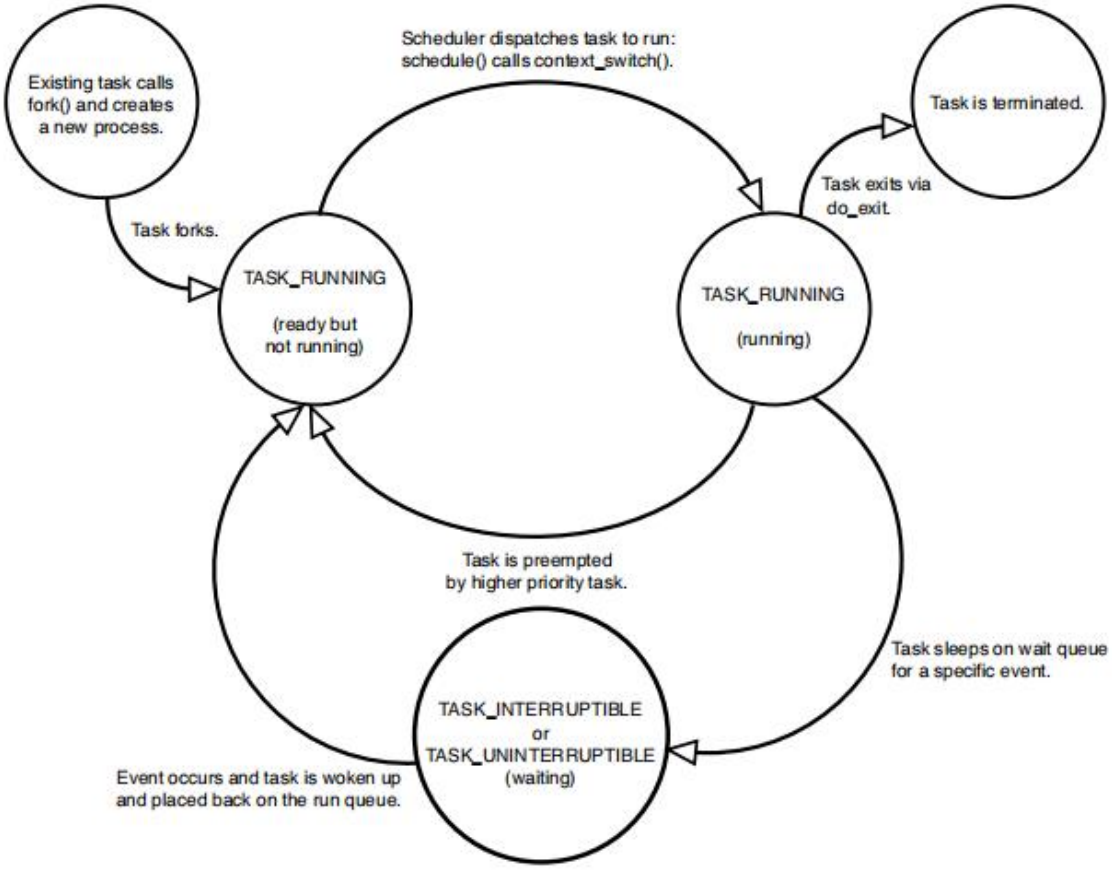

Fig 3.4 Flow chart of Process States [8]

System Calls:-

System calls provide a layer between the hardware and user-space processes.This layer serves three primary purposes. First, it provides an abstracted hardware interface for userspace.When reading or writing from a file, for example, applications are not concerned with the type of disk, media, or even the type of filesystem on which the file resides. Second, system calls ensure system security and stability.With the kernel acting as a middleman

between system resources and user-space, the kernel can arbitrate access based on permissions, users, and other criteria.

Kernel Development Process:

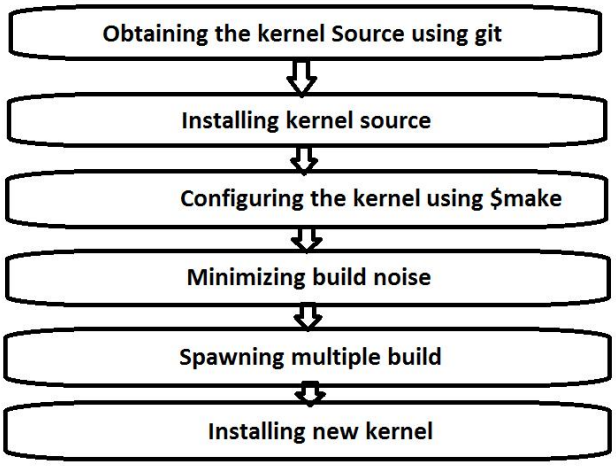

Fig 3.5 step wise process of kernel development

3.4 TRANSCEIVER

Transceiver:

In radio communication, a transceiver is an electronic device which is a combination of a radio transmitter and a receiver, hence the name. It can both transmit and receive radio waves using an antenna, for communication purposes.

SDR:

Over the past decade, modern wireless communication systems witnessed a new era of high data rates. The number of users of mobile devices has increased significantly. Different standards have been implemented not only to achieve high data rates, but also to compromise between area, power, quality and the large number of users. Each standard has its own transceiver which results in large area utilization and power consumption affecting the battery lifetime. The SDR is a proposed solution to tackle these challenges SDR is a radio communication system whose components are implemented by means of software using Universal Software Radio Peripheral (USRP) and GNU Radio. SDR not only provides cheap and flexible multi-standard-terminals for end users, but also offers a physical layer platform where functions are software defined. Dynamic Partial Reconfiguration (DPR) has proved itself in implementing SDR system on Field Programmable Gate Arrays (FPGAs). This work

implements the physical layer of Wi-Fi transceiver chain on Xilinx Zynq board supporting DPR flow. Data is modulated via real communication channel using the USRP RF section connected to the FPGA. A new approach is proposed to enable running the DPR flow and USRP application simultaneously on the same FPGA. USRP has become a popular platform for hardware-based research in the field of SDR and cognitive radio. Integrating the GNU Radio software with the USRP enables the user to build a complete open software radio system that supports host-based signal processing on any platform. The internal FPGA implemented inside the USRP kit is used to communicate with the PC for implementing extra functionalities.

SDR SYSTEM OVERVIEW :

The USRP Software Defined Radio Device is a tunable transceiver for prototyping wireless communication systems. It offers frequency ranges up to 6 GHz with instantaneous bandwidth up to 56 MHz. The USRP Hardware Driver (UHD) is connected to USB 3.0 whose data rate is up to 5Gbps. UHD is a hardware driver library serving all types USRPs and daughter-boards by providing unique APIs for software radio implementation. The driver can be standalone, or with third party applications such as: GNU Radio, Labview, and .Simulink

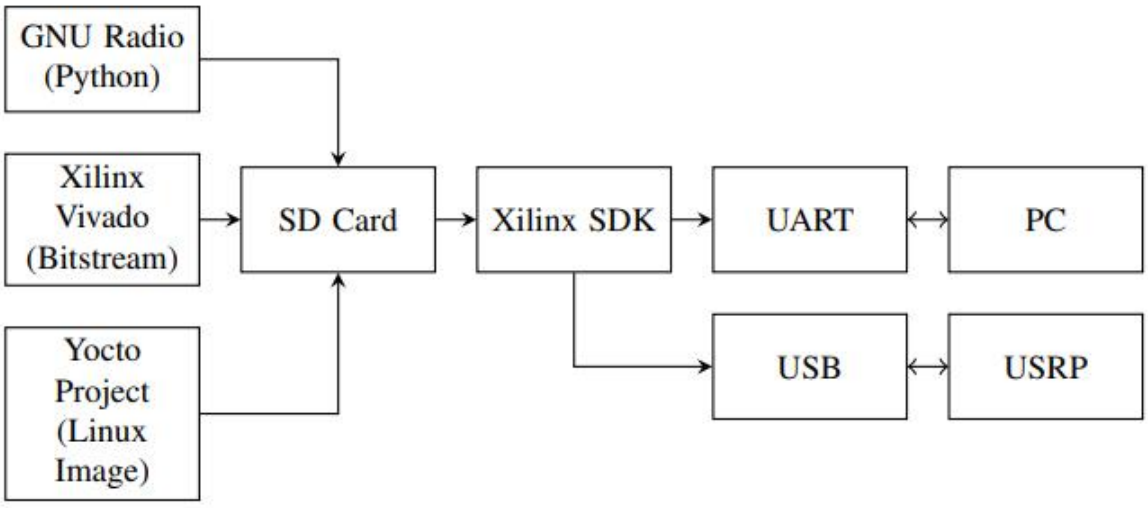

Fig 3.6: System peripherals [9]

Figure 3.6 shows the overall system implementation. The connection between the USRP kit and Zynq FPGA is through USB cable. The GNU Radio provides a soft model in the form of flow graph that enables configuring the USRP easily. Compiling the flow graph generates a python code that is executed from the Linux image through UART terminal.

The whole design flow is performed using Xilinx Vivado. The process invoked simulating the RTL design of the Wi-Fi transceiver, synthesis, placing, routing, and eventually bit stream file generation to program each FPGA. After building and configuring the Linux image, Xilinx SDK manages the following tasks:

- Configuring the PL side with the bitstream files generated by Xilinx Vivado. This is performed using a C-application that calls reserved APIs for configuring the PL side and the Partial Reconfiguration Controller PRC used to control the ICAP to support the DPR flow.
- Creating the bootable image from the Linux boot loader generated by the Yocto project.
- Propagating the test data through the Wi-Fi transceiver to generate the modulated symbols. Data is transferred from the DDR to the PL side through DMAs
- Executing the python application generated by the GNU Radio to configure the USRP.

The python application generated by the GNU Radio transmits the modulated symbols from the implemented Wi-Fi transmitter on the Zynq board to the USRP RF section as illustrated in Figure 3.7. The RF section uses Gaussian Minimum Shift Keying (GMSK) technique to transmit the data on high frequency range to the receiving antenna. The GNU Radio at the receiver side regenerates the modulated symbols and transmits them to the implemented Wi-Fi receiver on the other Zynq board.

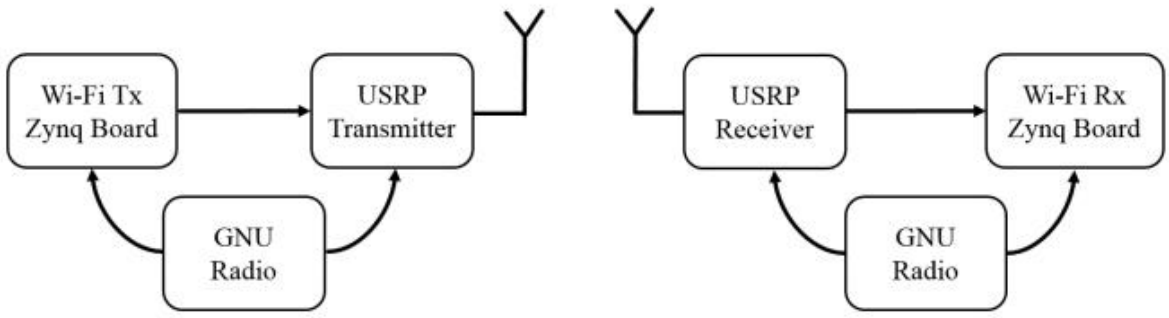

Fig 3.7: Transceiver block diagram [9]

WiFi TRANSCEIVER DESIGN:

The transmitter includes all blocks from scrambler to preamble. The rest are the receiver chain. Scrambler is responsible for randomizing the MAC layer data in order to prevent the presence of long 1s or 0s sequences. Convolutional encoder with coding rate equals to 1/2 is used for data replication in order to decrease the bit error rate (BER) and enable the decoder to deduce the correct transmitted bits.

Puncturing technique of coding rate 3/4 is used to stealing specific encoded bits to increase the coding rate. Interleaver is used to get rid of burst errors by rearranging the data bits. Symbols are then modulated using 16-QAM modulation scheme. Since Wi-Fi is Orthogonal Frequency Division Multiplexing (OFDM) based, the 128-point IFFT is used to modulate the symbols on several sub-carriers instead of single carrier in order to reduce the channel fading. The cyclic prefix is added to eliminate the Inter Symbol Interference (ISI).

The packet divider receives the modulated symbols and removes the reserved preamble bits from the stored data. The demapper specifies the decision region of the received real and imaginary symbols from the FFT, then converts the symbols to a stream of bits. Deinterleaving process is used to repeal the effect of the interleaver at the transmitter side. The depuncture pads dummy bits in the position of the removed punctured bits. Viterbi decoder is used because of its ability for error detection and correction. Descrambler uses the same equation used by the scrambler in the transmitter.

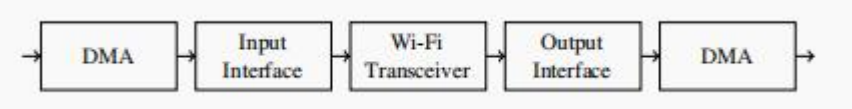

Fig 3.8. Test Environment on the PL side [9]

Figure 3.8 shows the established test environment on the PL side.

1) The input data is transferred from the DDR memory to be stored in the Direct Memory Access (DMA) that adjusts the rate according to the clock of each wireless communication system.

2) An intermediate block “Input Interface” is used to adjust the data input rate and system reset.

3) data is transferred through the system.

4) another intermediate block “Output Interface” is used to adjust output data rate for the DMA.

5) the output data is stored in a second DMA to be finally transferred to the DDR for verification.

USRP AND FPGA INTERFACING:

Xilinx Zynq-7000 kit provides two Cortex-A9 processors that share common memory and other peripherals. Asymmetric multiprocessing (AMP) mechanism allows both processors to concurrently run independent OS or bare-metal applications allowing each one to communicate with the other through a shared memory called On-Chip Memory (OCM).

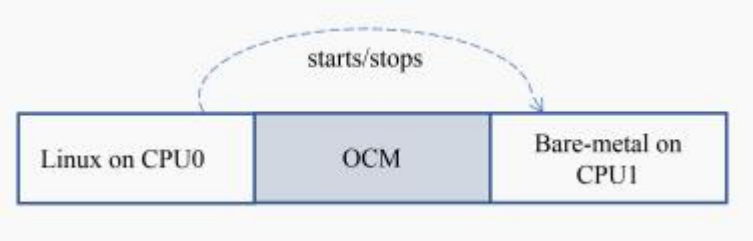

Fig 3.9. AMP System configuration [9]

As illustrated in Figure 3.9, the Linux image launched on CPU0, the system master is responsible for the following:

- Initializing the system.
- Controlling CPU1 startup.
- Communicating with CPU1 through the OCM.
- Running USRP applications using the USRP Hardware Driver (UHD) and GNU Radio installed on the file system.

• Interacting with the user through the UART terminal. The user has the ability to choose to configure the system or testing it.

The bare-metal application running on CPU1 generated by compiling the C-application that uses the DPR APIs is responsible for:

• loading the bitstream files to configure the PL side.

• managing the test data for Wi-Fi transceiver. This is performed by propagating the data from the SD card to the DDR then to the PL side through the DMA.

• communicating with CPU0 through OCM.

Synchronization between the two processors is being deployed in order to prevent conflicts on shared Hardware resources such as: OCM, global timer, and L2 cache.

The AMP system is divided into three software sections:

1) Xilinx native First stage boot loader (FSBL).

2) The Linux OS and USRP operating on CPU0.

3) The Bare-metal OS and SDK operating on CPU1.

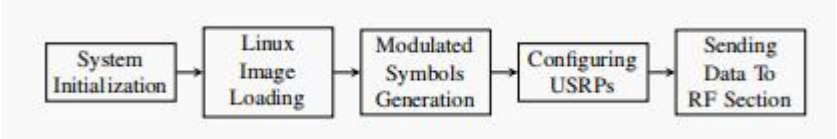

Fig 3.10. System initialization procedures [9]

According to Figure 3.10, the following procedures are being performed:

•**System Initialization**: Initially, after power on, the FSBL is launched on CPU0 to reset the PS system and program the PL side using the bitstream file. The FSBL is also responsible for loading the ELF application from the DDR and executing it.

• **Linux Image Loading**: The Linux image is loaded on CPU0 which starts CPU1.

• **Modulated Symbols Generation**: The bare-metal on CPU1 loads the ELF file on the SD card to pass the Wi-Fi test data to the chain and generate the modulation symbols.

• **Configuring USRPs**: The python application is launched on the Linux image through the UART terminal to configure the USRP kit.

• **Sending Data To RF Section**: Eventually, The USRP sends the modulated data through the RF section.

CHAPTER 4
RESULTS AND DISCUSSION

We studied and analyzed the standard wifi protocol IEEE 802.11 and analyzed its various formats i.e., IEEE 802.11 a/b/n/g. We studied the IEEE 802.11b protocol in detail and studied its packet format and its significance in Physical layer and MAC layer.The study of IEEE 802.11b involves understanding the technical specifications and performance characteristics of the standard. This includes knowledge of the modulation and encoding techniques used, the channel bandwidth and frequency range, and the data rate and range limitations.One of the key benefits of IEEE 802.11b is its compatibility with other 802.11 standards. This allows for easy integration with other wireless networks, and it has helped to establish IEEE 802.11b as a widely adopted standard for wireless networking.It operates in the crowded 2.4 GHz frequency band, which can lead to interference and reduced performance in some environments. The maximum data rate of 11 Mbps may not be sufficient for some applications that require high-speed data transfer.

The study of the OpenWiFi project involves understanding the technical specifications and performance characteristics of the hardware and software components. This includes knowledge of the hardware design and architecture used for communication between devices.One of the key benefits of the OpenWiFi project is its open-source nature. This allows for easy customization and modification of the hardware and software components to suit specific requirements or applications. The OpenWiFi project also supports a wide range of wireless communication standards, including IEEE 802.11a/b/g/n/ac/ax, which makes it a versatile and flexible solution for different applications and environments.

CHAPTER 5
SUMMARY AND CONCLUSIONS

The OpenWiFi project is an open-source initiative project that provides a hardware-agnostic WiFi stack for software-defined radios (SDRs). The project enables researchers and developers to experiment with custom WiFi protocols and algorithms on off-the-shelf SDRs, offering a valuable resource for the research and development of WiFi technologies.

The project's framework for implementing PHY and MAC layers of WiFi protocols supports a variety of SDR platforms, including USRP, LimeSDR, and PlutoSDR, and includes a set of tools and utilities for configuring and testing WiFi networks. Its open-source nature has enabled a community of contributors to develop and enhance the project continuously, and its hardware-agnostic design has made it accessible to a wide range of users.

Hardware implementation of the OpenWiFi project requires the use of SDRs that support the project's framework for implementing PHY and MAC layers of WiFi protocols. The project supports a variety of SDR platforms, including USRP, LimeSDR, and PlutoSDR, but it may require some hardware modifications or adaptations depending on the specific SDR used.The success of hardware implementation of the OpenWiFi project depends on the availability and compatibility of the chosen SDR platform, as well as the skills and experience of the user. However, with proper hardware selection and configuration, the OpenWiFi project provides a valuable resource for the research and development of WiFi technologies.

In conclusion, the OpenWiFi project offers a hardware-agnostic WiFi stack for SDRs that has significant potential for advancing the state of the art in wireless communications research and development. The success of hardware implementation depends on the availability and compatibility of the chosen SDR platform and the skills and experience of the user.

RECOMMENDATIONS FOR THE FUTURE WORK

(i) Future work should focus on improving the throughput, latency, and energy efficiency of the WiFi stack. This can be achieved through the development of new algorithms, protocols, and hardware acceleration techniques.

(ii) Future work should explore new security mechanisms and techniques to address emerging security threats such as hacking, eavesdropping, and data theft.

(iii) Future work should focus on interoperability testing with different devices and platforms to ensure that the WiFi stack works seamlessly with a variety of devices.

(iv) Future work should explore integration with projects such as GNU Radio, OpenAirInterface, and OpenLTE to provide a comprehensive platform for wireless communications research and development.

Overall, the OpenWiFi project has significant potential for future work and improvement, and its open-source nature enables a community of contributors to develop and enhance the project continuously. By focusing on these recommendations, the OpenWiFi project can continue to make significant contributions to the field of wireless networking research and development.

REFERENCES

[1] IEEE,"openwifi: a free and open-source IEEE802.11 SDR implementation on SoC" by Xianjun Jiao; Wei Liu;IEEE 91st Vehicular Technology Conference (VTC2020-Spring) (2020)

[2] "OpenRadio: A Linux-based Open Source SDR Platform for Education and Research" by Franco Frattolillo, Christian Vitale, Marco Gramaglia, et al. (2020)

[3] "OpenWiFi: A Linux-based Open-source Framework for Experimentation with WiFi Networks" by Christian Vitale, Marco Gramaglia, Xavier Costa-Pérez, et al. (2019)

[4] "Wireless Communications: Principles and Practice" by Theodore S. Rappaport;2001;Prentice Hall Communications Engineering and Emerging Technologies Series; 2nd edition

[5] Tektronix, "Wi-Fi: Overview of the 802.11 Physical Layer and Transmitter Measurements," Primer.

[6] OpenWi-Fi, "OpenWi-Fi: An Open Source IEEE 802.11 Reference Design," GitHub. URL: https://github.com/open-sdr/openwifi

[7] OpenWi-Fi, "OpenWi-Fi: An Open Source IEEE 802.11 Reference Design - Hardware," GitHub. URL: https://github.com/open-sdr/openwifi-hw.

[8] R. Love, "Linux Kernel Development Book", 3rd ed., Addison Wesley, 2010.

[9] Khadija Khaled, Chaymaa Osama,"Interfacing USRP Kit with Zynq-7000 Evaluation Kit",in 8th International Conference on Modern Circuits and Systems Technologies (MOCAST);2019

[10] Linux Foundation. (n.d.). A Beginner's Guide to Linux Kernel Development (LFD103). URL:https://training.linuxfoundation.org/training/a-beginners-guide-to-linux-kernel-development-lfd103/

[11] Introduction to Linux. URL: https://training.linuxfoundation.org/training/introduction-to-linux/