CAIRO UNIVERSITY

FACULTY OF ENGINEERING

ELECTRONICS & COMMUNICATIONS ENGINEERING DEPARTMENT

Optimizing the ZigBee Transceiver by Designing a Neural Network in the Rx and Designing the AES Encryption System

A Graduation Project Report Submitted to the Faculty of Engineering at Cairo University in Partial Fulfillment of the Requirements for the Degree of Bachelor in Electronics and Communications Engineering

By

Sherif Mahmoud said

Abdullah Mahmoud Abdella

Abdelfatah Sayed Abdelfatah

Saad Morsey Saad Elnaggar

Maya Medhat Mahmoud

Zyad Ahmed Fawzy

Under Supervision of

Dr. Rania Osama

Acknowledgment

We are incredibly fortunate to have had the support and advice of so many people during the course of our project, as their success and ultimate result depended heavily on it. We would not forget to thank them for their oversight and aid, without which we could not have accomplished anything. We would like to thank them from the bottom of our hearts for their patient advice, passionate support, and insightful criticism of this project effort. We appreciate the Department of Electronics and Communication Engineering, Cairo university for overseeing our project and giving us the chance to gain new knowledge and DR. Rania for her support. Lastly, we would want to express our gratitude for all of the love and encouragement we received from our families and coworkers.

Abstract

ZigBee technology has emerged as a prominent solution for low-power wireless communication in various applications, including home automation, industrial monitoring, healthcare, and smart agriculture. This paper presents the design and implementation of a Zigbee transmitter and receiver system aimed at providing reliable and energy-efficient communication between nodes in a Zigbee network, the Zigbee transmitter module is equipped with robust error correction and data encoding techniques to ensure reliable data transmission while minimizing power consumption, On the receiver side, we use a Recurrent Neural Networks (RNNs) as decoder to enhance signal processing and demodulation accuracy in noisy environments complementary The Zigbee transmitter and receiver modules are implemented and evaluated in real-world scenarios to assess their performance in terms of communication range, data throughput, and energy efficiency. Additionally, it examines the implementation of the Advanced Encryption Standard (AES) for robust security, ensuring data integrity and confidentiality. By integrating RNNs and AES, Zigbee can achieve superior performance and security, addressing both current and future challenges in wireless communication

Table of Contents

List of figures

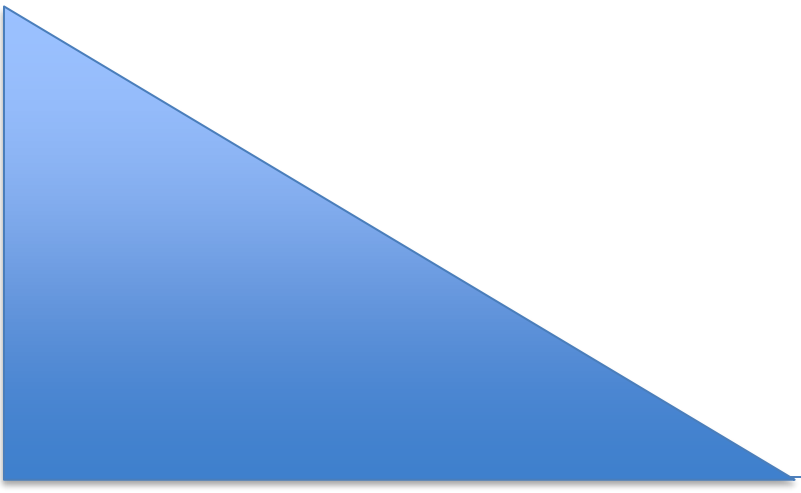
List of tables

List of abbreviations

Abbreviation	Definition
ADC	Analog digital converter
AES	Advanced Encryption Standard
API	Application Programming Interference
APL	Application Layer
ANN	Artificial Neural Network
CSA	Carry Save Adders
DSSS	Direct Spread spectrum Sequencing
D2D	Device to Device
D2TC	Device-to-Trust center
FIFO	First in first out
LPF	Low pass filter
LSTM	Long short-term memory
LUT	Lock Up Tables
MAC	Media Access Control
MLME	MAC sub–Layer Management Entity
MPDU	MAC protocol Data Units
MSE	Mean Square Error
NIST	National Institute of Standards and Technology
OSI	Open System Interconnection
O_QPSK	Offset Quadrature phase shift keying
PN	pseudo–Random Noise
RCON	Round constant
RNN	Recurrent Neural Network
UHV	Ultra-High Voltage

Chapter 1: Introduction

- **ZigBee Definition**
- **ZigBee Protocols Stack**
- **Motivation Of Work**
- **Applications of ZigBee**
- **Security**
- **Neural network**



1.1 ZigBee Definition

ZigBee is a standard that defines a set of communication protocols for low-data-rate short-range wireless networking ZigBee-based wireless devices operate in 868 MHz, 915 MHz, and 2.4 GHz frequency bands. The maximum data rate is 250 K bits per second. ZigBee is targeted mainly for battery-powered applications where low data rate, low cost, and long battery life are main requirements. In many ZigBee applications, the total time the wireless device is engaged in any type of activity is very limited; the device spends most of its time in a power-saving mode, also known as sleep mode. As a result, ZigBee enabled devices are capable of being operational for several years before their batteries need to be replaced [1].

1.2 ZigBee Protocols Stack

Protocol architecture is based on Open system interconnection (OSI). ZigBee builds on IEEE standard 802.15.4 which defines the physical and media access control (MAC) layers. ZigBee alliance defines the network layer and application layer [2].

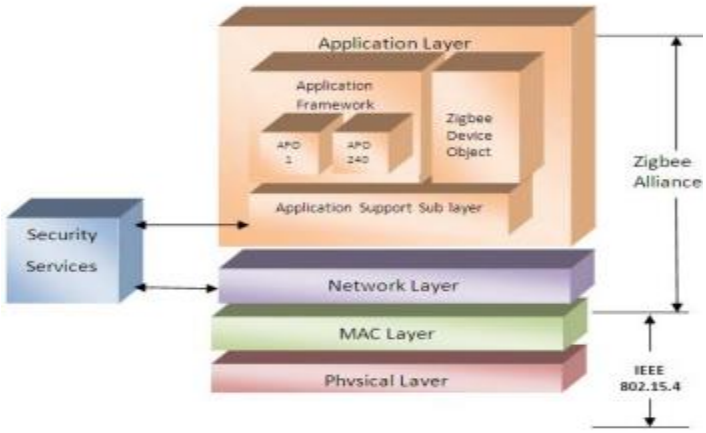


Figure 1.2.1: protocol stack of ZigBee

1.2.1 physical layer

The physical layer of the IEEE802.15.4 standard is the closest layer to the hardware, which controls and communicates with the radio transceiver directly. It handles all tasks involving the access to the ZigBee hardware, including initialization of the hardware, channel selection, link quality estimation, energy detection measurement and clear channel assessment to assist the channel selection. Supports three frequency bands, 2.45GHz band which using16 channels, 915MHz band which using 10 channels and 868MHz band using 1 channel. All three using Direct Spread Spectrum Sequencing (DSSS) access mode [3].

PHY Packet Fields

• Preamble (32 bits) – synchronization

• Start of Packet Delimiter (8 bits)

• PHY Header (8 bits) – PSDU length

• PSDU (0 to 1016 bits) – Data field

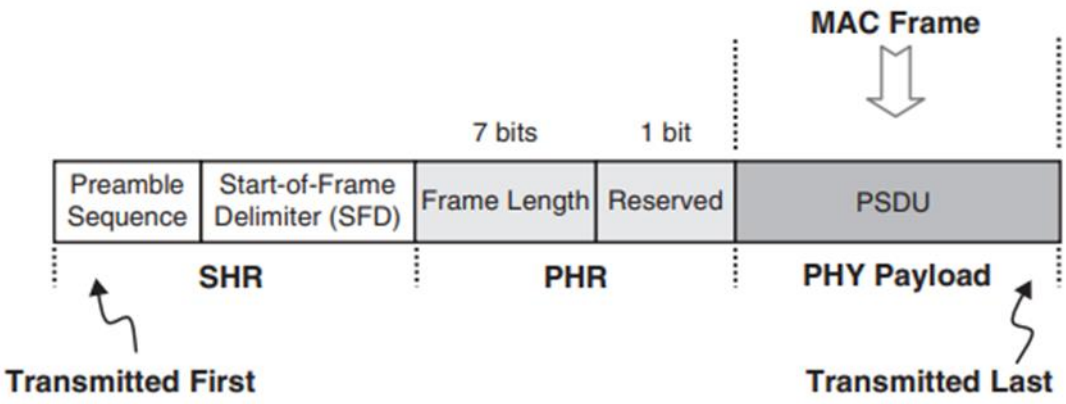


Figure 1.2.1.1: PPDU Format

1.2.2 MAC layer

This layer provides an interface between the physical layer and network layer. This provides

two services; MAC data services and MAC management service interfacing to the MAC sub–Layer Management Entity (MLME) Service Access Point called (MLME-SAP). The MAC data service enables the transmission and reception of MAC protocol Data Units (MPDUs) across the PHY data service. The MAC layer is responsible for generating beacons and synchronizing devices to the beacon signal in a beacon enabled service. It is also performing association and dissociation functions. It defines four frame structures: Beacon frame, Data frame, Acknowledge frame, MAC command frame. Basically, there are two types of topologies; star and peer to peer. Peer to peer topology can take different shapes depending on its restrictions. Peer to peer is known as mesh, if there is no restriction. Another form is tree topology. Interoperability is one of the advantages of the ZigBee protocol stack. ZigBee has a wide range of applications, so different manufacturers provide ZigBee devices. ZigBee devices can interact with each other regardless of manufacturer (even if the message is encrypted).

1.2.3 Network layer

Network layer interfaces between application layer and MAC Layer. This Layer is responsible for network formation and routing. Routing is the process of selection of path to relay the messages to the destination node. This forms the network involving joining and leaving of nodes, maintaining routing tables (coordinator/router), actual routing and address allocation. ZigBee coordinator or router will perform the route discovery. This layer Provides network wide security and allows low power devices to maximize their battery life. From the basic topologies, there are three network topologies are considered in IEEE802.15.4 are star, tree Network and mesh.

1.2.4 Application layer

The application (APL) layer is the highest protocol layer in the ZigBee wireless network and hosts the application objects. Manufacturers develop the application objects to customize a device for various applications. Application objects control and manage the protocol layers in a ZigBee device. There can be up to 240 application objects in a single device. The ZigBee standard offers the option to use application profiles in developing an application. An application profile is a set of agreements on application-specific message formats and processing actions. The use of an application profile allows further interoperability between the products developed by different vendors for a specific application. If two vendors use the

same application profile to develop their products, the product from one vendor will be able to interact with products manufactured by the other vendor as though both were manufactured by the same vendor.

1.3 Motivation Of Work

The motivation behind Zigbee technology stems from the need for reliable, low-power wireless communication solutions that can support various applications in diverse industries. Several key factors have contributed to the development and adoption of Zigbee:

1_**Low-Power Operation**: Zigbee was designed to operate on low-power devices, making it ideal for battery-operated sensors and actuators in applications such as home automation, industrial monitoring, and smart agriculture. Its efficient use of energy enables long battery life, reducing the need for frequent battery replacements and maintenance.

2_**Wireless Mesh Networking**: Zigbee supports mesh networking topology, allowing devices to communicate with each other through multiple hops. This feature enhances network reliability and coverage, as messages can be relayed through neighboring nodes, thereby extending the communication range and improving network robustness.

3_I**nteroperability**: Zigbee standards ensure interoperability among devices from different manufacturers, enabling seamless integration and compatibility within Zigbee networks. This interoperability fosters ecosystem growth and allows for the development of diverse applications and products that can communicate with each other seamlessly.

4_**Cost-Effectiveness:** Zigbee offers a cost-effective solution for implementing wireless communication networks, especially in large-scale deployments. Its low-cost hardware components and simplified network infrastructure contribute to overall cost savings in both deployment and maintenance, making it attractive for various commercial and industrial applications.

5_**Scalability:** Zigbee networks can scale from a few devices to hundreds or even thousands of nodes, depending on the application requirements. This scalability enables Zigbee technology to accommodate a wide range of deployment scenarios, from simple home automation

setups to complex industrial control systems.

6_**Reliability and Security:** Zigbee incorporates robust error detection and correction mechanisms to ensure reliable data transmission, even in noisy and interference-prone environments. Additionally, Zigbee standards include built-in security features such as encryption and authentication, safeguarding data privacy and protecting against unauthorized access and malicious attacks.

7_**Flexibility and Adaptability**: Zigbee is a versatile technology that can be customized and optimized for specific applications and use cases. Its flexibility allows for the implementation of various communication profiles, data rates, and network topologies to meet the diverse needs of different industries and applications.

1.4 Applications of ZigBee

1.4.1 Home patient monitoring:

A patient's blood pressure and heart rate, for example, can be measured by wearable devices. The patient wears a ZigBee device that interfaces with a sensor that gathers health-related information such as blood pressure on a periodic basis. Then the data is wirelessly transmitted to a local server, such as a personal computer inside the patient's home, where initial analysis is performed. Finally, the vital information is sent to the patient's nurse or physician via the Internet for further analysis [4]

1.4.2 Monitoring the structural health of large scale buildings :

In this application, several ZigBee-enabled wireless sensors (e.g., accelerometers) can be installed in a building, and all these sensors can form a single wireless network to gather the information that will be used to evaluate the building's structural health and detect signs of possible damage. After an earthquake, for example, a building could require inspection before it reopens to the public. The data gathered by the sensors could help expedite and reduce the cost of the inspection.

1.4.3 Fire Emergency Applications:

The University of California at Berkeley developed a project called Fire Information and Rescue Equipment (FIRE, 2005) to improve fire-fighting equipment. It can protect firefighters and help to relieve victims. Fire fighters are equipped with a Telos mote, a wearable computer, and a fireproof mask called FireEye. Wireless smoke and temperature sensors are deployed in a building to monitor the environment. When a fire emergency occurs, wireless sensors can provide real-time emergency-related information, which can be shown on the FireEye. The FireEye mask can also show other firefighters' locations. The Telos sensor monitors a fire fighter's heart rate and air tank level and propagates the information to other firefighters [5].

1.4.4 ZigBee Technology for UHV Transmission Lines:

Monitoring China will construct a strong Smart Grid with Informa ionization, automation and interactivity features. Ultra-High Voltage (UHV) network is regarded as the backbone frame for the strong Smart Grid. To monitor the UHV transmission lines is a component of ATO in Smart Grid and it is a necessary measure to provide the lines with security, stability and reliable operation. The transmission lines monitoring system mainly collects the environmental parameters of the lines and towers, including temperature, humidity, pollution, ice, wind and fire, lightning strike, stress status and so on [6].

1.4.5 Light Control Systems:

Light control is one of the classic examples of using ZigBee in a house or commercial building. In traditional light installation, to turn on or off the light it is necessary to bring the wire from the light to a switch. Installation of a new recessed light, for example, requires new wiring to a switch. If the recess light and the switch are equipped with ZigBee devices, no wired connection between the light and the switch is necessary. In this way, any switch in the house can be assigned to turn on and off a specific light[7].

1.4.6 Consumer Electronics:

In consumer electronics, ZigBee can be used in wireless remote controls, game controllers, a wireless mouse for a personal computer, and many other applications. In this section, we

briefly review the application of ZigBee in wireless remotes. An infrared (IR) remote controller communicates with televisions, DVDs, and other entertainment devices via infrared signals. The limitation of IR remotes is that they provide only one-way communication from the remote to the entertainment device. Also, IR signals do not penetrate walls and other objects and therefore require line of sight to operate properly. Radio frequency (RF) signals, however, easily penetrate walls and most objects. IEEE 802.15.4 is a proper replacement for IR technology in remote controls because of the low cost and long battery life of ZigBee-based wireless communication [8] . IEEE 802.15.4 can be used to create two-way communications between the remote control and the entertainment device. For example, song information or on-screen programming options can be downloaded into the remote itself, even when the remote control is not in the same room as the entertainment device.

1.5 Security

The latest version of ZigBee offers improvements in various aspects, including its low power consumption, flexibility, and cost-effective deployment. However, the challenges persist, as the upgraded protocol continues to suffer from a wide range of security weaknesses. Constrained wireless sensor network devices cannot use standard security protocols such as asymmetric cryptography mechanisms, which are resource-intensive and unsuitable for wireless sensor networks[9]. ZigBee uses the Advanced Encryption Standard (AES), which is the best recommended symmetric key block cipher for securing data of sensitive networks and applications. However, AES is expected to be vulnerable to some attacks in the near future. Moreover, symmetric cryptosystems have key management and authentication issues. To address these concerns in wireless sensor networks, particularly in ZigBee communications, in this paper, we propose a mutual authentication scheme that can dynamically update the secret key value of device-to-trust center (D2TC) and device-to-device (D2D) communications. In addition, the suggested solution improves the cryptographic strength of ZigBee communications by improving the encryption process of a regular AES without the need for asymmetric cryptography. To achieve that, we use a secure one-way hash function operation when D2TC and D2D mutually authenticate each other, along with bitwise exclusive OR operations to enhance cryptography. Once authentication is accomplished, the ZigBee-based participants can mutually agree upon a shared session key

and exchange a secure value. This secure value is then integrated with the sensed data from the devices and utilized as input for regular AES encryption. By adopting this technique, the encrypted data gains robust protection against potential cryptanalysis attacks. Finally, a comparative analysis is conducted to illustrate how the proposed scheme effectively maintains efficiency in comparison to eight competitive schemes. This analysis evaluates the scheme's performance across various factors, including security features, communication, and computational cost

1.6 Neural Network

The frequency noise could interfere with the signal, which causes distortion to the signal before its arrival at the receiver[10]. Therefore, filtering the signal from noise is essential to ensure carrying the signal from the sender to the receiver with less error. Therefore, removing signal noise in the signal is necessary to mitigate its negative sequences and increase its capability in industrial wireless sensor networks. Moreover, researchers have reported a positive impact of utilizing the Kalmen filter in detecting the modulated signal at the receiver side in different communication systems, including ZigBee. Meanwhile, artificial neural network (ANN) and machine learning (ML) models outperformed results for predicting signals for detection and classification purposes. This paper develops a neural network predictive detection method to enhance the performance of modulation. First, a simulation-based model is used to generate the modulated signal in the IEEE802.15.4 wireless personal area network (WPAN) standard. Then, Gaussian noise was injected into the simulation model. To reduce the noise of phase signals, a recurrent neural networks (RNN) model is implemented and integrated at the receiver side to estimate the phase signal. We evaluated our predictive-detection RNN model using mean square error (MSE), correlation coefficient, recall, and F1-score metrics. The result shows that our predictive-detection method is superior to the existing model due to the low MSE and correlation coefficient (R-value) metric for different signal-to-noise (SNR) values

1.7 Related work

Significant work has been done over the past few years in FPGA and ASIC implementation of IEEE 802.15.4 standard, we will summarize some of them in this section.

Many researchers tried to implement hardware for ZigBee protocol, Oh and Lee [1] implemented a 2.4 GHz radio transceiver for WPAN in 0.18−µm CMOS technology, in this implementation Offset Quadrature Phase Shift Keying (OQPSK) pulse shaping has been implemented for 2.4 GHz band of IEEE 802.15.4 standard. The modulator and demodulator for 868/915 MHz band of IEEE 802.15.4 standard have been implemented in [2] [3] for ZigBee wearable devices in medical applications.

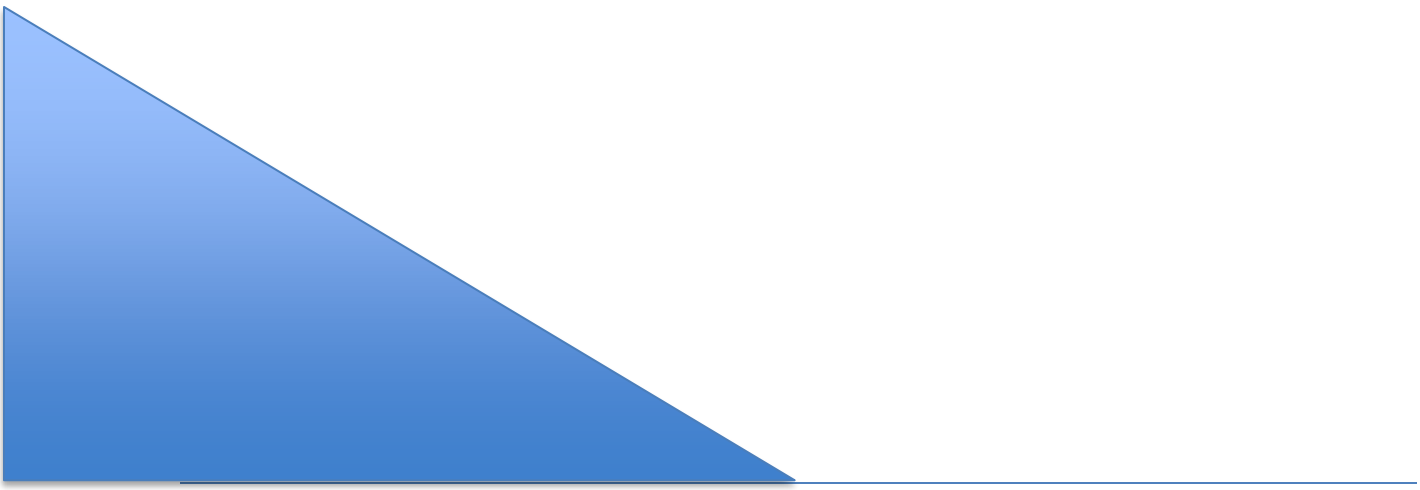
Ting Ting Meng et al [4]. has designed, implemented, and tested a Zigbee receiver using a commercially available off-the-shelf Zigbee transceiver. The receiver was defined in a Harris System-in-Package (SIP) and implemented on a Xilinx Virtex-4 LX60 field-programmable gate array (FPGA). The receiver includes components for carrier synchronization, intermediate frequency (IF) down-conversion, filtering, quadrature demodulation, chip synchronization, and dispreading.

Rafidah Ahmad et al [5]. has designed and implemented a Zigbee digital transmitter for an acknowledgement frame alone. It is modeled using Verilog Hardware Description Language and is then implemented on Spartan 3E FPGA. This digital transmitter consists of Cyclic Redundancy Check (CRC), Bit to Symbol, Symbol to Chip and OQPSK modulator, a pattern generator and logic analyzer.

There are many implementations of IEEE 802.15.4 standard, but we are the first ones who implement the receiver using RNN instead of the traditional demodulator, and we achieved very good performance by our design, we achieved the lowest BER comparing to the previous implementations.

Chapter 2: System of ZigBee

- **Transmitter**
- **Receiver**
- **AES**



2.1 Transmitter

The physical layer of the transmitter consists of 3 main blocks which are bit to symbol, symbol to chip and the modulator as shown as in figure 2.1.1. During each data symbol period, four information bits are used to select 1 of 16 nearly orthogonal pseudo-random noise (PN) sequences to be transmitted. The PN sequences for successive data symbols are concatenated, and the aggregate chip sequence is modulated onto the carrier using offset quadrature phase-shift keying (O-QPSK).[1]

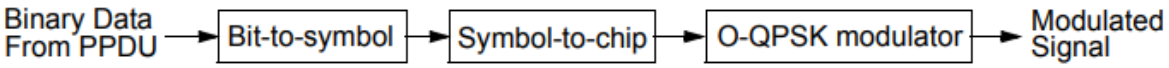


Figure 2.1.1 reference transmitter blocks.[1]

2.1.1Bit to symbol

Four information bits create one symbol. This means there are sixteen different possible combinations between 0000 and 1111. The combinations are used for chip mapping. Each octet is symbolized as 4 LSBs (b0, b1, b2, b3) (Least Significant Bit) map into one symbol, the rest (b4, b5, b6, b7) maps into another symbol. Each octet of the PPDU is processed through the modulation and spreading functions, as illustrated in Figure 2.1.1, sequentially, beginning with the Preamble field and ending with the last octet of the PSDU. Within each octet, the least significant symbol (b0, b1, b2, b3) is processed first and the most significant symbol (b4, b5, b6, b7) is processed second.[1]

2.1.2 Symbol to chip

Each data symbol shall be mapped into a 32-chip PN sequence as specified in Table 2.1.2.1. The PN sequences are related to each other through cyclic shifts and/or conjugation.[1]

Data Symbol (Decimal)	Data Symbol (Binary) (b_0 b_1 b_2 b_3)	Chip Values (c_0 c_1 c_2 … c_{31})
0	0000	11011001110000110101001000101110
1	1000	11101101100111000011010100100010
2	0100	00101110110110011100001101010010
3	1100	00100010111011011001110000110101
4	0010	01010010000101110110110011100011
5	1010	00110101001000101110110110011100
6	0110	11000011010100100010111011011001
7	1110	10011100001101010010001011101101
8	0001	10001100100101100000011101111011
9	1001	10111000110010010110000001110111
10	0101	01111011100011001001011000000111
11	1101	01110111101110001100100101100000
12	0011	00000111011110111000110010010110
13	1011	01100000011101111011100011001001
14	0111	10010110000001110111101110001100
15	1111	11001001011000000111011110111000

Table 2.1.2.1: Symbol-to-Chip Mapping.[1]

2.1.3 Modulator

The chip sequences representing each data symbol are modulated onto the carrier using O-QPSK with half sine pulse shaping. Even-indexed chips are modulated onto the in-phase (I) carrier, and odd-indexed chips are modulated onto the quadrature-phase (Q) carrier. Each data symbol is represented by a 32-chip sequence, and so the chip rate is 32 times the symbol rate. To form the offset between I-phase and Q-phase chip modulation, the Q-phase chips shall be delayed by Tc with respect to the I-phase chips, as illustrated in Figure 2.1.3.1, where Tc is the inverse of the chip rate.[1]

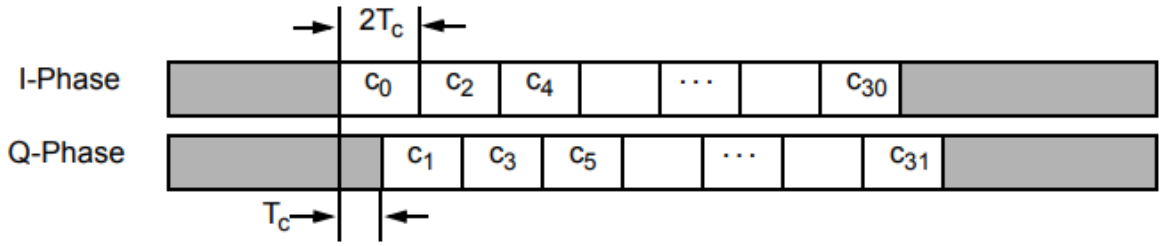


Figure 2.1.3.1: O-QPSK chip offsets.[1]

2.1.4 Pulse shaping

The half-sine pulse shape is used to represent each baseband chip. Fig. 2.1.4.1 shows the pulse shaping of the I and Q branch.

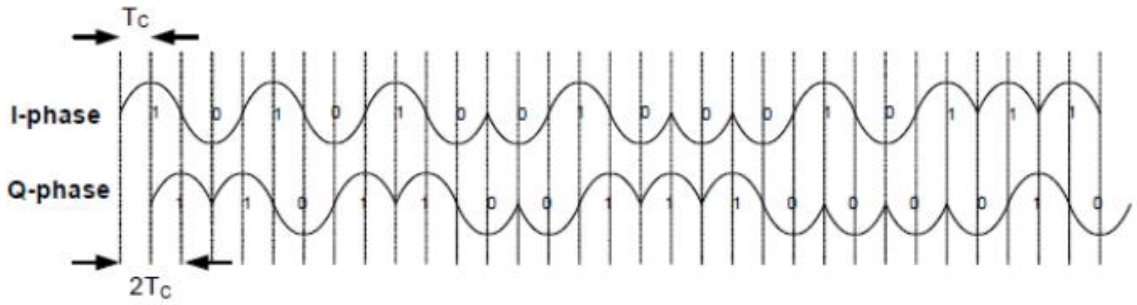


Figure 2.1.4.1: The pulse shaping of the I and Q branch.

2.2 Receiver

2.2.1 Decoding

The traditional receiver's **Demodulator** or **Detector** consists of two main blocks the first one is the **mixer** and the second is the **LPF** to reject the high frequency components as shown in figure 2.2.1.1.b and in that block, we get the signal down to the baseband frequency which means that in this block we inverse the work done by the modulator which is shown in figure 2.2.1.1.a.

After the demodulator **I** and **Q** branches are passed on to the ADC block. The **I** and **Q** branches could be combined to make a one signal vector, let's call it **X** and that vector is the input to a fundamental block which estimates the actual received binary bits, this block is called the

Decoder. In that block, if we have four points in the constellation space, we need to get the maximum likelihood point in the constellation space to the received vector. To do this function we have to get the correlation between the received vector and the four vectors of the constellation, and the largest result represents the maximum likelihood constellation point **Si** where i = 1:4 as shown in Figure 2.2.1.2.

(a) (b)

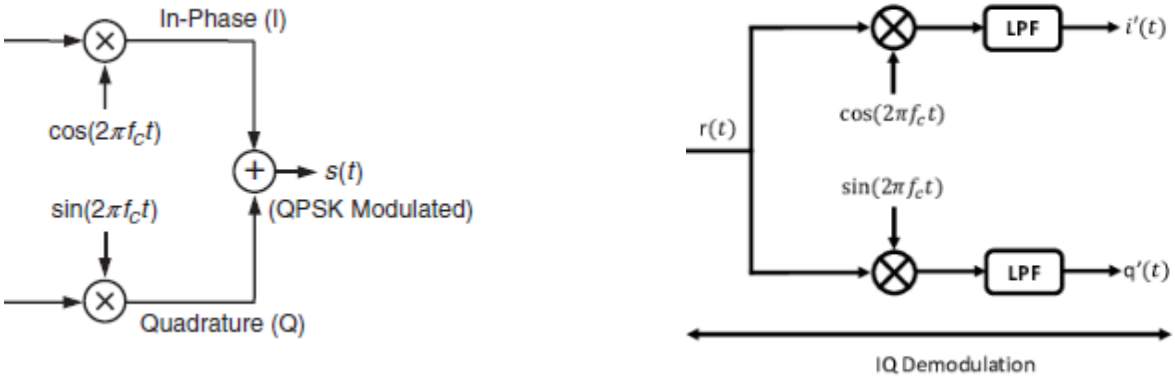


Figure 2.2.1.1: IQ modulator and demodulator.

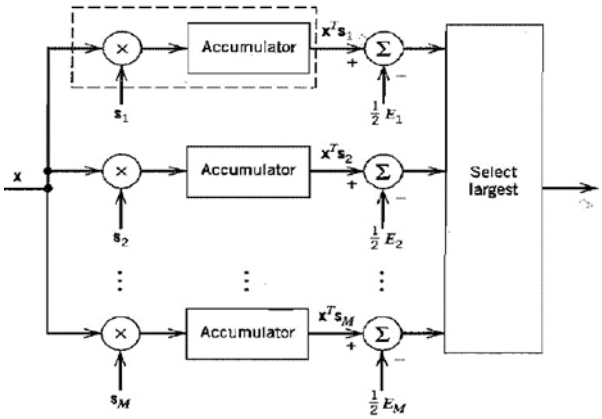


Figure 2.2.1.2: Decoder.

This approach suffers from several problems:

1- The phase offset between the local oscillator of the transmitter and the local oscillator of the receiver or the signal itself while it's transmission in its medium it can experience phase shifts due to many reasons like reflections.

2- The frequency offset between the local oscillator of the transmitter and the local oscillator of the receiver and that can cause a total failure of the communications as the frequency offset causes the constellation space to rotate so if intended symbol represents 00 the frequency offset can make the decoder to receive it as 01 due to the rotation as in figure 2.2.1.3.

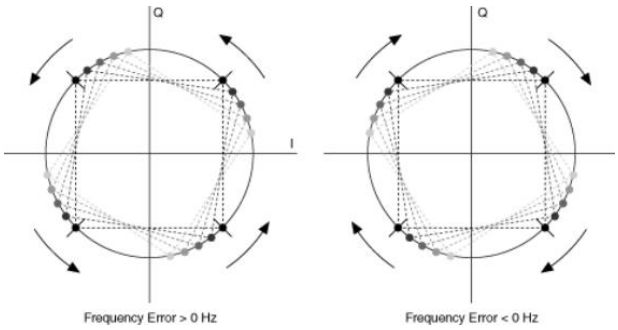


Figure 2.2.1.3: Constellation rotation due to FO.

3- The noisy environments lead to several issues such as low detection performance of the weak ZigBee OQPSK signals which are designed to operate in applications like the Internet of Things. So, the system should be very immune to the noise which could be initiated by human or natural phenomena besides the large-scale fading effects, and in both cases, it is considered a big challenge for signal reception. Therefore, removing noise in the OQPSK signal is necessary to mitigate its negative sequences and increase its capability in industrial wireless sensor networks.

In our project we propose a **Neural Network** which will replace the demodulator and the decoder, which means that the neural network would be trained to take the OQPSK signals and predict the binary bits which are represented by the OQPSK symbol received. The magic about the neural network approach is that we can include in our training all the negative effects which are experienced by the signals while its propagation through the medium. Surely

there are traditional techniques to mitigate these effects but we are here talking about a neural network that does all the computations on the signal and produces the pure binary desired bits.

As there are many types of neural networks and each type has a specific application to work with, our application is to detect a signal which also can be presented as a sequence of series samples and the best sequence detection neural networks is the **Recurrent Neural Network.** The term recurrent in **RNN** refers to the feedback nature in the RNNs as shown in figure 2.2.1.4 and because all the samples in a sequence have contribution in making the decision of the detection.

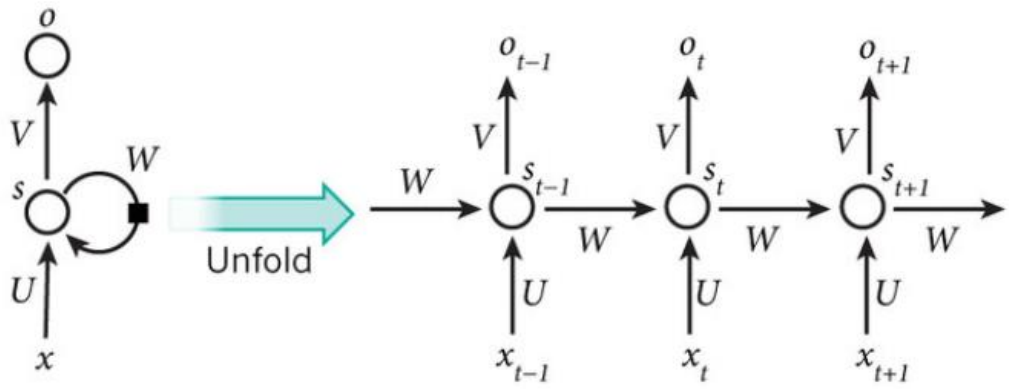


Figure 2.2.1.4: Folded and Unfolded RNN.

As shown in the previous figure the output at time instant ‘t’ depends on the input at time ‘t’ and the signal S at time ‘t-1’ which depends on the input at time ‘t-1’ and the previous inputs at time ‘t-2’, ‘t-3’, ……..,’0’.

To turn our demodulation and decoding to a neural network system fundamental change would occur to make the network fulfill its function.

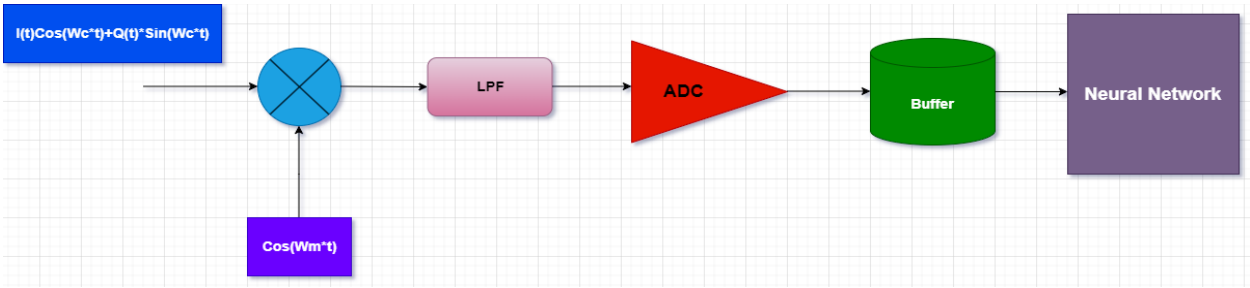
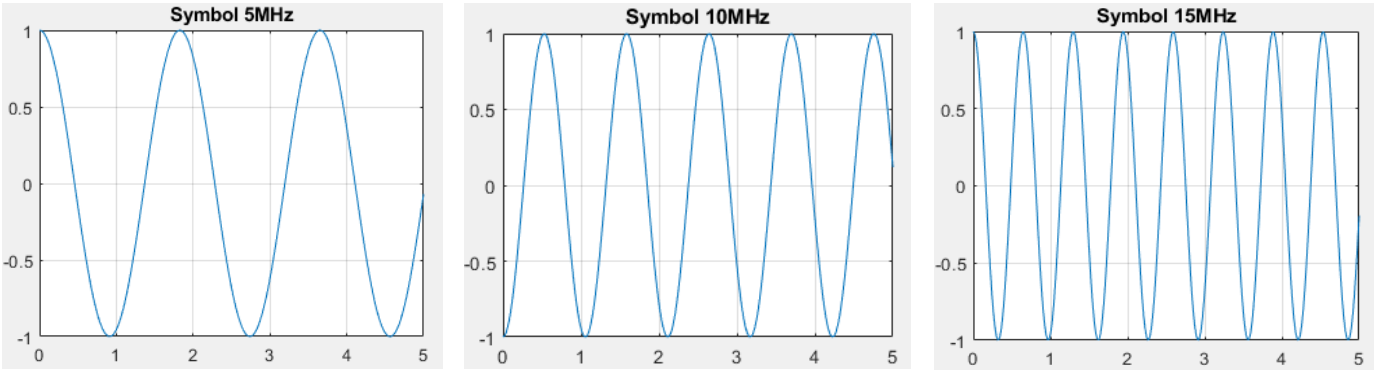


Figure 2.2.1.5: General architecture of the Receiver

The entire operation of the receiver is described in Figure 2.2.1.5. The input RF signal which is the actual output from the transmitter is multiplied by Cos (Wm*t) and this step exists because we cannot deal with the signal in the RF because in RF the signal frequency is 2.4GHz as the ZigBee standard so if we need to sample it we need a much higher frequency which is not less than 4.8GHz which does not make sense. There are two questions that need to be answered, the first is why the signal is multiplied by 'Cos' not by 'Sin'? Because the trigonometric identity says "Cos(A)Cos(B) = 0.5[Cos(A+B) + Cos(A-B)]" and "Sin(A)Cos(B) = 0.5[Sin(A+B) + Sin(A-B)]" then the Low Pass Filter will reject the 'A+B' terms while the 'A-B' terms will remain. The second question is what is the value of the remaining 'A-B' terms? Or in other words what is the value of "fc - fm" which is the frequency that will be sampled? The answer depends on what would happen if this value is high and what would happen if this value is low.



(a) *(b)* *(c)*

Figure 2.2.1.6: General architecture of the Receiver.

As shown in Figure 3.2.1.6 the difference between the three frequencies or three 'A- B'. For

figure (a) the signal is moving slowly relatively which means that the difference between the samples is too small and that puts a requirement on the network to remember information over many time steps and for figure (c) it's the opposite the signal is moving very fast which means the difference between the samples is very big difference where rapid changes in sequences can cause instability during training, making it difficult for the model to converge to a good solution. Both cases have bad effects in the training of the network so we will choose the figure (b) as the difference between the samples will be acceptable to the neural network. So 'A - B' will be 10 MHz which is the input frequency to the sampler of the ADC. The samples will be buffered in FIFO till the calculations of the last input sample are done, after that the next sample in the sequence will be fed to the network.

The simple introduced Recurrent neural network has a very bad performance when dealing with long input sequences. To illustrate this problem, we need first to talk about the **Gradient**

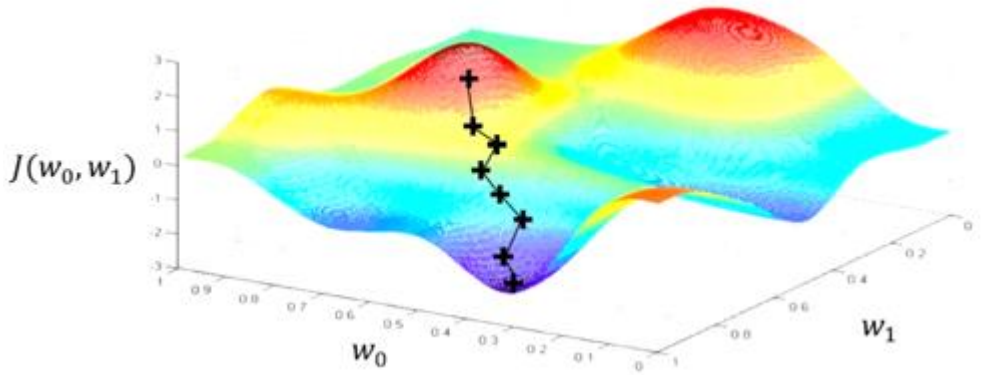


Figure 2.2.1.7: Gradient Descent

Descent. Let's assume that we have a network which simply consists of 2 nodes (x) —w0—> (node1) —w1—>(node2) –> (y') where 'x' is the input and 'y'' is the output, and let's define the **Loss Function** which tells us how much the predicted output differs from the actual output 'y'. Now for a single input and a single output we need to tune w0 and w1 to minimize the **Loss Function.** So, the loss function is a function of the network parameters J(w0, w1). The tuning operation that we mentioned earlier is done by computing the negative gradient (differentiation) of function J with respect to w0 and with respect to w1 using the chain rule. And that will give us the direction and magnitude the weights should be updated with to move towards the global minimum of the **Loss function** by repeating this operation till

convergence like the case in Figure 2.2.1.7. Note that the initial parameters values are randomly selected.

So, now what is the problem with the simple RNN? When the weight of the feedback loop which is called 'W' in figure 2.2.1.4 is greater than 1 let's make it 2 for example, and if our sequence is of length 50-time step so, by the end of the sequence the feedback signal will have value equals to the input value multiplied by 2 raised to the power 50 which is a Huge Number and that will cause the gradient descent steps to **explode** so the operation will not converge

And when the parameter 'W' is less than 1 let's take it 0.5 for example and if our sequence is of length 50 time step so, by the end of the sequence the feedback signal will have value equals to the input value multiplied by 0.5 raised to the power 50 which is a **Super close to zero Number** and that will cause the gradient descent steps to **vanish** so also the operation will not converge as the loss function point almost does not move.

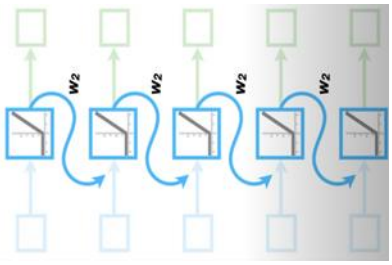


Figure 2.2.1.8.

This problem is called **Vanishing/Exploding Gradient** which appears with long sequences like our 250-time step sequence. So, we need to replace the simple RNN nodes with **LSTM** nodes which are more complex but have great performance with long sequences.

LSTMs in Figure 2.2.1.9 provide significant advantages over simple RNNs, especially for tasks involving long-term dependencies and complex temporal dynamics. Their gated architecture allows them to maintain and update information selectively, making them more robust, flexible, and capable of handling the vanishing gradient problem. These benefits make LSTMs the preferred choice for sequential data.

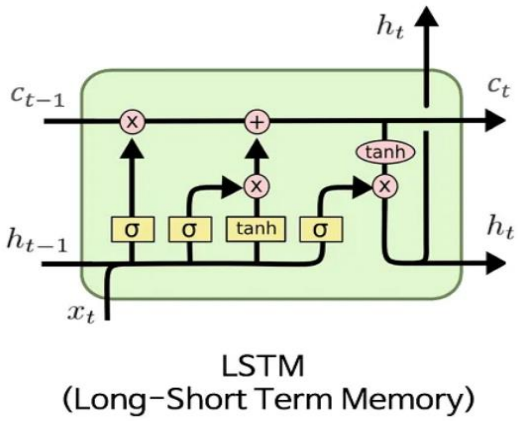


Figure 2.2.1.9: LSTM node.

The gated mechanism of the LSTM node helps it to overcome the simple RNN issues, where the LSTM has four gates: **Forget, Input, Cell, output** some resources ignore the Cell gate and embed it into the Input gate. There are two main signals in the LSTM nodes which are the inputs and the outputs in Figure 2.2.1.9 inputs: Ct-1 & ht-1, outputs: Ct & ht which are the feedback signals, note that in simple RNN it was only one feedback signal but in LSTM it's separated to two signal Cell State 'C' which holds the long-term information and Hidden State 'H' to hold the short-term information. Let's illustrate the function of each gate, **Forget Gate:** Determines which information from the cell state should be discarded. This helps in removing irrelevant or outdated information, **Input Gate:** Controls which new information should be added to the cell state. This helps in incorporating new, relevant information, **Output Gate:** Controls what part of the cell state should be output as the hidden state. This mechanism helps in deciding the output based on both new and old information.

The LSTM has robustness against the noise in the sequences due to its ability to maintain relevant information while discarding irrelevant or noisy information. This capability arises from the forget gate's ability to selectively forget parts of the cell state. This point has its great effect on the demodulation performance against noisy environments.

Now after we decide the type of our neural network nodes, we need to determine the network itself, how many layers needed the connections between the layers, the weights and biases of the network. All of these parameters are determined through the training phase of the network.

Before beginning in the training process, we need to establish the training data which is a very dangerous step, as this step will determine the impurities that our new decoder will be robust against.

The training data will be a set of vectors each vector is 250 samples that represent the signals that are suffering from impurities like:

1- Additive White Gaussian Noise that makes the SNR of the signal's ranges from [-13dB: 12dB].

2- Phase Offset between the local oscillators of the Rx and Tx ranges from [-45: 45] degrees.

3- Large Scale Fading ranges from [0.2: 1].

The hyper parameters of the network which is the number of layers and number of nodes per each layer, the learning rate which is the rate of updating the weights and biases.

The neural network consists of three layers, two hidden layers each one consists of five LSTM nodes and one output fully connected dense layer consists of four nodes to represent the predicted symbol in the constellation space. The dense node is only weighted summation of the last hidden layer with sigmoid activation function.

The used Loss function in the compilation of the network is Sparse Categorical Cross Entropy which is the categorical version of the Binary Cross Entropy loss function. The optimizer used is Adam optimizer which does the early mentioned gradient descent operation with learning rate equals 0.001. The learning rate is rating the weights are updated with which means $W_{new} = W_{old} - (\eta) * \frac{\delta J}{\delta W}$ where J is the loss function and η is the learning rate. And the metric of the performance is the accuracy which by the end of the training reached 94%. While the training data set reaches 6000 examples where the test data set reached 1200 examples. The training is done using the **TensorFlow** library.

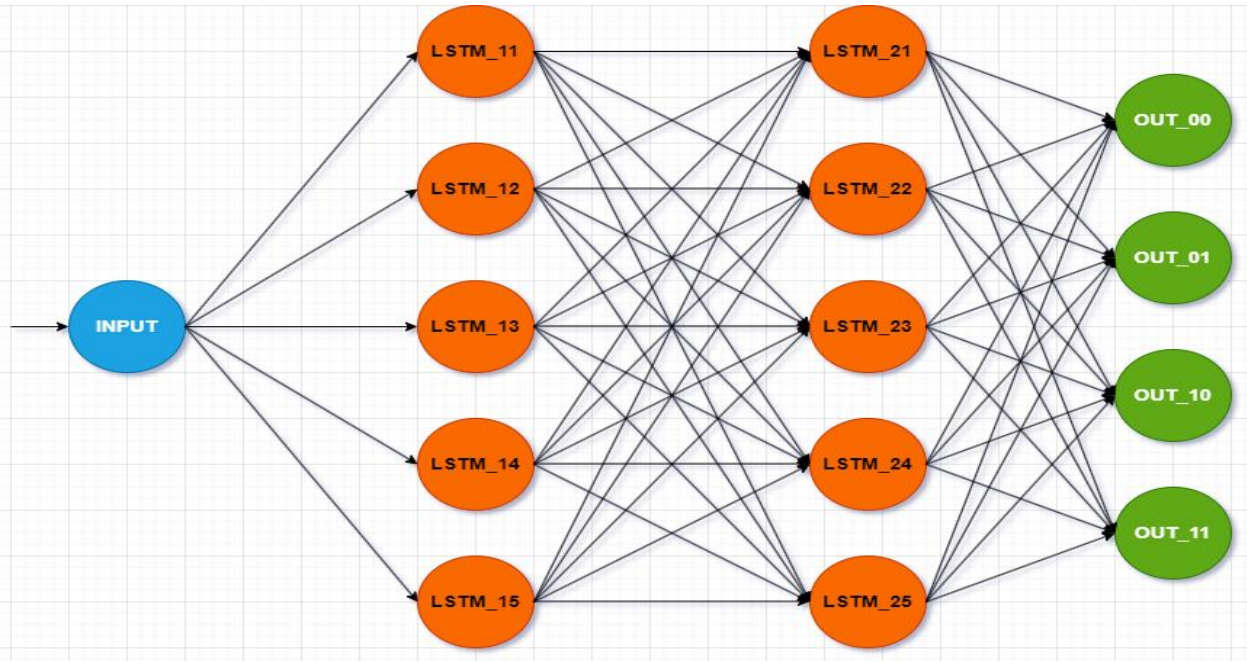


Figure2.2.1.10: The Neural Network

The feedback loops do not exist in the neural network in figure 2.2.1.10 to prevent the diagram from being complexed. But for our imagination let's setup in our minds how the feedback signals will be connected. For the first hidden layer each node from LSTM_11 to LSTM_15 has a connection with the output of each node in the same layer. We need to ask an important question here: which of the LSTM signals is the output of the LSTM node, is it the Cell state or the Hidden state? The answer is the Hidden state where the Cell state holds information subjected to its node only not the other nodes. The feedback connections that we mentioned earlier represent the vector ht-1 in figure 2.2.1.9. So, now how the hidden states vector is combined with the input signal the result is fed to a function? The answer is that in each node before each gate there are two matrix multiplication operations, one for the hidden state vector and the other is for the actual input signal. In our case the hidden vector has the shape [1x5] so before each gate this vector will be multiplied with a matrix called **Recurrent Kernel,** each gate has a different recurrent kernel of shape [5x1] so the resultant value is a single scaler. On the other hand, the input signal is multiplied by a matrix called **Kernel Matrix,** each gate has a different kernel matrix and its shape differs with the shape of the input. For example each node in the first layer has a single input signal so all the kernel matrices of this layer is a scalar, while each node in the second layer has five input signals

from the first hidden layer which means the input shape is [1x5] so all the kernel matrices of this layer have the shape [5x1].

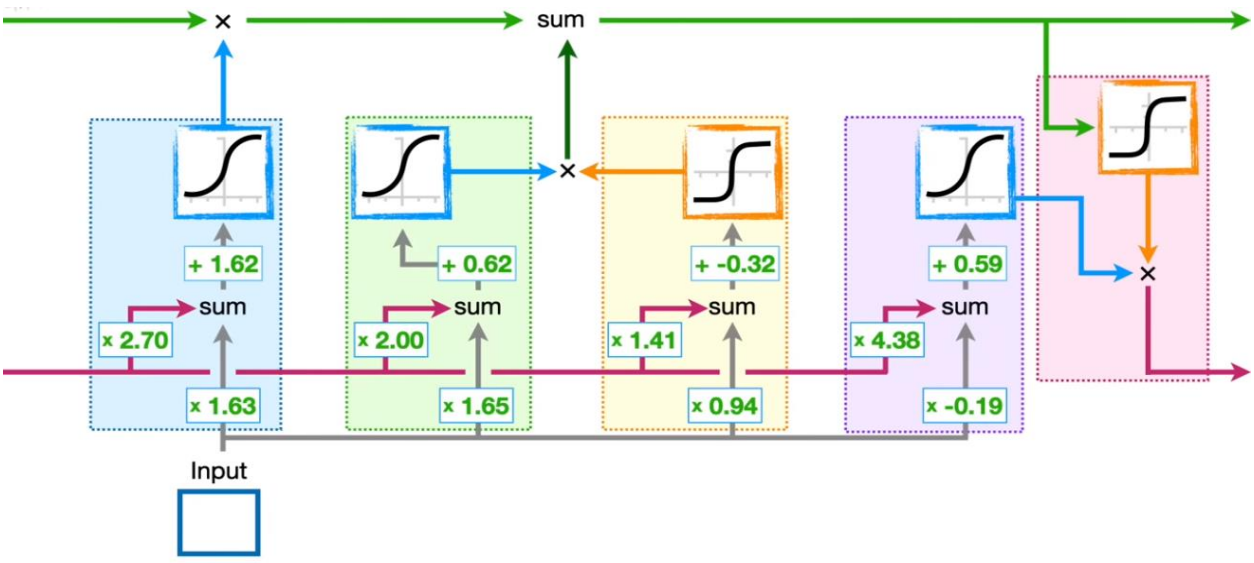


Figure 2.2.1.11: Kernel and Recurrent Kernel

In figure 2.2.1.11 the purple line represents the hidden state while the green line represents the cell state. As we can see that before each gate the input signal is multiplied by scalar this scalar in general is a matrix not a scalar but for the simplicity of the figure it's written as a scalar.

The values that will appear on the output layer will be like that [0.93 0.32 0.45 0.27]. These numbers represent how much the network trusts this symbol as a prediction. For example, the proposed numbers tell us that the predicted symbol is mostly the symbol [00].

So, we will design a block after the neural network that selects the maximum of these output values from the network and then pass it as binary values zeros and ones to the next block in the bit chain.

2.2.2 Chip to symbol

In this module the received serial data is stored in a 32-bit register, also the lookup table used in the transmitter for symbol to chip mapping is used in reverse order to decode the received symbol. The received chip sequence is compared with the 16 different chip sequences

transmitted by the transmitter. Here, the XNOR operation is carried out on received chip sequence with each of 16 different chip sequences used at the transmitter. In XNOR operation, the counter with maximum value provides the symbol transmitted by the lookup table of chip to symbol mapping.

2.2.3 Symbol to bit

In this block 4 bit received symbol from the chip to symbol block are sent to output in form of bits after each other

2.2.4 Frequency offset estimation

Frequency offset is the mismatch between the transmitter frequency and the receiver frequency; it is mainly caused by Frequency mismatch between the Tx and Rx oscillators and Doppler effect.

In the IEEE 802.15.4 standard, receiver sensitivity is defined as the minimum received signal power that results in a packet error rate (PER) of less than 1%. IEEE 802.15.4 requires only -85 dBm of sensitivity for operations in the 2.4 GHz ISM band. Although not required by the standard, commercially available transceivers can achieve sensitivity levels of -95 dBm to -100 dBm.

Frequency offset degrades the sensitivity level. Figure 2.2.4.1 shows an example of receiver sensitivity degradation versus input carrier frequency offset when there is no offset correction.

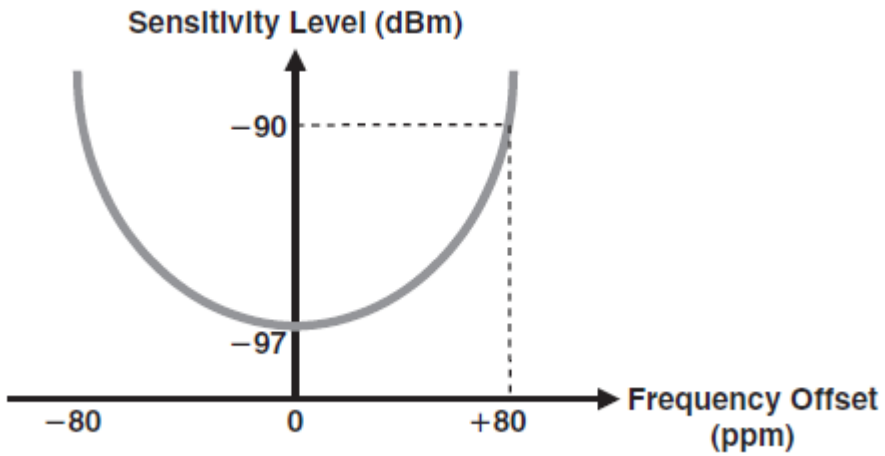


Figure 2.2.4.1: Example of Receiver Sensitivity Degradation Due to Frequency Offset

So, we need a frequency offset estimator to estimate the frequency offset to adjust the local oscillator frequency at the receiver to match the transmitter's frequency.

2.2.5 Packet edge detection

Packet edge detection is an operation that tells us: is there a packet of our standard over the air? This operation contains 2 main blocks: energy detection that tells us if there is a packet or not and cross correlation that detects if the packet is of our standard or not. To achieve this, we need to calculate a metric related to the energy and compare it with a threshold value to detect if there is a packet over the air or not.

The packet edge detection operation will produce a single bit. The bit indicates that we detected a packet on the channel, or we did not. There are four possible outcomes of this operation:

1- Detect: This means there is a packet.

2- False alarm: This means there is no packet over the air, but we detect a packet.

3- Miss: This means there was a packet, but we did not detect it.

So, we need to get the best value of the threshold to reduce the probability of missed packet and false alarm, if the threshold value is very high the probability of missed packet will increase and if the value is very low the probability of false alarm will increase.

2.2.5.2 Energy detection

This step is like sensing the channel if that there’s any symbol is sent over the channel .so if there’s any symbol is sent over the channel, the energy detection block works to check this data over a certain threshold or not, if it’s above this threshold then we go to the next state which is the cross correlation to check the is really belong to ZigBee protocol or another protocol. We achieve this by doing autocorrelations to a window of the input samples and comparing it with the threshold.

2.2.5.2 Cross correlation

After sensing the channel and finding that there is a packet there. We need to check if this packet belongs to Zigbee standard or another standard. So, we do cross correlation between the preamble of the packet and saved preamble for our standard packet. ZigBee packets typically start with a known preamble sequence. This preamble is 8 zero symbols. By performing a cross-correlation between the incoming signal and this known sequence, the system can detect the alignment and identify the precise start of the packet. Cross-correlation provides higher accuracy in detecting packet edges, especially in environments with noise and interference.

2.3 Advanced Encryption Standard (AES)

AES is the main block in MAC layer which is responsible for security (Encryption and Decryption of data between Transmitter and Receiver). AES is a symmetric block cipher with a block length of 128 bits and support for key lengths of 128, 192, and 256 bits. It was published by NIST (National Institute of Standards and Technology) in 2001 [14] & still used in most applications till now due its performance. Here, we assume a key length of 128 bits, which is likely to be the one most commonly implemented. so, we use AES-128 in Zigbee System.

AES-128 evaluate encryption or decryption over 10 rounds as shown in Figure 2.3.1. each round consists of 4 transformations will be discussed later.

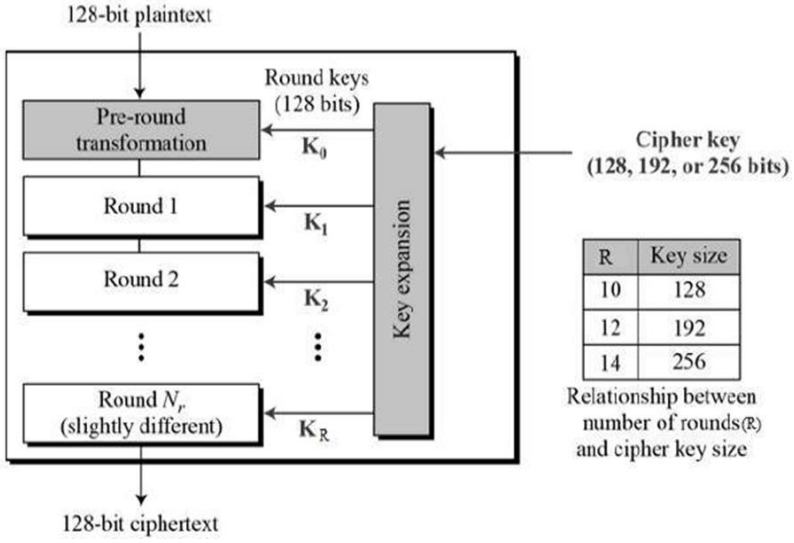


Figure 2.3.1: Flow of AES Algorithm

Figure 2.3.2 shows that the plaintext consists of a sequence of 128-bit which divided into 4 words. Each word consists of a sequence of 4 bytes.

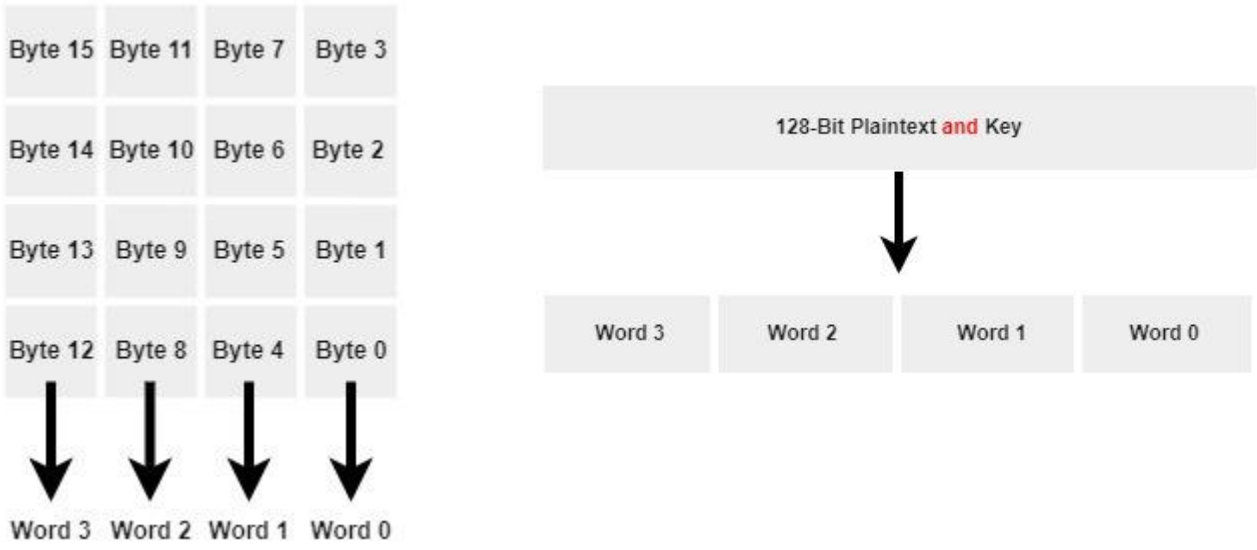


Figure 2.3.2: Structure of plaintext and key

Eventually, AES has 3 different algorithms to achieve its goal to secure information between transmitter and receiver.

2.3.1 Encryption Algorithm

AES-128 in transmitter of Zigbee's Mac Layer takes plaintext (Data required to be sent and encrypted) and the key used in both encryption and decryption so that we guarantee confidentiality between transmitter and receiver.

The AES encryption algorithm can be broken into three phases: the initial round, the main rounds, and the final round. All of the phases use the same sub-operations in different combinations as shown in Figure 2.3.1.1.

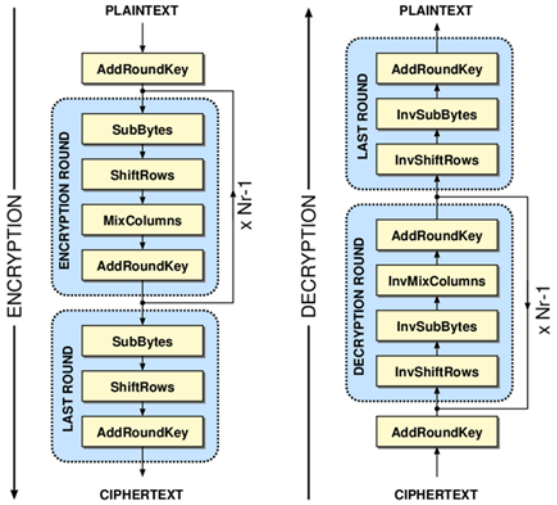


Figure 2.3.1.1: AES-128 Encryption and Decryption Algorithm

let's get in the rounds and illustrate each round by details.

1) Initial Round

   Initial round consists of one operation only which is add round key operation

2) Main Rounds

   Main rounds from round 1 to round 9 use 4 operation in order ((a) Sub Bytes (b) Shift Rows (c) Mix Columns (d) Add Round Key) as shown in Figure 2.3.1.1.

3) Final Round

   The final round (round 10) consists of 3 operations only in order ((a) Sub Bytes (b) Shift Rows (c) Add round key) as shown in Figure 2.3.1.1

Let's go deep into each operation and understand how it used to encrypt the data. As we shown before in Figure 2.3.1 that the plaintext and key are structured in a matrix 4 * 4 bytes.

2.3.1.1 Sub Bytes Transformation

Sub Bytes transformation is a nonlinear substitution step where each entry (byte) of the current state matrix is substituted by a corresponding entry in the AES S-Box Table shown in Figure 2.3.1.1.1 which used also in Key expansion algorithm.

For Example:

		y														
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figure 3.3.1.1.1:S-Box Table

00	3C	6E	47
1F	4E	22	74
0E	08	1B	31
54	59	0B	1A

Sub Byte

Transformation

63	EB	9F	A0
C0	2F	93	92
AB	30	AF	C7
20	CB	2B	A2

2.3.1.2 Shift Rows Transformation

Shift rows transformation is a transposition step where the four rows of the state are shifted cyclically to the left by offsets of 0, 1, 2, and 3. As shown in Figure 2.3.1.2.1

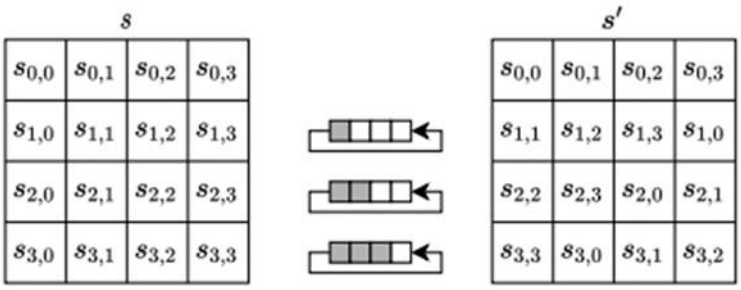


Figure 2.3.1.2.1: Shift Row Transformation

For Example:

63	EB	9F	A0
C0	2F	93	92
AB	30	AF	C7
20	CB	2B	A2

Shift Rows

Transformation

63	EB	9F	A0
2F	93	92	C0
AF	C7	AB	30
A2	20	CB	2B

2.3.1.3 Mix Columns Transformation

Mix Columns transformation is a linear mixing operation which multiplies fixed matrix against current State Matrix (column by column), But unlike standard matrix multiplication. Mix Columns perform matrix multiplication as per Galois field (2^8). The Fixed Matrix shown in Figure 2.3.1.3.1.

02	03	01	01
01	02	03	01
01	01	02	03
03	01	01	02

Thus

$$\begin{bmatrix} X3 \\ X2 \\ X1 \\ X0 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} x3 \\ x2 \\ x1 \\ x0 \end{bmatrix}$$

so that the individual output bytes are defined as follows:

Figure 2.3.1.3.1: Mix Columns Fixed Matrix

$$X3 = \{2.x3\} \oplus \{3.x2\} \oplus \{x1\} \oplus \{x0\}$$

$$X2 = \{x3\} \oplus \{2.x2\} \oplus \{3.x1\} \oplus \{x0\}$$

$$X1 = \{x3\} \oplus \{x2\} \oplus \{2.x1\} \oplus \{3.x0\}$$

$$X0 = \{3.x3\} \oplus \{x2\} \oplus \{x1\} \oplus \{2.x0\}$$

For Example:

63	EB	9F	A0
2F	93	92	C0
AF	C7	AB	30
A2	20	CB	2B

Mix Columns

Transformation

BA	84	E8	1B
75	A4	8D	40
F4	8D	06	7D
7A	32	0E	02

2.3.1.4 Add Round Key Transformation

Here is the most important transformation where the cipher key and other round keys role appear. Add round key transformation is simply done by xoring the current matrix state and the key specified by certain round.

For Example: Given the current state and round key as shown in Figure 2.3.2.4.1.

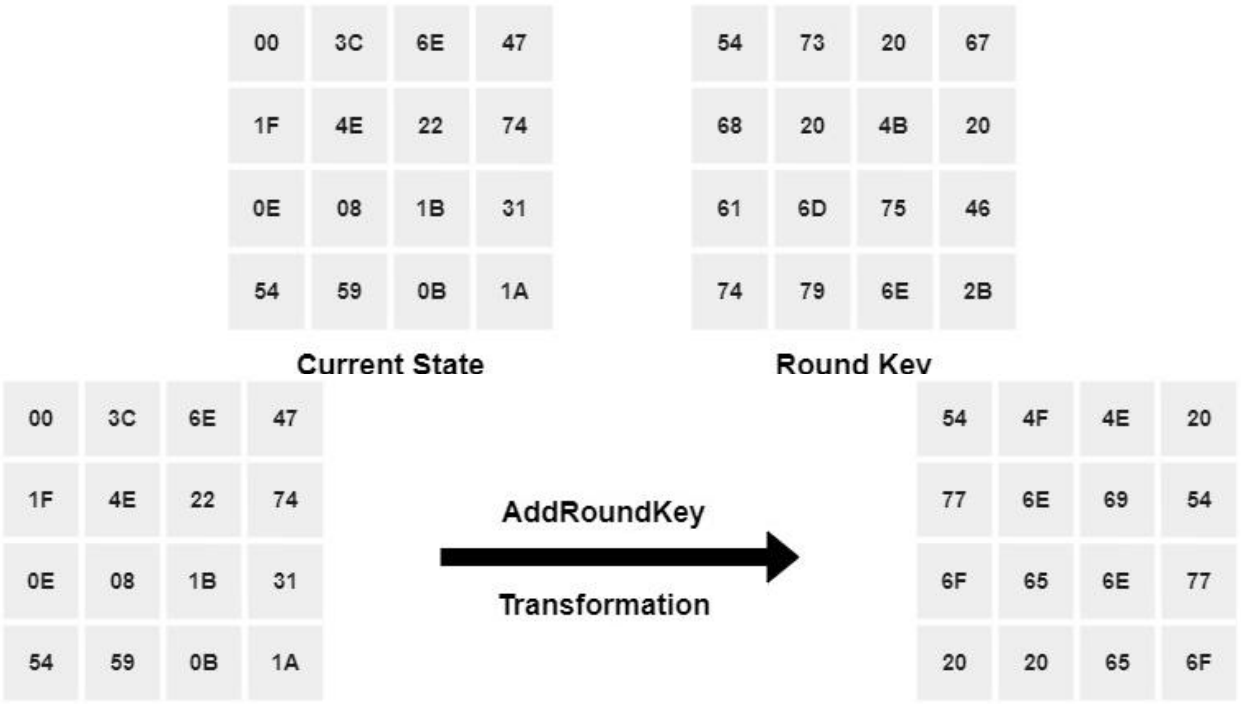


Figure 2.3.1.4.1: Add Round Key Transformation

2.3.2 Decryption Algorithm

AES-128 in receiver of Zigbee's Mac Layer takes ciphertext (encrypted data) and the key used in both encryption and decryption so that we guarantee confidentiality between transmitter and receiver, then the receiver could finally understand the information sent by transmitter.

The AES decryption algorithm can also be broken into three phases: the initial round, the main rounds, and the final round. All of the phases use the same sub-operations in different combinations as shown in Figure 2.3.1.1.

let's get in the rounds and illustrate each round by details.

1) Initial Round
   Initial round consists of one operation only which is add round key operation
   Note that: Add Round key transformation is identical to the forward add round key transformation, because the XOR operation is its own inverse.
2) Main Rounds
   Main rounds from round 1 to round 9 use 4 operation in order ((a) Inv Shift Rows (b) Inv Sub Bytes (c) Inv Mix Columns (d) Add Round Key) as shown in Figure 2.3.1.1.
3) Final Round
   The final round (round 10) consists of 3 operations only in order ((a) Inv Shift Rows (b) Inv Sub Bytes (c) Add Round Key) as shown in Figure 2.3.1.1

Let's go deep into each operation and understand how it used to decrypt the data. As we shown before in Figure 2.3.1 that the ciphertext and key are structured in a matrix 4 * 4 bytes.

2.3.2.1: Inv Sub Bytes Transformation

Inv Sub Bytes transformation is a nonlinear substitution step where each entry (byte) of the current state matrix is substituted by a corresponding entry in the AES inverse S-Box Table shown in Figure 2.3.2.1.1.

For Example:

63	EB	9F	A0
C0	2F	93	92
AB	30	AF	C7
20	CB	2B	A2

Inv Sub Bytes

Transformation

00	3C	6E	47
1F	4E	22	74
0E	08	1B	31
54	59	0B	6F

x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Figure 2.3.3.1.1: Inverse S-Box Table

2.3.2.2: Inv Shift Rows Transformation

Inv Shift rows transformation is a transposition step where the four rows of the state are shifted cyclically to the right by offsets of 0, 1, 2, and 3. As shown in Figure 2.3.2.2.1

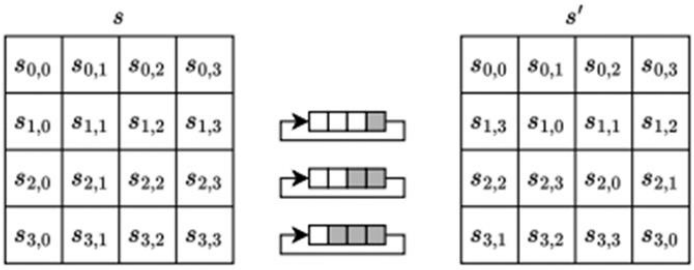


Figure 2.3.2.2.1: Inv Shift Row Transformation

For Example,

63	EB	9F	A0
2F	93	92	C0
AF	C7	AB	30
A2	20	CB	A2

Inv Shift Rows Transformation

63	EB	9F	A0
C0	2F	93	92
AB	30	AF	C7
20	CB	2B	A2

2.3.2.3 Inv Mix Columns Transformation

Inv Mix Columns transformation is a linear mixing operation which multiplies fixed matrix against current State Matrix (column by column), But unlike standard matrix multiplication. Inv Mix Columns perform matrix multiplication as per Galois field (2^8). The Fixed Matrix shown in Figure 2.3.3.3.1.

0E	0B	0D	09
09	0E	0B	0D
0D	09	0E	0B
0B	0D	09	0E

Thus

$$\begin{bmatrix} x3 \\ x2 \\ x1 \\ x0 \end{bmatrix} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} X3 \\ X2 \\ X1 \\ X0 \end{bmatrix}$$

Figure 2.3.2.3.1: Inv Mix Columns Fixed Matrix

so that the individual output bytes are defined as follows:

$$x3 = \{0E.X3\} \oplus \{0B.X2\} \oplus \{0D.X1\} \oplus \{09.X0\}$$

$$x2 = \{09.X3\} \oplus \{0E.X2\} \oplus \{0B.X1\} \oplus \{0D.X0\}$$

$$x1 = \{0D.X3\} \oplus \{09.X2\} \oplus \{0E.X1\} \oplus \{0B.X0\}$$

$$x0 = \{0B.X3\} \oplus \{0D.X2\} \oplus \{09.X1\} \oplus \{0E.X0\}$$

For Example:

BA	84	E8	1B
75	A4	8D	40
F4	8D	06	7D
7A	32	0E	02

Inv Mix Columns Transformation

63	EB	9F	A0
2F	93	92	C0
AF	C7	AB	30
A2	20	CB	2B

2.3.2.4 Add Round Key Transformation

Here is the most important transformation where the cipher key and other round keys role

appear. Add round key transformation is simply done by xoring the current matrix state and the key specified by certain round.

For Example: Given the current state and round key as shown in Figure 2.3.2.4.1.

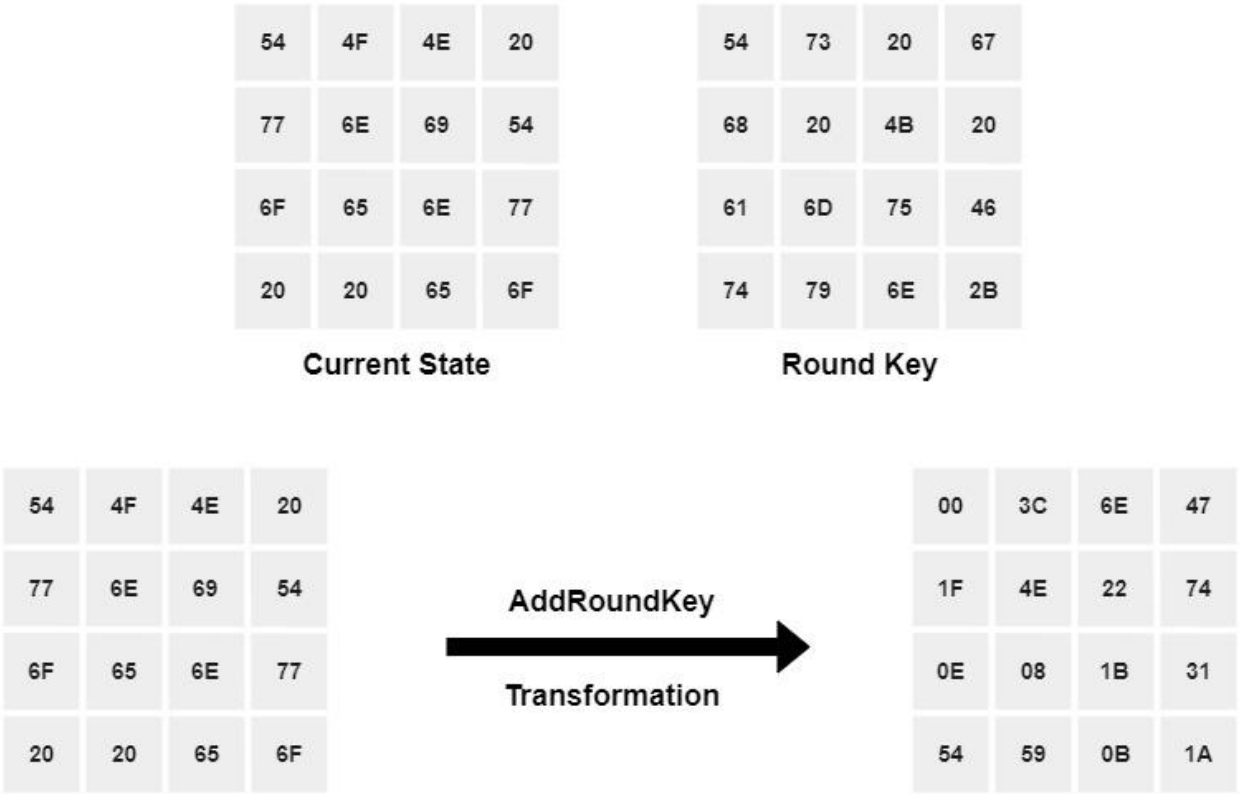


Figure 2.3.2.4.1: Add Round Key Transformation

2.3.3 Key Expansion Algorithm (Key Generator)

Key expansion algorithm used in both sides, Transmitter (during encryption process) and Receiver (during decryption process) of Zigbee's Mac layer.

AES-128 achieve encryption and decryption over 10 rounds. Each round uses a unique round key. All these round keys are generated by the 128-bit cipher key with the following algorithm shown in Figure 2.3.3.1

The upper words considered as the present round key and the lower words considered as the upcoming round key generated by some transformations.

For Example, the present round key is shown in Figure 2.3.3.2

2B	28	AB	09
7E	AE	F7	CF
15	D2	15	4F
16	A6	88	3C

Figure 2.3.3.2: State of the present round key

The least significant word of the present round key goes into Rot Word block which rotate the first byte of this word as shown in Figure 2.3.3.3

2B	28	AB	CF
7E	AE	F7	4F
15	D2	15	3C
16	A6	88	09

Figure 2.3.3.3: State of the present round key after Rot Word.

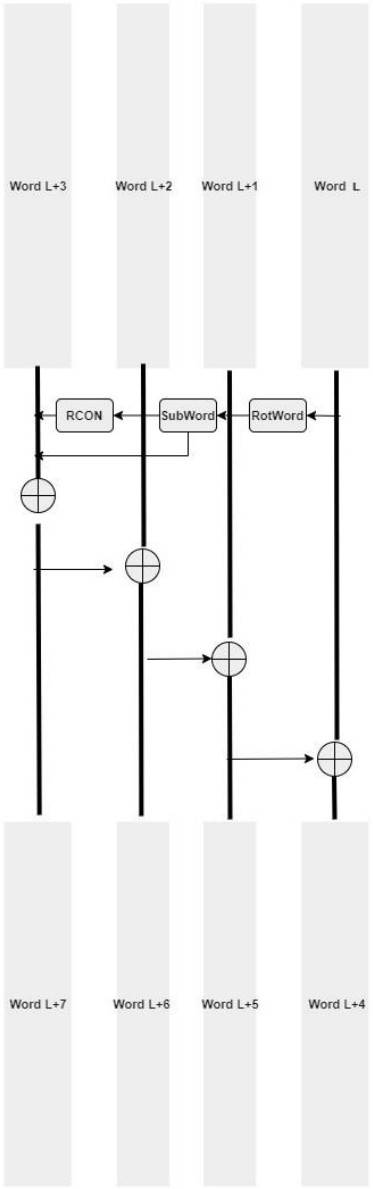


Figure 2.3.3.1: Key Expansion algorithm

Then the same word goes into the Sub Word block which called also S-BOX block.

	y															
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
x 7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figure 2.3.3.4: S-Box Table

Where X is the most significant 4 bits (equivalent to one hexadecimal) and Y is the least significant 4 bits, then the output of state round key After entering Sub Word block is shown in Figure 2.3.3.5.

After that the same word goes into the last block called RCON. RCON is a fixed-value word and this value depends on the round used in. the RCON table is shown in Figure 2.3.3.6. where j represents the round number and Rcon[j] represents the fixed-value word used in this round.

Finally, output word of Sub Word, RCON [j] and the most significant word of the present round key all are xored together to get the most significant word of the upcoming round key as shown in Figure 2.3.3.7. let's assume we want to get 1st round key so we use Rcon[1] word.

2B	28	AB	8A
7E	AE	F7	84
15	D2	15	EB
16	A6	88	01

Figure 2.3.3.5: State after RotWord

j	Rcon[j]	j	Rcon[j]
1	[01,00,00,00]	6	[20,00,00,00]
2	[02,00,00,00]	7	[40,00,00,00]
3	[04,00,00,00]	8	[80,00,00,00]
4	[08,00,00,00]	9	[1b,00,00,00]
5	[10,00,00,00]	10	[36,00,00,00]

Figure 2.3.3.6: RCON Table

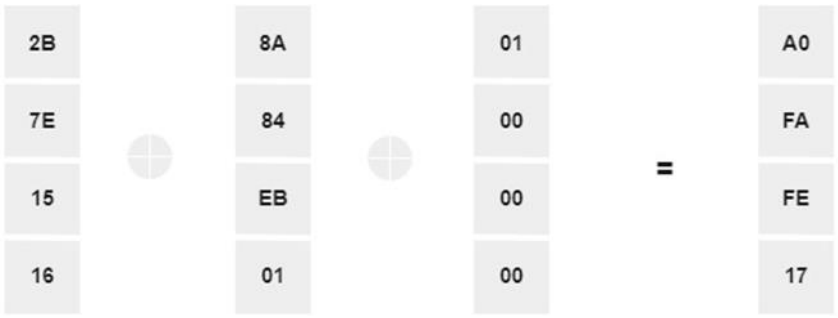


Figure 2.3.3.7: The most significant word of upcoming round key

The 2nd most significant word of the upcoming round key (W2), we can get by XORing the 1st most significant word of the upcoming round key and the 2nd most significant word of the present round key like illustration shown before in Figure 2.3.1.1. and similarly to W1 and W0 of the upcoming round key as shown in Figure 2.3.3.8.

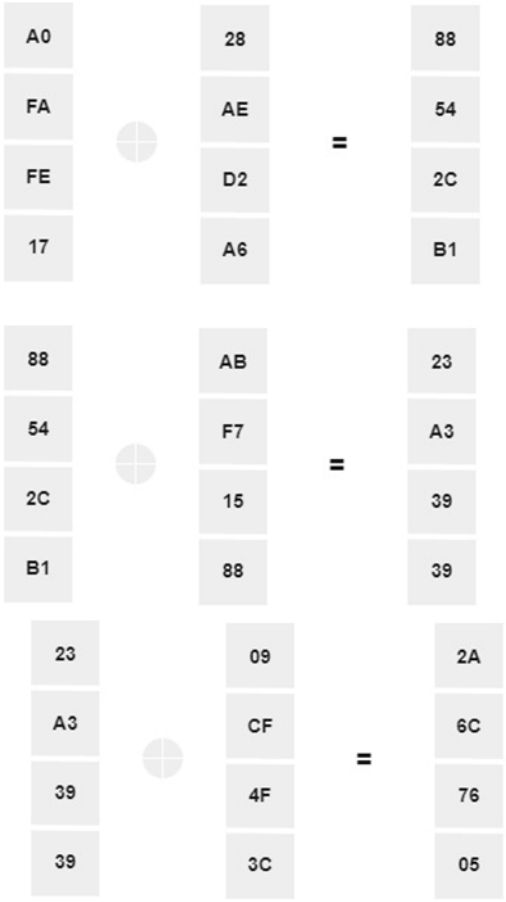


Figure 2.3.3.8: W2 & W1 & W0 of the upcoming round key

Generating the other round keys happens by repeating all the previous operations till round 10 in the same order because we use AES-128 which consists of 10 rounds as shown in Figure 2.3.1

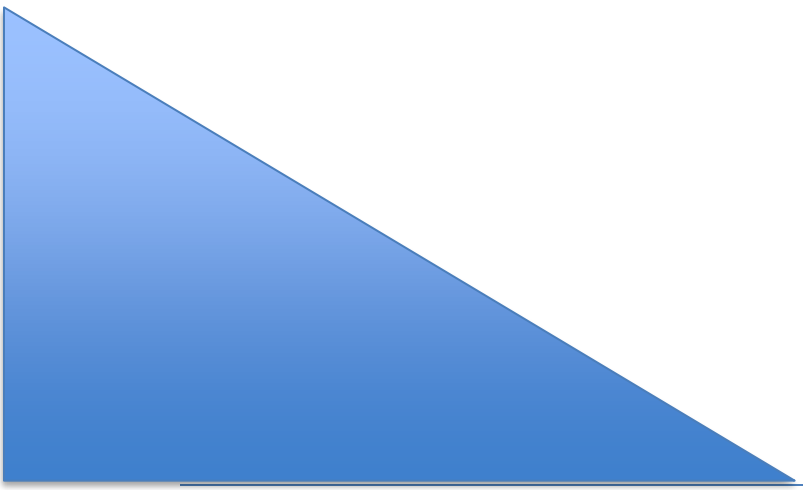
Now we can write the upcoming round key which shown in figure 2.3.3.9

A0	88	23	2A
FA	54	A3	6C
FE	2C	39	76
17	B1	39	05

Figure 2.3.3.9: State of the upcoming round key

Chapter 3: Implementation

- **Transmitter**
- **Receiver**
- **AES**



3.1 Transmitter

3.1.1Bit to symbol

The bit to symbol block consists of 4 shift registers and the symbol is ready after 4 clock cycles. The I/O specifications are shown in table 3.1.1.1. And the I/O diagram is shown in figure 3.1.1.2. The implementation of this block is shown in figure 3.1.1.3.

Pin name	Type	Size	Description
CLK	In	1	clock.
RST	In	1	reset.
EN	In	1	Enable signal for the block.
Data_in	In	1	Serial data in.
Data_out	Out	4	Output symbol.

Table 3.1.1.1. I/O specifications.

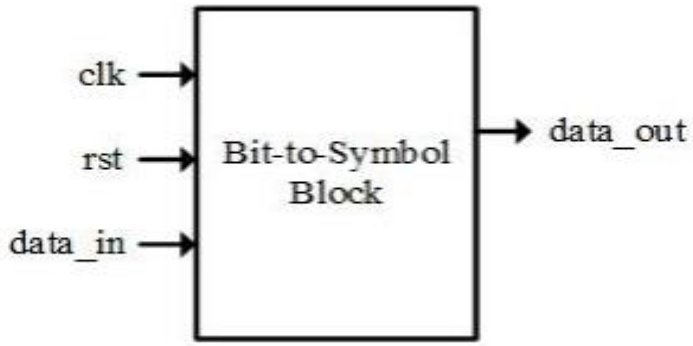


Figure 3.1.1.1. I/O diagram.

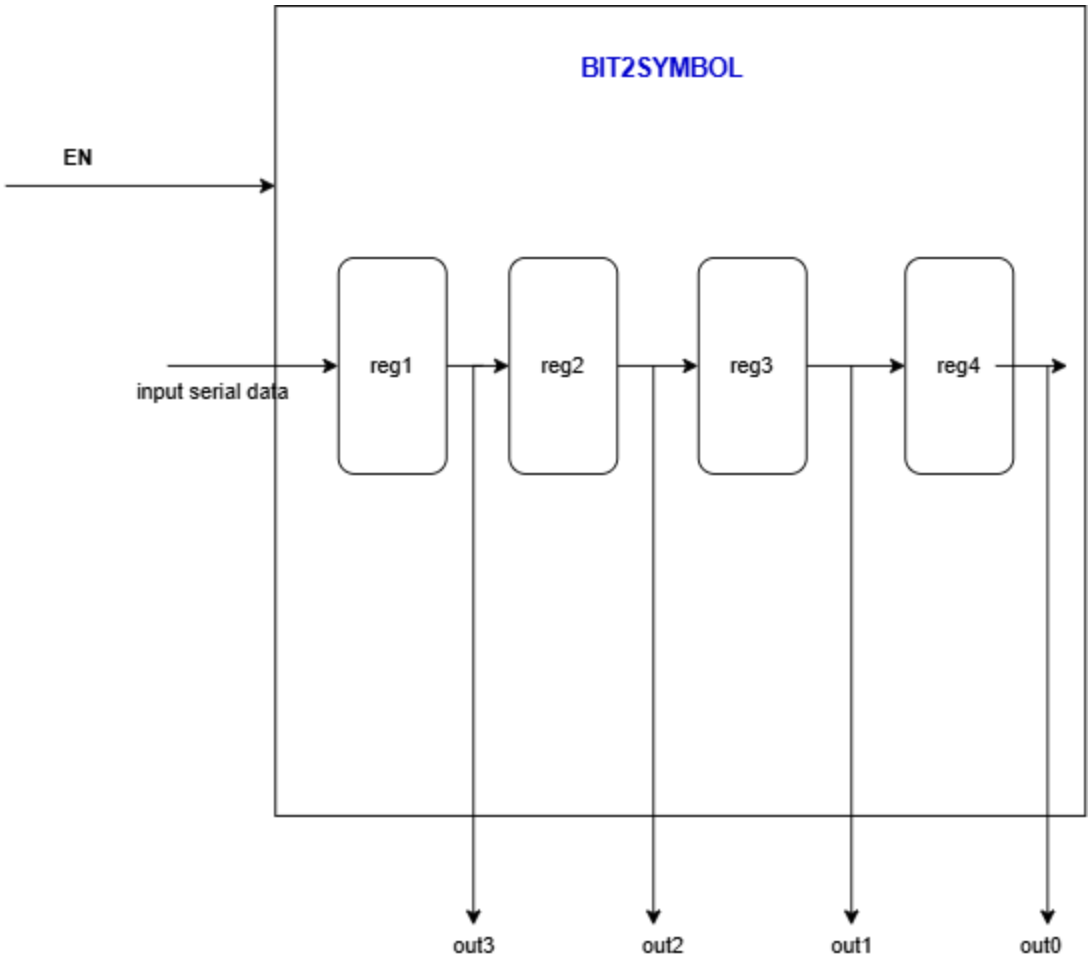


Figure 4.1.1.2 block implementation.

3.1.2 Symbol to chip

This block takes the input symbol and maps that symbol to one of 32 chip PN sequences. It consists of LUT for the input symbol and based on it chooses the output. The I/O specifications are shown in table 3.1.2.1. And the I/O diagram is shown in figure 4.1.2.2. The implementation of this block is shown in figure 3.1.2.3.

Pin name	Type	Size	Description
CLK	In	1	clock.
RST	In	1	reset.
EN	In	1	Enable signal for the block.
Data_in	In	4	symbol in.
Data_out	Out	32	Output sequence.
Data_valid	Out	1	Output valid

Table 3.1.2.1. I/O specifications.

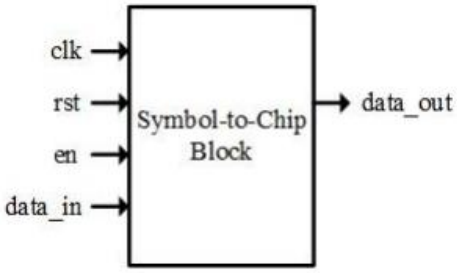


Figure 3.1.2.2. I/O diagram.

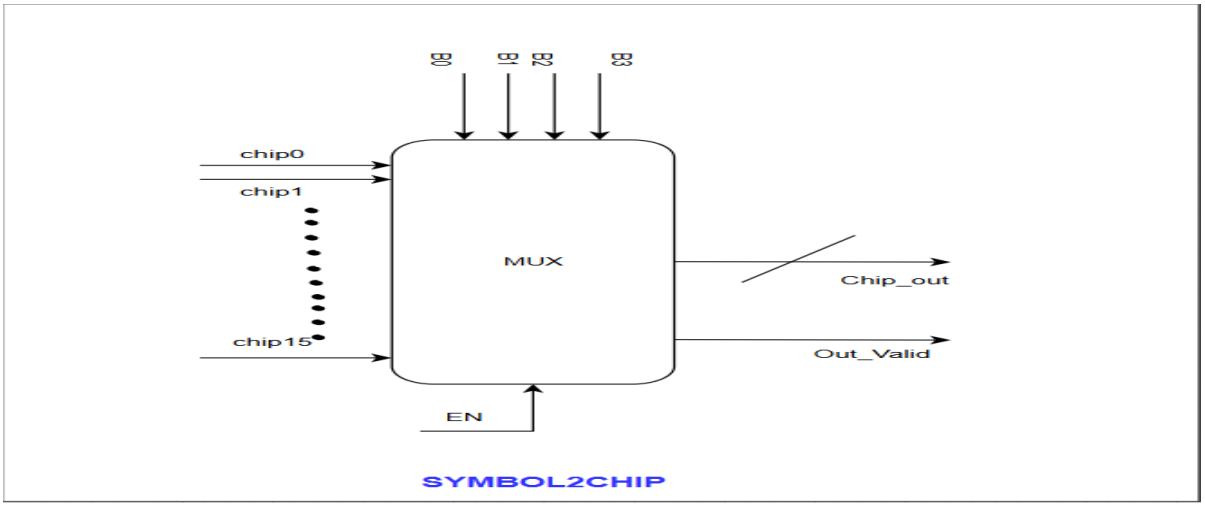


Figure 3.1.2.3 block implementation.

3.1.3 Modulator

The chips are driven to the input of a parallel to serial block. This module works like a demultiplexer. It separates the odd indexed bits and even indexed bits into the different channels which affects the transmission of data in the wireless medium positively. This process is named as I-Q channel separation. O-QPSK modulation is implemented in this block.

This block consists of a serializer and T flip flop to control the clock of even and odd output registers.

The I/O specifications are shown in table 3.1.3.1. And the I/O diagram is shown in figure 3.1.3.2. The implementation of this block is shown in figure 3.1.3.3.

Pin name	Type	Size	Description
CLK	In	1	clock.
RST	In	1	reset.
EN	In	1	Enable signal for the block.
Data_in	In	32	chip in.
Even_Data_out	Out	1	Even bits output.
Odd_Data_out	Out	1	Odd bits branch.
Even_Data_valid	Out	1	Even branch valid
Odd_Data_valid	Out	1	Odd branch valid

Table 3.1.3.1. I/O specifications.

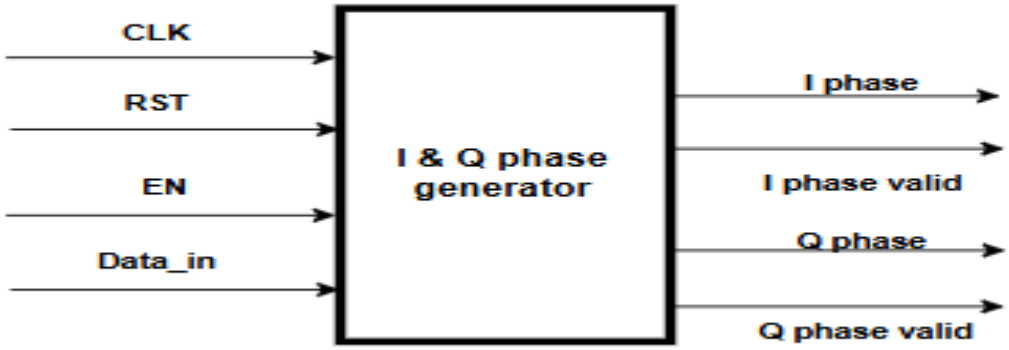


Figure 3.1.3.2. I/O diagram.

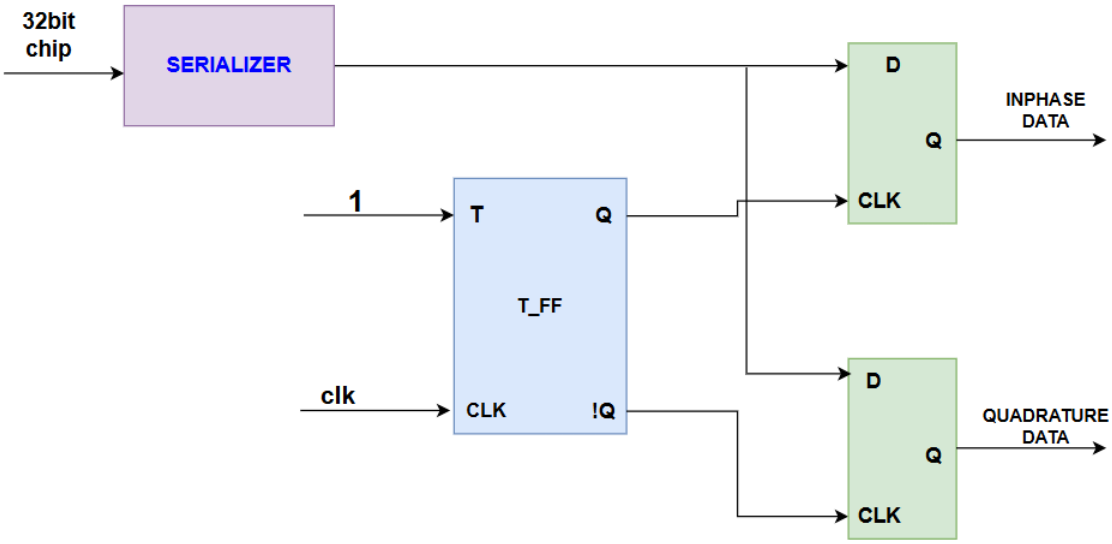


Figure3.1.3.3 block implementation.

3.1.4 Pulse shaping

Pulse shaping is the last block in the transmitter chain. It consists of two modules one for even bit streams and the other one for odd bit streams. Each bit consists of 10 samples of fixed-point numbers; each number is 10 bits. 1 bit for the sign, 1 bit for the real part and the last 8 bits is for the fraction part. Two different constant arrays, one for bit one and one for bit zero, are initialized before synthesis. They include the sampling values for a sine wave. If the block gets the data '1', it gives the values in the array for bit one to the output and does the same thing for the bit zero.

The samples for each symbol make together the half sine shape. The samples and its fixed-point implementation for the one and zero are shown in table 3.1.4.1.

For even module, the I/O specifications are shown in table 3.1.4.2. And the I/O diagram is shown in figure 3.1.4.3.

For odd module, the I/O specifications are shown in table 3.1.4.4. And the I/O diagram is shown in figure 3.1.4.5.

One		Zero	
Value	Fixed point	Value	Fixed point
0	0000000000	0	0000000000
0.309	0001001111	-0.309	1001001111
0.588	0010010110	-0.588	1010010110
0.809	0011001111	-0.809	1011001111
0.951	0011110011	-0.951	1011110011
1	0100000000	-1	1100000000
0.951	0011110011	-0.951	1011110011

0.809	0011001111	-0.809	1011001111
0.588	0010010110	-0.588	1010010110
0.309	0001001111	-0.309	1001001111

Table 3.1.4.1. The samples and its fixed-point implementation.

Pin name	Type	Size	Description
CLK	In	1	clock.
RST	In	1	reset.
EN	In	1	Enable signal for the block.
Data_in	In	1	Even bits stream.
Even_samples_out	Out	10	Even samples output.
Even_Data_valid	Out	1	Even branch valid

Table 3.1.4.2. I/O specifications.

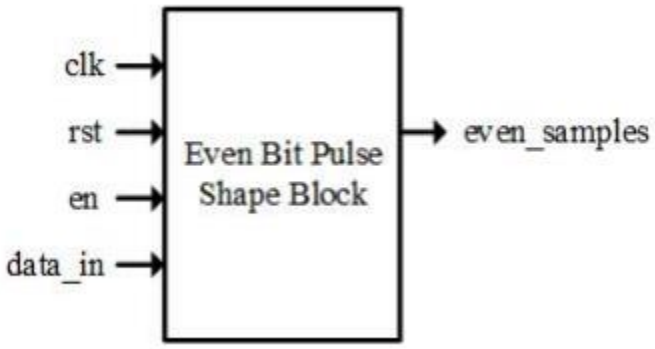


Figure 3.1.4.3. I/O diagram.

Pin name	Type	Size	Description
CLK	In	1	clock.
RST	In	1	reset.
EN	In	1	Enable signal for the block.
Data_in	In	1	Odd bits stream.
odd_samples_out	Out	10	Odd samples output.
Odd_Data_valid	Out	1	Even branch valid

Table 3.1.4.4. I/O specifications.

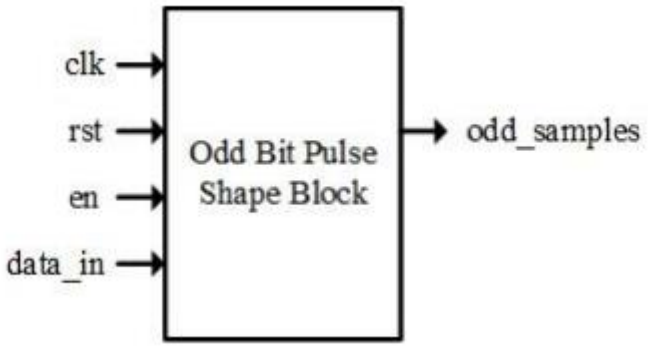


Figure 3.1.4.5. I/O diagram.

3.2 Receiver

3.2.1 Decoder

Fixed Point

To estimate the number of bits in the integer and fractional parts of the fixed-point operations, we need first to estimate the minimum resolution of the floating-point numbers of the system. The system numbers are represented in the samples of the input signals which means the resolution of the ADC block which will sample the analog filtered signal, beside the neural network parameters which is the weights and the biases in each layer of the three layers.

To know the minimum resolution of floating numbers we need to use the simulation to measure the error percentage at different floating resolutions. Originally the resolution of the parameters means the parameters values will be like '0.18304272'. By using the original resolution, we will invoke the neural network with 200 signals with random SNR from -13dB to 12dB and count the number of the wrong predictions of the neural network.

Figure 3.2.1.1 shows the number of errors versus the decreasing resolution. The number of errors seems to be almost constant till and after that the number of errors shoots to bigger numbers. So, we will stand on that. By this point we can determine the number of fractional bits in the fixed-point representation by using the logarithm of base 2 of which means bits. So, the number of the fractional bits is 10, now what about the integer part? Now let's illustrate something, in this system it's very rare to find any large number, where a large number means a number has a value more than three as an integer part, as well as almost all the weights are less than one so, the multiplications will not make the integer value explode. So, it's more than safe to set the number of the integer part to be only 3 bits. So, the basic length of the fixed-point numbers is: 1 sign bit, 3 integer bits, 10 fractional bits. The fixed-point width is 14 bits.

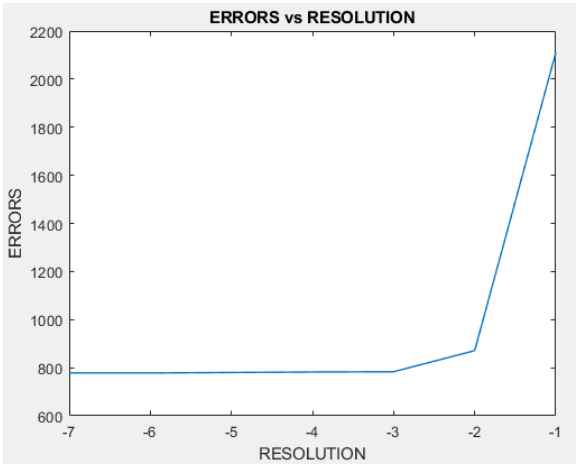


Figure 3.2.1.1.

Kogge Stone Adder

Due to the intensive nature of the computations of the neural network applications, we look forward to limiting the delay of these computations as much as possible. And to reach our goal we need to guarantee the speed of our blocks and this speed comes by the parallelism of the architectures. So, we need to use hardware architectures which provide these parallelism approaches in the addition operations which are repeated very frequently in our application.

The fastest adders that deal with large widths are the parallel prefix adders, one of these adders is the Kogge Stone Adder. These adders do not deal with the operands directly as it firstly computes the generate and the propagate signals of every two corresponding bits in the two operands. From the table in figure 3.2.1.2 we can determine how to get the generate and the propagate signals logically, where 'x' is the bit number of the corresponding two bits of the two operands.

A	B	Cin	Cout	S	G	D	P
0	0	0	0	0	0	1	0
0	0	1	0	1	0	1	0
0	1	0	0	1	0	0	1
0	1	1	1	0	0	0	1
1	0	0	0	1	0	0	1
1	0	1	1	0	0	0	1
1	1	0	1	0	1	0	0
1	1	1	1	1	1	0	0

Table 3.2.1.2: Full Adder truth table with Generate and Propagate.

After that there are a number of stages which equal to where 'N' is the width of the operators as shown in figure 3.2.1.3. So, the complexity of the Kogge Stone Adder is. Now what is the operation that is done in this stage? Each stage consists of a number of operations called **Dot Operation.** This operation takes four inputs and gives two outputs. The inputs are two different generate signals and two different propagate signals and the two outputs are a generate signal and a propagate signal. Logically the dot operation is two and gates and one or gate. We have to mention a crucial point about our kogge stone adder design, that we designed this adder with the ability to have a carry in as input but does not have the ability to produce a carry out. The reason why we did that is the nature of the numbers that we deal with in that application, which means that there is no chance add two numbers and the result needs more than the specified number of integer bits to be written to, in addition to that we have to keep the flow of the bits with minimum width as much as possible.

We designed two kogge stone adders with different widths: 14-Bit Adder and 28-Bit Adder.

The 14-bit adder is to add 14-bit numbers which is the general width of everything, while the 28-Bit adder is to add the outputs of the multipliers existing in the system.

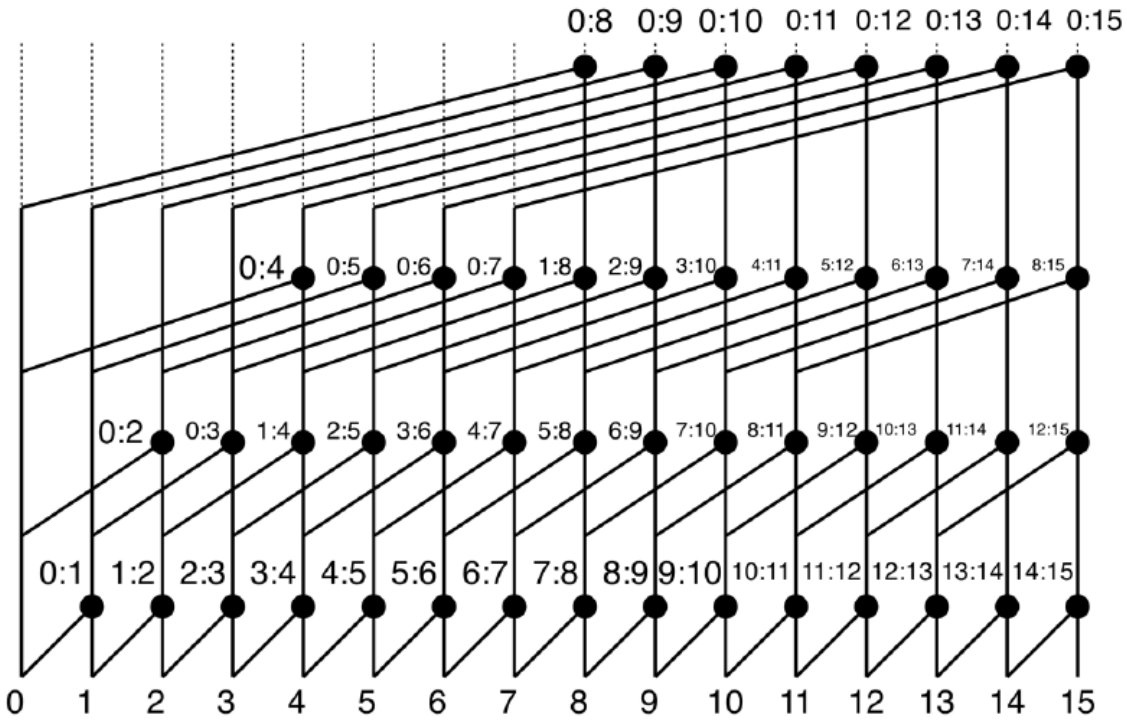


Figure 3.2.1.3: 16 Bit Kogge stone adder.

Wallace Tree Signed Multiplier

The Wallace Tree Multiplier is an efficient hardware implementation of a digital multiplier, designed to multiply two binary numbers quickly, it uses a technique that reduces the number of partial products and accelerates their summation, making it particularly useful in high-speed arithmetic circuits like our application.

Let's illustrate the mechanism of this multiplier and how it uses the parallelism to generate the results. Let's start with the parietal products which is the result of multiplying each single bit of the multiplier operand by the multiplicand, logically we do ANDing between a bit from one operand and the whole other operand. Then we have to do two essential steps, firstly we have to shift each partial product left by a number of times equal to the location of the multiplier bit of the multiplier operand. The second step which is essential to make this multiplier a signed multiplier is to do the sign extension which is done by extending the sign bit after the left shifting. Now how many times do we need to extend the sign bit? After the

shifting we have to extend the sign bit till the total number of bits in the partial product reach 2N, where 'N' is the width of the input operands like in Figure 3.2.1.4.

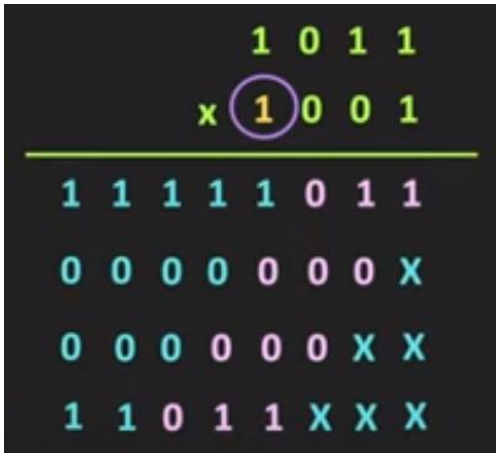


Figure 3.2.1.4: Example about 4-Bit signed multiplication.

By reaching this point, the rest of the multiplication operation is addition. But we aim to do the addition in a parallel manner to accelerate the multiplication operation as much as possible. The Wallace Tree method organizes the summation of these partial products into a tree structure, which significantly reduces the delay by minimizing the number of sequential addition steps. And this is done by the following, instead of summing the partial products in a straightforward manner, the Wallace Tree Multiplier uses carry-save adders (CSAs). CSAs generate partial sums and carry values without immediately propagating carries, allowing multiple additions to be performed in parallel.

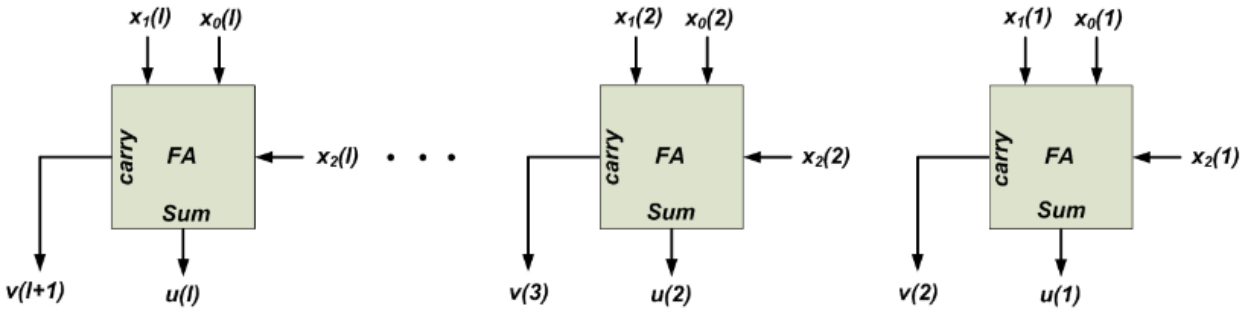


Figure 3.2.1.5: L-bit Carry Save Adder.

The CSA in figure 3.2.1.5 takes three inputs of width L, and produces two outputs of width L, one of these outputs is the **Sum** while the other is the **Carry**. Note that the carry of each full

adder is not connected to the next full adder. So, as soon as the inputs are ready the Sum and the Carry produced at once, and every sum bit is independent of the carry of the last full adder.

Now we have the partial products and the CSA adder which takes 3 inputs at a time. So, now we need to spread the partial products on the adders, but how many partial products do we have? We know that the input width is 14-bit so we have 14 partial products. So, in our tree the first stage of CSA adders will contain 4 adders with 12 partial products.

The difference between signed and unsigned multipliers is two points, one of them we talked about earlier and it was the sign extension, the second point is that the last partial product must be subtracted and not added. The reason behind the subtraction is if the multiplier was positive so, its sign bit is zero so the last partial product is zeros so, adding or subtracting it is the same. But if the multiplier was negative, the last partial product is a number multiplied by a negative sign so, it's a negative value so, it will be subtracted and by that whether the multiplier was negative or positive the last partial product will be subtracted.

So, beside the 4 adders in the first stage there is a new adder whose operands are: PP[12] and the ones complement of PP[13] and value one (28'b1).

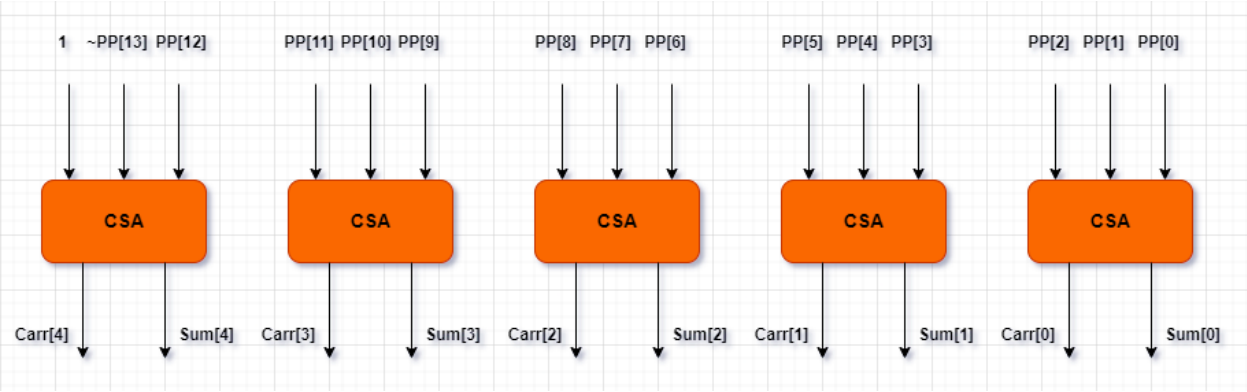


Figure 3.2.1.6: First stage of the Wallace Tree Multiplier.

Now all we have to do is just to add all the Sums and Carries in the rest of the tree. Note that the width of the partial product is double the input width which means 28-bit width.

The final CSA adder will produce two outputs Sum and a carry, this sum and carry will be summed using regular Ripple Carry Adder to produce the output of the multiplier.

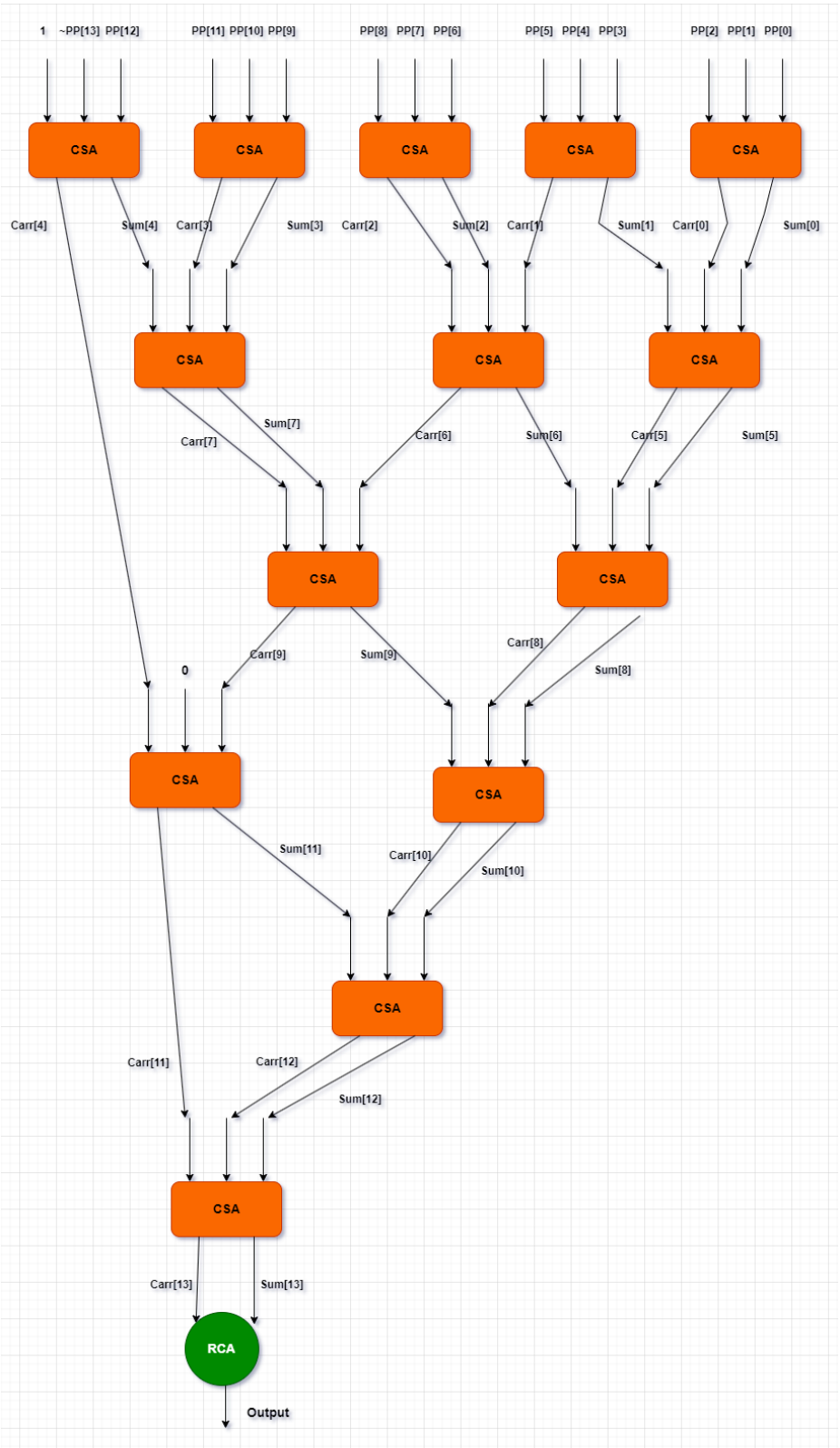


Figure 3.2.1.7: The CSA tree of the multiplier.

Nonlinear Functions

There are two nonlinear functions in the LSTM node, they are the **tanh** function and the **sigmoid** function. To implement any nonlinear function in a digital design, the most efficient method is to use a lookup table which contains samples from the function values. These samples are ordered in the memory in some way so that the address of the output sample is the corresponding input value that produces this output.

So, there is a tradeoff between the size of the memory and the accuracy of the implemented function. But there is another way to implement a nonlinear function in a relatively small memory, this is done by not storing the output samples directly, but by playing the interpolation game. The interpolation states that if I know the outputs of two graphically adjacent inputs and I have a third input in between so I can deduce its output with accuracy depends on the difference between the known two inputs.

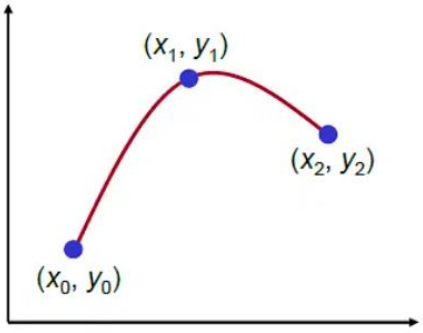


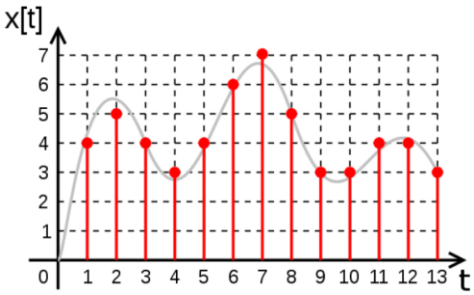
Figure 3.2.1.8.

In figure 3.2.1.8, if y0 and y2 are known for their inputs so, we can compute y1 using the following equation: $y_1 = y_0 + \frac{(x_1 - x_0)}{(x_2 - x_0)}(y_2 - y_0)$.

And by representing this equation in a simple form like that: $y_1 = C_1 x_1 + C_2$ where $C_1 = \frac{(y_2 - y_0)}{(x_2 - x_0)}$ and $C_2 = y_0 - x_0 \frac{(y_2 - y_0)}{(x_2 - x_0)}$. So, for the pair x2 and x0, any input between them we can get its output using this equation. So, after sampling the nonlinear function by the appropriate sampling rate like the case in figure 3.2.1.9, and for each pair of samples we get the two constants, these constants are the values to be stored in the lookup table of the function.

Figure 3.2.1.9.

Now there is a very important question, how to order the constants in a way that allows the input to be the address directly? Now if the function has a step which means it's constant from 0 to 1 and from 1 to 2 so, we can easily look just at the integer part of the input and store the constants of range 0 at address 0 and so on. But our case is different so, to think about this problem we can decide firstly which bits range of the input we need to make it the address, and what about the sign bit how will we deal with it in this problem.



If we think about it, we will deduce that we have to make the integer part of the fixed-point number included in the address, as that will decide how far the implemented function will accept inputs and we need to get the maximum of that. While the chosen fractional bits to be included in that address will decide the difference between two samples or in other words the resolution of the lookup table, the more fractional bits the more accuracy.

We chose the memory to be 64 locations which is a small number as there are a lot of function blocks to be instantiated in the system. This means that the address is 6-bits width so we will look at the integer bits and the first 3 fractional bits.

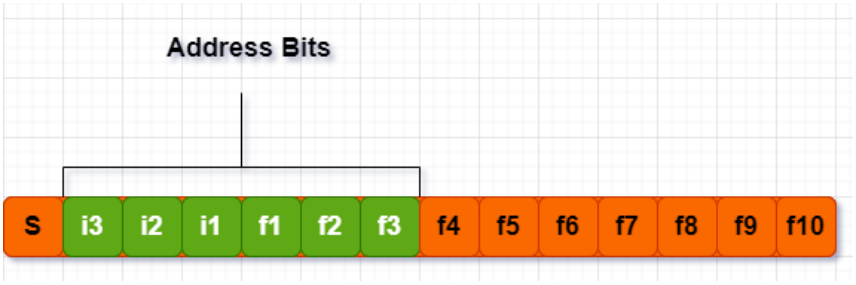
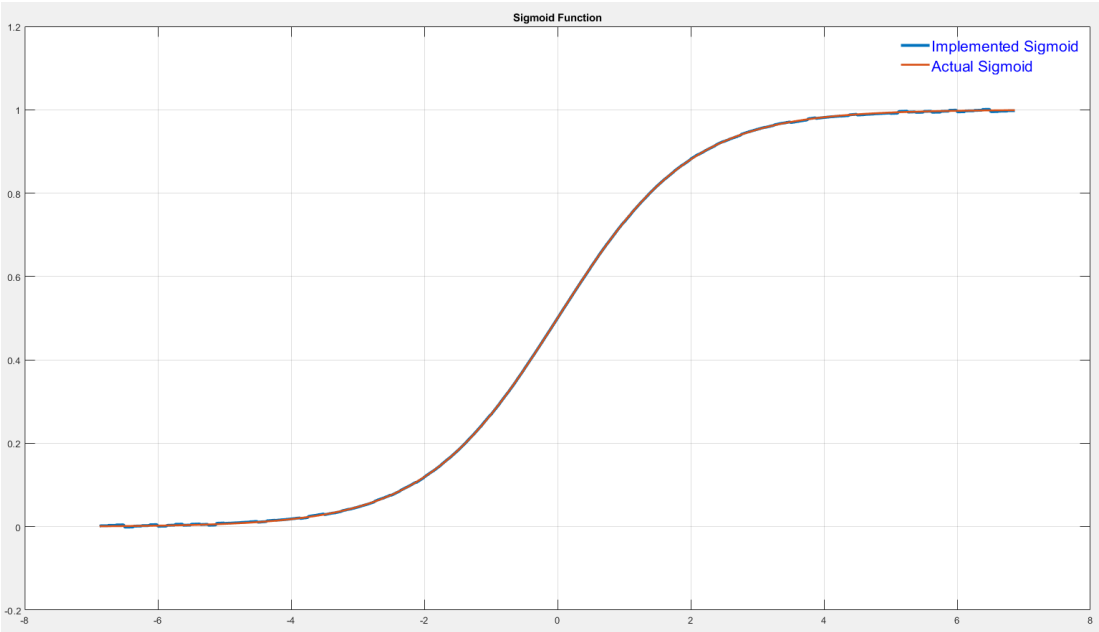


Figure 3.2.1.10: Address bits.

Fortunately, the two functions have symmetry around the Y-axis so we can exploit this feature and store only the positive part. So, if the input is positive so the operation is done directly, otherwise if the input is negative, we get its 2's complement and feed it to the table and according to the symmetry of the function we can adapt the final value to be the right output of the function.

If it was a sigmoid function, the output will be 1 - the function output. While if it was a tanh

function, the right output will be the negative value of the function's output or logically the right output will be the 2's complement as the tanh function is an odd function.



.

Figure 3.2.1.11: Sigmoid Function.

In figure 3.2.1.11, we can see the quantization error between the implemented and the actual sigmoid. We need to illustrate that there are two errors we have to consider, the first is the error due to interpolation, while the second is due to the quantization. The error due to interpolation is plotted in figure 3.2.1.12 in blue, this error has its significant values in the range [0:4], while the quantization error has its significance in the range [4:7] as shown in figure 3.2.1.12 in red. The reason behind that is, in interpolation error in the outer range the actual function is almost constant so the inputs within the samples will nearly take the same values, while in the inner range the function has its greatest slope so, the inputs within the samples would vary in the error value from the beginning of the its interval to the end of it, that is why it takes the shape of arcs. While in quantization error, the quantization tracks the changes of the function in its inner range with the high slope, but when the function becomes nearly constant the quantized function must follow its steps directly without tracking the actual function.

All of that is applicable to the **tanh** function and the **sigmoid** function, but to follow the

symmetry of these functions the behavior with the **tanh** differs from the behavior with the **sigmoid** when the input is a negative value.

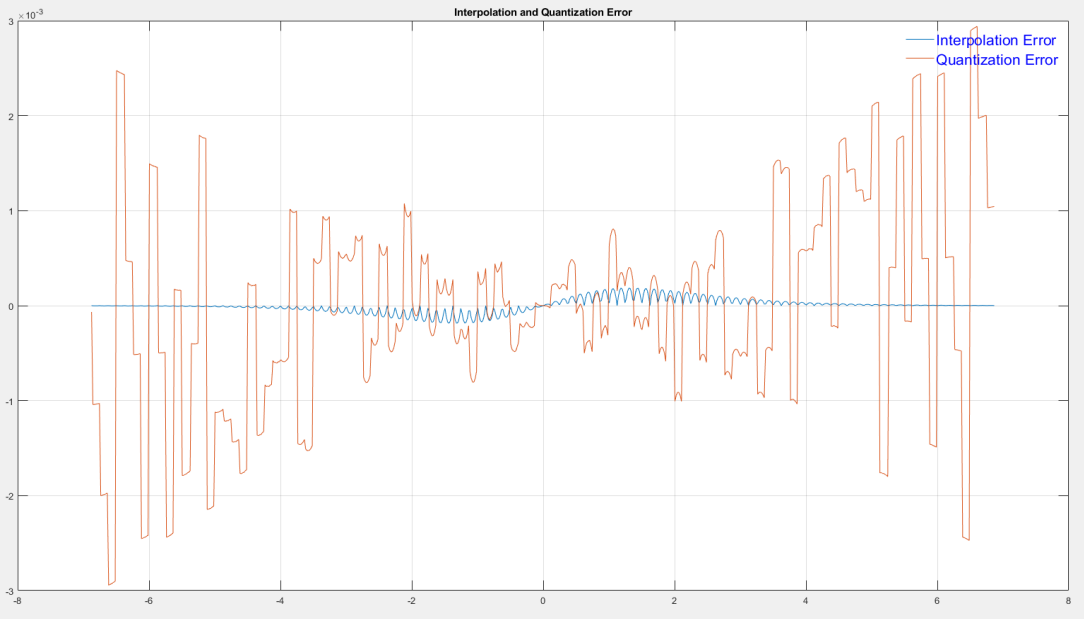


Figure 3.2.1.12: Interpolation and Quantization errors.

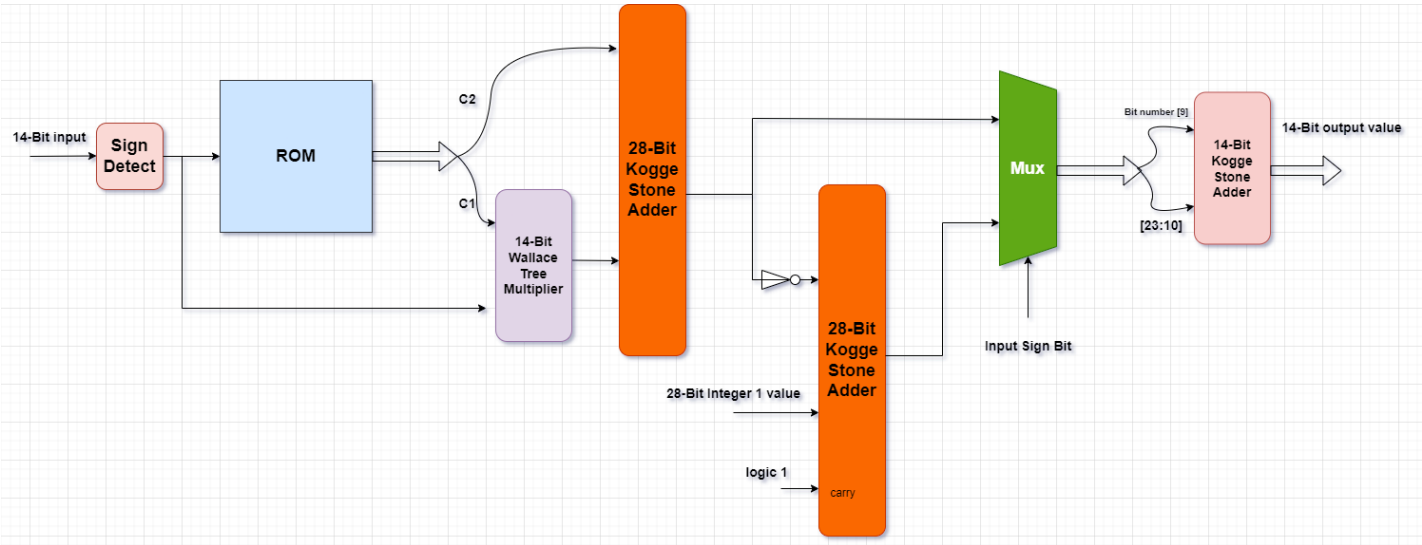


Figure 3.2.1.13: Function Architecture.

LSTM Layer Operations

Since we have two LSTM layers, so the operations done by each layer are the same as they are of the same type, but the only difference we have to consider is that each node in the second LSTM layer has five inputs from the first LSTM layer and five hidden state signals from the last time step. While each node of the first layer has a single input which is the input sample and five hidden state signals from the last time step.

If we think about the implementation of these two layers, we can say that we will make a design for a single LSTM node and instantiate it many times needed to construct the two hidden layers. This idea has a lot of problems as for each instantiation we have to consider that every instantiated node has specific weights and biases which will be exhausting to write the parameters of each node in the instantiation as there are many weights and how can us pass these values to a new instantiation in hardware description languages. And another reason behind the complexity of doing that is, that the API of the machine learning platforms does not provide the parameters of its model node by node, no this is wrong, they provide big matrices representing the whole layer and other matrices to represent another layer and so on. So, in our design we will deal with layers not with nodes.

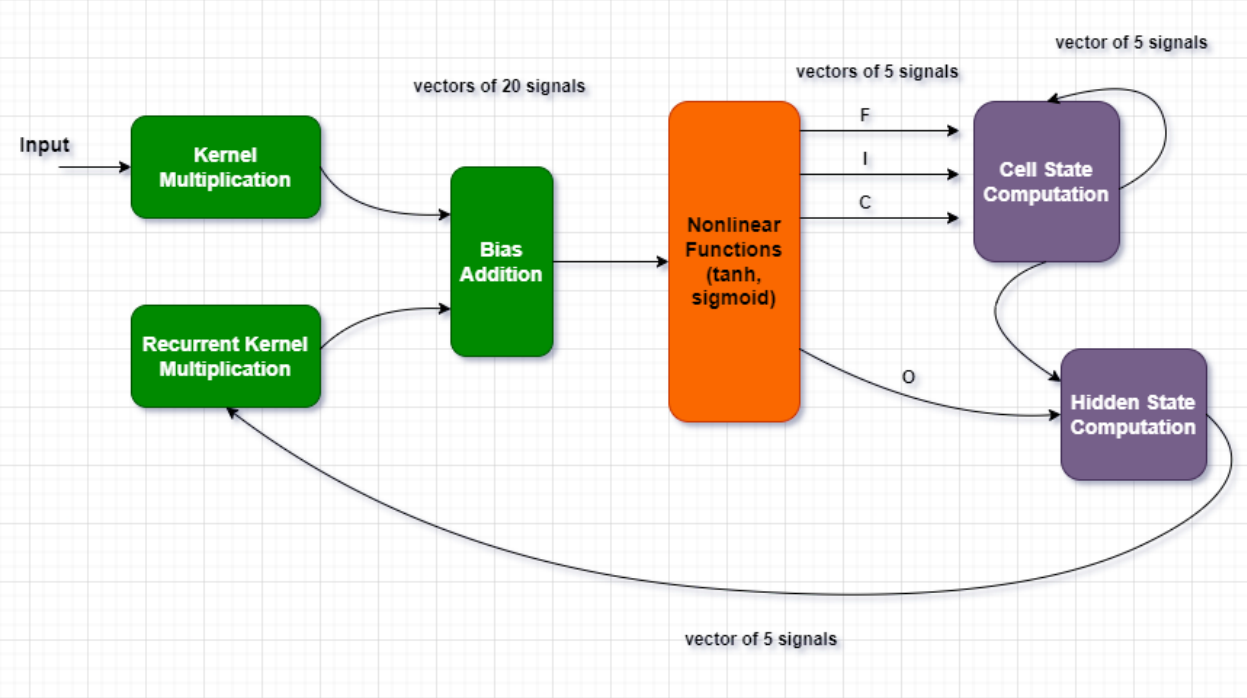


Figure 3.2.1.14: LSTM layer architecture.

The figure 3.2.1.14 illustrates the architecture of the LSTM layer. We need to mention that this figure shows specifically the first hidden layer, and to imagine the second hidden layer so we have to consider the input to be a vector of five signals. We also need to mention that the **TensorFlow** library provides three matrices to each LSTM layer (Kernel matrix, Recurrent Kernel matrix, Bias matrix). Now let's illustrate each block separately.

Kernel Multiplication

In that block the input vector is multiplied by Kernel matrix. The first layer input vector size is [1x1] and the Kernel matrix of this layer is [1x20]. Note that the LSTM node has four gates with four kernel elements and there are five nodes per this layer and that is the reason why the kernel matrix is [1x20]. So, the multiplication of these matrices would result in a matrix of size [1x20].

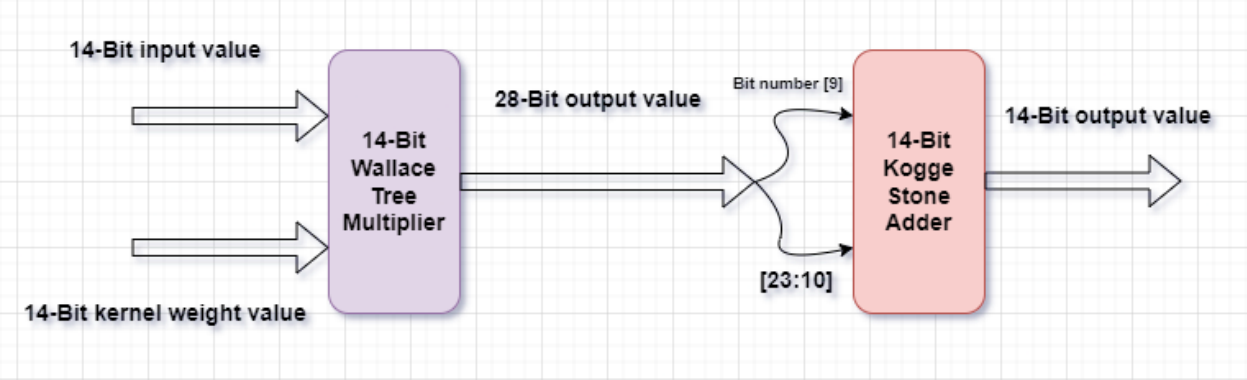


Figure 3.2.1.15: The building block of the Kernel Multiplier unit.

The dataflow illustrated in figure 3.2.1.15 will be instantiated 20 times to construct the kernel multiplier block. Why do not we make the 28-Bit output the direct output? Because the output of this block will pass for many times over other multipliers and adders, we have to confine the dataflow to a specific width which is familiar with the coming block and this width the 14-Bit value. Which bits of the 28-Bit output should be passed to the next blocks? The multiplication operation duplicates the fixed-point fields so, the first 20 bits are fraction bits we need the most significant ten bits [19:10], and the least significant three integer bits [22:20], and a sign bit which is represented in the rest of bits due to the sign extension. Why do not we directly take the desired bits? To achieve high accuracy results, we need to approximate the selected value to the nearest number and do not just take the desired bits directly. The approximation concept is just like the approximation with real numbers. This

approximation is done by looking at the last unselected bit on the right side. Let's look at the following example: the following number is a 28 bit number {xxxx,[0,001, 0000000000], 1xxxxxxxxx}. Bit number nine is one so, the selected 14-Bits will be as follows [0,001, 0000000001]. This operation has a great impact on the accuracy of the blocks. Note that after each multiplication in the system we have to do this operation.

Recurrent Kernel Multiplication

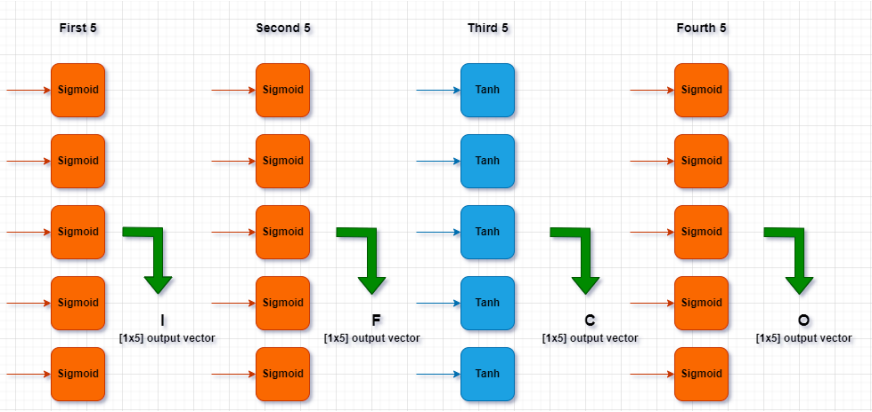
This block is the same for the two layers. Because in the two layers the hidden state is a vector of size [1x5] so the recurrent kernel matrix has the size [5x20] so, the output of this block is a matrix or we can call it a vector of size [1x20]. Note that this block operates in parallel with the kernel multiplication block in the two layers. All multiplications are done using Wallace Tree multiplier and all the additions are done using Kogge Stone adder.

Bias Addition

This block function is to take the two output vectors of the kernel multiplication block and the recurrent kernel multiplication block, adding them to each other then adding the bias value associated with the layer of operation. The bias matrix size is [1x20] so, we add three vectors of size [1x20] which will produce a vector of size [1x20].

Nonlinear Functions

This block is a wrapper which wraps all the gates nonlinear functions of each node in the layers, where each node contains four functions [3xSigmoid, 1xTanh]. So, this wrapper has 5x[3xSigmoid, 1xTanh]. This block has a vector of size [1x20] as input, originally this vector is ordered like that the first five inputs will path through five sigmoid functions "**I"** and the second five "**F"** are the same and also the fourth five **"O"**, while the third five "**C"** will be passed over a tanh function.



Cell State Computation Block

This block is in charge of computing the cell state vector which carries the long-term information. The inputs of that block are the “I”, “F”, “C” which are the outputs of the nonlinear functions block, in addition to the cell state vector of the previous time step which means that we have a feedback loop in this block. Note that the size of the cell state vector is [1x5]. Each element of the new cell state vector is computed by multiplying output of the forget gate “F” by the previous cell state, which analogues to how much of the long-term memory should I remember, then we add that with the result of multiplying the input gate “I” by the cell gate “C”. This multiplication is not a matrix multiplication, it's element by element multiplication. This would result in a vector of size [1x5] which is the new **Cell State** vector. The feedback loop is controlled by placing a register with an enable signal, this enable signal would be asserted as soon as the “I”, “F”, “C” are ready, to enable the previous cell state vector to compute the new one.

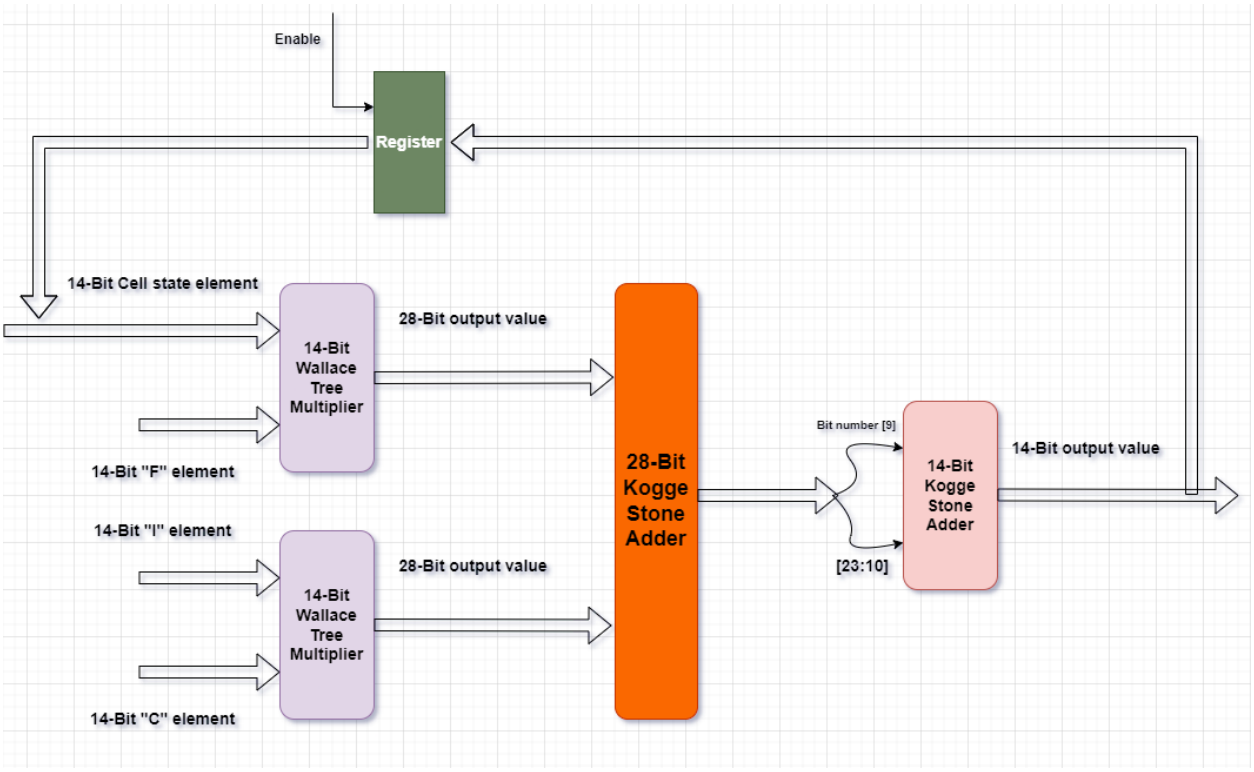


Figure 3.2.1.16: Cell State building block.

The architecture in figure 3.2.1.16 will be instantiated 5 times to produce the 5 five elements of the Cell State vector.

Hidden State Computation Block

This block is in charge of computing the hidden state vector which carries the short-term information. The inputs of that block are the immediate or the current time step cell state and the output gate values which are presented by the nonlinear functions block. The hidden state is computed by passing the cell state over a tanh function and multiplying the result by the “O” vector so the output is also a [1x5] vector. The output hidden state will be registered in a register with an enable signal to control the timing of the layer output.

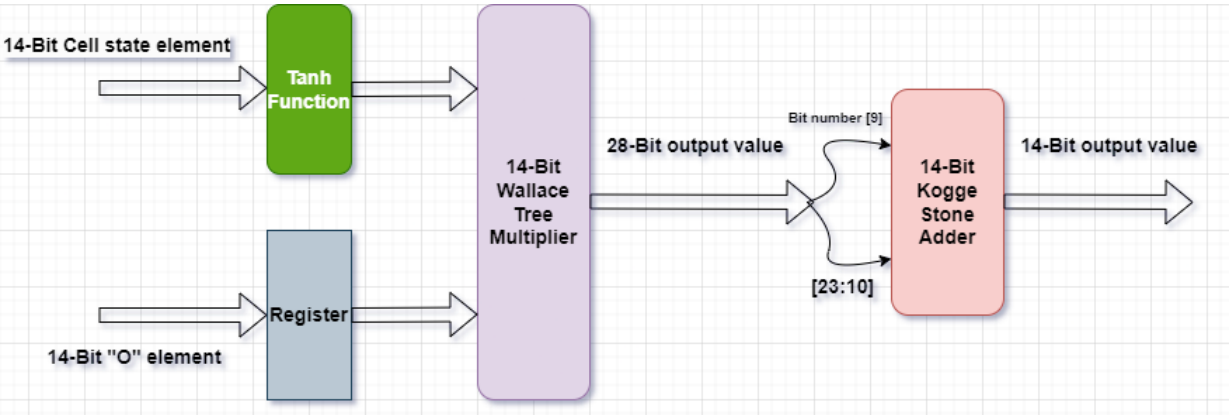


Figure 3.2.1.17: The building block of the hidden state computation block.

Controller

The controller function is to raise the enable signals at specific times to preserve the timing of the dataflow. We need to mention that after every stage is figure 3.2.1.14 the output is registered, but the connection between the Cell State block and the Hidden State block is not registered. In figure 3.2.1.17 the Tanh block has an internal register to register the address before the ROM, and because of that the input “O” is registered. So, how many cycles should we count between the active edge of accepting a new input and the active edge with the appearance of the output new Hidden state? The kernel and recurrent kernel both work in parallel so this takes one cycle, and one cycle for the bias addition block, and one cycle for the nonlinear function block, and one cycle for the tanh block in hidden state calculation and finally one cycle of the hidden state register so, the total number of clock cycles to consume one input is 5 clock cycles. The registers with enable signals are the register of the nonlinear function block which is essential to prevent the cell state value from many undesired variations, and the enable signal of the register on the feedback signal of the cell state which is essential to make the cell state unchanged till the coming iteration, and finally the register on the output hidden state to provide the new hidden state to the current layer and the next layer in the specified time. So, this controller has three output enable signals and has a triggering input signal to inform the neural network that a new receiving operation starts. This controller is the same for the two LSTM layers.

The controller is based on a finite state machine which consists of three states, the first is the

IDLE where the network is sleep and no operations are done, the second state is **Matrix Multiplication State** where the kernel and recurrent kernel multiplications and the bias addition are done which means that this state lasts for two clock cycles then a transition is done automatically to the next state which is the **Cell Hidden State** where the enable signals are activated. Figure 3.2.1.18 shows the state diagram of the controller.

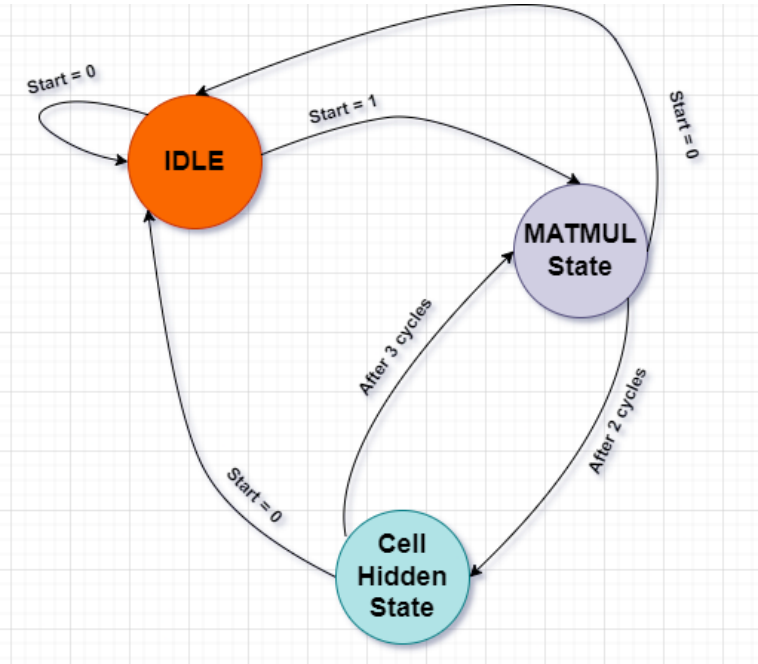


Figure 3.2.1.18: State diagram of the controller.

Output Layer

This layer is a dense layer or a fully connected layer which means that each node in this layer is connected to the five layers of the last hidden layer and it's nothing but the weighted sum of the activations of the previous layer in addition to an activation function like the sigmoid function to represent the percentage of the prediction. The network has four classes to choose between them so it has four outputs, the last hidden layer has five nodes so, this layer has to translate the five to four. This is done by multiplying the input vector which has the size [1x5] by a kernel matrix that has the size [5x4] then adding the bias vector which has size [1x4] then adding the result which has size [1x4] by four sigmoid functions and the result is the final result of the neural network. Note that this layer starts to work only when the 250 time step is over by triggering a valid signal by counting the 250 steps.

Comparator

The function of this block is to determine which of the output layer vector elements has the maximum value to make a decision about the final binary value [00, 01, 10, 11]. This is done by subtracting each element from the others using Kogge stone adders and the element without any negative output is the maximum.

3.2.2 Chip to symbol

First storing lookup table in register “chip sequence” this table was used in symbol to chip in transmitter each chip sequence consist of 32 bit so using “bit count “to count 32 bits of received sequence then start to compare the received sequence with 16 sequences we have in lookup table after that with using xnor if two bits matched then it will give 1 also each xnor have its own counter then compare between those counter until find highest counter then compare between bits to find which of the sequences, the received sequence belong to even there are mistakes in bits given in received sequence it will detect which symbol we have and store it in “symbol”

Pin name	Type	Size	Description
clk	In	1	Clock
Reset	In	1	Reset
en_receive	In	1	Enable to receive
recived_chip_seq	In	32	Received chip sequence
Symbol	Out	4	Output symbol
valid_symbol	Out	1	Counter end valid

Table 3.2.2.1. I/O specifications.

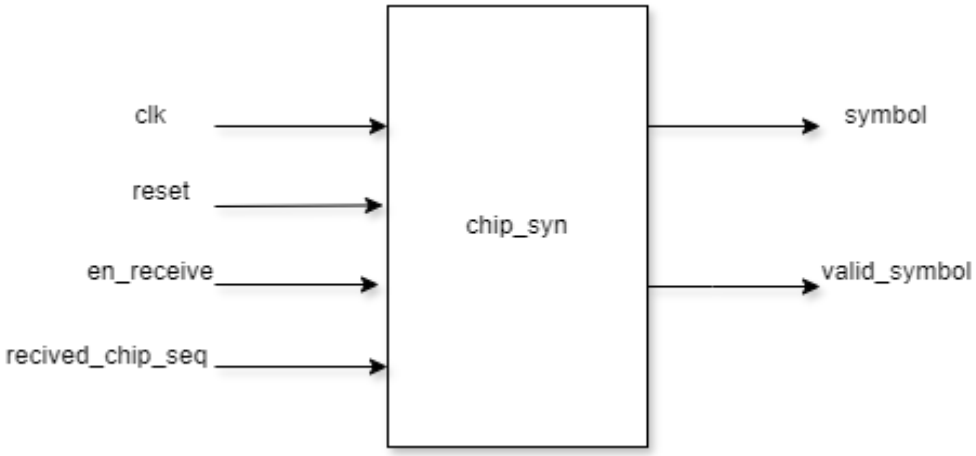


Figure 3.2.2.1. I/O diagram.

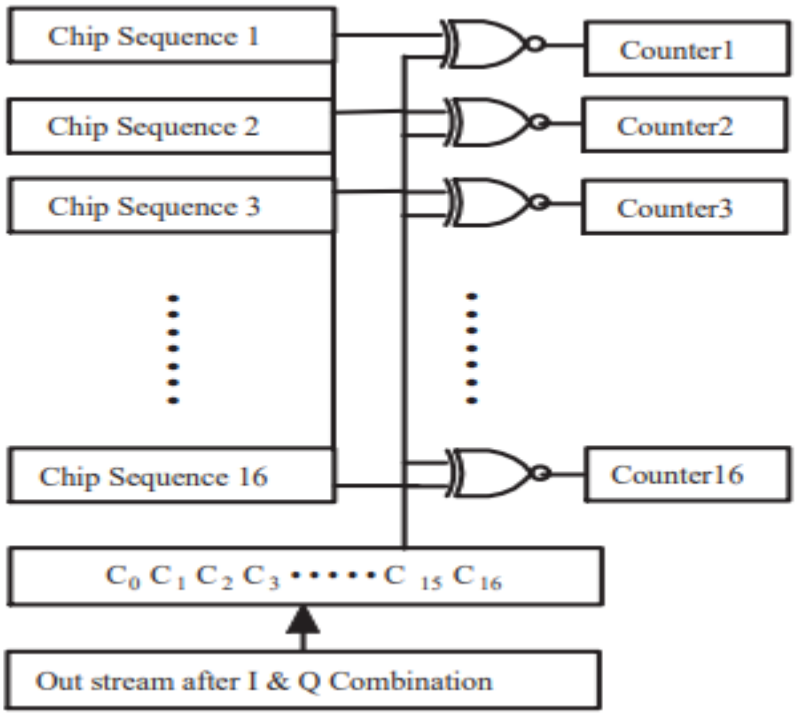


Figure 3.2.2.1: chip synchronizer implementation

3.2.3 Symbol to bit

The symbol stored in “IN” then if En is 1 the output will get out bit by bit in “Data_Out”

Pin name	Type	Size	Description
CLK	In	1	Clock

RST	In	1	Reset
EN	In	1	Enable
IN	In	4	Input symbol
Data_Out	Out	1	Output bit
Valid_Out	Out	1	Output valid

Table 3.2.3.1. I/O specifications.

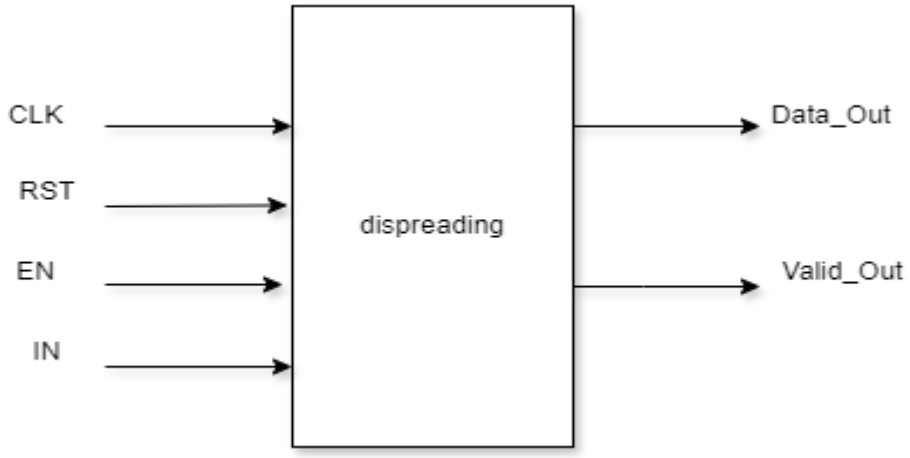


Figure 3.2.3.1. I/O diagram.

3.2.4 Frequency offset estimation

As we have a new design of the receiver using RNN instead of traditional demodulator, we don't have to separate the I and Q branch, we just get the signal down to intermediate frequency and try to map this signal back to its corresponding bit pairs by using RNN.

So, we have only the sum of two branches, not the separated I and Q branches. Because of this the traditional ways of frequency offset detector are not working. For example, if we want to use the CORDIC algorithm to get the arccosine function which is used to estimate frequency, we will face the problem that the CORDIC algorithm's input must be the separated 2 branches.

So, we had to develop a new algorithm to estimate the frequency of the received signal. We have estimated the frequency of the signal by this simple equation:

Estimated frequency = (Arccosine(X1) - Arccosine(X2))/ (12*t_sampling*2*π)

- Arccosine is the inverse cosine function: we notice that the signal we received is very close to the cosine signal. We used lookup table to carry out this function by saving the inverse cosine values, but we found that we will need very big table or we have not very good result because of the values in between the values stored in the lookup table, so we used linear interpolation formula which is:

$$y = C1 * x + C2$$

Which: $C1 = (y2 - y1)/(x2 - x1)$

$$C2 = y1 - x1 * C1$$

So, we save the values of C1 and C2 in the lookup table instead.

- X1 and also X2 are the average values of 4 samples taken from the input signals.

The architecture is shown in Figure 3.2.4.1

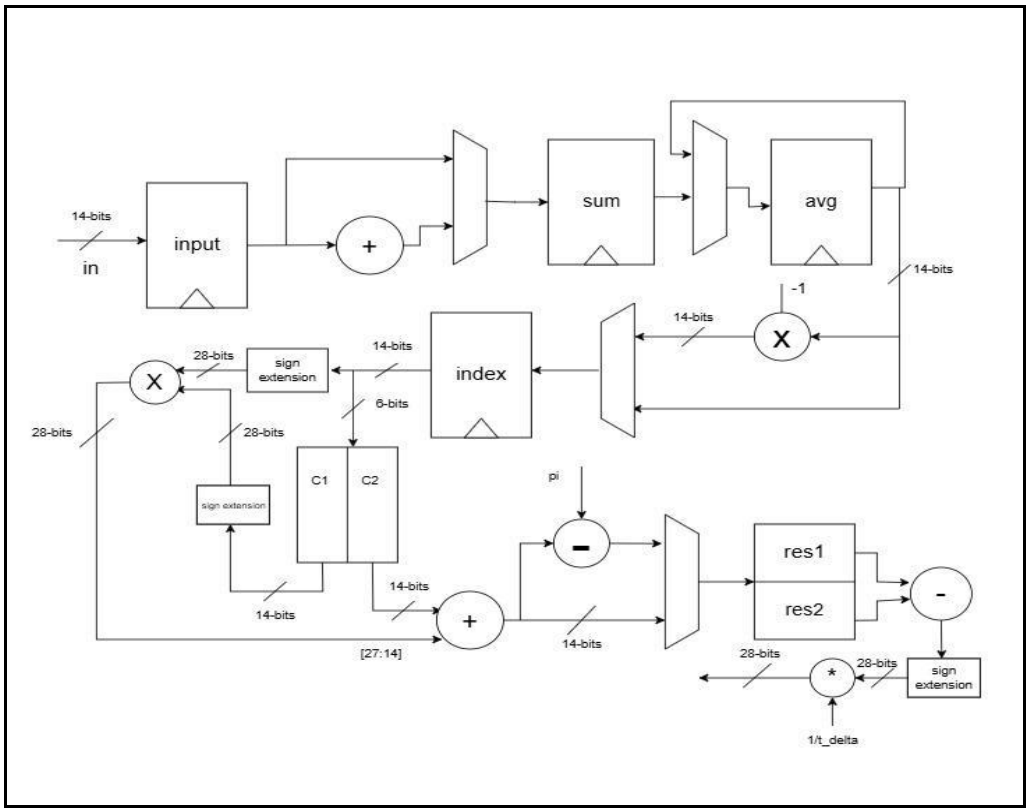


Figure 3.2.4.1: frequency estimation block architecture

3.2.5 Packet edge detection

The implementation of Packet edge detection consists of three modules: energy detection module, correlator and controller. The energy detection main idea is to sense the medium. Correlator's main idea is to compute cross correlation between input and saved preamble. And the controller to control them and turn on and off the modules when needed.

The I/O diagram of packet edge detection top module is shown in the Figure 3.2.5.1

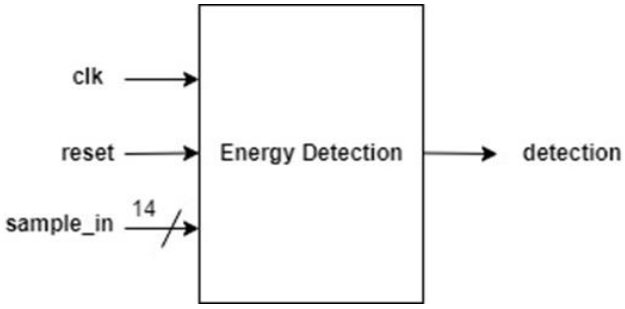


Figure 3.2.5.1: top module of packet edge detection

3.2.5.1 Energy detection

To implement energy detection, we first assume that each sample take 14 bits to achieve a perfect accuracy.

If we take only one window and do autocorrelation, then compare the result with the threshold we will face a problem in choosing the threshold. As shown in the Figure3.2.5.1.1 if we take only one window the output will be dependent of the distance between the transmitter and the receiver, so after choosing the threshold the transmitter may move away from the receiver and the signal of the transmitter will be considered a noise.

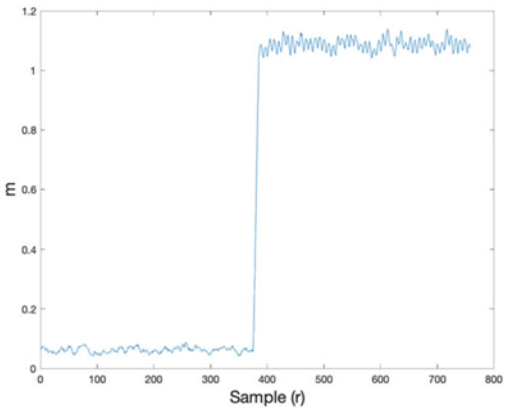


Figure 3.2.5.1.1: result of energy packet edge detection using single window

So, we need to normalize the value of this metric, we use another approach called double window packet edge detection as shown in Figure 3.2.5.1.2, we used two windows of the same length, each measuring autocorrelation the same way the single window did above then divide them on each other to get a normalized value of 1 for the matric.[16]

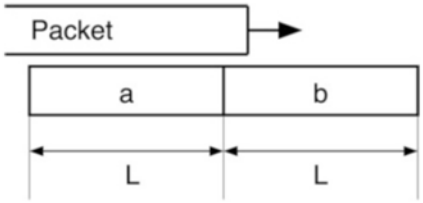


Figure 3.2.5.1.2: double window packet edge detection

The output of this approach shown in Figure 3.2.5.1.3, we notice that there is a spike when it detects a signal otherwise the normalized value remains 1 so we can put the threshold a value bigger than 1.

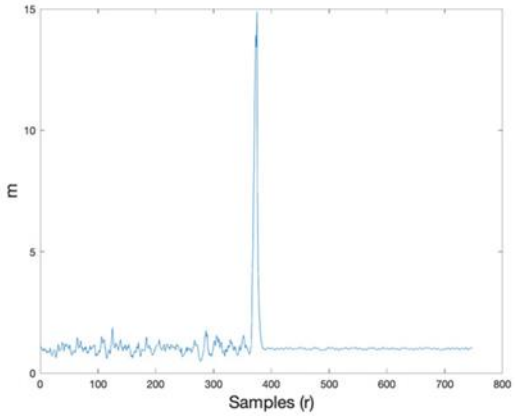


Figure 3.2.5.1.3: result of energy packet edge detection using double windows

The implementation of energy detection block is shown in Figure3.2.5.1.4

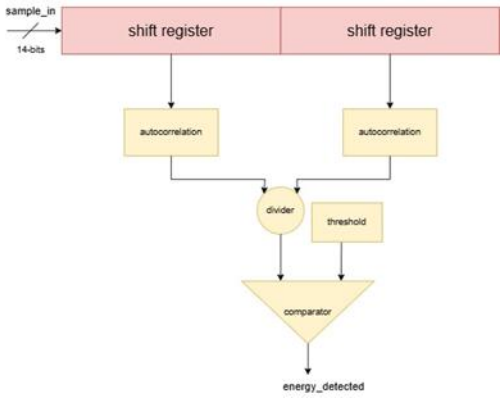


Figure 3.2.5.1.4: energy packet edge detection architecture

3.2.5.2 Cross correlation

To implement the cross correlation, first we store the preamble samples in memory. Then we Load the shift register with the first sample of the input signal. Then Multiply each bit of the shift register by the corresponding bit of the reference signal and Sum the products to get a correlation value. Then Shift the input signal through the register one bit at a time. Finally Repeat the cross-correlation calculation for each new position of the shift register. Until we finish all eight symbols of preamble. Then we compare the sum with the threshold. We choose the threshold to be 75% of the final correlation value taking in consideration the noise of the channel. The full value of cross-correlation is 62.08, So the threshold is 46.56. And we compare after each two symbols for the false alarm if there is a false alarm and this packet is not for us.

Figure 3.2.5.2.1 shows us the block diagram of this block.

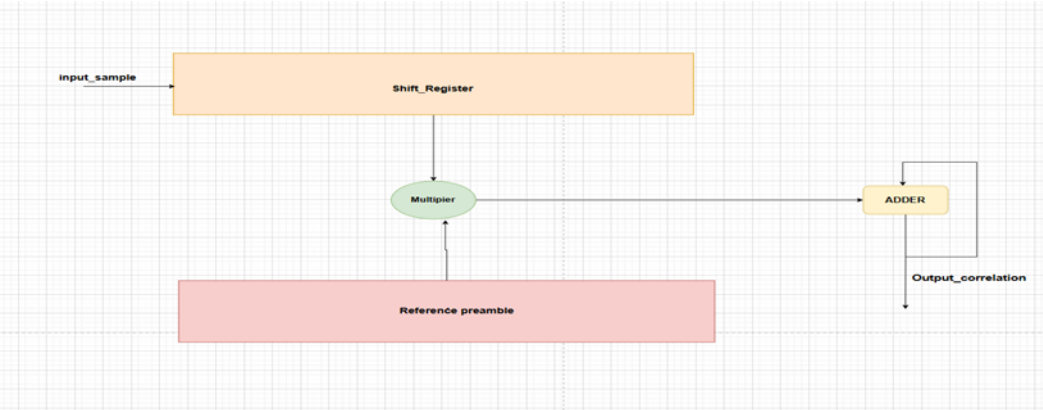


Figure 3.2.5.2.1. The block diagram of the correlator

.

3.2.5.2 Controller

The controller is a FSM to control the whole block and turn them on and off. The state machine consists of three states as shown in figure 3.2.5.2.1. First state is scanning state in this state we enable the energy detection module only to sense the medium once it senses there is a packet we turn it off and move with the input to correlator we turn it on and when it finishes with detect we move to RNN state or when it finishes with false alarm, we return to state scan. When we reach to state RNN we start the RNN and let it take the input and turn off the synchronizer waiting for RNN to end its work. Then we start sensing again and so on.

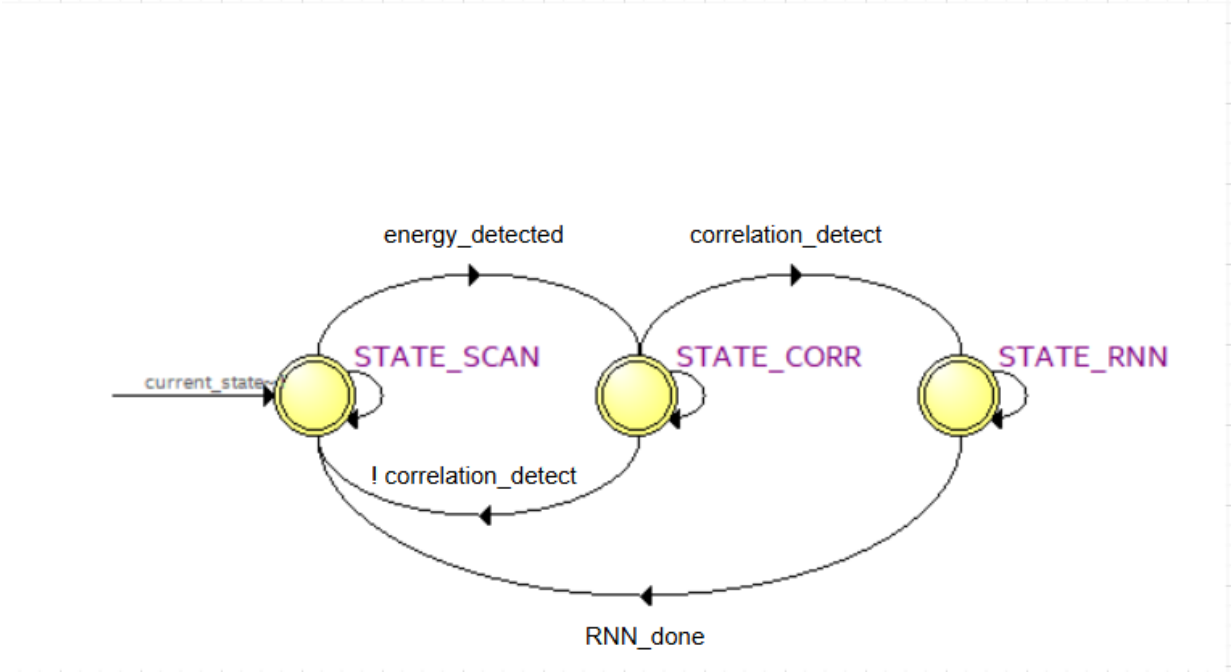


Figure 3.2.5.2.1. The state machine for the controller.

3.3: Advanced Encryption Standard (AES)

The I/O diagram of AES Top module is shown in Figure 3.3.1

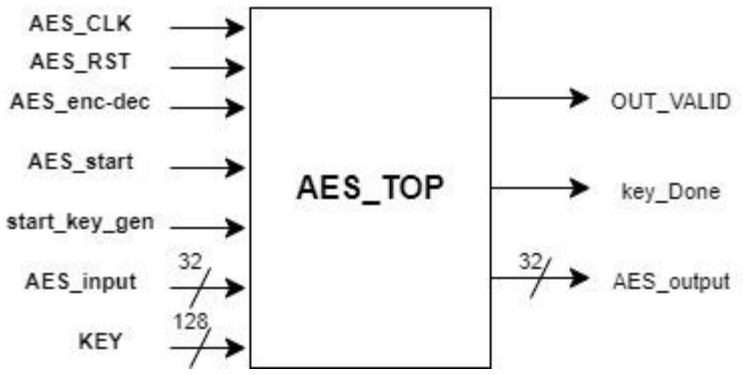


Figure 3.3.1: I/O Diagram

The AES hardware architecture presented here in Figure 3.3.2 is for AES-128 encryption and decryption, it has a small area compared with other AES architecture [15].

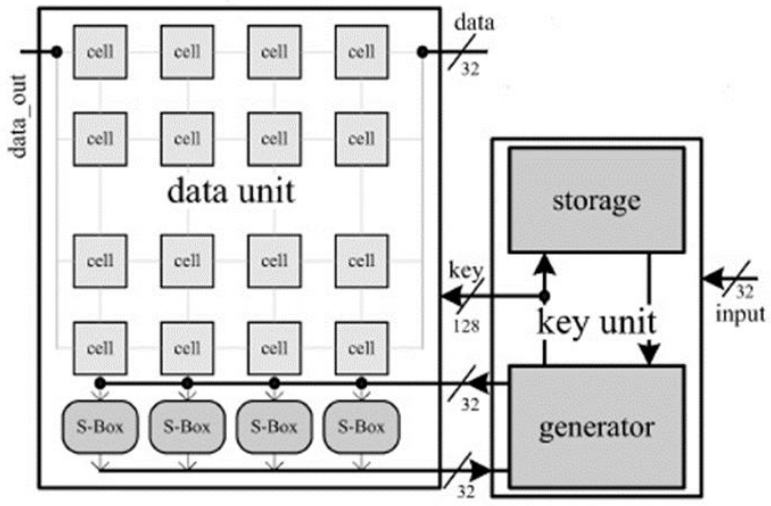


Figure 3.3.2: AES TOP Module

The AES architecture has 3 main components which are presented in the following sections

3.3.1: Controller

The controller is the most important module of the architecture as it acts like the brain of AES-128.Here's the I/O Diagram of the controller shown in Figure 3.3.1.1.

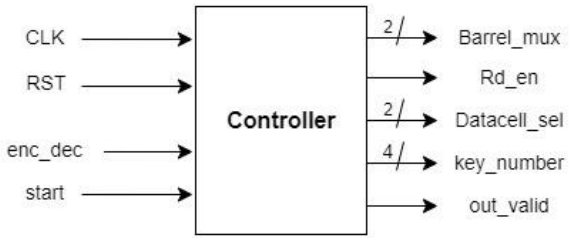
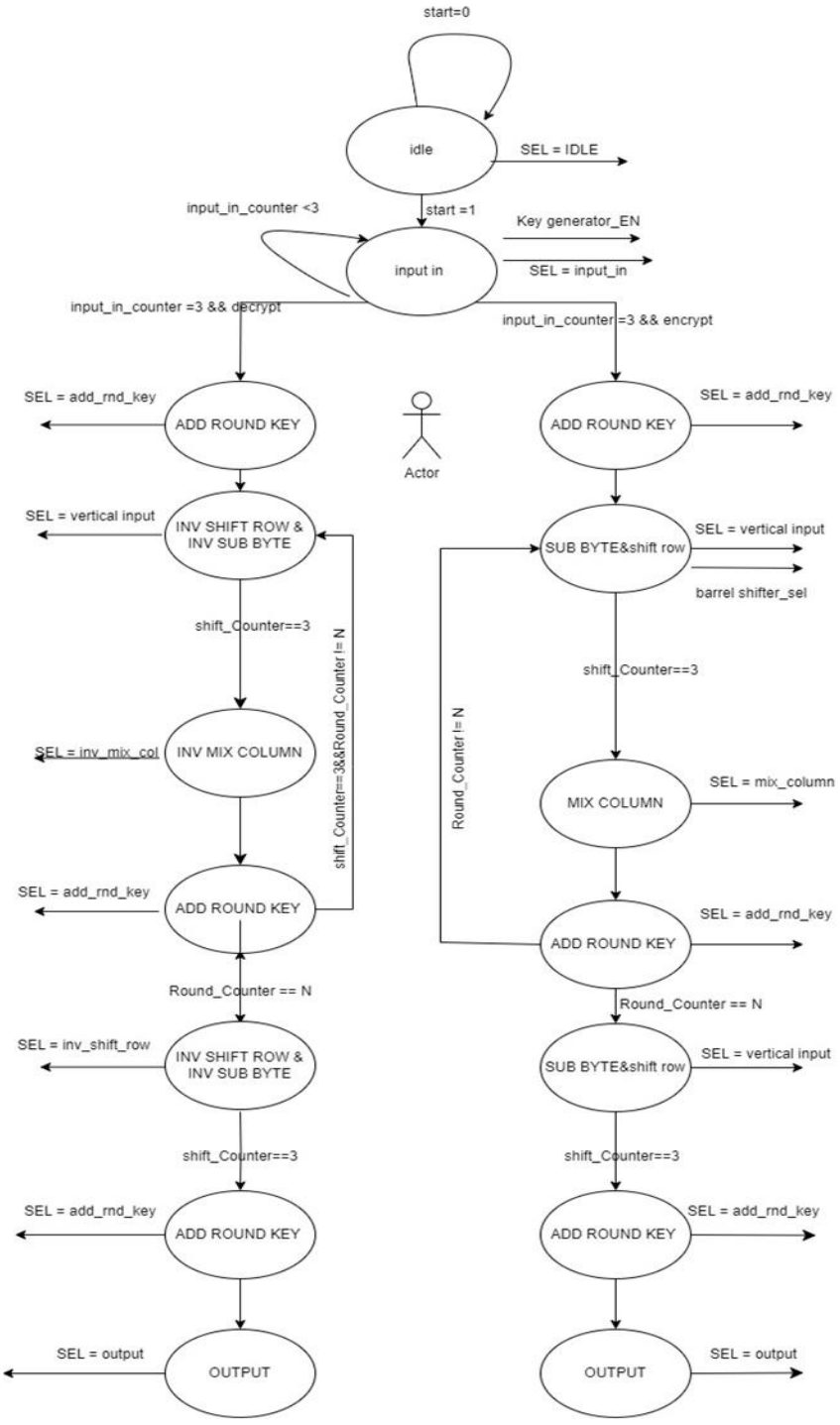


Figure 3.3.1.1: I/O Diagram

To understand well the role of the controller let's see the state diagram of the controller as shown in Figure 3.3.1.2.at beginning if the start port is turned to one, then the block is work and transition happens from IDLE state to Input state. The input state remains 4 clock cycles to start input the data from the interface to the data cells, after the input state is done after 4 clock cycles. The next transition is done with respect to the enc_dec port which responsible for encryption process (when enc_dec port = 1'b1) or decryption process (when enc_dec port = 1'b0). Let's consider the enc-dec port = 1'b1, which means the controller is going to the encryption process. So, the transition went to ADD_round_key state which make the selection of the data cell is turned to the addroundkey option and the operation is done in one clock cycle (by XORing each data cell with the corresponding cipher key).

The next transition is from the add_round _key state to sub byte & shift row state which make the selection of the data cell is turned to the vertical input and this enables the architecture to work to achieve the two functions sub bytes and shift rows in parrel in 4 clock cycles, as row 3 of data cells after the 1st clock cycle will be the input to the s-boxes block afterwards entered the barrel shifter block and shift left by 3 bytes as the barrel mux selection will be 2'b11. During the 2nd clock cycle, row 2 of data cells will be the input to the s-boxes block after wards entered the barrel shifter block and shift left by 2 bytes as the barrel mux selection will be 2'b10. During the 3rd clock cycle, row 1 of data cells will be the input to the s-boxes block after wards entered the barrel shifter block and shift left by 1 byte as the barrel mux selection will be 2'b01. During the 4th clock cycle, row 0 of data cells will be the input to the s-boxes block after wards entered the barrel shifter block and keep it with no change as the barrel mux

selection will be 2’b00.



Notice that after each clock cycle each row of data cells is transited down one row by the vertical input selection which is activated through these 4 clock cycles.

The next transition is from sub byte & shift row state to mix column state which make the selection of data cell is turned to the mixcolumn_in and this enables the mix column function in one clock cycle.

The next transition is from mix column state to add_round_key state which make the selection of the data cell is turned to the add round key option to perform the function of add round key (By XORing each data cell with the corresponding round one key) and that happen in one clock cycle, over all here we can say that the first round of encryption is done and it is repeated 9 rounds with 6 clock cycles per each round.

Then the last round happens in 5 clock cycles only as it consists of sub byte & shift row state (4 clock cycles) and add round key state (1 clock cycle).

Now, Let's consider the enc-dec port = 1'b0, which means the controller is going to the decryption process.

So, the transition went to ADD_round_key state which make the selection of the data cell is turned to the add round key option and the operation is done in one clock cycle (by XORing each data cell with the corresponding round 10 key).

The next transition is from the add_round _key state to inv-sub byte & inv-shift row state which make the selection of the data cell is turned to the vertical input and this enables the architecture to work to achieve the two functions inv-sub bytes and inv-shift rows in parrel in 4 clock cycles, as row 3 of data cells after the 1st clock cycle will be the input to the s-boxes block afterwards entered the barrel shifter block and shift right by 3 bytes as the barrel mux selection will be 2'b11. During the 2nd clock cycle, row 2 of data cells will be the input to the s-boxes block after wards entered the barrel shifter block and shift right by 2 bytes as the barrel mux selection will be 2'b10. During the 3rd clock cycle, row 1 of data cells will be the input to the s-boxes block after wards entered the barrel shifter block and shift right by 1 byte as the barrel mux selection will be 2'b01. During the 4th clock cycle, row 0 of data cells will be the input to the s-boxes block after wards entered the barrel shifter block and keep it with no change as the barrel mux selection will be 2'b00. Notice that after each clock cycle each row of data cells is transited down one row by the vertical input selection which is activated through these 4 clock cycles.

The next transition is from inv-sub byte & inv-shift row state to inv-mix column state which make the selection of data cell is turned to the mixcolumn_in and this enables the inv-mixcolumn function in one clock cycle.

The next transition is from inv-mix column state to add_round_key state which make the selection of the data cell is turned to the addround key option to perform the function of add round key (By XORing each data cell with the corresponding round 9 key) and that happen in one clock cycle, over all here we can say that the first round of decryption is done and it is repeated 9 rounds with 6 clock cycles per each round.

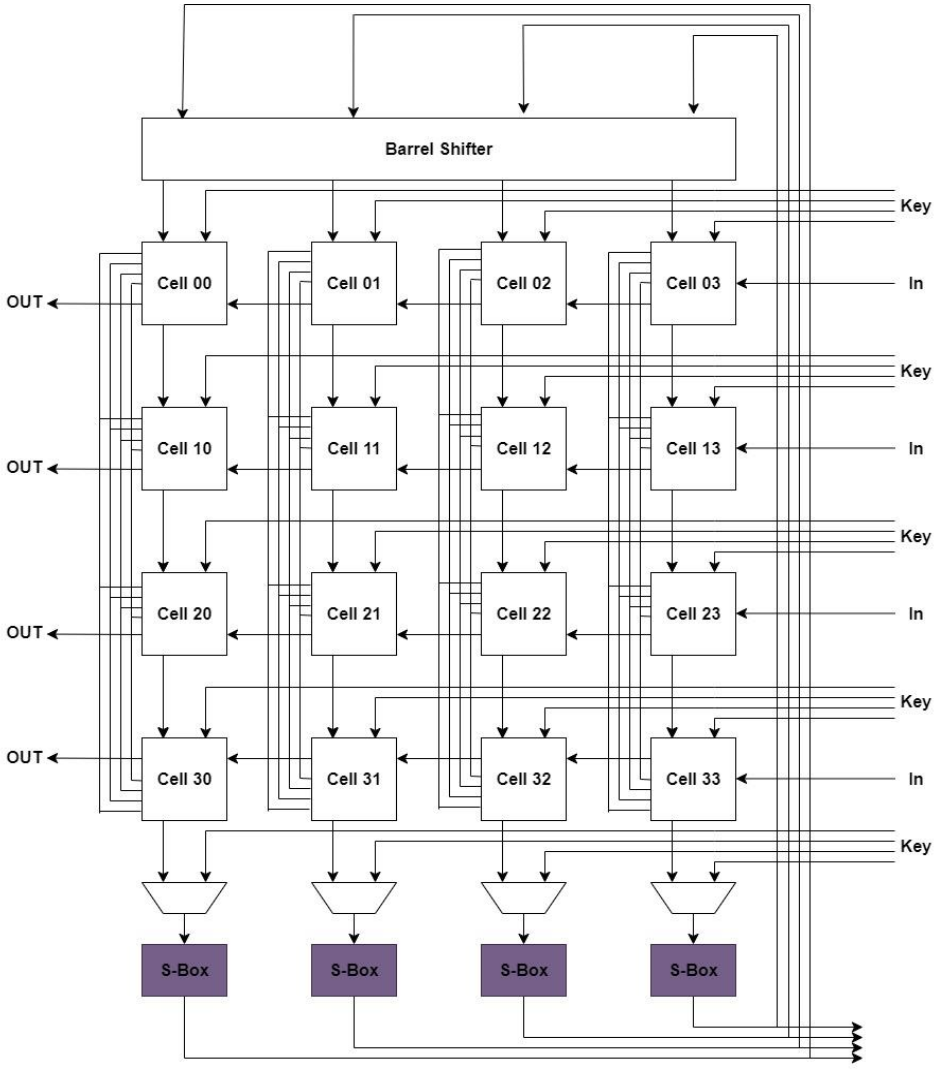
Then the last round happens in 5 clock cycles only as it consists of inv-sub byte & inv-shift row state (4 clock cycles) and add round key state (1 clock cycle).

3.3.2: Data Unit

The data unit is the main module of the architecture. It can perform any kind of AES encryption or decryption round using the round key that is assigned to its key input. The data unit is the biggest and the most important component of the AES architecture. It stores the current 128-bit state of an encryption or decryption and can perform any number and type of encryption/ decryption rounds on this state. Consequently, all four AES transformations (Sub Bytes, Shift Rows, Mix Columns, and Add Round Key) and the corresponding inverse transformations are implemented within the data unit. For the Add Round Key transformation, a round key needs to be provided by the key unit.

The data unit consists of 3 blocks as shown in Figure 3.3.2.1, the cell, S-Box and barrel shifter. To load a data block, the input data is shifted column by column from the right side. To compute a normal AES round, the registers are rotated vertically to perform the Inv-/Sub Bytes and the Inv-/Shift Rows transformation row by row. In the first clock cycle, the Inv-/Sub Bytes transformation starts for row three. Because the implementation of the S-Boxes is pipelined, the result of this Inv-/Sub Bytes transformation is stored in row zero two clock cycles later. Using the pipelined S-Boxes and the Barrel shifter between row zero and output of S-Boxes, the Inv-/Sub Bytes and the Inv-/Shift Rows transformations can be applied to all 16 by bytes of the state within four clock cycles.

Figure 3.3.2.1: Architecture of Data Unit



In the fifth and sixth clock cycle of a normal AES round, the Inv-/Mix Columns and the Add Round Key transformations are performed by all data cells in parallel. Since the S-Boxes are not used by the data unit during these clock cycles, they can be utilized by the key unit to perform the key expansion for the next round key.

Let's take each part of the data unit and open it to know how it's implemented.

3.3.2.1: Data Cell

The design of the data cells is crucial for the overall architecture of the data unit. The data cells serve as storage elements of the AES state and perform the Inv-/Mix Columns and the Add Round Key transformation as shown in Figure 3.3.2.1.1.

The I/O diagram of the data cell shown in Figure 3.3.3.1.2

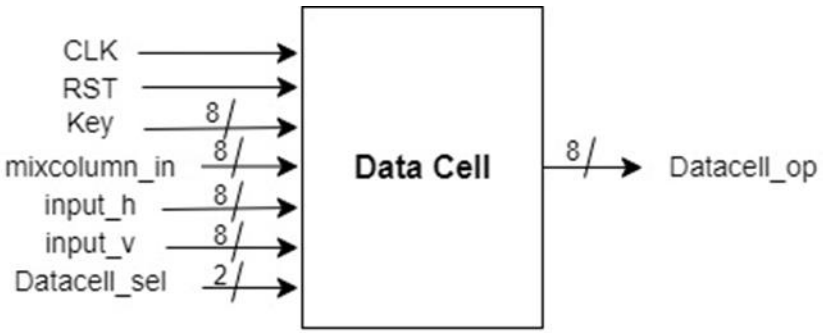


Figure 3.3.2.1.2: I/O Diagram of Data cell

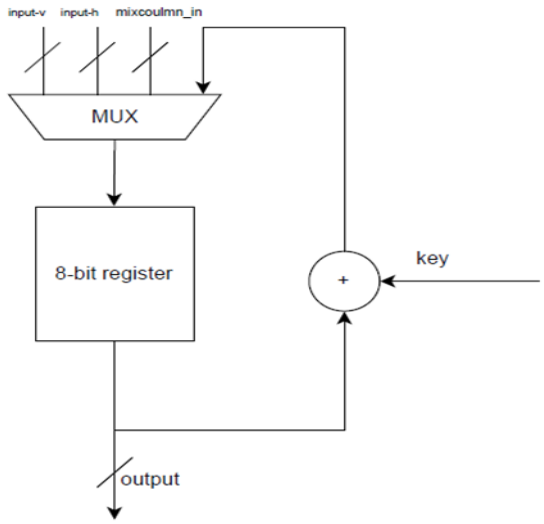


Figure 3.3.2.1.1: Architecture of Data cell

3.3.3.2: Barrel Shifter

The Barrel shifter is the block used to perform the Shift rows and Inv-Shift rows transformations in both encryption or decryption. The I/O Diagram of Barrel shifter is shown in Figure 3.3.3.2.1.

The implementation of the barrel is done by 2 multiplexers, the first one whose selection line is labeled sel is used to identify we work in shift rows (sel = 1) or inv shift rows (sel = 0) and the other multiplexer whose selection lines are labeled (shiftAmt) is used to identify how many numbers of bytes are shifted. notice that all shifting in the RTL by the concatenation property.

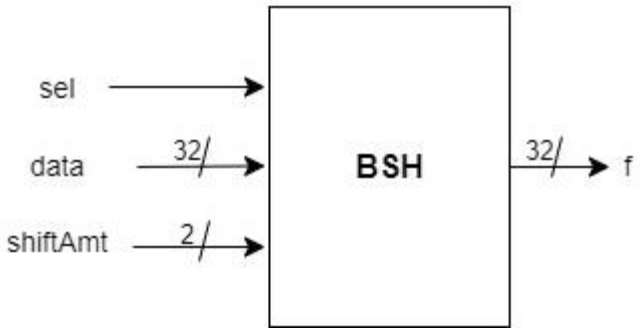


Figure 3.3.3.2.1: I/O Diagram of Barrel Shifter

3.3.3.3: Inv-Mix Columns

There're 4 blocks in the Data unit block. Each block works over a column of data cells which performs the Mix column and inverse Mix column functions as shown in Fig. 3.3.3.3.1.

let's remember the function of the Mix columns (used in Encryption). The function of Mix columns is done by multiplication of Galio Field (2^8) of the current state by fixed matrix under these equations:

$$X3 = \{2.x3\} \oplus \{3.x2\} \oplus \{x1\} \oplus \{x0\}$$

$$X2 = \{x3\} \oplus \{2.x2\} \oplus \{3.x1\} \oplus \{x0\}$$

$$X1 = \{x3\} \oplus \{x2\} \oplus \{2.x1\} \oplus \{3.x0\}$$

$$X0 = \{3.x3\} \oplus \{x2\} \oplus \{x1\} \oplus \{2.x0\}$$

Where X3 is the most significant byte which stored in cell 0,j in each column of the four columns of data cells.

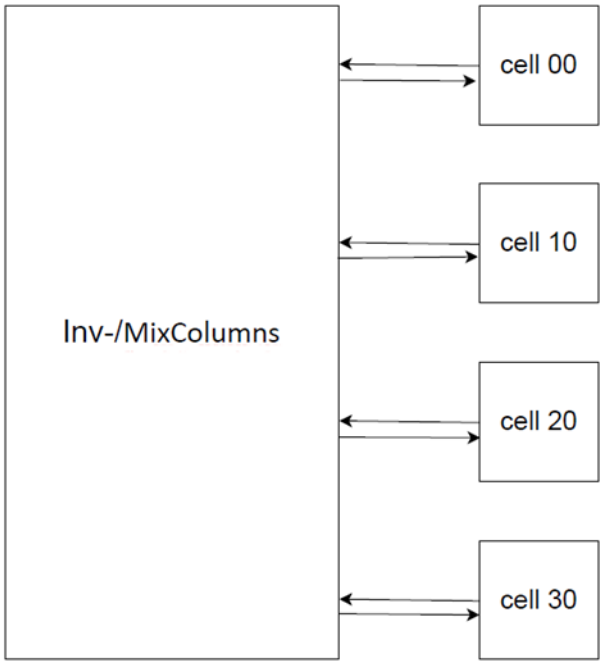


Figure 3.3.3.3.1

However, because they're multiplying by the fixed constants 1, 2 and 3, it's easier to implement than a general GF(2^8) multiplication.

- Multiplying by 1 is easy; it's exactly what you'd expect

- Multiplying by 2 is equivalent to shifting the number left by one, and then XORing the value 0x1B if the high bit had been one (the value 0x1B came from the field representation) and if the high bit is zero, we don't need to XOR anything (or XOR in a 0x00 constant) as shown in Figure 3.3.3.3.2.
- Multiplying by 3 is equivalent to multiplying by 2 (see above), and then XORing that with the original value, since 3=2⊕1.

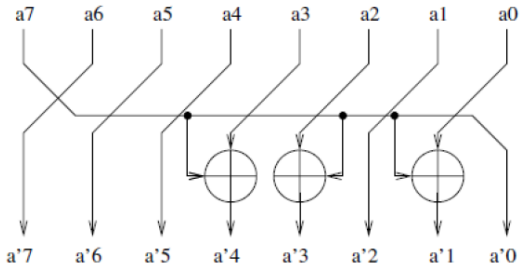


Figure 3.3.3.3.2: Multiplication by 2

Now, let's remember the inverse of this operation (used in decryption). The function of inv-Mix columns is done by multiplication of Galio Field (2^8) of the current state by fixed matrix under these equations:

$$x3 = \{0E.X3\} \oplus \{0B.X2\} \oplus \{0D.X1\} \oplus \{09.X0\}$$

$$x2 = \{09.X3\} \oplus \{0E.X2\} \oplus \{0B.X1\} \oplus \{0D.X0\}$$

$$x1 = \{0D.X3\} \oplus \{09.X2\} \oplus \{0E.X1\} \oplus \{0B.X0\}$$

$$x0 = \{0B.X3\} \oplus \{0D.X2\} \oplus \{09.X1\} \oplus \{0E.X0\}$$

Where x3 is the most significant byte which stored in cell 0, j in each column of the four columns of data cells.

Since multiplication is distributive over addition in GF(2^8), we can rewrite the multiplication of two elements of GF(2^8) as a linear combination of products of the first element and a single-termed polynomial in GF(2^8).

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} X3 \\ X2 \\ X1 \\ X0 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} X3 \\ X2 \\ X1 \\ X0 \end{bmatrix} + \begin{bmatrix} 0C & 08 & 0C & 08 \\ 08 & 0C & 08 & 0C \\ 0C & 08 & 0C & 08 \\ 08 & 0C & 08 & 0C \end{bmatrix} \begin{bmatrix} X3 \\ X2 \\ X1 \\ X0 \end{bmatrix}$$

And we can implement

- 0C*X = (2*2*(2+1)) X
- 08*X = (2*2*2) X

So we can Implement the Inv-/Mix Columns block as shown in Figure 3.3.3.3.3

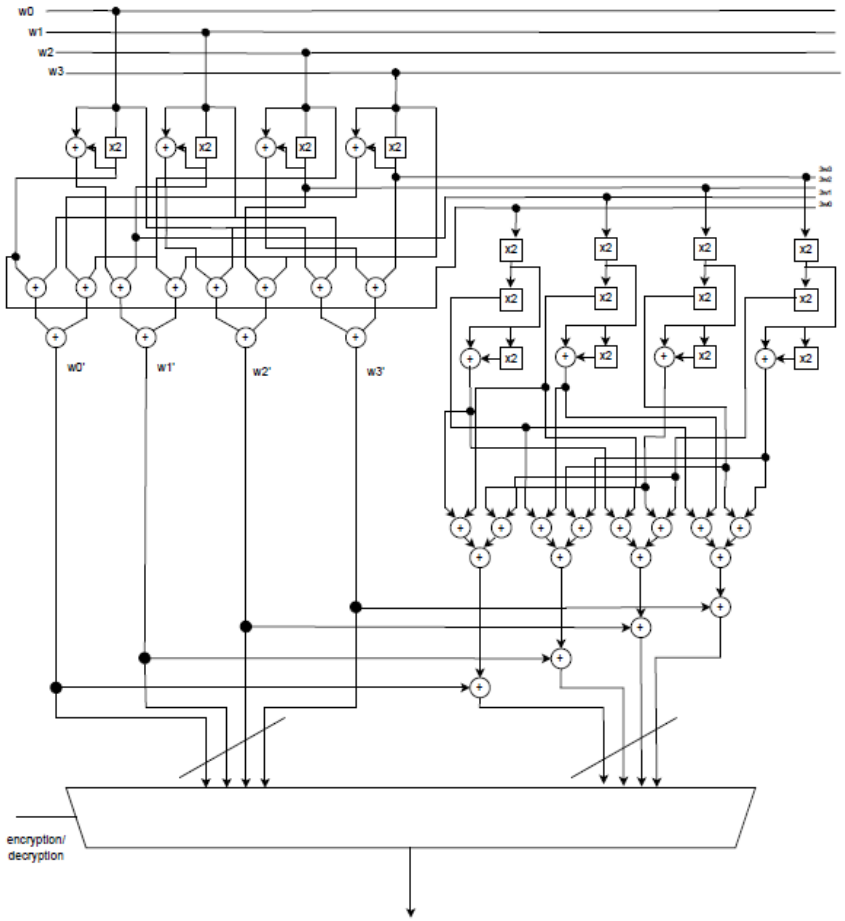


Figure 3.3.3.3.3: Architecture of Inv-/Mix column Block

3.3.3.4: S-Box

The S-box block is the block used to perform Inv-/Sub bytes Function. The block is simply consisting of a look up table as we know before in S-box & inv S-box tables shown in Figures 2.3.1.4 & 2.3.3.1.1 respectively, that the Inv-/Sub bytes Functions are linear substitution of bytes

Here in Figure 3.3.3.4.1, we can see the I/O diagram of S-Box

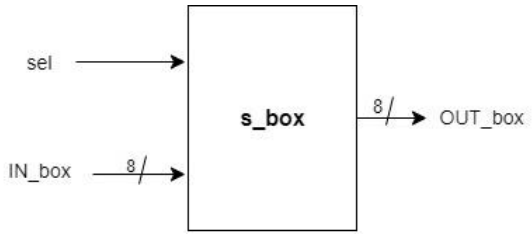


Figure 3.3.3.4.1: I/O Diagram of S-Box

3.3.2: Key Unit

The key unit is used to store keys and to calculate the key expansion function, so it consists of two main parts: storage and key generator as shown in Figure 3.3.1 and S-box block used in key generator.

The key generator takes a 128-bits key as an input and gives all the expanded keys that are used in all rounds after 90 clock cycles. These keys are stored in the storage block then it can be used in both encryption and decryption, if we want to use these keys in encryption, we read the keys from the memory upside down and we do the opposite in the decryption.

Storage

It is just registers used in storing the calculated keys from the key generator block for the Data unit to use.

Key generator

It is the major part of the key unit, its architecture is shown in Figure 3.3.2.1

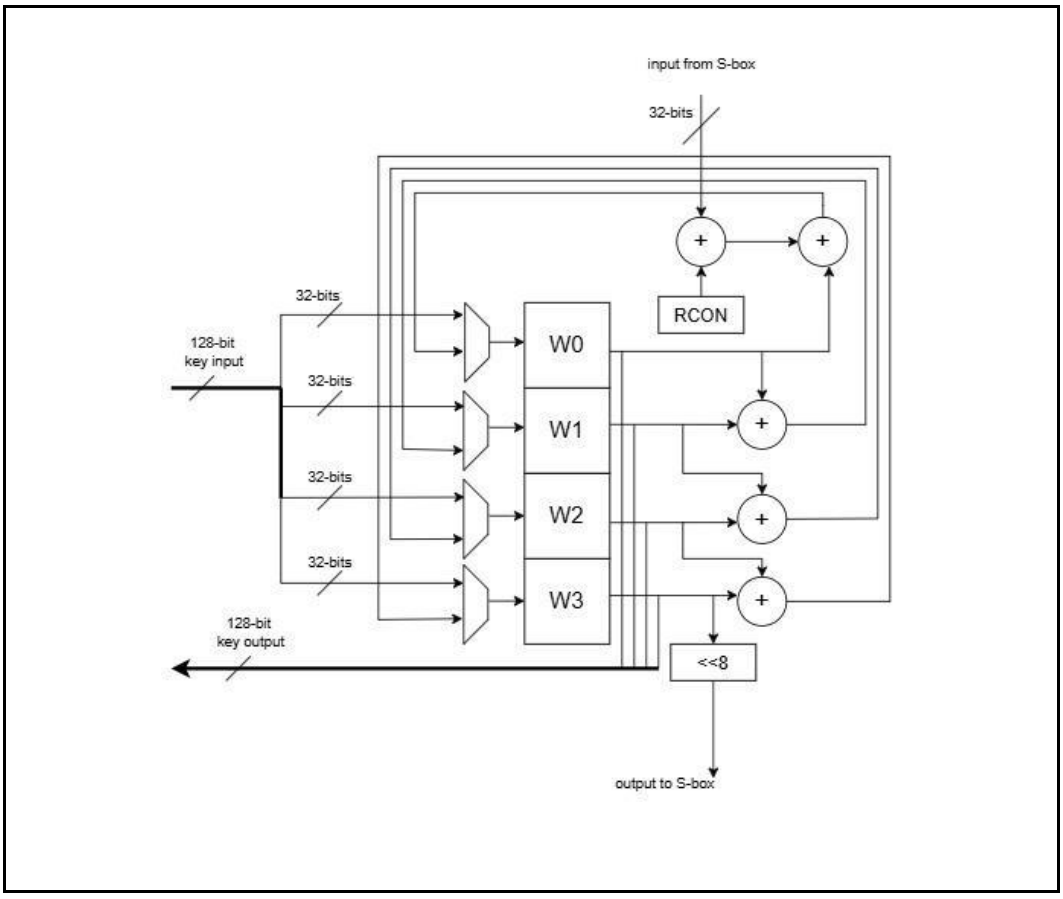
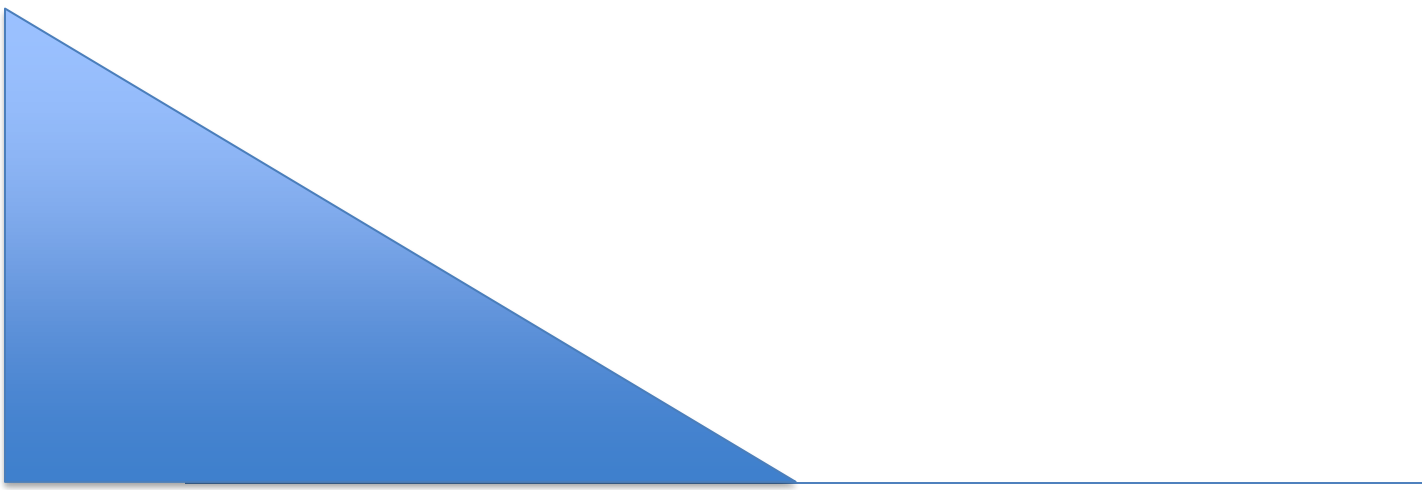


Figure 3.3.2.1 : key generator architecture

.

Chapter 4 : Results and simulations

- **Transmitter**
- **Receiver**
- **AES**



4.1.1 Bit to symbol

As shown in figure 5.1.1.1 the result of simulation of bit to symbol block. Each four bits input transformed to a four bits symbol. When input is 1 0 1 0 in serial the output is 1010 symbols.

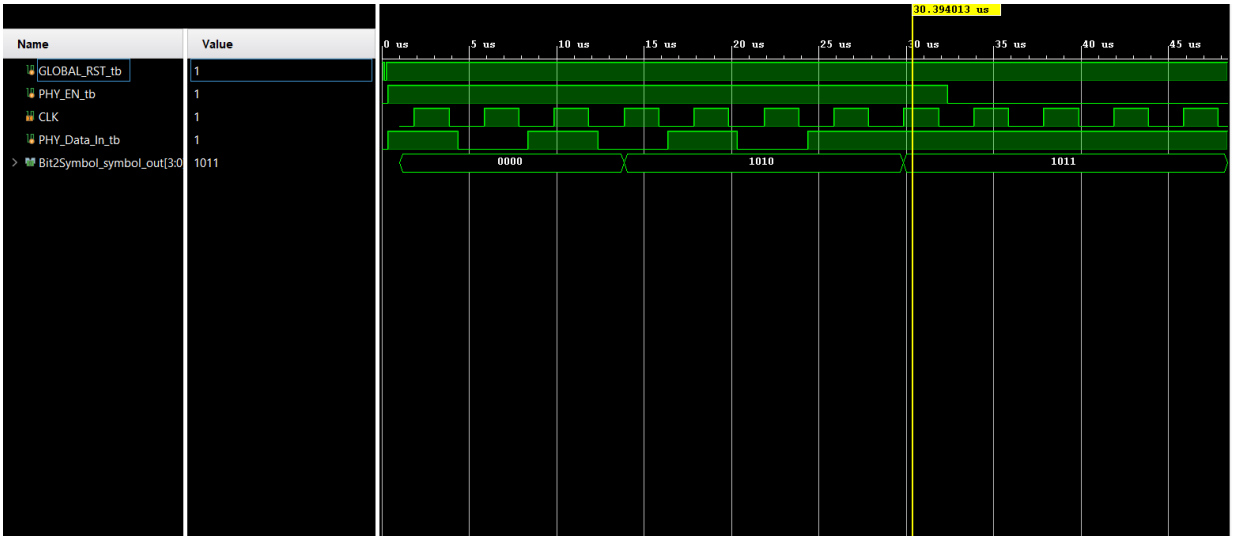


Figure 4.1.1.1 simulation of bit to symbol block.

4.1.2 Symbol to chip

As shown in figure 5.1.2.1 the simulation of symbol to chip block each symbol input is transformed to 32 bits chip sequence.

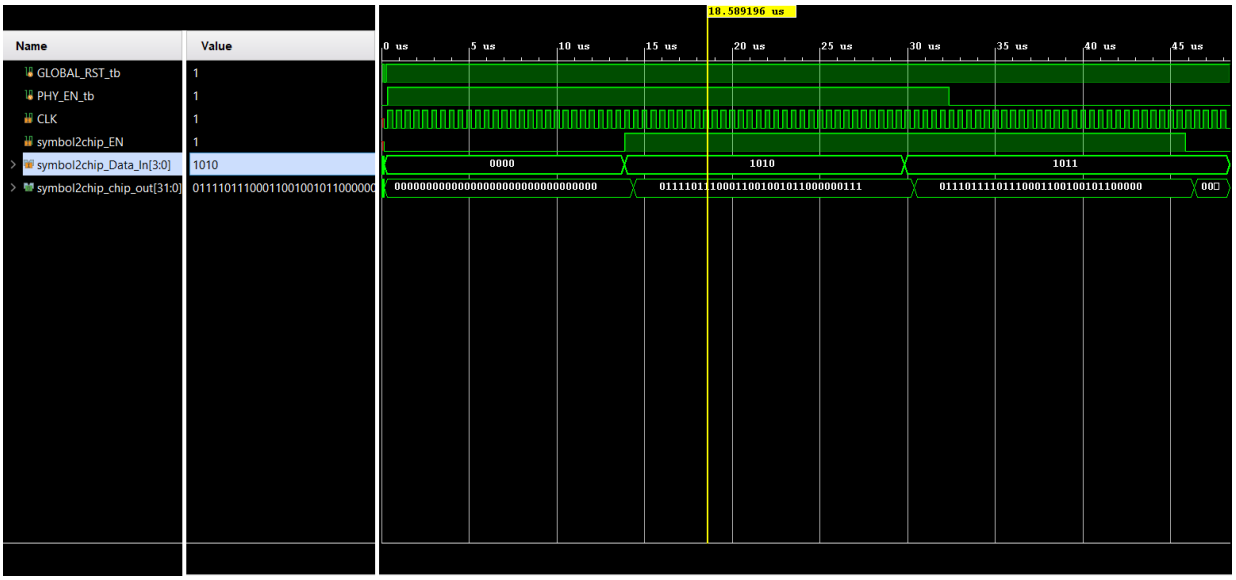


Figure 5.1.2.1. simulation of symbol to chip block

4.1.3 Modulator

In this block as discussed earlier in section 3.1.3. It generates the even bits stream and odd bit stream separated by Tc. The figure 5.1.3.1 shows the result of this block. The even bit stream is delayed by Tc.

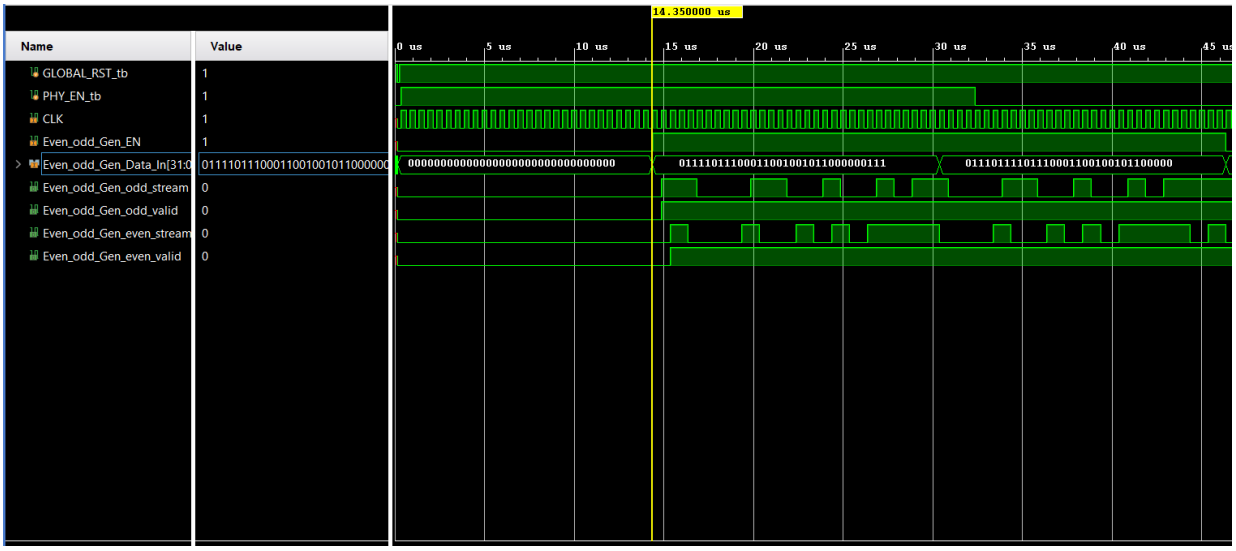


Figure .4.1.3.1. Simulation of modulator.

4.1.4 Pulse shaping

In this block the output is a fixed-point binary vector representing the samples of each symbol. The figure 5.1.4.1 shows the output of this block as binary output each bit is transformed to 10 samples. The figure 5.1.4.2 shows us the same output but when changing the waveform style to analog wave to make sure the symbol shape is half sine.

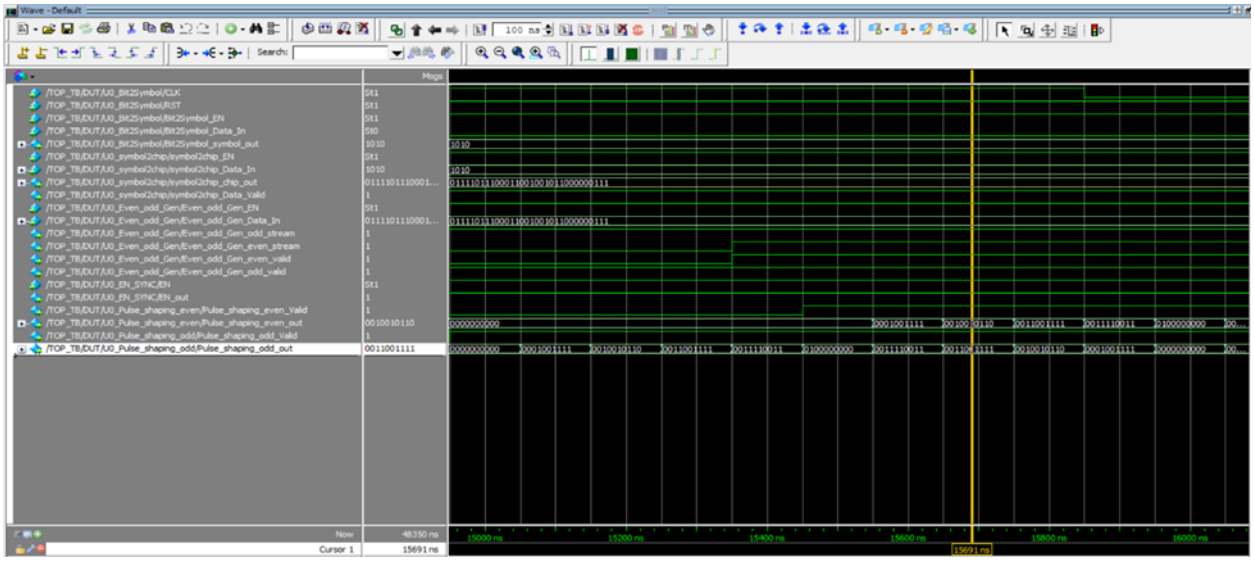


Figure4.1.4.1 simulation of pulse shaping.

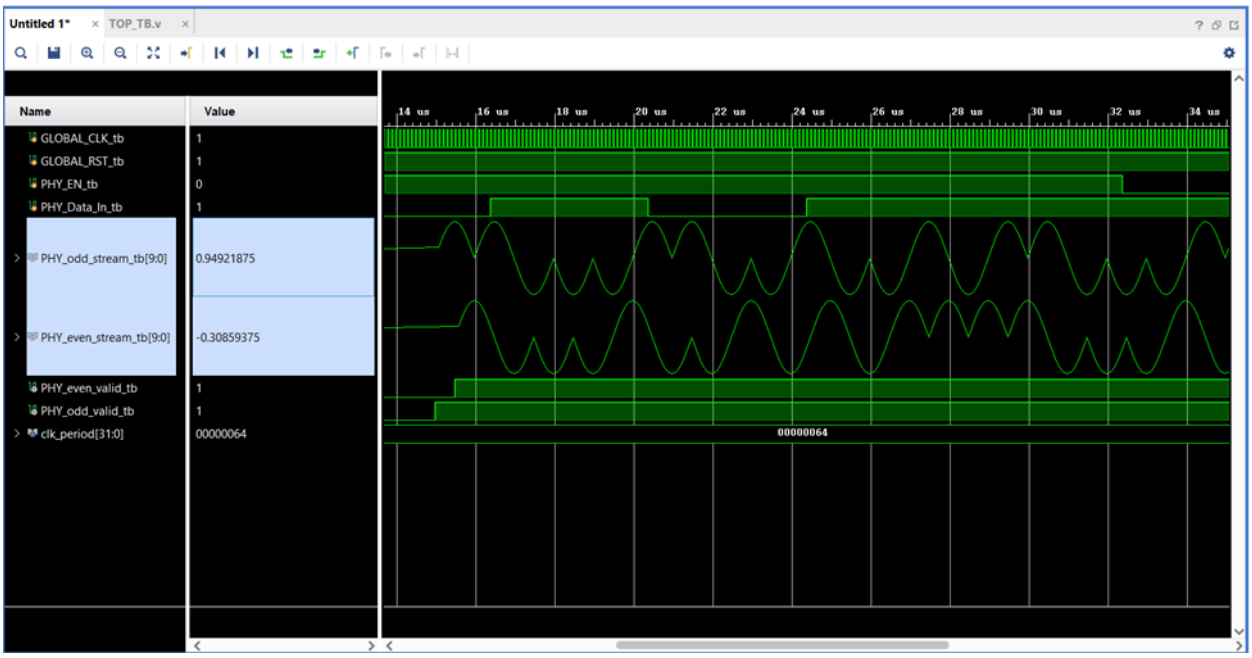


Figure 4.1.4.2. output of pulse shaping in analog waveform.

4.1.5 Synthesis of transmitter

We do synthesis of the transmitter chain from bit to symbol block to end using ICC2 compiler. The schematic output of synthesis is shown in figure 4.1.5.1.

And area and power reports are shown in figure 4.1.5.2 and 4.1.5.3.

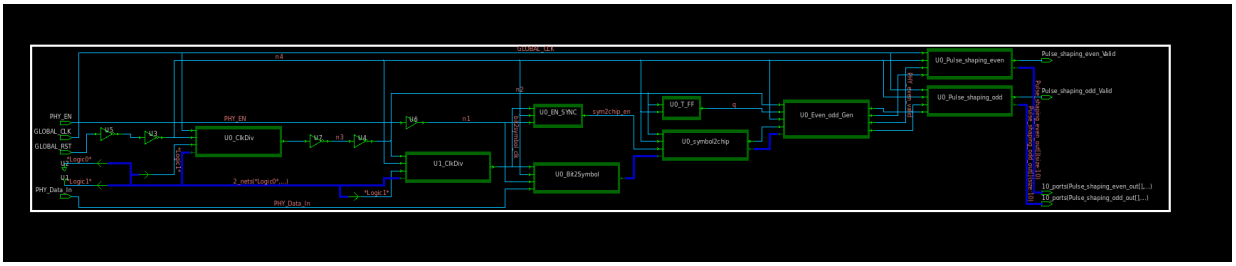


Figure 4.1.5.1 netlist of TX.

```
10
11    scmetro_tsmc_cl013g_rvt_ss_1p08v_125c (File: /home/IC/Assignments/system/std_cells/scmetro_tsmc_cl013g_rvt_ss_1p08v_125c.db)
12
13 Number of ports:                          181
14 Number of nets:                           694
15 Number of cells:                          527
16 Number of combinational cells:            413
17 Number of sequential cells:               104
18 Number of macros/black boxes:               0
19 Number of buf/inv:                        113
20 Number of references:                      13
21
22 Combinational area:               3064.126850
23 Buf/Inv area:                      571.876213
24 Noncombinational area:            2722.883791
25 Macro/Black Box area:                0.000000
26 Net Interconnect area:          182700.922119
27
28 Total cell area:                  5787.010641
29 Total area:                     188487.932760
30
31 Hierarchical area distribution
32 ------------------------------
33
34                                  Global cell area          Local cell area
35                                  ------------------  ----------------------------
36 Hierarchical cell                Absolute   Percent  Combi-     Noncombi-  Black-
37                                  Total      Total    national   national   boxes   Design
38 --------------------------------  ---------  -------  ---------  ---------  ------  ------------------------
39 TX_TOP                           5787.0106    100.0    23.5340     0.0000  0.0000  TX_TOP
40 U0_Bit2Symbol                     349.4799      6.0   114.1399   235.3400  0.0000  Bit2Symbol
41 U0_ClkDiv                         403.6081      7.0   258.8740   144.7341  0.0000  ClkDiv_0
42 U0_EN_SYNC                        108.2564      1.9     4.7068   103.5496  0.0000  EN_sync
43 U0_Even_odd_Gen                   947.2435     16.4   483.6237   371.8372  0.0000  Even_odd_Gen
44 U0_Even_odd_Gen/add_40             91.7826      1.6    91.7826     0.0000  0.0000  Even_odd_Gen_DW01_inc_0
45 U0_Pulse_shaping_even             811.9230     14.0   383.6042   428.3188  0.0000  Pulse_shaping_even
46 U0_Pulse_shaping_odd             1087.2708     18.8   662.4821   424.7887  0.0000  Pulse_shaping_odd
47 U0_T_FF                            30.5942      0.5     0.0000    30.5942  0.0000  T_FF
48 U0_symbol2chip                   1610.9023     27.8   756.6181   854.2842  0.0000  symbol2chip
49 U1_ClkDiv                         414.1984      7.2   284.7614   129.4370  0.0000  ClkDiv_1
50
```

Figure 4.1.5.2 Area report of TX.

```
Library(s) Used:

    scmetro_tsmc_cl013g_rvt_ss_1p08v_125c (File: /home/IC/Assignments/system/std_cells/scmetro_tsmc_cl013g_rvt_ss_1p08v_125c.db)


Operating Conditions: scmetro_tsmc_cl013g_rvt_ss_1p08v_125c   Library: scmetro_tsmc_cl013g_rvt_ss_1p08v_125c
Wire Load Model Mode: top

Design        Wire Load Model            Library
------------------------------------------------
TX_TOP                tsmc13_wl30       scmetro_tsmc_cl013g_rvt_ss_1p08v_125c


Global Operating Voltage = 1.08
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1pW


--------------------------------------------------------------------------------
                                  Switch   Int      Leak     Total
Hierarchy                         Power    Power    Power    Power    %
--------------------------------------------------------------------------------
TX_TOP                            8.72e-03 8.25e-03 3.68e+06 2.06e-02 100.0
  U0_Pulse_shaping_odd (Pulse_shaping_odd)
                                  2.96e-04 2.47e-03 7.12e+05 3.47e-03  16.8
  U0_Pulse_shaping_even (Pulse_shaping_even)
                                  2.86e-04 2.46e-03 5.08e+05 3.25e-03  15.8
  U0_Even_odd_Gen (Even_odd_Gen)  7.13e-04 2.60e-04 6.44e+05 1.62e-03   7.8
    add_40 (Even_odd_Gen_DW01_inc_0)     0.000    0.000 8.37e+04 8.37e-05   0.4
  U0_T_FF (T_FF)                  1.24e-04 8.27e-05 2.25e+04 2.29e-04   1.1
  U0_symbol2chip (symbol2chip)    2.23e-03 8.25e-04 9.87e+05 4.04e-03  19.6
  U0_EN_SYNC (EN_sync)            8.26e-05 2.78e-05 4.48e+04 1.55e-04   0.8
  U0_Bit2Symbol (Bit2Symbol)      3.88e-04 7.24e-05 1.70e+05 6.30e-04   3.1
  U1_ClkDiv (ClkDiv_1)            1.05e-03 2.85e-04 2.78e+05 1.61e-03   7.8
  U0_ClkDiv (ClkDiv_0)            2.94e-03 1.73e-03 2.85e+05 4.95e-03  24.0
1
```

Figure 4.1.5.3 power report of TX.

4.1.6 STA of transmitter

We do static timing analysis to make sure there are no violations in setup and hold time.

The output of STA is there are no violations in setup or hold as shown in figure 4.1.6.1.

```
****************************************
Report : constraint
         -all_violators
Design : TX_TOP
Version: K-2015.06
Date   : Sat Jul  6 17:06:08 2024
****************************************

This design has no violated constraints.
1
```

Figure 4.1.6.1 STA of TX.

4.1.7 Formality of transmitter

formality or formal verification is used to check the matching between the output of the synthesis tool and the original reference RTL code. The formality steps were done through script to the Synopsys formality tool and here are the results showing that the formal verification has passed as shown in figure 4.1.7.1 and 4.1.7.2.

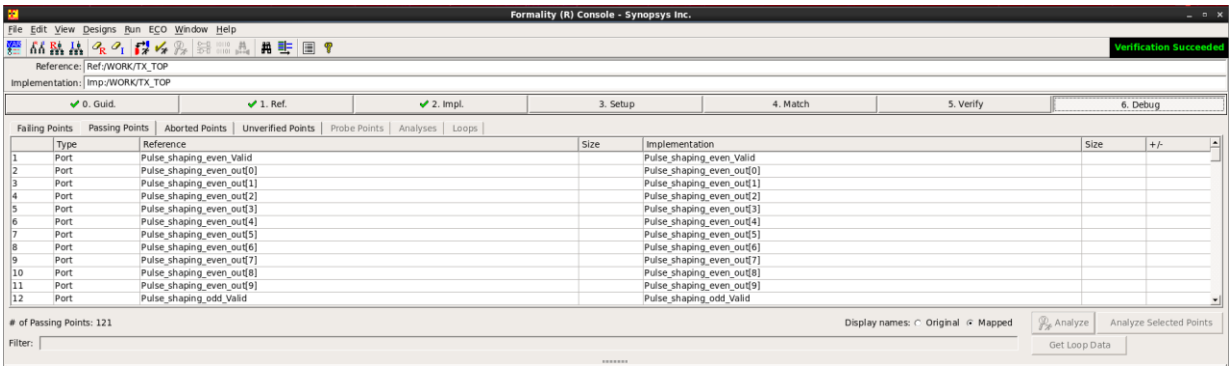


Figure 4.1.7.1 formality of TX.

```
1  ************************************************
2  Report         : failing_points
3
4  Reference      : Ref:/WORK/TX_TOP
5  Implementation : Imp:/WORK/TX_TOP
6  Version        : L-2016.03-SP1
7  Date           : Sat Jul  6 17:20:14 2024
8  ************************************************
9
10 No failing compare points.
11
12 1
```

Figure 4.1.6.1 formality report of TX.

4.2 Receiver

4.2.1 Decoder

In this section we will show comparisons between the implemented kogge stone adder and the adder exists in the 130n meter library “scmetro_tsmc_cl013g”, in addition to comparisons between the implemented wallace tree multiplier and the multiplier exists in the 130n meter library “scmetro_tsmc_cl013g”.

Kogge Stone Adder VS TSMC Adder

This comparison includes the major design fields which are the power, area, and delay. Remember that the most important purpose of designing the kogge stone adder is to optimize the performance of the system in terms of the speed or the delay, even if there is a tradeoff with the power and the area.

```
D0_26/Go (Dot_2)                          0.00       1.85 f
U73/Y (AOI21X1M)                          0.13       1.98 r
U72/Y (XNOR2X1M)                          0.17       2.15 r
S[27] (out)                               0.00       2.15 r
data arrival time                                    2.15
-------------------------------------------------------------
```

(a)

```
add_1_root_add_55_2/U1_26/CO (ADDFX2M)                  0.33       9.00 f
add_1_root_add_55_2/U1_27/Y (XOR3XLM)                   0.21       9.21 r
add_1_root_add_55_2/SUM[27] (add_DW01_add_0)            0.00       9.21 r
S[27] (out)                                             0.00       9.21 r
data arrival time                                                  9.21
--------------------------------------------------------------------------
```

(b)

Figure 4.2.1.1: (a) Kogge Stone max paths, (b) Lib Adder max paths

In figure 42.1.1 we can see that the max path in the implementation of the kogge stone adder is 2.15 nsec, and for the library adder its 9.21 nsec which means it's nearly five times the kogge stone adder delay which is perfect to us. Note that the comparison is made by the 28-Bit kogge stone adder and a 28-Bit library adder.

```
--------------------------------------------------------------------------------
                                   Switch   Int      Leak     Total
Hierarchy                          Power    Power    Power    Power    %
--------------------------------------------------------------------------------
Kogge_Stone_ADDER_28               7.51e-02    0.208 1.56e+06    0.285 100.0
```

(a)

```
--------------------------------------------------------------------------------
                                   Switch   Int      Leak     Total
Hierarchy                          Power    Power    Power    Power    %
--------------------------------------------------------------------------------
add                                1.05e-02    0.170 8.56e+05    0.182 100.0
```

(b)

Figure 4.2.1.2: (a) Power of KSA, (b) Power of Lib Adder in mW

We can see the cost of the delay optimization in figure 4.2.1.2, where the power dissipation of the kogge stone adder is nearly double of that of the library adder.

```
Total cell area:                2393.407851
Total area:                   undefined
```

(a)

```
Total cell area:                  847.224003
Total area:                   undefined
```

(b)

Figure4.2.1.3: (a) KSA area, (b) Lib Adder area

Another cost paid in the required area by the kogge stone adder which is more than the double of the other adder.

Wallace Tree Multiplier VS TSMC Multiplier

```
U22/CO (ADDFX1M)                                        0.33       9.67 f
U21/CO (ADDFX1M)                                        0.31       9.98 f
U20/Y (XOR3XLM)                                         0.45      10.43 f
C[27] (out)                                             0.00      10.43 f
data arrival time                                                 10.43
-------------------------------------------------------------------------
```

(a)

```
DP_OP_30J1_125_2485_U22/CO (ADDFX1M)                    0.33      10.78 f
DP_OP_30J1_125_2485_U21/CO (ADDFX1M)                    0.31      11.09 f
U296/Y (XOR2XLM)                                        0.14      11.23 f
U305/Y (MX2XLM)                                         0.26      11.49 f
result_reg[27] (out)                                    0.00      11.49 f
data arrival time                                                 11.49
-------------------------------------------------------------------------
```

(b)

Figure4.2.1.4: (a) WTM max path, (b) Lib Multiplier max path.

```
-------------------------------------------------------------------------------
                                   Switch   Int      Leak     Total
Hierarchy                          Power    Power    Power    Power    %
-------------------------------------------------------------------------------
WallaceTreeMul                        0.419    2.058 8.54e+06    2.486 100.0
```

(a)

Hierarchy	Switch Power	Int Power	Leak Power	Total Power	%
mult	0.527	1.647	9.55e+06	2.183	100.0

Figure4.2.1.5: (a) WTM power, (b) Lib Multiplier power in mW

```
Total cell area:                  9670.120663
Total area:                     undefined
```

(a)

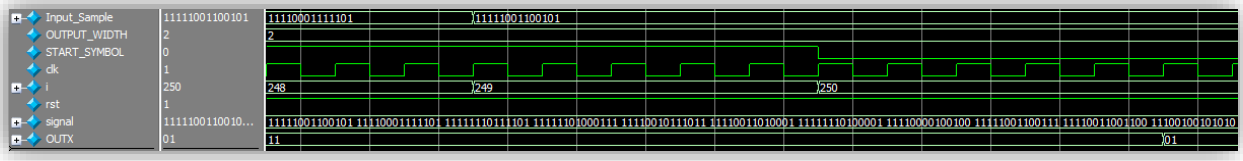
```
Total cell area:                 12664.822251
Total area:                     undefined
```

(b)

Figure4.2.1.6: (a) WTM area (b) Lib Multiplier area

From the last three figures we conclude that the designed wallace tree multiplier is more efficient in terms of the delay and the area consumed, while in terms of the power we can say that both are the same.

Decoder Simulation



This simulation is done using a signal of Symbol 2 (01) with SNR equals 2dB. The output OUTX changed to “01” by the end of the input samples after 250 steps.

General Receiver Results

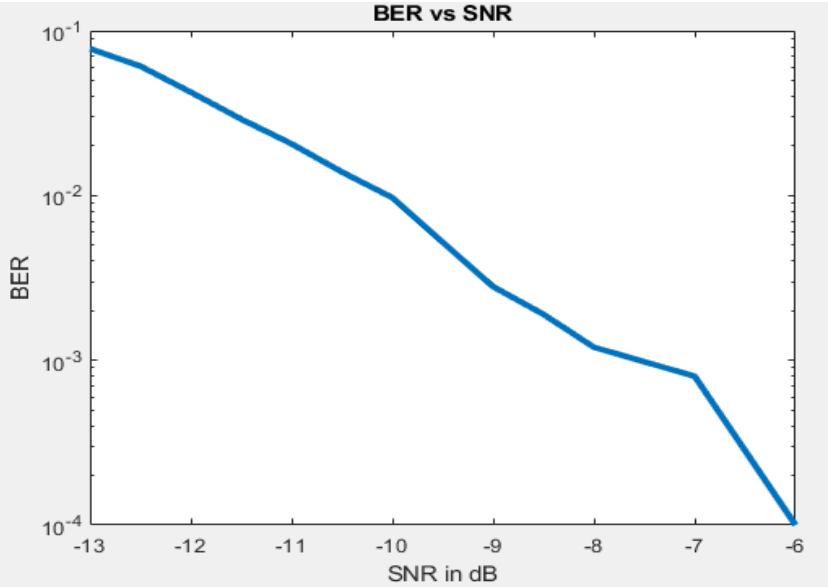


Figure 4.2.1.7: Simulation BER

In Figure 4.2.1.7, the ideal performance of the new decoder is shown. What I mean by the ideal is that this graph is the output of simulation that uses 64-Bit floating-point numbers with very high resolution. So, to get closer to this result by the implementation we have only to use more fractional bits in the fixed-point implementation.

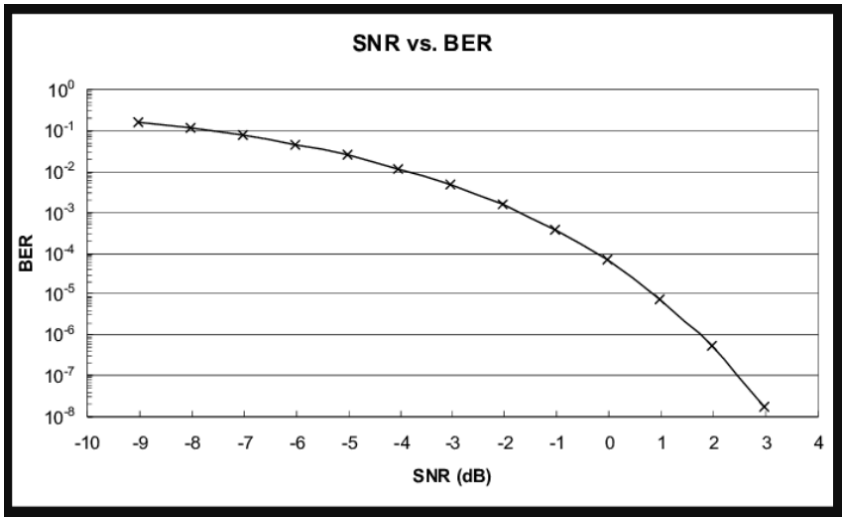


Figure 4.2.1.8: Traditional Rx BER

Now we need to measure how much the result in Figure 4.2.1.7 is better than the other performances. So, we compare with the performance in Figure 4.2.1.8 from [13]. We can say that the new decoder can achieve the same BER at SNR smaller than the traditional receiver by 6 dB which is great.

Figure 4.2.1.9: Implementation BER

In Figure 4.2.1.10, we report that the degradation in the performance due to the implementation is nearly 10 dB. And we need to mention that the problem is not with the efficiency of the implementation, but with the very precise resolution needed to achieve the performance in Figure 4.2.1.7. As we mentioned in chapter 3 “implementation”, the fixed-point number’s fractional field is of length 10 bits but to get the performance in Figure 4.2.1.7 this length should not be less than 24 bits.

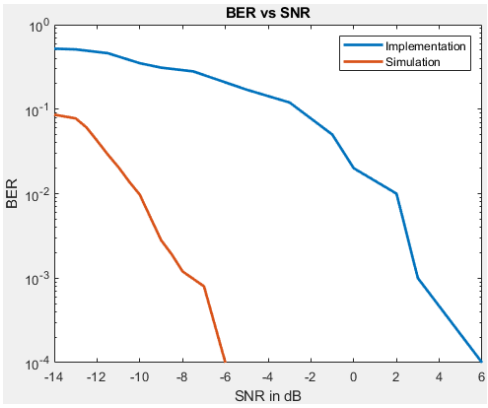


Figure 4.2.1.10

ASIC Reports

The RTL design is synthesized by **Design Compiler** tool from Synopsys using three versions of 130 nano meter tsmc library which are SS, FF, TT to test the best and worst path delay under the following constrains:

1- CLK period 25nsec which means CLK frequency 40MHz.
2- CLK setup uncertainty or worst negative skew 0.2nsec.
3- CLK hold uncertainty or worst positive skew 0.1nsec.
4- CLK raising and falling transition time 0.05nsec.
5- Input and Output delay 5nsec.
6- Setting wire delay model from the SS library.

The power results:

Module	Total Power	Static Power
Comparator	0.036 mW	1.15e+7 pW
Output Layer	0.732 mW	0.484 mW
LSTM Layer 1	8.254 mW	4.92 mW
LSTM Layer 2	17.548 mW	3.2 mW
NN_TOP	26.6 mW	8.62 mW

From the previous table we can see the price paid to get the performance in Figure 4.2.1.9. This incredible power report is the result of the huge number of parallel adders and multipliers. The LSTM layer 2 consumes power more than double that of Layer 1, and that is because of the difference in number of inputs so layer 2 has more multipliers and adders.

For comparison we can look at the following table which contain the power results of traditional ZigBee transceiver that was a 2023 Graduation Project which reported the total power of the receiver is 1.65mW only which means that our decoder only consumes those times sixteen:

Module	Total Power	Dynamic Power	Static Power
Rx Top	1.65 mW	0.93 mW	0.72 mW
Control Unit	0.09 mW	0.05 mW	0.04 mW
Correlator Bank	0.18 mW	0.1 mW	0.08 mW
Data Sampler	0.02 mW	0.01 mW	0.01 mW
Symbol Detection	0.02 mW	0.01 mW	0.01 mW
Matched Filters	1.09 mW	0.53 mW	0.56 mW
Delay I	0.27 mW	0.24 mW	0.03 mW

The area results:

Module	**Total Cell Area**
Comparator	0.0001 cm2
Output Layer	0.0051 cm2
LSTM Layer 1	0.0340 cm2
LSTM Layer 2	0.0513 cm2
NN_TOP	0.0907 cm2

The timing results:

Setup or Hold	**No. max/min paths**	**Max/min path slack**
Setup	20	0.00 nsec
Hold	20	0.63 nsec

We need to mention that the generated netlist is functionally equivalent to the RTL and this verification is done using the Formality tool from Synopsys.

FPGA Reports

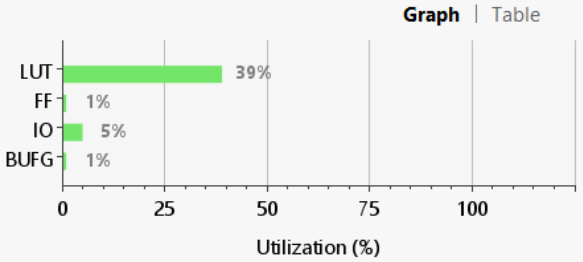
To evaluate the performance and functionality of our design to ensure the accuracy and efficiency, we implemented it on a Field-Programmable Gate Array (FPGA) of type "ZYNQ ULTRASCALE+ ZCU104 Evaluation Board" with frequency 38MHz and the reports of the post implementation design are generated by "Xilinx Viv ado" tool as flows:

Power	
Total On-Chip Power:	**0.846 W**
Junction Temperature:	**25.8 °C**
Thermal Margin:	74.2 °C (75.0 W)
Effective ϑJA:	1.0 °C/W
Power supplied to off-chip devices:	0 W

Timing	**Setup**
Worst Negative Slack (WNS):	11.407 ns
Total Negative Slack (TNS):	0 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	10194

Timing	Setup | **Hold**
Worst Hold Slack (WHS):	0.052 ns
Total Hold Slack (THS):	0 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	10194

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start
✓ synth_1	constrs_2	synth_design Complete!								90950	4167	0.0	0	0	7/7/24, 2:57 PM
✓ impl_1	constrs_2	write_bitstream Complete!	11.407	0.000	0.052	0.000	0.000	0.846	0	90259	4167	0.0	0	0	7/7/24, 3:24 PM



4.2.2 Chip to symbol

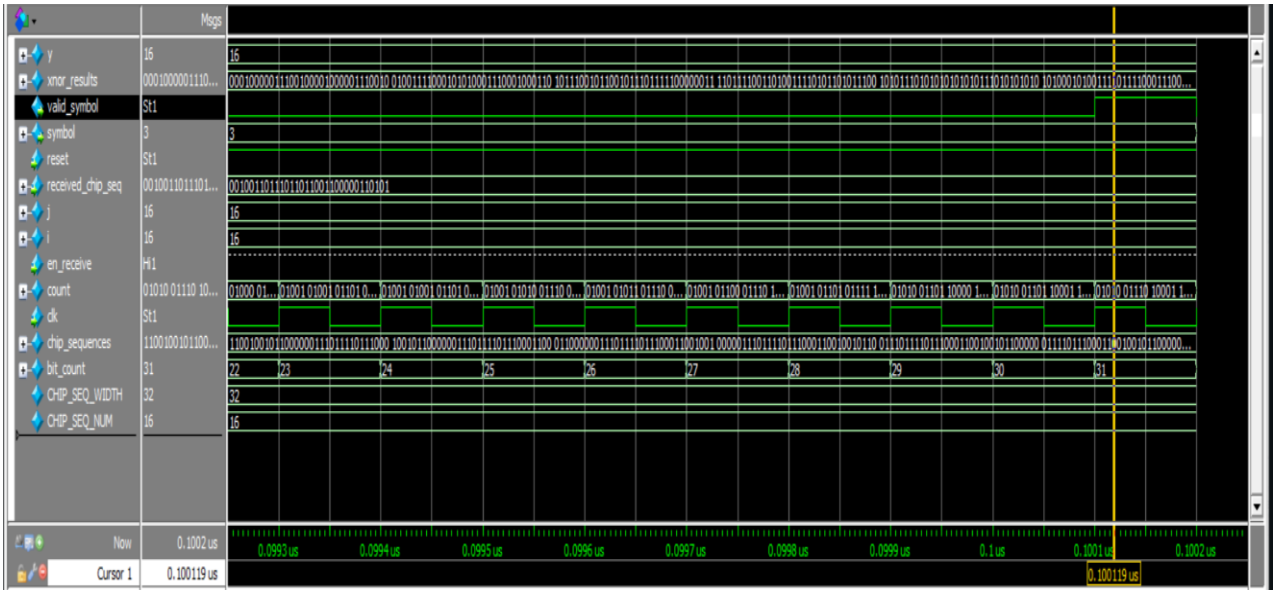


Figure 4.2.2.1. simulation

As shown in Fig 4.2.2.1 that received chip sequence was entered with 3 wrong bits, the block continues to compare between bits until counter reach to 31 then valid symbol be 1 so it indicates that now it will get out the actual received symbol that is 3 in this case so it takes 32 cycle to get out the right symbol

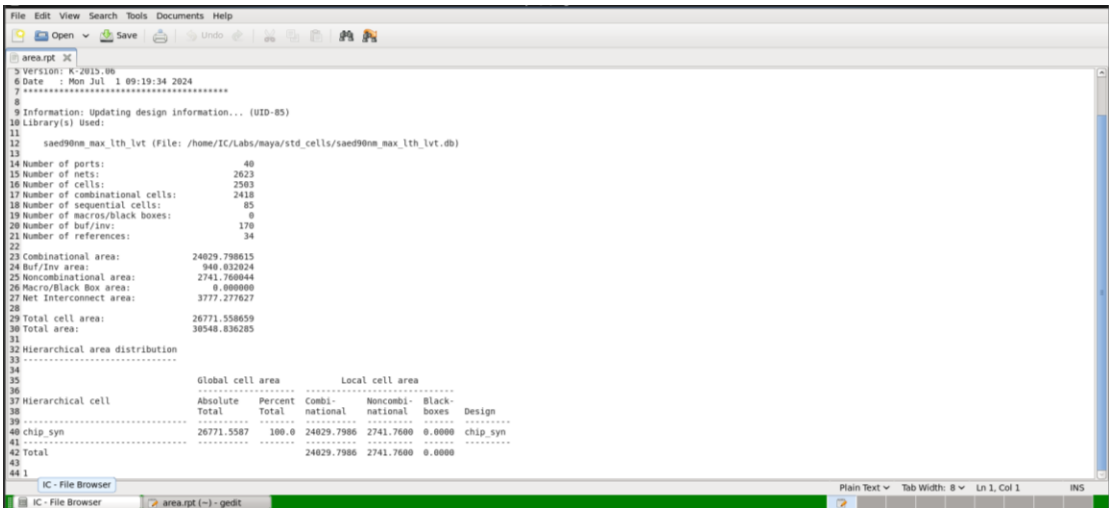


Figure 6.2.2.2.Area Power

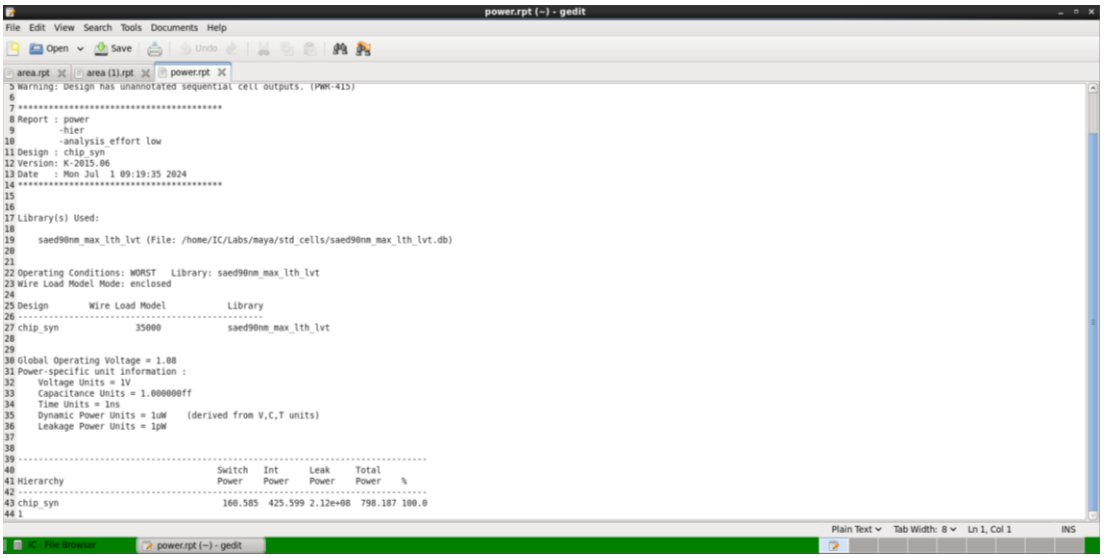


Figure 4.2.2.3. Power Report

4.2.3 Symbol to bit

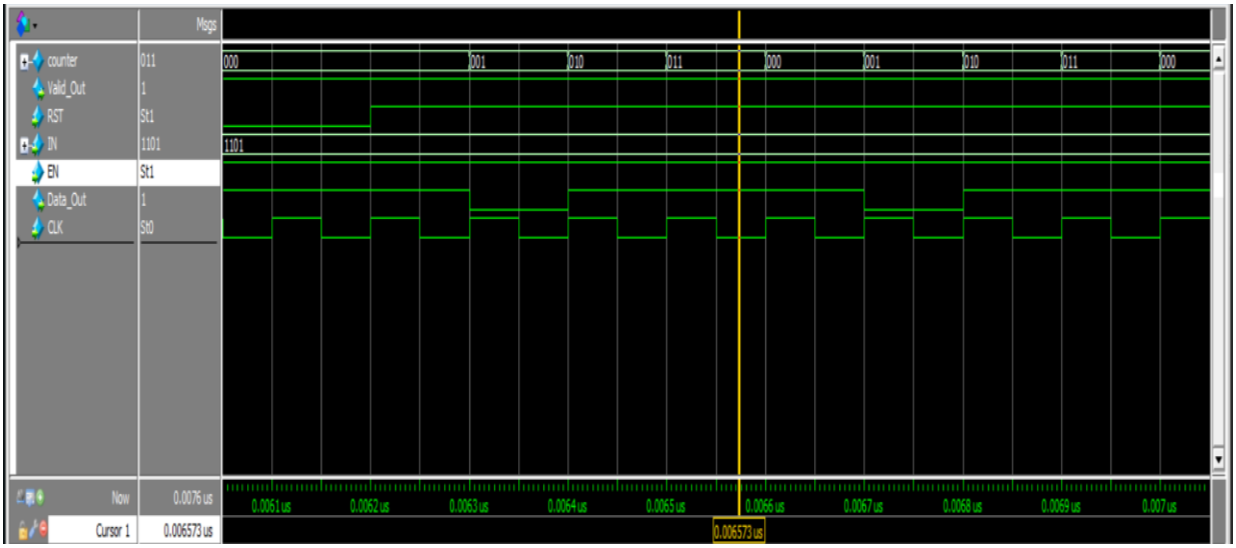


Figure4.2.3.1. Simulation

As shown in Fig 4.2.3.1 that entered symbol is 1101 and counter start to count when EN and RST are 1 after that 1 bit is out each clock cycle until counter reach to 3 it stops so it takes 4 clock cycles to get out all bits

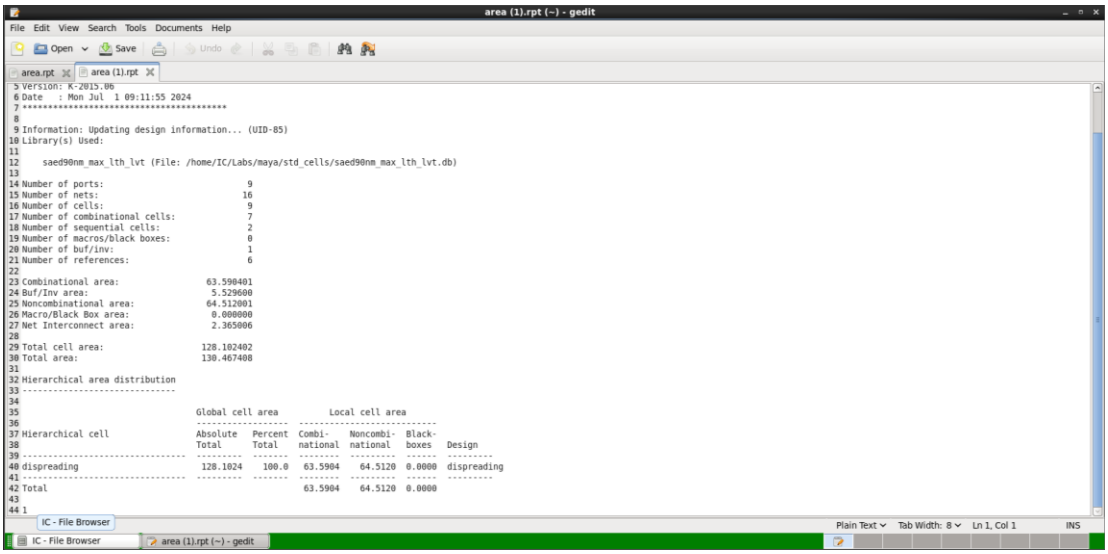


Figure4.2.3.2. Area Report

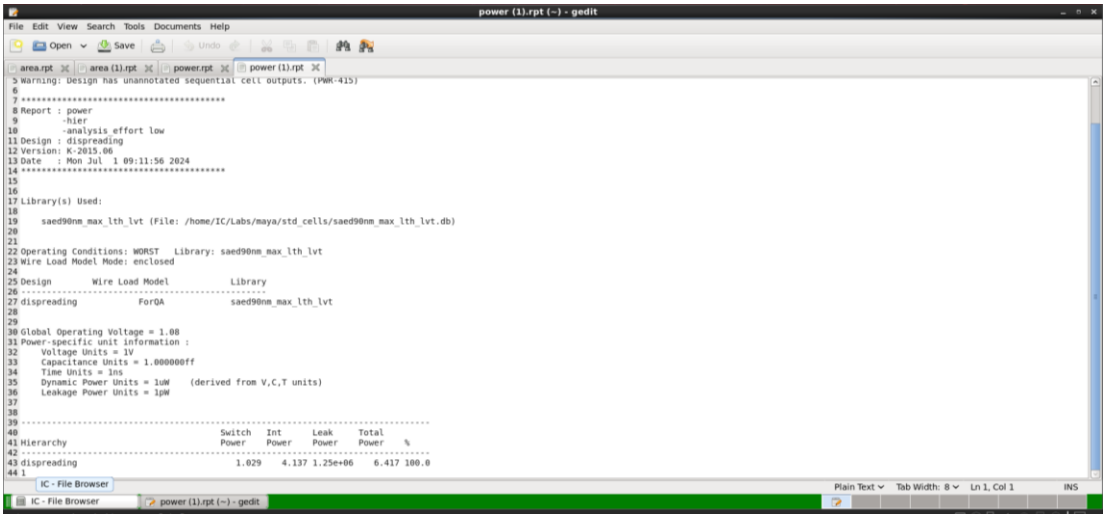


Figure4.2.3.3. Power Report

4.2.4 Frequency offset estimation

We created a MATLAB simulation for this algorithm, and we calculated the difference between the inverse cosine function and our implementation of it, as shown in Figure 4.2.4.1 we noticed that this difference is very small.

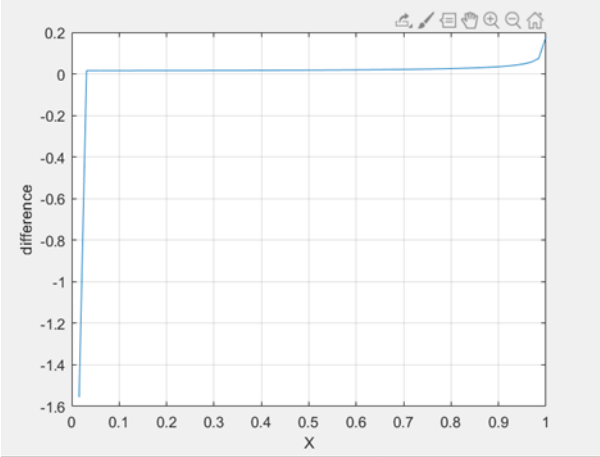


Figure 4.2.4.1: difference between the inverse cosine function and our implementation of it

We toke the samples and converted it to 14-bits fixed point signed number by creating a MATLAB function for doing this, after that we toke these samples and tested our architecture with them, we found that the output of the simulation is the same of the output of the hardware architecture.

Frequency offset estimator block takes eight consecutive samples as an input, every sample is

14-bit length, and gives us a 28-bit output which is the estimated frequency.

It takes 13 clock cycles for this block to give the output as shown in the Figure 4.2.4.2

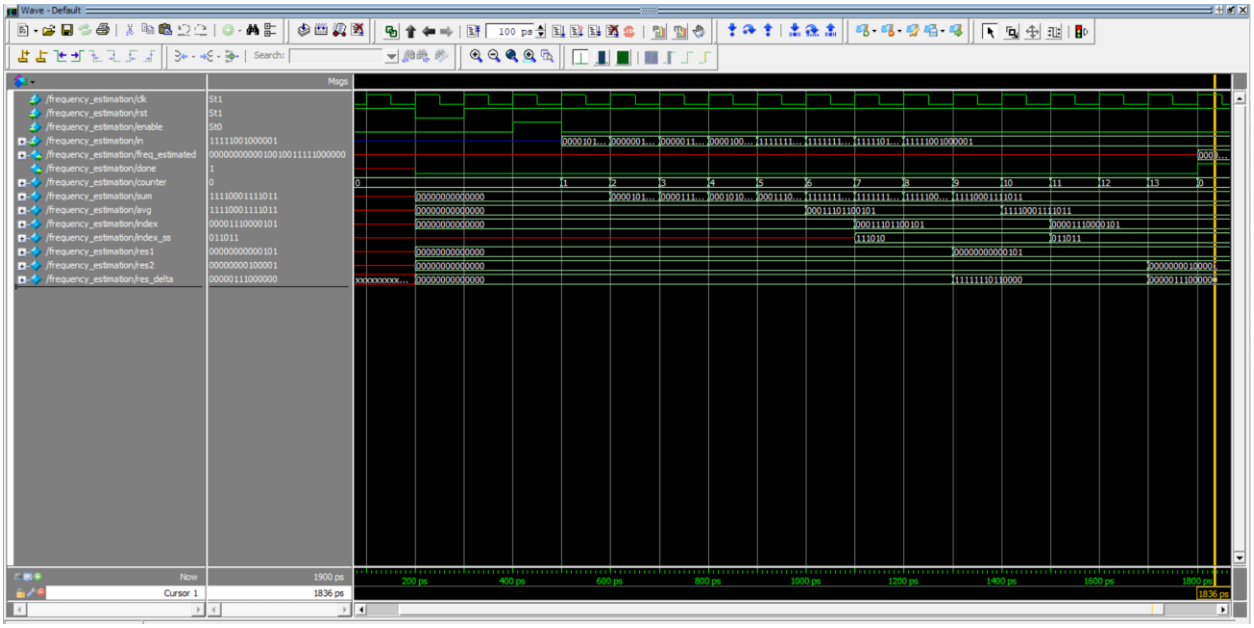


Figure4.2.4.2: frequency offset estimator's simulation

As shown in Figure 4.2.4.3 the output of the hardware block is the same as the output from the MATLAB simulator.

```
estimated_freq_noise =

   1.0867e+07
```

Figure 4.2.4.3: frequency estimated from MATLAB simulation

After that we did synthesis using ICC2 compiler we got the gate-level netlist, power and area reports as shown in figures 4.2.4.4 and 4.2.4.5 and 4.2.4.6 and 4.2.4.7

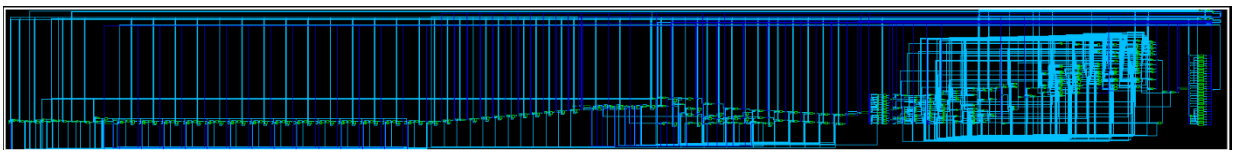


Figure 4.2.4.4: gate level netlist

```
****************************************
Report : area
Design : frequency_estimation
Version: K-2015.06
Date   : Sun Jul  7 02:10:40 2024
****************************************

Library(s) Used:

    scmetro_tsmc_cl013g_rvt_ss_1p08v_125c (File: /home/IC/Assignments/system/std_cells/scmetro_tsmc_cl013g_rvt_ss_1p08v_125c.db)

Number of ports:                          405
Number of nets:                          1899
Number of cells:                         1266
Number of combinational cells:           1148
Number of sequential cells:               112
Number of macros/black boxes:               0
Number of buf/inv:                        132
Number of references:                      46

Combinational area:              18531.848373
Buf/Inv area:                      465.973214
Noncombinational area:            2905.272333
Macro/Black Box area:               0.000000
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:                 21437.120707
Total area:                 undefined

Hierarchical area distribution
------------------------------

                                  Global cell area           Local cell area
                                  -----------------  ----------------------------
Hierarchical cell                 Absolute    Percent  Combi-      Noncombi-  Black-
                                  Total       Total    national    national   boxes   Design
--------------------------------  ----------  -------  ----------  ---------  ------  ------------------------------
frequency_estimation              21437.1207    100.0   2149.8309  2905.2723  0.0000  frequency_estimation
mult_117                           2535.7885     11.8   2535.7885     0.0000  0.0000  frequency_estimation_DW02_mult_2
mult_186                           1039.0261      4.8   1039.0261     0.0000  0.0000  frequency_estimation_DW02_mult_0
r120                                405.9615      1.9    405.9615     0.0000  0.0000  frequency_estimation_DW01_add_3
r153                              11819.9515     55.1  11819.9515     0.0000  0.0000  frequency_estimation_DW02_mult_1
r159                                261.2274      1.2    261.2274     0.0000  0.0000  frequency_estimation_DW01_add_1
sub_179                             320.0624      1.5    320.0624     0.0000  0.0000  frequency_estimation_DW01_sub_1
--------------------------------  ----------  -------  ----------  ---------  ------  ------------------------------
Total                                                  18531.8484  2905.2723  0.0000

1
```

Figure 4.2.4.5: area report

```
****************************************
Report : power
       -hier
       -analysis_effort low
Design : frequency_estimation
Version: K-2015.06
Date   : Sun Jul  7 02:10:42 2024
****************************************


Library(s) Used:

    scmetro_tsmc_cl013g_rvt_ss_1p08v_125c (File: /home/IC/Assignments/system/std_cells/scmetro_tsmc_cl013g_rvt_ss_1p08v_125c.db)


Operating Conditions: scmetro_tsmc_cl013g_rvt_ss_1p08v_125c   Library: scmetro_tsmc_cl013g_rvt_ss_1p08v_125c
Wire Load Model Mode: top


Global Operating Voltage = 1.08
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1pW


--------------------------------------------------------------------------------
                                   Switch   Int      Leak     Total
Hierarchy                          Power    Power    Power    Power    %
--------------------------------------------------------------------------------
frequency_estimation               4.43e-03    0.151 1.71e+07    0.172 100.0
  sub_179 (frequency_estimation_DW01_sub_1)
                                     0.000    0.000 3.05e+05 3.05e-04   0.2
  mult_117 (frequency_estimation_DW02_mult_2)
                                     0.000    0.000 2.29e+06 2.29e-03   1.3
  r120 (frequency_estimation_DW01_add_3)
                                   2.08e-03 3.50e-02 4.04e+05 3.75e-02  21.8
  r153 (frequency_estimation_DW02_mult_1)
                                     0.000    0.000 1.00e+07 1.00e-02   5.8
  r159 (frequency_estimation_DW01_add_1)
                                     0.000    0.000 2.59e+05 2.59e-04   0.2
  mult_186 (frequency_estimation_DW02_mult_0)
                                     0.000    0.000 8.59e+05 8.59e-04   0.5
1
```

Figure 4.2.4.6: power report

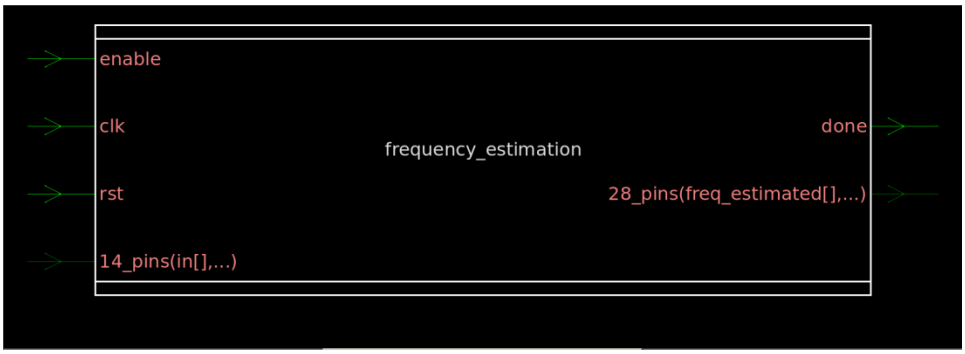


Figure 4.2.4.6: top module

4.2.5 Packet edge detection

Figure 4.2.5.1 shows the simulation of the packet edge detection block. In this figure we see the RNN start signal is one and the value of correlation is 62.08 when we enter all eight symbols like that in the preamble.

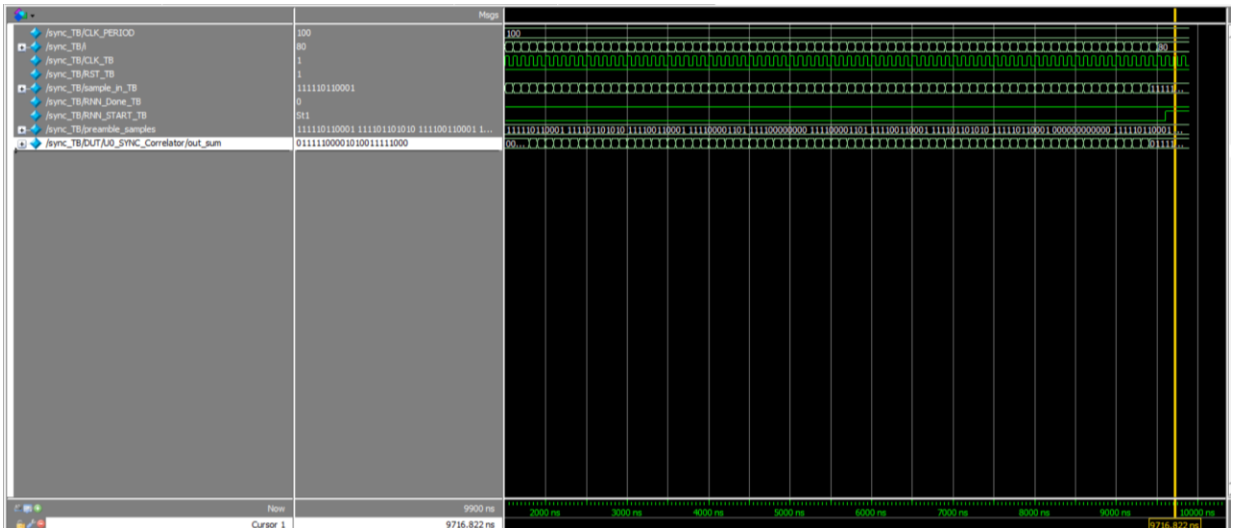


Figure4.2.5.1 simulation result of packet edge detector.

After that we do synthesis using ICC2 compiler we got gate-level netlist and power and area reports are shown in figures 4.2.5.2 and 4.2.5.3 and 4.2.5.4

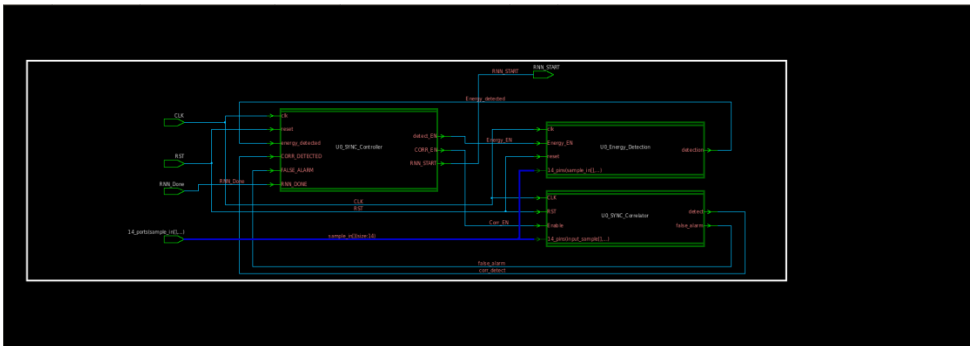


Figure 4.2.5.3: gate level netlist

```
13 Date   : Sun Jun 30 21:51:41 2024
14 ****************************************
15
16
17 Library(s) Used:
18
19     scmetro_tsmc_cl013g_rvt_ss_1p08v_125c (File: /home/IC/Assignments/system/std_cells/scmetro_tsmc_cl013g_rvt_ss_1p08v_125c.db)
20
21
22 Operating Conditions: scmetro_tsmc_cl013g_rvt_ss_1p08v_125c   Library: scmetro_tsmc_cl013g_rvt_ss_1p08v_125c
23 Wire Load Model Mode: top
24
25
26 Global Operating Voltage = 1.08
27 Power-specific unit information :
28     Voltage Units = 1V
29     Capacitance Units = 1.000000pf
30     Time Units = 1ns
31     Dynamic Power Units = 1mW    (derived from V,C,T units)
32     Leakage Power Units = 1pW
33
34
35 --------------------------------------------------------------------------------
36                                   Switch   Int      Leak     Total
37 Hierarchy                          Power    Power    Power    Power    %
38 --------------------------------------------------------------------------------
39 TX_TOP                            2.15e-03 4.47e-02 1.71e+06 4.85e-02 100.0
40   U0_Pulse_shaping_odd (Pulse_shaping_odd)
41                                      0.000 1.29e-02 2.63e+05 1.32e-02  27.2
42   U0_Pulse_shaping_even (Pulse_shaping_even)
43                                      0.000 1.29e-02 2.71e+05 1.32e-02  27.2
44   U0_Even_odd_Gen (Even_odd_Gen)    2.41e-06 2.69e-03 4.62e+05 3.16e-03   6.5
45     add_40 (Even_odd_Gen_DW01_inc_0)     0.000    0.000 8.47e+04 8.47e-05   0.2
46   U0_T_FF (T_FF)                    4.82e-05 4.99e-04 1.58e+04 5.63e-04   1.2
47   U0_symbol2chip (symbol2chip)         0.000 2.05e-04 1.04e+04 2.16e-04   0.4
48   U0_EN_SYNC (EN_sync)                 0.000 1.28e-03 4.27e+04 1.32e-03   2.7
49   U0_Bit2Symbol (Bit2Symbol)        3.72e-04 3.71e-03 1.87e+05 4.27e-03   8.8
50   U1_ClkDiv (ClkDiv_1)              1.48e-04 1.54e-03 2.28e+05 1.92e-03   4.0
51   U0_ClkDiv (ClkDiv_0)              1.58e-03 8.87e-03 2.30e+05 1.07e-02  22.0
52 1
```

Figure4.2.5.3 power report.

area.rpt

```
U0_SYNC_Correlator/mult_68_I70      3305.3503      0.9    2878.2082      0.0000  0.0000  SYNC_Correlator_DW02_mult_10
U0_SYNC_Correlator/mult_68_I70/FS_1
                                    427.1421       0.1     427.1421      0.0000  0.0000  SYNC_Correlator_DW01_add_55
U0_SYNC_Correlator/mult_68_I72      3305.3503      0.9    2878.2082      0.0000  0.0000  SYNC_Correlator_DW02_mult_8
U0_SYNC_Correlator/mult_68_I72/FS_1
                                    427.1421       0.1     427.1421      0.0000  0.0000  SYNC_Correlator_DW01_add_52
U0_SYNC_Correlator/mult_68_I73      3340.6513      0.9    2913.5092      0.0000  0.0000  SYNC_Correlator_DW02_mult_7
U0_SYNC_Correlator/mult_68_I73/FS_1
                                    427.1421       0.1     427.1421      0.0000  0.0000  SYNC_Correlator_DW01_add_51
U0_SYNC_Correlator/mult_68_I74      2914.6859      0.8    2487.5438      0.0000  0.0000  SYNC_Correlator_DW02_mult_6
U0_SYNC_Correlator/mult_68_I74/FS_1
                                    427.1421       0.1     427.1421      0.0000  0.0000  SYNC_Correlator_DW01_add_49
U0_SYNC_Correlator/mult_68_I75      2914.6859      0.8    2487.5438      0.0000  0.0000  SYNC_Correlator_DW02_mult_5
U0_SYNC_Correlator/mult_68_I75/FS_1
                                    427.1421       0.1     427.1421      0.0000  0.0000  SYNC_Correlator_DW01_add_48
U0_SYNC_Correlator/mult_68_I76      2002.7434      0.5    1575.6013      0.0000  0.0000  SYNC_Correlator_DW02_mult_4
U0_SYNC_Correlator/mult_68_I76/FS_1
                                    427.1421       0.1     427.1421      0.0000  0.0000  SYNC_Correlator_DW01_add_46
U0_SYNC_Correlator/mult_68_I77      2914.6859      0.8    2487.5438      0.0000  0.0000  SYNC_Correlator_DW02_mult_3
U0_SYNC_Correlator/mult_68_I77/FS_1
                                    427.1421       0.1     427.1421      0.0000  0.0000  SYNC_Correlator_DW01_add_45
U0_SYNC_Correlator/mult_68_I78      2914.6859      0.8    2487.5438      0.0000  0.0000  SYNC_Correlator_DW02_mult_2
U0_SYNC_Correlator/mult_68_I78/FS_1
                                    427.1421       0.1     427.1421      0.0000  0.0000  SYNC_Correlator_DW01_add_43
U0_SYNC_Correlator/mult_68_I79      3340.6513      0.9    2913.5092      0.0000  0.0000  SYNC_Correlator_DW02_mult_1
U0_SYNC_Correlator/mult_68_I79/FS_1
                                    427.1421       0.1     427.1421      0.0000  0.0000  SYNC_Correlator_DW01_add_42
U0_SYNC_Correlator/mult_68_I8       2914.6859      0.8    2487.5438      0.0000  0.0000  SYNC_Correlator_DW02_mult_72
U0_SYNC_Correlator/mult_68_I8/FS_1
                                    427.1421       0.1     427.1421      0.0000  0.0000  SYNC_Correlator_DW01_add_148
U0_SYNC_Correlator/mult_68_I80      3305.3503      0.9    2878.2082      0.0000  0.0000  SYNC_Correlator_DW02_mult_0
U0_SYNC_Correlator/mult_68_I80/FS_1
                                    427.1421       0.1     427.1421      0.0000  0.0000  SYNC_Correlator_DW01_add_1
U0_SYNC_Correlator/mult_68_I9       3340.6513      0.9    2913.5092      0.0000  0.0000  SYNC_Correlator_DW02_mult_71
U0_SYNC_Correlator/mult_68_I9/FS_1
                                    427.1421       0.1     427.1421      0.0000  0.0000  SYNC_Correlator_DW01_add_147
--------------------------------  -----------  -------  -----------  ----------  ------  ----------------------------
Total                                                   348340.8570  33500.6486  0.0000

1
```

Figure4.2.5.4 Area report.

4.3 AES

First, we used an example to check how our results meet the AES standard in this example
Plaintext: 54 77 6F 20 4F 6E 65 20 4E 69 6E 65 20 54 77 6F

KEY (cipher key): 54 68 61 74 73 20 6D 79 20 4B 75 6E 67 20 46 75

Ciphertext: 29 C3 50 5F 57 14 20 F6 40 22 99 B3 1A 02 D7 3A

As we said in implementation section 3.3.1. that the encryption process start when the start port is turned 1’b1 and the enc_dec port is turned 1’b1 too. Figure 4.3.1 shows that the input plaintext entered the architecture of the cells in 4 clock cycles from start port activated

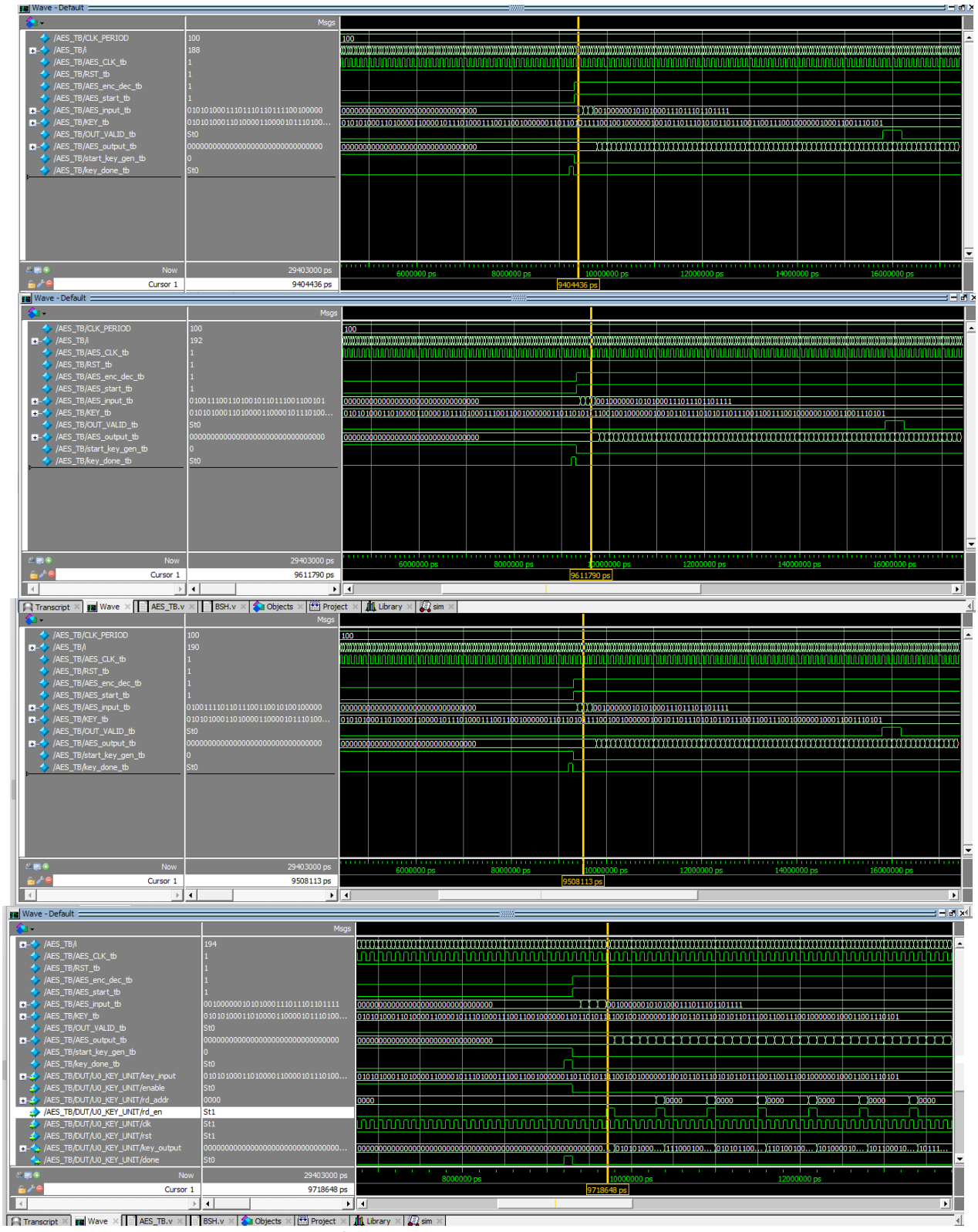
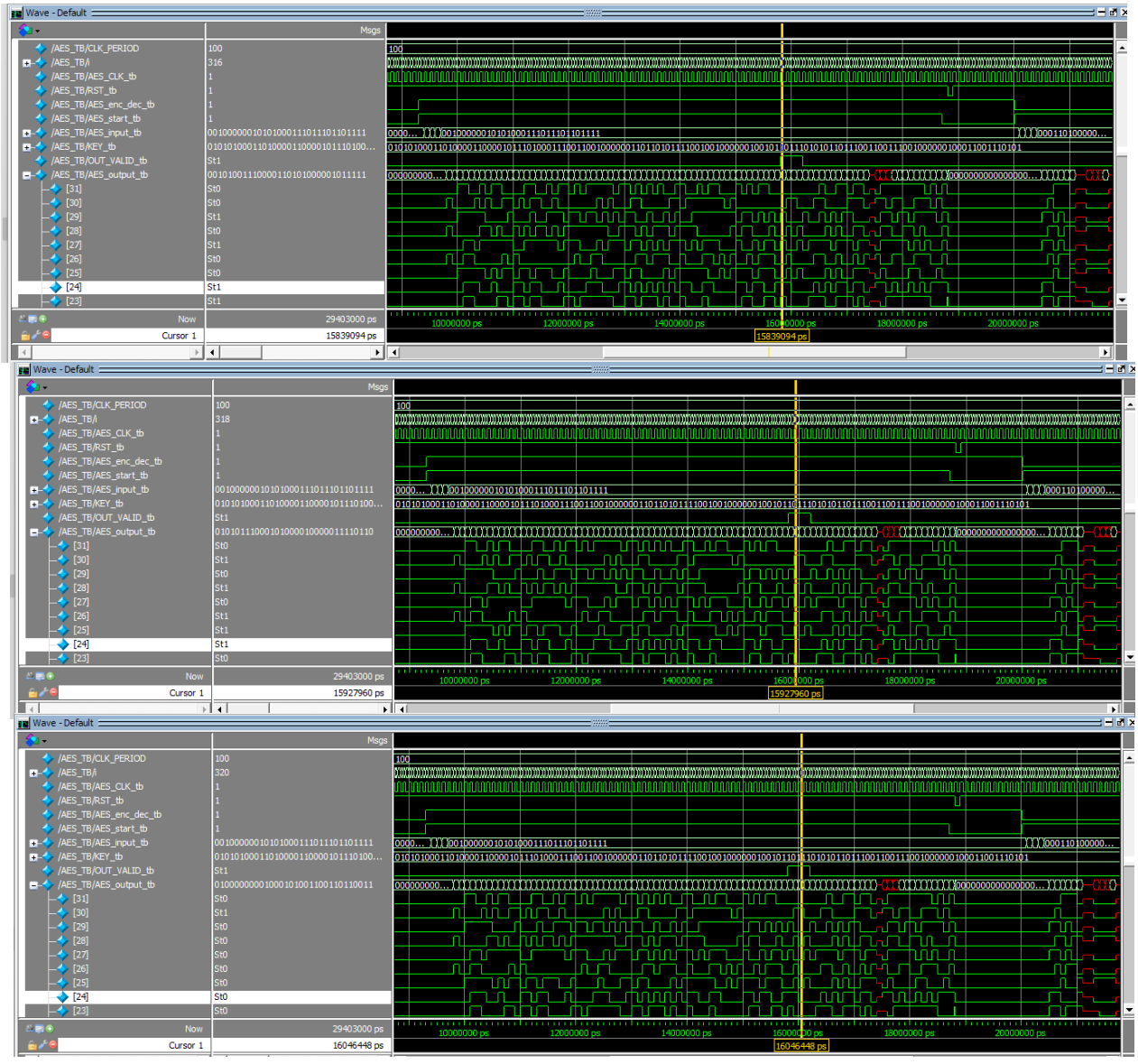


Figure 4.3.1: Simulation of Input entry over 4 clock cycles during encryption process

After that the encryption process start from add round key which takes one clock cycle in initial round to the main 9 rounds which consists of 4 functions (sub bytes, shift rows, mix columns and add round key) which takes six clock cycles for each round, then the last round which consists of 3 functions (sub bytes, shift rows and add round key) which takes five clock cycles, so we can say the total rounds take sixty clock cycles then the out_valid turned On so

the output port carries the ciphertext over another 4 clock cycles and Figure 4.3.2. shows that the output ciphertext gauarntees the example ciphertext.



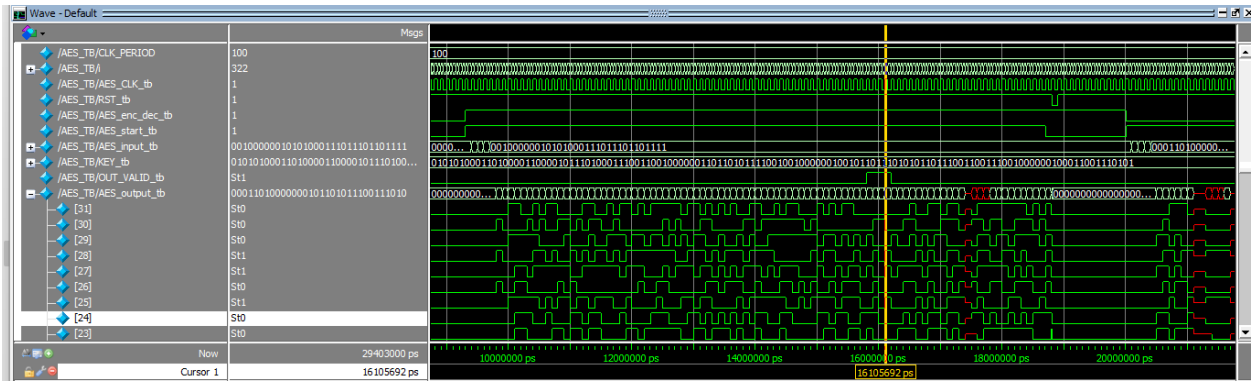
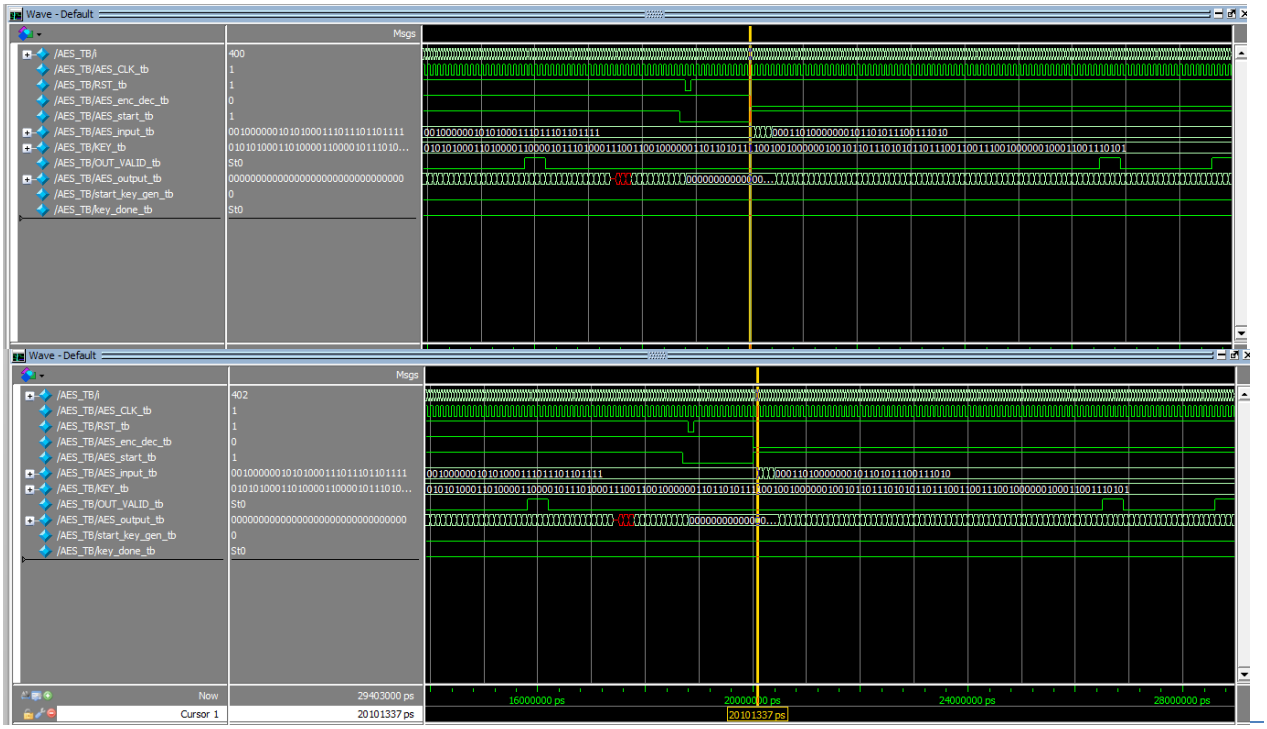


Figure4.3.2: The simulation of the ciphertext over 4 clock cycles during encryption process

Finally we can say the total encryption process including the input and output transitions of data cells between architecture and interfaces takes 68 clock cycles.

Let's now check the decryption process we will take the ciphertext as an input to check if the decryption works well or not if the output of the decryption process is the plaintext then the decryption process works well. To start the decryption process we enabled the start port and activate enc_dec port to 1'b0 to start the decryption process. Figure 4.3.3.shows that the input ciphertext entered the architecture of the cells in 4 clock cycles from start port activated



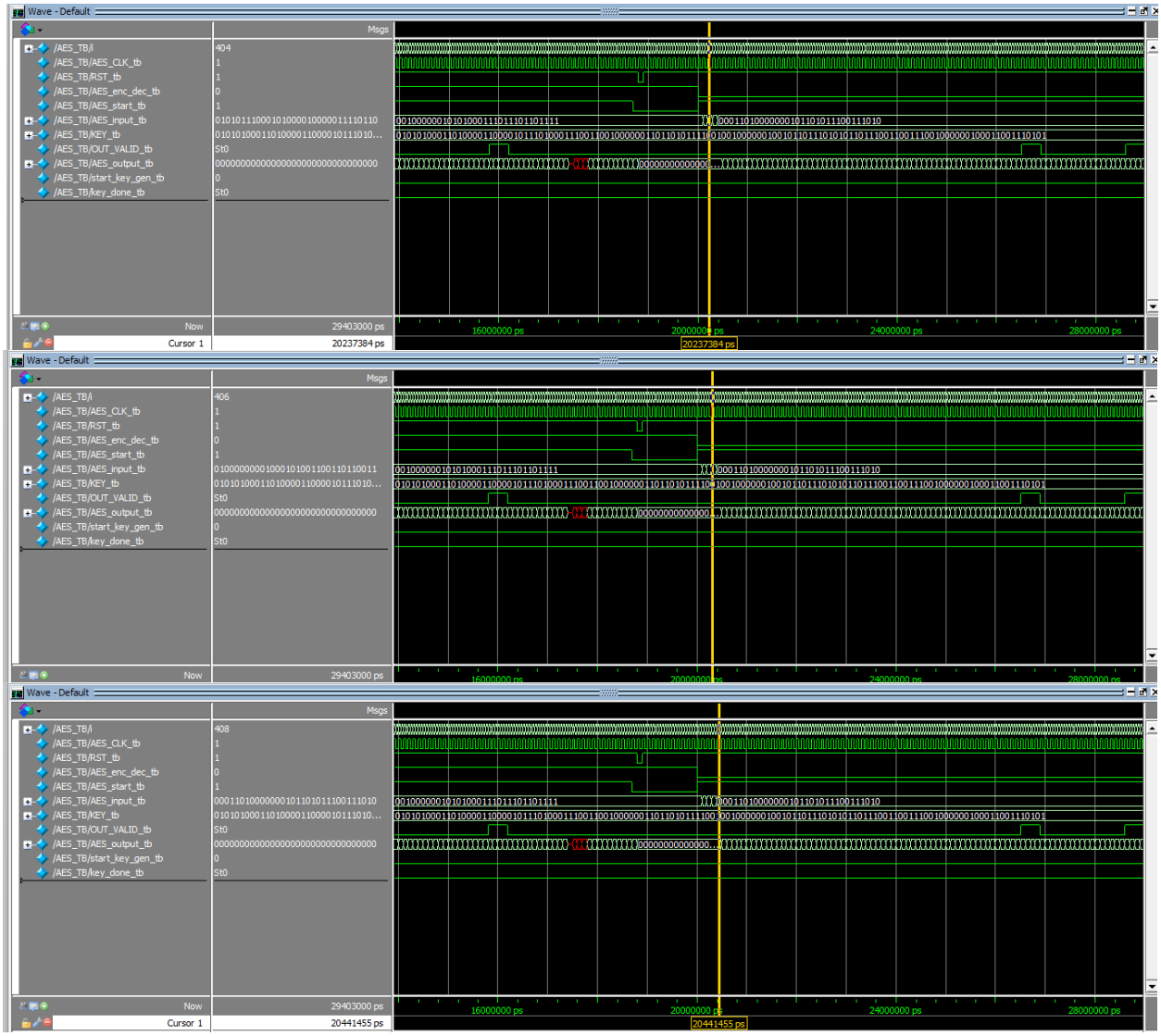


Figure4.3.3: Simulation of Input entry over 4 clock cycles during decryption process

After that the decryption process start from add round key which takes one clock cycle in initial round to the main 9 rounds which consists of 4 functions (inv-shift rows, inv-sub bytes,inv-mix columns and add round key) which takes six clock cycles for each round, then the last round which consists of 3 functions (inv-shift rows, inv-sub bytes,and add round key) which takes five clock cycles, so we can say the total rounds take sixty clock cycles then the out_valid turned On so the output port carries the plaintext over another 4 clock cycles and

Figure 4.3.4. shows that the output plaintext gauarntees the example plaintext. Finally we can say the total decryption process including the input and output transitions of data cells between architecture and interfaces takes 68 clock cycles.

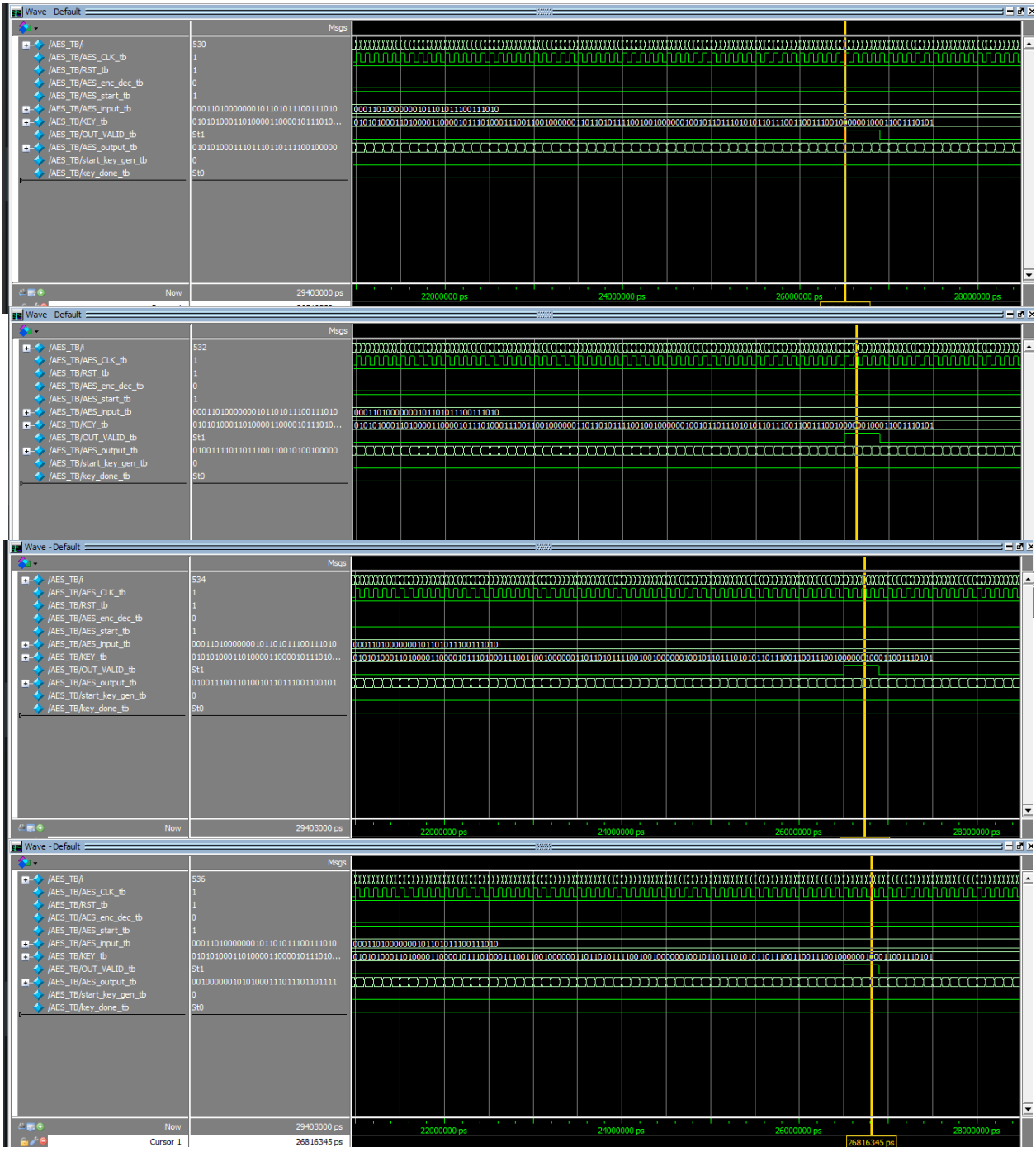


Figure4.3.4: The simulation of the plaintext over 4 clock cycles during decryption process

After that we do synthesis using ICC2 compiler with a clock period 150 ns (with no violations in setup and hold time) also power and area reports are shown in figures 4.3.5 and 4.3.6

```
****************************************
Report : area
Design : AES_TOP
Version: K-2015.06
Date   : Sat Jul  6 20:08:06 2024
****************************************

Information: Updating design information... (UID-85)
Warning: Design 'AES_TOP' contains 1 high-fanout nets. A fanout number of 1000 will be used for delay calculations involving these nets. (TIM-134)
Library(s) Used:

    scmetro_tsmc_cl013g_rvt_ss_1p08v_125c (File: /home/IC/Labs/AES/std_cells/scmetro_tsmc_cl013g_rvt_ss_1p08v_125c.db)

Number of ports:                         4678
Number of nets:                         16181
Number of cells:                        11481
Number of combinational cells:           9350
Number of sequential cells:              1964
Number of macros/black boxes:               0
Number of buf/inv:                       1587
Number of references:                       6

Combinational area:              67214.281390
Buf/Inv area:                     7079.027297
Noncombinational area:           57265.283936
Macro/Black Box area:                0.000000
Net Interconnect area:         1536694.581879

Total cell area:                124479.565325
Total area:                    1661174.147204
```

Figure 4.3.6: Area Report

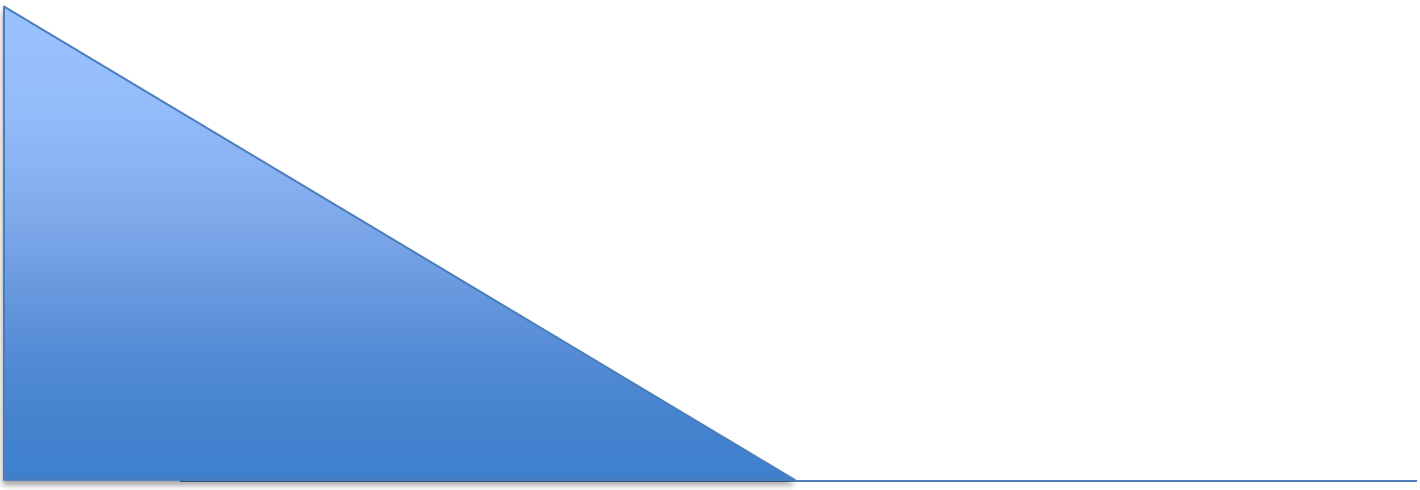
area.rpt | power.rpt | syn_script.tcl | syn.log

```
****************************************

Library(s) Used:

    scmetro_tsmc_cl013g_rvt_ss_1p08v_125c (File: /home/IC/Labs/AES/std_cells/scmetro_tsmc_cl013g_rvt_ss_1p08v_125c.db)


Operating Conditions: scmetro_tsmc_cl013g_rvt_ss_1p08v_125c   Library: scmetro_tsmc_cl013g_rvt_ss_1p08v_125c
Wire Load Model Mode: top

Design        Wire Load Model            Library
------------------------------------------------
AES_TOP               tsmc13_wl10       scmetro_tsmc_cl013g_rvt_ss_1p08v_125c


Global Operating Voltage = 1.08
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1pW


--------------------------------------------------------------------------------
                                   Switch   Int      Leak     Total
Hierarchy                          Power    Power    Power    Power    %
--------------------------------------------------------------------------------
AES_TOP                             1.123    0.271 8.37e+07    1.478 100.0
  U0_KEY_UNIT (key_unit)         2.52e-02    0.213 4.89e+07    0.287  19.4
    sbox4 (sbox_1)               2.80e-03 5.76e-04 2.98e+06 6.35e-03   0.4
    sbox3 (sbox_2)               2.81e-03 5.76e-04 2.98e+06 6.37e-03   0.4
```

Figure 4.3.5: Power Report

Chapter 5: Conclusion and Future Work

- Conclusion
- Future work



Integrating RNNs as decoder in Zigbee receivers can significantly enhance the performance of wireless communication systems by providing robust and adaptive signal processing capabilities. RNNs, with their ability to model temporal dependencies, are well-suited for handling the complex and dynamic nature of wireless signals. This can lead to improved accuracy in demodulation, especially in environments with high noise or interference, Additionally, implementing AES encryption in Zigbee networks ensures a high level of security for data transmission. AES is a widely recognized encryption standard known for its strength and efficiency. By securing communication with AES, Zigbee networks can protect against unauthorized access and ensure data integrity and confidentiality.

Benefits

1. **Improved Demodulation Performance**: RNNs can adapt to varying signal conditions, potentially outperforming traditional demodulation techniques.
2. **Enhanced Security**: AES provides robust encryption, safeguarding data against eavesdropping and tampering.
3. **Resilience to Noise and Interference**: RNNs can better handle the variability and unpredictability of real-world wireless channels.
4. **Adaptability**: RNNs can be trained to optimize performance for specific environments and conditions.

Challenges

1. **Computational Complexity**: RNNs require significant computational resources for training and real-time processing, which could be a limitation for resource-constrained Zigbee devices.
2. **Latency**: The processing delay introduced by RNNs might impact real-time communication performance.
3. **Implementation Complexity**: Integrating RNNs and AES into existing Zigbee frameworks requires sophisticated software and hardware design.
4. **Energy Consumption**: Both RNN processing and AES encryption can increase the energy consumption of Zigbee devices, affecting battery life.

5.2 Future work

- Using neural network to estimate frequency offset
- **Hybrid Models:** Explore hybrid models that combine RNNs with other machine learning techniques to further improve demodulation performance and energy efficiency.
- **Cross-Layer Security:** Develop cross-layer security approaches that integrate AES encryption with other security measures at different layers of the Zigbee protocol stack.
- **Reducing number of multipliers and adders using pipeline**

References

1_**[BOOK] Zigbee** wireless networking

2_ https://ieeexplore.ieee.org/abstract/document/5942102/

3_**Zigbee**: A low power wireless technology for industrial **applications**

4_ Ball Grid Array (BGA) Package, Intel® Packaging Databook, Chapter 14, Available at http://www.intel.com

5_ https://link.springer.com/content/pdf/10.1007/3-540-37366-7_16.pdf

6_ Xiuping Zhang; Guangjie Han; Changping Zhu; Yan Dou; Jianfeng Tao;" Research of Wireless Sensor Networks based on ZigBee for Miner Position", [J] International Symposium on Computer, Communication, Control and Automation, IEEE. 29 July 2010Pg1 - 5

7_ Freescale Semiconductor, available at www.freescale.com/zigbee

8_H. Chen, C. Wang, J. Li, C. Wang and G. Xu, "Application of OnLine Monitoring Technologies for UHV AC Transmission Lines," Power System Technology, vol. 33, May 2009, pp. 55-5

9_ https://cdn.techscience.cn/files/iasc/2023/TSP_IASC-36-3/TSP_IASC_33243/TSP_IASC_33243.pdf

10_ https://ieeexplore.ieee.org/abstract/document/6977558/

11 Oh, Nam-Jin, and Sang-Gug Lee. "Building a 2.4-GHz radio transceiver using IEEE 802.15. 4." *IEEE Circuits and Devices Magazine* 21.6 (2006): 43-51.

12_ Wang CC, Huang CC, Huang JM, Chang CY, Li CP. ZigBee 868/915-MHz modulator/demodulator for wireless personal area network. IEEE Transactions on very large scale integration (VLSI) systems. 2008 Jun 27;16(7):936-9.

13_ Wang CC, Huang JM, Lee LH, Wang SH, Li CP. A low-power 2.45 GHz ZigBee transceiver for wearable personal medical devices in WPAN. In2007 Digest of Technical Papers International Conference on Consumer Electronics 2007 Jan 10 (pp. 1-2). IEEE.

14_ Ting Ting Meng, Chen Zhang, Peter Athanas, "An FPGA-Based Zigbee Receiver on the Harris Software Defined Radio SIP", SDR Forum Technical Conference, Denver, Colorado, 2007.

15_Rafidah Ahmad, Othman Sidek, Wan Md. Hafizi Wan Hassin, Shukri Korakkottil Kunhi Mohd, and Abdullah Sanusi Husain, "Verilog-Based design and implementation of Design Tansmitter for Zigbee Applications", International Journal of Emerging Sciences, 723-724, December 2011, ISSN: 2222- 4254, © IJES.

16- Deep, V., & Elarabi, T. (2017, May). Efficient IEEE 802.15. 4 ZigBee standard hardware design for IoT applications. In *2017 International Conference on Signals and Systems (ICSigSys)* (pp. 261-265). IEEE.

17-Chaudhary, A., Rusia, J., Gourav, K., Tripathi, P., Pandey, J., Majumdar, S., ... & Verma, S. (2017, February). Design and simulation of physical layer blocks of ZigBee transmitter. In *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC)* (pp. 347-351). IEEE.

18- Supare, V., & Dhadwe, M. (2015). Design & Implementation of MQAM based zigbee Transceiver using HDL. *International Journal of Engineering Research and General Science*, *3*(1).

19- Ahmad, R., Sidek, O., & Mohd, S. K. K. (2014). Implementation of a Verilog-based digital receiver for 2.4 GHZ Zigbee applications on FPGA. *J. Eng. Sci. Technol*, *9*, 135-152.

20- Adaptive Packet Sizing for OTAP of PsoC Based Interface BoardinWSN_ICM2010.

21-Abbas, K. (2022). *From Algorithms to Hardware Architectures: Using Digital Radios as a Design Example*. Springer Nature.

22- Advanced Encryption Standard (AES) | NIST

23- A highly regular and scalable AES hardware architecture | IEEE Journals & Magazine | IEEE Xplore