Amirkabir University of Technology

(Tehran Polytechnic)

Department of Computer Engineering

Bachelor Thesis

Implementation and Comparison of Two Communication Types Between FPGAs Using Convolutional Neural Networks

By
Amir Behnam

Supervisor

Dr. Hamid Reza Zarandi

September 2024

بسم الله الرحمن الرحیم

To the One
from whom alone grace may be hoped...

Acknowledgments

I would like to sincerely express my gratitude to my esteemed supervisor, Dr. Hamidreza Zarandi, for his invaluable guidance and dedicated support throughout this project. Without his insightful advice and unwavering encouragement, the completion of this research and the preparation of this thesis would not have been possible.I also extend my deepest thanks to the distinguished Dr. Morteza Sahebzamani for graciously serving as the examiner of this thesis.

Abstract

In recent years, convolutional neural networks (CNNs) have been introduced as one of the most powerful and effective tools in various fields of deep learning. With their use of complex and flexible layer structures, these networks have been able to provide highly accurate and effective results in tasks such as image recognition, medical data analysis, natural language processing, and even automatic pattern recognition in large datasets. The growing development of these networks' applications has increased the demand for fast and efficient processing. Therefore, the use of FPGA boards as a suitable platform for implementing them has gained significant attention due to their high parallelization capabilities and energy efficiency. In this project, two communication methods between FPGAs using convolutional neural networks were implemented and studied. Both wired and wireless modes were used for the communication implementation, and the UART protocol was employed. The results obtained show that the wired communication method significantly outperforms the wireless method. These findings highlight the importance of selecting the appropriate communication method to enhance the performance and speed of convolutional neural networks and can pave the way for the development of optimized applications in various fields.

Keywords:

Convolutional Neural Networks, Deep Learning, Communication Methods, FPGA, UART Protocol

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

In this section, the introduction of the thesis is presented. First, the problem is described; then, the necessity, objectives, and motivation of the project are stated. Finally, the structure of the report and the upcoming chapters are explained.

1-1- Problem Statement

In today's world, with the rapid advancement of technology and the growing need for faster and more efficient data processing, the use of digital integrated circuits such as FPGAs has become increasingly widespread. FPGAs are used in many applications—such as image processing, machine learning, artificial neural networks, and embedded systems—due to their high flexibility and reconfigurability. However, to make optimal use of the processing power of FPGAs, establishing connections and communication between multiple FPGAs is essential.

One of the key issues in this area is the type and method of connecting two FPGAs, which can significantly impact system performance. Various types of connections can be used between FPGAs, and choosing the appropriate type of connection can directly affect overall system efficiency.

In this thesis, we implement two different types of connections between two FPGAs using a convolutional neural network. For this purpose, two distinct connection methods are implemented between FPGAs with the aid of a convolutional neural network, and their performance is evaluated. The results of this study can be helpful in selecting the appropriate connection method for similar applications and in improving the efficiency of FPGA-based systems.

1-2- Significance and Objectives of the Project

Given the significant importance of FPGAs in implementing convolutional neural networks and the critical role of communication between them in the system's overall performance, the necessity of this project becomes evident. This research aims to examine the impact of two different types of connections on the capabilities of FPGAs and to analyze the performance of these connections from various perspectives. A detailed assessment of how these connections influence processing speed and other key performance aspects—essential for real-time applications—constitutes a major part of this project.

The results of this study are expected to contribute significantly to enhancing the capabilities of FPGA-based systems and facilitating their integration with existing technologies. Moreover, these findings can pave the way for improved future designs, particularly in fields that demand faster and more efficient processing. In addition, this research can lead to a deeper understanding of the characteristics and limitations of each connection type, ultimately contributing to the development of more intelligent and flexible systems for complex and advanced applications.

1-3- Thesis Structure

In the first chapter of this report, an introduction to the project's main subject was presented. Chapter Two covers the fundamental theoretical concepts, including the architecture of convolutional neural networks, the UART communication protocol, and the tools and boards used. Chapter Three is dedicated to the implementation of the module structures and the method of interaction and connection between the project's components, along with detailed explanations. In Chapter Four, the implementation results of both modes are thoroughly presented and compared. Chapter Five, which is the final chapter, analyzes the outcomes of the project and concludes the thesis by offering suggestions for future work.

Chapter 2: Fundamental Concepts

2-1- Introduction to Neural Networks

A neural network is a computational model inspired by the way biological neural networks in the human brain process information. The development of neural networks dates back to the early 20th century. Artificial neural network models are based on the functioning of neurons and are inspired by biological neural networks. These networks consist of interconnected layers of nodes or neurons, each performing a specific mathematical operation.

The basic structure of a neural network includes an input layer, one or more hidden layers, and an output layer. Each neuron receives a number of input signals from other neurons, multiplies these inputs by weights to simulate synaptic interactions, sums the weighted inputs, and combines them with a bias value (usually one). This result is then passed through a nonlinear activation function, which generates the neuron's output signal.

The network learns by adjusting the weights of these connections through a process called training, which is typically performed using the backpropagation method [1]. During training, the network is exposed to a large dataset, and the weights are adjusted to minimize the difference between predicted and actual outputs. An overview of a neural network is shown in Figure 2-1 [2].

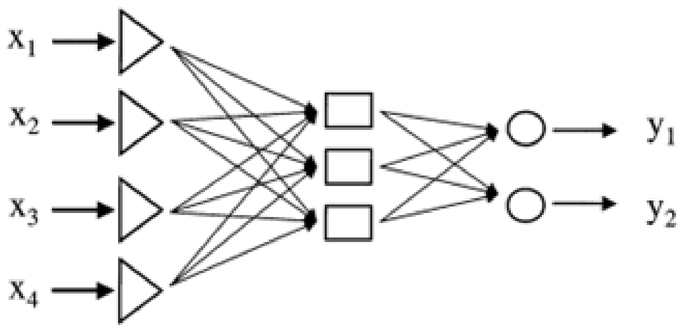

Figure 2-1. Neural Network Architecture [2]

Neural networks are powerful tools for a wide range of applications, including image and speech recognition, natural language processing, and autonomous systems. Their ability to learn and generalize from large datasets makes them especially effective for complex tasks that are difficult to program explicitly. Deep learning, a subset of machine learning that involves neural networks with many hidden layers, has led to significant advances in artificial intelligence. These deep neural networks can automatically extract hierarchical features from raw data, enabling the development of models that achieve high performance across various domains [3].

As research in the field of neural networks continues to evolve, these networks are expected to play an increasingly vital role in advancing technology and solving real-world problems.

2-2- Convolutional Neural Network (CNN)

Convolutional neural networks (CNNs) are a specialized type of artificial neural network primarily designed for processing structured grid-like data, such as images. These networks are highly effective in tasks like image classification, object detection, and segmentation. They consist of multiple layers, including convolutional layers, pooling layers, and fully connected layers. Convolutional layers apply filters to the input data to extract features, while pooling layers reduce the dimensionality of the data, making the network more efficient and less prone to overfitting. This hierarchical structure allows the network to learn and recognize patterns at different levels of abstraction—from simple edges to complex objects [3]. The structure of a convolutional neural network is illustrated in Figure 2-2 [4].

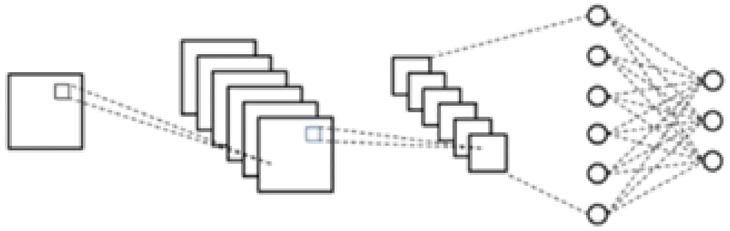

Figure 2-2. Convolutional Neural Network Architecture [4]

One of the key advantages of convolutional neural networks (CNNs) is their ability to automatically learn spatial hierarchies of features from input images, making them highly effective for computer vision tasks. These networks are inspired by the human visual system, where neurons in the visual cortex respond to specific regions of the visual field. This local connectivity is mirrored in CNNs, where each neuron is connected to a small region of the input data. Moreover, CNNs exhibit translation invariance, meaning they can recognize objects regardless of their position within the image. This property, combined with their ability to process large volumes of data, makes them powerful tools for various applications, including medical image analysis, autonomous driving, and facial recognition.

2-2-1- Convolutional Layer

The convolutional layer in a convolutional neural network is designed to automatically and adaptively learn spatial hierarchies of features from input images. The main operation in a convolutional layer is the convolution operation, in which a small matrix called a filter moves across the input image, and the dot product between the filter elements and the corresponding regions of the input image is computed.

This process results in the creation of a feature map, which represents the presence of learned features—such as edges, textures, or more complex patterns—at various locations in the input image [5]. An overview of the computations in this layer is shown in Figure 2-3 [6].

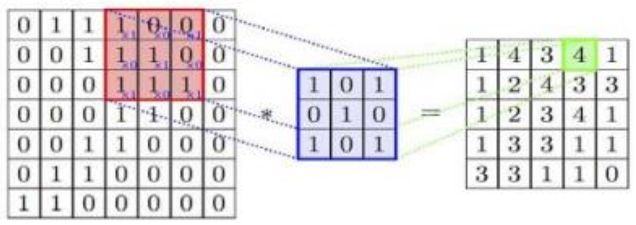

Figure 2-3. Convolutional Layer Operations [6]

Convolutional layers typically use multiple filters to learn different features, resulting in the production of multiple feature maps. These maps are then passed through activation functions to introduce non-linearity. Including a bias helps the model better approximate underlying patterns by allowing the activations to shift, thereby enhancing the model's ability to learn from data. This combination of convolution operations, bias addition, and activation function use constitutes the core mechanism through which convolutional neural networks identify and learn hierarchical features from images—an essential process for tasks such as image classification, object detection, and segmentation [5].

The convolutional layer is the first layer capable of extracting features from images. Convolution operations allow us to preserve the relationship between different parts of an image, since pixels are related to their nearby neighbors. For example, applying a small filter with a regular stride over an image results in a smaller-sized image. This operation reduces the image dimensions without eliminating the spatial relationships between pixels [7].

2-2-1-1- Stride

Stride in convolutional neural networks refers to the size of the filter's movement across the input image. When the stride is equal to one, the filter moves one pixel at a time, resulting in a feature map with dimensions similar to the original input. Larger strides, such as two or more, cause the filter to move several pixels at once, producing a smaller output feature map. Stride controls the downsampling rate and affects both the resolution and computational efficiency of the network. By adjusting the stride, a convolutional neural network can balance between preserving fine spatial details and reducing the computational load and data dimensions [3].

2-2-1-2- Padding

Padding in convolutional neural networks refers to the technique of adding extra pixels (usually zeros) around the borders of an input image before applying the convolution operation. The main goal of padding is to preserve the spatial dimensions of the input volume as it passes through the convolutional layers [3]. Without padding, the spatial dimensions of the feature maps shrink with each convolutional layer because the kernel cannot fully cover the edges and corners of the input image. By adding padding, the kernel can convolve over the entire input image, ensuring that the output feature map maintains the same spatial dimensions as the input or at least reduces the size reduction caused by the convolution operation [5]. Common padding types include "same padding," which adds enough zeros to preserve the output size when using a stride of one, and "valid padding," which applies no padding, resulting in a smaller output size. This operation is crucial for preserving spatial information, reducing edge effects, and enabling efficient feature learning across the image. An overview of the padding process can be seen in Figure 2-4 [8].

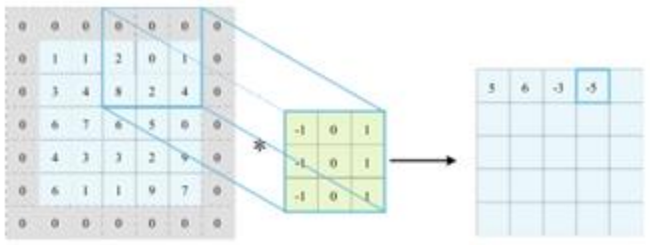

Figure 2-4. Padding [8]

2-2-1-3- Activation Function

The activation function in convolutional neural networks introduces nonlinearity to the model, allowing it to learn and represent complex patterns in the data. After each convolution operation, the activation function is applied to the resulting feature map, transforming the input signal. The Rectified Linear Unit (ReLU) is one of the most common activation functions, which sets all negative values to zero and leaves positive values unchanged, making training faster and more efficient. This helps mitigate issues like vanishing gradients, which can disrupt the learning process in deep networks. Other activation functions such as sigmoid and hyperbolic tangent (tanh) are used in specific contexts, especially for output layers where limited or probabilistic outputs are required. For example, the sigmoid function maps input values to a range between zero and one, making it suitable for binary classification tasks. By introducing nonlinearity, activation functions enable convolutional neural networks to approximate a wide range of functions and capture complex data structures, which are essential for tasks such as image classification, object detection, and segmentation. This non-linear transformation allows convolutional neural networks to model complex relationships and interactions in the data, providing the flexibility and power needed for various computer vision applications. An overview of activation functions can be seen in Figure 5-2 [9].

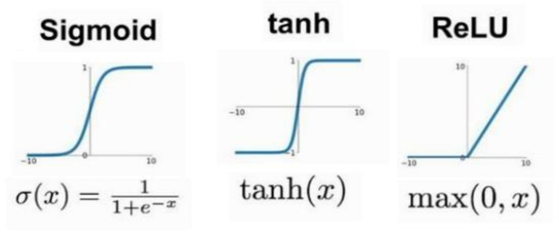

Figure 2-5. Types of Activation Functions [9]

2-2-2- Pooling Layer

In convolutional neural networks, pooling layers are essential for reducing the spatial dimensions (width and height) of the input feature maps while preserving their depth. This reduction helps decrease computational load, reduce the number of parameters, and control overfitting [3]. Two common types of pooling are average pooling and max pooling.

Average pooling involves taking the average of all values in a specific subregion of the input feature map. For example, using a defined window and a certain stride, the average pooling operation calculates the average of each block of pixels and creates a single pixel in the output feature map that represents this average. This process smooths the input by summarizing information in each section and reduces noise. This method is particularly useful in scenarios where the precise location of features is not as important as their general presence in a region. However, average pooling can sometimes dilute important features by averaging all values together [5]. An overview of the average pooling process can be seen in Figure 2-6 [10].

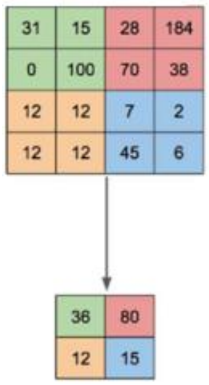

Figure 2-6. Average Pooling [10]

On the other hand, max pooling selects the highest value from each subregion. Using a similar window and stride, max pooling picks the highest value from each block. This method effectively captures the most prominent features within each window, preserving the strongest activations while eliminating less important information. This characteristic can be useful for emphasizing critical features and ensuring that the most essential aspects of the input data are retained by the network. Max pooling performs well in practice as it preserves the most prominent features, which are often relevant for tasks like object detection and classification [5]. An overview of the max pooling process can be seen in Figure 2-7 [11].

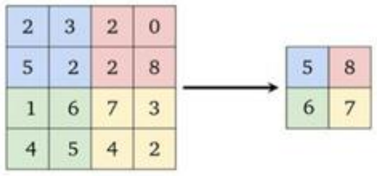

Figure 2-7. Max Pooling [11]

2-2-3- Flattening Operation

Flattening operation refers to the transformation of a two-dimensional matrix of pixel values into a one-dimensional vector. This step occurs after the pooling layer to prepare the data for fully connected layers, which perform classification or regression tasks. The operation takes the multi-dimensional output from the convolutional and pooling layers and converts it into a long vector. This process simplifies the data structure so that it can be fed into fully connected layers that expect one-dimensional input [5]. An overview of the flattening operation can be seen in Figure 2-8 [6].

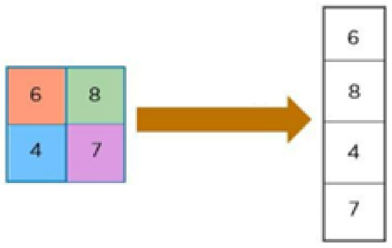

Figure 2-8. Flattening Operation [6]

2-2-4- Fully Connected Layer

A fully connected layer is an essential component in convolutional neural networks, typically used in the final stages of the network. This layer is designed to integrate and combine the features extracted by the convolutional and pooling layers to make the final decision or prediction. In this layer, each neuron is connected to all neurons in the previous layer, which allows the model to consider all learned features from the previous layers [3].

The main operation in the fully connected layer is matrix multiplication between the input vector (which can be considered as a mapping of the features from the previous layers) and the corresponding weights of the layer. After this multiplication, a bias is added to each neuron, and the result is passed through an activation function (such as the ReLU function) [5]. The final output of this layer is a vector, where the values represent the probabilities of different classes (in classification tasks) or the predicted values (in regression tasks). An overview of the fully connected layer operation can be seen in Figure 2-9 [12].

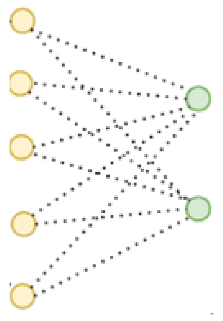

Figure 2-9. Fully Connected Layer Operation [12]

Fully connected layers, due to their large number of parameters and connections, often contribute a significant amount of computational complexity and learning capacity to the network. These layers play a crucial role in mapping the low- and mid-level features extracted by the convolutional layers to a higher-dimensional output space, making them essential for tasks such as image classification and object detection. However, due to the large number of parameters, these layers require fine-tuning to avoid overfitting the model.

2-3- UART Protocol

UART is a hardware communication protocol used for asynchronous serial communication between devices. Unlike synchronous communications, it does not require a shared clock signal between the transmitting and receiving devices. Instead, it relies on transmitting data packets that include start bits, data bits, parity bits, and stop bits to ensure accurate data transmission. The start bit signals the beginning of a data packet, followed by data bits (usually 7, 8, or 9 bits), an optional parity bit for error checking, and one or more stop bits to indicate the end of the packet [13]. This structure effectively manages data transmission even in the absence of a synchronized clock, making it suitable for various applications such as microcontrollers, Bluetooth modules, and serial ports. The configuration of the UART data structure is shown in Figure 2-10 [13].

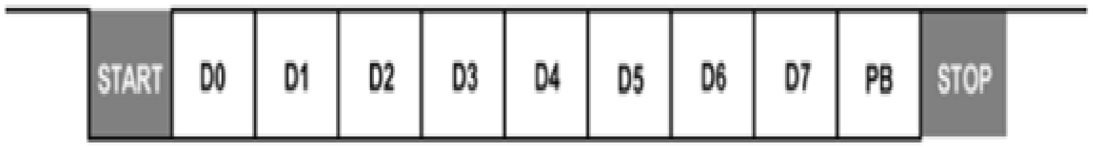

Figure 2-10. UART Data Frame Structure [13]

The communication process begins with the transmission of a start bit by the sender (TX) to alert the receiver (RX) about the incoming data packet. The start bit pulls the transmission line (which is typically held high) low for one bit, signaling the RX device that a new data frame is beginning. The data bits are then sent sequentially, with the least significant bit transmitted first. A parity bit can be used to detect errors in the transmitted data by ensuring the total number of ones is either even or odd, depending on the chosen parity mode. Finally, one or more stop bits indicate the end of the packet, allowing the receiver to prepare for the next transmission [14]. The receiver reads the incoming bits at a preset baud rate, which must match the sender's rate for successful communication. The absence of a shared clock simplifies the wiring requirements and allows for flexible data transmission speeds, but precise timing is essential to avoid data loss or corruption. The interaction between the sender and receiver in the UART protocol is shown in Figure 2-11 [15].

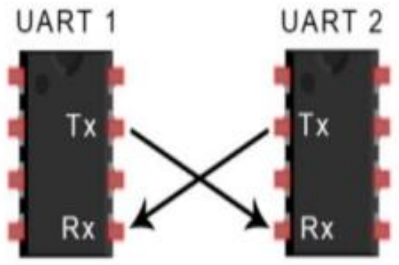

Figure 2-11. Interaction Between UART Transmitter and Receiver [15]

In practical applications, this communication interface is typically found in embedded systems, where simplicity and ease of implementation are critical. Microcontrollers often include internal modules to facilitate communication with peripherals such as sensors, displays, and communication modules. Additionally, it is widely used in computer serial ports, providing a simple method for connecting external devices like modems and serial consoles. Despite its relatively low data transmission speed compared to other communication protocols, the robustness and ease of use of this protocol make it a valuable tool for various serial communication needs in consumer and industrial electronics.

2-4- Experimental Setup and Boards

For implementing the project, boards and tools were used, each of which is briefly explained in this section.

- Two AVA3S400 boards from the Spartan-3 family

The AVA3S400 board is built using the XS3C400 chip from Xilinx. The primary purpose of this board is to utilize the features of this chip for implementing digital and processing algorithms through HDL programming languages.

Table (2-1) describes the features and resources of the Spartan3 AVA3S400.

Table 2-1. Specifications of the Spartan3 AVA3S400 Board

Quantity / Value	Description
4 Mb	Memory
1	Number of Alarm Signals
56000	Distributed RAM Bits
16	Number of Multi-function Switches
16	Dedicated Multipliers
4	Digital Clock Manager (DCM)
400000	System Gates
264	Maximum User I/O Pins
16	Number of LEDs
896	Number of Configurable Logic Blocks (CLBs)
480 Mb/s	Maximum Operating Frequency
288000	Block RAM Bits
4	Push-button Input
4	Number of 7-Segment Displays

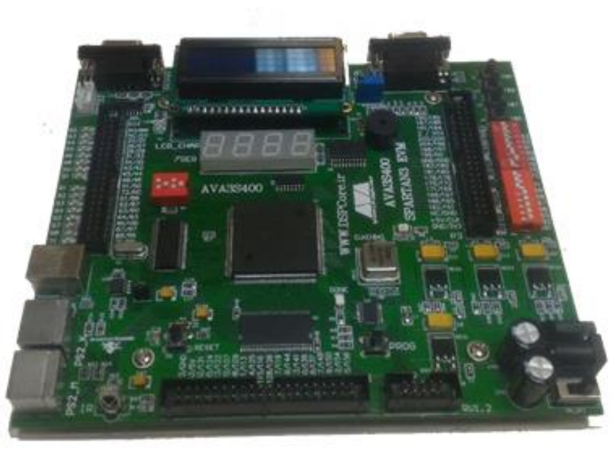

Figure 2-12. Spartan3 AVA3S400 Board

- Two ESP32-WROOM-32 modules

This powerful Wi-Fi and Bluetooth module, developed by Espressif Systems, is widely recognized for its capabilities in IoT projects and wireless communications. Below is a detailed overview of its features:

1) Wireless Communication: This module is strong in terms of connectivity and supports both Wi-Fi and Bluetooth. It complies with the Wi-Fi 802.11 b/g/n standards, enabling easy network access, and also supports Bluetooth 4.2 and Bluetooth Low Energy for seamless communication with various devices.

2) Power Management: Designed with energy efficiency in mind, the module offers advanced power management features. Its built-in voltage regulators operate within a range of 2.7V to 3.6V, ensuring stable performance and supporting various low-power modes, making it ideal for battery-powered projects.

3) Memory: The module is equipped with 4MB of flash memory for storing the operating system, program code, and other data. It also includes 448KB of ROM and 16KB of RTC SRAM, providing sufficient storage space for complex applications.

4) Security Features: Prioritizing security, the module includes features such as secure boot, flash encryption, and hardware-accelerated cryptography, making it suitable for applications where data protection is critical.

5) Microcontroller: At its core lies a powerful dual-core processor capable of operating at 240 MHz. With 520KB of SRAM and a wide range of peripherals, the microcontroller is suitable for demanding tasks and real-time applications.

6) Ports and I/O: The module offers extensive I/O capabilities, including 36 GPIO pins configurable as digital input or output. These pins support various functionalities such as PWM, ADC, DAC, and touch sensors, with 12-bit ADCs and 8-bit DACs enabling precise analog input and output.

7) Applications: Widely used in IoT projects, home automation, wearable electronics, and sensor networks, the module’s powerful processing and robust communication features make it an ideal solution for developers aiming to create smart and connected devices.

8) Communication Protocols: The module supports various communication protocols, with significant focus on UART. This flexibility enables direct integration with external sensors, peripherals, and other devices, making it compatible with a wide range of applications.

9) Programming and Software Support: The module is highly programmable and compatible with development environments such as Arduino IDE, MicroPython, and Espressif's ESP-IDF. It can be programmed in languages like C/C++, Python, and Lua, with extensive libraries and APIs available for digital and analog I/O, Wi-Fi, Bluetooth, and other peripherals.

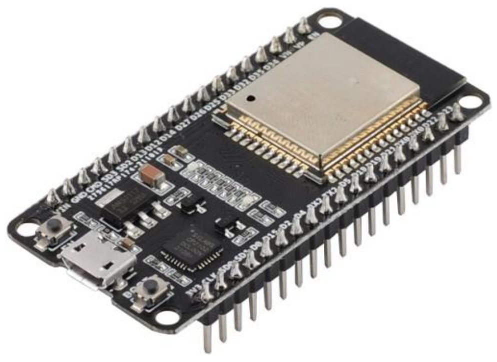

Figure 2-13. ESP32-WROOM-32 Module

2-5- Tools

- ISE is a software suite developed by Xilinx for designing and programming FPGA devices. It provides a comprehensive set of tools and features to support the entire FPGA design flow—from initial design entry to bitstream generation.

ISE consists of various modules and components, some of which include:

1) Project Navigator: The main tool for managing and organizing FPGA projects, allowing users to create and manage design files, run simulations, and perform synthesis and implementation.

2) Initial Design Environment: Provides an environment for early-stage design, enabling users to develop their designs from the ground up and use initial simulations and analyses to improve design quality.

3) Advanced Implementation Tools: Includes tools for mapping the design onto the FPGA, optimizing placement and routing, and generating the final bitstream.

4) Built-in Simulator: Enables simulation of digital designs written in hardware description languages, offering waveform viewing, interactive debugging, and testbench creation.

5) Real-Time Debug and Analysis: Allows for examining the behavior of digital circuits during execution and identifying issues instantly. This feature improves development speed and reduces errors.

6) Power Analysis Tool: Enables analysis and optimization of FPGA power consumption, particularly useful for power-intensive projects.

7) Comprehensive Reporting: Generates detailed reports on various design and implementation stages, including resource usage, timing, and optimizations, helping in evaluating and improving the design.

8) Resource Optimization Tool: Helps optimize the use of FPGA resources like LUTs, registers, and RAM blocks, enhancing design efficiency and performance.

9) XDL Design Language: XDL is an internal format used to represent the design hierarchy and interconnections of digital circuits. It plays a crucial role in implementation and bitstream generation.

10) Modular Design Support: ISE supports modular design, allowing users to break down complex projects into smaller modules, each of which can be developed and tested independently. This improves design structure and maintainability.

- The Arduino IDE is an open-source software platform created for programming Arduino boards. With a user-friendly interface and a variety of tools, it simplifies the process of writing, compiling, and uploading code to Arduino microcontrollers. Key features of the Arduino IDE include:

1) Multi-Platform Support: The IDE is available for various operating systems including Windows, macOS, and Linux, enabling users from different platforms to utilize it.

2) Integrated Editor: It includes an editor that allows users to write Arduino code in C++. Features such as syntax highlighting and auto-completion simplify and accelerate the coding process.

3) Library Manager: The IDE provides a library management tool that allows users to easily search for, install, and manage libraries. Libraries consist of ready-made

code that adds extra functionality to projects and simplifies interaction with sensors, actuators, and communication protocols.

4) One-Click Upload: The IDE automatically converts code into machine-readable instructions. With a single button press, the compiled code can be uploaded directly to the Arduino board.

5) Serial Monitor: This built-in tool enables communication with the Arduino board via the serial interface. It displays outputs and data sent from the Arduino, making it extremely useful for debugging and monitoring code performance.

6) Internal Debugger: One of the features of the Arduino IDE is the built-in debugger, which allows users to step through their code and identify and fix issues during development.

7) Board Manager: This tool allows installation and configuration of board support packages. It helps users select the correct board model for their project and install the required drivers.

8) Plugin Support: The IDE supports various plugins that users can install to extend its capabilities. These may include additional tools, new themes, or advanced features for the development environment.

2-6- Summary

In this section, the architecture of the convolutional neural network and its computational methods were comprehensively reviewed. Then, the architecture of the UART protocol and its components were explained. Finally, the tools and execution environment used in this research were thoroughly evaluated.

Chapter 3: Design and Implementation

In this section, the implementation method, module structures, and the complete execution of the project will be thoroughly explained.

3-1- Overall System Architecture

The overall process of the project is as follows: initially, the first FPGA receives the input data and performs calculations, generating intermediate results. For further computations, these intermediate results are sent to the second FPGA. This transfer enables the second FPGA to perform the final computations based on the progress made by the first FPGA. By distributing the workload between the two FPGAs, the system achieves parallel processing, resulting in more efficient execution. Finally, upon completing the calculations, the second FPGA generates the final output. This output represents the culmination of the entire convolutional neural network computations and serves as the expected result. For a better understanding of the project, the overall system for both wired and wireless modes is shown in Figures (3-1) and (3-2), respectively.

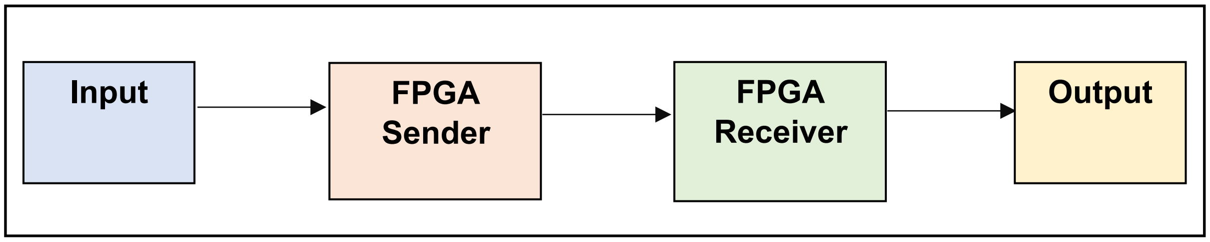

Figure 3-1. System Overview for Wired Mode

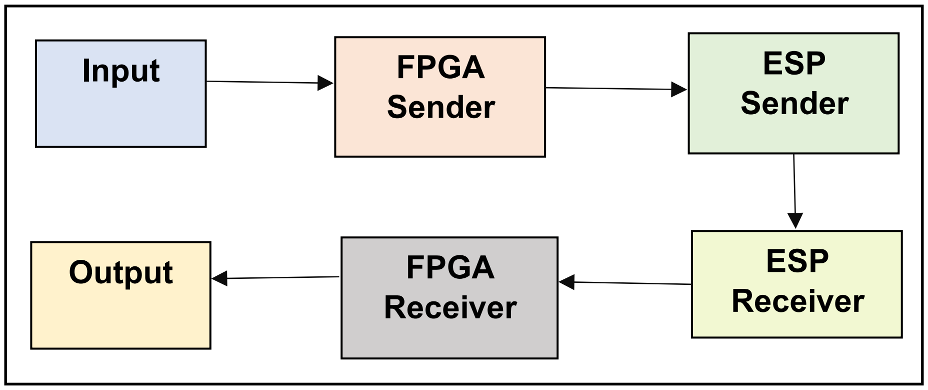

Figure 3-2. System Overview for Wireless Mode

Based on the explanations provided, we will now review the structure of the modules and then provide a detailed description of the system architecture for both wired and wireless modes.

3-2- Implemented Architecture Model for the Convolutional Neural Network

In this research, to implement the convolutional neural network, four modules are used, including the convolutional layer, the pooling layer, the flattening operation, and the fully connected layer. These modules operate hierarchically and sequentially, performing operations efficiently. In Figure (3-3), the structure and execution sequence of the implemented model are clearly shown, providing an overview of how this network functions.

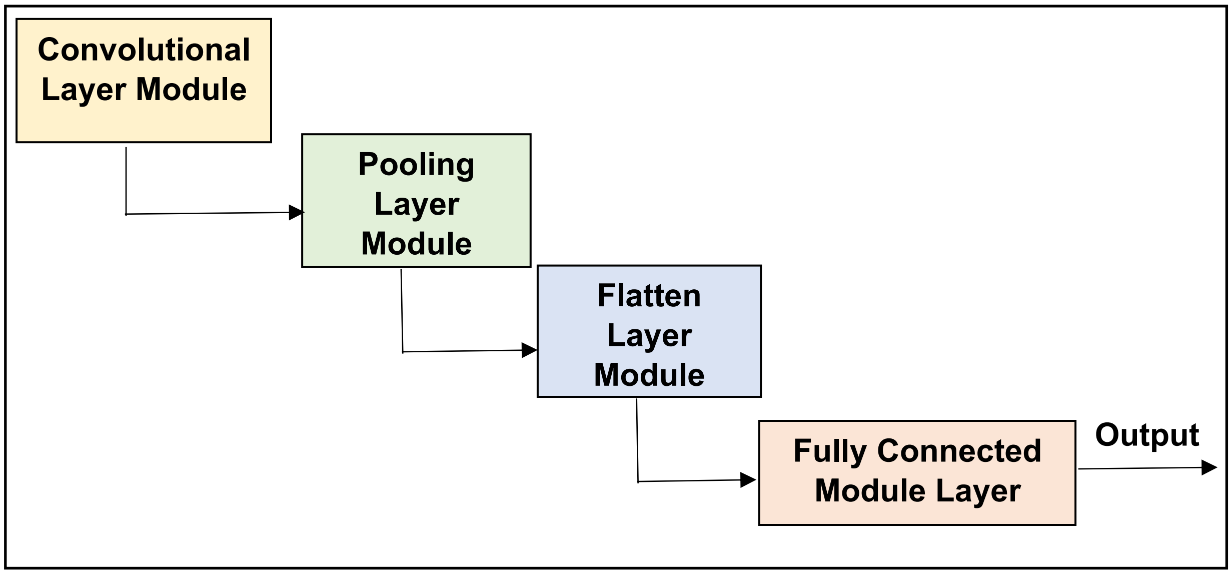

Figure 3-3. Implemented Model Architecture for the Convolutional Neural Network

Next, we will examine the structure of the convolutional neural network modules.

3-2-1- Convolutional Layer Module

Figure (3-4) shows the convolutional layer module. This module is designed to perform mathematical operations in which each filter moves over the input, multiplies the filter values with the corresponding input values, and then sums them to produce a single value in the output feature map. The pin details of the module are listed in Table (3-1).

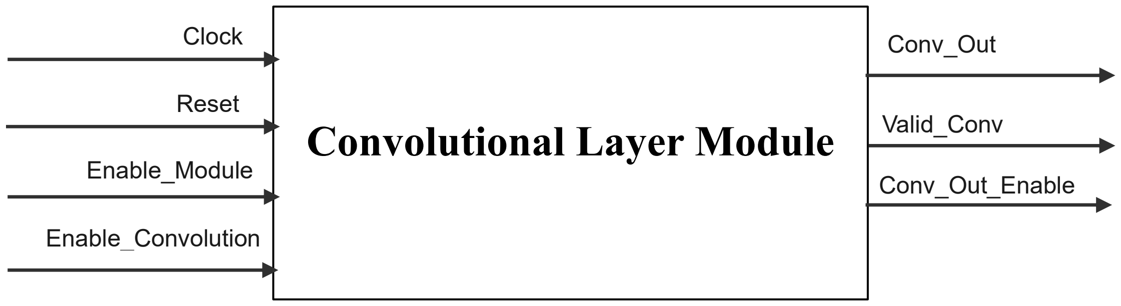

Figure 3-4. Convolutional Layer Module

Table 3-1. Description of Signals in the Convolutional Layer Module

Description	Signal Width	Signal Type	Signal
Input clock signal	1	Input	Clock
Module reset	1	Input	Reset
Activation of the convolution layer	1	Input	Enable_Convolution
Module activation	1	Input	Enable_Module
Indicates the validity of the output data	1	Output	Valid_Conv
Activation of the first layer's output stream	1	Output	Conv_Out_Enable
Output of the convolutional layer / This output is buffered and sent to the next layer 16 times for processing.	4	Output	Conv_Out

The complete process of this module is as follows: initially, the convolution operation starts with a sliding window that moves across the input image and collects pixel values in a two times two window. These values are then multiplied by the corresponding filter weights, and the results are summed in two stages. A bias is added to the final sum, and a ReLU activation function is applied, producing either a positive value or zero. The output is then stored in a buffer. This operation is repeated for each step on the image, with the results stored in the buffer after each step. The buffered outputs are sequentially sent to the next layer, ensuring that each processed output is correctly transferred when ready. This entire process is repeated 16 times, corresponding to the total number of two times two regions in the five times five image, and after processing all regions, the layer is reset for the next operation. It is worth mentioning that the input data for this module is hard-coded.

3-2-2- Pooling Layer Module

Figure (3-5) shows the pooling layer module. This module is designed to reduce the spatial dimensions of the input feature map. This is done by applying the maximum pooling operation, which involves selecting the maximum value from a fixed-size window. The details of the module's pins are provided in Table (3-2).

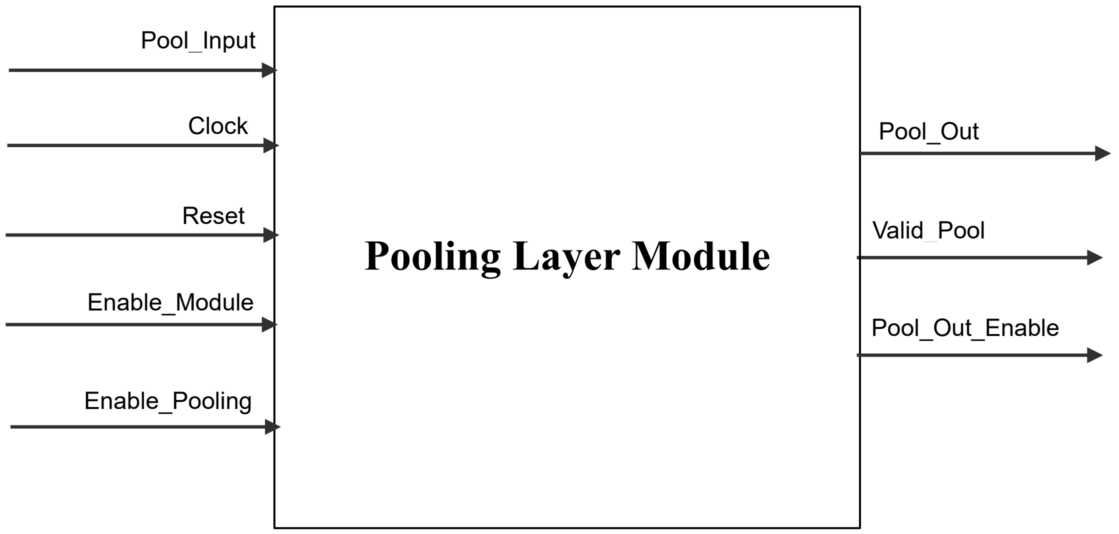

Figure 3-5. Pooling Layer Module

Table 3-2. Description of Signals in the Pooling Layer Module

Description	Signal Width	Signal Type	Signal
Input data of the pooling layer	4	Input	Pool_Input
Input clock signal	1	Input	Clock
Module reset	1	Input	Reset
Activation of the pooling layer	1	Input	Enable_Pooling
Module activation	1	Input	Enable_Module
Indicates the validity of the output data	1	Output	Valid_Pool
Activation of the second layer's output stream	1	Output	Pool_Out_Enable
Output of the pooling layer / This output is buffered and sent to the next layer 4 times for processing.	4	Output	Pool_Out

The complete process of this module is as follows: it sequentially loads the input data into a sliding window, which moves across the image based on the filter size and stride. The windowed data is stored in a FIFO buffer, and the maximum value within the window is calculated through comparison. This maximum value is then stored in an output buffer, which is transferred to the next layer. The data is transferred multiple times to the next layer, controlled by the stride, until all valid output values are generated. Control signals manage the window shifting, FIFO operations, and output generation, ensuring correct operation throughout the process through initialization and coordination.

3-2-3- Flattening Operation Module

Figure (3-6) shows the Flatten Layer module. This module is designed to convert a multi-dimensional feature map into a one-dimensional vector. This is done by reshaping the feature map, making it possible to feed it into fully connected layers for classification. The details of the module's pins are listed in Table (3-3).

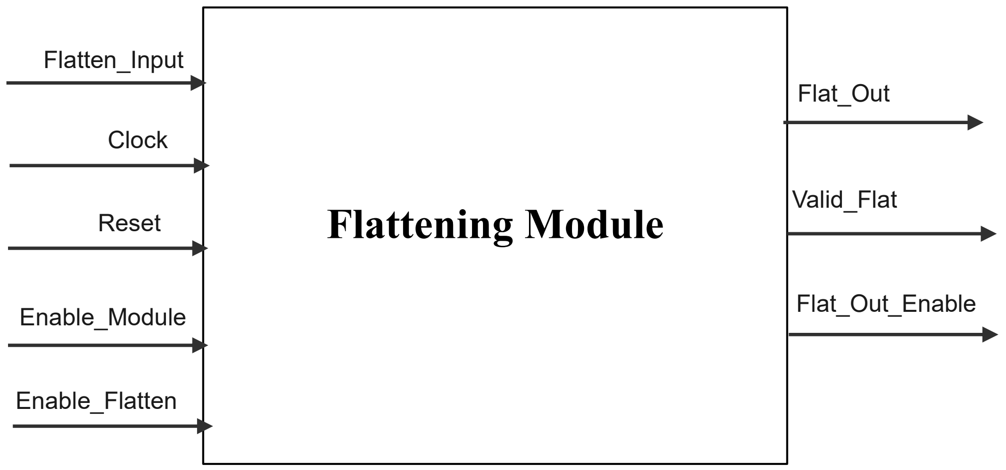

Figure 3-6. Flattening Module

Table 3-3. Description of Signals in the Flattening Module

Description	Signal Width	Signal Type	Signal
Input data of the flattening operation	4	Input	Flatten_Input
Input clock signal	1	Input	Clock
Module reset	1	Input	Reset
Activation of the flattening operation	1	Input	Enable_Flatten
Module activation	1	Input	Enable_Module
Indicates the validity of the output data	1	Output	Valid_Flat
Activation of the third layer's output stream	1	Output	Flat_Out_Enable
Output of the flattening operation / This output is buffered and sent to the next layer 4 times for processing.	4	Output	Flat_Out

The precise process of the Flatten operation module is as follows: This module converts a two-dimensional feature map into a one-dimensional vector by sequentially processing pixel values using a sliding window. This two times two window is continuously updated with input data and stores the values in four output buffers. When the buffers are filled and the stage conditions are met, the module sends the buffered data to the next layer. Control signals manage the timing of data transfer, ensuring that pixel values are transferred in a controlled and sequential manner. This process facilitates smooth data transfer through the network stages.

3-2-4- Fully Connected Layer Module

Figure (3-7) represents the fully connected layer module. This module is designed to compute the output by performing matrix multiplication between the flattened input vector and a weight matrix. Then, a bias is added to the sum of the values, and an activation function is applied to generate a set of output values. The module's pin details are listed in Table (3-4).

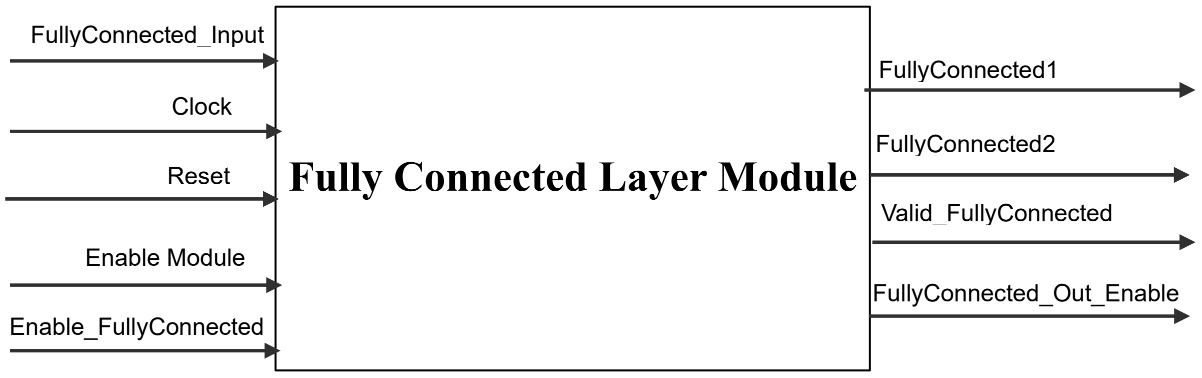

Figure 3-7. Fully Connected Layer Module

Table 3-4. Description of Signals in the Fully Connected Layer Module

Description	Signal Width	Signal Type	Signal
Input data of the fully connected layer	4	Input	FullyConnected_Input
Input clock signal	1	Input	Clock
Module reset	1	Input	Reset
Activation of the fully connected layer	1	Input	Enable_FullyConnected
Module activation	1	Input	Enable_Module
Indicates the validity of the output data	1	Output	Valid_FullyConnected
Activation of the fourth layer's output stream	1	Output	FullyConnected_Out_Enable
First output of the fully connected layer	4	Output	FullyConnected1
Second output of the fully connected layer	4	Output	FullyConnected2

The exact process of the fully connected layer module is as follows: The input data is first multiplied by predefined weights, and the multiplication results create average values. These results are then summed in two stages so that contributions from all weights are combined. The summed values are adjusted by adding the biases. Then, a ReLU activation function is applied to the bias-adjusted results, setting any negative values to zero. The activated results are stored in output buffers. Control signals manage each stage, ensuring proper sequencing and activation of the operations.

3-3- Implemented Architecture Model for the UART Protocol

In this research, the UART protocol has been used to implement communication between two FPGA devices. Some of the reasons for choosing this protocol are:

1) The protocol is very simple and practical.
2) It is cost-effective.
3) Communication between the two FPGAs requires only two channels, TX and RX.
4) This protocol is asynchronous, meaning each FPGA can send data independently.
5) It does not require complex hardware.
6) It is very suitable for point-to-point communication between FPGA devices.
7) Implementing this protocol requires minimal software and hardware resources.
8) It is energy-efficient.

The UART module is shown in Figure (3-8). In Table (3-5), the signals and pins of the UART module are described. The UART module consists of three sub-modules.

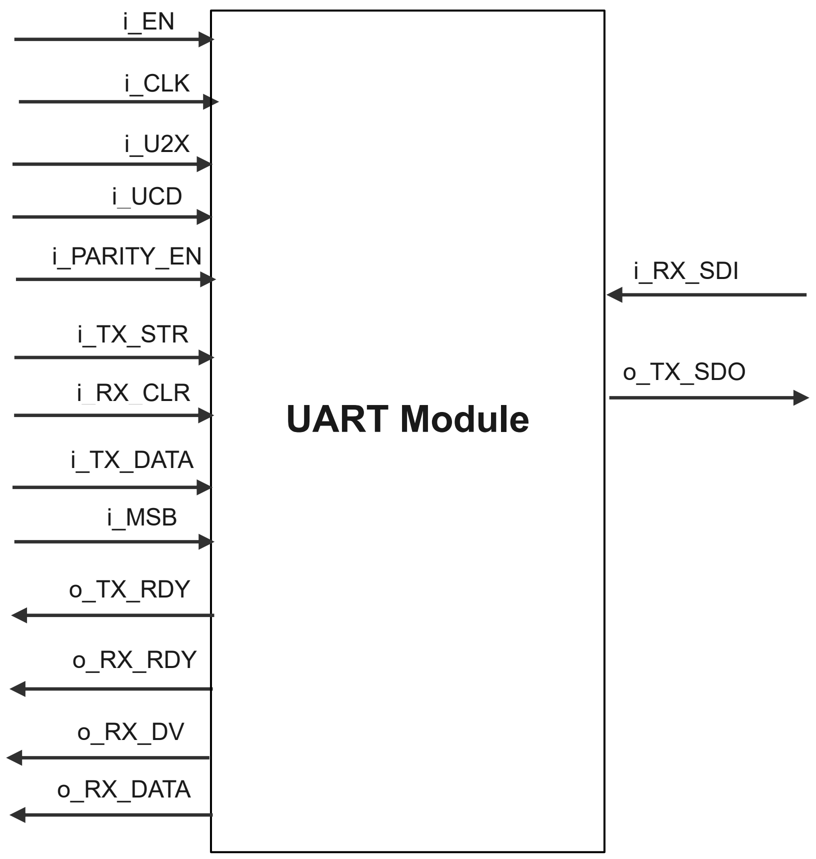

Figure 3-8. UART Module

Table 3-5. Description of Signals in the UART Module

Description	Signal Width	Signal Type	Signal
Input clock signal	1	Input	i_CLK
Parity bit enable	1	Input	i_PARITY_EN
Double the transmission speed	1	Input	i_U2X
Start data transmission	1	Input	i_TX_STR
Module activation	1	Input	i_EN
Set transmission speed	16	Input	i_UCD
Read received data	1	Input	i_RX_CLR
Transmitted data	8	Input	i_TX_DATA
Serial RX channel	1	Input	i_RX_SDI
Data order selector: MSB-first or LSB-first	1	Input	i_MSB
Received data	8	Output	o_RX_DATA
Serial TX channel	1	Output	o_TX_SDO
Module ready to transmit	1	Output	o_TX_RDY
Ready to receive new data	1	Output	o_RX_RDY
Indicates correctness or error in received data	1	Output	o_RX_DV

3-3-1- Clock Generation Logic Module

Figure (3-9) shows the clock generation module, which is designed to provide the appropriate clock for the transmission and reception sections of the UART module. The pin details of this module are provided in Table (3-6).

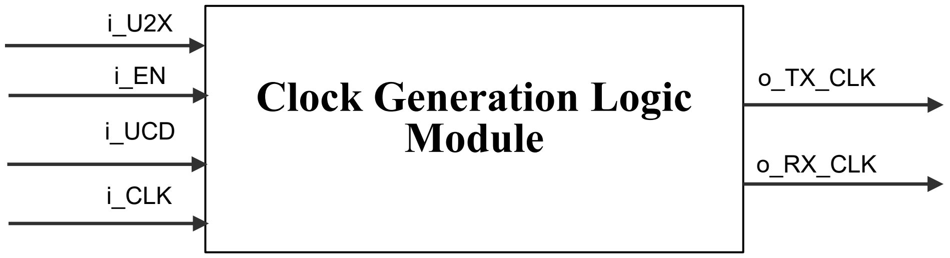

Figure 3-9. Clock Generation Logic Module

Table 3-6. Description of Signals in the Clock Generation Logic Module

Description	Signal Width	Signal Type	Signal
Input clock signal	1	Input	i_CLK
Set transmission speed	1	Input	i_UCD
Double the transmission speed	1	Input	i_U2X
Module activation	1	Input	i_EN
TX unit clock	1	Output	o_TX_CLK
RX unit clock	1	Output	o_RX_CLK

3-3-2- Transmitter Module

Figure (3-10) shows the transmitter module, which is designed to send one byte of data serially. The pin details of this module are provided in Table (3-7). The state machine of this module is illustrated in Figure (3-11).

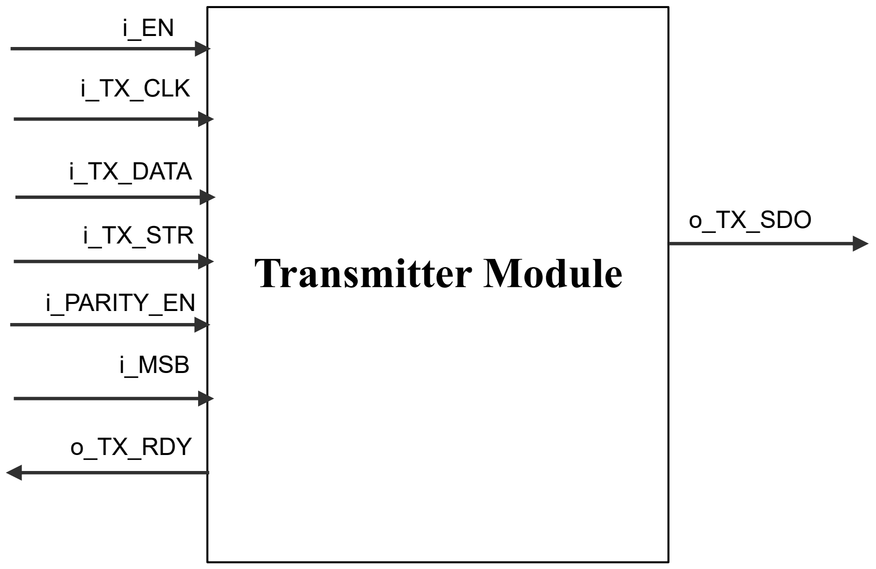

Figure 3-10. Transmitter Module

Table 3-7. Description of Signals in the Transmitter Module

Description	Signal Width	Signal Type	Signal
Module activation	1	Input	i_EN
TX unit clock	1	Input	i_TX_CLK
Start data transmission	1	Input	i_TX_STR
Transmitted data	8	Input	i_TX_DATA
Parity bit enable	1	Input	i_PARITY_EN
Data order selector: MSB-first or LSB-first	1	Input	i_MSB
Module ready to transmit	1	Output	o_TX_RDY
Serial TX channel	1	Output	o_TX_SDO

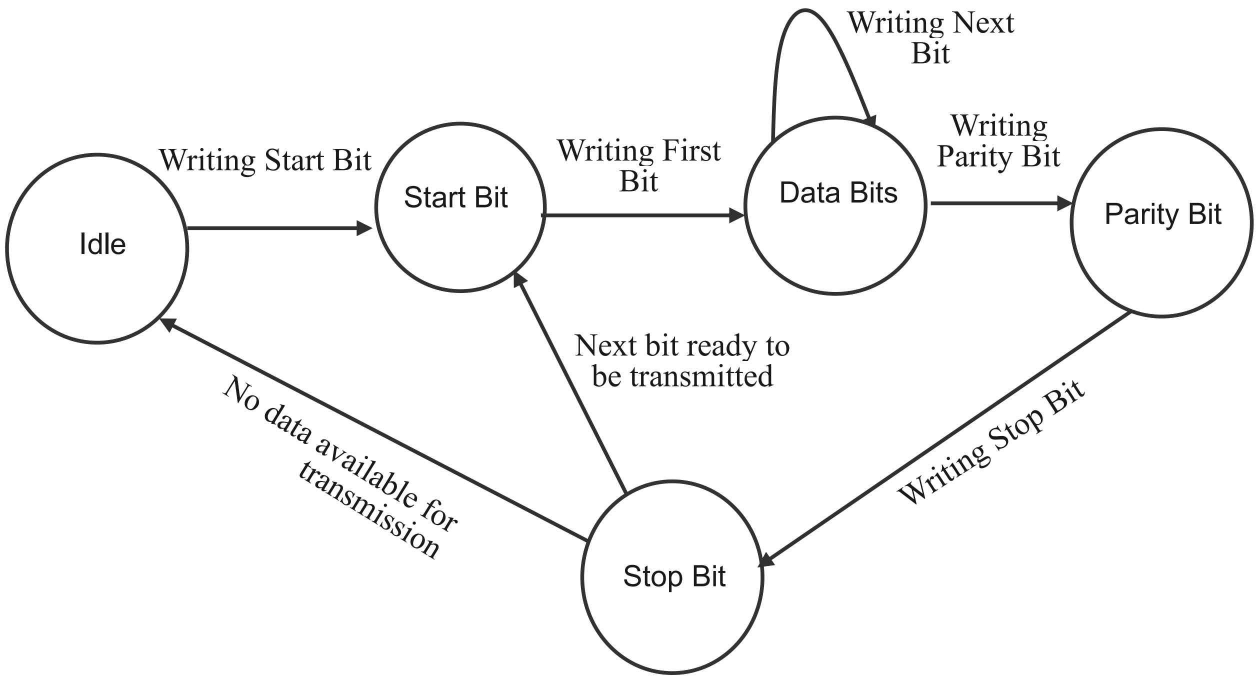

Figure 3-11. Transmitter Module State Machine

3-3-3- Receiver Module

Figure (3-12) illustrates the receiver module, which is designed to receive one byte of data serially. The pin details of this module are provided in Table (3-8). The state machine of this module is depicted in Figure (3-13).

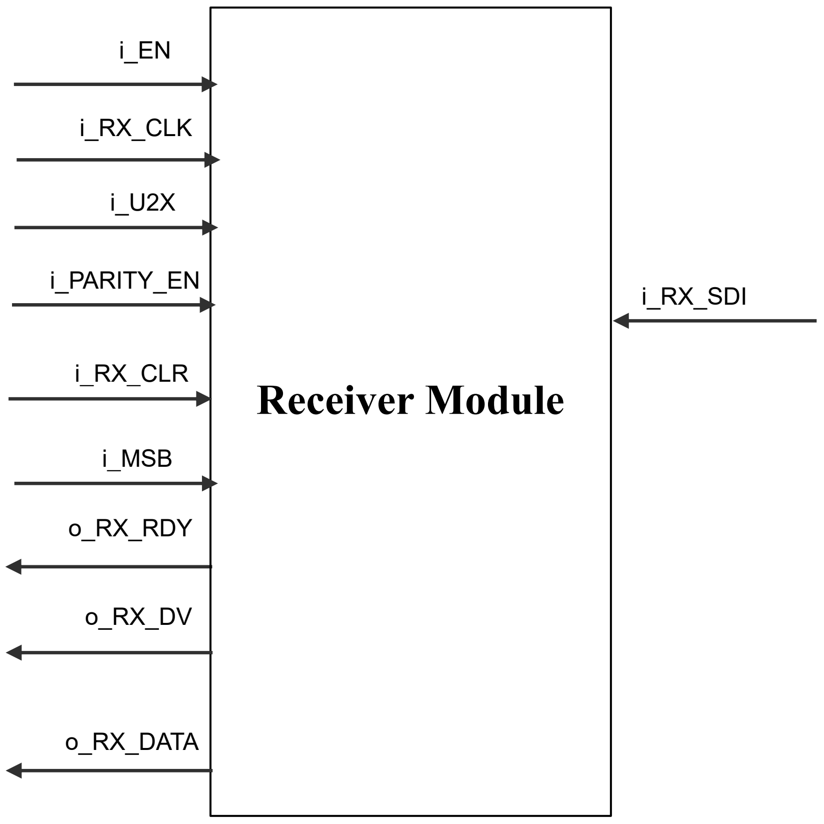

Figure 3-12. Receiver Module

Table 3-8. Description of Signals in the Receiver Module

Description	Signal Width	Signal Type	Signal
Module activation	1	Input	i_EN
RX unit clock	1	Input	i_RX_CLK
Double the transmission speed	1	Input	i_U2X
Parity bit enable	1	Input	i_PARITY_EN
Read received data	1	Input	i_RX_CLR
Serial RX channel	1	Input	i_RX_SDI
Data order selector: MSB-first or LSB-first	1	Input	i_MSB
Ready to receive new data	1	Output	o_RX_RDY
Indicates correctness or error in received data	1	Output	o_RX_DV
Received data	8	Output	o_RX_DATA

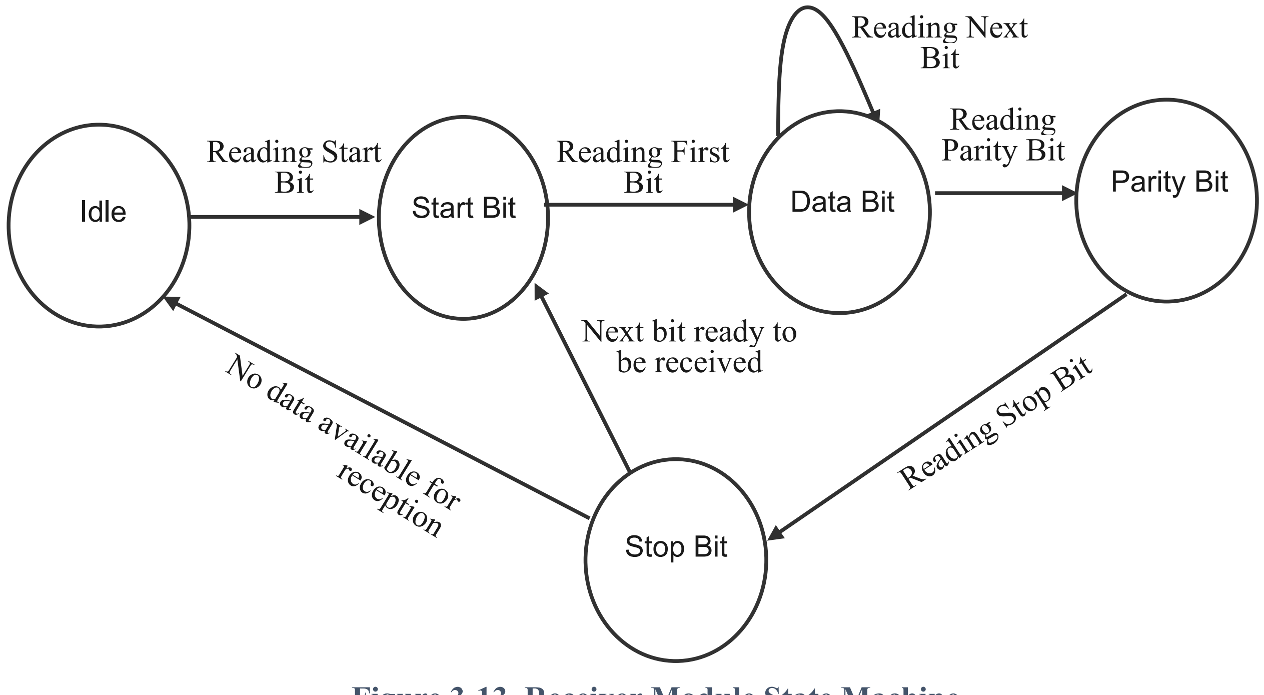

Figure 3-13. Receiver Module State Machine

3-4- Implementation of Communication Between Two ESP32 Modules

This section discusses the implementation of communication between the two ESP32 modules, which is used in the wireless mode of the system. An overview of this communication process is illustrated in Figure (3-14).

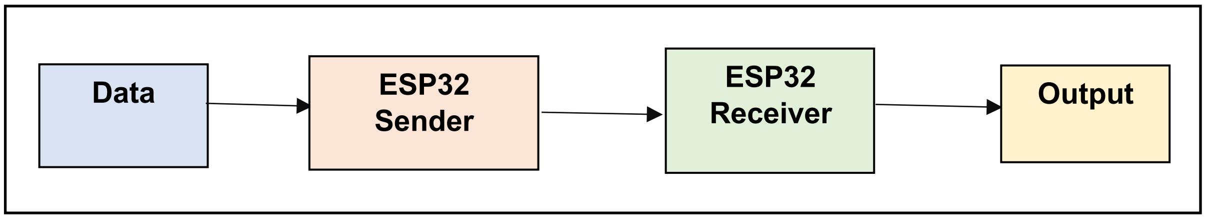

Figure 3-14. Block Diagram of Interaction Between Two ESP32 Modules

The communication between the two ESP32 devices involves a combination of UART and Wi-Fi protocols. The first device receives data from an external source, such as an FPGA, via its UART interface. When data arrives at its UART receive pin, the transmitting ESP32 captures these bytes and prepares them for wireless transmission. Using its Wi-Fi capabilities, the device creates a UDP packet to encapsulate the received data. This packet is then sent over the Wi-Fi network to a specific IP address and port where the second device is awaiting reception.

For the devices to communicate effectively, they must first recognize and connect to each other. The first device connects to the Wi-Fi network provided by the second device, which acts as a Wi-Fi access point. During this process, the first device searches for a specific SSID broadcast by the second device, using pre-shared credentials for authentication and to establish a secure link. Once connected, the two devices can now reliably interact. The first device sends data to the IP address and port associated with the second device, confirming identification and connection between them. The structure of this communication is illustrated in Figure (3-15).

```
const char* ssid = "ESP32_AP";

const char* password = "12345678";

const char* udpAddress = "192.168.4.1";

const int udpPort = 1234;
```

Figure 3-15. Communication Structure Between Two ESP32 Modules

At the receiving end, the second ESP32 is configured as a Wi-Fi access point to manage incoming connections. It continuously monitors its designated UDP port for any incoming data packets from the first unit. Upon receiving a packet, the device extracts the encapsulated data and directs it to its UART interface for further processing, if necessary. This setup enables seamless and wireless transmission of data initially received via UART, allowing the two units to communicate effectively over a network, even when physically separated.

3-5- System Implementation

In Section 3-1, the overall architecture of the system was described. In this section, the implementation of the system in both wired and wireless modes is examined in detail. First, the wired implementation of the system will be explained, followed by a discussion of the differences and necessary additions for the wireless implementation. This comprehensive analysis provides a solid foundation for fully understanding how the system operates under different conditions.

3-5-1- System Implementation for Wired Mode

This section provides a comprehensive explanation of the system implementation for the wired mode. To balance the workload between the two FPGA devices, the computations of the convolutional neural network have been divided between them—such that the first FPGA handles the convolutional layer, pooling layer, and flattening operation, while the second FPGA processes the fully connected layer. The overall system process is illustrated in Figure (3-16).

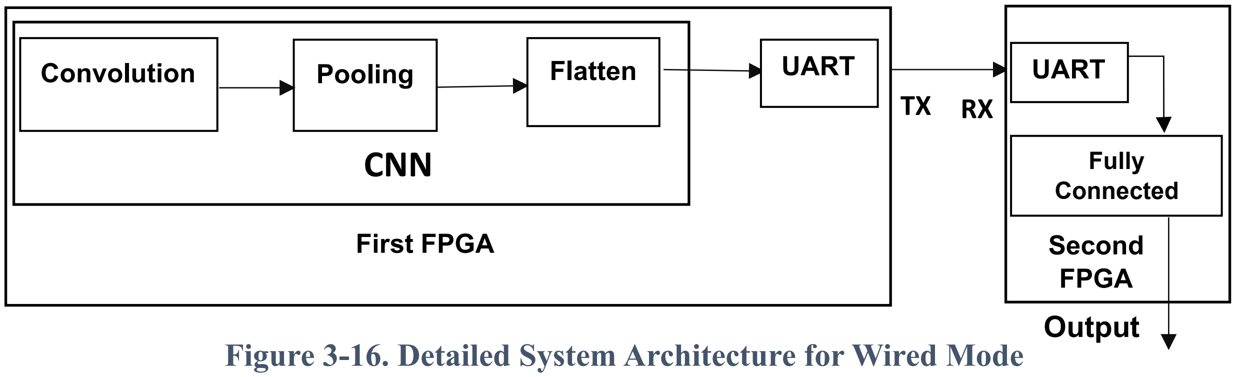

Figure 3-16. Detailed System Architecture for Wired Mode

The processing in this system begins with the convolutional layer module. Once the computations in this layer are complete, the pooling layer is activated, which reduces the dimensions of the image. The resulting outputs are sequentially sent to the flattening module. Upon the completion of operations in this module, the neural network computations on the first FPGA are finalized. To transfer the intermediate results to the second FPGA, the UART module is used. This module sends the results bit by bit through the TX line. The transmission begins with a start bit, followed by the data bits, and ends with a stop bit. The receiving device monitors the RX line to detect these bits and reconstructs the original data upon reception. The start and stop bits are crucial, as they mark the beginning and end of each data packet, ensuring the receiver interprets the data stream correctly. The reconstructed data is then forwarded as input to the fully connected module, which performs the necessary computations and generates the final output. The system pin configurations for the first and second FPGA are provided in Tables 3-9 and 3-10, respectively.

Table 3-9. Description of Signals in the First FPGA

Description	Signal Width	Signal Type	Signal
Input clock signal	1	Input	Clock
Start neural network data processing	1	Input	Start
Module reset	1	Input	Reset
First output of the transmitting FPGA	1	Input	Output11
Second output of the transmitting FPGA	1	Output	Output12
Third output of the transmitting FPGA	1	Output	Output13
Fourth output of the transmitting FPGA	8	Output	Outpu14
Check data readiness for transmission	1	Output	Valid_Data_1
Activate data stream	1	Output	EnableData
Serial TX channel	1	Output	TX
Serial RX channel	1	Input	RX

Table 3-10. Description of Signals in the Second FPGA

Description	Signal Width	Signal Type	Signal
Input clock signal	1	Input	Clock
Serial RX channel	1	Output	RX
Serial TX channel	1	Input	TX
Module operation reset	1	Output	Reset
First output of the receiving FPGA	1	Output	Output21
Second output of the receiving FPGA	1	Output	Output22
Readiness for the next processing stage	1	Output	Valid_Data_2

3-5-2- System Implementation for Wireless Mode

The system implementation in the wireless mode is similar to the wired mode described in Section 3-5-1. The main difference lies in how the intermediate data is transferred between the two FPGAs, where wireless communication is used in this case. The overall system process is illustrated in Figure 3-17. It is worth mentioning that the port pins and signals for this implementation are the same as those presented in Tables 3-9 and 3-10 in Section 3-5-1.

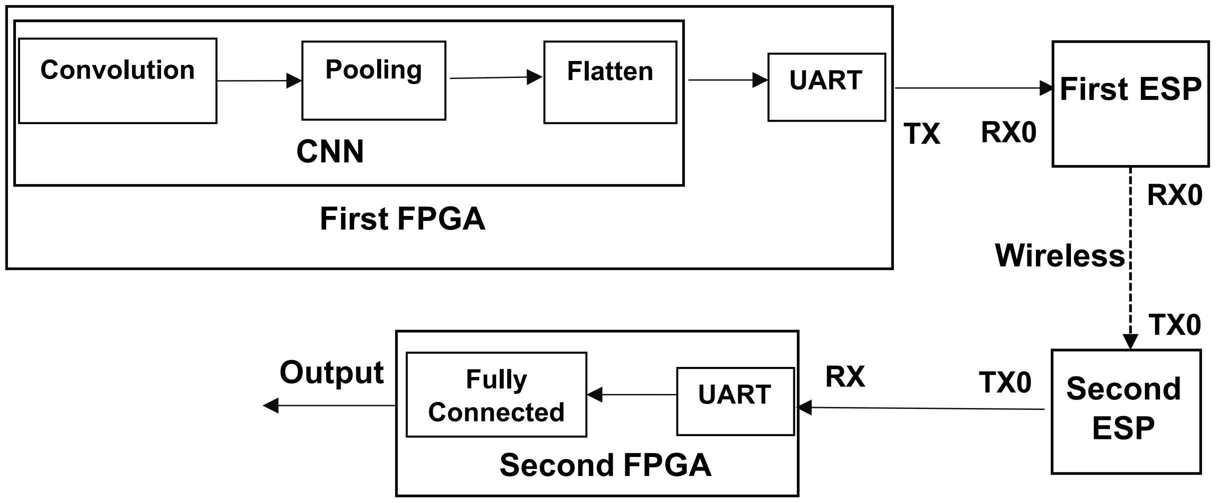

Figure 3-17. Detailed System Architecture for Wireless Mode

The system process begins with performing mathematical operations—just like in the wired method—on the first device. After computations are completed on the first FPGA, instead of sending the intermediate results directly via a wired connection to the second FPGA, the results are first transmitted through the TX line of the UART module to the RX0 input of the first ESP32 module. This module, upon receiving the data, wirelessly transmits them to the second ESP32 module. The second module then forwards the received data via its TX0 output to the RX input of the UART module connected to the second FPGA. This wireless transmission enables communication between the FPGAs without the need for direct wiring. Once the second FPGA receives the data, the process continues similarly to the wired mode, performing fully connected layer computations to produce the final output.

3-6- Summary

In this section, the implementation of the project was discussed. First, the overall system structure was reviewed, followed by an explanation of the convolutional neural network (CNN) and UART module structures. Then, the interaction between the two ESP32 devices was described. Finally, the system architecture and implementation in both wired and wireless modes were explained.

Chapter 4: Implementation Results

In this section, the project implementation images, the results obtained, and the comparison between the two project modes are reviewed.

4-1- Implementation Images and Hardware Results

In this section, we examine the sample computations used in the hardware implementation of the project. As explained in Chapter 3, initially, the transmitting FPGA performs the computations for the convolutional layer, pooling layer, and flattening operations, and then sends the intermediate results to the receiving FPGA. The receiving FPGA is responsible for performing the fully connected layer operations and generating the final output. In Figure (4-1), the computations and values used are shown.

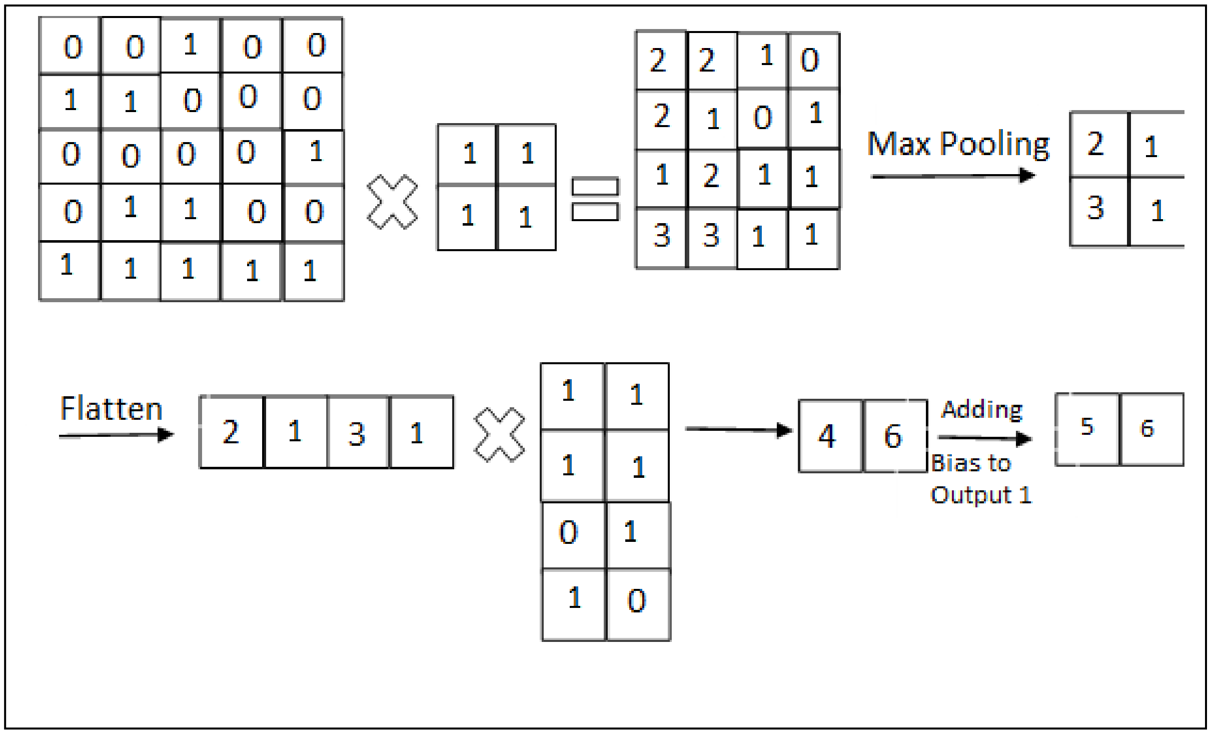

Figure 4-1. Example Calculations Used in Hardware Implementation

According to Figure (4-1), it is expected that the first FPGA will produce four outputs, and the second FPGA will generate two outputs. The implementation results on the FPGA boards for the wired and wireless modes are shown in Figures (4-2) and (4-3), respectively.

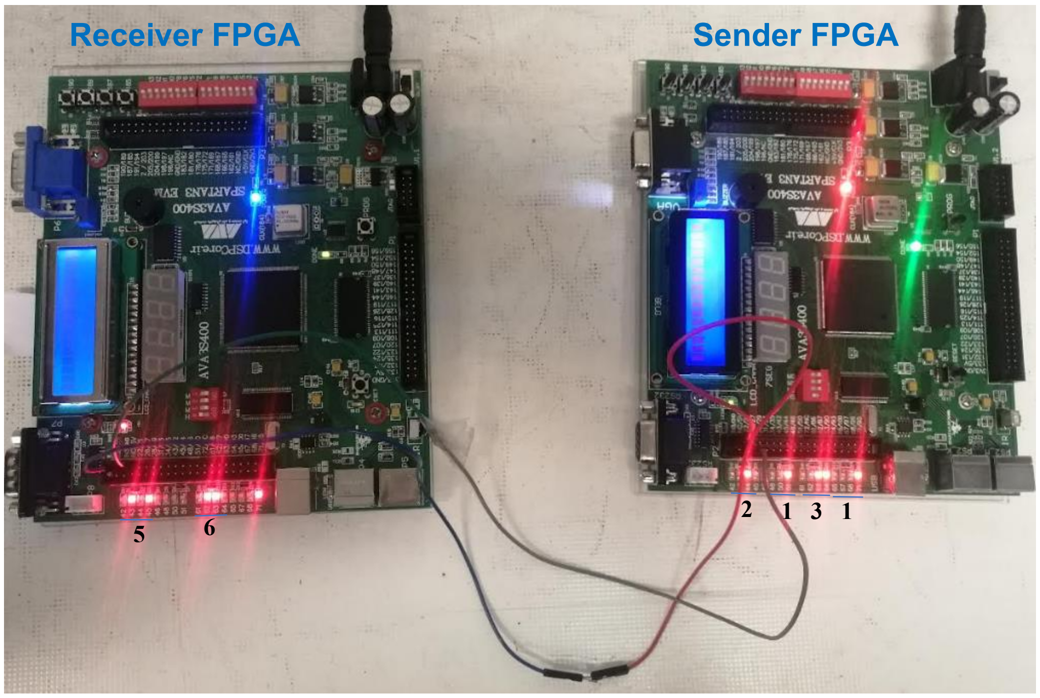

Figure 4-2. Hardware Implementation of the Project in Wired Mode

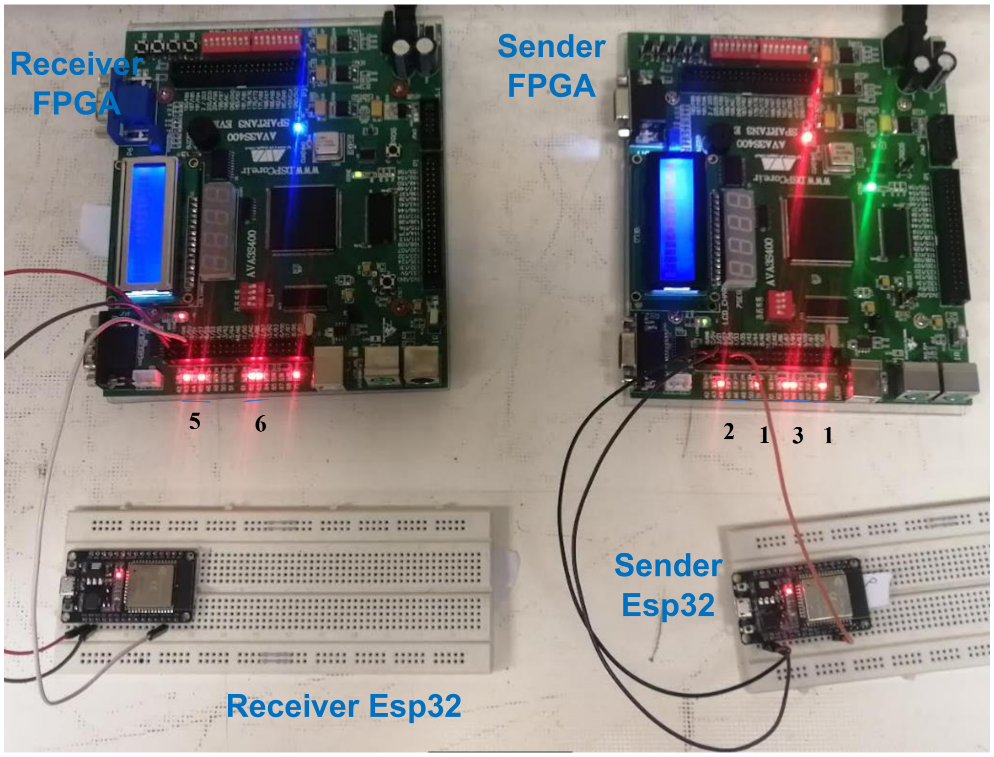

Figure 4-3. Hardware Implementation of the Project in Wireless Mode

Based on the outputs of Figures (4-2) and (4-3) and their alignment with the outputs of Figure (4-1), it can be concluded that the project is functioning correctly.

4-2- Comparison of the Two Project Modes

In this section, we compare the two implemented modes using various features and parameters. The comparison between these two system modes is presented in Table (4-1).

The system speed and transmission time have been calculated using the following formulas:

(4-1) $$Baud = \frac{CF}{8(UBRRn+1)} \qquad Transmission\ time = \frac{1}{Baud}$$

The system speed and transmission time are derived from the baud rate and transmission time relationships, respectively. In these equations, CF represents the system clock frequency, and UBRRn indicates the value stored in the register to set the baud rate.

Table 4-1. Comparison of the Features of the Two Implemented Modes

Wireless Communication	Wired Communication	Feature
Six wires	Two wires	**Number of wires used**
Higher, due to the cost of ESP32 modules and breadboards	Lower, as only cables and connectors are needed	**Cost**
More scalable; adding more devices is easier without additional wiring	Limited to the number of UART ports and available physical connections	**Scalability**
38,461.5 bps	76,923 bps	**Speed**
416 µs	208 µs	**Transfer time**
Higher, due to the extra power required for wireless communication	Lower	**Power consumption**
Lower immunity to interference	High immunity to interfaces	**Noise immunity**
Wider Range	Limited to wire length	**Range**
Possibility of data loss due to noise	Very high reliability; ensures accurate results without data loss	**Reliability**
More complex setup required, especially for managing wireless connections and synchronizing CNN processing across FPGAs	Simple, stable, and direct connections	**Ease of implementation**
More challenging, requires debugging wireless connection and synchronizing CNN processes across both FPGAs	Easier to set up and maintain, with fewer issues during operation	**Setup and maintenance**
Potential security risks from wireless communication, although encryption mitigates some concerns	More secure with minimal risk, as physical access to the wires is needed	**Security**
No physical connection needed between FPGAs, but wiring is still needed between ESP32 and FPGA	Physical space required for cabling between FPGAs	**Physical limitations**

Based on the explanations in the above table, it can be concluded that the wired mode performs better.

4-3- Summary

The hardware of the project has been examined in both wired and wireless modes. Finally, the features of both system modes are compared, and it is concluded that the wired mode performs better.

Chapter 5: Conclusion and Recommendations

In the final chapter of this project, we will summarize the previous chapters and draw conclusions. Finally, suggestions for future work will be presented.

5-1- Summary and Conclusion

This project focused on the implementation and comparison of wired and wireless communication between two FPGAs using the convolutional neural network (CNN) framework. The main objective was to assess the performance, reliability, and practicality of both communication methods in a real-world FPGA environment, with the goal of determining the most suitable approach for high-performance applications.

Initially, a solid theoretical foundation was established, exploring fundamental concepts such as CNN architecture, layer functions, and the complex computations involved in neural network operations. Then, the UART protocol was examined for implementing communications in the project. Understanding these basic concepts was crucial for the subsequent implementation stages and provided the necessary background for comparing communication methods.

Next, we focused on the precise design and development of system modules, including a detailed explanation of each component and the overall system architecture. This process involved designing the CNN modules on the FPGA, detailing the UART communication modules, and managing the key signals that facilitated communication between the two FPGAs. This careful breakdown was essential for ensuring the system operated effectively and for providing a clear understanding of how each component interacted to achieve seamless communication.

In the final stage, we demonstrated the wired and wireless communication methods using a practical example to showcase CNN performance on the FPGA. A comparative analysis showed that the wired communication method outperformed the wireless method. While the wireless approach offers flexibility, it faces challenges with data transfer rates and signal stability, especially in environments with potential interference. This project emphasizes the importance of selecting the appropriate communication method based on the specific needs and constraints of the application, highlighting the trade-offs between performance and flexibility that engineers must consider when designing FPGA-based systems.

5-2- Recommendations

In this section, we explore ideas that can expand the scope of this project in the future and lead to better results. The goal of these suggestions is to improve the performance of the project and enhance its practical applications. One important step that can be taken to improve this project is the use of more advanced convolutional neural network models, such as LeNet5. Due to its more complex structure and greater number of layers, this model would be an excellent choice for expanding this research. Employing LeNet5 could lead to a more balanced distribution of the computational load between the two FPGAs, thereby increasing the workload of both devices more effectively. Additionally, using this model could significantly improve the accuracy and efficiency of the system.

Moreover, considering the use of other communication protocols, such as SPI and I2C, for the implementation of this research could be seriously explored. These protocols, with their specific features, may offer suitable options for data transfer between various components of the system. A more detailed evaluation of these protocols could assist in the optimal selection of communication methods for future projects, ultimately leading to improved system performance.

Finally, another proposed strategy to expand this research is increasing the number of FPGA boards used. This increase would be designed in such a way that each FPGA board would be dedicated to the computation of a specific layer of the convolutional neural network. By doing so, the distribution of computations among different boards would be optimized, and in addition to increasing processing speed, it would also enable the use of more complex neural networks with additional layers. This step could significantly enhance the project's capacity to handle more complex and larger-scale computations.

References

[1] Haykin, S. (2009). *Neural Networks and Learning Machines* (3rd ed.). Pearson Education India.

[2] Agatonovic-Kustrin, S., & Beresford, R. (2000). Basic Concepts of Artificial Neural Network (ANN) Modeling and Its Application in Pharmaceutical Research. *Journal of Pharmaceutical and Biomedical Analysis, 22*(5), 717-727.

[3] Bengio, Y., Goodfellow, I., & Courville, A. (2017). *Deep learning* (Vol. 1). MIT Press.

[4] Phung, V. H., & Rhee, E. J. (2019). A High-Accuracy Model Average Ensemble of Convolutional Neural Networks for Classification of Cloud Image Patches on Small Datasets. *Applied Sciences, 9*(21), 4500.

[5] Aghdam, H. H., & Heravi, E. J. (2017). Guide to Convolutional Neural Networks. *Springer.*

[6] Shyam, R. (2021). Convolutional Neural Network and Its Architectures. *Journal of Computer Technology & Applications, 12*(2), 6-14.

[7] Véstias, M. P. (2021). Convolutional Neural Network. In M. Khosrow-Pour (Ed.), *Encyclopedia of Information Science and Technology* (5th ed., pp. 12-26). IGI Global.

[8] Zhao, X., Wang, L., Zhang, Y., Han, X., Deveci, M., & Parmar, M. (2024). A Review of Convolutional Neural Networks in Computer Vision. *Artificial Intelligence Review, 57*(4), 99.

[9] Jayawardana, R., & Bandaranayake, T. S. (2021). Analysis of Optimizing Neural Networks and Artificial Intelligent Models for Guidance, Control, and Navigation Systems. *International Research Journal of Modernization in Engineering, Technology and Science, 3*(3), 743-759.

[10] Yani, M., Budhi Irawan, S., & Setiningsih, S. C. (2019). Application of Transfer Learning Using Convolutional Neural Network Method for Early Detection of Terry's Nail. In *Journal of Physics: Conference Series* (Vol. 1235, No. 1, p. 012001). IOP Publishing.

[11] Guissous, A. E. (2019). Skin Lesion Classification Using Deep Neural Network. *arXiv preprint arXiv:1911.07817*.

[12] Mishra, V., & Kane, L. (2023). A Survey of Designing Convolutional Neural Network Using Evolutionary Algorithms. *Artificial Intelligence Review, 56*(6), 5095-5132.

[13] Lambert, T. R. (2017). An Introduction to Microcontrollers and Embedded Systems. *Auburn University. July*, *344*.

[14] Gupta, A., & Gupta, A. (2019). UART Communication. In *The IoT hacker's handbook: A practical guide to hacking the Internet of Things* (pp. 59-80).

[15] Ramdeane, A., & Lynch, L. (2020). Low-Cost Seismic Data Acquisition System Based on Open-Source Hardware and Software Tools. *The University of the West*