International Journal of Embedded and Wireless Communication Systems
Volume 2, Issue 1 2026

# Wireless Communication Between Embedded Systems based on FPGA

**Mr. Souvinka Roy**
**Department of Electronics & Communication Engineering Techno International Batanagar Kolkata, India**

*Research Article*

**Date of Submission: 11-10-2025** **Date of Acceptance: 10-12-2025** **Date of Publication: 20-01-2026**

**Abstract**
Nowadays, most applications need a wireless connection to easily send and receive data across different networks. In most instances, this was achieved by using microcontrollers and RF transceiver modules. This project aims to provide a wireless interface layer between end-user devices using different wireless media by using an interconnect model. For this project, an FPGA-based model is being used. Using FPGAs rather of microcontrollers has many advantages, such as lower power consumption, quicker processing, and more efficiency. We also look at a number of wireless media and choose the best one. The project will establish communication using UDP packets to allow wireless data transfer using dependable Wi-Fi technology.

***Keywords***: ***Verilog, FPGA, NodeMCU, and wireless communication.***

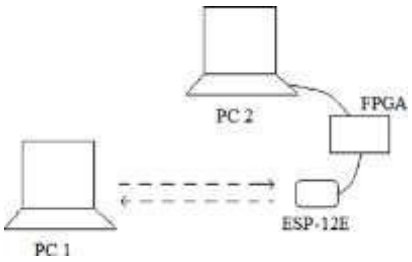
## 1. Introduction

In the modern world, technology is essential to human existence. Manual labour is increasingly being replaced by automation and machine-controlled applications. Microcontrollers are used to electrically control the majority of the systems. In several sectors, FPGAs are now taking the place of the traditional Microcontrollers. Compared to conventional implementations, FPGA (Field Programmable Gate Array) systems provide several benefits [1]. You can incorporate some functionality into software and some into hardware by adding an FPGA to a design. Since complexity is cheap in software and costly in FPGAs, where you have a restricted number of logic parts, anything involving complicated decision making should generally be in software. Software often performs poorly when it comes to precise timing and a variety of bit-level methods. FPGAs are used for low-level data communications and test equipment that must observe and carry out low-level protocols because they have high-performance bit-level processing that is more efficient than software (high-performance cryptography is a good example) and are a good way to sample and generate I/O with accurate timing [2]. This project's goal is to provide wireless connection between embedded systems by using FPGAs for data transfer and control needs. modest data rates (1-2 Mbps) and a modest range (50-55 feet) should be used for communication.

## 2. Hardware Setup

Figure 1 provides a description of the hardware configuration or block diagram. As we can see, the ESP-12E wireless module and FPGA have made it possible for PC1 and PC2 to communicate wirelessly. Details on the various hardware parts and wireless media are provided in the section that follows.

**Fig. 1:** Functional block diagram of hardware setup




### 2.1. Xilinx Digilent Nexys 3 FPGA Board

The Nexys 3 Spartan 6 board is chosen among the various FPGA boards (such as the Spartan 3E and Spartan 6) because of its Adept Programming feature and UART interface. Based on the Xilinx Spartan-6 LX16 FPGA, the Nexys 3 is a fully functional digital circuit creation platform.

Compared to the Spartan-3 500E FPGA of the Nexys 2, the Spartan-6 is optimised for high performance logic and provides more than 50% greater capacity, performance, and resources. The Nexys 3 has an enhanced set of peripherals in addition to the Spartan-6 FPGA, such as 16 Mbytes of cellular RAM, a 10/100 Ethernet PHY, 32 Mbytes of Micron's most recent Phase Change non-volatile memory, a USB-UART port, a USB host port for mouse and keyboard, and an upgraded high-speed expansion connector [3].

This board has a number of characteristics, including:
A 324 BGA package containing a Xilinx Spartan-6 LX16 FPGA.
Its cellular RAM is 16 Mb (x16).
Additionally, Nexys 3 has 16 Mb parallel PCM non-volatile memory.

**Implemented Work**

Range Testing

This test was carried out to get the Wi-Fi ranges in terms of signal strength. The range measurements were calculated by observing the signal strength and RSSI levels in the Arduino serial monitor window.

**Table 2:** Range Test Results




| Distance(feet) | Signal strength(dBm) |
|---|---|
| 0 | -61 |
| 5 | -71 |
| 10 | -77 |
| 15 | -79 |
| 20 | -79 |
| 25 | -82 |
| 35 | -90 |
| 40 | -92 |
| 50 | -93 |
| 55 | -93 |

**Fig. 2:** Xilinx Nexys 3 FPGA board

### 2.2. Wireless Medium Selection

For the selection of the wireless medium a survey was carried out and their characteristic parameters were observed and noted. The wireless medium considered for this survey was Bluetooth, ZigBee and Wi-Fi. The following Table summarizes the Wireless Medium Survey that was carried out.

**Table 1:** Wireless Medium Survey Results [4]

| Category | ZigBee | Bluetooth | Wi-Fi |
|---|---|---|---|
| Range | 75+ m | 10m | 50+ m |
| Data Rate | 250 kbps | 1 Mbps | 1-54 Mbps |
| Freq Range | 868Mhz,916Mhz,2.4Ghz | 2.4Ghz | 2.4 GHz |
| Network Nodes | 65535 | 8 | 50 |
| Linking Time | 30ms | Up to 10s | Up to 3s |
| Cost | Low | Low | High |
| Power | Low | Medium | High |

The wireless medium considered was Wi-Fi medium so as to work with moderate range (55feet) and moderate data rate (1-2 Mbps).

### 2.3. Wi-Fi Module Selection

For the Wi-Fi module a survey observing its various at- tributes was carried out. We had considered PMOD Wi-Fi modules, ESP8266 for this survey. Finally Wi-Fi ESP8266 (NodeMCU) is selected due to its simplicity, cost-effectiveness and ability to flash its firmware [5].

Some features of Wi-Fi Module NodeMCU:

- NodeMCU has a L106 32-bit RISC microprocessor core running at 80 MHz
- 64 KiB of instruction RAM, 96 KiB of data RAM.
- It has integrated TR switch, LNA, power amplifier and matching network.
- WEP or WPA/WPA2 authentication or open networks.
- Also it has 16 GPIO pins.



**Fig. 3:** ESP8266-12E NodeMCU v-1.0

### 2.2. UDP packet transmission

UDP uses a simple connectionless communication model with a minimum of protocol mechanism. UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram [6]. It has no hand-shaking dialogues, and thus exposes the user's program to any unreliability of the underlying network.

#### 2.2.1. UDP transmission between two NodeMCUs:

For this test we connect both NodeMCU module to different ports and program them via Arduino IDE [7].
Steps for programming the NodeMCUs are:

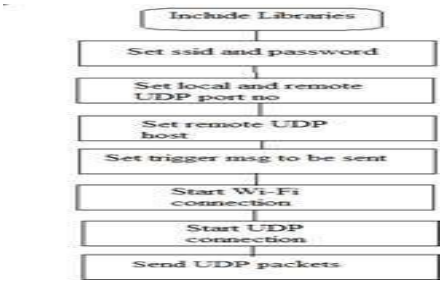



**Fig. 4:** Sender NodeMCU flowchart

As seen in Figure 4, the first step is to include the libraries of ES-PWiFi.h and UDP.h on the Arduino IDE. Then two constant vari-ables are declared to hold the value of ssid and password of the Wi-Fi network. Later the local and remote UDP port no is set. This can be any of the available UDP port numbers.
Then the remote UDP host id is set which is the IP address of the receiver NodeMCU module.
The trigger message to be sent is stored in a character array which can be any string of characters. Then Wi-Fi connection and UDP connection is then turned on by the command UDP.begin(). Now the UDP packets can be sent using the command UDP.beginPacket().

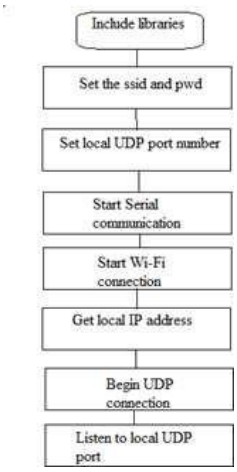






**Fig. 5:** Receiver NodeMCU flowchart

As seen in Figure 5, the receiver side flowchart has been given. Again the first step is to include the ESPWiFi and UDP header files. The ssid and password is stored in a constant character. The local UDP port number is set. Then the serial communication and Wi-Fi connection is started. The local IP address is allotted to the NodeMCU by the Wi-Fi network and then the UDP connection is begun. The incoming packets can then be received at the local UDP port by parsing the packets using UDP.parsePacket(). Figure 6 shows the output window.

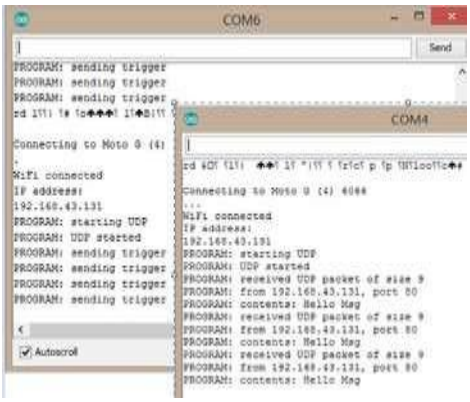



**Fig. 6:** Output window at Arduino serial monitor

### 3.2.3. UDP Packet Transmission from PC to NodeMCU:

For this test we connect a single NodeMCU module to Laptop and try to send packets from PC to NodeMCU. In order to send the packets we use the concept of Socket programming in Linux sys-tem [8].

Steps for Socket Programming are given in Figure 7.

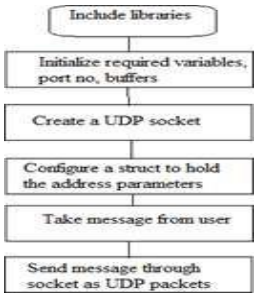



**Fig. 7:** Flowchart for socket programming (Linux)

Also we can observe from Figure 8 that the message typed in the terminal window of Linux system is available at the serial port connected to the NodeMCU module.

### 2.3. FIFO memory for Read and Write Data

In this project we use the FIFO memory to write and read data into the FPGA board. FIFO stands for First In First Out, and the cod-ing is done in Verilog language. Two FIFO memories were used. One for writing the data and one for reading the data. Before im-plementing on hardware the FIFO code was tested on software and simulated using the ISim software. The FIFO coded was of 32 bits word data and the depth of the FIFO was 8 i.e. (32x8) .The simu-lation results are as given below:



**Fig. 9:** Simulation waveforms for FIFO memory

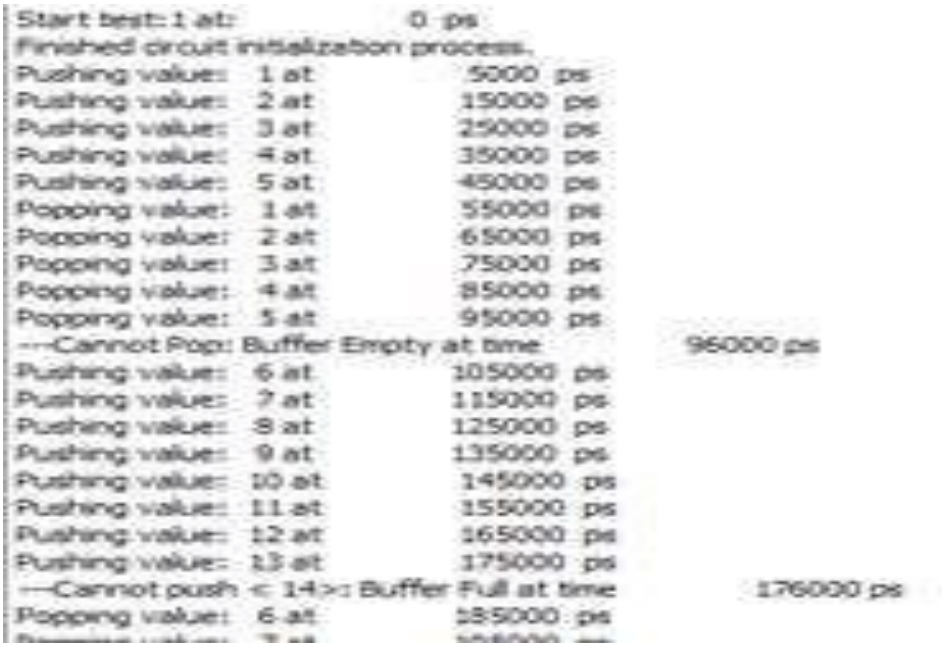



**Fig. 10:** Console output for simulation

Figure 9 shows the FIFO memory locations being pushed (write) and popped (read) with data and Figure 10 shows the particular operation details on the console window.

### 2.4. Final implementation

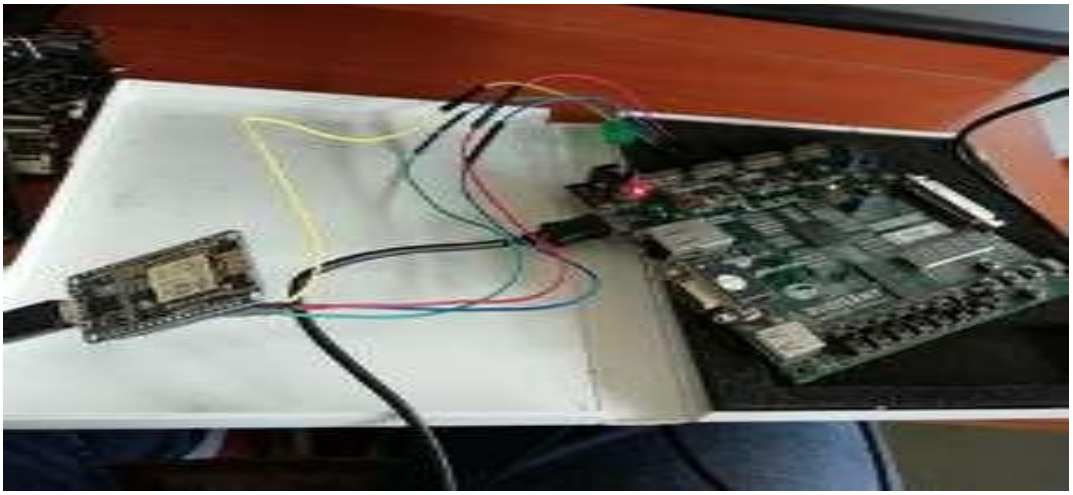


**Fig. 11:** Actual Hardware Setup

Figure 11 shows the actual hardware setup. We can now combine the above individual implementations to transmit and receive data from client (Linux User) to server that is NodeMCU.

3.4.1 Transmission using default format of UDP Packet:

The NodeMCU is connected to FPGA Board via the GPIO pins.

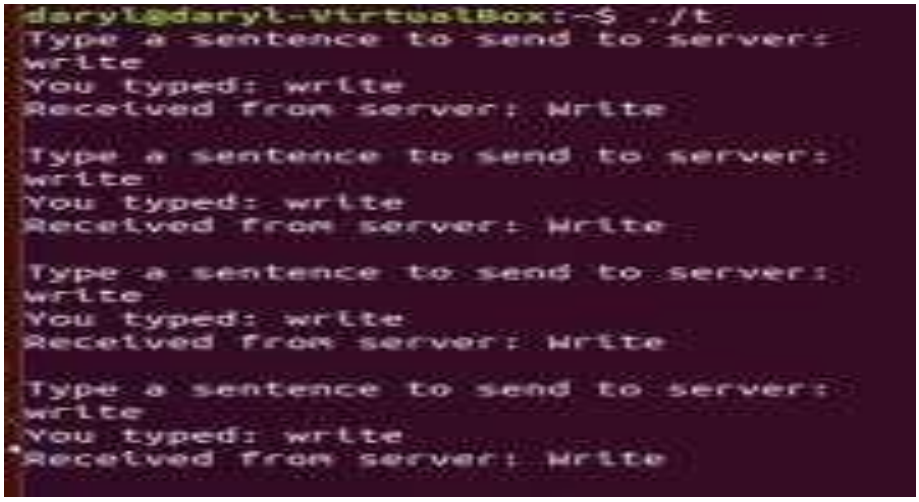



**Fig. 12:** Client side Write command




**Fig. 13:** Client side Read command

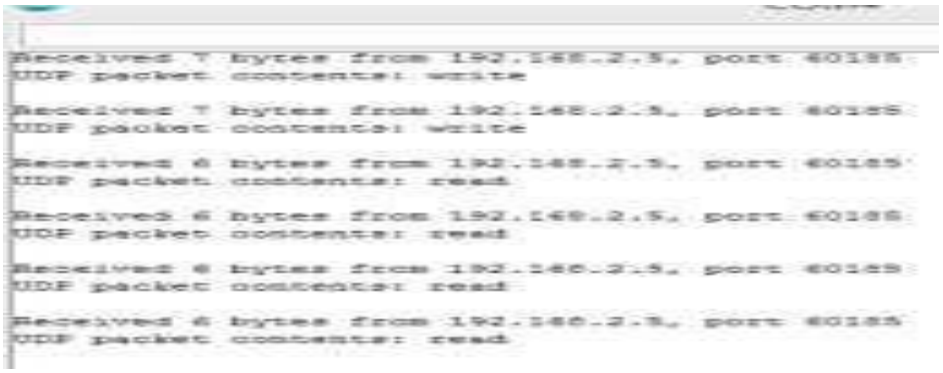



**Fig. 14:** Client side Read command

As we can see from Figure 12 and 13, we give the write or read command from the Linux client. This command is received as a UDP packet at the NodeMCU which is connected to the Nexys 3 FPGA board. Depending on the command write or read enable pins are activated on the FPGA board which allows the internal FIFO to read or write data to the FIFO memory.

The FIFO output is again then given to the NodeMCU module via GPIO pins which is further transmitted to the Linux client as UDP packets. This is shown in Figure 14. This approach though causes the transmission to have increased overhead bits as each bit is sent as a character value which requires one byte of space. New format of packet structure is shown below.

### 3.4.2. Transmission with formatting of UDP packet:

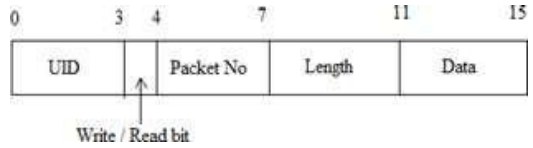



**Fig. 15:** New formatted UDP packet structure

Figure 15 shows the frame structure of the UDP Packet which is a total of 2 bytes.
The frame structure consists of:

- UID : Unique Identification Number (4 bits)
- W/R bit: Bit is 1 for Write operation and 0 for Read
- Packet No: Unique packet id for each transaction(3)
- Length: Shows the length of the entire packet (4 bits).
- Data: 4 bit Data being transmitted/received.




**Fig. 16:** Client side terminal (Linux)




**Fig. 17:** NodeMCU monitor output

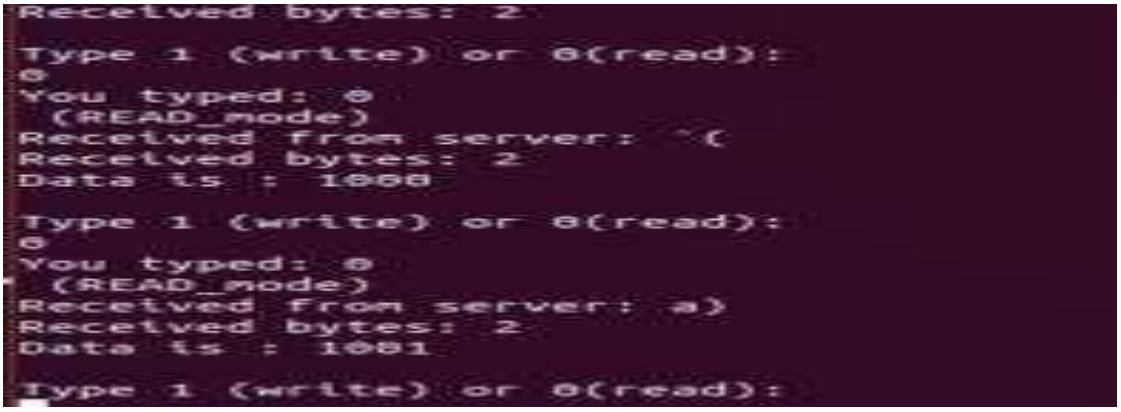



**Fig. 18:** Data read back at client side

Thus Figure 16 and 17 shows the terminal window of the Linux client from where the command to write and read data is taken from the user side. The client is first asked whether to "write" or "read" data. Once write mode is selected by pressing 1, the client can now input the data to be sent. In the above example data en-tered is binary 1000. The packet structure for the same will look like 01101001 00101000. The significance of each bit is explained earlier.
Now the character representation of this will look like "h(".
Figure 17 shows the server side output for NodeMCU module. As the data entered was 1000 in binary form, the entire packet struc-

## 3. Conclusion

Thus, we have effectively used FPGA and NodeMCU to provide wireless communication between embedded devices. This study suggests a method for switching to wireless communication for FPGAs from the current wired one. In an industrial workplace, where the whole network is operated on a single Wi-Fi network, this kind of communication is possible. Since the apps are primarily intended for monitoring and control, UDP packets may be used for communication. Additionally, employing UDP packets speeds up transmission.

## 4. Future Scope

We can now extend this method to use NodeMCUs linked to a specific wireless network to interact with additional FPGAs. Additionally, work on the SPI protocol to establish a network of FPGAs by connecting many slave FPGAs. Additionally, instead of using user-defined data for each transfer, UART proto-col may be used to convey data from the server side for a continuous pool of data to be communicated.

Reference

P.H.W.Leong, "Recent trends in fpga architectures and applica-tions",4th IEEE International Symposium on Electronic Design, Test and Applications 2008, Hong Kong, Jan., pp. 137–141, 2008."
J. Serrano, "Introduction to FPGA design," 2008.
Nexys 3Reference Manual,Digilent.[Online]. Available:log-ic/nexys-3/reference-manual
A. J. V.Abinayaa, "Case study on comparison of wireless technolo-gies in industrial applications," International Journal of Scientific and Research Publications., vol. 4, no. 2, pp. 137–141, 2 2014.
ESP8266EX Datasheet, Espressif Systems IOT Team, 2015, rev 4.3
B.A.Forouzan TCP/IP Protocol Suite, 2nd ed., S. C. Fegan, Ed.McGraw-Hill Higher Education,2002.
E.A.C.documentation.(2018,Feb.)Udp- esp8266-arduino@ONLINE.
W. W. Gay, Linux Socket Programming: By Example. Indianapolis, IN, USA: Que Corp., 2000.