International Journal of Engineering & Technology, 7 (4.41) (2018) 66-70

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Wireless Communication Between Embedded Systems based on FPGA

Daryl Narakadan[1]*, Sadhana Pai[1], Manita Rajput[1]

[1]Fr.C.Rodrigues Institute of Technology,Vashi, Mumbai Maharshtra ,INDIA
**Corresponding author E-mail:darylgeorge007@gmail.com*

Abstract

Nowadays most of the applications require wireless connectivity to seamlessly send and receive information in different networks. In most of the scenarios this was achieved via microcontrollers and RF transceiver modules. The scope of this project is to implement a wireless interface layer between end user device running on different wireless medium using a interconnect model. In this project we use a model that is based on FPGA. Using FPGA rather than microcontroller has its various advantages such as low power consumption, high computation speed and better efficiency. Also for the wireless medium we research on various medium and select the most compatible one. The project will establish communication using UDP packets, which would facilitate wireless transfer of data over robust Wi-Fi technology.

Keywords: *FPGA;NodeMCU;Verilog;Wireless Communication.*

1. Introduction

In today's world technology plays an important and vital role in the lives of human beings. Automation and machine controlled applications are being widely used to replace manual labor. Majority of the systems are electronically controlled using microcontrollers. FPGAs are now replacing the conventionally used Microcontrollers in various fields. Systems based on FPGAs (Field Programmable Gate Arrays) provide many advantages over conventional implementations [1]. Adding an FPGA to a design gives you the flexibility to put some functionality in software and some in hardware. Generally, anything that involves complex decision making should be in software, because complexity is cheap in software and expensive in FPGAs where you have a limited number of logic elements. Software is poor at many kinds of bit-level algorithms and is usually poor at precise timing. FPGA has high-performance bit-level processing which is more efficient than software (a good example is high-performance cryptography) and is a good way to sample and generate I/O with accurate timing, so they're used for low-level data communications and for test equipment that needs to observe and perform low-level protocols [2]. The scope of this project is to have wireless communication between embedded systems using FPGAs for control and data transmission requirements. The communication should take place at moderate range (50-55 feet) and moderate data-rates (1-2Mbps).

2. Hardware Setup

The hardware setup or the block diagram has been described in Figure 1. As we can see the wireless communication between PC1 and PC2 has been facilitated with the use of Wireless module ESP-12E and FPGA. The following section gives details regarding the different hardware components and wireless medium.

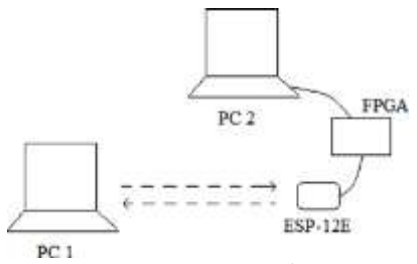


Fig. 1: Functional block diagram of hardware setup

2.1. Xilinx Digilent Nexys 3 FPGA Board

Out of the available FPGA boards (e.g. Spartan 3E, Spartan6) the Nexys 3 Spartan 6 board is selected due to its UART port, Adept Programming function. The Nexys 3 is a complete, ready-to-use digital circuit development platform based on the Xilinx Spartan-6 LX16 FPGA.

The Spartan-6 is optimized for high performance logic, and offers more than 50% higher capacity, higher performance, and more resources as compared to the Nexys 2's Spartan-3 500E FPGA. In addition to the Spartan-6 FPGA, the Nexys 3 offers an improved collection of peripherals including 32Mbytes of Micron's latest Phase Change non-volatile memory, a 10/100 Ethernet PHY, 16 Mbytes of Cellular RAM, a USB-UART port, a USB host port for mouse and keyboard, and an improved high-speed expansion connector [3].

Some of the features of this board are:

- Xilinx Spartan-6 LX16 FPGA in a 324 BGA package.
- It has 16 Mb cellular RAM(x16).
- Nexys 3 also includes 16 Mb parallel PCM non-volatile memories
- It has 10/100 Ethernet PHY.

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

3. Implemented Work

3.1. 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

3.2. 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.

3.2.1. 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:

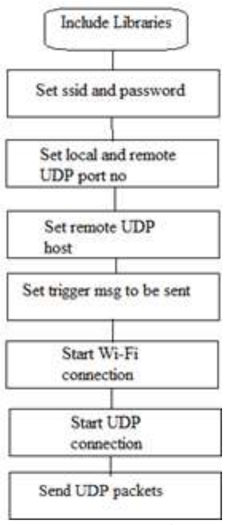


Fig. 4: Sender NodeMCU flowchart

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().

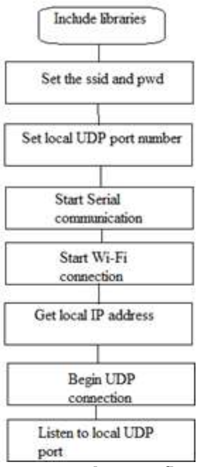
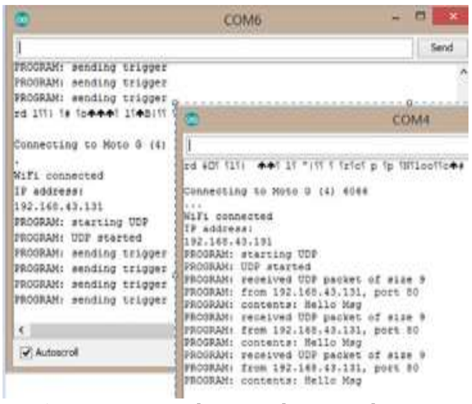


Fig. 5: Receiver NodeMCU flowchart

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.



```
PROGRAM: sending trigger
PROGRAM: sending trigger
PROGRAM: sending trigger
[illegible]

Connecting to Moto G (4)
.
WiFi connected
IP address:
192.168.43.131
PROGRAM: starting UDP
PROGRAM: UDP started
PROGRAM: sending trigger
PROGRAM: sending trigger
PROGRAM: sending trigger
PROGRAM: sending trigger
```

```
[illegible]

Connecting to Moto G (4) [illegible]
...
WiFi connected
IP address:
192.168.43.191
PROGRAM: starting UDP
PROGRAM: UDP started
PROGRAM: received UDP packet of size 9
PROGRAM: from 192.168.43.131, port 80
PROGRAM: contents: Hello Msg
PROGRAM: received UDP packet of size 9
PROGRAM: from 192.168.43.131, port 80
PROGRAM: contents: Hello Msg
PROGRAM: received UDP packet of size 9
PROGRAM: from 192.168.43.131, port 80
PROGRAM: contents: Hello Msg
```

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 system [8].

Steps for Socket Programming are given in Figure 7.

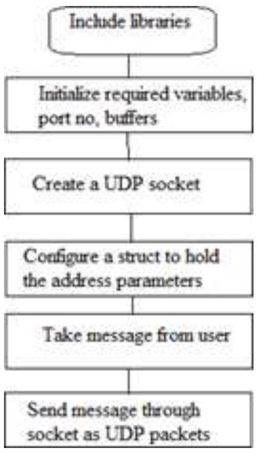
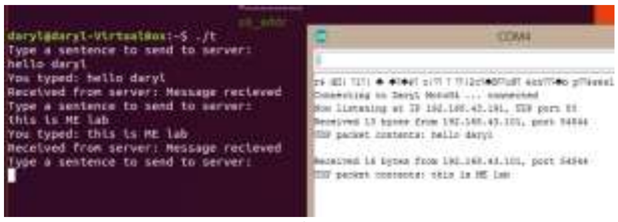

Include libraries
Initialize required variables, port no, buffers
Create a UDP socket
Configure a struct to hold the address parameters
Take message from user
Send message through socket as UDP packets

Fig. 7: Flowchart for socket programming (Linux)



```
daryl@daryl-VirtualBox:~$ ./t
Type a sentence to send to server:
hello daryl
You typed: hello daryl
Received from server: Message recieved
Type a sentence to send to server:
this is ME lab
You typed: this is ME lab
Received from server: Message recieved
Type a sentence to send to server:
```

```
[illegible]
Connecting to Daryl NodeMCU ... connected
Now listening at IP 192.168.43.191, UDP port [illegible]
Received 13 bytes from 192.168.43.101, port 54844
UDP packet contents: hello daryl

Received 14 bytes from 192.168.43.101, port 54844
UDP packet contents: this is ME lab
```

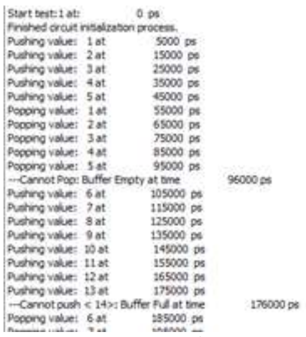
Fig. 8: UDP packet from PC to NodeMCU

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.

3.3. 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 coding is done in Verilog language. Two FIFO memories were used. One for writing the data and one for reading the data. Before implementing 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 simulation results are as given below:

Fig. 9: Simulation waveforms for FIFO memory



```
Start test:1 at:              0 ps
Finished circuit initialization process.
Pushing value:  1 at           5000 ps
Pushing value:  2 at          15000 ps
Pushing value:  3 at          25000 ps
Pushing value:  4 at          35000 ps
Pushing value:  5 at          45000 ps
Popping value:  1 at          55000 ps
Popping value:  2 at          65000 ps
Popping value:  3 at          75000 ps
Popping value:  4 at          85000 ps
Popping value:  5 at          95000 ps
---Cannot Pop: Buffer Empty at time            96000 ps
Pushing value:  6 at         105000 ps
Pushing value:  7 at         115000 ps
Pushing value:  8 at         125000 ps
Pushing value:  9 at         135000 ps
Pushing value: 10 at         145000 ps
Pushing value: 11 at         155000 ps
Pushing value: 12 at         165000 ps
Pushing value: 13 at         175000 ps
---Cannot push < 14>: Buffer Full at time        176000 ps
Popping value:  6 at         185000 ps
```

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.

3.4. Final implementation

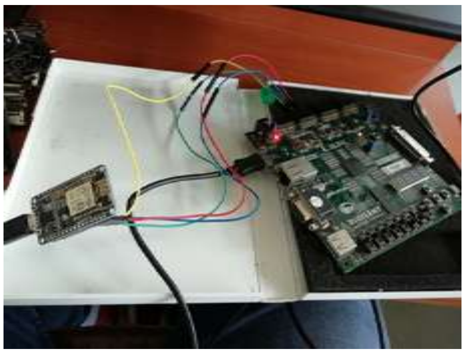
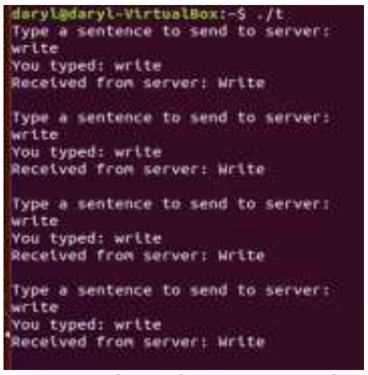


Fig. 11: Actual Hardware Setup

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.

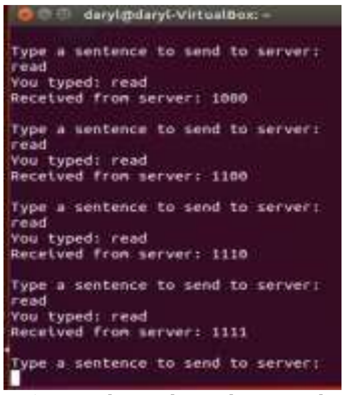
3.4.1 Transmission using default format of UDP Packet:

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



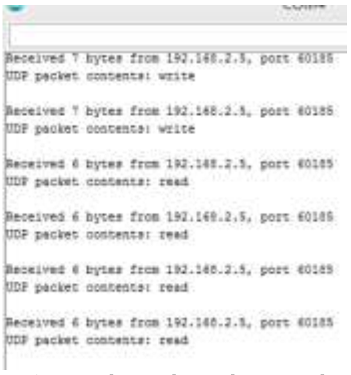
```
daryl@daryl-VirtualBox:~$ ./t
Type a sentence to send to server:
write
You typed: write
Received from server: Write

Type a sentence to send to server:
write
You typed: write
Received from server: Write

Type a sentence to send to server:
write
You typed: write
Received from server: Write

Type a sentence to send to server:
write
You typed: write
Received from server: Write
```

Fig. 12: Client side Write command



```
daryl@daryl-VirtualBox: ~

Type a sentence to send to server:
read
You typed: read
Received from server: 1000

Type a sentence to send to server:
read
You typed: read
Received from server: 1100

Type a sentence to send to server:
read
You typed: read
Received from server: 1110

Type a sentence to send to server:
read
You typed: read
Received from server: 1111

Type a sentence to send to server:
```

Fig. 13: Client side Read command



```
Received 7 bytes from 192.168.2.5, port 60185
UDP packet contents: write

Received 7 bytes from 192.168.2.5, port 60185
UDP packet contents: write

Received 6 bytes from 192.168.2.5, port 60185
UDP packet contents: read

Received 6 bytes from 192.168.2.5, port 60185
UDP packet contents: read

Received 6 bytes from 192.168.2.5, port 60185
UDP packet contents: read

Received 6 bytes from 192.168.2.5, port 60185
UDP packet contents: read
```

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:

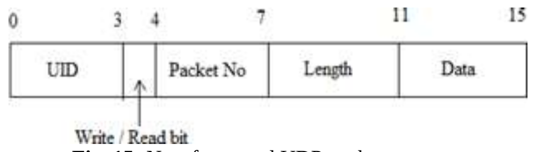
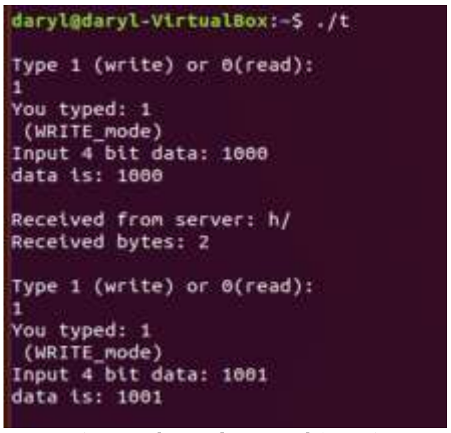


Fig. 15: New formatted UDP packet structure

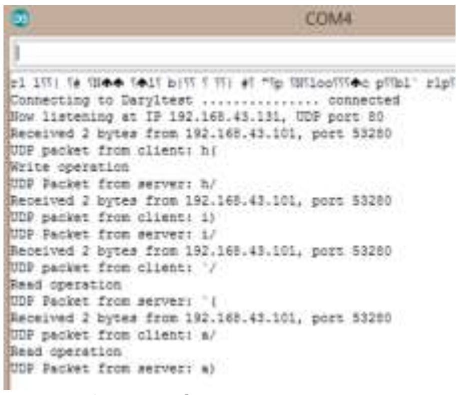
Figure 15 shows the frame structure of the UDP Packet which is a total of 2 bytes.
The frame structure consists of:

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



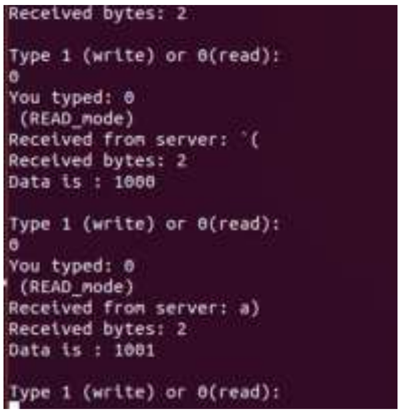
```
daryl@daryl-VirtualBox:~$ ./t

Type 1 (write) or 0(read):
1
You typed: 1
 (WRITE_mode)
Input 4 bit data: 1000
data is: 1000

Received from server: h/
Received bytes: 2

Type 1 (write) or 0(read):
1
You typed: 1
 (WRITE_mode)
Input 4 bit data: 1001
data is: 1001
```

Fig. 16: Client side terminal (Linux)



```
COM4

[illegible]
Connecting to Daryltest ............... connected
Now listening at IP 192.168.43.131, UDP port 80
Received 2 bytes from 192.168.43.101, port 53280
UDP packet from client: h(
Write operation
UDP Packet from server: h/
Received 2 bytes from 192.168.43.101, port 53280
UDP packet from client: i)
UDP Packet from server: i/
Received 2 bytes from 192.168.43.101, port 53280
UDP packet from client: `/
Read operation
UDP Packet from server: `(
Received 2 bytes from 192.168.43.101, port 53280
UDP packet from client: a/
Read operation
UDP Packet from server: a)
```

Fig. 17: NodeMCU monitor output



```
Received bytes: 2

Type 1 (write) or 0(read):
0
You typed: 0
 (READ_mode)
Received from server: `(
Received bytes: 2
Data is : 1000

Type 1 (write) or 0(read):
0
You typed: 0
 (READ_mode)
Received from server: a)
Received bytes: 2
Data is : 1001

Type 1 (write) or 0(read):
```

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 entered 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-

ture translates to "h(" in character format. This is then decoded and sent to the FPGA to be stored in FIFO memory.
Once the read command is issued as shown in Figure 18, the same data is read from the memory and sent back to client. The data sent is "`(" in character format which translates to 01100001 00101000. The last 4 bits can then be extracted to get the data back, which is 1000. Thus by doing the formatting of UDP packet structure we have also reduced the number of overhead bits and reduced the bandwidth usage for transmission and reception of UDP packets.

4. Conclusion

Thus we have successfully established wireless communication between embedded systems using FPGA and NodeMCU. This paper proposes an approach to replace the existing wired communication for FPGAs with wireless one. Such a type of communication can be achieved in an industrial workspace where the entire network is run on a specific Wi-Fi network. As the applications are mostly targeted at monitoring and controlling purposes we can use UDP packets for communication. Also using UDP packets increases the speed of transmission.

5. Future Scope

Further we can now expand this approach to communicate to other FPGAs via NodeMCU's connected to a particular Wireless Network. Also work on SPI protocol to connect a number of slaves FPGA in order to create a network of FPGAs. Also UART protocol can be used to send data from server side for a constant pool of data to be transmitted rather than making it user defined for every transmission.

Acknowledgement

I am thankful to my academic supervisor Prof. Sadhana Pai who has been a great support in the process of designing this project. Her experience and wisdom has been of great help to me. I express deep gratitude towards my external supervisor, Dr. Raghunandan Shukla who has helped me in every possible aspect of this project and guided me with his expertise and wisdom. I am also deeply thankful to Prof. Manita Rajput for her incessant readiness to help me and review my work. Her critical feedback on every aspect of my work helps me to improve my work at each and every step.

References

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