HARWARE MONITORING USING ETHERNET LwIP CORE

Tejas Chavan and Rithvik Ballal

Portland State University

ECE 544 – Embedded System Design with FPGA

Final Project

March 2, 2018

INTRODUCTION

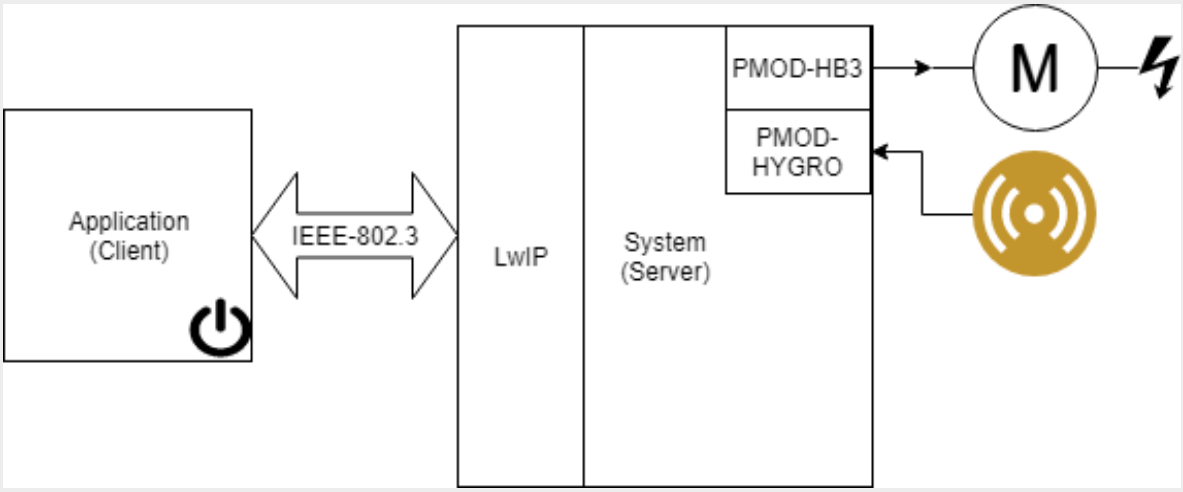
The basic idea behind this project is to remotely monitor a hardware through the network. To implement this we have used an Ethernet LwIP core to establish a point to point connection between the server and Client.

To begin with we have built an Ethernet based hardware to create a server using the microblaze softcore. This hardware includes major ethernet IP cores like AXI EthernetLite and Ethernet PHY MII to Reduced MII (Explained in detail in the further sections).

We have developed a QT based GUI application to create a client system that would ask for connection with the server in order to access a resource (Hardware). This client server relationship is established using the LwIP stack which is the lighter version of TCP/IP stack.

Once the connection is established, packets are transferred between server and client to monitor the hardware, The hardware used here is a motor which is connected to the pmod-HB3 and a pmod-HYGRO temperature and humidity sensor.

BLOCK DIAGRAM



- The block diagram clearly represents an application system which is a client (QT GUI). It communicates with the server model which is a microblaze softcore dumped on the Nexys4 DDR fpga.
- The communication takes place using the LwIP stack.

A. Light Weight Internet Protocol (LwIP)

- LwIP is a small implementation of the TCP/IP suite. The major focus of LwIP is to reduce the RAM usage while still having a full scale TCP. This makes LwIP suitable for use in embedded systems.
- To implement this protocol we have used the RAW API.
- The raw API (sometimes called native API) is an event-driven API designed to be used without an operating system that implements zero-copy send and receive. This API is also used by the core stack for interaction between the various protocols. It is the only API available when running lwIP without an operating system.

B. SERVER (Microblaze)

- The server is built on the Nexys4 DDR Fpga with the microblaze processor. The server follows TCP/IP packet transfer protocol.
- The default IP address of the server is 192.168.1.10 which is used by the client to communicate with the server.
- Upon accepting connection with the client, the server sends and receives packets with the client to access the hardware.

C. CLIENT (QT GUI Application).

- The client here is built using the QT application. A GUI has been created and customized to create a socket, connect to it and eventually connect to the server to access the hardware.
- The GUI has a dial, that can be rotated to increase or decrease the duty cycle of the motor connected to the server. Moreover the client continuously keeps on sending packets to the server to read the temperature and humidity of the environment near the pmod. If the temperature crosses a certain limit, it sets the motor to maximum duty cycle by sending a packet to the server.

D. HARDWARE (pmod-HB3, pmod-HYGRO)

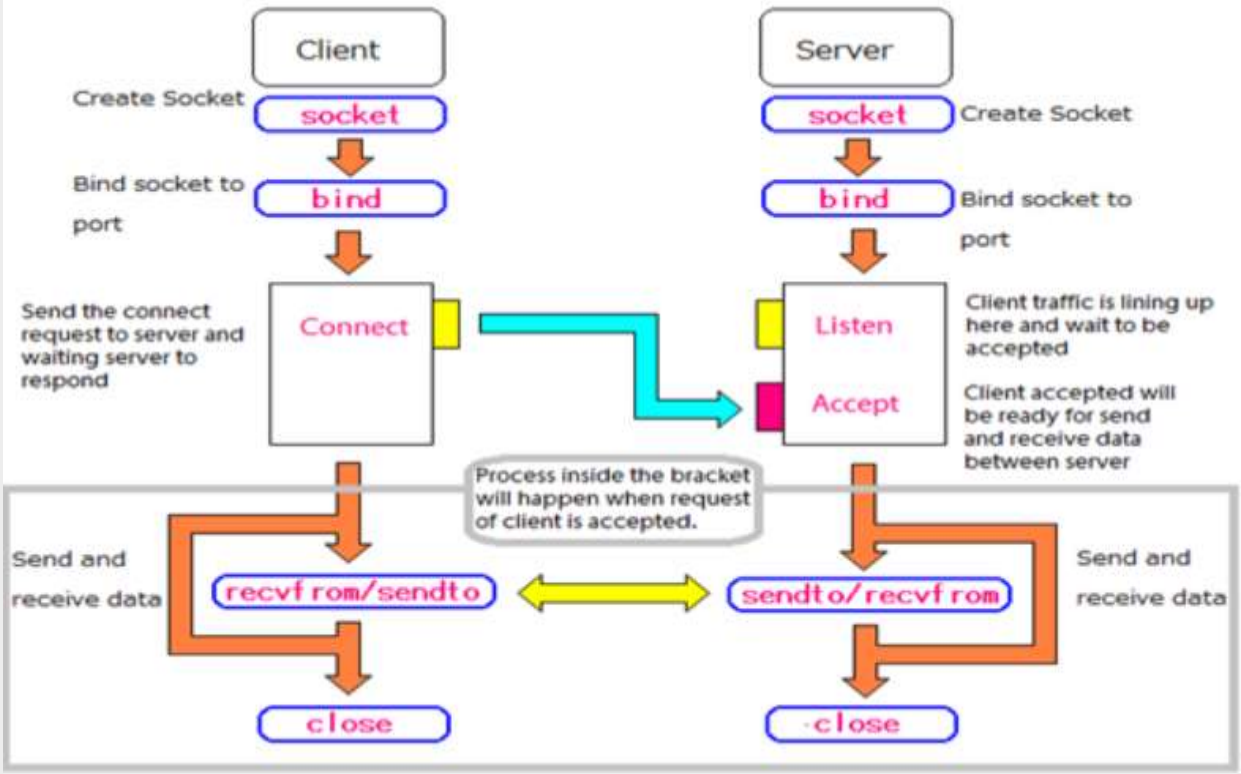
- The hardware that we have used in this system is a 12V computer fan which is driven by a pmod-HB3 motor driver. There's a temperature and humidity sensor pmod-HYGRO.
- The client continuously sends packets to the server requesting for temperature and humidity values.
- Upon receiving the value it displays it on the GUI. Internally it ckecks if the temperature is less than a certain limit (30deg Celsius in our case). If the

temperature crosses this value, the client sends a packet to turn on the fan by setting the duty cycle to 99.

OPERATION:

A. SERVER SIDE:

- The major operation on the server side is performed using **Socket Programming**. The concept involves various steps starting from connect to bind and continuous sending and receiving of packets.



Following are the detail steps involved in socket programming:

- **Socket:** The socket acts as an internal endpoint for sending or receiving data at a single node in a computer network. To create a connection a socket has to be created on both the client and the server side.

  A new TCP connection identifier(protocol control block i.e. PCB) is created with the **tcp_new()** function. This PCB can then be either set to listen for new incoming connections or be explicitly connected to another host

- **Bind** :
  Binds the new PCB to a local IP address and a port number. The IP address can be specified as IP_ADDR_ANY in order to bind the connection to all local IP addresses.

```
err_t
tcp_bind(struct tcp_pcb *pcb, ip_addr_t *ipaddr, u16_t port) ;
```

- **Listen**: Commands a PCB to start listening for incoming connections. When an incoming connection is accepted, a function specified with the tcp_accept() function will be called.
  The PCB will have to be bound to a local port with the tcp_bind() function.

```
tcp_listen(pcb);
```

- **Accept**: The client that is accepted will be able to send and receive data with the server. Here a call back function is specified, that should be called when a listening has been connected to another host.

```
void
tcp_accept(struct tcp_pcb *pcb, tcp_accept_fn accept);
```

- **Send and Receive:** This includes a set of functions that are executed continuously, for sending and receiving data between the client and the server. Initially the client needs to send a request packet to the server with a command stating that it needs to access a resource.
  The server receives the packet using the API :

```
void
tcp_recv(struct tcp_pcb *pcb, tcp_recv_fn recv);
```

  Once the packet has been received, the server write to the client with a data using the API call:

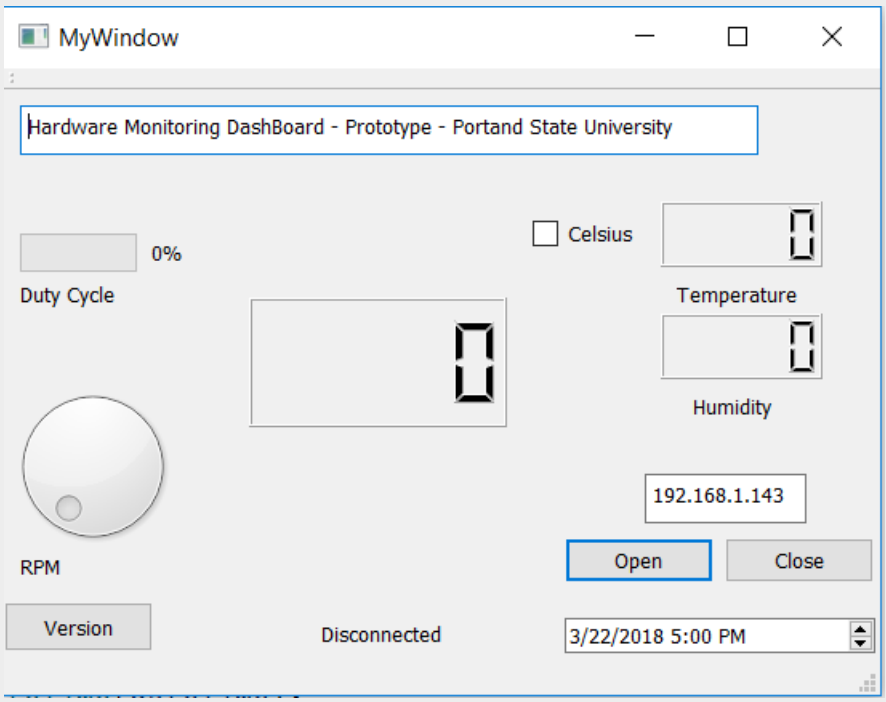
```
err_t
tcp_write(struct tcp_pcb *pcb, const void *arg, u16_t len, u8_t apiflags);
```

The server never sends the packet immediatety, rather waits in the expectation of more data being sent soon. The packet can be sent instantaneously using the API call:

```
err_t
tcp_output(struct tcp_pcb *pcb);
```

B. CLIENT SIDE :

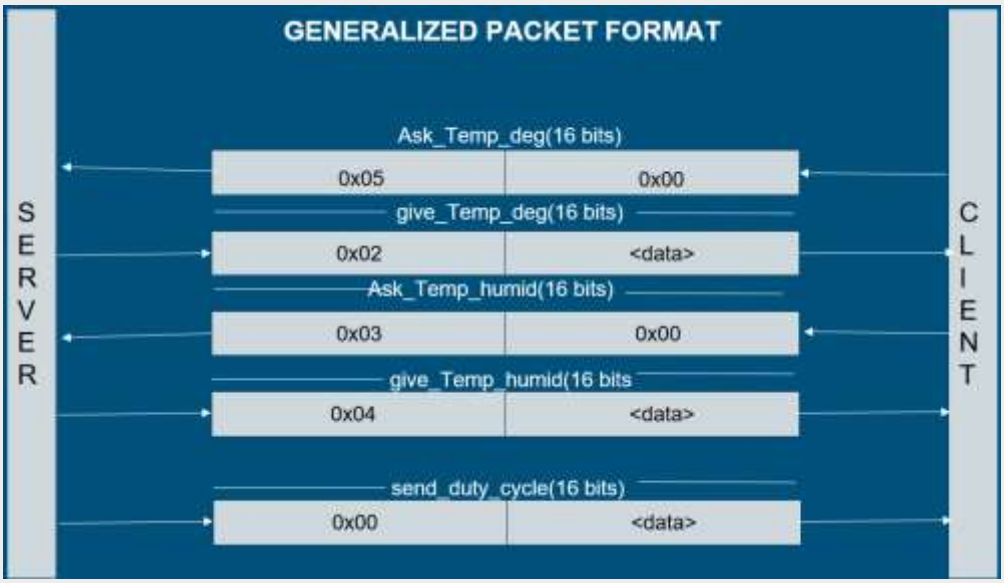
- The Client side coding and GUI was done in QT Creator using C++.



- The above figure is the GUI built to communicate with server.
- The communication between the GUI and the backend code(C++) is done using Signal and Slots technique.
- The Signal is kind of a function which is emitted when an event(like pressing a open button in the above figure) occurs and Slot is the back-end function(methods in C++) which gets called when the corresponding signal is emitted.
- For example: When an open button is pressed, **accepted()** signal is emitted.The corresponding **connectToServer()** slot is called, which will try to connect to the server using the ip_address in the dialog box above the open button.
- The connection between the client and a server is established using the **QTcpSocket::connectToHost()** function, which will take ip_address and the port number has its argument.
- The function **QTcpSocket::waitForConnected()** with time the socket will wait for connection to be establish. If the connection is not established by that time, then GUI will throw an error stating the “Connection cannot be established due to time-out”.

- The data is transmitted from the client to the server using **QTcpSocket::write()** function.
- The acknowledgement and the data from the server is got using **QTcpSocket::readAll()** function.
- The **QTcpSocket::bytesAvailable()** function is used to check whether there is any data to be read from the socket.
- A Threading technique is used to continuously polling for temperature and humidity changes from the server.
- The Thread is started only when the connection is establish. The thread is started when the run method of the thread is called.
- The Thread is terminated when the connection is lost.
- The polling of the Thread is used to check the temperature change. If the temperature crosses 30 degrees, a fan-on command is sent through the socket to the server so that the fan can be on as a result, temperature can be reduced.

PACKET FORMAT



- The packet format is generalized as a 16 bit array, with the 8 MSBs allotted for Command and the 8 LSBs are given for Data.
- Initially when the Client gets connected with the server, it continuously requests for temperature and humidity values by sending request packets.
  Here the request for temperature in degree is made by sending a packet with command as 0X05 and data as 0. Similarly, it sends a request packet for humidity

with command as 0x03 and data as 0. This request is made at regular intervals of 1 sec.

- Upon receiving the request packets the server calls the temperature and humidity APIs and writes the data to the client using a packet with command as 0x02 for temperature and 0x04 for humidity respectively.
- Upon receiving the packets, the client checks if the temperature is greater than a set value (30degree Celsius) in our case.
  If the temperature is higher, then it sends a packet with command as 0x00 and data with duty cycle information to the server.
- When the server receives the packet, it decodes it and by identifying the command as 0x00, sets the duty cycle of the motor.

CHALLENGES FACED

- Understanding the QT Application was a huge challenge as we were new to the application and it functioning.
- It took a while to figure out the data conversion between Unicode to ASCII. The QT sends data in the dorm of Unicode which cannot be decode in the microblaze as it works on ASCII values. Thus the unicode data had to be converted into ASCII by using a few functions and libraries in QT and then send it to the server.
- Implementing threading functionality in QT took a lot of efforts. The continuous request for temperature and humidity had to done in background using threading model which was not as easy to implement as freeRTOS. But in the end we pulled it off by emitting a flag when ever the client wanted to send a packet for duty cycle.

INDIVISUAL CONTRIBUTIONS:

Rithvik Ballal :

- Worked on the Client side of the system.
- Designed and implemented the QT based GUI application using C++.
- Performed Data conversion from Unicode to ASCII . Implemented a threading model to send continuous packets to the server.

Tejas Chavan:

- Worked on the Server side of the system.
- Built the hardware using various ethernet IPs on microblaze softcore.
- Performed socket programming on the server side using various APIs for LwIP ethernet protocol. Configure the pmod-HYGRO and HB3 to access the temperature and set duty cycle.

FUTURE SCOPE

- The major optimization would be to build a web server to create a multi-point communication system and build multiple clients to access the resource on the server.
- Integrate a wifi module and maybe involve a bigger hardware system like a robot involving a stepper motor.

CONCLUSION

- Ethernet is one of the major and fastest means of data transfer for long distance communication.
- Its high throughput characteristic can be used in controlling and monitoring hardware's and machineries located at remote location. As LwIP is reliable considering its request-acknowledgement protocol, it leads to secure data transfer without much packet loss.

REFERENCES

- http://www.nongnu.org/lwip/2_0_x/index.html
  This is a major source of all informations related to LwIP and its APIs

- https://reference.digilentinc.com/learn/programmable-logic/tutorials/arty-getting-started-with-microblaze-servers/start
  This link helped in getting started with the hardware for the server side.

- https://forum.qt.io/
  QT forum for application building