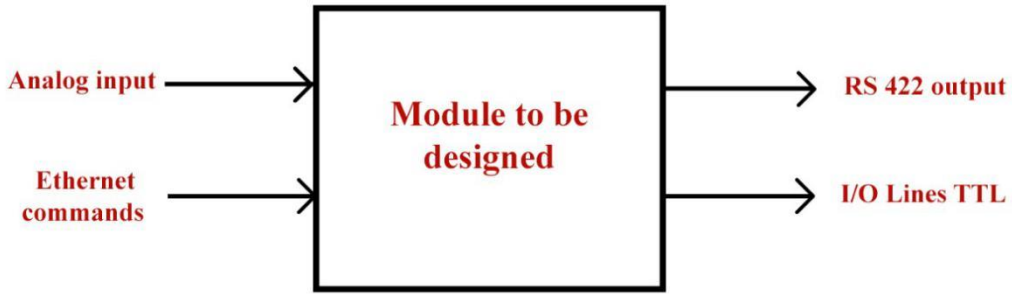
COMMUNICATION MODULE

Department of Electronics and Communication Engineering

Vasavi College of Engineering, Hyderabad

ABSTRACT

A module should be designed that takes analog inputs from sensors, Ethernet commands from PLC and produces required outputs in the form of TTL logic for former input and serial output for latter input. The proposed solution is realized using Raspberry Pis as central parts of module which is equipped with other peripherals to behave as a complete working module. The module is intended to be used in submarines to communicate between PLCs and antennas and also drive large relays in response to sensor data.



NOTES:
1) Analog inputs will be from tentatively from temperature and pressure sensors's.
2) COTS solution is preferable.

APPROACHES

1. An Arduino(Micro controller) based solution was proposed earlier. It has some problems. Arduino boards don’t have Ethernet compatible port so it requires external Ethernet shield. In order to get the serial output(rs232/422) the smart boards have to be stacked on the Arduino which requires PHPOC shield to be attached externally. This makes the hardware more complex. Even after completion of module(COTS) , Arduino could not be compatible with further enhancements of the product.

2. A Raspberry Pi(SoC) based solution. It has a built-in Ethernet port, several GPIO pins. It(python) has also got huge library to make the code simple and efficient compared to the C/C++ based Arduino software. There are many serial output shields available that can be directly stacked upon the Raspberry Pi. Further, any modification in future could be easily made. These features made us to proceed with the Raspberry Pi based solution.

COMPONENTS USED

The following are the components of the module designed:

1. Raspberry Pi 3 Model B+ (2 Nos)

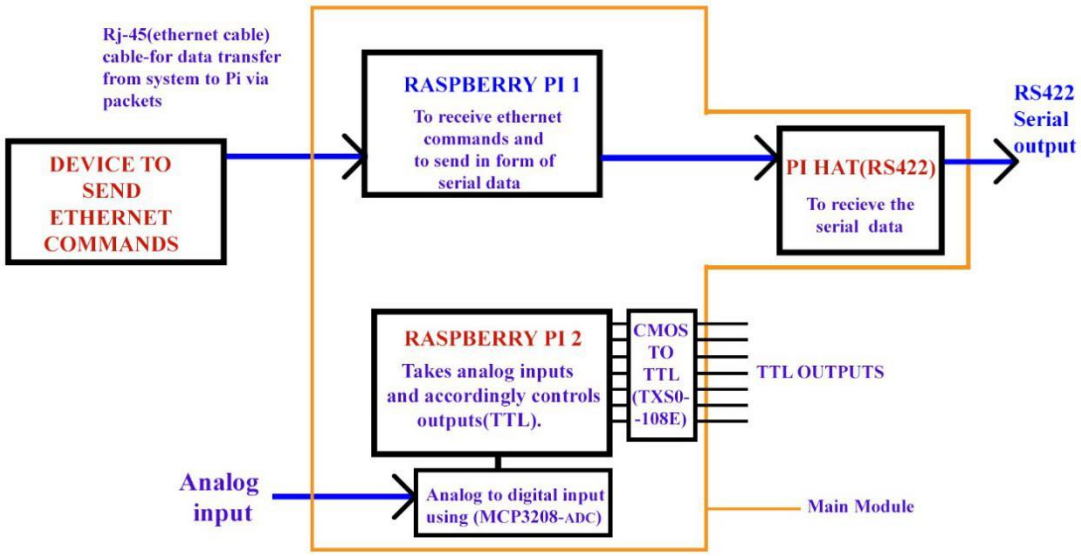
2. MCP3208-Analog to Digital Converter

3. Isolated RS485/RS422 Raspberry Pi HAT

4. CMOS to TTL(TXS0108E) high speed full duplex 8 channel logic level converter

METHOD

Fig-1



The entire procedure is divided into 2 parts. Each Raspberry Pi acts as the central part in each sub module.

PART-1: Design of the first sub-module

The Raspberry Pi-1 here receives Ethernet commands as inputs through an Ethernet cable(Rj-45). It is programmed to convert the commands and send them to a slave device(using Serial Communication) utilizing Pi-HAT. The Pi-HAT is fixed to Raspberry Pi-1 and using this the slave device receives the corresponding serial data to perform necessary actions as per the instructions. The Pi-HAT converts full duplex CMOS to full duplexRS422(4 wire) or half to duplex RS485(2 wire) using transceiver switching. It has the feature of adjustable controlling a transmitter/receiver via GPIO pin.

The communication between PLC and Raspberry Pi-1 is setup by Socket programming. The PLC acts as server and Raspberry Pi-1 acts as client. The code used for server corresponds to code-1, similarly code-2 is used for Raspberry Pi-1.

CODE-1

```
import socket


serv = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # creating server object of socket type
serv.bind(('169.254.199.57',5005))
serv.listen(999)
string=''
while True:
    print("\n in loop ")
    conn, addr = serv.accept() #accepts the connection from client
    from_client = ''
    data = conn.recv(4096) #max no.of bytes
    data=data.decode('utf-8') #decoding the bytes information encoded in utf-8 uni-code
    from_client += data
    print (from_client)
    choice=input("\n1)To Enter the command\n2)To exit\t") #Menu driven, user specific
    if(choice=='1'):
        string=input()
        msg=bytes(string,'utf-8')#UTF-8variable width character encoding in Unicode
        conn.send(msg)
    elif(choice=='2'):
        string = '%#%#%#'          # a random flag to understand the termination
        msg=bytes(string,'utf-8')
        conn.send(msg)
        conn.close()
        break
    else:
        print("invalid input")
print ('client disconnected')
```

CODE-2

```
import time
import serial
import socket
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)
GPIO.setup(11,GPIO.OUT)

#To establish serial connection
ser = serial.Serial(port ='/dev/serial0', baudrate = 9600 , parity = serial.PARITY_NONE, stopbits =
serial.STOPBITS_ONE, bytesize = serial.EIGHTBITS, timeout = 1)
temp=''
while True:
        client = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        client.connect(('169.254.199.57',5005)) #Server IP adress
        string = "Receiver is active"
        msg = bytes(string,'utf-8')
        client.send(msg)
        from_server = client.recv(4096)
        from_server = from_server.decode('utf-8')
        if(from_server=='%#%#%#'):
                client.close()
                close1='%#%#%#'
                close2=close1.encode('utf-8')
                ser.write(close2)
                ser.close()
                break
        else:
```

```
if(from_server=='onLed'):
        GPIO.output(11,GPIO.HIGH)       #Testing the TTL output using Led
elif(from_server=='offLed'):
        GPIO.output(11,GPIO.LOW)
from_server=from_server+'$' #'$' is just used to detect last byte of data
msgInBytes = from_server.encode('utf-8')
ser.write(msgInBytes) #sending serial data
mread=ser.read(1) #recieving from other side serially in bytes
temp=mread.decode('utf-8')
if(temp=='y'):
        print('RECIEVED!')
        continue
#time.sleep(0.13)
```

PART-2: Design of the second sub-module

The Raspberry Pi-2 takes analog data from sensors and since the GPIOs are digital pins, an Analog to Digital Converter(MCP 3208) is employed. MCP3208 works through SPI protocol and hence it has high data transfer rate. To summarize, Raspberry Pi-2 is programmed to receive analog signals and respond accordingly. The code-3 corresponds to this function.

CODE-3

```
import spidev
import time
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)
GPIO.setup(11,GPIO.OUT)
```

```
spi=spidev.SpiDev()
spi.open(0,0)
spi.max_speed_hz=1000000

MAX_CH= 8
adc_values = [0 for i in range(MAX_CH) ]


def analog_read(ch):
        if( (ch>MAX_CH-1) or (ch<0)):
                return -1
        r = spi.xfer2([1, (8+ch)<<4, 0])
        val = ((r[1]&3)<<8)+r[2]
        return val


while True:
  try:
          time.sleep(0.1)
          ch = 0
          GPIO.output(11,GPIO.LOW)
          #for ch in range(0,7):
          adc_values[ch] = analog_read(ch)
          if (adc_values[ch]>95):
                  GPIO.output(11,GPIO.HIGH)
          print(adc_values)
          break
  except KeyboardInterrupt:
                break
print("done")
```

TESTING

Additional components used: Raspberry Pi 3 Model B+, LM35 Sensor.

Testing sub-module 1:

Since we don't have access to Pi-HAT compatible slave device, the conversion of Ethernet commands to serial data is verified using another Raspberry Pi(test) at the serial output end . A personal desktop computer is used to send the Ethernet commands. The Tx and Rx pins of the Raspberry Pi-1 are connected to Rx and Tx pins of the test Raspberry Pi respectively as shown in Fig-2. An LED is set up to test the TTL output. The Raspberry Pi's GPIOs give CMOS level logic and hence a CMOS to TTL converter(TXS0108E) is used to which the LED is connected. The submodule-1 is shown in Fig-3,4.

Fig-2:

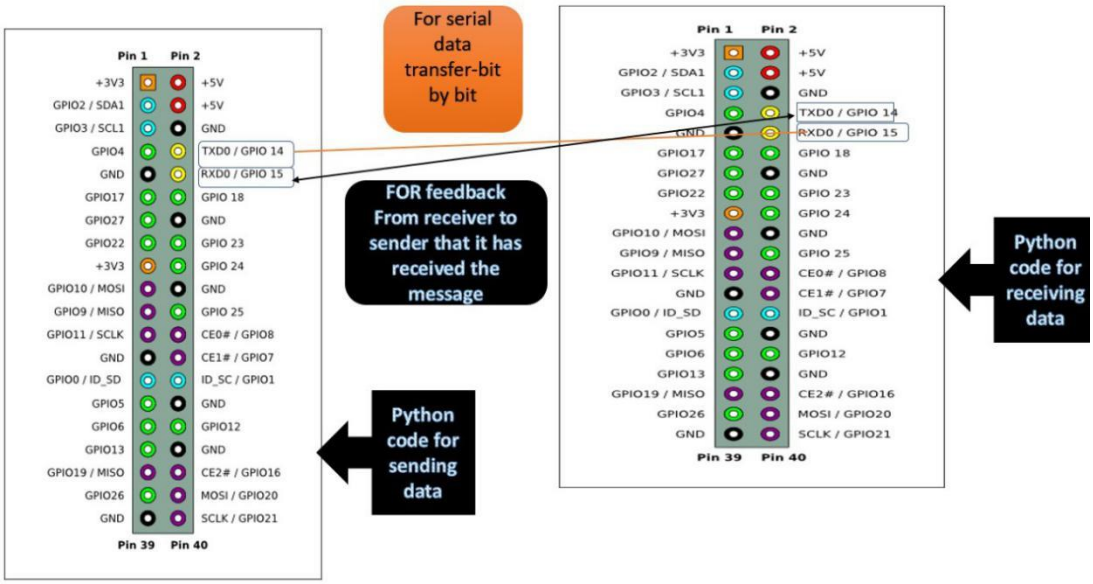


Fig-3:

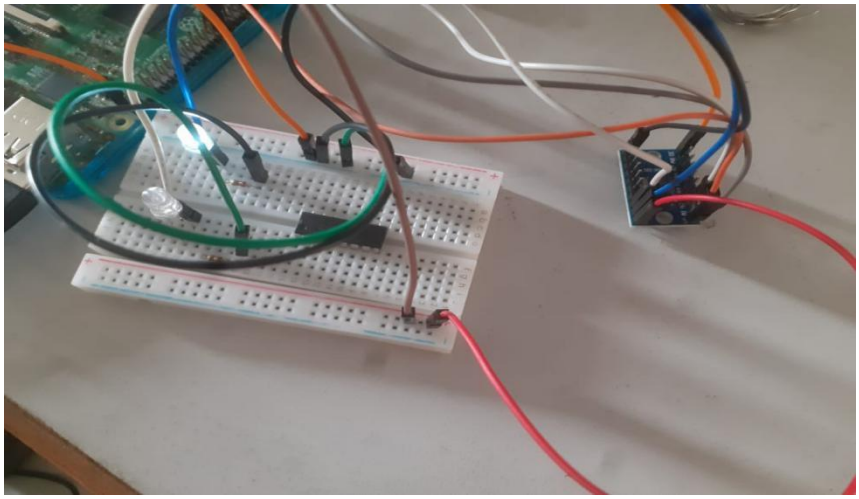
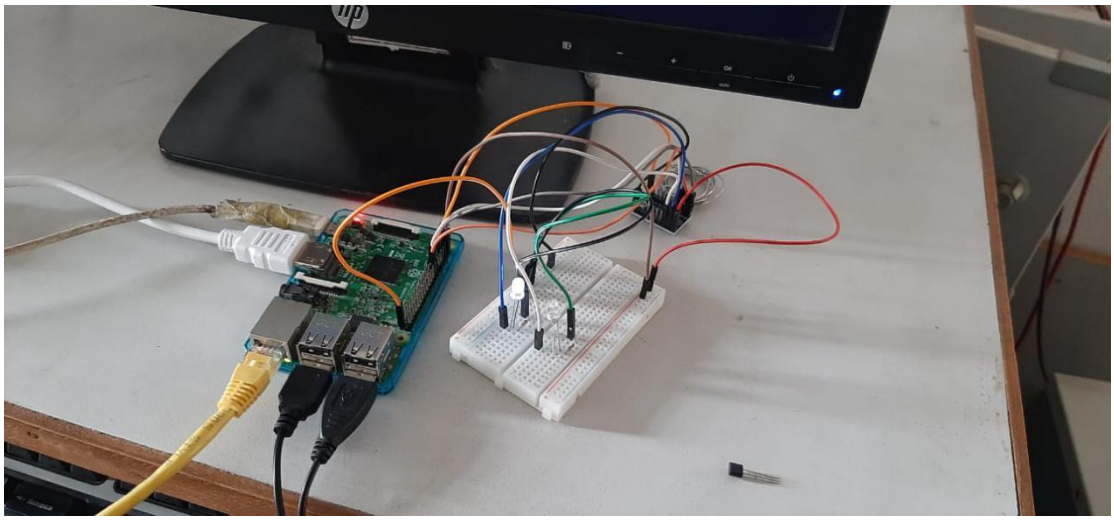


Fig-4:



To check the serial output at receiver's end, the test Raspberry Pi is programmed with the following code:

CODE-4

```
import time

import serial

ser=serial.Serial(port='/dev/serial0',baudrate=9600,parity=serial.PARITY_NONE,stopbits =

serial.STOPBITS_ONE, bytesize = serial.EIGHTBITS, timeout = 1)

)

data2=''

temp=''

t2=''
```

```
t1=''
while 1:
    data=ser.read() #reading from otherside serially in bytes
    temp=data.decode('utf-8') #decoding into a string
    if(temp=='$')
        print(data2)
        t1='y'
        t2=t1.encode('utf-8')
        sedr.write(t2) #sending feedback to mediator
        continue
    data2+=temp
    if(data2=='%#%#%#')
        print('connection is closed')
        ser.close()
        break
```

Testing sub-module 2:

The sub-module 2 is tested with LM-35(Temperature) sensor for analog input. The readings are displayed on a monitor and an LED is connected which glows when the temperature raises beyond the threshold set in the CODE-3. The connections between LM35 sensor and ADC are shown in Fig-5. The sub-module 2 is shown in Fig-6,7.

Fig-5:

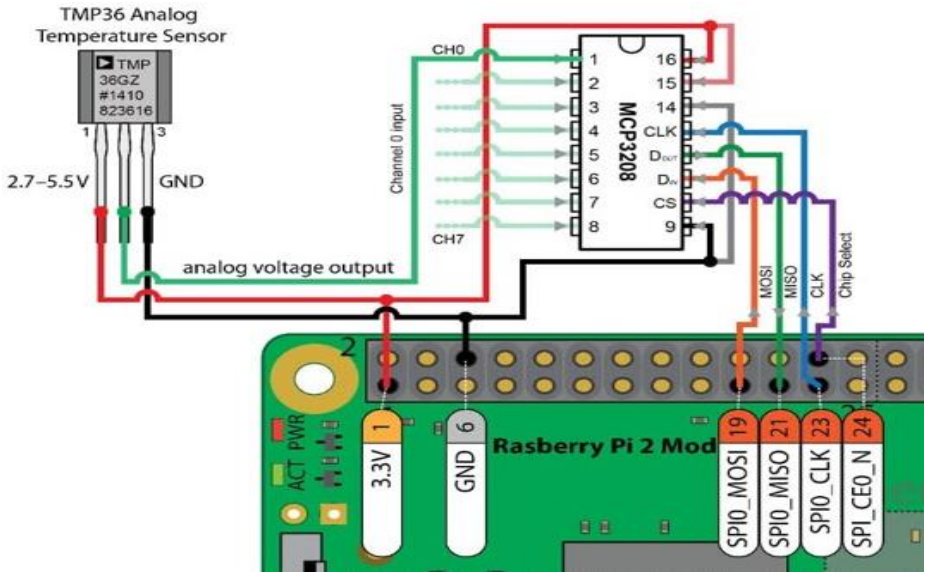

TMP36 Analog
Temperature Sensor
TMP
36GZ
#1410
823616
1
3
2.7–5.5V
GND
analog voltage output
Channel 0 input
CH0
CH7
MCP3208
1
2
3
4
5
6
7
8
16
15
14
CLK
CS
9
MOSI
MISO
CLK
Chip Select
ACT PWR
3.3V 1
GND 6
Rasberry Pi 2 Mod
SPI0_MOSI 19
SPI0_MISO 21
SPI0_CLK 23
SPI_CE0_N 24

Fig-6:

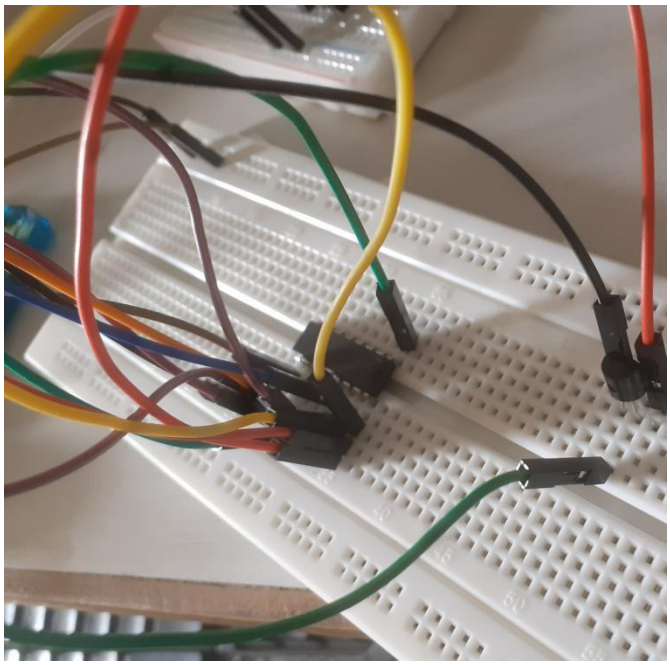
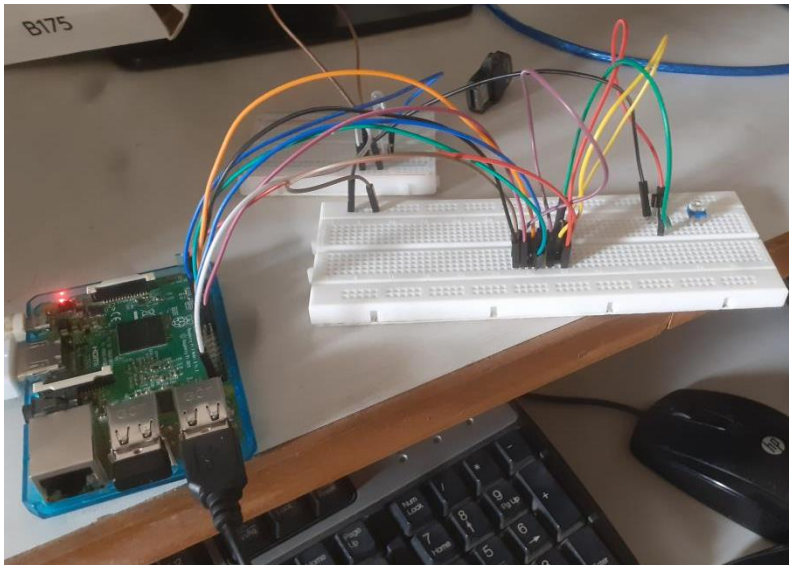


Fig-7:



Future Developments:

1. Both sub-modules can be integrated using multi-threading and instead of stacking the Pi-HAT, the female header of Pi-HAT can be dissolved and only the required pins should be connected which in turn saves the GPIO pins of the central Raspberry Pi.

2. Developing a code which works well with any kind of Analog sensor.