Name: Kamlesh Khagad SR.No 17965, DESE, IISc
Name: Pranjal Jaiswal SR.No 17996, DESE, IISc
Name: Sumit Kumar SR.No 17896, DESE, IISc
Course: E3-257, Embedded System Design

Project Report: CAN bus communication with Sensor hub

1 Objective

The main objective of the project is to establish a working CAN link between a "client" and a "server". The user interacts with client and requests data from server via CAN bus. The server measure the data from the Sensor hub and sends it to client via the same can bus.

2 CAN Protocol

2.1 Introduction

The Controller Area Network (CAN) is a serial communication bus designed for robust and flexible performance in harsh environments, and particularly for industrial and automotive applications.

Originally invented by Bosch and later codified into the ISO11898-1 standard, CAN defines the data link and physical layer of the Open Systems Interconnection (OSI) model, providing a low-level networking solution for high-speed in-vehicle communications. In particular, CAN was developed to reduce cable wiring, so the separate electronic control units (ECUs) inside a vehicle could communicate with only a single pair of wires.[1]

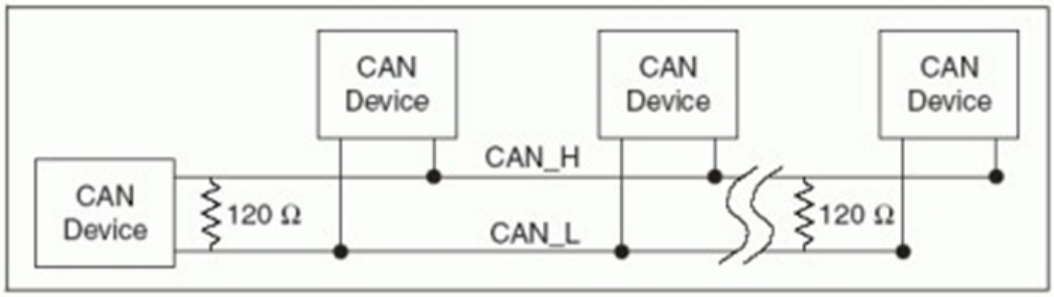

Figure 1: CAN BUS

2.2 CAN message Frame

The original ISO standard laid out what is called Standard CAN. Standard CAN uses an 11-bit identifier for different messages, which comes to a total of 211, i.e. 2048, different message IDs. CAN was later modified; the identifier was expanded to 29 bits, giving 229 identifiers. This is called Extended CAN. CAN uses a multi-master bus, where all messages are broadcast on the entire network. The identifiers provide a message priority for arbitration.

The TM4C123GH6PM CAN controller conforms to the CAN protocol version 2.0 (parts A and B). Message transfers that include data, remote, error, and overload frames with an 11-bit identifier (standard) or a 29-bit identifier (extended) are supported. Transfer rates can be programmed up to 1 Mbps.

Name: Kamlesh Khagad SR.No 17965, DESE, IISc
Name: Pranjal Jaiswal SR.No 17996, DESE, IISc
Name: Sumit Kumar SR.No 17896, DESE, IISc
Course: E3-257, Embedded System Design

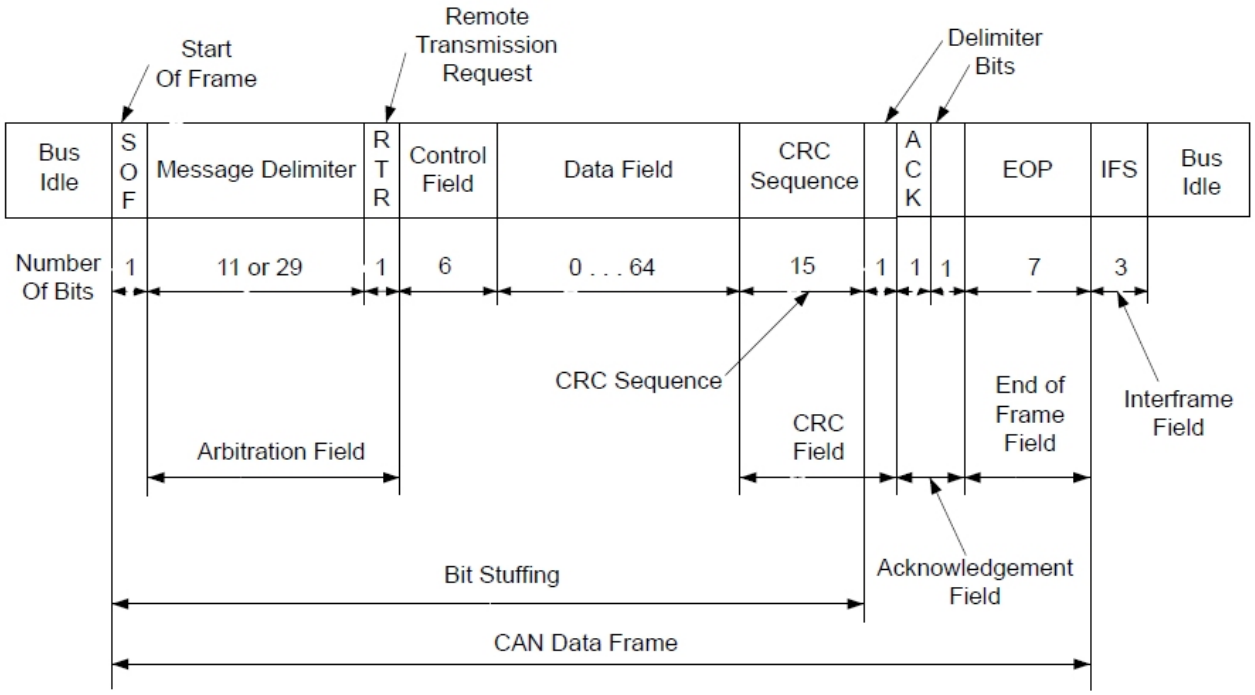

Figure 2: CAN frame [2]

2.3 CAN message Types

1. **Data Frame :** A standard CAN data frame makes use of the identifier, the data, and data length code, the cyclic redundancy check, and the acknowledgement bits. Both the RTR and IDE bits are dominant in data frames.

2. **Remote Transmission Request :** A CAN remote frame looks similar to a data frame except for the fact that it does not contain any data. It is sent with the RTR bit in a recessive state; this indicates that it is a remote frame. Remote frames are used to request data from a node.

3. **Error Frame :** When a node detects an error in a message on the CAN bus, it transmits an error frame. This results in all other nodes sending an error frame. Following this, the node where the error occurred retransmits the message.

4. **Overload Frame :** The overload frame works similar to Error Frame but is used when a node is receiving frames faster than it can process them. This frame provides a time buffer so the node can catch up.

2.4 Bandwidths

Bandwidth	Bus Length
1 Mbits/sec	40 m
500 kbits/sec	100 m
100 kbits/sec	500 m
50 kbits/sec	1000 m

Name: Kamlesh Khagad SR.No 17965, DESE, IISc
Name: Pranjal Jaiswal SR.No 17996, DESE, IISc
Name: Sumit Kumar SR.No 17896, DESE, IISc
Course: E3-257, Embedded System Design

3 Components Used

3.1 List of Components

1. Tiva C Series TM4C123GXL LaunchPad Boards – 2 nos
2. BOOSTXL-SENSHUB Sensor Hub BoosterPack – 1 nos
3. MCP 2551, Can bus transceiver boards – 2nos
4. Jumper wires and two 120 ohm resistors for CAN BUS
5. 5V Power Supply to supply MCP 2551
6. Two LAPTOPS to act as server and client

3.2 Images of Components

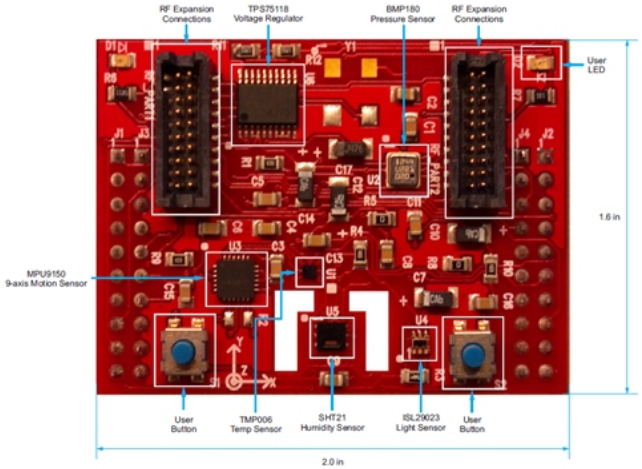

Figure 3: BOOSTXL-SENSHUB Sensor HUB [3]

Figure 4: MCP 2551 CAN transceiver [4]

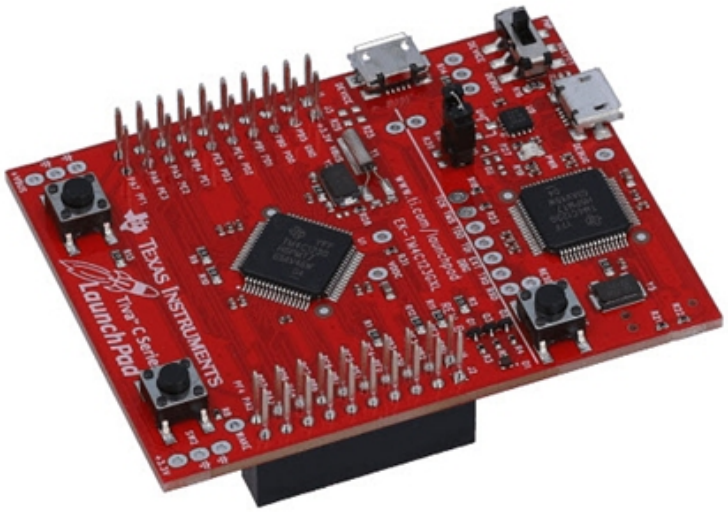

Figure 5: Tiva C Series TM4C123GXL LaunchPad [2]

Name: Kamlesh Khagad SR.No 17965, DESE, IISc
Name: Pranjal Jaiswal SR.No 17996, DESE, IISc
Name: Sumit Kumar SR.No 17896, DESE, IISc
Course: E3-257, Embedded System Design

4 Sensors Used

The Sensor Hub comes with a variety of sensors, each with their own library and functions. But to show CAN functionality we chose only two of them -

1. **BMP 180 Pressure sensor :** this gives both pressure and temperature
2. **ISL 29023 light sensor :** to give intensity reading

These readings are transmitter from sensor hub to "server" via I2C communication with the api functions of respective sensors.

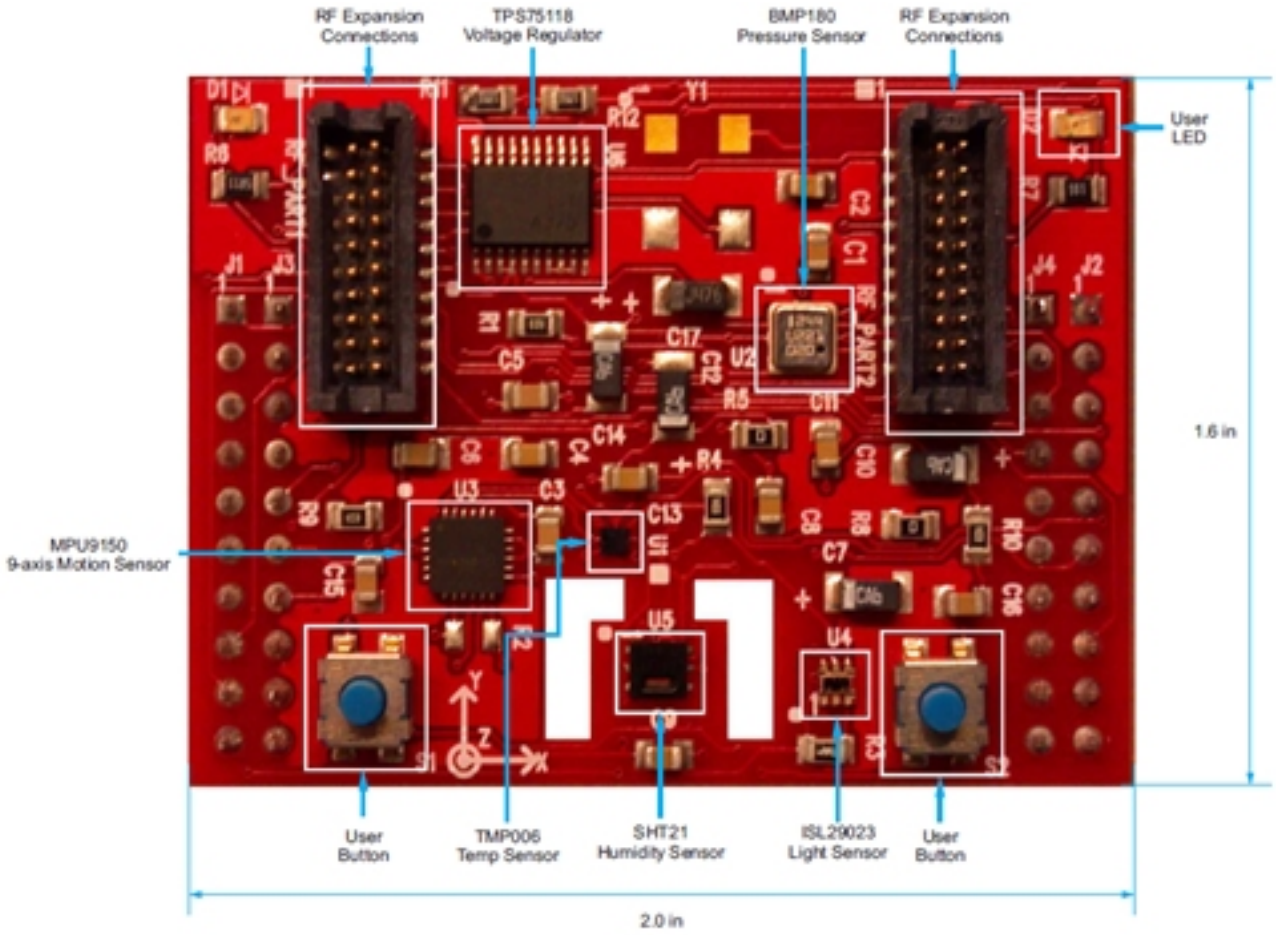

Figure 6: Enlarged BOOSTXL-SENSHUB Sensor HUB [3]

Name: Kamlesh Khagad SR.No 17965, DESE, IISc
Name: Pranjal Jaiswal SR.No 17996, DESE, IISc
Name: Sumit Kumar SR.No 17896, DESE, IISc
Course: E3-257, Embedded System Design

5 Block Diagram

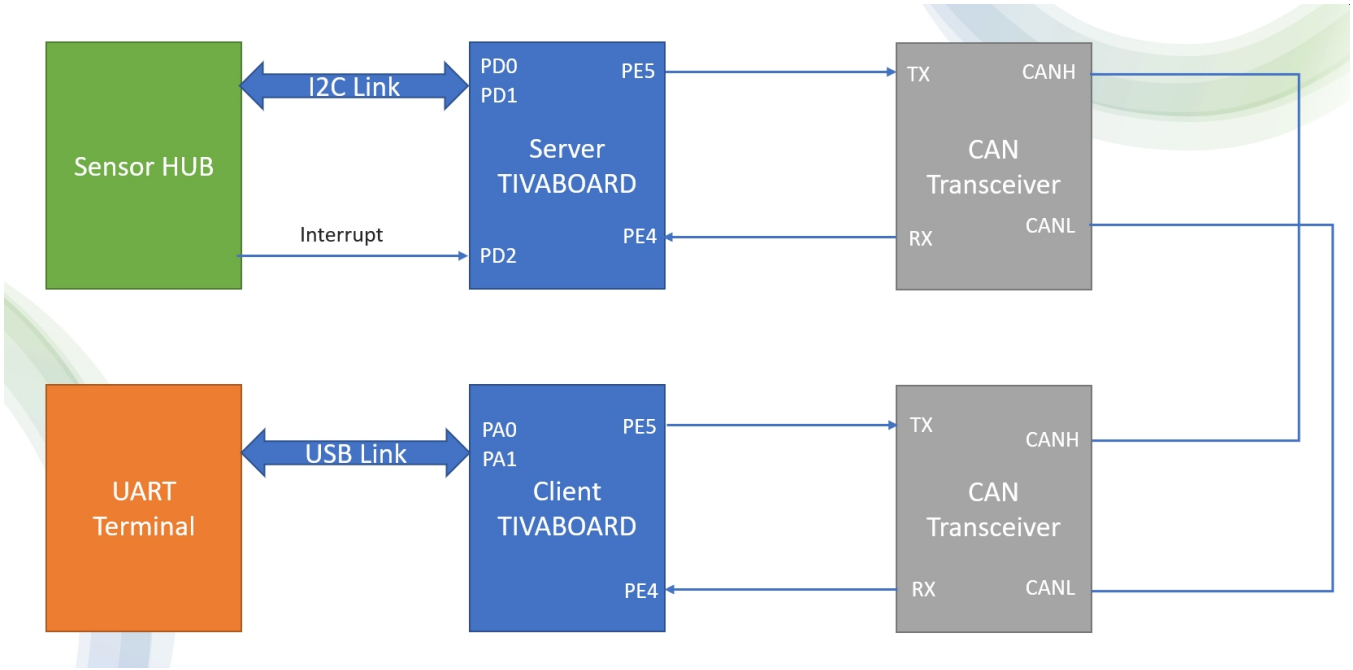

Figure 7: Project Block Diagram

5.1 Description of Block Diagram

1. Sensor Hub is connected to "server" board via I2C link to pins PD0 (SCL) and PD1 (SDA). Also an interrupt trigger is connected from sensor hub to pin PD2
2. Server is connected to server side transceiver with pins PE5 and PE4 as Tx and Rx respectively. same connection is also made at lient end.
3. CANH and CANL are connected directly on breadboard with 120 Ω termination resistances.
4. We can see interact with client via UART terminal from our PC.

5.2 Working

1. The basic idea of this project is to implement a working CAN bus communication.
2. Sensor Hub to gathers the data.
3. This data is then transferred to "server" at regular intervals using I2C communication and Sensor API calls.
4. User interacts with "client" and requests for data, E.g. Temperature, Pressure or Illumination.
5. Request is sent via CAN bus, the latest reading is then replied by the server over CAN bus.
6. The received data is then displayed on UART terminal.

Name: Kamlesh Khagad SR.No 17965, DESE, IISc
Name: Pranjal Jaiswal SR.No 17996, DESE, IISc
Name: Sumit Kumar SR.No 17896, DESE, IISc
Course: E3-257, Embedded System Design

6 CAN packet format

Using the CAN library we are sending/receiving 64 bits of data at time i.e. 8 bytes a a time

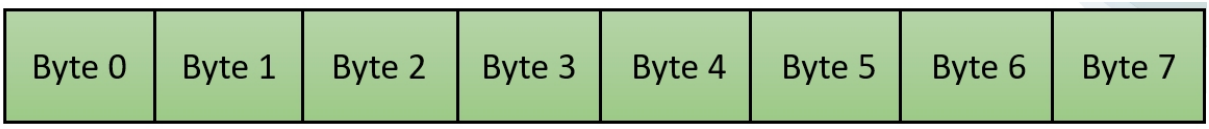

Figure 8: Packet Format

Request Packet (from client to server) –

1. Byte 0 = “0” always
2. Byte 1 = “1” - temperature, “2” - pressure, “3” - Light Intensity
3. Byte 2 - 7 = for future expansion

Reply Packet (from server to client) –

1. Byte 0 = “1” - temperature, “2” - pressure, “3” - Light Intensity
2. Byte 1,2 = Fractional part of reading
3. Byte 3 and 4 = Integer part of reading
4. Byte 5-7 = for future expansion

Name: Kamlesh Khagad SR.No 17965, DESE, IISc
Name: Pranjal Jaiswal SR.No 17996, DESE, IISc
Name: Sumit Kumar SR.No 17896, DESE, IISc
Course: E3-257, Embedded System Design

7 Finished Project Images

7.1 Circuit

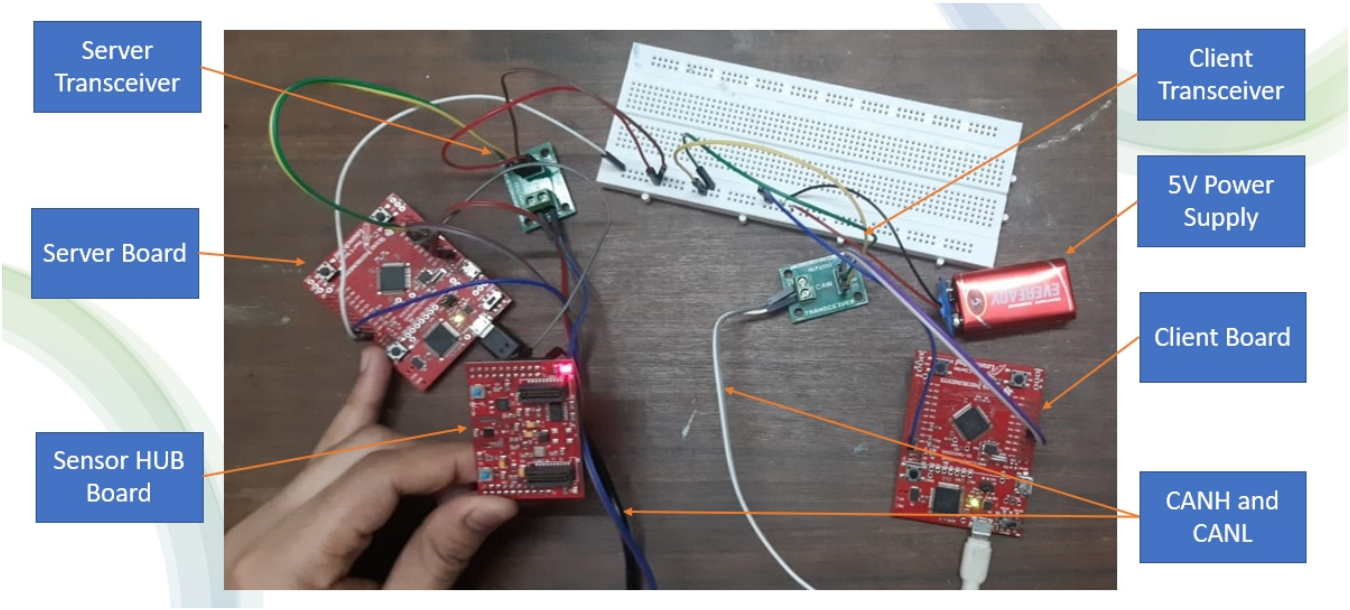

Figure 9: Circuit with Labelling

7.2 Terminal View

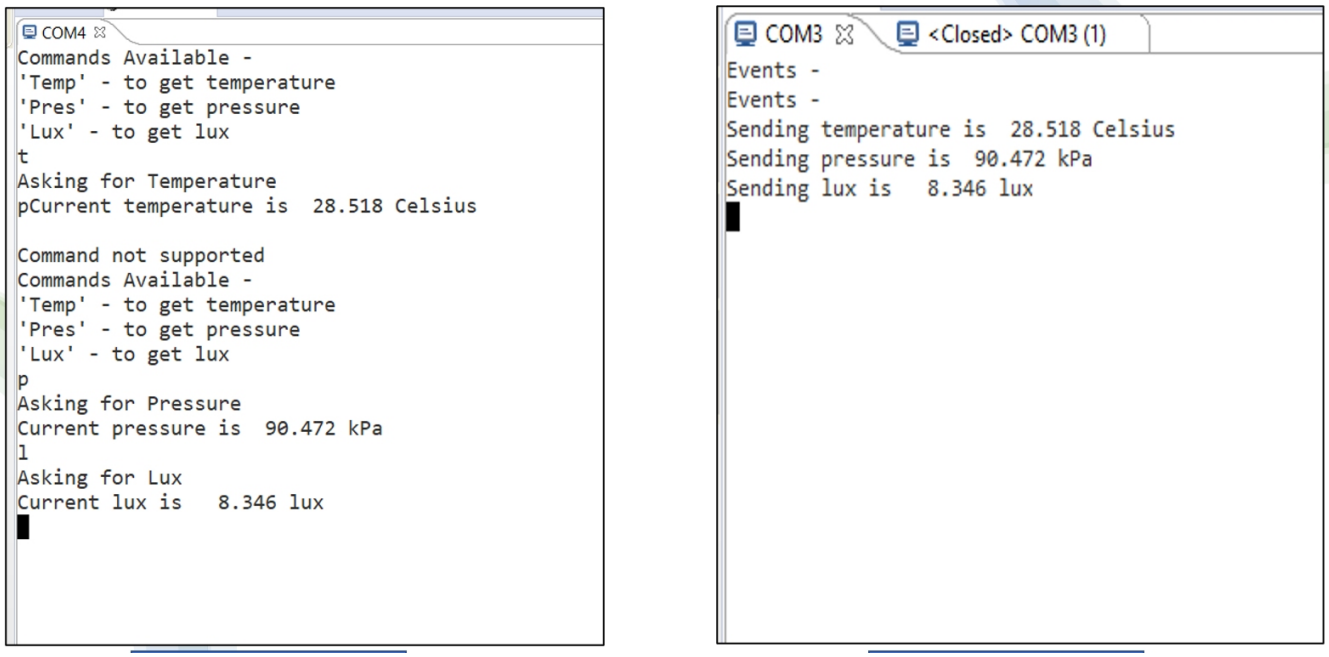

Figure 10: Terminal Images Client side(left) and Server side (right)

Name: Kamlesh Khagad SR.No 17965, DESE, IISc
Name: Pranjal Jaiswal SR.No 17996, DESE, IISc
Name: Sumit Kumar SR.No 17896, DESE, IISc
Course: E3-257, Embedded System Design

A Objects and Instances

A.1 master_heah.h

1. tCANMsgObject sCANRxMessage - Receive CAN object
2. tCANMsgObject sCANTxMessageTransmit CAN object

A.2 sensor_head.h

1. tI2CMInstance g_sI2CInst - Global instance structure of I2C master driver
2. tBMP180 g_sBMP180Inst - Global instance structure for BMP180 Driver
3. tISL29023 g_sISL29023Inst - Global instance structure for ISL29023 Driver

B Function Definitions

B.1 master_heah.h

1. InitConsole(void)- Initialises UART console
2. SimpleDelay(void) - Provides Delay of 1 sec
3. InitCanbus(void) - Initialises CAN bus communication
4. send_can(char str[10]) - Function to make and send CAN frames
5. receive_can(void) - Function o receive can frame
6. do_branch(char str[10]) - Funcion to handle uart input
7. print_reading(void) - Print the received reading
8. send_reading(void) - Send the reading
9. menu(void) - To print menu
10. CANIntHandler(void) - Can interrupt handler
11. Uart_Handler(void) - Uart interrupt handler

B.2 sensor_head.h

1. BMP180AppCallback(void* pvCallbackData, uint_fast8_t ui8Status) - Callback function BMP180
2. InitI2C(void) - Initialise I2C communication and sensors
3. get_temp_pres(void) - To measure temp and pressure
4. ISL29023AppCallback(void *pvCallbackData, uint_fast8_t ui8Status) - Callback function ISL29023
5. ISL29023AppI2CWait(char *pcFilename, uint_fast32_t ui32Line) - Wait till I2C transaction is complete
6. ISL29023AppAdjustRange(tISL29023 *pInst) - Adjust range of LUX
7. get_lux(void) - Function to meaure latest LUX

8. BMP180I2CIntHandler(void) - I2C interrupt handler
9. SysTickIntHandler(void) - Systick handler
10. IntGPIOd(void) - Port D interrupt handler

C Libraries to be linked

The following libraries are need to be included other than the ones included by default

1. inc/hw_can.h
2. inc/hw_ints.h
3. inc/hw_types.h
4. inc/hw_memmap.h
5. driverlib/can.h
6. driverlib/gpio.h
7. driverlib/interrupt.h
8. driverlib/pin_map.h
9. driverlib/sysctl.h
10. driverlib/uart.h
11. utils/uartstdio.h
12. sensorlib/i2cm_drv.h
13. sensorlib/hw_bmp180.h
14. sensorlib/bmp180.h
15. sensorlib/hw_isl29023.h
16. sensorlib/isl29023.h

D Tips

1. We have used the code from examples of TIVA compiler but some of the code didn't run correctly so we have modified it.
2. The CAN example can be found in "examples/peripherals/can"
3. sensor-hub example can be found in "examples/boards/ek-tm4c123gxl-boostxl-senshub"
4. If the UART terminal is not responding mostlikely cause is the clock declaration line in "InitConsole" function. Change it

Name: Kamlesh Khagad SR.No 17965, DESE, IISc
Name: Pranjal Jaiswal SR.No 17996, DESE, IISc
Name: Sumit Kumar SR.No 17896, DESE, IISc
Course: E3-257, Embedded System Design

References

[1] Stephen St. Michael *Introduction to CAN*. All about Circuits.
`https://www.allaboutcircuits.com/technical-articles/introduction-to-can-controller-area-network/`

[2] Texas Instruments *TIVA C series datasheet*. Texas Instruments.

[3] Texas Instruments *BOOSTXL-SENSHUB datasheet*. Texas Instruments.

[4] NSK Electronics
`https://www.nskelectronics.in/`