Conestoga College

Embedded Systems Development

Capstone Project Technical Report

RESPIRATION SENSOR

Submitted by:

Mentor: Ilya Peskov

Aneez Bashorun
Contact: abashorun4929@conestogac.on.ca

Srinidhi Krishnan Bengaluru
Contact: skrishnanbengal0136@conestogac.on.ca

Hiren Kumar Tank
Contact: htank8365@conestogac.on.ca

TABLE OF CONTENT

INTRODUCTION

Getting a good night's sleep is extremely important for our general well-being and a healthy life. However, with today's fast-paced lifestyle, a lot of us are unable to get the required sleep every night. Lack of quality sleep has massive effects on an individual's well-being. Research has shown that unhealthy sleep patterns leads to diseases such as heart attack, high blood pressure, stroke and reduced productivity.

Considering these sleep effects, people are becoming more conscious about their health and importance of sleep for their daily lives. Sleep monitoring has been gaining traction in the health monitoring and technology world. However, current technologies available for monitoring sleep have several limitations and our product aims to improve on some of these limitations.

PROBLEM STATEMENT

Sleep Apnea is one of the most common of all the sleeping disorders. This sleeping disorder is challenging to track and detect for a couple of reasons. First, Doctors cannot usually detect this sleeping condition during routine office visits as no specific tests can help diagnose the condition. Secondly, Individuals with this sleeping disorder don't even know they have it because it occurs during their sleep periods.

PROPOSED SOLUTION

Our solution differs from Conventional solutions as we will be leveraging the power of a recently released Respiration Sensor (XeThru X2M200) to track respiratory and sleep activities without the need of wearables.

ABSTRACT

RESP is a Respiration Monitoring System which provides an insight into one's general state of health by monitoring vital data such as Respirations per minute(RPM) and Movement Intensity to improve the quality of life and productivity.

At the core of the project is the recently released sleep monitoring and respiratory sensor called the X2M200.This sensor offers a non-contact approach to monitor the contraction and expansion of the diaphragm, making it possible for us to get valuable data such as RPM, breathing pattern and distance. This information is utilized in our sleep monitoring application.By

integrating this sensor with a microcontroller(ESP8266) with on board WIFI module, we were able to transmit and store the data to a cloud platform called IBM Watson and also live graph the data from the sensor during its operation. This data can then be accessed by individuals that have been granted security access, making this information extremely useful to medical practitioners who will like to go back in time to view information about their patients' health for diagnoses.

ACKNOWLEDGEMENT

We would like to convey our special thanks to our mentor Ilya Peskov who guided, motivated and gave us constructive criticism all through the stages of the project. We would also like to acknowledge with much appreciation the crucial effort of technology tool room staff Karen who gave suggestions and permissions with the assembling and packaging of our project. Last but not the least, special thanks to our program co-ordinator Rudy Hofer for sponsoring us the XeThru X2M200 Sensor and providing us with technical expertise during the challenging phases of the project.

COST ANALYSIS

S/N	COMPONENTS USED	COST PER UNIT (CAD)
1.	X2M200	$250
2.	ESP8266	$20
3	Power Supply Module	$16.25
4.	Ribbon cable	$12
5.	Enclosure	$16
6.	Vero Board	$10
7.	Female Headers	$2
8.	Cables/wires	$5
9.	Miscellaneous	$100
10.	**Total**	**$431.25**

TECHNOLOGY/ KEY COMPONENTS USED

XeThru Module: The X2M200 Respiration Sensor offers both Non-Contact Sleep and Respiration monitoring in a single sensor. The X2M200 System-on-chip (SOC) provides accurate measurement of a person's Breathing Frequency, together with Distance, Chest Movement in 'mm' and Signal quality information.

The X2M200 sensor module is a complete system for remote sensing. The X2M200 sensor emits electromagnetic pulses, where most of the energy is emitted in a +/- 30 degrees' angle perpendicular from the sensor. The electromagnetic pulses penetrate through different materials.

The sensor can report the following six states:

1. No Movement: No presence is detected.
2. Movement: Presence, but no identifiable breathing movement
3. Movement: Tracking: Presence and possible breathing movement detected
4. Breathing: Valid breathing movement detected.
5. Initializing: The sensor initializes after the sleep profile is executed.
6. Unknown: The sensor is in an unknown state and requires a profile and user settings to be loaded.

When in breathing state, the following additional data is reported:

1. RPM: Respirations per Minute
2. Object Distance: Distance in meters to where breathing is detected.
3. Object Movement: Relative movement of breathing is detected.
4. Signal Quality: A relative number from 0 to 10 indicates highest signal quality

ESP-12E Wi-Fi Module: ESP8266EX with Tensillica L106 32-bit microcontroller (MCU), which features extra low power consumption and 16-bit RISC. The CPU clock speed is 80 MHz. It can also reach a maximum value 160 MHZ. ESP8266EX is often integrated with external sensor and other specific devices through its GPIOs.

Battery: 3800mA Lithium polymer battery is being used to power our product. This battery was enough to power out product for 17.94 hours approximately when it is fully charged.

Bluemix: IBM Bluemix is a cloud platform as a service (PaaS) developed by IBM. It supports several programming languages and services as well as integrated DevOps to build, run, deploy and manage applications on the cloud. Bluemix is based on cloud foundry open technology and runs on SoftLayer.

BACKGROUND THEORY

The core part of the project relies on the Data from the XeThru X2M200 Sensor module. It's very much essential to understand the working and the Serial protocol of the X2M200 Sensor module. The X2M200 Respiration Module consists of two profiles, Respiration Profile and Sleep Profile that measures the Respiration rate of a Human Target. The default factory setting for the X2M200 sensor module is the Sleep Profile but for the project we used the Respiration profile. Hence, the module must be loaded with the correct profile data for the desired application. Please refer the Methodology and References to understand more regarding the Serial Protocol of the XeThru Sensor Module.

METHODOLOGY/ STEPS TAKEN IN DEVELOPMENT PROCESS

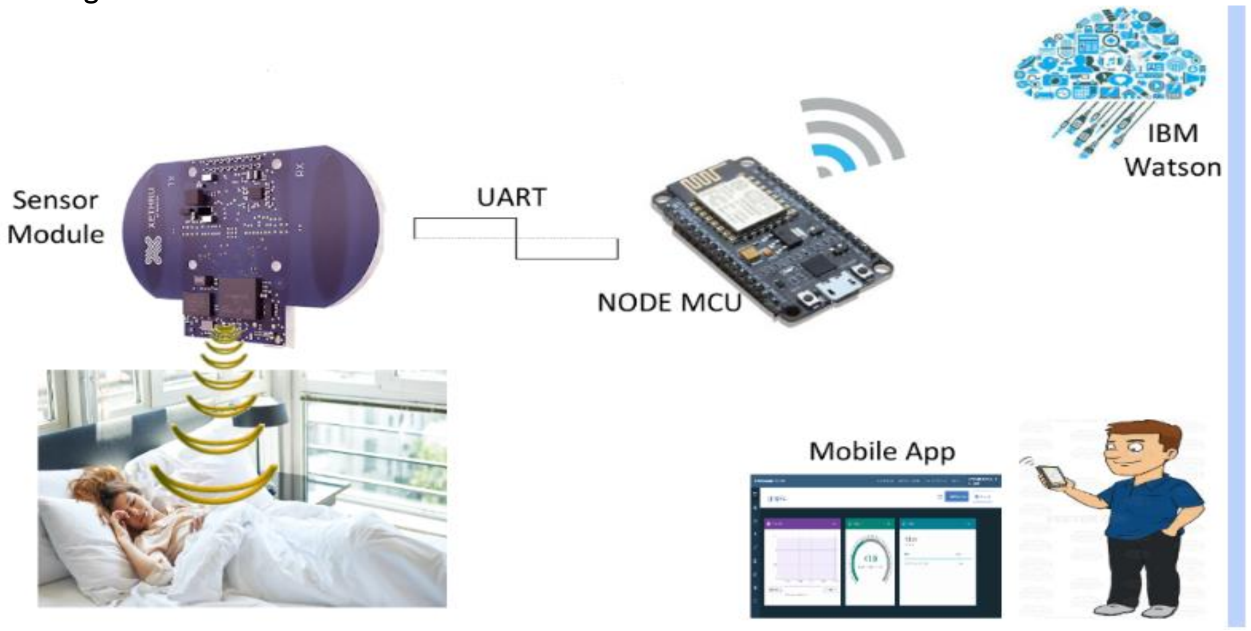
1. Planning: We followed the following procedure to plan our product

Phase 1: The preliminary focus at this stage was to understand the features of the module, read and acquire the raw data samples from the inbuilt Microcontroller connected to the (X2M200) sensor. This data will be displayed simultaneously on Serial monitor. Displaying the data will aid in determining real time synchronization of values.

Phase 2: In this phase, we wanted to save the Sensor data to a cloud platform and also live plot the sensor data.

Phase 3: In this phase our main goal was to package and build our Minimum Viable Product.

2. Design:



The design procedure that we followed is very professional. Our main aim was to take the respiration data and plot the data online. So we went step by step.

1. We got the data in sensor (XeThru module). So now, we have the data, we needed a processor to process this data.

2. We then chose the Node MCU. The main reason to choose this MCU is that it is also a Wi-Fi module. So it was a bonus for us. The Node MCU had both the processor and the Wi-Fi module on the same Board and was also very cheaply available in the market.

3. After Processing the data, it is ready to be sent to the cloud. Now our next task was to find a platform that could save the sensor data and also plot the data.

4. We first started with Plotly, the site which could plot our data. But we were not able to plot as the API was old, and it didn't have proper library available online.

5. Now we were looking for the similar services like plotly which would do our task. We found the IBM Watson, the IOT platform provided by IBM. There was also a drawback. The platform was available for trial version only for one month. But we had two different IDs to overcome this problem.

6. So there was procedure to be followed to plot the data online. The reference link is already available in the References.

7. So now we had the library to directly convert the data to JSON format, which is a format that the IBM Watson website accepts the Data.

8. Now we also built the application on the website by following the procedures given in the IBM Watson website.

9. The application screenshot could be seen in the above figure.

10. So now we were able to get the data from the XeThru module and plot the data online in real time.

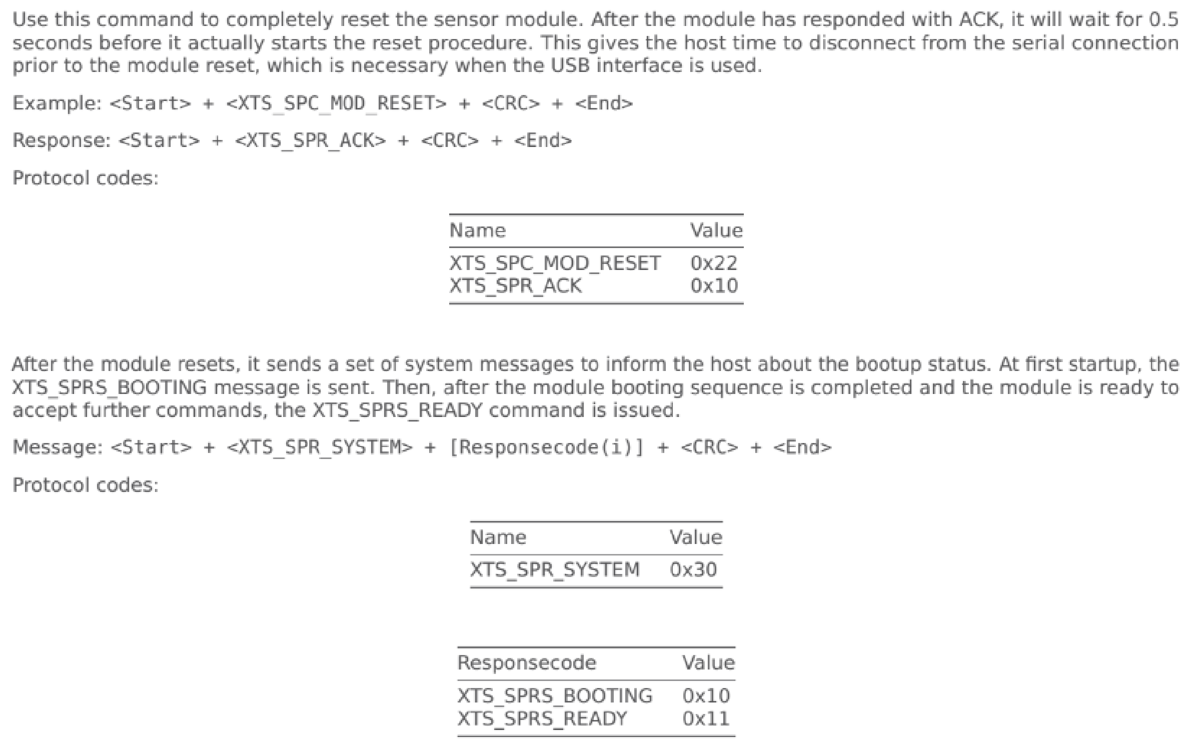
3. Implementation: We followed the following steps for Implementation

Writing the Serial Protocol

We started with the Teensy 3.1 to process our XeThru data. As there was no Library to directly communicate with the XeThru module, we had to go through the data sheet of the XeThru module. So now how it works is described below.

Reset Module → Load application → Execute Application

Reset module:


Use this command to completely reset the sensor module. After the module has responded with ACK, it will wait for 0.5 seconds before it actually starts the reset procedure. This gives the host time to disconnect from the serial connection prior to the module reset, which is necessary when the USB interface is used.

Example: `<Start> + <XTS_SPC_MOD_RESET> + <CRC> + <End>`

Response: `<Start> + <XTS_SPR_ACK> + <CRC> + <End>`

Protocol codes:

Name	Value
XTS_SPC_MOD_RESET	0x22
XTS_SPR_ACK	0x10

After the module resets, it sends a set of system messages to inform the host about the bootup status. At first startup, the XTS_SPRS_BOOTING message is sent. Then, after the module booting sequence is completed and the module is ready to accept further commands, the XTS_SPRS_READY command is issued.

Message: `<Start> + <XTS_SPR_SYSTEM> + [Responsecode(i)] + <CRC> + <End>`

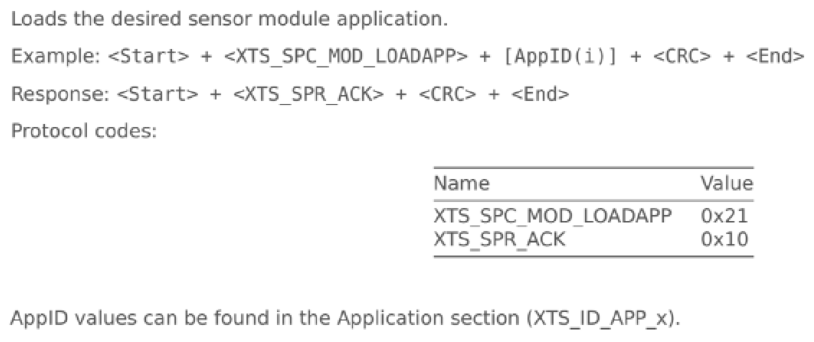
Protocol codes:

Name	Value
XTS_SPR_SYSTEM	0x30

Responsecode	Value
XTS_SPRS_BOOTING	0x10
XTS_SPRS_READY	0x11

The above figure describes the procedure to reset the module. So the example contains the data format we need to send and the response is what we get from the XeThru module. So we implemented the reset module code using C program which is available in the appendix.

Load Application:


Loads the desired sensor module application.

Example: `<Start> + <XTS_SPC_MOD_LOADAPP> + [AppID(i)] + <CRC> + <End>`

Response: `<Start> + <XTS_SPR_ACK> + <CRC> + <End>`

Protocol codes:

Name	Value
XTS_SPC_MOD_LOADAPP	0x21
XTS_SPR_ACK	0x10

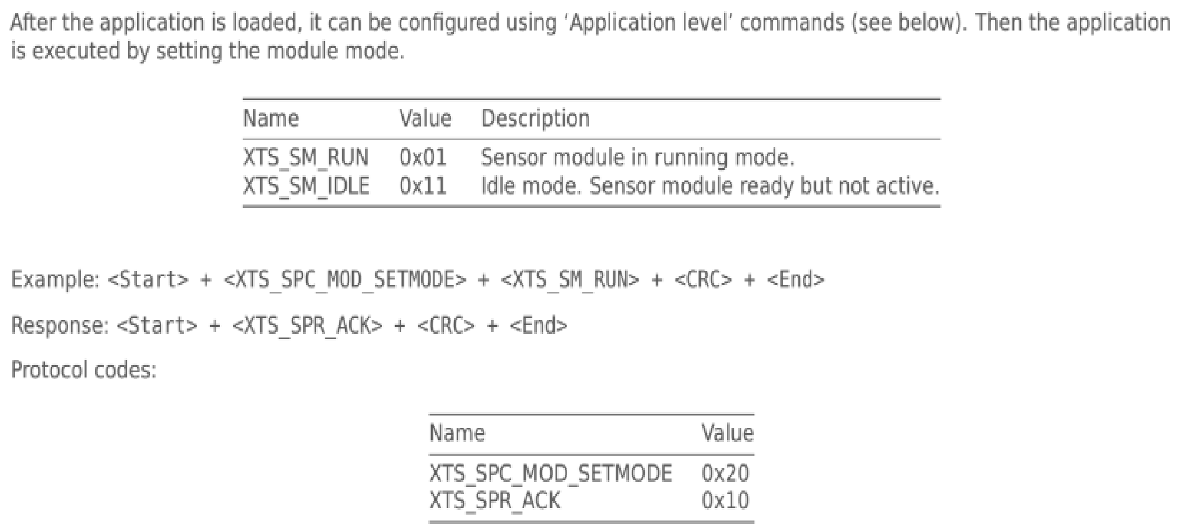
AppID values can be found in the Application section (XTS_ID_APP_x).

The above figure describes the procedure to load the application once the module is reseted.
So there are two different applications available that can be loaded.

1. Respiration application
2. Sleep application

So we implemented the respiration application. So using the c code we wrote the algorithm to implement it.

Execute Application:



After the application is loaded, it can be configured using 'Application level' commands (see below). Then the application is executed by setting the module mode.

Name	Value	Description
XTS_SM_RUN	0x01	Sensor module in running mode.
XTS_SM_IDLE	0x11	Idle mode. Sensor module ready but not active.

Example: <Start> + <XTS_SPC_MOD_SETMODE> + <XTS_SM_RUN> + <CRC> + <End>

Response: <Start> + <XTS_SPR_ACK> + <CRC> + <End>

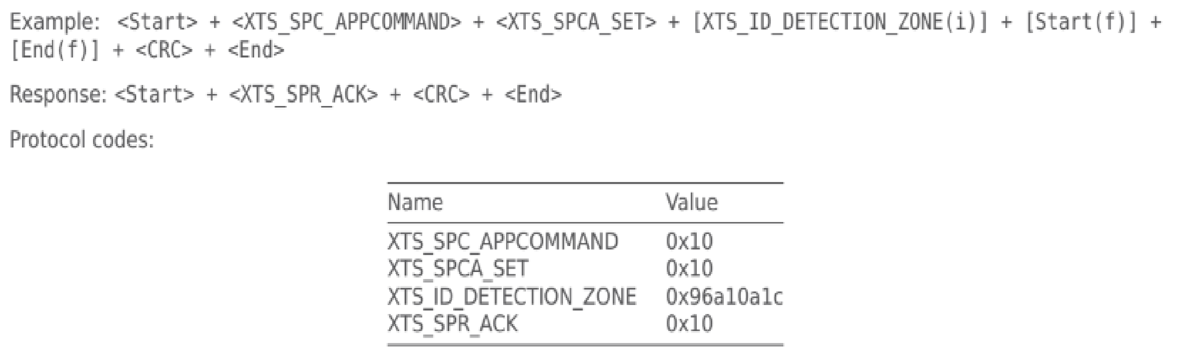
Protocol codes:

Name	Value
XTS_SPC_MOD_SETMODE	0x20
XTS_SPR_ACK	0x10

The above screenshot contains the procedure to Execute the Respiration application. So with help of C code we wrote the algorithm to implement the execution process.

But before the module could start giving the data, we need to set some parameters. The parameters are listed below:

Set detection zone:



Example: <Start> + <XTS_SPC_APPCOMMAND> + <XTS_SPCA_SET> + [XTS_ID_DETECTION_ZONE(i)] + [Start(f)] + [End(f)] + <CRC> + <End>

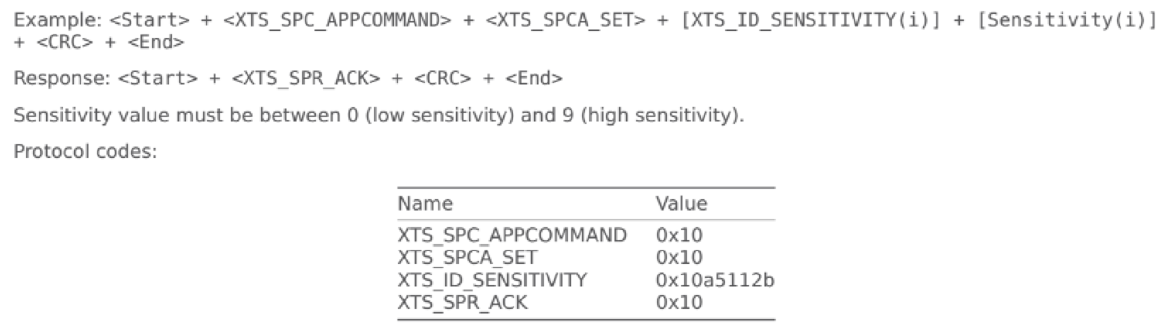
Response: <Start> + <XTS_SPR_ACK> + <CRC> + <End>

Protocol codes:

Name	Value
XTS_SPC_APPCOMMAND	0x10
XTS_SPCA_SET	0x10
XTS_ID_DETECTION_ZONE	0x96a10a1c
XTS_SPR_ACK	0x10

The above screenshot contains the procedure to set the detection zone. With help of C code, we were able to write the algorithm to set the detection zone.

Set Sensitivity:



Example: <Start> + <XTS_SPC_APPCOMMAND> + <XTS_SPCA_SET> + [XTS_ID_SENSITIVITY(i)] + [Sensitivity(i)] + <CRC> + <End>

Response: <Start> + <XTS_SPR_ACK> + <CRC> + <End>

Sensitivity value must be between 0 (low sensitivity) and 9 (high sensitivity).

Protocol codes:

Name	Value
XTS_SPC_APPCOMMAND	0x10
XTS_SPCA_SET	0x10
XTS_ID_SENSITIVITY	0x10a5112b
XTS_SPR_ACK	0x10

The above screenshot explains the procedure to set the sensitivity zone. With help of C code, we were now able to write the algorithm to implement the Sensitivity field.

So with help of the datasheet we were able to write a robust code to get the respiration data from the XeThru module.

Integrating with IBM Watson

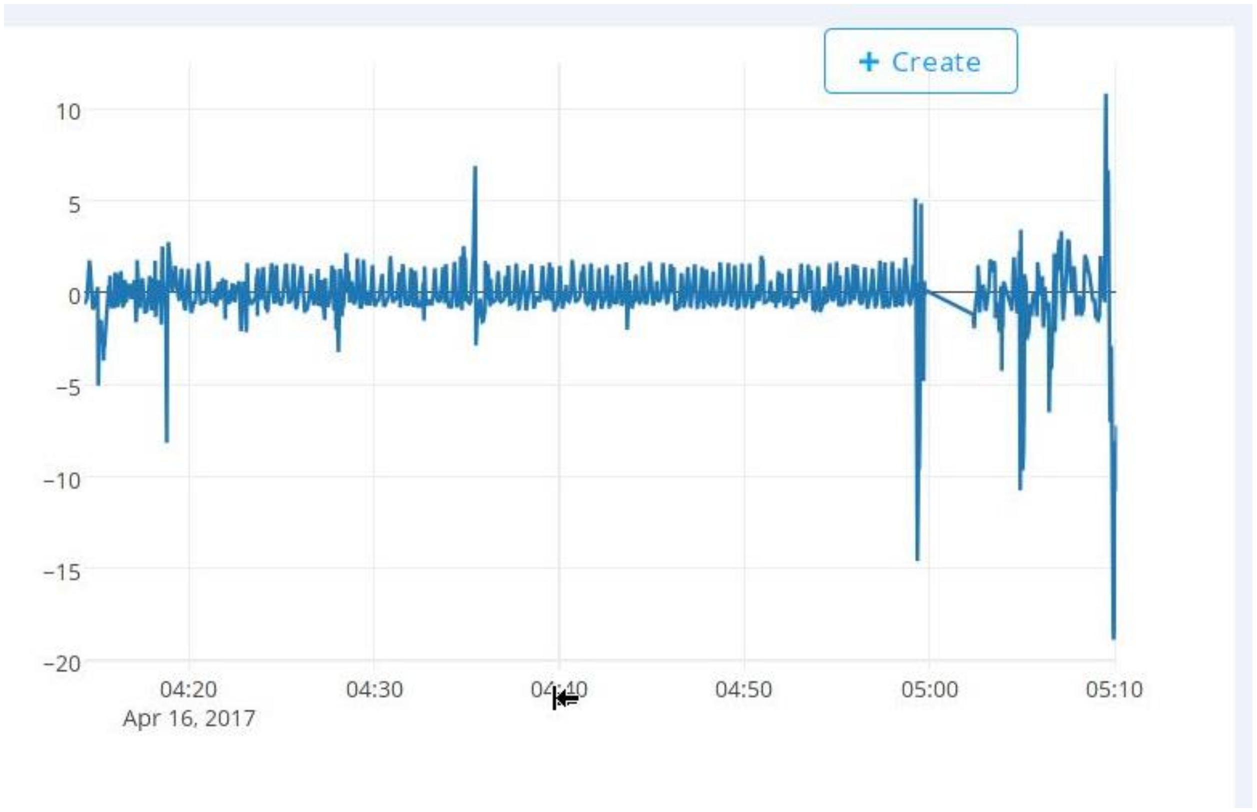
As we had the data and Wi-Fi module, we now had to send the data over the INTERNET to plot it in real-time. We started with the Plotly. But we were not able to plot the live data on Plotly Web Server. So we chose IBM Watson. We used the Node MCU module as this was also a Wi-Fi module. The source code that we wrote for the Teensy was having Arduino platform. So now to port the code to the Node MCU was not a problem. We were easily able to port the code to the Node MCU.

Now that we had the data from the XeThru module, our next target was to send the data to IBM Watson with help of the Node MCU. As there was library available from IBM Watson. We just needed to figure out the data and send the data to the IBM Watson.

The link to procedure that is to be followed to create the account on IBM Watson is given in the references at the end of the project report.

<u>RESULTS/ANALYSIS</u>

The end goal of this project is to gather sleep data in the form of breathing pattern, Respirations per minute(RPM) and Distance. These data will become useful when analysed and used for the Sleep Apnea application. The diagram below shows a screenshot of the breathing pattern of a test subject during regular night.



The figure above shows a breathing pattern of a healthy adult male. The time from 4:20a.m to 5:00 a.m. is the time period when the individual is in deep sleep. It can be noticed that during this period the breathing pattern is relatively stable with just a few spikes occurring at few intervals. Post 5:00a.m shows when the individual woke up and more activities and spikes can be noticed from his breathing pattern. By carrying out an in depth study on this pattern we can conclude that this individual is having a healthy sleep cycle and doesn't have Sleep Apnea as no pauses can be noticed from his breathing pattern.

RECOMMENDATION

All the data required for Sleep monitoring application depends on the XeThru Sensor reliability. The Xethru's limitation of 10 seconds pauses makes it a difficult task to accurately detect the presence of Sleep Apnea. Therefore, it is recommended to use Dual sensors with proper Time offsets to alternate the cycle of Data capture.

SCHEMATICS

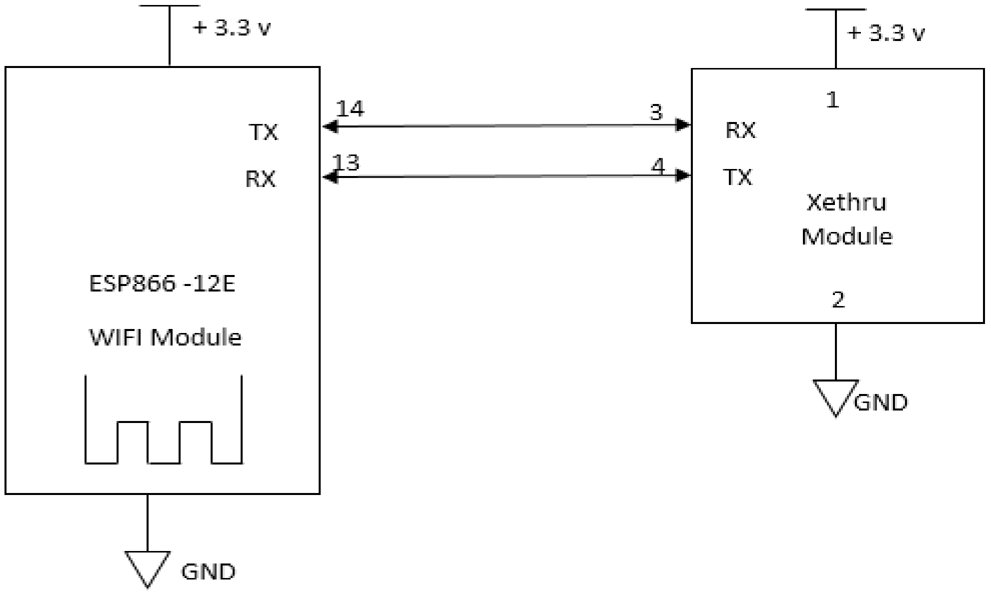


Fig 1

APPENDICES

References:

[1]. https://www.xethru.com/community/resources/xethru-serial-protocol.14/

[2]. https://mintbox.in/media/esp-12e.pdf

[3]. https://www.nhlbi.nih.gov/health/health-topics/topics/sleepapnea

[4]. https://www.xethru.com/sleep-monitoring.html

[5]. http://www.webmd.com/sleep-disorders/guide/sleep-101

[6]. https://tuts.codingo.me/category/iot

[7]. https://hackaday.io/post/17879

[8]. https://room-15.github.io/blog/2015/03/26/esp8266-at-command-reference/

[9]. http://www.webmd.com/sleep-disorders/guide/sleep-101

[10]. http://stackoverflow.com/questions/32404231/proper-post-request-to-stream-data-to-plotly-from-arduino

Source Code:

```
#include <SoftwareSerial.h>

#include <ESP8266WiFi.h>

#include "PubSubClient.h" // https://github.com/knolleary/pubsubclient/releases/tag/v2.3


SoftwareSerial mySerial(13, 14); // RX, TX


//-------- Customise these values -----------

const char* ssid = "teamASH";

const char* password = "ash12345";


#define ORG "4r31dt"

#define DEVICE_TYPE "ESP8266"

#define DEVICE_ID "5ccf7f8f7589"

#define TOKEN "Gf?&Ve4!BoXLTBEj!P"

//-------- Customise the above values --------


char server[] = ORG ".messaging.internetofthings.ibmcloud.com";

char topic[] = "iot-2/evt/status/fmt/json";

char authMethod[] = "use-token-auth";

char token[] = TOKEN;

char clientId[] = "d:" ORG ":" DEVICE_TYPE ":" DEVICE_ID;


WiFiClient wifiClient;

PubSubClient client(server, 1883, NULL, wifiClient);



/* radar constants */
```

```
#define BUF_SIZE 128


// Flag Byte codes

const unsigned  char  startByte   = 0X7D;

const unsigned  char  stopByte    = 0X7E;

const unsigned  char  escapeByte  = 0X7F;


// Byte code to reset the module

const unsigned  char  XTS_SPC_MOD_RESET = 0X22;


// Byte code for acknowledgement

const unsigned  char  XTS_SPR_ACK       = 0X10;


// Byte codes to inform the host regarding bootup status

const unsigned  char  XTS_SPR_SYSTEM    = 0X30;

const unsigned  char  XTS_SPRS_BOOTING  = 0X10;

const unsigned  char  XTS_SPRS_READY    = 0X11;


// Ping commands to check connection to the module

const unsigned  char  XTS_SPC_PING             = 0X01;       // ping command code

const unsigned  long  XTS_DEF_PINGVAL          = 0Xeeaaeaae;  // ping seed value

const unsigned  char  XTS_SPR_PONG             = 0X01;       // pong response code

const unsigned  long  XTS_DEF_PONGVAL_READY     = 0xaaeeaeea;  // module is ready

const unsigned  long  XTS_DEF_PONGVAL_NOTREADY  = 0xaeeaeeaa;  // module is not ready


// Byte codes to load the application

const unsigned  char  XTS_SPC_MOD_LOADAPP = 0X21;
```

```
// Byte codes to execute the application

const unsigned  char  XTS_SM_RUN  = 0X01; // Sensor module in running mode

const unsigned  char  XTS_SM_IDLE = 0X11; // Idle mode. Sensor module ready but not active

const unsigned  char  XTS_SPC_MOD_SETMODE = 0X20;


// Byte codes for LED control

const unsigned  char  XTS_SPC_MOD_SETLEDCONTROL = 0X24;

const unsigned  char  XT_UI_LED_MODE_OFF        = 0X00;

const unsigned  char  XT_UI_LED_MODE_SIMPLE     = 0X01;

const unsigned  char  XT_UI_LED_MODE_FULL       = 0X02;


// Byte codes for Error Handling

const unsigned  char  XTS_SPR_ERROR = 0X20;

const unsigned  char  XTS_SPRE_NOT_RECOGNIZED = 0X01; // command not recognized

const unsigned  char  XTS_SPRE_CRC_FAILED     = 0X02; // checksum failed

const unsigned  char  XTS_SPRE_APP_INVALID    = 0X20; // command recognized, but invalid


// Application commands

const unsigned  char  XTS_SPC_APPCOMMAND    = 0X10;

const unsigned  char  XTS_SPCA_SET          = 0X10;


// Byte code to set Detection zone

const unsigned  long  XTS_ID_DETECTION_ZONE  = 0X96a10a1c;


// Byte code to set Sensitivity

const unsigned  long  XTS_ID_SENSITIVITY  = 0x10a5112b;


// Byte codes for Respiration application
```

```
const unsigned  long  XTS_ID_APP_RESP     = 0X1423a2d6;

const unsigned  char  XTS_SPR_APPDATA     = 0X50;

const unsigned  long  XTS_ID_RESP_STATUS  = 0X2375fe26;


//Detection Zone constants

const long DETECTIONZONE_0_5m = 0X3f000000;

const long DETECTIONZONE_1_2m = 0X3f99999a;

const long DETECTIONZONE_1_0m = 0X3f800000;

const long DETECTIONZONE_1_7m = 0X3fd9999a;

const long DETECTIONZONE_1_8m = 0X3fe66666;

const long DETECTIONZONE_2_5m = 0X40200000;


//Sensitivity constants

const long SENSITIVITY  = 0X00000005;

unsigned char recv_buf[BUF_SIZE];   // Buffer for receiving data from radar. Size is 32 Bytes

static int resetFlag = 0;



void flush_buffer() {


  for (int x = 0; x < BUF_SIZE; x++) {

    recv_buf[x] = 0;

  }

  Serial.print("\n receive buffer flushed...\n");

  return;

}
```

```
void send_command(const unsigned char * cmd, int len) {

  // Calculate CRC
  char crc = startByte;
  for (int i = 0; i < len; i++) {
    crc ^= cmd[i];
  }

  // Send startByte + Data + crc + stopByte
  mySerial.write(startByte);
  mySerial.write(cmd, len);
  mySerial.write(crc);
  mySerial.write(stopByte);
}


void reset_module()
{
  if (Serial.available() || mySerial.available()) {
    Serial.printf("\n Serial ports cleared \n");
    flush_buffer();
  }

  delay(1000);
  send_command(&XTS_SPC_MOD_RESET, 1);
  delay(2000);
  receive_data(); // acknowledgement
  delay(1000);
  receive_data(); // booting state

  delay(1000);

  receive_data(); // ready state

}



void load_respiration_app() {

  //Fill send buffer

  unsigned char send_buf[5];

  send_buf[0] = XTS_SPC_MOD_LOADAPP;

  send_buf[4] = (XTS_ID_APP_RESP >> 24) & 0xff;

  send_buf[3] = (XTS_ID_APP_RESP >> 16) & 0xff;

  send_buf[2] = (XTS_ID_APP_RESP >> 8) & 0xff;

  send_buf[1] =  XTS_ID_APP_RESP & 0xff;


  //Send the command

  send_command(send_buf, 5);

  delay(2000);

  receive_data();

}



void set_detetction_zone() {

  //Fill send buffer

  unsigned char send_buf[14];


  send_buf[0] = XTS_SPC_APPCOMMAND;

  send_buf[1] = XTS_SPCA_SET;
  send_buf[5] = (XTS_ID_DETECTION_ZONE >> 24) & 0xff;
  send_buf[4] = (XTS_ID_DETECTION_ZONE >> 16) & 0xff;
  send_buf[3] = (XTS_ID_DETECTION_ZONE >> 8) & 0xff;
  send_buf[2] =  XTS_ID_DETECTION_ZONE & 0xff;

  send_buf[9] = (DETECTIONZONE_0_5m >> 24) & 0xff;
  send_buf[8] = (DETECTIONZONE_0_5m >> 16) & 0xff;
  send_buf[7] = (DETECTIONZONE_0_5m >> 8) & 0xff;
  send_buf[6] =  DETECTIONZONE_0_5m & 0xff;

  send_buf[13] = (DETECTIONZONE_1_2m >> 24) & 0xff;
  send_buf[12] = (DETECTIONZONE_1_2m >> 16) & 0xff;
  send_buf[11] = (DETECTIONZONE_1_2m >> 8) & 0xff;
  send_buf[10] =  DETECTIONZONE_1_2m & 0xff;

  //Send the command
  send_command(send_buf, 14);
  delay(2000);
  receive_data();
}


void set_sensitivity() {
  //Fill send buffer
  unsigned char send_buf[10];

  send_buf[0] = XTS_SPC_APPCOMMAND;
  send_buf[1] = XTS_SPCA_SET;


  send_buf[5] = (XTS_ID_SENSITIVITY >> 24) & 0xff;

  send_buf[4] = (XTS_ID_SENSITIVITY >> 16) & 0xff;

  send_buf[3] = (XTS_ID_SENSITIVITY >> 8) & 0xff;

  send_buf[2] =  XTS_ID_SENSITIVITY & 0xff;


  send_buf[9] = (SENSITIVITY >> 24) & 0xff;

  send_buf[8] = (SENSITIVITY >> 16) & 0xff;

  send_buf[7] = (SENSITIVITY >> 8) & 0xff;

  send_buf[6] =  SENSITIVITY & 0xff;


  //Send the command

  send_command(send_buf, 10);

  delay(2000);

  receive_data();

}



// Execute respiration application

void execute_app() {

  //Fill send buffer

  unsigned char send_buf[2];

  send_buf[0] = XTS_SPC_MOD_SETMODE;

  send_buf[1] = XTS_SM_RUN;


  //Send the command

  send_command(send_buf, 2);
  delay(2000);
  receive_data();
}



float getDistance() {
  unsigned char hexToInt[4] = {0};
  unsigned int  signbit;
  unsigned int  exponent;
  unsigned int  mantissa;
  unsigned int  uintVal;
  unsigned int* uintPtr = NULL;

  hexToInt[3] = recv_buf[21];
  hexToInt[2] = recv_buf[20];
  hexToInt[1] = recv_buf[19];
  hexToInt[0] = recv_buf[18];

  uintPtr =  (unsigned int*)(hexToInt);
  uintVal  = *uintPtr;
  signbit =  (uintVal & 0x80000000) >> 31;
  exponent = (uintVal & 0x7F800000) >> 23 ;
  mantissa = (uintVal & 0x007FFFFF) | 0x00800000;

  return (float)((signbit == 1) ? -1.0 : 1.0) * mantissa / pow(2.0, (127 - exponent + 23));
}
```

```
float getMovement() {
  unsigned char hexToInt[4] = {0};
  unsigned int  signbit;
  unsigned int  exponent;
  unsigned int  mantissa;
  unsigned int  uintVal;
  unsigned int* uintPtr = NULL;

  hexToInt[3] = recv_buf[25];
  hexToInt[2] = recv_buf[24];
  hexToInt[1] = recv_buf[23];
  hexToInt[0] = recv_buf[22];

  uintPtr =  (unsigned int*)(hexToInt);
  uintVal  = *uintPtr;
  signbit =  (uintVal & 0x80000000) >> 31;
  exponent = (uintVal & 0x7F800000) >> 23 ;
  mantissa = (uintVal & 0x007FFFFF) | 0x00800000;

  return (float)((signbit == 1) ? -1.0 : 1.0) * mantissa / pow(2.0, (127 - exponent + 23));
}


void receive_data() {

  // Get response
  char last_char = 0x00;
  int recv_len = 0; //Number of bytes received
```

```
while (!mySerial.available());


//Wait for start character

while (mySerial.available())

{

 char c = mySerial.read();  // Get one byte from X2M200 Xethru module


 /*if (c == escapeByte)

   {

   // If it's an escape character –

   // ...ignore next character in buffer

   //c = mySerial.read();

   continue;

   }*/


 if (c == startByte)

 {

   // If it's the start character –

   // We fill the first character of the buffer and move on

   recv_buf[0] = startByte;

   recv_len = 1; //flag to check that a byte is received

   break;

 }

}


// Start receiving the rest of the bytes

while (mySerial.available())
```

```
{
  // read a byte
  char cur_char = mySerial.read();  // Get one byte from Xethru module

  if (cur_char == -1)
  {
    continue;
  }

  // Fill response buffer, and increase counter
  recv_buf[recv_len] = cur_char;
  recv_len++;

  // is it the stop byte?
  if (cur_char == stopByte)
  {
    if (last_char != escapeByte)
      break;  //Exit this loop
  }

  // Update last_char
  last_char = cur_char;
}

// Calculate CRC
char crc = 0;
char escape_found = 0;
```

```
// CRC is calculated without the crc itself and the stop byte, hence the -2 in the counter
for (int i = 0; i < recv_len - 2; i++)
{
  // We need to ignore escape bytes when calculating crc
  if (recv_buf[i] == escapeByte && !escape_found)
  {
    escape_found = 1;
    continue;
  }
  else
  {
    crc ^= recv_buf[i];
    escape_found = 0;
  }
}

// Check if calculated CRC matches the recieved
if (crc == recv_buf[recv_len - 2])
{
  Serial.print("\n Received Data is Correct\n");

  for (int i = 0; i < BUF_SIZE; i++)
  {
    Serial.printf("%x  ", recv_buf[i]);
  }

  Serial.print("\n Received All Data \n");
}
```

```
  else
  {
    Serial.print("\n Received Data is Not Correct \n");
  }

}



void setup() {
  // Open serial communications and wait for port to open:
  Serial.begin(115200);
  mySerial.begin(115200);
  delay(2000);
  flush_buffer();
  Serial.println("\n\n Starting.... \n\n");

  // start reset
  Serial.print("\n Starting RESET......\n");
  reset_module();
  Serial.print("\n RESET Successfull...\n");
  flush_buffer();

  // load application
  Serial.print("\n Loading Respiration APPLICATION......\n");
  load_respiration_app();
  Serial.print("\n Loading APPLICATION Successfull...\n");
  flush_buffer();
```

```
// set detection zone
Serial.print("\n Setting Detection Zone......\n");
set_detetction_zone();
Serial.print("\n Detection Zone Set Sucessfully...\n");
flush_buffer();

// set sensitivity
Serial.print("\n Setting Sensitivity......\n");
set_detetction_zone();
Serial.print("\n Sensitivity Set Sucessfully...\n");
flush_buffer();

// execute application
Serial.print("\n Executing Respiration APPLICATION......\n");
execute_app();
Serial.print("\n Executing APPLICATION Successfull...\n");
flush_buffer();

Serial.print("Connecting to "); Serial.print(ssid);
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.print(".");
}
Serial.println("");
Serial.print("WiFi connected, IP address: "); Serial.println(WiFi.localIP());
```

```
}



void loop() {
  // put your main code here, to run repeatedly:

  if (!client.connected()) {
    Serial.print("Reconnecting client to ");
    Serial.println(server);
    while (!client.connect(clientId, authMethod, token)) {
      Serial.print(".");
      delay(500);
    }
    Serial.println();
  }

  delay(100);
  receive_data();
  int state_data = (int)recv_buf[10];
  Serial.printf("\n state data = %d \n", state_data);

  if (state_data == 0)
  {
    int rpm = (int)recv_buf[14];
    float distance = getDistance();
    float movement = getMovement();
    //Serial.printf("\n rpm = %d\tdistance = %f m\tmovement = %f mm \n\n", rpm, distance,movement);
    //Serial.print("rpm:"); Serial.println(rpm);
    //Serial.print("movement:"); Serial.println(movement);
    //Serial.print("distance:"); Serial.println(distance);

    if(movement > 15.0) movement = 15.0;
    if(movement < -15.0) movement = -15.0;
    String payload = "{\"d\":{\"Name\":\"18FE34D81E46\"";
    payload += ",\"movement\":";
    payload += movement;
    payload += ",\"rpm\":";
    payload += rpm;
    payload += ",\"distance\":";
    payload += distance;
    payload += "}}";

    Serial.print("Sending payload: ");
    Serial.println(payload);

    if (client.publish(topic, (char*) payload.c_str())) {
      Serial.println("Publish ok");
    } else {
      Serial.println("Publish failed");
    }
  }
}
```

--- Thank You --