Transmit and Receive LIS3DH Accelerometer Data by Integrating Arduino Mega 2560 Microcontrollers and NRF24L01 Trans receiver modules.

Monty Sourjah

**Goal:**

The goal of this project is to transmit and receive measured accelerometer data using a microcontroller and NRF24L01 module and display the measured values in the PC serial monitor.

Step 1

Collect and display the LIS3DH “X”,” Y”,” Z” values in the PC serial monitor implementing the I2C protocol. Setup the NRF24L01 trans receiver to transmit “X”,” Y” and “Z” values to the receiver.

(LIS3DH+Mega 2560+NRF24L01+PC serial monitor).

Step 2

Setup the receiver NRF24L01 module to receive data from the transmitter and display received data in the PC serial monitor.

(Mega 2560+NRF24L01+PC serial monitor)

**Deliverables:**

The purpose of this project is to fetch the accelerometer data from LIS3DH sensor and wirelessly transmit using the NRF24L01 to a computer serial monitor. There are many applications of this concept one of them is used in multi-copter flight controllers.

# Components

LIS3DH Accelerometer

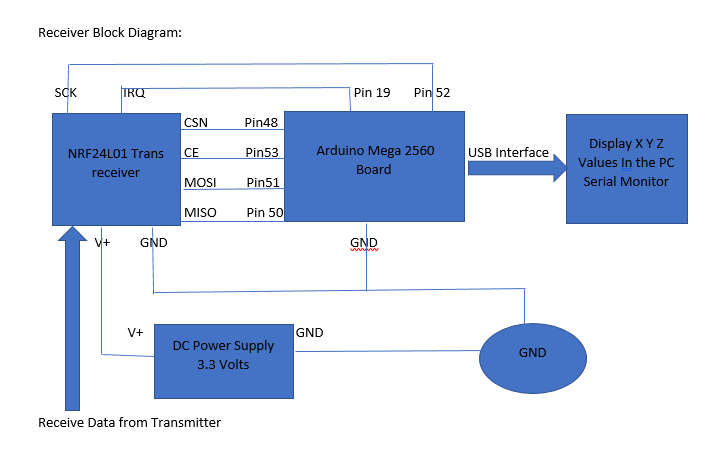
## Accelerometers are devices that measure acceleration which is the rate of change of the [velocity](http://en.wikipedia.org/wiki/Velocity) of an object. They measure in meters per second squared (m/s2) or in G-forces (g). A single G-force for us here on planet Earth is equivalent to 9.8 m/s2, but this does vary slightly with elevation. Accelerometers are useful for sensing vibrations in systems or for orientation applications. Accelerometers are electromechanical devices that sense either static or dynamic forces of acceleration. Static forces include gravity, while dynamic forces can include vibrations and movement. This sensor communicates over I2C or SPI (our library code supports both) so you can share it with a bunch of other sensors on the same I2C bus. There's an address selection pin so you can have two accelerometers share an I2C bus.

## Arduino Mega 2560 board

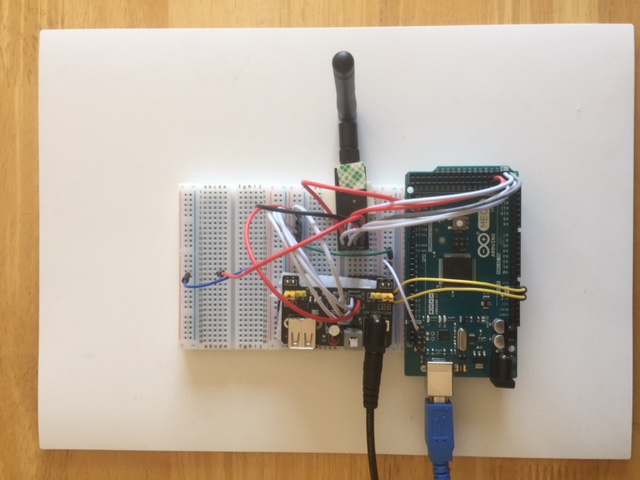
The Arduino Mega 2560 is a microcontroller board is based on the ATmega2560 chip by Atmel. It has 54 digital input/output pins (of which 14 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. The board can operate on an external supply of 6 to 20 volts.

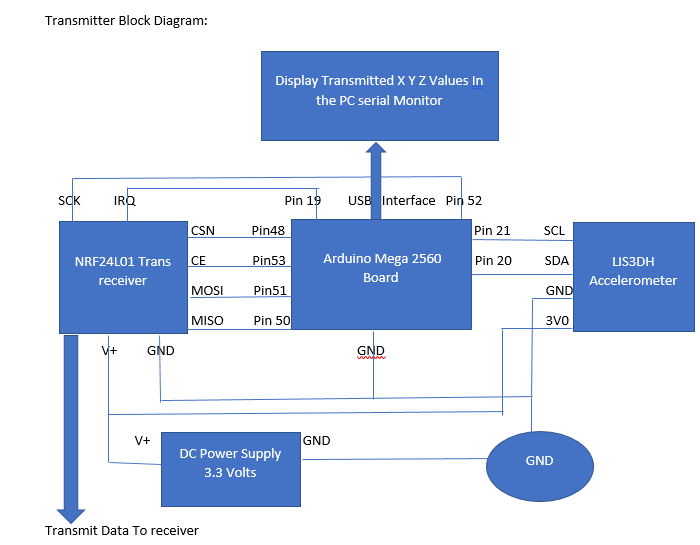
## NRF24L01

The nRF24L01 is a highly integrated, ultra-low power 2Mbps RF transceiver.  It uses the 2.4 GHz band and it can operate with baud rates from 250 kbps up to 2 Mbps. If used in open space and with lower baud rate its range can reach up to 100 meters. The module can use 125 different channels which gives a possibility to have a network of 125 independently working modems in one place. Each channel can have up to 6 addresses, or each unit can communicate with up to 6 other units at the same time. he operating voltage of the module is from 1.9 to 3.6V.

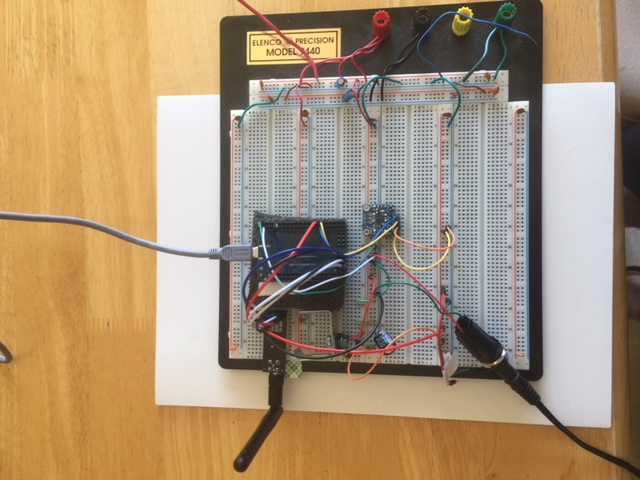


Receiver Setup:



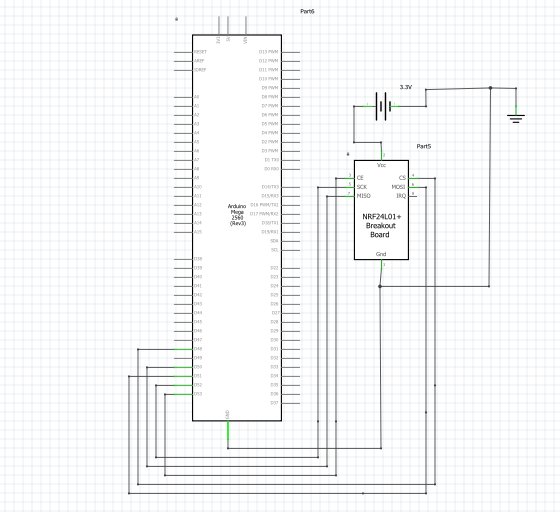


Transmitter Setup:

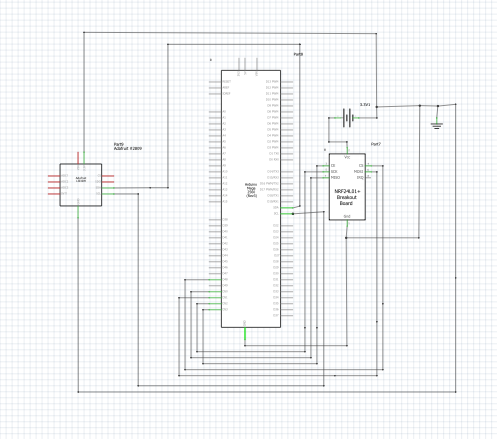


# Schematics

Receiver:



Transmitter:



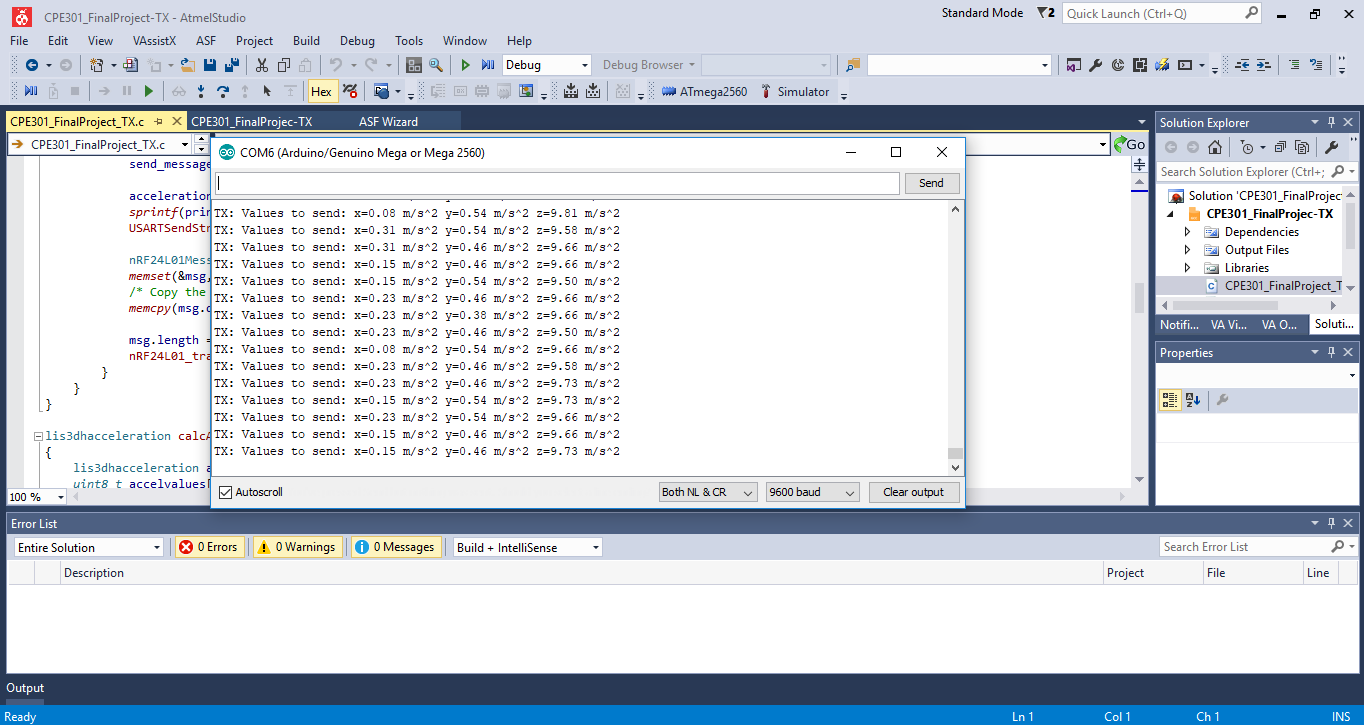
# Implementation

LIS3DH interfaced with Arduino mega 2560 board using the I2C protocol. Arduino mega 2560 I2C pins are pin #21(SCL) and pin#20(SDA).

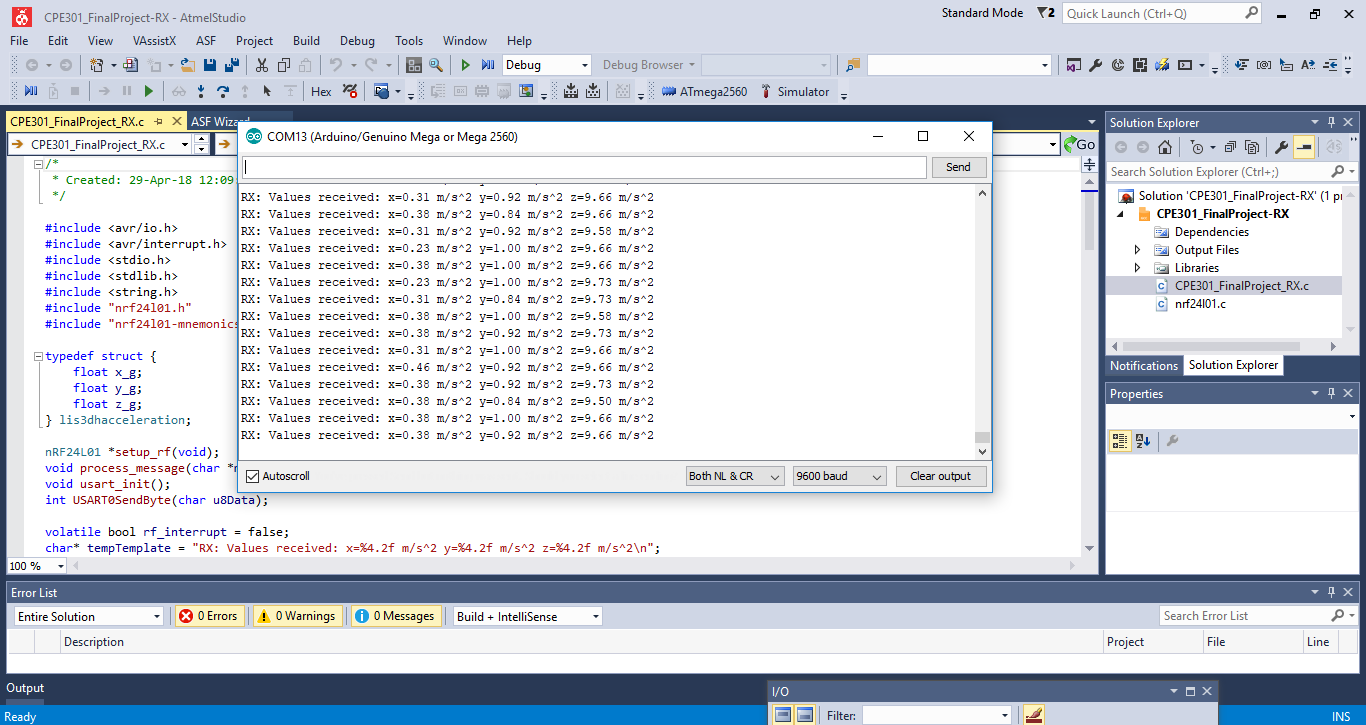
NRF24L01 is connected to Arduino mega 2560 using the SPI protocol. Arduino mega 2560 SPI pins are MOSI (pin #51), MISO(pin#50), SCK(pin#52), CE(pin#53), CSN(pin#48) and IRQ(pin#21).

# Snapshots

Transmitter Output:



Receiver Output:



# Links

<http://www.youtube.com/watch?v=1GTnE82lnQ8>

<https://github.com/ballasl/CPE301_Final-Project>



Transmitter Flow Chart:



Transmitter Code:

/\*

\* lis3dh.c

\*

\* Created: 28-Apr-18 6:24:02 AM

\*/

#include <avr/io.h>

#include <util/twi.h>

#include "lis3dh.h"

#include "usart.h"

//#define ISDEBUGGING

#ifdef ISDEBUGGING

inline void USARTDbgSendStr(char\* \_str)

{

USARTSendStr(\_str);

}

#else

inline void USARTDbgSendStr(char\* \_str)

{

return;

}

#endif

/\* BDIV = ((CPU\_CLOCK\_FREQ/I2C\_BAUD)-16)/(2\*prescalar) \*/

/\* For prescaler = 1 and data rate = 100,000 \*/

#define BDIV (FOSC / 100000 - 16) / 2 + 1 // BDIV=73

/\*\*\*\*\*\*\*\*\*\*\* I2C specific \*\*\*\*\*\*\*\*\*\*\*/

inline void i2c\_write(unsigned char data)

{

TWDR = data;

TWCR = (1<<TWINT) | (1<<TWEN); /\* clear interrupt to start transmission \*/

while ((TWCR & (1<<TWINT))==0); /\* wait for transmission \*/

}

inline void i2c\_init(void)

{

TWSR=0x0; /\* Set prescalar to 1 \*/

TWBR=BDIV; /\* Set bit rate register. See above \*/

TWCR=(1<<TWEN); /\* Enable the TWI module \*/

}

inline void i2c\_start(void)

{

TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN); /\* send start condition \*/

while ((TWCR & (1<<TWINT)) == 0) ; /\* wait for transmission \*/

}

inline void i2c\_stop(void)

{

TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO); /\* send stop condition \*/

}

/\* \*\*\*\*\*\*\*\*\*\* LIS3DH specific \*\*\*\*\*\*\*\*\*\*/

#define TWI\_SLA\_LIS3DH 0b00110000 /\* LIS3DH\_DEFAULT\_ADDRESS for 5 volt //If SDO/SA0 is 3V is 0b00110010 \*/

#define MAX\_ITER 200

*uint8\_t* twst;

int lis3dh\_write\_byte(*uint8\_t* addr, *uint8\_t* value)

{

*uint8\_t* sla, n = 0;

int rv = 0;

sla = TWI\_SLA\_LIS3DH;

restart:

if (n++ >= MAX\_ITER)

return -1;

begin:

USARTDbgSendStr("pointW0\n");

i2c\_start();

switch (twst = *TW\_STATUS*)

{

case *TW\_REP\_START*: /\* OK, but should not happen \*/

case *TW\_START*:

break;

case *TW\_MT\_ARB\_LOST*:

goto begin;

default:

return -1; /\* Not in start condition \*/

}

USARTDbgSendStr("pointW1\n");

i2c\_write(sla | *TW\_WRITE*);

switch (twst = *TW\_STATUS*)

{

case *TW\_MT\_SLA\_ACK*:

break;

case *TW\_MT\_SLA\_NACK*: /\* Device busy writing \*/

USARTDbgSendStr("point1Wnack\n");

goto restart;

case *TW\_MT\_ARB\_LOST*:

USARTDbgSendStr("point1Wlost\n");

goto begin;

default:

goto error; /\* must send stop condition \*/

}

USARTDbgSendStr("pointW2\n");

i2c\_write(addr);

switch (twst = *TW\_STATUS*)

{

case *TW\_MT\_DATA\_ACK*:

break;

case *TW\_MT\_DATA\_NACK*:

goto quit;

case *TW\_MT\_ARB\_LOST*:

goto begin;

default:

goto error; /\* must send stop condition \*/

}

USARTDbgSendStr("pointW3\n");

i2c\_write(value);

switch (twst = *TW\_STATUS*)

{

case *TW\_MT\_DATA\_NACK*:

goto error; /\* device write protected \*/

case *TW\_MT\_DATA\_ACK*:

rv++;

break;

default:

goto error;

}

quit:

i2c\_stop();

USARTDbgSendStr("pointW5\n");

return rv;

error:

USARTDbgSendStr("pointW4\n");

rv = -1;

goto quit;

}

int lis3dh\_read\_bytes(*uint8\_t* addr, int len, *uint8\_t* \*buf)

{

*uint8\_t* sla, twcr, n = 0;

int rv = 0;

sla = TWI\_SLA\_LIS3DH;

/\* First cycle: Master transmitter mode \*/

restart:

if (n++ >= MAX\_ITER)

return -1;

begin:

USARTDbgSendStr("pointR0\n");

i2c\_start();

switch (twst = *TW\_STATUS*)

{

case *TW\_REP\_START*: /\* OK, but should not happen \*/

case *TW\_START*:

break;

case *TW\_MT\_ARB\_LOST*:

goto begin;

default:

return -1; /\* Not in start condition \*/

}

USARTDbgSendStr("pointR1\n");

i2c\_write(sla | *TW\_WRITE*);

switch (twst = *TW\_STATUS*)

{

case *TW\_MT\_SLA\_ACK*:

break;

case *TW\_MT\_SLA\_NACK*: /\* Device busy writing \*/

goto restart;

case *TW\_MT\_ARB\_LOST*:

goto begin;

default:

goto error; /\* must send stop condition \*/

}

USARTDbgSendStr("pointR2\n");

i2c\_write(addr);

switch (twst = *TW\_STATUS*)

{

case *TW\_MT\_DATA\_ACK*:

break;

case *TW\_MT\_DATA\_NACK*:

goto quit;

case *TW\_MT\_ARB\_LOST*:

goto begin;

default:

goto error; /\* must send stop condition \*/

}

/\* Next cycle(s): master receiver mode \*/

USARTDbgSendStr("pointR3\n");

i2c\_start();

switch (twst = *TW\_STATUS*)

{

case *TW\_START*: /\* OK, but should not happen \*/

case *TW\_REP\_START*:

break;

case *TW\_MT\_ARB\_LOST*:

goto begin;

default:

goto error;

}

/\* send SLA+R \*/

USARTDbgSendStr("pointR4\n");

i2c\_write(sla | *TW\_READ*);

switch (twst = *TW\_STATUS*)

{

case *TW\_MR\_SLA\_ACK*:

break;

case *TW\_MR\_SLA\_NACK*:

goto quit;

case *TW\_MR\_ARB\_LOST*:

goto begin;

default:

goto error;

}

for (twcr = (1<<TWINT) | (1<<TWEN) | (1<<TWEA); len > 0; len--)

{

if (len == 1)

twcr = (1<<TWINT) | (1<<TWEN); /\* send NAK this time \*/

TWCR = twcr; /\* clear int to start transmission \*/

while ((TWCR & (1<<TWINT)) == 0) ; /\* wait for transmission \*/

switch (twst = *TW\_STATUS*)

{

case *TW\_MR\_DATA\_NACK*:

len = 0; /\* force end of loop \*/

/\* FALLTHROUGH \*/

case *TW\_MR\_DATA\_ACK*:

\*buf++ = TWDR;

rv++;

if(twst == *TW\_MR\_DATA\_NACK*) goto quit;

break;

default:

goto error;

}

}

quit:

TWCR = (1<<TWINT) | (1<<TWSTO) | (1<<TWEN); /\* send stop condition \*/

return rv;

error:

rv = -1;

goto quit;

}

int setup\_lis3dh()

{

// CTRL\_REG1 (0x20):

// ODR3 ODR2 ODR1 ODR0 LPen Zen Yen Xen

// ODR3 ODR2 ODR1 ODR0

// 0 1 0 0 -> 50 Hz

// CTRL\_REG1: enable x,y,z@50Hz = 0x47

USARTDbgSendStr("CTRL\_REG1\n");

if (lis3dh\_write\_byte(CTRL\_REG1,0x47)==-1)

return -1;

// CTRL\_REG2 (21h):

// HPM1 HPM0 HPCF2 HPCF1 FDS HPCLICK HPIS2 HPIS1

// CTRL\_REG2: no filters

USARTDbgSendStr("CTRL\_REG2\n");

if (lis3dh\_write\_byte(CTRL\_REG2,0x00)==-1)

return -1;

// CTRL\_REG3 (22h):

// I1\_CLICK I1\_AOI1 I1\_AOI2 I1\_DRDY1 I1\_DRDY2 I1\_WTM I1\_OVERRUN --

// CTRL\_REG3: no interrupts

USARTDbgSendStr("CTRL\_REG3\n");

if (lis3dh\_write\_byte(CTRL\_REG3,0x00)==-1)

return -1;

// CTRL\_REG4 (23h):

// BDU BLE FS1 FS0 HR ST1 ST0 SIM

// CTRL\_REG4: BDU (Block data update), HR (High Resolution Output), 4G scale.

USARTDbgSendStr("CTRL\_REG4\n");

if (lis3dh\_write\_byte(CTRL\_REG4, 0x90)==-1)

return -1;

// CTRL\_REG5 (24h)

// BOOT FIFO\_EN -- -- LIR\_INT1 D4D\_INT1 0 0

// Normal mode, FIFO disabled

USARTDbgSendStr("CTRL\_REG5\n");

if (lis3dh\_write\_byte(CTRL\_REG5,0x00)==-1)

return -1;

// CTRL\_REG6 (25h):

// I2\_CLICKen I2\_INT1 0 BOOT\_I1 0 - - H\_LACTIVE -

// All zero.

USARTDbgSendStr("CTRL\_REG6\n");

if (lis3dh\_write\_byte(CTRL\_REG6,0x00)==-1)

return -1;

return 0;

}

/\*

\* CPE301\_Final Project\_Rx

\* Created: 29-Apr-18 12:09:07 PM

\*/

#include <avr/io.h>

#include <avr/interrupt.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "nrf24l01.h"

#include "nrf24l01-mnemonics.h"

typedef struct {

float x\_g;

float y\_g;

float z\_g;

} lis3dhacceleration;

nRF24L01 \*setup\_rf(void);

void process\_message(char \*message);

void usart\_init();

int USART0SendByte(char u8Data);

volatile bool rf\_interrupt = false;

char\* tempTemplate = "RX: Values received: x=%4.2f m/s^2 y=%4.2f m/s^2 z=%4.2f m/s^2\n";

void USARTSendStr(char\* \_str);

#define FOSC 16000000 // Clock Speed

#define BAUD 9600

#define MYUBRR FOSC/16/BAUD - 1

int main(void) {

*uint8\_t* address[5] = { 0x01, 0x01, 0x01, 0x01, 0x01 };

usart\_init();

nRF24L01 \*rf = setup\_rf();

sei();

nRF24L01\_listen(rf, 0, address);

*uint8\_t* addr[5];

nRF24L01\_read\_register(rf, CONFIG, addr, 1);

while (true) {

if (rf\_interrupt) {

rf\_interrupt = false;

while (nRF24L01\_data\_received(rf)) {

nRF24L01Message msg;

nRF24L01\_read\_received\_data(rf, &msg);

/\* Write to UART RX for display on a RS232 terminal \*/

process\_message((char \*)msg.data);

}

nRF24L01\_listen(rf, 0, address);

}

}

return 0;

}

nRF24L01 \*setup\_rf(void) {

nRF24L01 \*rf = nRF24L01\_init();

//DDRB |= (1<<DDB1);

rf->ss.port = &PORTL;

rf->ss.pin = PINL1;

rf->ce.port = &PORTB;

rf->ce.pin = PINB0;

rf->sck.port = &PORTB;

rf->sck.pin = PINB1;

rf->mosi.port = &PORTB;

rf->mosi.pin = PINB2;

rf->miso.port = &PORTB;

rf->miso.pin = PINB3;

EICRA |= (1<<ISC01);

EIMSK |= (1<<INT0);

nRF24L01\_begin(rf);

return rf;

}

void process\_message(char \*message) {

lis3dhacceleration acceleration;

char printBuffer[64];

*memcpy*(&acceleration, message, sizeof(acceleration));

*sprintf*(printBuffer, tempTemplate, acceleration.x\_g, acceleration.y\_g, acceleration.z\_g);

USARTSendStr(printBuffer);

}

// nRF24L01 interrupt

ISR(INT0\_vect) {

rf\_interrupt = true;

}

void usart\_init()

{

UBRR0H = (MYUBRR) >> 8;

UBRR0L = MYUBRR;

UCSR0B |= (1 << TXEN0); // Enable Transmitter

UCSR0C |= (1 << UCSZ01) | (1 << UCSZ00); // Set frame: 8data, 1 stop

}

int USART0SendByte(char u8Data)

{

//wait while previous byte is completed

while(!(UCSR0A&(1<<UDRE0))){};

// Transmit data

UDR0=u8Data;

return 0;

}

void USARTSendStr(char\* \_str)

{

int thesize=*strlen*(\_str);

for (*uint8\_t* i=0; i<thesize;i++)

{

USART0SendByte(\_str[i]);

}

};

USART:

/\*

\* usart.c

\*

\* Created: 28-Apr-18 3:16:40 PM

\*/

#include "usart.h"

void usart\_init()

{

UBRR0H = (MYUBRR) >> 8;

UBRR0L = MYUBRR;

UCSR0B |= (1 << TXEN0); // Enable transmitter

UCSR0C |= (1 << UCSZ01) | (1 << UCSZ00); // Set frame: 8data, 1 stop

}

int USART0SendByte(char u8Data)

{

//wait while previous byte is completed

while(!(UCSR0A&(1<<UDRE0))){};

// Transmit data

UDR0=u8Data;

return 0;

}

void USARTSendStr(char\* \_str)

{

int thesize=*strlen*(\_str);

for (*uint8\_t* i=0; i<thesize;i++)

{

USART0SendByte(\_str[i]);

}

}

Receiver Flow Chart:



Receiver Code:

/\*

\* Transmitter.c

\*

\* Created: 28-Apr-18 6:17:02 AM

\*/

#include <avr/io.h>

#include <avr/interrupt.h>

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

#include "nrf24l01.h"

#include "lis3dh.h"

#include "usart.h"

void setup\_timer(void);

nRF24L01 \*setup\_rf(void);

lis3dhacceleration calcAcceleration();

volatile bool rf\_interrupt = false;

volatile bool send\_message = false;

char\* tempTemplate = "TX: Values to send: x=%4.2f m/s^2 y=%4.2f m/s^2 z=%4.2f m/s^2\n";

int main(void)

{

lis3dhacceleration acceleration;

*uint8\_t* to\_address[5] = { 0x01, 0x01, 0x01, 0x01, 0x01 };

char printBuffer[64];

nRF24L01 \*rf = setup\_rf();

setup\_timer();

usart\_init();

i2c\_init();

/\* Is there a LIS3DH? \*/

*uint8\_t* deviceid=0;

lis3dh\_read\_bytes(WHO\_AM\_I, 1, &deviceid);

if (deviceid != 0x33)

{

USARTSendStr("No LIS3DH detected!\n");

return -1;

}

else

USARTSendStr("LIS3DH is available\n");

setup\_lis3dh();

sei();

while (1)

{

if (rf\_interrupt) {

rf\_interrupt = false;

int success = nRF24L01\_transmit\_success(rf);

if (success != 0)

{

nRF24L01\_flush\_transmit\_message(rf);

}

}

if (send\_message) {

send\_message = false;

acceleration = calcAcceleration();

*sprintf*(printBuffer, tempTemplate, acceleration.x\_g, acceleration.y\_g, acceleration.z\_g);

USARTSendStr(printBuffer);

nRF24L01Message msg;

*memset*(&msg,0,sizeof(msg));

/\* Copy the whole structure to be sent to receiver \*/

*memcpy*(msg.data, &acceleration, sizeof(acceleration));

msg.length = sizeof(acceleration) + 1;

nRF24L01\_transmit(rf, to\_address, &msg);

}

}

}

lis3dhacceleration calcAcceleration()

{

lis3dhacceleration accel;

*uint8\_t* accelvalues[6];

int i;

/\* Read sequentially the LIS3DH registers from OUT\_X\_L to OUT\_Z\_H in one single swallop using autoincrement \*/

lis3dh\_read\_bytes(OUT\_X\_L | 0x80,6,accelvalues); /\* 0x80 for autoincrement \*/

/\* Calculate acceleration for all 3 axis \*/

for (i=0;i<6;i=i+2)

{

/\* Second byte is the high-byte in big endian order (could have been changed in CTRL\_REG4) \*/

int tempValue = (((int)accelvalues[i+1])<<8) + (int)accelvalues[i];

float tempFloat = (float) tempValue / LIS3DH\_RANGE\_4\_G\_DIVIDER \* SENSORS\_GRAVITY\_EARTH;

switch (i)

{

case 0:

accel.x\_g = tempFloat;

break;

case 2:

accel.y\_g = tempFloat;

break;

case 4:

accel.z\_g = tempFloat;

break;

}

}

return accel;

}

nRF24L01 \*setup\_rf(void) {

nRF24L01 \*rf = nRF24L01\_init();

rf->ss.port = &PORTB;

rf->ss.pin = PINB2;

rf->ce.port = &PORTB;

rf->ce.pin = PINB1;

rf->sck.port = &PORTB;

rf->sck.pin = PINB5;

rf->mosi.port = &PORTB;

rf->mosi.pin = PINB3;

rf->miso.port = &PORTB;

rf->miso.pin = PINB4;

EICRA |= (1<<ISC01); // Falling edge generates interrupt

EIMSK |= (1<<INT0); // Use INT0 (PD2)

nRF24L01\_begin(rf);

return rf;

}

// nRF24L01 interrupt

ISR(INT0\_vect) {

rf\_interrupt = true;

}

// setup timer to trigger interrupt every second when at 16MHz

void setup\_timer(void) {

TCCR1A = 0;

// set up timer with CTC mode and prescaling = 256

TCCR1B |= (1 << WGM12)|(1 << CS12);

// initialize counter

TCNT1 = 0;

/\*

Initialize compare value

With prescalar = 256, the frequency is 16,000,000 Hz / 256 -> period = 0.000016 sec

TimerCount=Requireddelay/period -1

For example, for 1 sec delay -> TimerCount = 1/0.000016 -1 = 62499

Let's consider this as the value for our delay.

\*/

OCR1A = 62499; // value for 1 second delay

// Enable compare interrupt

TIMSK1 |= (1 << OCIE1A);

}

// each one second interrupt

ISR(TIMER1\_COMPA\_vect) {

send\_message = true;

}

Cost Analysis:

Arduino Mega board $ 38.50

<https://store.arduino.cc/usa/arduino-mega-2560-rev3>

Mega2560 Pro Mini $ 14.00

<http://www.epalsite.com/store/>

NRF24L01 Trans receiver 2 units $ 11.00

<https://www.amazon.com/gp/product/B01IK78PQA/ref=oh_aui_detailpage_o01_s00?ie=UTF8&psc=1>

DC Power Module 2 units $ 1.80

<http://www.electrodragon.com/product/mutiple-output-dc-power-module/>

LIS3DH Accelerometer $ 4.95

<https://www.adafruit.com/product/2809>

Jumper Wires $ 7.29

<https://www.amazon.com/Solderless-Flexible-Breadboard-Jumper-100pcs/dp/B005TZJ0AM/ref=pd_lpo_vtph_23_lp_tr_img_2?_encoding=UTF8&psc=1&refRID=5G72HVJJQYJJT8SMWFAB>

Total Cost $ 77.54

Conclusion:

The project goal is successfully achieved according to the design specifications. The NRF24L01 trans receiver with the antenna from MakerFocus used in this project and have a range up to 1100 meters. Very easy to connect the module to the microcontroller using the SPI protocol and nrf24l01.h library. The next breakout board used in this project is the LIS3DH accelerometer from Adafruit.LIS3DH is a low power triple-axis accelerometer. It can be interfaced with the microcontroller either SPI or I2C protocols.

I am supposed to use the Atmega328p Xplained mini board for the final project very unfortunately it didn’t work. I bought 2 more boards from microchip these boards fail too. Because of these reasons I have to switch to Arduino mega2560 boards which I had spare ones in my tool box. I explained to Dr. Venki about the situation and got the approval to use the 2560 mega boards for this project. I am using two different 2560 mega boards. For the transmitter setup I am using Mega2560 Pro Mini board and for the receiver the genuine Arduino mega2560 board from Arduino.CC. Both boards are running the above mention “C” codes. To upload “C” code from Atmel studio7 to Arduino mega 2560 boards need to use the external tool feature in the tools menu.

Let me take this opportunity to thank Dr. Venki for giving me the necessary hardware and knowledge to achieve the goal of the final project.

References

<https://learn.adafruit.com/adafruit-lis3dh-triple-axis-accelerometer-breakout/wiring-and-test>

<http://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01>

<https://www.arduino.cc/en/Guide/ArduinoMega2560>