Fall | 13

# Accelerometer RGB LED control

Vishal Shah
Rebel Sequeira
Pratiksha Patil
Pranali Dhuru
Chris Blackden

George Mason University

## Introduction

The ECE 511 course gave us the opportunity to team up and work on Texas Instrument MSP 430 Launchpad. For all of us this was the first time trying to work with MSP 430 Launch pad, so we wanted to do something which involved learning of concepts associated with MSP 430 and at the same time make the process of learning to be fun for all. So thinking forward in the direction of "cool and fun" we came to this idea of a person being able to change the color of lights by the motion of their hand. Yes, that does sound amazing and hence we decided to do it. Also, the idea involved learning of concepts, which are associated with motion sensing and wireless transmission of data, which we believe is a part of almost device around you, for example, a smartphone.

Since the idea involved wireless data transmission, there had to be a transmitter and a receiver. We decided to use the Launchpad with the accelerometer to sense the movement of a person's hand and use Xbee to transfer the data wirelessly for the transmitter part and on the receiver side have the Xbee receive the data and be connected to the Launchpad which will also have a RGB LED connected to it. This was our project for the Fall 2013 semester and we named it **Accelerometer RGB LED Control**

**Table of Contents**

## Motivation

The goal was to change the light on the RGB LED by the movement of a person's hand. Since none of us had any experience on the concepts involved we decided this could be a fun rollercoaster. While trying to look a little deeper, we thought that the challenges we could face was the smooth and successfully transition of light from one color to another based on the movement of the hand and without any kind of delay, and for that the wireless data transmission had to be on point each time.
To break it down, the first step would be to get the accelerometer's analog values to be converted into digital values because of the wireless data transmission between the transmitter and receiver, which was the second step. And along with this, the end part would be we being able to see the light changing colors smoothly.
These were the three major steps and we could use the inbuilt features of the Launchpad and select the components to be interfaced accordingly. After working out on the details and figuring out on things can possibly work, we knew that this project was a possibility. We went into the details of the above-mentioned steps and had a plan in our mind and we took a kick-start on our project.
Also, this project could be used as a children's toy, a night lamp or for a dance party. This sounded exciting and hence this project was a fun way to learn for all of us.

## Overview

On the transmitter side we are using the Launchpad with a Xbee module and an accelerometer and on the receiver side we have another Launchpad with a Xbee module and a RGB LED connected to it. Depending on the motion of the hand, the accelerometer gives the X, Y and Z values, which are analog in nature and are converted to digital by MSP430's on board ADC. The digital values are transmitted via Xbee and then mapped on the R, G and B outputs of the RGB LED. Based on these values, the Red, Green and Blue LED's light and combine to give a different color. The picture on the right shows the various colors possible with Red, Green and Blue light combinations.

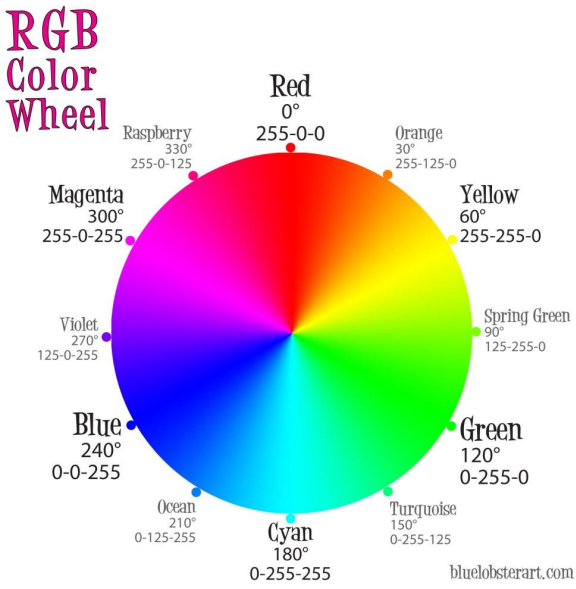
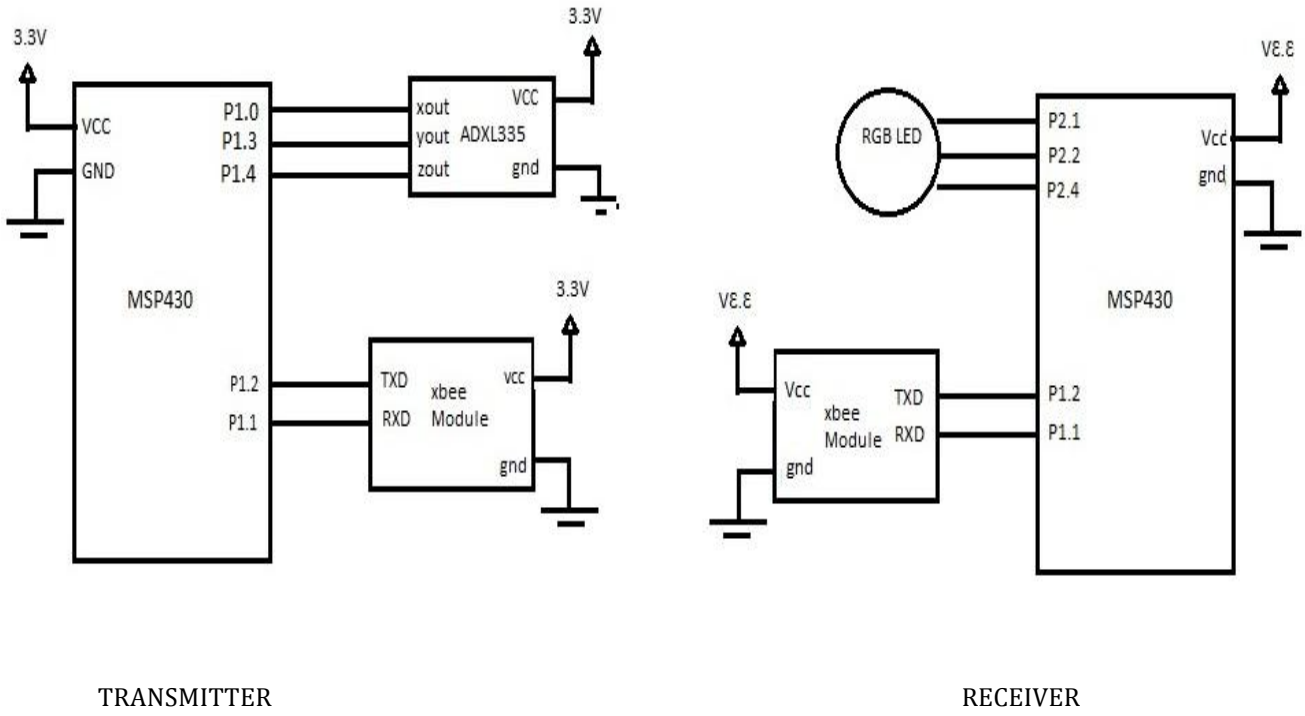



**Figure 1 - RGB Color Wheel**

**Components**

2x TI MSP430 Launchpad
1x RGB LED
1x ADXL335
2x Xbee Modules

**Block Diagram and Interfacing**




TRANSMITTER RECEIVER

**Figure 2 - Block Diagram**

The picture above shows the block diagram of the Accelerometer RGB LED Control. It shows the pin connections and the Vcc and GND connections. Let us look at each part in detail

**Hardware Interface**

**Transmitter**

**MSP430 Launchpad** – The Launchpad is given 3.3V Vcc and is connected to the ground. The pins P1.0, P1.3 and P1.4 of the Launchpad are the ADC10 analog input A0, A1 and A4 respectively, which are used here to get the analog values of X, Y amd Z axis from the accelerometer which is connected to it. These analog values are the inputs to the 10 bit ADC which is onboard MSP430. Pins P1.1, P1.2 are the UART receive data input and UART transmit data output pins respectively and are connected to the Xbee transmit and receive pins for wireless data transmission.

**ADXL335** – The ADXL335 is given 3.3V Vcc and is connected to ground. The ADXL 335 has 3 pins namely xout, yout and zout which gives out analog values in those directions based on the movement. These pins giving the analog output is connected to the ADC10 input pins of the Launchpad.

**Xbee Module** – The Xbee modules are used for the wireless transmission of data, hence the transmit and receive pins of the Xbee are connected to the transmit and receive pins of the MSP430. The Vcc is 3.3V and the Xbee module is connected to ground.

**Receiver**

**MSP430 Launchpad** – The P1.1, P1.2 pins which are the receiver and transmitter pins respectively are connected to the Xbee module, in order to receive the ADC value from the transmitter. The pins ****

**RGB LED** – The 3 pins of the LED R, G and B are given Vcc while the 4$^{th}$ pin is connected to the ground.

**Xbee Module** – The transmit and receive pins of the Xbee is connected to the respective pins of the MSP430 to pass on received data to the MSP430.

**Software Interface**

**ADXL335 and MSP430 Launchpad**

The Accelerometer ADXL 335 provides Analog values In the form of individual pins X,Y and Z determining the 3 axes. The goal was to extract these values via MSP 430 analog channels and convert it into digital. As the MSP 430 has a 10-bit ADC these values were converted over a range of 0 to 1024. Each axes of the ADC has a 0g point above which the values change in the positive range and below which the values change in the negative range. This value is found to be around 1.45V and 1.55V. On conversion on a 0-1024 range is around 450-490. At this range the Accelerometer is at 0g with no color output. When it goes above 1.55V the variations are prominent from 1.55V to 2 V and below 1.45V in the range of 1V to 1.45V. These were maped accordingly in the 10-bit range creating a smooth transition using a mapping function.

Mapping function:

```
long map(long x, long in_min, long in_max, long out_min, long out_max)
{
 return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}
```

Example for xpin:

```
 if(xpin>490)
 {
  analogValue=map(xpin,490,600,0,150);
  TA1CCR1 = analogValue;
 }
 else if(xpin<450)
 {
  analogValue=map(xpin,340,450,150,0);
  TA1CCR1 = analogValue;
 }
 else
 {
  TA1CCR1=0;
 }

  if(zpin>490)
 {
  analogValue=map(zpin,490,600,0,150);
  TA1CCR2 = analogValue;
 }
 else if(zpin<450)
 {
  analogValue=map(zpin,340,450,150,0);
  TA1CCR2 = analogValue;
 }
 else
 {
```

```
  TA1CCR2=0;
 }.
```

**RGB LED**

The RGB LED changed the intensity using a PWM. We connected the RGB LED pins to each of the axis X, Y and Z respectively. We provided a max duty cycle of 110 generated at the output of the mapping function. When we put these value variations based on the movement of the accelerometer it created a smooth transition in colors depending on the axes. This created a combination of colors in different axes creating a number of secondary colors due to a mixture of the 2 major colors.

**Xbee**

We were using the Xbee Series 2 Module by Digi International. In order for the 2 Xbee's to be able to communicate with each other, we need to configure the Xbee's with the correct firmware first. One of the Xbee has to be installed with a Coordinator firmware so that the particular Xbee can act as a coordinator point for the other Xbees which will be the end devices, which means the router. Any Xbee that is trying to get connected to the coordinator Xbee in order to send and receive data has to first have Router firmware installed on the Xbee. Hence, after one Xbee being configured as a coordinator the remaining other Xbee's trying to communicate with the coordinator has to have the Router firmware installed. The firmwares can be installed on the Xbee using X-CTU software which is available for Windows Operating System. Using this software, one can select from a list of available firmwares which can be installed on the Xbee modules. The Xbee module can be configured using Xbee USB Dongle or Arduino. We used the Arduino to configure the Xbees.
After the Xbee's have been configured with the correct firmwares, the Xbee's now have to know who is the coordinator and who is the router. In other words, Xbee's parameters like channel ID and PAN ID have to be same for the both them so that they belong to the same network. After which, the coordinator should have the router's address stored in it's destination address field and the router should have the coordinator's address in it's destination address field. This configuring of addresses can be done using X-CTU software for Windows Operating System.

```
//-------------------------------------------------------------------------
// Outputs one byte using the Timer_A UART
//-------------------------------------------------------------------------
void TimerA_UART_tx(unsigned char byte)
{
  while (TACCTL0 & CCIE);             // Ensure last char got TX'd
  TACCR0 = TAR;                       // Current state of TA counter
  TACCR0 += UART_TBIT;                // One bit time till first bit
  TACCTL0 = OUTMOD0 + CCIE;           // Set TXD on EQU0, Int
  txData = byte;                      // Load global variable
  txData |= 0x100;                    // Add mark stop bit to TXData
  txData <<= 1;                       // Add space start bit
}

//-------------------------------------------------------------------------
// Timer_A UART - Transmit Interrupt Handler
```

```
//-------------------------------------------------------------------------------
#pragma vector = TIMER0_A0_VECTOR
__interrupt void Timer_A0_ISR(void)
{
  static unsigned char txBitCnt = 10;

  TACCR0 += UART_TBIT;                  // Add Offset to CCRx
  if (txBitCnt == 0) {                  // All bits TXed?
    TACCTL0 &= ~CCIE;                   // All bits TXed, disable interrupt
    txBitCnt = 10;                      // Re-load bit counter
  }
  else {
    if (txData & 0x01) {
      TACCTL0 &= ~OUTMOD2;              // TX Mark '1'
    }
    else {
      TACCTL0 |= OUTMOD2;               // TX Space '0'
    }
    txData >>= 1;
    txBitCnt--;
  }
}
//-------------------------------------------------------------------------------
// Timer_A UART - Receive Interrupt Handler
//-------------------------------------------------------------------------------
#pragma vector = TIMER0_A1_VECTOR
__interrupt void Timer_A1_ISR(void)
{
  static unsigned char rxBitCnt = 8;
  static unsigned char rxData = 0;



  switch (__even_in_range(TA0IV, TA0IV_TAIFG)) { // Use calculated branching
    case TA0IV_TACCR1:                  // TACCR1 CCIFG - UART RX
      TACCR1 += UART_TBIT;              // Add Offset to CCRx
      if (TACCTL1 & CAP) {              // Capture mode = start bit edge
        TACCTL1 &= ~CAP;                // Switch capture to compare mode
        TACCR1 += UART_TBIT_DIV_2;      // Point CCRx to middle of D0
      }
      else {
        rxData >>= 1;
        if (TACCTL1 & SCCI) {           // Get bit waiting in receive latch
          rxData |= 0x80;
        }
        rxBitCnt--;
        if (rxBitCnt == 0) {            // All bits RXed?
```

```
            rxBuffer = rxData;          // Store in global variable
            rxBitCnt = 8;               // Re-load bit counter
            TACCTL1 |= CAP;             // Switch compare to capture mode
            __bic_SR_register_on_exit(LPM0_bits);  // Clear LPM0 bits from 0(SR)
          }
        }
        if(rxBuffer=='A')
        {
         count_rx=0;
         count_rx++;
        }
        break;

  }
}
```

**Flow Chart**

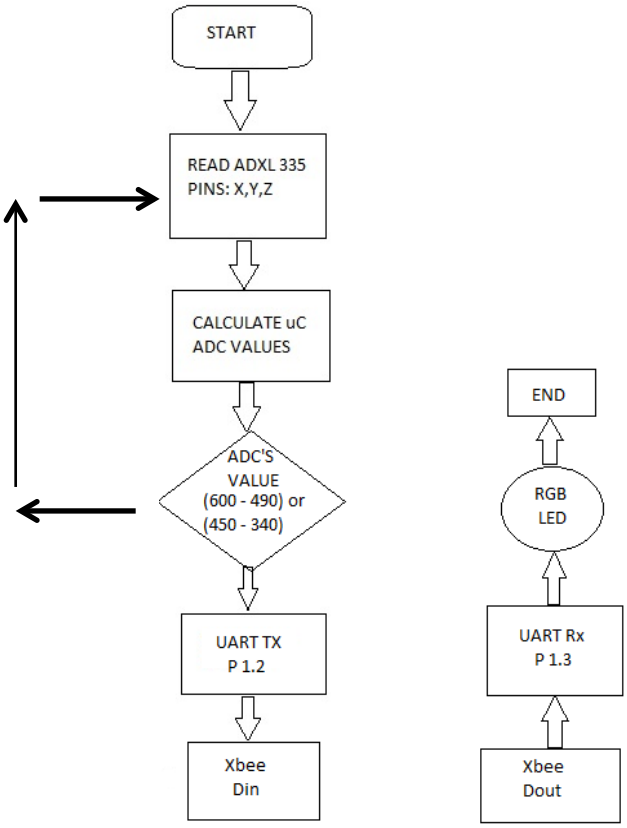



**Figure 3 – Flow Chart**

## Results and Conclusions

### MSP430 and ADXL 335

To break down our goal into 2 major parts, it would be to achieve the goal first with wired connections and then go wireless with it. First we started with trying to calibrate the accelerometer with the MSP430. We kept watching the values at the output of the accelerometer and trying to find the digital transmission period using debug mode. After finding a constant range in the variation and agreeing on certain range of values we used some mathematics to convert this range in digital.

### MSP430 and RGB LED

After the digital values were visible the LED had to just receive these values in Analog again. Using a PWM and a duty cycle of 110 it was quite convenient to vary these values and implement them onto the RGB LED changing colors based on the direction.

### Xbee Module

After the goal been achieved with wired connections, we wanted to make it go wireless. We initially started with configuring the Xbees using the MSP430, which acted like an USB Dongle to connect the Xbee to a PC. The Xbee was connected to the MSP430 using it's UART pins being connected to the Xbee transmit and receive pins. The Xbee was detected on the PC but the remaining pins such as RTS and CTS of the Xbee couldn't get enough response from the MSP430 which, by the time we realized, we had already blown one of the Xbees. We managed to order another Xbee on one day shipping (since we did not have enough days until the presentation) after not being able to find it at any stores in the area, while we configured the other Xbee using Arduino as an alternative approach, which worked out perfectly fine.
After the Xbees were configured with the correct firmwares and addresses, we tried on a dummy code to test the communication between the 2 Xbees. The dummy code executed perfectly on both the MSP430s and we were able to transmit stable values. But as soon as we started transmitting varying values through the aceelerometer the colors just started flickering with no transition on the output.

### PCB

We designed the PCB for the transmitter and the receiver parts. However, due to lack of time we were only able to order the transmitter part and have it shipped to us. We soldered the socket of the IC of the MSP430 Launchpad on the PCB and also got the accelerometer on it. Only if we could get the Xbee working and tested on time then we could solder it and have the entire PCB ready for the transmitter part.

**Lessons Learnt**

We under estimated the process of configuring the Xbees with MSP430. We should have immediately tried an alternate way to configure the Xbees without wasting further time with trying to configure it with the MSP430.
Under estimated PCB fabrication and shipping time.
Time management

**Conclusion**

We could achieve our goal with wired connections perfectly and be able to see the smooth transition of one color to another and make it look natural. However, we were not able to implement the Xbees due to facing challenges in having to configure them and then later were able to transmit the values on the other end of the xbee but the output of the LED turn out to be just a flickering light on each of the LED pins due to incorrect and non-varying transmission of data..

## Appendix

### Team Members and Task Division

Chris Blackden – Hardware construction, PCB layout
Pranali Dhuru – Xbee Configuration
Pratiksha Patil – Xbee Configuration
Rebel Sequeira – Software implementation
Vishal Shah – Software Implementation

### Components

2x Texas Instrument MSP430 Launchpad

Product link - http://www.ti.com/tool/msp-exp430g2
Data Sheet - http://www.ti.com/lit/ds/slas735j/slas735j.pdf

2x Xbee Modules

Data Sheet - ftp://ftp1.digi.com/support/documentation/90000866_A.pdf
Product link - http://www.digi.com/products/model?mid=3008

ADXL 335

Data Sheet – http://www.analog.com/static/imported-files/data_sheets/ADXL335.pdf

RGB LED

**Full Schematic**

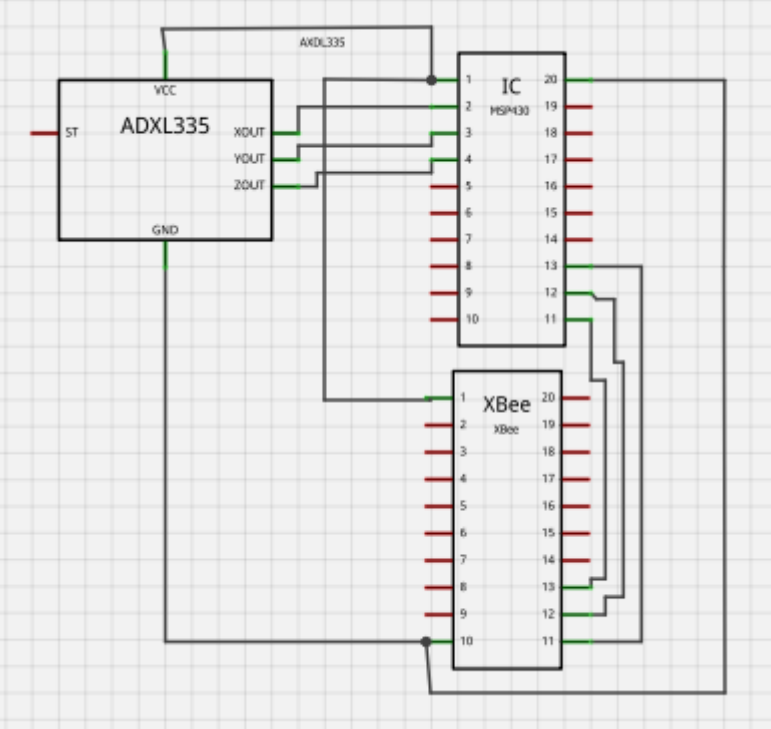



**Figure 4 – Transmitter Schematic**

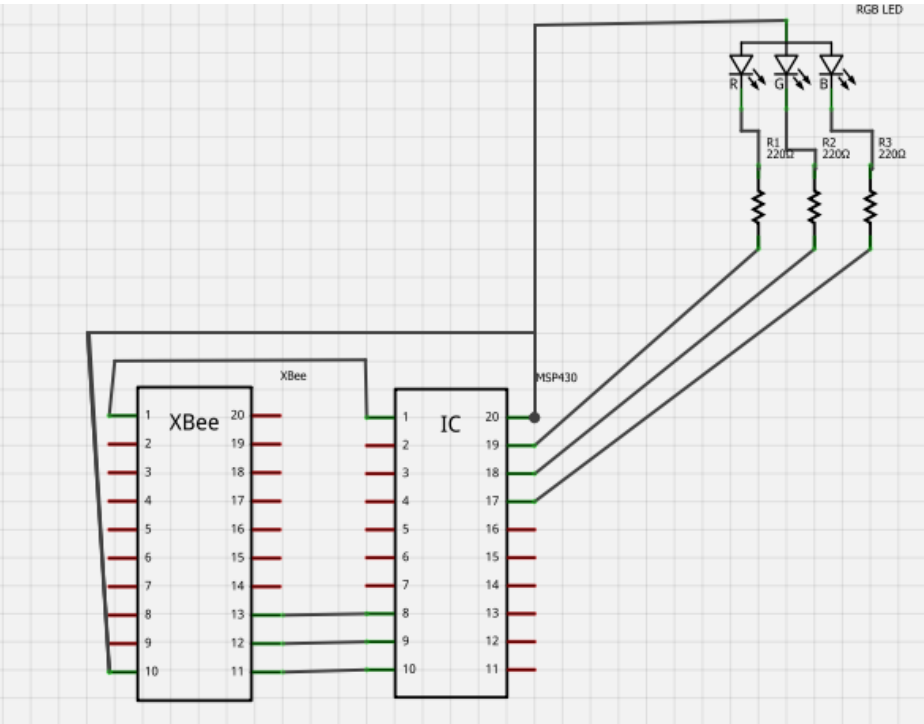



**Figure 5 – Receiver Schematic**

**PCB Layout**

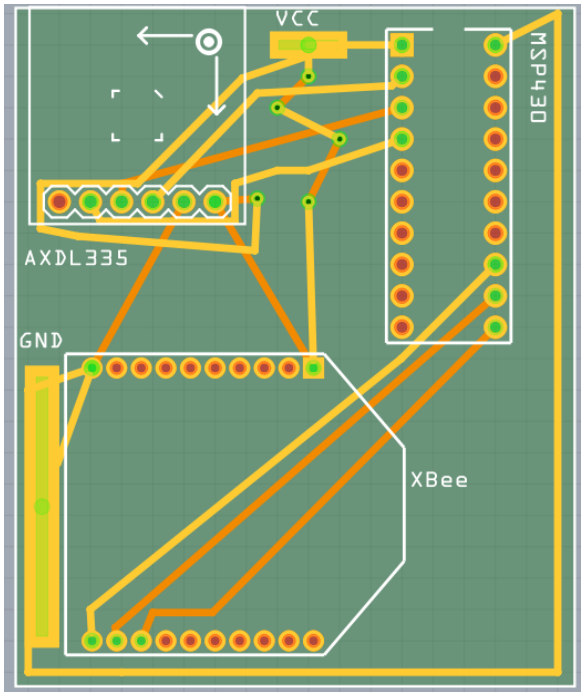



**Figure 6 – PCB Transmitter**

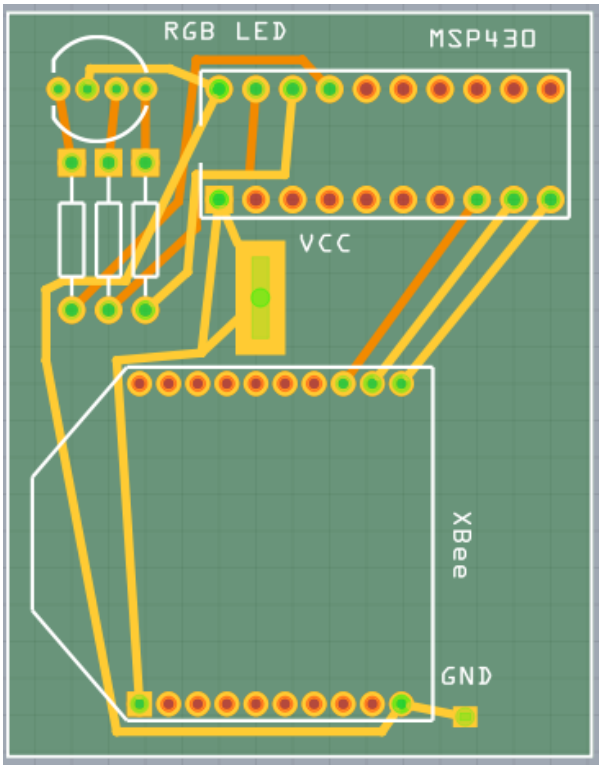



**Figure 7 – PCB Receiver**

**SOURCE CODE**

**Full Transmitter Code:**

```
#include "msp430g2553.h"
#include "adc10.h"
#include "wdt.h"
#include "gpio.h"
#include "stdbool.h"
#include "timera.h"

#define BAUDRATE   9600



//-------------------------------------------------------------------------
// Hardware-related definitions
//-------------------------------------------------------------------------
#define UART_TXD   0x02                 // TXD on P1.1 (Timer0_A.OUT0)
#define UART_RXD   0x04                 // RXD on P1.2 (Timer0_A.CCI1A)

//-------------------------------------------------------------------------
// Conditions for BAUDRATE Baud SW UART, SMCLK = 1MHz
//-------------------------------------------------------------------------


#define UART_TBIT_DIV_2     (1000000 / (BAUDRATE * 2))
#define UART_TBIT           (1000000 / BAUDRATE)

//-------------------------------------------------------------------------
// Global variables used for full-duplex UART communication
//-------------------------------------------------------------------------
unsigned int txData;                    // UART internal variable for TX
unsigned char rxBuffer;                 // Received UART character
unsigned int count_rx=0;
int ADCvalueRX;
//-------------------------------------------------------------------------
// Function prototypes
//-------------------------------------------------------------------------
void TimerA_UART_init(void);
void TimerA_UART_tx(unsigned char byte);


int k = 0;
unsigned int ADCValue;   // Measured ADC Value
/**
* Reads ADC 'chan' once using AVCC as the reference.
```

```
**/

void Single_Measure(unsigned int chan)
{
 ADC10CTL0 &= ~ENC;                                  // Disable ADC
 ADC10CTL0 = ADC10SHT_3 + ADC10ON + ADC10IE;         // 16 clock ticks, ADC On, enable ADC interrupt
 ADC10CTL1 = ADC10SSEL_3 + chan;                     // Set 'chan', SMCLK
 ADC10CTL0 |= ENC + ADC10SC;               // Enable and start conversion
}
//-------------------------------------------------------------------------------
// Function configures Timer_A for full-duplex UART operation
//-------------------------------------------------------------------------------
void TimerA_UART_init(void)
{
 TACCTL0 = OUT;                      // Set TXD Idle as Mark = '1'
 TACCTL1 = SCS + CM1 + CAP + CCIE;      // Sync, Neg Edge, Capture, Int
 TACTL = TASSEL_2 + MC_2;              // SMCLK, start in continuous mode
}
//-------------------------------------------------------------------------------
// Outputs one byte using the Timer_A UART
//-------------------------------------------------------------------------------
void TimerA_UART_tx(unsigned char byte)
{
 while (TACCTL0 & CCIE);               // Ensure last char got TX'd
 TACCR0 = TAR;                       // Current state of TA counter
 TACCR0 += UART_TBIT;                  // One bit time till first bit
 TACCTL0 = OUTMOD0 + CCIE;             // Set TXD on EQU0, Int
 txData = byte;                      // Load global variable
 txData |= 0x100;                    // Add mark stop bit to TXData
 txData <<= 1;                       // Add space start bit
}

//-------------------------------------------------------------------------------
// Timer_A UART - Transmit Interrupt Handler
//-------------------------------------------------------------------------------
#pragma vector = TIMER0_A0_VECTOR
__interrupt void Timer_A0_ISR(void)
{
 static unsigned char txBitCnt = 10;

 TACCR0 += UART_TBIT;                  // Add Offset to CCRx
 if (txBitCnt == 0) {                // All bits TXed?
   TACCTL0 &= ~CCIE;                 // All bits TXed, disable interrupt
   txBitCnt = 10;                    // Re-load bit counter
 }
 else {
   if (txData & 0x01) {
```

```
      TACCTL0 &= ~OUTMOD2;            // TX Mark '1'
    }
    else {
      TACCTL0 |= OUTMOD2;             // TX Space '0'
    }
    txData >>= 1;
      txBitCnt--;
  }
}
/**
* ADC interrupt routine. Pulls CPU out of sleep mode for the main loop.
**/
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR (void)
{
   ADCValue = ADC10MEM;                              // Saves measured value.
   __bic_SR_register_on_exit(CPUOFF);        // Enable CPU so the main while loop continues
}


long map(long x, long in_min, long in_max, long out_min, long out_max)
{
 return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}


int Acc(char channel)
{

 int res, ch;

 ch = channel * 0x1000;

 ADC10CTL0 &= ~ENC;
 ADC10CTL0 |= ADC10ON;
 ADC10CTL1 = ch + ADC10SSEL_3;

 ADC10CTL0 |= ENC + ADC10SC;

 while(ADC10CTL1&ADC10BUSY);

 res = ADC10MEM;
 ADC10CTL0 &= ~(ADC10ON + ENC);

 return res;
}
```

```
void uart_converter(int analogValue)
{
 {
    char ADCvalue1;
  char ADCvalue2;

  char ADCvalueRX1;
  char ADCvalueRX2;


   ADCvalue1 = analogValue & 0xFF;
     ADCvalue2 = analogValue >> 8;

     // Echo received character
      TimerA_UART_tx('A');
       TimerA_UART_tx(ADCvalue1);
       TimerA_UART_tx(ADCvalue2);


      for(k = 0; k<10000; k++);

   }
}

void main()
{
 unsigned int xpin;
 unsigned int ypin;
 unsigned int zpin;


 int analogValue;

 WDTCTL = WDTPW + WDTHOLD;                              //Watchdog Timer Disabled

 P1DIR =0xE6;                                  //Initialise Pins and Special Functions
 P1SEL=0x00;
 P1IE=0x00;
 P2DIR |= BIT1;                  // P2.1 to output
 P2SEL |= BIT1;                  // P2.1 to TA1.1

 P2DIR |= BIT2;                  // P2.2 to output
 P2SEL |= BIT2;                  // P2.2 to TA1.1

 P2DIR |= BIT4;                  // P2.4 to output
 P2SEL |= BIT4;                  // P2.4 to TA1.2
```

```
P2DIR |= BIT5;                          // P2.5 to output
P2SEL |= BIT5;                          // P2.5 to TA1.2


DCOCTL= CALDCO_1MHZ;
BCSCTL1=CALBC1_1MHZ;

P1OUT=0x00;
BCSCTL3 = (BCSCTL3 & ~(BIT4+BIT5)) | LFXT1S_2;          // use internal VLO for ACLK
P2SEL  &= ~(BIT6+BIT7);                         // PxSEL=0, PxSEL2=0 for I/O use
P2SEL2 &= ~(BIT6+BIT7);                          // PxSEL=0, PxSEL2=0 for I/O use
P2DIR |= BIT6;                          // P2.6 to output
P2SEL |= BIT6;                          // P2.6 to TA0.1
P1SEL = UART_TXD + UART_RXD;          // Timer function for TXD/RXD pins
P1DIR = 0xFF & ~UART_RXD;             // Set all pins but RXD to output
while(1)
{

 xpin=Acc(0);
 ypin=Acc(3);
 zpin=Acc(4);

    TA1CCR0 = 110;                          // PWM Period
 TA1CCTL1 = OUTMOD_7;                       // CCR1 reset/set
 TA1CCTL2 = OUTMOD_7;                       // CCR1 reset/set

  uart_converter(450);
 if(xpin>490)
{
 analogValue=map(xpin,490,600,0,150);
 uart_converter(analogValue);
}
else if(xpin<450)
{
 analogValue=map(xpin,340,450,150,0);
 uart_converter(analogValue);
}
else
{
 uart_converter(0);
}

  if(ypin>490)
{
 analogValue=map(ypin,490,600,0,150);
 uart_converter(analogValue);
}
```

```
 else if(ypin<450)
 {
  analogValue=map(ypin,340,450,150,0);

  uart_converter(analogValue);
 }
 else
 {
  uart_converter(0);
 }
  if(zpin>490)
 {
  analogValue=map(zpin,490,600,0,150);
  uart_converter(analogValue);
 }
 else if(zpin<450)
 {
  analogValue=map(zpin,340,450,150,0);
  uart_converter(analogValue);
 }
 else
 {
  uart_converter(0);
 }
 TA1CTL = TASSEL_2 + MC_1;                  // SMCLK, up mode


 }
}
```

**Full Receiver code:**

```
#include "msp430g2553.h"
//#include "central.h"
#include "adc10.h"
#include "wdt.h"
#include "gpio.h"
#include "stdbool.h"
#include "timera.h"

#define BAUDRATE   9600

//-----------------------------------------------------------------------
// Hardware-related definitions
//-----------------------------------------------------------------------
#define UART_TXD   0x02                    // TXD on P1.1 (Timer0_A.OUT0)
#define UART_RXD   0x04                    // RXD on P1.2 (Timer0_A.CCI1A)
```

```
//-------------------------------------------------------------------------
// Conditions for BAUDRATE Baud SW UART, SMCLK = 1MHz
//-------------------------------------------------------------------------


#define UART_TBIT_DIV_2     (1000000 / (BAUDRATE * 2))
#define UART_TBIT           (1000000 / BAUDRATE)

//-------------------------------------------------------------------------
// Global variables used for full-duplex UART communication
//-------------------------------------------------------------------------
unsigned int txData;                    // UART internal variable for TX
unsigned char rxBuffer;                 // Received UART character
unsigned int count_rx=0;
  int ADCvalueRX;
//-------------------------------------------------------------------------
// Function prototypes
//-------------------------------------------------------------------------
void TimerA_UART_init(void);
void TimerA_UART_tx(unsigned char byte);


int k = 0;
unsigned int ADCValue;   // Measured ADC Value
/**
* Reads ADC 'chan' once using AVCC as the reference.
**/

//-------------------------------------------------------------------------
// Function configures Timer_A for full-duplex UART operation
//-------------------------------------------------------------------------
void TimerA_UART_init(void)
{
   TACCTL0 = OUT;                       // Set TXD Idle as Mark = '1'
   TACCTL1 = SCS + CM1 + CAP + CCIE;      // Sync, Neg Edge, Capture, Int
   TACTL = TASSEL_2 + MC_2;             // SMCLK, start in continuous mode
}
//-------------------------------------------------------------------------
// Outputs one byte using the Timer_A UART
//-------------------------------------------------------------------------
void TimerA_UART_tx(unsigned char byte)
{
   while (TACCTL0 & CCIE);              // Ensure last char got TX'd
   TACCR0 = TAR;                       // Current state of TA counter
   TACCR0 += UART_TBIT;                 // One bit time till first bit
   TACCTL0 = OUTMOD0 + CCIE;            // Set TXD on EQU0, Int
```

```
  txData = byte;                    // Load global variable
  txData |= 0x100;                  // Add mark stop bit to TXData
  txData <<= 1;                     // Add space start bit
}

//-------------------------------------------------------------------------------
// Timer_A UART - Transmit Interrupt Handler
//-------------------------------------------------------------------------------
#pragma vector = TIMER0_A0_VECTOR
__interrupt void Timer_A0_ISR(void)
{
  static unsigned char txBitCnt = 10;

  TACCR0 += UART_TBIT;              // Add Offset to CCRx
  if (txBitCnt == 0) {              // All bits TXed?
    TACCTL0 &= ~CCIE;               // All bits TXed, disable interrupt
    txBitCnt = 10;                  // Re-load bit counter
  }
  else {
    if (txData & 0x01) {
      TACCTL0 &= ~OUTMOD2;          // TX Mark '1'
    }
    else {
      TACCTL0 |= OUTMOD2;           // TX Space '0'
    }
    txData >>= 1;
    txBitCnt--;
  }
}
//-------------------------------------------------------------------------------
// Timer_A UART - Receive Interrupt Handler
//-------------------------------------------------------------------------------
#pragma vector = TIMER0_A1_VECTOR
__interrupt void Timer_A1_ISR(void)
{
  static unsigned char rxBitCnt = 8;
  static unsigned char rxData = 0;



  switch (__even_in_range(TA0IV, TA0IV_TAIFG)) { // Use calculated branching
    case TA0IV_TACCR1:              // TACCR1 CCIFG - UART RX
      TACCR1 += UART_TBIT;          // Add Offset to CCRx
      if (TACCTL1 & CAP) {          // Capture mode = start bit edge
        TACCTL1 &= ~CAP;            // Switch capture to compare mode
        TACCR1 += UART_TBIT_DIV_2;  // Point CCRx to middle of D0
      }
```

```
        else {
          rxData >>= 1;
          if (TACCTL1 & SCCI) {          // Get bit waiting in receive latch
            rxData |= 0x80;
          }
          rxBitCnt--;
          if (rxBitCnt == 0) {           // All bits RXed?
            rxBuffer = rxData;           // Store in global variable
            rxBitCnt = 8;               // Re-load bit counter
            TACCTL1 |= CAP;              // Switch compare to capture mode
            __bic_SR_register_on_exit(LPM0_bits);  // Clear LPM0 bits from 0(SR)
          }
        }
         if(rxBuffer=='A')
        {
          count_rx=1;
          count_rx++;
        }
        break;

  }
}
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{
 while (!(IFG2&UCA0TXIFG));              // USCI_A0 TX buffer ready?
 TA1CCR1  = UCA0RXBUF;                   // TX -> RXed character
}
void main()
{
  unsigned int xpin;
  unsigned int ypin;
  unsigned int zpin;
          char ADCvalue1;
  char ADCvalue2;

  char ADCvalueRX1;
  char ADCvalueRX2;

  int analogValue;

  WDTCTL = WDTPW + WDTHOLD;                                  //Watchdog Timer Disabled

  P1DIR =0xE6;                                        //Initialise Pins and Special Functions
  P1SEL=0x00;
  P1IE=0x00;
  P2DIR |= BIT1;                                    // P2.1 to output
```

```
  P2SEL |= BIT1;                            // P2.1 to TA1.1

  P2DIR |= BIT2;                            // P2.2 to output
  P2SEL |= BIT2;                            // P2.2 to TA1.1

  P2DIR |= BIT4;                            // P2.4 to output
  P2SEL |= BIT4;                            // P2.4 to TA1.2

  P2DIR |= BIT5;                            // P2.5 to output
  P2SEL |= BIT5;                            // P2.5 to TA1.2


  DCOCTL= CALDCO_1MHZ;
  BCSCTL1=CALBC1_1MHZ;


  P1OUT=0x00;
  BCSCTL3 = (BCSCTL3 & ~(BIT4+BIT5)) | LFXT1S_2;        // use internal VLO for ACLK
  P2SEL  &= ~(BIT6+BIT7);                    // PxSEL=0, PxSEL2=0 for I/O use
  P2SEL2 &= ~(BIT6+BIT7);                    // PxSEL=0, PxSEL2=0 for I/O use
  P2DIR |= BIT6;                            // P2.6 to output
  P2SEL |= BIT6;                            // P2.6 to TA0.1
  P1SEL = UART_TXD + UART_RXD;          // Timer function for TXD/RXD pins
  P1DIR = 0xFF & ~UART_RXD;             // Set all pins but RXD to output
  while(1)
 {

  TA1CCR0 = 110;                            // PWM Period
  TA1CCTL1 = OUTMOD_7;                      // CCR1 reset/set
  TA1CCTL2 = OUTMOD_7;                      // CCR1 reset/set



  TA1CTL = TASSEL_2 + MC_1;                 // SMCLK, up mode


  if(count_rx==2)
  TA1CCR1  =  rxBuffer;
  if(count_rx==2)
  TA1CCR2  =  rxBuffer;


      for(k = 0; k<10000; k++);

}
 }
```