**INDOOR LOCALIZATION**

**THE UNIVERSITY OF TEXAS AT ARLINGTON**

**EE6314-ADVANCED EMBEDDED MICROCONTROLLER SYSTEM**

**REPRESENTED BY RF TEAM**

**Submitted By-**

Vimal Vankar

Dhaval Padia

Disha Shah

Pooja Katiyar

Luv Sachdeva

Nihit Bhavsar

Anupama Malleshappa

Introduction

This paper represents the implementation of Indoor Localization, the goal of which is to find out the position of a mobile object in indoor areas. The project uses ultrasonic transmitter and receivers to determine that, the operating frequency of which is 24.2kHz. RF modules are used to provide the reference sync signal which operates at the frequency of 418MHz.

Interfacing Diagram

1. Reference Source

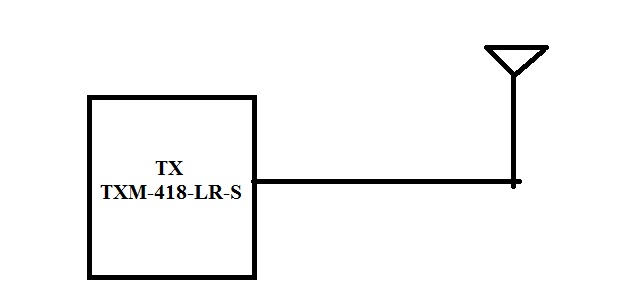
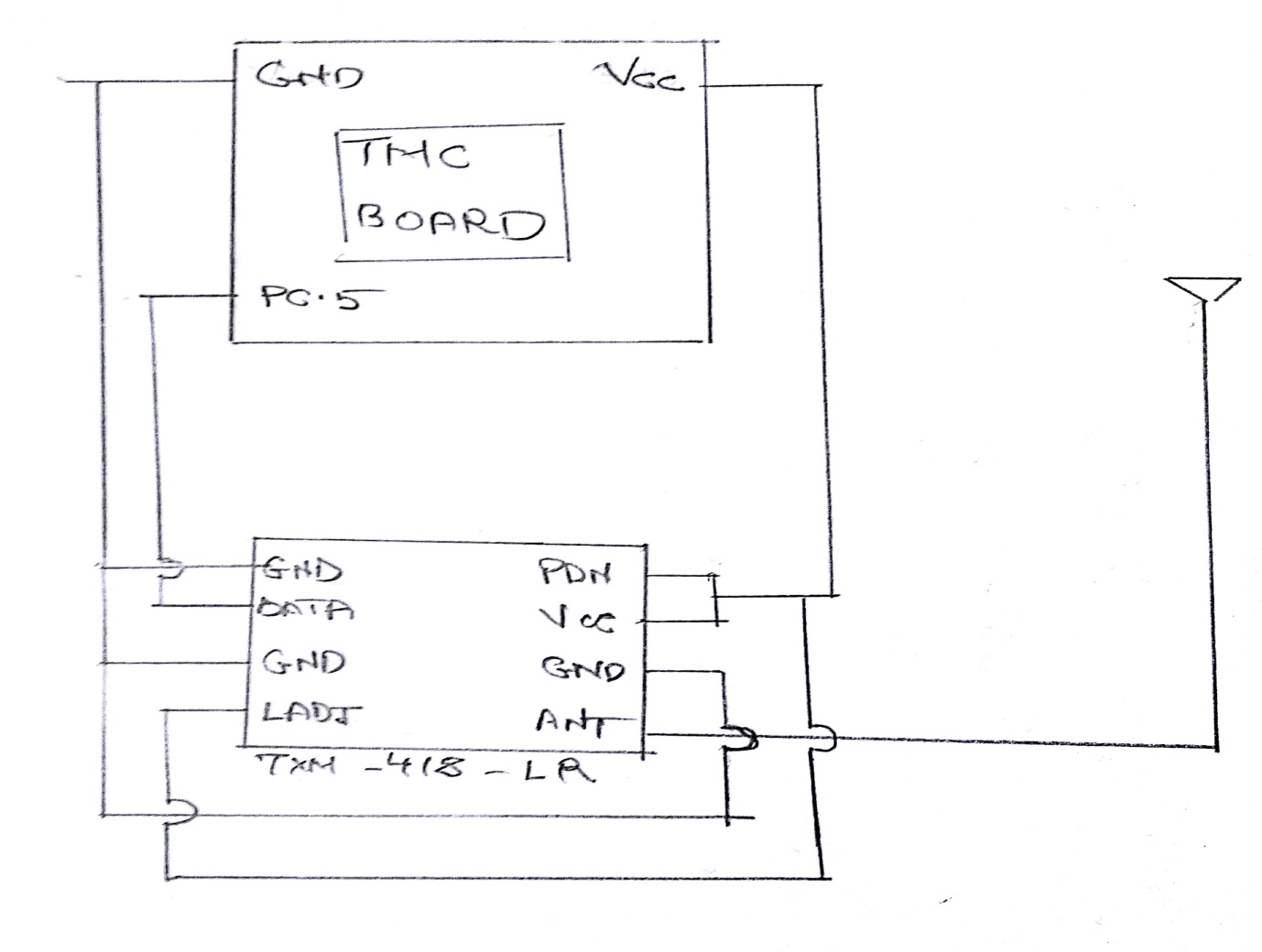


Figure.1: Transmitter block diagram

 Figure.2: Transmitter Circuit diagram

The reference source uses TXM-418-LR RF module to transmit the sync signal and the data packet containing the information such as the room number, x, y and z location of the three beacons.

1. Ultrasonic Transmitters

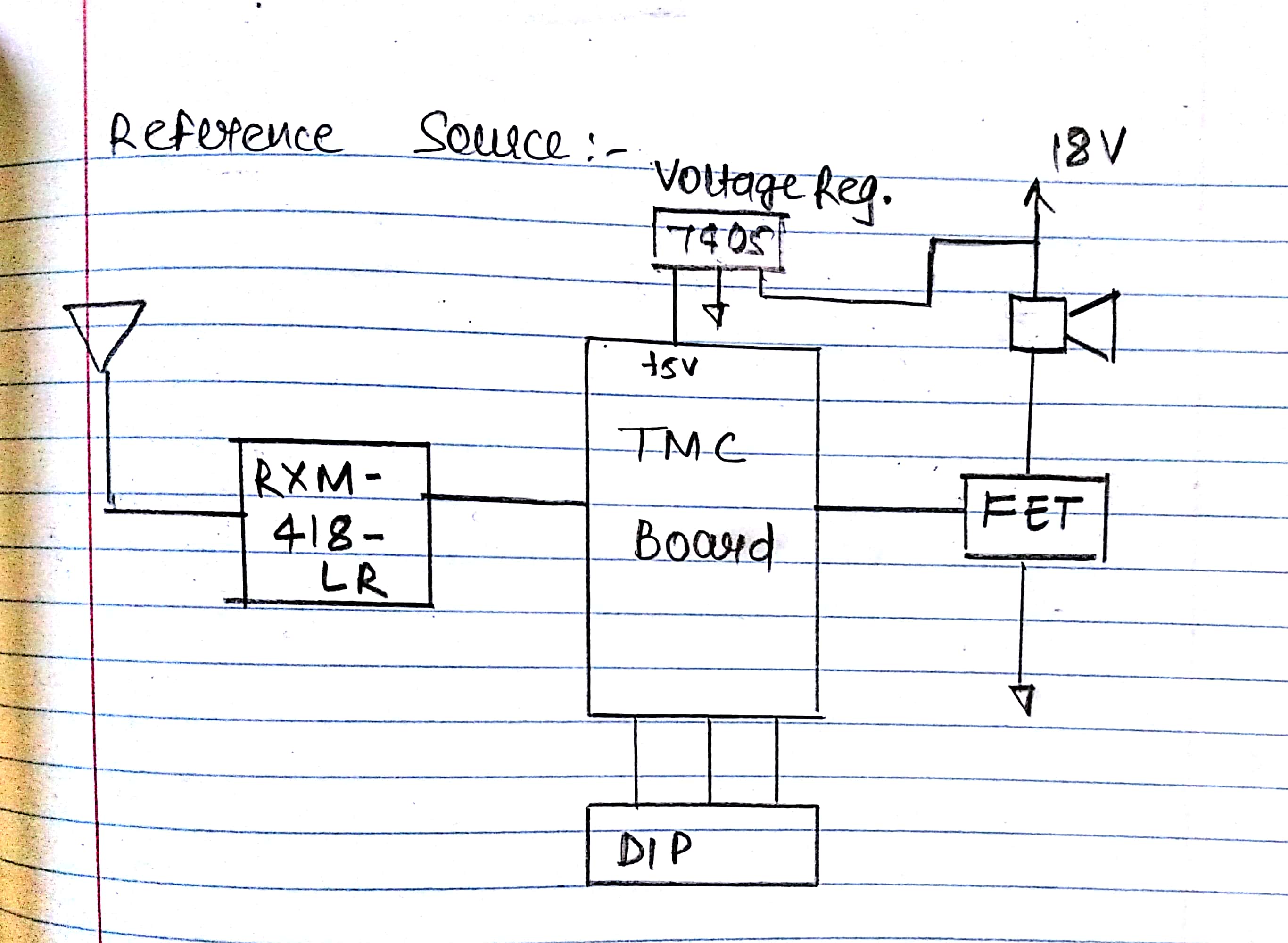


Figure.3: Ultrasonic transmitter block diagram

In ultrasonic transmitter, we used RXM-14-LR to receive the sync signal and applied the logic for sync detection. So, as soon as the sync is detected, according to the IDs of each beacon they will start transmitting. For generating the 24,.2 kHz square wave from microcontroller board, PWM is used which is used as the trigger to FET. Ultrasonic speakers are connected to FET in series with 18V power supply, this will result in to the square of 30 burst.

7405 voltage regulator is used in order to give constant 5V to the microcontroller board and 18V to the speaker.

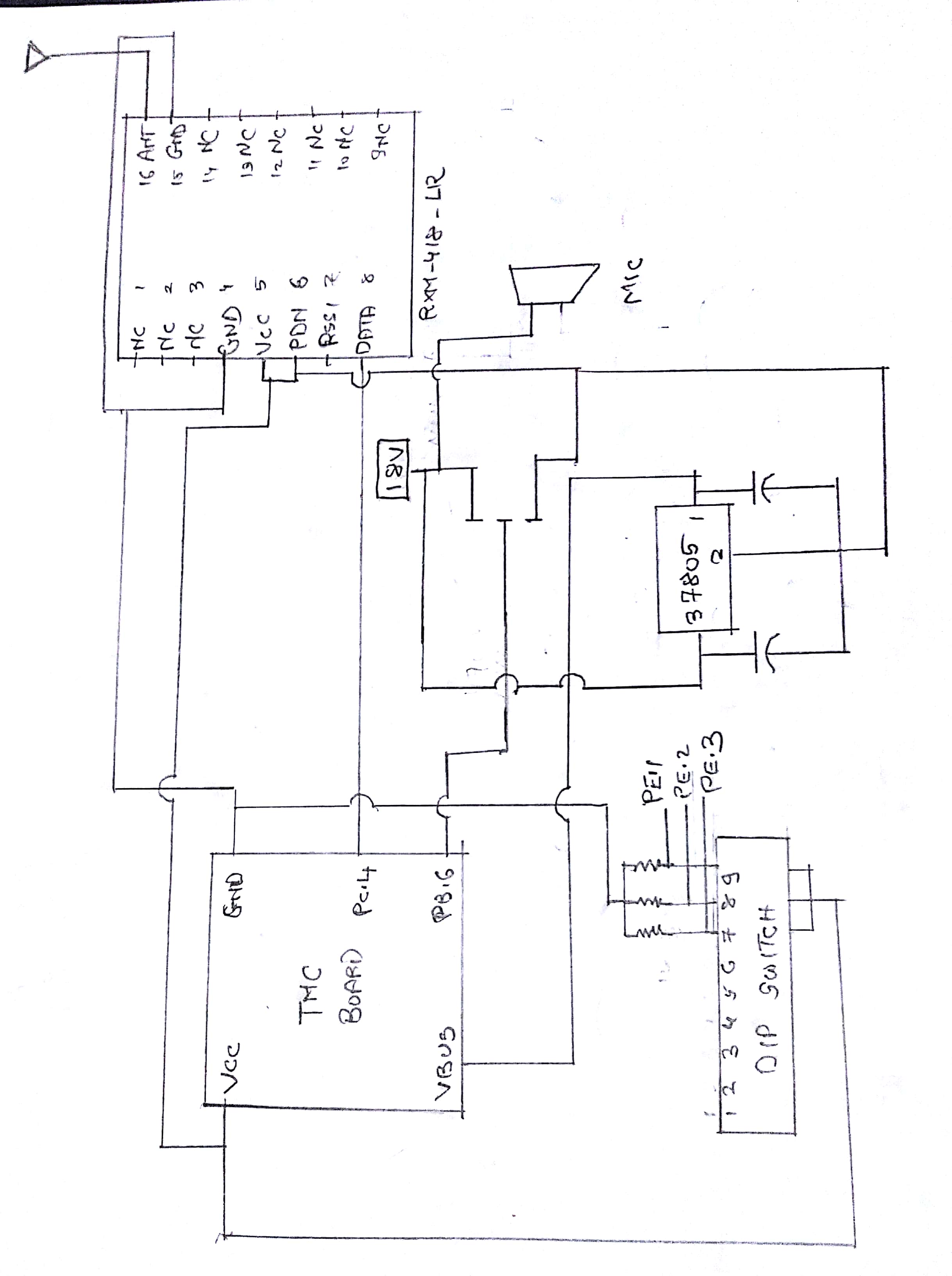


Figure 4: Ultrasonic Transmitter circuit diagram

Project Code-

1. **Reference source code (sync signal and data packets)**

|  |
| --- |
| **#include** <stdint.h>  **#include** <stdbool.h>  **#include** "tm4c123gh6pm.h"  **#include** <strings.h>  **#define** BLUE\_LED (\*((**volatile** uint32\_t \*)(0x42000000 + (0x400253FC-0x40000000)\*32 + 2\*4))) // on-board blue LED  **#define** X1\_COOR 0  **#define** X2\_COOR 496  **#define** X3\_COOR 496  **#define** Y1\_COOR 0  **#define** Y2\_COOR 0  **#define** Y3\_COOR 240  uint32\_t sum;  uint8\_t data[50];  **char** packet[256];  **void** **initHw**()  {  // Configure HW to work with 16 MHz XTAL, PLL enabled, system clock of 40 MHz  SYSCTL\_RCC\_R = SYSCTL\_RCC\_XTAL\_16MHZ | SYSCTL\_RCC\_OSCSRC\_MAIN | SYSCTL\_RCC\_USESYSDIV | (4 << SYSCTL\_RCC\_SYSDIV\_S);  // Note UART on port C must use APB  SYSCTL\_GPIOHBCTL\_R = 0;  // Enable GPIO port A to F peripherals  SYSCTL\_RCGC2\_R = SYSCTL\_RCGC2\_GPIOA | SYSCTL\_RCGC2\_GPIOD | SYSCTL\_RCGC2\_GPIOE | SYSCTL\_RCGC2\_GPIOF | SYSCTL\_RCGC2\_GPIOB | SYSCTL\_RCGC2\_GPIOC;  // Configure port F  GPIO\_PORTF\_DIR\_R = 0x04; // bits 2 is outputs for blue led  GPIO\_PORTF\_DR2R\_R = 0x04; // set drive strength to 2mA (not needed since default configuration -- for clarity)  GPIO\_PORTF\_DEN\_R = 0x04; // enable bit 2  // Configure UART1 pins  SYSCTL\_RCGCUART\_R |= SYSCTL\_RCGCUART\_R1; // turn-on UART1, leave other uarts in same status  GPIO\_PORTC\_DEN\_R |= 0x30; // default, added for clarity  GPIO\_PORTC\_AFSEL\_R |= 0x30; // default, added for clarity  GPIO\_PORTC\_PCTL\_R |= GPIO\_PCTL\_PC5\_U1TX | GPIO\_PCTL\_PC4\_U1RX;  //UART1 CONFIGURATION  UART1\_CTL\_R = 0; // turn-off UART1 to allow safe programming  UART1\_CC\_R = UART\_CC\_CS\_SYSCLK; // use system clock (40 MHz)  UART1\_IBRD\_R = 2083; // r = 40 MHz / (Nx115.2kHz), set floor(r)=21, where N=16 //1200 BUADRATE  UART1\_FBRD\_R = 21; // round(fract(r)\*64)=45  UART1\_LCRH\_R = UART\_LCRH\_WLEN\_8 | UART\_LCRH\_FEN; // configure for 8N1 w/ 16-level FIFO  UART1\_CTL\_R = UART\_CTL\_TXE | UART\_CTL\_RXE | UART\_CTL\_UARTEN; // enable TX, RX, and module  }  **void** **waitMicrosecond**(uint32\_t us)  {  // Approx clocks per us  **\_\_asm**("WMS\_LOOP0: MOV R1, #6"); // 1  **\_\_asm**("WMS\_LOOP1: SUB R1, #1"); // 6  **\_\_asm**(" CBZ R1, WMS\_DONE1"); // 5+1\*3  **\_\_asm**(" NOP"); // 5  **\_\_asm**(" B WMS\_LOOP1"); // 5\*3  **\_\_asm**("WMS\_DONE1: SUB R0, #1"); // 1  **\_\_asm**(" CBZ R0, WMS\_DONE0"); // 1  **\_\_asm**(" B WMS\_LOOP0"); // 1\*3  **\_\_asm**("WMS\_DONE0:"); // ---  // 40 clocks/us + error  }  **void** **putcUart1**(**char** c)  {  **while** (UART1\_FR\_R & UART\_FR\_TXFF);  UART1\_DR\_R = c;  }  // Blocking function that writes a string when the UART buffer is not full  **void** **putsUart1**(**char**\* str)  {  uint8\_t i;  **for** (i = 0; i < **strlen**(str); i++)  putcUart1(str[i]);  }  **void** **putcUart0**(**char** c)  {  **while** (UART0\_FR\_R & UART\_FR\_TXFF);  UART0\_DR\_R = c;  }  // Blocking function that writes a string when the UART buffer is not full  **void** **putsUart0**(**char**\* str)  {  uint8\_t i;  **for** (i = 0; i < **strlen**(str); i++)  putcUart0(str[i]);  }  **void** **SumWords**(**void**\* data, uint16\_t size\_in\_bytes)  {  uint8\_t\* pData = (uint8\_t\*)data;  uint16\_t i;  uint8\_t phase = 1;  uint16\_t data\_temp;  **for** (i = 0; i < size\_in\_bytes; i++)  {  **if** (phase)  {  data\_temp = \*pData;  sum += data\_temp << 8;  }  **else**  sum += \*pData;  phase = 1 - phase;  pData++;  }  }  // Completes 1's compliment addition by folding carries back uint8\_to field  uint16\_t **getChecksum**()  {  uint16\_t result;  // this is based on rfc1071  **while** ((sum >> 16) > 0)  sum = (sum & 0xFFFF) + (sum >> 16);  result = sum & 0xFFFF;  **return** ~result;  }  **void** **initialize\_data**()  {  data[0] = ((X1\_COOR & 0xFF00) >> 8) + 1;  data[1] = (X1\_COOR & 0x00FF) + 1;  data[2] = ((Y1\_COOR & 0xFF00) >> 8) + 1;  data[3] = (Y1\_COOR & 0x00FF) + 1;  data[4] = ((X2\_COOR & 0xFF00) >> 8) + 1;  data[5] = (X2\_COOR & 0x00FF) + 1;  data[6] = ((Y2\_COOR & 0xFF00) >> 8) + 1;  data[7] = (Y2\_COOR & 0x00FF) + 1;  data[8] = ((X3\_COOR & 0xFF00) >> 8) + 1;  data[9] = (X3\_COOR & 0x00FF) + 1;  data[10] = ((Y3\_COOR & 0xFF00) >> 8) + 1;  data[11] = (Y3\_COOR & 0x00FF) + 1;  data[12]='\0';  }  **int** **main**(**void**)  {  uint16\_t chksum;  uint8\_t i = 0;  uint8\_t j = 0;  uint8\_t len;  sum = 0;  initialize\_data();  **char** \*data\_ptr = &data[0];  initHw();  BLUE\_LED = 1;  waitMicrosecond(100000);  BLUE\_LED = 0;  len = 12;  SumWords(data\_ptr,len);  packet[i++] = len+2;  chksum = getChecksum();  **while**(j < len)  {  packet[i] = \*data\_ptr++;  i++;  j++;  }  packet[i++] = ((chksum & 0xFF00) >> 8) ;  packet[i++] = (chksum & 0x00FF);  packet[i] = '\0';  **while**(1)  {  BLUE\_LED ^= 1;  UART1\_DR\_R = 0x41;  UART1\_DR\_R = 0x42;  UART1\_DR\_R = 0x43;  UART1\_DR\_R = 0x44;  UART1\_DR\_R = 0x45;  putsUart1(packet);  putsUart0(packet);  waitMicrosecond(250000);  }  } |

1. **Ultrasonic transmitter code (sync detection, checksum calculation and PWM generation)**

|  |
| --- |
| **#include** <stdint.h>  **#include** <stdbool.h>  **#include** "tm4c123gh6pm.h"  **#include** <strings.h>  **#define** BLUE\_LED (\*((**volatile** uint32\_t \*)(0x42000000 + (0x400253FC-0x40000000)\*32 + 2\*4))) // on-board blue LED  **#define** ORANGE\_LED (\*((**volatile** uint32\_t \*)(0x42000000 + (0x400043FC-0x40000000)\*32 + 2\*4))) // on-board blue LED  //pbs for ids  **#define** PUSHBUTTON1 (\*((**volatile** uint32\_t \*)(0x42000000 + (0x400243FC-0x40000000)\*32 + 1\*4))) //  **#define** PUSHBUTTON2 (\*((**volatile** uint32\_t \*)(0x42000000 + (0x400243FC-0x40000000)\*32 + 2\*4))) //  **#define** PUSHBUTTON3 (\*((**volatile** uint32\_t \*)(0x42000000 + (0x400243FC-0x40000000)\*32 + 3\*4))) //  uint8\_t seq\_count=0;  bool sync\_detected = false;  uint8\_t data\_len;  uint32\_t sum;  **char** buffer[256];  **char** data[256];  uint8\_t id;  **static** uint8\_t k = 0;  uint8\_t data\_pos = 0;  uint32\_t time\_constant = 300000;  uint32\_t delta\_t;  bool data\_r = false;  **char** sync\_seq[6]={0x41,0x42,0x43,0x44,0x45};  uint16\_t x1,y1,x2,y2,x3,y3;  **void** **initHw**()  {  // Configure HW to work with 16 MHz XTAL, PLL enabled, system clock of 40 MHz  SYSCTL\_RCC\_R = SYSCTL\_RCC\_XTAL\_16MHZ | SYSCTL\_RCC\_OSCSRC\_MAIN | SYSCTL\_RCC\_USESYSDIV | (4 << SYSCTL\_RCC\_SYSDIV\_S);  // Note UART on port A must use APB  SYSCTL\_GPIOHBCTL\_R = 0;  // Enable GPIO port A to F peripherals  SYSCTL\_RCGC2\_R = SYSCTL\_RCGC2\_GPIOA | SYSCTL\_RCGC2\_GPIOD | SYSCTL\_RCGC2\_GPIOE | SYSCTL\_RCGC2\_GPIOF | SYSCTL\_RCGC2\_GPIOB | SYSCTL\_RCGC2\_GPIOC;  // Configure port F  GPIO\_PORTF\_DIR\_R = 0x04; // bits 2 is outputs for blue led  GPIO\_PORTF\_DR2R\_R = 0x04; // set drive strength to 2mA (not needed since default configuration -- for clarity)  GPIO\_PORTF\_DEN\_R = 0x04; // enable bit 2  // Configure port F  GPIO\_PORTA\_DIR\_R = 0x04; // bits 2 is output for sync\_detection LED  GPIO\_PORTA\_DR2R\_R = 0x04; // set drive strength to 2mA (not needed since default configuration -- for clarity)  GPIO\_PORTA\_DEN\_R = 0x04; // enable bit 2  //Configure Port E for push buttons  GPIO\_PORTE\_DIR\_R = 0x00;  GPIO\_PORTE\_DEN\_R = 0x0E; // enable pushbuttons  GPIO\_PORTE\_PUR\_R = 0x0E; // enable internal pull-up for push button  // Configure UART1 pins  SYSCTL\_RCGCUART\_R |= SYSCTL\_RCGCUART\_R1; // turn-on UART1, leave other uarts in same status  GPIO\_PORTC\_DEN\_R |= 0x30; // default, added for clarity  GPIO\_PORTC\_AFSEL\_R |= 0x30; // default, added for clarity  GPIO\_PORTC\_PCTL\_R |= GPIO\_PCTL\_PC5\_U1TX | GPIO\_PCTL\_PC4\_U1RX;  //uart1  UART1\_CTL\_R = 0; // turn-off UART0 to allow safe programming  UART1\_CC\_R = UART\_CC\_CS\_SYSCLK; // use system clock (40 MHz)  UART1\_IBRD\_R = 2083; // r = 40 MHz / (Nx115.2kHz), set floor(r)=21, where N=16  UART1\_FBRD\_R = 21; // round(fract(r)\*64)=45  UART1\_LCRH\_R = UART\_LCRH\_WLEN\_8; // configure for 8N1 w/o FIFO  UART1\_CTL\_R = UART\_CTL\_TXE | UART\_CTL\_RXE | UART\_CTL\_UARTEN; // enable TX, RX, and module  UART1\_IM\_R = UART\_IM\_RXIM; // turn-on RX interrupt  NVIC\_EN0\_R |= 1 << (INT\_UART1-16); // turn-on interrupt 21 (UART0)  // PWM Configuration  SYSCTL\_RCGCPWM\_R |= SYSCTL\_RCGCPWM\_R0;  GPIO\_PORTB\_DEN\_R |= 0xC0;  GPIO\_PORTB\_AFSEL\_R |= 0xC0;  GPIO\_PORTB\_PCTL\_R = GPIO\_PCTL\_PB6\_M0PWM0;  PWM0\_0\_CTL\_R = 0;  PWM0\_0\_GENA\_R = PWM\_0\_GENA\_ACTCMPAD\_ZERO|PWM\_0\_GENA\_ACTLOAD\_M;  PWM0\_0\_LOAD\_R = 0x674;  PWM0\_0\_CMPA\_R = 0x33A;  PWM0\_0\_CTL\_R |= 1;  PWM0\_ENABLE\_R |= PWM\_ENABLE\_PWM0EN;  }  **void** **waitMicrosecond**(uint32\_t us)  {  // Approx clocks per us  **\_\_asm**("WMS\_LOOP0: MOV R1, #6"); // 1  **\_\_asm**("WMS\_LOOP1: SUB R1, #1"); // 6  **\_\_asm**(" CBZ R1, WMS\_DONE1"); // 5+1\*3  **\_\_asm**(" NOP"); // 5  **\_\_asm**(" B WMS\_LOOP1"); // 5\*3  **\_\_asm**("WMS\_DONE1: SUB R0, #1"); // 1  **\_\_asm**(" CBZ R0, WMS\_DONE0"); // 1  **\_\_asm**(" B WMS\_LOOP0"); // 1\*3  **\_\_asm**("WMS\_DONE0:"); // ---  // 40 clocks/us + error  }  **void** **sum\_data**(**void**\* data, uint16\_t size\_in\_bytes)  {  uint8\_t\* pData = (uint8\_t\*)data;  uint16\_t i;  uint8\_t phase = 1;  uint16\_t data\_temp;  **for** (i = 0; i < size\_in\_bytes; i++)  {  **if** (phase)  {  data\_temp = \*pData;  sum += data\_temp << 8;  }  **else**  sum += \*pData;  phase = 1 - phase;  pData++;  }  }  // Completes 1's compliment addition by folding carries back uint8\_to field  uint16\_t **getChecksum**()  {  uint16\_t result;  // this is based on rfc1071  **while** ((sum >> 16) > 0)  sum = (sum & 0xFFFF) + (sum >> 16);  result = sum & 0xFFFF;  **return** ~result;  }  **void** **Uart1Isr**()  {  **char** in;  **static** uint8\_t i=0;  in = UART1\_DR\_R;  **if**(sync\_detected==true)  {  buffer[k] = in;  //data\_pos = k;  k++;  **if**(k==buffer[0]+1)  {  data\_r = true;  }  }  **if** (in == sync\_seq[i] && sync\_detected == false )  {  seq\_count++;  i++;  **if**(seq\_count == 5)  {  ORANGE\_LED ^= 1;  sync\_detected = true;  seq\_count=0;  i=0;  k=0;  }  }  **else**  {  seq\_count=0;  i=0;  }  }  // for id  **void** **readPbs**()  {  id = (PUSHBUTTON1<<2)|(PUSHBUTTON2<<1)|(PUSHBUTTON3);  }  **void** **decode\_coordinates**()  {  x1 = ((((data[0] & 0xFF)-1) << 8) + ((data[1] & 0xFF)-1));  y1 = ((((data[2] & 0xFF)-1) << 8) + ((data[3] & 0xFF)-1));  x2 = ((((data[4] & 0xFF)-1) << 8) + ((data[5] & 0xFF)-1));  y2 = ((((data[6] & 0xFF)-1) << 8) + ((data[7] & 0xFF)-1));  x3 = ((((data[8] & 0xFF)-1) << 8) + ((data[9] & 0xFF)-1));  y3 = ((((data[10] & 0xFF)-1) << 8) + ((data[11] & 0xFF)-1));  }  **int** **main**(**void**)  {  initHw();  **static** **int** byte\_count = 0;  BLUE\_LED = 1;  waitMicrosecond(1000000);  BLUE\_LED = 0;  **while**(1)  {  **if**(sync\_detected==true && data\_r == true)  {  data\_len = buffer[0];  **for**(byte\_count=0;byte\_count<data\_len;byte\_count++)  {  data[byte\_count]=buffer[byte\_count+1];  }  **if**(byte\_count == data\_len)  {  data[byte\_count] = '\0';  sum = 0;  sum\_data(data,data\_len);  **if**(getChecksum() == 0)  {  decode\_coordinates();  BLUE\_LED ^= 1;  readPbs();  delta\_t = 100000 + id\*time\_constant;  waitMicrosecond(delta\_t);  PWM0\_ENABLE\_R |= PWM\_ENABLE\_PWM0EN;  waitMicrosecond(1200);  PWM0\_ENABLE\_R &= ~PWM\_ENABLE\_PWM0EN;  }  sync\_detected = false;  k=0;  data\_r = false;  }  }  }  } |

Problems and Trouble Shooting:

ZigBee Modules:

Initially, we worked on ZigBee circuit and were able to transmit and receive the signals. But the problem regarding ZigBee module is at the receiving end. We observed that received signal was not in sync and was giving a delay of 40 microseconds at the receivers.

After discovering the delay, we tried to rectify the problem by switching from Zigbee to Linx-LR TX/RX modules.

RF modules:

After getting RF modules we first checked it for synchronization between the transmitted signal and respective received signal on the beacons. The delay between transmission and reception was approximately 55 microseconds and the delay between adjacent receivers was 1.5 microseconds. The delay between receivers was negligible and can be ignored.

Troubleshooting:

When we start understanding the circuit we came to know that our receiver was receiving garbage values whenever the transmitter wasn't transmitting. And so we implemented squall circuit by which we were able to eliminate the garbage values at the receiver.After that, the receiver was able to receive the correct signal but it was not able to detect between level zero and one as we applied the concept of preamble according to which we were giving a training sequence of 55..... in order to let receiver, decide between zero and one. By this method, we were able to receive correct data.

Contribution of Teammates

1. Vimal K Vankar

* Soldering Transmitter and Receiver RF Circuits.
* Testing transmitter and Receiver Circuits with the team.
* Check of RF modules for correct data reception and transmission- Software.
* Designing and experimenting different software for transmission and reception for RF modules.
* Design of software for ZigBee modules for transmission and reception check.
* Always Active to help team.

1. Luv Sachdeva

* Soldering of RF Modules.
* Configuration of Zigbee Modules.
* De-soldering of Zigbee Modules and RF modules whenever required.
* Setting up beacons and binding them appropriately.
* Debugging of sum calculation and checksum code.
* Debugging of hardware whenever occurred.

1. Anupama Malleshappa

* Worked with the team on Testing the RF Modules.
* Coding of RF Reference sync source with Pooja and Disha.
* Coding of RF message TX and RX code with checksum, sending coordinates and decoding it.
* Debugging when problems occurred at different stage with the team.
* Helped with entire team in final integration.

1. Nihit Bhavsar

* Last minute problem solving
* Coding Support
* Soldering and integration
* Configuring Zigbee modules

1. Dhaval Padia

* Designed FET circuit
* Soldering and integration
* Helped in configuring RF transmitter and receiver modules
* Worked with the team on testing the RF modules
* Dealing with issue of noise reception and testing squelch circuit for that issue
* Pole axis arrangement

1. Disha Shah

* Soldering of RF modules, FET and 7805 on gpb and zigbee modules
* Testing of zigbee module on XCTU as well as with code for data reception
* Testing of delays between RF transmitter and receiver and receiver to receiver
* Checking the range and sync signal on oscilloscope for RF and zigbee modules
* Dealing with issue of noise reception and testing squelch circuit for that issue
* Debugging and solving hardware issues
* Testing rf modules output, range by integrating ultrasonic receiver circuit and debugging rf tx and Rx code for different burst cycles and sync timings
* Helped in software for sending location (x,y,z coordinates) from reference source and decoding it on ultrasonic transmitter side

1. Pooja Katiyar

* Zigbee configuration using XCTU and AT commands
* Checking the delay time between transmission and reception of zigbees
* Interfacing Ultrasonic transmitter with microcontroller using FET
* Configuration of RF modules and testing it for the maximum delay between transmission and reception and for different receivers.
* Dealing with issue of noise reception and testing squelch circuit for that issue
* Coding of RF TX and RX code for sync detection and checksum calculation
* Debugging and troubleshooting hardware and software issues
* Integration of all modules and testing the ultrasonic transmitter for different burst