EE318 Electronic Design Lab Project Report, EE Dept, IIT Bombay, April 2007

TUM-TUM TRACKER

Group No. B09

Sushant Arora (06007008)

M.Siddharth (06007010)

Ankur Sharma (06007023)

Supervisor : V.Rajababu

ABSTRACT:

Tum-tum is the internal transportation (bus) of IITB. The EDL project is to design a system that tracks the current location of the tum-tum and updates a display unit at each stop.We plan to implement this using a distributed RF network. Whenever a tum-tum reaches a stop, a receiver at the stop receives data from the transmitter on the tum-tum and identifies the tum-tum. This along with other relevant data is forwarded to other stops and the whole system updates the current location of the tum-tum. This process of communication between stops is carried out every time the tum-tum reaches a stop.

INTRODUCTION:

IIT Bombay uses an internal bus transport system. A small number of buses ply as part of this system and they do so at varying frequencies during the day. As of now, there is no way to estimate how much time it will take for the next bus to arrive at a particular stop. Given that the distances involved are not very large, often one wants to know if it will take less time to wait for the next bus or to simply walk the distance. Also, if one is in a particular hurry, one would like to know if the bus is about to arrive soon before considering a more expensive alternative like an auto.

Keeping these issues in mind, the need for a system that can predict the arrival of the next appropriate bus is felt. This project is an attempt at doing exactly this. The project uses wireless communication between buses and bus-stops to communicate the movement of buses within the campus.

DESIGN:

The system is based on a Radio Frequency (RF) network. The design is essentially based around the RF circuits used. The system consists of the RF modules, micro-controllers to send and receive data through these modules and an LCD to display the current location of the tum-tum.

Hardware:

We are using Amplitude Shift Keying (ASK) transmitter and receiver at two frequencies (315 MHz and 434 MHz). These modules have a pin-out of Vcc, Gnd, Data and Antenna. Wire antenna is used. The specifications of these modules is as follows,

- Frequency: 315/434 MHz.
- Transmitter: Saw filter based ASK hybrid transmitter
- Transmitter supply voltage: 3~12V.
- Receiver: ASK super heterodyne receiver with PLL synthesizer and crystal oscillator.
- Receiver Supply voltage: 5V
- Receiver IF frequency: 500 KHz
- Receiver Sensitivity: -105dBm
- Output power: 4~16dBm
- Data rate 200bps to 3Kbps depending on the supply

Length of antenna used is nearly 0.25 * wavelength of the module.

For 434 MHz, length = 17.3 cm

For 315 MHz, length = 23.8 cm

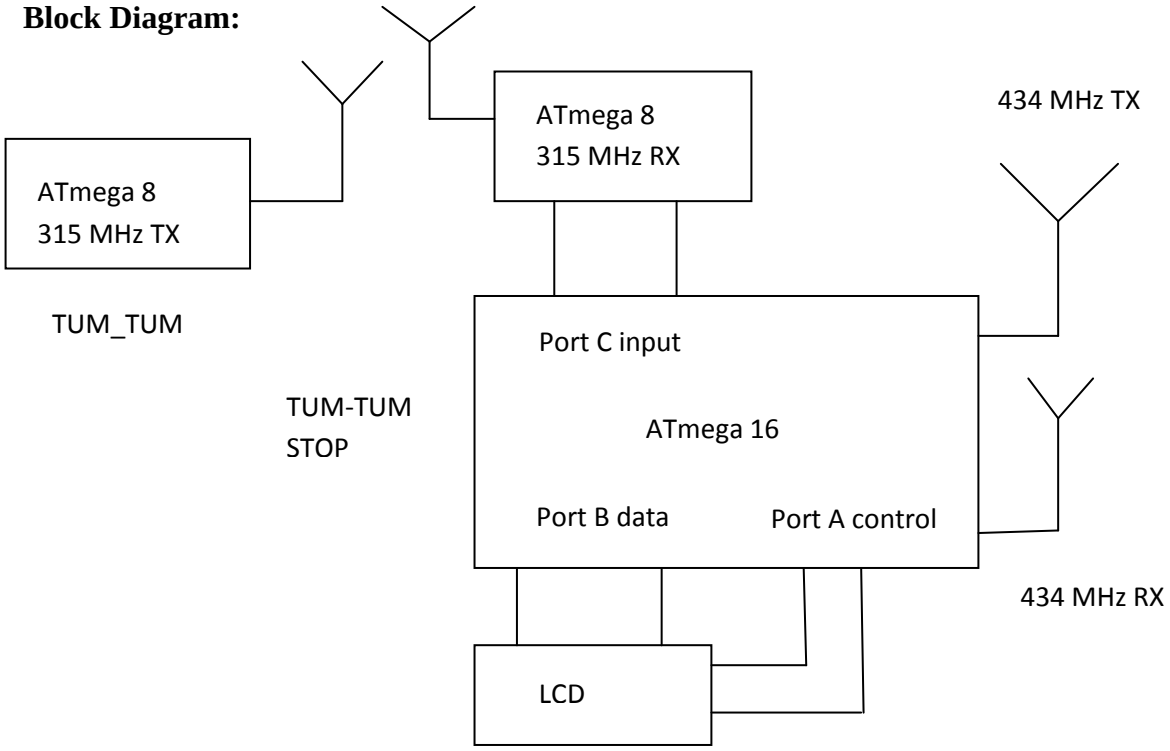
We operated these modules at 5 V. Data was provided using the Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART) facility of the ATmega AVR microcontrollers. Communication between tum-tum and stop is done using 315 MHz TX-RX. Communication between the stops is done using 434 MHz TX-RX. ATmega 8 is used as the

operating uC for the tum-tum transmitter due to its smaller size which results in a compact TX. An ATmega 8 is used at the receiving end (stop) to receive data from the tum-tum. The data received is then sent to the main uC, an ATmega 16.

The ATmega 16 has the following major functions:

1] To receive data from the ATmega 8 which identifies is the tum-tum has reached the stop.
2] To transmit data at 434 MHz using ASK TX whenever tum-tum arrives or it receives data from the previous stop.
3] To receive data from the previous stop using a 434 MHz ASK RX.
4] To display relevant information on a 16X2 LCD display.

Block Diagram:



ALGORITHM:

We designed the algorithm for many stops but implemented it for three stops only. The algorithm makes use of the fact that events occur in a particular order. We assume the sops to be in a straight line. This is not valid in general but is a very good approximation if we are considering only a single route. The tum-tum will ply from end point 1 to end point 2 with the stops appearing in a pre-defined manner. The flow of control is explained below.

For the start stop,

1] Poll for data at 315 MHz
2] If correct data is received, tell other stops that tum-tum is here and update display
3] Poll for data at 434 MHz
4] If correct data is received, check if it was received from the previous stop or it is the same as the data transmitted
5] If the data is not received from the previous stop or is the same, go to step 3
6] If data is received from previous stop, display current location of tum-tum.
7] Check is tum-tum is one stop away from this stop.
8] If it is one stop away go to step 1 else go to step 3

For the middle stops,

1] Poll for data at 434 MHz
2] If correct data received, check if it was received from the previous stop or it is the same as the data transmitted.
3] If the data is not received from the previous stop or is the same, go to step 1
4] If data is received from previous stop, display current location of tum-tum.
5] Transmit this data to other stops at 434 MHz
6] Check is tum-tum is one stop away from this stop.
7] If it is one stop away continue else go to step 1
8] Poll for data at 315 MHz
9] If correct data is received, tell other stops that tum-tum is here and update display

For the end stop,

1] Poll for data at 434 MHz
2] If correct data received, check if it was received from the previous stop or it is the same as the data transmitted.
3] If the data is not received from the previous stop or is the same, go to step 1
4] If data is received from previous stop, display current location of tum-tum.
5] Check is tum-tum is one stop away from this stop.
6] If it is one stop away continue else go to step 1
7] Poll for data at 315 MHz
8] If correct data is received, tell other stops that tum-tum is here and update display

The fact that the stops receive data only from the ones next to them is to prevent any spurious data from being accepted. It also prevents them from receiving data from their own transmitters. We also reject data that is the same as the data being sent. This is to prevent any loops being formed during data transfer. If this condition was not put, then two middle stops will keep transmitting data to each other.

Please note that all this assumes only a single tum-tum and that the stops are reached in a pre-defined manner. If we were to use multiple tum-tums, an interrupt driven USART would have to be used.

Data being sent:

All the data being sent is sent in packet form with start bytes 0x00 and 0x55.

Tum-tum transmitter (315 MHz) sends only its tum-tum id (tid) .tid data is of 8 bits (1 byte)

Example: Let the tum-tum id be 0x03. The the following data packet is transmitted.

0x00 0x55 0x03 0xfd

The stop transmitter (434 MHz) sends 2 bytes of data. It contains information about tum-tum id (tid), direction in which the tum-tum is moving (dir), pole which transmitted the data received (ppid), the current stop of the bus (sid) and the previous stop of the bus (psid).

tid – 4 bits, dir – 1 bit, ppid – 3 bits, sid – 4 bits, psid – 4 bits

It has a sum byte = data1 + data2 and a checksum data1 ^ 0xff. These conditions are tested at the receiver end to verify if the data received is correct or not.

Example: Let the tum-tum be at stop 2 with the previous stop being stop 1. It is moving in direction 0 and we are watching the display at stop 4. The data packet received is from stop 3

tid = 0x03, dir = 0, ppid = 0x03, sid = 0x02, psid = 0x01

data1 = 0x63 data2 = 0x12. Data packet received is

0x00 0x55 0x63 0x12 0x75 0xf9

For proper data transmission via USART, we are using an external crystal of 12 MHz. The baudrate is 9600 bps. We empirically send 100 data packets. Then 100 bytes of data is received at a time. It is tested is some valid data has been received. If yes processing begins, otherwise the data is received again. An assumption that we are able to receive data at least 4 times in this fashion is made i.e the transmission is on for at least 4 loops of 100 bytes of data.

SOFTWARE:

The codes for various tasks have been provided in appendix

TESTING:
We are using ASK transmitter receivers at 315 and 434 MHz frequencies.

Characterization of the transmitter modules

1] Vcc = 5 V Gnd = 0 V Data = 5 V

434 MHz Output = -10 dBm
315 MHz Output = -13 dBm

2] Vcc = 5 V Gnd = 0 V Data = square wave 0.5 V to 4.5 V of frequency 2 kHz

434 MHz Output = -9 dBm peak
315 MHz Output = -11 dBm peak

3] With wire antenna in bread board and data square wave 0.5 V to 4.5 V of frequency 2 kHz

434 MHz Output = -15 dBm peak
315 MHz Output = -18 dBm peak

4] With wire antenna in PCB and data squarewave 0.5 V to 4.5 V of frequency 2 kHz

434 MHz Output = -23 dBm peak
315 MHz Output = -24 dBm peak

From the transmission characteristics as observed in the spectrum analyser, we decided to use the 315 MHz Tx-Rx for tum-tum to stop communication and the 434 MHz Tx-Rx for stop to stop communication.

The USART protocol was tested first with wires and then wirelessly. The data received was not very accurate during wireless transmission. Sometimes it was correct and sometimes not. Also, we had to send packets of data. This was implemented using start bytes and checksum. The packet received was tested and then displayed. Even this was not very accurate. Finally, we decided to send a large number of packets. These would be received in bulk and then processed to find a correct packet. Theoretically, at least one packet would be correct from a large number of packets. This method worked and so we went forward with this for the remaining project.

Interrupt driven USART was tested with the purpose of receiving data from the tum-tum whenever it comes near a stop. This would have enabled us to use multiple tum-tums. The interrupt enabled USART worked well with wires but failed during wireless testing. It received arbitrary data and behaved randomly.

Some important points that we noted through the testing:

1] Range of the transmitter – receiver pair: nearly 70 m

2] If two transmitters are on at the same time, the receiver doesn't receive any correct data

3] If the transmitter at a stop is on, the receiver's sensitivity decreases by a large margin. It reduces to roughly 8 m. We tried putting an enable pin at the Vcc of the transmitter. The transmitter was switched on whenever needed. The receiver was able to receive the data correctly but the transmitter's characteristics dipped very badly.

4] Initially, the plan was to operate the system using batteries. However, theTx and Rx modules drain a lot of the battery power and the voltage drop across them decreases. The modules are very dependent on the voltage applied across them. This resulted in very poor transmission and hence the receivers were unable to obtain any data.

DEMONSTRATION:

We were unable to give any demonstration due to problems with the modules. We now describe what we had prepared for demonstration and was working nicely prior to the demonstration. We had a starting point, stop A. This was polling for the Tx at 315 MHz. When the Tx reaches the stop, it displays that the Tx is there. It also transmits this data to the next stop which displays data accordingly. The next stop sees that the Tx is one stop away and starts polling for the Tx. If we move the Tx closer to the next stop, it displays that the Tx has reached there.

CONCLUSION:

We have simulated a tum-tum tracker in laboratory environment. The demonstration was not successful due to problems with the transmitter-receiver modules. These modules are very unreliable over longer distances. Also, multiple modules should not be used. One way communication works very well but the system is not suited for cascaded applications.

REFERENCES:

ATmega 8,16 , 32 datasheets

NR-RF 401 TX and RX modules data from the net

www.avrfreaks.net

www.google.com for general help during troubleshooting while receiving data packets

Appendix

Code fragments for various tasks

Header files used:

```
# include <avr/io.h>
# include <util/delay.h>
```

Bit/Pin manipulations

```
# define sbi(x,y) x|=_BV(y)
# define cbi(x,y) x&=~_BV(y)
# define tbi(x,y) x^=_BV(y)
```

USART

Constants needed for Tx-Rx

```
# define FOSC 12000000
# define USART_BAUDRATE 4800
# define BAUD_PRESCALE (((FOSC / (USART_BAUDRATE * 16UL))) - 1)
```

Code to initialize USART in uC

```
void USART_Initialize(void)
{
        UCSRB = ((1 << RXEN) | (1 << TXEN));   // Turn on the tx and rx circuitry
        UCSRC = ((1<<URSEL) | (3<<UCSZ0));
        UBRRL = BAUD_PRESCALE;        // Load lower 8-bits of the baud rate value into
                                      the low byte of the UBRR register
        UBRRH = (BAUD_PRESCALE >> 8); // Load upper 8-bits of the baud rate value
                                                into the high byte of the UBRR register
}
```

Code for Transmission via USART

```
void USART_Transmit( unsigned char data )
{
   while ( !( UCSRA & (1<<UDRE)) ) ;
   UDR = data;
}
```

Code for Reception via USART

```
unsigned char USART_Receive( void )
{
   while ( !(UCSRA & (1<<RXC)) ) ;
   return (UDR);
}
```

LCD:

Pins alloted for LCD control signals

```
# define RS PA0
# define RW PA1
# define EN PA2
```

Code to initialize LCD

```
void init_lcd(void)
{
        cbi(PORTA,RW);
        cbi(PORTA,RS);
        PORTB = 0x38;
        sbi(PORTA,EN);
        cbi(PORTA,EN);
        _delay_ms(100);
        cbi(PORTA,RW);
        cbi(PORTA,RS);
        PORTB = 0x0e;
        sbi(PORTA,EN);
        cbi(PORTA,EN);
        _delay_ms(100);
        cbi(PORTA,RW);
        cbi(PORTA,RS);
        PORTB = 0x06;
        sbi(PORTA,EN);
        cbi(PORTA,EN);
        _delay_ms(100);
}
```

Code to clear LCD

```
void clr_lcd(void)
{
        cbi(PORTA,RW);
        cbi(PORTA,RS);
        PORTB = 0x01;
        sbi(PORTA,EN);
        cbi(PORTA,EN);
        _delay_ms(100);
}
```

Code to write data into LCD

```
void write_lcd(unsigned char data)
{
        sbi(PORTA,RS);
        PORTB = data;
        sbi(PORTA,EN);
        cbi(PORTA,EN);
        _delay_ms(100);
}
```

Display function to display current location.

dir = 0 means towards H13
dir = 1 means towards MGT
for our three stops 0-MGT 1-H8 2-H13
// 0x00 = here
// 0x01 = 5 mins
// 0x02 = 10 mins
// 0x03 = 15 mins
// 0xff = Not applicable
// Display function
// t1 is towards H13
// t2 is towards MGT
We have assumed that the distance between stops is uniform i.e 5 mins. Accordingly the code incorporates the time it will take for a tum-tum moving in a particular direction to reach a particular stop. This then calls the display function. In some cases, like at the end stops, it is meaningless for a tum-tum to go towards that end stop. This is represented by NA.

```
void disp(unsigned char t1, unsigned char t2)
{
        clr_lcd();
        if(t1 == 0xff)
        {
                write_lcd('H');
                write_lcd('1');
                write_lcd('3');
                write_lcd(' ');
                write_lcd('N');
                write_lcd('A');
                write_lcd(' ');
        }
        if(t1 == 0x00)
```

```
{
        write_lcd('H');
        write_lcd('1');
        write_lcd('3');
        write_lcd(' ');
        write_lcd('H');
        write_lcd('R');
        write_lcd('E');
        write_lcd(' ');
}
if(t1 == 0x01)
{
        write_lcd('H');
        write_lcd('1');
        write_lcd('3');
        write_lcd(' ');
        write_lcd('5');
        write_lcd('m');
        write_lcd(' ');
}
if(t1 == 0x02)
{
        write_lcd('H');
        write_lcd('1');
        write_lcd('3');
        write_lcd(' ');
        write_lcd('1');
        write_lcd('0');
        write_lcd('m');
        write_lcd(' ');
}
if(t1 == 0x03)
{
        write_lcd('H');
        write_lcd('1');
        write_lcd('3');
        write_lcd(' ');
        write_lcd('1');
        write_lcd('5');
        write_lcd('m');
        write_lcd(' ');
}

if(t2 == 0xff)
```

```
{
        write_lcd('M');
        write_lcd('G');
        write_lcd('T');
        write_lcd(' ');
        write_lcd('N');
        write_lcd('A');
        write_lcd(' ');
}
if(t2 == 0x00)
{
        write_lcd('M');
        write_lcd('G');
        write_lcd('T');
        write_lcd(' ');
        write_lcd('H');
        write_lcd('R');
        write_lcd('E');
        write_lcd(' ');
}
if(t2 == 0x01)
{
        write_lcd('M');
        write_lcd('G');
        write_lcd('T');
        write_lcd(' ');
        write_lcd('5');
        write_lcd('m');
        write_lcd(' ');
}
if(t2 == 0x02)
{
        write_lcd('M');
        write_lcd('G');
        write_lcd('T');
        write_lcd(' ');
        write_lcd('1');
        write_lcd('0');
        write_lcd('m');
        write_lcd(' ');
}
if(t2 == 0x03)
{
        write_lcd('M');
```

```
                write_lcd('G');
                write_lcd('T');
                write_lcd(' ');
                write_lcd('1');
                write_lcd('5');
                write_lcd('m');
                write_lcd(' ');
        }
        _delay_ms(5000);
}
```

Code used for 315 MHz transmitter

```
# define tid 0x01                   // Tum-Tum ID

        data = tid;
        chk = data      ^       0xff;
        for(;;)                                         // Loop 1 to simulate tum-tum 1
        {
                USART_Transmit(0x00);
                USART_Transmit(0x55);
                USART_Transmit(data);
                USART_Transmit(chk);
        }
```

It continuously transmits the data.

Code for receiver in ATmega 8 that receives from the 315 MHz transmitter

Port Definitions

```
        DDRC = 0xff;
        PORTC = 0x00;
```

Initialization

```
        USART_Initialize();
        unsigned int i,j,ctr,c;
        unsigned char a[105];

        re:
        c = 1;
        i = 0;
        j = 0;
```

Code to receive data packets using UsART

```
        while(c)
        {
```

```
        PORTC = 0x00;

        for(i=0;i<100;i++)
                a[i] = 0x00;

        for(i=0;i<100;i++)
                a[i] = USART_Receive();

        for(i=0;i<100;i++)
        {
                if(a[i] == 0x55)
                {
                        j = i;
                        for(i=j+1;a[i]!=0x00;i++);
                        ctr = i;
                        if((ctr-j) == 3)
                        {
                                if((a[j+1]+a[j+2]) ==0xff)
                                {
                                        c = 0;
                                        break;
                                }
                        }
                }
        }
}
```

To transfer data to output port

```
        PORTC = a[j+1];
        _delay_ms(500);

        goto re;
```

Code for 434 MHz transmitter

```
        for(i=0;i<200;i++) // Loop
        {
                USART_Transmit(0x00);
                USART_Transmit(0x55);
                USART_Transmit(data1);
                USART_Transmit(data2);
                USART_Transmit(sum);
                USART_Transmit(chk);
        }
```

Code for 315 MHz receiver in ATmega16

```
for(;;)
{
        tid = (PINC & 0x03);
        if(tid)
                break;
}
```

Code for 434 MHz receiver in ATmega 16

```
while(c)
{
        for(i=0;i<100;i++)
                a[i] = 0x00;

        for(i=0;i<100;i++)
                a[i] = USART_Receive();

        for(i=0;i<100;i++)
        {
                if(a[i] == 0x55)
                {
                        j = i;
                        for(i=j+1;a[i]!=0x00;i++);
                        ctr = i;
                        if((ctr-j) == 3)
                        {
                        if(((a[j+1] + a[j+2]) == a[j+3]) && ((a[j+1] + a[j+4]) == 0xff))
                                {
                                        c = 0;
                                        break;
                                }
                        }
                }
        }
}
```

Code to process data, to test whether it is correct or not, to modify for transmission

Decode the data received

```
tid = temp1 & 0x0f;
dir = (temp2>>4) & 0x01;
ppid = (temp3>>5) & 0x07;
psid = temp4 & 0x0f;
sid = (temp5>>4) & 0x0f;
```

Check is the data is received from the nearest stops or not

```
if((ppid != 0x01) && (ppid != 0x03))
    goto re1;
```

Check is data received is same as data sent or not

```
if(temp4 == data2)
    goto re1;
```

Modify received data to trasnmit it to other poles

```
data1 = tid|(dir<<4)|(pid<<5);
data2 = psid|(sid<<4);
sum = data1 + data2;
chk = data1 ^ 0xff;
```

Completed code for first stop.

```
// Header that includes all the functions
# include "RF.h"

// Stop ID unique for each pole
# define pid 0x02
// Middle pole

int main(void)
{

    // PORT definitions
    DDRB = 0xff;
    PORTB = 0x00;
    DDRA = 0xff;
    PORTA = 0x00;
    DDRC = 0x00;
    PORTC = 0xff;

    // Initialization of USART and LCD
    init_lcd();
    clr_lcd();
    USART_Initialize();

    // Initialize the variables
    unsigned int i,j,ctr,c;
    unsigned char a[105];
    unsigned char sid, dir, tid, psid, ppid;
    unsigned char temp1,temp2,temp3,temp4,temp5;
```

```
unsigned char data1, data2,sum,chk;

i = 0;
j = 0;
ctr = 1;

re1:
c = 1;

// Code to receive data packets using UsART for 434 MHz
while(c)
{
	for(i=0;i<100;i++)
		a[i] = 0x00;

	for(i=0;i<100;i++)
		a[i] = USART_Receive();

	for(i=0;i<100;i++)
	{
		if(a[i] == 0x55)
		{
			j = i;
			for(i=j+1;a[i]!=0x00;i++);
			ctr = i;
			if((ctr-j) == 5)
			{
			if(((a[j+1] + a[j+2]) == a[j+3]) && ((a[j+1] + a[j+4]) == 0xff))
				{
					c = 0;
					break;
				}
			}
		}
	}
}

temp1 = a[j+1];
temp2 = a[j+1];
temp3 = a[j+1];
temp4 = a[j+2];
temp5 = a[j+2];

// Decode the data received
```

```
tid = temp1 & 0x0f;
dir = (temp2>>4) & 0x01;
ppid = (temp3>>5) & 0x07;
psid = temp4 & 0x0f;
sid = (temp5>>4) & 0x0f;

if((ppid != 0x01) && (ppid != 0x03))
        goto re1;

// Display
if(!dir)
        disp(0x01,0x03);
else
        disp(0x03,0x01);

// Modify received data to trasnmit it to other poles
data1 = tid|(dir<<4)|(pid<<5);
data2 = psid|(sid<<4);
sum = data1 + data2;
chk = data1 ^ 0xff;

// Transmit at 434 MHz
for(i=0;i<200;i++) // Loop
{
        USART_Transmit(0x00);
        USART_Transmit(0x55);
        USART_Transmit(data1);
        USART_Transmit(data2);
        USART_Transmit(sum);
        USART_Transmit(chk);
}

if((sid != 0x01) && (sid != 0x03))
        goto re1;

// Polling for 315 MHz
for(;;)
{
        tid = (PINC & 0x03);
        if(tid)
                break;
}

psid = sid;
```

```
	sid = pid;

	// Display
	if(!dir)
		disp(0x00,0x02);
	else
		disp(0x02,0x00);
	write_lcd(0x30+tid);

	// Modify received data to trasnmit it to other poles
	data1 = tid|(dir<<4)|(pid<<5);
	data2 = psid|(sid<<4);
	sum = data1 + data2;
	chk = data1 ^ 0xff;

	// Transmit at 434 MHz
	for(i=0;i<100;i++) // Loop
	{
		USART_Transmit(0x00);
		USART_Transmit(0x55);
		USART_Transmit(data1);
		USART_Transmit(data2);
		USART_Transmit(sum);
		USART_Transmit(chk);
	}

	goto re1;
}
```

Completed code for second stop

```
// Header that includes all the functions
# include "RF.h"

// Stop ID unique for each pole
# define pid 0x02
// Middle pole

int main(void)
{

	// PORT definitions
	DDRB = 0xff;
	PORTB = 0x00;
	DDRA = 0xff;
```

```
PORTA = 0x00;
DDRC = 0x00;
PORTC = 0xff;

// Initialization of USART and LCD
init_lcd();
clr_lcd();
USART_Initialize();

// Initialize the variables
unsigned int i,j,ctr,c;
unsigned char a[105];
unsigned char sid, dir, tid, psid, ppid;
unsigned char temp1,temp2,temp3,temp4,temp5;
unsigned char data1, data2,sum,chk;

i = 0;
j = 0;
ctr = 1;

re1:
c = 1;

// Code to receive data packets using UsART for 434 MHz
while(c)
{
        for(i=0;i<100;i++)
                a[i] = 0x00;

        for(i=0;i<100;i++)
                a[i] = USART_Receive();

        for(i=0;i<100;i++)
        {
                if(a[i] == 0x55)
                {
                        j = i;
                        for(i=j+1;a[i]!=0x00;i++);
                        ctr = i;
                        if((ctr-j) == 5)
                        {
                        if(((a[j+1] + a[j+2]) == a[j+3]) && ((a[j+1] + a[j+4]) == 0xff))
                                {
                                        c = 0;
```

```
                                        break;
                                    }
                                }
                            }
                        }
}

temp1 = a[j+1];
temp2 = a[j+1];
temp3 = a[j+1];
temp4 = a[j+2];
temp5 = a[j+2];

// Decode the data received
tid = temp1 & 0x0f;
dir = (temp2>>4) & 0x01;
ppid = (temp3>>5) & 0x07;
psid = temp4 & 0x0f;
sid = (temp5>>4) & 0x0f;

if((ppid != 0x01) && (ppid != 0x03))
        goto re1;
if(temp4 == data2)
        goto re1;
// Display
if(!dir)
        disp(0x01,0x03);
else
        disp(0x03,0x01);

// Modify received data to transmit it to other poles
data1 = tid|(dir<<4)|(pid<<5);
data2 = psid|(sid<<4);
sum = data1 + data2;
chk = data1 ^ 0xff;

// Transmit at 434 MHz
for(i=0;i<200;i++) // Loop
{
        USART_Transmit(0x00);
        USART_Transmit(0x55);
        USART_Transmit(data1);
        USART_Transmit(data2);
        USART_Transmit(sum);
```

```
		USART_Transmit(chk);
	}

	if((sid != 0x01) && (sid != 0x03))
		goto re1;

	// Polling for 315 MHz
	for(;;)
	{
		tid = (PINC & 0x03);
		if(tid)
			break;
	}

	psid = sid;
	sid = pid;

	// Display
	if(!dir)
		disp(0x00,0x02);
	else
		disp(0x02,0x00);
	write_lcd(0x30+tid);

	// Modify received data to trasnmit it to other poles
	data1 = tid|(dir<<4)|(pid<<5);
	data2 = psid|(sid<<4);
	sum = data1 + data2;
	chk = data1 ^ 0xff;

	// Transmit at 434 MHz
	for(i=0;i<100;i++) // Loop
	{
		USART_Transmit(0x00);
		USART_Transmit(0x55);
		USART_Transmit(data1);
		USART_Transmit(data2);
		USART_Transmit(sum);
		USART_Transmit(chk);
	}

	goto re1;
}
```