Reliable Wireless Communication Based on LISA

Abhishek Singh
Computer Engineering Department, College of Engineering
San Jose State University, San Jose, CA 95192
E-mail: singh.abhishek21@gmail.com

Abstract

This paper is about creating reliable wireless communication based on Linear Invariant Synchronisation Algorithm (LISA), data scrambling and LBC. LISA is used to extract sync field which helps in extracting start bit for payload. Scrambling is used to improve clock recovery and data reception capabilities by removing long sequences of zeros and ones. Error detection is done by LBC block. Two nodes which are based on LPC1769 are able to successfully communicate on noisy channel based on LISA, data scrambling and LBC block.

1. Introduction

This project mains at establishing reliable wireless communication which should work even for slightly noisy wireless channel. Implementation is based on LPC1769 which is an ARM Cortex-M3 based microcontroller. LPC1769 operates at 120Mhz which gives sufficient speed for data processing.

Heart of reliable communication is based on LISA. It helps in extracting payload depending upon timing information retrieved from sync field. For maintaining signal integrity use scrambling/descrambling. For error detection and correction, LBC is used.

2. Methodology

Synchronisation and scrambling algorithm is programmed in C Language and runs on LPC1769 controller based development board. This section gives detailed hardware and software design of development board to achieve reliable wireless communication.

2.1. Objectives and Technical Challenges

LISA implementation incorporates detecting alternating pattern of ones and zeros at higher nibble and incrementing number at lower nibble, and jump to start bit of payload based on this information. This should also support customizing confidence level by giving variable level of sync pattern match.

Scrambling/Descrambling algorithm improve signal integrity by reducing long sequences of ones and zeros. This is applied only on payload keeping sync field unchanged. Supports flexible order of scrambling and descrambling.

Error at receiver is detected by doing Linear Block Coding at transmitter side and finding syndrome matrix at receiver side.

2.2. Problem Formulation and Design

System consist of console, power supply, embedded system, RF transmitter and receiver. Console provide interface for user to program and debug LISA algorithm on embedded board. This setup supports both wired and wireless mode of data transfer between Ni and Nj. Wired mode is only used for debugging purpose and recommended way for this project is wireless communication.

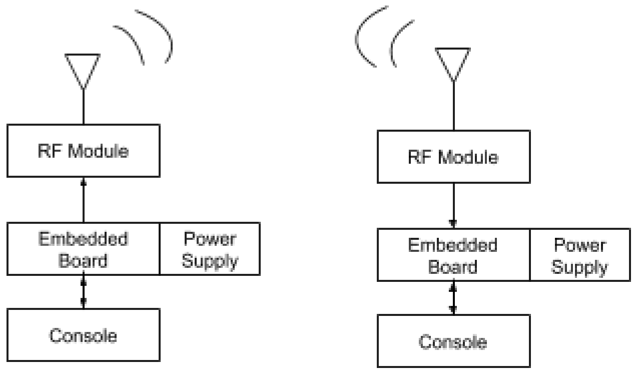


Figure 1. System Block Diagram

Transmitter part consist of source data, scrambling block, LISA block and RF module for transmission of data. Refer below figure for blocks involved in transmission of data.

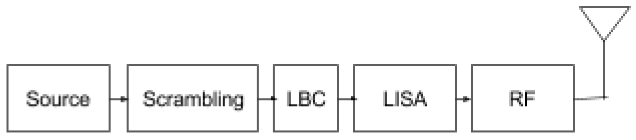


Figure 2. Transmitter Block Diagram

Receiver part consist of RF module for receiving scrambled data with sync field, which goes through descrambling and LISA block to retrieve original data. Refer below figure for blocks involved in reception of data and extraction of original data.

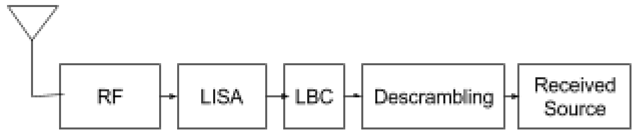


Figure 3. Receiver Block Diagram

3. Implementation

This section gives detailed design of hardware and design including hardware pinouts and software algorithm. Refer below diagram for flow control.

3.1. Hardware Design

3.1.1 Power Supply

Power supply module provide 5V regulated power supply based on LM7805. Below diagram shows circuit diagram for it.

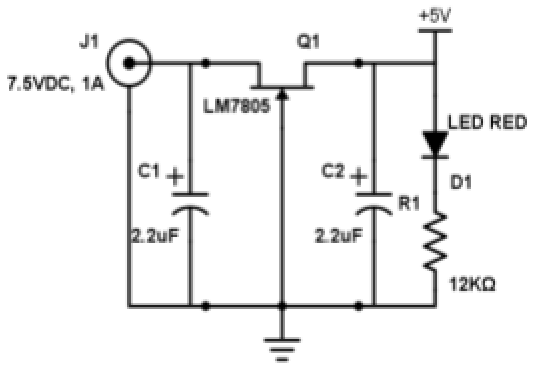


Figure 4. Power Supply Design

LM7805 is used to generate 5V regulated power supply for board. It has 3 pins. First pin is input which accept 9V DC power supply. Second pin is ground. Third pin is output, which gives 5V power supply. Red LED (L1) is connected at output of 7805 to indicate board is powered up. 5V and ground pin is taken out for testing purpose.

3.1.2 Embedded System

Embedded system is based on LPC1769 microcontroller. It receives 5V input from power supply module. Console module programs LPC1769 via USB mini-B cable.

J6-2 is power supply and J6-1 is ground. GPIO pins are used of data transmission. J6-49 is configured as transmitter and J6-50 is configured as receiver. Transmitter and receiver pin is connected to RJ45 connector for wired communication or wireless communication (through RF module). Below is circuit diagram for embedded system.

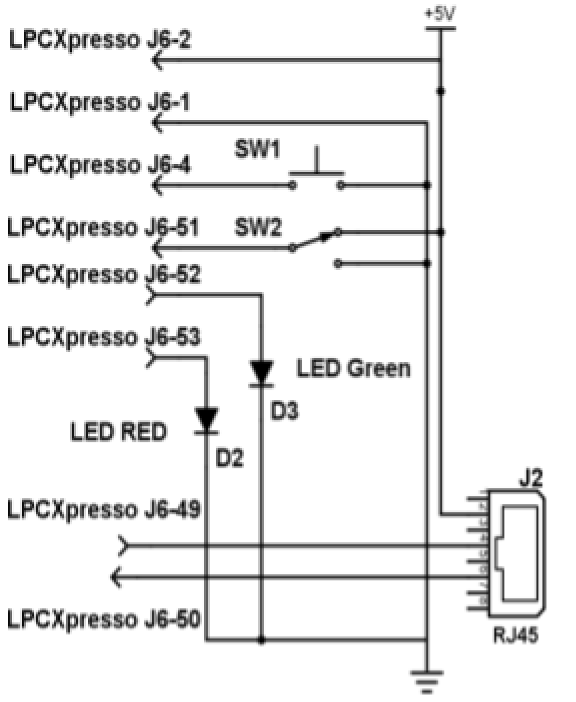


Figure 5. LPCXpresso connection

For debugging purpose two LED are provided. Green and red LED is part of output debugging circuit. Green LED is connected to J6-52 which refers to transmitter data and red LED is connected to J6-53 which refer to receiver data. Switch is part of input debugging circuit which is connected to J6-51. Along with this a reset switch connected at J6-4 is also provided which is used to reset LPC1769 controller. Refer table for LPCXpresso GPIO and RJ45 connection details.

Item No	Description	LPC pin	Note
1	Power Supply	J6-2	Power supply to LPCXpresso
2	Ground	J6-1	Ground for LPCXpresso
3	Transmitter	J6-49	LPC1769 P[2].7
4	Receiver	J6-50	LPC1769 P[2].8
5	Digital	J6-53	LPC1769 P[2].11

	Output - Red LED		
6	Digital Input - Green LED	J6-52	LPC1769 P[2].12
7	Toggle Switch	J6-4	LPC1769 reset
8	Sliding switch	J6-51	LPC1769 P[2].10

Table 1. LPCXpresso GPIO connectivity design

Pin No.	Description	Note
1	NC	Not used
2	VCC	5V Power
3	NC	Not used
4	Transmit	Transmit data
5	NC	Not used
6	Receiver	Receive data
7	NC	Not used
8	GND	Ground

Table 2. Pin Assignment of RJ45(J2)

3.1.3 RF Module

RF module uses MX-05V/XD-RF-5V transmitter and receiver pair. These modules operates at 33MHz frequency. Below is circuit design for RF board.

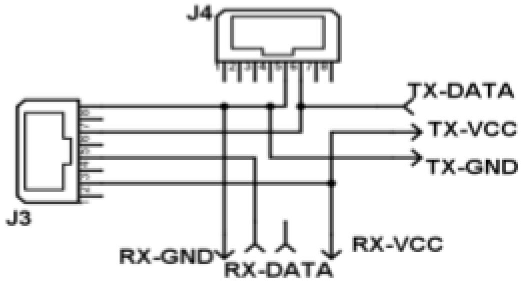


Figure 6. RF Board Circuit Design

Antenna is used at transmitter and receiver end to facilitate data reception in noisy wireless channel. Refer table for connectivity detail for RF45 connectors (J3, J4).

Pin No.	Description	Note
1	NC	Not used
2	VCC	5V Power
3	NC	Not used
4	Receiver	Receive data
5	NC	Not used
6	Transmit	Transmit data
7	NC	Not used
8	GND	Ground

Table 3. Pin Assignment of RJ45(J3)

Pin No.	Description	Note
1	NC	Not used
2	NC	Not used
3	NC	Not used
4	NC	Not used
5	GND	Ground
6	Transmit	Receive data
7	NC	Not used
8	NC	Not used

Table 4. Pin Assignment of RJ45(J4)

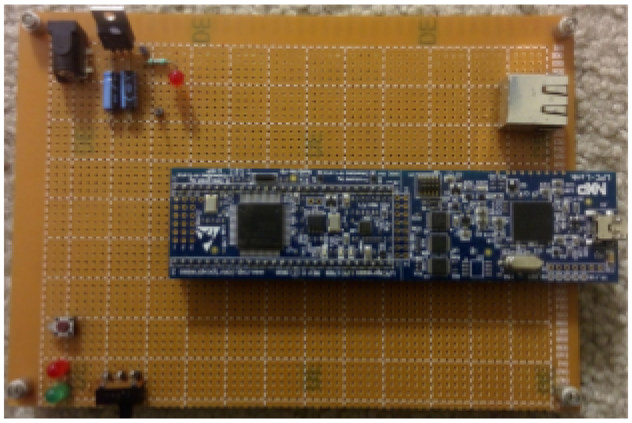


Figure 7. Power Supply and Embedded System

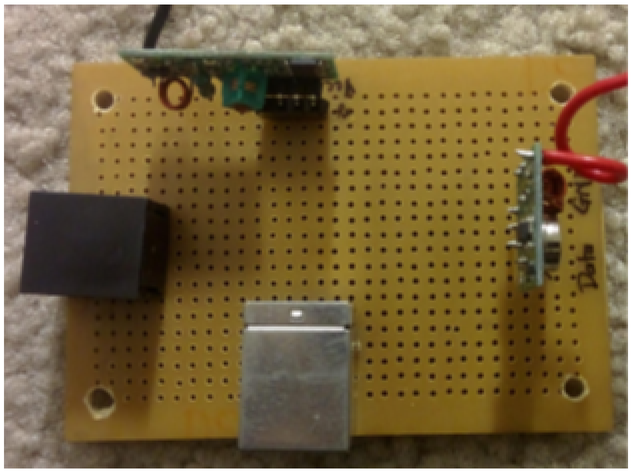


Figure 8. RF Board

3.2. Software Design

3.2.1 Algorithm

3.2.1.1 Master Mode Algorithm

1. Initise GPIO pins of transmitter and output debugging circuit.
2. Scramble user data.
3. Store sync pattern (0x50-0x5F and 0xA0-0xAF) in binary format in a queue.
4. Append scrambled user data in binary format to the queue.
5. Transmit all data from the queue.
6. Initial receiver GPIO pins.
7. Receive data from GPIO pin and store in queue.
8. Process data for sync pattern.
9. Apply descrambling and print retrieved original data if sync pattern found. Otherwise go to step 7.

3.2.1.2 Slave Mode Algorithm

1. Initise GPIO pins of receiver and output debugging circuit.
2. Receive data from GPIO pin and store it in a queue.
3. Process data for sync field.
4. Apply descrambling and print retrieved original data(acknowledgment from Nj) if sync pattern found. Otherwise go to step 2.
5. Initial transmitter GPIO pins.
6. Flush queue.
7. Store sync pattern (0x50-0x5F and 0xA0-0xAF) in binary format in a queue
8. Scramble acknowledgement data.
9. Append scrambled data in binary format to the queue.
10. Transmit all data from the queue.

3.2.1.3 Data Extraction Based on LISA Algorithm

1. Check if queue is not empty. If not empty go to step 2, otherwise loopback.
2. Retrieve 8 elements from the queue and store it in a integer. 1 byte represents 1 bit of binary data received.
3. Compare it with sync field. If match found, find the index of match and calculate offset of payload. If not found shift by 1 byte and repeat from 1 steps.
4. Jump to start bit of payload and print data.

3.2.1.4 Scrambling Algorithm

1. Initial condition for each delay element is zero.
2. Extract ((order/2) +1) delay element(x1) and order delay element(x2) from scrambled data.
3. XOR input (S) with x1 and x2.

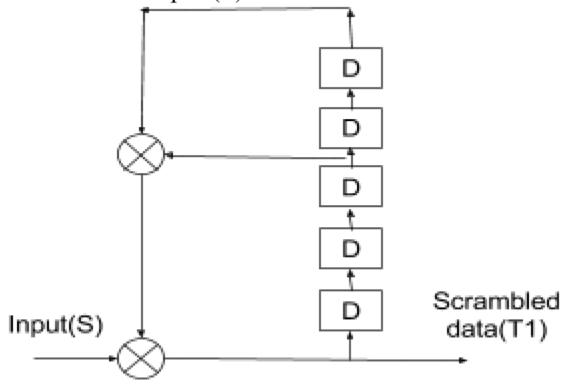


Figure 9. Scrambler Block Diagram

3.2.1.5 Descrambling Algorithm

1. Initial condition for each delay element is zero.
2. Extract ((order/2) +1) delay element(x1) and order delay element(x2) from scrambled data(T2).
3. XOR scrambled data (T2) with x1 and x2.

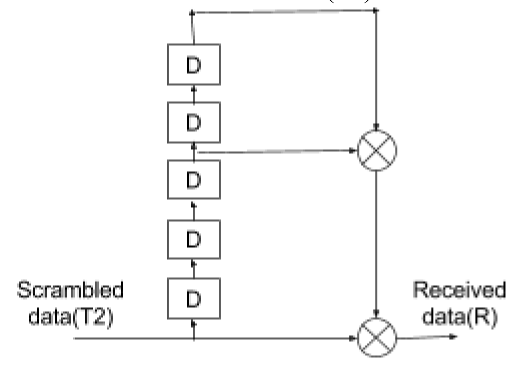


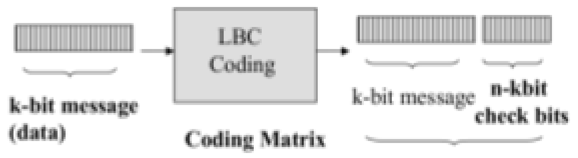
Figure 10. Scrambler Block Diagram

3.2.1.6 Linear Block Coding Algorithm

1. Prepare a generator matrix by below equation.

$$G_{kxn} = [I_{kxk} \vdots P_{kx(n-k)}]$$

2. Incoming data is multiplied by generated matrix to give coded words.



3.2.1.7 LBC Error Detection Algorithm

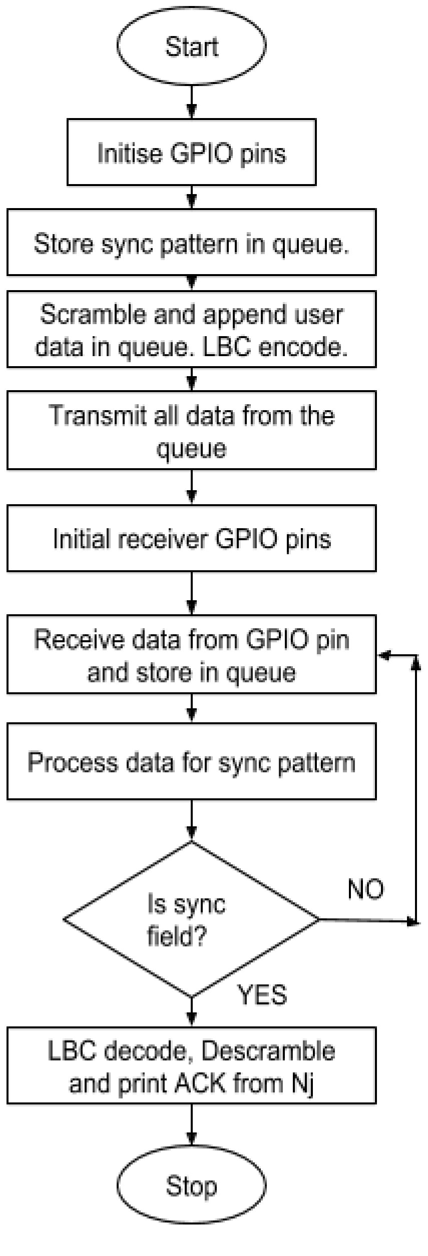
1. Prepare parity check matrix by using following mathematical expression.

$$H^{T}_{n,n-k} = \begin{pmatrix} P_{k,n-k} \\ \hdashline I_{n-k,n-k} \end{pmatrix}_{n,n-k}$$

2. Calculate syndrome matrix by multiplying parity check matrix with received data.
3. Error is detected if matrix check matrix is non zero.
4. If data is error free, extract actual data, else send NACK.

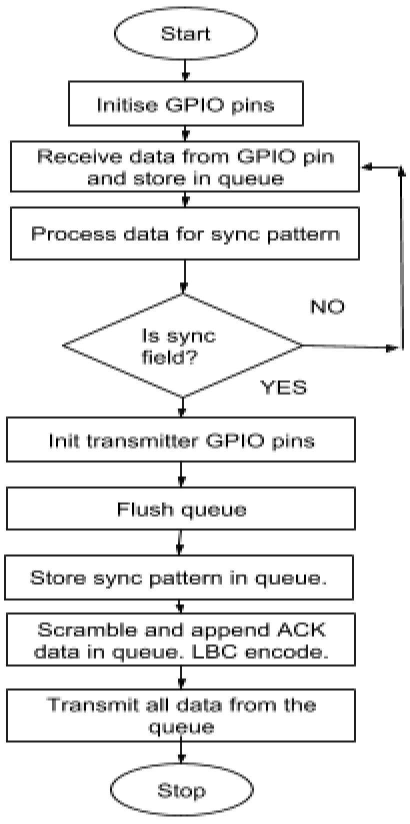
3.2.2 Flow Chart

3.2.2.1 Master Mode


Start
Initise GPIO pins
Store sync pattern in queue.
Scramble and append user data in queue. LBC encode.
Transmit all data from the queue
Initial receiver GPIO pins
Receive data from GPIO pin and store in queue
Process data for sync pattern
Is sync field?
NO
YES
LBC decode, Descramble and print ACK from Nj
Stop

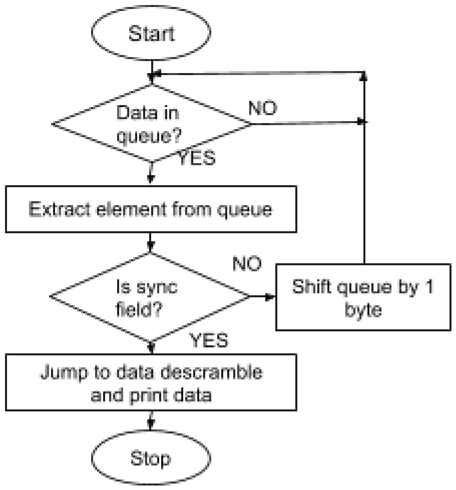
Master Mode

3.2.2.2 Slave Mode


Start
Initise GPIO pins
Receive data from GPIO pin and store in queue
Process data for sync pattern
Is sync field?
NO
YES
Init transmitter GPIO pins
Flush queue
Store sync pattern in queue.
Scramble and append ACK data in queue. LBC encode.
Transmit all data from the queue
Stop

Slave Mode

3.2.2.3 Data Extraction Based on LISA Algorithm



Data Extraction

3.2.3 Pseudo Code

3.2.3.1 Finding Sync Pattern

```
    while(isQueueFull()){
        unsigned int i = 0;
        unsigned int d = 0;
        if(flag1 == 0)
        {
            for(i = 0; i < 8; i++)
            {
                prev[i] = dequeue();
                d += pow(2, i)*prev[i];
            }
            flag1 = 1;
        }
        else
        {
            for(i = 0; i < 7; i++)
            {
                prev[i] = prev[i+1];
                d += pow(2, i)*prev[i];
            }
            prev[i] = dequeue();
            d += pow(2, i)*prev[i];
        }
        if(matchStartPattern((char)d)){
            if(flag == 0){
                match = matchedIndex;
                flag = 1;
            }
            flag1 = 0;
            count++;
        }else{
            match = -1;
            flag = 0;
            count = 0;
            flag1 = 1;
            lookupPosition = -1;
        }

        if(flag == 1 && (count == matchLevel)){
            printf("Pattern found at index
%d\n", match);
            offset = calculateOffset(d);
            printf("Offset = %d\n", offset);
            matchedIndex = 0;
            lookupPosition = -1;
            found = 1;
            break;
        }
    }
```

3.2.3.2 Finding Payload Offset

```
    unsigned int lowerNibble = 0;
```

```
   unsigned int upperNibble = 0;
   unsigned int offset = 0;

   lowerNibble = (data & 0x0F);
   upperNibble = (((data>>4) & 0x0F));
   if(upperNibble == 5){
       offset += 16;
   }

   offset += (15 - lowerNibble);
   return (offset*8);
```

3.2.3.3 Finding Payload Offset

```
 unsigned int i = 0;
 while((i < START_BYTES)  && (lookupPosition ==
-1)){
       if(data == lookup[i]){
           matchedIndex = i;
           lookupPosition = i;
           return 1;
       }
       i++;
   }
   if(lookupPosition != -1)
   {
       lookupPosition++;
       if(data == lookup[lookupPosition]){
           matchedIndex = i;
           return 1;
       }
   }
```

3.2.3.4 Display Data

```
if(found)
{
       for(ptr  =  data;  i  <  (DATA_BYTES*8);
i+=8)
       {
              printf("%c",
binaryToString(ptr));
              ptr += 8;
       }
       printf("\n");
}
```

3.2.3.5 Scrambling Data

```
void scramble_data(int order, char *s_data, int
data_size)
{
  int i = 0;
  char x1, x2;
  for(i = 0; i < data_size; i++)
  {
       x1 = 0;
       x2 = 0;
        if(i >= ((order/2) + 1))
        {
               x1  =  s_data[i  -  ((order/2)  +
1)];
        }
        if(i >= order)
        {
              x2 = s_data[i - order];
        }
        s_data[i] = (queue[(32*8) + i]^x1^ x2);
  }
}
```

3.2.3.6 Descrambling Data

```
void descramble_data(int order, char *des_data,
char *data, int data_size)
{
  int i = 0;
  char x1, x2;
  for(i = 0; i < data_size; i++)
  {
       x1 = 0;
       x2 = 0;
        if(i >= ((order/2) + 1))
        {
                x1 = data[i - ((order/2) + 1)];
        }
        if(i >= order)
        {
                x2 = data[i - order];
        }
        des_data[i] = (data[i]^x1^x2) ;
  }
}
```

3.2.3.7 LBC Encode

```
void lbc_encode(char *dest, int data_size, char
*src)
{
   char G[8][12] = {G0, G1, G2, G3, G4, G5, G6,
G7};
   int k;
   int z = 0;
   for(k = 0; k < (data_size); k+=8)
   {
       int i, j;
       for(i = 0; i < 12; i++)
       {
           for(j = 0; j < 8; j++)
           {
                dest[z + i] += (G[j][i]*src[k +
j]);
           }
           dest[z + i] = (dest[z + i]%2);
       }
       z += 12;
   }
}
```

3.2.3.8 Check Error

```
bool check_error(int data_size, char *temp)
{
   char T[12][4] = {T0, T1, T2, T3, T4, T5, T6,
T7, T8, T9, T10, T11};
   char R[1024] = {0};
   int k;
   int z = 0;
   for(k = 0; k < (data_size); k+=12)
   {
       int i, j;
       for(i = 0; i < 4; i++)
       {
           for(j = 0; j < 12; j++)
           {
                 R[z + i] += (T[j][i]*temp[k +
j]);
           }
           R[z + i] = (R[z + i]%2);
           if(R[z + i] == 1)
               return false;
       }
       z += 4;
   }
   return true;
}
```

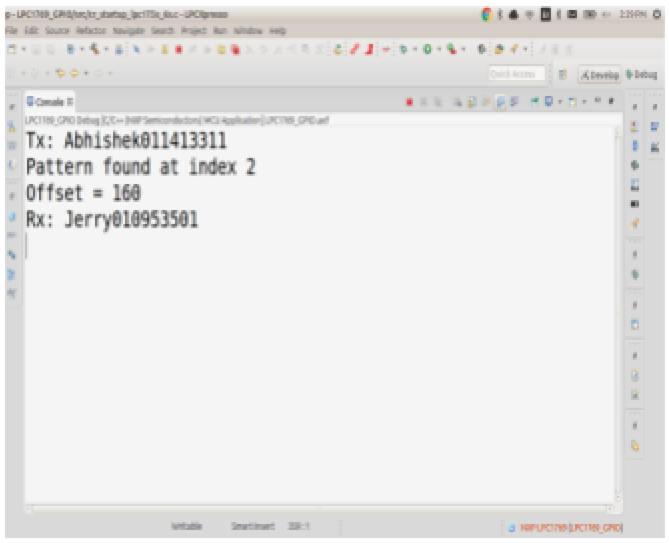
3.2.3.9 LBC Decode

```
void lbc_decode(char *dest, int data_size, char
*src)
{
   int i = 0;int k = 0;
   for(i = 0; i < (data_size); i += 12)
   {
       int j = 0;
       for(j = 0; j < 8; j++)
       {
           dest[j + k] = src[i + j];
       }
       k +=8;
   }
}
```

4. Testing and Verification

1. Hardware circuit was tested using multimeter for checking connectivity between different modules.

2. Once proper connection was established, GPIO port was tested using continuous stream of ones and zeros and displaying it on LED's.
3. Whole setup was tested on wired communication, to check performance of data transfer between Ni and Nj in noiseless channel.
4. After this, setup was tested for wireless communication with very less data rate, aiming at very less noise interface.
5. Optime data rate was achieved based on trial and error method in wireless communication.
6. Scrambling and descrambling was tested for 5, 7 and 9th order.
7. LBC encoding and decoding working properly on 8 bit message(k=8) and 12 bit coded word length(n=12).
8. With LBC block, data is more reliable but takes longer time due to repeated sending if an error is detected. Apart from this, number of bits of payload increase by 1.5 times the original data for n= 12 and k =8.



```
Tx: Abhishek011413311
Pattern found at index 2
Offset = 160
Rx: Jerry010953501
```

Figure 11. Console Output

Achieved following flow control using LISA. Refer below figure.

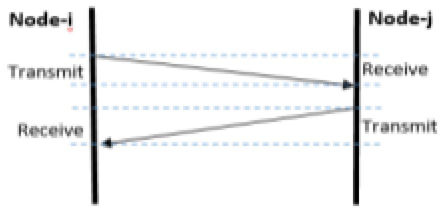


Figure 12. Flow Control

5. Conclusion

Setup was successfully able to transfer data between Ni and Nj over both wired and wireless channel. Receiver was able to extract payload based on LISA. It even worked for slightly noisy channel. With scrambling and descrambling, system was able to extract original data. Error upto 2 bit was properly detected. Code at the receiver can be optimized to use less memory for storing data. Flexible length of sync pattern, scrambling/descrambling order can be implemented depending upon noise in the channel .

6. References

[1] H. Li, "Author Guidelines for CMPE 146/242 Project Report", Computer Engineering Department, College of Engineering, San Jose State University, March 6, 2006, pp. 1.
[2] H. Li. "*Lecture Notes of CMPE 245",* Computer Engineering Department, College of Engineering, San Jose State University, March 6, 2006, pp. 1.
[3] LPC1769 datasheet
[4] LM7805 datasheet

7. Appendix

7.1 Bill of Materials

Item No.	Components	Label	Quantity	Note
1	9V DC power supply	-	1	Powers board
2	Power Jack	J1	1	Jack for power supply
3	Capacitors	C1,	2	Filter capacitors

	(2.2µF)	C2		
4	Resistor (12KΩ)	R1	1	Current limiting
5	LM7805	Q1	1	Voltage regulator
6	Red LED	D1, D2	2	Power LED, RX LED
7	Green LED	D3	1	TX LED
8	Push Button Switch	SW1	1	Reset switch
9	Sliding Switch	SW2	1	Digital Input
10	RJ45 connector	J2, J3, J4	3	Embedded board to RF module, Wired communication
11	LPCXpresso 1769	M1	1	Processing unit
12	433MHz RF transmitter	TX	1	RF transmission
13	433MHz RF receiver	RX	1	RF receiver
14	Male pin head	U2, U3	2	Power and ground testing
15	28 pin female header	U4, U5	2	Connector for LPCXpresso
15	PCB standoffs	-	8	Hold Embedded and RF board
16	CAT-5 cable	C1	1	For Connecting Embedded and RF board.

7.2 Source Code

```
/*
==============================================
================================
Name        : Lab3.c
Author      : Abhishek Singh
Version     : 1
Copyright   : $(copyright)
Description : main definition
==============================================
================================
*/

#ifdef __USE_CMSIS
#include "LPC17xx.h"
#endif

#include <cr_section_macros.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include <time.h>
#include "lbc.h"

#define START_BYTES 32
#define DATA_BYTES  32
#define BUFFER_SIZE 1024
#define CORRUPTION_RANGE (5*8)

#define TX_RX_PORT_ABHI 2
#define LED_TX_RX_PORT_ABHI 2
#define TX_PIN_ABHI 7
#define RX_PIN_ABHI 8
#define LED_TX_PIN_ABHI 11
#define LED_RX_PIN_ABHI 12

#define TX_RX_PORT_JERRY 2
#define LED_TX_RX_PORT_JERRY 0
#define TX_PIN_JERRY 12
#define RX_PIN_JERRY 11
#define LED_TX_PIN_JERRY 3
```

```
#define LED_RX_PIN_JERRY 2

#define delayval 0.5
#define ORDER 7
#define DELAY1 100
//DEMO 2
//#define DELAY2 78000 // 10bytes/sec
//#define DELAY2 15000 // 100bytes/sec
//#define DELAY2 150   // 1kbps working on it

//DEMO 3
#define DELAY2 60000 // 10bytes/sec
//#define DELAY2 10000 // 100bytes/sec
//#define DELAY2 150   // 1kbps working on it


#define ABHI 1

static char lookup[START_BYTES] ={ 0x50, 0x51,
0x52, 0x53, 0x54, 0x55, 0x56, 0x57,
                                           0x58,
0x59, 0x5A, 0x5B, 0x5C, 0x5D,0x5E, 0x5F,

                        0xA0, 0xA1, 0xA2, 0xA3,
0xA4, 0xA5, 0xA6, 0xA7,

                        0xA8, 0xA9, 0xAA, 0xAB,
0xAC, 0xAD, 0xAE, 0xAF
                                     };

signed int front = -1;
signed int rear = -1;
char *queue = NULL;
uint32_t matchedIndex = 0;
signed int lookupPosition = -1;
signed int found = 0;
int rx_size;

//Initialize the port and pin as outputs.
void   GPIOinitOut(uint8_t   portNum,   uint32_t
pinNum)
{
       if (portNum == 0)
       {
              LPC_GPIO0->FIODIR   |=   (1   <<
pinNum);
       }
       else if (portNum == 1)
       {
              LPC_GPIO1->FIODIR   |=   (1   <<
pinNum);
       }
       else if (portNum == 2)
       {
              LPC_GPIO2->FIODIR   |=   (1   <<
pinNum);
       }
       else
       {
              puts("Not a valid port!\n");
       }
}

//Initialize the port and pin as input.
void   GPIOinitIn(uint8_t   portNum,   uint32_t
pinNum)
{
       if (portNum == 0)
       {
              LPC_GPIO0->FIODIR   &=   ~(1   <<
pinNum);
       }
       else if (portNum == 1)
       {
              LPC_GPIO1->FIODIR   &=   ~(1   <<
pinNum);
       }
       else if (portNum == 2)
       {
              LPC_GPIO2->FIODIR   &=   ~(1   <<
pinNum);
       }
       else
       {
              puts("Not a valid port!\n");
       }
}

void setGPIO(uint8_t portNum, uint32_t pinNum)
{
	if (portNum == 0)
	{
		LPC_GPIO0->FIOSET = (1 << pinNum);
	}
	else if (portNum == 1)
	{
		LPC_GPIO1->FIOSET = (1 << pinNum);
	}
	else if (portNum == 2)
	{
		LPC_GPIO2->FIOSET = (1 << pinNum);
	}
	else
	{
		printf("Not Valid port to set!\n");
	}
}

//Deactivate the pin
void clearGPIO(uint8_t portNum, uint32_t pinNum)
{
	if (portNum == 0)
	{
		LPC_GPIO0->FIOCLR = (1 << pinNum);
	}
	else if (portNum == 1)
	{
		LPC_GPIO1->FIOCLR = (1 << pinNum);
	}
	else if (portNum == 2)
	{
		LPC_GPIO2->FIOCLR = (1 << pinNum);
	}
	else
	{
		printf("Not Valid port to clear!\n");
	}
}

// Generate delay at Tx/Rx side
/*void delay(unsigned long int count)
{
   unsigned long int i;
	for(i = 0; i < count; i++);
}*/

void delay(uint32_t count)
{
	LPC_TIM0->TCR = 0x02;
	LPC_TIM0->PR = 0x00;
	LPC_TIM0->MR0 = count*delayval*(9000000 / 1000-1);
	LPC_TIM0->IR = 0xff;
	LPC_TIM0->MCR = 0x04;
	LPC_TIM0->TCR = 0x01;
	while (LPC_TIM0->TCR & 0x01);
	return;
}



char *integerToBinary2(char data)
{
	char *str = (char *)malloc(9);
	int i = 0;
	for(i=7; i>=0; i--)
		str[i] = ((data >> i) & 1);
   str[8] = '\0';
	return str;
}

char binaryToString(char *data)
```

```c
{
   int i;
   int d = 0;
   for(i = 0; i < 8; i++)
   {
       d += pow(2, i)*data[i];
   }
   int c = d;
   return c;
}

void enqueue(char data)
{
   front++;
   if(front == (BUFFER_SIZE - 1)){
       printf("Queue Full\n");
       return;
   }
   queue[front] = data;
}

void enqueueBinary(char data)
{
        char *str = integerToBinary2(data);

        int i = 0;
        for(i=0; i<8; i++)
        {
                enqueue(str[i]);
        }
}

char dequeue()
{
  rear++;
  return queue[rear];
}

int isQueueFull()
{
   if(rear < front)
       return 1;
   else
       return 0;
}

// Add sync data in binary format to queue
void addStartPattern()
{
   int i;
   for(i=0; i<START_BYTES; i++){
       enqueueBinary(lookup[i]);
   }
}

// Add data in binary format to queue
void addData(char *data)
{
   while(*data != '\0'){
       enqueueBinary(*data);
       data++;
   }
}

//  Calculate  offset  for  payload  once  sync
pattern is found
int calculateOffset(char data)
{
   unsigned int lowerNibble = 0;
   unsigned int upperNibble = 0;
   unsigned int offset = 0;

   lowerNibble = (data & 0x0F);
   upperNibble = (((data>>4) & 0x0F));

   if(upperNibble == 5){
       offset += 16;
   }
   offset += (15 - lowerNibble);

   return (offset*8);
}

// Make data with sync pattern
int matchStartPattern(char data)
{
   unsigned int i = 0;
    while((i < START_BYTES)  && (lookupPosition
== -1)){
       if(data == lookup[i]){
           matchedIndex = i;
           lookupPosition = i;
           return 1;
       }
       i++;
   }
   if(lookupPosition != -1)
   {
        lookupPosition++;
       if(data == lookup[lookupPosition]){
           matchedIndex = i;
           return 1;
       }
   }
   return 0;
}

// Extarct data from queue and make for sync
pattern, if match found return start
// address of payload
void  extractData(char  *data,  unsigned  int
matchLevel)
{
   unsigned int offset;
   unsigned int flag = 0;
   unsigned int count = 0;
   int match = -1;

   int prev[8] = {0};
   unsigned int flag1 = 0;

   while(isQueueFull()){
       unsigned int i = 0;
       unsigned int d = 0;
       if(flag1 == 0)
       {
           for(i = 0; i < 8; i++)
           {
               prev[i] = dequeue();
               d += pow(2, i)*prev[i];
           }
           flag1 = 1;
       }
       else
       {
           for(i = 0; i < 7; i++)
           {
               prev[i] = prev[i+1];
               d += pow(2, i)*prev[i];
           }
           prev[i] = dequeue();
           d += pow(2, i)*prev[i];
       }
       if(matchStartPattern((char)d)){
           if(flag == 0){
               match = matchedIndex;
               flag = 1;
           }
           flag1 = 0;
           count++;
       }else{
           match = -1;
           flag = 0;
           count = 0;
           flag1 = 1;
           lookupPosition = -1;
       }

       if(flag == 1 && (count == matchLevel)){
                printf("Pattern found at index
%d\n", match);
           offset = calculateOffset(d);
           printf("Offset = %d\n", offset);
           matchedIndex = 0;
           lookupPosition = -1;
           found = 1;
           break;
       }
   }

   while(offset--){
       dequeue();
   }

   int i = 0;
   while(isQueueFull())
   {
       data[i] = dequeue();
       i++;
       if(i == 7)
       {
        rx_size = binaryToString(data);
        printf("%d\n", rx_size);
       }
       if(i == (rx_size*12 + 8))
        break;
   }
//    data[i] = '\0';
}

// Corrupt sync pattern, This is needed to
simulation noise in channel.
void corruptStartSequence()
{
   srand(time(NULL));
   int i;
   for(i=0; i < 5; i++){
                    int   corruptionIndex   =
rand()%CORRUPTION_RANGE;
                printf("Corrupt  Index:  %d\n",
corruptionIndex);
                      queue[corruptionIndex]   =
~queue[corruptionIndex];
   }
}

void scramble_data(int order, char *s_data, int
data_size)
{
        int i = 0;
        char x1, x2;
        for(i = 0; i < data_size; i++)
        {
                if(i >= ((order/2) + 1))
                {
                        x1    =    s_data[i    -
((order/2) + 1)];
                }
                else
                {
                        x1 = 0;
                }
                if(i >= order)
                {
                        x2 = s_data[i - order];
                }
                else
                {
                        x2 = 0;
                }
                s_data[i] = (queue[(33*8) + i] ^
x1 ^ x2);
        }
}

void descramble_data(int order, char *des_data,
char *data, int data_size)
{
        int i = 0;
        char x1, x2;
        for(i = 0; i < data_size; i++)
        {
                if(i >= ((order/2) + 1))
                {
                        x1 = data[i - ((order/2)
+ 1)];
                }
                else
                {
                        x1 = 0;
                }

                if(i >= order)
                {
                        x2 = data[i - order];
                }
                else
                {
                        x2 = 0;
                }
                des_data[i]   =   (x1   ^   x2)   ^
data[i];
        }
}

void lbc_encode(char *dest, int data_size, char
*src)
{
   char G[8][12] = {G0, G1, G2, G3, G4, G5, G6,
G7};
   int k;
   int z = 0;
   for(k = 0; k < (data_size); k+=8)
   {
       int i, j;
       for(i = 0; i < 12; i++)
       {
           for(j = 0; j < 8; j++)
           {
                dest[z + i] += (G[j][i]*src[k +
j]);
           }
           dest[z + i] = (dest[z + i]%2);
       }
       z += 12;
   }
}

bool check_error(int data_size, char *temp)
{
   char T[12][4] = {T0, T1, T2, T3, T4, T5, T6,
T7, T8, T9, T10, T11};
   char R[1024] = {0};

   int k;
   int z = 0;
   for(k = 0; k < (data_size); k+=12)
   {
       int i, j;
       for(i = 0; i < 4; i++)
       {
          for(j = 0; j < 12; j++)
          {
                R[z + i] += (T[j][i]*temp[k +
j]);
          }
          R[z + i] = (R[z + i]%2);
          if(R[z + i] == 1)
              return false;
       }
       z += 4;
   }
   return true;
}

void lbc_decode(char *dest, int data_size, char
*src)
{
   int i = 0;
   int k = 0;
   for(i = 0; i < (data_size); i += 12)
   {
       int j = 0;
       for(j = 0; j < 8; j++)
       {
           dest[j + k] = src[i + j];
       }
       k +=8;
   }
}

bool Tx()
{
       char txStr[32];
       printf("Data to be transmitted = ");
       scanf("%s", &txStr);
       getchar();
       getchar();
#ifdef ABHI
       printf("Start Tx Logic\n");
       //char *txStr= "Abhishek011413311";
       front = -1;
               GPIOinitOut(LED_TX_RX_PORT_ABHI,
LED_TX_PIN_ABHI);
```

```
   GPIOinitOut(TX_RX_PORT_ABHI, TX_PIN_ABHI);
        clearGPIO(LED_TX_RX_PORT_ABHI,
LED_TX_PIN_ABHI);
        if(queue == NULL)
        {
                queue           =           (char
*)malloc(sizeof(char)*BUFFER_SIZE);
                if(queue == NULL)
                {
        //return -1;
                }
        }
                 char      *data      =      (char
*)malloc(sizeof(char)*DATA_BYTES*8);
   if(data == NULL){
        //return -1;
   }
   printf("Tx: %s\n", txStr);
   strcpy(data, txStr);
        addStartPattern();
        char len = (strlen(txStr) + 2);
        printf("%d\n", len);
        enqueueBinary(len);
        enqueueBinary('A');
        enqueueBinary('D');
        addData(data);

        // Scrambling
        char s_data[1024];
        scramble_data(ORDER, s_data, len*8);
        memcpy(queue + 256 + 8, s_data, len*8);

        // LBC encoding
        char lbc_data[2048] = {0};
        lbc_encode(lbc_data, len*8, queue + 256
+ 8);
        memcpy(queue   +   256   +   8,   lbc_data,
len*12);

        //queue[270]    =    !queue[270];    //To
introduce 1 bit error

   int i = 0;
   for(i = 0; i < (front + (len*8*0.5)); i++)
        {
                if(queue[i])
                {
setGPIO(LED_TX_RX_PORT_ABHI, LED_TX_PIN_ABHI);
                        setGPIO(TX_RX_PORT_ABHI,
TX_PIN_ABHI);
                        delay(DELAY1);
                }
                else
                {
clearGPIO(LED_TX_RX_PORT_ABHI,
LED_TX_PIN_ABHI);
clearGPIO(TX_RX_PORT_ABHI, TX_PIN_ABHI);
                        delay(DELAY1);
                }
        }
        clearGPIO(LED_TX_RX_PORT_ABHI,
LED_TX_PIN_ABHI);
        clearGPIO(TX_RX_PORT_ABHI, TX_PIN_ABHI);
        GPIOinitOut(LED_TX_RX_PORT_ABHI,
LED_RX_PIN_ABHI);
   GPIOinitIn(TX_RX_PORT_ABHI, RX_PIN_ABHI);
        clearGPIO(LED_TX_RX_PORT_ABHI,
LED_RX_PIN_ABHI);

        front = 0;
        memset(queue, 0, BUFFER_SIZE);
        memset(data, 0, DATA_BYTES*8);
        delay(DELAY2);
        printf("TX Done\n");

        printf("Start Rx Logic\n");
        while(1)
        {
                if  (LPC_GPIO2->FIOPIN  &  (1  <<
RX_PIN_ABHI)) //Wireless
                {
                        queue[front] = 1;
setGPIO(LED_TX_RX_PORT_ABHI, LED_RX_PIN_ABHI);
               }
               else
               {
                       queue[front] = 0;

clearGPIO(LED_TX_RX_PORT_ABHI,
LED_RX_PIN_ABHI);
               }

               front++;
               delay(DELAY1);
               if(front == (BUFFER_SIZE - 1))
               {
                       break;
               }
       }
       rear = -1;
       lookupPosition = -1;
       extractData(data, 10); // Extract actual
data from stream of data

       // LBC decoding
       if(check_error(rx_size*12, lbc_data))
       {
               char  lbc_received_data[1024]  =
{0};
               lbc_decode(lbc_received_data,
rx_size*12, data + 8);
               memcpy(data,  lbc_received_data,
rx_size*8);
       }
       else
       {
               printf("Error!!\n");
               return false;
       }

       // Descrambling
       char des_data[1024];
       descramble_data(ORDER,  des_data,  data,
rx_size*8);
       memcpy(data, des_data, rx_size*8);

       char *ptr;
       int k = 0;
       printf("Rx: ");
       for(ptr = (data + 16); k < ((rx_size -
2)*8); k+=8)
       {
               printf("%c",
binaryToString(ptr));
               ptr += 8;
       }
       printf("\n");
       memset(queue, 0, BUFFER_SIZE);
       clearGPIO(LED_TX_RX_PORT_ABHI,
LED_RX_PIN_ABHI);
       printf("Rx Done!\n");
#else
#endif
       return true;
}

void Rx()
{
       getchar();
       getchar();
#ifdef ABHI
       // Rx Logic
       printf("Start Rx Logic\n");
       GPIOinitOut(LED_TX_RX_PORT_ABHI,
LED_RX_PIN_ABHI);
   GPIOinitIn(TX_RX_PORT_ABHI, RX_PIN_ABHI);
       clearGPIO(LED_TX_RX_PORT_ABHI,
LED_RX_PIN_ABHI);
       if(queue == NULL)
       {
               queue          =          (char
*)malloc(sizeof(char)*BUFFER_SIZE);
               if(queue == NULL){
                       //return -1;
               }
       }
```

```
                char     *data     =     (char
*)malloc(sizeof(char)*DATA_BYTES*8);
   if(data == NULL){
       //return -1;
   }

       memset(queue, 0, BUFFER_SIZE);
       memset(data, 0, DATA_BYTES*8);
       front = 0;
       // Recieve data
       while(1)
       {
               if  (LPC_GPIO2->FIOPIN  &  (1  <<
RX_PIN_ABHI)) //Wireless
               {
                       queue[front] = 1;

setGPIO(LED_TX_RX_PORT_ABHI, LED_RX_PIN_ABHI);
               }
               else
               {
                       queue[front] = 0;

clearGPIO(LED_TX_RX_PORT_ABHI,
LED_RX_PIN_ABHI);
               }

               front++;
               delay(DELAY1);
               if(front == (BUFFER_SIZE - 1))
               {
                       break;
               }
       }
       rear = -1;
       lookupPosition = -1;
       extractData(data, 10); // Extract actual
data from stream of data

       // LBC decoding
       if(check_error(rx_size*12, data + 8))
       {
               char   lbc_received_data[1024]   =
{0};
               lbc_decode(lbc_received_data,
rx_size*12, data + 8);
               memcpy(data,   lbc_received_data,
rx_size*8);
       }
       else
       {
               printf("Error!!\n");
               return;
       }

       // Descrambling
       char des_data[1024];
       descramble_data(ORDER,  des_data,  data,
rx_size*8);
       memcpy(data, des_data, rx_size*8);

       printf("Rx: ");
       char *ptr;
       int k = 0;
       for(ptr = (data + 16); k < ((rx_size -
2)*8); k+=8)
       {
               printf("%c",
binaryToString(ptr));
               ptr += 8;
       }
       printf("\n");


       // Tx Logic
       printf("Start Tx Logic\n");
       char *txStr= "Abhishek011413311";
       printf("Tx: %s\n", txStr);
       memset(queue, 0, BUFFER_SIZE);
       memset(data, 0, DATA_BYTES*8);



       clearGPIO(LED_TX_RX_PORT_ABHI,
LED_RX_PIN_ABHI);
```

```
                GPIOinitOut(LED_TX_RX_PORT_ABHI,
LED_TX_PIN_ABHI);
   GPIOinitOut(TX_RX_PORT_ABHI, TX_PIN_ABHI);
        clearGPIO(LED_TX_RX_PORT_ABHI,
LED_TX_PIN_ABHI);

        front = -1;
   strcpy(data, txStr);
        addStartPattern();
        char len = (strlen(txStr) + 2);
        printf("%d\n", len);
        enqueueBinary(len);
        enqueueBinary('A');
        enqueueBinary('D');
        addData(data);

        // Scrambling
        char s_data[1024];
        scramble_data(ORDER, s_data, len*8);
        memcpy(queue + 256 + 8, s_data, len*8);

        // LBC encoding
        char lbc_data[2048] = {0};
        lbc_encode(lbc_data, len*8, queue + 256
+ 8);
        memcpy(queue  +  256  +  8,  lbc_data,
len*12);

        //queue[270]   =   !queue[270];   //To
introduce 1 bit error

   int i = 0;
   for(i = 0; i < (front + (len*8*0.5)); i++)
        {
                if(queue[i])
                {

setGPIO(LED_TX_RX_PORT_ABHI, LED_TX_PIN_ABHI);
                        setGPIO(TX_RX_PORT_ABHI,
TX_PIN_ABHI);
                        delay(DELAY1);
                }
                else
                {

clearGPIO(LED_TX_RX_PORT_ABHI,
LED_TX_PIN_ABHI);

clearGPIO(TX_RX_PORT_ABHI, TX_PIN_ABHI);
                        delay(DELAY1);
                }
        }
        clearGPIO(LED_TX_RX_PORT_ABHI,
LED_TX_PIN_ABHI);
        clearGPIO(TX_RX_PORT_ABHI, TX_PIN_ABHI);
        printf("Tx Done!\n");
#else
#endif
}


int main(void)
{
        printf("Welcome to LISA!\n");
        while(1)
        {
                printf("Please select any of the
following      options\n1.      Transmitter\n2.
Receiver\nChoice = ");
                int choice = 0;
                scanf("%d", &choice);
                switch(choice)
                {
                        case    1:    printf("Tx
selected\n");
                                Tx();
                                break;
                        case    2:    printf("Rx
selected\n");
                                Rx();
                                break;
                        default:printf("Invalid
choice! Try again\n");
                }
        }
   return 0;
```

```
}


/*
* lbc.h
*
*  Created on: 06-Dec-2016
*      Author: abhishek
*/

#ifndef LBC_H_
#define LBC_H_

#define G0 1, 0, 0, 0, 0, 0, 0, 0,  1, 1, 0, 0
#define G1 0, 1, 0, 0, 0, 0, 0, 0,  0, 1, 1, 0
#define G2 0, 0, 1, 0, 0, 0, 0, 0,  0, 0, 1, 1
#define G3 0, 0, 0, 1, 0, 0, 0, 0,  1, 0, 0, 1
#define G4 0, 0, 0, 0, 1, 0, 0, 0,  1, 0, 1, 0
#define G5 0, 0, 0, 0, 0, 1, 0, 0,  0, 1, 0, 1
#define G6 0, 0, 0, 0, 0, 0, 1, 0,  1, 1, 1, 0
#define G7 0, 0, 0, 0, 0, 0, 0, 1,  0, 1, 1, 1

#define T0  1, 1, 0, 0
#define T1  0, 1, 1, 0
#define T2  0, 0, 1, 1
#define T3  1, 0, 0, 1
#define T4  1, 0, 1, 0
#define T5  0, 1, 0, 1
#define T6  1, 1, 1, 0
#define T7  0, 1, 1, 1

#define T8  1, 0, 0, 0
#define T9  0, 1, 0, 0
#define T10 0, 0, 1, 0
#define T11 0, 0, 0, 1

#endif /* LBC_H_ */
```