Implementation of LISA Processing Engine Using RF Module

Vidushi Jain (013859547)
Computer Engineering Department, College of Engineering
San Jose State University, San Jose, CA 94303
E-mail: vidushi.jain@sjsu.edu

Abstract

This paper aims to establish wireless communication between two nodes (Ni and Nj) using RF Module which is interfaced with LPC1769, 32bit ARM Cortex-M3 microcontroller. This can be achieved by implementing LISA algorithm for synchronization and verification of payload. The main challenge was to achieve the high-speed data rate with oversampling i.e. 1kbps with 100% confidence level. The challenges were overcome by thorough understanding of the Timer-Interrupts. RF communication operate in ISM range 433MHz. This report covers both the aspects of the design hardware and software to reproduce the design.Using MCUXpresso as Integrated Development Environment for debug and run the code for rotating squares and random trees the goal was accomplished.

1. Introduction

The objective of this lab is to design and implement the LISA processing engine on LPC1769. For this, a RJ45 of CAT5 category is interfaced to LPC1769Xpresso. For design, debug and test the software part of the lab, MCUXpresso, an Integrated Development Environment is used and NXP Timer Interrupt reference code is taken as base code for this implementation. To test input/output ports of LPC1769 module we have performed GPIO testing just to toggle TX pin based on the set/reset bit in the payload. After a successful testing of GPIO Ports we have integrated our LISA in the Timer Interrupt code which is followed by landline testing between two LPC1769 using Ethernet cable. LISA consists of sync field of predefined length and payload. Sync field identifies starting of payload which contains message. This makes receiver in synchronize with transmitter. At transmitter end we are sending stream of data at sampling frequency but at the receiver end we are doing oversampling i.e. multiply by 10.so, that we can get 10 samples/bit. To maintain data synchronization.

The major technical challenge with this lab is to receive data at 1kpbs bit rate after oversampling. With detailed understanding of the Timer-Interrupts and thorough analysis of samplers per bit helps us to overcome this challenge.

2. Methodology

This section explains the overall system layout from software as well as hardware aspects and This section also describes in-depth goals and challenges of the project and the design implemented to solve those challenges.

2.1. Objectives and Technical Challenges

The following Objectives are taken into consideration:

1. Understanding how Timer-Interrupts works at software level as well as at hardware level.
2. Understanding of LISA and its relevance in data synchronization through matching sync fields and payload extraction out of received data.
3. Designing of Embedded board as well as RF board with a power circuit to provide continues 5V supply to the platform.
4. Debugging of hardware as well as software using on board LED's or external LED's.
5. Study working of Receiver and Transmitter modules and interface them with the LPC1769 board.
6. Develop a software program to transmit data and receive data through the GPIO peripheral controller.

Technical challenges faced during implementation:

1. Soldering the LPC1769Xpresso module and RJ45 on wire wrapping board along with power supply and testing it for working.
2. Study of the pin diagram of LPC1769, RJ45 Port and RF modules to interface to LPC as per the requirement.

2.2 Problem Formulation and Design

LISA algorithm is implemented to get synchronization for wireless communication between two embedded systems. Setting up Wireless Communication necessitates hardware-software design for the whole system. Testing with Land-line Communication and then implementing and verify the wireless Communication between two boards is the final stage of this lab.

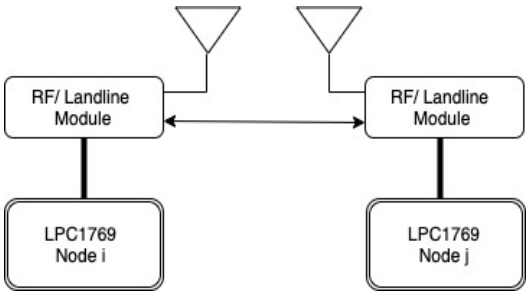

Figure 1: System Design Block

Power Source for system requires 5V steady voltage. We implemented voltage regulator down converting from 9V to 5V. Microcontroller and RF board are connected using

RJ45 connector. The program for these functionality is flashed on controller. There is a symmetrical system on the other side which is ready for transmission/receiving signals. System design can be shown by following diagram:

3. Implementation

We implemented its software programming on Embedded C platform using MCUXpresspo IDE.

Program design flow is as follows:

- Predefined sync field of 32 bytes, payload and arbitrary data fills buffer of 1K.
- After running code on Node-i and in Node-j embedded board user need to enter sampling rate & oversampling rate.
- Node-i and Node-j is configurable as transmitter & receiver based on user choice.
- Say, Node-i act as transmitter and sends data to j Node along with the sync code and some arbitrary data.
- Node-j acts as a receiver and receives data and fill its local buffer and asked user to enter confidence level.
- Start of payload is identified as per given confidence level and using predefined sync field in bit stream.
- Then Payload will be printed on the output screen.

3.1. Hardware Design

The Power circuit comprising of extern power supply of 3.3-5V produces regulated output which is given to the RF Modules and for LPC1769 Development board we are using 5v USB Cable. Pin1 of the LPC1769 board is Ground while Pin 28 is VCC. Pin2 of Port0 is configured as Output while Pin3 of Port0 is configured as Input through software. We are using One DIP Switch to use same RF board for Wired and wireless testing. By only turning-on 3 and 5 button of dip switch we can test RF Modules as data pins of TX and RX are connected to Pin3 and Pin5 of dip switch. However, for wired testing we are just turning on 4 and 6 pin of Dip Switch.

RJ45 connector is used to connect the Embedded board with the RF board. Transmitter and receiver RF Modules are given 5V supply from the external power supply and grounds are common with RJ45 connector. Six LED's are placed on each RF board to debug the RJ45 TX/RX and RF Module TX/RX and for GPIO pins data transmission & reception.

Following is the table for Pin Connections between LPC1769, RF module and RJ45 connector:

LPC1769 board	RF board	RJ45 connector
VCC	VCC	Pin 8
GND	GND	Pin 2
P0.2 (GPIO)	Transmitter	Pin 6
P0.3 (GPIO)	Receiver	Pin 4

Table 1 – Pin Connections

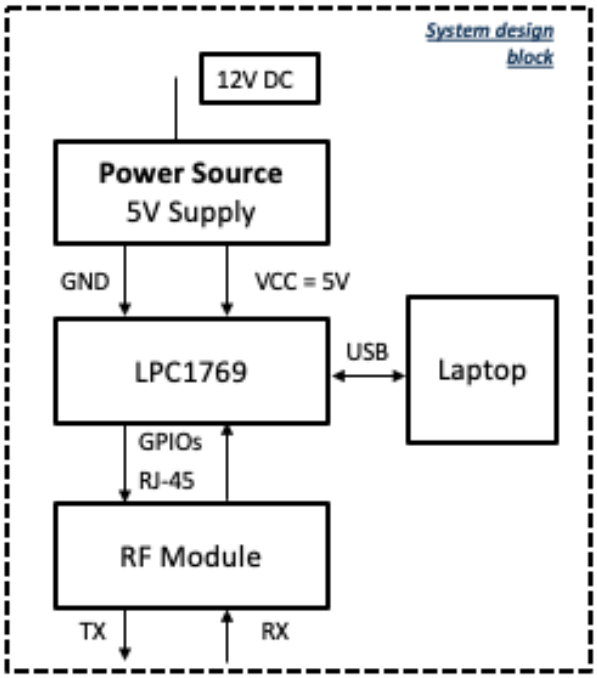

Figure 2. Pin diagram of System Design

3.1.1 System Layout

The components are connected as shown in the figure 1.

Pin Connections of LPC1769 to RJ45 and Nodes to RF Module Transmitter & Receiver

Pin No.	Functionality	Notes
J2-1	GND	Ground
J2-28	VCC	Power-In of LPC 1769
J2-21	GPP P0.2 port Output	GPIO Pin used as Output during RF communication
J2-22	GPP P0.3 port Input	LPC GPIO Pin used as Input during RF communication

Table 2 – Pin connections of LPC to RJ45

Pin Nos.	Description	Notes
1	VCC (5-12v DC)	External power supply (Wall mount adaptor)
2	Data Pin	LPC1769 GPP P0.2
3	GND	Common GND
4	Antenna (lambda/4)	23-33cm, matching impedance = 50 ohm

Table 3 – Pin Connections of RF Transmitter to Node-i

Pin Nos.	Description	Notes
1	Antenna	23-33 cms (Wire)
2 and 3	GND	Common GND LPC Pin J2-1
4	Data Pin	LPC 1769 GPP P0.3
5	Vcc (5v DC)	External Power

Table 4 – Pin Connections of RF Receiver to Node-j

3.1.2 Schematic Designs of boards

3.1.1 LPC Embedded Board

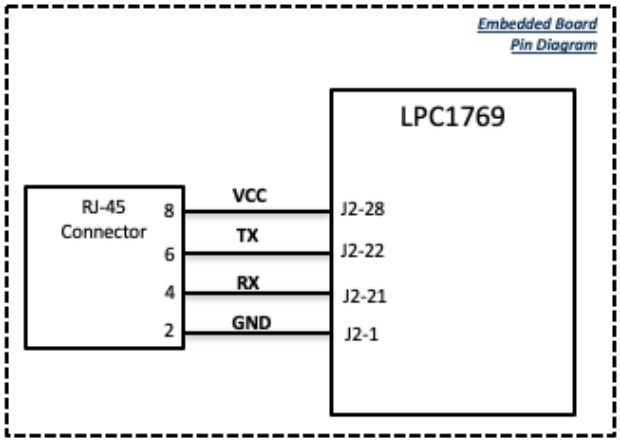

Figure 3. Pin diagram of LPC and RJ-45 Connector

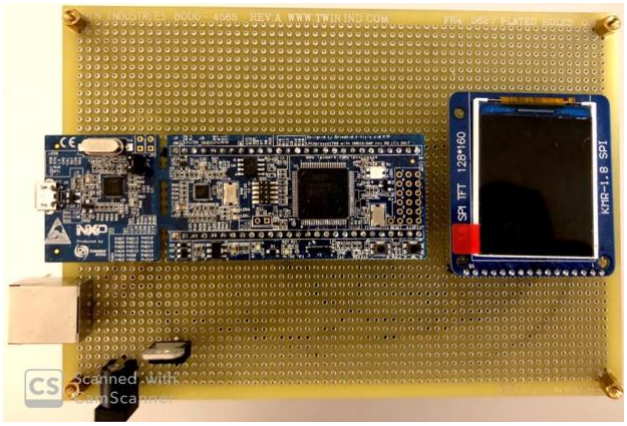

Figure 4. LPC1769 Embedded board

3.1.2 RF Board

The connections of a RF board along with external power supply is shown below.

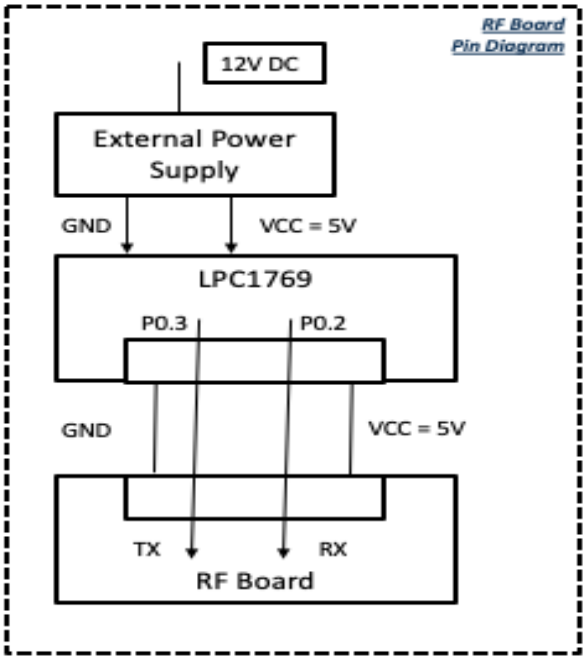

Figure 5. Pin diagram of LPC and RF Board

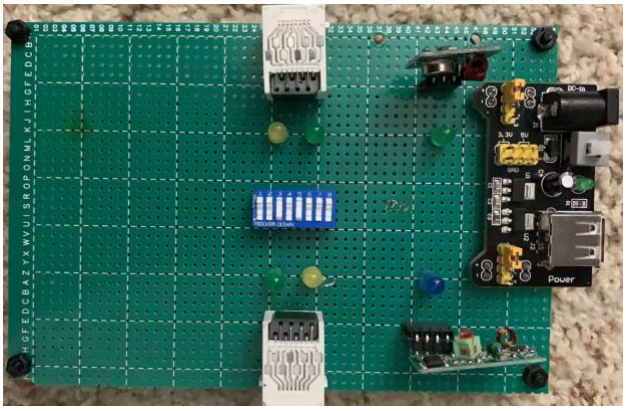

Figure 6. RF Board with Transmitter & Receiver

Power to the RF Module circuit is coming from external power supply connected on RF Board and RJ-45 ground connections are also shorted along the RF Module grounds.

3.1.3 RF board interfaced with Embedded board

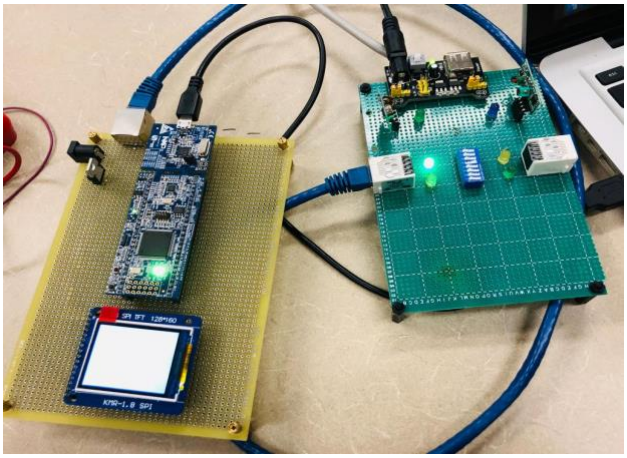

Figure 7. LPC-RF board in Connected State

3.1.3 Bill of Materials

S.No	Name and Description	Quantity
1	LPC Xpresso Module (ARM Cortex – M3)	1
2	External 3.3-5V Power Supply	1
3	Printed Circuit Board	2
4	Power Adaptor	1
5	LED's(Green/Yellow/Blue)	6
6	DIP switch	1
7	Registers	1
8	Receiver module (Rx)	1
9	Transmitter module (Tx)	1
10	RJ-45 Connectors	3
11	RJ-45 Cables	3
12	Stand-offs	10
13	Single Core Wires	1 pack

Table 5. Bill of Materials

Pin Nos.	Description	Notes
J1	RJ-45 I/F with Embedded Board	4 POS (Vcc, GND, GPPx, GPPy)
J2	RJ-45 I/F as Landline/RF testing	Vcc, GND, Tx, Rx
RF1	Receiver Rx (RF Module)	RF Receiver module
RF2	Transmitter Tx (RF Module)	RF Transmitter Module
D1	Green LED's & Yellow LED's	LED for transmit & receiver testing
DIP Switches	Landline vs RF Selection	Toggle between wired and wireless communication

Table 6 - Bill of Material for RF Board

3.2. Software Design

This section given us the complete software design of the system including flowchart, algorithm and pseudo code.

3.2.1 Algorithm of LISA Implementation

Step 1: Declare a structure with structure member sync code and payload data.
Step 2: Copy static char buffer of sync code and payload to the respective structure member and create a Packet.
Step 3: Ask the user at which sampling frequency & oversampling rate user wants to operate and he wants to operate as a transmitter or as a receiver initially.
Step 4: If the user selects to act as a transmitter, then the user transmits the data by attaching the sync field to the data which was initially entered by the user. After transmission, the module goes into receiving state to get the acknowledgement.
Step 5: After creating a packet convert bytes to bits and fill in transmit data buffer and start transmission.
Step 6: If the user selects to act as a receiver, it will be continuously listening till it gets proper data in which sync field is present. Once it gets data which contains the sync field, it disables interrupts to stop listening and then sends an acknowledgement signal.
Step 7: At receiving end get all the data. After successful receiving of 1024 bits reconvert bits to byte data to feed data to LISA algorithm to verify sync codes and identify the payload
Step 8: Get confidence level from user for LISA algorithm.
Step 9: If calculated confidence level is greater than or equal to user entered confidence the Payload will be identified and displayed on the Output Screen.

3.2.3 Software Flow Chart

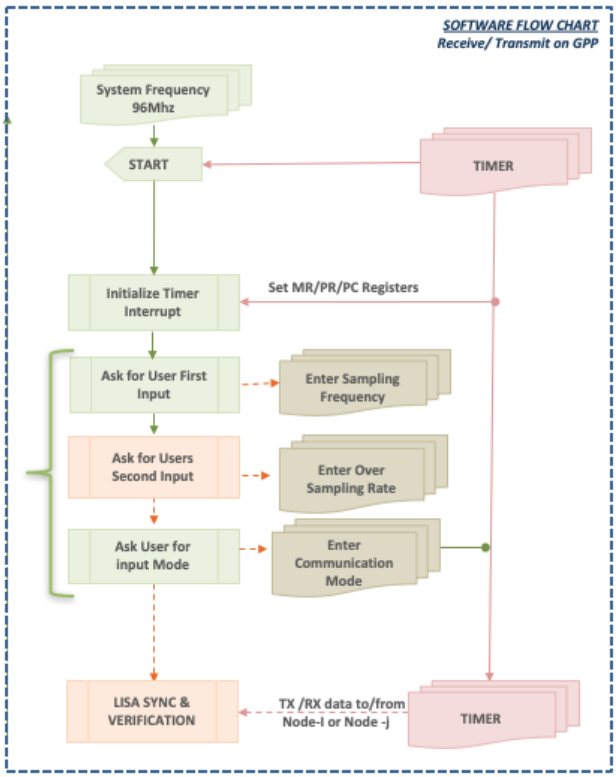

Figure 8. Software Flow Diagram

3.2.4 Node-i and Node-j Flow Control diagram

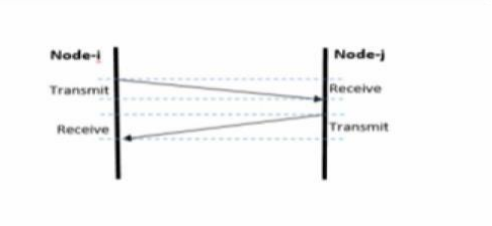

Figure 6. Node-i and Node-j Flow Diagram

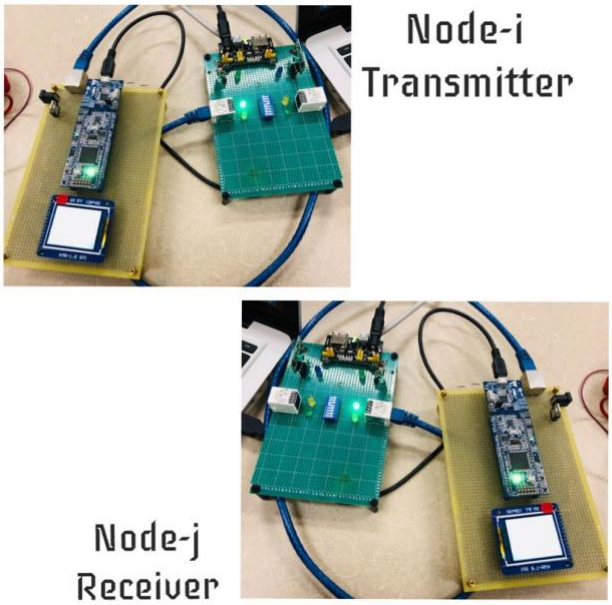

Figure 7. Node-i and Node-j Boards Actual View

4. Testing and Verification

We had carried out thorough testing, first we have written LISA implementation on tests its corner cases on x86 without using embedded platform. Later on, we have interfaced our Lisa algorithm with LPC1769_GPIO along with timer implementation in which software will ask user to choose communication mode whether Node1 will act as a transmitter or receiver based on user input we will perform task by sending data at 50ms and at other node we were receiving at same frequency and identifying payload but this was first verified and tested with wired connections i.e. called landline testing and each functionality in our code at every iteration and verified it with valid tests on received and transmitted data. Then we have migrated our whole code to the timer interrupt handler and first verified & tested sending transmitted data at sampling frequency and receiving data at same sampling frequency along with oversampling frequency with Ethernet wires and then tested wirelessly. Later tested whole system both on Hardware and Software side for wireless Communication. We checked Connectivity on each pin on circuit board inputs and outputs. We debug the error code and resolved bugs before running software program.

5. Conclusion

We were able to establish wired as well as wireless communication between two LPC Embedded nodes using Ethernet cables and using RF Modules. we were also able to integrate our LISA algorithm implementation with RF module and with b) Landline Ethernet cables also. The LISA algorithm was implemented to develop synchronized communication between two wireless embedded systems. We could identify the payload successfully at receiver end @1kbps of frequency. We understood the capability of LISA and got hands-on experience of data transmission on wireless technology. Also, we learnt how to test and debug the hardware design.

6. Acknowledgement

I would like to convey my sincere thanks to Dr. Harry Li for his insightful classes and explanation about Embedded wireless systems in detail. His classes always helped us to understand Wireless technology and his course material is blend of theoretical as well as practical knowledge which helped me to understand and perform this project with thorough understanding and fulfilment.

7. References

[1] H. Li, “Author Guidelines for CMPE 127/245 Project Report”, *Lecture Notes of CMPE 127/245*, Computer Engineering Department, College of Engineering, San Jose State University.

[2] LPCXpresso 1769 datasheet http://www.nxp.com/documents/data_sheet/ Datasheethttp://www.nxp.com/documents/data_sheet/LPC1769_68_67_66_65_64_63.pdf

[3] UM10360 user manual http://www.nxp.com/documents/user_manual/UM10360.pdf

[4] Receiver and Transmitter product specifications on vendor site.

[5] LPCXpresso-LPC1769-CMSIS-DAP https://www.embeddedartists.com/sites/default/files/docs/schematics/LPCXpresso1769_CD_revD1.pdf

8. Appendix

```
/*
===============================================
=====================================
 Name        : timer.c
 Author      : $VIDUSHI JAIN
 Version     : V1.10
 Copyright   : $(copyright)
 Description : RF WIRELESS TESTING
===============================================
=====================================
*/

#include "board.h"
#include <stdio.h>
#include "chip.h"
#include "myLisa.h"
#include "gpio.h"
```

```
#define SAMPLING_RATE_HZ     100
#define OVER_SAMPLING_RATE_HZ      1
#define TICKRATE_HZ1 (11 * SAMPLING_RATE_HZ
* OVER_SAMPLING_RATE_HZ)/* 11 ticks per second
*/
uint32_t sampling_rate,   oversampling_rate;

#define LED_GREEN_PORT 3
#define LED_GREEN_PIN            25
#define MAX_DATA                 1024
#define TX_PIN             2
#define RX_PIN             3
#define DATA_TX_RX_PORT      0

int start_tx = 0;
int start_rx = 0;
int bit_index = 0;
int input_mode = 0;
int counter = 0;
int bit_count = 0;
int check_bit = 0;

void send_one_bit(char* transmit_data, int bit_index)
{
  if(transmit_data[bit_index] == 0x01)
  {
          LPC_GPIO3->SET = (1<<LED_GREEN_PIN);
          LPC_GPIO->SET = (1<<TX_PIN);
  }
  else
  {
          LPC_GPIO3->CLR = (1<<LED_GREEN_PIN);
          LPC_GPIO->CLR = (1<<TX_PIN);
  }
}

/**
 * @brief Handle interrupt from 32-bit timer
 * @return Nothing
 */
void TIMER0_IRQHandler(void)
{
 if(Chip_TIMER_MatchPending(LPC_TIMER0, 1))
 {
   if(start_tx)
   {
          if(bit_index < data_length) {
send_one_bit(tx_data,bit_index);
           bit_index += 1;
          }
          else {
          bit_index = 0;
          start_tx = 0; input_mode = 0;
          LPC_GPIO->CLR = (1<<TX_PIN);
          printf("\nTransmission Done\n");
    }
 }
 else if(input_mode == 2)
 {
          if(LPC_GPIO->PIN & (1 << RX_PIN))
              start_rx = true;

          if(start_rx) {
           if(LPC_GPIO->PIN & (1 << RX_PIN)){
        check_bit += 1;
      }
      else {
            check_bit += 0;
           }
          counter++;

          if (counter == oversampling_rate) {
float sampling = check_bit/ oversampling_rate;
if(sampling < 0.5)
          received_data[bit_count] = '0';
else
 received_data[bit_count] = '1';

bit_count++;
total_data_received++;
counter = 0; check_bit = 0;
}

if(bit_count == MAX_DATA) {
   bit_count = 0;
   data_received = 1;
   start_rx = 0;
 }}}
Chip_TIMER_ClearMatch(LPC_TIMER0, 1);
}}

/**
 * @brief          Main routine for taking input from user
 * @return         Function should not exit.
 */
int main(void)
{
 uint32_t timerFreq,tick_rate;

SystemCoreClockUpdate();
Board_Init();

packet_s P;
create_data_packet(&P);
convert_data_packet(&P);

LPC_GPIO3->DIR |= (1<<LED_GREEN_PIN);
LPC_GPIO->DIR |= (1 << TX_PIN);
LPC_GPIO->DIR &= ~(1 << RX_PIN);

printf("\nPlease Enter Sampling rate:");
scanf("%d",&sampling_rate);
printf("\nPlease Enter Over Sampling rate:");
scanf("%d",&oversampling_rate);

/* Enable timer 1 clock */
Chip_TIMER_Init(LPC_TIMER0);

/* Timer rate is system clock rate */
timerFreq = Chip_Clock_GetSystemClockRate();
tick_rate = (11 * sampling_rate * oversampling_rate) ;
```

```
/* Timer setup for match and interrupt at
TICKRATE_HZ */
Chip_TIMER_Reset(LPC_TIMER0);
Chip_TIMER_MatchEnableInt(LPC_TIMER0, 1);
Chip_TIMER_SetMatch(LPC_TIMER0, 1, (timerFreq /
tick_rate));
Chip_TIMER_ResetOnMatchEnable(LPC_TIMER0, 1);
Chip_TIMER_Enable(LPC_TIMER0);

/* Enable timer interrupt */
NVIC_ClearPendingIRQ(TIMER0_IRQn);
NVIC_EnableIRQ(TIMER0_IRQn);

while (1) {
  if(input_mode == 0){
 printf("\n\nEnter a command to activate TX(1) & RX(2):
\n");
 scanf("%d", &input_mode);
 if(input_mode == 1) {
    start_tx = 1; bit_index = 0;
 }
 else if(input_mode == 2) {
   if(data_received == 1) {
            reconvert_data_packet(received_data,total_data_
received);
            check_confidence_and_ret_payload(&P,data_le
n);

            memset(payload_data,0,data_len);
       total_data_received = 0;
            data_received = 0;
            data_len = 0;
            input_mode = 0;
    }
   }
  }
 __WFI();
}
  return 0;
}


/*
=======================================
=====================================
 Name        : myLisa.c
 Author      : Vidushi Jain
 Version     : V1.00
 Description : LISA HOMEWORK ASSIGNMENT
=======================================
=====================================
*/

#include "myLisa.h"
#include <stdio.h>


char arbitrary_data[] = "Embedded Wireless Architecture
Description Embedded wireless architecture with basic
communication protocols and hands-on labs with state of
the art embedded system development tools. Prerequisites:
CMPE 180A and 180D, classified standing, or instructor
co";

char tx_data[1024];
int data_length = 0;
char received_data[1024];
char data_received = 0;
int total_data_received = 0;
char payload_data[512];

char sync_data[SYNC_SIZE] = {    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};

static int powerofTwo[] =
{
                1,2,4,8,16,32,64,128
};

void convertDataToBinary(char data)
{
        int i = 0;
        for(i=7; i>=0; i--) {
                tx_data[data_length] = ((data >> i) & 1);
                data_length += 1;
        }
}

void create_data_packet(packet_s* P1)
{
        //Add SYNC Data to the TX Packet
        memcpy(P1->bytes, sync_data, SYNC_SIZE);

        //Add Payload to the TX Packet
        memcpy(P1->Payload, MSG, PAYLOAD_SIZE);
}

void convert_data_packet(packet_s* P)
{
        for (int i = 0; i < SYNC_SIZE; i++)
        {
          convertDataToBinary(P->bytes[i]);
        }

        for (int i = 0; i < PAYLOAD_SIZE; i++)
        {
                convertDataToBinary(P->Payload[i]);
        }
}

char convert_8_bits_to_char(const char* bits)
{
        int byte = 0;
        int count = 7;
        for(int i = 0;i < 8;i++)
        {
          byte += (bits[i] - 48) * powerofTwo[count--];
        }
```

```
  // char c = byte - '0';
            return (char)byte;
}
int data_len = 0;

void reconvert_data_packet(char* rx_buff, int size)
{
   printf("\n\n----------\n");
            int bit_count = 0;
            char buffer[8] = {0};

            for (int i = 0; i < size; i++)
            {
     // printf("%u[%d] ",rx_buff[bit_count],bit_count);
                        buffer[bit_count++] = rx_buff[i];
                        if (bit_count == 8) {
                           payload_data[data_len] =
convert_8_bits_to_char(buffer);
          printf("%x ",payload_data[data_len]);
                 // printf("%x
",convert_8_bits_to_char(buffer));
          data_len += 1;
                                    bit_count = 0;
                        }
            }
}


void check_confidence_and_ret_payload(packet_s*
P1,int length)
{
   int confidence_level = 29;
 //  printf("\nPlease Enter Confidence Level:");
 //  scanf("%d",&confidence_level);

   float confidence_percent = 0.0;
   int payload_position = 0,calculated_confidence = 0;
   for(int i=0; i<SYNC_SIZE; i++)
   {
      for(int j = payload_position; j<length; j++)
      {
         if(payload_data[j] == sync_data[i])
         {
            payload_position = j;
            calculated_confidence += 1;
            break;
         }
      }
   }
   printf("Calculated Confidence:%d | Payload Start
Address:%d\n",calculated_confidence,payload_position);


   if(calculated_confidence >= confidence_level)
   {
      confidence_percent =
((float)calculated_confidence/32)*100;
      printf("\nConfidence percent: %f%%\n",
confidence_percent);
detect_and_print_payload(payload_position+1,payload_data
); //Find the payload of the data
   }
   else
   {
      printf("\nSorry! Entered Confidence level is Less than
Calculated Confidence Level");
      printf("\nCan't identified Payload\n");
   }
}

void detect_and_print_payload(int payload_address,char*
buf)
{
           char identified_payload[PAYLOAD_SIZE];
   memcpy(identified_payload,buf +
payload_address,PAYLOAD_SIZE);
   printf("\n-------Payload Identified-------
\n%s",identified_payload);
   printf("\n------------------------------\n");
}
```