**Implementation of LISA Algorithm for landline/wireless communication using RF module**

*Vatsal Makani*

*Graduate Student, Computer Engineering Department, San Jose State University, San Jose, CA 94303*

*E-mail:* [*vatsalkirit.makani@sjsu.edu*](mailto:vatsalkirit.makani@sjsu.edu)

# Abstract

*The purpose of this project is to establish a landline communication and wireless communication between two nodes using RF module interfaced with LPC 1769, 32 bit ARM Cortex-M3 microcontroller. The landline communication was achieved through RF board interfacing through another RF board with CAT5e cable and LISA algorithm for data synchronization. The wireless communication using RF module was also done by implementing LISA algorithm. The data was oversampled at the rate 10 times more than the rate of transmission and checked with the sync field at the receiver end to verify the working of LISA algorithm. The wireless transmission and reception can be achieved by RF module at the data rate of 1kbps.*

# Introduction

To test input and output ports of LPC 1769, we performed GPIO testing followed by Landline testing between two controllers using Ethernet Cable, RJ-45 of CAT5 category. Then we Implemented Linear Invariant Synchronization Algorithm (LISA) in code which consists of sync field of predefined length and payload. Sync field identifies starting of payload which contains message. The message is extracted from the payload and displayed at the receiver end. This makes receiver in synchronize with transmitter.

In sending data steam, it might be possible that if consecutive ones or zeros are sent by transmitter at the rate 10X more than receiver, then even if we identify starting of the message using sync field, receiver might lose some bits of same ones and zeros. To eliminate this problem, we implemented oversampling technique which is taking the average of all the bit streams sent by the transmitter. Once the buffer is full, we can take the average of all the data and decide bit 0 or bit 1 based on number of 0s and 1s. Hence, when the transmitter transmits 100 bits, the receiver shall take the average and receive 10 bits to avoid data loss. Here we get bit stream based on message to be transmitted.

# Methodology

This section describes in-depth goals and challenges of the project and the design implemented to solve those challenges.

# Objectives and Technical Challenges

The objectives of the project can be enumerated as follows:

1. **LISA Algorithm**
   * + Design a power circuit to provide continues 5V supply to the platform.
     + Design the Project board with debugging capabilities wherein testing is possible with LED, switches, and other components.
     + Study working of Receiver and Transmitter modules and interface them with the LPC1769 board.
     + Develop a software program that shall send the data in form of sync field (32 bytes) and payload. Sync field shall contain the payload which indicated the start of message.
     + Extracting the message transmitted at the receiver end with the help of payload and sync field.
2. **Landline and wireless testing**
   * + Develop a board called RF board that shall interface with the microcontroller board using CAT5e cable and RJ45 connectors.
     + Interface the RF board with another RF board and microcontroller which creates a wired communication.
     + Interface GPIO pins of microcontroller with RJ45 connector with the common ground.
     + Develop a software program to transmit data and also receive data through the GPIO interface of microcontroller based on LISA algorithm.
     + The power circuit needs to be designed such that it gives a constant 5V DC supply without any fluctuations.
     + Interface the RF modules with RJ45 and using a DIP switch to switch between landline and wireless communication.

Challenging part of landline communication was implementing LISA algorithm in software program. For the wireless communication, challenging part was to understand payload format at receiver, extracting the data from payload and hardware design for RF module to achieve wireless communication functionality in our project system.

# Problem Formulation and Design

1. **LISA Algorithm: hardware and software 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 the wireless Communication between two boards were the main sections of the project.

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 functionalities 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:

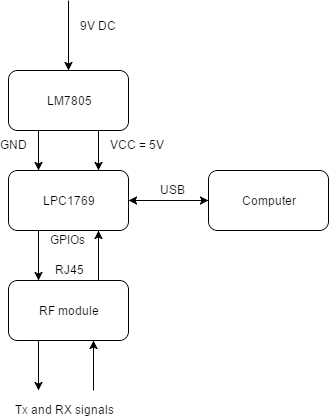


Fig.3: System Design Block

We Implemented software programming on Embedded C platform using MCUXpresso IDE.

Program design flow is as follows:

* Predefined sync field, payload and arbitrary data fills buffer of 1K.
* Node-I as transmitter sends data at an oversampling rate. The transmitter uses a timer interrupt to achieve the oversampling rate.
* Node-j as receiver receives data at a normal sampling rate (oversampling rate/10) using timer interrupt.
* User is asked for required confidence level
* Start of payload is identified as per given confidence level and using predefined sync field in bit stream.
* Message is extracted from the payload based on the confidence level and receiver gets original message.
* The payload size is kept more than twice the size of the sync field.

## 2.3 System Layout

The components are connected as shown in the diagram below.

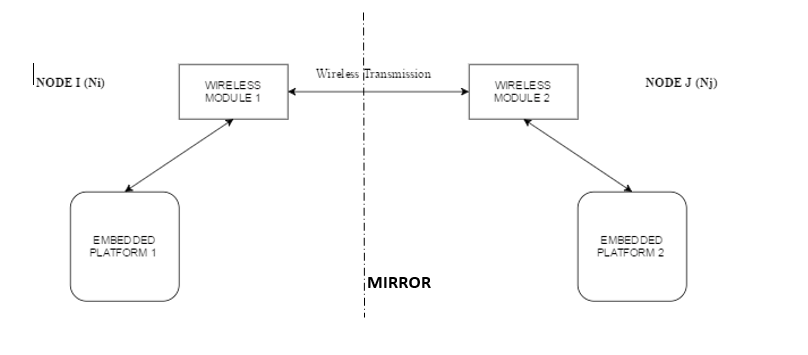


Figure 4 Project Diagram (Generic)

### The pins of the LPC1769 used in landline testing:

|  |  |  |
| --- | --- | --- |
| Pin No. | Functionality | Notes |
| J2-1 | GND | Ground |
| J2-28 | PWR (4.5-5.5  VDC) | Power-In of LPC  1769 |
| P0.2  (J2-21) | GPP P0.2 port Input/Output | GPIO Pin used as Input during RF  communication |
| P0.3  (J2-22) | GPP P0.3 port Input/Output | LPC GPIO Pin used  as Output during RF communication |

Table 1 – Connectivity

### The pin connections needed to achieve wireless communication are as follows:

### Pin connection of LPC Node to RF Transmitter MX-FS-03V:

### Image result for mx-fs-03v rx code

|  |  |
| --- | --- |
| Description | MX-FS-03V RF Transmitter |
| PWR Vcc (5VDC) | External power  Supply (Pin 2 VCC) |
| Data Pin | LPC1769 GPP P0.2 |
| GND | Common GND to power supply and RJ45 (Pin1- GND) |

Table 2 – Pin table of RF Transmitter MX-FS-03V to LPC1769

### Pin connection of RF Receiver MX-05V to LPC NODE:

### Image result for mx-05v rx code

|  |  |  |
| --- | --- | --- |
| Pin Nos. | Description | Notes |
| 1 | GND | Common GND  LPC Pin J2-1 |
| 2 and 3 | Data Pin | LPC 1769 GPP P0.3 |
| 4 | PWR (5VDC) | External Power supply |

Table 3 – Pin table of RF Receiver MX-05V to LPC1769

## Switching between wireless and landline communication

* We shall be using a DIP Switch for toggling between landline and wireless communication.
* The RX and TX of the RJ45 from one embedded board(Ni) will be connected to either DATA pins of RF modules or TX and RX of the RJ45 connected to the other embedded board(NJ)
* The block diagram of the DIP Switch connection is shown as below:

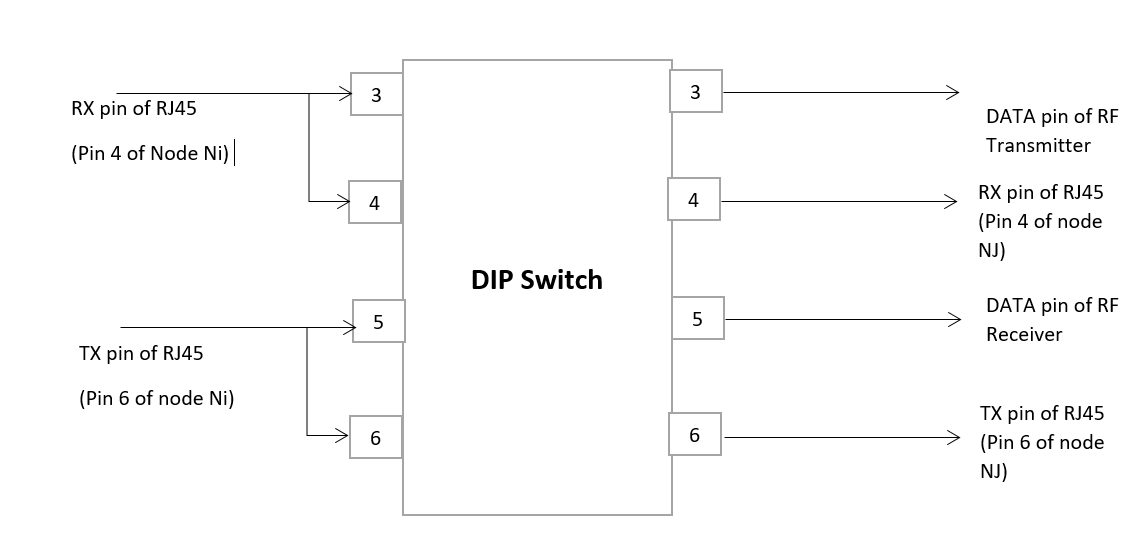


Fig.4: DIP Switch connections

# Implementation

Below section explains the implementation of the project in hardware software perspective. It is divided into 2 sections: Hardware design and software implementation.

## Hardware Design

* The Power circuit comprising of LM7805 produces regulated output which is given to the LPC1769 Development board.
* Pin 01(J2-1) of the LPC1769 board is Ground while Pin 28(J2-28) is Vcc. Pin 2 of Port 0(J2-21) is set as Input while Pin 3 of port 0(J2-22) is set to Output through software.
* The push button acts as a debugger and gives Vcc when pushed through the SPDT switch. The LED corresponding to the switch will glow when it is toggled.
* RJ45 connector is used to connect the two boards– LPC board and RF board. RF Transmitter and receiver devices are given 5V supply from external power supply.
* One LED is placed on each board to debug the receiver and transmitter and GPIO pins (P0.2 and P0.3).
* DATA pins of the RF modules are connected to DIP Switch. LEDs are placed for RF modules for both RX and TX.
* Following is the table for Pin Connections between LPC1769, RF module and RJ45 connector:

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

Table 4 – Pin Connections

## RF Board

The connections between LPC1769 Board and RF Modules can be shown as below.

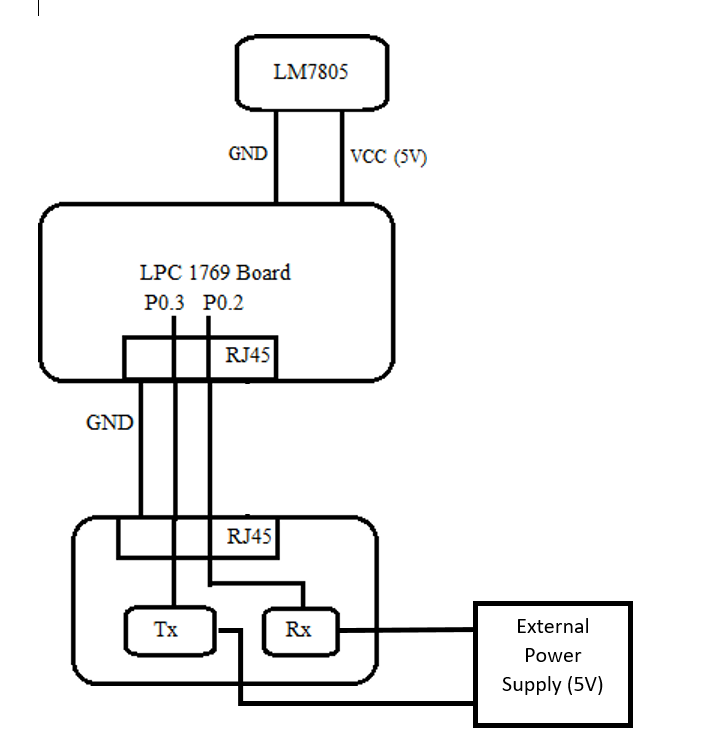


Figure 5. LPC-RF Board Connection Schematic

Power to this circuit is received from the external power supply by Hiletgo. Below is the snapshot of the same. This power is given to the RF Receiver and RF Transmitter modules to power them up.



## Bill of materials

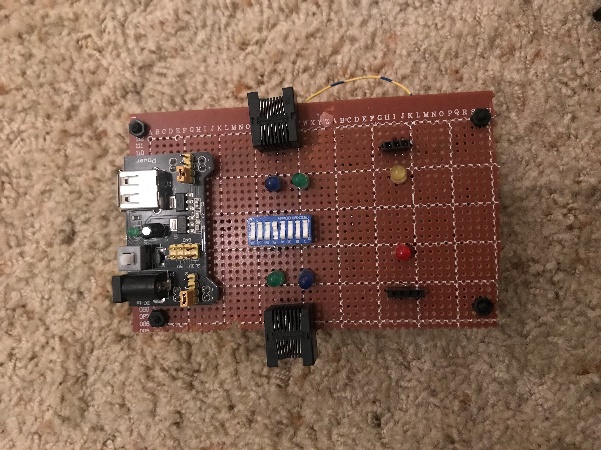
|  |  |  |
| --- | --- | --- |
| Sr  no | Name and Description | Quantity |
| 1 | LPC Xpresso Module (ARM  Cortex – M3) | 1 |
| 2 | LM7805 Power Regulator | 1 |
| 3 | Printed Circuit Board | 2 |
| 4 | DC Power Supply 9V 1.67A | 1 |
| 5 | Red LED | 1 |
| 6 | 10uF Capacitor | 1 |
| 7 | SPDT switch | 3 |
| 9 | 330 Ohms Resistors | 5 |
| 12 | Stand-offs | 10 |
| 13 | 8p8c RJ-45 Connectors | 2 |
| 14 | Cat5e Cables | 4 |

Table 5. Bill of Materials

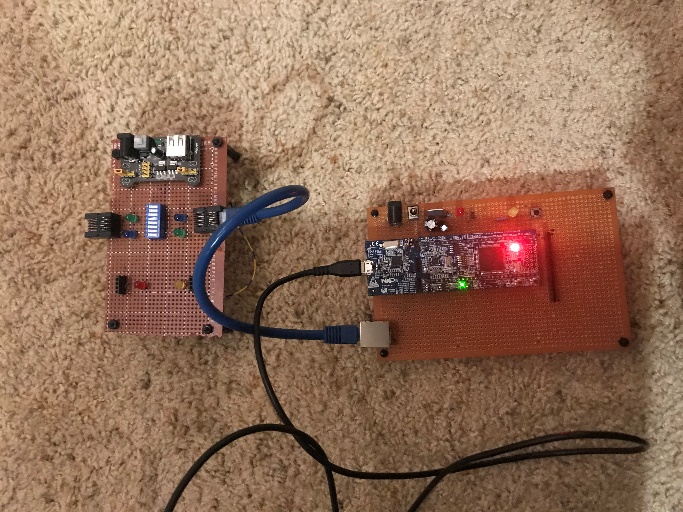
|  |  |  |
| --- | --- | --- |
| Pin Nos. | Description | Notes |
| J1 | RJ-45, I/F as  Landline and wireless testing to Nj(8p8c) | GND, Tx, Rx (2pcs) |
| RF1 | Receiver Rx (RF  Module) | RF Receiver module |
| RF2 | Transmitter Tx  (RF Module) | RF Transmitter  Module |
| D1 | Red LEDs for  Output testing (3 pcs) | LED to indicate transmitted data |
| D2 | Blue LEDs for Input testing (3 pcs) | LED to indicate received data |
| SW2 | Landline vs. RF | Toggle between  wired and wireless communication |

Table 6 - Bill of Material for RF Board

RF Board:



RF Module with Project board:



* 1. **Software Design**

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

## Pseudo Code

**Step 1:** Declare all variables and functions.

**Step 2:** Get confidence level from user for LISA algorithm.

**Step 3:** Initialize the sync field and get the payload that the user wants to transmit. Initialize the timer interrupt at transmitter to transmit at oversampling rate.

**Step 4:** Ask the user if he wants to operate as a transmitter or as a receiver initially. Pressing 1 will lead to transmission and 2 for reception.

**Step 5:** 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 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.

## Flowchart

* + - 1. **LISA Algorithm for Landline and Wireless communication**

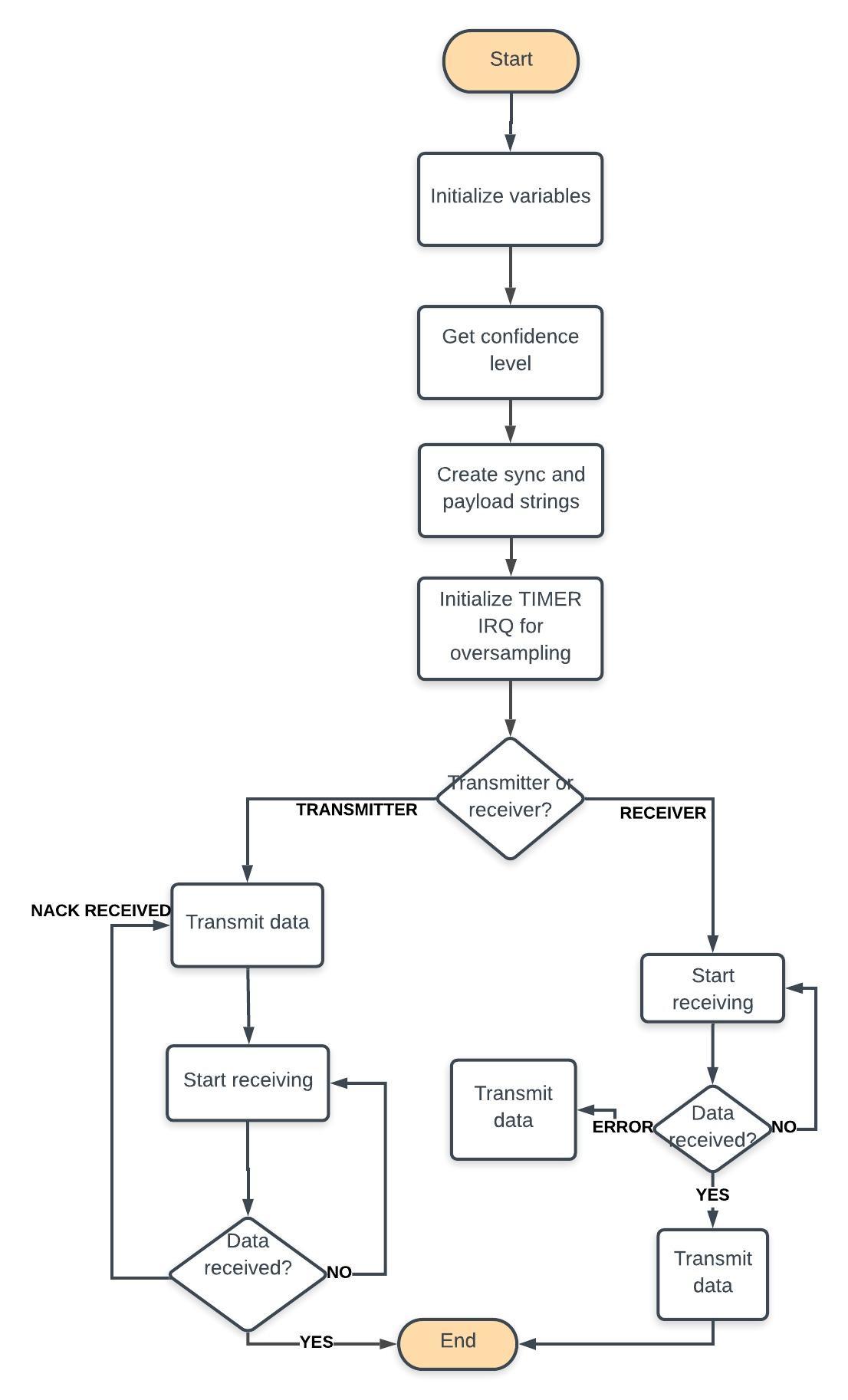
****

Fig. 6 LISA Algorithm

### 

# Testing and Verification

We carried out thorough testing, first on small level and then adding each functionality in our code at every iteration and verified it with valid tests on received and transmitted data. Then we tested across whole system both on Hardware and Software side. We checked Connectivity on each pins on circuit board inputs and outputs. We debug the error code and resolved bugs before running software program.

# Conclusion

In summary, we established wireless communication using between two LPC nodes interfaced with LISA algorithm with RF module. We averaged the data stream from the transmitter end to receiver end with the help of oversampling at the transmitter end. The LISA algorithm was implemented to develop synchronized communication between two wireless embedded systems. We tested the landline communication between 2 LPC modules and 2 RF boards and the data were correctly received at the receiver end. We also switched from landline to wireless communication and we achieved the same results at 1Kbps data rate.

# Acknowledgement

I express my gratitude to Dr. Harry Li for continuous guidance provided for this project, whenever required. Blend of theoretical as well as practical knowledge I got in lectures helped me to understand and perform this project with thorough understanding and fulfilment.

# References

[1] H. Li, “Author Guidelines for CMPE 245 Project Report”, *Lecture Notes of CMPE 245*, Computer Engineering Department, College of Engineering, San Jose State University, 2017.

[2] LPCXpresso1769 Datasheet Datasheet<http://www.nxp.com/documents/data_sheet/LPC1> 769\_ 68\_67\_66\_65\_64\_63.pdf

[3] LM7805 Voltage Regulator Datasheet https://[www.sparkfun.com/datasheets/Components/LM78](http://www.sparkfun.com/datasheets/Components/LM78) 05.pdf

[4] UM10360 user manual

[http://www.nxp.com/documents/user\_manual/UM10360.](http://www.nxp.com/documents/user_manual/UM10360) pdf

[5] Receiver and Transmitter product specifications on vendor site: <https://www.ebay.com/i/382919081134?chn=ps&norover=1&mkevt=1&mkrid=711-117182-37290-0&mkcid=2&itemid=382919081134&targetid=503482142324&device=c&mktype=pla&googleloc=9031934&poi=&campaignid=6469750549&mkgroupid=79220335322&rlsatarget=pla-503482142324&abcId=1141176&merchantid=6296724&gclid=EAIaIQobChMIubSWms-f5QIVConICh3Q8QQyEAYYASABEgLTt_D_BwE>

[6] LPCXpresso-LPC1769-CMSIS-DAP https://[www.embeddedartists.com/sites/default/files/docs/](http://www.embeddedartists.com/sites/default/files/docs/) schematics/LPCXpresso1769\_CD\_revD1.pdf

# Appendix

TX Code for LISA: Landline and Wireless

**Timer.c**

/\*

\* @brief Blinky example using timers

\*

\* @note

\* Copyright(C) NXP Semiconductors, 2014

\* All rights reserved.

\*

\* @par

\* Software that is described herein is for illustrative purposes only

\* which provides customers with programming information regarding the

\* LPC products. This software is supplied "AS IS" without any warranties of

\* any kind, and NXP Semiconductors and its licensor disclaim any and

\* all warranties, express or implied, including all implied warranties of

\* merchantability, fitness for a particular purpose and non-infringement of

\* intellectual property rights. NXP Semiconductors assumes no responsibility

\* or liability for the use of the software, conveys no license or rights under any

\* patent, copyright, mask work right, or any other intellectual property rights in

\* or to any products. NXP Semiconductors reserves the right to make changes

\* in the software without notification. NXP Semiconductors also makes no

\* representation or warranty that such application will be suitable for the

\* specified use without further testing or modification.

\*

\* @par

\* Permission to use, copy, modify, and distribute this software and its

\* documentation is hereby granted, under NXP Semiconductors' and its

\* licensor's relevant copyrights in the software, without fee, provided that it

\* is used in conjunction with NXP Semiconductors microcontrollers. This

\* copyright, permission, and disclaimer notice must appear in all copies of

\* this code.

\*/

#include <cr\_section\_macros.h>

#include "board.h"

#include <stdio.h>

#include <math.h>

#include "syncField.h"

#include "myLisa.h"

#include <string.h>

#include <stdbool.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Private types/enumerations/variables

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define TICKRATE\_HZ1 (11) /\* 11 ticks per second \*/

#define DATA\_TX\_RX\_PORT 0

#define TX\_PIN 2

#define RX\_PIN 3

#define GREEN\_LED\_PORT 3

#define GREEN\_LED\_PIN 25

char sync\_field[SYNC\_FIELD\_BYTES] = {0};

char data\_sync\_bits[SYNC\_BITS] = {0};

char data\_in\_bits[1024] = {0};

char tx\_in\_bytes[128] = {0};

char tx\_in\_bits[1024] = {0};

int tx\_size\_in\_bytes = 0;

int tx\_size\_in\_bits = 0;

unsigned int oversampling\_count = 0;

unsigned int bit\_count = 0;

int sum = 0;

bool process\_data = false;

static int flag, sampling\_rate=100, oversampling\_rate = 10;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Public types/enumerations/variables

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int tx\_bit\_index = 0;

int start\_tx = 0;

bool start\_bit = false;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Private functions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Public functions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*

\* @brief Handle interrupt from 32-bit timer

\* @return Nothing

\*/

static void create\_frame(void)

{

tx\_in\_bytes[0] = 0xFF;

strncpy(tx\_in\_bytes + 1, lisaSyncField, SYNC\_FIELD\_BYTES); // +1 i.e from location tx\_in\_bytes[1]

strncpy(tx\_in\_bytes + 33, &source\_tag, 1); // +32 (offset of sync field)

strncpy(tx\_in\_bytes + 34, &destination\_tag, 1); // +1

strncpy(tx\_in\_bytes + 35, payload\_data, strlen(payload\_data)); // +1

strncpy(tx\_in\_bytes + 35 + strlen(payload\_data), end\_of\_payload, 1); // +strlen(payload\_data)

tx\_size\_in\_bytes = 1 + SYNC\_FIELD\_BYTES + 1 + 1 + strlen(payload\_data) + 1;

tx\_size\_in\_bits = tx\_size\_in\_bytes \* 8;

}

void TIMER0\_IRQHandler(void)

{

if (Chip\_TIMER\_MatchPending(LPC\_TIMER0, 1)) {

#if 1

if (start\_tx == 1) {

if(tx\_bit\_index < tx\_size\_in\_bits)

{

if(tx\_in\_bits[tx\_bit\_index] == '1')

{

LPC\_GPIO3->SET = (1<<GREEN\_LED\_PIN);

LPC\_GPIO->SET = (1<<TX\_PIN);

}

else

{

LPC\_GPIO3->CLR = (1<<GREEN\_LED\_PIN);

LPC\_GPIO->CLR = (1<<TX\_PIN);

}

tx\_bit\_index += 1;

}

else {

tx\_bit\_index = 0;

start\_tx = 0;

flag = 0;

LPC\_GPIO->CLR = (1<<TX\_PIN);

}

}

#endif

Chip\_TIMER\_ClearMatch(LPC\_TIMER0, 1);

}

}

/\*\*

\* @brief main routine for blinky example

\* @return Function should not exit.

\*/

int main(void)

{

uint32\_t timerFreq;

/\* Generic Initialization \*/

SystemCoreClockUpdate();

Board\_Init();

strncpy(sync\_field, lisaSyncField, SYNC\_FIELD\_BYTES);

convert\_bytes\_to\_bits(sync\_field, SYNC\_FIELD\_BYTES, data\_sync\_bits);

create\_frame();

convert\_bytes\_to\_bits(tx\_in\_bytes, tx\_size\_in\_bytes, tx\_in\_bits);

/\* Enable timer 1 clock \*/

Chip\_TIMER\_Init(LPC\_TIMER0);

/\* Timer rate is system clock rate \*/

timerFreq = Chip\_Clock\_GetSystemClockRate(); // 96Mhz

LPC\_GPIO3->DIR |= (1 << GREEN\_LED\_PIN);

LPC\_GPIO->DIR |= (1 << TX\_PIN); // Set as output

LPC\_GPIO->DIR &= ~(1 << RX\_PIN); // Set as input

/\* 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 / (TICKRATE\_HZ1 \* sampling\_rate \* oversampling\_rate)));

Chip\_TIMER\_ResetOnMatchEnable(LPC\_TIMER0, 1);

Chip\_TIMER\_Enable(LPC\_TIMER0);

/\* Enable timer interrupt \*/

NVIC\_ClearPendingIRQ(TIMER0\_IRQn);

NVIC\_EnableIRQ(TIMER0\_IRQn);

/\* LEDs toggle in interrupt handlers \*/

while (1) {

if (flag == 0) {

printf("Enter 1 to transmit\n");

scanf("%d", &flag);

if (flag==1) break;

}

\_\_WFI();

}

start\_tx = 1;

return 0;

}

**myLisa.c:**

#include "myLisa.h"

#include <stdlib.h>

#include <math.h>

#include "syncField.h"

#include <stdio.h>

#include <string.h>

#include <time.h>

static const char alphabets[] =

{

'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h',

'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p',

'q', 'r', 's', 't', 'u', 'v', 'w', 'x',

'y', 'z'

};

static const int powerOfTwo[] =

{

1, 2, 4, 8, 16, 32, 64

};

/\*\*

\* @brief getSetBits

\* @param c

\* @return

\*/

int get\_set\_bits(char c) {

int result = 0;

for (int i = 0; i < 8; i++)

{

result += (c >> i) & 1;

}

return result;

}

void print\_array(const char \*array, int start, int end, const char\* format) {

for (int i = start; i < end; i++) {

printf(format, array[i]);

}

printf("\n");

}

void corrupt\_sync\_field(int corrupt\_percentage, char \*sync\_field, int sync\_field\_size) {

int to\_Corrupt = (int)(ceil(((double)(corrupt\_percentage) \* sync\_field\_size \* 8.0)/100));

srand(time(NULL));

char temp[SYNC\_FIELD\_BYTES] = {0};

strncpy(temp, sync\_field, sync\_field\_size);

while (to\_Corrupt) {

size\_t byte\_index= (size\_t)((size\_t)(rand())%sync\_field\_size);

int bit\_Index= rand()%8;

if ((sync\_field[byte\_index] ^ (1 << bit\_Index)) != temp[byte\_index])

{

sync\_field[byte\_index] ^= (1 << bit\_Index);

to\_Corrupt--;

}

}

}

void convert\_bytes\_to\_bits(const char \*bytes, int size, char \*bits) {

int bit\_pos = 0;

for (int byte = 0; byte < size; byte++) {

for (int bit = 7; bit >= 0; bit--) {

bits[bit\_pos] = (bytes[byte] & (1 << bit)) ? '1' : '0';

bit\_pos++;

}

}

}

/\*\*

\* @brief convertBitsToByte: This function shall convert binary hex value to a byte of type char

\* @param bits: Input string of 8 chars only i.e 01010000 -> Shall return char 'P'

\* @return: Returns char corresponding to the input

\*/

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--];

}

return (char)(byte);

}

void write\_text\_file(const char \*filename, const char\* file\_mode, const char \*data\_to\_write, int number\_of\_bytes\_to\_write) {

FILE \*fp;

fp = fopen(filename, file\_mode);

if (fp == NULL) {

printf("Error writing file\n");

exit(1);

}

fwrite(data\_to\_write, number\_of\_bytes\_to\_write, 1, fp);

fclose(fp);

}

void read\_text\_file(const char \*filename, char \*data\_to\_read) {

FILE \*fp;

fp = fopen(filename, "r");

if (fp == NULL) {

printf("Error reading file\n");

exit(1);

}

fseek(fp, 0l, SEEK\_END);

int bytes\_to\_read = (int)ftell(fp);

fseek(fp, 0l, SEEK\_SET);

fread(data\_to\_read, bytes\_to\_read, 1, fp);

printf("File size: %d\n", bytes\_to\_read);

fclose(fp);

}

void generate\_junk\_data(int junk\_size, char \*junk\_data) {

int i = 0;

while ((junk\_size--) && (junk\_size != 0)) {

junk\_data[i++] = alphabets[rand()%26];

}

}

/\*\*

\* @brief getCorruptLevelFromTestData

\* @param start

\* @param end

\* @param data

\* @param sync

\* @return

\*/

int get\_corruption\_level(int start, int end, const char \*data, const char \*sync) {

int corruption\_level = 0;

for (int i = start, j = 0; i < end; i++, j++) {

char c = (data[i] ^ sync[j]);

if (c != 0){

corruption\_level += get\_set\_bits(c);

}

}

return corruption\_level;

}

int get\_confidence\_level(int start, int end, const char \*data, const char\* sync) {

#define BIT\_MODE 0

#if BIT\_MODE

int confidence\_level = 0;

for (int i = start, j = 0; i < end; i++, j++) {

char c = data[i] ^ sync[j];

if (c == 0) {

confidence\_level++;

}

}

return confidence\_level;

#else

int confidence\_level = 0;

int number\_of\_matched\_bits = 0;

for (int i = 0, bit\_num = start; bit\_num < end; i++, bit\_num++) {

// Iterate over each bit within a byte

if ((data[bit\_num] ^ sync[i]) == 0) {

number\_of\_matched\_bits++;

if (number\_of\_matched\_bits == 8) {

confidence\_level++;

number\_of\_matched\_bits = 0;

}

}

else {

number\_of\_matched\_bits = 0;

}

}

return confidence\_level;

#endif

}

int get\_start\_location\_of\_payload(const char \*data, int data\_size, const char \*lisa\_sync\_field, int sync\_size, int confidence\_level, int corrupt\_level) {

(void)corrupt\_level;

int start\_position\_of\_sync\_field = -1;

int start\_bit\_of\_payload = -1;

int actual\_Confidence = (int)(ceil((double)(confidence\_level \* (sync\_size/8))/100));

int max\_confidence = 0;

for (int j = 0; j < data\_size - sync\_size; j++) {

int level = get\_confidence\_level(j, j+SYNC\_BITS, data, lisa\_sync\_field);

if (max\_confidence < level) {

max\_confidence = level;

}

if (max\_confidence >= actual\_Confidence)

{

start\_position\_of\_sync\_field = j;

start\_bit\_of\_payload = start\_position\_of\_sync\_field + SYNC\_BITS;

break;

}

}

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("Start position of sync field: %d\n", start\_position\_of\_sync\_field);

printf("Actual confidence level: %d\n", actual\_Confidence);

printf("Observed confidence level: %d\n", max\_confidence);

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

return start\_bit\_of\_payload;

}

RX Code for LISA: Landline and Wireless

**Timer.c**

/\*

\* @brief Blinky example using timers

\*

\* @note

\* Copyright(C) NXP Semiconductors, 2014

\* All rights reserved.

\*

\* @par

\* Software that is described herein is for illustrative purposes only

\* which provides customers with programming information regarding the

\* LPC products. This software is supplied "AS IS" without any warranties of

\* any kind, and NXP Semiconductors and its licensor disclaim any and

\* all warranties, express or implied, including all implied warranties of

\* merchantability, fitness for a particular purpose and non-infringement of

\* intellectual property rights. NXP Semiconductors assumes no responsibility

\* or liability for the use of the software, conveys no license or rights under any

\* patent, copyright, mask work right, or any other intellectual property rights in

\* or to any products. NXP Semiconductors reserves the right to make changes

\* in the software without notification. NXP Semiconductors also makes no

\* representation or warranty that such application will be suitable for the

\* specified use without further testing or modification.

\*

\* @par

\* Permission to use, copy, modify, and distribute this software and its

\* documentation is hereby granted, under NXP Semiconductors' and its

\* licensor's relevant copyrights in the software, without fee, provided that it

\* is used in conjunction with NXP Semiconductors microcontrollers. This

\* copyright, permission, and disclaimer notice must appear in all copies of

\* this code.

\*/

#include <cr\_section\_macros.h>

#include "board.h"

#include <stdio.h>

#include <math.h>

#include "syncField.h"

#include "myLisa.h"

#include <string.h>

#include <stdbool.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Private types/enumerations/variables

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define TICKRATE\_HZ1 (11) /\* 11 ticks per second \*/

#define DATA\_TX\_RX\_PORT 0

#define TX\_PIN 2

#define RX\_PIN 3

#define GREEN\_LED\_PORT 3

#define GREEN\_LED\_PIN 25

char sync\_field[SYNC\_FIELD\_BYTES] = {0};

char data\_sync\_bits[SYNC\_BITS] = {0};

char data\_in\_bits[1024] = {0};

char rx\_buff[1024] = {0};

unsigned int oversampling\_count = 0;

unsigned int bit\_count = 0;

int sum = 0;

bool process\_data = false;

static int flag, sampling\_rate=100, oversampling\_rate = 1;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Public types/enumerations/variables

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int tx\_bit\_index = 0;

int start\_tx = 0;

bool start\_bit = false;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Private functions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Public functions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*

\* @brief Handle interrupt from 32-bit timer

\* @return Nothing

\*/

void TIMER0\_IRQHandler(void)

{

if (Chip\_TIMER\_MatchPending(LPC\_TIMER0, 1)) {

#if 1

if (LPC\_GPIO->PIN & (1 << RX\_PIN)) {

start\_bit = true;

}

if (start\_bit == true) {

if (LPC\_GPIO->PIN & (1 << RX\_PIN)) {

sum += 1;

} else {

sum += 0;

}

oversampling\_count += 1;

}

if (oversampling\_count == oversampling\_rate) {

oversampling\_count = 0;

float avg = sum/oversampling\_rate;

if (avg < 0.5) {

rx\_buff[bit\_count] = '0';

} else {

rx\_buff[bit\_count] = '1';

}

bit\_count += 1;

sum = 0;

}

if (bit\_count == 1024) {

bit\_count = 0;

process\_data = true;

}

#endif

Chip\_TIMER\_ClearMatch(LPC\_TIMER0, 1);

}

}

void process\_received\_data(void) {

if (process\_data == true) {

start\_bit = false;

process\_data = false;

strncpy(data\_in\_bits, rx\_buff, sizeof(rx\_buff));

int confidence\_level = 10;

int corrupt\_percentage = 0;

int start\_of\_payload = get\_start\_location\_of\_payload(data\_in\_bits, sizeof(data\_in\_bits), data\_sync\_bits, SYNC\_BITS, confidence\_level, corrupt\_percentage);

printf("Payload Starts at %d\n", start\_of\_payload);

if (start\_of\_payload != -1) {

char temp[8] = {0};

int count = 0;

for (int i = start\_of\_payload; i < sizeof(data\_in\_bits); i++) {

temp[count++] = data\_in\_bits[i];

if (count == 8) {

printf("%c", convert\_8\_bits\_to\_char(temp));

count = 0;

}

}

}

flag = 0;

}

}

/\*\*

\* @brief main routine for blinky example

\* @return Function should not exit.

\*/

int main(void)

{

uint32\_t timerFreq;

/\* Generic Initialization \*/

SystemCoreClockUpdate();

Board\_Init();

strncpy(sync\_field, lisaSyncField, SYNC\_FIELD\_BYTES);

convert\_bytes\_to\_bits(sync\_field, SYNC\_FIELD\_BYTES, data\_sync\_bits);

/\* Enable timer 1 clock \*/

Chip\_TIMER\_Init(LPC\_TIMER0);

/\* Timer rate is system clock rate \*/

timerFreq = Chip\_Clock\_GetSystemClockRate(); // 96Mhz

LPC\_GPIO3->DIR |= (1 << GREEN\_LED\_PIN);

LPC\_GPIO->DIR |= (1 << TX\_PIN); // Set as output

LPC\_GPIO->DIR &= ~(1 << RX\_PIN); // Set as input

/\* 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 / (TICKRATE\_HZ1 \* sampling\_rate \* oversampling\_rate)));

Chip\_TIMER\_ResetOnMatchEnable(LPC\_TIMER0, 1);

Chip\_TIMER\_Enable(LPC\_TIMER0);

/\* Enable timer interrupt \*/

NVIC\_ClearPendingIRQ(TIMER0\_IRQn);

NVIC\_EnableIRQ(TIMER0\_IRQn);

/\* LEDs toggle in interrupt handlers \*/

while (1) {

if (flag == 0) {

printf("Press 2 to Receive\n");

scanf("%d", &flag);

}

if(flag==2) break;

\_\_WFI();

}

process\_received\_data();

return 0;

}

**myLisa.c:**

#include "myLisa.h"

#include <stdlib.h>

#include <math.h>

#include "syncField.h"

#include <stdio.h>

#include <string.h>

#include <time.h>

static const char alphabets[] =

{

'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h',

'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p',

'q', 'r', 's', 't', 'u', 'v', 'w', 'x',

'y', 'z'

};

static const int powerOfTwo[] =

{

1, 2, 4, 8, 16, 32, 64

};

/\*\*

\* @brief getSetBits

\* @param c

\* @return

\*/

int get\_set\_bits(char c) {

int result = 0;

for (int i = 0; i < 8; i++)

{

result += (c >> i) & 1;

}

return result;

}

void print\_array(const char \*array, int start, int end, const char\* format) {

for (int i = start; i < end; i++) {

printf(format, array[i]);

}

printf("\n");

}

void corrupt\_sync\_field(int corrupt\_percentage, char \*sync\_field, int sync\_field\_size) {

int to\_Corrupt = (int)(ceil(((double)(corrupt\_percentage) \* sync\_field\_size \* 8.0)/100));

srand(time(NULL));

char temp[SYNC\_FIELD\_BYTES] = {0};

strncpy(temp, sync\_field, sync\_field\_size);

while (to\_Corrupt) {

size\_t byte\_index= (size\_t)((size\_t)(rand())%sync\_field\_size);

int bit\_Index= rand()%8;

if ((sync\_field[byte\_index] ^ (1 << bit\_Index)) != temp[byte\_index])

{

sync\_field[byte\_index] ^= (1 << bit\_Index);

to\_Corrupt--;

}

}

}

void convert\_bytes\_to\_bits(const char \*bytes, int size, char \*bits) {

int bit\_pos = 0;

for (int byte = 0; byte < size; byte++) {

for (int bit = 7; bit >= 0; bit--) {

bits[bit\_pos] = (bytes[byte] & (1 << bit)) ? '1' : '0';

bit\_pos++;

}

}

}

/\*\*

\* @brief convertBitsToByte: This function shall convert binary hex value to a byte of type char

\* @param bits: Input string of 8 chars only i.e 01010000 -> Shall return char 'P'

\* @return: Returns char corresponding to the input

\*/

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--];

}

return (char)(byte);

}

void write\_text\_file(const char \*filename, const char\* file\_mode, const char \*data\_to\_write, int number\_of\_bytes\_to\_write) {

FILE \*fp;

fp = fopen(filename, file\_mode);

if (fp == NULL) {

printf("Error writing file\n");

exit(1);

}

fwrite(data\_to\_write, number\_of\_bytes\_to\_write, 1, fp);

fclose(fp);

}

void read\_text\_file(const char \*filename, char \*data\_to\_read) {

FILE \*fp;

fp = fopen(filename, "r");

if (fp == NULL) {

printf("Error reading file\n");

exit(1);

}

fseek(fp, 0l, SEEK\_END);

int bytes\_to\_read = (int)ftell(fp);

fseek(fp, 0l, SEEK\_SET);

fread(data\_to\_read, bytes\_to\_read, 1, fp);

printf("File size: %d\n", bytes\_to\_read);

fclose(fp);

}

void generate\_junk\_data(int junk\_size, char \*junk\_data) {

int i = 0;

while ((junk\_size--) && (junk\_size != 0)) {

junk\_data[i++] = alphabets[rand()%26];

}

}

/\*\*

\* @brief getCorruptLevelFromTestData

\* @param start

\* @param end

\* @param data

\* @param sync

\* @return

\*/

int get\_corruption\_level(int start, int end, const char \*data, const char \*sync) {

int corruption\_level = 0;

for (int i = start, j = 0; i < end; i++, j++) {

char c = (data[i] ^ sync[j]);

if (c != 0){

corruption\_level += get\_set\_bits(c);

}

}

return corruption\_level;

}

int get\_confidence\_level(int start, int end, const char \*data, const char\* sync) {

#define BIT\_MODE 0

#if BIT\_MODE

int confidence\_level = 0;

for (int i = start, j = 0; i < end; i++, j++) {

char c = data[i] ^ sync[j];

if (c == 0) {

confidence\_level++;

}

}

return confidence\_level;

#else

int confidence\_level = 0;

int number\_of\_matched\_bits = 0;

for (int i = 0, bit\_num = start; bit\_num < end; i++, bit\_num++) {

// Iterate over each bit within a byte

if ((data[bit\_num] ^ sync[i]) == 0) {

number\_of\_matched\_bits++;

if (number\_of\_matched\_bits == 8) {

confidence\_level++;

number\_of\_matched\_bits = 0;

}

}

else {

number\_of\_matched\_bits = 0;

}

}

return confidence\_level;

#endif

}

int get\_start\_location\_of\_payload(const char \*data, int data\_size, const char \*lisa\_sync\_field, int sync\_size, int confidence\_level, int corrupt\_level) {

(void)corrupt\_level;

int start\_position\_of\_sync\_field = -1;

int start\_bit\_of\_payload = -1;

int actual\_Confidence = (int)(ceil((double)(confidence\_level \* (sync\_size/8))/100));

int max\_confidence = 0;

for (int j = 0; j < data\_size - sync\_size; j++) {

int level = get\_confidence\_level(j, j+SYNC\_BITS, data, lisa\_sync\_field);

if (max\_confidence < level) {

max\_confidence = level;

}

if (max\_confidence >= actual\_Confidence)

{

start\_position\_of\_sync\_field = j;

start\_bit\_of\_payload = start\_position\_of\_sync\_field + SYNC\_BITS;

break;

}

}

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("Start position of sync field: %d\n", start\_position\_of\_sync\_field);

printf("Actual confidence level: %d\n", actual\_Confidence);

printf("Observed confidence level: %d\n", max\_confidence);

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

return start\_bit\_of\_payload;

}