**Implementation of Cognitive Radio and Wireless Communication using LoRa module**

*Vatsal Makani-013731614*

*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 wireless communication between two nodes using SX 1276, Long Range (LoRa) module interfaced with LPC 1769, 32 bit ARM Cortex-M3 microcontroller. The wireless communication using LoRa module was also done by implementing an RF board with LoRa module interfacing with LPC1769 embedded board with the help of RJ45-Cat5e cable or a ribbon cable. Long Range module provides functionality to adapt to the desired condition by changing parameters which can be achieved by software programming. The wireless transmission and reception can be achieved by RF module at the different bit rates by changing the parameters in the code.*

# Introduction

To test input and output ports of LPC 1769 and before implementing wireless communication using LoRa module, we implemented Linear Invariant Synchronization Algorithm (LISA) in code and basic RF modules 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. This proves the functionality of LPC1769 to control the wireless communication.

Cognitive Radio communication enables user to make system as intelligent as possible by programming required parameters in software. This functionality makes wireless communication more flexible and adaptive to situations. We have used LoRa module which also allows more dynamic wireless communication for longer range on different frequencies varying from 137-1020 MHz It can be programmed in such a way that receiver can inform transmitter to change some parameters in the middle of communication. We interfaced LoRa module with LPC 1769 by using SPI interfacing.

# 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. **Hardware Design**
   * + Design the Project board with power circuit to provide continues 5V supply to the platform and debugging capabilities by providing 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. **LoRa Module:**

* Design a prototype board circuit to mount the LoRa module and provide the continuous 3.3V supply to the platform.
* Design the Project board with debugging capabilities wherein testing is possible with passive components.
* Checking the connections and making sure that hardware connections are not shorted.
* Powering up the LoRa module and checking the version number as 12 which ensure proper hardware connections.

1. **Cognitive Radio:**

O The objectives of Cognitive Radio includes:

* To establish adaptive and robust wireless communication between two nodes Ni and Nj.
* To make communication more intelligent by pre-programming different transmitter and receiver parameters such as Coding Rate, Bandwidth, Frequency Hopping, etc.
* The challenging part of Cognitive radio was to understand different means to achieve cognitive functionality in our project system and implementing several functionalities such as frequency hopping in software program.

Challenging part of LoRa communication was ensuring the hardware connectivity and checking the version number 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

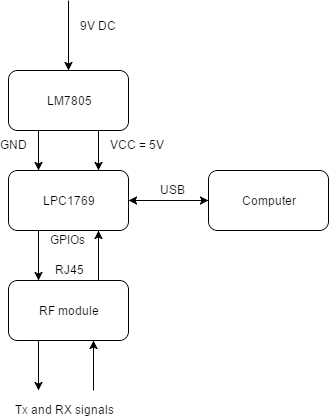
1. **Cognitive Radio: hardware and software design**

As a Problem Statement of Cognitive Radio, we were given certain functionalities or parameters enumerated as Spreading Factor, Transmitter Power, Channel Filter, etc. We were supposed to make communication as smart as possible by modifying Registers and modes of different LoRa (Long Range) module Registers.

Initially to ensure communication of LoRa module with microcontroller, we did Handshaking, which includes SPI interfacing with controller and then we performed Handshaking with other LoRa module having same mirror circuit on node-j side. Getting correct version of LoRa module on both sides ensures that handshaking between two nodes is done. For this, both modules should be operating on same frequency.

Changing different functionalities includes changing predefined modes of registers in given module through software programming. Long Range module operates on two different modes namely LoRa mode and FSK/OOK mode, each containing different set of registers and modes. Accessing those register may require changing mode to one of this mode.

Power Source for system requires 5V steady voltage. We implemented voltage regulator down converting from 9V to 5V. Microcontroller and RF board containing the LoRa module are connected using a ribbon cable with 16-pin 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:



Ribbon cable

Figure 1 System Design Block

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

Program design flow is as follows:

* The user has an option to either select whether the LoRa module shall be in receiving mode or transmitting mode.
* The frequency is selected by the user at the input and few parameters like power, bitrate, spread factor etc. is set
* Node-I acts as transmitter sends data at a frequency selected by the user input and sets the corresponding bit rate and frequency in registers.
* API created the packet to append the data to be transmitted at the input and sent across the bandwidth.
* Node-j as receiver receives data at a frequency which is set same as the transmitter in order to avoid interference

## 2.3 System Layout

The components are connected as shown in the diagram below.

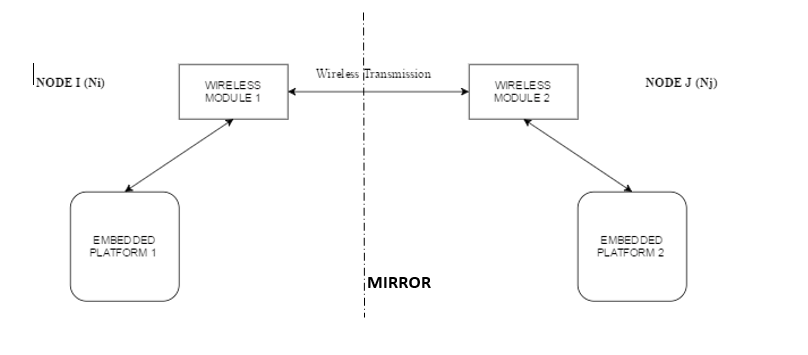


Figure 2 Project Diagram (Generic)

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

|  |  |  |
| --- | --- | --- |
| Pin No. | Functionality | Notes |
| J2-1 | GND | Ground |
| J2-28 | PWR (3.3  VDC) | Power-In of LPC  1769 |
| P0.2  (J2-21) | GPP P0.2 port Input/Output | GPIO Pin used as Input during RF  communication |
| J2-5 | SPI MOSI pin | LPC MOSI Pin used  as master out and slave in for RF communication |
| J2-6 | SPI MISO pin | LPC MISO Pin used  as master in and slave out for RF communication |
| J2-7 | SPI SCK1 pin | LPC SCK Pin used  as master clock for RF communication |
| J2-8 | SPI SSEL1 pin | LPC SSEL Pin used  as slave select for RF communication |

Table 1 – Connectivity

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

### Below is the image of the LoRa module SX1276 purchased from the below link:

### <https://www.digikey.com/product-detail/en/semtech-corporation/SX1276RF1IAS/SX1276RF1IAS-ND/4490401>

### Image result for sx1276

Figure 3 LoRa module SX1276RF1IAS

|  |  |
| --- | --- |
| LPC 1769 | LoRa module |
| J2-28 (3.3VDC) | VR\_ANA (Pin 2) |
| J2-28 (3.3VDC) | PFO\_HF (Pin 22) |
| J2-28 (3.3VDC) | VDD\_FEN (Pin 34) |
| J2-1  (GND) | VR\_DIG (Pin 4) |
| J2-1  (GND) | VBAT2 (Pin 24) |
| J2-1  (GND) | GND (Pin 32) |
| J2 -5  (MOSI) | VBAT (Pin 3) |
| J2-6  (MISO) | DIO0 (Pin 8) |
| J2-7  (SCK1) | RFI\_LF (Pin 1) |
| J2-8  (SSEL1) | NRESET (Pin 7) |
| J2-21  (P0.2 GPIO) | DIO2 (Pin 10) |

Table 2 – Pin table of LoRa module SX1276 to LPC1769

## 3 Implementation

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

## RF Board

* 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.
* 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.
* Ribbon cable with 16 pin connectors is used to connect the two boards– LPC board and RF board. LoRa module SX1276 are given 5V supply from external power supply.
* SPI lines MOSI MISO SCK and SSEL of LPC1769 are communicating to LoRA and GPIO pin P0.2 is also connected to LoRA module.

## RF Board

The connections between LPC1769 Board and RF Module containing LoRa module can be shown as below.

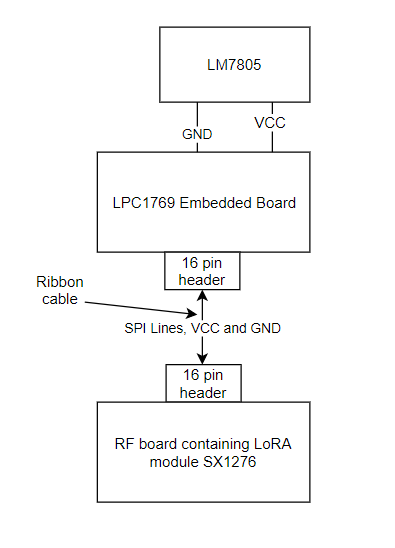


Figure 4 LPC-RF Board Connection Schematic

Power to this circuit is received from the external power supply from the embedded board. This power is given to the LoRa module 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 | 8 pin 2 row male header | 1 |
| 14 | Ribbon cable with 16 pin connector | 1 |

Table 5. Bill of Materials

|  |  |  |
| --- | --- | --- |
| PN | Description | Notes |
| SX1276 | LoRa module |  |
| - | 8 pin 2 row male header | Headers to interface the ribbon cable |
| - | Female headers | Headers to interface the LoRa module |

Table 6 - Bill of Material for RF Board

RF Board:

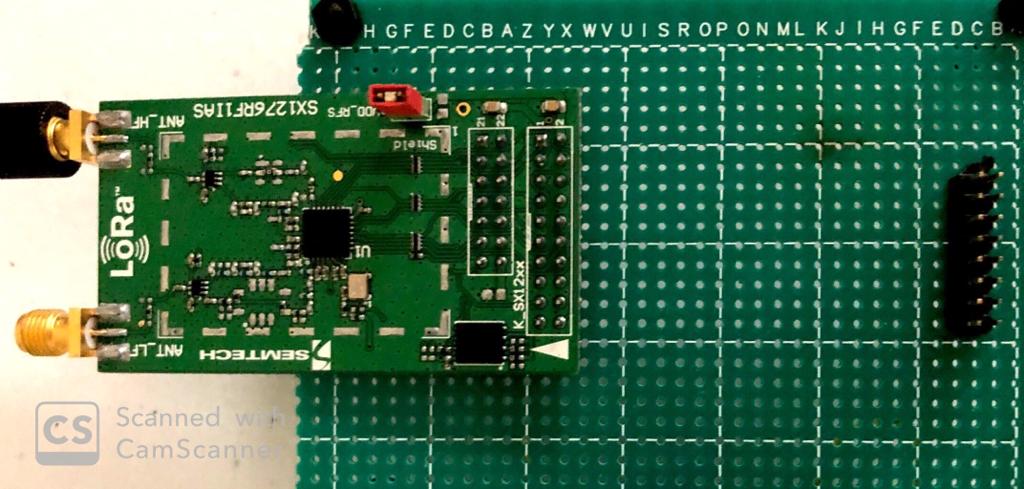


Figure 5 LoRa module in RF board

RF Module with Project board:

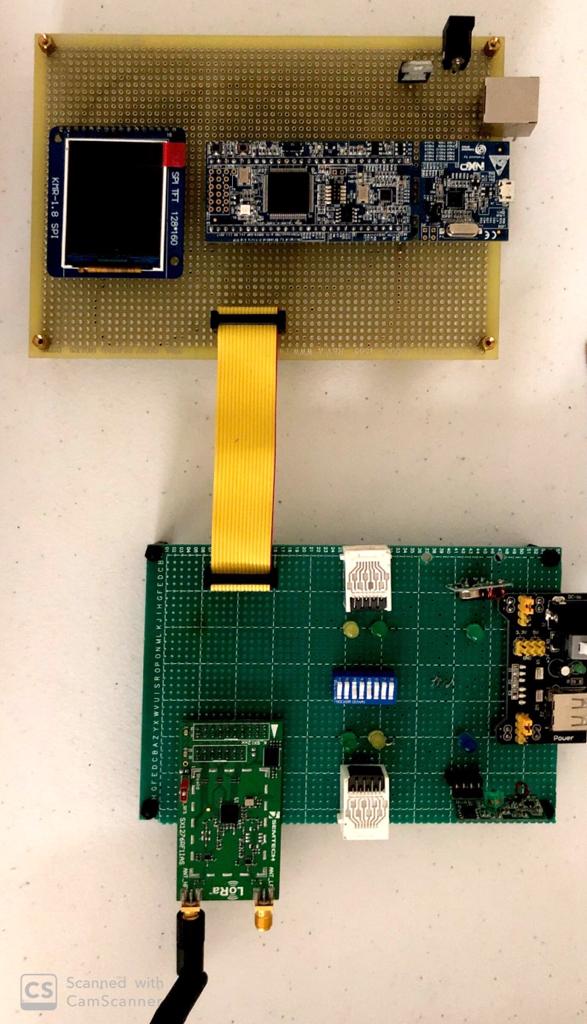


Figure 6 LoRa wireless communication setup

* 1. **Software Design**

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

## Pseudo Code

**Step 1:** Do the handshaking between microcontroller and

LoRa module.

**Step 2**: Perform Handshaking between two LoRa modules

interfaced with controllers on node-i and node-j side.

**Step3**: Change the values of different Registers and

modes in software using defined registers on transmitter

side. For that, change operation mode to LoRa or FSK as

per register mode.

**Step4**: perform similar or related changes on code on

receiver side as well to get correct data at receiver side.

**Step5**: send Acknowledgement in form of signal to

transmitter to inform transmitter whether any

functionality needs to be changed or not,

**Step6**: received data after acknowledgement is sent to

know whether changed parameter is operating or not.

**Step7**: keep changing functionality in code untile desired

## performance is achieved by system.

## Flowchart for Cognitive Radio showing Wireless communication

****

# 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.> pdf

[5] SemTech SX 1276 module Data sheet

https://www.semtech.com/images/datasheet/sx1276\_77\_7

8\_79.pdf

[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

Code for LoRa wireless communication

**main.c**

/\*

===============================================================================

Name : RF\_Handshaking.c

Description : RF and PWM code for LPC1769

CTI One Corporation released for Dr. Harry Li for CMPE 245 Class use ONLY!

===============================================================================

\*/

#ifdef \_\_USE\_CMSIS

#include "LPC17xx.h"

#endif

#include <cr\_section\_macros.h>

#include <stdio.h>

#include <stdbool.h>

#include "LoRa.h"

#include"common.h"

#define RF\_Receive 0

#define TransmittACk 0

#define ack\_start\_stop 0

int rfInit(void);

char receiveData=0;

int packetSize;

int frequency;

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

\* @brief wait for ms amount of milliseconds

\* @param ms : Time to wait in milliseconds

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

static void delay\_ms(unsigned int ms)

{

unsigned int i,j;

for(i=0;i<ms;i++)

for(j=0;j<50000;j++);

}

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

\* @brief wait for delayInMs amount of milliseconds

\* @param delayInMs : Time to wait in milliseconds

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

static void delay(uint32\_t delayInMs)

{

LPC\_TIM0->TCR = 0x02; /\* reset timer \*/

LPC\_TIM0->PR = 0x00; /\* set prescaler to zero \*/

LPC\_TIM0->MR0 = delayInMs \* (9000000 / 1000-1);

LPC\_TIM0->IR = 0xff; /\* reset all interrrupts \*/

LPC\_TIM0->MCR = 0x04; /\* stop timer on match \*/

LPC\_TIM0->TCR = 0x01; /\* start timer \*/

/\* wait until delay time has elapsed \*/

while (LPC\_TIM0->TCR & 0x01);

}

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

\* main : Main program entry

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

int main(void)

{

uint8\_t frequency\_select;

printf("System clock is %d\n",SystemCoreClock);

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

printf("1. 724MHz \n");

printf("2. 750MHz \n");

printf("3. 790MHz \n");

printf("4. 800MHz \n");

printf("5. 845MHz \n");

printf("6. 850MHz \n");

printf("7. 868MHz \n");

printf("8. 910MHz \n");

printf("9. 1020MHz \n");

printf("10. 1040MHz \n");

printf("11.\tCustom \n");

printf("\nSelect the frequency:");

scanf("%d",&frequency\_select);

switch(frequency\_select)

{

case 1: frequency = 724000000; break;

case 2: frequency = 750000000; break;

case 3: frequency = 790000000; break;

case 4: frequency = 800000000; break;

case 5: frequency = 845000000; break;

case 6: frequency = 850000000; break;

case 7: frequency = 868000000; break;

case 8: frequency = 910000000; break;

case 9: frequency = 1020000000; break;

case 10:frequency = 1040000000; break;

case 11:printf("Enter your custom frequency in MHz:");

scanf("%d\n",&frequency);

frequency \*= 1000000;

break;

default: printf("Invalid Selection\n");

break;

}

LoRabegin(frequency);

int counter =0;

//timer\_initialise();

/\* Start the tasks running. \*/

//vTaskStartScheduler();

/\* If all is well we will never reach here as the scheduler will now be

running. If we do reach here then it is likely that there was insufficient

heap available for the idle task to be created. \*/

#if RF\_Receive

while(1)

{

packetSize = parsePacket(0);

if (packetSize)

{

counter = 0;

//NVIC\_EnableIRQ(TIMER0\_IRQn);

//received a packet

// printf("Received packet '");

// read packet

while (available())

{

counter = 0;

receiveData = read();

//printf("%c",receiveData);

if(receiveData == 'A')

{

printf("Received packet '");

printf("%c",receiveData);

printf("' with RSSI ");

printf("%d\n",packetRssi());

}

else if(receiveData == 'B')

{

printf("Received packet '");

printf("%c",receiveData);

printf("' with RSSI ");

printf("%d\n",packetRssi());

}

else if(receiveData == 'C')

{

printf("Received packet '");

printf("%c",receiveData);

printf("' with RSSI ");

printf("%d\n",packetRssi());

}

else if(receiveData == 'D')

{

printf("Received packet '");

printf("%c",receiveData);

printf("' with RSSI ");

printf("%d\n",packetRssi());

}

else if(receiveData == 'E')

{

printf("Received packet '");

printf("%c",receiveData);

printf("' with RSSI ");

printf("%d\n",packetRssi());

}

else if(receiveData == 'F')

{

printf("Received packet '");

printf("%c",receiveData);

printf("' with RSSI ");

printf("%d\n",packetRssi());

}

}

// print RSSI of packet

// printf("Received packet '");

// printf("%c",receiveData);

// printf("' with RSSI ");

// printf("%d\n",packetRssi());

}

}

#endif

#if TransmittACk

const char buffer[] = "Data from LPC1769";

char Acknowledgement;

Acknowledgement = 'A';

while(1)

{

printf("Start Sending data \n");

delay\_ms(1000);

LoRabeginPacket(0);

//writebyte(Acknowledgement);

write(buffer, sizeof(buffer));

LoRaendPacket();

printf("Data sent \n");

}

#endif

}

void check\_ack()

{

printf("1 \n");

// NVIC\_EnableIRQ(TIMER0\_IRQn);

//NVIC\_DisableIRQ(TIMER0\_IRQn);

}

**LoRa.c:**

/\*

\* LoRa.c

\*

\* Created on: Oct 29, 2017

\* CTI One Corporation released for Dr. Harry Li for CMPE 245 Class use ONLY!

\*/

/\*

\* LoRa.cpp

\*

\* Created on: Oct 29, 2017

\* CTI One Corporation released for Dr. Harry Li for CMPE 245 Class use ONLY!

\*/

#include <stdio.h>

#include <stddef.h>

#include "LoRa.h"

#include "ssp.h"

#include "extint.h"

// registers

#define REG\_FIFO 0x00

#define REG\_OP\_MODE 0x01

#define REG\_FRF\_MSB 0x06

#define REG\_FRF\_MID 0x07

#define REG\_FRF\_LSB 0x08

#define REG\_PA\_CONFIG 0x09

#define REG\_LNA 0x0c

#define REG\_FIFO\_ADDR\_PTR 0x0d

#define REG\_FIFO\_TX\_BASE\_ADDR 0x0e

#define REG\_FIFO\_RX\_BASE\_ADDR 0x0f

#define REG\_FIFO\_RX\_CURRENT\_ADDR 0x10

#define REG\_IRQ\_FLAGS 0x12

#define REG\_RX\_NB\_BYTES 0x13

#define REG\_PKT\_RSSI\_VALUE 0x1a

#define REG\_PKT\_SNR\_VALUE 0x1b

#define REG\_MODEM\_CONFIG\_1 0x1d

#define REG\_MODEM\_CONFIG\_2 0x1e

#define REG\_PREAMBLE\_MSB 0x20

#define REG\_PREAMBLE\_LSB 0x21

#define REG\_PAYLOAD\_LENGTH 0x22

#define REG\_RSSI\_WIDEBAND 0x2c

#define REG\_DETECTION\_OPTIMIZE 0x31

#define REG\_DETECTION\_THRESHOLD 0x37

#define REG\_SYNC\_WORD 0x39

#define REG\_DIO\_MAPPING\_1 0x40

#define REG\_VERSION 0x42

// modes

#define MODE\_LONG\_RANGE\_MODE 0x80

#define MODE\_SLEEP 0x00

#define MODE\_STDBY 0x01

#define MODE\_TX 0x03

#define MODE\_RX\_CONTINUOUS 0x05

#define MODE\_RX\_SINGLE 0x06

// PA config

#define PA\_BOOST 0x80

// IRQ masks

#define IRQ\_TX\_DONE\_MASK 0x08

#define IRQ\_PAYLOAD\_CRC\_ERROR\_MASK 0x20

#define IRQ\_RX\_DONE\_MASK 0x40

#define MAX\_PKT\_LENGTH 255

#define LOW 0

#define HIGH 1

#define function\_not\_required 0

int \_packetIndex=0;

int \_implicitHeaderMode=0;

int \_frequency=0;

#if function\_not\_required

void (\*\_onReceive)(int);

#endif

char receiveLora=0;

#if function\_not\_required

void onReceivedata(int packetSize) {

// received a packet

printf("Received packet ");

int i=0;

// read packet

for (i = 0; i < packetSize; i++) {

printf("%c\n",(char)read());

}

// print RSSI of packet

print("' with RSSI ",packetRssi());

}

void EINT3\_IRQHandler(void)

{

LPC\_SC->EXTINT = EINT3; /\* clear interrupt \*/

printf("Interrupt triggered\n");

// Toggle Led On

//LPC\_GPIO0->FIOPIN ^= (1<<2);

handleDio0Rise();

/\*Clear interrupts \*/

LPC\_GPIOINT->IO0IntClr = 0xFFFFFFFF;

}

#endif

void gpioInit()

{

// Select P0.2 as GPIO for RESET

LPC\_PINCON->PINSEL0 &= ~(3<<4);

// P0.3 as GPIO

LPC\_PINCON->PINSEL0 &= ~(3<<6);

// P0.2 as output For RESET

LPC\_GPIO0->FIODIR |= (1<<2);

// P0.3 as input For DIO0

//LPC\_GPIO0->FIODIR &= ~(1<<3);

//LPC\_GPIOINT->IO0IntEnR |= (1<<3);

//NVIC\_EnableIRQ(EINT3\_IRQn);

}

void digitalWrite(uint8\_t pin, uint8\_t value)

{

printf("Pin : %d, value %d\n",pin,value);

if(value == 1)

{

LPC\_GPIO0->FIOPIN |= (1<<pin);

}

else if(value == 0)

{

LPC\_GPIO0->FIOPIN &= ~(1<<pin);

}

}

int LoRabegin(long frequency)

{

// setup pins

uint8\_t version =0;

int i=0;

gpioInit();

// perform reset

digitalWrite(LORA\_DEFAULT\_RESET\_PIN, LOW);

for(i=0;i<100000;i++);

digitalWrite(LORA\_DEFAULT\_RESET\_PIN, HIGH);

for(i=0;i<10000000;i++);

// set SS high

CHIP\_DESELECT();

//digitalWrite(\_ss, HIGH);

SSP1Init();

// start SPI

//SPI.begin();

// check version

version = readRegister(REG\_VERSION);

printf("Version is %x\n",version);

if (version != 0x12) {

return 0;

}

// put in sleep mode

sleep();

// set frequency

setFrequency(frequency);

// set base addresses

writeRegister(REG\_FIFO\_TX\_BASE\_ADDR, 0);

writeRegister(REG\_FIFO\_RX\_BASE\_ADDR, 0);

// set LNA boost

writeRegister(REG\_LNA, readRegister(REG\_LNA) | 0x03);

// set output power to 17 dBm

setTxPower(17);

// put in standby mode

idle();

return 1;

}

void end()

{

// put in sleep mode

sleep();

// stop SPI

//SPI.end();

}

int LoRabeginPacket(int implicitHeader)

{

// put in standby mode

idle();

if (implicitHeader)

{

implicitHeaderMode();

}

else {

explicitHeaderMode();

}

// reset FIFO address and paload length

writeRegister(REG\_FIFO\_ADDR\_PTR, 0);

writeRegister(REG\_PAYLOAD\_LENGTH, 0);

return 1;

}

int LoRaendPacket()

{

uint8\_t rOut=0;

// put in TX mode

writeRegister(REG\_OP\_MODE, MODE\_LONG\_RANGE\_MODE | MODE\_TX);

// wait for TX done

while((readRegister(REG\_IRQ\_FLAGS) & IRQ\_TX\_DONE\_MASK) == 0);

// clear IRQ's

writeRegister(REG\_IRQ\_FLAGS, IRQ\_TX\_DONE\_MASK);

return 1;

}

int parsePacket(int size)

{

int packetLength = 0;

int irqFlags = readRegister(REG\_IRQ\_FLAGS);

if (size > 0)

{

implicitHeaderMode();

writeRegister(REG\_PAYLOAD\_LENGTH, size & 0xff);

}

else

{

explicitHeaderMode();

}

// clear IRQ's

writeRegister(REG\_IRQ\_FLAGS, irqFlags);

if ((irqFlags & IRQ\_RX\_DONE\_MASK) && (irqFlags & IRQ\_PAYLOAD\_CRC\_ERROR\_MASK) == 0)

{

// received a packet

\_packetIndex = 0;

// read packet length

if (\_implicitHeaderMode)

{

packetLength = readRegister(REG\_PAYLOAD\_LENGTH);

}

else

{

packetLength = readRegister(REG\_RX\_NB\_BYTES);

}

// set FIFO address to current RX address

writeRegister(REG\_FIFO\_ADDR\_PTR, readRegister(REG\_FIFO\_RX\_CURRENT\_ADDR));

// put in standby mode

idle();

}

else if (readRegister(REG\_OP\_MODE) != (MODE\_LONG\_RANGE\_MODE | MODE\_RX\_SINGLE))

{

// not currently in RX mode

// reset FIFO address

writeRegister(REG\_FIFO\_ADDR\_PTR, 0);

// put in single RX mode

writeRegister(REG\_OP\_MODE, MODE\_LONG\_RANGE\_MODE | MODE\_RX\_SINGLE);

}

return packetLength;

}

int packetRssi()

{

return (readRegister(REG\_PKT\_RSSI\_VALUE) - (\_frequency < 868E6 ? 164 : 157));

}

float packetSnr()

{

return ((int8\_t)readRegister(REG\_PKT\_SNR\_VALUE)) \* 0.25;

}

size\_t writebyte(uint8\_t byte)

{

return write(&byte, sizeof(byte));

}

size\_t write(const uint8\_t \*buffer, size\_t size)

{

int currentLength = readRegister(REG\_PAYLOAD\_LENGTH);

size\_t i=0;

// check size

if ((currentLength + size) > MAX\_PKT\_LENGTH)

{

size = MAX\_PKT\_LENGTH - currentLength;

}

// write data

for (i = 0; i < size; i++)

{

writeRegister(REG\_FIFO, buffer[i]);

}

// update length

writeRegister(REG\_PAYLOAD\_LENGTH, currentLength + size);

return size;

}

int available()

{

return (readRegister(REG\_RX\_NB\_BYTES) - \_packetIndex);

}

int read()

{

if (!available()) {

return -1;

}

\_packetIndex++;

return readRegister(REG\_FIFO);

}

int peek()

{

if (!available()) {

return -1;

}

// store current FIFO address

int currentAddress = readRegister(REG\_FIFO\_ADDR\_PTR);

// read

uint8\_t b = readRegister(REG\_FIFO);

// restore FIFO address

writeRegister(REG\_FIFO\_ADDR\_PTR, currentAddress);

return b;

}

void flush()

{

}

#if function\_not\_required

void onReceive(void(\*callback)(int))

{

\_onReceive = callback;

//writeRegister(REG\_DIO\_MAPPING\_1, 0x00);

if (callback)

{

writeRegister(REG\_DIO\_MAPPING\_1, 0x00);

//attachInterrupt(digitalPinToInterrupt(\_dio0), onDio0Rise, RISING);

}

else

{

//detachInterrupt(digitalPinToInterrupt(\_dio0));

}

}

void receive(int size)

{

if (size > 0)

{

implicitHeaderMode();

writeRegister(REG\_PAYLOAD\_LENGTH, size & 0xff);

}

else

{

explicitHeaderMode();

}

writeRegister(REG\_OP\_MODE, MODE\_LONG\_RANGE\_MODE | MODE\_RX\_CONTINUOUS);

}

#endif

void idle()

{

writeRegister(REG\_OP\_MODE, MODE\_LONG\_RANGE\_MODE | MODE\_STDBY);

}

void sleep()

{

writeRegister(REG\_OP\_MODE, MODE\_LONG\_RANGE\_MODE | MODE\_SLEEP);

}

void setTxPower(int level)

{

if (level < 2)

{

level = 2;

}

else if (level > 17)

{

level = 17;

}

writeRegister(REG\_PA\_CONFIG, PA\_BOOST | (level - 2));

}

void setFrequency(long frequency)

{

\_frequency = frequency;

printf("frequency is %d,%d\n",frequency,\_frequency);

uint64\_t frf = ((uint64\_t)frequency << 19) / 32000000;

writeRegister(REG\_FRF\_MSB, (uint8\_t)(frf >> 16));

writeRegister(REG\_FRF\_MID, (uint8\_t)(frf >> 8));

writeRegister(REG\_FRF\_LSB, (uint8\_t)(frf >> 0));

}

void setSpreadingFactor(int sf)

{

if (sf < 6)

{

sf = 6;

}

else if (sf > 12)

{

sf = 12;

}

if (sf == 6)

{

writeRegister(REG\_DETECTION\_OPTIMIZE, 0xc5);

writeRegister(REG\_DETECTION\_THRESHOLD, 0x0c);

}

else

{

writeRegister(REG\_DETECTION\_OPTIMIZE, 0xc3);

writeRegister(REG\_DETECTION\_THRESHOLD, 0x0a);

}

writeRegister(REG\_MODEM\_CONFIG\_2, (readRegister(REG\_MODEM\_CONFIG\_2) & 0x0f) | ((sf << 4) & 0xf0));

}

void setSignalBandwidth(long sbw)

{

int bw;

if (sbw <= 7.8E3) {

bw = 0;

} else if (sbw <= 10.4E3) {

bw = 1;

} else if (sbw <= 15.6E3) {

bw = 2;

} else if (sbw <= 20.8E3) {

bw = 3;

} else if (sbw <= 31.25E3) {

bw = 4;

} else if (sbw <= 41.7E3) {

bw = 5;

} else if (sbw <= 62.5E3) {

bw = 6;

} else if (sbw <= 125E3) {

bw = 7;

} else if (sbw <= 250E3) {

bw = 8;

} else /\*if (sbw <= 250E3)\*/ {

bw = 9;

}

writeRegister(REG\_MODEM\_CONFIG\_1, (readRegister(REG\_MODEM\_CONFIG\_1) & 0x0f) | (bw << 4));

}

void setCodingRate4(int denominator)

{

if (denominator < 5) {

denominator = 5;

} else if (denominator > 8) {

denominator = 8;

}

int cr = denominator - 4;

writeRegister(REG\_MODEM\_CONFIG\_1, (readRegister(REG\_MODEM\_CONFIG\_1) & 0xf1) | (cr << 1));

}

void setPreambleLength(long length)

{

writeRegister(REG\_PREAMBLE\_MSB, (uint8\_t)(length >> 8));

writeRegister(REG\_PREAMBLE\_LSB, (uint8\_t)(length >> 0));

}

void setSyncWord(int sw)

{

writeRegister(REG\_SYNC\_WORD, sw);

}

void crc()

{

writeRegister(REG\_MODEM\_CONFIG\_2, readRegister(REG\_MODEM\_CONFIG\_2) | 0x04);

}

void noCrc()

{

writeRegister(REG\_MODEM\_CONFIG\_2, readRegister(REG\_MODEM\_CONFIG\_2) & 0xfb);

}

uint8\_t random()

{

return readRegister(REG\_RSSI\_WIDEBAND);

}

/\*

void setPins(int ss, int reset, int dio0)

{

\_ss = ss;

\_reset = reset;

\_dio0 = dio0;

}

void dumpRegisters(Stream& out)

{

for (int i = 0; i < 128; i++) {

out.print("0x");

out.print(i, HEX);

out.print(": 0x");

out.println(readRegister(i), HEX);

}

}

\*/

void explicitHeaderMode()

{

\_implicitHeaderMode = 0;

writeRegister(REG\_MODEM\_CONFIG\_1, readRegister(REG\_MODEM\_CONFIG\_1) & 0xfe);

}

void implicitHeaderMode()

{

\_implicitHeaderMode = 1;

writeRegister(REG\_MODEM\_CONFIG\_1, readRegister(REG\_MODEM\_CONFIG\_1) | 0x01);

}

#if function\_not\_required

void handleDio0Rise()

{

int irqFlags = readRegister(REG\_IRQ\_FLAGS);

int i=0;

// clear IRQ's

writeRegister(REG\_IRQ\_FLAGS, irqFlags);

if ((irqFlags & IRQ\_PAYLOAD\_CRC\_ERROR\_MASK) == 0) {

// received a packet

\_packetIndex = 0;

// read packet length

int packetLength = \_implicitHeaderMode ? readRegister(REG\_PAYLOAD\_LENGTH) : readRegister(REG\_RX\_NB\_BYTES);

// set FIFO address to current RX address

writeRegister(REG\_FIFO\_ADDR\_PTR, readRegister(REG\_FIFO\_RX\_CURRENT\_ADDR));

// if (\_onReceive) {

// \_onReceive(packetLength);

// }

for (i = 0; i < packetLength; i++)

{

receiveLora = read();

printf("Receive data is %c\n",receiveLora);

}

// reset FIFO address

writeRegister(REG\_FIFO\_ADDR\_PTR, 0);

}

}

#endif

uint8\_t readRegister(uint8\_t address)

{

return singleTransfer(address & 0x7f, 0x00);

}

void writeRegister(uint8\_t address, uint8\_t value)

{

singleTransfer(address | 0x80, value);

}

uint8\_t singleTransfer(uint8\_t address, uint8\_t value)

{

uint8\_t response=0;

//digitalWrite(\_ss, LOW);

CHIP\_SELECT();

//SPI.beginTransaction(\_spiSettings);

response = ssp1Transfer(address);

//printf("response %x\n",response);

response = ssp1Transfer(value);

//printf("response %x\n",response);

CHIP\_DESELECT();

//digitalWrite(\_ss, HIGH);

return response;

}

#if function\_not\_required

void onDio0Rise()

{

handleDio0Rise();

}

#endif

**ssp.c:**

/\*

\* spi.cpp

\*

\* Created on: Oct 29, 2017

\* CTI One Corporation released for Dr. Harry Li for CMPE 245 Class use ONLY!

\*/

#include "ssp.h"

//#define CHIP\_SELECT (LPC\_GPIO0->FIOCLR |= (0x1<<6))

//#define CHIP\_DESELECT (LPC\_GPIO0->FIOSET |= (0x1<<6))

#define SSP\_BUFSIZE 128

/\* statistics of all the interrupts \*/

volatile uint32\_t interrupt0RxStat = 0;

volatile uint32\_t interrupt0OverRunStat = 0;

volatile uint32\_t interrupt0RxTimeoutStat = 0;

volatile uint32\_t interrupt1RxStat = 0;

volatile uint32\_t interrupt1OverRunStat = 0;

volatile uint32\_t interrupt1RxTimeoutStat = 0;

uint8\_t sendBuffer[SSP\_BUFSIZE];

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

\*SSP1\_IRQHandler - SSP1 interrupt handler

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

void SSP1\_IRQHandler(void)

{

uint32\_t regValue;

regValue = LPC\_SSP1->MIS;

if ( regValue & SSPMIS\_RORMIS ) /\* Receive overrun interrupt \*/

{

interrupt1OverRunStat++;

LPC\_SSP1->ICR = SSPICR\_RORIC; /\* clear interrupt \*/

}

if ( regValue & SSPMIS\_RTMIS ) /\* Receive timeout interrupt \*/

{

interrupt1RxTimeoutStat++;

LPC\_SSP1->ICR = SSPICR\_RTIC; /\* clear interrupt \*/

}

/\* please be aware that, in main and ISR, CurrentRxIndex and CurrentTxIndex

are shared as global variables. It may create some race condition that main

and ISR manipulate these variables at the same time. SSPSR\_BSY checking (polling)

in both main and ISR could prevent this kind of race condition \*/

if ( regValue & SSPMIS\_RXMIS ) /\* Rx at least half full \*/

{

interrupt1RxStat++; /\* receive until it's empty \*/

}

return;

}

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

\* SSP1Init - Initialize SSP1 module

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

void SSP1Init()

{

// Enable SSP1

LPC\_SC->PCONP |= (1 << SSP1\_PCONP\_ENABLE);

// Clock selection for SSP1

LPC\_SC->PCLKSEL0 |= (3<<20);

/\* PIN select P0.6 - SSP1 SSEL1, P0.7 - SSP1 SCK1

P0.8 - SSP1 MISO1, P0.9 - SSP1 MOSI1 \*/

LPC\_PINCON->PINSEL0 &= ~((0x3<<12)|(0x3<<14)|(0x3<<16)|(0x3<<18));

LPC\_PINCON->PINSEL0 |= ((0x2<<12)|(0x2<<14)|(0x2<<16)|(0x2<<18));

LPC\_PINCON->PINSEL0 &= ~(3<<12); //P0.6 as gpio

LPC\_GPIO0->FIODIR |= (1<<6); // SSP0 P0.6 defined as Outputs

// DSS data to 8-bit, Frame format SPI, CPOL = 0, CPHA = 0, and SCR is 15

LPC\_SSP1->CR0 = 0x0707;

// SSP1 CPSR clock pre-scale register, master mode, minimum divisor is 0x02

LPC\_SSP1->CPSR = 0x2;

/\* Enable the SSP Interrupt \*/

NVIC\_EnableIRQ(SSP1\_IRQn);

/\* Master mode \*/

LPC\_SSP1->CR1 &= ~(SSPCR1\_MS);

/\* SSP Enable \*/

LPC\_SSP1->CR1 = SSPCR1\_SSE;

LPC\_SSP1->IMSC = SSPIMSC\_RORIM | SSPIMSC\_RTIM;

}

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

\* ssp1Send - send data over SSP1

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

uint8\_t ssp1Send(uint8\_t \*buf, uint32\_t length)

{

uint32\_t i=0;

uint8\_t Dummy = 0;

for( i = 0 ; i < length ; i++ )

{

/\* Move on only if NOT busy and TX FIFO not full. \*/

while((LPC\_SSP1->SR & (SSP\_STAT\_TNF|SSP\_STAT\_BSY)) != SSP\_STAT\_TNF);

LPC\_SSP1->DR = \*buf;

buf++;

while((LPC\_SSP1->SR & (SSP\_STAT\_BSY|SSP\_STAT\_RNE)) != SSP\_STAT\_RNE);

/\* Whenever a byte is written, MISO FIFO counter increments, Clear FIFO

on MISO. Otherwise, when SSP1Receive() is called, previous data byte

is left in the FIFO. \*/

Dummy = LPC\_SSP1->DR;

}

return Dummy;

}

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

\* ssp1Transfer - Transmit and receive byte on SSP1

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

uint8\_t ssp1Transfer(uint8\_t dataByte)

{

uint8\_t data=0;

LPC\_SSP1->DR = dataByte;

while(LPC\_SSP1->SR & (1<<4));

data = LPC\_SSP1->DR;

return data;

}

/\*

uint8\_t ssp1Transfer(uint8\_t dataByte)

{

uint8\_t dummy=0;

sendBuffer[0] = dataByte;

dummy = ssp1Send((uint8\_t \*)sendBuffer, 1 );

return dummy;

}