**EE445L – Lab 10: Wireless Serial Communication**

Shih-Hao Fan, Samuel Wu, Kevin Gilbert, Gilberto Rodriguez III

04/14/14

**1.0 OBJECTIVE**

The purpose of this lab is to get familiar with the XBee module and the UART transmit/receive functionalities. Sam and Howard worked on the transmitting part of the XBee, and Kevin and Gilberto worked on the receiving part of the XBee.

**2.0 HARDWARE DESIGN**

The transmmision and receive part of the hardware are the same.



**3.0 SOFTWARE DESIGN**

Transmitting code

// UART2TestMain.c

// Runs on LM3S1968

// Tests the UART0 to implement bidirectional data transfer to and from a

// computer running HyperTerminal. This time, interrupts and FIFOs

// are used.

// Daniel Valvano

// October 9, 2011

/\* This example accompanies the book

"Embedded Systems: Real Time Interfacing to the Arm Cortex M3",

ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2011

Copyright 2011 by Jonathan W. Valvano, valvano@mail.utexas.edu

You may use, edit, run or distribute this file

as long as the above copyright notice remains

THIS SOFTWARE IS PROVIDED "AS IS". NO WARRANTIES, WHETHER EXPRESS, IMPLIED

OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF

MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE.

VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL,

OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

For more information about my classes, my research, and my books, see

http://users.ece.utexas.edu/~valvano/

\*/

// U0Rx (VCP receive) connected to PA0

// U0Tx (VCP transmit) connected to PA1

#include "UART2.h"

#include "inc/hw\_types.h"

#include "driverlib/sysctl.h"

#include "XBee.h"

static char HOWARD[100];

//---------------------OutCRLF---------------------

// Output a CR,LF to UART to go to a new line

// Input: none

// Output: none

void OutCRLF(void){

UART0\_OutChar(CR);

UART0\_OutChar(LF);

}

//debug code

int main(void){

char x;

char i;

SysCtlClockSet(SYSCTL\_SYSDIV\_4 | SYSCTL\_USE\_PLL | SYSCTL\_OSC\_MAIN | SYSCTL\_XTAL\_8MHZ);

PLL\_Init();

SysTick\_Init();

Output\_Init();

UART\_Init(); // initialize UART

XBee\_init();

while(1){

x = UART0\_InChar();

//x = 'x';

UART0\_OutChar(x);

printf("%c", x);

XBee\_CreateTxFrame(1, &x);

XBee\_SendTxFrame(1);

/\*if(UART0\_InChar() == 0x7E)

{

UART0\_InChar();

UART0\_InChar();

UART0\_InChar();

UART0\_InChar();

if(UART0\_InChar() == 0x00)

{

//return true;

}

UART0\_InChar();

//return false;

}\*/

//SysTick\_Wait10ms(2);

//UART1\_InString(HOWARD,100);

//HOWARD[0] = UART1\_InChar();

/\*if(HOWARD[5] == 0x00)

{

printf("sent good data");

} \*/

}

}

// UART2.c

// Runs on LM3S1968

// Use UART0 to implement bidirectional data transfer to and from a

// computer running HyperTerminal. This time, interrupts and FIFOs

// are used.

// Daniel Valvano

// October 9, 2011

// Modified by EE345L students Charlie Gough && Matt Hawk

// Modified by EE345M students Agustinus Darmawan && Mingjie Qiu

/\* This example accompanies the book

"Embedded Systems: Real Time Interfacing to the Arm Cortex M3",

ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2011

Program 5.11 Section 5.6, Program 3.10

Copyright 2011 by Jonathan W. Valvano, valvano@mail.utexas.edu

You may use, edit, run or distribute this file

as long as the above copyright notice remains

THIS SOFTWARE IS PROVIDED "AS IS". NO WARRANTIES, WHETHER EXPRESS, IMPLIED

OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF

MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE.

VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL,

OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

For more information about my classes, my research, and my books, see

http://users.ece.utexas.edu/~valvano/

\*/

// U0Rx (VCP receive) connected to PA0

// U0Tx (VCP transmit) connected to PA1

#include "FIFO.h"

#include "UART2.h"

#include "lm3s1968.h"

void DisableInterrupts(void); // Disable interrupts

void EnableInterrupts(void); // Enable interrupts

long StartCritical (void); // previous I bit, disable interrupts

void EndCritical(long sr); // restore I bit to previous value

void WaitForInterrupt(void); // low power mode

#define FIFOSIZE 16 // size of the FIFOs (must be power of 2)

#define FIFOSUCCESS 1 // return value on success

#define FIFOFAIL 0 // return value on failure

// create index implementation FIFO (see FIFO.h)

AddIndexFifo(Rx0, FIFOSIZE, char, FIFOSUCCESS, FIFOFAIL)

AddIndexFifo(Rx1, FIFOSIZE, char, FIFOSUCCESS, FIFOFAIL)

AddIndexFifo(Tx0, FIFOSIZE, char, FIFOSUCCESS, FIFOFAIL)

AddIndexFifo(Tx1, FIFOSIZE, char, FIFOSUCCESS, FIFOFAIL)

// Initialize UART0

// Baud rate is 115200 bits/sec

void UART\_Init(void){

/\* SYSCTL\_RCGC1\_R |= SYSCTL\_RCGC1\_UART0; // activate UART0

SYSCTL\_RCGC2\_R |= SYSCTL\_RCGC2\_GPIOA; // activate port A

RxFifo\_Init(); // initialize empty FIFOs

TxFifo\_Init();

UART0\_CTL\_R &= ~UART\_CTL\_UARTEN; // disable UART

UART0\_IBRD\_R = 325; // IBRD = int(50,000,000 / (16 \* 9600)) = int(325.521)

UART0\_FBRD\_R = 33; // FBRD = int(0.521 \* 64 + 0.5) = 34

// 8 bit word length (no parity bits, one stop bit, FIFOs)

UART0\_LCRH\_R = (UART\_LCRH\_WLEN\_8|UART\_LCRH\_FEN);

UART0\_IFLS\_R &= ~0x3F; // clear TX and RX interrupt FIFO level fields

// configure interrupt for TX FIFO <= 1/8 full

// configure interrupt for RX FIFO >= 1/8 full

UART0\_IFLS\_R += (UART\_IFLS\_TX1\_8|UART\_IFLS\_RX1\_8);

// enable TX and RX FIFO interrupts and RX time-out interrupt

UART0\_IM\_R |= (UART\_IM\_RXIM|UART\_IM\_TXIM|UART\_IM\_RTIM);

UART0\_CTL\_R |= UART\_CTL\_UARTEN; // enable UART

GPIO\_PORTA\_AFSEL\_R |= 0x03; // enable alt funct on PA1-0

GPIO\_PORTA\_DEN\_R |= 0x03; // enable digital I/O on PA1-0

// UART0=priority 2

NVIC\_PRI1\_R = (NVIC\_PRI1\_R&0xFFFF00FF)|0x00004000; // bits 13-15

NVIC\_EN0\_R |= NVIC\_EN0\_INT5; // enable interrupt 5 in NVIC

EnableInterrupts();\*/

SYSCTL\_RCGC1\_R |= SYSCTL\_RCGC1\_UART0; // activate UART0

SYSCTL\_RCGC2\_R |= SYSCTL\_RCGC2\_GPIOA; // activate port A

SYSCTL\_RCGC1\_R |= SYSCTL\_RCGC1\_UART1; // activate UART1

SYSCTL\_RCGC2\_R |= SYSCTL\_RCGC2\_GPIOD; // activate port D

Rx0Fifo\_Init(); // initialize empty FIFOs

Rx1Fifo\_Init();

Tx0Fifo\_Init();

Tx1Fifo\_Init();

UART0\_CTL\_R &= ~UART\_CTL\_UARTEN; // disable UART

UART0\_IBRD\_R = 325; // IBRD = int(50,000,000 / (16 \* 9600)) = int(325.521)

UART0\_FBRD\_R = 33; // FBRD = int(0.521 \* 64 + 0.5) = 34

// 8 bit word length (no parity bits, one stop bit, FIFOs)

UART0\_LCRH\_R = (UART\_LCRH\_WLEN\_8|UART\_LCRH\_FEN);

UART0\_IFLS\_R &= ~0x3F; // clear TX and RX interrupt FIFO level fields

// configure interrupt for TX FIFO <= 1/8 full

// configure interrupt for RX FIFO >= 1/8 full

UART0\_IFLS\_R += (UART\_IFLS\_TX1\_8|UART\_IFLS\_RX1\_8);

// enable TX and RX FIFO interrupts and RX time-out interrupt

UART0\_IM\_R |= (UART\_IM\_RXIM|UART\_IM\_TXIM|UART\_IM\_RTIM);

UART0\_CTL\_R |= UART\_CTL\_UARTEN; // enable UART

GPIO\_PORTA\_AFSEL\_R |= 0x03; // enable alt funct on PA1-0

GPIO\_PORTA\_DEN\_R |= 0x03; // enable digital I/O on PA1-0

// UART0=priority 2

UART1\_CTL\_R &= ~UART\_CTL\_UARTEN; // disable UART

UART1\_IBRD\_R = 325; // IBRD = int(50,000,000 / (16 \* 9600)) = int(325.521)

UART1\_FBRD\_R = 34; // FBRD = int(0.521 \* 64 + 0.5) = 34

// 8 bit word length (no parity bits, one stop bit, FIFOs)

UART1\_LCRH\_R = (UART\_LCRH\_WLEN\_8|UART\_LCRH\_FEN);

UART1\_IFLS\_R &= ~0x3F; // clear TX and RX interrupt FIFO level fields

// configure interrupt for TX FIFO <= 1/8 full

// configure interrupt for RX FIFO >= 1/8 full

UART1\_IFLS\_R += (UART\_IFLS\_TX1\_8|UART\_IFLS\_RX1\_8);

// enable TX and RX FIFO interrupts and RX time-out interrupt

UART1\_IM\_R |= (UART\_IM\_RXIM|UART\_IM\_TXIM|UART\_IM\_RTIM);

UART1\_CTL\_R |= UART\_CTL\_UARTEN; // enable UART

GPIO\_PORTD\_AFSEL\_R |= 0x0C; // enable alt funct on PD3-2

GPIO\_PORTD\_DEN\_R |= 0x0C; // enable digital I/O on PD3-2

// UART1=priority 2

NVIC\_PRI1\_R = (NVIC\_PRI1\_R&0xFF0000FF)|0x00404000; // bits 13-15 and 21-23

NVIC\_EN0\_R |= NVIC\_EN0\_INT5; // enable interrupt 5 in NVIC

//NVIC\_PRI1\_R = (NVIC\_PRI1\_R&0xFF00FFFF)|0x00400000; // bits 21-23

NVIC\_EN0\_R |= NVIC\_EN0\_INT6; // enable interrupt 6 in NVIC

EnableInterrupts();

}

// copy from hardware RX FIFO to software RX FIFO

// stop when hardware RX FIFO is empty or software RX FIFO is full

void static copyHardwareToSoftware(void){

char letter;

while(((UART0\_FR\_R&UART\_FR\_RXFE) == 0) && (Rx0Fifo\_Size() < (FIFOSIZE - 1))){

letter = UART0\_DR\_R;

Rx0Fifo\_Put(letter);

}

}

// copy from software TX FIFO to hardware TX FIFO

// stop when software TX FIFO is empty or hardware TX FIFO is full

void static copySoftwareToHardware(void){

char letter;

while(((UART0\_FR\_R&UART\_FR\_TXFF) == 0) && (Tx0Fifo\_Size() > 0)){

Tx0Fifo\_Get(&letter);

UART0\_DR\_R = letter;

}

}

// input ASCII character from UART

// spin if RxFifo is empty

unsigned char UART0\_InChar(void){

char letter;

while(Rx0Fifo\_Get(&letter) == FIFOFAIL){};

return(letter);

}

// output ASCII character to UART

// spin if TxFifo is full

void UART0\_OutChar(unsigned char data){

while(Tx0Fifo\_Put(data) == FIFOFAIL){};

UART0\_IM\_R &= ~UART\_IM\_TXIM; // disable TX FIFO interrupt

copySoftwareToHardware();

UART0\_IM\_R |= UART\_IM\_TXIM; // enable TX FIFO interrupt

}

// at least one of three things has happened:

// hardware TX FIFO goes from 3 to 2 or less items

// hardware RX FIFO goes from 1 to 2 or more items

// UART receiver has timed out

void UART0\_Handler(void){

if(UART0\_RIS\_R&UART\_RIS\_TXRIS){ // hardware TX FIFO <= 2 items

UART0\_ICR\_R = UART\_ICR\_TXIC; // acknowledge TX FIFO

// copy from software TX FIFO to hardware TX FIFO

copySoftwareToHardware();

if(Tx0Fifo\_Size() == 0){ // software TX FIFO is empty

UART0\_IM\_R &= ~UART\_IM\_TXIM; // disable TX FIFO interrupt

}

}

if(UART0\_RIS\_R&UART\_RIS\_RXRIS){ // hardware RX FIFO >= 2 items

UART0\_ICR\_R = UART\_ICR\_RXIC; // acknowledge RX FIFO

// copy from hardware RX FIFO to software RX FIFO

copyHardwareToSoftware();

}

if(UART0\_RIS\_R&UART\_RIS\_RTRIS){ // receiver timed out

UART0\_ICR\_R = UART\_ICR\_RTIC; // acknowledge receiver time out

// copy from hardware RX FIFO to software RX FIFO

copyHardwareToSoftware();

}

}

//------------UART\_OutString------------

// Output String (NULL termination)

// Input: pointer to a NULL-terminated string to be transferred

// Output: none

void UART0\_OutString(char \*pt){

while(\*pt){

UART0\_OutChar(\*pt);

pt++;

}

}

//------------UART\_InUDec------------

// InUDec accepts ASCII input in unsigned decimal format

// and converts to a 32-bit unsigned number

// valid range is 0 to 4294967295 (2^32-1)

// Input: none

// Output: 32-bit unsigned number

// If you enter a number above 4294967295, it will return an incorrect value

// Backspace will remove last digit typed

unsigned long UART0\_InUDec(void){

unsigned long number=0, length=0;

char character;

character = UART0\_InChar();

while(character != CR){ // accepts until <enter> is typed

// The next line checks that the input is a digit, 0-9.

// If the character is not 0-9, it is ignored and not echoed

if((character>='0') && (character<='9')) {

number = 10\*number+(character-'0'); // this line overflows if above 4294967295

length++;

UART0\_OutChar(character);

}

// If the input is a backspace, then the return number is

// changed and a backspace is outputted to the screen

else if((character==BS) && length){

number /= 10;

length--;

UART0\_OutChar(character);

}

character = UART0\_InChar();

}

return number;

}

//-----------------------UART\_OutUDec-----------------------

// Output a 32-bit number in unsigned decimal format

// Input: 32-bit number to be transferred

// Output: none

// Variable format 1-10 digits with no space before or after

void UART0\_OutUDec(unsigned long n){

// This function uses recursion to convert decimal number

// of unspecified length as an ASCII string

if(n >= 10){

UART0\_OutUDec(n/10);

n = n%10;

}

UART0\_OutChar(n+'0'); /\* n is between 0 and 9 \*/

}

//---------------------UART\_InUHex----------------------------------------

// Accepts ASCII input in unsigned hexadecimal (base 16) format

// Input: none

// Output: 32-bit unsigned number

// No '$' or '0x' need be entered, just the 1 to 8 hex digits

// It will convert lower case a-f to uppercase A-F

// and converts to a 16 bit unsigned number

// value range is 0 to FFFFFFFF

// If you enter a number above FFFFFFFF, it will return an incorrect value

// Backspace will remove last digit typed

unsigned long UART0\_InUHex(void){

unsigned long number=0, digit, length=0;

char character;

character = UART0\_InChar();

while(character != CR){

digit = 0x10; // assume bad

if((character>='0') && (character<='9')){

digit = character-'0';

}

else if((character>='A') && (character<='F')){

digit = (character-'A')+0xA;

}

else if((character>='a') && (character<='f')){

digit = (character-'a')+0xA;

}

// If the character is not 0-9 or A-F, it is ignored and not echoed

if(digit <= 0xF){

number = number\*0x10+digit;

length++;

UART0\_OutChar(character);

}

// Backspace outputted and return value changed if a backspace is inputted

else if((character==BS) && length){

number /= 0x10;

length--;

UART0\_OutChar(character);

}

character = UART0\_InChar();

}

return number;

}

//--------------------------UART\_OutUHex----------------------------

// Output a 32-bit number in unsigned hexadecimal format

// Input: 32-bit number to be transferred

// Output: none

// Variable format 1 to 8 digits with no space before or after

void UART0\_OutUHex(unsigned long number){

// This function uses recursion to convert the number of

// unspecified length as an ASCII string

if(number >= 0x10){

UART0\_OutUHex(number/0x10);

UART0\_OutUHex(number%0x10);

}

else{

if(number < 0xA){

UART0\_OutChar(number+'0');

}

else{

UART0\_OutChar((number-0x0A)+'A');

}

}

}

//------------UART\_InString------------

// Accepts ASCII characters from the serial port

// and adds them to a string until <enter> is typed

// or until max length of the string is reached.

// It echoes each character as it is inputted.

// If a backspace is inputted, the string is modified

// and the backspace is echoed

// terminates the string with a null character

// uses busy-waiting synchronization on RDRF

// Input: pointer to empty buffer, size of buffer

// Output: Null terminated string

// -- Modified by Agustinus Darmawan + Mingjie Qiu --

void UART0\_InString(char \*bufPt, unsigned short max) {

int length=0;

char character;

character = UART0\_InChar();

while(character != CR){

if(character == BS){

if(length){

bufPt--;

length--;

UART0\_OutChar(BS);

}

}

else if(length < max){

\*bufPt = character;

bufPt++;

length++;

UART0\_OutChar(character);

}

character = UART0\_InChar();

}

\*bufPt = 0;

}

// copy from hardware RX FIFO to software RX FIFO

// stop when hardware RX FIFO is empty or software RX FIFO is full

void static copyHardwareToSoftware1(void){

char letter;

while(((UART1\_FR\_R&UART\_FR\_RXFE) == 0) && (Rx1Fifo\_Size() < (FIFOSIZE - 1))){

letter = UART1\_DR\_R;

Rx1Fifo\_Put(letter);

}

}

// copy from software TX FIFO to hardware TX FIFO

// stop when software TX FIFO is empty or hardware TX FIFO is full

void static copySoftwareToHardware1(void){

char letter;

while(((UART1\_FR\_R&UART\_FR\_TXFF) == 0) && (Tx1Fifo\_Size() > 0)){

Tx1Fifo\_Get(&letter);

UART1\_DR\_R = letter;

}

}

// input ASCII character from UART

// spin if RxFifo is empty

unsigned char UART1\_InChar(void){

char letter;

while(Rx1Fifo\_Get(&letter) == FIFOFAIL){};

return(letter);

}

// output ASCII character to UART

// spin if TxFifo is full

void UART1\_OutChar(unsigned char data){

while(Tx1Fifo\_Put(data) == FIFOFAIL){};

UART1\_IM\_R &= ~UART\_IM\_TXIM; // disable TX FIFO interrupt

copySoftwareToHardware1();

UART1\_IM\_R |= UART\_IM\_TXIM; // enable TX FIFO interrupt

}

// at least one of three things has happened:

// hardware TX FIFO goes from 3 to 2 or less items

// hardware RX FIFO goes from 1 to 2 or more items

// UART receiver has timed out

void UART1\_Handler(void){

if(UART1\_RIS\_R&UART\_RIS\_TXRIS){ // hardware TX FIFO <= 2 items

UART1\_ICR\_R = UART\_ICR\_TXIC; // acknowledge TX FIFO

// copy from software TX FIFO to hardware TX FIFO

copySoftwareToHardware1();

if(Tx1Fifo\_Size() == 0){ // software TX FIFO is empty

UART1\_IM\_R &= ~UART\_IM\_TXIM; // disable TX FIFO interrupt

}

}

if(UART1\_RIS\_R&UART\_RIS\_RXRIS){ // hardware RX FIFO >= 2 items

UART1\_ICR\_R = UART\_ICR\_RXIC; // acknowledge RX FIFO

// copy from hardware RX FIFO to software RX FIFO

copyHardwareToSoftware1();

}

if(UART1\_RIS\_R&UART\_RIS\_RTRIS){ // receiver timed out

UART1\_ICR\_R = UART\_ICR\_RTIC; // acknowledge receiver time out

// copy from hardware RX FIFO to software RX FIFO

copyHardwareToSoftware1();

}

}

//------------UART\_OutString1------------

// Output String (NULL termination)

// Input: pointer to a NULL-terminated string to be transferred

// Output: none

void UART1\_OutString(char \*pt){

while(\*pt){

UART1\_OutChar(\*pt);

pt++;

}

}

//------------UART\_InUDec1------------

// InUDec accepts ASCII input in unsigned decimal format

// and converts to a 32-bit unsigned number

// valid range is 0 to 4294967295 (2^32-1)

// Input: none

// Output: 32-bit unsigned number

// If you enter a number above 4294967295, it will return an incorrect value

// Backspace will remove last digit typed

unsigned long UART1\_InUDec(void){

unsigned long number=0, length=0;

char character;

character = UART1\_InChar();

while(character != CR){ // accepts until <enter> is typed

// The next line checks that the input is a digit, 0-9.

// If the character is not 0-9, it is ignored and not echoed

if((character>='0') && (character<='9')) {

number = 10\*number+(character-'0'); // this line overflows if above 4294967295

length++;

UART1\_OutChar(character);

}

// If the input is a backspace, then the return number is

// changed and a backspace is outputted to the screen

else if((character==BS) && length){

number /= 10;

length--;

UART1\_OutChar(character);

}

character = UART1\_InChar();

}

return number;

}

//-----------------------UART\_OutUDec1-----------------------

// Output a 32-bit number in unsigned decimal format

// Input: 32-bit number to be transferred

// Output: none

// Variable format 1-10 digits with no space before or after

void UART1\_OutUDec(unsigned long n){

// This function uses recursion to convert decimal number

// of unspecified length as an ASCII string

if(n >= 10){

UART1\_OutUDec(n/10);

n = n%10;

}

UART1\_OutChar(n+'0'); /\* n is between 0 and 9 \*/

}

//---------------------UART\_InUHex1----------------------------------------

// Accepts ASCII input in unsigned hexadecimal (base 16) format

// Input: none

// Output: 32-bit unsigned number

// No '$' or '0x' need be entered, just the 1 to 8 hex digits

// It will convert lower case a-f to uppercase A-F

// and converts to a 16 bit unsigned number

// value range is 0 to FFFFFFFF

// If you enter a number above FFFFFFFF, it will return an incorrect value

// Backspace will remove last digit typed

unsigned long UART1\_InUHex(void){

unsigned long number=0, digit, length=0;

char character;

character = UART1\_InChar();

while(character != CR){

digit = 0x10; // assume bad

if((character>='0') && (character<='9')){

digit = character-'0';

}

else if((character>='A') && (character<='F')){

digit = (character-'A')+0xA;

}

else if((character>='a') && (character<='f')){

digit = (character-'a')+0xA;

}

// If the character is not 0-9 or A-F, it is ignored and not echoed

if(digit <= 0xF){

number = number\*0x10+digit;

length++;

UART1\_OutChar(character);

}

// Backspace outputted and return value changed if a backspace is inputted

else if((character==BS) && length){

number /= 0x10;

length--;

UART1\_OutChar(character);

}

character = UART1\_InChar();

}

return number;

}

//--------------------------UART\_OutUHex1----------------------------

// Output a 32-bit number in unsigned hexadecimal format

// Input: 32-bit number to be transferred

// Output: none

// Variable format 1 to 8 digits with no space before or after

void UART1\_OutUHex(unsigned long number){

// This function uses recursion to convert the number of

// unspecified length as an ASCII string

if(number >= 0x10){

UART1\_OutUHex(number/0x10);

UART1\_OutUHex(number%0x10);

}

else{

if(number < 0xA){

UART1\_OutChar(number+'0');

}

else{

UART1\_OutChar((number-0x0A)+'A');

}

}

}

//------------UART\_InString1------------

// Accepts ASCII characters from the serial port

// and adds them to a string until <enter> is typed

// or until max length of the string is reached.

// It echoes each character as it is inputted.

// If a backspace is inputted, the string is modified

// and the backspace is echoed

// terminates the string with a null character

// uses busy-waiting synchronization on RDRF

// Input: pointer to empty buffer, size of buffer

// Output: Null terminated string

// -- Modified by Agustinus Darmawan + Mingjie Qiu --

void UART1\_InString(char \*bufPt, unsigned short max) {

int length=0;

char character;

character = UART1\_InChar();

while(character != CR){

if(character == BS){

if(length){

bufPt--;

length--;

UART1\_OutChar(BS);

}

}

else if(length < max){

\*bufPt = character;

bufPt++;

length++;

UART1\_OutChar(character);

}

character = UART1\_InChar();

}

\*bufPt = 0;

}

#include "XBee.h"

#include <stdbool.h>

void XBee\_init()

{

//printf("XBee initializing.");

do{

UART1\_OutChar('X');

SysTick\_Wait10ms(110);

UART1\_OutChar('+');

UART1\_OutChar('+');

UART1\_OutChar('+');

SysTick\_Wait10ms(110);

}while(!checkResponse());

do{sendATCommand(setDA, DestinationLo);} while(!checkResponse());

//printstatus();

do{sendATCommand(setDH, 0);} while(!checkResponse());

//printstatus();

do{sendATCommand(setMY, MY);} while(!checkResponse());

//printstatus();

do{sendATCommand(setAP, 1);} while(!checkResponse());

printstatus();

do{sendATCommand(endCommand, -1);} while(!checkResponse());

printstatus();

SysTick\_Wait10ms(50);

Output\_Clear();

printf("XBee is initialized.");

SysTick\_Wait10ms(50);

Output\_Clear();

printf("Ready to transfer...\r");

//SysTick\_Wait10ms(50);

//Output\_Clear();

}

void printstatus(void)

{

Output\_Clear();

printf("XBee initializing.");

SysTick\_Wait10ms(50);

Output\_Clear();

printf("XBee initializing..");

SysTick\_Wait10ms(50);

Output\_Clear();

printf("XBee initializing...");

SysTick\_Wait10ms(50);

Output\_Clear();

printf("XBee initializing....");

SysTick\_Wait10ms(50);

}

void static sendATCommand(int command, int number)

{

char cNumber[2];

char toSend[6];

sprintf(cNumber, "%X", number);

switch(command)

{

case setDA:

strcpy(toSend, "ATDL");

strcat(toSend, cNumber);

UART1\_OutString(toSend);

UART1\_OutChar(CR);

SysTick\_Wait10ms(2);

//checkResponse();

break;

case setDH:

strcpy(toSend, "ATDH");

strcat(toSend, cNumber);

UART1\_OutString(toSend);

UART1\_OutChar(CR);

SysTick\_Wait10ms(2);

//checkResponse();

break;

case setMY:

strcpy(toSend, "ATMY");

strcat(toSend, cNumber);

UART1\_OutString(toSend);

UART1\_OutChar(CR);

SysTick\_Wait10ms(2);

//checkResponse();

break;

case setAP:

strcpy(toSend, "ATAP");

strcat(toSend, cNumber);

UART1\_OutString(toSend);

UART1\_OutChar(CR);

SysTick\_Wait10ms(2);

//checkResponse();

break;

case endCommand:

strcpy(toSend, "ATCN");

UART1\_OutString(toSend);

UART1\_OutChar(CR);

SysTick\_Wait10ms(2);

//checkResponse();

break;

}

}

bool checkResponse()

{

char response[2];

UART1\_InString(response, 2);

if(response[0] == 'O' && response[1] == 'K')

{

//printf("good response\r");

return true;

}

else

{

//printf("bad response\r");

return false;

}

}

void XBee\_CreateTxFrame(unsigned short length, char message[])

{

int i = 0;

ID = 1;

Frame[0] = (char)StartDelimiter;

Frame[1] = (char)LengthHi;

Frame[2] = (char)length+5;

Frame[3] = (char)API;

Frame[4] = (char)ID;

Frame[5] = (char)DestinationHi;

Frame[6] = (char)DestinationLo;

Frame[7] = (char)Opt;

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

{

Frame[8+i] = message[i];

}

Frame[8+length] = (char)Checksum(length, message);

newID();

}

void XBee\_SendTxFrame(unsigned short length)

{

int i = 0;

for( i = 0; i < length + 9; i++)

{

UART1\_OutChar(Frame[i]);

}

}

unsigned short Checksum(unsigned short length, char message[])

{

unsigned int checksum = 0;

int i = 0;

checksum = 0xFF - (API + ID + DestinationHi + DestinationLo + Opt);

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

{

checksum -= message[i];

}

return checksum & 0xFF;

}

void newID()

{

if(ID == 255)

{

ID = 1;

}

else

{

ID++;

}

}

Receiving part

// UART2TestMain.c

// Runs on LM3S1968

// Tests the UART0 to implement bidirectional data transfer to and from a

// computer running HyperTerminal. This time, interrupts and FIFOs

// are used.

// Daniel Valvano

// October 9, 2011

/\* This example accompanies the book

"Embedded Systems: Real Time Interfacing to the Arm Cortex M3",

ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2011

Copyright 2011 by Jonathan W. Valvano, valvano@mail.utexas.edu

You may use, edit, run or distribute this file

as long as the above copyright notice remains

THIS SOFTWARE IS PROVIDED "AS IS". NO WARRANTIES, WHETHER EXPRESS, IMPLIED

OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF

MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE.

VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL,

OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

For more information about my classes, my research, and my books, see

http://users.ece.utexas.edu/~valvano/

\*/

// U0Rx (VCP receive) connected to PA0

// U0Tx (VCP transmit) connected to PA1

#include "UART2.h"

#include "inc/hw\_types.h"

#include "driverlib/sysctl.h"

#include "XBee.h"

#define MAXMSGSIZE 100

#define DELIMITER 0x7E

//---------------------OutCRLF---------------------

// Output a CR,LF to UART to go to a new line

// Input: none

// Output: none

void OutCRLF(void){

UART\_OutChar(CR);

UART\_OutChar(LF);

}

//debug code

int main(void){

char checkDelimiter;

char msg[MAXMSGSIZE];

int msgSize;

int msgIndex;

int i;

SysCtlClockSet(SYSCTL\_SYSDIV\_4 | SYSCTL\_USE\_PLL | SYSCTL\_OSC\_MAIN |

SYSCTL\_XTAL\_8MHZ);

PLL\_Init();

SysTick\_Init();

Output\_Init();

UART\_Init(); // initialize UART

XBee\_init();

while(1)

{

checkDelimiter = UART\_InChar();

/\*UART\_InString(msg, MAXMSGSIZE);

printf("%d", msg[0]);

printf("%d", msg[1]);\*/

if(checkDelimiter == DELIMITER)

{

//printf("got good data");

if(receiveRxFrame(msg, MAXMSGSIZE, &msgSize))

{

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

{

printf("%c", msg[i]);

}

//printf("\r");

}

else

{

printf("Checksum was wrong or msg too long.\r");

}

}

}

/\*unsigned char i;

char string[20]; // global to assist in debugging

unsigned long n;

//

// Set the clocking to run at 50MHz from the PLL.

//

SysCtlClockSet(SYSCTL\_SYSDIV\_4 | SYSCTL\_USE\_PLL | SYSCTL\_OSC\_MAIN |

SYSCTL\_XTAL\_8MHZ);

UART\_Init(); // initialize UART

OutCRLF();

for(i='A'; i<='Z'; i=i+1){// print the uppercase alphabet

UART\_OutChar(i);

}

OutCRLF();

UART\_OutChar(' ');

for(i='a'; i<='z'; i=i+1){// print the lowercase alphabet

UART\_OutChar(i);

}

OutCRLF();

UART\_OutChar('-');

UART\_OutChar('-');

UART\_OutChar('>');

while(1){

UART\_OutString("InString: ");

UART\_InString(string,19);

UART\_OutString(" OutString="); UART\_OutString(string); OutCRLF();

UART\_OutString("InUDec: "); n=UART\_InUDec();

UART\_OutString(" OutUDec="); UART\_OutUDec(n); OutCRLF();

UART\_OutString("InUHex: "); n=UART\_InUHex();

UART\_OutString(" OutUHex="); UART\_OutUHex(n); OutCRLF();

}\*/

}

// UART2.c

// Runs on LM3S1968

// Use UART0 to implement bidirectional data transfer to and from a

// computer running HyperTerminal. This time, interrupts and FIFOs

// are used.

// Daniel Valvano

// October 9, 2011

// Modified by EE345L students Charlie Gough && Matt Hawk

// Modified by EE345M students Agustinus Darmawan && Mingjie Qiu

/\* This example accompanies the book

"Embedded Systems: Real Time Interfacing to the Arm Cortex M3",

ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2011

Program 5.11 Section 5.6, Program 3.10

Copyright 2011 by Jonathan W. Valvano, valvano@mail.utexas.edu

You may use, edit, run or distribute this file

as long as the above copyright notice remains

THIS SOFTWARE IS PROVIDED "AS IS". NO WARRANTIES, WHETHER EXPRESS, IMPLIED

OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF

MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE.

VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL,

OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

For more information about my classes, my research, and my books, see

http://users.ece.utexas.edu/~valvano/

\*/

// U0Rx (VCP receive) connected to PA0

// U0Tx (VCP transmit) connected to PA1

#include "FIFO.h"

#include "UART2.h"

#include "lm3s1968.h"

void DisableInterrupts(void); // Disable interrupts

void EnableInterrupts(void); // Enable interrupts

long StartCritical (void); // previous I bit, disable interrupts

void EndCritical(long sr); // restore I bit to previous value

void WaitForInterrupt(void); // low power mode

#define FIFOSIZE 16 // size of the FIFOs (must be power of 2)

#define FIFOSUCCESS 1 // return value on success

#define FIFOFAIL 0 // return value on failure

// create index implementation FIFO (see FIFO.h)

AddIndexFifo(Rx, FIFOSIZE, char, FIFOSUCCESS, FIFOFAIL)

AddIndexFifo(Tx, FIFOSIZE, char, FIFOSUCCESS, FIFOFAIL)

// Initialize UART1

// Baud rate is 9600 bits/sec

void UART\_Init(void){

SYSCTL\_RCGC1\_R |= SYSCTL\_RCGC1\_UART1; // activate UART0

SYSCTL\_RCGC2\_R |= SYSCTL\_RCGC2\_GPIOD; // activate port A

RxFifo\_Init(); // initialize empty FIFOs

TxFifo\_Init();

UART1\_CTL\_R &= ~UART\_CTL\_UARTEN; // disable UART

UART1\_IBRD\_R = 325; // IBRD = int(50,000,000 / (16 \* 9600)) = int(325.5208)

UART1\_FBRD\_R = 34; // FBRD = int(0.5208 \* 64 + 0.5) = 34

// 8 bit word length (no parity bits, one stop bit, FIFOs)

UART1\_LCRH\_R = (UART\_LCRH\_WLEN\_8|UART\_LCRH\_FEN);

UART1\_IFLS\_R &= ~0x3F; // clear TX and RX interrupt FIFO level fields

// configure interrupt for TX FIFO <= 1/8 full

// configure interrupt for RX FIFO >= 1/8 full

UART1\_IFLS\_R += (UART\_IFLS\_TX1\_8|UART\_IFLS\_RX1\_8);

// enable TX and RX FIFO interrupts and RX time-out interrupt

UART1\_IM\_R |= (UART\_IM\_RXIM|UART\_IM\_TXIM|UART\_IM\_RTIM);

UART1\_CTL\_R |= UART\_CTL\_UARTEN; // enable UART

GPIO\_PORTD\_AFSEL\_R |= 0x0C; // enable alt funct on PA1-0

GPIO\_PORTD\_DEN\_R |= 0x0C; // enable digital I/O on PA1-0

// UART1=priority 2

NVIC\_PRI1\_R = (NVIC\_PRI1\_R&0xFF00FFFF)|0x00400000; // bits 13-15

NVIC\_EN0\_R |= NVIC\_EN0\_INT6; // enable interrupt 5 in NVIC

EnableInterrupts();

}

// copy from hardware RX FIFO to software RX FIFO

// stop when hardware RX FIFO is empty or software RX FIFO is full

void static copyHardwareToSoftware(void){

char letter;

while(((UART1\_FR\_R&UART\_FR\_RXFE) == 0) && (RxFifo\_Size() < (FIFOSIZE - 1))){

letter = UART1\_DR\_R;

RxFifo\_Put(letter);

}

}

// copy from software TX FIFO to hardware TX FIFO

// stop when software TX FIFO is empty or hardware TX FIFO is full

void static copySoftwareToHardware(void){

char letter;

while(((UART1\_FR\_R&UART\_FR\_TXFF) == 0) && (TxFifo\_Size() > 0)){

TxFifo\_Get(&letter);

UART1\_DR\_R = letter;

}

}

// input ASCII character from UART

// spin if RxFifo is empty

unsigned char UART\_InChar(void){

char letter;

while(RxFifo\_Get(&letter) == FIFOFAIL){};

return(letter);

}

// output ASCII character to UART

// spin if TxFifo is full

void UART\_OutChar(unsigned char data){

while(TxFifo\_Put(data) == FIFOFAIL){};

UART1\_IM\_R &= ~UART\_IM\_TXIM; // disable TX FIFO interrupt

copySoftwareToHardware();

UART1\_IM\_R |= UART\_IM\_TXIM; // enable TX FIFO interrupt

}

// at least one of three things has happened:

// hardware TX FIFO goes from 3 to 2 or less items

// hardware RX FIFO goes from 1 to 2 or more items

// UART receiver has timed out

void UART1\_Handler(void){

if(UART1\_RIS\_R&UART\_RIS\_TXRIS){ // hardware TX FIFO <= 2 items

UART1\_ICR\_R = UART\_ICR\_TXIC; // acknowledge TX FIFO

// copy from software TX FIFO to hardware TX FIFO

copySoftwareToHardware();

if(TxFifo\_Size() == 0){ // software TX FIFO is empty

UART1\_IM\_R &= ~UART\_IM\_TXIM; // disable TX FIFO interrupt

}

}

if(UART1\_RIS\_R&UART\_RIS\_RXRIS){ // hardware RX FIFO >= 2 items

UART1\_ICR\_R = UART\_ICR\_RXIC; // acknowledge RX FIFO

// copy from hardware RX FIFO to software RX FIFO

copyHardwareToSoftware();

}

if(UART1\_RIS\_R&UART\_RIS\_RTRIS){ // receiver timed out

UART1\_ICR\_R = UART\_ICR\_RTIC; // acknowledge receiver time out

// copy from hardware RX FIFO to software RX FIFO

copyHardwareToSoftware();

}

}

//------------UART\_OutString------------

// Output String (NULL termination)

// Input: pointer to a NULL-terminated string to be transferred

// Output: none

void UART\_OutString(char \*pt){

while(\*pt){

UART\_OutChar(\*pt);

pt++;

}

}

//------------UART\_InUDec------------

// InUDec accepts ASCII input in unsigned decimal format

// and converts to a 32-bit unsigned number

// valid range is 0 to 4294967295 (2^32-1)

// Input: none

// Output: 32-bit unsigned number

// If you enter a number above 4294967295, it will return an incorrect value

// Backspace will remove last digit typed

unsigned long UART\_InUDec(void){

unsigned long number=0, length=0;

char character;

character = UART\_InChar();

while(character != CR){ // accepts until <enter> is typed

// The next line checks that the input is a digit, 0-9.

// If the character is not 0-9, it is ignored and not echoed

if((character>='0') && (character<='9')) {

number = 10\*number+(character-'0'); // this line overflows if above 4294967295

length++;

UART\_OutChar(character);

}

// If the input is a backspace, then the return number is

// changed and a backspace is outputted to the screen

else if((character==BS) && length){

number /= 10;

length--;

UART\_OutChar(character);

}

character = UART\_InChar();

}

return number;

}

//-----------------------UART\_OutUDec-----------------------

// Output a 32-bit number in unsigned decimal format

// Input: 32-bit number to be transferred

// Output: none

// Variable format 1-10 digits with no space before or after

void UART\_OutUDec(unsigned long n){

// This function uses recursion to convert decimal number

// of unspecified length as an ASCII string

if(n >= 10){

UART\_OutUDec(n/10);

n = n%10;

}

UART\_OutChar(n+'0'); /\* n is between 0 and 9 \*/

}

//---------------------UART\_InUHex----------------------------------------

// Accepts ASCII input in unsigned hexadecimal (base 16) format

// Input: none

// Output: 32-bit unsigned number

// No '$' or '0x' need be entered, just the 1 to 8 hex digits

// It will convert lower case a-f to uppercase A-F

// and converts to a 16 bit unsigned number

// value range is 0 to FFFFFFFF

// If you enter a number above FFFFFFFF, it will return an incorrect value

// Backspace will remove last digit typed

unsigned long UART\_InUHex(void){

unsigned long number=0, digit, length=0;

char character;

character = UART\_InChar();

while(character != CR){

digit = 0x10; // assume bad

if((character>='0') && (character<='9')){

digit = character-'0';

}

else if((character>='A') && (character<='F')){

digit = (character-'A')+0xA;

}

else if((character>='a') && (character<='f')){

digit = (character-'a')+0xA;

}

// If the character is not 0-9 or A-F, it is ignored and not echoed

if(digit <= 0xF){

number = number\*0x10+digit;

length++;

UART\_OutChar(character);

}

// Backspace outputted and return value changed if a backspace is inputted

else if((character==BS) && length){

number /= 0x10;

length--;

UART\_OutChar(character);

}

character = UART\_InChar();

}

return number;

}

//--------------------------UART\_OutUHex----------------------------

// Output a 32-bit number in unsigned hexadecimal format

// Input: 32-bit number to be transferred

// Output: none

// Variable format 1 to 8 digits with no space before or after

void UART\_OutUHex(unsigned long number){

// This function uses recursion to convert the number of

// unspecified length as an ASCII string

if(number >= 0x10){

UART\_OutUHex(number/0x10);

UART\_OutUHex(number%0x10);

}

else{

if(number < 0xA){

UART\_OutChar(number+'0');

}

else{

UART\_OutChar((number-0x0A)+'A');

}

}

}

//------------UART\_InString------------

// Accepts ASCII characters from the serial port

// and adds them to a string until <enter> is typed

// or until max length of the string is reached.

// It echoes each character as it is inputted.

// If a backspace is inputted, the string is modified

// and the backspace is echoed

// terminates the string with a null character

// uses busy-waiting synchronization on RDRF

// Input: pointer to empty buffer, size of buffer

// Output: Null terminated string

// -- Modified by Agustinus Darmawan + Mingjie Qiu --

void UART\_InString(char \*bufPt, unsigned short max) {

int length=0;

char character;

character = UART\_InChar();

while(character != CR){

if(character == BS){

if(length){

bufPt--;

length--;

UART\_OutChar(BS);

}

}

else if(length < max){

\*bufPt = character;

bufPt++;

length++;

UART\_OutChar(character);

}

character = UART\_InChar();

}

\*bufPt = 0;

}

#include "XBee.h"

char previousMsgID;

void XBee\_init()

{

bool goodResponse = false;

char response[2];

do

{

UART\_OutChar('X');

SysTick\_Wait10ms(110);

UART\_OutChar('+');

UART\_OutChar('+');

UART\_OutChar('+');

SysTick\_Wait10ms(110);

UART\_InString(response, 2);

}

while(response[0] != 'O' && response[1] != 'K');

//printf("Good Response\r");

//sendATCommand(setRE, -1);

sendATCommand(setDA, destinationAddress);

sendATCommand(setDH, 0);

sendATCommand(setMY, myAddress);

sendATCommand(setAP, 1);

//sendATCommand(setIU, -1);

sendATCommand(endCommand, -1);

}

static void sendATCommand(int command, int number)

{

char cNumber[2];

char toSend[6];

char response[2];

sprintf(cNumber, "%X", number);

switch(command)

{

/\*case setRE:

do

{

strcpy(toSend, "ATRE");

UART\_OutString(toSend);

UART\_OutChar(CR);

SysTick\_Wait10ms(2);

UART\_InString(response, 2);

}

while(response[0] != 'O' || response[1] != 'K');

printf("Good Response\r");

break;\*/

case setDA:

do

{

strcpy(toSend, "ATDL");

strcat(toSend, cNumber);

UART\_OutString(toSend);

UART\_OutChar(CR);

SysTick\_Wait10ms(2);

UART\_InString(response, 2);

}

while(response[0] != 'O' || response[1] != 'K');

printf("Good Response\r");

break;

case setDH:

do

{

strcpy(toSend, "ATDH");

strcat(toSend, cNumber);

UART\_OutString(toSend);

UART\_OutChar(CR);

SysTick\_Wait10ms(2);

UART\_InString(response, 2);

}

while(response[0] != 'O' || response[1] != 'K');

printf("Good Response\r");

break;

case setMY:

do

{

strcpy(toSend, "ATMY");

strcat(toSend, cNumber);

UART\_OutString(toSend);

UART\_OutChar(CR);

SysTick\_Wait10ms(2);

UART\_InString(response, 2);

}

while(response[0] != 'O' || response[1] != 'K');

printf("Good Response\r");

break;

case setAP:

do

{

strcpy(toSend, "ATAP");

strcat(toSend, cNumber);

UART\_OutString(toSend);

UART\_OutChar(CR);

SysTick\_Wait10ms(2);

UART\_InString(response, 2);

}

while(response[0] != 'O' || response[1] != 'K');

printf("Good Response\r");

break;

/\*case setIU:

do

{

strcpy(toSend, "ATIU1");

UART\_OutString(toSend);

UART\_OutChar(CR);

SysTick\_Wait10ms(2);

UART\_InString(response, 2);

}

while(response[0] != 'O' || response[1] != 'K');

printf("Good Response\r");

break;\*/

case endCommand:

do

{

strcpy(toSend, "ATCN");

UART\_OutString(toSend);

UART\_OutChar(CR);

SysTick\_Wait10ms(2);

UART\_InString(response, 2);

}

while(response[0] != 'O' || response[1] != 'K');

printf("Good Response\r");

break;

}

}

bool receiveRxFrame(char\* msg, int msgMaxLength, int\* msgSize)

{

char MSG[2];

char length;

char copyLength;

char source;

char APInumber;

char destination;

char RSSI;

char opt;

char checkSum;

char receivedCheckSum;

int i;

\*msgSize = 0;

checkSum = 0;

length = UART\_InChar();

length = UART\_InChar();

copyLength = length;

APInumber = UART\_InChar();

checkSum += APInumber;

copyLength--;

source = UART\_InChar();

copyLength--;

source = UART\_InChar();

copyLength--;

checkSum += source;

RSSI = UART\_InChar();

copyLength--;

checkSum += RSSI;

opt = UART\_InChar();

copyLength--;

checkSum += opt;

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

{

if(copyLength == 0)

{

break;

}

\*msg = UART\_InChar();

checkSum += \*msg;

\*msgSize = \*msgSize + 1;

msg++;

copyLength--;

}

receivedCheckSum = UART\_InChar();

if((0xFF - checkSum == receivedCheckSum) && copyLength == 0)

{

return true;

}

else

{

printf("%d", copyLength);

return false;

}

}

**4.0 Measurement Data**

**4.1 Estimate the maximum bandwidth of your XBee wireless link.**

The maximum bandwidth for our XBee wireless is 960 Byte/sec.

**4.2 Estimate the range of your XBee wireless link.**

The range of the XBee wireless link is between 20meters and 100 meters. The actual measurement of our XBee range is 20 meters.

**5.0 Analysis and Discussion**

This lab was focused on the use of the wireless Xbee module. UART pins were used to communicate between the LM3S1968 Evaluation board and the Xbee device. The transmitting node communicated through UART0 via a USB connection to a computer where string input was captured. Once collected, the keyboard data was packed into a frame and transmitted through UART1 to the Xbee. The receiving node scanned UART1 input for the frame start delimiter [0x7E]. Once captured, the frame was translated and the embedded message was displayed onto the on-board OLED.