**EE445L – Lab10: Wireless Serial Communication**

Harley Ross and Dalton Altstaetter

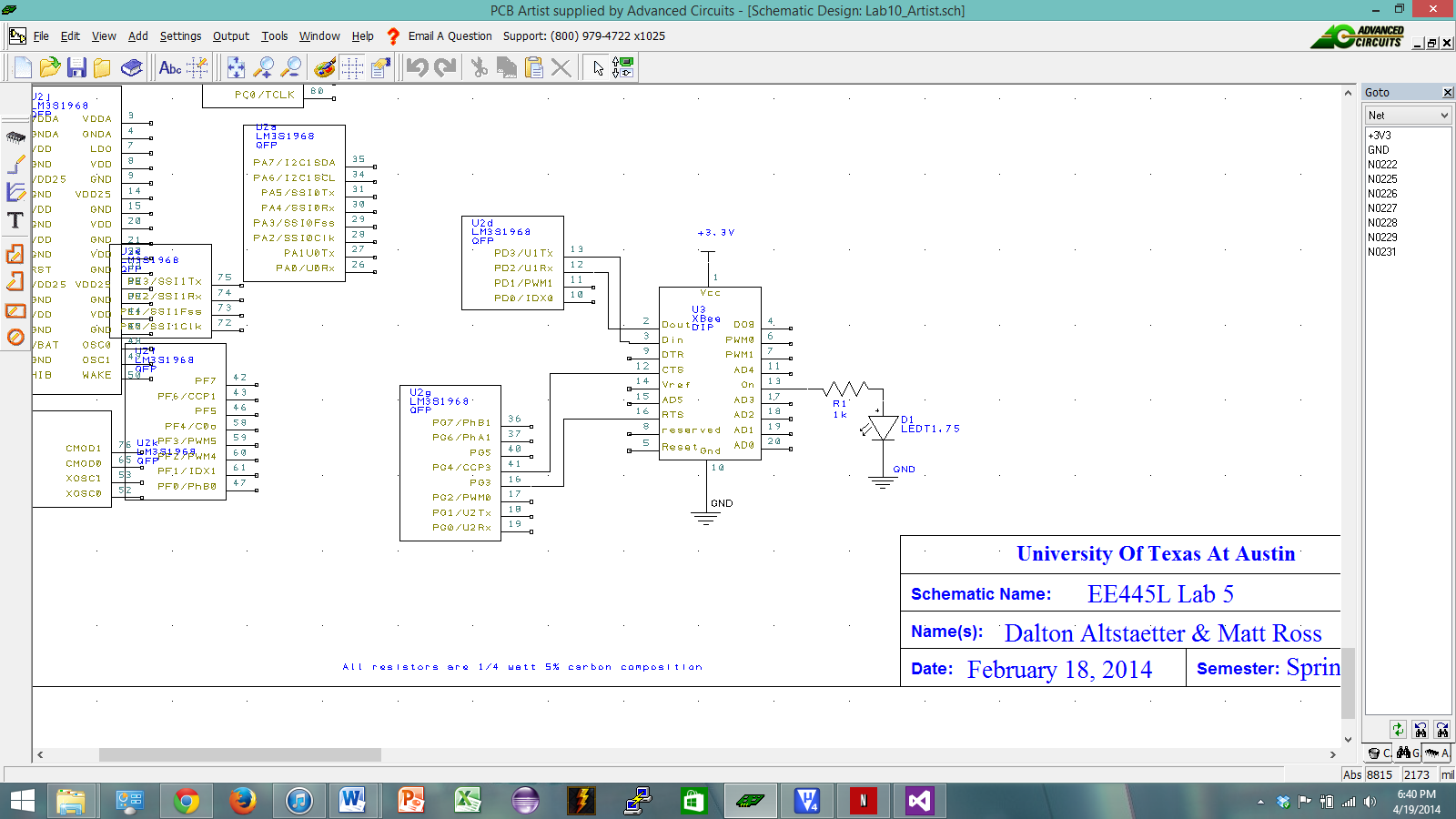
4/15/14

**OBJECTIVES**

The goals of this lab are to develop debugging techniques for transmitter and receiver systems. We will also learn how to use FIFO and other data structures to implement communications. Lastly, we will implement a text communication system between a PC and an OLED display using IEEE 802.15.4- ZigBee wireless module.

**HARWARE DESIGN**

SCH



**SOFTWARE DESIGN**

ZigBee

// XBee.c

#include "Xbee.h"

#include "systick.h"

#include "string.h"

#include "UART2.h"

static void sendATCommand(char\* input);

unsigned char destination[2] = {0x00,0x4F};

unsigned char startDelimiter = 0x7E;

unsigned char opt = 0x00;

unsigned short length;

unsigned short numBytes;

unsigned char checksum;

static void check(char\* r, int\* flagPtr)

{

int i;

char checkString[4] = "OK";

checkString[2] = CR;

for(i=0; i < 3; i++)

{

if(r[i] != checkString[i])

{

\*flagPtr = 1;

}

}

}

void XBee\_Init(void)

{

int flag;

char response[20];

flag = 0;

OutCRLF\_UART0();

UART1\_OutChar('X'); // send to XBee

UART0\_OutChar('X'); // echo to user

SysTick\_Wait10ms(110); // guard time delay

UART1\_OutString("+++"); // send to XBee for AT cmd mode

UART0\_OutString("+++"); OutCRLF\_UART0();// echo to user

UART1\_InString(&response[0], 5);

SysTick\_Wait10ms(110); // guard time delay

UART0\_OutString(&response[0]); OutCRLF\_UART0();

//check(&response[0], &flag);

if(strcmp(response,"OK")) //flag)

{

UART0\_OutString("Error entering AT Command Mode");

}

sendATCommand("ATDL50"); OutCRLF\_UART1(); // sets destination address to 79

sendATCommand("ATDL"); OutCRLF\_UART1(); // sets destination address to 79

//UART1\_InString(&response[0],20); UART0\_OutString(response);

sendATCommand("ATDH0"); OutCRLF\_UART1(); // sets destination high address to 0

sendATCommand("ATDH"); OutCRLF\_UART1(); // sets destination address to 79

//UART1\_InString(&response[0],20); UART0\_OutString(response);

sendATCommand("ATMY51"); OutCRLF\_UART1(); // sets my address to 78

sendATCommand("ATMY"); OutCRLF\_UART1(); // sets destination address to 79

//UART1\_InString(&response[0],20); UART0\_OutString(response);

sendATCommand("ATAP1"); OutCRLF\_UART1(); // set for API mode 1

sendATCommand("ATAP"); OutCRLF\_UART1(); // sets destination address to 79

sendATCommand("ATBD"); OutCRLF\_UART1(); // sets destination address to 79

sendATCommand("ATID"); OutCRLF\_UART1(); // sets destination address to 79

sendATCommand("ATCH"); OutCRLF\_UART1(); // sets destination address to 79

sendATCommand("ATCN"); OutCRLF\_UART1(); // ends the AT Command mode

OutCRLF\_UART0();

UART0\_OutString("Done with AT Cmd Mode"); OutCRLF\_UART0();

// also check ATBD == 3 to make sure the baud rate is set at 9600 bits/sec

//ATCH parameter<CR> changes the channel range between 0x0B and 0x1A (default 0x0C)

//ATID parameter<CR> changes the personal area network ID range between 0x0000 and 0xFFFF (default 0x3332)

}

//sendATCommand – sends an AT command repeatedly until it receives a reply that it was correctly received

//This routine receives the various parameters associated with an AT command as input then transmits the formatted

//command to the XBee module. After a blind-cycle delay, the routine checks if the command has been successfully

//received by determining if the module has returned the ‘OK’ character string.

static void sendATCommand(char\* input)

{

int flag;

char r[20];

flag = 0;

UART1\_OutString(input); // send at cmd

OutCRLF\_UART1();

SysTick\_Wait10ms(2); // delay

UART1\_InString(r,19); // get OK<CR> response

UART0\_OutString("Should get OK :");

// OutCRLF\_UART0();

if(!strcmp(r,"OK"))

{

UART0\_OutString(r);//"We got Correct Response");

OutCRLF\_UART0();

return;

}

UART0\_OutString(r);//"We got Correct Response");

OutCRLF\_UART0();

return;

check(r, &flag); // check for the OK<CR> response

while(flag)

{

flag = 0;

UART1\_OutString(input);

OutCRLF\_UART1();

SysTick\_Wait10ms(2);

UART1\_InString(r,19);

UART0\_OutString("Should get OK :");

UART0\_OutString(r); // think this get the OK<CR> response

check(r, &flag);

}

}

//XBee\_TxStatus – determine transmit status

//When the XBee module transmits an API transmit data frame it will receive an acknowledgement from the

//destination module if the frame was received without errors. The status of the transmission will be sent to the

//LM3S1968 via an API transmit status frame. This routine returns a ‘1’ if the transmission was successful and a ‘0’

//otherwise. The following figure shows a response the XBee returns after the transmitter sends a TxFrame that was

//properly received by the other computer, measured on XBee pin 2 Dout.

int XBee\_TxStatus(void)

{

//length + 4

char buffer[50];

int x, y, length, returnedChecksum, ok, checksum;

int id, dest\_high, dest\_low, success, api;

api = 0x01;

UART1\_InString(buffer,19); // need to change this to InChar, till CR, maybe just null char

//get byte values

id = buffer[4];

dest\_high = buffer[5];

dest\_low = buffer[6];

success = buffer[8];

//calculate length of message

x = buffer[1];

y = buffer[2];

length = (x << 8) + y;

//calculate check sum

checksum = 0xFF - (api + id + dest\_high + dest\_low + success);

//find returned checksum value

returnedChecksum = buffer[length + 3];

//find ok value

ok = buffer[length + 2];

//check ok and if the check sums match

if(ok == 0 && returnedChecksum == checksum)

{

return 1;

}

return 0;

}

//-------------------------------------------------------------------------------------------------

void XBee\_SendTxFrame(void)

{

int i;

char\* XbeeString;

char string[25] = {0};

UART0\_OutString("InString0: ");

UART0\_InString(&string[0],19); OutCRLF\_UART0();// get the message to send from the user

XbeeString = XBee\_CreateTxFrame(&string[0]); // create message

//UART1\_OutString(XbeeString); OutCRLF\_UART1(); // send to XBee Wireless

UART0\_OutString("Sent: ");

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

{

UART1\_OutChar(XbeeString[i]); // send to XBee Wirelessy

UART0\_OutChar(XbeeString[i]); // echo to user

}

OutCRLF\_UART1();

OutCRLF\_UART0();

//UART0\_OutString("Sent: "); UART0\_OutString(XbeeString); OutCRLF\_UART0(); // tell us what we sent

if(!XBee\_TxStatus())

{

UART0\_OutString("Error, acknowledge not received"); OutCRLF\_UART0();

}

else

{

UART0\_OutString("Message sent successfully"); OutCRLF\_UART0();

}

}

//-------------------------------------------------------------------------------------------------

char\* XBee\_CreateTxFrame(char\* string)

{

static unsigned char ID = 1;

int i;

static char message[25];

unsigned char sum;

checksum = 0;

// zero message buffer

for(i=0;i<25;i++)

{

message[i] = 0;

}

numBytes = 0;

i=0;

while(\*(string+i) != 0)

{

numBytes++;

i++;

}

length = numBytes+5; // 5 counts for the API, ID, Destination, & OPT bytes

//for(numBytes = 0; \*string != NULL; numBytes++){}

message[0] = startDelimiter;

message[1] = (unsigned char)((length & 0xFF00)>>8); // get top byte of length

message[2] = (unsigned char)(length & 0x00FF); // get lower byte of length

message[3] = 0x01; // API mode 1

message[4] = ID;

message[5] = destination[0];

message[6] = destination[1];

message[7] = opt;

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

{

// fill the message array

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

}

ID = (ID+1)%256; // keep in range of an unsigned char

if(ID == 0)

{

ID = 1; // make sure the ID never equals zero

}

sum = message[3]+message[4]+message[5]+message[6]+message[7];

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

{

sum += message[i+8];

}

checksum = 0xFF-sum;

message[numBytes+9] = checksum;

return &message[0];

}

//-------------------------------------------------------------------------------------------------

**MEASUREMENT DATA**

Estimated Maximum Bandwidth: 2MHz

Estimated range: 30m

**ANALYSIS AND DISCUSSION**

An important part of this lab was understanding how the UART works to communicate to the Zigbee and the PC. Then using the UART, we had to understand the communication protocols for communication to occur between the Zigbees. Calculating the checksum and making sure the protocols were followed precisely were the most difficult parts and required the most debugging.