UNIVERSITY OF CALIFORNIA, DAVIS
Department of Electrical and Computer Engineering
EEC 172 Spring 2014, Professor Kent Wilken

LAB 3: Local, Bi-directional Wireless IR Remote Texting

Verification (105 points):	Due May 14 or 15 at the end of your lab session.
	Late verification deadline May 21 (-15 points)
Lab code:	Due as a SmartSite upload May 21 at 9:00AM
Lab report (45 points):	Due as a hard copy in class on May 21 **(**Late lab report -10 points)

Objective:

The use of wireless embedded systems is increasing rapidly. An objective of this lab is to expose students to the use of an example wireless embedded-system module, XBee, which is based on the IEEE 802.15.4 standard, an RF standard widely used in wireless embedded systems.

A second objective is to introduce the Spark Core and its web-based integrated development environment. We will use the Spark Core in Lab 4 to connect the IR remote to the web.

New Hardware:

2 XBee 802.15.4 Series 1 RF modules. Warning: XBees operate on 3.3v not 5v! Connecting an XBee to 5v could destroy the part!!

2 XBee socket adapter boards. Converts tight pin spacing on XBee module to 0.1" pin spacing needed for the proto board.

2 LEDs and 220Ω resistors for monitoring XBee activity (optional)

1 AT&T RC1534801 Remote Control (in addition to the one from Lab 2)

1 Vishay TSOP1236 (a second copy of the device used in Lab 2)

1 100Ω resistor and 100μF capacitor for noise filtering on TSOP1236

1 Channel Number and 1 Personal Area Network (PAN) ID assigned by your TA to avoid interference in the lab between groups.

1	Spark Core
1	Nokia 5110 compatible display for the Spark Core
1	2" x 7" breadboard

Documentation:

XBee Product Manual. Posted on SmartSite and available at:
http://www.digi.com/support/getasset?fn=90000982&tp=3

The Spark Core API descriptions at: http://docs.spark.io/#/firmware

Part A: XBee Transparent Mode

The XBee RF module interfaces to a host device, such as the Stellaris LM3S8962 or the Spark Core, through an asynchronous serial port (UART). By default, a XBee module operates in *Transparent Mode* – all UART data received through the Data In (DIN) pin is queued for RF transmission and all RF data received is presented on the Data Out (DOUT) pin.

For XBee device configuration you will use the vendor's X-CTU utility, which runs on Windows. XCTU is installed on the lab PCs and is available for download from http://www.digi.com/support/productdetail?pid=3352&type=utilities. The X-CTU User's Guide is posted to SmartSite and can be found at ftp://ftp1.digi.com/support/documentation/90001003_a.pdf. Note that a new Next Generation X-CTU (version 6) has been recently released. The descriptions given here are for Legacy XCTU (version 5).

To use the X-CTU utility, program the LM3S8962 to relay serial data back and forth between the X-CTU program and an XBee module attached to the LM3S8962. You can modify the **`uart_echo`** example found in the StellarisWare folder (C:\StellarisWare\boards\ek-lm3s8962\uart_echo) to implement this relay.

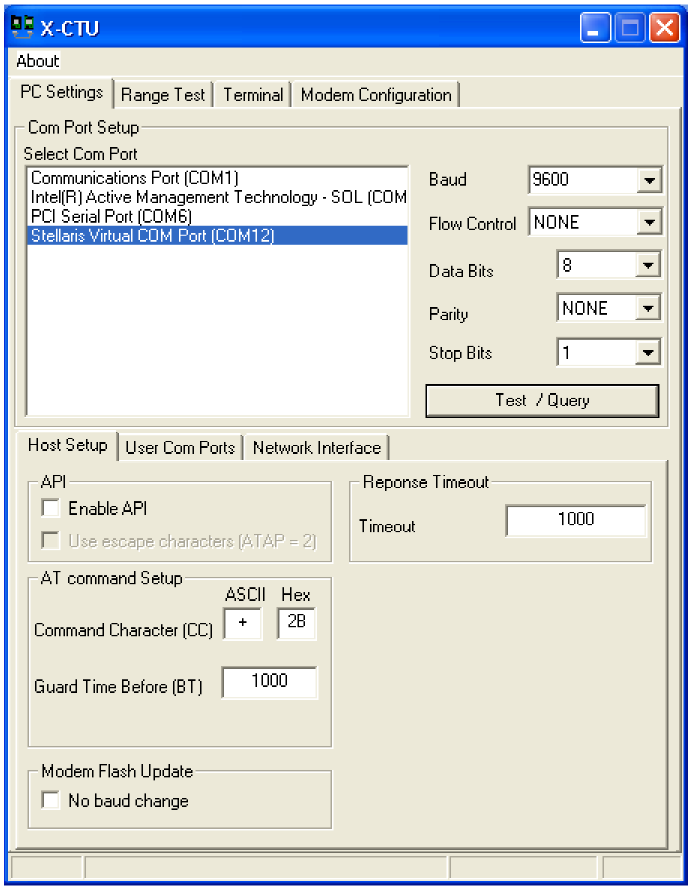
Your modified uart_echo program should:

1. The default baud rate for the XBee and for X-CTU is 9600 so you should configure both Stellaris UART ports (UART0 and UART1) to operate at 9600 baud, 8 data bits, no parity bit, 1 stop bit and no flow control.

2. When a character is received from the X-CTU program on UART0, the character should be sent to the XBee module on UART1. Similarly, when a character is received from the XBee on UART1, the character should be sent to the X-CTU program via UART0. An easy way to do this is to use character-receive interrupts for both UART0 and UART1.

3. We recommend that you use the UARTCharGet() and UARTCharPut() APIs for communicating with the UART module in this lab rather than the UARTCharGetNonBlocking() and UARTCharPutNonBlocking()APIs used in **`uart_echo`**. The Stellaris UART module contains a 16-character buffer. A call to UARTCharPutNonBlocking() will return immediately if the buffer is full with a FALSE return value, without inserting the character. A retry would be necessary in the calling procedure to avoid losing the character. In contrast, UARTCharPut() waits until buffer space is available and the character is inserted before returning. The buffer can become full on output because the microcontroller can put characters in the buffer much faster than they can be sent across the RF channel; especially at 9600 baud.

For the initial XBee test using X-CTU, select the Stellaris Virtual Com Port and set the serial communication parameters as shown in the window below. With your modified **uart_echo** program running on the LM3S8962 and your XBee powered up, click the **Test/Query** button.

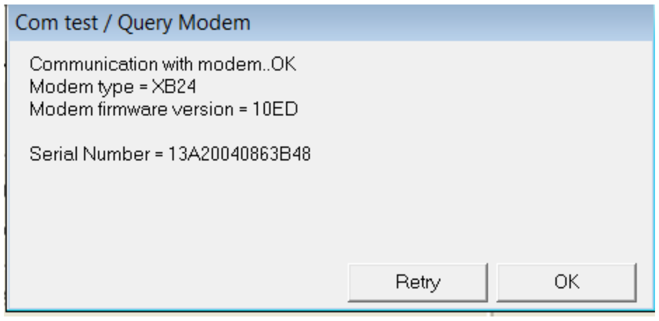


Communications Port (COM1)
Intel(R) Active Management Technology - SOL (COM
PCI Serial Port (COM6)
Stellaris Virtual COM Port (COM12)

Baud: 9600
Flow Control: NONE
Data Bits: 8
Parity: NONE
Stop Bits: 1

Enable API
Use escape characters (ATAP = 2)
Timeout: 1000
Command Character (CC): ASCII + Hex 2B
Guard Time Before (BT): 1000
No baud change

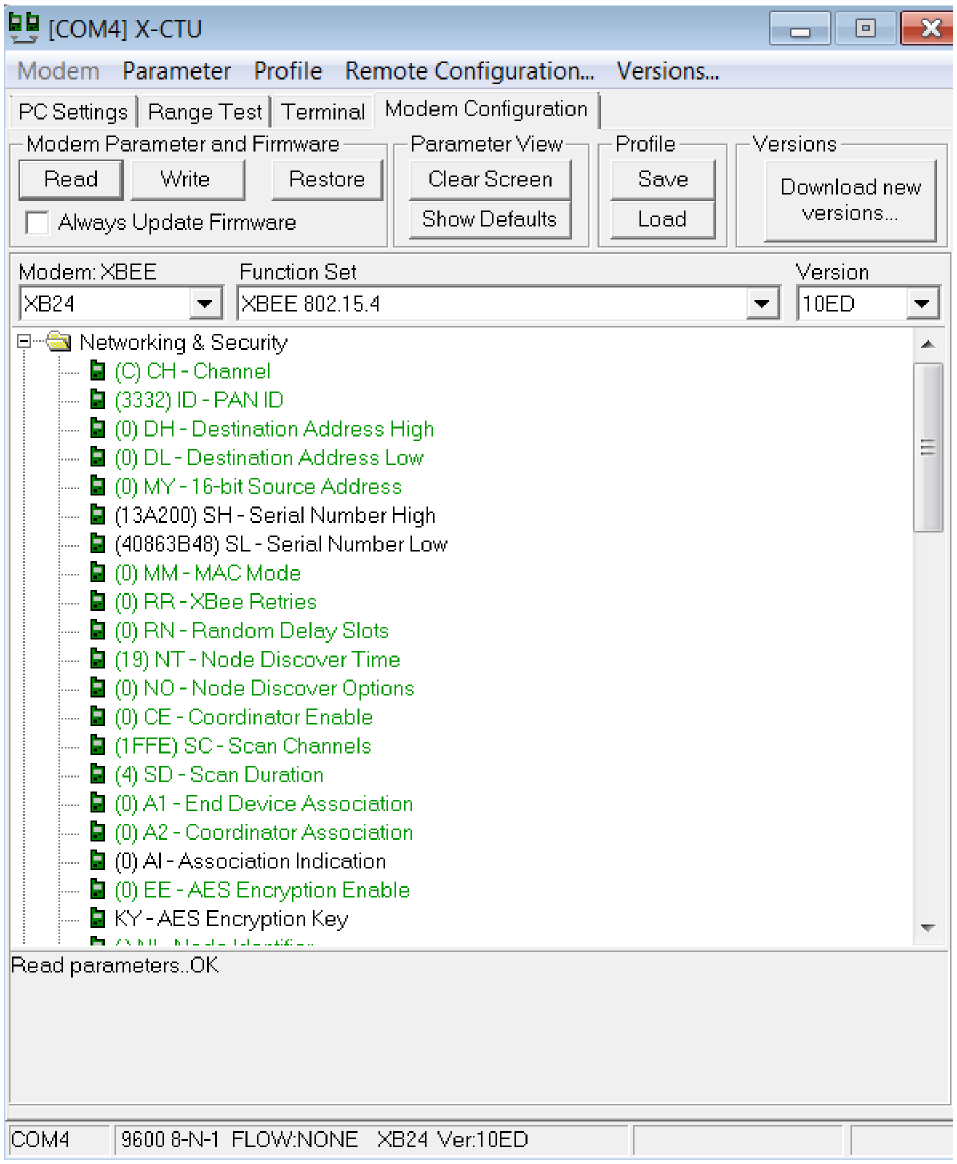
If the X-CTU program successfully communicates with the XBee module, you will see a status window with the XBee's serial number and firmware version, as shown below.



Communication with modem..OK
Modem type = XB24
Modem firmware version = 10ED

Serial Number = 13A20040863B48

Next click the **`Modem Configuration`** tab and then click on the **`Restore`** button. This will restore the default parameter values in the XBee's non-volatile memory. Click on the **`Read`** button and you should see the result shown below, except that your serial number will differ.


[COM4] X-CTU
Modem Parameter Profile Remote Configuration... Versions...
PC Settings Range Test Terminal Modem Configuration
Modem Parameter and Firmware
Read Write Restore
Always Update Firmware
Parameter View
Clear Screen
Show Defaults
Profile
Save
Load
Versions
Download new versions...

```
Modem: XBEE    Function Set       Version
XB24           XBEE 802.15.4      10ED

Networking & Security
    (C) CH - Channel
    (3332) ID - PAN ID
    (0) DH - Destination Address High
    (0) DL - Destination Address Low
    (0) MY - 16-bit Source Address
    (13A200) SH - Serial Number High
    (40863B48) SL - Serial Number Low
    (0) MM - MAC Mode
    (0) RR - XBee Retries
    (0) RN - Random Delay Slots
    (19) NT - Node Discover Time
    (0) NO - Node Discover Options
    (0) CE - Coordinator Enable
    (1FFE) SC - Scan Channels
    (4) SD - Scan Duration
    (0) A1 - End Device Association
    (0) A2 - Coordinator Association
    (0) AI - Association Indication
    (0) EE - AES Encryption Enable
    KY - AES Encryption Key

Read parameters..OK

COM4    9600 8-N-1  FLOW:NONE   XB24  Ver:10ED
```

If you get the message "The modem configuration file was not found. Would you like to check the web site for updates? (Recommended)" Click 'Yes' so that your PC has the configuration file that matches the latest firmware that is installed on the XBee.

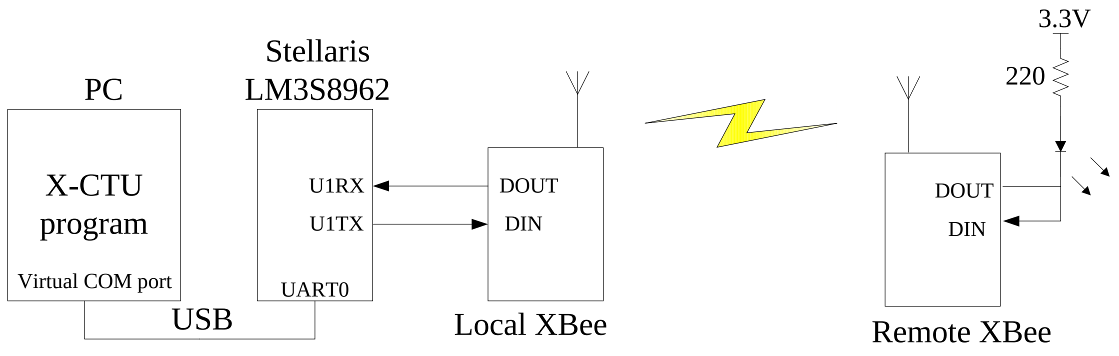
Your two XBee modules will communicate using a specific channel number and Personal Area Network (PAN) number assigned by your TA. Each group within a lab section will be assigned a different channel (RF frequency) to physically isolate communication between groups in that lab section. Each group will use a PAN that is unique across all lab sections to logically isolate communication between groups from different sections who happen to be using the same channel.

In XCTU click on the **`CH:Channel`** parameter and enter your group's assigned channel number. Then click on the **`ID:PAN ID`** to set your assigned PAN ID. Then click the **`Write`** button to save these changes to the XBee's non-volatile memory.

Next, follow the same procedure to set the assigned channel and PAN ID for your second XBee module.

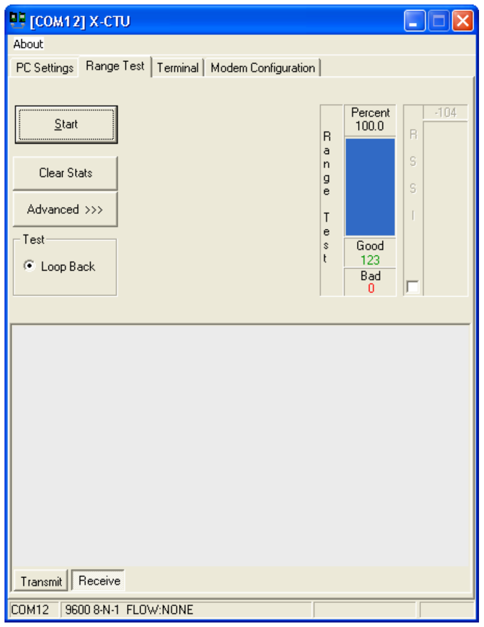
The second more substantial test of your XBee modules' functionality will be a loopback test, as shown in the figure below. The remote XBee should have its DOUT tied to it DIN pin so that any RF data received on DOUT is sent to DIN and then transmitted (echoed) back to the sender over the RF link. To verify that your remote

XBee is receiving data, it may be useful to connect an LED in series with a 220Ω resistor to the DOUT/DIN node on the remote XBee as shown in the figure below. A blinking LED will indicate data activity.

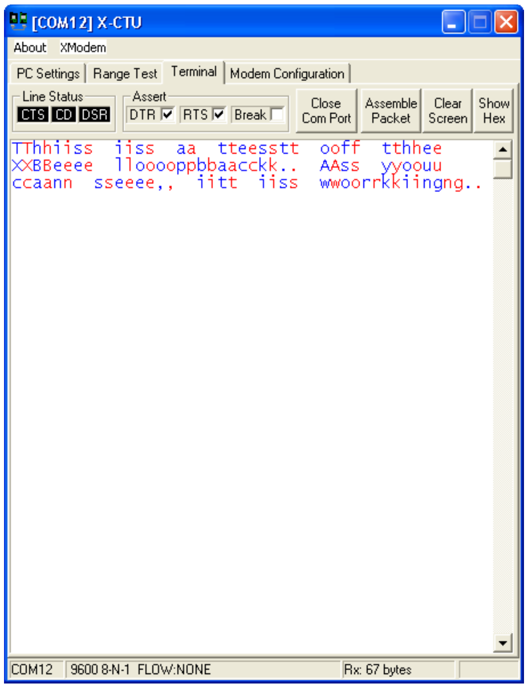


XBee Loopback test

Using X-CTU, click on the Range Test tab and then click on Start. Using the default transmission packet, you should see the number of successful tests count upward and remain 100% good as long as you run the test. For example, in the window below there were 123 good tests (packets transmitted and received) and no lost packets.



Another way to perform a loopback test is to use the terminal. Click on the terminal feature and type some characters. You should see the characters you type displayed in blue, and also the echoed character from the remote XBee in red. Thus, every character that you type should be displayed twice due to the echo from the remote XBee, as shown in the figure below. Notice that it is possible if characters are typed quickly that the characters will be transmitted as a group, and then the group of characters will be echoed by the remote XBee.



```
TThhiiss  iiss  aa  tteesstt  ooff  tthhee
XXBBeeee  llooooppbbaacckk..  AAss  yyoouu
ccaann  sseeee,,  iitt  iiss  wwoorrkkiingng..
```

```
COM12   9600 8-N-1  FLOW:NONE                Rx: 67 bytes
```

Using XCTU, demonstrate to your TA that you have correctly assigned your Channel number and PAN ID and can successfully loopback using the XCTU terminal.

Part B: Simple Wireless Communication From the LM3S8962 to the Spark Core

You will program the Spark Core to receive data from the LM3S8962 via the XBees connected to each microcontroller.

For the Spark Core, start by following the instructions at www.spark.io/start

After you have connected with the Spark Core, go to http://docs.spark.io/#/firmware to learn about the APIs the Spark Core uses for interacting with the hardware. Serial Communication will be used to interact with XBee.

Go to http://docs.spark.io/#/hardware to learn about the Spark Core hardware. As a UART device the XBee will be connected to the RX and TX pins.

Demonstrate the following to your TA:

– When a '1' character is typed in the XCTU terminal, the LM3S8962 will send the character to the Spark Core and the program running on the Spark Core will turn on the LED connected to pin D7.
– When the Spark Core receives a '0' character from its XBee, the LED is turned off.

Part C: Wireless IR Remote Texting from the Stellaris to the Spark Core

In this part of the lab you will send wireless text messages via the XBees from the Stellaris LM3S8962 to the Spark Core, displaying the messages locally on the Stellaris OLED display and remotely on the Spark Core display.

Messages will be sent as a packet. Each message will be formed character by character using the IR remote (as in Lab 2) and then sent when a delimiter button is pressed. The preferred delimiter button is **`ENTER`** , however **`ENTER`** is not used by some TV set up codes, in which case another button should be used.

You will use the XBee's API mode to send the text messages. API mode uses structured frames that include a start byte, length bytes, API-specific data, and a checksum, as described starting on p. 57 of the XBee Product Manual.

It will be necessary to use X-CTU to change some XBee settings relative the settings in Parts A and B:

1. For each XBee, set **`AP: API Enable`** to "1" to enable API mode.
2. Set **`My:16-bit Source Address`** to "1" for the XBee to be attached to the Spark Core. Leave **`My:16-bit Source Address`** as "0" for the XBee to be attached to the LM3S8962 processor.
3. Click **`Write`** to save the change for each XBee.
4. Click **`Read`** to confirm the changes.

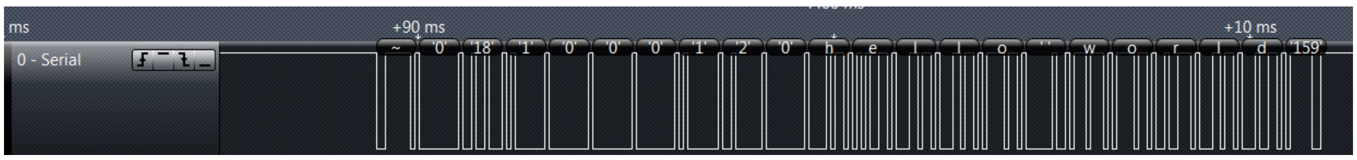
XBee API communication can be done using 64-bit addresses (the hard-wired serial number of each device) or 16-bit addresses, an assigned address. For simplicity we will use 16-bit addresses. In an adaptive wireless network 16-bit addresses might be assigned by a network coordinator node when a node joins the network. Here we are manually assigning the 16-bit addresses, which are the **`My:16-bit Source Address`** values set above.

The processors will use **`Transmit Request:16-bit address`** messages to send the text messages. This message format is described on page 62 of the XBee Product Manual.
For simplicity, we will only have one-way communication (no responses) so the Frame ID will be set to "0". The destination address will be 0x0001 (the **`My`** address assigned to the Spark Core above) or 0x0000 (the **`My`** address assigned to the LM3S8962) . For simplicity, we will also disable acknowledgements, so Options byte = 0x01. The **`RF Data`** field will hold the text message.

Each frame ends with a checksum. See http://www.digi.com/support/kbase/kbaseresultdetl?id=2206 for example checksum calculations.

The each processor will receive the packet in the format described under **`Receive Packet:16-bit address`**, shown on page 63 of the XBee Product Manual.

You should use the logic analyzer to confirm that the packets that are being sent and received are correct, using the Async Serial protocol analyzer that is built into the logic analyzer UI (see. page 11 of the Saleae User Guide). The protocol analyzer will show the printable ASCII characters that is contained in the packet, and the numeric value of bytes that are not printable characters, as illustrated below in the XBee packet that sends the message 'hello world'.


ms
+90 ms
+10 ms
0 - Serial
'~' '0' '18' '1' '0' '0' '0' '1' '2' '0' 'h' 'e' 'l' 'l' 'o' ' ' 'w' 'o' 'r' 'l' 'd' '159'

<u>Connecting the Nokia 5110 Display to the Spark Core</u>

We will communicate with the Nokia display using the driver posted on GitHub: (https://github.com/sparkfun/GraphicLCD_Nokia_5110).

The Nokia 5110 is connected to your Spark Core processor as shown in the following table:

Nokia 5110 Signal / Pin	Spark Core Signal
RST, pin 1	D6
CE, pin 2	D7
DC, pin 3	D5
Din, pin 4	D4
Clk, pin 5	D3
Vcc, pin 6	3V3
BL, pin 7	Not connected
Gnd, pin 8	GND

Table 1. Wiring Chart for Interfacing Nokia 5110 to Spark Core

This driver is using the Spark Core pins D3-D7 as general-purpose outputs (GPIO) rather than using the Spark Core's internal SPI peripheral.

Modifications to Driver Software

Two modifications to the driver software are needed for it to work well.

1. The contrast value should be set to 0xB8 in the LCDInit() function

```
LCDWrite(LCD_COMMAND, 0xB8); //Set LCD Vop (Contrast)
```

2. The \ character on line 97 incorrectly causes the next line to be a continuation of the end-of-line comment. The end-of-line comment needs to be removed or enclosed in full comment delimiters as shown below.

   Change
   ```
   ,{0x02, 0x04, 0x08, 0x10, 0x20} // 5c \
   ```
   to
   ```
   ,{0x02, 0x04, 0x08, 0x10, 0x20} /* 5c \ */
   ```

Demonstrate to your TA that you can received a text message on the Stellaris board and then forward and display the message on the Spark Core board. Use the logic analyzer to show that API frame that is sent.

Part D: Texting Directly to the Spark Core

In this part of the lab you will port your IR remote decoding code from the Stellaris microcontroller to the Spark Core so that you can send text messages directly to the Spark Core board and display the messages on the Nokia LCD. Under the Spark Core firmware documentation (http://docs.spark.io/#/firmware), under Other Functions, you will want to explore the Time and Interrupt APIs for use in decoding the IR remote pulses using the Spark Core. You will use the same IR remote code for the Spark Core as you did for the Stellaris.

Demonstrate to your TA that you can receive a text message from the IR remote using the Spark Core and can display the message locally.

Part E: Bi-Directional Texting

Demonstrate to your TA bi-directional texting between the Stellaris and Spark Core boards. On each display show the messages sent and received in a manner that they can be distinguished, e.g., in a different intensity (greyscale), in all caps vs. all lower case, etc.

Lab Report:

At a minimum your lab report should include:

- Your verification sheet.
- A top-level description of the project and how it is solved.
- A top-level description of your code, its organization and its operation.
- A hard copy of your well commented code.
- Logic analyzer screen shots of a sent message and the corresponding received message, with the packet fields labeled.
- A description of any noteworthy difficulties you encountered in constructioning your project.
- A description of anything about your lab this is interesting, insightful, amazing or amusing that might be warrant bonus points.