Lab 10 Wireless Serial Communication

This laboratory assignment accompanies the book, Embedded Systems: Real-Time Interfacing to ARM Cortex M Microcontrollers, ISBN-13: 978-1463590154, by Jonathan W. Valvano, copyright © 2012.

Goals

- Develop debugging techniques for transmitter (Tx) and receiver (Rx) systems,
- Use FIFOs and other data structures to implement communication,
- Implement a text communication system between a PC and an OLED display via an IEEE 802.15.4 – **ZigBee** wireless module.

Review

- Valvano Section 3.7 FIFO Queues
- Valvano Sections 3.4.4, 4.9, and 5.7 UART
- Valvano Chapter 11 on communication systems
- http://users.ece.utexas.edu/~valvano/EE345L/Labs/Fall2011/XBeeManual.pdf
- http://users.ece.utexas.edu/~valvano/EE345L/Labs/Fall2011/Zigbeeinfo.pdf

Starter files

- **UART2_1968.zip** – serial communication
- **Your Lab1 OLED driver**.

Background

ZigBee is the commercial name given to devices that communicate wirelessly employing the IEEE 802.15.4 standard. A **wireless personal area network** (WPAN) is a computer network used for communication among computer devices close to one person or one device. The This standard is intended to provide relatively slow, i.e., less than 250 kb/s, low power, short range, i.e., 10 to 75m, and low cost wireless communication for remote control and process automation applications. ZigBee components offer long battery life and the ability to operate in large ad hoc mesh networks. They typically operate in the 2.4 GHz Industrial-Scientific-Medical (ISM) unlicensed band that is shared with Wi-Fi (IEEE 802.11) and Bluetooth (IEEE 802.15.1) communications. The IEEE 802.15.4 standard describes how information is represented via radio frequency signals, i.e., the **physical** or **PHY** layer, and how communications are formatted and controlled, i.e., the **media access control** or **MAC** layer. Communication requirements beyond the PHY and MAC, e.g., flow control, error recovery, and routing, remain to be implemented within the application employing the ZigBee communication.

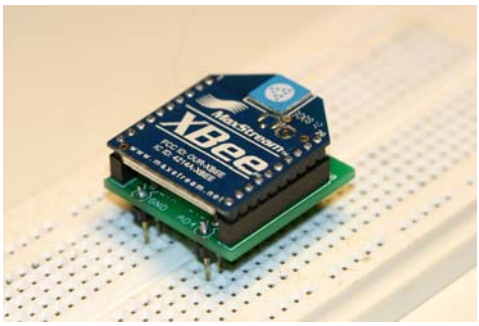

Figure 10.1. The MaxStream XBee module employs ZigBee to communicate serial data.

Ordering Information www.sparkfun.com

XBee 1mW ZigBee Module,	Sparkfun No. WRL-08664,	$22.95
XBee Module Breakout Board,	Sparkfun No. BOB-08276,	$2.95
Two 2mm 10 pin XBee Sockets,	Sparkfun No. PRT-08272,	$1.00*2
0.1" 40 pin Break Away Headers,	Sparkfun No. PRT-00116,	$2.50

Specifications

This lab will implement simplex serial data communication from a PC, running HyperTerminal or PuTTY, connected through a Tx system to an Rx system that displays the data on an OLED module. The Tx system will communicate with the Rx system via a ZigBee wireless link. Two partnerships will implement the serial communication link; one partnership will construct the Tx system and the other will construct the Rx system. The flow of data is as follows

1. Person types ASCII characters into HyperTerminal (or PuTTY);
2. The PC (via its COM) sends data to the UART0 of the Tx LM3S1968 through the USB cable;
3. `UART0_InChar` (using interrupt synchronization) of the main program of Tx system;
4. The main program in the Tx formats a transmit message as an API 1 frame (`XBee_CreateTxFrame`);
5. The main program in the Tx sends it to XBee using `UART1_OutChar` (`XBee_SendTxFrame`);

6. XBee passes the frame to the other XBee wirelessly;
7. XBee sends data to UART1 receiver;
8. **`UART1ISR`** receives the frame, putting it into a FIFO;
9. **`UART1_InChar`** collects the incoming data, getting it from the FIFO;
10. The main program in the receiver unpacks message from API 1 frame;
11. The main program in the receiver display on local OLED.

The basic idea is the ASCII characters on a PC are transmitted to an OLED via the XBee channel. The UART module must be written at a low level, like the book, without calling StellarisWare driver code. Other code (OLED, GPIO, timer, and PLL) can use StellarisWare driver code.

Interfaces

The Rx system consists of a LM3S1968, an XBee module and the OLED display. The Tx system consists of a LM3S1968 and an XBee module. The Rx and Tx systems involve providing 3.3V power for the XBee module and interfacing it to the LM3S1968. Both the Rx and Tx systems will communicate serial data between their LM3S1968 and XBee module. The default baud rate of the XBee is 9600 bits/sec. The default format is 1 start bit, 8 data bits, no parity bit, and 1 stop bit. The Tx system will use UART0 to communicate with the PC and UART1 to communicate with the XBee. The Rx system will employ UART1 to communicate with the XBee.

XBee Module Driver

1. Public Functions

`XBee_Init` – initialize the XBee module

The XBee module is controlled by a series of 'AT' commands. (This command set was originally developed to control asynchronous telephone modems and was developed by the Hayes Company. The AT prefix was an abbreviation for 'ATtention.') This routine sets the 16-bit module identifier of both the transmitter and receiver modules using the 'MY' and 'DL' AT command suffixes. This routine also conditions the module to operate in **Application Programming Interface** (API) mode 1. When operating in this mode, the module accepts data to be transmitted as structured frames rather than as a transparent serial communication device and it permits to the transmitter to automatically be notified if the receiver correctly received the transmission. In this protocol, all message characters are ASCII characters. E.g., A is letter 'A' or $41. Numbers are also in ASCII, as hex numbers. For example the number 79 is $4F and transmitted as ASCII '4' ($34), ASCII 'F' ($46). The ASCII character <CR> is the carriage return (13 or $0D) and is used to terminate an AT command. The space is optional between the command and the parameter.

LM3S to XBee	XBee response to LM3S	Meaning
`X` wait 1.1s **`+++`** wait 1.1s	**`OK<CR>`**	Enter command mode
`ATDL4F<CR>` wait 20ms	**`OK<CR>`**	Sets destination address to 79
`ATDH0<CR>` wait 20ms	**`OK<CR>`**	Sets destination high address to 0
`ATMY4E<CR>` wait 20ms	**`OK<CR>`**	Sets my address to 78
`ATAP1<CR>` wait 20ms	**`OK<CR>`**	API mode 1 (sends/receive packets)
`ATCN<CR>` wait 20ms	**`OK<CR>`**	Ends command mode

Some of the default parameters are channel (CH=12), PAN (ID= 0x3332 or 13106) destination high address (DH=0), and baud rate (BD=3, for 9600 bits/sec)

`XBee_CreateTxFrame` – create an API frame

This routine creates an API transmit data frame consisting of a *start delimiter*, *frame length*, *frame data*, and *checksum* fields. The *frame data* field contains *destination address* and transmission options information. Increment the Frame Id (byte 5 in the figure at the bottom page 57) from 1 to 255, and then back to 1 again.

`XBee_SendTxFrame` – send an API frame

This routine transmits the API transmit data frame to the XBee module via UART1. The following figure shows a TxFrame message measured at the input of the XBee, pin 3 Din. The checksum is the bottom 8 bits of the calculation $FF-($01+$FD+$00+$02+$00+$48+$65) = $52.

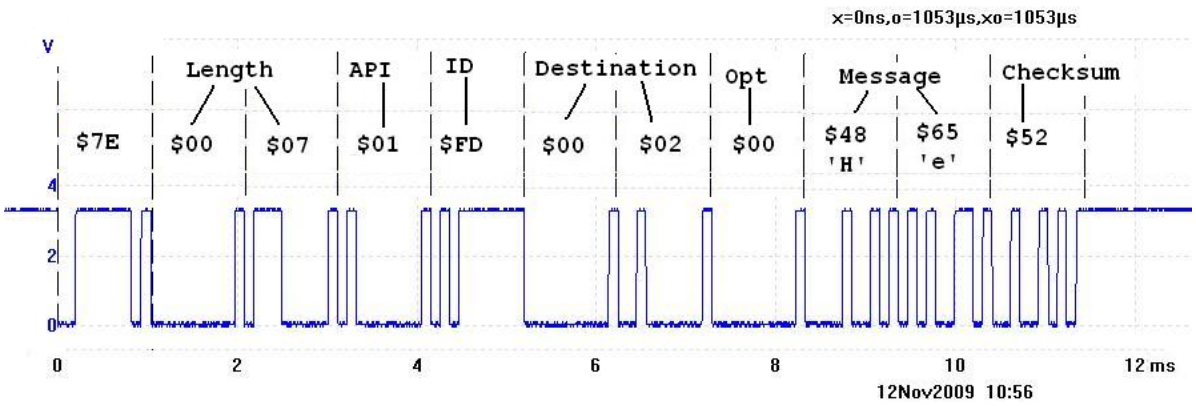

Figure 10.2. Transmit API frame type $01 used to send data.

`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.

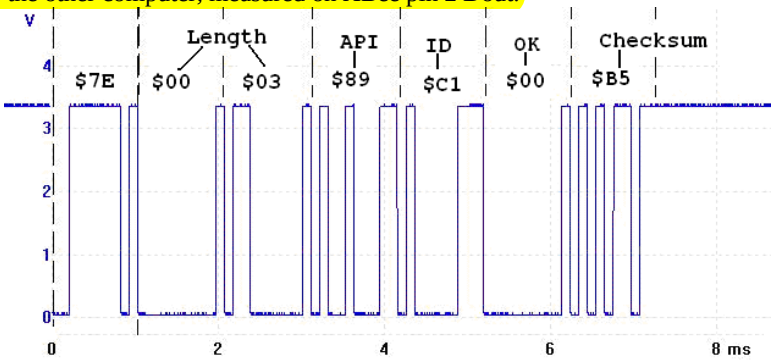

Figure 10.3. Transmit status API frame type $89 used to acknowledge message.

2. Private Functions

`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.

Tx System Design

The Tx system initializes communication with UART0 to receive serial data from the PC. It also initializes UART1 to enable communication with the XBee module. Data from the PC is stored in the FIFO. (The FIFO is required because the HyperTerminal utility will be used to communicate serial data between the PC and the Tx LM3S1968. HyperTerminal can take inputs from both the PC keyboard and files. The data rate between the PC and the Tx system will typically be much larger than that between the Tx LM3S1968 and the XBee module.) The Tx LM3S1968 will continually poll the Tx FIFO and transmits any available data to the XBee module.

Rx System Design

The Rx LM3S1968 initializes UART1, the XBee module, and the OLED display. Data from the XBee module is stored in the Rx FIFO. The main program on the Rx LM3S1968 continually polls the Rx FIFO and any available data is displayed on the OLED using the OLED driver routines.

`XBee_ReceiveRxFrame` – accepts messages into the receiver

When the XBee module receives an API data frame it sends a message to the LM3S1968. The following figure shows one such message frame as measured at XBee pin 2 Dout. RSSI is the signal strength.

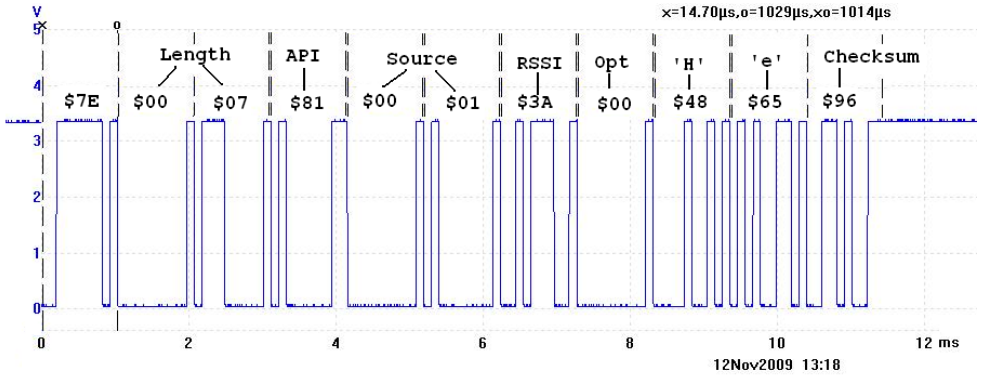

Figure 10.4. Receive API frame type $81 used to receive data.

Preparation

Part a) Produce a schematic for your system, e.g., Rx system or Tx system. Your schematic should illustrate all connections; have labels for all chips and their pin numbers, and values for all passive components.

Part b) Develop the **XBee.h** and **XBee.c** device driver files.

Part c) Write simple main programs you can use to test the device.

Procedure

Part a) Connect the XBee module to the LM3S1968. When not plugged into a protoboard, keep the part plugged into the pink foam.

Part b) Debug the system in a bottom up manner. Write simple main programs in the Rx and Tx systems to test the low-level functionality of the interface. Add appropriate debugging instruments such as profiles and dumps. Connect unused pins from both devices to a single logic analyzer and record a thread profile (which program runs when) as a stream of data is passed from the Tx system to the Rx system.

Part c) Write a main program that implements either the transmitter or the receiver.

Part d) The goal in this section is to measure maximum bandwidth of your channel without using hardware flow control in the PC COM port. Furthermore, your goal is to determine which component limits bandwidth. We know the human typing, the human reading and the OLED display are slow, and so we will eliminate these three components. You will use the **Transfer->SendTextFile** command in HyperTerminal to send a long sequence with a simple pattern of characters. Modify the receiver so it does not display data on the OLED. Rather, the receiver will check for this pattern of characters and count the number of errors. You can adjust the baud rate between the PC and the Tx system to set the target bandwidth. Add minimally intrusive debugging instruments to determine if and where data is lost.

Part e) The goal in this section is to measure maximum range of your system. Using the same test procedure developed in part c) find the maximum distance where the system performs reliable communication.

Deliverables (exact components of the lab report)

A) Objectives (1/2 page maximum)
B) Hardware Design
   Rx or Tx schematic
C) Software Design (a hardcopy software printout is due at the time of demonstration)
   Hardcopy of your Rx or Tx system software
D) Measurement Data
   Estimate the maximum bandwidth of your XBee wireless link.
   Estimate the range of your XBee wireless link.
E) Analysis and Discussion (1 page maximum)

Checkout

Demonstrate that data from both the PC's keyboard and files can be communicated to the OLED display.
Upload your software and lab manual, as instructed by your TA.

Hints

0) If the XBee stops operating, I suggest you power-cycle the module (turn off power, then restore power). A particular problem occurs when the software running on the receiver microcontroller is reset while the rest of the system is still running. When the software attempts to initialize the XBee, the XBee should still be receiving data from the transmitter, since of course the transmitter system is unaffected by the receiver microcontroller being reset. The receiver software is looking for a short string ending with the characters "OK\r" in response to the "X" and "+++" commands. However, the software in the receiver will hang at this point if the incoming data is text from the transmitter. To fix this, either briefly shut off the transmitter system until the receiver system initializes correctly or power-cycle the receiver.

1) Adapt code from the UART2 (UART2 means interrupt driven and does not refer to the specific UART port on the microcontroller) starter files to implement the required UART0 and UART1 interfaces.

2) Be careful to flush all data out of the XBee when receiving messages and receiving responses. One way to flush the XBee is to read all the characters that exist until there is no more data from the XBee for more than 12 bit times.

3) The following TxFrame was sent from the Transmitter system to the XBee. It was ignored by the XBee. Why?

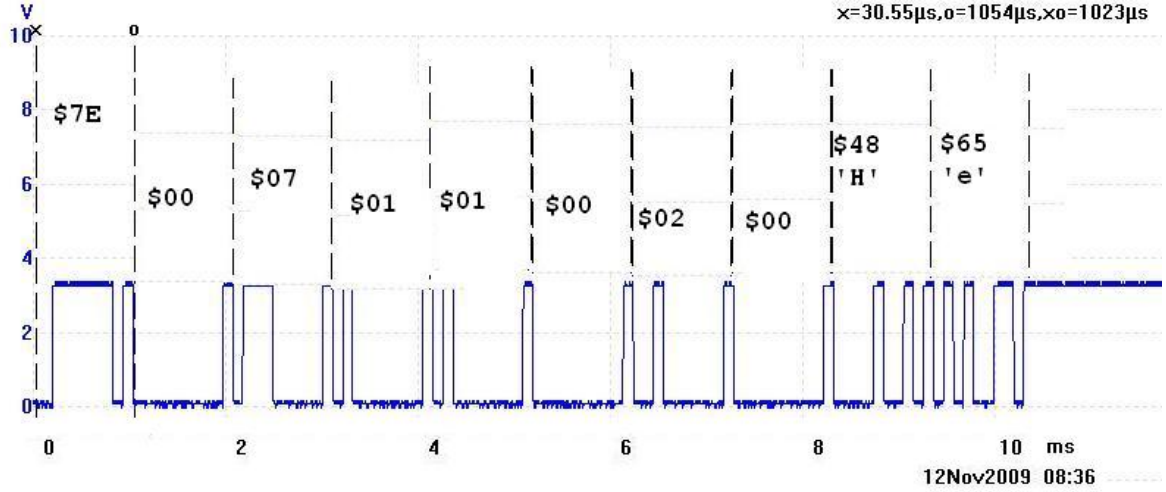

Figure 10.5. A bad transmit API frame.