Lab 10g Wireless Serial Communication

This laboratory assignment accompanies the book, Embedded Microcomputer Systems: Real Time Interfacing, Third edition, by Jonathan W. Valvano, published by Cengage, copyright © 2010.

Goals

- Develop an interface between embedded system components operating at different voltage levels,
- Develop debugging techniques for transmitter (Tx)/receiver(Rx) systems,
- Adapt a **virtual port** to provide serial communication with a wireless module,
- Implement a text communication system between a PC and a previously used LCD display via an IEEE 802.15.4 – **ZigBee** wireless module.

Review

- Valvano Section 4.3 FIFO Queues,
- Valvano Sections 6.1-2 Input Capture and Output Compare,
- Valvano Section 7.5 Serial Communications Interface
- http://users.ece.utexas.edu/~valvano/EE345L/Labs/Fall2010/STML4931CZ33.pdf
- http://users.ece.utexas.edu/~valvano/Datasheets/LM317LZ.pdf
- http://users.ece.utexas.edu/~valvano/Datasheets/7407.pdf
- http://users.ece.utexas.edu/~valvano/EE345L/Labs/Fall2010/XBeeManual.pdf
- http://users.ece.utexas.edu/~valvano/EE345L/Labs/Fall2010/Zigbeeinfo.pdf
- **Starter files:**
  - **SCIB_DP512.zip** – serial communication using input capture and output compare interrupts,
  - **SCIA_DP512.zip** – FIFO queues with a interrupting SCI0 interface,
  - **SCI1_DP512.zip** – Busy-wait SCI1 interface,
  - **Your Lab3 LCD driver** – 4-bit interface for HD44780-based LCD display.

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.

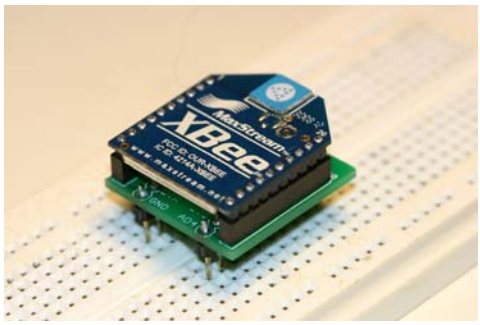

Figure 10.1. The MaxStream XBee module employs ZigBee to communicate serial data.

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
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, connected through a Tx system to an Rx system that displays the data on an LCD 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

Person types ASCII characters into HyperTerminal (or PuTTY);
The PC (via its COM) sends it to the SCI1 of the Tx 9S12 using the special cable (RS232 voltages);
`SCI1_InChar` (using interrupt synchronization) of the main program of transmitter 9S12;
The main program in the Tx formats a transmit message as an API 1 frame (**`XBee_CreateTxFrame`**);
The main program in the Tx sends it to XBee using **`SCIb_OutChar`** (**`XBee_SendTxFrame`**);
SCIb uses output compare PT3 to XBee through the 7407 level shifter;
XBee passes the frame to the other XBee wirelessly;
XBee sends data through 7407 level shifter, using input capture PT2 and output compare 4;
`SCIbISR` receives the frame, putting it into a FIFO;
`SCIb_InChar` collects the incoming data, getting it from the FIFO;
The main program in the receiver unpacks message from API 1 frame;
The main program in the receiver display on local LCD.

The basic idea is the ASCII characters on a PC are transmitted to a LCD via the XBee channel.

Interfaces

The Rx system consists of a DP512, an XBee module and an HD44780 LCD display. The Tx system consists of a DP512 and an XBee module. The Rx and Tx systems involve providing 3.3V power for the XBee module and interfacing it to the 5V DP512. The 3.3 V regulated power is developed via a LM317L. The LM317L regulator can be powered from the DP512 5V supply as long as the total current in the system is less than 100 mA.

Both the Rx and Tx systems will communicate serial data between their DP512 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. Both SCI0 and SCI1 on the Tech Arts DP512 board are already interfaced to RS-232 voltage level convertors for their serial data signals. Rather than using hardware to adapt these signals to the XBee modules, two pins of Port T will be employed to implement an interrupt driven **virtual port**. The **virtual port** uses two input capture interrupts (C2F C3F), one output compare interrupt (C4F) and two pins of a Port T (PT2 PT3) to simulate the operation of a hardware serial data transmitter/receiver. (Using the **virtual port** also frees SCI0 for use with the serial monitor and SCI1 for communication with the PC.)

The **virtual port** voltage level conversion required for both the Rx data – the XBee module pin 2 DOUT – and the Tx data – the XBee module pin 3 DIN – is accomplished using SN7407 open-collector buffers powered from the 5 V supply. (The SN7407 open-collector outputs require external pull-up resistors to pull the output to a high level. The XBee module and the DP512 provide configurable pull-up resistors on their inputs. The XBee module has its pull-up on DIN enabled at reset. The DP512 disables the pull-ups for pins on Port T and requires that software enable them.)

The Tx system will use SCI1 to communicate with the PC. (SCI0 remains available for the serial monitor). The Rx system will employ the HD44780 LCD display as developed in Lab 3.

There is a special cable to use between the PC and the 9S12 transmitter. To test the cable, download and run this starter file. http://users.ece.utexas.edu/~valvano/Starterfiles/SCI1_DP512.zip

1 RxD ———— black———— 3 TxD
2 Ground ————green ———— 5 Ground
3 TxD ———— yellow———— 2 RxD
4 not connected——red

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 is optional.

9S12 to XBee	XBee response to 9S12	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, then back to 1 again.

`XBee_SendTxFrame` – send an API frame

This routine transmits the API transmit data frame to the XBee module via the **virtual port**. 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.

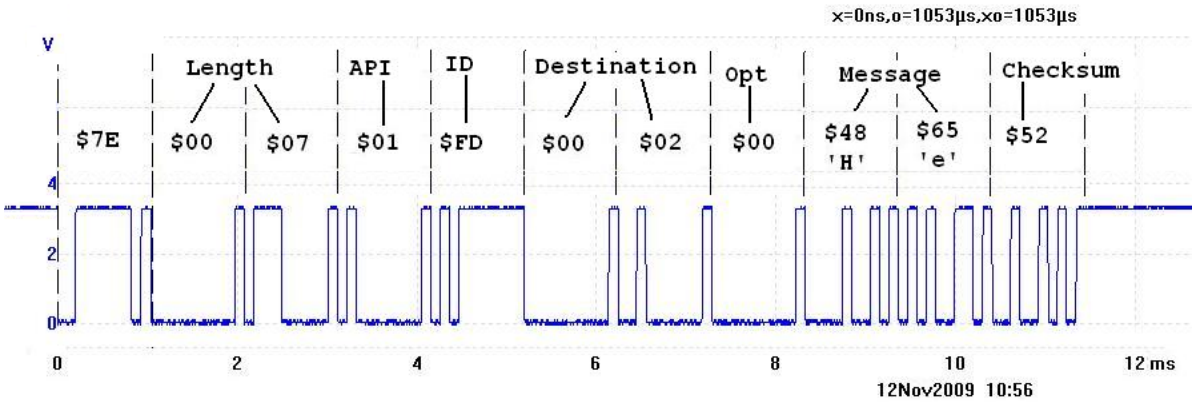

Figure 10.2. Transmit API frame type $01 used to send data.

`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 DP512 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.

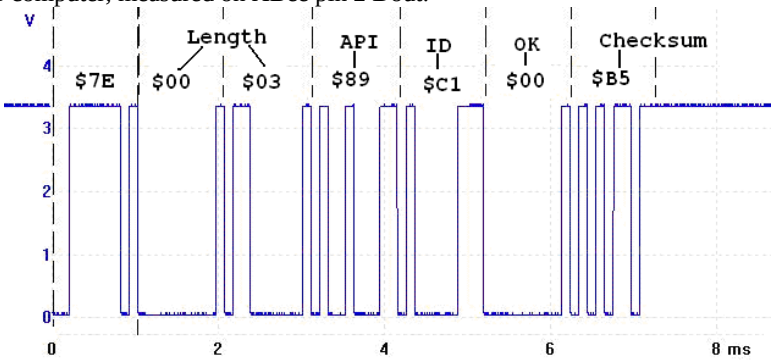

Figure 10.3. Transmit status API frame type $89 used to acknowledge message.

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 DP512 initializes communication with SCI1 to receive serial data from the PC. It also initializes the **virtual port** to enable communication with the XBee module. Data from the PC is stored in the Tx DP512's Tx FIFO. (The FIFO is required because the HyperTerminal utility will be used to communicate serial data between the PC and the Tx system. 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 DP512 and the XBee module.) The Tx DP512 will continually poll the Tx FIFO and transmits any available data to the XBee module.

Rx System Design

The Rx DP512 initializes the **virtual port**, the XBee module, and the LCD display. Data from the XBee module is stored in the Rx FIFO. The main program on the Rx DP512 continually polls the Rx FIFO and any available data is displayed on the LCD using the LCD driver routines.

`XBee_ReceiveRxFrame` – accepts messages into the receiver
When the XBee module receives an API data frame it sends a message to the 9S12. The following figure shows one such message frame as measured at XBee pin 2 Dout. RSSI is the signal strength.

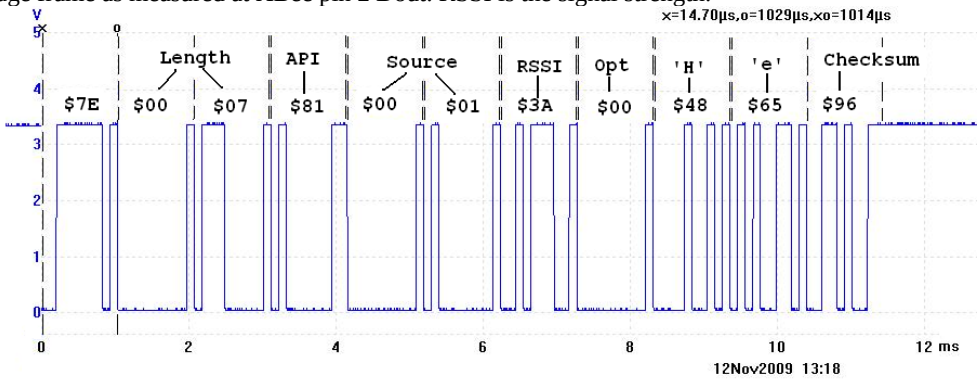

Figure 10.4. Receive API frame type $81 used to receive data.

Preparation

Part a) Produce a schematic for your partnerships 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) Build the physical ZigBee module as instructed by your TA.

Step 1) Place the two female sockets into the top of the PCB, turn it over and set the three parts down on a flat surface. Take special care to align the sockets at 90 degrees to the PCB. Solder the small female sockets from the bottom of the PCB board. Make one or two solder connections on each socket, and flip it over and verify sockets are at 90 degrees, before soldering the remaining connections.

Step 2) Place the two male-male headers (0.1in) into the bottom of the PCB and plug the combination into a solder-less protoboard. The two headers should be snuggly inserted into the protoboard and the PCB should lay parallel to the protoboard. Solder the male-male headers from the top of the PCB board.

Step 3) Gently insert the ZBee module into the small female sockets.

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. Furthermore, your goal is to determine which component limits bandwidth. We know the human typing, the human reading and the LCD 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 LCD. 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 LCD display.
A hardcopy printout of your software will be given to your TA, and graded for style at a later time.

Hints

1) Adapt code from the starter files to implement the required **virtual port**, FIFOs, and LCD driver.
2) Interface an LED to the XBee module status indicator pin, pin 13, to determine if the module is operating.
3) The following TxFrame was sent from the Transmitter 9S12 to the XBee. It was ignored by the XBee. Why?

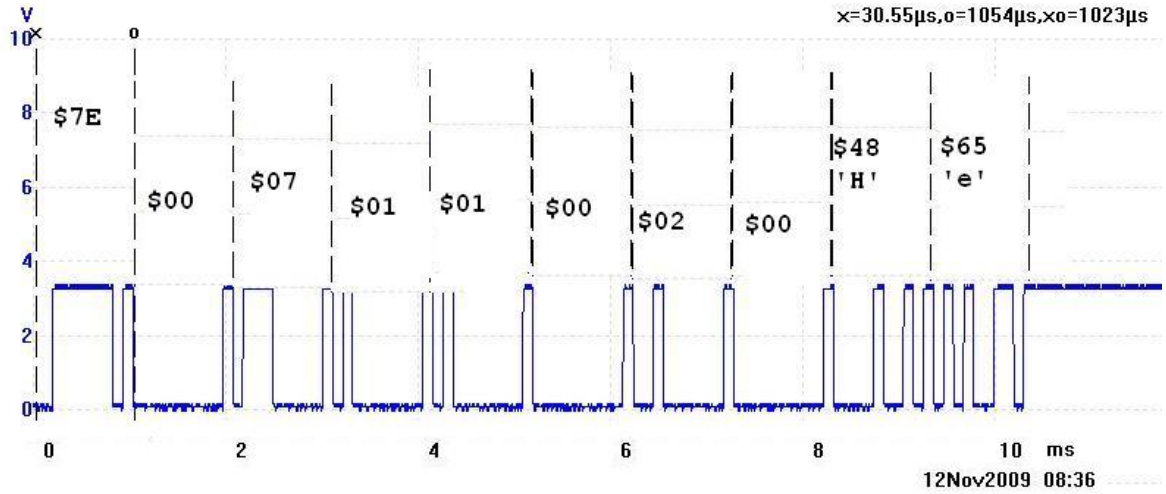

Figure 10.5. A bad transmit API frame.