Tanmay Sane
Sean Hicks
Advanced Embedded – Lab3

Serial Communication Using XBee

Objective: This lab exercise provides an introduction to XBee communication using theRenesas RX63N development board. It aims to establish communication between the RX63N board and an XBee coordinator node to transmit a character and receive a response based on a hidden cipher(Enigma). Based on send/receive pairs, the cipher is then broken.

Requirements:

The requirements were communicated by the course instructor and are listed here for reference:

1. The code generated must be written in C for the RX63N Evaluation Board.
2. The code must be well commented and easy to follow.
3. A single data byte payload should be sent from your board to another board using an XBee module.
4. For every message sent to the other board, you will receive a single message with a single data byte in return.
5. You data byte payload should be a value between 0x20 and 0x7E (Character ‘space’ to ‘~’).
6. Show the entered code on the top row of the LCD. Before any switch is pressed, show the character 'space' (0x20).
7. All characters from 0x20 (‘space’) to 0x7F (‘~’) are used.
8. When SW1 of the board is pressed, the displayed character "increments" by one, i.e. 'A' increases to 'B'.
9. When SW2 of the board is pressed, the displayed character "decrements" by one, i.e. 'Z' increases to 'Y'.
10. The incrementing of the character will roll over from ' ~ ' to ' space '.
11. The decrementing of the character will roll over from ' space ' to ' ~ '.
12. When SW3 of the board is pressed, the current character should be sent via the XBee module to the other board.
13. When the character code is sent from a board, it should NOT disappear from the top row of the LCD.
14. When the returned character code is received from the other board, it should be displayed on the second line of the LCD.
15. Packets sent through the XBee should be constructed using the “Transmit Request” API Type. (See XBee Series 2 documentation for details on the Transmit Request API type)
16. The boards should run without HEW running on a PC.
17. Complete your Lab Report.
18. Bring the new board to the lab TA and demonstrate the new code (without the HEW application running). When the TA checks your board, he will also take your lab report. You will not need to include a printout or soft copy all of the code – just “snippets”.

Tanmay Sane
Sean Hicks
Advanced Embedded – Lab3

Background:
This lab demands knowledge of basic UART functionality and familiarization with XBee protocol.

General steps:

1. Obtain the XBee network parameters necessary to communicate with the coordinator.
2. Flash the appropriate XBee firmware.
3. Test XBee operation using an alternate application, if available.
4. Review documentation for the RX63N to determine initialization requirements for the UART to communicate using the application header on its evaluation board.
5. Port the switch functions from Lab 2 to meet the new requirements.
6. Test and debug the switch functions.
7. Test and debug the UART functionality.
8. Connect the XBee to the RX63N application header and execute, debugging as necessary.
9. Optionally, decode the cipher and report the equation to the course instructor.

Detailed Steps:

1. The XBee communicates at 9600 baud with 16-bit Network Address 0 (coordinator) on PAN ID 2222. Most of the pertinent modem configuration settings are shown in Figure 1.

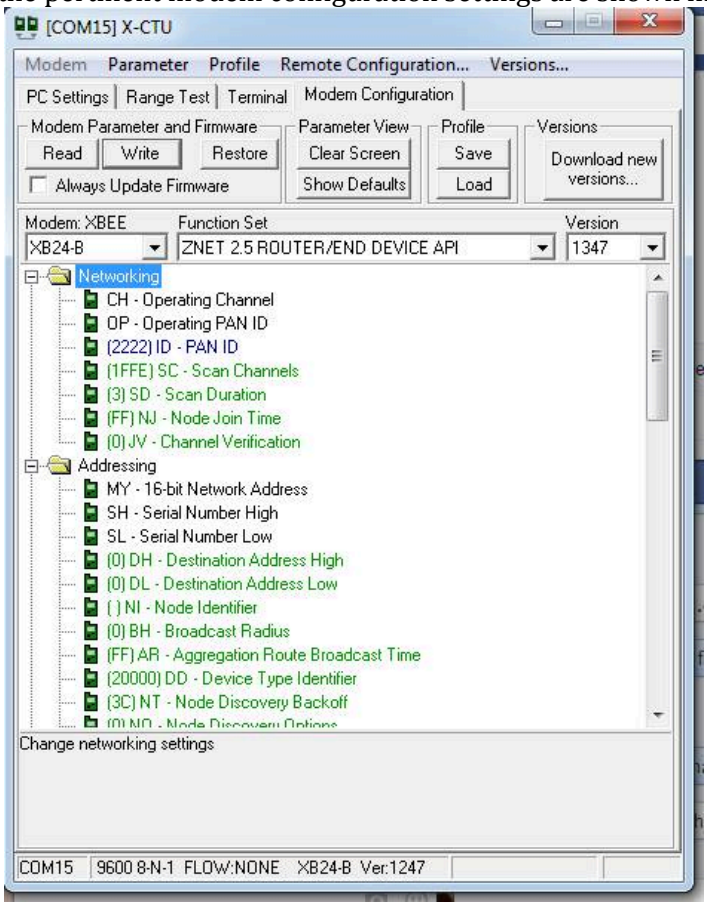

Figure 1: Xbee Modem Configuration

2. XBee operation was verified using a secondary controller currently used by lab reporter Sane. This action provided advance knowledge of send/receive pairs, allowing the team to decode the cipher (discussed below).
4. Using the RX63N datasheet and user manual, the UART is initialized as follows:

```
#define BAUDRATE   9600
…
/* Set up the UART I/O port and pins. */
MPC.P50PFS.BYTE = 0x4A; /* P50 is TxD2 */
MPC.P52PFS.BYTE = 0x4A; /* P52 is RxD2 */
PORT5.PDR.BIT.B0 = 1; /* TxD2 is output. */
PORT5.PDR.BIT.B2 = 0; /* RxD2 is input. */
PORT5.PMR.BIT.B0 = 1; /* TxD2 is peripheral. */
PORT5.PMR.BIT.B2 = 1; /* RxD2 is peripheral. */

/* initialization of sci */
sci_uart_init();
sci_tx_int_enable();
sci_rx_int_enable();
```

The output of the receive routines is polled in the main loop:

```
while(!sci_read_ct_get()) //Rx wait
{
   for (int j=0; j<17; j++)
   {
      a[j]=sci_get_char(); //get received character
   }
   sprintf(Buff1, "%c", a[15]);
   lcd_display(LCD_LINE2, Buff1);
   for(int rj=0;rj<5;rj++)
   {
      w_pred();
   }
}

for (int j=0; j<17; j++)
{
   a[j]=sci_get_char(); //get received character
}
```

Transmission occurs by placing a byte in the transmit buffer and allowing the interrupt service routines to take control:

```
sci_put_char(0x7E); //Delimiter
sci_put_char(0x00);//Length
sci_put_char(0x15);//Length
sci_put_char(0x10);//Identifier
sci_put_char(0x00);//Frame ID
//64 Bit Ad
sci_put_char(0x00);
sci_put_char(0x00);
sci_put_char(0x00);
sci_put_char(0x00);
sci_put_char(0x00);
sci_put_char(0x00);
sci_put_char(0xFF);
sci_put_char(0xFF);
//16 bit
sci_put_char(0x00);
sci_put_char(0x01);
//Broadcast Radius
sci_put_char(0x00);
//Disable ACk
sci_put_char(0x01);

//Data
sci_put_char(dig1);
//Cheksum
sci_put_char(0x00);
// Transmission
```

5. The switch functions from Lab 2 were modified to conform to the requirements as follows:

```
if(SW1==0 ) { temp++; dig1++; if (dig1 > 126) { dig1 =32; } }

if (SW2==0 ) { temp++; dig1--; if (dig1 < 32) { dig1 =126; } }
sprintf(Buff, "%c", dig1); lcd_display(LCD_LINE1, Buff);

if (SW3==0) { sci_put_char(0x7E); //DElimiter
sci_put_char(0x00);//Length sci_put_char(0x15);//Length
sci_put_char(0x10);//Identifier sci_put_char(0x00);//Frame ID
//64 Bit Ad sci_put_char(0x00); sci_put_char(0x00);
sci_put_char(0x00); sci_put_char(0x00); sci_put_char(0x00);
sci_put_char(0x00); sci_put_char(0xFF); sci_put_char(0xFF);
//16 bit sci_put_char(0x00); sci_put_char(0x01); //Broadcast
Radius sci_put_char(0x00); //Disable ACk sci_put_char(0x01);
```

```
//Data sci_put_char(dig1); //Cheksum sci_put_char(0x00); //
Transmission

w_pred();//wait period ct=100000; while(!sci_read_ct_get())
//Rx wait { for (int j=0; j<17; j++) {

a[j]=sci_get_char(); //get received character }

sprintf(Buff1, "%c", a[15]); lcd_display(LCD_LINE2, Buff1);
for(int rj=0;rj<5;rj++) {w_pred(); }
}
}
}
```

6. After porting the switch functions, the investigators used the LCD screen to debug.
7. With the switch functions operating as designed, the investigators then used the LCD screen and an oscilloscope to debug serial communication to the XBee module.
8. The investigators integrated the XBee and RX63N and ensured correct system behavior.
9. Table 1 provides several send/receive pairs:

Send (*x*)		**Receive (*y*)**	
Character	*Decimal*	*Character*	*Decimal*
'f'	102	'C'	67
Space	32	Space	32
'!'	33	Space	32
'z'	122	'M'	77
'9'	57	','	44

Table 1: Send/Receive Pairs

By inspection, the equation to determine the received character *y*, based on the sent character *x* is the following:

$$y = \text{floor}[\frac{x-32}{2}] + 32$$

The floor function is only mathematically necessary outside the context of C code, where the implied integer division effects truncation without calling an additional function.

Important observations:
The XBee serial pins sink more current than the RX63N pins can source, necessitating a current buffer circuit.

In this lab we learned: XBee communication & encryption technique used.