Name: Disha Srivastava
Id: dsrivas2@uncc.edu
Name: Swapnil Modak
Id: smodak@uncc.edu

ECGR – 6185
Advanced Embedded System
Final Report

Date: 4/10/15

TITLE:
Motesquito

TEAM MEMBERS:
Disha Srivastava, Swapnil Modak

OBJECTIVE:
Create a new layer for the Xbee Library to allow the user to XBee configure its parameters upon startup.

MOTIVATION:
A WSN (Wireless Sensor Network) development platform called Motesquito is being designed to avoid the limitations presented by the TinyOS platform. Motesquito is designed for widely used hardware and software, where support communities are large. The board will include a large software library and sockets for the wireless module, so the user can easily change radio modules of their preference. Currently, the Digi XBee radio modules are being used in the platforms initial design, but will be expanded to include many more radios. The XBee is one of the most common Zigbee wireless interfaces in use today by both professionals and hobbyists.

Due to its reliability and ease of use, Digi's XBee radio modules have become a very popular tool used for wireless communication. In a robotics application for example, an XBee module would be used to send and receive data between a beacon and an independently traversing robot almost instantaneously.

Renesas and Arduino have development boards with XBee slots embedded into the hardware. The two most common are Renesas Sakura and Arduino Fio. Although these boards have XBee slots, the placement of these slots are positioned on the boards in such a way that the user has limited antenna selection, due to the lack of physical space. The Sakura boards XBee socket is located on the underside, which is impractical considering the headers for interfacing with sensors and peripherals are on the top. Also, because the XBee is attached on the underside of the Sakura board, it prevents the user from attaching a battery back on that side, which would be the most Arduino Fio Board. Renesas Sakura Board Front and Back View appropriate location. The Arduino Fio's XBee socket positions the XBee such that the antenna faces inward towards the other components, preventing the use of modules that use antennas connected via SMA. On some Fio models, the XBee header is located on the bottom of the board, introducing the same problems that the Sakura board's XBee position presents.

HARDWARE:

Motesquito was designed for portability, adaptability, and high performance. Specific components were carefully chosen for the design to optimize these three features. The placement of these components, their ease of use, and their functionality were highly considered as well. With this in mind, research was conducted to determine what boards were typically used by developers. Many Arduino platforms were observed in the first stage of the design process. The Fio, MEGA, and Uno were studied in particular because each individually provided the adaptability aspect that would contribute to the design of this development board. Of these platforms, the Arduino MEGA supplies the most functionality, and in turn, the best performance. The MEGA board features the ATmega2560 microprocessor that has been recognized for larger amount of peripherals which could be used to interface with many sensors directly on a sensor board. The ATmega 1280 microprocessor has been used on an earlier version of Arduino's MEGA platform. The difference between the chips is that the ATmega 1280 can operate at higher frequencies at lower voltages. Both chips have been used on two different versions of the Motesquito that have been created.

Name: Disha Srivastava
Id: dsrivas2@uncc.edu
Name: Swapnil Modak
Id: smodak@uncc.edu

ECGR – 6185
Advanced Embedded System
Final Report

Date: 4/10/15

For the testing of library Renesas RX63N board has been used along with Xbee Series 1 & Series 2 radios. However this library is platform independent so it can also be used with other development board such as Arduino as well.

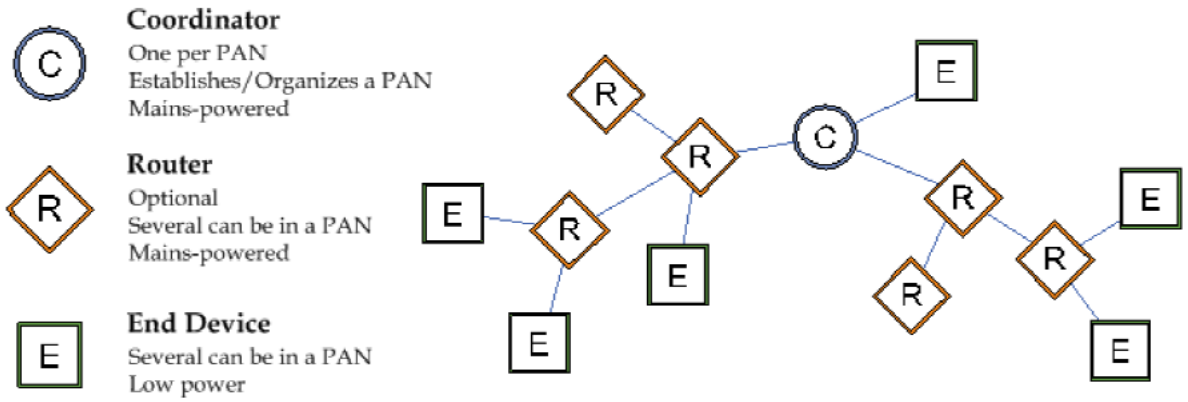
SOFTWARE:

A software library is being developed to provide simple interfaces to Motesquito's peripherals. Due to the hardware being similar to the Arduino Mega, it is possible for a user to load the Arduino bootloader onto the microcontroller and use the Arduino XBee library to interface with the XBee module. However, for applications where the user needs full control of the hardware, a C library will be created to provide simple functions to construct and send packages, receive packages, and change XBee settings. The library will also support both series 1 and series 2 XBee modules, and expand to other types of wireless modules which have been developed to use the XBee footprint. By using the XBee footprint and allowing the wireless module to be interchangeable, it is easy for developers to change the wireless interface. The use of avrgcc and XBee makes debugging problems easy because the hardware is widely documented and has large support groups online. The library will include a hardware abstraction layer, so users can use the software with other microcontrollers.

While testing the IDE used is High Performance Embedded Workshop by Renesas.

WORK DONE:

To implement the library functions documentation about Xbee and AT commands were studied.
Xbee modules provide an easy way to add wireless capacity to an embedded application. It works under the 802.15.4 standard for point-to-point and star communications at over the baud rates of 250kbps. Xbee operates either in a transparent data mode or in a packet-based application programming interface (API) mode. In the transparent mode data coming into the Data IN (DIN) pin is directly transmitted over-the-air to the intended receiving radios without any modifications. Incoming packets can be directly addressed to one target or broadcasted to multiple targets. In API mode the data is wrapped in a packet structure that allows for addressing, parameter setting and packet delivery feedback including remote sensing and control of digital I/O and analog pins. ZigBee defines three different device types: coordinator, router, and end device.



Name: Disha Srivastava
Id: dsrivas2@uncc.edu
Name: Swapnil Modak
Id: smodak@uncc.edu

ECGR – 6185
Advanced Embedded System
Final Report

Date: 4/10/15

A **coordinator** has the following characteristics:

- Selects a channel and PAN ID (both 64-bit and 16-bit) to start the network
- Can allow routers and end devices to join the network
- Can assist in routing data
- Cannot sleep--should be mains powered
- Can buffer RF data packets for sleeping end device children.

A **router** has the following characteristics:

- Must join a ZigBee PAN before it can transmit, receive, or route data
- After joining, can allow routers and end devices to join the network
- After joining, can assist in routing data
- Cannot sleep--should be mains powered.
- Can buffer RF data packets for sleeping end device children.

An **end device** has the following characteristics:

- Must join a ZigBee PAN before it can transmit or receive data
- Cannot allow devices to join the network
- Must always transmit and receive RF data through its parent. Cannot route data.
- Can enter low power modes to conserve power and can be battery-powered.

Various parameters were defined in the Xbee_Config.h file alongwith a certain default value. The range of all possible values is also specified as comments with the defined parameters.

```
#ifdef XBEES1
/***************** Networking & Security*******************/

    #define CH C /* Set/read the channel number (Uses 802.15.4 channel numbers) */
    #define ID 1234 /* Set the PAN(Personal Aread Network) ID. Set ID = 0xFFFF to send message to all PANs */
    #define DH 0x0000 /* Range: 0x0 to 0xFFFFFFFF */
    #define DL 0xFFFF  /* Lower bits set to FFFF to broadcast for PAN */
    #define MY D /* Set/read 16-bit source address of the modem */
    #define CE 0 /* Set/Read the coordinator setting: 0-End Device, 1-Coordinator.*/
    #define RR 0 /* Set the number of retries the modem will execute in addition to the 3 retries provided by the
                  * 802.15.4 MAC. For each XBee entry, 802.15.4 MAC can execute upto 3 retries */
    #define RN 0 /* Set/read the minimum value of the back-off exponent in the CSMA-CA algorithm that is used for
                * collision avoidance. If RN=0, collision avoidance is disabled for the first iteration of the algorithm */
```

Since all the command need to be sent via UART, the AT command need to be converted into a string and stored in a buffer. This buffer is then used to send the commands byte-by-byte.
Following lines of code are used to send the command as a string.
The sprint line convert the AT command ‘ATID’ and its assigned value to a string and stores it into the ‘cmd_buff’ buffer

```
#ifdef XBEES1
    #ifdef ID
        sprintf(cmd_buff, "ATID %d\r", ID);
        delay1();
        sendCommand();
        printf("ID sent\n");
        delay1();
        readResponse();
    #endif
```

The sendCommand() function takes the data stored in cmd_buff[] and passes it to USART_vSendByte() which transmits it via the UART.

```
void sendCommand(void)
{
    i=0;
    while(cmd_buff[i]!='\r')
    {
        s(cmd_buff[i]);
        i++;
    }
    USART_vSendByte(cmd_buff[i]);
}
```

The readResponse() function is used to read the receive data buffer and check for characters ‘0’ & ‘K’. Print OK if found else print Not OK

```
void readResponse(void)
{
    SCI6.SSR.BIT.ORER = 0;
    while(RX_BUFFER_IS_FULL()) {};
    if(XBEE_RDR == 'O' || XBEE_RDR == 'K')
        {
            printf("\nOK");
        }
    else
        printf("Not OK");
}
```

Here the USART_vSendByte() is used to send ‘+++’ command as ‘0x2B’ one byte at a time, and wait for response OK by the Xbee. This command puts the Xbee in command mode enabling other AT commands to work.

```
    USART_vSendByte(0x2B);
        USART_vSendByte(0x2B);
        USART_vSendByte(0x2B);
        // wait for 'OK'
        SCI6.SSR.BIT.ORER = 0;
        while(RX_BUFFER_IS_FULL()) {};
        if(XBEE_RDR == 'O' || )
        {
                printf("O for +++\n");
        }
        while(RX_BUFFER_IS_FULL()) {};
        if(XBEE_RDR == 'K')
        {
                printf("K for +++\n");
        }
```

FUTURE IMPLEMENTATION

The XbeeConfig.h will be tested on a sparkfun redboard to check for implementation of the parameters by uncommenting them in the header file for both Xbee series 1 as well as 2 modules.

```
/*
 * XbeeConfig.h
 *
 * Created: 7/11/2014 4:55:59 PM
 *  Author: Waron
 */


#ifndef XBEECONFIG_H_
#define XBEECONFIG_H_
#include "Xbee.h"

//#define XBEES2

#ifdef XBEES1
void delay1(void);
void sendCommand(void);
void readResponse(void);

// Networking & Security

       #define CH 'B'  //Set/read the channel number (Uses 802.15.4 channel numbers)
       #define ID 1444 // Set the PAN(Personal Aread Network) ID. Set ID = 0xFFFF to send message to
all PANs
       #define DH 0x0000 // Range: 0x0 to 0xFFFFFFFF
       #define DL 0xFFFF  // Lower bits set to FFFF to broadcast for PAN
       #define MY 'I' // Set/read 16-bit source address of the modem
       #define CE 0 // Set/Read the coordinator setting: 0-End Device, 1-Coordinator.
       #define RR 0 // Set the number of retries the modem will execute in addition to the 3 retries
provided by the
                    // 802.15.4 MAC. For each XBee entry, 802.15.4 MAC can execute upto 3 retries
```

```
	#define RN 0 // Set/read the minimum value of the back-off exponent in the CSMA-CA algorithm that is used for
		// collision avoidance. If RN=0, collision avoidance is disabled for the first iteration of the algorithm
	#define NT 17 // Set/read Node Discover Time register. This sets the maximum time for the Node Discover command
		// Range: 0x1-0xFC(Default: 19)
	#define NO 0 // Enable node discover self-response. Range: 0x0 - 0x1(Default: 0)
	#define SC 0x1F0E // Read/set list of channels to scan for Active and Energy scans as bitfield.
		// Range: 0x0-0xFFFF(Default: 1FFE)
	#define SD 8 // Set the Scan duration exponent. Set End device SD=BE of beaconing coordinator. Scan time= N*(2^SD)*15.36ms.

	#ifdef COORDINATOR
	#define A2 1 // Set/read Coordinator association options. Options enabled when bits are set: bit2-Allow Association, bit1-Allow
		// Channel reassignment, bit0-Allow PanId reassignment.
	#define EE 0 // Enables AES encryption
	#define KY 0x20 // Sets key used for encryption and decryption. This register can not be read.
	#define NI 0x20 // Set/Read Node Identifier string
	#endif
// RF Interfacing

	#define PL 1 //	Select/Read transmitter output power
		// Default: 4(Highest)
	#define CA 0x2A // Set the Clear Channel Assessment(CCA) threshold. If the modem detects energy above the CCA threshold,
		// it will not transmit. Unit: -dBm

// Sleep Modes (NonBeacon)- Only end device
	#ifdef END_DEVICE
	#define A1 0 // Set/read End Device association options. Options enabled when bits are set: bit3-Poll coordinator on pin wake, bit2-Auto Associate,
		// bit1-Allow Channel reassignment, bit0-Allow PanId reassignment.
	#define SM 0 // Set/read sleep mode: Pin Hibernate is lowest power, Pin Doze provides the fastest wake up, Cyclic Sleep Remote with or without pin
		// wake up. Sleep Coordinator setting is for SM parameter compatibility with version 106; ATCE should be used going forward.
	#define ST 1388 // Set the time period of inactivity before activating Sleep Mode. This parameter is used only with the Cyclic
		// Sleep settings(SM=4-5).
	#define SP 0 // Set Cyclic sleep period for cyclic sleeping remotes. Coordinator will discard indirect messages after a period of 2.5*SP,
		// set Coordinator SP=0 to send direct messages. Maximum sleep period: 268 seconds. Range: 0x0-0x68B0
	#define DP 3E8 // Set sleep period for sleeping remotes that are configured for association but that are not associated to
		// a Coordinator. Range: 0x1-0x68B0 (Default: 3E8)

	#define SO 0 // Set sleep options. Bit field options include:
		// 0x01- Enables/Disables sending a wakeup poll request on devices configured for cyclic sleep sampling.
		// 0x02- Suppresses sending ADC/DIO samples when waking form cyclic sleep.
		// All other options bit should be set to 0
	#endif

// Serial Interfacing

	#define RO 3 // Set the number of character times of inter-character delay required before transmission begins. Set to 0 to transmit characters
		// as they arrive instead of buffering them into one RF packet. Range: 0x0- 0xFF
```

```
	#define AP 0 // Enable : 1
	             // Disable : 0
	#define BD 9600 // Set Baud rate for communication between modem serial port and host
	#define NB 2 // Configure Parity for the UART
	             // No Parity:0 Even Parity:1 Odd Parity:2 Mark Parity:3 Space Parity:4

// I/O Settings

	#define D8 0 // Configure options for the DI8 line of the module. Options include: Digital
Input. This line is also used with Pin Sleep
	#define D7 1 // Configure options for the DIO7 line of the module. Options include: CTS flow
control, Digital Input and Output.
	#define D6 0 // Configure options for the DIO6 line of the module. Options include: RTS flow
control, Digital Input and Output.
	#define D5 1 // Configure options for the DIO5 line of the module. Options include:
Associated LED indicator (blinks when associated),Digital Input and Output.
	#define D4 0 // Configure options for the DIO4 line of the module. Options include: Analog to
Digital converter,Digital Input and Output.
	#define D3 0 // Configure options for the DIO3 line of the module. Options include: Analog to
Digital converter,Digital Input and Output.
	#define D2 0 // Configure options for the DIO2 line of the module. Options include: Analog to
Digital converter,Digital Input and Output.
	#define D1 0 // Configure options for the DIO1 line of the module. Options include: Analog to
Digital converter,Digital Input and Output.
	#define D0 0 // Configure options for the DIO0 line of the module. Options include: Analog to
Digital converter,Digital Input and Output.

	#define PR 0xFF // Set bitfield to configure internal pull-up resistors status for I/O lines.
	                // 1= internal pull-up enabled
	                        // 0= no internal pull-up
	                        // Range: 0x0- 0xFF
	                        // Bitfield map: Bit 7-DIN(P3), Bit 6-DI8/SLEEP_RQ(P9), Bit 5-
DIO6/RTS(P16), Bit 4-DIO0(P20), Bit 3-DIO1(P19),
	                       // Bit 2-DIO2(P18), Bit 1-DIO3(P17), Bit 0-DIO4(P11).
	#define IU 1 // Enables I/O data received to be sent out UART. The data is sent using an API
frame regardless of the current ATAP mode. Default: 1
	#define IT 1 // Set number of samples to collect before transmission. The maximum number of
samples is dependent on the number of enabled I/O lines.
	             // Range: 0x1-0xFF (Default: 1).
	#define IC 0 // Set values for change detect monitoring. Each bit enables monitoring of DIO0-
DIO7 for changes. If detected, data is transmitted
	             // with DIO data only. Any samples queued waiting for transmission will be sent
first.
	                      // Range: 0x0-0xFF (Default: 0)
	#define IR 0 // Set sample rate. When set, this parameter causes the modem to sample all
enabled DIO and ADC at a specified interval.
	             // Range: 0x0-0xFFFF (Default: 0)
	#define PO 1 // Select/Read function for PWM0. Disabled:0 RSSI:1 PWM Output:2
	#define P1 0 // Select/Read function for PWM1. Disabled:0 RSSI:1 PWM Output:2
	#define PT 0xFF // Set output timeout value for both PWM outputs. Range: 0x0-0xFF(Default:FF)
	#define RP 28 //Set PWM time register. Set duration of PWM signal output on the RSSI pin(P6).
	           // Range: 0x0-0xFF (Default:28)

// I/O Line Passing

	#define IA 0xFFFFFFFFFFFFFFFF // Set/read addresses of module to which outputs are bound.
Range: 0-16 hexadecimal characters.
	#define T0 0xFF // Set/read output timeout value. When output is set, due to I/O line passing
to a non-default level a timer is started which when
	                // expired will set output to its default level. Range: 0x0-0xFF(Default:FF)
	#define T1 0xFF // Set/read output timeout value. When output is set, due to I/O line passing
to a non-default level a timer is started which when
	                        // expired will set output to its default level. Range: 0x0-
0xFF(Default:FF)
```

```
	#define T2 0xFF // Set/read output timeout value. When output is set, due to I/O line passing
to a non-default level a timer is started which when
				// expired will set output to its default level. Range: 0x0-
0xFF(Default:FF)
	#define T3 0xFF // Set/read output timeout value. When output is set, due to I/O line passing
to a non-default level a timer is started which when
				// expired will set output to its default level. Range: 0x0-
0xFF(Default:FF)
	#define T4 0xFF // Set/read output timeout value. When output is set, due to I/O line passing
to a non-default level a timer is started which when
				// expired will set output to its default level. Range: 0x0-
0xFF(Default:FF)
	#define T5 0xFF // Set/read output timeout value. When output is set, due to I/O line passing
to a non-default level a timer is started which when
				// expired will set output to its default level. Range: 0x0-
0xFF(Default:FF)
	#define T6 0xFF // Set/read output timeout value. When output is set, due to I/O line passing
to a non-default level a timer is started which when
				// expired will set output to its default level. Range: 0x0-
0xFF(Default:FF)
	#define T7 0xFF // Set/read output timeout value. When output is set, due to I/O line passing
to a non-default level a timer is started which when
				// expired will set output to its default level. Range: 0x0-
0xFF(Default:FF)


// Diagnoistics

	#define DD 10000 // Read the device type identifier value. Range: 0x0= 0xFFFFFFFF


// AT Command Options

	#define CT 0x26 // Set command mode timeout parameter. If no valid commands have been
received
		// within this time period, modem returns to Idle Mode from AT Command Mode.
				// Range: 0x2-0x1770(Deafult:64)
	#define GT 0x3E8 // Set required period of silence before, after and between the Command Mode
Characters
			// of the Command Mode Sequence (GT + CC + GT). The period of silence is used
to prevent inadvertent entrance into AT Command Mode.
					// Range: 0x2-0xCE4(Default:3E8)
	#define CC 0x1C // Set/read character value to be used to break from data mode to command
mode. The default value <2Bh> is the ASCII code for the plus ('+') character.
			// Range: 0x0-0xFF(Default: 2B).

#endif


#ifdef XBEES2

// Networking

	#define ID 4212 // Set the PAN (Personal Area Network) ID the device should join.
			// Range: 0 - 0xFFFFFFFFFFFFFFFF
					// Set ID=0 (default) for the device to join any PAN ID.
	#define SC FFFF // Set/read list of channels to scan (active and energy scans) when forming a
PAN as bitfield. Scans are initiated during coordinator
			// startup: Bit 15 = Chan 0x1A . . . Bit 0 = Chan 0x0B
					// Range: 0x1-0xFFFF (Default:FFFF)
	#define SD 4 // Set/read the Scan Duration exponent. The exponent configures the duration of
the active scan (PAN scan) on each channel in the SC
				//    channel mask when attempting to join a PAN. Scan Time = SC * (2 ^
SD) * 15.36ms. (SC=# channels)
```

```
                                // Range: 0x0-0x07(Default:3)
        #define ZS 0 // Set/read the ZigBee stack profile setting. 0=Network Specific, 1=ZigBee-2006,
2=ZigBee-PRO
                                // Range: 0x0-0x2(Default:0)
        #define NJ FF // Set/read the Node Join time. The value of NJ determines the time (in
seconds) that the device will allow other devices to join to it.
                        // If set to 0xFF, the device will always allow joining.
                                 // Range: 0x0-0xFF(Default:FF)
        #define NW 0 // Set/read the network watchdog timeout. If set to a non-zero value, the
network watchdog timer is enabled on a router. The router will leave
                       // the network if is does not receive valid communication within (3 * NW)
minutes. The timer is reset each time data is received from or sent to a coordinator,
                                // or if a many-to-one broadcast is received.
                                // Range: 0x0-0x64FF(Default:0)
        #define  JV 1 // Set the channel verification setting. If enabled, a router will verify a
coordinator exists on the same channel after joining or power cycling to
                        // ensure it is operating on a valid channel, and will leave if a coordinator
cannot be found (if NJ=0xFF). If disabled, the router will remain on the same channel
                                // through power cycles. Default: 0
        #define JN 0 // Set the join notification setting. If enabled, the module will transmit a
broadcast node identification frame on power up and when joining.
                       // This action blinks the Assoc LED rapidly on all devices that receive the
data, and sends an API frame out the UART of API devices. This function should be disabled
                                // for large networks. Default: 0

// Addressing

        #define DH 0 // Set the upper 32 bits of the 64 bit destination extended address.
0x000000000000FFFF is the broadcast address for the PAN. 0x0000000000000000 can be used
                                // to address the Pan Coordinator.
                                // Range: 0x0-0xFFFFFFFF(Default:0)
        #define DL 0 // Set the lower 32 bits of the 64 bit destination extended address.
0x000000000000FFFF is the broadcast address for the PAN. 0x0000000000000000 can be used to
                       // address the Pan Coordinator.
                                // Range: 0x0-0xFFFFFFFF(Default:0)
        #define NI 0 // Set Node Identifier string. 0-20 ASCII characters. Default: ''
        #define NH 1E // Set the maximum hops limit. This limit sets the maximum number of broadcast
hops (BH) and determines the unicast timeout (unicast timeout = (50 * NH) + 100).
                        // A unicast timeout of 1.6 seconds (NH=30) is enough time for the data and
acknowledgment to traverse about 8 hops.
                                 // Range:0x0-0xFF(Default:1E)
        #define BH 0 // Set the transmission radius for broadcast data transmissions. Set to 0 for
maximum radius.
                                // Range:0x0-0x1E(Default:0)
        #define AR FF // Set/read the time between aggregation route broadcast times. An aggregation
route broadcast creates a route on all devices in the network back to the device
                        // that sends the aggregation broadcast. Setting AR to 0xFF disables
aggregation route broadcasting. Setting AR to 0 sends one broadcast.
                                 // Range:0x0-0xFF(Default:FF)
        #define DD 30000 // Set the device type identifier value. This can be used to differentiate
multiple XBee-based products.
                           // Range:0x0-0xFFFFFFFF(Default:30000)
        #define NT 3C // Set Node Discovery backoff register. This sets the maximum delay for Node
Discovery responses to be sent (ND, DN).
                        // Range:0x20-0xFF(Default:3C)
        #define NO 0 // Sets the node discovery options register. Options include 0x01 - Append DD
value to end of node discovery, 0x02 - Return devices own ND response first.
                      // Range:0x0-0x3(Default:0)
        #define CR 3 // Set/read threshold for the number of PAN id conflict reports that must be
received by the network manager within one minute to trigger a PAN id change.
                      // Range:0x1-0x3F(Default:3)


// RF Interfacing
```

```
	#define PL 4 // Select/Read transmitter output power. Power levels relative to PP: 0=-10dB, 1=-6dB, 2=-4dB, 3=-2dB, 4=0dB. Default: 4
	#define PM 1 // Select/Read module boost mode setting. If enabled, boost mode improves sensitivity by 1dB and increases output power by 2dB, improving
		// the link margin and range. Default: 1


// Security- Coordinator
	#ifdef COORDINATOR
	#define EE 0 // Enable or disable ZigBee encryption. Default:0
	#define EO 0 // Set the ZigBee Encryption options: Bit0 = Acquire / Transmit network security key unencrypted during joining.
		// Range: 0x0-0x1(Default: 0)
	#define KY 0 // Sets key used for encryption and decryption (ZigBee trust center link key). This register can not be read. 0-32 hexadecimal characters. Default:''
	#endif
// Serial Interfacing

	#define BD 9600 // Set Baud rate for communication between modem serial port and host
	#define NB 0 // Configure Parity for the UART
		// No Parity:0 Even Parity:1 Odd Parity:2 Mark Parity:3 Space Parity:4
	#define SB 0 // Configure the number of stop bits for the UART. This setting is only valid for no parity, even parity, or odd parity.
	#define D7 1 // Configure options for the DIO7 line of the module. Options include: CTS flow control, Digital Input and Output, or RS-485 enable control.
	#define D6 0 // Configure options for the DIO6 line of the module. Options include: RTS flow control, and Digital Input and Output.
	#define AP 1 // Enables API mode.
			// Range: 0x1-0x2
	#define AO 0 // Set the API output mode register value. 0 - Received Data formatted as native API frame format. 1 - Received RF data formatted as Explicit Rx-Indicator.
		// 3 - Same as 1, plus received ZDO requests are passed out the UART.


// Sleep Modes Only End devices
	#ifdef END_DEVICE
	#define SM 0 // Set/read sleep mode: Pin Hibernate is lowest power, Cyclic Sleep wakes on timer expiration, Cyclic Sleep Pin-Wake wakes on timer expiration or when
		// Sleep_Rq (module pin 9) transitions from a high to a low state. If SM is set to 0, the XBee is a router, otherwise it is an end device.
	#define SN 1 // Set/read the number of cyclic sleep periods used to calculate end device poll timeout. If an end device does not send a poll request to its parent coordinator
		// or router within the poll timeout, the end device is removed from the child table. The poll timeout is calculated in milliseconds as (3 * SN * (SP * 10ms)),
			// minimum of 5 seconds. i.e. if SN=15, SP=0x64, the timeout is 45 seconds.
		// Range: 0x1-0xFFFF
	#define SO 0 // Set/read sleep options. Bitfield options include: 0x02 - Wake for ST time on each cyclic wake (after sleeping for SN sleep periods), 0x04 - Enable extended cyclic
		//sleep (sleep for entire SN*SP time - possible data loss). All other option bits should be set to 0.
			// Range: 0x1-0xFF
	#define SP 20 // Set/read Cyclic sleep period for cyclic sleeping remotes. Set SP on parent (Coordinator) to match the largest SP of its end device children. On a router or coordinator,
		// SP determines the transmission timeout when sending to a sleeping end device. SP also determines how long the parent will buffer a message for a sleeping child.
		// Range: 0x20-0xAF0 (Default:20)
	#define ST 1388 // Set/read time period of inactivity (no serial or RF data is sent or received) before activating Sleep Mode. The ST parameter is used only with Cyclic Sleep settings (SM=4-5).
		// Range: 0x1-0xFFFE (Default:1388)
```

```
	#define PO 0 // Set/read the poll rate in 10 msec units when set as a sleepy end device. Setting this to 0 (default) enables polling at 100ms (default rate). Adaptive polling may allow the end device
		// to poll more rapidly for a short time when receiving RF data.
		// Range: 0x0-0x3E8 (Default:0)
	#endif
// I/O Settings

	#define D0 1 // Configure options for the AD0/DIO0 line of the module. Options include: Enabling commissioning button functionality[1], Analog to Digital converter[2],Digital Input[3], Digital Output, Low[4]
		 // and Digital Output, High[5]. Default: 1
	#define D1 0 // Configure options for the AD1/DIO1 line of the module. Options include: Analog to Digital converter,Digital Input and Output. Default: 0
	#define D2 0 // Configure options for the AD2/DIO2 line of the module. Options include: Analog to Digital converter,Digital Input and Output. Default: 0
	#define D3 0 // Configure options for the AD3/DIO3 line of the module. Options include: Analog to Digital converter,Digital Input and Output. Default: 0
	#define D4 0 // Configure options for the DIO4 line of the module. Options include: Digital Input and Output. Default: 0
	#define D5 1 // Configure options for the DIO5/Assoc line of the module. Options include: Associated LED indicator (blinks when associated),Digital Input and Output. Default: 1
	#define P0 1 // Configure options for the DIO10 line of the module. Options include: RSSI PWM Output, Digital Input and Output. Default: 1
	#define P1 0 // Configure options for the DIO11 line of the module. Options include: Digital Input and Output. Default: 0
	#define P2 0 // Configure options for the DIO12 line of the module. Options include: Digital Input and Output. Default: 0
	#define PR 1FFF // Set/read bitfield to configure internal pullup resistors status for I/O lines. 1=internal pullup enabled, 0=no internal pullup. Bitfield map: (13)DIO7/CTS, (12)-DIO11, (11)-DIO10/PWM0,
		// (10)-DIO12, (9)-On/Sleep, (8)Associate, (7)-DIN/Config, (6)-Sleep_Rq, (5)-RTS, (4)-AD0/DIO0, (3)-AD1/DIO1, (2)-AD2/DIO2, (1)-AD3/DIO3, (0)-DIO4
			// Range: 0x0-0x3FFF (Default: 1FFF)
	#define LT 0 // Set/read the Associate LED blink rate. This value determines the blink rate of the Associate/DIO5 pin if D5=1 and the module has started a network. Setting LT to 0 will use the default blink time (250ms).
		// Range: 0x0A-0xFF (Default: 0)
	#define RP 28 // Set/read PWM timer register. Set duration of PWM (pulse width modulation) signal output on the RSSI pin (P6). The signal duty cycle is updated with each received packet or APS acknowledgment and is
		 // shut off when the timer expires.
			// Range: 0x0-0xFF (Default: 28)
	#define DO 1 // Bit0 - Reserved. Bit1 - Reserved. Bit2 - 0/1 = First or Best Response. Bit3 - Reserved.
		// Range: 0x0-0xFF (Default: 1)


// I/O Sampling


	#define IZ 0 // Set the IO sampling rate to enable periodic sampling. If set >0, all enabled digital IO and analog inputs will be sampled and transmitted every IR milliseconds. IO Samples are transmitted to the address
		// specified by DH+DL.
			// Range: 0x32-0xFFFF (Default: 0)
	#define IC 0 // Bitfield that configures which digital IO pins should be monitored for change detection. If a change is detected on an enabled digital IO pin, a digital IO sample is immediately transmitted to the address
		// specified by DH+DL.
			// Range: 0x0-0xFFFF (Default: 0)
	#define VP 0 // Configure the supply voltage high threshold. If the supply voltage measurement equals or drops below this threshold, the supply voltage will be appended to an IO sample transmission. Since the operating Vcc
```

```
                    // range for the XBee is 2100-3600 mV, after scaling by 1024/1200, the useful
range for this parameter is 0,0x0700-0x0c00.
                            // Range: 0x0-0xFFFF (Default:0)
#endif


int i;
char cmd_buff[15]=' ';

void sendCommand(void)
{
       i=0;
       while(cmd_buff[i]!='\r')
       {
              USART_vSendByte(cmd_buff[i]);
              i++;
       }
       USART_vSendByte(cmd_buff[i]);
}

void readResponse(void)
{
       SCI6.SSR.BIT.ORER = 0;
       while(RX_BUFFER_IS_FULL()) {};
       if(XBEE_RDR == 'O' || XBEE_RDR == 'K')
              {
                     printf("\nOK");
              }
       else
              printf("Not OK");
}

void delay1(void)
{
       for(int d=0; d<10000; d++)
       {
              for(int e=0; e<8000; e++)
              {
              }
       }
}


int XbeeSetup()
{
       char d;

       char wr_buff[15]=' ';

       char RxData = ' ';
       char temp_string[2];
       printf("+++\n");

              USART_vSendByte(0x2B);
                     USART_vSendByte(0x2B);
                     USART_vSendByte(0x2B);
                     // wait for 'OK'
              //     delay1();
                     SCI6.SSR.BIT.ORER = 0;
                     RxData = XBEE_RDR;
                     printf("%c", RxData);
                     while(RX_BUFFER_IS_FULL()) {};

                     if(XBEE_RDR == 'O' )
```

```
                {
                        printf("O for +++\n");
                }
                while(RX_BUFFER_IS_FULL()) {};
                if(XBEE_RDR == 'K')
                {
                        printf("K for +++\n");
                }
                else
                printf("Baddddd");

        #ifdef XBEES1
                #ifdef ID
                        sprintf(cmd_buff, "ATID %d\r", ID);
                        delay1();
                        sendCommand();
                        printf("ID sent\n");
                        delay1();
                        readResponse();
                #endif

                #ifdef CH
                        sprintf(cmd_buff, "ATCH %c\r", CH);
                        delay1();
                        sendCommand();
                        printf("\nChannel set");
                        delay1();
                        readResponse();
                #endif

                #ifdef DH
                        sprintf(cmd_buff, "ATDH %x\r", DH);
                        delay1();
                        sendCommand();
                        printf("\nDestination High set");
                        delay1();
                        readResponse();
                #endif

                #ifdef DL
                        sprintf(cmd_buff, "ATDL %x\r", DL);
                        delay1();
                        sendCommand();
                        printf("\nDestination Low set");
                        delay1();
                        readResponse();
                #endif

                #ifdef MY
                        sprintf(cmd_buff, "ATMY %c\r", MY);
                        delay1();
                        sendCommand();
                        printf("\nSource address set");
                        delay1();
                        readResponse();
                #endif

                #ifdef CE
                        sprintf(cmd_buff, "ATCE %d\r", CE);
                        delay1();
                        sendCommand();
                        printf("\nDevice set");
                        delay1();
                        readResponse();
```

```
#endif

#ifdef RR
	sprintf(cmd_buff, "ATRR %d\r", RR);
	delay1();
	sendCommand();
	printf("\nRetries set");
	delay1();
	readResponse();
#endif

#ifdef RN
	sprintf(cmd_buff, "ATRN %d\r", RN);
	delay1();
	sendCommand();
	printf("\nCollision Avoid set");
	delay1();
	readResponse();
#endif

#ifdef NT
	sprintf(cmd_buff, "ATNT %d\r", NT);
	delay1();
	sendCommand();
	printf("\nDiscover time set");
	delay1();
	readResponse();
#endif

#ifdef NO
	sprintf(cmd_buff, "ATNO %d\r", NO);
	delay1();
	sendCommand();
	printf("\nNode discover set");
	delay1();
	readResponse();
#endif

#ifdef SC
	sprintf(cmd_buff, "ATSC %x\r", SC);
	delay1();
	sendCommand();
	printf("\nScan channel set");
	delay1();
	readResponse();
#endif

#ifdef SD
	sprintf(cmd_buff, "ATSD %d\r", SD);
	delay1();
	sendCommand();
	printf("\nScan duration set");
	delay1();
	readResponse();
#endif

#ifdef PL
	sprintf(cmd_buff, "ATPL %d\r", PL);
	delay1();
	sendCommand();
	printf("\nO/P power set");
	delay1();
	readResponse();
#endif
```

```
#ifdef CA
        sprintf(cmd_buff, "ATCA %x\r", CA);
        delay1();
        sendCommand();
        printf("\nCCA set");
        delay1();
        readResponse();
#endif

#ifdef END_DEVICE
#ifdef A1
        sprintf(cmd_buff, "ATA1 %d\r", A1);
        delay1();
        sendCommand();
        printf("\nInactive period set");
        delay1();
        readResponse();
#endif
#endif

        #ifdef END_DEVICE
#ifdef SM
        sprintf(cmd_buff, "ATSM %d\r", SM);
        delay1();
        sendCommand();
        printf("\nInactive period set");
        delay1();
        readResponse();
#endif
#endif

#ifdef END_DEVICE
#ifdef ST
        sprintf(cmd_buff, "ATST %d\r", ST);
        delay1();
        sendCommand();
        printf("\nInactive period set");
        delay1();
        readResponse();
#endif
#endif

#ifdef END_DEVICE
#ifdef SP
        sprintf(cmd_buff, "ATSP %x\r", SP);
        delay1();
        sendCommand();
        printf("\nCyclic Sleep set");
        delay1();
        readResponse();
#endif
#endif

#ifdef END_DEVICE
#ifdef DP
        sprintf(cmd_buff, "ATDP %x\r", DP);
        delay1();
        sendCommand();
        printf("\nSleep period set");
        delay1();
        readResponse();
#endif
#endif
```

```
#ifdef END_DEVICE
#ifdef SO
        sprintf(cmd_buff, "ATSO %x\r", SO);
        delay1();
        sendCommand();
        printf("\nSleep options set");
        delay1();
        readResponse();
#endif
#endif

#ifdef RO
        sprintf(cmd_buff, "ATRO %d\r", RO);
        delay1();
        sendCommand();
        printf("\nInter-char delay set");
        delay1();
        readResponse();
#endif

#ifdef AP
        sprintf(cmd_buff, "ATAP %d\r", AP);
        delay1();
        sendCommand();
        printf("\nAP mode set");
        delay1();
        readResponse();
#endif

#ifdef BD
        sprintf(cmd_buff, "ATBD %d\r", BD);
        delay1();
        sendCommand();
        printf("\nBaudrate set");
        delay1();
        readResponse();
#endif

#ifdef NB
        sprintf(cmd_buff, "ATNB %d\r", NB);
        delay1();
        sendCommand();
        printf("\nParity set");
        delay1();
        readResponse();
#endif

#ifdef D8
        sprintf(cmd_buff, "ATD8 %d\r", D8);
        delay1();
        sendCommand();
        printf("\nPull-up resistor set");
        delay1();
        readResponse();
#endif

#ifdef D7
        sprintf(cmd_buff, "ATD7 %d\r", D7);
        delay1();
        sendCommand();
        printf("\nPull-up resistor set");
        delay1();
        readResponse();
```

```
#endif

#ifdef D6
	sprintf(cmd_buff, "ATD6 %d\r", D6);
	delay1();
	sendCommand();
	printf("\nPull-up resistor set");
	delay1();
	readResponse();
#endif

#ifdef D5
	sprintf(cmd_buff, "ATD5 %d\r", D5);
	delay1();
	sendCommand();
	printf("\nPull-up resistor set");
	delay1();
	readResponse();
#endif

#ifdef D4
	sprintf(cmd_buff, "ATD4 %d\r", D4);
	delay1();
	sendCommand();
	printf("\nPull-up resistor set");
	delay1();
	readResponse();
#endif

#ifdef D3
	sprintf(cmd_buff, "ATD3 %d\r", D3);
	delay1();
	sendCommand();
	printf("\nPull-up resistor set");
	delay1();
	readResponse();
#endif

#ifdef D2
	sprintf(cmd_buff, "ATD2 %d\r", D2);
	delay1();
	sendCommand();
	printf("\nPull-up resistor set");
	delay1();
	readResponse();
#endif

#ifdef D1
	sprintf(cmd_buff, "ATD1 %d\r", D1);
	delay1();
	sendCommand();
	printf("\nPull-up resistor set");
	delay1();
	readResponse();
#endif

#ifdef D0
	sprintf(cmd_buff, "ATD0 %d\r", D0);
	delay1();
	sendCommand();
	printf("\nPull-up resistor set");
	delay1();
	readResponse();
#endif
```

```
#ifdef PR
	sprintf(cmd_buff, "ATPR %x\r", PR);
	delay1();
	sendCommand();
	printf("\nPull-up resistor set");
	delay1();
	readResponse();
#endif

#ifdef IU
	sprintf(cmd_buff, "ATIU %d\r", IU);
	delay1();
	sendCommand();
	printf("\nUART datasent set");
	delay1();
	readResponse();
#endif

#ifdef IT
sprintf(cmd_buff, "ATIT %x\r", IT);
delay1();
sendCommand();
printf("\nNo.of samples set");
delay1();
readResponse();
#endif

#ifdef IC
sprintf(cmd_buff, "ATIC %x\r", IC);
delay1();
sendCommand();
printf("\nChange detect set");
delay1();
readResponse();
#endif

#ifdef IZ
sprintf(cmd_buff, "ATIR %x\r", IZ);
delay1();
sendCommand();
printf("\nSample rate set");
delay1();
readResponse();
#endif

#ifdef PO
sprintf(cmd_buff, "ATPO %d\r", PO);
delay1();
sendCommand();
printf("\nPWM0 function set");
delay1();
readResponse();
#endif

#ifdef P1
sprintf(cmd_buff, "ATP1 %d\r", P1);
delay1();
sendCommand();
printf("\nPWM1 function set");
delay1();
readResponse();
#endif
```

```
#ifdef PT
sprintf(cmd_buff, "ATIU %x\r", PT);
delay1();
sendCommand();
printf("\nTimeout PWM set");
delay1();
readResponse();
#endif

#ifdef RP
sprintf(cmd_buff, "ATRP %x\r", RP);
delay1();
sendCommand();
printf("\nPWM register set");
delay1();
readResponse();
#endif

#ifdef IA
sprintf(cmd_buff, "ATIA %x\r", IA);
delay1();
sendCommand();
printf("\nModule address set");
delay1();
readResponse();
#endif

#ifdef T0
sprintf(cmd_buff, "ATT0 %x\r", T0);
delay1();
sendCommand();
printf("\nTimeout value set");
delay1();
readResponse();
#endif

#ifdef T1
sprintf(cmd_buff, "ATT1 %x\r", T1);
delay1();
sendCommand();
printf("\nTimeout value set");
delay1();
readResponse();
#endif

#ifdef T2
sprintf(cmd_buff, "ATT0 %x\r", T2);
delay1();
sendCommand();
printf("\nTimeout value set");
delay1();
readResponse();
#endif

#ifdef T3
sprintf(cmd_buff, "ATT0 %x\r", T3);
delay1();
sendCommand();
printf("\nTimeout value set");
delay1();
readResponse();
#endif

#ifdef T4
```

```
sprintf(cmd_buff, "ATT0 %x\r", T4);
delay1();
sendCommand();
printf("\nTimeout value set");
delay1();
readResponse();
#endif

#ifdef T5
sprintf(cmd_buff, "ATT0 %x\r", T5);
delay1();
sendCommand();
printf("\nTimeout value set");
delay1();
readResponse();
#endif

#ifdef T6
sprintf(cmd_buff, "ATT0 %x\r", T6);
delay1();
sendCommand();
printf("\nTimeout value set");
delay1();
readResponse();
#endif

#ifdef T7
sprintf(cmd_buff, "ATT0 %x\r", T7);
delay1();
sendCommand();
printf("\nTimeout value set");
delay1();
readResponse();
#endif

#ifdef DD
sprintf(cmd_buff, "ATDD %x\r", DD);
delay1();
sendCommand();
printf("\nDevice Identifier set");
delay1();
readResponse();
#endif

#ifdef CT
sprintf(cmd_buff, "ATCT %x\r", CT);
delay1();
sendCommand();
printf("\nCommand mode timeout set");
delay1();
readResponse();
#endif

#ifdef GT
sprintf(cmd_buff, "ATGT %x\r", GT);
delay1();
sendCommand();
printf("\nSilence time set");
delay1();
readResponse();
#endif

#ifdef CC
sprintf(cmd_buff, "ATCC %x\r", CC);
```

```
		delay1();
		sendCommand();
		printf("\nChar to break set");
		delay1();
		readResponse();
		#endif

		sprintf(wr_buff, "ATWR\r");
			int j=0;
			while(wr_buff[j]!='\r')
			{
				USART_vSendByte(wr_buff[j]);
				j++;
			}
			USART_vSendByte(wr_buff[j]);
		delay1();
			while(RX_BUFFER_IS_FULL()) {};
			if(XBEE_RDR == 'O')
			{
				printf("\nO for WR");
			}
			 if( XBEE_RDR == 'K')
			 {
				printf("\nK for WR");
			}
		sprintf(cmd_buff, "ATCN\r");
		delay1();
		sendCommand();
		printf("\nExiting");
		delay1();
		readResponse();

		printf("Done: Series 1");
#endif

#ifdef XBEES2
				USART_vSendByte(0x2B);
				USART_vSendByte(0x2B);
				USART_vSendByte(0x2B);
				// wait for 'OK'
				SCI6.SSR.BIT.ORER = 0;
				while(RX_BUFFER_IS_FULL()) {};
				if(XBEE_RDR == 'O' )
				{
					printf("O for +++\n");
				}
				while(RX_BUFFER_IS_FULL()) {};
				if(XBEE_RDR == 'K')
				{
					printf("K for +++\n");
				}
	#ifdef ID
	sprintf(cmd_buff, "ATID %d\r", ID);
	delay1();
	sendCommand();
	printf("ID sent\n");
	delay1();
	readResponse();
	#endif

	#ifdef SC
	sprintf(cmd_buff, "ATSC %c\r", SC);
	delay1();
	sendCommand();
```

```
        printf("\nScan channel set");
        delay1();
        readResponse();
        #endif

        #ifdef DH
        sprintf(cmd_buff, "ATDH %x\r", DH);
        delay1();
        sendCommand();
        printf("\nDestination High set");
        delay1();
        readResponse();
        #endif

        #ifdef DL
        sprintf(cmd_buff, "ATDL %x\r", DL);
        delay1();
        sendCommand();
        printf("\nDestination Low set");
        delay1();
        readResponse();
        #endif

        #ifdef SD
        sprintf(cmd_buff, "ATSD %x\r", SD);
        delay1();
        sendCommand();
        printf("\nScan duration set");
        delay1();
        readResponse();
        #endif

        #ifdef ZS
        sprintf(cmd_buff, "ATZS %x\r", ZS);
        delay1();
        sendCommand();
        printf("\nZigbee stack set");
        delay1();
        readResponse();
        #endif

        #ifdef NJ
        sprintf(cmd_buff, "ATNJ %x\r", NJ);
        delay1();
        sendCommand();
        printf("\nNode Join set");
        delay1();
        readResponse();
        #endif

        #ifdef NW
        sprintf(cmd_buff, "ATNW %x\r", NW);
        delay1();
        sendCommand();
        printf("\nWatchdog set");
        delay1();
        readResponse();
        #endif

        #ifdef JV
        sprintf(cmd_buff, "ATJV %d\r", JV);
        delay1();
        sendCommand();
        printf("\nChannel verification set");
```

```
        delay1();
        readResponse();
        #endif

        #ifdef JN
        sprintf(cmd_buff, "ATJN %d\r", JN);
        delay1();
        sendCommand();
        printf("\nJoin notification set");
        delay1();
        readResponse();
        #endif

        #ifdef NI
        sprintf(cmd_buff, "ATNI %d\r", NI);
        delay1();
        sendCommand();
        printf("\nNode identifier set");
        delay1();
        readResponse();
        #endif

        #ifdef NH
        sprintf(cmd_buff, "ATNH %x\r", NH);
        delay1();
        sendCommand();
        printf("\nHop limit set");
        delay1();
        readResponse();
        #endif

        #ifdef BH
        sprintf(cmd_buff, "ATBH %x\r", BH);
        delay1();
        sendCommand();
        printf("\nTx radius set");
        delay1();
        readResponse();
        #endif

        #ifdef AR
        sprintf(cmd_buff, "ATAR %x\r", AR);
        delay1();
        sendCommand();
        printf("\naggregation time set");
        delay1();
        readResponse();
        #endif

        #ifdef NT
        sprintf(cmd_buff, "ATNT %x\r", NT);
        delay1();
        sendCommand();
        printf("\nNode discovery set");
        delay1();
        readResponse();
        #endif

        #ifdef NO
        sprintf(cmd_buff, "ATNO %x\r", NO);
        delay1();
        sendCommand();
        printf("\nNode discover set");
        delay1();
```

```
        readResponse();
        #endif

        #ifdef CR
        sprintf(cmd_buff, "ATCR %x\r", CR);
        delay1();
        sendCommand();
        printf("\nThreshold set");
        delay1();
        readResponse();
        #endif

        #ifdef PL
        sprintf(cmd_buff, "ATPL %d\r", PL);
        delay1();
        sendCommand();
        printf("\nO/P power set");
        delay1();
        readResponse();
        #endif

        #ifdef PM
        sprintf(cmd_buff, "ATPM %d\r", PM);
        delay1();
        sendCommand();
        printf("\nBoost mode set");
        delay1();
        readResponse();
        #endif

        #ifdef ST
        sprintf(cmd_buff, "ATST %d\r", ST);
        delay1();
        sendCommand();
        printf("\nInactive period set");
        delay1();
        readResponse();
        #endif

        #ifdef SP
        sprintf(cmd_buff, "ATSP %x\r", SP);
        delay1();
        sendCommand();
        printf("\nCyclic Sleep set");
        delay1();
        readResponse();
        #endif

        #ifdef SO
        sprintf(cmd_buff, "ATSO %x\r", SO);
        delay1();
        sendCommand();
        printf("\nSleep options set");
        delay1();
        readResponse();
        #endif

        #ifdef AP
        sprintf(cmd_buff, "ATAP %d\r", AP);
        delay1();
        sendCommand();
        printf("\nAP mode set");
        delay1();
        readResponse();
```

```
#endif

#ifdef BD
sprintf(cmd_buff, "ATBD %d\r", BD);
delay1();
sendCommand();
printf("\nBaudrate set");
delay1();
readResponse();
#endif

#ifdef NB
sprintf(cmd_buff, "ATNB %d\r", NB);
delay1();
sendCommand();
printf("\nParity set");
delay1();
readResponse();
#endif

#ifdef PR
sprintf(cmd_buff, "ATPR %x\r", PR);
delay1();
sendCommand();
printf("\nPull-up resistor set");
delay1();
readResponse();
#endif

#ifdef IC
sprintf(cmd_buff, "ATIC %x\r", IC);
delay1();
sendCommand();
printf("\nChange detect set");
delay1();
readResponse();
#endif

#ifdef IR
sprintf(cmd_buff, "ATIR %x\r", IR);
delay1();
sendCommand();
printf("\nSample rate set");
delay1();
readResponse();
#endif

#ifdef P0
sprintf(cmd_buff, "ATP0 %d\r", P0);
delay1();
sendCommand();
printf("\nDIO10 set");
delay1();
readResponse();
#endif

#ifdef P1
sprintf(cmd_buff, "ATP1 %d\r", P1);
delay1();
sendCommand();
printf("\nDIO11 set");
delay1();
readResponse();
#endif
```

```
#ifdef P2
sprintf(cmd_buff, "ATP2 %d\r", P2);
delay1();
sendCommand();
printf("\nDIO12 set");
delay1();
readResponse();
#endif

#ifdef RP
sprintf(cmd_buff, "ATRP %x\r", RP);
delay1();
sendCommand();
printf("\nPWM register set");
delay1();
readResponse();
#endif

#ifdef D0
sprintf(cmd_buff, "ATD0 %d\r", D0);
delay1();
sendCommand();
printf("\nAD0/DIO0 set");
delay1();
readResponse();
#endif

#ifdef D1
sprintf(cmd_buff, "ATD1 %d\r", D1);
delay1();
sendCommand();
printf("\nAD1/DIO1 set");
delay1();
readResponse();
#endif

#ifdef D2
sprintf(cmd_buff, "ATD2 %d\r", D2);
delay1();
sendCommand();
printf("\nAD2/DIO2 set");
delay1();
readResponse();
#endif

#ifdef D3
sprintf(cmd_buff, "ATD3 %d\r", D3);
delay1();
sendCommand();
printf("\nAD3/DIO3 set");
delay1();
readResponse();
#endif

#ifdef D4
sprintf(cmd_buff, "ATD4 %d\r", D4);
delay1();
sendCommand();
printf("\nDIO4set");
delay1();
readResponse();
#endif
```

```
        #ifdef D5
        sprintf(cmd_buff, "ATD5 %d\r", D5);
        delay1();
        sendCommand();
        printf("\nDIO5 set");
        delay1();
        readResponse();
        #endif

        #ifdef DD
        sprintf(cmd_buff, "ATDD %x\r", DD);
        delay1();
        sendCommand();
        printf("\nDevice Identifier set");
        delay1();
        readResponse();
        #endif

        #ifdef EE
        sprintf(cmd_buff, "ATEE %d\r", EE);
        delay1();
        sendCommand();
        printf("\nZigbee encryption set");
        delay1();
        readResponse();
        #endif

        #ifdef EO
        sprintf(cmd_buff, "ATEO %x\r", EO);
        delay1();
        sendCommand();
        printf("\nEncryption options set");
        delay1();
        readResponse();
        #endif

        #ifdef KY
        sprintf(cmd_buff, "ATKY %x\r", KY);
        delay1();
        sendCommand();
        printf("\nEncryption keys set");
        delay1();
        readResponse();
        #endif

        #ifdef SB
        sprintf(cmd_buff, "ATSB %d\r", SB);
        delay1();
        sendCommand();
        printf("\nUART stop bits set");
        delay1();
        readResponse();
        #endif

        #ifdef D7
        sprintf(cmd_buff, "ATD7 %d\r", D7);
        delay1();
        sendCommand();
        printf("\nDIO7 set");
        delay1();
        readResponse();
        #endif

        #ifdef D6
```

```
        sprintf(cmd_buff, "ATD6 %d\r", D7);
        delay1();
        sendCommand();
        printf("\nDIO6 set");
        delay1();
        readResponse();
        #endif

        #ifdef AO
        sprintf(cmd_buff, "ATAO %d\r", AO);
        delay1();
        sendCommand();
        printf("\nAPI register set");
        delay1();
        readResponse();
        #endif

        #ifdef SM
        sprintf(cmd_buff, "ATSM %d\r", SM);
        delay1();
        sendCommand();
        printf("\nSleep mode set");
        delay1();
        readResponse();
        #endif

        #ifdef SN
        sprintf(cmd_buff, "ATSN %x\r", SN);
        delay1();
        sendCommand();
        printf("\nSleep period set");
        delay1();
        readResponse();
        #endif

        #ifdef PO
        sprintf(cmd_buff, "ATPO %x\r", PO);
        delay1();
        sendCommand();
        printf("\nPoll rate set");
        delay1();
        readResponse();
        #endif

        #ifdef LT
        sprintf(cmd_buff, "ATLT %x\r", LT);
        delay1();
        sendCommand();
        printf("\nLED blink set");
        delay1();
        readResponse();
        #endif

        #ifdef DO
        sprintf(cmd_buff, "ATDO %x\r", DO);
        delay1();
        sendCommand();
        printf("\nBits set");
        delay1();
        readResponse();
        #endif

        #ifdef VP
        sprintf(cmd_buff, "ATV+ %x\r", VP);
```

```
	delay1();
	sendCommand();
	printf("\nSupply voltage set");
	delay1();
	readResponse();
	#endif

	sprintf(wr_buff, "ATWR\r");
	int j=0;
	while(wr_buff[j]!='\r')
	{
		USART_vSendByte(wr_buff[j]);
		j++;
	}
	USART_vSendByte(wr_buff[j]);
	delay1();
	while(RX_BUFFER_IS_FULL()) {};
	if(XBEE_RDR == 'O')
	{
		printf("\nO for WR");
	}
	if( XBEE_RDR == 'K')
	{
		printf("\nK for WR");
	}

		sprintf(cmd_buff, "ATCN\r");
		delay1();
		sendCommand();
		printf("\nExiting");
		delay1();
		readResponse();

	printf("Done:Series 2");
	#endif  //Series 2

	return 1;
}


#endif /* XBEECONFIG_H_ */
```