APPLICATION NOTE

AN422

Using the iDigi™ BL4S100 Add-On Kit ZB with the ConnectPort™ X4 Gateway

Introduction

The iDigi BL4S100 Add-On Kit ZB is designed for use with the iDigi X4 Starter Kit ZB. The starter kit uses Digi's ConnectPort™ X4 gateway with the iDigi Platform that links remote devices with enterprise business applications. The iDigi BL4S100 Add-On Kit ZB provides a BL4S100 single-board computer to add embedded wireless control.

The BL4S100 adds intelligent, programmable I/O to the iDigi Platform and offers localized control of devices connected to a ZigBee network. Instructive sample code is provided that can be used as a template for your own application.

The ConnectPort X4 gateway runs iDigi Dia (Device Integration Application). Dia simplifies the development of custom applications for remote monitoring and sensor networking. Dia is written in Python™, and runs on Digi's gateway devices.

The BL4S100 uses Dynamic C to manage and report I/O changes.

The iDigi BL4S100 Add-On Kit ZB provides **`.yml`** configuration files to build the Dia for the ConnectPort X4 gateway; the Add-On Kit also provides **`.c`** sample programs to run on the BL4S100. All you have to do is adapt the configuration to your environment, then build Dia using Python installed on your PC. Dia is uploaded to the ConnectPort, and a corresponding Dynamic C sample program runs on the BL4S100 — you can watch the results right away.

What Else You Will Need

Besides what is supplied with the Add-On Kit, you will need:

- PC to host Dynamic C - the PC must have an available USB port to program the BL4S100 and an Ethernet interface to the Digi ConnectPort X4 gateway or to the BL4S100.
- iDigi X4 Starter Kit ZB - This starter kit is sold separately from the iDigi BL4S100 Add-On Kit ZB.

Example Applications

There are a number of applications well-suited for the BL4S100 with an iDigi Platform. The following is a partial list:

- Low-cost wireless embedded control applications
- Remote monitoring of equipment, devices, locations
- Data-logging applications

Hardware Setup

The Getting Started instructions included with the Add-On Kit describes how to set up and program the BL4S100. This application note describes how the Digi ConnectPort™ X4 interfaces with the BL4S100 and its Demo Board on a ZigBee network.

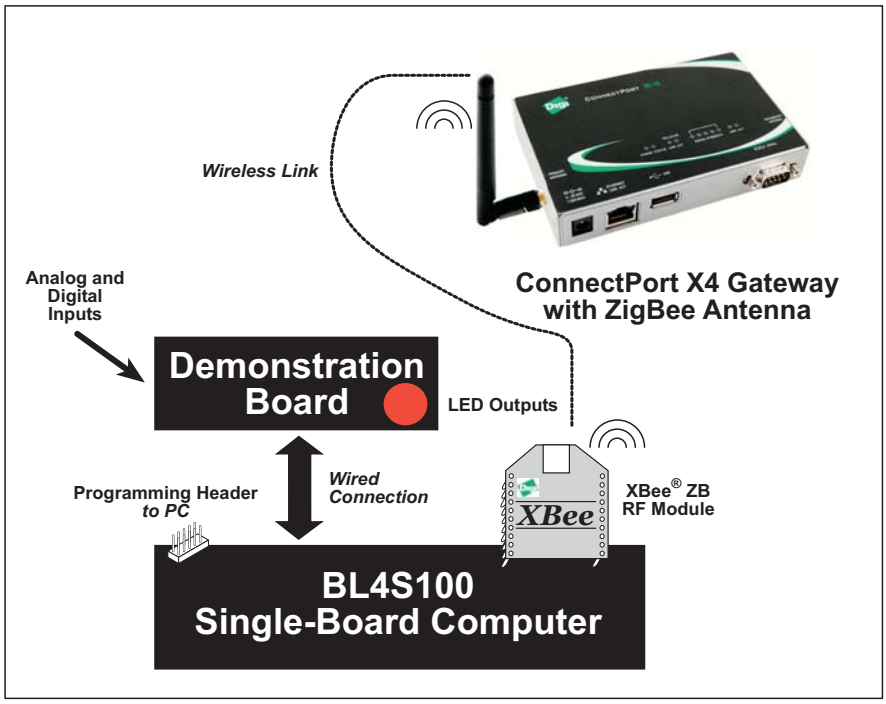


Figure 1. iDigi BL4S100 Add-On Kit ZB Setup

The Demo Board is connected to the BL4S100 via the wire assembly included in the iDigi BL4S100 Add-On Kit ZB.

Demo Board Connections

1. Use wires to connect screw-terminal header J3 on the Demo Board to header J4 on the BL4S100. The connections are shown in Figure 2, with the green wire to GND and the blue wire to +V.

   Figure 3 shows the I/O connections.

2. Make sure that your BL4S100 is connected to your PC via the programming cable and that the power supply is connected to the BL4S100 and plugged in as described in the ***iDigi™ BL4S100 Add-On ZB Kit Getting Started*** instructions.

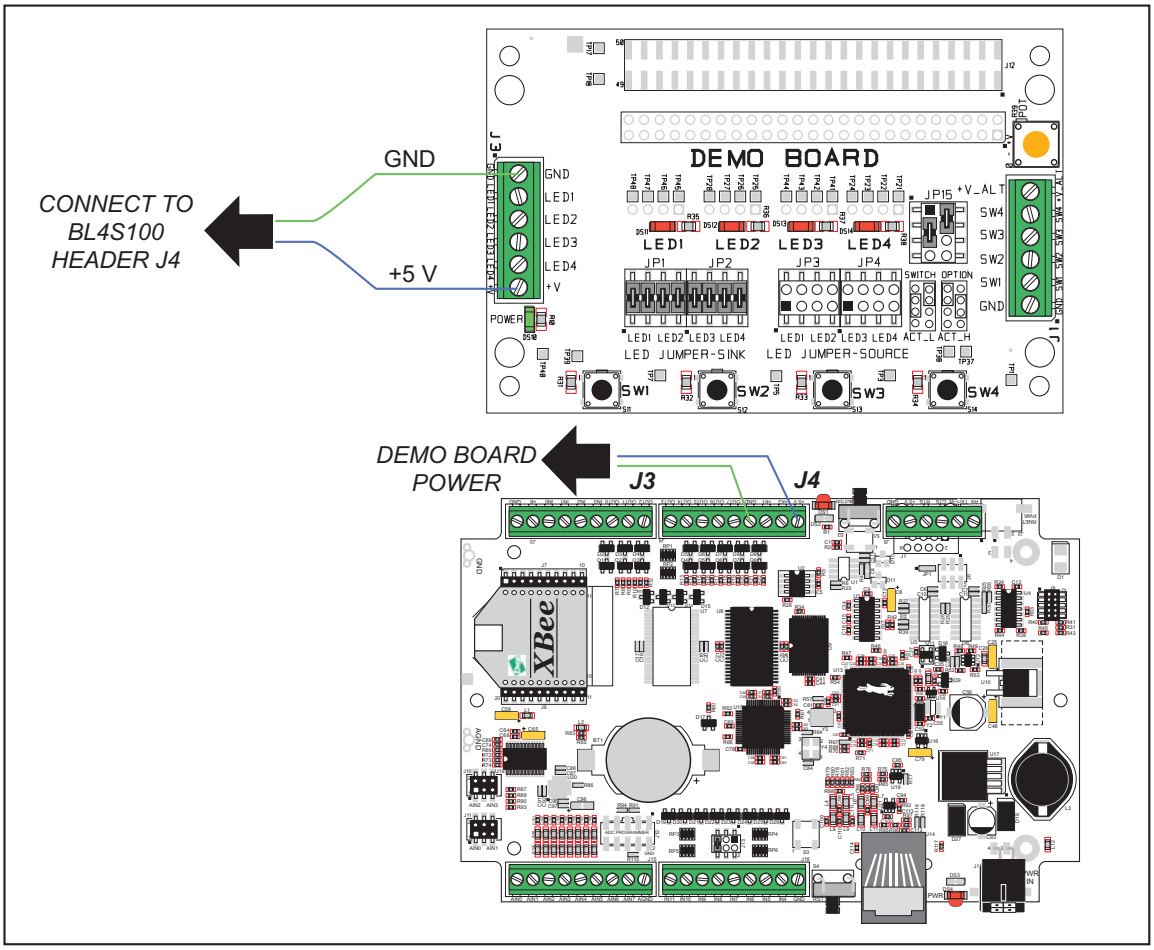


Figure 2. Power Supply Connections Between BL4S100 and Demo Board

CAUTION: If you are using your own power supply with the Demo Board, note that the maximum power supply input voltage the Demo Board can handle is + 12 V DC. Do not use a higher power supply voltage.

The Demo Board can be used to illustrate I/O activity via LEDs and pushbutton switches. To run the sample programs, the LEDs and switches on the Demo Board must be wired to the BL4S100. Complete wiring information is provided at the top of each relevant sample program file.

Figure 3 shows the pinouts for the input signals on screw-terminal header J1 and the outputs on screw-terminal header J3. It also shows all the I/O connections between the Demo Board and the BL4S100

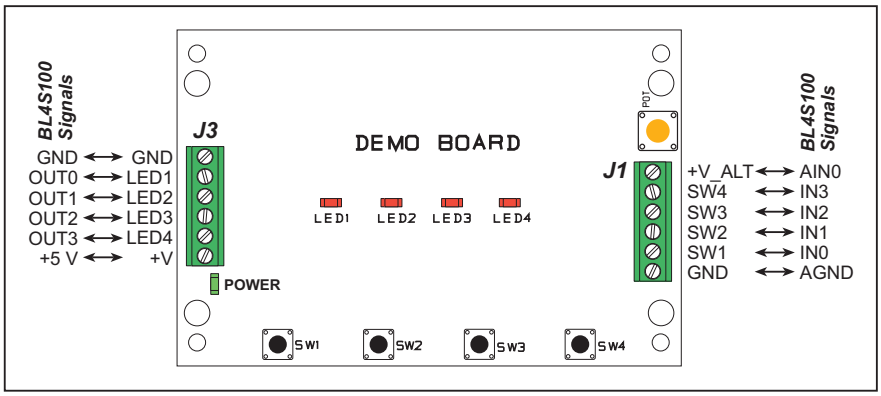


Figure 3. Demo Board I/O Connections

Table 1 provides the locations of the I/O signals from the BL4S100.

Table 1. Demo Board Connections to BL4S100

BL4S100 Header	Signals
Header J3	IN0–IN3
	OUT0–OUT2
Header J4	OUT3, +5 V, GND
Header J15	AIN0, AGND

NOTE: Header J3 shown in the above table corresponds to J3 in Figure 2, not the J3 header shown in Figure 3.

NOTE: Both boards have multiple signals labeled GND. Use the GND closest to the signal(s) being grounded.

Figure 4 summarizes the Demo Board jumper settings required for the sample programs in the iDigi BL4S100 Add-On Kit ZB.

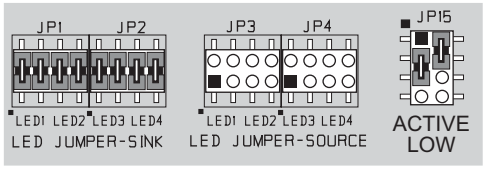


Figure 4. Demo Board Jumper Settings

Refer to Appendix C of the ***BL4S100 User's Manual*** for complete information on configuring and using the Demo Board.

Identify ZigBee Network Settings

The ***iDigi X4 Starter Kit Getting Started Guide*** explains how to set up the ConnectPort X4 hardware and set up a network based on the devices included with the iDigi X4 Starter Kit. The ConnectPort X4 gateway serves as the coordinator, and the BL4S100 is set up by default as a router.

The ConnectPort X4 will have a default extended PAN ID of 0x0000000000000000. Since the ConnectPort X4 gateway is the coordinator, this default extended PAN ID will result in the ConnectPort X4 picking a random PAN ID to use as its operating ID. Other devices on that ZigBee network must use the same extended PAN ID. These devices will be either routers or end devices, and the default extended PAN ID of 0x0000000000000000 will cause them to associate with the first ZigBee network they find and use that operating ID.

Unless you have more than one ConnectPort X4 gateway or have other nearby ZigBee networks, you may use the default extended PAN ID of 0x0000000000000000 with the BL4S100 and other devices. Otherwise, Digi recommends you make up your own extended PAN ID, which you can then apply to the ConnectPort X4, and use the same extended PAN ID for the BL4S100 and other devices.

If you have not previously configured your ConnectPort X4, you can use Digi Device Discovery Utility available for download on the Digi web site www.digigreen.com.

Once you have configured the ConnectPort X4, point your web browser at the ConnectPort's IP address.

1. In the menu at the right-hand side, click "Administration > System Information."
2. The "System Information" page will come up, and be open to the "General" horizontal tab. Click on the "XBee Network" horizontal tab.
3. At the top of this tab, there will be a table similar to:

   PAN ID: 0x14e2 - 0x1ff66966d5f3c750
   Channel: 0x0d (2415 MHz)
   Gateway Address: 00:13:a2:00:40:30:ff:0f!
4. The extended PAN ID is the 16 character number, 0x1ff66966d5f3c750, in the above example.
5. Also, make note of the Gateway Address. You will need this as the ConnectPort's extended address in some of the samples.

Dynamic C Setup

1. Start Dynamic C by double-clicking on the Dynamic C icon on your desktop or in your **Start** menu. Set the "Compiler" tab in the Dynamic C **Options > Project Options** menu to store the program in flash memory. Then click on the "Communications" tab and select the serial port used for the USB connection in the dropdown menu. Also, verify that "Use USB to Serial Converter" is selected to support the USB programming cable.
2. Click the "Defines" tab, and enter the following:

```
DEFAULT_EXTPANID = "enter your extended PAN ID here"
XBEE_GPIO_IOCHANGE
IOCHANGE_ADDR_IEEE = "enter your ConnectPort address here"
IOCHANGE_ENDPOINT = 0xDA
IOCHANGE_SECONDS = 5
```

The macros you entered have the following meanings:

- **DEFAULT_EXTPANID** is the extended PAN ID of the ZigBee network from the previous section.
- **IOCHANGE_ADDR_IEEE** is the extended address of the ConnectPort X4 gateway from the previous section. Note that this address must be specified as a string of octets. For example, if the extended address is "00:13:a2:00:40:0a:38:4d!" enter the following in the "Defines" tab:

  ```
  "\x00\x13\xa2\x00\x40\x0a\x38\x4d"
  ```

- The **IOCHANGE_SECONDS** and **IOCHANGE_ENDPOINT** can be left at their default values.

Click **OK**.

ConnectPort X4 Gateway Setup

To get ready to set up the ConnectPort X4 gateway, you will need the extended address of the XBee ZB RF module on the BL4S100, and you will need to install Python 2.4.3 on your PC to develop script that matches the 2.4.3 version of Python on the ConnectPort X4 gateway.

To get the extended address of the XBee ZB RF module, a sample program from the Dynamic C **Samples\iDigi\BL4S100** directory must first be running on the BL4S100.

Run a Rabbit Sample Program

Once the Rabbit Demo Board, USB programming cable and power are connected to the BL4S100, you are ready to run some Rabbit sample programs.

1. Use the **File** menu to open the sample program **gpio_endpoint.c**, which is in the Dynamic C **Samples\iDigi\BL4S100** directory.
2. Press function key **F9** to compile and run the program.

Use the ConnectPort's web interface to identify the extended address of the XBee ZB RF module on the BL4S100. Click under **Configuration > XBee Network** in the menu on the right-hand side to get the display that contains the extended address of the BL4S100. The **gpio_endpoint.c** sample sets the node id of the BL4S100 to "BL4S1xx GPIO Server." You may have to click **Refresh** if you do not see the BL4S100's node id.You should be able to copy and paste the Extended Address into the Dia configuration (see below).

Install Python Software

Since the ConnectPort X4 gateway runs Python script version 2.4.3, you must have Python 2.4.3 installed on your computer. You can download the correct version of Python from:

www.python.org/download/releases/2.4.3/

CAUTION: Python versions other than 2.4.3 will not work with the ConnectPort X4 gateway. Install the download in the default **C:\Python24** location to match the directory path expected by the build script used to prepare files to upload to the ConnectPort X4 gateway.

TIP: To access Python from the command window, you will need to update the **Path** environment variable to include the correct path to Python. Go to **Control Panel > System** and click on **Environmental Variables** on the "Advanced" tab. Then **Edit** the *System variables* to add the Python directory path **C:\Python24** to the **Path** line. Click **OK** to save the *System variables* settings and click **OK** again to save what you entered via the "Advanced" tab. If you choose not to add Python to your **Path**, you can still access Python by typing **C:\Python24\python**.

Once you have downloaded and set up the directory path to Python successfully, retrieve the Dia files from the www.digigreen.com page.

ConnectPort Software Setup

This setup focuses on the **gpio.yml** file. The setup is similar for the other **.yml** files.

1. Edit **demos\rabbit\gpio\gpio.yml** using the Python IDLE text editor.To access IDLE, open the Python shell (**Start > Python 2.4 > IDLE**), then do a **File > Open** from the Python shell. Since white space is significant in Python, do not use Dynamic C or other applications that may insert tabs instead of spaces. It is imperative that you preserve the existing spacing.

   Set the **extended_address** parameter to the address of the XBee ZB RF module on your BL4S100 that you found using the ConnectPort's web interface. This parameter should already be in the correct format, i.e., digits with a **:** between each pair, and a **!** at the end, as in this example:

   ```
   "00:13:a2:00:40:0a:05:8b!"
   ```

   For right now, leave the other settings alone. For reference, there are a couple of options.

   - The Web browser can poll the Dia, and the Dia can poll the Rabbit (**poll_rate** under **gpio** is non-zero).
   - The Web browser can poll the Dia, and the Rabbit can push digital and analog data up to the Dia (to enable pushing data from the Rabbit, define **XBEE_GPIO_IOCHANGE** in each BL4S100 sample program).

   We will use the second option, so there is no need for the GPIO client driver on the ConnectPort X4 gateway to poll the XBee module on the BL4S100. However, if you decide to change this or any other setting in **gpio.yml**, there is no need to recompile, as long as you do not add more devices.

2. Next build the **dia.zip** file for the **.yml** file. From the start menu, open a command window (**Start Menu > Run > cmd**), change to the location of the Dia directory at the cmd prompt, for example, **cd C:\Dia**, then enter the following cmd (you need to build a new **dia.zip** file each time you have a new **.yml** file or add/remove a device or presentation in the.**.yml** file)

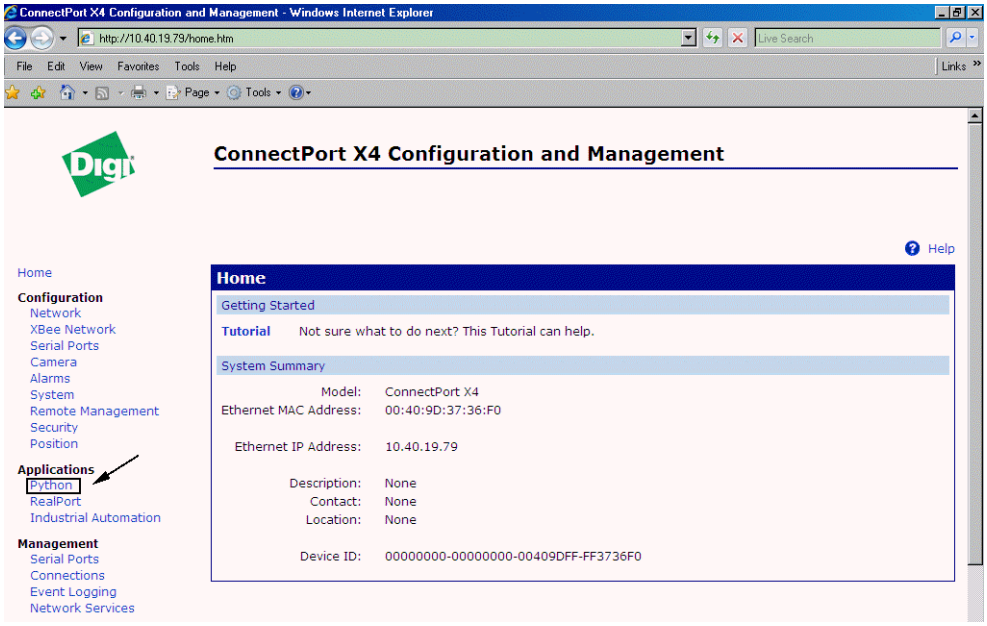
   ```
   python make.py demos\rabbit\gpio\gpio.yml
   ```

   You will see output similar to the following.

```
Analyzing files...
Compiling files...
Zipping files...
Finished writing archive bin\dia.zip
Adding 'demos\rabbit\gpio\gpio.yml' as 'dia.yml'...
```

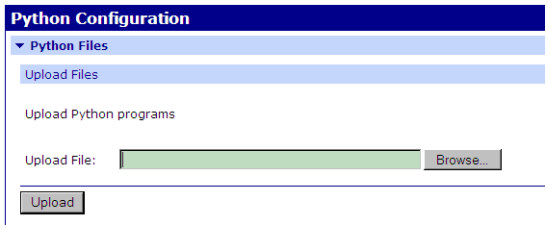
ConnectPort X4 Configuration and Management

Use the ConnectPort X4 Web interface in the Digi Device Discovery Application to upload the **`dia.zip`** and **`gpio.yml`** files to the ConnectPort X4 gateway.



1. Click on **Applications > Python** in the menus along the left side.

2. Next to the **Upload File**: field, click on **Browse**, and navigate to the location of your Dia installation, and then to the **`bin`** directory. Click on **`dia.zip`**, then click on **Open**, and then click the **Upload** button.
3. Follow the same steps to upload **`gpio.yml`** from the **`demos\rabbit\gpio`** directory.



4. Finally, upload **`dia.py`** from the Dia installation directory. You only need to do this once for a given version of the Dia.

Run Sample Program on ConnectPort X4

The sample program on the ConnectPort X4 is started via a Telnet connection from your PC.

1. Start a command session using your PC or workstation (**Run > cmd**).
2. At the cmd prompt, use Telnet to open the IP address of the ConnectPort X4 gateway; for example, **`telnet <IP address>`** where IP address is the address of your ConnectPort X4 gateway.
3. At the ConnectPort's #> prompt, type in **`py dia.py gpio.yml`**, and press **Enter**.

The output will resemble that shown below.

```
#> py dia.py gpio.yml

Determining platform type...Digi Python environment found.

Digi Device Integration Application Version 1.1.1

Using settings file: gpio.yml
Starting Channel Manager...
Starting Device Driver Manager...
XBeeDeviceManager(xbee_device_manager): node '00:13:a2:00:40:0a:00:c0!'
  moved to CONFIGURE state.
Starting Presentation Manager...
XBeeDeviceManager(xbee_device_manager): configuration done for node
  '00:13:a2:00:40:0a:00:c0!' promoting to RUNNING state.
XBeeGPIOClient: Discovery request sent: 1009988894.0
receive_info: io_count: 9
requesting names: ('<9B', 0, 1, 2, 3, 4, 5, 6, 7, 8) from endpoint 41
receive_names:  0 02 LED1
receive_names:  1 02 LED2
receive_names:  2 02 LED3
receive_names:  3 02 LED4
receive_names:  4 82 SW1
receive_names:  5 82 SW2
receive_names:  6 82 SW3
receive_names:  7 82 SW4
receive_names:  8 a3 AIN0
8 AIN0: unit: 'V' lower: 0.0 upper: 10.0
Core services started.
```

Do not quit the Telnet session since it will show what is happening when you use the Web browser next.

View Dia Presentation in Web Browser

Open your browser to http://<IP address>/dia.html, where <IP address> is the IP address of the Connect-Port X4 gateway. You should now be able to press a switch on the Demo Board, and see the corresponding change on the Web page; on the Web page, change an LED from On to Off or Off to On, then click **Apply Changes**, and watch the LEDs on the Demo Board toggle.

What Next?

Modify some of the settings. For example, turn off auto-update by the Rabbit (i.e., remove the macro **`XBEE_GPIO_IOCHANGE`** in the "Defines" tab of the Project Options dialog) and turn on auto-polling in the Dia GPIO driver (the "poll_rate" setting under "gpio" in **`gpio.yml`** should be at least 2 seconds for best results).

Sample Programs

Several sample programs are available to illustrate the sharing of data between the BL4S100 single-board computer and the ConnectPort X4 gateway. Note that corresponding sample programs must be run on each device as shown in Table 2. Follow the instructions provided in the sample programs to run the corresponding sample program on its device.

Table 2. List of Corresponding Sample Programs

BL4S100 Sample Programs	ConnectPort X4 Sample Programs	Comments
gpio_custom_control.c gpio_endpoint.c gpio_rabbitweb.c	gpio.yml	Requires Demo Board to be connected to the BL4S100
named_attributes.c	named_attributes.yml simple_chat.yml	
gpio_solar_farm.c	gpio.yml	Demo Board not needed
simple_chat.c simple_strings.c	simple_chat.yml simple_chat.py	

NOTE: The **simple_chat.py** sample program is available in the Dynamic C **Samples\iDigi\BL4S100\python** directory

The BL4S100 sample programs are on the Dynamic C CD, and can be found in the Dynamic C **Samples\iDigi\BL4S100** directory. The ConnectPort X4 sample programs are downloaded with the Dia files from the www.digigreen.com page, and can be found in the **demos\rabbit** directory.

The rest of this section discusses the BL4S100 sample programs. Click on **Restart device** in the Digi Device Discovery Application or **Administration > Reboot** on the ConnectPort Web page to restart the ConnectPort X4 gateway before trying out each sample program.

GPIO Sample Programs

The **gpio.yml** sample program from the **demos\rabbit\gpio** folder, **dia.zip**, and **dia.py** must be installed on the ConnectPort X4 to use it with the following BL4S100 sample programs.

Before running a new Dynamic C sample program, the ConnectPort must be rebooted, and Dia restarted. To restart Dia, telnet to the ConnectPort, and type:

```
python dia.py gpio.yml
```

- **gpio_custom_control.c**—This sample program shows how to use the XBee GPIO server endpoint to manage digital outputs on the BL4S100. It uses the four digital outputs connected to LEDs on the Demo Board. Pressing pushbutton switches on the Demo Board toggles the LEDs, and the GPIO client on the ConnectPort X4 can read and write those outputs.

  Once you see the LED toggling with the Web browser, you can go to port 4146 on the ConnectPort X4 via a Telnet session to interact with the Dia console. The following is an example of how to enter the IP address and port at the cmd prompt.

  ```
  telnet <IP address> 4146
  ```

- **gpio_endpoint.c**—This sample program shows how to use the XBee GPIO server endpoint to manage I/O on the BL4S100. It uses the four digital inputs connected to switches, four digital outputs connected to LEDs, and one analog input connected to a potentiometer on the Demo Board.

  Once you see the LED toggling with the Web browser, you can go to port 4146 on the ConnectPort X4 via a Telnet session to interact with the Dia console. The following is an example of how to enter the IP address and port at the cmd prompt.

  ```
  telnet <IP address> 4146
  ```

- **gpio_rabbitweb.c**—This sample program shows how to use the XBee GPIO server endpoint to manage I/O on the BL4S100. It uses all the digital and analog I/O on the BL4S100. Four of the outputs are configured to be connected to LEDs and four inputs are configured to be connected to pushswitches such as those on the Demo Board.

  You will need to connect the RJ-45 jack on the BL4S100 to your network (or directly to your PC) to connect to a Web server running on the Rabbit before you compile and run this sample program. Use the IP address provided in the "Connect your browser to http://..." message in the Dynamic C **STDIO** window for the IP address to enter in your Web browser. Changes made on the Dia Web interface will be reflected on the RabbitWeb interface, and vice versa.

  Instead of using a static IP address on the "Defines" tab under **Options > Project Options**, you may use DHCP by changing the **#define TCPCONFIG 1** line to **#define TCPCONFIG 5**. No IP address entry is then needed on the Dynamic C "Defines" tab of **Options > Project Options**.

  Observe the LEDs toggling with the Web browser. Pressing the pushbutton switches on the Demo Board or adjusting the potentiometer will change the values on the Web page. Toggling an LED to Off or On (or 0 or 1) will turn the LED off or on the Demo Board.

  If you have a second Ethernet jack, hub, switch, or a second PC, use it to connect to the RJ-45 jack on the ConnectPort X4 gateway; otherwise swap your Ethernet cable from the BL4S100. Open the Web browser to http://<IP address>/dia.html, where <IP address> is the IP address of the ConnectPort X4 gateway, and again observe the LEDs toggling.

  Setting the "refresh" channel to a positive number of X seconds causes the Rabbit to update the digital values on the ConnectPort automatically, as well as send analog data every X seconds. Set the "refresh" to a minimum of 2 seconds.

  You can also go to port 4146 on the ConnectPort X4 via a Telnet session to interact with the Dia console. The following is an example of how to enter the IP address and port at the cmd prompt.

  ```
  telnet <IP address> 4146
  ```

  Changes made via the console will be reflected on both of the two Web interfaces, and vice versa.

- **gpio_solar_farm.c**—This sample program shows how to use the XBee GPIO server endpoint to control digital and analog I/O on the BL4S100. Instead of being tied to physical I/O pins on the BL4S100, the GPIO client on the ConnectPort X4 gateway accesses virtual I/O signals that are updated by a function call in the sample program.

  Setting the "refresh" channel to a positive number of X seconds causes the Rabbit to update the channel values on the ConnectPort automatically, as well as send analog data every X seconds. Set the "refresh" to a minimum of 2 seconds.

  You can use the Dynamic C **STDIO** window to interact with the sample program. Press the 1, 2 and 3 keys to change the weather. Use the Z and X keys to rotate the virtual solar panel to track the sun and produce more electricity.

You can also use a Telnet session on the ConnectPort X4 to interact with the Dia console. The following is an example of how to enter the IP address and port at the cmd prompt.

```
telnet <IP address> 4146
```

Changes made via the console will be reflected on the Web interface, and vice versa.

Other Sample Programs

Before a different configuration of Dia can be run (i.e., a different **.yml** file), Dia must be rebuilt. After building Dia, upload **bin\dia.zip** and the **.yml** file to the ConnectPort.

Named Attributes Sample

The **named_attributes.yml** config file is used with the Dynamic C **named_attributes.c** program. Dia needs to be recompiled for this sample:

```
python make.py demos\rabbit\named_attributes\named_attributes.yml
```

The **named_attributes.yml** file from the **demos\rabbit\named_attributes** folder and **bin\dia.zip** must be installed on the ConnectPort X4 to use with the **named_attributes.c** BL4S100 sample program; **dia.py** needs to be uploaded only once and is likely to have been uploaded already.

- **named_attributes.c**— This sample program shows how to set up and use an endpoint with the Dynamic C **XBEE_API.LIB** library in order to report values for multiple “named attributes” and allow a remote device to request changes to those values.

  Clients can get and (for read/write attributes) set the attributes by name. Requests are sent as plain text, using the following format:

Requests (sent to the Rabbit):

name? — Sends the value of attribute <name>.
name — Sends the value of attribute <name> (? operator is implied).
name=value — Set attribute <name> to <value>.

In requests, name can end with an asterisk (*) as a wildcard to match any remaining characters. For example, "led*?" will match "led1", "led2", "led3", and "led4" (and therefore generate four responses).

Wildcards work for setting values as well (for example, dout*=1 to set all digital outputs to 1).

NOTE: Limits of the frame buffering between the BL4S100 and its XBee ZB RF module may cause some responses to be dropped if too many attributes are requested at once (that is, via the auto refresh Web page).

Responses (from the Rabbit):

name! — Error response, no attribute called <name>.
name!value — Error response, unable to set <name> to <value>.
name:value — Successful response, attribute <name> is set to <value>. (Sent in response to query (?) or assign (=) operators.)

Observe the LEDs with the Web browser. Pressing the pushbutton switches on the Demo Board or adjusting the potentiometer will change the display on the Web page. Setting an LED to Off or On (or 0 or 1) will turn the LED off or on the Demo Board.

Go to port 4146 on the ConnectPort X4 via the Telnet session to interact with the Dia console. The following is an example how to enter the IP address and port at the cmd prompt.

```
telnet <IP address> 4146
```

Changes made via the console will be reflected on the Web interface, and vice versa.

The **named_attributes.c** sample program uses the same mechanism for transport as do the samples **simple_strings.c** and **simple_chat.c**, but has a set of formatted message types to convey specific information. Therefore, **named_attributes.c** can also be used with **simple_chat.yml** to allow you to experiment with the named attributes protocol.

Simple Chat Sample

The **simple_chat.yml** config file is used with the Dynamic C **simple_chat.c** sample program. Dia needs to be recompiled for this sample:

```
python make.py demos\rabbit\simple_chat\simple_chat.yml
```

The **simple_chat.yml** file from the **demos\rabbit\simple_chat** folder and **dia.zip** must be installed on the ConnectPort X4 to use it with the **simple_chat.c** and **simple_strings.c** BL4S100 sample programs; **dia.py** needs to be uploaded only once and is likely to have been uploaded already.

The **simple_strings.c** and **simple_chat.c** sample programs are similar; **simple_chat.c** is an easier starting point to write your own application.

The **simple_chat.py** script may be used instead of **simple_chat.yml** to give a non-Dia example of how to interface with the XBee on the ConnectPort X4 gateway. It uses the ConnectPort as a Zigbee-TCP bridge and sends messages between the two.

- **simple_chat.c**— This sample program shows how to set up and use endpoints with the Dynamic C **XBEE_API.LIB** library in order to send simple strings between ZigBee-enabled devices.

  Use the Web browser. to see the last string sent from the BL4S100 and enter strings to send to the BL4S100.

  Go to port 4146 on the ConnectPort X4 via the Telnet session to interact with the Dia console. The following is an example how to enter the IP address and port at the cmd prompt.

  ```
  telnet <IP address> 4146
  ```

  Changes made via the console will be reflected on the Web interface, and vice versa.

- **simple_strings.c**— This sample program shows how to set up and use endpoints with the Dynamic C **XBEE_API.LIB** library in order to send simple strings between ZigBee-enabled devices.

  Use the Web browser. to see the last string sent from the BL4S100 and enter strings to send to the BL4S100.

  Go to port 4146 on the ConnectPort X4 via the Telnet session to interact with the Dia console. The following is an example how to enter the IP address and port at the cmd prompt.

  ```
  telnet <IP address> 4146
  ```

  Changes made via the console will be reflected on the Web interface, and vice versa.

Appendix — Reference Information

Sample Program Walkthrough

Let's examine some of the code in the **gpio_rabbitweb.c** sample program.

XBee ZB RF Module Configuration Macros

The XBee ZB RF module defaults to a router.

```
#define XBEE_ROLE NODE_TYPE_ROUTER
```

The XBee ZB RF module may also be configured as a coordinator or an end device by installing the appropriate firmware and defining the macro to **NODE_TYPE_COORD** or **NODE_TYPE_ENDDEV**, respectively.

Set a Node ID to identify this node on the network. The Node ID is a string of up to 20 characters.

```
#define NODEID_STR "XBee GPIO RabbitWeb"
```

Set the gain appropriately for the voltage range on the analog inputs. The GAIN_X2 macro from the **BL4S1XX.LIB** library corresponds to a voltage range of 0–20 V, See the ***BL4S100 User's Manual*** for more information.

```
DEMO_GAIN GAIN_X2
```

Enable I/O change notification feature of the GPIO endpoint

```
#define XBEE_GPIO_IOCHANGE
```

The following user-defined macros set fields in the "Device Info" response.

```
#define XBEE_GPIO_MANUFACTURER 0x101e
#define XBEE_GPIO_FIRMWARE_VER 0x0100
#define XBEE_GPIO_DEVICE_TYPE 0xB100
```

Network Configuration Macros

The network configuration used at startup is defined. These macros are used by the parameters in the function calls, and you may change the macro definitions to suit your needs.

```
#define TCPCONFIG 1
```

The **TCPCONFIG** macro tells Dynamic C to select your configuration from a list of default configurations in the Dynamic C **LIB\TCPIP\TCP_CONFIG.LIB** library. The usual default of 1 for the **TCPCONFIG** macro will set the IP configuration to **10.10.6.100**, the netmask to **255.255.255.0**, and the nameserver and gateway to **10.10.6.1**. Use a value of 5 for the **TCPCONFIG** macro to

determine the network parameters from a DHCP server on the network. Use function lookup (**Ctrl-H**) on **TCPCONFIG** for additional instructions on setting the TCP/IP networking configuration.

The HTTP server macros set up the RabbitWeb interface for the Web browser. The **USE_RABBITWEB** macro is defined to **1** to use the HTTP server enhancements. The **HTTP_MAXSERVERS** macro is set to **4** (four servers at a time).

```
#define USE_RABBITWEB 1
#define HTTP_MAXSERVERS 4
```

Other Sample Program Information

Libraries that are not in **DEFAULT.H** or brought in by other libraries must have a **#use** in the sample program.

```
#use "XBee_GPIO_Server.lib"
#use "XBee_GPIO_BL4S100.lib"
#use "BL4S1xx.lib"
```

The I/O channel setup is described using names up to 20 characters long.

```
XBEE_GPIO_CONFIG_START
   // Digital Outputs       Type,        "Name", I/O Channel
   XBEE_GPIO_CONFIG_ENTRY( LED_OUT,     "LED1",     0 ),
   XBEE_GPIO_CONFIG_ENTRY( LED_OUT,     "LED2",     1 ),
   XBEE_GPIO_CONFIG_ENTRY( LED_OUT,     "LED3",     2 ),
   XBEE_GPIO_CONFIG_ENTRY( LED_OUT,     "LED4",     3 ),
   XBEE_GPIO_CONFIG_ENTRY( DIGITAL_OUT, "OUT4",     4 ),
   XBEE_GPIO_CONFIG_ENTRY( DIGITAL_OUT, "OUT5",     5 ),
   XBEE_GPIO_CONFIG_ENTRY( DIGITAL_OUT, "OUT6",     6 )
   XBEE_GPIO_CONFIG_ENTRY( DIGITAL_OUT, "OUT7",     7 ),

   // Digital Inputs
   XBEE_GPIO_CONFIG_ENTRY( SWITCH_IN,    "SW1",     0 ),
   XBEE_GPIO_CONFIG_ENTRY( SWITCH_IN,    "SW2",     1 ),
   XBEE_GPIO_CONFIG_ENTRY( SWITCH_IN,    "SW3",     2 ),
   XBEE_GPIO_CONFIG_ENTRY( SWITCH_IN,    "SW4",     3 ),
   XBEE_GPIO_CONFIG_ENTRY( DIGITAL_IN,   "IN4",     4 ),
   XBEE_GPIO_CONFIG_ENTRY( DIGITAL_IN,   "IN5",     5 ),
   XBEE_GPIO_CONFIG_ENTRY( DIGITAL_IN,   "IN6",     6 ),
   XBEE_GPIO_CONFIG_ENTRY( DIGITAL_IN,   "IN7",     7 ),
   xBEE_GPIO_CONFIG_ENTRY( DIGITAL_IN,   "IN8",     8 ),
   XBEE_GPIO_CONFIG_ENTRY( DIGITAL_IN,   "IN9",     9 ),
   XBEE_GPIO_CONFIG_ENTRY( DIGITAL_IN,  "IN10",    10 ),
   XBEE_GPIO_CONFIG_ENTRY( DIGITAL_IN,  "IN11",    11 ),

   // Analog Inputs
   XBEE_GPIO_CONFIG_ENTRY( ANALOG_IN,   "AIN0",     0 ),
   XBEE_GPIO_CONFIG_ENTRY( ANALOG_IN,   "AIN1",     1 ),
   XBEE_GPIO_CONFIG_ENTRY( ANALOG_IN,   "AIN2",     2 ),
   XBEE_GPIO_CONFIG_ENTRY( ANALOG_IN,   "AIN3",     3 ),
   XBEE_GPIO_CONFIG_ENTRY( ANALOG_IN,   "AIN4",     4 ),
   XBEE_GPIO_CONFIG_ENTRY( ANALOG_IN,   "AIN5",     5 ),
   XBEE_GPIO_CONFIG_ENTRY( ANALOG_IN,   "AIN6",     6 ),
   XBEE_GPIO_CONFIG_ENTRY( ANALOG_IN,   "AIN7",     7 ),
```

The **XBEE_GPIO_IOCHANGE** is defined to include code to send all I/O changes automatically to a particular client.

```
#ifdef XBEE_GPIO_IOCHANGE
   #define CH_REFRESH 100
   XBEE_GPIO_CONFIG_ENTRY( ANALOG_OUT, "refresh", CH_REFRESH ),
#endif
```

The endpoint table defines what endpoints the application has and what cluster IDs are associated with it. Each endpoint is associated with a profile and a device. The device is not important in ZigBee, but the profile is very important. A profile defines what commands and data formats and functions are available on a network. Any device implementing some or all of these items may join the network and be used or use other devices.

A network may support more than one profile.

Applications using the **XBEE_API.LIB** library must define an endpoint table, even if it is empty.

```
ENDPOINT_TABLE_BEGIN
   // start with any user-defined endpoints
   // ENDPOINT_TABLE_ENTRY(Endpt, desc, profile, device, flags,
   //  IC, OC, ICL, OCL),
   // include the XBee GPIO endpoint
   XBEE_GPIO_ENDPOINT
ENDPOINT_TABLE_END
```

The background pages for the Web pages associated with the RabbitWeb display are imported.

```
#ximport "pages/iosample.html"  iosample_html
#ximport "pages/iovalues.html"  iovalues_html
#ximport "pages/iosample.css"   iosample_css
#ximport "pages/rabbitlogo.png" rabbitlogo_png
```

These imported files are associated with Web server URLs.

```
SSPEC_RESOURCETABLE_START
   SSPEC_RESOURCE_XMEMFILE("/", iosample_html),
   SSPEC_RESOURCE_XMEMFILE("/iovalues.html", iovalues_html),
   SSPEC_RESOURCE_XMEMFILE("/iosample.css", iosample_css),
   SSPEC_RESOURCE_XMEMFILE("/rabbitlogo.png", rabbitlogo_png)
SSPEC_RESOURCETABLE_END
```

Function Reference Guide

Function for use with the XBee ZB RF modules are in the Dynamic C **LIB\Rabbit4000\XBee\XBEE_API.LIB** library. These ZigBee-specific functions are described in ***An Introduction to ZigBee***, which is included in the online documentation set.

Functions specific to the iDigi BL4S100 Add-On Kit ZB are described in this Application Note.

XBee_GPIO_BL4S100 Library

The Dynamic C **LIB\Rabbit4000\iDigi\XBee_GPIO_BL4S100.LIB** library supports running the XBee GPIO server on a BL4S100 single-board computer. As long as your application is only presenting physical I/O on the BL4S100 to the XBee GPIO server interface, you can have this library handle all output changes by defining three macros and **#use** this library in your application:

Macros

XBEE_GPIO_DISABLE_ANALOG	This macro disables the analog inputs, making them unavailable to the GPIO endpoint.
XBEE_GPIO_SWITCH_CLOSED	**XBEE_GPIO_INPUT_LOW** (default value) if digital input is low when switch is closed.
	XBEE_GPIO_INPUT_HIGH if digital input is high when switch is closed.
XBEE_GPIO_LED_ON	**XBEE_GPIO_OUTPUT_SINK** (default value) if LED is active low (lit when output is low)
	XBEE_GPIO_OUTPUT_SOURCE if LED is active high (lit when output is high).

hw_set_digital_out

```
int hw_set_digital_out (int signal, int value);
```

DESCRIPTION

Hardware-specific function for setting digital (or tristate) output.

PARAMETERS

signal Signal number to update, used to look up I/O channel in xbee_gpio_config.

value New value for the digital output, should be one of the following:

XBEE_GPIO_OUTPUT_LOW
XBEE_GPIO_OUTPUT_HIGH
XBEE_GPIO_OUTPUT_OFF
XBEE_GPIO_OUTPUT_ON
XBEE_GPIO_OUTPUT_SINK
XBEE_GPIO_OUTPUT_SOURCE
XBEE_GPIO_OUTPUT_TRISTATE

RETURN VALUE

0 = output set.

-EINVAL = chosen signal is invalid or not a digital output; new value for signal is invalid for its signal type.

-EPERM = channel is not configured as a digital output.

hw_set_analog_out

```
int hw_set_analog_out (int signal, float value);
```

DESCRIPTION

Hardware-specific function for setting analog output.

PARAMETERS

signal	Signal number to update, used to look up I/O channel in xbee_gpio_config.
value	New value for analog output.

RETURN VALUE

-EINVAL: the specific BL4S100 does not have analog outputs; this is just a stub function call that always returns an error.

xbee_gpio_read_inputs

```
void xbee_gpio_read_inputs();
```

DESCRIPTION

Reads analog and digital inputs, and pass updated values to **xbee_gpio_set_{dig | ana}** in functions.

Should only be called after xbee_gpio_hw_init().

xbee_gpio_hw_init

```
void xbee_gpio_hw_init();
```

DESCRIPTION

Initializes I/O pins on the BL4S100 using configuration set with **XBEE_GPIO_CONFIG_START**, **_ENTRY**, and **_END** macros.

Associates the analog I/O ranges with their appropriate signals in the **xbee_gpio** structure.

XBee_GPIO_Server Library

The Dynamic C **LIB\Rabbit4000\iDigi\XBee_GPIO_SERVER.LIB** library implements a GPIO endpoint server using a protocol documented in this library. Your application will need to set up some tables, define some macros, and provide some support functions in order to use this library.

You will probably find it easiest to use one of the sample programs in the **Samples/iDigi/BL4S100** directory as a starting point for your own application, or at least as an example of the required code to make use of this library.

When your application sets up the endpoint table using the **ENDPOINT_TABLE_BEGIN**, **_ENTRY**, and **_END** macros in the **XBEE_API.LIB** library, you will need to include **XBEE_GPIO_ENDPOINT** in the list to register the GPIO server's endpoint handlers with the library:

```
ENDPOINT_TABLE_BEGIN
   // start with any user-defined endpoints
   // Endpt, desc, profile, device, flags, IC, OC, ICL, OCL)
   ENDPOINT_TABLE_ENTRY(2, 0, 42, 1, 0, 1, 0, &StringInCluster, NULL),
   // include the XBee GPIO Endpoint
   XBEE_GPIO_ENDPOINT
ENDPOINT_TABLE_END
```

You need to provide two support functions for this library to call after validating a request to change a digital or analog output. These functions are where your application actually changes the physical outputs:

```
int xbee_gpio_hw_digout(int signal, int value);
int xbee_gpio_hw_anaout(int signal, float value);
```

One of the following values will be passed to **xbee_gpio_hw_digout()**:

```
XBEE_GPIO_OUTPUT_LOW
XBEE_GPIO_OUTPUT_HIGH
XBEE_GPIO_OUTPUT_OFF
XBEE_GPIO_OUTPUT_ON
XBEE_GPIO_OUTPUT_SINK
XBEE_GPIO_OUTPUT_SOURCEXBEE_GPIO_OUTPUT_TRISTATE
```

Both functions should return 0 if the change was successful, or any non-zero value to indicate an error (and to cancel the change).

You need to set up a table to describe the I/O signals made available via the GPIO endpoint. You should use the following macros to define the **xbee_gpio_config** table:

```
XBEE_GPIO_CONFIG_START
XBEE_GPIO_CONFIG_ENTRY(type, name, channel)
XBEE_GPIO_CONFIG_END
```

PARAMETERS

type — one of the following.

Special Types:
DISABLED — I/O signal that cannot be read or written

Outputs:
DIGITAL_OUT — output with two states, low and high
LED_OUT — output with two states, off and on
SINK_OUT — output set to sink or tristate
SOURCE_OUT — output set to source or tristate
TRISTATE_OUT — output set to sink, source, or tristate
ANALOG_OUT — general analog output

Inputs:
DIGITAL_IN — input with two states, low and high|
SWITCH_IN — input with two states, open and closed
ANALOG_IN — general analog input

name — a string used to identify the I/O signal to the GPIO client. This name can be from zero to 20 characters long.

channel — an 8-bit value used to map the I/O signal to a physical I/O on the device. It is not sent to the client, but can be used by the **xbee_gpio_hw_digout()** and **xbee_gpio_hw_anaout()** functions when updating the physical outputs.

If you are not using analog I/O on the GPIO endpoint, you need to have the following:

```
#define XBEE_GPIO_DISABLE_ANALOG
```

so the libraries do not generate compiler errors with references to the analog range table.

If the signal table includes any analog I/O, you need to describe their ranges by using the following macros to define the **xbee_gpio_ana_range** table.

```
XBEE_GPIO_ANA_RANGE_START
   XBEE_GPIO_ANA_RANGE_ENTRY(IN, ch, lower, upper, units, mode, gain)
   XBEE_GPIO_ANA_RANGE_ENTRY(OUT, ch, lower, upper, units, pol, initial_out)
XBEE_GPIO_ANA_RANGE_END
```

PARAMETERS

type	must be either **IN** or **OUT**.
ch	to the **XBEE_GPIO_ANA_RANGE_ENTRY** macro is used to match the entry to an analog input from the **xbee_gpio_config** table.
lower **upper**	define the valid range of values for this input.
units	a zero- to 15-byte string describing the units for the input (e.g., "V", "mA", "%").
mode **gain**	For inputs, integers that can be used by your program's startup code to initialize the inputs.
pol **initial_out**	For outputs, **pol** (an integer) and **initial_out** (float) can be used by your program's startup code to initialize the outputs.

Your application is also responsible for keeping a table of inputs up to date. This library provides two function for your application to report the current value of each digital and analog input:

```
xbee_gpio_set_digin
xbee_gpio_set_anain
```

Call **xbee_gpio_init()** at the start of your application, and call **xbee_gpio_tick()** on a regular basis to handle GPIO requests.

When updating the I/O values, your application can make use the macro **XBEE_GPIO_CONFIG_COUNT** to loop through signals in the **xbee_gpio_config** table that are defined by the X**BEE_GPIO_CONFIG_START**, **_ENTRY**, and **_END** macros.

You can read the following elements from each entry.

xbee_gpio_config[signal].type

The type defined by the macro **XBEE_GPIO_CONFIG_ENTRY**, prefixed with **XBEE_GPIO_TYPE_** (for example, **XBEE_GPIO_TYPE_DIGITAL_OUT**, **XBEE_GPIO_TYPE_LED_OUT**, **XBEE_GPIO_TYPE_ANALOG_IN**, **XBEE_GPIO_TYPE_SWITCH_IN**).

xbee_gpio_config[signal].name

Zero- to 20-byte null-terminated string defined by the **XBEE_GPIO_CONFIG_ENTRY** macro.

xbee_gpio_config[signal].channel

8-bit channel defined by the macro **XBEE_GPIO_CONFIG_ENTRY**.The **xbee_gpio** table provides access to current signal values and a pointer to analog ranges.

xbee_gpio[signal].value.digita

Current value of a digital signal.

xbee_gpio[signal].value.analog

Current value of an analog signal.

xbee_gpio[signal].range

For analog inputs and outputs, you can use **far xbee_gpio_ana_range_t *** to access the following elements:

```
float lower, upper; // upper and lower range, set to 0 if no range
char units[16];

// for inputs
int u.in.mode; // mode setting for ADC (from RANGE maco)
int u.in.gain; // gain setting for ADC (from RANGE macro)

// for outputs
int u.out.polarity; // polarity setting for DAC (from RANGE macro)
float u.out.initial_out; // starting value for DAC (from RANGE macro)
```

Optional Macros

XBEE_GPIO_MANUFACTURER	16-bit value sent in the Device Info Response. Defaults to 0x101E (Digi's ZigBee Manufacturer ID).
XBEE_GPIO_FIRMWARE_VER	16-bit value sent in the Device Info Response. Defaults to 0x0000.
XBEE_GPIO_DEVICE_TYPE	16-bit value sent in the Device Info Response. Defaults to 0x0000.
XBEE_GPIO_DISABLE_ANALOG	Define this macro if your program does not use of any of the analog I/O types.
XBEE_GPIO_IOCHANGE	Define this macro to include code to send all digital I/O changes automatically to a particular client. If analog I/O has been enabled, the library can send the status of all analog signals automatically at a fixed frequency (like every 5 seconds). See the function help for **xbee_gpio_iochange_destination()** for additional details.
XBEE_GPIO_VERBOSE	Library will print status messages to **STDOUT**.
XBEE_GPIO_DEBUG	Functions will be debuggable (for example, you can set breakpoints and single-step into them).

xbee_gpio_init

```
void xbee_gpio_init();
```

DESCRIPTION

Initialize the GPIO Endpoint handler.

RETURN VALUE

None.

SEE ALSO

xbee_gpio_tick

xbee_gpio_tick

```
int xbee_gpio_tick(api_frame_t *frame);
```

DESCRIPTION

Tick function to run the GPIO Endpoint. Must be called periodically for the endpoint to work. Returns the result of calling **xbee_tick(frame)**.

PARAMETER

frame	pointer to **api_frame_t** structure to receive a copy of the last frame processed by the **XBEE_API.LIB** library.

NOTE: The received frame may be a response to a packet sent by **XBEE_API.LIB**, or a nonresponse frame from another device on the network. Check the frame type (**frame->cmd.api_id**) and ID (**frame->cmd.u.frame_id**) to confirm that it is the expected response.

RETURN VALUE

Value of **xbee_tick(frame)**. See that function's description for a list of possible return values.

SEE ALSO

xbee_gpio_init, xbee_tick

xbee_gpio_set_digin

```
int xbee_gpio_set_digin (int signal, int value);
```

DESCRIPTION

Updates the GPIO server's reading for a digital input.

PARAMETERS

signal Signal number to update (offset into **xbee_gpio_config**).

value New value for digital input, should be one of the following:

XBEE_GPIO_INPUT_LOW
XBEE_GPIO_INPUT_HIGH
XBEE_GPIO_INPUT_OPEN
XBEE_GPIO_INPUT_CLOSED

RETURN VALUE

0 — value for digital input changed.

-EINVAL — invalid signal or value passed in.

SEE ALSO

xbee_gpio_set_digout, xbee_gpio_set_anain, xbee_gpio_set_anaout

xbee_gpio_set_digout

```
int xbee_gpio_set_digout (int signal, int value);
```

DESCRIPTION

Updates the GPIO server's reading for a digital output, then calls **xbee_gpio_hw_digout()** to update the physical output on the device.

PARAMETERS

signal signal number to update (offset into **xbee_gpio_config**).

value new value for digital output, should be one of the following:

XBEE_GPIO_OUTPUT_LOW
XBEE_GPIO_OUTPUT_HIGH
XBEE_GPIO_OUTPUT_OFF
XBEE_GPIO_OUTPUT_ON
XBEE_GPIO_OUTPUT_SINK
XBEE_GPIO_OUTPUT_SOURCE
XBEE_GPIO_OUTPUT_TRISTATE

RETURN VALUE

0 — value for digital input changed.

-EINVAL — invalid signal or value passed in.

SEE ALSO

xbee_gpio_set_digin, xbee_gpio_set_anain, xbee_gpio_set_anaout

xbee_gpio_set_anain

```
int xbee_gpio_set_anain (int signal, float value);
```

DESCRIPTION

Updates the GPIO server's reading for an analog input.

PARAMETERS

signal signal number to update (offset into **xbee_gpio_config**).

value new value for analog input.

RETURN VALUE

0 — value for digital input changed.

-EINVAL — invalid signal or value passed in.

SEE ALSO

xbee_gpio_set_digin, xbee_gpio_set_digout, xbee_gpio_set_anaout

xbee_gpio_set_anaout

```
int xbee_gpio_set_anout (int signal, float value);
```

DESCRIPTION

Updates the GPIO server's reading for an analog output, then calls **xbee_gpio_hw_digout()** to update the physical output on the device.

PARAMETERS

signal signal number to update (offset into **xbee_gpio_config**).

value new value for analog output.

RETURN VALUE

0 — value for digital input changed.

-EINVAL — invalid signal or value passed in.

SEE ALSO

xbee_gpio_set_digin, xbee_gpio_set_digout, xbee_gpio_set_anain

xbee_gpio_iochange_destination

```
void xbee_gpio_iochange_destination(const byte ieee_addr[8],
   int dest_endpoint, int analog_refresh);
```

DESCRIPTION

Updates the destination address and endpoint used for sending I/O change notifications. Also allows setting the analog refresh frequency (how often analog I/O readings are sent to the destination).

PARAMETERS

ieee_addr	64-bit (8-byte) IEEE address to send to, or **NULL** to turn off I/O change notifications.
dest_endpoint	Endpoint to send to at the destination.
analog_refresh	how often to send analog readings to the destination. Set to a negative value to turn off I/O change notifications for the given destination. Set to 0 to report digital changes only. Set to any positive value to report analog values every <analog_refresh> seconds.

RETURN VALUE

None.

SEE ALSO

xbee_gpio_tick

XBee_iDigi Library

The Dynamic C **LIB\Rabbit4000\iDigi\XBee_iDigi.LIB** library provides the common functions used by the sample programs for the iDigi BL4S100 Add-On Kit ZB.

bytes2hexstr

```
char far *bytes2hexstr (char far *dest, const byte far *src,
   int bytes, char separator);
```

DESCRIPTION

Converts a range of bytes to a string using the specified separator character.

PARAMETERS

dest	pointer to the string buffer to print to, needs to be large enough to hold 3 * <bytes> characters.
src	pointer to bytes to convert into a hex string.
bytes	number of bytes to convert.
separator	separator to use between bytes, or 0 for no separator.

RETURN VALUE

NULL if there is an error in parameters, otherwise returns <dest>.

xbee_init_or_exit

```
void xbee_init_or_exit(int verbose);
```

DESCRIPTION

Initializes the XBee ZB RF module, or exits on failure.

PARAMETER

verbose	Set to 1 to have messages printed to STDOUT while connecting to the wireless network.

RETURN VALUE

None.

findNodeByAddress

```
int findNodeByAddress(const byte ieee[8]);
```

DESCRIPTION

Searches the XBee node table for an entry with the requested 64-bit IEEE address.

PARAMETER

ieee pointer to the 64-bit (8-byte) IEEE address.

RETURN VALUE

Index into the node data table of the node with the given address or -1 if the node wasn't found.

SEE ALSO

GET_NODE_DATA

XBee_Named_Attributes Library

The Dynamic C **LIB\Rabbit4000\iDigi\XBee_NAMEd_ATTRIBUTES.LIB** library implements a simple named attribute protocol on a single endpoint/cluster. Clients can get and (for read/write attributes) set the attributes by name. Requests are sent as plaintext, using the following format.

Requests (sent to the Rabbit):

name? — Sends the value of attribute <name>.
name — Sends the value of attribute <name> (? operator is implied).
name=value — Set attribute <name> to <value>.

In requests, name can end with an asterisk (*) as a wildcard to match any remaining characters. For example, "led*?" will match "led1", "led2", "led3", and "led4" (and therefore generate four responses).

Wildcards work for setting values as well (for example, dout*=1 to set all digital outputs to 1).

NOTE: Limits of the frame buffering between the BL4S100 and its XBee ZB RF module may cause some responses to be dropped if a wildcard matches more than 8 names. If you have lots of attributes, request them in batches (for example, din*, dout*, ain*).

Responses (from the Rabbit):

name! — Error response, no attribute called <name>.
name!value — Error response, unable to set <name> to <value>.
name:value — Successful response, attribute <name> is set to <value>. (Sent in response to query (?) or assign (=) operators.)

Required Macros

You need to tell the library what endpoint, profile, and cluster to listen on for named attribute requests.

```
XBEE_ATTRIB_ENDPOINT: 1 to 219 (0xDB)
XBEE_ATTRIB_CLUSTER:0 to 65535 (0xFFFF)
XBEE_ATTRIB_PROFILE:0 to 65535 (0xFFFF)
```

When your application sets up the endpoint table using the **XBEE_API.LIB** library's **ENDPOINT_TABLE_BEGIN**, **_ENTRY**, and **_END** macros, you will need to include **ENDPOINT_XBEE_ATTRIB** in the list to register the named attribute's endpoint handlers with the **XBEE_API.LIB** library:

```
ENDPOINT_TABLE_BEGIN
   // start with any user-defined endpoints
   // (Endpt, desc, profile, device, flags, IC, OC, ICL, OCL)
   ENDPOINT_TABLE_ENTRY(2, 0, 42, 1, 0, 1, 0, &StringInCluster, NULL),
   // include the XBee named attribute endpoint
   ENDPOINT_XBEE_ATTRIB
ENDPOINT_TABLE_END
```

You need to set up a table to describe the attributes made available via this Endpoint. Use the following macros to define the **xbee_attrib** table.

```
XBEE_ATTRIBS_START
   XBEE_ATTRIBS_ENTRY(name, type, channel, getter, setter)
XBEE_ATTRIBS_END
```

PARAMETERS

name — the name of the attribute; it can be up to 19 characters long.

type — the data type, must be one of the following

TYPE_BINARY — 0 or 1
TYPE_BYTE — 0 to 255
TYPE_INT — 32768 to 32767
TYPE_WORD — 0 to 65535
TYPE_LONG — -2,147,483,648 to 2,147,483,647
TYPE_ULONG — 0 to 4,294,967,295
TYPE_FLOAT — any floating-point (real) number
TYPE_STRING — null-terminated string

channel — an integer that will be passed to your getter and setter functions (next two parameters).

getter — a function pointer to a function with the following prototype:

```
int *get_attrib(int channel, xbee_attrib_value_t *value);
```

It should read the attribute and copy it to the value passed in the **type** parameter.

setter — set to **NULL** for a read-only attribute, or a function pointer to a function with the following prototype:

```
int *set_attrib(int channel, const xbee_attrib_value_t *value);
```

Optional Macros

XBEE_ATTRIB_VERBOSE	Causes library to print status messages to **STDOUT**.
XBEE_ATTRIB_DEBUG	Causes functions to be debuggable (for example., you can set breakpoints and single-step into them).