Application Note

How to Test and Control GreenPAK using a Website

AN-CM-224

Abstract

This project shows how a GreenPAK™ design can be tested, debugged, and controlled using a website. GreenPAK ICs come with a register control feature. In this way, GreenPAK registers can be controlled on-the-fly.

This Application Note comes complete with design files which can be found in the References section.
# Contents

Abstract ........................................................................................................................................... 1

Contents ........................................................................................................................................ 2

Figures ........................................................................................................................................ 2

Tables .......................................................................................................................................... 3

1 Terms and Definitions .................................................................................................................. 4

2 References ................................................................................................................................. 4

3 Introduction ............................................................................................................................... 5

4 Server Side ................................................................................................................................. 5

  4.1 Hosting a Website and Routing Incoming Request ................................................................. 5

  4.2 Checking Destination I2C Address ....................................................................................... 6

  4.3 Writing to the GreenPAK Registers .................................................................................... 8

  4.4 Reading from GreenPAK Registers .................................................................................... 9

5 How to use the website ............................................................................................................... 11

  5.1 Inputs .................................................................................................................................... 13

  5.2 Outputs ................................................................................................................................ 14

  5.3 Chart ..................................................................................................................................... 15

  5.4 Other Actions ...................................................................................................................... 16

6 Example: Sharp Sensor Node ..................................................................................................... 16

7 Hardware Connections ............................................................................................................... 18

  7.1 Module Settings .................................................................................................................... 19

8 Settings / Connections ............................................................................................................... 20

  8.1 Arduino IDE settings ............................................................................................................. 20

  8.2 Editing Codes ....................................................................................................................... 21

9 Conclusion ................................................................................................................................. 21

Revision History ............................................................................................................................. 22

# Figures

Figure 1: Server-Side Html Codes ................................................................................................. 5

Figure 2: Routing .......................................................................................................................... 6

Figure 3: Control Code Values ..................................................................................................... 6

Figure 4: ControlGPAK Function ............................................................................................... 7

Figure 5: CheckDevice function ................................................................................................. 7

Figure 6: WriteOne function ....................................................................................................... 8

Figure 7: WriteAll function ........................................................................................................ 9

Figure 8: ReadOne function ....................................................................................................... 10

Figure 9: MultipleRead function ............................................................................................... 10

Figure 10: 12 Tooth BLDC Motor ............................................................................................. 11

Figure 11: Read from File/GreenPAK ...................................................................................... 11

Figure 12: Register Table ............................................................................................................ 12

Figure 13: Write to GreenPAK ................................................................................................... 12

Figure 14: Add module section .................................................................................................... 13

Figure 15: Add Input Modal ....................................................................................................... 14

Figure 16: Add Output Modal .................................................................................................... 15

Figure 17: Play and Operations buttons .................................................................................... 15

Figure 18: Sharp Sensor Distance-voltage Graph ...................................................................... 17

---

Application Note

Revision 1.0

01-Mar-2018

CFR0014

2 of 22

© 2022 Renesas Electronics Corporation
How to Test and Control GreenPAK using a Website

Figure 19: PGA Settings.............................................................................................................. 18
Figure 20: Hardware Connections.............................................................................................. 18
Figure 21: The F6 Register......................................................................................................... 19
Figure 22: Adding “adc_out” Input............................................................................................ 19
Figure 23: Adding “Distance “Output....................................................................................... 20
Figure 24: Created “Distance_sensor” Module ........................................................................ 20
Figure 25: Ssid, Password and Ip ............................................................................................. 21
Figure 26: SDA and SCL Settings.............................................................................................. 21

Tables

Table 1: I2C Addresses according to Control Code.................................................................. 7
1 Terms and Definitions

IOT          Internet of things
I²C          Inter-integrated circuit
ASM          Asynchronous state machine
CNT          Counter

2 References

For related documents and software, please visit:

GreenPAK™ Programmable Mixed-Signal Products | Renesas

Download our free GreenPAK Designer software [1] to open the .gp files [2] and view the proposed circuit design. Use the GreenPAK development tools [3] to freeze the design into your own customized IC in a matter of minutes. Renesas Electronics provides a complete library of application notes [4] featuring design examples as well as explanations of features and blocks within the IC.

[2] AN-CM-224 How to Test and Control GreenPAK using a Website .gp, GreenPAK Design File, Renesas Electronics
3 Introduction

This project shows how a GreenPAK design can be tested, debugged, and controlled using a website. GreenPAK ICs come with a register control feature. In this way, GreenPAK registers can be controlled on-the-fly. A GreenPAK and Lolin NodeMcu are used in this project. In addition, some web technologies have been used to create a website. However, this app note will not include client-side codes. You can look at this link if you want to review it. This app note consists of several sections. These are:

- Server Side
- Usage of Website
- Example: Sharp Sensor Node
- Settings / Connections

4 Server Side

The server-side code is written with an ESP8266 / Arduino core. In this way, you can use the ESP8266 development boards like the ESPDuino, SparkFun ESP8266 Thing, ESPPresso Lite, WeMos and even the Generic ESP8266 Board as a server. In this project, the Lolin NodeMcu will be used, so the expressions will be on the Lolin NodeMcu.

The Lolin NodeMcu was used to communicate with the GreenPAK registers and host the website. The Lolin NodeMcu and GreenPAK are communicated to each other via I²C protocol. Thanks to GreenPAK, device configurations can be changed on-the-fly. With this feature, GreenPAK registers can be controlled via the website.

Operations done by the server side are:

- Hosting a website and routing incoming request
- Checking destination I²C address
- Writing to the GreenPAK registers
- Reading from the GreenPAK registers

4.1 Hosting a Website and Routing Incoming Request

The HTML and jQuery codes, which were written for the website, take up space in the Lolin NodeMcu's memory. Therefore, the code written for the website is hosted on the remote server. Only HTML, head and body tags are generated as shown in Figure 1. The contents of these tags were pulled from the remote server using jQuery.

```html
website = "<html>
    <head>
    </head>
    <body>
    </body>
</html>"
```

Figure 1: Server-Side Html Codes

Each incoming request was directed to the relevant functions.
4.2 Checking Destination I2C Address

GreenPAKs have 16 selectable I2C addresses. These addresses can be determined using the control code register. The last 3 bits of register are fixed (as 0). The first 4 bits can be changed. The values that the control code register can receive are shown in Figure 3.

![Control Code Values](image)

Figure 3: Control Code Values

According to these values, the I2C addresses of GreenPAK are shown in Table 1.
# How to Test and Control GreenPAK using a Website

## Table 1: I2C Addresses according to Control Code

<table>
<thead>
<tr>
<th>Control Code</th>
<th>Padding</th>
<th>I2C Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>000</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>000</td>
<td>8</td>
</tr>
<tr>
<td>0010</td>
<td>000</td>
<td>16</td>
</tr>
<tr>
<td>0011</td>
<td>000</td>
<td>24</td>
</tr>
<tr>
<td>0100</td>
<td>000</td>
<td>32</td>
</tr>
<tr>
<td>0101</td>
<td>000</td>
<td>40</td>
</tr>
<tr>
<td>0110</td>
<td>000</td>
<td>48</td>
</tr>
<tr>
<td>0111</td>
<td>000</td>
<td>56</td>
</tr>
<tr>
<td>1000</td>
<td>000</td>
<td>64</td>
</tr>
<tr>
<td>1001</td>
<td>000</td>
<td>72</td>
</tr>
<tr>
<td>1010</td>
<td>000</td>
<td>80</td>
</tr>
<tr>
<td>1011</td>
<td>000</td>
<td>88</td>
</tr>
<tr>
<td>1100</td>
<td>000</td>
<td>96</td>
</tr>
<tr>
<td>1101</td>
<td>000</td>
<td>104</td>
</tr>
<tr>
<td>1110</td>
<td>000</td>
<td>112</td>
</tr>
<tr>
<td>1111</td>
<td>000</td>
<td>120</td>
</tr>
</tbody>
</table>

The `controlGPAK` function controls the I2C address that is sent. It returns 0 if there is a device capable of I2C communication at this address. If there is no device at this address or there is a problem with the communication, it returns a number other than 0. It does this using the wire library.

```c
int controlGPAKadr(int adr) {
    Wire.beginTransmission(adr);
    return Wire.endTransmission();
}
```

**Figure 4: ControlGPAK Function**

The `checkDevice` function checks all the addresses one by one as shown in Table 1. If there is a device at the target, it adds the address of the device to the response message. Finally, it notifies the suitable devices to the user.

```c
void checkDevice(void) {
    int I2C_adr[16] = {0, 8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88, 96, 104, 112, 120};
    String response = "";
    int x = 0;
    for (x = 0; x < 16; x++) {
        if (controlGPAKadr(I2C_adr[x]) == 0) {
            x++;
            response += String(I2C_adr[x]) + ":";
        }
    }
    if (x == 0)
        server.send (200, "text/html", "No devices found.");
    else
        server.send (200, "text/html", response);
}
```

**Figure 5: CheckDevice function**
NOTE: The `checkDevice` function only checks whether a suitable device is available at the destination addresses. This device may not be a GreenPAK.

### 4.3 Writing to the GreenPAK Registers

A GreenPAK can be controlled by changing its registers. The task of each register is different. For the addresses and tasks of the registers, you can refer to the Appendix in the GreenPAK’s datasheet.

The `writeOne` function writes the data that the user sends to the corresponding GreenPAK register.

```c
void writeOne()
{
    int I2C_addr;

    if (|server.hasArg("nvaData") | server.hasArg("adr") | server.hasArg("RegAdr") )
    {
        server.send(200, "text/plain", "Error Missing arguments");
        return;
    }

    I2C_addr = server.arg("adr").toInt();
    if (controlGPAdr(I2C_addr) != 0)
    {
        server.send(200, "text/plain", "Error Could not connect GreenPAK!");
        return;
    }

    device_address = I2C_addr;
    writeI2C(bytes(server.arg("RegAdr").toInt()), bytes(server.arg("nvaData").toInt()));
    server.send(200, "text/html", "Success Value successfully written to register");
}
```

**Figure 6: WriteOne function**

The `writeOne` function receives 3 data from the user. These are:

- **NvmData**: Register content,
- **Adr**: The target GreenPAK I2C address,
- **RegAdr**: Destination register address

The function first checks the content of the submitted data. If any of the `nvmData`, `Adr` or `RegAdr` data cannot be found, it returns an error message. After that, it checks whether there is a suitable GreenPAK at the I2C address sent by the user. If it cannot communicate with the GreenPAK, it returns an error message.

Finally, it writes the data sent to the register. The `writeI2C` function takes arguments in byte type. Therefore, incoming data is first converted to integer type from String type. Later, it is converted to byte type from integer type. After writing to the GreenPAK register is finished, the successful message is sent to the user.

The `writeAll` function writes all the data which is sent by user to the registers.
How to Test and Control GreenPAK using a Website

```c
void writeAll(void) {
    int I2C_addr;

    if (!server.hasArg("nvmData%5B%5D") || !server.hasArg("adr")) {
        server.send(200, "text/plain", "Error Missing arguments");
        return;
    }

    I2C_addr = server.arg("adr").toInt();
    if (ControlPAKAddr(I2C_addr) != 0) {
        server.send(200, "text/plain", "Error Could not connect to GPKA");
        return;
    }

    device_address = I2C_addr;
    for (i = 0; i < 256; i++) {
        writeI2C(byte(i), byte(server.arg(i).toInt()));
    }

    server.send(200, "text/html", "Success Values successfully written to registers ");
}
```

**Figure 7: WriteAll function**

The function receives two facets of data from the user. These are `nvmData` array and `Adr`. The function works similarly to the `writeOne` function. Unlike `writeOne`, `nvmData%5B%5D` data is checked instead of `nvmData` data. Since `nvmData` is an array in this function, it is submitted as `nvmData []`. The characters `% 5B% 5D` correspond to the characters `[ ]` in the percent-encoding. Then, the GreenPAK I2C address is checked. Finally, the `nvmData` array is written to the GreenPAK registers with the for loop. The message `successful` is sent to the user after the process is finished.

NOTE: Some registers in the GreenPAK are closed for writing. The GreenPAK target register does not generate any errors during processing, even if it is turned off to write. At this point, no errors are encountered during sequential writing.

### 4.4 Reading from GreenPAK Registers

Data can be also read from GreenPAK registers. At this point, information such as whether the pin is high or low, the counted data of the CNT blocks, and which state the ASM is currently executing can be reached.

The `readOne` function reads the contents of the register requested by user and sends it to the user.
void readOne(void) {
    int I2C_adr;
    String response = "";

    if (!server(hasArg("adr") || !server.hasArg("Regadr")) {
        server.send(200, "text/plain", "Error Missing arguments");
        return;
    }
    I2C_adr = server.arg("adr").toInt();
    if (!controlGPAKadr(I2C_adr) != 0) {
        server.send(200, "text/plain", "Error Could not connect GPAK");
        return;
    }

    device_address-I2C_adr;
    response += String(readI2C[byte][server.arg("Regadr").toInt()]);

    server.send ( 200, "text/html", response );
}

Figure 8: ReadOne function

The `readOne` function takes two inputs which are `RegAdr` and `adr`. It controls the content of the submitted data as it were in the `writeOne` function. Then it checks the GreenPAK I2C address. Finally, it adds the content it reads from the GreenPAK to the response message and sends it to the user.

The `readAll` function reads all GreenPAK registers and returns the contents to the user.

void readAll(void) {
    int I2C_adr;
    String response = "";

    if (!server.hasArg("adr")) {
        server.send(200, "text/plain", "Error Missing arguments");
        return;
    }
    I2C_adr = server.arg("adr").toInt();
    if (!controlGPAKadr(I2C_adr) != 0) {
        server.send(200, "text/plain", "Error Could not connect GPAK");
        return;
    }

    device_address-I2C_adr;
    for (i = 0; i < 255; i++) {

        response += String(readI2C[byte][i]) + ",";
    }
    response += String(readI2C[byte][255]);
    server.send ( 200, "text/html", response );
}

Figure 9: MultipleRead function
void multipleRead(void){
    int I2C_adr;
    String response = "";

    int num_reg=0;
    if (!server.hasArg("adr")||!server.hasArg("num_reg")){
        server.send(200, "text/plain", "Error Missing arguments");
        return;
    }
    I2C_adr = server.arg("adr").toInt();
    if (controlPAKAddr(I2C_adr) != 0) {
        server.send(200, "text/plain", "Error Could not connect GPAK");
        return;
    }
    num_reg = server.arg("num_reg").toInt();
    device_address = I2C_adr;
    for (i = 0; i < num_reg-1; i++) {
        response += String(readI2C(byte(server.arg(i).toInt())))+ "-";
    }
    response += String(readI2C(byte(server.arg(num_reg-1).toInt())));
    server.send (200, 'text/html', response);
}

Figure 10: 12 Tooth BLDC Motor

NOTE: Some registers may be closed for reading. You can see the I²C Serial Command Register Protection in the datasheet.

5 How to use the website

Using the web site, the user can read registers, write values to registers and visualize register values by creating modules.

The user can read the registers data from a project or from a GreenPAK connected to the Lolin NodeMcu. The user can also scan for possible GreenPAK I²C addresses.

![Read from File/GreenPAK](image)

Figure 11: Read from File/GreenPAK

After the reading process, the register is shown in Figure 12.
How to Test and Control GreenPAK using a Website

The user can write or read the register data one by one using the buttons in the table. Also, GreenPAK can be programmed using the write all button.

![Register Table](image)

**Figure 12: Register Table**

The user reads and monitors the register data via the modules. The user can create a new module or load the previously created modules. When modules are created, the user is asked to specify inputs and outputs. Each module must have a unique name.

![Write to GreenPAK](image)

**Figure 13: Write to GreenPAK**
5.1 Inputs

Inputs are determined by the register data in GreenPAK at a specific I²C address. An input can contain more than one register data. Registers data can be combined, truncated and masked.

Each input is constructed with a name, I²C and bits information. Inputs must have unique names. Patterns used when determining Bits information;

- **RegAddress**
  It takes the 8-bit data at the specified register address without any action.

  **Example**
  Register f6: 00001111
  Bits: f6
  Result: 00001111

  Note: Register address must be a hexadecimal number between 00 and FF.

- **RegAddress<x:y>**
  It gets the range of between x and y of the data at the specified register address. If x is greater than y, the data is only truncated. If x is less than y, the data is truncated first, then the bits are sorted in reverse order.

  **Example**
  Register f6: 00010111
  Bits: f6<4:1>
  Result: 1011

  or
  Bits: f6<1:4>
  Result: 1101

  Note: x and y values must be written as <x: y>. Also, x and y must be a number between 0-7.

- **RegAddress<x>**
  It takes the xth bit of the data at the specified Register address.

  **Example**
How to Test and Control GreenPAK using a Website

Register f6: 00010000
Bits: f6<4>
Result: 1
Note: x must be a number between 0 and 7.

- **[Padding]**
Unlike previous patterns, the user can add the desired padding value.

*Example*

Bits: [001001]
Result: 001001

Note: Padding values must be binary and between [ and ] characters.
The user can also combine this data. To merge, the user must put comma ("," between the patterns.

*Example*

Register f5: 00110001
Register f6: 11110000
Bits: f5<5:3>, f6
Result: 1101110000

![ADD INPUT]

The added inputs can be deleted at any time.

5.2 Outputs

Outputs are the values that will be shown in the chart. Charts are drawn using output values. Outputs are functions that contain input. More than one input can be used when output is created. Thanks to this, more than one GreenPAK register data can be shown in a single module.

Output functions are solved using the JavaScript eval function. In this way, statements such as if can be used in addition to mathematical operations in the function section. To use predefined inputs in the output function, the @ character must be placed in front of the input name and there must be a space after it. If there are predefined inputs, input buttons are added automatically under the function section. The user can add the inputs more easily by clicking on these buttons.
**Figure 16: Add Output Modal**

**Example**

Register f6: 00000111: 7
Register f5: 00001000: 8

**Inputs**

- i1: f5<3>, f6<3:0> Result: 10111: 23
- i2: f5<3:2>, [010] Result: 10010: 18

**Outputs**

- o1: @i1 * Math.sqrt( @i2 ) Result: 23 * 4,24 = 97,58
- o2: if (@i1 > @i2 )
  { @i1 ; }
  else
  { @i2 ;}
Result: 23

After you have written the output function, you can test your function by clicking the test button. It is enough to enter the inputs on the modal that opens and click on the **Calculate** button.

### 5.3 Chart

Modules convert the outputs to charts using Google chart. When each module is created, it creates a line chart showing the outputs. Modules are initially _stop_ positions. The user can operate the mode by clicking on the **play** button.

**Figure 17: Play and Operations buttons**

Register addresses and I²C addresses that are related with the modules are stored in an object. When the module is run, the required I²C addresses and register addresses are taken from this object and sent to Lolin NodeMcu. Thus, only the register information necessary for the operation of the modules is obtained. This object is changed when a module is stopped. If there is no other module that needs the related register, the register status will be _stop_.

Charts are refreshed by default every 10 seconds. GreenPAK register values are read every 10 seconds.
5.4 Other Actions

The user can add the previously saved modules by clicking on the operations button found in the add module. Alternatively, the user can save all the modules to a file. Modules are saved in a txt file in JSON format. Also, after the user creates the module, the Operations button appears in the module section.

The actions the user can make by clicking on this button;

■ Chart editor
The user can change the properties of the chart that show the outputs. The user can also change properties such as the chart type and the colors of the outputs. Once the user clicks on the OK button, the changes are applied and the chart is re-drawn. Chart editor is created using google chart editor.

■ Save Module
The user can only save this module by clicking on this button. Inputs, outputs, chart options of the module are saved.

Note: The output data of the module are not saved in save operations. So, when the module is restored, it is created from scratch.

■ Delete Module
The user can delete this module by clicking this button. The module is removed from the page and deleted from the required objects.

■ Download data
The user can download output data in csv format by clicking this button. Columns are separated by semicolon (";") in the downloaded file. The user can use this file to examine the data.

6 Example: Sharp Sensor Node

In this example, distance measurements with the Sharp GP2Y0A41SK0F Analog Distance Sensor (4-30 cm) were checked. These measurements were converted to graphs using the module.

The Sharp sensor is an analog sensor. This example can be used when creating modules with other analog sensors. Since there is no Analog Digital Converter (ADC) in GreenPAK (SLG46531V), the ADC in GreenPAK4 (SLG46620V) was used. The AN-1102 app note was used for this. The distance-voltage output graph of the Sharp sensor is shown in Figure 18.
The Sharp sensor output can be up to 3.1 V depending upon distance. However, the $V_{\text{ref}}$ of the ADC in the SLG46620V is 1.2 V. Therefore, the Sharp sensor output was divided into 4 to bring it to a level lower than 1.2 V.

Figure 18: Sharp Sensor Distance-voltage Graph
How to Test and Control GreenPAK using a Website

Figure 19: PGA Settings

7 Hardware Connections

Connections were made as shown in AN-1102. Since the Lolin NodeMcu communicates in 3.3 V levels, the GreenPAK and GreenPAK4 were supplied with 3.3 V. The Sharp sensor operates between 4.5 V and 5.5 V, hence the supply is derived separately. Connections can be seen in Figure 20.
7.1 Module Settings

As shown in Figure 20, the 8-bit incoming data from the SLG46620V is connected to the pins of the SLG46531V. These pins are 19-18-17-16-15-14-13-12 respectively. The digital input values of these pins are stored in the F6 register, as shown in the Matrix Input table on the SLG46531V datasheet.

<table>
<thead>
<tr>
<th>reg=1988&gt; Matrix Input 48</th>
<th>Pin12 Digital Input</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>reg=1989&gt; Matrix Input 49</td>
<td>Pin13 Digital Input</td>
<td>Valid</td>
<td>Invalid</td>
</tr>
<tr>
<td>reg=1970&gt; Matrix Input 50</td>
<td>Pin14 Digital Input</td>
<td>Valid</td>
<td>Invalid</td>
</tr>
<tr>
<td>reg=1971&gt; Matrix Input 51</td>
<td>Pin15 Digital Input</td>
<td>Valid</td>
<td>Invalid</td>
</tr>
<tr>
<td>reg=1972&gt; Matrix Input 52</td>
<td>Pin16 Digital Input</td>
<td>Valid</td>
<td>Invalid</td>
</tr>
<tr>
<td>reg=1973&gt; Matrix Input 53</td>
<td>Pin17 Digital Input</td>
<td>Valid</td>
<td>Invalid</td>
</tr>
<tr>
<td>reg=1974&gt; Matrix Input 54</td>
<td>Pin18 Digital Input</td>
<td>Valid</td>
<td>Invalid</td>
</tr>
<tr>
<td>reg=1975&gt; Matrix Input 55</td>
<td>Pin19 Digital Input</td>
<td>Valid</td>
<td>Invalid</td>
</tr>
</tbody>
</table>

**Figure 21: The F6 Register**

So, the F6 register is used when adding an input. Let’s give the name *adc_out* to the input we added.

![Figure 22: Adding “adc_out” Input](image)

The Sharp sensor is not a linear sensor. The distance formula for this sensor;

\[
 v = Sensor \ output \ voltage \\
 Distance = 12.38 \times v^{-1.1}
\]

To find the value \( v \), the \( V_{ref} \) value must be divided into resolution and this value must be multiplied by *adc_out*.

Since we set the gain of the sensor output voltage to 0.25, the value we find will be 1 in 4 of the actual value. So, we need to multiply the found value by 4.

As a result, the value of \( v \):

\[
 v = \frac{V_{ref}(1.2) - Resolution(256) \times 4 \times adc\_out}{0.01875 \times adc\_out}
\]

Distance:

\[
 12.38 \times Math.pow( v, -1.1)
\]

In case of a problem with the sensor, the *adc_out* variable can be zero. In this case, the function will go to infinity. We can use an if statement to prevent it. As a result, the final state of the output function;

```javascript
var v = 0.01875 * @adc_out;
if ( v != 0 ){
    12.38 * Math.pow( v, -1.1);
} else {
    false;
}
```
If the variable \( v \) is zero, the function will return false and the module will be automatically paused.

![Add Output Form](image)

**Figure 23: Adding “Distance “Output**

After adding the output to the module, save the module by clicking the save module button. Now the module is ready to operate. You can start the module by clicking on the play button. If you want, you can change the chart type by clicking Operations button, you can save the module, you can download measured distances in csv format.

![Chart](image)

**Figure 24: Created “Distance_sensor” Module**

8 Settings / Connections

8.1 Arduino IDE settings

The Lolin NodeMcu card is programmed using the Arduino IDE. Processes to introduce the Arduino IDE to the Lolin NodeMcu board are:

1. Open Arduino IDE
2. Click File -> preferences
How to Test and Control GreenPAK using a Website

3. Click the button next to Additional Boards Manager URLs
5. into the opened window and click ok
6. Click tools -> board -> boards manager
7. Find the ESP8266 in the opened window and load it

Insert the development board into the computer after you identify Lolin NodeMcu and select NodeMcu 1.0 (ESP-12E Module) from the tools -> board. Also, do not forget to select the port that the board is connected to from Tools -> Port.

8.2 Editing Codes

The user must make a few adjustments before loading the codes into the Lolin NodeMcu.

First, you should set the “ssid” and “password” variables in the server-side code according to the Wi-Fi network to use. If you use the Generic ESP8266 board, you should set GPIO 0 and GPIO 2 as SDA and SCL for I2C communication. You can use the default pins for other boards.

It can also keep client-side codes on a remote server specified by the user, if desired. Client-side codes are located in the GitHub server. It can be used without making any changes. Finally, the necessary settings must be made to get a static IP to the Lolin NodeMcu. You can access the website using this IP address.

```c
const char *ssid = 'WIFI_NAME';
const char *password = "WIFI_PASSWORD";
 IPAddress ip(192, 168, 43, 200);
```

Figure 25: Ssid, Password and Ip

```c
// Change this to the Wire.begin(0,2) for Generic esp8266
Wire.begin();
```

Figure 26: SDA and SCL Settings

9 Conclusion

With the techniques outlined in this app note, you can control GreenPAKs remotely. This makes it easier to use a GreenPAK as a sensor node in applications like IOT and home automation.
## Revision History

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>01-Mar-2018</td>
<td>Initial version</td>
</tr>
</tbody>
</table>
IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES ("RENESAS") PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD-PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers who are designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only to develop an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third-party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising from your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

© 2024 Renesas Electronics Corporation. All rights reserved.