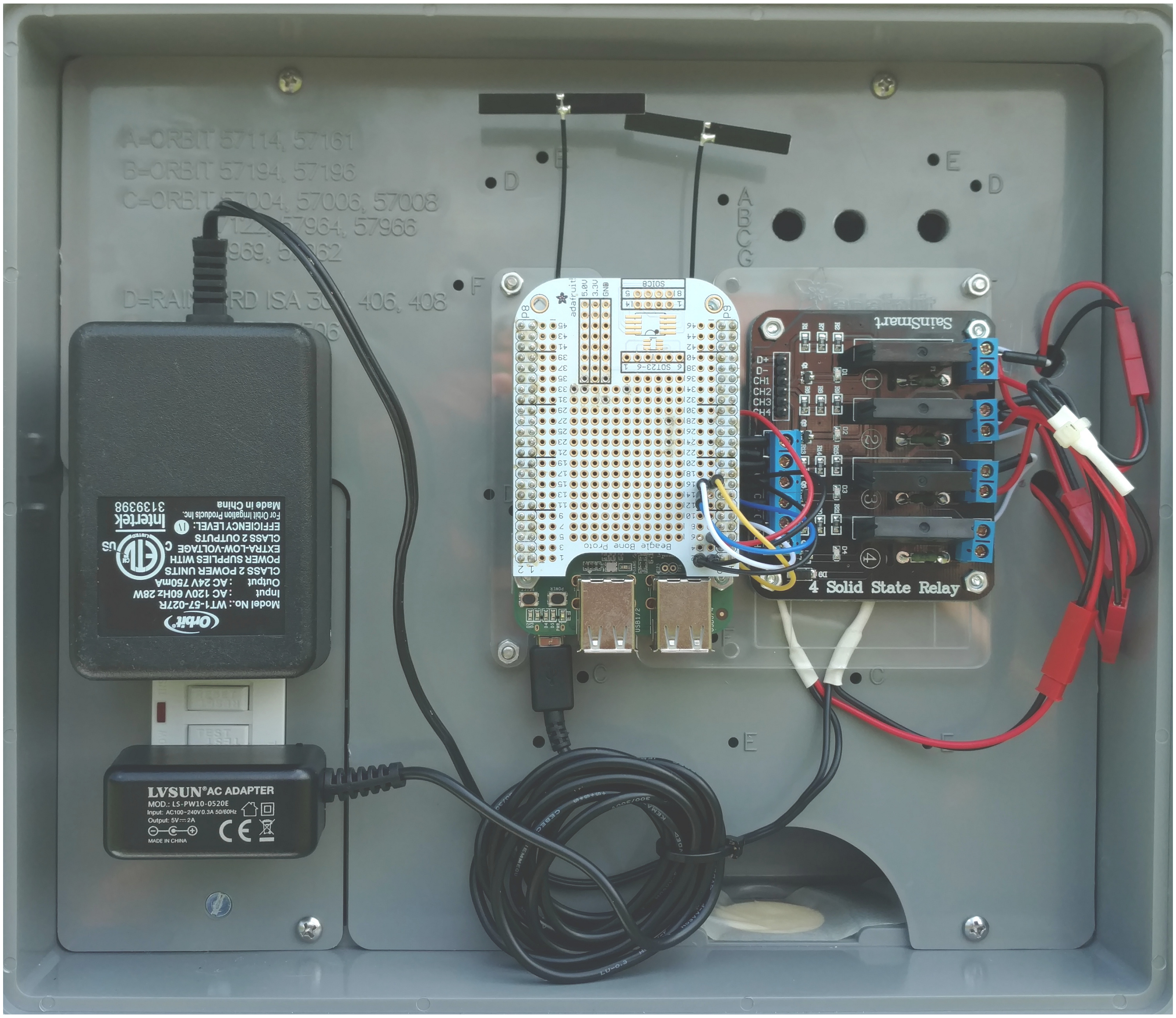

Irrigation Controller Using Beaglebone Green Wireless, Node.js, and Ecmascript 6

Gregory Raven

May 28, 2017

Irrigation Controller Using Beaglebone Green Wireless, Node.js, and Ecmascript 6

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This is the documentation for an embedded GNU/Linux project using the Beaglebone Green Wireless (BBGW) development board. The project repository is located here:

https://github.com/Greg-R/irrigate-control

The Debian-based GNU/Linux distribution used on the BBGW can be downloaded from this page:

http://beagleboard.org/latest-images

The “IOT” (non-GUI) image was chosen, as this provides the shortest path to get the project up and running.

A listing of additional resources is found in the Resources chapter.

1.1 Project Goals

I’ve published two other projects on Hackster.io. These other projects are interesting, and good learning tools, but they do not do anything useful in the real world. After a couple of years of "going up the learning curve" on embedded device development, it was time to create something practical!

This project performs the simple home automation task of controlling a lawn irrigation system. For many years, I had been controlling the irrigation system manually by flipping a circuit breaker to turn the pump motor on and off. The old timing unit had failed years ago, and I had never replaced it. So now I wanted to be able to control the system via a web browser without having to leave my home office desk.

Although in principle the automation is simple, the underlying technology is complex! There was a significant investment in time learning the technology required to implement the project.

1.2 Technologies

The development board chosen is the Beagle Bone Green Wireless (BBGW). I have significant experience with the regular Beagle Bone Green, and the WIFI capability is required for this project. The board will be mounted remotely with access to power only (no wired ethernet is possible).

The Debian-based GNU/Linux distribution used on the BBGW can be downloaded from this page:

http://beagleboard.org/latest-images

The “IOT” (non-GUI) image was chosen, as this provides the shortest path to get the project up and running.

Node.js version v7.9.0 was used. This is a much later version than what is included in the image. A section on ungrading is included.

The project’s Javascript code uses several "Ecmascript 6" constructs. In my opinion this release of Javascript is a significant improvement in the language. Some of the strange quirks of Javascript are eliminated!

Two-way communication between the web browser and the BBGW was done with “WebSockets“. The client side WebSocket is built into the browser. Using the latest updates in Ubuntu 16.04, both Firefox and Chromium browsers include this capability. The Chrome browser of an Android phone was also found to work.

Here is a good reference on the client (browser) side WebSockets:

https://developer.mozilla.org/en-US/docs/Web/API/WebSocket

The server side uses a Node.js package called “ws”:

https://github.com/websockets/ws

Since this is a “real world” project, there has to be an interface with real actuators. This small “solid-state-relay” board proved to be ideal for taking the GPIO outputs of the BBGW and doing something real:

https://www.amazon.com/gp/product/B00ZZVQR5Q/ref=oh_aui_detailpage_o00_s00?ie=UTF8&psc=1

This board has four individual relays. This project uses only three.

Note that this type of relay can switch AC power only. This was ideal for this project, as the irrigation system is powered by 24VAC.

The other common components of the irrigation system are listed in the reference section at the end of this document.

“Universal IO” was used to set the pin multiplexer to GPIO mode:

https://github.com/cdsteinkuehler/beaglebone-universal-io

Chapter 2

System Diagrams

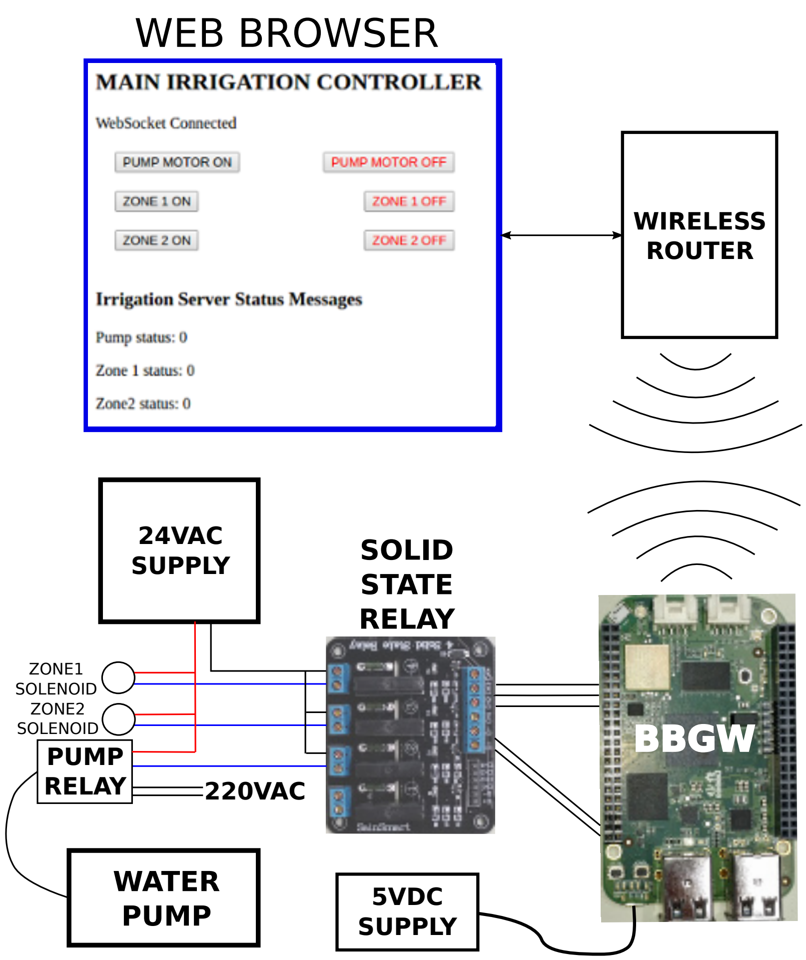

Figure 2.1: Beagle Bone Green Wireless Irrigation Control System

The above diagram shows the main components of the system. A reference section is included which has a complete parts list.

2.1 GNU/Linux Operating System on Host ARM Processor

The command uname -a on the BBGW used to develop this project reports this:

```
Linux beaglebone 4.4.48-ti-r88 #1 SMP Sun Feb 12 01:06:00 UTC 2017 armv7l
GNU/Linux
```

Chapter 3

Ecmascript 6

This page shows which Ecmascript 6 features are implemented in major versions of Node:

`http://node.green/`

Javascript has become a feature rich and large language! Some of the ES6 constructs used in this project:

- class
- Map
- arrow function
- Proxy
- const and let

A good reference for ES6 is at the Mozilla Developer Network:

`https://developer.mozilla.org/en-US/docs/Web/JavaScript`

A quick summary of the ES6 constructs follows.

3.0.1 class

The new "class" keyword does not provide new functionality. What is does is allow a Javascript class object to be written in a more traditional object-oriented style. This does make things easier for a person used to other language's syntax for defining classes.

3.0.2 Map

The "Map" is a new data structure object which is similar to what is called a "hash", dictionary, or associative array. Prior to ES6, the Javascript object was used functionally as a Map, however, this was kind of a hack.

In this project, the Map object is used in the pumpActuator object.

3.0.3 Arrow Function

"Arrow Functions" are a simplification of the syntax used when defining a function. However, they are also important in determining the the scope of "this" within the function. The arrow functions capture the "this" of the enclosing scope. This was found to be convenient in the design of the pumpActuator class.

```
this.pumpMap = new Map([['pumpmotor', 0],
                        ['zone1', 0],
                        ['zone2', 0]]);
```

The "pumpMap" is a simple data structure to store the state of the pumpmotor and the two zone actuators.

3.0.4 Proxy

Of the above ES6 constructs, the only one requiring detailed explanation is the "Proxy". The construct is used to implement the so-call "Observer" pattern.

The instantiation of the "pumpHashProxy" is done in the constructor of the pumpActuator class:

```
this.pumpHashProxy = new Proxy(this.pumpMap, this.pumpObserver());
```

The Proxy's constructor takes two arguments, which in this case is this.pumpMap and this.pumpObserver(). The second argument requires some explanation.

```
pumpObserver() {
    return {
        set: (target, property, value, receiver) => {
            console.log(`Setting ${property} to ${value}.`);
            this.pumpControl(property, value);
            target[property] = value;
            return true;
        }
    };
}
```

The above class method is a little bizarre. This method merely returns a Javascript object. The object in this case has a single key "set" and the value is an arrow function with four parameters: target, property, value, and receiver.

The Proxy creates a sort of "watcher" or "observer" of pumpMap. The proxy intercepts changes written to or read from the target object (the first parameter of the Proxy constructor).

Using the key "set" causes the function (the set key's value) to be executed when the value is written to. Note that the function has access to the intercepted property and value, and these are used in call the function “this.pumpControl”. This does the physical setting of the GPIOs. The data structure is also updated (target[property]=value) and “true” is returned to indicate a successful set.

The proxy does a sort of “intercept” of writes to the object and then can perform custom actions based on the the function assigned to “set”. Similar functionality for reads can be done with the “get” key. The object can contain both and custom read and write functions can be used. This is very powerful!

Functionally what the Proxy does is intercept the write to the data structure which stores the state of the pumpmotor and the zone solenoids. The intercept runs the “set” function and changes the physical state of the GPIOs.

The write to the data structure is done by the server in file websocketserver.js:

```
Object.assign(pumpObject.pumpHashProxy, controlObject);
```

The “controlObject” in this case is an incoming WebSocket message (using JSON notation) from the browser which looks like this example:

```
{"pumpmotor":0}
```

The write to the data structure in the pumpActuator object is done by the server in file websocketserver.js:

```
Object.assign(pumpObject.pumpMapProxy, controlObject);
```

The “Object.assign” is a shortcut which simply overwrites the value in pumpMapProxy with the value from controlObject. The Proxy intercepts this write, and then executes the custom set function.

The Proxy allows a custom behavior to be executed when a data structure is written to or read from. This is very powerful! In this particular project with only three controls it does not standout, however, this sort of “Observer” pattern is very scalable and could be very advantageous in a much larger and more complex system.

3.0.5 const and let

const creates a read-only reference to a block-scoped value. let is also block-scoped, but it is a variable.

let and const solve the crazy problem of hard to understand scope and “hoisting” of the var type variables of previous versions of Javascript. var was not used anywhere in this project.

const and let are a huge improvement to Javascript!

Chapter 4

GPIO Control with sysfs Virtual File System

The “PRU Firmware” are two binary files which are placed in the directory /lib/firmware. These files must have specific names as follows:

- am335x-pru0-fw
- am335x-pru1-fw

The Makefile includes cp commands to copy the firmwares to the /lib/firmware directory.

4.1 PID Firmware in PRU0: Digital Feedback Loop (PRU_PID_0.c)

This C program defines a struct “shared_mem” which contains another struct “pid_data”. This same struct is also defined in the PRU1 code. It is this common data structure which allows the two PRUs to exchange data.

The following code fragment shows how the PRU shared memory is arranged:

```
#pragma DATA_SECTION(share_buff, ".share_buff")
volatile far struct shared_mem share_buff;
```

In addition to the code in the C files, the “DATA_SECTION” must be defined in the linker command file AM335x_PRU.cmd:

```
  PAGE 2:
PRU_SHAREDMEM : org = 0x00010000 len = 0x00002FA8 CREGISTER=28 /* 12kB Shared RAM */
        GLB_BUF         : org = 0x00012FA8 len = 0x00000058 /* Shared buf in Shared RAM */
```

This must also appear in “SECTIONS”:

```
SECTIONS {
        (...other memory allocations)
        .share_buff > GLB_BUF, PAGE 2
}
```

The implementation of the PID controller is done using the usual infinite while loop.

A function "init_pid" sets the shared_mem struct to some initial values. Another while loop looks for the init_flag variable to go high, which is a signal from PRU1 that the system is initialized and ready to start.

The main PID control loop is based on the function "update_pid". The function reads current values from the shared_mem struct and calculates an error value. Using the PID controller design pattern, errors for proportional, integral, and derivatives terms are defined in terms of C assignment statements. The terms are summed to create a total error variable "output_f".

The code uses a "trick" to emulate floating point mathematics using only fixed integers:

```
output = output_f >> SHIFT;
```

where the "SHIFT" was defined as:

```
#define SHIFT    0x0E
```

The above trick is also applied in the integral statement.

A couple of if statements bound the output within limits set in the shared_mem struct.

The loop control statement implements the negative feedback control:

```
pid->output = pid->max_output - output;
```

4.2 The Firmware in PRU1: PID Control (PRU_IO_1.c)

The firmware in PRU1 is less concerned about math, and more concerned with communicating with the world outside the PRU-ICSS. The C code sets up the RemoteProc messaging framework to allow communications with Linux user-space. PRU1 is also responsible for writing to the PWM and reading data from the Quadrature Decoder.

The same shared_mem struct as seen in PRU0 code is defined. PRU1 needs to both read and write from this data structure. PRU0 processes the data and returns an output value to write to the PWM which is determined by the PID calculations.

After initialization, the code enters an infinite while loop. The while loop services three tasks:

1. A RemoteProc Messaging interrupt bit is polled, and if it has been set this means that a message has been sent from Linux user-space. The message is received, and then an "interrupt service routine" function is executed. The ISR consists mainly of a case statement with several character strings used as codes to either set or read variables in the shared_mem struct. This is the mechanism whereby the user-space program can control the motor RPM (setpoint) and parameters of the PID control loop.

2. Write the current output value to the PWM:

   ```
   CT_ECAP.CAP2_bit.CAP2 = share_buff.pid.output;
   ```

This statement is totally cryptic, but it does indeed write a variable to the PWM function of PRU1 and sets the waveform duty-cycle which in applied to the input of the DRV8833 motor control IC.

3. Read the Quadrature Decoder output. This is done using the utility function “get_enc_rpm()”. Since this is the controlled parameter of the feedback control loop, the value is written to the shared_mem struct for processing by the PID calculations in PRU0.

The above is only a high-level description, as the code’s features are too numerous to describe every function. The curious reader is invited to examine the code which is published to the Git repository for further details.

4.3 Quadrature Decoder Tuning

The “Quadrature Decoder” function is contained within the PWMSS module. This is a complex system with many tunable parameters.

The original TI project recommends using an LED to signal “overflow/underflow” indication from the decoder. This proved to be important. The published values for the decoder parameters do not work with the recommended eBay motor-encoder.

The LED indicator is connected to header pin P8-39. The LED is connected in series with a 1.2kΩ resistor. The ground end of the resistor is connected to P8-45.

Universal IO was used to connect the PRU to header pin P8-39 as follows:

```
config-pin P8.39 pruout
```

The above command is included in the shell script “pru_gpio_config”.

Under/over-flow is indicated by a blinking LED which is implemented in the function get_enc_rpm().

The following original and modified values are from the function init_eqep().

4.3.1 Original Quadrature Decoder/Encoder Parameters

This is the original setting for use with the TI recommended motor-encoder; this did not work well with the eBay motor-encoder. The LED was blinking quite a lot indicating under/overflow in the decoder circuit.

```
PWMSS1.EQEP_QCAPCTL = 0x0073;
```

Another parameter which must be adjusted is the "ticks per revolution". Due to using a different motor-encoder, and the fact that the motor is geared, this parameter must be changed if the RPM calculation is to be done correctly. Here is the original parameter:

```
/* Encoder definitions */
#define TICKS_PER_REV       16
```

4.3.2 Modified Quadrature Decoder/Encoder Parameters

This value was empirically adjusted until the LED stopped blinking. The rotation of the motor was “noisy” prior to this being adjusted. With this new value, the motor control changed to smooth and steady.

```
PWMSS1.EQEP_QCAPCTL = 0x0070;
```

The parameter for "ticks per revolution" with the eBay motor-encoder is an early estimate:

```
/* Encoder definitions */
#define TICKS_PER_REV       40
```

The above is an early estimate, and it should be re-examined and revised if necessary. Since this parameter effects the loop dynamics, and the PID parameters had been adjusted for a stable loop, this parameter was left as-is for future optimization.

Chapter 5

Solid State AC Relays

Figure 5.1: DRV8833 Break-out board (2 boards showing with view of top and bottom sides)

The recommended motor driver IC is the Texas Instruments DRV8833:

http://www.ti.com/lit/ds/symlink/drv8833.pdf

This devices works perfectly with this project and is inexpensive. Several eBay sellers offer a "break-out board" with the IC and several external components mounted with break-board friendly header pin holes. The board shown in the photo above even includes a surface mounted LED power indicator!

The connections to the board are as follows:

1. ULT PIN:mode set. Low level is sleep mode
2. OUT1,OUT2:1-channel H-bridge controlled by IN1/IN2
3. OUT3,OUT4:2-channel H-bridge controlled by IN3/IN4
4. EEP PIN:Output protection. Default no need to connect.
5. VCC:3-10V
6. GND

From the above list, only 2, 5 and 6 are used in this project.

IN1 is connected to the PWM output of the BBG, which is header P9.42. The GND pin requires a connection to one of the grounds on the BBG such as P8.1 or P8.2.

VCC should be connected to an 8Volt DC power supply, however, the exact voltage is not critical. A solid ground connection should be made between the 8Volt supply and the DRV8833 board.

OUT1 and OUT2 should be connected to the motor power terminals.

Chapter 6

Configuration of the Beagle Bone Green Wireless

The default configuration of the Beagle Bone Green Wireless is as a “access point” (a wireless router). This is not a desired configuration for a dedicated embedded device as used in this project.

The following process re-configures the BBGW to a non-access point mode.

The goal is to have a working wireless substitute for the ethernet connector which does not exist on the BeagleBone Green Wireless. This is as a typical "headless" embedded project with the primary access using a terminal and ssh.

Download and expand the IOT bone image per this link and flash to micro-sd. I used this one:

https://debian.beagleboard.org/images/bone-debian-8.7-iot-armhf-2017-03-19-4gb.img.xz

Write the image to the micro-sd (I put the micro-sd in a USB adapter plugged into my Ubuntu workstation):

```
sudo dd if=bone-debian-8.7-iot-armhf-2017-01-15-4gb.img of=/def/sdb bs=8M
```

Eject the micro-sd from workstation and insert in the BBGW micro-sd slot. Connect a USB 3.3V serial device to the "debug serial header". The USB network connection could be substituted, however, my experience with this is that using the serial device is solid and will work consistently. Also, since the BBGW doesn’t have a dedicated power connector. It uses the micro-USB. It is my preference to use a high-current dedicated USB power supply and ignore this as a possible network connection.

Power-up the BBGW and wait for the boot process to complete. Open a (bash) terminal and use the screen utility to connect via the serial USB device.

```
screen /dev/ttyUSB0 115200
```

I had to hit enter after the above command to get to the login prompt. The login user is debian and the password is temppwd.

You may have to install screen:

```
sudo apt-get install screen
```

After logging in, a good thing to do first is to run this shell script:

```
cd /opt/scripts/tools
sudo ./grow_partition.sh
```

Next:

```
ifconfig
```

You should see 4 different network resources (not showing the full output here):

```
SoftAp0
lo
usb0
wlan0
```

The network resource SoftAp0 represents an "access point". The BBGW is configured as a wireless router! That is not the desired configuration, and fortunately this is easily removed. Edit the file:

```
/etc/default/bb-wl18xx
```

Change the line:

```
TETHER_ENABLED=yes
```

to

```
TETHER_ENABLED=no
```

Save and exit, and reboot, and login.

ifconfig should now show only 3: lo, usb0, and wlan0.

Now to configure WIFI! It is assumed you have a home wireless router and you know the SSID and passphrase. The router should be configured for DHCP (automatic assignment of IP addresses). From a terminal:

```
sudo connmanctl
connmanctl> scan wifi
Scan completed for wifi
connmanctl> services
    (your router broadcast)         (router info)
connmanctl> agent on
Agent registered
connmanctl> connect (copy router info here)
Agent RequestInput (router info)
  Passphrase = [ Type=psk, Requirement=mandatory, Alternates=[ WPS ] ]
  WPS = [ Type=wpspin, Requirement=alternate ]
Passphrase? (your passphrase)
```

```
Connected (router info)
connmanctl> quit
```

The above configuration is permanent and will survive reboot. An outstanding page with good info on connman:

https://wiki.archlinux.org/index.php/Connman

Another very good thing is to login to your router and use this to determine if your BBGW is successfully connected. And remember the router may have security settings which may block it from connecting. Also, rather than attempting to force a fixed IP address on the BBGW, I used the "address reservation" feature so that the IP address assigned by the router will be the same each time it connects. This is done using the MAC address of the BBGW.

After the above configuration is done, shutdown and remove the USB serial device. Power up the BBGW and wait for it to boot, and then using a terminal and ssh you should be able to connect to the BBGW as if an ethernet cable was connected:

ssh debian@(the assigned IP address)

After logging in you should have internet connectivity, so don't forget to:

sudo apt-get update

Here is an example USB serial device. This should be on your tool kit list:

https://www.amazon.com/gp/product/B01AFQ00G2/ref=oh_aui_search_detailpage?ie=UTF8&psc=1

Chapter 7

Device Tree Requirements

This project requires only three GPIOs. Rather than editing device tree files, and having to deal with potential bugs caused by this, the great config-pin utility was used. This utility is provided by the Universal IO project.

The Universal IO project is located at this Github repository:

`https://github.com/cdsteinkuehler/beaglebone-universal-io`

Universal IO is included with the most recent Debian-based IOT images.

Chapter 8

Running the Project

It is assumed the numerous steps described in prior chapters have been completed to enable the RemoteProc framework drivers and configure the BBG for compiling PRU C code. The Device Tree must have been successfully edited and re-compiled and installed, and the shell scripts prumodin and prumodout have been copied to /usr/bin.

In order to run the project and successfully control the motor, follow these steps:

- run “make” in the software repository directory. Some warnings or errors may be ignored. Check that the C code files prumsg.c, PRU_PID_0.c and PRU_IO_1.c compile and firmware files am335x-pru0-fw and am335x-pru1-fw are copied to /lib/firmware. The user-space binary prumsg should be copied to /usr/bin.
- Run command

  ```
  prumodin
  ```

  at the command line. This command should start firmware execution in the PRUs. If all goes well, the motor should begin turning with a default setpoint value of 3000.
- Finally, the user-space program can be used.

  ```
  sudo ./prumsg s 4000
  ```

  The above command changes the setpoint to 4000 rpm. The motor speed should increase.

The PRU firmwares will continue to run. To stop them, issue the commands

```
sudo prumodout
sudo rmmod pru_rproc
sudo rmmod pruss
```

at the command line, and the PRUs will be halted. The motor will stop.

8.1 User-space Program prumsg Command Listing

The user-space executable file prumsg is capable of several control and monitoring functions. These commands are issued from a shell and a complete listing of the possible commands is listed below.

Table 8.1: Commands of User-space Program prumsg

Example command	Command function
prumsg 30 s 5000	Set setpoint (RPM)
prumsg 30 p 300	Set Kp, proportional feedback coefficient
prumsg 30 i 300	Set Ki, integral feedback coefficient
prumsg 30 d 300	Set Kd, derivative feedback coefficient
prumsg 30 o 3000	Set output PWM duty cycle (see note below)
prumsg 30 rs	Readback setpoint (RPM)
prumsg 30 rp	Readback Kp
prumsg 30 ri	Readback Ki
prumsg 30 rd	Readback Kd
prumsg 30 re	Readback encoder RPM
prumsg 30 ro	Readback output PWM

Notes on the above:

1. If not operating as root, “sudo” will be required.
2. “30” in the table above refers to the character device /dev/rpmsg_pru30. “31” can also be used, as this character device is also established between PRU1 and user-space.
3. The example for setting the PWM (prumsg 30 o 3000) will not have effect with the PID controller running. However, this command is useful for debugging purposes. If the PID controller is not running in PRU0, then the command will work and the PWM output will change. The simplest way to do this is to remove firmware am335x-pru0-fw from directory /lib/firmware. Reboot and restart the system. PRU1 will still function in system control mode, but the PID calculations will not be performend by PRU0 and the system will operate in “open-loop” mode. This is excellent for checking the PWM output to the motor driver IC and the motor connections. The motor should properly respond to changes in the PWM duty cycle by issuing the prumsg 30 s (pwm value) command.

8.2 Server and GUI Interface from TI Project

The TI project includes a very clever PHP web page and server interface. This was found to be mostly functional, and the server shell script and web page implementation is included in the pru_pid_server directory of the Git repository.

The server is easy to run. Simply copy the contents of the pru_pid_server directory to /var/www/html which should already be included in the Beaglebone Green IOT image.

Now, using a browser on the local network, browse to this URL:

```
10.0.0.2:8080
```

In the above example, the BBG is set to a static IP of 10.0.0.2.

This was found to partially function, the graphics successfully updates, however, the capability to update the setpoint and PID parameters did not function.

This project does not support this function. Since the Beagleboard project is heavily invested in node.js and "Bonescript", it would probably make sense to change this function from PHP/html to a node-based web interface and use a web-socket for data exchange and parameter control. This feature may be added in the future to this project.

Chapter 9

Resources

9.1 Github repository for this project

https://github.com/Greg-R/

9.2 Beagle Bone Green Wireless

https://www.seeedstudio.com/

9.3 The PRU GPIO Spreadsheet

Use "git clone" to download this repository:

https://github.com/selsinork/beaglebone-black-pinmux

The spreadsheet file contained in this repository is pinmux.ods. The LibreOffice suite has a spreadsheet application which will read this file.

This spreadsheet is extremely useful when configuring the PRU or other functions to the Beaglebone pin multiplexer.