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

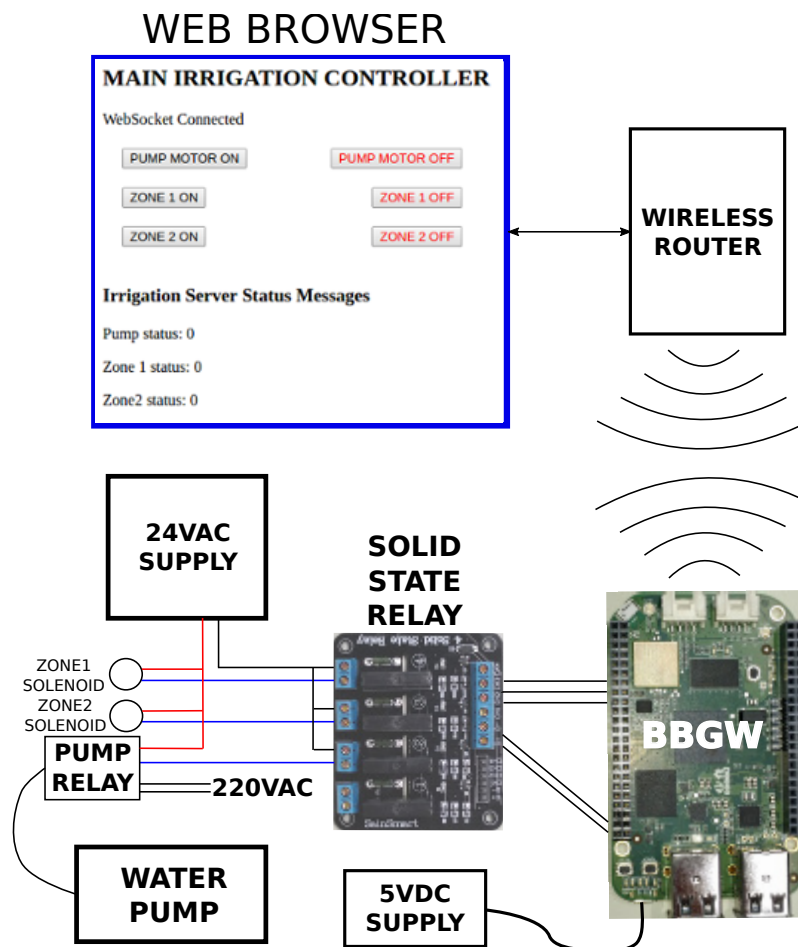**Figure 2.1: Beagle Bone Green Wireless Irrigation Control System**

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
- 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 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.

Here is the code:

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 the Beagle Bone Green Wireless

Figure 6.1: Data Pathways on the AM335X SOC

TI has provided example code and kernel drivers for the "RemoteProc and RemoteProc Messaging Framework". A detailed explanation of this framework is available here:

http://processors.wiki.ti.com/index.php/PRU-ICSS_Remoteproc_and_RPMsg

This framework provides a means of controlling and communicating with the PRUs from userspace, and this project is totally dependent on these functions.

The Remoteproc framework automatically does the job of loading the PRU firmwares from userspace into the PRUs. Via a sysfs entry, the PRUs can be started and halted from the command line. These functions are described in the chapter "Shell Scripts".

The examples provided in the PRU Software Support Package show how to use provided functions to send and receive data from PRU to ARM or ARM to PRU. This is done via character devices which appear in the usual /dev directory. The standard POSIX functions read/write/open/close work with these character devices. This allows for typical systems programming technique to be applied when working with the PRUs.

This project uses two character devices assigned to PRU1. These character devices are used to assign and read back PID parameters to the control system via the user-space executable prumsg.

This project did not require modifications to the loadable kernel modules in the RemoteProc framework. The modules provided with the IOT Debian-based distribution were used as-is.

6.1   The Remoteproc and RPMsg Kernel Modules

"Loadable Kernel Modules" (LKMs) must be active for this project to function. At the shell command line, execute this command:

```
lsmod
```

This will list the LKMs currently loaded. The modules associated with Remoteproc are:

- pruss

- pru_rproc

- pruss_intc

There are two modules associated with RPMsg, and these will appear in the list only after the firmwares are loaded into the PRUs:

```
virtio_rpmsg_bus
rpmsg_pru
```

The Remoteproc kernel modules may not be loaded at boot (depending on boot configuration). However, they can be loaded (or unloaded) anytime after boot with the following commands:

```
modprobe pru_rproc
rmmod pru_rproc
```

"modprobe" is the load command; "rmmod" is the remove command.

Perhaps a better method of controlling the Remoteproc framework is via the sys virtual file system. A set of shell scripts is included in the repository which includes these commands in the "prumodout" and "prumodin" scripts.

6.2   Files Associated with RemoteProc in the Compilation Process

There are some interesting files in the root directory of the github repository for this project:

- resource_table_0.h

- resource_table_1.h

- AM335x_PRU.cmd

These files were copied verbatim from the PRU Support Package.

Jason Reeder of Texas Instruments has provided an explanation of the files required to compile firmwares for the PRUs:

> There are four files needed in order to build a C project for the PRU using TI's C compiler. Each of the examples and labs in the pru-software-support-package include these files:
>   1. yourProgramFile.c
>   This is your C program that you are writing for the PRU.
>   2. AM335x_PRU.cmd

This is a command linker file. This is the way that we describe the physical memory map, the constant table entries, and the placement of our code and data sections (into the physical memory described at the top of the file) to the PRU linker. There are some neat things that can be done as far as placing code and data in exact memory locations using this file, but for the majority of projects, this file can remain unchanged.

3. resource_table_*.h
This file will create a header in the elf binary (generated .out file) that is needed by the RemoteProc Linux driver while loading the code into the PRUs. For the examples in the pru-software-support-package, there are two types of resources that the PRU can request using the resource_table_*.h file:
interrupts - letting RemoteProc configure the PRU INTC interrupts saves code space on the PRUs.
vrings - requesting vrings in the resource_table file is necessary if rpmsg communication is desired (since the ARM/Linux needs to create the vrings in DDR and then notify the PRU where the vrings were placed) even if no resources are needed, the RemoteProc Linux driver expects the header to exist. Because of this, many examples in the package contain an empty resource_table header file (resource_table_empty.h)

4. Makefile
Makefile to build your PRU C program either on the target, on your Linux machine, or even on a Windows machine. The comment at the top of the Makefile tries to explain the environment variable needed for a successful build and how to set it on each of the three supported build development environments. (Note that this is in reference to the Makefiles provided with the PRU Support Package. The custom Makefile provided with this project does not have this feature.)

Chapter 7

Device Tree Requirements

This project requires a custom "Device Tree Include". This is a device tree fragment which is inserted into the top device tree file. The dtsi directory located in the software directory contains the file and and a README file which explains how to edit the device tree source file.

The same file, which in this case is "am335x-bonegreen.dts", must be modified in order to active the RemoteProc framework kernel drivers. It is recommended to add the include statement at the same time the RemoteProc is activated. This step is included in the RemoteProc and PRU Compiler step-by-step process in Chapter 9.

There is one PRU GPIO output enabled on header P9 and this is used to monitor Quadrature Decoder under/overflow. This configuration is done in the same file "pru_gpio_config" which is sourced by .bashrc as discussed in Chapter 6.

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 performed 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/pru-pid-motor

9.2 The Texas Instruments PRU PID Motor Demonstration Project

http://processors.wiki.ti.com/index.php/PRU_Training:_PRU_PID_Motor_Demo

There is also a PDF file which describes the project in detail:

http://www.ti.com/lit/ug/tidubj6/tidubj6.pdf

9.3 Texas Instruments PRU Code Generation tools

http://software-dl.ti.com/codegen/non-esd/downloads/download.htm#PRU

9.4 Beagle Bone Green

https://www.seeedstudio.com/SeeedStudio-BeagleBone-Green-p-2504.html

9.5 The Remoteproc Framework and Remote Messaging

http://processors.wiki.ti.com/index.php/PRU-ICSS_Remoteproc_and_RPMsg

9.6   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.

9.7   The BeagleScope project

"Beaglescope" includes some very important information on the RemoteProc framework:

https://github.com/ZeekHuge/BeagleScope