Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

LoRa Power Management

Robert Cooney
B.A.(Mod.) Computer Science
Final Year Project May 2018
Supervisor: Stephen Farrell

School of Computer Science and Statistics

O'Reilly Institute, Trinity College, Dublin 2, Ireland

DECLARATION

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university

_____ _____

Name Date

Acknowledgements

Firstly I would like to thank my supervisor Stephen Farrell for his guidance, advice, and expertise over the course of this project. I would like to thank Kerry Hartnett for his assistance during the project. I would also like to thank my friends and family for their continued support throughout the year.

Abstract

This report details the update made to an existing solar powered LoRa gateway with the aim of improving system up time, via enhanced power management, together with redesigning the code base to bring it up to modern standards. The aforementioned update involved replacing legacy hardware from the previous system, and rewriting the existing code base to improve efficiency and maintainability, in addition to allowing customisation of operation parameters. It works as a standalone system, in addition to providing a stepping stone for new ventures into the intersection between the areas of solar power and the Internet of Things.

Solar power is becoming increasingly prevalent as governments and private entities look to move away from the use of fossil fuels towards renewable energy. As this transition is made, solar panel owners will look to optimize the amount of energy generated, in order to achieve maximal efficiency.

The Internet of Things (IoT) is also an area of rapid growth, with the number of IoT-connected devices growing exponentially year-on-year. Accordingly, the number of gateways used to connect devices with the cloud is also growing as increased network capacity is required.

Contents

Chapter 1

Introduction

This report presents the final year project titled LoRa Power Management, in the area of Networks and Telecommunications. The project aims to update an existing solar powered LoRa gateway, with the goal of improving system up time using more efficient power management, and also to bring the system code base up to modern standards.

1.1 Project Overview

1.1.1 Solar Energy

At of 2011, world energy consumption was 10 terawatts per year, by 2050 that figure is forecast to triple to 30 terawatts per year[Raz11]. With such a large growth in energy consumption, along with the rapid exhaustion of fossil fuels, renewable sources of energy are being looked to as the solution to these impending energy deficits. As one of the most developed renewable power sources in terms of technology, solar is a viable answer for many bodies looking to move away from non-renewable sources. The system described in this project uses solar energy as its power source. It aims to take into consideration the varying nature of solar irradiance when in operation.

1.1.2 LoRa Gateway

LoRa gateways are used by Internet of Things devices to connect with the cloud, in order to relay data and receive downlink communications. Operators of gateways aim for maximal system up time, in order to provide the most reliable service possible. As the system described is used to power a LoRa gateway, it follows that an aim of this project is to maximize gateway operation time, to provide ample coverage to IoT devices within its range.

Figure 1.1: Previous gateway implementation

The previous implementation of the solar powered gateway, in operation January-September 2017, was handed down before beginning the project (see fig. 1.1)

1.1.3 On Board Computer

Some hardware from the old implementation had become faulty, and would need to be replaced. The system requires a low power board, as one with a high current draw would greatly affect the up time of both the power management program and the gateway by using unnecessary energy. In-built RTC and power management capabilities are also requirements of the board, in order for the system to power on and off at designated times and battery thresholds. The system connects to phidgets, which are data I/O boards that relay sensor information from the solar panels back to the computer, in addition to providing a visual output in the form of a screen that displays the current device state (see figure 1.2).

Figure 1.2: Phidget screen displaying voltage, current draw, date and time

1.2 Reader's Guide

- Chapter 2 will cover the project background and related work.
- Chapter 3 will discuss the body of work including project methodology and set up.
- Chapter 4 will present the conclusions and suggest potential future work in this area.
- Chapter 5 will provide the reference list.
- Chapter 6 will contain documentation for the code.

Chapter 2

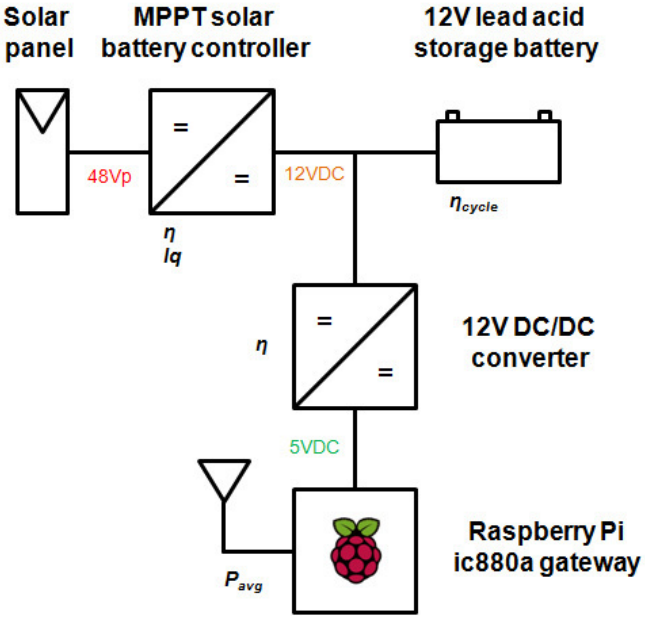
Background

2.1 Motivation

The Internet of Things (IoT) is a rapidly growing area. According to Forbes, the global market for IoT is project to grow from "$2.99T in 2014 to $8.9T in 2020, attaining a 19.92% Compound Annual Growth Rate"[For17]. With its expansion comes the demand for additional capacity, to allow new devices to connect with the cloud. Considering the fledgling nature of the market and the relatively low barrier to entry when compared to other telecommunications markets such as cellular, there is much work that can be done in improving the state of IoT. From security management, to coverage optimisation and maximising up time, IoT solutions still have plenty of distance left to cover.

Solar power is also an area of rapid expansion. Between 2000 and 2017, solar power has seen a growth factor of 57 [Nan17]. Combining the areas of solar and IoT enables work in an intersection between two quickly developing industries, and allows for the creation of new and exciting technologies. For instance, as IoT gateway operators look to spread their coverage across hard to reach areas such as barren deserts, the difficulty of supplying power via mains to these gateways grows significantly. However by making use of solar energy as a power source, these hubs could be set up with remarkably less effort than setting up power lines across such an area. These solutions would look to make use of power management theories such as those discussed in this report, to maximize the efficiency of each hub.

Figure 2.1: Solar Powered Gateway, Epyon, *TheThingsNetwork 2017*



2.2 State-of-the-art

The intersection between solar power and IoT gateways is a mostly unexplored area. There is a briefly documented solution available on the web, posted in February of 2017 [DSG17], the design for which can be seen in figure 2.1. The aforementioned system was relatively successful, able to stay up for "6 days" during cloudy weather. However, apart from this project there is a distinct lack of work in the solar/IoT cross section. There are products for sale in the IoT device area, such as the Cypress S6SAE101A00SA1002 Solar-Powered IoT Device Kit [CYP18], but these devices are small sensors with low power requirements. This is in contrast with IoT gateways, which are far more energy intensive, and as such require a considerably larger solar array to provide power.

Chapter 3

Project

3.1 Methodology

The approach to creating the system was as follows:

- Inspect the previous implementation, and retain the adequately working parts (both hardware and software)
- Replace defunct hardware pieces with up-to-date alternatives
- Run old software implementation on new hardware as proof-of-concept
- Design new software system
- Implement, test and tweak new system

3.2 Setup

Before beginning work on implementing the software system, it was necessary to bring the hardware up-to-date. Some parts of the handed-down equipment had become defunct and required replacement. The Eurotech Proteus board, used to control operation of the gateway, was no longer functioning. The solar charge controller, which regulates the solar energy received from the panels was also non-functional.

A Raspberry Pi 3 was acquired to run the power management application [rPi18], along with a solar charge controller to power the Pi via energy harvested by the panels [SCC18]. As the Pi does not have built-in RTC and power management capabilities, a WittyPi[WPI17] was also obtained, which adds these functionalities to the Raspberry Pi. The WittyPi is a small extension board that attaches to the Raspberry Pi, as can be seen in figure 3.1.

Once the necessary equipment was purchased, system assembly began. After being attached to the WittyPi, the Raspberry Pi was connected to the solar charge controller via USB to micro USB cable. The Pi connects to the phidget interface via USB, to display current status on screen and track battery voltage. The solar charge controller was wired into the batteries and a generator as the power source, as solar panels are not usable from inside the development office. The assembly outline can be seen in figure 3.2.

Post assembly, software setup was done as follows:

- Install Pi from NOOBS
- Follow instructions to use phidgets with the Pi[PHI14]
- Clone and make the pbm mercurial repo[PBM17]
- Install WittyPi on Raspberry Pi[WPI18]

Figure 3.1: Raspberry Pi with WittyPi

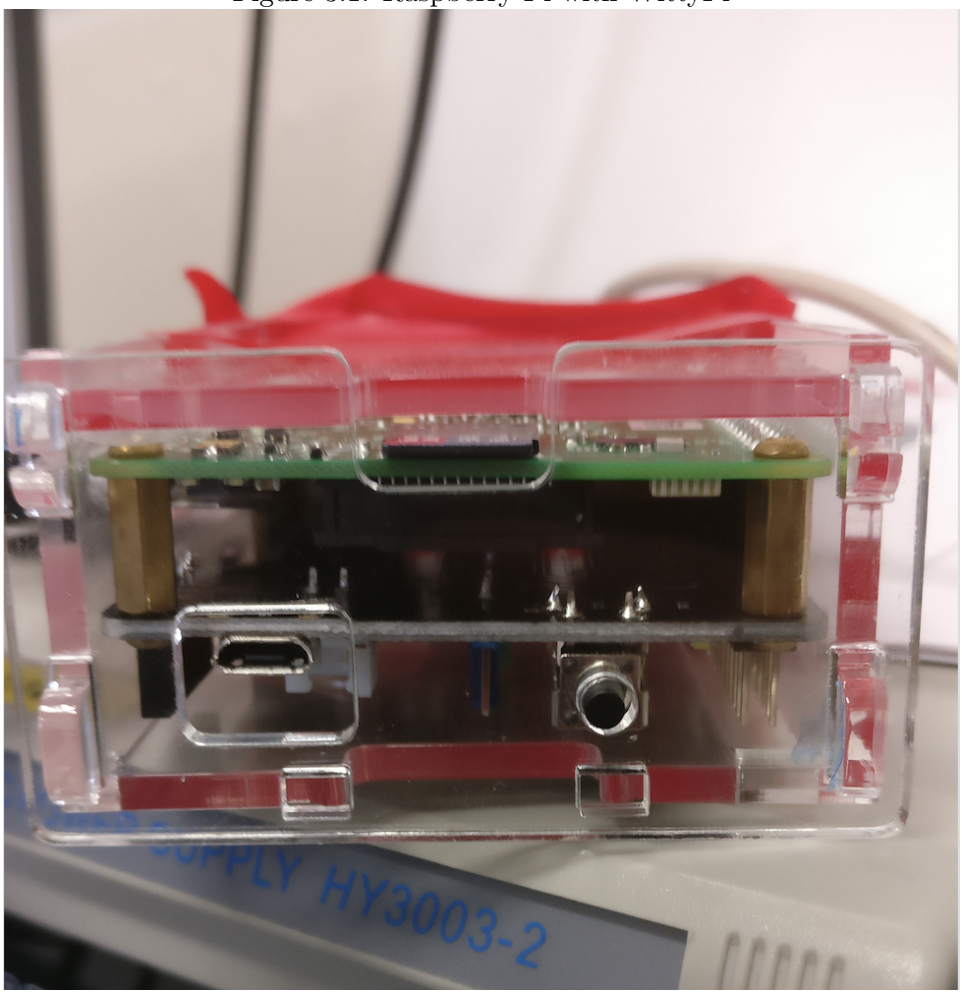
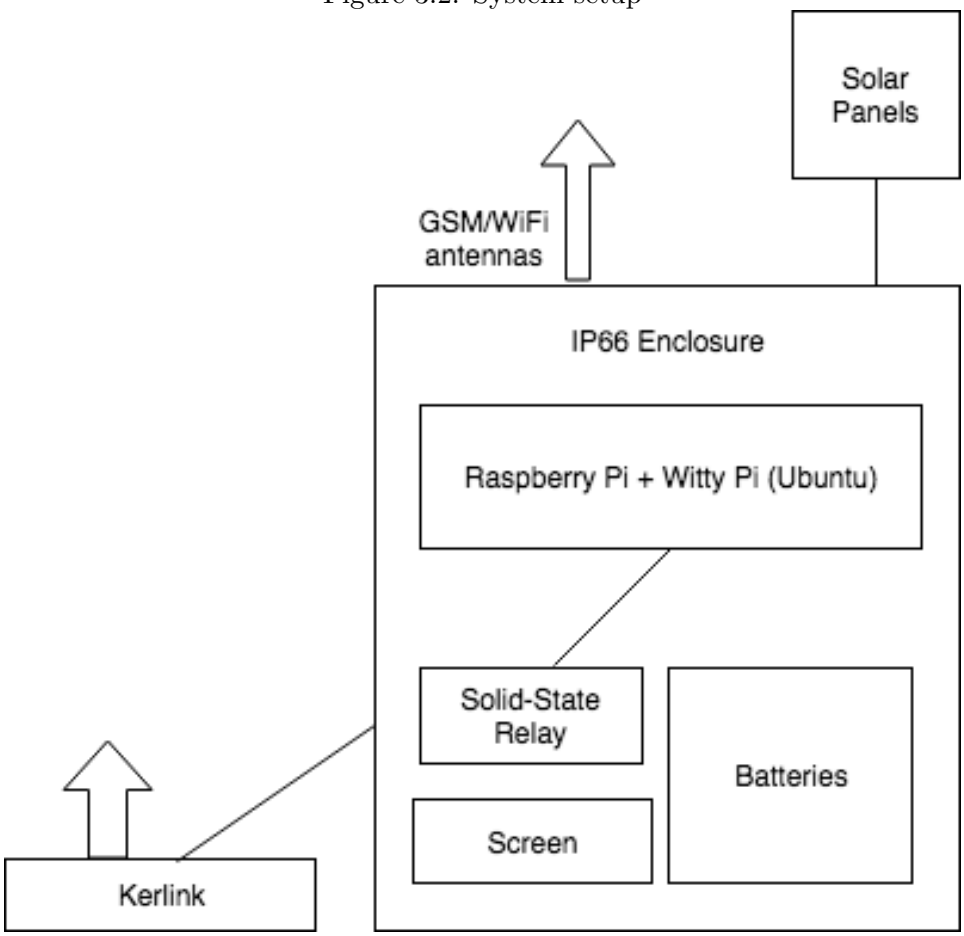


Figure 3.2: System setup



3.3 Design

The previous implementation of the system came from a 2017 project to build a solar powered LoRa gateway[PBM17]. The general process of the previous model was kept in mind when designing the new system, as it was assessed to be well-designed but poorly implemented.

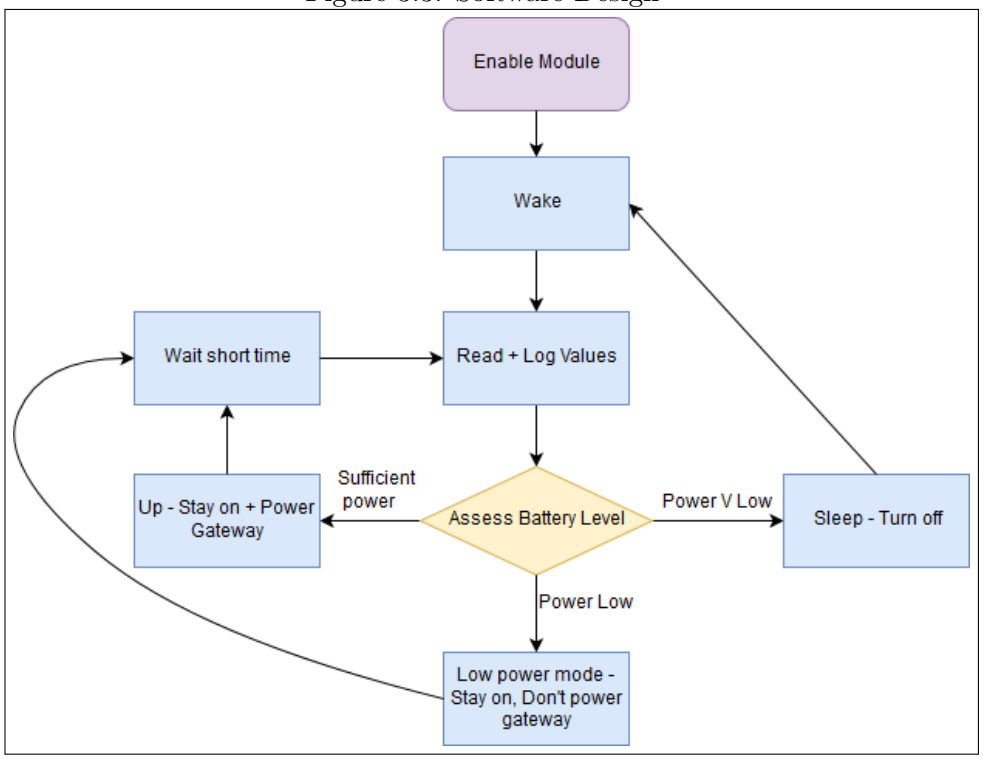
The control flow was the first piece of design work created. It is relatively simple, with the main decision being which state to transition to after assessing the battery level (eg. Up/Low Power/Sleep). When awake, the voltage and current values are consistently read and subsequently logged. In

Up and *Low power* states, the system will stay on and wait a short time before re-reading the aforementioned values and re-assessing the battery level. When transitioning to a *Sleep* state, the system will shut down for a period of time. How long the system stays powered down is dependent on sleep parameter passed to the program which can be manually configured. The program flow is described in figure 3.3.

The program compiles from a main C file, two libraries and a utilities bash script, in contrast with the previous implementation which used eleven files in total. It was ascertained that this was the optimal method to design the program as the main function of the main file would comprise the majority of the power management capability of the system, and as such there was no need to split the program into more files.

There is also a configuration option available to users, where they can choose the "mode" in which the system operates, eg. *Greedy* mode will turn off for shorter periods of time, resulting in more log results, but the more frequent startup and shutdown cycle uses more power than *Conservative* or *Moderate* mode. Relative sleep patterns were also added later in the design process, whereby the user can set the system to stay on and turn off for constant periods of time, eg. stay on for 50 minutes, then turn off for 10 minutes, stay on for 50 minutes, off for 10, etc.

Figure 3.3: Software Design



3.4 Implementation

Two files were used from the previous implementation of the project[PBM17] - handlers and phidgets. These file libraries are available from the phidgets website [PHA18]. The rest of the fragmented code base was rewritten, with some usable functions salvaged. In particular,

```
updateLogfile
```

and

```
updateSnapshotFile
```

were re-used, as they create log files. Maintaining consistent logging formats simplifies comparison between the old and new power management programs. A full list of re-used functions can be found at the end of the chapter.

There are three C files used in the program. These are:

- pbmd.c - Contains main loop, in addition to power management algorithm
- phidgets.c - Functionality for interacting with phidgets, including reading values such as voltage and current, and updating screen to display system status
- handlers.c - Handles attaching phidgets to the system and returning error responses given during operation

Utilities.sh contains a number of commands for the WittyPi, used to shut down and start up the system. This file is provided by WittyPi, and is available after installing the software for the WittyPi[WSC18]. The Makefile can be used to compile the program after the necessary software has been installed, by running

```
make
```

from inside the *stable* directory of the code base. The newpbm initscript is added to

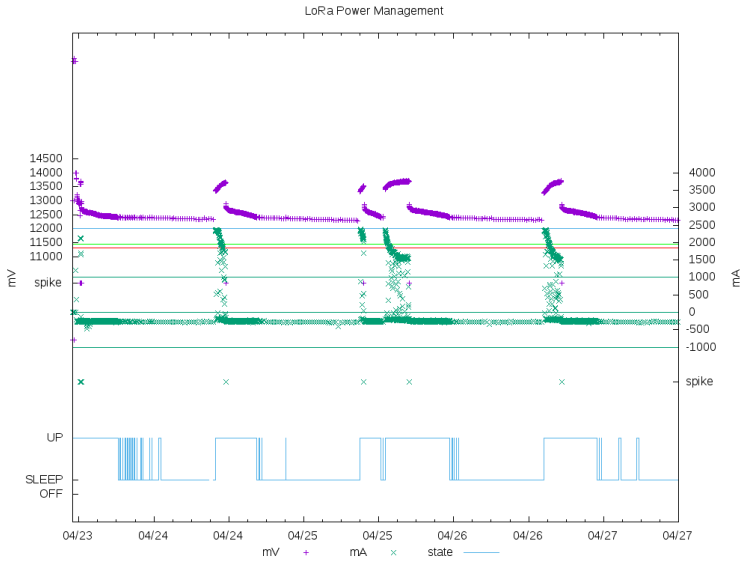
```
/etc/init.d
```

to enable the program to run when the Raspberry Pi starts up. When running, the program will consistently check and log the power values (every 30 seconds). By default, the operation mode is *Moderate*, which means that when power is too low, the system will turn off for 20 minutes before turning on again. By editing the initscript, the user can change the operation mode to *Greedy* or *Conservative*, which will cause the system to turn off for 10 and 30 minutes respectively, when power is sufficiently low. The system also provides a relative sleep mode, which can be activated by adding two numbers to the program parameters in the initscript - sleep duration and uptime duration. For instance

```
DAEMON_ARGS="GREEDY SIM 20 20"
```

will result in the system staying up for 20 minutes, then turning off for 20 minutes, and so on. Full documentation for the code can be found in the Appendix, Chapter 6 of the report. There is also a Github Repository containing the code available online[GIT18].

Figure 3.4: Multi-Day Results



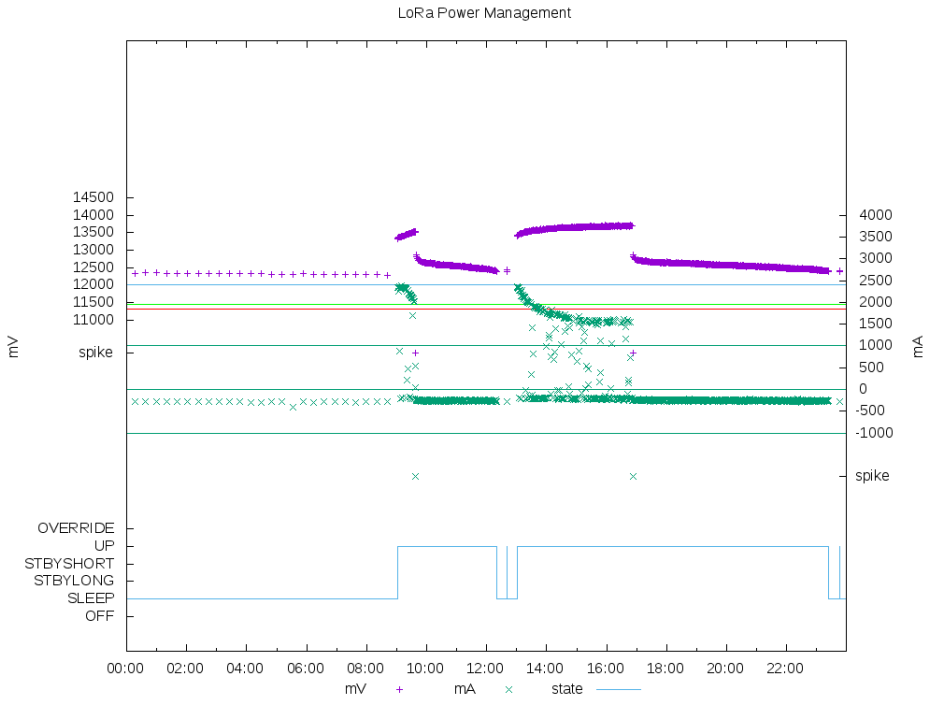
3.5 Testing

Testing was done in-house, using a combination of simulated voltages, battery logs from the previous implementation and manual control of power supply. During early tests, it was found that values logged by the system were overestimating the battery voltage. Subsequently, the method for calculating the voltage values was recalibrated.

Late-stage test results can be seen in figure 3.4 and 3.5, when the system was tested over a number of days. Figure 3.4 shows results over each of the days, while 3.5 shows a single day. The tests were successful, with the system staying on (UP) when it was expected to, and turning off (SLEEP) at the threshold voltage.

Once the voltage drops below 12400 mV, the system turns off for 20 minutes before coming back up to check if there is sufficient power to stay up, as can be seen clearly in figure 3.5. Gaps between the log values from 00:00 until 09:00 indicate this. These logs provide proof that the system aims to maximize uptime, insofar as it is feasible, by sleeping for relatively short periods of time and only when the battery voltage is low enough to prevent the system maintaining constant uptime.

Figure 3.5: One-Day Breakdown



3.6 Updates

Tweaks to the code included:

- Recalibrating voltage and current conversions from readings.
- Updating threshold voltages to reduce likelihood of brownout.
- Adding relative sleep mode as configuration option.
- Creating scripts to demonstrate system functionality.
- Changing point at which program records values to improve consistency of output logs.
- Increasing frequency at which values are logged.

3.7 Salvaged Functions

These functions were salvaged from the previous implementation of this project.

```
updateLogfile
updateSnapshotfile
getTimeString
clearSnapFile
getProgName
kerOn
kerOff
```

Chapter 4

Conclusion

The goal of improving system up time has been achieved, by means of using threshold voltages to decide when the system should shut down and start up. This is in contrast to the handed-down implementation, which required the user to define specific times of day to start up and shut down (11am to 4pm specifically). The system can stay up so long as it has sufficient power, as can be seen in section 3.5.

The code base update has also been successful, considering the following metrics:

- Condensing the number of files used in compilation from eleven to four while keeping file lengths at a reasonable length.
- Documentation for code in the main file to improve readability and maintainability.
- Overall more concise and maintainable code

Some issues were encountered initially when interfacing with the system hardware, due to the author's lack of knowledge in the area of phidgets. This was added to by the daunting nature of the functions provided by the phidget library, with long names such as *CPhidgetInterfaceKitHandle*. These difficulties were overcome with assistance from the project supervisor, in addition to documentation available online[DOC18], which helped develop a strong understanding of the aforementioned area.

Another issue encountered was that the system could not be connected to the solar panels from the development office, due to the distance. This particularly affected the system testing, but was overcome by using a mix of different sources to test functionality, including passing previous voltage logs

as parameters in the test case, in addition to using a manually controlled power supply.

4.1 Future Work

With some changes and updates, the current version of the system could look to be ready for commercial distribution. With the addition of an API, corporate users could look to set up multiple copies of the system in various locations to both provide network coverage and collect solar power statistics. As the current project uses Witty Pi scripts, it is not portable in its entirety to other hardware setups, however with some small tweaking to the shut-down and start-up calls it can be adapted to fit many Ubuntu systems.

In another direction, the computer system could be used by private bodies, such as home-owners. The use of solar panels to generate power in the home is becoming ever more prevalent, and this project could be used as a guide to setting up a solar power harvester at home. Some changes may be necessary, such as the removal of the IoT gateway if undesired, and wiring the system into the home in such a way as to provide power, but these are not unreasonably large.

More testing and updating could be done to allow further user configuration of system set up based on preferences and environment. The project described in this report could be used as a proof of concept for future collaborations between the two areas. It is clear with the rapid growth of both, their paths may well cross to a greater degree in the time to come.

Chapter 5

References

[Raz11]T.M.Razykov, C.S.Ferekides, D.Morel, E.Stefanakos, H.S.Ullal, H.M.Upadhyayae "Solar photovoltaic electricity: Current status and future prospects" in *Solar Energy* Volume 85, Issue 8, August 2011. DOI: https://doi.org/10.1016/j.solener.2010.12.002

[For17]Forbes, 30/04/2018, retrieved from https://www.forbes.com/sites/louiscolumbus/2017/12/10/2017-roundup-of-internet-of-things-forecasts/#40bca6561480

[rPi18]Raspberry Pi, 30/04/2018, retrieved from https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[SCC18]ALLPOWERS Charge Controller, 30/04/2018, retrieved from http://iallpowers.com/index.php?c=product&id=371

[PHI14]Getting Started with Phidgets on the Raspberry Pi, 30/04/2018, retrieved from http://www.instructables.com/id/Getting-Started-with-Phidgets-on-the-Raspberry-Pi/

[WPI18]WittyPi User Manual, 30/04/2018, retrieved from http://www.uugear.com/doc/WittyPi2_UserManual.pdf

[Nan17]Nancy M. Haegel "Terawatt-scale photovoltaics: Trajectories and challenges" in *Science* 356, pp. 141-143. DOI: https://doi.org/10.1126/science.aal1288

[DSG17]Designing a Solar Powered Gateway, 30/04/2018, retrieved from https://www.thethingsnetwork.org/forum/t/designing-a-solar-powered-gateway/5599

[CYP18]Cypress S6SAE101A00SA1002 Solar-Powered IoT Device Kit , 01/05/2018, retrieved from http://www.cypress.com/documentation/development-kitsboards/s6sae101a00sa1002-solar-powered-iot-device-kit

[WSC18]WittyPi software install scripts, 01/05/2018, retrieved from http://www.uugear.com/repo/WittyPi2/installWittyPi.sh

[WPI17]WittyPi website, 02/05/2018, retrieved from http://www.uugear.com/witty-pi-realtime-clock-power-management-for-raspberry-pi/

[GIT18]Github repository for LoRa Power Management, 01/05/2018, retrieved from https://github.com/sftcd/loradtn-pi

[PBM17] Solar Powered LoRa gateway, Stephen Farrell, Kerry Hartnett, Eoin Meehan, 01/05/2018, retrieved from https://basil.dsg.cs.tcd.ie/code/n4c/pbm/

[DOC18] Phidget documentation, 01/05/2018, retrieved from https://www.phidgets.com/docs/Main_Page

Chapter 6

Appendix - Documentation

LoRa Power Management

1.0

Generated by Doxygen 1.8.14

Contents

1 Documentation for LoRa gateway power management program

Main file
Phidget handlers
Phidget functions

2 File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

3 File Documentation

3.1 C:/Users/Robert/iCloudDrive/Documents/loradtn-pi/stable/handlers.c File Reference

Phidget handlers.

```
#include "handlers.h"
```

Functions

- int **IFK_DetachHandler** (CPhidgetHandle IFK, void ∗userptr)
- int **IFK_ErrorHandler** (CPhidgetHandle IFK, void ∗userptr, int ErrorCode, const char ∗unknown)
- int **IFK_OutputChangeHandler** (CPhidgetInterfaceKitHandle IFK, void ∗userptr, int Index, int Value)
- int **IFK_InputChangeHandler** (CPhidgetInterfaceKitHandle IFK, void ∗userptr, int Index, int Value)
- int **IFK_SensorChangeHandler** (CPhidgetInterfaceKitHandle IFK, void ∗userptr, int Index, int Value)
- int **IFK_AttachHandler** (CPhidgetHandle IFK, void ∗userptr)
- int **LCD_AttachHandler** (CPhidgetHandle IFK, void ∗userptr)
- int **LCD_DetachHandler** (CPhidgetHandle IFK, void ∗userptr)
- int **LCD_ErrorHandler** (CPhidgetHandle IFK, void ∗userptr, int ErrorCode, const char ∗unknown)
- void **setHandlers** (CPhidgetInterfaceKitHandle IFK, CPhidgetTextLCDHandle LCD)

3.1.1 Detailed Description

Phidget handlers.

Date

30 April 2018

3.2 C:/Users/Robert/iCloudDrive/Documents/loradtn-pi/stable/pbmd.c File Reference

State machine + logging for power management daemon.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <phidget21.h>
#include <syslog.h>
#include "phidgets.h"
```

Macros

- #define **_GNU_SOURCE**
- #define **BATTERY_LOG** "/var/log/battery-new.log"
- #define **SNAP_LOG** "/var/log/battery-snapshot-new.log"
- #define **BOOTFLAGS** "/etc/bootflag-new"
- #define **SLEEP_STATE** 0
- #define **UP_STATE** 1
- #define **LOW_POWER** 2
- #define **OVERRIDE** 3
- #define **GREEDY_STR** "GREEDY"
- #define **MODERATE_STR** "MODERATE"
- #define **SIMULATE_STR** "SIM"
- #define **NOSIM_STR** "NOSIM"
- #define **DEPLETE** "DEP"
- #define **GREEDY_SLEEP** 10
- #define **MODERATE_SLEEP** 20
- #define **CONSERVATIVE_SLEEP** 60
- #define **SECONDS_TO_MINUTES** 60
- #define **SLEEP_DURATION** 30
- #define **SIMULATE** 0
- #define **NO_SIMULATE** 1
- #define **min**(a, b) (a < b ? a : b)

Functions

- int updateLogfile (FILE *logfile, int voltage, int amps, int dutAmps, int state, int spiking, const char *prog↩ Name)

  Updates snapshot file showing snapshot of recent activity.
- int updateSnapshotfile (const char *snapFileName, int voltage, int amps, int state, int spiking, char *mode)

  Updates snapshot file showing snapshot of recent activity.
- void getTimeString (int wakeTimeSec, char *wakeTimeStr)

  Gets time as a string.
- int clearSnapFile (char *snapFileName)

  Clears snapshot file.
- char * getProgName (char *path)

  Gets program name.
- char * getStateDesc (int state)

  Gets description of current state as string.
- int simulateVoltage (struct tm *time, int currentV)

  Simulates voltage for testing purposes.
- int simulateAmps ()

  Simulates amps for testing purposes.
- int simulateDUTAmps ()

  Simulates DUT amps for testing purposes.
- char * concat (const char *s1, const char *s2)

  Concatenates strings.
- char * createStartupString (int timeToSleep)

  Creates startup string to be given to wittyPi as parameter.
- char * createShutdownString (int timeToWait)

  Creates shutdown string to be given to wittyPi as parameter.
- int getSleepDuration (char *mode)

Get length of time to sleep for.

- void kerOn ()

  Turn on kerlink Gateway.

- void kerOff ()

  Turn off kerlink Gateway.

- int getVoltMode (char ∗mode)

  Check if voltages should be read or simulated.

- void **init** ()
- int main (int argc, char ∗argv[ ])

  Handles phidget interactions, state machine, logging.

Variables

- CPhidgetTextLCDHandle **LCD**
- CPhidgetInterfaceKitHandle **IFK**
- char ∗ **batteryLogLocation**
- char ∗ **snapshotLocation**
- char **bootflag**
- char ∗ **mode**
- int **sleepDuration**
- int **sleepTime**
- int **wakeTime**
- int **voltMode**
- int **batteryDeplete**
- FILE ∗ **logFile**

3.2.1 Detailed Description

State machine + logging for power management daemon.

Author

Robert Cooney

Date

23 April 2018

3.2.2 Function Documentation

3.2.2.1 clearSnapFile()

```
int clearSnapFile (
            char * snapFileName )
```

Clears snapshot file.

Parameters

snapFileName	Name of the snapshot file

3.2.2.2 concat()

```
char * concat (
            const char * s1,
            const char * s2 )
```

Concatenates strings.

Parameters

s1	First string to be concatenated
s2	Second string to be concatenated

Returns

Result (s1 + s2)

3.2.2.3 createShutdownString()

```
char * createShutdownString (
            int timeToWait )
```

Creates shutdown string to be given to wittyPi as parameter.

Parameters

timeToWait	How long to wait until shutting down

Returns

String used to tell Pi when to shutdown

3.2.2.4 createStartupString()

```
char * createStartupString (
            int timeToSleep )
```

Creates startup string to be given to wittyPi as parameter.

Parameters

timeToSleep	How long to sleep for in seconds

Returns

String used to tell Pi when to startup

3.2.2.5 getSleepDuration()

```
int getSleepDuration (
            char * mode )
```

Get length of time to sleep for.

Parameters

mode	Current operation mode

Returns

How long to sleep for in seconds

3.2.2.6 getStateDesc()

```
char * getStateDesc (
            int state )
```

Gets description of current state as string.

Parameters

state	The current state of device - Sleep/low power/override/up

Returns

State as a string

3.2.2.7 getTimeString()

```
void getTimeString (
            int wakeTimeSec,
            char * wakeTimeStr )
```

Gets time as a string.

Parameters

wakeTimeSec	How long from now the time is in seconds
wakeTimeStr	The desired time in string format

3.2.2.8 getVoltMode()

```
int getVoltMode (
            char * mode )
```

Check if voltages should be read or simulated.

Parameters

mode	Simulate string parameter (SIM or NOSIM)

Returns

Int that describes mode (simulate or no simulate)

3.2.2.9 main()

```
int main (
            int argc,
            char * argv[] )
```

Handles phidget interactions, state machine, logging.

Clearing Pi shutdown and startup times

< Set up handlers for the Interface Kit & TextLCD

Get simulated voltages if necessary, otherwise get actual values
Continue loop until time to shutdown arrives

3.2.2.10 updateLogfile()

```
int updateLogfile (
            FILE * logfile,
            int voltage,
            int amps,
            int dutAmps,
            int state,
            int spiking,
            const char * progName )
```

Updates snapshot file showing snapshot of recent activity.

Parameters

logfile	Name of log file
voltage	Current voltage
amps	Current amps
dutAmps	Current amps of device under test
state	Current state
spiking	Whether voltage is spiking
progName	Name of running program

Returns

0 if all ok, -1 if error occurs

3.2.2.11 updateSnapshotfile()

```
int updateSnapshotfile (
            const char * snapFileName,
            int voltage,
            int amps,
            int state,
            int spiking,
            char * mode )
```

Updates snapshot file showing snapshot of recent activity.

Parameters

snapFileName	Name of snapshot file
voltage	Current voltage
amps	Current amps
state	Current state
spiking	Whether voltage is spiking
mode	Current mode

Returns

0 if all ok, -1 if error occurs

3.3 C:/Users/Robert/iCloudDrive/Documents/loradtn-pi/stable/phidgets.c File Reference

Functions for interacting with phidgets.

```
#include "phidgets.h"
```

Macros

- #define **PATH_MAX** 1024
- #define **LCDINDEX** 0
- #define **IFKINDEX** 0
- #define **BUFFER_SIZE** 21

Functions

- void **display_generic_properties** (CPhidgetHandle phid)
- void **display_IFK_properties** (CPhidgetInterfaceKitHandle phid)
- void **display_LCD_properties** (CPhidgetTextLCDHandle phid)
- int **getVoltage** (CPhidgetInterfaceKitHandle IFK)
- int **getAmps** (CPhidgetInterfaceKitHandle IFK)
- int **getDUTAmps** (CPhidgetInterfaceKitHandle IFK)
- void **closeCPhidget** (CPhidgetHandle handle)
- int **testVoltageSpike** (int ∗prevVoltage, time_t ∗prevVoltageTime, int ∗voltage, time_t ∗voltageTime)
- int **phidgetInit** ()
- CPhidgetInterfaceKitHandle **createInterfaceKit** ()
- CPhidgetTextLCDHandle **createLCDHandle** ()
- void **setupHandlers** (CPhidgetInterfaceKitHandle IFK, CPhidgetTextLCDHandle LCD)
- void **openPhidgets** (CPhidgetInterfaceKitHandle IFK, CPhidgetTextLCDHandle LCD)
- int **checkLCD** (CPhidgetTextLCDHandle LCD, CPhidgetInterfaceKitHandle IFK)
- void **setStartupDisplay** (CPhidgetTextLCDHandle LCD)
- int **checkIFK** (CPhidgetInterfaceKitHandle IFK, CPhidgetTextLCDHandle LCD)
- void **spikeError** (int spikeCount, CPhidgetTextLCDHandle LCD, CPhidgetInterfaceKitHandle IFK)
- int **updateDisplay** (int voltage, int amps, char ∗wakeTimeStr, char ∗stateDescription, CPhidgetTextLCD↩ Handle LCD, char ∗mode)

Variables

- char **topBuffer** [BUFFER_SIZE]
- char **bottomBuffer** [BUFFER_SIZE]

3.3.1 Detailed Description

Functions for interacting with phidgets.

Date

30 April 2018

Index