Spring 2017

# Smart Fan

Jacob A. Carroll
jac259@zips.uakron.edu

Joshua B. Blanchard
jbb47@zips.uakron.edu

Peter A. Gross
pag35@zips.uakron.edu

Joshua D. Riegel
jdr78@zips.uakron.edu

Smart Fan

Jacob Carroll

Department of Electrical and Computer Engineering

**Honors Research Project**

Submitted to

*The Honors College*

Approved:

[illegible] Date 17 Apr 2017
Honors Project Sponsor (signed)

Alex De Abreu Garcia
Honors Project Sponsor (printed)

[illegible] Date 4/17/17
Reader (signed)

Gregory A. Lewis
Reader (printed)

[illegible] Date 04/21/17
Reader (signed)

Nghi Tran
Reader (printed)

Accepted:

[illegible] Date 04/24/17
Department Head (signed)

Joan Carletta
Department Head (printed)

[illegible] Date 04/21/17
Honors Faculty Advisor (signed)

Nghi Tran
Honors Faculty Advisor (printed)

____________ Date ________
Dean, Honors College

# Honors Research Project

## Smart Fan

The goal of the Smart Fan project is to design and implement a programmable, Internet-connected box fan that offers users fine control over the specifics of its operation. The Smart Fan boasts a powerful DC motor controlled fully by a Raspberry Pi, a small computer located on the physical device. This computer runs scripts to use both user input and sensor data to operate the fan in accordance with the user's choices. The user offers input to the system through an Android smartphone application, and can do so anywhere in the world so long as both the smartphone and Smart Fan maintain an Internet connection.

**Jacob Carroll**, Computer Engineering, is the Software Lead responsible for coordinating the software effort of the design process. This includes overseeing the development of all software for the project and assisting in making high-level design choices, such as the general functionality of each operation mode of the Smart Fan. He is also responsible for developing the entirety of the Android application used to control the Smart Fan. This entails the creation of a robust and aesthetically pleasing application that offers intuitive control over each aspect of the operation of the Smart Fan. The application must be full-bodied and visually pleasant, but it must also minimize wireless data usage and battery consumption. This requires significant design consideration when developing the application. The resulting software allows the Smart Fan to function excellently and it has received high praise from those who have seen it.

# Smart Fan

Project Design Report

Design Team 11

Joshua Blanchard

Jacob Carroll

Peter Gross

Joshua Riegel

Dr. De Abreu-Garcia

April 28th, 2017

**Table of Contents**

**List of Figures**

**List of Tables**

## Abstract

Modern box fans offer a simple solution to cooling an area. However, the simplicity that makes these fans appealing also limits their potential. A smart box fan that preserves its original simplicity while providing enhanced technological features such as smartphone connectivity and variable speed selection should lead to ease of use, improved performance, and increased customer satisfaction. To this end, a regular box fan will be instrumented and technology-enhanced, thus "smart fan." The standard AC motor will be replaced with a DC motor for finer speed selection over the allowable range. The smart fan will offer three categories of operation: manual control, scheduled operation, and temperature-sensitive automation. Manual control operates the smart fan similarly to a standard box fan. Scheduled operation specifies the operation of the fan for any time. Temperature-sensitive automation will offer two modes. The first is a single-source mode that operates using a local temperature sensor. The second is a dual-source mode–intended for use in a window–that uses the temperature sensor and a weather API. Both modes operate by mapping speeds to a range of temperatures. The smart fan will allow users to run it when they want, how they want.

[JAC]

## Key Features

1. Worldwide Remote Access
2. Variable Speed Control
3. Temperature Sensing Operation Mode
4. Scheduled Operation Mode
5. Manual Operation Mode
6. Bidirectional Blade Rotation
7. Physical Button Interface

[JBB]

# 1. Problem Statement

## 1.1 Project Need

As technology advances, users demand more options and sophisticated features from their devices. With the increased accessibility of the Internet and the advent of smartphones, one such feature is smartphone connectivity. A device that has remained stagnant in its implementation is the common box fan. Typical box fans are limited to three speeds which are usually controlled by a physical knob. The potential of a box fan is limited by this simplicity. Adding smartphone connectivity and programmatic operation, the usefulness of the box fan can be greatly increased, as it will run when and how the user wants. For example, desirable options would include temperature sensing, scheduled operations, a wide breadth of fan speeds, and bidirectional airflow. Being able to control the device from a smartphone would allow users the ability to update its runtime parameters from anywhere at any time. Further, conditional operations would also allow a user to set up the device once, and then never have to worry about it again – set it and forget it.

[JBB]

## 1.2 Project Objective

The objective of this project is to design a system by which a box fan is controlled by a smartphone application via a microprocessor. Using a microprocessor adds programmability and precision control over the operating range of the box fan. This includes, but is not limited to, usage of a continuous speed curve as opposed to a set of finite discrete speeds. The application would grant the user direct control over the operation of the fan, as well as display information such as room temperature and running times. The system would also have internet connectivity to allow the application to control the fan remotely via an internet connection. The system could be programmed to run the fan under pre-specified conditions, such as during a specific time of day or when the room reaches a given temperature threshold. Other features might include motion activation and sleep settings to run the fan for a specified time duration and to help reduce total running time and power consumption.

[JAC]

## 1.3 Research Survey

A device to provide an improved electrical circuit for an air conditioner with a temperature switch is described in patent #5,088,645. The device described in this patent can respond quickly and operate reliably without regard to the condition of the air circulating system. The electrical circuit of this air conditioning unit has several components. There is a series-connected temperature switch that opens when the temperature of the motor reaches a predetermined threshold. There is a thermostat which positions its connector piece to “open” or “closed” by comparing the room temperature with a user-set temperature level. There is also a relay which turns the compressor on or off. Finally, there is a second temperature switch which opens when the electric heater temperature reaches a given threshold.

A self-programming temperature control system to automatically adjust the temperature control set point of a heating and cooling system based upon the present or anticipated occupancy status of the space to be air-conditioned is described in patent #5,131,455. The control system detects the occupancy status of the space and creates an occupancy record at regular time intervals. The stored time-stamped data is then provided to a programmed processor, which is instructed to read and process the past occupancy record of the space to derive an anticipated occupancy status for the space and thus control the temperature of the space accordingly.

A standard fan design with the addition of a microcontroller, memory, and the ability to interface with a host computer is described in patent #6,318,965. The host computer is able to communicate with and send commands to the microcontroller. The microcontroller processes these commands and then dynamically controls the fan speed via an attached motor. The device also has an optional temperature sensor to sense and relay ambient temperatures to the host computer. These sensor readings are then used to generate commands to control the fan speed to maintain a relative temperature.

A controller that can be attached to different kinds of fans and turn them on or off is described in patent #8,441,155. The controller uses a programmable schedule or schedules to determine when the fan will turn on and off. The controller has up to three buttons that can be used to override a programmed schedule. These buttons can also override a programmed schedule for a set amount of time.

Cui, et al., describe a design for a smart fan that uses a Programmable System-on-Chip (PSoC) as its main control [5]. The control, with the support of an external auxiliary circuit and wireless remote control, allows functions such as switching between manual control and temperature control, natural wind, etc.

The design described by Pang, et al., of a microcontroller-based motor controller with a heat sensor is used to vary the speed of a fan motor with respect to ambient temperature in a smart home [6]. The controller is embedded into a stand fan. A phase control method is selected to implement the design. The power delivery to the motor is determined by the firing angle of a Triac that controls the AC power supply. The Triac allows precise control over the input power to the motor. In addition, a hand-clap circuit is used as a switch to activate or deactivate the motor.

Guillot, et al., describe a design for a fan control unit that would require high motor currents, of up to 30A, and a 20kHz Pulse Width Modulation (PWM) signal [7]. Using a programmable integrated circuit they are able to monitor the current during switching as well as the electromagnetic interference (EMI) that occurs with high current and high frequency switching. This allows Guillot, et al., to counteract some of that interference using a 16-bit microcontroller and a control area network to interface with the control unit.

[PAG, JAC]

## 1.4 Marketing Requirements

1. The smart fan should be comparable or superior to current box fans in terms of minimum and maximum speed, dimensions, noise level, and energy rating.
2. The smart fan should be able to receive input from two temperature sources and make adjustments to its functionality accordingly.
3. The smart fan should have standard physical controls.
4. The smart fan should be affordable.
5. The Android application should have a robust, easy-to-use interface.
6. The Android application should allow users to control all fan settings remotely.
7. The Android application should require minimal power and cellular data consumption.

[PAG]

## 1.5 Objective Tree

The Priority Objective Tree shown in Figure 1 demonstrates the priority levels for the various features offered by the smart fan.

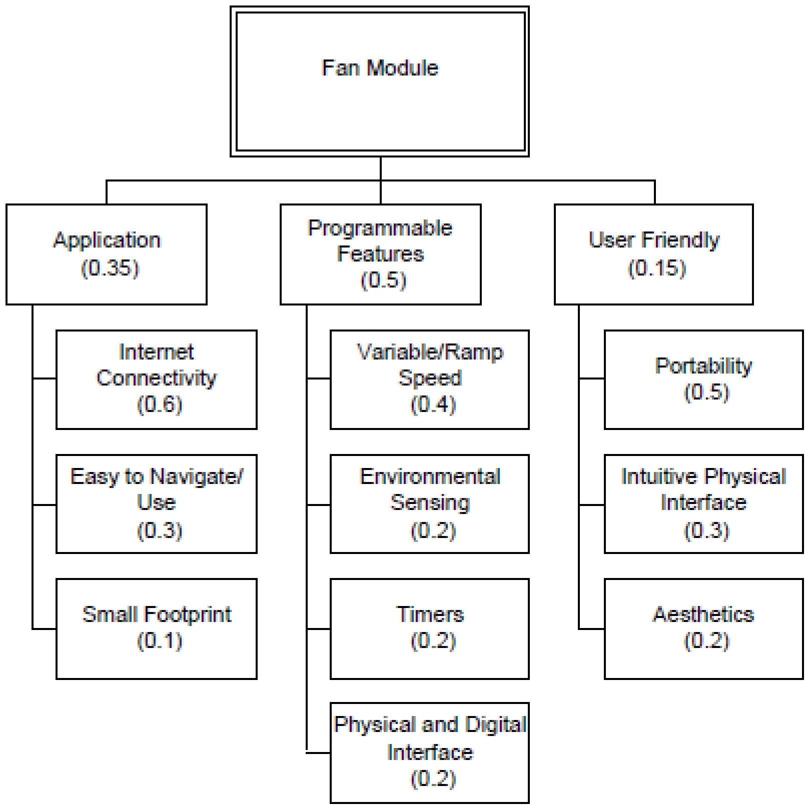

Fan Module
Application (0.35)
Programmable Features (0.5)
User Friendly (0.15)
Internet Connectivity (0.6)
Easy to Navigate/ Use (0.3)
Small Footprint (0.1)
Variable/Ramp Speed (0.4)
Environmental Sensing (0.2)
Timers (0.2)
Physical and Digital Interface (0.2)
Portability (0.5)
Intuitive Physical Interface (0.3)
Aesthetics (0.2)

*Figure 1. Priority Objective Tree.*

[JBB, JAC, JDR]

## 2. Design Requirements Specification

*Table 1. Smart Fan Design Requirements.*

| Marketing Requirements | Engineering Requirements | Justification |
|---|---|---|
| Supports #1<br>Opposes #4 | 1. The motor should have a minimum speed rating of 3600 rpm. | To compete with current fans, the smart fan should have a comparable max speed and wider range of speeds. |
| Supports #7<br>Opposes #6 | 2. The Android application should use less than 30MB of cellular data per month. | Most cellular data packages have data caps and users would not want a single application to dominate their usage. |
| Supports #7<br>Opposes #5, #6 | 3. The Android application should use less than 5% of the smartphone's battery life per day. | Smartphones often have batteries that are drained within less than a day. Users would not want a single application to drain their phone battery. |
| Supports #2 | 4. The temperature sensor(s) should be accurate to within ±1°F. | Minute changes in temperature are undetectable by an average person, but accuracy will still be important to display the correct temperature. |
| Supports #3<br>Opposes #6 | 5. The physical interface of the fan should update the fan speed within 2 seconds. | Standard box fans are able to change speeds within seconds; the smart fan should be comparable. |
| Supports #5<br>Opposes #6, #7 | 6. The Android application should update the fan's operation parameters within 2 seconds. | The application should have latency comparable to that of the physical controls of the smart fan. |
| Supports #1 | 7. The pulse width modulation of the microprocessor should have a frequency of at least 25kHz. | To avoid the audible range of humans, the frequency of the PWM must be greater than 20kHz. 25kHz is chosen to account for any possible deviation. |
| Supports #1 | 8. The noise level should be within ±5 dB of a standard box fan for a given RPM. | At comparable RPM the noise level of the smart fan should be within ±5 dB of standard box fans. |
| Supports #1 | 9. The motor should operate with at least 25 discrete speeds. | Standard box fans only offer three speeds; to simulate a continuous curve of speed, more steps are required. |
| Supports #1, #2<br>Opposes #3 | 10. The smart fan should offer at least 4 operation modes. | To offer a substantial improvement over standard box fans the smart fan |

| | | |
|---|---|---|
| | | will need to nontrivially exceed the functionality of standard box fans. |
| Marketing Requirements<br>1. The smart fan should be comparable or superior to current box fans in terms of minimum and maximum speed, dimensions, noise level, and energy rating.<br>2. The smart fan should be able to receive input from two temperature sources and make adjustments to its functionality accordingly.<br>3. The smart fan should have standard physical controls.<br>4. The smart fan should be affordable.<br>5. The Android application should have a robust, easy-to-use interface.<br>6. The Android application should allow users to control all fan settings remotely.<br>7. The Android application should require minimal power and cellular data consumption. | | |

[JBB, JAC, JDR, PAG]

## 3. Accepted Technical Design

### 3.1 Hardware Theory of Operation

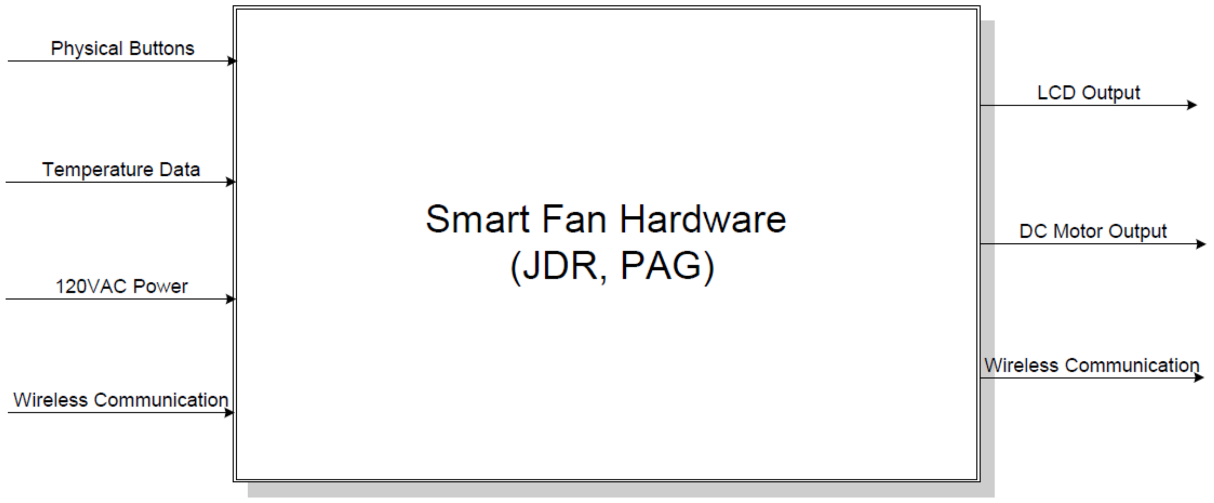


*Figure 2. Hardware Block Diagram Level 0.*

*Table 2. Hardware Block Diagram Level 0: Smart Fan Hardware.*

| *Module* | Smart Fan Hardware |
|---|---|
| *Inputs* | ○ Physical Buttons – Physical button interface for manual override.<br>○ Temperature Sensing – Temperature sensor to read ambient room temperature.<br>○ 120VAC Power – Power from the 120V wall outlet.<br>○ Wireless Communication – Wireless data from the phone application. This data will contain the user's inputs and selections on the fan's function. |
| *Outputs* | ○ LCD Output – Fan module to display basic information to the user like fan speed, room temperature, and current operation mode.<br>○ DC Motor Output – The motor of the box fan, selected to rotate in both directions at speeds of at least 3600 rpm.<br>○ Wireless Communication – Wireless data from the sensors and microprocessor to be sent to the phone application and web server. |
| *Functionality* | This is the main fan module. It will house all of the desired functions, such as various operation modes, sensor data, PWM signal for motor speed and direction control. |

[PAG]

To control the fan and allow the system to run smoothly, there will be four main inputs and three main outputs, as shown in Figure 2. The four main inputs are described in Table 2. The smart fan will be fed from a 120V AC wall outlet because it will be stationary during most, if not all, of its lifetime, and thus it does not need a portable power supply. The main input to the system will be the Android application, which users will control from a mobile device. This will allow users to remotely control the fan from anywhere. Users will have the ability to turn the fan on as they leave work or while on their way home, as well as control the operation mode accordingly. The fan will also have a physical button interface, located on the top of the unit, that will allow the user to control the fan without using the phone application. There will be a separate manual mode for the physical control buttons. The last main input to the system will be the outputs of the temperature sensors. These sensors will allow the fan to sense the ambient temperature of the room while in the temperature sensing mode.

One of the main outputs of the system will be displayed in a Liquid Crystal Display (LCD), giving the users basic information about the fan, including speed settings in manual mode. The DC motor will receive a PWM signal from the microprocessor that will control both its speed and direction of rotation. Lastly, the microprocessor will send data to the phone application, updating the current settings within the application to keep the application and microprocessor synchronized.

The system will be broken up into six subsystem modules shown in Figure 3. Those will consist of the power supply, the display, the sensor, the motor control, the physical interface, and the processor. Each module has their own external input or output. The processor itself has either an input or an output coming to or from each of the other modules. For inputs, the motor will receive 120VAC, the sensors will measure the temperature, the buttons on the physical interface will be pressed, and the processor will receive wireless data from the web server or android application. For outputs, the display will show information depending on the operation and the motor driver will cause the motor to run at the set speed. Table 3 through Table 8 explain more in depth the inputs and outputs of each subsystem.

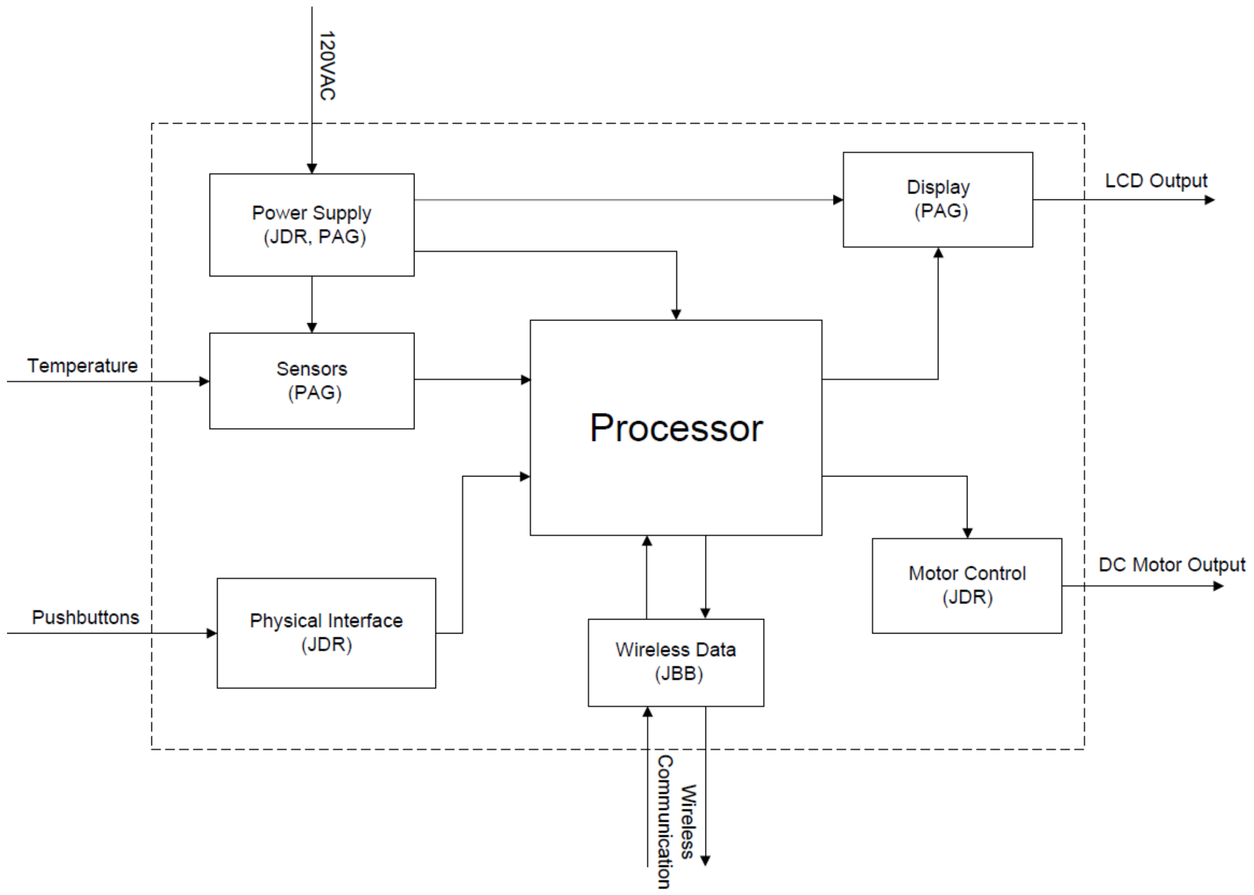


*Figure 3. Hardware Block Diagram Level 1.*

The power supply will take in 120VAC from a wall outlet and provide power rails to all other modules of the system. The power supply will feature its own state setting switch that would power or turn off the system completely, with no other way of turning the system back on other than having the state of the switch turned back to on.

*Table 3. Hardware Block Diagram Level 1: Power Supply.*

| | |
|---|---|
| *Module* | Power Supply |
| *Inputs* | ○ 120VAC – Power from wall outlet, 120VAC at 60Hz. |
| *Outputs* | ○ Display Power – Power to LCD display.<br>○ Microprocessor Power – Power to microprocessor (5V, 2A).<br>○ Sensor Power – Power to temperature sensors (5V).<br>○ Motor Power – Power to DC Motor (24V). |
| *Functionality* | The power supply will be designed by the team. While wall outlets will be used as the main power source, multiple DC voltage signals to power the different system components will be needed. The AC voltage signal will be used to generate the various power levels that will be needed throughout the system. |

*Table 4. Hardware Block Diagram Level 1: Sensors.*

| *Module* | Sensors |
|---|---|
| *Inputs* | ○ Room Temperature – Ambient temperature to be recorded by the temperature sensor to be used in the temperature operation mode.<br>○ Sensor Power – Power rail from the power supply. |
| *Outputs* | ○ Sensor Data – Sensor voltage that is translated to a temperature value by the microprocessor using a calibration curve. |
| *Functionality* | ○ The sensors will allow information about the ambient temperature of the room to be used in the temperature-sensitive operation modes. The data will be compared with the user settings and the temperature data from a weather API to determine the fan speed and/or direction. |

*Table 5. Hardware Block Diagram Level 1: Physical Interface.*

| *Module* | Physical Interface |
|---|---|
| *Inputs* | ○ Physical Buttons – Buttons that users can use should their phone be out of reach or the application be unavailable. |
| *Outputs* | ○ Button Signals – Electrical signals resulting from pressing the physical buttons. |
| *Functionality* | The physical interface will allow for a backup way to operate the fan if the phone application happens to fail. It will give the user enough information to manually operate the fan, easily and intuitively. |

*Table 6. Hardware Block Diagram Level 1: Microprocessor.*

| | |
|---|---|
| *Module* | Microprocessor |
| *Inputs* | ○ Microprocessor Power – Power rail from the power supply.<br>○ Sensor Data – Data from the temperature sensors.<br>○ Button Signals – Signals from the physical interface buttons. If activated, will override the current operation mode and activate the manual operation mode.<br>○ Wireless Commands – Updates from the phone application about newly selected operation mode and modified settings. |
| *Outputs* | ○ Display Controls – Parallel communication control to the LCD screen.<br>○ PWM Motor Control – PWM signal to control the speed and direction of the DC motor.<br>○ Wireless Application Update Info – Wireless updates to the phone application with data like current temperature, PWM value, and current operation mode. |
| *Functionality* | A Raspberry Pi 3 (RPi) will be the main controller of the hardware. It will take in all of the data, process it and output signals to control all of the fan's operations. It will take data from the phone application, as well as data from the sensors and fan buttons to decide to what value to set the PWM, what information to send to the LCD to display, and to control the motor to the correct speed. It will also store information about the overall state of the system, including the current temperature and PWM value. |

*Table 7. Hardware Block Diagram Level 1: Display.*

| | |
|---|---|
| *Module* | Display |
| *Inputs* | ○ Display Power – Power rail from the power supply.<br>○ Display Controls – Parallel communication control from the microprocessor. This will tell the LCD what and when to display. |
| *Outputs* | ○ LCD Output – Fan module to display basic information to the user like fan speed, room temperature, and current operation mode. |
| *Functionality* | The main function of the LCD is to display feedback information which would allow users to operate the smart fan with the physical interface instead of the phone application. Feedback will include information such as operation mode and fan speed. |

*Table 8. Hardware Block Diagram Level 1: Motor Control.*

| | |
|---|---|
| *Module* | Motor Control |
| *Inputs* | ○ Motor Power – Power rail for the DC motor.<br>○ PWM Motor Control – PWM signal from the microprocessor to control the fan speed and direction. |
| *Output* | ○ DC Motor Output – After the PWM signal goes through a motor controller circuit it will be sent to the DC motor to set the fan speed and direction. |
| *Functionality* | The purpose of the motor control is to allow a low current PWM signal from the microprocessor to run and change the DC motor speed, which requires a higher voltage and current input to operate properly. |

[PAG, JDR]

**Error! Reference source not found.** shows more depth with the system. Each subsystem module shows the general purpose and the blocks that will be designed. The power supply includes the switch, some reverse current protection, and a transformer with regulators to obtain the rail voltages. The sensors will take in their sampling data which is sent to the microprocessor. The LCD will take in information to display and will be powered by a voltage rail going through a resistor network. The interface will consist of a power switch, a switch for direction of airflow, and a rotary encoder to adjust the speed. The motor control will consist of protection from high switching current that will be resulting from the H-bridge circuit designed to control the input to the DC motor. The wireless communication will send and receive data to and from the android application and web server. The wireless communication will also collect data from a weather API to be used alongside the two-source temperature control operation. These blocks and their respective sub-blocks are described in more detail in Table 9 through Table 14. Each table details one of the larger blocks and describes each of its sub-blocks to consolidate the related sub-blocks into one table.

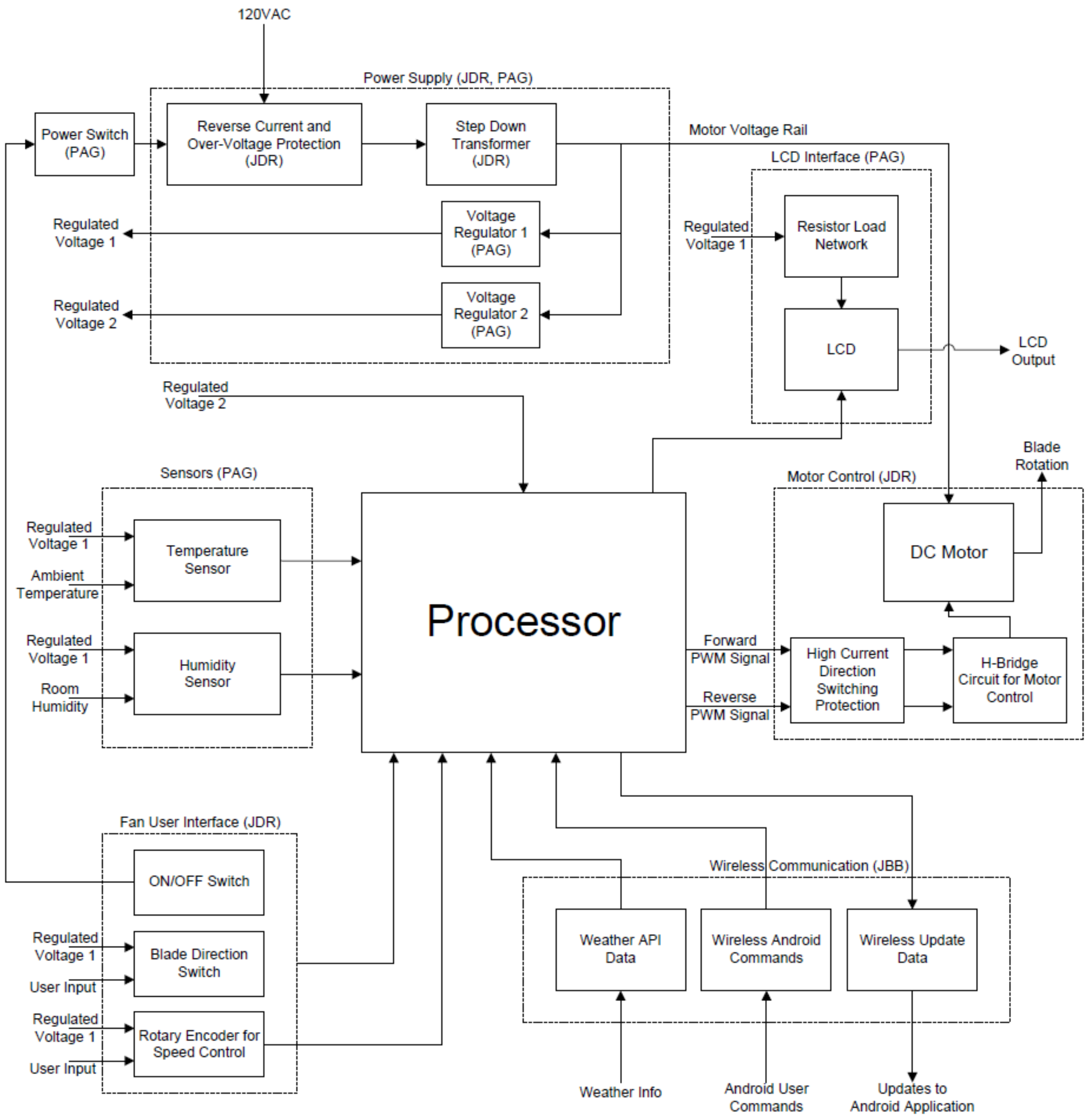


*Figure 4. Hardware Block Diagram Level 2.*

*Table 9. Hardware Block Diagram Level 2: Power Supply.*

| *Module* | Power Supply |
|---|---|
| *Inputs* | ○ Reverse Current Protection Circuit<br>▫ 120VAC power source from wall outlet.<br>▫ ON/OFF Switch.<br>○ Step Down Transformer<br>▫ Protected AC voltage signal.<br>○ Voltage Regulator Circuits<br>▫ Stepped down DC voltage from the transformer. |
| *Outputs* | ○ Reverse Current Protection Circuit<br>▫ Protected AC Voltage Signal.<br>○ Step Down Transformer<br>▫ Stepped down 24V DC voltage line for the motor.<br>○ Voltage Regulator Circuits<br>▫ Regulated voltage lines supplied for the various components. |
| *Functionality* | ○ Reverse Current Protection Circuit<br>▫ Protects the power supply and the entire system from reversing the input power or from inputting a higher voltage than expected.<br>○ Step Down Transformer<br>▫ Takes in the protected AC voltage signal from the wall outlet, it steps it down to 24V using a transformer and then rectifies the stepped down AC signal to a DC signal to power the motor.<br>○ Voltage Regulator Circuits<br>▫ To provide sufficient power to the subsystem modules. Two of these will be needed for different voltage lines. |

*Table 10. Hardware Block Diagram Level 2: LCD Interface.*

| *Module* | LCD Interface |
|---|---|
| *Inputs* | ○ Resistor Load Network<br>▫ Regulated Voltage 1.<br>○ LCD Controls<br>▫ Values to set the proper display state of the LCD. |
| *Outputs* | ○ Resistor Load Network<br>▫ Divided voltage and current that will meet the LCD specifications.<br>○ LCD Output<br>▫ Physical display that provides basic information to the user like fan speed, room temperature, and current operation mode. |
| *Functionality* | ○ Resistor Load Network<br>▫ Needed to properly load the LCD screen and control the current going into the LCD.<br>○ LCD<br>▫ Provides the user the necessary information to operate the fan in manual mode. |

*Table 11. Hardware Block Diagram Level 2: Motor Control.*

| *Module* | Motor Control |
|---|---|
| *Inputs* | ○ Direction Switching Circuit Protection<br>▫ Forward PWM signal from the processor.<br>▫ Reverse PWM signal from the processor<br>○ H-Bridge Motor Control Circuit<br>▫ Protected forward signal.<br>▫ Protected reverse signal.<br>○ DC Motor Output<br>▫ 24V from the power supply.<br>▫ Speed control signal. |
| *Outputs* | ○ Direction Switching Circuit Protection<br>▫ Forward protected output.<br>▫ Reverse protected output.<br>○ H-Bridge Motor Control Circuit<br>▫ Speed control signal.<br>○ DC Motor Output<br>▫ Mechanical motion for the fan blades. |
| *Functionality* | ○ Direction Switching Circuit Protection<br>▫ Provides hardware protection in addition to the software protection against switching the direction of the blades and causing dangerous high currents in the motor.<br>○ H-Bridge Motor Control Circuit<br>▫ Controls the motor by giving the proper signal to control the speed based on the PWM signal given by the processor, also easily allows the direction of the motor to be changed.<br>○ DC Motor Output<br>▫ Allows more fluidity in speed selection as compared to an AC induction motor. |

*Table 12. Hardware Block Diagram Level 2: Sensors.*

| *Module* | Sensors |
|---|---|
| *Inputs* | ○ Temperature Sensor<br>▫ Regulated voltage from the power supply.<br>▫ Temperature measurement samples.<br>○ Humidity Sensor<br>▫ Regulated voltage from the power supply.<br>▫ Humidity measurement samples. |
| *Outputs* | ○ Temperature Sensor<br>▫ Analog voltage fed to the processor.<br>○ Humidity Sensor<br>▫ Analog voltage fed to the processor. |
| *Functionality* | ○ Temperature Sensor<br>▫ Providing temperature information to the processor to facilitate the operation of the temperature sensing mode and to display to the user.<br>○ Humidity Sensor<br>▫ Providing humidity information to the processor to display to the user. |

*Table 13. Hardware Block Diagram Level 2: Fan User Interface.*

| *Module* | Fan User Interface |
|---|---|
| *Inputs* | ○ Blade Direction Switch<br>▫ Regulated voltage from the power supply.<br>▫ User input to toggle the switch.<br>○ Rotary Encoder for Speed Control<br>▫ Regulated voltage from the power supply.<br>▫ User input to rotate the encoder. |
| *Outputs* | ○ ON/OFF Switch<br>▫ Toggle switch to activate power supply and power the system.<br>○ Blade Direction Switch<br>▫ Toggle switch to the processor to change the direction of the blade movement.<br>○ Rotary Encoder for Speed Control<br>▫ Output to the processor to adjust the fan speed. |
| *Functionality* | ○ ON/OFF Switch<br>▫ Allows the user to power the system on or off without using the Android application.<br>○ Blade Direction Switch<br>▫ Allows the user to change the direction of the fan blades while the system is in manual mode,<br>○ Rotary Encoder for Speed Control<br>▫ Allows the user to adjust the speed of the fan while the system is in manual mode. |

*Table 14. Hardware Block Diagram Level 2: Wireless Communication.*

| *Module* | Wireless Communication |
|---|---|
| *Inputs* | ○ Weather API Data<br>▫ Weather information.<br>○ Wireless Android Commands<br>▫ Android application user commands.<br>○ Wireless Update Data<br>▫ Information from processor about the state of the system. |
| *Outputs* | ○ Weather API Data<br>▫ Weather data – outside temperature and humidity.<br>○ Wireless Android Commands<br>▫ Commands to change the operation mode of the fan.<br>○ Wireless Update Data<br>▫ Update information to the Android application. |
| *Functionality* | ○ Weather API Data<br>▫ Tells the processor what the outside temperature and humidity are to make decisions while in temperature sensing mode.<br>○ Wireless Android Commands<br>▫ Based on the user's input in the application, the command will alter the settings that determine the operation of the system.<br>○ Wireless Update Data<br>▫ Updates the application on the current operation mode, PWM signal and other relevant system information. |

[JDR, PAG]

## 3.2 Motor Control

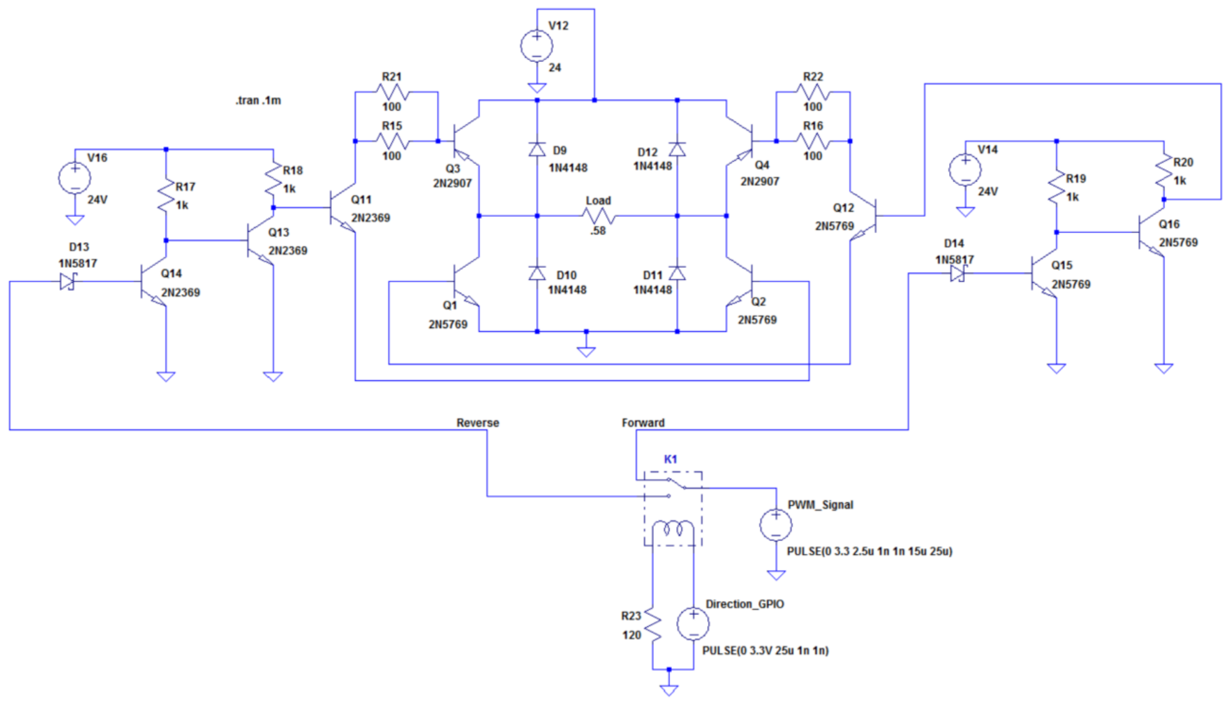


*Figure 5. Schematic for H-bridge motor control circuit.*

[JDR]

### 3.2.1 Theory of Operation

The circuit above in Figure 5 will control and drive the DC motor. The two halves of the circuit are identical and consist first of a PWM signal that will come from one of the General Purpose Input Output (GPIO) pins on the RPi. The PWM signal will control the speed of the motor using a hardware PWM signal. The hardware PWM can switch at much higher frequencies and thus was chosen over the software PWM. A separate GPIO pin on the RPi will control the direction of the motor's rotation through a relay. When the GPIO pin is set to a high voltage, current will run through the coil of the relay. This current will switch the relay on, and change which circuit the PWM signal is connected to. The relay will default to one of the sides so when the GPIO pin is set to a low voltage, the motor will run in the forward direction. When the GPIO pin is set to a high voltage the motor will run in the reverse direction.

The PWM voltage source–the leftmost portion–it is set to have a period of 25µs and a duty cycle of 60%. These values fully demonstrate the functionality of the circuit; the PWM signal from the RPi has been set at 40kHz for this project. This frequency is above the 20kHz threshold of the human hearing range and is slow enough to obtain a useful period of 25µs. This is explained further in Section 3.14 Software Calculations. This signal goes through a Schottky diode to a transistor in order to protect the system against reverse currents. The RPi can only output a signal with a 3.3V peak voltage, so two transistors are used to switch to a higher voltage rail to direct more current through the load. The new, higher-voltage signal is directed into the base of transistor Q5 in Figure 5, putting the transistor into saturation mode. In saturation mode the transistor will function as a switch. Once Q5 is turned on, current is allowed to enter the base of transistors Q2 and Q3. When these two transistors are turned on current will flow through the motor, represented as the resistor Load in the center of the H-bridge configuration in Figure 5, and the motor will rotate forward. The described circuit is mirrored over the H-bridge in Figure 5. This mirrored section is connected to the other circuit of the relay and will drive the motor in reverse.

Figure 6 shows a simulation of the PWM signal and the load's reaction to the PWM. The signal is at the bottom and the load reaction is at the top. The signal is set to have a period of 25µs and a duty cycle of 60%. This simulation shows how well the load reacts to the signal when the right transistors are chosen.

[JDR]

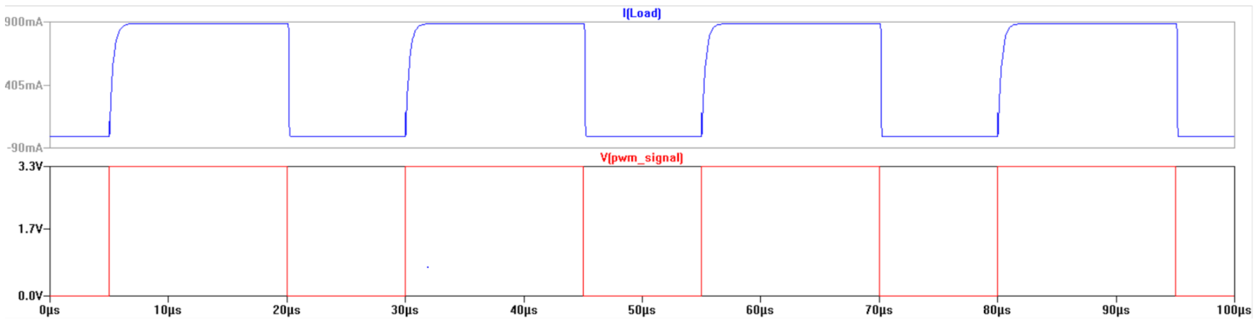


*Figure 6. Simulation showing current through H-bridge load.*

Figure 7 and Figure 8 show the current into the base of the four transistors in the H-bridge circuit. In Figure 7, transistors Q2 and Q3 are "on" and reacting to the PWM signal just like how the load reacted in Figure 6 which means that the motor will be running in the forward direction and transistors Q1 and Q4 are "off" which is why they are seeing no current except for a few small spikes. In Figure 8, the motor is running in reverse which is why Q2 and Q3 are now "off", seeing the same small spikes as Q1 and Q4 in the previous figure and Q1 and Q4 are "on" and reacting now reacting to the PWM signal.

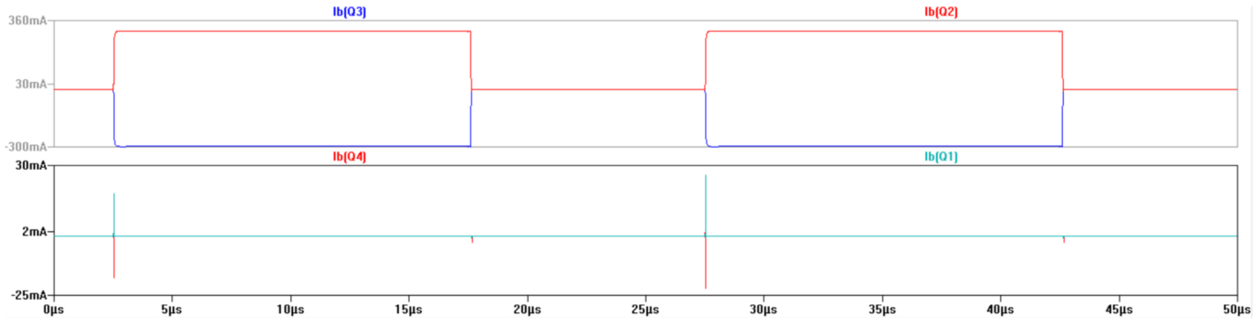


*Figure 7. The current into the base of the H-bridge transistors for forward rotation.*

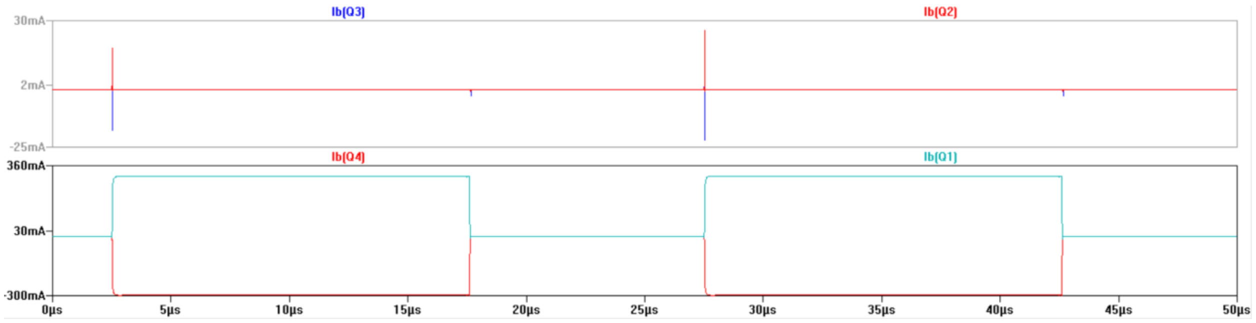


*Figure 8. The current into the base of the H-bridge transistors for reverse rotation.*

Figure 9 shows the function of the relay to control the direction of the motor. The GPIO pin for direction control is the signal on top, the PWM signal from the RPi is in the middle and the current through the load is on the bottom. This shows how the motor is not affected until the GPIO pin switches the direction.

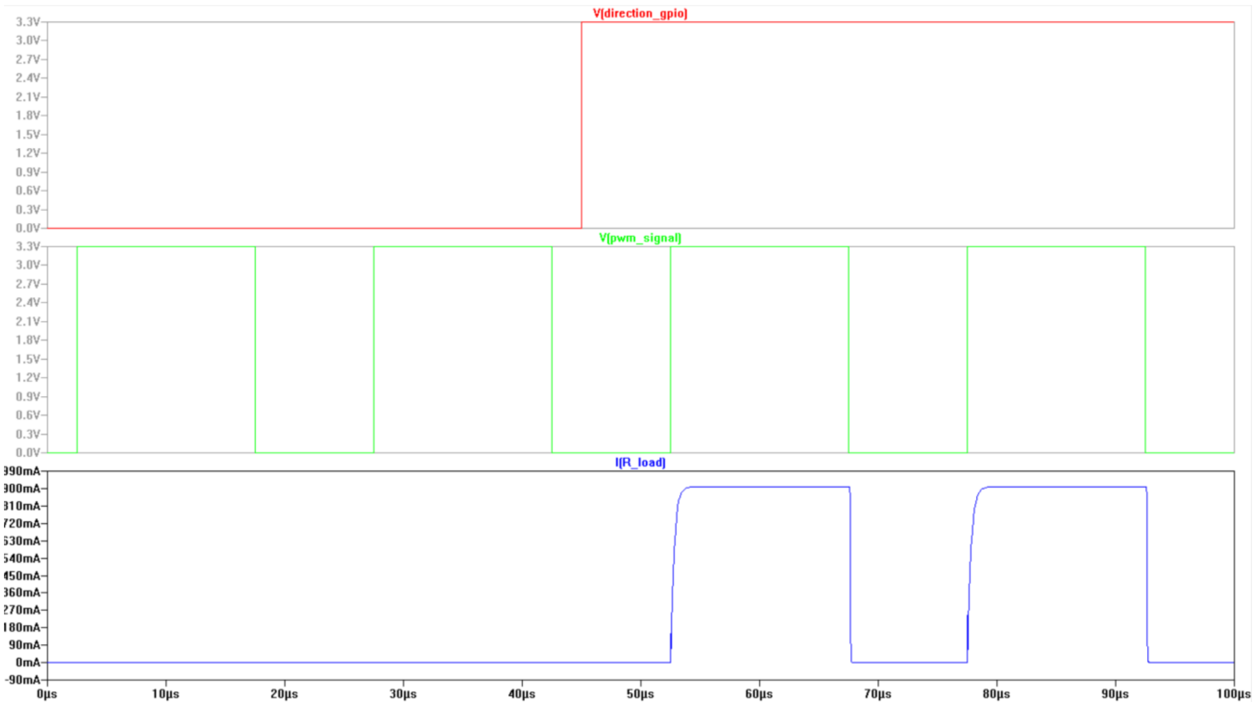


*Figure 9. Simulation showing the PWM being activated by the direction GPIO pin and relay.*

[JDR]

### 3.2.2 Component Selection

Each component was chosen to fit the needs of the circuit and specifications for the motor controller. The first components to be selected were the NPN transistors. These transistors need to switch quickly enough to accurately react to the PWM signal with a period of 25µs. An issue arose in that the components have an internal capacitance between the junctions. So even though most transistor switching times were in the nanosecond range, having a low internal capacitance was a high priority to account for this additional capacitance. In the simulation output shown in Figure 6, the input PWM signal is shown at the bottom and the current through the Load is at the top. The Load has a time constant associated with it that causes the motor to be off for some of the time when the PWM voltage is high. The PNP transistors were chosen with the same priority in mind except that the PNP transistors consume significantly more power than the NPN transistors.

The diodes used in the H-bridge circuit protect against reverse current and against current spikes that can occur when either the power supply or the motor is suddenly turned off or when the motor direction is changed too quickly. Standard diodes were chosen as they will not see much current or power. The Schottky diodes mentioned previously are needed to minimize the voltage drops on the PWM signal. The high voltage of the PWM signal is only 3.3V, so standard drops could nontrivially affect the signal. A diode is needed in series with the GPIO pin signal to protect against reverse current but the typical 0.7V voltage drop from a standard diode would impact the PWM signal too greatly. Schottky diodes have a smaller forward voltage drop, typically in the range of 0.15V to 0.4V. This allows for a protection diode that does not significantly reduce the PWM signal voltage.

The relay allows for the use of only one PWM signal because the relay can use a standard GPIO pin to switch direction instead of needing two PWM pins, one for each direction of rotation. The relay is a small 4.5V relay that will work well with the 3.3V output from the RPi. The relay was also chosen because a very small amount of current is needed through the coil to activate the internal switch of the relay. This benefits the system because each GPIO pin of the RPi can only output 16mA of current.

The resistor values were chosen to maximize the current through the Load and minimize the current and power consumed by the transistors in the H-bridge circuit. The 1kΩ resistors do not have a significant current draw as they do not need to handle much power. The 100Ω resistors are connected in parallel to achieve a smaller equivalent resistance and thus increase the current through the Load. These resistors will have a significant amount of current passed through them and as such were chosen to be higher-power through-hole resistors.

[JDR]

### 3.2.3 Motor Selection

The chosen motor is a brushed DC motor with a nominal voltage of 24V. This means that the motor will run at full power when given 24V. The speed of the motor will be regulated using the PWM signal from the RPi. Driving the motor with a PWM signal with a 50% duty cycle will cause the motor to run as if it were receiving 12V; that is, the motor will run at half speed. One of the main factors for choosing this motor is the relatively high stall torque. A high stall torque ensures that the motor will be able to start and run smoothly with the load of the fan blades attached to the drive shaft. Another major factor was the no-load RPM of 5600 RPM. Although it is common for brushed DC motors to have RPM ratings in this range, the specification still needed to be met. The size of the motor was another consideration. The motor is relatively small at 3.1 inches in diameter and 4.0 inches in length. The final consideration was the current the motor would draw, with a no-load current rated at 2.1 amps. This current draw is low compared to other motors with similar characteristics that were considered.

A tachometer was used to measure the speed of the motor at the three speeds. These measurements are shown in Table 15, in the Hardware Calculations section. Torque calculations were performed early in the design process but were disregarded later after they were found to be inaccurate as a result of limited knowledge of motor speed at each speed setting.

[JDR]

### 3.2.4 Feedback

In order to accurately control the motor speed there will need to be some sort of feedback in the system. Without feedback, the RPi will set the speed of the motor with no information regarding the actual speed of the motor. In the event that the motor runs faster or slower than directed, the RPi should be notified so that the control signal can be adjusted accordingly. To account for this, a feedback signal from the motor will pass to the RPi the actual speed of the motor. The actual speed will be subtracted from the desired speed to obtain the speed error in the system. A transfer function will be designed and used in a controller to compensate for the error. An accurate control system cannot be designed until a physical motor and system are implemented. The operation of the system will be measured to obtain an open loop plant transfer function. The general model for the control system is described in Figure 10. The respective transfer function blocks will be replaced with functions that are found through experimentation.

[JDR]

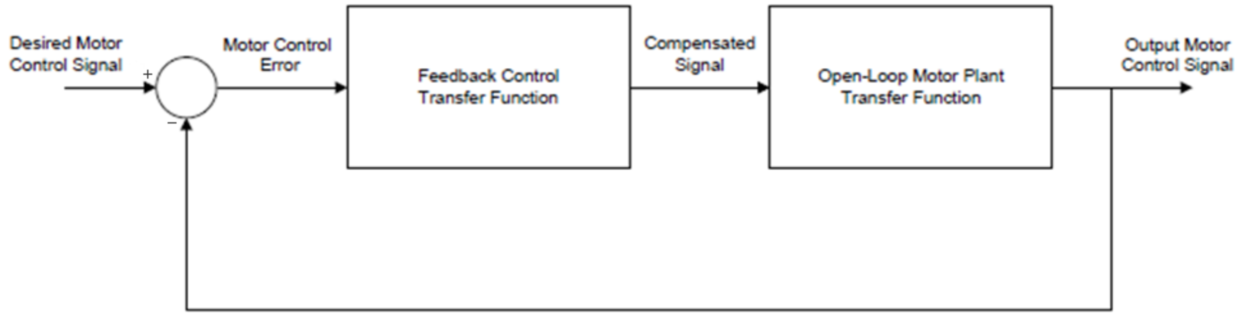

Desired Motor Control Signal
+
−
Motor Control Error
Feedback Control Transfer Function
Compensated Signal
Open-Loop Motor Plant Transfer Function
Output Motor Control Signal

*Figure 10. Unity feedback control model for motor control system.*

[JDR]

### 3.3 Power Supply

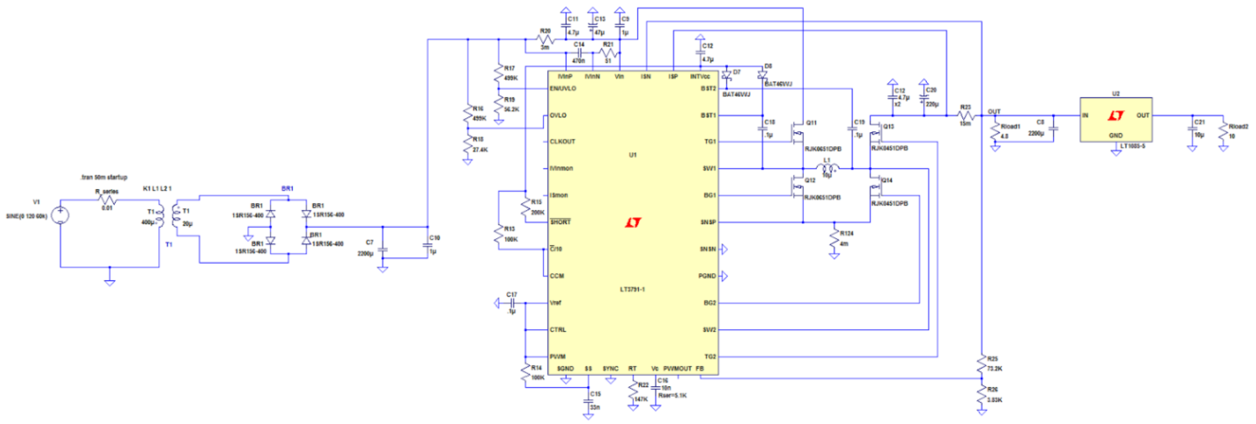


*Figure 11. The 120V/24V,5V AC/DC power supply.*

[PAG]

The circuit shown in Figure 11 is the power supply that will provide power to the modules of the system. The power supply will take as inputs the sinusoidal voltage and current provided by a wall outlet. The system will be powered by 120VAC because the system will remain stationary near a power outlet. This eliminates the need for a battery supply and other issues associated with the use of batteries. The voltage is provided in AC and as such needs to be stepped down, rectified, smoothed out, and cleared of noise. The modified input voltage is then regulated to the desired DC values needed to power the various modules of the system.

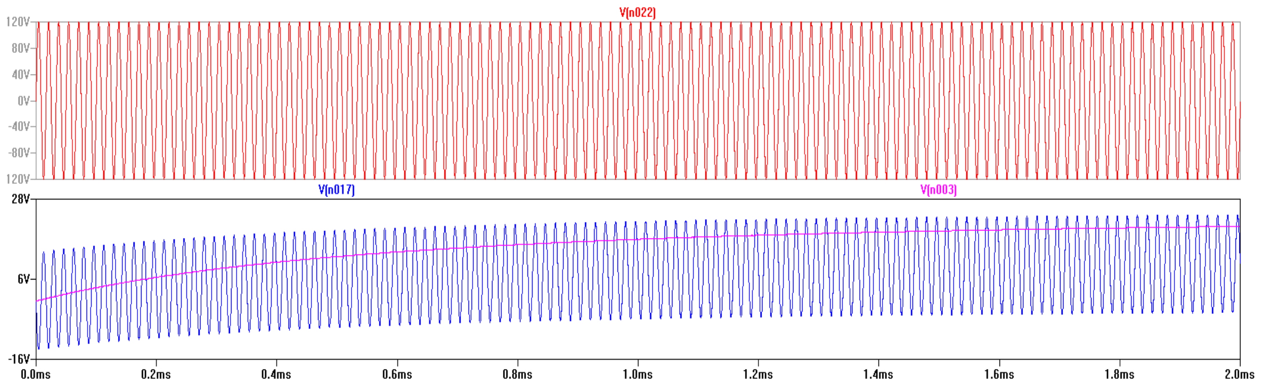


*Figure 12. The 120VAC primary voltage and the transformed 24VAC secondary voltage in which the capacitor smooths the voltage after the voltage is rectified.*

[PAG]

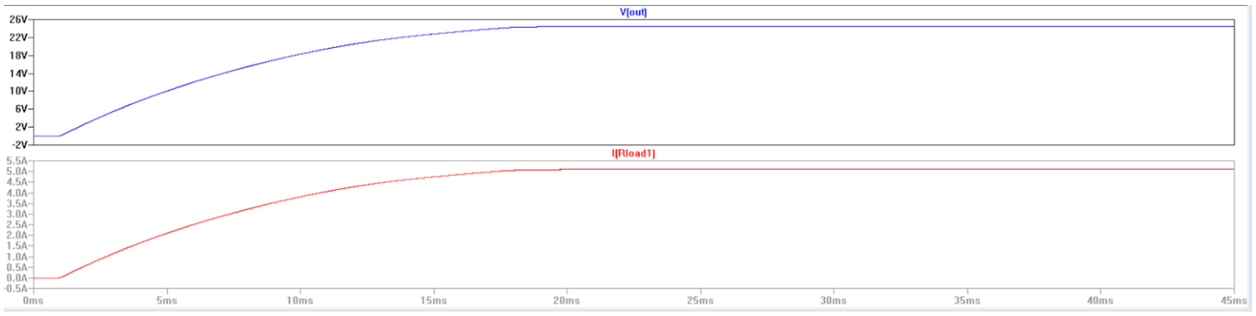


*Figure 13. Voltage and Current output from the LT3791 providing 24V 5A.*

[PAG]

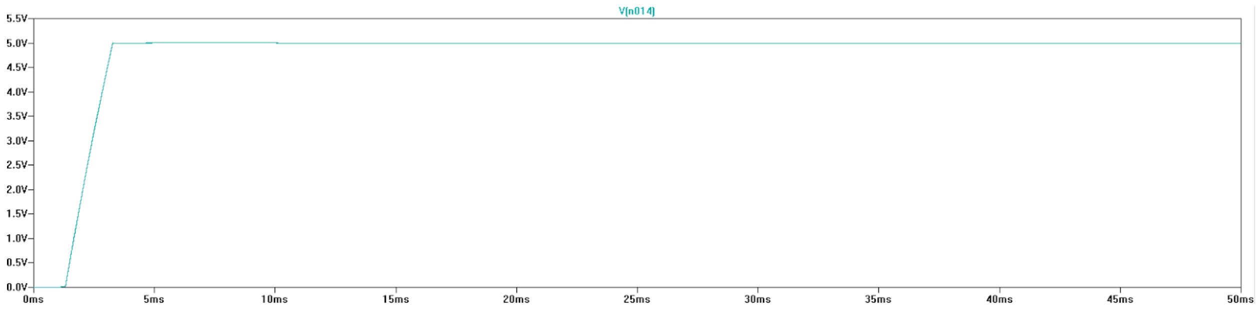


*Figure 14. Voltage output from the LT1085 providing 5V.*

[PAG]

The supply will provide two power rails, one at 24VDC and one at 5VDC. Starting with 120VAC at 60Hz, a transformer is used to obtain a secondary voltage of 24VAC. The 24V is full-wave rectified using a diode bridge. A 2200µF capacitor is used in parallel with a 2µF capacitor to smooth the full wave to 24VDC. This process is seen in the simulation on Figure 12. The LT3791 is a buck-boost voltage regulator that will take in the 24VDC. The LT3791 will use the configuration shown in Figure 11 to provide a 24V 5A output that was simulated in Figure 13. This new voltage will be used to provide power to the DC motor. This rail will also be connected to a LT1085 regulator that will drop the voltage down to 5V as shown in Figure 14.

The parts were selected to fit the voltage and current needs of the system. The LT3791 is a regulator that drives an H-bridge based on the input and the desired output. The regulator provides both constant voltage and current at up to 98.5% efficiency. This regulator is an ideal DC/DC voltage regulator that uses both step-up and step-down conversion as needed to obtain a constant voltage. The LT3791 has a wide range of input voltages and can be used to supply the great amount of power that the motor may require. The LT1085 regulator provides 5V 3A and uses the 24V from the LT3791 as an input. This 5V rail will power the RPi, the sensors, the LCD, and the push buttons.

[PAG]

## 3.4 Physical Interface

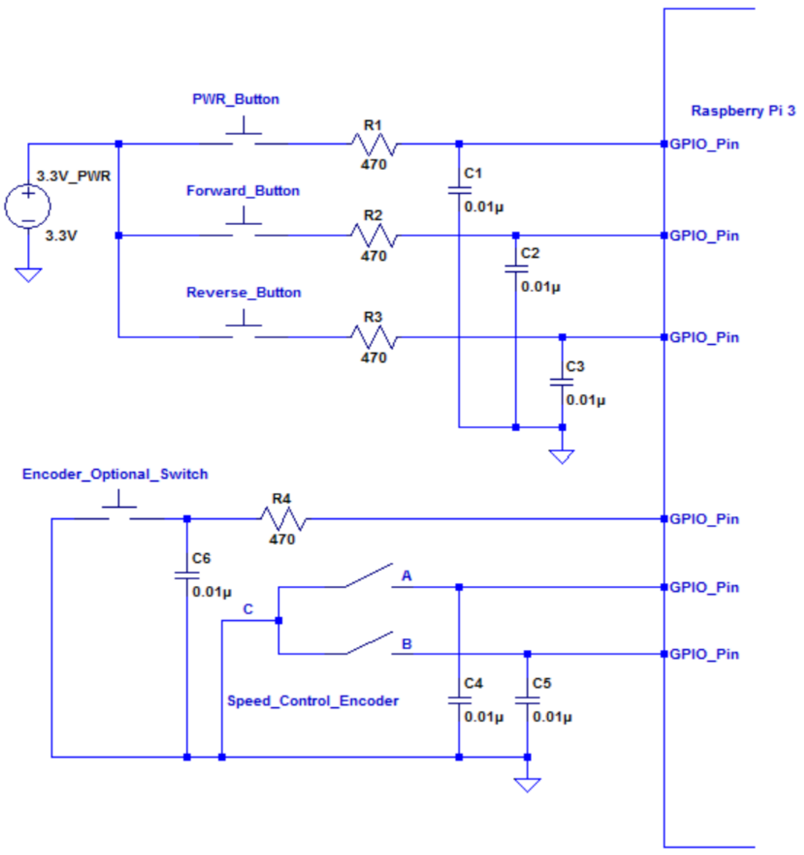


*Figure 15. Schematic for physical interface.*

[JDR]

The schematic shown in Figure 15 illustrates the physical button interface that will allow the user to interact with and operate the smart fan without the Android application. The interface contains three buttons and a rotary encoder. The first button is a power button that will simply turn the fan on or off. The second and third buttons will dictate the direction of airflow; one will be labeled for forward airflow and the other will be labeled for reverse airflow. Each button will be connected to a GPIO pin on the RPi through a 470Ω resistor. Each button will also have a capacitor connected to ground that will debounce the buttons and reduce noise on the inputs.

The rotary encoder will have three terminals: a common and two output pins that will be connected to two GPIO pins on the RPi. Within the encoder the outputs are connected to switches that open and close relative to the direction in which the encoder dial is turned. This allows the fan speed to be changed according to the number of increments and direction in which the encoder dial is turned. The encoder also has an optional push button. This is not needed for the current application of the rotary encoder, however it will be connected as shown in Figure 15 in the case that a need is found.

[JDR]

## 3.5 Sensor

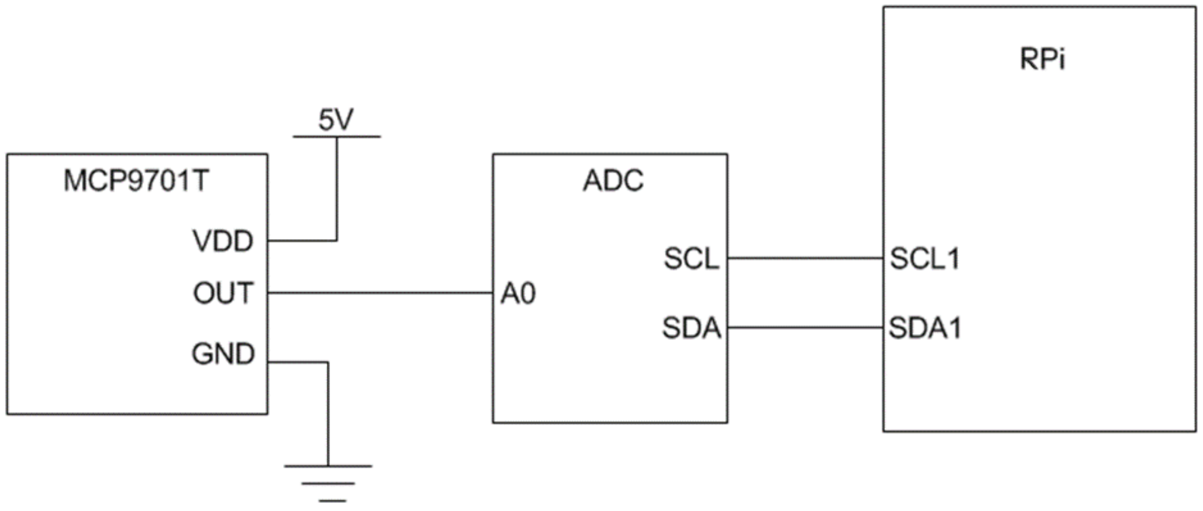


*Figure 16. Temperature sensor configuration for the RPi.*

[PAG]

The temperature sensor design is a simple configuration, as shown in Figure 16. The analog sensor has three pins: ground, input, and output. The power supply will provide 5V to the input pin. The output will be connected to the ADC, which will subsequently be connected to the RPi. The sensor can measure temperatures in the range of -10°C to 125°C, with an accuracy of +/- 1°C. The sensor exceeds the project requirements and, as such, is a simple fit in the design. This sensor information will be useful in the temperature control operation of the system.

[PAG]

## 3.6 Hardware Calculations

*Table 15. Power Calculations for AC Box Fan.*

| Fan Setting | Voltage (V) | Current (A) | Power (W) | Speed (RPM) |
|---|---|---|---|---|
| 0 | 122.22 | 0.0 | 0.0 | 0 |
| 1 | 121.72 | 0.503 | 61.2 | 730 |
| 2 | 121.52 | 0.565 | 68.4 | 950 |
| 3 | 121.52 | 0.632 | 75.1 | 1100 |

[JDR]

The power measurements shown in Table 15 were taken from a standard box fan that uses an AC induction motor. These measurements and calculations allow the understanding of the power consumption and current drawn by the motor. These measurements will be used in the smart fan design. The design calls for a DC motor whose specifications meet or exceed those found in the AC motor measurements. Torque calculations were also done using the power and the speed of the motor at each fan setting. The maximum torque calculated will be used to choose a DC motor that is rated to exceed the torque output performance of current fan models.

[JDR]

The temperature range of the average household is kept between 60 degrees and 80 degrees Fahrenheit. The human body is not sensitive enough to the point where a one degree difference from the desired temperature would be noticeable. This allows a window of tolerance in terms of the accuracy of the temperature which the fan needs to maintain when in the temperature control mode. The temperature sensor will be chosen to exceed the range necessary to cover any extreme circumstances.

[PAG]

## 3.7 Software Theory of Operation

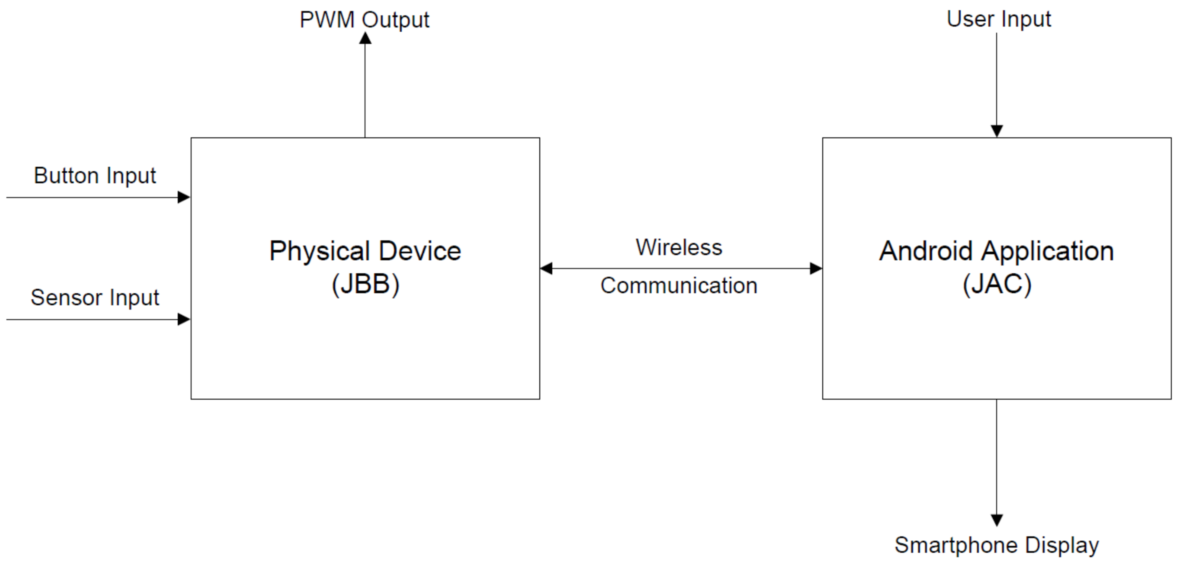


*Figure 17. Software Block Diagram Level 0.*

[JAC]

The Level 0 Software Block Diagram shown in Figure 17 displays the most basic form of the system, split between the Physical Device and the Android Application. This demonstrates the core inputs and outputs of the system. The inputs are user controls and choices as well as communication between the components. The outputs are the resulting driving signal and the feedback to the user. Each side functions relatively independent of the other, sending updates to keep the other informed of its operations. The inputs and outputs of the Level 0 Diagram are explained in Table 16 and Table 17.

*Table 16. Software Block Diagram Level 0: Physical Device.*

| *Module* | Physical Device |
|---|---|
| *Inputs* | ○ Button Input – Physical button inputs to the module.<br>○ Sensor Input – Physical sensors measure ambient temperature.<br>○ Wireless Communication – User preferences and instructions received from the Android application. |
| *Outputs* | ○ PWM Output – Driving signal sent to the motor circuitry.<br>○ Wireless Communication – Operation information such as temperature data passed to the Android application to be displayed to the user. |
| *Functionality* | Instructions from the Android application are interpreted here and used to control the operation of the smart fan. |

*Table 17. Software Block Diagram Level 0: Android Application.*

| *Module* | Android Application |
|---|---|
| *Inputs* | ○ User Input – Inputs and adjustments made by the user.<br>○ Wireless Communication – Operation information such as temperature data passed from the physical device. |
| *Outputs* | ○ Smartphone Display – Output display to interface with the user.<br>○ Wireless Communication – User preferences and instructions sent to the physical device. |
| *Functionality* | The Android Application will allow the user to change any settings and control their fan module from anywhere. They can use the application to set timers and program the fan to start or stop to allow them to have a room cooled by the time the user gets home. |

[JBB, JAC]

The Level 1 Software Block Diagram shown in Figure 18 depicts a more in-depth view of the system. While the inputs and outputs are the same as those of the Level 0 Diagram, this diagram more clearly illustrates how each block functions. The two blocks are further broken down into their subsystems and display the distribution of work within each block. The workings of the system are explained in more detail in Table 18 through Table 21.

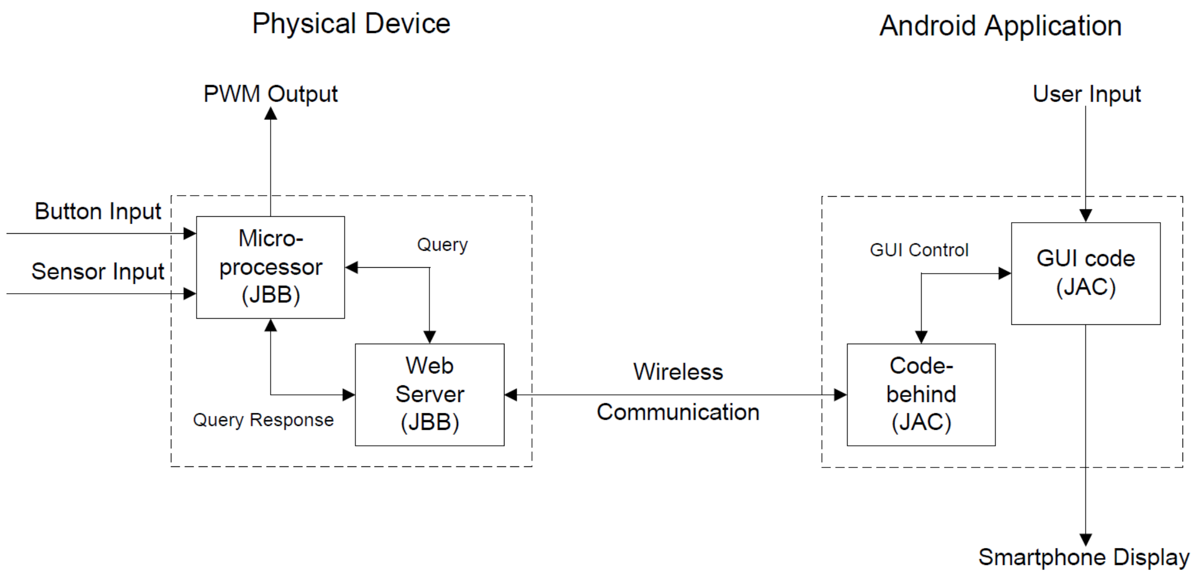


*Figure 18. Software Block Diagram Level 1.*

[JAC]

*Table 18. Software Block Diagram Level 1: Microprocessor.*

| *Module* | Microprocessor |
|---|---|
| *Inputs* | ○ Button Input – Physical button inputs to the module.<br>○ Sensor Input – Physical sensors measure ambient temperature.<br>○ Commands – Instructions given from the web server that direct the PWM. |
| *Outputs* | ○ PWM Output – Driving signal sent to the motor circuitry.<br>○ Temperature – Interpreted sensor readings passed to the web server. |
| *Functionality* | This block handles the hardware-software interpretation. |

*Table 19. Software Block Diagram Level 1: Web Server.*

| *Module* | Web Server |
|---|---|
| *Inputs* | ○ Temperature – Interpreted sensor readings passed from the microprocessor.<br>○ Wireless Communication – User preferences and instructions received from the Android application. |
| *Outputs* | ○ Commands – Instructions given to the microprocessor that direct the PWM.<br>○ Wireless Communication – Operation information such as temperature data passed to the Android application to be displayed to the user. |
| *Functionality* | Hosts stored operation parameters and bears the bulk of the computational effort done by software. |

*Table 20. Software Block Diagram Level 1: Code-behind.*

| *Module* | Code-behind |
|---|---|
| *Inputs* | ○ Wireless Communication – Operation information such as temperature data passed from the web server to be interpreted and passed to the GUI code.<br>○ GUI Control – Parameters passed in from the GUI code when calling code-behind functions. |
| *Outputs* | ○ GUI Control – Results from operations performed in the code-behind to display information to the user.<br>○ Wireless Communication – User preferences and instructions sent to the web server. |
| *Functionality* | The code-behind handles the computational effort of the Android application. Any and all functions being run at the Android application are done in code-behind. |

*Table 21. Software Block Diagram Level 1: GUI code.*

| *Module* | GUI code |
|---|---|
| *Inputs* | ○ User Input – Inputs and adjustments made by the user.<br>○ GUI Control – Results from operations performed in the code-behind to display information to the user. |
| *Outputs* | ○ GUI Control – Parameters passed to the code-behind when calling functions.<br>○ Smartphone Display – Output display to interface with the user. |
| *Functionality* | The GUI code programmatically determines the display. |

[JBB, JAC]

The Software Level 2 Block Diagram shown in Figure 19 further expands the system and conveys more about the languages and protocols being used in the system. This strays slightly from the previous diagram as the system approaches a more realistic layout and more accurately displays the final design. The components of the system are detailed in Table 22 through Table 29.

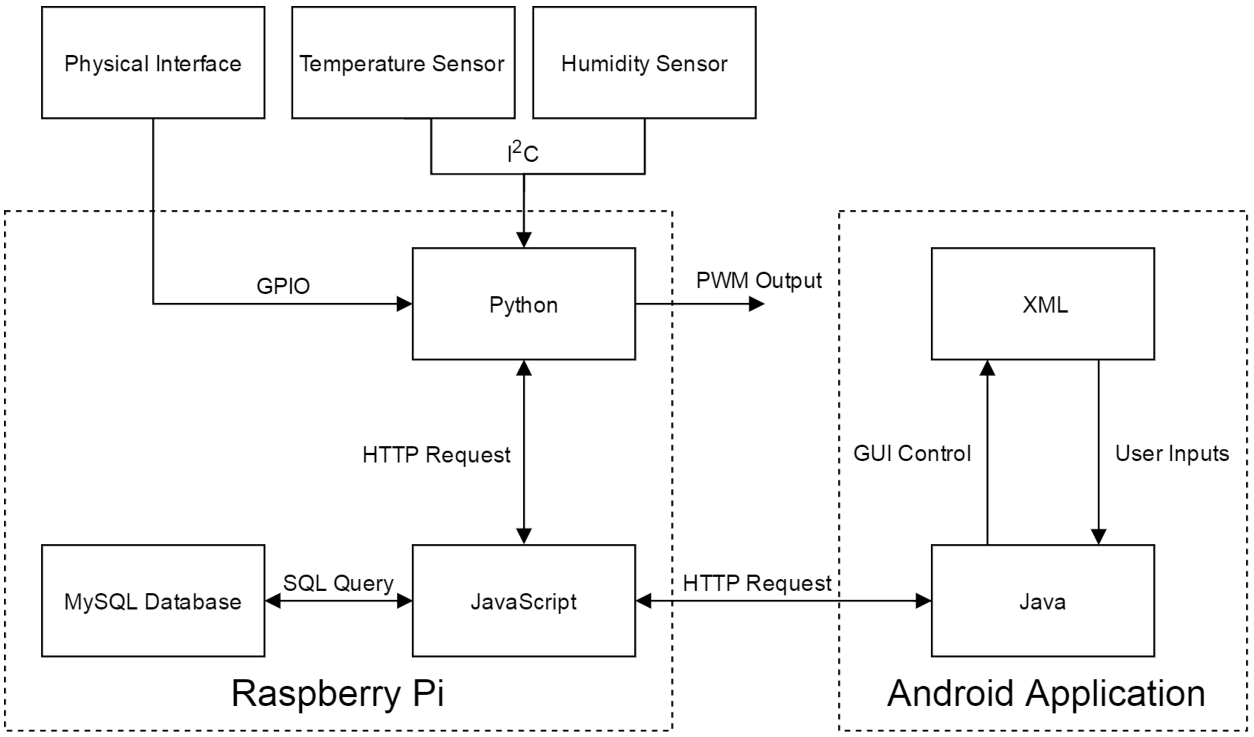


*Figure 19. Software Block Diagram Level 2.*

[JBB]

The RPi was chosen as the microprocessor. This choice directly affected the decisions made regarding other components. The RPi was chosen for several reasons: direct PWM output, integrated Bluetooth and wireless local area network (WLAN) modules, support for the Inter-Integrated Circuit ($I^2C$) protocol, numerous and easily accessible general-purpose input/output (GPIO) pins, and overall flexibility and potential for developing and running code.

Support for a direct PWM output is key to the success of this project. The PWM output will be used to drive the motor of the fan, and so direct support was considered essential when deciding on a microprocessor; all other potential benefits of microprocessors were secondary to the PWM.

The Bluetooth and WLAN modules are a core requirement for the success of this project. These modules grant the physical device access to the networks through which communication with the Android application can be attained. The modules could have been acquired and implemented separately from the microprocessor, but would have required extraneous effort to incorporate them into the system.

To communicate with the temperature and humidity sensors, two protocols were considered: the $I^2C$ bus and the Serial Peripheral Interface (SPI) bus. $I^2C$ and SPI are comparable in complexity and effectivity regarding implementation, however $I^2C$ was chosen not arbitrarily but rather because it would reduce development time and costs due to previously obtained experience and hardware.

The number and ease of use of GPIO pins was a lower priority than some other considerations made when selecting the RPi but still warranted attention. The microprocessor must have a sufficient number of GPIO pins such that the physical interface and the driver circuit can be controlled. The RPi has forty pins that are allotted to various purposes. Some of these pins are multipurpose with the ability to function as a GPIO pin, support a serial communication, or fulfill one of several other applications. Twenty-six of the forty pins on the RPi can be used as a GPIO, which is more than enough to implement the necessary functionality.

The final considerations and biggest benefits offered by the RPi are the flexibility and potential regarding the development and execution of software. The RPi has an enormous amount of computational power compared to other microprocessors and can be used to handle multiple tasks at once. Boasting a 1.2GHz 64-bit quad-core processor with a gigabyte of memory, the RPi can host both the web server and the database, run the scripts that will operate the hardware components, and sustain all computational effort involved in the project.

The computational power offered by the RPi allows for the potential to satisfy up to four of the engineering requirements listed in Table 1. The RPi will be able to handle all of the computational effort needed by the project. This allows Engineering Requirements #2 and #3 to be satisfied by reducing the necessary communication with and computation at the Android application. The RPi also has the power to perform operations quickly. This allows Engineering Requirements #5 and #6 to be satisfied by reducing the delay between receiving and realizing a command.

[JBB, JAC]

*Table 22. Software Block Diagram Level 2: Physical Interface.*

| | |
|---|---|
| *Module* | Physical Interface |
| *Inputs* | None |
| *Outputs* | ○ GPIO – The signals from the buttons and switches are sent to the RPi over the GPIO pins so the RPi can update the runtime parameters of the smart fan accordingly. |
| *Functionality* | The physical interface will consist of user controls and display that present the user with the ability to make updates to the fan’s operation without needing to use their smartphone. All components will communicate with the software through the GPIO pins of the RPi. |

*Table 23. Software Block Diagram Level 2: Temperature Sensor.*

| | |
|---|---|
| *Module* | Temperature Sensor |
| *Inputs* | None |
| *Outputs* | ○ $I^2C$ – The data obtained by the sensors is sent to the RPi through the RPi's physical pins using the $I^2C$ protocol. |
| *Functionality* | The ambient temperature sensor will allow users to set the fan to run based on the temperature in the room. The data retrieved by the sensor will be sent to the RPi using the $I^2C$ protocol. |

*Table 24. Software Block Diagram Level 2: Humidity Sensor.*

| | |
|---|---|
| *Module* | Humidity Sensor |
| *Inputs* | None |
| *Outputs* | ○ $I^2C$ – The data obtained by the sensors is sent to the RPi through the RPi's physical pins using the $I^2C$ protocol. |
| *Functionality* | The ambient humidity sensor will allow users to remotely view the humidity in the surrounding area. The data retrieved by the sensor will be sent to the RPi using the $I^2C$ protocol. |

*Table 25. Software Block Diagram Level 2: Python.*

| | |
|---|---|
| *Module* | Python |
| *Inputs* | ○ $I^2C$ – The data obtained by the sensors is sent to the RPi through the RPi's physical pins using the $I^2C$ protocol.<br>○ GPIO – The signals from the buttons and switches are sent to the RPi over the GPIO pins so the RPi can update the runtime parameters of the smart fan accordingly.<br>○ HTTP Request – These are the web server's responses to GET and POST requests generated by the Python scripts to deliver new information or to acknowledge a successful POST. |
| *Outputs* | ○ HTTP Request – These are GET and POST requests generated by the Python scripts to get or post new information.<br>○ PWM Output – This is the driving signal generated by the RPi to operate the DC motor in accordance with the parameters specified by the user. |
| *Functionality* | The Python scripts running on the RPi will handle the low-level operations of the system, such as setting the PWM value and facilitating communication with the hardware components. |

HTTP requests were chosen to facilitate communication between the Android application and the web server because the data being transmitted should be minimal and infrequent. HTTP requests are asynchronous and as such can greatly reduce the amount of data being transferred between the two systems. This reduction promotes satisfaction of Engineering Requirement #2 as listed in Table 1.

*Table 26. Software Block Diagram Level 2: JavaScript.*

| *Module* | JavaScript |
|---|---|
| *Inputs* | ○ SQL Query – Data retrieved from a select query.<br>○ HTTP Request – These are GET and POST requests generated by the Android application to get or post new information.<br>○ HTTP Request – These are GET and POST requests generated by the Python scripts to get or post new information. |
| *Outputs* | ○ SQL Query – Query to store or select data to or from the MySQL database.<br>○ HTTP Request – These are the web server's responses to GET and POST requests generated by the Android application to deliver new information or to acknowledge the successful POST.<br>○ HTTP Request – These are the web server's responses to GET and POST requests generated by the Python scripts to deliver new information or to acknowledge the successful POST. |
| *Functionality* | The JavaScript running on the RPi will manage the software operations of the system, such as HTTP requests and database queries. This will be handled in the form of a web server implemented in the Node.js engine. |

The JavaScript module is a web server built using Node.js, a JavaScript framework designed to support highly customizable web servers. Node.js makes it possible to design a web server that only requires the most essential communication. For this project, the communications will be asynchronous and contain very minimal data so Node.js was determined to be the best solution available.

*Table 27. Software Block Diagram Level 2: MySQL Database.*

| *Module* | MySQL Database |
|---|---|
| *Inputs* | ○ SQL Query – Query to store or select data to or from the MySQL database. |
| *Outputs* | ○ SQL Query – Data retrieved from a select query. |
| *Functionality* | The MySQL database will store information about the fan's operation. The majority of the software will run using locally stored variables, but all settings will be backed up to the MySQL database after completion to preserve them during loss of power or software crashes. |

All variations of Structured Query Language (SQL) are comparable and can accomplish the same results. MySQL was chosen because it is a commonly-used and open-source version that has a large number of resources available for use.

Table 28. Software Block Diagram Level 2: Java.

| *Module* | Java |
|---|---|
| *Inputs* | ○ HTTP Request – These are the web server’s response to GET and POST requests generated by the Android application to get or post new information.<br>○ User Inputs – User interactions with XML-defined items that trigger events. |
| *Outputs* | ○ HTTP Request – These are GET and POST requests generated by the Android application to get or post new information.<br>○ GUI Control – Any updates to visual components from the Java. |
| *Functionality* | The Java code implements the actions performed by the application. The system is event-driven, meaning that event listeners are created to call functions after an associated event occurs. These events can range from a button being pressed to receiving an HTTP request. |

Table 29. Software Block Diagram Level 2: XML.

| *Module* | XML |
|---|---|
| *Inputs* | ○ GUI Control – Any updates to visual components from the Java. |
| *Outputs* | ○ User Inputs – User interactions with XML-defined items that trigger events. |
| *Functionality* | XML in Android defines the look and feel of the application. Included are styles, objects such as buttons and text fields, and other visual components. All related functionality is implemented in the Java; XML simply dictates what the user sees. |

Java and Extensible Markup Language (XML) are the two standard languages used in Android application development. There are other alternatives but a greater number of resources and tools are available for development in Java and XML. Google, the proprietor of Android, has released a free integrated development environment (IDE) called Android Studio that allows for quick and intuitive development of Android applications. The smartphone application for this project will be fully developed using this IDE.

[JAC]

## 3.8 Microprocessor Flowchart

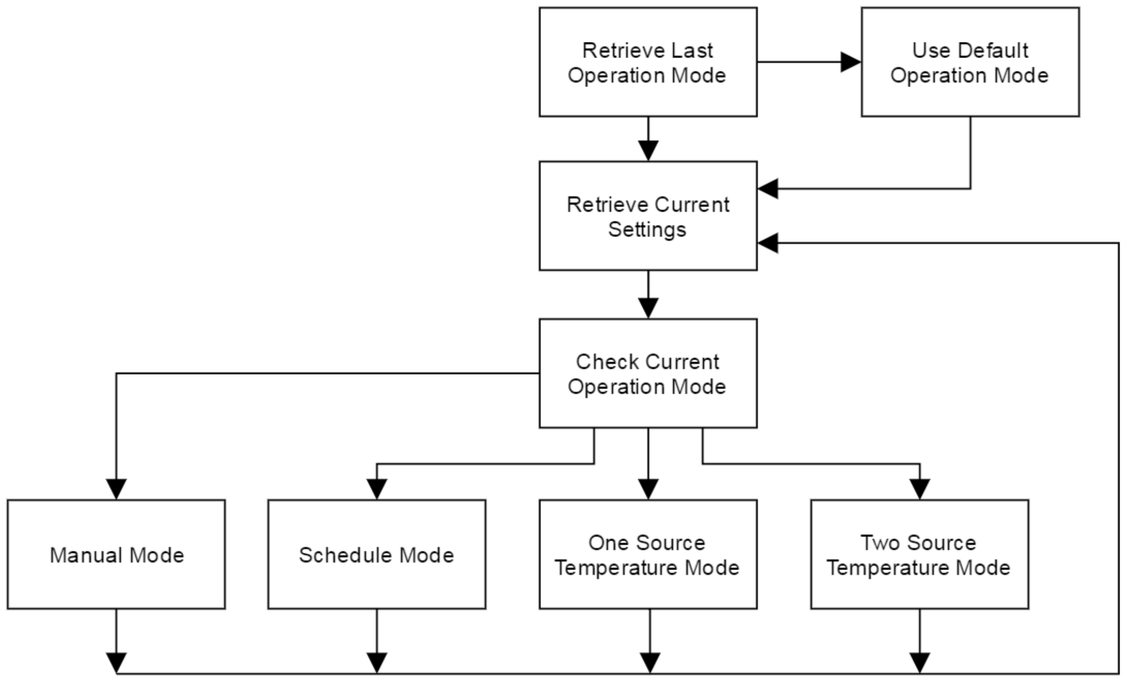

Retrieve Last Operation Mode
Use Default Operation Mode
Retrieve Current Settings
Check Current Operation Mode
Manual Mode
Schedule Mode
One Source Temperature Mode
Two Source Temperature Mode

*Figure 20. Flowchart for the Microprocessor Software.*

[JBB]

The order of the code operation for the microprocessor is shown in Figure 20. Upon being powered up the microprocessor will check for a previously stored operation mode. If one exists the mode of operation is determined, otherwise the default mode is used. The process will then enter an endless loop that, after a period of time, checks the operation mode again and runs a function corresponding to which operation mode is currently running. The current PWM frequency is passed to the selected function, using the reserved value of -1 to denote a disabled PWM. The four possible functions (manual, scheduled, one-source temperature-sensing, and two-source temperature-sensing) will retrieve relevant data by querying the database and then decide if the PWM value should be updated or should be maintained. When any of these functions finish, the current PWM value is passed to the main loop and the loop begins again. Handling these computations on the microprocessor instead of the Android application supports Engineering Requirement #3 shown in Table 1, because the computational intensity performed by the smartphone will be greatly reduced. The functions required for the operation of the main flowchart is shown in Table 30 through Table 32.

[JBB]

## 3.9 Microprocessor Application Functions

*Table 30. Function table for the MainLoop function of the Microprocessor Software.*

| | |
|---|---|
| *Function* | MainLoop |
| *Parameters* | None |
| *Returns* | None |
| *Description* | MainLoop retrieves the previously used operation mode from the web server. If one is unavailable a default operation mode will be used. After obtaining the operation mode, the function will attempt to retrieve all necessary settings from the web server. MainLoop will call the corresponding function based on the operation mode and pass in the settings and the current PWM value. If the PWM value is not set, the passed value will be -1. |
| *Called by* | Automatically called when the RPi is started. |

The FanData class shown below will be used to hold and transfer all settings required for operating the fan. Below is a snippet of the class framework. Additional variables will be necessary in the final implementation.

```
class FanData(object):
        def __init__(self,fanSpeed,roomTemp):
                self.fanSpeed = fanSpeed
                self.roomTemp = roomTemp
```

*Table 31. Function table for the GetCurrentFanData function of the Microprocessor Software.*

| | |
|---|---|
| *Function* | GetCurrentFanData |
| *Parameters* | None |
| *Returns* | Current Settings JSON object |
| *Description* | GetCurrentFanData retrieves the current settings from the web server. The settings are returned as a JSON object that is interpreted and stored into a python FanClass object. |
| *Called by* | MainLoop |

```
def GetCurrentFanData():
        r = requests.get(url + '/CurrentFanData')
        json_test = json.loads(r.text)
        fanData = FanClass.FanData(int(json_test['Class']['Fan_Speed'])
                        ,int(json_test['Class']['Room_Temp']))
        return fanData
```

*Table 32. Function table for the PostUpdateFanData function of the Microprocessor Software.*

| | |
|---|---|
| *Function* | PostUpdateFanData |
| *Parameters* | Current Settings |
| *Returns* | None |
| *Description* | PostUpdateFanData accepts the current settings as a python FanClass object and converts them to a JSON object that is sent to the web server. |
| *Called by* | ManualMode, ScheduleMode, One- & TwoSourceTemperatureMode |

```
def PostUpdateFanData(fanData):
        data = {}
        data["Room_Temp"] = fanData.roomTemp
        data["Fan_Speed"] = fanData.fanSpeed
        json_data = json.dumps(data)

        r = requests.post(url + '/UpdateFanData', data)
        print r.text
```

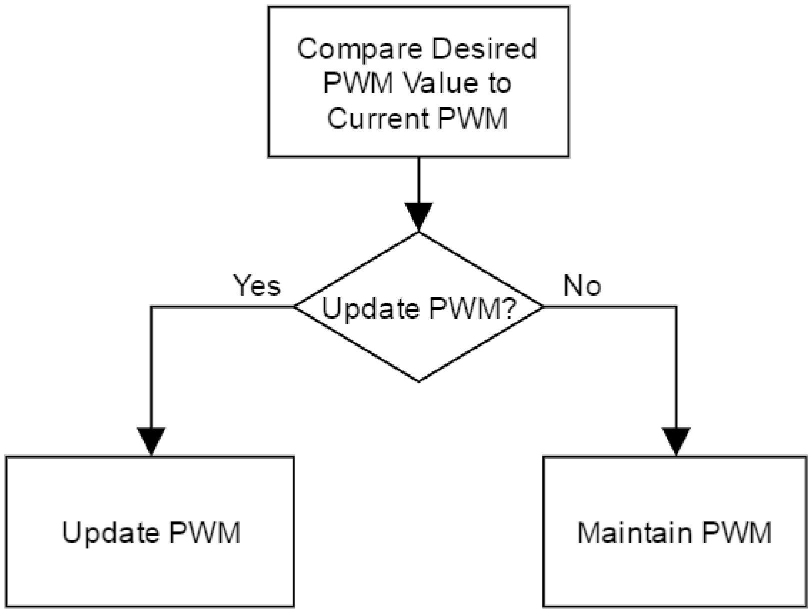


*Figure 21. Flowchart for the Manual Operation Mode.*

[JBB]

The manual operation mode is shown in Figure 21. This mode will check the desired value of the PWM provided by the user and compare that against the current PWM value. If these two values are different then the PWM value will be updated. The functions required for the operation of the manual operation flowchart is shown in Table 33 through Table 36.

*Table 33. Function table for the ManualMode function of the Microprocessor Software.*

| | |
|---|---|
| *Function* | ManualMode |
| *Parameters* | Current Settings, Current PWM Value |
| *Returns* | New PWM Value |
| *Description* | ManualMode determines the appropriate value to set the PWM, based on the given settings. ManualMode will only take into account the desired speed value provided by the user and the current PWM value. If the PWM value is the same as the current value, nothing will be changed to facilitate a continuous operation of the motor. All of the functions will handle PWM change in this way. If the PWM value is not the same ManualMode will adjust the PWM value towards the desired value. ManualMode will only adjust the PWM by a set amount in order to protect the motor. If this step does not reach the final value, ManualMode will run again in the next iteration of MainLoop. |
| *Called by* | MainLoop |

Using the specified PWM declaration shown in Table 34 supports Engineering Requirement #7 shown in Table 1. The hardware PWM of the RPi has a frequency range of 1Hz to 19MHz while the software PWM of the RPi has a frequency of only 100Hz. Therefore, in order to satisfy this engineering requirement it was necessary to choose the hardware PWM rather than the software PWM. The functions shown below display the utilization of the required PWM.

*Table 34. Function table for the SetupPWM function of the Microprocessor Software.*

| | |
|---|---|
| *Function* | SetupPWM |
| *Parameters* | None |
| *Returns* | None |
| *Description* | SetupPWM configures the naming convention for the RPi pins and specifies the type of PWM output (either hardware or software). |
| *Called by* | MainLoop |

```
def setuppwm():
        wiringpi.wiringPiSetup()
```

*Table 35. Function table for the SetPWM function of the Microprocessor Software.*

| | |
|---|---|
| *Function* | SetPWM |
| *Parameters* | PWM Value |
| *Returns* | Success – boolean value |
| *Description* | SetPWM sets the PWM pin of the RPi to the desired value. If this operation succeeds SetPWM returns true, otherwise SetPWM returns false. |
| *Called by* | ManualMode, ScheduleMode, One- & TwoSourceTemperatureMode |

```
def setpwm(pwm):
    OUTPUT = 2
    TURN_OFF = 0
    PIN_TO_PWM = 1 # GPIO 18 on the RPI

    if pwm == -1:
        wiringpi.pinMode(PIN_TO_PWM, TURN_OFF)
        result = True

    elif pwm >= 0 and pwm <= 47:
        wiringpi.pinMode(PIN_TO_PWM, OUTPUT)
        configurepwm()
        wiringpi.pwmWrite(PIN_TO_PWM, pwm)
        result = True

    else:
        result = False

    return result
```

*Table 36. Function table for the ConfigurePWM function of the Microprocessor Software.*

| | |
|---|---|
| *Function* | ConfigurePWM |
| *Parameters* | None |
| *Returns* | None |
| *Description* | ConfigurePWM sets the required settings for the PWM pin to have a frequency of 40kHz and to have 48 discrete steps. |
| *Called by* | ManualMode, ScheduleMode, One- & TwoSourceTemperatureMode |

```
def configurepwm():
        wiringpi.pwmSetMode(wiringpi.PWM_MODE_MS)
        wiringpi.pwmSetClock(10)
        wiringpi.pwmSetRange(48)
```

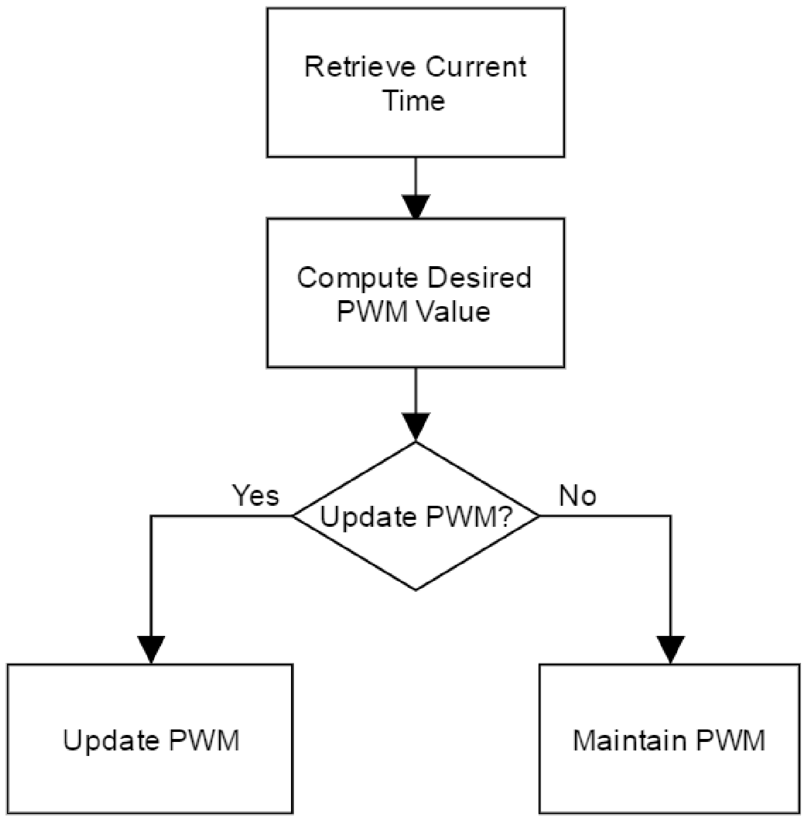


*Figure 22. Flowchart for the Schedule Operation Mode.*

[JBB]

The schedule operation mode is shown in Figure 22. This mode will check if the current time is within the desired operation time periods. If it is then the PWM value will be updated to the desired PWM value if it is different than the current PWM value. The functions required for the operation of the schedule operation flowchart is shown in Table 37.

*Table 37. Function table for the ScheduleMode function of the Microprocessor Software.*

| | |
|---|---|
| *Function* | ScheduleMode |
| *Parameters* | Current Settings, Current PWM Value |
| *Returns* | New PWM Value |
| *Description* | ScheduleMode behaves similarly to ManualMode in function, however, ScheduleMode allows the user to also provide time ranges to which speed values can be mapped. |
| *Called by* | MainLoop |

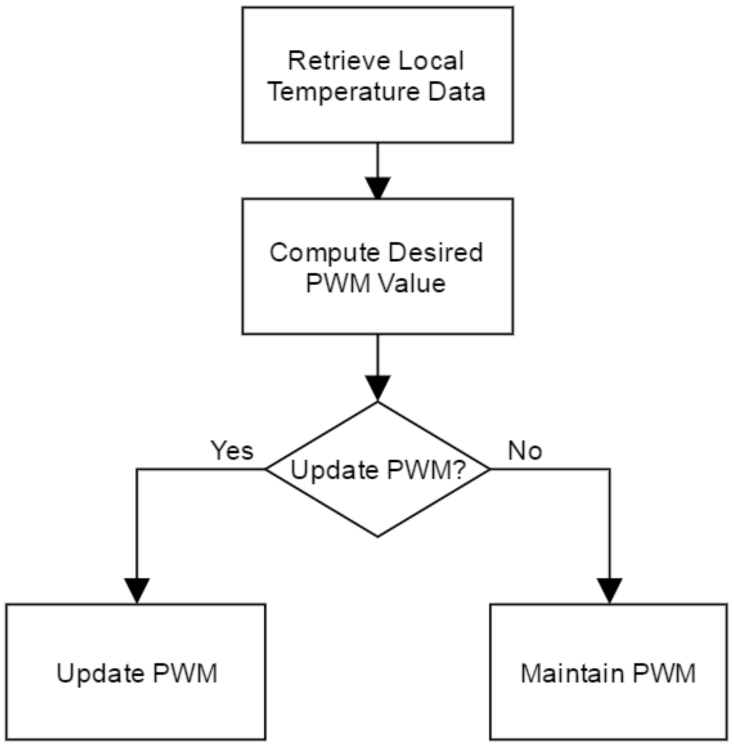


*Figure 23. Flowchart for the One Source Temperature Mode.*

[JBB]

The one source temperature operation mode is shown in Figure 23. This mode will check the ambient temperature by accessing the temperature sensor. Based on the ambient temperature value and settings set by the user the microprocessor will calculate the desired PWM value. If the desired PWM value is different than the current PWM value the PWM will be updated. The functions required for the operation of the schedule operation flowchart is shown in **Error! Reference source not found.** through **Error! Reference source not found.**.

*Table 38. Function table for the OneSourceTemperatureMode function of the Microprocessor Software.*

| | |
|---|---|
| *Function* | OneSourceTemperatureMode |
| *Parameters* | Current Settings, Current PWM Value |
| *Returns* | New PWM Value |
| *Description* | OneSourceTemperatureMode adjusts the speed of the fan based on the temperature value of the temperature sensor and the given settings. The user will set a low and a high temperature threshold. With these thresholds the user will also provide a desired fan speed at the given threshold. The speed of the fan will then be adjusted to a curve between these two values as the temperature fluctuates between the two given temperatures. |
| *Called by* | MainLoop |

*Table 39. Function table for the ReadTemp function of the Microprocessor Software.*

| | |
|---|---|
| *Function* | ReadTemp |
| *Parameters* | None |
| *Returns* | Temperature Value |
| *Description* | ReadTemp accesses channel 0 of the ADC--the channel connected to the temperature sensor--to retrieve the current temperature. This temperature value is transmitted to the RPi using $I^2C$. |
| *Called by* | OneSourceTemperatureMode, TwoSourceTemperatureMode |

```
def readTemp()
        adc = Adafruit_ADS1x15.ADS1015()
        GAIN = 1

value = adc.read_adc(0, gain=GAIN)

        # Apply corrective function to attain temp value

        return temp
```

*Table 40. Function table for the ReadHumidity function of the Microprocessor Software.*

| | |
|---|---|
| *Function* | ReadHumidity |
| *Parameters* | None |
| *Returns* | Humidity Value |
| *Description* | ReadHumidity accesses channel 1 of the ADC, the channel connected to the humidity sensor, to retrieve the current humidity. This humidity value is transmitted to the RPi using $I^2C$. |
| *Called by* | OneSourceTemperatureMode, TwoSourceTemperatureMode |

```
def readHumidity()
        adc = Adafruit_ADS1x15.ADS1015()
        GAIN = 1

        value = adc.read_adc(1, gain=GAIN)

        # Apply corrective function to attain humidity value

        return humidity
```

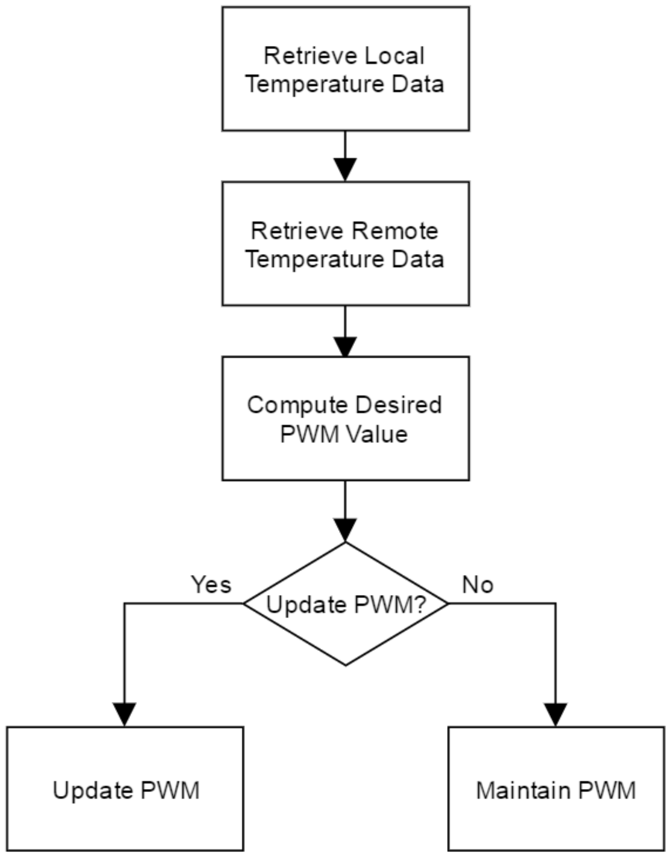

Retrieve Local Temperature Data
Retrieve Remote Temperature Data
Compute Desired PWM Value
Update PWM?
Yes
No
Update PWM
Maintain PWM

*Figure 24. Flowchart for the Two Source Temperature Operation Mode.*

[JBB]

The two source temperature operation mode is shown in Figure 24. This mode will check the ambient temperature by accessing the temperature sensor and will check the local temperature by accessing a weather API. Based on the differential of the ambient temperature and local temperature and settings set by the user the microprocessor will calculate the desired PWM value. If the desired PWM value is different than the current PWM value the PWM will be updated. The functions required for the operation of the schedule operation flowchart is shown in Table 41 through Table 44.

*Table 41. Function table for the TwoSourceTemperatureMode function of the Microprocessor Software.*

| | |
|---|---|
| *Function* | TwoSourceTemperatureMode |
| *Parameters* | Current Settings, Current PWM Value |
| *Returns* | New PWM Value |
| *Description* | TwoSourceTemperatureMode will behave similarly to how OneSourceTemperatureMode behaves, except with an additional temperature input from a weather API. This operation mode is intended for operation in a window. To facilitate this the weather API will provide the outside temperature based on location data and the temperature sensor will provide the inside temperature. The speed and direction of the fan will be decided based on the temperature difference between the two sources. |
| *Called by* | MainLoop |

*Table 42. Function table for the WeatherCurrentAddress function of the Microprocessor Software.*

| | |
|---|---|
| *Function* | WeatherCurrentAddress |
| *Parameters* | Address |
| *Returns* | Weather Information |
| *Description* | WeatherCurrentAddress passes an address to LatitudeLongitude and uses the resultant data to call a weather API to retrieve the local weather information. |
| *Called by* | TwoSourceTemperatureMode |

```
def CurrentAddress(address):
        location = LocationRequest.LatLng(address)

        weather = Current(location.lat, location.lng)
        return weather
```

*Table 43. Function table for the WeatherCurrent function of the Microprocessor Software.*

| | |
|---|---|
| *Function* | WeatherCurrent |
| *Parameters* | Latitude and Longitude Coordinates |
| *Returns* | Weather Information |
| *Description* | WeatherCurrent uses latitude and longitude coordinates to access a weather API. The API returns the weather information as a JSON object. |
| *Called by* | WeatherCurrentAddress |

```
def Current(lat = 41.07472, lng = -81.52201):
        query_url = api_url % (api_key, int(float(lat)), int(float(lng)))
        r = requests.get(query_url)

        if r.status_code != 200:
                print "Error:", r.status_code
        else:
                json_weather = r.json()
                return json_weather
```

*Table 44. Function table for the LatitudeLongitude function of the Microprocessor Software.*

| | |
|---|---|
| *Function* | LatitudeLongitude |
| *Parameters* | Address |
| *Returns* | Latitude and Longitude Coordinates |
| *Description* | LatitudeLongitude uses an address to access a Google API that returns the corresponding latitude and longitude for the address. |
| *Called by* | WeatherCurrentAddress |

```
def LatLng(address):
        address = address.replace(" ","+")
        query_url = api_url + address + api_key

        r = requests.get(query_url)

        if r.status_code != 200:
                print "Error:", r.status_code
        else:
                json_weather = r.json()
                lat =  json_weather['results'][0]['geometry']['bounds']['northeast']['lat']
                lng = json_weather['results'][0]['geometry']['bounds']['northeast']['lng']

                return ReturnValue(lat, lng)
```

[JBB]

## 3.10 Web Server Flowchart

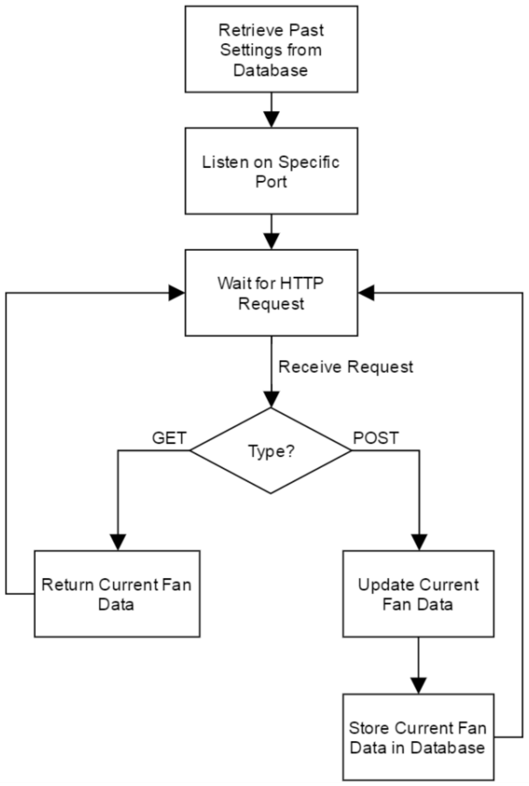


*Figure 25. Flowchart for the Web Server Software.*

[JBB]

The establishment of the web server and the continued operation to facilitate the remote access of the web server is shown in Figure 25. Upon startup the web server attempts to retrieve past settings from the database. The web server will then listen for HTTP requests on a previously specified port. Using HTTP requests supports Engineering Requirement #2 shown in Table 1, because HTTP requests can be handled asynchronously and conservatively. By reducing the total number of HTTP requests required in a given day the amount of data usage can be limited to under 30MB a month. These HTTP requests are handled based on their type. If a GET request is received the web server will return the current fan data and if a POST request is received the web server will set the current fan data to the newly received data and update the database. The functions required for the operation of the main flowchart is shown in Table 45 through Table 47.

[JBB]

## 3.11 Web Server Application Functions

*Table 45. Function table for the Port_Listen function of the Web Server Software.*

| | |
|---|---|
| *Function* | Port_Listen |
| *Parameters* | None |
| *Returns* | None |
| *Description* | Port_Listen sets the web server to listen for HTTP Requests on a specific port number. |
| *Called by* | Automatically called when the Web Server is started |

```
App.listen(port, function(err) {
        if (err) {
                return console.log('something bad happened', err)
        }
        console.log('Listening on port: ' + port)
})
```

*Table 46. Function table for the Get_CurrentFanData function of the Web Server Software.*

| | |
|---|---|
| *Function* | Get_CurrentFanData |
| *Parameters* | None |
| *Returns* | Current Settings |
| *Description* | Get_CurrentFanData returns the current settings for the fan when a request is received. |
| *Called by* | HTTP Request |

```
app.get('/CurrentFanData', function(request, response) {
        response.writeHead(200, {"Content-Type": "application/json"});
        var json = JSON.stringify({
                Class:data
        });
        response.end(json);
})
```

*Table 47. Function table for the Post_UpdateFanData function of the Web Server Software.*

| | |
|---|---|
| *Function* | Post_UpdateFanData |
| *Parameters* | Settings Update |
| *Returns* | None |
| *Description* | Post_UpdateFanData accepts a JSON object containing updated settings information and it interprets it. The new settings then replace the old as the current settings. |
| *Called by* | HTTP Request |

```
app.post('/UpdateFanData', function(request, response) {
        var Room_Temp = request.body.Room_Temp;
        var Fan_Speed = request.body.Fan_Speed;

        data = new fanData(Room_Temp, Fan_Speed);
        response.send('Success');
});
```

[JBB]

## 3.12 Android Application Flowchart

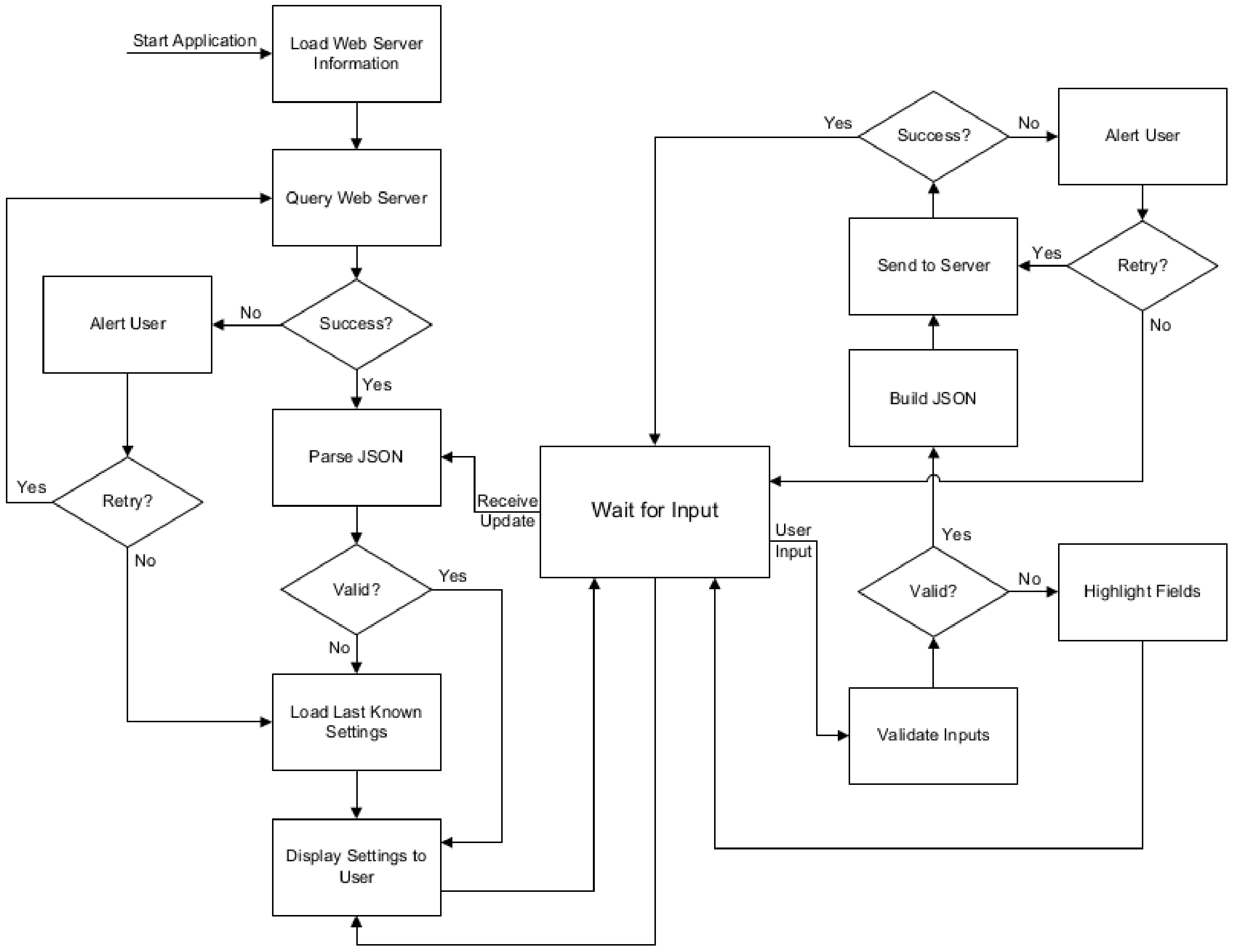


*Figure 26. Flowchart for the Android Application.*

[JAC]

The Flowchart for the Android Application in Figure 26 shows the order of code operation of the Android application. Upon being brought to focus the application will load the information necessary to communicate with the web server hosted at the RPi. The application will then attempt to query the web server using these settings. If the connection is unsuccessful, the application will alert the user and prompt them to retry the connection. On a successful connection the application will attempt to parse the response from the web server. If no settings are found the application will load the last known settings from the latest instance of the application. These settings are stored locally at the Android device. After loading all relevant information and displaying them on the GUI, the application will enter an idle state where it will wait for user input provided through the GUI or for an update from the server. If the user provides a GUI input to make changes the application will send any updated values to the server. Upon receiving a server update the application will revise its local settings with the new information.

[JAC]

## 3.13 Android Application Functions

*Table 48. Function table for the LoadWebServerInformation function of the Android application software.*

| | |
|---|---|
| *Function* | LoadWebServerInformation |
| *Parameters* | None |
| *Returns* | None |
| *Description* | LoadWebServerInformation pulls locally stored data necessary for communication with the web server hosted at the RPi. The data it retrieves will include the IP address and any parameters necessary to generate an HTTP GET request to query the RPi. This data is used by the Query Web Server function which makes the actual call. |
| *Called by* | Automatically called when the application is started or brought to focus |

LoadWebServerInformation in Table 48 also hosts the initial operations of the application, which includes calling other functions to populate fields and instantiate the GUI.

*Table 49. Function table for the QueryWebServer function of the Android application software.*

| | |
|---|---|
| *Function* | QueryWebServer |
| *Parameters* | Web Server Information |
| *Returns* | JSON Object |
| *Description* | QueryWebServer handles the acquisition of data from the web server when requested by the Android application. It uses the connection information passed as parameters to generate an HTTP GET request. This request is sent to the RPi in an attempt to contact the server and receive a response. If the operation is unsuccessful a custom, reserved JSON object will be returned to signify the failure. Otherwise, the received JSON object is returned. |
| *Called by* | LoadWebServerInformation |

QueryWebServer is used as specified in Table 49 in the initial startup of the application to establish contact with the web server.

*Table 50. Function table for the AlertUser function of the Android application software.*

| | |
|---|---|
| *Function* | AlertUser |
| *Parameters* | Error Information |
| *Returns* | Retry – boolean value |
| *Description* | AlertUser is called when an attempt to contact the web server fails. A pop-up alert is generated and displayed to the user. This message informs them that the operation was unsuccessful, gives them information about the error, and gives them the option to either close the alert or retry the connection. Choosing to retry returns true and closing the alert returns false. |
| *Called by* | LoadWebServerInformation, UserInput |

AlertUser is used to display information to the user. Its functionality is detailed in Table 50.

*Table 51. Function table for the ParseJSON function of the Android application software.*

| | |
|---|---|
| *Function* | ParseJSON |
| *Parameters* | JSON object |
| *Returns* | Raw parameters – list of pairs |
| *Description* | ParseJSON is passed a JSON object containing information received in a message from the web server. The JSON will contain a set of associated keys and values. These keys and values are separated out into individual pairs and returned as a list of pair objects. |
| *Called by* | LoadWebServerInformation, ReceiveUpdate |

ParseJSON obtains the information contained in messages from the web server as described in Table 51. Its complement, BuildJSON, packages information into a JSON string to send to the web server, as shown in Table 58.

*Table 52. Function table for the LoadLastKnownSettings function of the Android application software.*

| | |
|---|---|
| *Function* | LoadLastKnownSettings |
| *Parameters* | None |
| *Returns* | Raw parameters – list of pairs |
| *Description* | LoadLastKnownSettings obtains from local storage the last known settings saved during the most recent update. These values will be stored as a set of pair objects containing keys and values for various parameters and settings determined by the user. These objects are collected and returned as a list of pairs. |
| *Called by* | LoadWebServerInformation, ReceiveUpdate |

LoadLastKnownSettings in Table 52 is called when the application is unable to retrieve a valid response from the web server, either because the returned information is invalid or because the connection attempt was unsuccessful and the user selected to not retry. In the case of an unsuccessful connection, this function serves to load the GUI with information so the user does not see empty fields. It does not resolve the connection issues and does not prevent the user from interacting with the application. Each attempt to access the web server is independent and the application can still be used offline.

*Table 53. Function table for the DisplaySettingsToUser function of the Android application software.*

| | |
|---|---|
| *Function* | DisplaySettingsToUser |
| *Parameters* | Raw parameters – list of pairs |
| *Returns* | None |
| *Description* | DisplaySettingsToUser uses the key and value stored in each pair to populate a field on the GUI. |
| *Called by* | LoadWebServerInformation, ReceiveUpdate |

DisplaySettingsToUser in Table 53 takes the information from an update and loads it into the GUI.

*Table 54. Function table for the ReceiveUpdate function of the Android application software.*

| | |
|---|---|
| *Function* | ReceiveUpdate |
| *Parameters* | JSON Object |
| *Returns* | None |
| *Description* | ReceiveUpdate is called when the application receives an HTTP POST request from the web server at the RPi. This will occur when changes are made using the physical interface and the Android application is active. These updates allow the application to stay synchronized with the web server. |
| *Called by* | Automatically called when an update from the web server is received |

ReceiveUpdate is a shell for the operations being handled after receiving an update and only facilitates the use of other functions as detailed in Table 54.

*Table 55. Function table for the UserInput function of the Android application software.*

| | |
|---|---|
| *Function* | UserInput |
| *Parameters* | None |
| *Returns* | None |
| *Description* | UserInput handles the user's attempt to effect their changes by pressing the Send button. |
| *Called by* | Send button |

UserInput is a shell for the operations being handled after the user attempts to save a settings update and only facilitates the use of other functions as detailed in Table 55.

*Table 56. Function table for the ValidateInputs function of the Android application software.*

| | |
|---|---|
| *Function* | ValidateInputs |
| *Parameters* | None |
| *Returns* | Valid – boolean value<br>(Optional) Invalid Fields – list of GUI objects |
| *Description* | ValidateInputs checks whether or not the values entered by the user are valid. Invalid inputs include empty fields, characters that are not alphanumeric, and values that are outside the acceptable ranges for inputs.<br><br>If the entries are invalid, the erroneous fields are highlighted and the function returns false. Otherwise the function returns true.<br><br>The Invalid Fields object is only returned when one or more inputs have been evaluated to be invalid and contains a list of the offending fields to be highlighted. |
| *Called by* | UserInput |

As shown in
Table 56, ValidateInputs ensures that the user enters only information that can be used properly. ValidateInputs is used in tandem with HighlightFields to inform the users of errors, as shown in Table 57.

*Table 57. Function table for the HighlightFields function of the Android application software.*

| | |
|---|---|
| *Function* | HighlightFields |
| *Parameters* | Invalid Fields |
| *Returns* | None |
| *Description* | HighlightFields iterates through the list of invalid fields passed in and highlights each item in red. The function may also alert the user with a banner or by updating a text field. |
| *Called by* | UserInput |

*Table 58. Function table for the BuildJSON function of the Android application software.*

| | |
|---|---|
| *Function* | BuildJSON |
| *Parameters* | None |
| *Returns* | JSON Object |
| *Description* | BuildJSON pulls values out of the fields on the GUI. These values are associated with corresponding keys and used to build a list of pair objects. These pair objects are used to generate a JSON object, which is subsequently returned. |
| *Called by* | UserInput |

*Table 59. Function table for the SendToServer function of the Android application software.*

| | |
|---|---|
| *Function* | SendToServer |
| *Parameters* | JSON Object |
| *Returns* | Success – boolean value |
| *Description* | SendToServer builds an HTTP POST request to send the passed in JSON object to the web server at the RPi. The function will use the response code generated by the web server to decide if the operation was successful. On a success, the function returns true. Otherwise the function returns false. |
| *Called by* | UserInput |

[JAC]

As shown in Table 59, SendToServer sends updates to the web server to be stored and incorporated into the operation.

## 3.14 Software Calculations

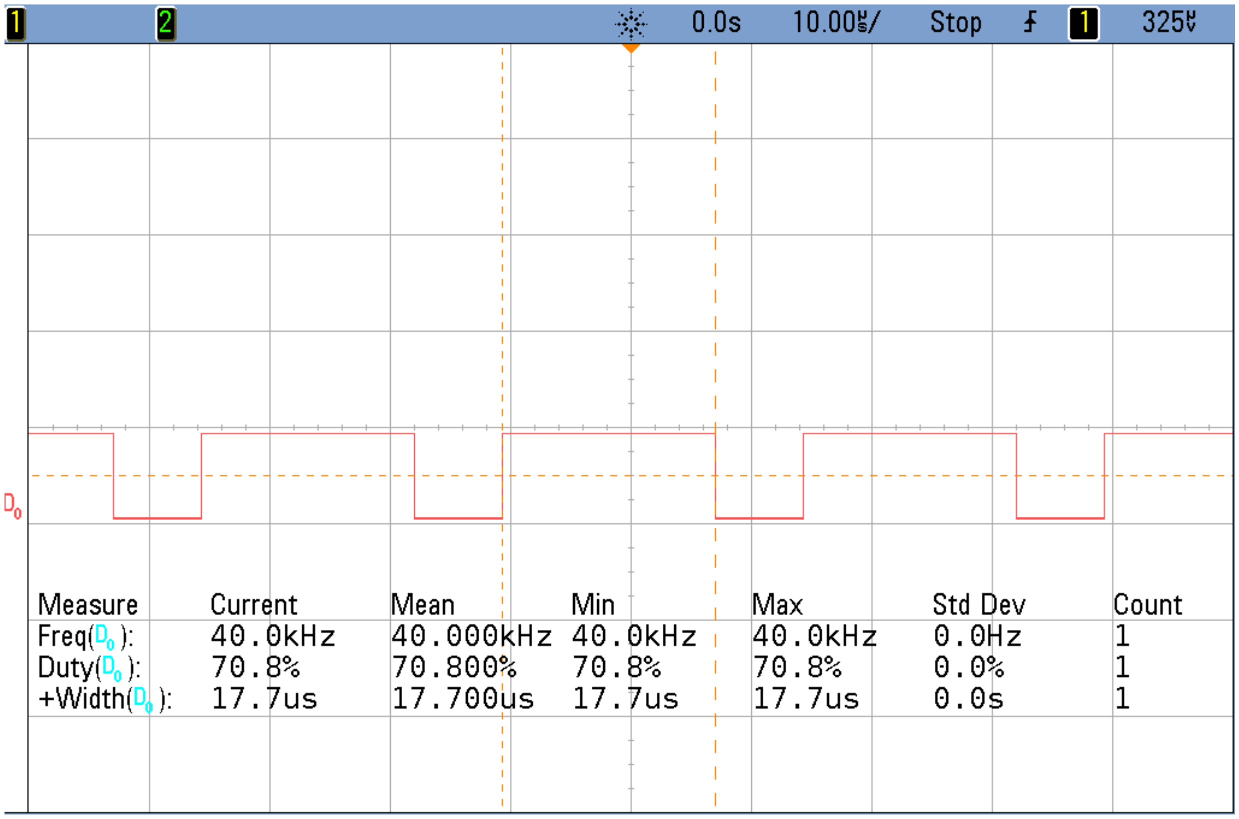
The spectrum of human hearing ranges from 20Hz to 20kHz. In order to avoid audible vibrations from the motor the PWM frequency must be kept outside of this range. A frequency below 20Hz would be far too slow to drive a motor effectively, so the frequency must instead exceed 20kHz. To account for possible fluctuations, the frequency should remain around 25kHz. This requirement is listed as Engineering Requirement #7 in Table 1.

In order to meet this engineering requirement, the function ConfigurePWM, shown in Table 36, function is used to set the PWM frequency. The ConfigurePWM function sets the PWM frequency to 40kHz by adjusting the clock frequency and setting the range of the PWM. As shown above in the ConfigurePWM code snippet, the clock divisor is set to ten and the range is set to forty-eight. Setting the clock divisor to ten divides the RPi clock of 19.2MHz down to 1.92MHz and setting the range to forty-eight divides the 1.92MHz into forty-eight discrete steps leaving a PWM frequency of 40kHz. Having a frequency of 40kHz for the PWM exceeds the required minimum frequency of 25kHz for the engineering requirement. Figure 27 shows an example PWM output from the RPi.

$$Frequency\ (Hz) = \frac{19.2MHz}{Clock\ Divisor * Range}$$

$$= \frac{19.2MHz}{10 * 48}$$

$$= 40kHz$$



| Measure | Current | Mean | Min | Max | Std Dev | Count |
|---|---|---|---|---|---|---|
| Freq(D0): | 40.0kHz | 40.000kHz | 40.0kHz | 40.0kHz | 0.0Hz | 1 |
| Duty(D0): | 70.8% | 70.800% | 70.8% | 70.8% | 0.0% | 1 |
| +Width(D0): | 17.7us | 17.700us | 17.7us | 17.7us | 0.0s | 1 |

*Figure 27. Output Waveform of RPi PWM.*

It is important to take into consideration the amount of data consumed by the Android application in its communication with the web server. With the prevalence of data caps for cellular data plans, excessive usage from unnecessary communication could deter possible users. The application should use no more than 1 MB of data per day yielding a worst case of only 30 MB of data consumption over a month.

[JBB]

# 4. Parts

## 4.1 Parts List

*Table 60. Smart Fan Parts List.*

| Qty. | Refdes | Part Num. | Description |
|---|---|---|---|
| 1 | | 181 | Standard LCD 16x2 + extras - white on blue |
| 1 | U1 | 3055 | Raspberry Pi 3 vB |
| 1 | | 1988 | GPIO Ribbon Cable for Raspberry Pi Model A+/B+/Pi 2/Pi 3 - (40 pins) |
| 1 | | 2029 | Assembled Pi Cobbler Plus - Breakout Cable - for Pi B+/A+/Pi 2/Pi 3 |
| 1 | | 2258 | Adafruit Raspberry Pi B+ / Pi 2 / Pi 3 Case - Smoke Base - w/ Clear Top |
| 1 | | 1995 | 5V 2.4A Switching Power Supply w/ 20AWG 6' MicroUSB Cable |
| 1 | | 3082 | Aluminum Heat Sink for Raspberry Pi 3 - 15 x 15 x 15mm |
| 1 | | 3259 | 8GB MicroSD Card with NOOBS 1.9 |
| 1 | U2 | 1083 | ADS1015 12-Bit ADC - 4 Channel with Programmable Gain Amplifier |
| 1 | | 571 | Adafruit Perma-Proto Half-sized Breadboard PCB - 3 Pack! |
| 8 | Q1, Q2, Q5, Q6, Q7, Q8, Q9, Q10 | 2SD2444KT146R | NPN Transistor |
| 2 | Q3, Q4 | BCP69-16,115 | PNP Transistor |
| 4 | D1, D2, D3, D4 | ES2A | Standard Diode |
| 2 | D5, D6 | CUS520,H3F | Schottky Diode |
| 4 | R1, R2, R3, R4 | FMP300FTF73-100R | 100 Ohm Resistor |
| 4 | R5, R6, R7, R8 | LR1F1K0 | 1k Ohm Resistor |
| 1 | M1 | E30-150 | DC Motor |
| 4 | R9, R10, R11, R12 | CRG1206F470R | 470 Ohm Resistor |
| 6 | C1, C2, C3, C4, C5, C6 | CC0805KRX7R9BB103 | 0.01µF Capacitor |
| 1 | SW4 | 377 | Rotary Encoder + Extras |
| 1 | SW1 | 3105 | Power Button |
| 2 | SW2, SW3 | 1445 | Forward and Reverse Switches |
| 1 | T1 | FS24-100-C2-B | 120/24V transformer |
| 1 | BR1 | DF04M | diode bridge |

| | | | |
|---|---|---|---|
| 1 | U1 | LT3791EFE-1#PBF | 24V voltage regulator |
| 1 | U2 | LT1085CT-5#PBF | 5V voltage regulator |
| 1 | U3 | MCP9700A-E/TO-ND | temperature sensor |
| 4 | Q11, Q12, Q13, Q14 | RJK0651DPB-00#J5 | NMOS |
| 2 | D7, D8 | BAT46WH,115 | Schottky Diode |
| 1 | L1 | 39S103C | Inductor |
| 2 | C7, C8 | UVR1V222MHD | 2200uF Capacitor |
| 2 | C9, C10 | 8.60021E+11 | 1uF Capacitor |
| 3 | C11, C12, C22 | 400PX4R7MEFCTA8X11.5 | 4.7uF Capacitor |
| 1 | C13 | EEU-FC1H470 | 47uF Capacitor |
| 1 | C14 | C322C474M5U5TA | 470nF Capacitor |
| 1 | C15 | K333K15X7RF5TH5 | 33nF Capacitor |
| 1 | C16 | C315C103K5R5TA7303 | 10nF Capacitor |
| 3 | C17, C18, C19 | K104K10X7RF5UH5 | 0.1uF Capacitor |
| 1 | C20 | UVY1E221MED1TD | 220uF Capacitor |
| 1 | C21 | FK16X7R1E106K | 10uF Capacitor |
| 2 | R13, R14 | CF14JT100K | 100k Ohm Resistor |
| 1 | R15 | CF14JT200K | 200k Ohm Resistor |
| 2 | R16, R17 | MFR-25FBF52-499K | 499k Ohm Resistor |
| 1 | R18 | RNF14FTD27K4 | 27.4k Ohm Resistor |
| 1 | R19 | MFR-25FBF52-56K2 | 56.2k Ohm Resistor |
| 1 | R20 | CS5DR003E | 3m Ohm Resistor |
| 1 | R21 | CF14JT51R0 | 51 Ohm Resistor |
| 1 | R22 | HVR2500001473FR500 | 147k Ohm Resistor |
| 1 | R23 | BR3FB15L0 | 15m Ohm Resistor |
| 1 | R24 | 14AFR004E | 4m Ohm Resistor |
| 1 | R25 | MFR-25FBF52-73K2 | 73.2k Ohm Resistor |
| 1 | R26 | MFR-25FBF52-3K83 | 3.83k Ohm Resistor |
| 1 | K1 | HY1-4.5V | 3.3V Relay |
| 1 | R27 | RNMF14FTC120R | 120 Ohm Resistor |

[PAG, JDR]

## 4.2 Budget

*Table 61. Smart Fan Budget.*

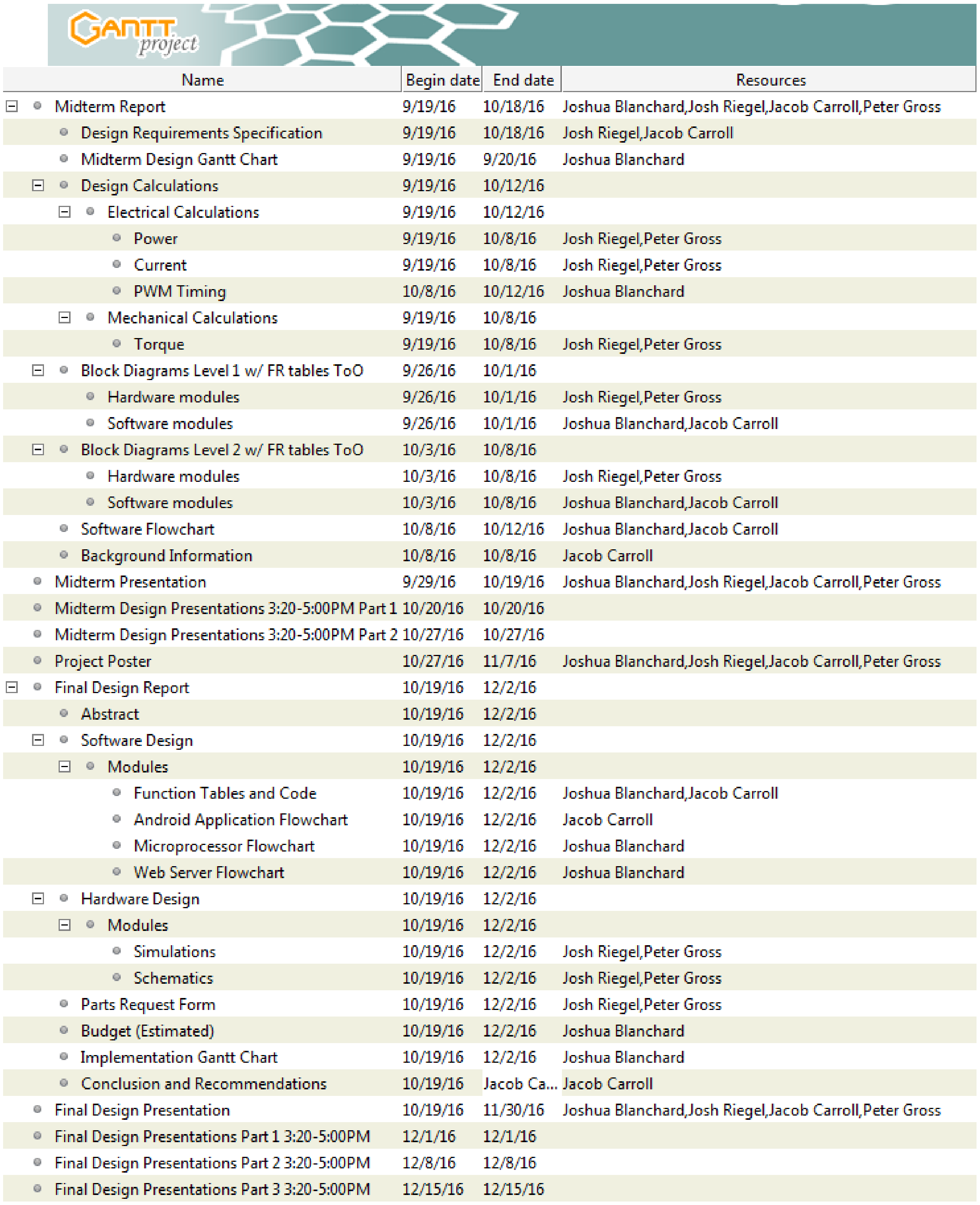
| | | | **Unit** | **Total** |
|---|---|---|---|---|
| **Qty.** | **Part Num.** | **Description** | **Cost** | **Cost** |
| 1 | 181 | Standard LCD 16x2 + extras - white on blue | 9.95 | 9.95 |
| 1 | 3055 | Raspberry Pi 3 vB | 39.95 | 39.95 |
| 1 | 1988 | GPIO Ribbon Cable for Raspberry Pi Model A+/B+/Pi 2/Pi 3 - (40 pins) | 2.95 | 2.95 |

| | | | | |
|---|---|---|---|---|
| 1 | 2029 | Assembled Pi Cobbler Plus - Breakout Cable - for Pi B+/A+/Pi 2/Pi 3 | 6.95 | 6.95 |
| 1 | 2258 | Adafruit Raspberry Pi B+ / Pi 2 / Pi 3 Case - Smoke Base - w/ Clear Top | 7.95 | 7.95 |
| 1 | 1995 | 5V 2.4A Switching Power Supply w/ 20AWG 6' MicroUSB Cable | 7.95 | 7.95 |
| 1 | 3082 | Aluminum Heat Sink for Raspberry Pi 3 - 15 x 15 x 15mm | 1.95 | 1.95 |
| 1 | 3259 | 8GB MicroSD Card with NOOBS 1.9 | 9.95 | 9.95 |
| 1 | 1083 | ADS1015 12-Bit ADC - 4 Channel with Programmable Gain Amplifier | 9.95 | 9.95 |
| 1 | 571 | Adafruit Perma-Proto Half-sized Breadboard PCB - 3 Pack! | 12.50 | 12.50 |
| 8 | 2SD2444KT146R | NPN Transistor | 0.47 | 3.76 |
| 2 | BCP69-16,115 | PNP Transistor | 0.40 | 0.80 |
| 4 | ES2A | Standard Diode | 0.49 | 1.96 |
| 2 | CUS520,H3F | Schottky Diode | 0.21 | 0.42 |
| 4 | FMP300FTF73-100R | 100 Ohm Resistor | 0.39 | 1.56 |
| 4 | LR1F1K0 | 1k Ohm Resistor | 0.13 | 0.52 |
| 1 | E30-150 | DC Motor | 79.00 | 79.00 |
| 4 | CRG1206F470R | 470 Ohm Resistor | 0.10 | 0.40 |
| 6 | CC0805KRX7R9BB103 | 0.01µF Capacitor | 0.10 | 0.60 |
| 1 | 377 | Rotary Encoder + Extras | 4.50 | 4.50 |
| 1 | 3105 | Power Button | 1.95 | 1.95 |
| 2 | 1445 | Forward and Reverse Switches | 0.95 | 1.90 |
| 1 | FS24-100-C2-B | 120/24V transformer | 5.27 | 5.27 |
| 1 | DF04M | diode bridge | 0.41 | 0.41 |
| 1 | LT3791EFE-1#PBF | 24V voltage regulator | 10.15 | 10.15 |
| 1 | LT1085CT-5#PBF | 5V voltage regulator | 6.78 | 6.78 |
| 1 | MCP9700A-E/TO-ND | temperature sensor | 0.31 | 0.31 |
| 4 | RJK0651DPB-00#J5 | NMOS | 1.37 | 5.48 |
| 2 | BAT46WH,115 | Schottky Diode | 0.38 | 0.76 |
| 1 | 39S103C | Inductor | 1.50 | 1.50 |
| 2 | UVR1V222MHD | 2200uF Capacitor | 0.98 | 1.96 |
| 2 | 8.60021E+11 | 1uF Capacitor | 0.11 | 0.22 |
| 3 | 400PX4R7MEFCTA8X11.5 | 4.7uF Capacitor | 0.46 | 1.38 |
| 1 | EEU-FC1H470 | 47uF Capacitor | 0.34 | 0.34 |
| 1 | C322C474M5U5TA | 470nF Capacitor | 0.49 | 0.49 |
| 1 | K333K15X7RF5TH5 | 33nF Capacitor | 0.22 | 0.22 |
| 1 | C315C103K5R5TA7303 | 10nF Capacitor | 0.25 | 0.25 |
| 3 | K104K10X7RF5UH5 | 0.1uF Capacitor | 0.18 | 0.54 |
| 1 | UVY1E221MED1TD | 220uF Capacitor | 0.30 | 0.30 |

|  |  |  |  |  |
|---|---|---|---|---|
| 1 | FK16X7R1E106K | 10uF Capacitor | 0.63 | 0.63 |
| 2 | CF14JT100K | 100k Ohm Resistor | 0.10 | 0.20 |
| 1 | CF14JT200K | 200k Ohm Resistor | 0.10 | 0.10 |
| 2 | MFR-25FBF52-499K | 499k Ohm Resistor | 0.10 | 0.20 |
| 1 | RNF14FTD27K4 | 27.4k Ohm Resistor | 0.10 | 0.10 |
| 1 | MFR-25FBF52-56K2 | 56.2k Ohm Resistor | 0.10 | 0.10 |
| 1 | CS5DR003E | 3m Ohm Resistor | 10.42 | 10.42 |
| 1 | CF14JT51R0 | 51 Ohm Resistor | 0.10 | 0.10 |
| 1 | HVR2500001473FR500 | 147k Ohm Resistor | 0.47 | 0.47 |
| 1 | BR3FB15L0 | 15m Ohm Resistor | 0.78 | 0.78 |
| 1 | 14AFR004E | 4m Ohm Resistor | 1.77 | 1.77 |
| 1 | MFR-25FBF52-73K2 | 73.2k Ohm Resistor | 0.10 | 0.10 |
| 1 | MFR-25FBF52-3K83 | 3.83k Ohm Resistor | 0.10 | 0.10 |
| 1 | HY1-4.5V | 3.3V Relay | 3.74 | 3.74 |
| 1 | RNMF14FTC120R | 120 Ohm Resistor | 0.10 | 0.10 |
|  |  |  | **Total** | $262.69 |

[PAG, JDR]

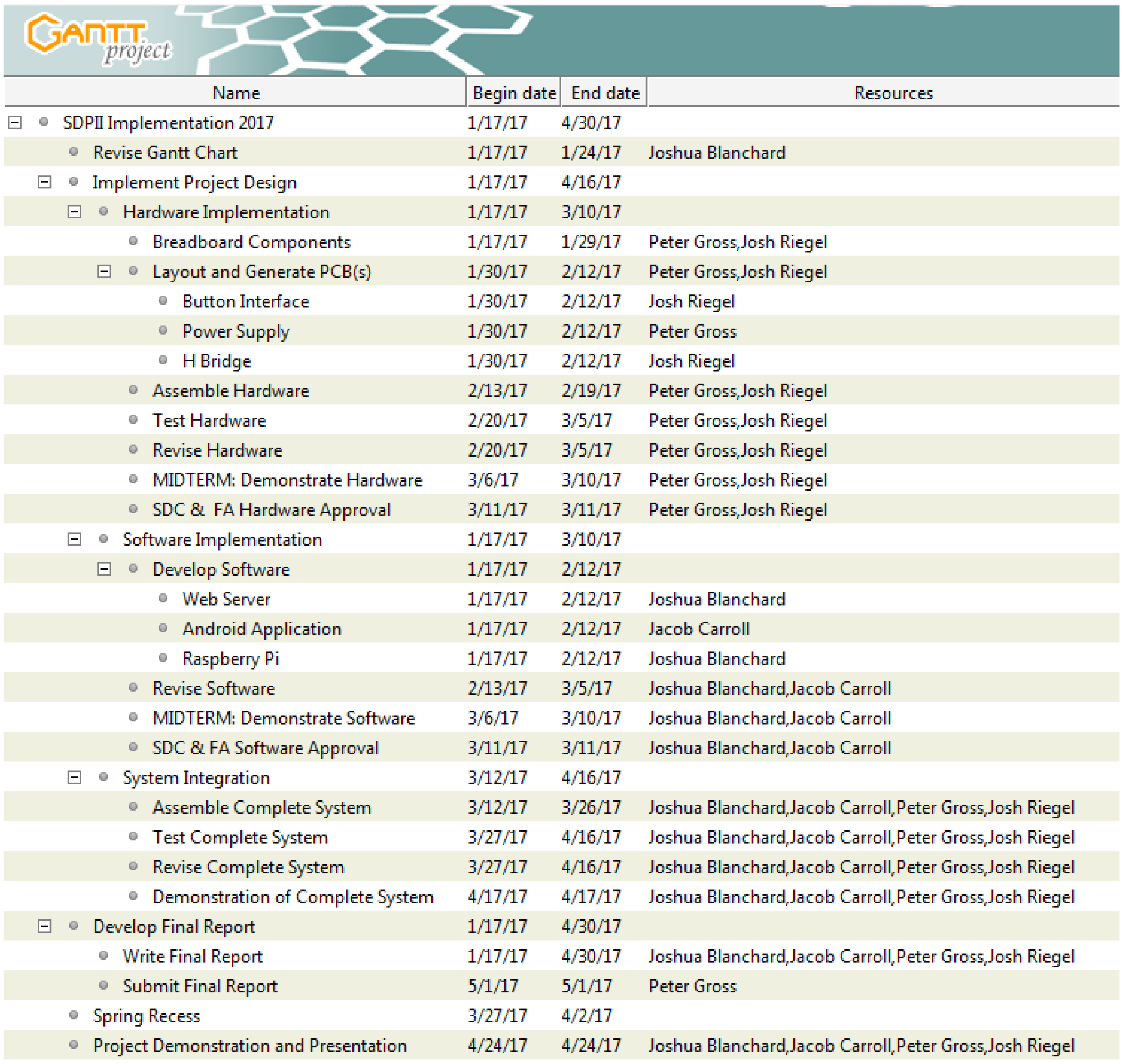
## 5. Project Schedule



| Name | Begin date | End date | Resources |
|---|---|---|---|
| Midterm Report | 9/19/16 | 10/18/16 | Joshua Blanchard,Josh Riegel,Jacob Carroll,Peter Gross |
| Design Requirements Specification | 9/19/16 | 10/18/16 | Josh Riegel,Jacob Carroll |
| Midterm Design Gantt Chart | 9/19/16 | 9/20/16 | Joshua Blanchard |
| Design Calculations | 9/19/16 | 10/12/16 | |
| Electrical Calculations | 9/19/16 | 10/12/16 | |
| Power | 9/19/16 | 10/8/16 | Josh Riegel,Peter Gross |
| Current | 9/19/16 | 10/8/16 | Josh Riegel,Peter Gross |
| PWM Timing | 10/8/16 | 10/12/16 | Joshua Blanchard |
| Mechanical Calculations | 9/19/16 | 10/8/16 | |
| Torque | 9/19/16 | 10/8/16 | Josh Riegel,Peter Gross |
| Block Diagrams Level 1 w/ FR tables ToO | 9/26/16 | 10/1/16 | |
| Hardware modules | 9/26/16 | 10/1/16 | Josh Riegel,Peter Gross |
| Software modules | 9/26/16 | 10/1/16 | Joshua Blanchard,Jacob Carroll |
| Block Diagrams Level 2 w/ FR tables ToO | 10/3/16 | 10/8/16 | |
| Hardware modules | 10/3/16 | 10/8/16 | Josh Riegel,Peter Gross |
| Software modules | 10/3/16 | 10/8/16 | Joshua Blanchard,Jacob Carroll |
| Software Flowchart | 10/8/16 | 10/12/16 | Joshua Blanchard,Jacob Carroll |
| Background Information | 10/8/16 | 10/8/16 | Jacob Carroll |
| Midterm Presentation | 9/29/16 | 10/19/16 | Joshua Blanchard,Josh Riegel,Jacob Carroll,Peter Gross |
| Midterm Design Presentations 3:20-5:00PM Part 1 | 10/20/16 | 10/20/16 | |
| Midterm Design Presentations 3:20-5:00PM Part 2 | 10/27/16 | 10/27/16 | |
| Project Poster | 10/27/16 | 11/7/16 | Joshua Blanchard,Josh Riegel,Jacob Carroll,Peter Gross |
| Final Design Report | 10/19/16 | 12/2/16 | |
| Abstract | 10/19/16 | 12/2/16 | |
| Software Design | 10/19/16 | 12/2/16 | |
| Modules | 10/19/16 | 12/2/16 | |
| Function Tables and Code | 10/19/16 | 12/2/16 | Joshua Blanchard,Jacob Carroll |
| Android Application Flowchart | 10/19/16 | 12/2/16 | Jacob Carroll |
| Microprocessor Flowchart | 10/19/16 | 12/2/16 | Joshua Blanchard |
| Web Server Flowchart | 10/19/16 | 12/2/16 | Joshua Blanchard |
| Hardware Design | 10/19/16 | 12/2/16 | |
| Modules | 10/19/16 | 12/2/16 | |
| Simulations | 10/19/16 | 12/2/16 | Josh Riegel,Peter Gross |
| Schematics | 10/19/16 | 12/2/16 | Josh Riegel,Peter Gross |
| Parts Request Form | 10/19/16 | 12/2/16 | Josh Riegel,Peter Gross |
| Budget (Estimated) | 10/19/16 | 12/2/16 | Joshua Blanchard |
| Implementation Gantt Chart | 10/19/16 | 12/2/16 | Joshua Blanchard |
| Conclusion and Recommendations | 10/19/16 | Jacob Ca... | Jacob Carroll |
| Final Design Presentation | 10/19/16 | 11/30/16 | Joshua Blanchard,Josh Riegel,Jacob Carroll,Peter Gross |
| Final Design Presentations Part 1 3:20-5:00PM | 12/1/16 | 12/1/16 | |
| Final Design Presentations Part 2 3:20-5:00PM | 12/8/16 | 12/8/16 | |
| Final Design Presentations Part 3 3:20-5:00PM | 12/15/16 | 12/15/16 | |

*Figure 28. Final Design Gantt chart.*

[JBB]



| Name | Begin date | End date | Resources |
|---|---|---|---|
| SDPII Implementation 2017 | 1/17/17 | 4/30/17 | |
| Revise Gantt Chart | 1/17/17 | 1/24/17 | Joshua Blanchard |
| Implement Project Design | 1/17/17 | 4/16/17 | |
| Hardware Implementation | 1/17/17 | 3/10/17 | |
| Breadboard Components | 1/17/17 | 1/29/17 | Peter Gross,Josh Riegel |
| Layout and Generate PCB(s) | 1/30/17 | 2/12/17 | Peter Gross,Josh Riegel |
| Button Interface | 1/30/17 | 2/12/17 | Josh Riegel |
| Power Supply | 1/30/17 | 2/12/17 | Peter Gross |
| H Bridge | 1/30/17 | 2/12/17 | Josh Riegel |
| Assemble Hardware | 2/13/17 | 2/19/17 | Peter Gross,Josh Riegel |
| Test Hardware | 2/20/17 | 3/5/17 | Peter Gross,Josh Riegel |
| Revise Hardware | 2/20/17 | 3/5/17 | Peter Gross,Josh Riegel |
| MIDTERM: Demonstrate Hardware | 3/6/17 | 3/10/17 | Peter Gross,Josh Riegel |
| SDC & FA Hardware Approval | 3/11/17 | 3/11/17 | Peter Gross,Josh Riegel |
| Software Implementation | 1/17/17 | 3/10/17 | |
| Develop Software | 1/17/17 | 2/12/17 | |
| Web Server | 1/17/17 | 2/12/17 | Joshua Blanchard |
| Android Application | 1/17/17 | 2/12/17 | Jacob Carroll |
| Raspberry Pi | 1/17/17 | 2/12/17 | Joshua Blanchard |
| Revise Software | 2/13/17 | 3/5/17 | Joshua Blanchard,Jacob Carroll |
| MIDTERM: Demonstrate Software | 3/6/17 | 3/10/17 | Joshua Blanchard,Jacob Carroll |
| SDC & FA Software Approval | 3/11/17 | 3/11/17 | Joshua Blanchard,Jacob Carroll |
| System Integration | 3/12/17 | 4/16/17 | |
| Assemble Complete System | 3/12/17 | 3/26/17 | Joshua Blanchard,Jacob Carroll,Peter Gross,Josh Riegel |
| Test Complete System | 3/27/17 | 4/16/17 | Joshua Blanchard,Jacob Carroll,Peter Gross,Josh Riegel |
| Revise Complete System | 3/27/17 | 4/16/17 | Joshua Blanchard,Jacob Carroll,Peter Gross,Josh Riegel |
| Demonstration of Complete System | 4/17/17 | 4/17/17 | Joshua Blanchard,Jacob Carroll,Peter Gross,Josh Riegel |
| Develop Final Report | 1/17/17 | 4/30/17 | |
| Write Final Report | 1/17/17 | 4/30/17 | Joshua Blanchard,Jacob Carroll,Peter Gross,Josh Riegel |
| Submit Final Report | 5/1/17 | 5/1/17 | Peter Gross |
| Spring Recess | 3/27/17 | 4/2/17 | |
| Project Demonstration and Presentation | 4/24/17 | 4/24/17 | Joshua Blanchard,Jacob Carroll,Peter Gross,Josh Riegel |

*Figure 29. Proposed Implementation Gantt chart.*

[JBB]

## 6. Design Team Information

Joshua Blanchard (CpE), Project Lead
Jacob Carroll (CpE), Software Lead
Peter Gross (EE), Archivist
Josh Riegel (EE), Hardware Lead

## 7. Conclusions and Recommendations

This Smart Fan will allow users to run it when they want, how they want. The device expands upon an antiquated technology by introducing Internet-connectivity and programmability. The stages through which this project was developed allowed for a step-by-step approach during which feedback could be obtained. Block diagrams, schematics, and function tables offer a multi-leveled view of the system and how it will be implemented. During the design process many requirements from an engineering point-of-view were considered, but it was also important to keep in mind the perspective and interests of a common user.

Recommendations for the future would include a more proactive schedule in which simulations and experimentation are performed earlier in the process. Without implementing or simulating anything with actual components the design process was more difficult due to incorrect assumptions and misconceptions.

[JAC]

## 8. References

1. Bell, “Self-programmable temperature control system for a heating and cooling system,” U.S. Patent 5 088 645, Feb 18, 1992.

2. Y. Tsuchiyama, “Low power *electrical fan* motor and heater thermal protection circuit for air conditioner,” U.S. Patent 5 131 455, July 21, 1992.

3. R. M. Nair, “Intelligent internal fan controller,” U.S. Patent 6 318 965, Nov 20, 2001.

4. P. Simard, “Electric timer for controlling power to a fan,” U.S. Patent 8 441 155, May 14, 2013.

5. R. X. Cui, H. L Zhao, G. Chen and C. G. Guo, “Smart fan design based on PSoC,” in *Applied Mechanics and Materials*, vol. 347-350, pp. 252-256, 2013. doi: 10.4028/www.scientific.net/AMM.347-350.252

6. C.-H. Pang, J.-V. Lee, Y.-D. Chuah, Y.-C. Tan and N. Debnach, “Design of a microcontroller based fan motor controller for smart home environment,” in *International Journal of Smart Home*, vol. 7, iss. 4, pp. 233-246, 2013.

7. L. Guillot, P. Dupuy, M. Plachy, A. El-Habibi and E. Serre, "Automotive Fan Control with Smart Power Switch [From Mind to Market]," in *IEEE Industrial Electronics Magazine*, vol. 2, no. 1, pp. 4-54, March 2008. doi: 10.1109/MIE.2008.917148

[JDR]

# 9. Appendices

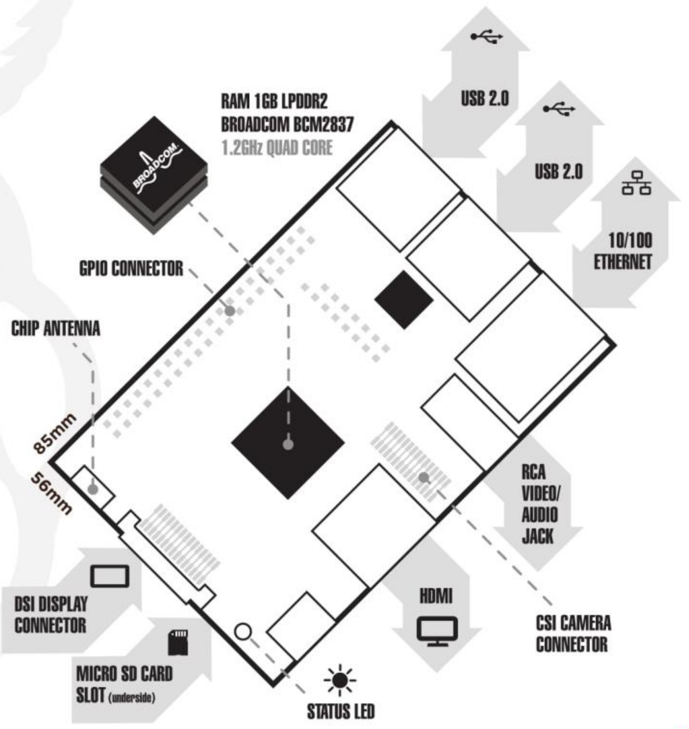
## 9.1 Appendix A: Datasheets

### 9.1.1 Raspberry Pi 3 Model B Datasheet

Raspberry Pi

**Raspberry Pi 3 Model B**

| | |
|---|---|
| **Product Name** | **Raspberry Pi 3** |
| **Product Description** | The Raspberry Pi 3 Model B is the third generation Raspberry Pi. This powerful credit-card sized single board computer can be used for many applications and supersedes the original Raspberry Pi Model B+ and Raspberry Pi 2 Model B. Whilst maintaining the popular board format the Raspberry Pi 3 Model B brings you a more powerful processer, 10x faster than the first generation Raspberry Pi. Additionally it adds wireless LAN & Bluetooth connectivity making it the ideal solution for powerful connected designs. |
| **RS Part Number** | **896-8660** |



www.rs-components.com/raspberrypi

Raspberry Pi

# Raspberry Pi 3 Model B

## Specifications

| | |
|---|---|
| **Processor** | Broadcom BCM2387 chipset.<br>1.2GHz Quad-Core ARM Cortex-A53<br>802.11 b/g/n Wireless LAN and Bluetooth 4.1 (Bluetooth Classic and LE) |
| **GPU** | Dual Core VideoCore IV® Multimedia Co-Processor. Provides Open GL ES 2.0, hardware-accelerated OpenVG, and 1080p30 H.264 high-profile decode.<br>Capable of 1Gpixel/s, 1.5Gtexel/s or 24GFLOPs with texture filtering and DMA infrastructure |
| **Memory** | 1GB LPDDR2 |
| **Operating System** | Boots from Micro SD card, running a version of the Linux operating system or Windows 10 IoT |
| **Dimensions** | 85 x 56 x 17mm |
| **Power** | Micro USB socket 5V1, 2.5A |

## Connectors:

| | |
|---|---|
| **Ethernet** | 10/100 BaseT Ethernet socket |
| **Video Output** | HDMI (rev 1.3 & 1.4<br>Composite RCA (PAL and NTSC) |
| **Audio Output** | Audio Output 3.5mm jack, HDMI<br>USB 4 x USB 2.0 Connector |
| **GPIO Connector** | 40-pin 2.54 mm (100 mil) expansion header: 2x20 strip<br>Providing 27 GPIO pins as well as +3.3 V, +5 V and GND supply lines |
| **Camera Connector** | 15-pin MIPI Camera Serial Interface (CSI-2) |
| **Display Connector** | Display Serial Interface (DSI) 15 way flat flex cable connector with two data lanes and a clock lane |
| **Memory Card Slot** | Push/pull Micro SDIO |

## Key Benefits

- Low cost
- 10x faster processing
- Consistent board format
- Added connectivity

## Key Applications

- Low cost PC/tablet/laptop
- Media centre
- Industrial/Home automation
- Print server
- Web camera
- Wireless access point
- Environmental sensing/monitoring (e.g. weather station)
- IoT applications
- Robotics
- Server/cloud server
- Security monitoring
- Gaming

RS

www.rs-components.com/raspberrypi

### 9.1.2 Triad Magnetics FS24-100-C2 Datasheet

TRIAD MAGNETICS

**CLASS 2/3 TRANSFORMER**

## FS24-100-C2

**Description:**
The FS24-100-C2 is a series/parallel primary and dual secondary, split bobbin design which operates with either a parallel input of 115V or a series input of 230V. The secondaries are 12V @ 0.1A each. They can be used independently (up to 300V difference between them) or in series for double the voltage or in parallel for double the current. The split bobbin design eliminates the need for costly electrostatic shielding.

**Electrical Specifications (@25C)**
1. Maximum Power: 2.5VA
2. Primary: Series: 230V; Parallel: 115V
3. Secondaries: 12.0V @ 0.1A each
4. Voltage Regulation: 25% TYP @ full load to no load
5. Temperature Rise: 25C TYP
6. Hipot tested 100% at 4200 VRMS pri to sec
7. Hipot tested 100% at 2160 VRMS sec to sec
8. Inherently Limited. No fusing required.

**Construction:**
Three flange bobbin construction with primaries and secondaries wound side by side for low capacitive coupling. UL Class F Insulation System (155°C).

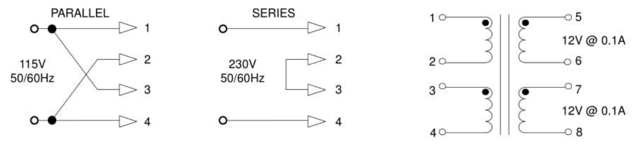
**Agency File:**
UL: File E65390, UL 5085-1 & 3 (1585), Class 2 not wet / Class 3 wet Transformer
cUL: File E65390, UL 5085-1 & 3 (1585) For Canadian Use (CSA 22.2, No.66.3-06)
TUV Certificate No.: R72120839, EN61558, Safety Isolating xfmr, general use

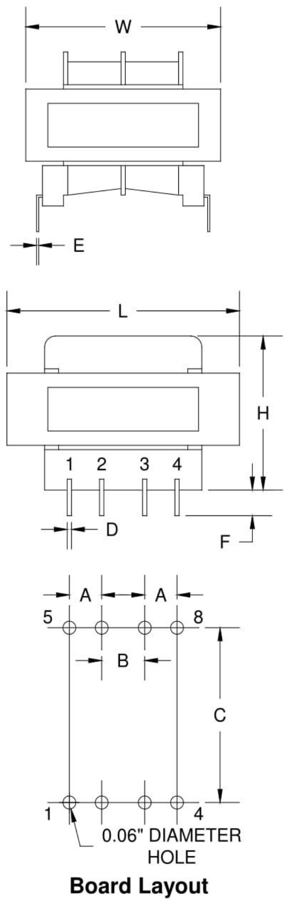
**Dimensions:** Units in inches.

| H | W | L | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|
| 1.187 | 1.125 | 1.375 | 0.250 | 0.250 | 1.200 | 0.041 | 0.020 | 0.234 |

Weight: 0.25 lbs

**Schematic:**





**Board Layout**

**RoHS Compliance:** As of manufacturing date February 2005, all standard products meet the requirements of 2011/65/EU, known as the RoHS initiative.

**Note:** Order and shipping documentation may have a "-B" suffix; this indicates Bulk packaging but does not show on the actual part number marked on the transformer.

*Upon printing, this document is considered "uncontrolled". Please contact Triad Magnetics website for the most current version. For soldering and washing information please see http://www.triadmagnetics.com/faq.html

| | | |
|---|---|---|
| www.TriadMagnetics.com<br>Phone: (951) 277-0757<br>Fax: (951) 277-2757 | 460 Harley Knox Blvd.<br>Perris, CA 92571 | Publish Date: June 6, 2016 |

### 9.1.3 Linear Technology LT1085 Datasheet

LINEAR TECHNOLOGY

## LT1083/LT1084/LT1085 Fixed

## 3A, 5A, 7.5A Low Dropout Positive Fixed Regulators

### FEATURES

- Three-Terminal 3.3V, 3.6V, 5V and 12V
- Output Current of 3A, 5A or 7.5A
- Operates Down to 1V Dropout
- Guaranteed Dropout Voltage at Multiple Current Levels
- Line Regulation: 0.015%
- Load Regulation: 0.1%
- 100% Thermal Limit Functional Test
- Adjustable Versions Available

### APPLICATIONS

- High Efficiency Linear Regulators
- Post Regulators for Switching Supplies
- Constant Current Regulators
- Battery Chargers

| DEVICE | OUTPUT CURRENT* |
|---|---|
| LT1083 | 7.5 Amps |
| LT1084 | 5.0 Amps |
| LT1085 | 3.0 Amps |

*For a 1.5A low dropout regulator see the LT1086 data sheet.

### DESCRIPTION

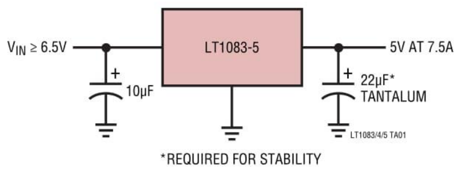
The LT®1083 series of positive adjustable regulators are designed to provide 3A, 5A and 7.5A with higher efficiency than currently available devices. All internal circuitry is designed to operate down to 1V input to output differential and the dropout voltage is fully specified as a function of load current. Dropout is guaranteed at a maximum of 1.5V at maximum output current, decreasing at lower load currents. On-chip trimming adjusts the output voltage to 1%. Current limit is also trimmed, minimizing the stress on both the regulator and power source circuitry under overload conditions.

The LT1083 series devices are pin compatible with older three-terminal regulators. A 10µF output capacitor is required on these new devices; however, this is usually included in most regulator designs.

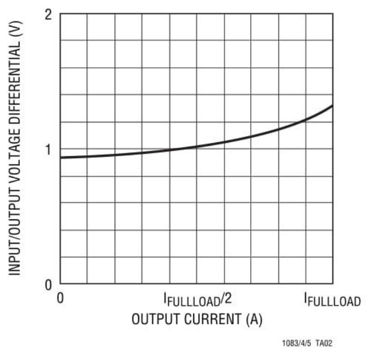
Unlike PNP regulators, where up to 10% of the output current is wasted as quiescent current, the LT1083 quiescent current flows into the load, increasing efficiency.

### TYPICAL APPLICATION

5V, 7.5A Regulator

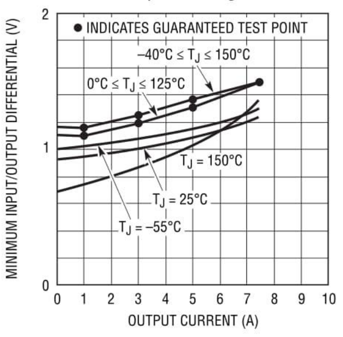


Dropout Voltage vs Output Current



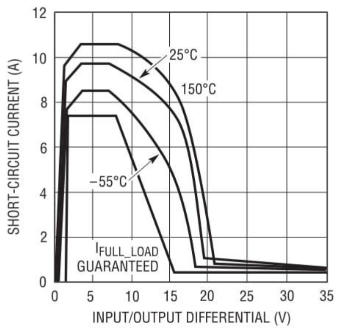
## TYPICAL PERFORMANCE CHARACTERISTICS

LT1083 Dropout Voltage



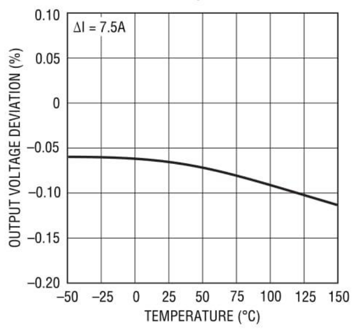
LT1083/4/5 FIXED G01

LT1083 Short-Circuit Current



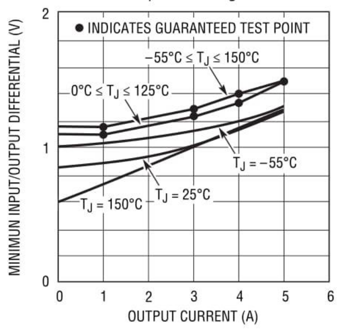
LT1083/4/5 FIXED G02

LT1083 Load Regulation



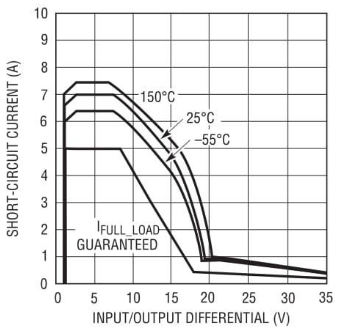
LT1083/4/5 FIXED G03

LT1084 Dropout Voltage



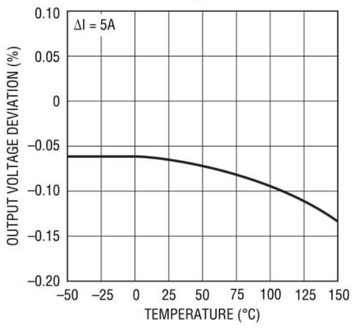
LT1083/4/5 FIXEDG04

LT1084 Short Circuit Current



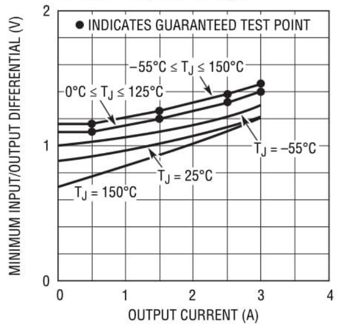
LT1083/4/5 FIXED G05

LT1084 Load Regulation



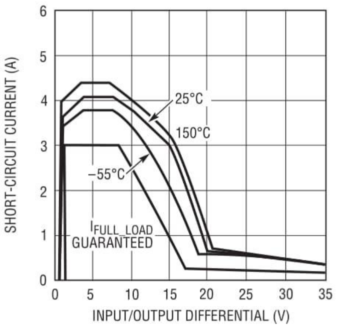
LT1083/4/5 FIXED G06

LT1085 Dropout Voltage



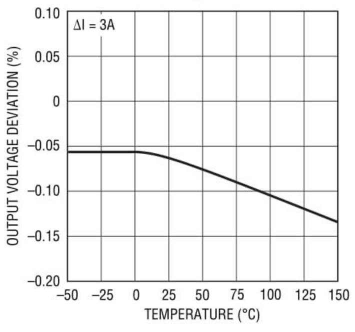
LT1083/4/5 FIXED G07

LT1085 Short-Circuit Current



LT1083/4/5 FIXED G08

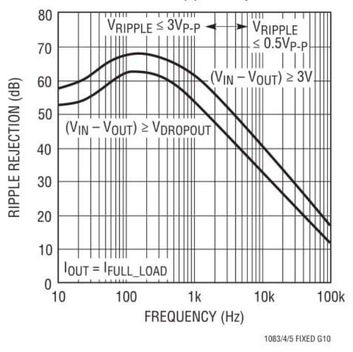
LT1085 Load Regulation



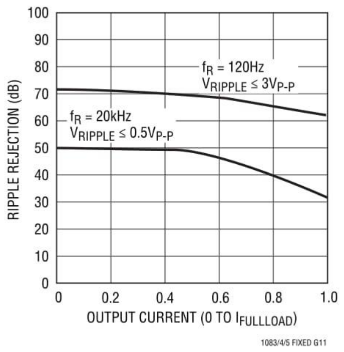
LT1083/4/5 FIXED G09

## TYPICAL PERFORMANCE CHARACTERISTICS

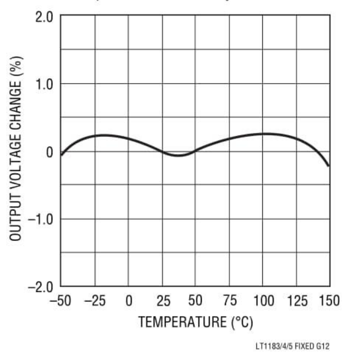
LT1083/4/5-5 Ripple Rejection



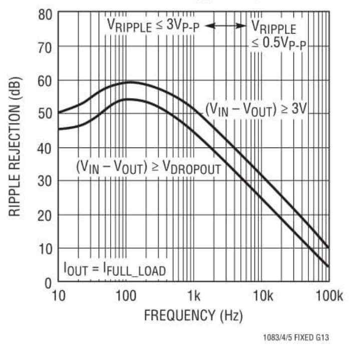
LT1083/4/5-5 Ripple Rejection vs Current



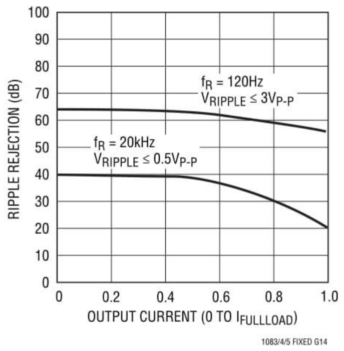
Temperature Stability



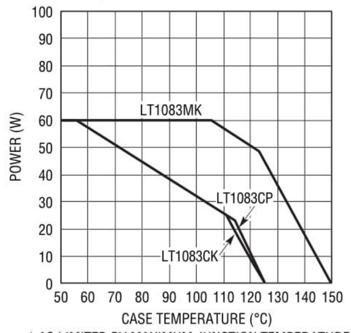
LT1083/4/5-12 Ripple Rejection



LT1083/4/5-12 Ripple Rejection vs Current

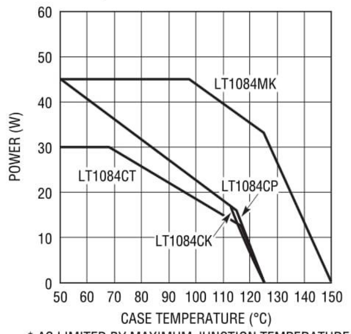


LT1083 Maximum Power Dissipation*



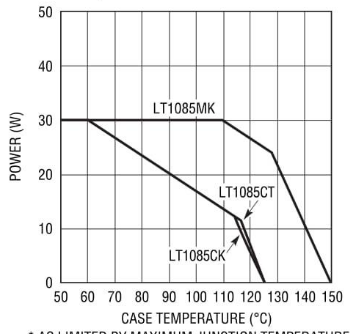
* AS LIMITED BY MAXIMUM JUNCTION TEMPERATURE

LT1084 Maximum Power Dissipation*



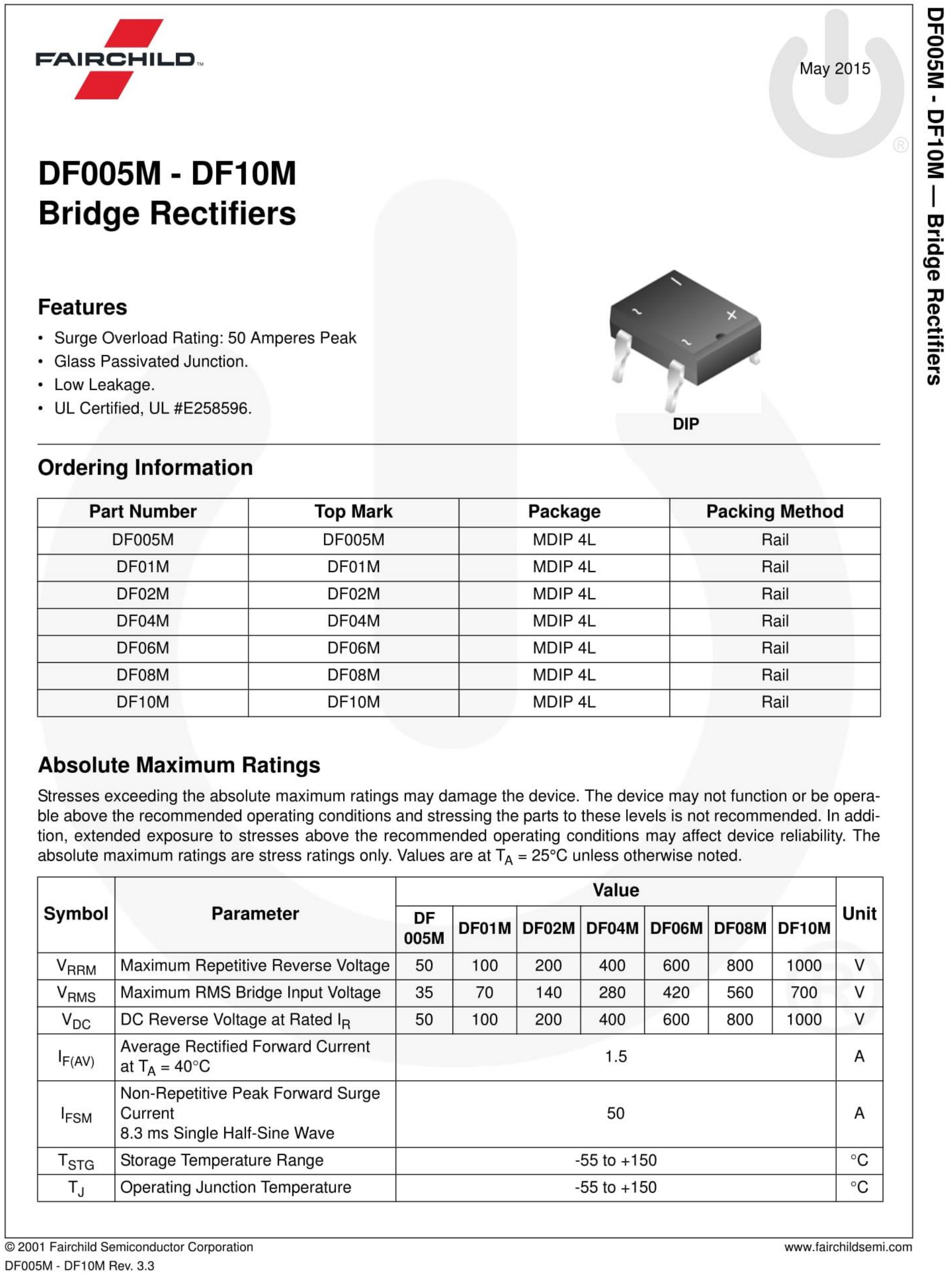
* AS LIMITED BY MAXIMUM JUNCTION TEMPERATURE

LT1085 Maximum Power Dissipation*



* AS LIMITED BY MAXIMUM JUNCTION TEMPERATURE

## 9.1.4 Fairchild Semiconductor DF04M Datasheet



FAIRCHILD

May 2015

DF005M - DF10M — Bridge Rectifiers

# DF005M - DF10M
# Bridge Rectifiers

### Features

- Surge Overload Rating: 50 Amperes Peak
- Glass Passivated Junction.
- Low Leakage.
- UL Certified, UL #E258596.

DIP

### Ordering Information

| Part Number | Top Mark | Package | Packing Method |
|---|---|---|---|
| DF005M | DF005M | MDIP 4L | Rail |
| DF01M | DF01M | MDIP 4L | Rail |
| DF02M | DF02M | MDIP 4L | Rail |
| DF04M | DF04M | MDIP 4L | Rail |
| DF06M | DF06M | MDIP 4L | Rail |
| DF08M | DF08M | MDIP 4L | Rail |
| DF10M | DF10M | MDIP 4L | Rail |

### Absolute Maximum Ratings

Stresses exceeding the absolute maximum ratings may damage the device. The device may not function or be operable above the recommended operating conditions and stressing the parts to these levels is not recommended. In addition, extended exposure to stresses above the recommended operating conditions may affect device reliability. The absolute maximum ratings are stress ratings only. Values are at $T_A$ = 25°C unless otherwise noted.

| Symbol | Parameter | Value | | | | | | | Unit |
|---|---|---|---|---|---|---|---|---|---|
| | | DF 005M | DF01M | DF02M | DF04M | DF06M | DF08M | DF10M | |
| $V_{RRM}$ | Maximum Repetitive Reverse Voltage | 50 | 100 | 200 | 400 | 600 | 800 | 1000 | V |
| $V_{RMS}$ | Maximum RMS Bridge Input Voltage | 35 | 70 | 140 | 280 | 420 | 560 | 700 | V |
| $V_{DC}$ | DC Reverse Voltage at Rated $I_R$ | 50 | 100 | 200 | 400 | 600 | 800 | 1000 | V |
| $I_{F(AV)}$ | Average Rectified Forward Current at $T_A$ = 40°C | 1.5 | | | | | | | A |
| $I_{FSM}$ | Non-Repetitive Peak Forward Surge Current 8.3 ms Single Half-Sine Wave | 50 | | | | | | | A |
| $T_{STG}$ | Storage Temperature Range | -55 to +150 | | | | | | | °C |
| $T_J$ | Operating Junction Temperature | -55 to +150 | | | | | | | °C |

© 2001 Fairchild Semiconductor Corporation
DF005M - DF10M Rev. 3.3

www.fairchildsemi.com

## Thermal Characteristics

Values are at $T_A = 25°C$ unless otherwise noted.

| Symbol | Parameter | Value | Unit |
|---|---|---|---|
| $P_D$ | Power Dissipation | 3.1 | W |
| $R_{\theta JA}$ | Thermal Resistance, Junction-to-Ambient(1), per Leg | 40 | °C/W |

**Note:**
1. Device mounted on PCB with 0.5 inch × 0.5 inch (13 mm × 13 mm).

## Electrical Characteristics

Values are at $T_A = 25°C$ unless otherwise noted.

| Symbol | Parameter | Conditions | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|
| $V_F$ | Forward Voltage, per Element | $I_F = 1.0$ A | | | 1.1 | V |
| $I_R$ | Reverse Current, per Element at Rated $V_R$ | $T_A = 25°C$ | | | 5.0 | µA |
| | | $T_A = 125°C$ | | | 500 | |
| $I^2t$ | Rating for Fusing ($t < 8.35$ ms) | | | | 10 | $A^2s$ |
| $C_J$ | Typical Capacitance, per Leg | $V_R = 4.0$ V, f = 1.0 MHz | | 25 | | pF |

## Typical Performance Characteristics

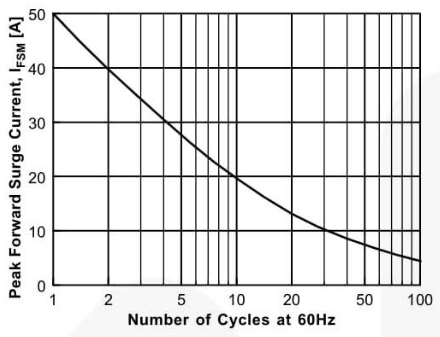


Figure 1. Non-Repetitive Surge Current

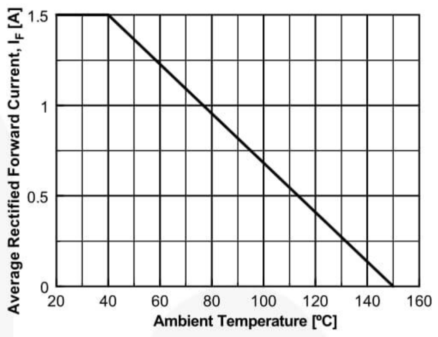


Figure 2. Forward Current Derating Curve

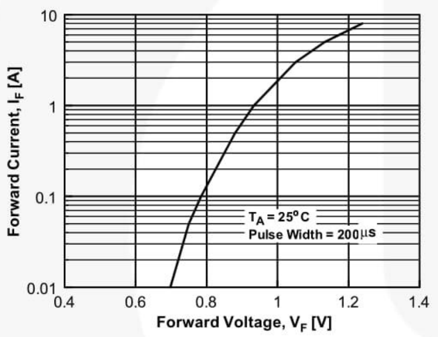


Figure 3. Forward Voltage Characteristics

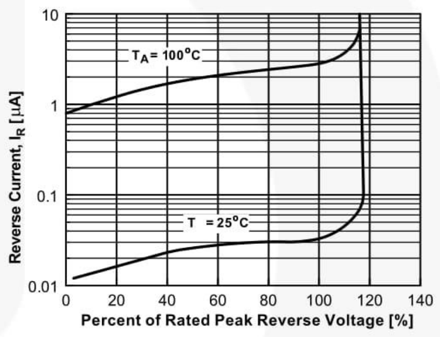
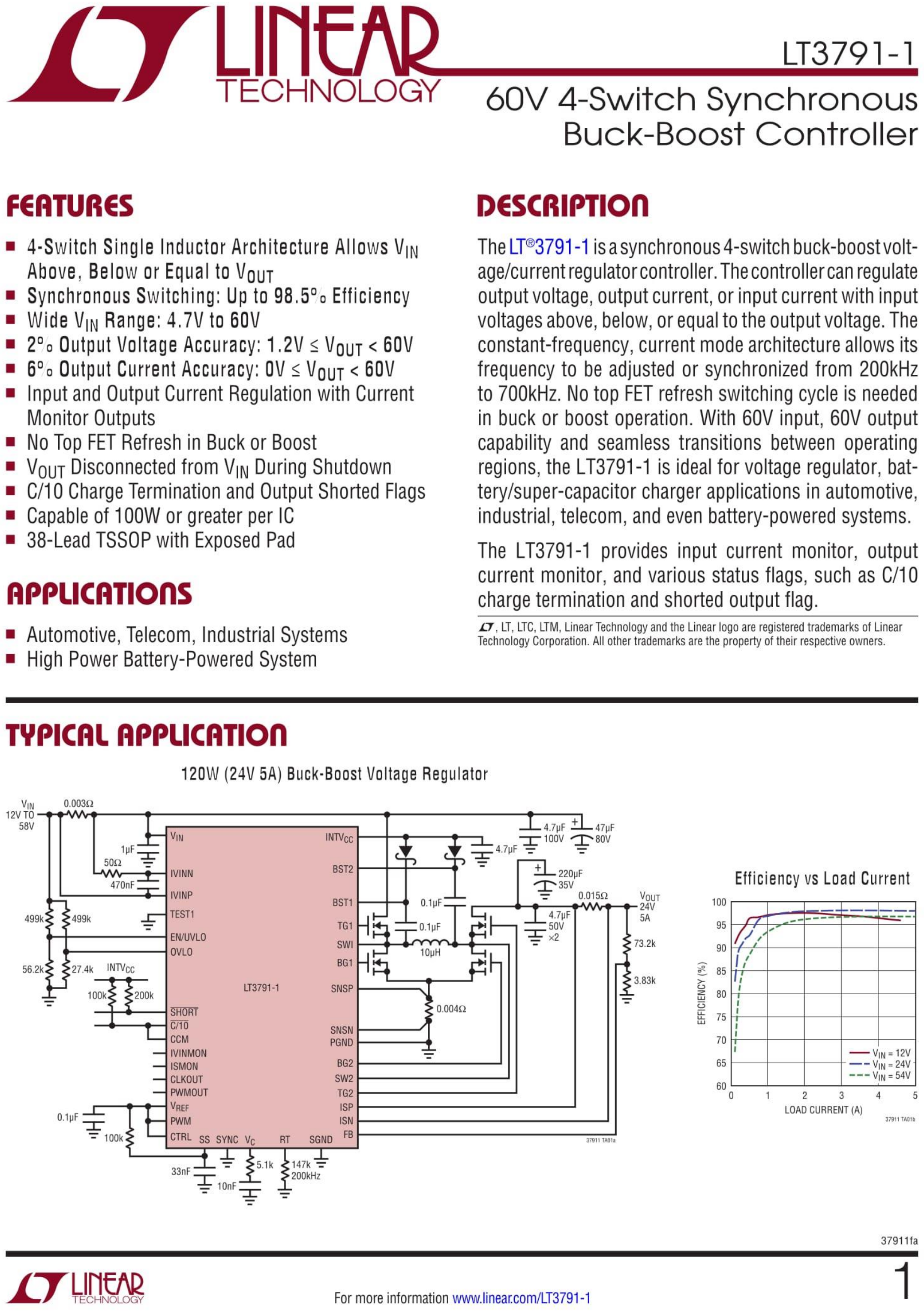


Figure 4. Reverse Current vs. Reverse Voltage

## 9.1.5 Linear Technology LT3791 Datasheet



LT3791-1

60V 4-Switch Synchronous Buck-Boost Controller

### FEATURES

- 4-Switch Single Inductor Architecture Allows $V_{IN}$ Above, Below or Equal to $V_{OUT}$
- Synchronous Switching: Up to 98.5% Efficiency
- Wide $V_{IN}$ Range: 4.7V to 60V
- 2% Output Voltage Accuracy: $1.2V \le V_{OUT} < 60V$
- 6% Output Current Accuracy: $0V \le V_{OUT} < 60V$
- Input and Output Current Regulation with Current Monitor Outputs
- No Top FET Refresh in Buck or Boost
- $V_{OUT}$ Disconnected from $V_{IN}$ During Shutdown
- C/10 Charge Termination and Output Shorted Flags
- Capable of 100W or greater per IC
- 38-Lead TSSOP with Exposed Pad

### APPLICATIONS

- Automotive, Telecom, Industrial Systems
- High Power Battery-Powered System

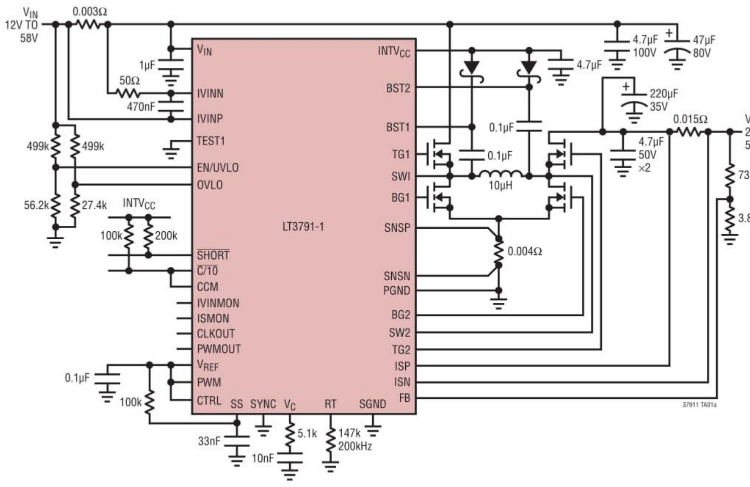
### DESCRIPTION

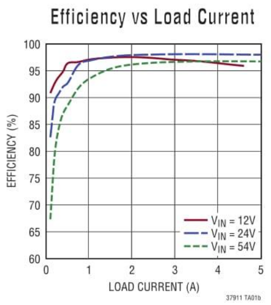
The LT®3791-1 is a synchronous 4-switch buck-boost voltage/current regulator controller. The controller can regulate output voltage, output current, or input current with input voltages above, below, or equal to the output voltage. The constant-frequency, current mode architecture allows its frequency to be adjusted or synchronized from 200kHz to 700kHz. No top FET refresh switching cycle is needed in buck or boost operation. With 60V input, 60V output capability and seamless transitions between operating regions, the LT3791-1 is ideal for voltage regulator, battery/super-capacitor charger applications in automotive, industrial, telecom, and even battery-powered systems.

The LT3791-1 provides input current monitor, output current monitor, and various status flags, such as C/10 charge termination and shorted output flag.

### TYPICAL APPLICATION

120W (24V 5A) Buck-Boost Voltage Regulator





37911fa

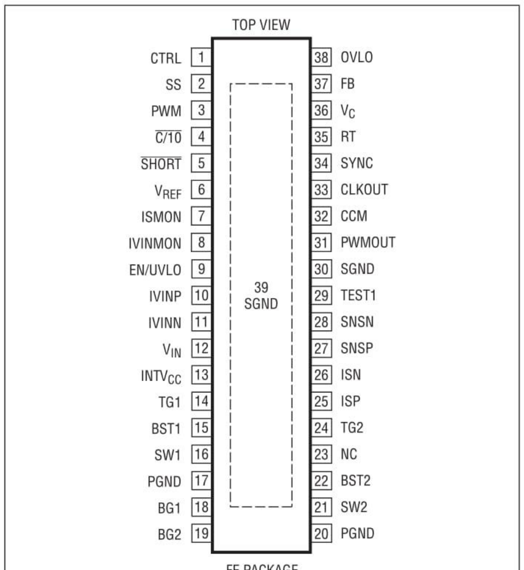
## ABSOLUTE MAXIMUM RATINGS

(Note 1)

Supply Voltages

| | |
|---|---|
| Input Supply ($V_{IN}$) | 60V |
| SW1, SW2 | –1V to 60V |
| $\overline{C/10}$, $\overline{SHORT}$ | 15V |
| EN/UVLO, IVINP, IVINN, ISP, ISN | 60V |
| $INTV_{CC}$, (BST1-SW1), (BST2-SW2) | 6V |
| CCM, SYNC, RT, CTRL, OVLO, PWM | 6V |
| IVINMON, ISMON, FB, SS, VC, $V_{REF}$ | 6V |
| IVINP-IVINN, ISP-ISN, SNSP-SNSN | ±0.5V |
| SNSP, SNSN | ±0.3V |
| Operating Junction Temperature (Notes 2, 3) | |
| LT3791E-1/LT3791I-1 | –40°C to 125°C |
| LT3791H-1 | –40°C to 150°C |
| LT3791MP-1 | –55°C to 150°C |
| Storage Temperature Range | –65°C to 150°C |
| Lead Temperature (Soldering, 10 sec) | 300°C |

## PIN CONFIGURATION



FE PACKAGE
38-LEAD PLASTIC TSSOP
$T_{JMAX}$ = 150°C, $\theta_{JA}$ = 28°C/W
EXPOSED PAD (PIN 39) IS SGND, MUST BE SOLDERED TO PCB

## ORDER INFORMATION

| LEAD FREE FINISH | TAPE AND REEL | PART MARKING* | PACKAGE DESCRIPTION | TEMPERATURE RANGE |
|---|---|---|---|---|
| LT3791EFE-1#PBF | LT3791EFE-1#TRPBF | LT3791FE-1 | 38-Lead Plastic TSSOP | –40°C to 125°C |
| LT3791IFE-1#PBF | LT3791IFE-1#TRPBF | LT3791FE-1 | 38-Lead Plastic TSSOP | –40°C to 125°C |
| LT3791HFE-1#PBF | LT3791HFE-1#TRPBF | LT3791FE-1 | 38-Lead Plastic TSSOP | –40°C to 150°C |
| LT3791MPFE-1#PBF | LT3791MPFE-1#TRPBF | LT3791FE-1 | 38-Lead Plastic TSSOP | –55°C to 150°C |

Consult LTC Marketing for parts specified with wider operating temperature ranges. *The temperature grade is identified by a label on the shipping container.

For more information on lead free part marking, go to: http://www.linear.com/leadfree/
For more information on tape and reel specifications, go to: http://www.linear.com/tapeandreel/

## ELECTRICAL CHARACTERISTICS

The ● denotes the specifications which apply over the full operating junction temperature range, otherwise specifications are at $T_A$ = 25°C (Note 2). $V_{IN}$ = 12V, $V_{EN/UVLO}$ = 12V unless otherwise noted.

| PARAMETER | CONDITIONS | | MIN | TYP | MAX | UNITS |
|---|---|---|---|---|---|---|
| Input | | | | | | |
| $V_{IN}$ Operating Voltage | | | 4.7 | | 60 | V |
| $V_{IN}$ Shutdown $I_Q$ | $V_{EN/UVLO}$ = 0V | | | 0.1 | 1 | µA |
| $V_{IN}$ Operating $I_Q$ (Not Switching) | FB = 1.3V, $R_T$ = 59.0k | | | 3.0 | 4 | mA |

## PIN FUNCTIONS

**CTRL (Pin 1):** Output Current Sense Threshold Adjustment Pin. Regulating threshold $V_{(ISP-ISN)}$ is 1/10th of ($V_{CTRL}$ – 200mV). CTRL linear range is from 200mV to 1.1V. For $V_{CTRL}$ > 1.3V, the current sense threshold is constant at the full-scale value of 100mV. For 1.1V < $V_{CTRL}$ < 1.3V, the dependence of the current sense threshold upon $V_{CTRL}$ transitions from a linear function to a constant value, reaching 98% of full scale by $V_{CTRL}$ = 1.2V. Connect CTRL to $V_{REF}$ for the 100mV default threshold. Force less than 175mV (typical) to stop switching. Do not leave this pin open.

**SS (Pin 2):** Soft-start reduces the input power sources surge current by gradually increasing the controller's current limit. A minimum value of 22nF is recommended on this pin. A 100k resistor must be placed between SS and $V_{REF}$ for the LT3791-1.

**PWM (Pin 3):** A signal low turns off switches, idles switching and disconnects the $V_C$ pin from all external loads. The PWMOUT pin follows the PWM pin. PWM has an internal 90k pull-down resistor. If not used, connect to $INTV_{CC}$.

**$\overline{C/10}$ (Pin 4):** C/10 Charge Termination Pin. An open-drain pull-down on $\overline{C/10}$ asserts if FB is greater than 1.15V (typical) and $V_{(ISP-ISN)}$ is less than 10mV (typical). To function, the pin requires an external pull-up resistor.

**$\overline{SHORT}$ (Pin 5):** Output Shorted Pin. An open-drain pull-down on $\overline{SHORT}$ asserts if FB is less than 400mV (typical). To function, the pin requires an external pull-up resistor.

**$V_{REF}$ (Pin 6):** Voltage Reference Output Pin, Typically 2V. This pin drives a resistor divider for the CTRL pin, either for output current adjustment or for temperature limit/compensation of the output load. Can supply up to 200µA of current.

**ISMON (Pin 7):** Monitor pin that produces a voltage that is ten times the voltage $V_{(ISP-ISN)}$. ISMON will equal 1V when $V_{(ISP-ISN)}$ = 100mV.

**IVINMON (Pin 8):** Monitor pin that produces a voltage that is twenty times the voltage $V_{(IVINP-IVINN)}$. IVINMON will equal 1V when $V_{(IVINP-IVINN)}$ = 50mV.

**EN/UVLO (Pin 9):** Enable Control Pin. Forcing an accurate 1.2V falling threshold with an externally programmable hysteresis is generated by the external resistor divider and a 3µA pull-down current. Above the 1.2V (typical) threshold (but below 6V), EN/UVLO input bias current is sub-µA. Below the falling threshold, a 3µA pull-down current is enabled so the user can define the hysteresis with the external resistor selection. An undervoltage condition resets soft-start. Tie to 0.3V, or less, to disable the device and reduce $V_{IN}$ quiescent current below 1µA.

**IVINP (Pin 10):** Positive Input for the Input Current Limit and Monitor. Input bias current for this pin is typically 90µA.

**IVINN (Pin 11):** Negative Input for the Input Current Limit and Monitor. The input bias current for this pin is typically 20µA.

**$V_{IN}$ (Pin 12):** Main Input Supply. Bypass this pin to PGND with a capacitor.

**$INTV_{CC}$ (Pin 13):** Internal 5V Regulator Output. The driver and control circuits are powered from this voltage. Bypass this pin to PGND with a minimum 4.7µF ceramic capacitor.

**TG1 (Pin 14):** Top Gate Drive. Drives the top N-channel MOSFET with a voltage equal to $INTV_{CC}$ superimposed on the switch node voltage SW1.

**BST1 (Pin 15):** Bootstrapped Driver Supply. The BST1 pin swings from a diode voltage below $INTV_{CC}$ up to a diode voltage below $V_{IN}$ + $INTV_{CC}$.

**SW1 (Pin 16):** Switch Node. SW1 pin swings from a diode voltage drop below ground up to $V_{IN}$.

**PGND (Pins 17, 20):** Power Ground. Connect these pins closely to the source of the bottom N-channel MOSFET.

**BG1 (Pin 18):** Bottom Gate Drive. Drives the gate of the bottom N-channel MOSFET between ground and $INTV_{CC}$.

**BG2 (Pin 19):** Bottom Gate Drive. Drives the gate of the bottom N-channel MOSFET between ground and $INTV_{CC}$.

**SW2 (Pin 21):** Switch Node. SW2 pin swings from a diode voltage drop below ground up to $V_{OUT}$.

## PIN FUNCTIONS

**BST2 (Pin 22):** Bootstrapped Driver Supply. The BST2 pin swings from a diode voltage below $INTV_{CC}$ up to a diode voltage below $V_{OUT}$ + $INTV_{CC}$.

**NC (Pin 23):** No Connect Pin. Leave this pin floating.

**TG2 (Pin 24):** Top Gate Drive. Drives the top N-channel MOSFET with a voltage equal to $INTV_{CC}$ superimposed on the switch node voltage SW2.

**ISP (Pin 25):** Connection Point for the Positive Terminal of the Output Current Feedback Resistor.

**ISN (Pin 26):** Connection Point for the Negative Terminal of the Output Current Feedback Resistor.

**SNSP (Pin 27):** The Positive Input to the Current Sense Comparator. The $V_C$ pin voltage and controlled offsets between the SNSP and SNSN pins, in conjunction with a resistor, set the current trip threshold.

**SNSN (Pin 28):** The Negative Input to the Current Sense Comparator.

**TEST1 (Pin 29):** This pin is used for testing purposes only and must be connected to SGND for the part to operate properly.

**SGND (Pin 30, Exposed Pad Pin 39):** Signal Ground. All small-signal components and compensation should connect to this ground, which should be connected to PGND at a single point. Solder the exposed pad directly to the ground plane.

**PWMOUT (Pin 31):** Buffered Version of PWM Signal for Driving Output Load Disconnect N-Channel MOSFET. The PWMOUT pin is driven from $INTV_{CC}$. Use of a MOSFET with a gate cutoff voltage higher than 1V is recommended.

**CCM (Pin 32):** Continuous Conduction Mode Pin. When the pin voltage is higher than 1.5V, the part runs in fixed frequency forced continuous conduction mode and allows the inductor current to flow negative. When the pin voltage is less than 0.3V, the part runs in discontinuous conduction mode and does not allow the inductor current to flow backward. This pin is only meant to block inductor reverse current, and should only be pulled low when the output current is low. This pin must be either connected to $INTV_{CC}$ (pin 13) for continuous conduction mode across all loads, or it must be connected to the $\overline{C/10}$ (pin 4) with a pull-up resistor to $INTV_{CC}$ for continuous conduction mode at heavy load and for discontinuous conduction mode at light load.

**CLKOUT (Pin 33):** Clock Output Pin. A 180° out-of-phase clock is provided at the oscillator frequency to allow for paralleling two devices for extending output power capability.

**SYNC (Pin 34):** External Synchronization Input Pin. This pin is internally terminated to GND with a 90k resistor. The internal buck clock is synchronized to the rising edge of the SYNC signal while the internal boost clock is 180° phase shifted.

**RT (Pin 35):** Frequency Set Pin. Place a resistor to GND to set the internal frequency. The range of oscillation is 200kHz to 700kHz.

**$V_C$ (Pin 36):** Current Control Threshold and Error Amplifier Compensation Point. The current comparator threshold increases with this control voltage. The voltage ranges from 0.7V to 1.9V.

**FB (Pin 37):** Voltage Loop Feedback Pin. FB is intended for constant-voltage regulation. The internal transconductance amplifier with output $V_C$ will regulate FB to 1.2V (typical) through the DC/DC converter. If the FB input is regulating the loop and $V_{(ISP-ISN)}$ < 10mV, the $\overline{C/10}$ pull-down is asserted. If the FB pin is less than 400mV, the $\overline{SHORT}$ pull-down is asserted.

**OVLO (Pin 38):** Overvoltage Input Pin. This pin is used for OVLO, if OVLO > 3V then SS is pulled low, the part stops switching and resets. Do not leave this pin open.

## OPERATION

The LT3791-1 is a current mode controller that provides an output voltage above, equal to or below the input voltage. The LTC proprietary topology and control architecture uses a current sensing resistor in buck or boost operation. The sensed inductor current is controlled by the voltage on the $V_C$ pin, which is the output of the feedback amplifiers A11 and A12. The $V_C$ pin is controlled by three inputs, one input from the output current loop, one input from the input current loop, and the third input from the feedback loop. Whichever feedback input is higher takes precedence, forcing the converter into either a constant-current or a constant-voltage mode.

The LT3791-1 is designed to transition cleanly between the two modes of operation. Current sense amplifier A1 senses the voltage between the IVINP and IVINN pins and provides a pre-gain to amplifier A11. When the voltage between IVINP and IVINN reaches 50mV, the output of A1 provides IVINMON_INT to the inverting input of A11 and the converter is in constant-current mode. If the current sense voltage exceeds 50mV, the output of A1 increases causing the output of A11 to decrease, thus reducing the amount of current delivered to the output. In this manner the current sense voltage is regulated to 50mV.

The output current amplifier works similar to the input current amplifier but with a 100mV voltage instead of 50mV. The output current sense level is also adjustable by the CTRL pin. Forcing CTRL to less than 1.2V forces ISMON_INT to the same level as CTRL, thus providing current-level control. The output current amplifier provides rail-to-rail operation. Similarly if the FB pin goes above 1.2V the output of A11 decreases to reduce the current level and regulate the output (constant-voltage mode).

The LT3791-1 provides monitoring pins IVINMON and ISMON that are proportional to the voltage across the input and output current amplifiers respectively.

The main control loop is shut down by pulling the EN/UVLO pin low. When the EN/UVLO pin is higher than 1.2V, an internal 14μA current source charges soft-start capacitor $C_{SS}$ at the SS pin. The $V_C$ voltage is then clamped a diode voltage higher than the SS voltage while the $C_{SS}$ is slowly charged during start-up. This soft-start clamping prevents abrupt current from being drawn from the input power supply.

The top MOSFET drivers are biased from floating bootstrap capacitors C1 and C2, which are normally recharged through an external diode when the top MOSFET is turned off. A unique charge sharing technique eliminates top FET refresh switching cycle in buck or boost operation. Schottky diodes across the synchronous switch M4 and synchronous switch M2 are not required, but they do provide a lower drop during the dead time. The addition of the Schottky diode typically improves peak efficiency by 1% to 2% at 500kHz.

### Power Switch Control

Figure 1 shows a simplified diagram of how the four power switches are connected to the inductor, $V_{IN}$, $V_{OUT}$ and GND. Figure 2 shows the regions of operation for the LT3791-1 as a function of duty cycle D. The power switches are properly controlled so the transfer between regions is continuous. When $V_{IN}$ approaches $V_{OUT}$, the buck-boost region is reached.

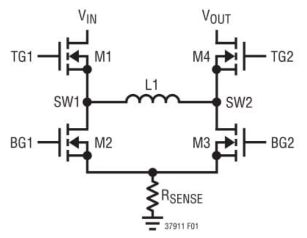
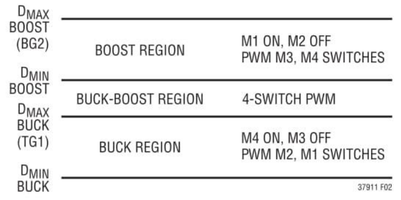


Figure 1. Simplified Diagram of the Output Switches



| | | |
|---|---|---|
| $D_{MAX}$ BOOST (BG2) | | |
| | BOOST REGION | M1 ON, M2 OFF PWM M3, M4 SWITCHES |
| $D_{MIN}$ BOOST | | |
| | BUCK-BOOST REGION | 4-SWITCH PWM |
| $D_{MAX}$ BUCK (TG1) | | |
| | BUCK REGION | M4 ON, M3 OFF PWM M2, M1 SWITCHES |
| $D_{MIN}$ BUCK | | |

Figure 2. Operating Regions vs Duty Cycle

## OPERATION

### Buck Region ($V_{IN} > V_{OUT}$)

Switch M4 is always on and switch M3 is always off during this mode. At the start of every cycle, synchronous switch M2 is turned on first. Inductor current is sensed when synchronous switch M2 is turned on. After the sensed inductor current falls below the reference voltage, which is proportional to $V_C$, synchronous switch M2 is turned off and switch M1 is turned on for the remainder of the cycle. Switches M1 and M2 will alternate, behaving like a typical synchronous buck regulator. The duty cycle of switch M1 increases until the maximum duty cycle of the converter in buck operation reaches $D_{MAX(BUCK,\ TG1)}$, given by:

$$D_{MAX(BUCK,TG1)} = 100\% - D_{(BUCK\text{-}BOOST)}$$

where $D_{(BUCK\text{-}BOOST)}$ is the duty cycle of the buck-boost switch range:

$$D_{(BUCK\text{-}BOOST)} = 8\%$$

Figure 3 shows typical buck operation waveforms. If $V_{IN}$ approaches $V_{OUT}$, the buck-boost region is reached.

### Buck-Boost Region ($V_{IN} \sim V_{OUT}$)

When $V_{IN}$ is close to $V_{OUT}$, the controller is in buck-boost operation. Figure 4 and Figure 5 show typical waveforms in this operation. Every cycle the controller turns on switches M2 and M4, then M1 and M4 are turned on until 180° later when switches M1 and M3 turn on, and then switches M1 and M4 are turned on for the remainder of the cycle.

### Boost Region ($V_{IN} < V_{OUT}$)

Switch M1 is always on and synchronous switch M2 is always off in boost operation. Every cycle switch M3 is turned on first. Inductor current is sensed when synchronous switch M3 is turned on. After the sensed inductor current exceeds the reference voltage which is proportional to $V_C$, switch M3 turns off and synchronous switch M4 is turned on for the remainder of the cycle. Switches M3 and M4 alternate, behaving like a typical synchronous boost regulator.

The duty cycle of switch M3 decreases until the minimum duty cycle of the converter in boost operation reaches $D_{MIN(BOOST,BG2)}$, given by:

$$D_{MIN(BOOST,BG2)} = D_{(BUCK\text{-}BOOST)}$$

where $D_{(BUCK\text{-}BOOST)}$ is the duty cycle of the buck-boost switch range:

$$D_{(BUCK\text{-}BOOST)} = 8\%$$

Figure 6 shows typical boost operation waveforms. If $V_{IN}$ approaches $V_{OUT}$, the buck-boost region is reached.

### Low Current Operation

The LT3791-1 is recommended to run in forced continuous conduction mode at heavy load by pulling the CCM pin higher than 1.5V. In this mode the controller behaves as a continuous, PWM current mode synchronous switching regulator. In boost operation, switch M1 is always on, switch M3 and synchronous switch M4 are alternately turned on to maintain the output voltage independent of the direction of inductor current. In buck operation, synchronous switch M4 is always on, switch M1 and synchronous switch M2 are alternately turned on to maintain the output voltage independent of the direction of inductor current. In the forced continuous mode, the output can source or sink current.

However, reverse inductor current from the output to the input is not desired for certain applications. For these applications, the CCM pin must be connected to $\overline{C/10}$ (pin 4) with a pull-up resistor to $INTV_{CC}$ (see front page Typical Application). Therefore, the CCM pin will be pulled lower than 0.3V for discontinuous conduction mode by the $\overline{C/10}$ pin when the output current is low. In this mode, switch M4 turns off when the inductor current flows negative.

## OPERATION

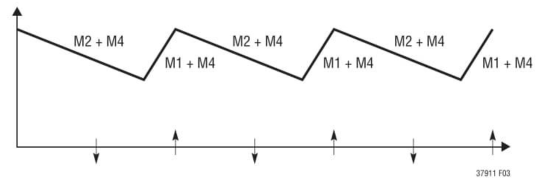


Figure 3. Buck Operation ($V_{IN} > V_{OUT}$)

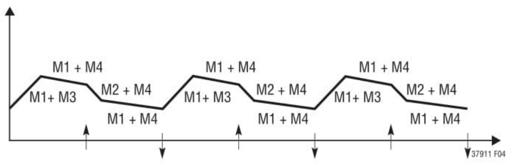


Figure 4. Buck-Boost Operation ($V_{IN} \leq V_{OUT}$)

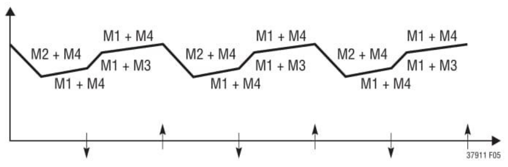


Figure 5. Buck-Boost Operation ($V_{IN} \geq V_{OUT}$)

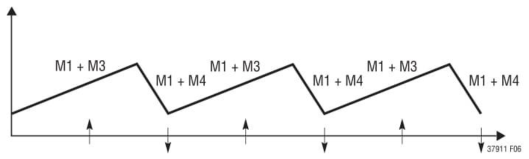


Figure 6. Boost Operation ($V_{IN} < V_{OUT}$)

## 9.1.6 Bourns PEC11 Datasheet

**Features**

- Push switch option
- Compact, rugged design
- High reliability
- Metal bushing/shaft

BOURNS® PRO AUDIO

**PEC11 Series - 12 mm Incremental Encoder**

### Electrical Characteristics

| | |
|---|---|
| Output | 2-bit quadrature code |
| Closed Circuit Resistance | 3 ohms maximum |
| Contact Rating | 1 mA @ 5 VDC |
| Insulation Resistance | 100 megohms @ 250 VDC |
| Dielectric Withstanding Voltage | |
| Sea Level | 300 VAC minimum |
| Electrical Travel | Continuous |
| Contact Bounce (15 RPM) | 5.0 ms maximum** |
| RPM (Operating) | 60 maximum** |

### Environmental Characteristics

| | |
|---|---|
| Operating Temperature Range | -30 °C to +70 °C (-22 °F to +158 °F) |
| Storage Temperature Range | -40 °C to +85 °C (-40 °F to +185 °F) |
| Humidity | MIL-STD-202, Method 103B, Condition B |
| Vibration | 30 G |
| Contact Bounce | 10~55~10 Hz / 1 min. / Amplitude 1.5 mm |
| Shock | 100 G |
| Rotational Life | 30,000 cycles minimum |
| Switch Life | 20,000 cycles minimum |
| IP Rating | IP 40 |

### Mechanical Characteristics

| | |
|---|---|
| Mechanical Angle | 360 ° continuous |
| Torque | |
| Running | 50 to 200 gf.cm (0.68 to 2.7 oz.-in.) |
| Mounting | 10.2 kgf.cm (8.83 lb.-in.) maximum |
| Shaft Side Load (Static) | 2.04 kgf (4.5 lbs.) minimum |
| Weight | 5 gm (0.17 oz.) maximum |
| Terminals | Printed circuit board terminals |
| Soldering Condition | |
| Wave Soldering | Sn95.5/Ag2.8/Cu0.7 solder with no-clean flux: 260 °C max. for 3-5 seconds |
| Hand Soldering | Not recommended |
| Hardware | One flat washer and one mounting nut supplied with each encoder |

### Switch Characteristics

| | |
|---|---|
| Switch Type | Contact Push ON Momentary SPST |
| Power Rating (Resistive Load) | 10 mA at 5 V DC |
| Switch Travel | 0.5 ± 0.2 mm |
| Switch Actuation Force | 610 ± 306 gf (8.47 ± 4.24 oz.-in.) |

### How To Order

**PEC11 - 4 0 20 F - S 0012**

Model

Terminal Configuration
- 4 = PC Pin Horizontal/Rear Facing

Detent Option
- 0 = No Detents (12, 18, 24 pulses)
- 1 = 18 Detents (18 pulses)
- 2 = 24 Detents (12, 24 pulses)
- 3 = 12 Detents (12 pulses)

Standard Shaft Length
- 15 = 15.0 mm
- 20 = 20.0 mm
- 25 = 25.0 mm
- 30 = 30.0 mm

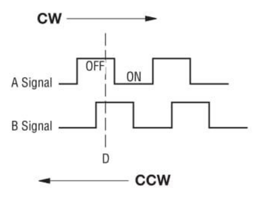
Shaft Style
- F = Metal Flatted Shaft
- K = Metal Knurled Shaft[1]

Switch Configuration
- S = Push Momentary Switch
- N = No Switch

Resolution
- 0012 = 12 Pulses per 360 ° Rotation
- 0018 = 18 Pulses per 360 ° Rotation
- 0024 = 24 Pulses per 360 ° Rotation

[1] Metal knurled shaft with no switch is available in 15, 20 and 30 mm shaft lengths.
Metal knurled shaft with push momentary switch is available in 15 and 20 mm shaft lengths.

### Quadrature Output Table



*RoHS Directive 2002/95/EC Jan 27, 2003 including Annex.
**Devices are tested using standard noise reduction filters. For optimum performance, designers should use noise reduction filters in their circuits.
Specifications are subject to change without notice.
Customers should verify actual device performance in their specific applications.

## Applications

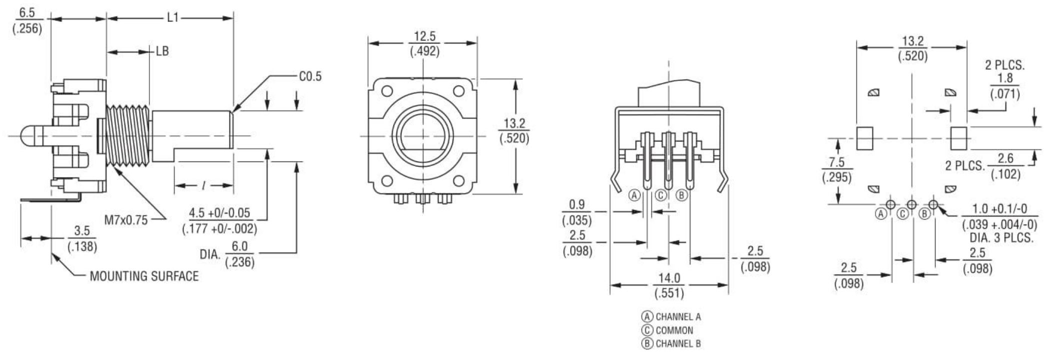
Level control, tuning and timer settings in:

- Audio-visual equipment
- Consumer electric appliances
- Radios
- Musical instrumentation
- Communications equipment

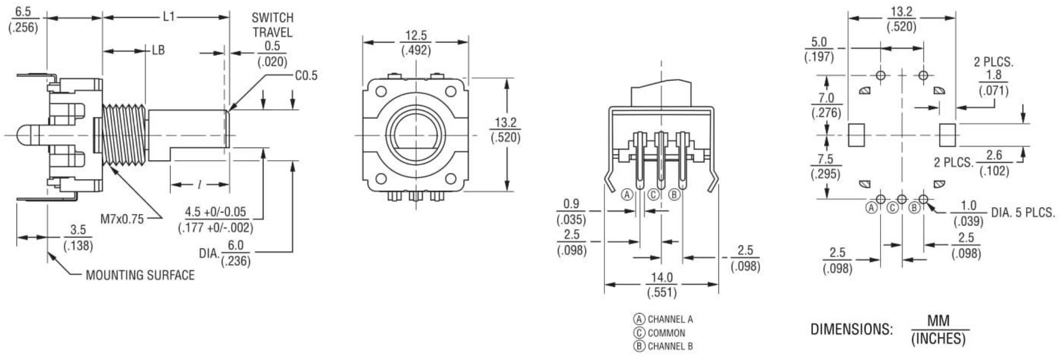
## PEC11 Series - 12 mm Incremental Encoder

### Product Dimensions

**PEC11-4xxxF-Nxxxx**



**PEC11-4xxxF-Sxxxx**



| L1 | LB | *l* |
|---|---|---|
| 15 (.591) | 5.0 (.197) | 7.0 (.276) |
| 20 (.787) | 7.0 (.276) | 10.0 (.394) |
| 25 (.984) | 7.0 (.276) | 12.0 (.472) |
| 30 (1.181) | 7.0 (.276) | 12.0 (.472) |

Specifications are subject to change without notice.
Customers should verify actual device performance in their specific applications.

## 9.1.7 ROHM Semiconductor 2SD2444K Datasheet

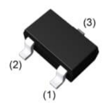
# 2SD2444K

Power Transistor (15V,1A) Datasheet

| Parameter | Value |
|---|---|
| $V_{CEO}$ | 15V |
| $I_C$ | 1A |

●**Outline**

SMT3

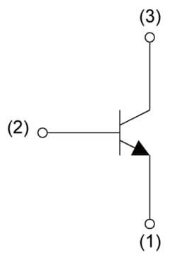


SOT-346
SC-59

●**Features**

1)Low saturation voltage,
$V_{CE(sat)}$=0.3V(Max.) at $I_C/I_B$=400mA/20mA
2)$I_C$=1A.
3)Complements the 2SB1590K.

●**Inner circuit**



(1) Emitter
(2) Base
(3) Collector

●**Application**

LOW FREQUENCY POWER AMPLIFIER

●**Packaging specifications**

| Part No. | Package | Package size | Taping code | Reel size (mm) | Tape width (mm) | Basic ordering unit.(pcs) | Marking |
|---|---|---|---|---|---|---|---|
| 2SD2444K | SMT3 | 2928 | T146 | 180 | 8 | 3000 | BS |

●**Absolute maximum ratings** ($T_a$ = 25°C)

| Parameter | Symbol | Values | Unit |
|---|---|---|---|
| Collector-base voltage | $V_{CBO}$ | 15 | V |
| Collector-emitter voltage | $V_{CEO}$ | 15 | V |
| Emitter-base voltage | $V_{EBO}$ | 6 | V |
| Collector current | $I_C$ | 1 | A |
| | $I_{CP}$ *1 | 2 | A |
| Power dissipation | $P_D$ *2 | 200 | mW |
| Junction temperature | $T_j$ | 150 | ℃ |
| Range of storage temperature | $T_{stg}$ | -55 to +150 | ℃ |

●**Electrical characteristics** ($T_a$ = 25°C)

| Parameter | Symbol | Conditions | Values | | | Unit |
|---|---|---|---|---|---|---|
| | | | Min. | Typ. | Max. | |
| Collector-base breakdown voltage | $BV_{CBO}$ | $I_C$ = 50μA | 15 | - | - | V |
| Collector-emitter breakdown voltage | $BV_{CEO}$ | $I_C$ = 1mA | 15 | - | - | V |
| Emitter-base breakdown voltage | $BV_{EBO}$ | $I_E$ = 50μA | 6 | - | - | V |
| Collector cut-off current | $I_{CBO}$ | $V_{CB}$ = 12V | - | - | 500 | nA |
| Emitter cut-off current | $I_{EBO}$ | $V_{EB}$ = 5V | - | - | 500 | nA |
| Collector-emitter saturation voltage | $V_{CE(sat)}$ | $I_C$ = 400mA, $I_B$ = 20mA | - | - | 300 | mV |
| DC current gain | $h_{FE}1$ | $V_{CE}$ = 2V, $I_C$ = 50mA | 180 | - | 390 | - |
| | $h_{FE}2$ | $V_{CE}$ = 2V, $I_C$ = 800mA | 80 | - | - | |
| Transition frequency | $f_T$ | $V_{CE}$ = 2V, $I_E$ = -50mA, f = 100MHz | - | 200 | - | MHz |
| Output capacitance | $C_{ob}$ | $V_{CB}$ = 10V, $I_E$ = 0A, f = 1MHz | - | 15 | - | pF |

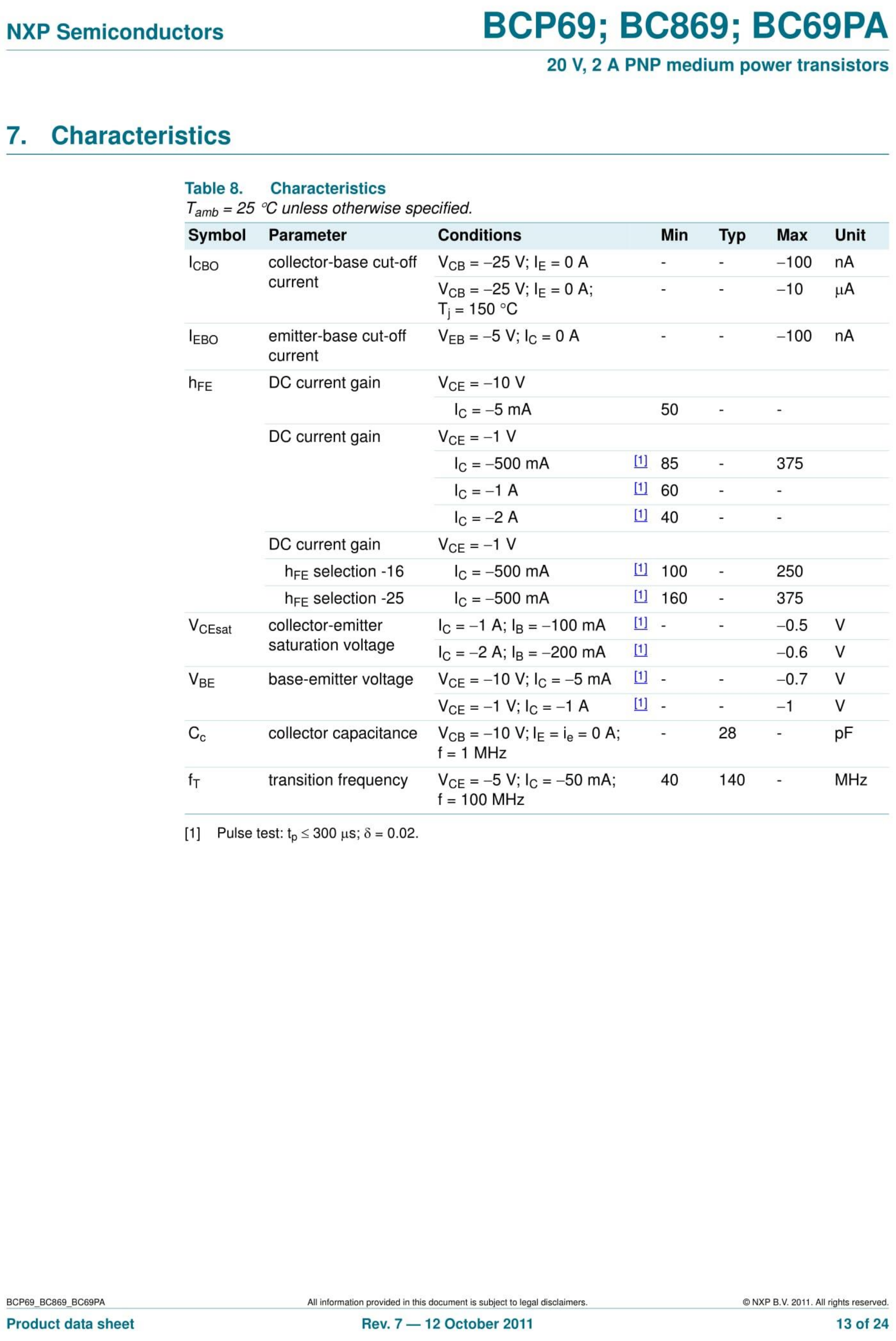
hFE values are calssified as follows :

| rank | R | - | - | - | - |
|---|---|---|---|---|---|
| $h_{FE}1$ | 180-390 | - | - | - | - |

*1 Pw=10ms Single Pulse

*2 Each terminal mounted on a reference land.

## 9.1.8 NXP Semiconductors BCP69 Datasheet



## 7. Characteristics

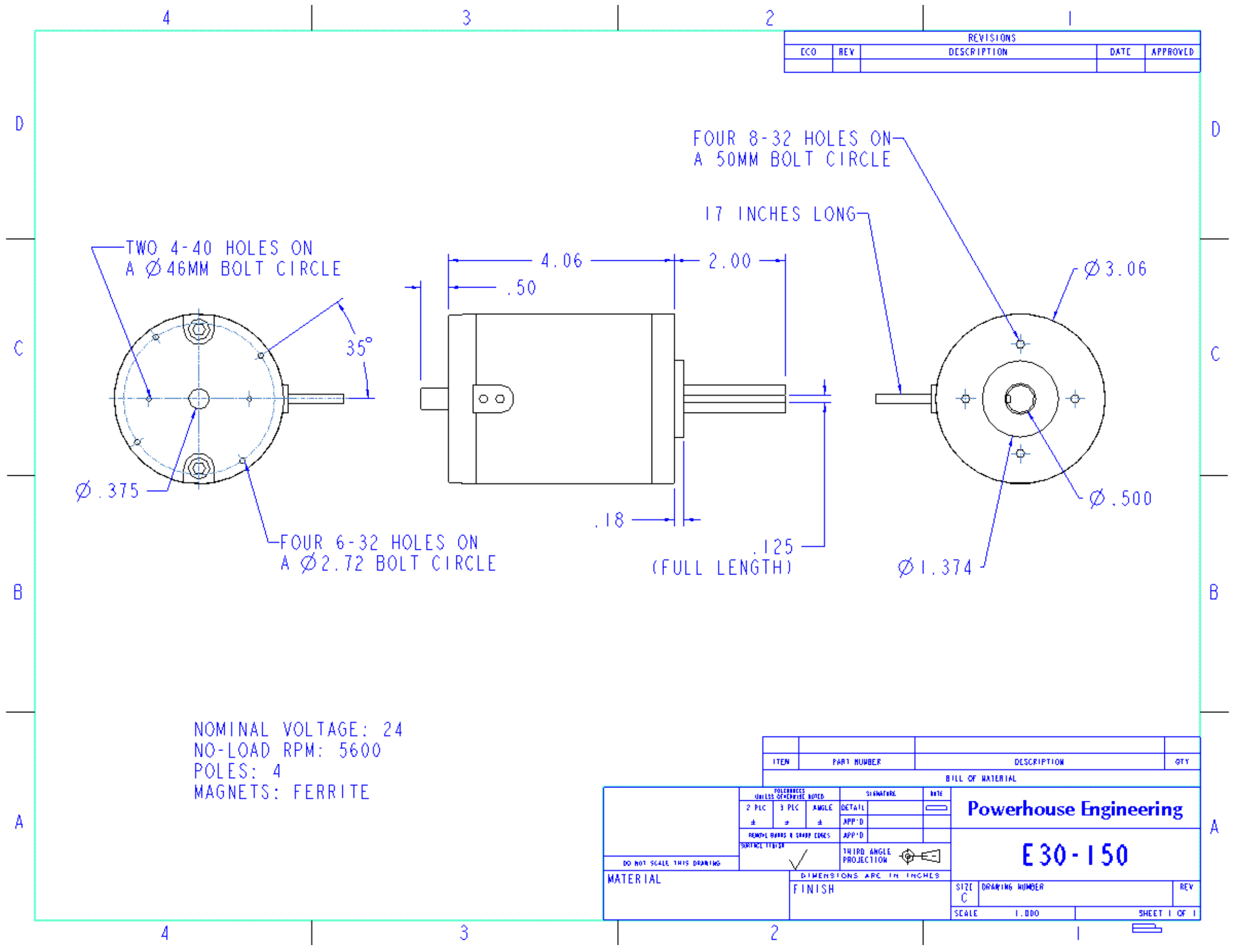
Table 8. Characteristics

*$T_{amb}$ = 25 °C unless otherwise specified.*

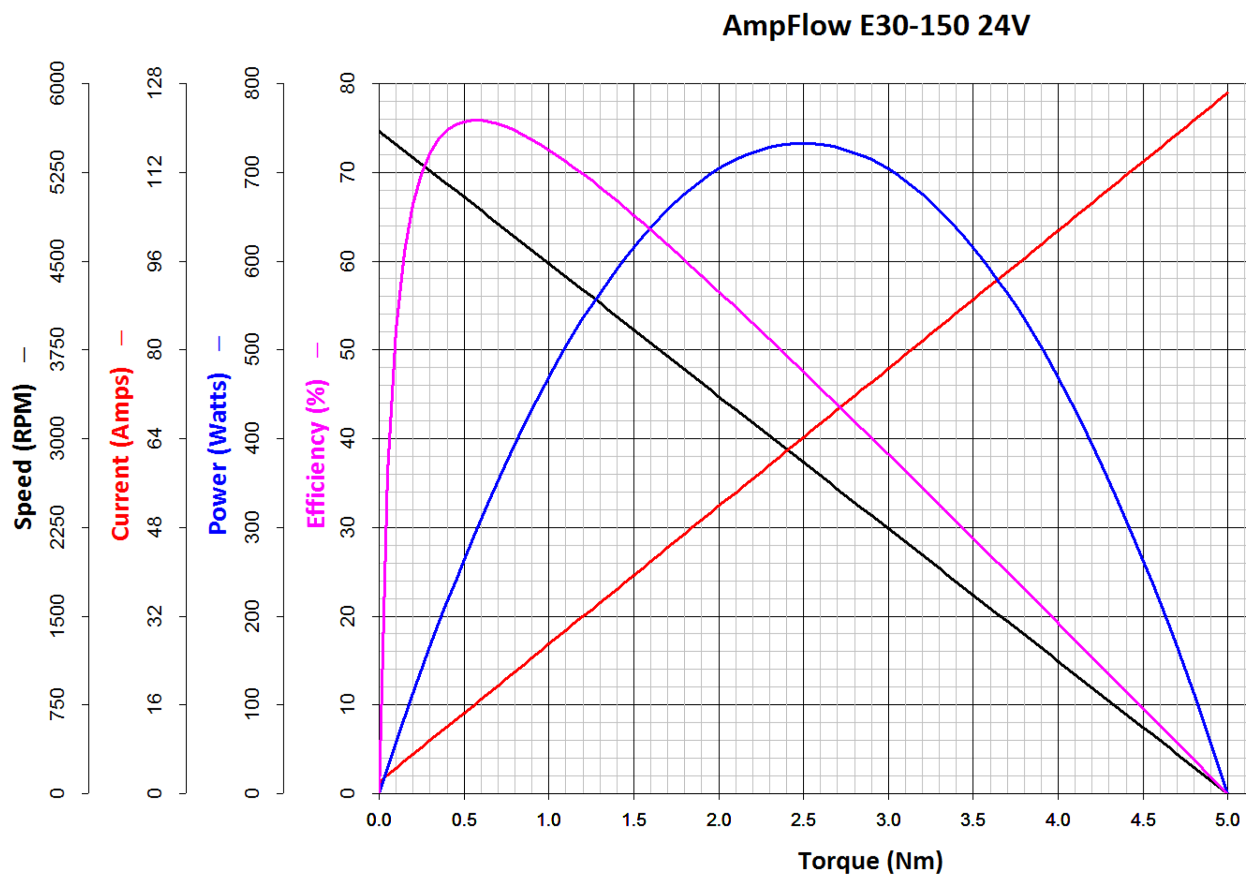
| Symbol | Parameter | Conditions | | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|---|
| $I_{CBO}$ | collector-base cut-off current | $V_{CB}$ = −25 V; $I_E$ = 0 A | | - | - | −100 | nA |
| | | $V_{CB}$ = −25 V; $I_E$ = 0 A; $T_j$ = 150 °C | | - | - | −10 | μA |
| $I_{EBO}$ | emitter-base cut-off current | $V_{EB}$ = −5 V; $I_C$ = 0 A | | - | - | −100 | nA |
| $h_{FE}$ | DC current gain | $V_{CE}$ = −10 V | | | | | |
| | | $I_C$ = −5 mA | | 50 | - | - | |
| | DC current gain | $V_{CE}$ = −1 V | | | | | |
| | | $I_C$ = −500 mA | [1] | 85 | - | 375 | |
| | | $I_C$ = −1 A | [1] | 60 | - | - | |
| | | $I_C$ = −2 A | [1] | 40 | - | - | |
| | DC current gain | $V_{CE}$ = −1 V | | | | | |
| | $h_{FE}$ selection -16 | $I_C$ = −500 mA | [1] | 100 | - | 250 | |
| | $h_{FE}$ selection -25 | $I_C$ = −500 mA | [1] | 160 | - | 375 | |
| $V_{CEsat}$ | collector-emitter saturation voltage | $I_C$ = −1 A; $I_B$ = −100 mA | [1] | - | - | −0.5 | V |
| | | $I_C$ = −2 A; $I_B$ = −200 mA | [1] | | | −0.6 | V |
| $V_{BE}$ | base-emitter voltage | $V_{CE}$ = −10 V; $I_C$ = −5 mA | [1] | - | - | −0.7 | V |
| | | $V_{CE}$ = −1 V; $I_C$ = −1 A | [1] | - | - | −1 | V |
| $C_c$ | collector capacitance | $V_{CB}$ = −10 V; $I_E$ = $i_e$ = 0 A; f = 1 MHz | | - | 28 | - | pF |
| $f_T$ | transition frequency | $V_{CE}$ = −5 V; $I_C$ = −50 mA; f = 100 MHz | | 40 | 140 | - | MHz |

[1] Pulse test: $t_p \leq 300$ μs; δ = 0.02.

## 9.1.9 AmpFlow E30-150 Schematic



### 9.1.10 AmpFlow E30-150 Characteristics

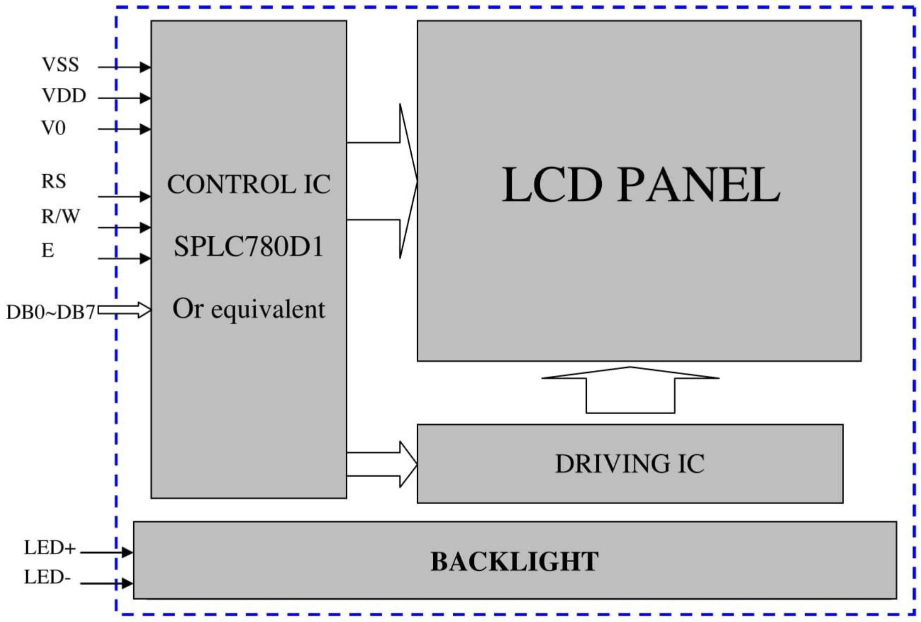


## 9.1.11 TINSHARP TC1602A-09T Datasheet

### FUNCTIONS & FEATURES

| | |
|---|---|
| Construction | : COB(Chip-on-Board) |
| Display Format | : 16x2 Characters |
| Display Type | : STN, Transmissive, Negative, Blue |
| Controller | : SPLC780D1 or equivalent controller |
| Interface | : 4-bit\8-bit parallel interface |
| Backlight | : white\side lights |
| Viewing Direction | : 6 O 'clock |
| Driving Scheme | : 1/16 Duty Cycle, 1/5 Bias |
| Power Supply Voltage | : 5.0 V |
| $V_{LCD}$ Adjustable For Best Contrast | : 5.0 V ($V_{OP}$.) |
| Operation temperature | : -10 ℃ to +60 ℃ |
| Storage temperature | : -20 ℃ to +70 ℃ |

### BLOCK DIAGRAM



## INTERFACE PIN FUNCTIONS

| Pin No. | Symbol | Level | Description |
|---|---|---|---|
| 1 | VSS | 0V | Ground. |
| 2 | VDD | +5.0V | Power supply for logic operating. |
| 3 | V0 | -- | Adjusting supply voltage for LCD driving. |
| 4 | RS | H/L | A signal for selecting registers:<br>1: Data Register (for read and write)<br>0: Instruction Register (for write), Busy flag-Address Counter (for read). |
| 5 | R/W | H/L | R/W = “H”: Read mode.<br>R/W = “L”: Write mode. |
| 6 | E | H/L | An enable signal for writing or reading data. |
| 7 | DB0 | H/L | This is an 8-bit bi-directional data bus. |
| 8 | DB1 | H/L | |
| 9 | DB2 | H/L | |
| 10 | DB3 | H/L | |
| 11 | DB4 | H/L | |
| 12 | DB5 | H/L | |
| 13 | DB6 | H/L | |
| 14 | DB7 | H/L | |
| 15 | LED+ | +5.0V | Power supply for backlight. |
| 16 | LED- | 0V | The backlight ground. |

## ABSOLUTE MAXIMUM RATINGS ( Ta = 25℃ )

| Parameter | Symbol | Min | Max | Unit |
|---|---|---|---|---|
| Supply voltage for logic | $V_{DD}$ | -0.3 | +7.0 | V |
| Supply voltage for LCD | $V_o$ | 0 | $V_{DD}$ +0.3 | V |
| Input voltage | $V_I$ | -0.3 | $V_{DD}$ +0.3 | V |
| Normal Operating temperature | $T_{OP}$ | -20 | +70 | ℃ |
| Normal Storage temperature | $T_{ST}$ | -30 | +80 | ℃ |

**Note:** Stresses beyond those given in the Absolute Maximum Rating table may cause operational errors or damage to the device. For normal operational conditions see AC/DC Electrical Characteristics.

## DC ELECTRICAL CHARACTERISTICS

| Parameter | Symbol | Condition | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| Supply voltage for logic | VDD | -- | 4.8 | 5.0 | 5.2 | V |
| Supply current for logic | IDD | -- | -- | 35 | 40 | mA |
| Operating voltage for LCD | VLCD | -10℃ | | | | |
| | | 25℃ | 4.8 | 5.0 | 5.2 | V |
| | | +60℃ | | | | |
| Input voltage “H” level | VIH | -- | 0.7 VDD | -- | VDD+0.3 | V |
| Input voltage “L” level | VIL | -- | 0 | -- | 0.2VDD | V |

## LED BACKLIGHT CHARACTERISTICS

| COLOR | Wavelength λ p(nm) | Operating Voltage(±0.15V) | Spectral line half width Δ λ (nm) | Forward Current (mA) |
|---|---|---|---|---|
| white | -- | 3.2 | -- | 30 |

**NOTE:** Do not connect +5V directly to the backlight terminals. This will ruin the backlight.

## 9.1.12 Panasonic HY1-4.5V Datasheet

**Panasonic** Automation Controls Catalog

**Non-polarized 1 Form C relay that realizes nominal operating power of 150 mW**

# HY RELAYS

RoHS compliant

### FEATURES

1. **Nominal operating power: High sensitivity of 150mW (Single side stable type)**
   A nominal operating power of 150 mW (minimum operating power of 84 mW) has been achieved.
2. **The use of gold-clad twin contacts ensures high contact reliability.**
3. **Sealed construction**

### TYPICAL APPLICATIONS

1. **Telecommunications equipment**
2. **Security equipment**
3. **Test and Measurement equipment**
4. **Consumer electronic and Audio visual equipment**

### ORDERING INFORMATION

**HY** 1 ☐ - ☐

Contact arrangement
1: 1 Form C

Sensitivity
Nil: High sensitivity 150 mW
Z: Standard 200 mW

Nominal coil voltage (DC)
1.5, 3, 4.5, 5, 6, 9, 12, 24 V

Note: In case of 5 V drive circuit, it is recommended to use 4.5 V type relay.

### TYPES

| Contact arrangement | Nominal coil voltage | 150mW type | 200mW type |
|---|---|---|---|
| | | Part No. | Part No. |
| 1 Form C | 1.5V DC | HY1-1.5V | HY1Z-1.5V |
| | 3V DC | HY1-3V | HY1Z-3V |
| | 4.5V DC | HY1-4.5V | HY1Z-4.5V |
| | 5V DC | HY1-5V | HY1Z-5V |
| | 6V DC | HY1-6V | HY1Z-6V |
| | 9V DC | HY1-9V | HY1Z-9V |
| | 12V DC | HY1-12V | HY1Z-12V |
| | 24V DC | HY1-24V | HY1Z-24V |

Standard packing: Tube: 50 pcs.; Case: 2,000 pcs.

## RATING

### 1. Coil data

| Contact arrangement | Nominal coil voltage | Pick-up voltage (at 20°C 68°F) | Drop-out voltage (at 20°C 68°F) | Nominal operating current [±10%] (at 20°C 68°F) | Coil resistance [±10%] (at 20°C 68°F) | Nominal operating power | Max. applied voltage (at 70°C 158°F) |
|---|---|---|---|---|---|---|---|
| 1 Form C | 1.5V DC | 75%V or less of nominal voltage (Initial) | 10%V or more of nominal voltage (Initial) | 100mA | 15Ω | 150mW | 140%V of nominal voltage |
| | 3V DC | | | 50mA | 60Ω | | |
| | 4.5V DC | | | 33.3mA | 135Ω | | |
| | 5V DC | | | 30mA | 166Ω | | |
| | 6V DC | | | 25mA | 240Ω | | |
| | 9V DC | | | 16.7mA | 540Ω | | |
| | 12V DC | | | 12.5mA | 960Ω | | |
| | 24V DC | | | 6.25mA | 3,840Ω | | |
| | 1.5V DC | 75%V or less of nominal voltage (Initial) | 10%V or more of nominal voltage (Initial) | 133.3mA | 11.25Ω | 200mW | 120%V of nominal voltage |
| | 3V DC | | | 66.7mA | 45Ω | | |
| | 4.5V DC | | | 44.5mA | 101.2Ω | | |
| | 5V DC | | | 40mA | 125Ω | | |
| | 6V DC | | | 33.3mA | 180Ω | | |
| | 9V DC | | | 22.2mA | 405Ω | | |
| | 12V DC | | | 16.7mA | 720Ω | | |
| | 24V DC | | | 8.3mA | 2,880Ω | | |

### 2. Specifications

| Characteristics | Item | | Specifications |
|---|---|---|---|
| Contact | Arrangement | | 1 Form C |
| | Initial contact resistance, max. | | Max. 100 mΩ (By voltage drop 6 V DC 1A) |
| | Contact material | | Ag+Au clad |
| Rating | Nominal switching capacity | | 1 A 30 V DC (resistive load) |
| | Max. switching power | | 30 W (DC) (resistive load) |
| | Max. switching voltage | | 60 V DC |
| | Max. carrying current | | 2 A |
| | Max. switching current | | 1 A (30 V DC) |
| | Min. switching capacity (Reference value)*1 | | 1mA 1 V DC |
| | Nominal operating power | | 150/200mW |
| Electrical characteristics | Insulation resistance (Initial) | | Min. 100MΩ (at 500V DC) Measurement at same location as "Initial breakdown voltage" section. |
| | Breakdown voltage (Initial) | Between open contacts | 500 Vrms for 1min. (Detection current: 10mA) |
| | | Between contact and coil | 1,000 Vrms for 1min. (Detection current: 10mA) |
| | Temperature rise (at 20°C 68°F) | | Max. 50°C (By resistive method, nominal coil voltage applied to the coil, nominal switching capacity.) |
| | Operate time [Set time] (at 20°C 68°F) | | Max. 5 ms (Nominal coil voltage applied to the coil, excluding contact bounce time.) |
| | Release time [Reset time] (at 20°C 68°F) | | Max. 4 ms (Nominal coil voltage applied to the coil, excluding contact bounce time.) (without diode) |
| Mechanical characteristics | Shock resistance | Functional | Min. 98 m/s² (Half-wave pulse of sine wave: 11 ms; detection time: 10µs.) |
| | | Destructive | Min. 980 m/s² (Half-wave pulse of sine wave: 6 ms.) |
| | Vibration resistance | Functional | 10 to 55 Hz at double amplitude of 1 mm (Detection time: 10µs.) |
| | | Destructive | 10 to 55 Hz at double amplitude of 2 mm |
| Expected life | Mechanical | | Min. $10^7$ (at 180 cpm) |
| | Electrical | | Min. $10^5$ (1 A 30 V DC resistive) (at 20 cpm) |
| Conditions | Conditions for operation, transport and storage*2 | | Ambient temperature: –40°C to +70°C –40°F to +158°F Humidity: 5 to 85% R.H. (Not freezing and condensing at low temperature) |
| | Max. operating speed (at rated load) | | 20 cpm |
| Unit weight | | | Approx. 1.8 g .063 oz |

Notes: *1 This value can change due to the switching frequency, environmental conditions, and desired reliability level, therefore it is recommended to check this with the actual load.
*2 Refer to 6. Conditions for operation, transport and storage mentioned in AMBIENT ENVIRONMENT.