Solar Powered Three-Phase Motor

Submitted To

Fernando de la Cuadra

Prepared By

Jaime Luengo Rozas

Mentor: Ross Baldick

Escuela Técnica Superior de Ingeniería (ICAI)
Grado en Ingeniería electromecánica
Rama eléctrica
2016-2017

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor D. **Jaime Luengo Rozas**

DECLARA ser el titular de los derechos de propiedad intelectual de la obra: "Solar Powered Three-Phased Motor", que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección.
b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
e) Asignar por defecto a estos trabajos una licencia Creative Commons.
f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente).*

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

a) Que la Universidad identifique claramente su nombre como autor de la misma
b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
c) Solicitar la retirada de la obra del repositorio por causa justificada.
d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

El autor se compromete a:

a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión.

MOTOR TRIFÁSICO ALIMENTADO POR PANELES SOLARES

Autor: Luengo Rozas, Jaime
Director: Baldick, Ross
Entidades colaboradoras: The University of Texas at Austin, ICAI-Universidad Pontificia de Comillas

RESUMEN DEL PROYECTO

Objetivo

En áreas geográficas aisladas y desfavorecidas donde el acceso a la red eléctrica es limitado, existe la necesidad de sistemas de generación de energía autónomos para desarrollar actividades que se consideran de primera necesidad para sus habitantes. El hecho de que estas zonas coincidan con unas de las áreas terrestres con el mayor nivel de luminosidad y que el precio de los paneles solares siga disminuyendo, hace de la energía solar una solución muy atractiva. Este proyecto desarrolla un sistema capaz de recibir energía solar desde unos paneles fotovoltaicos y producir energía mecánica en forma de unidades de aire acondicionado o bombeo de agua. Entre sus especificaciones, debe ser capaz de controlar la frecuencia del motor bajo varios niveles de luminosidad sin que el motor se bloquee. Además, al estar aislado de la red y evitar la compra de ningún tipo de dispositivo de almacenamiento eléctrico, el sistema tiene que consumir instantáneamente toda la energía producida por los paneles solares.

Metodología

Se recibieron de inicio cuatro paneles solares, un motor (bomba de agua) y un inversor trifásico. El inversor trifásico se concluyó que no cumplía las características necesarias del sistema y se optó por comprar uno nuevo [1], ya que era crucial para comprobar el correcto funcionamiento del resto de subsistemas. De este modo, las características del motor y de los paneles solares se convirtieron en restricciones de nuestro diseño. El proyecto se dividió en tres etapas.

1. Caracterización

Los paneles y el motor, como restricción del proyecto tuvieron que ser caracterizados para diseñar el resto de subsistemas conforme a sus características.

1.1. Paneles solares

Mediante una resistencia variable se sacaron las curvas I-V y P-V [2] de la figura 1 para saber en qué valor de tensión se conseguiría la mayor cantidad de potencia, siendo esta de 93 W a los 18 V DC. Se verificó que las curvas eran las mismas para los cuatro paneles solares.

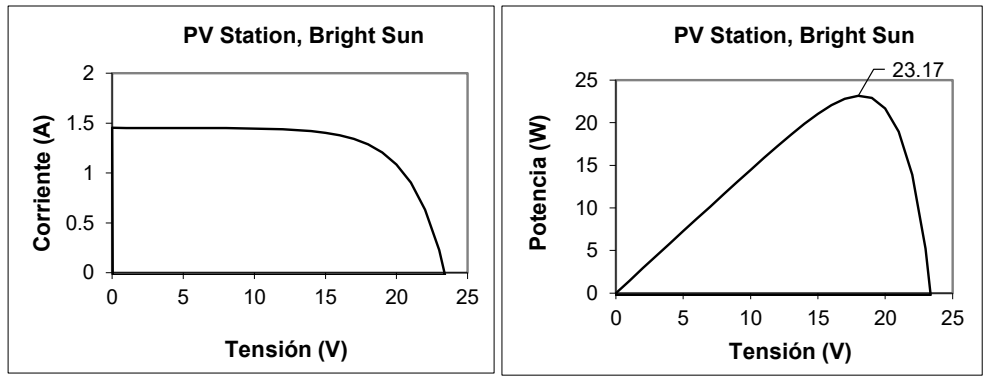

Figura 1. I-V y P-V características del panel solar

1.2. Motor-bomba de agua

El proveedor, Flair America, no disponía de características del motor más allá de la tensión, 230 V; corriente, 0.43 A; frecuencia, 60 Hz y potencia nominal, 1/8 HP [3]. Esto influenció notablemente el tipo de control que se seleccionó para el motor.

2. Diseño y desarrollo

Aparte de elevar la tensión de los paneles, se necesitaba disminuirla para arrancar el motor realizando *soft-start* [4]. Para ello, se utilizó el diseño de la figura 2 en el que el inversor aparte de convertir continua en trifásica, con el factor de modulación del PWM, también reduce la tensión. Los paneles fotovoltáicos reciben energía solar transformándola en corriente DC, el boost converter eleva la tensión dependiendo del valor de luminosidad dictaminado por la fotorresistencia y el inversor trifásico la pasa a alterna trifásica con una frecuencia que depende linealmente de la tensión anterior.

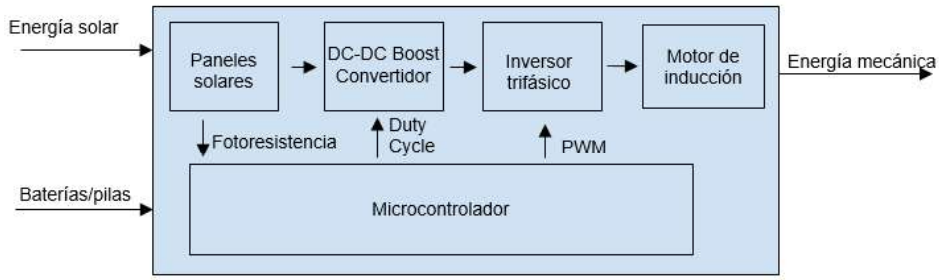

Figura 2. Diagrama de bloques del sistema

2.1. Convertidor DC-DC boost

Diseñado para operar en modo continuo [2] y crear tensiones cercanas a los 300 V. Se extrajo la figura 3, en la que se somete al convertidor a todo su rango de duty cycle para comprobar que se permanece en modo continuo en todo el rango.

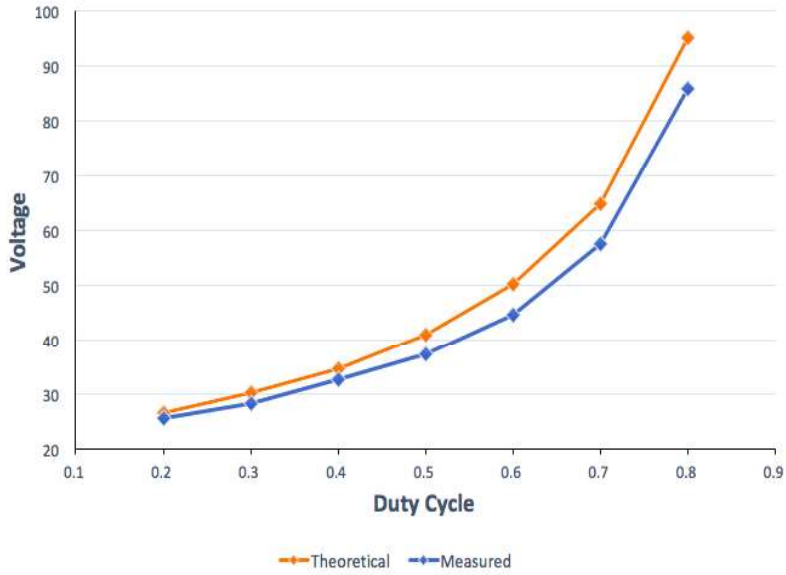

Figura 3. DC-DC Boost Tensión vs. Duty Cycle

2.2. Sensores

Se comenzó con sensores de tensión y corriente tanto en continua como alterna y sirvieron para implementar el algoritmo de seguimiento del punto de máxima potencia (MPPT). Este subsistema, tras ser construido y evaluado, se dejó para ser rediseñado en el futuro debido a que no se alcanzó la sensibilidad necesaria. En su lugar, se diseñó un circuito con una fotorresistencia para captar el nivel de luminosidad y distinguir claramente las etapas intermedias que hay entre soleado y totalmente nublado.

2.3. Software

Se implementan algoritmos de control para ajustar la relación de transformación del DC-DC convertidor y la frecuencia del inversor trifásico, dependiendo del nivel de luz. El control V/f se eligió para evitar la costosa compra de sensores de alta sensibilidad, por ausencia de datos suficientes del motor y por no haber necesidad de un control preciso del par. Dentro del control V/f, se eligió lazo abierto sin compensación del deslizamiento tras compararlo en Simulink con el control que incluye el deslizamiento [4]. Como se puede observar en la figura 4, el error en el seguimiento de la referencia de velocidad es aceptable para nuestra aplicación y los cambios en corriente y par son más suaves.

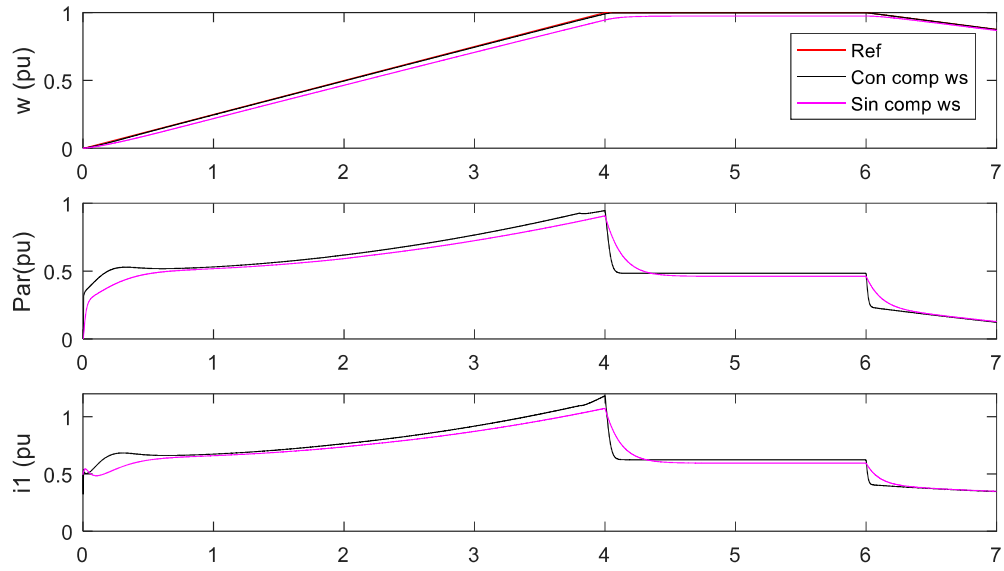

Figura 4. Comparativa del control V/f en lazo abierto con y sin compensamiento del deslizamiento

3. Integración:

Tras testar individualmente los sistemas, la integración se llevó a cabo en el siguiente orden:

- Inversor y motor a tensión de la pared, 120 V y 30 Hz, usando un rectificador
- Boost convertidor, inversor y motor con 300 V en el DC bus y 60 Hz
- Sistema completo incluyendo sensores y software

Resultados

Los resultados demostraron que el sistema era capaz de cambiar de una velocidad mecánica a otra cuando cambiaba la luminosidad, utilizando eficientemente la energía extraída de los paneles solares. Además se satisfizo el requerimiento de no bloquear el motor en estos eventos como se puede ver en la figura 5.

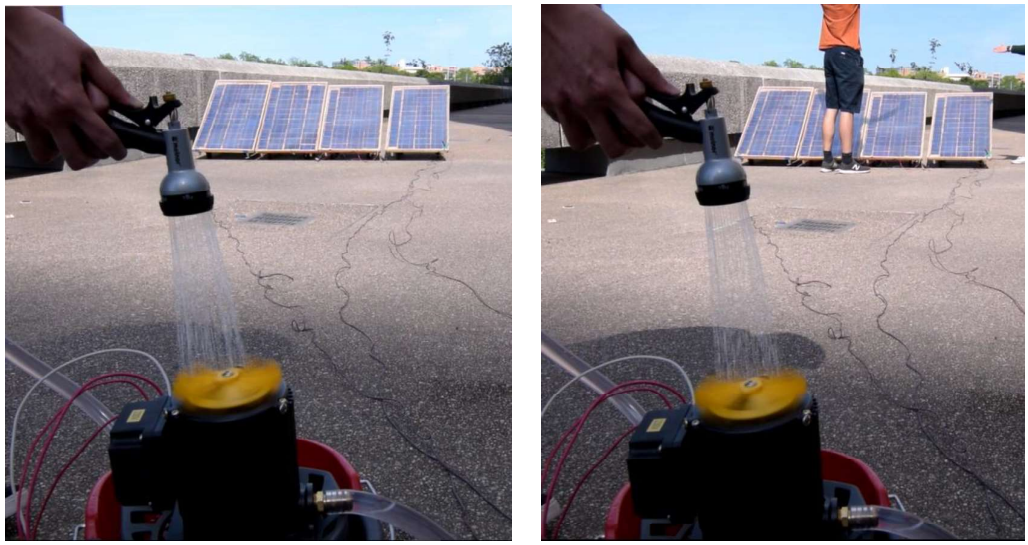

Figura 5. Cambio del caudal entre soleado y sombreado sin bloqueo del motor

Se comprobó que el sistema con la máxima luminosidad operaba a 300 V en el bus DC, que correspondía con la tensión nominal del motor de 230 V_{rms}. La electrónica de potencia se instaló en un banco de ensayo para hacerla más manejable y clara.

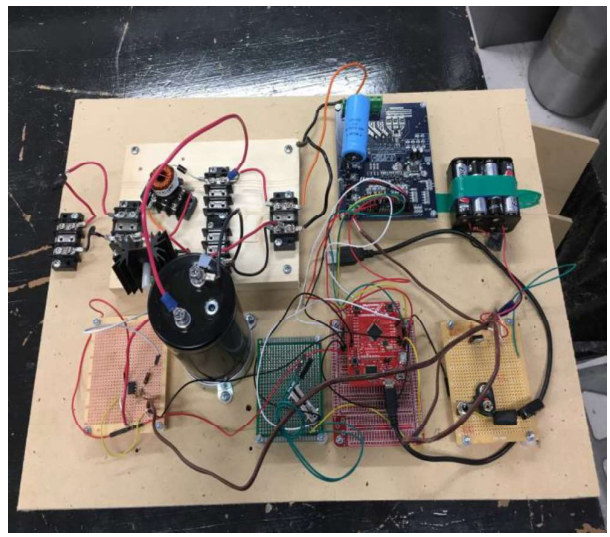

Figura 6. Banco de ensayos para electrónica de potencia

El coste total del sistema, sin paneles solares ni motor, es de $200. Al priorizar seguridad sobre coste varios componentes fueron seleccionados con un margen de seguridad de hasta 3 veces su valor típico de operación elevando el coste, lo que indica que se podría conseguir a menor

precio.

Conclusión

Se ha diseñado con éxito un sistema autosuficiente que funciona completamente fuera de la red y modula su punto de funcionamiento basado en los cambios de luminosidad. Utilizando una fotorresistencia para estimar los cambios en la luz solar incidente, el microcontrolador varía la tensión y la frecuencia del motor para impedir que el motor se detenga debido desequilibrios entre la potencia demandada por el motor y la proveída. Manteniendo la seguridad y la sostenibilidad como una prioridad, el sistema diseñado podría tener importantes implicaciones para aplicaciones aisladas de la red. En última instancia, se espera que este proyecto sea de acceso público para facilitar el acceso remoto a las comunidades subdesarrolladas al igual que alternativas de socorro en casos de desastre.

Referencias

[1] "Filterable Cooling Pumps," Flair America. [Online]. Available: http://flairamerica.net/pumps/filterable-cooling-pumps/. [Accessed: 06-Apr-2017].

[2] M. Flynn, The University of Texas at Austin, Austin, TX, EE 462L: Power Electronics Laboratory course, Spring 2017.

[3] "UM2019 User manual", www.st.com, 2017. [Online]. Available: http://www.st.com/content/ccc/resource/technical/document/user_manual/group0/5a/6c/0d/5f/b0/c4/43/00/DM00266643/files/DM00266643.pdf/jcr:content/translations/en.DM00266643.pdf [Accessed: 07- Apr- 2017].

[4] Lukas Sigirist, Universidad Pontificia de Comillas, Madrid, Spain. Máquinas Eléctricas y Accionamientos, Spring 2016.

SOLAR POWERED THREE-PHASE MOTOR

EXECUTIVE SUMMARY

Objective

For rural and underserved communities, in addition to other scenarios where access to grid power is limited, there exists a need for self-sufficient systems to deliver energy for a variety of applications. The fact that these areas have the highest solar potential and that the price of solar cells follows to decline, makes solar energy an increasingly attractive source of power for these areas. This project develops a system capable of receiving solar power from photovoltaic panels and producing mechanical energy in in the form of air conditioning units or water pumps. Additional criteria for the project includes independent operation from an electric grid and the ability to drive a variable frequency motor under various solar levels without stalling. As the project is off-grid and avoids the purchase of any energy storage, the system also has to instantaneously use any power produced by the solar panels.

Methodology

Four solar panels, a motor-water pump and a three-phase were received from the start. The inverter was found to do not served the necessary functions for the system and a new one was purchased [1], since it was vital to test the rest of the subsystems and it was a much cheaper option than building a new one with individual components. Therefore, the induction motor and solar panels' characteristics were constraints for the design. The project was divided in three stages: characterization, design and implementation, and integration.

1. Characterization

As constraints for the project, the panels and the motor, had to be characterized to design the other subsystems consistent with their parameters.

1.1. Solar Panels

Utilizing a variable power resistor, the I-V and P-V curves [2], shown in figure 1, were obtained in order to know at which voltage the solar panels would give the maximum power. This resulted to be 93 W at 18 V DC and coherent with the motor's rated power. It was verified that the curves were the same for the four solar panels.

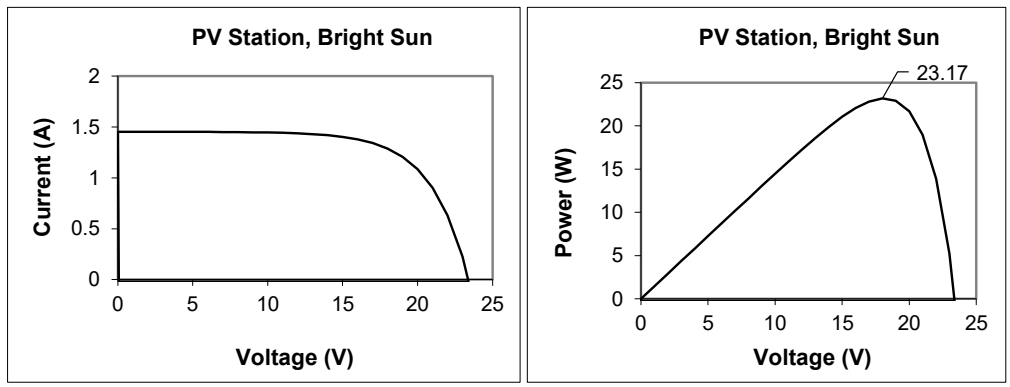

Figure 1. Solar I-V and P-V curves of each panel

1.2. Motor-water pump

Flair America, the provider, had no data available from the motor beyond its nominal voltage, 230 V; current, 0.43 A; frequency 60 Hz; and power, 1/8 HP [3]. This affected the type of control used for the motor.

2. Design and implementation

In addition to raising the voltage of the panels, it was necessary to decrease it to start the motor by performing "soft-start"[4]. For this purpose, the design of figure 2 was used in which the inverter, besides converting DC to three-phase AC, with the modulation factor of the PWM, can also reduce the voltage. The PV panels receive solar energy transforming it into DC electricity, the boost converter raises the voltage depending on the value of luminosity dictated by the photoresistor and the three-phase inverter converts it to three-phase AC with a frequency that depends linearly on the previous voltage.

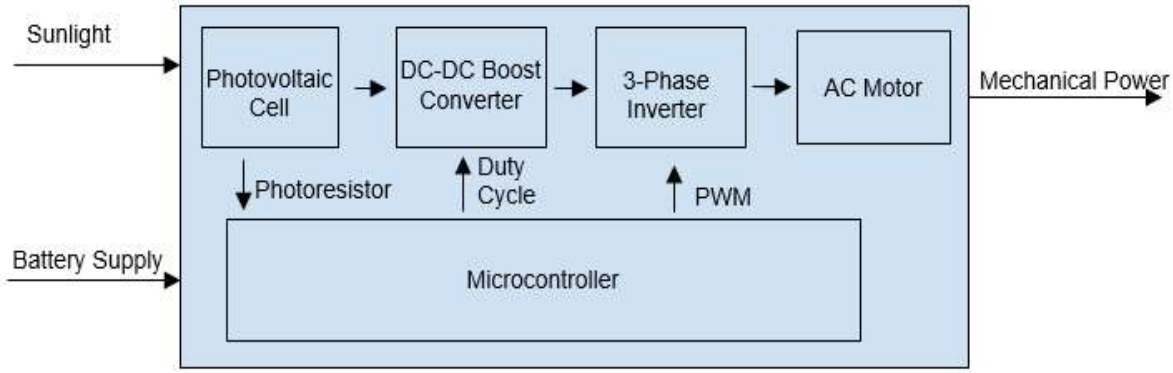

Figure 2. System block diagram

2.1. DC-DC boost converter

Designed to operate in Continuous Conduction Mode (CCM) [2] and hold voltages up to 300 V. Figure 3 was extracted from testing the Boost converter in its whole range of duty cycles, to check that it stays on CCM the whole range.

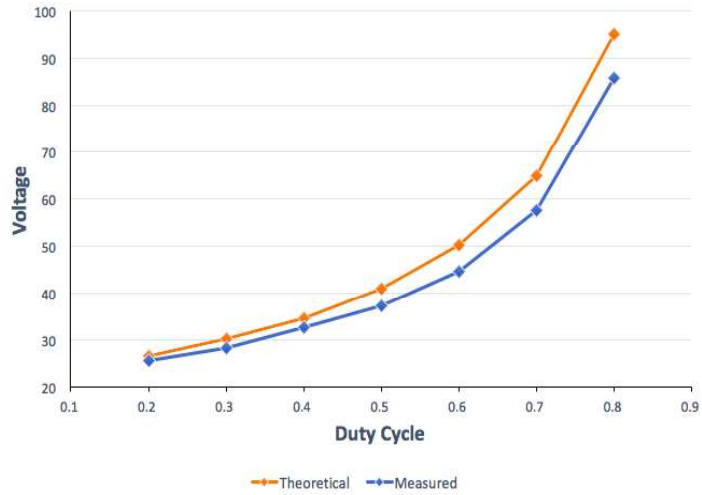

Figure 3. DC-DC Boost Operation Voltage vs. Duty Cycle

2.2. Sensors

Initially designed for voltage and current in both DC and AC side. They serve to implement Maximum Power Point Tracking (MPPT) in the control. After development and testing of this subsystem it resulted to not reach the level of sensitivity wanted for the system and were left for reevaluation in the future. It was replaced by a photo resistor circuit that senses the light level and clearly distinguishes the intermediate stages between fully sunny and fully cloudy.

2.3. Software

Implementation of control algorithms to adjust the voltage ratio in the boost converter and the three-phase inverter frequency, depending of the light level. V/f motor control was selected to avoid buying expensive accurate sensors and due to the non-necessity of accurate torque control and the non-availability of motor data. Within V/f control, the open loop that excludes slip was selected after using Simulink to compare with the open loop control that accounts for slip [4]. It can observed in figure 4 how the error for the speed is acceptable for the application and the changes in torque and current are smoother.

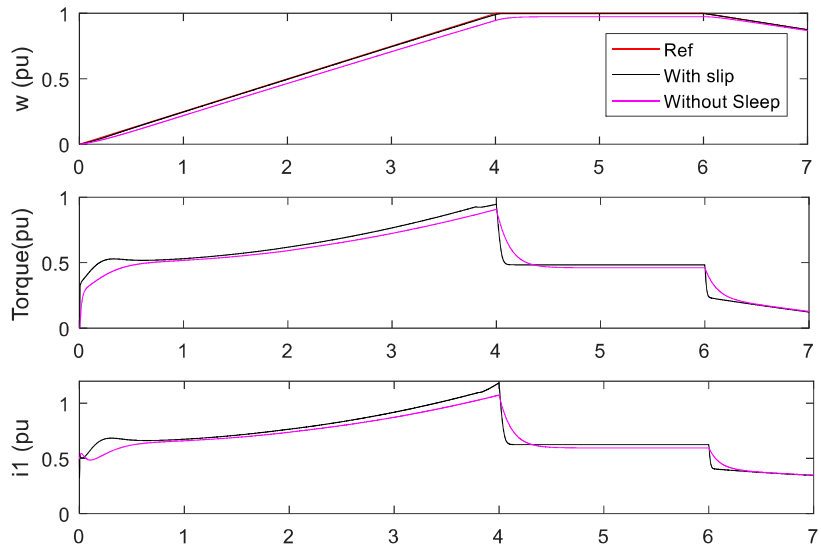

Figure 4. Open-loop V/f control with and without slip simulation

3. Integration

After testing individually each subsytem, they were integrated in the following order:

- Inverter and motor at wall outlet's voltage 120 V, 30 Hz, using a rectifier
- Boost converter, inverter and motor at 300 V in DC bus and 60 Hz
- Full system including sensing and software

Results

The results showed that the system was able to adjust to a new mechanical speed when the

sunlight changed while utilizing efficiently the energy drawn from the solar panels. In addition, as shown in figure 5, it fulfilled the requirement of not stalling when subjected to this scenario.

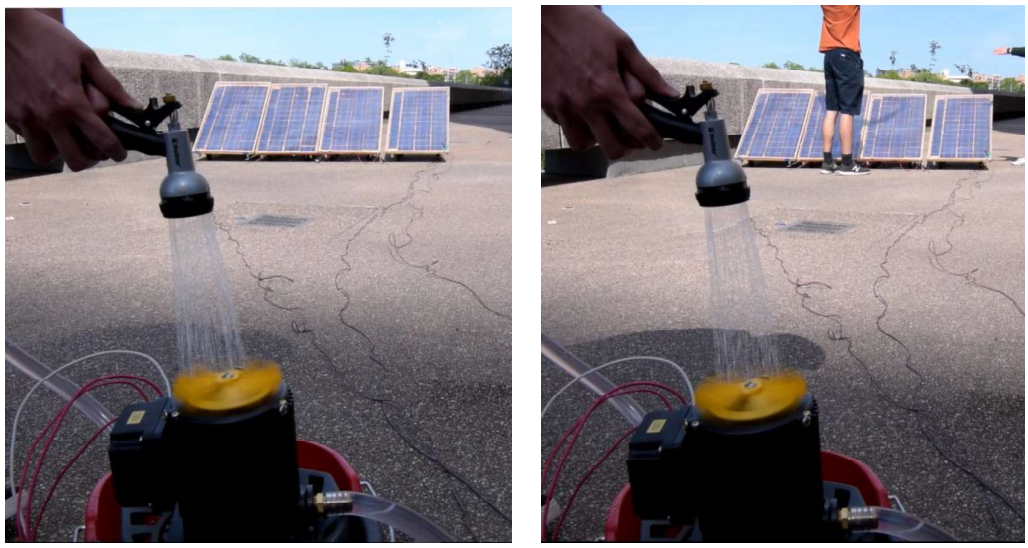

Figure 5. Change on flow rate due to manual fluctuation on sunlight avoiding stalling

It was also observed that at maximum lighting the DC-DC converter was outputting 300 V, that corresponded to nominal voltage of the motor of 230 V_{rms}. The power electronics were organized and mounted to a test bench to make them more manageable and clear.

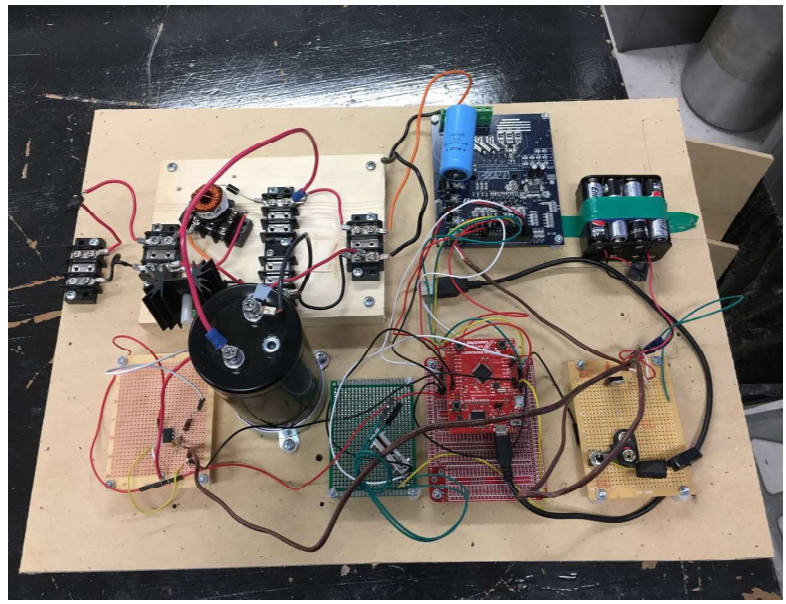

Figure 6. Power electronics test bench

The overall cost of the system, excluding solar panels and motor, was $200. However, since was prioritized over cost, using a safety margin up to three times the operating value of some components, the cost was higher than what it could be achieved.

Conclusion

A self-sufficient system was designed that runs completely off-grid and modulates its operating point based on changes in solar insolation. Using a photoresistor to estimate the changes in incident sunlight, the microcontroller varies the voltage and frequency driving the motor to prevent the motor from stalling due to insufficient input power. Keeping safety and sustainability a priority, the system designed could have major implications for off-grid power applications with small improvements such as overcurrent protection and a user interface. Ultimately, it is hoped that this project will be open sourced to facilitate remote power to under-developed communities and disaster relief efforts.

References

[1] "Filterable Cooling Pumps," Flair America. [Online]. Available: http://flairamerica.net/pumps/filterable-cooling-pumps/. [Accessed: 06-Apr-2017].

[2] M. Flynn, The University of Texas at Austin, Austin, TX, EE 462L: Power Electronics Laboratory course, Spring 2017.

[3] "UM2019 User manual", www.st.com, 2017. [Online]. Available: http://www.st.com/content/ccc/resource/technical/document/user_manual/group0/5a/6c/0d/5f/b0/c4/43/00/DM00266643/files/DM00266643.pdf/jcr:content/translations/en.DM00266643.pdf. [Accessed: 07- Apr- 2017].

[4] Lukas Sigirist, Universidad Pontificia de Comillas, Madrid, Spain. Electric Machines and Drives, Spring 2016.

Acknowledgements

To my parents, who were there at all times to support me.

To ICAI, for the constant effort in internationalizing the school and letting me experience other engineering culture. For the dedication of many of its professors that taught me how engineering can be a passion more than a job. To all my friends with whom I shared struggles and joy.

To UT Austin ECE department for allowing me to take part of Senior Design. Thanks to Ross Baldick for his hospitality during my stay in Texas and lead during the project. Finally, thanks to Mark Flynn for his altruist help and recommendations, especially in the Power Electronics part of the project.

Contents

List of Figures

Chapter 1: Introduction

1.1 Motivation

For rural and underserved communities, in addition to other scenarios where access to grid power is limited, there exists a need for self-sufficient systems to deliver energy for a variety of applications. As the scientific community continues to develop renewable technology, the price of solar cells has begun to decline. This has made solar energy an increasingly attractive source of power, especially for small operations. The form of this energy, at the time of generation, is direct current (DC). In contrast, many household appliances and motors use alternating current (AC) for both historical and practical reasons. Engineering a system capable of powering a device constrains consumers to purchasing a fully customized system for their specific application. This, in turn, eliminates the benefits which can be obtained from high-volume products in terms of lower system cost. The design solution presented within this document aims to reduce the cost seen by the end user by introducing a degree of modularity into solar installations.

1.2 Objective

This project develops a system capable of receiving solar power from photovoltaic panels and producing mechanical energy in the form of air conditioning units or water pumps. Additional criteria for the project includes independent operation from an electric grid and the ability to drive a variable frequency motor under various solar levels without stalling. As the project is off-grid and avoids the purchase of any energy storage, the system also has to instantaneously use any power produced by the solar panels. Safety, cost and functionality were a priority of the design, given the desire of making the project open source so it could be built in these underserved and isolated areas.

1.3 Prior Art

Prior to the project design, patents and academic articles were examined and discussed applicable research conducted on motor-control algorithms and DC-DC Converters. Existing research was reviewed in fields including current MPPT algorithms, VFD methods, and variable sunlight operation techniques to optimize the efficiency of the system.

1.3.1 Prior Art Exclusive of Patent Information

The IEEE Xplore database was the primary source for academic articles relating to power maximization algorithms and motor control.

1.3.1.1 Maximum Power Point Tracking Algorithm

MPPT describes the process of changing the operating point of a device to extract maximum power or useful work. For a solar panel, the maximum power point is a specific voltage and current level on the knee of its I-V curve. Sengar describes two basic algorithms in his review of the literature [1]. Under varying light levels, Perturb and Observe (P&O) and Incremental Conductance are the most popular techniques for MPPT. P&O simply varies the voltage of the solar panel and measures the power extracted. If the power is greater than in the previous time step, it moves the voltage in the same direction as it did in the previous time step. Otherwise, it moves the voltage in the opposite direction. This is called a hill-climbing algorithm. P&O is simple to implement, but as sunlight decreases, the power vs. voltage relationship of the solar panel becomes more flat, and the algorithm will tend to oscillate around the MPPT rather than operate at the actual MPPT. Additionally, P&O becomes unstable in rapidly changing conditions. On the other hand, Incremental Conductance is a method of comparing the instantaneous panel conductance with the incremental panel conductance to determine the magnitude and direction of the perturbation. Although it uses a different relationship to find the maximum power point, the algorithm is similar to P&O. Incremental Conductance is fast and efficient but requires complex control circuits.

Patel and Agarwal discuss some of the problems with simple MPPT algorithms under partial shading conditions and their approach to a solution [2]. Under ideal conditions, the current vs. voltage relationship for a solar panel has a predictable shape with an easily identified maximum power point; correspondingly, the power vs. voltage relationship has a single peak. When a solar panel is partially shaded, the current vs. voltage and power vs. voltage relationships of the panel are characterized by multiple peaks. As such, simple hill-climbing algorithms are insufficient for MPPT because they can potentially settle upon a local maxima rather than the global maximum. By studying the typical behavior of partially shaded solar panels, Patel and Agarwal developed a modified P&O algorithm that uses memorization, or storage of previously measured or calculated values, and exploration to find the maximum power point when a disturbance such as partial shading is detected. The authors report a reduction in tracking time by a factor of 10.

1.3.1.2 Variable Frequency Driving of an Induction Motor

The article written by Khoucha and Lagoun describes the Direct Torque Control (DTC) method of driving an induction motor, and compares different H-Bridge designs and their efficiencies [3]. A typical one phase H-bridge has 4 MOSFETs, set up like the letter H, with one MOSFET at each corner and a load connected in the middle. The control mechanism for the MOSFETS implements pulse width modulation (PWM), making the switches open and close at specific times, to adjust the duty cycle of each MOSFET so that the resultant waveform is nearly sinusoidal. The article claims that by implementing an asymmetrical multilevel H-Bridge inverter design that has more than four MOSFETS, i.e. one “leg” of the inverter has more components than the other (an asymmetrical design), switching power losses can be minimized compared to a typical H-bridge.

The design includes a 3 phase inverter that converts dc voltage to an AC waveform that is suitable for powering a motor. To save on both cost and time, work to implement this aspect of the design the project was then a priority.

1.3.2 Patents Search and Findings

A number of patents were found related to the problem of solar power and efficiency maximization through Google Patent Search. In particular, the focus were patents that could assist in integrating the subsystems, such as the DC-DC converter designs and the MPPT algorithms.

1.3.2.1 Maximizing Power Production at Low Sunlight

The system invented by George Shu-Xing Cheng, Steven L. Mulkey, and Andrew J. Chow and described in patent application number US 12/789,637 was filed on March 7, 2013 [4]. It describes an interface of mini-DC-DC voltage converters, power conversion circuitry, and a microcontroller to increase the efficiency of a PV system in low-light conditions. The system increases the efficiency of the solar-powered system and thus maximizes the aggregate of solar power generation per day. To accomplish this, the low-sunlight maximal power production system contains circuitry to power the dc control components even in low-sunlight conditions.

The low-light maximal efficiency patent operates by attaching a DC-DC boost converter to each of the solar panels. These converters boost the output voltage of the solar panel, which in turn reduces the current. Controlling the MOSFET with a microcontroller extracts the maximum power from the panel since the DC-DC boost converter is capable of changing the load to match the maximum power point of the photovoltaic cells. The dc power combiner produces a single output voltage and current from the four panels. Some of this power is consumed by the control electronics, such as the microcontroller, and the rest is converted to alternating current to drive the mechanical load. The microcontroller monitors the light levels of the solar panel and, upon detecting a low-light state, it devotes one panel to powering itself so that the microcontroller, and thus the systems, remains operational. If the light level does not provide sufficient power to the microcontroller from one solar panel, the whole system shuts down. Particular elements of this design can help gaining insight into how to manipulate power supplied to the motor. A Block Diagram of their full system is shown below.

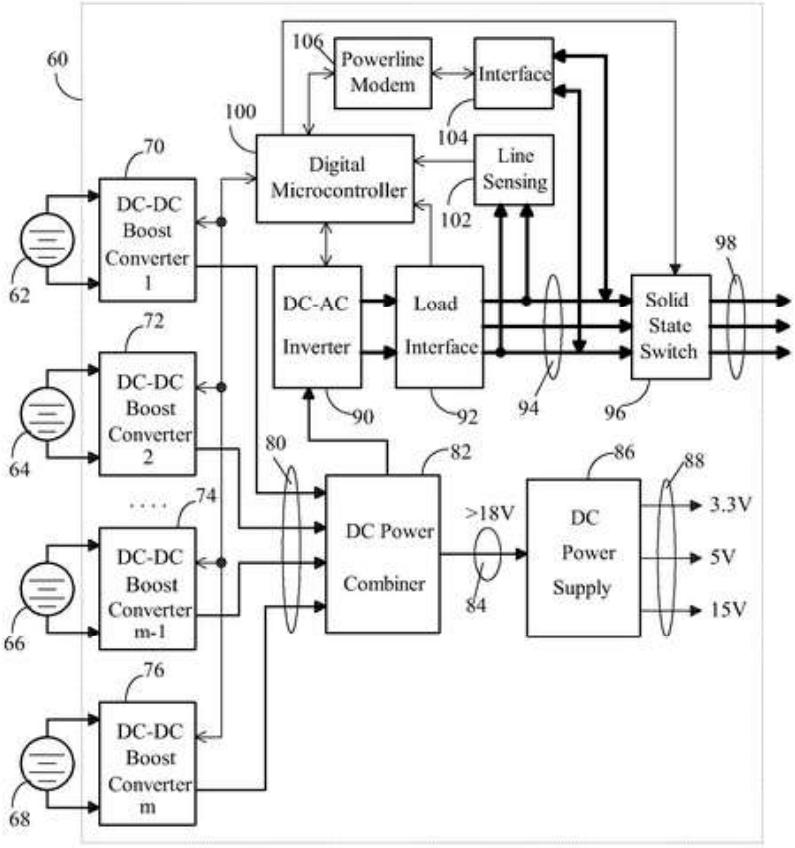

Figure 1: Low-Light Maximal Power System Block Diagram [4]

1.3.3 Impact of Prior Art Search on Design Decision

Several aspects of the research into the prior art on power point tracking, variable frequency motors, and solar power efficiency have influenced the designs for the various phases of the ultimate product.

At first, the design for MPPT involved only the Perturb & Observe method to identify the maximum power point of the solar panel. A conclusion after researching Patel and Agarwal's paper is that faster and more precise methods exist, and it was explored further to find the most appropriate algorithm for the project. In particular, the design required a fast operating algorithm to ensure the motor does not stall when light levels change on the solar panel, and therefore, this article is critical to the final design.

Given enough time to redesign the 3 phase inverter, the research on VFD would have been particularly useful. The asymmetric design would have increased the performance and efficiency of the system. Additionally, the article mentions that the system is capable of filtering low harmonics. Typical H-Bridge designs struggle with this task and the harmonics can act as a source of vibration on the motor shaft. Damage to the motor by this kind of

disturbance can be reduced by using flexible shaft couplers. However, a more effective design would be to implement an H-Bridge that does not result in these stresses in the first place.

The project goal is to maximize the amount of energy from a solar array converted into mechanical power. The microcontroller provides sufficient control to operate the system, and the power cost of running one microcontroller is outweighed by the benefits of the optimization that it could be achieved. Thus, Cheng's patent design is useful for maximizing the operating time of the system per day. His findings suggest that by providing uninterrupted power to the microcontroller powered, it can be maximized the daily output of the mechanical load. Constrained by the project parameters to operate independently of grid power, the microcontroller ultimately had to receive power from battery supplies.

1.3.4 Conclusion

The research on past projects, patents, and other designs related to solar powered motor function and MPPT, provided with a more comprehensive idea of what problems it had to be overcome and how engineers have solved related issues. And even after not being able to implement MPPT in the final product, it was helpful to include in the report so future engineers can make use of it. Several subsystem designs within the proposed solution, such as tracking algorithms, variable frequency motor control methods and solar power maximization techniques, have been developed independently. Moving forward they were integrated the methods that other researchers and engineers have proposed to solve the problems of optimizing both the efficiency and practicality of off-grid solar power.

Chapter 2: Design Problem

Current solar technologies primarily convert solar power to a constant voltage, limiting it storage and transfer capabilities. These limits are a result of the thermal losses suffered by direct current systems and the inefficiencies of modern battery technology. In the interest of minimizing these losses, it was designed and built a system capable of converting collected solar energy to alternating electric current and delivering this power immediately to a mechanical load.

Converting solar energy to mechanical power for a general load introduces losses to the system from heat dissipated by vibrations and thermal losses from high frequency switching. To address these adverse effects, there were placed controls on parameters such as electrical frequency and load voltage. Variations in sunlight intensity complicated the characterization of these parameters. The design took into account the effects of cloud cover and other environmental variations to maximize power efficiency in any location, despite changing conditions. In addition, the system had to be compatible with any three-phase motor, so as to not limit use cases based on a particular type of mechanical load such as a water pump or an air conditioning unit.

2.1 Stakeholder Needs Analysis

Dr. Baldick is the primary stakeholder in this project. His primary concern was to create a functional prototype that was independent of the power grid. Listed below are other needs that were discussed with Dr. Baldick as essential to the end design.

- Safe for end-users
- Maintenance-free
- Reasonably priced
- Compatible with unknown mechanical loads
- Compatible with unknown solar power arrangements

With these priorities in mind, the costs were minimized by focusing on designing a system built from the existing designs and subsystems prior to the project. These materials include the designs for the three-phase inverter, the pulse-width modulation (PWM) algorithm, the DC-DC converter, and the solar panels.

2.2 Relevant Standards

It was needed to comply with standards for electrical device safety as defined in the National Electrical Code (NEC) because it was aimed towards the completion of an electrical device [6].

The NEC is maintained by the National Fire Protection Association (NFPA), and spells out the measures which should be taken in the device implementation to keep the end users safe. Primarily it is considered the standards which apply to electrical isolation and spacing, material types, and temperature ratings of the system. Standards which apply to connection with a grid did not apply to the project since it was implemented a self sustaining design.

Consumer power electronics which deal with the distribution also fall under the guidelines described in the UL-1741 standard [7].The requirements (such as enclosure material, groundings, etc.) imposed by this standard on grid independent and photovoltaic systems must be met in order to receive the UL-1741 certification. While it was unlikely that it was applied for the formal recognition for adherence to these guidelines, they gave a set of metrics to keep in mind during the design and build phase.

The device built throughout the course of this project was intended to function with any motor, and therefore it was needed have an understanding on the limits and operating standards of three phase motors in general. More specifically, it was required that the motor used operates within the specifications set out by the National Electric Manufacturers Association (NEMA) [8].This ensured reliable compatibility between the device and any motor made to the appropriate code.

The relevant standards are explained with more detail in appendix A.

2.3 Design Functionality

The system converts solar energy to mechanical energy in two stages. The first stage is a “boost converter” which raises the dc voltage level such that the minimum voltage requirement of the motor was met. The second stage, the “three phase inverter,” modulates the input voltage to generate alternating current to drive the motor. This method allows to store mechanical energy or run appliances in areas with no electrical grid access. Algorithms on a microcontroller enables the motor to maintain an optimized frequency and voltage output that allow the motor to run at maximum efficiency. It was also implemented a calibration method for new solar panel configurations and motors. This allow the system to adapt to consumer needs and maximize its power efficiency.

2.4 Use Cases

The main use for the complete project is to provide power to a mechanical load in areas where there is no access to an electrical grid. This would be particularly useful in rural areas where access to utility power lines is limited. For these areas, storing mechanical potential energy is typically more affordable and accessible for users. This stored energy could then be used to drive irrigation systems for long-term use in farming or other similar situations. The system

would provide a reliable method for "smoothing applications" in which existing utilities such as air conditioning could be operated by the solar panel. Additionally, in remote areas without electrification, the cost that it would take to create a grid could be mitigated by using this system instead. Another use case of the system would be in disaster relief scenarios where power is maintained despite a severed grid connection. This could also apply to countries where rolling blackouts on the main grid are a common occurrence. During blackouts, the system could continue to power key utilities. The design should be useful in any situation where using grid power is unreasonable or impossible.

2.5 Ethical Considerations

To engineer a consumer utility appliance, user safety, environmental impact, and professional integrity needed to be treated as critical considerations in the design. Another objective of this project was to be available to the ordinary user in a variety of locations, so safety considerations are of the utmost concern, as well as thorough documentation and ultimate functionality.

Safety concerns involve the hardware in the system because there were high voltages and currents produced by the solar panels and boosted throughout the system. This concern combined with the device being used in a number of environmental conditions posed safety concerns as well, as the system must be robust enough to handle variations in temperature, humidity, and wind, among other variables. Hardware components were used rated for at least 1.5 times the maximum theoretical voltage and current it was expected in the system. Because the system operated at such high voltages, extra considerations were made to create a safe housing for the device that is capable of shielding it from all weather conditions, and possible misuse by the user. It is planned to Open-Source the project upon completion. Because of this, it was always considered safety a priority over cost when deciding on hardware components to implement in the circuit designs.

Another objective was to design a device that was environmentally friendly. It was aimed to reduce the usage of hazardous material, such as lead, in the design. Because this design is intended for rural applications, the system had to create as little pollution and waste as possible. It also had ensure that the electromagnetic radiation emitted from the system is minimal and safe; while it would be cheaper to create a system that produces damaging harmonics and emits electrical signals, it had not to interfere with other electronics or signals, such as radio waves, that could be in proximity of the user. The end goal was to design a system that maximizes power efficiency, reduce the unnecessary use of grid power, and reduce the need for electrical energy storage through immediate mechanical use, and therefore reduce the overall waste in performing a mechanical task.

2.6 Project Deliverables

The desired final products of the project are listed below in order of priority. Firstly, Dr. Baldick would like a fully integrated system that includes hardware, software, and documentation of all the subsystems involved. The solar panel, DC-DC boost, 3 phase inverter, and motor should work together to produce a mechanical output from the solar panel's power. For these items, the circuit diagrams, PCB design, code interfaced onto the microcontroller, and lab reports detailing the work were provided. This includes code for the maximum power point tracking, pulse width modulation, voltage and current sensing, and implementation of safety measures by limiting voltage exposed to hardware components. This code would, ideally, include a calibration method and implement machine learning to adapt to the changes of the system over time. Dr. Baldick requested that, provided there was enough time, the hardware design should be revisited and improve it to operate more efficiently. However, this was a stretch goal, rather than an expectation, because the previous groups have had trouble producing a finalized product due to unsolved challenges. Ultimately, the purpose was to design a prototype to show at Senior Design Open House: a water pump operated by the solar panel.

2.7 Operating Environment Specifications

To have a fully functioning prototype, the design must operate in fluctuating outdoor environments; for example, the temperature, humidity, or level of sunlight could change at any given moment. Table 1 in Appendix B lists the design specifications related to environmental factors.

2.8 Performance Specifications

The system must operate off-grid in dynamic environmental conditions and change power supplied to the motor based on input power from the solar panel. It must be able to calibrate to any solar panel and three-phase motor supplied by the user, and operate at the most efficient point based on the calibration and incident sunlight without dependence on any energy sources other than the PV panel. Table 2 in Appendix B details the final desired performance specifications.

Chapter 3: Design Solution and Implementation

The goal of the project was to implement a system that converts DC power from a solar panel to drive a variable-frequency three-phase load, such as a motor, with AC power. The system implementation is unique in that there is no energy storage or connection to a grid. Therefore, all the power needed for the motor must be provided instantaneously by the solar panels. Five subsystems have been integrated that are required for this project: an array of four solar panels, a DC-DC boost converter, a three-phase inverter, a microcontroller, and an AC three-phase motor.

An array of four solar panels in series produces DC power when exposed to sunlight. At the output of the system, this power is transformed into mechanical energy by a motor. The DC voltage that the solar panel supplies is not at the appropriate level to feed to the motor, so a boost converter is essential to step up the voltage to a level that can operate the motor. A three-phase inverter converts the DC power to the AC power required by the motor. Both the DC-DC converter and the three-phase inverter are controlled by a microcontroller to ensure that the behavior of these systems is within the constraints of the motor and allows for control of the VFD (variable frequency drive). Below is a block diagram that shows the integration of the complete system and the electrical connections required.

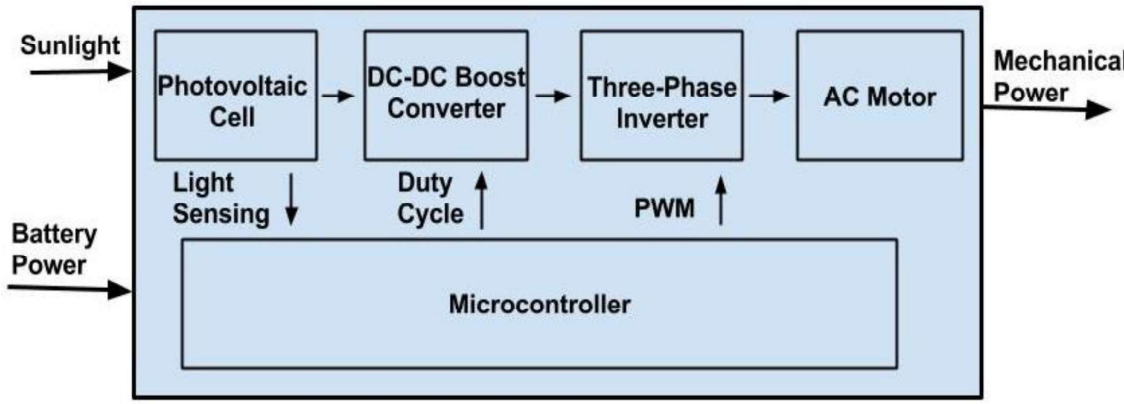

Figure 2. High-Level Hardware Block Diagram

3.1 Solar Panels

Solar Panels are composed of arrays of photovoltaic (PV) cells. These PV cells generate electric energy through the use of a typical P-N junction. Typical solar panel efficiency are 14%, meaning that only 14% of the sunlight energy that hits the solar panel is transferred to electrical power. When photons hit the n-type surface of the junction, a current is generated as the silicon absorbs light and produces excess holes/electrons. These flow to an external circuit

and can be used to power systems that use DC power. Therefore, the circuit of the solar cell can be generalized as a current source with a diode attached as shown below:

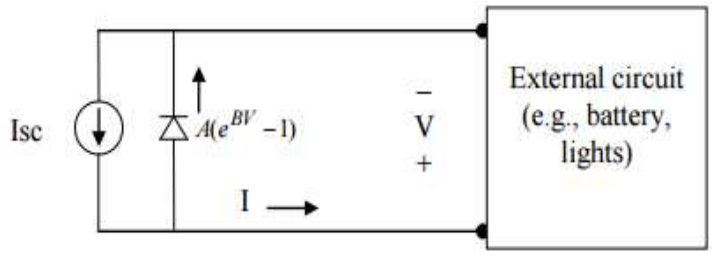

Figure 3. Equivalent Circuit of a Solar Cell [9]

Notice that the diode allows current to flow in the case of an open circuit external circuit. The excess charges return to the p-n junction. Because of the diode, the I-V characteristics of the solar panel is nonlinear.

Since the Voc of one PV cell is very low (0.5-0.6V), many of them are connected in series to yield a higher Voc. To drive more power, many of these series connections are placed in parallel, forming a 2-D array of solar cells, called a solar panel. The I-V characteristics of a typical solar panel can be seen below.

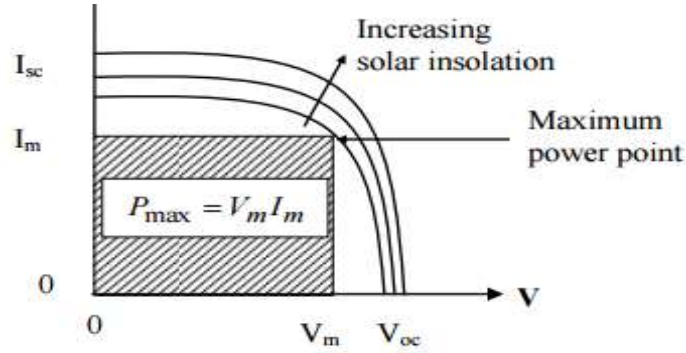

Figure 4. I-V Characteristics of Solar Panel [9]

As seen above, the I-V is nonlinear and there exists a point where the power (Power = Current * Voltage) is maximized. As the sunlight incident on the panel increases, it is referred as increasing solar insolation. More sunlight hitting the solar panel means that more power can be produced. Since solar power is relatively expensive to produce, it is important to maximize the power output. This can be accomplished by using a DC-DC converter to draw the maximum power from the panel by adjusting the load to match the maximum power point. However, this is not covered in this lab.

When orienting a solar panel, it is important to understand the sun position so as to maximize the capture of solar energy. Because of high wind loads, solar panels are usually fixed in a single position. To understand the position, it is needed to understand a couple of values first.

The Sun Declination Angle (δ) is the angle between the equator and a line drawn from the center of the earth to the sun. This changes based upon the day in the year. The formula is shown below.

$$\delta = 23.45 \bullet \frac{\pi}{180} \bullet \sin\left(2\pi \bullet \frac{(284+n)}{365}\right) \quad (1)[9]$$

where n is the day, being January 1st a value of n equal 1.

The equation of the time in minutes is as follows:

$$\begin{aligned} &\text{for } 1 \le n \le 106,\ E_{qt} = -14.2 \bullet \sin\left(\pi \bullet \frac{(n+7)}{111}\right), \\ &\text{for } 107 \le n \le 166,\ E_{qt} = 4.0 \bullet \sin\left(\pi \bullet \frac{(n-106)}{59}\right), \\ &\text{for } 167 \le n \le 246,\ E_{qt} = -6.5 \bullet \sin\left(\pi \bullet \frac{(n-166)}{80}\right), \\ &\text{for } 247 \le n \le 365,\ E_{qt} = 16.4 \bullet \sin\left(\pi \bullet \frac{(n-247)}{113}\right) \end{aligned} \quad (2)[9]$$

So, with these, it can be calculate the Tsolar (time) as shown below:

$$T_{solar} = T_{local} + \frac{E_{qt}}{60} + \frac{\left(Longitude_{timezone} - Longitude_{local}\right)}{15} \quad (3)[9]$$

Tlocal is local standard time and longitude time zone is the longitude at the eastern edge of the time zone.

The hour angle (in radians) is:

$$\omega = \pi \bullet \left(\frac{12 - T_{solar}}{12}\right) \quad (4)[9]$$

The Sun Zenith Angle is the angle that the sun is off of the normal line to the surface of the earth and can be measured the short circuit current of the solar panel by connecting a wire to the output terminals. This value was found to be 4.0A.

$$\cos\left(\theta_{sun}^{zenith}\right) = \sin(\lambda)\sin(\delta) + \cos(\lambda)\cos(\delta)\cos(\omega) \quad (5)[9]$$

Where λ is the latitude of the location of the panel.

The Sun Azimuth angle is the angle the sun is located at in comparison to North and can be calculated using the following equations:

$$f_{VE} = \cos(\delta)\sin(\omega) \quad (6)[9]$$

$$f_{VS} = -\sin(\delta)\cos(\lambda) + \cos(\delta)\sin(\lambda)\cos(\omega) \quad (7)[9]$$

$$\text{If } f_{VE} \geq 0,\ \phi_{sun}^{azimuth} = \cos^{-1}\left(\frac{-f_{VS}}{\sqrt{f_{VE}^2 + f_{VS}^2}}\right),$$

$$\text{If } f_{VE} < 0,\ \phi_{sun}^{azimuth} = \pi + \cos^{-1}\left(\frac{f_{VS}}{\sqrt{f_{VE}^2 + f_{VS}^2}}\right) \quad \text{(8a, 8b) [9]}$$

Now that the sun's orientation is known, now the panel can be oriented such that it is maximized the cos(β incident) = 1. This βincident is the angle that the vector of the sun's rays makes with the solar panel. When perpendicular, power sent to the panel surface is maximized. The equation below is a simplified version of how to orient the panel to maximize β incident.

$$\cos\beta_{incident} = \sin\theta_{sun}^{zenith}\sin\theta_{panel}^{tilt}\cos\left(\phi_{sun}^{azimuth} - \phi_{panel}^{azimuth}\right) + \cos\theta_{sun}^{zenith}\cos\theta_{panel}^{tilt} \quad \text{(9) [9]}$$

Where Θ (tilt), panel is the angle that the panel is off of the earth's surface, and Φ (azimuth),panel is the direction that the solar panel faces relative to true North is 0°.

With this value the total power incident on the panel can now be computed using the following:

$$P_{incident} = DH + \frac{(GH - DH)}{\cos(\theta_{sun}^{zenith})} \bullet \cos(\beta_{incident})\ \text{W/m}^2 \quad \text{(10) [9]}$$

Where DH is the diffuse horizontal light that is not from the disk of the sun, and GH is the total light on the panel.

Multiplying this by the efficiency η (14% for this particular array) of the solar panel, it can be calculated the amount of electric energy provided by the solar panel.

The implemented panels were constructed prior to this project and were integrated into the final prototype. They were built mounts for each panel that angled them at 30 degrees, the optimal angle for maximum solar potential (insolation) based on Austin's latitude.

Figure 5. Set of four solar panels

3.2 DC-DC Converter

The DC-DC converter is intended to either buck (decrease) or boost (increase) the input voltage as the available solar power varies. The output voltage is linearly proportional to the duty cycle observed in the periodic signal, provided by the microcontroller, at the gate of the MOSFETs. The LM5022MM/NOPB is a low side non-isolated single-ended primary-inductor converter regulator that has ten pins for feedback control, deadtime control, current amplification, and current sensing [10]. This chip thus served as a gate driver for the MOSFET. Combined with the microcontroller, it was able to control the duty cycle of the MOSFET and thus the boost ratio. This module is capable of handling an input voltages between 6 and 95V DC.

The first intention was to use a single-ended primary-inductor converter (SEPIC), which would be able to increase and decrease the DC voltage relative to the solar panel output. The SEPIC can increase or decrease the voltage depending on the duty cycle sent to its MOSFET. However, it has more components than other converters, making it much more complex to build and debug. In addition, the three-phase inverter (see Section 3.3) can step down the voltage, and a DC-DC converter able to simply step up the voltage would be enough. Therefore, it was chosen the boost converter.

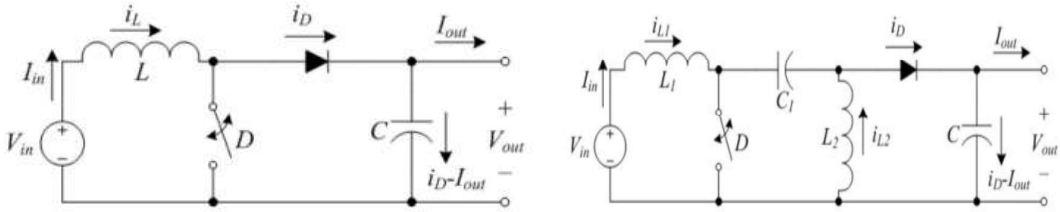

Figure 6: Circuitry Comparison between Boost Converter (left) and SEPIC (right) [9]

The DC-DC boost converter circuit is used to increase the incoming DC voltage to a higher output DC voltage. The incoming DC voltage should be ripple free. As seen in figure 7 below,

the circuit has a power electronic switch that opens/closes at a variable duty cycle in order to control the Vout DC voltage. The capacitor C is used to make sure that the Vout is as close to constant the ideal theoretical calculation of Vout, and makes sure that any ripple is minimized. The .01Ω resistor is not a part of a standard Boost Converter, but is included to measure the output current of the device and act as a fuse in case of overcurrent. When the circuit is acting under normal operating conditions, then the circuit is in "continuous conduction", which means that iL is always greater than 0A.

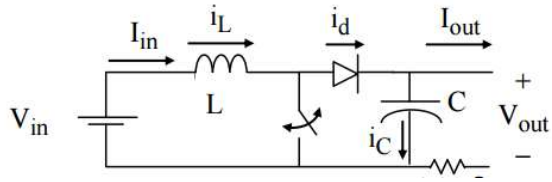

Figure 7. DC-DC boost converter circuit diagram [9]

In this case of continuous conduction, the circuit has two states: switch closed and switch open, which can be seen in figure 8a and 8b below.

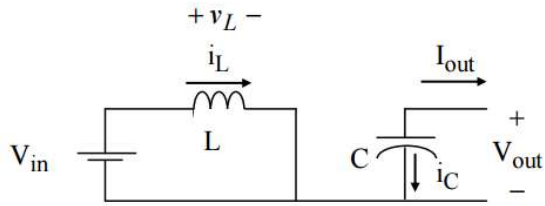

Figure 8a: DC-DC boost converter when switch is closed for DT (seconds) [9]

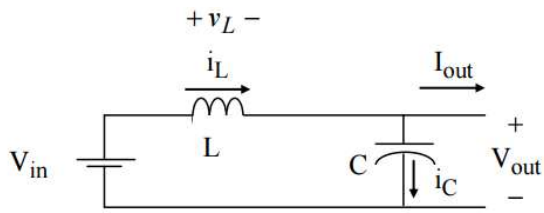

Figure 8b. DC-DC boost converter when switch is open for (1-D)T (seconds) continuous conduction [9]

As seen in the diagrams, when the switch is closed, the inductor is allowed to charge it's power. This is described with the following formula:

$$\frac{di_L}{dt} = \frac{v_L}{L} = \frac{V_{in}}{L}, \quad 0 \le t \le DT \qquad (11)\ [9]$$

Upon opening, the current is then directed through the diode, charging the capacitor.

$$\frac{di_L}{dt} = \frac{v_L}{L} = \frac{V_{in} - V_{out}}{L}, \quad DT < t < T \qquad (12)\ [9]$$

When the circuit switches to closed again, this charge stored in the capacitor provides the Vout

for the circuit. Knowing this, it can be used Kirchoff Voltage Law to describe the behavior of the Boost Converter.

$$\frac{(V_{in})DT + (V_{in} - V_{out})(1-D)T}{T} = 0 \qquad (13)\ [9]$$

Simplifying this, Vout can then be written as:

$$V_{out} = \frac{V_{in}}{1-D}. \qquad (14)\ [9]$$

Ignoring power loss across components, this is the general formula for a DC-DC boost converter.

If the assumption that the circuit operates exclusively in continuous conduction is not made, the boost converter is on discontinuous conduction. That is, when the inductor current falls to 0A,the capacitor will try to feed the inductor. The diode prevents this from happening, and thus all the power to the load is provided by the capacitor. This does not allow the capacitor to reach the charge necessary to operate at its theoretical capability. Another thing to note is that the voltage across the inductor during discontinuous conduction is 0V. Figure 9 below shows what happens to the circuit during discontinuous conduction. This is considered a 3rd state, in addition to the ones shown in figure 8a and 8b.

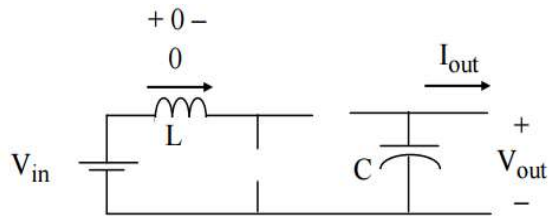

Figure 9. DC-DC boost converter, discontinuous conduction [9]

On the gate driver part of the circuit, an obstacle that had to be addressed was grounding. The MOSFET switching requires a certain level of voltage between its gate and source. This voltage in the circuit is the difference between power ground, and the gate signal provided by the gate driver. However, if the gate signal was referenced to a ground with a different voltage than the power ground, the MOSFET could be turned on or off indefinitely. Because of this, power and gate driver grounds were both tied together as appear in figure 10.

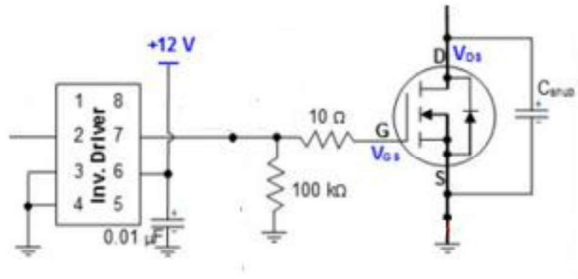

Figure 10. DC-DC Boost Converter gate driver [9]

This gate driver was controlled by the microcontroller that emitted the PWM for the correspondent duty cycle.

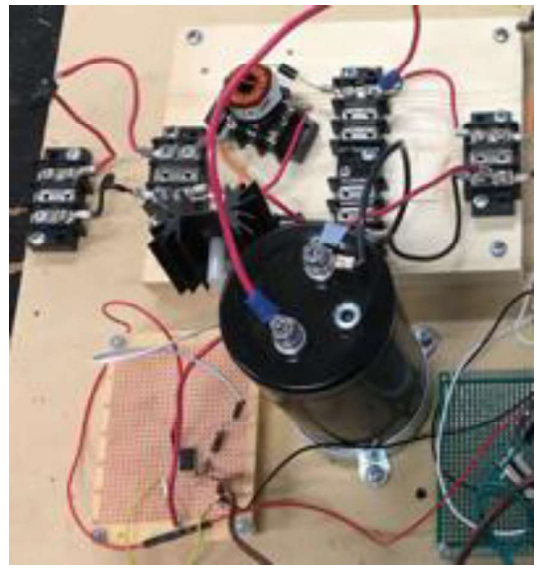

Figure 11. DC-DC boost converter built

In figure 11 it can be appreciated the size of the capacitor that had to be oversized for safety, with a voltage rating of 600V.

3.3 Three-Phase Inverter

Solar power generation produces a DC voltage output while motors and various other loads rely on either single or three phase sinusoidal power in order to function. In the project it was intended on incorporating a three phase motor in the interest of mechanical and electrical stability. The three phase inverter, staged directly after the DC-DC converter, serves the purpose of providing sinusoidal power to the motor. A successful inverter must perform at minimal power consumption where active elements are present. In addition to this, it was needed modify the frequency of the inverter output as determined by algorithms carried out by the microcontroller. As a whole, an inverting system consists of a PWM source, high power transistors (either MOSFETs or IGBTs), an integrated circuit driver to modulate the gates of

the transistors, and passive components (e.g. resistors, capacitors, and inductors). In the application, a microcontroller outputs the PWM signal to the driver integrated circuit (IC). The driver generates pairs of outputs linearly proportional to the received PWM signal from the microcontroller such that the second output is the negation of the first. These outputs are connected to transistors which effectively short the next stage of passive elements to either power or ground. Finally, passive elements are placed before the load to smooth the output waveform from the transistor stage. In the case of a three phase inverter, this process is carried out for three PWM signals which are 120 degrees offset from one another.

3.4 Motor (Water-Pump)

The system works for any three-phase load, but it was chosen to develop the system with a motor, one of the most common three-phase loads. A motor transforms electrical to mechanical energy by creating a magnetic field between its stator and rotor. This magnetic field is produced by powering three windings located in the stator, separated 120 degrees physically and electrically, with AC current. This creates a constant rotatory magnetic field that induces the rotor to spin at almost the same speed as the frequency of the time varying magnetic field. The difference between this two speeds is called the slip. The larger slip indicates an increase in torque which the mechanical load is consuming, and therefore an increase in the delivered power. In the case, the mechanical load is a water pump.

The motor-water pump, Flair America SP-8130 is designed to operate at 230V, 0.43A and 60Hz. However, due to the variation in solar power, the voltage fed to the motor it is adjusted such that the mechanical power delivered is maximized. To start the motor without consuming a large current, the voltage and the electrical frequency must be ramped up from zero to the desired value in a finite time interval, this is called “soft start”. Additionally, if the motor is set to operate at a particular power level, but the power changes due to fluctuations in the sunlight, the motor is at risk of stalling. Stalling occurs when a mechanical load prevents the rotor from turning. If the motor attempts to draw more power than the amount available at it terminals, the motor will stall and heat up significantly. Therefore, it will needed to implement a motor control algorithm capable of changing from one mechanical frequency to another over a time interval which is sufficiently long to prevent the motor from stalling. From equation 15 below, when the water pump is in steady state, the electric torque is equal to the mechanical torque.

$$m_{elec} - m_{mec} = J\frac{dw_{mec}}{dt} \quad (15)\ [11]$$

Where m_{elec} is the electrical torque applied by the motor, m_{mec} is the mechanical torque applied by the load, J is the rotational inertia of the whole system, and $w_{mec}$is the mechanical speed. In order to increase the mechanical frequency by dw_{mec} over a finite time interval, the electric torque must also increase. The increase in electric torque is inversely proportional to the time interval, and therefore the time interval must be carefully selected such that the power drawn into the motor does not exceed the power the system is capable of delivering.

Figure 12. Motor-water pump Flair America SP-8130

3.5 Water housing

To make the demo self-sufficient at the same time that representing the idea of storing mechanical energy a double tank water housing was designed and built where water flowed upwards from the water pump to the second tank and then back to the first one to not overflow it.

Figure 13. Water housing unit

3.6 Sensing

Sensing of power involves taking measurements at various locations of the circuit. There are four types of power sensing: DC voltage, DC current, AC voltage, and AC current. These values allow to calculate the input power from the solar panel as well as the output power to the motor. Initially, it was planned to implement these types of sensing, but it was not successful due to low resolution of the sensors used. When trying to integrate the sensing into the system, current changes were not being able to be measured because they were in mA range. Nevertheless, the code to implement this sensing in the control algorithm and display it is provided in Appendix C.

Because current sensing was not attained for the prototype, instead a photoresistor was used to estimate the intensity of sunlight incident on the solar panels. This did not allow to measure the input and output power, and as such did not allow accurate power consumption measurements. It did, however, allow to control the motor frequency and voltage based on the incident sunlight.

3.7 Software

In each of the modules mentioned thus far, it is specified a mandated level of control. A microcontroller is used to provide this control and determine the behavior of the system. The environmental and inherently variable conditions to which the system is subjected requires to build a robust and logic-oriented solution. Using a microcontroller allows to obtain precise behavior with the flexibility needed to ensure stable operation. Starting from the DC-DC converter, the pulse train, whose duty cycle determines the electric potential between the output terminals, is provided from the microcontroller. Special thanks to Dr. Valvano for allowing to use his code for the PWM, bus frequency, interruptions and display that is provided as well in appendix C.

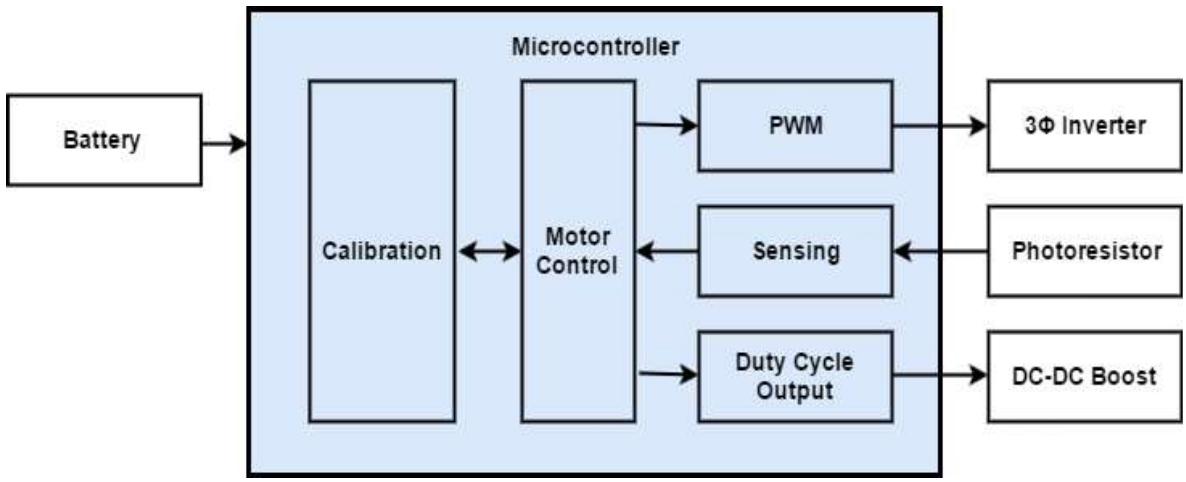

Figure 14. Software Block Diagram for Microcontroller Function

The software that controls the system runs on a TI-TM4C123GXL microcontroller, which was chosen because of the familiarity with it from the embedded systems class. As seen in figure 14, the primary modules are the PWM, duty cycle output, photoresistor sensing, motor control, and calibration. The main logic of the system exists in the motor control algorithm. The value sensed from the photoresistor determines the operating point of the system. The motor control algorithm then uses the other modules to operate the system at the desired operating point.

The motor control implemented was selected to fulfill these previous requirements. An open loop system, where the motor decides by itself at which speed is running at a fixed voltage, apart from being highly inefficient, it would inevitably stall. An analogy can be expressed in terms of the regular diesel motor used in cars. Whenever a car driver want to go from certain speed to another 50 mph higher, motors by default will need a time higher than 2 seconds to reach it. This is due to the fact that the diesel valve can just let in a limited quantity of fuel and the same with the pistons. In the electric world, the constraints for how much power can the motor consume is given by the power source that is feeding the motor and the current rating. However, for some events, the current rating can be surpassed before any damage occurs to the motor or a protection device starts operating. On the other hand, the power source that normally feeds a motor is the electric grid. This enables the motor to never be constraint by the power source since the electric grid can provide infinite power. The application is unique in the fact that the power source is the constraint for how much power can be delivered to the motor. The solar panels, as shown in figure 4, for a certain level of solar insulation they can provide a limited amount of power. The problem arises when the light level changes suddenly to a brighter one, and the motor follows it with an instant increase in mechanical speed. To achieve this instant change in mechanical speed, following equation 15, the motor will need an immense amount of power that the solar panels cannot provide and will stall. For that purpose a motor control is needed.

The two motor controls that were discussed were: DQ vectorial control and V/f control.

DQ vectorial control, as depicted in figure 15, is normally use to have an accurate control of the torque as is needed in amuzement park rides. However, in the application it was not needed. Another negative characteristic of this model is that it required many data from the motor that had not been provided by Flair America. Finally, as explained previously, the DC current sensing had not been proven to be sensible enough and it was crucial for this type of control as well as the speed encoder that was not intended to integrate in the product to not make it overly expensive.

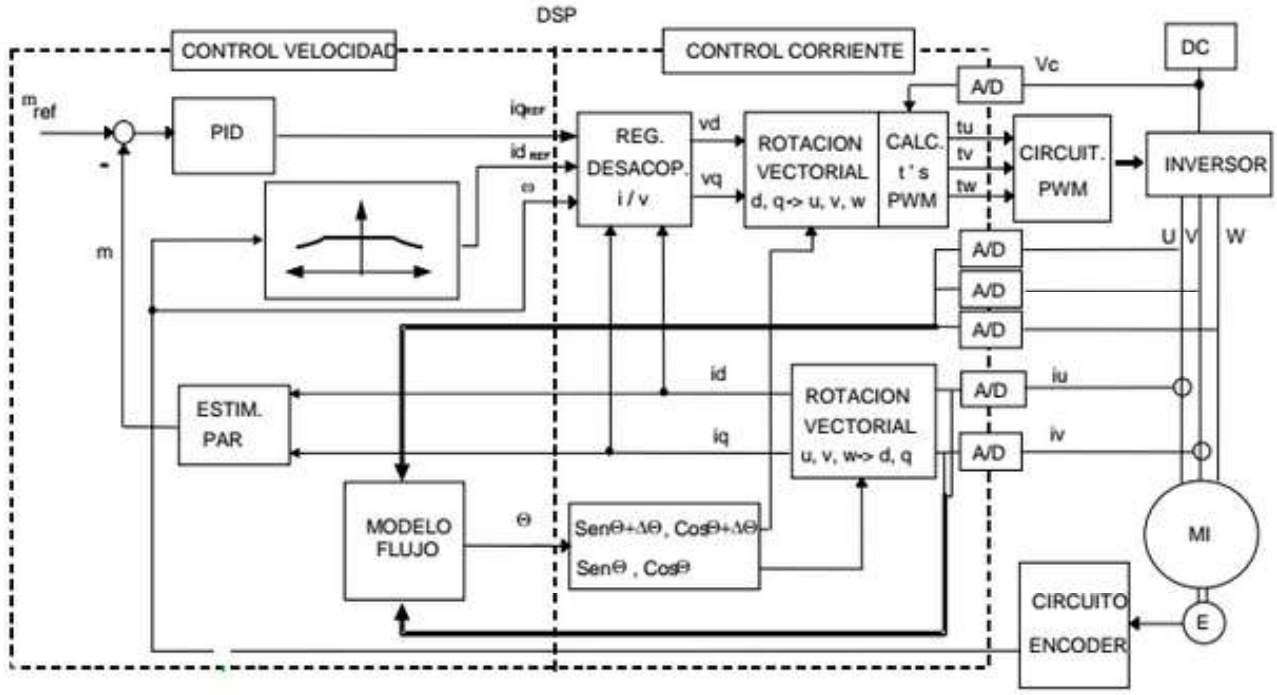

Figure 15. DQ torque control scheme for induction machine [11]

Therefore, the final decision was to implement the V/f control. Within V/f Control, there is a scheme that took into account slip, figure 16, and the one that does not, figure 17. The advantage a priori of the last one is that in did not need any data from the motor apart from the voltage rating. In this case, with the SP-8130 motor (water-pump) the only data available was: 1/8 HP power, 230 V voltage, 0.43 A current and 60 Hz frequency [13].

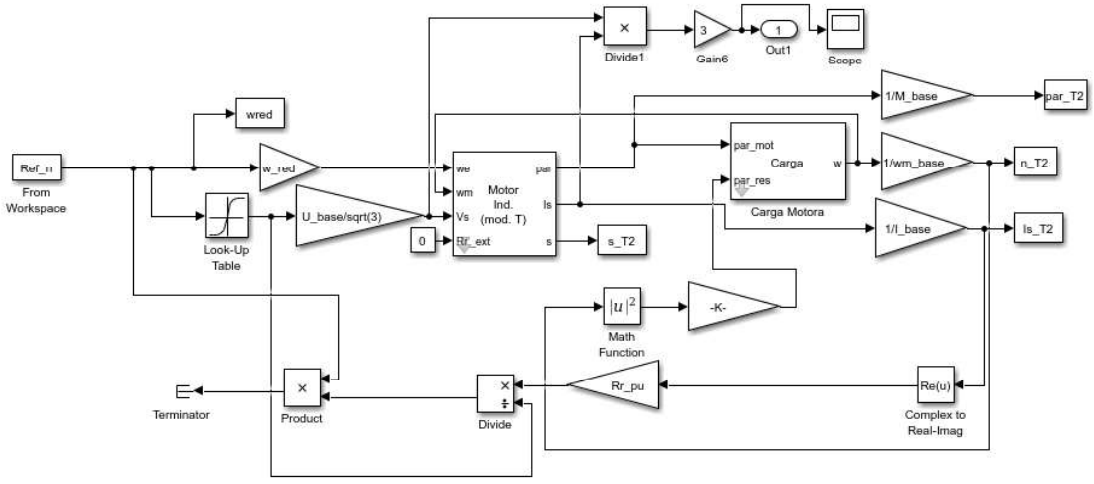

Figure 16 Simulink V/f control without slip [11]

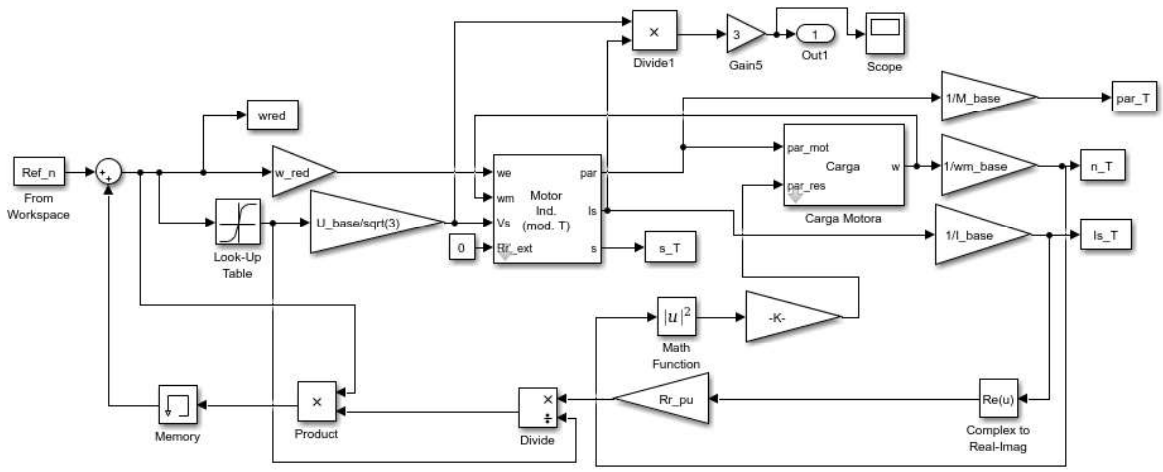

Figure 17. Simulink V/f control with slip [11]

To compare these two schemes they were simulated in MATLAB Simulink a ramp of speed to see what was the difference between both schemes. It can be observed in figure 18 how the error for the speed is acceptable for the application and the changes in torque and current are smoother.

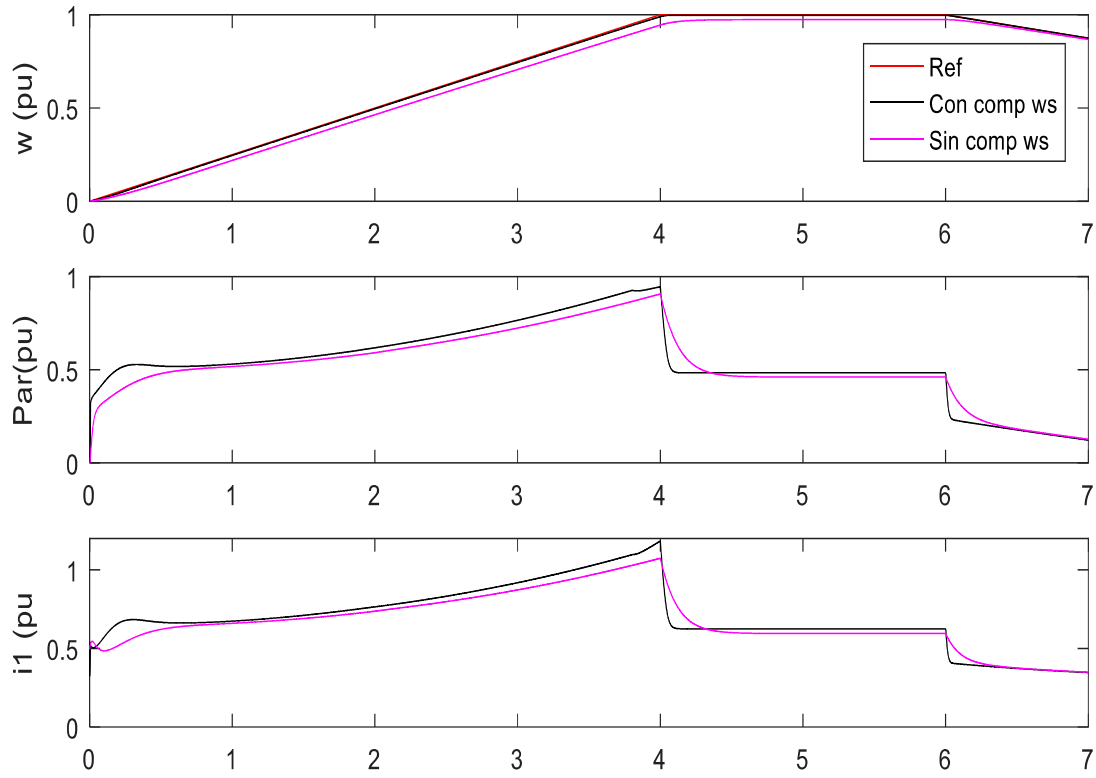

Figure 18. Open-loop V/f control with and without slip simulation in Simulink

To apply this algorithm to the usual V/f motor control [11] it was needed to add another block. The idea behind it was that the energy had to be transmitted instantly and as efficient as possible. Figure 19 shows the motor control loop where the additional block is the one that relates the power to the frequency. In this case, it was approximated it by a quadratic function, since it is characteristic of water-pumps, that did not start and zero and the maximum was at the maximum level of solar insulation.

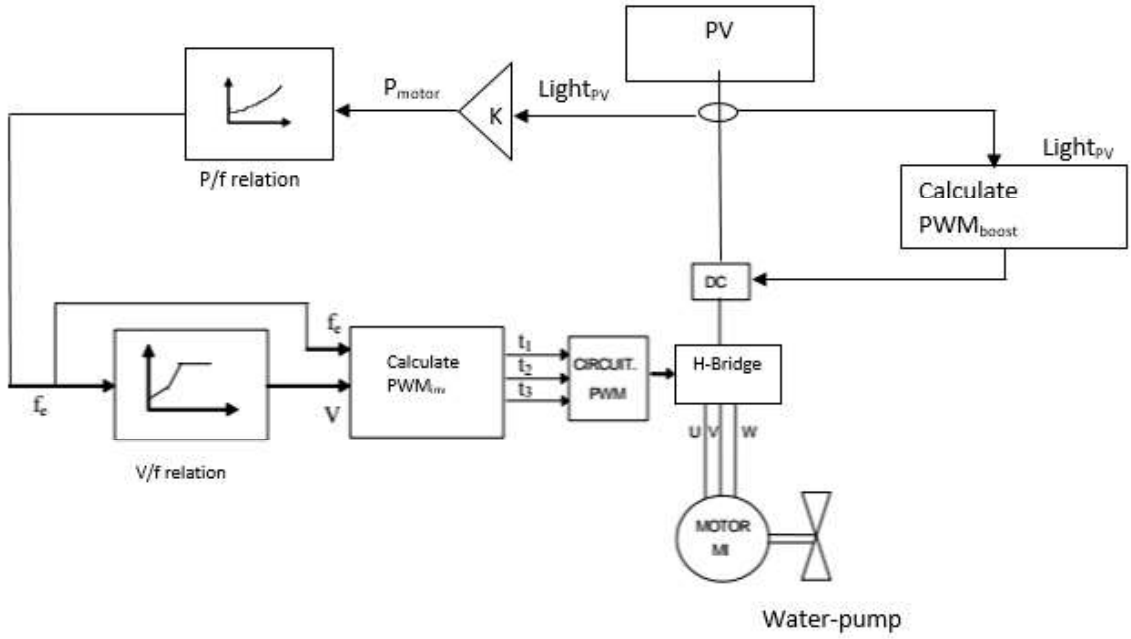

Figure 19. Motor control of the solar powered three-phase motor

The algorithm maintains a ratio between the voltage and frequency of the motor according to the operating point specified by the photoresistor. The calibration data is simply hard-coded to the specifications of the motor. The algorithm controls the rest of the system by outputting the desired frequency in the PWM module and the desired voltage to the boost converter as a duty cycle output. If the boost converter is unable to lower the output voltage enough to reach the desired operating point, the PWM module can also reduce the output voltage from the three-phase inverter by modulating the amplitude of the PWM signal.

Chapter 4: Subsystem testing and evaluation

The test and evaluation for the system begun by testing each submodule individually, followed by testing with adjacent submodules. This section details the individual testing for the solar panels, DC-DC boost converter, three-phase inverter, motor, and microcontroller, and finally, the full system.

4.1 Solar Panels

The solar panels were tested for their I-V and P-V relationships, as well as the open circuit voltage and short circuit current. These values were important for designing the voltage and current ratings for rest of the system. To begin, by-pass diodes were soldered onto the leads of the solar panels and configured the panels to be in-series. The nominal voltage was 88V open circuit, and the nominal current was 3.5A short circuit. Below, figures 20 and 21 show the I-V and P-V curves of 1 panel.

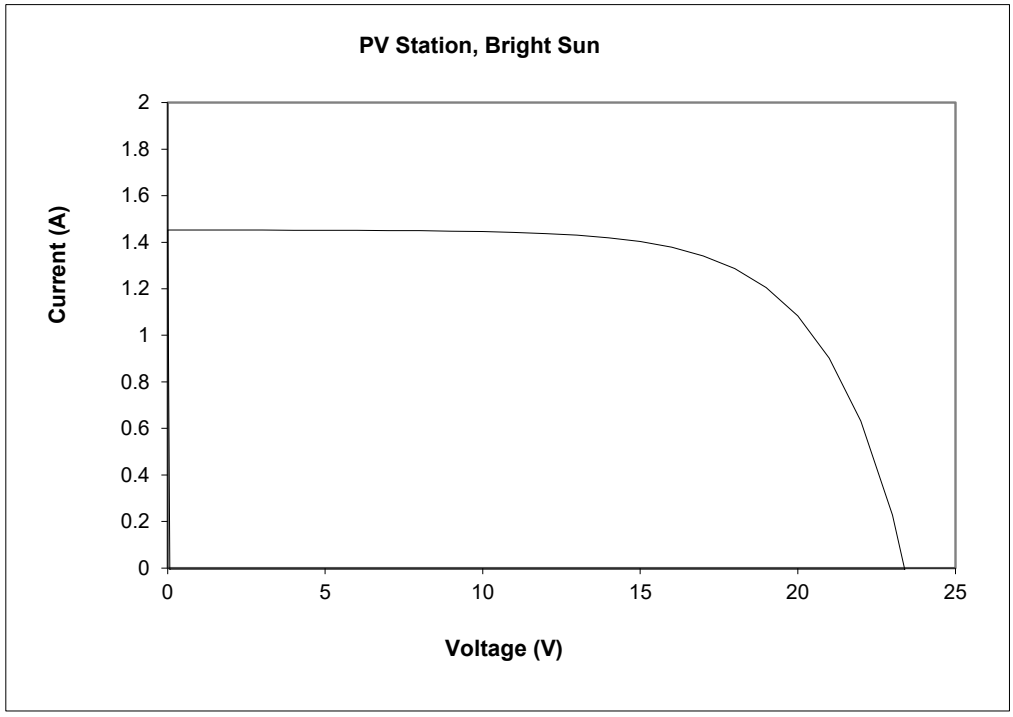

Figure 20. Current to Voltage Curve of One Solar Panel

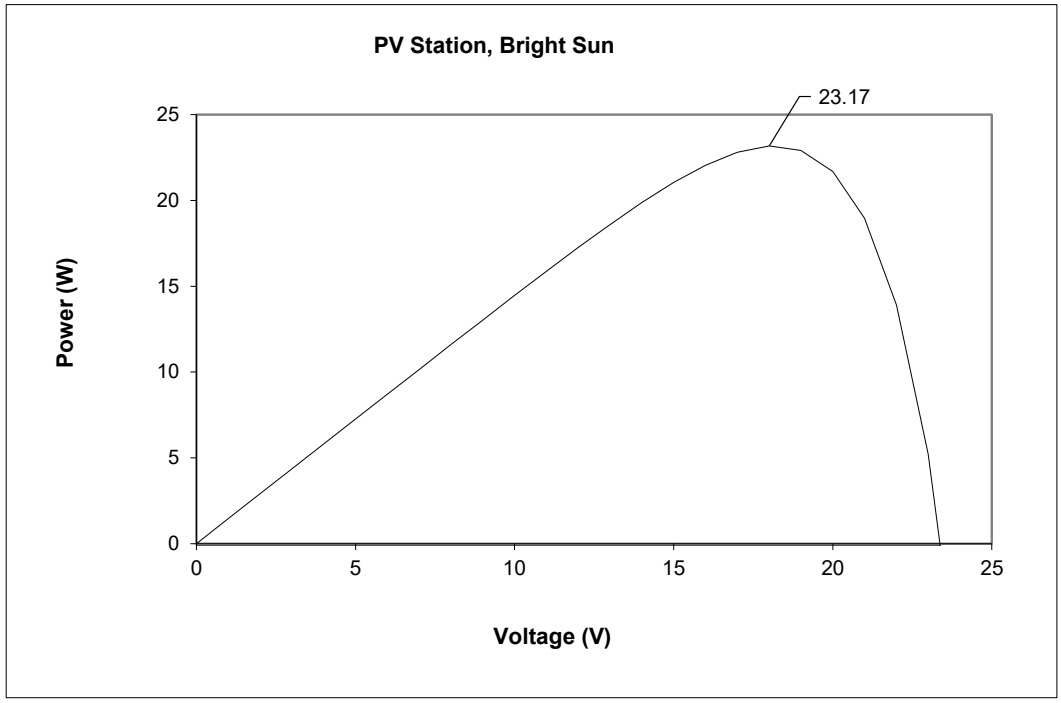

Figure 21. Power to Voltage Curve of One Solar Panels

In comparison to this panel, which had an open circuit voltage of 24V, two of the solar panels showed lower voltage levels around 22V.

A potential problem may be the individual construction of the solar cells into an array. The arrays were made 6 years ago and since the plastic screen that houses them are adhered to the frame, they could not be inspected without damaging the containment unit. Since they were not rigorously tested at the time for functionality and hazardous conditions such as spot shading, the solar cells may get hotter than their intended operation point. It is recommended that during operation, it is not allowed edge case conditions to occur, such as partial shading of the solar panel, or overheating of the cells. The demo focused on proof of concept of the entire system and using the solar panels with optimal functionality was of secondary importance.

During the demo, a gale pushed one of the solar panels and ripped out the leads from a solar cell, breaking it beyond repair. With electrical tape and some wires, the broken array element was by-passed and continued for the system to output 79V (DC).

4.2 DC-DC Boost Converter

The DC-DC converter boosts the solar panels output voltage to a level that can be used by the motor. A gate driver circuit is required to drive the MOSFET switching operations. The goal was to reach a boosting ratio that ranged between 1:1 and 4:1. For the initial testing of the boost

converter and the gate driver circuit, the gate driver successfully amplified a pulse width modulated (PWM) signal from 3.3V to 12V. The driver circuit was then attached to feed into the MOSFET gate signal of the boost converter. The boost converter did not produce the expected boost. For an input of 23V and duty cycle of 50%, the output voltage was 40V. The expected voltage output of the boost converter was around 45V accounting for the forward voltage drop of the diode and by using the standard formula for a boost converter given in equation 14.

It was also noticed that the boosting ratio was saturating at 2.5:1 when going above 50% duty cycle (D > 0.5). To successfully run the motor at its maximum power level this issue needed to be overcome and achieve a boosting ratio of 4:1.

After driving the circuit between 10% and 80% duty cycle while probing the drain to source and inductor voltage in the oscilloscope, it was realized some improvements needed to be made. The core issue was that the circuit was being driven to discontinuous mode at high voltage conversion ratios. This occurs when the inductor current waveform falls below zero. One thing that was not considered in the initial design was the DC amperage capacity rating for the inductor, which was later increased. The initial thought was that the inductance value for the input and control was too low, but after an experiment of connecting and combining several inductors in series, the circuit was still going into discontinuous mode. For the given input voltage and the time that the MOSFET remained closed, it was found that the magnetic dipoles in the inductor saturated before allowing the inductor to discharge into the circuit for its DC amperage capacity. The final decision was to use the inductor with the highest current ratio and cast aside the previous design that constrained the inductors in series to the minimum current rating. Furthermore, the output capacitor had to follow the change of replacing the inductor, because the voltage at which it was boosting now to was 290V (DC) with an input of 80V (DC). The initial capacitor used only had a 200V (DC) rating which exceeded the output value and would have broken the dielectric in the electrolytic capacitor, incurring in danger to surrounding people. As for the raise in capacitance, since the voltage increased from 180Vdc to 290V (DC), the capacitance needed to be higher to reduce the magnitude in voltage ripple. The results shown in figure 22 correlate with the fact that power losses are not being considered.

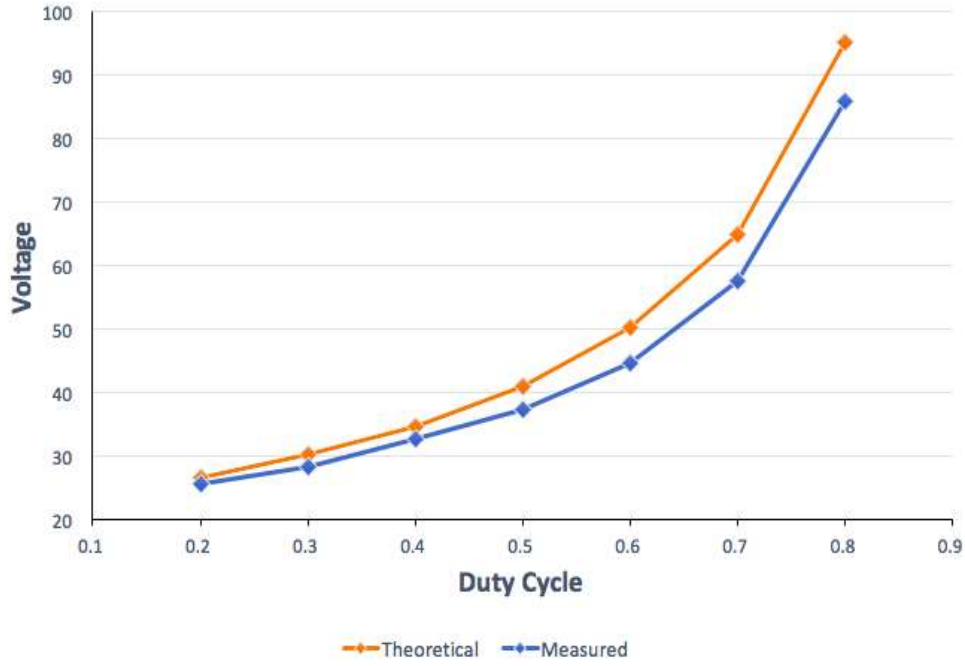

Figure 22. DC-DC Boost Operation Voltage vs. Duty Cycle

4.3 Three-Phase Inverter

Testing for the three-phase inverter first involved a lot of testing and debugging to make designs for the own three-phase inverter to work. After failing to get the original three-phase inverter design to work properly, an evaluation inverter, the STMicroelectronics STEVAL-IPM10F [12], was purchased that was fully completed and provides the exact functionality needed for all three phases of AC output. To test the board, it was first ensured that the microcontroller PWM was functioning correctly for all three phases. A 68uF capacitor was attached at the DC input of the PCB to maintain stability at the maximum output voltage. The board was then tested using all three phases of the PWM from the microcontroller, with an input voltage of 20VDC from a voltage generator in the lab. At first, establishing the necessary connections from the microcontroller to the evaluation board was troublesome, but after careful examination of the data sheet, it successfully generated the output from the three-phase inverter at the frequency of choice. The output was, after low-pass filtering (a reasonable assumption because of the inductance of the motor), three sine waves at the frequency specified by the PWM output of the microcontroller (20Hz and 30Hz). To complete the testing of the three-phase inverter, the three output phases were connected to the motor and ensured that the motor spun, both dry and in water. The motor pumped water through a hose to a height of 2ft at 20 Hz and an input voltage of 70VDC. This ensured the proper functionality of the three-phase inverter when provided with the input DC voltage from the boost converter.

It is unclear why the original designs for the three-phase inverter did not work despite the

efforts to debug it, and the decision to order the completed board was in the interest of time. This was a good decision because it resulted in a reliable subsystem and were able to work on integrating it into the rest of the system. After breaking one three-phase inverter while testing the whole system with the motor and DC-DC boost converter, the backup PCB had to be used. To prevent the same problem from recurring, the correct connections were ensured with the pins on the microcontroller by documenting the pin connections for both the three-phase inverter and the required connections on the microcontroller. The PCB was also isolated from the motor, water, and other electronic subsystems to avoid short-circuiting any connections. It was always purchased an extra three-phase inverter board to keep on-hand to avoid project delays.

4.4 Sensing

The photoresistor used for sensing the level of solar insolation was tested by connecting it to a voltage divider circuit with a potentiometer. With the full system in place, the photoresistor was calibrated by adjusting the potentiometer value until the maximum operating point was achieved under maximum lighting conditions, and shading the sensor resulted in a reduction of the operating point but not stalling.

4.5 Motor (Water-Pump)

It was aimed to make sure that the three-phase motor received from the manufacturer had no anomaly during its operation, that is, no discrepancies between the rated input voltage to output mechanical power. To test the motor, it was needed to input a three-phase AC voltage at the motor terminals and immerse the water pump, attached to the motor, into the water. The testing components used were, apart from the motor, a diode bridge rectifier and an isolation transformer plugged into the wall to convert a single-phase AC waveform into a DC voltage. This is then used by the three-phase inverter to convert it to the three-phase AC voltage needed by the motor. On the other side of the motor, the mechanical system was formed by a hose and two water tanks intended to show the elevation of water from one height to another. The motor-water pump was mounted on top of the first tank, properly isolating the motor side from the water, and the hose came from the motor to the second tank.

A critical factor for this testing was safety, since it consisted in pumping water near electrical conductors and power sources. In the electrical side of the test, the voltage was set below 70 V, in the output of the DBR and the current was not above 0.5 A. These values are the ratings for the DBR and the Motor respectively. On the mechanical side, the water tanks were correctly isolated from the electrical components, setting a safety distance of 5 feet.

Regarding the actual operation of the test, the voltage was set to 90V at the output of the DBR, the frequency to 20 Hz, and the height of the second tank was changed from 0 to 2 feet, to see the change in current and water flow. The results were measured from the output side of the

DBR, DC current and voltage. From which the power at the water pump side can be calculated as follows:

$$P_{wp}^{m} = V_{dc} I_{dc} \eta_m \eta_{inv} \quad (3)\ [14]$$

Efficiency of the motor and the inverter are given by their respective specifications [13][12]. The theoretical power consumed by the water pump is:

$$P_{wp}^{T} = \frac{Q * h * g * \rho}{\eta_{wp}} \quad (4)\ [14]$$

This is the mechanical power consumed by the water pump. The efficiency, ηwp, of the water-pump is extrapolated from the manufacturer specifications [13]; gravity (g) and water density (ρ) are known constants. Therefore, for the experiment, the water flow (Q) was regulated controlling the frequency, and the height, adjusting the second water tank. The results are compared in the following chart:

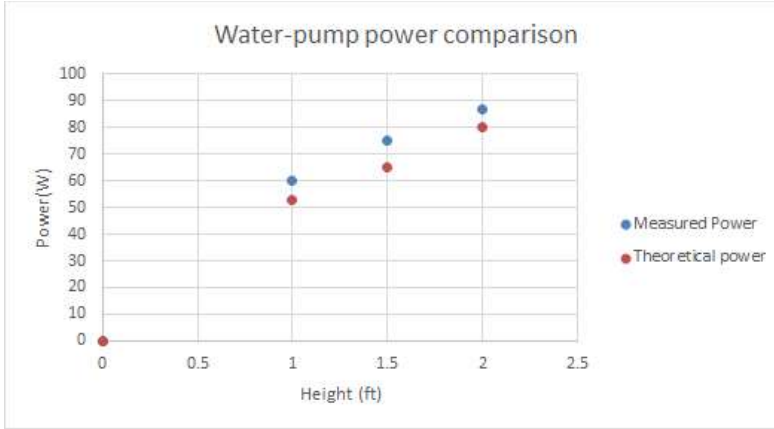

Figure 23. Measured Power and Theoretical Power Consumed by the Water Pump

It can be observed how there is a 10% error between measured and theoretical power. This is because a fixed efficiency for the motor is being used for the calculations. However, running at a different speed and voltage to what is rated it can be expected not to be this same value. A more accurate calculation could be made through the T-model of the motor [11]. However, the specifications of the motor are not enough to calculate all its parameters.

In conclusion, the motor provided the expected power and is a reliable component that proofed to work during the whole project. A recommendation for the future would be to avoid letting the motor stay still after stalling because in this condition the motor acts as short circuit drawing a lot of current through its windings. Instead, the motor should be restarted immediately, manually if it was necessary.

4.6 Water housing

The water-housing test simply entailed checking watertight seal between tubes and water tank and no overflowing. For this purpose, the motor-water pump was run to check for any these problems. The result was positive and no further development was needed.

4.7 Software

The software portion of the project consists primarily of a motor control algorithm that uses PWM, converter duty cycle, and sensing modules to control the operation of the entire system. Each module was tested individually and in conjunction with the full system to verify correctness. The design settled on has some drawbacks which are addressed in each section with suggestions for viable solutions.

4.7.1 PWM

The output characteristics of the PWM module can be tested using an oscilloscope. For the module to be correct, it must output three phases of positive and negative PWM at frequency *f* with at least 500 ns of dead-time at a switching frequency of 10 kHz. In addition, the amplitude modulation should not exceed 0.86 to avoid overmodulation. To test the module, it was first confirmed that the duty cycle has the proper switching frequency using the frequency measurement. The cursors were used to ensure that the dead-time between the positive and negative duty cycles is at least the minimum required by the three-phase inverter. It was attempted to over modulate the output and confirm that the amplitude of the output does not increase. Using the filtering feature, that each phase was confirmed to be oscillating at the desired frequency *f*. Finally, by probing each combination of two filtered outputs simultaneously, it was checked that each phase is offset by 120 degrees from the others. With the oscilloscope the above conditions were verified and the PWM module satisfies all of them. It was complete and worked as expected.

4.7.2 Converter Duty Cycle

The duty cycle output can be tested similarly to the PWM module using an oscilloscope. For it to be correct, the module must simply output a duty cycle *d* at a 30 kHz switching frequency. The software should limit *d* to be between 0.1 and 0.85 according to the safe operating range of the DC-DC converter. To test the module, the frequency was measured and confirmed that it was correct for the DC-DC converter. Then, the duty cycle *d* was verified to be indeed outputted by the module. Finally, it was attempted to exceed the safe range of duty cycles and confirm that the software does not allow it. With the oscilloscope the above conditions were

verified and the duty cycle module satisfies all of them. It is complete and works as expected.

4.7.3 Motor Control

The motor control algorithm can be tested using an oscilloscope and voltage source. By varying the input sensing from the photoresistor using the voltage source, it can be observed the output PWM frequency, amplitude, and converter duty cycle on the oscilloscope. The output PWM frequency and converter duty cycle must correspond to the motor operating point given the level of solar insolation detected by the photoresistor. For instance, when the input sensing reads its maximum value, the output PWM frequency must be the nominal frequency of the motor and the converter duty cycle must yield the nominal voltage of the motor after being converted to AC by the three-phase inverter. The other operating points are dictated by the motor power vs. voltage and voltage vs. frequency relationships. If the boost converter is unable to reach a low enough voltage for the appropriate operating point, the output PWM amplitude should decrease to yield the correct voltage at the output of the three-phase inverter. Likewise, if the boost converter is unable to reach a high enough voltage for the appropriate operating point, the output PWM frequency must not increase past the frequency corresponding to the highest voltage attainable. The motor control algorithm was tested in the lab and found that it satisfied all requirements. With the full system connected, the algorithm behaves as expected.

4.7.4 Full System

After testing each subsystem, a test bench was built in which the modules were mounted for ease of setup and use. A diagram depicting the layout of the test bench can be found in figure 24 below as well as the physical test bench in figure 25.

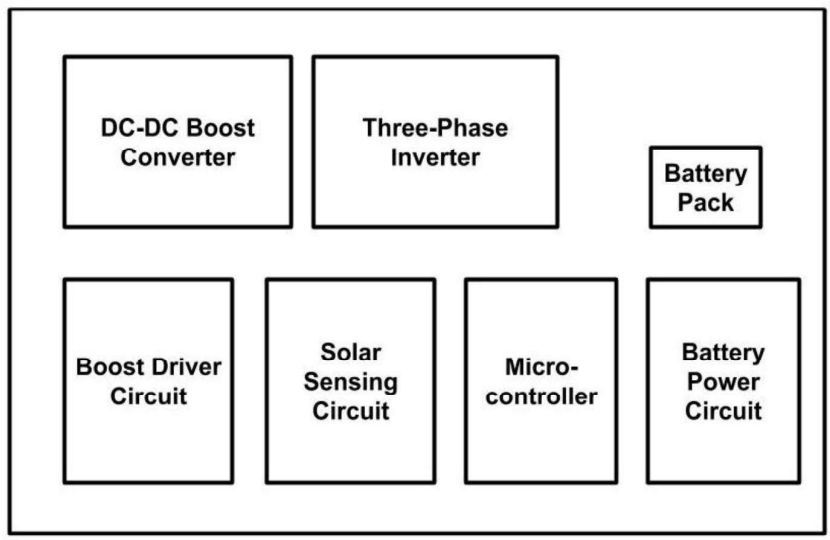

Figure 24. Test Bench Diagram

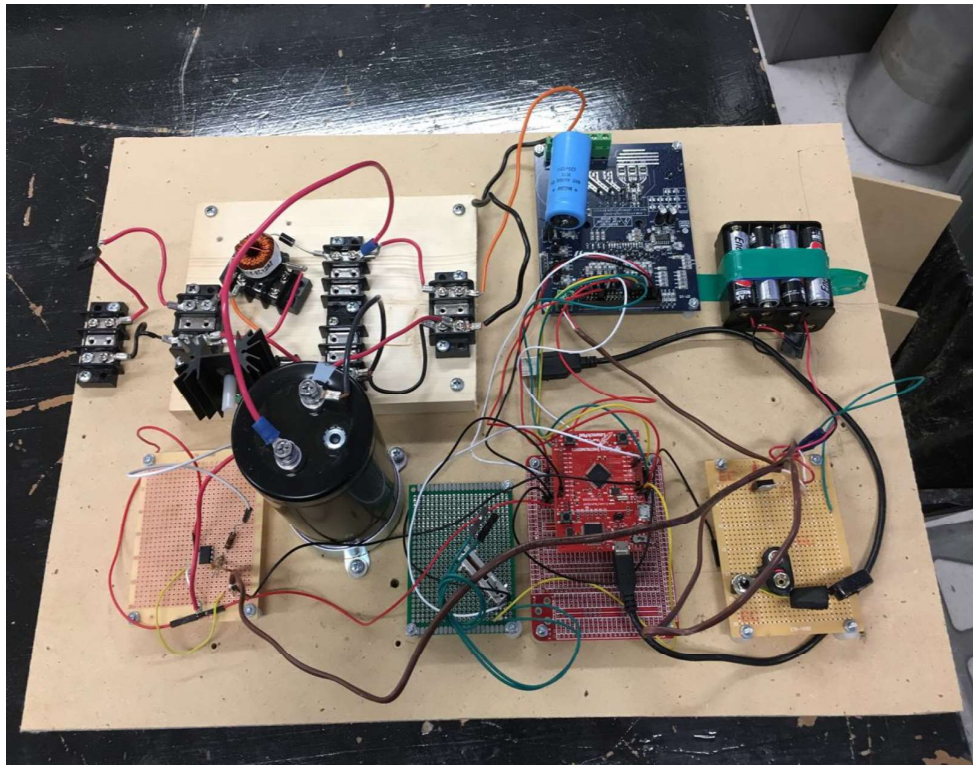

Figure 25. Test Bench

The full system was tested indoors using the DC power sources in the lab and successfully pumped water to a height of 7 ft. The entire system was then taken outside and successfully pumped water to a height of 5 ft. under maximal solar insolation. In figure 26, one of the panels was partially shaded with the photoresistor and observed that the output frequency of the motor decreased, but the motor did not stall. Upon unshading the panel and sensor, the motor resumed operation at full speed. This was considered a successful test of the complete system.

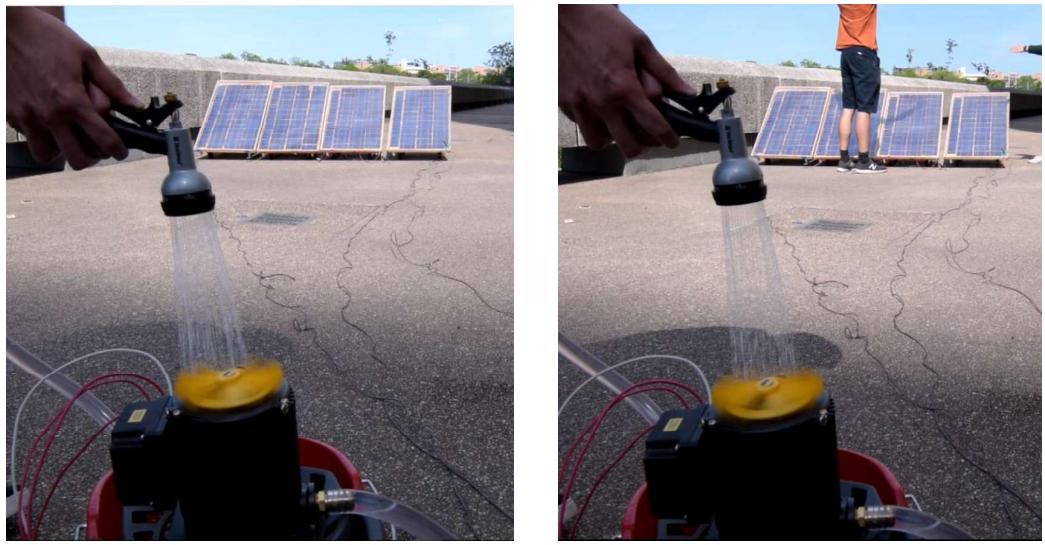

Figure 26. Change on flow rate due to manual fluctuation on sunlight avoiding stalling

Chapter 5: Time and Cost Considerations

Time was the most limiting constraint for the project, even after thoroughly following the work break down and Gantt Chart set at the start of the year, present in appendixes D and E. Yet cost was a big concern because of the desired accessibility of the design. Time spent ordering and awaiting new parts that were properly rated for the designs halted progress during both the design of the three-phase inverter and the boost converter. As designing and redesigning portions of the subsystems, there were required new parts that were rated for specific voltages based on the power provided by the solar panels, the desired boost level, and the operating point of the motor. The designs were changed for multiple systems and had to wait for new parts, which stalled the progress of the whole project. For example, the motor could not be tested without having a functional three-phase inverter. Because the system was based on so many essential subsystems, waiting for any one part pushed back the progress of the entire system.

This project was intended to be accessible and affordable to anyone who has a need for off-grid power, so the components ordered simplified the implementation and lowered the cost. This was a challenge because most of the parts needed to be rated for high voltage, current, and power levels, which generally cost more. Overall, the total prototype, excluding the motor and the solar panels, cost about $200. It was, however, consider safety a priority over cost, and bought parts that were more expensive, though still affordable, if they were more appropriately rated for a specific subsystem. For example, the decision to buy a fully functional three-phase inverter with all three phases integrated into one chip significantly sped up the process of full integration, and lowered the cost compared to buying the individual components. This allowed to put more time and effort into designing the boost converter and other subsystems. In general, safety was prioritized over cost concerns, as described in the next section.

Chapter 6: Safety and Ethical Aspects of the Design

As the project is meant to bring power to off-grid and underserved locations, design was intended to be as safe and sustainable as possible. Taking this into account, components were chosen with voltage and current ratings that were about twice the expected values. For example, the DC-DC boost converter required a capacitor that needed to handle 330V, and there were ordered capacitors rated for 600V. To add to the safety features, a test bench on which to stabilize each component of the system was constructed. The optimal layout for the test bench setup was one that required the shortest wire lengths and minimized crossing connections. This feature sped up the testing setup and kept the work area organized, decreasing the possibility of misconnections. A cover to keep any water from falling on the electrical components was included, as well as any people from touching the components.

In addition, there is an emphasis in the potential for sustainability with the project, especially that in underserved locations. As the system is off-grid and would, with more development, be completely self-sufficient, it has major implications in the future of power production using solar power. This project could be used to pump water into a water tower, irrigate a field, or provide air conditioning in locations where such luxuries would be unattainable. Thehope is for this project, in the future, to be open-source, affordable, and safe for anyone to construct if desired. Considering this, everything was documented so that the future work on this project will have the full picture of the progress, as well as how best to proceed from here to implement maximum power point tracking and therefore maximize the efficiency of the system. In conclusion, this system is intended to be safe, easy to use, and self-sufficient for anyone who might need to use three-phase power without a grid.

Chapter 7: Recommendations

There are several points of failure that are expected and inherent to the implemented design. The following problems with the motor control algorithm, listed in general order of importance, can be fixed to improve system performance and comply with the requirements specified on Appendix B. First, rather than using a photoresistor as a first-order approximation for the power available from the solar panels, DC sensing should directly detect the voltage of the solar panels and the current drawn by the system. This would also allow for a Perturb and Observe implementation of MPPT. During the research phase, shunt monitors, isolation amplifiers, and hall effect sensors were examined to measure current. Hall effect sensors combined with a large gain op-amp proved to be the most promising, but the other approaches may work as well. Second, a time constant should be added to prevent the control algorithm from varying the speed of the motor too quickly and possibly causing a stall. Third, a startup procedure should be implemented so that the motor does not need to be started by hand. Such a procedure would involve operating at a fixed startup frequency and slowly increasing voltage so that the motor will start without such a high inrush current, which the solar panels are unable to provide. Fourth, the algorithm should detect stall conditions using sensing and shut down system output to prevent high current from running through the motor. The algorithm should then wait for an amount of time before attempting to restart the system using the startup procedure. Fifth, rather than using a simple quadratic approximation, the algorithm should use a measured power vs. voltage curve for the motor. This will ensure that the motor is operating at the most efficient operating point given the available power. Sixth, the control algorithm should sense the output DC voltage of the boost converter to close the control loop for the converter. Seventh, and finally, with AC power sensing, the algorithm could produce quantitative results for the efficiency of the system.

The entire circuit could also be compacted into a PCB to allow for a more marketable and presentable device. Once current sensing and MPPT are complete, the stretch goals for the project could then be attempted. A user interface could be implemented with the system so that the system can take in solar and motor specs and run the system without the user having to change any of the code. Lastly, DC voltage regulators could be included to provide power to the gate driver circuit, three-phase inverter, and microcontroller directly from the solar panel, eliminating the need for batteries.

To a similar project that would attempt to do the same system, there are many recommendations in order to not fall into the same struggles or fails. Firstly, it is greatly recommended that technical work is started in the first semester. Upon Dr. Baldick's request, that was the mandate followed this semester and found that the research and technical understanding of the work done the first semester allowed to realize that a new implementation circuit was needed to be built from scratch. It is also suggested incorporating some sort of overcurrent protection. In the rush to finish the design, the inclusion of fuses was overlooked to open the circuit in case of high currents. When purchasing components, purchase more than the necessary amount, especially MOSFETs, as they are very likely to break from static discharge, melt from high

temperatures, or break over time. Lastly, not shy away from buying existing solutions to subsystems. After two months developing a three-phase inverter before it was found a fully implemented PCB that was already on the market and integrated all three phases, providing all the functionality needed. In summary, most of the recommendations have to do with reducing the risk due to long timelines for development and ordering parts.

Chapter 8: Conclusion

To conclude, an off-grid system was designed capable of delivering solar power to a three-phase load. A self-sufficient system that runs completely off-grid and modulates its operating point based on changes in solar insolation was successfully designed. Using a photoresistor to estimate the changes in incident sunlight, the microcontroller varies the voltage and frequency driving the motor to prevent the motor from stalling due to insufficient input power. In this regard, Dr. Baldick's requirements were successfully met. However, changes to the initial design were forced to guarantee baseline functionality that in the future could be improved. For example, the decision to install the photoresistor instead of the sensing circuitry fell short of the initial plans to implement motor control based on accurate power sensing. However, implementing MPPT based on power sensing both at the output of the solar panels and the input to the motor, to maximize efficiency, would be a reasonable next step. Keeping safety and sustainability a priority, the system designed could have major implications for off-grid power applications with small improvements such as overcurrent protection and a user interface. Ultimately, the project is hoped to be open sourced to facilitate remote power to under-developed communities and disaster relief efforts.

References

[1] S. Sengar, "Maximum Power Point Tracking Algorithms for Photovoltaic System: A Review," *Int. Review of Appl. Eng. Research*, vol. 4, no. 2, pp. 147-154, 2014.

[2] H. Patel, V. Agarwal, "Maximum Power Point Tracking Scheme for PV Systems Operating Under Partially Shaded Conditions," *IEEE Trans. Ind. Electron.*, vol. 55, no. 4, pp. 1689-1698, Apr. 2008.

[3] F. Khoucha *et al.,* "A Comparison of Symmetrical and Asymmetrical Three-Phase H-Bridge Multilevel Inverter for DTC Induction Motor Drives," *IEEE Transactions on Energy Conversion*, vol. 26, no. 1, pp. 64-72, Mar. 2011

[4] George Shu-Xing Cheng, Steven L. Mulkey, and Andrew J. Chow "Model-free adaptive (MFA) control with intelligent engine and loop inspection" U.S. Patent 8 594 813, November, 26, 2013.

[5] Adam Barroso et al., "Prior Art Report for Project MEEPA," Elec. Eng. Snr. Des., Univ. Texas, Austin, Tech. Rep., Apr. 2015.

[6] M. W. Earley et al., National Electrical Code Handbook. Quincy, Mass: National Fire Protection Association, 2011. Print.

[7] "Standard for Inverters, Converters, Controllers and Interconnection System Equipment for Use With Distributed Energy Resources", UL1741, 2010.

[8] "NEMA ICS 7.2-2015," National Electrical Manufacturers Association, 2015.

[9] M. Flynn, The University of Texas at Austin, Austin, TX, EE 462L: Power Electronics Laboratory course, Spring 2017.

[10] "LM5022 Data Sheet" TI. [Online]. Available: http://www.ti.com/lit/ds/symlink/lm5022.pdf LM5022 [Accessed: 23-Mar-2017].

[11] Lukas Sigirist, Universidad Pontificia de Comillas, Madrid, Spain. Electric Drives, Spring 2016.

[12] "UM2019 User manual", www.st.com, 2017. [Online]. Available: http://www.st.com/content/ccc/resource/technical/document/user_manual/group0/5a/6c/0d/5f/b0/c4/43/00/DM00266643/files/DM00266643.pdf/jcr:content/translations/en.DM00266643.pdf. [Accessed: 07- Apr- 2017].

[13] "Filterable Cooling Pumps," Flair America. [Online]. Available: http://flairamerica.net/pumps/filterable-cooling-pumps/. [Accessed: 06-Apr-2017].

[14] E.G.Strangas, “Notes for an Introductory Course On Electrical Machines and Drives” MSU Electrical Machines and Drives Laboratory. [Online]. Available: http://www.egr.msu.edu/~fzpeng/ECE320/ECE320-Notes-Part1.pdf

Appendix A: Relevant Standards

Article 690 from NFPA within the National Electric Code [2] specifies that the dc voltage within a PV system shall be permitted to have voltages of 600 volts or less. Both the dc circuit and the inverter output must have overcurrent protection. This also serves the purpose of protecting the panels from high current source connections. The maximum current shall be calculated by summing the parallel PV module rated short circuit currents and multiplying by 1.25. PV systems which operate at 80 volts dc or greater must be protected by an arc-fault circuit interrupter or implement some other measure to protect against an arc-faults. Single-conductor cable Type USE-2 is allowed for use within outdoor locations which are exposed. If the system is to be permanently connected to the PV source, it must be labeled in accordance with 690.31(G)(3). The system must have WARNING PHOTOVOLTAIC POWER SOURCE written on all modules which remain connected to the solar panels. The connections must be polarized and incompatible with receptacles used by other electrical systems nearby. Connectors to the solar panels must be of latching or locking type. The grounding member of the connector must be have a first to make and last to break contact with its matching connector. UL-1741 certification [3] places requirements on the implementation of photovoltaic systems to keep the user from electrical harm. To decrease the risk of reaching an unstable electric condition, the energization time for dc circuits must be under two seconds. ANSI/NFPA 70 [2] requires that in use cases of varying environmental factors such as humidity and temperature, electrical devices must be isolated by an enclosure preventing electric arcing.

The motor is expected to be connected to the system to comply with NFPA 70 - Articles 430 & 440 by National Electric Manufacturers Association (NEMA). Motors which operate at 50 or 60 Hz shall have nameplate marking to indicate as such. There shall be a connection either on the housing itself or from the motor wiring which allows for an external device to access the grounded conductor of the motor. The motor shall have thermal protection such that the rated current is not exceeded by 170%.

Appendix B: Tables

Table 1. Operating Environment Specifications

Operating Conditions	Description	Specification
Weatherproofing	Housing of electronics must be waterproof, resistant to erosion, dust, and wind	System will be able to run in rainy conditions. Is expected to survive 10 years, outside.
Temperature (external)	System must operate in outdoor environments	0-40° C
Temperature (internal)	System must not overheat internal electronic components	0-100° C Determined by components chosen for final design and their temperature tolerances; May require internal cooling fans or a heat sink
Humidity	System must operate in outdoor environments	0-100%
Theft-proof	Electronics must be housed in a unit that prevents tampering	All internal electronics must be covered and tightly sealed
Vibration proof	System should be able to be moved without damaging the internal electronics	Housing and parts will be sealed together to minimize internal movement and reduce the effect of internal torque generated from the motor.

Table 2. Performance Specifications

Requirement	Description	Specification
Operate off-grid	System must operate independently of any power from the electrical grid	All controls and the load must draw their power from the solar cells
Adapt to variation in light level	System must be able to change its operating point to compensate for increases and decreases in power from sunlight	Maximum Power Point Tracking of the solar panels; Perturb-and-observe

Operate without motor stall	Upon a decrease in power provided from the sun, the system must not allow the motor to stall	Reactive system that is capable of controlling the variable frequency motor
Calibrate unknown solar cell	System must be able to connect to a solar cell and determine a starting point on the I-V curve of the panel	Starting point calibration should be within 10% of the Maximum Power Point
Calibrate unknown motor load	System must be able to connect to an unknown motor and operate it	S parameters of motor will be input via UI; Calibration will sweep the frequencies and voltage operating points of the motor
Self-sustaining power	System should not need to be powered by batteries or other power sources	Solar panel will need to power the microcontroller, as well as the motor

Appendix C: Microcontroller Code

1. PWM

```
// PWM.c
// Runs on TM4C123
// Use PB4, PB6, and PB7 to generate pulse-width modulated outputs.
// Max Granat

// Three phase PWM
// Phase A+/- on PB6/7
// Phase B+/- on PB4/5
// Phase C+/- on PF2/3

// Converter duty cycle on PC4

/* This example accompanies the book
   "Embedded Systems: Real Time Interfacing to Arm Cortex M
Microcontrollers",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2016
   Program 6.8, section 6.3.2

   "Embedded Systems: Real-Time Operating Systems for ARM Cortex M
Microcontrollers",
   ISBN: 978-1466468863, Jonathan Valvano, copyright (c) 2017
   Program 8.4, Section 8.3

 Copyright 2017 by Jonathan W. Valvano, valvano@mail.utexas.edu
    You may use, edit, run or distribute this file
    as long as the above copyright notice remains
 THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS,
IMPLIED
 OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS
SOFTWARE.
 VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL,
INCIDENTAL,
 OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 For more information about my classes, my research, and my books, see
 http://users.ece.utexas.edu/~valvano/
 */
#include <stdint.h>
#include <math.h>

#include "tm4c123gh6pm.h"

#define PI (3.141592)
#define PWM_DEAD_BAND (4)
#define PWM_PERIOD (500) // 250 is 20 kHz switching frequency
#define CONVERTER_PERIOD (167) // 250 is 20 kHz switching frequency
#define PWM_PADDING (PWM_DEAD_BAND + 3)
#define PWM_RANGE ((((double) PWM_PERIOD) - PWM_PADDING) / 2)
#define PWM_CENTER (PWM_RANGE + PWM_PADDING)
#define FREQUENCY_CORRECTION_FACTOR (1.36)
#define RANGE_FACTOR_MAX (0.866)
```

```
#define NUM_ELEMENTS (256)
#define NUM_ELEMENTS_DOUBLE (256.0)

uint32_t pwm_counter = 0;
double range_factor = RANGE_FACTOR_MAX;
uint16_t cos_table[] = {32768,33572,34375,35178,35979,36779,37575,38369,
39160,39947,40729,41507,42279,43046,43807,44560,
45307,46046,46777,47500,48214,48919,49613,50298,
50972,51635,52287,52927,53555,54170,54773,55362,
55938,56499,57047,57579,58097,58600,59087,59558,
60013,60451,60873,61278,61666,62036,62389,62724,
63041,63339,63620,63881,64124,64348,64553,64739,
64905,65053,65180,65289,65377,65446,65496,65525,
65535,65525,65496,65446,65377,65289,65180,65053,
64905,64739,64553,64348,64124,63881,63620,63339,
63041,62724,62389,62036,61666,61278,60873,60451,
60013,59558,59087,58600,58097,57579,57047,56499,
55938,55362,54773,54170,53555,52927,52287,51635,
50972,50298,49613,48919,48214,47500,46777,46046,
45307,44560,43807,43046,42279,41507,40729,39947,
39160,38369,37575,36779,35979,35178,34375,33572,
32768,31963,31160,30357,29556,28756,27960,27166,
26375,25588,24806,24028,23256,22489,21728,20975,
20228,19489,18758,18035,17321,16616,15922,15237,
14563,13900,13248,12608,11980,11365,10762,10173,
9597,9036,8488,7956,7438,6935,6448,5977,
5522,5084,4662,4257,3869,3499,3146,2811,
2494,2196,1915,1654,1411,1187,982,796,
630,482,355,246,158,89,39,10,
0,10,39,89,158,246,355,482,
630,796,982,1187,1411,1654,1915,2196,
2494,2811,3146,3499,3869,4257,4662,5084,
5522,5977,6448,6935,7438,7956,8488,9036,
9597,10173,10762,11365,11980,12608,13248,13900,
14563,15237,15922,16616,17321,18035,18758,19489,
20228,20975,21728,22489,23256,24028,24806,25588,
26375,27166,27960,28756,29556,30357,31160,31963};


// Initialize PWM modules
// Phase A+/- on PB6/7
// Phase B+/- on PB4/5
// Phase C+/- on PF2/3
void PWM_Init() {
        volatile uint32_t delay;

        SYSCTL_RCGCPWM_R |= 0x03;              // activate PWM0-1
        delay = SYSCTL_RCGCPWM_R;             // allow time to finish
activating
        delay = SYSCTL_RCGCPWM_R;

        // Initialize Port B
        SYSCTL_RCGCGPIO_R |= 0x02;             // activate port B
        delay = SYSCTL_RCGCGPIO_R;             // allow time to finish
activating
        delay = SYSCTL_RCGCGPIO_R;
        GPIO_PORTB_AFSEL_R |= 0xF0;            // enable alt funct on PB4-7
        GPIO_PORTB_PCTL_R &= ~0xFFFF0000;      // configure PB4-7 as PWM0
        GPIO_PORTB_PCTL_R |= 0x44440000;
```

```
        GPIO_PORTB_AMSEL_R &= ~0xF0;          // disable analog
functionality on PB4-7
        GPIO_PORTB_DEN_R |= 0xF0;             // enable digital I/O on PB4-
7

        // Set up PWM divider
        SYSCTL_RCC_R |= SYSCTL_RCC_USEPWMDIV; // use PWM divider
        SYSCTL_RCC_R &= ~SYSCTL_RCC_PWMDIV_M; // clear PWM divider field
        SYSCTL_RCC_R += SYSCTL_RCC_PWMDIV_16; // configure for /16 divider

        // Initialize PB6-7 on PWM0
        PWM0_0_CTL_R = 0;                     // disable PWM while
initializing
        // PWM0, Generator A (PWM0/PB6) goes to 1 when count==reload and 0
when count==CMPA
        PWM0_0_GENA_R = (PWM_0_GENA_ACTLOAD_ONE|PWM_0_GENA_ACTCMPAD_ZERO);
        // PWM0, Generator B (PWM1/PB7) goes to 0 when count==reload and 1
when count==CMPA
        PWM0_0_GENB_R = (PWM_0_GENB_ACTLOAD_ZERO|PWM_0_GENB_ACTCMPAD_ONE);
        PWM0_0_DBCTL_R =
1;
        // enable dead-band generator for PWM0
        PWM0_0_DBRISE_R = PWM_DEAD_BAND;                    // configure
rising edge dead-band for PWM0
        PWM0_0_DBFALL_R = PWM_DEAD_BAND;                    // configure
falling edge dead-band for PWM0
        PWM0_0_LOAD_R = PWM_PERIOD - 1;                              //
cycles needed to count down to 0
        PWM0_0_CMPA_R = (PWM_PERIOD - 1)/2;   // count value when PWM0
toggles
        PWM0_0_CTL_R |= PWM_0_CTL_ENABLE;                   // start PWM0
in count down mode
        PWM0_ENABLE_R |=
(PWM_ENABLE_PWM0EN|PWM_ENABLE_PWM1EN);        // enable PWM0

        // Initialize PB4-5 on PWM1
        PWM0_1_CTL_R = 0;                     // disable PWM while
initializing
        // PWM1, Generator A (PWM2/PB4) goes to 1 when count==reload and 0
when count==CMPA
        PWM0_1_GENA_R = (PWM_1_GENA_ACTLOAD_ONE|PWM_1_GENA_ACTCMPAD_ZERO);
        // PWM1, Generator B (PWM3/PB5) goes to 0 when count==reload and 1
when count==CMPA
        PWM0_1_GENB_R = (PWM_1_GENB_ACTLOAD_ZERO|PWM_1_GENB_ACTCMPAD_ONE);
        PWM0_1_DBCTL_R =
1;
        // enable dead-band generator for PWM1
        PWM0_1_DBRISE_R = PWM_DEAD_BAND;                    // configure
rising edge dead-band for PWM1
        PWM0_1_DBFALL_R = PWM_DEAD_BAND;                    // configure
falling edge dead-band for PWM1
        PWM0_1_LOAD_R = PWM_PERIOD - 1;       // cycles needed to count
down to 0
        PWM0_1_CMPA_R = (PWM_PERIOD - 1)/2;   // count value when PWM1
toggles
        PWM0_1_CTL_R |= PWM_1_CTL_ENABLE;     // start PWM1
        PWM0_ENABLE_R |=
(PWM_ENABLE_PWM2EN|PWM_ENABLE_PWM3EN);        // enable PWM1

        // Initialize Port F
```

```
        SYSCTL_RCGCGPIO_R |= 0x20;              // activate port F
        delay = SYSCTL_RCGCGPIO_R;              // allow time to finish
activating
        delay = SYSCTL_RCGCGPIO_R;
        GPIO_PORTF_AFSEL_R |= 0x0C;             // enable alt funct on PF2-3
        GPIO_PORTF_PCTL_R &= ~0xFF00;                   // configure PF2-3 as
PWM3
        GPIO_PORTF_PCTL_R |= 0x5500;
        GPIO_PORTF_AMSEL_R &= ~0x0C;            // disable analog
functionality on PF2-3
        GPIO_PORTF_DEN_R |= 0x0C;               // enable digital I/O on PF2-
3

        // Initialize PF2-3 on PWM7
        PWM1_3_CTL_R = 0;                       // disable PWM while
initializing
        // PWM3, Generator A (PWM6/PF2) goes to 1 when count==reload and 0
when count==CMPA
        PWM1_3_GENA_R = (PWM_3_GENA_ACTLOAD_ONE|PWM_3_GENA_ACTCMPAD_ZERO);
        // PWM3, Generator B (PWM7/PF3) goes to 0 when count==reload and 1
when count==CMPA
        PWM1_3_GENB_R = (PWM_3_GENB_ACTLOAD_ZERO|PWM_3_GENB_ACTCMPAD_ONE);
        PWM1_3_DBCTL_R =
1;
        // enable dead-band generator for PWM3
        PWM1_3_DBRISE_R = PWM_DEAD_BAND;                        // configure
rising edge dead-band for PWM3
        PWM1_3_DBFALL_R = PWM_DEAD_BAND;                        // configure
falling edge dead-band for PWM3
        PWM1_3_LOAD_R = PWM_PERIOD - 1;         // cycles needed to count
down to 0
        PWM1_3_CMPA_R = (PWM_PERIOD - 1)/2;     // count value when output
rises
        PWM1_3_CTL_R |= PWM_3_CTL_ENABLE;       // start PWM3
        PWM1_ENABLE_R |= (PWM_ENABLE_PWM6EN|PWM_ENABLE_PWM7EN);       //
enable PWM3

        // Initialize PC4 (converter duty cycle) on PWM3
        SYSCTL_RCGCGPIO_R |= 0x04;              // activate port C
        delay = SYSCTL_RCGCGPIO_R;              // allow time to finish
activating
        delay = SYSCTL_RCGCGPIO_R;
        GPIO_PORTC_AFSEL_R |= 0x10;             // enable alt funct on PC4
        GPIO_PORTC_PCTL_R &= ~0x000F0000;       // configure PC4 as PWM3
        GPIO_PORTC_PCTL_R |= 0x00040000;
        GPIO_PORTC_AMSEL_R &= ~0x10;            // disable analog
functionality on PC4
        GPIO_PORTC_DEN_R |= 0x10;               // enable digital I/O on PC4

        // Initialize PB6-7 on PWM0
        PWM0_3_CTL_R = 0;                       // disable PWM while
initializing
        // PWM3, Generator A (PWM6/PC4) goes to 1 when count==reload and 0
when count==CMPA
        PWM0_3_GENA_R = (PWM_3_GENA_ACTLOAD_ONE|PWM_3_GENA_ACTCMPAD_ZERO);
        PWM0_3_LOAD_R = CONVERTER_PERIOD - 1; // cycles needed to count
down to 0
        PWM0_3_CMPA_R = (CONVERTER_PERIOD-1)/2;   // count value when PWM3
toggles
```

```
        PWM0_3_CTL_R |= PWM_3_CTL_ENABLE;                    // start PWM3
in count down mode
        PWM0_ENABLE_R |= PWM_ENABLE_PWM6EN;           // enable PWM3
}

// Set duty cycle for phase A
void set_duty_a(uint16_t duty) {
        PWM0_0_CMPA_R = PWM_PERIOD - (duty - 1);
}

// Set duty cycle for phase B
void set_duty_b(uint16_t duty){
        PWM0_1_CMPA_R = PWM_PERIOD - (duty - 1);
}

// Set duty cycle for phase C
void set_duty_c(uint16_t duty){
        PWM1_3_CMPA_R = PWM_PERIOD - (duty - 1);
}

// Update range factor 0 <= f <= 1, factor of RANGE_FACTOR_MAX
void set_range_factor(double new_factor) {
        if (new_factor > 1) {
                range_factor = RANGE_FACTOR_MAX;
        } else if (new_factor < 0) {
                range_factor = 0;
        } else {
                range_factor = new_factor * RANGE_FACTOR_MAX;
        }
}

double lookup(double phase) {
        double normalized = phase / (2 * PI);
        int quantized = (int) (normalized * NUM_ELEMENTS_DOUBLE);
        return cos_table[quantized % NUM_ELEMENTS] / 32768.0 - 1.0;
}

// Advance duty cycles for all three phases
// FIXME issue when changing frequency, phase information is
// not retained, so changing frequency too fast can result in
// jumps in output, should be fixed for maximum efficiency
void PWM_tick(uint32_t timer_frequency, double motor_frequency){
        double divisor = timer_frequency / (motor_frequency);
        double phase = (((double) pwm_counter) / divisor) * 2 * PI;
        double offset = 2 * PI / 3;
        double sine_a = lookup(phase);
        double sine_b = lookup(phase + offset);
        double sine_c = lookup(phase + 2*offset);
        uint16_t duty_a = (uint16_t) (PWM_CENTER + (sine_a * PWM_RANGE *
range_factor));
        uint16_t duty_b = (uint16_t) (PWM_CENTER + (sine_b * PWM_RANGE *
range_factor));
        uint16_t duty_c = (uint16_t) (PWM_CENTER + (sine_c * PWM_RANGE *
range_factor));

        set_duty_a(duty_a);
        set_duty_b(duty_b);
        set_duty_c(duty_c);

        pwm_counter++;
```

```
        if (pwm_counter > divisor) {
                pwm_counter = 0;
        }
}

// Set converter duty cycle 0 <= d <= 1
void set_converter_duty(double duty) {
        double duty_period = CONVERTER_PERIOD - (CONVERTER_PERIOD * duty);
        if (duty_period < 3)
                PWM0_3_CMPA_R = 3;
        else if (duty_period >= CONVERTER_PERIOD - 1)
                PWM0_3_CMPA_R = CONVERTER_PERIOD - 2;
        else
                PWM0_3_CMPA_R = (uint16_t) duty_period;
}
```

2. Microcontroller frequency

```
// PLL.c
// Runs on LM4F120 and TM4C123
// A software function to change the bus frequency using the PLL.
// Daniel Valvano

/* This example accompanies the book
   "Embedded Systems: Real Time Interfacing to Arm Cortex M
Microcontrollers",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2014
   Program 2.10, Figure 2.37

 Copyright 2014 by Jonathan W. Valvano, valvano@mail.utexas.edu
    You may use, edit, run or distribute this file
    as long as the above copyright notice remains
 THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS, IMPLIED
 OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE.
 VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL,
 OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
*/
#include <stdint.h>
#include "tm4c123gh6pm.h"

#include "PLL.h"

// The #define statement SYSDIV2 in PLL.h
// initializes the PLL to the desired frequency.

// bus frequency is 400MHz/(SYSDIV2+1) = 400MHz/(7+1) = 50 MHz
// see the table at the end of this file

#define SYSCTL_RIS_PLLLRIS      0x00000040  // PLL Lock Raw Interrupt Status
#define SYSCTL_RCC_XTAL_M       0x000007C0  // Crystal Value
#define SYSCTL_RCC_XTAL_6MHZ    0x000002C0  // 6 MHz Crystal
#define SYSCTL_RCC_XTAL_8MHZ    0x00000380  // 8 MHz Crystal
```

```
#define SYSCTL_RCC_XTAL_16MHZ   0x00000540  // 16 MHz Crystal
#define SYSCTL_RCC2_USERCC2     0x80000000  // Use RCC2
#define SYSCTL_RCC2_DIV400      0x40000000  // Divide PLL as 400 MHz vs. 200
                                            // MHz
#define SYSCTL_RCC2_SYSDIV2_M   0x1F800000  // System Clock Divisor 2
#define SYSCTL_RCC2_SYSDIV2LSB  0x00400000  // Additional LSB for SYSDIV2
#define SYSCTL_RCC2_PWRDN2      0x00002000  // Power-Down PLL 2
#define SYSCTL_RCC2_BYPASS2     0x00000800  // PLL Bypass 2
#define SYSCTL_RCC2_OSCSRC2_M   0x00000070  // Oscillator Source 2
#define SYSCTL_RCC2_OSCSRC2_MO  0x00000000  // MOSC

// configure the system to get its clock from the PLL
void PLL_Init(void){
  // 0) configure the system to use RCC2 for advanced features
  //    such as 400 MHz PLL and non-integer System Clock Divisor
  SYSCTL_RCC2_R |= SYSCTL_RCC2_USERCC2;
  // 1) bypass PLL while initializing
  SYSCTL_RCC2_R |= SYSCTL_RCC2_BYPASS2;
  // 2) select the crystal value and oscillator source
  SYSCTL_RCC_R &= ~SYSCTL_RCC_XTAL_M;   // clear XTAL field
  SYSCTL_RCC_R += SYSCTL_RCC_XTAL_16MHZ;// configure for 16 MHz crystal
  SYSCTL_RCC2_R &= ~SYSCTL_RCC2_OSCSRC2_M;// clear oscillator source field
  SYSCTL_RCC2_R += SYSCTL_RCC2_OSCSRC2_MO;// configure for main oscillator
source
  // 3) activate PLL by clearing PWRDN
  SYSCTL_RCC2_R &= ~SYSCTL_RCC2_PWRDN2;
  // 4) set the desired system divider and the system divider least
significant bit
  SYSCTL_RCC2_R |= SYSCTL_RCC2_DIV400;  // use 400 MHz PLL
  SYSCTL_RCC2_R = (SYSCTL_RCC2_R&~0x1FC00000) // clear system clock divider
field
                  + (SYSDIV2<<22);      // configure for 80 MHz clock
  // 5) wait for the PLL to lock by polling PLLLRIS
  while((SYSCTL_RIS_R&SYSCTL_RIS_PLLLRIS)==0){};
  // 6) enable use of PLL by clearing BYPASS
  SYSCTL_RCC2_R &= ~SYSCTL_RCC2_BYPASS2;
}


/*
SYSDIV2  Divisor  Clock (MHz)
 0        1       reserved
 1        2       reserved
 2        3       reserved
 3        4       reserved
 4        5       80.000
 5        6       66.667
 6        7       reserved
 7        8       50.000
 8        9       44.444
 9        10      40.000
 10       11      36.364
```

11	12	33.333
12	13	30.769
13	14	28.571
14	15	26.667
15	16	25.000
16	17	23.529
17	18	22.222
18	19	21.053
19	20	20.000
20	21	19.048
21	22	18.182
22	23	17.391
23	24	16.667
24	25	16.000
25	26	15.385
26	27	14.815
27	28	14.286
28	29	13.793
29	30	13.333
30	31	12.903
31	32	12.500
32	33	12.121
33	34	11.765
34	35	11.429
35	36	11.111
36	37	10.811
37	38	10.526
38	39	10.256
39	40	10.000
40	41	9.756
41	42	9.524
42	43	9.302
43	44	9.091
44	45	8.889
45	46	8.696
46	47	8.511
47	48	8.333
48	49	8.163
49	50	8.000
50	51	7.843
51	52	7.692
52	53	7.547
53	54	7.407
54	55	7.273
55	56	7.143
56	57	7.018
57	58	6.897
58	59	6.780
59	60	6.667
60	61	6.557
61	62	6.452
62	63	6.349

63	64	6.250
64	65	6.154
65	66	6.061
66	67	5.970
67	68	5.882
68	69	5.797
69	70	5.714
70	71	5.634
71	72	5.556
72	73	5.479
73	74	5.405
74	75	5.333
75	76	5.263
76	77	5.195
77	78	5.128
78	79	5.063
79	80	5.000
80	81	4.938
81	82	4.878
82	83	4.819
83	84	4.762
84	85	4.706
85	86	4.651
86	87	4.598
87	88	4.545
88	89	4.494
89	90	4.444
90	91	4.396
91	92	4.348
92	93	4.301
93	94	4.255
94	95	4.211
95	96	4.167
96	97	4.124
97	98	4.082
98	99	4.040
99	100	4.000
100	101	3.960
101	102	3.922
102	103	3.883
103	104	3.846
104	105	3.810
105	106	3.774
106	107	3.738
107	108	3.704
108	109	3.670
109	110	3.636
110	111	3.604
111	112	3.571
112	113	3.540
113	114	3.509
114	115	3.478

```
 115      116     3.448
 116      117     3.419
 117      118     3.390
 118      119     3.361
 119      120     3.333
 120      121     3.306
 121      122     3.279
 122      123     3.252
 123      124     3.226
 124      125     3.200
 125      126     3.175
 126      127     3.150
 127      128     3.125
*/
```

3. Multithreading

```
// SysTickInts.c
// Runs on LM4F120/TM4C123
// Use the SysTick timer to request interrupts at a particular period.
// Daniel Valvano
// October 11, 2012

/* This example accompanies the book
   "Embedded Systems: Real Time Interfacing to Arm Cortex M
Microcontrollers",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2014

   Program 5.12, section 5.7

 Copyright 2014 by Jonathan W. Valvano, valvano@mail.utexas.edu
    You may use, edit, run or distribute this file
    as long as the above copyright notice remains
 THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS,
IMPLIED
 OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS
SOFTWARE.
 VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL,
INCIDENTAL,
 OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 For more information about my classes, my research, and my books, see
 http://users.ece.utexas.edu/~valvano/
 */

#include <stdint.h>
#include "tm4c123gh6pm.h"
#include "SysTickInts.h"

#define NVIC_ST_CTRL_CLK_SRC    0x00000004  // Clock Source
#define NVIC_ST_CTRL_INTEN      0x00000002  // Interrupt enable
#define NVIC_ST_CTRL_ENABLE     0x00000001  // Counter mode

void DisableInterrupts(void); // Disable interrupts
void EnableInterrupts(void);  // Enable interrupts
```

```
long StartCritical (void);    // previous I bit, disable interrupts
void EndCritical(long sr);    // restore I bit to previous value
void WaitForInterrupt(void);  // low power mode

// **************SysTick_Init*********************
// Initialize SysTick periodic interrupts
// Input: interrupt period
//        Units of period are 12.5ns (assuming 50 MHz clock)
//        Maximum is 2^24-1
//        Minimum is determined by length of ISR
// Output: none
void SysTick_Init(uint32_t period){
  NVIC_ST_CTRL_R = 0;         // disable SysTick during setup
  NVIC_ST_RELOAD_R = period-1;// reload value
  NVIC_ST_CURRENT_R = 0;      // any write to current clears it
  NVIC_SYS_PRI3_R = (NVIC_SYS_PRI3_R&0x00FFFFFF)|0x40000000; // priority 2
                              // enable SysTick with core clock and
interrupts
  NVIC_ST_CTRL_R |= 0x07; //CLK INT EN on
}
```

4. Motor Control

```
#include <stdint.h>
#include <math.h>

#define NOMINAL_POWER (93.21)
#define NOMINAL_VOLTAGE (230.0)
#define NOMINAL_FREQUENCY (60.0)
#define STARTUP_VOLTAGE (19.0)
#define DUTY_CYCLE_MIN (0.1)
//#define DUTY_CYCLE_MAX (0.85)
#define DUTY_CYCLE_MAX (0.80) // Due to inductor saturation (may be fixed?)

// Estimate motor voltage for a desired power output from quadratic
// FIXME for maximum efficiency this profile should be measured
double calculate_motor_voltage(double pv_power) {
        return NOMINAL_VOLTAGE * sqrt(pv_power / NOMINAL_POWER);
}

// V/f control, linear relationship between V and f from
// (0, startup_voltage) to (f_nom / 2, v_nom / 2) and
// (f_nom / 2, v_nom / 2) to (f_nom, v_nom)
double calculate_motor_frequency(double motor_voltage) {
        double slope = 0;
        double frequency = 0;
        if (motor_voltage < NOMINAL_VOLTAGE / 2) {
                slope = (NOMINAL_FREQUENCY / 2) / (NOMINAL_VOLTAGE / 2 -
STARTUP_VOLTAGE);
                frequency = (motor_voltage - STARTUP_VOLTAGE) * slope;
```

```
        } else {
                slope = NOMINAL_FREQUENCY / NOMINAL_VOLTAGE;
                frequency = (motor_voltage - NOMINAL_VOLTAGE) * slope +
NOMINAL_FREQUENCY;
        }

        if (frequency < 0)
                return 0;
        else
                return frequency;
}

// Calculate inverter duty cycle needed to produce a desired output voltage
double calculate_converter_duty(double desired_converter_voltage, double
pv_voltage) {
        // V_out = V_in / (1 - D)
        // D = 1 - V_in / V_out
        double duty_cycle = 1 - pv_voltage / desired_converter_voltage;
        if (duty_cycle < DUTY_CYCLE_MIN)
                return DUTY_CYCLE_MIN;
        else if (duty_cycle > DUTY_CYCLE_MAX)
                return DUTY_CYCLE_MAX;
        else
                return duty_cycle;
}

// Calculate the converter voltage resulting from a given duty cycle
double calculate_converter_voltage(double pv_voltage, double duty_cycle) {
        return pv_voltage / (1 - duty_cycle);
}
```

5. Full System Control

```
#include <stdint.h>
#include <stdio.h>

#include "PLL.h"
#include "PWM.h"
#include "MotorControl.h"
#include "SysTickInts.h"
#include "ST7735.h"
#include "Sensing.h"

#define TIMER_PERIOD_K (100) // Lower is faster, but problems below 100

#define TIMER_PERIOD (TIMER_PERIOD_K * 1000)
#define TIMER_FREQUENCY (80000 / TIMER_PERIOD_K)

double frequency = 0;

void DisableInterrupts(void); // Disable interrupts
```

```
void EnableInterrupts(void);  // Enable interrupts

void Delay10ms(uint32_t count){
        uint32_t volatile time;
  while(count>0){
    time = 72724;  // 0.01sec at 80 MHz
    while(time){
                time--;
    }
    count--;
  }
}

void SysTick_Handler(void){
        // Advance PWM phases
        PWM_tick(TIMER_FREQUENCY, frequency);
}

int main() {
        while(1) {
                Delay10ms(10);

                // Read photoresistor value from PE1
                int a[9];
                ADC_In(a);
                double val = ((double) a[3]) / 4096.0;


                // Read data from sensing
                double pv_voltage = 72; // Voltage available from PV, should
be sensed
                double pv_power = val * 93.21; // First order estimate for
power available

                // Calculate motor voltage from P-V relationship
                double motor_voltage = calculate_motor_voltage(pv_power); //
Estimated curve, not measured
                // Calculate converter voltage necessary to produce desired
motor voltage
                double  desired_converter_voltage  =  4  *  motor_voltage  /
3.141592; // Estimate, should be measured or sensed
                //  Calculate  converter  duty  cycle  necessary  to  produce
desired converter voltage
                double                    converter_duty                    =
calculate_converter_duty(desired_converter_voltage, pv_voltage);
                // Calculate actual converter voltage from given duty cycle
                double              actual_converter_voltage                =
calculate_converter_voltage(pv_voltage, converter_duty);
                // Make  adjustments  if  system  is  unable  to  reach  desired
converter voltage
                double buck_factor = 1;
```

```
                // If converter voltage is too high, buck output at three
phase inverter
                if (actual_converter_voltage > desired_converter_voltage) {
                        buck_factor     =     desired_converter_voltage     /
actual_converter_voltage;
                // If converter voltage is not high enough, adjust motor
voltage
                }       else       if       (actual_converter_voltage       <
desired_converter_voltage) {
                        motor_voltage = 3.141592 * actual_converter_voltage /
4;
                }

                // Calculate motor frequency from V-f relationship
                frequency = calculate_motor_frequency(motor_voltage);

                // Set output parameters
                // Set bucking of three-phase inverter if necessary
                set_range_factor(buck_factor);

                // Set converter duty cycle
                set_converter_duty(1  -  converter_duty);  //  Inverting  gate
driver
        }
}

int SystemInit() {
        DisableInterrupts();
        PLL_Init();
        ADC_Init();
        SysTick_Init(TIMER_PERIOD);

        //ST7735_InitR(INITR_REDTAB); // Problems with LCD printing
        PWM_Init();
        EnableInterrupts();
        return 0;
}
```

6. Voltage and Current sensing

```
//Sensing.c

/*
      Pin assignments-
            PE0: PV-DC voltage pin (AIN3)

            PE1: PV-DC current pin (AIN2)

            PE2: DC-DC voltage pin (AIN1)

            PE3: a-leg AC voltage pin (AIN0)
            PE4: b-leg AC voltage pin (AIN9)
```

```
            PE5: c-leg AC voltage pin (AIN8)

            PD0: a-leg AC current pin (AIN7)
            PD1: b-leg AC current pin (AIN6)
            PD2: c-leg AC current pin (AIN5)
            */

#include <stdint.h>
#include "tm4c123gh6pm.h"
#include "ST7735.h"

void DisableInterrupts(void); // Disable interrupts
void EnableInterrupts(void);  // Enable interrupts

/***********ADC_Init()************
      Init Pins PE0-5, PD 0-2 for ADC functionality
      Use ADC0 module to handle all inputs
      Input: None
      Output: None
      */
void ADC_Init(void){
      volatile int delay =0;
      SYSCTL_RCGCGPIO_R |= 0x10;      // 1) activate clock for Port E
      while((SYSCTL_PRGPIO_R&0x10) == 0){};

      SYSCTL_RCGCGPIO_R |= 0x08;      // 1) activate clock for Port D
      while((SYSCTL_PRGPIO_R&0x08) == 0){};

      SYSCTL_RCGCADC_R |= 0x01;       // 2) activate ADC0
      delay=SYSCTL_RCGCGPIO_R;
      delay=SYSCTL_RCGCGPIO_R;

      GPIO_PORTE_DIR_R &= ~0x3F;       // 3) make PE0 PE1 PE2 PE3 PE4 PE5
input
      GPIO_PORTE_AFSEL_R |= 0x3F;     // 4) enable alternate function on PE0
PE1 PE2 PE3 PE4 PE5
      GPIO_PORTE_DEN_R &= ~0x3F;       // 5) disable digital I/O on PE0 PE1
PE2 PE3 PE4 PE5
                                                      // 5a) configure PE4 as
?? (skip this line because PCTL is for digital only)
      GPIO_PORTE_PCTL_R = GPIO_PORTE_PCTL_R&0xFF000000;
      GPIO_PORTE_AMSEL_R |= 0x3F;      // 6) enable analog functionality on
PE0 PE1 PE2 PE3 PE4 PE5

      GPIO_PORTD_DIR_R &= ~0x07;      // 3) make PD0 PD1 PD2
      GPIO_PORTD_AFSEL_R |= 0x07;     // 4) enable alternate function on PD0
PD1 PD2
      GPIO_PORTD_DEN_R &= ~0x07;       // 5) disable digital I/O on PD0 PD1
PD2
                                                      // 5a) configure PE4 as
?? (skip this line because PCTL is for digital only)
      GPIO_PORTD_PCTL_R = GPIO_PORTD_PCTL_R&0xFFFFF000;
```

```
      GPIO_PORTD_AMSEL_R |= 0x07;      // 6) enable analog functionality on
PD0 PD1 PD2


      ADC0_PC_R &= ~0xF;              // 8) clear max sample rate field
      ADC0_PC_R |= 0x7;               //    configure for 125K samples/sec
      ADC0_SSPRI_R = 0x3210;          // 9) Sequencer 3 is lowest priority

      ADC0_ACTSS_R &= ~0x0001;        // 10) disable sample sequencer 0
      ADC0_ACTSS_R &= ~0x0008;        // 10) disable sample sequencer 3
      ADC0_EMUX_R &= ~0x000F;         // 11) seq0 is software trigger
      ADC0_EMUX_R &= ~0xF000;         // 11) seq3 is software trigger
      ADC0_SSMUX0_R = 0x76532109;     // 12) set channels for SS0
      ADC0_SSMUX3_R = 0x0008;         // 12) set channels for SS3
      ADC0_SSCTL0_R = 0x60000000;     // 13) no TS7 D7, yes IE7 END7
      ADC0_SSCTL3_R = 0x0006;         // 13) no TS0 D0, yes IE0 END0
      ADC0_IM_R &= ~0x0001;           // 14) disable SS0 interrupts
      ADC0_IM_R &= ~0x0008;               // 14) disable SS3 interrupts
      ADC0_ACTSS_R |= 0x0001;         // 15) enable sample sequencer 0
      ADC0_ACTSS_R |= 0x0008;         // 15) enable sample sequencer 3

      ADC0_SAC_R = 0x04;                          //  16)  enable  hardware
oversampling; A N means 2^N (16 here) samples are averaged; 0<=N<=6
}

/***********ADC_In***************
      Busy-wait Analog to digital conversion
      Input: none
      Output: ten 12-bit result of ADC conversion -- value between 0 and
4095,
      data[0] is ADC9 (PE4) 0 to 4095
      data[1] is ADC0 (PE3) 0 to 4095
      data[2] is ADC1 (PE2) 0 to 4095
      data[3] is ADC2 (PE1) 0 to 4095
      data[4] is ADC3 (PE0) 0 to 4095
      data[5] is ADC5 (PD2) 0 to 4095
      data[6] is ADC6 (PD1) 0 to 4095
      data[7] is ADC7 (PD0) 0 to 4095
      data[8] is ADC8 (PE5) 0 to 4095
      */

void ADC_In(int data[9]){
      DisableInterrupts();
      ADC0_PSSI_R = 0x0001; //write to PSSI bit 0
      while((ADC0_RIS_R&0x01)==0){};   //busy-wait
      data[0] = ADC0_SSFIFO0_R&0xFFF;     //read from SSFIFO0
      data[1] = ADC0_SSFIFO0_R&0xFFF;
      data[2] = ADC0_SSFIFO0_R&0xFFF;
      data[3] = ADC0_SSFIFO0_R&0xFFF;
      data[4] = ADC0_SSFIFO0_R&0xFFF;
      data[5] = ADC0_SSFIFO0_R&0xFFF;
      data[6] = ADC0_SSFIFO0_R&0xFFF;
```

```
	data[7] = ADC0_SSFIFO0_R&0xFFF;
	ADC0_ISC_R = 0x0001;    //clear sample complete flag (write to ISC bit
3)

	ADC0_PSSI_R = 0x0008; //write to PSSI bit 3
	while((ADC0_RIS_R&0x08)==0){};   //busy-wait
	data[8] = ADC0_SSFIFO3_R&0xFFF;     //read from SSFIFO3
	ADC0_ISC_R = 0x0008;    //clear sample complete flag (write to ISC bit
3)
	EnableInterrupts();

}


/***********ADC_Calib*************
	return the analog equivelent values of the sensing data
	Calibration for the ADC ports to convert ADC value (0-4095) to analog
value (i.e 3.3V -> 4095 in ADC -> 220V returned)
	Input: 1) value between 0 and 8 to index which ADC to be converted,
chooses formula to convert with (switch statement)
	Output: double with calculated value
	*/
double ADC_Calib (int choice){
	double retValue = 0;
	volatile int tempData = 0;
	int data[9] = {0};
	ADC_In(data);
	tempData = data[choice];

	switch(choice){
		case 0: //AIN9, b-leg AC voltage on PE4
			ST7735_OutUDec(9);
	//ignore this for now, will need to experimentally calibrate then
format to linear data then formula then implement
			break;
		case 1: //AIN0, a-leg AC voltage on PE3
			ST7735_OutUDec(0);
			break;
		case 2: //AIN1, DC-DC voltage on PE2
			ST7735_OutUDec(1);
			break;
		case 3: //AIN2, PV-DC current on PE1
			ST7735_OutUDec(2);
			break;
		case 4: //AIN3, PV-DC voltage on PE0
			ST7735_OutUDec(3);
			break;
		case 5: //AIN5, c-leg AC current on PD2
			ST7735_OutUDec(5);
			break;
		case 6: //AIN6, b-leg AC current on PD1
			ST7735_OutUDec(6);
```

```
                  break;
            case 7: //AIN7, a-leg AC current on PD0
                  ST7735_OutUDec(7);
                  break;
            case 8: //AIN8, c-leg AC voltage on PE5
                  ST7735_OutUDec(8);
                  break;
      }
      return retValue;
}

/***********ADC_Print*************
      print the raw ADC data 0-4095
      if setup == 1 sets up screen with ADC data pairings if first / new
screen printed to reduce flickering of all screen
      else prints ADC values in the correct location (number starts in 6th
location, in the indexed (ith) row)
      Input: 9 int array with ADC data
      Output: None
      */
void ADC_Print(int setup){
      int data[9] = {0};
      if(setup==1){
            ST7735_FillScreen(0x0000);
            for(int i=0; i<9; i++){
                  ST7735_SetCursor(0,i);
                  ST7735_OutString("ADC"); //19 characters each line max
                  switch(i){
                        case 0:
                              ST7735_OutUDec(9);
                              break;
                        case 1:
                              ST7735_OutUDec(0);
                              break;
                        case 2:
                              ST7735_OutUDec(1);
                              break;
                        case 3:
                              ST7735_OutUDec(2);
                              break;
                        case 4:
                              ST7735_OutUDec(3);
                              break;
                        case 5:
                              ST7735_OutUDec(5);
                              break;
                        case 6:
                              ST7735_OutUDec(6);
                              break;
                        case 7:
                              ST7735_OutUDec(7);
                              break;
```

```
                        case 8:
                              ST7735_OutUDec(8);
                              break;
                  }
                  ST7735_OutString(": ");
            }
      } //end setup == 1
      else{
                  ADC_In(data);
            for(int i=0; i<9; i++){
                        ST7735_SetCursor(6,i);
                        ST7735_OutString("    ");
                        ST7735_SetCursor(6,i);
                        ST7735_OutUDec(data[i]);
            }
      }
}

/***********error*************
      returns 1 if system has error
      Error is defined as: ???? Consult Baldick for advice and definitions
      Input: 9 int array with ADC data (calibrated)
      Output: 0 if no error, 1 if error
      */
int error (){
                                                //needs  defining  and  then
coding
      // PV voltage significantly higher than rated
      // PV current significantly higher than rated
      // Output power significantly less than input power
      // Output power significantly greater than input power
      // Output power significantly greater than 220 V
      // Output power significantly higher than inverter rating
      // Converter voltage significantly less than PV voltage / 0.85
      // Converter voltage significantly greater than PV voltage / 0.1
      // Converter voltage significantly higher than rated
      return 0;
}

/***********getPvPower*************
      returns power from the DC sensing side, DC Voltage * DC Current
      Input: 9 int array with ADC data (calibrated)
      Output: double value, no truncation
      */
double getPvPower(){
      return ADC_Calib(4) * ADC_Calib(3);
}

/***********getPvVoltage*************
      returns voltage from the PV sensing side
      Input: none
      Output: double value, no truncation
```

```
      */
double getPvVoltage(){
      return ADC_Calib(4);
}

/***********getAcPower*************
      returns power from the AC sensing side
      = (AC Voltage1 * AC Current1)*3 = active power assuming all three
phases are working
      Input: 9 int array with ADC data (calibrated)
      Output: double value, no truncation
      */
double getAcPower(){
      return ADC_Calib(1) * ADC_Calib(7);
}

/***********getDcConverterVoltage*************
      returns voltage from the DC converter side
      Input: 9 int array with ADC data (calibrated)
      Output: double value, no truncation
      */

double getDcConverterVoltage(){
      return ADC_Calib(2);
}
```

7. LCD driver

```
/*************************************************
  This is a library for the Adafruit 1.8" SPI display.
  This library works with the Adafruit 1.8" TFT Breakout w/SD card
  ----> http://www.adafruit.com/products/358
  as well as Adafruit raw 1.8" TFT displayun
  ----> http://www.adafruit.com/products/618

  Check out the links above for our tutorials and wiring diagrams
  These displays use SPI to communicate, 4 or 5 pins are required to
  interface (RST is optional)
  Adafruit invests time and resources providing this open source code,
  please support Adafruit and open-source hardware by purchasing
  products from Adafruit!

  Written by Limor Fried/Ladyada for Adafruit Industries.
  MIT license, all text above must be included in any redistribution
 *************************************************/

// ST7735.c
// Runs on LM4F120/TM4C123
// Low level drivers for the ST7735 160x128 LCD based off of
// the file described above.
//    16-bit color, 128 wide by 160 high LCD
// Daniel Valvano
// September 12, 2013
```

```
// Augmented 7/17/2014 to have a simple graphics facility
// Tested with LaunchPadDLL.dll simulator 9/2/2014
// Last Modified: 3/6/2015

/* This example accompanies the book
   "Embedded Systems: Real Time Interfacing to Arm Cortex M
Microcontrollers",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2014

 Copyright 2014 by Jonathan W. Valvano, valvano@mail.utexas.edu
    You may use, edit, run or distribute this file
    as long as the above copyright notice remains
 THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS,
IMPLIED
 OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS
SOFTWARE.
 VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL,
INCIDENTAL,
 OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 For more information about my classes, my research, and my books, see
 http://users.ece.utexas.edu/~valvano/
 */

// Backlight (pin 10) connected to +3.3 V
// MISO (pin 9) unconnected
// SCK (pin 8) connected to PA2 (SSI0Clk)
// MOSI (pin 7) connected to PA5 (SSI0Tx)
// TFT_CS (pin 6) connected to PA3 (SSI0Fss)
// CARD_CS (pin 5) unconnected
// Data/Command (pin 4) connected to PA6 (GPIO), high for data, low for
command
// RESET (pin 3) connected to PA7 (GPIO)
// VCC (pin 2) connected to +3.3 V
// Gnd (pin 1) connected to ground
#include <stdio.h>
#include <stdint.h>
#include "ST7735.h"
#include "tm4c123gh6pm.h"

// 16 rows (0 to 15) and 21 characters (0 to 20)
// Requires (11 + size*size*6*8) bytes of transmission for each character
uint32_t StX=0; // position along the horizonal axis 0 to 20
uint32_t StY=0; // position along the vertical axis 0 to 15
uint16_t StTextColor = ST7735_YELLOW;

#define ST7735_NOP     0x00
#define ST7735_SWRESET 0x01
#define ST7735_RDDID   0x04
#define ST7735_RDDST   0x09

#define ST7735_SLPIN   0x10
#define ST7735_SLPOUT  0x11
#define ST7735_PTLON   0x12
#define ST7735_NORON   0x13

#define ST7735_INVOFF  0x20
#define ST7735_INVON   0x21
#define ST7735_DISPOFF 0x28
#define ST7735_DISPON  0x29
```

```
#define ST7735_CASET   0x2A
#define ST7735_RASET   0x2B
#define ST7735_RAMWR   0x2C
#define ST7735_RAMRD   0x2E

#define ST7735_PTLAR   0x30
#define ST7735_COLMOD  0x3A
#define ST7735_MADCTL  0x36

#define ST7735_FRMCTR1 0xB1
#define ST7735_FRMCTR2 0xB2
#define ST7735_FRMCTR3 0xB3
#define ST7735_INVCTR  0xB4
#define ST7735_DISSET5 0xB6

#define ST7735_PWCTR1  0xC0
#define ST7735_PWCTR2  0xC1
#define ST7735_PWCTR3  0xC2
#define ST7735_PWCTR4  0xC3
#define ST7735_PWCTR5  0xC4
#define ST7735_VMCTR1  0xC5

#define ST7735_RDID1   0xDA
#define ST7735_RDID2   0xDB
#define ST7735_RDID3   0xDC
#define ST7735_RDID4   0xDD

#define ST7735_PWCTR6  0xFC

#define ST7735_GMCTRP1 0xE0
#define ST7735_GMCTRN1 0xE1

#define TFT_CS                  (*((volatile uint32_t *)0x40004020))
#define TFT_CS_LOW              0           // CS normally controlled by
hardware
#define TFT_CS_HIGH             0x08
#define DC                      (*((volatile uint32_t *)0x40004100))
#define DC_COMMAND              0
#define DC_DATA                 0x40
#define RESET                   (*((volatile uint32_t *)0x40004200))
#define RESET_LOW               0
#define RESET_HIGH              0x80

#define SSI_CR0_SCR_M           0x0000FF00  // SSI Serial Clock Rate
#define SSI_CR0_SPH             0x00000080  // SSI Serial Clock Phase
#define SSI_CR0_SPO             0x00000040  // SSI Serial Clock Polarity
#define SSI_CR0_FRF_M           0x00000030  // SSI Frame Format Select
#define SSI_CR0_FRF_MOTO        0x00000000  // Freescale SPI Frame Format
#define SSI_CR0_DSS_M           0x0000000F  // SSI Data Size Select
#define SSI_CR0_DSS_8           0x00000007  // 8-bit data
#define SSI_CR1_MS              0x00000004  // SSI Master/Slave Select
#define SSI_CR1_SSE             0x00000002  // SSI Synchronous Serial Port
                                            // Enable
#define SSI_SR_BSY              0x00000010  // SSI Busy Bit
#define SSI_SR_TNF              0x00000002  // SSI Transmit FIFO Not Full
#define SSI_CPSR_CPSDVSR_M      0x000000FF  // SSI Clock Prescale Divisor
#define SSI_CC_CS_M             0x0000000F  // SSI Baud Clock Source
#define SSI_CC_CS_SYSPLL        0x00000000  // Either the system clock (if
the
```

```
                                        // PLL bypass is in effect) or
the
                                        // PLL output (default)
#define SYSCTL_RCGC1_SSI0       0x00000010  // SSI0 Clock Gating Control
#define SYSCTL_RCGC2_GPIOA      0x00000001  // port A Clock Gating Control
#define ST7735_TFTWIDTH  128
#define ST7735_TFTHEIGHT 160

#define ST7735_NOP     0x00
#define ST7735_SWRESET 0x01
#define ST7735_RDDID   0x04
#define ST7735_RDDST   0x09

#define ST7735_SLPIN   0x10
#define ST7735_SLPOUT  0x11
#define ST7735_PTLON   0x12
#define ST7735_NORON   0x13

#define ST7735_INVOFF  0x20
#define ST7735_INVON   0x21
#define ST7735_DISPOFF 0x28
#define ST7735_DISPON  0x29
#define ST7735_CASET   0x2A
#define ST7735_RASET   0x2B
#define ST7735_RAMWR   0x2C
#define ST7735_RAMRD   0x2E

#define ST7735_PTLAR   0x30
#define ST7735_COLMOD  0x3A
#define ST7735_MADCTL  0x36

#define ST7735_FRMCTR1 0xB1
#define ST7735_FRMCTR2 0xB2
#define ST7735_FRMCTR3 0xB3
#define ST7735_INVCTR  0xB4
#define ST7735_DISSET5 0xB6

#define ST7735_PWCTR1  0xC0
#define ST7735_PWCTR2  0xC1
#define ST7735_PWCTR3  0xC2
#define ST7735_PWCTR4  0xC3
#define ST7735_PWCTR5  0xC4
#define ST7735_VMCTR1  0xC5

#define ST7735_RDID1   0xDA
#define ST7735_RDID2   0xDB
#define ST7735_RDID3   0xDC
#define ST7735_RDID4   0xDD

#define ST7735_PWCTR6  0xFC

#define ST7735_GMCTRP1 0xE0
#define ST7735_GMCTRN1 0xE1

// standard ascii 5x7 font
// originally from glcdfont.c from Adafruit project
static const uint8_t Font[] = {
  0x00, 0x00, 0x00, 0x00, 0x00,
  0x3E, 0x5B, 0x4F, 0x5B, 0x3E,
  0x3E, 0x6B, 0x4F, 0x6B, 0x3E,
```

```
0x1C, 0x3E, 0x7C, 0x3E, 0x1C,
0x18, 0x3C, 0x7E, 0x3C, 0x18,
0x1C, 0x57, 0x7D, 0x57, 0x1C,
0x1C, 0x5E, 0x7F, 0x5E, 0x1C,
0x00, 0x18, 0x3C, 0x18, 0x00,
0xFF, 0xE7, 0xC3, 0xE7, 0xFF,
0x00, 0x18, 0x24, 0x18, 0x00,
0xFF, 0xE7, 0xDB, 0xE7, 0xFF,
0x30, 0x48, 0x3A, 0x06, 0x0E,
0x26, 0x29, 0x79, 0x29, 0x26,
0x40, 0x7F, 0x05, 0x05, 0x07,
0x40, 0x7F, 0x05, 0x25, 0x3F,
0x5A, 0x3C, 0xE7, 0x3C, 0x5A,
0x7F, 0x3E, 0x1C, 0x1C, 0x08,
0x08, 0x1C, 0x1C, 0x3E, 0x7F,
0x14, 0x22, 0x7F, 0x22, 0x14,
0x5F, 0x5F, 0x00, 0x5F, 0x5F,
0x06, 0x09, 0x7F, 0x01, 0x7F,
0x00, 0x66, 0x89, 0x95, 0x6A,
0x60, 0x60, 0x60, 0x60, 0x60,
0x94, 0xA2, 0xFF, 0xA2, 0x94,
0x08, 0x04, 0x7E, 0x04, 0x08,
0x10, 0x20, 0x7E, 0x20, 0x10,
0x08, 0x08, 0x2A, 0x1C, 0x08,
0x08, 0x1C, 0x2A, 0x08, 0x08,
0x1E, 0x10, 0x10, 0x10, 0x10,
0x0C, 0x1E, 0x0C, 0x1E, 0x0C,
0x30, 0x38, 0x3E, 0x38, 0x30,
0x06, 0x0E, 0x3E, 0x0E, 0x06,
0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x5F, 0x00, 0x00,
0x00, 0x07, 0x00, 0x07, 0x00,
0x14, 0x7F, 0x14, 0x7F, 0x14,
0x24, 0x2A, 0x7F, 0x2A, 0x12,
0x23, 0x13, 0x08, 0x64, 0x62,
0x36, 0x49, 0x56, 0x20, 0x50,
0x00, 0x08, 0x07, 0x03, 0x00,
0x00, 0x1C, 0x22, 0x41, 0x00,
0x00, 0x41, 0x22, 0x1C, 0x00,
0x2A, 0x1C, 0x7F, 0x1C, 0x2A,
0x08, 0x08, 0x3E, 0x08, 0x08,
0x00, 0x80, 0x70, 0x30, 0x00,
0x08, 0x08, 0x08, 0x08, 0x08,
0x00, 0x00, 0x60, 0x60, 0x00,
0x20, 0x10, 0x08, 0x04, 0x02,
0x3E, 0x51, 0x49, 0x45, 0x3E, // 0
0x00, 0x42, 0x7F, 0x40, 0x00, // 1
0x72, 0x49, 0x49, 0x49, 0x46, // 2
0x21, 0x41, 0x49, 0x4D, 0x33, // 3
0x18, 0x14, 0x12, 0x7F, 0x10, // 4
0x27, 0x45, 0x45, 0x45, 0x39, // 5
0x3C, 0x4A, 0x49, 0x49, 0x31, // 6
0x41, 0x21, 0x11, 0x09, 0x07, // 7
0x36, 0x49, 0x49, 0x49, 0x36, // 8
0x46, 0x49, 0x49, 0x29, 0x1E, // 9
0x00, 0x00, 0x14, 0x00, 0x00,
0x00, 0x40, 0x34, 0x00, 0x00,
0x00, 0x08, 0x14, 0x22, 0x41,
0x14, 0x14, 0x14, 0x14, 0x14,
0x00, 0x41, 0x22, 0x14, 0x08,
```

```
0x02, 0x01, 0x59, 0x09, 0x06,
0x3E, 0x41, 0x5D, 0x59, 0x4E,
0x7C, 0x12, 0x11, 0x12, 0x7C, // A
0x7F, 0x49, 0x49, 0x49, 0x36, // B
0x3E, 0x41, 0x41, 0x41, 0x22, // C
0x7F, 0x41, 0x41, 0x41, 0x3E, // D
0x7F, 0x49, 0x49, 0x49, 0x41, // E
0x7F, 0x09, 0x09, 0x09, 0x01, // F
0x3E, 0x41, 0x41, 0x51, 0x73, // G
0x7F, 0x08, 0x08, 0x08, 0x7F, // H
0x00, 0x41, 0x7F, 0x41, 0x00, // I
0x20, 0x40, 0x41, 0x3F, 0x01, // J
0x7F, 0x08, 0x14, 0x22, 0x41, // K
0x7F, 0x40, 0x40, 0x40, 0x40, // L
0x7F, 0x02, 0x1C, 0x02, 0x7F, // M
0x7F, 0x04, 0x08, 0x10, 0x7F, // N
0x3E, 0x41, 0x41, 0x41, 0x3E, // O
0x7F, 0x09, 0x09, 0x09, 0x06, // P
0x3E, 0x41, 0x51, 0x21, 0x5E, // Q
0x7F, 0x09, 0x19, 0x29, 0x46, // R
0x26, 0x49, 0x49, 0x49, 0x32, // S
0x03, 0x01, 0x7F, 0x01, 0x03, // T
0x3F, 0x40, 0x40, 0x40, 0x3F, // U
0x1F, 0x20, 0x40, 0x20, 0x1F, // V
0x3F, 0x40, 0x38, 0x40, 0x3F, // W
0x63, 0x14, 0x08, 0x14, 0x63, // X
0x03, 0x04, 0x78, 0x04, 0x03, // Y
0x61, 0x59, 0x49, 0x4D, 0x43, // Z
0x00, 0x7F, 0x41, 0x41, 0x41,
0x02, 0x04, 0x08, 0x10, 0x20,
0x00, 0x41, 0x41, 0x41, 0x7F,
0x04, 0x02, 0x01, 0x02, 0x04,
0x40, 0x40, 0x40, 0x40, 0x40,
0x00, 0x03, 0x07, 0x08, 0x00,
0x20, 0x54, 0x54, 0x78, 0x40, // a
0x7F, 0x28, 0x44, 0x44, 0x38, // b
0x38, 0x44, 0x44, 0x44, 0x28, // c
0x38, 0x44, 0x44, 0x28, 0x7F, // d
0x38, 0x54, 0x54, 0x54, 0x18, // e
0x00, 0x08, 0x7E, 0x09, 0x02, // f
0x18, 0xA4, 0xA4, 0x9C, 0x78, // g
0x7F, 0x08, 0x04, 0x04, 0x78, // h
0x00, 0x44, 0x7D, 0x40, 0x00, // i
0x20, 0x40, 0x40, 0x3D, 0x00, // j
0x7F, 0x10, 0x28, 0x44, 0x00, // k
0x00, 0x41, 0x7F, 0x40, 0x00, // l
0x7C, 0x04, 0x78, 0x04, 0x78, // m
0x7C, 0x08, 0x04, 0x04, 0x78, // n
0x38, 0x44, 0x44, 0x44, 0x38, // o
0xFC, 0x18, 0x24, 0x24, 0x18, // p
0x18, 0x24, 0x24, 0x18, 0xFC, // q
0x7C, 0x08, 0x04, 0x04, 0x08, // r
0x48, 0x54, 0x54, 0x54, 0x24, // s
0x04, 0x04, 0x3F, 0x44, 0x24, // t
0x3C, 0x40, 0x40, 0x20, 0x7C, // u
0x1C, 0x20, 0x40, 0x20, 0x1C, // v
0x3C, 0x40, 0x30, 0x40, 0x3C, // w
0x44, 0x28, 0x10, 0x28, 0x44, // x
0x4C, 0x90, 0x90, 0x90, 0x7C, // y
0x44, 0x64, 0x54, 0x4C, 0x44, // z
```

```
0x00, 0x08, 0x36, 0x41, 0x00,
0x00, 0x00, 0x77, 0x00, 0x00,
0x00, 0x41, 0x36, 0x08, 0x00,
0x02, 0x01, 0x02, 0x04, 0x02,
0x3C, 0x26, 0x23, 0x26, 0x3C,
0x1E, 0xA1, 0xA1, 0x61, 0x12,
0x3A, 0x40, 0x40, 0x20, 0x7A,
0x38, 0x54, 0x54, 0x55, 0x59,
0x21, 0x55, 0x55, 0x79, 0x41,
0x21, 0x54, 0x54, 0x78, 0x41,
0x21, 0x55, 0x54, 0x78, 0x40,
0x20, 0x54, 0x55, 0x79, 0x40,
0x0C, 0x1E, 0x52, 0x72, 0x12,
0x39, 0x55, 0x55, 0x55, 0x59,
0x39, 0x54, 0x54, 0x54, 0x59,
0x39, 0x55, 0x54, 0x54, 0x58,
0x00, 0x00, 0x45, 0x7C, 0x41,
0x00, 0x02, 0x45, 0x7D, 0x42,
0x00, 0x01, 0x45, 0x7C, 0x40,
0xF0, 0x29, 0x24, 0x29, 0xF0,
0xF0, 0x28, 0x25, 0x28, 0xF0,
0x7C, 0x54, 0x55, 0x45, 0x00,
0x20, 0x54, 0x54, 0x7C, 0x54,
0x7C, 0x0A, 0x09, 0x7F, 0x49,
0x32, 0x49, 0x49, 0x49, 0x32,
0x32, 0x48, 0x48, 0x48, 0x32,
0x32, 0x4A, 0x48, 0x48, 0x30,
0x3A, 0x41, 0x41, 0x21, 0x7A,
0x3A, 0x42, 0x40, 0x20, 0x78,
0x00, 0x9D, 0xA0, 0xA0, 0x7D,
0x39, 0x44, 0x44, 0x44, 0x39,
0x3D, 0x40, 0x40, 0x40, 0x3D,
0x3C, 0x24, 0xFF, 0x24, 0x24,
0x48, 0x7E, 0x49, 0x43, 0x66,
0x2B, 0x2F, 0xFC, 0x2F, 0x2B,
0xFF, 0x09, 0x29, 0xF6, 0x20,
0xC0, 0x88, 0x7E, 0x09, 0x03,
0x20, 0x54, 0x54, 0x79, 0x41,
0x00, 0x00, 0x44, 0x7D, 0x41,
0x30, 0x48, 0x48, 0x4A, 0x32,
0x38, 0x40, 0x40, 0x22, 0x7A,
0x00, 0x7A, 0x0A, 0x0A, 0x72,
0x7D, 0x0D, 0x19, 0x31, 0x7D,
0x26, 0x29, 0x29, 0x2F, 0x28,
0x26, 0x29, 0x29, 0x29, 0x26,
0x30, 0x48, 0x4D, 0x40, 0x20,
0x38, 0x08, 0x08, 0x08, 0x08,
0x08, 0x08, 0x08, 0x08, 0x38,
0x2F, 0x10, 0xC8, 0xAC, 0xBA,
0x2F, 0x10, 0x28, 0x34, 0xFA,
0x00, 0x00, 0x7B, 0x00, 0x00,
0x08, 0x14, 0x2A, 0x14, 0x22,
0x22, 0x14, 0x2A, 0x14, 0x08,
0xAA, 0x00, 0x55, 0x00, 0xAA,
0xAA, 0x55, 0xAA, 0x55, 0xAA,
0x00, 0x00, 0x00, 0xFF, 0x00,
0x10, 0x10, 0x10, 0xFF, 0x00,
0x14, 0x14, 0x14, 0xFF, 0x00,
0x10, 0x10, 0xFF, 0x00, 0xFF,
0x10, 0x10, 0xF0, 0x10, 0xF0,
```

```
0x14, 0x14, 0x14, 0xFC, 0x00,
0x14, 0x14, 0xF7, 0x00, 0xFF,
0x00, 0x00, 0xFF, 0x00, 0xFF,
0x14, 0x14, 0xF4, 0x04, 0xFC,
0x14, 0x14, 0x17, 0x10, 0x1F,
0x10, 0x10, 0x1F, 0x10, 0x1F,
0x14, 0x14, 0x14, 0x1F, 0x00,
0x10, 0x10, 0x10, 0xF0, 0x00,
0x00, 0x00, 0x00, 0x1F, 0x10,
0x10, 0x10, 0x10, 0x1F, 0x10,
0x10, 0x10, 0x10, 0xF0, 0x10,
0x00, 0x00, 0x00, 0xFF, 0x10,
0x10, 0x10, 0x10, 0x10, 0x10,
0x10, 0x10, 0x10, 0xFF, 0x10,
0x00, 0x00, 0x00, 0xFF, 0x14,
0x00, 0x00, 0xFF, 0x00, 0xFF,
0x00, 0x00, 0x1F, 0x10, 0x17,
0x00, 0x00, 0xFC, 0x04, 0xF4,
0x14, 0x14, 0x17, 0x10, 0x17,
0x14, 0x14, 0xF4, 0x04, 0xF4,
0x00, 0x00, 0xFF, 0x00, 0xF7,
0x14, 0x14, 0x14, 0x14, 0x14,
0x14, 0x14, 0xF7, 0x00, 0xF7,
0x14, 0x14, 0x14, 0x17, 0x14,
0x10, 0x10, 0x1F, 0x10, 0x1F,
0x14, 0x14, 0x14, 0xF4, 0x14,
0x10, 0x10, 0xF0, 0x10, 0xF0,
0x00, 0x00, 0x1F, 0x10, 0x1F,
0x00, 0x00, 0x00, 0x1F, 0x14,
0x00, 0x00, 0x00, 0xFC, 0x14,
0x00, 0x00, 0xF0, 0x10, 0xF0,
0x10, 0x10, 0xFF, 0x10, 0xFF,
0x14, 0x14, 0x14, 0xFF, 0x14,
0x10, 0x10, 0x10, 0x1F, 0x00,
0x00, 0x00, 0x00, 0xF0, 0x10,
0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xF0, 0xF0, 0xF0, 0xF0, 0xF0,
0xFF, 0xFF, 0xFF, 0x00, 0x00,
0x00, 0x00, 0x00, 0xFF, 0xFF,
0x0F, 0x0F, 0x0F, 0x0F, 0x0F,
0x38, 0x44, 0x44, 0x38, 0x44,
0x7C, 0x2A, 0x2A, 0x3E, 0x14,
0x7E, 0x02, 0x02, 0x06, 0x06,
0x02, 0x7E, 0x02, 0x7E, 0x02,
0x63, 0x55, 0x49, 0x41, 0x63,
0x38, 0x44, 0x44, 0x3C, 0x04,
0x40, 0x7E, 0x20, 0x1E, 0x20,
0x06, 0x02, 0x7E, 0x02, 0x02,
0x99, 0xA5, 0xE7, 0xA5, 0x99,
0x1C, 0x2A, 0x49, 0x2A, 0x1C,
0x4C, 0x72, 0x01, 0x72, 0x4C,
0x30, 0x4A, 0x4D, 0x4D, 0x30,
0x30, 0x48, 0x78, 0x48, 0x30,
0xBC, 0x62, 0x5A, 0x46, 0x3D,
0x3E, 0x49, 0x49, 0x49, 0x00,
0x7E, 0x01, 0x01, 0x01, 0x7E,
0x2A, 0x2A, 0x2A, 0x2A, 0x2A,
0x44, 0x44, 0x5F, 0x44, 0x44,
0x40, 0x51, 0x4A, 0x44, 0x40,
0x40, 0x44, 0x4A, 0x51, 0x40,
```

```
  0x00, 0x00, 0xFF, 0x01, 0x03,
  0xE0, 0x80, 0xFF, 0x00, 0x00,
  0x08, 0x08, 0x6B, 0x6B, 0x08,
  0x36, 0x12, 0x36, 0x24, 0x36,
  0x06, 0x0F, 0x09, 0x0F, 0x06,
  0x00, 0x00, 0x18, 0x18, 0x00,
  0x00, 0x00, 0x10, 0x10, 0x00,
  0x30, 0x40, 0xFF, 0x01, 0x01,
  0x00, 0x1F, 0x01, 0x01, 0x1E,
  0x00, 0x19, 0x1D, 0x17, 0x12,
  0x00, 0x3C, 0x3C, 0x3C, 0x3C,
  0x00, 0x00, 0x00, 0x00, 0x00,
};


static uint8_t ColStart, RowStart; // some displays need this changed
static uint8_t Rotation;           // 0 to 3
static enum initRFlags TabColor;
static int16_t _width = ST7735_TFTWIDTH;   // this could probably be a
constant, except it is used in Adafruit_GFX and depends on image rotation
static int16_t _height = ST7735_TFTHEIGHT;


// The Data/Command pin must be valid when the eighth bit is
// sent.  The SSI module has hardware input and output FIFOs
// that are 8 locations deep.  Based on the observation that
// the LCD interface tends to send a few commands and then a
// lot of data, the FIFOs are not used when writing
// commands, and they are used when writing data.  This
// ensures that the Data/Command pin status matches the byte
// that is actually being transmitted.
// The write command operation waits until all data has been
// sent, configures the Data/Command pin for commands, sends
// the command, and then waits for the transmission to
// finish.
// The write data operation waits until there is room in the
// transmit FIFO, configures the Data/Command pin for data,
// and then adds the data to the transmit FIFO.
// NOTE: These functions will crash or stall indefinitely if
// the SSI0 module is not initialized and enabled.
void writecommand(uint8_t c);


void writedata(uint8_t c);

// Subroutine to wait 1 msec
// Inputs: None
// Outputs: None
// Notes: ...
void Delay1ms(uint32_t n){uint32_t volatile time;
  while(n){
    time = 72724*2/91;  // 1msec, tuned at 80 MHz
    while(time){
                time--;
    }
    n--;
  }
}

// Rather than a bazillion writecommand() and writedata() calls, screen
```

```
// initialization commands and arguments are organized in these tables
// stored in ROM.  The table may look bulky, but that's mostly the
// formatting -- storage-wise this is hundreds of bytes more compact
// than the equivalent code.  Companion function follows.
#define DELAY 0x80
static const uint8_t
  Bcmd[] = {                  // Initialization commands for 7735B screens
    18,                       // 18 commands in list:
    ST7735_SWRESET,   DELAY,  //  1: Software reset, no args, w/delay
      50,                     //     50 ms delay
    ST7735_SLPOUT ,   DELAY,  //  2: Out of sleep mode, no args, w/delay
      255,                    //     255 = 500 ms delay
    ST7735_COLMOD , 1+DELAY,  //  3: Set color mode, 1 arg + delay:
      0x05,                   //     16-bit color
      10,                     //     10 ms delay
    ST7735_FRMCTR1, 3+DELAY,  //  4: Frame rate control, 3 args + delay:
      0x00,                   //     fastest refresh
      0x06,                   //     6 lines front porch
      0x03,                   //     3 lines back porch
      10,                     //     10 ms delay
    ST7735_MADCTL , 1      ,  //  5: Memory access ctrl (directions), 1
arg:
      0x08,                   //     Row addr/col addr, bottom to top
refresh
    ST7735_DISSET5, 2      ,  //  6: Display settings #5, 2 args, no delay:
      0x15,                   //     1 clk cycle nonoverlap, 2 cycle gate
                              //     rise, 3 cycle osc equalize
      0x02,                   //     Fix on VTL
    ST7735_INVCTR , 1      ,  //  7: Display inversion control, 1 arg:
      0x0,                    //     Line inversion
    ST7735_PWCTR1 , 2+DELAY,  //  8: Power control, 2 args + delay:
      0x02,                   //     GVDD = 4.7V
      0x70,                   //     1.0uA
      10,                     //     10 ms delay
    ST7735_PWCTR2 , 1      ,  //  9: Power control, 1 arg, no delay:
      0x05,                   //     VGH = 14.7V, VGL = -7.35V
    ST7735_PWCTR3 , 2      ,  // 10: Power control, 2 args, no delay:
      0x01,                   //     Opamp current small
      0x02,                   //     Boost frequency
    ST7735_VMCTR1 , 2+DELAY,  // 11: Power control, 2 args + delay:
      0x3C,                   //     VCOMH = 4V
      0x38,                   //     VCOML = -1.1V
      10,                     //     10 ms delay
    ST7735_PWCTR6 , 2      ,  // 12: Power control, 2 args, no delay:
      0x11, 0x15,
    ST7735_GMCTRP1,16      ,  // 13: Magical unicorn dust, 16 args, no
delay:
      0x09, 0x16, 0x09, 0x20, //     (seriously though, not sure what
      0x21, 0x1B, 0x13, 0x19, //      these config values represent)
      0x17, 0x15, 0x1E, 0x2B,
      0x04, 0x05, 0x02, 0x0E,
    ST7735_GMCTRN1,16+DELAY,  // 14: Sparkles and rainbows, 16 args +
delay:
      0x0B, 0x14, 0x08, 0x1E, //     (ditto)
      0x22, 0x1D, 0x18, 0x1E,
      0x1B, 0x1A, 0x24, 0x2B,
      0x06, 0x06, 0x02, 0x0F,
      10,                     //     10 ms delay
    ST7735_CASET  , 4      ,  // 15: Column addr set, 4 args, no delay:
      0x00, 0x02,             //     XSTART = 2
```

```
      0x00, 0x81,             //     XEND = 129
    ST7735_RASET  , 4      ,  // 16: Row addr set, 4 args, no delay:
      0x00, 0x02,             //     XSTART = 1
      0x00, 0x81,             //     XEND = 160
    ST7735_NORON  ,   DELAY,  // 17: Normal display on, no args, w/delay
      10,                     //     10 ms delay
    ST7735_DISPON ,   DELAY,  // 18: Main screen turn on, no args, w/delay
      255 };                  //     255 = 500 ms delay
static const uint8_t
  Rcmd1[] = {                 // Init for 7735R, part 1 (red or green tab)
    15,                       // 15 commands in list:
    ST7735_SWRESET,   DELAY,  //  1: Software reset, 0 args, w/delay
      150,                    //     150 ms delay
    ST7735_SLPOUT ,   DELAY,  //  2: Out of sleep mode, 0 args, w/delay
      255,                    //     500 ms delay
    ST7735_FRMCTR1, 3      ,  //  3: Frame rate ctrl - normal mode, 3 args:
      0x01, 0x2C, 0x2D,       //     Rate = fosc/(1x2+40) * (LINE+2C+2D)
    ST7735_FRMCTR2, 3      ,  //  4: Frame rate control - idle mode, 3
args:
      0x01, 0x2C, 0x2D,       //     Rate = fosc/(1x2+40) * (LINE+2C+2D)
    ST7735_FRMCTR3, 6      ,  //  5: Frame rate ctrl - partial mode, 6
args:
      0x01, 0x2C, 0x2D,       //     Dot inversion mode
      0x01, 0x2C, 0x2D,       //     Line inversion mode
    ST7735_INVCTR , 1      ,  //  6: Display inversion ctrl, 1 arg, no
delay:
      0x07,                   //     No inversion
    ST7735_PWCTR1 , 3      ,  //  7: Power control, 3 args, no delay:
      0xA2,
      0x02,                   //     -4.6V
      0x84,                   //     AUTO mode
    ST7735_PWCTR2 , 1      ,  //  8: Power control, 1 arg, no delay:
      0xC5,                   //     VGH25 = 2.4C VGSEL = -10 VGH = 3 *
AVDD
    ST7735_PWCTR3 , 2      ,  //  9: Power control, 2 args, no delay:
      0x0A,                   //     Opamp current small
      0x00,                   //     Boost frequency
    ST7735_PWCTR4 , 2      ,  // 10: Power control, 2 args, no delay:
      0x8A,                   //     BCLK/2, Opamp current small & Medium
low
      0x2A,
    ST7735_PWCTR5 , 2      ,  // 11: Power control, 2 args, no delay:
      0x8A, 0xEE,
    ST7735_VMCTR1 , 1      ,  // 12: Power control, 1 arg, no delay:
      0x0E,
    ST7735_INVOFF , 0      ,  // 13: Don't invert display, no args, no
delay
    ST7735_MADCTL , 1      ,  // 14: Memory access control (directions), 1
arg:
      0xC8,                   //     row addr/col addr, bottom to top
refresh
    ST7735_COLMOD , 1      ,  // 15: set color mode, 1 arg, no delay:
      0x05 };                 //     16-bit color
static const uint8_t
  Rcmd2green[] = {            // Init for 7735R, part 2 (green tab only)
    2,                        //  2 commands in list:
    ST7735_CASET  , 4      ,  //  1: Column addr set, 4 args, no delay:
      0x00, 0x02,             //     XSTART = 0
      0x00, 0x7F+0x02,        //     XEND = 127
    ST7735_RASET  , 4      ,  //  2: Row addr set, 4 args, no delay:
```

```
      0x00, 0x01,             //     XSTART = 0
      0x00, 0x9F+0x01 };      //     XEND = 159
static const uint8_t
  Rcmd2red[] = {              // Init for 7735R, part 2 (red tab only)
    2,                        //  2 commands in list:
    ST7735_CASET  , 4      ,  //  1: Column addr set, 4 args, no delay:
      0x00, 0x00,             //     XSTART = 0
      0x00, 0x7F,             //     XEND = 127
    ST7735_RASET  , 4      ,  //  2: Row addr set, 4 args, no delay:
      0x00, 0x00,             //     XSTART = 0
      0x00, 0x9F };           //     XEND = 159
static const uint8_t
  Rcmd3[] = {                 // Init for 7735R, part 3 (red or green tab)
    4,                        //  4 commands in list:
    ST7735_GMCTRP1, 16      , //  1: Magical unicorn dust, 16 args, no
delay:
      0x02, 0x1c, 0x07, 0x12,
      0x37, 0x32, 0x29, 0x2d,
      0x29, 0x25, 0x2B, 0x39,
      0x00, 0x01, 0x03, 0x10,
    ST7735_GMCTRN1, 16      , //  2: Sparkles and rainbows, 16 args, no
delay:
      0x03, 0x1d, 0x07, 0x06,
      0x2E, 0x2C, 0x29, 0x2D,
      0x2E, 0x2E, 0x37, 0x3F,
      0x00, 0x00, 0x02, 0x10,
    ST7735_NORON  ,    DELAY, //  3: Normal display on, no args, w/delay
      10,                     //     10 ms delay
    ST7735_DISPON ,    DELAY, //  4: Main screen turn on, no args w/delay
      100 };                  //     100 ms delay


// Companion code to the above tables.  Reads and issues
// a series of LCD commands stored in ROM byte array.
void static commandList(const uint8_t *addr) {

  uint8_t numCommands, numArgs;
  uint16_t ms;

  numCommands = *(addr++);               // Number of commands to follow
  while(numCommands--) {                 // For each command...
    writecommand(*(addr++));             //   Read, issue command
    numArgs  = *(addr++);                //   Number of args to follow
    ms       = numArgs & DELAY;          //   If hibit set, delay follows
args
    numArgs &= ~DELAY;                   //   Mask out delay bit
    while(numArgs--) {                   //   For each argument...
      writedata(*(addr++));              //     Read, issue argument
    }

    if(ms) {
      ms = *(addr++);             // Read post-command delay time (ms)
      if(ms == 255) ms = 500;     // If 255, delay for 500 ms
      Delay1ms(ms);
    }
  }
}


// Initialization code common to both 'B' and 'R' type displays
```

```
void static commonInit(const uint8_t *cmdList) {
  volatile uint32_t delay;
  ColStart  = RowStart = 0; // May be overridden in init func

  SYSCTL_RCGCSSI_R |= 0x01;  // activate SSI0
  SYSCTL_RCGCGPIO_R |= 0x01; // activate port A
  while((SYSCTL_PRGPIO_R&0x01)==0){}; // allow time for clock to start

  // toggle RST low to reset; CS low so it'll listen to us
  // SSI0Fss is temporarily used as GPIO
  GPIO_PORTA_DIR_R |= 0xC8;             // make PA3,6,7 out
  GPIO_PORTA_AFSEL_R &= ~0xC8;          // disable alt funct on PA3,6,7
  GPIO_PORTA_DEN_R |= 0xC8;             // enable digital I/O on PA3,6,7
                                        // configure PA3,6,7 as GPIO
  GPIO_PORTA_PCTL_R = (GPIO_PORTA_PCTL_R&0x00FF0FFF)+0x00000000;
  GPIO_PORTA_AMSEL_R &= ~0xC8;          // disable analog functionality on
PA3,6,7
  TFT_CS = TFT_CS_LOW;
  RESET = RESET_HIGH;
  Delay1ms(500);
  RESET = RESET_LOW;
  Delay1ms(500);
  RESET = RESET_HIGH;
  Delay1ms(500);

  // initialize SSI0
  GPIO_PORTA_AFSEL_R |= 0x2C;           // enable alt funct on PA2,3,5
  GPIO_PORTA_DEN_R |= 0x2C;             // enable digital I/O on PA2,3,5
                                        // configure PA2,3,5 as SSI
  GPIO_PORTA_PCTL_R = (GPIO_PORTA_PCTL_R&0xFF0F00FF)+0x00202200;
  GPIO_PORTA_AMSEL_R &= ~0x2C;          // disable analog functionality on
PA2,3,5
  SSI0_CR1_R &= ~SSI_CR1_SSE;           // disable SSI
  SSI0_CR1_R &= ~SSI_CR1_MS;            // master mode
                                        // configure for system clock/PLL
baud clock source
  SSI0_CC_R = (SSI0_CC_R&~SSI_CC_CS_M)+SSI_CC_CS_SYSPLL;
//                                        // clock divider for 3.125 MHz
SSIClk (50 MHz PIOSC/16)
//  SSI0_CPSR_R = (SSI0_CPSR_R&~SSI_CPSR_CPSDVSR_M)+16;
                                        // clock divider for 8 MHz SSIClk
(80 MHz PLL/24)
                                        // SysClk/(CPSDVSR*(1+SCR))
                                        // 80/(10*(1+0)) = 8 MHz (slower
than 4 MHz)
  SSI0_CPSR_R = (SSI0_CPSR_R&~SSI_CPSR_CPSDVSR_M)+10; // must be even
number
  SSI0_CR0_R &= ~(SSI_CR0_SCR_M |       // SCR = 0 (8 Mbps data rate)
                  SSI_CR0_SPH |         // SPH = 0
                  SSI_CR0_SPO);         // SPO = 0
                                        // FRF = Freescale format
  SSI0_CR0_R = (SSI0_CR0_R&~SSI_CR0_FRF_M)+SSI_CR0_FRF_MOTO;
                                        // DSS = 8-bit data
  SSI0_CR0_R = (SSI0_CR0_R&~SSI_CR0_DSS_M)+SSI_CR0_DSS_8;
  SSI0_CR1_R |= SSI_CR1_SSE;            // enable SSI

  if(cmdList) commandList(cmdList);
}
```

```
//------------ST7735_InitB------------
// Initialization for ST7735B screens.
// Input: none
// Output: none
void ST7735_InitB(void) {
  commonInit(Bcmd);
  ST7735_SetCursor(0,0);
  StTextColor = ST7735_YELLOW;
  ST7735_FillScreen(0);                 // set screen to black
}


//------------ST7735_InitR------------
// Initialization for ST7735R screens (green or red tabs).
// Input: option one of the enumerated options depending on tabs
// Output: none
void ST7735_InitR(enum initRFlags option) {
  commonInit(Rcmd1);
  if(option == INITR_GREENTAB) {
    commandList(Rcmd2green);
    ColStart = 2;
    RowStart = 1;
  } else {
    // colstart, rowstart left at default '0' values
    commandList(Rcmd2red);
  }
  commandList(Rcmd3);

  // if black, change MADCTL color filter
  if (option == INITR_BLACKTAB) {
    writecommand(ST7735_MADCTL);
    writedata(0xC0);
  }
  TabColor = option;
  ST7735_SetCursor(0,0);
  StTextColor = ST7735_YELLOW;
  ST7735_FillScreen(0);                 // set screen to black
}


// Set the region of the screen RAM to be modified
// Pixel colors are sent left to right, top to bottom
// (same as Font table is encoded; different from regular bitmap)
// Requires 11 bytes of transmission
void static setAddrWindow(uint8_t x0, uint8_t y0, uint8_t x1, uint8_t y1) {

  writecommand(ST7735_CASET); // Column addr set
  writedata(0x00);
  writedata(x0+ColStart);     // XSTART
  writedata(0x00);
  writedata(x1+ColStart);     // XEND

  writecommand(ST7735_RASET); // Row addr set
  writedata(0x00);
  writedata(y0+RowStart);     // YSTART
  writedata(0x00);
  writedata(y1+RowStart);     // YEND

  writecommand(ST7735_RAMWR); // write to RAM
}
```

```
// Send two bytes of data, most significant byte first
// Requires 2 bytes of transmission
void static pushColor(uint16_t color) {
  writedata((uint8_t)(color >> 8));
  writedata((uint8_t)color);
}


//------------ST7735_DrawPixel------------
// Color the pixel at the given coordinates with the given color.
// Requires 13 bytes of transmission
// Input: x     horizontal position of the pixel, columns from the left
edge
//               must be less than 128
//               0 is on the left, 126 is near the right
//        y     vertical position of the pixel, rows from the top edge
//               must be less than 160
//               159 is near the wires, 0 is the side opposite the wires
//        color 16-bit color, which can be produced by ST7735_Color565()
// Output: none
void ST7735_DrawPixel(int16_t x, int16_t y, uint16_t color) {

  if((x < 0) || (x >= _width) || (y < 0) || (y >= _height)) return;

//  setAddrWindow(x,y,x+1,y+1); // original code, bug???
  setAddrWindow(x,y,x,y);

  pushColor(color);
}


//------------ST7735_DrawFastVLine------------
// Draw a vertical line at the given coordinates with the given height and
color.
// A vertical line is parallel to the longer side of the rectangular
display
// Requires (11 + 2*h) bytes of transmission (assuming image fully on
screen)
// Input: x     horizontal position of the start of the line, columns from
the left edge
//        y     vertical position of the start of the line, rows from the
top edge
//        h     vertical height of the line
//        color 16-bit color, which can be produced by ST7735_Color565()
// Output: none
void ST7735_DrawFastVLine(int16_t x, int16_t y, int16_t h, uint16_t color)
{
  uint8_t hi = color >> 8, lo = color;

  // Rudimentary clipping
  if((x >= _width) || (y >= _height)) return;
  if((y+h-1) >= _height) h = _height-y;
  setAddrWindow(x, y, x, y+h-1);

  while (h--) {
    writedata(hi);
    writedata(lo);
  }
```

```
}


//------------ST7735_DrawFastHLine------------
// Draw a horizontal line at the given coordinates with the given width and
color.
// A horizontal line is parallel to the shorter side of the rectangular
display
// Requires (11 + 2*w) bytes of transmission (assuming image fully on
screen)
// Input: x     horizontal position of the start of the line, columns from
the left edge
//        y     vertical position of the start of the line, rows from the
top edge
//        w     horizontal width of the line
//        color 16-bit color, which can be produced by ST7735_Color565()
// Output: none
void ST7735_DrawFastHLine(int16_t x, int16_t y, int16_t w, uint16_t color)
{
  uint8_t hi = color >> 8, lo = color;

  // Rudimentary clipping
  if((x >= _width) || (y >= _height)) return;
  if((x+w-1) >= _width)  w = _width-x;
  setAddrWindow(x, y, x+w-1, y);

  while (w--) {
    writedata(hi);
    writedata(lo);
  }
}


//------------ST7735_FillScreen------------
// Fill the screen with the given color.
// Requires 40,971 bytes of transmission
// Input: color 16-bit color, which can be produced by ST7735_Color565()
// Output: none
void ST7735_FillScreen(uint16_t color) {
  ST7735_FillRect(0, 0, _width, _height, color);  // original
//  screen is actually 129 by 161 pixels, x 0 to 128, y goes from 0 to 160
}


//------------ST7735_FillRect------------
// Draw a filled rectangle at the given coordinates with the given width,
height, and color.
// Requires (11 + 2*w*h) bytes of transmission (assuming image fully on
screen)
// Input: x     horizontal position of the top left corner of the
rectangle, columns from the left edge
//        y     vertical position of the top left corner of the rectangle,
rows from the top edge
//        w     horizontal width of the rectangle
//        h     vertical height of the rectangle
//        color 16-bit color, which can be produced by ST7735_Color565()
// Output: none
void ST7735_FillRect(int16_t x, int16_t y, int16_t w, int16_t h, uint16_t
color) {
  uint8_t hi = color >> 8, lo = color;
```

```
  // rudimentary clipping (drawChar w/big text requires this)
  if((x >= _width) || (y >= _height)) return;
  if((x + w - 1) >= _width)  w = _width  - x;
  if((y + h - 1) >= _height) h = _height - y;

  setAddrWindow(x, y, x+w-1, y+h-1);

  for(y=h; y>0; y--) {
    for(x=w; x>0; x--) {
      writedata(hi);
      writedata(lo);
    }
  }
}


//------------ST7735_Color565------------
// Pass 8-bit (each) R,G,B and get back 16-bit packed color.
// Input: r red value
//        g green value
//        b blue value
// Output: 16-bit color
uint16_t ST7735_Color565(uint8_t r, uint8_t g, uint8_t b) {
  return ((b & 0xF8) << 8) | ((g & 0xFC) << 3) | (r >> 3);
}


//------------ST7735_SwapColor------------
// Swaps the red and blue values of the given 16-bit packed color;
// green is unchanged.
// Input: x 16-bit color in format B, G, R
// Output: 16-bit color in format R, G, B
uint16_t ST7735_SwapColor(uint16_t x) {
  return (x << 11) | (x & 0x07E0) | (x >> 11);
}


//------------ST7735_DrawBitmap------------
// Displays a 16-bit color BMP image.  A bitmap file that is created
// by a PC image processing program has a header and may be padded
// with dummy columns so the data have four byte alignment.  This
// function assumes that all of that has been stripped out, and the
// array image[] has one 16-bit halfword for each pixel to be
// displayed on the screen (encoded in reverse order, which is
// standard for bitmap files).  An array can be created in this
// format from a 24-bit-per-pixel .bmp file using the associated
// converter program.
// (x,y) is the screen location of the lower left corner of BMP image
// Requires (11 + 2*w*h) bytes of transmission (assuming image fully on
screen)
// Input: x     horizontal position of the bottom left corner of the image,
columns from the left edge
//        y     vertical position of the bottom left corner of the image,
rows from the top edge
//        image pointer to a 16-bit color BMP image
//        w     number of pixels wide
//        h     number of pixels tall
// Output: none
// Must be less than or equal to 128 pixels wide by 160 pixels high
```

```
void ST7735_DrawBitmap(int16_t x, int16_t y, const uint16_t *image, int16_t
w, int16_t h){
  int16_t skipC = 0;                      // non-zero if columns need to be
skipped due to clipping
  int16_t originalWidth = w;              // save this value; even if not
all columns fit on the screen, the image is still this width in ROM
  int i = w*(h - 1);

  if((x >= _width) || ((y - h + 1) >= _height) || ((x + w) <= 0) || (y <
0)){
    return;                               // image is totally off the screen,
do nothing
  }
  if((w > _width) || (h > _height)){      // image is too wide for the
screen, do nothing
    //***This isn't necessarily a fatal error, but it makes the
    //following logic much more complicated, since you can have
    //an image that exceeds multiple boundaries and needs to be
    //clipped on more than one side.
    return;
  }
  if((x + w - 1) >= _width){              // image exceeds right of screen
    skipC = (x + w) - _width;             // skip cut off columns
    w = _width - x;
  }
  if((y - h + 1) < 0){                    // image exceeds top of screen
    i = i - (h - y - 1)*originalWidth;    // skip the last cut off rows
    h = y + 1;
  }
  if(x < 0){                              // image exceeds left of screen
    w = w + x;
    skipC = -1*x;                         // skip cut off columns
    i = i - x;                            // skip the first cut off columns
    x = 0;
  }
  if(y >= _height){                       // image exceeds bottom of screen
    h = h - (y - _height + 1);
    y = _height - 1;
  }

  setAddrWindow(x, y-h+1, x+w-1, y);

  for(y=0; y<h; y=y+1){
    for(x=0; x<w; x=x+1){
                                          // send the top 8 bits
      writedata((uint8_t)(image[i] >> 8));
                                          // send the bottom 8 bits
      writedata((uint8_t)image[i]);
      i = i + 1;                          // go to the next pixel
    }
    i = i + skipC;
    i = i - 2*originalWidth;
  }
}


//------------ST7735_DrawCharS------------
// Simple character draw function.  This is the same function from
// Adafruit_GFX.c but adapted for this processor.  However, each call
// to ST7735_DrawPixel() calls setAddrWindow(), which needs to send
```

```
// many extra data and commands.  If the background color is the same
// as the text color, no background will be printed, and text can be
// drawn right over existing images without covering them with a box.
// Requires (11 + 2*size*size)*6*8 (image fully on screen; textcolor !=
bgColor)
// Input: x         horizontal position of the top left corner of the
character, columns from the left edge
//        y         vertical position of the top left corner of the
character, rows from the top edge
//        c         character to be printed
//        textColor 16-bit color of the character
//        bgColor   16-bit color of the background
//        size      number of pixels per character pixel (e.g. size==2
prints each pixel of font as 2x2 square)
// Output: none
void ST7735_DrawCharS(int16_t x, int16_t y, char c, int16_t textColor,
int16_t bgColor, uint8_t size){
  uint8_t line; // vertical column of pixels of character in font
  int32_t i, j;
  if((x >= _width)            || // Clip right
     (y >= _height)           || // Clip bottom
     ((x + 5 * size - 1) < 0) || // Clip left
     ((y + 8 * size - 1) < 0))   // Clip top
    return;

  for (i=0; i<6; i++ ) {
    if (i == 5)
      line = 0x0;
    else
      line = Font[(c*5)+i];
    for (j = 0; j<8; j++) {
      if (line & 0x1) {
        if (size == 1) // default size
          ST7735_DrawPixel(x+i, y+j, textColor);
        else {  // big size
          ST7735_FillRect(x+(i*size), y+(j*size), size, size, textColor);
        }
      } else if (bgColor != textColor) {
        if (size == 1) // default size
          ST7735_DrawPixel(x+i, y+j, bgColor);
        else {  // big size
          ST7735_FillRect(x+i*size, y+j*size, size, size, bgColor);
        }
      }
      line >>= 1;
    }
  }
}


//------------ST7735_DrawChar------------
// Advanced character draw function.  This is similar to the function
// from Adafruit_GFX.c but adapted for this processor.  However, this
// function only uses one call to setAddrWindow(), which allows it to
// run at least twice as fast.
// Requires (11 + size*size*6*8) bytes of transmission (assuming image
fully on screen)
// Input: x         horizontal position of the top left corner of the
character, columns from the left edge
```

```
//        y         vertical position of the top left corner of the
character, rows from the top edge
//        c         character to be printed
//        textColor 16-bit color of the character
//        bgColor   16-bit color of the background
//        size      number of pixels per character pixel (e.g. size==2
prints each pixel of font as 2x2 square)
// Output: none
void ST7735_DrawChar(int16_t x, int16_t y, char c, int16_t textColor,
int16_t bgColor, uint8_t size){
  uint8_t line; // horizontal row of pixels of character
  int32_t col, row, i, j;// loop indices
  if(((x + 5*size - 1) >= _width)  || // Clip right
     ((y + 8*size - 1) >= _height) || // Clip bottom
     ((x + 5*size - 1) < 0)        || // Clip left
     ((y + 8*size - 1) < 0)){         // Clip top
    return;
  }

  setAddrWindow(x, y, x+6*size-1, y+8*size-1);

  line = 0x01;        // print the top row first
  // print the rows, starting at the top
  for(row=0; row<8; row=row+1){
    for(i=0; i<size; i=i+1){
      // print the columns, starting on the left
      for(col=0; col<5; col=col+1){
        if(Font[(c*5)+col]&line){
          // bit is set in Font, print pixel(s) in text color
          for(j=0; j<size; j=j+1){
            pushColor(textColor);
          }
        } else{
          // bit is cleared in Font, print pixel(s) in background color
          for(j=0; j<size; j=j+1){
            pushColor(bgColor);
          }
        }
      }
      // print blank column(s) to the right of character
      for(j=0; j<size; j=j+1){
        pushColor(bgColor);
      }
    }
    line = line<<1;   // move up to the next row
  }
}
//------------ST7735_DrawString------------
// String draw function.
// 16 rows (0 to 15) and 21 characters (0 to 20)
// Requires (11 + size*size*6*8) bytes of transmission for each character
// Input: x         columns from the left edge (0 to 20)
//        y         rows from the top edge (0 to 15)
//        pt        pointer to a null terminated string to be printed
//        textColor 16-bit color of the characters
// bgColor is Black and size is 1
// Output: number of characters printed
uint32_t ST7735_DrawString(uint16_t x, uint16_t y, char *pt, int16_t
textColor){
  uint32_t count = 0;
```

```
  if(y>15) return 0;
  while(*pt){
    ST7735_DrawCharS(x*6, y*10, *pt, textColor, ST7735_BLACK, 1);
    pt++;
    x = x+1;
    if(x>20) return count;  // number of characters printed
    count++;
  }
  return count;  // number of characters printed
}

//-----------------------fillmessage-----------------------
// Output a 32-bit number in unsigned decimal format
// Input: 32-bit number to be transferred
// Output: none
// Variable format 1-10 digits with no space before or after
char Message[12];
uint32_t Messageindex;

void fillmessage(uint32_t n){
// This function uses recursion to convert decimal number
//   of unspecified length as an ASCII string
  if(n >= 10){
    fillmessage(n/10);
    n = n%10;
  }
  Message[Messageindex] = (n+'0'); /* n is between 0 and 9 */
  if(Messageindex<11)Messageindex++;
}
//********ST7735_SetCursor*****************
// Move the cursor to the desired X- and Y-position.  The
// next character will be printed here.  X=0 is the leftmost
// column.  Y=0 is the top row.
// inputs: newX  new X-position of the cursor (0<=newX<=20)
//         newY  new Y-position of the cursor (0<=newY<=15)
// outputs: none
void ST7735_SetCursor(uint32_t newX, uint32_t newY){
  if((newX > 20) || (newY > 15)){       // bad input
    return;                             // do nothing
  }
  StX = newX;
  StY = newY;
}
//-----------------------ST7735_OutUDec-----------------------
// Output a 32-bit number in unsigned decimal format
// Position determined by ST7735_SetCursor command
// Color set by ST7735_SetTextColor
// Input: 32-bit number to be transferred
// Output: none
// Variable format 1-10 digits with no space before or after
void ST7735_OutUDec(uint32_t n){
  Messageindex = 0;
  fillmessage(n);
  Message[Messageindex] = 0; // terminate
  ST7735_DrawString(StX,StY,Message,StTextColor);
  StX = StX+Messageindex;
  if(StX>20){
    StX = 20;
    ST7735_DrawCharS(StX*6,StY*10,'*',ST7735_RED,ST7735_BLACK, 1);
  }
```

```
}



#define MADCTL_MY  0x80
#define MADCTL_MX  0x40
#define MADCTL_MV  0x20
#define MADCTL_ML  0x10
#define MADCTL_RGB 0x00
#define MADCTL_BGR 0x08
#define MADCTL_MH  0x04

//------------ST7735_SetRotation------------
// Change the image rotation.
// Requires 2 bytes of transmission
// Input: m new rotation value (0 to 3)
// Output: none
void ST7735_SetRotation(uint8_t m) {

  writecommand(ST7735_MADCTL);
  Rotation = m % 4; // can't be higher than 3
  switch (Rotation) {
   case 0:
     if (TabColor == INITR_BLACKTAB) {
       writedata(MADCTL_MX | MADCTL_MY | MADCTL_RGB);
     } else {
       writedata(MADCTL_MX | MADCTL_MY | MADCTL_BGR);
     }
     _width  = ST7735_TFTWIDTH;
     _height = ST7735_TFTHEIGHT;
     break;
   case 1:
     if (TabColor == INITR_BLACKTAB) {
       writedata(MADCTL_MY | MADCTL_MV | MADCTL_RGB);
     } else {
       writedata(MADCTL_MY | MADCTL_MV | MADCTL_BGR);
     }
     _width  = ST7735_TFTHEIGHT;
     _height = ST7735_TFTWIDTH;
     break;
  case 2:
     if (TabColor == INITR_BLACKTAB) {
       writedata(MADCTL_RGB);
     } else {
       writedata(MADCTL_BGR);
     }
     _width  = ST7735_TFTWIDTH;
     _height = ST7735_TFTHEIGHT;
    break;
   case 3:
     if (TabColor == INITR_BLACKTAB) {
       writedata(MADCTL_MX | MADCTL_MV | MADCTL_RGB);
     } else {
       writedata(MADCTL_MX | MADCTL_MV | MADCTL_BGR);
     }
     _width  = ST7735_TFTHEIGHT;
     _height = ST7735_TFTWIDTH;
     break;
```

```
  }
}


//------------ST7735_InvertDisplay------------
// Send the command to invert all of the colors.
// Requires 1 byte of transmission
// Input: i 0 to disable inversion; non-zero to enable inversion
// Output: none
void ST7735_InvertDisplay(int i) {
  if(i){
    writecommand(ST7735_INVON);
  } else{
    writecommand(ST7735_INVOFF);
  }
}
// graphics routines
// y coordinates 0 to 31 used for labels and messages
// y coordinates 32 to 159  128 pixels high
// x coordinates 0 to 127   128 pixels wide

int32_t Ymax,Ymin,X;        // X goes from 0 to 127
int32_t Yrange; //YrangeDiv2;

// *************** ST7735_PlotClear ********************
// Clear the graphics buffer, set X coordinate to 0
// This routine clears the display
// Inputs: ymin and ymax are range of the plot
// Outputs: none
void ST7735_PlotClear(int32_t ymin, int32_t ymax){
  ST7735_FillRect(0, 32, 128, 128, ST7735_Color565(228,228,228)); // light
grey
  if(ymax>ymin){
    Ymax = ymax;
    Ymin = ymin;
    Yrange = ymax-ymin;
  } else{
    Ymax = ymin;
    Ymin = ymax;
    Yrange = ymax-ymin;
  }
  //YrangeDiv2 = Yrange/2;
  X = 0;
}

// *************** ST7735_PlotPoint ********************
// Used in the voltage versus time plot, plot one point at y
// It does output to display
// Inputs: y is the y coordinate of the point plotted
// Outputs: none
void ST7735_PlotPoint(int32_t y){int32_t j;
  if(y<Ymin) y=Ymin;
  if(y>Ymax) y=Ymax;
  // X goes from 0 to 127
  // j goes from 159 to 32
  // y=Ymax maps to j=32
  // y=Ymin maps to j=159
  j = 32+(127*(Ymax-y))/Yrange;
  if(j<32) j = 32;
  if(j>159) j = 159;
```