**Solar Powered Motor Drive**

**Submitted To**

**Travis Forbes**

**Dr. Ross Baldick**

**The University of Texas at Austin**

**Prepared By**

**Niran Morar**

**Achal Shah**

**Nesreen El-Khalil**

**Upahar Sood**

**EE464 Senior Design Project**

**Electrical and Computer Engineering Department**

**University of Texas at Austin**

**Fall 2010**

**CONTENTS**

**TABLES iv**

**FIGURES v**

**EXECUTIVE SUMMARY vi**

**1.0 INTRODUCTION 1**

**2.0 DESIGN PROBLEM STATEMENT 2**

**2.1 OPERATION REQUIREMENTS 2**

**2.1.1 Illumination Levels 2**

**2.1.2 Motor Power Characteristics 2**

**2.1.3 Continuous Motor Operation 3**

**2.2 ENVIRONMENTAL REQUIREMENTS 3**

**3.0 DESIGN PROBLEM SOLUTION 4**

**3.1 H-BRIDGE INVERTING CIRCUIT 5**

**3.1.1 Input/Output Specifications 5**

**3.1.2 Operation Requirements 6**

**3.1.3 Design Choice 6**

**3.2 PULSE WIDTH MODULATION 7**

**3.2.1 Input/Output Specifications 7**

**3.2.2 Operation Requirements 7**

**3.2.3 Design Choice 7**

**3.3 USER INTERFACE 8**

**3.3.1 Input/Output Specifications 8**

**3.3.2 Operation Requirements 8**

**3.3.3 Design Choice 9**

**3.4 TRACKING ALGORITHM 9**

**3.4.1 Input/Output Specifications 9**

**3.4.2 Operation Requirements 9**

**3.4.3 Design Choice 10**

**3.5 ADDITIONAL CONSIDERATIONS 10**

**3.6 ESITMATED COST ANAYSIS 10**

**4.0 DESIGN IMPLEMENTATION 12**

**4.1 H-BRIDGE INVERTING CIRCUIT 12**

**4.2 PULSE WIDTH MODULATION 12**

**4.3 TRACKING ALGORITHM 12**

**4.4 USER INTERFACE 13**

**5.0 TEST AND EVALUATION 14**

**5.1 SUBSYSTEM TESTING 14**

**5.1.1 H-Bridge Inverting Circuit 14**

**5.1.2 PWM 16**

**5.1.3 User Interface 17**

**5.1.4 Tracking Algorithm 18**

**5.2 SYSTEM TESTING 19**

**5.2.1 Phase One Testing 19**

**5.2.2 Phase Two Testing 19**

**5.2.3 Phase Three Testing 20**

**6.0 TIME AND COST CONSIDERATIONS 21**

**7.0 SAFETY AND ETHICAL ASPECTS OF DESIGN 23**

**8.0 RECOMMENDATIONS 25**

**9.0 CONCLUSIONS 28**

**REFERENCES 29**

**APPENDIX A – SCHEMATIC DIAGRAMSA-1**

**APPENDIX B – PCB LAYOUTSB-1**

**APPENDIX C – CODE DOCUMENTATION C-1**

**TABLES**

1 PWM Options and Differences8

2 Major Expenses *11*

3 Recorded IC Driver Pin Output *15*

**FIGURES**

1 System Block Diagram 5

2 VA-C MOSFETs Operation *6*

3 H-Bridge Switch Output *16*

4 Sinusoidal Three Phase PWM *17*

**EXECUTIVE SUMMARY**

The purpose of this report is to detail our design solution and final evaluation results of our Solar

Powered Motor Drive prototype, a design project we developed in EE364D. Our group designed

a prototype of a system that incorporates the grid independence provided by solar panels as well

as the efficiency of three-phase conversion. Traditional motor systems implement single-phase

conversion, which have less efficiency compared to three-phase motor systems. Our system

improves upon this existing design by using three-phase conversion to deliver maximum power

to an attached motor.

During the first half of the report, we discuss the design problem our project addresses. The

efficiency levels found in single-phase motor systems is relatively low compared to three-phase

systems, and these inefficient systems are also characterized by a high occurance of motor stalls.

Our solar powered motor drive improves upon existing single-phase systems by using three-

phase conversion and a feedback system to maximize power output. The operational and

environmental requirements are also listed such has continuous motor operation, and

sunlight.We then describe the high-level solution our group compiled. We provided a flow

diagram that illustrated the major modules as well as the interactions amongst them.

The major modules include the H-bridge inverting circuit, pulse width modulation module, the

user interface, and the tracking algorithm. Along with the description of each module’s

operation, we detail the design choices we considered, and our final decision. Our design choice

for each module entailed some challenges that our group had to overcome. Later, our design

choices were tested, and the results were recorded.

During the second half of the report, a section is dedicated to system integration and testing. Our

overall system testing was divided into three phases, and we outline each phase of testing with

the procedures we conducted and the results we obtained. After the system and integration

testing section, we talk about some time and cost considerations that we encountered. The costs

we incurred were $430 over our estimation.The bulk of this extra cost came from the purchase of

a new MCBSTR9 microcontroller. Our time constraints follow our cost analysis. Due to our

microcontroller burning out and the delays during some testing phases, we were unable to

integrate the H-bridge inverting circuit with the other modules. Also, we were unable to purchase

PCB boards for our system due to the time constraints mentioned later in the paper.

Safety and Ethics are important factors our group investigated. We listed our major safety and

ethical concerns in order of importance. Safety concerns are related to the high voltage and high

current nature of our system, an d the ethical aspects are related to our project’s utilization of

green energy. Finally, at the end of the report, we provide some final recommendations for

groups in the future who want to expand on our project. Our final recommendations give some

insight into some opportunities that may have improved the development of our project.

**1.0 INTRODUCTION**

The purpose of this report is to detail our design solution and final evaluation results of our Solar Powered Motor Drive prototype, a design project we developed in EE364D. Ultimately, the converter system will be used in areas without stable power supplies such as rural Texas or remote areas of Africa and the Middle East to power applications such as water pumps. Our group designed a prototype of a system that incorporates the grid independence provided by solar panels as well as the efficiency of three-phase conversion. Traditional motor systems implement single-phase conversion, which have less efficiency compared to three-phase motor systems. Our system improves upon this existing design by using three-phase conversion to deliver maximum power to an attached motor. The product includes a three-phase converter that consists of an H-bridge inverting circuit and a microcontroller.

Two team members, Niran and Achal, have already studied power systems and motor properties during previous courses. The other members, Nesreen and Upahar, have experience with software systems and programming. With their previous knowledge and our collective interest in the technical topics, we decided to undertake this project initiated by Professor Ross Baldick. In this report, we will describe the design problem our project addresses by discussing our project goals, project motivations, and design requirements. Then we move to detail our design solution along with specifications and specific criteria that must be met for a successful project. We also provide a system flow diagram for a high-level illustration of our system and the major modules: the tracking algorithm, the H-Bridge inverter circuit, the pulse width modulation (PWM) module, and the user interface. After this section, we detail each module of our system in depth by discussing design choices, design challenges and testing results relevant to each module. We continue to discuss system integration and testing. This section details the three phases of testing we conducted to evaluate our project as a whole. Later, we dedicate a section to time and cost considerations for our project as well as a section dedicated to safety and ethical issues of our project. Finally, we conclude our report with a description of several final recommendations.

**2.0 DESIGN PROBLEM STATEMENT**

Our group designed a solar powered motor drive that incorporates the strengths of three-phase conversion as well as the advantages of grid-independence. The efficiency levels found in single-phase motor systems is relatively low compared to three-phase systems, and these inefficient systems are also characterized by a high occurance of motor stalls. Our solar powered motor drive improves upon existing single-phase systems by using three-phase conversion and a feedback system to maximize power output. By avoiding the use of line power and utilizing solar panels, the system is a self-regulating, off-the-grid product that converts solar energy into three-phase power, which can be used to automate and facilitate mechanical processes such as running a motor. Our final product can be utilized for water pumps, which provides users with benefits such as maximum power output, easy mobility, and motor longevity. The following sections detail the operational and environmental requirements that must be met for a successful project.

**2.1 OPERATIONAL REQUIREMENTS**

Three key operational requirements of the Solar Powered Motor Drive are indicative of our system's proper operation. The following sections will address the requirements in order of importance, starting with the measurement of illumination levels, input of motor characteristics, and the continuous operation of the motor.

**2.1.1 Illumination Levels**

The illumination level will be measured in terms of Watts per meter and will be obtained from an illumination reading device, a light dependant resistor, placed near the solar panel. The intensity is an integral part of the tracking algorithm. Given the efficiency of solar panels, (10-15%), our expected output efficiency levels are around 10-12% [1], taking into consideration the power losses from the circuit. The power losses are associated with losses over the inverter circuit due to components such as diodes, transistors, and capacitors.

**2.1.2 Motor Power Characteristics**

Providing the power characteristics of the different types of motors is crucial to extract maximum power output. Failure to follow the power characteristic curve would cause the motor to stall, overheat and possibly burnout. The motor data will be supplied to our system through the user interface.

**2.1.3 Continuous Motor Operation**

Motor stalls may indicate that our tracking algorithm is not providing the correct feedback and can also suggest that the input voltage is under operational levels. For example, after sunset, the motor will not work since there is no minimum input voltage of 25 V being provided. The minimum value might change once we acquire the motor and perform tests that confirm the threshold voltage is within acceptable limits. However, if during operation the input voltage falls below the threshold or significantly drops from the previous input voltage, the motor could cease to work.

**2.2 ENVIRONMENTAL REQUIREMENTS**

Two key environmental requirements are required to operate our system: protective casing and sunlight. First, a protective casing is needed to house the circuitry and microcontroller. Since the final product will be used in an outdoor setting, we have to ensure that external elements do not interfere with the operation of the system or damage the product. Our prototype will be housed in a waterproof and electrically insulated casing with the solar panels placed at a height greater than two feet above ground level.

Another requirement for system operation is sunlight. The three-phase motor and the microcontroller utilize solar energy to operate. A minimum of 5 V needs to be supplied by the solar panels to initiate a working system [2]. To ensure maximum energy efficiency, the device will be placed in an open location like a field where it has unhindered and continuous exposure to the sun. The solar panels should be positioned at an appropriate angle on a stand depending on the geographic location and season to ensure appropriate sunlight access at different times of year.

**3.0 DESIGN PROBLEM SOLUTION**

Our system converts solar energy to a three-phase signal to be supplied to a three-phase inductive motor. Feedback from the motor is supplied back to the system to adjust the frequency of the three-phase signal to maximize the motor’s output efficiency. We determined that our system requires a DC to three-phase AC conversion system and a feedback control system. These systems can be implemented either as a hardware-based product or as a mixture of a hardware and software.

We decided to implement the system using a combination of hardware and software components. The hardware components are the H-bridge inverting circuit, the DC to three-phase conversion system, and the microcontroller, the feedback control system. The microcontroller will be composed of a three software items: PWM, tracking algorithm, and user interface. The microcontroller manipulates the signal to ensure that the output three-phase signal will operate the connected motor load at its maximum potential, thus meeting our operational requirment. The operations within the microcontroller also ensure the absence of motor stalls, another operational requirement.

The operation of our system can be summarized in four basic steps. First, the H-bridge circuit takes the DC source and converts it into a three-phase signal with the use of the PWM module. The PWM waveform is determined by the tracking algorithm. The tracking algorithm itself tracks the solar intensity, current and rotational speed of the motor through the use of sensors. The tracking algorithm also considers the user input of motor and solar panel characteristics to calculate the optimal operating frequency of the motor. Figure 1 illustrates the interactions of the four basic steps. In the following sections, we will discuss our reasoning behind our final design choices for each module, including a cost/benefit analysis, and how our design choice meets our project specifications.

******

Figure 1: System Block Diagram

**3.1 H-BRIDGE INVERTING CIRCUIT**

The H-bridge inverting circuit is responsible for the conversion of the DC voltage to a three-phase signal using three pairs of MOSFETs orientated in parallel, with the phase output in between each pair. Also, the inverting circuit acts as power regulator when combined with a PWM. The following sections describe the input/output specifications, operational requirements, and design choice.

**3.1.1 Input/Output Specifications**

The circuit has two inputs, solar panels and PWM signal, and one output, a three-phase signal. The solar panel is the DC input source to the H-bridge inverting circuit. The PWM waveform directs each MOSFET to switch on and off. Each phase that composes the output signal is produced in a similar manner. An example of the phase being produced is shown in Figure 2.

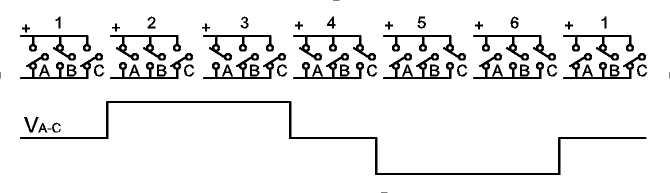


Figure 2: VA-C MOSFETs Operation (From Source [3])

**3.1.2 Operational Requirements**

This module has four operational requirements: voltage, current, store capacity, and MOSFET switching rate. To fulfill output voltage and current as well as storage capacitance and switching frequency, we are excepting to operate motors with voltage ratings up to 250 VAC and current ratings up to 30 A. The associated frequencies range from 0 to 60 Hz. Therefore, the MOSFETs must handle these maximum ratings. Due to the expected voltage ratings of the motors, we will use a power capacitor to store twice as much energy then required to output a smooth three-phase signal; therefore, we will use a capacitor that can store up to 500V. The switching frequency that the MOSFETS need to handle is up to 150 kHz because the PWM samples up to 150 kHz to obtain a smooth sinusoidal three-phase output.

**3.1.3 Design Choice**

We have chosen to design our system with the six-MOSFET H-Bridge inverting circuit as opposed to three single-phase H-Bridge inverting circuits because the former design requires fewer components; we will utilize six fewer MOSFETs, and we will not need a phase corrector to sync the output into a three phase signal. Fewer components leads to heat reduction. However, rigorous testing will be required to test the six-MOSFET H-Bridge inverting circuit to ensure proper operation.

**3.2 Pulse Width ModulatioN**

The PWM control system produces a waveform based on the input of the tracking algorithm that feed into the H-Bridge inverting circuit to produce the three-phase signal. The details of the input/output specifications, operational requirements, and design choice are provided below

**3.2.1 Input/Output Specifications**

The PWM consists of one input and one output, the tracking algorithm and a three-phase sinusoidal PWM waveform, respectively. Specifically, the PWM receives an updated carrier frequency from the tracking algorithm.The output of the PWM is square wave represention of a sinewave. The square wave is produced by comparing the sinewave with a triangle wave. The square waveform’s high and low values correspond to the on and off states of the MOSFETs in the H-bridge inverting circuit.

**3.2.2 Operational Requirements**

There are two main requirements associated with the PWM. The first requirement is a maximum sampling rate of 150 kHz to correspond with the MOSFETs used in the H-bridge inverting circuit. By sampling at 150 kHz, the H-bridge inverting circuit will be able to produce a smooth sinusoidal waveform to be fed into the motor. The second requirement is that the PWM should be able to produce reference signals with different periods. This specification helps change the carrier frequency output of the H-bridge inverting circuit.

**3.2.3 Design Choice**

We decided to go with a microcontroller version of the PWM rather than the circuit-based PWM because the ability to modify and debug the system exists. Debugging and changing the circuit-based PWM is much more difficult because this will require redesigning the circuit. Also, using the microcontroller over the circuit will reduce the number of components as well as heat production. The only difference between the two choices is the conversion rate, which is negligible. The features of each system are outlined in Table 1 below.

Table 1: PWM Options and Differences

|  |  |
| --- | --- |
| Circuit-based PWM | Microcontroller-based PWM |
| * Fast Operation | * Eliminates the need of a separate PWM circuit |
| * Built and tested prototype | * Cost friendly |
| * Buffered chips can overheat | * Negligible heat production |
| * Cost wise ineffective | * Easier to debug and program |
| * More components to take care of | * Comparitively slow operation |
|  | * Dependent on micro controller  power supply |

**3.3 User Interface**

The user interface allows the user to specify the motor characteristics that is later used by the tracking algorithm to calculate the optimal operating frequency of the motor. To implement the solution successfully, certain input/output specifications and operational requirements must be met, which we specify below, followed by our design choice.

**3.3.1 Input/Output Specifications**

The module consists of two inputs, an on/off switch and a keypad, and two outputs, the system state and the LCD display. For the input of the motor and solar panel characteristics, the user enters numerical values through the keypad. The on/off switch determines the state of the system. The LCD displays the menu items and promptsthe user for information.

**3.3.2 Operational Requirments**

There are two operational requirements for the module: the receipet of comprehensive data and the handling of error states. First, the user interface must be comprehensive and obtain all the required information to send to the tracking algorithm.Any data not collected may skew the calculations in the tracking algorithm. Second, the interface must be capable of error handling user input such as invalid numbers and data pair entries.

**3.3.3 Design Choice**

We decided to implement a LCD/keypad interface as opposed to a keyboard/serial monitor interface. The serial monitor interface is more user-friendly because it has better debugging capabilities; however, the interface does require a connection to a PC. Even though you cannot display a lot of information on the LCD, the LCD/keypad interface can ultimately be used remotely which is why we decided to implement this interface. The implementation will use the on-board LCD and a 4x3 keypad.

**3.4 Tracking Algorithm**

The main function of the tracking algorithm is to provide a feedback system to control the motor frequency. The following sections detail all the required specifications and requirements of the tracking algorithm as well as our final design choice.

**3.4.1 Input/Output Specifications**

The tracking algorithm requires three inputs: the motor characteristics, motor sensor data, and illumination levels. It generates one output, an optimal frequency value. The motor characteristics will be used in the calculations that determine the motor characteristic curve. The illumination levels help to determine where on the characteristic curve lies the optimal frequency. The motor sensors help by providing a safeguard against operating outside the frequency range on the characteristic curve. By locating the optimal frequency, the algorithm outputs the frequency value to the PWM module, thus altering the period of the waveform. With the curve and the given illumination level, the optimal frequency is calculated during every loop iteration in the feedback system. The algorithm requires the motor characteristics, illumination levels, and motor sensor data as input and utilizes dynamic updating to output the desired frequency to the motor.

**3.4.2 Operational Requirements**

Two requirements are needed to ensure correct operation of the tracking algorithm. Optimally, each calculation through the feedback loop should be updated with a frequency of 10 Hz. By updating the frequency every 1/10th of a second, a gradual increase or decrease in the frequency can happen at a faster rate, thus becoming unnoticeable to the motor. Also, the tracking algorithm should be sensitive to motor slippage (stalls). Instead of simply calculating the frequency value, the algorithm should also check the motor sensors and determine if the current frequency lies outside the normal range.

**3.4.3 Design Choice**

We chose to implement a closed-loop feedback based system as opposed to the open-loop feedback system. This decision was made based on the basis that our motor will be rotating faster than 1000 rpm. The closed feedback system will be implemented using a QRB1134 reflective object sensor [4]. The sensor is a system with an infrared emitting diode and a highly sensitive phototransistor. The sensor will be attached to the shaft of the motor and will sense light pulses through holes within the shaft of the operating motor. Using captured sensor measurements, through input-capture channels in the microcontroller by and calculating times for one rotation (four holes), we will estimate the torque and mechanical frequency characteristics for the motor.

**3.5 Additional conSiderationS**

One additional consideration we had to take into account is the type of microcontroller we are utilizing. The microcontroller used for the control system will be the Arm Cortex-M3 based STM32F1003 [2]. It has a voltage requirement of 2.0 to 3.6 V to be powered and will be powered using a regulated voltage line from the solar panel to ensure its power requirements are met. It has a maximum clock frequency of 72 MHz, which is appropriate for our application to generate varying PWM signal frequencies and reflective sensor analog to digital sampling rates. It has 64 high-speed I/O ports.

**3.6 ESTIMATED Cost Analysis**

The total estimated cost of our prototype is approximately $1247. The biggest chunk of our expenses pertains to hardware. The single biggest expense that stands right now is the solar panels at a market value of $750, 56% of the total expected cost [5]. Fortunately since there are solar panels mounted on the ENS rooftop, our team plans to utilize them for testing purpose. This also helps us reduce our cost initially. The next biggest expense is the three-phase motor marked at price of $192 [2]. The microcontroller that we are planning to use in our project is also expensive. It costs about $200, but fortunately we have been provided with it, in a previous class. So we are just going to utilize it. Some of the key features that this microcontroller has are PWM and low-power consumption. The other major part of our project is the H-Bridge inverting circuit. The construction should cost us close to $60 [6]. Components of the circuit include resistors, transistors, and PCB board. Table 2, below, provides a complete outline of our major expenses.

Table 2: Major Expenses

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Component** | **Item Description** | **Quantity** | **Est. Cost/item($)** | **Est. Overall Cost($)** |
| Cooling Fan |  | 1 | 2.00 | 2.00 |
| Microcontroller | ARM STM 32 | 1 | 200.00 | 200.00 |
| Transistors | Mouser #512-IRFP140A | 6 | 5.00 | 30.00 |
| Casing | 12x6x6 inches | 1 | 50.00 | 50.00 |
| Capacitors | 250 V 1500 uF | 1 | 30.00 | 30.00 |
| Inductors | H-Bridge circuit | 1 | 1.50 | 1.50 |
| Motors | Load purposes | 1 | 192.00 | 192.00 |
| Steel Stand | For solar panels | 1 | 50.00 | 50.00 |
| Solar panels | Power Source | 1 | 750.00 | 750.00 |
| PCB/breadboard | Choice | 1 | 30.00 | 30.00 |
|  |  |  | Total Cost | 1247.00 |

Since we are using the microcontroller provided to us in previous classes, the most suitable environment for us to code will also be the one we used the last time. We will opt for free licensed programs available to university students. The KEIL development software needed for developing microcontroller code is available for free on the manufacturer’s website.

**4.0 DESIGN IMPLEMENTATION**

In the following section, we talk about our design challenges and implementation of each module. We include details about modifications we made to the original design solution.

**4.1 H-Bridge Inverting Circuit**

Our original implementation of the H-bridge utilized the IC driver chip for each pair of MOSFETs. The chip took an input, a PWM signal, and converted it into an inverted and non-inverted signal. Testing this design with a simulated PWM signal using the Agile wave generator, we found that the low output pin of the IC driver chip was not outputting a signal. This result caused a change in implementation to an inverting chip, TC1426CPA, non-inverting chip, TC1427CPA, and optocoupler, 6N136, circuit. With this type of implementation, the inverting chip and non-inverting chip are placed separately between a pair of MOSFETs. The new implementation was tested with a PWM signal and both chips were able to output an amplified PWM signal to fire the MOSFETs.This implementation not only fire the MOSFETs but also adds electrical protection to the microcontroller to prevent electrical feedback.

**4.2 Pulse Width Modulation**

We implemented this module using the MCBSTM32 microcontroller. We achieved simple PWM outputs; however, when trying to produce a sinusoidal three-phase PWM, we experienced two issues. First, synchronization between the phases were not maintained which may cause issues with operation of the motor. Second, the module was generating only half of the waveform. These setbacks prompted us to purchase a MCBSTR9 microcontroller which consisted of a built-in motor controller as well as 128 ports. We successfully obtained a three-phase PWM signal with 100% accuracy in phase difference.

**4.3 Tracking Algorithm**

The outline of our algorithm design solution consisted of the implementation of three sensors: the current sensor, light sensor, and tachometer. We successfully implemented the light sensor and tachometer. For the light sensor, we looked at two implementations, a photo diode and a Light Depenent Resistor (LDR). The photo diode did not have transducer like properties. The LDR exhibited a linear relationship between voltage and light intensity. As a result, we implemented the light sensor using the LDR because of its greater accuracy. With regards to the tachometer, we built the simple circuit on a breadboard using the OPA2325. The schematic is located in Appendix C. However, we were unable to implement the current sensor due to fear of high voltage and current. We attempted many different circuitries using op-amps such as LM 741, LM833, and other differential and instrumention amplifier.

**4.4 User Interface**

We implemented the LCD/keypad interface because it can be used remotely which satisfies our project goal. We tried implementing a 4x4 matrix keypad, but we ran into issues with the microcontroller reading. We decided to implement a more familiar model, a 4x3 single-bus keypad. However, we did face some design challenges. We did have to take into consideration key bouncing. So, we put in delays to account for the switches; however, the LCD delays combined with the key delays create latency issues of reading the key being pressed on the keypad. Also, when designing the user interface, we took into consideration all the error states that might occur, such as invalid numbers or leaving an entry blank.

**5.0 TEST AND EVALUATION**

Our test and evaluation plan involved testing each module of our four modules: H-Bridge Inverting Circuit, PWM, Tracking Algorithm, and User Interface. After each module has been tested, we planned to test the overall system in three phases where we gradually combine each module. Our first phase was to combine the User Interface and the PWM. The next phase was to combine the User Interface, PWM, and Tracking Algorithm. The final phase was to combine this all modules together with the H-Bridge Inverting Circuit. The following sections will describe the complications and successes of each type of testing.

**5.1 SUBSYSTEM TESTING**

Testing each module of our system reduces the number of bugs encountered in system integration.The following sections detail our testing procedures and results of each module in our system.

**5.1.1 H-bridge Inverting Circuit**

The original H-Bridge circuit using a single IC driver chip for each pair of MOSFETs is documented in Appendix A tested and proven working by the following tests. We started out test building the circuit as describe in a PBC schematic layout. We first found that there was a missing ground from both the MOSFET as well as the capacitor from pins five and seven. This fault was found through debug when testing the low side for zero volts in comparison to the high side for ten volts. We were seeing that there were ten volts at the low side and 5 volts at the high side. During debug, we formulated the Table 3, where column two is from a working single phase H-Bridge Inverting Circuit were the third column showed the output of our pins before debug. The last column shows the output of the pins after correction.

Table 3: Recorded IC Driver Pin Output

|  |  |  |  |
| --- | --- | --- | --- |
| PIN | Working Model | Before Debug | After Debug |
| 1 | 0 V | 0.002 V | 0 V |
| 2 | 0 V | 11.61 V | 0 V |
| 3 | 11.68 V | 0 V | 11.68 V |
| 4 | 0 V | 0.394 V | 0 V |
| 5 | 0 V | 6.56 V | 0 V |
| 6 | 10.60 V | 6.56 V | 10.60 V |
| 7 | 11.87 V | 11.87 V | 11.87 V |
| 8 | N/A | N/A | N/A |
| 9 | N/A | N/A | N/A |
| 10 | N/A | N/A | N/A |
| 11 | 0 V | 9.85 V | 0 V |
| 12 | 0 V | 9.85 V | 0 V |
| 13 | 11.54 V | 11.61 V | 11.54 V |
| 14 | N/A | N/A | N/A |

We tested the Inverting driver chips with a simple 50 percent duty cycle PWM from the Agile waveform generator to the output of the low and high were operating correctly. From the output we realized that the expected input PWM signal needed to range +/- 12 volts, but the expected output of our PWM module ranges between 0 to 3.3 volts. Therefore we had to redevelop the circuit. The new circuit uses an optocoupler and an inverting or non-inverting chip, the schematic is located in Appendix A. Originally, when designing the circuit we didn’t understand the purpose behind the optocoupling chips and decided to leave them out. When connecting and testing the circuit with a variac, 25 volt transformer and diode bridge rectifier, we found that our circuit had a short because the current on the variac was greater than ten amps. We were expecting much less than one amp. At that point, we added the optocouplers and tested again and measured the current to be 1 mA . Next we tested the outputs at the MOSFET, and found that we could not read a signal. Then we started to test each chip and found that the 12 volt DC to DC converters were getting very hot. This was not a good sign, so we reinspected and rewired our circuit, which resolved the heating issue. We tested the new Inverting circuit with a 50 percent duty cycle PWM from the STM32 and the resulting output waveform is presented below in figure 3. The wave confirmed we were ready to test with our microcontroller PWM, which will explain in section 5.2.3 Phase Three Testing.

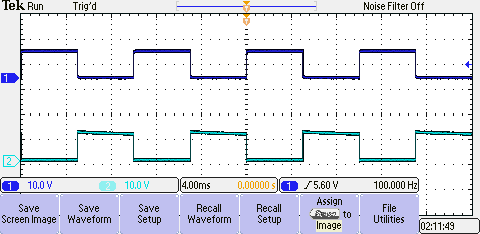


Figure 3: H-Bridge Switch Output

**5.1.2 PWM**

We started our coding of the PWM in the STM32 microcontroller in three steps: generating a simple square wave PWM with constant duty cycle, a varying duty cycle period PWM, and a sinusoidal PWM. The square wave PWM at constant duty cycle period was up and running within a week as seen in Figure 3. The varying period duty cycle took two more weeks with success. We were able to change the duty cycle from 10 to 100 hertz with steps of 0.5 hertz. In the third step we saw complications of the output results. Due to poor documentation of the STM32, the sinusoidal PWM was outputting a comparison waveform, lacking comparison points. We continued to debug the issue for three weeks where we sought help from Dr. Valvano and forums online. After searching endlessly, we decided to order the STR9 microcontroller. The STR9 is designed for motor control and three phase sinusoidal PWM generation.

Within two and half weeks, we had the sinusoidal PWM up and running with a few issues. Some output ports were not outputting and there was a one second delya with the output PWM signal. The missing output pins were resolved with by debugging source code and the one second delay was resolved by re-soldering the power and ground pin headers. The corrected waveform is shown below in figure 4. We will discuss how the integration progress in sections 5.2.1 Phase One Testing and 5.2.2 Phase Two Testing.

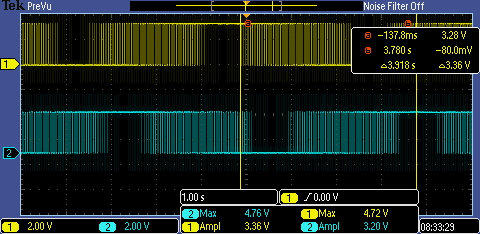


Figure 4: Sinusoidal Three-Phase PWM

**5.1.3 User Interface**

The user interfaces was done in two parts: keypad and LCD. Two different keypads were used: 4x4 keypad and a 4x 3 keypad. The main difference between the two keypads was the type of configuration. The matrix configuration of the 4x4 keypad uses a sense and read combination to read a pushed key. This configuration was causing issues with our microcontroller. We had to resolve the mid-state issue, where the key is between the on and off position. This issue caused the microcontroller to freeze or lock up. This issue was resolved, and we could read in keys. Next, we compared this method to the series keypad and found that the key read faster. Therefore, we decided to stay with the 4x 3 keypad. To test the key pressing, we used a two-step method: one on breadboard the other on the microcontroller. On the breadboard, we tested different keys and verified that only a single led lit up. The next step was to map the input value of a key on the microcontroller. We used two different General Purpose Input Ouput (GPIO) ports. For each GPIO port, we used a selective number of ports; therefore, we had to develop masks to read only those ports. After determining the masks, we pushed each key and recorded what value was associated with each key. The correlation of the keypad pins to the microcontroller pins is located in Appendix C.

Next, we got the LCD and keypad to interface together. Most of the LCD code was given as starter code with the microcontroller. A few newer fuctions had to be written for writing the keypad value and writing decimal values to the LCD display. The only issue that arose with testing the user interface was the latency between key pressing and key reading. During parts of the menu prompts, a key button had to be pressed for longer than a second for the number to be read. At this point, we were ready to integrate the user interface, despite the latency issues, with the rest of the modules which will be discussed in 5.2.1 Phase One Testing.

**5.1.4 Tracking Algorithm**

The algorithm will be written in C language and testing using the KEIL software development program. The algorithm had two parts: tracking the power and tracking the light intensity. Each part of the code is a separate interrupt where the power is calculated every second and the light intensity is calculated every five seconds. Reading times were determined by our group and sponsor on the basis that there is cloud movement constantly and reading second or faster will cause us to contantly change the frequency and cause us to loose output motor effiecently. The simplier of the two algorithms was tracking the power. The the algorithm will read in a current and RPM of the motor and relate it to voltage calculate a power and compare it to a previously calculated power. If the power has increased, the algorithm will increase the frequency of the PWM; otherwise, decrease the frequency. The other algorithm works by reading in the light intensity and references to a voltage and frequency. Then it accordingly attempts to reach the target frequency by either increases or decreasing the the frequency by one hertz every second untill the target is reached. This frequency decrease takes precedent over the tracking of the power. We allowed a tolerance of two percent in power changes and a 5 percent in light intensity changes. We observed the frequency value being updated by viewing the value on the LCD.

To read the motor’s speed, current, and light intensity, we had to develop circuitry. The motor’s speed is calculated using a tachometer and the OPA2325 shown in Appendix A. The tachometer was developed and tested in a previous course. The code is known to work correctly. The light intensity sensor was developed using a LDR and a 10 kilo-ohm resistor shown in Appendix A. To test the LDR circuit, we used an LED flashlight and oscilloscope in a dark room and observed the changing values corresponding to the varying light intensities. As for the current sensing circuit, we looked into different op-amps available in ENS to judge what type of circuity and chip was needed. We found that no chips were available in ENS that could sustain measurements greater than one amp. This setback stopped us from making a decision of what chip to get because the shipping prices cost more than the part itself. So at the end, we decided to leave out the current sensing circuit. With the completion of the sensors, we were ready to integrate the tracking algorithm with the other modules in section 5.2.2 Phase Two Testing.

**5.2 SYSTEM TESTING**

To evaluate our project as a whole, we conducted a comprehensive system test. The testing was divided into three phases, and the procedures for each phase are described below.

**5.2.1 Phase One Testing**

Our phase one testing involves combining the PWM and the User Interface. In initial integration, we found that the user interface was adding a time delay due to pins used in the PWM module which controlled a timer. This issue was resolved by remapping the input to another GPIO port. Determining the root cause behind this delay took some time with reading of some technical documents of the microcontroller. The next step was to see if we were able to start and stop the PWM with a button. This button was needed to ensure that the motor is not running when new data is been inserted for new solar panels or motor; otherwise, a motor running on a wrong configuration could burn out. Next, we tested to see if the lowest frequency was beening applied to the PWM. We tested this with three points 20 Hz, 50 Hz, and 120 Hz.

**5.2.2 Phase Two Testing**

In Phase Two Testing, we integrated the tracking algorithm to the PWM and the user interface. Due to lab restrictions, we were not able to test the PWM changes and constraints placed by the user interface. But the code integration went through smoothly, seeing no effect on output of the PWM. Because we did not implement a current sensor, we had to conduct this phase of testing without the tracking algorithm.

**5.2.3 Phase Three Testing**

In Phase Three, we integrated the H-Bridge inverting circuit with the other modules. We planned to start testing by first turning off the tracking algorithm and using a constant sinuosoidal PWM. At this point, we started to test our cicuit with lab instructions from EE362L to see if our H-Bridge was operating properly. We placed the oscilloscope probes on the drain and source of the MOSFETs to observe the firing of the MOSFETs.We observed the firing of the MOSFETs; however, when we tried to reproduce this switching on another pair, our microcontroller burned out, causing us to stop testing. We were unable to complete testing of our H-Bridge circuit.

**6.0 TIME AND COST CONSIDERATIONS**

At the start of the semester, we developed a detailed plan on how each of the modules will be developed and integrated once they are completed. However we faced a multitude of problems in the development and design implementation phase given our inexperience with three-phase PWM generation and the H-bridge inverting circuit. We spent the first few weeks, researching and reading documents about each module and analyzing different techniques of generating PWM signals and determining which method helps in producing the most accurate modulation and phase difference required for correctly switching the MOSFETS on the H-bridge circuit. At the same time, we had to make sure we could generate signals at the appropriate frequency to avoid damaging the motor.

We started with the very basic approach; developing a triangle comparator wave at a high frequency and using a sine-table to compare and switch the PWM signal accordingly. However we did not anticipate that PWM generation with the MCBSTM32 would be unreliable and cause phase synchronization problems. After testing on the MCBSTM32 system proved fruitless after two weeks, we decided to switch to a new evaluation board, the MCBSTR9. The microcontroller added 250 dollars to our project costs. At the same time, we had to extend the PWM completion deadline by two weeks.

Once we received the microcontroller we realized that there was a huge learning curve to the motor controller module, since there was limited documentation available to understand the functionality of this module. We also had to modify the user-interface and sensor driver codes, developed originally for the MCBSTM32, to be compatible with the MCBSTR9 registers and ports. We were able to develop and translate the code but it cost us a week’s worth of time to understand the functionality of the multiple registers on the board.

The H-bridge module was developed and tested individually long before the three-phase PWM development was complete, hence we could not anticipate any future issues within the circuitry. Therefore once we finished the three-phase PWM signal generation, integration with the H-bridge brought up various problems in the circuit such as passive MOSFETs and current surges due to the unreliable nature of isolation circuitry. Addition of isolation circuitry and careful reconstruction of the circuit did seem to fix most issues and pressured by time we decided to integrate the PWM and the H-Bridge modules. However, after observing proper firing of the MOSFETS and the correct signal output through the H-bridge for an hour, we faced a burnout of the processor chip on the board. At that point, we had no option but to switch back to our original microcontroller and this cost us all the progress made in the past month.

We switched back to our original microcontroller, but the lessons learned from this microcontroller we were able to develop a working three-phase PWM signal at the correct frequency using the advanced timer capabilities of the new microcontroller. Experience with PWM generation and a better understanding of the functionality of important registers and interrupts made it easy to redevelop and transfer code back to the original microcontroller. However we could not get most of the tracking algorithm and the motor interface completed before the Open-house.

In the meantime we could not translate our circuit designs for the keypad, light, current and motor speed sensor and H-bridge circuitry to final PCBs owing to the uncertainty and unreliable nature of the system. We also had to consider the high costs, about 400 dollars, to get the PCBs shipped on time for the Open-House demonstration. Hence we decided to present our final design on breadboards. We have outlined some of the costs mentioned above in Table xx.

**7.0 SAFETY AND ETHICAL ASPECTS OF DESIGN**

Of all the possible safety and ethical aspects of our project, we will identify and detail three safety issues and three ethical aspects.The major safety concerns are high voltage and current transmission, MOSFET sensitivity to moisture and temperature, and isolation of microcontroller ports from the high current lines. After discussing our safety concerns, we will move on to discuss the ethical issues related to our solar powered motor drive. The three ethical aspects are green energy, grid independence and the possibility of electrocution.

One safety concern is the high voltage and current present in our system. Because the H-bridge module utilizes and outputs high levels of voltage and current, we plan to make sure that the user will know that they should be cautious. On a bright sunny day the solar panels can generate an appreciable amount of power/voltage. The most vulnerable areasa would be the lines that carry this high voltage and current from the solar panel to the H-bridge inverter and from the H-bridge inverter to the motor. To ensure safety, we recommend insulating these lines, so that the only live contacts would be the sockets. Moreover, there should be a warning label on the box indicating high voltage and current, and hence they should not be touched or tampered with.

Another safety concern lies with the MOSFETs. They are are crucial part for the H-bridge inverter module to correctly operate. At the same time, these MOSFETs are very sensitivie to moisture, temperature and electrostatic charge. Since the prototype will be placed out in the open environment in a casing, it is quite possible that the MOSFETs might acquire moisture on or near surrounding surfaces. And with high currents involved, this could potentially lead to arcing and short circuit behaviour. Moreover, because of the high amount of switiching of the MOSFETs and the closed casing, there is a possibility of undesirable increases in temperature. This could potentially lead to components or the whole project to burn out which is dangerous. To overcome this issue, the casing should include circulation which removes moisture and heat.

Lastly, there needs to be protection between the microcontroller and the H-bridge circuit from high current and voltage. The safety concern does not only apply to human life, but also to the equipment’s safety. Losing the microcontroller could cause the motor and the H-bridge circuit to burn out, causing harm to a person . We implemented optocouplers to help us accomplisht this, however our microcontroller still burned out. So we suggest using some addition protective circuitry to the existing design.

Now moving on to the ethical aspect of our project, one, the project heavily helps on working for public interest and helps to boost the green energy efforts. Second, because our project promotes off-the-grid-independece, it help local rural and remote areas. Lastly, one ethical issue we can foresee is the possibility of someone get electrocuted eventhough the system is labeled properly because is it located in a reach area.

**8.0 RECOMMENDATIONS**

This section outlines the recommendations that our present team consider to be essential and helpful to future teams. This part of the paper covers modifications to our approach that we would have made if we started over, and also future recommendations for teams planning to continue our project.

Looking back at our project, we identified certain areas to modify and improve, like modeling our complete circuitry, doing more research for support and documentation of our components, and starting our design phase in 364D. During our open house presentation, it was suggested to us that we should incorporate LabView simulations to model our circuitry. Using LabView to model our complete circuitry would help identify problem areas before physically implementing the circuit design.

We also faced significant problems in obtaining sufficient explanation and examples to utilize registers required to generate PWM signals.This in turn caused integration and compatibility problems. However, we figured out the initialization system and were able to successfully implement the user interface. We recommend the use of components and systems with adequate support so that we do not waste valuable time attempting to understand basic functionality.

Hence, we recommend starting the design phase in EE 364D. If we had been able to implement any of the modules during the course of the last semester we may not have faced integration and implementation problems. We would have gone into more detail with specific figures and calculations related to the PWM signal or the design of the H-bridge circuit. Consulting with other TAs and professors would also have helped gain more knowledge about our project.

Dr. Ross Baldick is going to carry forward with a different team. We have worked on the project for two long semesters and would like to give few future recommendations for other teams. This would help them prevent falling into any generic loopholes and also help provide clear precise deliverables. The future recommendations we have are adding protective circuitry, making an effective power management system, adding a battery to store, and implement a current sensor.

The PWM signal, that is generated from the microcontroller and is supplied to the H-bridge inverting module, is responsible for the modification of the frequency of the system. And since the H-Bridge circuitry has high amounts of voltage and current associated with it, we must make sure that the microcontroller is completely isolated from high current lines. We utilized optocouplers but even then our microcontroller burned out. Thus to avoid burning out more microcontrollers we would strongly suggest to use of additional protective circuitry as well as pretested breadboards to ensure adequate signal isolation.

We also suggest the implementation of a power management system. Two different modules, namely the microcontroller and the H-bridge inverting module, need different levels of voltage input. Moreover our project is eventually going to be implemented in a deserted environment in a self-sustaining manner, which means that we must be able to power the microcontroller and the H-bridge module from the solar panel itself. This means implementing a power management system that successfully accomplishes these using power regulators. Hence a certain percentage of the input voltage from the solar panel must be stepped down to tolerable level to ensure equipment safety. The use of a power regulator would also ensure that the equipment is not damaged during an inadvertent voltage or current spike. As mentioned ealier the solar panel could provide more voltage than what the system needs to function at its optimal level. Instead of wasting the useful energy future teams could design a battery system to store it. This stored energy from the battery could prove useful during extensive cloud cover or running the motor during nighttime.

We need to make sure that we have an accurate idea of the input signals to the three phase motor namely the voltage, current and the frequency. We could not develop a current sensing circuit to meet our needs and so we recommend the use of a high current tolerant current sensing circuit. There should be safety checks to make sure we do not supply current to the motor, and also an additional input to the tracking algorithm. With continued research we while we prepared for our final demonstration we identified the 10 ampere tolerant ACS714 hall-effect based current sensor as a viable option. We recommend future teams to consider the inclusion of this sensor in their design or research similar high current tolerant sensor circuitry. The current sensor is vital aspect of the closed-loop feedback system allowing the measurement of the power output of the motor.

Finally, we faced significant problems in testing, requiring frequent access to oscilloscopes to test our modules. We identified a special logic analyzer peripheral, compatible with both our microcontrollers, which is useful in visualizing and debugging signal outputs in software. This comes in very handy when we do not have access to an oscilloscope. Future teams with access to this peripheral would be able to configure the logic analyzer through the computer interface of the microcontroller and will be subsequently be able to view different graphs and results.

**9.0 CONCLUSIONS**

The goal of our project was to develop a three-phase converter based on a variable frequency drive powered by a solar panel aimed at promoting green energy and grid independence. We developed and implemented the designs of four key modules: H-Bridge module, PWM, User-Interface, and tracking algorithm. Our project consisted of a DC to three-phase AC converter H-bridge inverting circuit that is controlled by a microcontroller. The control is based on the feedback from sensors placed near the solar panel and the motor to ensure maximum power transfer from the DC input to the motor output. One of our achievements was successfully generating and testing three sine-modulated signals phased off at 120 degrees to each other using the output ports of the microcontroller. We also succeeded in implementing and testing a keypad and LCD based user-interface system allowing users to enter motor and solar panel characteristics. Important parts of the tracking algorithm, such as tracking motor speed and solar panel intensity, followed by the successful integration and testing of the PWM and user-interface as well as the PWM and the tracking algorithm modules, were also completed. We identified problems within our design, when we burned out our microcontroller while testing PWM compatibility with the H-Bridge circuit. We found that our PWM generator ports and H-Bridge circuit were unreliable and even strong isolation circuitry could not prevent high current flow through signal output ports. We also found that we could not use common shunt resistor based current sensing circuits to acquire motor current output. The burnout of the microcontroller wiped out some of the important progress made in our system, and led to problems in completing, integrating, and testing the tracking algorithm with the rest of the system. Looking back at the project, we recommend the use of better protection circuitry from high current, use better modeling tools like LabView to simulate functionality before physical implementation of circuits, and conducting comprehensive research of important hardware components before applying them to the final design. We also recommend starting early and consulting relevant professors and resources before planning allocation of time on specific implementations.The PCB layout designs and code documentation in Appendix B and C, respectively.

**REFERENCES**

[1] W. Mack Grady, “Power Electronics, Solar Power, I-V Characteristics”, Internet:

<http://users.ece.utexas.edu/~grady/_4_EE362L_Solar.pdf> . October 7 2008, [Accessed: March 29, 2010]

[2]  “STM32F1003 data sheet” [Online] <http://www.st.com/mcu/download2.php?file=13587.pdf&info=Datasheet%20STM32F103x%20from%20Search&url=http://www.st.com/stonline/products/literature/ds/13587.pdf>. [Accessed: March 28, 2010]

[3] N. Mohan, T. M. Undeland and W. P. Robbins, “Power Electronics: Converters, Applications, and Design”, Third Edition, John Wiley & Sons, Inc., 2003.

[4] “QRB1134 reflective object sensor data sheet” [Online] <http://www.fairchildsemi.com/ds/QR/QRB1134.pdf>. [Accessed: March 28, 2010]

[5] “Solar Panel” [Online]. Available: <http://www.homedepot.com/catalog/pdfImages/0f/0f64d9e4-dd66-4cd7-9fa0-4352fd8b67dd.pdf> .[Accessed: March 31 2010].

[6] “ NPN Transistors” [Online]. Available: <http://www.radioshack.com/product/index.jsp?productId=2062586&CAWELAID=107590986>. [Accessed: March 31 2010]

**APPENDIX A – SCHEMATIC DIAGRAMS**

**APPENDIX A – SCHEMATIC DIAGRAMS**

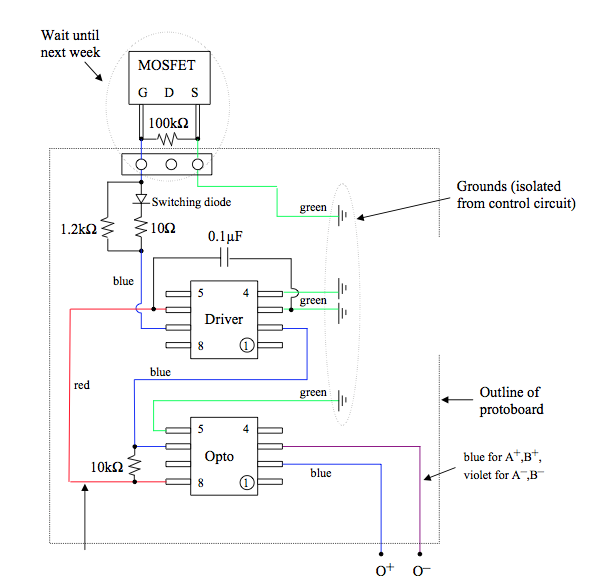
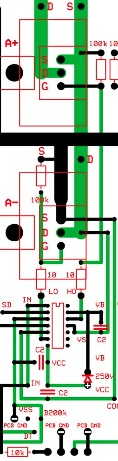


Figure 1: H-Bridge using Inverting/Non-Inverting Chip and Optocoupler5

**APPENDIX A – SCHEMATIC DIAGRAMS**

*Fig 2 – H-Bridge using IC chip, and for a single phase*

**APPENDIX A – SCHEMATIC DIAGRAMS**

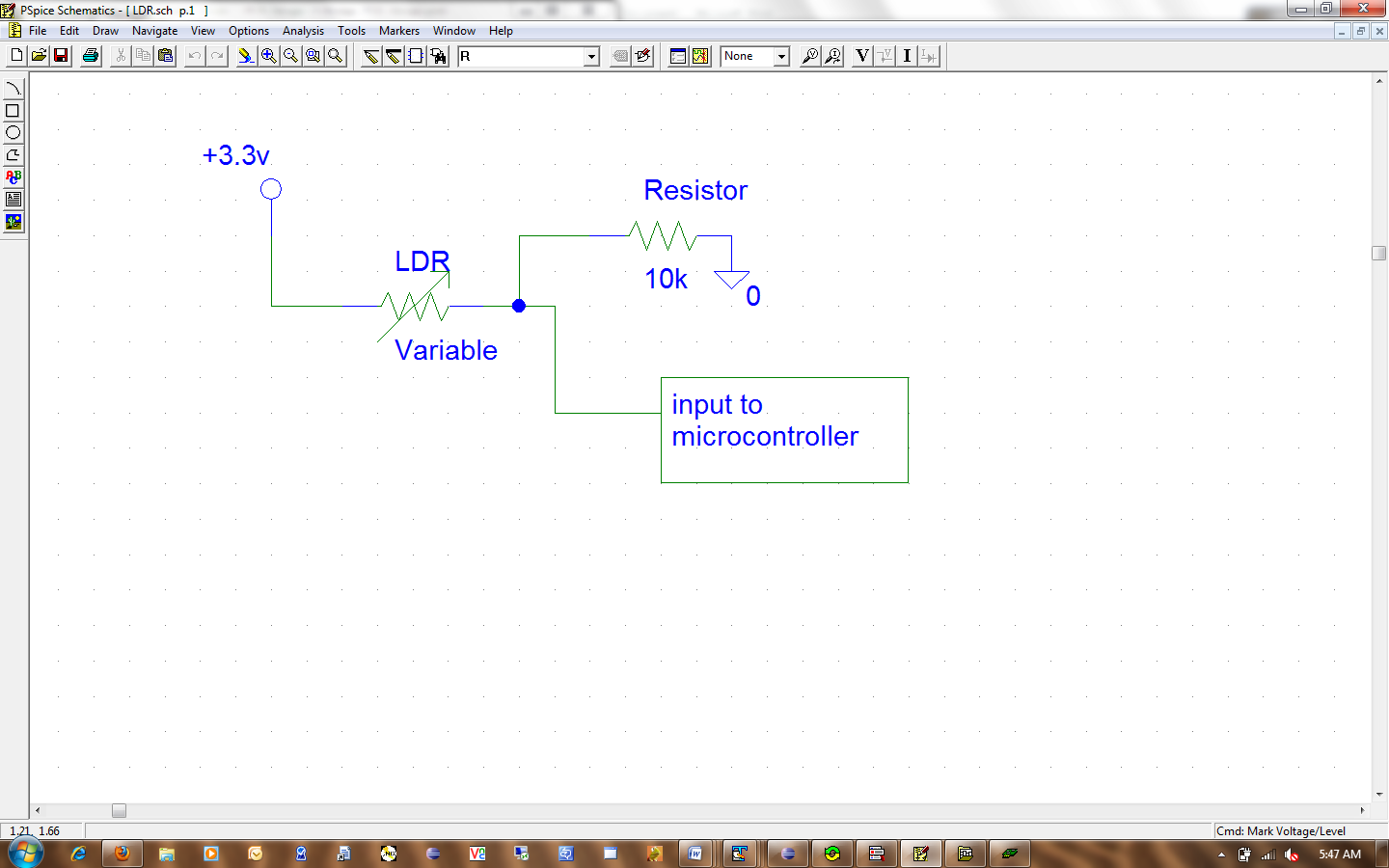


Figure 5: Schematic of our Light Detecting Sensor.

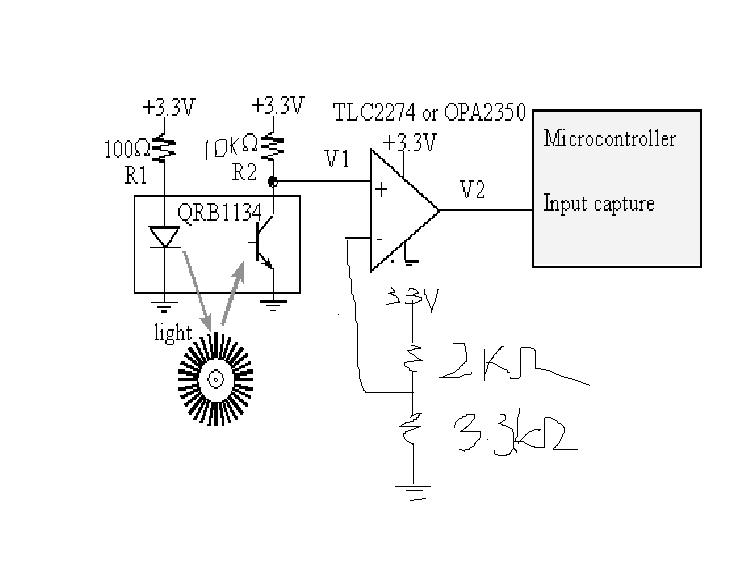
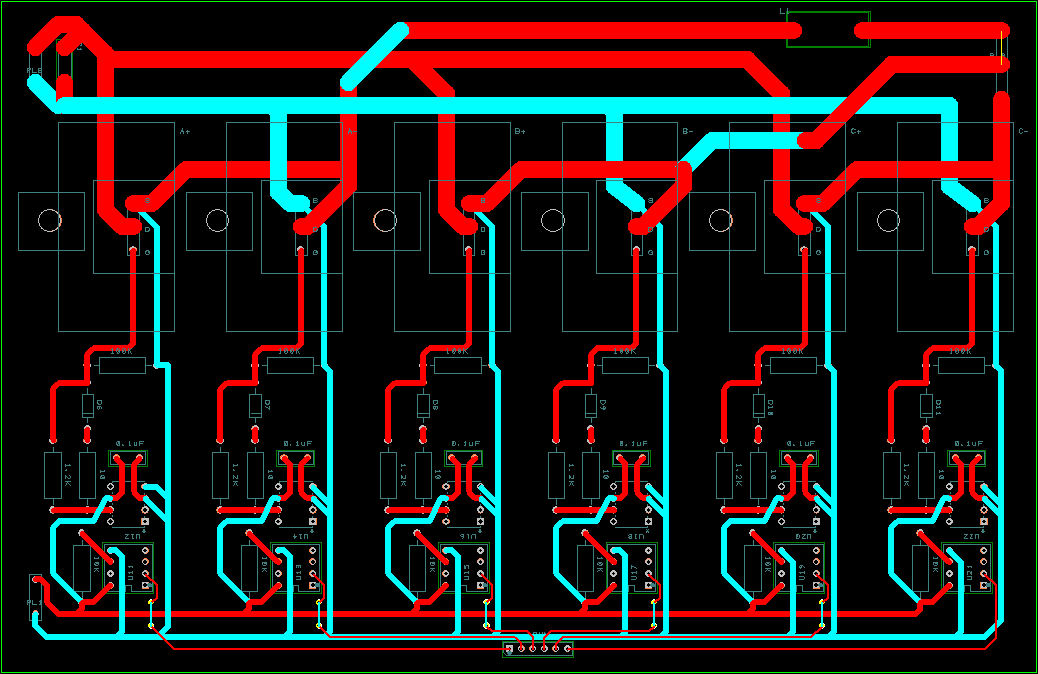


Figure 6: Schematics for Tachometer Sensor.

**APPENDIX B – PCB LAYOUTS**

**APPENDIX B – PCB LAYOUTS**



**Fig3 . Possible PCB design of our H-Bridge circuit.**

**APPENDIX B – PCB LAYOUTS**

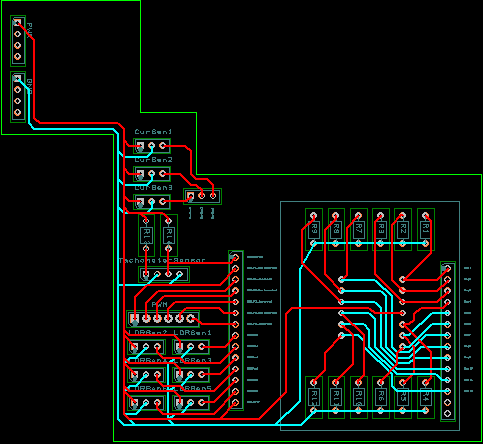
****

Figure 4: Possible PCB design of our User Interface/ Keypad

**APPENDIX C – CODE DOCUMENTATION**

# Code Documentation

## File List

**ADC.h**  C-3

**ADC.c**  C-4

**IRQ.c**  C-5

**Keypad.c**  C-7

**Keypad.h**  C-8

**LCD.c**  C-9

**LCD.h**  C-12

**main.c**  C-13

**PWM.c**  C-20

**PWM.h**  C-22

**Tracking.c**  C-26

**Tracking.h**  C-28

## ADC.h

### Enumerations

* enum **ADC\_Channel** { **PA0** = 0, **PA1** = 1, **PA2** = 2, **PA3** = 3, **PA4** = 4, **PA5** = 5, **PA6** = 6 }

### Functions

* void ADC\_SoftwareInit (void)
* unsigned short int **ADC\_getValue** (int chan)

### Enumeration Type Documentation

#### enum ADC\_Channel

**Enumerator:**

***PC0***

***PC1***

***PC2***

***PC3***

***PC4***

***PC5***

***PC6***

**ADC.C**

#include <stdio.h>

#include <stm32f10x\_lib.h>

#include "STM32\_Init.h"

#include "ADC.h"

### Functions

* void ADC\_SoftwareInit ()

ADC initialization of ports

* unsigned short int **ADC\_getValue** (int chan)

Get ADC converted value from ports

### Function Documentation

#### unsigned short int ADC\_getValue (int chan)

*Inputs: Channel*

*Outputs: ADC value*

#### void ADC\_SoftwareInit (void)

*Inputs: None*

*Outputs: None*

## IRQ.c

#include <stm32f10x\_lib.h>

### Functions

* void **updateFrequency** (unsigned short frequency)

*Update frequency value*

* void EXTI0\_IRQHandler (void)

*Handle external port hardware interrupts*

* void TIM3\_IRQHandler (void)

*Handle timer interrupts*

* void **MC\_IRQHandler** (void)

*Handle motor controller interrupts*

### Variables

* unsigned short **sinetable** [255]

*Array to store sinetable*

* unsigned short **sine\_offset1** = 0

*Sine wave offset for signal 1*

* unsigned short **sine\_offset2** = 85

*Sine wave offset for signal 2*

* unsigned short **sine\_offset3** = 170

*Sine wave offset for signal 3*

* unsigned short **period** = 0

*Period value*

### Function Documentation

#### void EXTI0\_IRQHandler (void)

*Inputs: None*

*Outputs: None*

#### void TIM1\_UP\_IRQHandler (void)

*Inputs: None*

*Outputs: None*

#### void TIM3\_IRQHandler (void)

*Inputs: None*

*Outputs: None*

#### void updateFrequency (unsigned short frequency)

*Inputs: Frequency Value*

*Outputs: None*

### Variable Documentation

#### unsigned short period = 0

#### unsigned short sine\_offset1 = 0

#### unsigned short sine\_offset2 = 85

#### unsigned short sine\_offset3 = 170

#### unsigned short sinetable[255]

**Initial value:** {128, 131, 134, 137, 141, 144, 147, 150, 153, 156, 159, 162, 165, 168, 171, 174, 177,

180, 183, 186, 189, 191, 194, 197, 199, 202, 205, 207, 209, 212, 214, 217, 219, 221, 223, 225, 227, 229, 231, 233, 235,

236, 238, 240, 241, 243, 244, 245, 246, 248, 249, 250, 251, 252, 252, 253, 254, 254, 255, 255, 255, 256, 256, 256, 256,

256, 256, 256, 255, 255, 254, 254, 253, 253, 252, 251, 250, 249, 248, 247, 246, 245, 243, 242, 240, 239, 237, 236, 234,

232, 230, 228, 226, 224, 222, 220, 218, 215, 213, 211, 208, 206, 203, 201, 198, 195, 193, 190, 187, 184, 181, 179, 176,

173, 170, 167, 164, 161, 158, 155, 152, 148, 145, 142, 139, 136, 133, 130, 126, 123, 120, 117, 114, 111, 108, 104, 101,

98, 95, 92, 89, 86, 83, 80, 77, 75, 72, 69, 66, 63, 61, 58, 55, 53, 50, 48, 45, 43, 41, 38, 36, 34, 32, 30, 28, 26,

24, 22, 20, 19, 17, 16, 14, 13, 11, 10, 9, 8, 7, 6, 5, 4, 3, 3, 2, 2, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 2, 2, 3,

4, 4, 5, 6, 7, 8, 10, 11, 12, 13, 15, 16, 18, 20, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 42, 44, 47, 49, 51, 54,

57, 59, 62, 65, 67, 70, 73, 76, 79, 82, 85, 88, 91, 94, 97, 100, 103, 106, 109, 112, 115, 119, 122, 125}

## Keypad.c

#include <stm32f10x\_lib.h>

#include <RTL.h>

#include "keypad.h"

#include "STM32\_Init.h"

### Functions

* void key\_init(void)

*Initialize keypad ports*

* unsigned short **keyScan** (void)

*Read keypad inputs*

### Variables

* unsigned short **Key\_value**
* unsigned short **Key\_value2**
* unsigned char **key1** = 0
* unsigned char **key2** = 0
* unsigned char **key3** = 0

### Function Documentation

### void key\_init(void)

#### Inputs: none

*Outputs: none*

#### unsigned short keyScan (void)

#### Inputs: none

*Outputs: key value*

### Variable Documentation

#### unsigned char key1 = 0

#### unsigned char key2 = 0

#### unsigned char key3 = 0

#### unsigned short Key\_value

#### unsigned short Key\_value2

#### 

## Keypad.h

### Defines

* #define **KEY\_4**  0x0020
* #define **KEY\_5**  0x0040
* #define **KEY\_6**  0x0080
* #define **KEY\_7**  0x0008
* #define **KEY\_8**  0x0010
* #define **KEY\_9**  0x0020
* #define **KEY\_STAR**  0x0080
* #define **KEY\_HASH**  0x0080
* #define **KEY\_1**  0x0040
* #define **KEY\_2**  0x0004
* #define **KEY\_3**  0x0010
* #define **KEY\_0**  0x0020
* #define **KEY\_ALL1**  0x00E0
* #define **KEY\_ALL2**  0x00F8
* #define **KEY\_ALL3**  0x00F0

### Functions

* void **Key\_Init** (void)
* void **Key\_Init2** (void)
* unsigned short **keyScan** (void)

#### 

## LCD.c

#include <RTL.h>

#include <91x\_lib.H>

#include "LCD.h"

### Defines

* #define **LCD\_DATA**  GPIO8->DR[0xFF<<2]
* #define **LCD\_E**  GPIO9->DR[0x01<<2]
* #define **LCD\_RW**  GPIO9->DR[0x02<<2]
* #define **LCD\_RS**  GPIO9->DR[0x04<<2]
* #define **LCD\_CTRL**  GPIO9->DR[0x07<<2]
* #define **E\_ENA**  0x01
* #define **E\_DIS**  0x00
* #define **RW\_READ**  0x02
* #define **RW\_WRITE**  0x00
* #define **RS\_DATA**  0x04
* #define **RS\_INST**  0x00

### Functions

* void **LCD\_init** (void)
* void **LCD\_load** (U8 \*fp, U32 cnt)
* void **LCD\_gotoxy** (U32 x, U32 y)
* void **LCD\_cls** (void)
* void **LCD\_cur\_off** (void)
* void **LCD\_on** (void)
* void **LCD\_outDec** (short int number)
* void **LCD\_putc** (U8 c)
* void **LCD\_puts** (U8 \*sp)
* void **LCD\_bargraph** (U32 val, U32 size)
* void **Fixed\_Fix2Str** (unsigned short fixedNumber)

### Define Documentation

#### #define E\_DIS  0x00

#### #define E\_ENA  0x01

#### #define LCD\_CTRL  GPIO9->DR[0x07<<2]

#### #define LCD\_DATA  GPIO8->DR[0xFF<<2]

#### #define LCD\_E  GPIO9->DR[0x01<<2]

#### #define LCD\_RS  GPIO9->DR[0x04<<2]

#### #define LCD\_RW  GPIO9->DR[0x02<<2]

#### #define RS\_DATA  0x04

#### #define RS\_INST  0x00

#### #define RW\_READ  0x02

#### #define RW\_WRITE  0x00

### Function Documentation

#### void Fixed\_Fix2Str (unsigned short fixedNumber)

#### Inputs: fixedNumber

*Outputs: key value*

#### void LCD\_cls (void)

#### Inputs: none

*Outputs: none*

#### void LCD\_cur\_off (void)

#### Inputs: none

*Outputs: none*

#### void LCD\_gotoxy (U32 x, U32 y)

#### Inputs: x, y coordinates

*Outputs: none*

#### void LCD\_init (void)

#### Inputs: none

*Outputs: none*

#### void LCD\_load (U8 \* fp, U32 cnt)

#### Inputs: none

*Outputs: none*

#### void LCD\_on (void)

#### Inputs: none

*Outputs: none*

#### void LCD\_outDec (short int number)

#### Inputs: number to output to LCD

*Outputs: none*

#### void LCD\_putc (U8 c)

#### Inputs: character to be put to LCD

*Outputs: none*

#### void LCD\_puts (U8 \* sp)

#### Inputs: String to be put to the LCD

*Outputs: none*

## LCD.h

### Functions

* void **lcd\_init** (void)
* void **lcd\_clear** (void)
* void **LCD\_putchar** (char c)
* void **set\_cursor** (int column, int line)
* void **lcd\_print** (char \*string)
* void **lcd\_bargraph** (int value, int size)
* void **lcd\_bargraphXY** (int pos\_x, int pos\_y, int value)
* void **LCD\_outDec** (short int number)
* unsigned short **Fixed\_Str2Fix** (char string[])
* void **Fixed\_Fix2Str** (unsigned short fixedNumber)

## main.c

#include <stdio.h>

#include <ctype.h>

#include <string.h>

#include <RTL.h>

#include <stm32f10x\_lib.h>

#include "STM32\_Init.h"

#include "LCD.h"

#include "KeyPad.h"

#include "Tracking.h"

#include "ADC.h"

### Defines

* #define **ESC**  0x1B

*ESCAPE character code.*

* #define **MAX\_CURRENT**  1000

*Maximum Current Allowed.*

* #define **S3**  0x2000

*PC13: S3.*

* #define **S2**  0x0001

*PA0 : S2.*

* #define UNBOUNCE\_CNT  5

*unbounce the keys*

### Functions

* void **getline** (char \*, int)

*external function: getline for Serial interface*

* void **serial\_init** (void)

*external function: Serial Initialization*

* int **GetKey** (void)

*external function: Get Input Key*

* \_\_task void **init** (void)

*Task 1 'init': Initialize.*

* \_\_task void **get\_escape** (void)

*Task 7 'get\_escape': check if ESC (escape character) was entered.*

* \_\_task void **command** (void)

*Task 8 'command': command processor.*

* \_\_task void **lcd** (void)

*Task 2 'lcd': LCD/Keypad interface to get user input.*

* \_\_task void **lcd2** (void)

*Task 9 'LCD2': Display measurements when Motor is running.*

* \_\_task void **photo** (void)

*Task 4 'photo\_sensor': Read the photo intensity value input.*

* \_\_task void **motor** (void)

*Task 3 'Motor': Reads input to start PWM generation, current sensing and power calculation.*

* int **S2Pressed** (void)

*S2Pressed: check if S3 is pressed (unbounced).*

* int **S3Pressed** (void)

*S3Pressed: check if S3 is pressed (unbounced).*

* int **main** (void)

*Main: Initialize and start RTX Kernel.*

* \_\_task void **current\_sensor** (void)

*Task 5 'current\_sensor': Read the current value input using the ADC port.*

* \_\_task void **tracker** (void)

*Task 6 'tracker': Calculates the power.*

### Variables

* const char **menu** []

*Serial Menu.*

* OS\_TID t\_command

*assigned task id of task: command*

* OS\_TID **t\_getesc**

*assigned task id of task: get\_esc*

* OS\_TID **t\_lcd**

*assigned task id of task: lcd*

* OS\_TID **t\_tach**

*assigned task id of task: tachometer*

* OS\_TID **t\_photo**

*assigned task id of task: photo sensor*

* OS\_TID t\_KeyPad

*assigned task id of task: keypad*

* OS\_TID **t\_motor**

*assigned task id of task: motor*

* OS\_TID t\_tracker

*assigned task id of task: tracker*

* OS\_TID **t\_lcd2**

*assigned task id of task: lcd2*

* char **PWM\_state** = 0

*PWM ON/OFF.*

* short **AD\_current** = 330

*Last AD current value read in interrupt.*

* short AD\_photo

*Last AD photo value read in interrupt.*

* short **tach** = 0

*Tachometer ADC output.*

* char **cmdline** [16]

*storage for command input line*

* char **vinput** [16]

*storage for voltage input line*

* char **finput** [16]

*storage for frequency input line*

* char **linput** [16]

*storage for light intensity input line*

* char **lvinput** [16]

*storage for light voltage input line*

* BIT escape

*flag: mark ESCAPE character entered*

* int **key** = 100

*Variable for key.*

* int current\_freq = 0

*Current Frequency.*

* int current\_volt = 0

*Current Voltage.*

* int **X2** = 0

*X^2 component.*

* int **X** = 0

*X component.*

* int constant

*Constant component.*

* int intensity = 0

*Intensity.*

### Define Documentation

#### #define ESC  0x1B

ESCAPE character code.

#### #define MAX\_CURRENT  1000

Maximum Current Allowed.

#### #define S2  0x0001

PA0 : S2.

#### #define S3  0x2000

PC13: S3.

#### #define UNBOUNCE\_CNT  5

unbounce the keys

### Function Documentation

#### \_\_task void command (void)

Task 8 'command': command processor.

#### \_\_task void current\_sensor (void)

Task 5 'current\_sensor': Read the current value input using the ADC port.

#### \_\_task void get\_escape (void)

Task 7 'get\_escape': check if ESC (escape character) was entered.

#### int GetKey (void)

external function: Get Input Key

#### void getline (char \*, int)

external function: getline for Serial interface

#### \_\_task void init (void)

Task 1 'init': Initialize.

#### \_\_task void lcd (void)

Task 2 'lcd': LCD/Keypad interface to get user input.

#### \_\_task void lcd2 (void)

Task 9 'LCD2': Display measurements when Motor is running.

#### int main (void)

Main: Initialize and start RTX Kernel.

#### \_\_task void motor (void)

Task 3 'Motor': Reads input to start PWM generation, current sensing and power calculation.

#### \_\_task void photo (void)

Task 4 'photo\_sensor': Read the photo intensity value input.

#### int S2Pressed (void)

S2Pressed: check if S3 is pressed (unbounced).

#### int S3Pressed (void)

S3Pressed: check if S3 is pressed (unbounced).

#### void serial\_init (void)

external function: Serial Initialization

#### \_\_task void tracker (void)

Task 6 'tracker': Calculates the power.

### Variable Documentation

#### short AD\_current = 330

Last AD current value read in interrupt.

#### short AD\_photo

Last AD photo value read in interrupt.

#### char cmdline[16]

storage for command input line

#### int constant

**Initial value:** 0

int power = 0

Constant component.

Power Value

#### int current\_freq = 0

Current Frequency.

#### int current\_volt = 0

Current Voltage.

#### BIT escape

flag: mark ESCAPE character entered

#### char finput[16]

storage for frequency input line

#### int intensity = 0

Intensity.

#### int key = 100

Variable for key.

#### char linput[16]

storage for light intensity input line

#### char lvinput[16]

storage for light voltage input line

#### const char menu[] = "+----------+-------------+------------------------------------+\n"

Serial Menu.

#### char PWM\_state = 0

PWM ON/OFF.

#### OS\_TID t\_command

assigned task id of task: command

#### OS\_TID t\_getesc

assigned task id of task: get\_esc

#### OS\_TID t\_KeyPad

assigned task id of task: keypad

#### OS\_TID t\_lcd

assigned task id of task: lcd

#### OS\_TID t\_lcd2

assigned task id of task: lcd2

#### OS\_TID t\_motor

assigned task id of task: motor

#### OS\_TID t\_photo

assigned task id of task: photo sensor

#### OS\_TID t\_tach

assigned task id of task: tachometer

#### OS\_TID t\_tracker

assigned task id of task: tracker

#### short tach = 0

Tachometer ADC output.

#### char vinput[16]

storage for voltage input line

#### int X = 0

X component.

#### int X2 = 0

X^2 component.

## PWM.c

#include "PWM.H"

#include "91x\_gpio.h"

#include "91x\_mc.h"

### Functions

* unsigned short **getPWMfrequency** (void)

Returns the current frequency of operation

* void **setPWMfrequency** (unsigned short freq)

Sets the current frequency of operation

* void **PWM\_start** (void)

Start PWM motor output

* void **PWM\_stop** (void)

Stop PWM motor output

* void **PWM\_init** ()

Initialize PWM initialization

### Variables

* unsigned short **Delay**
* unsigned short **PWMfrequency**
* GPIO\_InitTypeDef \* **GPIO\_InitStruct1**
* MC\_InitTypeDef \* **MC\_InitStruct**

### Function Documentation

#### unsigned short getPWMfrequency (void)

#### Inputs: none

*Outputs: Current Frequency*

#### void PWM\_init (void)

#### Inputs: none

*Outputs: none*

#### void PWM\_start (void)

#### Inputs: none

*Outputs: none*

#### void PWM\_stop (void)

#### Inputs: none

*Outputs: none*

#### void setPWMfrequency (unsigned short freq)

#### Inputs: none

*Outputs: none*

### Variable Documentation

#### unsigned short Delay

#### GPIO\_InitTypeDef\* GPIO\_InitStruct1

#### MC\_InitTypeDef\* MC\_InitStruct

#### unsigned short PWMfrequency

#### 

## PWM.h

### Defines

* #define **AMPLITUDE**  1
* #define **MC\_ODCS\_Set**  0x0001
* #define **MC\_ODCS\_Reset**  0x00FE
* #define **MC\_CMS\_Set**  0x0004
* #define **MC\_CMS\_Reset**  0x00FB
* #define **MC\_CPC\_Set**  0x0008
* #define **MC\_CTC\_Set**  0x0010
* #define **MC\_PCE\_Set**  0x0020
* #define **MC\_PCE\_Reset**  0xFFDF
* #define **MC\_TCE\_Set**  0x0040
* #define **MC\_TCE\_Reset**  0x00BF
* #define **MC\_DTE\_Set**  0x0080
* #define **MC\_DTE\_Reset**  0x007F
* #define **MC\_TCB\_Set**  0x0004
* #define **MC\_TCB\_Reset**  0x00FB
* #define **MC\_STC\_Set**  0x0008
* #define **MC\_TES\_Set**  0x0010
* #define **MC\_TES\_Reset**  0x00EF
* #define **MC\_CCPT\_Set**  0x0020
* #define **MC\_CCPT\_Reset**  0x005F
* #define **MC\_DISEST\_Set**  0x0040
* #define **MC\_DISEST\_Reset**  0x003F
* #define **MC\_DTS\_Set**  0x0001
* #define **MC\_DTS\_Reset**  0x00FE
* #define **MC\_SDT\_Set**  0x0002
* #define **MC\_C0SE\_Set**  0x0004
* #define **MC\_C0SE\_Reset**  0x00FB
* #define **MC\_CUSE\_Set**  0x0008
* #define **MC\_CUSE\_Reset**  0x00F7
* #define **MC\_CVSE\_Set**  0x0010
* #define **MC\_CVSE\_Reset**  0x00EF
* #define **MC\_CWSE\_Set**  0x0020
* #define **MC\_CWSE\_Reset**  0x00D0
* #define **MC\_RSE\_Set**  0x0040
* #define **MC\_RSE\_Reset**  0x00BF
* #define **MC\_GPI\_Set**  0x0080
* #define **MC\_GPI\_Reset**  0x007F
* #define **MC\_PUH\_Set**  0x0020
* #define **MC\_PUH\_Reset**  0x005F
* #define **MC\_PUL\_Set**  0x0010
* #define **MC\_PUL\_Reset**  0x006F
* #define **MC\_PVH\_Set**  0x0008
* #define **MC\_PVH\_Reset**  0x0077
* #define **MC\_PVL\_Set**  0x0004
* #define **MC\_PVL\_Reset**  0x007B
* #define **MC\_PWH\_Set**  0x0002
* #define **MC\_PWH\_Reset**  0x007D
* #define **MC\_PWL\_Set**  0x0001
* #define **MC\_PWL\_Reset**  0x007E
* #define **MC\_ODS\_Set**  0x0040
* #define **MC\_ODS\_Reset**  0xFF3F
* #define **MC\_ESC\_Clear**  0x4321
* #define **MC\_PCR1\_TIN\_MASK**  0xFFFC
* #define **MC\_OPR\_Mask**  0x0040
* #define **MC\_UDCS\_Mask**  0x0002

### Functions

* unsigned short **getPWMfrequency** (void)
* void **setPWMfrequency** (unsigned short freq)
* void **PWM\_start** (void)
* void **PWM\_stop** (void)
* void **PWM\_init** (void)

### Define Documentation

#### #define AMPLITUDE  1

#### #define MC\_C0SE\_Reset  0x00FB

#### #define MC\_C0SE\_Set  0x0004

#### #define MC\_CCPT\_Reset  0x005F

#### #define MC\_CCPT\_Set  0x0020

#### #define MC\_CMS\_Reset  0x00FB

#### #define MC\_CMS\_Set  0x0004

#### #define MC\_CPC\_Set  0x0008

#### #define MC\_CTC\_Set  0x0010

#### #define MC\_CUSE\_Reset  0x00F7

#### #define MC\_CUSE\_Set  0x0008

#### #define MC\_CVSE\_Reset  0x00EF

#### #define MC\_CVSE\_Set  0x0010

#### #define MC\_CWSE\_Reset  0x00D0

#### #define MC\_CWSE\_Set  0x0020

#### #define MC\_DISEST\_Reset  0x003F

#### #define MC\_DISEST\_Set  0x0040

#### #define MC\_DTE\_Reset  0x007F

#### #define MC\_DTE\_Set  0x0080

#### #define MC\_DTS\_Reset  0x00FE

#### #define MC\_DTS\_Set  0x0001

#### #define MC\_ESC\_Clear  0x4321

#### #define MC\_GPI\_Reset  0x007F

#### #define MC\_GPI\_Set  0x0080

#### #define MC\_ODCS\_Reset  0x00FE

#### #define MC\_ODCS\_Set  0x0001

#### #define MC\_ODS\_Reset  0xFF3F

#### #define MC\_ODS\_Set  0x0040

#### #define MC\_OPR\_Mask  0x0040

#### #define MC\_PCE\_Reset  0xFFDF

#### #define MC\_PCE\_Set  0x0020

#### #define MC\_PCR1\_TIN\_MASK  0xFFFC

#### #define MC\_PUH\_Reset  0x005F

#### #define MC\_PUH\_Set  0x0020

#### #define MC\_PUL\_Reset  0x006F

#### #define MC\_PUL\_Set  0x0010

#### #define MC\_PVH\_Reset  0x0077

#### #define MC\_PVH\_Set  0x0008

#### #define MC\_PVL\_Reset  0x007B

#### #define MC\_PVL\_Set  0x0004

#### #define MC\_PWH\_Reset  0x007D

#### #define MC\_PWH\_Set  0x0002

#### #define MC\_PWL\_Reset  0x007E

#### #define MC\_PWL\_Set  0x0001

#### #define MC\_RSE\_Reset  0x00BF

#### #define MC\_RSE\_Set  0x0040

#### #define MC\_SDT\_Set  0x0002

#### #define MC\_STC\_Set  0x0008

#### #define MC\_TCB\_Reset  0x00FB

#### #define MC\_TCB\_Set  0x0004

#### #define MC\_TCE\_Reset  0x00BF

#### #define MC\_TCE\_Set  0x0040

#### #define MC\_TES\_Reset  0x00EF

#### #define MC\_TES\_Set  0x0010

#### #define MC\_UDCS\_Mask  0x0002

## Tracking.c

#include <stdio.h>

#include "Tracking.h"

### Defines

* #define **PHOTO\_TOLERANCE**  0.02
* #define **POWER\_TOLERANCE**  0.05

### Functions

* void **set\_VFTable** (char **vinput**[16], char **linput**[16])
* void **voltage\_calc** (unsigned short equation)
* void **RPM\_calc** (unsigned short current\_frequency)
* void **power\_calculate** (unsigned short current\_frequency)
* void **get\_voltage** ()
* void **get\_current** ()
* void **get\_RPM** ()

### Variables

* char \* **vpoints**
* char \* **lpoints**

### Define Documentation

#### #define PHOTO\_TOLERANCE  0.02

#### #define POWER\_TOLERANCE  0.05

### Function Documentation

#### void power\_calculate (unsigned short current\_frequency)

*Inputs: current PWM frequency*

*Outputs: None*

#### void RPM\_calc (unsigned short current\_frequency)

*Inputs: current PWM period*

*Outputs: None*

#### void set\_VFTable (char vinput[16], char finput[16])

*Inputs: Voltage and frequency input table*

*Outputs: None*

#### void set\_LIVTable (char vinput[16], char linput[16])

*Inputs: Voltage and light intensity input table*

*Outputs: None*

#### void voltage\_calc (unsigned short equation)

*Inputs: equation parameters*

*Outputs: None*

### Variable Documentation

#### char\* lpoints

#### char\* vpoints

#### 

## Tracking.h

### Defines

* #define **MIN\_FREQUENCY**  20

### Functions

* void **set\_VFTable** (char **vinput**[16], char **linput**[16])
* void **voltage\_calc** (unsigned short equation)
* void **RPM\_calc** (unsigned short current\_frequency)
* void **power\_calculate** (unsigned short current\_frequency)
* void **get\_voltage** (void)
* void **get\_current** (void)
* void **get\_RPM** (void)

### Define Documentation

#### #define MIN\_FREQUENCY  20