TEXAS INSTRUMENTS

Stepper Motor Control

With the DRV8824EVM, MSP430F1612 CC and Stellaris LM3S9B96 CC

4/25/2011

Leslie Thomas
Patrick O'Hara
Thomas Volinski
Kole Reece

Facilitator: Dr. Bingsen Wang
Sponsor: Mr. Tim Adcock

Executive Summary

Texas Instruments has developed an evaluation module for their newest motor driver the DRV8824. This evaluation module comes with a MSP430F1612 on board to control the motor driver. The DRV8824 manages the pulse-width modulation (PWM) while allowing for the MSP430 to control the DRV8824.

Texas Instruments has moved to a more modular design for their motor drivers. They are providing the analog DRV models with sockets for their controlCARDs. These controlCARDs contain their popular microcontrollers, and allow for motor control with interchangeable microcontrollers.

To fit the DRV8824EVM into this modular design the board will be partitioned into two parts, a motor driving board and a controlCARD. The motor driving board will have a DIMM socket that will be able to accept multiple controlCARDs that can all have different microcontrollers onboard. The controlCARD will contain the MSP430 microcontroller with a matching pin out. The multiple microcontrollers on the controlCARDs will be able to be programmed to control the DRV8824 motor driver, and provide a modular approach to motor control.

Acknowledgments

Throughout the semester there are many people who have aided in the development of our motor controlCARD and evaluation module. We would like to sincerely thank:

- Professor Michael Shanblatt: for covering our bill when we went way over budget.
- Professor Bingsen Wang: for giving our team the guidance needed and making it to every Thursday meeting.
- Mrs. Roxanne Peacock: for assisting in purchasing our parts, and communicating effectively with us throughout the semester.
- Mr. Brian Wright and Mr. Gregg Mulder: for helping us understand the necessary files for PCB fabrication and for assembling our EVM for us.
- Mr. Greg Motter: for showing us how to complete a FAST diagram and construct a house of quality.
- Mr. Jacob Scott: for assembling our most difficult PCB and replacing parts when they burnt out.
- Mr. Tim Adcock: for giving our design team the opportunity to work with Texas Instruments, providing us with all the hardware and software tools, and keeping in constant contact throughout the semester.

Table of Contents

Chapter 1

Introduction

There are many motors on the market today such as the brushed DC, brushless DC, AC induction, and stepper motors. They are each used for certain applications in many devices today, including cars, air conditioners and microwaves. Before a motor can be implemented into a project it is necessary to determine the speeds the motor will operate at and if the amount of torque will be sufficient for the application. This is why it is a smart idea to build a test circuit or purchase a developmental kit before implementing a motor in a project. Many motors require a precise driving system for the control of a motor's speed and direction. It also needs special hardware and precisely timed signals to be sent to the circuit attached to the motor. The simplest solution is to purchase a motor control integrated circuit (IC) and control that IC with a microcontroller (MCU).

Design Team 6's (DT6) project focuses specifically on the control of a stepper motor with a DRV8824, which is a motor control IC created by Texas Instruments (TI). DT6 was given this IC on an evaluation module called the DRV8824EVM, seen in Figure 1. This evaluation module allows the customer to connect a two phase stepper motor to the board and test the motor with all the controls the DRV8824 has to offer through a Windows General User Interface (GUI). The evaluation module controls the DRV8824 with an MSP430F1612 MCU, this MCU is the component that controls the DRV8824 and communicates with the Windows GUI.

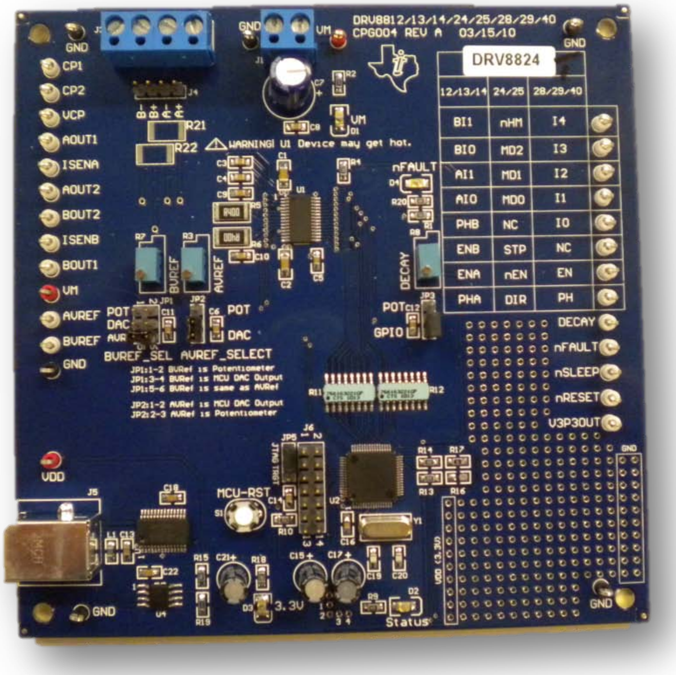


Figure 1. DRV8824EVM

Since not all customers want to use MSP430F1612 MCU in their projects TI faces a problem with appealing to a large market. TI offers a large assortment of MCUs and 3 different families; they are the MSP430, Stellaris, and C2000 families. TI often puts their most popular MCUs on dual inline memory modules (DIMM) cards called controlCARDs. TI has many developmental kits that have slots in them so these controlCARDs can be used. TI has designated DT6 three tasks to make the DRV8824EVM more modular.

DT6 must replace the MCU on the evaluation module with a DIMM female connector, this will allow for the customer to insert controlCARD into the DRV8824EVM with the MCU that matches the customer's needs. The second task is to create a MSP430F1612 controlCARD that will work with the newly modified DRV8824EVM. The third task is to port the code from the MSP430F1612 controlCARD to the Stellaris LM3S9B96 controlCARD and demonstrate its working functions with the Windows GUI.

Background

Stepper motors are a very unique type of motor; they are one of the only motors that need no feedback to know what the motor will end up doing. One of the reasons for this is because stepper motors are among the only motors to have axial flux. This means the length of the motor is polarized instead of the face as seen in Figure 2.

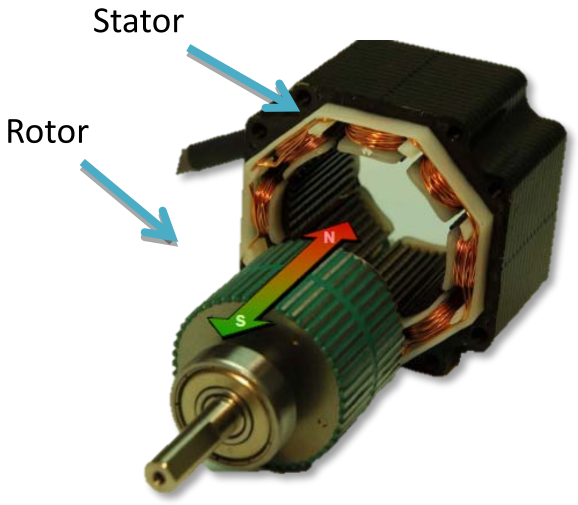


Figure 2. Stepper Motor showing axial flux

The motor is moved by current excited through the stator at its phases. Figure 3 shows how the stator would be configure for a 2-phase bi-polar motor. There is an A and B phase and for each phase there is an opposite phase at a 90 degree rotation of the starting point. To turn a motor, current would first be sent through Phase A this would attract the teeth of the rotor to the teeth on the stator and they would align, while A’ would push the teeth away. Next phase B would turn on and the teeth would move from aligning with A to aligning with B. Then A would have current sent in the opposite direction turning A’ on which would move the teeth of the rotor to align with A’ and so forth. This is how a full step would be implemented. Figure 4 further illustrates the alignment of the teeth.

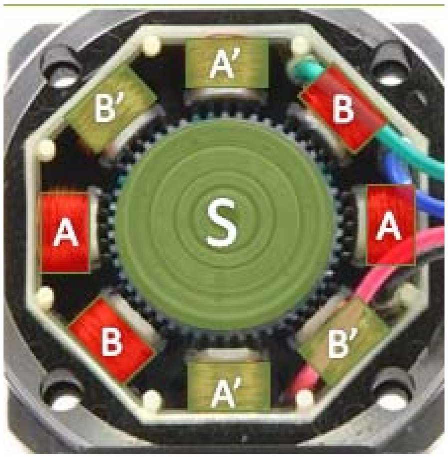


Figure 3. Stepper Motor Innards

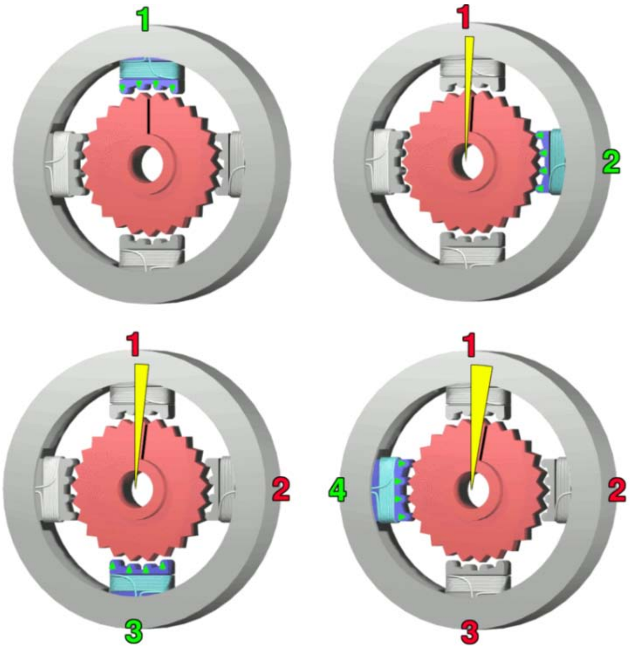
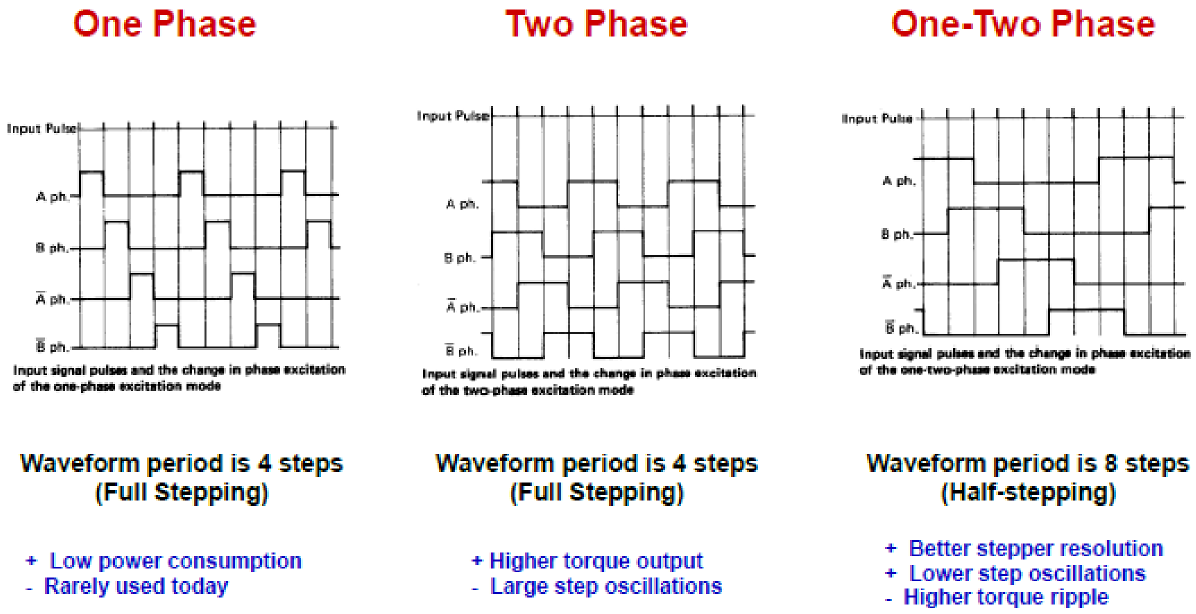


Figure 4. Full Stepping Teeth Alignment

This method of movement sets the stepper motor apart from other motors because the stator and rotor never touch, this means great life span. Since the rotor and stator do not touch there is no mechanical wear. The motor has over speed protection since if the switching speed exceeds the motors capabilities it will simply stop moving and not damage the motor. Also since this motor moves one step at a time it has excellent low speed control. There are some downfalls though. With stator and rotor being separated, the motor requires large amounts of power to move and it is also much larger than other types of motors with similar torque.

The stepper motor is capable of moving in a higher resolution than just 1 full step. This is useful because the larger amount of steps the more torque that can be produced. Figure 5 below shows on the left the method just described. Full stepping is created by moving from one phase to another. This consumes less power but creates very little torque. To gain more torque more than 1 phase must be turned on at a time. The graph to the far right shows how higher resolution can be gained by switching between one and two phases. Even higher precision can be gained by using pulse width modulation on the phases.


One Phase
Two Phase
One-Two Phase
Input Pulse
A ph.
B ph.
Ā ph.
B̄ ph.
Input signal pulses and the change in phase excitation of the one-phase excitation mode
Input signal pulses and the change in phase excitation of the two-phase excitation mode
Input signal pulses and the change in phase excitation of the one-two-phase excitation mode
Waveform period is 4 steps (Full Stepping)
Waveform period is 4 steps (Full Stepping)
Waveform period is 8 steps (Half-stepping)
+ Low power consumption
- Rarely used today
+ Higher torque output
- Large step oscillations
+ Better stepper resolution
+ Lower step oscillations
- Higher torque ripple

Source: Technical Information on Stepping Motors, Oriental Motor

Figure 5. Stepping Characteristics

DT6 is using the DRV8824 motor driver IC to generate the necessary PWM cycles to phase A and B. This IC has a very straight forward interface and can power a 24V motor at

1.6As. The IC requires one supply voltage from this a 3.3V output is created, this is very nice since this chip can power the MCU, a separate power supply is not needed. The DRV8824, as seen in Figure 6, has 7 general port inputs:

nEnable	turns on/off the current going to the motor
DIR	tells the motor to rotate clockwise or counter-clockwise
MODE0,1,2	1/2,1/4,1/8 stepping resolution and can be mixed together to make 1/32
nSLEEP	puts the motor in/out of sleep mode
nRESET	clears all the signals to the motor when input is low

There are two output status pins:

nHOME	is set high when 1 full step has been made, and low for all time else
nFAULT	is set high when IC has overheated or pulling too much current

There are three analog input pins:

AVREF	scales the current through Phase A based on the voltage at this pin
BVREF	scales the current through Phase B based on the voltage at this pin
DECAY	scales the decay of the PWM signal based of the voltage at this pin

There is one pin that takes a pulse:

STEP	every rising edge the motor moves 1 step

There are four outputs that connect directly to the motor:

AOUT1	Phase A
AOUT2	Phase A’
BOUT1	Phase B
BOUT2	Phase B’

ISENA and ISENB can be used to measure the current running through Phase A and B respectfully. Having nHOME and nFAULT is a really great feature since they can be tied back to the controlling MCU and handle the event of a fault and also determine if the motor is running at the correct speed.

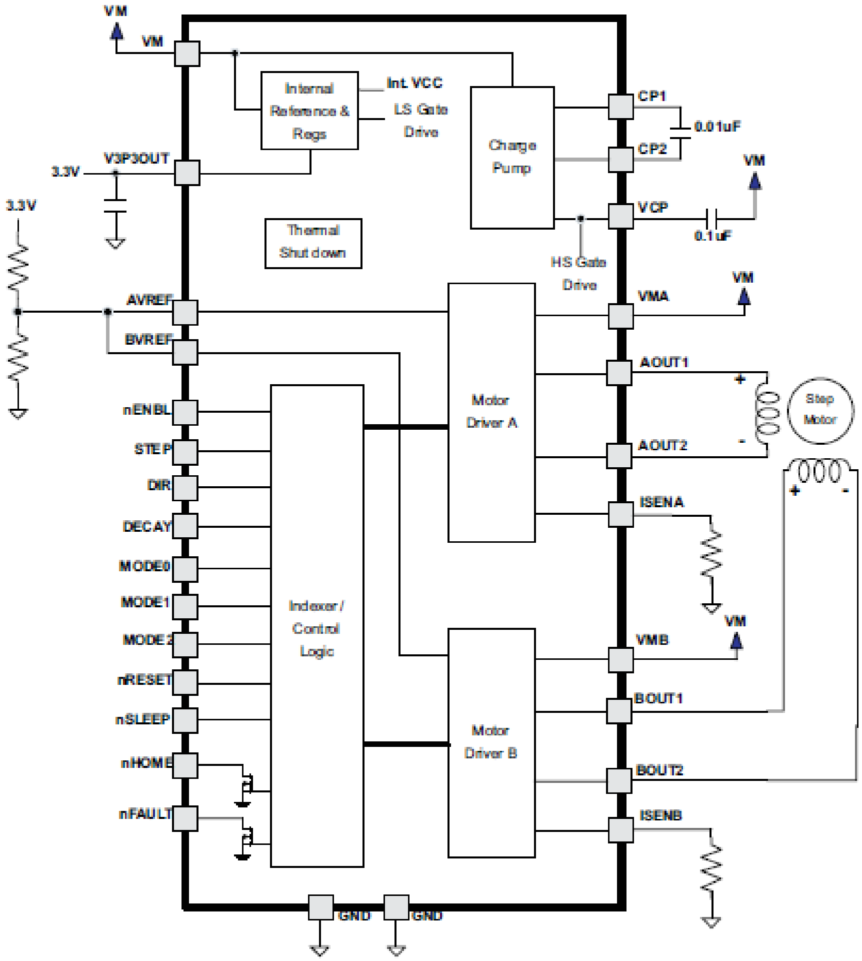


Figure 6. DRV8824

Chapter 2

Fast Diagram

Before beginning the actual design work of the project, DT6 had to sit down and figure out all of the components that were needed to complete the project. The first step in this process was to create a FAST diagram which outlines every process that is needed to control the motor. The diagram can be seen below in Figure 7.

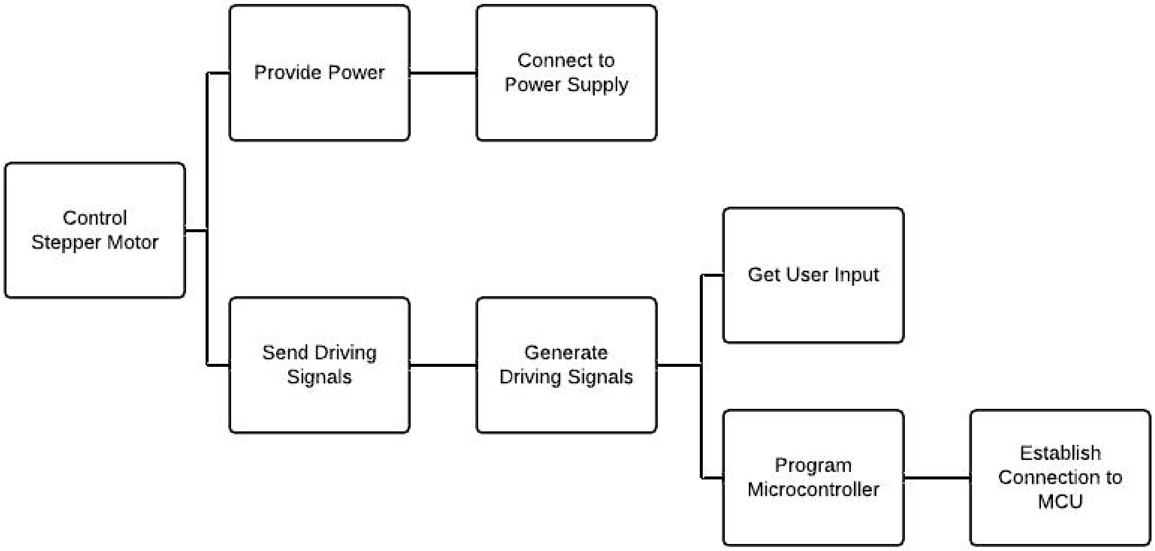


Figure 7. Fast Diagram

As seen in Figure 7 the most important processes to control the motor are sending driving signals, generating those driving signals, getting user input and programming the microcontroller. These processes are the main parts of the design. To send the driving signals DT6 will have to create a driver board with the DRV8824 IC. What will generate these signals to drive the motor? This will be done by two microcontrollers, the MSP430F1612 and the Stellaris LM3S9B96. They will need to connect to the DRV8824 board, so a DIMM slot will be implemented on the board and the microcontrollers will each be placed on their own controlCARD. After these steps are completed DT6 will have to program the microcontrollers to accept user input from the windows application to control the motor.

Conceptual Design

DT6 now knows what parts are needed for the design. There are a total of five physical components to the design. As shown in Figure 8, these components are: the PC, the controlCARD with the MSP430, the controlCARD with the Stellaris, the EVM board which has the DRV8824 and the stepper motor. The PC will communicate with the controlCARDs by a USB connection. The PC will also be able to communicate with the Stellaris by Ethernet. It will also use the USB connection to program the microcontrollers. The PC also has a GUI that will be able to sends signals to the microcontrollers. Each controlCARD is connected to the DRV8824 through the DIMM100 slot that will be installed on the EVM board. The microcontrollers will process the signals from the PC and then send the signals to the DRV8824 which will drive the motor.

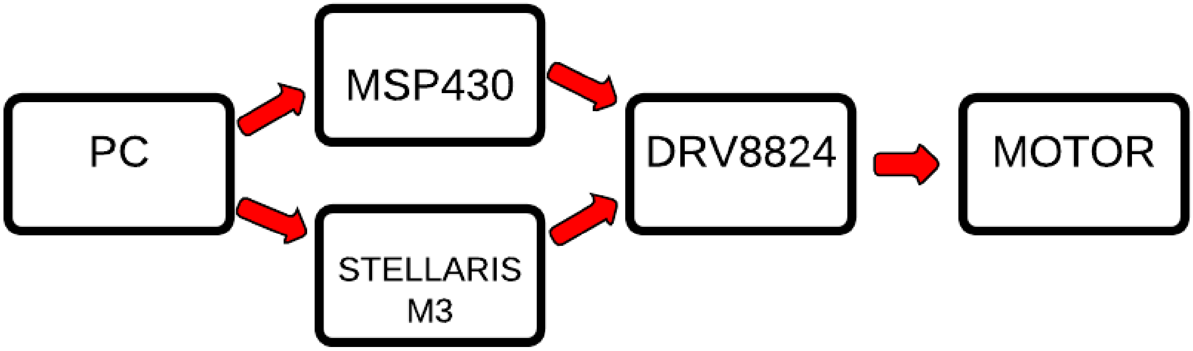


Figure 8. Completed System

Proposed Design

The proposed solution encompasses the hardware redesign of the DRV8824 evaluation board, and the design of the MSP430F1612 DIMM controlCARD. In addition, software to control the motor will need to be ported to the Stellaris M3 DIMM cards. The hardware redesign consists of removing the digital section of the DRV8824EVM and replacing it with a DIMM slot. To achieve this, the pin outs of the M3 Cortex card will be carefully studied in order to design a functional MSP430F1612 controlCARD.

There are three major steps for accomplishing our project.

- Removal of the MSP430F1612 and definition of DIMM ports

  The DRV8824EVM has a MSP430 microcontroller on board that will be removed. There will then be input and output wires that need to be connected to the DIMM slot. The DIMM ports will be ported based on the pins of the Stellaris controlCARDs, so both can control the motor driver.

- Design of MSP430F1612 controlCARD

  Once the definition of the DIMM ports is selected then the design of the MSP430F1612 controlCARD can take place. The code will be the same that the DRV8824EVM came stock with, but the controlCARD will be a removable extension of the DRV8824EVM.

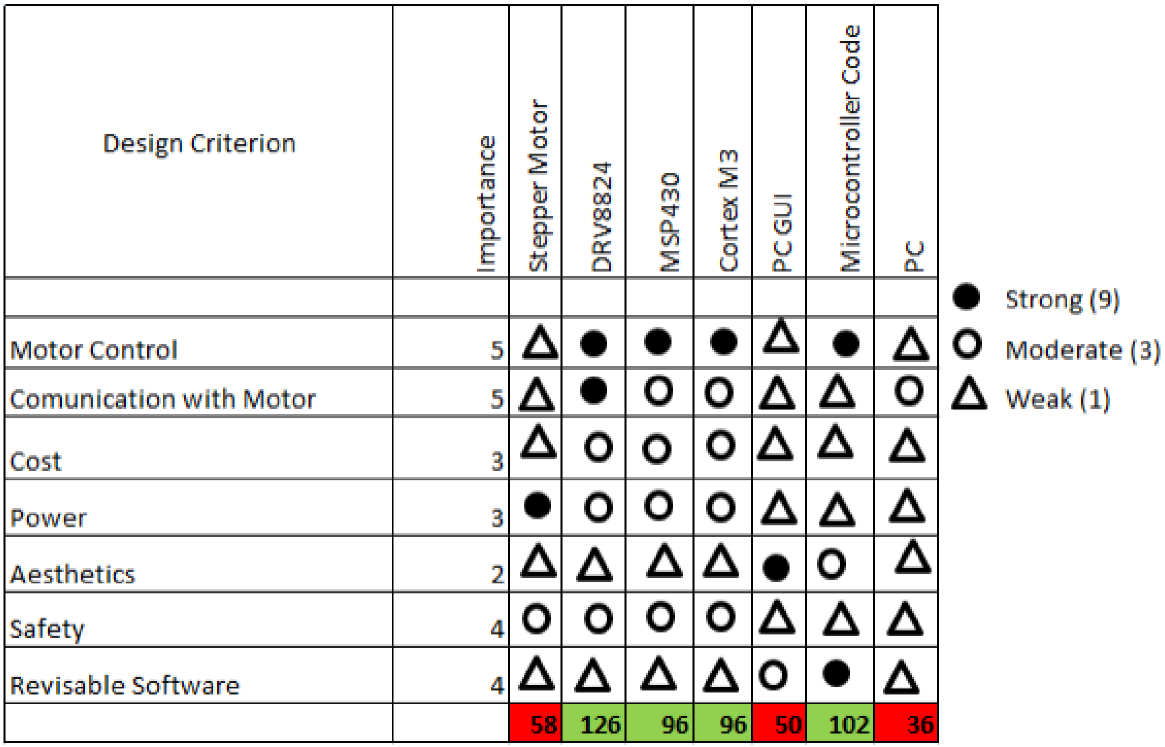
- Coding of the Stellaris controlCARD

  Since the Stellaris has never been programmed to work with the DRV8824 motor driver before, the code from the MSP430F1612 will be ported to the Stellaris controlCARD. This will require much porting of 16-bit registers to 32-bit registers. The reason this will work is because ARMs architecture allows for 32-

bit micro-controllers to still run 16-bit instructions.

Ranking of Conceptual Designs

The next part of the design was to figure out which components were most important to the final design and which parts needed more time to be completed. This was accomplished by creating a house of quality to determine the critical customer requirements (CCRs) of the project. The house of quality can be seen below in Figure 9.



Design Criterion	Importance	Stepper Motor	DRV8824	MSP430	Cortex M3	PC GUI	Microcontroller Code	PC
Motor Control	5	△	●	●	●	△	●	△
Comunication with Motor	5	△	●	○	○	△	△	○
Cost	3	△	○	○	○	△	△	△
Power	3	●	○	○	○	△	△	△
Aesthetics	2	△	△	△	△	●	○	△
Safety	4	○	○	○	○	△	△	△
Revisable Software	4	△	△	△	△	○	●	△
		58	126	96	96	50	102	36

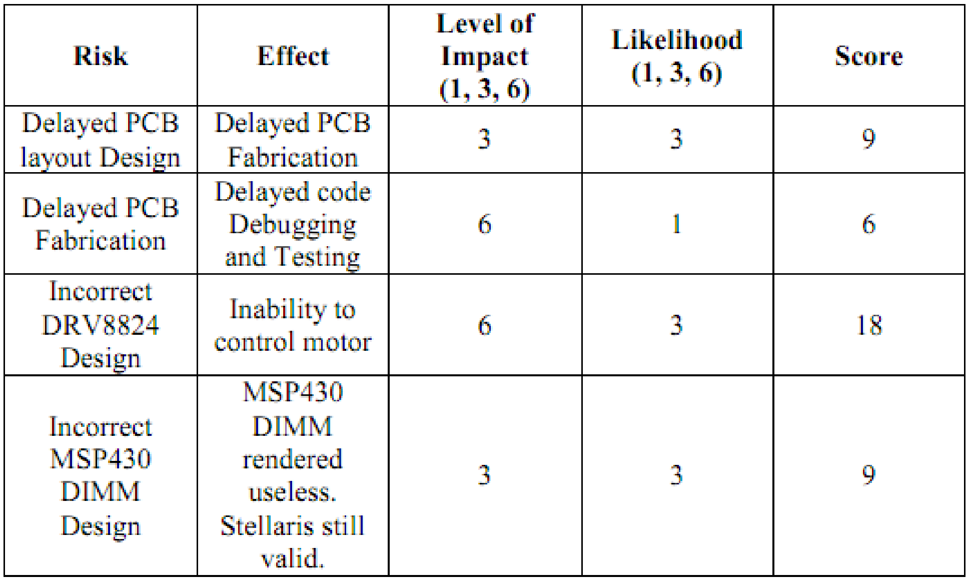
● Strong (9)
○ Moderate (3)
△ Weak (1)

Figure 9. House of Quality to determine CCRs.

As shown in Figure 9, the components of the design and the design criterion were compared. Each design criterion was given an importance value (1 being least important, 5 being most). Then each component is compared to each of the design criteria and assigned a value of 9, 3, or 1. Then by multiplying the importance value with the correlation value and adding up each column, the resulting numbers at the bottom of the table are calculated. The values highlighted green are the most vital parts of the design. Whereas the values highlighted red do not have much to do with the design portion of the project. DT6 concluded that the most important part of the project was to finish the DRV8824 EVM. This was because without a working EVM it would be impossible to have a running stepper motor by design day. The two controlCARDs and programming of the cards were also very important to the design.

Risk Analysis

There are some major risks associated with our project. Figure 10 below lists these risks. Each risk is given a rating of 1,3 or 6 for the impact of the risk and the likelihood it occurs. 1 is for low risk and 6 is for high risk. These two categories are multiplied together for each risk to get the final risk value. DT6 has concluded that an incorrect DRV8824 design would greatly affect DT6's ability to finish the project



Risk	Effect	Level of Impact (1, 3, 6)	Likelihood (1, 3, 6)	Score
Delayed PCB layout Design	Delayed PCB Fabrication	3	3	9
Delayed PCB Fabrication	Delayed code Debugging and Testing	6	1	6
Incorrect DRV8824 Design	Inability to control motor	6	3	18
Incorrect MSP430 DIMM Design	MSP430 DIMM rendered useless. Stellaris still valid.	3	3	9

Figure 10. Risk Analysis Chart

Proposed Schedule

Since the EVM was the most important part of the design, DT6 decided the first task that needed to be completed was the DRV8824 board. This is also due to the amount of time it would need to be fabricated. This board as well as the MSP430 controlCARD needed to be completed early in case there was a mistake and needed to be re-fabricated. DT6's goal was to complete the design of these two boards by 3/1/11. The DRV8824 EVM was actually finished 3/3/11 and the controlCARD was finished 3/21/11. The EVM was finished roughly on time and the controlCARD was a couple weeks late. This was not a huge problem because DT6 had planned for delays in the design process in their GANTT chart. DT6 also planned to be done with coding the Stellaris controlCARD by 4/20/11. This task was successfully completed on time. DT6's GANTT chart can be seen in its entirety in the appendix.

Proposed Solution Cost

Although DT6 spend over $1000 designing there DRV8824EVM and MSP430F1612 controlCARD this could be reduced by mass producing the product. The production of the two boards can be broken down into several different items. Fabrication of the printed circuit board for the DRV88xxEVM is estimated at $25 per board for large quantities. Parts for each board would be estimated at $20 also when ordered in bulk. The final part of the production for the hardware is the assembly which is estimated at about $25 per board. As with anything, the cost associated with each board manufactured decreases as the number of boards increases.

After the EVM and the controlCARD are produced they are ready for distribution. Texas Instruments has a vast website containing all of their solutions to motor control. These solutions can be ordered off of the website or over the phone with a sales representative. The

DRV8824EVM would be priced comparably to the other evaluation modules at $100 per board. TI also has multiple warehouses located in several locations which will allow for the product to be produced in large quantities and stored until the customer places an order.

Chapter 3

MSP430 controlCARD Pin Layout

A major part of the MSP430 controlCARD design was to decide which MSP430 pins would connect to the pins on the actual controlCARD. The first part of developing a design was to figure out which pins on the microcontroller were vital to motor control. This was accomplished by analyzing the DRV8824 EVM schematic and DRV8824 datasheet. The schematic shows the entire DRV board wiring connections including which MSP430 pins send signals directly to the DRV8824 and hints at what registers the MSP430 is using. From this schematic there were a total of 20 pins that were directly related to controlling the stepper motor. These pins were the following:

- P3.0 – SEL0 // P3.1 – SEL1
  - Select signals used to determine which type of motor driver IC is connected to the microcontroller.
- P4.1 – GDECAY
  - Decay signal where a low value corresponds to a slow decay and a high value corresponds to a fast decay.
- P4.2 – nEN
  - Enable signal. Motor is enabled when this value is low.
- P4.3 – STP
  - PWM signal. This is the signal that determines the speed of the motor. The motor will step on each rising edge, so the higher the frequency the faster the motor will run.
- P4.4 – DIR
  - Signal that controls the direction that the motor spins.
- P4.5 – NC
  - No connection pin. Even though this is a no connection pin, this NC pin on the MSP430 is connected to the NC pin on the DRV8824 so the signal still needs to be able to leave the controlCARD.
- P4.6 – nRESET
  - When reset signal is low system is reset.
- P4.7 – nSLEEP
  - When sleep signal is low system is put to sleep.
- P5.0 – MD2 // P5.1 – MD1 // P5.3 – MD0

    - Mode 0-2 signals. These signals are set to a certain combination to cut the speed of the motor by 1/2, 1/4, 1/8, 1/16, or 1/32 of full speed.
- P5.2 – nHM
    - The home signal is set low when the stepper motor is at the home state.
- P6.6 – AGVREF // P6.7 – BGVREF
    - These signals are analog reference voltages for bridge A and B.

The remaining 5 signals that need to come off of the controlCARD are the necessary signals for JTAG. JTAG is mainly used for downloading and debugging integrated circuits. These signals include TDI and TDO which are the input and output signals. There is also the TCK and TRESTn signals which are the clock and reset signals for debugging. The final signal is TMS, this signal is the test mode select signal. The DRV8824 EVM schematic can be seen in its entirety in the appendix.

Now that the important signals are known, the next step in the pin layout process was to figure out which controlCARD pins should be connected to the signals listed above. This was done by comparing existing controlCARD pin outs to figure out some standards of controlCARD design. The MSP430 controlCARD pin out was based on the layouts of two other controlCARDs. These cards were the Stellaris LM3S9B95 and the C2000 F28335. By analyzing these cards some similarities were noticed. First some background on the 100 pin DIMM card. The 100 pin card has 50 pins on each side, and each side is broken into three sections of pins. The first section has 6 pins, the seconds has 16 and the third has 28 pins. The first section on both cards had 2 pins connected to an isolated 3.3V and included the TX and RX pins. These pins are the receiving and transmitting pins used in the UART protocol. This protocol is used for communicating with the microcontroller through Recommended Standard (RS232). The second section is where all of the analog signals were placed and the third section is where all of the GPIO/Special function signals were connected. Another similarity noticed was that all of the ground and 5V pins were the exact same on both layouts. It was also noticed that the last few GPIO pins is where they placed the JTAG signals on both cards.

With all this information gathered, it was time to actually pick the controlCARD pins for the signals. For the MSP430 controlCARD, all of the ground and 5V pins were placed in the same location as the Stellaris and C2000 cards. The two important analog signals, AGVREF and BGVREF, were connected in the second section because that is where the Stellaris cards had the analog signals. The JTAG signals were placed at the bottom of section three and all the other important GPIO signals were placed at the beginning of the third section since this is where PWM is located. After all of these pins were placed, the other signals were just connected to any open controlCARD pins. The final MSP430 controlCARD pinout can be seen in the appendix as well as the pin layouts for the Stellaris LM3S9B95 and the C2000 F28335.

PCB Design

A large part of DT6's project was to modify current hardware and create new hardware, the new DRV8824EVM and the MSP430 conrolCard. For both of these items, the Altium Designer software package was used to generate the design files. Altium is a professional grade printed circuit board, PCB, CAD software package. While Altium has a relatively steep learning curve, it is a very powerful tool capable of creating complex circuits on a large scale. Texas Instruments is moving towards using this software for all their PCB designs, which is why DT6 chose to use the tool.

The first circuit that DT6 created was the DRV8824EVM. This is an evaluation module, EVM, for control of a stepper motor. The EVM previously contained the MSP430F1612 microcontroller on the circuit board. Texas Instruments required that the MSP430 and all supporting circuitry be removed from the EVM and replaced with a DIMM socket. The design files for the original EVM were provided for modification. The EVM is four layer printed circuit board. There are two main routing layers, top and bottom layers, one ground plane layer, and a third routing layer used strictly for test pins on the EVM. The use of a ground plane is very useful as it removes the need to route ground to every component on the circuit that is connected to ground. Alternatively, all that is required is to create a via down to the ground plane near the component connected to ground.

While modifying the original EVM design files, there was one problem that proved to be the most challenging. The footprint for the DIMM sockets proved to be difficult to find. The required libraries were not included with Altium. Altium hosts several libraries containing many different manufacturers' parts on their website. Once the required library was located all that was left was routing the signals to their appropriate place. As mentioned earlier, and located in the appendix, the appropriate pin connections were already determined.

The second PCB design that was required was the design of the MSP430 controlCARD. The controlCARD is a 100-pin dual inline memory module containing the MSP430 microcontroller and supporting circuitry previously on the EVM. The controlCARD is also a four layer board containing the same top and bottom layers, ground plane, and a third routing layer. The controlCARD however has several other aspects that are different from the EVM. Because the controlCARD needs to fit into a defined socket it has specific dimensions, defined in the appendex.

These dimensions proved to be one of the more challenging aspects related to the controlCARD. As there was no template containing the appropriate dimensions for the controlCARD, the card was created from scratch. AutoCAD was used to create a drawing containing the correct dimensions. This drawing was imported into Altium to create the basic board layout. Once the board dimensions were set in Altium design could continue.

The controlCARD proved to be more challenging because it requires custom specifications. These specifications refer to the contact points on the edge of the card. These contacts are called “gold fingers” and required a specific layout to be correct. This layout was not documented very well. Contact was made with several Altium representatives to determine the correct layout. The card edges require the solder mask to cover all of the pins, however the paste for the pins must have a negative overlap with the top and bottom layer pads.

Overall both PCB designs were successful. Both are fully functional and operate as expected to control the motor. A few minor changes to the controlCARD are recommended if a second iteration were to occur. The first modification is to add vdd and ground vias to allow the controlCARD to be powered externally aside from USB. Pins 1 and 51 would also be changed to

3.3V Isolated. DT6 would also change the 3.3V signal on the EVM to be supplied from the output of the DRV8824 instead of being supplied from the DIMM.

DRV8824 C++ Driver

This project included porting code from the MSP430F1612 to the Stellaris LM3S9B96. To accomplish this, research of datasheets and TI forums was necessary. The end goal was to create a header file that would contain all the information to control a DRV8824 with a Stellaris MCU. The resulting code looks much different than the MSP430 code because the Stellaris differs from the MSP430 in many ways; including processing speed, number of pins, number of special functions each pin has, and way code is written.

The MSP430F1612 is a 8MHz MCU which contains 64 pins, 48 of which can be used as General Port Input Output (GPIO). The Stellaris LM3S9B96 is a 80MHz MCU which contains 100 pins. MSP430s pins have generally only 2 functions, 1 being GPIO which means it only outputs or inputs a high/low signal, and 1 special function such as communication, pulse width modulation (PWM), or analog to digital converter (ADC). The Stellaris' have GPIO on all pins and many special functions. This is because the Stellaris architecture has muxing inside the chip that allow for the switching of functions between pins.

To being porting the code DT6 first had to learn how the MSP430 handled the control of the DRV8824. The general programing method for the MSP430's are there is a list of variables, these variables can represent control registers and bits in the control registers. For example the code below P1DIR is saying that the direction register of port 1 is going to be set to what is on the right hand side of the equal sign, since BIT0 is on the right hand side then P1.0 is now an output, and the next like P1OUT sets the port 1 registers output, and since BIT0 is there again now the pin at P1.0 will have a 3.3V output.

```
P1DIR = BIT0;   // turn LEDS off
P1OUT |= BIT0;    // make LEDS outputs
```

Now for the Stellaris on the other hand uses variables to represent registers and control bits but it sets them with functions. The code below is how the Stellaris sets port A0 to 3.3V. SysCtlPeripheralEnable() function is used to set ports to GPIO or their specific special function. GPIOPinTypeGPIOOutput() function takes the base port (A-H) and the pin (0-7) and sets the direction register to an output at that pin. GPIOPinWrite() function set the pin to an high or low output.

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PIN_0);
GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_0, GPIO_PIN_0);
```

This may look like more work but setting up function like this is makes it very simple to implement the special features in the code.

DT6 has created a header file that drives the DRV8824, it includes the functions to accelerate and decelerate the motor. There is also function that set the GPIO pins on the DRV8824 and the frequency of the pulses. Below is a list of the function that controls the non GPIO features. DRV8824_init() is a function that initializes all the ports to the DRV8824 and sets the PWM to a 50% duty cycle. Frequency takes a number in hertz then sends that frequency to the PWM. ACCEL and DECEL, accelerate and decelerate the motor starting at the current frequency at time of call to a desired frequency at a certain acceleration rate and acceleration time base. The STEP_ON() function is used to move the motor a certain amount of steps. STEP_HL() inverts the PWM signal.

```
extern void DRV8824_init();
extern void Frequency(ulFreq);
extern void ACCEL(pastFreq, newFreq, AccelRate, AccelTimeBase );
extern void DECEL(pastFreq, newFreq, AccelRate, AccelTimeBase );
extern void STEP_ON(cnt, Freq);
extern void STEP_HL(output);
```

Every GPIO pin on the DRV8824 has a corresponding function to set it that pin to a high or low signal and the function is in the format of PinName(High/Low). All of the actual pins are defined in the header so there is no need to call the pin name directly this makes it very easy to switch to other Stellaris MCUs. The code below shows an example of how to change the Direction pin on the DRV8824, since DT6 has placed defines at the top of the DRV8824 header now simply replace GPIO_PORTA_BASE and GPIO_PIN_5 with the correct pin on the controlCARD the customer is using and the code will be executable.

```
#define DIR_BASE       GPIO_PORTA_BASE
#define DIR_PIN        GPIO_PIN_5
```

DT6 overcame many problems when creating this header file. When running a stepper motor even when it is running at full speed with microstepping on it is still under 100KHz, and since the Stellaris runs at 80MHz it was causing issues with accurate PWM signals at low frequency. This was eventually overcome by dividing the clock by 4 and then another 16, this reduced the frequency of the PWM cycle to a max of 1.25MHz, this eventually allowed for the bandwidth of speed for the stepper motor to be better than the MSP430. Another issue DT6 had to overcome was to implement the ability to count the number of pulse that had gone by. Since the Stellaris uses function that implement interrupt and there was no function that counted PWM cycles DT6 had to come up with a solution. This problem was eventually solved by creating an interrupt that could count each time a pulse went from low to high.

Windows GUI

One of DT6’s major task was to implement the Stellaris motor driving code with the Windows GUI, this involved learning how the Windows GUI communicated with the MSP430 and how to take these inputs and create a desired output on the motor through the Stellaris MCU. Almost all of TIs solutions offer a USB to COM options, this means that when a USB is plugged into the computer it is recognized as COM port (Serial Port), this is one of the simplest communication protocols to implement because it can communicate with as low as 2 wires. Below is the code to initialize UART which I the serial communication protocol.

```
//
// Initialize the UART.
//
 SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
 GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
 UARTStdioInit(0);
```

Next the Windows GUI communicates at 9600 bauds, no parity bit, data bits 8, 1 stop bit, and no flow control. So to implement this in the Stellaris code the line that contains communication parameters in UARTStdioInit() need to be changed to the code below.

```
MAP_UARTConfigSetExpClk(g_ulBase, MAP_SysCtlClockGet(), 9600,
                (UART_CONFIG_PAR_NONE | UART_CONFIG_STOP_ONE |
                 UART_CONFIG_WLEN_8));
```

The Windows GUI sends the MCU 5 bytes of information, the first byte contains the operation code, and this is either the GPIO, DAC, Step, or Pulse setting. The remaining 4 bytes contain information for each of the parameters and change depending on what is being typed into the GUI. The MCU only sends 3 bytes back containing its firmware revision code. There are 9 operation codes that are used for the stepper motor and they are listed below.

```
Write GPIO Data
      [OPCODE = 3][ GPIO DATA ][ Not Used ][ Not Used ][ Not Used ]

Disable PWM
      [OPCODE = 0x0C][ Timer # ][ Not Used ][ Not Used ][ Not Used ]

Set Timer Output
```

```
        [OPCODE = 0x0E][ Timer # ][ Not Used ][ Not Used ][ Not Used ]

Pulse Timer Output
        [OPCODE = 0x0F][Timer Used][ Pulse Length Hi ][ Pulse Length Lo ][ Not Used ]

Write GPIO Data on SPI Port
        [OPCODE = 0x16][ GPIO DATA ][ Not Used ][ Not Used ][ Not Used ]

START STEPPER
        [OPCODE = 0x17][ Frequency Hi ][ Frequency Lo ][ Accel Rate ][ Accel Time Base ]

STOP STEPPER
        [ OPCODE = 0x18][ Frequency Hi ][Frequency Lo ][ Not Used ][ Not Used ]

STEPPER_SPEED
        [OPCODE = 0x19][ Frequency Hi ][ Frequency Lo ][ Accel Rate ][Accel Time Base ]

MOVE_STEPS
        [OPCODE = 0x1A][ Frequency Hi ][ Frequency Lo ][ STEPS Hi ][STEPS Lo ]
```

The best way to implement this communication protocol is to store the 5 bytes that are received from the Windows GUI into a buffer and then use case and if statements to determine the operation that needs to be performed. Below is the base case statement for the operation code, this is the same as the MSP430 code, but entering the code for each case statement things now have to be changed.

```
        // Switch statement for Operation Code
        switch(SerialBuffer[0])
        {
                case ( 0x03 ): // Write GPIO
                case ( 0x0C ): // nEnable & Step (Low)
                case ( 0x0E ): // nEnable & Step (High)
                case ( 0x0F ): // Pulse 1 Step
                case ( 0x16 ): // Modes
                case ( 0x17 ): // Start Stepper
                case ( 0x18 ): // Stop stepper
                case ( 0x19 ): // Update stepper
                case ( 0x1A ): // Move Steps
        }
```

Sense the MSP430 calculates frequency based of number of counts in the timer and the Stellaris uses a function that take a frequency in hertz the Stellaris has to convert timer counts to a usable frequency. Below is how the start stepper case is handled, the first byte contains 17 which activates the start stepper case statement, now the next two bytes are used to get the 16 bit timer count that is used for frequency. This is why SerialBuffer[1] is multiplied by 256 then added to SerialBuffer[2], so this give a number that does not represent a frequency, but since the MSP430 is running at 4MHz the frequency can be calculated by dividing 4MHz by the timer count. SerialBuffer[3] is the acceleration rate and SerialBuffer[4] is the acceleration time base, this is all the parameters needed for the ACCEL() function. ACCEL() is going to now accelerate the motor from 0 Hz to (ulFreq) Hz at a rate of (AccelRate) Hz per (AccelTimeBase) milliseconds.

```
case ( 0x17 ): //Start Stepper
      DesiredStepperSpeed = (SerialBuffer[1] * 256)
                                        + SerialBuffer[2];
      ulFreq = (4000000/ DesiredStepperSpeed);
      AccelRate = SerialBuffer[3];
      AccelTimeBase = SerialBuffer[4];
      ACCEL( 0, ulFreq, AccelRate, AccelTimeBase );
      pastFreq = ulFreq;
  break;
```

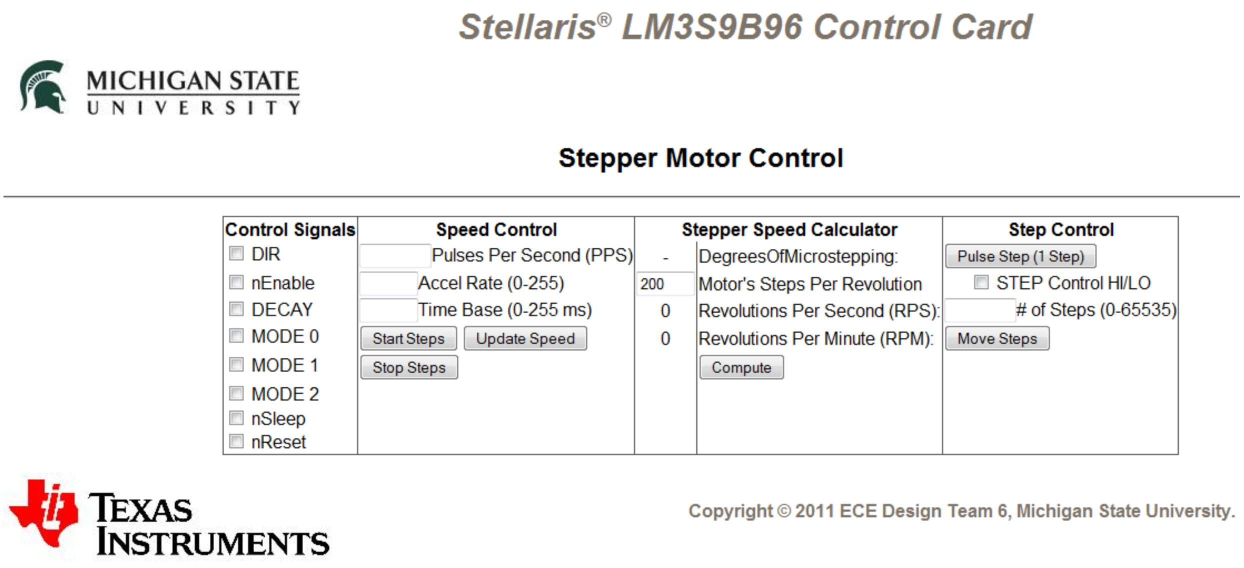
DT6 was faced with some problems when connecting the Stellaris to the Windows GUI since the GUI was not optimized for the Stellaris but the MSP430. The Windows GUI was created to send information to go directly to a register on an MSP430 meaning that SerialBuffer would be the only variable on the right hand side of an equal sign. This meant that the Stellaris code in certain portions had to do conversions before having useful data to generate an output. DT6 also discovered inaccurate commenting in the original MSP430 code provided by TI, this added confusion to the project when bytes from the Windows GUI were not cooperating with the Stellaris code when following the guidelines of the commented code. There was one problem that could not be avoid and that was the implementation of the DAC, this was not able to work with the Stellaris because the MCU provided does not have a DAC on the chip.

Ethernet Motor Control

The Stellaris LM3S9B96 has an Ethernet port attached to it, so DT6 decided it would be useful to be able to control the motor through this port. The Stellaris family of microcontrollers is used for graphical LCD screens and networking. DT6 added the goal of creating a webpage that would have the same capabilities as the Windows GUI to control the motor.

The first step was to figure out how to program the Stellaris controlCARD. DT6 found some samples of code online that would be a good starting point for this portion of the project. The program that was used as a starting point for this GUI was a program that demonstrated controlling the Stellaris card through the Ethernet port by pressing a button on the webpage that would turn the user led on and off.

The next step was to create a webpage that looked very similar to the Windows GUI and had the same functionality as that application. Figure 11 shown below is a snapshot of what the webpage looks like.



Stellaris® LM3S9B96 Control Card

Stepper Motor Control

Control Signals	Speed Control	Stepper Speed Calculator		Step Control
☐ DIR	Pulses Per Second (PPS)	-	DegreesOfMicrostepping:	Pulse Step (1 Step)
☐ nEnable	Accel Rate (0-255)	200	Motor's Steps Per Revolution	☐ STEP Control HI/LO
☐ DECAY	Time Base (0-255 ms)	0	Revolutions Per Second (RPS):	# of Steps (0-65535)
☐ MODE 0	Start Steps Update Speed	0	Revolutions Per Minute (RPM):	Move Steps
☐ MODE 1	Stop Steps		Compute	
☐ MODE 2				
☐ nSleep				
☐ nReset				

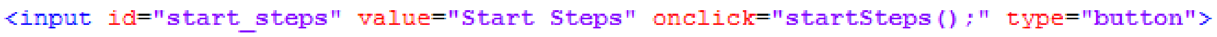
Copyright © 2011 ECE Design Team 6, Michigan State University.

Figure 11. Motor Control Webpage

All of this code was written in HTML with embedded JavaScript. The way the code works is once a button or check box is clicked a function in the JavaScript will be called. This function will then create a string and place this string in a buffer that gets sent to another program. Each button and checkbox has its own function associated with it. For example when the button “start steps” is pressed the values in the “pulses per second”, “accel rate” and “time base” boxes are read in the JavaScript function. From there a string with all of the above information is created and sent to another file where all the calculations will be done. The code for the above webpage can be found online at http://www.egr.msu.edu/classes/ece480/capstone/spring11/group06/index.html.

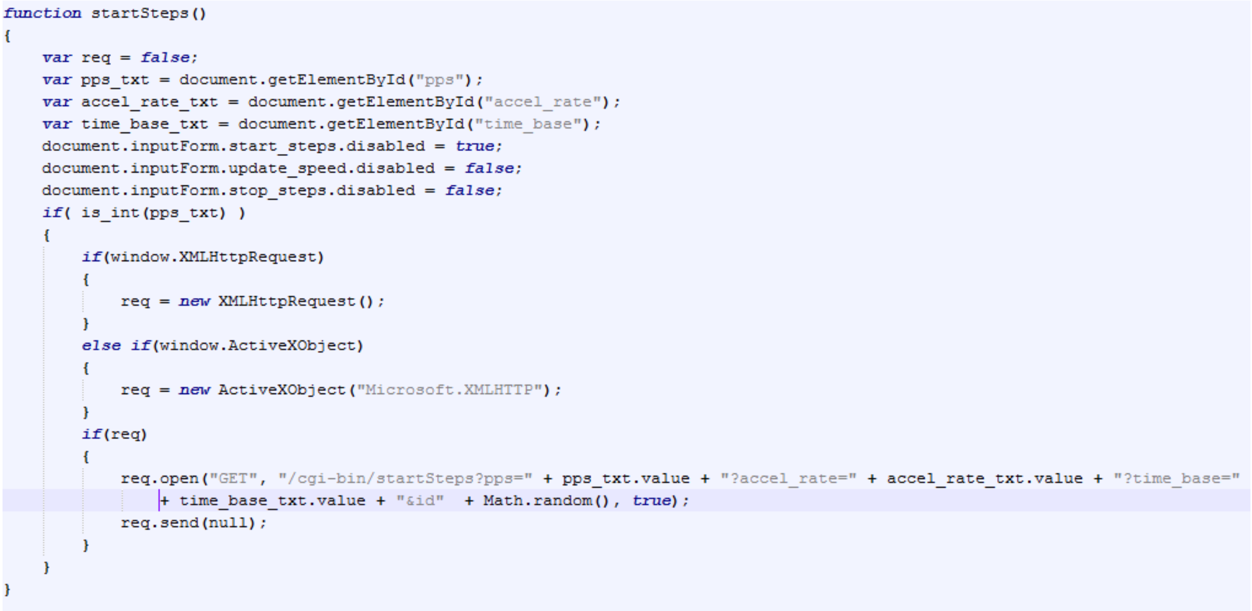
Now that the information from the webpage has been received the actual calculations can be done. IAR Embedded Workbench was the debugging program used to write the code and program the microcontroller. There are 3 main files that DT6 wrote for relaying the information from the webpage to motor. The first file is just used to look at the string sent from the JavaScript and call the appropriate functions. These function calls could be in either of the other two files. One of the files, the DRV8824 driver file, is also shared by the Windows application program. It includes functions to initialize ports, turn certain signals on an off and PWM signal creation. This is where all the actual work is done. The other file is only used by the Ethernet motor control program. This file will parse the string to extract values such as PPS and convert them into decimal form. It also handles any functions that are not common to both the webpage and the windows application.

The process of how the information from the webpage gets to the motor will now be shown. The function code for starting the motor will be used as the example.



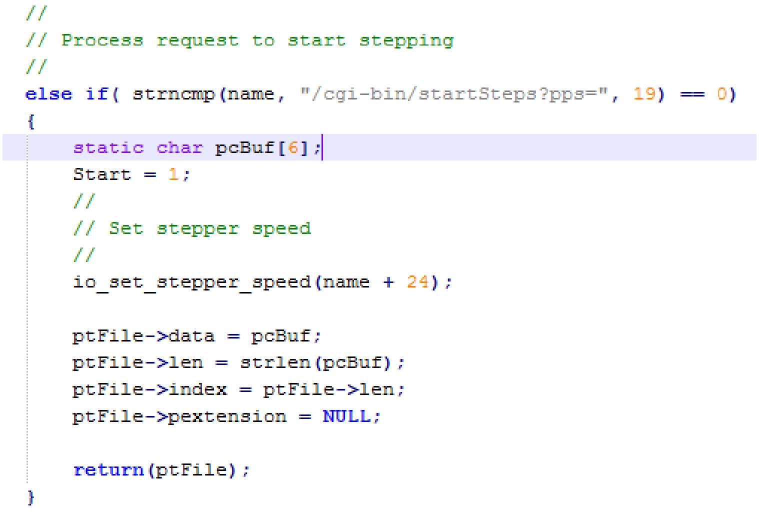
```
<input id="start_steps" value="Start Steps" onclick="startSteps();" type="button">
```

The above line of HTML code calls the function startSteps() when the “start steps” buttons is pressed.



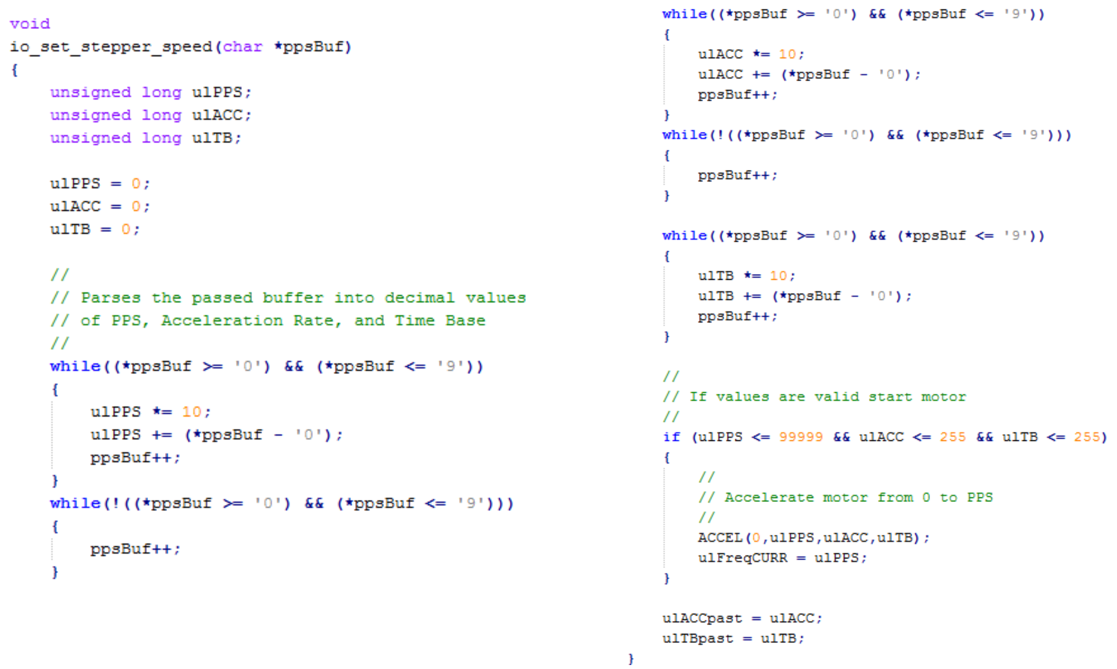
```
function startSteps()
{
    var req = false;
    var pps_txt = document.getElementById("pps");
    var accel_rate_txt = document.getElementById("accel_rate");
    var time_base_txt = document.getElementById("time_base");
    document.inputForm.start_steps.disabled = true;
    document.inputForm.update_speed.disabled = false;
    document.inputForm.stop_steps.disabled = false;
    if( is_int(pps_txt) )
    {
        if(window.XMLHttpRequest)
        {
            req = new XMLHttpRequest();
        }
        else if(window.ActiveXObject)
        {
            req = new ActiveXObject("Microsoft.XMLHTTP");
        }
        if(req)
        {
            req.open("GET", "/cgi-bin/startSteps?pps=" + pps_txt.value + "?accel_rate=" + accel_rate_txt.value + "?time_base="
                + time_base_txt.value + "&id"  + Math.random(), true);
            req.send(null);
        }
    }
}
```

The above JavaScript function first grabs the values of pulses per second, acceleration rate and time base from the webpage form. It then sends an http request to communicate with the program loaded on the microcontroller. If that request is successful it sends a string denoting that start steps has been pressed and it also includes the values of the three values obtained earlier.



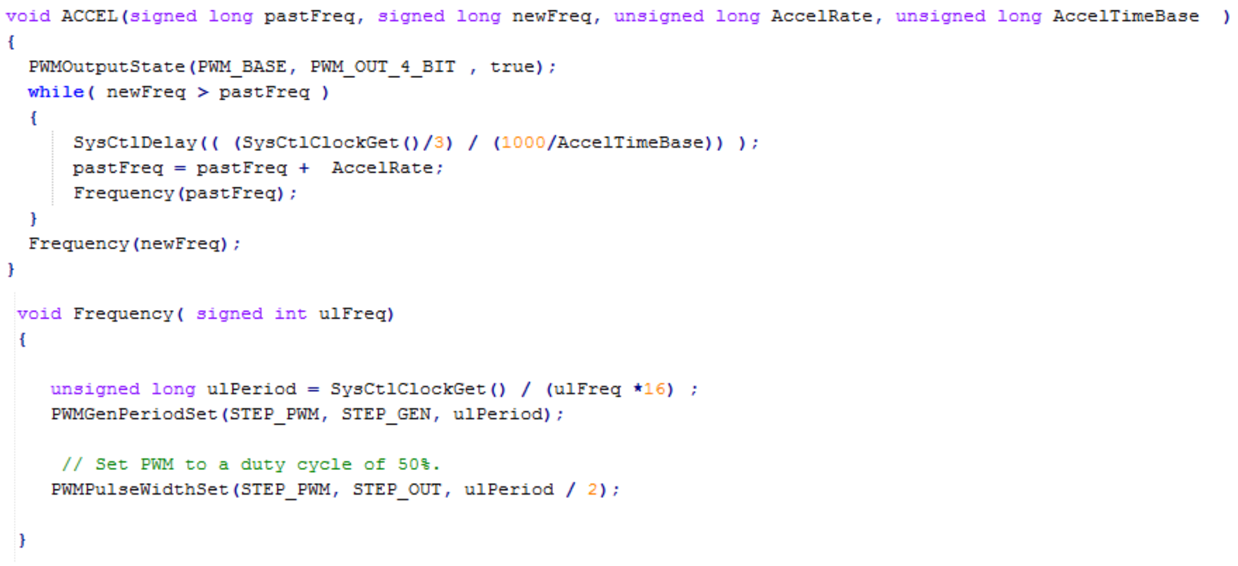
```
//
// Process request to start stepping
//
else if( strncmp(name, "/cgi-bin/startSteps?pps=", 19) == 0)
{
    static char pcBuf[6];
    Start = 1;
    //
    // Set stepper speed
    //
    io_set_stepper_speed(name + 24);

    ptFile->data = pcBuf;
    ptFile->len = strlen(pcBuf);
    ptFile->index = ptFile->len;
    ptFile->pextension = NULL;

    return(ptFile);
}
```

The above if statement looks for the startSteps string. If it finds it, it will call the function io_set_stepper_speed and send the piece of the string that contains the three values to that function.



```
void
io_set_stepper_speed(char *ppsBuf)
{
    unsigned long ulPPS;
    unsigned long ulACC;
    unsigned long ulTB;

    ulPPS = 0;
    ulACC = 0;
    ulTB = 0;

    //
    // Parses the passed buffer into decimal values
    // of PPS, Acceleration Rate, and Time Base
    //
    while((*ppsBuf >= '0') && (*ppsBuf <= '9'))
    {
        ulPPS *= 10;
        ulPPS += (*ppsBuf - '0');
        ppsBuf++;
    }
    while(!((*ppsBuf >= '0') && (*ppsBuf <= '9')))
    {
        ppsBuf++;
    }
    while((*ppsBuf >= '0') && (*ppsBuf <= '9'))
    {
        ulACC *= 10;
        ulACC += (*ppsBuf - '0');
        ppsBuf++;
    }
    while(!((*ppsBuf >= '0') && (*ppsBuf <= '9')))
    {
        ppsBuf++;
    }

    while((*ppsBuf >= '0') && (*ppsBuf <= '9'))
    {
        ulTB *= 10;
        ulTB += (*ppsBuf - '0');
        ppsBuf++;
    }

    //
    // If values are valid start motor
    //
    if (ulPPS <= 99999 && ulACC <= 255 && ulTB <= 255)
    {
        //
        // Accelerate motor from 0 to PPS
        //
        ACCEL(0,ulPPS,ulACC,ulTB);
        ulFreqCURR = ulPPS;
    }

    ulACCpast = ulACC;
    ulTBpast = ulTB;
}
```

The above function first parses the three values form the string and converts them into decimals. Then it checks to see if the values are legal inputs. Then the values get sent off to the ACCEL function where the actual signals will be set.



```
void ACCEL(signed long pastFreq, signed long newFreq, unsigned long AccelRate, unsigned long AccelTimeBase  )
{
  PWMOutputState(PWM_BASE, PWM_OUT_4_BIT , true);
  while( newFreq > pastFreq )
  {
      SysCtlDelay(( (SysCtlClockGet()/3) / (1000/AccelTimeBase)) );
      pastFreq = pastFreq +  AccelRate;
      Frequency(pastFreq);
  }
  Frequency(newFreq);
}

 void Frequency( signed int ulFreq)
 {

    unsigned long ulPeriod = SysCtlClockGet() / (ulFreq *16)  ;
    PWMGenPeriodSet(STEP_PWM, STEP_GEN, ulPeriod);

     // Set PWM to a duty cycle of 50%.
    PWMPulseWidthSet(STEP_PWM, STEP_OUT, ulPeriod / 2);

 }
```

The ACCEL and Frequency functions are shown above. The ACCEL function keeps stepping the frequency up based on the time base and acceleration rate. Every time it does this it calls the frequency function which is where the new PWM step signal with the new frequency is created. So the PWM keeps stepping up its frequency until it reaches the desired frequency. This is basically how the controlling the stepper motor through a webpage works. All of the above code with all of the functions can be found in its entirety online at http://www.egr.msu.edu/classes/ece480/capstone/spring11/group06/index.html.

Chapter 4

Product Verification

The two PCBs fabricated by DT6 can be seen in Figure 12 below. To see full demonstration video of the DRV8824EVM and MSP430F1612 CC working, please visit DT6 at http://www.egr.msu.edu/classes/ece480/capstone/spring11/group06/media.html. This demonstration will show that the hardware is correct and demonstrates the functionality of the Windows and Ethernet GUI.

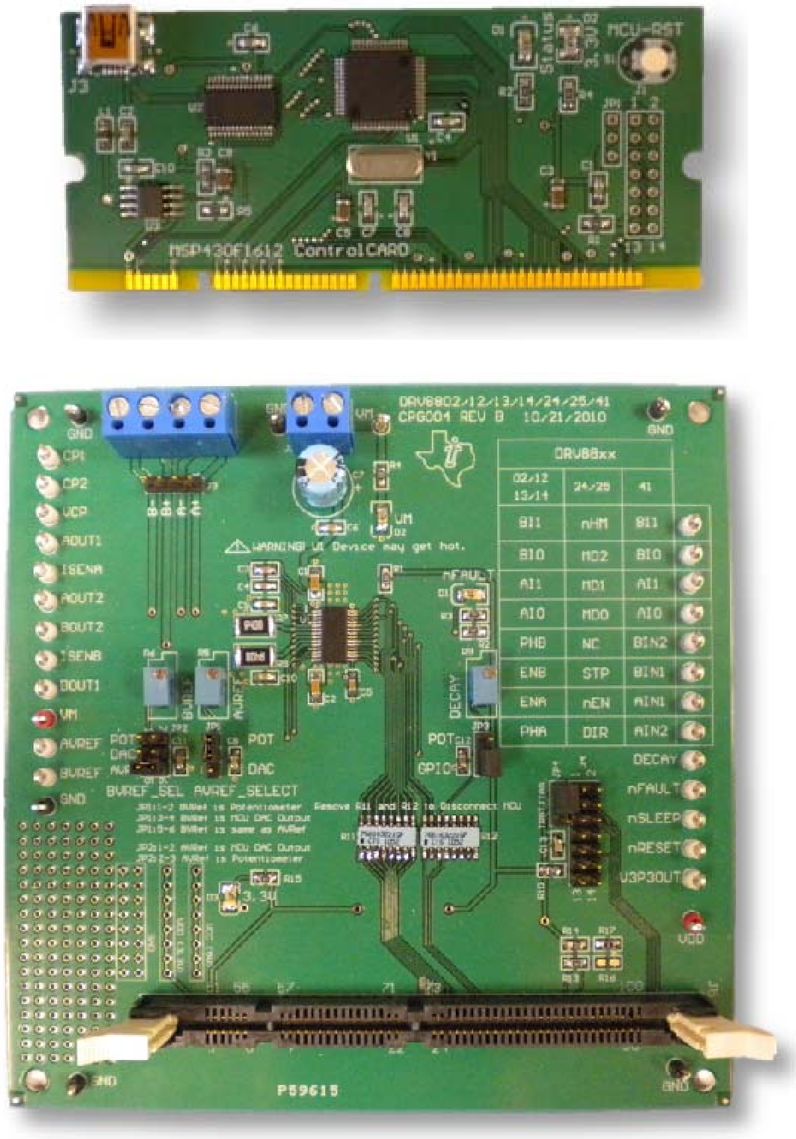
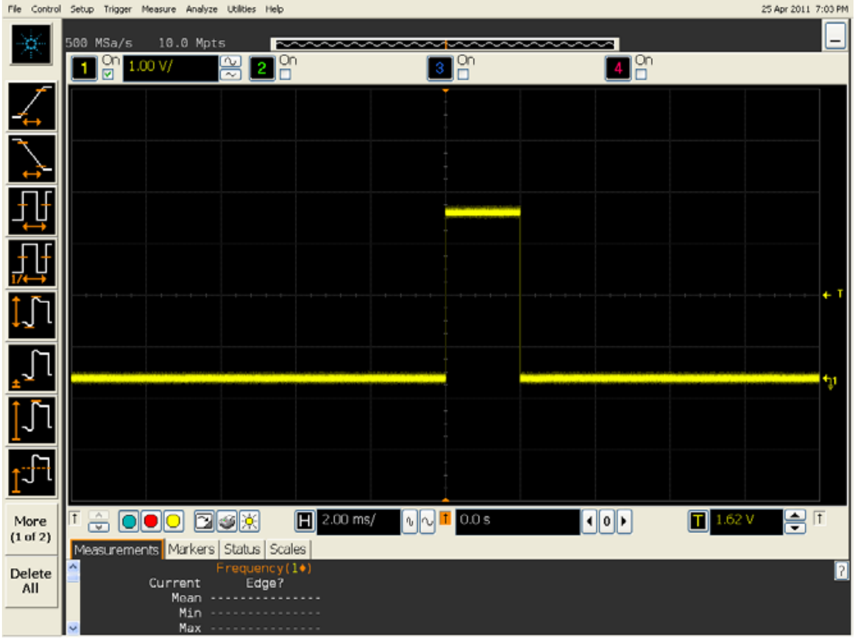


Figure 11. MSP430F1612 CC and DRV8824EVM

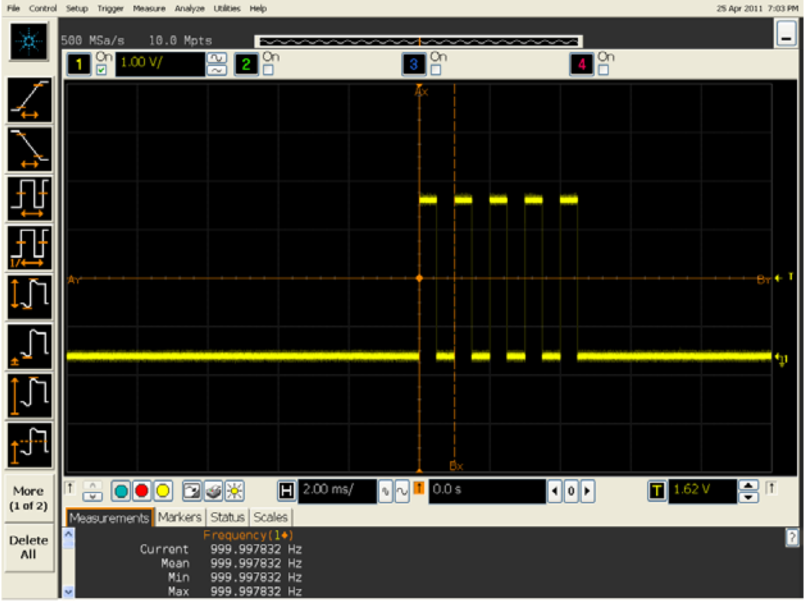
In order to prove some of the functionality of the DRV8824EVM in this paper let's look at some waveforms produced from running the motor. The first waveform (Figure 13) shows the input of the STEP pin that is generated by the Stellaris. This is the pulse 1 function of the DRV8824 Driver.



```
          Frequency(1♦)
Current      Edge?
   Mean ---------------
    Min ---------------
    Max ---------------
```

Figure 12. Pulse 1 Step

The next waveform (Figure 14) shows the STEP_ON() function being implemented with an input of 5 steps and a frequency of 1KHz. This waveform shows exactly that there is 5 pulses

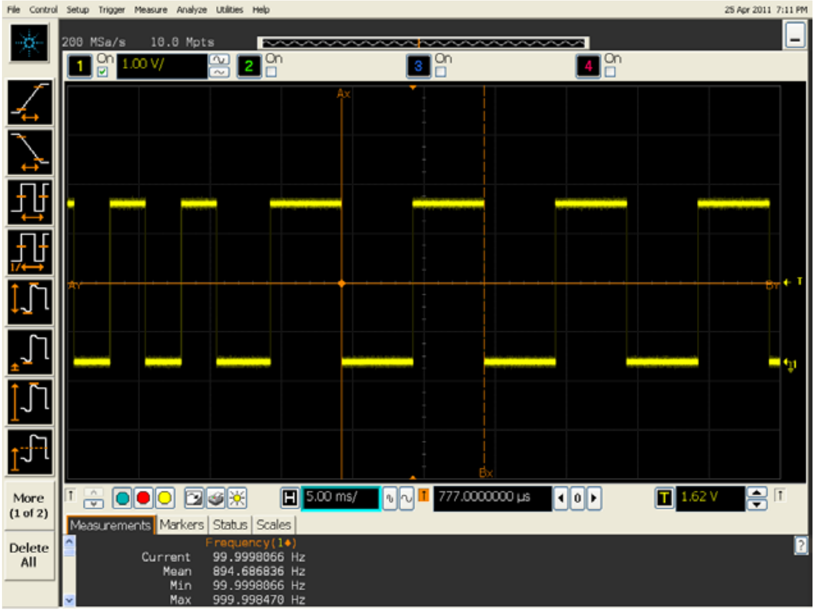


```
          Frequency(1♦)
Current  999.997832 Hz
   Mean  999.997832 Hz
    Min  999.997832 Hz
    Max  999.997832 Hz
```

Figure 13. Pulse 5 Steps

in succession and the frequency markers state it is 999.997Hz which is 1KHz.

This last waveform (Figure 15) shows the output slowing down from 1KHz at an acceleration rate of 100Hz per 40ms using the STOP() function. The waveform shows a 200Hz and 100Hz pulse next to each other which is the expected output.



```
           Frequency(1♦)
   Current   99.9998066 Hz
      Mean   894.686836 Hz
       Min   99.9998066 Hz
       Max   999.998470 Hz
```

Figure 14. Stop Function

In conclusion, DT6 code for the Stellaris is fully function and works better than the MSP430 that was provided by TI. DT6 coded Stellaris works better because the faster speeds of the Stellaris allow the PWM to generate faster frequencies then the MSP430. Both fabricated boards work and have been tested extensively, every pin on the MSP430F1612 and Stellaris LM3S9B95 has been turned high and low through code and checked at the output on the DRV8824EVM to ensure the hardware is correct. The MSP430 controlCARD was loaded with the code provided by TI and tested against the original DRV8824EVM to ensure they compared. Code was developed for the Stellaris and tested with an oscilloscope as seen above till DT6 was confident the code was fully functional.

Chapter 5

Summary and Conclusions

Texas Instruments has been working on making their evaluation modules more modular. They have many customers and most customers prefer different microcontrollers. Customers want their boards to be able to accept any type of TI microcontroller to control the driver IC that is that board, but TI does not want to design a new board for every customer. They want to be able to design one board to sell to all customers, and have the customer buy a controlCARD with the microcontroller of their liking. This will save Texas Instruments and their customers a lot of money, along with allowing Texas Instruments to produce a larger inventory because only one type of evaluation module will need to be produced. TI has tasked DT6 with creating one of these EVMs from a non-modular EVM and a controlCARD to fit the new EVM, while programming another controlCARD to be compatible with the new EVM. DT6 had the following three main tasks to accomplish.

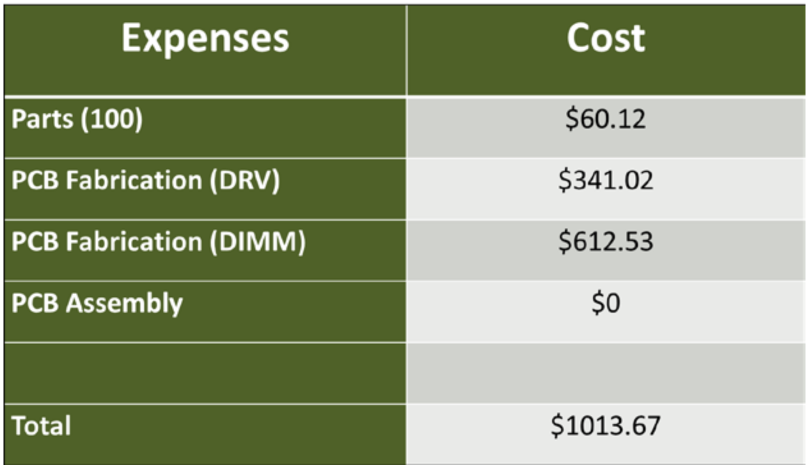
- Design a new DRV8824 board with a controlCARD slot.
- Design a MSP430 controlCARD that was compatible with this new board.
- Program a Stellaris controlCARD to control the DRV8824.

Task 1 was designing a new DRV8824 board with a controlCARD slot. The board was designed and sent out to the fabrication house roughly on time according to the schedule. Testing of this board was a success, although there was a minor setback. A power supply was hooked up in reverse and as a result the DRV8824 was burned out. Luckily, there were extra chips and DT6 found a person who had the expertise to solder the new chip onto the board. After this setback there were no other problems with the board.

Task 2 was designing a MSP430 controlCARD that was compatible with the new board. This board was designed and sent to a fabrication house a little later than expected because of complications with designing gold finger on a DIMM. The board ended up being shipped out three week later than hoped, but this setback was a not an issue because DT6 left enough time at the end to allocate for a fabrication delay. When the board returned it was programmed and connected to the DRV board, the two boards worked together seamlessly to control the stepper motor.

Task 3 was programming the Stellaris controlCARD to control the stepper motor. This task was a success. DT6 was able to get all aspects of the Windows application to work with the Stellaris controlCARD, with the exception of the DAC control since the Stellaris does not have a DAC on chip. DT 6 even went above and beyond by also getting the Stellaris to control the stepper motor with a webpage through the Ethernet port.

The budget for this project was $500. DT6 went over budget because of the fabrication requirements. This was due to the ECE shop only being able to produce 2 layered boards, while DT6 needed 4 layer boards. A breakdown of DT6 budget can be seen in Figure 16 below. The parts for this project were relatively cheap, since the whole board is basically surface mount each resistor and capacitor is around 7 cents. The price for parts was reduced by about $30 by receiving three TI parts as samples. The fabrication to get the DRV fabricatred was $573.44, this



Expenses	Cost
Parts (100)	$60.12
PCB Fabrication (DRV)	$341.02
PCB Fabrication (DIMM)	$612.53
PCB Assembly	$0
Total	$1013.67

Figure 16. Budget

was reduced after request for discounts to the price listed above. The controlCARD was $1010.45 this was the intial qoute DT6 received, it took much negoiation to reduce the price to what is seen in the table.

There are many future projects that could expand on what DT6 has done. A team could use the DRV board that DT6 created to design and program a controlCARD with a C2000 microcontroller. They could also do roughly the same project DT6 did, but with a different type of motor driver IC or a different type of motor while still using the Stellaris contorlCard. They could use and brushed or brushless DC motor for example. Combining some of these tasks together would create a good design project for a future group.

Overall the project came together more smoothly than anyone could have expect considering no members of DT6 had PCB design skills prior to this project. There were some minor setbacks, but DT6 accomplished all the tasks that were given to them and even went the extra mile working on controlling the stepper motor through the Ethernet port. DT6 considers this project a complete success, and was honored to work with Texas Instruments this semester.

Appendix 1

Leslie Thomas (LT)

LT completed many tasks this semester along with his main tasks he assigned himself this semester, which were creating the pinout for the DRV8824 and MSP430 DIMM along with create the Stellaris header file for the DRV8824. He started the semester off by understanding how the DRV8824EVM works, which pins were necessary to be PWM and GPIO. He then proceed understand how the LM3S9B96 controlCARD was wired up and which pins were could be shared between the new MSP430 controlCARD. He then compiled information with TJs research and together designed the pinout for the DRV8824EVM and MSP430 controlCARD that Pat used to make a PCB. LT provided support for Pat with researching footprints and checking all the connections he made before fabrication. LT found the fabricator to use, compiled the Gerber files, created the bill of materials, and then placed all the orders for the parts in the project.

For the software end, LT was the lead programmer behind creating the header files to be used with Windows GUI along with Pat and TJs Ethernet solutions, with contributions from there code added by TJ. LT created an interrupt for the Stellaris controlCARD that can count the number of pulses of a PWM signal. He also created the functions to move the motor a certain amount of steps, accelerate and decelerate the motor, invert the PWM signal, and also turn the GPIO high or low based off a boolean input. LT studied the MSP430 code and used this information to read the RS232 signal with the Stellaris controlCARD. He achieved a fully working solution for the Windows GUI with no changing of the initial GUI provided by TI and compiled this solution with the DRV8824 header file. LT overcame all the bugs in his code and has not been able to force the code to crash since his last revision.

Thomas Volinski (TJ)

TJ had two main tasks that needed to complete for our project to be successful. One was hardware based and one was software based.

For the hardware, he looked up everything he could on the MSP430F1612. He figured out which pins were necessary for controlling the motor. He figured out which pins needed to come off of the controlCARD and connect to the DRV8824. TJ analyzed a Stellaris and a C2000 controlCARD to determine where to place the ground and 5V pins on the controlCARD as well as where which GPIO signals should be connected to which controlCARD pins. LT and TJ designed the final pin layout based on TJs research.

A smaller task TJ accomplished was going through all of the MSP430 code Design Team 6 had to determine how they set up their code and to figure out what all of the important registers do. From there he looked into all documentation he could find about the Stellaris LM3S9B96 to find the registers that were equivalent to the MSP430 registers so that when Design Team 6 began coding the conversion of code would be much simpler.

TJ's main software task, was working with Pat on being able to control the Stellaris controlCARD through its Ethernet port. Pat and TJ created a webpage that has the same functionality as the windows application. TJ and Pat programmed the entire process of taking information from the webpage, sending it to the program on the microcontroller, and creating the required PWM signal as well as turning other signals on or off. Pat and TJ were successfully able to turn all of the control signals on and off. They also were able to start, stop and update the motors speed. TJ and Pat were able to make the motor pulse one step or any given number of steps. Finally, they were able to compute the RPMs of the motor. TJ also worked on the aesthetics of the webpage.

Patrick O’Hara

Through the entire project, Patrick worked on several aspects. The two main parts of the project that Patrick worked on was the DRV8824EVM and the MSP430 controlCARD. Patrick modified the supplied design files for the EVM to remove the MSP430 microcontroller from the board. After the MSP430 microcontroller was removed, a 100-pin DIMM socket was added and the pin out created by TJ and LT was used to make the appropriate connected on the EVM. After completion of the EVM printed circuit board, Patrick started work on the DIMM. A template to create the controlCARD was not supplied, so the design started from an AutoCAD drawing. Patrick created the AutoCAD drawing using references to the standard dimensions for 100-pin DIMM cards. He was then able to import this drawing into Altium Designer to create the board cutout. He worked in accord with LT to place the appropriate parts on the board. After the parts were placed, Patrick routed the circuit. Research was done by LT, TJ, and Patrick to determine the correctness for the “gold fingers” and the dimensions on the ControlCARD.

When both of the PCB designs were completed, Patrick started to work with TJ to programming the Stellaris microcontroller. Patrick worked with TJ to create the web interface to the microcontroller. The HTML interface sends commands from the web to the microcontroller to control the motor. The interface was modeled from the original Windows application provided by Texas Instruments. Patrick and TJ worked together to complete the web page, along with coding the protocol for communication between the web page and the microcontroller. They also worked together to code the appropriate response of the microcontroller. While working on all parts of the PCB design and the software, Patrick provided appropriate work to both TJ and LT to check his work. If either teammate noticed an error, the error was eradicated.

Kole Reece

Technical tasks included the design of DIMM control card and research on MSP430 code for porting to Stellaris platform. The design started with going over the controlCARD provided by the previous semesters design team and looking for templates to design the controlCARD. No DIMM standards or templates were found. A controlCARD template was created with AutoCAD and the controlCARD design was undertaken by other team members. The MSP430 communicates with the Windows GUI using universal asynchronous receiver/ transmitter (UART). In order to port the MSP430 code over to the selection on the GUI interface, it signals to the MSP430 to complete some operation. Work on porting the code involved identifying the operation code looking at the values that were sent to the MSP430, identifying the registers that were updated and the values that were sent. A simple example of this would be to configure the digital to analog converter. The expected OPCODE would be 5 DAC12_0CTL. Digital to analog converter would be configured based on serial buffer inputs 1 and 2 from the computer and the output voltage would be set using the DAC12_0DAT. All the registers that pertained to the application were “mapped out “using the above method.

Appendix II

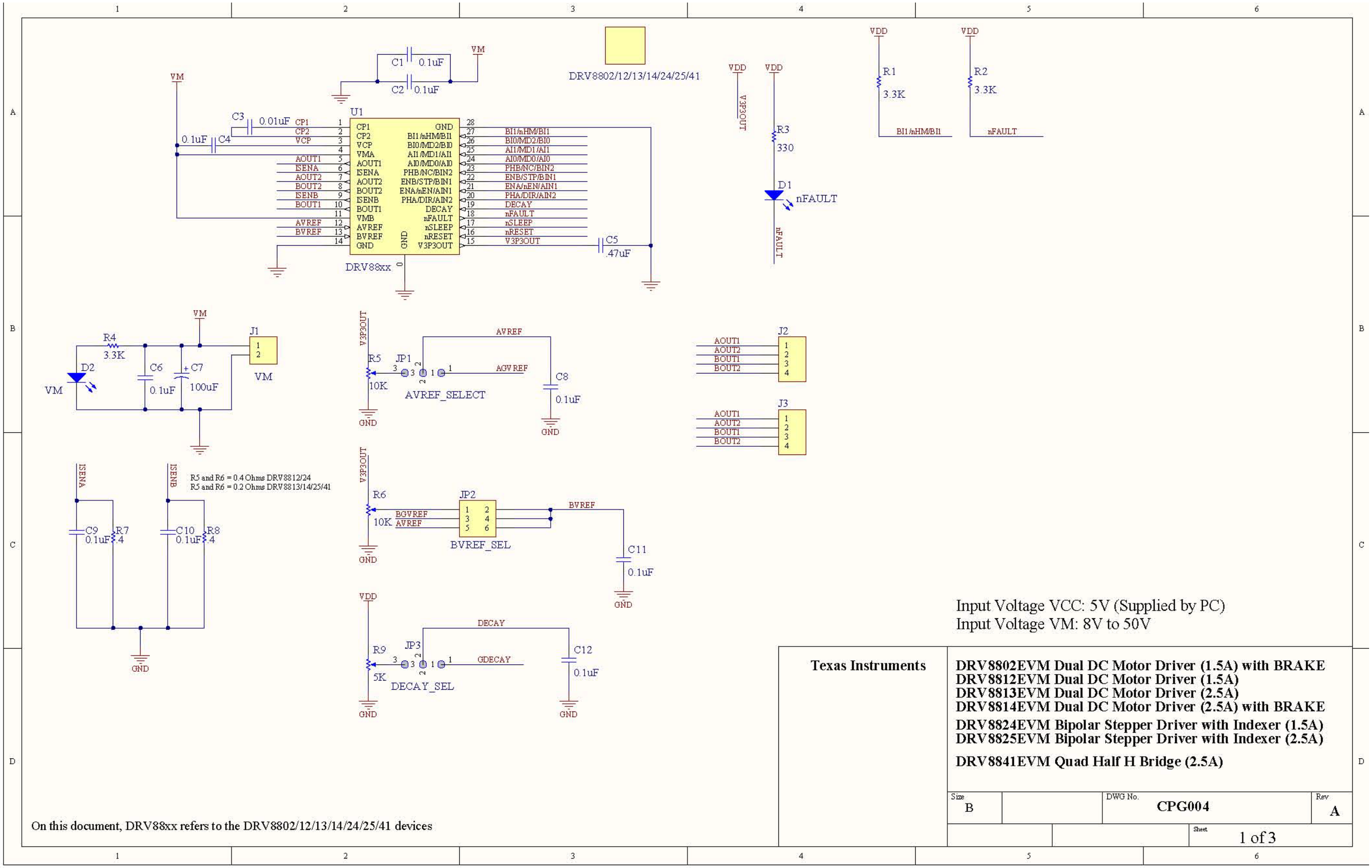
References:

"MSP430F1612 Datasheet." Texas Instruments.
http://focus.ti.com/lit/ds/symlink/msp430f1612.pdf

"DRV8824 Datasheet." Texas Instruments.
http://focus.ti.com/lit/ds/symlink/msp430f2013.pdf

"DRV88xx Schematic" Texas Instruments.
CC – LM3S9B95 Schematic Rev. 0. Texas Instruments. PDF.

"Getting Started with PCB Design", Altium.
http://www.altium.com/files/Altiumdesigner6/LearningGuides/TU0117%20Getting%20Started%20with%20PCB%20Design.PDF



Input Voltage VCC: 5V (Supplied by PC)
Input Voltage VM: 8V to 50V

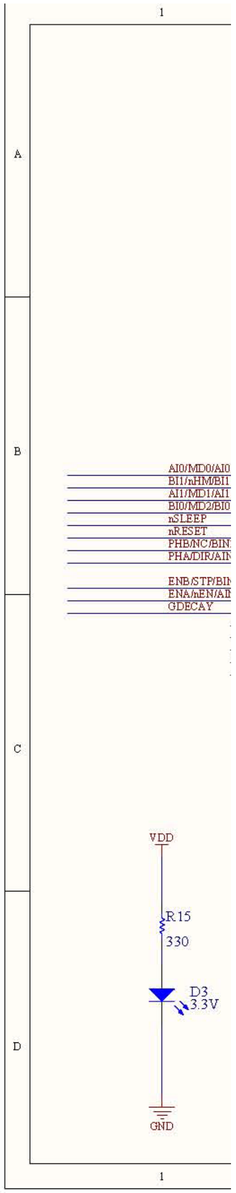
Texas Instruments

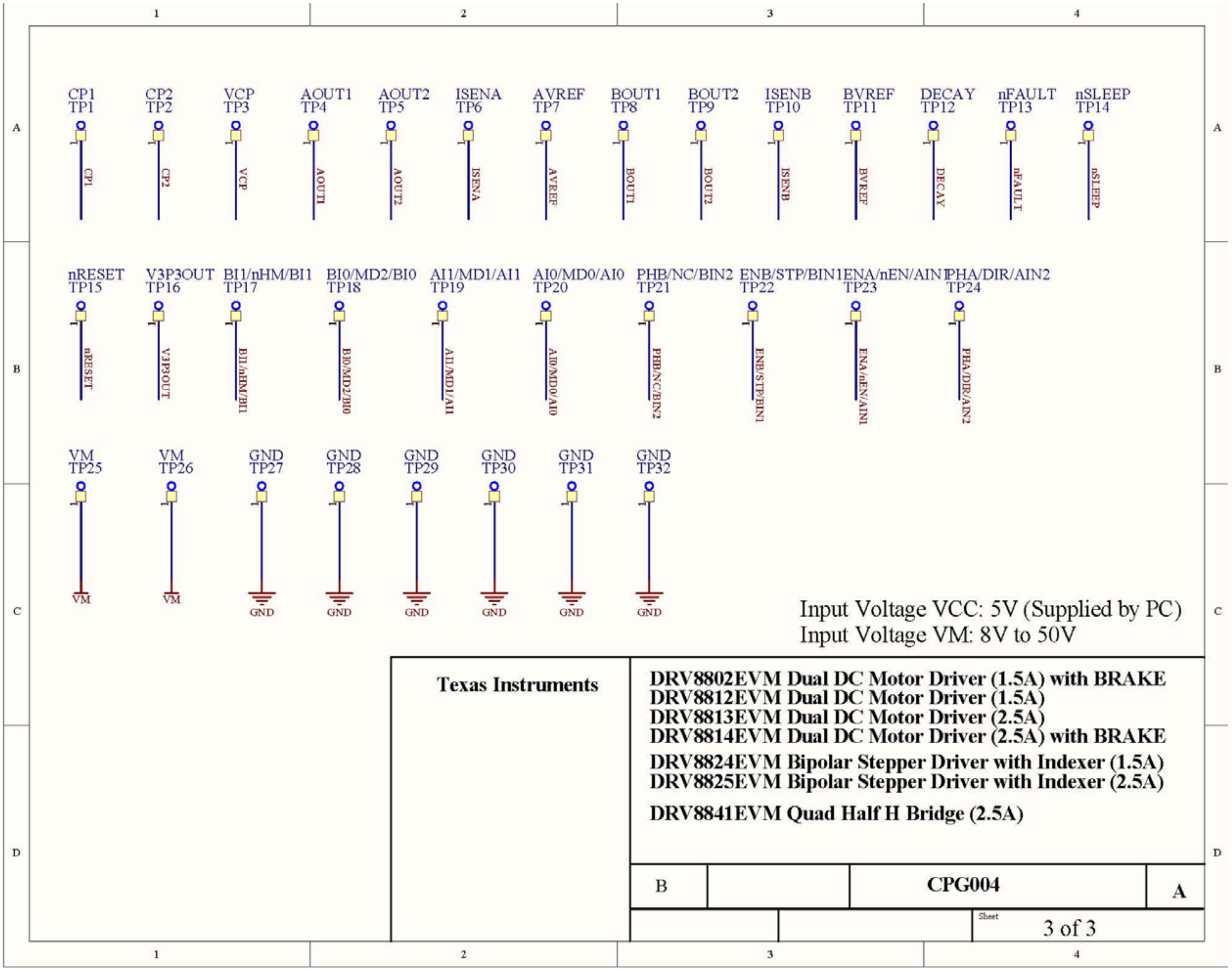
DRV8802EVM Dual DC Motor Driver (1.5A) with BRAKE
DRV8812EVM Dual DC Motor Driver (1.5A)
DRV8813EVM Dual DC Motor Driver (2.5A)
DRV8814EVM Dual DC Motor Driver (2.5A) with BRAKE
DRV8824EVM Bipolar Stepper Driver with Indexer (1.5A)
DRV8825EVM Bipolar Stepper Driver with Indexer (2.5A)
DRV8841EVM Quad Half H Bridge (2.5A)

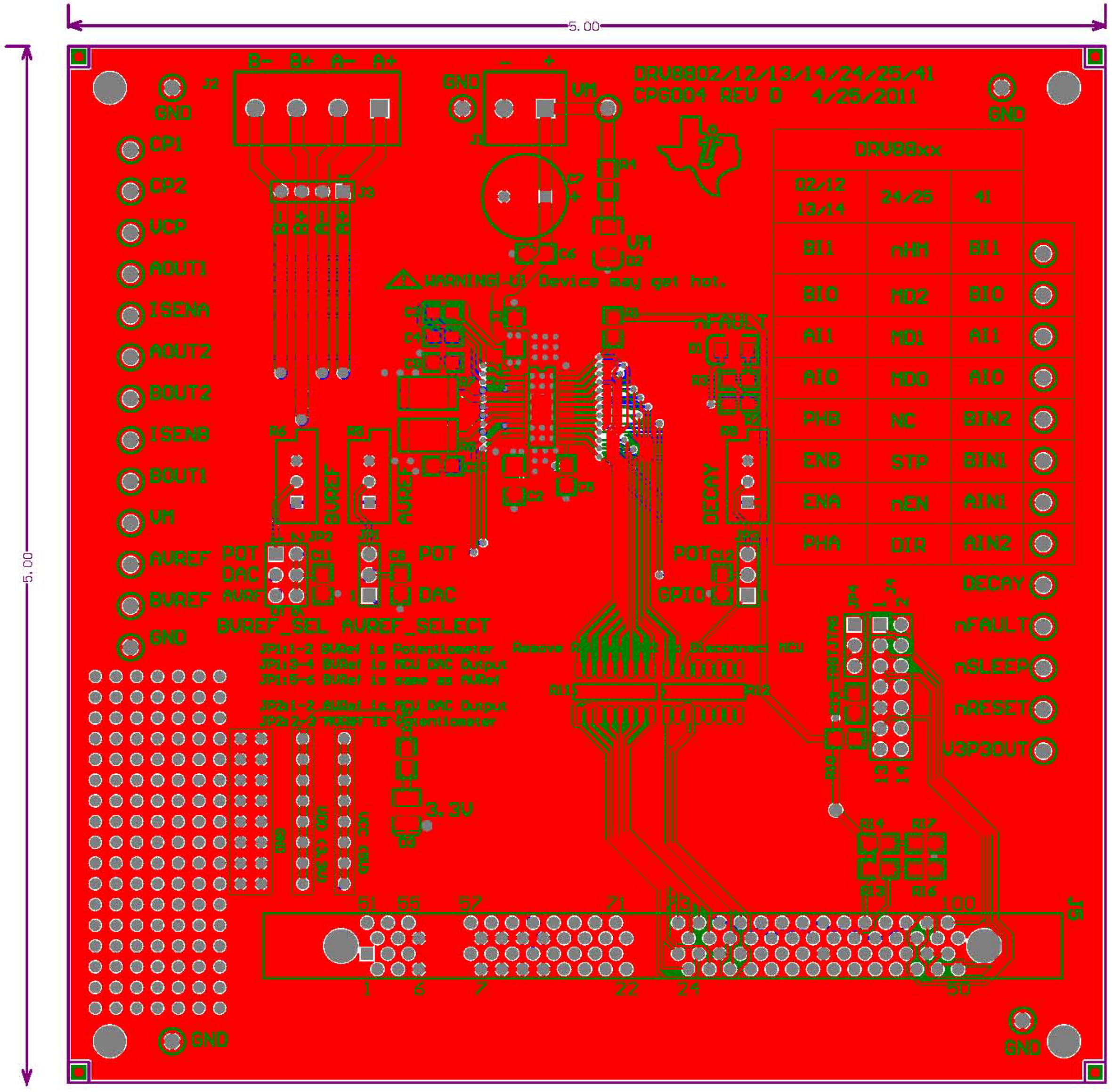
Size	DWG No.	Rev
B	CPG004	A

Sheet 1 of 3

On this document, DRV88xx refers to the DRV8802/12/13/14/24/25/41 devices


AI0/MD0/AI0
BI1/nHM/BI1
AI1/MD1/AI1
BI0/MD2/BI0
nSLEEP
nRESET
PHB/NC/BIN
PHA/DIR/AIN
ENB/STP/BIN
ENA/nEN/AIN
GDECAY
VDD
R15
330
D3
3.3V
GND


CP1
TP1
CP2
TP2
VCP
TP3
AOUT1
TP4
AOUT2
TP5
ISENA
TP6
AVREF
TP7
BOUT1
TP8
BOUT2
TP9
ISENB
TP10
BVREF
TP11
DECAY
TP12
nFAULT
TP13
nSLEEP
TP14
nRESET
TP15
V3P3OUT
TP16
BI1/nHM/BI1
TP17
BI0/MD2/BI0
TP18
AI1/MD1/AI1
TP19
AI0/MD0/AI0
TP20
PHB/NC/BIN2
TP21
ENB/STP/BIN1
TP22
ENA/nEN/AIN1
TP23
PHA/DIR/AIN2
TP24
VM
TP25
VM
TP26
GND
TP27
GND
TP28
GND
TP29
GND
TP30
GND
TP31
GND
TP32
VM
GND
Input Voltage VCC: 5V (Supplied by PC)
Input Voltage VM: 8V to 50V
Texas Instruments
DRV8802EVM Dual DC Motor Driver (1.5A) with BRAKE
DRV8812EVM Dual DC Motor Driver (1.5A)
DRV8813EVM Dual DC Motor Driver (2.5A)
DRV8814EVM Dual DC Motor Driver (2.5A) with BRAKE
DRV8824EVM Bipolar Stepper Driver with Indexer (1.5A)
DRV8825EVM Bipolar Stepper Driver with Indexer (2.5A)
DRV8841EVM Quad Half H Bridge (2.5A)
B
CPG004
A
Sheet
3 of 3
1
2
3
4
A
B
C
D



Layer Stack Up Detail for: CPG004 DRV88xxEVM

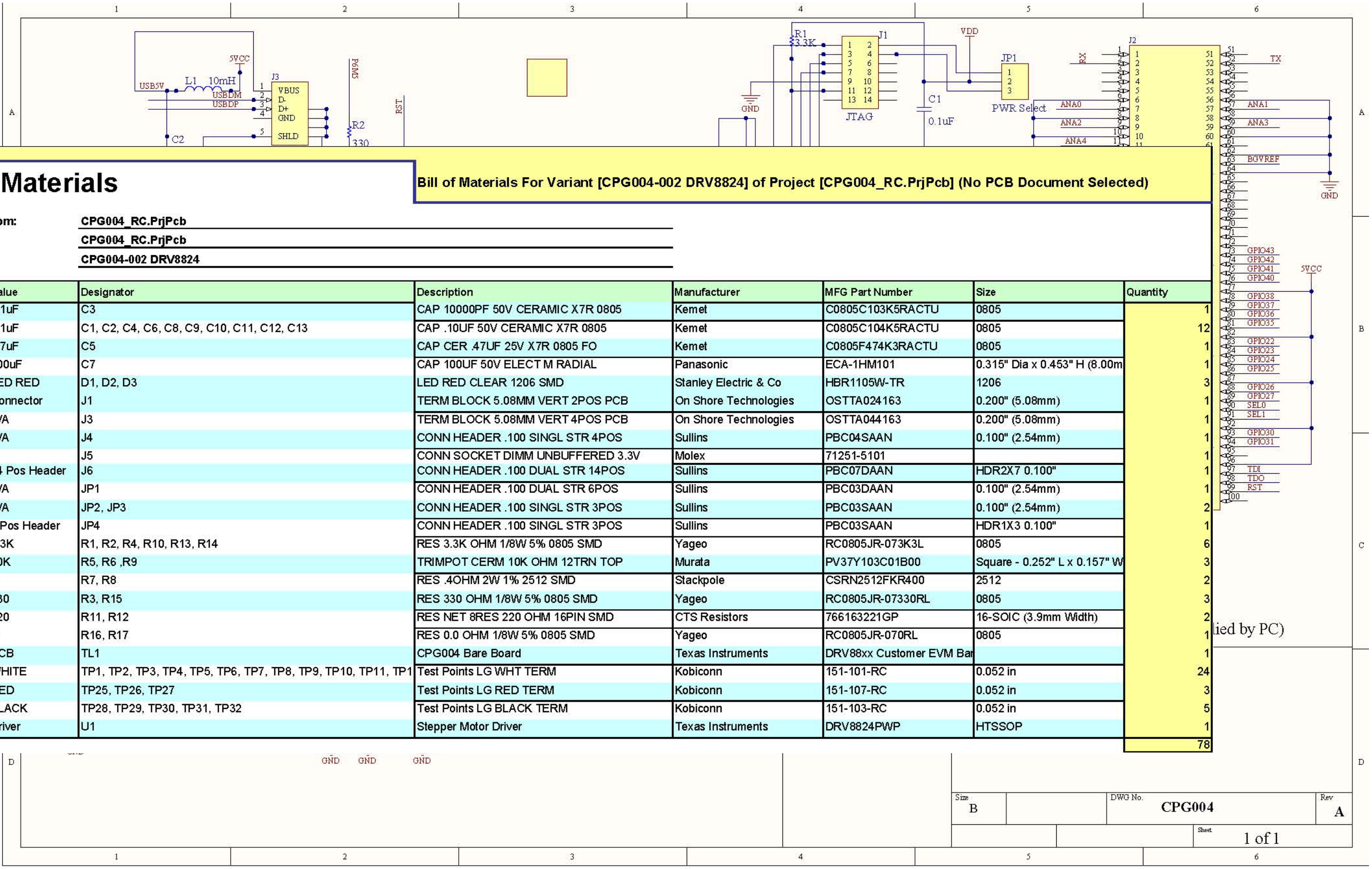
Layer Name

Top Overlay

Top Layer

MidLayer1 - KELVIN

BottomLayer

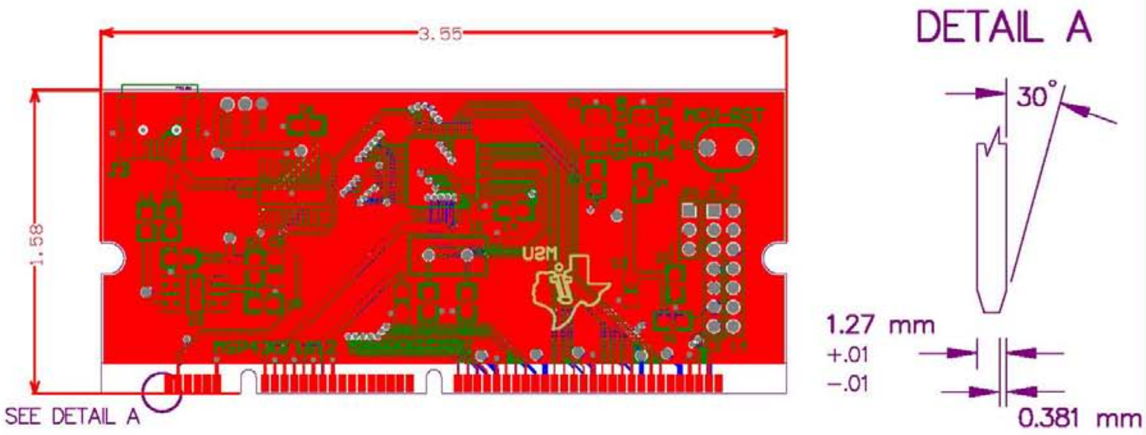


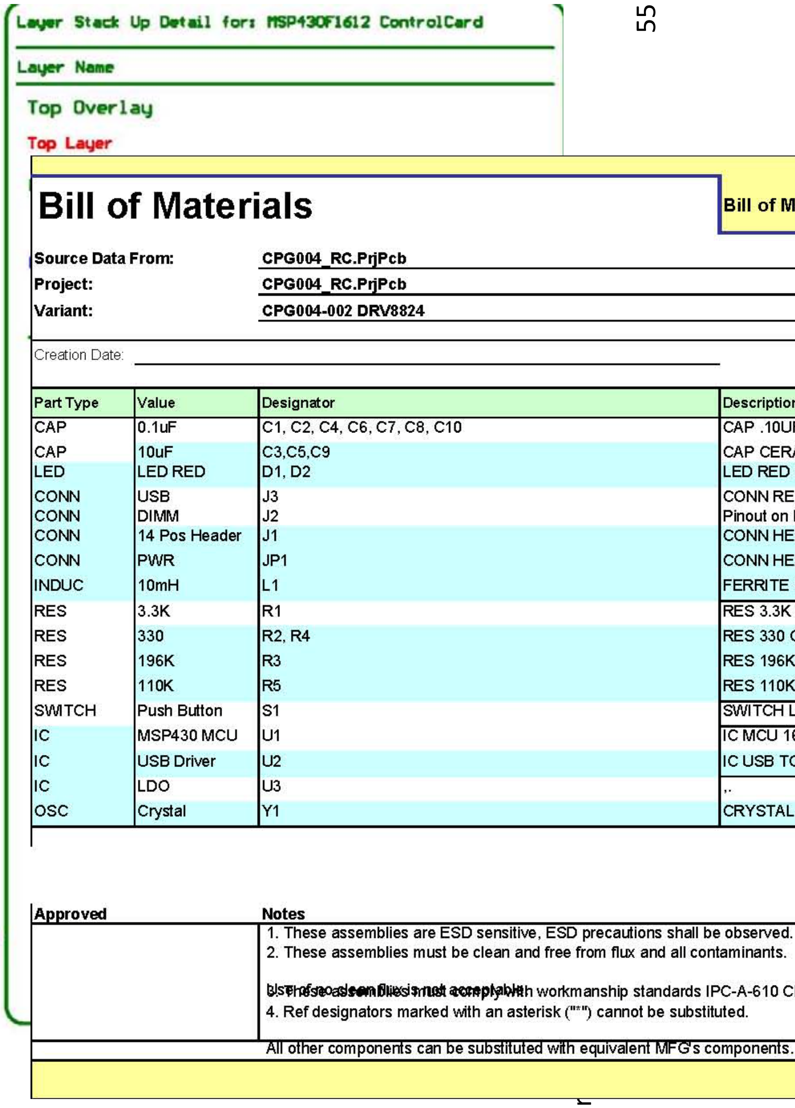
Bill of Materials

Bill of Materials For Variant [CPG004-002 DRV8824] of Project [CPG004_RC.PrjPcb] (No PCB Document Selected)

Source Data From: CPG004_RC.PrjPcb
Project: CPG004_RC.PrjPcb
Variant: CPG004-002 DRV8824

Part Type	Value	Designator	Description	Manufacturer	MFG Part Number	Size	Quantity
CAP	.01uF	C3	CAP 10000PF 50V CERAMIC X7R 0805	Kemet	C0805C103K5RACTU	0805	1
CAP	0.1uF	C1, C2, C4, C6, C8, C9, C10, C11, C12, C13	CAP .10UF 50V CERAMIC X7R 0805	Kemet	C0805C104K5RACTU	0805	12
CAP	.47uF	C5	CAP CER .47UF 25V X7R 0805 FO	Kemet	C0805F474K3RACTU	0805	1
CAP	100uF	C7	CAP 100UF 50V ELECT M RADIAL	Panasonic	ECA-1HM101	0.315" Dia x 0.453" H (8.00m	1
LED	LED RED	D1, D2, D3	LED RED CLEAR 1206 SMD	Stanley Electric & Co	HBR1105W-TR	1206	3
CONN	Connector	J1	TERM BLOCK 5.08MM VERT 2POS PCB	On Shore Technologies	OSTTA024163	0.200" (5.08mm)	1
CONN	N/A	J3	TERM BLOCK 5.08MM VERT 4POS PCB	On Shore Technologies	OSTTA044163	0.200" (5.08mm)	1
CONN	N/A	J4	CONN HEADER .100 SINGL STR 4POS	Sullins	PBC04SAAN	0.100" (2.54mm)	1
CONN		J5	CONN SOCKET DIMM UNBUFFERED 3.3V	Molex	71251-5101		1
CONN	14 Pos Header	J6	CONN HEADER .100 DUAL STR 14POS	Sullins	PBC07DAAN	HDR2X7 0.100"	1
CONN	N/A	JP1	CONN HEADER .100 DUAL STR 6POS	Sullins	PBC03DAAN	0.100" (2.54mm)	1
CONN	N/A	JP2, JP3	CONN HEADER .100 SINGL STR 3POS	Sullins	PBC03SAAN	0.100" (2.54mm)	2
CONN	3 Pos Header	JP4	CONN HEADER .100 SINGL STR 3POS	Sullins	PBC03SAAN	HDR1X3 0.100"	1
RES	3.3K	R1, R2, R4, R10, R13, R14	RES 3.3K OHM 1/8W 5% 0805 SMD	Yageo	RC0805JR-073K3L	0805	6
RES	10K	R5, R6 ,R9	TRIMPOT CERM 10K OHM 12TRN TOP	Murata	PV37Y103C01B00	Square - 0.252" L x 0.157" W	3
RES	.4	R7, R8	RES .4OHM 2W 1% 2512 SMD	Stackpole	CSRN2512FKR400	2512	2
RES	330	R3, R15	RES 330 OHM 1/8W 5% 0805 SMD	Yageo	RC0805JR-07330RL	0805	3
RES	220	R11, R12	RES NET 8RES 220 OHM 16PIN SMD	CTS Resistors	766163221GP	16-SOIC (3.9mm Width)	2
RES	0	R16, R17	RES 0.0 OHM 1/8W 5% 0805 SMD	Yageo	RC0805JR-070RL	0805	1
TL1	PCB	TL1	CPG004 Bare Board	Texas Instruments	DRV88xx Customer EVM Bar		1
TEST POINT	WHITE	TP1, TP2, TP3, TP4, TP5, TP6, TP7, TP8, TP9, TP10, TP11, TP1	Test Points LG WHT TERM	Kobiconn	151-101-RC	0.052 in	24
TEST POINT	RED	TP25, TP26, TP27	Test Points LG RED TERM	Kobiconn	151-107-RC	0.052 in	3
TEST POINT	BLACK	TP28, TP29, TP30, TP31, TP32	Test Points LG BLACK TERM	Kobiconn	151-103-RC	0.052 in	5
IC	Driver	U1	Stepper Motor Driver	Texas Instruments	DRV8824PWP	HTSSOP	1
							78




Layer Stack Up Detail for: MSP430F1612 ControlCard

Layer Name

Top Overlay

Top Layer

Bill of Materials

Bill of M

Source Data From: CPG004_RC.PrjPcb
Project: CPG004_RC.PrjPcb
Variant: CPG004-002 DRV8824

Creation Date: ______

Part Type	Value	Designator	Descriptior
CAP	0.1uF	C1, C2, C4, C6, C7, C8, C10	CAP .10UI
CAP	10uF	C3,C5,C9	CAP CER/
LED	LED RED	D1, D2	LED RED
CONN	USB	J3	CONN RE
CONN	DIMM	J2	Pinout on I
CONN	14 Pos Header	J1	CONN HE
CONN	PWR	JP1	CONN HE
INDUC	10mH	L1	FERRITE
RES	3.3K	R1	RES 3.3K
RES	330	R2, R4	RES 330 C
RES	196K	R3	RES 196K
RES	110K	R5	RES 110K
SWITCH	Push Button	S1	SWITCH L
IC	MSP430 MCU	U1	IC MCU 16
IC	USB Driver	U2	IC USB TC
IC	LDO	U3	,,
OSC	Crystal	Y1	CRYSTAL

Approved	Notes
	1. These assemblies are ESD sensitive, ESD precautions shall be observed. 2. These assemblies must be clean and free from flux and all contaminants. Use of no-clean flux is not acceptable. 3. These assemblies must comply with workmanship standards IPC-A-610 C 4. Ref designators marked with an asterisk ("*") cannot be substituted.

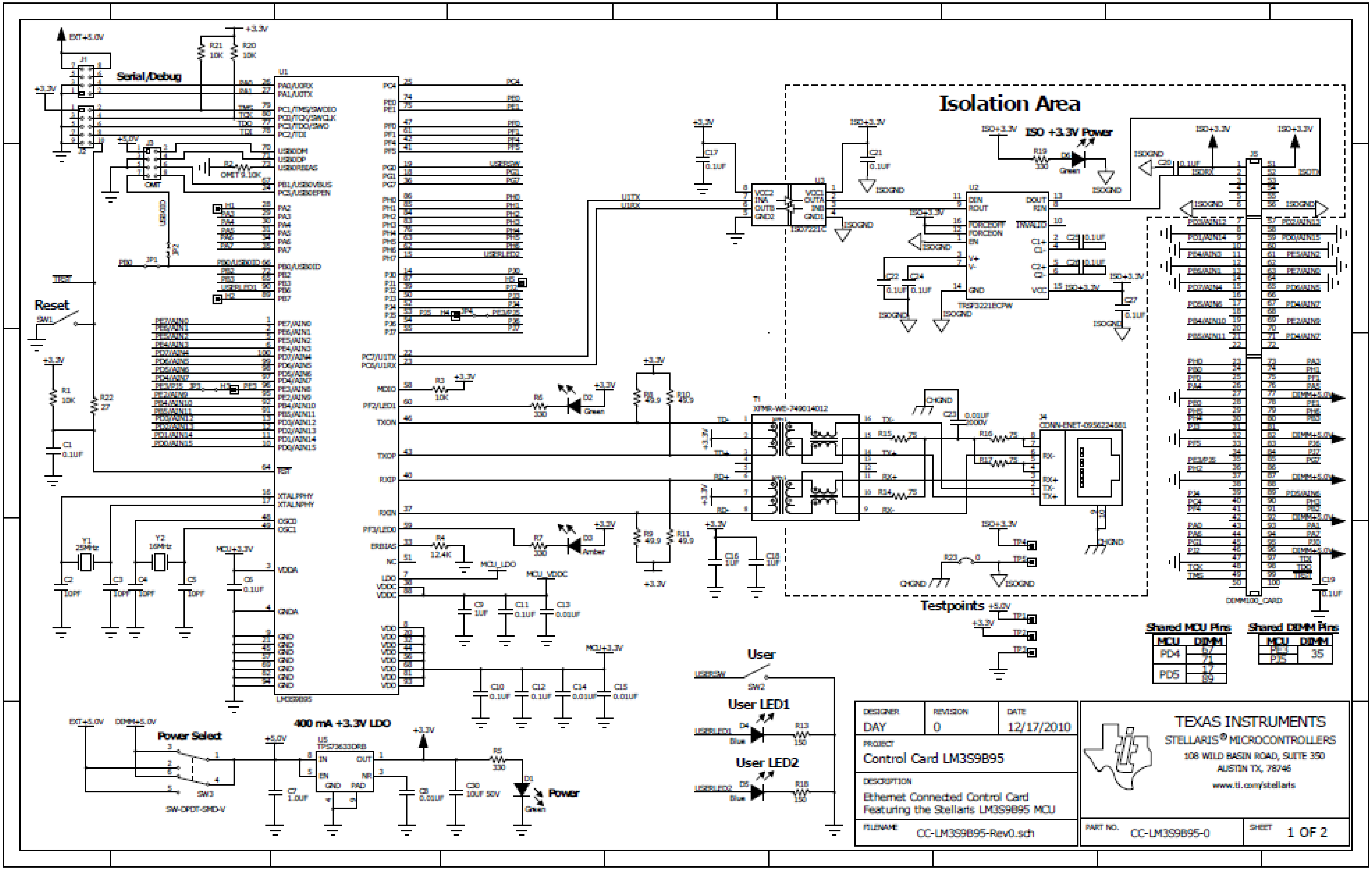
All other components can be substituted with equivalent MFG's components.

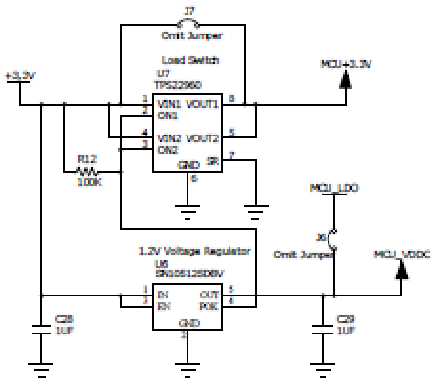
Pin Por

MSP430F1612 controlCARD Pinout

VDD	3.3V	1	51	3.3V	
32	RX	2	52	TX	
		3	53		
		4	54		
	GND	6	56	GND	
59	P6.0/A0	7	57	P6.1/A1	60
	GND	8	58	GND	
61	P6.2/A2	9	59	P6.3/A3	2
	GND	10	60	GND	
3	P6.4/A4	11	61	spare	
	GND	12	62	GND	
5	P6.6/A6	13	63	P6.7/A7	6
	GND	14	64	GND	
		15	65		
		16	66		
		17	67		
		18	68		
		19	69		
		20	70		
		21	71		
		22	72		
47	P5.3/UCLK1	23	73	P4.7/TBCLK	43
46	P5.2/SOMI1	24	74	P4.6/TB6	42
45	P5.1/SIMO1	25	75	P4.5/TB5	41
44	P5.0/STE1	26	76	P4.4/TB4	40
	GND	27	77	5V	
39	P4.3/TB3	28	78	P4.2/TB2	38
48	P5.4/MCLK	29	79	P4.1/TB1	37
34	P3.6/UTXD1	30	80	P4.0/TB0	36
	spare	31	81	P3.7/URXD1	35
	spare	32	82	5V	
12	P1.0	33	83	P2.2	22
13	P1.1	34	84	P2.3	23
14	P1.2	35	85	P2.4	24
15	P1.3	36	86	P2.5	25
	GND	37	87	5V	
16	P1.4	38	88	P2.6	26
17	P1.5	39	89	P2.7	27
18	P1.6	40	90	P3.0/SEL0	28
19	P1.7	41	91	P3.1/SEL1	29
	spare	42	92	5V	
20	P2.0	43	93	P3.2	30
21	P2.1	44	94	P3.3	31
	spare	45	95	spare	
	spare	46	96	5V	
	GND	47	97	TDI	55
57	TCK	48	98	TDO	54
56	TMS	49	99	TRESTn	58
		50	100		

🟥 PWR
🟩 GND
🟨 MSP430 to


Serial/Debug
Isolation Area
ISO +3.3V Power
Reset
Testpoints
User
User LED1
User LED2
Power Select
400 mA +3.3V LDO
Power
Shared MCU Pins
MCU DIMM
PD4 67 71
PD5 17 89
Shared DIMM Pins
MCU DIMM
PJ5 35
LM3S9B95
ISO7221C
TRSF3221ECPW
XFMR-WE-749014012
CONN-ENET-0956224881
DIMM100_CARD
TPS73633DRB
SW-DPDT-SMD-V
DESIGNER
DAY
REVISION
0
DATE
12/17/2010
PROJECT
Control Card LM3S9B95
DESCRIPTION
Ethernet Connected Control Card
Featuring the Stellaris LM3S9B95 MCU
FILENAME
CC-LM3S9B95-Rev0.sch
TEXAS INSTRUMENTS
STELLARIS® MICROCONTROLLERS
108 WILD BASIN ROAD, SUITE 350
AUSTIN TX, 78746
www.ti.com/stellaris
PART NO.
CC-LM3S9B95-0
SHEET
1 OF 2



Omit U6, U7, and R12 and populate J6 and J7 to accommodate future revisions.

DESIGNER	REVISION	DATE
DAY	0	12/17/2010

PROJECT
Control Card LM3S9B95

DESCRIPTION
LM3S9B95 Rev C3 Errata 2.4 work-around.

FILENAME CC-LM3S9B95-Rev0.sch

TEXAS INSTRUMENTS
STELLARIS® MICROCONTROLLERS
108 WILD BASIN ROAD, SUITE 350
AUSTIN TX, 78746
www.ti.com/stellaris

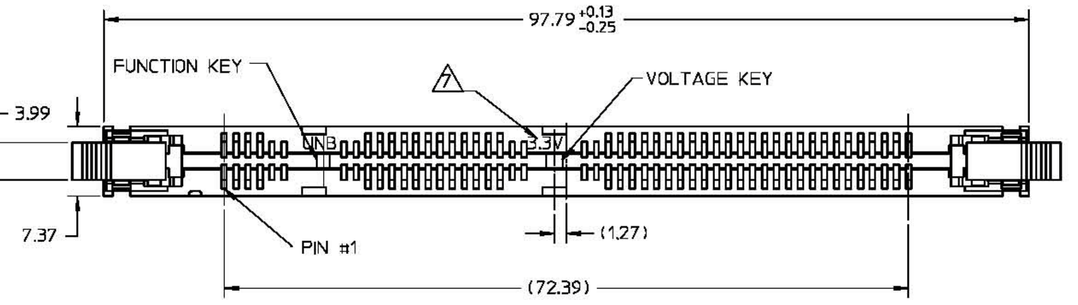
PART NO. CC-LM3S9B95-0

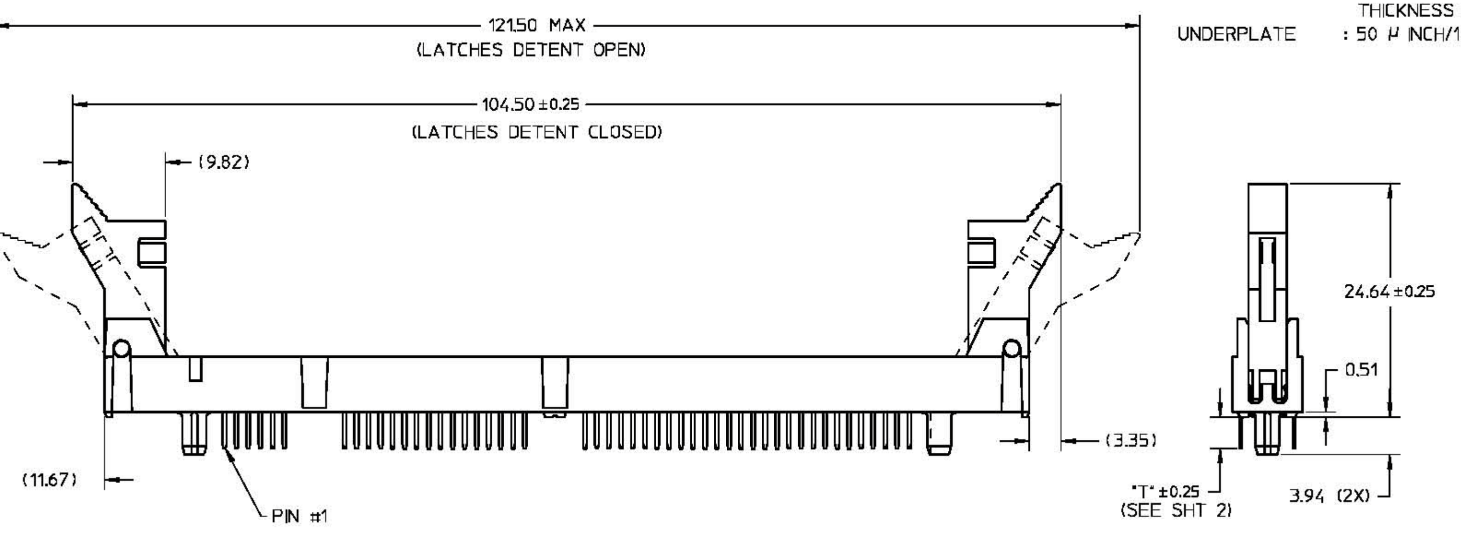
SHEET 2 OF 2

F28335 controlCARD [R1.0] DIMM100 pin-out

V33D-ISO	1	51	V33D-ISO
ISO-RX-RS232	2	52	ISO-TX-RS232
	3	53	
	4	54	
	5	55	
GND_ISO	6	56	GND_ISO
ADCIN-B0	7	57	ADCIN-A0
GND	8	58	GND
ADCIN-B1	9	59	ADCIN-A1
GND	10	60	GND
ADCIN-B2	11	61	ADCIN-A2
GND	12	62	GND
ADCIN-B3	13	63	ADCIN-A3
GND	14	64	GND
ADCIN-B4	15	65	ADCIN-A4
	16	66	
ADCIN-B5	17	67	ADCIN-A5
GPIO-58 / MCLKR-A / XD21 (EMIF)	18	68	GPIO-59 / MFSR-A / XD20 (EMIF)
ADCIN-B6	19	69	ADCIN-A6
GPIO-60 / MCLKR-B / XD19 (EMIF)	20	70	GPIO-61 / MFSR-B / XD18 (EMIF)
ADCIN-B7	21	71	ADCIN-A7
GPIO-62 / SCIRX-C / XD17 (EMIF)	22	72	GPIO-63 / SCITX-C / XD16 (EMIF)
GPIO-00 / EPWM-1A	23	73	GPIO-01 / EPWM-1B / MFSR-B
GPIO-02 / EPWM-2A	24	74	GPIO-03 / EPWM-2B / MCLKR-B
GPIO-04 / EPWM-3A	25	75	GPIO-05 / EPWM-3B / MFSR-A / ECAP-1
GPIO-06 / EPWM-4A / SYNCI / SYNCO	26	76	GPIO-07 / EPWM-4B / MCLKR-A / ECAP-2
GND	27	77	+5V in
GPIO-08 / EPWM-5A / CANTX-B / ADCSOC-A	28	78	GPIO-09 / EPWM-5B / SCITX-B / ECAP-3
GPIO-10 / EPWM-6A / CANRX-B / ADCSOC-B	29	79	GPIO-11 / EPWM-6B / SCIRX-B / ECAP-4
GPIO-48 / ECAP5 / XD31 (EMIF)	30	80	GPIO-49 / ECAP6 / XD30 (EMIF)
GPIO-84	31	81	GPIO-85
GPIO-86	32	82	+5V in
GPIO-12 / TZ-1 / CANTX-B / MDX-B	33	83	GPIO-13 / TZ-2 / CANRX-B / MDR-B
GPIO-15 / TZ-4 / SCIRX-B / MFSX-B	34	84	GPIO-14 / TZ-3 / SCITX-B / MCLKX-B
GPIO-24 / ECAP-1 / EQEPA-2 / MDX-B	35	85	GPIO-25 / ECAP-2 / EQEPB-2 / MDR-B
GPIO-26 / ECAP-3 / EQEPI-2 / MCLKX-B	36	86	GPIO-27 / ECAP-4 / EQEPS-2 / MFSX-B
GND	37	87	+5V in
GPIO-16 / SPISIMO-A / CANTX-B / TZ-5	38	88	GPIO-17 / SPISOMI-A / CANRX-B / TZ-6
GPIO-18 / SPICLK-A / SCITX-B	39	89	GPIO-19 / SPISTE-A / SCIRX-B
GPIO-20 / EQEPA-1 / MDX-A / CANTX-B	40	90	GPIO-21 / EQEPB-1 / MDR-A / CANRX-B
GPIO-22 / EQEPS-1 / MCLKX-A / SCITX-B	41	91	GPIO-23 / EQEPI-1 / MFSX-A / SCIRX-B
GPIO-87	42	92	+5V in
GPIO-28 / SCIRX-A / Resv / TZ5	43	93	GPIO-29 / SCITX-A / Resv / TZ6
GPIO-30 / CANRX-A	44	94	GPIO-31 / CANTX-A
GPIO-32 / I2CSDA / SYNCI / ADCSOCA	45	95	GPIO-33 / I2CSCL / SYNCO / ADCSOCB
GPIO-34 / ECAP1 / XREADY (EMIF)	46	96	+5V in
GND	47	97	TDI
TCK	48	98	TDO
TMS	49	99	TRSTn
EMU1	50	100	EMU0

NOTES:
1. CARD SLOT ACCEPTS 1.27±0.10 MM MODULE THICKNESS. (MEASURED OVER P.C. PADS).
   P.C. BOARD THICKNESS 1.57 MM +0.18/-0.13
2. ALL PEGS ARE INTERFERENCE FITS TO PCB UNLESS NOTED ON THE DWG.
3. REFER TO THE PRODUCT SPEC. PS-87630-004 FOR PERFORMANCE SPECIFICATIONS.
4. RECOMMENDED MODULE LAYOUT SHALL BE PER MO-161.
5. RECOMMENDED PLATING ON MODULE PADS:0.76 μ M MIN HARD GOLD (Au) OVER 2.0 μ M MIN NICKEL (Ni).
6. PRODUCT WILL HAVE DATE CODE STAMPED ON SIDE OF HOUSING.
7. △ VOLTAGE VALUE MARKED ACCORDING TO ITS RELATIVE KEY POSITION.
8. MATERIALS:
   HOUSING - LCP, GLASS FILLED, UL 94V-0, COLOR: BLACK
   TERMINAL - PHOSPHER BRONZE
   LATCHES - HIGH TEMP NYLON, GLASS FILLED, UL 94V-0, COLOR: BEIGE
9. PLATING:
   CONTACT AREA , GOLD(Au) : THICKNESS = 15 μ INCH/(0.38μ M)
   SOLDERTAILS : TIN (Sn) (SEE PART NUMBER TABLE)
   THICKNESS = 75μ INCH/(1.91μ M)
   UNDERPLATE : 50 μ INCH/1.27μ M MIN NICKEL (Ni).





D	OBS TIN/LEAD P/N EC NO: S2006-0734 DRWN:MLONG 2006/02/27 CH'KD:IHO 2006/02/28 APPR:GGLEE 2006/03/01	QUALITY SYMBOLS ▼=0 ▽=0
REV	DESCRIPTION	

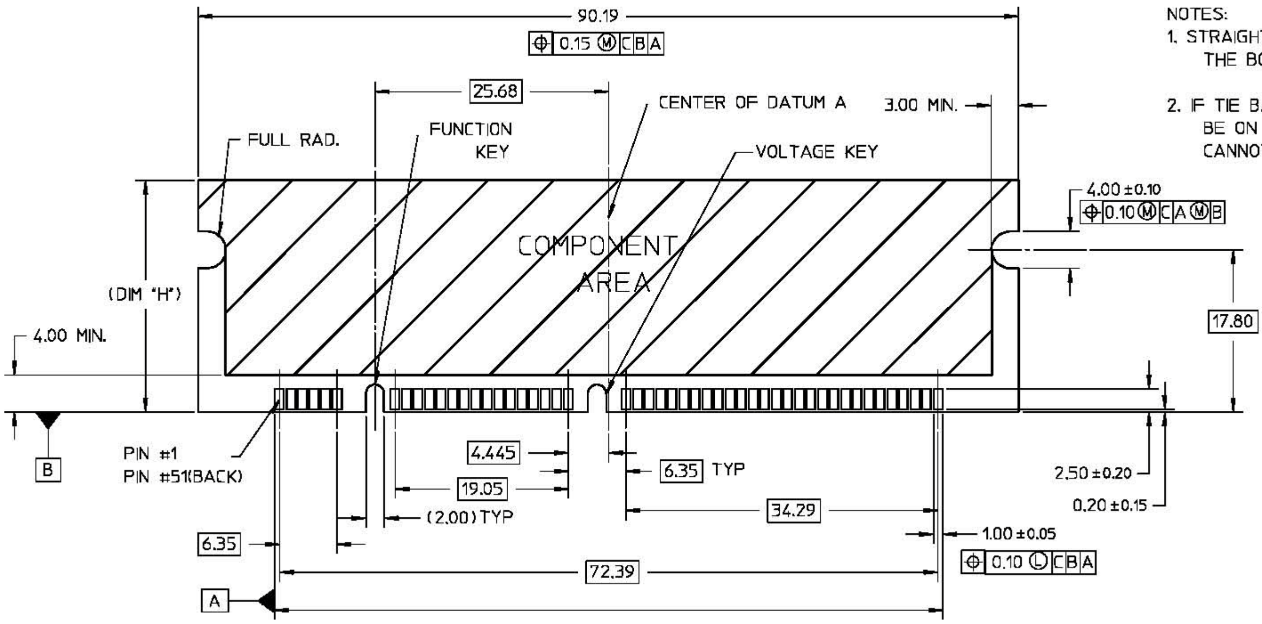
GENERAL TOLERANCES (UNLESS SPECIFIED)	mm	INCH
4 PLACES	±---	±---
3 PLACES	±---	±---
2 PLACES	±0.15	±---
1 PLACE	±---	±---
ANGULAR ± 1 °		
DRAFT WHERE APPLICABLE MUST REMAIN WITHIN DIMENSIONS		

DIMENSION STYLE	SCALE	DESIGN UNITS	
MM ONLY	NTS	METRIC	THIRD ANGLE PROJECTION

DRAWN BY	DATE	TITLE
CLCHUA	2000/04/12	DIMM, 1.27MM PITCH 100 CKTS, MULTI KEY
CHECKED BY	DATE	
SQUEK	2000/04/20	
APPROVED BY	DATE	
SKTOH	2000/04/20	molex MOLEX INCORPORATED
MATERIAL NO.	DOCUMENT NO.	SHEET NO.
SEE TABLE	SD-87630-001	1 OF 3

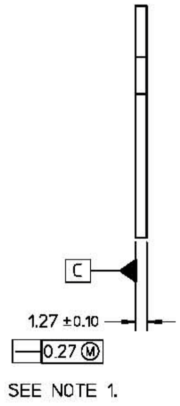
SIZE A3 THIS DRAWING CONTAINS INFORMATION THAT IS PROPRIETARY TO MOLEX INCORPORATED AND SHOULD NOT BE USED WITHOUT WRITTEN PERMISSION

me_A3_P_AM_T
0 2004/06/28



(OUTSIDE EDGE OF END PADS)

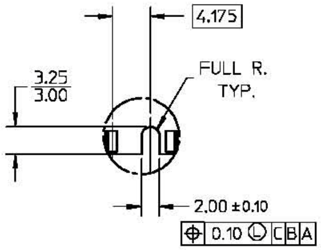
RECOMMENDED MODULE LAYOUT
100 CKT 3.3V UNBUFFERED



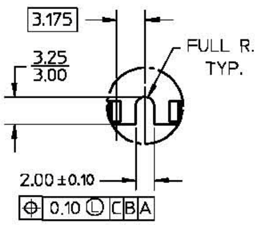
SEE NOTE 1.

NOTES:

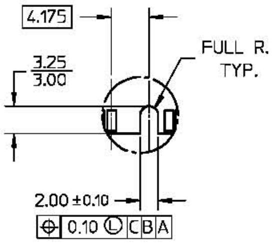
1. STRAIGHTNESS OF MODULE APPLIES TO THE AREA FROM THE BOTTOM OF THE CARD UP 4.00 MM.
2. IF TIE BARS ARE ATTACHED TO PADS, THE TIE BAR SHOULD BE ON THE INTERNAL LAYER, SO THAT THE REMNANT CANNOT CAUSE DAMAGE TO THE CONTACTS.



FUNCTION KEY
(OFFSET RIGHT)



VOLTAGE KEY
CENTER (3.3V)



VOLTAGE KEY
OFFSET RIGHT (2.5V)

REV	DESCRIPTION
D	OBS TIN/LEAD P/N EC NO: S2006-0734 DRWN:MLONG 2006/02/27 CH'KD:IHO 2006/02/28 APPR:GGLEE 2006/03/01

QUALITY SYMBOLS
▼=0
Ⓒ=0

GENERAL TOLERANCES (UNLESS SPECIFIED)	mm	INCH
4 PLACES	± ---	± ---
3 PLACES	± ---	± ---
2 PLACES	± 0.15	± ---
1 PLACE	± ---	± ---
ANGULAR ± 1 °		

DRAFT WHERE APPLICABLE MUST REMAIN WITHIN DIMENSIONS

DIMENSION STYLE	SCALE	DESIGN UNITS	
MM ONLY	NTS	METRIC	◎ ◁ THIRD ANGLE PROJECTION

DRAWN BY	DATE
CLCHUA	2000/04/12
CHECKED BY	DATE
SQUEK	2000/04/20
APPROVED BY	DATE
SKTOH	2000/04/20

TITLE
DIMM, 1.27MM PITCH
100 CKTS, MULTI KEY

molex MOLEX INCORPORATED

MATERIAL NO. SEE TABLE

DOCUMENT NO. SD-87630-001

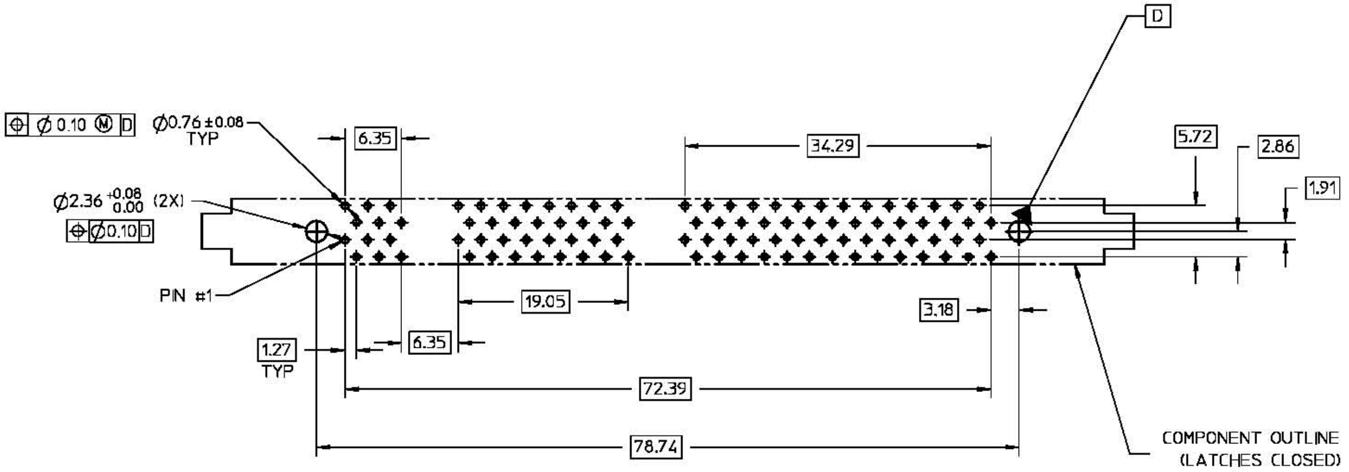
SHEET NO. 3 OF 3

SIZE A3

THIS DRAWING CONTAINS INFORMATION THAT IS PROPRIETARY TO MOLEX INCORPORATED AND SHOULD NOT BE USED WITHOUT WRITTEN PERMISSION

tb_frame_A3_P_AM_T
Rev. D 2004/06/28

PART NUMBER	PACKAGING TYPE	FUNCTION KEY	VOLTAGE KEY	(DIM"H")	DIM "T"
87630-1001	TRAY	OFFSET RIGHT	CENTER (3.3V)	(25.40)	3.23
87630-0005			RIGHT (2.5V)	(30.50)	
87630-1011	TUBE		CENTER (3.3V)	(25.40)	
87630-1012			RIGHT (2.5V)	(30.50)	



RECOMMENDED P.C.BOARD HOLE PATTERN
(CONNECTOR SIDE)

REV	DESCRIPTION
D	OBS TIN/LEAD P/N EC NO: S2006-0734 DRWN:MLONG 2006/02/27 CH'KD:IHD 2006/02/28 APPR:GGLEE 2006/03/01

QUALITY SYMBOLS
▼=0
▽=0

GENERAL TOLERANCES (UNLESS SPECIFIED)

	mm	INCH
4 PLACES	±---	±---
3 PLACES	±---	±---
2 PLACES	±0.15	±---
1 PLACE	±---	±---
ANGULAR ± 1 °		

DRAFT WHERE APPLICABLE MUST REMAIN WITHIN DIMENSIONS

DIMENSION STYLE	SCALE	DESIGN UNITS	
MM ONLY	NTS	METRIC	THIRD ANGLE PROJECTION

	DATE
DRAWN BY CLCHUA	2000/04/12
CHECKED BY SQUEK	2000/04/20
APPROVED BY SKTOH	2000/04/20

TITLE DIMM, 1.27MM PITCH 100 CKTS, MULTI KEY

MOLEX INCORPORATED

MATERIAL NO. SEE TABLE

DOCUMENT NO. SD-87630-001

SHEET NO. 2 OF 3

SIZE A3

THIS DRAWING CONTAINS INFORMATION THAT IS PROPRIETARY TO MOLEX INCORPORATED AND SHOULD NOT BE USED WITHOUT WRITTEN PERMISSION

lb_frame_A3_P_AM_T
Rev. D 2004/06/28

```
//***************************************************************************
//
// DRV8824.c - Prototypes for Stellaris Motor Control.
// Author: Leslie Thomas (LT)
// co-authors: TJ Volinski & Pat O'Hara
//
// Copyright (c) 2011
//
//
// 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.
// LMI SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR
// CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
//
//***************************************************************************



#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_pwm.h"
#include "inc/hw_sysctl.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/interrupt.h"
#include "driverlib/pwm.h"
#include "driverlib/gpio.h"
#include "driverlib/rom.h"
#include "driverlib/sysctl.h"
#include "driverlib/DRV8824.h"



//*****************************************************************************
//
// Initalize GPIO and PWM port/s
//
//*****************************************************************************
unsigned int uiCount;
unsigned int uiStepCount;
unsigned long ulScale;

void PWM2Int(void)
{
    PWMGenIntClear(STEP_PWM, STEP_GEN, PWM_INT_CNT_ZERO);

    uiCount++;

    if(uiCount >= uiStepCount)
    {
      PWMOutputState(STEP_PWM, STEP_BIT , false);
      PWMIntDisable(STEP_PWM,STEP_INT);
      PWMGenIntTrigDisable(STEP_PWM, STEP_GEN, PWM_INT_CNT_ZERO);
      uiCount = 0;
      PWMGenConfigure(STEP_PWM, STEP_GEN,
                          PWM_GEN_MODE_UP_DOWN | PWM_GEN_MODE_NO_SYNC);
      Frequency(0);
    }
}

void DRV8824_init(void)
{
    //
    // Enable the peripherals used by this example.
    //

    // PWM Enabled for STEP
    SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM);
    SysCtlPWMClockSet(SYSCTL_PWMDIV_16);

    // GPIO for everything else
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOH);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);

    // Step
    GPIOPinConfigure(GPIO_PE0_PWM4);
    GPIOPinTypePWM(STEP_BASE, STEP_PIN);

    // Direction
    GPIOPinTypeGPIOOutput(DIR_BASE, DIR_PIN);
    GPIOPinWrite(DIR_BASE, DIR_PIN, 0);

    // nSLEEP
    GPIOPinTypeGPIOOutput(SLEEP_BASE, SLEEP_PIN);
    GPIOPinWrite(SLEEP_BASE, SLEEP_PIN, 0);

    // RESET
    GPIOPinTypeGPIOOutput(RESET_BASE, RESET_PIN);
    GPIOPinWrite(RESET_BASE, RESET_PIN, 0);

    // nENABLE
    GPIOPinTypeGPIOOutput(ENABLE_BASE, ENABLE_PIN);
    GPIOPinWrite(ENABLE_BASE, ENABLE_PIN, 0);

    // Mode0
    GPIOPinTypeGPIOOutput(MODE_0_BASE, MODE_0_PIN);
    GPIOPinWrite(MODE_0_BASE, MODE_0_PIN, 0);

    // Mode1
    GPIOPinTypeGPIOOutput(MODE_1_BASE, MODE_1_PIN);
    GPIOPinWrite(MODE_1_BASE, MODE_1_PIN, 0);

    // Mode2
    GPIOPinTypeGPIOOutput(MODE_2_BASE, MODE_2_PIN);
    GPIOPinWrite(MODE_2_BASE, MODE_2_PIN, 0);

    // NC
    GPIOPinTypeGPIOOutput(NC_BASE, NC_PIN);
    GPIOPinWrite(NC_BASE, NC_PIN, 0);

    // Decay
    GPIOPinTypeGPIOOutput(DECAY_BASE, DECAY_PIN);
    GPIOPinWrite(DECAY_BASE, DECAY_PIN, 0);


    //
    // Configure PWM
    //

    PWMGenConfigure(STEP_PWM, STEP_GEN,
                        PWM_GEN_MODE_UP_DOWN | PWM_GEN_MODE_NO_SYNC);

    //
    // Enable the PWM generator.
    //
    PWMGenEnable(STEP_PWM, STEP_GEN);
}


//*****************************************************************************
//
// This function sets the PWM frequency given to ulFreq at a 50% duty cycle
// This function changes the PWM clock divide to gain better resolution at
// higher speeds
//*****************************************************************************

void Frequency( signed int ulFreq)
{
  if(ulFreq>40000)
   {
     SysCtlPWMClockSet(SYSCTL_PWMDIV_2);
     ulScale = 2;
   }
   else if(ulFreq>20000)
   {
     SysCtlPWMClockSet(SYSCTL_PWMDIV_4);
     ulScale = 4;
   }
   else if(ulFreq > 10000)
   {
     SysCtlPWMClockSet(SYSCTL_PWMDIV_8);
     ulScale = 8;
   }
   else
   {
     SysCtlPWMClockSet(SYSCTL_PWMDIV_16);
     ulScale = 16;
   }

   unsigned long ulPeriod = SysCtlClockGet() / (ulFreq * ulScale) ;
   PWMGenPeriodSet(STEP_PWM, STEP_GEN, ulPeriod);
   PWMPulseWidthSet(STEP_PWM, STEP_OUT, ulPeriod / 2);

}


//*****************************************************************************
//
```

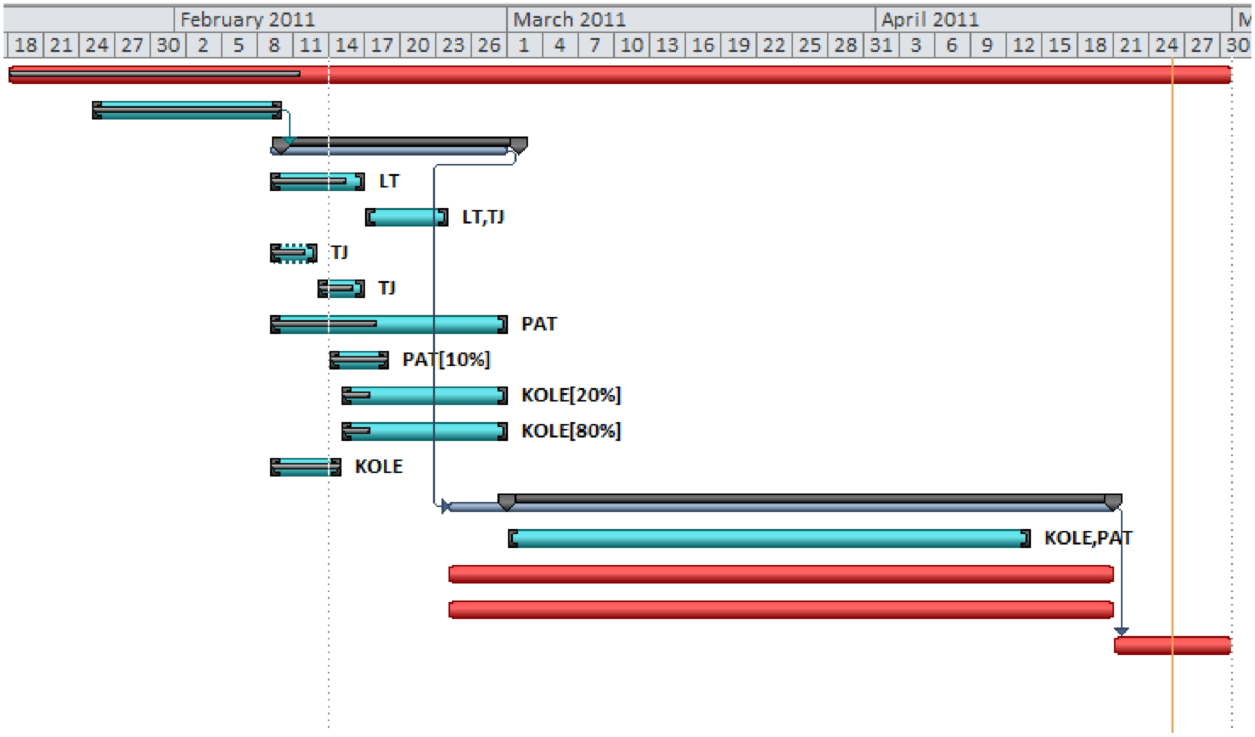
```
// Accelarate the motor to new frequency starting at the past frequency
// With the Acceleration Rate and Acceleration Time Base provided
//
//*****************************************************************************

void ACCEL(signed long pastFreq, signed long newFreq, unsigned long AccelRate,
unsigned long AccelTimeBase  )
{
  PWMOutputState(PWM_BASE, PWM_OUT_4_BIT , true);

  while( newFreq > pastFreq )
  {
      SysCtlDelay(( (SysCtlClockGet()/3) / (1000/AccelTimeBase)) );
      pastFreq = pastFreq +  AccelRate;
      Frequency(pastFreq);
  }
  Frequency(newFreq);
}

//*****************************************************************************
//
// Decelarate the motor to new frequency starting at the past frequency
// With the Acceleration Rate and Acceleration Time Base provided
//
//*****************************************************************************

void DECEL( signed long pastFreq, signed long newFreq, unsigned long AccelRate,
unsigned long AccelTimeBase )
{

    while( newFreq < pastFreq )
    {

      SysCtlDelay(( (SysCtlClockGet()/3) / (1000/AccelTimeBase)) );
      pastFreq = pastFreq -  AccelRate ;
      Frequency(pastFreq);


    }
    if( newFreq < 62 )
    {
      PWMOutputState(STEP_PWM, STEP_BIT , false);
      Frequency(0);
    }
    else
    {
      Frequency(newFreq);
    }

}


//*****************************************************************************
//
// Initiates number of steps and sets the frequency
// Deadband of 10ns is used to allow the interrupt to see the first pulse
//
```

```
//
//*****************************************************************************


void STEP_ON(unsigned int cnt,signed int Freq)
{
        uiStepCount = cnt;
        PWMGenConfigure(STEP_PWM, STEP_GEN,
                  PWM_GEN_MODE_DOWN | PWM_GEN_MODE_DB_NO_SYNC);
        PWMDeadBandEnable(STEP_PWM, STEP_GEN,10,0);
        PWMGenIntClear(STEP_PWM, STEP_GEN, PWM_INT_CNT_ZERO);
        PWMGenIntTrigEnable(STEP_PWM, STEP_GEN, PWM_INT_CNT_ZERO);
        Frequency(Freq);
        IntEnable(INT_PWM2 );
        PWMIntEnable(STEP_PWM,STEP_INT);
        PWMOutputState(STEP_PWM, STEP_BIT , true);
        PWMGenEnable(STEP_PWM, STEP_GEN);

}

//*****************************************************************************
//
// Inverts the STEP signal
//
//*****************************************************************************


void STEP_HL(tBoolean output)
{
  PWMOutputInvert(STEP_PWM, STEP_BIT , output);
}


//*****************************************************************************
//
// GPIO outputs
//
//*****************************************************************************


void Direction(tBoolean output)
{
    GPIOPinWrite(DIR_BASE, DIR_PIN, output ? DIR_PIN : 0);
}

void nSleep(tBoolean output)
{
    GPIOPinWrite(SLEEP_BASE, SLEEP_PIN, output ? SLEEP_PIN : 0);
}

void nReset(tBoolean output)
{
    GPIOPinWrite(RESET_BASE, RESET_PIN, output ? RESET_PIN : 0);
}

void nEnable(tBoolean output)
{
    GPIOPinWrite(ENABLE_BASE, ENABLE_PIN, output ? ENABLE_PIN : 0);
}

void nDecay(tBoolean output)
{
    GPIOPinWrite(DECAY_BASE, DECAY_PIN, output ? DECAY_PIN : 0);
}

void Mode_0(tBoolean output)
{
    GPIOPinWrite(MODE_0_BASE, MODE_0_PIN, output ? MODE_0_PIN : 0);
}

void Mode_1(tBoolean output)
{
    GPIOPinWrite(MODE_1_BASE, MODE_1_PIN, output ? MODE_1_PIN : 0);
}

void Mode_2(tBoolean output)
{
    GPIOPinWrite(MODE_2_BASE, MODE_2_PIN, output ? MODE_2_PIN : 0);
}

void NC(tBoolean output)
{
    GPIOPinWrite(NC_BASE, NC_PIN, output ? NC_PIN : 0);
}


//*****************************************************************************
//
// Checks if GPIO input is on
//
//*****************************************************************************

tBoolean is_Direction_on(void)
{
    return(GPIOPinRead(DIR_BASE, DIR_PIN));
}

tBoolean is_nEnable_on(void)
{
    return(GPIOPinRead(ENABLE_BASE, ENABLE_PIN));
}

tBoolean is_nSleep_on(void)
{
    return(GPIOPinRead(SLEEP_BASE, SLEEP_PIN));
}

tBoolean is_nReset_on(void)
{
    return(GPIOPinRead(RESET_BASE, RESET_PIN));
}

tBoolean is_Mode_0_on(void)
{
    return(GPIOPinRead(MODE_0_BASE, MODE_0_PIN));
}

tBoolean is_Mode_1_on(void)
{
    return(GPIOPinRead(MODE_1_BASE, MODE_1_PIN));
}

tBoolean is_Mode_2_on(void)
{
    return(GPIOPinRead(MODE_2_BASE, MODE_2_PIN));
}

tBoolean is_nDecay_on(void)
{
    return(GPIOPinRead(DECAY_BASE, DECAY_PIN));
}

tBoolean is_STEP_HL_on(void)
{
    return(GPIOPinRead(STEP_BASE, STEP_PIN));
}
```

```
//*****************************************************************************
//
// DRV8824.h - Prototypes for Stellaris Motor Control.
// Author: Leslie Thomas (LT)
// co-authors: TJ Volinski & Pat O'Hara
//
// Copyright (c) 2011
//
//
// 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.
// LMI SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR
// CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
//
//*****************************************************************************

#ifdef __cplusplus
extern "C"
{
#endif



//*****************************************************************************
//
// Define Ports
//
//*****************************************************************************


#define STEP_BASE       GPIO_PORTE_BASE  // PE0 on LM3S9B95 CC
#define STEP_PIN        GPIO_PIN_0
#define STEP_PWM  PWM_BASE
#define STEP_GEN  PWM_GEN_2
#define STEP_OUT  PWM_OUT_4
#define STEP_BIT  PWM_OUT_4_BIT
#define STEP_INT  PWM_INT_GEN_2     // Interrupt for Step

#define DIR_BASE        GPIO_PORTA_BASE
#define DIR_PIN         GPIO_PIN_5

#define SLEEP_BASE      GPIO_PORTA_BASE
#define SLEEP_PIN       GPIO_PIN_3

#define RESET_BASE      GPIO_PORTH_BASE
#define RESET_PIN       GPIO_PIN_1

#define ENABLE_BASE     GPIO_PORTE_BASE
#define ENABLE_PIN      GPIO_PIN_1

#define DECAY_BASE      GPIO_PORTH_BASE
#define DECAY_PIN       GPIO_PIN_6

#define MODE_0_BASE     GPIO_PORTH_BASE
#define MODE_0_PIN      GPIO_PIN_0
#define MODE_1_BASE     GPIO_PORTF_BASE
#define MODE_1_PIN      GPIO_PIN_0

#define MODE_2_BASE     GPIO_PORTA_BASE
#define MODE_2_PIN      GPIO_PIN_4

#define NC_BASE         GPIO_PORTF_BASE
#define NC_PIN          GPIO_PIN_1


//*****************************************************************************
//
// Functions
//
//*****************************************************************************

extern void DRV8824_init(void);
extern void Frequency( signed int ulFreq);
extern void ACCEL( signed long pastFreq, signed long newFreq, unsigned long
AccelRate, unsigned long AccelTimeBase );
extern void DECEL( signed long pastFreq, signed long newFreq, unsigned long
AccelRate, unsigned long AccelTimeBase );
extern void STEP_ON(unsigned int cnt, signed int Freq);
extern void STEP_HL(tBoolean output);

extern void Direction(tBoolean output);
extern void nSleep(tBoolean output);
extern void nReset(tBoolean output);
extern void nEnable(tBoolean output);
extern void nDecay(tBoolean output);
extern void Mode_0(tBoolean output);
extern void Mode_1(tBoolean output);
extern void Mode_2(tBoolean output);
extern void NC(tBoolean output);

tBoolean is_Direction_on(void);
tBoolean is_nEnable_on(void);
tBoolean is_nSleep_on(void);
tBoolean is_nReset_on(void);
tBoolean is_Mode_0_on(void);
tBoolean is_Mode_1_on(void);
tBoolean is_Mode_2_on(void);
tBoolean is_nDecay_on(void);
tBoolean is_STEP_HL_on(void);




//*****************************************************************************
//
// Mark the end of the C bindings section for C++ compilers.
//
//*****************************************************************************
#ifdef __cplusplus
}
#endif
```

```
//***************************************************************************
**
//
// main.c - Windows DRV8824 GUI protocol
// Author: LT Thomas
// Copyright (c) 2011
//
// 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.
// LMI SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR
// CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
//  This is part of revision 4652 of the EK-LM3S9B92 Firmware Package.
//
//***************************************************************************
**
#include "inc/hw_ints.h"

#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/gpio.h"
#include "driverlib/pwm.h"
#include "driverlib/rom.h"
#include "driverlib/sysctl.h"
#include "driverlib/DRV8824.h"
#include "driverlib/interrupt.h"
#include "utils/uartstdio.h"


unsigned long AccelRate;
unsigned long AccelTimeBase;
signed long ulPeriod;
signed long ulFreq;
signed long pastFreq;

unsigned int cnt;
unsigned int stpCount;

//***************************************************************************
**
//
// The error routine that is called if the driver library encounters an
error.
//
//***************************************************************************
**
#ifdef DEBUG
void
__error__(char *pcFilename, unsigned long ulLine)
{
}
#endif

//code
```

```
//***************************************************************************
**
//
// This example connects the CC-LM3S9B95
//
//***************************************************************************
**


int
main(void)
{
    cnt = 0;

   // unsigned long ulPeriod;
    char SerialBuffer[5];

    //
    // Set the clocking to run directly from the crystal.
    //
    SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN |
                       SYSCTL_XTAL_16MHZ);


    //
    // Initialize the UART.
    //
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    UARTStdioInit(0);


    DRV8824_init();

    //
    // Loop forever while the PWM signals are generated.
    //
    int DesiredStepperSpeed;
    while(1)
    {
      // Reads the 5 bytes from the windows GUI
      SerialBuffer[0] = UARTgetc();
      SerialBuffer[1] = UARTgetc();
      SerialBuffer[2] = UARTgetc();
      SerialBuffer[3] = UARTgetc();
      SerialBuffer[4] = UARTgetc();


      // Switch statement for different commands
      switch(SerialBuffer[0])
      {

        // Write GPIO ( DIR DECAY nSleep, nReset )
        case (0x03):
          Direction(SerialBuffer[1] & 0x10);
          nSleep( SerialBuffer[1] & 0x80  );
          nReset( SerialBuffer[1] & 0x40  );
          nDecay( SerialBuffer[1] & 0x02  );


        // nEnable & StepLow
        case ( 0x0C ):
          switch(SerialBuffer[1])
          {
          case (0x02):
            nEnable(false);
            break;
          case (0x03):
            STEP_HL(false);
            break;
          }

            break;

        // nEnable & Step High
        case ( 0x0E ):
          switch(SerialBuffer[1])
          {
          case (0x02):
            nEnable(true);
            break;
          case (0x03):
            STEP_HL(true);
            break;
          }

            break;

        case ( 0x0F ):// Pulse 1 Step
            if( SerialBuffer[1] & 0x03 )
            {
              stpCount = 1;
              STEP_ON(stpCount, 250);


            }

            break;

        case ( 0x1A ): // move steps
              DesiredStepperSpeed = (SerialBuffer[1] * 256) +
SerialBuffer[2];   //Configure the Frequency Rate
              ulFreq = (4000000/ DesiredStepperSpeed);
              stpCount = SerialBuffer[3]*256 + SerialBuffer[4];
              STEP_ON(stpCount, ulFreq);

            break;


        case ( 0x16 ): // Modes
          Mode_0(SerialBuffer[1] & 0x08);
          Mode_1(SerialBuffer[1] & 0x02);
          Mode_2(SerialBuffer[1] & 0x01);
          break;

        case ( 0x17 ): //Start Stepper

              DesiredStepperSpeed = (SerialBuffer[1] * 256) +
SerialBuffer[2];
              ulFreq = (4000000/ DesiredStepperSpeed);
              AccelRate = SerialBuffer[3];
              AccelTimeBase = SerialBuffer[4];
              ACCEL( 62, ulFreq, AccelRate, AccelTimeBase );
              pastFreq = ulFreq;

          break;

      case ( 0x18 ): //stop stepper
              DECEL(pastFreq, 0, AccelRate, AccelTimeBase);
              pastFreq = 0;
              break;

      case ( 0x19 ):  //update stepper

              DesiredStepperSpeed = (SerialBuffer[1] * 256) +
SerialBuffer[2];
              ulFreq = (4000000/ DesiredStepperSpeed);
              AccelRate = SerialBuffer[3];
              AccelTimeBase = SerialBuffer[4];
              if( ulFreq > pastFreq )
                ACCEL( pastFreq , ulFreq, AccelRate, AccelTimeBase );
              else
                DECEL( pastFreq , ulFreq, AccelRate, AccelTimeBase);

              pastFreq = ulFreq;
              break;


      }
    }

}
```



Task Name	Duration	Start	Finish	Predecessors	Resource Names	WBS Predecessors
Semester	75 days	Tue 1/18/11	Sat 4/30/11			
Project Planning	12 days	Tue 1/25/11	Wed 2/9/11			
PCB Fab for DIMM and EVM	**14 days**	**Thu 2/10/11**	**Tue 3/1/11**	**2**		**2**
DIMM Standards	6 days	Wed 2/9/11	Wed 2/16/11		LT	
Review F1612 Code	5 days	Thu 2/17/11	Wed 2/23/11		LT,TJ	
PWM Pins	4 days	Wed 2/9/11	Sat 2/12/11		TJ	
USB to SPY wire	4 days	Sun 2/13/11	Wed 2/16/11		TJ	
Remove Digital From PCB	14 days	Wed 2/9/11	Mon 2/28/11		PAT	
Contact TI	5 days	Mon 2/14/11	Fri 2/18/11		PAT[10%]	
Make Webpage Look Better	10 days	Tue 2/15/11	Mon 2/28/11		KOLE[20%]	
PCB Tutorials & DIMM Fab	10 days	Tue 2/15/11	Mon 2/28/11		KOLE[80%]	
Add Pictures To Website	4 days	Wed 2/9/11	Mon 2/14/11		KOLE	
Research Coding & Wait for Parts & Test	**37 days**	**Tue 3/1/11**	Wed 4/20/11	**3**		**3**
Assemble Boards	32 days	Tue 3/1/11	Wed 4/13/11		KOLE,PAT	
Code Cortex	40 days	Thu 2/24/11	Wed 4/20/11		LT	
Code F1612	40 days	Thu 2/24/11	Wed 4/20/11		TJ	
Design Day Prep	8 days	Thu 4/21/11	Sat 4/30/11	13		4