Integration of Motor-Drive-Controllers in Ultrasound Probes

Alois Moosleitner

Graz, April 2014

Institute of Electronics
Graz University of Technology

Adviser: Ass.Prof. Dipl.-Ing. Dr.techn. Bernd Eichberger

Abstract

GE Healthcare Austria & Co OG is a manufacturer of ultrasound diagnostic equipment and worldwide leader in the area of 3D/4D-ultrasound. The headquarters in Zipf/Upper Austria is a global "Center of Excellence" inside the GE Healthcare concern. Because there are many different applications of ultrasound, various types of probes and consoles are being developed. Each specific application requires an optimal pairing of these components. Therefore, the decoupling of probe and console is desirable in order to handle different pairings. The purpose of this thesis is selection and integration of the controller, which is needed for driving the stepper motor of a mechanic volume probe. The controller should be integrated into the probe, not into the console. So it is possible to use the probe on consoles without an integrated motor controller. First of all, different stepper motor controller-ICs should be compared. Then the selected IC should be integrated in a mechanic volume probe. Therefore, the needed peripheral electronics also have to be developed. Furthermore, a prototype of this probe will be built. This prototype should be checked with respect to image quality and electromagnetic compatibility (EMC). Finally, suggestions for improving the image quality should be worked out if needed and the design has to be verified and validated. The project handling of the implementation and the connected project management are also part of this master's thesis.

Kurzfassung

GE Healthcare Austria GmbH & Co OG ist Spezialist für diagnostische Ultraschallsysteme und Weltmarktführer im Bereich 3D/4D-Ultraschall. Der Unternehmenssitz in Zipf/Oberösterreich ist globales „Center of Excellence“ innerhalb des GE Healthcare Konzerns. Da es für die Ultraschalltechnologie zahlreiche Anwendungen gibt, werden verschiedene Sonden und Konsolen entwickelt, um jeweils die optimale Paarung dieser Komponenten bereitstellen zu können. Eine Entkopplung von Sonde und Konsole ist aus diesem Grund erstrebenswert, um verschiedene Paarungen anbieten zu können. Zweck dieser Arbeit sind Auswahl und Einbindung der Steuerung, die benötigt wird, um den Schrittmotor einer mechanischen Volumensonde zu betreiben. Diese Steuerung soll nicht in die Konsole, sondern in die Sonde integriert werden, um die Sonde auch auf Konsolen ohne Motorsteuerung betreiben zu können. Es sollen zunächst verschiedene, am Markt erhältliche Motor-Drive-Controller-ICs verglichen werden. Der so ausgewählte IC soll dann in eine mechanische Volumensonde integriert werden. Hierfür muss die nötige Peripherieelektronik entwickelt werden. Als nächster Schritt wird ein Prototyp der Sonde gebaut. Der Prototyp soll hinsichtlich Bildqualität und elektromagnetischer Verträglichkeit (EMV) untersucht werden. Abschließend sollen, falls nötig, Verbesserungsvorschläge für die Bildqualität erarbeitet und das Design verifiziert und validiert werden. Die Abwicklung der Implementierung und das damit verbundene Projektmanagement ist ebenfalls Teil dieser Arbeit.

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, _____ _____

Date Signature

Eidesstattliche Erklärung[1]

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am _____ _____

Datum Unterschrift

[1] Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008; Genehmigung des Senates am 1.12.2008

Acknowledgements

First of all I would like to express my gratitude to **Professor Eichberger**, whose lectures Elektronik Mk VO and LU I have enjoyed very much and which inspired me to pick an electronics project for my master's thesis. He also supported me during the project and guided me through the writing procedure.

Special thanks to several people at GE Healthcare, especially to **Andreas Kremsl**, **Christian Kreutzer**, **Yafang Cheng**, **Thomas Rittenschober**, and all the others in the probes engineering department for their continuous support. Without the help of every person at any time the finalization and the success of this project would not have been possible.

Not at least I have to thank my whole family for their support during my student life. They have always been a source of motivation and energy in difficult times.

Furthermore, I would like to give special thanks to **Johanna Pillichshammer** for the spelling and grammar checking.

Contents

1 Introduction

Ultrasound can be used for many different applications. GE Healthcare uses it in medical equipment for the diagnosis of different diseases, the supervision of pregnancies and other examinations concerning women's health. There are many different types of examinations which require different examination tools.
GE Healthcare Austria & Co OG is a manufacturer of ultrasound diagnostic and worldwide leader in the area of 3D/4D-ultrasound. The headquarters in Zipf/Upper Austria is a global "Center of Excellence" inside the GE Healthcare concern. The plant in Zipf is specialized in the women's health segment. Caused by the big diversity of examinations in this segment, different probes and consoles are developed. Each specific application requires an optimal pairing of these components. Therefore, the decoupling of probe and console is desirable in order to handle different pairings.
The purpose of this thesis is selection and integration of the controller, which is needed for driving the stepper motor of a mechanic volume probe. The controller should be integrated into the probe, not into the console. So it is possible to use the probe on consoles without an integrated motor controller. In every actual volume probe solution the motor controller is located on the console-side.
Additionally, the control interface should be standardized to be usable for prospective probe development projects. Therefore, some driver-ICs with different interfaces should be compared. Then the selected driver-IC should be integrated in a mechanic volume probe. The needed peripheral electronics, especially the needed microcontroller (μC) for signal conversion, also has to be selected. Furthermore, a prototype of this probe should be built. This prototype should be checked according to image quality and electromagnetic compatibility (EMC).
Finally, suggestions for improvements according to image quality should be worked out if needed and the design has to be verified and validated. The project handling of the implementation and the connected project management are also part of this master's thesis.

2 Initial Situation

The actual motor controller is located on the console-side. A μC receives the reference values of the SIN- and COS-current, which should flow through the motor windings under the control of a supervising μC. The received values are compared to the measured current values and the new actuating values are set. This is called "closed loop current regulation". The same type of regulation should be used in the probe-sided motor control solution.
The power supply can generate voltages from -36 V to +36V. These voltages are needed to supply both half-bridges of a bipolar motor's windings. This type of power supply should be replaced by a simpler solution.
An advantage of the actual motor controller is that at the console-side, there is much space available. Hence, there is no installation space problem.

2.1 GE Healthcare

GE Healthcare is a division of GE Technology Infrastructure, which is itself a division of General Electric (GE). It employs more than 46,000 people worldwide and is headquartered in Little Chalfont, Buckinghamshire, United Kingdom.
GE Healthcare Austria is specialist for diagnostic ultrasound systems and worldwide leader in 3D/4D-ultrasound. The plant in Zipf/Upper Austria is a global "Center of Excellence" inside the GE Healthcare concern.
The actual field of activity of GE Healthcare Austria can be divided into the system-side and the probe-side.

2.1.1 Systems/Consoles

Source: http://www3.gehealthcare.com/en/Products/Categories/Ultrasound/Voluson/Voluson_E8, 06/FEB/2014

Figure 1: Voluson E8 Expert

To give an overview of all consoles which were/are developed in Zipf here is list of the current console series:

- Voluson E8 Expert
- Voluson E6
- Voluson S8
- Voluson S6

Source: http://www.gehealthcare.com/usen/ultrasound/voluson/signature_series/voluson.html, 06/FEB/2014

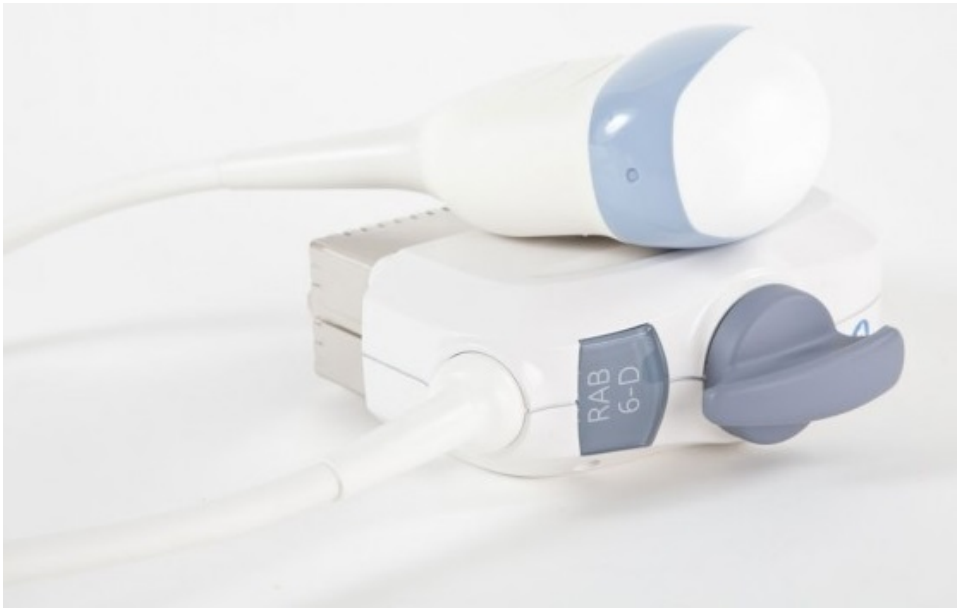
Figure 2: Voluson *i*

- Voluson P8
- Voluson *i*
- Voluson e
- Voluson 730

2.1.2 Probes

Also there are different probes which were/are developed in Zipf. Here is a list of the current probe projects:

- IC5-9-D
- IC5-9-RS
- RAB4-8-D
- RAB4-8-RS
- RAB6-D
- RM6C
- eM6C



Source: http://www.medcorpllc.com/ge-rab6-d-probe.html, 06/FEB/2014

Figure 3: RAB6-D Probe

2.2 General Information about Ultrasound Probes

The basic principle of ultrasound imaging is quite simple. A piezo-electric element generates an acoustic wave with frequencies in the range of few megahertz. This wave passes through several layers of the examined object, gets reflected and returns to the piezo element. The returning wave generates an electric signal, which gives an image of the examined body region after processing.
However, there are difficulties in the development of ultrasound probes. Many different physical effects influence the wave passing through the examined body. Another difficulty results from the space problem. Probes which are introduced vaginal or rectal must not exceed predefined dimensions. This requires a high degree of miniaturization of the used components in a probe. Additionally, there are several laws that define boundary conditions, for example the maximal surface temperature or that the used surface materials have to be biocompatible.
These are just a few of the boundaries concerning the development of an ultrasound probe.

2.3 Motive of this Project

A problem of the actual solution is that the cable between console and probe is very long (approximately 2.5 m) and the relative high motor current (approximately 2 A) flowing through the cable causes electromagnetic fields. These fields influence the signal lines of the piezo elements and the image quality can suffer if the signal lines are not shielded or another protection technology is used. A probe-sided motor controller would just need the DC-power supply lines and three lines for a serial peripheral interface (SPI), which do not require additional filtering.

In general, the number of wires in the cable should be minimized. This becomes possible when the motor controller does not have to send current measurement values through the cable. Also the number of high-current wires can be reduced from four (two for SIN, two for COS) to two (voltage supply, ground).

3 Systems Engineering

In order to rise the development speed and ensure a project's success, it is convenient to use a predefined development methodology. There are different methods for different types of projects. One of the most popular in engineering development is the systems engineering method.

3.1 Process Model, Development Method

As defined in one of the components of the systems engineering model, this project should be developed **'from rough to fine'**. This means, that at the beginning of the project all available boundary conditions are defined and in the following steps the details increase.
In general, all possible **variants** should be considered in the beginning of the project. In the following steps only the right variants will remain for further development. When the methods of systems engineering are used the right way, the best variant will be the remaining one at the end of the project.
With the developing project both the degree of detail and the overall project maturity will rise. In systems engineering this means to think in **project phases**.
For the specific project phases there are methods, which help to solve problems as efficiently as possible. One of them is called **problem solution cycle**.
These are the main components of systems engineering.

3.2 Project Management

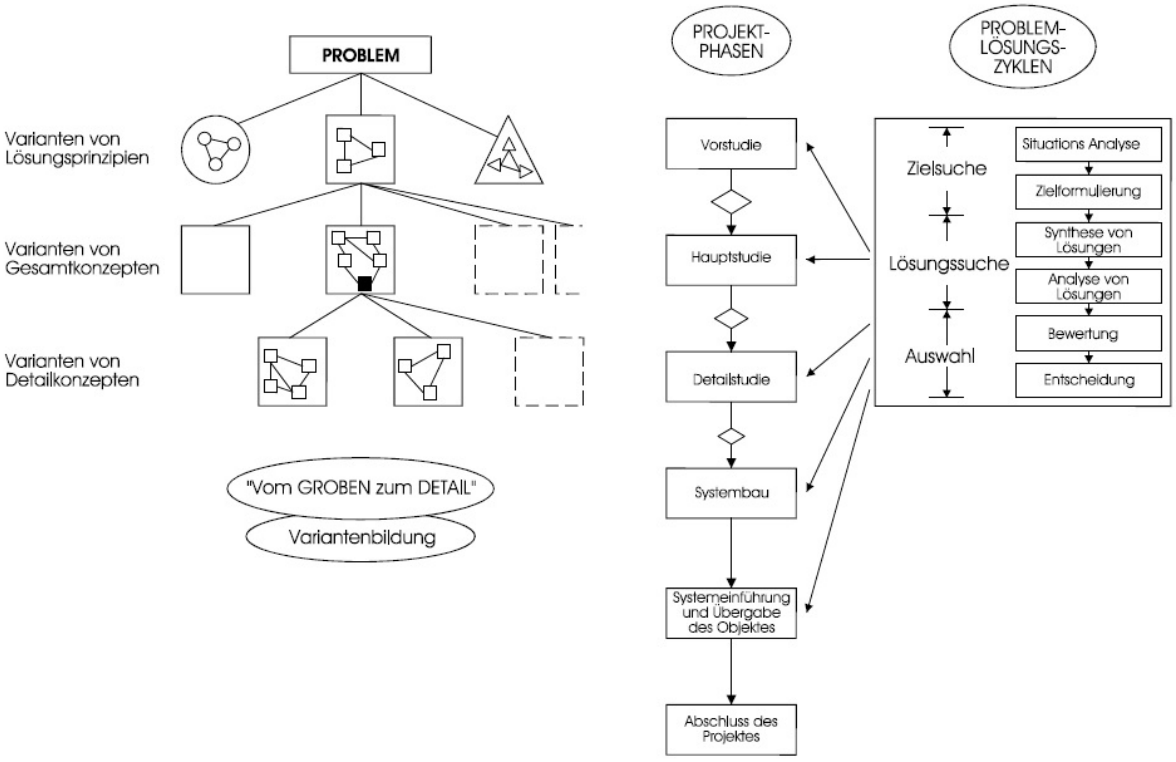
The project has started in mid-July 2013. To achieve the standard duration for master's theses of approximately six months, the project was determined to be finished at the end of February 2014.

3.2.1 Project Plan

To get a better feeling for the actual status, the project is divided into five phases. At the end of each phase there has to be a result. Based on this result more detailed project milestones of the following phases are defined.

3.2.2 Phase 1: Project Start

- Familiarization



Source: [1]

Figure 4: Components of Systems Engineering

- Literature research
- Definition of boundary conditions
- Setting up communication with other departments inside GE Healthcare Austria
  - Systems Engineering
  - Software Engineering
  - Probes Manufacturing
  - New Product Introduction (NPI)
- Comparison of preselected motor control driver-ICs
- Selection of further motor control driver-ICs
- Comparison of different μCs

Result Phase 1: **Two** motor driver-ICs selected

3.2.3 Phase 2: Evaluation

- Order evaluation boards of the driver-ICs
- Selection of a μC
- Assemble the test equipment
- Interface implementations
- Functional tests of the driver-ICs

Result Phase 2: **One** driver-IC and **one** μC selected

3.2.4 Phase 3: Implementation & Integration

- Selection of adaptable probes
- Build space analysis of the selected probes
- Detailed definition of the interface between motor controller and console
- Design of mechanical parts
- Printed Circuit Board (PCB) Prototyping
  - Selection of peripheral parts for the driver-IC and the μC
  - Draft of the schematics
  - Draft of the board layout

Result Phase 3: Motor controller board ordered

3.2.5 Phase 4: Firmware Completion & Electromagnetic Compatibility (EMC)

- Firmware completion (error event handling, safety features)
- Preparing a console for testing issues
- Build up the prototype probe with integrated motor controller board
- Integration tests
- EMC measurements
- Documentation

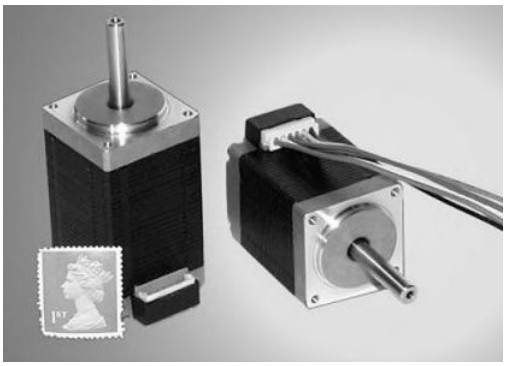
Result Phase 4: Options for signal quality improvement and miniaturization

3.2.6 Phase 5: Documentation

- Finalize documentation
- Project completion

4 Motor Control

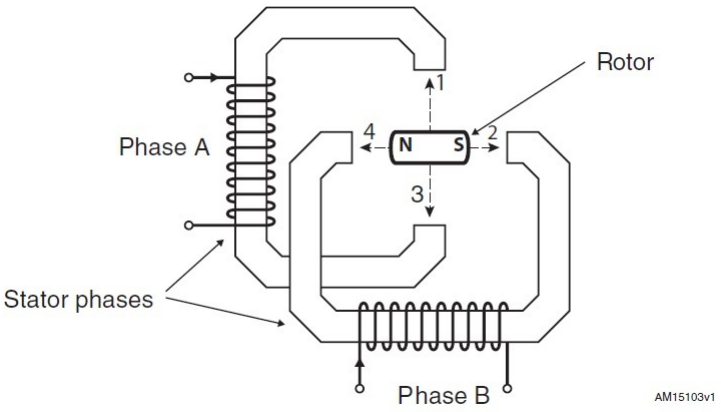
4.1 General Motor Data



Source: [2]

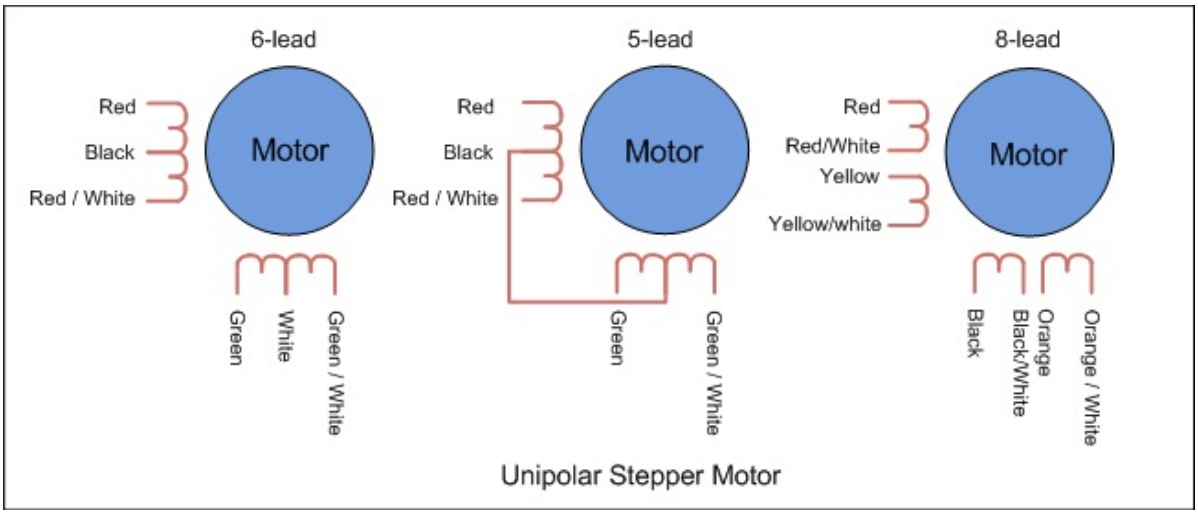
Figure 5: Picture of a Stepper Motor

The used motors are called "steppers" or "hybrid motors". There are two basic wiring methods: **unipolar** and **bipolar**. In ultrasound probes the bipolar method is used because it requires fewer wires. The difference between these methods can be explained by simple examples.



Source: ST Microelectronics Application Note AN4158

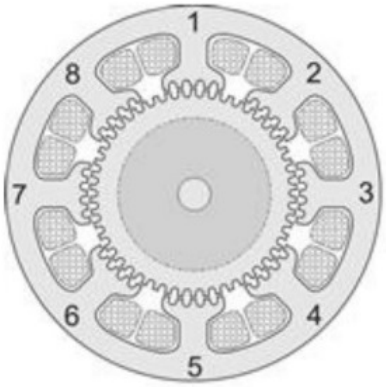
Figure 6: Basic two-phase bipolar Stepper Motor Function Principle



Source: http://www.engineersgarage.com/articles/stepper-motors?page=5, 07/OCT/2013

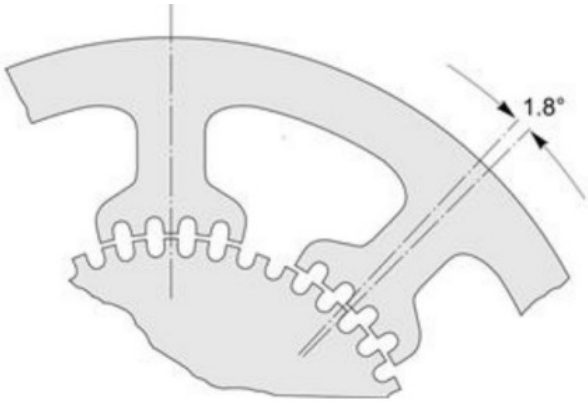
Figure 7: Basic two-phase unipolar Stepper Motor Function Principle

When the electromagnetic field changes in a specific way, the rotor makes one full step. The most common count of full steps is 200 for one rotation. This means, that the rotor turns 1.8°/step. To let the rotor turn clockwise (seen from the shaft end), the polarity sequence in figure ten has to be applied.



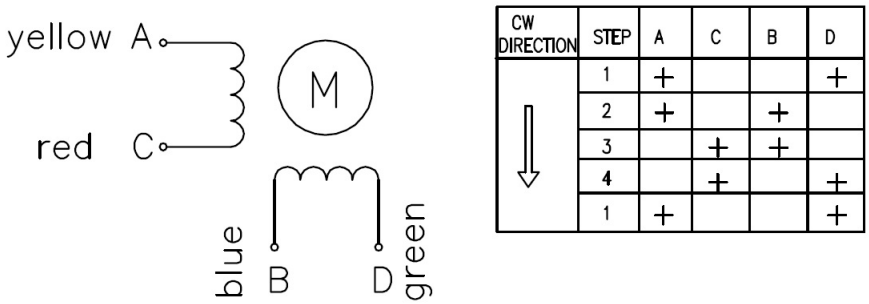
Source: [2]

Figure 8: Section through a Stepper Motor



Source: [2]

Figure 9: Step Angle



Source: FL25STH48-1204A Motor Data Sheet

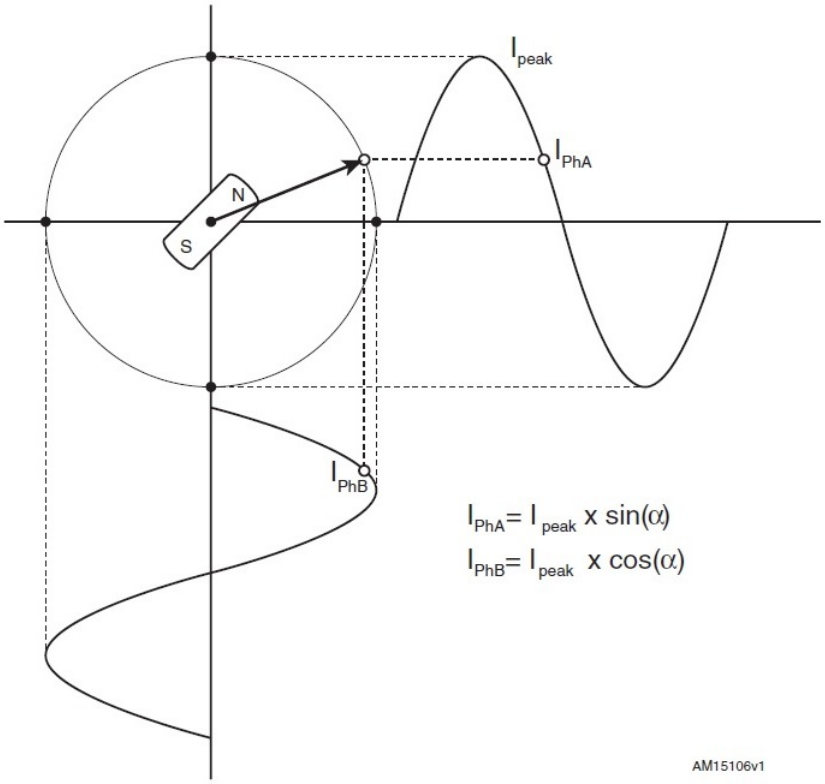
Figure 10: Polarity Sequence

4.2 Microstepping

For applications where exact positioning is needed, the count of steps provided by the mechanical construction of a stepper motor is not enough. Fortunately, it is possible to make the steps as fine as needed with electric methods. This can be done by microstepping.
The optimal phase currents should be sinusoidal. It is not possible to generate absolutely perfect sinusoidal phase currents with a μC and a driver-IC, but the degree of discretization can be very high. Thus, the phase current looks almost sinusoidal.

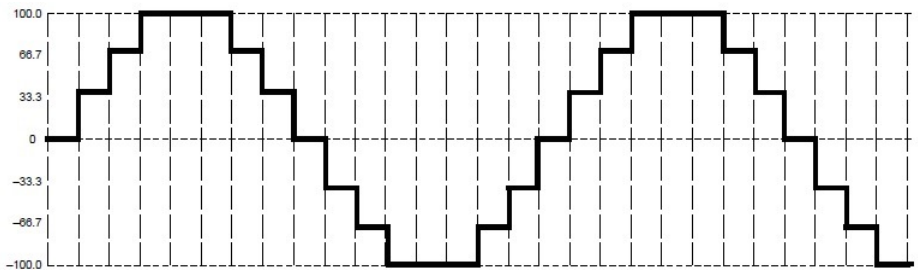
4.3 Pulse Width Modulation (PWM)

The discretization of the phase currents can be done by PWM. Many μCs have integrated PWM-modules which can generate these signals. These signals have to get amplified by a driver-IC to supply an appropriate current to the motor windings.
The average current fed to the winding is controlled by turning the switch between supply and winding on and off at a high frequency. The longer the switch is turned on in relation to the off period the higher is the power supplied to the winding.
The PWM switching frequency has to be much higher than what would affect the winding. Typical switching frequencies for motors range from a few kHz to a few tens of kHz. The higher the frequency is, the more sinusoidal is the motor current. An upper boundary for the frequency is the needed calculation time performed in the μC. **The selected frequency should be chosen as high as possible, but not higher than what would exceed the needed calculations duration**.



Source: ST Microelectronics Application Note AN4158

Figure 11: Principle of Microstepping



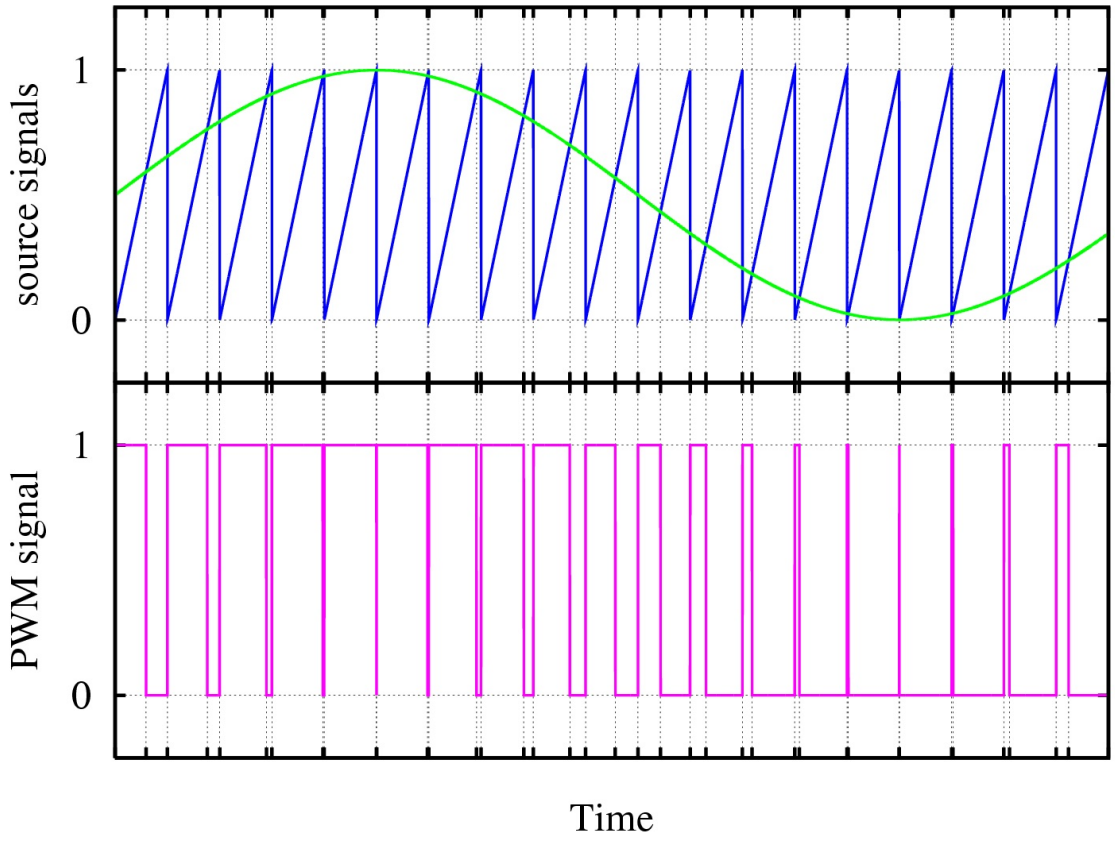
Source: Allegro A4986 Data Sheet

Figure 12: Discretization of the Phase Current (here: quarter-step-mode)

4.4 Electrical Boundary Conditions

4.4.1 Supply Voltages

On the side of the console there is a power supply unit (PSU), which generates a regulated output voltage of 12 V. This voltage should be used to supply the motor controller board. As a result, the maximum current is mainly limited by this voltage and the impedance of the system.



Source: http://en.wikipedia.org/wiki/Pulse-width_modulation, 08/OCT/2013

Figure 13: PWM Principle

$$I_{max} = \frac{U_{Supply}}{Z} \tag{1}$$

To supply the control logic of the board, a regulated voltage of 3.3 V is needed. It can be provided by a standard linear regulator.

4.4.2 Reference Signals

The reference values of the SIN- and the COS-current are transferred alternately per SPI at a frequency of 50 kHz. This means that a **pair of new values** is transferred at a rate of **25 kHz**. In general a SPI consists of the following four lines:

- Serial Data Input (SDI)
- Serial Data Output (SDO)
- Serial Clock (SCK)
- Slave Select/Chip Select (SS)

In this case, the SDO line is not needed because signals, such as error messages, can be transferred over another interface. In the following, the SPI between the console-side and the motor controller board is called **SPI1**.
The communication between the supervising μC and the μC on the motor controller board is **word-wide**. One word consists of 16 bits. The structure of these words can be explained by an example.

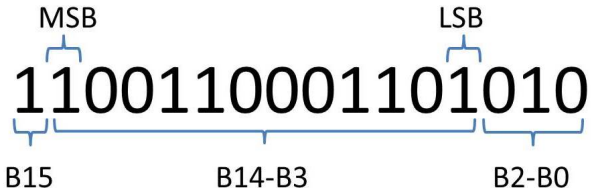


Figure 14: SPI1 Word Data Format

- **Bit 15**
  0 ... SIN-value
  1 ... COS-value

- **Bits 14-3**
  The reference value is transferred as an unsigned 12 bit integer number. The transfer of the value starts with bit 14, the most significant bit (MSB). The last bit of this group (bit 3) is called the least significant bit (LSB).
  0x000 ... maximal negative value
  0x800 ... zero value
  0xFFF ... maximal positive value

- **Bits 2-0**
  These bits are used for parity-checking.
  101...SIN-value was transferred successfully
  010...COS-value was transferred successfully

This control interface was selected because the three signal lines are easy accessible and there have to be no changes on the console-side.

4.5 Mechanical Boundary Conditions

The main boundary condition is the space available in an ultrasound probe. As a result, the motor controller board has to be as small as possible. Furthermore, there are different types of

ultrasound probes as mentioned in the introduction: abdominal, rectal, vaginal, and others. To minimize the amount of different parts just one motor controller board should get developed which fits into all different probe types. To find an optimal solution, all probes have to get investigated according to free spaces.
Some probes (e.g. RM6C, RRE) do not offer any available free space. The motor controller board can not get integrated into these probes.

4.5.1 Probes with free Spaces

RAB

The most capable probes for the integration of the motor controller board are the RAB-probes. There are two possible locations:

- Space between motor and body (1)
- Space between motor and multiplexer-boards (MUX-boards) (2)

Space (1) is a free room with the dimensions **36x20x7 mm**. The disadvantage of this space is that the assembly process has to be changed.
Space (2) would be ideal to mount the motor-controller-board because the assembly process of the probe just needs minimal adaption and the board does not have to be very flat. So there is place for electrolyte capacitors and other higher parts.

RIC

In RIC-probes, the available space is small but well proportioned. The free space in the shaft in front of the motor can be used to house the motor controller board. There are just the rod or the rope, which is connected to the mechanical parts in the probe head, and the flexes.

RNA

At the opposite side of the volume compensation tube, there is a cubic space of **40x20x4.5 mm** available.

RSP

There is enough space for an additional board. The board can be fixed at the carrier block.

RSM

With an additional carrier, the motor controller board can be integrated.

4.6 Thermal Boundary Conditions

The surface temperature of an ultrasound probe must not exceed the predefined limit of 43°C. Due to the fact that many heat-sources are already installed in ultrasound probes and that most of the probes are operated at the upper temperature limits, an additional heat source must be as

low as possible. As a result, the used components must be selected according to performance, build space and also heat dissipation.

4.7 μC

To convert the signals coming from the SPI and to generate the control signals for the driver-IC a μC is needed. There is a broad range of available μCs on the market. To ease the selection, the following boundary conditions can be defined:

- Small package
- Low dissipation
- 3.3 V supply voltage
- 2 SPIs
- 2 PWM channels
- 16 or 32 bit wide data path
- 2 Analog-to-digital converter (ADC) channels
- Internal oscillator

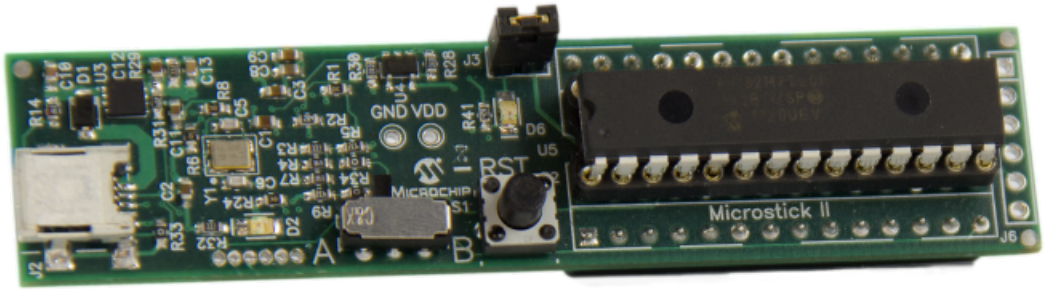
Within these boundary conditions the **dsPIC33EP256MC202** [3] μC from **Microchip** is a good choice.
For evaluation issues, there has to be a quick solution. To avoid designing an extra evaluation board for the dsPIC33EP256MC202, a different μC with the same functions and an already existing evaluation board can be used. The **dsPIC33FJ128MC802** [4] from Microchip offers all the needed functions except the small package, but for evaluation issues a bigger package is acceptable.
The selected evaluation board is called **Microstick II** [5].

4.8 Motor Control Methods

There are basically three methods to control a stepper motor, which are listed and described below.

4.8.1 Open Loop - Fixed Voltage Control

> *In classic voltage control, the rated motor voltage is applied to the windings. When a higher voltage power supply is used, such as 24V, the motor rated voltage is reduced by the use of a chopper, which is implemented with the Pulse Width Modulation (PWM) module.*
> *Stepper motors are designed to run reliably at the rated current, as instructed by the manufacturer. The rated motor voltage is based on that current and the*



Source:
http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en556208, 04/NOV/2013

Figure 15: Microstick II

winding resistance. However, the voltage across the motor can be higher than that, as long as the current is kept at all times at the rated value or lower. [6]

4.8.2 Open Loop - Fixed Current Control

When using fixed voltage control, the motor is driven with the rated voltage, which allows the current to rise from zero to the rated current value in a fixed amount of time. At a certain motor speed, which depends on the motor inductance and the drive voltage, the current will not rise fast enough through the motor coil to reach the rated motor current and torque will be lost. This presents a problem when higher speeds are required by the system.
As the motor speeds up, the step time is getting smaller and the current amplitude is falling more and more, until the rotor eventually stalls. To overcome this problem, the easiest solution is to increase the drive voltage as the motor speeds up in order to have a maximum current amplitude equal to the rated motor current and extend the maximum torque versus speed range. [6]

4.8.3 Closed Loop Current Control

A simple control loop is used for controlling the current amplitude. The maximum amplitude of the current in both motor windings is sampled during one complete sine wave. If the maximum current amplitude is lower than the desired value, the drive voltage is increased gradually by adjusting the PWM duty cycle until the desired current amplitude is reached. If the current is too high the duty cycle is decreased, but not less than the initial value corresponding to the rated motor voltage.
As long as the drive voltage is higher than the rated motor voltage, this method provides an extended speed range over the classic open loop approach. Another advantage of using this algorithm is that there is no need to retune for different

motors. [6]

The motor torque is proportional to the current. This can be reached by sampling the current amplitude more often than just at every end of a sine wave. When the current measurement gets triggered at the PWM frequency, a good result can be achieved.

4.9 Bridge Configurations

Basically, there are two types of bridges that can be used to drive one winding of a stepper motor. Both configurations have advantages and disadvantages.

Properties of a Half-Bridge Configuration

- Four FETs for two phases
- Positive and negative supply voltage + ground

Properties of a Full-Bridge Configuration

- Eight FETs for two phases
- Positive supply voltage + ground

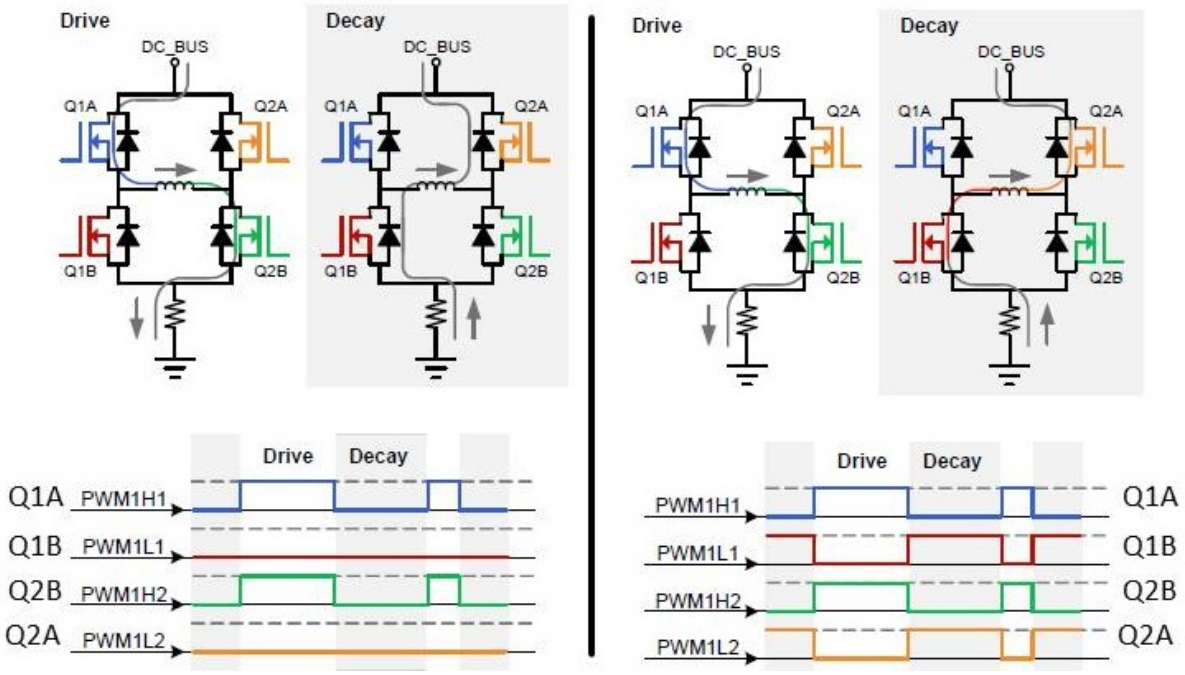
As defined in the boundary conditions +12 V are available. For this reason the use of a **full-bridge** is appropriate.

4.10 Decay Modes

4.10.1 Fast Decay Mode

This mode is active when the voltage across the de-energized winding is reversed, which produces a fast current drop. For this reason this mode is called "fast decay mode".
The advantage of using this method is that the decaying current is flowing through the MOSFET body diodes only briefly, until the MOSFET turns ON. The MOSFET has a lower ON-resistance and thus, the dissipated power will be much lower, which presents an advantage to the overall system power dissipation.
Another advantage of the fast decay mode is the simplicity of the current feedback circuit, since motor current can be read from the shunt resistor at all times. When the winding is driven, the current is positive. While the current is dropping during fast decay mode, the current will be negative since the voltage is reversed across the winding. Therefore, current is available on the shunt resistor at all times.
With a slight variation on the drive signals, we have something called "reverse

decay mode". Reverse decay mode behaves like fast decay mode until the current reaches zero, at which point it forces the current in the opposite direction. For short decay times though, until the current reaches zero, this is not an issue. If reverse decay is continued after the current has dropped to zero, then negative current will be generated when a positive current is desired, and vice versa. Reverse decay generates the lowest possible dissipated power in the fast decay configuration. [6]



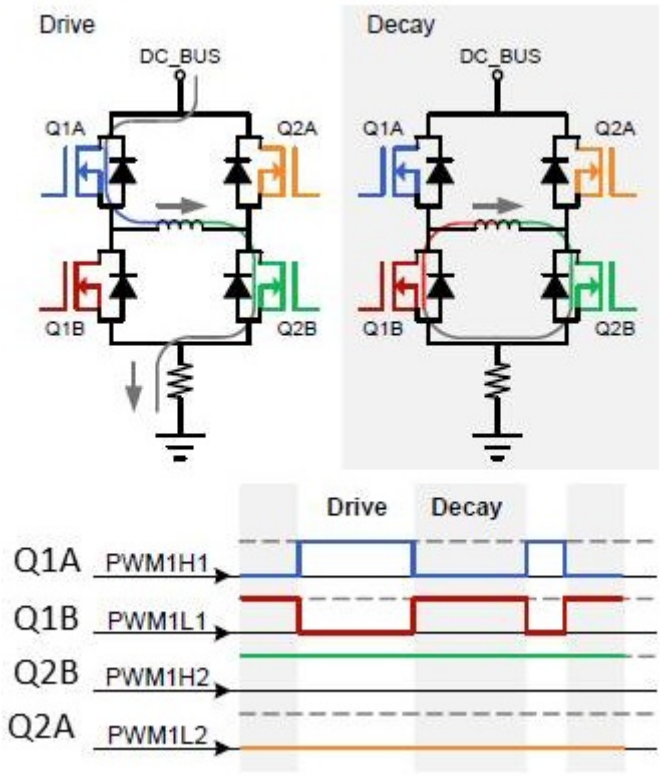
Source: [6]

Figure 16: Left: Fast Decay, Right: Fast Decay (Reverse)

4.10.2 Slow Decay Mode

Slow decay is entered by shorting the motor winding when it is not driven by the supply voltage. This is achieved by keeping one of the drive MOSFETs (Q2A) opened at all times.
***Current measurement is not possible** in slow decay modes with the shunt resistor circuit used for current sensing. This is because in slow decay modes, current is not flowing through the shunt resistor since it recirculates through the motor and MOSFETs or diodes.* [6]

With this knowledge it becomes clear that one of the fast decay modes has to be used when a constant current control loop should be implemented. Additionally, in **fast decay (reverse) mode** the least power gets dissipated.



Source: [6]

Figure 17: Slow Decay Mode

4.11 Current Measurement

Caused by the usage of a full-bridge configuration, the winding current can just be measured in the following way.

> *Current measurement in the full-bridge configuration brings up some challenges. First of all, the measuring shunt resistor is located between the ground and the low side MOSFETs, which means that no current will be visible unless there is a path opened between DC_BUS and ground. The path can either be one high-side MOSFET plus the opposite low-side MOSFET, or the body diodes of the same MOSFETs when they are turned OFF.*
> *When the motor winding is energized, the shunt current will always be positive, regardless of the current direction in the motor winding. Whenever the winding is in fast decay, the shunt current will be negative. In all slow decay modes there is no current flowing through the shunt resistor.* [6]

4.12 Low-pass Filtering

For the best positioning results at low speeds, the motor currents should be sinusoidal.
To achieve the best result, PWM period resolution and PWM frequency should be as high as possible. Optimizing these parameters brings better results but there will always be small ripples without additional filters.
The simplest way to get rid of the PWM-caused ripple is to add low-pass filters between the driver-IC and the motor windings. The order of the filter, which has to be used, depends on the requirements on the image quality. There will be less distortions in the image when there is a better filter. Therefore, different filtering solutions should be evaluated in the prototyping phase.
Additionally, the inertia of the whole power train helps to make the operation of the motor smoother.

5 Evaluation of available Driver-ICs on the Market

5.1 Boundary Conditions

- Small package
- Low dissipation
- 3.3 V logic supply voltage
- Integrated field effect transistors (FETs)
- PWM interface or SPI
- Min. 2 A peak current
- Full-bridge configuration
- Up to 30 V operation voltage range

5.2 Interfaces

5.2.1 PWM Interface

The standard-solution to control a stepper motor is to send low-power PWM-signals and amplify them with the driver. For the purposes of this project, the best fitting driver with the PWM interface is the **DRV8841** from **Texas Instruments**.

5.2.2 SPI

Since there are several driver ICs, which are controlled via SPI and the interface to the console-side is SPI, also one of these drivers has to be evaluated. The intention is to control the driver-IC directly from the console-side via high-level-commands. In this configuration the μC on the probe-side would not be needed.
The **L6470 from ST Microelectronics** fits best to the requirements.

5.3 Comparison of the Drivers

Package Size

Both of the selected ICs are available in a Shrink Small-Outline Package (SSOP) with 28 pins. This is the smallest package available.

Dissipation

The generated heat must be transferred away from the IC. There are many different options for heat sinking. The most common are:

- Thermal connection at the IC's bottom (thermal pad)
- Additional cooling body mounted on the IC's top
  - Cooling body without fan
  - Cooling body with fan
  - Water cooled body
- Heat conduction through the pins

The most powerful option would be an additional cooling body, but due to the space boundaries this option should not be used. The least powerful would be heat sinking through the pins, because the used package has very small pins. Using this method, the cooling performance would be inadequate. A **thermal pad at the IC's bottom** delivers the optimal cooling performance compromise and is also quite simple.
The driver data sheets deliver the total power dissipation information.

Driver	**Total Power Dissipation**	**Test Conditions**
L6470	5 W	25 °C Ambient Temperature [7]
DRV8841	4.8 W	Value approximated by Equation 2 in Data Sheet [8]

Table 1: Power Dissipation

The power dissipation values vary because of the temperature dependency of the FETs. The total dissipated power rises with rising ambient temperature.

5.4 Evaluation Boards

To test the driver ICs, boards with peripheral electronic parts (bypass capacitors, resistors, ...) are needed. Most of the IC suppliers offer evaluation boards.

5.4.1 L6470 Evaluation Board

Source: [9]

Figure 18: EVAL6470H

Functions and Interfaces [9]

- SPI 10 Pin Header
- Power Connector Screw Terminal
- Motor Phase Screw Terminals
- Measurement Loops
- Test Software
- 1/128 Microsteps

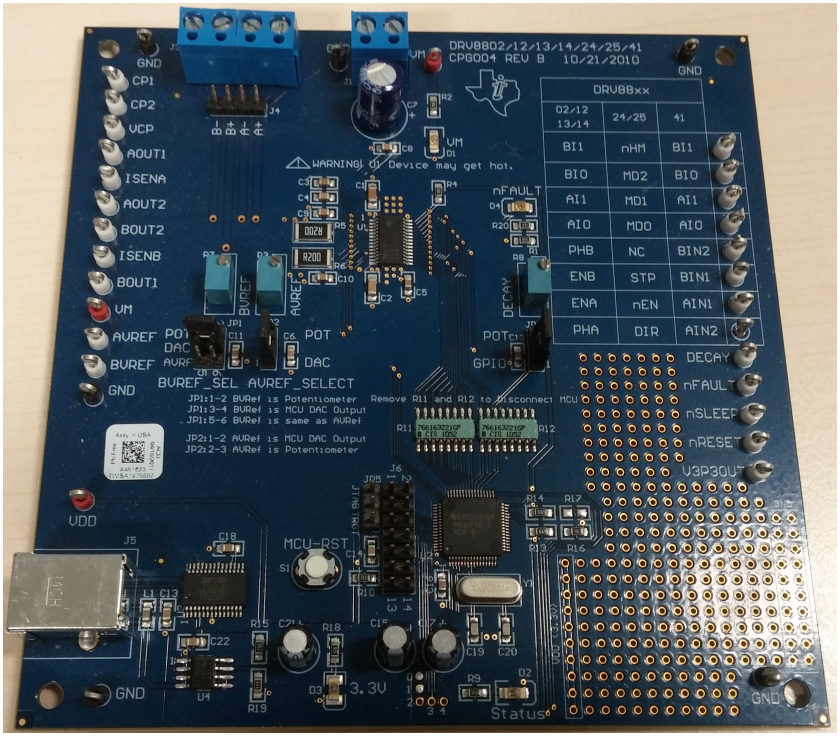


Figure 19: DRV8841 Evaluation Board

5.4.2 DRV8841 Evaluation Board

Functions and Interfaces [10]

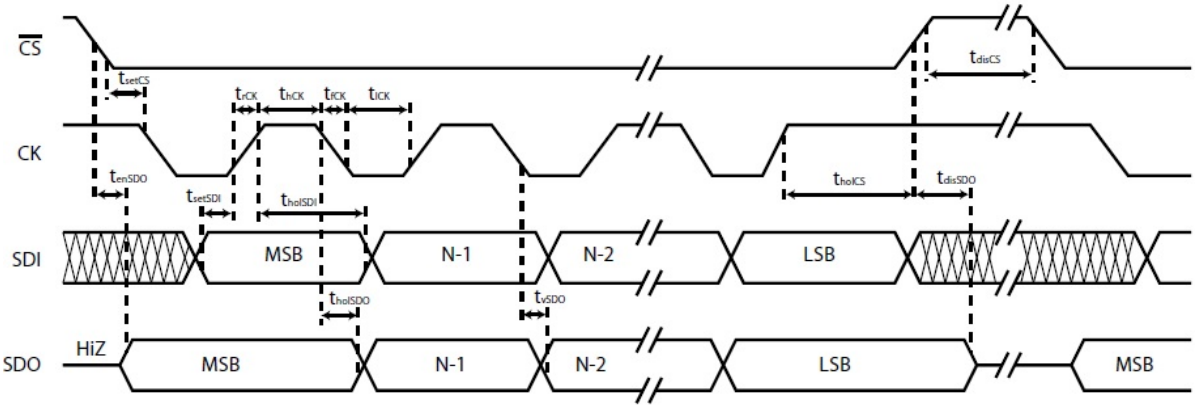
- PWM Input Interfaces
- Current Level Interfaces
- Jumpers for the Reference Voltage Source
- Jumper for the Decay Mode
- USB Connector
- Power Connector Screw Terminal
- Motor Phase Screw Terminals
- Measurement Loops
- Test Software
- Microstepping Resolution controlled by μC

5.5 Microstepping Functions

L6470 Microstepping Functions

With this IC a microstepping resolution of 128 microsteps can be realized. The command actualization can be performed with the following timing specifications:

> *The integrated 8-bit serial peripheral interface (SPI) is used for a synchronous serial communication between the host microprocessor (always master) and the L6470 (always slave).*
> *The SPI uses chip select (CS), serial clock (CK), serial data input (SDI) and serial data output (SDO) pins. When CS is high, the device is unselected and the SDO line is inactive (high-impedance).*
> *The communication starts when CS is forced low. The CK line is used for synchronization of data communication.*
> *All commands and data bytes are shifted into the device through the SDI input, most significant bit first. The SDI is sampled on the rising edges of the CK.*
> *All output data bytes are shifted out of the device through the SDO output, most significant bit first. The SDO is latched on the falling edges of the CK. When a return value from the device is not available, an all zero byte is sent.*
> *After each byte transmission the CS input must be raised and be kept high for at least t_{disCS} in order to allow the device to decode the received command and put the return value into the SHIFT register.* [9]



Source: [9]

Figure 20: SPI1 Timing Diagram

Most of the commands consist of three bytes. Additionally, there always has to be some time between commands. The following calculation gives the maximal actualization rate for commands:

$$T_{\text{SCK\_min}} = \frac{1}{f_{\text{SCK\_max}}} = \frac{1}{5MHz} = 200ns \tag{2}$$

$$T_{\text{Command\_min}} = 3 \cdot (8 \cdot T_{\text{SCK\_min}} + T_{\text{disCS\_min}}) = 3 \cdot (8 \cdot 200ns + 800ns) = 7.2\mu s \tag{3}$$

$$f_{\text{Command\_max}} = \frac{1}{T_{\text{Command\_min}}} = \frac{1}{7.2\mu s} \approx 139kHz. \tag{4}$$

This frequency can be compared to a PWM update frequency.

DRV8841 Microstepping Functions

The big advantage of this IC is that the degree of microstepping is controlled by the μC. Also the PWM update frequency is set by the μC but there is a limit of 100 kHz.

5.6 Final Selection

Both of the selected driver ICs are available in a SSOP with 28 pins. So they are equal according to build space requirements.
The total power dissipation varies with the ambient temperature and is mostly proportional to the current level for both of the driver-ICs.
They are also equal according to logic supply voltage, peak current and bridge configuration.
The driver-IC from STMicroelectronics offers a fully integrated solution, which can control a motor without an additional μC. The disadvantage of this IC is, that **not all boundary conditions can be held**. For direct motor control the console-side has to get adapted, because the L6470 driver only can be operated when the supervising μC on the console-side sends high-level motor control commands.
The **most important factor** is the microstepping resolution. The L6470 driver offers a maximal resolution of 128 microsteps, the DRV8841 driver does not have a limit concerning the degree of microstepping. An advantage of the L6470 driver would be the higher PWM update rate, but the SPI1 update rate is just 25 kHz.
With this knowledge it is obvious that the **DRV8841 driver IC has to be selected**.

5.7 Supervision Functions

5.7.1 Overcurrent Protection

> *An analog current limit circuit on each FET limits the current through the FET by removing the gate drive. If this analog current limit persists for longer than the OCP time, all FETs in the H-bridge will be disabled and the nFAULT pin will be driven low. The device will remain disabled until either nRESET pin is applied, or VM is removed and re-applied.*
> *Overcurrent conditions on both high and low side devices; i.e., a short to ground, supply, or across the motor winding will all result in an overcurrent shutdown. Note that overcurrent protection does not use the current sense circuitry used for PWM current control, and is independent of the ISENSE resistor value or VREF voltage.* [8]

When the nFAULT pin is driven low, the μC will pass through a failure message to the console-side.

5.7.2 Thermal Shutdown

> *If the die temperature exceeds safe limits, all FETs in the H-bridge will be disabled and the nFAULT pin will be driven low. Once the die temperature has fallen to a safe level operation will automatically resume.* [8]

5.7.3 Undervoltage Lockout (UVLO)

> *If at any time the voltage on the VM pins falls below the undervoltage lockout threshold voltage, all circuitry in the device will be disabled and internal logic will be reset. Operation will resume when VM rises above the UVLO threshold.* [8]

The typical UVLO threshold is 7.8 V.

5.7.4 Thermal Protection

> *The DRV8841 has thermal shutdown (TSD) as described above. If the die temperature exceeds approximately 150°C, the device will be disabled until the temperature drops to a safe level.*
> *Any tendency of the device to enter TSD is an indication of either excessive power dissipation, insufficient heatsinking, or too high ambient temperature.* [8]

5.7.5 Stall Detection

The concept for stall detection in ultrasound probes is a quite simple one. A hall-sensor detects the angular position of the rotor and sends a signal to the supervising μC on the console-side. This signal has to be sent in a specific time window. When the signal is sent outside this time window, the μC interprets this error as stall and a restart routine is initialized.

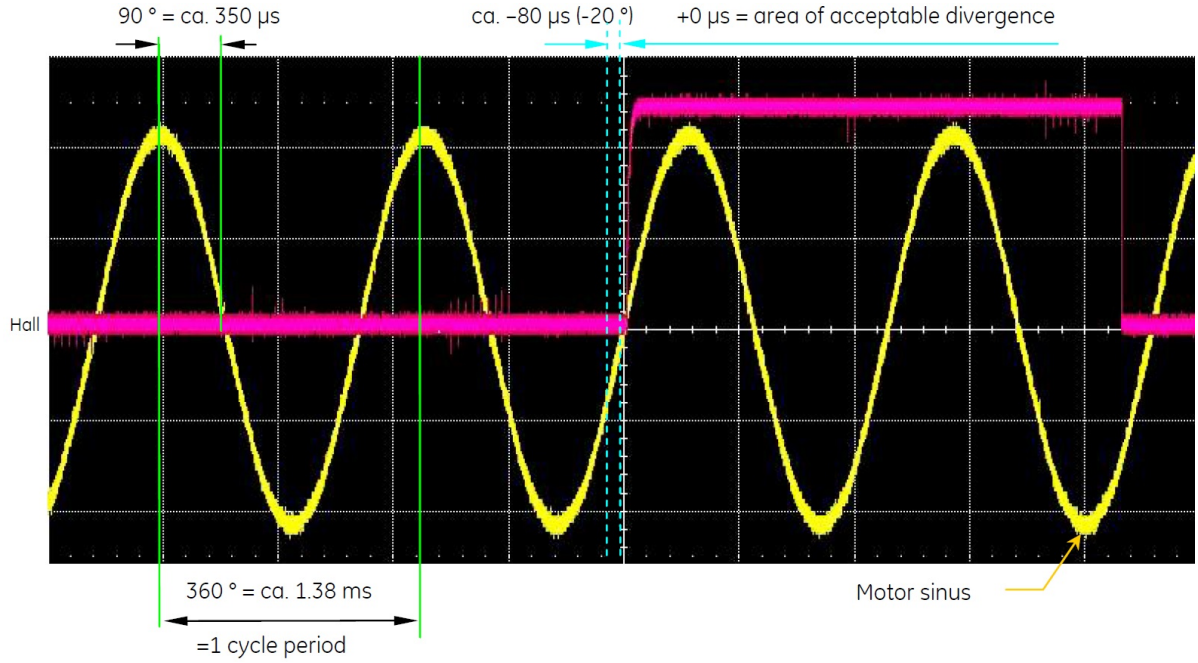
6 Signal Processing

6.1 Integrated Development Environment (IDE)

Microchip offers a IDE especially for their devices. The actual version is called MPLAB X. This IDE can be combined with different compilers, offering the possibility to use freeware or special compilers for Microchip devices.

6.2 Compiler

There are several freeware compilers. The disadvantage of freeware compilers is that they are designed for numerous processors. As a result, it is possible that the same compiled code does not give the same performance on different devices. To receive optimal results, special compilers for Microchip devices are needed.



Source: [11]

Figure 21: Hall Signal

For the evaluation μC (dsPIC33FJ128MC802) the compiler **C30** from Microchip can be used. The prototyping μC (dsPIC33EP256MC202) is newer than the one used for evaluation issues. The C30 compiler does not support this device but Microchip offers the **XC16** compiler too. The used assembler version is called **ASM30**.

6.3 Firmware

There are different methods to write firmware. The easiest way is to use a **high-level language like C or C++** and let the compiler do the rest. Another method is to program the firmware on **assembler** level. This means that the programmer uses the processor's instruction set to write programs.
According to performance, programming on assembler level is often better because the programmer just uses the best suited instructions. It is possible that high level compilers add instructions which are not needed.
The advantage of using high-level languages is the programming comfort. The programmer can use the same language for many different devices.
The most important firmware parts can be found in the chapter "List of Code-Listings" in the appendix.

6.3.1 Firmware for the Evaluation of the L6470 IC

The usage of this driver-IC brings some competitions with it. First of all, there are the high-level operation commands via SPI. The most important commands for this project are:

- SetParam(PARAM,VALUE)
- GetParam(PARAM)
- Run(DIR,SPD)
- Move(DIR,N_STEP)
- SoftStop()
- HardStop()
- SoftHiZ()
- HardHiZ()

For evaluation purposes, the μC on the console-side cannot be adapted. So the driving signals have to be generated by an additional μC. The problem is the calculation of these driving signals. The reference values of the SIN- and COS-current are transferred via SPI. This can be explained by the following figure.

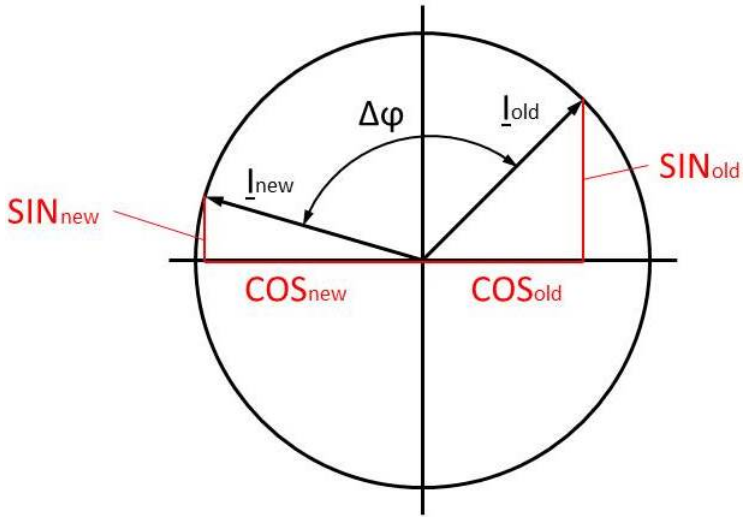


Figure 22: Current Vectors

The number of microsteps, which is needed to reach a certain angular position within an incremental time step, is calculated using the following equations:

$$n_{\mu\text{Steps}} = N_{\mu\text{Steps}} \cdot \frac{\Delta\varphi}{2 \cdot \pi} \tag{5}$$

$$\Delta\varphi = \varphi_{\text{new}} - \varphi_{\text{old}} = arcsin\left[\frac{SIN_{\text{new}}}{RES/2}\right] - arcsin\left[\frac{SIN_{\text{old}}}{RES/2}\right] \tag{6}$$

With these equations, and under consideration of the quadrants of the SIN- and COS-current-values, the angles of the current pointers can be calculated. The magnitude of the reference pointer is for evaluation issues always one. Although these calculations are trivial, some follow-up problems have to be solved. The selected μC is designed for the calculation of integers. It is also possible to calculate floating point values, but the calculations lasts too long for the timing specifications of the SPI1-module ($T_{\text{SPI1\_Cycle}} = 20\mu s$).
The solution of this problem is a **first-order Taylor series approximation**.

$$arcsin\left[\frac{SIN_{\text{i}}}{RES/2}\right] \cong \frac{SIN_{\text{i}}}{RES/2} + 1/6 \cdot \left[\frac{SIN_{\text{i}}}{RES/2}\right]^3 + 3/40 \cdot \left[\frac{SIN_{\text{i}}}{RES/2}\right]^5 + ... \tag{7}$$

To keep the calculation simple, just the first term of the Taylor series approximation is used.

$$arcsin\left[\frac{SIN_{\text{i}}}{RES/2}\right] \cong \frac{SIN_{\text{i}}}{RES/2} \tag{8}$$

The value which has to be set in the SPEED-register of the driver-IC, can be calculated/approximated by following equation:

$$SPEED = N \cdot N_{\mu\text{Steps}} \cdot \frac{2^{28} \cdot 250}{10^9} \cdot \frac{SIN_{\text{new}} - SIN_{\text{old}}}{RES/2} \tag{9}$$

With this value and the knowledge of the rotational direction resulting from the old and new SIN- and COS-reference values the Run(DIR,SPD)-method can be executed.
It is clear that this variant **responds slowly** to changes coming from the supervising μC because of the signal conversion. Furthermore, current regulation is not possible.
Without the adaption of the interface between the supervising μC on the console-side and the motor drive controller, **this driver cannot be used**.

6.3.2 Firmware for the Evaluation of the DRV8841 IC

Because of the PWM-interface between μC and driver-IC, the **driver-IC reacts immediately** to changes coming from the μC. The only thing that is done in the driver-IC is the amplification of the PWM signals. This enables high dynamic operation of the motor, which is needed for several examination modes.
The operation of this driver-IC is quite simple. Every transferred value via SPI1 can be directly interpreted as SIN- or COS-duty cycle. This means that the transferred value is compared to a maximal value and the duty cycle can be calculated by a simple division. The division factor depends on the resolution of the transferred values and the ADC-resolution.

$$ADC\_RESOLUTION\_FACTOR = \frac{ADC\_RESOLUTION\_MAX}{ADC\_RESOLUTION} \tag{10}$$

Another advantage results from the chosen operation mode of the PWM module. It is called **complementary mode**. This means that the module's low channel always is the inverse of the high channel. When a current value of zero is desired, the duty cycle has to be 50%. For the maximal positive value the duty cycle is 100%. The maximal negative value can be reached by setting the duty cycle to 0%. So it can be seen that the added **offset does not have to be removed**.
More details can be found in the code listings in the appendix.

6.3.3 Prototype Firmware

The prototype firmware is quite similar to the evaluation firmware of the DRV8841. The used μCs are similar but not equal. This means that some registers have different names, or that initialization routines have to be done in different ways.
A big advantage is the higher oscillator frequency of the prototyping μC. This also makes higher PWM frequencies possible.
For more details see the code listings in the appendix.

7 Prototyping

7.1 Console Simulation

The number of consoles, which can be used for evaluation issues, is very small. But there are several adapter boards and other devices, which are able to simulate different functions of a console.

7.1.1 SPI Simulation

One of this adapter boards is able to simulate the SPI, which sends the current reference values for the motor-drive-controller.

The needed software for the operation of this adapter board was developed for **Windows XP**. To be able to operate the board with **Windows 7** the application has to be compiled for Windows 7. To do this **Microsoft Visual Studio 2008 Professional** can be used.

7.1.2 Power Supply Simulation

To simulate the PSU located on the console, a common laboratory PSU is used. It offers three different stabilized voltages with current limitation. Then, it is possible to supply the console simulation board, the motor-driver-controller-board and an additional device if needed.

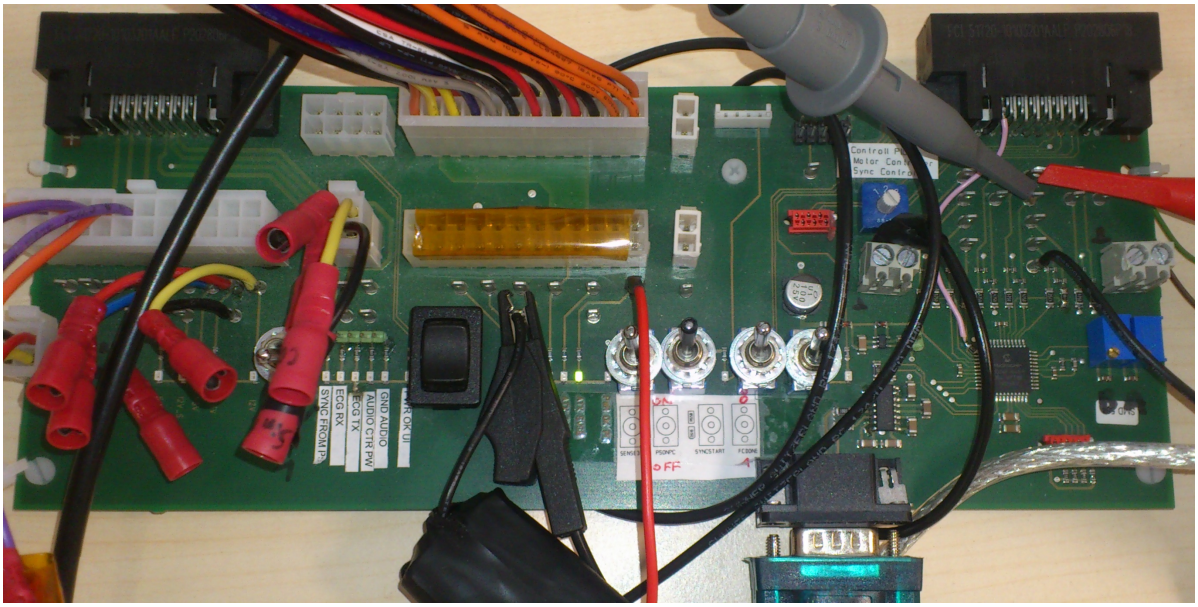


Figure 23: SPI Simulation Board

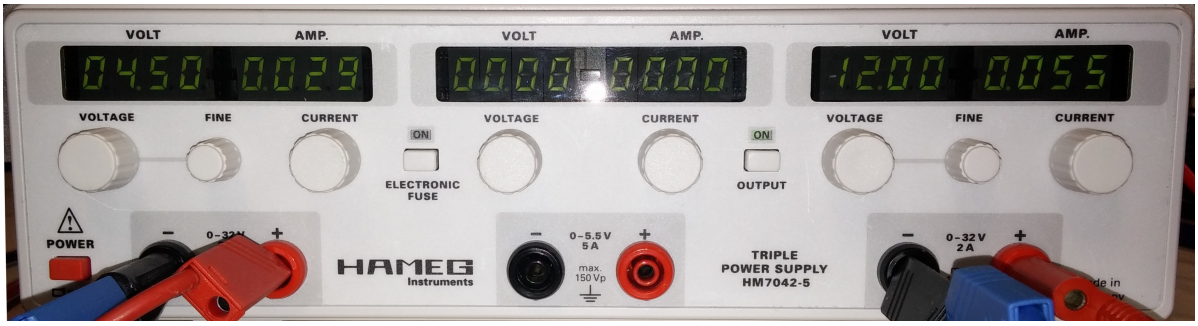


Figure 24: Power Supply Unit

7.2 Schematics

For prototyping issues, many different development tools are available. The use of freeware-tools would be the best way to keep development costs low. Most of the development tool vendors offer freeware versions of their products. The disadvantage of these versions is that not all functions are available.
For schematics design, a freeware-version of EAGLE meets this project's requirements best. The used version is 6.5.0.
The complete schematics can be found in the appendix.

7.3 PCB Prototyping

In addition to schematics design, EAGLE offers a module for PCB design. The disadvantage of the freeware-version is that the maximal count of signal layers is two. So another tool has to be used.
Most of the electrical engineers at GE Healthcare use the Cadence Suite for PCB design. Additionally, most of the time there is an unused license at the site in Zipf.

7.3.1 Position Plans

To get a very small board, both sides are used for the placement of parts.

Position Plan Bottom Layer

At this side of the board big components like the inductors and big ceramic capacitors for filtering are located. It makes sense to use this side of the board because it is uncertain whether the filters are needed for the final version of the motor-drive-controller.

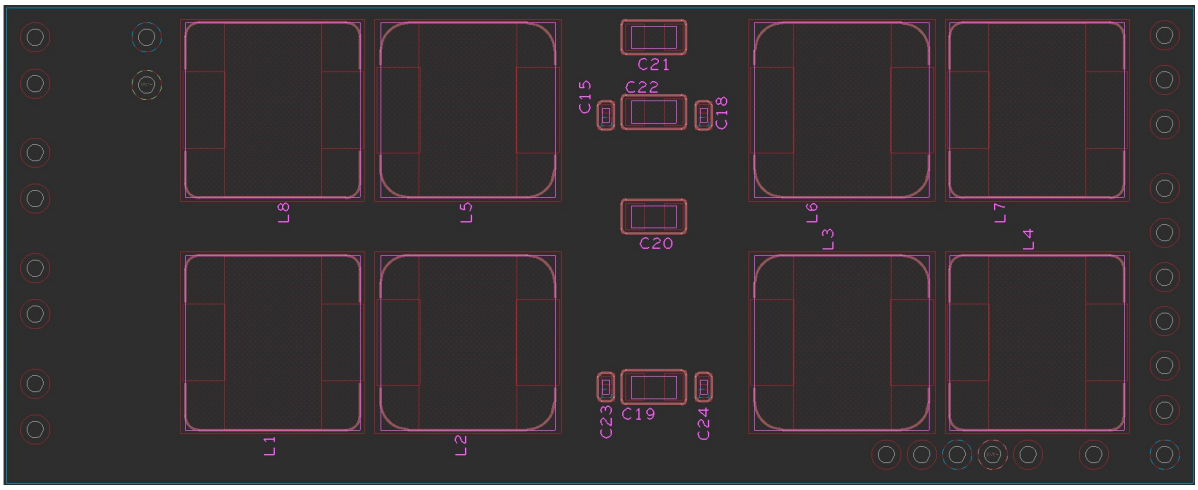


Figure 25: Position Plan Bottom Layer

Position Plan Top Layer

All other parts are located on the top layer of the board. The jumper positions can be changed to test different functions of the board. There are the following test configurations:

- Motor current comes from the console
- Motor current comes from the probe-sided motor-drive-controller
  - Filters on
  - Filters off

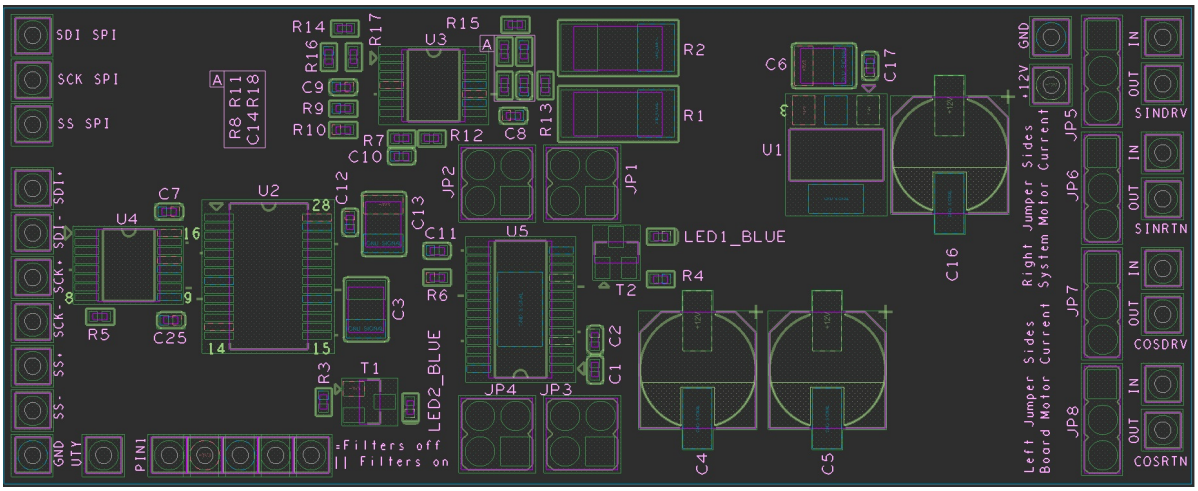


Figure 26: Position Plan Top Layer

7.3.2 Signal Layers

For small boards it is necessary to place the components as close as possible. In this project's case, both sides of the board are used. With a rising amount of components, the number of signal lines also rises. One difficulty in miniaturization is the placement of these lines. When there is not enough space to place all lines on one layer, additional signal layers can be added. For this prototype two additional layers are used.

Signal Colors

- Ground
- +3.3 V
- +12 V
- Motor Lines

Layer 2: Ground Layer

This layer was added to have a ground potential with low impedance.

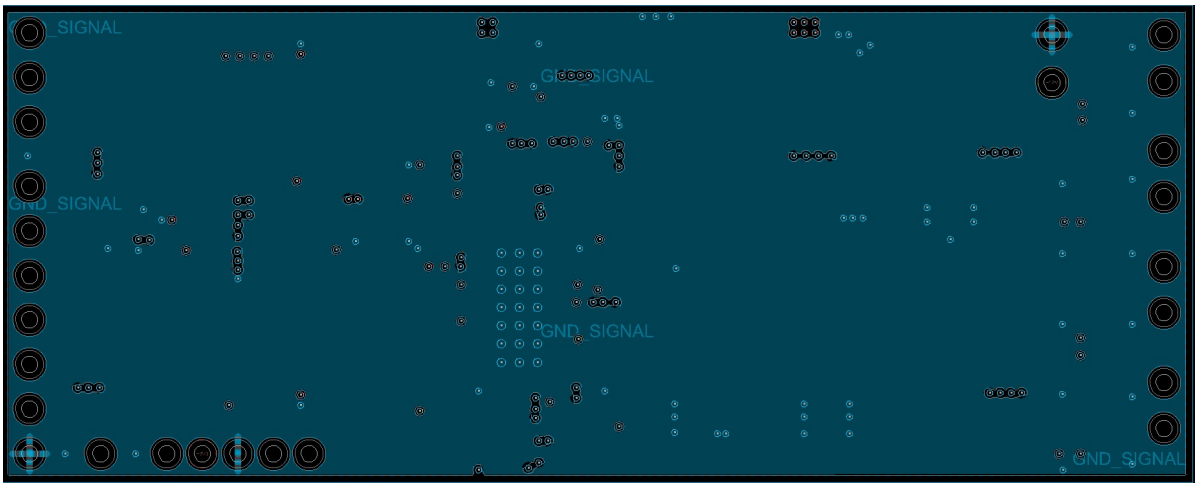


Figure 27: Layer 2: Ground Layer

Layer 3: Signal Layer

The main purpose of this layer is linking the components located on the surface. Furthermore, this layer is used to distribute the logic supply voltage of 3.3 V, the ground signal and the motor lines.

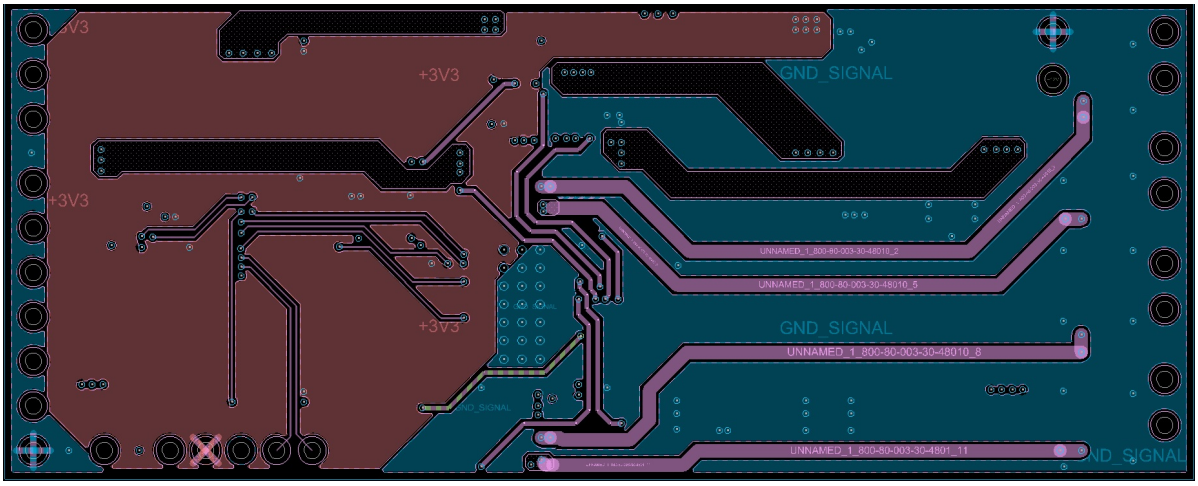
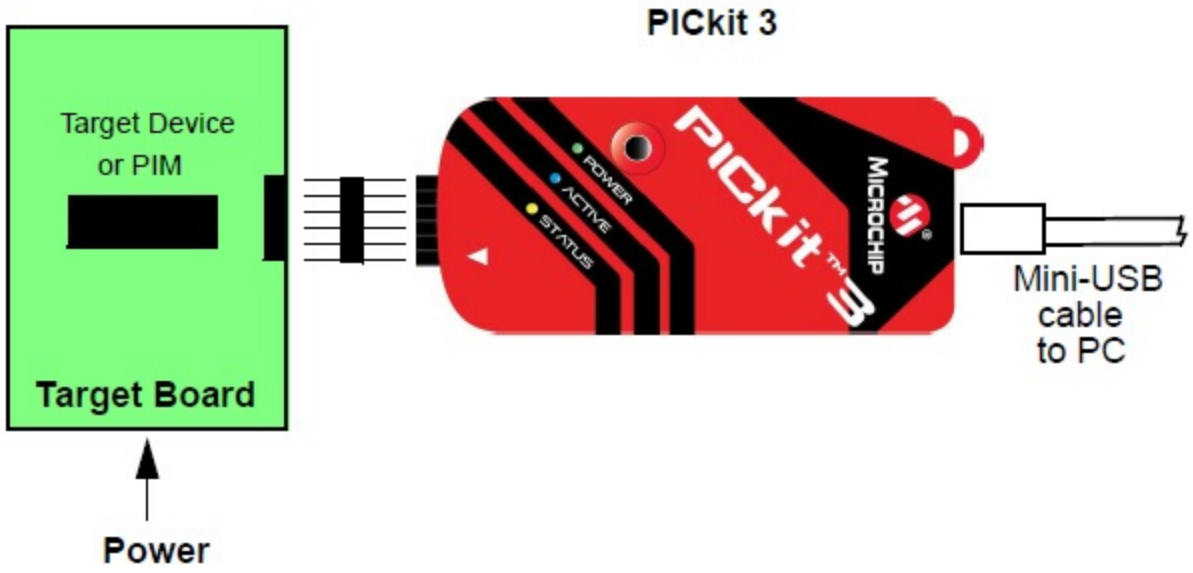


Figure 28: Layer 3: Signal Layer

7.4 In-Circuit Programming/Debugging

There always has to be a way to program and debug a μC (at least one time). On the μC-evaluation board (Microstick II) the circuitry for programming and debugging is integrated. The use of the same solution for the prototype board would not be ideal because of the **build space requirements**. So a different solution has to be found. The best build-space saving way is to use an external device like the **PICkit 3** from Microchip.



Source: [12]

Figure 29: PICkit 3 Programming/Debugging Topology

7.5 Low Voltage Differential Signaling (LVDS)

For different components of an ultrasound probe the common transmission technology between console and probe is LVDS. Also for the SPI of the motor-drive-controller this technology should be used.
The SPI consists of three signals lines:

- Serial Data Input (SDI)
- Serial Clock (SCK)
- Slave Select (SS)

As a result, six signal lines are needed in the cable:

- SDI+
- SDI-

- SCK+
- SCK-
- SS+
- SS-

7.6 Current Measurement Topology

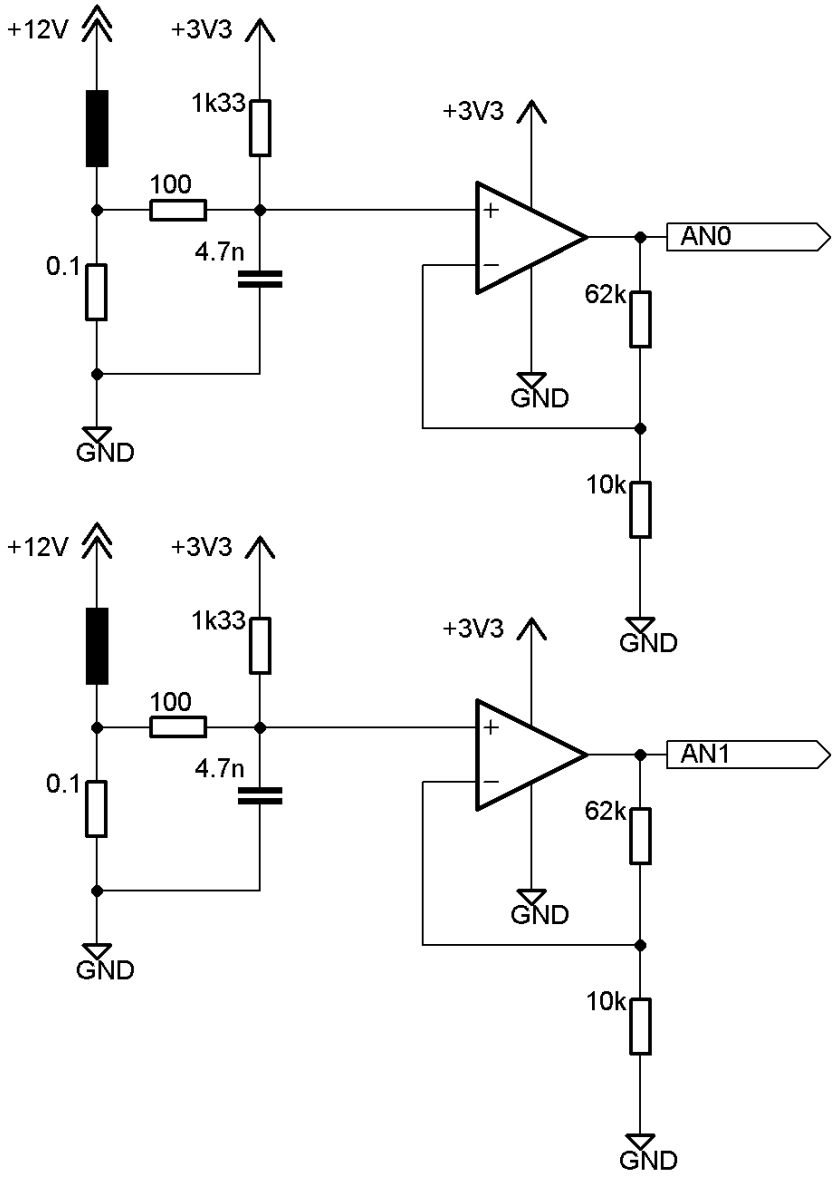


Figure 30: Current Measurement

7.6.1 Amplification Topology

There is a boundary condition for the topology of the current measurement coming from the console-side. The maximal current flowing through a motor winding should be 2.3 A.

$$I_{\max} = 2.3A \tag{11}$$

The value of the shunt resistor should be as small as possible to minimize dissipation.

$$R_{\text{Shunt}} = 0.1\Omega \tag{12}$$

This gives the following voltage drop at the shunt resistor:

$$U_{\text{Shunt}} = \pm 0.23V \tag{13}$$

To avoid negative voltages at the ADC-module of the μC, half of the logic supply voltage is added to the measured voltage.

$$U_{\text{Offset}} = \frac{U_{\text{Logic\_Supply}}}{2} = \frac{3.3V}{2} = 1.65V \tag{14}$$

With this data the voltage amplification of the operation amplifier can be calculated.

$$A_{\text{U}} = \frac{U_{\text{out}}}{U_{\text{in}}} = \frac{U_{\text{Logic\_Supply}} - U_{\text{Offset}}}{U_{\text{Shunt}}} = \frac{1.65V}{0.23V} = 7.17 \approx 7.2 \tag{15}$$

For an ideal, non-inverting operational amplifier the following configuration results:

$$A_{\text{U}} = 7.2 = 1 + \frac{R_{\text{f}}}{R_{\text{N}}} \tag{16}$$

$$\rightarrow \frac{R_{\text{f}}}{R_{\text{N}}} = 6.2 \tag{17}$$

The offset voltage can be realized with a voltage divider.

$$R_{\text{Voltage\_Divider}} = \frac{U_{\text{Logic\_Supply}} \cdot R_{100} - U_{\text{Shunt\_Offset}} \cdot R_{100}}{U_{\text{Shunt\_Offset}}} = \frac{3.3V \cdot 100\Omega - 0.23V \cdot 100\Omega}{0.23V} \approx 1335\Omega \tag{18}$$

7.7 Filter Topologies

As described in the section 'Low-pass Filtering', different filter topologies should be evaluated.

7.7.1 Motor Winding acts as Low-pass Filter

The easiest was to smooth the motor current is to use the inductance of the motor winding as low-pass filter. The evaluation of the image quality will give feedback about the performance of this topology. The action of the motor inductance can be explained by the following equations.

$$u_{\mathrm{L}} = L \cdot \frac{di}{dt} \tag{19}$$

By integrating this equation in time, the following result can be found for the motor current:

$$i = \frac{1}{L} \cdot \int u_{\mathrm{L}} dt \tag{20}$$

The inductance of the motor winding has an averaging character. The measured results also show this effect.

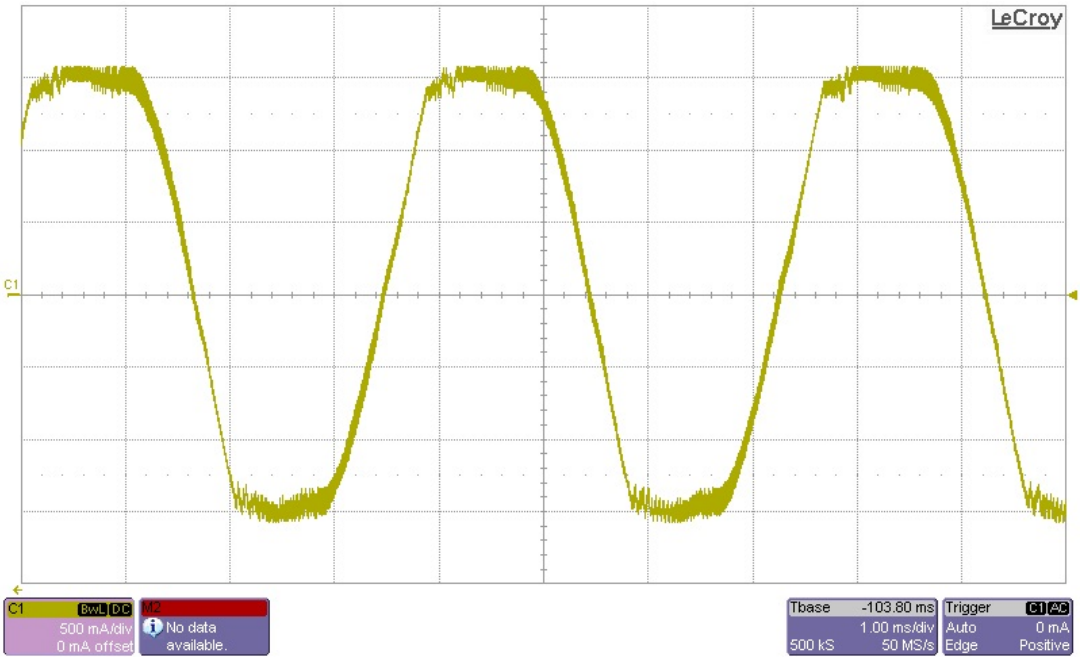


Figure 31: Motor Current without additional Filter, 1

7.7.2 Motor Winding with Second Order Filter

There are different types of filters. The most common ones are RC, RL and LC. For our issues there has to be a minimum power dissipation. As a result, the usage of RC and RL does not make sense. Furthermore, there is a difference between real and ideal parts. For example, an ideal inductor would have no serial resistances. So there can never be an ideal LC-topology but the behavior is quite similar.

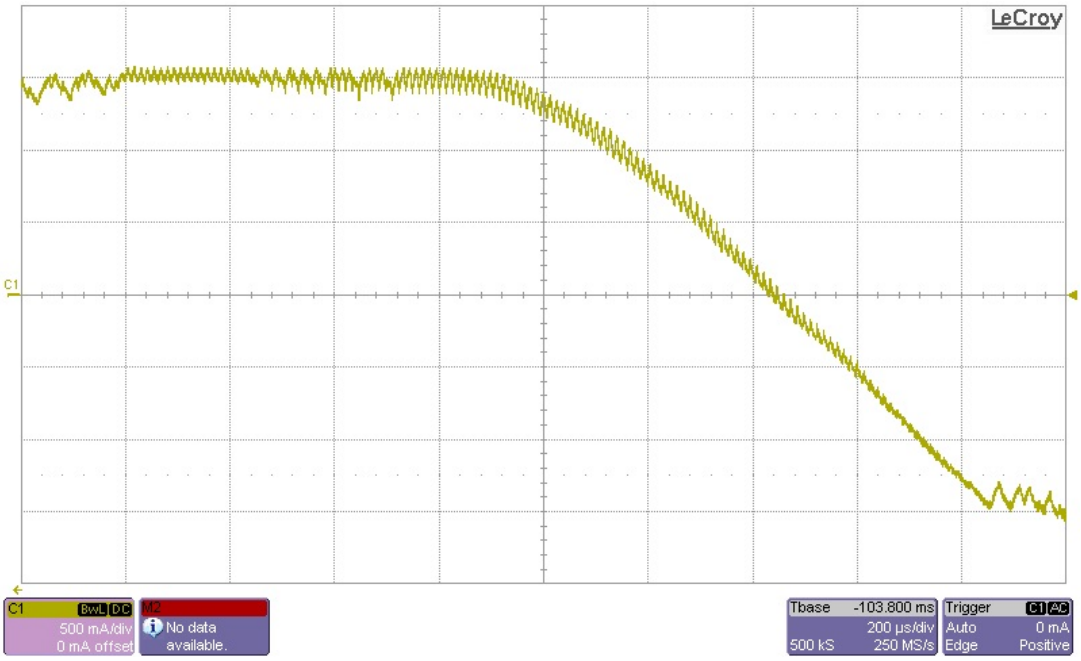


Figure 32: Motor Current without additional Filter, 2

To realize this filter **two inductors and one capacitor** are added. This gives a second order filter. **Second order** means that after the cutoff-frequency the magnitude drops **-40 dB/decade**. By choosing the part parameters, the cutoff frequency of the filter can be set.

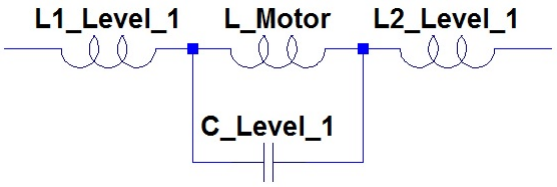


Figure 33: Second Order Filter

This configuration is symmetrical. To be able to calculate the part parameters, just one half of the filter can be used. This gives a simple LC-configuration.

The cutoff-frequency can be calculated by the following equation.

$$f_{\text{Cutoff}} = \frac{1}{2 \cdot \pi \cdot \sqrt{L \cdot \frac{C}{2}}} \tag{21}$$

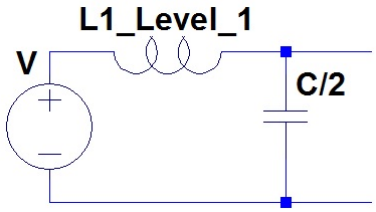


Figure 34: Half of Second Order Filter

Calculation Example

$$L = 330\mu H \tag{22}$$

$$C = 3.3\mu F \tag{23}$$

$$\tag{24}$$

$$f_{\text{Cutoff}} = \frac{1}{2 \cdot \pi \cdot \sqrt{L \cdot \frac{C}{2}}} = \frac{1}{2 \cdot \pi \cdot \sqrt{330\mu H \cdot 1.65\mu F}} \approx 6821Hz \tag{25}$$

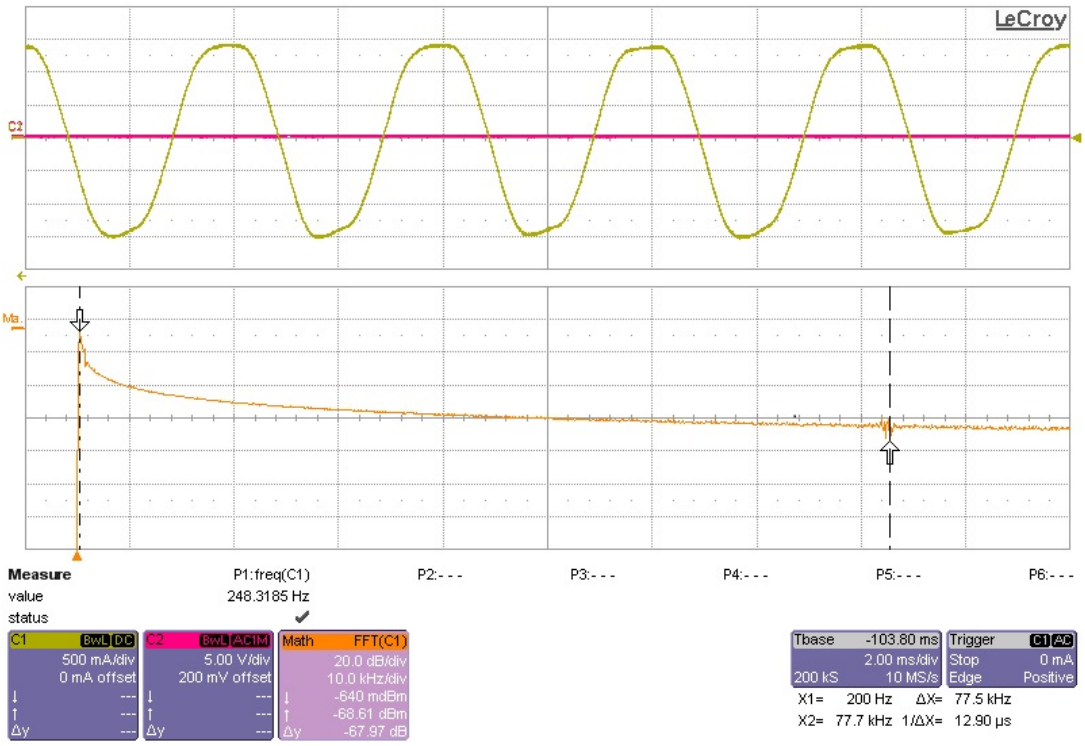


Figure 35: Motor Current with additional Second Order Filter

As expected, the measured data shows that the PWM ripple is nearly removed. By using the Fast Fourier Transformation (FFT) it can be seen that there are still ripples at the PWM frequency but the magnitudes are very small.

7.7.3 Motor Winding with Fourth Order Filter

The next logic step is to increase the degree of the filter for a bigger amplitude drop above the cutoff frequency. The addition of a second filter stage gives a **fourth order** filter, which means a drop of **-80 dB/decade** in the Bode-plot.

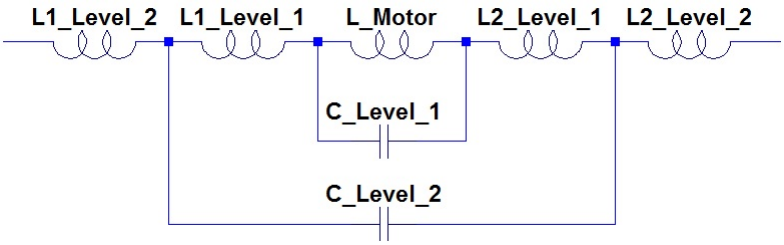


Figure 36: Fourth Order Filter

This configuration is symmetrical, too. Hence, just half of the filter has to be calculated.

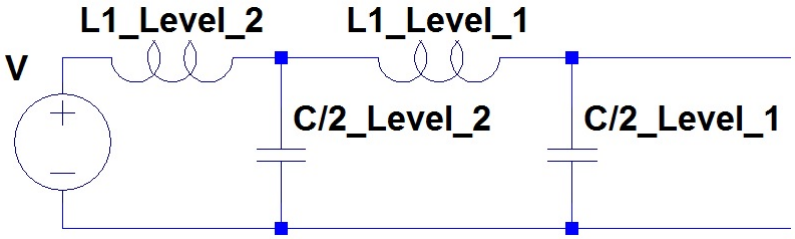


Figure 37: Half of Fourth Order Filter

The part parameters of the second filter stage should differ to the first level.
For this reason, the inductance of the second filter level is just ten percent of the first level's inductance. To achieve the same cutoff-frequency for both levels, the capacitor of the second stage has to be ten times higher than in the first stage.
The evaluation of the ultrasonic image quality shows which degree of filtering is needed.

8 Current Control

8.1 Frequency Response

To get the possibility of current control the frequency response of the motor winding and the filters has to be considered. Therefore, the current reference values transferred via SPI have to

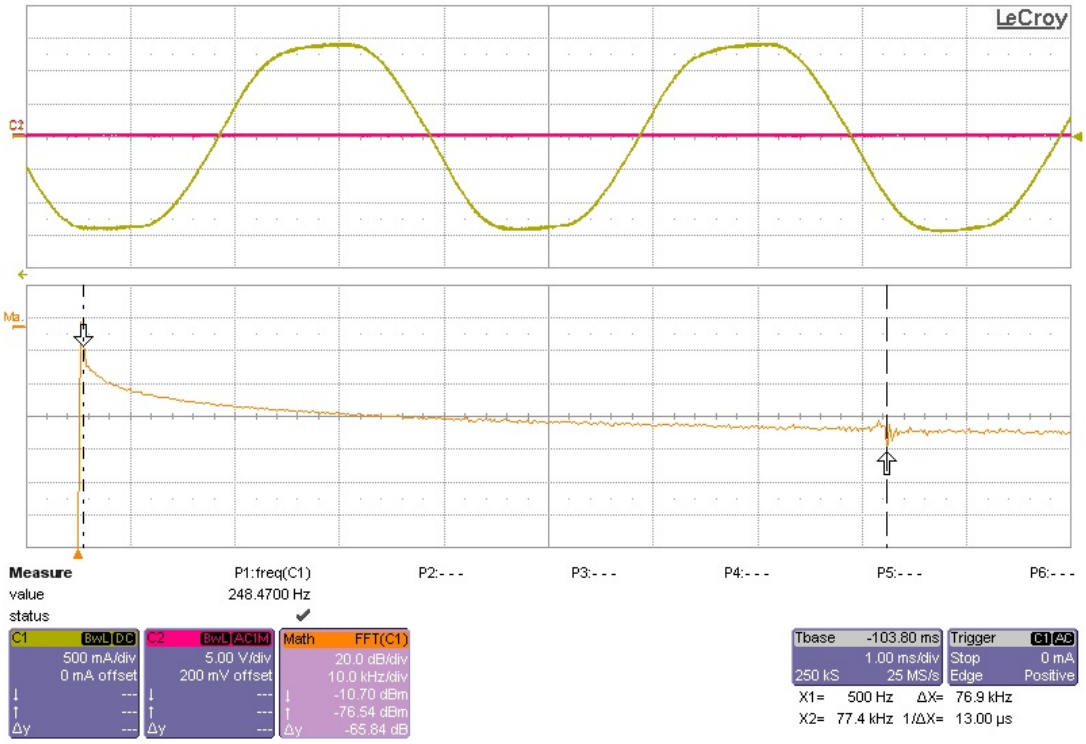


Figure 38: Motor Current with additional Fourth Order Filter

be adapted with a digital filter.

8.1.1 Motor Current Sinus Frequency Calculation

First of all, the sinus frequency of the motor current has to be calculated. This can be done by different ways. Two very simple ways should be explained by the following equations.

Search for repeated Values

The simplest way of finding the motor current sinus frequency is to look for repeated values. This becomes possible because the transferred values result from look-up tables.

$$f_{\text{Sinus}} = \frac{f_{\text{Sample}}}{N_{\text{Samples/Period}}} \tag{26}$$

Calculation with the Arc Cosine-Function

As a second, calculation-intense option the cosine-function can be used to calculate the motor current sinus frequency. This can be done with the following equations:

$$y_{\text{k-1}} = A \cdot cos(-\omega + \phi) = A \cdot cos(\omega) \cdot cos(\phi) + A \cdot sin(\omega) \cdot sin(\phi) \tag{27}$$

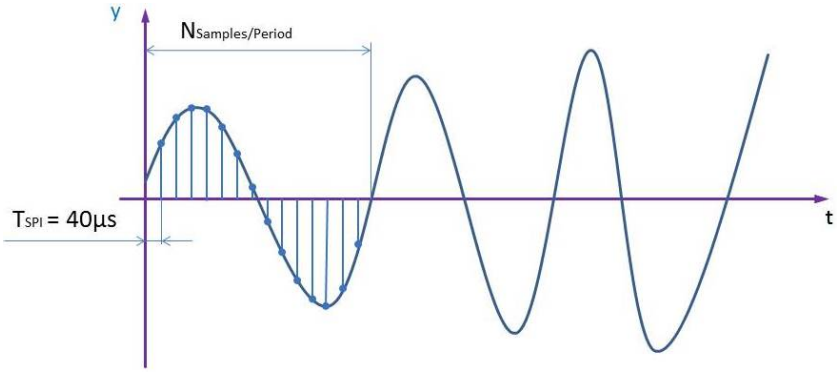


Figure 39: Motor Sinus Frequency Calculation, Method 1

$$y_{\mathrm{k}} = A \cdot cos(\phi) \tag{28}$$

$$y_{\mathrm{k+1}} = A \cdot cos(\omega + \phi) = A \cdot cos(\omega) \cdot cos(\phi) - A \cdot sin(\omega) \cdot sin(\phi) \tag{29}$$

$$y_{\mathrm{k-1}} + y_{\mathrm{k+1}} = 2 \cdot A \cdot cos(\omega) \cdot cos(\phi) = 2 \cdot y_{\mathrm{k}} \cdot cos(\omega) \tag{30}$$

$$\rightarrow \boxed{f_{\mathrm{Sinus\_k\text{-}1}} = \frac{f_{\mathrm{Sample}}}{2 \cdot \pi} \cdot arccos\left(\frac{y_{\mathrm{k\text{-}2}} + y_{\mathrm{k}}}{2 \cdot y_{\mathrm{k\text{-}1}}}\right)} \tag{31}$$

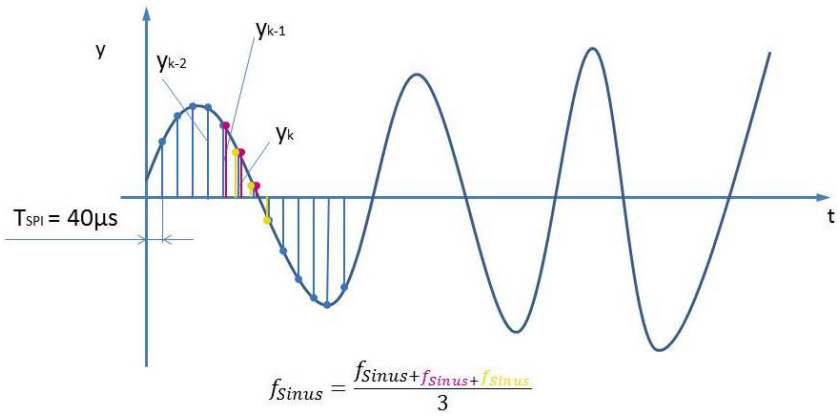


Figure 40: Motor Sinus Frequency Calculation, Method 2

8.1.2 Frequency Response Values

With the knowledge of the motor current sinus frequency and the part parameters of the filter components the frequency response of the system can be calculated. A very simple option to realize this would be the usage of look-up tables.

8.2 Controller Selection

To achieve the desired parameters for the closed loop current control, a few calculations have to be done. With the knowledge of motor parameters, like winding resistance, inductance and rated current the controller design is quite simple. For this application the use of **two PI-controllers**, one for each winding, is a good choice.
First of all, the transfer function of one motor winding is needed.

Motor Transfer Function

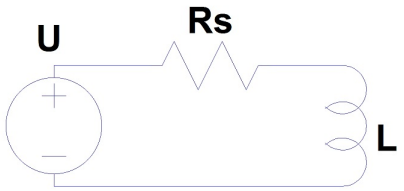


Figure 41: Motor Winding Schematics

$$V \cdot u = i(t) \cdot R_S + L \cdot \frac{di(t)}{dt} \tag{32}$$

The Laplace-transformation of the equation above is:

$$V \cdot u(s) = i(s) \cdot R_S + L \cdot i(s) \cdot s \tag{33}$$

As a result, the transfer function of one motor winding is:

$$G_M = \frac{i(s)}{u(s)} = \frac{V}{L \cdot s + R_S} \tag{34}$$

Controller Transfer Function [13]

As mentioned before, a PI-controller fits best to the requirements. The generalized continuous transfer function of such a controller is:

$$G_{PI}(t) = \frac{i(t)}{e(t)} = K_P \cdot \left[e(t) + \frac{1}{T_N} \cdot \int_0^t e(\tau) d\tau \right] \tag{35}$$

The Laplace-transformation for this equation is:

$$G_{PI}(s) = \frac{\bar{i}(s)}{\bar{e}(s)} = \frac{K_P}{T_N} \cdot \left(\frac{1 + T_N \cdot s}{s} \right) \tag{36}$$

Now the controller parameters K_P and T_N can be found by using the boundary conditions.

$$K = K_P / T_N \tag{37}$$

$$T_N = L/R_S \tag{38}$$

This gives the continuous controller equation:

$$G_{PI}(s) = K \cdot \left(\frac{1 + L/R_S \cdot s}{s} \right) \tag{39}$$

Discretization of the Transfer Functions

For the discretization of the transfer functions the so called "'Tustin"'-discretization should be used to achieve simple results.

$$G = \frac{K}{R_S} \tag{40}$$

$$p_1 = L + R_S \cdot \frac{T_{Sample}}{2} \tag{41}$$

$$p_2 = L - R_S \cdot \frac{T_{Sample}}{2} \tag{42}$$

$$G_{PI}{}^*(z) = G_{PI}(s)\big|_{s=\frac{2}{T_{Sample}} \cdot \frac{z-1}{z+1}} = K \cdot \frac{\frac{2 \cdot L \cdot (z-1)}{R_S \cdot T_{Sample} \cdot (z+1)} + 1}{\frac{2 \cdot L \cdot (z-1)}{R_S \cdot T_{Sample} \cdot (z+1)}} = G \cdot \left(\frac{p_1 \cdot z - p_2}{z-1} \right) \tag{43}$$

$$G_M{}^*(z) = G_M(s)\big|_{s=\frac{2}{T_{Sample}} \cdot \frac{z-1}{z+1}} = \frac{V/R_S}{L/R_S \cdot \frac{2}{T_{Sample}} \cdot \frac{z-1}{z+1} + 1} = \frac{V \cdot \frac{T_{Sample}}{2} \cdot (z+1)}{p_1 \cdot z - p_2} \tag{44}$$

Closed Loop Transfer Function

The system transfer function can be calculated the following way:

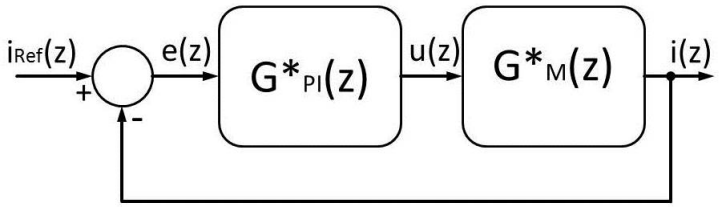


Figure 42: Current Control Block Diagram

$$G_{Loop}{}^*(z) = \frac{G_{PI}{}^*(z) \cdot G_M{}^*(z)}{1 + G_{PI}{}^*(z) \cdot G_M{}^*(z)} = \frac{1}{\frac{1}{G \cdot V} \cdot \frac{2}{T_{Sample}} \cdot \frac{z-1}{z+1} + 1} \tag{45}$$

$$G_{Loop}(s) = \frac{1}{\frac{R_S}{K \cdot V} \cdot s + 1} \tag{46}$$

Discrete Controller Implementation

$$G_{\mathrm{PI}}{}^{*}(z) = G \cdot \left(\frac{p_1 \cdot z - p_2}{z-1} \right) = G \left[p_1 \cdot \frac{z}{z-1} - p_2 \cdot \frac{z}{z \cdot (z-1)} \right] = \frac{u(z)}{e(z)} \tag{47}$$

$$\rightarrow u(z) = G_{\mathrm{PI}}{}^{*}(z) \cdot e(z) \tag{48}$$

$$u(z) = u(z) \cdot \frac{(z-1)}{(z-1)} = u(z) \cdot \frac{z}{z-1} - u(z) \cdot \frac{z}{z \cdot (z-1)} \bullet\!\!\sim\!\!\circ\, u_{\mathrm{k}} - u_{\mathrm{k\text{-}1}} \tag{49}$$

For the discrete gain, a multiplication factor of 4 is used in order to get more resolution from fixed point calculations by avoiding underflows. [6]

$$G_0 = 4 \cdot \frac{K}{R_{\mathrm{S}}} \tag{50}$$

$$\rightarrow \boxed{u_{\mathrm{k}} = G_0 \cdot p_1 e_{\mathrm{k}} - G_0 \cdot p_2 \cdot e_{\mathrm{k\text{-}1}} + u_{\mathrm{k\text{-}1}}} \tag{51}$$

This equation is the basic for the controlling algorithm.

Anti-Windup

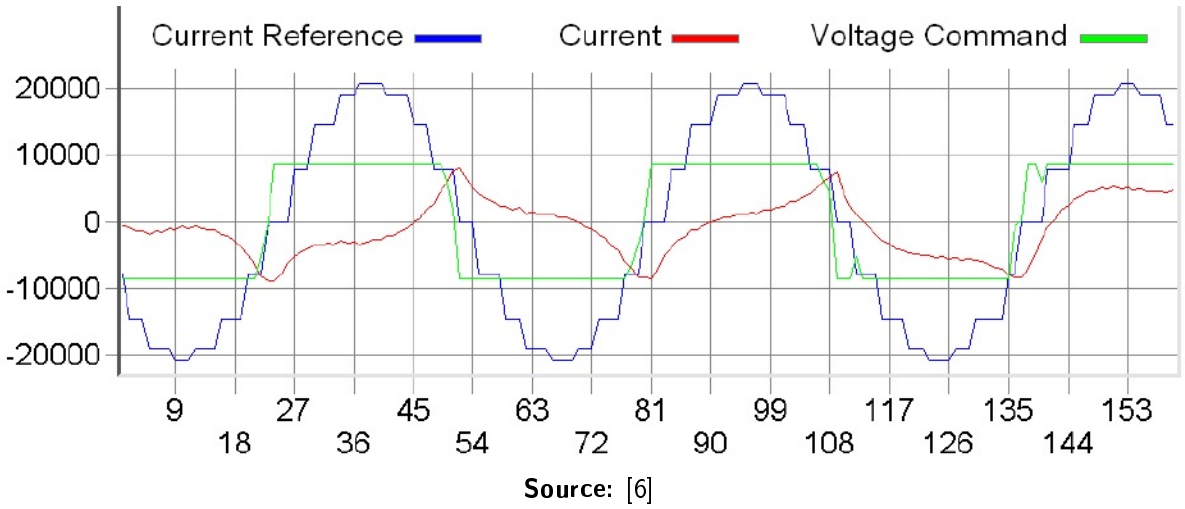
Limiting the controller output leads to a problem called accumulator wind-up. The output is saturated but the PI integrator accumulator keeps counting and grows until it eventually saturates. When the error is returning from the saturation area, the accumulator value is much higher than normal for that specific error value and, as a result, the system response slows. To prevent this effect, the accumulator also has to be compensated. [6]

This can be done by the addition of the so-called "anti-windup gain" G_{W} in the differences equation.

$$acc_{\mathrm{k}} = G_0 \cdot p_1 e_{\mathrm{k}} - G_0 \cdot p_2 \cdot e_{\mathrm{k\text{-}1}} + acc_{\mathrm{k\text{-}1}} - G_{\mathrm{W}} \cdot (acc_{\mathrm{k\text{-}1}} - u_{\mathrm{k\text{-}1}}) \tag{52}$$

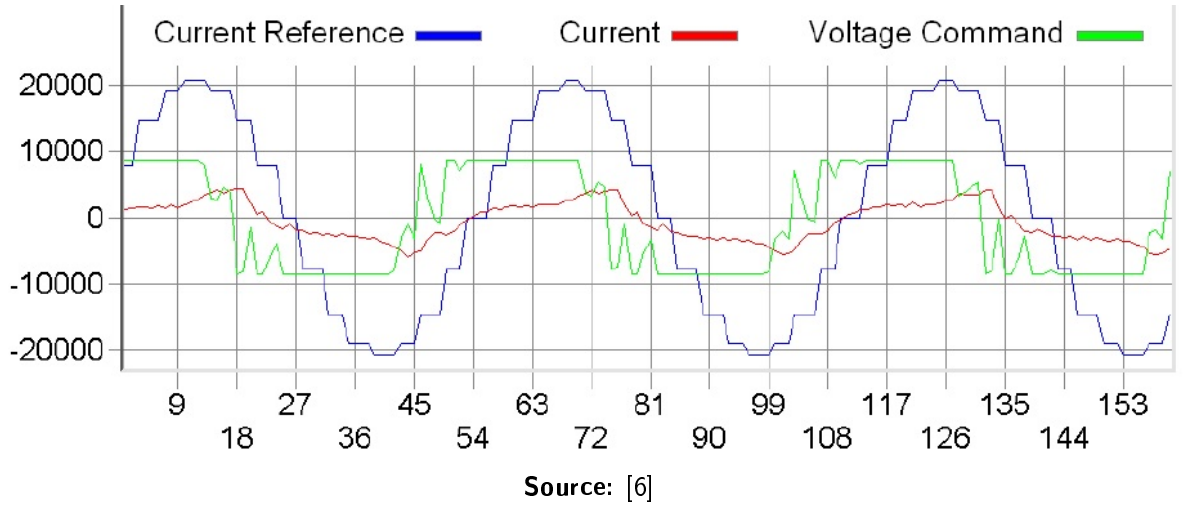
Phase Advance

By changing the value of the anti-windup gain, different controller behaviors are achieved. For low speeds, it is good to have a small gain so that the current tracks the reference as precise as possible. At higher speeds, when the DC_BUS voltage is not strong enough to bring the current to the reference value and the fast decay rate is not sufficient to bring the current down in the allocated time frame for one step, the anti-windup gain helps to change the phase of the current, thereby allowing transition to higher speeds, which otherwise could not be reached. Keeping the anti-windup gain low will result in the motor eventually stalling as the speed increases. [6]



Source: [6]

Figure 43: Low Anti-Windup Gain



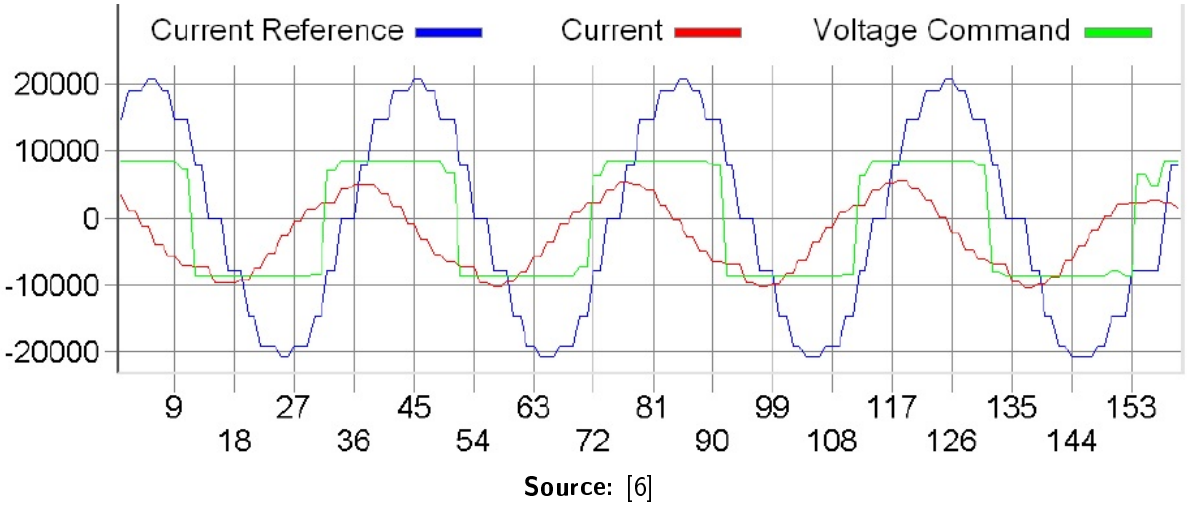
Source: [6]

Figure 44: High Anti-Windup Gain

The high anti-windup gain forces the controller output voltage to exit saturation sooner and therefore changes the phase of the winding current relative to the desired current. With this phase advance, the current has enough time to rise into the winding before the rotor pole reaches the energized stator pole. Further increasing the speed, the current amplitude keeps dropping until it eventually changes phase forced by the back-EMF. At this point, the current amplitude will begin to rise again and the phase advance and motor back-EMF work together to keep the motor running. [6]

The motor torque at 2400 RPM is strong enough to operate the motor under a light load. As a comparison, the maximum speed achieved in the open loop control

modes with the same motor is around 200 RPM.
The current waveform reference plays an important role here. If it is closer to a sine wave, the current will follow it better and the motor will have better torque. At high speeds, it is best to use the smallest possible microsteps, in order to obtain the best motor torque. However, at high speeds, the microstep changing rate becomes faster than the output frequency of 40 kHz. The dsPIC DSC device might also run out of time to execute all of the step changes if they are very fast. For these reasons, a value of approximately 20 µs for one microstep is implemented as the lower limit for one microstep time, regardless of the microstepping resolution used. This means the top speed is higher for low resolution modes, such as full, half or quarter step, and lower for high resolution microstepping. [6]

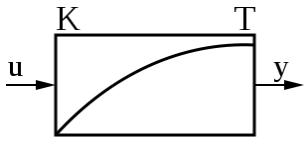


Source: [6]

Figure 45: Phase Advance

8.3 ADC Triggering

The chosen hardware setup causes a few difficulties. Caused by the full-bridge configuration and the fast-decay operation mode, low-pass filtering of the current flowing through the shunt resistor does not help to deliver correct values to the ADC module at any time. Correct values can only be sampled by setting the correct ADC triggering point for each PWM period.
To be able to adjust the triggering points, the reference values transferred via SPI have to be filtered digitally.
With this PT1-filtered reference values the triggering points can be calculated. Closer information about the triggering points can be found in the SPI1 interrupt subroutine in the appendix.



Source: http://commons.wikimedia.org/wiki/File:Pt1-controller-symbol-1.svg, 23/FEB/2014

Figure 46: PT1 Element

Calculation Example

By choosing the desired rise time of a step response, all needed parameters can be calculated.

$$T_{\text{Sample}} = 25\mu s \tag{53}$$
$$R_{\text{S}} = 1.8\Omega \tag{54}$$
$$L = 1.1mH \tag{55}$$
$$T_{\text{rise}} = 70\mu s \tag{56}$$
$$\tag{57}$$

$$K = 3 \cdot \frac{R_{\text{S}}}{T_{\text{rise}} \cdot V} = 3 \cdot \frac{1.8\Omega}{70\mu s \cdot 12V} = 6428.57\frac{\Omega}{Vs} \tag{58}$$

$$G = \frac{K}{R_{\text{S}}} = \frac{6428.57\frac{\Omega}{Vs}}{1.8\Omega} = 3571.425\frac{1}{Vs} \tag{59}$$

$$G_0 = 4 \cdot \frac{K}{R_{\text{S}}} = 4 \cdot \frac{6428.57\frac{\Omega}{Vs}}{1.8\Omega} = 14285.7\frac{1}{Vs} \tag{60}$$

$$p_1 = L + R_{\text{S}} \cdot \frac{T_{\text{Sample}}}{2} = 1.1mH + 1.8\Omega \cdot \frac{25\mu s}{2} = 0.0011225H \tag{61}$$

$$p_2 = L - R_{\text{S}} \cdot \frac{T_{\text{Sample}}}{2} = 1.1mH - 1.8\Omega \cdot \frac{25\mu s}{2} = 0.0010775H \tag{62}$$

Example for an anti-windup gain for "low speeds":

$$G_{\text{W}} = 500 \tag{63}$$

Example for an anti-windup gain for "high speeds":

$$G_{\text{W}} = 17000 \tag{64}$$

The following differences equation should be used for low speeds.

$$acc_{\text{k}} \approx 16 \cdot e_{\text{k}} - 15 \cdot e_{\text{k-1}} + acc_{\text{k-1}} - 500 \cdot (acc_{\text{k-1}} - u_{\text{k-1}}) \tag{65}$$

8.4 Stability

To check the stability of the system, the loop transfer function of the system has to be transformed into the q-area.
To approximate the q-transfer function, the following formula can be used.

$$G_{\text{Loop}}{}^{\#}(q) \approx G_{\text{Loop}}(s)\big|_{s=q} \cdot \left(1 - q \cdot \frac{T_{\text{Sample}}}{2}\right) \tag{66}$$

$$G_{\text{Loop}}{}^{\#}(q) \approx \frac{1}{\frac{R_{\text{S}}}{K \cdot V} \cdot q + 1} \cdot \left(1 - q \cdot \frac{T_{\text{Sample}}}{2}\right) \tag{67}$$

According to the expression above, the **system is stable**. This can be seen by two criterions: [14]

- The numerator's degree equals the denominator's degree.
- There are just denominators polynomials with a negative real part.

9 Evaluation of the Image Quality

The **main criterion** for the quality of a motor controller is the resulting image quality. The goal of this project is to get a motor controller which provides at least the same image quality as the console-sided controller.
There are many different modes for volume imaging. To keep the number of tests small, the most critical mode should be chosen for evaluation purposes. It is called "Doppler Color Mode".

9.1 Electromagnetic Compatibility

Most of the image distortions result from electromagnetic coupling effects. To give an overview, here is a list of the most common coupling mechanisms:

- Galvanic coupling
- Capacitive coupling
- Inductive coupling
- Waveguide coupling
- Radiation coupling

9.1.1 Possible Solutions for EMC Problems

A common method to get rid of coupling effects is to **cover a board with copper foil** and connect this foil to ground. This method can be used for the reduction of **coupling effects between different boards**. When the EMC-problem is **board-internal**, this method **will not work**.
For board-internal problems there are different design improvements.

- Shorten wire lengths
- Avoid conductor loops
- Add ground layers
- ...

9.2 Test Results

9.2.1 Test Configuration 1

Test Setup, Settings

- Probe: RAB6-D
- Console: Voluson E8 Expert
- PSU: HAMEG HM7042-5
- PWM frequency: 40 kHz
- Open loop voltage control
- No filters

Test Results

The operation of the probe is possible. The stepper motor performs the required moves without current control. **Distortions can be seen in any mode of operation**.

Possible Improvements

- Higher PWM frequency
- Higher degree of low-pass filter
- Closed loop current control

9.2.2 Test Configuration 2

Test Setup, Settings

- Probe: RAB6-D
- Console: Voluson E8 Expert
- PSU: HAMEG HM7042-5
- PWM frequency: 100 kHz
- Open loop voltage control
- No filters

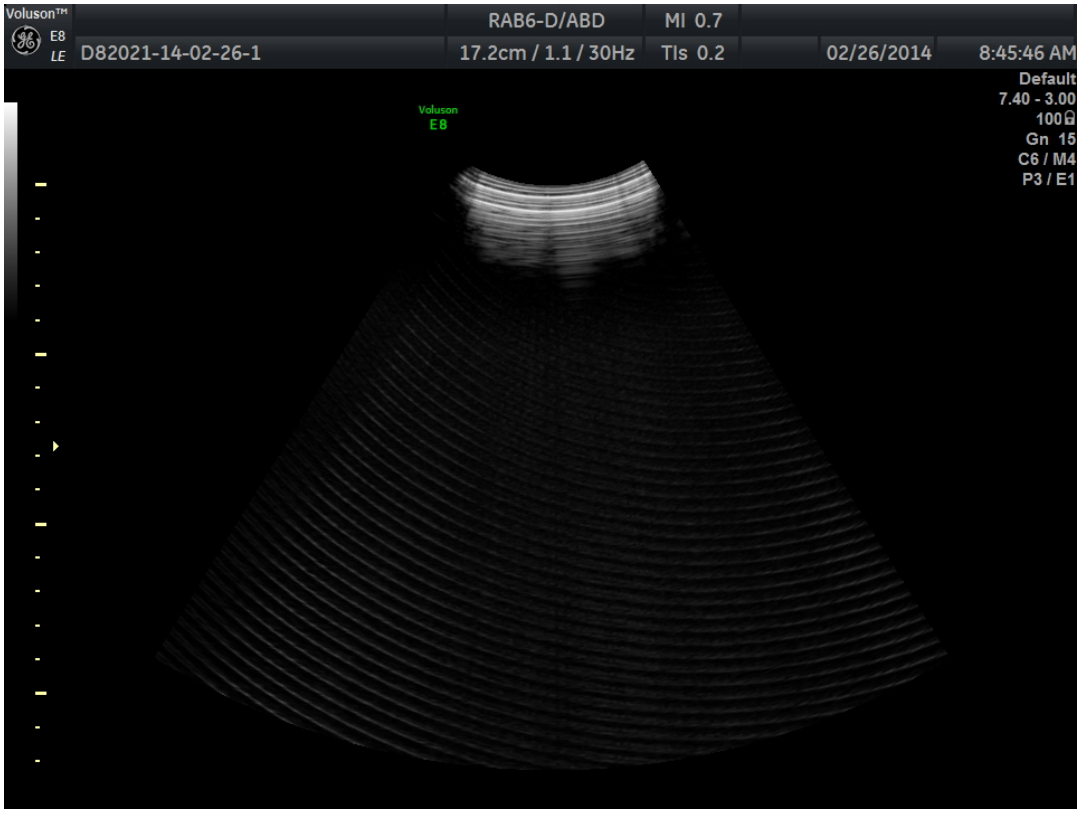


Figure 47: B-Image without additional Filter

Test Results

The image quality rises with the PWM frequency but there are still distortions in the B-image. The driver is operated at the maximal PWM frequency. This makes a higher PWM frequency than 100 kHz impossible.

Possible Improvements

- Higher degree of low-pass filter

9.2.3 Test Configuration 3

Test Setup, Settings

- Probe: RAB6-D
- Console: Voluson E8 Expert
- PSU: HAMEG HM7042-5
- PWM frequency: 100 kHz
- Open loop voltage control
- Second order filter

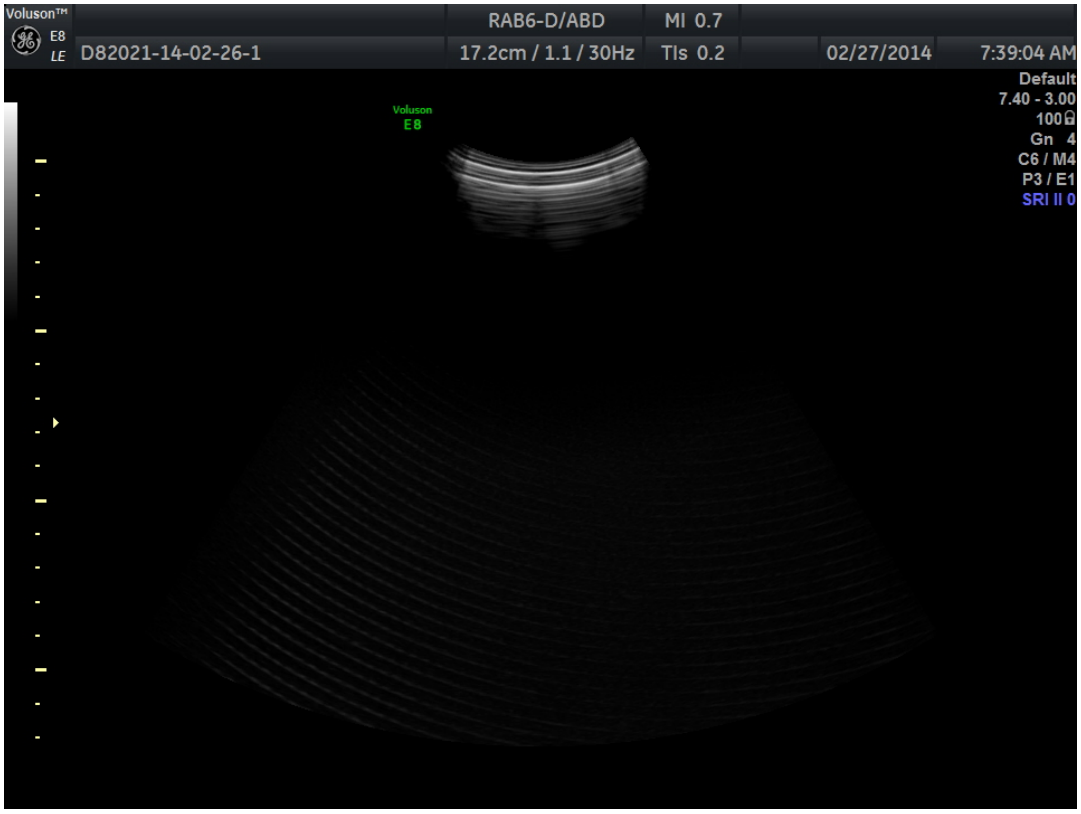


Figure 48: B-Image with Second Order Filter

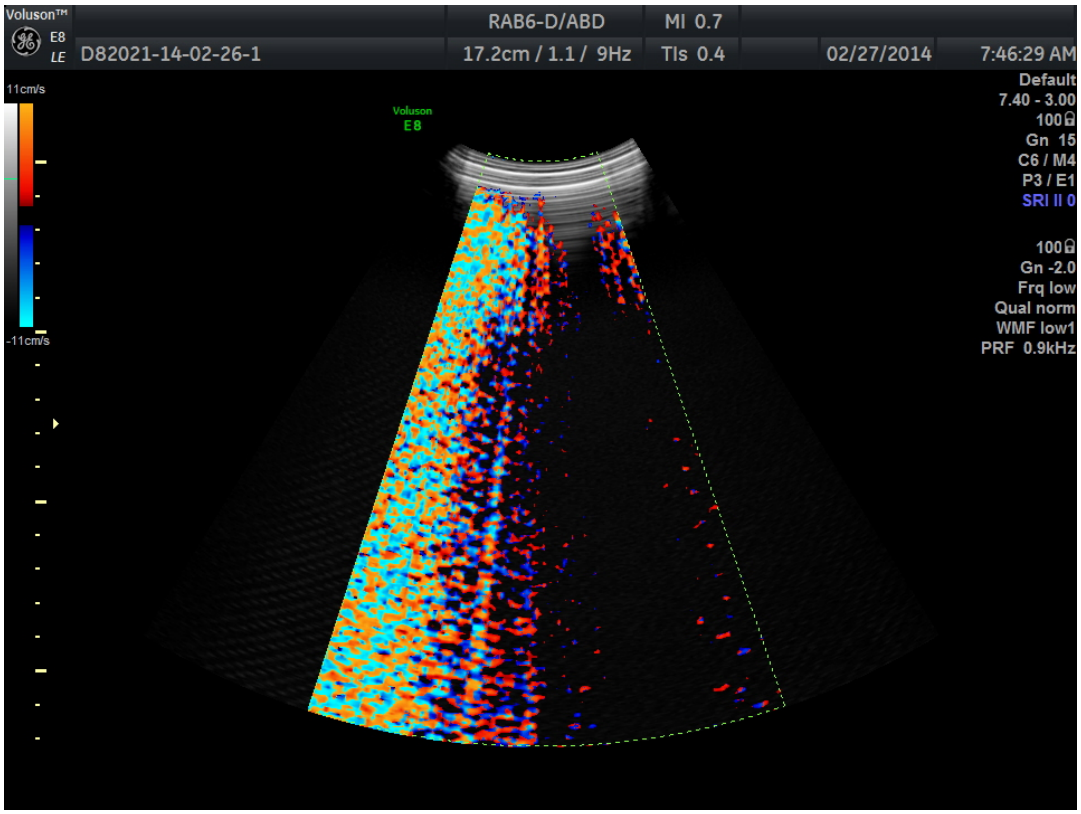


Figure 49: Doppler Color Mode, Second Order Filter

Test Results

There are less distortions in the B-image but they are not gone entirely.

Possible Improvements

- Higher degree of low-pass filter

9.2.4 Test Configuration 4

Test Setup, Settings

- Probe: RAB6-D
- Console: Voluson E8 Expert
- PSU: HAMEG HM7042-5
- PWM frequency: 100 kHz
- Open loop voltage control
- Fourth order filter

Test Results

The console-sided and probe-sided motor-drive-controllers deliver the **same B-image quality**. Also in the critical "Doppler Color Mode" the image quality is the same.

Possible Improvements

- Usage of smaller filter parts

10 Design Verification and Validation

10.1 Supply Voltage

To be able to run the motor at the required high speeds and additional mechanical load, the supply voltage has to be higher. 30 V should be an adequate voltage.

10.2 PCB

10.2.1 Motor Lines

The smallest voltage drop can be achieved when the **high-current lines** are **as wide as possible**. In the current PCB design there is not enough space available for wider motor lines. In further PCB designs this should be considered.

10.2.2 Board Thickness

For better stability and because of the jumpers and other evaluation equipment the board is thicker than necessary. For a repetition part the board can be much thinner.

10.3 Evaluation Parts

The following components are not needed for a repetition part:

- Several jumpers
- Utility Pad
- LED1_Blue, LED2_Blue, T1, T2, R3, R4
- Motor current input pads
- GND-pad next to the LVDS-pads
- R5
- R14, R15

10.4 SPI, LVDS

The currently used **quad LVDS line-receiver** offers four signal output channels. The SPI that is used needs three lines, where just the SDI- and the SCK-lines are operated at high frequencies. The frequency of the SS-line is much lower. That is why this line does not have to be transferred via LVDS. As a result, **just two LVDS-channels are needed**.

10.5 Current Regulation Feature of the Driver-IC

The used driver-IC offers a **current regulation feature**. This feature is not needed because the current regulation function is performed by the μC. The only thing, which has to be done, is to set the four driver input pins to ground. By connecting these pins directly to ground, four GPIO-pins at the μC are free. This gives the possibility to use a smaller μC with less GPIO-pins or to use these pins for other functions.

10.6 Operational Amplifiers

The used **μC offers internal operational amplifiers**. This function can also be used instead of the separate operational amplifier used in the prototype configuration.
One operational amplifier needs **three pins**. The selected μC offers **up to three operational amplifiers**.
When the internal operational amplifiers of the μC should not be used, the selected operational amplifier can be replaced by a smaller one with just two channels. A good choice would be the **MCP6022** from Microchip.

11 Conclusion and Forecast

The goal of this project is to build a board that is able to drive several motors in ultrasound probes. With the boundary conditions, which were defined at the start of the project, this would not have been possible anyway. As mentioned in the design verification, the **supply voltage has to be higher**. As a result, a probe-sided motor-drive-controller does not work for any probe without the adaption of the PSU on the console-side.
When the PSU has to be adapted anyway, one of those which provide a **positive and a negative voltage** should be used. This makes a half-bridge configuration possible and eases the needed current control significantly.
According to the space problem in ultrasound probes, the actual prototype has to be adapted. In the actual design there are jumpers and other parts, which are not necessary for a repetition part. By **reducing the number of components**, a board can be built, which fits into all selected probes.

Formula Symbol List

Symbol	Description	Unit
$\Delta\varphi$	Angle between two current Pointers	rad
τ	Time	s
acc	Accumulator Value	1
e	Current Error	A
f	Frequency	Hz
i	Time dependent Current	A
n	Counting Number	1
p	Pole Constant	H
q	Complex Variable	1/s
s	Laplace Variable	1/s
t	Time	s
u	Duty Cycle Value	1
z	Discrete Variable	1
COS	Cosine Value	1
G	Transfer Function / Constant	1 / More possible Values
I	Current	A
K	Constant	More possible Values
L	Inductance	H
N	Number of Sinus Cycles for one Revolution	1
$N_{\mu Steps}$	Number of μSteps for one Sinus Cycle	1
R	Resistance	Ω
RES	Resolution of the Reference Values	1
SIN	Sinus Value	1
SPEED	32 Bit Driver-IC Register Value	1
T	Period Duration	s
U	Voltage	V
V	DC Voltage	V
VM	Motor Voltage	V
Z	Impedance	Ω

Formula Symbol List Subindizes

μSteps	μSteps for one Sinus Cycle
disCS	Disable Cable Select
f	Follower
k	Control Variable
max	Maximal Value
min	Minimal Value
new	New Value
old	Old Value
peak	Peak Value
L	Inductance
M	Motor
N	Connected to negative Voltage / Reset Time
P	Proportional
PhA, PhB	Phase A, B
PI	Proportional Integral
Ref	Reference Value
S	Serial
SCK	Serial Clock
Shunt	For Measurement
U	Voltage Amplification
W	Windup

List of Acronyms

Acronym	Description
μC	Microcontroller
ADC	Analog-to-digital Converter
DAC	Digital-to-analog Converter
DIR	Direction
ESR	Equivalent serial Resistance
EMC	Electromagnetic Compatibility
FET	Field Effect Transistor
FFT	Fast Fourier Transformation
GE	General Electric
GND	Ground
GPIO	General Purpose Input/Output
HiZ	High Impedance
IC	Integrated Circuit
IDE	Integrated Development Environment
LSB	Least significant Bit
LVDS	Low Voltage Differential Signaling
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
MSB	Most significant Bit
MUX	Multiplexer
N_STEP	Number of Steps
NPI	New Product Introduction
PARAM	Parameter
PCB	Printed Circuit Board
PLL	Phase Locked Loop
PSU	Power Supply Unit
PWM	Pulse Width Modulation
RAB	Real-time, abdominal
RIC	Real-time, intracavitary
RM6C	Real-time, 6 MHz, convex
RNA	Real-time, neonatal
RSM	Real-time, small Parts, musculoskeletal
RSP	Real-time, small Parts
SCK	Serial Clock
SDI	Serial Data Input
SDO	Serial Data Output
SOIC	Solid Oxide Integrated Circuit
SPD	Speed
SPI	Serial Peripheral Interface
SS	Slave Select/Chip Select
SSOP	Shrink Small-Outline Package

List of Figures

List of Tables

Code-Listings

Evaluation Firmware for ST L6470 + Microchip dsPIC33FJ128MC802

General Initialization

```
// Settings
#define VERSION_NUMBER 0

// Current Value + Offset
#define SPI1_data_mask 0x7FF8
#define ContPatternMask (0x7FFF - SPI1_data_mask) // for 3Bit Check Mask 0x0007
// SPI Transmission Parity Check
#define SinContPattern 0x0005   // Reference Value SIN: 101
#define CosContPattern 0x0002 // Reference Value COS: 010
#define SPI_Reset_Time 100      // Reset Recovery Time
#define RESOLUTION 4096

// Function Prototypes
void Init_SPI1(void);
void Init_SPI2(void);
void Write_SPI2(short command);

void Delay_ms (unsigned int delay);
void Delay_us(unsigned int usec);
void Delay(void);
int Sign(int x);
signed int GetDirection(int sin_or_cos_value, signed int oldSIN, signed int oldCOS, signed int newVALUE);
int GetQuadrant(signed int SIN, signed int COS);

void SetParam(unsigned int param, unsigned int value_byte_2, unsigned int value_byte_1_and_0);
void Run(unsigned char direction, unsigned int speed_byte_2, unsigned int speed_byte_1_and_0);
void ResetDevice(void);
void SoftStop(void);
void HardStop(void);
void SoftHiZ(void);
void HardHiZ(void);

// Interrupt Subroutines
void __attribute__((__interrupt__, no_auto_psv)) _SPI1Interrupt(void);
void __attribute__((__interrupt__, no_auto_psv)) _SPI2Interrupt(void);
```

Port Initialization

```
#include <p33FJ128MC802.h>
```

```
void Init_Ports(void)
{
    // Port A
    PORTA = 0;
    LATA = 0;
    // Data Direction Registers Configuration
    // TRISBit = 1 --> Input, TRISBit = 0 --> Output
    TRISAbits.TRISA0 = 0; // RED LED output

    // Port B
    PORTB = 0;
    LATB = 0;
    // Data Direction Registers Configuration
    TRISBbits.TRISB12 = 1; // SDI1
    TRISBbits.TRISB13 = 1; // SCK1
    TRISBbits.TRISB14 = 1; // SS1
    TRISBbits.TRISB15 = 0; // SDO1

    // Data Direction Registers Configuration
    TRISBbits.TRISB2 = 1; // SDI2
    TRISBbits.TRISB3 = 0; // SDO2
    TRISBbits.TRISB4 = 0; // SCK2
    TRISBbits.TRISB7 = 0; // SS2
}
```

Initialization of the SPI1 Module

```
#include <p33FJ128MC802.h>

void Init_SPI1(void)
{
    // Assign SPI1-Registers
    RPINR20bits.SDI1R = 12;
    RPINR20bits.SCK1R = 13;
    RPINR21bits.SS1R = 14;
    RPOR7bits.RP15R = 7;

    // Clear Buffer Register
    SPI1BUF = 0x0000;

    IFS0bits.SPI1IF = 0;
    IEC0bits.SPI1IE = 1; //SPI1 Event Interrupt
    IPC2bits.SPI1IP = 4; //SPI1 Datentransfer

    // Configuration of SPI1CON1
    SPI1CON1bits.MSTEN = 0; // Slave Mode selected
    SPI1CON1bits.DISSCK = 0; // just for Master Mode
    SPI1CON1bits.DISSDO = 1; // no SDO in Slave Mode
    SPI1CON1bits.MODE16 = 1; // 16 Bit Bandwidth
    SPI1CON1bits.SMP = 0; // in Slave Mode always 0
```

```
    SPI1CON1bits.SSEN = 1; // SS1 active for Slave Mode

    // Definition of Clock Priority and Data Exchange
    SPI1CON1bits.CKE = 0; // Data Exchange on CLK idle to CLK active
    SPI1CON1bits.CKP = 0; // CLK idle=0, CLK active=1

    // Configuration of SPI1STAT
    SPI1STATbits.SPISIDL = 0; // SPI continues working in Idle Mode
    IPC2bits.SPI1IP = 4; // SPI1 Data Transfer
    // IPC2bits.SPI1EIP = 4; // SPI1 Error
    IFS0bits.SPI1IF = 0;
    // IFS0bits.SPI1EIF = 0;
    SPI1STATbits.SPIROV = 0; // Clear Error Flag
    IEC0bits.SPI1IE = 1; // SPI1 Event Interrupt
    // IEC0bits.SPI1EIE = 1; // SPI1 Error Interrupt
    // while (PORTBbits.RB14 == 0); // activate, when SS is high
    SPI1STATbits.SPIEN = 1; // Switch SPI1 Module on
}
```

Initialization of the SPI2 Module

```
#include <p33FJ128MC802.h>

void Init_SPI2(void)
{
    // Assign SPI2-Registers
    RPINR22bits.SDI2R = 2; // Serial data input to RP2 = Pin 6
    RPOR1bits.RP3R = 0b01010; // SDO2 mapped to RP3
    RPINR22bits.SCK2R = 4; // Serial clock to  RP4 = Pin 11
    RPOR2bits.RP4R = 0b01011; // SCK2 mapped to RP4
    RPINR23bits.SS2R = 7; // Slave selecto to RP7 = Pin 16
    RPOR3bits.RP7R = 0b01100; // SS2 mapped to RP7

    /* The following code shows the SPI register configuration for
   Master mode */
    IFS2bits.SPI2IF = 0; // Clear the Interrupt Flag
    IFS2bits.SPI2EIF = 0; // Clear the error interrupt flag
    IEC2bits.SPI2IE = 0; // Disable the Interrupt
    IPC8bits.SPI2IP = 4; // SPI2 Priority

    // SPI1CON1 Register Settings
    SPI2CON1bits.DISSCK = 0; // Internal Serial Clock is Enabled
    SPI2CON1bits.DISSDO = 0; // Data output is enabled

    // SDO2 pin is controlled by the module
    SPI2CON1bits.MODE16 = 0; // Communication is byte-wide (8 bits)
    SPI2CON1bits.CKE = 0; // Serial output data changes on transition
   from Idle clock state to active clock state
    SPI2CON1bits.CKP = 0; // Idle state for clock is a low level; //
   active state is a high level
    SPI2CON1bits.MSTEN = 1; // Master mode enabled
```

```
    SPI2CON1bits.SMP = 0; // Input data sampled at middle of data
    output time
    SPI2CON1bits.PPRE = 0b010; // Primary prescale 4:1
    SPI2CON1bits.SPRE = 0b110; // Secondary prescale 2:1
    SPI2CON2 = 0x0000;
    SPI2STATbits.SPIEN = 1; // Enable SPI module

    // Interrupt Controller Settings
    IFS2bits.SPI2IF = 0; // Clear the Interrupt Flag
    IFS2bits.SPI2EIF = 0; /// Clear the error interrupt flag
    IEC2bits.SPI2IE = 1; // Enable the Interrupt

    Write_SPI2(0x0000); // Write data to be transmitted
}
```

SPI1 Interrupt Service Routine

```
#include "p33FJ128MC802.h"
#include "init.h"
#include <dsp.h>

void __attribute__((__interrupt__, no_auto_psv)) _SPI1Interrupt(void)
    // Duration: approx. 620 ns
{
    // LATAbits.LATA0 = 1; // Switch RED LED on

    extern volatile signed int refSIN, refCOS;
    extern volatile int oldDIR;
    extern volatile char MotorDriver_STOP;
    signed int buffered_Data;
    extern volatile char SPI_ERROR;
    signed int oldSIN = 0, oldCOS = 0;
    signed int deltaSIN = 0, deltaCOS = 0;
    int newDIR;

    buffered_Data = SPI1BUF;
    // Compute received Data
    if (SPI_ERROR == 0)
    {
        MotorDriver_STOP = 0;
        oldSIN = refSIN;
        oldCOS = refCOS;

        // Parity Check
        if (buffered_Data > 0) // SIN
        {
            if ((buffered_Data & ContPatternMask)== SinContPattern) //
    Parity check
            {
                buffered_Data = buffered_Data & SPI1_data_mask; //
    Separation of data and check bits
```

```
                buffered_Data = buffered_Data >> 3;
                refSIN = buffered_Data - (RESOLUTION / 2);
                deltaSIN = refSIN - oldSIN;
                newDIR = GetDirection(0, oldSIN, oldCOS, refSIN);
                if(oldDIR != newDIR) HardStop();
                if(newDIR != -1) Run(newDIR, deltaSIN, 0x0000);
                oldDIR = newDIR;
            }
            else
            {
                SPI_ERROR = 1;
                refSIN = 0;
                refCOS = 0;
                MotorDriver_STOP = 1;
            }
        }
        else // COS
        {
            if ((buffered_Data & ContPatternMask)== CosContPattern) //
    Parity check
            {
                buffered_Data = buffered_Data & SPI1_data_mask; //
    Separation of data and check bits
                buffered_Data = buffered_Data >> 3;
                refCOS = buffered_Data - (RESOLUTION / 2);
                deltaCOS = refCOS - oldCOS;
                newDIR = GetDirection(1, oldSIN, oldCOS, refCOS);
                if(oldDIR != newDIR) HardStop();
                if(newDIR != -1) Run(newDIR, deltaCOS, 0x0000);
                oldDIR = newDIR;
            }
            else
            {
                SPI_ERROR = 1;
                refSIN = 0;
                refCOS = 0;
                MotorDriver_STOP = 1;
            }
        }
    }
    SPI1STATbits.SPIROV = 0; // Received overflow flag: no overflow has
    occured
    IFS0bits.SPI1IF = 0; // Clear SPI1 interrupt flag
    IFS0bits.SPI1EIF = 0; // Clear SPI1 error interrupt flag


    // LATAbits.LATA0 = 0; // Switch RED LED off
}

// Returns the new direction calculated by the transmitted value
signed int GetDirection(int sin_or_cos_value, signed int oldSIN, signed
    int oldCOS, signed int newVALUE)
```

```
{
    int oldQuadrant = GetQuadrant(oldSIN, oldCOS);

    if(sin_or_cos_value == 0) // SIN
    {
        if((oldQuadrant == 1) || (oldQuadrant == 4))
        {
            if((newVALUE - oldSIN) > 0) return 1;
            else return 0;
        }
        if((oldQuadrant == 2) || (oldQuadrant == 3))
        {
            if((newVALUE - oldSIN) < 0) return 1;
            else return 0;
        }
    }
    else // COS
    {
        if((oldQuadrant == 3) || (oldQuadrant == 4))
        {
            if((newVALUE - oldCOS) > 0) return 1;
            else return 0;
        }
        if((oldQuadrant == 1) || (oldQuadrant == 2))
        {
            if((newVALUE - oldCOS) < 0) return 1;
            else return 0;
        }
    }
    return -1;
}

// Returns the quadrant (1, 2, 3, 4) calculated by SIN, COS
int GetQuadrant(signed int SIN, signed int COS)
{
    if((SIN >= 0) & (COS > 0)) return 1; // Q1
    if((SIN > 0) & (COS <= 0)) return 2; // Q2
    if((SIN <= 0) & (COS < 0)) return 3; // Q3
    if((SIN < 0) & (COS >= 0)) return 4; // Q4
    else return 0; // SIN = 0, COS = 0
}
```

Evaluation Firmware for TI DRV8841 + Microchip dsPIC33FJ128MC802

General Initialization

```
// Settings
#define VERSION_NUMBER 0
```

```
// Current Value + Offset
#define SPI1_data_mask 0x7FF8
#define ContPatternMask (0x7FFF - SPI1_data_mask) // for 3 bit parity check 0x0007
// SPI Transmission Check Pattern
#define SinContPattern 0x0005   // for reference value SIN 101
#define CosContPattern 0x0002 // for reference value COS 010
#define SPI_Reset_Time 100

#define AI0 LATBbits.LATB7
#define AI1 LATBbits.LATB4
#define BI0 LATBbits.LATB3
#define BI1 LATBbits.LATB2

#define TABLE_SIZE 256 //sinewave look-up table size

// Function Prototypes
void Init_SPI1(void);
void Init_PWM(void);
void Init_Timer1(void);
void Init_ADC(void);
void SetPWM(void);
void UpdateTimer1(void);
void Measure_Current(void);

void Delay_ms (unsigned int delay);
void Delay_us(unsigned int usec);
void Delay(void);

// Interrupt Subroutines
void __attribute__((__interrupt__, no_auto_psv)) _SPI1Interrupt(void);
void __attribute__((__interrupt__, no_auto_psv)) _T1Interrupt(void);
void __attribute__((__interrupt__,auto_psv)) _ADC1Interrupt(void);
void __attribute__((__interrupt__,no_auto_psv)) _MPWM1Interrupt(void);
void __attribute__((__interrupt__,no_auto_psv)) _MPWM2Interrupt(void);
```

Port Initialization

```
#include <p33FJ128MC802.h>
#include "init.h"

void Init_Ports(void)
{
    // Port A
    PORTA = 0;
    LATA = 0;
    // Data Direction Registers Configuration
    // TRISBit = 1 --> Input, TRISBit = 0 --> Output
    TRISAbits.TRISA0 = 0; // RED LED output
    TRISAbits.TRISA1 = 1; // ADC Input

```

```
    // Port B
    PORTB = 0;
    LATB = 0;
    // System Side
    // Data Direction Registers Configuration
    TRISBbits.TRISB12 = 1; // SDI1
    TRISBbits.TRISB13 = 1; // SCK1
    TRISBbits.TRISB14 = 1; // SS1

    // Driver Side
    TRISBbits.TRISB15 = 1; // nFAULT
    TRISBbits.TRISB2 = 0; // BI1
    TRISBbits.TRISB3 = 0; // BI0
    TRISBbits.TRISB4 = 0; // AI1
    TRISBbits.TRISB7 = 0; // AI0
    TRISBbits.TRISB8 = 0; // BIN1
    TRISBbits.TRISB9 = 0; // BIN2
    TRISBbits.TRISB10 = 0; // AIN1
    TRISBbits.TRISB11 = 0; // AIN2
}
```

Initialization of the SPI1 Module

```
#include <p33FJ128MC802.h>

void Init_SPI1(void)
{
    // Assign SPI1-Registers
    RPINR20bits.SDI1R = 12;
    RPINR20bits.SCK1R = 13;
    RPINR21bits.SS1R = 14;

    // Clear Buffer Register
    SPI1BUF = 0x0000;

    IFS0bits.SPI1IF = 0;
    IEC0bits.SPI1IE = 1; // SPI1 event interrupt
    IPC2bits.SPI1IP = 4; // SPI1 data transfer

    SPI1CON1bits.MSTEN = 0; // Slave Mode selected
    SPI1CON1bits.DISSCK = 0; // just for master mode
    SPI1CON1bits.DISSDO = 1; // no SDO in slave mode
    SPI1CON1bits.MODE16 = 1; // 16 bit bandwidth
    SPI1CON1bits.SMP = 0; // always 0 in slave mode
    SPI1CON1bits.SSEN = 1; // SS is active

    SPI1CON1bits.CKE = 0; // Data change CLK idle to CLK active
    SPI1CON1bits.CKP = 0; // CLK idle=0, CLK active=1

    SPI1STATbits.SPISIDL = 0; // SPI continues working in idle mode
    IPC2bits.SPI1IP = 4; // SPI1 data transfer
```

```
29      IFS0bits.SPI1IF = 0;
        SPI1STATbits.SPIROV = 0; // Clear error flag
31      IEC0bits.SPI1IE = 1; // SPI1 Event Interrupt
        SPI1STATbits.SPIEN = 1; // Switch on SPI module
33  }
```

Initialization of the PWM Modules

```
1   #include <p33FJ128MC802.h>
    #include "userparameters.h"
3   #include "init.h"

5   void Init_PWM(void)
    {
7       // Clear Duty Cycle Registers
        P1DC3 = 0;
9       P2DC1 = 0;

11      P1TPER = PWM_FCY;          // Setup PWM period
        P2TPER = PWM_FCY;          // Setup PWM period
13
        PWM1CON1bits.PMOD3 = 0; // PWM3-pair is in complementary output
    mode
15      PWM1CON1bits.PEN3L = 1; // PWM3L is enabled
        PWM1CON1bits.PEN3H = 1; // PWM3H is enabled
17      PWM2CON1bits.PMOD1 = 0; // PWM1-pair is in complementary output
    mode
        PWM2CON1bits.PEN1L = 1; // PWM1L is enabled
19      PWM2CON1bits.PEN1H = 1; // PWM1H is enabled

21      PWM1CON2 = 0x0002;         // special event postcale 1:1
                                   // updates to the PDCx registers are sync
    to the PWM time base
23                                 // Output overrides via the PxOVDCON
    register are sync to the PWM time base

25      PWM2CON2 = 0x0002;         // special event postcale 1:1
                                   // updates to the PDCx registers are sync
    to the PWM time base
27                                 // Output overrides via the PxOVDCON
    register are sync to the PWM time base

29      // Set Dead Time
        P1DTCON1bits.DTAPS = 0;
31      P1DTCON1bits.DTBPS = 0;
        P1DTCON1bits.DTA = 10;
33      P1DTCON1bits.DTB = 20;

35      P1DTCON2bits.DTS3A = 0;
        P1DTCON2bits.DTS3I = 1;
37
```

```
    P2DTCON1bits.DTAPS = 0;
    P2DTCON1bits.DTBPS = 0;
    P2DTCON1bits.DTA = 10;
    P2DTCON1bits.DTB = 20;
    P2DTCON2bits.DTS1A = 0;
    P2DTCON2bits.DTS1I = 1;

    // P1SECMP: Special Event Compare Count Register
    // Phase of ADC capture set relative to PWM cycle: 0 offset and counting up
    P1SECMP = 1;

    // Pins are controlled by the PWM module
    P1OVDCON = 0b0011000000000000;
    P2OVDCON = 0b0000001100000000;

    // PWM Prescale 1:1, PWM Postscale 1:1
    P1TCON = 0x0000;
    P2TCON = 0x0000;

    //Continuous Up/Down Count Mode with Interrupts for Double Update of Duty Cycle
    P1TCONbits.PTMOD = 0b11;
    P2TCONbits.PTMOD = 0b11;

    //// Special Event Trigger for ADC
    // Select Special Event time base direction such that trigger will occur
    // when PWM time base is counting upwards
    P1SECMPbits.SEVTDIR = 1;
    P2SECMPbits.SEVTDIR = 1;

    // Select PWM Special Event Trigger Output Postscale value to 1:1
    PWM1CON2bits.SEVOPS = 0b0000;
    PWM2CON2bits.SEVOPS = 0b0000;

    /* Assign special event compare value */
    P1SECMPbits.SEVTCMP = 0;
    P2SECMPbits.SEVTCMP = 0;

    IFS3bits.FLTA1IF = 0; // Clear FLTA1 interrupt flag
    IFS4bits.FLTA2IF = 0; // Clear FLTA2 interrupt flag

    // Disable immidiate Updates
    PWM1CON2bits.IUE = 0;
    PWM2CON2bits.IUE = 0;

    //Interrupt Configuration
    IFS3bits.PWM1IF = 0; // Clear flag
    IEC3bits.PWM1IE = 1; // Enable interrupt
    IFS4bits.PWM2IF = 0; // Clear flag
    IEC4bits.PWM2IE = 0; // Disable interrupt
```

```
87
    // Set Interrupt Priorities
89  IPC14bits.PWM1IP = 3;

91  // Enable PWM
    P1TCONbits.PTEN = 1;
93  P2TCONbits.PTEN = 1;

95  // Set Current Levels to 38%
    AI0 = 0;
97  AI1 = 1;
    BI0 = 0;
99  BI1 = 1;
}
```

ADC Initialization

```
#include <p33FJ128MC802.h>
2 #include "userparameters.h"
#include "init.h"
4
void Init_ADC(void)
6 {
    AD1CON1bits.AD12B = 1; // 12-bit ADC operation
8   AD1CON2bits.VCFG = 0b000; // Select AVDD, AVSS as reference supply
    AD1CON3bits.ADRC = 0; // ADC Clock is derived from Systems Clock
10  AD1CON3bits.SAMC = 0; // Auto Sample Time = 0 * TAD
    AD1CON3bits.ADCS = 2; // ADC Conversion Clock TAD = TCY * (ADCS +
    1) = (1/40M) * 3 =
12  // 75 ns (13.33 MHz)
    // ADC Conversion Time for 10-bit Tconv = 12 * TAD = 900 ns (1.1
    MHz)
14
    //AD1CHS0/AD1CHS123: Analog-to-Digital Input Select Register
16  AD1CHS0bits.CH0SA = 1; // MUXA +ve input selection (AIN1) for CH0
    AD1CHS0bits.CH0NA = 0; // MUXA -ve input selection (VREF-) for CH0
18  AD1CHS123bits.CH123SA = 0; // MUXA +ve input selection (AIN0) for
    CH1
    AD1CHS123bits.CH123NA = 0; // MUXA -ve input selection (VREF-) for
    CH1
20
    AD1CON2bits.CHPS = 0; // Converts channels CH0
22
    //AD1PCFGH/AD1PCFGL: Port Configuration Register
24  AD1PCFGL = 0xFFFF;
    // AD1PCFGH = 0xFFFF;
26  AD1PCFGLbits.PCFG1 = 0; // AN1 as Analog Input

28  AD1CON1bits.SIMSAM = 1; // Samples CH0 and CH1 simultaneously

30  AD1CSSLbits.CSS1 = 1; // Select AN1 for input scan
```

```
    AD1CON1bits.FORM = 0b00; // Data Output Format: Unsigned Integer

    /* Choose ADC1 trigger source such that MCPWM1 module stops sampling and
     starts conversion */
    AD1CON1bits.SSRC = 0b011;
    AD1CON1bits.ASAM = 1; // ADC Sample Control: Sampling begins immediately after conversion

    AD1CON1bits.ADDMABM = 1; // DMA buffers are built in conversion order mode
    AD1CON2bits.SMPI = 0; // SMPI must be 0

    IPC3bits.AD1IP = 1; // Set Interrupt Priority

    IFS0bits.AD1IF = 0; // Clear the Analog-to-Digital interrupt flag bit
    IEC0bits.AD1IE = 1; // Enable Analog-to-Digital interrupt

    AD1CON1bits.ADON = 1; // Turn on the ADC
}
```

SPI1 Interrupt Service Routine

```
#include "p33FJ128MC802.h"
#include "init.h"
#include "userparameters.h"
#include <dsp.h>

void __attribute__((__interrupt__, no_auto_psv)) _SPI1Interrupt(void)
    // Duration: approx.
{

    extern volatile signed int refSIN, refCOS;
    extern volatile char MotorDriver_STOP;
    signed int buffered_Data;
    extern volatile char SPI_ERROR;

    extern volatile int pwmOutSIN, pwmOutCOS; // Temporary variable to hold PWM duty cycle values during calculations

    buffered_Data = SPI1BUF;
    // Compute received Data
    if (SPI_ERROR == 0)
    {
        MotorDriver_STOP = 0;

        // Parity Check
        if (buffered_Data > 0) // SIN
        {
```

```
            if ((buffered_Data & ContPatternMask)== SinContPattern) // Parity check
            {
                buffered_Data = buffered_Data & SPI1_data_mask; // Separation of data and check bits
                buffered_Data = buffered_Data >> 3;
                refSIN = buffered_Data / 4;
                if(refSIN < PWM_MAX) pwmOutSIN = refSIN;
                else pwmOutSIN = PWM_MAX;
            }
            else
            {
                SPI_ERROR = 1;
                refSIN = 0;
                refCOS = 0;
                MotorDriver_STOP = 1;
            }
        }
        else // COS
        {
            if ((buffered_Data & ContPatternMask)== CosContPattern) // Parity check
            {
                buffered_Data = buffered_Data & SPI1_data_mask; // Separation of data and check bits
                buffered_Data = buffered_Data >> 3;
                refCOS = buffered_Data / 4;
                if(refCOS < PWM_MAX) pwmOutCOS = refCOS;
                else pwmOutCOS = PWM_MAX;
            }
            else
            {
                SPI_ERROR = 1;
                refSIN = 0;
                refCOS = 0;
                MotorDriver_STOP = 1;
            }
        }
        SetPWM();
    }

    SPI1STATbits.SPIROV = 0; // Received overflow flag: no overflow has occured
    IFS0bits.SPI1IF = 0; // Clear SPI1 interrupt flag
    IFS0bits.SPI1EIF = 0; // Clear SPI1 error interrupt flag
}
```

Prototype Firmware for TI DRV8841 + Microchip dsPIC33EP256MC202

General Initialization

```
// General Settings
#define VERSION_NUMBER 0

// Oscillator Settings
#define F_OSC 130 // Varies between 120 and 140 MHz
#define F_CY F_OSC / 2 // Measured at room temperature without tuning

// Current Value + Offset
#define SPI1_DATA_MASK 0x7FF8
#define CONT_PATTERN_MASK (0x7FFF - SPI1_DATA_MASK) // at 3Bit Check
    Pattern 0x0007
// SPI Transmission Check Pattern
#define SIN_CONT_PATTERN 0x0005   // for Reference SIN: 101
#define COS_CONT_PATTERN 0x0002 // for Reference COS: 010
#define SPI_Reset_Time 100     // Time for refreshment at SPI-Reset in
    ms
#define SPI_RESOLUTION 4096
#define SPI_HALF_RESOLUTION (SPI_RESOLUTION / 2)
#define SPI_NEGATIVE_HALF_RESOLUTION (SPI_HALF_RESOLUTION * -1)
#define T_SPI 0.00004 //s, SPI sample time

// ADC Settings
#define ADC_MAX_RESOLUTION 4096 // Maximal Resolution: 12 Bit
#define ADC_RESOLUTION 1024 // Resolution: 10 Bit
#define ADC_HALF_RESOLUTION (ADC_RESOLUTION / 2) // 512
#define ADC_NEGATIVE_HALF_RESOLUTION (ADC_HALF_RESOLUTION * -1) // -512
#define ADC_TOLERANCE ADC_HALF_RESOLUTION / 100 // 1% Tolerance
#define ADC_LOWEST_ACCEPTABLE_VALUE (ADC_HALF_RESOLUTION -
    ADC_TOLERANCE)
#define T_SAMPLE 0.00001 // Value in s, T_SAMPLE ~ 1 / PWM_FCY = 1 /
    ~100 kHz

// Scaling Factors
#define SPI_DIV_ADC_RESOLUTION_FACTOR (SPI_RESOLUTION / ADC_RESOLUTION)
#define SPI_DIV_ADC_SHIFT 2

// PWM Settings
#define PWM_FCY 95.21484 // Value in kHz, PWM_FCY = FOSC * 1000000 /
    ADC_RESOLUTION
#define PWM_TPER 1365 // PWM_TPER = FOSC / PWM_FCY
#define PWM_FACTOR (int)(SPI_RESOLUTION / PWM_TPER) // PWM_FACTOR =
    SPI_RESOLUTION / PWM_TPER
#define PWM_MAX PWM_TPER // Maximal Duty Cycle Value
#define PWM_ZERO PWM_MAX / 2
#define PWM_MIN 0 // Minimal Duty Cycle Value
#define PWM_DC_DIFFERENCE_THRESHOLD (PWM_TPER / 200) // Needed for
    correct triggering
```

```
41 #define PWM_PHASE2 310 // Needed for synchronization of PWM1 and PWM2

43 // 100% Current (Driver Settings)
   #define AI0 0
45 #define AI1 0
   #define BI0 0
47 #define BI1 0

49 // C Function Prototypes
   void Init_Ports(void);
51 void Init_SPI1(void);
   void Init_PWM(void);
53 void Init_ADCs(void);
   void Init_Driver(void);
55 void Init_Controller(void);
   void Init_Controller_History(void);
57 void Set_ADC_Trigger(void);
   int GetQuadrant(signed int SIN, signed int COS);
59 void Init_Timer1(void);
   void SecondTickEvent(void);
61
   // C Interrupt Subroutines
63 void __attribute__((__interrupt__, no_auto_psv)) _SPI1Interrupt(void);
   void __attribute__((interrupt, no_auto_psv)) _AD1Interrupt(void);
65
   // Assembler Function Prototypes
67 extern void CALC_PT1_FILTERED_SIN(void);
   extern void CALC_PT1_FILTERED_COS(void);
69 extern void CALC_PT1_FILTERED_SIN_PWM(void);
   extern void CALC_PT1_FILTERED_COS_PWM(void);
71 extern void CORRECT_SIN(void);
   extern void CORRECT_COS(void);
73 extern void CALC_SIN_DC(void);
   extern void CALC_COS_DC(void);
75 extern void TEST(void);
```

Controller Parameters

```
1 // Control Method Selection
  // 0 ... Open Loop Voltage Control
3 // 1 ... Closed Loop Current Control, I-Controller
  // 2 ... Closed Loop Current Control, PI-Controller
5 // 3 ... Closed Loop Current Control, I-, Magnitude-Controller
  // 4 ... Test
7 #define CONTROL_METHOD 0

9 // Filter Selection
  // 0 ... All filters off
11 // 2 ... 2. order filter
  // 4 ... 4. order filter
13 #define FILTER_METHOD 4
```

```
// Voltage Supply Parameters
#define V 12 // Value in V
#define I_MAX 2 // Value in A, Maximal current

// Motor Parameters FL20STH42-1004A
#define R_WINDING 3.5 // Ohm
#define R_LINES 0.0 // Ohm
#define L_WINDING 0.001 // H
#define R (R_WINDING + R_LINES)

// Settings for PT1 Filter Reference Values
#define K_PT1 2

// I-Controller Settings
#define I_ERROR_GAIN

// PI-Controller Settings
#define T_RISE 0.0001 // Value in s, Step function response time
#define K_FACTOR 3 // Gain factor for error amplification
#define G_FACTOR 1.0 // Gain factor needed to avoid underflows
```

Port Initialization

```
#include <p33Exxxx.h>
#include "init.h"

void Init_Ports(void)
{
    // TRISBit = 1 --> Input, TRISBit = 0 --> Output

    // Port A
    PORTA = 0;
    LATA = 0;
    // Data Direction Registers Configuration
    TRISAbits.TRISA0 = 1; // ADC SIN_RTN Input
    TRISAbits.TRISA1 = 1; // ADC COS_RTN Input
    TRISAbits.TRISA2 = 1; // nFAULT
    TRISAbits.TRISA3 = 0; // nSLEEP
    TRISAbits.TRISA4 = 1; // LVDS Line Receiver OUT 4

    // Port B
    PORTB = 0;
    LATB = 0;
    // Data Direction Registers Configuration
    TRISBbits.TRISB0 = 1; // nSS1
    TRISBbits.TRISB1 = 0; // DECAY
    TRISBbits.TRISB4 = 0; // nRESET
    TRISBbits.TRISB5 = 0; // Utility Pad
    TRISBbits.TRISB6 = 0; // BI1
    TRISBbits.TRISB7 = 1; // SCK1
```

```
28      TRISBbits.TRISB8 = 0; // BI0
        TRISBbits.TRISB9 = 1; // SDI1
30      TRISBbits.TRISB10 = 0; // AI1
        TRISBbits.TRISB11 = 0; // AI0
32      TRISBbits.TRISB12 = 0; // PWM2H
        TRISBbits.TRISB13 = 0; // PWM2L
34      TRISBbits.TRISB14 = 0; // PWM1H
        TRISBbits.TRISB15 = 0; // PWM1L
36  }
```

Initialization of the SPI1 Module

```
#include <p33Exxxx.h>
2
void Init_SPI1(void)
4  {
       PMD1bits.SPI1MD = 0;
6
       SPI1STATbits.SPIEN = 1;
8
       // Clear Buffer Register
10     SPI1BUF = 0x0000;

12     IEC0bits.SPI1IE = 0; // SPI1 event interrupt
       SPI1STATbits.SPIEN = 0; // Switch SPI1 module off
14
       SPI1CON1bits.MODE16 = 1; // 16 bit bandwidth
16     SPI1CON1bits.MSTEN = 0; // Slave mode selected
       SPI1CON1bits.DISSCK = 0; // Just for master Mode
18     SPI1CON1bits.DISSDO = 1; // no SDO in slave mode

20     SPI1CON1bits.SMP = 0; // in slave mode always 0
       SPI1CON1bits.SSEN = 1; // SS1 activ for slave mode
22
       SPI1CON1bits.CKE = 0; // Data Change CLK idle to CLK active
24     SPI1CON1bits.CKP = 0; // CLK idle=0, CLK active=1

26     SPI1STATbits.SPISIDL = 0;
       IPC2bits.SPI1IP = 6;
28     SPI1STATbits.SPIROV = 0; // Clear error flag
       SPI1STATbits.SPIEN = 1;
30
       IFS0bits.SPI1IF = 0;
32     IFS0bits.SPI1EIF = 0;
       IEC0bits.SPI1IE = 1; // SPI1 event interrupt
34 }
```

Initialization of the PWM Modules

```c
#include <p33Exxxx.h>
#include "init.h"

// Working with independent Time Bases
void Init_PWM(void)
{
    PDC1 = 0; // Clear Duty Cycle Registers
    PDC2 = 0;

    PTPER = PWM_TPER;

    IOCON1 = 0x0000;
    IOCON2 = 0x0000;

    // Dead Time Settings
    PWMCON1bits.DTC = 0b00;
    PWMCON2bits.DTC = 0b00;

    DTR1 = 0;
    DTR2 = 0;

    ALTDTR1 = 0;
    ALTDTR2 = 0;

    IOCON1bits.PENH = 1; // PWM1 Module controls PWM1H Pin
    IOCON1bits.PENL = 1; // PWM1 Module controls PWM1L Pin
    IOCON2bits.PENH = 1; // PWM2 Module controls PWM2H Pin
    IOCON2bits.PENL = 1; // PWM2 Module controls PWM2L Pin
    IOCON1bits.POLH = 0; // PWM1H Pin is active-high
    IOCON1bits.POLL = 0; // PWMxL Pin is active-high
    IOCON2bits.POLH = 0; // PWM2H Pin is active-high
    IOCON2bits.POLL = 0; // PWM2L Pin is active-high
    IOCON1bits.PMOD = 0b00; // PWM1 I/O Pin Pair is in the
   Complementary Output Mode
    IOCON2bits.PMOD = 0b00; // PWM2 I/O Pin Pair is in the
   Complementary Output Mode
    IOCON1bits.OVRENH = 0; // PWM1 Generator controls PWM1H Pin
    IOCON1bits.OVRENL = 0; // PWM1 Generator controls PWM1L Pin
    IOCON2bits.OVRENH = 0; // PWM2 Generator controls PWM2H Pin
    IOCON2bits.OVRENL = 0; // PWM2 Generator controls PWM2L Pin
    IOCON1bits.SWAP = 0; // PWMxH and PWMxL Pins are mapped to their
   respective Pins
    IOCON2bits.SWAP = 0;
    IOCON1bits.FLTDAT = 0b00; // Values for PWM1H and PWM1L in case of
   fault
    IOCON2bits.FLTDAT = 0b00;

    PTCON = 0x0000;

    PTCON2 = 0x0000; // Divide by 1 to generate PWM

    PWMCON1bits.MDCS = 0; // Master duty cycle is not used
```

```
    PWMCON2bits.MDCS = 0;
    PWMCON1bits.CAM = 0; // Edge-aligned mode is enabled
    PWMCON2bits.CAM = 0;
    PWMCON1bits.ITB = 0; // PTPER register provides timing for this PWM generator
    PWMCON2bits.ITB = 0;
    PWMCON1bits.TRGIEN = 0; // A trigger event generates an interrupt request
    PWMCON2bits.TRGIEN = 0;
    PWMCON1bits.CLIEN = 0; // Current limit interrupt is disabled
    PWMCON2bits.CLIEN = 0;
    PWMCON1bits.FLTIEN = 0; // Fault interrupt is disabled
    PWMCON2bits.FLTIEN = 0;
    PWMCON1bits.TRGSTAT = 0; // No trigger interrupt is pending
    PWMCON2bits.TRGSTAT = 0;
    PWMCON1bits.CLSTAT = 0; // No current limit interrupt is pending
    PWMCON2bits.CLSTAT = 0;
    PWMCON1bits.FLTSTAT = 0; // No fault interrupt is pending
    PWMCON2bits.FLTSTAT = 0;
    PWMCON1bits.XPRES = 0; // External pins do not affect PWM operation
    PWMCON2bits.XPRES = 0;

    PDC1 = PTPER / 2; // Initialize as 0 Voltage
    PDC2 = PTPER / 2; // Initialize as 0 Voltage

    PTCONbits.SEVTPS = 0; // Special Event Trigger output postscaler set to 1:2 selection

    PTCONbits.EIPU = 0; // Update active period register immediately
    PWMCON1bits.IUE = 0;
    PWMCON2bits.IUE = 0;

    //Interrupt Configuration
    IPC23bits.PWM1IP = 4; // Set Interrupt Priorities
    IFS5bits.PWM1IF = 0; // Clear Flag
    IEC5bits.PWM1IE = 0; // Enable Interrupt
    IPC23bits.PWM2IP = 4;
    IFS5bits.PWM2IF = 0;
    IEC5bits.PWM2IE = 0;
    IPC14bits.PSEMIP = 4;
    IFS3bits.PSEMIF = 0;
    IEC3bits.PSEMIE = 1; // Enable special event interrupt

    // Fault Configuration
    FCLCON1bits.FLTSRC = 0b00000;
    FCLCON2bits.FLTSRC = 0b00000;
    FCLCON1bits.FLTMOD = 0b11; // Disable fault input
    FCLCON2bits.FLTMOD = 0b11;

    // Phase Shift for PWM2
    PHASE2 = PWM_PHASE2;
```

```
98      PTCONbits.PTEN = 1; // Enable PWM module
}
```

ADC Initialization

```
#include <p33Exxxx.h>

// Settings for 10 Bit, 2 Channels
void Init_ADCs(void)
{
    // Port Configuration
    ANSELA = 0x0000;
    ANSELAbits.ANSA0 = 1;// Ensure AN0/RA0 is analog
    ANSELAbits.ANSA1 = 1;// Ensure AN1/RA1 is analog
    TRISAbits.TRISA0 = 1; // ADC SIN_RTN Input
    TRISAbits.TRISA1 = 1; // ADC COS_RTN Input

    AD1CON1bits.AD12B = 0; // 10-bit ADC operation
    AD1CON2bits.VCFG = 0b000; // Select AVDD, AVSS as reference supply
    AD1CON3bits.ADRC = 0; // ADC Clock is derived from Systems Clock
    AD1CON3bits.SAMC = 0; // Auto Sample Time = 0 * TAD

    AD1CON1bits.SSRCG = 0;
    AD1CON1bits.SSRC = 0b011;

    AD1CON1bits.ASAM = 1; // ADC Sample Control: Sampling begins
    immediately after last conversion

    AD1CON1bits.FORM = 0b00; // Data Output Format: Unsigned Integer

    AD1CON1bits.SIMSAM = 1; // Simultanous sampling

    AD1CON2bits.CHPS = 0b01; // Converts channels CH0, CH1
    AD1CON2bits.BUFM = 0; // Single 16-Word Result Buffer

    AD1CHS0bits.CH0SA = 0b00001; // Channel 0 positive input is AN1
    AD1CHS0bits.CH0NA = 0; // Select Vref- for CH0 -ve input
    AD1CON2bits.CSCNA = 0; // No input scan
    AD1CSSH = 0x0000;
    AD1CSSL = 0x0000;

    AD1CHS123bits.CH123SA = 0; // AN0 -> CH1, AN1 -> CH2
    AD1CHS123bits.CH123NA = 0b00; // VREFL as negative input
    AD1CON2bits.ALTS = 0; // Alternate MUXA, MUXB input select disabled

    AD1CON3bits.ADCS = 7;
    // Minimal value for Tad: Tad_min = 117.5 ns
    // ADC Conversion Clock Tad=Tcy*(ADCS+1)= (1/65M)*8 = 123ns (8,13
    MHz)
    // ADC Sample Time: Tsamp = 3 * Tad = 369ns
    // ADC Conversion Time for 12-bit: Tconv = 14 * Tad = 1.72 us
```

```
45
     AD1CON4bits.ADDMAEN = 0; // Conversion results stored in ADCxBUF0
    register
47
     AD1CON2bits.SMPI = 0; // Interrupt is generated after every sample
49
     IPC3bits.AD1IP = 4; // Set Interrupt Priority
51
     IFS0bits.AD1IF = 0; // Clear the Analog-to-Digital interrupt flag
    bit
53   IEC0bits.AD1IE = 1; // Enable Analog-to-Digital interrupt

55   AD1CON1bits.ADON = 1; // Turn on the ADC
}
```

Timer1 Initialization

```
#include <p33Exxxx.h>
2
void Init_Timer1(void)
4{
     T1CON = 0;                // Timer reset
6    IFS0bits.T1IF = 0;        // Reset Timer1 interrupt flag
     IPC0bits.T1IP = 3;        // Timer1 Interrupt priority level=4
8    IEC0bits.T1IE = 1;        // Enable Timer1 interrupt

10   PR1 = 0xFDE8;             // Timer1 period register = 65000000 / 1000
     = 65000
     T1CONbits.TCS = 0;        // Clock = F_CY
12
     T1CONbits.TON = 1;        // Enable Timer1 and start the counter
14}
```

SPI1 Interrupt Service Routine

```
#include <p33Exxxx.h>
2#include "init.h"

4void __attribute__((__interrupt__, no_auto_psv)) _SPI1Interrupt(void)
    // Duration: approx.
{
6//     LATBbits.LATB5 = 1; // Set Utility Pad high

8    extern volatile signed int refSIN, refCOS;
     extern volatile signed int refSIN_PT1, refCOS_PT1;
10   extern volatile unsigned int refSIN_PWM, refCOS_PWM;

12   signed int buffered_Data;
     signed int temp = 0;
```

```
14      extern volatile unsigned char SPI_ERROR;
        extern volatile int MotorDriver_STOP;
16      extern volatile int SPI_DEADMAN_FLAG;

18      buffered_Data = SPI1BUF;

20      // Compute received Data
        if (SPI_ERROR == 0)
22      {
            SPI_DEADMAN_FLAG = 0;
24
            // Parity Check
26          if (buffered_Data > 0) // SIN: D15 = 0
            {
28              MotorDriver_STOP = 0;
                if ((buffered_Data & CONT_PATTERN_MASK)== SIN_CONT_PATTERN)
        // Parity check
30              {
                    // Separation of data and check bits
32                  buffered_Data = buffered_Data & SPI1_DATA_MASK;
                    temp = (buffered_Data >> 3); // Remove parity bits
34                  refSIN_PWM = temp / PWM_FACTOR;
                    refSIN = temp - SPI_HALF_RESOLUTION;  // Remove offset
36                  CALC_PT1_FILTERED_SIN();

38                  // Check Boundaries
                    if(refSIN_PT1 > SPI_HALF_RESOLUTION - 1)
40                  {
                        refSIN_PT1 = SPI_HALF_RESOLUTION - 1;
42                  }
                    if(refSIN_PT1 < SPI_NEGATIVE_HALF_RESOLUTION)
44                  {
                        refSIN_PT1 = SPI_NEGATIVE_HALF_RESOLUTION;
46                  }

48                  CALC_PT1_FILTERED_SIN_PWM();
                    PDC1 = refSIN_PWM;
50              }
                else
52              {
                    SPI_ERROR = 1;
54                  refSIN = 0;
                    refCOS = 0;
56                  MotorDriver_STOP = 1;
                }
58          }
            else if(buffered_Data == 0) // STOP
60          {
                MotorDriver_STOP = 1;
62              refSIN = 0;
                refCOS = 0;
64              PDC1 = PWM_ZERO;
```

```
            PDC2 = PWM_ZERO;
        }
        else // COS: D15 = 1
        {
            MotorDriver_STOP = 0;
            if ((buffered_Data & CONT_PATTERN_MASK) == COS_CONT_PATTERN
    ) // Parity check
            {
                // Separation of data and check bits
                buffered_Data = buffered_Data & SPI1_DATA_MASK;
                temp = (buffered_Data >> 3); // Remove parity bits
                refCOS_PWM = temp / PWM_FACTOR;
                refCOS = temp - SPI_HALF_RESOLUTION; // Remove offset
                CALC_PT1_FILTERED_COS();

                // Check Boundaries
                if(refCOS_PT1 > SPI_HALF_RESOLUTION - 1)
                {
                    refCOS_PT1 = SPI_HALF_RESOLUTION - 1;
                }
                if(refCOS_PT1 < SPI_NEGATIVE_HALF_RESOLUTION)
                {
                    refCOS_PT1 = SPI_NEGATIVE_HALF_RESOLUTION;
                }

                CALC_PT1_FILTERED_COS_PWM();
                PDC2 = refCOS_PWM;
            }
            else
            {
                SPI_ERROR = 1;
                refSIN = 0;
                refCOS = 0;
                MotorDriver_STOP = 1;
            }
        }
        Set_ADC_Trigger();
    }

    SPI1STATbits.SPIROV = 0; // Received overflow flag: no overflow has
    occured

    IFS0bits.SPI1IF = 0; // Clear SPI1 interrupt flag
    IFS0bits.SPI1EIF = 0; // Clear SPI1 error interrupt flag

//     LATBbits.LATB5 = 0; // Set Utility Pad low
}

void Set_ADC_Trigger(void)
{
//     LATBbits.LATB5 = 1; // Set Utility Pad high

```

```
    extern volatile signed int refSIN_PT1, refCOS_PT1;
    extern volatile signed int refSIN_PT1_PWM, refCOS_PT1_PWM;

    int quadrant = GetQuadrant(refSIN_PT1, refCOS_PT1);

    if(refSIN_PT1_PWM >= PWM_MAX) refSIN_PT1_PWM = PWM_MAX;
    if(refCOS_PT1_PWM >= PWM_MAX) refCOS_PT1_PWM = PWM_MAX;

//    int pwm_DC_difference = refSIN_PT1_PWM - refCOS_PT1_PWM;

    switch(quadrant)
    {
        case 1:
        {
//            LATBbits.LATB5 = 1; // Set Utility Pad high
            if(refSIN_PT1_PWM >= refCOS_PT1_PWM) SEVTCMP =
   refCOS_PT1_PWM / 2;
            if(refSIN_PT1_PWM < refCOS_PT1_PWM) SEVTCMP =
   refSIN_PT1_PWM / 2;
//            LATBbits.LATB5 = 0; // Set Utility Pad low
            break;
        }
        case 2:
        {
//            LATBbits.LATB5 = 1; // Set Utility Pad high
            SEVTCMP = PDC1 - 1;
//            LATBbits.LATB5 = 0; // Set Utility Pad low
            break;
        }
        case 3:
        {
//            LATBbits.LATB5 = 1; // Set Utility Pad high
            if(refSIN_PT1_PWM >= refCOS_PT1_PWM) SEVTCMP =
   refSIN_PT1_PWM / 2 + PWM_ZERO;
            if(refSIN_PT1_PWM < refCOS_PT1_PWM) SEVTCMP =
   refCOS_PT1_PWM / 2 + PWM_ZERO;
//            LATBbits.LATB5 = 0; // Set Utility Pad low
            break;
        }
        case 4:
        {
//            LATBbits.LATB5 = 1; // Set Utility Pad high
            SEVTCMP = PDC2 - 1;
//            LATBbits.LATB5 = 0; // Set Utility Pad low
            break;
        }
        case 0:
        {
            refSIN_PT1_PWM = PWM_ZERO;
            refCOS_PT1_PWM = PWM_ZERO;
            SEVTCMP = refSIN_PT1_PWM;
            break;
```

```
        }
    }
}

// Returns the quadrant (1, 2, 3, 4) calculated by SIN, COS
int GetQuadrant(signed int SIN, signed int COS)
{
    if((SIN >= 0) & (COS > 0)) return 1; // Q1
    if((SIN > 0) & (COS <= 0)) return 2; // Q2
    if((SIN <= 0) & (COS < 0)) return 3; // Q3
    if((SIN < 0) & (COS >= 0)) return 4; // Q4
    else return 0; // SIN = 0, COS = 0
}
```

Timer1 Interrupt Service Routine

```
#include <p33Exxxx.h>
#include "init.h"

int milliseconds = 0;

// Interrupt is generated every ms
void __attribute__((interrupt, no_auto_psv)) _T1Interrupt( void )
{
//    LATBbits.LATB5 = 1; // Set Utility Pad high
    IFS0bits.T1IF = 0; // Clear interrupt flag

    milliseconds++;
    if(milliseconds == 1000)
    {
        SecondTickEvent();
        milliseconds = 0;
    }

//    LATBbits.LATB5 = 0; // Set Utility Pad low
}

void SecondTickEvent(void)
{
//    LATBbits.LATB5 = 1; // Set Utility Pad high

    extern volatile int SPI_DEADMAN_FLAG;

    int i;
    for(i = 0; i < 1000; i++) Nop();

    SPI_DEADMAN_FLAG = 1;

//    LATBbits.LATB5 = 0; // Set Utility Pad low
}
```

References

[1] RAINHARD HABERFELLNER, ERNST FRICKE, OLIVIER DE WECK, SIEGFRIED VÖSSNER: *Systems Engineering: Grundlagen und Anwendung*. Orell Füssli Verlag AG Zürich, 2012

[2] HUGHES AUSTIN, DRURY BILL: *Electric Motors and Drives: Fundamentals, Types and Applications*. Newnes, 2013. – ISBN: 978-0-08-098332-5

[3] MICROCHIP: *dsPIC33EP256MC202 Data Sheet*. http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en558753. Version: 2013. – [Online; last visited: 09/OCT/2013]

[4] MICROCHIP: *dsPIC33FJ128MC802 Data Sheet*. http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en532302. Version: 2013. – [Online; last visited: 09/OCT/2013]

[5] MICROCHIP: *Microstick II Data Sheet*. http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en556208. Version: 2013. – [Online; last visited: 09/OCT/2013]

[6] MANEA SORIN: *Stepper Motor Control with dsPIC DSCs*. http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1824&appnote=en546027. Version: JUL 2011. – [Online; last visited: 10/OCT/2013]

[7] STMICROELECTRONICS: *dSPIN fully integrated microstepping motor driver with motion engine and SPI Data Sheet*. http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/CD00255075.pdf. Version: DEC 2012. – [Online; last visited: 19/OCT/2013]

[8] INSTRUMENTS TEXAS: *Dual H-Bridge Driver IC Data Sheet*. http://www.ti.com/lit/ds/slvsac0e/slvsac0e.pdf. Version: August 2013. – [Online; last visited: 19/OCT/2013]

[9] STMICROELECTRONICS: *Fully integrated stepper motor driver mounting the L6470 Data Brief*. http://www.st.com/st-web-ui/static/active/en/resource/technical/document/data_brief/DM00047222.pdf. Version: April 2012. – [Online; last visited: 19/OCT/2013]

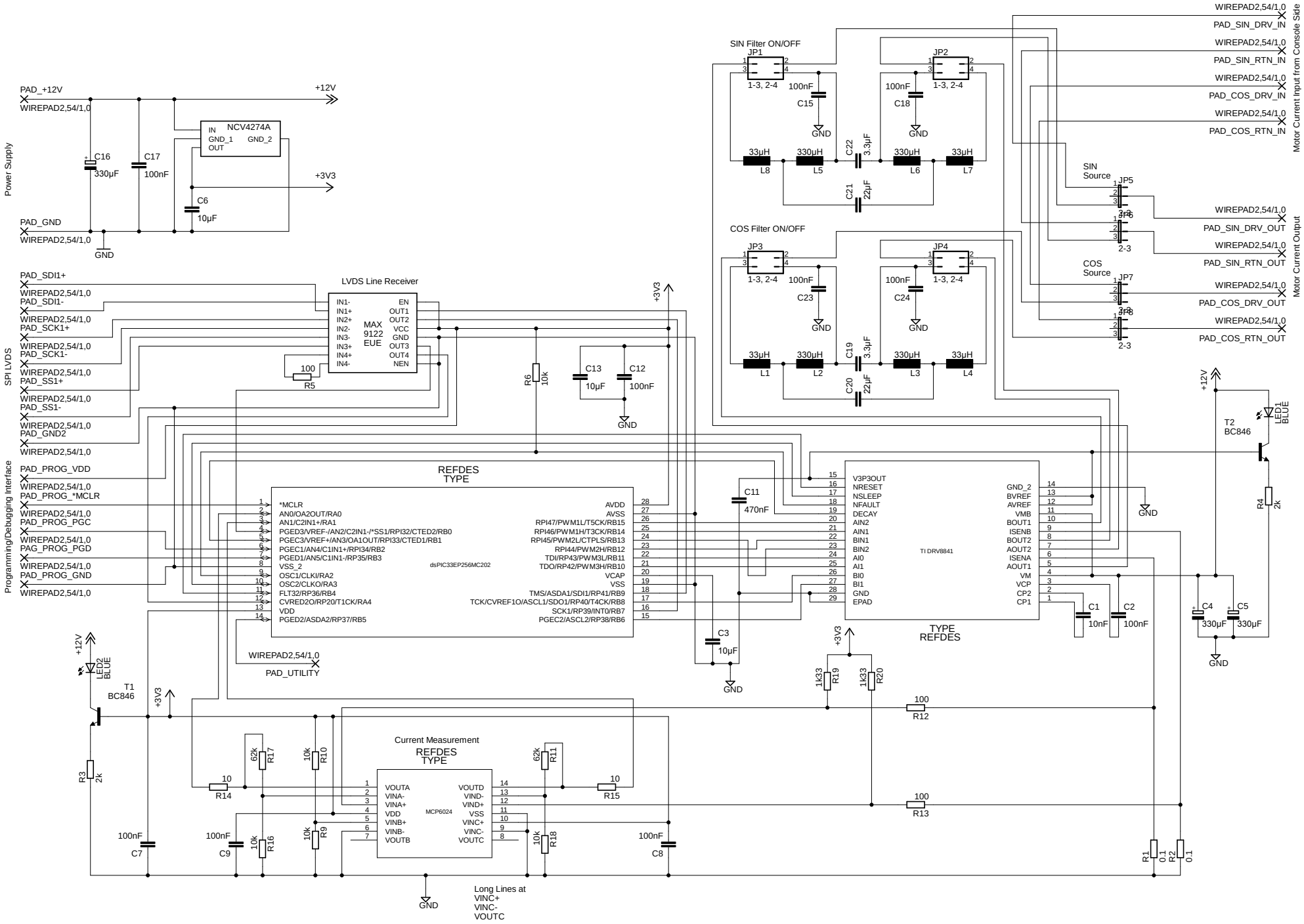
[10] INSTRUMENTS TEXAS: *CPG004_DRV88xx Evaluation Modules*. http://www.ti.com/lit/ug/slvu361a/slvu361a.pdf. Version: April 2011. – [Online; last visited: 19/OCT/2013]

[11] DANTER MANFRED: *RAB4-8-D Probe Data Sheet*. August 2007

[12] MICROCHIP: *PICkit 3 In-Circuit Debugger/Programmer User Guide*. 2013

[13] HOFER ANTON: *Skriptum zur Vorlesung Regelungstechnik I*. http://www.irt.tugraz.at. Version: 2011

[14] HOFER ANTON: *Skriptum zur Vorlesung Regelungstechnik II*. http://www.irt.tugraz.at. Version: 2008


Power Supply
PAD_+12V
WIREPAD2,54/1,0
+12V
NCV4274A
IN
GND_1
GND_2
OUT
C16
330µF
C17
100nF
C6
10µF
+3V3
PAD_GND
WIREPAD2,54/1,0
GND
SPI LVDS
PAD_SDI1+
PAD_SDI1-
PAD_SCK1+
PAD_SCK1-
PAD_SS1+
PAD_SS1-
PAD_GND2
LVDS Line Receiver
MAX 9122 EUE
IN1-
IN1+
IN2+
IN2-
IN3-
IN3+
IN4+
IN4-
EN
OUT1
OUT2
VCC
GND
OUT3
OUT4
NEN
R5
100
R6
10k
C13
10µF
C12
100nF
Programming/Debugging Interface
PAD_PROG_VDD
PAD_PROG_*MCLR
PAD_PROG_PGC
PAG_PROG_PGD
PAD_PROG_GND
REFDES
TYPE
dsPIC33EP256MC202
*MCLR
AN0/OA2OUT/RA0
AN1/C2IN1+/RA1
PGED3/VREF-/AN2/C2IN1-/*SS1/RPI32/CTED2/RB0
PGEC3/VREF+/AN3/OA1OUT/RPI33/CTED1/RB1
PGEC1/AN4/C1IN1+/RPI34/RB2
PGED1/AN5/C1IN1-/RP35/RB3
VSS_2
OSC1/CLKI/RA2
OSC2/CLKO/RA3
FLT32/RP36/RB4
CVRED2O/RP20/T1CK/RA4
VDD
PGED2/ASDA2/RP37/RB5
AVDD
AVSS
RPI47/PWM1L/T5CK/RB15
RPI46/PWM1H/T3CK/RB14
RPI45/PWM2L/CTPLS/RB13
RPI44/PWM2H/RB12
TDI/RP43/PWM3L/RB11
TDO/RP42/PWM3H/RB10
VCAP
VSS
TMS/ASDA1/SDI1/RP41/RB9
TCK/CVREF1O/ASCL1/SDO1/RP40/T4CK/RB8
SCK1/RP39/INT0/RB7
PGEC2/ASCL2/RP38/RB6
WIREPAD2,54/1,0
PAD_UTILITY
C11
470nF
C3
10µF
TI DRV8841
V3P3OUT
NRESET
NSLEEP
NFAULT
DECAY
AIN2
AIN1
BIN1
BIN2
AI0
AI1
BI0
BI1
GND
EPAD
GND_2
BVREF
AVREF
VMB
BOUT1
ISENB
BOUT2
AOUT2
ISENA
AOUT1
VM
VCP
CP2
CP1
C1
10nF
C2
100nF
C4
330µF
C5
330µF
TYPE
REFDES
SIN Filter ON/OFF
JP1
JP2
1-3, 2-4
100nF
C15
C18
33µH
330µH
L8
L5
L6
L7
C22
3.3µF
C21
22µF
COS Filter ON/OFF
JP3
JP4
C23
C24
L1
L2
L3
L4
C19
C20
SIN Source
COS Source
JP5
JP6
JP7
JP8
2-3
WIREPAD2,54/1,0
PAD_SIN_DRV_IN
PAD_SIN_RTN_IN
PAD_COS_DRV_IN
PAD_COS_RTN_IN
PAD_SIN_DRV_OUT
PAD_SIN_RTN_OUT
PAD_COS_DRV_OUT
PAD_COS_RTN_OUT
Motor Current Input from Console Side
Motor Current Output
+12V
LED1
BLUE
T2
BC846
R4
2k
1K33
R19
R20
100
R12
R13
R1
0.1
R2
0.1
T1
BC846
LED2
BLUE
R3
2k
Current Measurement
REFDES
TYPE
MCP6024
VOUTA
VINA-
VINA+
VDD
VINB+
VINB-
VOUTB
VOUTD
VIND-
VIND+
VSS
VINC+
VINC-
VOUTC
10
R14
R15
62k
R17
R11
10k
R10
R9
R16
R18
100nF
C7
C9
C8
Long Lines at
VINC+
VINC-
VOUTC

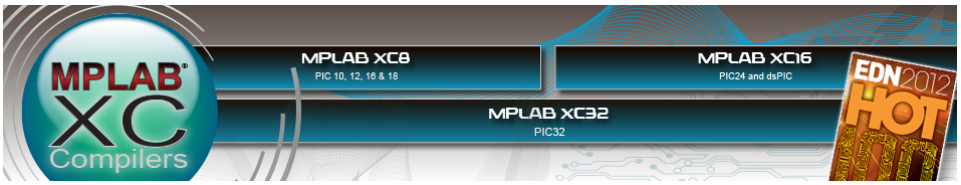
List of Components

Part	Value	Package	Material	Manufacturer	Manufacturer Part Nr.	Comment
PCB	-	-	FR-4	Beta Layout GmbH	-	PCB Motor Controller Board
C1	10 nF	C0402	Ceramic	Taiyo Yuden	UMK105BJ103KV-F	0402 Inch Package
C2	100 nF	C0402	Ceramic	TDK	C1005X5R1H104K050BB	0402 Inch Package
C3	10 µF	C1210	Ceramic	Taiyo Yuden	TMK325BJ106MN-T	1210 Inch Package, Maximal Tracelength to VCAP-Pin: 6mm
C4	330 µF	PANASONIC_F	Aluminium	Panasonic	EEEFT1V331AP	-
C5	330 µF	PANASONIC_F	Aluminium	Panasonic	EEEFT1V331AP	-
C6	10 µF	C1210	Ceramic	Taiyo Yuden	TMK325BJ106MN-T	1210 Inch Package, Maximal Tracelength to OUT-Pin: 6mm
C7	100 nF	C0402	Ceramic	TDK	C1005X5R1H104K050BB	0402 Inch Package, Maximal Tracelength to VDD-Pin: 6mm
C8	100 nF	C0402	Ceramic	TDK	C1005X5R1H104K050BB	0402 Inch Package, Maximal Tracelength to AVDD-Pin: 6mm
C9	100 nF	C0402	Ceramic	TDK	C1005X5R1H104K050BB	0402 Inch Package, Maximal Tracelength to VDD-Pin: 2mm
C10	4.7 nF	C0402	Ceramic	Taiyo Yuden	UMK105B7472KV-F	0402 Inch Package
C11	470 nF	C0402	Ceramic	Taiyo Yuden	LMK105BJ474MV-F	0402 Inch Package
C12	100 nF	C0402	Ceramic	TDK	C1005X5R1H104K050BB	0402 Inch Package
C13	10 µF	C1210	Ceramic	Taiyo Yuden	TMK325BJ106MN-T	1210 Inch Package, Maximal Tracelength to VCAP-Pin: 6mm
C14	4.7 nF	C0402	Ceramic	Taiyo Yuden	UMK105B7472KV-F	0402 Inch Package
C15	100 nF	C0402	Ceramic	TDK	C1005X5R1H104K050BB	0402 Inch Package, Maximal Tracelength to VDD-Pin: 6mm
C16	330 µF	PANASONIC_F	Aluminium	Panasonic	EEEFT1V331AP	-
C17	100 nF	C0402	Ceramic	TDK	C1005X5R1H104K050BB	0402 Inch Package, Maximal Tracelength to IN-Pin: 6mm
C18	100 nF	C0402	Ceramic	TDK	C1005X5R1H104K050BB	0402 Inch Package, Maximal Tracelength to VDD-Pin: 6mm
C19	3.3 µF	C1210	Ceramic	TDK	C3225X7R1E335K160AA	1210 Inch Package
C20	22 µF	C1206	Ceramic	TDK	C3216X5R1V226M160AC	1206 Inch Package
C21	22 µF	C1206	Ceramic	TDK	C3216X5R1V226M160AC	1206 Inch Package
C22	3.3 µF	C1210	Ceramic	TDK	C3225X7R1E335K160AA	1210 Inch Package
C23	100 nF	C0402	Ceramic	TDK	C1005X5R1H104K050BB	0402 Inch Package, Maximal Tracelength to VDD-Pin: 6mm
C24	100 nF	C0402	Ceramic	TDK	C1005X5R1H104K050BB	0402 Inch Package, Maximal Tracelength to VDD-Pin: 6mm
JP1	-	JP2Q	-	Preci-Dip	802-80-004-30-480101	-
JP2	-	JP2Q	-	Preci-Dip	802-80-004-30-480101	-
JP3	-	JP2Q	-	Preci-Dip	802-80-004-30-480101	-
JP4	-	JP2Q	-	Preci-Dip	802-80-004-30-480101	-
JP5	-	JP2	-	Preci-Dip	800-80-003-30-480101	-
JP6	-	JP2	-	Preci-Dip	800-80-003-30-480101	-
JP7	-	JP2	-	Preci-Dip	800-80-003-30-480101	-
JP8	-	JP2	-	Preci-Dip	800-80-003-30-480101	-
L1	33 µH	SM-NE29	-	Würth	7447709331	No Standard Package, Data Sheet appended
L2	330 µH	SM-NE29	-	BI Technologies	HM78-60330LFTR	No Standard Package, Data Sheet appended
L3	330 µH	SM-NE29	-	BI Technologies	HM78-60330LFTR	No Standard Package, Data Sheet appended
L4	33 µH	SM-NE29	-	Würth	7447709331	No Standard Package, Data Sheet appended
L5	330 µH	SM-NE29	-	BI Technologies	HM78-60330LFTR	No Standard Package, Data Sheet appended
L6	330 µH	SM-NE29	-	BI Technologies	HM78-60330LFTR	No Standard Package, Data Sheet appended
L7	33 µH	SM-NE29	-	Würth	7447709331	No Standard Package, Data Sheet appended
L8	33 µH	SM-NE29	-	Würth	7447709331	No Standard Package, Data Sheet appended
LED1	BLUE	CHIP-LED0805	-	OSRAM	LB QH9G-N1P2-35-1	0402 Inch Package
LED2	BLUE	CHIP-LED0805	-	OSRAM	LB QH9G-N1P2-35-1	0402 Inch Package
R1	0.1 Ω	R2512	-	Bourns	CRM2512-FX-R100ELF	2512 Inch Package
R2	0.1 Ω	R2512	-	Bourns	CRM2512-FX-R100ELF	2512 Inch Package
R3	2 kΩ	R0402	-	TE Connectivity	CRG0402F2K0	0402 Inch Package
R4	2 kΩ	R0402	-	TE Connectivity	CRG0402F2K0	0402 Inch Package
R5	100 Ω	R0402	-	TE Connectivity	CPF0402B100RE1	0402 Inch Package
R6	10 kΩ	R0402	-	Panasonic	ERA2AEB103X	0402 Inch Package
R7	62 kΩ	R0402	-	TE Connectivity	CPF0402B62KE	0402 Inch Package
R8	62 kΩ	R0402	-	TE Connectivity	CPF0402B62KE	0402 Inch Package
R9	10 kΩ	R0402	-	Panasonic	ERA2AEB103X	0402 Inch Package
R10	10 kΩ	R0402	-	Panasonic	ERA2AEB103X	0402 Inch Package
R11	62 kΩ	R0402	-	TE Connectivity	CPF0402B62KE	0402 Inch Package
R12	100 Ω	R0402	-	TE Connectivity	CPF0402B100RE1	0402 Inch Package
R13	100 Ω	R0402	-	TE Connectivity	CPF0402B100RE1	0402 Inch Package
R14	10 Ω	R0402	-	TE Connectivity	CPF0402B10RE	0402 Inch Package
R15	10 Ω	R0402	-	TE Connectivity	CPF0402B10RE	0402 Inch Package
R16	10 kΩ	R0402	-	Panasonic	ERA2AEB103X	0402 Inch Package
R17	62 kΩ	R0402	-	TE Connectivity	CPF0402B62KE	0402 Inch Package
R18	10 kΩ	R0402	-	Panasonic	ERA2AEB103X	0402 Inch Package
R19	1.33 kΩ	-	-	-	-	-
R20	1.33 kΩ	-	-	-	-	-
T1	-	SOT23	-	DiodesZetex	BC846A-7-F	SOT23 Package
T2	-	SOT23	-	DiodesZetex	BC846A-7-F	SOT23 Package
U1	-	SOT223	-	ON Semiconductor	NCV4274AST33T3G	SOT223 Package
U2	-	SSOP28_300MC	-	Microchip	DSPIC33EP256MC202-H/SS	SSOP28 Package
U3	-	TSSOP14_MC	-	Microchip	MCP6024-E/ST	SSOP14 Package
U4	-	16TSSOP	-	Maxim	MAX9121EUE+	SSOP16 Package
U5	-	PWP28_5P18X3P1	-	TI	DRV8841	SSOP28 Package

Software Tools

Source: https://www.microchip.com/pagehandler/en-us/family/mplabx/, 18/FEB/2014

Figure 50: Microchip MPLAB X Integrated Development Environment



Source: https://www.microchip.com/pagehandler/en-us/devtools/mplabxc/home.html, 18/FEB/2014

Figure 51: Microchip XC16 Compiler

Source: http://ww1.microchip.com/downloads/en/devicedoc/c30_users_guide_51284f.pdf, 18/FEB/2014

Figure 52: Microchip C30 C Compiler

XC16 Compiler Version: v1.20
Assembler Version: ASM30

Source: http://msdn.microsoft.com/de-DE/vstudio, 18/FEB/2014

Figure 53: Microsoft Visual Studio 2008 Professional

Source: http://www.cadsoft.de/welcome-to-cadsoft/, 18/FEB/2014

Figure 54: EAGLE Schematics Design and PCB Layout

Source: http://www.linear.com/solutions/1083, 18/FEB/2014

Figure 55: LTSpice IV

Source: http://www.cadence.com/products/orcad/pages/default.aspx, 18/FEB/2014

Figure 56: Cadence OrCAD 16.6