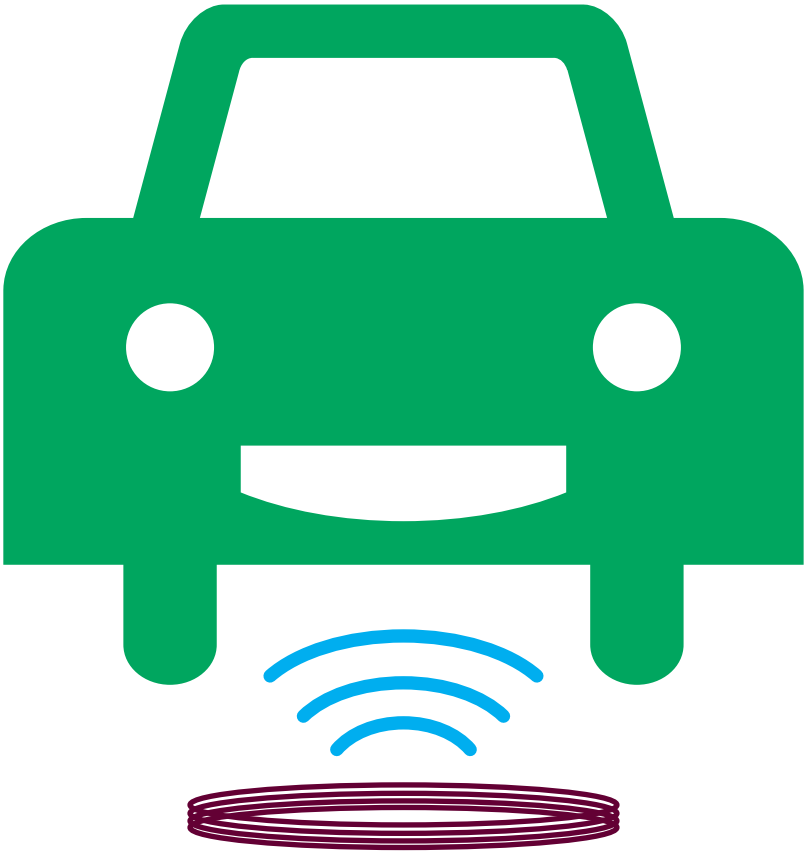
CHALMERS



# Construction of an Inductive Charging Station

## for an Electric Go Kart

*Master of Science Thesis*


CHRISTIAN EKMAN
CARL PETERSSON

Department of Energy and Environment
*Division of Electric Power Engineering*
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2014

# Construction of an Inductive Charging Station

## for an Electric Go Kart

CHRISTIAN EKMAN
CARL PETERSSON

Department of Energy and Environment
Division of Electric Power Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2014

Construction of an Inductive Charging Station
for an Electric Go Kart
CHRISTIAN EKMAN
CARL PETERSSON



Department of Energy and Environment
Division of Electric Power Engineering
Chalmers University of Technology
SE–412 96 Göteborg
Sweden
Telephone +46 (0)31–772 1000

Cover:
Inductive power transfer from a sending coil to an electric vehicle.

Chalmers Bibliotek, Reproservice
Göteborg, Sweden 2014

Construction of an Inductive Charging Station
for an Electric Go Kart
CHRISTIAN EKMAN
CARL PETERSSON
Department of Energy and Environment
Division of Electric Power Engineering
Chalmers University of Technology

# Abstract

An inductive charging station has been designed, constructed and is operational. It is able to transfer roughly 500W to the receiving side with a peak efficiency of approximately 85%. The system was fed with a voltage between 24 to 63V from a Direct Current (dc) supply. Then, it converted the dc voltage to a high frequency Alternating Current (ac) voltage. The ac voltage was fed to the coils which transferred the energy to the receiving side and to the resonance circuit. The conversion from ac to dc was made with the body diodes in the transistors.

**Index Terms:** Inductive charging, inverter, active rectification, rectifier, magnetic coils, resonance.

# Acknowledgements

We want start by thanking QRTECH as a company for having us there and for providing financial aid and support in all our endeavors. A special thanks also goes to Robert Karlsson at Chalmers University of Technology for his advice regarding coil design. We would also like to express our gratitude to everyone at QRTECH for their invaluable guidance, enthusiastic encouragement and for so generously giving their time when needed. Lastly, we want to express our appreciation to our advisor at QRTECH, Niclas Rasmussen, and our examiner, Torbjörn Thiringer, for their constructive suggestions and useful critique.

Carl & Christian
Göteborg, Sweden, 2014

# Abbreviations

| | |
|---|---|
| **ADC** | Analog-to-Digital Converter |
| **CRC** | Cyclic Redundancy Check |
| **EDA** | Electronic Design Automation |
| **EMC** | Electromagnetic Compatibility |
| **ESR** | Equivalent Series Resistance |
| **IPT** | Inductive Power Transfer |
| **JTAG** | Joint Test Action Group |
| **KCL** | Kirchoff's Current Law |
| **KVL** | Kirchoff's Voltage Law |
| **LF** | Low Frequency |
| **MCU** | Microcontroller Unit |
| **MOSFET** | Metal Oxide Semiconductor Field Effect Transistor |
| **PCB** | Printed Circuit Board |
| **RMS** | Root Mean Square |
| **SPI** | Serial Peripheral Interface |
| **VLF** | Very Low Frequency |
| **ZVS** | Zero-Voltage Switching |
| **Q factor** | Quality factor |
| **ac** | Alternating Current |
| **dc** | Direct Current |
| **eCAP** | Enhanced Capture |
| **ePWM** | Enhanced Pulse Width Modulator |

# Nomenclature

| | | |
|---|---|---|
| $C$ | [F] | Capacitance |
| $L$ | [H] | Inductance |
| $L_{12}$ | [H] | The mutual inductance between coils 1 and 2 |
| $MMF$ | [A-t] | Magnetomotive force |
| $N_i$ | | The number of turns of the coils where 1 denotes the primary side coil and 2 denotes the secondary side coil |
| $P$ | [W] | Active power |
| $Q$ | [VAr] | Reactive power |
| $S$ | [VA] | Apparent power |
| $W_i$ | | The coils in the magnetic circuit where 1 denotes the primary side and 2 denotes the secondary side symbol |
| $Z$ | [Ω] | Impedance |
| $\Phi$ | [Wb] | Magnetic flux |
| $\Phi_{12}$ | [Wb] | The mutual flux between coils 1 and 2 |
| $\omega_0$ | [rad/s] | The angular frequency at which the imaginary part of an impedance is zero |
| $\mathbf{S}_i$ | [$m^2$] | The surface of the coil where 1 denotes the primary side coil and 2 denotes the secondary side coil |
| $\vec{B}$ | [T] | Magnetic flux density |
| $\vec{E}$ | [V/m] | Electric field |
| $f_0$ | [Hz] | The frequency at which the imaginary part of an impedance is zero |
| $i$ | [A] | Electric current |
| $k$ | | The magnetic coupling between two coils |
| $k_f$ | | The coupling factor between two coils |
| $v$ | [V] | Voltage |

# Contents

# Chapter 1

# Introduction

This report covers the theory, simulations, design, construction and evaluation of an inductive charging station for an electric go kart which was provided by QRTECH at which this master's thesis was conducted.

## 1.1 Background

The climate is changing due to emissions of greenhouse gases in to the atmosphere. The change in the climate will cause more frequent and severe droughts, floods and storms along with several other environmental problems [1, 2]. The transportation sector emits large amounts of greenhouse gases. In 2011 the transportation sector emitted 28% of the total greenhouse gas emissions in the United States. Only electricity production had higher greenhouse gas emissions (33% of the total amount) [3]. If electric vehicles were used, it would be possible to use energy sources which have lower greenhouse gas emissions. This would lower the emission of greenhouse gases from the transportation sector and thus hopefully lowering the total amount of greenhouses gases that are released into the atmosphere every year.

Electric vehicles suffer from some drawbacks compared with cars that have internal combustion engines. The recharge time can be long, the range per charge is often shorter than the range on one full tank and the electric vehicle is often more expensive than a comparable ordinary vehicle. These factors deter consumers from buying electric vehicles [4]. A way of making electric vehicles more appealing to the customers can be to use inductive charging. Inductive charging will make the charging procedure more convenient for the owner since the charging can start immediately after the vehicle has been parked and no cables are needed to charge the vehicle [5]. The inductive charging system can be electrically insulated since no galvanic contact is needed. This will minimise the risk of electric shocks.

## 1.2 Aim

The aim of the project is to design and construct an inductive charging station. It should be dimensioned for an electric go kart which is provided by QRTECH. The charging station should function in a lab environment.

## 1.3 Problem

The objective of this master's thesis is to design and construct an inductive charging station dimensioned for an electric go kart. An overview of the charging system is illustrated in Figure 1.1.

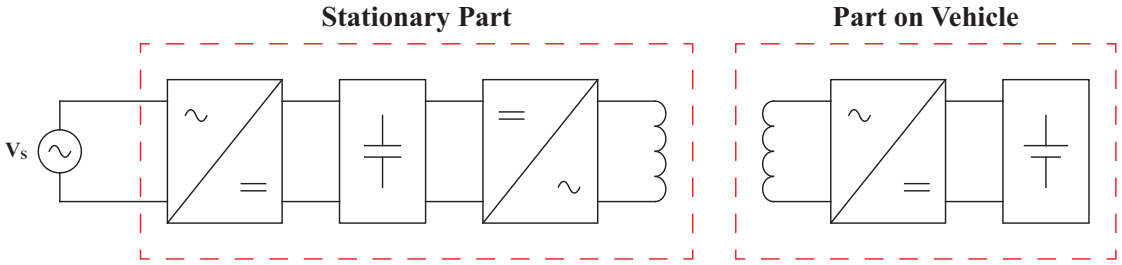



Figure 1.1: Layout of the inductive charging system.

The charging system includes power electronics, resonance circuits and coils. The power electronics will include an inverter and one synchronous rectifier. However, the synchronous rectifier will be designed to simplify future development of the charger and not for direct implementation. Instead, the body diodes of the transistors chosen for the synchronous rectifier will be used for rectification.

The inverter will provide an output which operates at the resonance frequency - 10 to 100kHz (Very Low Frequency (VLF) to Low Frequency (LF) band) - of both the sending and receiving side. The synchronous rectifier will be constructed using transistors as switches in the hope of achieving a higher efficiency than if rectification is handled by diodes. In order to ensure a smooth output dc voltage for the battery, a smoothing circuit consisting of a series inductor and parallel capacitors will be connected to the output of the rectifier on the sending side.

The resonance circuits will consist of the inductive coils and capacitors connected in parallel to the coils on the secondary side. The coils and capacitors will allow for resonance to be achieved as the receiving side of the charging system in principle will be a parallel resonance circuit.

The coils will be designed and constructed in such a way that they provide the correct voltage and power according to the specifications of the batteries on the electric go kart.

The design of the coils will be taken from a previous bachelor's thesis at Chalmers University of Technology and a conference paper from the University of Auckland.

## 1.4 Scope

The work will focus mainly on the design and construction of the power electronic converters needed on the receiving and sending side of the system. Dynamic charging - charging while driving - will not be considered.

In the design and construction of the power electronics, general system design, designing the Printed Circuit Board (PCB), programming the Microcontroller Unit (MCU), mounting the components and troubleshooting will be included. Requirements on Electromagnetic Compatibility (EMC) will not be considered.

The coils needed on the sending side and receiving side will be constructed based on an already existing solution, there will be little work done to optimize the coils, the main effort on the coils will be to manufacture them.

The project will be focused on the technical aspects. This means that the health effects from the electromagnetic waves will not be considered. Neither will the economic or environmental aspects of the project.

## 1.5 Method

The coils will be simulated using MATLAB/Simulink in order to form a basis of comparison for the wireless transfer of power (induction) that the coils will provide. The simulation will be performed using a model for a transformer as this is fundamentally the same object as the coils used for the wireless power transfer. The simulation will also serve to familiarize both of the team members to the concept of inductive charging.

The design of the system will include the layout of the system and what type and value of components that should be used.

Electronic Design Automation (EDA) software will be used for the design of circuit boards for the power electronics before adding components and soldering. The switch control of the transistors and the other functions of the MCU will be programmed using the programming language C.

The construction will take place in a lab environment and will be done by trial and error. This means that the evaluation and improvement of the system will occur simultaneously in parallel to the construction taking place.

## 1.6 Requirement Specification

Each battery cell has a maximum voltage level of 3.65V and the battery pack consists of two parallel connected groups of 15 battery cells in series. This means that the maximum voltage of the entire battery pack is $15 \cdot 3.65\text{V} = 54.75\text{V} \approx 55\text{V}$. The maximum charge current for the battery pack is 9A. This means that the maximum power transfer needed is $55\text{V} \cdot 9\text{A} = 495\text{W}$ on the receiving side. The efficiency target for the charger has been set to 90%. However, in order to leave some margin the sending side will be designed to be able to transfer enough power for an 80% efficiency. This would mean a sending side power of $495\text{W}/0.8 = 618.75\text{W} \approx 620\text{W}$.

# Chapter 2

# Theory

This chapter covers the theory involved in wireless transfer of power by means of induction. It includes both the physical means by which the transfer occurs but also ways to optimize the process.

## 2.1 Induction

When there is a current $I_1$ flowing through a closed coil $W_1$ there will also be a corresponding magnetic flux $\Phi_1$ . If a second coil, $W_2$, is placed in the vicinity of coil $W_1$ a part of the flux $\Phi_1$ will also flow through the secondary coil $W_2$. The part of the flux that flows through both coils is denoted as $\Phi_{12}$. This is the mutual flux of the coils. Faraday's law of induction states that the mutual flux will induce a current in the secondary coil. It is defined as

$$\nabla \times \vec{E} = -\frac{\partial \vec{B}}{\partial t} \tag{2.1}$$

where $\vec{E}$ is the electric field and $\vec{B}$ is the magnetic flux density [6, 7]. The mutual flux flowing through the surfaces of both coils can be formulated as

$$\Phi_{12} = \int_{\mathbf{S}_2} \vec{B}_1 \, \mathrm{d}s_2 \tag{2.2}$$

where $\mathbf{S}_2$ is the surface of the secondary coil $W_2$ and $\vec{B}_1$ is the magnetic flux density from the primary coil $W_1$ [6, 7]. Figure 2.1 illustrates what happens when there are two coils which transfer electric power through induction.

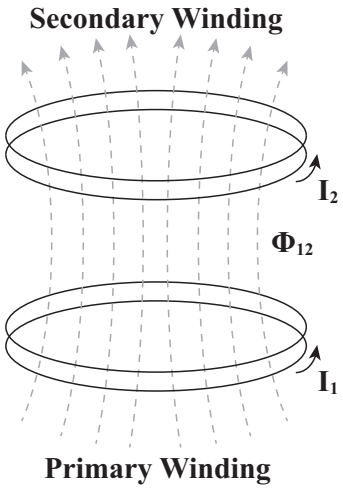



Figure 2.1: Illustration of the coils of an inductive power transfer system.

From the mutual flux given in (2.2) it is now possible to express the mutual inductance $L_{12}$ between the two coils as

$$L_{12} = \frac{\Phi_{12}}{I_1} \tag{2.3}$$

where $I_1$ is the primary side current.

### 2.1.1 Coupling Factor

The magnetic coupling between the primary and secondary coil forms the basis for the inductive power transfer. It is generally defined as

$$k = \frac{L_{12}}{\sqrt{L_1 L_2}} \tag{2.4}$$

where $L_1$ is the self-inductance of the primary coil and $L_2$ is the self-inductance of the secondary coil. However, in Inductive Power Transfer (IPT) systems the secondary coil typically covers a smaller area than the primary coil. This makes it difficult to know the distribution of the flux in the primary coil in particular. Since $L_1$ refers to the self-inductance of the entire primary coil the calculated coupling coefficient will be quite small in comparison to the actual coupling that is achieved since most of the primary coil might not be coupled with the secondary coil [8]. The coupling factor $k_f$ for IPT systems can instead be expressed as

$$k_f = N_2 \frac{L_{12}}{L_2} \tag{2.5}$$

where $N_2$ is the number of turns of the secondary coil [8].

## 2.2 Resonance

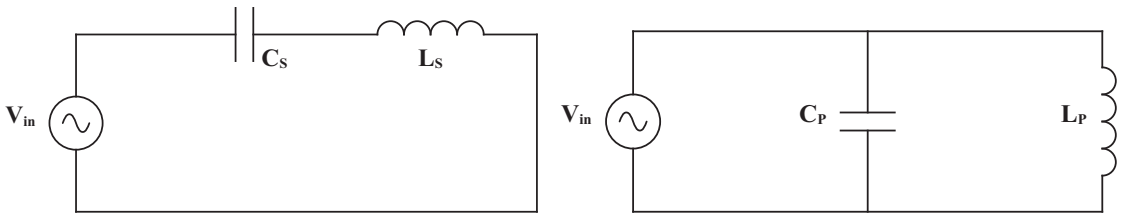
An alternating current flowing through an electric circuit will typically have a power transfer that containts both a real and imaginary part. The vector sum of the real and imaginary part is called the apparent power, $S$, and can be written as

$$S = P + jQ \tag{2.6}$$

where $P$ is the active power - the real part - and $Q$ is the reactive power - the imaginary part. For a certain frequency - the resonance frequency - the apparent power S will be equal to the active power $P$ and the reactive power $Q$ will be zero. Resonance occurs when the following happens. When the electric field in the capacitor dissipates the capacitor discharges through the inductor. This creates a magnetic field that stores the energy being fed into the inductor. The inductor then discharges current to the capacitor when the magnetic field dissipates. Thus, the capacitor is charged again and energy is stored in the electric field of the capacitor. This cycle will continue to repeat itself as long as the resonance frequency is maintained in the circuit. When this pendulum-like effect occurs the reactive power produced by the capacitor is equal to the reactive power consumed by the inductor and therefore the net reactive power in the circuit is zero.

### 2.2.1 Topologies

The two most basic resonance circuits can be seen in Figure 2.2. They are the series and parallel resonance circuits [7, 9].




(a) Series Resonance (b) Parallel Resonance

Figure 2.2: Series and parallel resonance circuits.

Both the series and parallel resonance circuit experience resonance when $\omega L = \frac{1}{\omega C}$ which means that the resonance frequency for both types of circuit can be described as

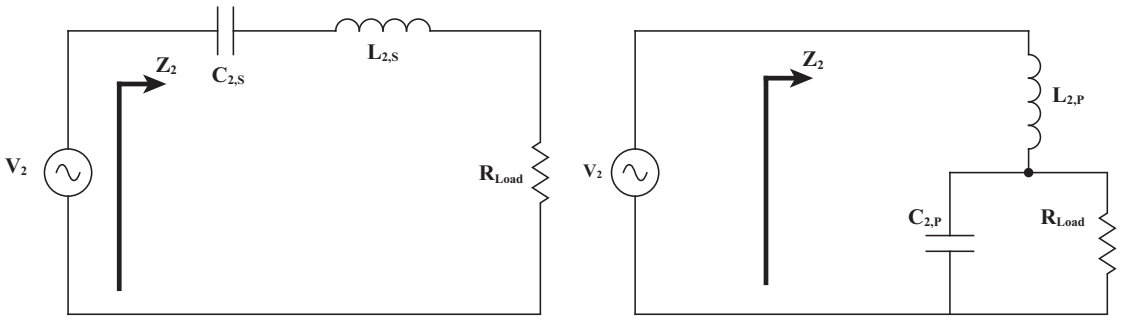
$$\omega_0 = \frac{1}{\sqrt{LC}} \implies f_0 = \frac{1}{2\pi\sqrt{LC}} \tag{2.7}$$

where $\omega_0$ is the resonance frequency in rad/s, $f_0$ is the resonance frequency in Hz, $L$ is the inductance and $C$ is the capacitance. The series resonance circuits is in resonance

when the series impedance is minimized while the parallel resonance circuits resonates when the parallel impedance is maximized [7, 9, 10].

### 2.2.2 Compensation

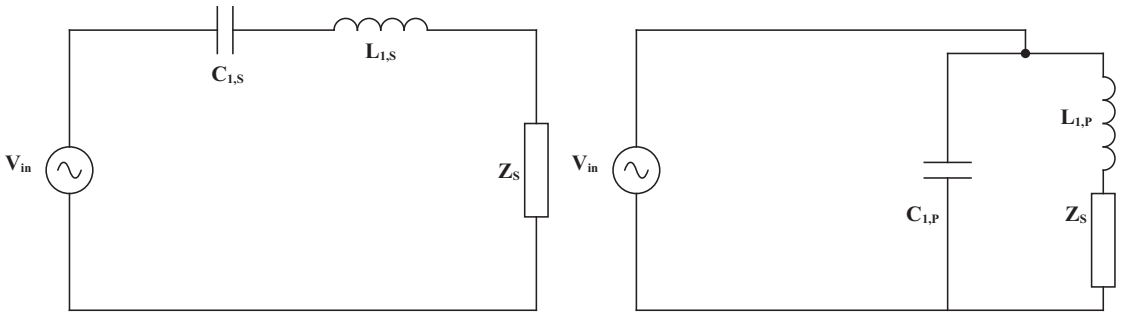
In order to achieve resonance at least one capacitor and one inductor is needed. As shown in Section 2.2.1 this typically results in either series or parallel resonance circuits. Both the series and parallel compensation connections are shown in Figure 2.3 referred to the secondary side [8, 10].




(a) Series Compensation (b) Parallel Compensation

Figure 2.3: Series and parallel compensation on the secondary side.

The series and parallel compensation connections referred to the primary side can be seen in Figure 2.4.




(a) Series Compensation (b) Parallel Compensation

Figure 2.4: Series and parallel compensation on the primary side.

$Z_S$ represents the total impedance of the secondary side, $Z_2$, as seen from the primary side. $Z_2$ is seen differently from the primary side depending on if the secondary side is

series or parallel compensated. In the case when the secondary side is series compensated $Z$ can be represented as

$$Z_S = \frac{\omega_0^2 L_{12}^2}{R_{Load}} \tag{2.8}$$

and in the case when the secondary side is parallel compensated $Z$ can be represented as

$$Z_S = \frac{\omega_0^2 L_{12}^2 \left(R_{Load} - j\omega_0 L_2\right)}{\omega_0^2 L_2^2} = \frac{L_{12}^2 R_{Load}}{L_2^2} - j\frac{\omega_0 L_{12}^2}{L_2} \tag{2.9}$$

From (2.8) it can be seen that in the case where the secondary side is parallel compensated the primary side will experience a phase shift due to the imaginary part of $Z$. This needs to be taken into consideration when choosing the value of the compensation capacitance on the primary side as this phase shift cannot be corrected from the secondary side. This also holds true for the secondary side if the primary side happens to be parallel compensated as the primary side will introduce a phase shift on the secondary side in that case. The necessary capacitances to be able to achieve resonance are presented in Table 2.1 with these phase shifts accounted for [10].

Table 2.1: The compensation needed to achieve resonance for different topologies.

| **Topology** | **Primary Side Compensation** | **Secondary Side Compensation** |
|---|---|---|
| series-series | $C_1 = \frac{1}{\omega_0^2 L_1}$ | $C_2 = \frac{1}{\omega_0^2 L_2}$ |
| series-parallel | $C_1 = \frac{1}{\omega_0^2\left(L_1 - \frac{L_{12}^2}{L_2}\right)}$ | $C_2 = \frac{1}{\omega_0^2 L_2}$ |
| parallel-series | $C_1 = \frac{1}{\omega_0^2 L_1}$ | $C_2 = \frac{1}{\omega_0^2\left(L_2 - \frac{L_{12}^2}{L_1}\right)}$ |
| parallel-parallel | $C_1 = \frac{1}{\omega_0^2\left(L_1 - \frac{L_{12}^2}{L_2}\right)}$ | $C_2 = \frac{1}{\omega_0^2\left(L_2 - \frac{L_{12}^2}{L_1}\right)}$ |

It is also possible to design IPT systems having compensation on only one side.

### 2.2.3 Q Factor

The Quality factor (Q factor) of a resonance circuit describes the amount of gain and the bandwidth at which it resonates around the centred frequency - the resonance frequency - of the resonance circuit. A resonance circuit having a higher Q factor will have a larger gain (the resonant circuit will be less damped) but it will also only resonate around a small range of frequencies around the resonance frequency. Therefore, a higher Q factor

allows for high frequency stability as it will not interact with any frequencies other than the resonance frequency but it will also be difficult to tune as the input frequency needs to be precise in order for resonance to occur. This difference between resonance circuits with different Q factors is illustrated in Figure 2.5 [7, 9].

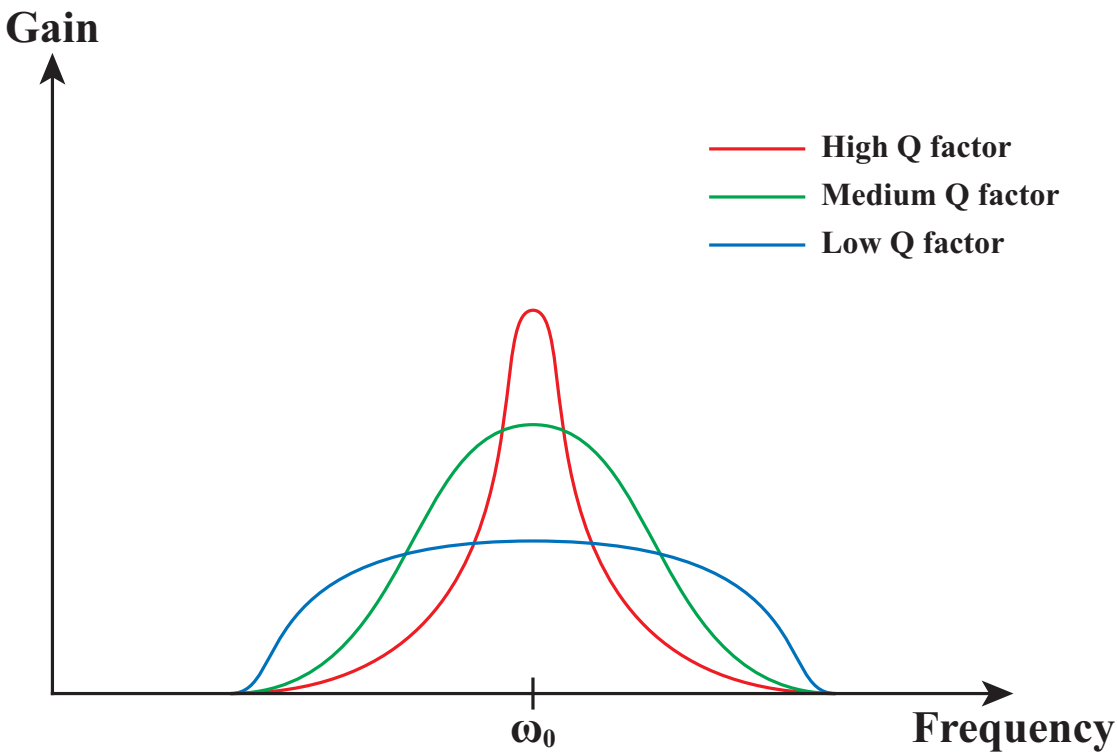



Figure 2.5: Difference in gain and frequency dependency with different Q factors.

In a series resonance circuit the Q factor can be expressed as

$$Q_{series} = \frac{\omega_0 L}{R} \tag{2.10}$$

and for parallel resonance circuits the Q factor is

$$Q_{parallel} = \frac{R}{\omega_0 L} \tag{2.11}$$

### 2.2.4 Maximum Power Transfer

The maximum power transfer in an IPT system can be related to the choice of frequency, Q factor and coupling factor. It can be expressed as

$$P_{Max} = \frac{\omega I_1^2 L_{12}^2 Q_2}{L_2} = \frac{\omega I_1^2 Q_2 k_f L_{12} N_1}{N_2} \tag{2.12}$$

where $Q_2$ is the Q factor of the secondary side [8].

## 2.3 The Skin Effect

When an alternating current is flowing through a conductor there will be a related magnetic field. The magnetic field will induce small currents in the conductor which in turn create their own magnetic fields [11]. This phenomenon is illustrated in Figure 2.6.

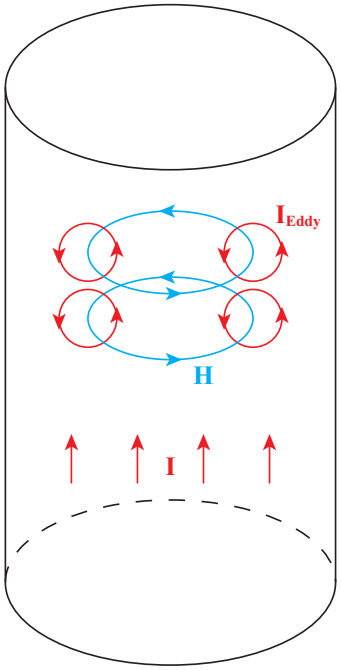



Figure 2.6: The induced eddy currents in a conductor that has an alternating current flowing through it.

The process of the eddy currents producing magnetic fields is described by Ampère's circuital law

$$\nabla \times \vec{B} = \mu_0 \left( \vec{J} + \varepsilon_0 \frac{\partial \vec{E}}{\partial t} \right) \tag{2.13}$$

where $\mu_0$ is the permeability of vacuum, $\varepsilon_0$ is the permittivity of vacuum and $\vec{J}$ is the current density [6, 7].

The induced magnetic fields due to the eddy currents in the conductor will in turn create an electric field due to Faraday's law as described by (2.1). This electric field will oppose any changes in the current intensity and force the electrons flowing in the conductor towards the outside of the conductor. This is what is called the skin effect. The consequence of the electrons being forced towards the outside of the conductor is that the tendency will be that the electrons only flow through a small part of the available area of the conductor, thus increasing the resistance. When the resistance is increased there will also be higher resistive losses [11]. Figure 2.7 shows the skin effect and the consequence it has on the current density in different parts of the conductor.

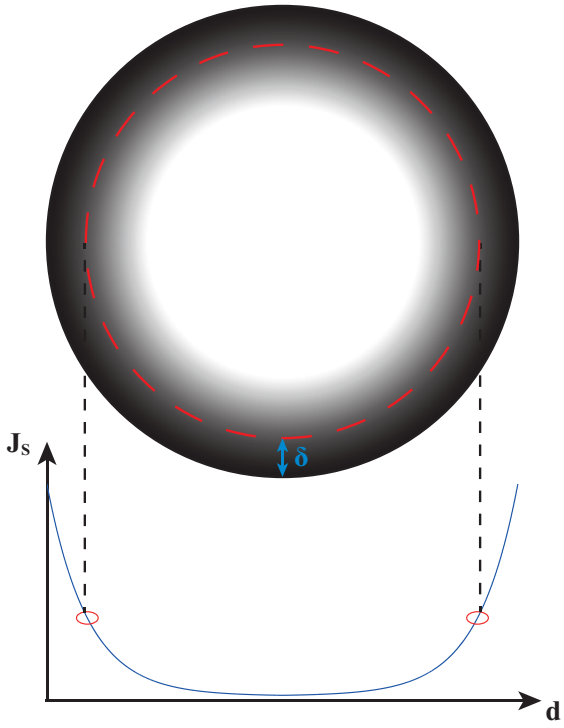



Figure 2.7: The skin effect and its consequences.

The skin depth $\delta$ is marked in Figure 2.7 which is the depth at which the current density has gone down by a factor of $e^{-1}$ which is approximately equal to a decrease of 63% in current density. The skin depth can be approximated by

$$\delta = \sqrt{\frac{2\rho}{\omega\mu_r\mu_0}} \tag{2.14}$$

where $\rho$ is the resistivity of the conductor, $\omega$ is the angular frequency and $\mu_r$ is the

relative permeability of the conductor [11]. The effect on the current density caused by the skin effect can now be expressed in terms of the skin depth as

$$J(d) = J_S e^{-d/\delta} \tag{2.15}$$

where *J(d)* is the current density at distance $d$ from the surface of the conductor and $J_S$ is the current density at the surface of the conductor [11].

### 2.3.1 Litz Wire

As was mentioned in Section 2.3 the skin effect contributes to higher resistive losses. There are, however, several ways of lessening the skin effect. One common method is by using litz wire instead of regular conductors. Litz wire consists of many strands of wire with small radii which are galvanically insulated from each other. The effect of replacing the regular conductors with litz wire is that since the strands have small radii the skin depth is usually comparably large. Since the radii of the strands are relatively small in comparison to the skin depth (at which the current density is decreased by 63%) the skin effect does not have as much impact. Figure 2.8 shows a representation of why litz wire reduces the skin effect.

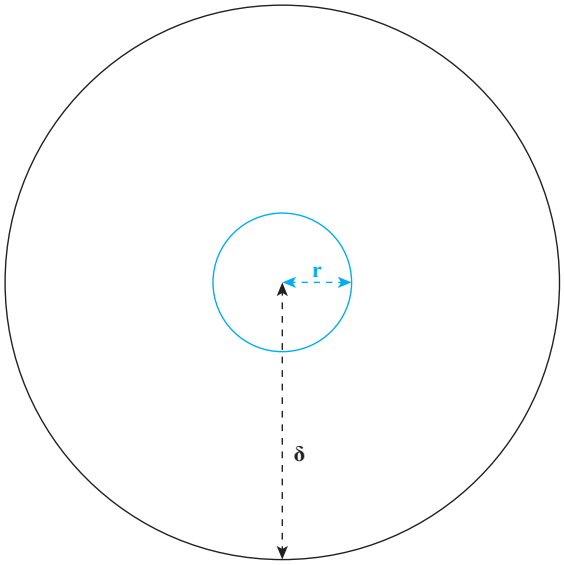



Figure 2.8: Litz wire in comparison to skin depth.

Figure 2.8 is drawn so that the ratio between the radius of the conductor and the skin depth is 50/200 which means that the skin effect only reduces the current density in the middle of the conductor by

$$\Delta J = \left(1 - e^{-50/200}\right) \cdot 100\% \approx 22.1\% \tag{2.16}$$

as given by (2.15). Compared to a regular conductor where the skin effect might easily reduce the current density by more than 63% at the centre of the conductor the reduction can be considered to be quite small.

## 2.4 Magnetism

Magnetism has two main causes. The first one is the orbital motion of electrons around the nuclei of atoms. They form small current loops which according to Ampère's law create a magnetic field and thus contribute a magnetic moment. The second source of magnetism is the spin (angular momentum) of electrons which is responsible for the majority of the magnetic moment of materials. For most materials the magnetic moment contributions are cancelled out due to electrons spinning in different directions or moving in opposite directions around the nuclei. However, there are some materials which have larger magnetic moments [7].

### 2.4.1 Varieties of Magnetism

There are different types of magnetism. The most common types are diamagnetism, paramagnetism and ferromagnetism [6, 7].

Diamagnetic materials have relative permeabilities slightly below 1 and are the kinds of materials which are slightly repelled in the presence of a magnet. These do not act as permanent magnets and only get weakly induced magnetic moments when an external magnetic field is applied. Some examples of diamagnetic materials are copper, silver and gold [6, 7].

Paramagnetic materials have relative permeabilities slightly above 1 and are the kind of materials which are slightly attracted in the presence of a magnet. These do act as permanent magnets but this effect is small and need to compete with thermal motion in order for the magnetic moments to keep aligned. These materials will also only get a weak induced magnetic moment when an external magnetic field is applied. Some examples of paramagnetic materials are aluminium, magnesium and titanium [6, 7].

Ferromagnetic materials have relative permeabilities which are far greater than 1. These are ideal for IPT purposes as the high permeabilities make it possible to direct the magnetic flux as wished. The kind of ferromagnetic materials that are particularly interesting for IPT systems are called ferrites. There are two types of ferrite materials - soft ferrite and hard ferrite. The main difference is that the magnetization of soft ferrite can change direction easily without much losses. Hard ferrites on the other hand have a high capacity to resist reversing the direction of the magnetization (coercivity). An example of uses for hard ferrite materials is as refrigerator magnets. However, for IPT purposes soft ferrites are preferred due to the high switching frequencies involved as soft ferrites have lower losses in situations which require high frequencies. The resistivity

for ferromagnetic materials is also high which limits the amount of current flowing through the material. Some examples of ferromagnetic materials are iron, cobalt and nickel [6, 7, 12, 13].

## 2.5 Power Electronics

The power electronics needed for an IPT system to function is an inverter on the primary side which converts the dc from the power supply to an ac in order to transfer power through the coils using the resonance frequency. A rectifier is then needed on the secondary side in order to convert the ac current from the coils to a dc current that can be fed to the battery.

### 2.5.1 dc/ac Conversion

Figure 2.9 illustrates a typical full-bridge inverter circuit.

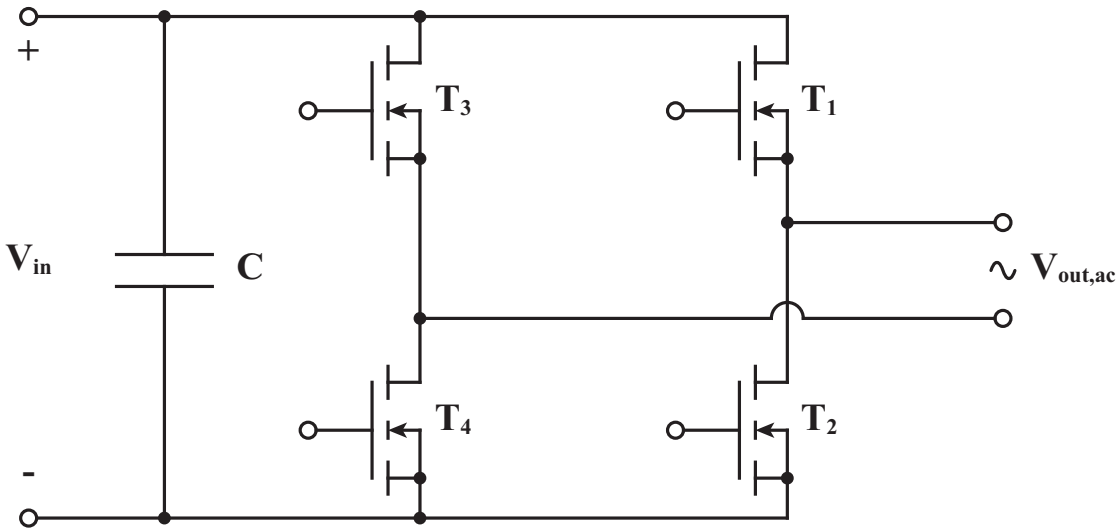



Figure 2.9: Layout of a full-bridge inverter circuit.

The full-bridge inverter would - in IPT systems - have a resonance circuit connected to the output. The passive components that make up the resonance circuit, inductances and capacitances, are what allows for zero-current switching which is preferable to minimize switching losses in the transistors. The resonance circuit can operate in the four quadrants shown in Figure 2.10.

When the resonance circuit operates in $Q_1$ the transistors $T_1$ and $T_4$ conduct. The current is moving from the positive leg of the input through $T_1$ onwards through the

resonance circuit. The current then moves back towards the inverter and through $T_4$ back to the input again through the negative leg.

The positive current through the inductor and capacitor will charge the capacitor and the voltage will eventually become greater than the input voltage. The current then starts reversing direction and go back through the positive leg of the input instead.

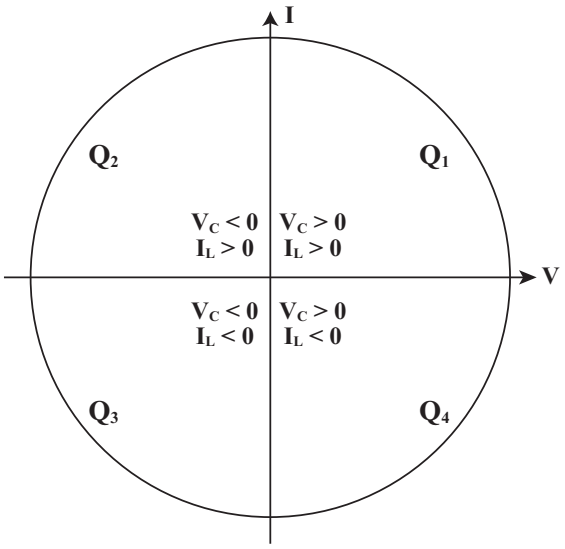



Figure 2.10: Layout of a full-bridge inverter circuit.

When this reversal of current direction occurs the transistors $T_1$ and $T_4$ are switched off and $T_2$ and $T_3$ are turned on. The inductor and capacitor in the resonance circuit now operate in $Q_4$ and the capacitor discharges through the inductor. The voltage over the capacitor now becomes negative and the resonance circuit operates in $Q_3$.

Now that the capacitor is negatively charged the voltage will eventually become greater than the input voltage again. The current starts reversing direction from negative to positive and go back through the negative leg of the input. The transistors $T_2$ and $T_3$ are turned off and $T_1$ and $T_4$ are turned on when the current switches direction. The capacitor is now discharging to the input and the resonance circuit operates in $Q_2$.

When the capacitor is discharging with a positive current moving from it the voltage will also start reversing polarity. When the voltage goes from negative to positive over the capacitor the resonance circuit switches from operating in $Q_2$ to operating in $Q_1$. This cycle will keep repeating itself until the power supply is turned off.

### 2.5.2 Active Rectification

Active rectification (aka synchronous rectification) works the same way as passive rectification. The main difference is that instead of using diodes for the rectification process

active components - such as transistors - are used. Figure 2.11 shows a typical active rectification circuit.

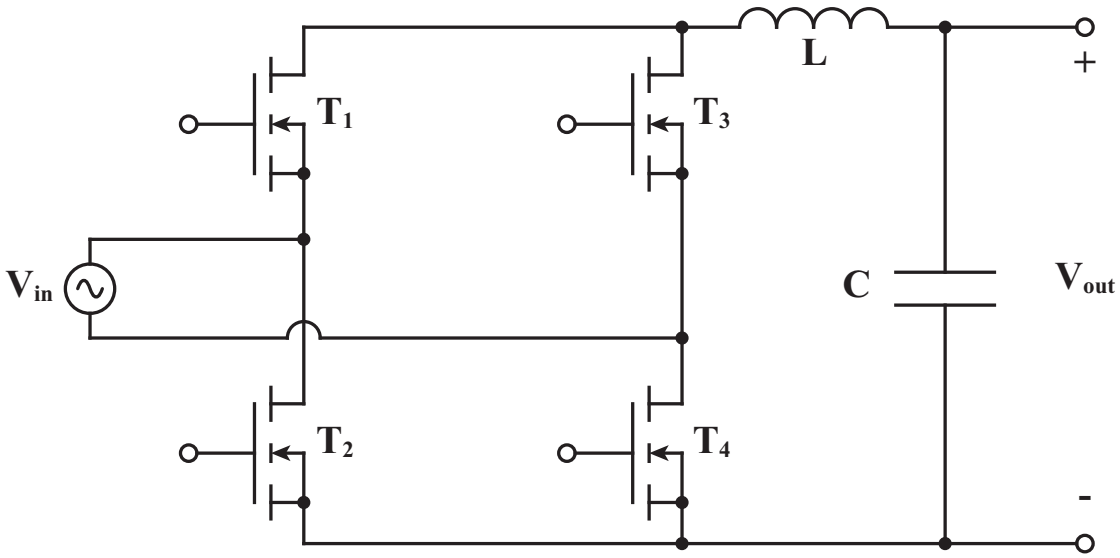



Figure 2.11: Layout of a synchronous rectifier circuit.

The active rectification circuit can - like the resonance circuit in Section 2.5.1 - typically operate in the four quadrants shown in Figure 2.12.

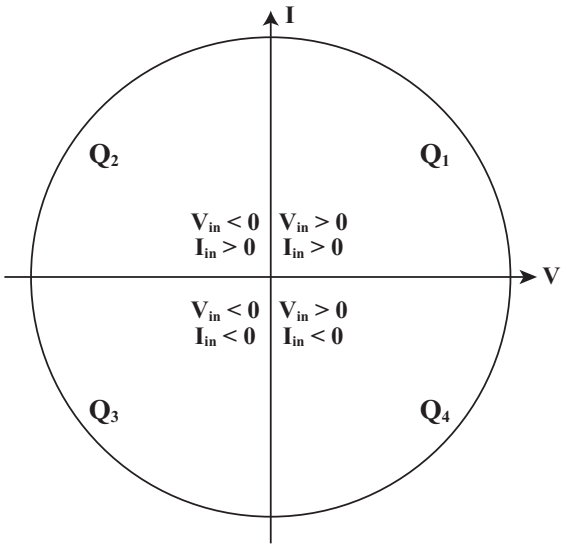



Figure 2.12: The four quadrants in which the rectifier can operate.

When the rectifier operates in quadrant 1 the transistors $T_1$ and $T_4$ conduct. The current

will then flow from the upper leg of the power supply upwards through $T_1$, through the load and then upwards through $T_4$ and back to the power supply through the lower leg.

When the voltage goes from positive to negative the inductance L will continue to force a positive current through the transistors. During this time the rectifier operates in quadrant 2. The negative voltage over the inductance will make the current approach zero at which time the transistors $T_1$ and $T_4$ are turned off and the transistors $T_2$ and $T_3$ are turned on.

When the input power operates in quadrant 3 the transistors $T_2$ and $T_3$ conduct. The current will then flow from the lower leg of the power supply, upwards through $T_3$, through the load, and then upwards through $T_2$ and back to the power supply through the upper leg.

When the voltage goes from negative to positive the inductance L will continue to force a negative current through the transistors. During this time the rectifier operates in quadrant 4. The positive voltage over the inductance will make the current approach zero at which time the transistors $T_2$ and $T_3$ are turned off and the transistors $T_1$ and $T_4$ are turned on. Thus, the rectifier operates in quadrant 1 again and the cycle repeats itself.

To control the sequence of shifting between when $T_1$ and $T_4$, and when $T_2$ and $T_3$ conduct, gate drive circuits are needed. Adding these circuits increases the complexity of the rectifier compared to a diode bridge rectifier. The reason why it is preferable to use an active rectifier instead of a passive rectifier despite the added complexity is illustrated by Figure 2.13.

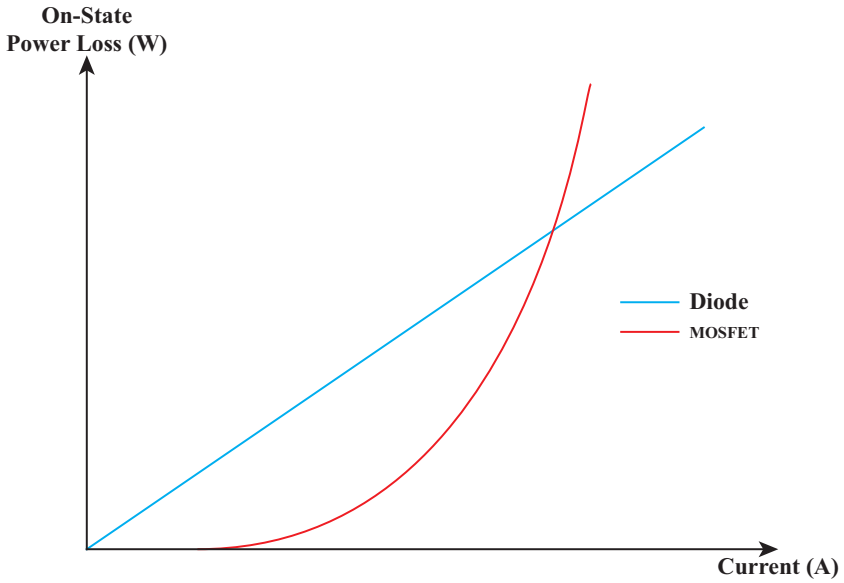



Figure 2.13: Comparison of the power consumption of a diode and a Metal Oxide Semiconductor Field Effect Transistor (MOSFET) with the same Si area.

From Figure 2.13 a conclusion can be drawn that in the area to the left of the intersection between the two lines it is preferable to use a MOSFET and in the area to the right of the intersection it is preferable to use a diode. The reason behind this is that the diode has a voltage drop with a large offset and a steeper current-voltage slope (lower resistance) so the power consumption will scale approximately linearly with current as the variable. The MOSFET however is a non-linear component that has both a voltage drop (though smaller than the diode's) and an on-state resistance ($R_{ds,on}$). According to Ohm's law the on-state resistance will create further voltage drops based on the current that flows through it. This means that there is a limit where the current is high enough to make it less efficient than the diode. However, below this threshold current the MOSFET will have lower losses than the diode. The losses over the diode can be calculated as

$$P_{loss,diode} = V_{drop} I \tag{2.17}$$

while the losses for the MOSFET can be calculated as

$$P_{loss,\text{MOSFET}} = \left(V_{ds,on} + R_{ds,on} I_d\right) I_d \tag{2.18}$$

where $V_{ds,on}$ is the forward voltage drop over the transistor and $I_d$ is the current flowing through the transistor.

# Chapter 3

# Simulations

The simulations have been performed in MATLAB/Simulink. This provides a basis of comparison for the real-world results acquired at the end of the master's thesis so as to be able to evaluate the results.

## 3.1 Simulation of Coils

The coils were modelled in MATLAB/Simulink by using the S-function block which allows the user to define the function it uses. The coils in an IPT system principally works similar to a transformer. The coils were modelled as the equivalent circuit of a transformer but without the iron loss resistance and with an added compensation capacitor as shown in Figure 3.1. The input signal to the S-function block is generated from several blocks in MATLAB/Simulink which creates a square wave signal with blanking time.

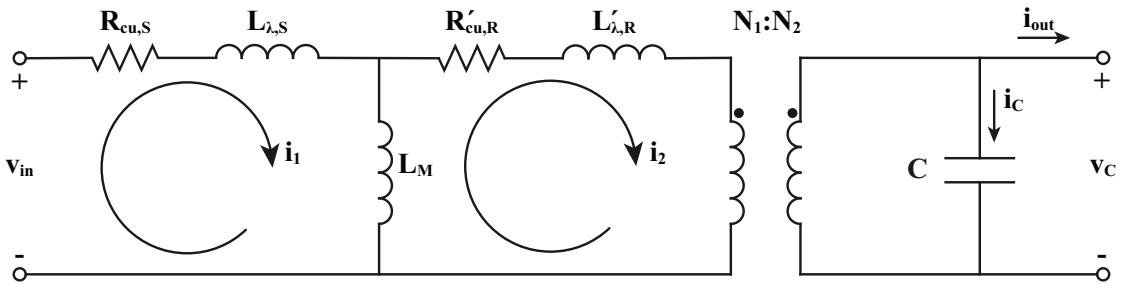



Figure 3.1: Equivalent circuit for the inductive charging station.

The components in Figure 3.1 are $L_{\lambda,S}$ and $L_{\lambda,R}$ which are the leakage inductances on the sending and receiving side, and $R_{cu,S}$ and $R_{cu,S}$ which are the copper resistances on the sending and receiving side. Also, $L_M$ which is the magnetizing inductance and C which is the compensation capacitor. $N_1$ and $N_2$ in the figure denotes the number of turns on each side. Kirchoff's Voltage Law (KVL) can be used to describe the currents

and voltages in the circuit from these components and the number of turns. Loop 1 which is shown by current $i_1$ in the figure yields the equation

$$0 = v_{in} - R_{cu,S} i_1 - L_{\lambda,S} \frac{\mathrm{d}i_1}{\mathrm{d}t} - L_M \frac{\mathrm{d}(i_1 - i_2)}{\mathrm{d}t} \tag{3.1}$$

and loop 2 which is shown by current $i_2$ yields the equation

$$0 = L_M \frac{\mathrm{d}(i_1 - i_2)}{\mathrm{d}t} - \left(\frac{N_1}{N_2}\right)^2 R_{cu,R} i_2 - \left(\frac{N_1}{N_2}\right)^2 L_{\lambda R} \frac{\mathrm{d}i_2}{\mathrm{d}t} - \frac{N_1}{N_2} v_c \tag{3.2}$$

The ideal transformer's input and output current are the same except for the scaling according to the turns ratio of the transformer. Therefore, the capacitor current can be expressed as

$$i_c = \frac{N_1}{N_2} i_2 - i_{out} = C \frac{\mathrm{d}v_c}{\mathrm{d}t} \tag{3.3}$$

with which the derivative of the voltage can also be described. With these equations the relations between the currents and voltages can be described by state space matrices. Expressing 3.1, 3.2 and 3.3 in matrix form gives

$$\underbrace{\begin{bmatrix} v_{in} \\ 0 \\ i_{out} \end{bmatrix}}_{\mathrm{u}} = \underbrace{\begin{bmatrix} L_{\lambda S} + L_M & -L_M & 0 \\ L_M & -L_M - \left(\frac{N_1}{N_2}\right)^2 L_{\lambda R} & 0 \\ 0 & 0 & -C \end{bmatrix}}_{\mathrm{L}} \underbrace{\begin{bmatrix} \frac{\mathrm{d}i_1}{\mathrm{d}t} \\ \frac{\mathrm{d}i_2}{\mathrm{d}t} \\ \frac{\mathrm{d}v_C}{\mathrm{d}t} \end{bmatrix}}_{\dot{\mathrm{x}}} + \underbrace{\begin{bmatrix} R_{cu,S} & 0 & 0 \\ 0 & -\left(\frac{N_1}{N_2}\right)^2 R_{cu,R} & -\frac{N_1}{N_2} \\ 0 & \frac{N_1}{N_2} & 0 \end{bmatrix}}_{\mathrm{R}} \underbrace{\begin{bmatrix} i_1 \\ i_2 \\ v_C \end{bmatrix}}_{\mathrm{x}} \tag{3.4}$$

which can be rearranged as

$$\dot{x} = L^{-1} u - L^{-1} R x \tag{3.5}$$

This expression can be used in the S-function in MATLAB/Simulink to simulate the dynamics of the coils and compensation capacitor.

## 3.2 Simulation of Rectifier

The active rectifier and the smoothing circuit together with the resistive load were modelled as shown in Figure 3.2.

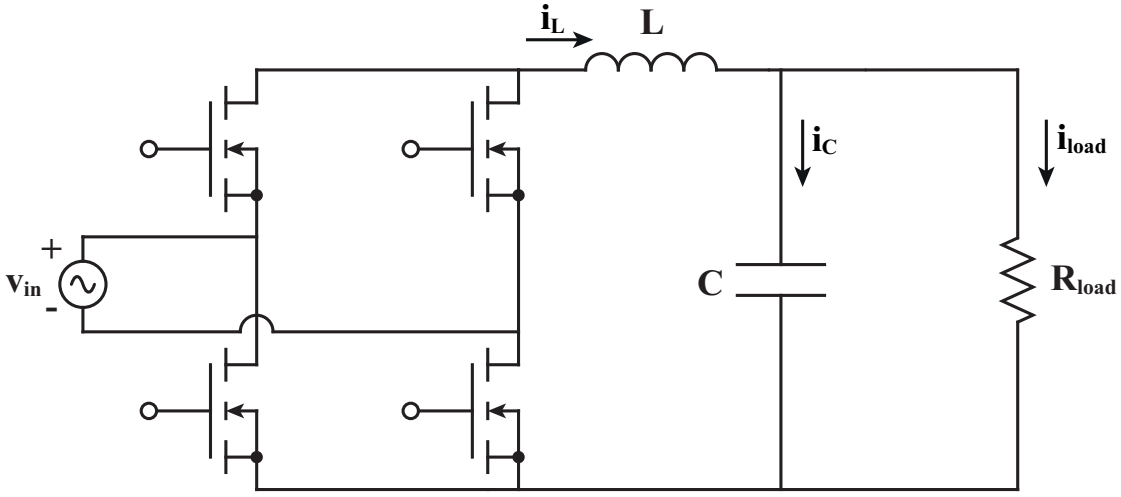



Figure 3.2: Equivalent circuit for the rectifier and battery.

The current through the inductor relates to the voltage over the inductor as

$$\frac{\mathrm{d}i_L}{\mathrm{d}t} = \frac{v_{rect} - v_C}{L} \tag{3.6}$$

where $v_{rect}$ is the voltage from the rectifier bridge and $v_C$ is the capacitor voltage. The voltage over the capacitor is calculated as

$$\frac{\mathrm{d}v_c}{\mathrm{d}t} = \frac{i_L - i_{load}}{C} \tag{3.7}$$

where $i_L$ is the current through the inductor and $i_{load}$ is the current through the resistive load. The current through the resistive load follows

$$i_{load} = \frac{v_C}{R_{load}} \tag{3.8}$$

where $R_{load}$ is the resistance of the power resistor used as load. The input voltage to the rectifier bridge is the capacitor voltage from the equivalent circuit shown in Figure 3.1. The MOSFETs are modelled as resistances that switch between $R_{on}$ and $R_{off}$. The different pair of MOSFETs turn on when $v_{in} \geq v_c$ or $v_{in} \leq -v_c$ and turns off when $i_L = 0$.

## 3.3 Results

The measured parameters from the coils were given as input to the MATLAB/Simulink simulation. The first comparison is for an air gap of 5cm. The input voltage was 63V from the dc supply and the load on the receiving side was 6.9 Ω. The capacitance of the compensation capacitor had to be adjusted to 2.6 $\mu$F compared with the 2 $\mu$F used in the lab environment. The high frequency oscillations that can be seen at the zero crossings are probably caused by the reverse recovery charge in the transistors' body diodes. This is a phenomenon that occurs in real life but it has not been taken into account in the simulations. Therefore, the oscillations are present in the measurements but not in the simulations. The measurements from the case of a 5 cm air gap is shown in Figure 3.3 and the results from the simulation in Figure 3.4.

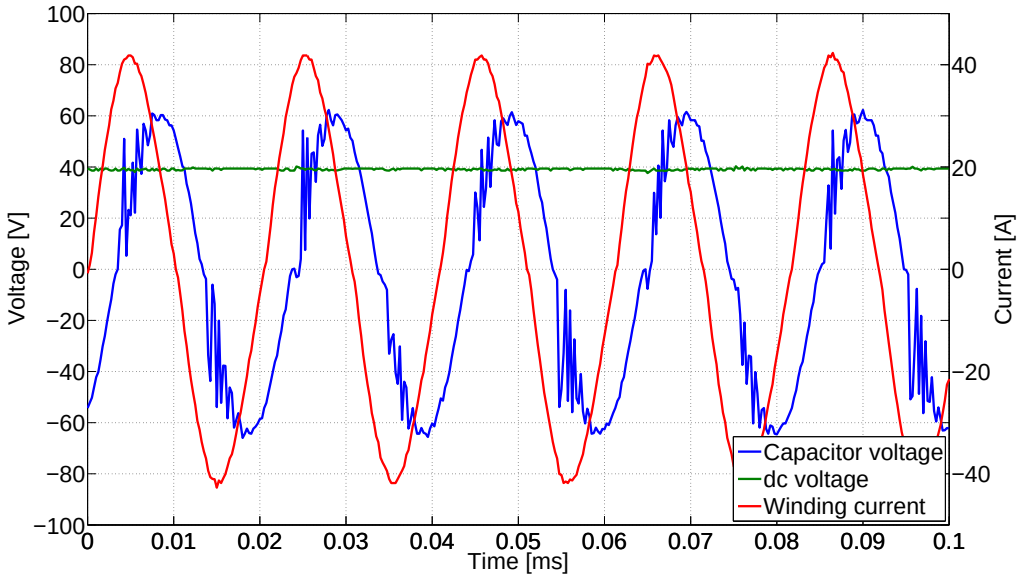



Figure 3.3: Measured voltage over the compensation capacitor, current in the secondary winding and the dc voltage over the load at an air gap of 5cm.

The dc power supply was set to 29V when measurements were taken for a 2 cm air gap. The load used was 6.9 Ω. The capacitance had to be adjusted from 2 $\mu$F to 3.45 $\mu$F to achieve the same results in the simulation as the measurements.

The dc power supply was set to 40V when measurements were taken for a 10 cm air gap. The load used was 6.9 Ω. The capacitor was adjusted to 2.225 $\mu$F to achieve the highest possible output voltage. The output voltage in the simulation did not reach the same output voltage as in the measurement.

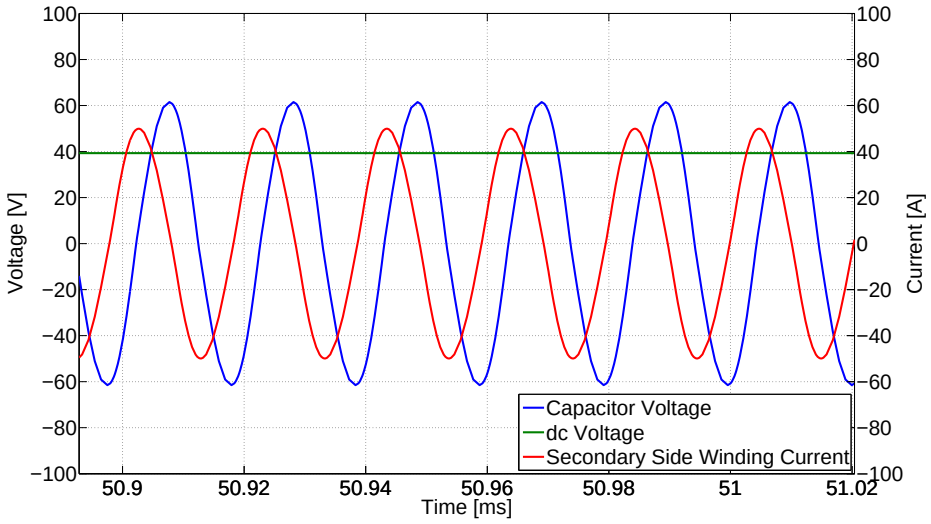



Figure 3.4: Simulated voltage over the compensation capacitor, current in the secondary winding and the dc voltage over the load at an air gap of 5cm.

The measurements from the 2 cm air gap are shown in Figure 3.5.

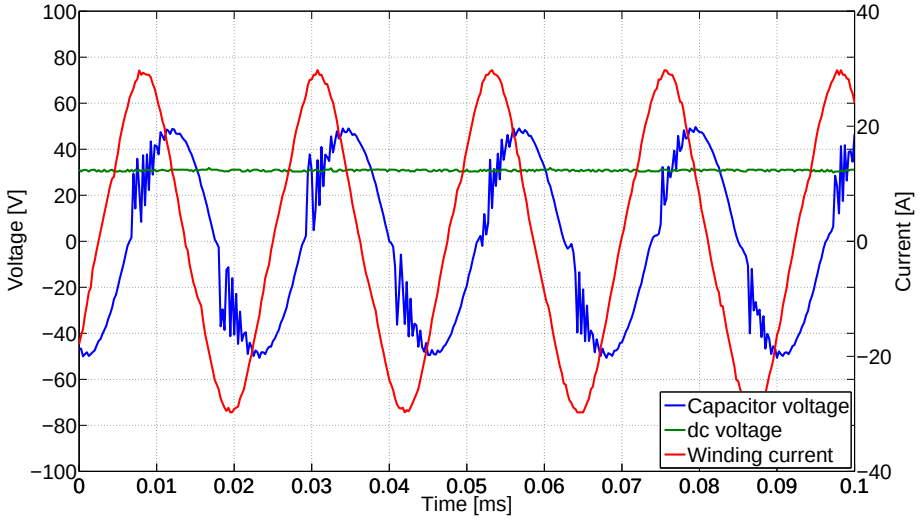



Figure 3.5: Measured voltage over the compensation capacitor, current in the secondary winding and the dc voltage over the load at an air gap of 2cm.

The results from the simulations can be seen in Figure 3.6.

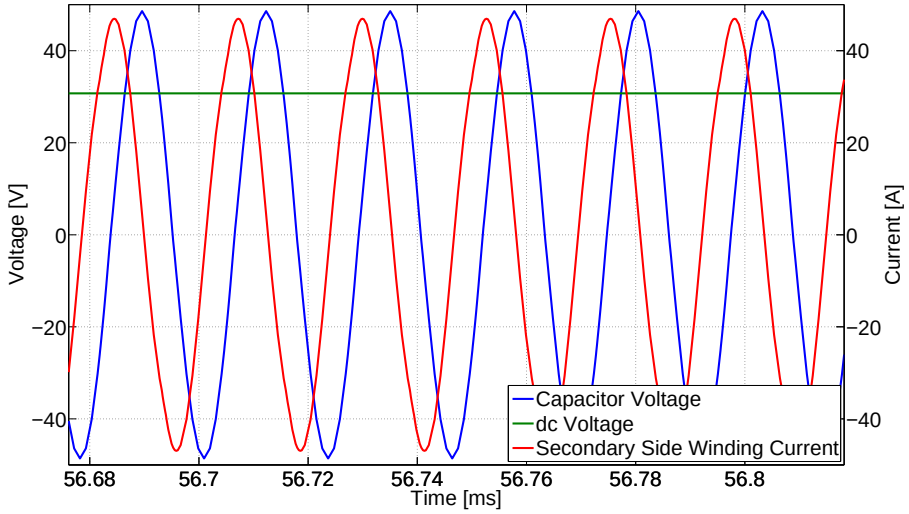



Figure 3.6: Simulated voltage over the compensation capacitor, current in the secondary winding and the dc voltage over the load at an air gap of 2cm.

The measurements performed with a 10 cm air gap is shown in Figure 3.7.

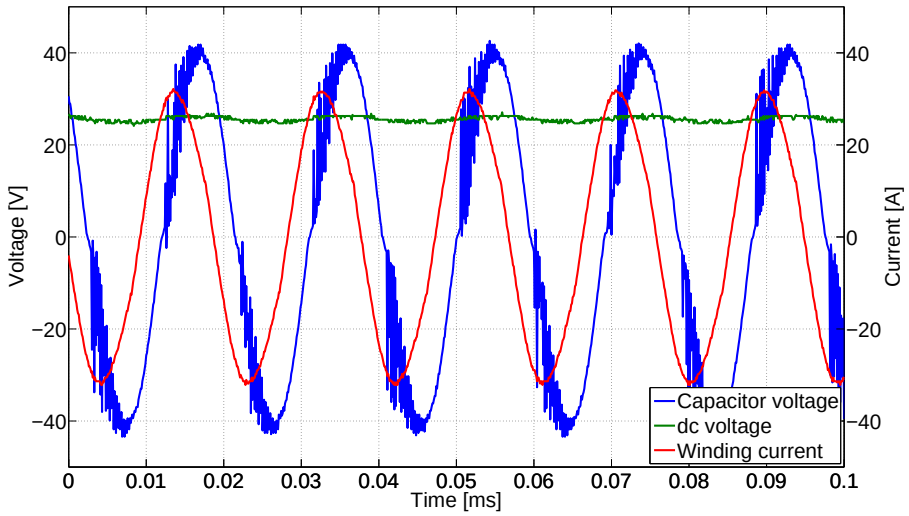



Figure 3.7: Measured voltage over the compensation capacitor, current in the secondary winding and the dc voltage over the load at an air gap of 10cm.

The simulations of the 10 cm air gap power transfer can be seen in Figure 3.8.

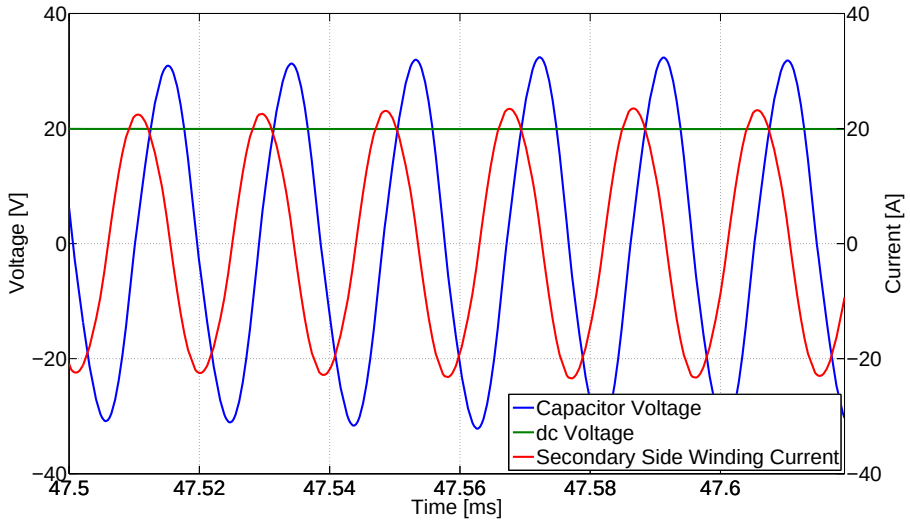



Figure 3.8: Simulated voltage over the compensation capacitor, current in the secondary winding and the dc voltage over the load at an air gap of 10cm.

## 3.4 Simulation Errors

There may be multiple reasons that the simulations deviate slightly from the measurements. First, the components in the simulation circuit have been simplified and it is plausible that the measured parameters contain measurement errors. Secondly, the iron losses in the magnetic core have been neglected in the simulations. Thirdly, all circuit elements are considered to be ideal with no parasitic elements anywhere. Fourthly, the transformer model has also been simplified. For instance, there are no stray capacitances between the primary and secondary side. No stray capacitances due to the surroundings has been taken into account either. Lastly, the numerical solver will introduce numerical errors in the simulation as the step size cannot be set to an infinitesimal size. Setting a very fine step size causes the calculations to become very large and the simulations to take a very long time. Because of these constraints there will always be some numerical errors in the results from the simulations.

# Chapter 4

# System Design

A complete inductive charging system has been constructed for an electric go kart. This chapter covers the construction of the separate parts as well as the complete charging system and how it works.

## 4.1 Coils and Resonance Circuit

The coil design that was chosen is based on the design used in a conference paper from the University of Auckland and in a bachelor's thesis from Chalmers University of Technology [14, 15]. The design is illustrated in Figure 4.1 where the coils are wound around the narrow middle part. There are two parallel coils wound on each side.

The reason why this design was chosen is that it is smaller than most other commonly used designs while still maintaining a high efficiency. The working principle is as follows. The magnetic flux will flow from one end of the ferrites on the primary side to the corresponding end of the ferrites on the secondary side. The flux will then travel through the narrow path of ferrites to the other end of the ferrites on the secondary side. The magnetic flux will then proceed to travel through the air to the other end on the primary side back through the narrow path of ferrites on the primary side and back to the first end where it started. Since the air gap will be small in this case, the majority of the path the magnetic flux takes is therefore through the ferrite rather than the air; this is preferable since ferrites have a much higher permeability than air. Therefore, the magnetic reluctance for the path that the magnetic flux travels through is reduced. This is because the correlation between reluctance and permeability is

$$\mathscr{R} = \frac{l}{\mu_0 \mu_r A} \tag{4.1}$$

The ensuing reduction of magnetic reluctance will in turn increase the magnetic flux

according to

$$\Phi = \frac{MMF}{\mathscr{R}} \tag{4.2}$$

where $MMF$ is the magnetomotive force.

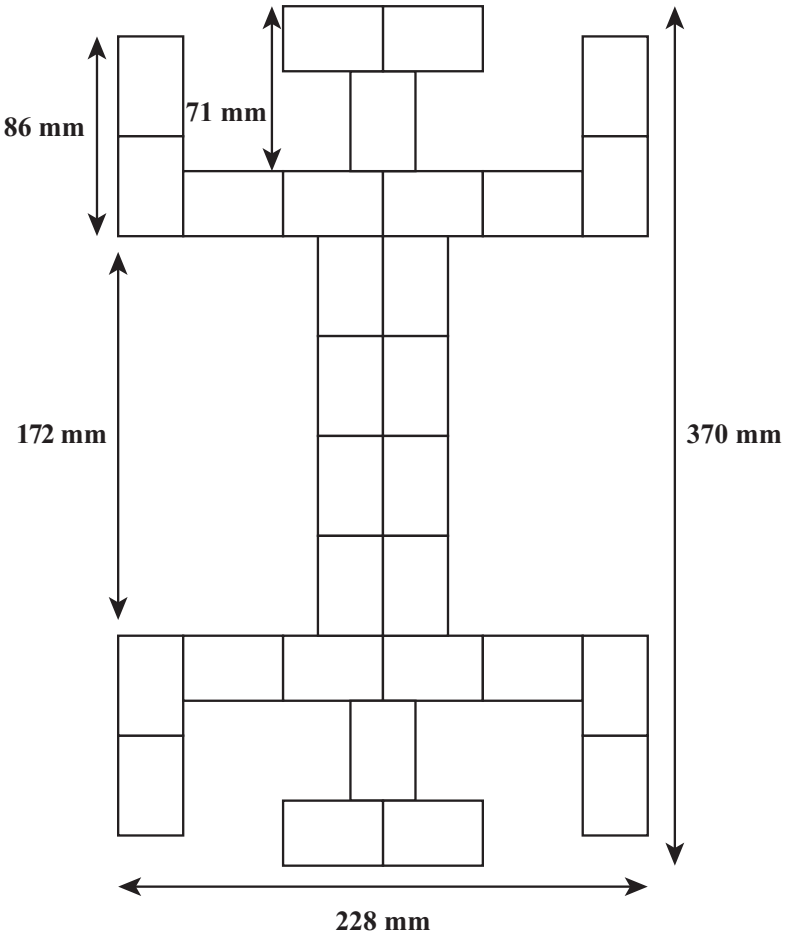



Figure 4.1: The layout of the ferrites for the coils.

The actual coils are made out of litz wire. It was uncertain how high the current and voltage over the coils would be due to the resonance in the system. Therefore, a wide margin was set from the 9A the battery should be charged with, up to 100A RMS. Furthermore, it was assumed that the litz wire could withstand 4A/mm$^2$ without getting too hot. The area of litz wire needed is therefore

$$A = \frac{100\text{A}}{4\text{A/mm}^2} = 25\text{mm}^2 \tag{4.3}$$

The area of the litz wire is $0.94\text{mm}^2$. To obtain enough area to achieve sufficient cooling several wires were connected in parallel. The number of parallel wires needed can be calculated as

$$n = \frac{25\text{mm}^2}{0.94\text{mm}^2/\text{wire}} = 26.59... \approx 27\text{wires} \tag{4.4}$$

The height where the litz wire is wound has a 3 mm plastic plate and three ferrite plates, each with a height of 4.1 mm. The total height where the litz wire is wound is therefore

$$h = 3 + 4.1 \cdot 3 = 15.3\text{mm} \tag{4.5}$$

The width where the litz wire is wound is set by the width of the ferrite plates. There are two ferrite plates breadthways with a width of 28 mm each. The width where the litz wire is wound is therefore

$$w = 2 \cdot 28 = 56\text{mm} \tag{4.6}$$

The length needed to be able to accommodate four turns on both the primary and secondary side is calculated as

$$L_{4turns} = \frac{1}{0.6}(2 \cdot w + 2 \cdot h) \cdot 4turns \cdot 27wires \tag{4.7}$$

where 0.6 is the margin chosen as it is impossible to pack the litz wire ideally; there will always be pockets of air. The result from the calculation is

$$L_{4turns} = \frac{1}{0.6}(2 \cdot 56 + 2 \cdot 15.3) \cdot 4turns \cdot 27wires = 25668\text{mm} = 25.668\text{m} \tag{4.8}$$

This is the length needed for one winding. However, there are two parallel windings on each side in the design and both a sending and receiving side. The total length of litz wire needed is therefore

$$L_{tot} = L_{4turns} \cdot 2windings \cdot 2sides = 102.672\text{m} \tag{4.9}$$

### 4.1.1 Resonance Circuit

The capacitance values for the resonance circuit was chosen in order to achieve resonance at a frequency of 49 kHz optimally placed with an airgap of 5 cm. The resonance circuit, seen in Figure 4.2 and Figure 4.3, was built using 18 $1\mu$F polypropylene film capacitors. Polypropylene film capacitors were chosen because it allows a high rate of change of the voltage. This was necessary since the resonance frequency needed to be on the order of tens of kilohertz. The resonance circuit was designed to be able to

withstand the highest amount of voltage and current that could be achieved with the available capacitors.

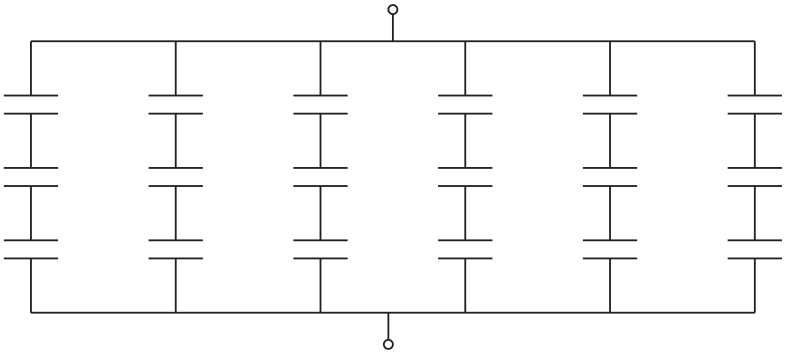


Figure 4.2: The configuration of the capacitors in the capacitor bank.

This resonance frequency was chosen because it achieved a higher efficiency than lower frequencies and any further increase in frequency did not yield a higher efficiency.

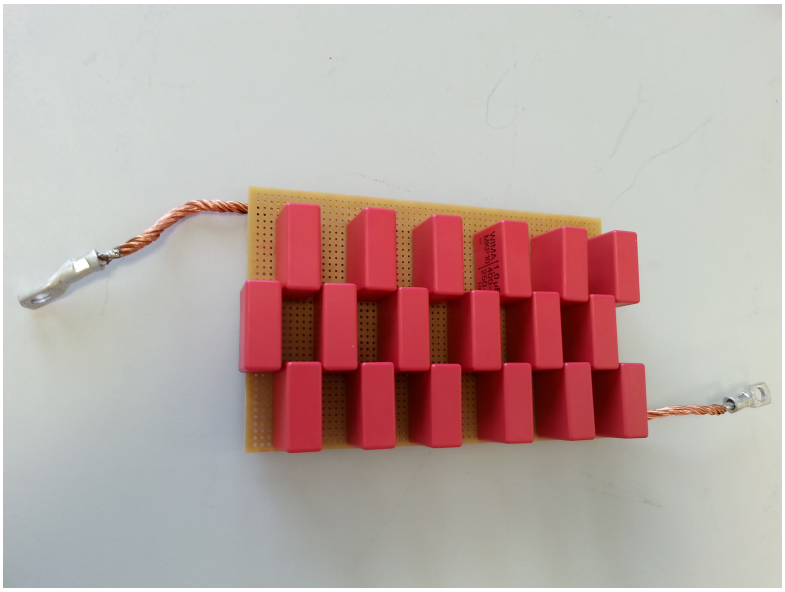


Figure 4.3: The configuration of the capacitors in the capacitor bank.

## 4.2 dc/dc Converters

Two different types of voltage regulators have been used for the circuit boards - linear regulators and switched regulators. Some components needed a very stable 1.9V and

3.3V supply. Linear regulators were chosen for these applications since they deliver a very precise voltage. However, the components which are not voltage ripple sensitive and which require higher currents are supplied by switched regulators. This is to ensure a higher output current and lower losses. The various regulators were chosen based on input and output voltage, output current, and efficiency. The passive components connected to the regulators have been chosen based on the datasheets supplied by the manufacturers.

### 4.2.1 Switched dc/dc Converter (15V)

The regulator LM5010 was used to convert the high input voltage (20V-68V) to 15V. The 15V will be fed to the gate drivers and the 1.9V and 3.3V regulators. The circuit diagram for the LM5010 can be seen in Figure 4.4.

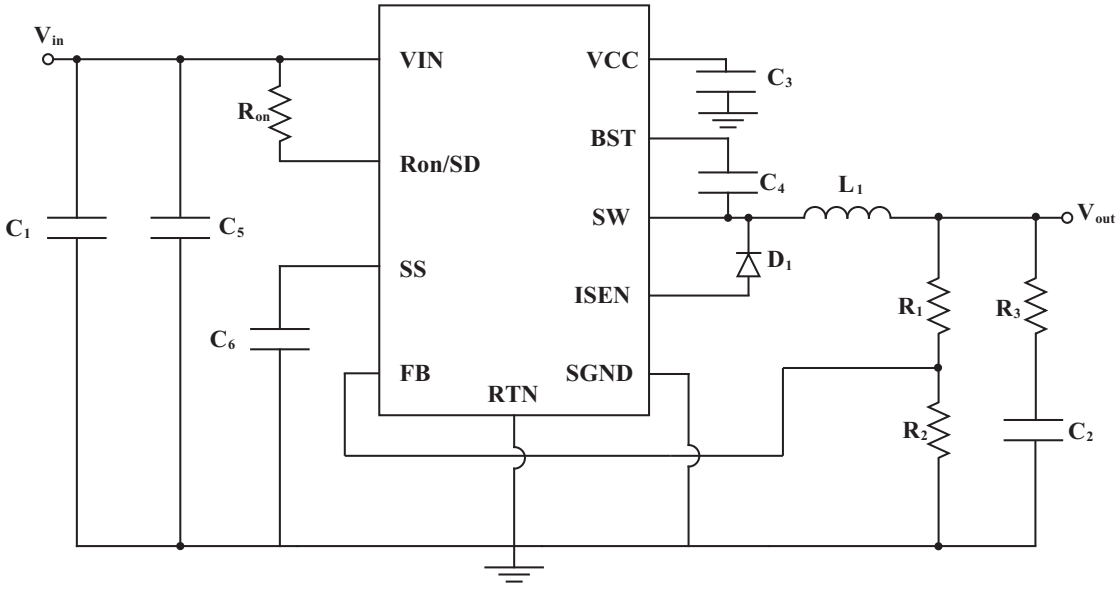



Figure 4.4: The LM5010 with its surrounding circitry.

The feedback loop for the regulator consists of two resistors. The ratio between resistor $R_1$ and $R_2$ sets the output voltage and was calculated as

$$\frac{R_1}{R_2} = \frac{V_{out}}{2.5} - 1 = \frac{15}{2.5} - 1 = 5 \tag{4.10}$$

and since $R_2$ should be set to 2kΩ - as specified by the datasheet - $R_1$ was chosen to 10kΩ. The value of the output inductor was based on

$$L = \frac{V_{out}(V_{in(max)} - V_{out})}{I_{OR} F_{S(min)} V_{in(max)}} = \frac{15\text{V}(68\text{V} - 15\text{V})}{0.3\text{A} \cdot 476.7\text{kHz} \cdot 68\text{V}} = 81.75\mu\text{H} \tag{4.11}$$

where $V_{out}$ is the output voltage, $V_{in(max)}$ is the highest input voltage that might be used, $F_{S(min)}$ is the lowest switching frequency that the device might use, and $I_{OR}$ is the largest allowed ripple current at minimum load. Based on (4.11) the inductor chosen had an inductance of 100$\mu$H. The inductor can conduct a continuous current of 2.5A and saturates at 3.1A. The regulator, LM5010, has a minimum allowed current of 150mA and a minimum output voltage ripple of 25mV. Otherwise it will not function as intended. To ensure correct functionality an extra Equivalent Series Resistance (ESR) is added to the output capacitor. The ESR needed to stay above the minumum ripple is

$$ESR_{min} = \frac{25\text{mV}(R_1 + R_2)}{R_2\, I_{OR(min)}} = \frac{25\text{mV}(10\text{k}\Omega + 2\text{k}\Omega)}{2\text{k}\Omega \cdot 39.3\text{mA}} = 3.82\Omega = R_3 \tag{4.12}$$

where $I_{OR(min)}$ is the minimum ripple current. Based on this, $R_3$ was set to 5.1Ω. With $R_3$ set, $I_{OR(min)}$ was calculated as

$$\begin{aligned} I_{OR(min)} &= \frac{V_{out}(V_{in(min)} - V_{(out)})}{L_{max}\, F_{S(max)} V_{in(min)}} = \\ &= \frac{15\text{V}(20\text{V} - 15\text{V})}{1.2 \cdot 100\mu\text{H} \cdot (635.6\text{kHz} + 0.25 \cdot 635.6\text{kHz}) \cdot 20\text{V}} = 39.3\text{mA} \end{aligned} \tag{4.13}$$

where $L_{max}$ is the maximum inductor value (with an assumed ±20% deviation), $F_{S(max)}$ is the maximum operating switching frequency of the device. The datasheet specifies that the output capacitor, $C_2$, should be at least 3.3$\mu$F and based on this the capacitor value chosen was 6.8uF. The input capacitor $C_1$ was then calculated as

$$C_1 = \frac{I_O \cdot t_{on}}{\Delta V} = \frac{1\text{A} \cdot 1.664\mu\text{s}}{1\text{V}} = 1.664\mu\text{F} \tag{4.14}$$

where $I_O$ is the output current, $\Delta V$ is the variation in input current and $t_{on}$ is

$$\begin{aligned} t_{on,max} &= \frac{1.18 \cdot 10^{-10} \cdot (R_{ON} + 1.4\text{k}\Omega) \cdot 1.25}{V_{in} - 1.4\text{V}} + 67\text{ns} = \\ &= \frac{1.18 \cdot 10^{-10} \cdot (200\text{k}\Omega + 1.4\text{k}\Omega) \cdot 1.25}{20\text{V} - 1.4\text{V}} + 67\text{ns} = 1.664\mu\text{s} \end{aligned} \tag{4.15}$$

where $R_{on}$ is a resistance connected to the EN pin on the regulator and $V_{in}$ is the input voltage. The other constants are given in the datasheet of the device. $C_5$ should dampen transients, have low ESR and be placed close to the device. Based on these criteria a 100nF capacitor was chosen as $C_5$. $C_6$ determines the start-up time for the regulator. It was set to 46nF which gives a start-up time of 10ms. The other component values were chosen as specified in the datasheet.

### 4.2.2 Switched dc/dc Converter (3.3V and 1.9V)

Two separate TPS54233 regulators were used to convert 15V down to 1.9V and 3.3V. The circuit diagrams for these is shown in Figure 4.5. The converter can deliver 2A with an output voltage down to 0.8V. Two decoupling capacitors were used on the input. One 10$\mu$F capacitor to reduce voltage ripple due to load transients and a 10nF capacitor to filter high frequencies. The voltage division connected to the EN pin sets the start and stop voltages for the regulator. The RC circuit connected to the COMP pin is the compensation circuit. The TPS54233 allows the designer to configure this filter in different ways if it is needed in the design. The standard values (Given in datasheet) for the RC circuit was used. The capacitor connected to the SS pin sets the soft start time for the regulator.

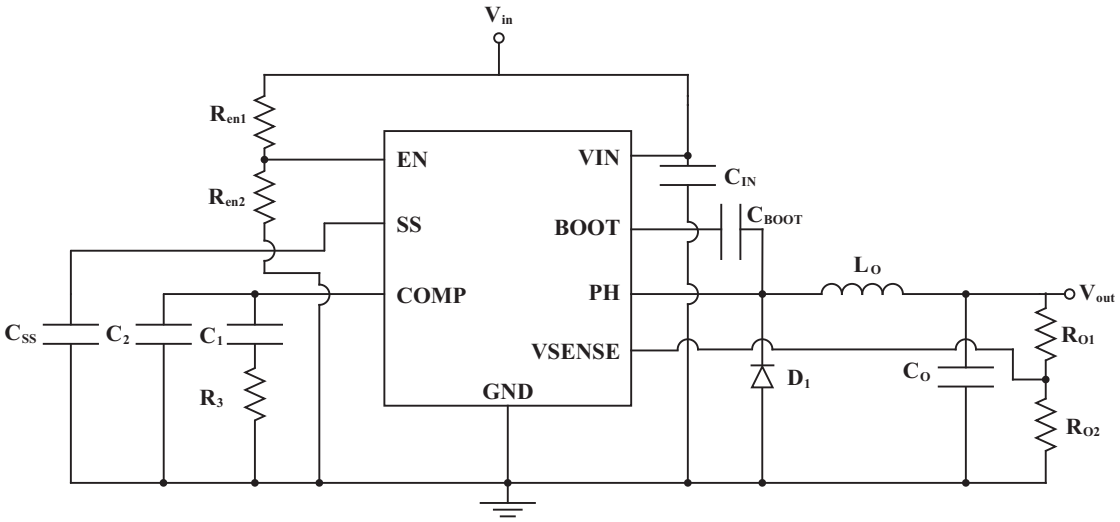



Figure 4.5: The TPS54233 together with all the external components.

The minimum output inductors was calculated according to

$$L_{min,1.9\text{V}} = \frac{V_{out(max)}(V_{in(max)} - V_{out})}{V_{in(max)} K_{ind} I_{out} F_{sw}} = \frac{1.9\text{V}(15\text{V} - 1.9\text{V})}{15\text{V} \cdot 0.3 \cdot 1\text{A} \cdot 300\text{kHz}} = 18.44\mu\text{H} \quad (4.16)$$

$$L_{min,3.3\text{V}} = \frac{V_{out(max)}(V_{in(max)} - V_{out})}{V_{in(max)} K_{ind} I_{out} F_{sw}} = \frac{3.3\text{V}(15\text{V} - 3.3\text{V})}{15\text{V} \cdot 0.3 \cdot 1\text{A} \cdot 300\text{kHz}} = 28.6\mu\text{H} \quad (4.17)$$

where $V_{out(max)}$ is the maximum output voltage, $V_{in(max)}$ is the maximum input voltage, and $I_{out}$ is the output current. $K_{ind}$ represents the relation between the ripple current that flows in the inductor relative the ripple current in the capacitor, and $F_{sw}$ is the switching

frequency of the TPS54233. A 22$\mu$H inductor able to conduct a continuous current of 4.1A and which saturates at 5A was chosen for the 1.9V regulator. A 33$\mu$H inductor able to conduct 1.7A continuous and which saturates at 1.9A was chosen for the 3.3V regulator. The minimum output capacitor was calculated as

$$C_{O,min,1.9\text{V}} = \frac{1}{2\pi R_O F_{CO,max}} = \frac{1}{2\pi \frac{1.9\text{V}}{1\text{A}} 25\text{kHz}} = 3.35\mu\text{F} \tag{4.18}$$

$$C_{O,min,3.3\text{V}} = \frac{1}{2\pi R_O F_{CO,max}} = \frac{1}{2\pi \frac{3.3\text{V}}{1\text{A}} 25\text{kHz}} = 1.93\mu\text{F} \tag{4.19}$$

where $R_O$ is the equivalent load and $F_{CO,max}$ is the practical crossover frequency of the filter. Two parallel 33$\mu$F capacitors were used in the output filter in both cases. In the feedback voltage division, $R_5$ was set to 10.3k$\Omega$ and $R_6$ was calculated according to

$$R_{6,1.9\text{V}} = \frac{R_5 V_{ref}}{V_{out} - V_{ref}} = \frac{10.3\text{k}\Omega \cdot 0.8\text{V}}{1.9\text{V} - 0.8\text{V}} = 7.491\text{k}\Omega \tag{4.20}$$

$$R_{6,3.3\text{V}} = \frac{R_5 V_{ref}}{V_{out} - V_{ref}} = \frac{10.3\text{k}\Omega \cdot 0.8\text{V}}{3.3\text{V} - 0.8\text{V}} = 3.296\text{k}\Omega \tag{4.21}$$

where $V_{ref}$ is given in the datasheet and $V_{out}$ is the desired output voltage. A resistance of 7.5k$\Omega$ was chosen as $R_6$ for the 1.9V regulator and a resistance of 3.3k$\Omega$ was chosen as $R_6$ for the 3.3V regulator.

### 4.2.3 Linear dc/dc converter (3.3V)

The linear converter TDA3663 was suitable to use for converting 15V to a low ripple 3.3V. The main reason was that it is able to conduct a current of 100mA, which is enough for the load it will drive. The circuit diagram for the regulator is shown in Figure 4.6. A decoupling capacitor of 1$\mu$F was placed at the input to the regulator. A capacitor of 33$\mu$F is placed at the output of the converter together with an extra ESR of 5.1$\Omega$ as the datasheet specifies that extra ESR is needed.

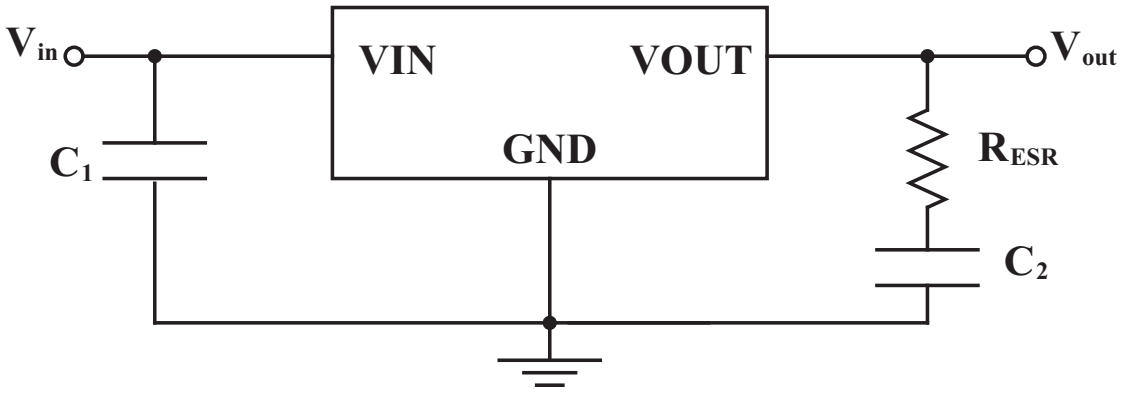



Figure 4.6: The linear regulator TDA3663 with the input and output capacitors.

### 4.2.4 Linear dc/dc converter (1.9V)

The linear regulator TPS76901 was used for the low ripple 1.9V rail as it can handle the input voltage and the output current requirements well. A decoupling capacitor of $1\mu$F was placed at the input. To ensure stable operation of the regulator an output capacitor of $4.7\mu$F and a resistance of $1\Omega$ was placed at the output. The circuitry for the regulator is shown in Figure 4.7. According to the datasheet $R_2$ should be set to 169k$\Omega$. $R_1$ was calculated as

$$R_1 = \left(\frac{V_O}{V_{ref}} - 1\right) R_2 = \left(\frac{1.9\text{V}}{1.224\text{V}} - 1\right) 169\text{k}\Omega = 93.36\text{k}\Omega \tag{4.22}$$

where $V_{ref}$ is the reference voltage and $V_O$ is the desired output voltage. As only certain resistor values are available to purchase a combination of resistors is used to obtain a value close to the one calculated. The resulting output voltage with the chosen values is

$$V_O = V_{ref}\left(1 + \frac{R_1}{R_2}\right) = 1.224\text{V}\left(1 + \frac{90.9\text{k}\Omega + 2.7\text{k}\Omega}{165\text{k}\Omega + 4.7\text{k}\Omega}\right) = 1.899\text{V} \tag{4.23}$$

where $V_O$ is the output voltage from the regulator.

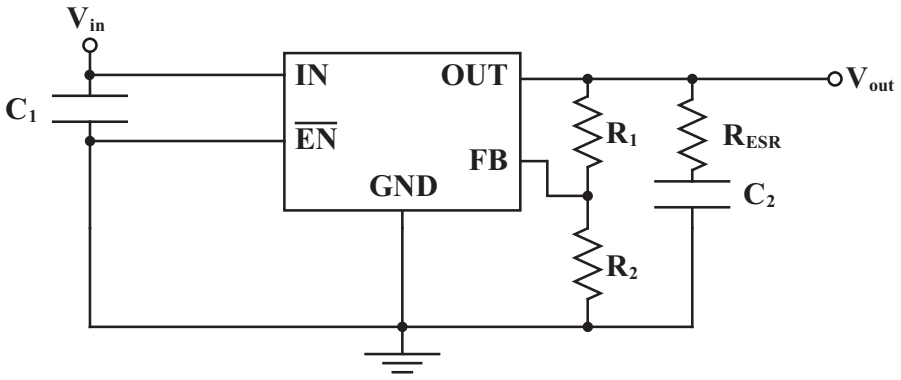



Figure 4.7: The regulator TPS76901 together with the external components.

## 4.3 MCU

The MCU that was used in this project is the Texas Instruments TMS320F28335. It was chosen because it is sufficiently fast with a clock frequency of 150MHz, there were experimenter's kits available already at QRTECH and there were also people at QRTECH that had previous experience working with that particular MCU.

### 4.3.1 Clock Configuration

The configuration of the MCU clock is shown in Figure 4.8. That particular configuration is specifically for when the internal clock of the MCU is used. It was decided that the internal clock would be used as adding an external clock would increase complexity and cost without adding much performance. The crystal shown in Figure 4.8 is specified at 30MHz and the capacitors $C_{L1}$ and $C_{L2}$ are specified as 33pF each.

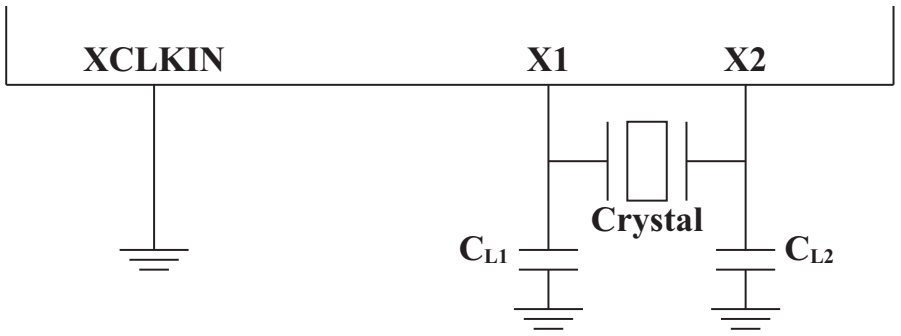



Figure 4.8: The configuration of the clock on the MCU.

### 4.3.2 ADC Reference

The configuration of the Analog-to-Digital Converter (ADC) submodule of the MCU is shown in Figure 4.9 and is taken from the datasheet.

The ADC inputs are connected to the measurements taken. On the primary side the inputs to the ADC are current, dc and ac voltage measurements and on the secondary side the ADC inputs are current and dc voltage measurements.

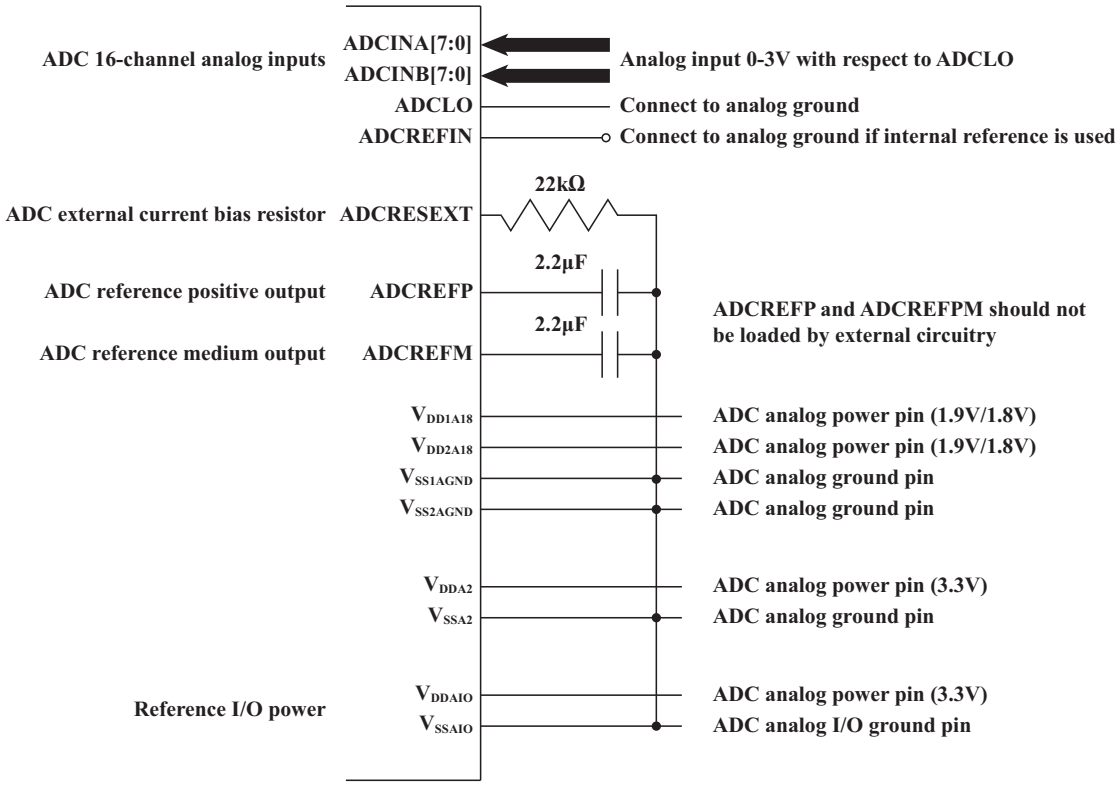



Figure 4.9: The connection of the ADC submodule of the MCU.

### 4.3.3 JTAG

The MCU has two ways of transferring and running C code. The first is writing the code to the flash memory of the MCU and the second is by connecting a Joint Test Action Group (JTAG) connector from the computer where the code is stored to the MCU each time the code is run. The latter was chosen as it was deemed to be more simple and less time consuming than writing to the flash memory as that would involve writing more code. The connection from the MCU to the JTAG connector is shown in Figure 4.10 and is taken from the datasheet for the MCU.

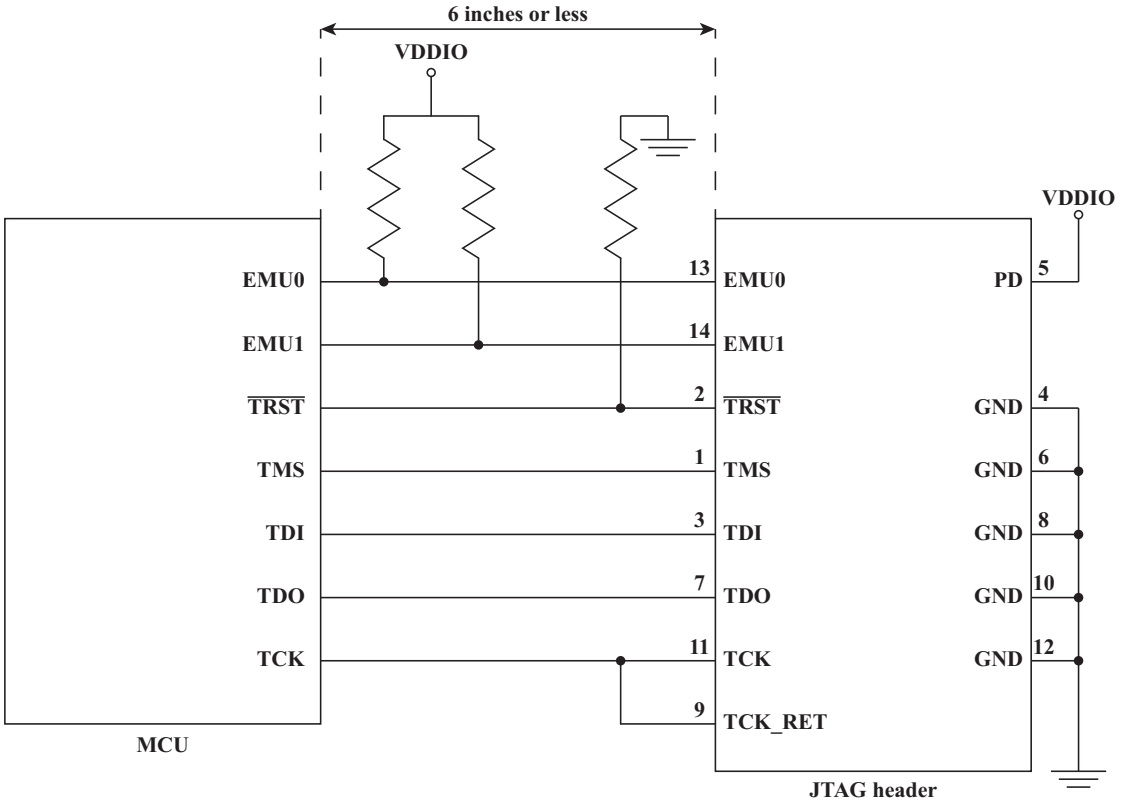



Figure 4.10: The connection between the JTAG connector and the MCU.

The resistors shown in Figure 4.10 all have a resistance of 2.2kΩ.

## 4.4 Radio Communication Circuit

The radio communication circuit, the Nordic Semiconductor nRF24L01+, works according to the Serial Peripheral Interface (SPI) protocol. It was chosen because it has a clear and simple to understand datasheet. It also has many functions that make it reliable and easy to implement such as automatic acknowledgement and automatic re-transmit. In addition to these functions the adress and Cyclic Redundancy Check (CRC) are relatively large making it unlikely that packets from other sources would be received by mistake. Thanks to these features it is relatively easy to implement in C code.

Besides the SPI there is one input and one output pin. The input pin is a Chip Enable (CE) pin. Its function is to enable the nRF24L01+ to either receive or transmit to another nRF24L01+ when it is high and when it is low to enable nRF24L01+ to receive or transmit instructions to the MCU.

The output pin is an interrupt (IRQ) pin. It is active low which means that when no

interrupt is present the output is high and when there is an interrupt the pin goes low. It gives interrupt signals in three situations. The first is if a message has been received, the second is if data has been transmitted and the third is if the maximum number of re-transmit attempts has been made without receiving an acknowledge packet from the receiving radio communication circuit.

## 4.5 Gate Driver

The gate driver, the IRS21867, is the component used to drive the transistors. Its circuit diagram is shown in Figure 4.11 The MCU cannot drive the transistors directly from its Enhanced Pulse Width Modulator (ePWM) output. This is because the MCU cannot deliver the voltage or the current needed to turn the transistors on and off. The gate driver receives the ePWM signal and makes sure that the transistors are operating as intended. The transistors need to be turned on and off very fast since the operating frequency is high. The better the ability to source and sink current the shorter the turn on and turn off time will be. The chosen gate driver has a source/sink capability of 4A and is able to drive a half-bridge. However, both the inverter and rectifier is a full-bridge consisting of four transistors. This means that two gate drivers are needed on each side - four in total. Thanks to its ability to drive two transistors simultaneously the complexity of the circuitry is decreased. Largely in part because the gate driver has built-in protection against short circuiting its half-bridge.

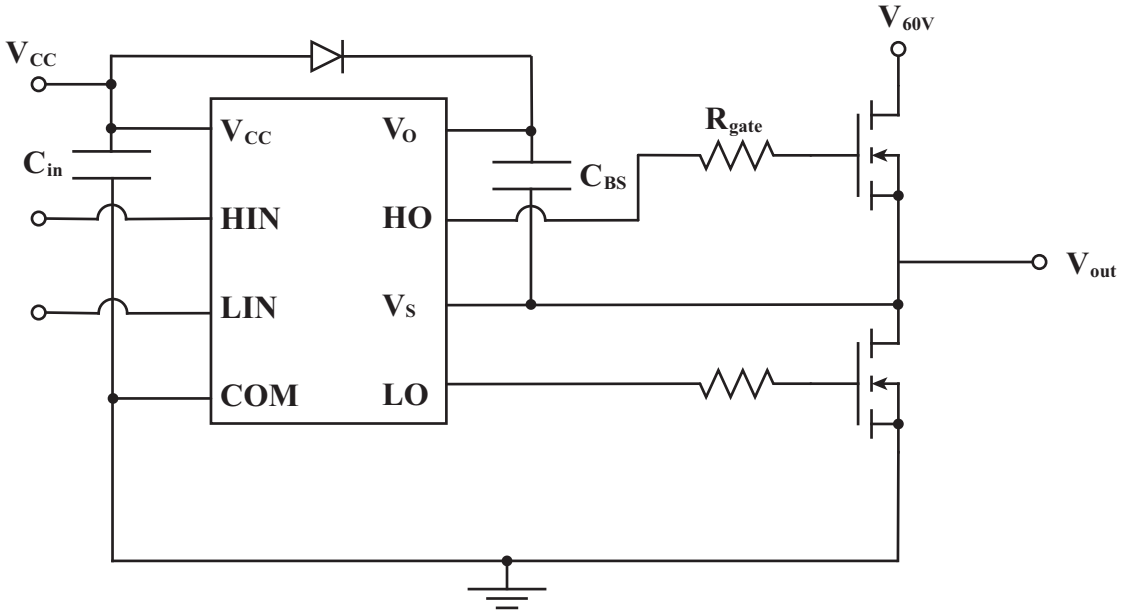



Figure 4.11: The IRS21867 together with its external circuitry and two transistors.

A $1\mu$F ceramic decoupling capacitor has been connected to the input of each gate driver. An external resistor is also needed in order to avoid having too large cuirrents through the gate driver component. The total resistance needed is

$$R_{gate} = \frac{U_{gate}}{I_{max,driver}} = \frac{15\text{V}}{4\text{A}} = 3.75\Omega \tag{4.24}$$

The transistor has an internal gate resistance of $0.8\Omega$. Therefore, the external gate resistance needs to be at least

$$R_{external} = R_{gate} - R_{transistor} = 3.75\Omega - 0.8\Omega = 2.95\Omega \tag{4.25}$$

Three $10\Omega$ resistances in parallel will give a total resistance of 3.33 $\Omega$. The total resistance will therefore be above the lowest allowed. With this setup the peak current from the gate driver is

$$I_{peak} = \frac{U_{gate}}{R_{total,gate}} = \frac{15\text{V}}{4.13\Omega} = 3.63\text{A}_{\text{peak}} \tag{4.26}$$

The external components needs to be able to dissipate a maximum power of

$$P_{gate} = C_{gate}U_{gate}^2 f = 57\text{nF} \cdot 15^2\text{V}^2 \cdot 100\text{kHz} = 1.3\text{W} \tag{4.27}$$

The resistances chosen can dissipate 0.5W each which gives a total dissipation capability of 1.5W. A bootstrap capacitor is needed to provide the upper transistor gate with a voltage above the source voltage. The bootstrap capacitor will be charged with the supply voltage (15V) and when the upper transistor is switched on, the bootstrap capacitor will be put on high potential and be able to supply the gate with its voltage. The bootstrap capacitor voltage variation is

$$\Delta V_{BS,max} = V_{DR} - V_{DBS} - V_{GS,min} = 15\text{V} - 1.2\text{V} - 2 \cdot 4\text{V} = 5.8\text{V} \tag{4.28}$$

where $\Delta V_{BS,max}$ is the maximum allowed voltage variation for the bootstrap capacitor, $V_{DR}$ is the supply voltage, $V_{DBS}$ is the voltage drop over the bootstrap diode and $V_{GS,min}$ is the minimum gate voltage to the transistor (2 x $V_{threshold}$). The charge needed in the bootstrap capacitor is

$$\Delta Q_{BS,min} = Q_G + Q_{LS} + \frac{I_{QBS,max}}{f_{sw}} = 570\text{nC} - 5\text{nC} - \frac{150\mu\text{A}}{10\text{kHz}} = 0.59\mu\text{C} \tag{4.29}$$

where $Q_G$ is the gate charge, $Q_{LS}$ is the levelshifter charge needed and $I_{QBS,max}$ is the current needed by the floating part. The minimum bootstrap capacitor is therefore

$$C_{BS,min} = \frac{\Delta Q_{BS,min}}{\Delta V_{BS,max}} = \frac{0.59\mu\text{C}}{5.8\text{V}} = 0.102\mu\text{F} \tag{4.30}$$

To make sure that there is a good margin relative the calculations, the following margin is applied

$$C_{BS} = 15 \cdot C_{BS,min} = 15 \cdot 0.102\mu\text{F} = 1.53\mu\text{F} \tag{4.31}$$

Two parallel $1\mu$F ceramic capacitors were used ($2\mu$F in total) as bootstrap capacitor.

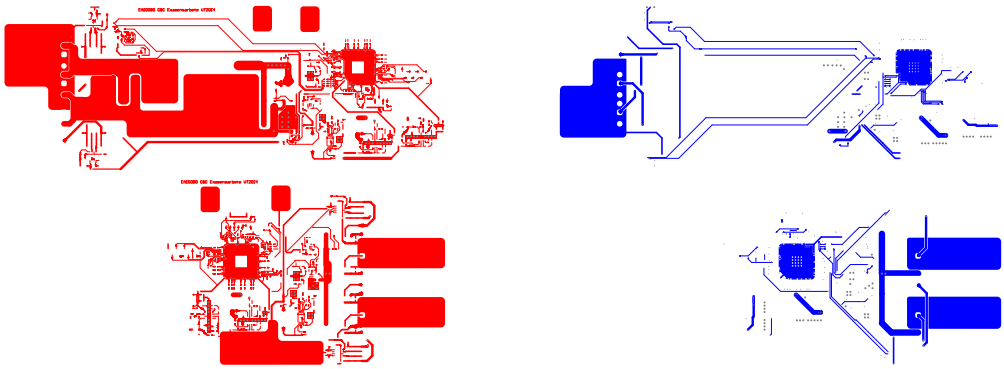
## 4.6 Decoupling Capacitors

The small decoupling capacitors ($100\text{nF} - 1\mu\text{F}$) are used to stabilise the voltage input to the different devices. It is common practice to use decoupling capacitors close to the power supply pins of the MCU or some other component which requires power. Decoupling capacitors is also used together with the H-bridge in the inverter to stabilise the voltage, the H-bridge has four $68\mu$F electrolytic capacitors. The alternating current that commutates through the H-bridge can cause voltage fluctuations in the dc link, these fluctuations are damped by the large decoupling capacitors.

## 4.7 PCB Layout

The PCB layout was made after all the active and passive components had been chosen and the basic circuitry design had been made. The PCB layout is the realisation of the basic circuit diagrams. It is the process where the placement of the different components are made and the conductive paths between the components are drawn. It is important to pay attention to all the requirements that are imposed on the circuit board to be able to make a well functioning PCB layout. The PCB layout is subjective to a high degree. There are some general guidelines to follow but most of the layout depends on the designer's decisions. It is difficult to automatically generate a PCB layout without a lot of effort put in to configuring the EDA software. Therefore, the PCB was manually designed.

In Figure 4.12 the top and bottom layer of the PCB layout is shown. The top and bottom layers are mostly used for signal routing while the two middle layers are used as ground and power planes. The tracks that are supposed to transfer power were made as wide as possible to minimise conductor impedance. The via holes are responsible for connecting the different planes. These can become warm if they conduct too much current and they can also be a limiting factor if the frequency is very high due to the extra inductance it adds.



(a) Top Layer (b) Bottom Layer

Figure 4.12: The top and bottom layer of the PCB layout.

## 4.8 H Bridge

Both the conversion from dc to ac on the primary side and the conversion from ac to dc on the secondary side is handled by transistors that are connected in an H bridge configuration. These will be discussed in greater detail henceforth.

### 4.8.1 Inverter

The inverter is situated on the primary side. It needed to be able to withstand at least 60V from the dc power supply as it was uncertain what dc voltage was needed on the primary side to yield a sufficient voltage on the secondary side. Transistors which can withstand a voltage of 75V was choosen for the inverter.

### 4.8.2 Rectifier

The rectifier is situated on the secondary side. A rectification stage without output filter will produce an output such as shown by the black curve in Figure 4.13. It is the absolute value of the incoming sine wave to the rectification stage. When an output filter is added the resulting output voltage will be a dc voltage. It will have an amplitude that is the average of the unfiltered output voltage as shown by the blue curve in Figure 4.13. The average can be calculated as

$$V_{avg} = \frac{1}{\pi}\int_0^{\pi} V_{peak} sin(\omega t)\mathrm{d}\omega t \tag{4.32}$$

and the result from this calculation will be

$$V_{avg} = \frac{1}{\pi}\left(-V_{peak}cos(\omega t)\right)\Big|_0^{\pi} = \frac{V_{peak}}{\pi}\left(-(-1)-(-1)\right) = \frac{2V_{peak}}{\pi} \tag{4.33}$$

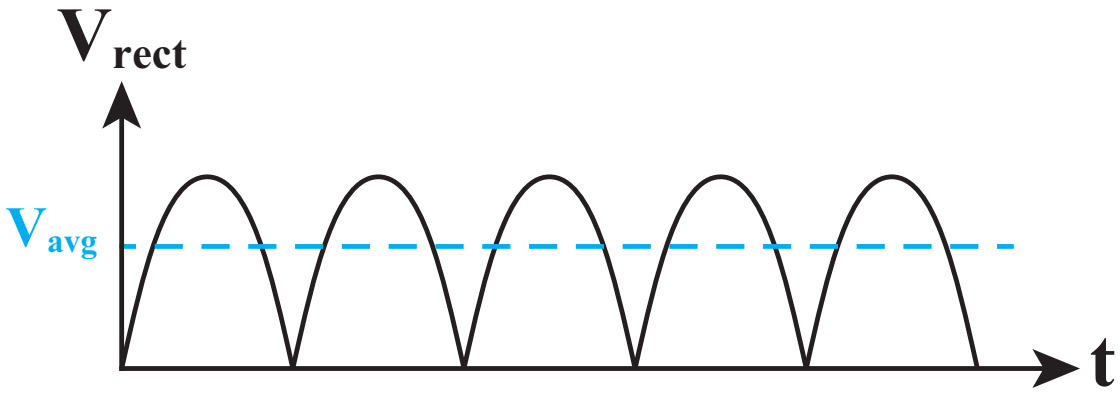



Figure 4.13: The output from a rectifier stage with and without output filter.

From (4.33) it can be seen that for $V_{avg} \approx 55V$ a sine wave with an amplitude of

$$V_{peak} = \frac{\pi \cdot V_{avg}}{2} = \frac{\pi \cdot 55V}{2} \approx 86.4V \tag{4.34}$$

is needed. In order to handle this voltage a transistor rated at 150V was chosen, which leaves a sufficient margin.

#### Output Filter

As the output filter is supposed to smooth out a rectified dc voltage with a frequency of about 10 kHz to 100 kHz the cut-off frequency of the output filter needs to be substantially lower than these values. The cut-off frequency was therefore set to approximately 1.1 kHz which is obtained by using a 100 $\mu$H inductor and three 68 $\mu$F capacitors as shown by

$$f_c = \frac{1}{2\pi\sqrt{LC}} = \frac{1}{2\pi\sqrt{100 \cdot 10^{-6} \cdot 3 \cdot 68 \cdot 10^{-6}}} = 1.1143...\text{kHz} \approx 1.1\text{kHz} \tag{4.35}$$

## 4.9 Measurements

Measurements were implemented in order to aid the MCU efficiently. On the primary side a current measurement on the negative leg near the battery, and a dc voltage measurement on the input from the dc power supply was implemented. The measurements

that were implemented on the secondary side were a current measurement near the battery on the negative leg, a dc voltage measurement over the battery, and an ac voltage measurement on the incoming sine wave from the magnetic coils.

### 4.9.1 Current Measurement

A small resistor is added on the negative leg close to the battery. The reason why the resistor is added on the negative leg is because of the much lower voltage potential compared to the positive leg. The resistor chosen had a resistance of 10mΩ to minimize losses in the current measurement. The current measurement is done by measuring the voltage over the resistor and then letting the MCU calculate the current flowing through it. However, as the resistor has such a small resistance the voltage over it will be small which will yield a low resolution on the measurement. Therefore, it is needed to amplify the measurement in order to enhance the resolution. To do this a differential amplifier is used. A schematic of the differential amplifier and how it is connected to the resistance on the negative leg is shown in Figure 4.14.

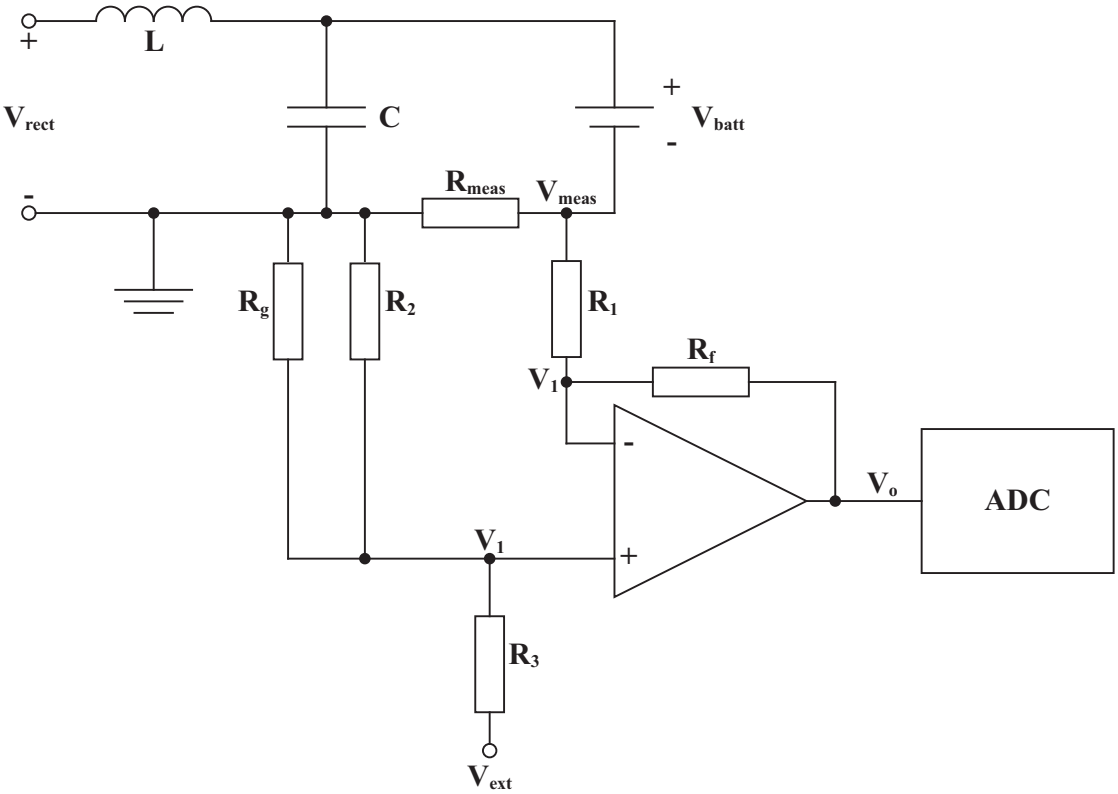



Figure 4.14: The differential amplifier used to amplify the current measurement signal.

By using Kirchoff's Current Law (KCL) an expression for the differential amplifier's

output voltage can be obtained. The expression for the negative pin is

$$\frac{V_{meas}-V_1}{R_1}+\frac{V_o-V_1}{R_f}=0 \tag{4.36}$$

and the expression for the positive pin is

$$\frac{0-V_1}{R_2}+\frac{V_{ext}-V_1}{R_3}+\frac{0-V_1}{R_g}=0 \tag{4.37}$$

By rewriting the expression for the positive pin the equation

$$V_1=\frac{R_2R_g}{R_3R_g+R_2R_g+R_2R_3}V_{ext} \tag{4.38}$$

for $V_1$ is obtained. Now, substituting $V_1$ in the expression for the negative pin will yield

$$V_o=R_f\left[V_{ext}\frac{R_2R_g}{R_3R_g+R_2R_g+R_2R_3}\left(\frac{1}{R_1}+\frac{1}{R_f}\right)-\frac{V_{meas}}{R_1}\right] \tag{4.39}$$

for the output voltage of the differential amplifier. Since the resistance used for the measurement has a value of 10mΩ and the maximum current to the battery is 9A the maximum voltage over the resistance will be

$$V_{meas}=0.01\cdot 9=0.09\text{V} \tag{4.40}$$

Therefore, if the resistance values in Table 4.1 are chosen and $V_{ext}=1.9$V the output voltage for a 9A current will be $V_o\approx 1.97$V and for a 0A current it will be $V_o\approx 2.38$V. This means that the output voltage range will be $\Delta V_o\approx 0.41$V. As the MCU can measure with an accuracy of 0.8mV it means that it will be able to measure the current between 0 and 9A in 0.41V/0.8mV $\approx$ 511 different levels. This equals to a current accuracy of 9A/511 $\approx$ 17mA which is enough for the applications it is used for in this project.

Table 4.1: The resistances chosen for the differential amplifier.

| **Resistor** | **Resistance value** |
|---|---|
| $R_1$ | 2.2kΩ |
| $R_2$ | 18kΩ |
| $R_3$ | 22kΩ |
| $R_f$ | 10kΩ |
| $R_g$ | 10kΩ |

### 4.9.2 dc Voltage Measurement

Since the MCU cannot measure voltages above 3V the output voltage to the battery cannot be readily measured. The voltage needs to be lowered in order to allow for measurements to be made. An easy way of doing this is by voltage division as shown in Figure 4.15.

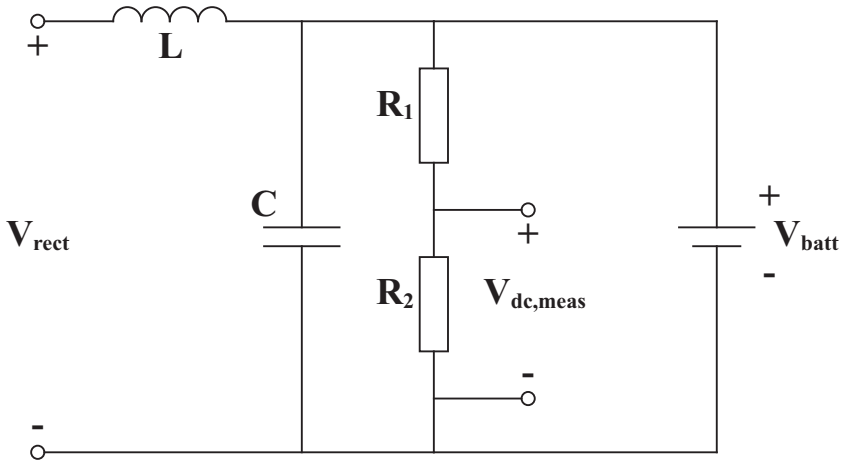



Figure 4.15: Voltage division of dc voltage measurement signal.

The voltage that $V_{dc,meas}$ measures is a fraction of the battery voltage and can be expressed as

$$V_{dc,meas} = \frac{R_2}{R_1 + R_2} V_{batt} \tag{4.41}$$

and it can be rewritten as

$$R_2 = \frac{V_{dc,meas}}{V_{batt} - V_{dc,meas}} R_1 \tag{4.42}$$

The ADC inputs can operate in the range between 0V to 3V, the battery voltage is 60V maximum and R1 is set to 100kΩ. $R_2$ can then be calculated to be

$$R_2 = \frac{3\text{V}}{60\text{V} - 3\text{V}} 100\text{k}\Omega \approx 5.3\text{k}\Omega \tag{4.43}$$

The closest value rounded downwards (so as to not reach a voltage level $V_{dc,meas} > 3$V) available in the Altium database is 5.1kΩ which yields a $V_{dc,meas}$ of

$$V_{dc,meas} = \frac{5.1\text{k}\Omega}{5.1\text{k}\Omega + 100\text{k}\Omega} 60\text{V} \approx 2.91\text{V} \tag{4.44}$$

This means that there will be 2.91V/0.8mV $\approx$ 3639 voltage levels that the MCU can detect because of its 0.8mV accuracy.

### 4.9.3 ac Voltage Measurement

The ac voltage measurement needs to bring the voltage down to the right voltage range for the MCU. This is because the ac voltage can also be negative but the ADC can only measure positive voltages. One way of circumventing this is by adding a third resistor connected to an external voltage as in Figure 4.16.

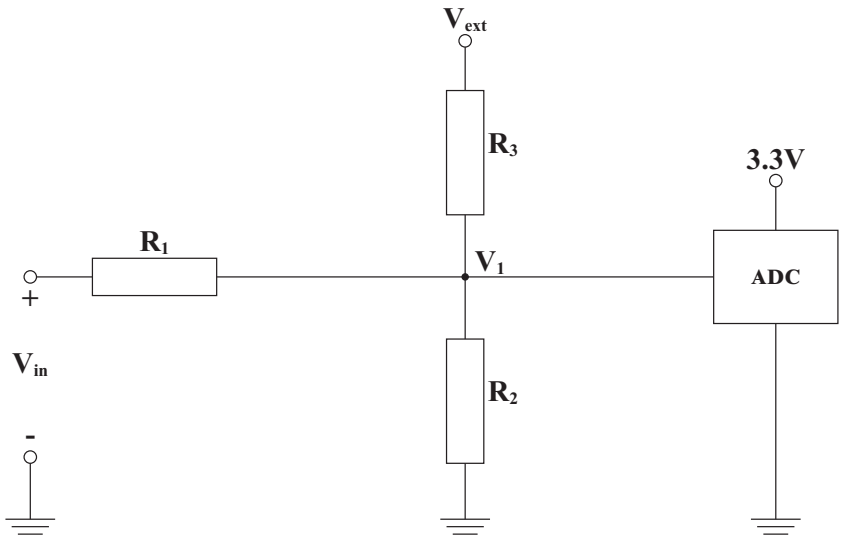



Figure 4.16: Voltage division of dc voltage measurement signal.

By using a third resistor connected to an external voltage source the measured voltage, $V_1$, is increased so there is always a positive voltage even when the incoming ac voltage is negative. By using KCL the expression for the circuit becomes

$$\frac{V_{in}-V_1}{R_1}+\frac{0-V_1}{R_2}+\frac{V_{ext}-V_1}{R_3}=0 \tag{4.45}$$

which can be rewritten as

$$V_1=\frac{\frac{V_{in}}{R_1}+\frac{V_{ext}}{R_3}}{\frac{1}{R_1}+\frac{1}{R_2}+\frac{1}{R_3}} \tag{4.46}$$

and if, to simplify the calculations, $R_2$ is chosen to be the same value as $R_3$ the expression can be further rewritten as

$$V_1=\frac{\frac{V_{in}}{R_1}+\frac{V_{ext}}{R_3}}{\frac{1}{R_1}+\frac{2}{R_3}} \tag{4.47}$$

The ratio between $R_1$ and $R_3$ can now be written as

$$\frac{R_1}{R_3} = \frac{V_1 - V_{in}}{V_{ext} - 2V_1} \tag{4.48}$$

Since it is not known what voltage levels will be measured $V_{in}$ is estimated to be 100V maximum and $V_1$ is set to 2.5V at that voltage level. As the ADC can measure up to 3.0V this leaves some margin in case the estimation is too low. $V_{ext}$ is set to 3.3V also. The ratio between $R_1$ and $R_3$ can now be calculated as

$$\frac{R_1}{R_3} = \frac{2.5 - 100}{3.3 - 2 \cdot 2.5} \approx 57.353 \tag{4.49}$$

Since most resistors have a maximum power rating of 0.25W or less the resistance value chosen for $R_1$ is chosen to be high, specifically 270kΩ. This means that the power dissipation in the resistance is $(100\text{V})^2/270\text{k}\Omega \approx 0.037\text{W}$ which is far lower than the specified maximum power rating. Using the ratio from (4.49) the value for $R_2$ and $R_3$ can be obtained as

$$R_2 = R_3 = 57.353R_1 \approx 4.7\text{k}\Omega \tag{4.50}$$

which is available in the E12 resistor series.

## 4.10 Software Implementation

The programming language used was C. On the primary side the peripherals used on the MCU are ADC and ePWM. Code has been prepared for the secondary side which entails the peripherals ADC, Enhanced Capture (eCAP), ePWM and SPI. The code on the secondary side has been written in order to simplify future development of the charger.

### 4.10.1 Primary Side

The MCU on the primary side has several functions it needs to carry out. The ADC submodule needs to measure the dc voltage and current to make sure they are not too high and also measure the voltage from a potentiometer in order to calculate the correct output frequency for the ePWM submodule. The ePWM submodule needs to set a square-wave output on the ePWM1a, ePWM1b, ePWM2a and ePWM2b pins with a precise frequency, given by the ADC submodule, and deadband. It is also responsible for keeping the ePWM2 pins in phase with the ePWM1 pins. The source code for the primary side is shown in Appendix A.

### 4.10.2 Secondary Side

The secondary side is responsible for converting the ac power into dc power for the battery. In order to do this effectively the MCU needs to utilize the ADC, eCAP, ePWM and SPI submodules. The ADC submodule is needed for measuring the dc voltage and current to ensure that they are not too high. The eCAP submodule needs to measure the period time and duty cycle of the incoming ac voltage in order to achieve Zero-Voltage Switching (ZVS). the ePWM submodule is needed to set a square-wave output to the ePWM1a, ePWM1b, ePWM2a and ePWM2b pins with a precise frequency, given by the eCAP submodule. It is also needed for synchronizing the square-wave output with the incoming ac voltage in order to achieve ZVS and to set the deadband. The SPI submodule is responsible for the automatic turn-on and turn-off for which communication with the primary side is needed and to communicate to the primary side to turn off the inverter in the case of a dangerously high dc voltage or current on the secondary side. The source code for the secondary side is shown in its entirety in Appendix B.

## 4.11 Setup

Figure 4.17 illustrates the inverter and its various parts and components.

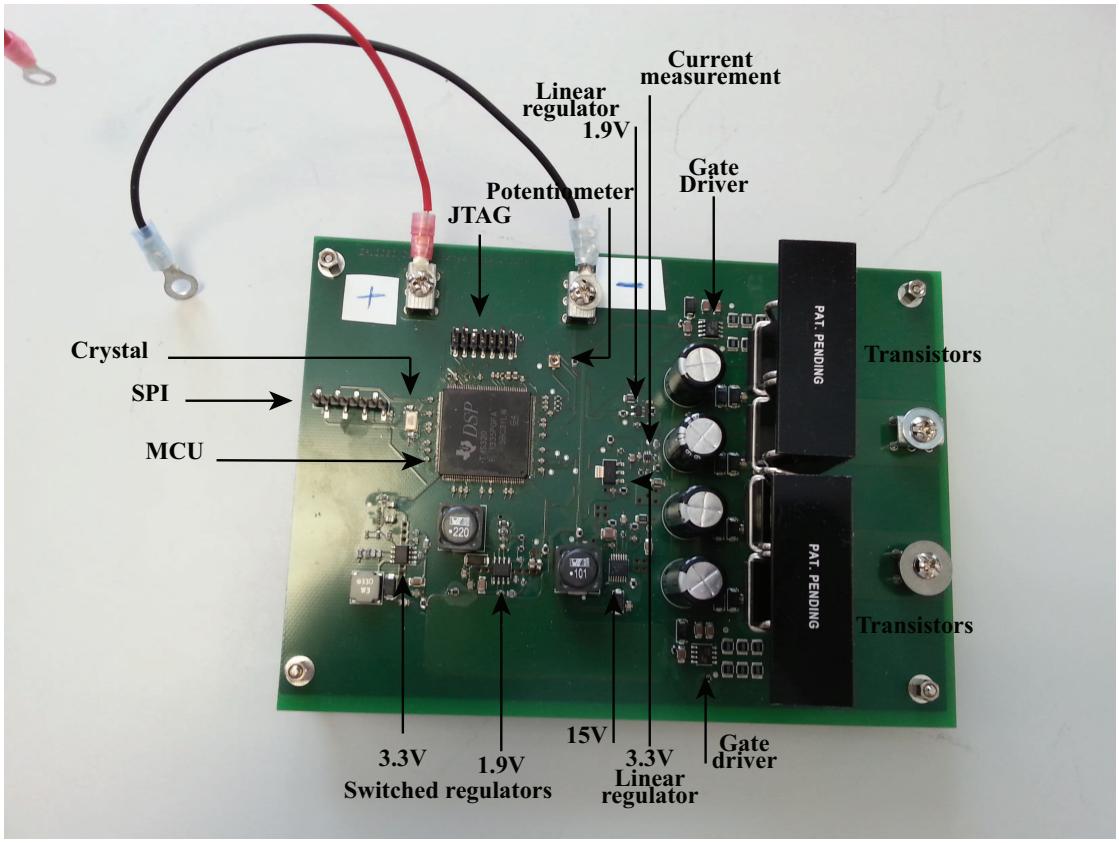



Figure 4.17: The setup of the inverter with its various parts marked.

The coils are shown in Figure 4.18

Figure 4.18: The coils and ferrites.

Figure 4.19 shows the rectifier and its various parts and components.

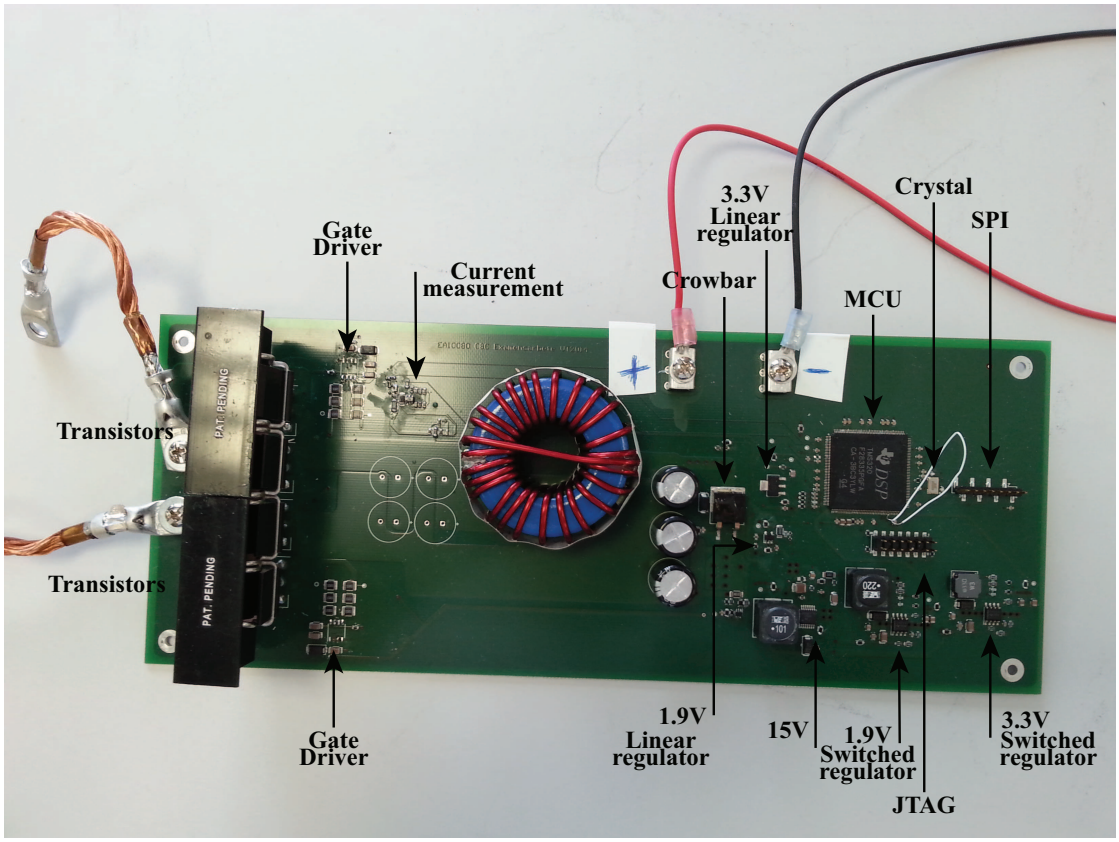



Figure 4.19: The setup of the rectifier with its various parts marked.

# Chapter 5

# Analysis

This chapter covers the evaluation of both the complete charging system as well as the separate parts. Measurements are presented as a basis for evaluation.

## 5.1 Coils and Resonance Circuits

The resonance circuit shown in Figure 4.2 is rated for less voltage and current that they need to withstand. Each capacitor is rated for 400V dc and 250V ac. However, at 50kHz each capacitor is only rated at 30V. With the setup used, the voltage is spread out across three capacitors which means that the total rated voltage of this particular setup is only 90V. When transferring maximum designed power using a 6.9Ω load the voltage is slightly above an amplitude of 90V. However, when using a lower load this voltage is surpassed by a larger amount. Therefore, it might not be able to tolerate charging a lower load for a longer period of time.

It is possible for the coils to be less than optimally placed by having an offset in at least one of the three dimensions. To evaluate the impact of having an offset measurements were taken when offsets were introduced in the x and y direction and for different air gaps. To simplify the evaluation process and ensuring that the ensuing analyses were correct, offsets were only introduced in one dimension at a time.

For all measurements taken when introducing different offsets a 6.9Ω resistor was used as load, the voltage was kept constant at 40V from the dc power supply and the efficiency referred to is between the dc power supply and the load. Measurements were taken both with the frequency optimized and with a constant frequency of 49kHz.

### 5.1.1 Offset in X and Y Evaluation

When an offset is introduced in the x or y direction the efficiency will be impacted. The change in efficiency when there is a offset in the x direction is shown in Figure 5.1. For

the x axis offset the optimal operating frequency was constant at 49kHz.

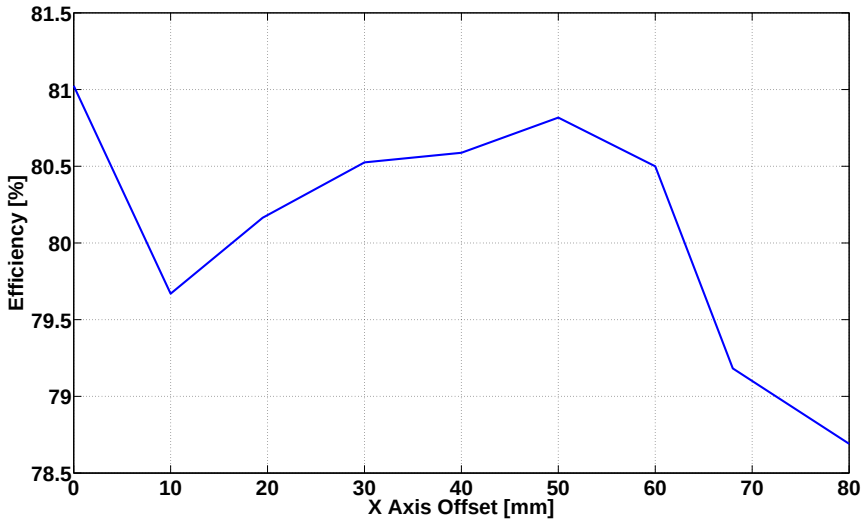



Figure 5.1: The efficiency of the charger versus the different offsets x direction.

The efficiency drops off slowly as the offset is increased, the drop off in efficiency increases above 50mm offset. The same measurement can be made with different offsets in the y direction as seen in Figure 5.2.

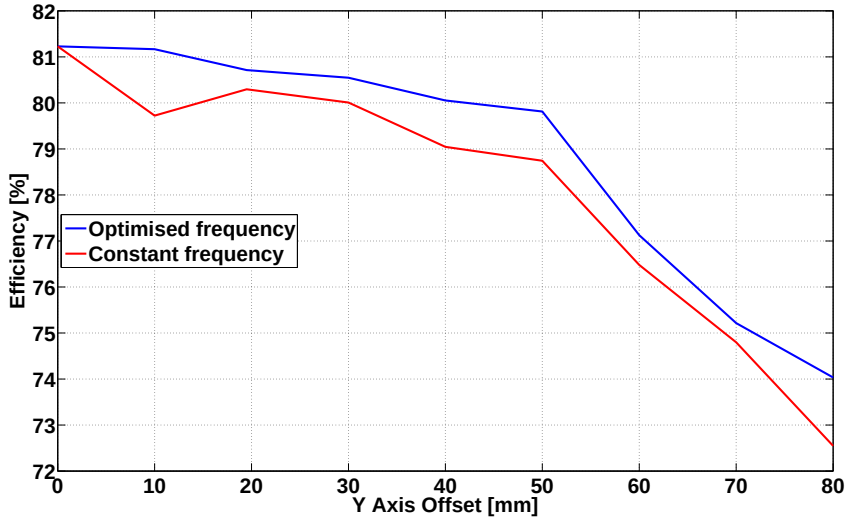



Figure 5.2: The efficiency of the charger versus the offset in y direction between the coils.

The measurements shown in Figure 5.1 and Figure 5.2 indicate a decrease in efficiency when the offset is increased in the different directions. However, the decrease in efficiency is quite small up to at least an 80mm offset.

The change in frequency when there is an offset in the x or y direction is much less clear than when the air gap is changed. The trend is neither for the frequency to increase or decrease but is instead approximately stable around 49kHz for all tested amounts of offset in those directions.

### 5.1.2 Air Gap Evaluation

The efficiency of the charger was measured for different air gaps as shown in Figure 5.3. It shows a negative correlation between an increased air gap and efficiency, the negative trend for the efficiency is more pronounced above 50mm. The efficiency is slightly higher when the operating frequency is optimised for each gap length rater than having a constant operating frequency.

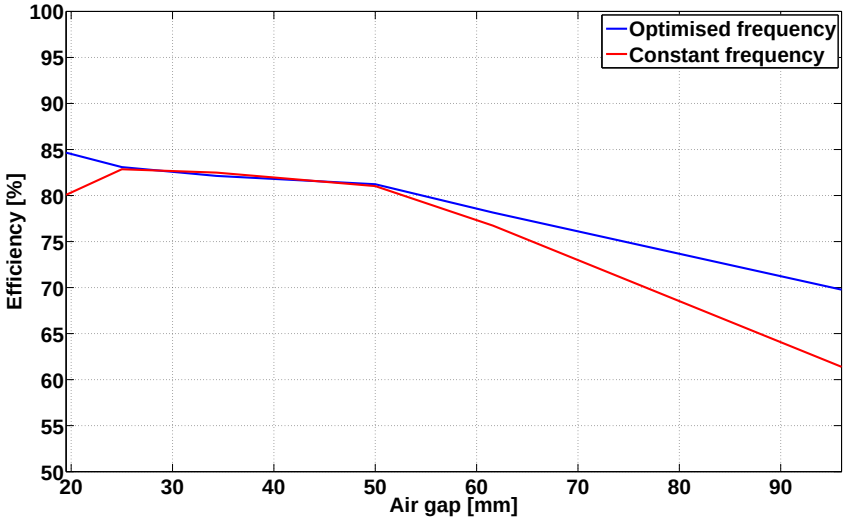



Figure 5.3: The efficiency of the charger versus the air gap between the coils.

The optimal frequency for maximum efficiency changes when there is an offset between the two coils. This is especially true when the air gap between the two sides change. The reason for this is that the mutual inductance between the two coils changes depending on the distance between them which in turn causes the resonance frequency of the system to change. The relation between air gap and optimal frequency for maximum efficiency is shown in Figure 5.4.

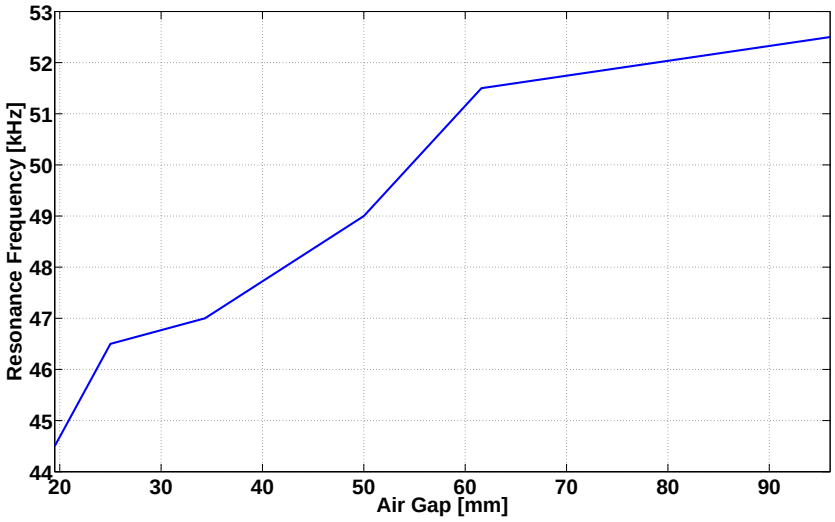



Figure 5.4: The optimal frequency for maximum efficiency versus air gap.

## 5.2 Inductance Measurement

The inductance of the magnetic circuit was measured using an ac source, a voltmeter and an current probe. The measurements were carried out using a sinusoidal voltage with a frequency of 30 kHz. The reactive part of the impedance was assumed to be much larger than the resistive part. Thus, the resistive part was neglected and the phase shift can be considered to be approximately 90 degrees as the impedance would be entirely inductive. The leakage factor $\sigma$ was calculated as

$$\sigma = 1 - \frac{L_{12}^2}{L_1 L_2} = 1 - K^2 = \frac{L_{sc}}{L_{oc}} = \frac{i_{oc}}{i_{sc}} \tag{5.1}$$

where the ratio between the open circuit current, $i_{oc}$, and the short circuit current, $i_{sc}$, was used to calculate the leakage factor. The primary and secondary inductances were calculated as

$$L_{1,2} = \frac{U_{peak}}{2\pi f I_{peak}} \tag{5.2}$$

Measurements were made from both sides of the magnetic circuit to measure both $L_1$ and $L_2$. The turns ratio a is given by

$$a = \sqrt{\frac{L_1}{L_2}} \tag{5.3}$$

and the mutual inductance $L_{12}$ was calculated as

$$L_{12} = \sqrt{L_1 L_2 (1-\sigma)} \tag{5.4}$$

With this information, the leakage and magnetising inductances can be calculated using

$$L_M = aL_{12} \tag{5.5}$$
$$L_{\lambda 1} = L_1 - aL_{12} \tag{5.6}$$
$$L_{\lambda 2} = L_2 - aL_{12} \tag{5.7}$$

The measurements of the inductances on both the primary and secondary side are shown in Figure 5.5.

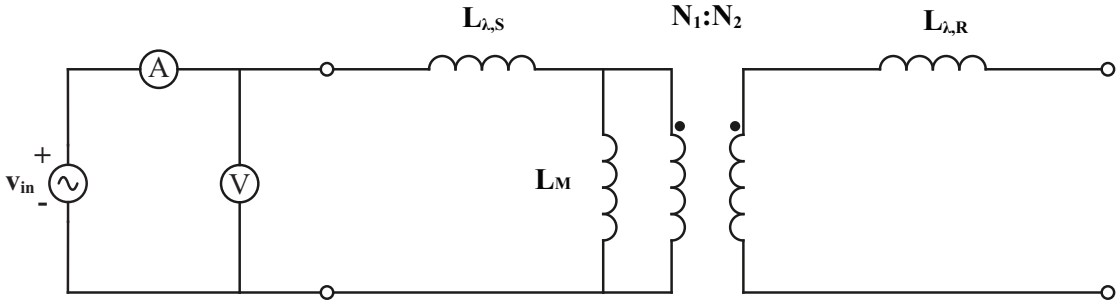



Figure 5.5: Measurement setup for the inductance measurement.

As the air gap increases the reluctance will increase due to the low permeability of air. This will lead to a decrease of the magnetising inductance. The relation between the air gap and the value of the inductances is illustrated in Figure 5.6. The changes in the inductances shown in the figure is one cause to the resonance frequency changing with air gap.

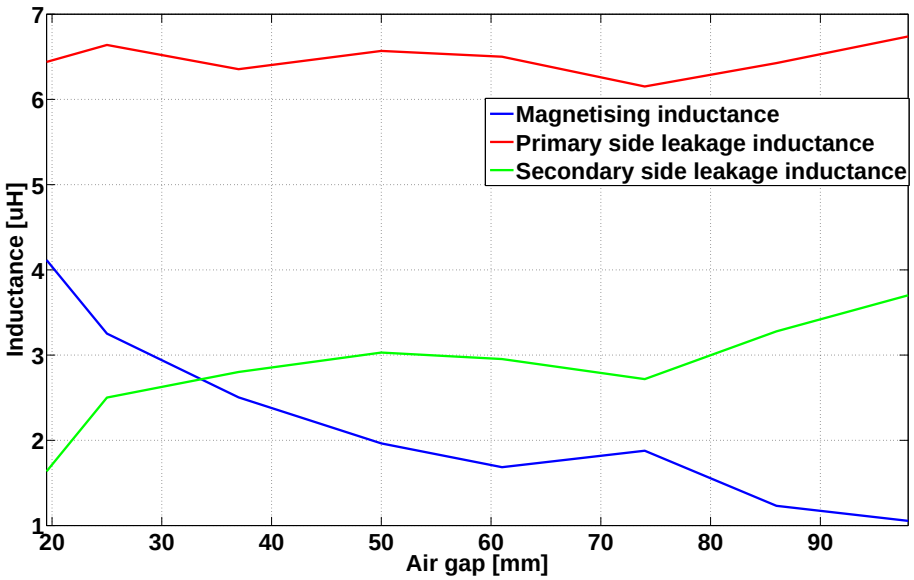



Figure 5.6: Primary, secondary and magnetising inductances versus the air gap.

## 5.3 Winding Resistance Measurement

To measure the dc resistance of the coils, a dc current was passed through the litz wire and the voltage over the wire was measured. The measurement setup is shown in Figure 5.7. This type of measurement was performed because the resistance was too low to measure directly with an Ω-meter. The calculated resistances based on these measurements are shown in Table 5.1.

Table 5.1: Winding resistance

| **Winding** | **Resistance** |
|---|---|
| 1 | 0.714mΩ |
| 2 | 0.698mΩ |
| 3 | 0.675mΩ |
| 4 | 0.679mΩ |
| *avg* | 0.691mΩ |

The coils are connected in parallel, with coil 1 and 4 connected on the primary side and coil 2 and 3 on the secondary side.

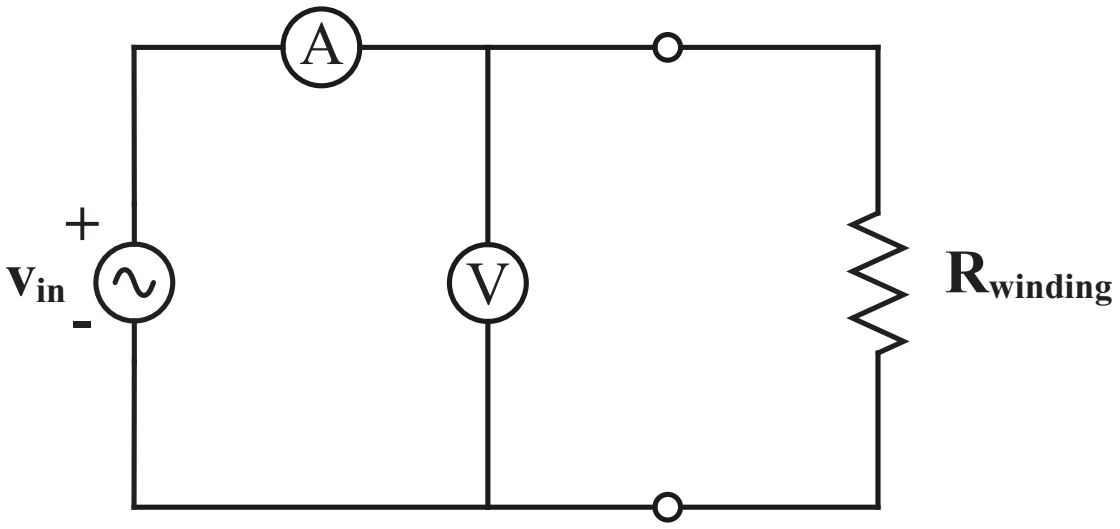



Figure 5.7: Measurement setup for the winding resistance measurement.

The total dc resistance on the primary side is therefore

$$R_{primary} = \frac{0.714\text{m}\Omega \cdot 0.679\text{m}\Omega}{0.714\text{m}\Omega + 0.679\text{m}\Omega} \approx 0.348\text{m}\Omega \tag{5.8}$$

and for the secondary side it is

$$R_{secondary} = \frac{0.698\text{m}\Omega \cdot 0.675\text{m}\Omega}{0.698\text{m}\Omega + 0.675\text{m}\Omega} \approx 0.343\text{m}\Omega \tag{5.9}$$

# Chapter 6

# Discussion

It has been shown that it is possible to transfer electric energy wirelessly with a competitive efficiency when compared to wired transmission. The convenience factor of using inductive charging most likely surpasses that of wired transfers enough to entice consumers despite the losses being higher. However, a standard is needed in order to ensure that all, or most, vehicles can be charged using the same charging spot.

Above a certain threshold voltage the necessity of active rectification can be questioned. As described in Section 2.5.2 the advantage of the transistor will diminish as the current increases. Even before this point diodes have certain advantages over transistors. Firstly, the complexity of the circuits are lower when using diodes since gate drivers and such are not needed. Secondly, the price is higher when using transistors as more components are needed to control the switching. However, one advantage that transistors have are that they can be used for controlling the output voltage. As the voltage is 55V in this case, diodes might still be a better choice for this design.

It was found that the currents through the coils were relatively high with the chosen design. This was due to the capacitor bank being placed in parallel on the secondary side. One way of mitigating this issue might be to place the capacitor bank in series with the coils on the secondary side. With this design the load is placed in series with both the coils and the capacitor bank. This means that the same current that flows to the load also flows through the coil and the capacitor bank. The advantage of having the same current is that it would be smaller relative to the parallel case. When a smaller current flows through the coil they can be designed to be smaller and more cost-effective.

The MCU that was chosen for this project needs two voltage supplies, both 1.9V and 3.3V. Instead, an MCU that only needs a 3.3V power supply could have been used. The MCU also lacked internal comparators which would have been useful for supplying the ePWM peripheral with phase and period. Furthermore, it would have been useful if the ADC peripheral would have had a faster sampling rate.

The shunt resistor used for the current measurement on the primary side got heated up during operation. To mitigate this, more of them could have been connected in

parallel. This would have the effect of both lowering the resistance and also spreading the power loss over several components. Thus, dividing the power dissipation across several components. Furthermore, the resistors used for the differential amplifier results in a relatively small span of samples that the ADC peripheral receives. These could have been chosen differently to increase the span and the precision of the measurements.

The fluctuations of the current near the power supply on the primary side were higher than expected. Therefore, larger capacitors would be preferable in order to decrease these fluctuations at the power supply.

# Chapter 7

# Conclusion

An inductive charging station has been designed, constructed and optimised for an electric go kart. The peak efficiency reached approximately 85%. This was achieved with a 2 cm air gap between the primary and secondary coil. The maximum supplied power was 605W and the corresponding power output during this operation was 479W. The nominal frequency chosen for this project was 49kHz. It was chosen mainly because it was in the middle of the desired frequency span and also because it was easy to obtain with the available capacitors. The input voltages from the power supply during the testing sessions were between 24 and 63V. The resonance circuit configuration was chosen to be a parallel resonance circuit on the secondary side only. The turns ratio of the magnetic coils were 4:3 because of the high voltage output when having a 4:4 ratio. The rectification of the ac input was handled by the body diodes of the transistors on the secondary side. The load on the secondary side consisted of a power resistor. During most tests the load was set to 6.9Ω.

Measurements have been carried out to investigate whether misalignment of the magnetic coils impact the efficiency of the charging system to a large degree. The efficiency did not vary appreciably when discplacements were introduced in the x and y direction. However, when the air gap was increased the efficiency quite rapidly diminished.

# References


[1] H. Riebeek, "Global warming," Feb. 2014. [Online]. Available: http://earthobservatory.nasa.gov/Features/GlobalWarming/

[2] NASA Earth Observatory, "World of Change: Global Temperatures," Feb. 2014. [Online]. Available: http://earthobservatory.nasa.gov/Features/WorldOfChange/decadaltemp.php

[3] United States Environmental Protection Agency, "Sources of Greenhouse Gas Emissions," Feb. 2014. [Online]. Available: http://www.epa.gov/climatechange/ghgemissions/sources.html

[4] The International Council on Clean Transportation, "Consumer Acceptance of Electric Vehicles in the US." Feb. 2014. [Online]. Available: http://www.epa.gov/oar/caaac/mstrs/dec2012/kodjak.pdf

[5] Nissan Motor Company, "Wireless charging system," Feb. 2014. [Online]. Available: http://www.nissan-global.com/EN/TECHNOLOGY/OVERVIEW/wcs.html

[6] D. K. Cheng, *Field and Wave Electromagnetics*, 2nd ed. Reading, Massachusetts, USA: Addison-Wesley, 1989.

[7] J. W. Jewett and R. A. Serway, *Physics for Scientists and Engineers with Modern Physics*, 7th ed. Stamford, Connecticut, USA: Cengage Learning, 2008.

[8] A. P. Hu, *Wireless/Contactless Power Supply*. Saarbrücken, Germany: VDM Verlag Dr. Müller Aktiengesellschaft & Co. KG, 2009.

[9] R. C. Dorf and J. A. Svoboda, *Introduction to Electric Circuits*. Hoboken, New Jersey, USA: John Wiley & Sons Ltd, 1993.

[10] M. Cederlöf, "Inductive Charging of Electrical Vehicles," master's thesis, KTH Royal Institute of Technology, 2012.

[11] Z. Popovic and B. D. Popovic, *Introductory Electromagnetics*. Upper Saddle River, New Jersey, USA: Prentice Hall, 1999.

[12] Ferronics Inc., "Company Profile," Feb. 2014. [Online]. Available: http://www.ferronics.com/company.html

[13] ——, "Ferrite Terminology," Feb. 2014. [Online]. Available: http://www.ferronics.com/files/terms.pdf

[14] M. Budhia, G. Covic, and J. Boys, "A new ipt magnetic coupler for electric vehicle charging systems," in *IECON 2010 - 36th Annual Conference on IEEE Industrial Electronics Society*. Glendale, AZ: IEEE, Nov 2010, pp. 2487 – 2492.

[15] J. Andersson, M. George, S. Mesic, D. Stenberg, and P. Stenberg, "Induktivt effektöverföringssystem med hög verkningsgrad," bachelor's thesis, Chalmers University of Technology, 2013.

# Appendix A

# C Code for the Primary Side

## Main Source File

```
/*
 * main.c
 */

#include "DSP28x_Project.h"
#include "Submodules/ePWM.h"
#include "Submodules/ADC.h"

interrupt void adc_isr();
void ePWM_update(ADC_INFO*);
void current_eval(ADC_INFO*);
void DC_eval(ADC_INFO*);
static void trip_zone();

// Global Variable, defined in adc.c
extern ADC_INFO adc_info;

// Variables in main.c
Uint32 freq;

void main()
{
        // Initialize System Control:
        // PLL, WatchDog, enable Peripheral Clocks
        // This function is found in the DSP2833x_SysCtrl.c file.
        InitSysCtrl();

        // Change the frequency of the hi speed clock
        EALLOW;
        SysCtrlRegs.HISPCP.all = 2;     // HSPCLK = SYSCLKOUT/ADC_MODCLK
        EDIS;

        // Disable CPU interrupts
        DINT;

        // Initialize the PIE control registers to their default state.
        // The default state is all PIE interrupts disabled and flags are cleared.
        // This function is found in the DSP2833x_PieCtrl.c file.
        InitPieCtrl();

        // Disable CPU interrupts and clear all CPU interrupt flags:
        IER = 0x0000;
        IFR = 0x0000;

        // Initialize the PIE vector table with pointers to the shell Interrupt Service Routines (ISR).
        // This will populate the entire table.
        // The shell ISR routines are found in DSP2833x_DefaultIsr.c.
        // This function is found in DSP2833x_PieVect.c.
        InitPieVectTable();

        // Initialize the ADC submodule
        InitAdc();

```

```
54          // Interrupts that are used in this example are re-mapped to ISR functions found within this file.
55          EALLOW;
56          PieVectTable.ADCINT = &adc_isr;                         // The interrupt will come from the adc
57          EDIS;
58
59          // Configure and start the PWM submodule
60          ePWM();
61
62          // Configure and start the ADC submodule
63          ADC();
64
65          // Enable CPU INT1 which is connected to ADC INT:
66          IER |= M_INT1;
67
68          // Enable ADC INTn in the PIE: Group 1 interrupt 6
69          PieCtrlRegs.PIEIER1.bit.INTx6 = 1;
70
71          // Enable Global Interrupts
72          EINT;
73
74          // start of conversion
75          AdcRegs.ADCTRL2.bit.SOC_SEQ1 = 1;
76
77          // Delay the ePWM submodule in order to put out the correct frequency
78          DELAY_US(100000L);
79
80          // Initialize GPIO pins for ePWM1
81          // These functions are in the DSP2833x_EPwm.c file
82          InitEPwm1Gpio();
83
84          // Initialize GPIO pins for ePWM2
85          // These functions are in the DSP2833x_EPwm.c file
86          InitEPwm2Gpio();
87
88          // Keep the MCU busy so it doesn't do anything stupid
89          for(;;);
90  }
91
92
93  interrupt void adc_isr()
94  {
95          // Evaluate current
96          current_eval(&adc_info);
97
98          // Evaluate DC voltage
99          DC_eval(&adc_info);
100
101         // Update the ePWM submodule
102         ePWM_update(&adc_info);
103
104         // Clear INT flag for this timer
105         AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1;
106
107         // Acknowledge this interrupt in order to receive more interrupts from group 1
108         PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
109 }
110
111 void ePWM_update(ADC_INFO *adc_info)
112 {
113         // Save the ADC input to variable
114         adc_info->adc_result1 = AdcRegs.ADCRESULT1>>4;
115
116         // Calculate new frequency for the ePWM
117         freq = 1.0E+4 + 22*adc_info->adc_result1;                       // 90 kHz / 4095 sample \approx 22 Hz/sample, 10kHz + 22
                Hz/sample * #samples
118
119         // ePWM1 reconfiguration
120         EPwm1Regs.TBPRD = (75*1.0E+6)/freq;                                     // System clock = 150MHz, (2D)/150MHz = 1/freq
121
122         // ePWM2 reconfiguration
123         EPwm2Regs.TBPRD = (75*1.0E+6)/freq;                                     // System clock = 150MHz, (2D)/150MHz = 1/freq
124 }
125
126 void current_eval(ADC_INFO *adc_info)
127 {
128         // Save the ADC input to variable
129         adc_info->adc_result2 = AdcRegs.ADCRESULT2>>4;                  // Bit shift ADCRESULT0 4 steps because it's a 12 bit register,
                not 16 bit
130
131         // Evaluate if current is too high or too low
132         if(adc_info->adc_result2 < adc_info->CurrentMax)
133         {
134                 trip_zone();
```

```
135         }
136 }
137
138 void DC_eval(ADC_INFO *adc_info)
139 {
140         // Save the ADC input to variable
141         adc_info->adc_result0 = AdcRegs.ADCRESULT0>>4;                  // Bit shift ADCRESULT0 4 steps because it's a 12 bit register,
                not 16 bit
142
143         // Evaluate if DC voltage is too high
144         if(adc_info->adc_result0 > adc_info->VoltageMax)
145         {
146                 trip_zone();
147         }
148 }
149
150 static void trip_zone()
151 {
152         EALLOW;
153         EPwm1Regs.TZFRC.bit.OST = 1;                                    // Force a one-shot trip-zone interrupt on ePWM1
154         EPwm2Regs.TZFRC.bit.OST = 1;                                    // Force a one-shot trip-zone interrupt on ePWM2
155         EDIS;
156 }
```

# ADC Source File

```
/*
 * ADC.c
 *
 *  Created on: 3 apr 2014
 *      Author: Christian Ekman
 */

#include "DSP28x_Project.h"
#include "ADC.h"

ADC_INFO adc_info;

void ADC()
{
        // Configure the ADC
        AdcRegs.ADCTRL1.bit.ACQ_PS = 0xF;                       // 16 ADC clock cycles window acquisition
        AdcRegs.ADCTRL3.bit.ADCCLKPS = 0;                       // 25 MHz;              ADC module clock = HSPCLK/2*ADC_CKPS   = 25.0MHz
              /(1*2) = 12.5MHz
        AdcRegs.ADCTRL1.bit.SEQ_CASC = 1;               // Cascaded mode
        AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0;            // ADCINA0
        AdcRegs.ADCCHSELSEQ1.bit.CONV01 = 1;            // ADCINA1
        AdcRegs.ADCCHSELSEQ1.bit.CONV02 = 2;            // ADCINA2
        AdcRegs.ADCTRL1.bit.CONT_RUN = 1;               // Continuous mode
        AdcRegs.ADCTRL3.bit.SMODE_SEL = 0;                      // Sequential Mode
        AdcRegs.ADCMAXCONV.bit.MAX_CONV1 = 2;           // # of conversions made

        // Enable interrupt sequencer 1
        AdcRegs.ADCTRL2.bit.INT_ENA_SEQ1 = 1;

        // interrupt after every conversion
        AdcRegs.ADCTRL2.bit.INT_MOD_SEQ1 = 0;

        // Set ADC_info values
        adc_info.AdcRegHandle = &AdcRegs;
        adc_info.adc_result0 = 0;                                       // Initial frequency
        adc_info.adc_result1 = 3000;                            // Initial current
        adc_info.adc_result2 = 2050;                            // Initial DC voltage
        adc_info.CurrentMax = 1600;                                     // 26.6A
        adc_info.CurrentMin = 3190;                                     // 1A
        adc_info.VoltageMax = 3974;                                     //
}
```

# ADC Header File

```
/*
 * ADC.h
 *
 *  Created on: 3 apr 2014
 *      Author: Christian Ekman
 */

#ifndef ADC_H_
#define ADC_H_

void ADC();

typedef struct
{
   volatile struct ADC_REGS *AdcRegHandle;
   Uint32 adc_result0;
   Uint32 adc_result1;
   Uint32 adc_result2;
   Uint16 CurrentMax;
   Uint16 CurrentMin;
   Uint16 VoltageMax;
}ADC_INFO;

#endif /* ADC_H_ */
```

# ePWM Source File

```
/*
 * ePWM.c
 *
 *  Created on: 3 apr 2014
 *      Author: Christian Ekman
 */

#include "DSP28x_Project.h"

void pwm_config(void);

#define deadband 100;                                                    // 0.67us, Deadband time  = deadband/150 MHz

void ePWM()
{
        //      Stop the Time Base Clock
        EALLOW;
        SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0;
        EDIS;

        //      Configure the ePWM1Regs
        pwm_config();

        //      Enable the Time Base Clock
        EALLOW;
        SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;
        EDIS;
}

void pwm_config()
{
        // Setup Time Base Clock ePWM1
        EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;                   // Count up/down
        EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;                          // Disable phase loading
        EPwm1Regs.TBCTL.bit.SYNCOSEL = TB_CTR_ZERO;                      // Synchronize with EPWMSYNCI
        EPwm1Regs.TBPHS.half.TBPHS = 0x0000;                             // Phase is 0
        EPwm1Regs.TBCTR = 0x0000;                                        // Clear counter
        EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;                         // Clock ratio to SYSCLKOUT
        EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;

        // Setup Time Base Clock ePWM2
        EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;                   // Count up/down
        EPwm2Regs.TBCTL.bit.PHSEN = TB_ENABLE;                           // Enable phase loading
        EPwm2Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_DISABLE;                  // Synchronize with EPWMSYNCI
        EPwm2Regs.TBPHS.half.TBPHS = 0x0000;                             // Phase is 0
        EPwm2Regs.TBCTR = 0x0000;                                                // Clear counter
        EPwm2Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;                         // Clock ratio to SYSCLKOUT
        EPwm2Regs.TBCTL.bit.CLKDIV = TB_DIV1;

        EALLOW;                                                                                 // Trip-zone registers are EALLOW protected

        // Setup trip-zone ePWM1
        EPwm1Regs.TZSEL.all = 0x0000;                                    // Disable trip-zone pins as interrupt sources for ePWM1
        EPwm1Regs.TZEINT.bit.OST = 1;                                    // Enable one-shot interrupt generation of ePWM1
        EPwm1Regs.TZCTL.bit.TZA = 2;                                     // Force ePWM1a to low state when tripped
        EPwm1Regs.TZCTL.bit.TZB = 2;                                     // Force ePWM1b to low state when tripped

        // Setup trip-zone ePWM2
        EPwm2Regs.TZSEL.all = 0x0000;                                    // Disable trip-zone pins as interrupt sources for ePWM2
        EPwm2Regs.TZEINT.bit.OST = 1;                                    // Enable one-shot interrupt generation of ePWM2
        EPwm2Regs.TZCTL.bit.TZA = 2;                                     // Force ePWM2a to low state when tripped
        EPwm2Regs.TZCTL.bit.TZB = 2;                                     // Force ePWM2b to low state when tripped

        EDIS;                                                                                   // End of writing to EALLOW protected registers

        // Setup shadow register load on ZERO ePWM1
        EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;                      // Enable shadow register of CMPA
        EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;                      // Enable shadow register of CMPB
        EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;                    // Load CMPA into shadow register at CTR = zero
        EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;                    // Load CMPB into shadow register at CTR = zero
```

```
71
72          // Setup shadow register load on ZERO ePWM2
73          EPwm2Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;                    // Enable shadow register of CMPA
74          EPwm2Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;                    // Enable shadow register of CMPB
75          EPwm2Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;                  // Load CMPA into shadow register at CTR = zero
76          EPwm2Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;                  // Load CMPB into shadow register at CTR = zero
77
78          // Setup deadband ePWM1
79          EPwm1Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;                 // Both rising and falling edge deadband enabled
80          EPwm1Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;                      // Active Hi Complementary
81          EPwm1Regs.DBCTL.bit.IN_MODE = DBA_RED_DBB_FED;                 // ePWM1a rising edge, ePWM1b falling edge
82          EPwm1Regs.DBRED = deadband;                                    // rising edge time = deadband
83          EPwm1Regs.DBFED = deadband;                                    // falling edge time = deadband
84
85          // Setup deadband ePWM2
86          EPwm2Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;                 // Both rising and falling edge deadband enabled
87          EPwm2Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;                      // Active Hi Complementary
88          EPwm2Regs.DBCTL.bit.IN_MODE = DBB_RED_DBA_FED;                 // ePWM2b rising edge, ePWM2a falling edge
89          EPwm2Regs.DBRED = deadband;                                    // rising edge time = deadband
90          EPwm2Regs.DBFED = deadband;                                    // falling edge time = deadband
91
92          // Set actions ePWM1
93          EPwm1Regs.AQCTLA.bit.ZRO = AQ_SET;                             // Set ePWM1a at CTR = ZRO, down count
94          EPwm1Regs.AQCTLA.bit.PRD = AQ_CLEAR;                           // Clear ePWM1a at CTR = PRD, up count
95          EPwm1Regs.AQCTLB.bit.ZRO = AQ_CLEAR;                           // Clear ePWM1b at CTR = ZRO, down count
96          EPwm1Regs.AQCTLB.bit.PRD = AQ_SET;                             // Set ePWM1b at CTR = PRD, up count
97
98          // Set actions ePWM2
99          EPwm2Regs.AQCTLA.bit.ZRO = AQ_CLEAR;                           // Clear ePWM2a at CTR = ZRO, up count
100         EPwm2Regs.AQCTLA.bit.PRD = AQ_SET;                             // Set ePWM2a at CTR = PRD, down count
101         EPwm2Regs.AQCTLB.bit.ZRO = AQ_SET;                             // Set ePWM2b at CTR = ZRO, down count
102         EPwm2Regs.AQCTLB.bit.PRD = AQ_CLEAR;                           // Clear ePWM2b at CTR = PRD, up count
103 }
```

# ePWM Header File

```
/*
 * ePWM.h
 *
 *  Created on: 3 apr 2014
 *      Author: Christian Ekman
 */

#ifndef EPWM_H_
#define EPWM_H_

void ePWM();

#endif /* EPWM_H_ */
```

# Appendix B

# C Code for the Secondary Side

## Main Source File

```
/*
 * main.c
 */

// Includes
#include "DSP28x_Project.h"
#include "Submodules/eCAP.h"
#include "Submodules/ePWM.h"
#include "Submodules/ADC.h"
#include "Submodules/SPI.h"
#include "Submodules/nRF24L01+.h"
#include "General/f2833xbmsk.h"

// Functions defined in main.c
interrupt void eCAP_isr();
interrupt void adc_isr();
interrupt void spiRxFifo_Isr();
interrupt void spiTxFifo_Isr();
interrupt void IRQ_isr();
static void ePWM_update();
static void current_eval();
static void DC_eval();
static void trip_zone();
static void clear_trip_zone();
static void enable_TX();
void disable_TX();

#define ComPrimSideCtrMax       1000000                         // Max value of ComPrimSideCtr
#define TransmitCtrMax          10000000                        // Max value of TransmitCtr

// Global variables declared in other source files
extern ADC_INFO adc_info;
extern Uint16 rdata;

// Global variables declared in this source file
Uint16 Init_nRF24L01_ctr = 0;

// nRF variables
static Uint16 gCommandSent = 0;                                 // Determines what stage in RX/TX the program is in
// nRF_action: 0 = transmit, 1 = receive, 2 = flush TX, 3 = flush RX, 4 = reuse TX payload
Uint16 nRF_action;                                                      // Determines what happens during TX/RX int
Uint16 gnRF_TX;                                                         // What's going in the TX buffer
Uint32 ComPrimSideCtr = 0;                                      // Counter for trying to contact the primary side
Uint32 TransmitCtr = 0;                                         // Counter for time since trying to transmit without interrupt
Uint16 PreviousTX;                                                      // TX from last interrupt

static Uint16 shutdown = 0;                                     // 0 = do not shut down, 1 = shut down

void main()
{
        // Initialize System Control:
        // PLL, WatchDog, enable Peripheral Clocks
        // This function is found in the DSP2833x_SysCtrl.c file.
```

```
54          InitSysCtrl();
55
56          // Change the frequency of the hi speed clock
57          EALLOW;
58          SysCtrlRegs.HISPCP.all = 2;     // HSPCLK = SYSCLKOUT/ADC_MODCLK
59          EDIS;
60
61          // Initialize GPIO06, pin 13, for EPWMSYNCI
62          // These functions are in the DSP2833x_EPwm.c file
63          InitEPwmSyncGpio();
64
65          // Initialize GPIO05, pin 12, for eCAP1
66          // These functions are in the DSP2833x_ECap.c file
67          InitECap1Gpio();
68
69          // Initialize GPIO54 to GPIO57 for SPI-A
70          // These functions are in the DSP2833x_Spi.c file
71          InitSpiaGpio();
72
73          // Initialize GPIO pins for extra SPI features
74          EALLOW;
75          // Initialize GPIO59 for SPI-A pin CE
76          GpioCtrlRegs.GPBMUX2.bit.GPIO59 = 0;                          // Set as GPIO pin
77          GpioCtrlRegs.GPBDIR.bit.GPIO59 = 1;                                  // Set as output
78          GpioCtrlRegs.GPBPUD.bit.GPIO59 = 1;                                  // Disable pull-up resistance
79
80          // Initialize GPIO58 for SPI-A pin IRQ
81          GpioCtrlRegs.GPBMUX2.bit.GPIO58 = 0;                          // Set as GPIO pin
82          GpioCtrlRegs.GPBDIR.bit.GPIO58 = 0;                                  // Set as input
83          GpioCtrlRegs.GPBQSEL2.bit.GPIO58 = 0;                         // Synchronize with SYSCLKOUT
84          GpioCtrlRegs.GPBPUD.bit.GPIO58 = 0;                                  // Enable pull-up resistance
85          GpioIntRegs.GPIOXINT3SEL.bit.GPIOSEL = 0x1A;           // XINT3 from GPIO58 - IRQ
86          EDIS;
87
88          // Configure XINT3
89          XIntruptRegs.XINT3CR.bit.ENABLE = 1;                          // Enable interrupt XINT3
90          XIntruptRegs.XINT3CR.bit.POLARITY = 0;                        // Interrupt generated on falling edge
91
92          // Disable CPU interrupts
93          DINT;
94
95          // Initialize the PIE control registers to their default state.
96          // The default state is all PIE interrupts disabled and flags are cleared.
97          // This function is found in the DSP2833x_PieCtrl.c file.
98          InitPieCtrl();
99
100         // Disable CPU interrupts and clear all CPU interrupt flags:
101         IER = 0x0000;
102         IFR = 0x0000;
103
104         // Initialize the PIE vector table with pointers to the shell Interrupt Service Routines (ISR).
105         // This will populate the entire table.
106         // The shell ISR routines are found in DSP2833x_DefaultIsr.c.
107         // This function is found in DSP2833x_PieVect.c.
108         InitPieVectTable();
109
110         // Initialize the ADC submodule
111         InitAdc();
112
113         // Interrupts that are used in this example are re-mapped to ISR functions found within this file.
114         EALLOW;
115         PieVectTable.ECAP1_INT = &eCAP_isr;                                  // Enable interrupts from the eCAP
116         PieVectTable.ADCINT = &adc_isr;                                      // Enable Interrupts from the ADC
117         PieVectTable.SPIRXINTA = &spiRxFifo_Isr;                      // Enable Interrupts from FIFO receiving SPI-A
118         PieVectTable.SPITXINTA = &spiTxFifo_Isr;                      // Enable interrupts from FIFO transmitting SPI-A
119         PieVectTable.XINT3 = &IRQ_isr;                                       // Enable interrupts from IRQ
120         EDIS;
121
122         // Configure and start the eCAP submodule
123         eCAP();
124
125         // Configure and start the ADC submodule
126         ADC();
127
128         // Configure and start the ePWM submodule
129         ePWM();
130
131         // Configure the SPI-A submodule
132         SPI();
133
134         // Enable CPU INT4 which is connected to eCAP1 INT:
135         IER |= M_INT4;
136         // Enable CPU INT1 which is connected to ADC INT:
```

```
137         IER |= M_INT1;
138         // Enable CPU INT6 which is connected to SPI-A INT
139         IER |= M_INT6;
140         // Enable CPU INT6 which is connected to XINT3
141         IER |= M_INT12;
142
143         // Enable eCAP INTn in the PIE: Group 4 interrupt 1
144         PieCtrlRegs.PIEIER4.bit.INTx1 = 1;
145         // Enable ADC INTn in the PIE: Group 1 interrupt 6
146         PieCtrlRegs.PIEIER1.bit.INTx6 = 1;
147         // Enable SPIRXINTA in the PIE: Group 6 interrupt 1
148         PieCtrlRegs.PIEIER6.bit.INTx1 = 1;
149         // Enable SPITXINTA in the PIE: Group 6 interrupt 2
150         PieCtrlRegs.PIEIER6.bit.INTx2 = 1;
151         // Enable XINT3 in the PIE: Group 12 interrupt 1
152         PieCtrlRegs.PIEIER12.bit.INTx1 = 1;
153
154         // Enable Global Interrupts
155         EINT;
156
157         // start of conversion
158         AdcRegs.ADCTRL2.bit.SOC_SEQ1 = 1;
159
160         // Delay the ePWM submodule in order to put out the correct frequency
161         DELAY_US(1000000L);
162
163         // Initialize GPIO pins for ePWM1
164         // These functions are in the DSP2833x_EPwm.c file
165         // Here instead of earlier to introduce delay
166         // to ePWM in order to start with correct frequency and phase
167         InitEPwm1Gpio();
168
169         // Initialize GPIO pins for ePWM2
170         // These functions are in the DSP2833x_EPwm.c file
171         // Here instead of earlier to introduce delay
172         // to ePWM in order to start with correct frequency and phase
173         InitEPwm2Gpio();
174
175         // Keep the MCU busy so it doesn't do anything stupid
176         for(;;);
177 }
178
179 interrupt void eCAP_isr()
180 {
181         // Update the configuration of the ePWM submodule
182         ePWM_update();
183
184         // Reset interrupt settings
185         ECap1Regs.ECCLR.bit.CEVT4 = 1;                                  // Interrupt after 4 events
186         ECap1Regs.ECCLR.bit.INT = 1;                                    // Clear interrupts
187         ECap1Regs.ECCTL2.bit.REARM = EC_ARM;                          // Re-arm the eCAP one-shot
188
189         // Acknowledge further interrupts from group 4
190         PieCtrlRegs.PIEACK.all |= PIEACK_GROUP4;
191 }
192
193 interrupt void adc_isr()
194 {
195         // Evaluate current
196         current_eval(&adc_info);
197
198         // Evaluate DC voltage
199         DC_eval(&adc_info);
200
201         // Clear INT flag for this timer
202         AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1;
203
204         // Acknowledge further interrupts from group 1
205         PieCtrlRegs.PIEACK.all |= PIEACK_GROUP1;
206 }
207
208 interrupt void spiTxFifo_Isr()
209 {
210         if(Init_nRF24L01_ctr < 33)
211         {
212                 // Set CE pin low
213                 GpioDataRegs.GPBCLEAR.bit.GPIO59 = 1;
214
215                 Init_nRF24L01(Init_nRF24L01_ctr);       // Continue initializing radio
216
217                 Init_nRF24L01_ctr++;                            // increment counter
218
219                 // Clear Interrupt flag
```

```
220                 SpiaRegs.SPIFFTX.bit.TXFFINTCLR=1;
221
222                 // Acknowledge further interrupts from group 6
223                 PieCtrlRegs.PIEACK.all |= PIEACK_GROUP6;
224                 return;
225         }
226
227         switch(nRF_action)
228         {
229         case 0:                                                                                 // Action =
              transmit
230                 if(gCommandSent < 4)
231                 {
232                         Transmit_nRF24L01(gnRF_TX, &gCommandSent);
233                 }
234                 // All registers set up for transmission
235                 else if(gCommandSent == 4)
236                 {
237                         GpioDataRegs.GPBSET.bit.GPIO59 = 1;                             // Set CE high to enable TX
238                         gCommandSent = (gCommandSent + 1) % 8;                  // Increment value every interrupt
239                 }
240                 // if transmission is not yet finished
241                 else if(gCommandSent == 5 && PieCtrlRegs.PIEIFR12.bit.INTx1 == 0)
242                 {
243                         GpioDataRegs.GPBCLEAR.bit.GPIO59 = 1;                   // Disable RX/TX mode by setting CE low
244                         //  If TX payload hasn't been received after TransmitCtrMax interrupts
245                         if(TransmitCtr == (Uint32)TransmitCtrMax)
246                         {
247                                 SpiaRegs.SPITXBUF = (Uint16)REUSE_TX_PL << 8;   // Re-use previous TX payload
248                                 GpioDataRegs.GPBSET.bit.GPIO59 = 1;             // Set CE high to enable TX
249                         }
250                         TransmitCtr = (TransmitCtr + 1) % (Uint32)TransmitCtrMax;
251                 }
252                 // If transmission is finished
253                 else if(gCommandSent == 5 && PieCtrlRegs.PIEIFR12.bit.INTx1 == 1)
254                 {
255                         Transmit_nRF24L01(gnRF_TX, &gCommandSent);
256                 }
257                 else if(gCommandSent == 6)
258                 {
259                         Transmit_nRF24L01(gnRF_TX, &gCommandSent);
260                 }
261                 else if(gCommandSent == 7)
262                 {
263                         PieCtrlRegs.PIEACK.all |= PIEACK_GROUP12;               // Acknowledge further interrupts from group 12
264                         gCommandSent = (gCommandSent + 1) % 8;                  // Increment value every interrupt
265                         nRF_action = 2;                                                                 // Action = Flush TX
266                 }
267                 break;
268         case 1:                                                                                 // Action = Receive
269                 if(gCommandSent < 4)
270                 {
271                         Receive_nRF24L01(&gCommandSent);
272                 }
273                 else if(gCommandSent == 4)
274                 {
275                         GpioDataRegs.GPBSET.bit.GPIO59 = 1;                     // Set CE high to enable TX
276                         *CommandSent = (*CommandSent + 1) % 6;          // Increment value every case
277                 }
278                 break;
279         case 2:                                                                                 // Action = Flush TX
280                 flushTX_nRF24L01(gnRF_TX);
281                 nRF_action = 1;                                                                 // Action = Receive
282                 break;
283         default:
284                 break;
285         }
286
287         // Clear Interrupt flag
288         SpiaRegs.SPIFFTX.bit.TXFFINTCLR=1;
289
290         // Acknowledge further interrupts from group 6
291         PieCtrlRegs.PIEACK.all |= PIEACK_GROUP6;
292 }
293
294 interrupt void spiRxFifo_Isr()
295 {
296         switch(nRF_action)
297         {
298         case 1:
299                 if(gCommandSent == 5 && PieCtrlRegs.PIEIFR12.bit.INTx1 == 1)
300                 {
301                         rdata = SpiaRegs.SPIRXBUF;                                      // Read receive data
```

```
302                         *CommandSent = (*CommandSent + 1) % 6;                  // Increment value every case
303                 }
304                 break;
305         default:
306                 break;
307         }
308
309         // If primary side is trying to start charging and the current is sufficiently high
310         if(rdata == 0x6666 && shutdown == 0 && adc_info.adc_result0 < adc_info.ChargeCurrLim)
311         {
312                 clear_trip_zone();
313         }
314         // Unknown message
315         else;
316
317         // Clear Overflow flag
318         SpiaRegs.SPIFFRX.bit.RXFFOVFCLR=1;
319
320         // Clear Interrupt flag
321         SpiaRegs.SPIFFRX.bit.RXFFINTCLR=1;
322         // Acknowledge further interrupts from group 6
323         PieCtrlRegs.PIEACK.all |= PIEACK_GROUP6;
324 }
325
326 interrupt void IRQ_isr()
327 {
328         GpioDataRegs.GPBCLEAR.bit.GPIO59 = 1;                   // Disable RX/TX mode by setting CE low
329
330         TransmitCtr = 0;                                                                        // Set counter to zero for next transmit attempt
331
332         enable_TX();
333 }
334
335 static void ePWM_update()
336 {
337         Uint32 t1;                                                                                      // Time of first rising edge
338         Uint32 t2;                                                                                      // Time of first falling edge
339         Uint32 t3;                                                                                      // Time of second rising edge
340         Uint32 Period;                                                                          // Period time, 1/freq
341         Uint32 Duty;                                                                            // Duty cycle
342
343         // Save time stamps to variables
344         t1 = ECap1Regs.CAP1;
345         t2 = ECap1Regs.CAP2;
346         t3 = ECap1Regs.CAP3;
347
348         // Calculate period time and duty cycle
349         Period = (t3-t1)/2;                                                                     // divided by 2 because of updown
                counter mode
350         Duty = (t2-t1)/2;                                                                       // divided by 2 because of updown
                counter mode
351
352         // Set frequency of ePWM
353         EPwm1Regs.TBPRD = Period & 0xFFFF;                                      // ((t3-t1)/2)*150MHz, & 0xFFFF 32-bit to 16-bit
                register
354         EPwm2Regs.TBPRD = Period & 0xFFFF;                                      // ((t3-t1)/2)*150MHz
355
356         // Set Compare values ePWM1
357         EPwm1Regs.CMPA.half.CMPA = Duty & 0xFFFF;                       // Set compare A value, ((t2-t1)/2)*150MHz
358         EPwm1Regs.CMPB = Duty & 0xFFFF;                                         // Set Compare B value
359
360         // Set compare values ePWM2
361         EPwm2Regs.CMPA.half.CMPA = Duty & 0xFFFF;                       // Set compare A value, ((t2-t1)/2)*150Mhz
362         EPwm2Regs.CMPB = Duty & 0xFFFF;                                         // Set compare B value
363 }
364
365 static void current_eval(ADC_INFO *adc_info)
366 {
367         // Save the ADC input to variable
368         adc_info->adc_result0 = AdcRegs.ADCRESULT0>>4;          // Bit shift ADCRESULT0 4 steps because it's a 12 bit register, not 16
                bit
369
370         // Evaluate if current is too high
371         if(adc_info->adc_result0 < adc_info->CurrentMax && shutdown == 0)
372         {
373                 trip_zone();
374                 //              shutdown = 1;
375                 gnRF_TX = 0xEEEE;                                                               // Set transmit data
376                 nRF_action = 0;                                                                 // Set nRF to transmit
377                 enable_TX();                                                                    // Enable nRF transmit
378                 return;
379         }
380 }
```

```
381
382 static void DC_eval(ADC_INFO *adc_info)
383 {
384         // Save the ADC input to variable
385         adc_info->adc_result1 = AdcRegs.ADCRESULT1>>4;          // Bit shift ADCRESULT1 4 steps because it's a 12 bit register, not 16
                bit
386
387         // Evaluate if DC voltage is too high
388         if(adc_info->adc_result1 > adc_info->VoltageMax && shutdown == 0)
389         {
390                 trip_zone();
391                 //              shutdown = 1;
392                 gnRF_TX = 0xEEEE;                                                       // Set transmit data
393                 nRF_action = 0;                                                         // Set nRF to transmit
394                 enable_TX();                                                            // Enable nRF transmit
395                 return;
396         }
397         // If the charging current is too low and both trip-zone and shutdown is off
398         else if(adc_info->adc_result0 > adc_info->CurrentMin && EPwm1Regs.TZFLG.bit.OST == 0 && shutdown == 0)
399         {
400                 gnRF_TX = 0x8888;                                                       // Set transmit data
401                 trip_zone();                                                            // Shut down ePWM
402                 if(gnRF_TX == PreviousTX)
403                 {
404                         return;
405                 }
406
407                 nRF_action = 0;                                                         // Set nRF to transmit
408                 enable_TX();                                                            // Enable nRF transmit
409                 PreviousTX = gnRF_TX;                                           // So it knows this is not a new event
410                 return;
411         }
412         // if trip-zone is on and the battery is not fully charged
413         else if(adc_info->adc_result1 < adc_info->ChargeVoltLim && EPwm1Regs.TZFLG.bit.OST == 1 && shutdown == 0)
414         {
415                 gnRF_TX = 0x5555;                                                       // Set transmit data
416                 if(GpioDataRegs.GPBCLEAR.bit.GPIO59 == 1)               // Communication off
417                 {
418                         switch(ComPrimSideCtr)                                  // Communication primary side counter
419                         {
420                         case 0:
421                                 nRF_action = 0;                                         // Set nRF to transmit
422                                 enable_TX();                                            // Enable nRF transmit
423                                 break;
424                         default:
425                                 break;
426                         }
427                         ComPrimSideCtr = (ComPrimSideCtr + 1) % ComPrimSideCtrMax;
428                 }
429
430                 return;
431         }
432         // if trip-zone is on and the battery is already charged
433         else if(adc_info->adc_result1 >= adc_info->ChargeVoltLim && EPwm1Regs.TZFLG.bit.OST == 1 && shutdown == 0)
434         {
435                 gnRF_TX = 0xA0A0;                                                       // Set transmit data
436                 if(gnRF_TX == PreviousTX)
437                 {
438                         return;
439                 }
440
441                 nRF_action = 0;                                                         // Set nRF to transmit
442                 enable_TX();                                                            // Enable nRF to transmit
443                 PreviousTX = gnRF_TX;                                           // So it knows this is not a new event
444                 return;
445         }
446         // The charger is charging as it is supposed to
447         else if(shutdown == 0)
448         {
449                 gnRF_TX = 0x3333;                                                       // Set transmit data
450                 if(gnRF_TX == PreviousTX)
451                 {
452                         return;
453                 }
454
455                 nRF_action = 0;                                                         // Set nRF to transmit
456                 enable_TX();                                                            // Enable nRF transmit
457                 PreviousTX = gnRF_TX;                                           // So it knows this is not a new event
458                 return;
459         }
460 }
461
462 static void trip_zone()
```

```
463 {
464         EALLOW;
465         EPwm1Regs.TZFRC.bit.OST = 1;                                    // Force one-shot trip-zone interrupt on ePWM1
466         EPwm2Regs.TZFRC.bit.OST = 1;                                    // Force one-shot trip-zone interrupt on ePWM2
467         EDIS;
468 }
469
470 static void clear_trip_zone()
471 {
472         EALLOW;
473         EPwm1Regs.TZCLR.bit.OST = 1;                                    // Clear the trip-zone on ePWM1
474         EPwm2Regs.TZCLR.bit.OST = 1;                                    // Clear the trip-zone on ePWM2
475         EPwm1Regs.TZCLR.bit.INT = 1;                                    // Clear global interrupt flag on ePWM1
476         EPwm2Regs.TZCLR.bit.INT = 1;                                    // Clear global interrupt flag on ePWM2
477         EDIS;
478 }
479
480 static void enable_TX()
481 {
482         SpiaRegs.SPICTL.bit.SPIINTENA = 1;
483         SpiaRegs.SPICTL.bit.TALK = 1;                                   // Enable MOSI pin
484 }
485
486 void disable_TX()
487 {
488         SpiaRegs.SPICTL.bit.TALK = 0;                                   // Disable MOSI pin
489         SpiaRegs.SPICTL.bit.SPIINTENA = 0;
490 }
```

# Main Header File

```
1  /*
2   * main.h
3   *
4   *  Created on: 28 apr 2014
5   *      Author: Christian Ekman
6   */
7
8  #ifndef MAIN_H_
9  #define MAIN_H_
10
11 void disable_TX();
12 void enable_TX();
13
14 #endif /* MAIN_H_ */
```

# ADC Source File

```
/*
 * ADC.c
 *
 *  Created on: 23 apr 2014
 *      Author: Christian Ekman
 */

#include "DSP28x_Project.h"
#include "ADC.h"

ADC_INFO adc_info;

void ADC()
{
        // Configure the ADC
        AdcRegs.ADCTRL1.bit.ACQ_PS = 0xF;                       // 16 ADC clock cycles window acquisition
        AdcRegs.ADCTRL3.bit.ADCCLKPS = 0;                       // 25 MHz;              ADC module clock = HSPCLK/2*ADC_CKPS   = 25.0MHz
              /(1*2) = 12.5MHz
        AdcRegs.ADCTRL1.bit.SEQ_CASC = 1;               // Cascaded mode
        AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0;            // ADCINA0
        AdcRegs.ADCCHSELSEQ1.bit.CONV01 = 4;            // ADCINA7
        AdcRegs.ADCTRL1.bit.CONT_RUN = 1;               // Continuous mode
        AdcRegs.ADCTRL3.bit.SMODE_SEL = 0;                      // Sequential Mode
        AdcRegs.ADCMAXCONV.bit.MAX_CONV1 = 1;           // # of conversions made

        // Enable interrupt sequencer 1
        AdcRegs.ADCTRL2.bit.INT_ENA_SEQ1 = 1;

        // interrupt after every other conversion
        AdcRegs.ADCTRL2.bit.INT_MOD_SEQ1 = 0;

        // Set ADC_info values
        adc_info.AdcRegHandle = &AdcRegs;                       // Set pointer
        adc_info.adc_result0 = 2950;                            // Set initial current
        adc_info.adc_result1 = 2900;                            // Set initial DC voltage
        adc_info.CurrentMax = 2693;                                     // Set maximum current allowed before shutdown occurs
        adc_info.CurrentMin = 3190;                                     // Set minimum current allowed before charging is turned off
        adc_info.VoltageMax = 3974;                                     // Set maximum DC voltage allowed before shutdown occurs
        adc_info.ChargeCurrLim = 3190;                          // Set minimum current for charging
        adc_info.ChargeVoltLim = 3040;                          // Set voltage limit for charging
}
```

# ADC Header File

```
/*
 * ADC.h
 *
 *  Created on: 23 apr 2014
 *      Author: Christian Ekman
 */

#ifndef ADC_H_
#define ADC_H_

void ADC();

typedef struct
{
   volatile struct ADC_REGS *AdcRegHandle;
   Uint32 adc_result0;
   Uint32 adc_result1;
   Uint16 CurrentMax;
   Uint16 CurrentMin;
   Uint16 VoltageMax;
   Uint16 ChargeVoltLim;
   Uint16 ChargeCurrLim;
}ADC_INFO;

#endif /* ADC_H_ */
```

# eCAP Source File

```
/*
 * eCAP.c
 *
 *  Created on: 23 apr 2014
 *      Author: Christian Ekman
 */

#include "DSP28x_Project.h"
#include "../General/f2833xbmsk.h"
//#include "DSP2833x_ECap_defines.h"

static void eCAP_config();

void eCAP()
{
        // Stop eCAP counter
        ECap1Regs.ECEINT.all = 0x0000;                                  // Disable all capture interrupts
        ECap1Regs.ECCLR.all = 0xFFFF;                                   // Clear all CAP interrupt flags
        ECap1Regs.ECCTL1.bit.CAPLDEN = CAPLDEN_DISABLE;                 // Disable CAP1-CAP4 register loads
        ECap1Regs.ECCTL2.bit.TSCTRSTOP = EC_FREEZE;                     // Make sure the counter is stopped

        // Configure the eCAP
        eCAP_config();

        // Start eCAP counter
        ECap1Regs.ECCTL2.bit.TSCTRSTOP = EC_RUN;                        // Start Counter
        ECap1Regs.ECCTL2.bit.REARM = EC_ARM;                            // arm one-shot
        ECap1Regs.ECCTL1.bit.CAPLDEN = CAPLDEN_ENABLE;                  // Enable CAP1-CAP4 register loads
        ECap1Regs.ECEINT.bit.CEVT4 = 1;                                 // Interrupt after 4 events
}

static void eCAP_config()
{
        // Configure peripheral registers
        ECap1Regs.ECCTL2.bit.CONT_ONESHT = ONE_SHOT_MODE;               // One-shot
        ECap1Regs.ECCTL2.bit.STOP_WRAP = ONESHT_CAP_EV4;                // Stop at 4 events
        ECap1Regs.ECCTL1.bit.CAP1POL = CAP1POL_RISING_EDGE;             // Rising edge
        ECap1Regs.ECCTL1.bit.CAP2POL = CAP2POL_FALLING_EDGE;        // Falling edge
        ECap1Regs.ECCTL1.bit.CAP3POL = CAP3POL_RISING_EDGE;             // Rising edge
        ECap1Regs.ECCTL1.bit.CAP4POL = CAP4POL_FALLING_EDGE;        // Falling edge
        ECap1Regs.ECCTL1.bit.CTRRST1 = CTRRST1_ABSOLUTE_TS;             // Do not reset counter at t1
        ECap1Regs.ECCTL1.bit.CTRRST2 = CTRRST2_ABSOLUTE_TS;             // Do not reset counter at t2
        ECap1Regs.ECCTL1.bit.CTRRST3 = CTRRST3_ABSOLUTE_TS;             // Do not reset counter at t3
        ECap1Regs.ECCTL1.bit.CTRRST4 = CTRRST4_DIFFERENCE_TS;       // Do reset counter at t4
        ECap1Regs.ECCTL2.bit.SYNCI_EN = SYNCI_DISABLE;                  // Disable sync in
        ECap1Regs.ECCTL2.bit.SYNCO_SEL = SYNCO_DISABLE;                 // Pass through
        ECap1Regs.ECCTL1.bit.PRESCALE = EC_DIV1;                        // SYSCLK/1
}
```

# eCAP Header File

```
/*
 * eCAP.h
 *
 *  Created on: 23 apr 2014
 *      Author: Christian Ekman
 */

#ifndef ECAP_H_
#define ECAP_H_

void eCAP();

#endif /* ECAP_H_ */
```

# ePWM Source File

```
/*
 * ePWM.c
 *
 *  Created on: 23 apr 2014
 *      Author: Christian Ekman
 */

#include "DSP28x_Project.h"

static void ePWM_config();

#define deadband 76;                                                    // 0.5us, Deadband time  = deadband*6.6e-9

void ePWM()
{
        // Stop the Time Base Clock
        EALLOW;
        SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0;                          // Disable time base clock synchronization
        EDIS;

        // Configure the ePWM
        ePWM_config();

        // Enable the Time Base Clock
        EALLOW;
        SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;                          // Enable time base clock synchronization
        EDIS;
}

static void ePWM_config()
{
        // Setup Time Base Clock ePWM1
        EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;                  // Count up/down
        EPwm1Regs.TBCTL.bit.PHSEN = TB_ENABLE;                          // Enable phase loading
        EPwm1Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;                      // Synchronize with EPWMSYNCI
        EPwm1Regs.TBPHS.half.TBPHS = 0x0000;                            // Phase is 0
        EPwm1Regs.TBCTL.bit.PHSDIR = 1;                                 // Count up after sync
        EPwm1Regs.TBCTR = 0x0000;                                       // Clear counter
        EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;                        // Clock ratio to SYSCLKOUT
        EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;

        // Setup Time Base Clock ePWM2
        EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;                  // Count up/down
        EPwm2Regs.TBPRD = (75*1.0E+6)/(50*1.0E+3);                      // Set initial time base period (50 kHz output) (2D)/SYSCLKOUT = 1/freq
        EPwm2Regs.TBCTL.bit.PHSEN = TB_ENABLE;                          // Enable phase loading
        EPwm2Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_DISABLE;                 // Synchronize with EPWMSYNCI
        EPwm2Regs.TBPHS.half.TBPHS = 0x0000;                            // Phase is 0
        EPwm2Regs.TBCTL.bit.PHSDIR = 1;                                 // Count up after sync
        EPwm2Regs.TBCTR = 0x0000;                                       // Clear counter
        EPwm2Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;                        // Clock ratio to SYSCLKOUT
        EPwm2Regs.TBCTL.bit.CLKDIV = TB_DIV1;

        EALLOW;                                                         // Trip-zone registers are EALLOW protected

        // Setup trip-zone ePWM1
        EPwm1Regs.TZSEL.all = 0x0000;                                   // Disable trip-zone pins as interrupt sources for ePWM1
        EPwm1Regs.TZEINT.bit.OST = TZ_ENABLE;                           // Enable one-shot interrupt generation of ePWM1
        EPwm1Regs.TZCTL.bit.TZA = TZ_FORCE_LO;                          // Force ePWM1a to low state when tripped
        EPwm1Regs.TZCTL.bit.TZB = TZ_FORCE_LO;                          // Force ePWM1b to low state when tripped

        // Setup trip-zone ePWM2
        EPwm2Regs.TZSEL.all = 0x0000;                                   // Disable trip-zone pins as interrupt sources for ePWM2
        EPwm2Regs.TZEINT.bit.OST = TZ_ENABLE;                           // Enable one-shot interrupt generation of ePWM2
        EPwm2Regs.TZCTL.bit.TZA = TZ_FORCE_LO;                          // Force ePWM2a to low state when tripped
        EPwm2Regs.TZCTL.bit.TZB = TZ_FORCE_LO;                          // Force ePWM2b to low state when tripped

        EDIS;                                                           // End of writing to EALLOW protected registers

        // Setup shadow register load on ZERO ePWM1
        EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;                     // Enable shadow register of CMPA
        EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;                     // Enable shadow register of CMPB
        EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;                   // Load CMPA into shadow register at CTR = zero
        EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;                   // Load CMPB into shadow register at CTR = zero

        // Setup shadow register load on ZERO ePWM2
```

```
76          EPwm2Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;                 // Enable shadow register of CMPA
77          EPwm2Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;                 // Enable shadow register of CMPB
78          EPwm2Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;               // Load CMPA into shadow register at CTR = zero
79          EPwm2Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;               // Load CMPB into shadow register at CTR = zero
80
81          // Setup deadband ePWM1
82          EPwm1Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;              // Both rising and falling edge deadband enabled
83          EPwm1Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;                   // Active Hi Complementary
84          EPwm1Regs.DBCTL.bit.IN_MODE = DBA_RED_DBB_FED;              // ePWM1a rising edge, ePWM1b falling edge
85          EPwm1Regs.DBRED = deadband;                                 // rising edge time = deadband
86          EPwm1Regs.DBFED = deadband;                                 // falling edge time = deadband
87
88          // Setup deadband ePWM2
89          EPwm2Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;              // Both rising and falling edge deadband enabled
90          EPwm2Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;                   // Active Hi Complementary
91          EPwm2Regs.DBCTL.bit.IN_MODE = DBB_RED_DBA_FED;              // ePWM2b rising edge, ePWM2a falling edge
92          EPwm2Regs.DBRED = deadband;                                 // rising edge time = deadband
93          EPwm2Regs.DBFED = deadband;                                 // falling edge time = deadband
94
95          // Set Compare values ePWM1
96  //      EPwm1Regs.CMPA.half.CMPA = EPwm1Regs.TBPRD/2;               // Set initial compare A value
97  //      EPwm1Regs.CMPB = EPwm1Regs.TBPRD/2;                         // Set initial compare B value
98
99          // Set compare values ePWM2
100         EPwm2Regs.CMPA.half.CMPA = EPwm2Regs.TBPRD/2;               // Set initial compare A value
101         EPwm2Regs.CMPB = EPwm2Regs.TBPRD/2;                         // Set initial compare B value
102
103         // Set actions ePWM1
104         EPwm1Regs.AQCTLA.bit.ZRO = AQ_SET;                          // Clear ePWM1A on event A, down count
105         EPwm1Regs.AQCTLA.bit.PRD = AQ_CLEAR;                        // Set ePWM1A on event A, up count
106         EPwm1Regs.AQCTLB.bit.ZRO = AQ_SET;                          // Clear ePWM1B on event B, down count
107         EPwm1Regs.AQCTLB.bit.PRD = AQ_CLEAR;                        // Set ePWM1A on event B, up count
108
109         // Set actions ePWM2
110         EPwm2Regs.AQCTLA.bit.ZRO = AQ_CLEAR;                        // Clear ePWM2A on event B, up count
111         EPwm2Regs.AQCTLA.bit.PRD = AQ_SET;                          // Set ePWM2a on event B, down count
112         EPwm2Regs.AQCTLB.bit.ZRO = AQ_CLEAR;                        // Clear ePWM2b on event A, down count
113         EPwm2Regs.AQCTLB.bit.PRD = AQ_SET;                          // Set ePWM2b on event A, up count
114 }
```

# ePWM Header File

```
1   /*
2    * ePWM.h
3    *
4    *  Created on: 23 apr 2014
5    *      Author: Christian Ekman
6    */
7
8   #ifndef EPWM_H_
9   #define EPWM_H_
10
11  void ePWM();
12
13  #endif /* EPWM_H_ */
```

# SPI Source File

```
/*
 * SPI.c
 *
 *  Created on: 23 apr 2014
 *      Author: Christian Ekman
 */

#include <DSP28x_Project.h>
#include "SPI.h"

Uint16 rdata;                                                   // Receive data buffer

void SPI()
{
        // Reset SPI
        SpiaRegs.SPICCR.bit.SPISWRESET=0;                       // Reset SPI

        // Configure transmission settings
        SpiaRegs.SPICCR.bit.CLKPOLARITY = 1;            // Data is output on falling edge and input on rising edge
        SpiaRegs.SPICCR.bit.SPILBK = 0;                         // Enable loop back mode
        SpiaRegs.SPICCR.bit.SPICHAR = 0x7;                      // 8 bit character length initially

        // Configure SPI run settings
        SpiaRegs.SPICTL.bit.OVERRUNINTENA = 0;          // Disable receiver overrun interrupts
        SpiaRegs.SPICTL.bit.CLK_PHASE = 0;                      // SPICLK signal delayed by a half-cycle
        SpiaRegs.SPICTL.bit.MASTER_SLAVE = 1;           // SPI configured as a master
        SpiaRegs.SPICTL.bit.TALK = 1;                           // Enable transmission
        SpiaRegs.SPICTL.bit.SPIINTENA = 1;                      // Enable SPI-A interrupt

        // Set bit transfer rate
        SpiaRegs.SPIBRR = 0x007F;                                       // SPI Baud Rate = LSPCLK/128 for SPIBRR = 0x7F

        // Configure transmission settings
        SpiaRegs.SPIFFTX.bit.TXFFIL = 0x2;                      // Transmit FIFO interrupt level 2 words, interrupt when TXFFST ==
              TXFFIL
        SpiaRegs.SPIFFTX.bit.TXFFIENA = 1;                      // Enable TX FIFO interrupt based on TXFFST == TXFFIL
        SpiaRegs.SPIFFTX.bit.TXFFST = 0;                        // Transmit FIFO is empty
        SpiaRegs.SPIFFTX.bit.TXFIFO = 0;                        // Reset FIFO pointer to zero
        SpiaRegs.SPIFFTX.bit.SPIFFENA = 1;                      // Enable SPI FIFO enhancements
        SpiaRegs.SPIFFTX.bit.SPIRST = 1;                        // Resume SPI FIFO transmit or receive

        // Configure receive settings
        SpiaRegs.SPIFFRX.bit.RXFFIL = 0x1;                      // Receive FIFO interrupt level 1 word, interrupt when RXFFST == RXFFIL
        SpiaRegs.SPIFFRX.bit.RXFFIENA = 1;                      // Enable RX FIFO interrupt based on RXFFST == RXFFIL
        SpiaRegs.SPIFFRX.bit.RXFFST = 0;                        // Receive FIFO is empty
        SpiaRegs.SPIFFRX.bit.RXFIFORESET = 0;           // Reset FIFO pointer to zero

        // Setup FIFO transmit delay bits
        SpiaRegs.SPIFFCT.all=0x05;                                      // 5 clk cycles delay between transfer from FIFO transmit buffer
               to transmit shift

        // Configure emulation settings
        SpiaRegs.SPIPRI.all=0x0010;                                     // Finish the current operation then stop, if emulation hits
              breakpoint

        // Enable SPI
        SpiaRegs.SPICCR.bit.SPISWRESET=1;                       // Enable SPI

        SpiaRegs.SPIFFTX.bit.TXFIFO=1;                          // Re-enable transmit FIFO operation
        SpiaRegs.SPIFFRX.bit.RXFIFORESET=1;                     // Re-enable receive FIFO operation
}
```

# SPI Header File

```
/*
 * SPI.h
 *
 *  Created on: 23 apr 2014
 *      Author: Christian Ekman
 */

#ifndef SPI_H_
#define SPI_H_

void SPI();

#endif /* SPI_H_ */
```

# nRF24L01+ Source File

```
/*
 * nRF24L01+.c
 *
 *  Created on: 23 apr 2014
 *      Author: Christian Ekman
 */

#include "nRF24L01+.h"
#include "../main.h"

extern Uint16 nRF_action;

void Init_nRF24L01(Uint16 init_ctr)
{
        switch(init_ctr)
        {
        case 0:
                // Write to config register
                SpiaRegs.SPICCR.bit.SPICHAR = 0x7;                              // 8 bit characters SPI
                SpiaRegs.SPITXBUF = (Uint16)(W_REGISTER | nRF24L01_CONFIG) << 8;
                return;
        case 1:
                SpiaRegs.SPITXBUF = (Uint16)CONFIG_INIT << 8;
                return;
        case 2:
                // Write to enable auto acknowledgement register
                SpiaRegs.SPITXBUF = (Uint16)(W_REGISTER | nRF24L01_EN_AA) << 8;
                return;
        case 3:
                SpiaRegs.SPITXBUF = (Uint16)EN_AA_INIT << 8;
                return;
        case 4:
                // Write to enabled RX addresses register
                SpiaRegs.SPITXBUF = (Uint16)(W_REGISTER | nRF24L01_EN_RXADDR) << 8;
                return;
        case 5:
                SpiaRegs.SPITXBUF = (Uint16)EN_RXADDR_INIT << 8;
                return;
        case 6:
                // Write to address widths register
                SpiaRegs.SPITXBUF = (Uint16)(W_REGISTER | nRF24L01_SETUP_AW) << 8;
                return;
        case 7:
                SpiaRegs.SPITXBUF = (Uint16)SETUP_AW_INIT << 8;
                return;
        case 8:
                // Write to automatic retransmission register
                SpiaRegs.SPITXBUF = (Uint16)(W_REGISTER | nRF24L01_SETUP_RETR) << 8;
                return;
        case 9:
                SpiaRegs.SPITXBUF = (Uint16)SETUP_RETR_INIT << 8;
                return;
        case 10:
                // Write to radio frequency register
                SpiaRegs.SPITXBUF = (Uint16)(W_REGISTER | nRF24L01_RF_CH) << 8;
                return;
        case 11:
                SpiaRegs.SPITXBUF = (Uint16)RF_CH_INIT << 8;
                return;
        case 12:
                // Write to RF setup register
                SpiaRegs.SPITXBUF = (Uint16)(W_REGISTER | nRF24L01_RF_SETUP) << 8;
                return;
        case 13:
                SpiaRegs.SPITXBUF = (Uint16)RF_SETUP_INIT <<8;
                return;
        case 14:
                // Write to status register
                SpiaRegs.SPITXBUF = (Uint16)(W_REGISTER | nRF24L01_STATUS) << 8;
                return;
        case 15:
                SpiaRegs.SPITXBUF = (Uint16)STATUS_INIT << 8;
                return;
        case 16:
                // Write to receive address data pipe 0 register
                SpiaRegs.SPITXBUF = (Uint16)(W_REGISTER | nRF24L01_RX_ADDR_P0) << 8;
                SpiaRegs.SPICCR.bit.SPICHAR = 0xF;                              // 16 bit characters SPI next case
                return;
        case 17:
                SpiaRegs.SPITXBUF = RX_ADDR_P0_INIT >> 16;              // most significant 2 bytes (out of 4)
                return;
```

```
82          case 18:
83                  SpiaRegs.SPITXBUF = RX_ADDR_P0_INIT & 0xFFFF;   // least significant 2 bytes (out of 4)
84                  SpiaRegs.SPICCR.bit.SPICHAR = 0x7;                              // 8 bit characters SPI next case
85                  return;
86          case 19:
87                  // Write to receive address data pipe 1 register
88                  SpiaRegs.SPITXBUF = (Uint16)(W_REGISTER | nRF24L01_RX_ADDR_P1) << 8;
89                  SpiaRegs.SPICCR.bit.SPICHAR = 0xF;                              // 16 bit characters SPI next case
90                  return;
91          case 20:
92                  SpiaRegs.SPITXBUF = RX_ADDR_P1_INIT >> 16;              // most significant 2 bytes (out of 4)
93                  return;
94          case 21:
95                  SpiaRegs.SPITXBUF = RX_ADDR_P1_INIT & 0xFFFF;   // least significant 2 bytes (out of 4)
96                  SpiaRegs.SPICCR.bit.SPICHAR = 0x7;                              // 8 bit characters SPI next case
97                  return;
98          case 22:
99                  // Write to transmit address register
100                 SpiaRegs.SPITXBUF = (Uint16)(W_REGISTER | nRF24L01_TX_ADDR) << 8;
101                 SpiaRegs.SPICCR.bit.SPICHAR = 0xF;                              // 16 bit characters SPI next case
102                 return;
103         case 23:
104                 SpiaRegs.SPITXBUF = TX_ADDR_INIT >> 16;                 // most significant 2 bytes (out of 4)
105                 return;
106         case 24:
107                 SpiaRegs.SPITXBUF = TX_ADDR_INIT & 0xFFFF;              // least significant 2 bytes (out of 4)
108                 SpiaRegs.SPICCR.bit.SPICHAR = 0x7;                              // 8 bit characters SPI next case
109                 return;
110         case 25:
111                 // Write to number of bytes in RX payload data pipe 0 register
112                 SpiaRegs.SPITXBUF = (Uint16)(W_REGISTER | nRF24L01_RX_PW_P0) << 8;
113                 return;
114         case 26:
115                 SpiaRegs.SPITXBUF = (Uint16)RX_PW_P0_INIT << 8;
116                 return;
117         case 27:
118                 // Write to number of bytes in RX payload data pipe 1 register
119                 SpiaRegs.SPITXBUF = (Uint16)(W_REGISTER | nRF24L01_RX_PW_P1) << 8;
120                 return;
121         case 28:
122                 SpiaRegs.SPITXBUF = (Uint16)RX_PW_P1_INIT << 8;
123                 return;
124         case 29:
125                 // Write to enable dynamic payload length register
126                 SpiaRegs.SPITXBUF = (Uint16)(W_REGISTER | nRF24L01_DYNPD) << 8;
127                 return;
128         case 30:
129                 SpiaRegs.SPITXBUF = (Uint16)DYNPD_INIT << 8;
130                 return;
131         case 31:
132                 // Write to the feature register
133                 SpiaRegs.SPITXBUF = (Uint16)(W_REGISTER | nRF24L01_FEATURE) << 8;
134                 return;
135         case 32:
136                 SpiaRegs.SPITXBUF = (Uint16)FEATURE_INIT << 8;
137                 return;
138         default:
139                 return;
140         }
141 }
142
143 void Transmit_nRF24L01(Uint16 nRF_TX,Uint16 *CommandSent)
144 {
145         switch(*CommandSent)
146         {
147         case 0:
148                 // Write to config register
149                 SpiaRegs.SPITXBUF = (Uint16)(W_REGISTER | nRF24L01_CONFIG) << 8;
150                 break;
151         case 1:
152                 SpiaRegs.SPITXBUF = (Uint16)CONFIG_TX << 8;             // Change nRF to TX mode
153                 break;
154         case 2:
155                 SpiaRegs.SPITXBUF = (Uint16)W_TX_PAYLOAD << 8;
156                 SpiaRegs.SPICCR.bit.SPICHAR = 0xF;                              // 16 bit characters SPI next case
157                 break;
158         case 3:                                                                                 // Last before TX
159                 SpiaRegs.SPITXBUF = nRF_TX;                                             // Load TX payload into buffer
160                 SpiaRegs.SPICCR.bit.SPICHAR = 0x7;                              // 8 bit characters SPI next case
161                 break;
162         case 5:                                                                                 // First after TX
163                 SpiaRegs.SPITXBUF = (Uint16)(W_REGISTER | nRF24L01_STATUS) << 8;
164                 break;
```

```
        case 6:
                SpiaRegs.SPITXBUF = (Uint16)CLEAR_INT << 8;             // clear interrupts
                break;
        default:
                break;
        }
        *CommandSent = (*CommandSent + 1) % 8;                          // Increment value every case
}

void Receive_nRF24L01(Uint16 *CommandSent)
{
        switch(*CommandSent)
        {
        case 0:
                SpiaRegs.SPITXBUF = (Uint16)(W_REGISTER | nRF24L01_CONFIG) << 8;
                break;
        case 1:
                SpiaRegs.SPITXBUF = (Uint16)CONFIG_RX << 8;             // Configure slave as RX
                break;
        case 2:
                SpiaRegs.SPITXBUF = (Uint16)R_RX_PAYLOAD << 8;  // Read RX payload
                SpiaRegs.SPICCR.bit.SPICHAR = 0xF;                              // 16 bit characters SPI next case
                break;
        case 3:
                SpiaRegs.SPITXBUF = 0xCCCC;                                             // Transmit dummy bytes
                break;
        default:
                break;
        }
        *CommandSent = (*CommandSent + 1) % 6;                          // Increment value every case
}

void flushTX_nRF24L01(Uint16 nRF_RX)
{
        SpiaRegs.SPITXBUF = (Uint16)FLUSH_TX << 8;                      // Flush TX

        if(nRF_RX == 0x5555)
        {
                return;
        }

        disable_TX();                                                                           // TX finished, disable it
        return;
}
```

# nRF24L01+ Header File

```
/*
 * nRF24L01+.h
 *
 *  Created on: 23 apr 2014
 *      Author: Christian Ekman
 */

#include "DSP28x_Project.h"

#ifndef NRF24L01__H_
#define NRF24L01__H_

// Define registers
#define nRF24L01_CONFIG                 0x00
#define nRF24L01_EN_AA                  0x01
#define nRF24L01_EN_RXADDR              0x02
#define nRF24L01_SETUP_AW               0x03
#define nRF24L01_SETUP_RETR             0x04
#define nRF24L01_RF_CH                  0x05
#define nRF24L01_RF_SETUP               0x06
#define nRF24L01_STATUS                 0x07
#define nRF24L01_OBSERVE_TX             0x08
#define nRF24L01_RPD                    0x09
#define nRF24L01_RX_ADDR_P0             0x0A
#define nRF24L01_RX_ADDR_P1             0x0B
#define nRF24L01_RX_ADDR_P2             0x0C
#define nRF24L01_RX_ADDR_P3             0x0D
#define nRF24L01_RX_ADDR_P4             0x0E
#define nRF24L01_RX_ADDR_P5             0x0F
#define nRF24L01_TX_ADDR                0x10
#define nRF24L01_RX_PW_P0               0x11
#define nRF24L01_RX_PW_P1               0x12
#define nRF24L01_RX_PW_P2               0x13
#define nRF24L01_RX_PW_P3               0x14
```

```
35  #define nRF24L01_RX_PW_P4               0x15
36  #define nRF24L01_RX_PW_P5               0x16
37  #define nRF24L01_FIFO_STATUS    0x17
38  #define nRF24L01_DYNPD                  0x1C
39  #define nRF24L01_FEATURE                0x1D
40
41  // Define command words
42  #define R_REGISTER                              0x00            // Read register
43  #define W_REGISTER                              0x20            // Write register
44  #define R_RX_PAYLOAD                    0x61            // Read receive payload
45  #define W_TX_PAYLOAD                    0xA0            // Write transmit payload
46  #define FLUSH_TX                                0xE1            // Flush transmit buffer
47  #define FLUSH_RX                                0xE2            // Flush receive buffer
48  #define REUSE_TX_PL                             0xE3            // Reuse last transmit payload
49  #define R_RX_PL_WID                             0x60            // Read receive payload width, over 32bytes flush
50  #define W_ACK_PAYLOAD                   0xA8            // Write acknowledge payload for pipe #...
51  #define W_TX_PAYLOAD_NOACK              0xB0            // Disable autoack on this specific packet
52  #define NOP                                             0xFF            // No operation
53
54  // Define initial register values
55  #define CONFIG_INIT                             0x1E            // All interrupts except max_rt, Enable 2-bit CRC, PWR_UP and TX mode
56  #define EN_AA_INIT                              0x01            // Only enable auto acknowledgement on data pipe 0
57  #define EN_RXADDR_INIT                  0x03            // Only enable data pipe 0 and 1
58  #define SETUP_AW_INIT                   0x02            // RX/TX address field width is 4 bytes
59  #define SETUP_RETR_INIT                 0xC3            // Automatic retransmission, wait 3000us and max 3 tries
60  #define RF_CH_INIT                              0x4B            // 2.475GHz radio frequency
61  #define RF_SETUP_INIT                   0x0E            // no cont. wave, don't force PLL, 2Mbps data rate and 0dBm output power
62  #define STATUS_INIT                             0x7E            // Clear status registers
63  #define RX_ADDR_P0_INIT                 0xE7E7E7E7      // Set receive address data pipe 0, 4 bytes
64  #define RX_ADDR_P1_INIT                 0xA2A2A2A2      // Set receive address data pipe 1, 4 bytes
65  #define TX_ADDR_INIT                    0xE7E7E7E7      // Set transmit address same to handle automatic acknowledge
66  #define RX_PW_P0_INIT                   0x2                     // 2 bytes in RX payload data pipe 0
67  #define RX_PW_P1_INIT                   0x2                     // 2 bytes in RX payload data pipe 1
68  #define DYNPD_INIT                              0x03            // Enable dynamic payload length data pipe 0
69  #define FEATURE_INIT                    0x06            // Disable dynamic ack, enable payload with ack & dynamic payload length
70
71  // Define run mode register values
72  #define CONFIG_RX                               0x1F            // All interrupts except max_rt, enable 2-bit CRC, PWR_UP and RX mode
73  #define CONFIG_TX                               0x1E            // All interrupts except max_rt, enable 2-bit CRC, PWR_UP and TX mode
74  #define CLEAR_INT                               0x7E            // Interrupt cleared
75
76  // Define functions in nRF24L01+.c
77  void Init_nRF24L01(Uint16 init_ctr);
78  void Read_nRF24L01(Uint16 nRF_RX,Uint16 *CommandSent);
79  void Receive_nRF24L01(Uint16 *CommandSent);
80  void Transmit_nRF24L01(Uint16 nRF_TX,Uint16 *CommandSent);
81  void flushTX_nRF24L01(Uint16 nRF_RX);
82
83  #endif /* NRF24L01__H_ */
```