Infineon Technologies

in cooperation with

Technical University of Munich
Chair of Circuit Design

Learning Platform for Practical Circuit Design

Prof. Ralf Brederlow, Matthias Ochs,
Kilian David, Adrian Keil, Eva Korek, Youssef Tarkhan

V3.0
July 13, 2022

Contents

List of Acronyms

ADC analog-to-digital converter
CCM continuous conduction mode
ESD electrostatic discharge
ESR equivalent series resistance
IC integrated circuit
MOSFET metal oxide semiconductor field effect transistor
NTC negative temperature coefficient
OpAmp operational amplifier
PWM pulse-width modulation
UVLO under-voltage lockout
UNIST Ulsan national institute of science and technology
PCB printed circuit board
FPU floating point unit

1. Introduction

Chapter Properties
Location: Home
Working time: 1 h
Content: Safety instructions and introduction to the board

Welcome to the voltage regulator lab course! Throughout this course you will learn the principles of switching converters and design a digital and analog controller for them. Along the way you will get familiar with working in an electronics lab and learn how to troubleshoot your circuits.

1.1. Safety

Before we start, let's quickly talk about safety. Throughout this lab you will work with voltages of no more than 40 V, so it is not dangerous for yourself. However, certain parts of the circuit (especially the load resistors) can get very hot during operation and could hurt you, so be careful.

Regarding the hardware: The board features extensive safety measures which protect the board in many conditions, yet it is still possible to damage the board. The biggest risk arises from electrostatic discharge (ESD). ESD is around us everyday, for example when you touch the door handle and you get zapped because your fleece clothing charged you - that is an ESD. If such a zap happens to your precious circuit board, it will be permanently

Figure 1.1.: Antistatic bags on the left and in the center, standard plastic bag on the right. Never put electronics in a standard plastic bag!

damaged. The worst is, that the integrated circuits (ICs) may not be obviously damaged but just behave false in some conditions. This makes it very hard to troubleshoot problems arising from ESD. To protect the board, it is best practice to always use an ESD wrist strap and an ESD mat. If those are not available, make sure to always touch a grounded metal (e.g. the case of the power supply) before touching the board. NEVER put the board on a standard plastic surface or worse in a plastic bag. There are special ESD bags which can be used to transport the board (see Figure 1.2). In addition, the board can be damaged if the power supply is connected to the output of the board instead of the input. So make sure that you always connect it to the input!
Furthermore, keep in mind that the microcontroller is only operating at 3.3V (or 5V). Therefore, the board will be damaged if its jumper pins are connected to anything higher than this!

1.2. The System

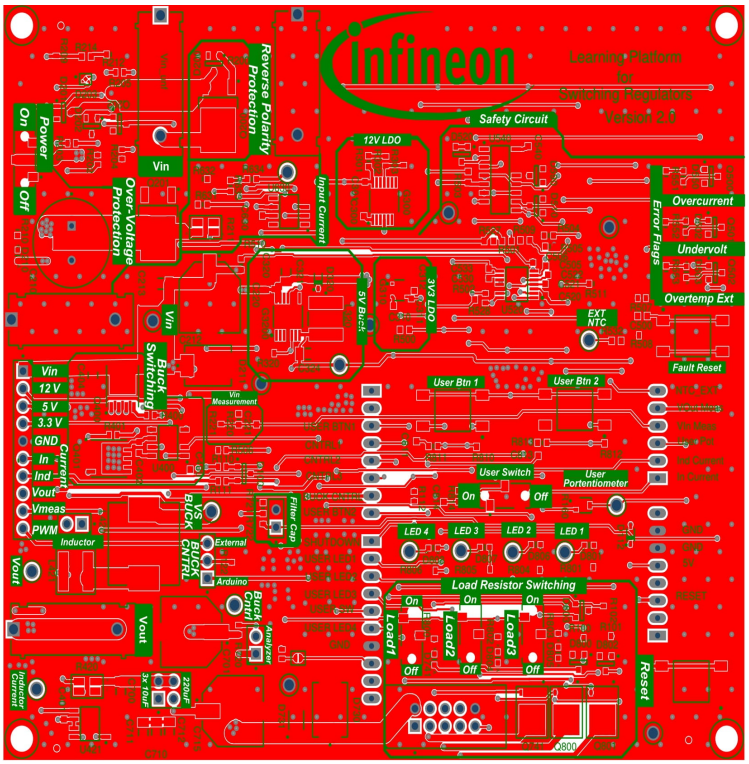


Figure 1.2.

With the safety precautions out of the way, let's get started and take a closer look at the board. While it might seem intimidating now, throughout this course we will take a closer look at the individual building blocks and you will understand how all of these components interact. In a nutshell, what you have in front of you is a switching converter with extensive protection features that can be controlled by you, either digitally or analog.

At the beginning of this course, we will familiarize ourselves with the board and the lab equipment, afterwards we will take a closer look at how switching converters work and in the end we will design our own controller.

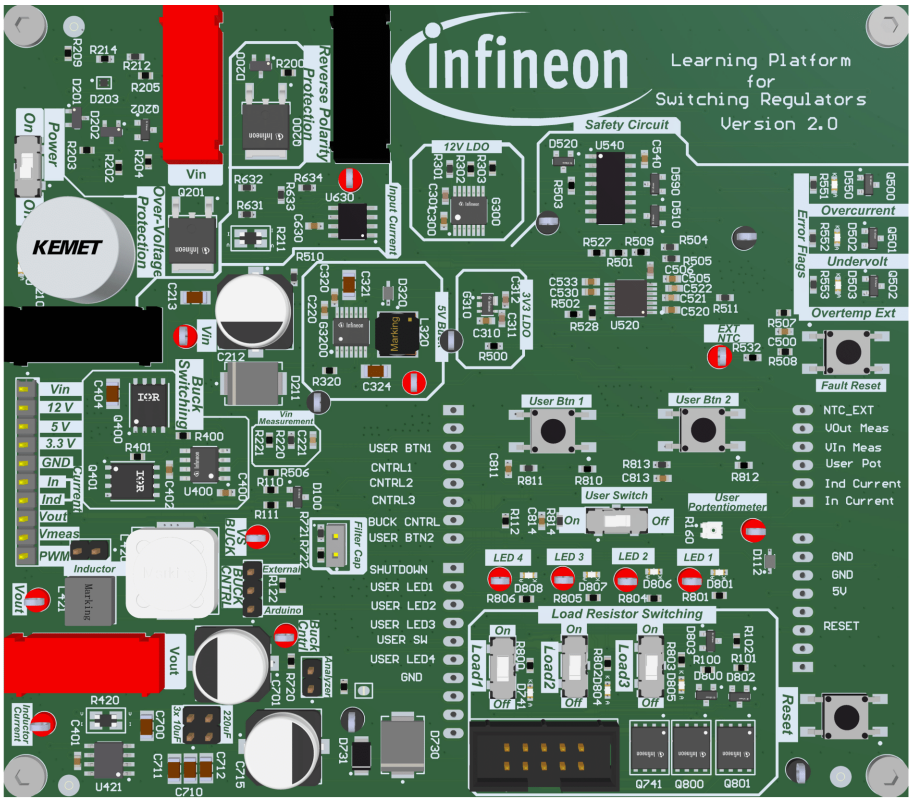


Figure 1.3.: Overview of the Mainboard. As you can see in this beautifully designed printed circuit board above the system's components are sorted into block structures (12V LDO, Reverse Polarity protection, Buck Switching ...etc). The aim behind this is to keep the board's functionality as intuitive as possible. (We'll get into more details shortly) The centerpiece of the system is the switching converter. It can be controlled either by an Arduino-compatible microcontroller board (attached from the bottom) or an external analog control board. Extensive protection features are implemented such that the board does not take damage, even if it is improperly controlled. To test the system, load resistors can be added through the connector on the left.

1.2.1. Board overview

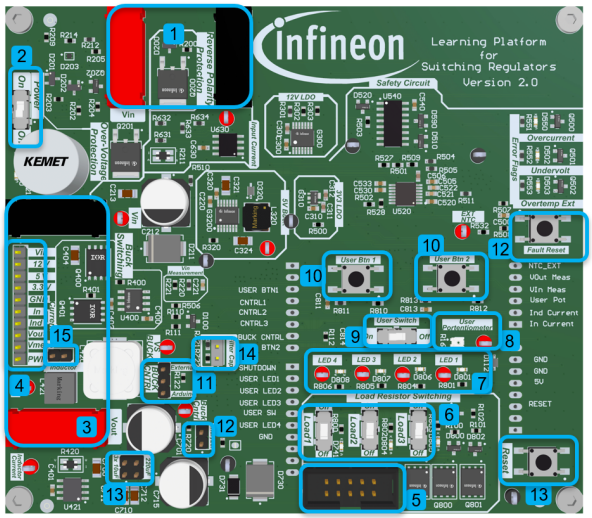


Figure 1.4.: Important elements of the board.

1) Input connectors: Connect these to the power supply.(Black connecter to GND)

2) Power switch: Turns the complete system on and off.

3) Output connectors: These can be connected to a multimeter or a consumer.

4) Connector for an external analog controller.

5) Load resistor connector: Connect the load resistor board here.

6) Load shutdown switches: Turn the loads on and off. (If the LED at the switch is on then the Load is shut down and off)

7) Programmable LEDs

8) User Potentiometer: Can be read by the Arduino and used freely in software.

9) User-programmable switch: Can be used freely in software.

10) User-Buttons: Can be used freely in software.

11) Buck Control jumpers: Choose whether you want programm the switching converter using the arduino connector pins (below the board) or using the left external control connector (labelled 4 in the figure on the left side .)

12) Safety system error latch reset button

13) Reset-Button for the Arduino.

1.2.2. Electrical Characteristics

- Nominal input voltage (system works): 1218
- Absolute maximum input voltage (system not damaged): -6060
- Output voltage: 040
- Maximum continuos current through the converter (either input or output): 2

1.2.3. Architecture

Figure 1.5 shows how all of these previously described features are implemented. The first block at the input implements the reverse polarity protection. It is important to do this directly at the input, as reverse-polarity would be lethal to the other circuitry. Next is the protection against over-voltage and the under-voltage lockout (UVLO). Both blocks break the connection between their input and output in case of a fault. In normal operation, they must carry the full power and are therefore associated with the power part (orange). The protected input voltage is used to supply the voltage regulators for the logic supply. Afterwards, the current going into the power stage is measured (input current) as well as the voltage (input voltage). The power stage is the heart of this board, performing the actual voltage conversion. To do this, it uses a non-inverting buck architecture which consists of two transistors that can switch an inductor between the input and the output of the power stage in various ways. A lower output than input voltage is produced.. Additionally there is also a current sensor in the power stage to measure the inductor current. The power stage can be controlled either by an analog control board (through the external control pins on the left side of the board) or an Arduino-compatible microcontroller board. To prevent damage in case of misuse, the input, output, and switch current as well as the output voltage and temperatures are constantly monitored. If a certain threshold is exceeded, the power stage is shut down and the error is signaled with the corresponding

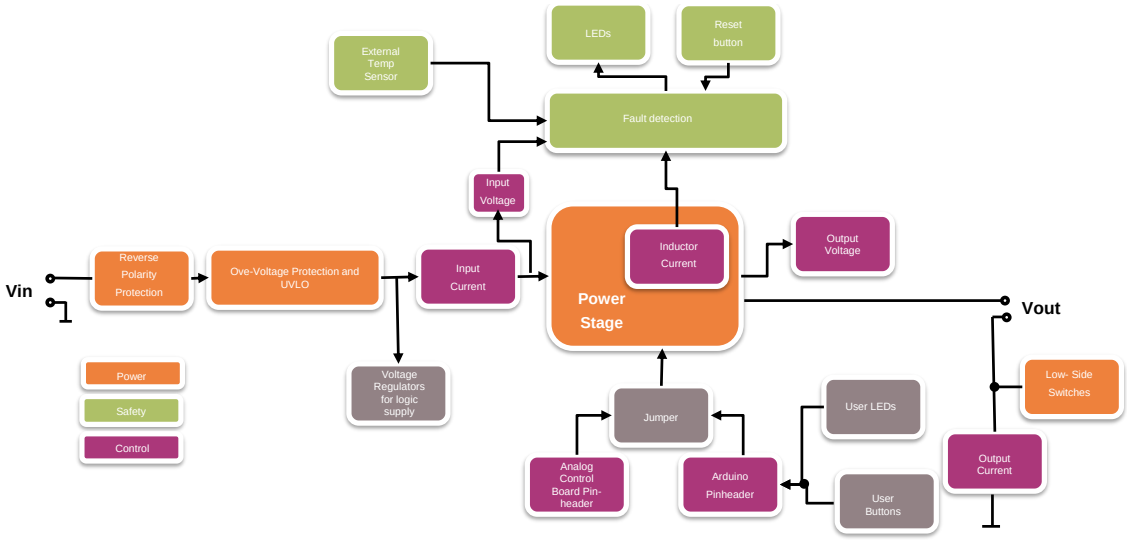


Figure 1.5.: Block diagram of the mainboard. The power blocks are drawn in orange, the safety blocks in green and the control blocks in purple. Grey blocks belong to neither of these categories. At the heart lies the power stage (dark orange) which transforms the protected input voltage to an output voltage of different magnitude. It is supervised by the safety elements at the top and controlled by either an Arduino or an analog control board at the bottom (selectable through the jumper).

LED. The power stage is only reactivated when the user presses the reset button.
In addition to the output terminals, there are three switchable outputs that can be hooked up to external load resistors.

2. Getting Started

Chapter Properties
Location: Lab
Working time: 1-2 days
Content: Introductory experiments to get familiar with the lab equipment, Arduino, and the board.

Note: This manual is designed for use with Infineon's XMC1100 Boot Kit. Nonetheless, you can use a different microcontroller board, as long as it is Arduino compatible. Therefore, this manual refers to the microcontroller board as "Arduino" whenever it does not matter which exact board you are using. However, keep in mind, that if you are using a different board than the XMC1100 Boot Kit, some instructions might not match (e.g. setting the PWM frequency).

2.1. Installation

1. Install the Arduino IDE: `https://www.arduino.cc/en/software`
2. Visit `https://github.com/Infineon/XMC-for-Arduino#supported-microcontroller-boards` and follow the steps.

2.2. Preparing the Hardware

2.2.1. XMC1100 Boot Kit

Pick up your Learning Platform and the XMC1100 Boot Kit. Make sure that the Boot Kit is set to 3.3 V. To do this, the small 0 Ω resistor needs to be in the left position as shown in Figure 2.1. As this is a 0 Ω resistor, you can also just desolder it and short the two pins on the left with solder. This is not an easy thing to do, so please ask for help from your supervisor if in doubt. Under no circumstances, the very left and the very right pin shall be shorted (that would mean a short circuit between 3.3V and 5V).

If you have just purchased the Learning Platform, you also have to solder the pin headers to it. If you have not much experience with soldering, please take a few minutes to do some research on good soldering techniques. A good guide can be found here for example: `https://learn.adafruit.com/adafruit-guide-excellent-soldering/tools`.
Now, connect the XMC Boot Kit to the Learning platform through the pin headers at the bottom.

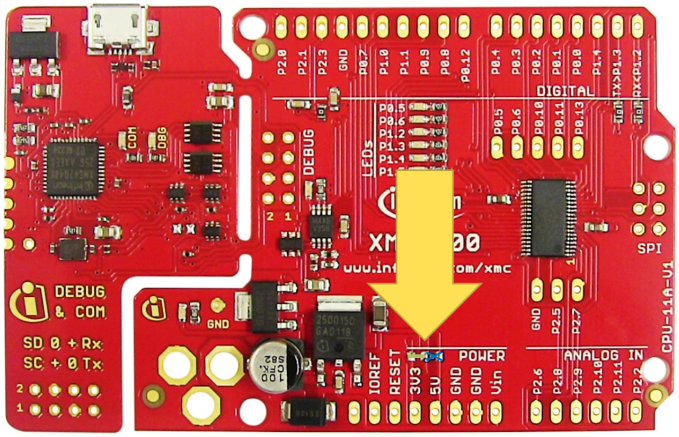


Figure 2.1.: Preparing the XMC Boot Kit. Make sure, that the 0 Ω resistor is soldered to the left.

Note: If you have to remove the Arduino, be very gentle when removing it and take your time. Do not apply much force! It is very easy to bend pins if the Arduino suddenly becomes loose. A good practice is to (gently!) pull the boards away from each other at the side of the USB connector until the Arduino is at a slight angle. Afterwards pull at the opposite side until the Arduino is angled in the other direction. Repeat this process until it becomes loose.

2.3. Blinky

We will start by letting an LED Blink on our Learning Platform. To to this, we need to power it first. So, grab a bench power-supply and two cables with banana jacks and turn on the power switch.

Exercises:

1. Turn on the power-supply without cables connected to it. Set it to 15 V.
2. Turn off the output of the power-supply and connect the cables to the mainboard. Make sure, that the polarity is correct. The positive terminal of the power-supply is usually marked red and goes to Vin+. The negative terminal of the power-supply is usually marked black and goes to Vin-.
3. Turn on the output of the power-supply. If the power LEDs on the mainboard do not light up, it is probably turned off (or you did something wrong). There is a small power switch at the bottom left side of the board. Locate it, and use it to turn the board on and off.
4. Connect the USB cable and open the Arduino IDE. Now, write the code to let one of the User-LEDs on the Learning Platform blink with a frequency of 2 Hz.

Hints:

1. There are very good tutorials on how to let an LED blink with the Arduino online. Search for them if necessary.
2. Generally, it is a good idea to turn off the board when uploading new code as pins can have random states in this process. However, the board will protect itself against damage but don't be surprised if suddenly a fault light is on after you uploaded new code. Simply reset it by pressing the fault-reset button.
3. Generally, it is a good idea to turn off the board when uploading new code as pins can have random states in this process. If suddenly a fault light is on after you uploaded new code simply reset it by pressing the fault-reset button.
4. If you have problems, make sure you selected the correct COM-Port.

2.4. Optimizing Blinky

Now, a User-LED is blinking. However, what about all the other pins? Are they configured as an input or an output? We left them undefined so depending on their configuration, this might cause trouble. Therefore, it is a good practice to initialize every pin. This becomes much more readable if you give every pin a name instead of a number. To do this, use the `#define` statement. E.g.

```
#define USER_LED1     9
#define USER_LED2     10
#define USER_LED3     11
#define USER_LED4     13
```

The compiler will replace any mention of `USER_LED1` with 4, so this does not change the program, it just makes it easier to understand. In addition, this saves you a lot of work if there would be a change in the hardware. Instead of having to change the number everywhere in the code, you just have to change it once at the definition. Generally, it is a good practice not to use "magic numbers" in the code - not only for pin-numbers but also in other places like if branches or loop conditions

Exercises:

1. Extend your code by giving every pin a meaningful name at the top of the program using the `#define` statement. It is a convention to write every letter of a defined name in capitals, such that it cannot be confused with variables.
2. Set the pin modes in the "setup" part of the code and give any output an initial value. Think about what makes sense.

Hints:

1. In Table 2.1, an overview of the Arduino Pins and their corresponding nets is shown. It is also printed on the printed circuit board (PCB).
2. Only Digital Pins have to be defined as Input, Analog Pins are configured as Input by default.

3. Buck_Cntrl should be initialized with LOW

2.5. Reading Buttons

Exercises:

1. Write a program which lights up an LED as long as Button1 is pressed.
2. Write another program which turns the LED on if Button1 is pressed and turns it off if it is pressed again.

Hints:

1. To read a button, you can use the `digitalRead` command.
2. Add a small delay after a button press was registered to avoid registering a press twice.

2.6. Serial Communication – Hello World

To send data from the Arduino to the PC, the Arduino has a serial interface (UART). The Serial Monitor can be used to view the data that is sent. The Serial Monitor is opened by

Table 2.1.: Wiring of the Arduino Pins

Pin	Net	Type
A0	In_Curr_ADC	Input
A1	Inductor_Curr_ADC	Input
A2	UserPoti	Input
A3	Vin_Meas	Input
A4	Vout_Meas	Input
A5	NTC_ext	Input
D0	Not connected	
D1	Not connected	
D2	UserBtn1	Input
D3	$\overline{\text{Cntrl1}}$	Output
D4	$\overline{\text{Cntrl2}}$	Output
D5	$\overline{\text{Cntrl3}}$	Output
D6	Buck_Cntrl_Arduino	Output
D7	UserBtn2	Input
D8	$\overline{\text{Shutdown}}$	Input
D9	UserLED1	Output
D10	UserLED2	Output
D11	UserLED3	Output
D12	UserSw	Input
D13	UserLED4	Output

clicking on the icon in the top-right corner of the Arduino IDE.

Exercises:

1. Use the `Serial.print` command to print "Hello World" once every second. Use a new line for each print.
2. It is important to note, that `Serial.print` takes a significant amount of time, unlike setting a pin or assigning a variable. Therefore, measure the time it takes to send "Hello World" 100x (without a delay). Print the time on the Serial Monitor afterwards.
3. Experiment with phrases of different length. What can you observe? Is there any difference if you use one `Serial.print` per word instead of printing it as a whole?
4. The serial communication can work with different speeds, called baud rates. The baud rate is set in the `Serial.begin` command. In the bottom right of the serial monitor you can choose different baud rates. Find the maximum baud rate that works reliably and observe the differences of different baud rates.

Hints:

1. Before you can use `Serial.print` in your program, the serial connection has to be initialized using `Serial.begin`.
2. You can use the `millis()` or `micros()` command to measure time on the Arduino.
3. The baud rate that is defined by `Serial.begin` has to match the one in the Serial Monitor. Otherwise you will receive random characters.

Note: If you measure the time of one signal `Serial.print` statement, it will be approximately the same for all baud rates. The reason is the following:

> "*As of Arduino IDE 1.0, serial transmission is asynchronous. If there is enough empty space in the transmit buffer, Serial.write() will return before any characters are transmitted over serial. If the transmit buffer is full then Serial.write() will block until there is enough space in the buffer.*" [1]

2.7. Reading from the Serial Monitor

The serial communication cannot be just used to send data but also to receive data using the `Serial.read` or the `Serial.parseInt` command.

Exercise:

1. Write a program which lets LED1 blink with the frequency you enter in the Serial Monitor.

Hints:

1. In the bottom right corner of the Serial Monitor, choose "no line ending".
2. If you don't know where to start, simply search online for "read numbers from serial monitor arduino" or something like that.

2.8. ADC

The Arduino can measure analog voltages with its built-in analog-to-digital converter (ADC). It can be accessed using the `analogRead()` command.

Exercises:

1. Write a program which measures the input voltage (the voltage from the power supply) and prints it on the Serial Monitor 10 times per second.

2. Output all three values in the format "Input-Voltage: xyz, ADC-Voltage: xyz, ADC-Value: xyz". Change the voltage at the power supply to check if your program is working.

Hints:

1. The input voltage is not directly connected to the ADC but through a voltage divider (R220, R221) as the ADC can only measure voltages up to 3.3 V (or 5 V, depending on your Arduino model).

2. As the ADC is delivering raw values, firstly you have to calculate the voltage at the ADC. Look up how this can be done. Secondly, calculate the voltage that is actually at the input.

3. The XMC1100 is running at 3.3 V.

4. You can set the resolution of the ADC using the `analogReadResolution()` command (`https://www.arduino.cc/reference/en/language/functions/zero-due-mkr-family/analogreadresolution/`). There are also commands to read the ADC resolution and the maximum value: `https://github.com/Infineon/XMC-for-Arduino/wiki/Analog-Functions-and-Additions`

5. If you have an Arduino that runs with 5V, it is wise to set the analog reference voltage to external. The mainboard is designed for 3.3V microcontrollers and by setting the analog reference voltage to external, no resolution is lost.
   WARNING: Do not use the analog reference with XMC Boards. There, the AREF-Pin is an output instead of an input!

2.9. Measuring Temperature

The new board does only have an external temperature sensor.

Even though the ADC can only measure voltages, it is still able to give us information about other parameters such as temperature and current if we use appropriate sensors. To measure temperature, the board features a negative temperature coefficient (NTC) thermistor which changes its resistance with temperature according to [2]

$$R_{NTC} = R_R \cdot e^{B \cdot \left(\frac{1}{T} - \frac{1}{T_R}\right)} \tag{2.9.1}$$

with the resistance R_R at the rated temperature T_R and the component constant B (all temperatures are in Kelvin).
If used in a voltage divider, a sensor can be built which measures the temperature and outputs a voltage as shown in Figure 2.2. The sensors on the mainboard (Int) and on the load-resistor board (Ext) are identical.

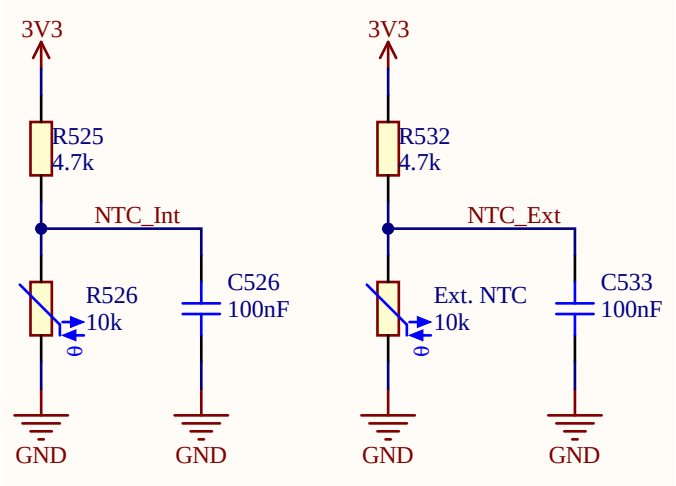


Figure 2.2.: Temperature measurement with an NTC thermistor. The capacitor is there to reduce noise.

Exercises:

1. Derive a formula to calculate the temperature from the measured voltage.
2. Write a program which displays the measured temperature on the serial monitor.
3. To test if your program is correct you can run some current through the load resistors which will heat them up. To do this, load the "Buck-unregulated.ino" program, that you can find in the folder "Arduino" on Moodle, onto the microcontroller.
   Warning: The load resistors can get very hot, do not touch them!

Hints:

1. The thermistor used on this board has the following parameters [3]

$$R_R = 10\ k\Omega$$
$$B = 3380\ K \tag{2.9.2}$$
$$T_R = 25\ ^\circ C = 298.15\ K$$

2. In order to get current running through the load-resistors, you have to generate some output voltage first.
3. The load resistors are enabled with the respective switches located at the bottom of the board.

4. You can obtain the temperature at which the safety-circuit triggers either by calculation or a measurement. Maybe you have to play a little bit around with the initial duty cycle which is defined on top of the program.

2.10. Measuring Current

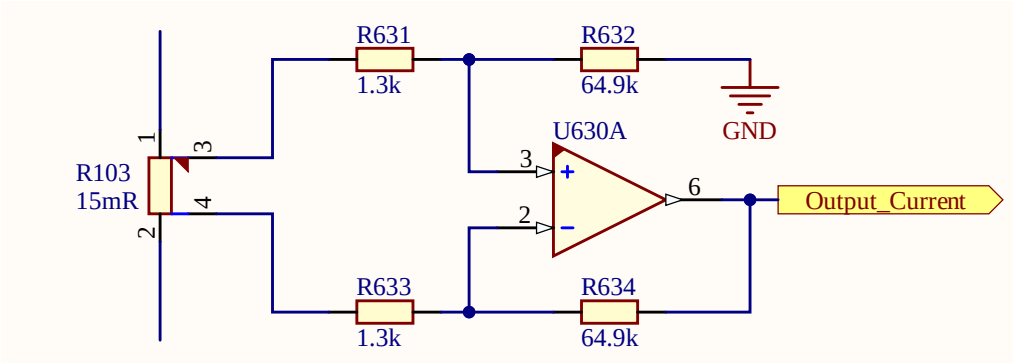


Figure 2.3.: Measurement of the output current using a shunt and a differential amplifier.

This task is not possible with the new board so you can skip this chapter
To measure current, we can take advantage of Ohm's law: $U = R \cdot I$. If we measure the voltage drop across a known resistor we can calculate the current that is flowing through it. This is known as current shunt monitoring. There are also other sensing methods like measuring the magnetic field the current generates in a wire - so called hall-effect current sensing - but they are not implemented on this board as they are generally preferred for larger currents. In Figure 2.3, the setup for the output current measurement is shown. The output current flows through R103 and generates a small voltage drop ($15\,\mathrm{m\Omega} = 15\,\mathrm{mV/A}$) which is amplified by the differential amplifier. The output of the differential amplifier is connected to an ADC-channel of the Arduino.

As current sensing is very common in electronic devices, you can buy the operational amplifier (OpAmp) with its resistors as one single IC - so called current shunt monitors. They often features specialized OpAmps and generally yield better performance at lower cost compared to the discrete approach. Therefore, they are used to sense the other currents on the board. The discrete approach for the output current was only chosen for educational purposes.

Exercises:

1. Calculate the gain of the differential amplifier (you may look up the formula) and derive an equation to calculate the output current from the measured voltage. Pay attention that the units match!

Hints:

1. To check if the program is correct, set Buck_Cntrl HIGH and turn the load resistors on and off.

2. If you compare the measured current with the one that is displayed on the power-supply, keep in mind that the board itself also draws some idle-current which has to be subtracted.

2.11. PWM

Exercises:

1. Read up on the basics of pulse-width modulation (PWM) and how a PWM signal can be generated with the Arduino. Make a drawing of a PWM-signal over time and mark the important parameters: Period T, on-time T_{on}, off-time T_{off}, and duty-cycle D.

2. Write a program which just slightly illuminates one of the onboard LEDs using PWM with a duty-cycle of 10 %.

3. Set the PWM frequency to 50 using the `setAnalogWriteFrequency(pin, f)` (only available on Infineon's Arduino-compatible microcontroller boards) with the frequency in Hz. Use `analogWrite(pin, duty * getAnalogWriteMaximum())` to set the Duty-Cycle of your PWM-signal.

4. Now, hook up an oscilloscope to the respective Arduino pin and observe the waveform. To get a meaningful image, it is important to set the vertical and horizontal resolution correctly and (most importantly) the trigger level.

5. On the Oscilloscope, add measurements for frequency and duty-cycle.

Hints:

1. Note that only Pin 3 (Cntrl_1), 4 (Cntrl_2), 6 (Buck_Cntrl), and 9 (UserLED1) are capable of PWM on the XMC 1100 Boot kit. (On the original Arduino Uno it is Pin 3, 5, 6, 9, 10, 11.)

2.12. Conclusion of this chapter

Congratulations! You have mastered the first important steps to becoming a circuit designer. So please, take a moment to relax and **write down** what you did, what was interesting, what you leaned. Basically everything that would be extremely annoying to forget. The laboratory journal is an essential to every circuit designer and trust me – nothing is more annoying than having to do an experiment twice because you didn't document it in the first place (as you thought you would not forget that) and then a few weeks later you are standing at the desk wondering what exactly happened in this experiment and what equipment you used...

3. Understanding the buck, the boost, and the buck-boost converter

Chapter Properties

```
Location:  3.1 & 3.3 at home; 3.2 in the lab
Working time:  1 day at home; 1 day in the lab
Content:  Understanding the theory of switching converters in
self-study.  Applying the knowledge in practical experiments.
Introduction to converter control.
```

3.1. Introduction

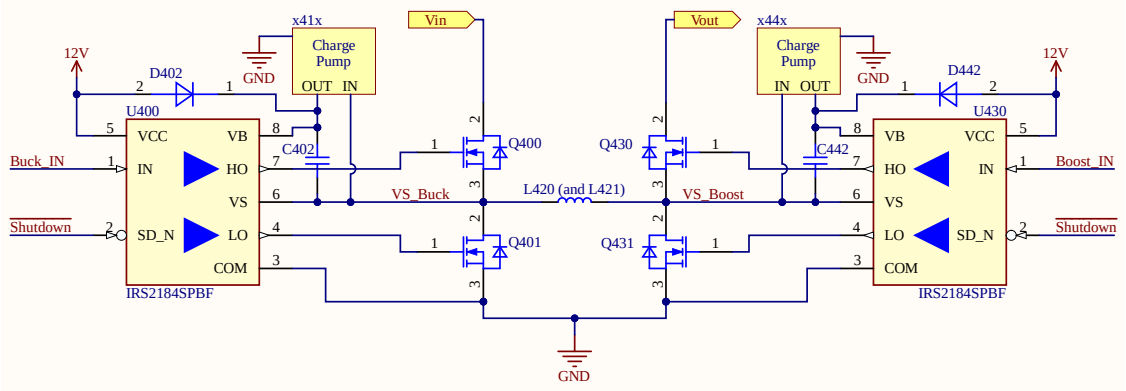


Figure 3.1.: Simplified schematic of the power stage of the previous board. At the center is the H-Bridge with its four power transistors and the inductor. It is controlled by the gate drivers, which have an additional charge pump to generate the voltage for the high-side transistors.

Note that fig. 3.1 shows the power stage of the previous board. In addition to the buck it contains a boost converter. Compare it to the schematics of the current board and locate the differences.
Before we get started with the experiments of the converters, you should understand the theory. So, read up the basics of the three converter types (focus on the buck converter as there is no boost converter on the new board). In the end it does not matter whether you get the knowledge from the literature, a youtube tutorial, some webpages or a colleague. Pick what works for you!

Some recommendations:

- Sections 9.6.4 to 9.6.9 of the Art of Electronics are a very good resource. The whole chapter 9 can be downloaded for free (`https://artofelectronics.net/wp-content/uploads/2016/02/AoE3_chapter9.pdf`).
- A more rigorous treatment can be found in Fundamentals of Power Electronics by Erickson and Maksimovic on page 15 to 29. In its following chapters, this book also goes into great detail and is highly recommended if you really want to dive into the topic of the modeling and control of switching converters. Additionally, chapter "8.3 Graphical Construction of Impedances and Transfer Functions" is absolutely amazing and will give you an intuitive understanding of bode plots. This book can be downloaded through TUM OPAC (`https://www.ub.tum.de/tumopac`).
- There is also a very good lecture series from the Ulsan national institute of science and technology (UNIST) that can be found on the Youtube-Channel "katkimshow": `https://www.youtube.com/playlist?list=PLmK1EnKxphinxBub5hL0ZoJXWoqjkGE19`. For the beginning, the video "Buck Converter Operation and Voltage Equation" may be helpful.

Exercises:

1. Why is the current at the inductor a sawtooth?
2. How does the output voltage waveform look like and why?
3. What is the dependency of output current, output voltage and load resistance?
4. Why is the average inductor current the same as the output current?
5. Calculate the output voltage in steady-state.
6. What happens to the inductor current waveform and output voltage waveform if...
   a) ...the inductance is increased?
   b) ...the capacitance is increased?
   c) ...the PWM frequency is increased?
7. How should you choose the inductance, capacitance and PWM frequency if the output ripple should be as small as possible?
8. Does the output ripple depend on the load current?

Now that you have understood the theory, we will take a look at the practical realization. On our board, the *Buck_ Cntrl* controls the switching of Q400 and Q401 to achieve the desired output voltage. If *Buck_ Cntrl* is HIGH, Q400 conducts; if it is LOW, Q401 conducts. In case of an error signal from the safety circuit (Over-current, Under-voltage or Over-temperature) an active low shut down signal is delivered which stops Q400 and Q401 from conducting.

In case you are interested in the buck-boost and boost converter, the individual modes in fig. 3.1 are achieved as follows:

- Buck Mode: Q430 is permanently conducting by setting *Boost_IN* to HIGH. Therefore, the right leg of the inductor is permanently connected to the output. We can ignored the right side of the circuit.
- Boost Mode: Q400 is permanently conducting by setting *Buck_IN* to HIGH. Therefore, the left leg of the inductor is permanently connected to the input. We can ignored the left side of the circuit. *Boost_IN* controls the switching of Q430 and Q431. If *Boost_IN* is HIGH, Q430 conducts; if it is LOW, Q431 conducts.
- Buck-Boost Mode: Both sides are switched inversely. That means either Q400 and Q431 are conducting or Q401 and Q430.

However, this is additional information and from now on we will focus on the buck converter only.

3.2. Converters without control

Exercises:

1. Use the program shown in Listing A.1 to test the buck converter. Try to understand how it works before uploading it. Check if the Arduino Pins are defined correctly and adjust if necessary.
2. Connect the oscilloscope to the testpoints Vout (TP431), Buck Cntrl (TP401), and Inductor current (TP600).
3. Play around with different input voltages and duty-cycles and observe the results. Save a meaningful image and make sure that you can explain why the curves look the way they do.
4. Connect the load resistor board to the system and observe what happens when the load changes.
5. Take an oscilloscope image at the moment when you flick the switch, so the moment when the current steps from one value to another. This is a very common scenario in reality, e.g. when a device turns an output on.
6. Take an oscilloscope image of the converter's response to a step in the duty-cycle from 30 % to 40 % with a 4 Ω load. Use a PWM-Frequency of 150 kHz. You will have to modify the program for this task.
   Please check which of the Capacitor and Inductor Jumpers are set (JP 421 and JP 710) and note the overall values of the inductor and the capacitor of the buck converter down. Also take an image where just V_{out} is visible such that the whole screen can be used for V_{out} and the complete transient effects are visible. In section 4.2 you will need this image to compare it to the mathematical model and to optimize it. Show the image to your supervisor to check if it is good.

Hints:

1. Be careful of the boards overheating when using the buck converter on the 220 µF jumper. Switch off the board in case of any unexpected observations.

2. The inductor-current sensor has a gain of 300 mV/A and it outputs 1.65 V at a current of 0 A to be able to measure positive and negative current. So:

$$I_{inductor} = \frac{V_{sensor} - 1.65\ V}{300\ mV/A} \tag{3.2.1}$$

3. Keep the board connected via USB and open the Serial Monitor to get the information about duty-cycle and output voltage.

4. To get an oscilloscope image of the exact moment when the load changes, connect the oscilloscope to the testpoint for the output current (inductor current). Trigger on the output current with the trigger mode set to single.

5. For the converter's response to a step in the duty-cycle, use the output voltage as the trigger.

6. Check the Safety-Circuit if the board does not behave the way you expect it to.

7. If an over-current fault is triggered at the start of the software, press the reset button ONCE. Your initialization sequence is probably not ideal but this will not cause any damage. If the over-current fault remains, check your settings and contact the supervisor.

3.3. Introduction to converter control

You may have noticed that if you switch the load resistors, the output voltage changes significantly and it takes quite a while until it recovers. It also does not fully recover but a slight deviation remains because of losses in the converter. To address these problems, we need to add a control loop.

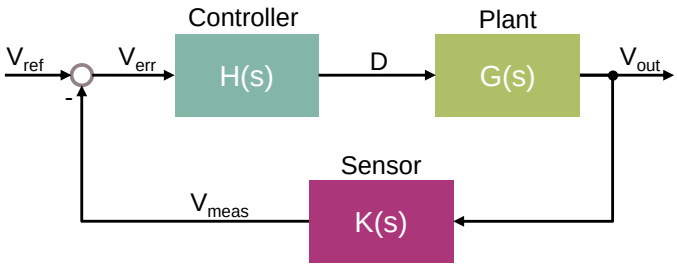


Figure 3.2.: Block diagram of a switching converter in voltage mode

In Figure 3.2, the general block diagram of a switching converter with voltage-mode control is shown[1]. The power stage produces an output voltage V_{out} that is measured by a sensor with the transfer function $K(s)$. Usually the sensor will have a low-pass characteristic to filter the switching noise on the output voltage. Afterwards, the error is calculated and fed into the controller H(s) which is usually a PI or PID controller. The controller outputs a PWM signal with a duty cycle D between 0 and 1. The power stage - our plant G(s) - then converts this to the output voltage.
In order to design the controller, we firstly need to model the power stage and the sensor. Afterwards we can properly choose the controller type, calculate its parameters and finally implement it in software or in an analog circuit.

[1] There is also the option to control a switching converter in current mode. In a nutshell, this means that the controller commands a current and then there is a second control loop which controls the current by commanding the appropriate duty-cycle. While this has some performance benefits it also complicates the analysis of the converter.

4. Modeling

Chapter Properties
Location: Home
Working time: 1 day
Content: Deriving and understanding the transfer function of the buck converter.

4.1. Modeling the ideal buck converter

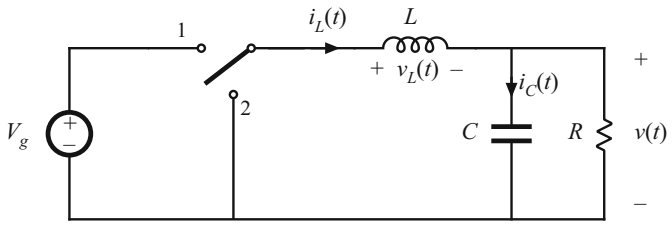
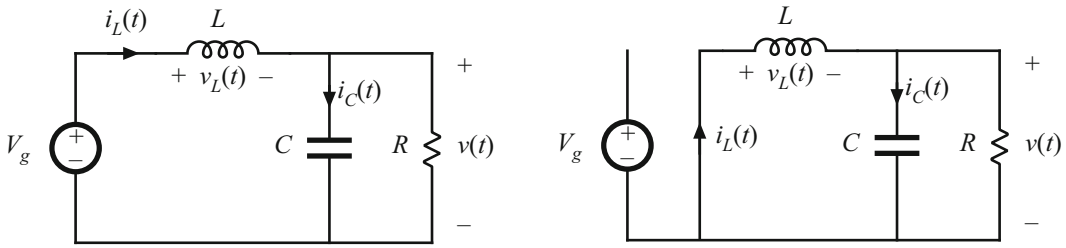


Figure 4.1.: Ideal Buck converter circuit [4, p. 18]



(a) Q400 conducting (b) Q401 conducting

Figure 4.2.: Equivalent circuit for both states of the ideal buck converter [4, p. 20]

We will again focus on the buck converter, as it is the most simple one of the three. If you are interested in the control of the boost converter, please refer to [5, pp. 52-60].
So, let's model the buck converter. The goal is to obtain a transfer function which describes the relation of output voltage and duty-cyle across the frequency range:

$$G(s) = \frac{V_{out}}{D} = \ ? \tag{4.1.1}$$

We will derive the transfer function for the ideal buck converter as shown in Figure 4.1.

As the switch can be either in position 1 or 2, there are two equivalent circuits as shown in Figure 4.2. To derive the transfer function, we will follow these steps:

1. Getting the basics done.

2. Formulating a set of differential equations. If they are non-linear, we have to linearize them around an operating point.

3. Conclude these equations in a state-space model (*Zustandsraummodell*).

4. Solve the state-space model using laplace transform and obtain the transfer function.

This process should be familiar from the control theory lecture. However, there we usually started with differential equations whereas here, we have to derive them from the circuit. Please note, that we assume that the system always operates in continuous conduction mode (CCM). Actually, this also holds true in reality because we are using two metal oxide semiconductor field effect transistors (MOSFETs) instead of one MOSFET and a diode.

Hints:

- The UNIST has also some very good videos on this topic at the very end of their playlist (`https://www.youtube.com/playlist?list=PLmK1EnKxphinxBub5hL0ZoJXWoqjkGE19`). Especially "DCDC-Converter Control: Modeling" and "DCDC-Converter Control: More Modeling" may be helpful.

- It is perfectly fine if you write the equations or even the whole chapter by hand and scan it afterwards to include it in the documentation. If you decide to write the equations on the computer, Latex is highly recommended.

4.1.1. The basics

Exercises:

1. Find a mathematical expression for D and its complementary D' in dependency on the switching period T_S and the on time T_{on}.

2. Write down the relationship of the switching period T_S and the switching frequency f_s.

Hints:

- Think simple! Just answer with short formulas..

- You can find all the information in Fundamentals of Power Electronics [4]

4.1.2. Formulating a set of differential equations

Now we need to formulate a set of differential equations. There are two state-space variables: The inductor current i_L and the capacitor voltage v_C. Find the differential equations for each switch position (so four equations in total).

Exercises:

1. Look up the Small Ripple Approximation and explain what it means. What assumption are we making? Why is it valid?

2. Formulate a mathematical expression for the inductor voltage and capacitor current in each switch position.

3. Use the previous result to find the differential equations for each switch position (so four equations in total).

4. To combine the equations from both switch positions into one, use State-Space Averaging (as introduced in [6]). This means, multiply the first equation with D and the second one with D' (respectively 1-D). Do this for i_L and v_C separately such that you end up with two equations.

5. Simplify these equations such that there is a summand with D and one without. If you did everything correctly, the differential equations should be linear. This is one of the reasons why the buck converter is easier than the boost and the buck-boost converter. There, the equations are non-linear and have to be linearized around an operating point before proceeding (see [5, pp. 53-54]).

4.1.3. Formulating the state-space model

Exercises:

1. Put the equations in matrix-form:

$$\underbrace{\begin{bmatrix} \frac{di_L}{dt} \\ \frac{dv_C}{dt} \end{bmatrix}}_{\underline{\dot{x}}} = \underbrace{\begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix}}_{\underset{\sim}{A}} \underbrace{\begin{bmatrix} i_L \\ v_C \end{bmatrix}}_{\underline{x}} + \underbrace{\begin{bmatrix} ? \\ ? \end{bmatrix}}_{\underline{b}} D \tag{4.1.2}$$

2. Find an expression for the output voltage in the form:

$$V_{out} = \underbrace{\begin{bmatrix} ? \\ ? \end{bmatrix}^T}_{\underline{c}^T} \underbrace{\begin{bmatrix} i_L \\ v_C \end{bmatrix}}_{\underline{x}} + \underbrace{?}_{d} \tag{4.1.3}$$

4.1.4. Obtaining the transfer function

Now we can finally obtain the transfer function. For this, the previously derived equations can be solved using the laplace transformation. Alternatively, we can use

$$G(s) = \underline{c}^T \left[(s \underset{\sim}{I} - \underset{\sim}{A})^{-1} \underline{b} \right] + d \tag{4.1.4}$$

which can be easily derived also using the laplace transformation (see [7, p. 272]).

Exercises:

1. Calculate the transfer function and simplify the expression.
2. Write the transfer function in the form

$$G(s) = k \cdot \frac{1}{\frac{s^2}{\omega_0^2} + \frac{s}{\omega_0 \cdot Q} + 1} \tag{4.1.5}$$

   and calculate k, ω_0 and Q. Use $V_{in} = 15\,\mathrm{V}$ and $R_{load} = 5\,\Omega$. Find the values for L and C in the schematic.
   k is the gain, ω_0 is the resonance frequency (in rad/s), and Q is the quality factor. The quality factor is a measure of dissipation in the system. A larger Q means less dissipation and also a higher peak at the resonance frequency and sharper phase change.

4.1.5. Working with the transfer function

Exercise: Enter your transfer function into Matlab and plot it. You can use the following script, you just have to modify V_{in}, ω_0 and Q.
If you haven't done it already, you need to install the "Control System Toolbox".

Listing 4.1: Matlab script to plot a transfer function

```
% Script to plot a transfer function
% Preliminary: Clear all variables, clear command window, close all figures
clear
clc
close all

% Create a special variable s that you can use in the following transfer
% function. Alternatively you could just use g = tf(numerator,denominator).
s = tf('s');

% Create the transfer function. Assigning variables with the specific
% values makes it much more readable. You could also let matlab caclulate
% w0 and Q form L, C, and R if you want an even more elegant solution.
Vin = 10;
w0 = 1e5;
Q = 10;
```

```
17 G = tf(Vin/(s^2/w0^2+s/(w0*Q)+1));
18
19 % Plot the transfer function from 1e2 rad/s to 1e7 rad/s
20 figure;
21 bode(G, {1e2,1e7})
22 grid on
23 title('My transfer function')
```

4.2. Differences between the real and the ideal world

Exercises:

1. Extend the script such that it also displays the step response to a duty-cycle step of 10 % (use the `step` command).
2. Compare it to the image you took in section 3.2. What is the main difference between our modeled step-response and the measured one? Do you have an idea why they differ?

As engineers we are used to simplifying the real world and slight deviations between modeling and measurements are perfectly fine. However, such an enormous difference is not acceptable. There must be something wrong with our model... But what did we miss? Well, remember that we modeled the components as ideal? In reality, the components aren't. And while we can neglect the resistance of the switches and the inductor or the capacitance of the traces, we cannot neglect the equivalent series resistance (ESR) R_C of the capacitor. It is too significant. Unfortunately, the derivation of the model including the capacitor ESR is a lot more complicated. Therefore, we skip the derivation (as it also doesn't help a lot to understand the behavior, it's just more math) and jump straight to the result [8, p. 5]:

$$G(s) = V_{in} \cdot \frac{1 + \frac{s}{\omega_z}}{\frac{s^2}{\omega_0^2} + \frac{s}{\omega_0 \cdot Q} + 1} \tag{4.2.1}$$

As you can see, the numerator changed and now there is a zero ω_z in addition to the quadratic pole. This is called the ESR-zero because it is caused by the capacitor ESR. It can be calculated as follows:

$$\omega_z = \frac{1}{R_C \cdot C} \tag{4.2.2}$$

The resonant frequency remains almost the same with

$$\omega_0 = \frac{1}{\sqrt{LC}} \cdot \sqrt{\frac{1}{1 + \frac{R_C}{R}}} \approx \frac{1}{\sqrt{LC}} \quad \text{if } R \gg R_C \tag{4.2.3}$$

However, the quality-factor changes significantly and is now

$$Q = \frac{1}{\omega_0} \cdot \frac{1}{\frac{L}{R} + C \cdot R_C} \tag{4.2.4}$$

Especially when R is large, the term $C \cdot R_C$ will dominate.

Exercises:

1. Write a matlab script to compare the refined model and the idealized one in the bode plot and the step response (again to a change in duty-cycle of 10 %).
2. Adjust R_C in your model until the step response of the refined model in matlab matches the one you measured. If you accomplished that, you can be pretty confident that your modeling is sufficient to design a good controller later on[1].
3. Explain the reasons why the characteristics of the refined model differ from the idealized model. Focus on the following points:
   a) Why is the peak in magnitude at the resonant frequency lower?
   b) Why is the phase change less sharp?
   c) Why does the phase only drop to around -90 deg and not -180 deg?
   d) Why is the oscillation more dampened?

Hints:

1. You can use the script in Listing A.2 to compare the models. You just have to modify the parameters of the transfer functions.
2. R_C should be between 0.1 Ω and 0.5 Ω.
3. When explaining the differences, take a look at the location of the quadratic pole and the zero.

4.3. Sensor

Now we have a good model of the plant and understand its key characteristics. Before we can finally start designing the controller, there is one small part left: Modeling the sensor $K(s)$. Its schematic looks as follows:

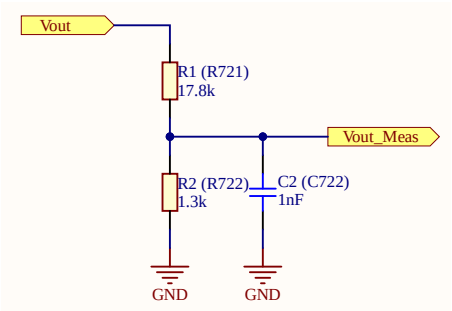


Figure 4.3.: Schematic of an output voltage sensor

[1] There are also ways to measure the transfer function in real-life, however this is more complicated than measuring the step response. If you're interested, you can read about it in [9].

You can see that the sensor is a voltage-divider with an additional capacitor at the output. This capacitor results in a low-pass characteristics. This low-pass is necessary to filter the high-frequency ripple that is present at the output of the voltage regulator. We just want to look at the average output voltage (and not the ripple) and a low-pass filter will serve this purpose. Usually its corner frequency is placed at 1/2 of the switching frequency because this provides a good compromise between sufficient filtering at not too much loss in phase margin for the open-loop (more on that later).

Exercises:

1. Check the schematics to get the correct values for the resistors of the voltage sensor.
2. Calculate the transfer function of the sensor:

$$K(s) = \frac{V_{out,meas}}{V_{out}} = \ ? \tag{4.3.1}$$

3. Calculate C such that the corner frequency (the pole) of the sensor is at 1/2 of the switching frequency. Assume a switching frequency of 100 kHz.
4. Plot the transfer function with Matlab.

Hints:

1. To calculate the transfer function, follow these steps:
   a) Calculate the impedance Z_C of the capacitor in dependency of s
   b) Combine R2 and C2 into one impedance Z_2 using the formula for parallel impedances
   c) Use the voltage divider formula to calculate $V_{out,meas}$
2. To make sure, that the result is correct, you can also model the circuit with LTSpice and run an AC simulation. As there are no simplification is this model, the results from LTSpice and Matlab should be identical.
   Pay attention to the units! By default Matlab displays rad/s whereas LTSpice displays Hz.

4.4. Open-loop transfer function

With the sensor transfer function at our hand, we can now take a look at the open-loop transfer function without a controller (or a controller with a static gain of 1). The open loop transfer function is simply

$$G_{openloop} = G(s) \cdot K(s) \cdot H(s) \tag{4.4.1}$$

where $H(s) = 1$ for the moment.

Exercises:

1. Calculate the open-loop transfer function.

2. Plot $G_{openloop}$ with matlab.
3. Would this system be stable?

Important: Use the refined model!

5. Controller Design

Chapter Properties
Location: Home
Working time: 0.5 days
Content: Understand the controller and design its parameters.

5.1. Understanding PID-Controller

Now its finally time to design the controller. There is a very good Youtube-Playlist called "Understanding PID-Controller" (`https://www.youtube.com/watch?v=wkfEZmsQqiA&list=PLn8PRpmsu08pQBgjxYFXSsODEF3Jqmm-y`) from Matlab for the basics of controller design. Look it up!
In addition let's quickly take a look at the more mathematical side of the PID controller. Its general transfer function is:

$$H(s) = k_P + k_I \cdot \frac{1}{s} + k_D \cdot s \tag{5.1.1}$$

with the parameters k_P, k_I, and k_D. The key to designing a good controller is to properly choose these parameters! In the bode-plot, the PID controller looks as follows:

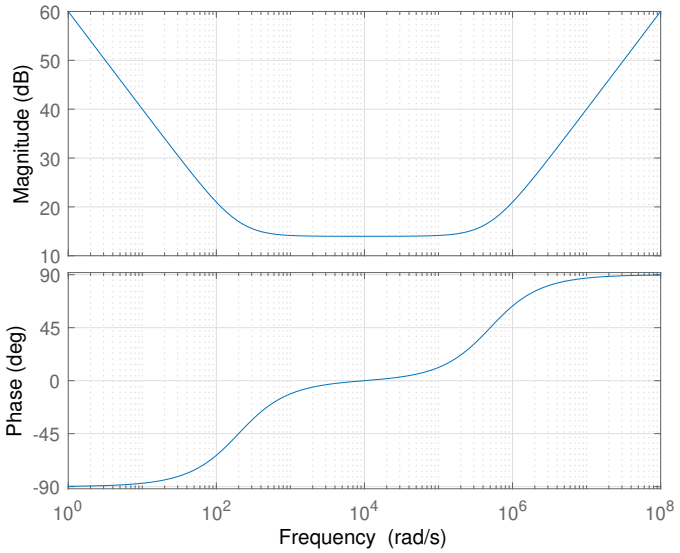


Figure 5.1.: Bode Plot of a PID controller

At low frequencies the I term dominates, causing a phase of $-90°$ and the magnitude falls with $-20\,\mathrm{dB}$/decade. At medium frequencies, the P term dominates with its static gain and 0° phase. At high frequencies, the D term dominates with a $+90°$ phase and $20\,\mathrm{dB}$/decade.
If you are wondering why P, I and D look the way they do in the bode plot, this can be explained rather quickly. Let's assume we have some input signal $f(t) = sin(\omega t)$. The responses of the individual elements are:

$$\begin{aligned}
&\text{I: } k_I \cdot \frac{1}{\omega} \cdot (-cos(\omega t)) \\
&\quad \Rightarrow \text{Amplitude} = k_I \cdot \frac{1}{\omega} \\
&\text{P: } k_P \cdot sin(\omega t) \\
&\quad \Rightarrow \text{Amplitude} = k_P \\
&\text{D: } k_D \cdot \omega \cdot cos(\omega t) \\
&\quad \Rightarrow \text{Amplitude} = k_D \cdot \omega
\end{aligned} \tag{5.1.2}$$

Note: If you plot $\frac{1}{\omega}$ a double logarithmic diagram, it results in a straight falling line.
Regarding the phase: A $-cos()$ lags $-90°$ behind a $sin()$ and the $cos()$ is 90° ahead...

5.2. Determining the controller parameters

For our system we don't need a PID controller, a PI controller is sufficient (more in section 5.3). The general transfer function of the PI controller is

$$H(s) = k_P + k_I \cdot \frac{1}{s} \tag{5.2.1}$$

with the parameters k_P and k_I.

Exercises:

1. Write a matlab script which displays the open-loop transfer function in the bode plot as well as the step-response of the closed loop. Follow these steps:
   a) Declare the transfer functions of the plant, the sensor and the controller. Use $k_P = 3,\ k_I = 0$ as starting values.
   b) Plot the open-loop transfer function. In addition, plot the transfer function of the controller $H(s)$ as well as the combined one of plant and sensor $G(s) \cdot K(s)$ into the same bode plot.
   c) Plot the step response of the **closed** loop.

2. The zero crossover frequency f_0 is also called "the bandwidth of the loop" or "the bandwidth of the system". Why is the zero crossover frequency of the open-loop the same as the bandwidth of the closed loop?

3. Now you have built the tool to determine the values for k_P and k_I. And with this at hand, the final process is rather simple:
   a) Gradually increase k_P until the bandwidth of the loop f_0 is around 10 % of the switching frequency. Look at the bode-plot and observe how the system changes. Assume a switching frequency of 100 kHz.
   b) Afterwards, gradually increase k_I as much as possible while still maintaining a phase margin of at least 60° (65° to 75° is better). Look at the bode-plot and observe how the system changes.
   c) If you are done, note down the values for k_P and k_I and check with your supervisor.

4. Design a second controller for a closed-loop bandwidth f_0 of just 20 Hz (this will be used for the digital control). An I controller ($k_P = 0$) is sufficient for this task. Note down the value for k_I.

Hints:

1. You can use the matlab script in Listing A.2 as a starting point. To increase the clarity of the diagram, use different line specifications (`https://de.mathworks.com/help/code/ref/linespec.html`).

2. The closed-loop transfer function is **not** $H(s) \cdot G(s) \cdot K(s)$. If you are not sure how to calculate the closed loop transfer function, take a look at you control theory script.

3. There is a very good German video on open-loop and closed-loop transfer functions: `https://youtu.be/N0m5B1ND2yI`

4. Pay attention to the units! Matlab displays rad/s whereas the switching frequency is in Hz.

5. k_I has to be orders of magnitude larger than k_P. This is normal. Start with $k_I = 5e3$.

6. To get the precise zero crossover frequency and the phase margin of a transfer function, you can use the `margin` command. E.g.:

```
[Gm,Pm,Wcg,Wcp] = margin(G * K * H);
fprintf('Crossover frequency:  %0.3g rad/s \n',Wcp);
fprintf('Phase Margin:  %0.1f deg \n',Pm);
```

7. Your report should contain the script as well as the bode-plots and step-responses for both controllers along with a short explanation.

5.3. Excursion: Why is a PI-Controller sufficient for us? Why do you need a PID controller on some other systems?

You may be wondering why we are using a PI controller instead of PID. What about the D-term? Don't we need that? The short answer is we don't need it in our controller because it is already in our plant!

But why do you need the D term at all? Well, the key to designing a stable system is to maintain a decent phase margin! And depending on our system, this is more or less complicated. A lot of systems in the real world have a transfer function like the ideal buck-converter: A system with a quadratic pole and no zeroes - a PT_2 system. Such systems have a phase change from 0° to −180° around the resonance frequency. Therefore, if we want to get a closed-loop bandwidth that is much greater than the resonance frequency (e.g. 1e5 rad/s in Figure 5.2) the phase margin would be almost zero.

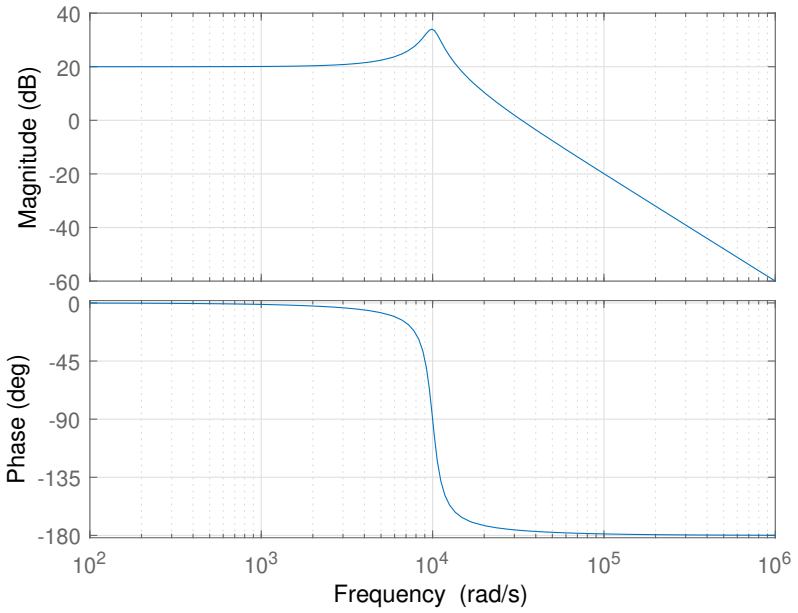


Figure 5.2.: Bode Plot of a PT_2 system.

What we need to do is to increase the phase after the resonance frequency. And that's where the D term with its phase of -90° comes into play. If we design the controller properly, the D element will boost the phase and increase the phase margin (Figure 5.3).

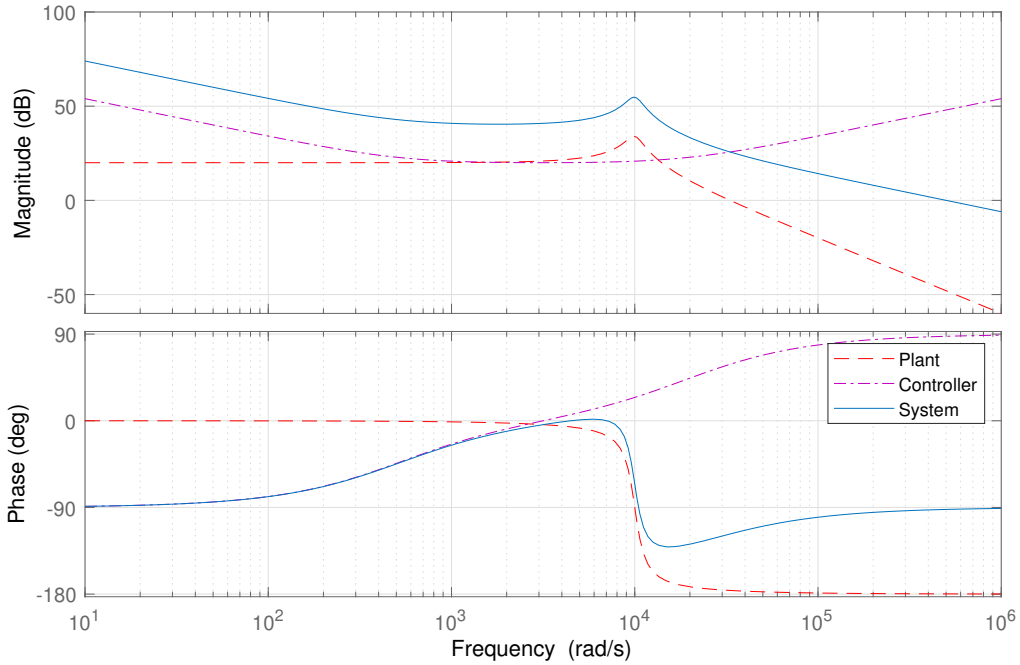


Figure 5.3.: Design of the open-loop transfer-function for the previous system.

So now, back to our system: Because of the capacitor ESR we don't have a PT_2 system but PDT_2 - there is an additional zero. Do you notice something? We don't need the D term in our controller because it is already in our plant. The ESR introduces an additional zero which causes a phase boost of 90°, just like the D term does. Or alternatively, one could argue, that the phase of our system only drops to $-90°$ and therefore, no additional phase boost is needed.

And what about the digital controller? Why is it just an I controller? A pure I controller is sufficient here, because up to the closed-loop bandwidth and even a decade beyond, the phase of our plant is 0°. Therefore, it's sufficient if the controller has a constant phase of $-90°$.

6. Digital Controller

Chapter Properties
Location: Home/Lab
Working time: 1-2 days
Content: Implementing a digital controller for our system.

6.1. Introduction

Now that we know which controller is needed, all that's left to do is to implement it. First, we will do this digitally on the Arduino. If we use the parameters which we determined in the previous chapter, this should result in a stable system with a decent step-response.

Before we get started, let's quickly recall the block diagram from section 3.3:

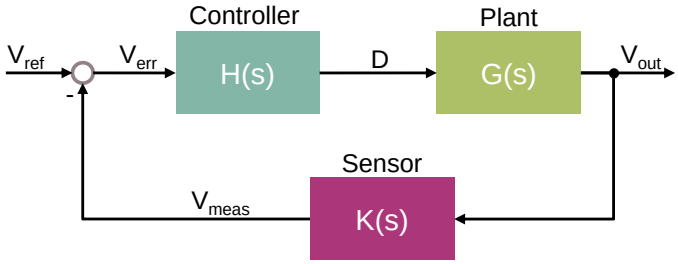


Figure 6.1.: Block diagram of a switching converter in voltage mode

We want to write a software which implements the controller H(s). So a software which takes an input voltage V_{err} and outputs a duty-cycle D according to

$$D(s) = V_{err}(s) \cdot H(s) = V_{err}(s) \cdot \left(k_p + k_i \cdot \frac{1}{s}\right) \tag{6.1.1}$$

Please note: V_{ref}, V_{err}, V_{Out}, and V_{meas} are all voltages that can actually exist in our system. However, the duty-cycle is just a parameter of a PWM-signal. It may seem strange that the controller's input is an analog voltage and the output is a PWM signal but mathematically there are no differences between a voltage like V_{err} and a duty-cycle - it's just a number. However, in reality there is of course a difference. Therefore, after our program has calculated the duty-cycle, it has to generate a PWM signal with a corresponding duty-cycle. We will do this by using the timer-peripheral of our microcontroller. In the analog control circuit we will need an additional block to perform this conversion from D to an actual PWM signal called the Modulator. More on that in chapter 8.
Also, as D is dimensionless, k_p and k_i would have to have the unit 1/V, but for the control

part we will simply ignore units and just treat all variables as dimensionless (with voltages given in volts and time in seconds).

6.2. Calculating the duty-cycle in time domain

In reality, we can only measure signals in the time domain. So we have to find a mathematical expression for $D(t)$ in dependency of $V_{err}(t)$. Because $\frac{1}{s}$ corresponds to an integrator, the controller block diagram looks as follows:

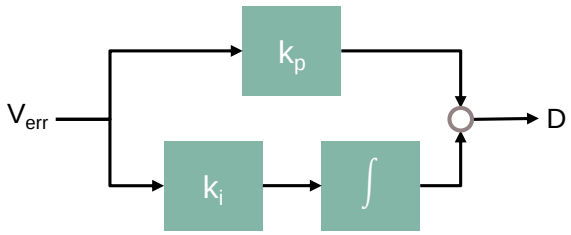


Figure 6.2.: Block diagram of a PI-controller with transfer function $H(s) = \left(k_p + k_i \cdot \frac{1}{s}\right)$.

Now we can easily see how to calculate $D(t)$:

$$D(t) = k_p \cdot V_{err}(t) + k_i \cdot \int_0^t V_{err}(\tau)\, d\tau \tag{6.2.1}$$

6.3. Approximating an integral in a time-discrete system

An analog controller *continuously* measures the output voltage. In contrast, a digital controller *samples* the output voltage with its ADC at discrete times. It also quantizes the signal, but we can neglect this effect as the ADC-resolution is high. The time between two measurements is called the sample-time t_{sample} with the corresponding frequency f_{sample}. In our program it will be determined by the time the ADC takes to do the conversion plus the time for all of the calculations. So instead of a continuos signal we just have discrete points as illustrated below:

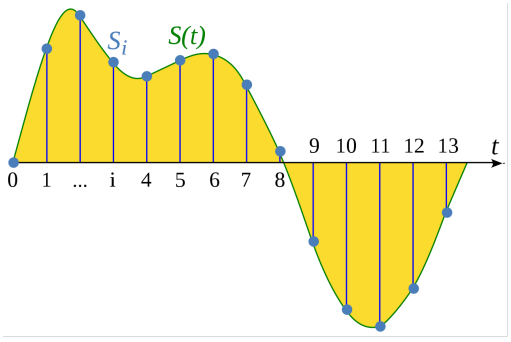


Figure 6.3.: Illustration of the sampling process. The continuous signal S(t) is represented with the green colored line while the discrete samples are indicated by the blue vertical lines.

How do we calculate an integral if we just have discrete measurement points? There are very sophisticated methods to approximate an integral but we will use the most simple one: We just assume that the value remains constant until the next sample point:

$$\begin{aligned} \int_0^t V_{err}(\tau)\, d\tau &\approx \sum_{j=0}^{N} V_{err}(t_j) \cdot \Delta T \\ \Rightarrow D(t) &= k_p \cdot V_{err}(t) + k_i \cdot \sum_{j=0}^{N} V_{err}(t_j) \cdot \Delta T \end{aligned} \tag{6.3.1}$$

Exercises:

1. Draw a sketch which shows how the integral and the approximation differ.

2. As it is not possible to measure V_{err} directly, think about a way how to calculate the value of the error with the help of V_{meas}.

6.4. Algorithm

With the math out of the way we can proceed to actually designing an algorithm.

1. **Initialization:** First, we will have to initialize our system, e.g. define all pins as input or output, set the PWM frequency and optionally set PWM and ADC resolution.

2. **Determining the sample-time:** In order to calculate the duty-cycle we need to approximate the integral as discussed in section 6.3. To do that, we need ΔT. There are two ways to obtain ΔT:

   a) Variable sample-time: Measure the time between two iterations of the loop. Easier to implement but almost impossible to simulate properly.

   b) Fixed sample-time: At the beginning of the loop, wait until the desired sample-time is reached. This means that if the measurements and calculations were faster than the desired sample-time, the program will wait until the next run is scheduled. Harder to implement but easy to simulate.

3. **Calculate the error voltage:** The other value that is needed to calculate the duty-cycle is the error voltage. Therefore, we have to measure the sensor voltage (not the output voltage!) and compare it to the reference voltage[1].

[1] If you use the output voltage to calculate the error, the controller's response will be much larger than calculated and the system may become unstable. The reason is that when designing the controller we calculated with an open-loop of $H(s) \cdot G(s) \cdot K(s)$. Therefore, we already took the attenuation of $K(s)$ into account. If we wanted to use the output voltage instead of the sensor voltage, $K(s)$ would be 1 and we would have to recalculate the controller parameters.

4. **Calculate the controller output:** Now we can proceed to actually calculating the duty-cycle using Equation 6.3.1:

$$D(t) = k_p \cdot V_{err}(t) + k_i \cdot \sum_{j=0}^{N} V_{err}(t_j) \cdot \Delta T$$

5. **Generate the PWM signal:** Last but not least, generate a PWM-signal with the calculated duty-cycle. Afterwards, go back to step 2.
   Please note: In reality the duty-cycle can never be larger than 1 or smaller than 0. However, mathematically there are no limits. There may occur a problem if you try to set a non-valid duty-cycle with `analogWrite`. Add a feature to your code that prevents this. The new board cannot deal with a duty-cycle of 100%. Therefore, please limit the maximum duty-cycle to 95%!

6.5. Implementation

Exercises:

1. Implement the software such that an output voltage of 5 V is achieved. Use the controller parameters determined in the previous chapter for a closed-loop bandwidth of 20 Hz.

2. Upload the program and test if it works. Use various loads. If it works, try other output voltages as well.

3. Look up the term integral windup and design a countermeasure.

4. Extend the program such that you can control the output voltage while the program is running. You can use an input from the serial monitor, the buttons or the user potentiometer. What range of values make sense for V_{ref} if V_{in} is 15 V?

   It is highly recommended to use the two user buttons for this task. Note that they only work on the practical boards which are marked with a green point.

5. Find a way to measure the sample-time, e.g. by toggling an LED. The current controller is designed for a sample-time of 1 ms. If your sample-time is less, you can also increase the bandwidth of the system. Use the script from the last section to determine the new controller parameters. As a rule of thumb, choose $f_0 \approx 1/50\ f_{sample}$.

6. Increase the PWM and ADC resolution. Are there any differences? Explain your observation.

Hints:

1. On moodle you can find the Arduino File "Buck-controller_Framework.ino" which can help you to write the code.

2. To determine V_{ref}, calculate the corresponding V_{meas} first. This can be done by using the voltage divider formula or by evaluating the transfer function at $s = 0$. Think about what V_{err} should be in steady-state.

3. Use a PWM-frequency of 150 kHz.

4. Mathematically, the duty-cycle is a value between 0 and 1. However, the `analogWrite` command expects a different range of values. Rescale the value accordingly. Make sure that the duty-cycle is always below 100%.

5. Use one variable to store the approximated integral $\sum_{j=0}^{N} V_{err}(t_j) \cdot \Delta T$.

6. If you use a fixed sample-time, start with a sample-time of 1 ms.

7. If you use a fixed sample-time, the calculations may never exceed this time, otherwise the system may become unstable. Therefore, it is a good idea to illuminate an LED if the system is too slow.

8. To test that the controller parameters work with a given sample-time you can use the provided Matlab Simulink model.

9. There are a lot of PID example projects for Arduino on the internet. This can help you in the beginning.

10. Prints on the serial monitor take very long. Only print if absolutely necessary for debugging and print as few characters as possible.

11. To measure the sample-time, serial prints are not well-suited as they cause significant delay.

12. To set the PWM and ADC resolution, take a look at `https://github.com/Infineon/XMC-for-Arduino/wiki/Analog-Functions-and-Additions`.

13. Bonus: If you want to significantly reduce the sample-time, get rid of all floating-point calculations. The XMC1100 does not have a dedicated floating point unit (FPU). Therefore, these calculations take very long. Instead, use 32-bit signed integers (`int32_t`) and calculate in microvolts. However, this is not easy! Only do this if you are done with all other exercises including the remaining chapters.

7. Measurements

Chapter Properties
Location: Lab
Working time: 0.5 days
Content: Evaluating the performance of the controller.

7.1. Stability

Now it's time to test our controller and evaluate its performance. The most important criteria is: Is our controller stable? If it is stable, how quickly does it react to disturbances? To answer this, measure the response of the system to a load-step.

Exercises:

1. Check if the controller is stable in a variety of operating points. To do this, switch the load resistors on and off and observe the output voltage on the oscilloscope. Do this test for output voltages of 1.5 V, 3.3 V, 5 V, 9 V, and 12 V.
   If it is not stable in any of the operating points, contact your supervisor. There may be an error in your controller software or in the parameters.

2. Measure the response of the system to a current step. Do this at $V_{out} = 3.3\,\mathrm{V}$. Try different current steps and take meaningful oscilloscope images.

3. Does the response look like the one in Matlab (qualitatively)?

Hints:

- At higher output voltages you cannot switch on all load resistors at the same time. Otherwise you will trigger the over-current protection.

- To obtain the step-response, set the trigger mode to single. It may be helpful to measure the output current as well and trigger on the current instead of the output voltage.

- Use DC-coupling for the output voltage. Why? The AC-coupling in the oscilloscope is achieved with a high-pass filter which lets only the high-frequency components pass. However, our controller is very slow compared to the frequencies the oscilloscope usually measures and may be below the high-pass filter's bandwidth. This can lead to an incorrect measurement.

7.2. Efficiency

Efficiency is a very important parameter of power supplies. Therefore, let's measure the efficiency of our system.

Exercises:

1. Set the output voltage to 3.3 V. Measure the output current for the full load using the inductor current sensor.
2. Measure the efficiency of the system at four different loads.

7.3. Output Ripple

Exercises:

1. Measure the ripple of the output voltage of the converter.
   a) What is the peak-to-peak amplitude of the ripple? Does it change with the load and output voltage?
   b) Test the different capacitors that can be added through the jumpers on the board. How do they affect the ripple? Give reasons for your observations.
   c) At which frequency does the ripple occur approximately? Where does it originate from?

Hint: Use AC-Coupling.

8. Analog Controller

Chapter Properties
Location: Design/Simulation at home; Implementation in the lab
Working time: 3-5 days
Content: Designing the circuit of the analog controller and implementing it on a breadboard.

8.1. Introduction

In this section, we will implement an analog version of our controller. As this controller will output an analog voltage and not a PWM signal, we also have to implement a device that can convert an analog voltage into a PWM signal. Such a device is called a modulator (sometimes also a PWM-generator). It is inserted in between the analog controller and the plant, resulting in the following block diagram:

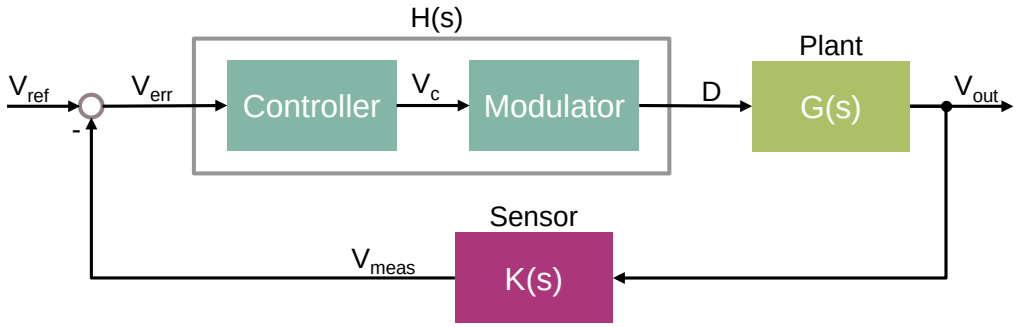


Figure 8.1.: Block diagram of a switching converter with an analog controller.

8.2. Designing the modulator

8.2.1. Getting to know the NE555

We will implement the modulator based on the forever-popular NE555 timer IC. Its internal circuit looks as follows:

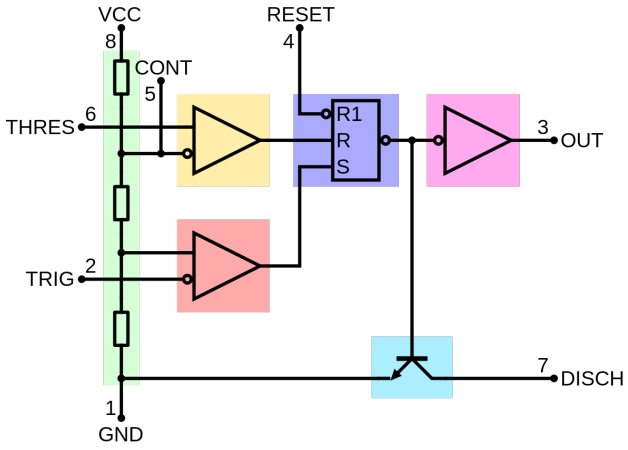


Figure 8.2.: Internal circuit of the NE555 timer.

Before we get started, familiarize yourself with the NE555. There are a lot of good tutorials on the internet. The datasheet may also be helpful.

Exercise: Explain the basic functionality of the IC and the 8 different pins.

8.2.2. Understanding and simulating the circuit

To implement the modulator, we will use the NE555 to generate a sawtooth waveform. Afterwards, the sawtooth waveform is compared to the output voltage of our controller in a comparator. This way, the comparator will produce a PWM signal with a duty-cycle that is proportional to the voltage. The waveforms are shown in Figure 8.3.

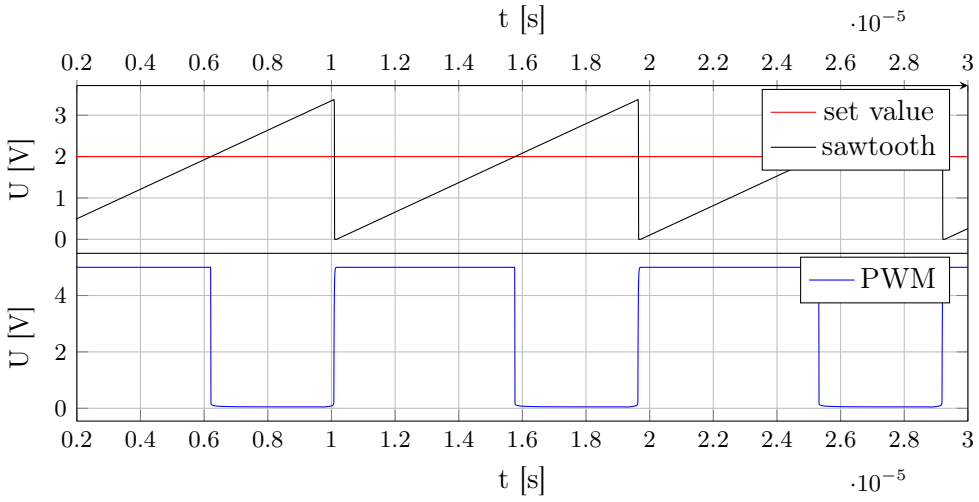


Figure 8.3.: Working principle of the modulator: An analog voltage is compared to a sawtooth waveform to generate a PWM signal with a duty-cycle that is proportional to the analog voltage.

To generate the sawtooth waveform, we will use the following circuit:

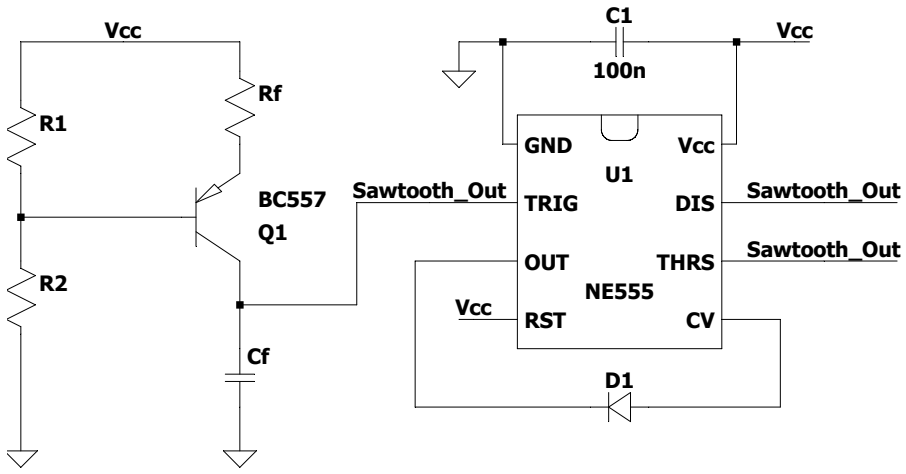


Figure 8.4.: NE555 Timer saw-tooth generator

Exercises:

1. Explain how the circuit works.
2. Dimension the circuit elements for a PWM frequency of $f_{pwm} = 100\,\text{kHz}$. Use $V_{cc} = 5\,\text{V}$.
3. Simulate the circuit in LTSpice and make sure that it works the way it is supposed to.
4. Sawtooth_Out charges from 0 V to $2/3V_{cc}$. Why?
5. The 100 nF capacitor C1 is a so-called decoupling capacitor. Look up this term an explain why decoupling capacitors are used.

Hints:

1. R_1, R_2, R_f and $Q1$ form a constant current source. This article may be helpful to understand it: `https://www.electronics-notes.com/articles/analogue_circuits/transistor/active-constant-current-source.php`.
   If you are not familiar with bipolar transistors, there are essentially just two things you have to understand for this circuit: 1. The base current is around 1/100 of the emitter current. 2. The voltage between the base and the emitter U_{be} will be 600 mV (assuming a silicon transistor).
2. Simulate the constant current source separately before simulating the whole circuit. You can use a resistor as a load for it.
3. Dimension R_1, R_2 and R_f such that $i_b << i_{R_2}$. Otherwise the usual voltage-divider formula is not valid.
4. Keep your resistors in the range of kΩs.

5. Use between 1 nF and 10 nF for C_f.
6. Pay attention that the voltage at the base of the transistor is larger than the maximum voltage at the capacitor C_f (otherwise the current will not be constant towards the end of the charging period).
7. f_{pwm} is set by C_f and the current charging C_f.
8. Assume no current flows into the Trig, Dis and Thrs pins of the NE555 during the charging phase.
9. Wires in a schematic that have the same name are connected.

8.2.3. Implementing the circuit

Exercises:

1. Implement the circuit for the saw-tooth generation on a breadboard.
2. Set the current limit of the power supply to 100 mA.
3. Probe the output of your circuit with an oscilloscope and compare it to your simulation. Does it work?
4. Examine the maximum frequency range you can produce with your design and describe the high-frequency breakdown behavior.
5. Set the circuit to the highest PWM frequency that can be produced without problems. You should be able to reach at least 100 kHz - 150 kHz.
   You can connect a resistor between Pin 5 and ground to adjust the internal voltage divider if your amplitude is too high.

Hints:

1. The pinout of the NE555 is the same as in the schematic. The pinout of the BC557 (the PNP-transistor) can be obtained from its datasheet. Look it up!
2. Use a 10 kΩ potentiometer for R_f to manually tune the frequency.
3. Use wire bridges whenever possible and only use long, flexible wires if necessary.
4. Use some kind of color code in your circuit. E.g. yellow for 5 V, blue for GND, green and purple for signals,...
5. Keep the 100 nF decoupling capacitor very close to the supply pins of the NE555. So ideally place if above the IC.
6. The effect of the diode is very interesting. You can exchange the diode with a different model or try the circuit without the diode and observe the differences.

7. Sometimes the resistors in the sets have been improperly sorted. Therefore, measure the value of every resistor and make sure it is correct before you use it. The same applies to capacitors. However, the value of the capacitor is written on it and you don't have to measure it. On smaller capacitors it has to be decoded first: The first two digits are the value and the third one is the multiplier (unit: pF). E.g. 472 means $47 \cdot 10^2$ pF = 470 pF.

8. **Important:** Electrolytic capacitors have a polarity. Reverse polarity can lead to serious damage and injuries. The side that is marked with a (dashed) line is the negative pole.

Before turning on your circuit, contact a supervisor to check your wiring.

8.3. Comparator

Now we will implement the comparator based on the LM393. The circuit compares the saw-tooth signal with the control signal from the PID controller, producing a PWM-signal, where the duty cycle is set by the PID control. The waveforms are shown in Figure 8.3.

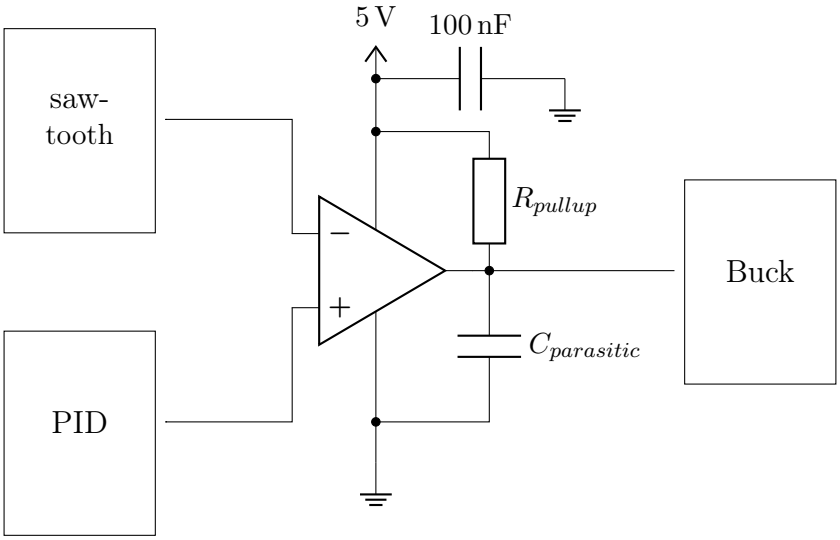


Figure 8.5.: Circuit Implementation of the PWM Generator using the LM393 Comparator

Exercises:

1. Explain why R_{pullup} is necessary.

2. Implement the circuit on the breadboard.

3. Use a potentiometer to generate an artificial PID signal, so you can set different duty-cycles.

4. Observe the influence of different R_{pullup} and C_{load} values on the rise time t_{rise} of the PWM signal. C_{load} simulates the capacitive load of the buck converter input. Explain your observations.

Hints:

1. Take a look at the internal schematic of the comparator on page 18 of its datasheet [10]. Especially the output stage is relevant for us.

2. Tie all unused pins to the negative supply voltage of the IC (in this case GND). This prevents unwanted switching of the unused comparator.

3. If you simulate this circuit you can use one of the comparators in the library like the LT1720. Alternatively, you can use the "UniversalOpamp2" model, but you should give it a 10x higher open-loop voltage gain ("Avol"), gain-bandwidth product ("GBW") and slew-rate ("Slew") in order to achieve a realistic speed. Furthermore, keep in mind that both the LT1720 and the "UniversalOpamp2" have a push-pull output stage which the LM393 does not have, so the effects of R_{pullup} and C_{load} will not be visible in the simulation.

4. Look up the pinout of the LM393.

8.4. Analog PID Controller

In this section, we will implement the PI controller we simulated earlier in Matlab as an analog circuit. Figure 8.6 shows the basic blocks of this circuit.

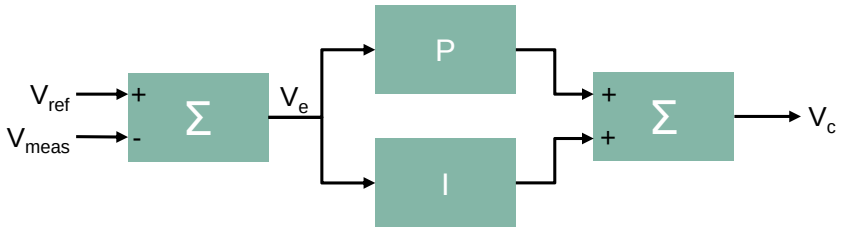


Figure 8.6.: Block diagram of an analog PI control circuit

8.4.1. Schematic design and simulation

Before we can add the controller to the modulator on the breadboard we have to design its schematic. To prevent faults we will also simulate it in LTSpice. Create a new, blank schematic in LTSpice where you can build the controller and test it separately before adding it to the simulation model of the buck converter and simulating the whole system. In this chapter you will have to design many circuits from scratch. If at any time you are unsure how to design the circuit, look it up.

Error Calculation

Exercises:

1. Design a differential amplifier with input voltages U_{ref} and U_{meas} and gain $G_e = 1$.

2. In reality U_{ref} and U_{meas} originate from voltage dividers. In order to prevent loading them use voltage followers at the input.

Hints:

1. You will need 3 OpAmps and 4 resistors for this task.
2. Since errors can be negative, you need a positive and negative supply.
3. Test your circuit in the simulation by connecting voltage sources to *Set* and *Measure*. In order to get a realistic behavior give the voltage sources a series resistance of 10 kΩ.
4. Use the "UniversalOpamp2" as the simulation model for the OpAmps (not "opamp2"!).
5. Make use of labels in LTSpice. Wires that have the same label are connected. Labels are placed with the F4 key.

P and I OpAmps

Exercises:

1. Design an inverting amplifier with the gain $-k_P$
2. Design an inverting integrator with the gain $-k_I$

Hints:

1. For k_P and k_I use the values you determined for the fast controller (with a closed-loop bandwidth of 10 % of the switching frequency).
2. Keep your resistors in the range of 1 kΩ to 10 kΩ and your capacitors in the range of 1 nF to 1 µF.
3. You will need 2 OpAmps, 3 resistors and 1 capacitor for this task.
4. In order to give the integrator an initial condition in LTSpice you can use the `.ic` command.

Summation

Exercises:

1. Design an inverting summing amplifier with a gain of $G_s = -3.3$
2. Explain why the gain has to be -3.3 (think about the modulator).

Hints:

1. You will need 1 OpAmp and 3 resistors for this task.

Simulating the whole system

Exercises:

1. Create a simulation mode of the whole system.
2. Test it with different loads, input and output voltages and make sure that it is stable.
3. Take a look at how the systems responds to a step in the load current. You can either use a current source for this or a resistor and a switch.

Hints:

1. You can use the "Buck_SawtoothPoti"-model from Moodle as a starting point.
2. Simulating the whole circuit including the NE555 may take some time to compute. Therefore, you can replace the NE555 circuit with a voltage source that generates the sawtooth waveform: "PULSE(0 3.3 0 10u 1n 1n 10u)".

8.4.2. Implementation

Exercises:

1. Implement the controller circuit on the breadboard. Make sure that each IC has a 100 nF decoupling-capacitor very close to its supply pins. These capacitors are not necessary in the simulation, because the voltage sources in the simulation are ideal.
2. Connect the controller to the PWM generator as shown in Figure 8.5.
3. The new board cannot deal with a duty-cycle of 100 %! Therefore, it is recommended to connect the output of the inverting summing amplifier to a resistor and a zener-diode which clamps the voltage to 3 V or 3.3 V.
4. Connect your controller to the Mainboard. There is a dedicated connector at the right side where all the required pins can be found.
5. Connect your controller SET input to a potentiometer as shown in Figure 8.7. This will give you a wider range of reference voltages.

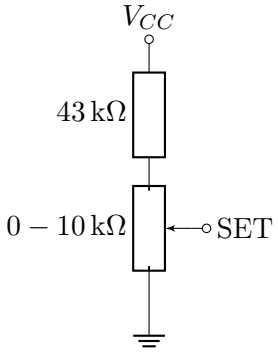


Figure 8.7.: Potentiometer for setting the output voltage setpoint

6. Probe the output voltage, the set-voltage, the error signal and the PWM signal with an oscilloscope.

Hints:

1. Connect V_{meas} from the Mainboard to the Measure input of the controller.

2. Tie all unused pins to the negative supply voltage of the IC (in this case $-5\,\mathrm{V}$). This prevents unwanted switching of the unused OpAmps.

3. To create the negative supply for the OpAmps, two outputs of the power supply can be connected in series:

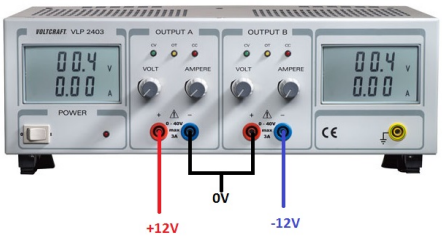


Figure 8.8.: How to wire the power supply to create a positive and negative supply voltage.

4. Look up the pinout of the LM358!

Before turning on your circuit, contact a supervisor to check your wiring!

If you have an error (probably it will not work on the first try) do proper debugging: Check every block individually for errors.

8.5. Tuning the controller (optional)

Exercises:

1. Reduce k_i by a factor of approx. 10 and observe the difference in the controller's response to a load-step. Explain the behavior using the bode-plot.

2. In chapter 5 we said that the crossover-frequency of the open-loop should be around 10 % of the PWM frequency. Back then, we assumed a PWM frequency of 100 kHz. If you were able to produce a PWM frequency of more than 100 kHz you can increase the bandwidth of the controller. Use the script from section 5.2 for this. Afterwards, test the response of your system again and compare it to the previous one.

3. Choose a value for k_i that results in a nice step-response.

8.6. Testing

Exercises:

1. Repeat the measurements from chapter 7 for the analog controller.
2. Compare both controllers. Which one is better and why?

8.7. Analog controller using the Type 2 Compensator

8.7.1. Understanding the Type 2 Compensator

Deriving the circuit

So far we are using are rather complex circuit with 6 OpAmps to build the analog controller. However, it doesn't have to be that complicated! Let's find out how to implement the complete analog controller with just one OpAmp.

To implement the P and the I term we used an inverting amplifier and an inverting integrator:

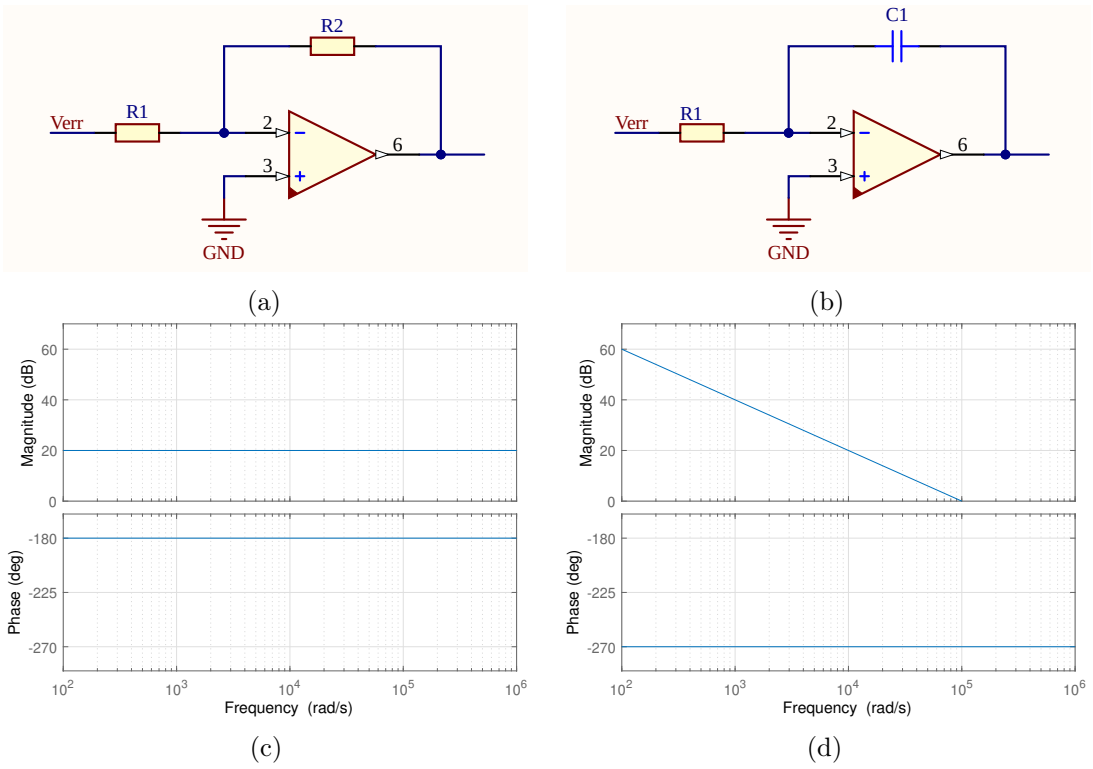


Figure 8.10.: Circuit and bode plot of the inverting amplifier and inverting integrator.

Now, what happens what if we simply combine both circuits like so?

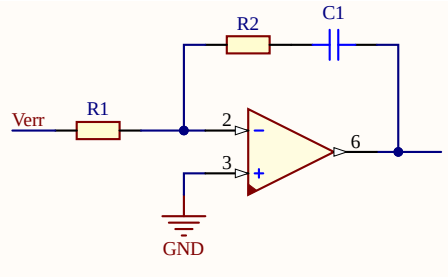


Figure 8.11.: Combining the inverting amplifier and the inverting integrator.

At low frequencies the impedance of C_1 is much higher than the impedance of R_2. Therefore, C_1 dominates and the system behaves like an integrator. At high frequencies the impedance of the capacitor becomes very low and R_2 starts to dominate. Now, the system behaves like an inverting amplifier. We can also see this in the bode-plot:

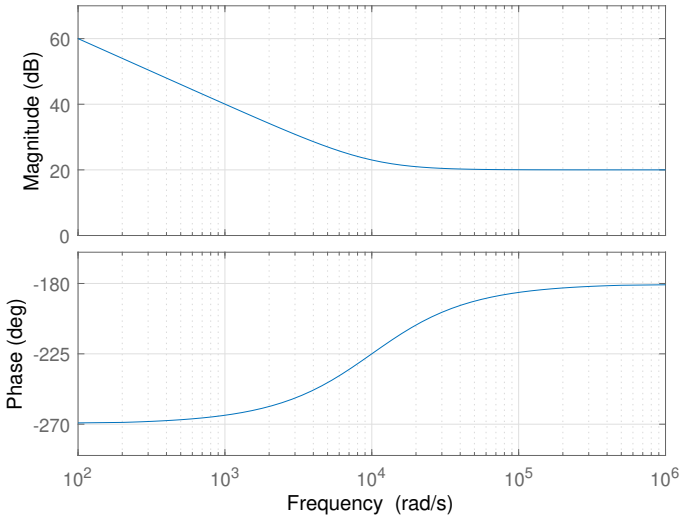


Figure 8.12.: Bode plot of the PI controller.

If you look at chapter 5, this is exactly the bode-plot of the PI-controller[1]: At low frequencies the I term dominates, at high frequencies the P term.

So, we already simplified the circuit from an individual OpAmp for P, I and summer each into one OpAmp for all. Can we also get rid of the differential amplifier? Of course we can! Right now the non-inverting input of the OpAmp is connected to ground. But if we connect it to a reference voltage, we will achieve the desired result. This approach also has the advantage that we no longer need a bipolar supply for the OpAmp. The reference voltage can be easily generated using a voltage divider.

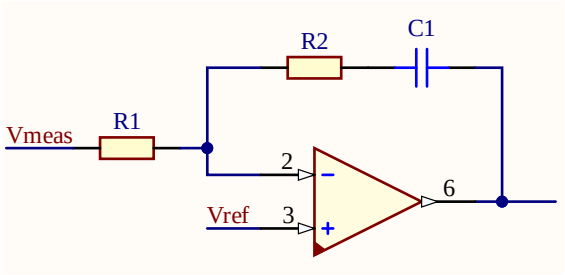


Figure 8.13.: PI controller including error calculation.

[1]With the exception that the phase is off by $-180°$ but we will see that this is actually desirable.

Right now, the whole controller looks as follows:

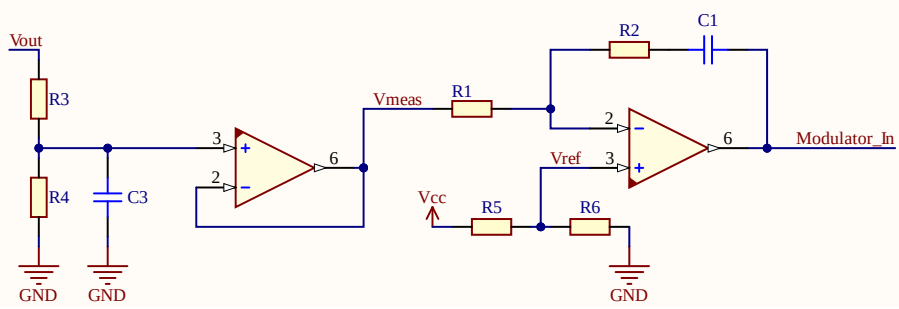


Figure 8.14.: Complete PI controller circuit.

Finally, let's get rid of the voltage follower, by connecting the voltage divider directly to the OpAmp:

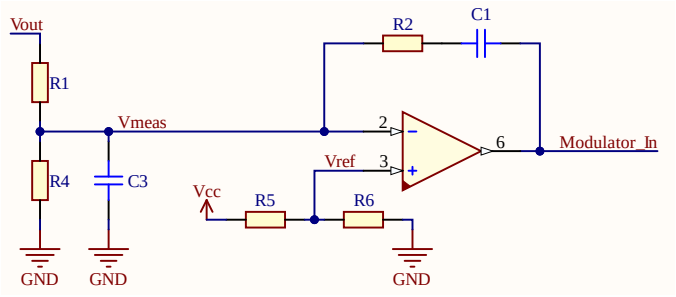


Figure 8.15.: Optimized circuit with just one OpAmp.

Now, the upper resistor of the voltage divider R_1 does enter the transfer function but this is not a problem. We just have to take care of it in our calculations. However, there is one problem: C_3 is useless in this circuit. Remember that it was previously used to perform the low-pass filtering of the measured voltage? But the OpAmp will always keep its inverting and non-inverting input at the same voltage, therefore the voltage across C_3 is constant and it has no effect.
We have to find another way to implement the low-pass filter - we can place it in the feedback path of the OpAmp:

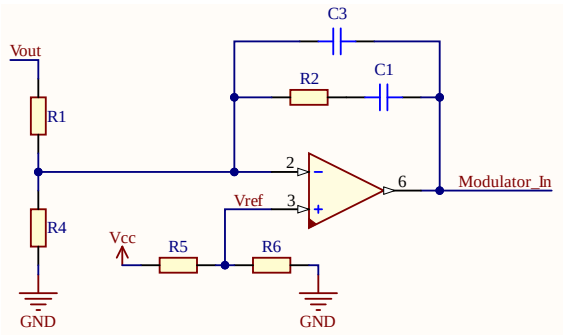


Figure 8.16.: Type 2 Compensator

If $C_3 \ll C_1$, the following happens: At low and medium frequencies everything stays the same. First C_1 dominates forming an inverting integrator, then R_2 dominates and we get the inverting amplifier. However, at high frequencies, the impedance of C_3 will be smaller than the one of R_2, so it starts to dominate and the magnitude drops with $-20\,\mathrm{dB/decade}$ again. That's excatly what we wanted! We successfully designed the whole analog controller with just one OpAmp. The bode-plot also looks as desired:

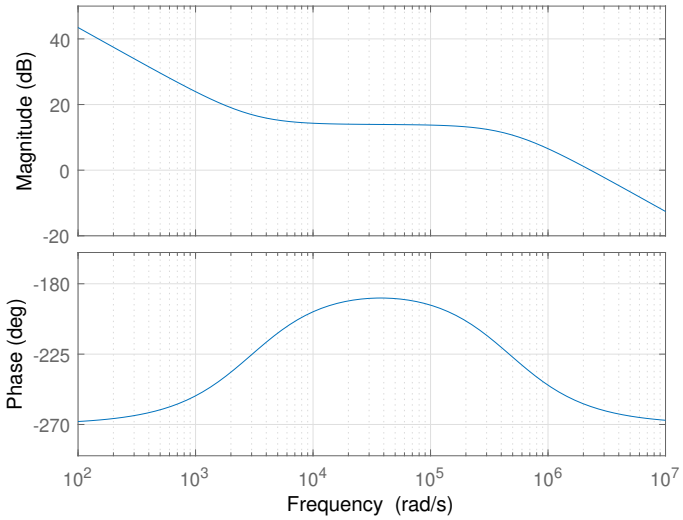


Figure 8.17.: Bode plot of the Type 2 Compensator.

Please keep in mind that we did not do anything to the modulator. The NE555 and the comparator are still needed.

Small-signal model

As the voltage across R_4 is constant, it essentially forms a constant current source. This means that in the small-signal model it is replaced by an open circuit:

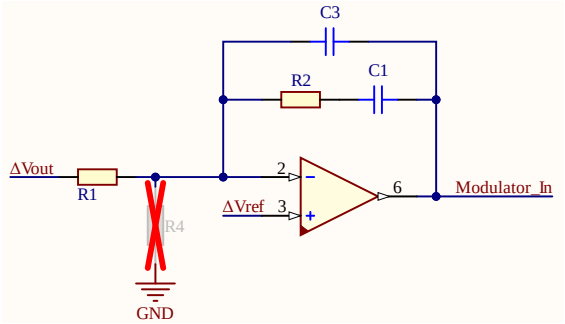


Figure 8.18.: Small-signal model of the Type 2 Compensator

The resulting small-signal block diagram looks as follows:

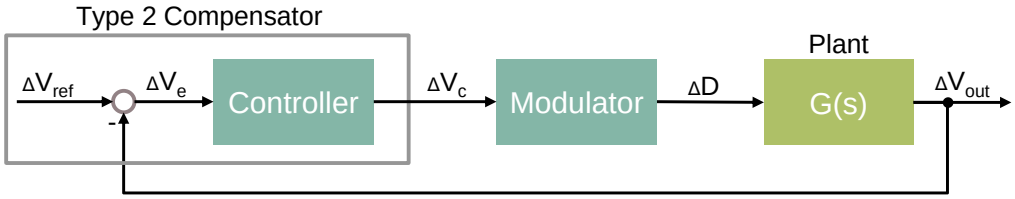


Figure 8.19.: Small-signal block diagram of the Type 2 Compensator.

There is no longer any sensor because V_{out} is fed directly into the Type 2 Compensator. The Type 2 Compensator also handles the "error calculation" as indicated by the grey box. However, there is no dedicated part that does that. Instead, it is taken advantage of the fact that the OpAmp will always keep its inputs at the same voltage. Therefore, the voltage across R_1 is the error voltage $\Delta V_e = \Delta V_{out} - \Delta V_{ref}$.

Exercise:
As the control loop changed, k_P and k_I have to be recalculated. However, just the gain changed. You can use the following formula

$$k_{new} = k_{old} \cdot V_{mod,amp} \cdot K(s = 0) \tag{8.7.1}$$

with the amplitude of the modulator $V_{mod,amp}$ and the DC-gain of the sensor $K(s = 0)$.

8.7.2. Implementation

Now it's time to dimension the circuit elements, simulate the circuit and build it. Let's quickly recall the circuit of the Type 2 Compensator:

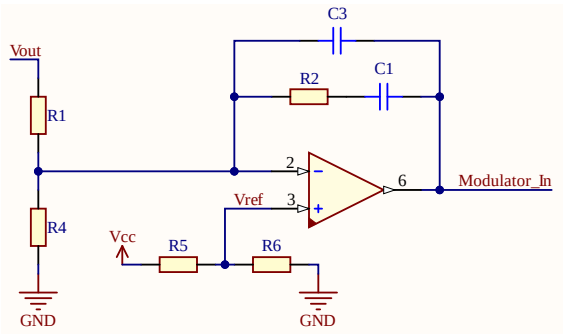


Figure 8.20.: Type 2 Compensator

R_1, R_2, C_1 and C_3 define the transfer function of the controller. Although it is not particularly complicated to determine the transfer function and calculate the poles and zeroes, it is quite tedious which is why we are skipping this part. The transfer function is [11, p. 4]:

$$\begin{aligned} H(s) &= \frac{1 + C_1 R_2 s}{(C_3 + C_1) R_1 s + R_1 C_1 C_3 R_2 s^2} \\ &\approx \frac{1 + C_1 R_2 s}{(C_1 R_1 s)(1 + C_3 R_2 s)} \quad for\ C_1 \gg C_3 \end{aligned} \tag{8.7.2}$$

It has two poles and one zero, where one pole lies at the origin. The location of the pole and zero are [11, p. 4]:

$$\begin{aligned} \omega_{p1} &\approx \frac{1}{R_2 C_3} \\ \omega_{z1} &= \frac{1}{R_2 C_1} \end{aligned} \tag{8.7.3}$$

The gain of the integrator formed by the pole at the origin is:

$$k_I \approx \frac{1}{R_1 C_1} \tag{8.7.4}$$

Exercises:

1. Dimension R_5 and R_6 such that $V_{ref} \approx 1.65\,\mathrm{V}$. This way the OpAmp can nicely pull the feedback network up and down without saturating.
2. Choose R_1 between $1\,\mathrm{k\Omega}$ and $10\,\mathrm{k\Omega}$ and calculate the other element values.
   - Use the new values for k_P and k_I which you calculated in the last section. If you were able to produce a PWM frequency of more than 100 kHz and calculated new controller parameters in the last section you can use them.
   - Place ω_{p1} at $1/2$ of the switching frequency.
   - ω_{z1} can be calculated with $\omega_{z1} = k_i/k_p$. Why?
3. Simulate the circuit and check if it works.
4. Build the circuit on a breadboard. Use a $10\,\mathrm{k\Omega}$ potentiometer for R_4 to set the output voltage.
5. Take care that the PWM duty-cycle never reaches 100%!
6. Connect your controller to the Mainboard and test it. How does it compare to the previous analog controller?

Hints:

1. To understand why $\omega_{z1} = k_i/k_p$ take a look at the transfer function of the classical PI controller and calculate the frequency of its zero.
2. You can use the existing modulator if you make sure that you can still test the old implementation if necessary. Alternatively you can also build the whole circuit from scratch.
3. If you want to be able to produce lower output voltages, add a resistor in series with the potentiometer.
4. The Boost input has to be permanently connected to V_{CC}.

5. This circuit measures V_{out} directly, you don't need V_{meas} from the Mainboard.
6. Tie the unused OpAmp pins to ground.
7. It's not necessary to run all tests from chapter 7 again. Looking at the response to a load-step is sufficient.

Before turning on your circuit, contact a supervisor to check your wiring!

Bibliography

[1] *Serial.write Reference*. Accessed: 2021-12-28. Arduino. URL: https://www.arduino.cc/reference/en/language/functions/communication/serial/write/.

[2] *NTC Thermistors. General technical information*. TDK. Jan. 2018. URL: https://www.tdk-electronics.tdk.com/download/531116/19643b7ea798d7c4670141a88cd993f9/%20pdf-general-technical-information.pdf.

[3] *NCU18XH103D60RB*. Accessed: 2021-06-20. muRata. URL: https://www.murata.com/en-gb/products/productdetail?partno=NCU18XH103D60RB.

[4] Robert W. Erickson and Dragan Maksimovic. *Fundamentals of Power Electronics*. 3rd ed. Springer International Publishing, 2020. ISBN: 978-3-030-43879-1. DOI: 10.1007/978-3-030-43881-4.

[5] Adrian Keil. "Development of a Learning Platform for Switching Regulators". Bachelor's Thesis. Technical University of Munich, 2021.

[6] R. D. Middlebrook and Slobodan Cuk. "A general unified approach to modelling switching-converter power stages". In: *1976 IEEE Power Electronics Specialists Conference*. 1976, pp. 18–34. DOI: 10.1109/PESC.1976.7072895.

[7] Jan Lunze. *Regelungstechnik 1. Systemtheoretische Grundlagen, Analyse und Entwurf einschleifiger Regelungen*. 12th ed. Springer Vieweg, 2020. ISBN: 978-3-662-60745-9. DOI: 10.1007/978-3-662-60746-6.

[8] Henry J. Zhang. *Modeling and Loop Compensation Design of Switching Mode Power Supplies. AN149*. Linear Technology. Jan. 2015. URL: https://www.analog.com/media/en/technical-documentation/application-notes/AN149fa.pdf.

[9] *AN-1889 How to Measure the Loop Transfer Function of Power Supplies. SNVA364A*. Texas Instruments. Apr. 2013. URL: https://www.ti.com/lit/an/snva364a/snva364a.pdf.

[10] Texas Instruments. *LM393B, LM2903B, LM193, LM293, LM393 and LM2903 Dual Comparators. SLCS005AE*. 2020. URL: https://www.ti.com/lit/ds/symlink/lm393.pdf.

[11] SW Lee. *Demystifying Type II and Type III Compensators Using Op-Amp and OTA for DC/DC Converters. SLVA662*. Texas Instruments. July 2014. URL: https://www.ti.com/lit/an/slva662/slva662.pdf.

List of Figures

A. Appendix

A.1. Code

Note: Do not copy code from this PDF! Use the files that are provided on Moodle instead. Sometimes, there are hidden characters in the PDF which result in errors during compilation.

Listing A.1: Arduino code to test the buck converter without control. The duty-cycle is set via button 1 and 2.

```
// Define all Arduino pins
// Digital Out
#define USER_LED1     4
#define USER_LED2     5
#define BUCK_PIN      6
#define BOOST_PIN     9
#define CURRENT_BLANK 10
#define USER_LED3     11
#define USER_LED4     13

// Digital In
#define USER_BTN1     3
#define USER_BTN2     7
#define SHUTDOWN      8
#define USER_SW       12

// Analog In
#define IN_CURR       A0
#define INDUCTOR_CURR A1
#define USER_POTI     A1
#define OUT_CURR      A2
#define VIN_MEAS      A3
#define VOUT_MEAS     A4
#define NTC           A5

// Other Definitions
#define DUTYCYCLE_0   64    // Initial dutycycle
#define PWM_FREQ      50000 // PWM Frequency in Hertz
#define ADC_max       1024  // Maximum value of the ADC
#define AREF          3.3   // Analog reference voltage
```

```
31 #define VOUT_SENSOR   14.7  // Vout = VOUT_SENSOR * Vmeas
32
33 // Global variables
34 uint8_t dutycycle;    // Variable to store the current dutycycle
35
36 // Function to read the output voltage
37 float getVoltage(uint8_t pin) {
38   return analogRead(pin) * AREF / ADC_max;
39 }
40
41 void setup() {
42   // Setup Pins
43   pinMode(BUCK_PIN, OUTPUT);
44   pinMode(BOOST_PIN, OUTPUT);
45   pinMode(USER_LED1, OUTPUT);
46   pinMode(USER_LED2, OUTPUT);
47   pinMode(USER_LED3, OUTPUT);
48   pinMode(USER_LED4, OUTPUT);
49   pinMode(USER_BTN1, INPUT);
50   pinMode(USER_BTN2, INPUT);
51   pinMode(USER_SW, INPUT);
52
53   // Start Serial Connection
54   Serial.begin(115200);
55   Serial.println("Buck-converter program is starting");
56
57   // Initialize all Pins
58   // Initial Condition of Power Stage: Output is connected to GND
59   digitalWrite(BUCK_PIN, LOW);
60   digitalWrite(BOOST_PIN, HIGH);
61
62   // Turn on all LEDs
63   digitalWrite(USER_LED1, HIGH);
64   digitalWrite(USER_LED2, HIGH);
65   digitalWrite(USER_LED3, HIGH);
66   digitalWrite(USER_LED4, HIGH);
67
68   // Set PWM-Frequency
69   setAnalogWriteFrequency(BUCK_PIN, PWM_FREQ);
70
71   // Initialize dutycycle
72   dutycycle = DUTYCYCLE_0;
73
74   // Start the system
75   delay(100);   // Small delay to ensure proper power-up
```

```
76    analogWrite(BUCK_PIN, dutycycle);
77  }
78
79  void loop() {
80    // USER_BTN2 increases the duty-cycle
81    if (digitalRead(USER_BTN2)) {
82      dutycycle += 16;
83      Serial.print("Dutycycle: ");
84      Serial.println(dutycycle);
85      analogWrite(BUCK_PIN, dutycycle);
86      // Blink LED 4
87      digitalWrite(USER_LED4, HIGH);
88      delay(100);
89      digitalWrite(USER_LED4, LOW);
90      // Delay for debouncing
91      delay(500);
92    }
93    // USER_BTN1 decreases the duty-cycle
94    if (digitalRead(USER_BTN1)) {
95      dutycycle -= 16;
96      Serial.print("Dutycycle: ");
97      Serial.println(dutycycle);
98      analogWrite(BUCK_PIN, dutycycle);
99      // Blink LED 3
100     digitalWrite(USER_LED3, HIGH);
101     delay(100);
102     digitalWrite(USER_LED3, LOW);
103     // Delay for debouncing
104     delay(500);
105   }
106   // Print the output voltage
107   Serial.print("Vout: ");
108   Serial.print(getVoltage(VOUT_MEAS)*VOUT_SENSOR);
109   Serial.println(" V");
110   // Short delay such that serial monitor is not completely flooded
111   delay(50);
112 }
```

Listing A.2: Matlab script to compare two transfer functions

```
% Script to compare two transfer functions
% Preliminary: Clear all variables, clear command window, close all figures
clear
clc
close all

% Change the output format to make numbers easier to read
format shortEng;

% Create a special variable s that you can use in the following transfer
% function. Alternatively you could just use g = tf(numerator,denominator).
s = tf('s');

% Idealized modeling
Vin = 10;
w0 = 1e5;
Q = 10;
G_ideal = tf(Vin/(s^2/w0^2+s/(w0*Q)+1));

% Refined modeling
w0 = 1e4;
Q = 2;
wz = 3e4;
G_refined = tf(Vin*(1+s/wz)/(s^2/w0^2+s/(w0*Q)+1));

% Compare in the bode plot
figure;
hold on
bode(G_ideal, {1e2,1e7})
bode(G_refined, {1e2,1e7})
grid on
title('Bode plot of the idealized and the refined modeling')
legend('idealized', 'refined')
hold off

% Compare the step responses
figure;
step(0.1*G_ideal, 0.1*G_refined)
grid on
title('Step response of the idealized and the refined modeling (to a 10%
    step)');
legend('loop comp','loop PID')
```

A.2. XMC1100 Boot Kit

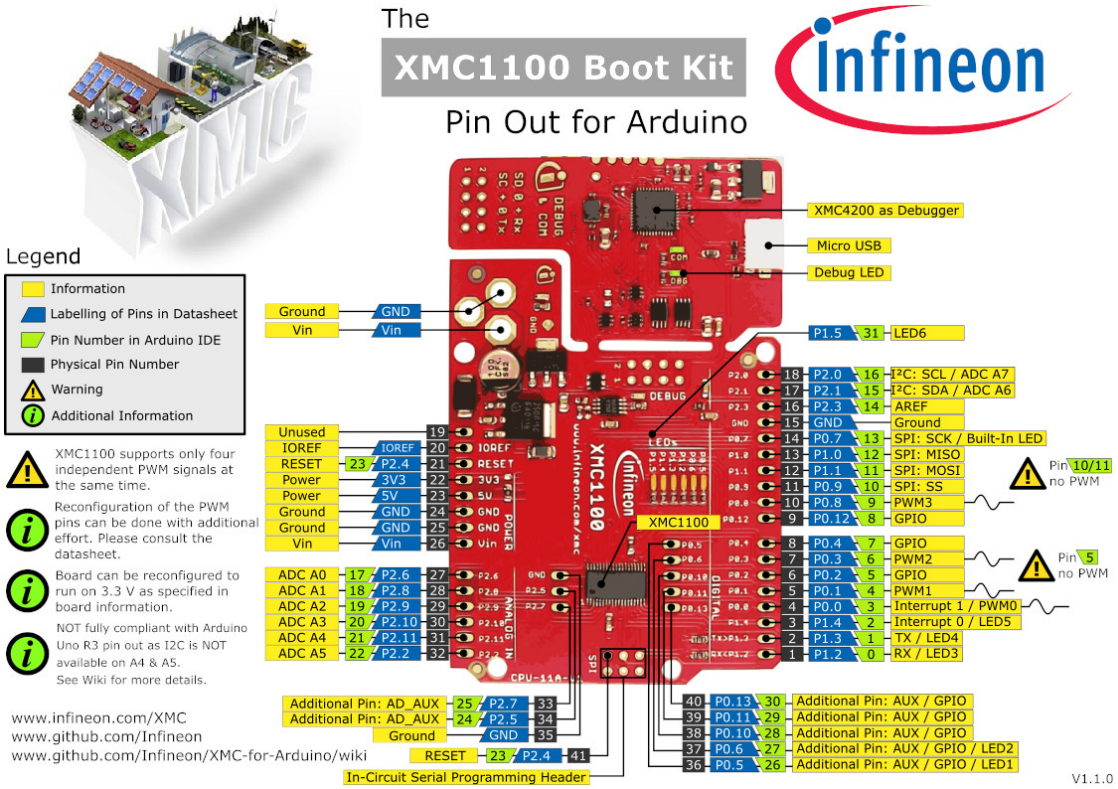


Figure A.1.: Pin-out of the XMC1100 Boot Kit.