PROJEKT LABOR

Spannungswächter WS 05/06

Inhaltsverzeichnis

1. Allgemeines zum Projekt

1.1. Projektbeschreibung

Thema des Projektlabors des Wintersemesters 2005/2006 war es, einen „Spannungswächter“ zu planen und aufzubauen.

Funktionalität und Eigenschaften des Spannungswächters

Aufgabe des Spannungswächters ist es die Netzqualität zu überwachen. Dabei sollen Über- und Unterspannungen detektiert, Stromausfälle erkannt, Minimal- und Maximalwerte gespeichert und Frequenzabweichungen detektiert werden. Der Spannungswächter ist dank seiner Größe als gut sichtbares Ausstellungsstück zur Visualisierung der netzspezifischen Daten voll einsatzfähig.

Über- und Unterspannungen werden entlang eines 14-Reihigen Bargraphen mit verschiedenfarbigen LEDs visualisiert. Da die Netzspannung nur zwischen -10% und +6% von 230V schwanken darf, wird nur ein für eine mögliche Über- oder Unterspannung relevanter Bereich angezeigt.
Minimal- und Maximalwerte der Spannung werden über Siebensegmentanzeigen ausgegeben ebenso wie die aktuelle Frequenz, die mit einer Genauigkeit von einem Millihertz anzeigt wird. Durch eine Kooperation mit dem Studiengang Industriedesign der UdK Berlin wurden designspezifische und ökologische Aspekte bei der Planung und Realisierung des Gehäuses des Spannungswächters mit beachtet und umgesetzt.

1.2. Aufteilung der Gruppen

Abbildung 1: Alle Teilnehmer des Projektlabors

Um ein effizienteres Arbeiten zu ermöglichen, war es unabdingbar, eine Gruppeneinteilung nach Baugruppen des Spannungswächters vorzunehmen. Diese Einteilung war vorerst den Teilnehmern des Projektlabores vorbehalten. Im Einvernehmen mit den Betreuern wurde das Projekt in folgende Gruppen eingeteilt.

1.Netzteil
Das Netzteil versorgt sämtliche Schaltungsteile mit Spannung. Hinzu kommt die Wandlung und Bereitstellung sowie eine galvanischen Trennung der zu messenden Signale.

2.Spannungspegel und Min-Max-Wertmessung
Mögliche Über- und Unterspannungen werden schaltungstechnisch registriert und Minimal -und Maximalwerte werden in Form eines seriellen Datenpakets an die Anzeige weitergeleitet.

3.Frequenzmessung
Die aktuelle Frequenz wird gemessen und in Form eines seriellen Datenpakets an die Anzeige weitergeleitet. Es können außerdem Minimal- und Maximalwerte abgerufen werden.

4.Akkubetrieb
Im Falle eines Stromausfalls wird der Spannungswächter über einen Akku versorgt. Geladen wird dieser während des Netzbetriebes.

5.Anzeige und Gehäuse
Über den Bargraphen und die 7-Segmentanzeigen werden die aktuellen Messwerte angezeigt. Die Bedienungselemente werden zum Schutz vor Manipulationen an der Rückseite des Gerätes angebracht.

Die Baugruppen werden als Platinen in einem 19-Zoll-Einschub in der Rückseite des Gehäuses integriert. Darunter werden Netzteil und Akku angeordnet.

1.2.1. Gruppe 1 – Netzteil

Im Folgenden werden die Mitglieder der Gruppe und deren sozialer Kommentar zur Arbeit im Projektlabor vorgestellt.

Ahmed Boukith (Partybeauftragter)

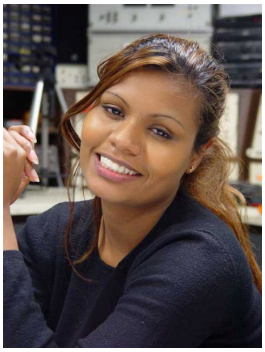
Sozialer Kommentar:
Hat das Projektlabor vorzeitig beendet.

Andreas Schuster (Webseite)

Sozialer Kommentar:
„Die Atmosphäre bei uns im Labor war eigentlich immer recht angenehm. Es war kein Problem mit den anderen auf einer freundschaftlichen Basis zu kommunizieren. Im Nachhinein muss ich aber sagen, dass es für ein produktiveres Arbeiten von Vorteil gewesen wäre, wenn ich beharrlicher auf gewisse Dinge bestanden bzw. bestimmte Sachen früher und energischer angesprochen hätte. Es wäre dann vielleicht für den ein oder anderen nicht immer so angenehm gewesen, aber solange man dabei nicht persönlich wird, ist das, so denke ich zumindest, schon ok. Private Dinge sind dann eh noch mal etwas ganz anderes. Auf jeden Fall kann ich sagen, dass das Projektlabor nicht nur aus fachlicher sondern auch aus persönlicher Sicht sehr lehrreich war.“

Carine Homssi Kambou (Ordnerbeauftragte)

Sozialer Kommentar:
„Wir konnten uns in der Teilgruppe gut verstehen. Am Anfang gab das Problem, dass wir mit einander nicht richtig kommuniziert haben, und mussten daher einige Dinge mindestens mehr als ein Mal machen. In der gesamten Gruppe war es auch ganz cool. Unser Betreuer hat dafür gesorgt, dass jeder seinen Platz in Labor findet, und dass es vor allem im Labor locker wird, das was die arbeit wirklich erleichtert hat.“



Irene Selvanathan (Abschlussberichtbeauftragte)

Sozialer Kommentar:
„In einer kleinen Gruppe in einem Projekt mitzumachen, dass ein so engen Zeitplan vorsieht, hat mich um einige Erfahrung reicher gemacht. Man merkt gar nicht wie schnell drei Monate vergehen können. Ich habe allerdings sehr viel dazugelernt und konnte mein technisches Wissen um einiges erweitern…in der Praxis läuft doch einiges anders als in der Theorie!“

Marc Sonnabend (Gehäusebeauftragter)

Sozialer Kommentar:
Hat das Projektlabor vorzeitig beendet.

Svetomir Najdanovic (Schnittstellenbeauftragter)

Sozialer Kommentar:

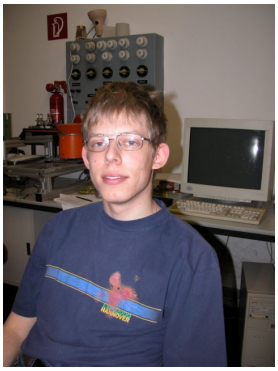
Johannes Twittmann (Betreuer)

1.2.2. Gruppe 2 – Peak- und Stromausfalldetektion

Im Folgenden werden die Mitglieder der Gruppe und deren sozialer Kommentar zur Arbeit im Projektlabor vorgestellt.

Stephan Körber (Schnittstellenbeauftragter)

Sozialer Kommentar:
„Das Projektlabor war eine sehr interessante, wenn auch ziemlich arbeitsaufwendige Erfahrung, die auf Grund des angenehmen Gruppenklimas sehr viel Spaß machte. Leider war der eingebrachte Ehrgeiz nicht bei allen Leuten gleich hoch, was meiner Meinung nach mit unter einen etwas negativen Eindruck hinterließ.“

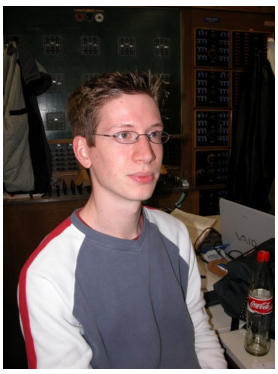


Andreas Bock

Sozialer Kommentar:
„Das Arbeiten im Projektlabor war sehr angenehm. Wir haben uns für die Entwicklung viel Zeit genommen, dadurch mussten wir nicht viel aufbauen und danach wieder abreißen. Meinungsverschiedenheiten wurden sachlich ausgetragen, die Stimmung war harmonisch. Trotzdem finde ich, dass wir die Zeit effektiv genutzt haben. Nur das erst-am-nächsten-Tag wieder nach Hause kommen wurde mit der Zeit ein wenig nervig, aber es gibt ja Nachtbusse. Zu kritisieren wären nur die z.T. etwas unklaren Aufgabenstellungen.“

Peer Günther (Gehäusebeauftragter & Webseite)

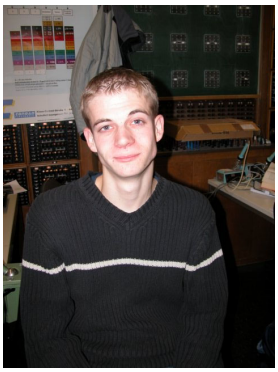
Sozialer Kommentar:
„Ich fand die Gruppenarbeit im Labor als sehr gut. Meine Gruppe funktionierte im Bezug auf die Zusammenarbeit sehr gut. Auch der Spaß kam nicht zu kurz. Negativ war die Integration von Salomon in die schon bestehende Gruppe, da er die 2 ersten Termine versäumte. Bis zum Schluss konnte er sich nicht richtig ins Gruppenleben einordnen, was sehr schade war. Die Kommunikation mit den anderen 4 Gruppen verlief in den meisten Fällen problemlos.“



Florian Unverferth (Abschlussberichtbeauftragter)

Sozialer Kommentar:

„Ich habe auf jeden Fall im Projektlabor eine Menge gelernt und Spaß hat es auch gemacht. Das lag größtenteils an der guten Gruppe in der ich war. Ich habe zum ersten Mal die komplette Entwicklung eines Projekts von der Idee an mitbekommen. Neu für mich war auch das Zusammenarbeiten innerhalb der Gruppe sowie auch das gruppenübergreifende Arbeiten, welches sich manchmal als schwierig erwies. Ich denke, dass das am Anfang hauptsächlich an den nicht ganz klar formulierten Aufgabenbereichen der Gruppen lag.“



Stefan Schulz (Partybeauftragter)

Sozialer Kommentar:

„Während des gesamten Projektes hatten wir sowohl untereinander als auch mit unserem Tutor ein sehr gutes und ausgelassenes Arbeitsklima, was zu einer entspannten Arbeitsbewältigung beitrug, bei der es auch nicht negativ aufgenommen wurde, wenn mal länger gemacht werden musste. Das Projekt hat mir persönlich das Fach Elektrotechnik näher gebracht und es aufgrund der praktischen Arbeit auch wesentlich interessanter gemacht, so dass ich Spaß an der Arbeit gefunden habe. Leider konnte ich keinen wirklichen Kontakt zu Salomon aufbauen.“

Hannes Neubert (Ordnerbeauftragter)

Sozialer Kommentar.

„Da wir uns größtenteils schon von früheren Vorlesungen und Tutorien kannten war das Arbeiten in der Gruppe von Anfang an sehr entspannt und familiär. Einzige Ausnahme stellte der siebte im Bunde da: Salomon. Er hat ein wenig Pech gehabt in diese Gruppe zu kommen. Da er uns nicht kannte und wir ihn nicht, herrschte Anfangs eine Kluft zwischen ihm und der Rest der Gruppe. Doch man versuchte ihn in die Gruppe zu integrieren, so dass er sich nicht so ausgegrenzt fühlen musste. Das Salomon dadurch nicht immer voll motiviert war kann ich daher verstehen. Das Gesamtklima in der Gruppe 2 würde ich damit trotzdem als sehr gut bezeichnen. Auch die Kontakte zu den anderen Gruppen wurden gut gepflegt und man kam mit allen aus.“

Salomon Dongho

Sozialer Kommentar:

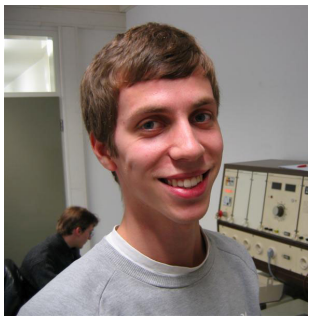
„Das Projektlabor war ein besonderes Erlebnis für mich, weil ich mein Wissen über die Grundlage der Elektrotechnik sowohl praktisch als auch theoretisch erweitern konnte. Aufbau der Schaltungen durch „PSpice“, ihr Verhalten, praktische Anwendungen des integrierten Schaltkreises und die Erstellung von Platinen mit Hilfe des Programms „EAGLE“ waren die Schwerpunkte, die ich neu gelernt habe. Am Anfang fiel es mir schwer, mit den Anderen in der Gruppe zusammen Überlegungen aufzustellen. Nach einiger Zeit konnte ich schon ganz gut mitmachen, wodurch ich mein Wissen von den letzten Semestern erneut anwenden konnte. Das fand ich sehr interessant. Außerdem hat mir die Zusammenarbeit mit den anderen Mitgliedern gefallen. Es gab Fälle, wo wir alle einfach nicht weiter kommen konnten aber dann waren unsere Betreuer immer für uns da, um uns weiter zu unterstützen und zu leiten. Ich bin dankbar für die Betreuer, die uns geholfen haben, weil wir ohne sie nicht weiter kommen konnten. Die Informationen im Internet waren auch sehr hilfsreich für uns.

Das Projektlabor empfehle ich jedem Student der Elektrotechnik, weil man am Ende wirklich etwas lernt, egal auf welches Niveau man steht. Man lernt etwas Neues und diese Erfahrung finde ich sehr wichtig für das weitere Studium.“

Andreas Deml (Betreuer)

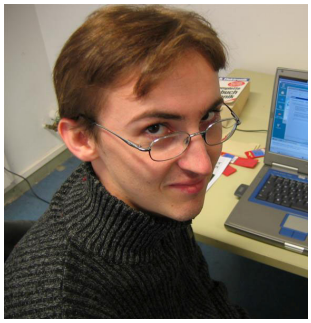
1.2.3. Gruppe 3 – Frequenzmessung

Im Folgenden werden die Mitglieder der Gruppe und deren sozialer Kommentar zur Arbeit im Projektlabor vorgestellt.



Constantin Wiesener (Gehäusebeauftragter)

Sozialer Kommentar:



Eugen Dischke

Sozialer Kommentar:

Jürgen Funck (Schnittstellenbeauftragter)

Sozialer Kommentar:

Martin Koletzko (Ordnerbeauftragter)

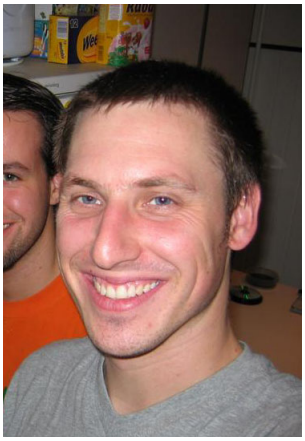
Sozialer Kommentar:

Paul Yanick Tcheg (Abschlussberichtbeauftragter)

Sozialer Kommentar:

Raoul Tsafack Fofe (Partybeauftragter)

Sozialer Kommentar:



Stefan Seifert (Betreuer)

1.2.4. Gruppe 4 – Anzeige + Effektivwert

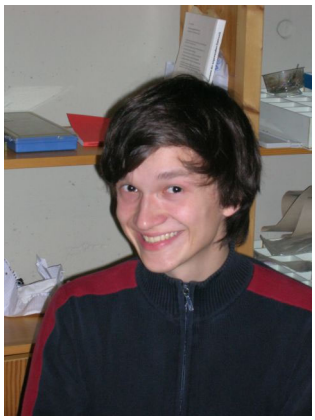
Im Folgenden werden die Mitglieder der Gruppe und deren sozialer Kommentar zur Arbeit im Projektlabor vorgestellt.

Daniel Hahn (Abschlussberichtbeauftragter)

Sozialer Kommentar:
„Ich bin völlig zufrieden mit der Atmosphäre in der Gruppe, die Zusammenarbeit klappt ganz gut, man lernt viel Neues und es wird wirklich so etwas wie Projektmanagement und Gruppenarbeit trainiert. Auch die Kommunikation mit den anderen Gruppen läuft reibungslos. Das ganze Projekt ist zwar anstrengend, macht aber trotzdem Spaß.“

Leopold Georgi (Schnittstellenbeauftragter)

Sozialer Kommentar:
„Ich bin prinzipiell mit unserer Gruppenatmosphäre sehr zufrieden, wir versuchen uns gegenseitig zu helfen und kommen gut miteinander aus, können aber auch jeder für uns selbst arbeiten. Alle sind fleißig und bemühen sich möglichst viel in der vorgesehenen Zeit zu schaffen, sind aber auch bereit weitere Freizeit in das Projekt zu stecken. Am Anfang ging es etwas schleppend, da wir uns eine Menge Wissen noch aneignen mussten und besonders in unserer Gruppe großer Diskussionsbedarf bestand, da wir es als Anzeigegruppe ja Allen recht machen müssen. Am Ende der Projektzeit ist mir besonders aufgefallen, dass die Kommunikation, das gegenseitige Verständnis und auch der menschliche Respekt zwischen einigen Teilnehmern des Projektlabors und Betreuern stark zu wünschen übrig ließ, das ist meine persönliche Meinung und ich will niemanden damit beleidigen!“



Moritz Göldner (Gehäusebeauftragter)

Sozialer Kommentar:
Da unsere Gruppe in vier seperate Kleingruppen aufgeteilt wurde kam es kaum zur Gruppenarbeit. Die Angelegenheiten, die im Team gelöst werden mussten haben mir mit den Personen dann aber auch Spaß gemacht. Das Verhältnis zu den meisten Betreuern war sehr gut und freundlich. Hervorzuheben ist auch das zeitliche Engagement der Betreuer, dass das eines Tutors bei weitem überschreitet.
Gestört hat mich die (fachliche) Überheblichkeit einiger Tutoren sowie die Art, wie das Forum von einigen Teilnehmern und Tutoren

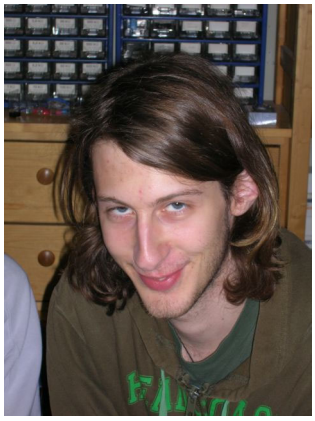
genutzt wurde.
Abschließend lässt sich sagen, dass das Projektlabor zwar eine sehr anstrengende und zeitaufwändige Veranstaltung ist, der Lerneffekt jedoch durch die Diskussionen und Fehlersuche bei weitem größer ist als in einem normalen Tutorium.

Philip Jung (Ordnerbeauftragter)

Sozialer Kommentar:

Unter sozialer Bewertung verstehe ich eine Beurteilung der Zusammenarbeit zwischen Studenten untereinander, Studenten zusammen mit den Betreuern und eine Bewertung der Arbeitsatmosphäre. Die Zusammenarbeit möchte ich unter verschiedenen Gesichtspunkten bewerten. Zum einen wäre da die Aufgabenstellung der Tutoren bzw. Obertutoren. Ich bin der Meinung, dass Anforderungen an unsere Schaltungen rechtzeitig gestellt werden müssen, so dass wir dem Zeitplan entsprechend arbeiten können. Werden die Aufgaben zu spät gestellt und wir zwei Termine vor dem Stöpseltermin, zu einem Zeitpunkt, zu dem unsere Schaltung noch nicht funktioniert hat, geschweige denn eine Platine geätzt war, dazu aufgefordert noch eine Schaltung für ein Zusatzfeature zu entwerfen, so finde ich das reichlich spät und unorganisiert von den Tutoren. Man sollte berücksichtigen, dass wir wenig Fachkenntnisse haben und keine Erfahrung mit dem Arbeitsablauf der Entwicklung eines Produktes. Schleichen sich irgendwo Fehler ein, so benötigen wir viel Zeit, diese zu beheben. Viele Fehler können wir nicht einmal erkennen, weil wir von verschiedenen möglichen Fehlerquellen noch nie etwas gehört haben. Man soll uns experimentieren lassen, auch Fehler begehen lassen, aber rechtzeitig eingreifen und uns den richtigen Weg oder vielmehr einen funktionierenden Weg zeigen, damit wir nicht die Motivation verlieren. Es ist, mit Ausnahme einiger Heimwerker unter uns natürlich, unser erstes Projekt gewesen. Wenn man uns mit Rat und Tat zur Seite steht, dann ist es Sinnvoll, wenn man uns die Fragen selbst beantworten lässt und nicht jede Antwort gleich serviert. So denken wir selbst nach und lernen, wie wir uns systematisch einen Lösungsweg selbst herleiten können, was auf lange Sicht und die vielen Fehler, die uns noch passieren werden gesehen, wesentlich effektiver ist, als wenn wir die vorgekaute Lösung eines Problems im Kurzzeitgedächtnis speichern und nach zwei Tagen vergessen haben. Es gibt aber eine Grenze, ab deren Überschreitung ich behaupten würde, das Pädagogen- Schüler- Spiel wird übertrieben. Ich möchte zur Erklärung meiner Ansicht an dieser Stelle das Bild eines Baumes mit vielen Ästen verwenden, auf dem ein Schüler, ein fast Blinder steht, und die Aufgabe vom Lehrer gestellt bekommt, der unten steht und ihm Anweisungen gibt, den einzigen Apfel, der am Baum ganz oben im Wipfel hängt zu pflücken. Der Apfel sei also die Lösung der Aufgabe, der Schüler jedoch fast blind. Gibt der Lehrer dem Schüler klare Anweisungen, mit welchem Fuß er auf welchen Ast treten muss, so erreicht er sicher sein Ziel. Verwirrt er den Schüler aber zu sehr mit Hinweisen, so kann es passieren, das dieser nicht mehr versteht, was der Lehrer meint und einen Fehltritt macht. Er stürzt kläglich vom Baum und muss von ganz von vorne anfangen. Dann möchte ich loswerden, dass ich sehr sinnvoll finde, dass wir unsere Arbeitsabläufe so professionell wie möglich gestalten sollen und genau arbeiten müssen, so wie es im späteren Berufsleben auch stattfinden wird. Was allerdings im Projektlabor im Bezug auf eine professionelle Produktion nicht berücksichtigt wurde, ist das Prinzip „ Zeit ist Geld“ oder „ Hat man noch wenig Zeit, so muss man als wichtigste Priorität termingerecht arbeiten und den schnellsten Lösungsweg für offen stehende Probleme einschlagen. Als Paradebeispiel dafür, dass nicht nach diesem Prinzip gearbeitet wurde nehme ich die in unseren Kreisen mittlerweile berüchtigte Platine mit ca. 600 Leuchtdioden und insgesamt 1000 Bauteilen in diese Bewertung mit auf. Es wurde

absolut darauf bestanden, dass wir die ganzen Leuchtdioden mit auf eine Ätzplatine nehmen, also das Board dafür entwerfen mussten. Das ergab ein Soloprojekt mit mehr als 40 Stunden Arbeit. Dabei entdeckten wir einen Fehler im Scematic von Eagle, unserem Platinenprogramm und mussten nach 15 Stunden arbeit noch einmal von vorne beginnen. Muss es wirklich sein, dass einige Tutoren ihren Perfektionismus nicht für einen Moment besiegen können und unseren Vorschlag unterstützen können, die Dioden einfach auf eine Lochplatine zu setzen. Natürlich wird das bei Siemens nicht so gemacht, aber bei Siemens gibt's auch Kohle fürs arbeiten und man hat nicht noch fünf andere Kurse zu bestehen, wie bei uns im Grundstudium. Ich sehe ein, dass man Fehler machen muss, die Zeitaufwand mit sich bringen, damit man sich ganz bestimmt einprägt, was man falsch gemacht hat, um diesen Fehler danach nie mehr zu begehen. Aber muss ein Fehler mit fünfzehn Überstunden bestraft werden? Des weiteren möchte ich mich der Meinung von Moritz anschließen. Es gab auch sehr viele positive Aspekte, aus denen ich viel gelernt habe. Auch die Arbeit unseres Tutors fand ich sehr gut..



Philipp Maier

Sozialer Kommentar:

„Ich persönlich muss sagen, dass das Projektlabor für mich eine interessante Erfahrung war. Ich halte das Projektlabor besonders im Hinblick auf Sozialkompetenz und deren Fortbildung für besonders hilfreich. Allerdings glaube ich das die Ausbildung der fachlichen Kompetenz vernachlässigt wird. Die Entwicklung des Projektes habe ich mit besonderem Interesse verfolgt. Es war interessant zu sehen wie euphorisch wir das Projekt angegangen sind und wie schnell wir von der Machbarkeit auf den Boden der Tatsachen zurückgeholt worden sind. Die Zusammenarbeit in der Gruppe habe ich als überwiegend angenehm empfunden. Es hat mir Spaß gemacht den Planungsprozess zu moderieren und Vorträge zu halten. Ich muss zugeben das mich die Probleme in der heißen Phase des Projektes psychisch sehr belastet haben. Aber mit jedem Fehler den ich in den Griff bekommen habe ging es mir wieder besser: »Kommt dem Ingenieur die Aufgabe lösbar vor hebt das die Stimmung im Labor« (Dietrich Drahtlos). Die Zusammenarbeit mit den anderen Gruppen verlief meiner Ansicht fast reibungslos. Hin und wieder kam es zu Missverständnissen die allerdings keine gravierenden technischen Schwierigkeiten nach sich zogen.“

Stefan Straube (Betreuer)

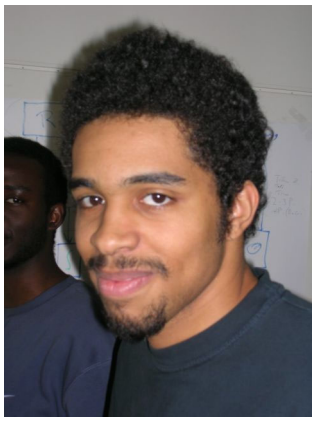
Marion Stahel
(Design)

1.2.5. Gruppe 5 – Unterbrechungsfreie Stromversorgung

Arbeitsklima

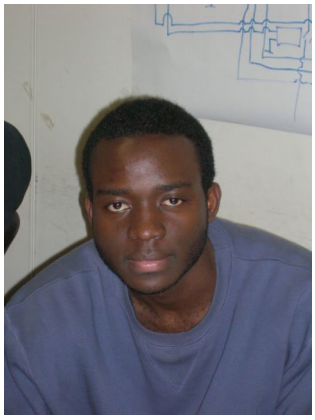
Zunächst bestand unsere Gruppe aus sechs Mitgliedern, nach ein Paar Wochen jedoch entschied sich ein Mitglied dazu, dass Projektlabor erst im nächsten Semester zu belegen, weshalb wir dann auf fünf schrumpften.
Am Arbeitsklima änderte sich aber nichts und es war sehr angenehm in der Gruppe zu arbeiten. Es gab keine Streitereien oder Probleme in der Verständigung. Jedes Gruppenmitglied war stets hilfsbereit und wenn jemand ein Problem mit der Schaltung oder dem Verständnis hatte, fand sich immer jemand, der helfen konnte.
Manchmal ging die Arbeit ein wenig schleppend voran und vor allem bei der Fehlersuche machte sich Lustlosigkeit breit, wenn der Fehler nicht gleich zu finden war.
Mit der Betreuerin kamen wir alle gut klar, sie musste uns allerdings manchmal etwas antreiben, weil wir den Zeitplan nicht immer fest vor Augen hatten. Aber auch das störte die lockere Atmosphäre in keiner Weise.
Die Vereinbarung von Zusatzterminen stellte sich manchmal als schwierig heraus, weil nicht immer jemand im Labor war, wenn wir Zeit hatten. Es kam auch vor, dass Zusatztermine sehr kurzfristig verabredet wurden, weshalb nicht immer alle da waren. Die Arbeit hat sich aber trotzdem gut auf alle verteilt.

Im Folgenden werden die Mitglieder der Gruppe und deren sozialer Kommentar zur Arbeit im Projektlabor vorgestellt.



Chenueh Abdongwa Florian Theurer (Ordnerbeauftragter)

Sozialer Kommentar:



Ezechiel Mouni Ngabmen

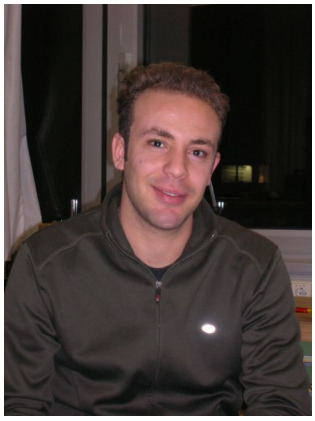
Sozialer Kommentar:

Fabian Cordes (Schnittstellenbeauftragter)

Sozialer Kommentar:

Fabian Undi (Gehäusebeauftragter)

Sozialer Kommentar:



Sami Ben Abdallah (Partybeauftragter)

Sozialer Kommentar:
Hat das Projektlabor vorzeitig beendet.

Sven Kriener (Abschlussberichtbeauftragter)

Sozialer Kommentar:
„Das Projektlabor war eine sehr wertvolle und interessante Erfahrung.Man konnte Pratisches als auch Teoretisches Lehren. Die Gruppenarbeit hat sehr gut funktioniert und jeder hat dort geholfen wo er konnte.
Leider benötigte das Projektlabor einen sehr großen Zeitaufwand, so dass andere Sachen manchmal darunter Leiden mussten.“

Kathleen Jerchel (Betreuerin)

2. Gruppe 1

2.1. Aufgabe der Gruppe

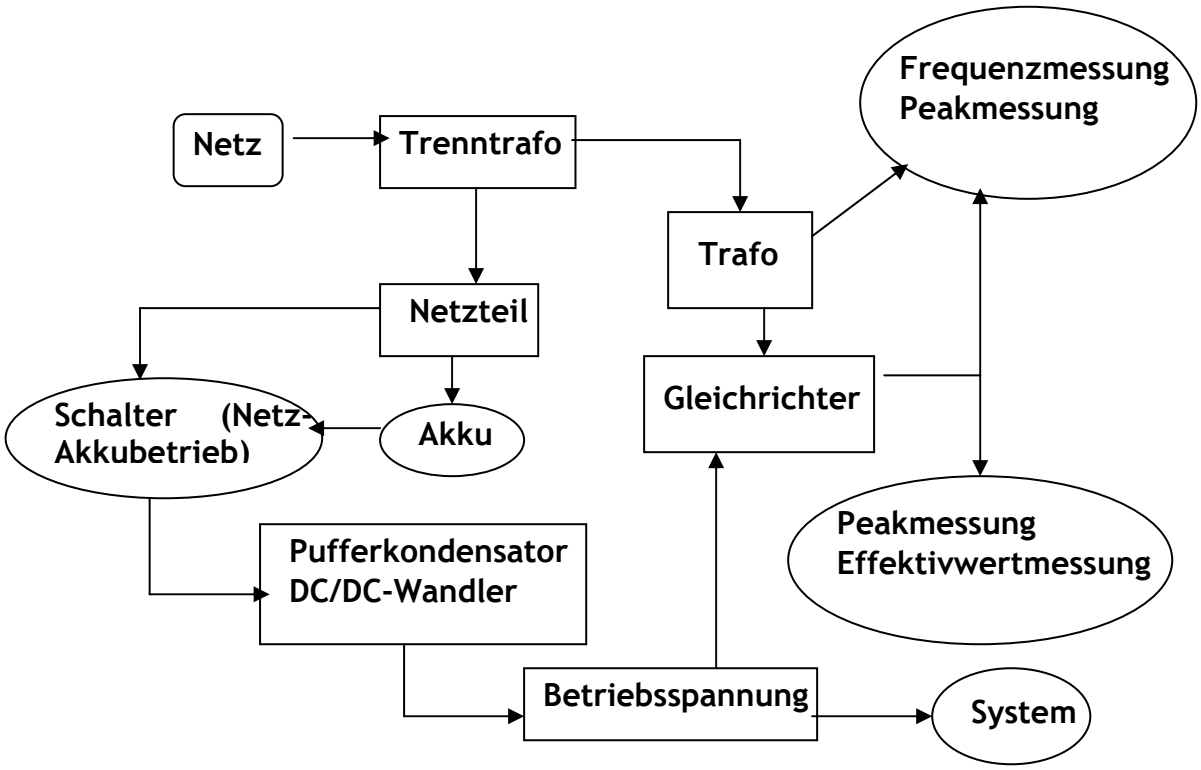
Zu jedem Gerät gehört auch ein entsprechendes Netzteil (NT), welches bestimmten Anforderungen entsprechen muss. Der Spannungswächter brauchte folgende Anforderungen an das Netzteil:

Das NT muss den Akku/USV mit einer Spannung von 30V DC und 3A Strom versorgen und im Falle eines Stromausfalls in der Lage sein weitere 50ms. Strom von 3A zu Garantieren (Umschalt- und Definitionszeit des Akkus);

Für die notwendigen Messungen ist zudem ein Präzisionsgleichrichter notwendig;

Diese Messgeräte benötigen unterschiedliche Spannungen dazu sind DC/DC-Wandler nötig.

Blockschaltbild



Die Aufgaben der Gruppe NT

Aufgaben der anderen Gruppen

Gruppenaufteilung

NT -> Svetomir Najdanovic und Irene Selvanathan

Gleichrichter -> Ahmed Boukith und Andreas Schuster

DC/DC-Wandler -> Carine Homssi Kambou und Marc Sonnabend

2.2. Das Netzteil

Anmerkungen

Aufgabe dieser Teilgruppe war es die Netzspannung in eine für das System nutzbare Spannung umzuwandeln. Festgelegt wurde, dass aus 230~V_{eff} 30=V mit einem Strom von 3A erzeugt werden soll.
Für die Realisierung standen folgende Schaltungen zur Auswahl:

1. Ein Trafonetzteil mit einem Spannungsregler zur Stabilisierung der Spannung.
2. Ein Trafonetzteil mit einer PW-Modulierten Spannungsregelung.
3. Ein reines Schaltnetzteil vom Typ Forward-Converter.

Die Wahl fiel auf Nr. 3, da Schaltnetzteile sowohl einen deutlich höheren Wirkungsgrad haben, als auch wesentlich kompakter sind als Trafonetzteile. Da es allerdings ein sehr hoher Aufwand ist, ein Schaltnetzteil zu konzipieren und zu bauen, wurde als Alternative, für den Fall, dass das Schaltnetzteil nicht fertig wird, Nr.1 gewählt.
Aufgrund der begrentzten Zeit wurden viele Bauteile der Schaltungen nicht von uns zu Fuß errechnet, sondern mit Quellen aus dem Internet. Einige Bauteile wurden sogar ohne jegliche Berechnungen aus Beispielschaltungen übernommen. Als Grundgerüst unserer verwendeten Schaltung dient Abb. 1.
Diese und Andere haben ihre Quelle im Buch: „Schaltnetzteile in der Praxis“ vom Vogel Verlag. Ein Verweis im Text auf „das Buch“ meint dieses. Als weitere Informationsquelle diente folgende Website: „http://schmidt-walter.fbe.fh-darmstadt.de/smps_e/spw_smps_e.html“

Theoretischer Teil

Primärseite

Festlegung der Eingangskapazität Ci

Als Erstes erfolgt die Festlegung der Eingangskapazität C_i (Abb.1). Um die Spannungen im betrieb möglichst konstant zu halten ist bei der Wahl der Kapazität darauf zu achten, dass die Brummspannung möglichst gering ist. Dies erreicht man durch eine große Kapazität, allerdings spielt auch die Größe der Bauteile selbst eine Rolle. Das Buch empfiehlt 1-2µF/W. Da wir wie anfangs erwähnt etwa eine Leistung von 100W umsetzen, hieße das für die Kapazität einen Wert von 100-200µF. Die Betriebsspannung der Kondensatoren muss mindestens $2 \cdot U_{eff} \cdot$wurzel(2)$\cdot 1{,}1$ sein (wird später erklärt), also 716V. Verwendet wurden 2 Kapazitäten a 330µF mit einer Betriebsspannung von 385V in Reihe geschaltet.

Bestimmung des Gleichrichters

Zu erfüllende Eigenschaften:
min. 800V Durchbruchspannung,
min. 0.6A Konstantstrom.
Die Wahl fiel auf B500C1500, da er den Ansprüchen mehr als genügte, und im Labor verfügbar war. Nähere Daten können dem Datenblatt entnommen werden.

Bestimmung des Schalt-MOSFETs

BUZ-80 wurde vom Buch empfohlen und z. T. verwendet. Alternativ wurde auch mit dem IRF-BE30 gearbeitet. Letzterer besitzt nach dem Datenblatt zu urteilen dieselben Eigenschaften wie der BUZ-80.

Berechnung des Haupttransformators

Tastverhältnis des MOSFETs

Die Natur der Schaltung gibt vor, dass der Transistor eine mindest Aus-Zeit von 50% haben muss, damit sich der Transformator nach der Ein-Zeit vollständig entmagnetisieren kann. Um das sicherzustellen, wird ein Tastverhältnis von 45% ein, und 55% aus gewählt.

Der Transformator

Der Transformator wurde wie bereits erwähnt mithilfe folgender Website ermittelt:
http://schmidt-walter.fbe.fh-darmstadt.de/smps_e/edw_smps_e.html
die Eingabedaten waren: V_{imin}/V_{imax} : 276/357V, V_{out} : 30V, I_{out} 3A, f : 40kHz

Als ideal schlägt das Programm die Kerne ETD29/E32 vor, die allerding aufgrund schlechter Verfügbarkeit nicht zu beschaffen waren… Verwendet wurde anstelle derer der Kern ETD34 mit 164 Primär- und 38 Sekundärwicklungen.

Alle anderen Bauteile, die nicht der Regel- und Überwachungsschaltung angehören wurden nach Vorhandensein im Labor hin ausgewählt.

Das Schalt-IC

Das im Buch vorgeschlagene IC von Siemens (TDA4718) wurde aufgrund seines Alters, seines Preises und der schlechten Verfügbarkeit nicht benutzt. Als Alternative bot sich TL494 an. Die Beispielschaltung für seine Peripherie entnahmen wir seinem Application Report (S.27). Für unsere Anwendung wurden folgende Dinge verändert:

- die Ansteuerund des Transistor-schaltenden Teils der Schaltung musste für den Gebrauch eines MOSFETs umkonzipiert werden.
- Soft-Start wurde deaktiviert.
- Strombegrenzung wurde für den Testaufbau vernachlässigt, da es sich um eine kontrollierte Umgebung handelt.

Das Feedback der Sekundärseite sollte wie in Abb.1 gezeigt über einen Optokoppler realisiert werden. Diese Schaltung hat folgendes Problem: sie kennt nur zwei Zustände: 1. Spannung zu niedrig, 2. Spannung in Ordnung/zu hoch. Das führt dazu, dass das IC den Transistor entweder im geschlossenen Zustand behält, oder ihn voll aufdreht. Da fehlte uns etwas an der PWM… Mit folgender Alternative konnten wir diese Teilschaltung durch einen simplen Spannungsteiler ersetzten: Mit den Betreuern kamen wir zu dem Schluss, dass die Primär- und Sekundärseiten nicht entkoppelt sein mussten, also verbunden wir die Massen miteinander.
Im Folgenden sei die gesamte Peripherie des ICs wie wir ihn nutzten wollten aufgezählt. Die Formeln für deren Berechnung finden sich in Application Report ab S.27. Die Benennung der Bauteile erfolgt wie in der Beispielschaltung auf selbiger Seite:

R_1: n.v.
R_2: n.v.
R_3: 4.7 k
R_4: 4.7 k
R_5: 0.47 k
R_6: 2.7 k
R_7: 7.3 k
R_8: 61.2 k
R_9: 5.1 k
R_{11}: n.v.
R_F: 47 k
R_T: 100 k
C_T: 250 pF

Die Versorgungsspannung für IC und Treiber wurden durch einen kleinen Zusatztrafo mit Spannungsregler gewährleistet. Resultierend daraus, dass die beiden Ausgangstransistoren für einen Strom von 200mA zugelassen sind und mit einer Frequenz von maximal 300kHz schalten, wurde auf eine komplexe Treiberschaltung verzichtet. Verwendet wurden folgende beiden Typen: Pull-Up und Pull-Down.

Testergebnisse der Testaufbauten

Testergebnis für den Schalt-IC

Der Testaufbau des ICs und seiner Peripherie funktioniert einwandfrei.

Testergebnis für den Gesamtaufbau

Beim ersten Einschaltvorgang floss ein so hoher Strom, dass die Sicherung des Trenntrafos durchbrannte. Die Vermutung, dass es an der Eingangskapazität liegt, lag nahe. Der Widerstand nach dem Gleichrichter wurde daraufhin erhöht. Beim zweiten Einschaltvorgang brannte erneut die Sicherung durch. Der Grund war der MOSFET BUZ-80 der aus unbekannten Gründen einen Kurzschluss verursachte und dadurch vernichtet wurde. Ersetzt haben wir ihn durch den IRFBE-30 der nach seinem Datenblatt mit dem BUZ-80 nahezu identisch ist. Alle weiteren Versuche wurden daraufhin mit dem IRFBE-30 durchgeführt. Es wurden im weiteren Verlauf mehrere Kombinationen aus den beiden Steuertransistoren des ICs und den Treiberschaltungen getestet, allerdings immer mit demselben Ergebnis, der Vernichtung des MOSFETs.

Aufgrund von Zeitmangel wurde auf eine zeitraubende Fehleranalyse verzichtet und die Alternativschaltung aufgebaut.

Prinzipieller Aufbau

Die Netzspannung wird von einem Trafo auf $30V_{eff}$ heruntertransformiert. Der Spannungsregler regelt auf die gewünschte Spannung mit einem Strom von 3A. Die großen Elkos garantieren im falle eines Stromausfalls 50ms lang für einen Reststrom von bis zu 3A am Ausgang.

Früherer Ansatz

Da die bisher entwickelten Schaltnetzteile nicht den gewünschten Anforderungen entsprachen, sind wir auf ein weitaus simpleres NT umgestiegen. Da ein einfacher Trafo NT nicht genügt und wir auch ein maximalen Strom von 3A bereitstellen müssen, entdeckten wir den Spannungsregler LM350T.

Dazu wurde von Svetomir Najdanovic folgende Schaltung konstruiert:

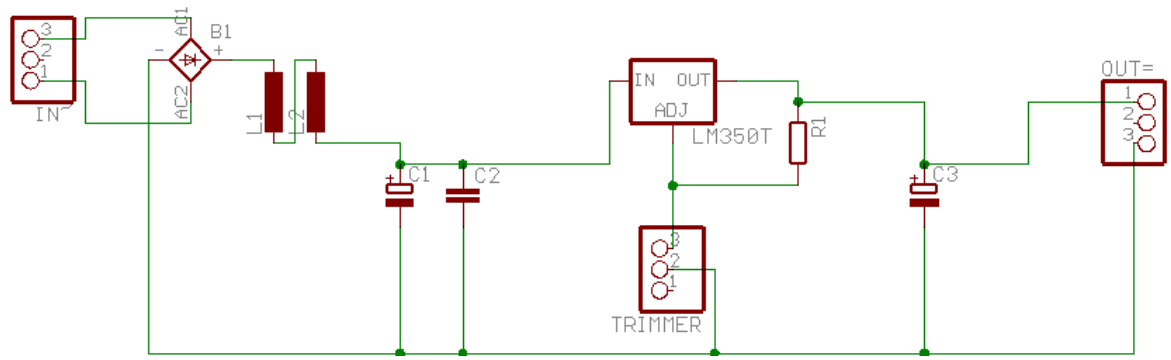


Bild 1: NT mit Spannungsregler LM350T entwickelt von Svetomir Najdanovic

Zwischen Netzspannung (230V/50Hz) und Gleichrichter kommt ein Trenntrafo und ein Trafo, welcher die Netzspannung zu $36V_{eff}$ herunter transformiert. Die Induktivität kontrolliert die Einschaltspannung, womit der Elko geladen wird. Nach der Glättung der Kapazitäten, regelt der Spannungsregler die Spannung auf die gewünschten 30V. Der Spannungsteiler mit dem Potentiometer sorgt für die richtige Dimensionierung, in unserem Fall 30V bei 3A.

Probleme bei der Entwicklung

Der Spannungsregler stieß mit der geforderten Leistung fast an seine Grenzen.
Die Idee, zwei Spannung parallel zu schalten wurde wieder verworfen, da diese Konstruktion nirgends in der Application-Note erwähnt wurde und daher nicht die Leistungen des Bauelements nicht gewährleisten könnte (obwohl eine derartige Schaltung im Testaufbau stabil funktionierte).
Laut Hersteller sollte man für die von uns geforderte Spannung zwar auch zwei Spannungswandler nehmen, allerdings gesteuert von einem OPV, wir fanden keinen OPV mit den vom Hersteller geforderten Eigenschaften, welcher mit einer $36V_{eff}$ umgehen kann.
Als wir den Trafo gegen einen kleineren Trafo mit $30V_{eff}$ austauschten, entstanden Eingangsströme von bis zu 27A, was dazu führt, dass die Spule schon bei 2A so viel

Energie in Wärme umsetzen musste, das ihre Kunststofffassung schmolz. Das lag vor allem an dem dünnen Draht, mit dem diese Spule gewickelt war.

Neues Konzept

Nach mehreren Testaufbauten haben wir uns kurzerhand für den Lowdrop-Spannungsregler LT1084 entschieden. Mit der kleineren Spule ist der LT1084 auch in der Lage Ströme von bis zu 6A am Ausgang zu kompensieren.

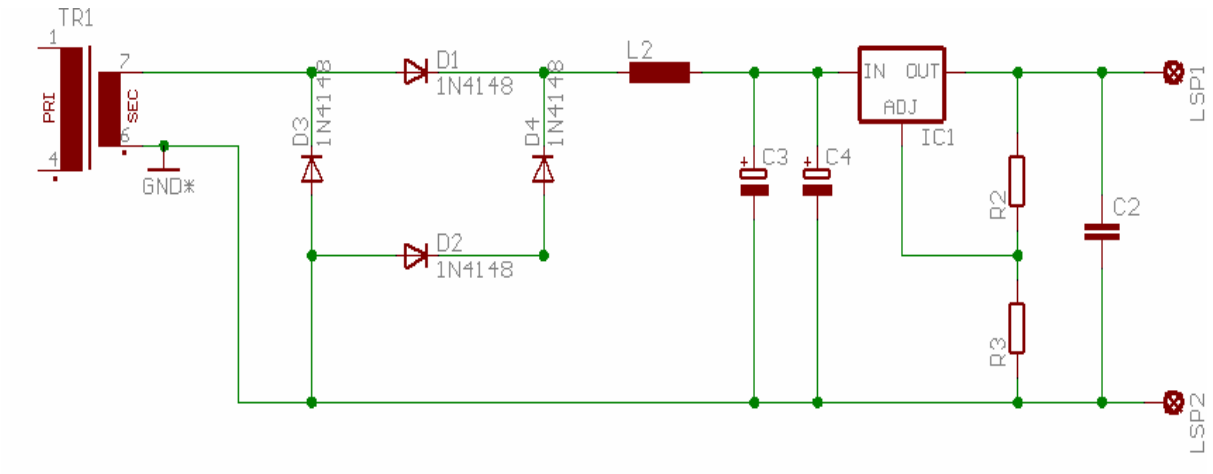


Bild 2: NT mit Lowdrop-Spannungsregler LT1084

Die oben aufgeführte Schaltung wurde von PSpice simuliert, um mehr Information über die entstehenden Spannungen und Ströme an den wichtigsten Stellen in Erfahrung zu bringen. Dazu setzten wir Messspitzen an dem Gleichrichter (Spannung), der Spule (Strom), dem Elko (Spannung), und als Last benutzen wir einen Widerstand, durch dessen Veränderungen wir den Strom steuerten zusätzlich setzten wir eine Messspitze für die Ausgangsspannung.

Daraus folgt folgende Tabelle:

R_{Last} in Ω	$U_{Gleichrichter}$ in V	I_{out} in A	ΔU in V	$P=\Delta U*I_{out}$ in V	$\hat{I}_L$ in A	$\hat{I}_{Leff}$ in A
90	45,27	0,0	15,27	0,0	26,97	1,35
60	44,60	0,5	14,6	7,3	26,98	1,82
30	42,90	1,0	12,9	12,9	27,03	2,16
25	42,30	1,2	12,3	14,76	27,04	2,40
20	41,50	1,5	11,5	17,25	27,07	2,80
15	40,00	2,0	10,0	20,0	27,10	3,50
10	37,85	3,0	7,85	23,55	27,20	4,90

Tabelle 1 zur PSpice-Simulation

Erläuterungen zum Simulationsergebnis

$I_{out} = \frac{U_{out}}{R_{last}}$

Strom am Ausgang

$\Delta U = U_{Gleichrichter} - U_{out}$

ΔU ist die Differenzspannung, die von dem Spannungsregler heruntergeregelt werden muss und P ist die Leistung, die der Regler effektiv umsetzen muss. Laut Hersteller kann der LT1084 bei Differenzspannungen (V_{in}-V_{out}) von 5V-25V Ströme zwischen 6A–0.6A zulassen, deshalb ist eine genaue Bestimmung der Differenzspannung des Reglers notwendig.

Die Leistungsumsetzung P ist wiederum für die Berechnung des Kühlkörpers wichtig, da der Spannungsregler während der Leistungsumsetzung einen beträchtlichen Teil in Wärme umsetzt.
Die Berechnung des Kühlkörpers folgt.

$\hat{I}_L$ ist der Spitzenwert des Einschaltstroms an der Spule (meist nur der erste Peak)

$\hat{I}_{Leff}$ ist der effektive Strom, der an der Spule anliegt, bei unterschiedlicher Last.

$$\hat{I}_{Leff} = \sqrt{\frac{1}{T} * \int_0^T i^2 * (t) dt}$$

Dimensionierung

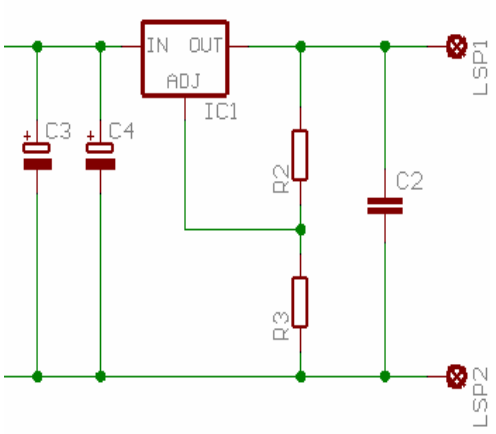


Bild 3: TL1084 mit dem Spannungsteiler R_2 und R_3

Spannungsteiler

Zur Dimensionierung der Widerstände R_2 und R_3 wird vom Hersteller in der Application-Note folgende Formel angegeben:

$$V_{out} = V_{ref} (1+ \frac{R_3}{R_2}) + I_{ADJ} R_2$$

$I_{ADJ} = 50\mu A$
$V_{ref} = 1{,}25V$
} Angaben vom Hersteller aus dem Datenblatt

Somit haben wir für R_2 =125Ω und R_3 = 2860Ω bestimmt, wobei das Verhältnis sehr wichtig ist. Zu beachten ist: es muss immer ein minimaler Laststrom von 10mA vorhanden sein.

Kühlkörper

Wie schon erwähnt braucht der LT1084 bei dieser Leistungsumsetzung einen Kühlkörper, den wir mit folgender Formel berechnen:

$$R_{thk} = \frac{\Delta T}{P} - (R_{thj\text{-}c} + R_{thm})$$

Die Werte sind in der Tabelle 2 aufgelistet. $R_{thj\text{-}c}$ und R_{thm} sind Angaben des Herstellers und ΔT wird gebildet aus der Differenz von T_j (siehe Datenblatt) und T_{amb} (unsere gewählte Umgebungstemperatur bzw. Gehäusetemperatur) und mit 0,8 (20% Toleranz) multipliziert.

max. Junction Temperatur (T_j)	150°C (Datasheet)
Umgebungstemperatur (T_{amb})	ca.40°C
therm. Widerstand beim Übergang Gehäuse/Kühlkörper (R_{thm})	0,4 K/W (pauschal)
therm. Widerstand Sperrschicht/Gehäuse ($R_{thj\text{-}c}$)	2,7 K/W (Datasheet)
max. Leistungsumsetzung des LT1084 (P)	23,55W siehe Tabelle 1

Tabelle 2 Variablen zur Berechnung des Kühlkörpers

So ergibt sich in unserem Fall ein Wärmewiderstand von 0.6367 K/W. In unserem Fall kam dazu der Typ SK34 88x35x100 (BxHxT) in Frage, welchen wir in einem Elektronik-Katalog entdeckten.

Spule

Nun blieb noch nach wie vor das Problem mit der Spule. Wir brauchten keine große Induktivität – in unserem Fall würden 50µH völlig ausreichen. Das Problem sind die zu hohen Anfangsströme, woraus folgt, dass die Drahtdicke (Querschnitt) verändert bzw. vergrößert werden muss.
Da wir in Zeitdruck gerieten und wir einen neuen Kern benötigten, bedienten wir uns an den vorhandenen ETD34, der voll funktionsfähig war. Wir mussten allerdings – abgesehen vom Drahtquerschnitt – die entsprechende Windungszahl berechnen.
Zur Berechnung der Windungszahl *N* benutzten wir die Formel

$$N = \frac{L}{A} * \frac{\hat{i}}{B_{\max}}$$

Hierbei ist unser L = 50µH, A ist der Querschnitt des Kerns, bei uns ist A = 96,77 mm^2. Mit B_{max} ist die Flussdichte den Ferritkerns gemeint, in unserem Fall ist B_{max} des Kerns mit dem Material N27 = 380mT. Für î haben wir 12A gewählt, da das absolute Maximum von 27A nur bei der ersten Halbwelle erreicht wird. Diese Werte in die Formel eingesetzt ergeben 16,3 Windungen für unseren Kern.

Anschließend muss die Querschnittsfläche *A* des Kupferdrahtes berechnet werden. Die Formel dafür ist aus dem Internet:

$$A = \sqrt[0,68]{\frac{I[A]}{9,1 * \Delta T[K]^{0,43}}}$$

I[A] = 12A, für ΔT[K] haben wir 10 eingesetzt (gemeint ist die Temperatur, um die sich der Draht erhöhen kann – die kann man selbst festlegen). Somit ergibt sich ein Querschnittsfläche von 0,349mm^2. Den Durchmesser *d* des Drahtes erhalten wir mit

$$d = 2 * \sqrt{\frac{A}{\pi}} = 0,67\text{mm}$$

Wir haben dennoch ein Draht mit dem Querschnitt von 0,8mm gewählt – für alle Fälle.

Test

Nach einem erneuten Testaufbau, bewährte sich diese Schaltung im Testlauf und mit den Elkos (2*10mF), ist laut PSpice bei Netzausfall ein 50ms. andauernder Strom von 3A möglich (nötig bis die USV den Kreislauf übernehmen kann).

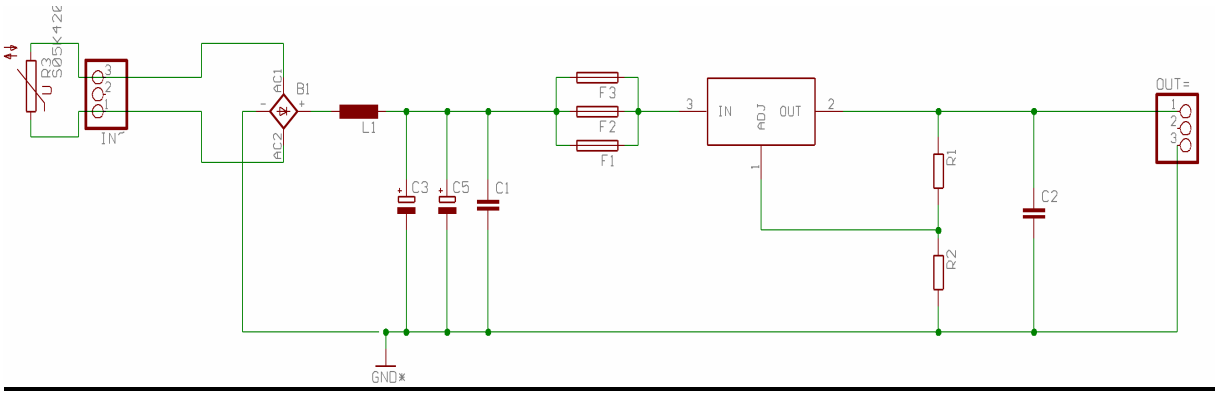
Beim „Stöpseltermin“ hat sich unser NT bewährt, jetzt fehlten noch wichtige Feinheiten, wie eine stabile Sicherung bei Überspannung.

Sicherung

Bei möglichen Überspannungen ist es wichtig dass kein einiges Bauteil schaden erleidet, aber das NT muss in der Lage sein, die Spannungsversorgung im Gang zu halten bzw. bei temporärer Abschaltung wieder selbstständig anzuspringen, wenn die Werte sich wieder normalisiert haben.

Wir entschieden uns daher vor der Spule einen Varistor (K420) und vor dem Spannungsregler drei in Reihe geschaltete Multifuse (R090) einzusetzen.

Endgültige Schaltung:



Platinenlayout 1:1:

2.3. Der Gleichrichter

Aufgabe:

Aufgabe unseres Gleichrichters sollte es sein ein Spannungssignal gleichzurichten. Da es sich bei diesem Signal um ein Messsignal handelt, war es wichtig darauf zu achten, dass bei der Gleichrichtung die Amplitude des Signals nicht verändert wird.

Schaltungen:

Es standen folgende Schaltunen zur Auswahl:

1.) Graetzschaltung/ Brückengleichrichter (Abbildung 2)

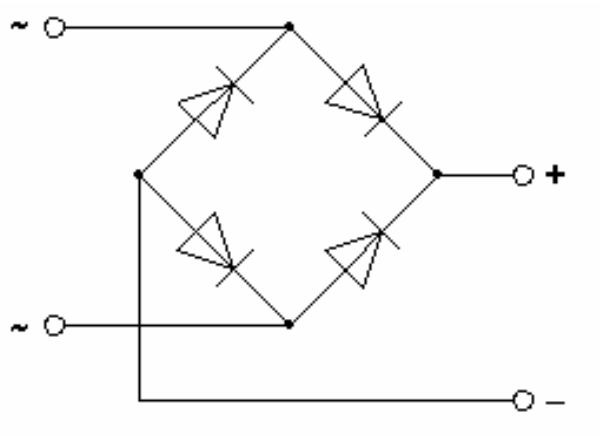


Abbildung 2 Brückengleichrichter

2.) Synchrongleichrichter (Abbildung 3)

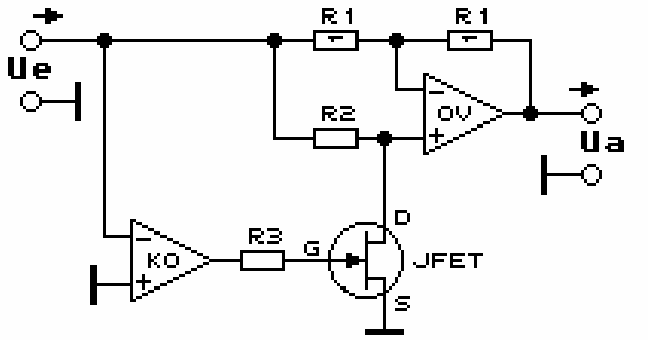


Abbildung 3 Synchrongleichrichter

3.) Vollweggleichrichter (Abbildung 4)

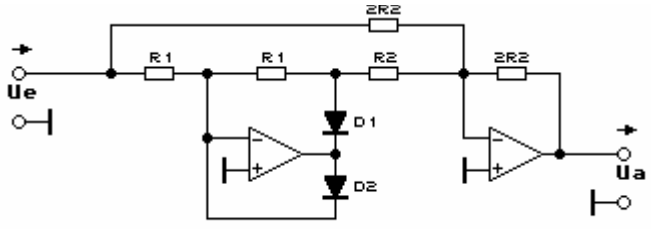


Abbildung 4 Vollweggleichrichter

Entscheidung

Da das Ausgangssignal die gleiche Amplitude wie das Eingangssignal haben sollte, schied der Brückengleichrichter von vorne herein aus. Da das Signal bei dieser Schaltung über 2 Dioden läuft, „verliert“ man damit etwa 1,4 V.
Beim Simulieren der Schaltungen in Pspice ist uns beim Synchrongleichrichter dann aufgefallen, dass das Ausgangsignal während der negativen Halbwelle des Eingangssignals nicht ganz die gewünschte Amplitude erreicht. In Abb.4 ist dies in den Kreisen zu erkennen.

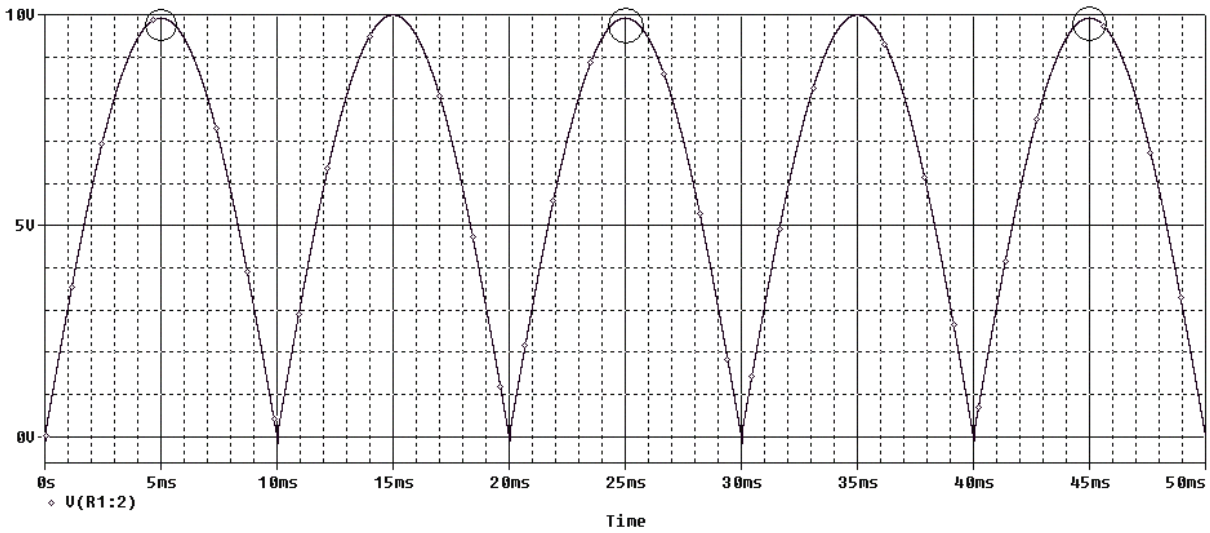


Abb. 4: Pspice-Simulation des Synchrongleichrichters

Beim Vollweggleichrichter war dies nicht der Fall (s. Abb. 5).

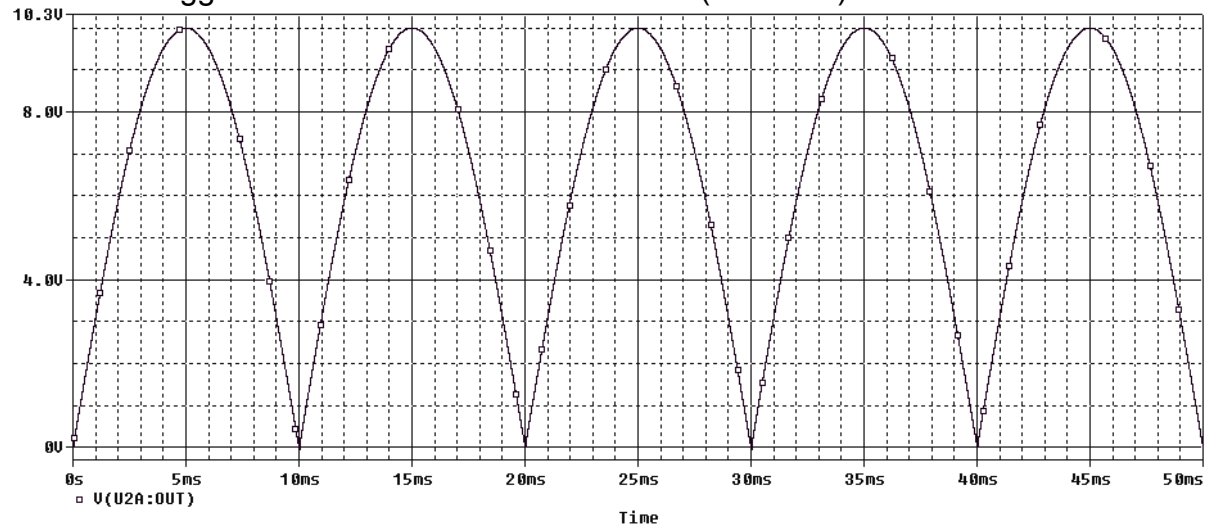


Abb. 5: Pspice-Simulations des Vollweggleichrichters

Daher kam nur der Vollweggleichrichter in Frage.

Anmerkung zu den Simulationen: Es wurde hier mit einer Amplitude von 10V simuliert. Im eigentlichen Spannungswächter ist die Amplitude so gewählt, dass der Effektivwert des Messsignals 9V beträgt. Dies macht im Ergebnis (abgesehen von der geänderten Amplitude) aber keinen Unterschied.

Funktionsweise

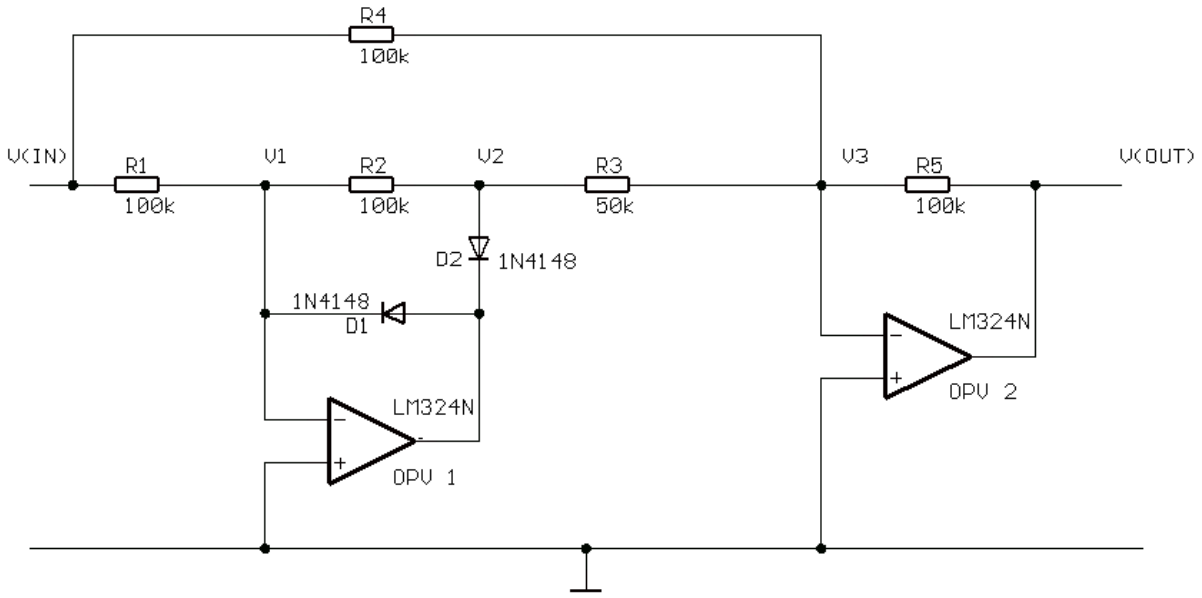


Abbildung 5 Vollweggleichrichter

Um diesen Gleichrichter (Abb. 6) zu verstehen, ist es sinnvoll sich die beiden darin enthaltenen Operationsverstärkerschaltungen getrennt voneinander anzusehen und jeweils zwischen einem positiven und einem negativen Eingangssignal [bei V(IN)] zu unterscheiden.

OPV 1

Liegt am Eingang [V(IN)] ein negatives Signal, so liegt auch am invertierenden Eingang (-) des OPV 1 ein negatives Signal. Der Operationsverstärker verstärkt invertierend und am Ausgang entsteht so ein positives Signal. Dieses Signal bewirkt, dass die Diode D1 leitet und die Diode D2 sperrt. Der Operationsverstärker wird also über D1 am invertierenden Eingang rückgekoppelt, wodurch das Potential am Punkt V1 auf Masse gezogen wird. Da durch den Widerstand R2 in diesem Fall kein Strom fließt, liegt auch der Punkt V2 auf Massepotential.
Liegt am Eingang [V(IN)] ein positives Signal, so liegt auch am invertierenden Eingang (-) des OPV 1 ein positives Signal. Der Operationsverstärker verstärkt invertierend und am Ausgang entsteht so ein negatives Signal. Dieses Signal bewirkt, dass die Diode D1 sperrt und die Diode D2 leitet. Der Operationsverstärker wird also über D2 und den Widerstand R2 am invertierenden Eingang rückgekoppelt. OPV 1 arbeitet in diesem Fall wie ein invertierender Verstärker und da R1 und R2 gleich groß sind, entsteht am Punkt V2 das selbe Potential wie bei V(IN) bloß mit umgekehrtem Vorzeichen. An V2 liegt also für den Fall eines positiven Eingangssignals ein negatives Signal

$$\text{Es gilt also: } V2 = \begin{cases} 0, \text{ für } V(IN) < 0 \\ -V(IN), \text{ für } V(IN) > 0 \end{cases}$$

OPV 2

OPV 2 ist über R5 am invertierenden Eingang rückgekoppelt. Das bedeutet, er arbeitet als Addierer. Addiert werden hier die Eingangssignale von V(IN) und V2, wobei zu beachten ist, dass der Widerstand, über den das Signal von V2 eingekoppelt wird, nur halb so groß ist wie der Widerstand, über den das Signal von V(IN) eingekoppelt wird, und R5. Das heißt, dass das Signal von V2 gegenüber dem Signal von V(IN) doppelt gewichtet wird.
Für V(OUT) gilt also: V(OUT) = - [2 x V2 + V(IN)]
Somit ist für den Fall eines...

a) negativen Eingangssignals: V(OUT) = - [2 x 0 + V(IN)] = - V(IN)
b) positiven. Eingangssignals: V(OUT) = - [2 x V(IN) + {- V(IN)}] = V(IN)

Es gilt also:

$$V(OUT) = \begin{cases} -V(IN), \text{ für } V(IN) < 0 \\ V(IN), \text{ für } V(IN) > 0 \end{cases}$$

Bauteilliste

- Zwei OPVs (LM324N), in einem IC
- Vier 100 kOhm Widerstände (höchstens 1% Toleranz)
- Ein 50 kOhm Widerstand (höchstens 1% Toleranz)
- Zwei Dioden (1N4148)

2.4. Der DC/DC-Wandler

Aufgabe

Uns wurde der Auftrag gegeben Schaltungen zu entwerfen, die die Spannungen umwandelten und diese konstant an die weiteren Teilen der großen Schaltung liefern bzw der DC/DC Wandler. Diese sollten aus der +24V vom Netzteil +15V, -15V und +5V machen.

Teilschaltungen

Zur Realisierung haben wir uns für die folgende Grundschaltungen entschlossen:

Der Abwärts-Wandler

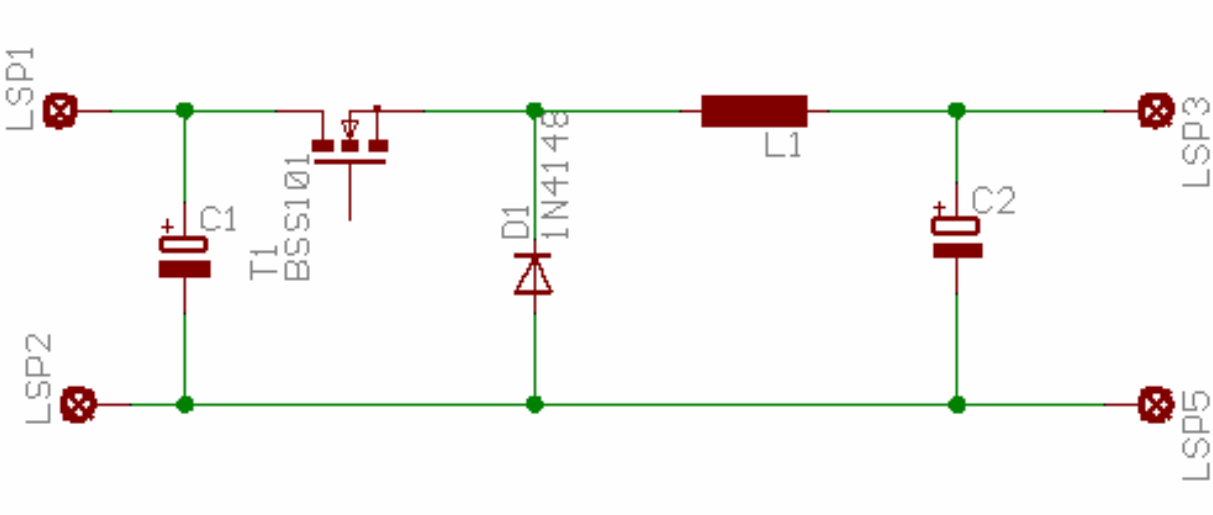
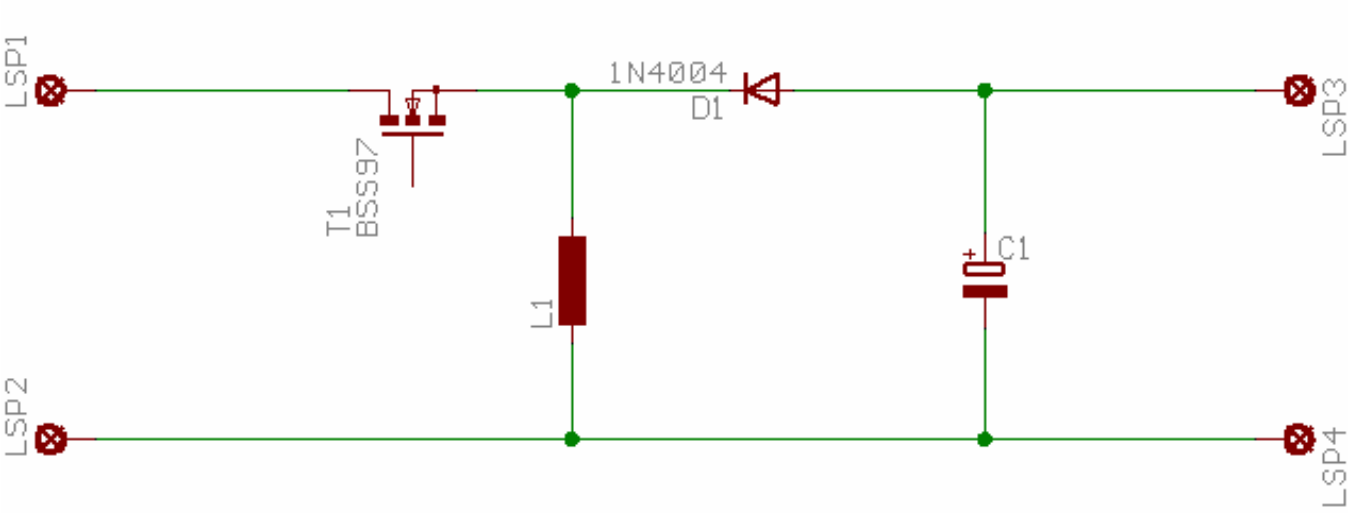


Abbildung 6

Diese Schaltung wird auch Step-down Converter bzw auch Tiefsetzeller genannt. Sie transformiert die 24V vom Netzteil auf +15V bzw +15V auf +5V.
Der MOSFET ist hier als Schalter betrachtet. Wenn also der Schalter geschlossen wird, dann lädt sich die Spule auf UL = Ue - Ua. Hier ist zu achten, dass der Schalter nur für eine gewisse Zeit zu bleibt, sonst würde man die Ausgangsspannungen (+15V bzw +5V), die wir haben wollen überschreiten. Dann muss der Schalter wieder geöffnet werden, dadurch sollte sich die Spule jetzt über den Kondensator und die Diode entladen.

Der Invertierende Wandler



Der invertierende -Wandler macht aus einer positiven Spannung (+15V) eine negative Spannung (-15V). Der Schalter spielt wieder hier die Rolle von Schalter.
Hier ist das Funktionsweise fast die Gleiche wie beim Aufwärts-Wandler, nur dass die Spannung an der Spule beim Zuschließen Ul =Ue beträgt; und beim geöffneten Schalter entlädt sich die Spule über den Kondensator Ul = -Ua.

Unsere wichtigste Aufgabe bestand darin den MOSFET anzusteuern. Wir kamen auf die Idee, unseren Schalter einfach mit Schmidttrigger zu steuern, so leicht war das eben aber nicht, weil wir am Ausgang keine Gleichspannung hatten, was nicht dem entsprach, was wir haben wollten.

Mit Hilfe unser Betreuer und durch Recherchieren wurde uns klar, dass wir PWM (Pulsweitermodulation) anwenden mussten, um dieses Problem zu lösen.

Zur PWM

Mit einem Schmidttrigger sollten wir einen Rechteckgenerator erzeugen, und das Signal sollte als Eingangspannung eines Integrators weiter gegeben werden. Dieser Integrator erzeugt ein Dreiecksignal. Das Signal wird weiter an einen Komparator gegeben und dort mit einer Referenzspannung verglichen.

Wir hatten das Problem, dass wir ohne negative Spannung unsere OPs nicht so leicht betreiben könnten, was uns noch mehr Arbeit brachte. Da die Aufgabe zu komplex wurde und wir nicht mehr im Zeitplan waren, haben uns entschlossen, LTSpice (SwitcherCAD) von „Linear Technologies“ zu benutzen. Diese liefert die folgenden Schaltungen und Simulationen:

Der endgültige Abwärts-Wandler

- Von +24V auf +15V

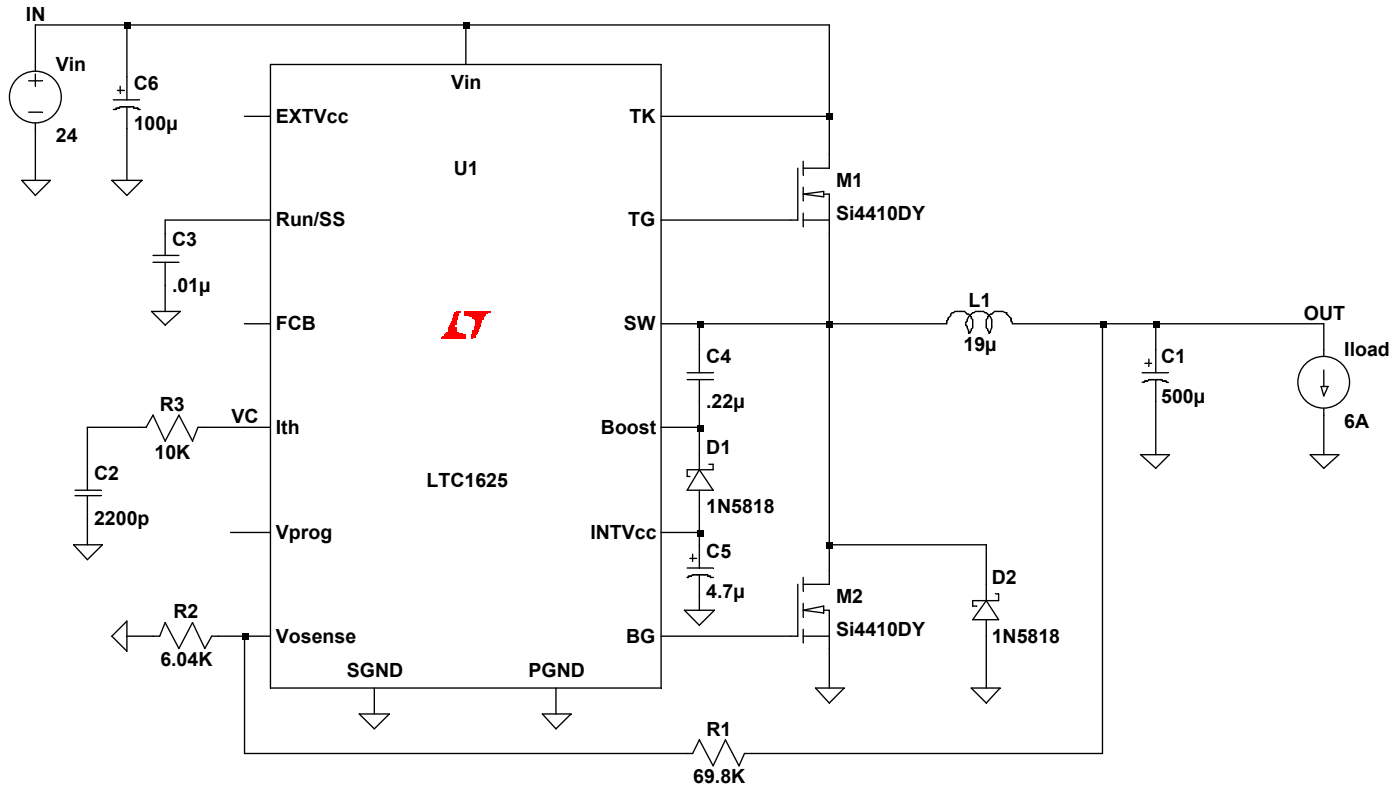


Abbildung 7

- Die dazugehörige Simulation
-

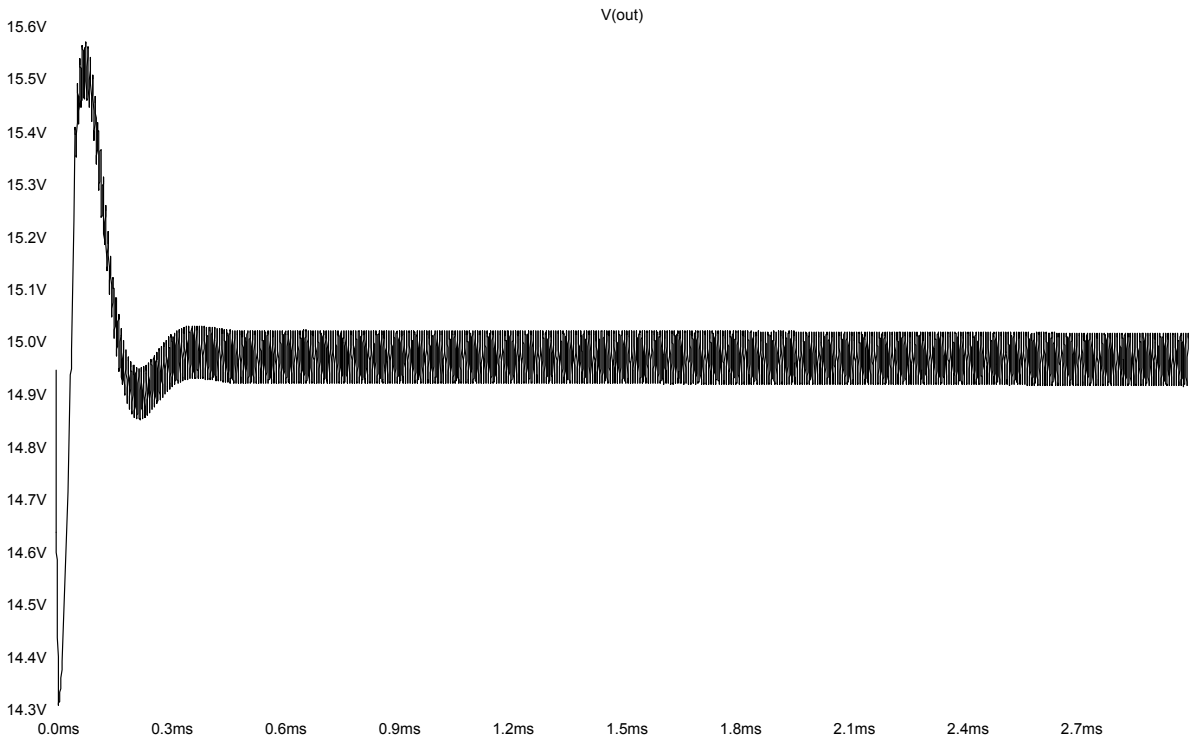


Abbildung 8

- Von +15V auf +5V

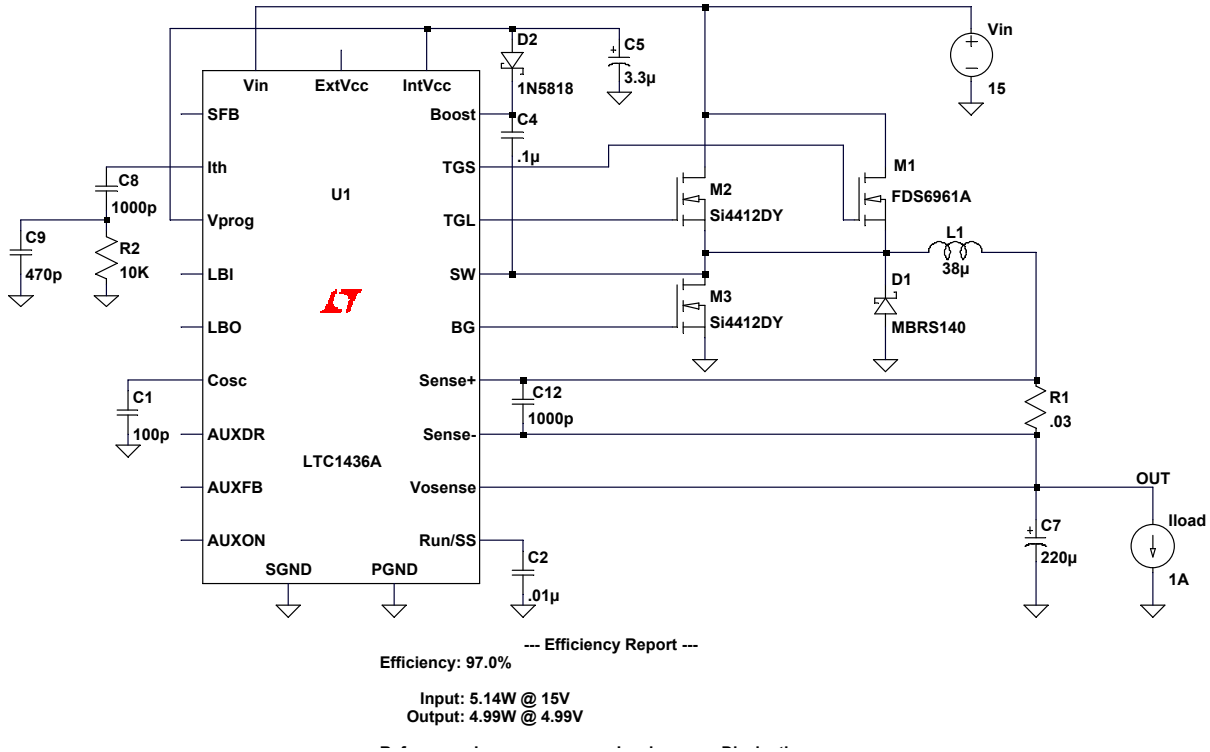


Abbildung 9

- Die dazugehörige Simulation

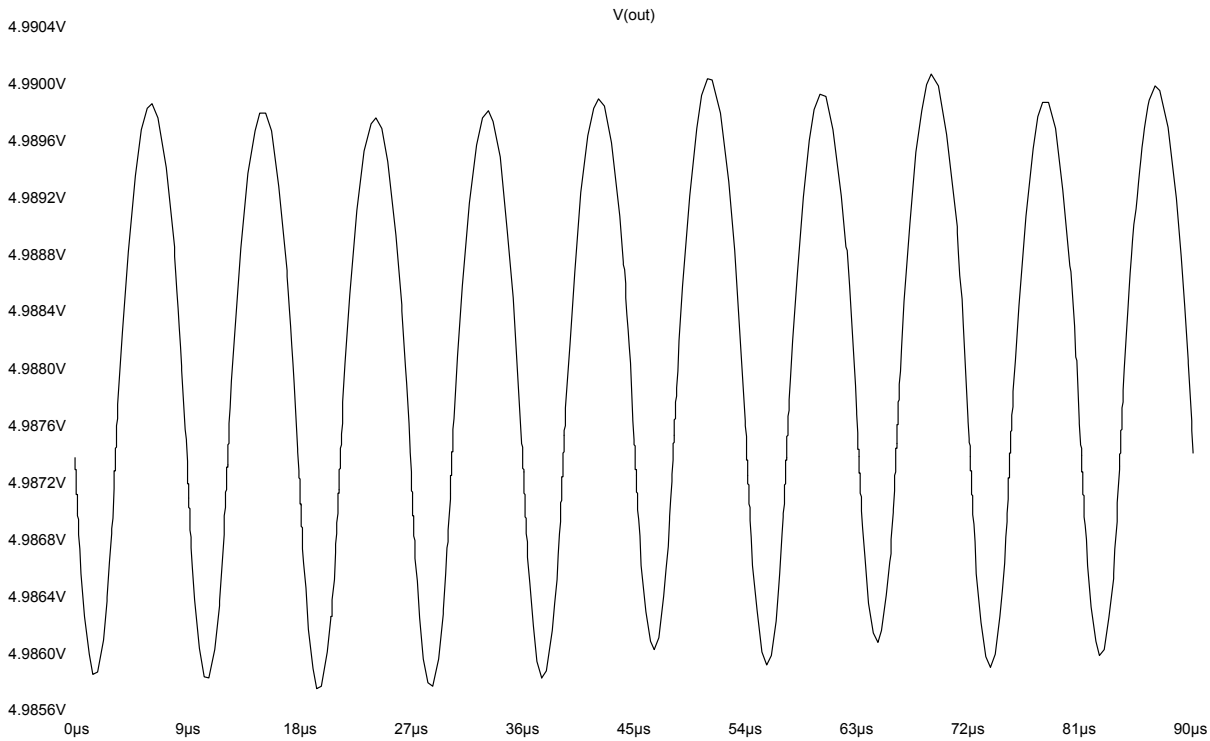


Abbildung 10

- Von +15V auf -15V

Der endgültige Invertierende –Wandler

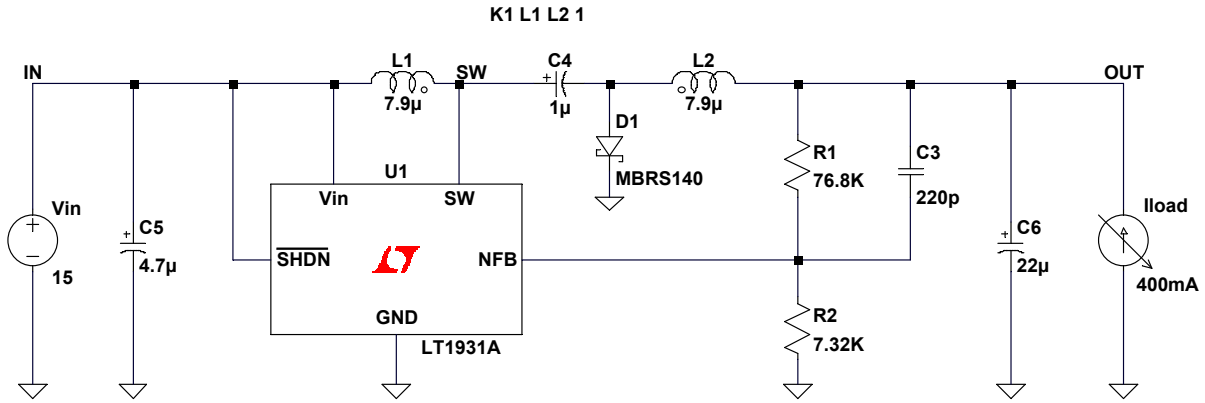


Abbildung 11

Die dazugehörige Simulation

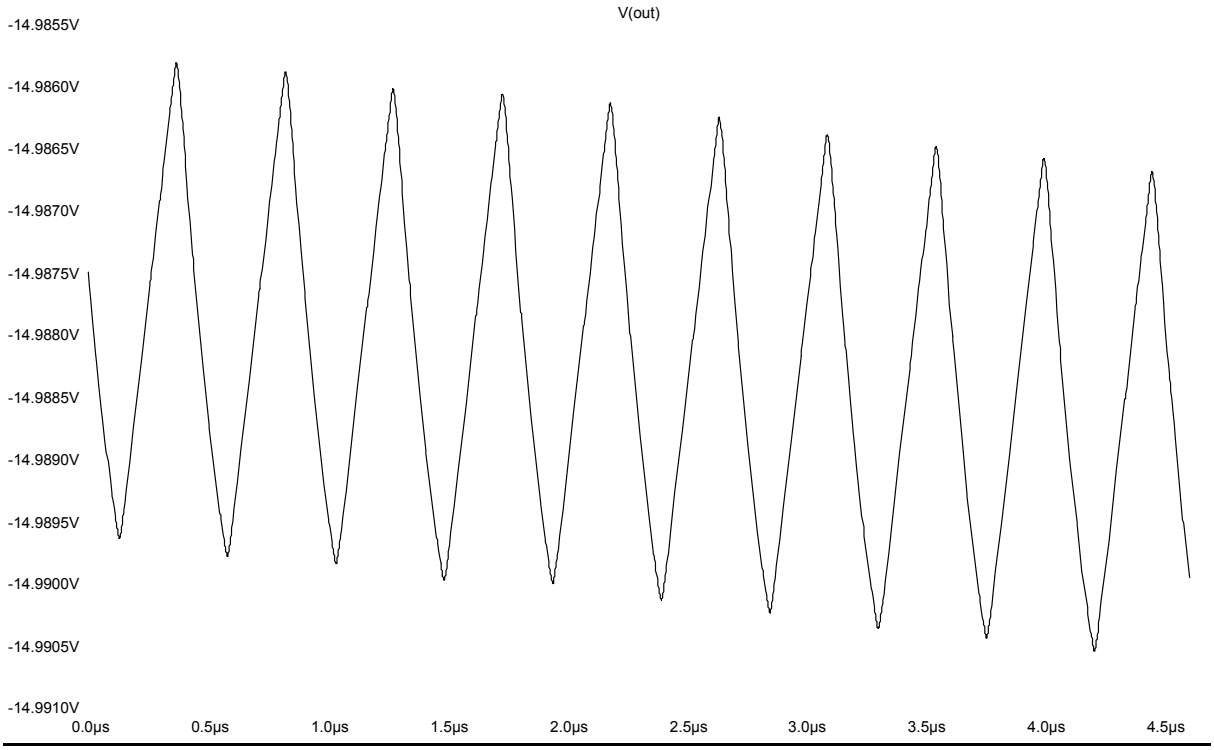


Abbildung 12

Das Board:

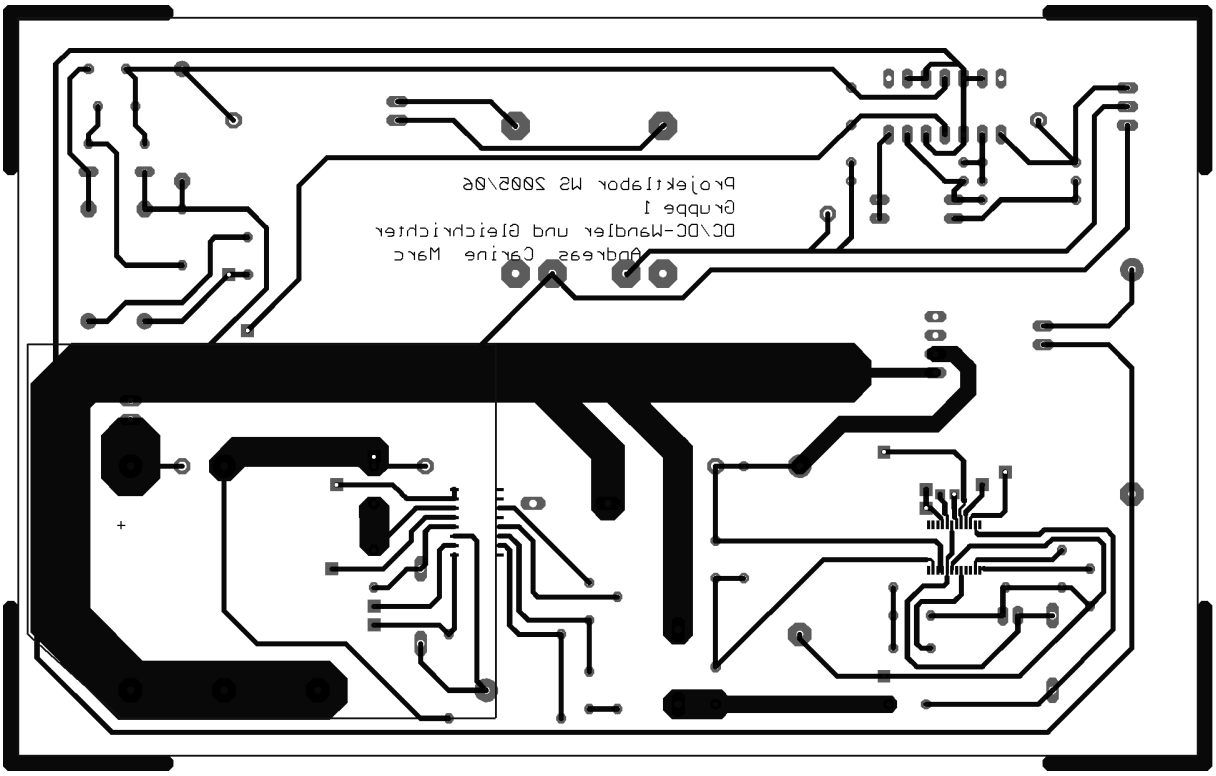


Abbildung 13

Dimensionierung der Bauteile für

- Die +15V

Widerstände	6.04kΩ, 69.8k, 10kΩ
Elektrolyte Kondensatoren	100µF, 500µF, 4.7µF
Kondensatoren	0.01µF, 2200pF, 0.22µF,
MOSFET	Si 4410DY, Si4410DY
Dioden	1N5818
Spulen	19µH,
Ic´s	LTC1625

- Die +5V

Widerstände	10kΩ, 0.03Ω
Elektrolyte Kondensatoren	3.3µF, 220µF
Kondensatoren	470pF, 1000pF, 100pF, 0.1µF, 0.01µF
MOSFET	Si4412DY, Si4436A
Dioden	MBRS140
Spulen	38µH
Ic´s	LTC1436A

- Die -15V

Widerstände	76.8kΩ, 7.32kΩ
Elektrolyte Kondensatoren	4.7µF, 1µF, 22µF
Kondensatoren	220pF
Spulen	7.9µH, µµ
Dioden	MBRS140
Ic´s	LT1931A

Die Spulen wurden von uns gewickelt.

3. Gruppe 2

3.1. Aufgabe der Gruppe

Die Aufgabe der Gruppe 2 bestand darin den maximalen und den minimalen Spitzenwert der Netzspannung zu messen, diesen über die gewünschte Zeitspanne (24 Stunden) zu speichern und außerdem jeweils ein Signal auszugeben, wenn ein Stromausfall oder eine Überspannung vorliegt.
Der minimale und maximale Spitzenwert und die Signale Stromausfall und Überspannung sollen an die Gruppe der Anzeige übermittelt werden; das Signal Stromausfall zeigt der Gruppe: „Unterbrechungsfreie Stromversorgung“ an, dass die Netzspannung nicht mehr vorhanden ist und der Spannungswächter seine Stromversorgung aus einer anderen Quelle beziehen muss.

Zur Bewältigung dieser Aufgaben hat unsere Gruppe drei Platinen entwickelt:
Analogplatine
Digitalplatine
Umschaltplatine.

3.2. Über- / Unterspannungsdetektion

Die Über- / Unterspannungsdetektion befindet sich auf der Analogplatine. Die Stückliste und die Ätzlayout befinden sich unter 3.4, während das Schematic im Anhang unter 10.2 zu finden ist.

Prinzipieller Aufbau der Schaltung

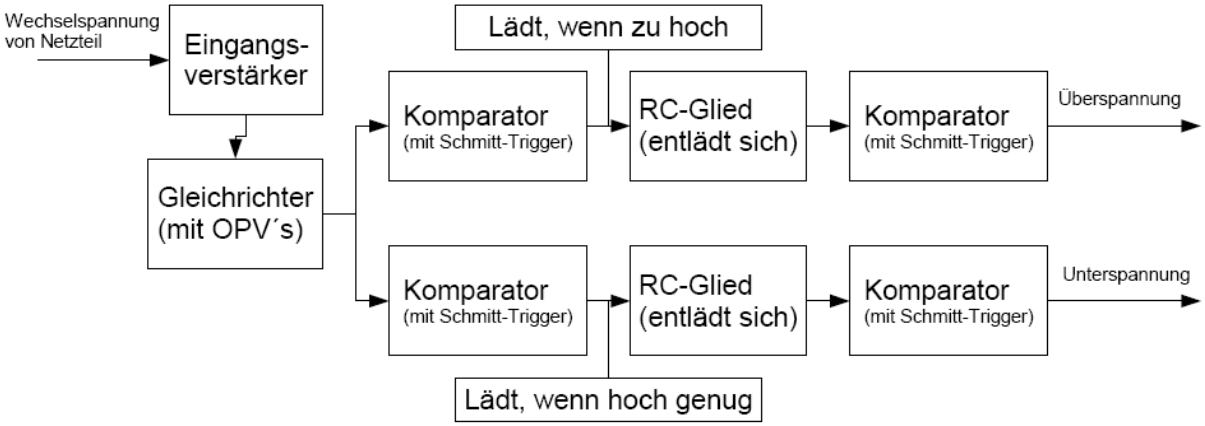


Abbildung 14: BSB Über-/Unterspannungsdetektion

Die eingehende Netzspannung wird mit Hilfe eines Eingängsverstärkers auf ein für unsere Schaltung verträgliches Niveau verringert. Danach wird das Signal ohne Spannungsverlust gleichgerichtet. Zwei Komparatoren mit Schmitt-Triggern vergleichen dann ob die Spannung zu hoch oder zu niedrig ist und laden dann RC-Glieder. Das RC-Glied für den Überspannungsfall lädt sich, wenn die Spannung zu

hoch ist und das RC-Glied im Unterspannungsfall lädt sich wenn die Spannung hoch genug ist. Somit ist das Ausgangssignal für den Überspannungsfall high-active und gibt ein Signal aus solange eine Überspannung vorliegt. Das Unterspannungssignal ist low-active, das heißt das Signal geht aus, wenn ein Stromausfall vorliegt.

Eingangsverstärker

Aufgabe

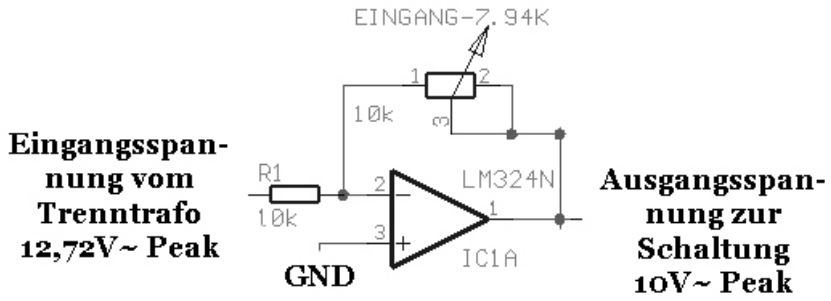


Abbildung 15: Eingangsverstärker

Der Eingangsverstärker soll die Eingangsspannung auf ein Niveau, das man mit OPVs nutzen kann, verringern. Wir haben uns hierbei für 10V~ Peak entschieden, da in diesem Bereich die OPV noch normal arbeiten und man die Vergleichspannungen nicht zu klein wählen muss.

Resultierende Schaltung

Als Lösung hierfür bot sich ein invertierender Verstärker an, da er einen Eingangspegel auf einen geringeren Ausgangspegel herunter setzen kann und dabei den Verlauf des Signals unverändert lässt, des weitern wird der OPV dabei nicht zu weit ausgesteuert. Die Invertierung spielt dabei keine Rolle, da wir ja mit symetrischer Wechselspannung arbeiten.

Der Spindeltrimmer oben an der Schaltung ermöglicht das variable Anpassen der Ausgangsspannung.

Funktionsprinzip

Der Spannungsteiler aus R1 und dem Spindeltrimmer führt dazu, dass die Spannung am Minus-Eingang des OPV's bei Aufsteuerung des Ausgangs, gegen 0V geht. Diese erreicht die Schaltung nun schon bei einer im Vergleich mit der Eingangsspannung geringere Ausgangsspannung und realisiert somit einen Verstärker mit einem geringeren Ausgangspegel, auf welchem durch den OPV aber viel mehr Leistung gezogen werden kann.

Simulation

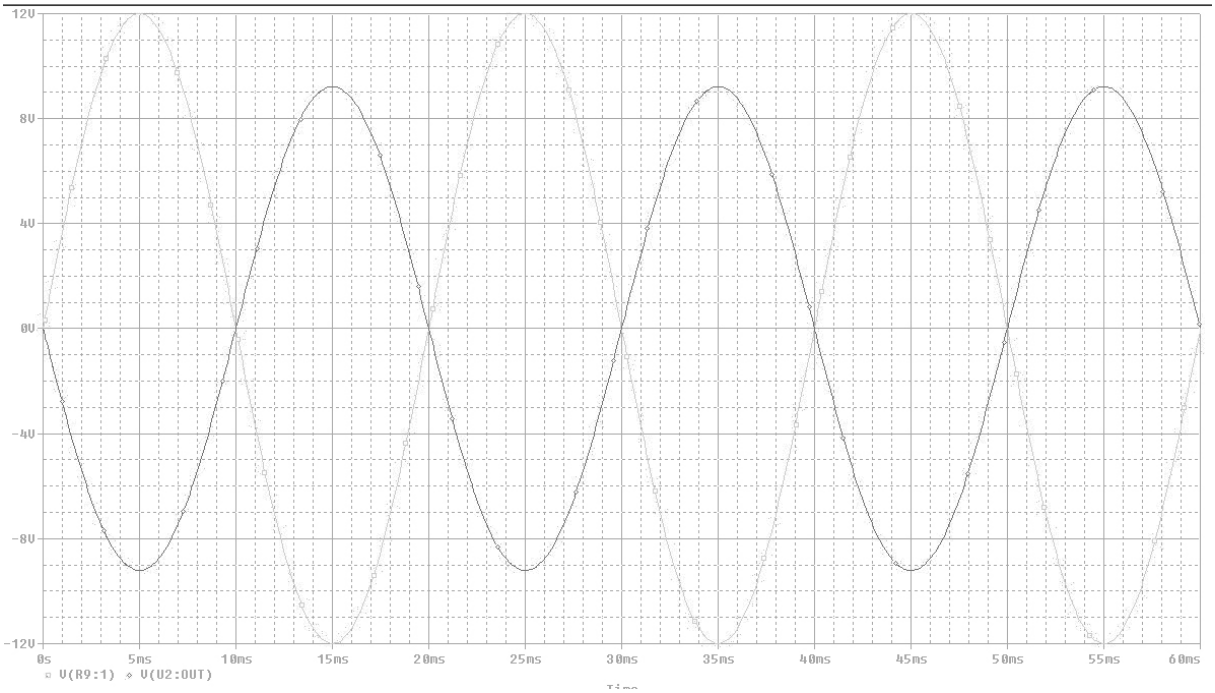


Abbildung 16: Simulation Invertierender Verstärker

Das graue ist das Eingangssignal und das schwarze das Ausgangssignal. Der geringere Pegel und die Invertierung sind sehr gut zu erkennen.

Der Präzisionsgleichrichter

Aufgabe

Eine Gleichrichtung des Signals ohne Spannungsverlust.

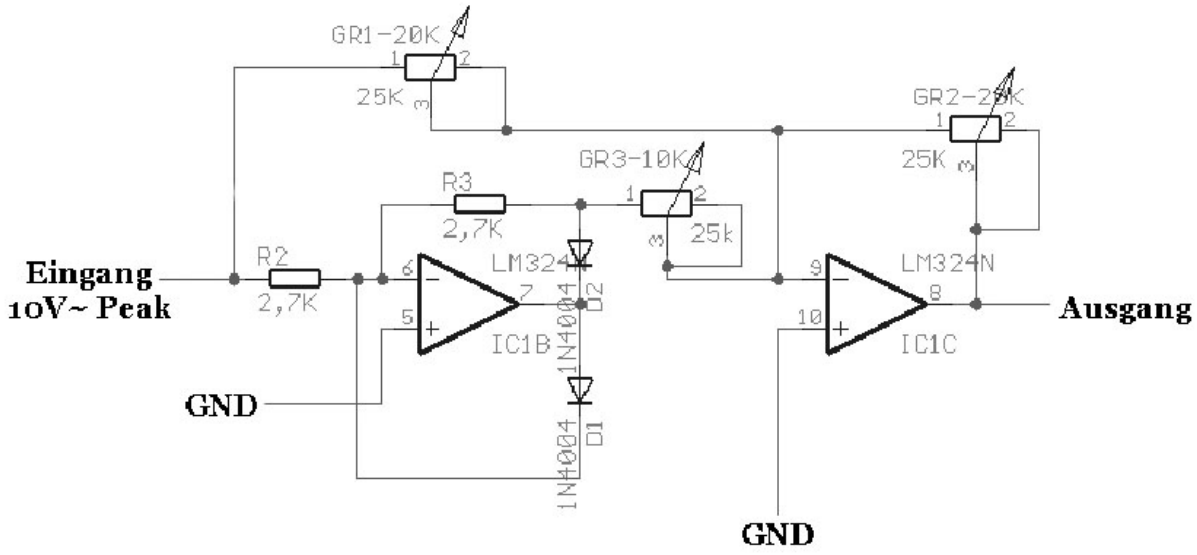


Abbildung 17: Präzisionsgleichrichter

Früherer Ansatz

Brückengleichrichter, mit dem dementsprechenden Spannungsverlust an den Dioden von 2x 0,7V und einer Kondensatorwirkung der Dioden, da der Ausgang des Gleichrichters ausschließlich auf Eingänge von OPVs geht und somit keine Wechselspannung mehr zur Verfügung steht.

Resultierende Schaltung

Als Lösung bot sich hierfür ein OPV Präzisionsgleichrichter an, da wir keine großen Leistungen benötigen und die Eingangsspannung die OPVs nicht in die Sättigung treibt, da diese schon auf eine Spannung gesenkt wurde, die die OPVs nicht zerstört.

Probleme

Es stellte sich sehr schnell heraus, das die verwendeten Widerstände sehr exakt sein müssen, da sonst eine Halbwelle größer ist, als die andere, und somit das Prinzip ad absurdum geführt wird.
Gelöst haben wir dies mit Spindeltrimmern, da wir diese beim Betrachten des Signals am Oszilloskop genau einstellen konnten.

Funktionsprinzip

Für eine Positive Halbwelle ergeben sich folgende Spannungen am Gleichrichter:

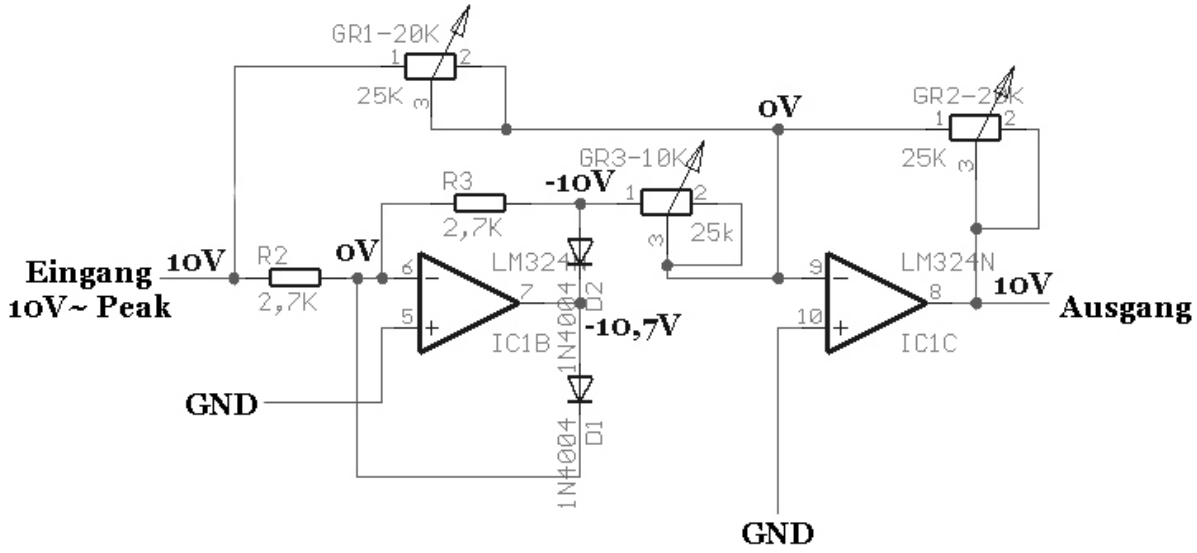


Abbildung 18: Positive Halbwelle am Präzisionsgleichrichter

Es stellt sich bei positiven Halbwellen ein Spannungsteiler zwischen GR1, GR2 und GR3 ein. Dabei kann man GR1 und GR2 als parallel geschaltet ansehen, da über sie, die gleiche Spannung abfällt, somit bildet sich mit GR3 eine Spannung von 0V am 2ten OPV aus und der Ausgang gibt die Spannung aus, welche auch am Eingang anliegt.
Dies wird ermöglicht, da die Diode D2 durchbricht und somit die negative Eingangsspannung, welche der erste OPV ausgibt (invertierender Verstärker, Verhältnis 1:1(+0,7V durch die Diode)), auf den zweiten OPV über GR2 ausgibt.

Für eine Negative Halbwelle:

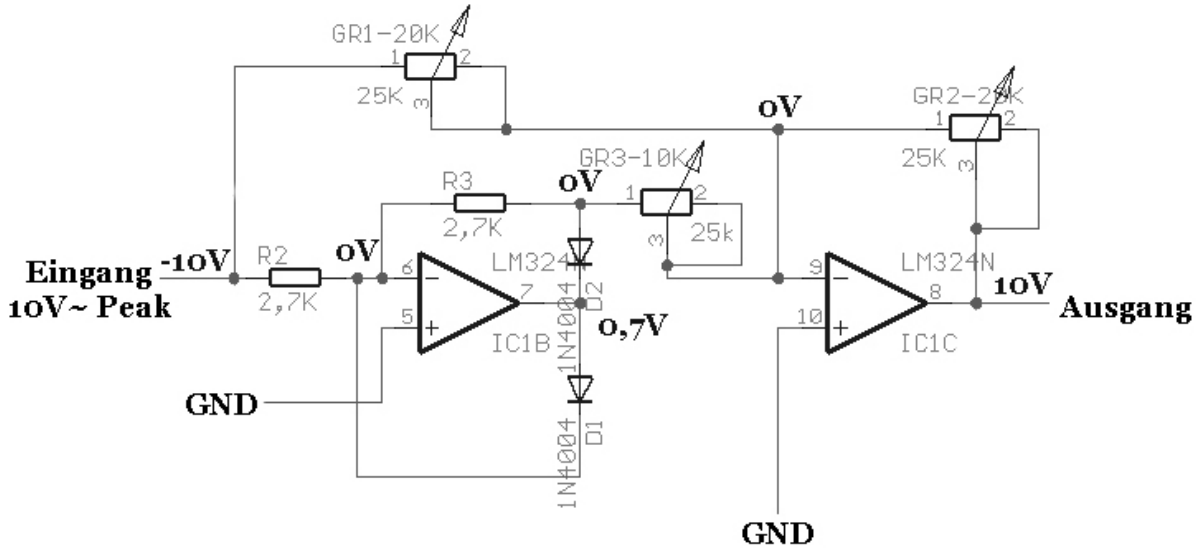


Abbildung 19: Negative Halbwelle am Präzisionsgleichrichter

Hierbei Bildet sich nur ein 1:1 Spannungsteiler zwischen GR1 und GR2 aus, welcher den Ausgang des zweiten OPV auf den Betrag der Eingangsspannung zwingt, damit dieser eine Spannungsdifferenz am Eingang von 0V hat. Dies wird ermöglicht durch die Diode D1 welche bei 0,7V am Ausgang des ersten OPV durchsteuert und somit dessen Eingang und den Knoten vor GR3 auf 0V zieht.

Simulation

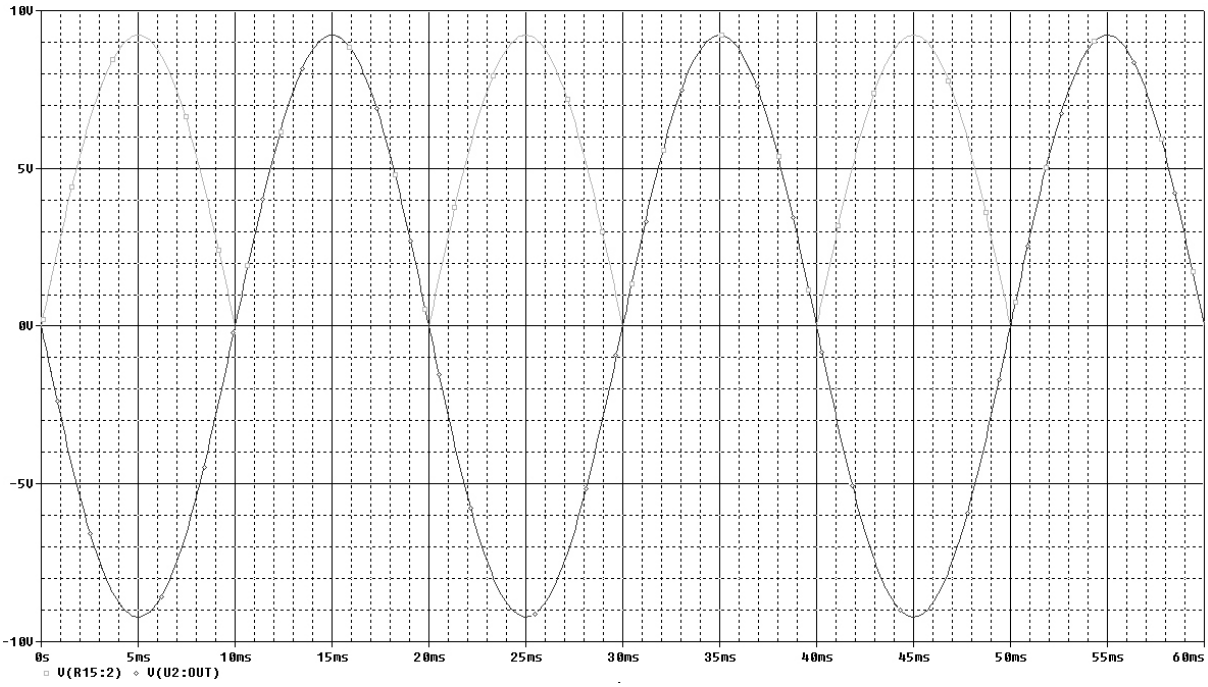


Abbildung 20: Gleichgerichtetes Signal

Man sieht nun, dass das Signal gleichgerichtet wurde.

Unterspannungsdetektion

Aufgabe

Ausgeben eines Signals bei Unterspannung.

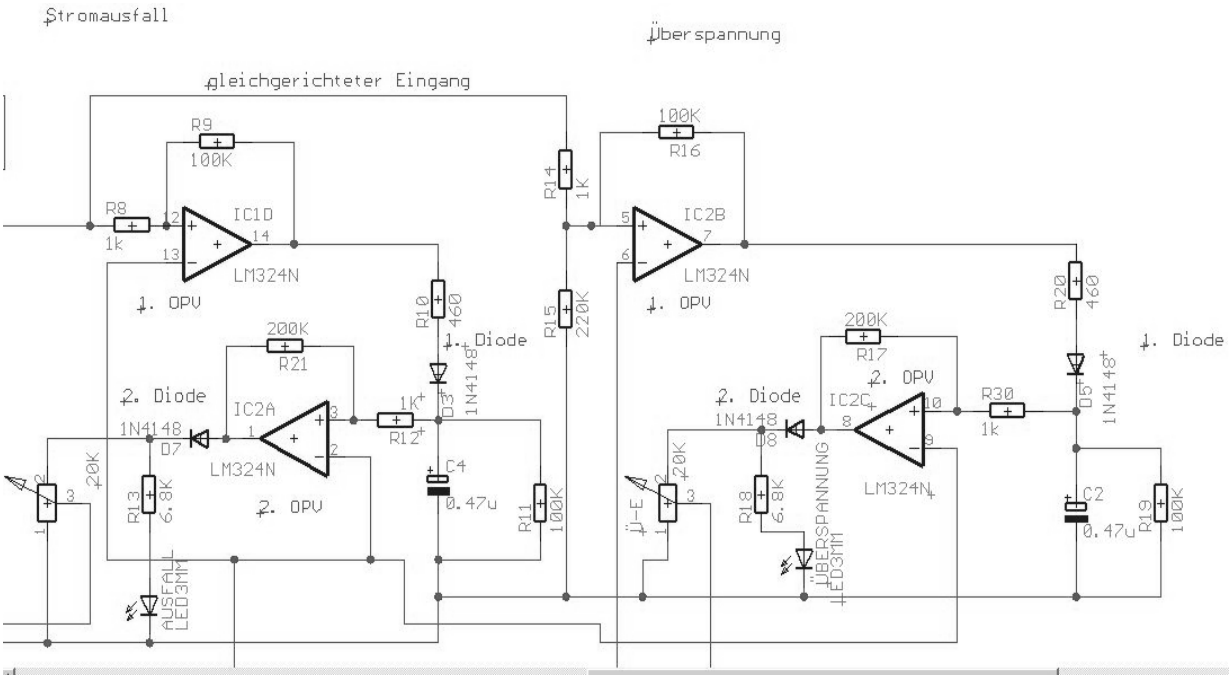


Abbildung 21: Schaltung Unterspannungsdetektion

Früherer Ansatz

Die erste Idee zur Realisierung der Überspannungs- / Stromausfall-Detektion basierte auf der Verwendung von Transistoren. Mit der gewonnen Erkenntnis, dass es Bauelemente gibt, welche genau das tun was wir realisieren wollten, verworfen wir die alte Idee und setzten gezielt Operationsverstärker (OPV) ein.

Funktionsprinzip

Das präzisionsgleichgerichtete Signal wird, mit Hilfe eines Komparators (OPV), mit einer Referenzspannung von 9V verglichen. Solange der Eingang größer als die Referenzspannung ist, geht der OPV in Sättigung. Das heißt es liegen +15V am Ausgang an. Hinter dem OPV befindet sich eine Diode, welche in diesem Fall in Durchlassrichtung betrieben wird. Ein Speicherkondensator wird geladen.
Fällt nun die Spannung unter die Referenzspannung von 9V, wird der OPV in negative Sättigung gehen (-15V). Nun wird die Diode aber sperren. Dies führt dazu, dass sich der Speicherkondensator über einen Widerstand entlädt. Dadurch liegt eine Spannung am positiven Eingang des zweiten OPV´s an.
Dies führt dazu dass der zweite OPV in positive Sättigung geht. Hinter dem befindet sich wiederum eine Diode welche nur sperrt wenn am Ausgang des zweiten OPV´s eine positive Spannung anliegt.
Der Ausgang zur Anzeigegruppe ist „Low-Aktiv“. Das heißt, dass im Unterspannungsfall ein Signal von 0V anliegt.

Die beiden OPV´s sind jeweils mit Schmitt-Triggern versehen, so dass immer eine Hysterese wirksam ist. So wird garantiert, dass der Kondensator immer genug zeit hat sich voll zu entladen.

Simulation

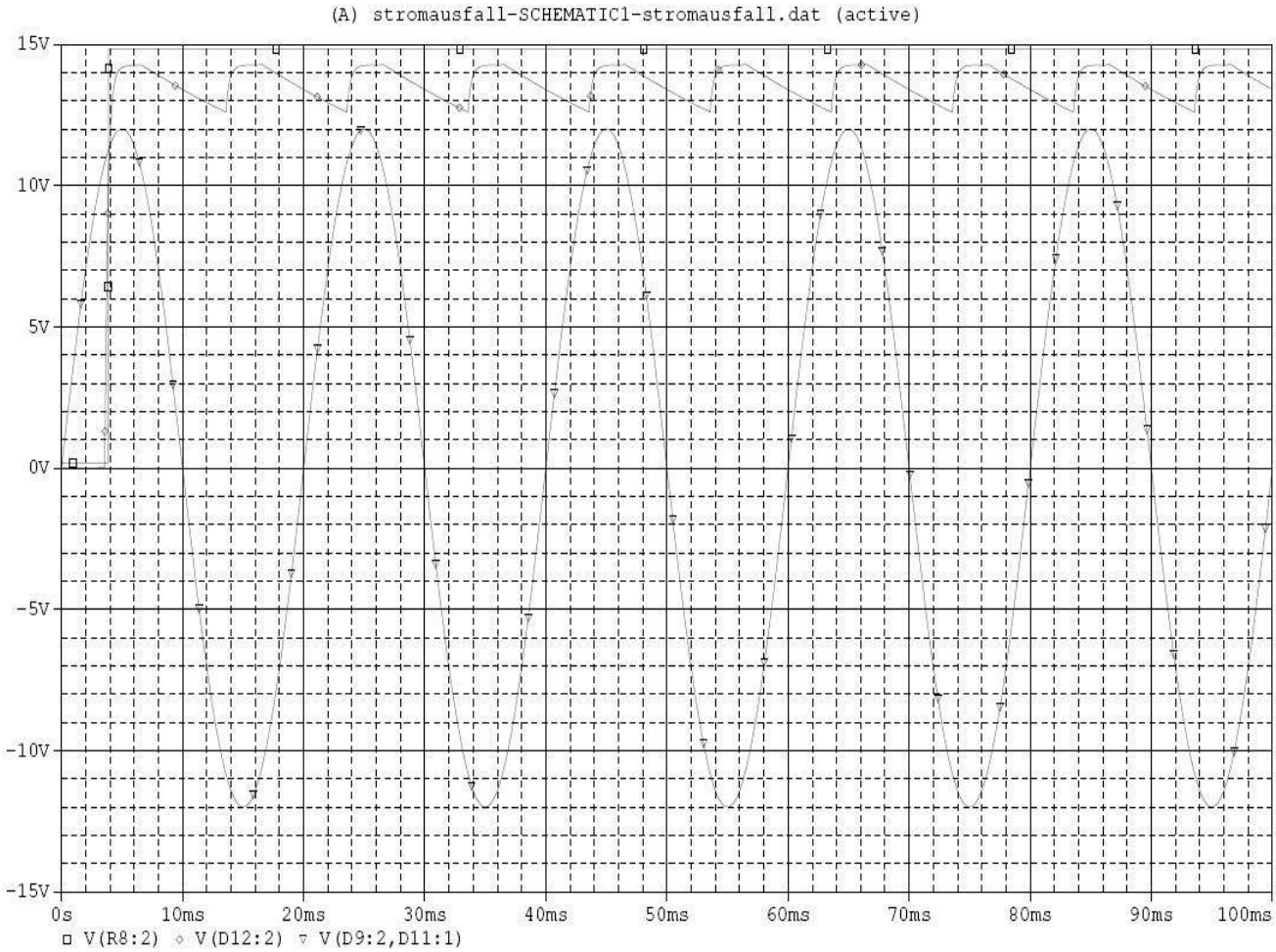


Abbildung 22: Normalfall (keine Unterspannung)

Man sieht hier das High-Signal bei 15V, d.h. keine Unterspannug.

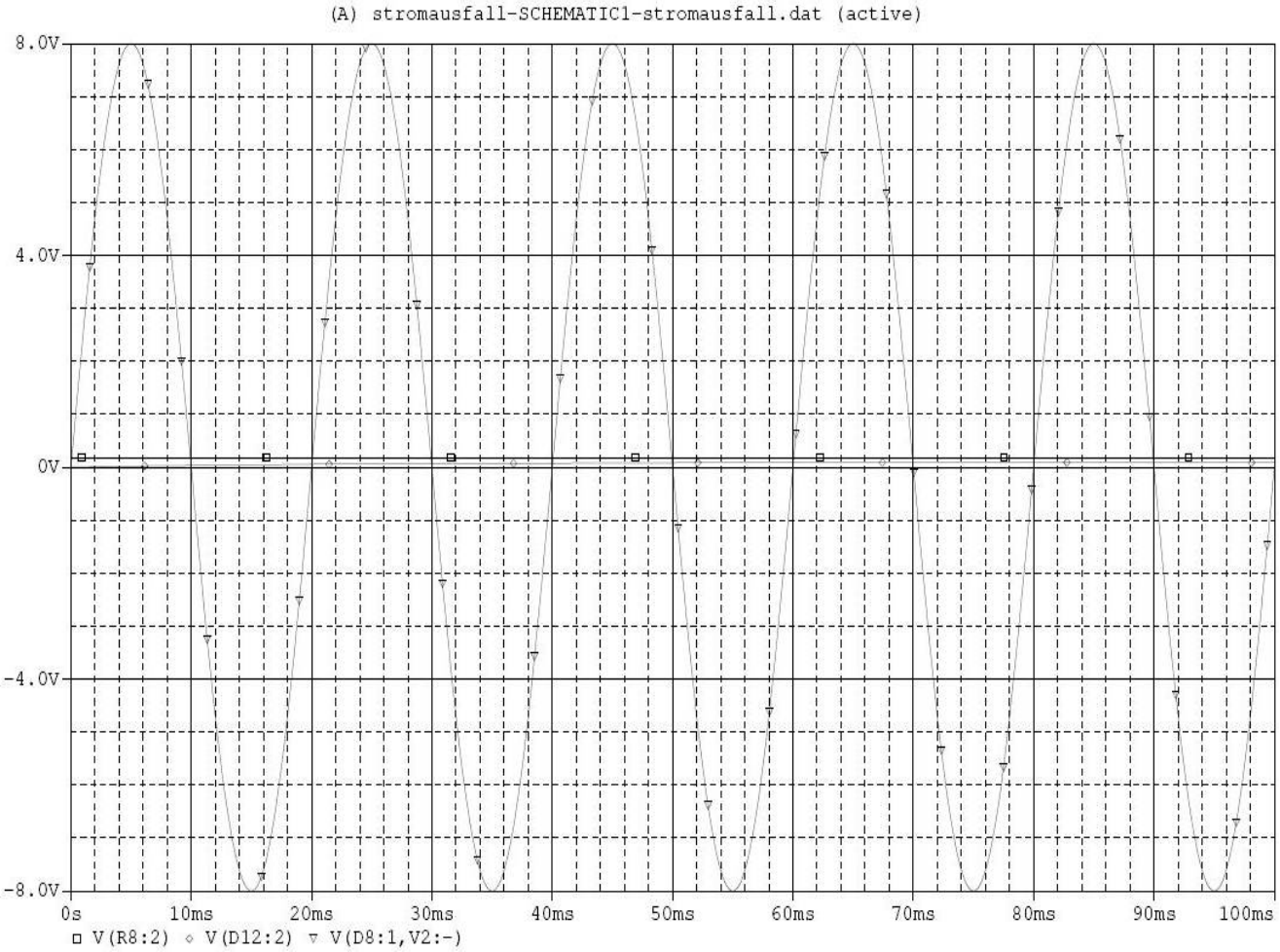


Abbildung 23: Stromausfall

Man sieht hier, dass das Signal aus Abbildung 22 nun ein Low-Signal ist.

Überspannungsdetektion

Aufgabe

Ausgabe eines Signals bei Überspannung.

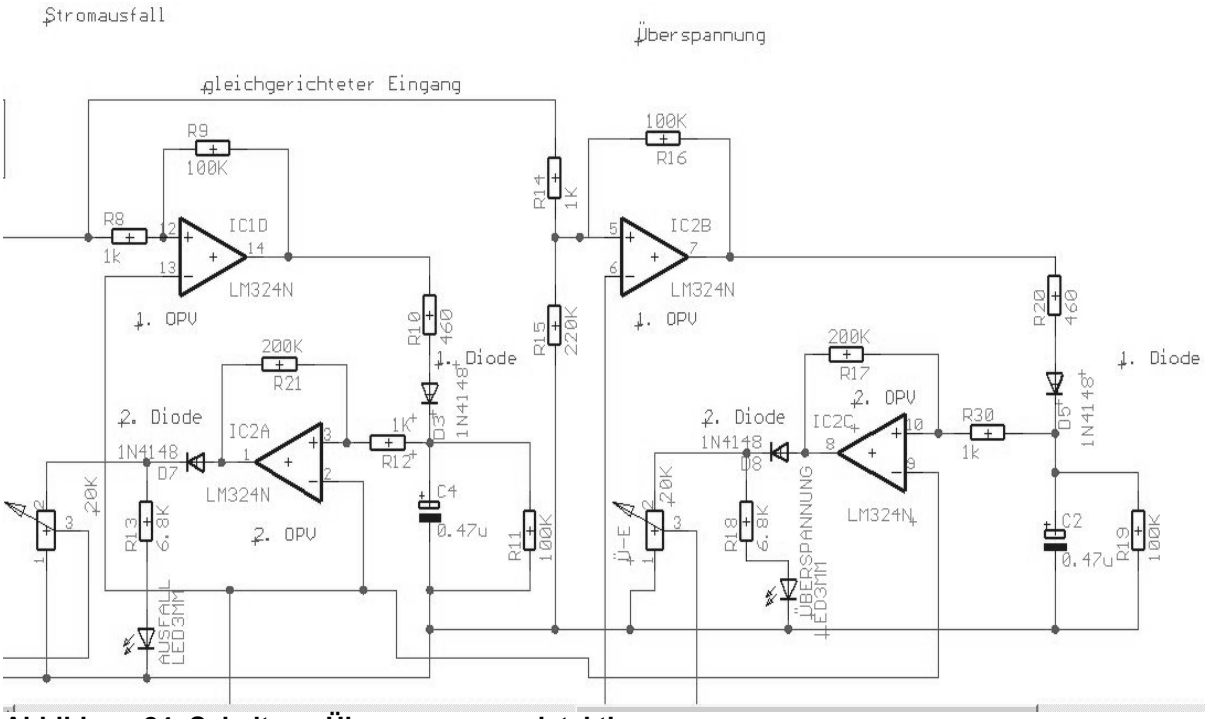


Abbildung 24: Schaltung Überspannungsdetektion

Funktionsprinzip

Die Überspannungsdetektion funktioniert bis auf ein paar Details analog zur Stromausfalldetektion:

Der erste OPV vergleicht das präzisionsgleichgerichtete Signal mit einer Referenzspannung von 10,6V (106% von 10V). Am Ausgang des OPV´s liegen also permanent -15V an, solange keine Überspannung am Eingang anliegt. Die erste Diode hinter dem OPV sperrt. Der Ladekondensator wird nicht geladen und der zweite OPV geht in negative Sättigung. Die zweite Diode sperrt ebenfalls.
Überschreitet die Eingangsspannung die Referenzspannung, so geht der OPV in positive Sättigung. Am Ausgang liegen somit +15V an. Die erste Diode wird in Durchlassrichtung betrieben und der Kondensator wird geladen, gleichzeitig schaltet der zweite OPV nun positiv durch, so daß ein Signal über den Bus zur Anzeige gesandt wird, da die zweite Diode nun auch in Durchlassrichtung betrieben wird.
Das Überspannungssignal ist somit „High-Aktiv“.

Auch hier wurden die OPV´s mit Schmitt-Triggern ausgestattet.

Simulation

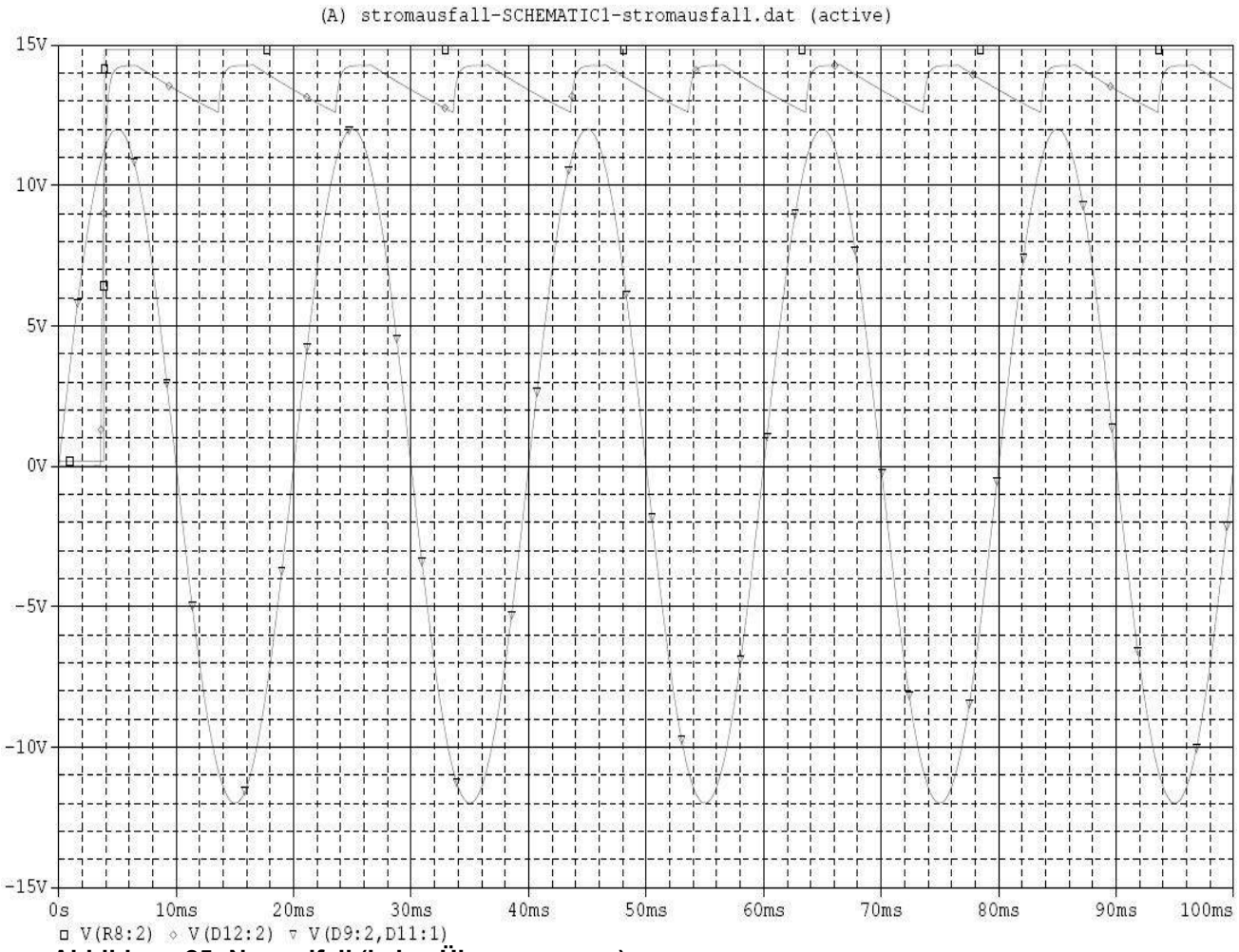


Abbildung 25: Normalfall (keine Überspannung)

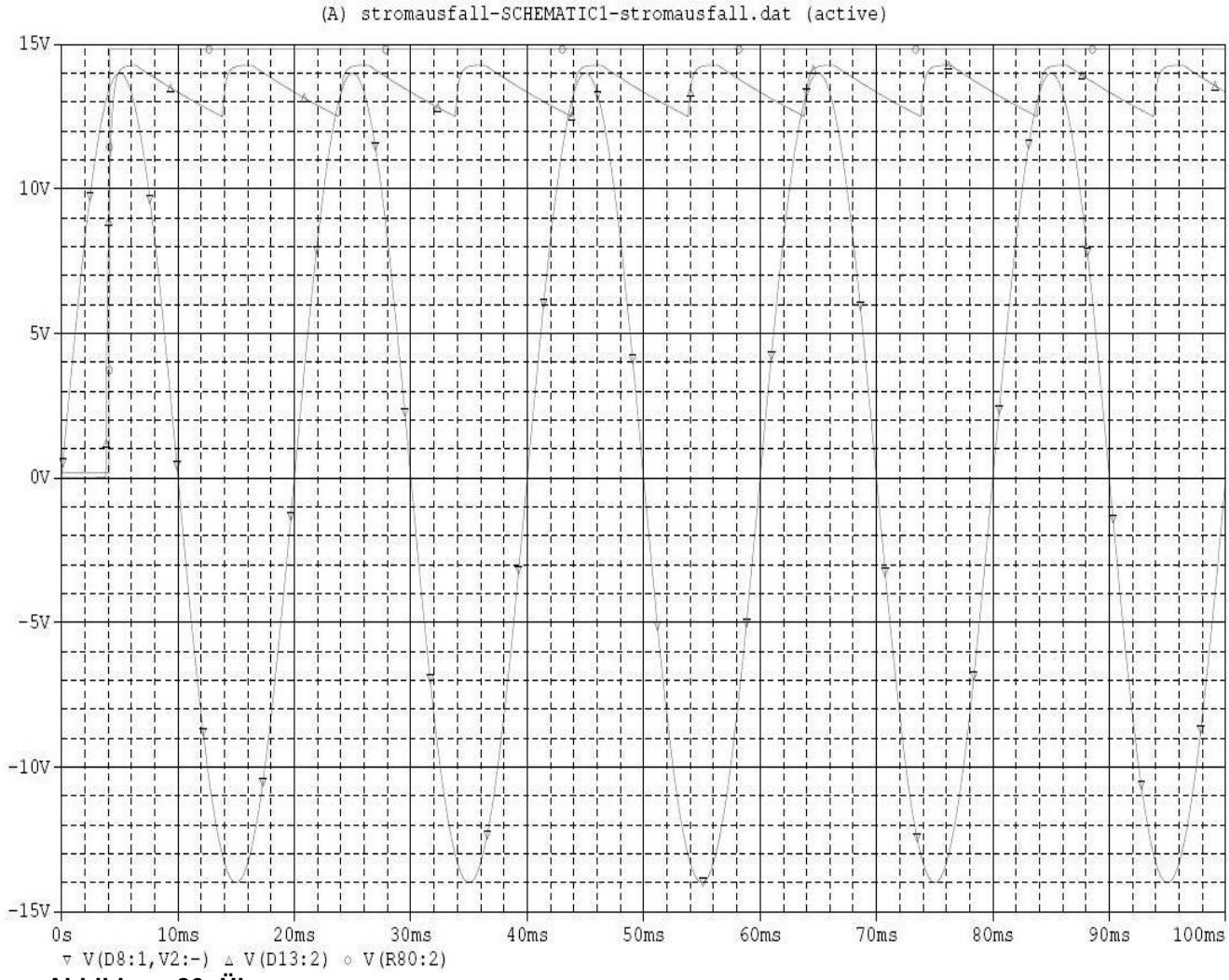


Abbildung 26: Überspannung

3.3. Peakdetektion

Die Peakdetektion befindet sich auf der Analog-, der Digitalteil- und auf der Umschalterplatine. Die Stücklisten und Ätzlayouts der Analogplatine sind unter 3.4 zu finden, während sie für die Digitalteilplatine unter 3.5 und für die Umschaltplatine unter 3.6 sind. Die Schematics sind alle im Anhang unter 10.2.

Prinzipieller Aufbau der Schaltung

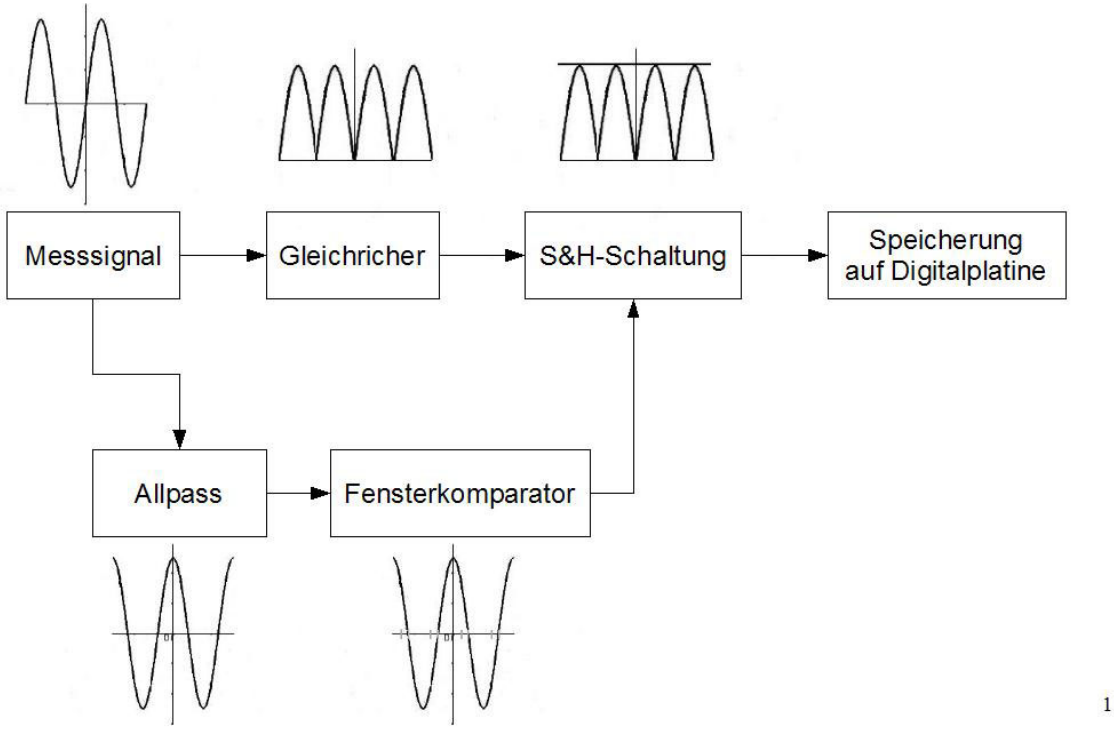


Abbildung 27: BSB Sample&Hold

Eine andere Aufgabe unsere Analogplatine besteht darin, die Scheitelwerte der am Eingang anliegenden Messspannung festzuhalten, und der digitalen Weiterverarbeitung bereitzustellen.
Wir haben uns an dieser Stelle für eine Sample&Hold–Schaltung entschieden. Unsere erste Aufgabe in diesem Bereich war, uns etwas zu überlegen, was den Messzeitraum der S&H–Schaltung steuert, da eine Dauermessung wegen der Wechseleigenschaften des Messsignals keine Werte liefern würde. Wir kamen zu einer Lösung, die einen Allpass zur Phasendrehung um 90° und einen Fensterkomparator zum steuern des Messfensters der S&H-Schalung benutzt.

Allpass

Aufgabe

Der Zweck der Allpassschaltung ist die Phasenverschiebung des Eingangssignals um 90°. Das ermöglicht uns, genau zu wissen, wo die maximalen Höhen der Eingangsspannung anliegen.

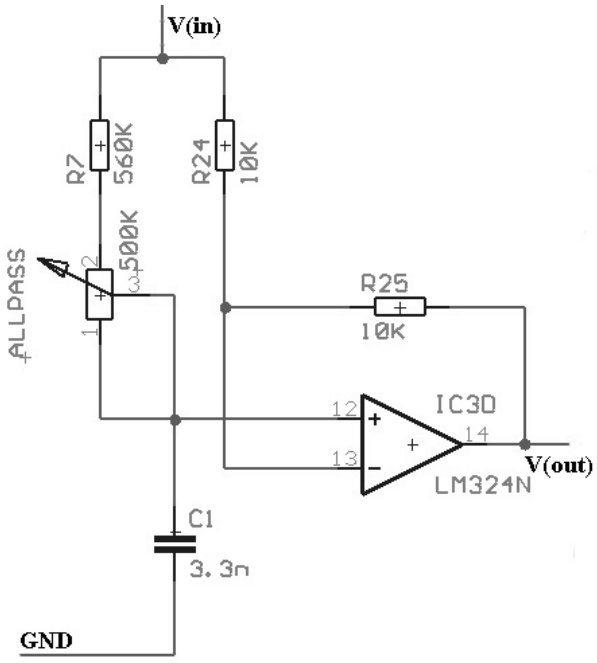


Abbildung 28: Allpass

Funktionsprinzip

Der Allpass hat, wie der Name schon sagt, eine Spannungsverstärkung von nahezu 1 über das ganze Frequenzspektrum. Das entscheidende an einem Allpass ist aber die gut einstellbare Phasenverschiebung. Der Bereich der Phasenverschiebung ist von 0° bis -180°. Für unsere Schaltung brauchten wir 90°, so dass die Nulldurchgänge der Ausgangsspannung des Allpasses relativ genau unter den Maxima des Eingangssignales, oder besser der gleichgerichteten Eingangsspannung, liegen. Da unsere Frequenz nahezu immer 50Hz beträgt, haben wir unseren Allpass wie folgt dimensioniert:

Der Kondensator hat ein Größe von 3.3nF und die Widerstände R7 und der 500 kΩ Spindeltrimmer sollen zusammen 960 kΩ betragen. Da wir den Allpass in der laufenden Schaltung noch anpassen können müssen, haben wir den Widerstand in zwei Teilen realisiert. Der erste ist ein Ohm'scher Widerstand mit 560 kΩ und der zweite Teil ist ein Spindeltrimmer mit maximal 500 kΩ. Dadurch haben wir die Einstellbarkeit erreicht.

Dieser Allpass bildet die Grundlage für die Sample&Hold – Schaltung auf unserer Analogplatine. Wenn wir aber nur diesen Nulldurchgang zum Messen benutzen

würden, bekommen wir nur ein sehr kurzes Signal (exakt der Nulldurchgang). Das würde aber unter Umständen nicht ausreichen, den wirklichen Maximalwert der Spannung festzuhalten. Deshalb haben wir uns entschieden noch einen Fensterkomparator als Hysterese einzuplanen.

Simulation

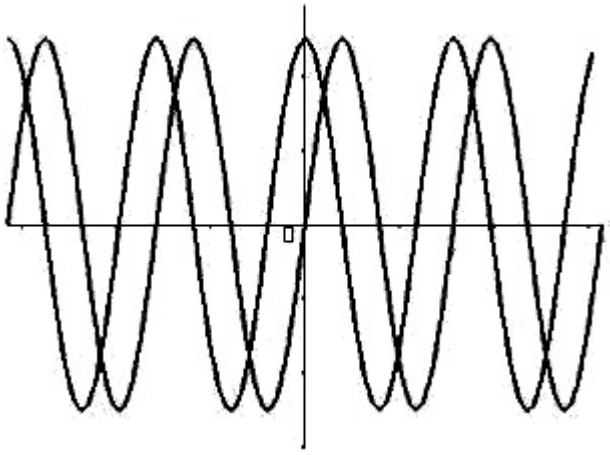


Abbildung 29: Phasenverschiebung beim Allpass

Wie man deutlich sieht, liegt der modellmäßig angenommene Sinus um 90° phasenverschoben zur Eingangsspannung. Damit liegt genau der Nulldurchgang des Allpassausgangs unter dem Maximalwert des Eingangssignals.

Fensterkomparator

Aufgabe

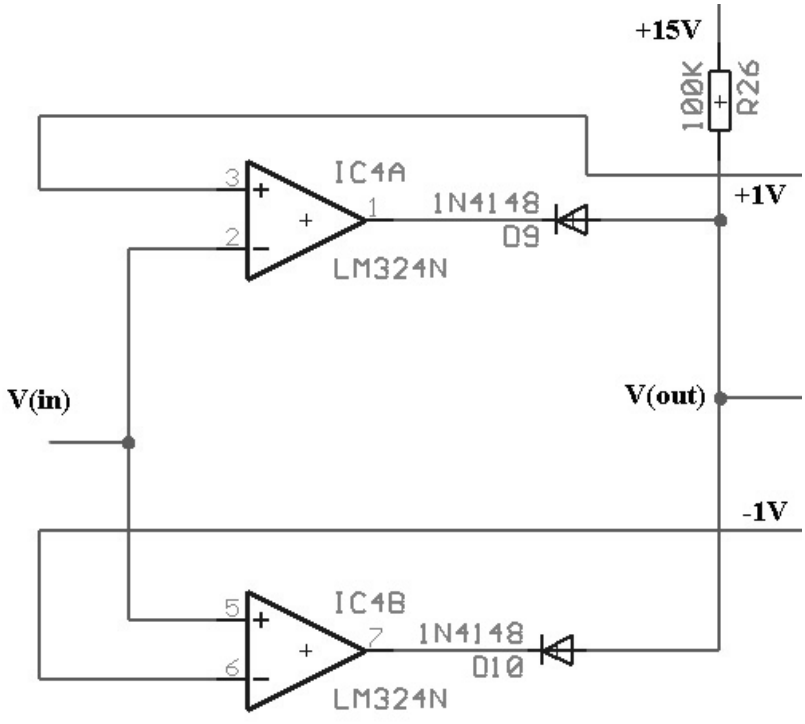


Abbildung 30: Fensterkomparator

Der Fensterkomparator soll das Messfenster für die Sample&Hold-Schaltung erzeugen.

Funktionsprinzip

Der realisierte Fensterkomparator besteht aus 2 Operationsverstärkern, deren positive Aussteuerung durch die zwei Dioden geblockt wird. Als Vergleichsspannungen haben wir jeweils 1 Volt gewählt. Zur genauen Ausrichtung des Fensters haben wir diese Spannungen (+1V, -1V) über Spindeltrimmer einstellbar gemacht. Am Eingang dieser Schaltung liegt das Ausgangssignal der Allpassschaltung an. Am Ausgang der Schaltung liegt ein Rechtecksignal an, welches die Sample&Hold – Schaltung steuert.

Der „Fensterkomparator“ hat also die folgenden Funktionen: Er erkennt erstens die Nulldurchgänge des Ausgangssignales des Allpass und schaltet dann durch und zweitens bietet er durch die, etwas von 0V verschiedenen Vergleichsspannungen, eine Hysterese, die das Messfenster generieren. Je einer der Operationsverstärker übernimmt einen Teil der Halbwellen. Die beiden Dioden verhindern, dass die „negative“ Aussteuerung der beiden Operationsverstärker zur Sample&Hold - Schaltung kommt.
Das so generierte Signal geht direkt zum Analogschalter der Sample&Hold – Schaltung.

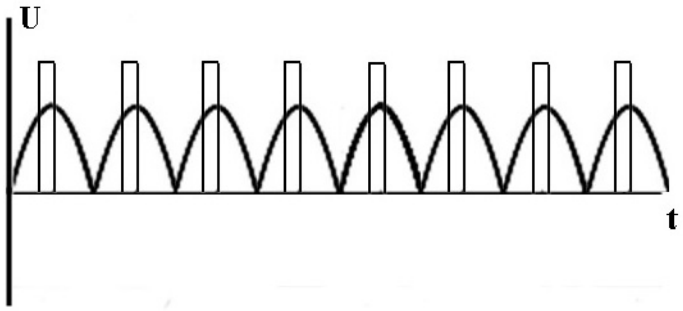


Abbildung 31: Das erzeugte Zeitfenster

Sample&Hold–Schaltung

Aufgabe

Die Sample&Hold–Schaltung soll den Spitzenwert der Netzspannung halten, damit man diesen ausgeben und auch speichern kann.

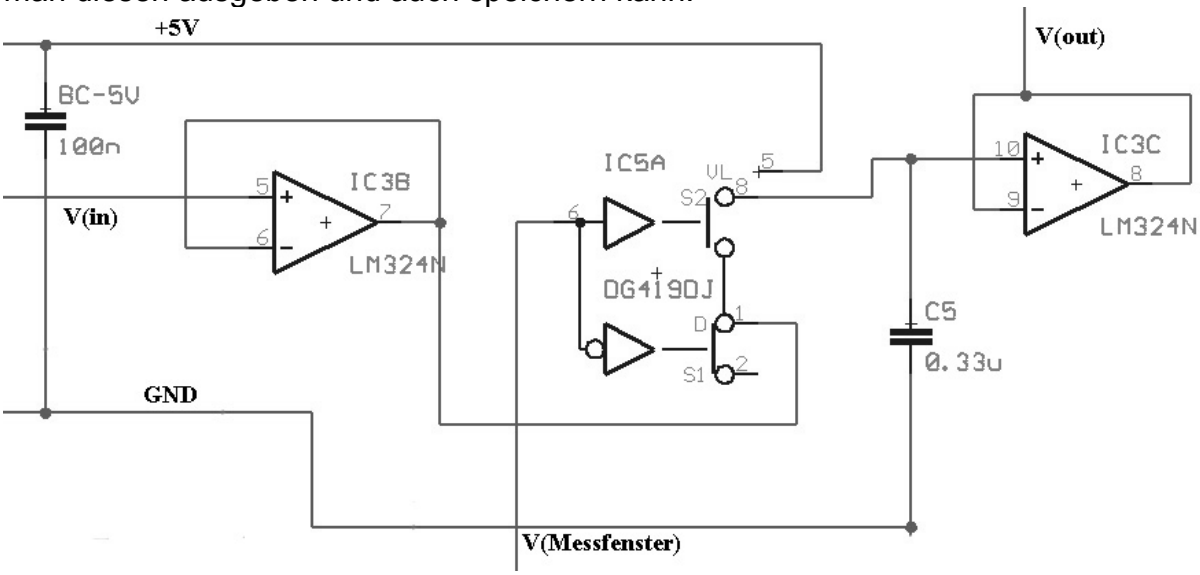


Abbildung 32: Sample&Hold

Funktionsprinzip

Die Aufgabe dieser Schaltung besteht darin, den Scheitelwert des Messsignals festzuhalten und bereitzustellen. Die Schaltung bekommt am Eingang V(in) das gleichgerichtete Messsignal. Alle Halbwellen werden also als positive Wellen betrachtet. Der zweite Eingang ist das Messfenster aus dem Fensterkomparator. Kern der Schaltung ist ein Analogschalter. Dieser dient dazu, den Messvorgang zu steuern. Wenn dieser Schalter durchschaltet, wird ein Messwert genommen und durch die Kapazität C5 gespeichert. Die Operationsverstärkerschaltung am Ausgang dient zur Entkopplung dieser Schaltung von der Digitalschaltung, die diesen Wert weiter verarbeitet. Auch der Eingangsoperationsverstärker dient zur Entkopplung. Der Kondensator BC – 5V dient, als Blockkondensator, zur Vermeidung hochfrequenter Spannungsanteile, die die Schaltung negativ beeinflussen.

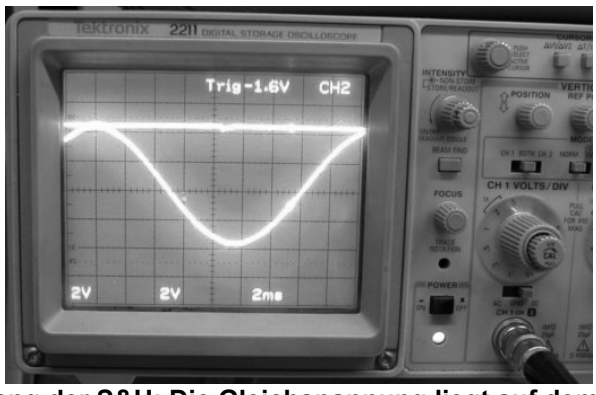


Abbildung 33: Ausgang der S&H: Die Gleichspannung liegt auf dem Scheitel des Eingangs

Vergleichsspannungen

Die Vergleichsspannungen für die Überspannung/Stromausfall-Detektion sowie für den Fensterkomparator wurden durch Spannungsregler realisiert.

Die Referenzdioden, die dabei verwendet werden eignen sich optimal zu Erzeugung der Referenzspannungen, da sich die Spannung an der Diode sehr unempfindlich gegenüber veränderlichen Strömen verhält.

Abbildung 34: Symbol der Diode

Die Besonderheit dieser Diode ist, dass sie zwischen der Referenz und der Anode immer genau 2.5V hält.

Berechnung der Referenzspannungen

Die Referenzspannungen werden über Spindeltrimmer justiert. Um dies zu gewährleisten befinden sich abgriffe für die Referenzspannungen auf der Platine zum nachjustieren. Nur so können wir gewährleisten, dass die Referenzspannungen unter verschiedenen Arbeitsbedingungen stimmen.

+9V Referenzspannung

Über A-R der Diode fallen bauartbedingt immer 2.5V ab. Mit dem R31 regelt man also den Strom der für die gewünschte Spannung fließen soll.

$$mit\ R = \frac{U}{I}\ folgt: \qquad \frac{2{,}5V}{1k\Omega} = 2{,}5mA$$

Über der Diode sollen unsere 9V abfallen. Somit muss über R-C (Referenz-Kathode) 6,5V abfallen.

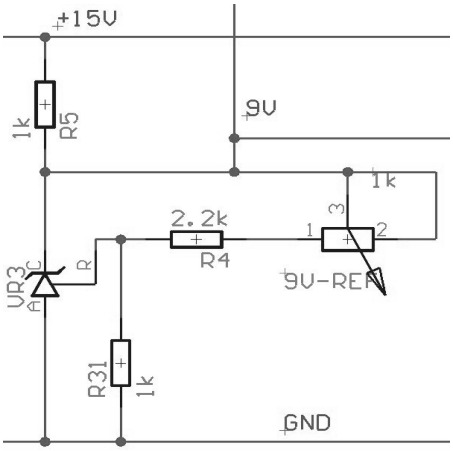


Abbildung 35: +9V Ref.

$\frac{6,5V}{2,5mA} = 2,6k\Omega$ Also muss gelten: R4+1kΩPoti=2,6kΩ also muss das Poti ungefähr auf 400 Ohm eingestellt werden.

+10,6V Referenzspannung

Über Anode und Referenzanschluss der Diode fallen wieder 2.5V ab. Also liegen am R32 2.5V an.

mit $R = \frac{U}{I}$ folgt $\frac{2,5V}{1k\Omega} = 2,5mA$ Also ist der Strom durch den Spannungsteiler 2.5mA groß.

Damit zwischen Anode und Kathode +10.6V anliegen müssen zwischen Referenz und Kathode 8.1V abfallen:

$\frac{8,1V}{2,5mA} = 3,24k\Omega$ Also muss gelten: R34+1kΩPoti=3,24kΩ somit muss der Spindeltrimmer auf ca. 540 Ohm eingestellt werden.

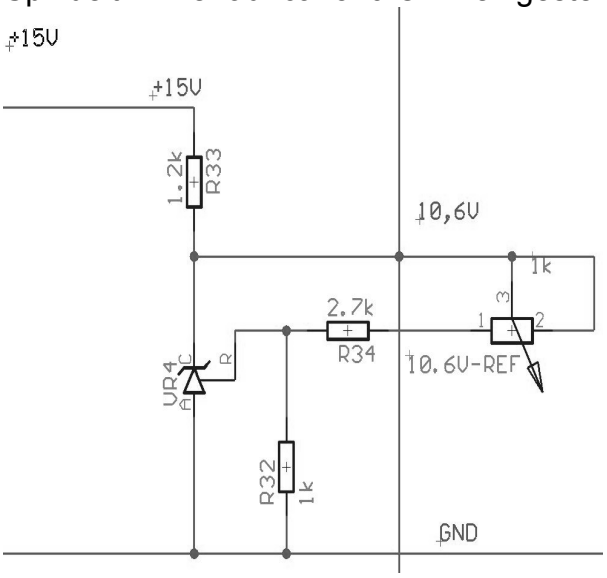


Abbildung 36: +10,6V Ref.

+1V Referenzspannung

Zwischen Anode und Referenz liegen 2.5V an. Damit müssen über R29, R23 und dem Spindeltrimmer insgesamt auch 2.5V anliegen. Außerdem müssen über R33 12.5V abfallen.
Damit über R23 genau +1V anliegt muss folgendes gelten:
Der Strom durch R23 muss 1mA groß sein und über R29 und dem Spindeltrimmer fallen die restlichen 1.5 ab.

$\frac{1,5V}{1mA} = 1,5k\Omega$ Da R29 1K Ohm groß ist, muss der Spindeltrimmer auf ca. 500 Ohm justiert werden.

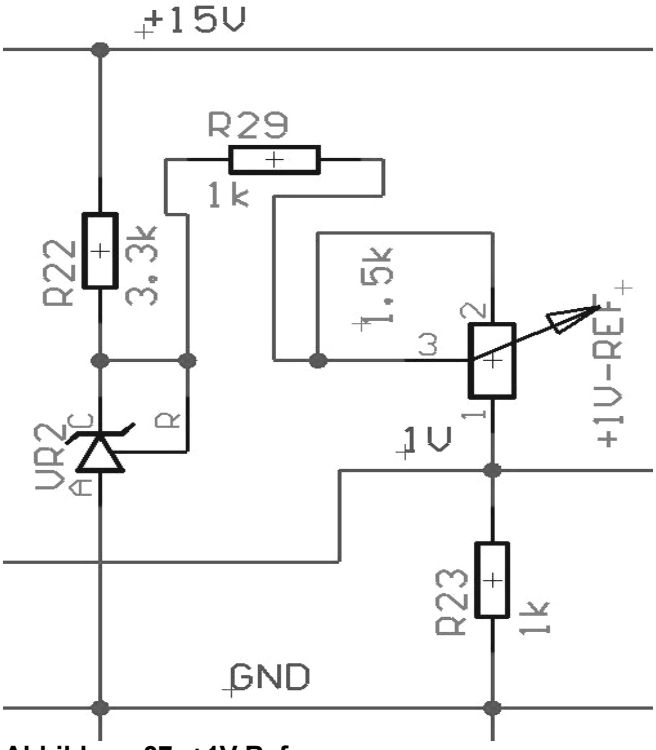


Abbildung 37: +1V Ref.

-1V Referenzspannung

Die Erzeugung der Referenzspannung erfolgt Analog zur Erzeugung der +1V Referenzspannung. Mit dem Unterschied, dass die Versorgungsspannung -15V beträgt und somit letztendlich -1V am R6 anliegen.

Über dem Referenzeingang der Diode und der Anode müssen -2.5V anliegen. Demzufolge muss über R27 -12.5V abfallen und über R6, R28 und dem Spindeltrimmer müssen insgesamt -2.5V abfallen.

Über R6 kann nur -1V abfallen, wenn dort ein Strom von -1mA fließt: $\frac{-1V}{1k\Omega} = -1mA$

Die restlichen -1.5V müssen über R28 und dem Spindeltrimmer abfallen. Also müssen wir den Spindeltrimmer wie folgt dimensionieren:

$\frac{-1,5V}{-1mA} = 1,5k\Omega$ also: R28+1,5kΩPoti=1,5kΩ und: 1kΩ+500Ω=1,5kΩ

Auch hier muss der Spindeltrimmer auf 500 Ohm justiert werden.

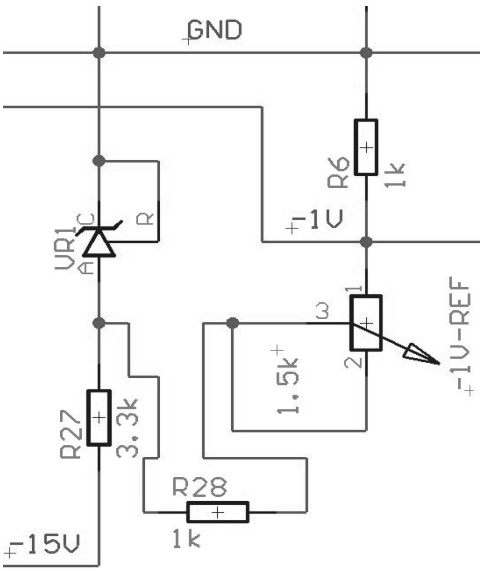


Abbildung 38: -1V Ref.

Funktionsweise der Schmitt-Trigger

Da auf der Analogplatine viele OPVs mit Schmitt-Trigger verbaut wurden soll hier auch noch ihre Funktionsweise erläutert werden.

Hierbei Handelt es sich um einen Komparator mit Hysterese:

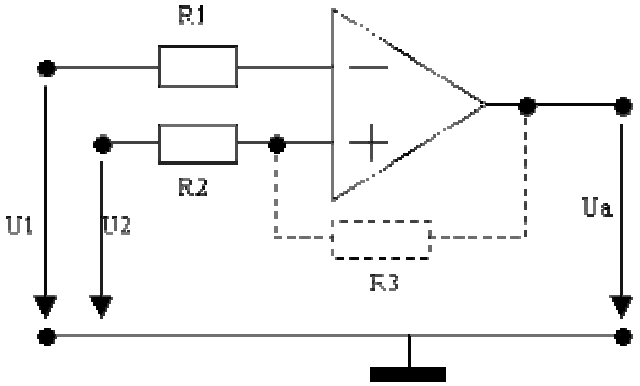


Abbildung 39: Komparator mit Hysterese

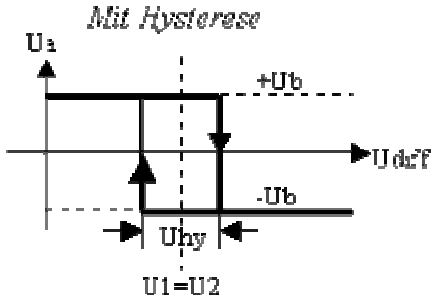


Abbildung 40: Hysteresekurve

$$U_{Hy} = \frac{R2}{R2+R3} \bullet \Delta Ua$$

Formel 1: Hysteresespannung

Es bildet sich ein Spannungsteiler zwischen R2 und R3 (R1 ist keine Pflicht, sollte aber gleich R2 sein (da der OPV eben doch einen kleinen Strom zieht.
Dadurch entsteht eine Hysteresekurve, welche verhindert, dass der OPV bei 2 gleichen Signalen hin und her schwingt.

Die oben beschriebenen Teilschaltungen sind alle auf der Analogplatine, die nachfolgenden Teilschaltungen sind zum Speichern der Peakwerte und befinden sich auf der Digitalteilplatine.

Aufgabe der Platine

Die Schaltung dient zum Halten der Minimal- und Maximalwerte.
Sie erhält eingangsseitig von der Analogplatine den aktuellen Spitzenwert der Netzspannung als herunterskalierte Gleichspannung und übergibt den Maximal- und Minimalwert als paralleles Digitalsignal an die Umschaltplatine (je 10 Bit).
Die Werte können bei Bedarf über ein Reset-Signal (Bus Pin 17) zurückgesetzt werden.

Vorige Schaltungen zur Speicherung der Spitzenwerte:

Die Speicherung sollte mit Hilfe eines Kondensators erfolgen. Dabei stießen wir auf die Problematik der bauteilbedingten Verluste im Laufe der geforderten maximalen Speicherzeit von 24 Stunden. Keine uns bekannte Schaltung zur Speicherung mit Hilfe eines Kondensators führte zu einer Lösung des Problems.

Aufgrund dessen entschieden wir uns die Speicherung digital zu realisieren.
Unsere erste Idee war das Analogsignal mit Hilfe eines AD-Wandlers zu digitalisieren und dann digital Maximum und Minimum zu halten. Dieser Ansatz scheiterte jedoch bereits in der frühen Planungsphase. Einerseits durch die Komplexität des Gebietes der AD-Wandler, zum anderen durch die viel zu aufwändige digitale Schaltung zur Auswertung des gewandelten Analogsignals. Diese Schaltung wäre mit Hilfe eines Mikrocontrollers zwar auf angemessene Dimensionen zu reduzieren gewesen, allerdings haben wir uns zunächst bewusst gegen diese Möglichkeit entschieden um nach weiteren, eher analog gehaltenen Schaltungen zu forschen. Wir behielten uns diese Variante als Alternative vor, falls wir zu keiner anderen Lösung gelangen würden.

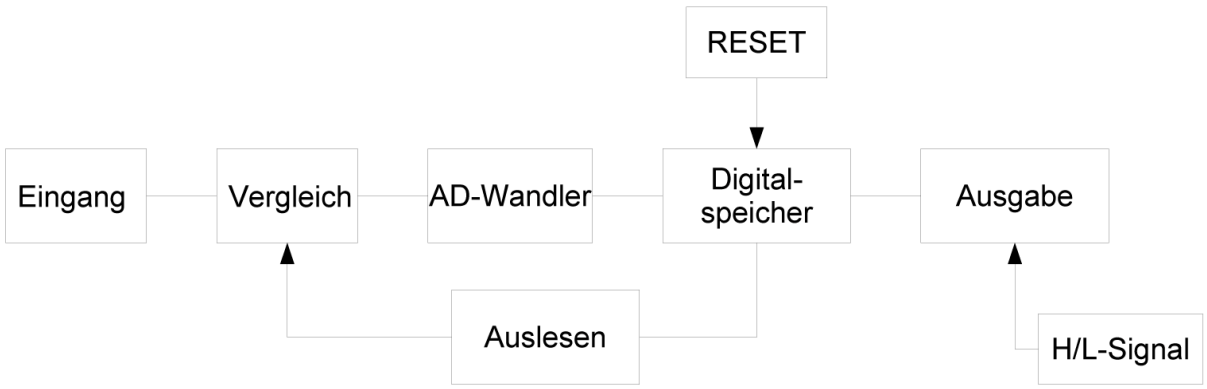


Abbildung 41: BSB der Digitalplatine

Prinzipieller Aufbau der Min./Max.-Speicherung

Letztendlich entwickelten wir die Idee die Speicherung digital vorzunehmen, allerdings analog zu vergleichen:

Die Schaltung wird als Erstes resetet, somit steht der Zähler auf dem Wert Null, demzufolge liegen am Ausgang des DA-Wandlers 0V an. Das zu bearbeitende Signal aus der Analogplatine wird mit einem Komparator mit dem Ausgang des DA-Wandlers verglichen. Wenn das Signal größer ist als das des DA-Wandlerausgangs gibt der Komparator den Takt über das Und-Gatter frei und der Zähler zählt aufwärts. Damit wird am Ausgang des DA-Wandlers das Maximalsignal gehalten.

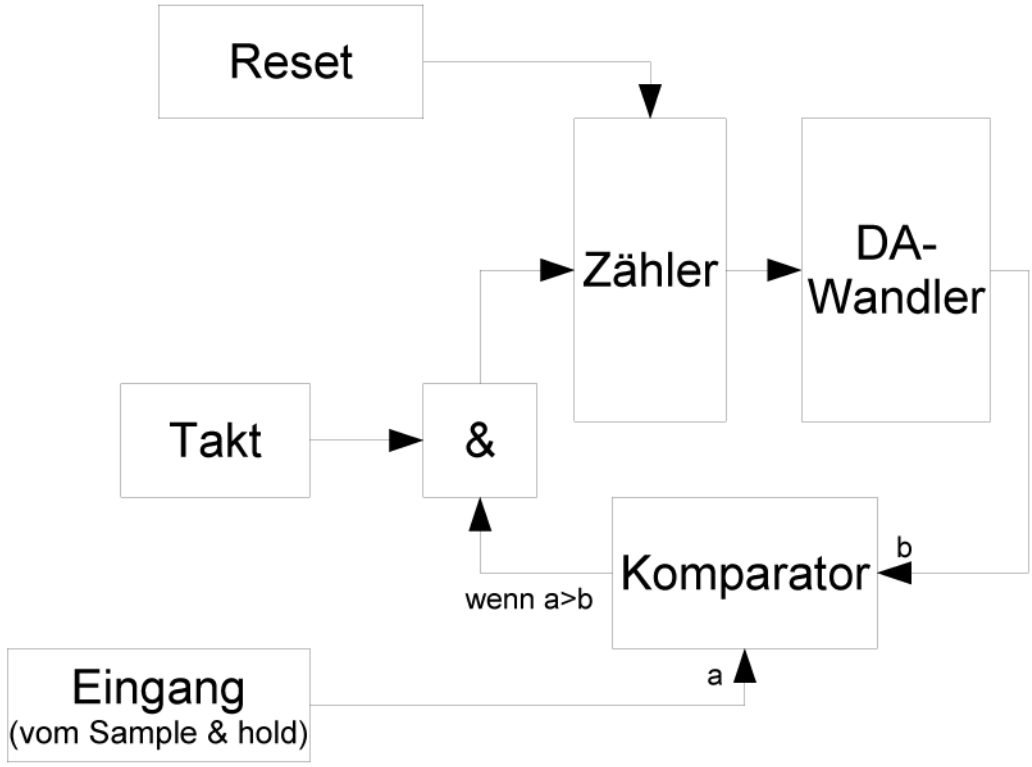


Abbildung 42: BSB Speicherung der maximalen Spitzenwerte

Für den Minimalwert wird nach dem Reset der Zähler auf den Maximalwert gesetzt. Ist das Signal der Analogplatine kleiner als das des DA-Wandlerausgangs, schaltet der Komparator den Takt frei, wodurch der Zähler abwärts zählt.

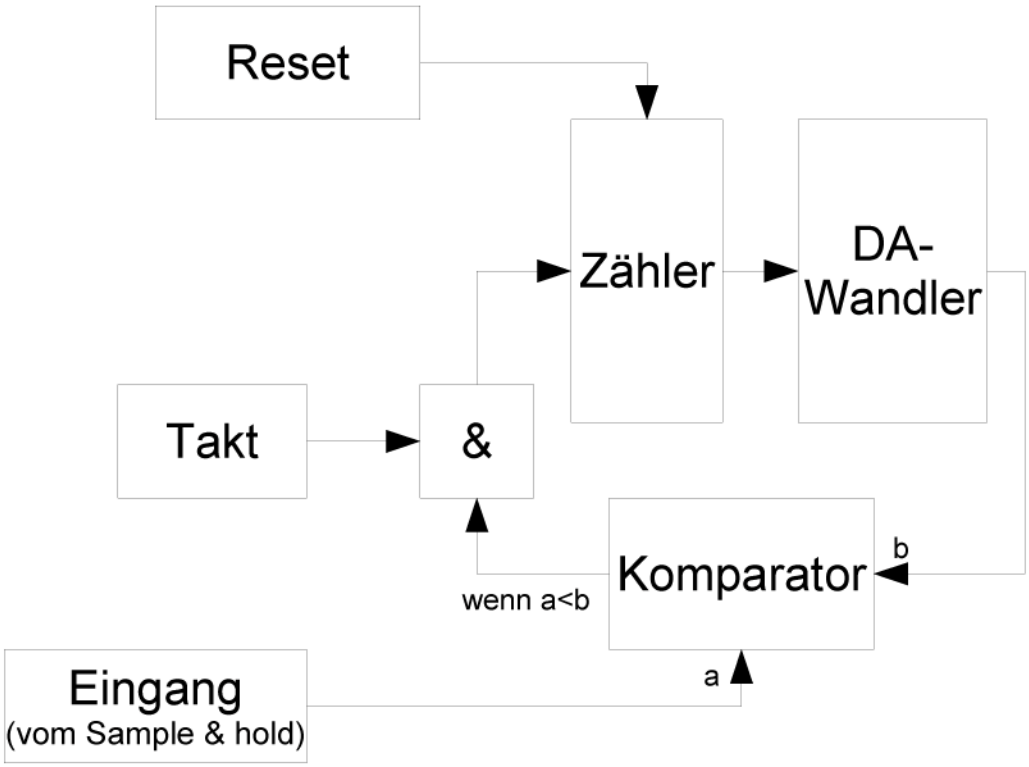


Abbildung 43: BSB Speicherung der minimalen Spitzenwerte

Eingangsteiler

Aufgabe

Herunterskalierung der Spannung.

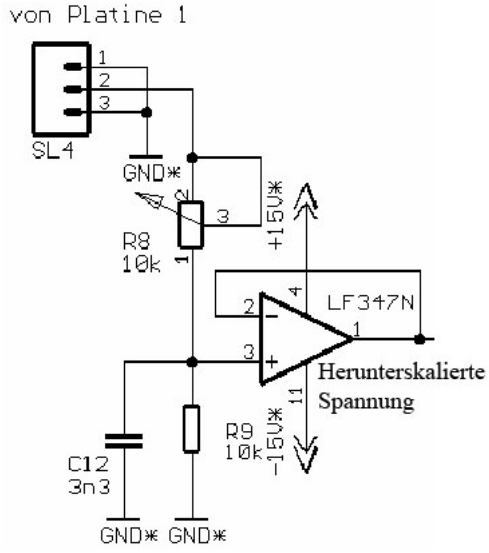


Abbildung 44: Eingangsteiler

Funktionsprinzip

Das Signal von der Analogplatine wird über einen einstellbaren Spannungsteiler auf die passende Spannung herunterskaliert und danach durch einen Impedanzwandler entkoppelt. Auf Grund der Leitungslängen ist es sinnvoll das Signal mit einem Kondensator zu blocken.

Taktteiler

Aufgabe

Die für die Schaltung zu hohe Frequenz des Quarzoszillators soll reduziert werden.

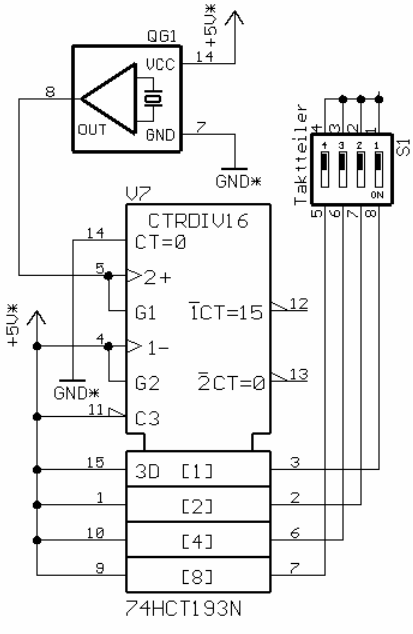


Abbildung 45: Taktteiler

Funktionsprinzip

Mit dem Zähler kann die Frequenz des Quarzoszillators (1MHz) geteilt werden. Mit dem DIP-Schalter kann man sich den gewünschten Teilungsfaktor auswählen.
1: 500kHz
2: 250kHz
3: 125kHz
4: 62,5kHz

Wichtig: Nur einen Schalter des DIP-Schalters auf ON stellen, da sonst Ausgänge kurzgeschlossen werden.

Minimum-Taktblocker

Aufgabe

Problembehebung bei den 4-Bit Zählern.

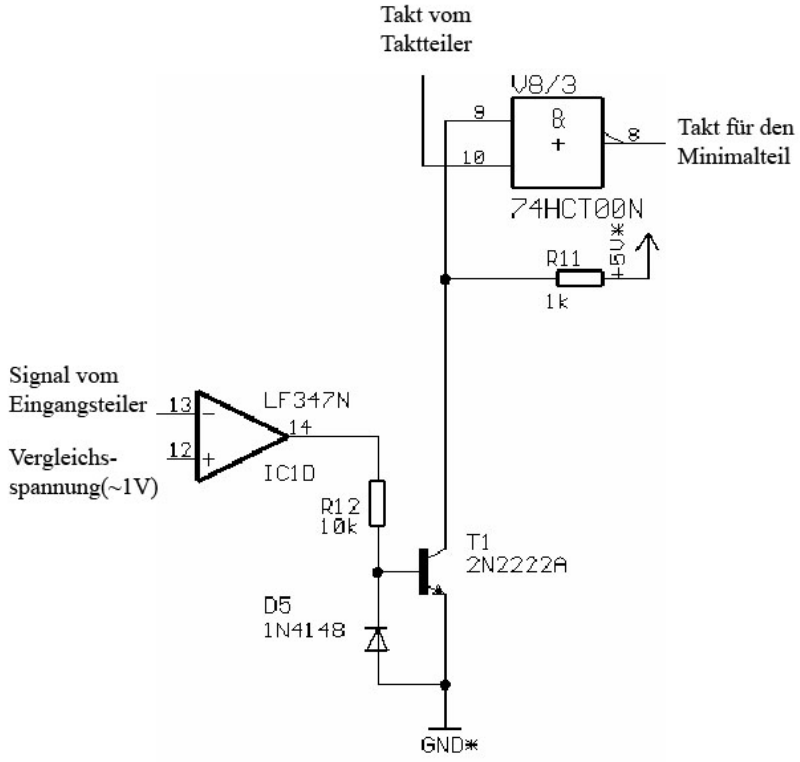


Abbildung 46: Minimum-Taktblocker

Funktionsprinzip

Bei dem Minimalteil der Schaltung besteht das Problem, dass die Zählerkette ab einem gewissen Minimum in einer Endlosschleife hängt, da Teile der Gesamtschaltung nicht mehr korrekt arbeiten. Daher wird unterhalb dieser Schwelle der Takt für den Minimumzähler gesperrt.
Da der OPV als Komparator am Ausgang nur +15V oder -15V ausgibt muss diese Spannung auf TTL-Pegel angepasst werden.
Die Diode D5 dient nur dem Schutz des Transistors.

Maximalwerthalter

Aufgabe

Speicherung des maximalen Spitzenwerts der Netzspannung.

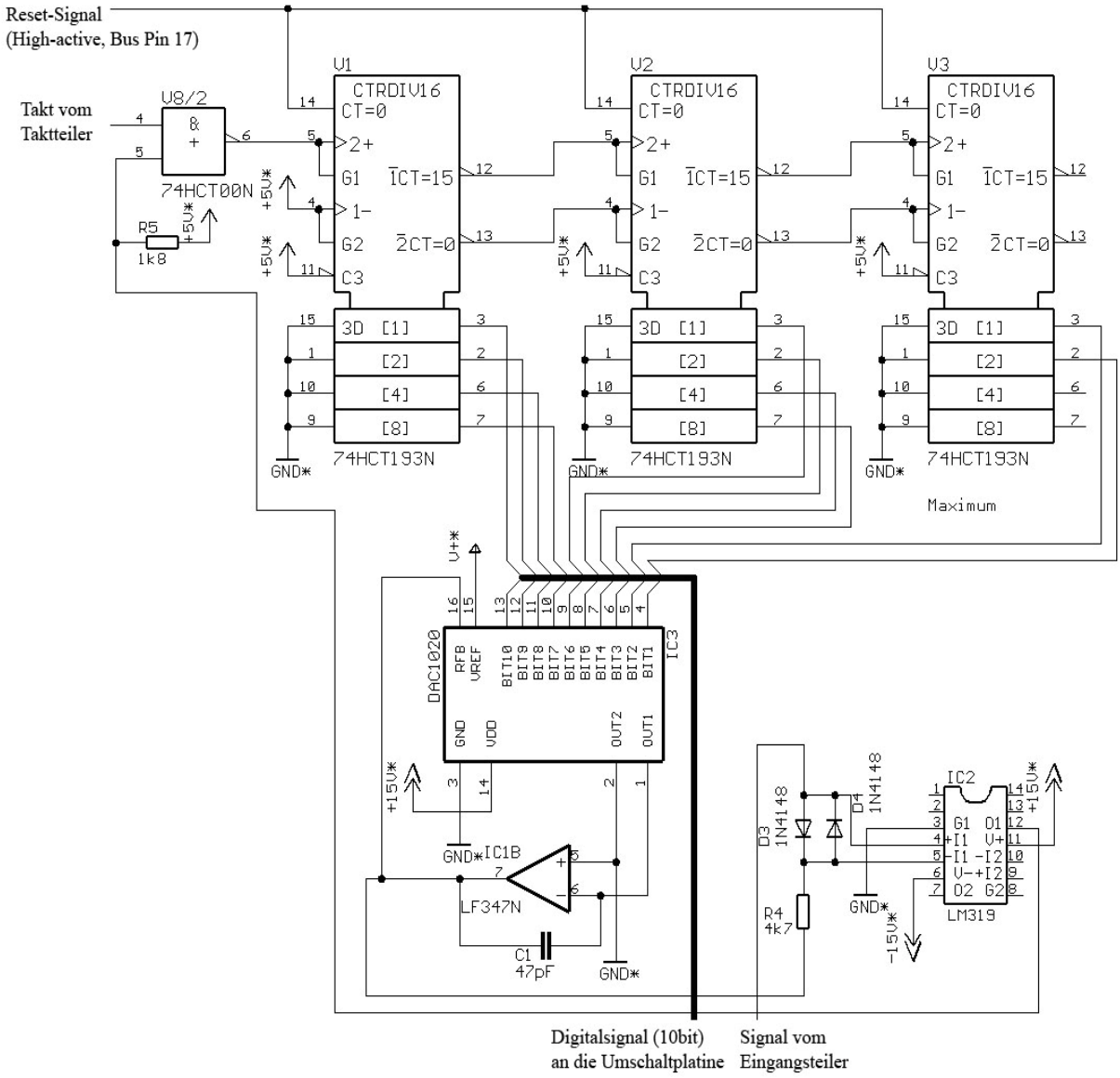


Abbildung 47: Maximalwerthalter

Funktionsprinzip

Das Signal vom Eingangsteiler wird mit dem Ausgang des DA-Wandlers verglichen. Wenn das Eingangsteilersignal größer ist, gibt der Komparator mit Hilfe des NAND-Gatters den Takt für den Zähler, welcher aus drei 4Bit-Zählern besteht, frei. Der Zähler hört auf aufwärts zu zählen, wenn der Ausgang des DA-Wandlers das Eingangsteilersignal erreicht hat.

Der OPV (IC1B) gehört zum DA-Wandler. Die Beschaltung ist dem anliegenden Datenblatt des DAC1020 zu entnehmen. An dem Ausgang des OPVs liegt die der Referenzeingangsspannung und den Signalen an den digitalen Eingängen entsprechende Ausgangsspannung an. Berechnung siehe folgende Formel:

$$U = -U_{ref} \cdot \frac{1023}{1024}$$

.

Wenn ein Hi-Lo-Puls auf der Resetleitung vorliegt, werden die Zähler auf Null gesetzt.
Die Dioden D3/D4 und der Widerstand R4 dienen nur dem Schutz des Komparators (LM319), da dieser nur eine Eingangsspannungsdifferenz von ±5V verträgt.
R5 ist der Pullup-Widerstand für den Open-Collector-Ausgang des LM319 (Emitter auf GND).

Minimalwerthalter

Aufgabe

Speicherung des minimalen Spitzenwerts der Netzspannung.

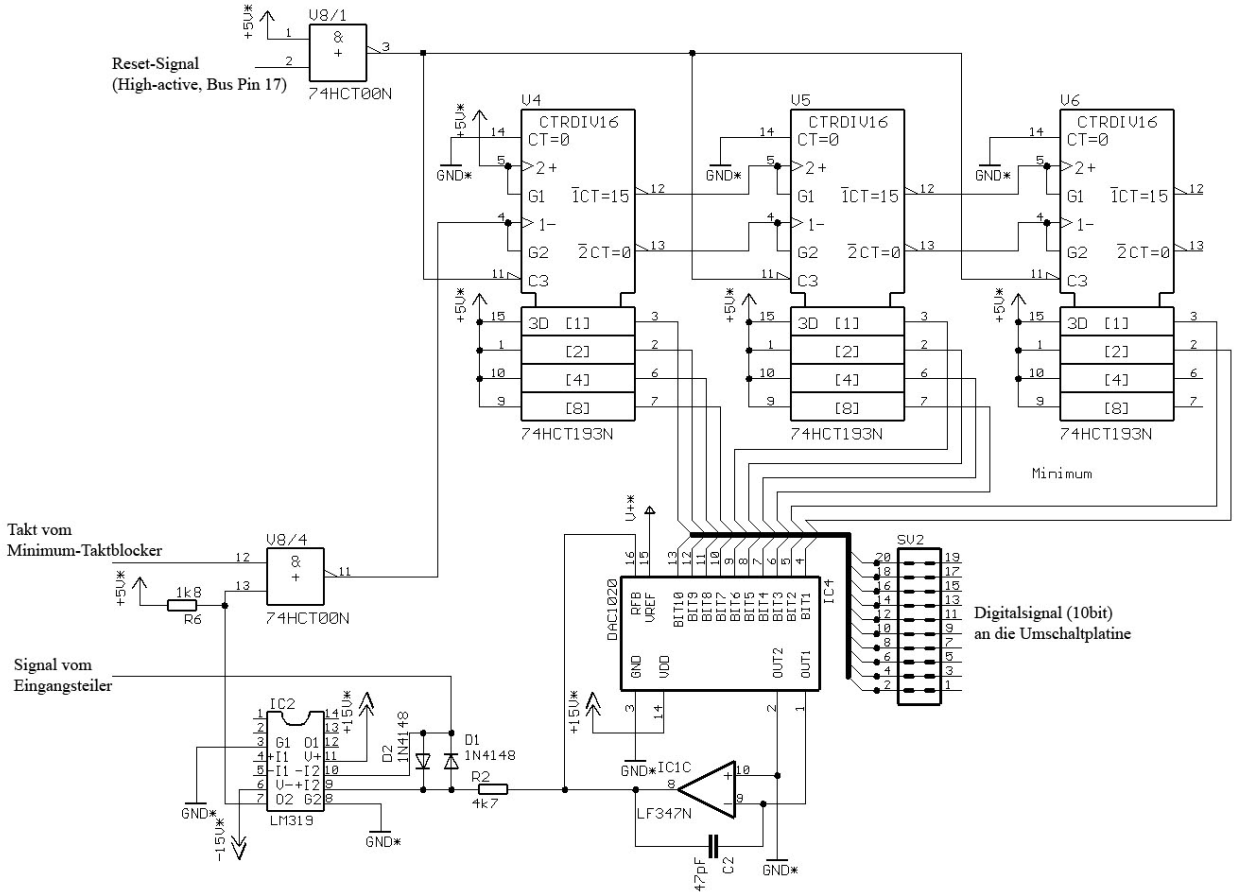


Abbildung 48: Minimalwerthalter

Funktionsprinzip

In dieser Teilschaltung wird der Minimumwert gespeichert. Die Arbeitsweise dieser Teilschaltung ist mit der vorigen weitgehend identisch, bis auf die Tatsache, dass die Zähler abwärts zählen. Dementsprechend sind die Komparatoreingänge vertauscht und zum Takten wird nicht der Aufwärts- sondern der Abwärtszähleingang verwendet.
Wenn ein Hi-Lo-Puls auf der Resetleitung anliegt, wird dieser durch das NAND-Gatter (V8/1), welches hier als Inverter verwendet wird, invertiert. Dies ist notwendig, da dieses Signal nicht auf die Reset-Eingänge der Zähler, sondern auf die Load-Eingänge (Pin 11), welche Low-active sind, gehen muss. Somit werden die Zähler nicht auf 00000000 sondern auf 11111111 gesetzt werden; zu diesem Zweck liegen alle Data-Inputs (Pins 15, 1,10 und 9) auf High.

Aufgabe der Umschaltplatine

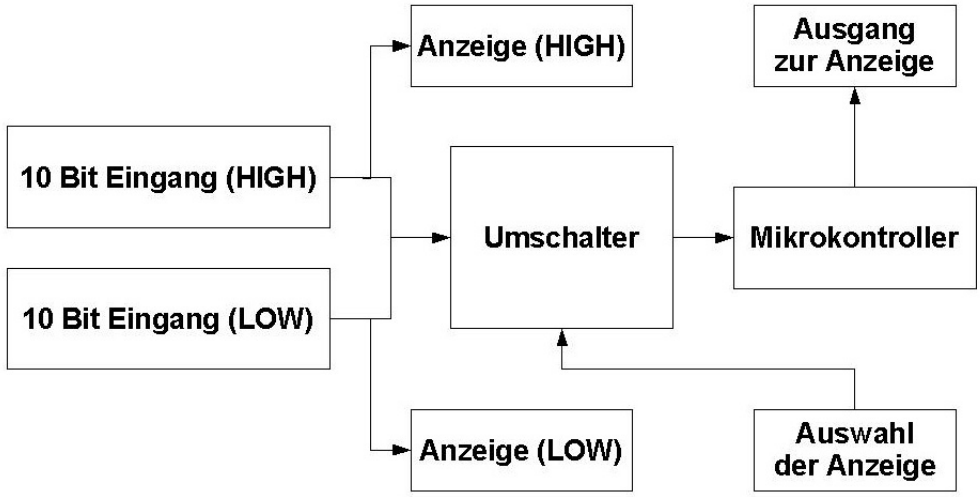


Abbildung 49: BSB der Umschaltplatine

Diese Platine ergab sich aus der Aufgabenstellung der Anzeigegruppe, welche mit Hilfe eines Steuersignals unsere High- oder Low-Werte auswählen und auf einen Mikrokontroller gelegt haben wollte. Des Weiteren benötigten wir zwingend, eine von der Anzeigegruppe unanhängige, visuelle Fehlerdiagnose.

Der Mikrocontroller

Aufgabe

Der Mikrocontroller soll die gespeicherten minimalen und maximalen Spitzenwerte der Netzspannung an die Gruppe der Anzeige seriell übertragen.

Funktionsprinzip

Die genaue Funktion hat die Gruppe 4 – Anzeige beschrieben, da der Mikrocontroller von dieser Gruppe stammt und nur auf unserer Platine eingebaut wurde.

Die visuelle Fehlerdiagnose

Aufgabe

Diese Schaltung hat die Aufgabe, den benötigten Strom für die Diagnose-LED's zu liefern, da diese nicht direkt von den Zählern gespeist werden können.

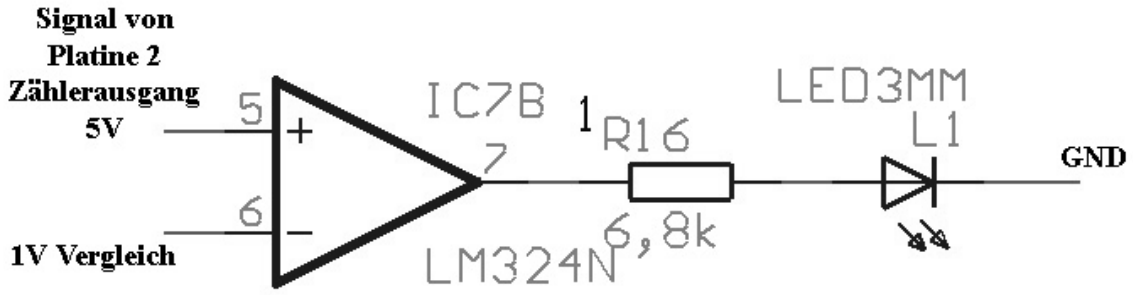


Abbildung 50: Einer der 20 Treiber mit Vorwiderstand und LED

Funktionsprinzip

Der Operationsverstärker LM324N wird als Komparator genutzt. Er vergleicht 1 Volt Vergleichsspannung am negativen Eingang mit der 5 Volt TTL-Spannung aus dem Zähler. Wenn der Zähler auf dem jeweiligen Ausgang einen high-Pegel hat, steuert der Komparator mit +13,5 Volt durch. Der Vorwiderstand begrenzt den Strom für die low-current-LED auf 2 mA, nach dem Ohm'schen Gesetz ergibt sich somit der Widerstand mit 6,8 kΩ. Der OPV wird asynchron betrieben, d.h. mit +15 Volt und GND.

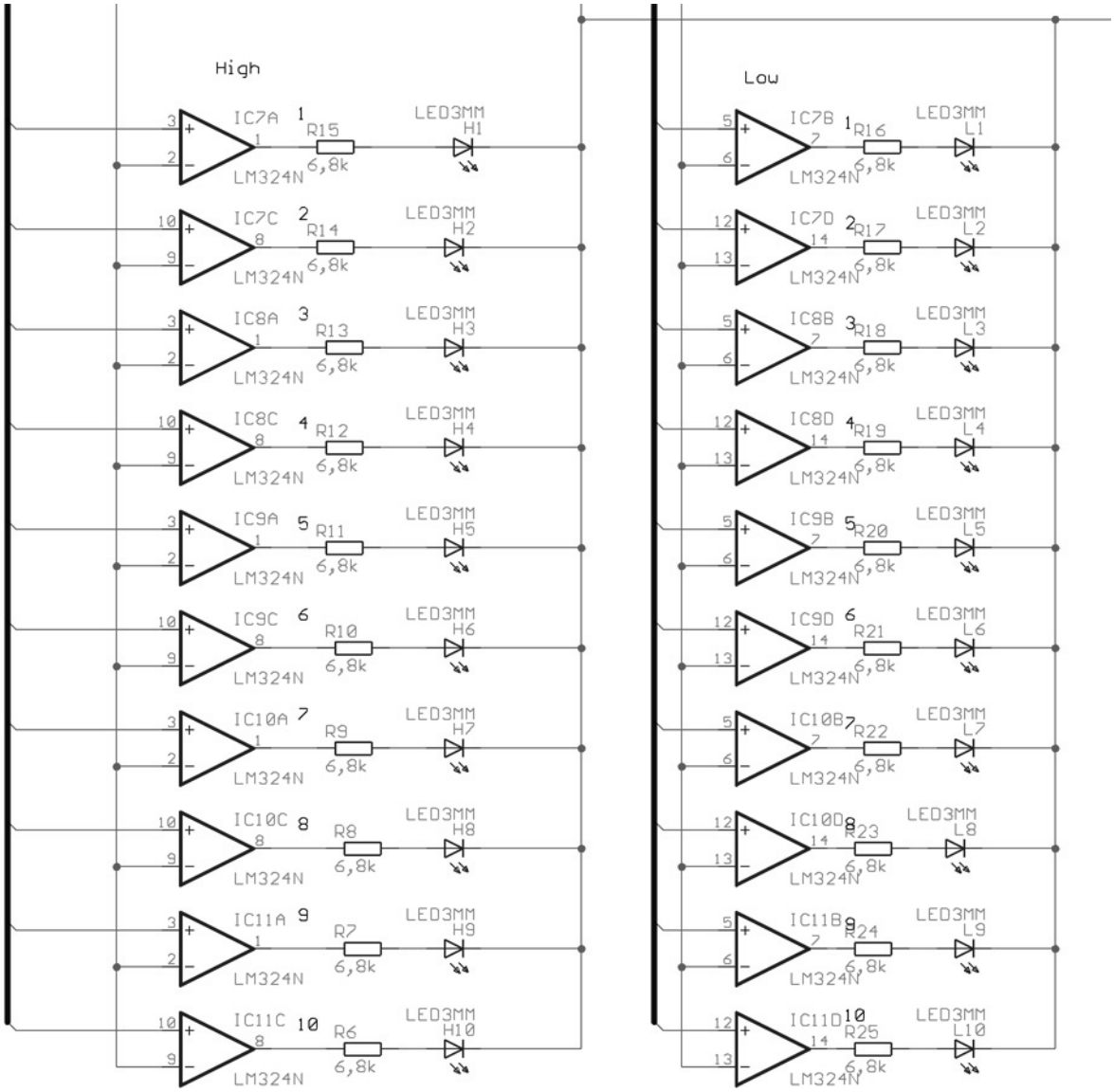


Abbildung 51: Die komplette Treiberstufe für die 2 mal 10 Bit

Der Umschalter

Aufgabe

Diese Schaltung hat die Aufgabe die 2 mal 10 Bit aus der Platine 2, je nach Auswahl der Anzeige, auf den Mikrokontroller zu schalten.

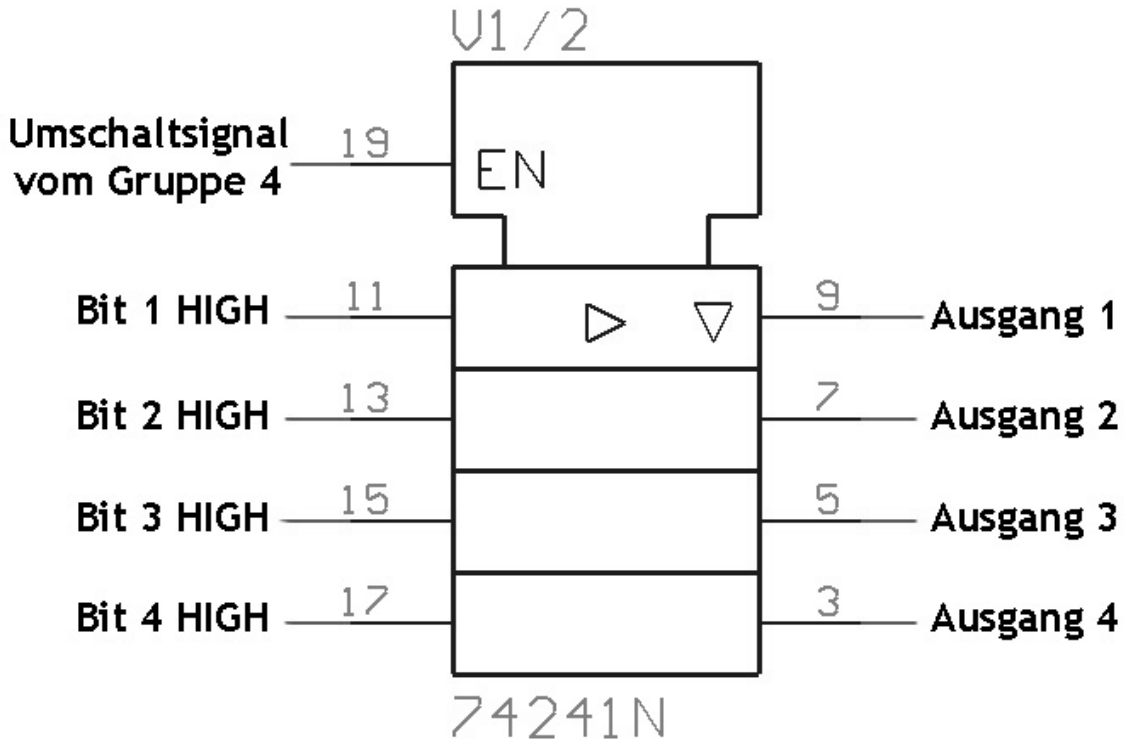


Abbildung 52: Einer der 3 Tri-States für die Umschaltung der HIGH-bits (nicht invertierter Eingang)

Früherer Ansatz

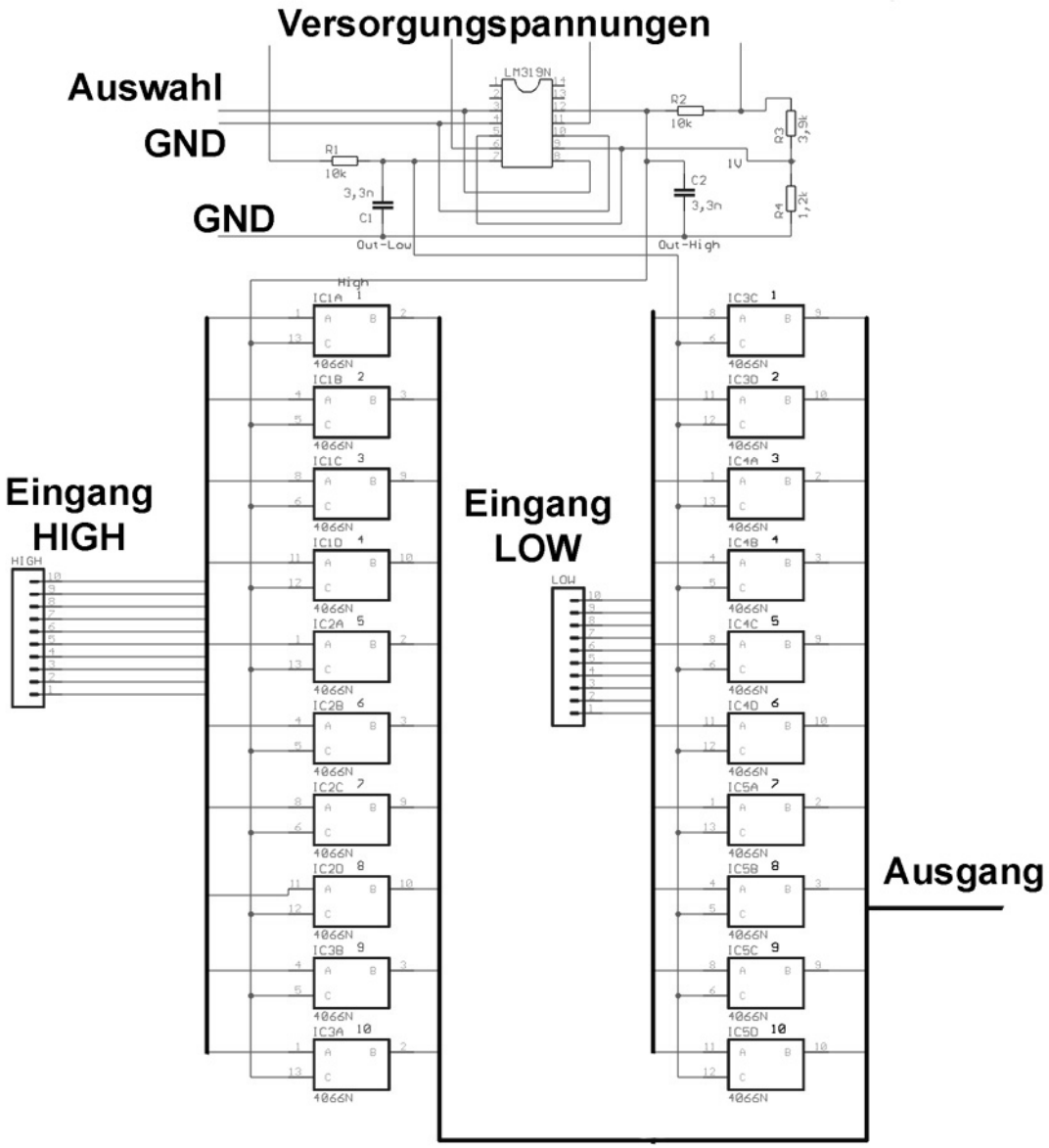


Abbildung 53: Voriger Ansatz mit Analogschaltern

Mit Hilfe von Analogschaltern sollte das Signal umgeschaltet werden. Dazu wurde zwischen jeden Ausgang und die 2 dazu passenden Eingänge, jeweils ein Analogschalter gesetzt, welcher immer dann durchschalten sollte, wenn das Auswahlsignal dies vorgab. Dabei entstand das Problem, dass die Analogschalter mit einer Zeitverzögerung geschaltet werden mussten, weil sonst die 2 Eingänge kurzgeschlossen werden hätten können. Des Weiteren mussten aus einem HIGH-LOW Signal 2 separate HIGH-Signale generiert werden. Dazu wurden zwei Komparatoren (LM319) mit einem RC-Glied zur Zeitverzögerung ausgestattet und auf die Analogschalter geschaltet.
Dieser Vorschlag wurde auf Grund einer einfacheren Lösung verworfen.

Funktionsprinzip

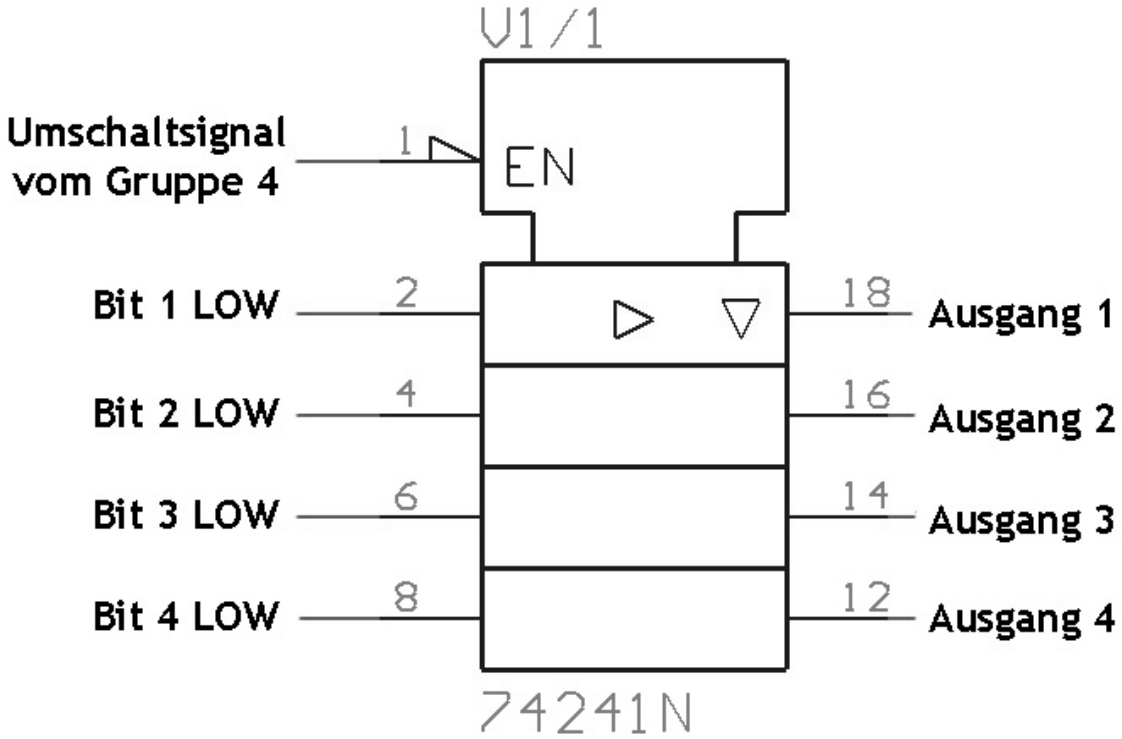


Abbildung 54: Der selbe 3 Tri-State für die LOW-Bits (invertierter Eingang)

Ist das Umschaltsignal der Anzeige-Gruppe LOW (ca. 0V), so werden an dem 3-State mit dem nicht invertierten Eingang (oberes Bild) die Ausgänge hochohmig und spielte somit keine Rolle mehr.
Der 3-State mit dem invertierten Eingang jedoch schaltet die Eingangssignale direkt auf die Ausgänge durch, es werden somit nur die LOW-Bits übergeben.
Analog dazu verhält sich der 3-State für ein HIGH-Signal, hierbei werden aber die HIGH-Bits durchgeschaltet, und somit dem Mikrokontroller zugeführt.
Die Ausgänge von LOW und HIGH werden verbunden, somit steht genau ein 10 Bit Ausgang zur Verfügung.

3.4. Analogplatine

Schnittstellen

Auf dem Bus

Eingänge:
- +15V (Pin 1)
- -15V (Pin 2)
- +5V (Pin3)
- GND (Pin4)
- 9V~ (Pin7)

Ausgänge:
- Stromausfall (5V, Pin 10, Low Active)
- Überspannung (5V, Pin 9, High Active)

Zur Platine 2

Das Maximum der Netzspannung (10V peak bei 230V effektiv) als Gleichspannung; 3 Pin Stecker (Pin 1 – GND, Pin 2 – Signal, Pin 3 – GND).

Auf der Platine

Messabgriffe

Zwischen den Spindeltrimmern, von Oben nach Unten:

Pin 1: +10,6V Referenzspannung (A)
Pin 2: +9V Referenzspannung (B)
Pin 3: -1V Referenzspannung (C)
Pin 4: +1V Referenzspannung (D)
Pin 5: Eingangsspannung vom Netzteil (9V~ eff) (E)
Pin 6: Ausgang nach dem Allpass (F)
Pin 7: Ausgang Überspannung (wie Bus Pin9) (G)
Pin 8: Ausgang Stromausfall (wie Bus Pin 10) (H)
Pin 9: Ausgang des Präzisionsgleichrichters (I)
Pin 10:Ausgang des Eingangsverstärkers (J)

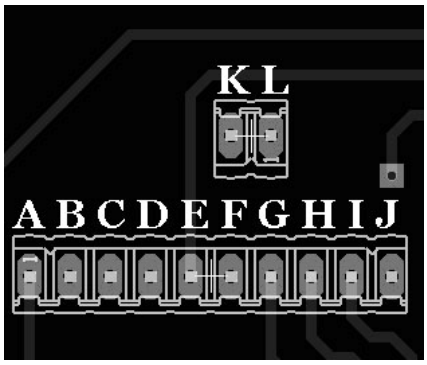


Abbildung 55: Messabgriffe

Vor dem 10 Pin Messstecker, von Oben nach Unten:

Pin 1: Das Maximum der Netzspannung (10V Peak bei 230V eff) als Gleichspannung (K)
Pin 2: GND (L)

Hinweise zur Inbetriebnahme

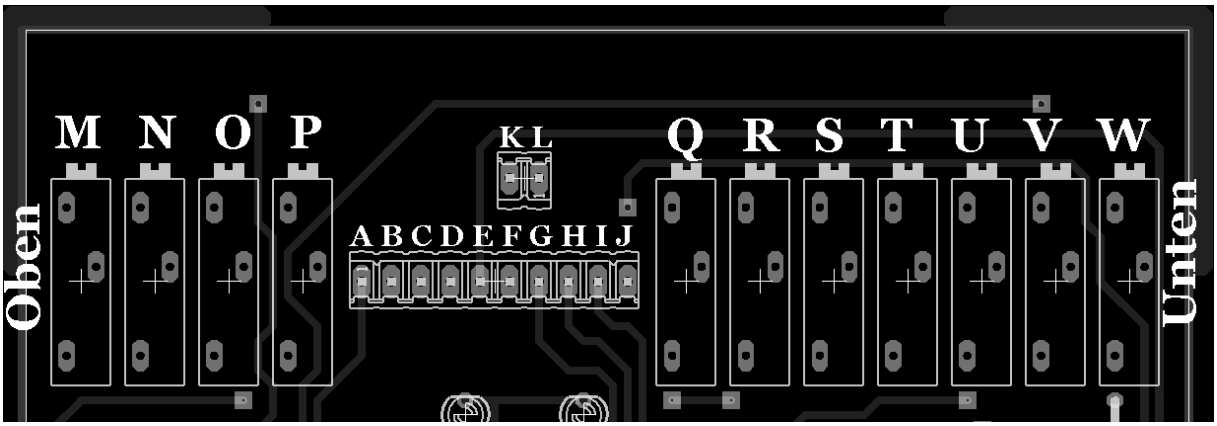


Abbildung 56: Einstellmöglichkeiten der Analogplatine

Spindeltrimmer von Oben nach unten:

Die ersten 4 vor dem Messabgriff:
+1V Referenzspannung (M)
–1V Referenzspannung (N)
+9V Referenzspannung (O)
+10,6V Referenzspannung (P)

Die 7 nach dem Messabgriff:
Eingangsverstärker: Verstärkungsverhältnis (Q)
Präzisionsgleichrichter: 20KΩ (R)
Präzisionsgleichrichter: 20KΩ (S)
Präzisionsgleichrichter: 10KΩ (T)
Ausgangsspannung des Überspannungssignals (U)
Ausgangsspannung des Stromausfallsignals (V)
Phase des Allpasses (W)

Stückliste

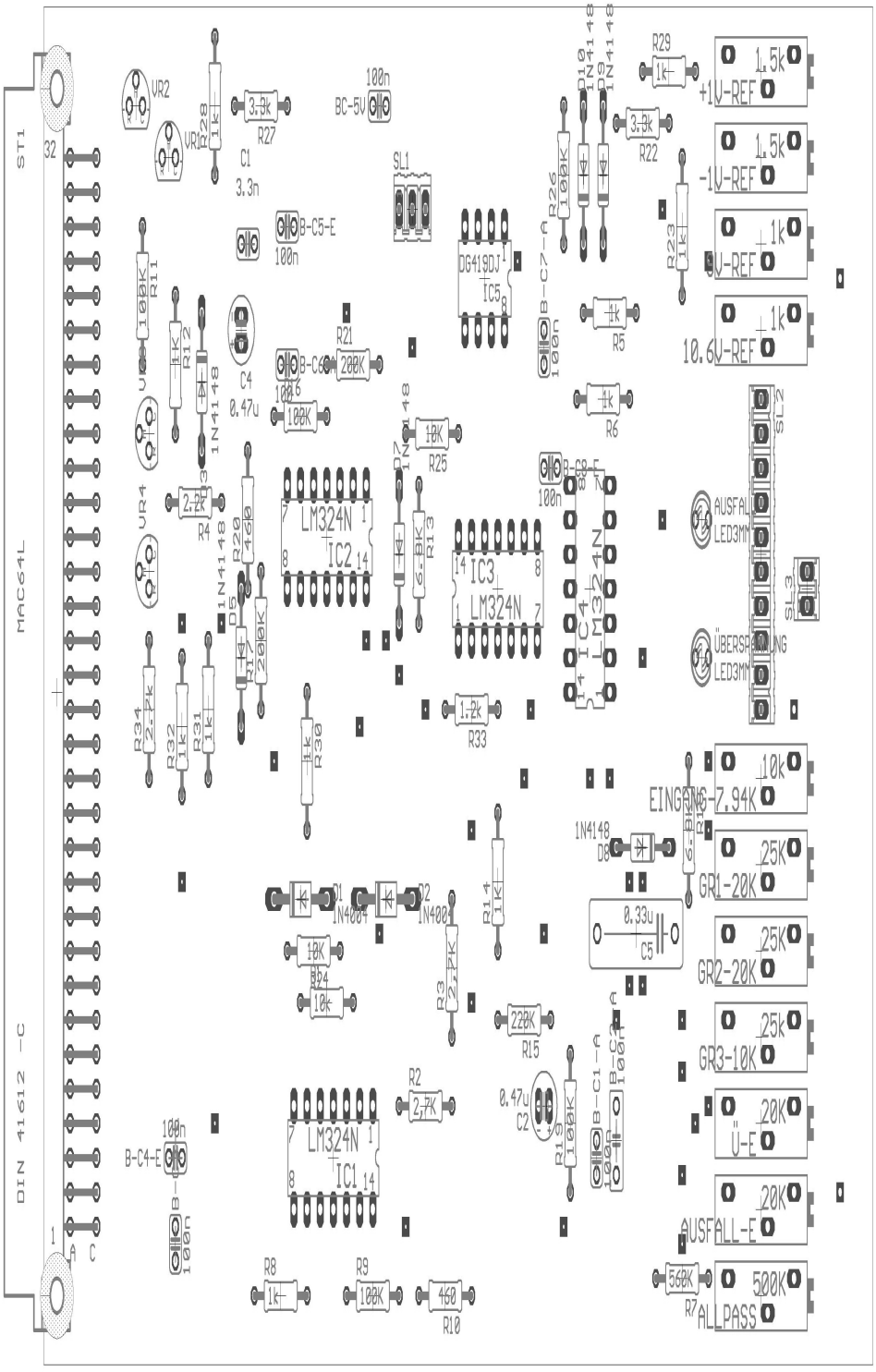


Abbildung 57: Bestückungsplan der Analogplatine

Kondensatoren (13 Stück):

- 9x 100nF Keramik
- 1x 3,3nF Keramik
- 2x 0,47µF Elko
- 1x 0,33µF Folie

Widerstände (34 Stück):

- 2x 460Ω
- 11x 1KΩ
- 1x 1,2KΩ
- 1x 2,2KΩ
- 3x 2,7KΩ
- 2x 3,3KΩ
- 2x 6,8KΩ
- 3x 10KΩ
- 5x100KΩ
- 3x 220KΩ
- 1x560KΩ

Dioden (16 Stück):

- 10x 1N4148 (oder vergleichbare)
- 2x LED Low Current 3mm oder 5mm
- 4x TL431CLP (einstellbare Zehnerdiode)

ICs (5 Stück):

- 4x LM324N (Quad OPV)
- 1x DG419DJ (Analogschalter)

Spindeltrimmer (11 Stück):

- 4x 2KΩ
- 5x 25KΩ (oder 3x25KΩ und 2x20KΩ)
- 1x 10KΩ
- 1x 500KΩ

Steckverbinder (4 Stück):

- 1x MAC64L (Busstecker 64Pol)
- 1x 10Pin Steckverbinder
- 1x 3Pin Steckverbinder
- 1x 2Pin Steckverbinder

Ätzlayouts

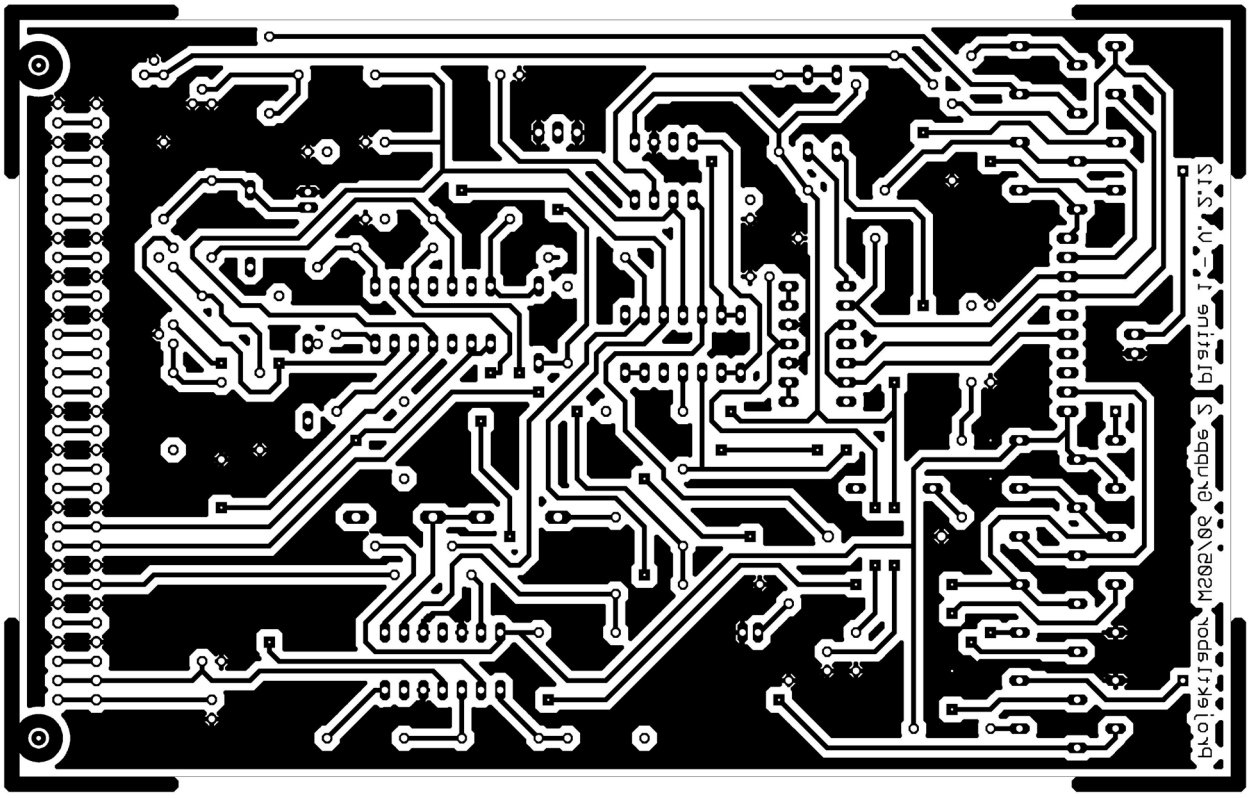


Abbildung 58: Bottomlayer der Analogplatine

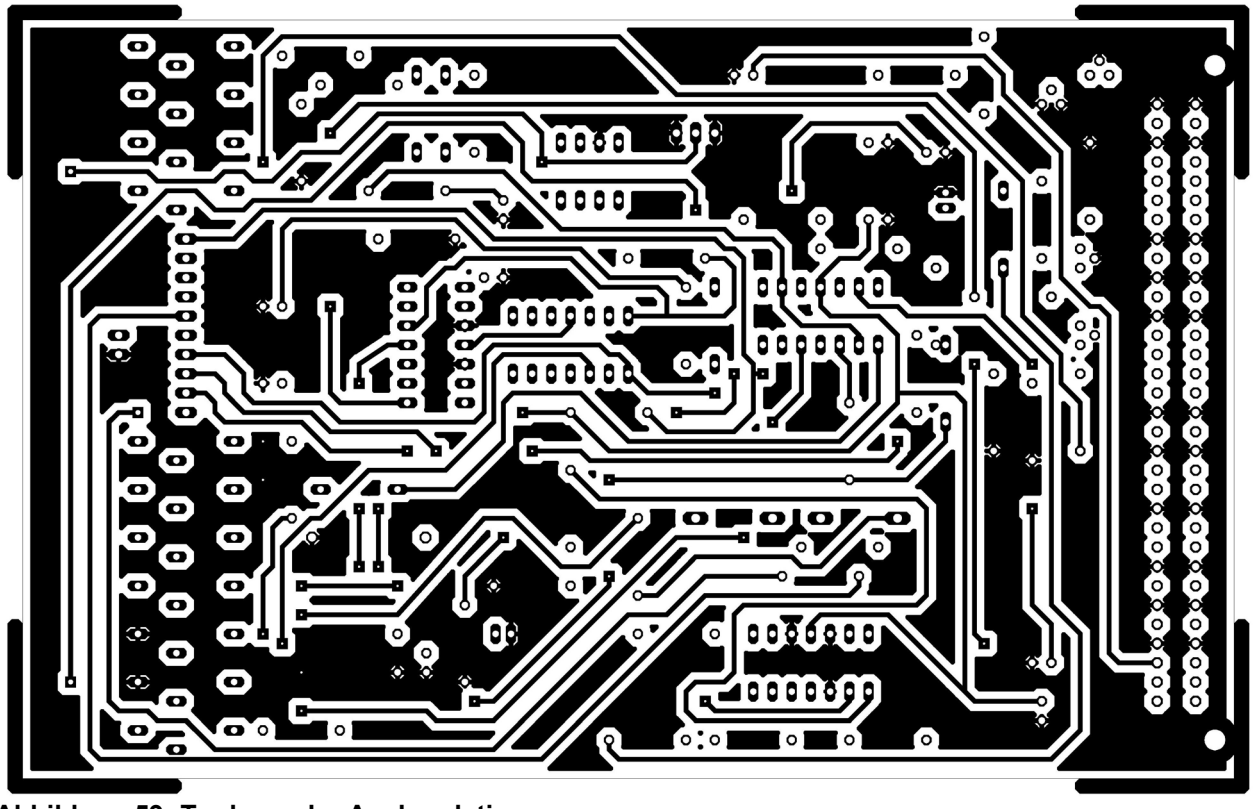


Abbildung 59: Toplayer der Analogplatine

3.5. Digitalteilplatine

Schnittstellen

SL4: 1 GND
2 Signal von Analogplatine
3 GND

SL3: 1 Referenzspannung
2 Vergleichsspannung für den Minimum-Taktblocker
3 Eingangsteiler-Ausgang
4 GND

SL6: 1 +5V
2 GND
3 -15V
4 +15V

SL5: 1 Messausgang Minimum (analog)
2 GND
3 Messausgang Maximum (analog)

SV1: Digitalausgang Maximum (10Bit parallel)

SV2: Digitalausgang Minimum (10Bit parallel)

2.7) Hinweise zur Inbetriebnahme:

Die Referenzspannungen sind wie folgt einzustellen:

SL3 Pin 1: -10V, einstellbar über R10
SL3 Pin 2: +1V, einstellbar über R16

Mit Hilfe von R8 kann die Eingangsspannung zwischen 50% und 100% eingestellt werden (siehe Eingangsteiler).

Stückliste

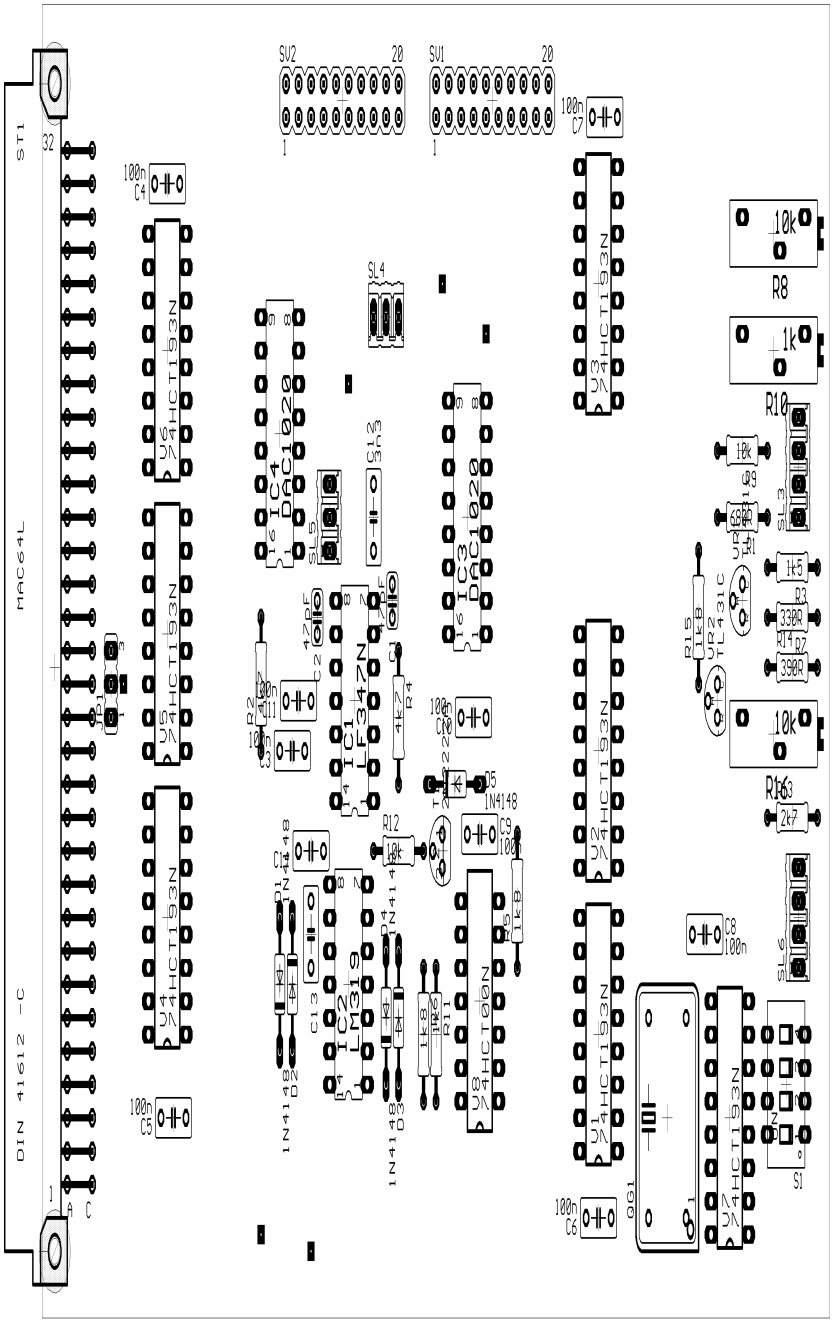


Abbildung 60: Bestückungsplan

Kondensatoren:

- 2x 47pF
- 9x 100nF
- 1x 3.3nF

Dioden:

- 5x 1N4148
- 2x TL431C

ICs:
- 1x LF347N
- 1x LM319
- 2x DAC1020

Quartz:
- 1x 1MHz Quartz

Widerstände:
- 1x 680Ohm
- 2x 4.7kOhm
- 1x 1.5kOhm
- 3x 1.8kOhm
- 1x 330Ohm
- 2x 10kOhm
- 1x 1kOhm
- 1x 2.7kOhm
- 1x 390Ohm
- 2x 10kOhm (Spindel)
- 1x 1kOhm (Spindel)

Transistor:
- 1x 2N2222A

Zähler:
- 7x 74HCT193N

NAND-Gatter:
- 1x 74HCT00N

Ätzlayouts

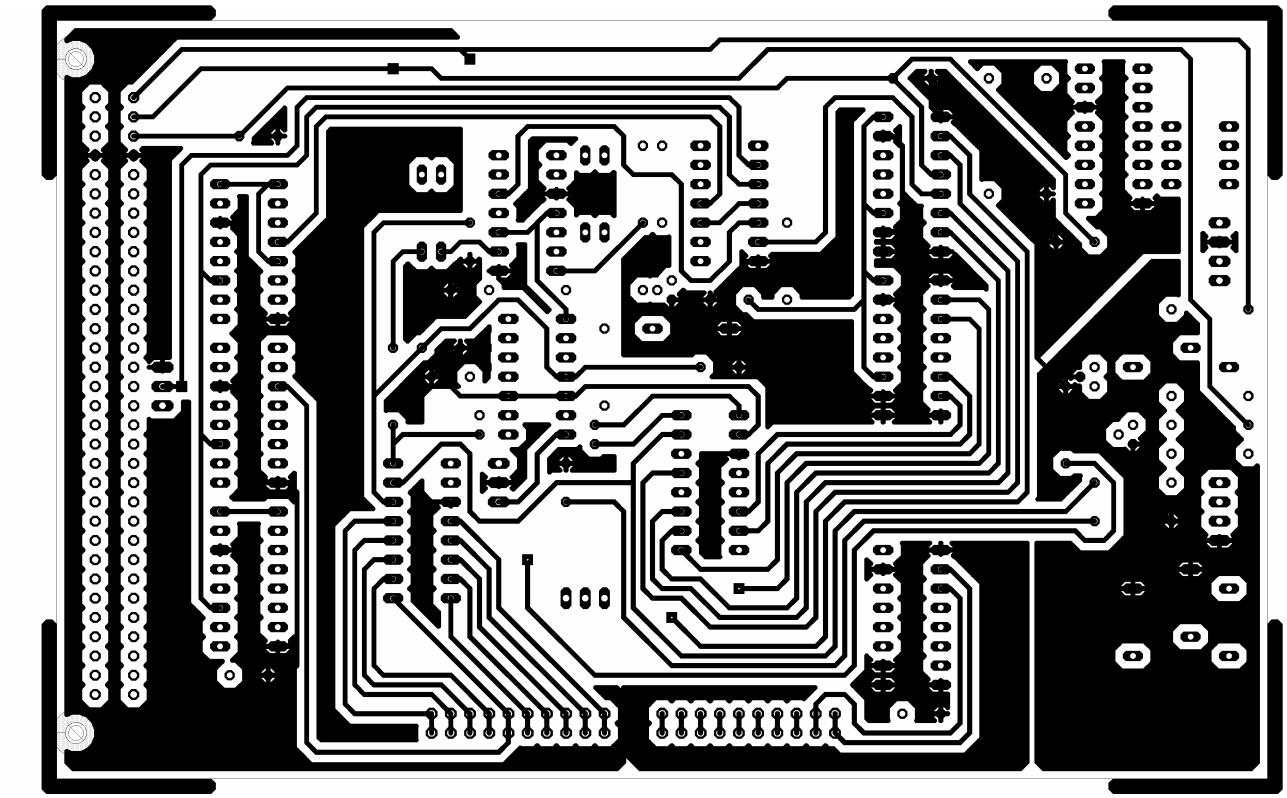


Abbildung 61: Toplayer der Digitalteilplatine

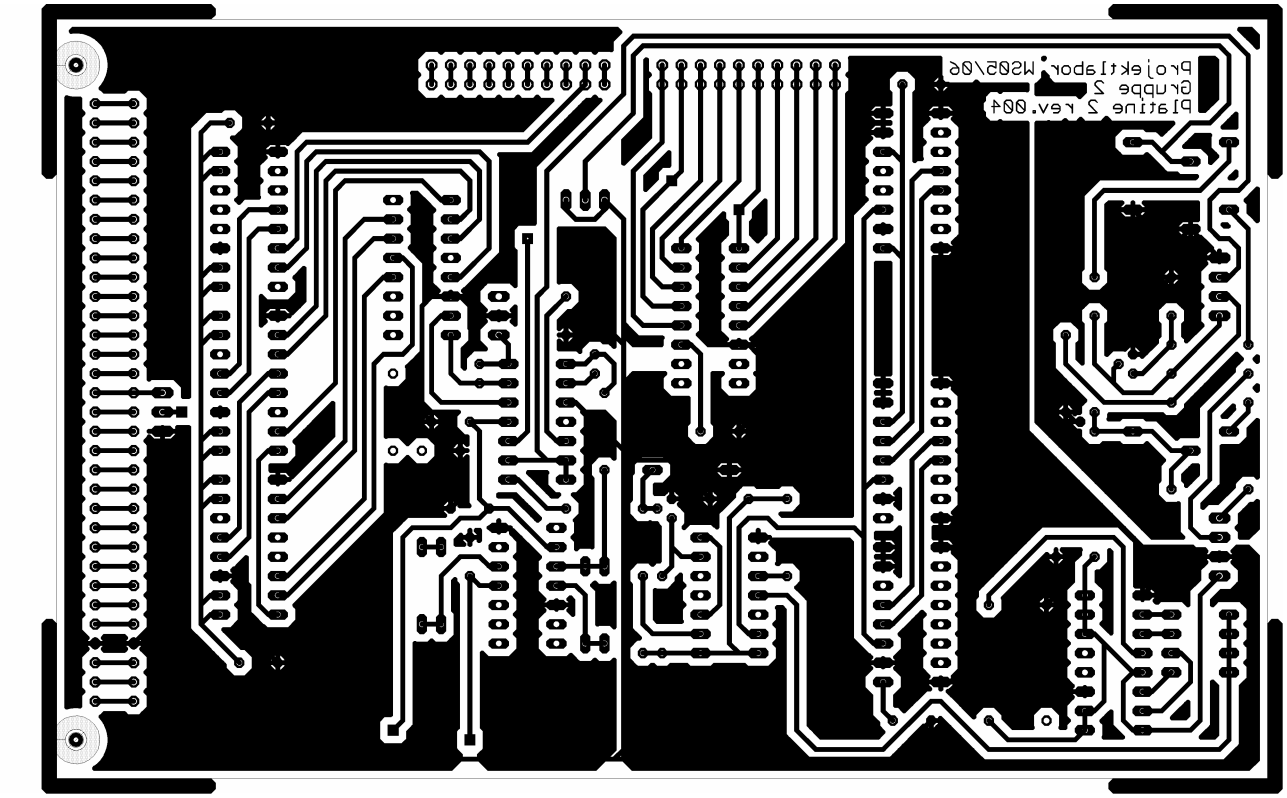


Abbildung 62: Toplayer der Digitalteilplatine

3.6. Umschaltplatine

Stückliste

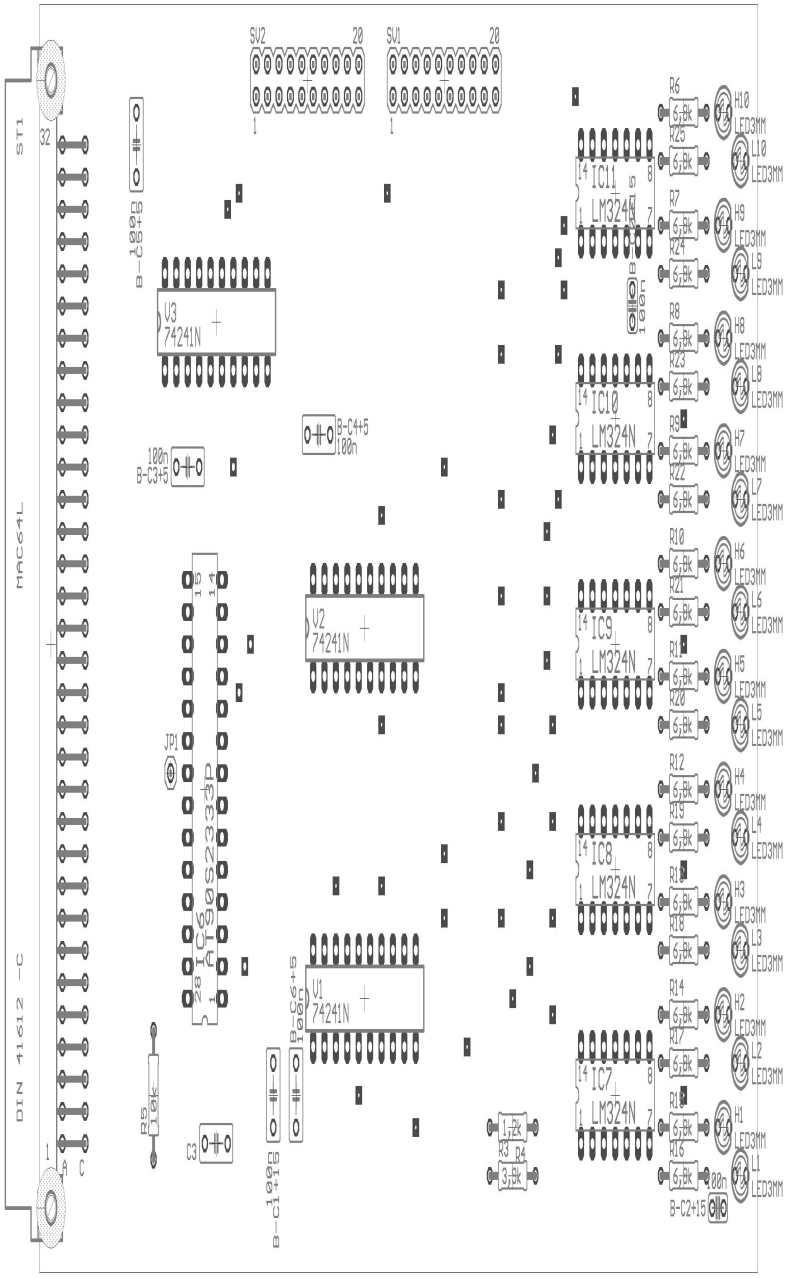


Abbildung 63: Bestückungsplan der Umschaltplatine

LED:

10x Gelbe 3mm LEDs
10x Grüne 3mm LEDs

ICs:

3x 74241N

1x AT90S2333P

Kondensatoren:
7x 100nF

Widerstände
20x 6,8kOhm
1x 1,2kOhm
1x 3,9kOhm
1x 10kOhm

Ätzlayout

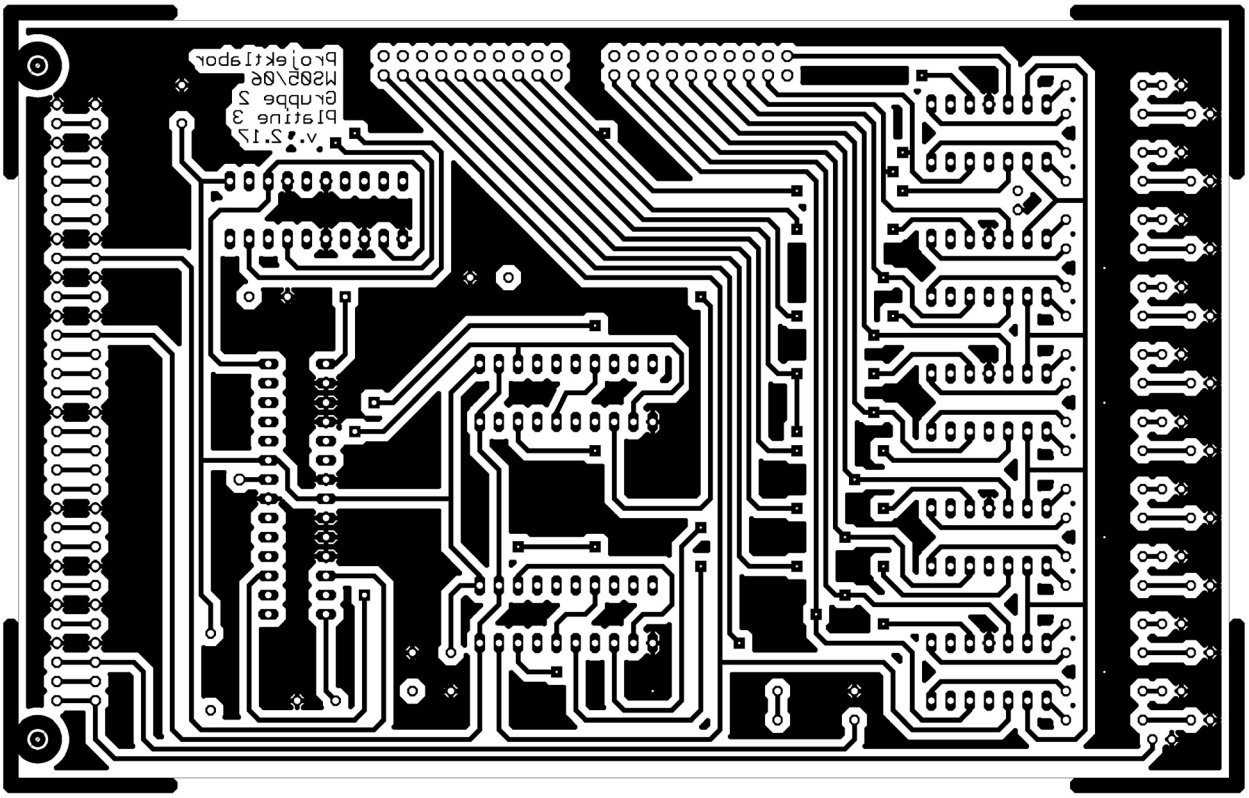


Abbildung 64: Bottomlayer der Umschaltplatine

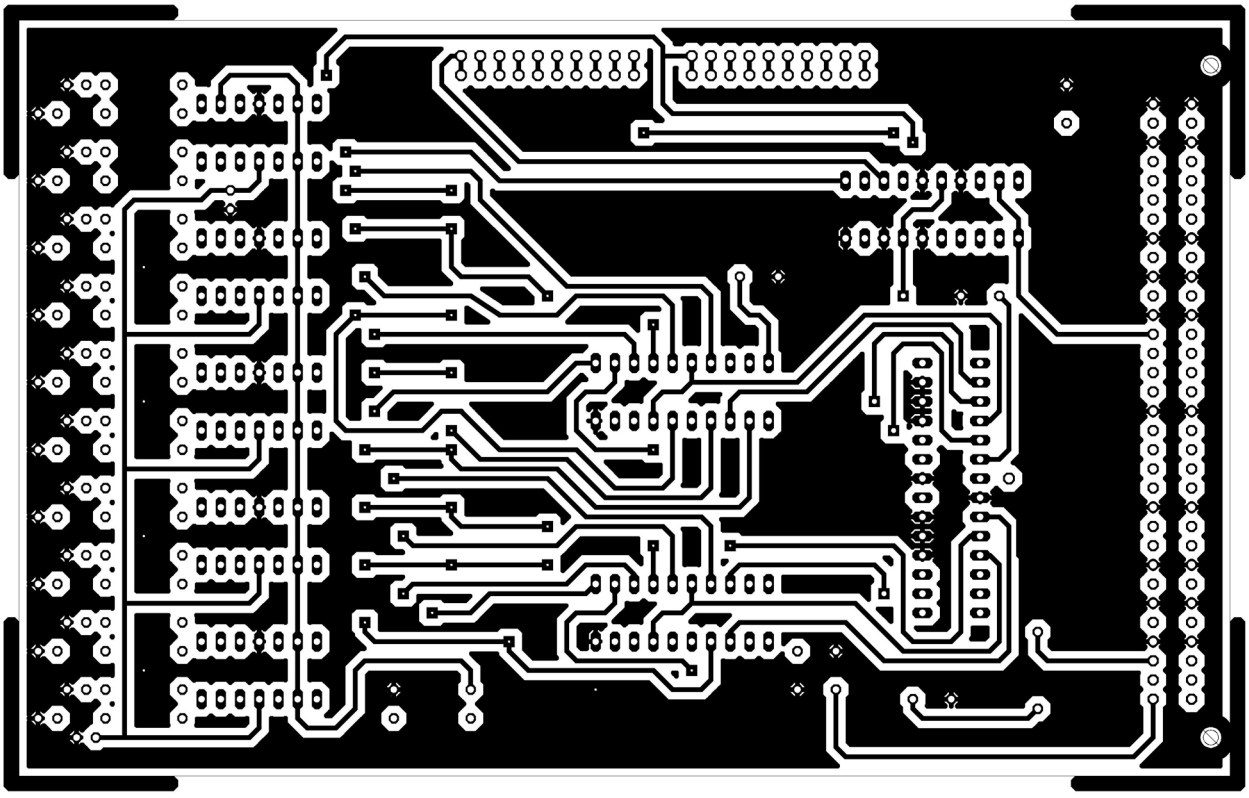


Abbildung 65: Toplayer der Umschaltplatine

3.7. Grundsätzliche Probleme

Probleme gab es vorwiegend durch Einstreuungen von außen, welche sich schon sehr früh bemerkbar machten. Bei den OPVs führte das zu unvorhergesehenen Folgen. Besonders starkes Aufschwingen konnte immer wieder von uns Festgestellt werden, selbiges konnte erfolgreich durch 100nF Blockkondensatoren an den Eingangsspannungen behoben werden, diese wurden an allen wichtigen Spannungen und vor wichtigen Bauteilen angebracht. Dies zeigte den erwünschten Erfolg.

4. Gruppe 3

4.1. Aufgabe der Gruppe

- **Beschreibung der Aufgabenstellung an die Gruppe „Frequenzmessung“**

Um ein zügiges und gewissenhaftes Arbeiten zu ermöglichen, war es notwendig, die neuralgischen Punkte unseres Schaltungsteils herauszuarbeiten.

- **Messgenauigkeit**

Da erwartungsgemäß nur geringe Abweichungen von der Grundfrequenz 50 Hz im deutschen Stromnetz vorkommen, ist es notwendig, eine Frequenzmessung mit einer sehr hohen Genauigkeit durchzuführen. Die Abweichungen bewegen sich laut Bewag (jetzt Vattenfall – sonst ändert sich nichts) im 10 mHz- Bereich.

- **Filterung des Messsignals**

Der Spannungsverlauf des deutschen Netzes ähnelt nur annähernd einem Sinus, somit ist es notwendig, das Messsignal ausreichend zu filtern. Da erwartungsgemäß nur Frequenzanteile oberhalb der 50 Hz das Signal verunreinigen werden, ist der Einsatz eines Tiefpasses mit sehr hoher Flankensteilheit notwendig, um den Rauschanteil, welcher anschließende Rechenschaltungen stören könnte, auf ein Minimum zu reduzieren.

- **Umsetzung des Sinussignals in ein Rechtecksignal**

Die anschließende Verarbeitung des Messsignals soll über einen Mikrocontroller in digitaler Form umgesetzt werden. Somit ist eine Umsetzung des Sinussignals in ein sehr genaues Rechtecksignal notwendig. Dabei sind in jedem Fall die Abstände der Nulldurchgänge einer Periode getreu zu halten.

- **Versendung des Messwertes als serielles Datenpaket**

Da der einsetzte Mikrocontroller ein Weitersenden der Daten in serieller Form ermöglicht, ist dies eine Datenleitung sparende und Fehlerquellen ausschließende Übertragung.

- **Optimierung der einzelnen Baugruppen auf hohe Genauigkeit**

Da wie in 3.a) erwähnt die Abweichungen von der Grundfrequenz im +/-10 mHz-Bereich liegen, muss jede Baugruppe auf höchste Genauigkeit überprüft werden.

- **Einteilung in Baugruppen**

Aus der in Abschnitt 3 aufgestellten Aufgabenstellung ergab sich nach Diskussion verschiedener Schaltungsvorschläge folgende Einteilung in Baugruppen, welche einzeln oder in Kleingruppen als Testaufbau vorerst realisiert werden sollten:

Aktiver Tiefpass-Filter 8. Ordnung zur Säuberung des Messsignals
Schmitt-Trigger, Spannungsbegrenzung und Digitalisierung
Frequenzteiler zur Mittlung über 50 Perioden
hochfrequenter Quarzoszillator als Takt für Mikrocontroller sowie Aufbau eines Quarzofens für eine Temperaturstabilisierung
Mikrocontroller zur Messung der ansteigenden Flanke, Division der Periodendauer sowie der seriellen Kodierung

Im Weiteren werden die Baugruppen im Einzelnen beschrieben. Dabei werden Entwicklungsverlauf, Berechnungen, Schaltpläne, Layouts und Gültigkeitsbereiche detailliert beschrieben.

4.2. Filterung des Messsignals

Aufgabe

Aus dem von dem Trenntrafo ausgelieferten verrauschten Signal von 12 V, sollten wir ein gefiltertes Signal an dem Schmitt-Trigger übergeben. Die Überlegung war das geeignete Filter zu realisieren, da erwartungsgemäß nur Frequenzen oberhalb der 50 Hz das Signal verunreinigen werden, daher ist der Tiefpass das geeignete Filter, um das Störsignal (Rauschen) auf einen geringen Anteil zu reduzieren.

Definition

Ein Tiefpass lässt tiefe Frequenzen durch und unterdrückt hohe Frequenzen.

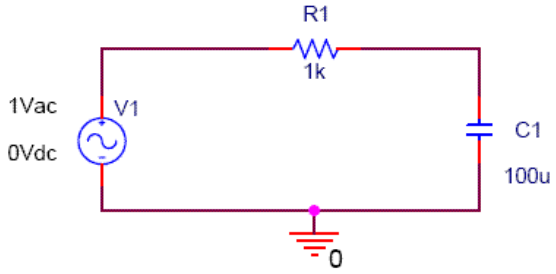


Abbildung 66. Passiver Tiefpass 1. Ordnung

Reflexion über die Lösung der Aufgabe

Probleme

Erste Schaltung

Als erster Schritt, haben wir uns die Art von Tiefpass überlegt, die wir brauchten. Da das Eingangssignal durch einen Tiefpass mit hoher Flankensteilheit gefiltert werden sollte, um ein Säuberes Ausgangssignal zu bekommen.
Mit einem passiven Tiefpass mit mehreren Stufen, hätten wir Nährungsweise dasselbe Ergebnis wie bei einem Aktiven Tiefpass erhalten. Wir brauchten einen Tiefpass, der nicht nur die Flanke am Ausgang realisieren konnte, sondern auch die Spannung nach jeder Stufe verstärken konnte, und die Auswahl eines aktiven Tiefpasses war schon ein positiver Punkt in unserer Überlegung.

So ergab sich folgender erster Entwurf der Schaltung:

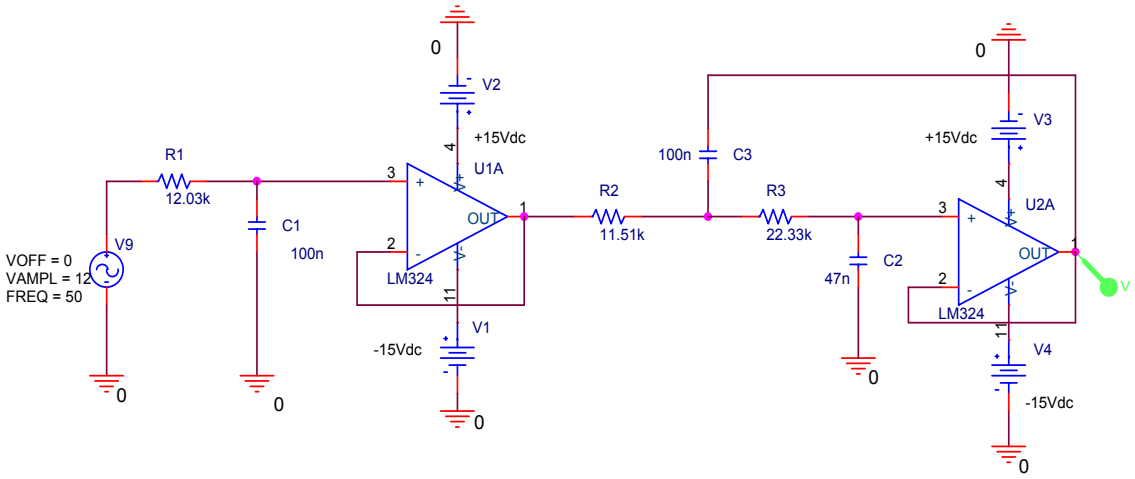


Abbildung 67: Bessel-Tiefpass 3.Ordnung

Die Gleichspannungsverstärkung des Gesamtfilters soll den Wert eins besitzen. Um das zu erreichen, muss auch der Impedanzwandler in der Filterstufe 1.Ordnung die Verstärkung $\alpha = 1$ erhalten.

Diese lieferte folgenden Spannungsverlauf:

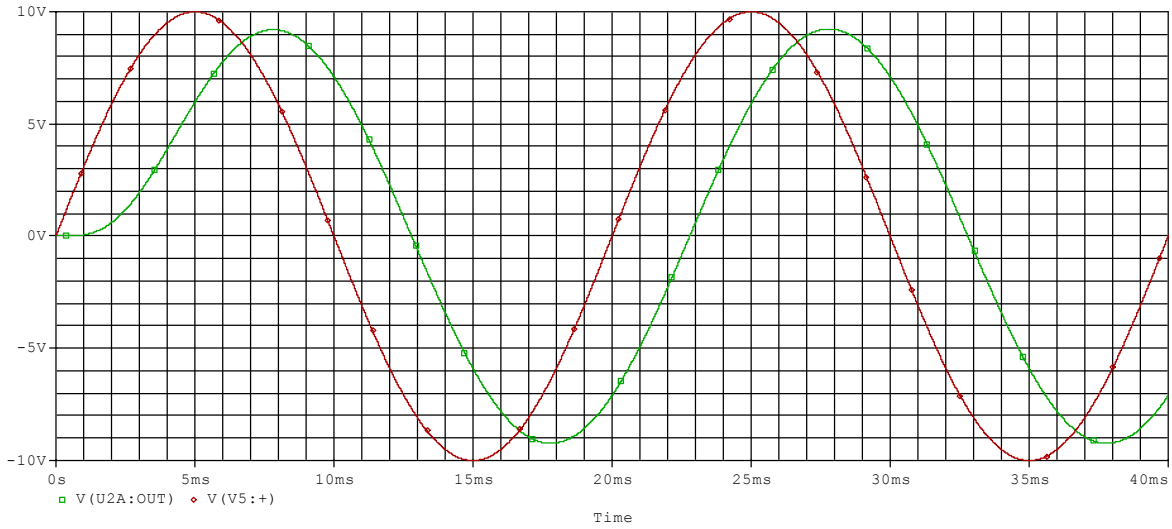


Abbildung 68: Spannungsverlauf des Bessel-Tiefpasses 3.Ordnung

Und der Frequenzgang sieht so aus:

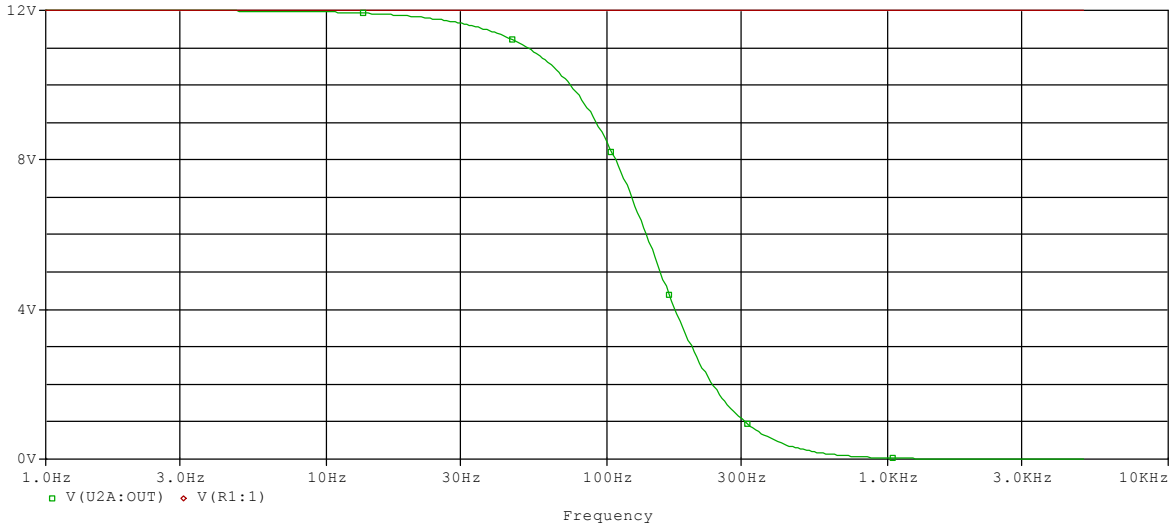


Abbildung 69: Frequenzgang des Bessel-Tiefpass 3.Ordnung

Also besitzt die Schaltung ein Verstärkungsfaktor kleiner als 1. Um unser Messsignal möglichst stabil zu halten, also den Signal-Rausch-Abstand möglichst groß zu halten, haben wir diese Schaltung verworfen und unsere Überlegungen mit Berücksichtigung von unseren Vorstellungen, neu orientiert
Deswegen mussten wir uns auf zwei wichtige Punkte konzentrieren.

Vermeidung einer erheblichen Senkung der Spannung und Belastung des Eingangssignals.
Eine hohe Flankensteilheit nach der Grenzfrequenz.

Aufgrund dieser Festlegung entstand der zweite Entwurf der Schaltung.

Zweite Schaltung

Der Bedarf an höherer Flankensteiheit hat uns auf den Bessel –Tiefpass 6.Ordnung geführt, in dem man nach dem Tiefpass 3.Ordnung eine Filterstufe aufgebaut hat. Der zweite Entwurf hat die obigen Kriterien erfüllt, deshalb hatten wir diesen aufgebaut, um einige Testläufe und Messungen durchzuführen.

Und so sah der Tiefpass aus:

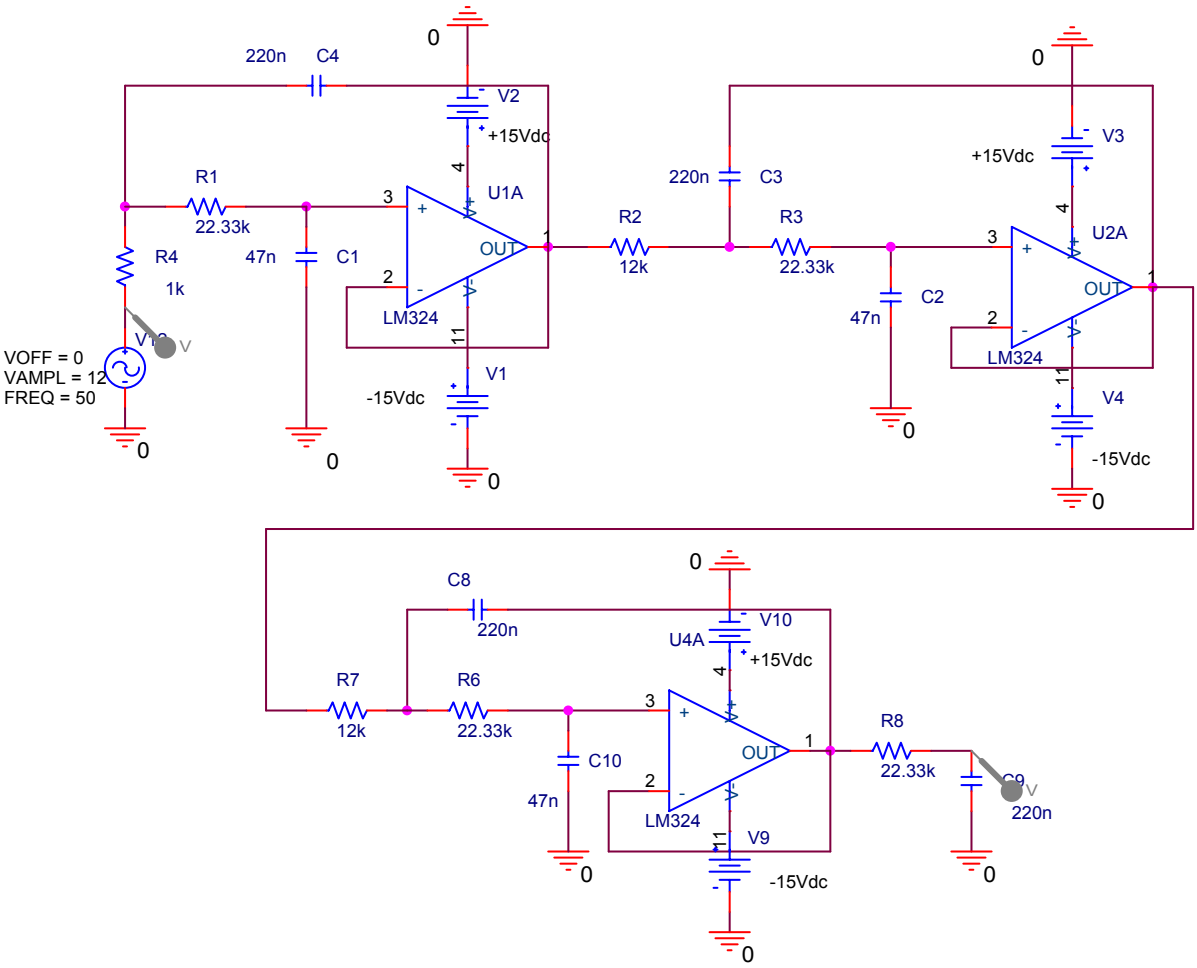


Abbildung 70:Tiefpass 6.Ordnung

Ein Bessel-Tiefpass 6-Ordnung stellte sich schon als ein geeignetes Filter heraus um unser Messsignal in ein möglichst Rauschenfreies Signal umzuwandeln.
Der Bessel-Tiefpass hat die Charakteristik, dass das Überschwingen oberhalb der 4.Ordnung wieder abnimmt. Nun haben wir den Frequenzgang simuliert welcher folgende Verlauf aufweist:

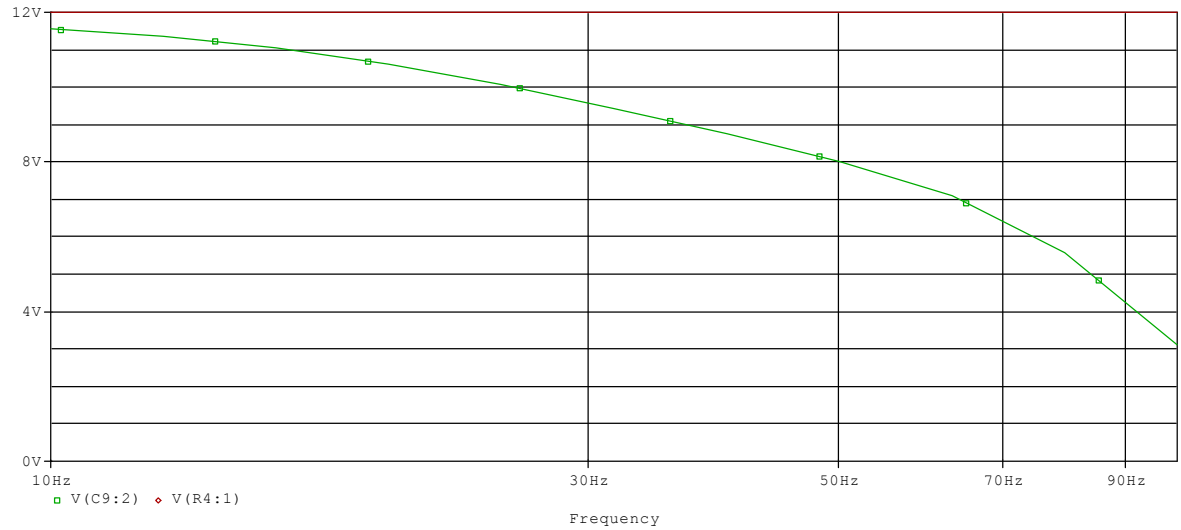


Abbildung 71: Simulationsergebnis des Tiefpasses 6.ordnung

Auf dem ersten Blick bemerkt man die Absenkung der Amplitude, nämlich von 12V bei 10 Hz auf 8,2 V bei 50 Hz.
Es entspricht den folgenden Spannungsverlauf:

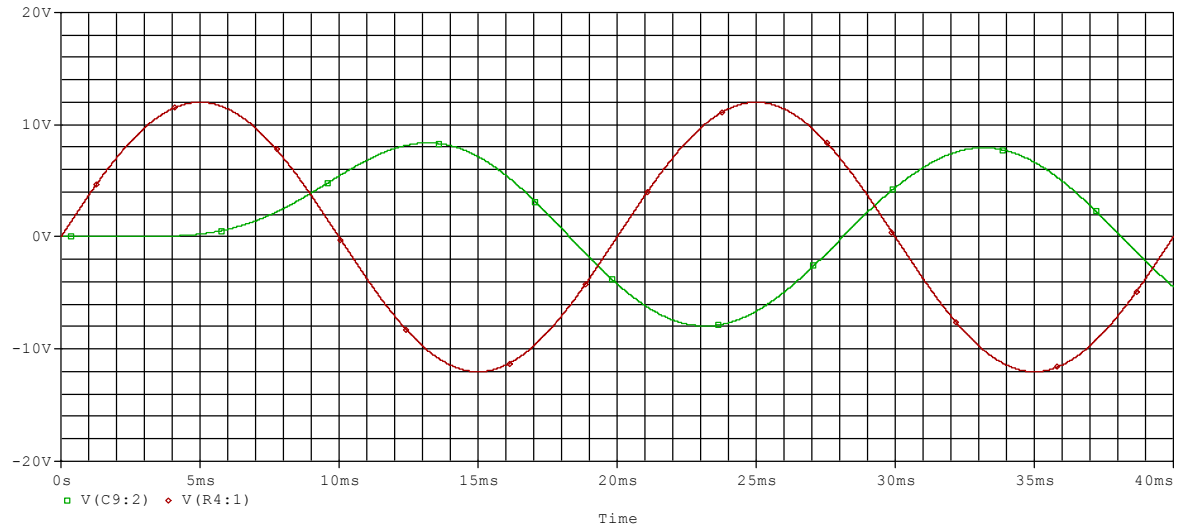


Abbildung 72: Spannungsverlauf des Tiefpasses 6.Ordnung

Das Besselfilter filtert alles, deshalb muss der Verlauf derart beginnen. Somit musste die Knickfrequenz in den Bereich von 55 – 60 Hz verschoben werden um die Amplitude unseres Signals möglichst nicht zu beeinflussen.
Wir haben nun, um die Flankensteilheit zu erhöhen, eine weitere Filterstufe integriert.

Dritter Entwurf:

Nachdem wir weiter eine Stufe hinter dem vorgehenden Filter siehe Abb.67)realisiert haben, ist der Tiefpass 6.Ordnung Tiefpass 8.Ordnung geworden.

Der Entwurf ist in Abbildung 70 dargestellt:

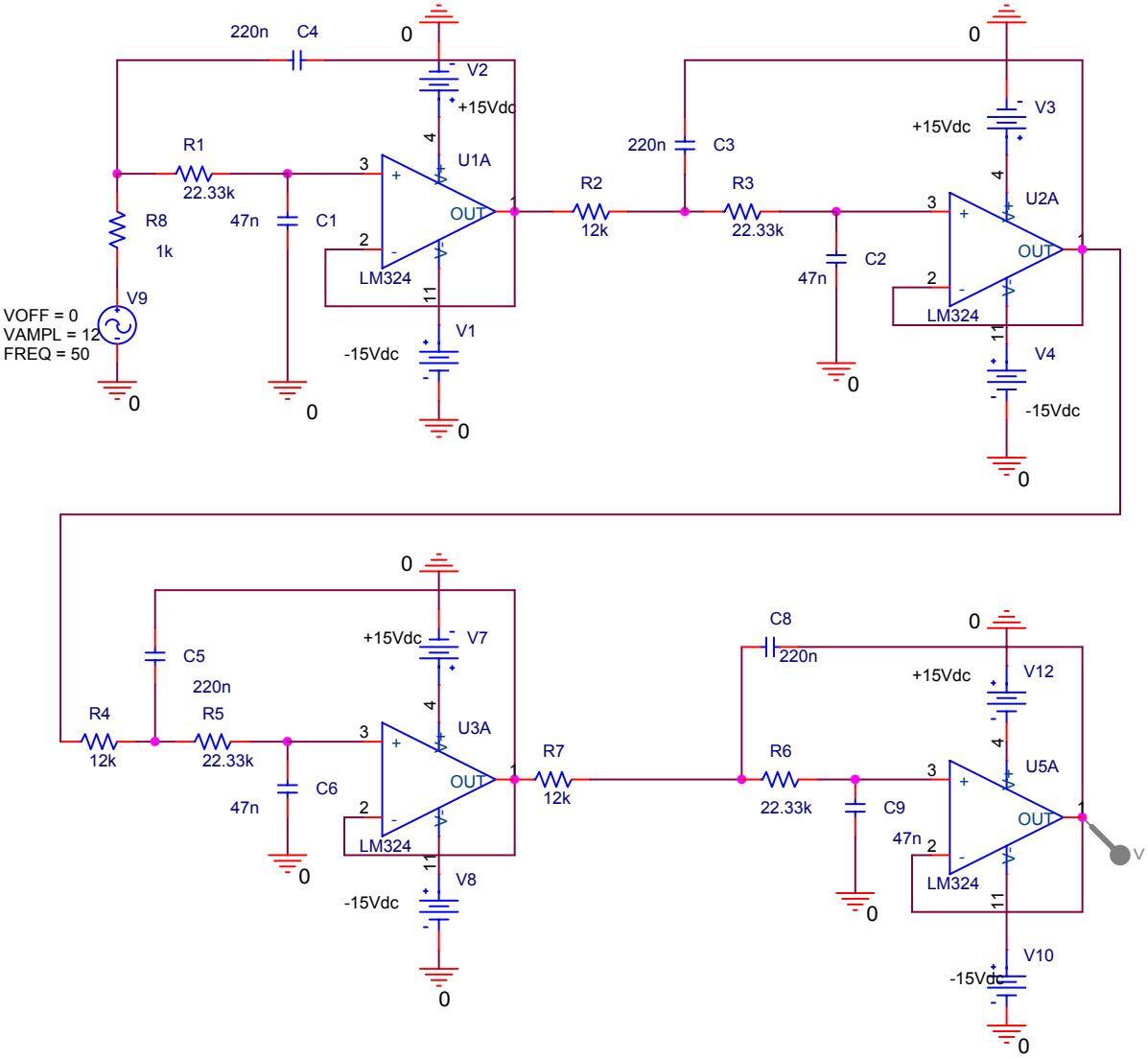


Abbildung 73: Tiefpass 8. Ordnung

Dies führte zu einem sehr reinen Sinussignal, jedoch wurde ein Verstärkungsfaktor von 1,26-pro-Verstärkerstufe übersehen.
Um nicht in den Sättigungsbereich der Operationsverstärker zu gelangen. Somit konnte ein Messsignal hoher Reinheit gewährleistet werden.

Lösung des Problems

Endgültige Schaltung

Um das Problem zu beheben, haben wir vor dem Filter einen invertierenden Verstärker aufgebaut. Wir haben den OP als Verstärker verwendet und nutzen hier die Möglichkeit der Gegenkopplung um definierte Verstärkungen zu erhalten.

Durch die Widerstände R_3 und R_4 (siehe Abbildung 71) wird die Verstärkung bestimmt.

Die Verstärkung lautet: $V = \frac{R_4}{R_3}$

Und damit ist die Spannung am Ausgang: $-U_a = U_e \times \frac{R_4}{R_3}$

Das Vorzeichen ist aufgrund des Invertierenden Verstärkers negativ.

Dieser sieht so aus:

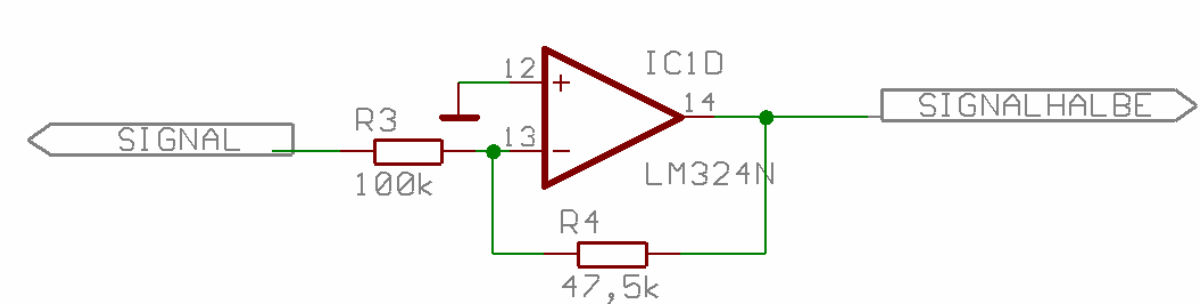


Abbildung 74: Invertierender Verstärker

So sieht letztlich unser Tiefpass aus:

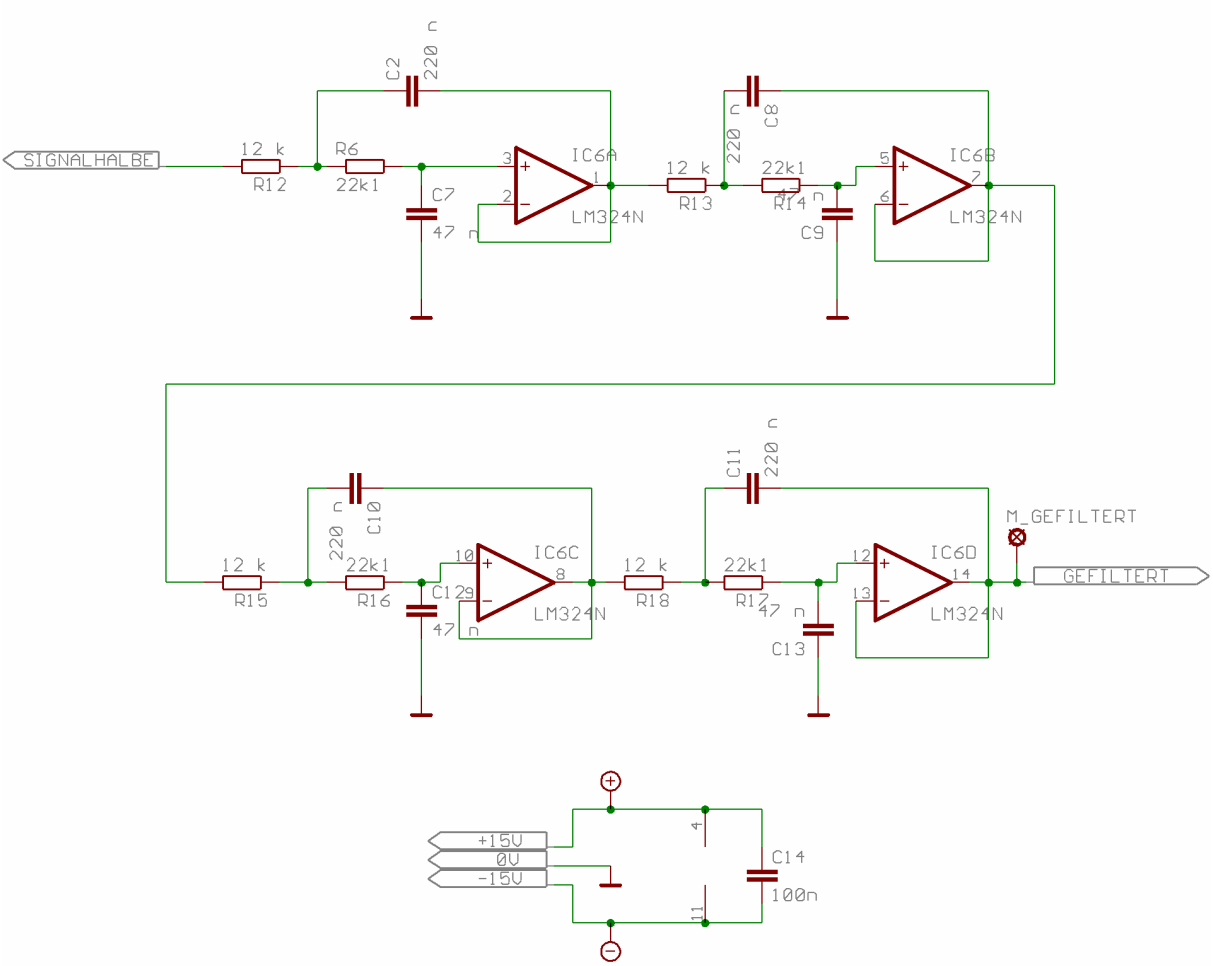


Abbildung 75: Tiefpass 8.Ordnung mit invertierender Vorverstärkung

Die Lage unseres Tiefpass auf der Platine(siehe die Zeichnung in orange)

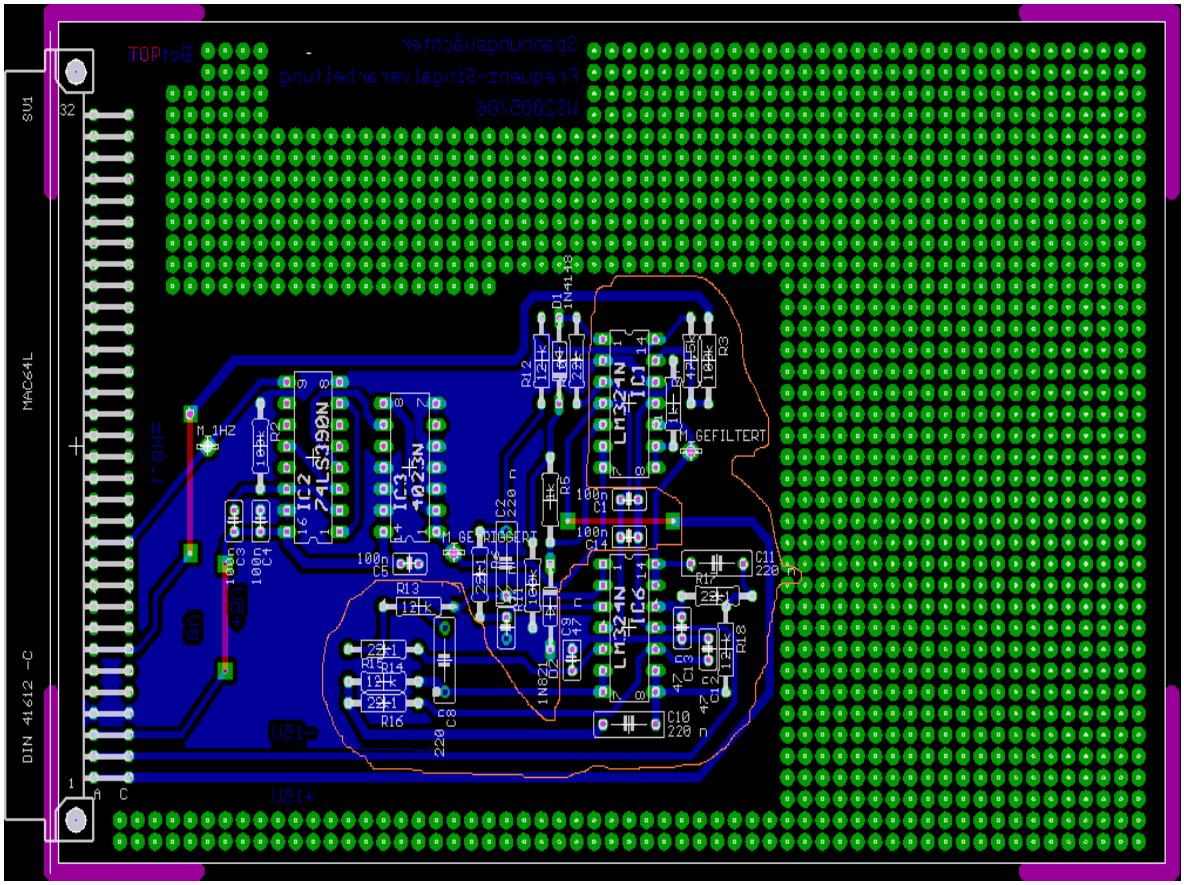


Abbildung 76: Stand des Tiefpasses auf der Platine

Bauelemente

Liste

Part	Value	Device	Package	Library	Sheet
C2	220 n	C-EU075-032X103.	C075-032X103	rcl	3
C7	47 n	C-EU025-025X050	C025-025X050	rcl	3
C8	220 n	C-EU075-032X103	C075-032X103	rcl	3
C9	47 n	C-EU025-025X050	C025-025X050	rcl	3
C10	220 n	C-EU075-032X103	C075-032X103	rcl	3
C11	220 n	C-EU075-032X103	C075-032X103	rcl	3
C12	47 n	C-EU025-025X050	C025-025X050	rcl	3
C13	47 n	C-EU025-025X050	C025-025X050	rcl	3
IC6	LM324N	LM324N	DIL14	linear	3
M_GEFILTERT	LSP13	LSP13	LSP13	solpad	3
R6	22k1	R-EU_0207/10	0207/10	rcl	3
R12	12 k	R-EU_0207/10	0207/10	rcl	3
R13	12 k	R-EU_0207/10	0207/10	rcl	3
R14	22k1	R-EU_0207/10	0207/10	rcl	3
R15	12 k	R-EU_0207/10	0207/10	rcl	3
R16	22k1	R-EU_0207/	10 0207/10	rcl	3
R17	22k1	R-EU_0207/10	0207/10	rcl	3
R18	12 k	R-EU_0207/10	0207/10	rcl	3

Tabelle 3

Pinbelegung und allgemeine Beschreibung des LM 324

DESCRIPTION

The LM124/SA534/LM2902 series consists of four independent, high-gain, internally frequency-compensated operational amplifiers designed specifically to operate from a single power supply over a wide range of voltages.

UNIQUE FEATURES

In the linear mode, the input common-mode voltage range includes ground and the output voltage can also swing to ground, even though operated from only a single power supply voltage.

The unity gain crossover frequency and the input bias current are temperature-compensated.

FEATURES

- Internally frequency-compensated for unity gain
- Large DC voltage gain: 100dB
- Wide bandwidth (unity gain): 1MHz (temperature-compensated)
- Wide power supply range Single supply: $3V_{DC}$ to $30V_{DC}$ or dual supplies: $\pm1.5V_{DC}$ to $\pm15V_{DC}$
- Very low supply current drain: essentially independent of supply voltage (1mW/op amp at $+5V_{DC}$)
- Low input biasing current: $45nA_{DC}$ (temperature-compensated)
- Low input offset voltage: $2mV_{DC}$ and offset current: $5nA_{DC}$
- Differential input voltage range equal to the power supply voltage
- Large output voltage: $0V_{DC}$ to V_{CC}-$1.5V_{DC}$ swing

PIN CONFIGURATION

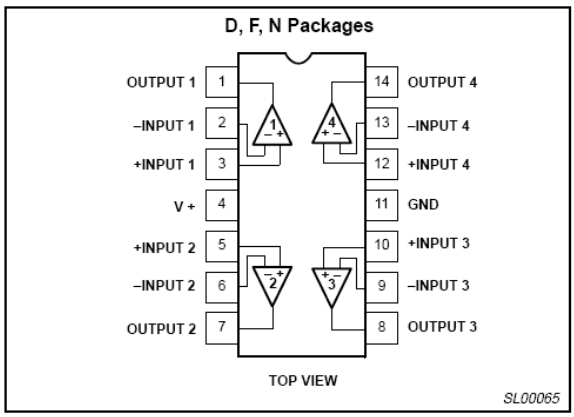


Figure 1. Pin Configuration

Abbildung 77: Pinbelegung und allgemeine Beschreibung des LM 324

Digitalisierung des Messsignals

Problemstellung

Um die Nulldurchgänge über unseren Mikrocontroller mit einem Taktsignal zu ermitteln, ist es notwendig den (gereinigten) Sinusspannungsverlauf in eine Rechteckspannung umzuwandeln die zwischen den logischen Pegeln LOW und HIGH alterniert.

Umsetzung

Hierfür wurde die Realisierung der Schaltung über einen nicht invertierenden Schmitt-Trigger und einer anschließenden Spannungsbegrenzung über eine Diodenschaltung vorgeschlagen.

Aufbau

Um einen nicht-invertierenden Schmitt-Trigger aufzubauen, legt man das Eingangssignal(Ue) auf den Fuß eines Mitkopplungs-Spannungsteilers (R1, R2) geben, und das ganze auf den nicht-invertierenden Eingang des OPV's anschliessen. Der invertierende Eingang wird auf Masse gelegt(Siehe Abb. 76).
R1: Eingangswiderstand.
R2: Widerstand der Mitkopplung.
Die Diode am Ausgang des OPV's leitet bei positiver Spannung, und sperrt bei Negativer. Dies ist eine Einweg-Gleichrichtung. Die Z-diode begrenzt anschließend die Spannung auf 4,9V.

Berechnung

Hat Ue einen großen positiven Wert, bleibt das Ausgangssignal maximal (Ua=Ua max). Verkleinert man Ue, bis die Differenzspannung Ud am Eingang des OPV's null ist, ändert sich Ua zunächst nicht. Erst wenn Ue den Wert Ue aus(Ausschaltpegel) erreicht oder unterschritten hat, springt Ua auf den minimalen Wert (Ua min). Bei diesem Zustand ist also
Ue = Ue aus = -(R1.Ua max)/R2
. Der neue Zustand ist stabil bis Ue der Wert
Ue ein = -(R1.Ua min)/R2 (Einschaltpegel)wieder erreicht oder überschreitet.

Dimensionierung

Für die Dimensionierung wurde für die Ein- und Ausschaltpegel Ue ein und Ue aus jeweils die Werte 0,5V und –0,5V festgelegt und am Ausgang wollten wir eine maximale Spannung von 10V haben. Über obige Formeln haben wir das Verhältnis von beiden Widerständen rausbekommen als:
R1/R2 = 1/20

Nach einer ersten Simulation in Pspice wurde ein Testaufbau realisiert und messtechnisch überprüft. Hierfür wurde folgende Schaltung verwendet:

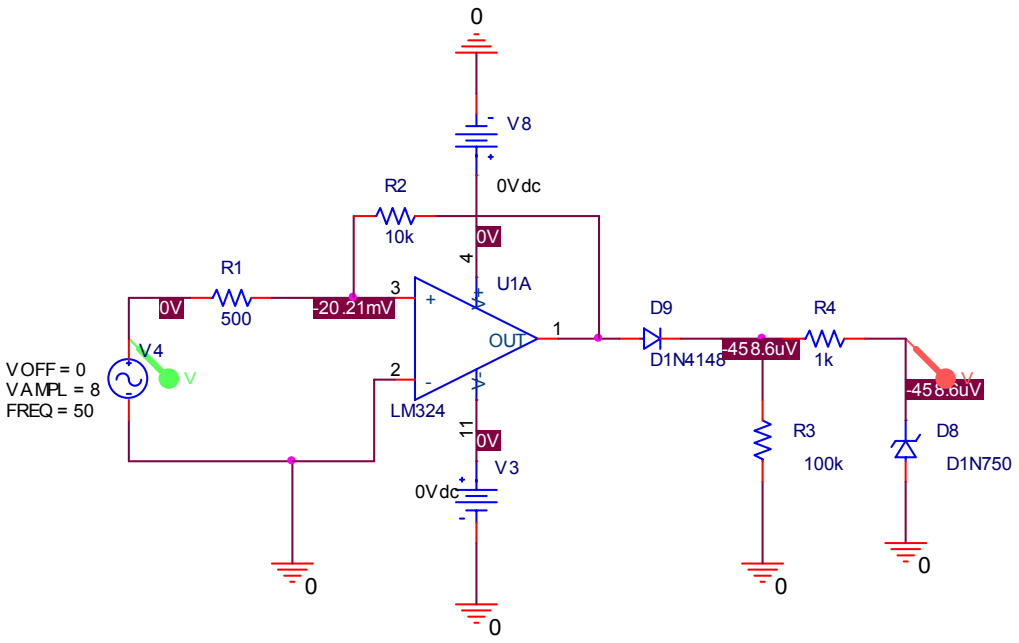


Abbildung 78: Schaltplan des Schmitt-Triggers

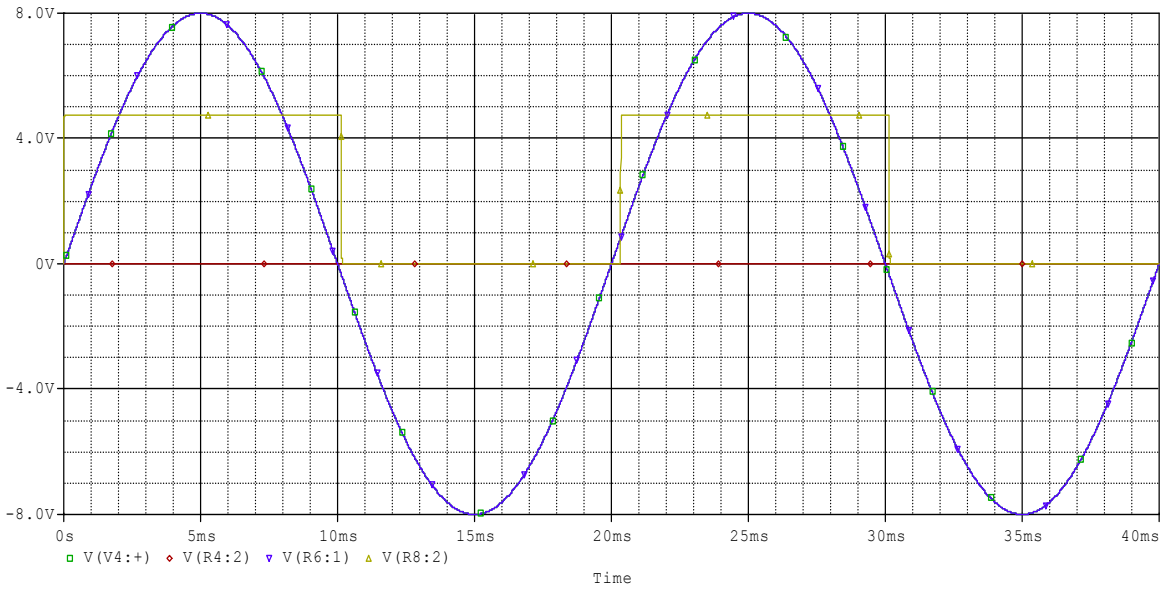


Abbildung 77: Simulationsergebnisse

Nach ersten Ergebnissen wurde erwogen eine Realisierung ohne Z-Diode zu ermöglichen, um eine Rauscheinstreuung vorzubeugen. Dafür wurden mehrere Operationsverstärker-Typen eingesetzt und mit einer Versorgungsspannung von +5V und 0V betrieben.

Dann entschied man für den ersten Aufbau und dessen Optimierung. Erst bei der Bestückung wurde festgestellt, dass die Dimensionierung der Vorwiderstände der Z-Dioden außer Acht gelassen wurde. Nach ebenso konnte auf einem hochauflösenden Oszilloskop festgestellt werden, dass sich aufgrund der Z-Dioden Spannungsspitzen im Rechteckformigen Verlauf des getriggerten Signals befanden. Diese konnten durch das Nachschalten eines NAND-Gatters beseitigt werden, wodurch nun ein reibungsloses Frequenzteil auf ein 1Hz-Signal nichts mehr im Weg stand.

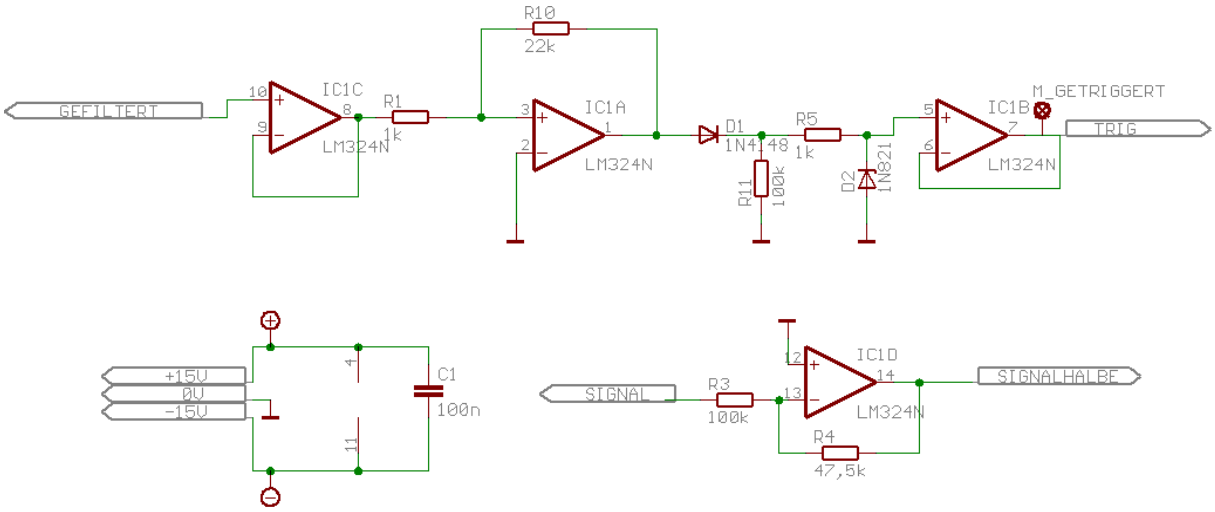


Abbildung 78a:Schaltplan des Schmitt-Triggers

4.3. Frequenzteiler:

Was ist ein Frequenzteiler?

Frequenzteiler sind Schaltungen, die die Frequenz eines Signals in einem bestimmten Verhältnis herunterteilen. Ein einfacher Dualzähler ist bereits ein einfacher Frequenzteiler. Man kann Frequenzteiler auch aus einzelnen Flip-Flops zusammenschalten.
Ein einzelnes Flip-Flop erzeugt eine Frequenzteilung im Verhältnis 2: 1. Mit zwei Flip-Flops kann ein Frequenzteiler für ein Verhältnis von 4: 1 aufgebaut werden. Die meisten Frequenzteiler haben ein festes Teilerverhältnis.

Aufbau und Funktionsweise

Wie schon erwähnt kann man einen Frequenzteiler aus einem einfachen Flip-Flop aufbauen, welches wiederum selber aus Nand-Gattern besteht. Die nachfolgende Abbildung 79 zeigt ein P-Spice Schematic eben dieser Schaltung.

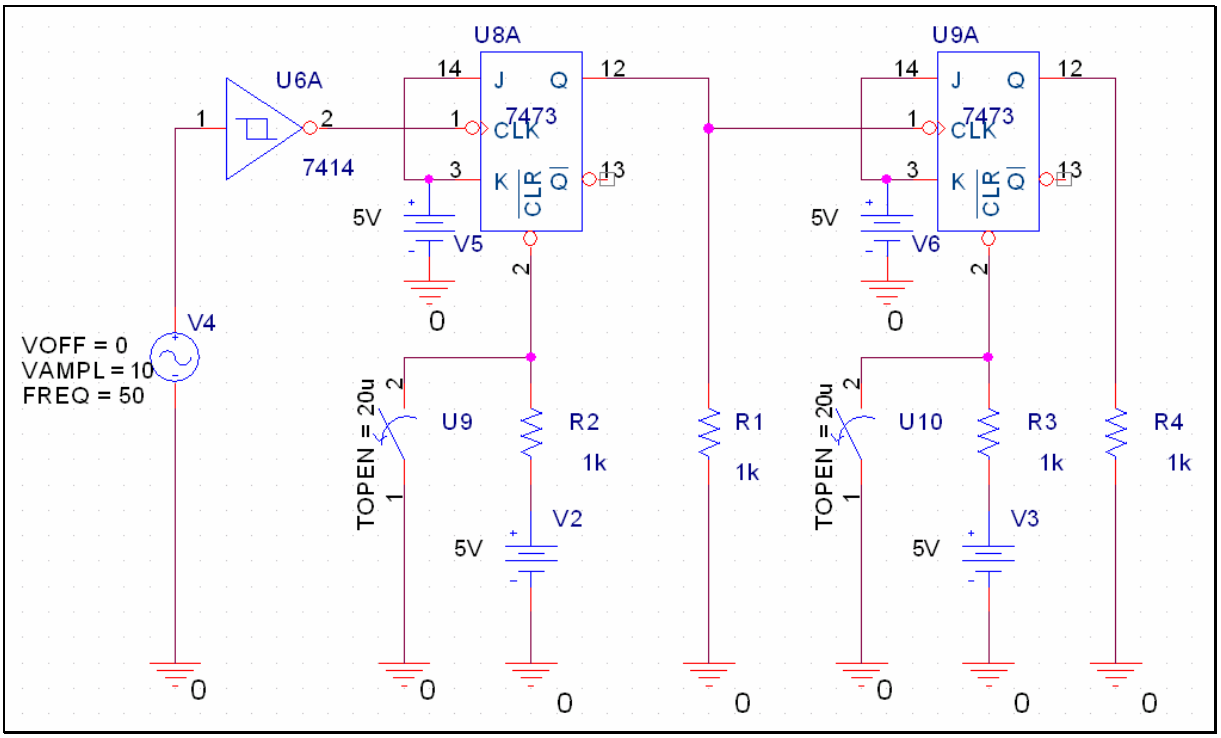


Abbildung 79: P-Spice Schematic des Frequenzteilers

Bei der Schaltung in Abbildung 79 wird zunächst ein 10V Signal mit einer Frequenz von 50Hz durch einen Schmitt-Trigger geleitet und dann auf den Clock-Eingang (CLK) des T-Flip-Flop. Abbildung 80 zeigt den zeitlichen Verlauf des Signals zum einen gemessen nach der Spannungsquelle (unten) und zum anderen nach dem Schmitt-Trigger (oben).

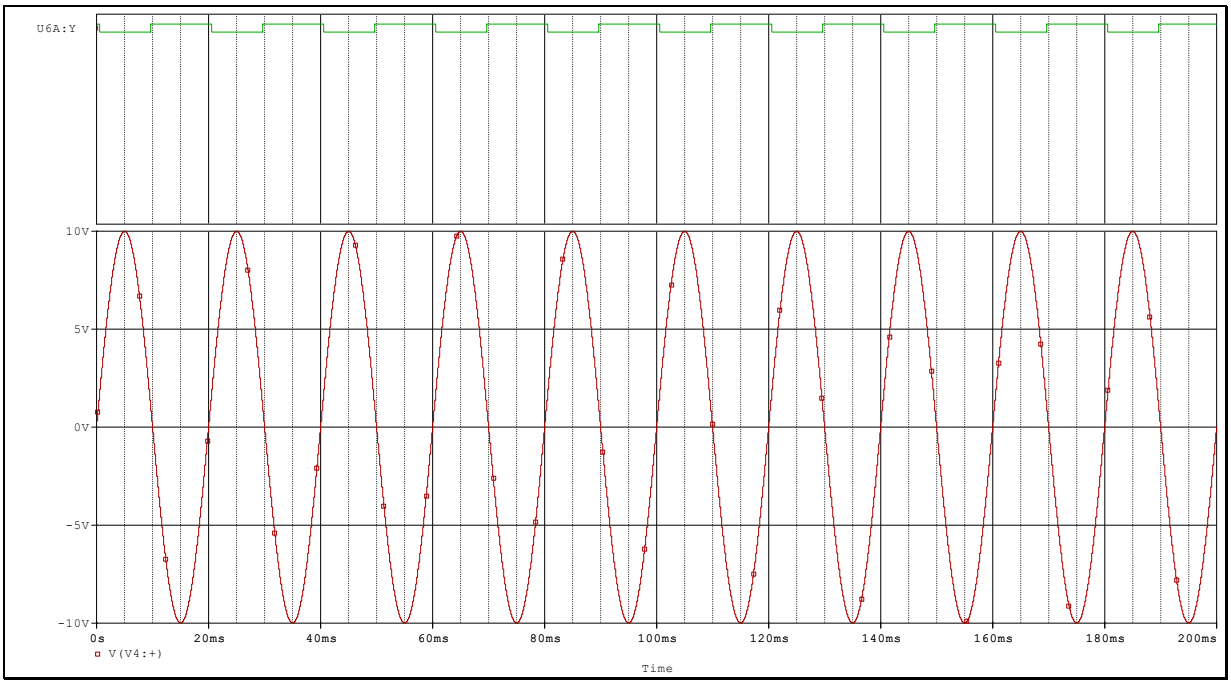


Abbildung 80: Simulationsergebnis des Frequenzteilers, Eingangsignal und Ausgang des Schmitt-Triggers

Nach dem Schmitt-Trigger ist das anfängliche, sinusförmige Signal zu einem invertierten Rechtecksignal gleicher Frequenz geworden. Dieses Signal wird nun auf den Clock-Eingang des Flip-Flops geleitet. Da das Flip-Flop das Signal durch seinen invertierten Eingang nur bei logisch Null weiterleitet wird das Signal hier durch 2 geteilt, wie man in der nun folgenden Abbildung 81 (Mitte) sehen kann. Und bei dem zweiten Flip-Flop passiert dasselbe noch mal und das Signal wird geviertelt.

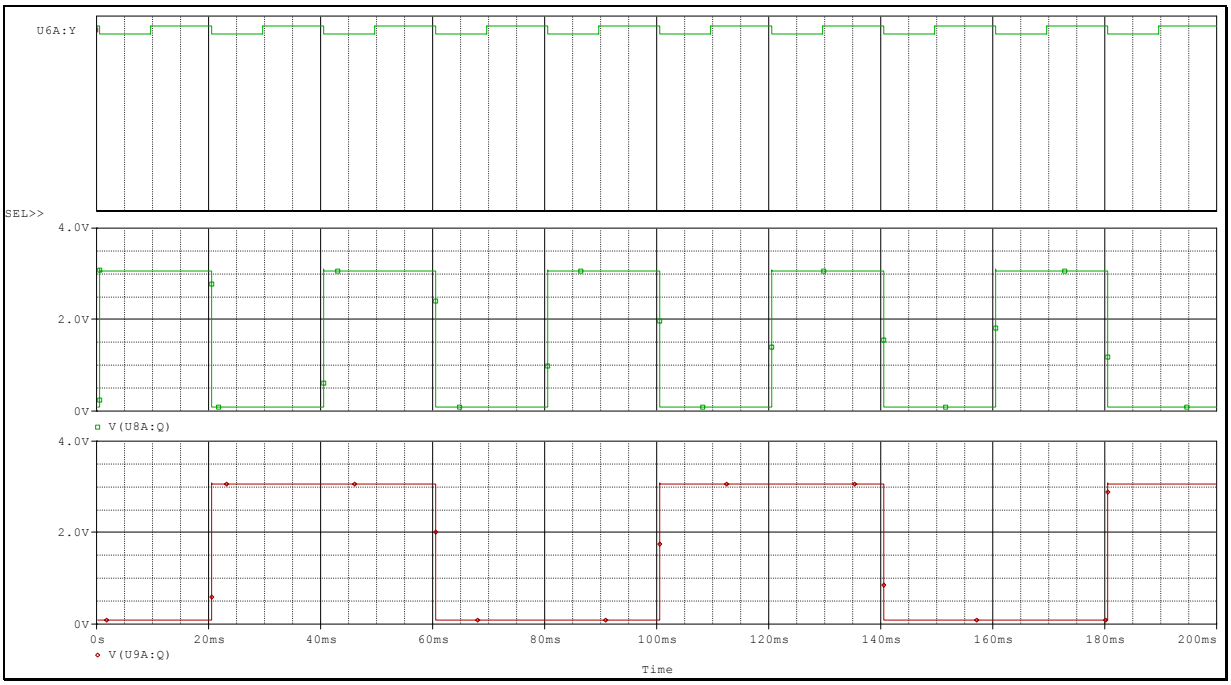


Abbildung 81: Simulation des diskreten Frequenzteilers

Wofür brauchen wir ihn in unserer Schaltung?

Er teilt unser 50Hz Signal auf 1Hz, damit wir jede Sekunde einen über 50 Perioden gemittelten Wert messen und anzeigen können.

Verwendete Bauelemente

Wir haben uns für das integrierte Bauteil 74HC390 (siehe Abbildung 82) entschieden, da es eine Platzersparnis im Layout sowie eine genauere Funktionsweise sicherstellt. Außerdem ist ein durch 50 Teiler diskret nicht so einfach zu realisieren.

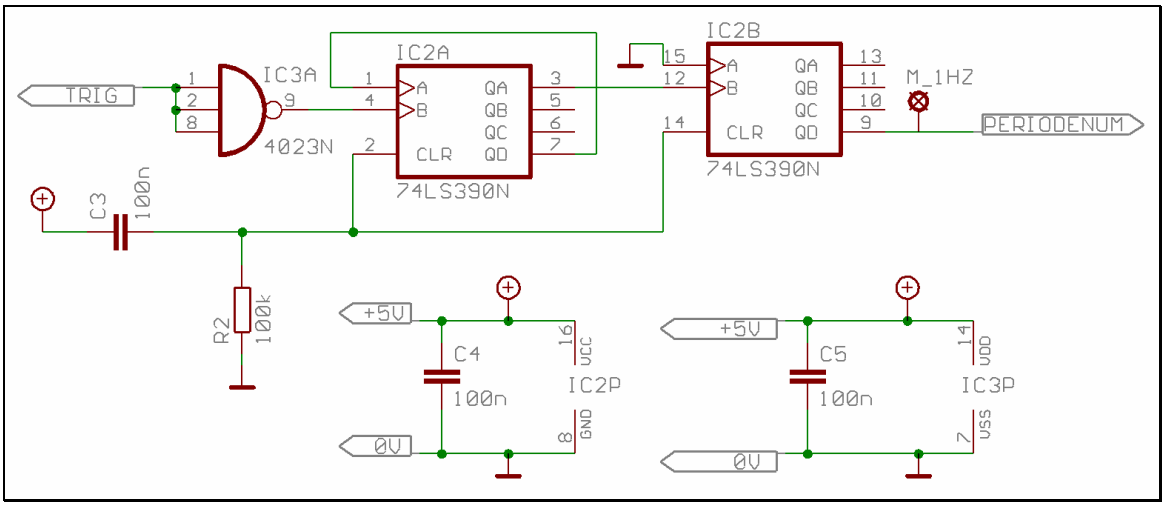


Abbildung 82: Eagle Schematic des Frequenzteilers

Bestückungsplan

Der Frequenzteiler befindet links auf unserer Signalverarbeitungsplatine (siehe Pfeil in Abbildung 83).

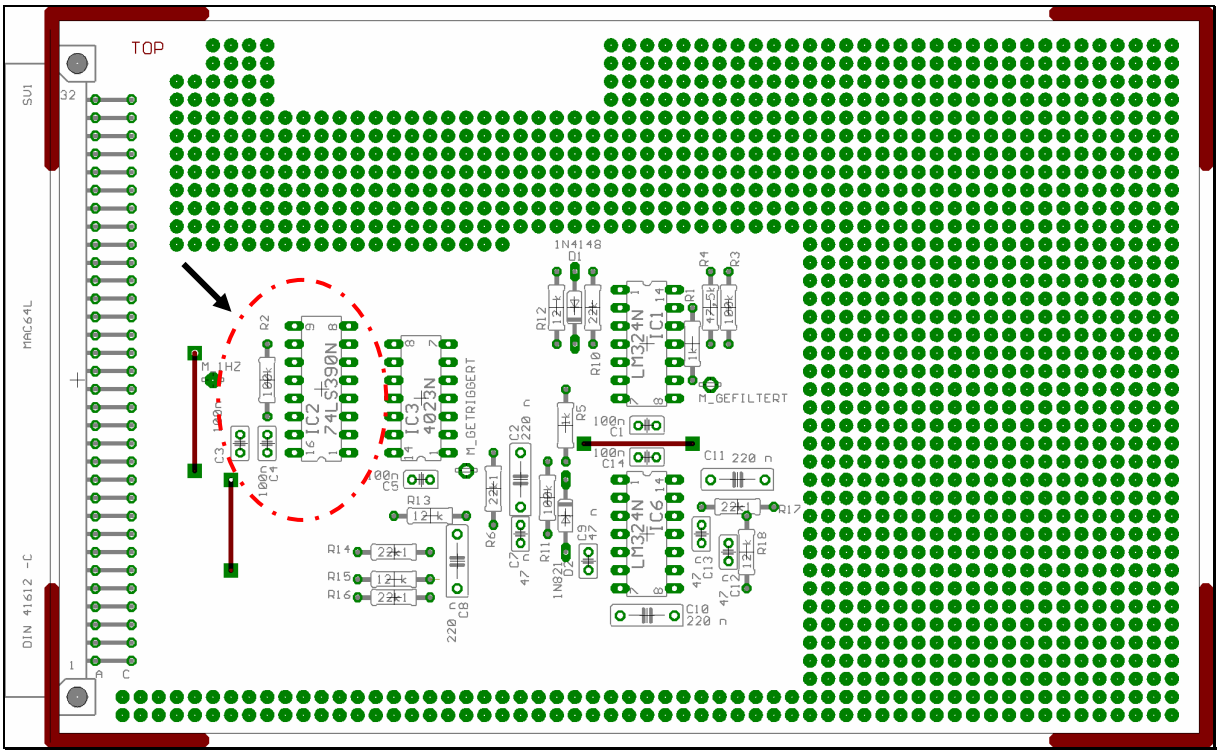


Abbildung 83: Eagle Board der Signalverarbeitungsplatine

Das Messsignal nach dem Frequenzteiler

Neben dem Frequenzteiler befindet sich ein Testpin, an dem das in Abbildung 84 dargestellte Signal, welches mit einem Oszilloskop gemessen wurde, anliegen sollte.

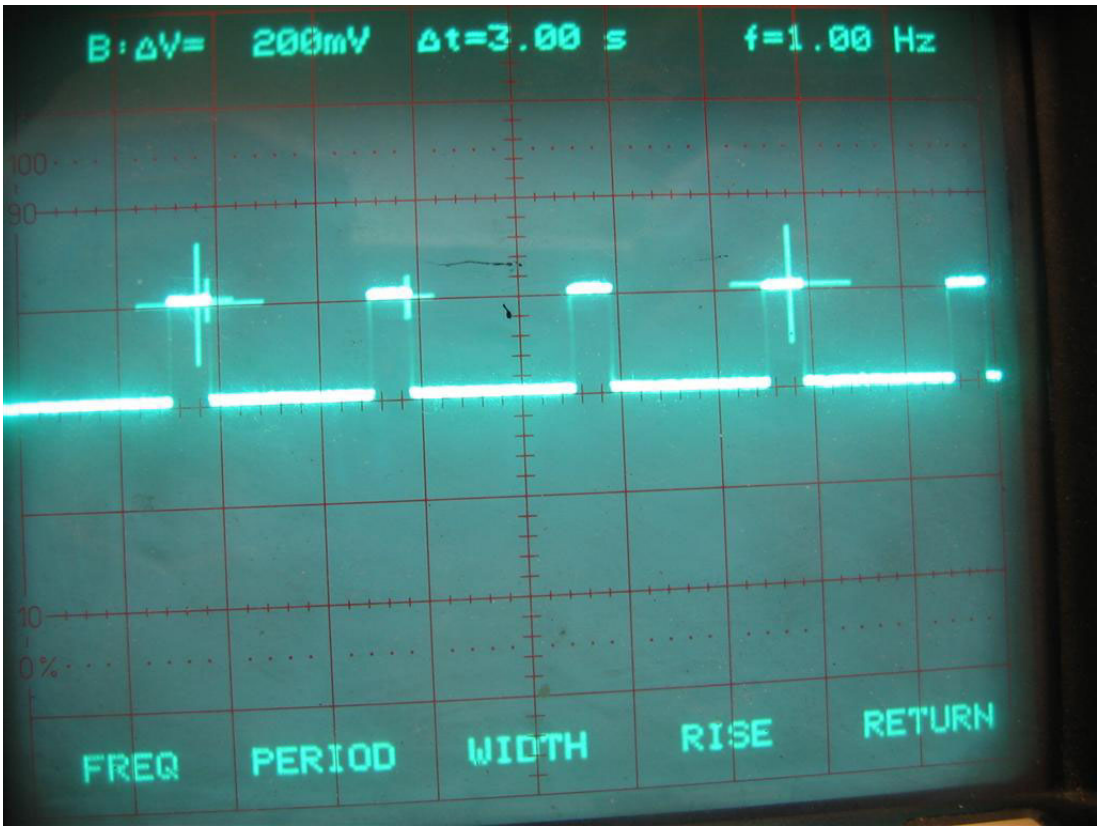


Abbildung 84: Ausgangssignal Frequenzteiler

4.4. Quarzoszillator mit Quarzofen

Teilprojektbeschreibung

Der Quarzoszillator ist die zentrale Zeitbasis der gesamten Schaltung, das erzeugte Taktsignal muss sehr hohe Genauigkeit aufweisen (-+5 PPM) damit das 50 Hz Signal genau genug bestimmt werden kann. Da Schwingquarze einen temperaturabhängigen Frequenzgang haben, musste die Temperatur am Schwingquarz konstant gehalten werden. Dies erfolgt mit einem Quarzofen, der die Temperatur am Quarz bei etwa 50°C konstant halten soll

Funktionsbeschreibung Ofen

Die Schaltung besteht im Groben aus einem Zeitgeber (NE555) und einem OPV der die Regelung übernimmt. Der Zeitgeber schaltet ein Netzwerk aus Analogschaltern (CD4066) jeweils so, dass die Schaltung 80% der Periodendauer einen Transistor heizt und 20 % der Periodendauer genutzt werden um die Temperatur zu messen und den Strom durch den Transistor so einzustellen, dass die voreingestellte Temperatur erreicht wird. Diese Einteilung ist nötig, da bei dieser Schaltung auf einen diskreten Temperatursensor verzichtet worden ist. Stattdessen, wird die Basisspannung am Transistor gemessen. Dafür muss jedoch regelmäßig zwischen Heizen und Messen umgeschaltet werden.

Aufbauanleitung

Der Quarzoszillator und der Heiztransistor müssen mit Hilfe eines Styroporgehäuses gegen Wärmeverlust isoliert werden. Lötet man an die Stelle der Pins des Quarzoszillators zuerst Lötnägel in die Platine. Anschließend legt man eine dünne Scheibe Styropor oder eines anderen isolierenden Materials darauf, so dass es von den Lötnägeln durchbohrt wird. Nun kann der Quarzofen auf den Lötnägeln festgelötet werden.
Anschließend kann der Heiztransistor eingelötet werden. Dabei ist unbedingt daraus zu achten, dass der Transistor mit der Kühlfahne nach oben hin eingelötet werden muss. Anschließend biegt den Transistor so um, dass das Ende der Kühlfahne, dass über das Gehäuse hinausragt, auf dem Quarzoszillator liegt. Zwischen der Kühlfahne des Transistors und dem Quarzoszillator wird eine Glimmerscheibe zur elektrischen Isolierung sowie etwas Wärmeleitpaste zur besseren thermischen Kopplung angebracht.
Schließlich höhlt man eine Quader Styropor, so aus dass man ihn als Kappe auf den Quarzoszillator samt Heiztransistor stecken kann.

Bedienungsanleitung

Die Kalibrierung des Quarzofens erfolgt einmal beim Aufstellen und sonst bei weiteren Wartungsarbeiten wenn nötig. Um den Ofen zu kalibrieren, muss der Strom am Transistor und die Temperatur am Quarz gemessen werden.

1. Der Strom wird mit dem Potentiometer auf etwa 30 mA eingestellt und die Temperatur wird gemessen.
2. sinkt der Strom deutlich unter 30 mA (dabei steigt die Temperatur) wird das das Potentiometer weiter in Richtung R8 gedreht so dass erneut ein Strom von 30mA durch den Transistor fließt
3. man wiederholt Schritt 2 bis sich eine Temperatur von etwa 50 °C einstellt. Sollte die Temperatur aufhören zu steigen, so wird ein höherer Heizstrom eingestellt bis die 50°C erreicht werden.
4. Sind die 50°C erreicht wird das Potentiometer wieder langsam zurückgedreht so dass die Temperatur nicht weiter steigt.

Nun ist der Quarzofen Kalibriert.

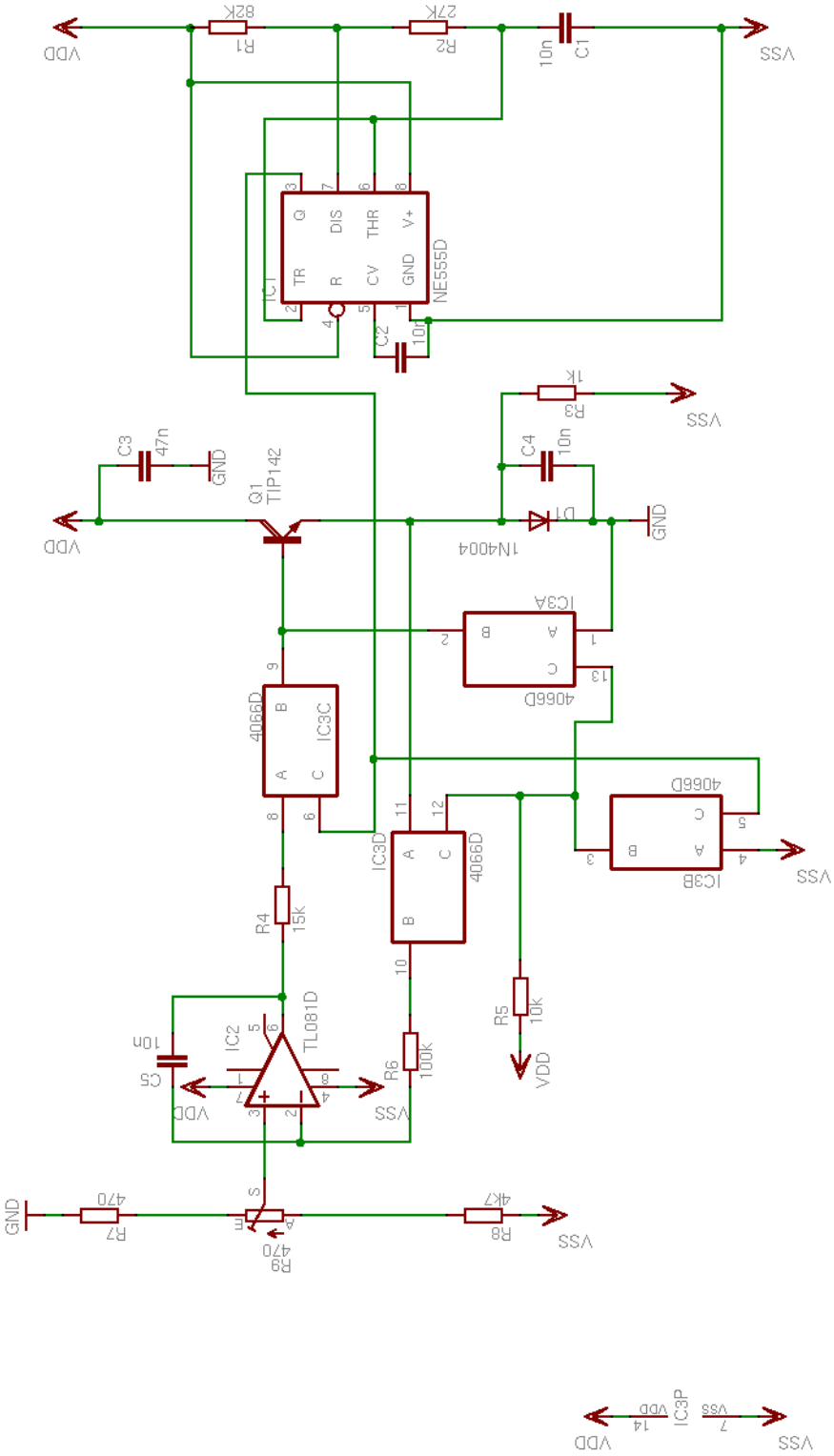


Abbildung 85

Technische Daten

Schwingfrequenz des Quarzoszillators	14 MHz
Betriebsspannung	+8V,-8V DC
Betriebsstrom des Ofens	30-80 mA
Anheizstrom des Ofens	1,2-1,5 A

Funktionsbeschreibung Quarzoszillator

Schaltplan

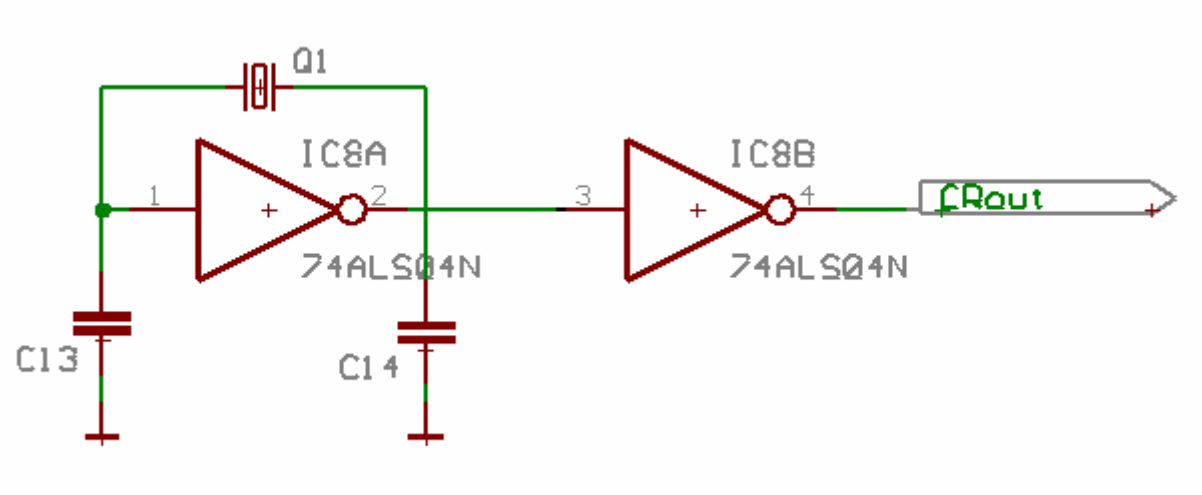


Abbildung 86

Beschreibung

Die Schaltung ist eine Standard Quarzoszillator Schaltung wie sie fast überall verwendet wird, (Mikrocontroller, Uhren ....) dabei handelt es sich um einen invertierenden Verstärker (logisches Inverter Gatter) der über den Quarz vom Ausgang auf dein Eingang zurück koppelt. Durch die 180° Phasenverschiebung des Verstärkers (invertierend), die 180° Phasenverschiebung des Quarzes (360° in Summe -> Mitkopplung) und die recht große Schleifenverstärkung schwingt die Schaltung auf der Quarzresonanzfrequenz

Notizen

Die Anfänglichen Forderungen nach einer Frequenzstabilität <20 PPM konnten in unserem Rahmen nicht erfüllt werden, da Quarzoszillatoren allein durch Alterungserscheinungen in einem rel. kleinem Zeitraum verhältnismäßig große Frequenzschwankungen aufweisen (5PPM/ Jahr). Aus Kostengründen konnten nur die Standardquarze verwenden werden, weshalb unser Augenmerk auf die Stabilisierung der Temperatur gelegt wurde. Nach zahlreichen Testversuchen, sind wir dazu übergegangen einen standard Quaroszillator zu verwenden weil dieser eine höhere Flankensteilheit aufweist

Layout

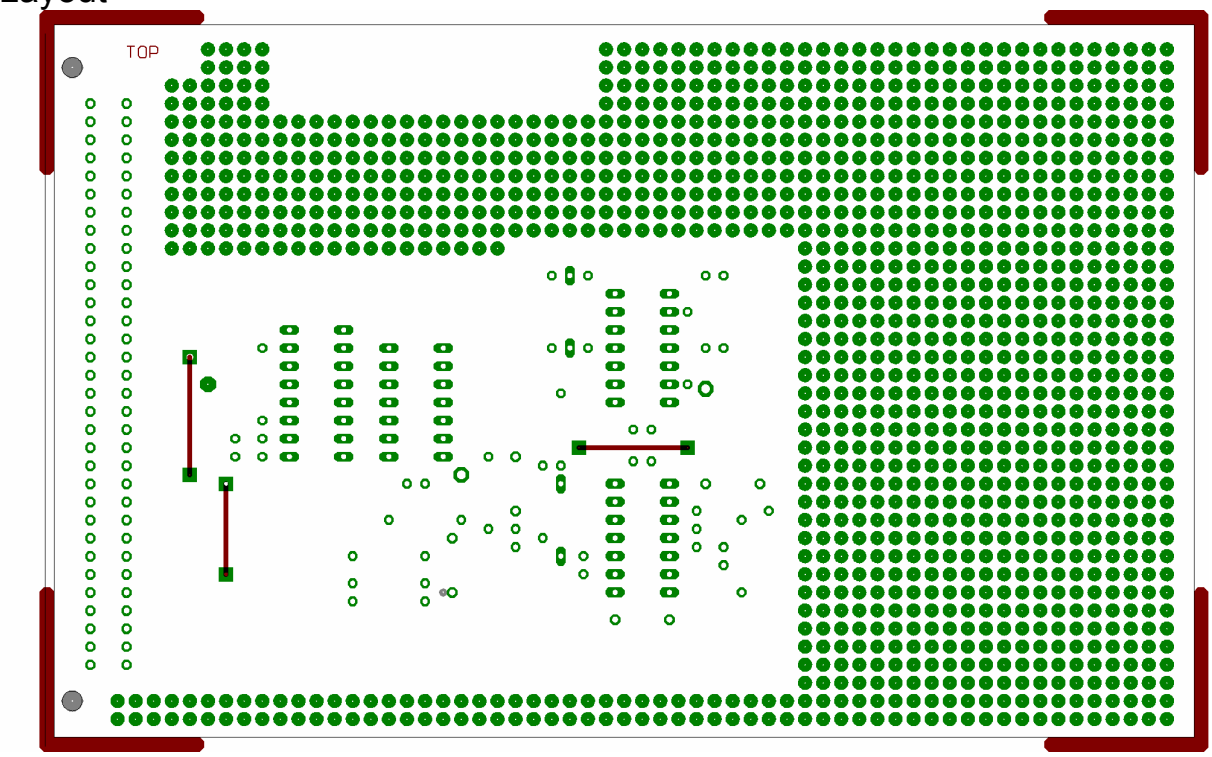
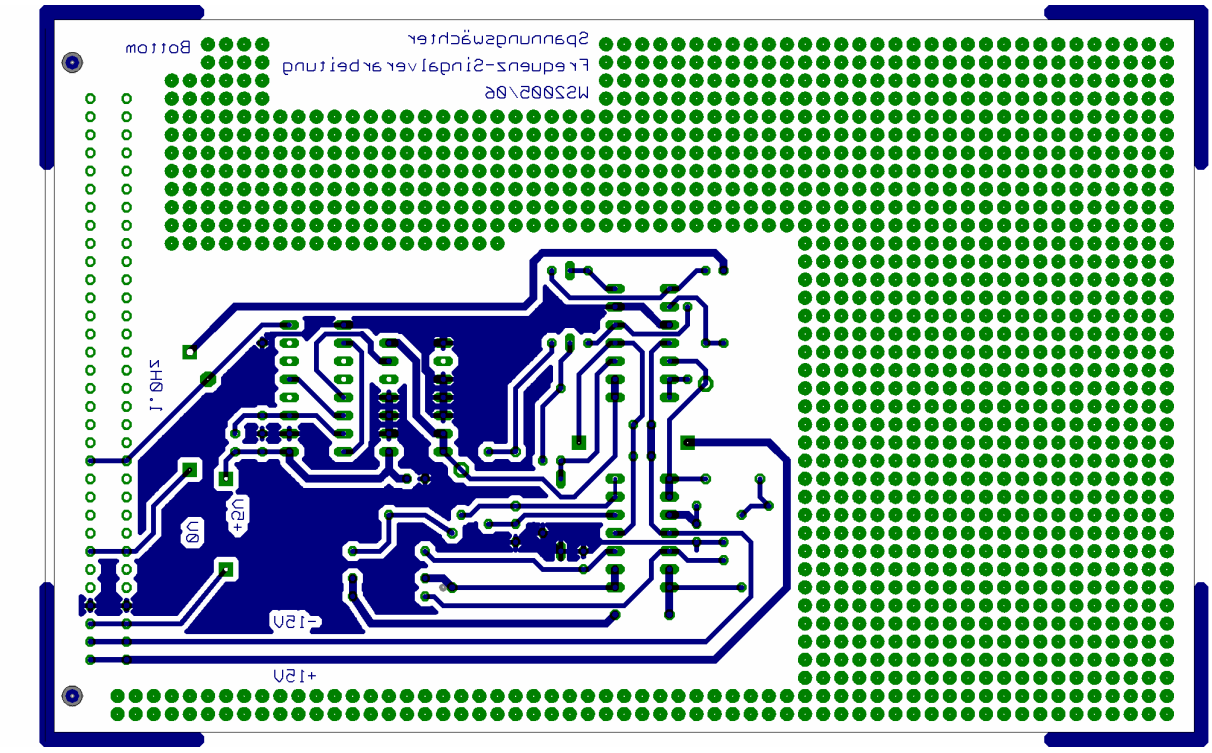


Abbildung 87: layout der gesamten Analogplatine



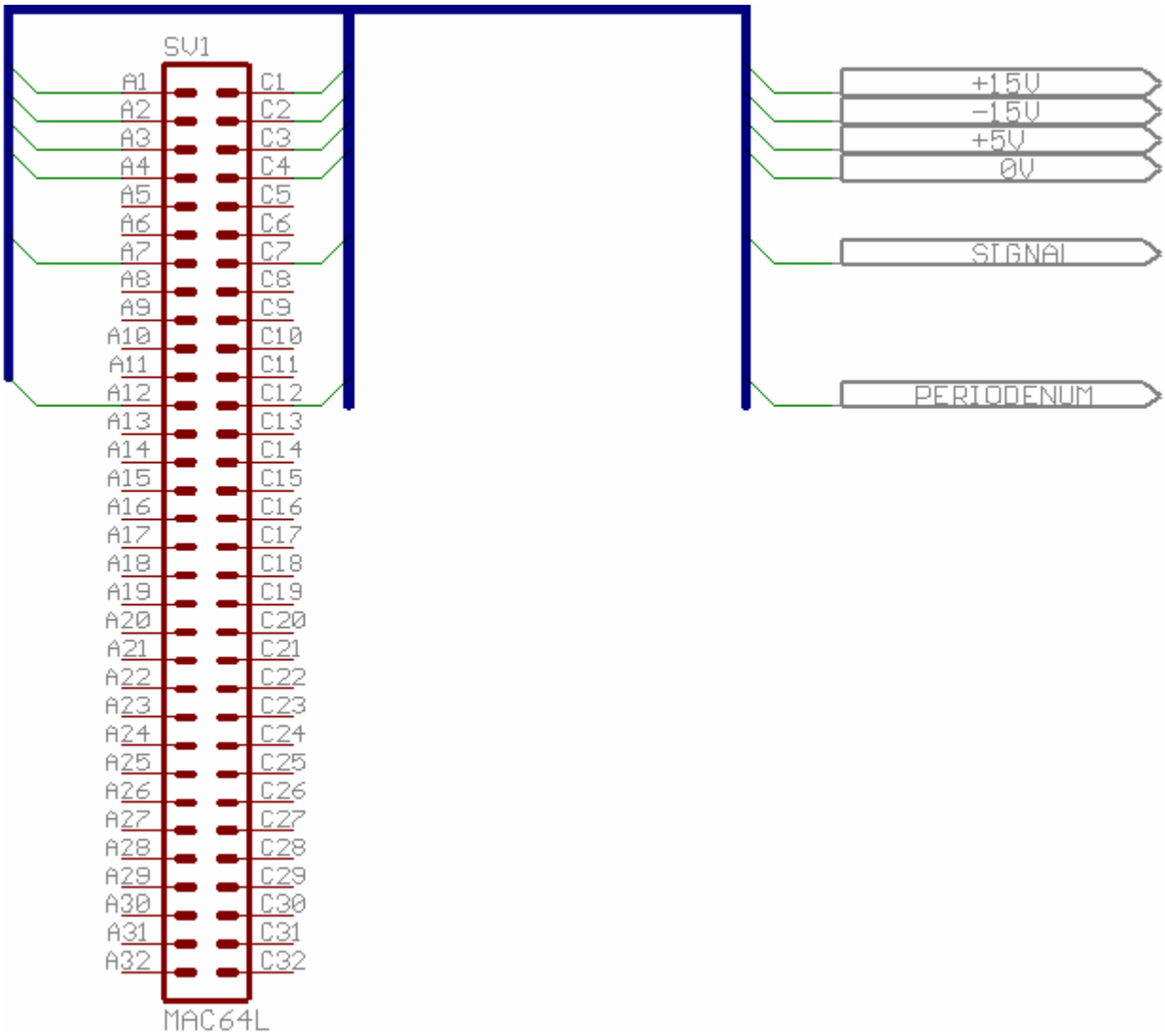


Abbildung 88: Stecker der Analogplatine

4.5. Genauigkeit der Messung und Umrechnung

Auswahl des Messverfahrens und Bestimmung der benötigten Messgenauigkeit

Wir haben festgestellt, dass wir die Frequenz mindestens auf $\pm 10\,mHz$ genau messen müssen, um die alltäglichen Schwankungen in der Netzfrequenz noch messen zu können. Sinn würde es aber machen zur Sicherheit, zu versuchen eine Genauigkeit von $\pm 1\,mHz$ zu erreichen. Im Folgenden wird versucht die Anforderungen an die anderen Schaltungsteile daraus herzuleiten.

absolute Frequenzmessung

Wie in [1] beschrieben kann man die Frequenz mit Hilfe einer Frequenzabhängigen Wien-Brücke Messen.
Da diese Variante aber schlecht automatisierbar ist und die bei Widerständen und Kondensatoren zu erwartenden Toleranzen zu stark das Messergebnis verfälschen würden, haben wir diese Methode nicht weiter verfolgt.

direkte Messung der Frequenz

Wie in [1] beschrieben errechnet sich der Messfehler bei der direkten Messung der Frequenz als:

$$\frac{\Delta f_x}{f_x} = \pm\left(\left|\frac{\Delta f_0}{f_0}\right| + \left|\frac{\Delta N_x}{N_x}\right|\right)$$

Hierbei ist f_x die zu messende Frequenz, f_0 die Frequenz der Zeitbasis und N_x die Anzahl der gezählten Perioden. N_x ist stets bis auf $\Delta N_x = \pm 1$ genau. Daraus folgt:

$$\frac{\Delta f_x}{f_x} = \pm\left(\left|\frac{\Delta f_0}{f_0}\right| + \left|\frac{1}{N_x}\right|\right)$$

$$\Leftrightarrow \frac{\Delta f_x}{f_x} = \pm\left(\frac{|\Delta f_0|}{f_0} + \frac{1}{N_x}\right)$$

$$\Leftrightarrow \Delta f_x = \pm\left(\frac{|\Delta f_0|}{f_0} + \frac{1}{N_x}\right) f_x$$

Dann ist:

$$\Delta f_x \geq \frac{f_x}{N_x}$$
$$\Leftrightarrow N_x \geq \frac{f_x}{\Delta f_x}$$
$$\Rightarrow N_x \geq \frac{50Hz}{1mHz}$$
$$\Leftrightarrow N_x \geq 50000$$

Wir müssten also unabhängig von der Genauigkeit der verwendeten Zeitbasis mindestens 50000 Perioden messen, um die gewünschte Genauigkeit zu erreichen. Das würde dann $\frac{50000}{50Hz} = 1000s \approx 16min$ dauern. In dieser Zeit hätten sich dann wahrscheinlich fast alle Frequenzschwankungen herausgemittelt. Deshalb ist schien diese Variante, die Frequenz zu messen nicht besonders erfolgsversprechend.

Messung der Periodendauer und Umrechnung in die Frequenz

Misst man die Periodendauer und berechnet daraus die Frequenz, so darf der Messfehler der Periodendauer nicht mehr als

$$T + \Delta T = \frac{1}{f + \Delta f}$$
$$\Delta T = \left| \frac{1}{f + \Delta f} - \frac{1}{f} \right|$$
$$\Delta T = \left| \frac{1}{50\ Hz + 1\ mHz} - \frac{1}{50\ Hz} \right|$$
$$\Delta T = 400\ ns$$

betragen. Dies ist somit auch die maximale Periodendauer unserer Zeitbasis. Diese hätte somit eine Frequenz von mindestens $f_{\min} = \frac{1}{400\ ns} = 2{,}5\ MHz$ zur Folge. Laut [1] bestimmt sich der Messfehler beim Messen der Periodendauer als

$$\frac{\Delta T_x}{T_x} = \pm\left(\left| \frac{\Delta T_0}{T_0} \right| + \frac{1}{N_x} + \frac{\Delta t_s}{T_x N_v} \right)$$

Somit ergibt sich für die Genauigkeit der Zeitbasis:

$$\frac{\Delta T_x}{T_x} \geq \frac{\Delta T_0}{T_0}$$
$$\Delta T_0 \leq \frac{1}{f_0} \frac{\Delta T_x}{T_x}$$
$$\Delta T_0 \leq \frac{400\ ns}{2{,}5\ MHz \cdot 20\ ms}$$
$$\Delta T_0 \leq 8\ ps$$

Dies entspricht einer Frequenzabweichung von

$$f_0 + \Delta f_0 = \frac{1}{T_0 + \Delta T_0}$$
$$\Delta f_0 = \left| \frac{1}{T_0 + \Delta T_0} - \frac{1}{T_0} \right|$$
$$\Delta f_0 = \left| \frac{1}{400\ ns + 8\,ps} - \frac{1}{400\ ns} \right|$$
$$\Delta f_0 = 50\ Hz$$

Daraus folgt eine relative Abweichung von $\frac{50\ Hz}{2{,}5\ MHz} \cdot 10^6 = 20\,ppm$ was mit einem guten Quarz durchaus zu erreichen ist. Die Anzahl der benötigten Zyklen der Zeitbasis ist dann mindestens:

$$\frac{\Delta T_x}{T_x} \geq \frac{1}{N_x}$$
$$N_x \geq \frac{T_x}{\Delta T_x}$$
$$N_x \geq \frac{20\ ms}{400\ ns}$$
$$N_x \geq 50000$$

Dies ist aber sowie so schon die Anzahl der Pulse, die bei der Zeitbasis mit $f_0 = 2{,}5\,MHz$ innerhalb einer Periodendauer auftreten würden $20\,ms \cdot 2{,}5\,MHz = 50000$. Schließlich müssen wir bei der Digitalisierung ein und denselben Punkt mit einer Genauigkeit von Δt_s treffen.

$$\frac{\Delta T_x}{T_x} \geq \frac{\Delta t_s}{T_x \cdot N_v}$$
$$\Delta t_s \leq N_v \cdot \Delta T_x$$

Wenn man davon ausgeht, dass eine Messung pro Sekunde durchaus angemessen ist, d.h. $N_V = 50$ dann ist

$$\Delta t_s \leq 50 \cdot 400\,ns$$
$$\Delta t_s \leq 20\,\mu s$$

Bei einer Amplitude von $\hat{U} = 20V$ müssten wir dann bei einem idealen Sinussignal, z.B. den Nulldurchgang bei:

$$U_{trigg} = \pm\hat{U} \cdot \sin(\omega \Delta t_s)$$
$$U_{trigg} = \pm 20V \cdot \sin(2\pi \cdot 50\,Hz \cdot 20\,\mu s)$$
$$U_{trigg} \approx \pm 2\,mV$$

1.4 Zusammenfassung

Wir haben somit festgestellt, dass sich durch Messung der Periodendauer mit anschließender Umrechnung in die Frequenz am ehesten die gewünschte Genauigkeit erreichen lässt.
Dies setzt allerdings voraus, dass:

- Unsere Zeitbasis mindestens eine Frequenz von $2{,}5\,MHz$ hat.
- dabei eine Frequenzabweichung von höchstens $\pm 50\,Hz \mathrel{\hat{=}} 20\,ppm$
- das Rauschen auf $\pm 2\,mV$ reduziert werden kann
- bei der Umrechnung keine erheblichen Ungenauigkeiten hinzu kommen

Umrechnung

Auswahl der Schaltung

Nun ergab sich folgendes Blockschaltbild:

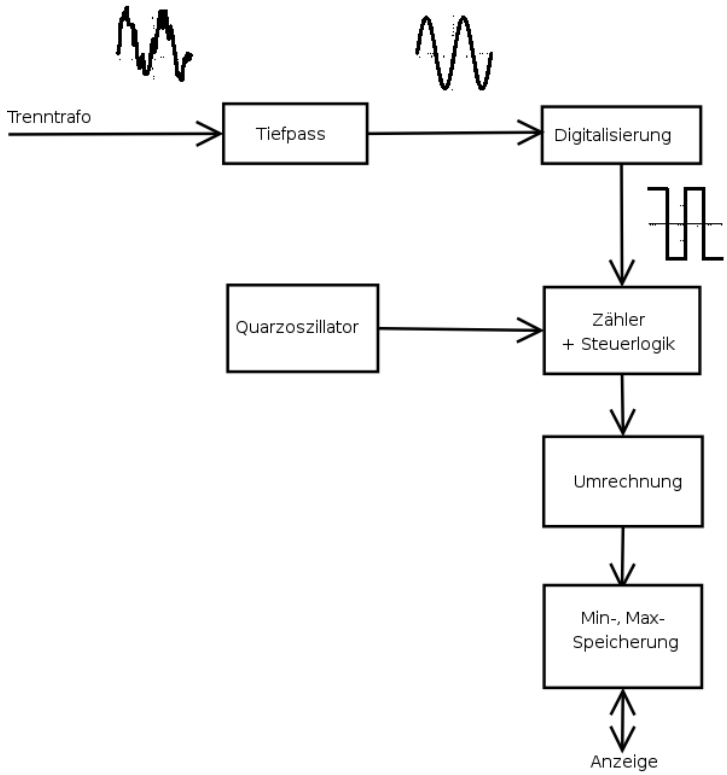


Abbildung 89 Blockschaltbild unserer Schaltung

Es war unstrittig, dass wir für die Messung der Periodendauer einen digitalen Zähler verwenden würden. Für die anschließende Umrechnung in die Frequenz wurden jedoch mehrere Modelle diskutiert:
Das digitale Signal für die Periodendauer wird in ein analoges umgewandelt und anschließend mit Hilfe einer Dividierer-Schaltung in die Frequenz umgerechnet. Schließlich wäre das analoge Signal für die Frequenz wieder in ein digitales umgewandelt worden, um es besser an die Anzeige übertragen zu können.
Die Periodendauer wird mit Hilfe einer diskreten Logik in die Frequenz umgerechnet.

Die Periodendauer wird mit Hilfe eines Mikrocontrollers in die Frequenz umgerechnet.
Wie sich schnell herausstellte waren bei der 1. Variante die bei der Umrechnung zu erwartenden Ungenauigkeiten viel zu groß. Außerdem wäre der Schaltunsaufwand insbesondere für die Realisierung einer Minimal- und Maximalwertspeicherung zu groß gewesen um ihn in der gegebenen Zeit zu bewältigen.
Bei der zweiten Variante bestand die Schwierigkeit darin, dass es keine einfachen Schaltungen zur digitalen Division gibt. Dies hätte man evtl. dadurch umgehen können, dass man die zu den gemessen Periodendauern zugehörigen Frequenzen in einer Wertetabelle in einem EEPROM abgespeichert hätte. Allerdings hätte dies unter Umständen relativ große Speicher benötigt, bei einem 24-Bit Signal für die Periodendauer wären bis zu 16 MB benötigt worden. Außerdem wäre auch hier die Speicherung von Minimal- und Maximalwerten nur mit erheblichem Schaltungsaufwand zu realisieren gewesen.
Somit haben wir uns schließlich für die dritte Variante entschieden, da sich so eine genaue Umrechnung sowie eine Speicherung von Minimal- und Maximalwerten bei einem angemessenen Schaltungsaufwand realisieren ließen.

erster Schaltungsentwurf

Unser erster Schaltungsentwurf sah danach so aus:
Die Schaltung bestand aus zwei Teilen:

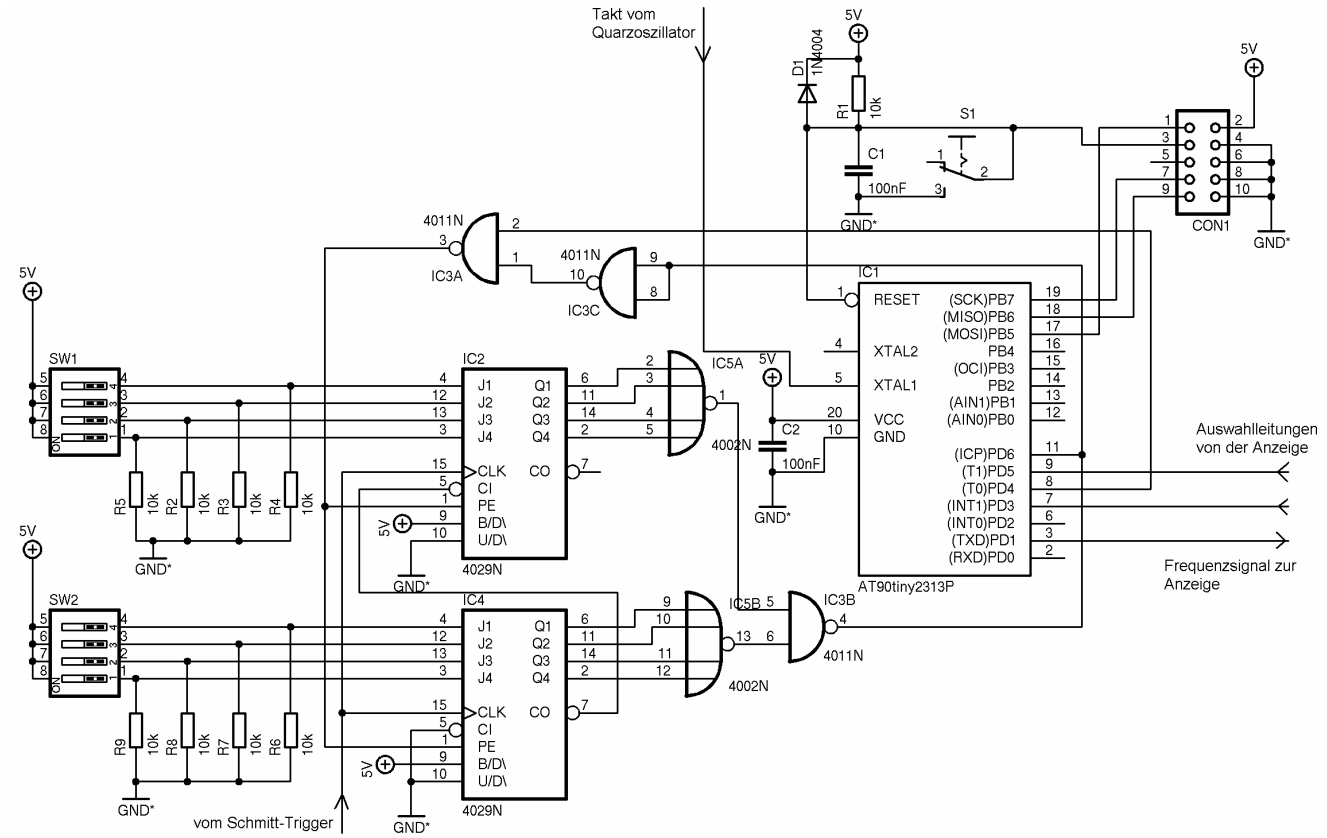


Abbildung 89 erster Schaltungsentwurf für den Digitalteil

einem Vorteiler, der eine jeweils nach einstellbar vielen Perioden einen Puls gibt
einem Mikrocontroller, der die Zeit zwischen zwei Pulsen misst, daraus die Frequenz berechnet, Minimal- und Maximalwerte speichert und das gemessene an die Anzeige überträgt

Vorteiler

Das 50 Hz Rechtecksignal, das aus der Netzspannung gewonnen wurde wird dazu genutzt zwei 4Bit-Presetable-Up/Down – Counter IC2 und IC4 (Zähler, die wahlweise auf oder abwärts zählen und deren Startwert gesetzt werden kann) zu takten. Der CO (Carry-Out Ausgang) von IC4 ist dabei auf den CI (Carry-In Eingang) von IC2 geschaltet, so dass beide zusammen einen 8Bit-Zaähler bilden. Dieser zählt dann von einem durch die Dip-Schalter SW2 und SW1 vorgegebenen Wert abwärts.
Ist der Zählerstand Null erreicht, sind alle Ausgänge Q1 bis Q2 auf LOW, so dass die Ausgänge der beiden NOR-Gatter IC5A und IC5B auf HIGH gehen. Dadurch ergibt sich eine fallende Flake am Ausgang von IC3B. Diese ist dann auf den ICP (Input-Capture) Pin des Mikrocontrollers IC1 geschaltet, so dass dieser das Ende einer Zählperiode feststellen kann.

Andererseits erzeugt sie jedoch auch über IC3A und IC3B eine steigende Flake auf den PE (Preset-Enable) Eingängen der beiden Zähler IC2 und IC4, so dass der Startwert erneut übernommen wird.
So wird also immer nach einer über die DIP-Schalter einstellbaren Zahl von Perioden der Netzspannung ein kurzer LOW-Impuls für die Zeitnahme ausgelöst.
Weiter kann der Mikrocontroller jeder Zeit die Zähler auf ihren Startwert zurücksetzen, indem er Pin 8 auf LOW zieht.

Mikrocontroller

Der Mikrocontroller übernimmt nun die Messung der Periodendauer, die Umrechnung in die Frequenz, die Speicherung- von Minimal- und Maximalwerten sowie die Übertragung an die Anzeige.
Das Taktsignal des Quarzoszillators liegt an Pin 5 von IC1 an. C2 zwischen Masse und der Betriebspannung dient dazu Stromspitzen, die durch die Taktung des Mikrocontrollers entstehen herauszufiltern. CON1 dient als Programmieradapter. Er ist mit den Programmierpins des Mirkocontrollers MOSI, MISO, SCK und RESET sowie Masse und der Betriebsspannung, so verbunden wie es unser Programmiergerät erfordert. R1, C1 und S1 bilden die RESET-Schaltung. Kurz nach dem Einschalten der Betriebsspannung lädt sich C1 über R1 auf. Dadurch wird der RESET-Eingang des Mikrocontrollers ausreichend lange auf LOW gehalten, um ein sicheres Anschwingen des Quarzoszillators zu gewährleisten bevor der Mikrocontroller anläuft. Alternativ kann der RESET-Eingang aber auch jederzeit durch drücken von S1 auf LOW gezogen werden. D1 schützt den RESET-Eingang vor Spannungsspitzen oberhalb der Betriebsspannung.
Schließlich wird die gemessene Frequenz über die serielle Schnittstelle des Mikrocontrollers an Pin 3 an die Anzeige gesendet wobei über Pin 9 und 7 zwischen dem aktuellen, minimalen und maximalen Wert gewählt werden kann.

Programm

Im Mikrocontroller spielt sich nun folgendes ab. Der interne 16 Bit Timer ist mit dem Takt des Mikrocontrollers getaktet. Bei jedem Überlauf wird eine Zählvariable erhöht, so dass über eine längere Zeit als von $2^{16} = 65536$ Taktzyklen gemessen werden kann. Tritt nun an Pin 6 eine fallende Flake auf, so wird der aktuelle Zählerstand automatisch in einem Register gesichert. Anschließend wird daraus und der aus der Anzahl der Timer-Überläufe seit der letzten fallenden Flake die Frequenz berechnet. Zum Schluss wird der Messwert noch über die serielle Schnittstelle an die Anzeige gesendet. Daneben werden stets die Pins 9 und 7 gepollt, um ggf. auf eine Anforderung nach einem Minimal- oder Maximalwert zu reagieren.

Erfahrungen

Nun ging es daran die Schaltung auf einem Steckbrett aufzubauen und zu testen.

- **Kontaktaufnahme zum Kontroller**

Zuerst bemühten wir uns darum den Mikrocontroller programmieren zu können. Hierfür musste lediglich die Reset-Schaltung aufgebaut werden und der Mirkocontroller geeignet mit einem Programmieradapter verbunden werden. Als Taktquelle verwendeten wir zunächst einen Funktionsgenerator.
Allerdings stellte es sich als ziemlich schwierig heraus die richtige Pinbelegung für den Programmieradapter unseres Betreuers, in Erfahrung zu bringen. Zwar gab es dazu im Internet eine ausführliche Dokumentation jedoch wurden dort mehrere Programmieradapter beschrieben, so dass es nicht einfach war die zu unserem Adapter passende Pinbelegung zu finden.

- **Messung der Genauigkeit**

Nachdem dieses Problem gelöst, der Vorteiler aufgebaut und eine erste Version unseres Quarzoszillators verfügbar war, konnten wir nun zum ersten Mal unsere Schaltung testen. Dazu schlossen wir ein 50 Hz Rechtecksignals eines Frequenzgenerators an den Pin 11 des Mikrocontrollers an. Zur Kontrolle haben wir das Signal noch parallel mit einem hochgenauen Frequenzzähler nachgemessen.
Leider stellte sich heraus, dass die von unserer Schaltung gemessene Frequenz um etwa $\pm 1Hz$ schwankte obwohl das angelegte Signal stets konstant bei 50 Hz lag.

- **Änderung der Vorteilerstufe**

Eine erste Untersuchung der Schaltung ergab, dass der LOW-Puls, der nach jeweils 50 Perioden der 50 Hz Signals durch den Vorteiler ausgelöst wurde, so kurz war, dass er sich kaum mit einem Oszilloskop nachweist ließ. Da es somit nicht möglich war zu überprüfen ob der Vorteiler überhaupt korrekt arbeitete, entschlossen wird uns ihn durch einen Frequenzteiler, der die Frequenz, um den Faktor 50 herunterteilen sollte zu ersetzen.
Dieser lieferte zwar nun ein gutes, konstantes Signal von einem 1 Hz die Schwankungen in der gemessenen Frequenz aber blieben.

- **Auswirkungen der Taktfrequenz**

Die Untersuchung des von unserem Quarzoszillators erzeugen Signals ergab, dass dieser ein Signal von sehr genau 10 MHz lieferte. Jedoch war die Zeit die sich das Signal auf LOW befand mit ungefähr 20 ns niedriger als der im Datenblatt des Mikrocontrollers angebende Minimalwert von 25 ns.
Um diesen Fehler auszuschließen ersetzten wir unseren selbstgebauten durch einen handelsüblichen in einem Gehäuse. Dieser lieferte nun ein einwandfreies Signal, an den Schwankungen in der von uns gemessenen Frequenz änderte sich aber nichts.
Weitere Versuchen mit unterschiedlichen Quarzen auf dem stk500 (einem Entwicklungskit für die AVR-Familie) eines der Mitglieder unserer Gruppe ergaben, dass die Schwankungen bei Verwendung eines 14,7456 MHz Quarzes nicht auftraten. Allerdings stellte sich heraus, dass die Schwankungen wieder da waren sobald man die Schaltung auf einem Steckbrett betrieb.

Daraus zogen wir den Schluss, dass parasitäre Kapazitäten auf dem Steckbrett ein Grund für die Ungenauigkeiten war. Deshalb versuchten wir die Schaltung noch einmal auf einer Lochrasterplatine aufzubauen. Da aber die Zeit schon relativ weit fortgeschritten war und wir schon die Platinen für unsere Schaltungen ätzen mussten wurde diese nie vollständig fertig gestellt.
Als nun eine geätzte Platine für unsere Schaltung zur Verfügung stand und die Bauteile aufgelötet waren, stellten wir fest, dass die Schwankungen bei Verwendung eines 10 MHz Oszillators nach wie vor auftraten, bei Verwendung eines 14,7456 MHz Oszillators jedoch nicht.
Zunächst hatten wir für dieses Phänomen keine schlüssige Erklärung. Eine Vermutung war, dass die Flanken des 1 Hz Signals am Pin 11 des Mikrocontrollers zeitlich sehr nahe an anderen Ereignissen, wie einem Timer-Überlauf im Mikrocontrollers lag, so dass ein kritischer Zustand erreicht wurde, in dem Mal auf das eine und Mal auf das andere Signal zuerst reagiert wurde und sich daraus die Schwankungen ergaben. Bei Verwendung eines 14,7456 MHz Quarzes trat diese Situation auf Grund des anderen Verhältnisses von Taktfrequenz und gemessener Frequenz demnach nicht auf. Dies bedeutete jedoch und ließ sich auch durch weitere Versuche verifizieren, dass bei Frequenzen am Eingang die etwas größer als 1 Hz waren, z.B. 1,4 Hz, das gleiche Phänomen auch bei 14,7456 MHz als Taktfrequenz auftrat.
Um das Problem zu umgehen gingen wir dazu über, den Zähltakt des Timers über einen internen Teiler auf 1/8 der Taktfrequenz des Mikrocontrollers herunterzuteilen. Dadurch wurde die Anzahl der Überläufe drastisch reduziert und so der Mikrocontroller entlastet. Parallel dazu verkürzten wir unsere Interruptroutinen und vereinfachten die bis dahin etwas umständliche Berechnung der Periodendauer aus der Anzahl der Überläufe und dem Timerstand.
Dadurch sank unsere Zählfrequenz zwar auf $\frac{14{,}7456\,MHz}{8} = 1{,}8432\,MHz$, was unter den geforderten $2{,}5\,MHz$ lag. Allerdings bekamen wir aber so unsere Messwerte auf $\pm 3\,mHz$ stabil, was aber noch im Rahmen war. Nach diesen Änderungen war es sogar möglich eine ähnliche Genauigkeit mit einer Taktfrequenz von 10 MHz zu erreichen.

Endgültige Schaltung

Im Folgenden wird nun die endgültige Schaltung des Digitalteils vorgestellt. Sie setzt sich aus einer Vorteilerstufe und der Beschaltung des Mikrocontrollers zusammen.

Schaltplan

Vorteiler

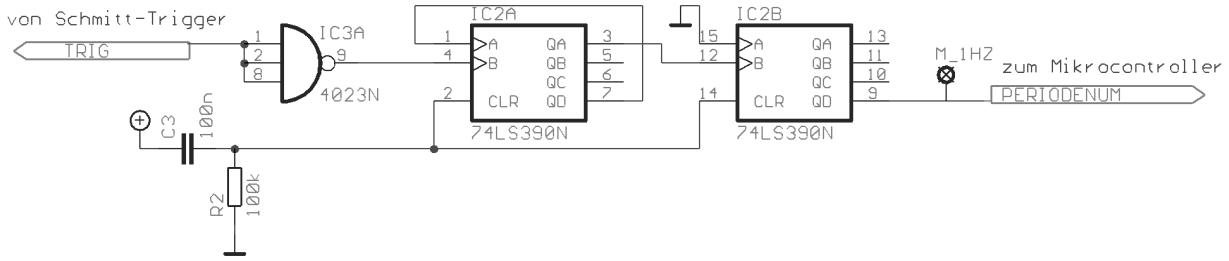


Abbildung 90 endgültige Schaltung der Vorteilerstufe

Die Schaltung zeigt einen Frequenzteiler von 50 Hz auf 1 Hz. Das 50 Hz Eingangssignal wird am Eingang von IC3A eingespeist. IC3A sorgt dafür, dass das Signal am Eingang des Frequenzteilers IC2A auf jeden Fall den Logikpegeln entspricht.
Anschließend wird die Frequenz des Signals von Pin 4 von zum Pin 7 von IC2A um den Faktor 5 heruntergeteilt. Danach erfolgt eine weitere Teilung um den Faktor 2 von Pin 1 zum Pin 3 von IC2A. Schließlich wird die Frequenz des Signals von Pin 12 zum Pin 9 von IC2B erneut um den Faktor 5 geteilt. So ergibt sich insgesamt eine Teilung um den Faktor $5 \cdot 2 \cdot 5 = 50$ und somit ein Signal von 1 Hz am Ausgang.
C3 und R1 dienen dazu die internen Zähler des Frequenzteilers beim Einschalten der Betriebsspannung zu reseten. Kurz nach einschalten der Betriebsspannung lädt sich C3 nämlich über R2 auf, so das für eine kurze Zeit ein HIGH-Pegel an den Pins 2 und 14 von IC2 anliegt.

Mikrocontroller

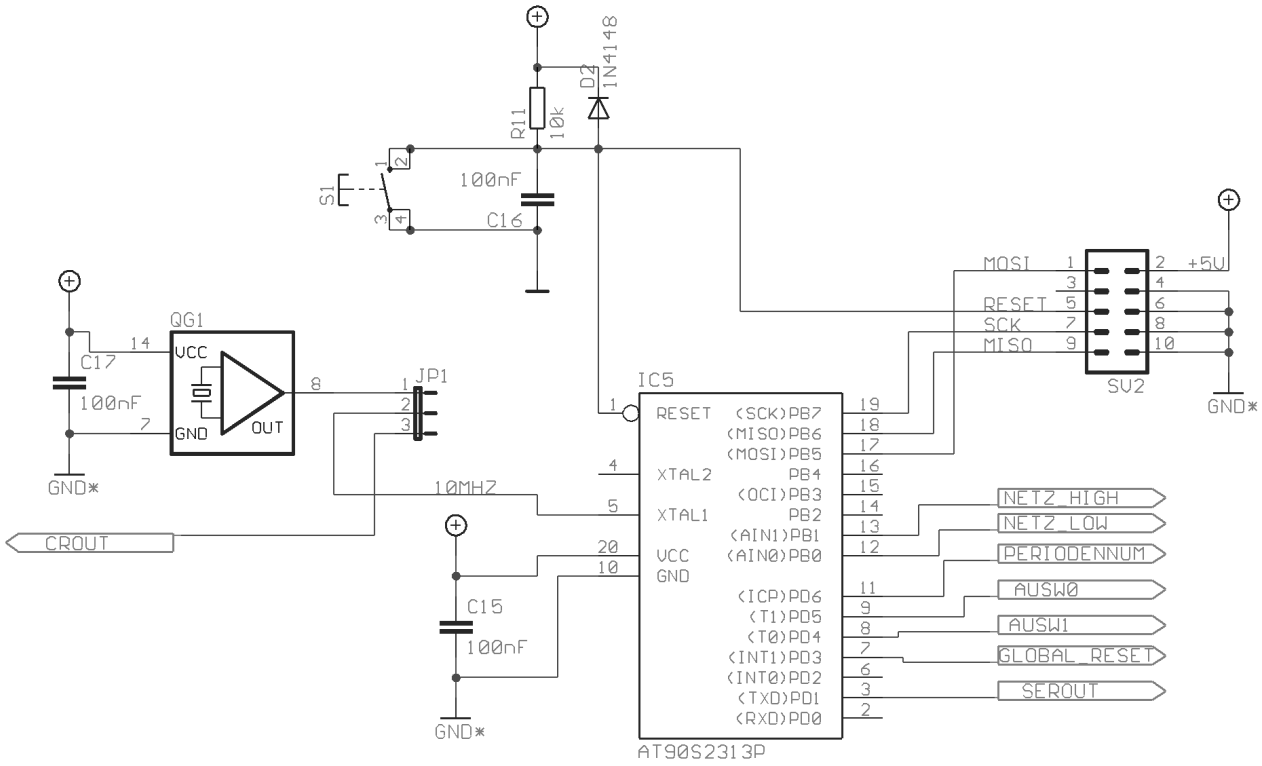
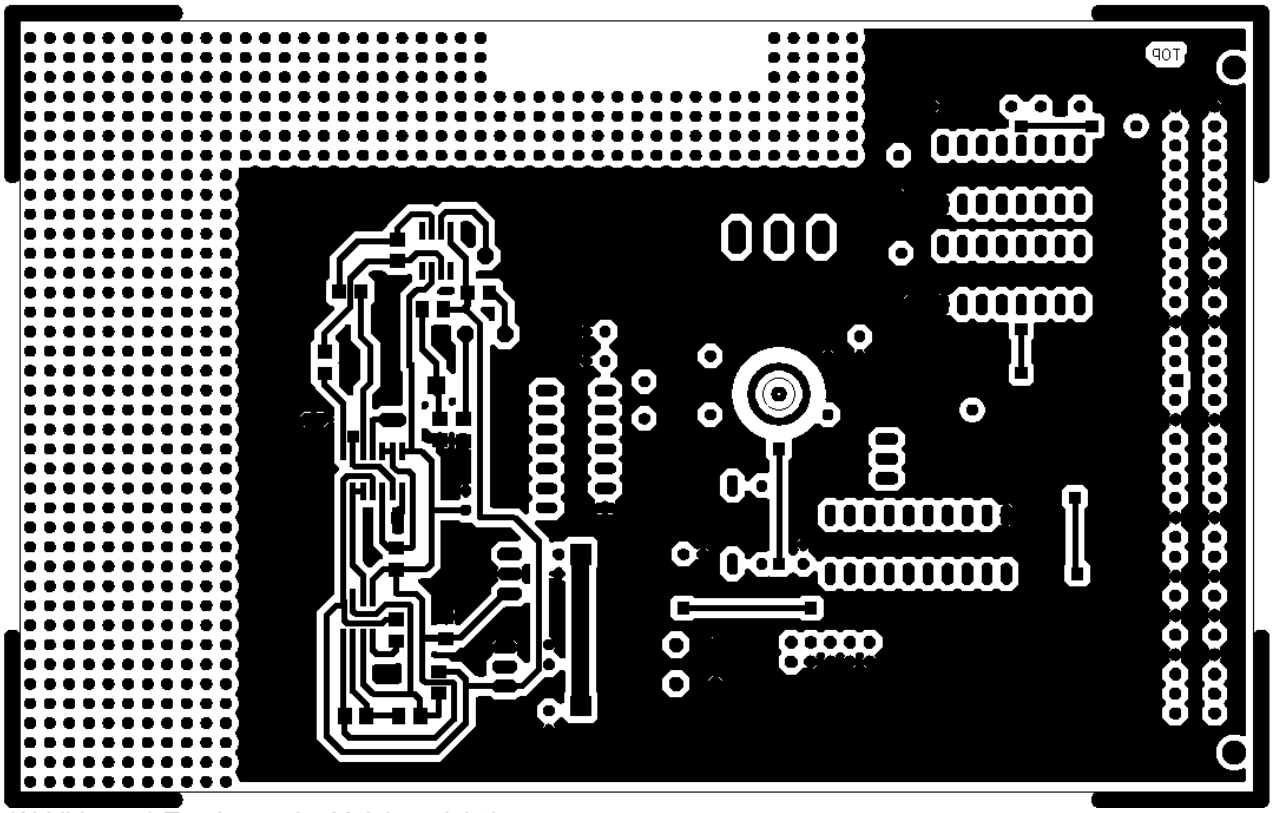


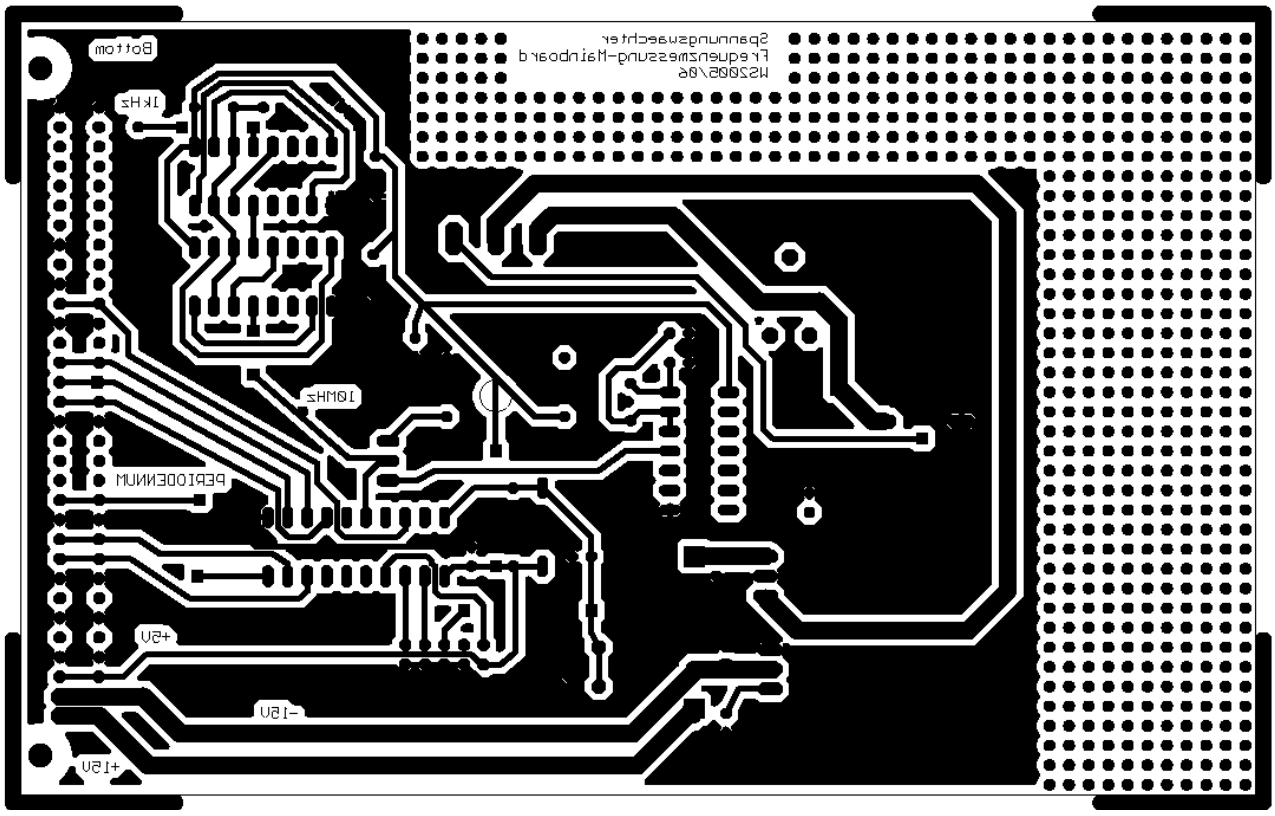
Abbildung 91 endgültige Beschaltung des Mikrocontrollers

Die Schaltung für den Mikrocontroller hat sich wenig verändert. S1 R11 und C16 sorgen wie bereits unter 2.2.2 beschrieben für einen sauberen Reset. SV2 dient als Programmieradapter. An JP1 kann man einstellen ob die Taktfrequenz vom Quarzoszillator QG1 oder von einer anderen Quelle bezogen werden soll. C17 und C15 puffern die Betriebsspannung gegenüber durch den Mirkocontroller oder Quarzoszillator erzeugten Stromspitzen.
Über die Pins 13 und 14 kann der Mikrocontroller erkennen ob im Netz eine Überspannung oder Unterspannung anliegt, über Pin 11 liegt das 1 Hz Signal, dessen Periodendauer gemessen wird an. Mit Pin 11 und 9 kann von der Anzeige ausgewählt werden ob der minimale, maximale oder aktuelle Wert der Frequenz übertragen werden soll. Weiter können über Pin 7 die Minimal- und Maximalwerte zurückgesetzt werden. Das serielle Signal zur Übertragung der Frequenz an die Anzeige liegt an Pin 3 an.

Layouts



(Abbildung 5) Top-Layer der Mainboardplatine



(Abbildung 6) Bottom-Layer der Mainboardplatine

Aufbau

Beim Aufbau der Schaltung sollte man zu aller erst die Durchkontaktierungen von oben bestücken. Weiter ist darauf zu achten, dass alle Fassungen mit der richtigen Orientierung eingelötet werden.
Wenn alle Fassungen und anderen Bauteile eingelötet sind schließt man die Betriebsspannung an die Platine an und überprüft ob sie an allen Punkten richtig anliegt. Anschließend trennt man die Betriebsspannung wieder von der Platine.
Erst danach werden die ICs in der richtigen Orientierung in die Fassungen gesteckt.

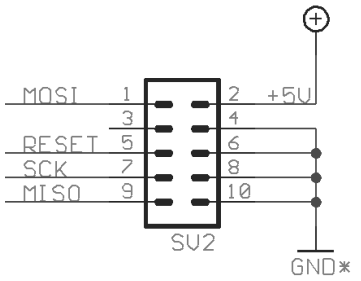
Inbetriebnahmen

Zur Inbetriebnahme wird zunächst, ein geeigneter Programmieradapter mit dem Wannenstecker auf der Platine verbunden. Außerdem muss ein Jumper auf die Pins 1 und 2 von JP1 gesteckt werden damit der Mikrocontroller von dem Quarzoszillator mit einem Takt versorgt wird.
Anschließend verbindet man die Betriebsspannung mit der Platine. Dabei ist darauf zu achten dass die Betriebsspannung in der richtigen Größe und Polung anliegt. Um eine Zerstörung der Schaltung durch falsches Anschließen zu vermeiden empfiehlt es sich die Strombegrenzung des Netzteils möglichst niedrig einzustellen. Im Normalfall sollte die Schaltung keinen Strom größer als $20\,mA$ ziehen.

Nach einschalten der Betriebsspannung kann nun der Mikrocontroller mit Hilfe des angeschlossenen Programmiergerätes programmiert werden. Danach schaltet man die Betriebsspannung ab und entfernt das Programmiergerät vollständig von dem Wannenstecker.
Man kann auch das Programmiergerät für eine spätere, erneute Programmierung vollständig in der Schaltung stecken lassen. Entfernt man jedoch das Programmiergerät nur teilweise, so dass immer noch Kabel mit der Schaltung verbunden sind jedoch nicht mehr mit dem Programmiergerät oder das Programmiergerät mit dem Rechner, so kann dies zu Fehlfunktionen der Schaltung führen. Es hat sich zum Beispiel gezeigt, dass die Schaltung meist nicht mehr zuverlässig funktioniert, wenn man ein kurzes Flachbandkabel ohne angeschlossenes Programmiergerät in dem Wannenstecker stecken lässt.
Nach erneutem Einschalten der Betriebsspannung sollte nun die Schaltung betriebsbereit sein. Sollte sie nicht wie erwartet funktionieren empfiehlt es sich mit den unter 4.5 beschriebenen Schritten fortzufahren.

4.5 Prüfanleitungen

Prüfen der Mikrocontrollerschaltung



(Abbildung 7) Pinbelegung des Programmiergerätes

Um die Mikrocontrollerschaltung testen zu können muss zuvor der Mikrocontroller programmiert werden. Ist es nicht möglich den Mikrocontroller in der Schaltung zu programmieren, hat das wahrscheinlich einen der folgenden Gründe:

- Es liegt entweder keine oder die falsche Betriebsspannung am Mikrocontroller an
  → Schließen sie die Betriebsspannung richtig an die Schaltung an und vergewissern sie sich das keine Kurzschlüsse auftreten. Hat einmal eine zu hohe oder falsch gepolte Betriebsspannung angelegen, kann es zur Zerstörung der ICs und des Quarzoszillators gekommen sein. In diesem Fall sollten sie alle ICs und den Quarzoszillator auswechseln.
- Der Quarzoszillator funktioniert nicht richtig
  → überprüfen sie mit einem Oszilloskop, ob an Pin 8 des Quarzoszillators ein symmetrisches 14,7456 MHz Rechtecksignal zwischen 0V und 5V anliegt. Ist dies nicht der Fall ist der Quarzoszillator defekt und muss ausgetauscht werden. Liefert der Quarzoszillator das richtige Taktsignal, überprüfen Sie ob dies auch am Pin 5 des Mikrocontrollers anliegt. Ist dies nicht der Fall haben sie wahrscheinlich den Jumper auf den Pins 1 und 2 von JP1 vergessen. Setzten Sie ihn jetzt.
- Der Wannenstecker ist verkehrt herum eingelötet
  → Löten sie ihn aus und richtig herum wieder ein
- Die Pinbelegung des verwendeten Programmiergerätes stimmt nicht mit der des Wannensteckers überein
  → Bauen Sie einen geeigneten Adapter oder verwenden Sie ein Programmiergerät mit passender Pinbelegung

Gelingt es überhaupt nicht den Mikrocontroller in der Schaltung zu programmieren kann dieser auch alternativ in einer anderen Schaltung programmiert werden und anschließend in die Schaltung eingesetzt werden.
Anschließend muss man das serielle Signal, das auf dem Bus mit einer geeigneten Anzeige verbinden. Hierfür verwendet man am besten die von uns im Rahmen des Projekts erstellte Testplatine. Sie besitzt bereits einen Pegelwandler und eine passende Buchse, um an die serielle Schnittstelle eines beliebigen PCs angeschlossen zu werden. Der Mikrocontroller verwendet dabei folgendes Datenformat: Baudrate: 2400, 8 Datenbits, keine Parität, zwei Stoppbits. Um die Ausgabe der Schaltung besser lesen zu können empfiehlt es sich das von uns geschrieben Programm („TestEmpf“) zu verwenden. Bitte beachten Sie, dass das Programm "TestEmpf" stets die COM1 als serielle Schnittstelle verwendet. Möchten sie eine andere Schnittstelle als COM1 verwenden, muss den Quelltext entsprechend anpassen und das Programm neu kompilieren. Die Ausgabe kann zwar ebenso mit einem beliebigen Terminal, z.B. Hyperterminal, verfolgt werden ist in diesem Falle aber nicht ohne weiteres lesbar.
Nach jedem Reset sendet der Mikrocontroller das Signal 0x11223344 über die serielle Schnittstelle, dies entspricht einer Frequenz von 4660 mHz. Überprüfen Sie durch mehrmaliges Drücken von S1 ob das Signal zuverlässig übertragen wird. Ist das nicht der Fall ist entweder die serielle Schnittstelle des Mikrocontrollers nicht richtig angeschlossen, Sie haben die serielle Schnittstelle des PCs nicht richtig

eingestellt oder das Programm im Mikrocontroller wurde für eine andere als die verwendete Taktfrequenz geschrieben.
Wird das Signal nach dem Reset richtig übertragen schließen sie ein 1 Hz Rechtecksignal zwischen 0V und 5V an Pin 11 des Mikrocontrollers an. Anschließend sollte stabil eine Frequenz von $50000\,mH \pm 3\,mHz$ angezeigt werden.
Ist dies der Fall ist funktioniert die Mikrocontrollerschaltung.

Prüfen des Frequenzteilers

Zum Prüfen des Frequenzteilers schließen Sie ein 50 Hz Rechtecksignal mit einem Pegel 0V und 5V an den Pin 4 des Frequenzteilers an. Anschließend überprüfen Sie mit einem Oszilloskop, ob die folgenden Signale anliegen:
Pin 1 und 7: 10 Hz Rechteck zwischen 0V und 5V mit Tastverhältnis (HIGH:LOW) 1:4
Pin 3 und 12: 5 Hz Rechteck zwischen 0V und 5V mit Tastverhältnis (HIGH:LOW) 1:1
Pin 9: 1 Hz Rechteck zwischen 0V und 5V mit Tastverhältnis (HIGH:LOW) 1:4

Zusammenfassung

Ergebnisse der einzelnen Baugruppen:

Filterung des Messsignals

Als Schaltung wurde ein aktives Besselfilter 8. Ordnung mit einer Knickfrequenz von 60 Hz umgesetzt. Da jede einzelne Filterstufe eine Verstärkung mit sich brachte, wurde das Eingangssignal mittels eines invertierenden Verstärkers halbiert, um nicht in den Sättigungsbereich der Operationsverstärker zu gelangen. Somit konnte ein Messsignal hoher Reinheit gewährleistet werden.

Digitalisierung des Messsignals

Als Schaltung wurde ein nicht invertierender Schmitt-Trigger mit anschließender Spannungsbegrenzung realisiert. Durch ein anschließendes NAND-Gatter wurde das Signal in ein digitales Rechtecksignal umgewandelt, welches mit der Messfrequenz zwischen Low und High alterniert.

Frequenzteiler

Das durch den Schmitt-Trigger digitalisierte 50Hz Signal wurde mittels eines Frequenzteilers in ein 1Hz-Signal geteilt, um eine Mittlung über 50 Perioden zu ermöglichen. Anschließende Testläufe haben ergeben, dass die Frequenzteilung kaum einen Einfluss auf die Genauigkeit des Messsignals hat.

Quarzoszillator mit Quarzofen

Als Quarzoszillator wurde ein Standardoszillator verwendet, der auf Grund hoher Genauigkeitsanforderungen Temperaturstabilisiert wurde. Der Quarzofen wird wegen des Platzsparens in SMD-Bauweise umgesetzt. Auswirkungen auf die Genauigkeit konnten bis jetzt nicht nachgewiesen werden, da ein Betrieb im Freien noch nicht möglich war.

Umrechnung über einen Mikrocontroller

Mittels der Messung der Periodendauer mit anschließender Umrechnung in die Frequenz ist am ehesten die gewünschte Genauigkeit von der Frequenzmessung zu erreichen. Daraus ergeben sich folgende Anforderungen:

Einsatz einer Zeitbasis von einer Frequenz von mindestens 10 Mhz

Frequenzabweichungen der Zeitbasis von höchstens 20 ppm

Minimierung des Rauschens

keine erheblichen Ungenauigkeiten bei der Umrechnung

Diese Anforderungen konnten in allen Bereichen umgesetzt werden. Der Mikrocontroller übernimmt die Messung der Periodendauer, die Umrechnung in die Frequenz und die Speicherung- von Minimal- und Maximalwerten. Die gemessene Frequenz wird über die serielle Schnittstelle des Mikrocontrollers an die Anzeige gesendet, wodurch sich die Anzahl der Verbindungsleistungen von x auf y verringert.

Ergebnisse der Gesamtschaltung:

Die Ergebnisse unserer Baugruppe wurden in den Teilberichten unserer Schaltungsteile beschrieben und sollen im voraus leistungsorientierten Fragestellungen unterzogen werden.

- Wurde das Gruppenziel erreicht?

In der Gesamtheit der Bearbeitung des Aufgabenfeldes Frequenzmessung kann festgestellt werden, dass die Vorüberlegungen und späteren Absprachen sowie Änderungen einiger Schaltungsteile zu einem nur zufriedenstellend Ergebnis geführt haben.

- Welche Genauigkeit wurde erreicht?

Um Referenzwerte zu unseren Messwerten zu ermitteln, benutzten wir den Frequenzzähler PM 6654C der Marke Philips. Dieser ermöglicht eine Messung der Frequenz mit einer Genauigkeit +/-10 µHz. Als Eingangssignal wurde ein digitaler Frequenzgenerator benutzt um mögliche Schwankungen zu minimieren. In verschieden Testläufen haben sich folgende Abweichungen ergeben:

Versuch 1: Erzeugen eines Sinussignals mit einem digitalen Signalgenerator

Signalgenerator 45 Hz – 55 Hz in 2.5 Hz-Schritten	***45 Hz***	***47.5 Hz***	***50 Hz***	***52.5 Hz***	***55 Hz***
Messung der Frequenz des Eingangssignals mit PM 6654C (Philips) in Hz	45.06529 – 45.06563	47.56916 – 47.56941	50.07286 – 50.07317	52.57667 – 52.57698	55.08015 – 55.08047
Frequenz des geteilten Signals PM 6654C (Philips) in Hz	0.9013120 – 0.9013140	0.95138630 – 0.9513873	1.0014613 – 1.0014630	1.0515341 – 1.0515355	1.1016058 – 1.1016078
Zu erwartender Wert (Messfrequenz / 50) in Hz	0.9013126	0.9538632	1.0014572	1.0515334	1.101603
Messung mit Mikrocontroller in Hz	45.064 – 45.065	47.569	50.072 – 50.073	52.576	55.080

In allen 5 Messungen konnte nachgewiesen werden, dass eine Genauigkeit der Schaltung von einem Millihertz erreicht wurde.

Im anschließenden Testlauf sollte die Genauigkeit der Schaltung beim Anliegen der Netzspannung als Messsignal überprüft werden. Hierfür wurde ein Trenntrafo benutzt, um das Messsignal galvanisch zu trennen, sowie die Spannung auf einen für unsere Schaltung zulässigen Pegel zu transformieren. Hierbei war zu beachten, dass unsere Schaltung über 50 Perioden mittelt, wodurch nur jede Sekunde ein Messwert zur Verfügung steht. Somit wurde das Gate des Frequenzzählers PM6654 ebenfalls auf eine Sekunde eingestellt, um möglichen Frequenzschwankungen der Netzspannung im selben Takt wie unsere Schaltung zu mitteln.

Versuch 2: Messung der Netz-Frequenz an verschiedenen Frequenzzählern

Signal aus dem Trenntrafo	***50 Hz***
Messung der Frequenz des Eingangssignals mit PM 6654C (Philips)	49.995 – 50.009
Messung mit Mikrocontroller	49.995 – 50.009

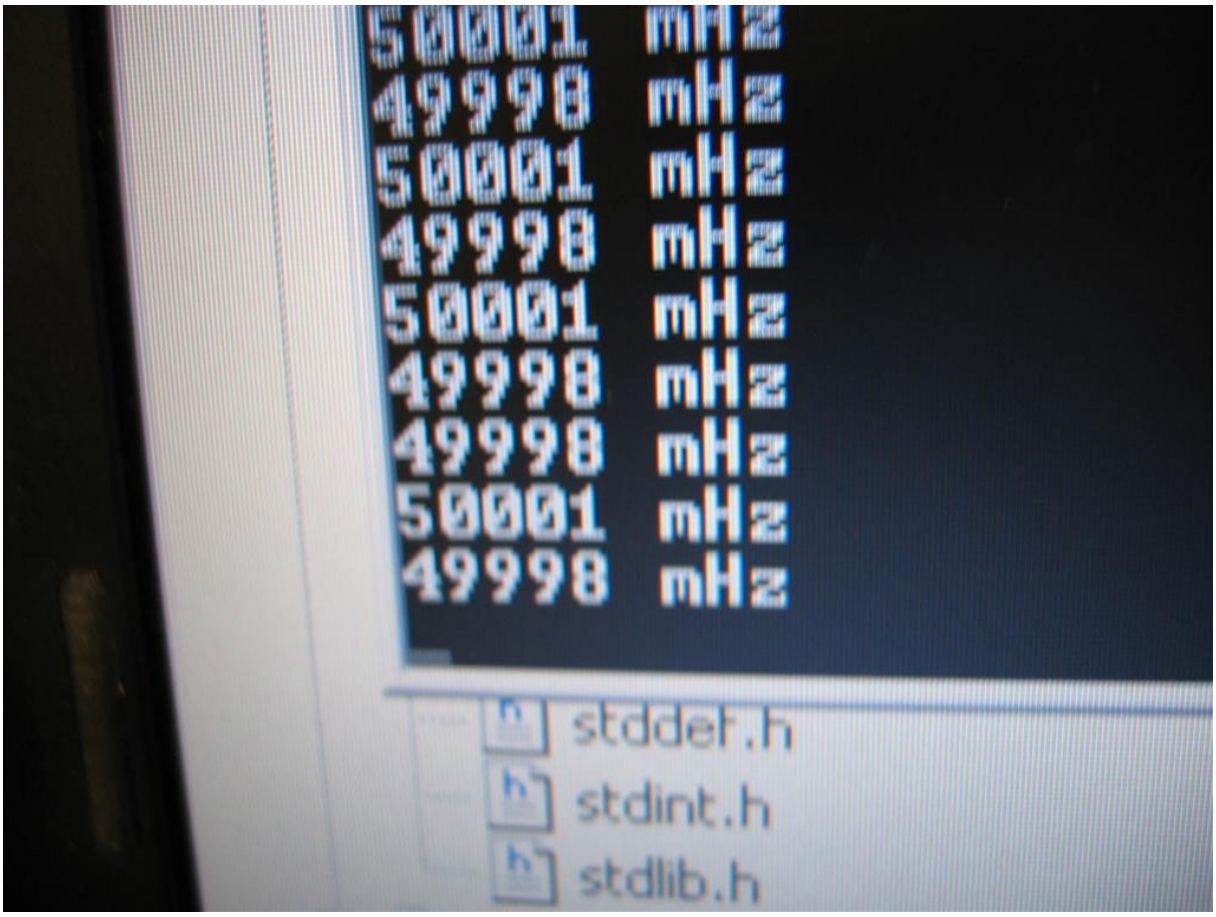


Abbildung 89: Frequenzmessung am Computer

Die Genauigkeit des ersten Testlaufs konnte durch den zweiten Versuch bestätigt werden. Was uns die Aussage treffen lässt, dass die projektierte Messgenauigkeit von +/-1 mHz erreicht wurde.

Projektmanagement Ergebnisse

1. Konnten wir unseren Zeitplan einhalten?
2. Wie Schätzen die Projektteilnehmer und Projektleiter ihre bzw. die Arbeit der anderen ein?

Paul:

...

Raoul:

...

Jürgen:

...

Eugen:

...

Martin:

...

Constantin:

...

Stefan:

...

4.6. Quellen:

[1] Tietze; Schenk: Halbleiter-Schaltungstechnik, Springer(12. Auflage, Erlangen und München, Januar 2002)

[2] http://www.krucker.ch/DiverseDok/Schmitt%20Trigger.pdf

[3] Quarzofen – Elektor

[4] Schmusch, Wolfgang: Elektronische Messtechnik, Vogel (5. Aufl., Würzburg, 2001)

[5]ATtiny2313Datasheet:http://www.atmel.com/dyn/resources/pdf_documents/doc2543.pdf

5. Gruppe 4

5.1. Aufgabe der Gruppe

Aufgabe der Anzeige, mit Effektivwertmessung, ist es die Messergebnisse der Frequenzgruppe und Peakdetektion darzustellen, sowie den Effektivwert. Dieser wird von der Anzeigegruppe selber ermittelt. Die Stromausfallzeit wird ebenfalls gemessen und dargestellt. Als Darstellung wurde ein Bargraph für den Effektivwert gewählt und mehrere 7-Segment-Anzeigen für die Peak- und Frequenzwerte, sowie die Ausfallzeit.

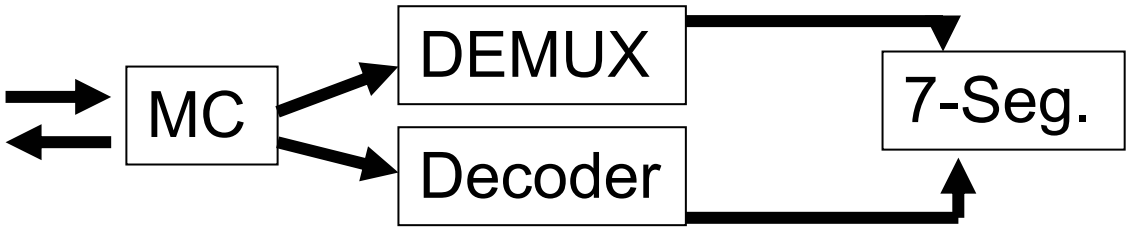


Abbildung 90 BSB Digitalschaltung und Anzeige

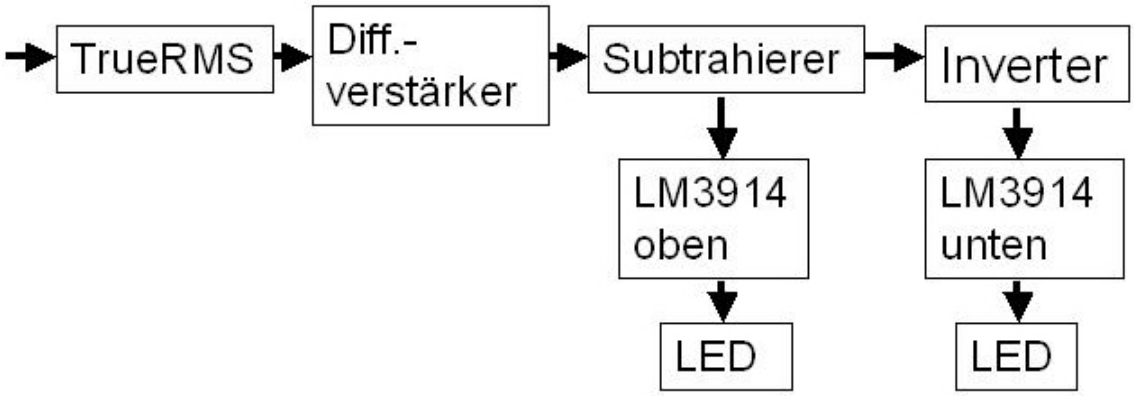


Abbildung 91 BSB Effektivwertschaltung und Anzeige

5.2. Digital-Schaltung - Elektronik

Aufgabe der Digitalelektronik

Diese Teilschaltung stellt die maximale minimale. und aktuelle Frequenz dar, sowie max./min Peakspannung und Ausfall- Überspannungszeit. Grund dieser Aufteilung ist, dass eine analoge Darstellung von Zahlen für diese Werte funktionell nicht möglich wäre.

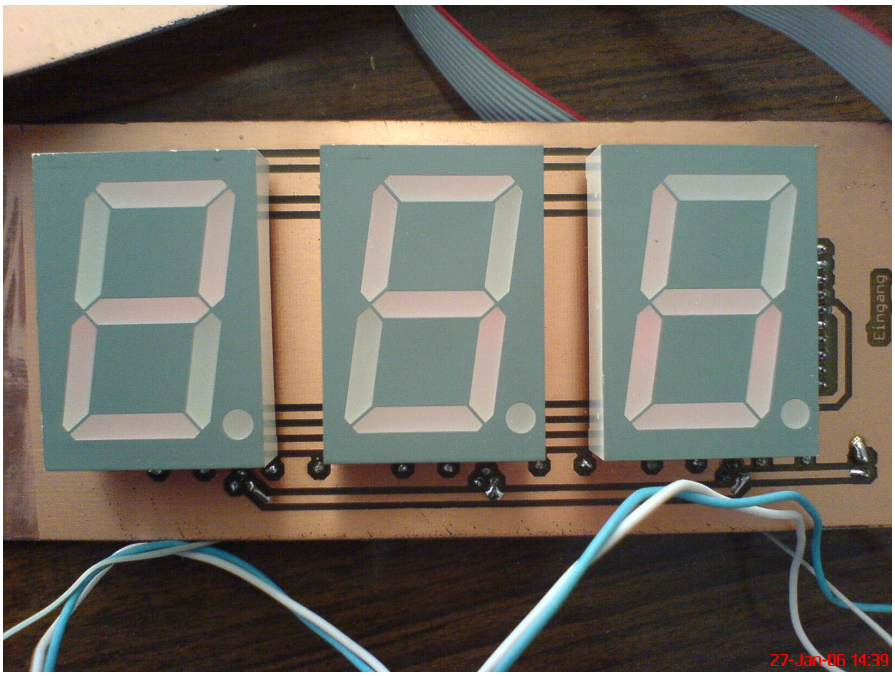


Abbildung 92 3er 7-SegmentPanel

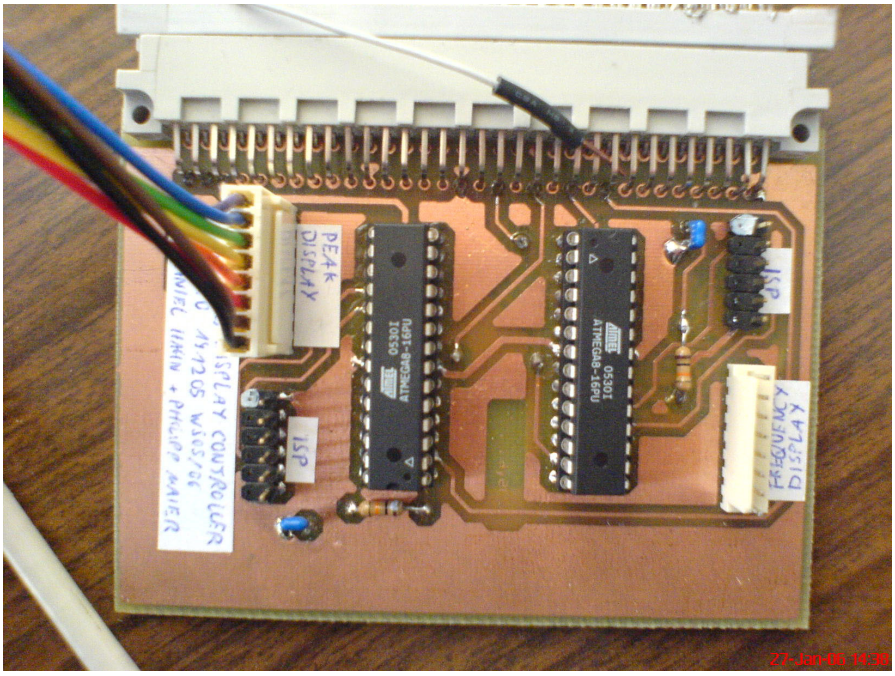


Abbildung 93 Controllerplatine

Prinzipielle Funktion der Schaltung

Eine Darstellung der Frequenz mit geringen Abweichungen oder die Darstellung der Ausfallzeit ist nicht mit einem Bargraphen oder anderen analogen Anzeigeelementen möglich, aber ließe sich sehr schön mit digitalen Zahlen darstellen.
Digitale Zahlen findet man z.B. in LCD-Displays oder auf herkömmlichen Monitoren (TFT), zu denen auch alle anderen XY-Matrixdisplays zählen.
Diese sind aber nicht für eine Darstellung im Sinne der Aufgabenstellung geeignet. LCD-Displays sind auf Entfernung nicht gut lesbar. Monitore wären nicht zweckmäßig für die eine solch einfache Anzeige.
Sehr bekannt sind die 7-Segmente. Sie haben eine hohe Leuchtstärke, da sie auf LEDs beruhen und können einzelne Ziffern darstellen.
Aus diesem Grund arbeitet die Schaltung mit 7-Segmenten.
Für eine reibungslose Anzeige sorgen Mikrocontroller (Atmel Mega8 AVR), sie sind gut geeignet um Daten zu empfangen und zu verarbeiten. Mit den verarbeiteten Daten können dann die TTLs 7447 und 74154 einfach angesteuert werde.
Beim TTL 74154 handelt es sich um ein Demultiplexer, der als Selektor betrieben wird.
Um die 7-Segmente anzusteuern, wird ein TTL 7447 Baustein verwendet. Dies ist ein BCD zu 7-Segment Decoder, welcher die Zahlen auf 7-Seg darstellt und digital bearbeitet. Das ist nötig, da diese in digitaler Form geliefert werden.
Theoretisch könnte man auf beide TTLs verzichten um ein paar 7-Segmente zum Leuchten zu bringen, doch da die Anzahl der 7-Segmente die Anzahl der Verfügbaren Pins am Mikrocontroller übersteigt ist dies nicht möglich.
Durch Hilfe der TTLs reduziert sich die Anzahl der benötigten Pins am Mikrocontroller erheblich und gleichzeitig wird eine Ansteuerung von vielen 7-Segmenten ermöglicht.
Da Peak und Frequenz viele Daten umfassen werden 2 Mikrocontroller mit prinzipiell gleicher Beschaltung für das jeweilige eingesetzt.

Zusammenfassung

Probleme	**Lösung**
Anzeige von Zahlwerten	Digitale Anzeige
Gute Lesbarkeit	7-Segmente mit LEDs
Empfang von zu verarbeitenden Daten	Mikrocontroller
Darstellung auf vielen 7-Seg.	DEMUX und Decoder zur Pinreduzierung

Tabelle 4 Zusammenfassung

Detaillierter Aufbau der Schaltung

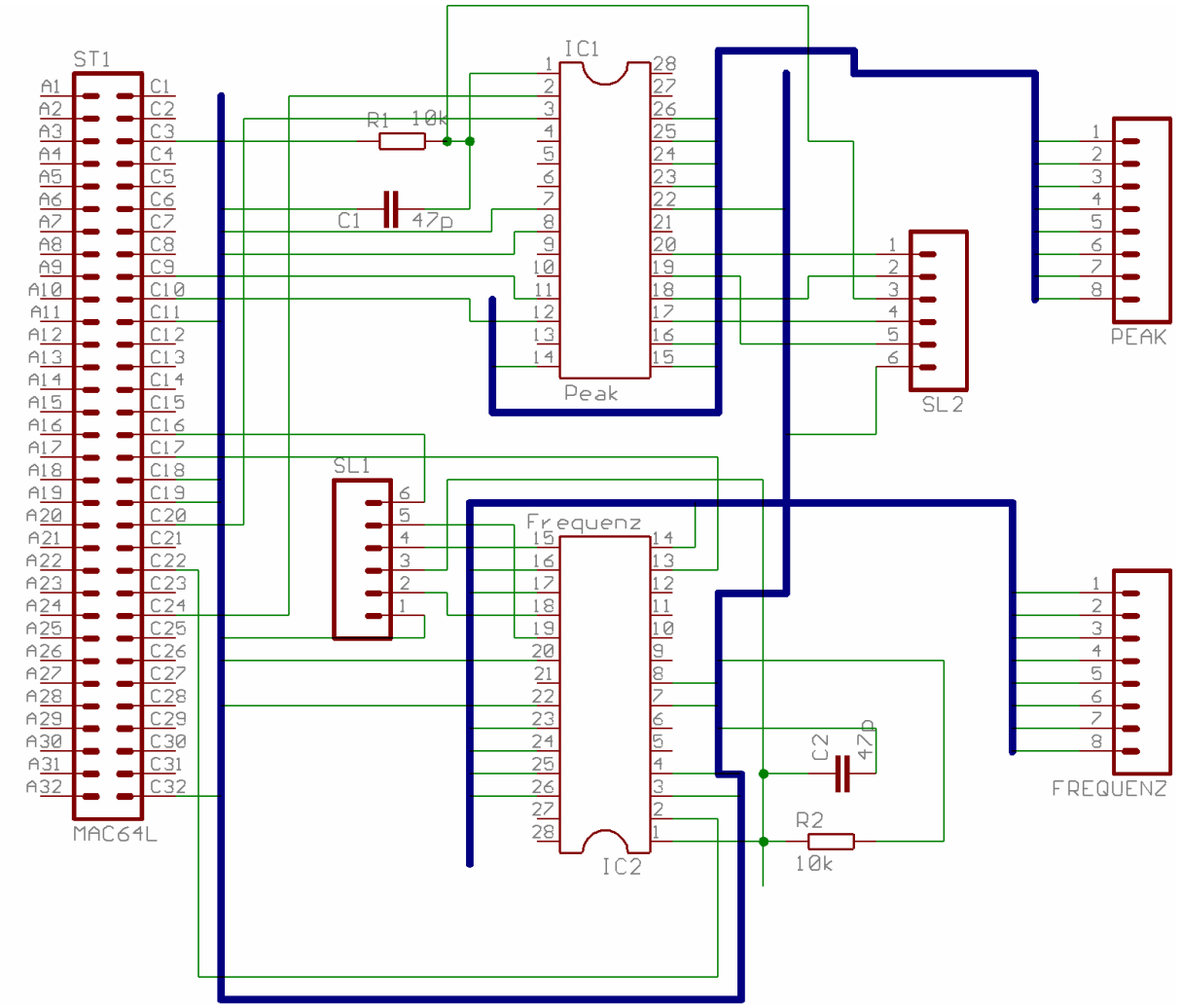


Abbildung 94 ControllerSchematic mit Bus

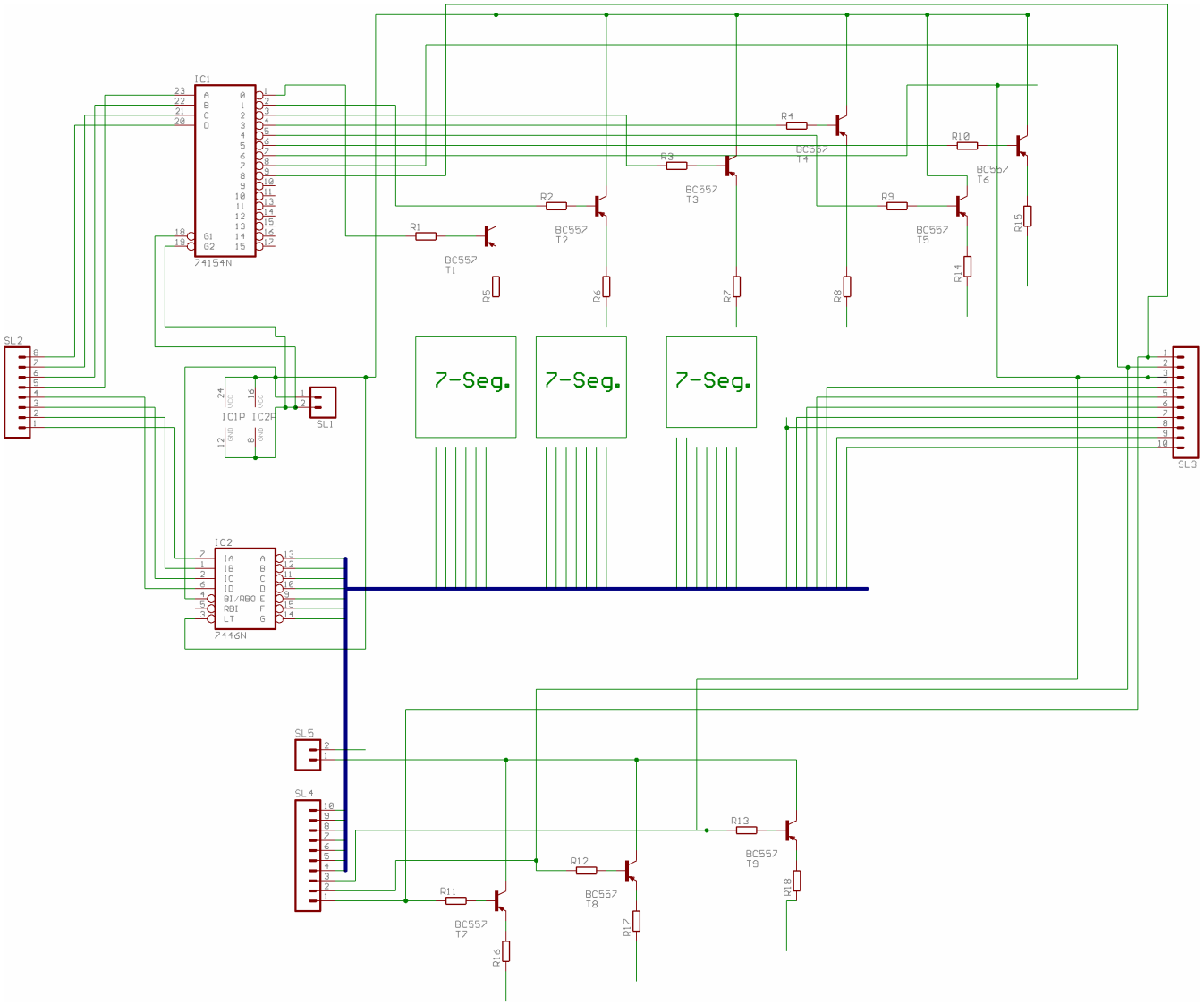


Abbildung 95 AnzeigePanel

Abbildung 94 ControllerSchematic mit BusAbbildung 94 und Abbildung 95 zeigen die gesamte Schaltung. Sie wird geteilt in eine Controllerschaltung, die im Bus steckt und in mehrere Panels für die 7-Segmentanzeige, beide Schaltungen sind durch Kabel verbunden.

Auf Grund der Unübersichtlichkeit wird im Folgenden, zu Erklärung der Funktion, auf das folgende Blockschaltbild zurückgegriffen.

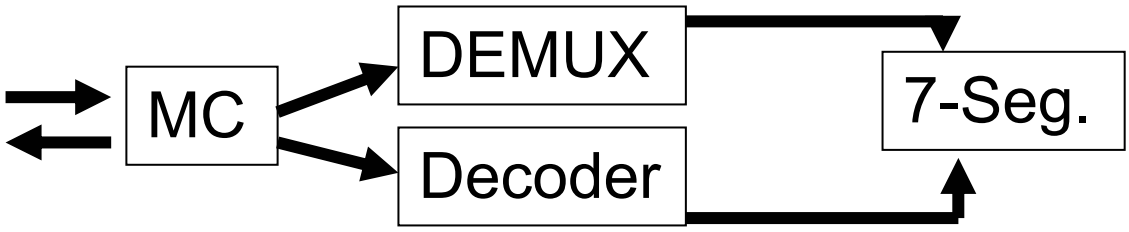


Abbildung 96 Blockschaltbild

Die nun erklärte Schaltung wird doppelt aufgebaut, einmal für die Anzeige der Frequenz und einmal für Peakspannung und Ausfallzeit.

Die Digitalanzeige hat digitale Eingangssignale zu verarbeiten. Dabei handelt es sich einmal um die maximale, minimale und aktuelle Frequenz und um die maximale und minimale. Peakspannung.
Für jeweils ein Signal ist ein Mikrocontroller zuständig, bei dem Peakcontroller kommen noch 2 Signale in Form von Pegeln für Überspannung und Unterspannung, die zum Zählen der Ausfallzeit genutzt werden.
Die Auswahl welcher Wert (maximale, minimale, aktuelle Frequenz, maximale, minimale. Peakspannung) angezeigt wird trifft die Anzeige selber, dazu gehen beim Frequenzcontroller 2 Leitungen an den Bus und beim Peakcontroller eine Leitung, da hier nur max./min ausgewählt werden muss.
Der Mikrocontroller verarbeitet dieses binäre Eingangssignal und ermittelt nun welche Zahl auf welchem 7-Segment angezeigt werden muss. Die anzuzeigende Zahl wird direkt an den Decoder als BCD Signal gesendet und die wievielte Anzeige diese Zahl anzeigen soll, wird als Binärcode an den DEMUX gesendet.
Für eine z.B. 6-stellige Zahl muss der Mikrocontroller die vorderste Ziffer der Zahl an das erste 7-Segment senden, die 2. Ziffer an das 2. Segment und so weiter. Da immer nur eine Zahl dargestellt wird geschieht das entsprechend schnell, so dass alle Ziffern zusammen als eine komplette Zahl wahrgenommen werden.
Der Decoder sendet seine Zahl immer an alle 7-Segmente, der DEMUX versorgt aber nur ein einziges Segment mit Strom, er ist sozusagen ein An/Aus-Schalter.
Um die 7-Segmente in dem kurzen Moment in dem sie aktiv sind mit genügend Strom zu versorgen, so das sie hell leuchten, befindet sich zwischen jedem Ausgang des DEMUX und dem 7-Segment eine Treiberschaltung. Sie besteht aus einem pnp-Transistor und 2 Widerständen. Wenn die Basis des Transistors durch den DEMUX auf Masse gezogen wird, schaltet der Transistor durch und das 7-Segment wird mit Strom versorgt. Die Widerstände begrenzen dabei den Strom.

Probleme

Problematisch an dieser Schaltung ist die Pulsweitenmodulation, die durch die kurze Ansteuerung der 7-Segmente entsteht. Um eine sichtbare, hell leuchtende Anzeige zu erhalten, muss man wesentlich größere Ströme in die 7-Segmente treiben, als im normalen Dauerbetrieb nötig wären. Diese Ströme sind leider nicht mittels Datenblatt rechnerisch zu ermitteln und müssen praktisch ausgetestet werden. Hier empfehlen sich Potis zum Testen.

Hinweise zur Inbetriebnahme

Die Widerstände auf dem Anzeigepanel müssen nach eigenem Gebrauch dimensioniert werden. Generell können die Widerstände zwischen Transistor und DEMUX sehr klein gewählt werden (1 Ohm). Die Widerstände zwischen Transistor und 7-Segment bestimmen die Leuchtstärke, man könnte, um die maximale Helligkeit zu erreichen, auf sie verzichten.

Bestückungsplan

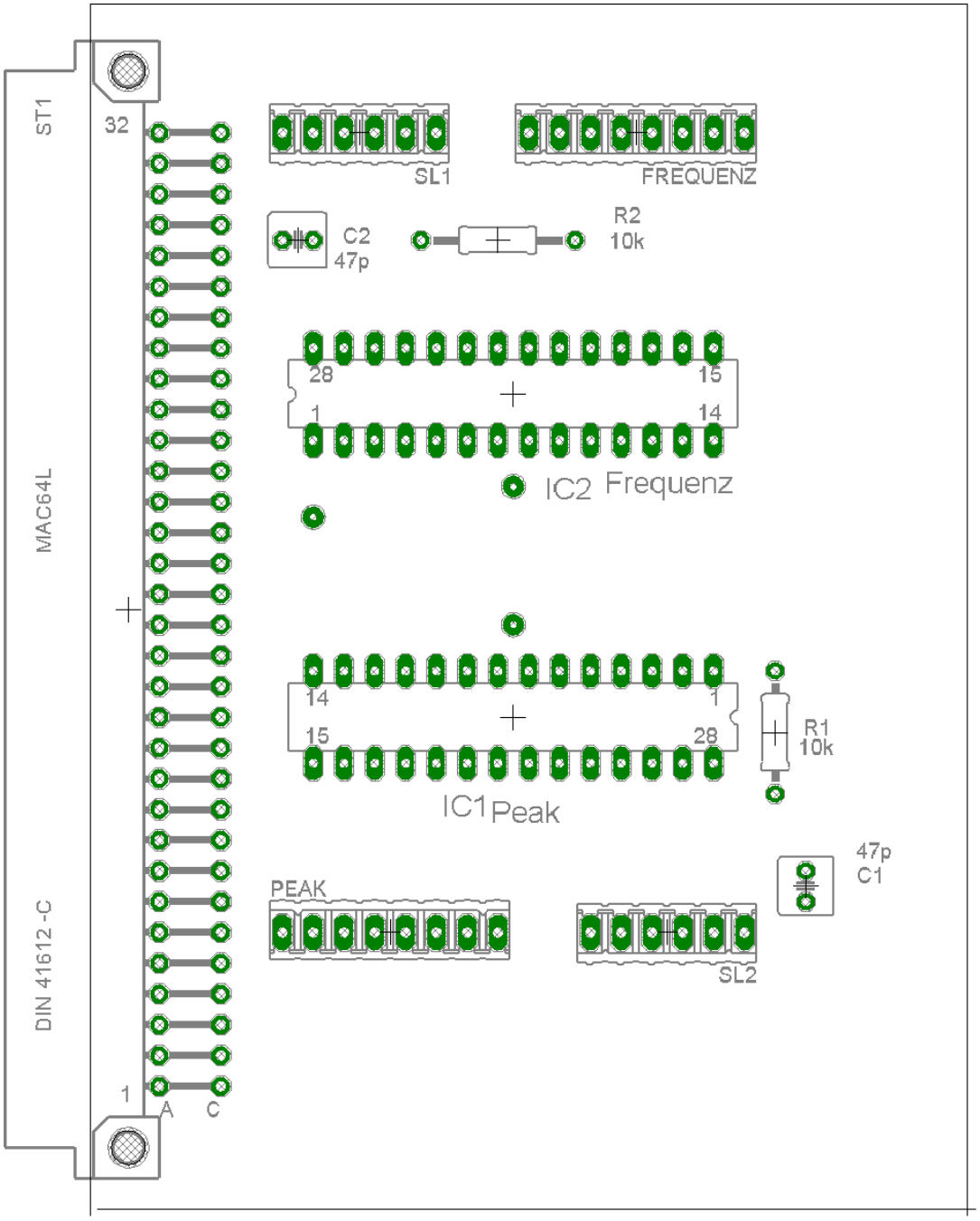


Abbildung 97 Bestückungsplan Digital-Controller

Bauteilliste Digital-Controller

- 2x AtmelMega8 Mikrocontroller
- 2x 10 kOhm
- 2x 47 pF Kondensatoren
- 2x 28er IC-Fassungen
- 2x 8er Busstecker
- 2x 6er Busstecker

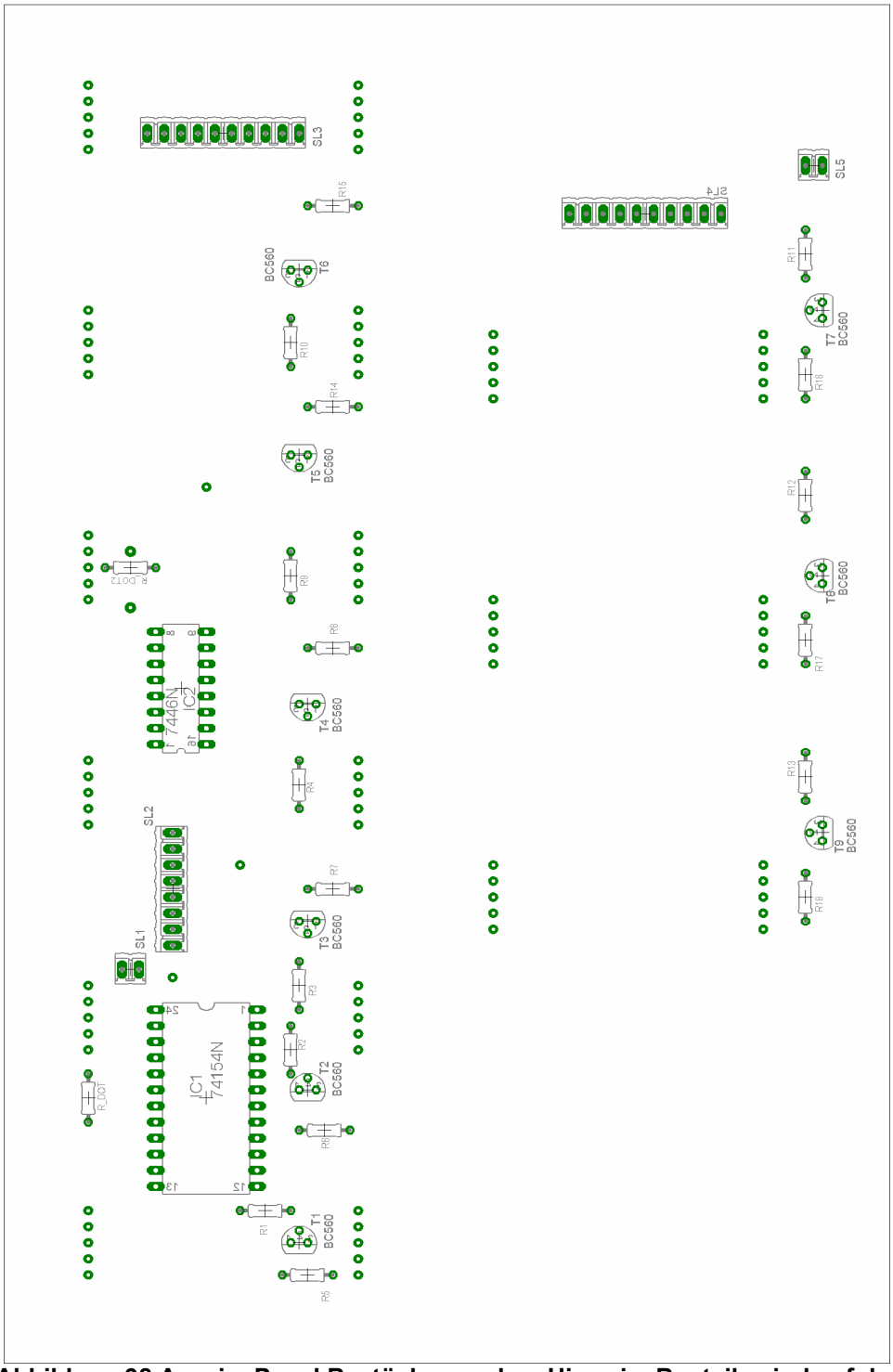


Abbildung 98 AnzeigePanel Bestückungsplan, Hinweis: Bauteile sind auf dem Bottom

Bauteilliste AnzeigePanel

- 2x IC 74154 Demultiplexer ,da obere Schaltungshälfte 2x gebaut wird
- 2x IC 7447 Decoder ,siehe Demultiplexer
- 15-16x BC560
  wenn man sich für 16 anzeigen entscheidet sind 16 nötig, normal 15
- 15-16x niederohmige Widerstände z.B. 1 Ohm
- 15-16x weitere Widerstände , zu dimensionieren
- 2x 8er Stecker , z.B. aus Sillsteifen bauen
- 2x 10er Stecker ,für Verbindung vom 6er Panel zum 3er Panel ,nur 1x
- 2x 1 – 1,2 kOhm Widerstände für R_DOT und R_DOT2
- 16x 38mm 7-Segmente, z.B. SC15-11EWA

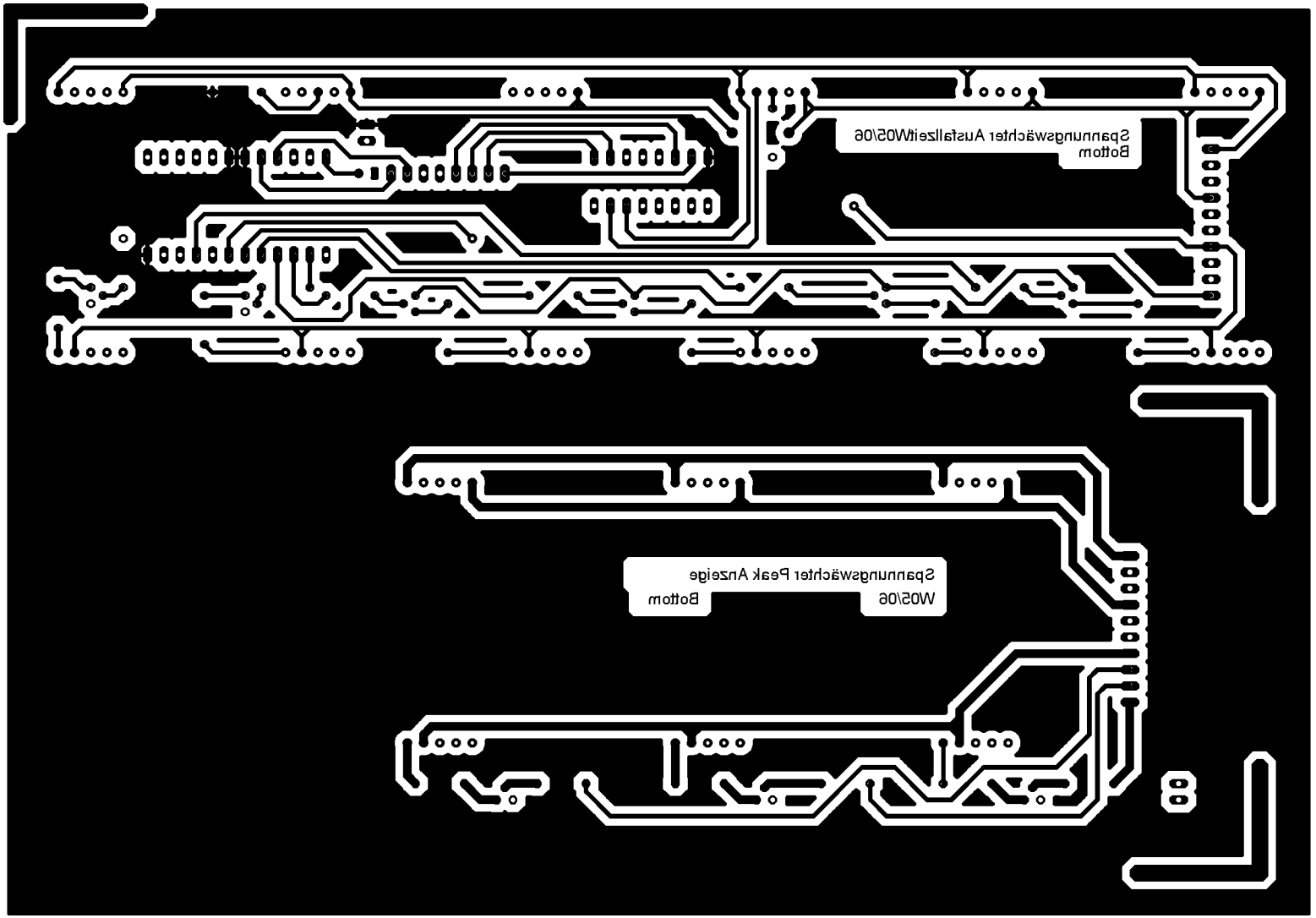


Abbildung 99 AnzeigePanel Bottom, bestehend aus 2 Teilen, ein 6er Segment, 2x zu bauen, ein 3er Segment, 1x zu bauen

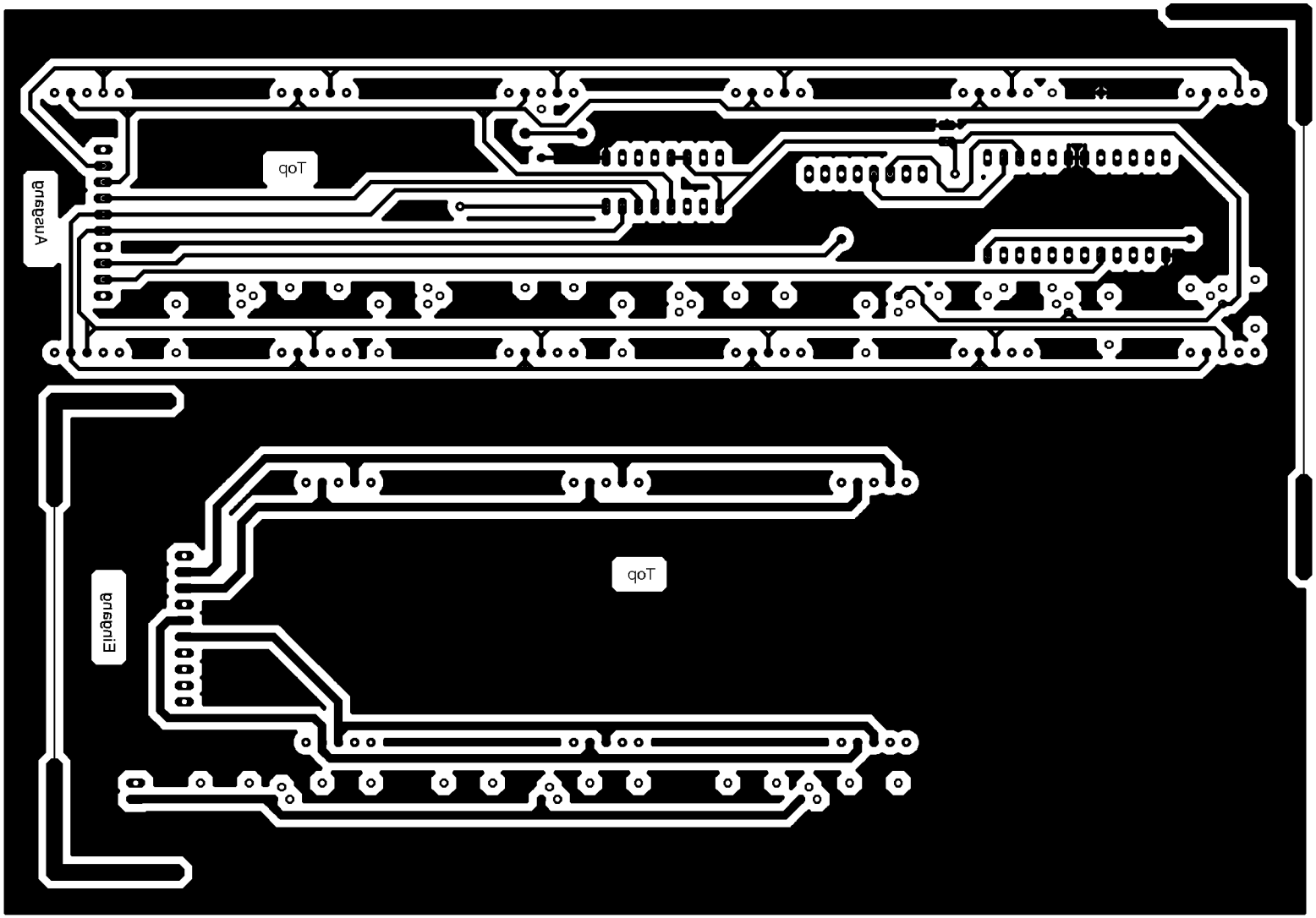


Abbildung 100 AnzeigePanel Top

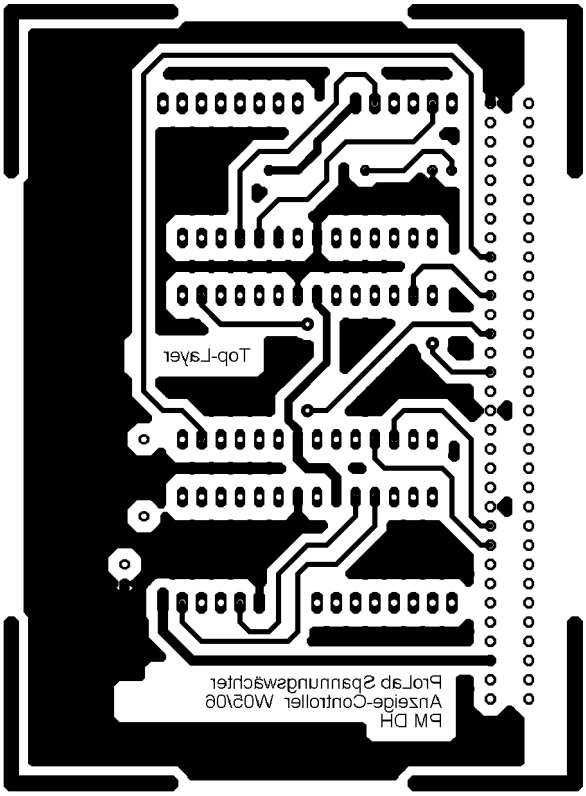


Abbildung 101 Digital-Controller Bottom

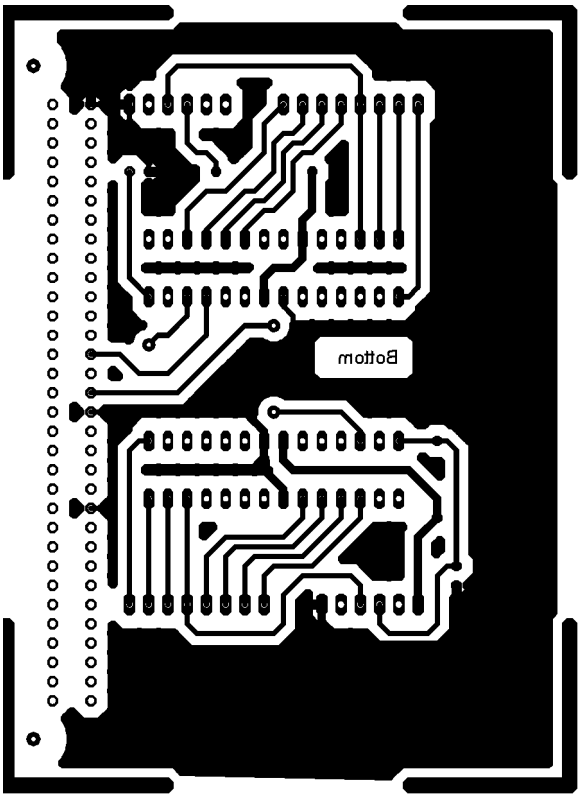


Abbildung 102 Digital-Controller Top

5.3. Digital-Schaltung-Programmcode

Aufgabe des Programmcodes

Es soll ein Anzeigepanel zur Messwertanzeige entworfen werden, welches Messwerte von einem Messgerät vollautomatisch anfordert und darstellt. Um Busleitungen zu sparen soll die Übertragung des eigentlichen Messwertes seriell erfolgen (UART bei 5V Pegel). Alle restlichen Steuerleitungen werden der Einfachheit parallel gestaltet.
Im konkreten Fall existiert im Geräteverbund ein Messgerät für Frequenz und eines für Peaks. Beide Messgeräte liefern verschiedene Messwerte. So misst das Frequenzmessgerät den maximalen, den minimalen und den aktuellen Frequenzwert. Das Peak Messgerät misst den maximalen und den minimalen Netzspannungswert. Zudem informiert es mittels zwei Steuerleitungen über einen Stromausfall bzw. über eine Spannungsspitze. Tritt eines der beiden letzteren Ereignisse ein wird eine Uhr gestartet und die Dauer der Störung festgehalten.
Dem Anzeigesystem soll alle 24h ein Resetsignal ausgeben werden um die Messgeräte zurückzusetzen.

Folgende Tabelle gibt eine Übersicht über die Anzeige und ihre Steuerleitungen die an die Messgeräte angeschlossen werden.

Anschlüse	Richtung	Beschreibung:
PD0 (UART)	Eingang	Serielle Messwertübetragungsleitung)*
PD1 (CTRLA) PD2 (CTRLB)	Ausgang	2Bit Parallell Leitung zur Anforderung des jeweiligen Messwertes (10 := Max, 01:= Min, 11:= Aktuell)
PD7 (AUXRESET)	Ausgang	Auxreset

Tabelle 5 Frequenzmessgerät

Anschlüsse	Richtung	Beschreibung:
PD0 (UART)	Eingang	Serielle Messwertübetragungsleitung**
PD1 (CTRLA)	Ausgang	2Bit Parallel Leitung zur Anforderung des jeweiligen Messwertes (0 := Min, 1:= Max)
PD7 (AUXRESET)	Ausgang	Auxreset (nicht angeschlossen)
PD6 (PMSGB)	Eingang	1Bit Parallel Leitung zur Meldung von Stromausfall. (0=Ausfall)
PD5 (PMSGA)	Eingang	1Bit Parallel Leitung zur Meldung von Überspannung. (1=Überspannung)

Tabelle 6 Peakmessgerät

*Die Übertragung erfolgt mittels eines besonderen Kommunikationsprotokolles auf Seite 155.

**Die Übertragung erfolgt mittels eines besonderen Kommunikationsprotokolles auf Seite 155, der übermittelte Wert entspricht $\frac{1}{2} * Messwert$, muss also zunächst einmal mit sich selbst addiert werden um den tatsächlichen Messwert zu erhalten.

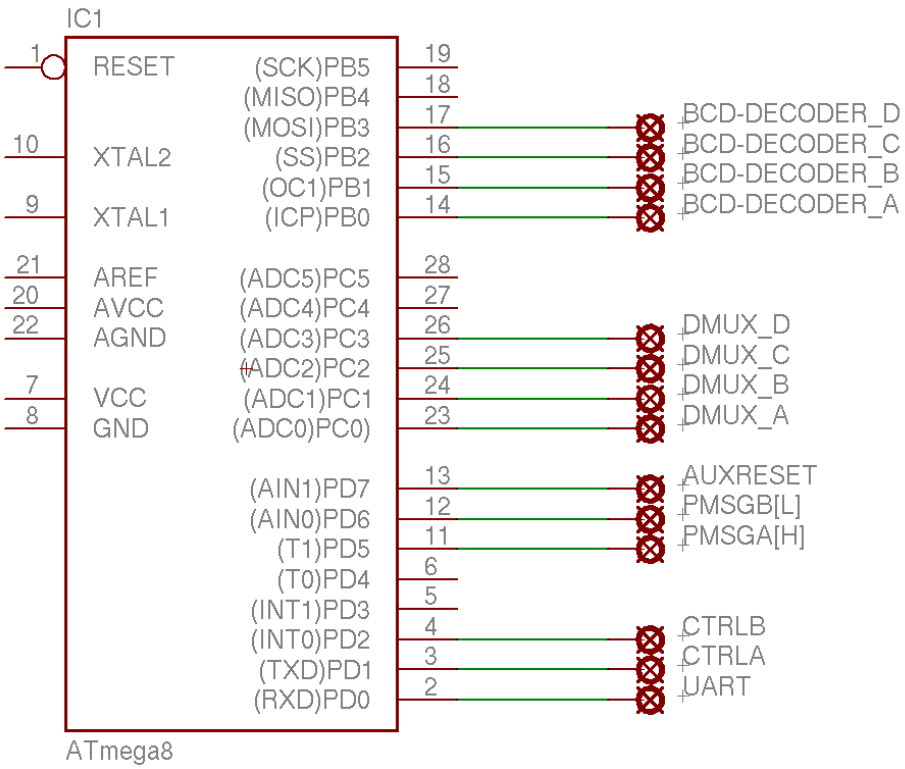


Abbildung 103 Anschlussübersicht auf dem Controller

Hardwarekonfiguration

Anzeige

Da nur Zahlen angezeigt werden müssen, werden 7-Segmente im Multiplexbetrieb verwendet. Die Software ist für die Ansteuerung von einem SN7447N als BCD Decoder an PD0-PD3 und einem SN74154 Demultiplexer an PC0-PC3 programmiert. Die genaue Beschaltung ist der Hardwarebeschreibung auf Seite 137 zu entnehmen. Die Software kann maximal 9 Elemente verarbeiten.

Funktionsbeschreibung

Das Multiplexingverfahren basiert auf der Idee jeweils eine 7 Segmentanzeige zyklisch und nur für Sekundenbruchteile einzuschalten. Während des Einschaltvorgangs wird für die entsprechende Anzeige der entsprechende BCD-Code (4-Bit Zahl) angelegt. Da da der Vorgang sehr schnell abläuft entsteht der Eindruck das alle Anzeigelemente dauerhaft eingeschaltet sind. Folgendes Bild illustriert das Prinzip:

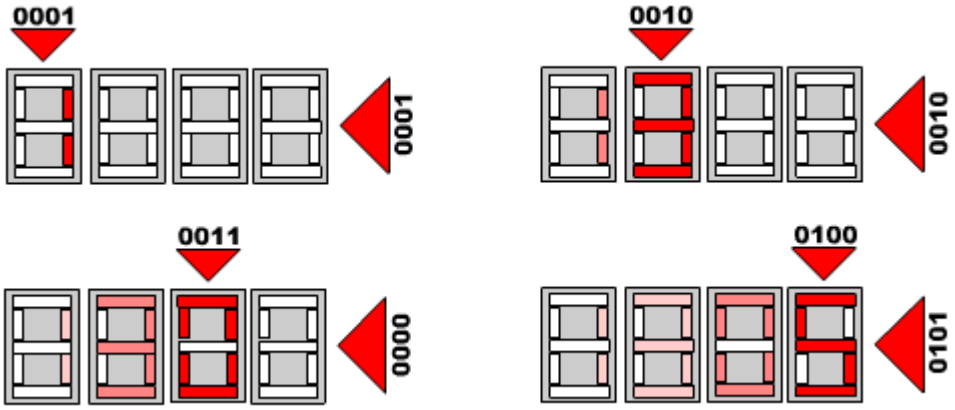


Abbildung 104 Die senkrecht stehenden Binärzahlen stellen den Bus für die Digits dar. Die waagrecht stehenden Binärzahlen geben an welches Segment gerade ausgewählt ist.

Softwarefunktionsweise

Steuerung

Für die Steuerung werden zwei unabhängige ATmega8 AVR Mikrocontroller verwendet. Die Software wird vollständig in Assembler verfasst. Zum Programmieren des Controllers wurde das freie Programm AVRDude verwendet. Assembliert wurde mit dem freien TAVRASM.

Beschreibung der Routinen:

Das System besteht aus einer Vielzahl von Unterprogrammen, die Teilweise kaskadenartig ausgeführt werden. Die Routinen sind alle so gestaltet, dass sie, sofern sie in einem bestimmten Betriebsmodus nicht benötigt werden, nicht ausgeführt werden. Das bedeutet das selbst wenn die Routine z.B. im Kernprogramm aktiviert ist, aber der Modus nicht zu ihr passt, sie trotzdem nicht ausgeführt wird.

Systemroutinen:

boot:	Initalisierung des Systems.
reboot:	Neustart erzwingen
halt:	System anhalten (wird im Normalbetrieb nicht verwendet)
atr:	Einschaltmeldung ausgeben
delay:	Verzögern, Grad der Verzögerung muss vorher in regParam fesgelegt werden
core:	Kernprogramm (siehe Zentrale Steuerrungsroutinen)
slow:	Timerinterrupt (siehe Zentrale Steuerrungsroutinen)
chrond:	Zeibasiskontrolle für *chron* (siehe Zentrale Steuerrungsroutinen)
chron:	Zyklisch auszuführende Routinen (siehe Zentrale Steuerrungsroutinen)

Displayroutinen:

subClearDisplay:	Display löschen (Dunkle Anzeige)
subDelayToDisplay:	Meldung ausgeben und warten. Grad der Verzögerung muss vorher in *regParam* fesgelegt werden
subDrawDisplay:	Anzeige neu aufbauen

Kontrollroutinen

subRunSwitchInput:	Steuerleitungen zum Messgerät verwalten (Zyklisches Umschalten) Achtung: Führt weitere eigene Unterroutinen automatisch aus.
subRunPeakClock:	Führt Peak Uhr aus. Achtung: Führt weitere eigene Unterroutinen automatisch aus.
SubTellAuxresetHigh:	Auxreset auf High ziehen
subTellAuxresetLow:	Auxreset auf Low ziehen

Routinenaufrufe

Core:

Core ist die erste Routine die nach dem Booten aufgerufen wird. In ihr werden die Aufrufe auf der Routinen eingetragen die bei jedem Progammzyklus einmal ausgeführt werden sollen:

subDrawDisplay:	*Auffrischen der Anzeige*
SubBin2toBCD:	Umrechnen der empfangenen Messwerte in für das Display verständliche BCD Symbole.
s*ubUpdateValues*:	Daten im Anzeigenspeicher aktualisieren.
subRunPeakClock:	Ausfallzeiten zählen (wenn erforderlich)

Slow:

Slow ist die Routine die duch den Timerinerrupt aufgerufen wird. Hier finden sich alle zeitkritischen Prozesse:

subTellAuxresetLow:	*Auxreset* Leitung wieder auf Low schalten, falls sie irgendwo im Progamm auf High gesetzt wurde.
s*ubRunSwitchInpu*t:	Umschalten der Steuerleitungen für den Messwert.
c*hrond*:	Steuerrungsroutine für *chron* ausführen. Achtung: *Chron* wird deaktiviert wenn diese Routine abgeschaltet wird.

Chron:

Chron wird je nach Einstellung der Zeitbasis zyklisch abgefahren. Hier stehen folgende Aufrufe:

subTellAuxresetHigh:	*Auxreset* auf High ziehen
reboot:	System neustarten (ist standardmäßig ausgeschaltet)

Konfiguration

Obwohl in der Schaltung zwei unabhängige Mikrocontroller arbeiten verwenden beide dieselbe Software. Sie unterscheiden sich lediglich durch ihre Konfiguration. Die Konfiguration ist in Dateien peak.conf und frequency.conf abgelegt. Vor dem übersetzen des Programms muss die gewünschte Konfigurationdatei eingebunden werden.

In beiden Konfigurationsdateien befinden sich folgende Felder. Zu beachten ist hier, das je nach Modi nicht jedes Feld tatsächlich zur Anwendung kommen kann. Es muss aber trotzdem vorhanden sein, weil es sonst beim Übersetzen zu Fehlern kommt.

.equ SYSTEM_MODE = 0x01
Betriebsart: 0x01 für Betrieb als Frequenzanzeige, 0x02 für Betrieb als Peakanzeige.

.equ SYSTEM_UART_ENABLE = 0x01
Serieller Datenempfang: Mit 0x01 ist der Datenempfang aktiviert, 0x02 deaktiviert den Datenempfang (z.B.: für Fehlersuche)

.equ SYSTEM_CHRONJOB_TIMING = 0x03
Zeitbasis für *chron*

.equ SWITCH_TIMING = 0x01
Zeitbasis für das Umschalten des Messwertes.

.equ DISPLAY_TIMING = 0x15
Zeitbasis für Multiplexing der Anzeige

.equ DISPLAY_DIGITS = 0x08
Anzahl der Anzeigeelemente. Kann auf 8 gelassen werden auch wenn weniger Anzeigeelemente angeschlossen sind.

.equ PEAK_CLOCK_TIMEBASE = 0x40
Zeitbasis für die Peak Anzeigeuhr.

Die Softwareversionsnummer ist im Kopf des Programms fest eingestellt:
.equ SYSTEM_VERSION_MAJOR = 0x00
.equ SYSTEM_VERSION_MINOR = 0x09
.equ SYSTEM_VERSION_REVISION = 0x00

Hinweise zur Inbetriebnahme

Sind mit Hilfe der Debugroutinen alle Hardwarefehler beseitigt und nachgewiesen, dass die Schaltung korrekt arbeitet, kann das System in Betrieb genommen werden.

Hierzu sollten alle Routinen in *core*, *slow* und *chron* ausgeschaltet werden und dann Schritt für Schritt aktiviert werden. Nach Einschalten der Versorgunsspannung wird auf den ersten 4 Elementen das Wort „boot“ angezeigt. Danach erscheint ein Fortschrittsbalken, welcher alle Segmente nacheinander einschaltet. Dann wird die Versionsnummer der Software und kurz darauf ihr Modus ausgegeben. (1= Frequenzanzeige, 2=Peakanzeige). Ist dies geschehen nimmt das System die Arbeit auf.

Debugroutinen

Um bei der Entwicklung auftretende Hardwarefehler zu finden wurde ein Satz Debugroutinen entwickelt. Diese Routinen sind im Normalbetrieb ausgeschaltet. Bevor sie verwendet werden dürfen, müssen in jedem Fall vorher alle Routinen in CORE, SLOW und CHRON ausgeschaltet werden. Der Aufruf der Routinen erfolgt immer über CORE.

subDisplayTest: Führt einen vollständigen Displaytest durch. Hierbei werden in langsamen Tempo alle Elemente einzeln und immer mit verschiedenen Zahlen angesteuert. Die Routine eignet sich deshalb besonders gut zum Finden von kalten Lötstellen im Anzeigepanel. Achtung: Eventuell muss die Betriebsspannung der Anzeige heruntergeregelt werden, da die Anzeigen hier eine längere Zeit eingeschaltet bleiben und sich das Verhältnis Strom pro Zeiteinheit verändert.

subLampTest: Zeigt auf allen Segmenten eine 8 an. So kann nachgewiesen werden das alle Segmente in Ordnung sind. Die Routine eignet sich jedoch schlecht zum Überprüfen der elektrischen Verbindungen, da mit ihr ein Kurzschluss unter Umständen nicht erkannt wird.

subSpecificLampTest: Stellt sich heraus das einige Segmente keine oder falsche Zeichen anzeigen ist es sinnvoll eben dieses Segment mit einer bestimmten Zahl einzuschalten um in aller Ruhe Messungen vornehmen zu können. (Achtung: Hierzu muss die Routine direkt verändert werden.)

subShowSerialProtocolOutput: Gibt die über das UART an PD0 empfangenen Messdaten (16BIT) in Form von ihrer Halbbytes auf den ersten 4 Elementen als Zahlen aus.
(Hinweis: 1111 entspricht in BCD dem Löschzeichen, die Anzeige bleibt dann dunkel.)

Probleme

Zuerst wurde die für den Betrieb notwendige Hardware zusammengestellt. Hierbei stellte sich schnell heraus, dass aufgrund der eng begrenzt verfügbaren Busleitungen und auf Grund der Tatsache, dass das Frequenzmessgerät ohnehin schon mit einem UART fähigen Mikrocontroller versehen ist, erschien eine serielle Datenübertragung am praktikabelsten.

Da sich die für das Peak-Messgerät zuständige Gruppe nicht dazu bewegen lies auch einen Mikrocontroller einzusetzen, der auch hier ein serielles Signal hätte bereitstellen können wurde unsererseits ein Mikrocontroller so programmiert, dass er 16 Signalleitungen einlesen und per UART verschicken konnte. Im Zuge dieser Nebenentwicklung (siehe Seite 155) wurde gleichzeitig ein Kommunikationsprotokoll (siehe Anhang1) entwickelt welches den Transport der Messwerte mit brauchbarer Sicherheit übertragen kann. Das Protokoll wurde auch im Frequenzmessgerät implementiert und stellt einen projektinternen Standard dar.

Aufgrund von Lieferschwierigkeiten bei den Anzeigelementen wurde die Entwicklung der Betriebssoftware für die Anzeige stark verzögert, konnte aber noch rechtzeitig beendet werden. Die Software wurde aus Gründen der Effizienz so programmiert, dass sie sowohl für das Frequenzanzeigepanel, als auch für das Peakanzeigepanel verwendet werden kann. Es mussten lediglich einige Konfigurationsparameter angepasst werden.

Die Programmierung verlief weitestgehend ohne gravierende Schwierigkeiten. Die Konvertierung von 16 Bit Binärcode in für BCD-Anzeigen verarbeitbare 4-Bit Zahlen gestaltete sich anfangs schwierig, funktionierte nach Behebung der Fehler auf Anhieb.

Kommunikationsprotokoll:

Bei dem Transfer der Daten kommt ein einfaches Kommunikationsprotokoll zum Einsatz. Ein Uart- Frame kann maximal 8 Datenbits enthalten. Das reicht für 16 Bit natürlich nicht aus. Es könnten zwar 2 mal 8 Bit übertragen werden, man wüsste dann auf der Empfängerstelle aber nicht genau welches der beiden Halbbytes das obere und welches das untere ist. Um die Konsistenz der Daten weiterhin zu gewährleisten müssen die einzelnen Datenpackete durchnummeriert werden. Deshalb werden ein paar Datenbits für eine Sequenznummer geopfert. Der Einfachheit halber verwendet unser IC die oberen 4 Datenbits eines Frames als Sequenznummer und die unteren 4 Datenbits eines Frames als Datenbits. Mit der Sequenznummer nummerieren wir unsere Teilsequenzen einfach durch, damit wir sie an der Empfängerstelle hinterher wieder in der richtigen Reihenfolge zusammensetzen können. Zur Sicherheit wurde festgelegt das es sich bei der Sequenznummer um eine von 0 verschiedene Zahl handeln muss, genauer: Die Menge der möglichen Sequenznummern ist {1,2,3,4} bzw. {0001,0010,0011,0100}.

Beispiel:
Nehmen wir an wir wollten die 16 Bit Zahl 1010111100001100 übertragen:

Zunächst zerhacken wir die Datenbits in 4 kleine 4 Bit Sequenzen:

1010
1111
0000
1100

Dann fügen wir eine bei 0001 loslaufende Sequenznummer hinzu:

00011010
00101111
00110000
01001100

Jetzt können die Daten problemlos über eine Uart Schnittstelle verschicket werden. Egal wie die Daten ankommen, die Gegenstelle wird immer wissen welches Packet welche Teilsequenz enthält.

Prolab Serial Transfer

„ProlabSerialTransfer“ ist ein mit Hilfe des Atmega8 Mikrocontrollers realisierter Parallell zu Seriell Wandler auf Basis des ATmega8. Er dient dazu ein paralleles 16 Bit Signal in ein Serielles UART Signal umzuwandeln. Bei der Entwicklung des Bauteils wurde sehr großen Wert auf einfache Verwendung gelegt. So wird z.B. zum erzeugen der Taktrate kein externer Quarz verwendet. Stattdessen kommt ein IC interner RC Oszillator zum Einsatz. Dieser ist zwar nicht genau, jedoch genau genug um eine Baudrate von 2400 Baud sicher generieren zu können. Desweiteren ist es nicht nötig das angelegte 16 Bit Byte in irgendeiner Form zu bestätigen. Der Controller sendet einfach fortwährend, so schnell er kann über die serielle Leitung. Aus diesem Grund eignet sich dieser IC nur zum Übertragen von Signalen die sehr viel langsamer getaktet sind als der IC sie einlesen kann. Schnelle Signale würden nur bruchstückhaft übertragen werden.

5.4. Effektivwert

Aufgabe der TrueRMS-Schaltung

Aufgabe war den Effektivwert einer Spannung zu bestimmen.
Um den Effektivwert mathematisch korrekt und präzise erfassen zu können, sollte möglichst eine True-RMS (Root-Mean-Square)-Messung erfolgen, da andere Möglichkeiten nicht genau genug sind. Der Effektivwert berechnet sich nach folgender Formel:

$$s_{\text{eff}} = \sqrt{\frac{1}{T}\int_0^T s^2(t)dt}$$

Formel 2 TrueRMS

S stellt ein reellwertiges Signal dar und ist in diesem Fall der Spannungsverlauf über der Zeit.
Es muss also eine Schaltung entworfen werden, die über analoge OPV-Rechenschaltungen diese Formel löst und aus einem beliebigen Wechselsignal den Effektivwert in Form von einer Gleichspannung ausgibt.

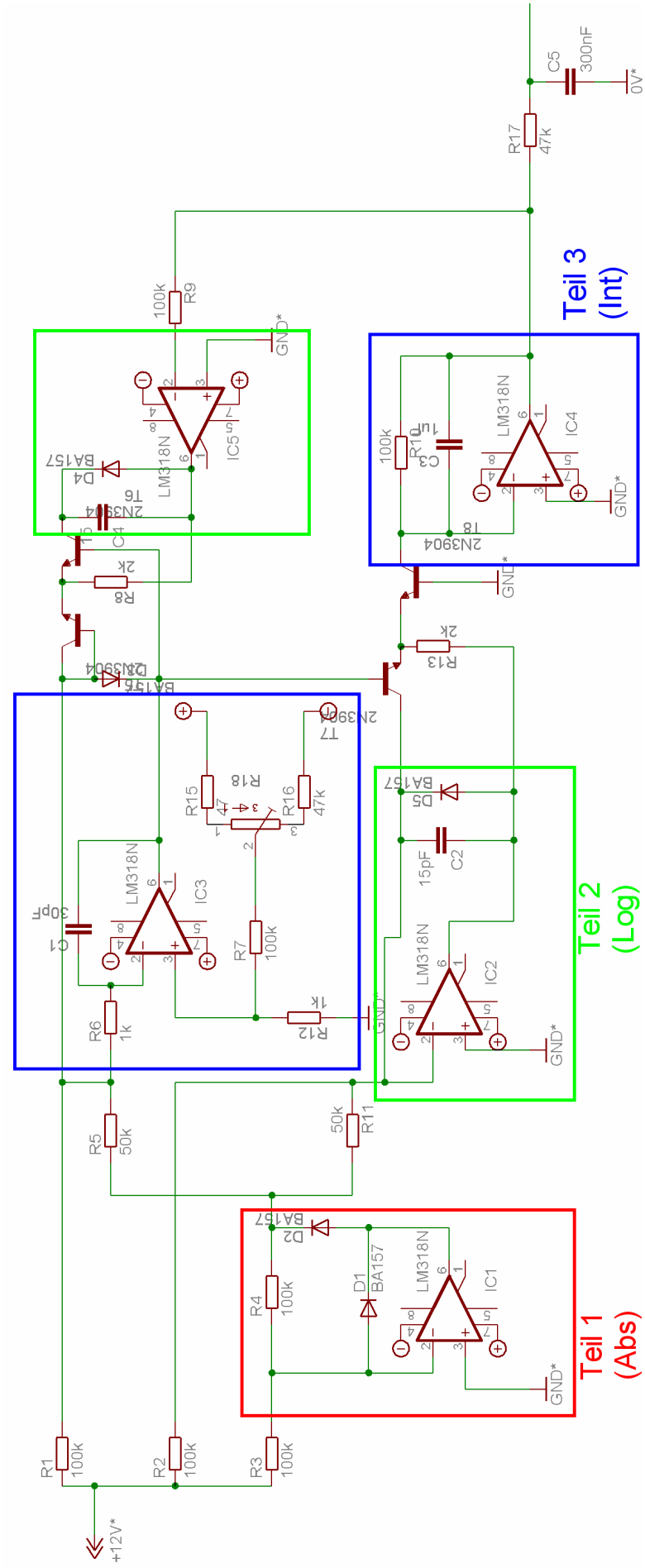


Abbildung 105 TrueRMS-Schaltung

Funktionsweise

Die Schaltung liefert den mathematisch korrekten Wert des Effektivwertes eines beliebig geformten Signals als Gleichspannungssignal. Der erste Stromkreis (Teil 1) dient der Absolutwertberechnung. Die beiden grün eingerahmten Stromkreise (Teil 2) dienen der Integration des Signals, die beiden blauen Kreise in Verbindung mit den Transistoren funktionieren als logarithmische Multiplizierer/Dividierer, mit denen das Signal potenziert und richtig zusammengesetzt wird.

Probleme

Das größte Problem ist der Aufbau der Schaltung und eventuelle Fehlersuche, Testschaltungen sollten sehr sorgfältig aufgebaut werden, da sie sehr schnell unübersichtlich werden und die defekte Stelle im Stromkreis nur sehr schwer lokalisiert werden kann. Es ist schwer die Spannungen in Teil 2 und Teil 3 (also alles nach der Absolutwertberechnung) korrekt zu messen und damit auf die eventuelle Fehlerquelle zu schließen.
Für eine präzise Messung ist es notwendig, das Ausgangssignal über einen geeigneten Tiefpass zu glätten und die Spannungsversorgungen der OPV zu entkoppeln. Weiterhin muss die Schaltung nach dem fertigen Aufbau über das Potentiometer am ersten Integrierer (welches das Offset des OPV einstellt) kalibriert werden.
Beim Aufbau fiel auf, dass darauf geachtet werden muss, die Sockel der Operationsverstärker etwas weiter auseinander zu setzen, da dies das Anlöten der Pins stark vereinfacht.

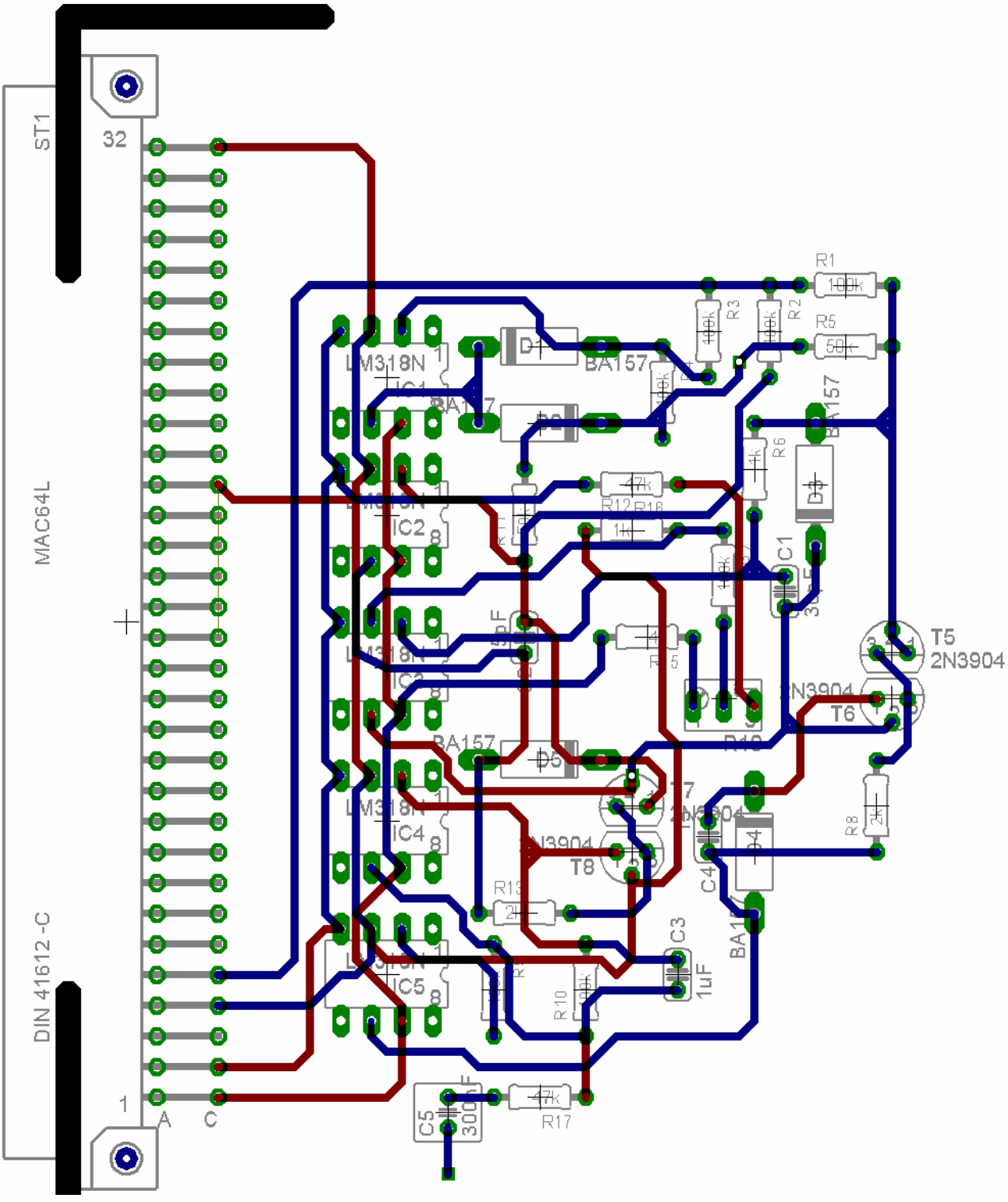


Abbildung 106 Eagle-Board (Original)

Bestückungsplan

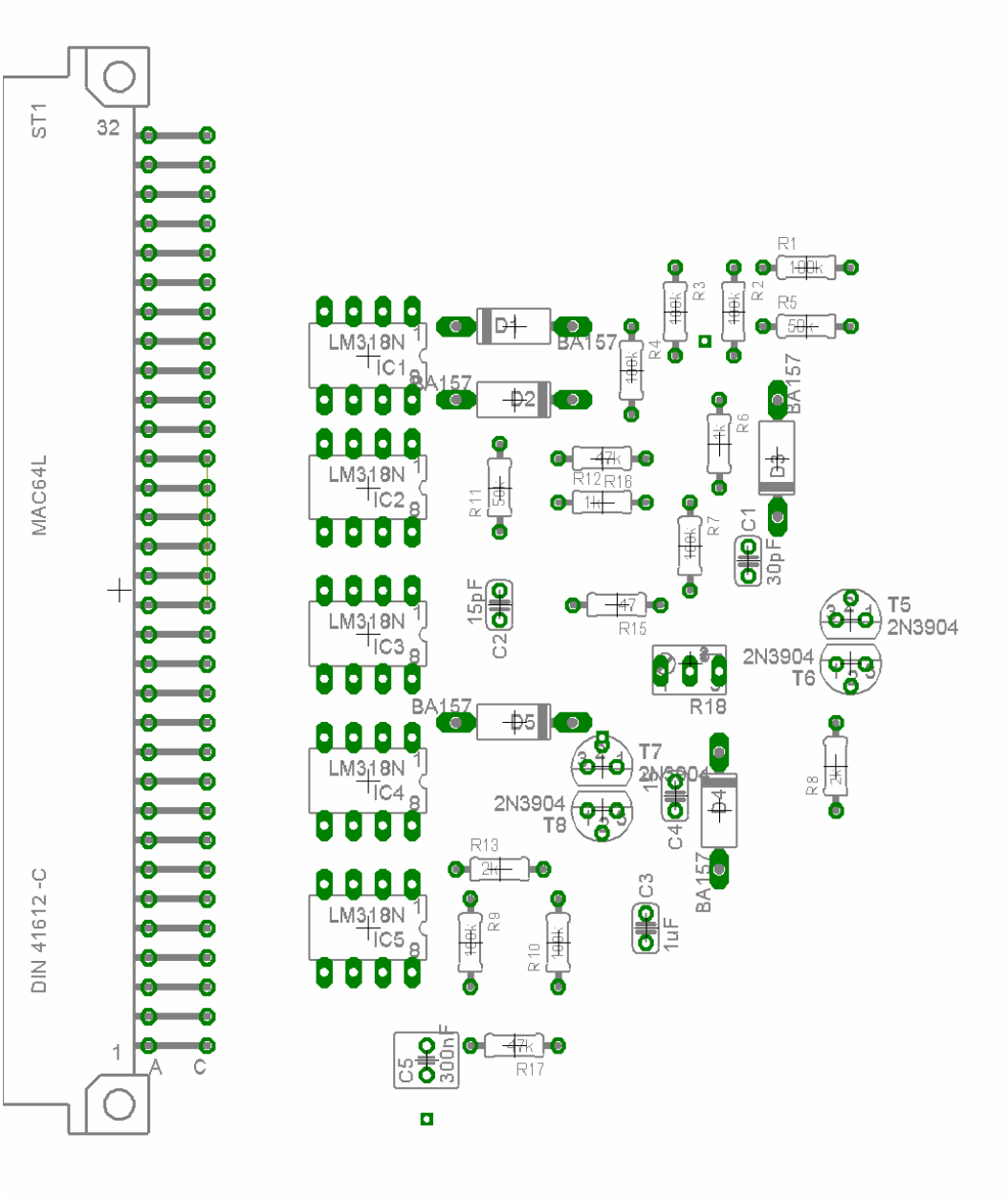


Abbildung 107 Bestückungsplan

Bauteile:

- 18 Widerstände wie im Plan eingezeichnet mit 1% max. Abweichung
- 5 Dioden D1N4448
- 5 Operationsverstärker LM118/ LM318
- 5 Kondensatoren unterschiedlicher Größe (2 * 15pF, 1 * 30 pF, 1 * 1uF 1 * 500nF zur Glättung des Signals + eventuell 5 * 100nF, um die OPV zu entkoppeln)
- 4 identische npn- Transistoren (Q2/Q3 und Q1/Q4 möglichst auf einem Chip)
- Es wurden Transistoren vom Typ 2N3904 benutzt, die mit Wärmeleitpaste und Schrumpfschläuchen aneinandergeklebt wurden.

- 1 Potentiometer 10k Ohm

Tabelle 7 Bauteile

Ätzlayouts

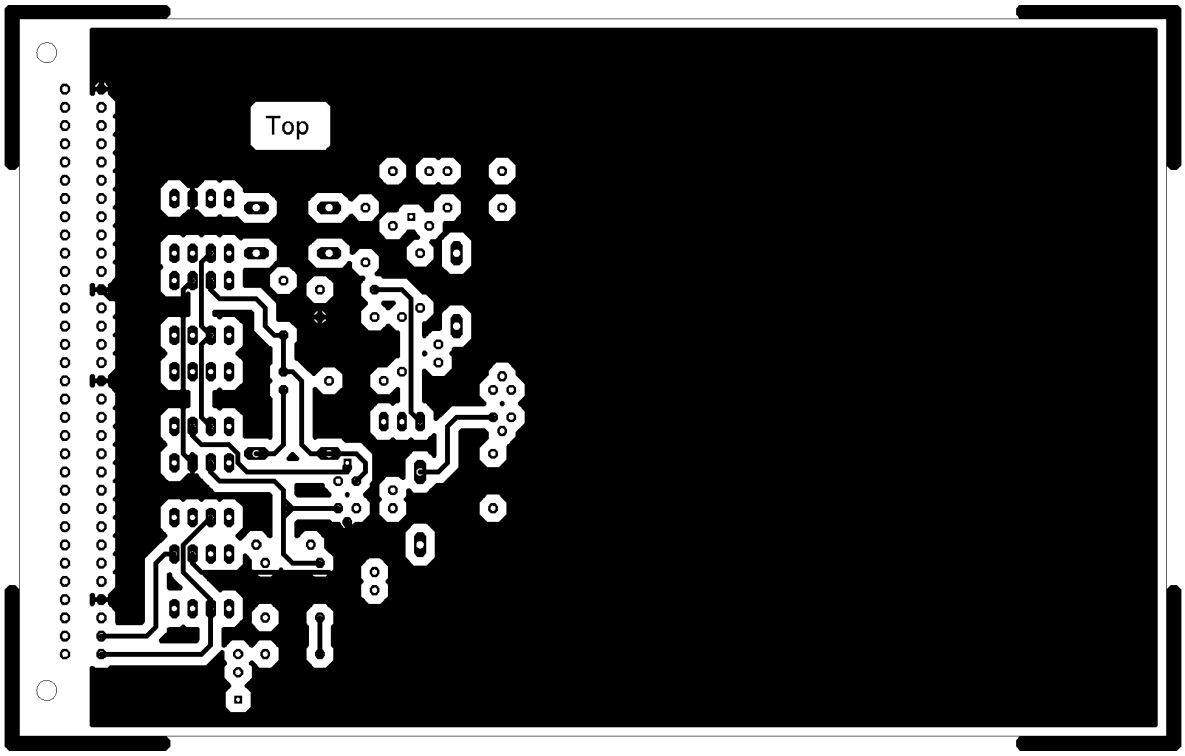


Abbildung 108 TrueRMS Top

:

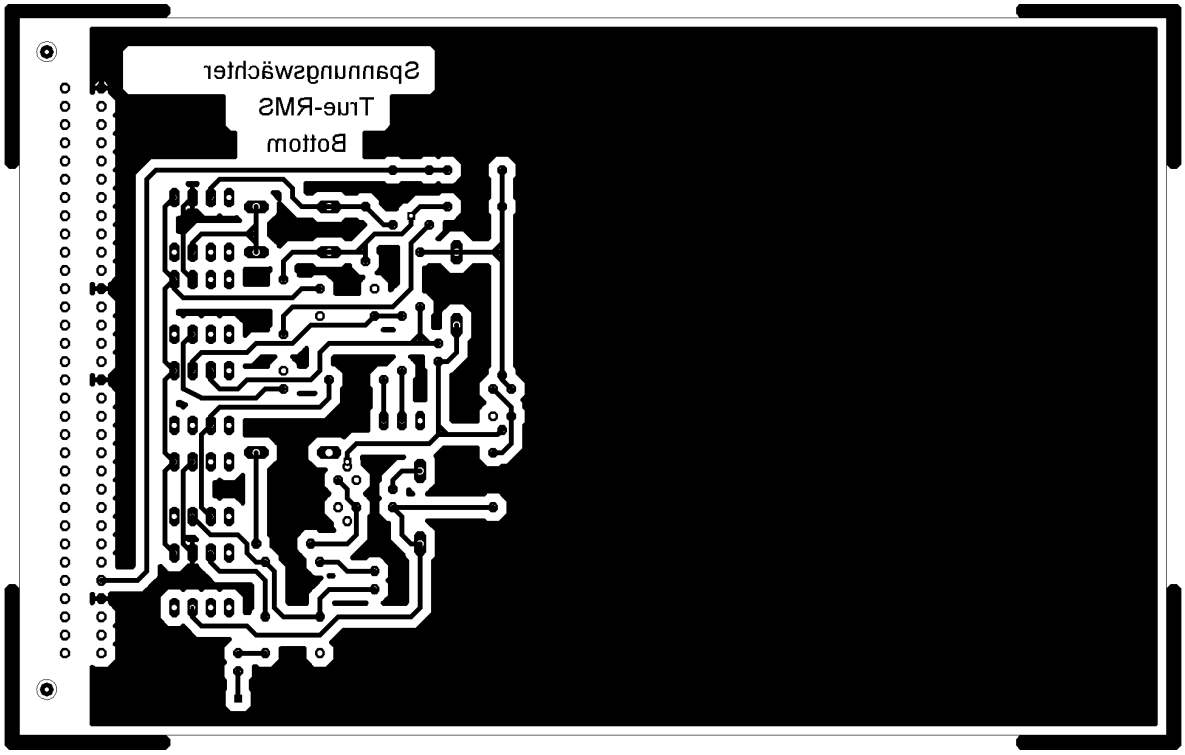


Abbildung 109 TrueRMS Bottom

5.5. Der Bargraph

Zu Beginn unserer Gruppenarbeit diskutierten wir, welche Messwerte wir mit dem Spannungswächter anzeigen wollen. Wir einigten uns auf Anzeigen zur Frequenz, der Peak-Spannung, der Ausfallzeit einen großen LED-Bargraphen, der die Anzeige des Effektivwertes der Netzspannung übernehmen soll. Ein Bargraph, dessen Optimalwert in der Mitte liegt und der die Abweichung nach oben bzw. nach unten anzeigt erschien, uns am sinnvollsten. Für die Stufen einigten wir uns auf 1 Volt Schritte. Zum einen, weil 1 Volt Schritte für einen Betrachter leicht einzuordnen sind. Zum anderen eignen sich die 1 Volt Schritte aus technischen Gründen gut. Die Widerstände zur Dimensionierung sind aufgrund ihrer Toleranz nur bedingt genau einzusetzen. Würden wir zu kleine Stufen im Komparator verwenden, könnte es sein, das die Toleranzen sich überlagern und zwei Stufen gleichzeitig leuchten. Wäre die Dimensionierung andererseits zu grob, so würde die Anzeige nicht genau genug unterscheiden können und damit nicht gut genug informieren. Zur Dimensionierung der Skala war uns folgendes vorgegeben:
Der Optimalwert liegt bei 230V
Die Abweichung beträgt maximal 10% unterhalb (207V) und maximal 6% oberhalb (244V) des Optimalwertes - bei größeren Abweichungen würde das öffentliche Netz abgeschaltet werden und die Anzeige auf Null zurückfallen.
Deshalb wurde beschlossen einen Bargraphen zu bauen, dessen Anzeige nach unten in 23 und nach oben in 27 Stufen vergleicht. Die ersten Designvorschläge sind im Anhang des Berichts zu finden. Das Design des Graphen und des gesamten Gehäuses übernahm Marion Stahel, die an der Universität der Künste in Berlin Industriedesign studiert.

Gesamte Elektronik :
Die Aufgabe bestand darin, eine Schaltung zu entwerfen, die den Bargraphen so mit Signalen versorgt, dass der aktuelle Effektivwert auf der Skala die entsprechenden Stufen zum leuchten bringt. Es gilt das Signal am Eingang unserer Gesamtschaltung richtig weiterzuleiten. Dieses Signal wird von einer anderen Arbeitsgruppe, der Netzteilgruppe, für uns aufbereitet. Die Netzspannung wurde von dieser Gruppe auf ein gleichgerichtetes Sinussignal transformiert, welches im Optimalfall einen Effektivwert von 9V anstatt 230V liefert. Die Transformation ist notwendig, da die Schaltung mit einer Versorgungsspannung von 15V betrieben wird und deswegen größere Signale nicht verarbeiten können.
Vom gleichgerichteten Eingangssignal muss zuerst der Effektivwert gemessen werden. Dafür wurde eine True-RMS Schaltung verwendet, mit der sich Leopold Georgi beschäftigt hat.

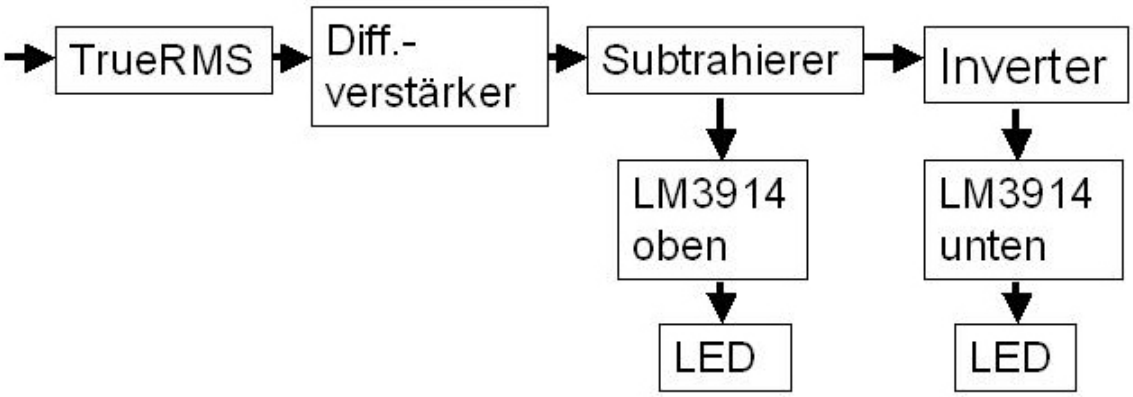


Abbildung 110 Analog BSB

Funktionsweise der Ansteuerung des Bargraphen

Ein weiteres Blockschaltbild *(Verweis)* deutet die einzelnen Schritte des Schaltalgorithmus an, der nach der Messung des Effektivwertes notwendig ist um den Bargraphen zu anzusteuern. Zur kurzen Erklärung: Der Effektivwert ist maximal plus 6% und minimal minus 10% . Der höchste Messwert aus der Effektivwertschaltung wäre also 9V + 6% * 9V = 9,54V, der niedrigste 9V- 10% * 9V= 8,1V. Die Bandbreite würde 9,54V- 8,1V= 1,44V betragen. Der Anzeigebereich auf dem Graphen soll bei 150V beginnen. Bis zur tatsächlichen Minimalabweichung bei 207V wären bei 1Volt-Schritten 207 Stufen-150 Stufen = 57 Stufen, die im Dauerbetrieb leuchten könnten und gar nicht in den Messbereich miteinbezogen werden müssen. Nur wenn das Netz aufgrund extremer Abweichungen abgeschaltet wird, müssen auch diese LEDs ausgehen. 244Stufen- 207Stufen+ 1Stufe= 38Stufen bilden den wichtigen Bereich von 207V bis 244V, nur hier finden Spannungsschwankungen statt. Es müssen also 1,44V Bandbreite in 38 Stufen aufteilen. Das würde bedeuten, dass pro Stufe ein Schwellensignal von 1,44V: 38V= 0.037V verarbeitet werden muss.
Deshalb muss die bewegliche Bandbreite von 1,44V weiterverarbeitet werde. Der untere Teil des Messwertes wird „abgeschnitten“, der verbleibende Rest wird noch um den Faktor 4,7 verstärkt.
Auf diese Weise kann die Bandbreite von 1,44V auf ein Vielfaches vergrößert und entsprechend auch die Stufenspannung pro 1Volt-Schritt auf 0,3V angehoben werden. Dazu wird ein Differenzverstärker verwendet, eine einfache Schaltung mit einem Operationsverstärker, deren Widerstände nach der entsprechenden Funktionsgleichung und dem gewünschten Ergebnis nach zu Dimensionieren sind. Zunächst wird die Differenzspannung vom Eingangssignal abgezogen. Diese Spannung wird mit einer Z-Diode von der Versorgungsspannung +15 V abgegriffen. Mit R1 = R2 = 1 kΩ und R3 = R4= 4,7 kΩ erreicht man eine Verstärkung von 4,7.
Die Verstärkung wurde so gewählt, dass sich das ausgegebene Signal nicht näher als 1,5Volt an die Betriebsspannung annähert, da sonst der Operationsverstärker keine optimale Signalverarbeitung mehr gewährleistet.
Das zurechtgeschnittene Signal wird jetzt über eine Schaltung aus Komperatoren, einem abgewandelten Schmidt-Trigger, der entsprechenden Zeile auf der Skala zugeteilt. Übersteigt das Eingangssignal einen bestimmten Schwellenwert beginnt die nächste Zeile von oben zu leuchten. Fällt das Eingangsignal zurück unter die Schwellenspannung, gehen die LEDs in dieser Zeile wieder aus. Die

Schwellenspannung wird über einen einfachen Spannungsteiler auf den negativen Eingang des OPs gelegt. Das Eingangssignal erreicht den positiven Eingang. Übersteigt das Eingangssignal die Schwellenspannung, liefert der Operationsverstärker an seinem Ausgang die positive Betriebsspannung. Ein gewöhnlicher Schmidt-Trigger wurde zu diesem Zweck etwas umgebaut. Normalerweise würde der Operationsverstärker im Falle einer Unterschreitung des Schwellensignals am Ausgang die negative Betriebsspannung liefern. Das würde aber die Leuchtdiode mit unter zerstören und außerdem soll die Diode nur bei Überschreitung der Schwelle leuchten. Deswegen wird der Pin für die negative Betriebsspannung des OPs auf Masse (0V) gelegt. Dann liegt kein Signal am Ausgang an (= 0V), wenn die Schwelle unterschritten wird. Zudem wird keine „Hysterese“ des Schmidt-Triggers benötigt, deshalb kann auf zwei Widerstände verzichtet werden. Als Operationsverstärker Wird der LM324 (Datenblatt ab Seite….) verwendet. Die Leuchtdioden werden mit dem Minuspol auf Masse gelegt.

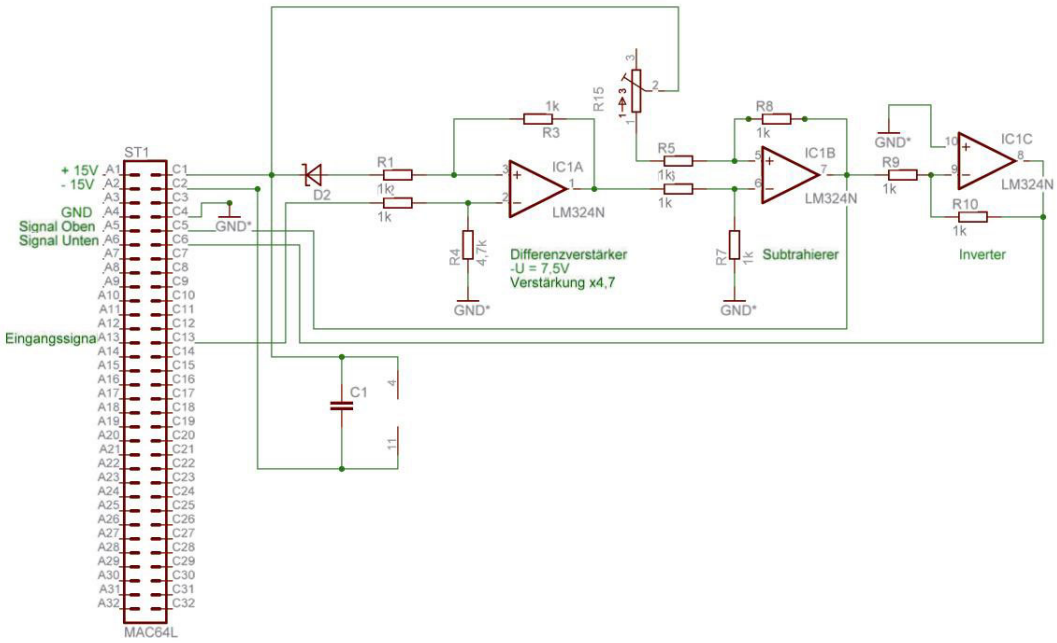


Abbildung 111 Analogschaltung für Baransteuerung

Einsparen von Bauteilen durch die Verwendung des LM3914:

Der LM3914 ist ein IC für zehn Stufen als integrierte Schaltung erhältlich. Fünf Stück vereinfachen und verbessern die Schaltung gewaltig. Das Datenblatt des LM3914 befindet sich im Anhang.
Die Beschaltung des LM3914 lässt sich anhand des Schaltbildes erkennen. Die Stromversorgung der Leuchtdioden ist exemplarisch an einer Leiste dargestellt.

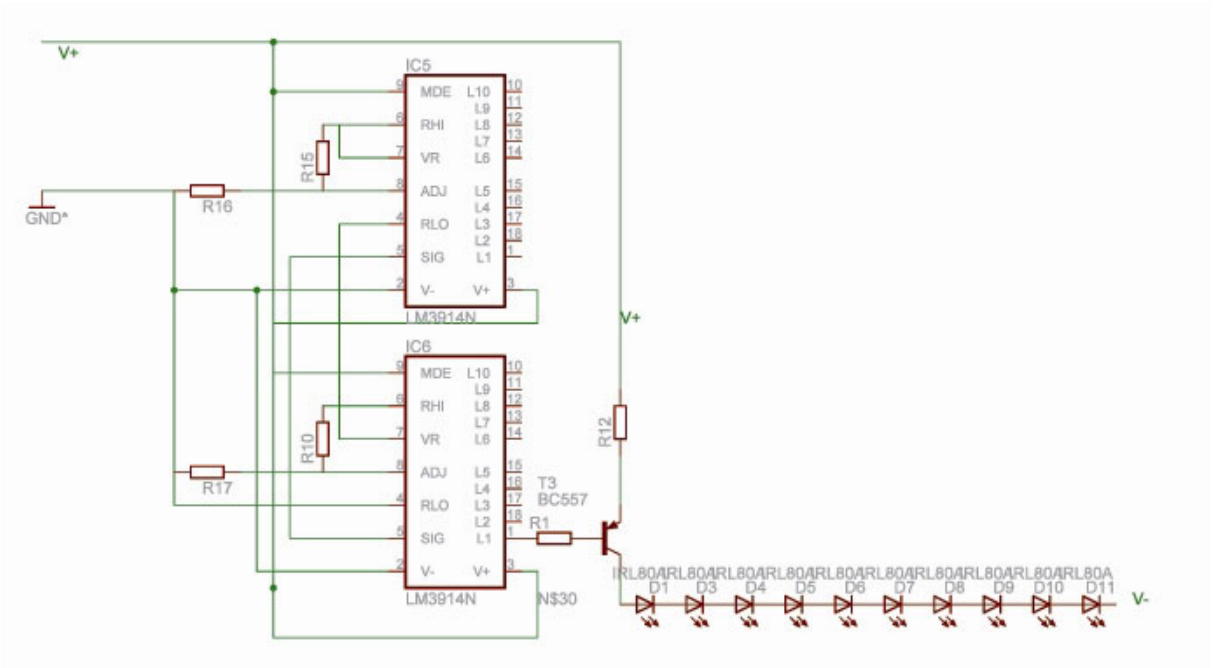


Abbildung 112 Ansteuerung der LEDs mit LM3914

Zwischen U-Vout und U-Vadj generiert der LM3914 selbst eine Spannung von 1,25V. Mit dieser Referenzspannung stellt man die oberste Schwelle- U-hi ein. Die unterste Schwelle U-lo erhellt man über einen weiteren Spannungsteiler. Die Dioden am Ausgang werden an der Kathode mit dem LM3914 verbunden, an der Anode mit der Betriebsspannung. Beim Unterschreiten des Schwellenspannung liegt an jedem Ausgang des Triggers, also auch am Minuspol der Diode die positive Betriebsspannung an, genau wie auf der anderen Seite der Diode. Es besteht also kein Spannungsgefälle, die Schwellenspannung Ud der Diode wird nicht überschritten, sie schaltet nicht. Bei Überschreitung einer Schwelle fällt das Signal am Ausgang ab, es entsteht ein genügend hoher Spannungsabfall über der Diode, sie beginnt zu leuchten.

Nach der Diskussion beim Schnittstellentermin zusammen mit den anderen Arbeitsgruppen wurde darüber gesprochen, ob es sinnvoll ist, auf dem Bargraphen 57 Stufen dauerhaft leuchten zu lassen. Das Argument dafür war der optische Eindruck. Das Argument dagegen war ein hoher Energieverbrauch für die vielen Dioden. Also wurden neue Designvorschläge entworfen, die im Folgenden aufgelistet sind.

1. Eine „abgespeckte“ Version des ursprünglich geplanten Bargraphs. Dieser soll „von Null an“ die momentane Spannung anzeigen, jedoch ohne die 57 dauerhaft leuchtenden Stufen
2. Es wird nur die Abweichung vom Normalwert (230V) dargestellt.
3. Wie 1., nur im Punktbetrieb. Das bedeutet, dass immer nur der tatsächliche Spannungswert mit einer Linie angezeigt wird.
4. Wie 2., nur im Punktbetrieb.

Letztendlich einigen wir uns auf Vorschlag 2.

Die Anzeige der Abweichung nach unten war eine neue Herausforderung. Die LEDs sollen nun bei Überschreitung der Schwelle nicht leuchten, bei Unterschreitung der Schwelle aber schon. Es wurden mehrere Schaltungen mit dem LM3914 ausprobiert, jedoch sollte keine dieser Anforderung gerecht werden. Auch die nicht, die als Zero Centimeter Bargraph im Datenblatt angegeben ist. Das Problem ist, dass egal ob Schwelle überschritten oder nicht, das Potentialgefälle zwischen Ausgang am Trigger und über die Diode zur Betriebsspannung nicht groß genug ist, die Diode zum Leuchten zu Bringen, bzw. nie klein genug wird , die Diode auszuschalten. Man hat entweder Dauerleuchten, oder immer abgeschaltete LEDs. Ein weiteres Problem war, dass wir unsere elf LEDs nicht direkt hinter dem LM3914 anbringen konnten sondern noch eine Schaltung zur Stromersorgung entwerfen mussten. Dies war bei „positiver“ Auslenkung kein Problem, ging jedoch Aufgrund der anderen Beschaltung des LM3914 nicht bei der „negativen“ Auslenkung.

Dieses Problem wurde mit einem Invertierer umgangen, das negative Signal (für die Auslenkung nach unten) wurde invertiert, anschließend kann eine Ansteuerung wie „nach oben“ verwendet werden.
Mit diesem Trick war es kein Problem die ICs miteinander zu verbinden und richtig einzustellen. Die Ansteuerung der LEDs wurde mit Hilfe eines PNP Transistors erstellt.

Ein weiteres Feature, was erst gegen Ende des Projektes entwickelt wurde, war die „Peak & Hold“ Schaltung. Diese sollte einmal erreichte Maximal- und Minimal-Werte des Effektivwertes speichern und als Punkte links und rechts vom Bargraphen darstellen. Das Problem der Schaltung war die Speicherung des Wertes ohne eine richtige Speicherzelle. Diese konnte aber dann mit Logikbauteilen realisiert werden und ist ein schönes Zusatzfeature.
Das OR-Glied ist an den Eingängen zu einen mit den Leuchtdioden einer LM-Stufe verbunden. Leuchten die Dioden dieser LM-Stufe, d.h. fällt ein genügend großes Signal darüber ab, so wird dieser Spannungsabfall genutzt, in dem über den zwei letzten Dioden das Signal von 4V bis 0V parallel abgegriffen wird, um damit dem OR-Glied eine „Eins“ als Signal zu senden. Leuchten die Leisten nicht, ist das abgefangene Signal für das OR-Gatter zu klein und wird als „Null“ interpretiert.
Am zweiten Eingang des OR-Glieds wird das Ausgangssignal des AND-Gliedes rückgekoppelt.
Der Ausgang des OR-Gliedes ist mit dem ersten AND-Eingang verbunden. Der Zweite AND-Eingang liegt an der Reset-Leitung. Dort liegt immer „Eins“ an und nur einmal am Tag „Null“, wenn alle Anzeigen zurückgestellt werden. Der AND-Ausgang führt zurück zum OR-Glied. Durch diese Rückkopplung kann, wie schon erwähnt wurde, ein Wert zwischengespeichert werden. Leuchten die Diodenleisten einer Stufe, liegt an OR in jedem Fall eine „Eins“, am Ausgang also auch und damit am AND-Ausgang auch (Reset ist immer auf „Eins“). Fällt die Spannung und die Diodenleisten leuchten nicht mehr, so liegt an OR immer noch das rückgekoppelte Signal mit „Eins“. Dadurch liegt am Ausgang von OR immer noch eine „Eins“, am Eingang von AND zweimal die „Eins“, und damit am Ausgang auch.
Nun zu den Peak-and-Hold Dioden. Sie leuchten, wenn an einem AND-Ausgang „Eins“ anliegt und am nächst höheren eine „ Null“. Dann ist der Spannungsabfall groß genug. Würde am UND-Ausgang darüber auch „Eins“ anliegen, würde die Diode

nicht mehr leuchten, weil kein Signal mehr darüber abfällt. Es leuchtet die Diode aus der nächst höheren Stufe.
Einmal am Tag wird ein Reset-Signal gesendet, welches durch seine „Null“ automatisch alle Ausgänge der UND- Schaltung auf Null zurücksetzt. Keine Diode leuchtet mehr bis neue Peaks erreicht werden.
Die oberste Stufe der Peak-And-Hold Schaltung, die „Dummie-Stufe“ hat einen besonderen Grund. Das „Eins“- Signal am AND-Ausgang ist eine Spannung von drei Volt. Das „Null“ -Signal eine Spannung von einem Volt. Das Spannungsgefälle von 2V von einer Stufe zu nächsten reicht genau, um eine Diode zum Leuchten zu bringen. Würde man die oberste Diode mit dem Minuspol auf Masse legen, so würde ein Spannungsgefälle von 3 Volt entstehen und es können mit dieser Spannungsdifferenz zwei Dioden leuchten.
Die Peak And Hold Schaltung gibt es jeweils für die positiven Peaks und die negativen Peaks.

Bauteilliste

9 x 1kOhm
2 x 4,7kOhm
1 X LM324N
1 X ZDiode
1 X Potentiometer 0-2kOhm
1 X 100µF
1 x Busstecker

5.6. Design

Einleitung

Der Spannungswächter wird den TU Studenten die aktuelle Berliner Netzqualität der Netzspannung anzeigen.

Gehäuse

Die Gehäuseform (s. Abb. 01) war schon relativ früh bestimmt. Die Dimensionen haben sich von anfänglich 180cm Höhe auf 125cm reduziert und das Gesamtvolumen liegt bei ca. 140 l. Grundfläche und Deckel sind Trapezförmig, das untere wird nach vorne schmaler während das obere nach hinten zusammen läuft. Die Front- und Rückwand sind plan, die beiden seitlichen sind in vier Richtungen verdrehte.
Die geschwungenen Seitenwände verleihen dem Spannungswächter eine gewisse Dynamik. Die Form ist durch das dominante Bargraph-Element inspiriert.
Die Front und Rückwand sind komplett abschraubbar; für den Fall, dass die Klappöffnung im unteren Bereich auf der Rückseite, für Reparaturen nicht ausreicht.

Innenleben

Im Gehäuse befindet sich ein 19“ Einschub, in dem alle Platinen (mit einigen Ausnahmen) ihren Platz finden, zwei Akku (mit dazugehörigen Platinen an einem extra Platz) und einige Netzteile (Trafo etc.).
Damit man das Innenleben von außen sehen kann, liegt der 19“ Einschub (s. Abb.) an der Rückwand des Gehäuses, wo eine Plexiglasscheibe den Einblick ermöglicht. Der Einschub steht mit seiner schmalen Seite auf einem 13,5 cm breiten Brettchen, das sich 20 cm über dem Boden befindet. Zusätzlich wird er an einer darunter und einer darüber (65 cm ü. Boden) liegenden, horizontalen Leiste festgeschraubt. Somit befindet sich der Einschub in der unteren Hälfte des Gehäuses, damit der Schwerpunkt tief bleibt.
Die beiden Akku mit einem Geicht von zusammen ca. 4-6 kg, werden ganz unten an der schmalen Stelle der Frontwand platziert, damit möglichst viel Platz im Bodenbereich für den Trafo frei bleibt. Von der Gewichtverlagerung her gesehen wäre es sinnvoller die schweren Akkus an der breiten Rückwand zu positionieren. Sobald die Trafo-Dimensionen genauer definiert sind kann man die Lage der Akkus noch einmal überdenken. Zwischen den beiden Akkus (je 9,5 x 4,6 x 15 cm) muss Platz für ein Temperaturmessfühler sein. Der Ladestand der Akkus wird über 8 LED’s an der Außenseite angezeigt (genaueres unter Oberfläche). Über zwei weitere LED’s wird angezeigt, in welchem Betriebsmodus sich die Akkus befinden, ‚Aufladem odus’ oder ‚in Betrieb’.

Oberfläche

Da bei diesem Projekt der Schwerpunkt auf der praktischen Erfahrung liegen soll, fällt die Wahl für das Oberflächenmaterial auf Leiterplatinen. Im übertragenen Sinne um das Innere nach außen hin sichtbar zu machen. Aus demselben Grund, wird das Fenster an der Rückwand angebracht.

Anfangs war die Oberfläche in grün geplant. Der Besuch in der Firma ANDUS hat jedoch ein vielseitigeres Angebot an Farben offenbart als erwartet. Nach gründlichem Durchstöbern der Platinenabfälle viel die Wahl auf gelbe und weiße Platinen mit silber-, gold- und bronzefarbenen Leiterbahnen. Für den zentralen Bargraphen eignete sich eine komplementär, violett-blaue schimmernde Platte.

Bargraph

Anfangs war der Bargraph so geplant, dass er nur nach oben ausschlägt und diese Abweichung sollte über ca. 100 cm dargestellt werden, bei einer Breite von 10 LED's, was sich als übertrieben herausstellte. Außerdem macht es wenig Sinn die Anzeige bei 0 Volt zu beginnen, da sich in den „ersten 200 Volt" nichts verändern würde. Es würde also nur unnötig Strom verbraucht und die gesamte Abweichung unter 230 Volt würde komplett vernachlässigt.
Der „neue" Bargraph (s. Abb.) besteht aus 546 LED's (462 Gelben und 84 Roten)
D.h. aus 42 übereinander liegenden Reihen, die jeweils eine Breite von 11 LED's haben. Beidseitig jeder Elferzeile befindet sich eine weitere LED, die den letzten Maximal- und Minimalwert festhält. Der Idealwert von 230 Volt wird durch einen zentralen Balken von fünf LED-Reihen angezeigt, jede weitere Reihe bedeutet 1 Volt Abweichung. Die Abweichung im positiven Bereich wird bis zu +6% Abweichung und im negativen Bereich bis zu -10% angezeigt. Somit reicht der Bargraph von 207 Volt bis 244 Volt. Die Werte sind seitlich in Fünferschritten angegeben.
Die LED's werden auf einer extra für das Layout gefertigten Lochrasterplatte angeordnet. Auf der Rückseite wird die geätzte Platine fest gelötet und von vorne werden die LED's mit einer violett-blauen Leiterplatine in die exakte Anordnung gebracht und eingefasst.

Warum gelbe und rote LED's?
Von der Farbbedeutung her würde es Sinn machen den Bargraphen nach dem Vorbild einer Ampel zu gestalten. Im Bereich des Idealwertes wäre er mit grünen LED's bestückt, bei mittlerer Abweichung mit gelben und bei kritischen Werten mit roten.
Das Erscheinungsbild des gesamten Bargraphen würde dadurch jedoch zu stark in drei Bereiche aufgebrochen und somit in Einzelteile zerfallen.
Die Anzeige soll die Abweichung in fließender Bewegung darstellen, daher ist sie in kleinen Einvolt-Schritten aufgebaut.
Um das Bargraph Element nicht aufzuspliten beschränkt sich die LED-Farbe im zentralen Teil auf nur einen Farbton, Gelb und seitlich Rot für die Spitzenwerte.
Warum Gelb?
Es steht nur Rot, Gelb und Grün zur Auswahl, weil wir uns aus ökologischen Gründen für Low Current LED's entschieden haben.
Rot fällt weg, weil es im Idealwertbereich unpassend ist und Grün, weil es bei starker Abweichung nicht viel Sinn machen würde. Gelb steht für Wechsel und Veränderung, was in jedem Bereich der Fall ist.
Neben den wirren Zeichnungen der Leiterplatten bringt die Reduktion auf nur gelbe LED's zumindest eine gewisse Ruhe in die Gesamterscheinung. Die Farbwiederholungen erzeugen eine Zusammengehörigkeit aller Elemente, es entsteht ein harmonisches Bild.

Siebensegmentanzeigen

Insgesamt gibt es drei 7-Segmentanzeigen. Eine dreistellige (rot) für die Spannungsanzeige, eine vierstellige (rot) für die Frequenz und eine sechsstellige (rot) für den Stromausfall der letzten 24h. Unter der Spannungsanzeige sind zwei LED's angebracht, die eine ist mit Maximum beschriftet und die andere mit Minimum. Bei der Frequenzanzeige gibt es drei LED's unter der Anzeige. Einmal für Max., einmal für Min. und einmal für den Mittelwert.
Je nach dem wie unruhig das Oberflächenmosaik der Leiterplatinen wird, ist es von Vorteil, die 7-Segmentanzeigen mit z.B. einer weißen oder gelben Umfassung hervor zu heben.

Akku-Anzeige

Über die Acht gelben LED's unten rechts auf der Rückseite wird der aktuelle Akkuladestand angezeigt. Die LED's liegen übereinender in einer Reihe und sind mit „empty" und „full" beschriftet.
Die beiden „Akkumodi" LED's befinden sich neben der Aufladeanzeige. Eine grüne für die „in Betrieb" Anzeige und eine rote für die „Auflademodus"- Anzeige.

An-/Aus-Schalter und Einstellungen

Der An-/Aus-Schalter befindet sich auf der Rückseite über dem 19" Einschubfenster. ? Die Platinen der Gruppe 2 müssen mit einem Schraubenzieher erreichbar sein. Für allgemeine Korrekturen und Reparaturen ist es sinnvoll, dass der gesamte Einschub, die Akkus und der Trafo zugänglich bleiben. Daher muss die untere Hälfte der Rückwand abnehmbar oder aufklappbar sein.

ökologische Aspekte

(genauere Ausführungen siehe Referatausarbeitung)

Gehäuse

Pluspunkte

Das Grundgerüst des Bargraphen ist aus Holz, also ein nachwachsender Rohstoff. Holz ist relativ umweltfreundlich, was die Herstellung und Entsorgung betrifft.
Die Platinen-Verkleidung könnte man als Aufschubrecycling bezeichnen. Ein Abfallprodukt wird eins zu eins übernommen und bekommt eine neue Funktion. Dadurch ist das Entsorgungsproblem nicht gelöst, es wird jedoch um die Lebenslänge eines weiteren Produktes verlängert.

Kritikpunkte

Ein Nachteil sind natürlich die Giftstoffe der Platinen deren Entsorgung für die Umwelt eine große Belastung ist.
Von dem Technikvolumen her gesehen könnte man die Höhe des Wächters mindestens halbieren und sein Volumen annähernd vierteln.
Die Informationen des Spannungswächters soll man jedoch aus einigen Metern Entfernung ablesen können, daher ist eine gewisse Größe notwendig.

Bargraph

Es werden Low Current LED’s verwendet, an Stelle von normalen LED’s. Diese verbrauchen nur einen Zehntel der Energie einer gewöhnlichen LED. Leuchtdioden haben zu dem den Vorteil, dass sie sehr robust sind und selten ersetzt werden müssen. Außerdem geben LED’s keine UV-Anteile ab.

6. Gruppe 5

6.1. Aufgabe der Gruppe

Die Aufgabe der Gruppe 5 ist im Falle eines Stromausfalls die Stromversorgung auf Akkubetrieb umzuschalten. Des Weiteren sind hierzu verschiedene Kontrollmechanismen, wie Spannungs- sowie Temperaturmessung der Akkus nötig. Diese Signale werden in der Logik ausgewertet, die dann die Schaltplatine steuert (siehe Abb. 1). Der Ladezustand, sowie der aktuelle Betriebszustand werden auf der Rückseite des Gehäuses durch LEDs gesondert angezeigt.

Zur Bewältigung dieser Aufgaben hat unsere Gruppe drei Platinen entwickelt:
1. Mainplatine
2. Bargraphplatine
3. Powerplatine.

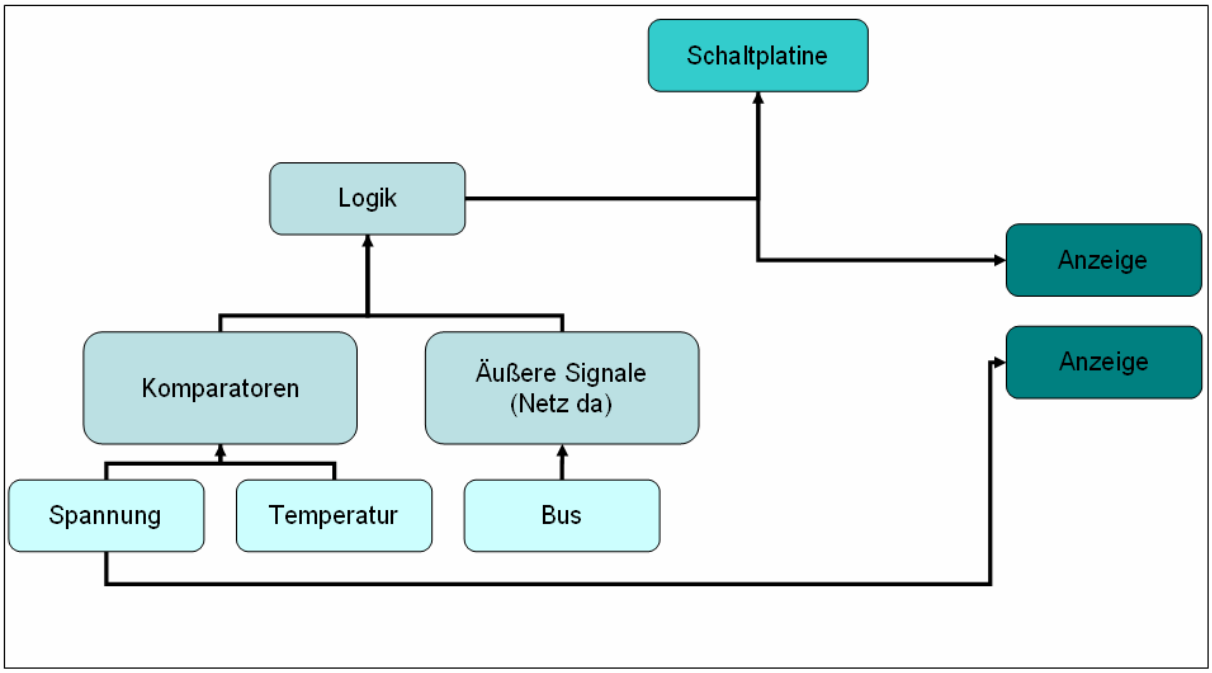


Abbildung 113 BSB

6.2. Akkumanagement

Inhalte der Mainplatine

Die Logik, Temperaturmessung, Messung der Akkuspannung.

Die Logik

Aufgabe

Die Logik soll die soll mit Hilfe der Eingangssignale Netz da, Temp_h, Temp_n, Akku_voll & Akku_leer entscheiden ob der Akku Benutzt oder Geladen werden soll.

Logik-Signale

	Logik Pegel	
	1	0
Netz	Netz ist da	Stromausfall
Temp_h	Akku Temperatur ist nicht zu hoch	Akku Temperatur ist zu hoch
Temp_n	Akku Temperatur ist nicht zu niedrig	Akku Temperatur ist zu niedrig hoch
Akku_leer	Akku ist leer	Akku ist nicht leer
Akku_voll	Akku ist voll	Akku ist nicht voll
Akku laden	Akku soll geladen werden	Akku soll nicht geladen werden
Akku benutzen	Akku soll benutzt werden	Akku soll nicht benutzt werden

Tabelle 8: Logik Signale

Logik-Tabelle

	Netz	Temp_h	Temp_n	Akku_leer	Akku_voll
Benutzen	0	1	1	0	/
Laden	1	1	1	/	1

Tabelle 9: Logik-Tabelle

Logik-Schaltung

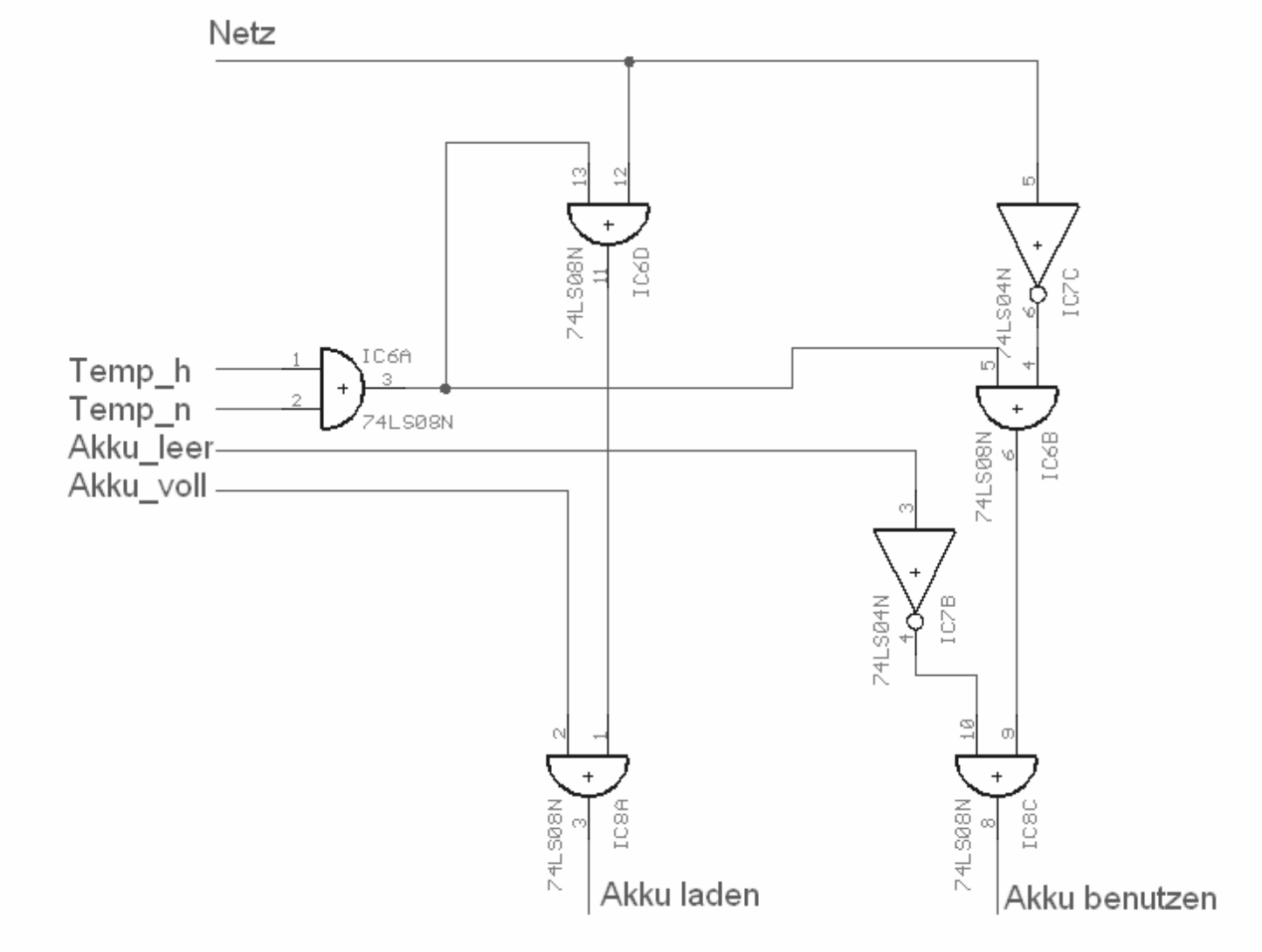


Abbildung 114: Logik-Schaltung

Die Logikschaltung funktioniert und die Ausgangssignale Akku laden und Akku benutzen werden auch optisch ausgegeben. Dies geschieht mit zwei LEDs, die sichauf der Bargraph-Platine befinden.

Akkuspannungsüberwachung

Aufgabe

Die Aufgabe dieser Schaltung ist die Überwachung der von den Akkumulatoren („Akku“) gelieferten Spannung.
Dieses ist notwendig, da die verwendeten Blei-Akkus empfindlich hinsichtlich einer Über- und Unterspannung reagieren und dabei beschädigt werden können. Des Weiteren kann anhand der von den Akkus gelieferten Spannung die momentane Ladesituation geschlussfolgert werden.

Durch die Datenblätter zeigte sich, dass die Spannung pro Akku-Zelle 1,8 V nicht unter- und 2,4 V nicht überschreiten darf. Bei Verwendung von zwei Blei-Akkus mit insgesamt zwölf Zellen ergibt sich somit eine Spannungsuntergrenze bei ca. 21 V bzw. eine Spannungsobergrenze bei ca. 29 V.

Fehler

Der Schmitt-Trigger (siehe Komponente) für die obere Grenze hätte mit einem invertierenden Schmitt-Trigger (siehe Komponente) eleganter verwirklicht werden können. Dieses ist aber zu spät aufgefallen. Das Problem wurde durch eine Invertierung des Signals in der Logik gelöst.

Aufbau und Funktion

Idee

Die Idee hinter der Schaltung ist der Vergleich der Akkuspannung mit zwei Werten als Grenzen zwischen Unter- und Überspannung und die Signalübermittlung an die Logik.

Blockschaltbild

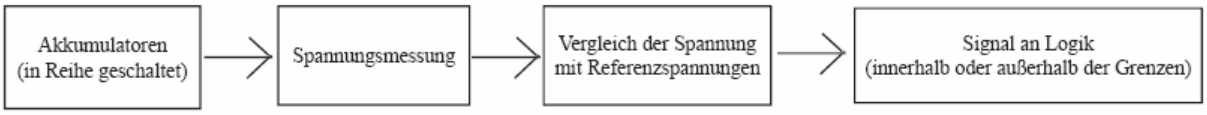


Abb. 1: Blockschaltbild

Die Spannung wird über die Akkus gemessen und wird mit zwei festgelegten Referenzspannungen verglichen. Befindet sich die Akkuspannung zwischen den Grenzen, wird ein positives Signal an die Logik weitergegeben. Ist die Akkuspannung außerhalb des „erlaubten Bereichs“ wird ein negatives Signal an die Logik geschickt, was wiederum den Abbruch des Ladens oder der Benutzung zur Folge hat.

Komponenten und Berechnung

Schmitt-Trigger

Ein Schmitt-Trigger ist ein Komparator an dessen Ausgang eine über einen Widerstand geleitete Rückkopplung angeschlossen ist.
Der Komparator (Abb.2) vergleicht die an den Eingängen anliegenden Spannungen. Ist die Spannung am Plus-Anschluss höher als die Spannung am Minus-Anschluss so schaltet der Komparator die Betriebsspannung Ub+, die am Betriebsspannung-Plus-Anschluss anliegt, durch. Ist die Plus-Spannung niedriger als die Minus-Spannung, wird die Betriebsspannung Ub-, also die Spannung am Betriebsspannung-Minus-Pin, am Ausgang eingestellt.

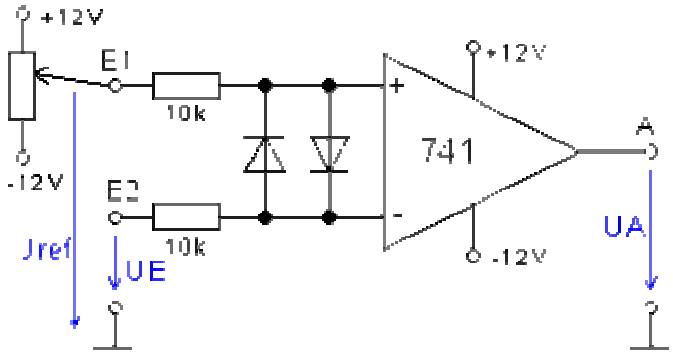


Abb. 2: Komparator

Der Schmitt-Trigger (Abb.3) ermöglicht es unterschiedliche Werte für die Umschaltvorgänge von High auf Low und von Low auf High, genannt Hysterese, einzustellen. Dieses wird benutz um ein Schwingen des Komparators zu verhindern, wenn die Messspannung um die Umschaltgrenze variiert.

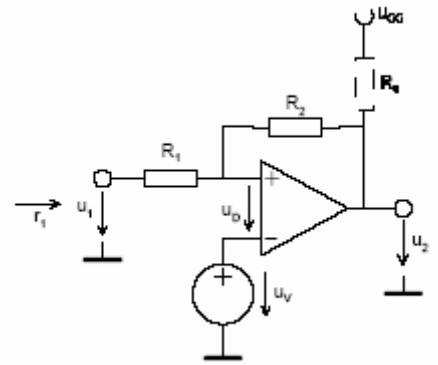


Abb. 3 Nichtinvertierender Schmitt-Trigger

Für die Berechnung wurden die Umschaltpunkte und R2 vorgegeben. Da der Akku nicht Überladen werden darf, wurde eine Akku-Ist-Voll-Grenze, bei der ein „Akku ist voll“-Signal an die Logik gesendet wird, bei 29V festgelegt. Die Akku-Ist-Nicht-Mehr-Voll-Grenze des

Komparators wurde auf 27V gelegt (Hysterese, siehe oben). R2 wurde frei bestimmt und mit diesen Werten Uv ausgerechnet.
Für den Komparator der unteren Grenze, also derjenige für den Entladungsschutz, wurde mit den Werten 22V und 24V analog gerechnet.

$$u_V = \frac{u_{T+} \cdot u_{SAT+} - u_{T-} \cdot u_{SAT-}}{u_{SAT+} - u_{SAT-} + u_{T+} - u_{T-}} = \frac{u_{SAT}\left(u_{T+} + u_{T-}\right)}{2u_{SAT} + u_{T+} - u_{T-}} \qquad \left(u_{SAT} = -u_{SAT-} = u_{SAT+}\right)$$

$$u_{T\pm} = \frac{u_V\left(R_1 + R_2\right) - u_{SAT\mp}R_1}{R_2} = \frac{u_V\left(R_1 + R_2\right) \pm u_{SAT}R_1}{R_2} \qquad \left(u_{T+} > u_{T-}\right)$$

$$R_1 = \frac{R_2\left(u_{T+} - u_{T-}\right)}{u_{SAT+} - u_{SAT-}} = \frac{R_2\left(u_{T+} - u_{T-}\right)}{2u_{SAT}}$$

$$u_H = \frac{\left(u_{SAT+} - u_{SAT-}\right)R_1}{R_2} = \frac{2R_1u_{SAT}}{R_2}$$

Invertierender Schmitt-Trigger

Der oben beschriebene Schmitt-Trigger schaltet wenn die Plus-Spannung die Minus-Spannung übersteigt und einen gewissen Hysterese-Wert übersteigt von Ub- auf Ub+, also von einem L-Pegel um auf einen H-Pegel.
Eine abgewandelte Form dieser Schaltung erreicht das Gegenteil der Umschaltvorgänge und ergibt den so genannten „invertierenden Schmitt-Trigger“. Dieser gibt am Ausgang genau die gegenteilige Spannung (statt Ub+ dann Ub-) also die invertierte Spannung aus.

Pspice

Die Schaltung wurde mittels PSpice erzeugt und ausgewertet (siehe Abbildungen). Dabei wurde stellvertretend für die beiden Akkus eine Spannungsquelle mit Sinusausgang eingesetzt. Die Amplitude wurde auf sechs Volt mit einer Offsetspannung von 24 Volt eingestellt. Damit wird ein Spannungsbereich von 18V bis 30V simuliert was einen leeren bis vollen Akku darstellt.
Diese simulierte Akkuspannung wird durch einen Spannungsteiler auf 1/3 heruntergeregelt und an den Eingang eines Impedanzwandlers angelegt.
Danach folgen zwei Schmitt-Trigger, einer für die obere Grenze und einer für die untere. Die Widerstände R5 und R1 bzw. R3 und R2 sind ausschlaggebend für die Hysterese (Beschreibung und Berechnung siehe oben).
Als Referenzspannungen wurden hierbei Spannungsquellen gewählt, die später als Spannungsteiler realisiert werden sollen.
Am Ausgang der Schmitt-Trigger kommt ein Signal, das entweder als Hi (ca. +15V) oder als Low (ca. -15V) zur Logik weitergeführt wird.

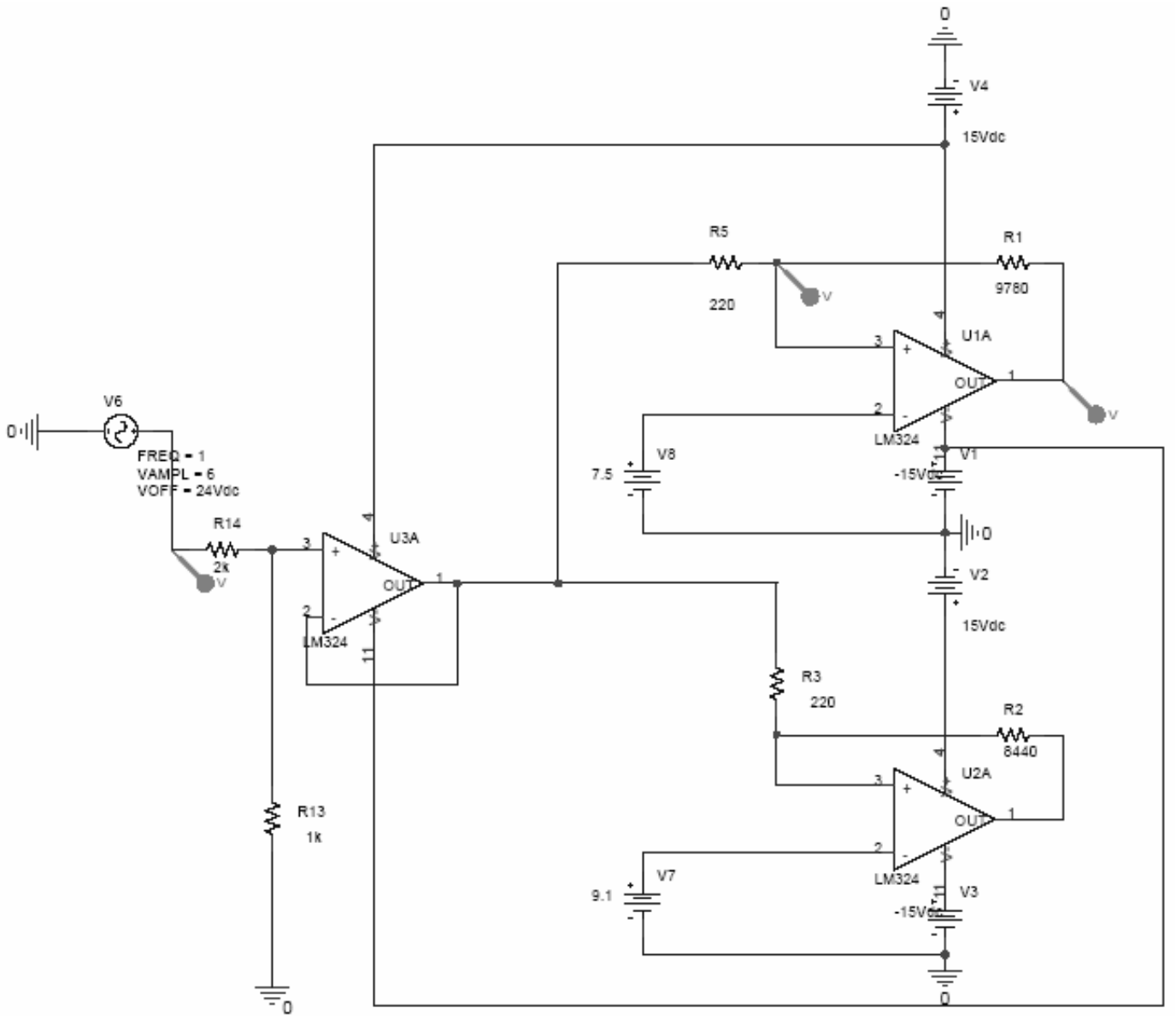


Abb. 4: PSpice-Schaltung Akkuspannungsüberwachung

Das Ergebnis der Messungen:

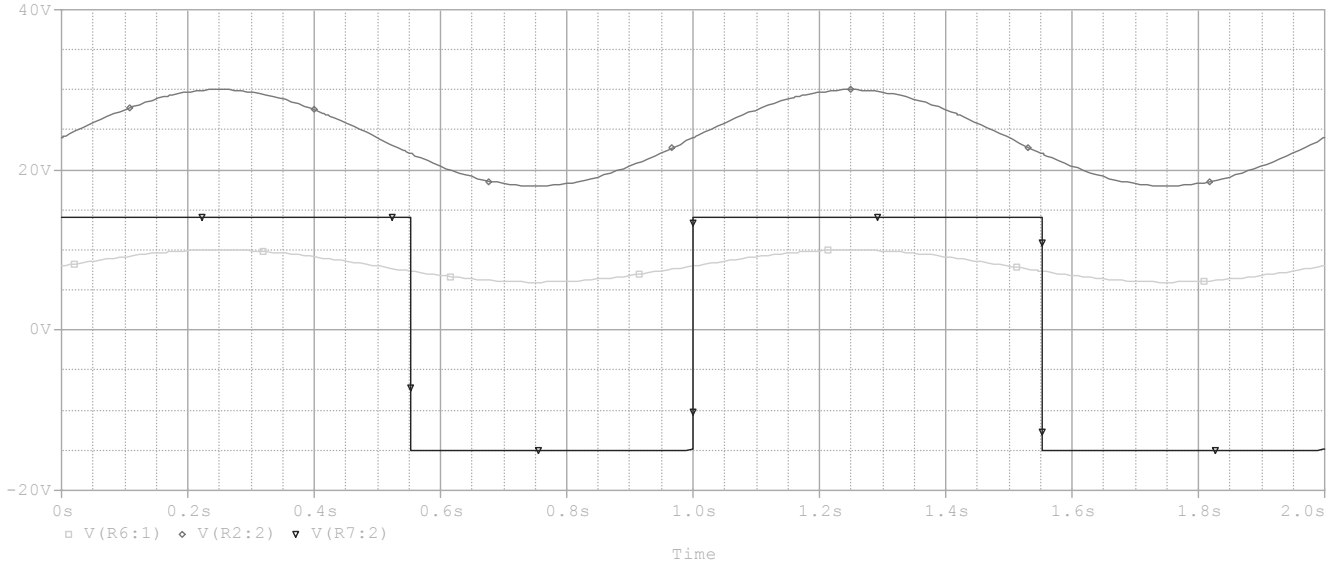


Abb. 5: PSpice-Simulation Akkuspannungsüberwachung

Eagle

Der oben beschriebene Schaltplan wurde in Eagle übernommen.
Von dem Stecker geht eine Leitung über den Impedanzwandler und teilt sich auf zu den zwei Schmitt-Triggern, deren Versorgungsspannung Ub+ +15V und Ub- -15V betragen. Die an dem Minus-Eingang anliegenden Referenzspannungen werden durch Potentiometer eingestellt.
An den jeweiligen Ausgängen sind Dioden auf Masse geschaltet, damit die -15 Volt, die bei einem L-Pegel am Ausgang herrschen, nicht über die Logik anliegen und diese Beschädigen.
Der Ausgang des Schmitt-Triggers, welcher für die obere Grenze verantwortlich ist, wird vor der Logik invertiert (siehe Abschnitt „Fehler“).

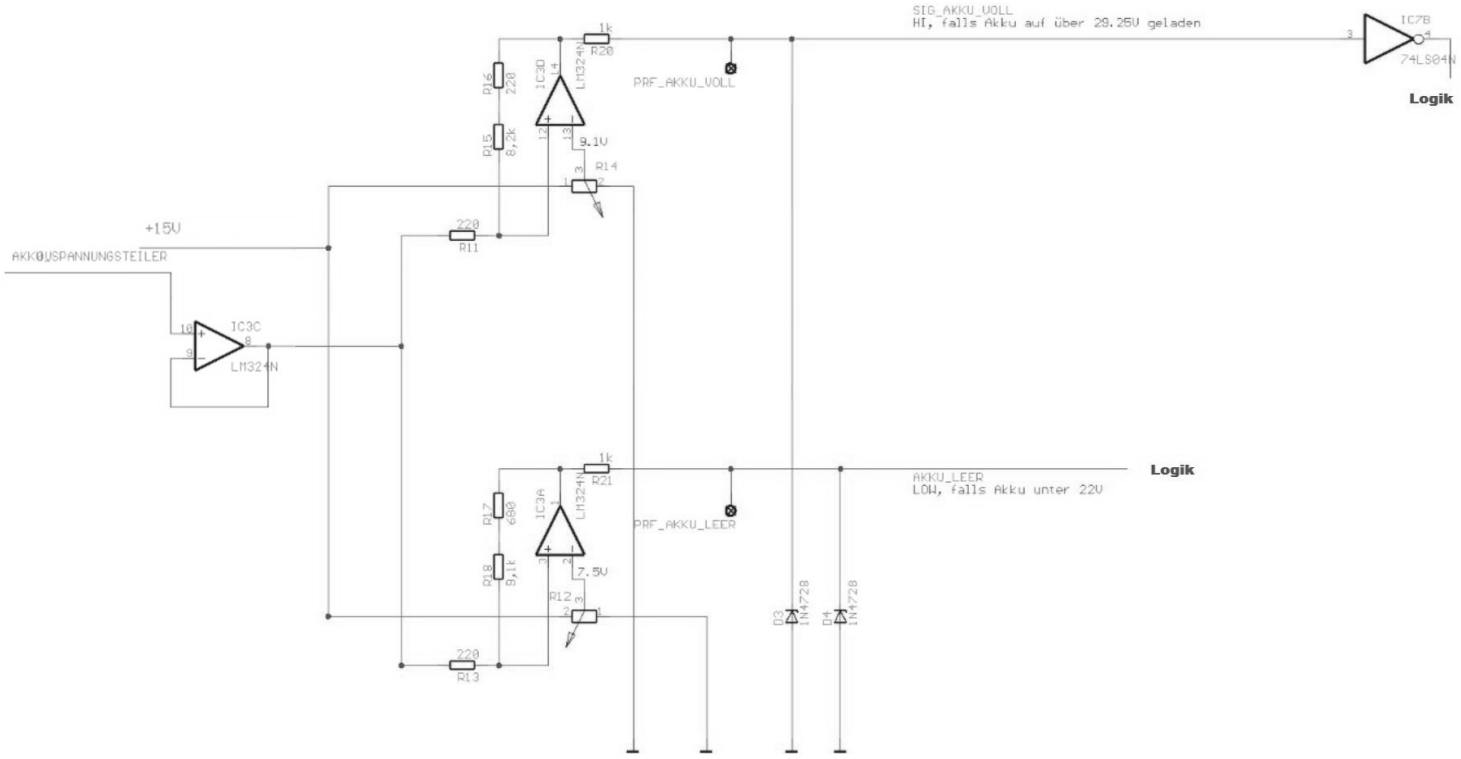


Abb.6: Eagle-Schematic

Quellen

http://www.elektronik-kompendium.de/sites/slt/0311271.htm
PSpice – Student
Eagle
http://www.krucker.ch/DiverseDok/Schmitt%20Trigger.pdf

Modul – Temperaturmessung

Aufgabe

Die Aufgabe des Moduls „Temperaturmessung“ ist, die Temperatur der Akkus zu messen und die gewonnenen Daten zu interpretieren, sodass diese Informationen dann später in der Logik verarbeitet werden können.
Es ist sehr wichtig die Temperatur der Akkus zu messen, da sie bei bestimmten Temperaturen nicht mehr die volle Leistung liefern können, bzw. sogar beschädigt oder zerstört werden können. Wir haben zwei kritische Grenztemperaturen festgelegt: -10°C und 40°C.
Die Kapazität der Akkus bei –10°C ist nur noch sehr gering und ein Betrieb des Spannungswächters bei Netzausfall ist bei solchen Temperaturen deshalb nur noch sehr kurz möglich. Bei über 40°C ist eine Beschädigung oder sogar die Zerstörung der Akkus möglich (Siehe Datenblatt Akkus!!!).
Die Temperatur wird mit einem Thermowiderstand, der zwischen beiden Akkus befestigt ist, gemessen. Es wird also die Oberflächentemperatur beider Akkus gleichzeitig gemessen. Zur Funktionsweise von Thermowiderständen siehe Referat „Thermowiderstände“.

Blockschaltbild des Moduls Temperaturmessung

Das Blockschaltbild zu diesem Modul ist sehr einfach: Das Thermoelement „misst“ die Temperatur und gibt die gewonnenen Daten an die Verarbeitung weiter. Hier wird entschieden, ob die Temperatur hoch genug oder niedrig genug ist. Dieses Signal wird dann an das Modul Logik gesendet und dort weiter verarbeitet.

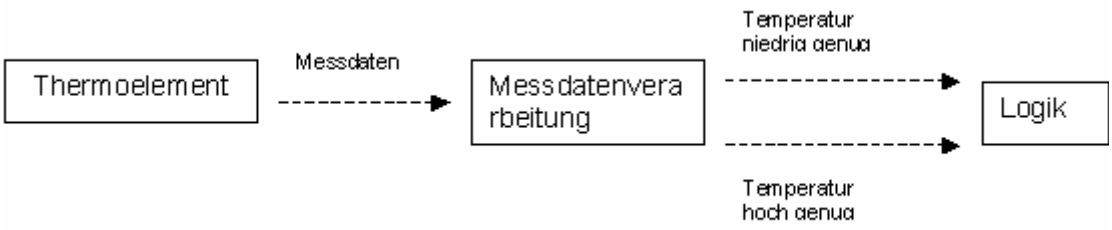


Abb. 6: Blockschaltbild

Ideen zur Messschaltung

Die erste Idee zur Realisierung der Schaltung sah folgendermaßen aus (Abb.2): Die Spannung über das Thermoelement (KTY) wird mit Hilfe zweier Komparatoren mit zwei „Schwellspannungen“ verglichen. Die Schwellspannungen werden dem Datenblatt des Thermoelements entnommen und dann mit zwei Spannungsteilern aufgebaut. Das analoge Signal der Komparatoren wird dann in ein digitales Signal umgewandelt, um dieses an die Logik zu senden

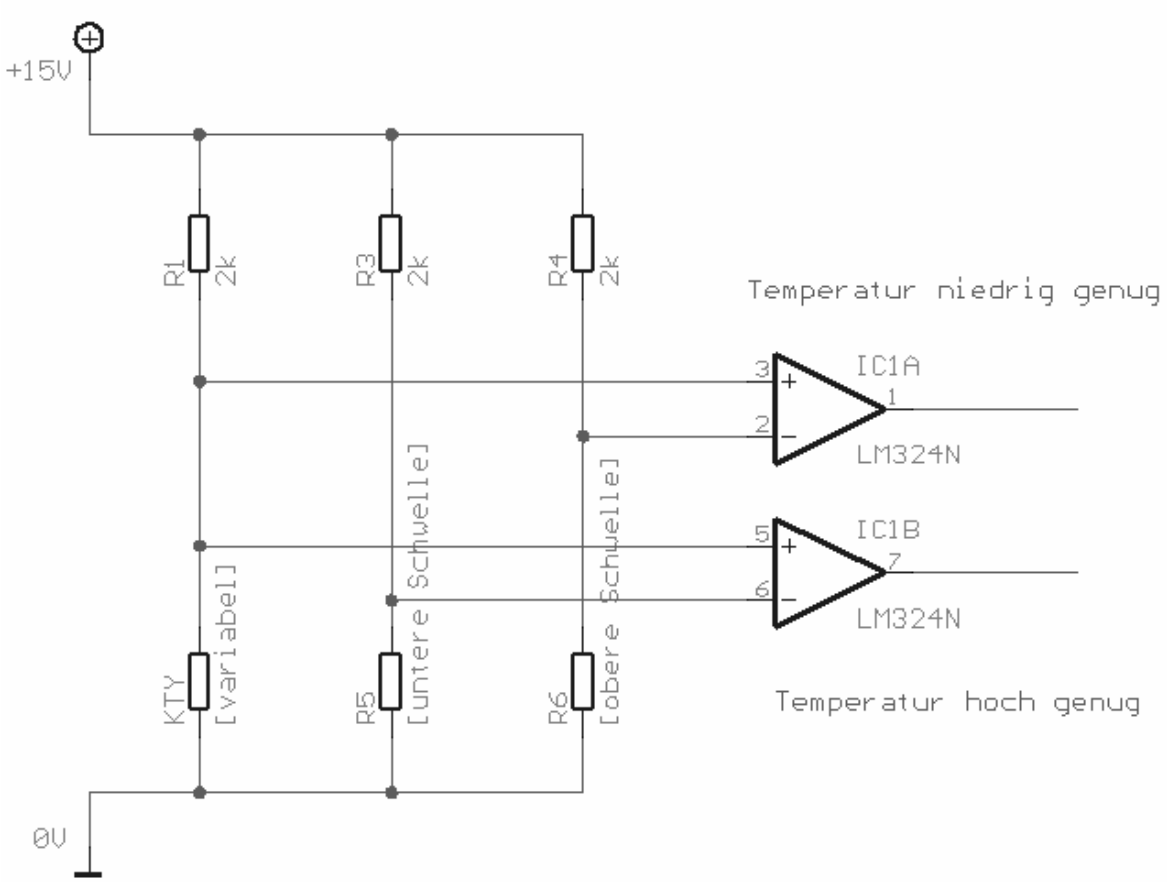


Abb. 7: Messschaltung

Prinzipiell funktioniert diese Schaltung bis auf ein paar Feinheiten recht gut, die endgültige Realisierung ging aber in die Richtung, den Widerstand über eine Messbrücke zu ermitteln, die so dimensioniert ist, dass die Differenzspannung U_D an der unteren Schwelle gleich null ist und bei der oberen einen Maximalwert annimmt. Die Differenzspannung U_D wird dann mit einem Differenzverstärker verstärkt, da sie sonst zu klein ist, um sie vernünftig auswerten zu können (siehe Abb.3).
Dahinter wird dann wiederum mit Komparatoren ermittelt, ob die Spannung zu klein, oder zu groß, d.h. „hoch genug“ oder „niedrig genug“ ist.
Es wurde vereinbart, dass wenn die Temperatur im erlaubten Bereich ist, das Signal an beiden Ausgängen auf High gesetzt ist. Wir wollten damit verhindern, dass wenn etwas an der Schaltung kaputt geht, die Temperaturmessung nicht einfach übergangen wird, sondern dass man darauf aufmerksam wird und den Schaden beheben kann.

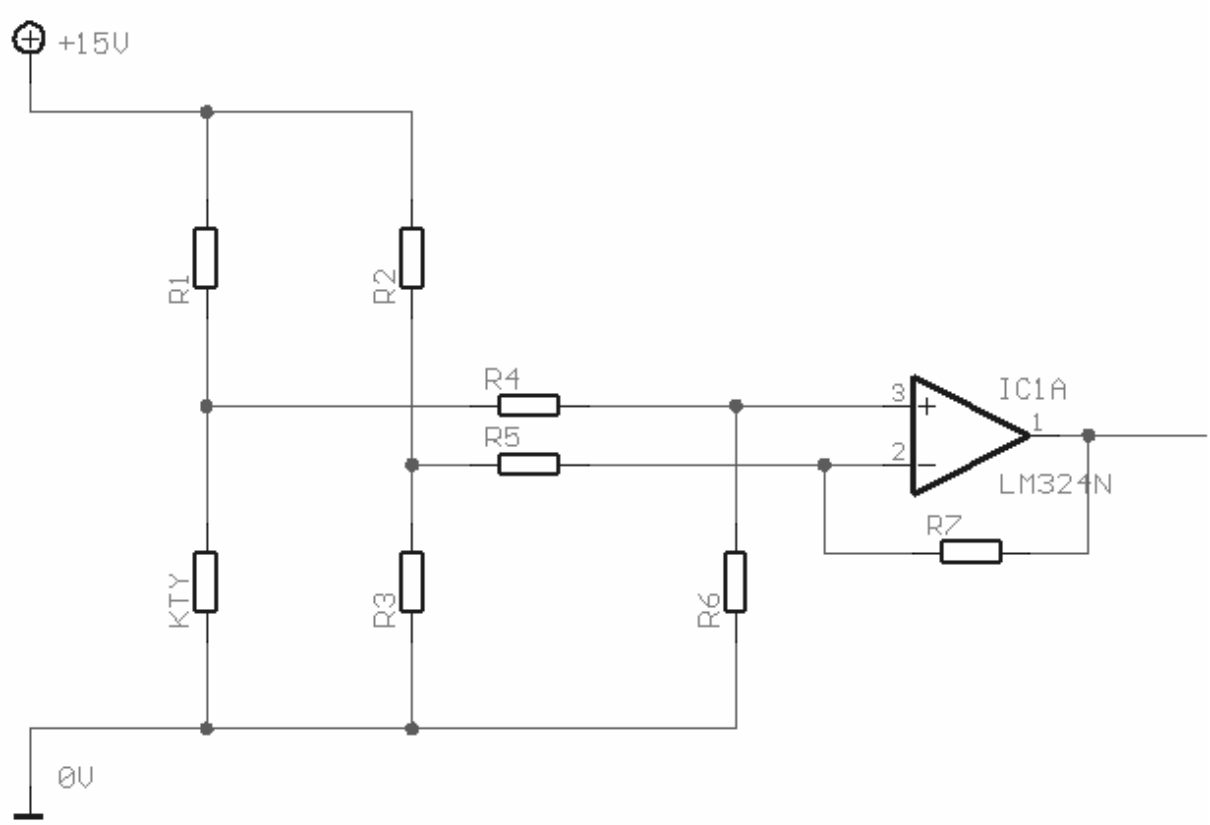


Abb. 8: Differenzverstärker

Um das Signal des Thermoelements sowie des anderen Spannungsteilers vom Differenzverstärker zu entkoppeln, muss vor R4 und R5 ein Impedanzwandler gesetzt werden.

Der letzte Teil der Schaltung könnte nun wieder mit zwei Komparatoren aufgebaut werden. Da jedoch zu erwarten ist, dass die Temperatur über längere Zeit konstant bleibt, bzw. sehr gering schwanken könnte und damit die Komparatoren anfangen könnten zu schwingen, haben wir sie durch Schmitt-Trigger ersetzt und einen Hysteresebereich von ±5°C vorgesehen.
Der letzte Teil der Schaltung knüpft am Ausgang des Differenzverstärkers aus Abb. 3 an. Es werden ein invertierender Schmitt-Trigger für das Signal „Temperatur niedrig genug“ und ein nicht invertierender Schmitt-Trigger für das Signal „Temperatur niedrig genug“ benutzt wie in Abb. 4 zu erkennen ist. Da der LM324 aber als Ausgangsspannung ± 15V hat, muss diese noch an die TTL-Logikbausteine angepasst werden, sodass als High-Pegel +5V und als Low-Pegel 0V weitergegeben werden. Das wird mittels einer Z-Diode realisiert. Das Signal ist nun vollständig mit der TTL-Logik kompatibel.
Um Ungenauigkeiten durch Widerstandstoleranzen bei den Spannungsteilern zu minimieren haben wir hier Spindeltrimmer eingesetzt, die beim in Betrieb nehmen sehr genau eingestellt werden können.
Die komplette Schaltung ist in Abb. 5 zu sehen.

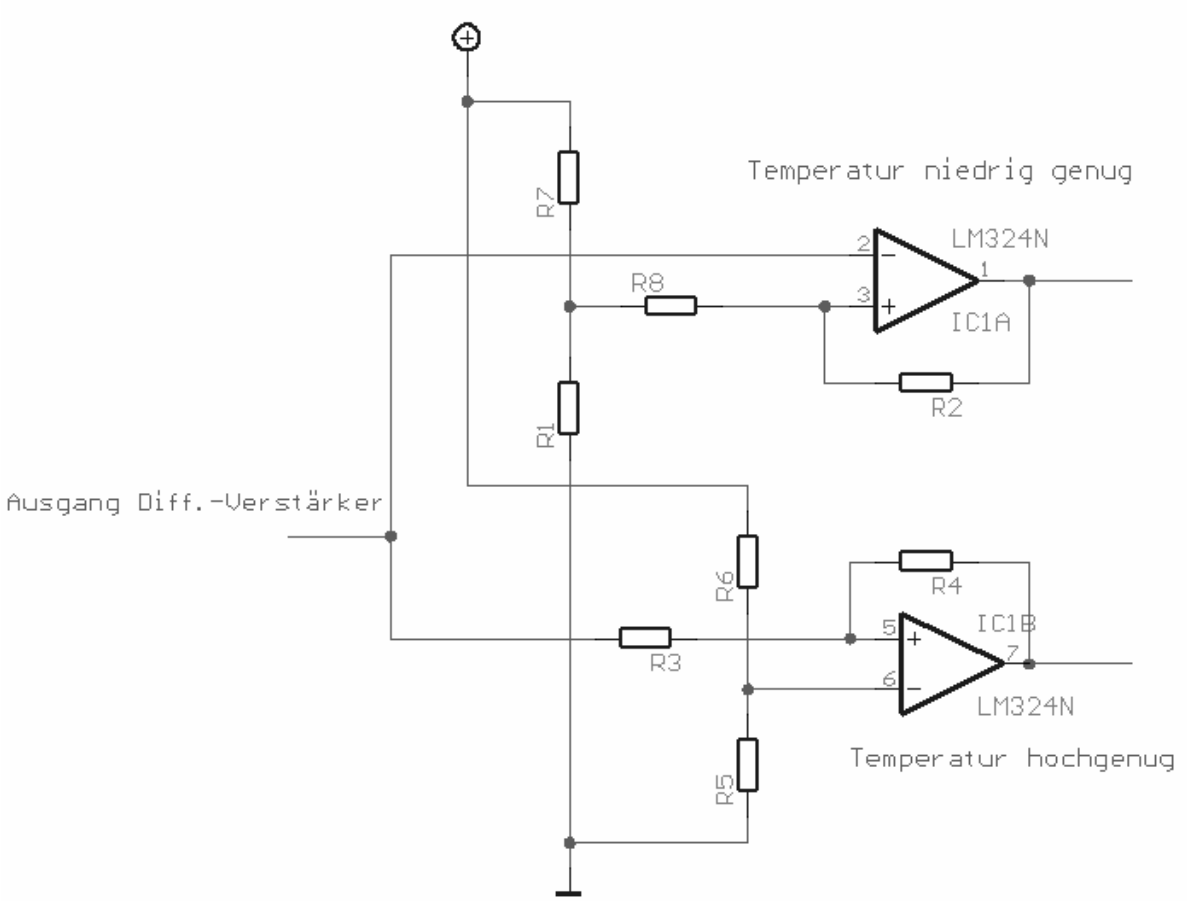


Abb. 9: Schmitt-Trigger

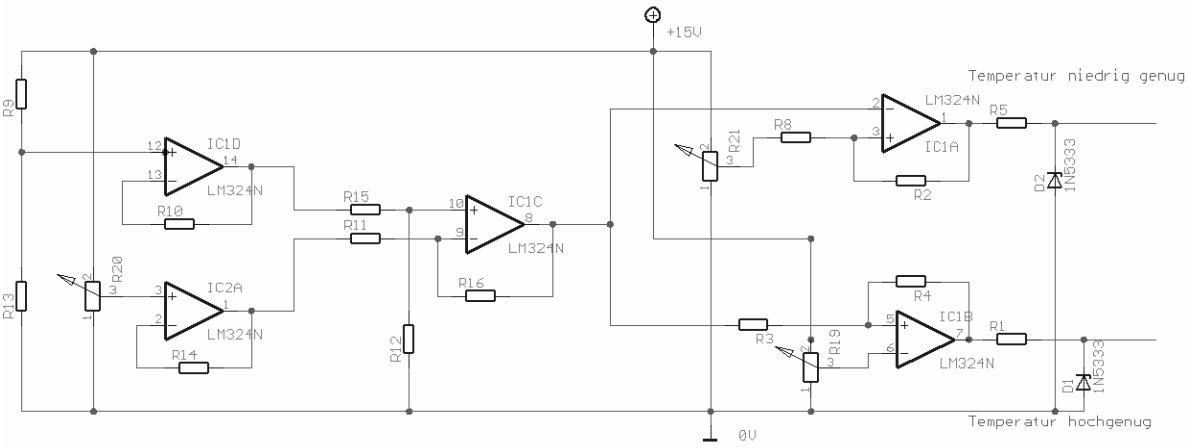


Abb. 10: Gesamtschaltung

Berechnung: der Temperaturmessung

Als erstes muss die Brückenschaltung berechnet werden. Als Grundlage dient hier die Schaltung aus Abb. 6 dessen Bezeichnungen benutzt werden. Obwohl der Spannungsteiler aus R2 und R4 mit einem Potentiometer realisiert wurde, muss er zur Berechnung wieder durch zwei Widerstände ersetzt werden, um die richtige Spannung ermitteln zu können.

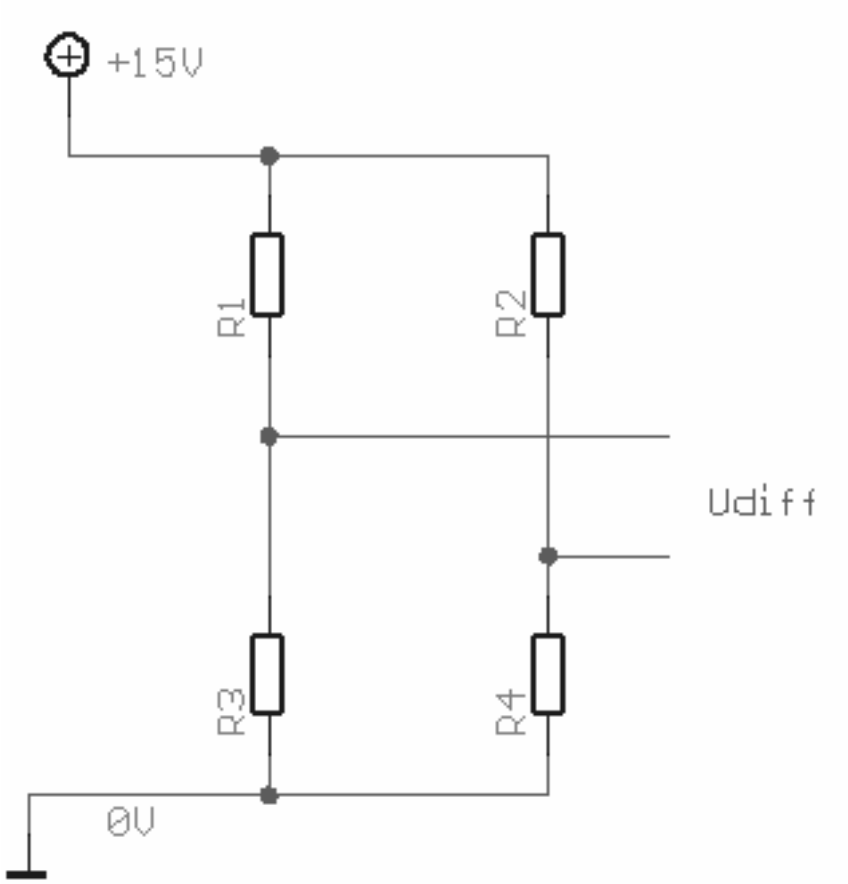


Abb. 11: Messbrücke

Die Widerstände R1 und R2 werden mit 2kΩ gewählt. Damit ergibt sich mit

$$\frac{U_b}{U_{R4}} = \frac{R2 + R4}{R4}$$

$$U_{R4} = \frac{R4}{R1 + R4} * U_b$$

Da die Differenzspannung bei –10°C U_b = 0V sein soll, wird aus dem Datenblatt des KTY entnehmen, dass der Widerstand bei –10°C 1,495kΩ beträgt. Der Widerstand R4 muss also ebenfalls diesen Wert haben. Damit ergibt sich die Spannung

$$U_{R4} = 6{,}42V = U_{R3}\,|_{-10°C}\,.$$

Nun wird noch die Spannung $U_{R3}\,|_{40°C}$ berechnet. Mit

$$\frac{U_{R3\,|\,40°C}}{U_b} = \frac{R3}{R1 + R3}$$

folgt:

$$U_{R3\,|\,40°C} = \frac{R3}{R1 + R3} * U_b \,.$$

Der Widerstand bei 40°C beträgt 2,245kΩ, eingesetzt ergibt das:

$$U_{R3\,|\,40°C} = 7{,}93V \,.$$

Die Differenzspannung U_{Diff} ergibt sich durch:

$$U_{Diff} = U_{R3\,|\,40°C} - U_{R4} = 1{,}51V \,.$$

Als nächstes wird der Differenzverstärker berechnet. Als Grundlage dient die Schaltung aus Abb. 7:

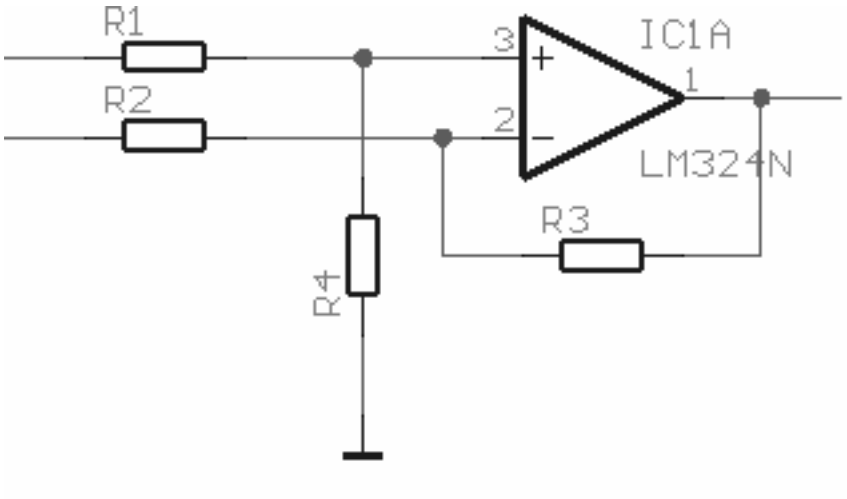


Abb. 12: Differenzverstärker

Für den Differenzverstärker gilt:

$R1 = R2$ und
$R3 = R4$

Die Verstärkung ergibt sich aus folgender Gleichung:

$$v = \frac{R3}{R1} \,.$$

Um eine Verstärkung von 5,6 zu erreichen, wird

$R1 = R2 = 1k\Omega$ und
$R3 = R4 = 5{,}6k\Omega$

gewählt.

Die Verstärkung v ist damit:

$$v = \frac{R3}{R1} = 5{,}6\,.$$

Somit ist die Ausgangsspannung des Differenzverstärkers bei 40°C nun

$$U_D = U_{Diff} \mid_{40°C} * v = 8{,}456V.$$

Die Hysterese der Schmitt-Trigger soll eine Weite von ±5°C haben, das bedeutet, dass noch die Werte U_{T+} und U_{T-} bestimmt werden müssen.
Hierzu werden die benötigten Widerstandswerte wieder aus dem Datenblatt entnommen:

$$U_{R3} \mid_{-10°C} = 6{,}42V \Rightarrow U_{Diff} = 0V$$
$$U_{R3} \mid_{0°C} = 6{,}59V \Rightarrow U_{Diff} = 0{,}16V$$
$$U_{R3} \mid_{35°C} = 7{,}79V \Rightarrow U_{Diff} = 1{,}37V$$
$$U_{R3} \mid_{40°C} = 7{,}93V \Rightarrow U_{Diff} = 1{,}51V$$

Hinter dem Differenzverstärker ergeben sich also folgende Spannungen:

$$U_{T+,\,inv.} = 8{,}42V$$
$$U_{T-,\,inv.} = 7{,}69V$$
$$U_{T+,\,n\ inv.} = 0{,}89V$$
$$U_{T-,\,n\ inv.} = 0V$$

Mit diesen Werten werden jetzt die Schmitt-Trigger dimensioniert. Die Komponenten des invertierenden Schmitt-Triggers (siehe Abb. 7), der das Signal „Temperatur niedrig genug“ generiert, lassen sich wie folgt berechnen:

$$U_v = \frac{U_{Sat} * (U_{T+} + U_{T-})}{2 * U_{Sat} - U_{T+} + U_{T-}}$$

$$R1 = \frac{R2 * (U_{T+} + U_{T-})}{2 * U_{Sat} - U_{T+} + U_{T-}}\,.$$

Wobei U_{Sat} die Spannung ist, die der Komparator am Ausgang liefert, wenn er die Sättigung erreicht. Es wird angenommen, es handelt sich um einen idealen Operationsverstärker, also kann für U_{Sat} die Versorgungsspannung mit ±15V angenommen werden. Nun wird R2 geeignet gewählt mit 390kΩ. Somit ergibt sich für U_v und R1:

$$U_v = 8{,}30\text{V},$$

$$R1 = 8936{,}17\Omega.$$

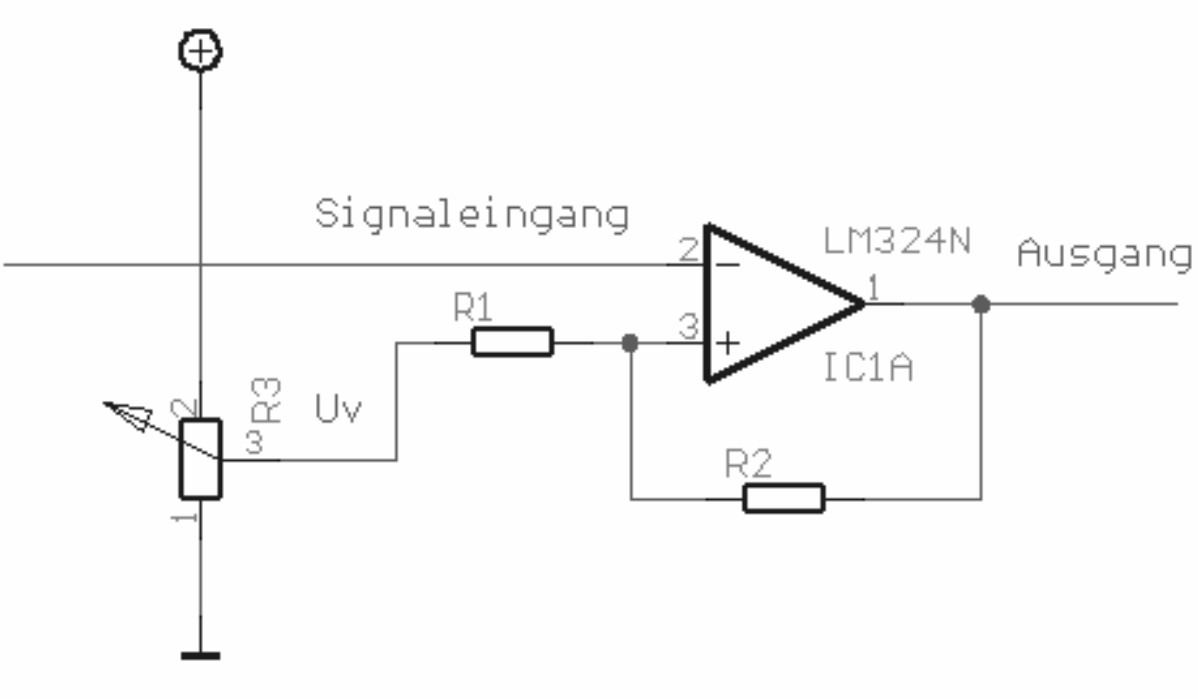


Abb. 13: Invertierender Schmitt-Trigger

Das gleiche muss anschließend für den nicht invertierenden Schmitt-Trigger (Abb. 8) durchgeführt werden. Nun müssen zur Berechnung allerdings andere Formeln benutzt werden:

$$U_v = \frac{U_{Sat} * (U_{T+} + U_{T-})}{2 * U_{Sat} + U_{T+} - U_{T-}},$$

$$R1 = \frac{R2 * (U_{T+} - U_{T-})}{2 * U_{Sat}}.$$

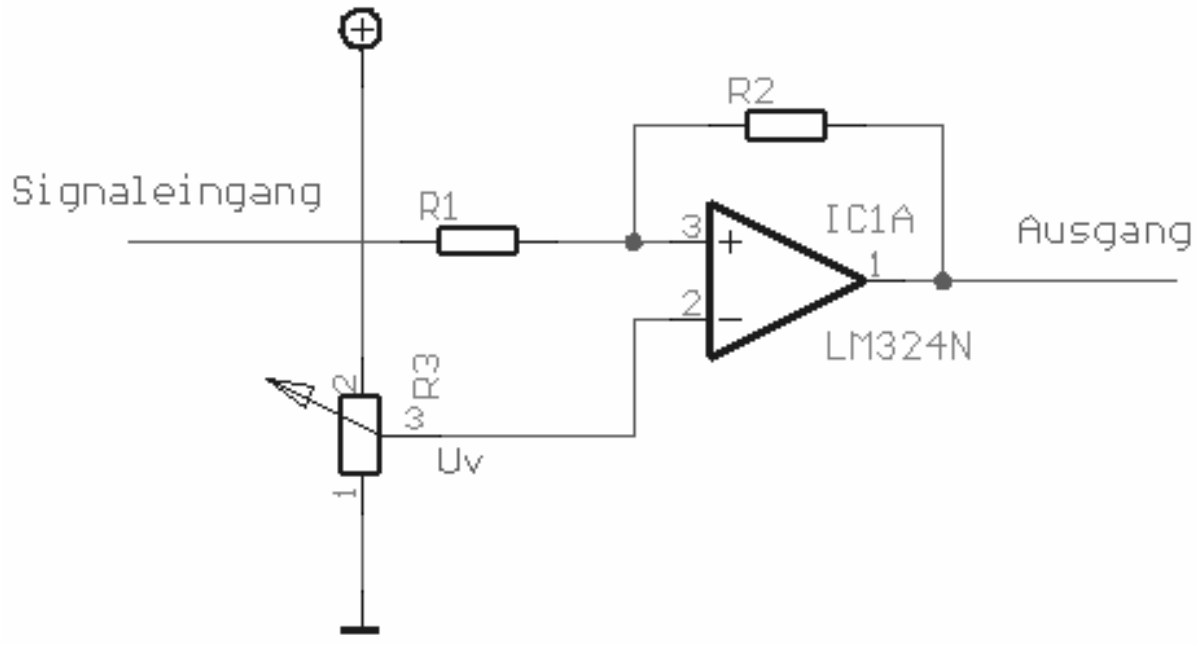


Abb. 14: Nicht invertierender Schmitt-Trigger

Auch hier wird für die Sättigungsspannung U_{Sat} angenommen, dass sie ungefähr der Versorgungsspannung entspricht. Für R2 wird wieder ein geeigneter Wert mit 180kΩ vorgegeben. Damit ergibt sich für U_v und R1:

$U_v = 446{,}31\text{mV}$,

$R1 = 5520\Omega$.

Simulation in PSpice

Jetzt, da alle erforderlichen Daten vorhanden sind, kann die Schaltung in PSpice aufgebaut und dort getestet werden. Die Spannungen U_v sind hier durch normale Spannungsquellen ersetzt. Im realen Aufbau müssen sie natürlich durch Widerstände, oder besser noch, durch Potentiometer ersetzt werden.

Die erste Schaltung und Simulation (Abb. 10 und 11) zeigen den ersten Teil der Schaltung bis zum Ausgang des Differenzverstärkers. Bei der Simulation wird der Widerstand, der den KTY darstellen soll, mit der Zeit verändert, es folgt das R-t-Diagramm in Abb. 11.

Der zweite Teil der Schaltung wird anders getestet. Wie Abb. 12 zeigt, wird das Eingangssignal mit einer Sinuswelle simuliert, um die Umschaltvorgänge in Abb. 13 ausreichend darzustellen.

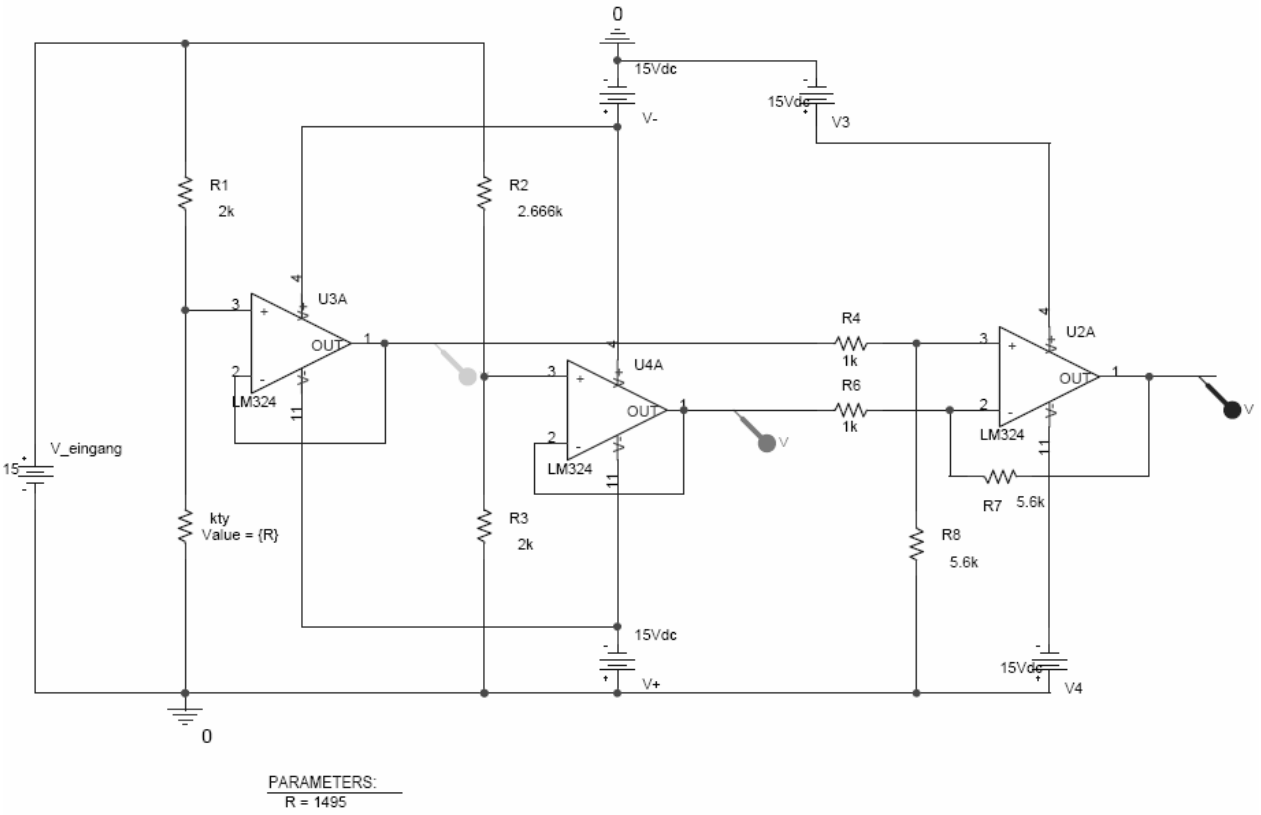


Abb. 15: PSpice Aufbau 1. Teil

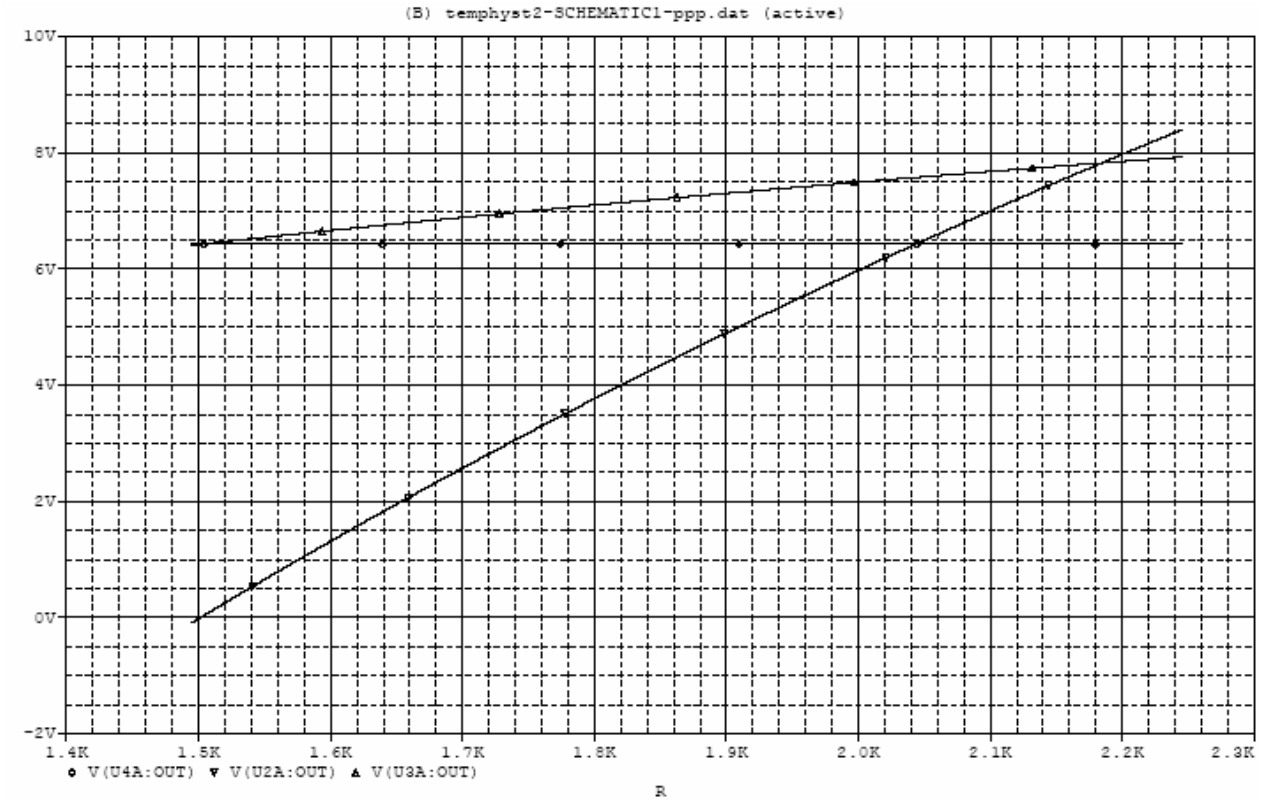


Abb. 16 PSpice Simulation Teil 1

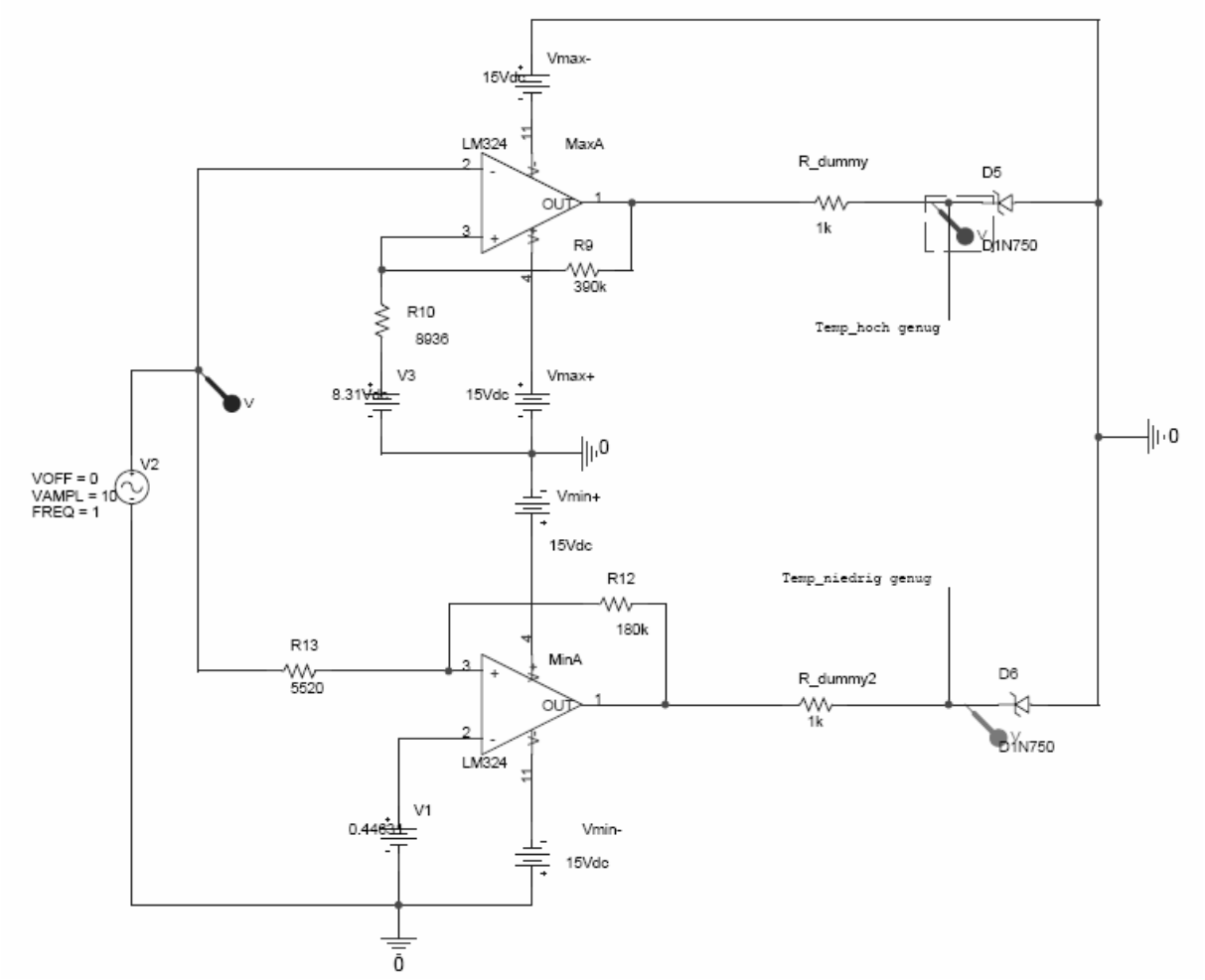


Abb. 17: PSpice Aufbau Teil 2

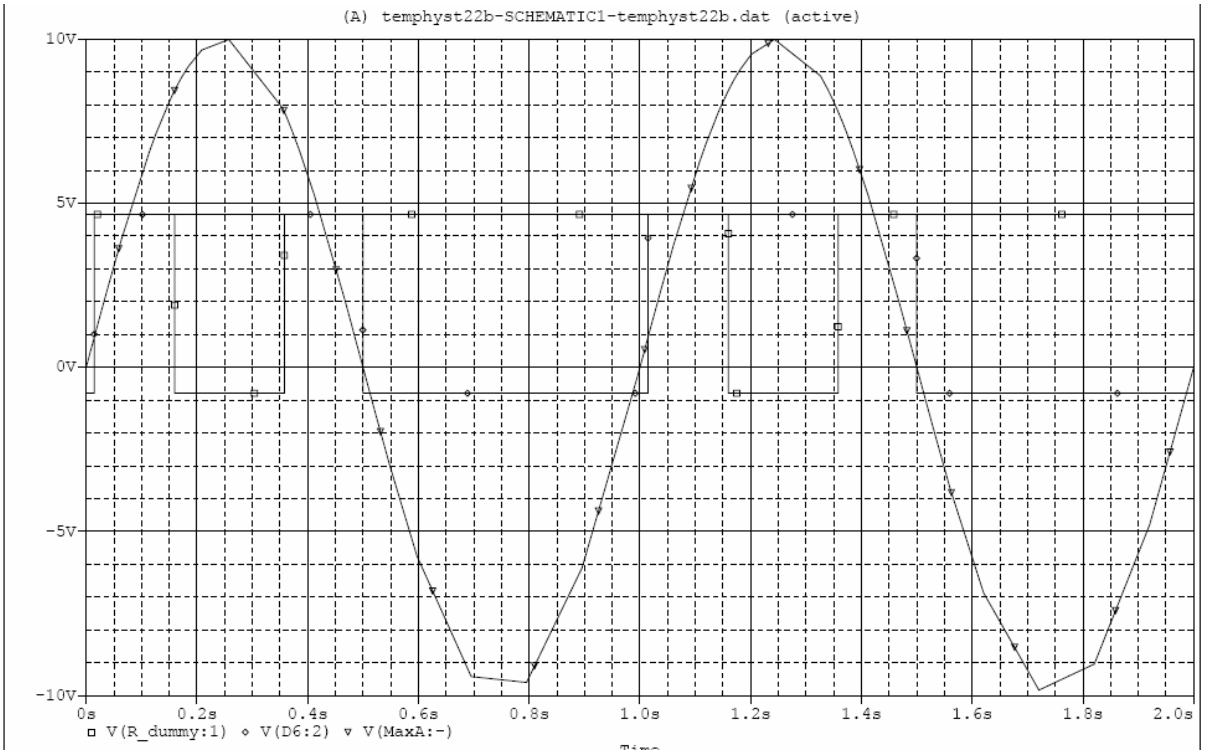


Abb. 18: PSpice Simulation Teil 2

Schnittstellen

Eingänge: +15V
-15V
+5V
GND
Sig NETZ (Singal Strom da?)

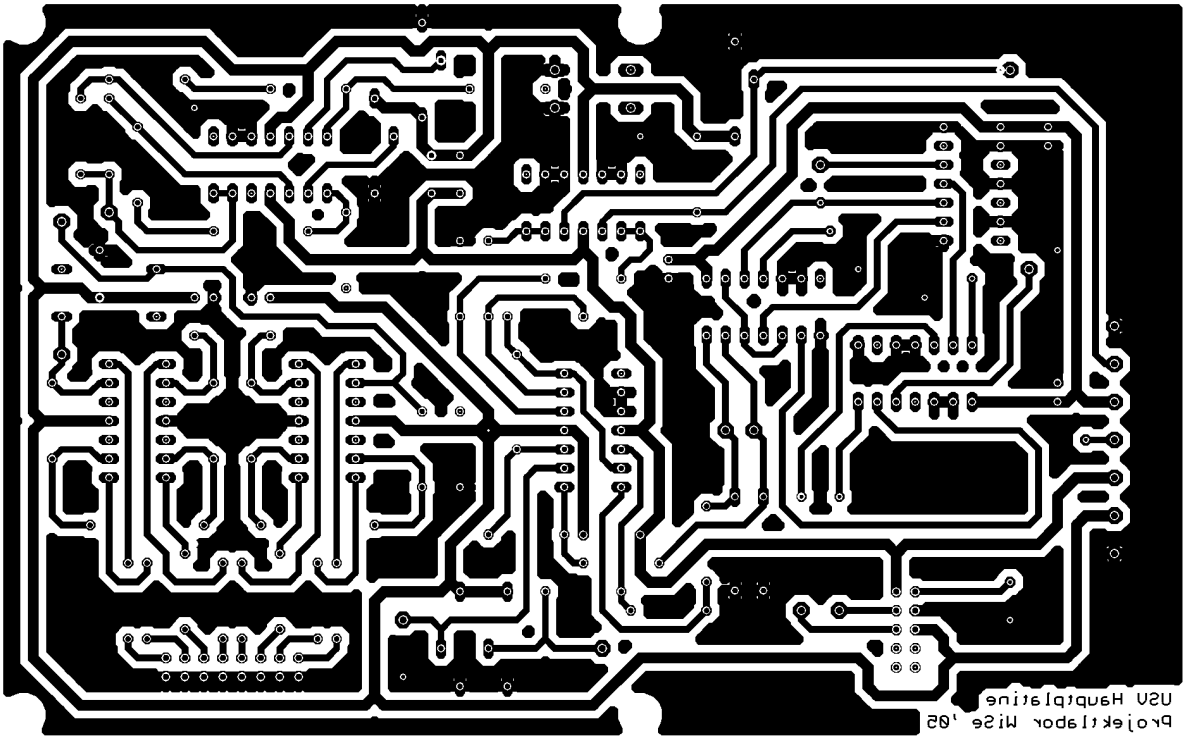
Stückliste

8 x 100nF
2 x 1N4728
2 x ZDIODE-12.5
5 x LM324N
1 x 74LS04N
2 x 74LS08N
1 x KTY
10 x 200 Ω
1 x 10k Ω
1 x 8,2k Ω
1 x 680 Ω
1 x 9,1k Ω
3 x 5,6k Ω
6 x 1k Ω
8 x 820 Ω
1x 180k Ω
1 x 390k Ω
1 x 9k Ω
1 x 2k Ω
1 x 2.666k Ω

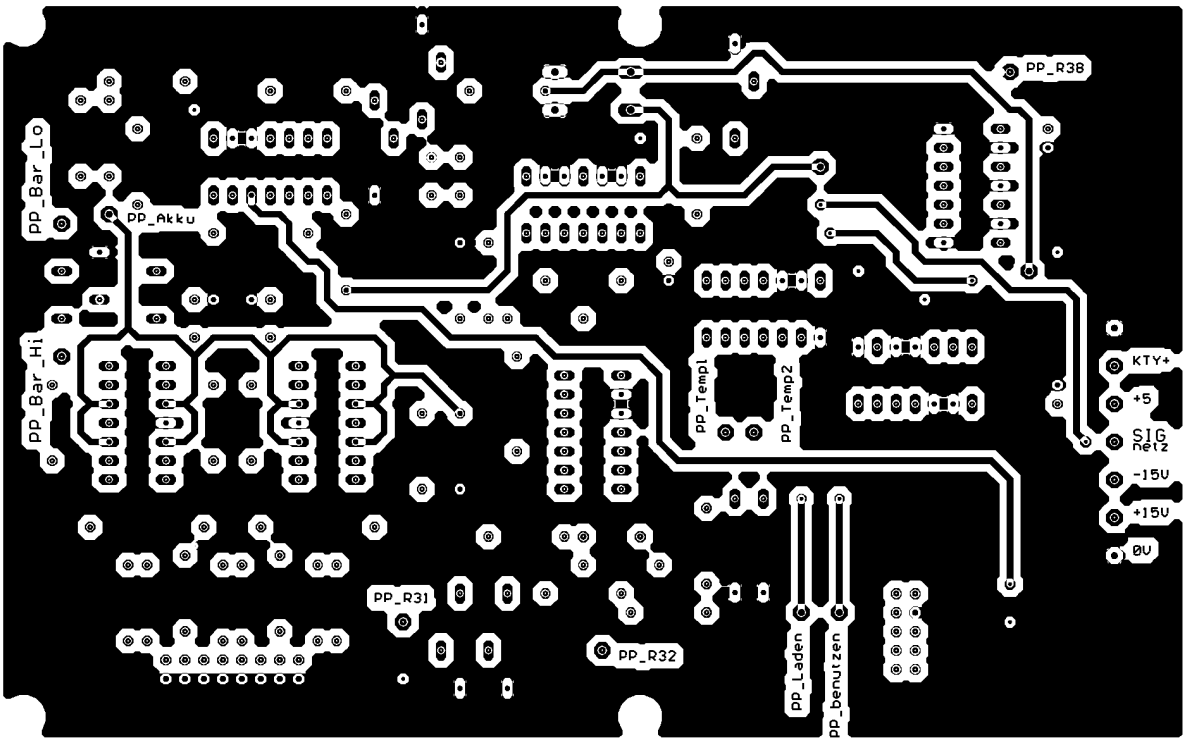
2 x 3 Pin Steckverbinder
1 x 14 Pin Steckverbinder

Ätzlayouts

Bottomlayer:



Toplayer:



6.3. Akku-Bargraph

Aufgabebeschreibung

Die Bargraphsanzeige zeigt dem Benutzer den Spannungszustand des Akkus.
Zur Realisierung unserer Bargraphschaltung, haben wir die Idee gehabt, eine Komparatorschaltung zu benutzen.

Funktionsweise

.Die Eingangsspannungen sind direkt an die Eingänge des OPV gelegt (siehe Abbildung 1). Am Ausgang wird die verstärkte Differenzspannung geliefert. Der OPV wird so beschaltet, dass die Ausgangsspannung null wird, falls die Eingangsspannug kleiner als die Referenzspannung ist. Eine Leuchtdiode wird in Reihe mit dem Komparator geschaltet. Das heißt, dass die Diode nur leucht, wenn die Eingangsspannung größer als die Referenzspannung ist. Der Vorwiderstand sorgt dafür, dass die Leuchtdiode nicht durch hohe Strom zerstört wird.

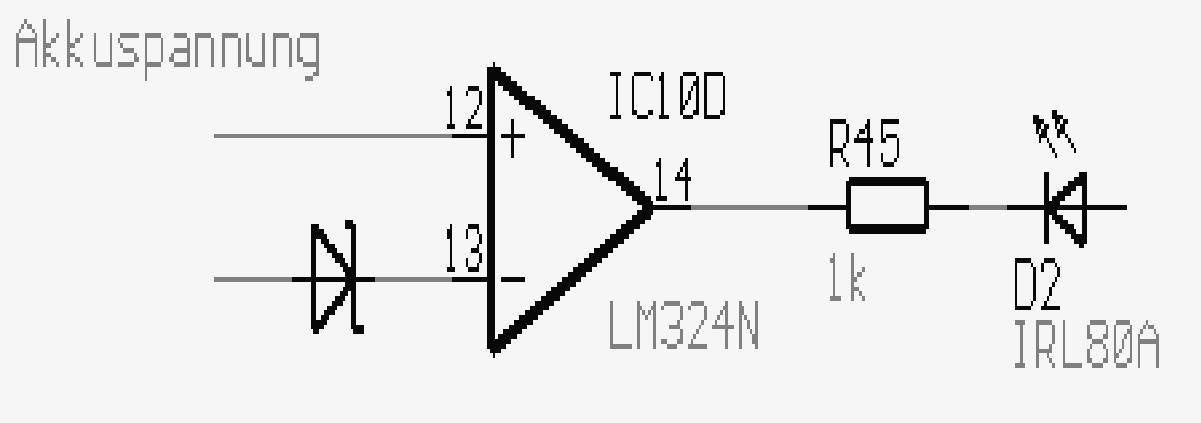


Abbildung 115: OPV mit LED

Für unsere gesamte Schaltung wird die Akkuspannung mit 8 Referenzspannungen verglichen (siehe Abbildung 2). Die Referenzspannungen sind mit Hilfe des Spannungsteilers aufgebaut. Auf der Abbildung steht einen Stecker, der zur LED-Platine führt.

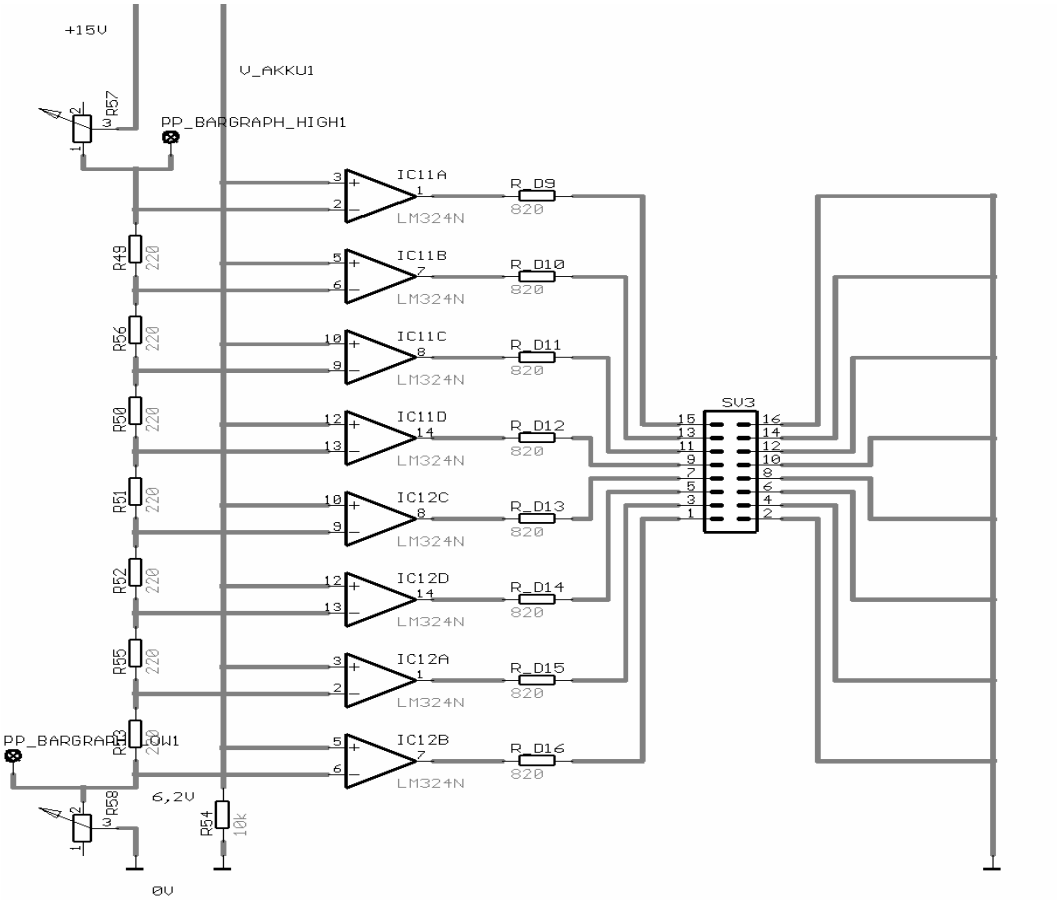


Abbildung 116: Bargraphansteuerung

PSpice Simulation

Beispielhaft wurde eine Simulation für 2 stufen mit Pspice-Simulation aufgebaut.

Die Bargraphansteuerung ist beispielhaft für den ersten und letzten Wert.

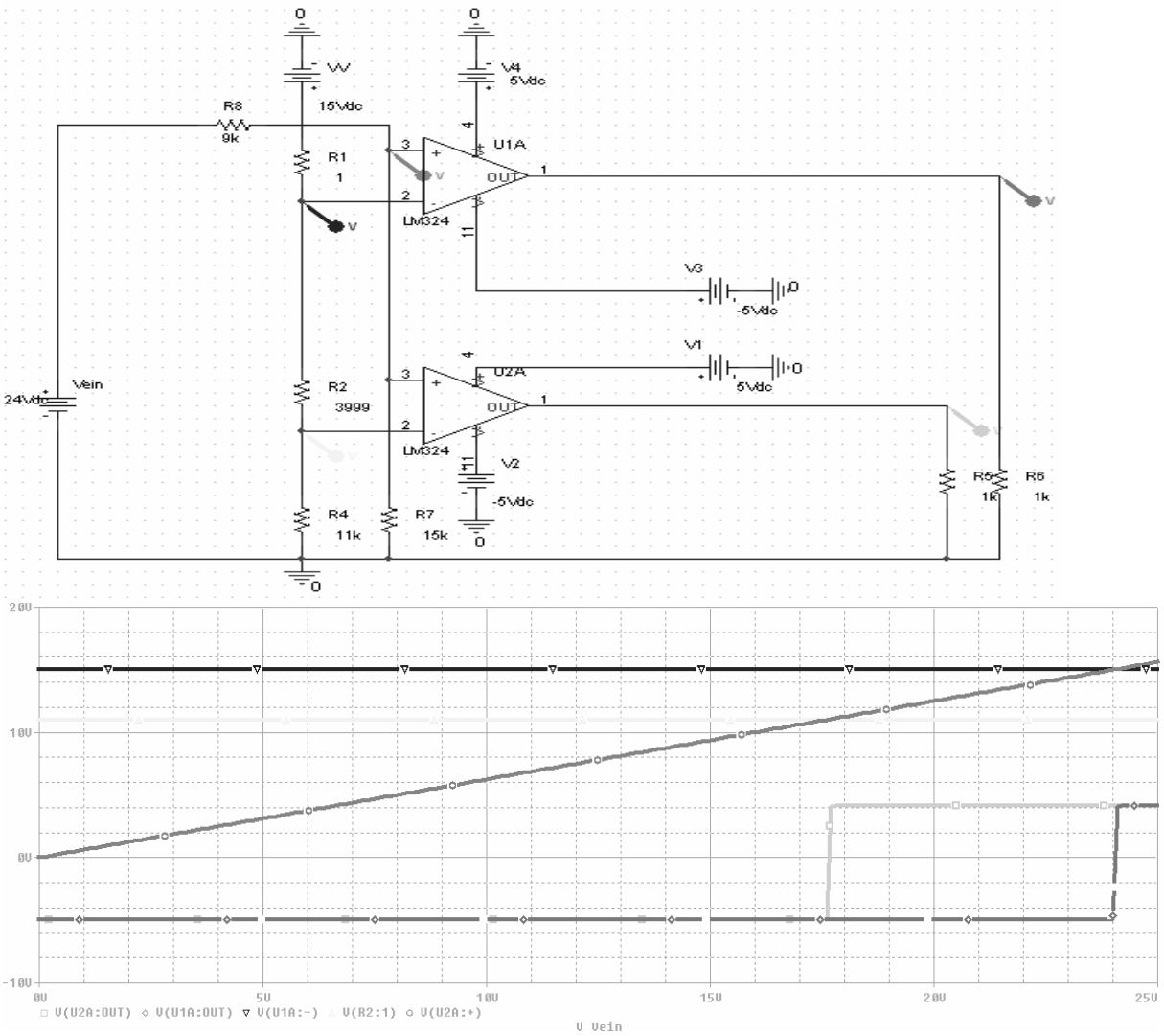
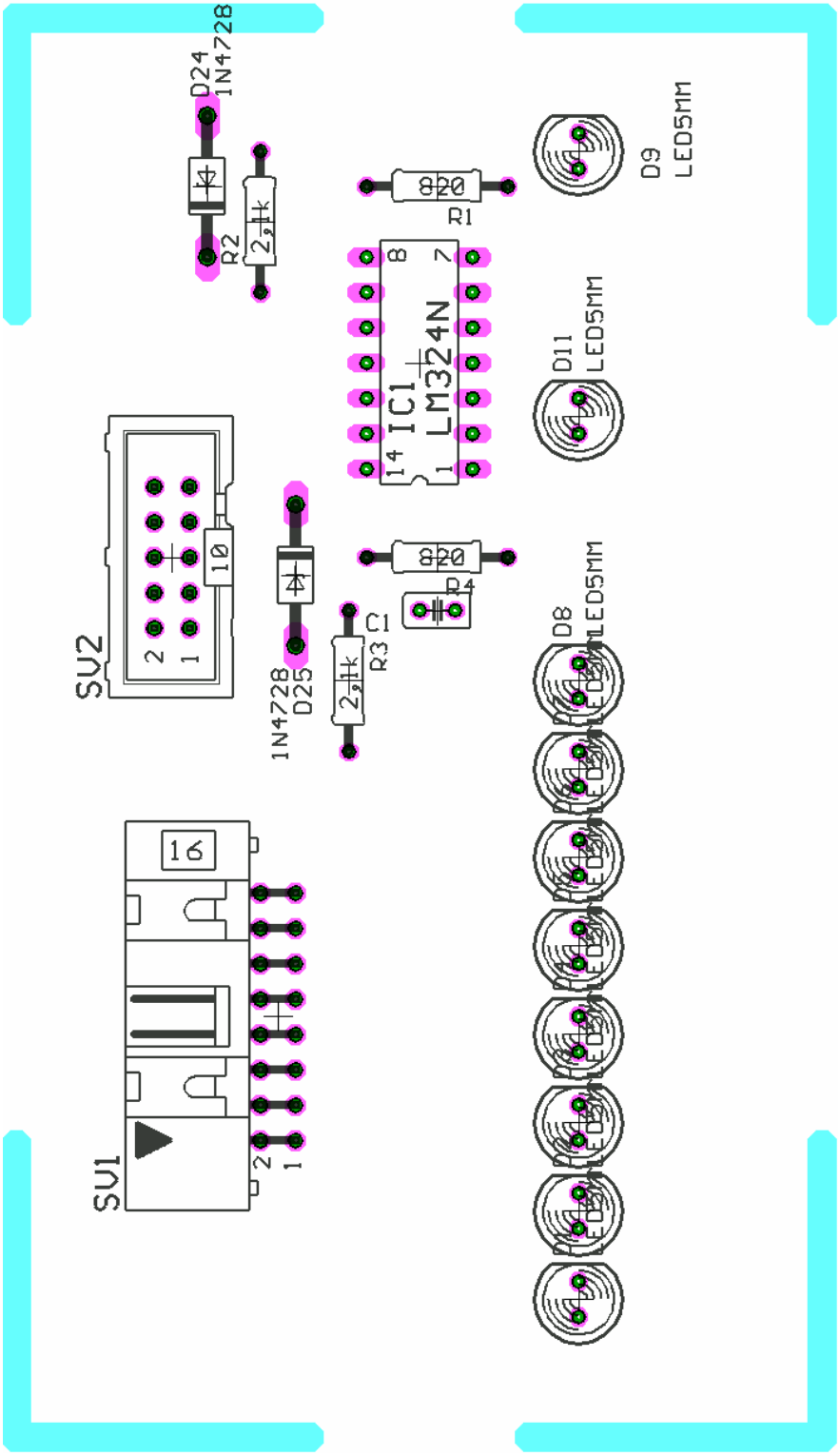


Abbildung 117: PSpice Simulation

Auf der Simulationsabbildung sind die Akkuspannung, die Referenzspannungen und die Ausgangspannungen zu sehen. Die Ausgangspannung bleibt null und konstant solange die Akkuspannung kleiner als die Referenzspannung und spricht sobald, dass die Akkuspannung größer als die Referenzspannung wird.
Die Schaltung der Bargraphanzeige wurde aufgebaut und erfolgreich getestet.

Stückliste



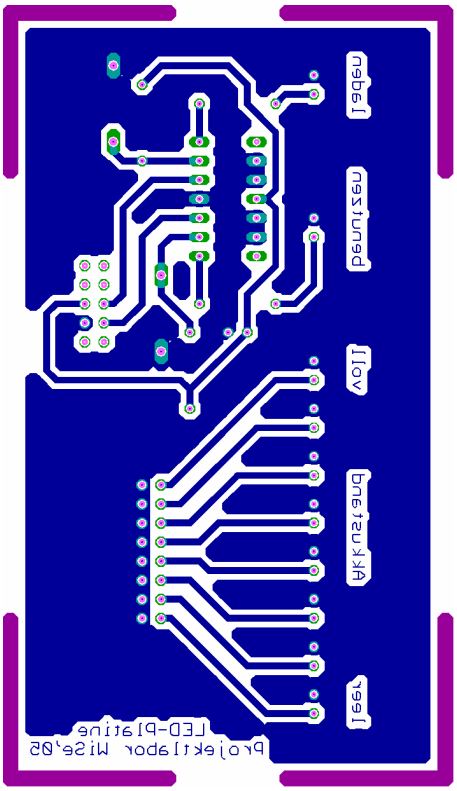
10 x LED
2 x 1N4728 Ω
1 x LM324N Ω
2 x 820 Ω
2 x 2,1k Ω
1 x 100nF
1 x 14 Pin Stecker

Hinweise

Die Vorwiderstände der LEDs befinden sich auf der Main-Platine.

Ätzlayouts

Bottom-Layer



6.4. Akkuversorgung

Um die verschiedenen Betriebsmodi der Notstromversorgung realisieren zu können, musste eine Schaltung entwickelt werden, die zwischen verschiedenen Beschaltungen des Akkumulators wechselt.

Dabei hatten wir zwischen drei Zuständen zu unterscheiden:

- Normalbetrieb
- Ladebetrieb
- Notstrombetrieb

Wir einigten uns darauf, die Umschaltung durch Relais zu lösen, die – im Gegensatz zu Halbleiterbauelementen – zwar relativ hohe Schaltzeiten aufweisen, dafür aber eine galvanische Trennung des Akkus vom Rest der Schaltung ermöglichen. Dies verhindert eine Entladung durch Restströme, die bei HL-Bauelementen immer vorhanden wären.

Strombegrenzung

Desweiteren war es nötig, im Ladebetrieb die Last für das Netzteil zu begrenzen. Ein ungebremster Ladevorgang hätte eine viel zu große Dimensionierung des Netzteils erforderlich gemacht. Daher entwarfen wir
eine Transistorschaltung die den Ladestrom in jedem Fall auf 500 mA
begrenzt. (Abbildung 118) Gleichzeitig stabilisiert diese Schaltung durch
die beiden Z-Dioden die Ladepannung auf 29V.

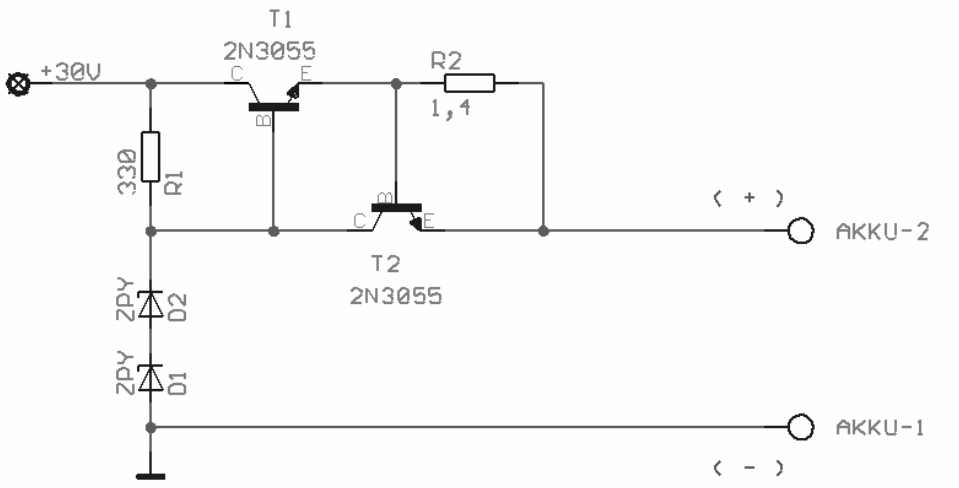


Abbildung 118 Ladestrombegrenzung

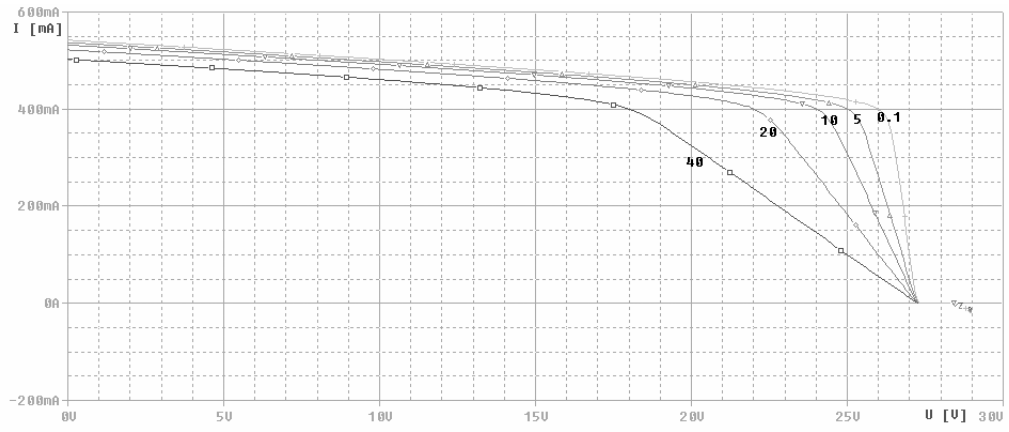


Abbildung 119 Simulation der Ladestrombegrenzung

Abgetragen ist der Ladestrom I über die Akkuspannung von 0 bis 30V.
Man erkennt, das der Ladestrom selbst bei sehr kleinem Innenwiderstand des Akkus niemals über eine bestimmte Größe anwächst, die Transistoren verhindern dies.
Wird der Strom durch R2, und damit die Basis- Emitter-Spannung von T2 zu groß, dann steuert dieser auf, und die Basis-Emitter-Spannung von T1 sinkt.
T1 begrenzt so den Strom der durch R2 fließen kann.

Relais

Bei den Relais handelt es sich um Monostabile Relais.
Sie schalten nur solange der On-Strom sie durchfließt, und springen in den Ausgangszustand zurück, wenn der Strom abbricht. Dies wirkt sich positiv auf das Betriebsverhalten aus, da im Falle einbrechender Betriebsspannung der Akku automatisch abgeschaltet wird, und sich nicht weiter entladen kann.
Um den Umschaltvorgang auszulösen muss in die Spulen der Relais ein Strom > 20 mA eingeprägt werden. Weil die Logikbausteine diesen nicht aushalten, mussten wir eine Treiberschaltung entwerfen, welche die TTL-Signale auswerten, und die Relais entsprechend schalten sollte.

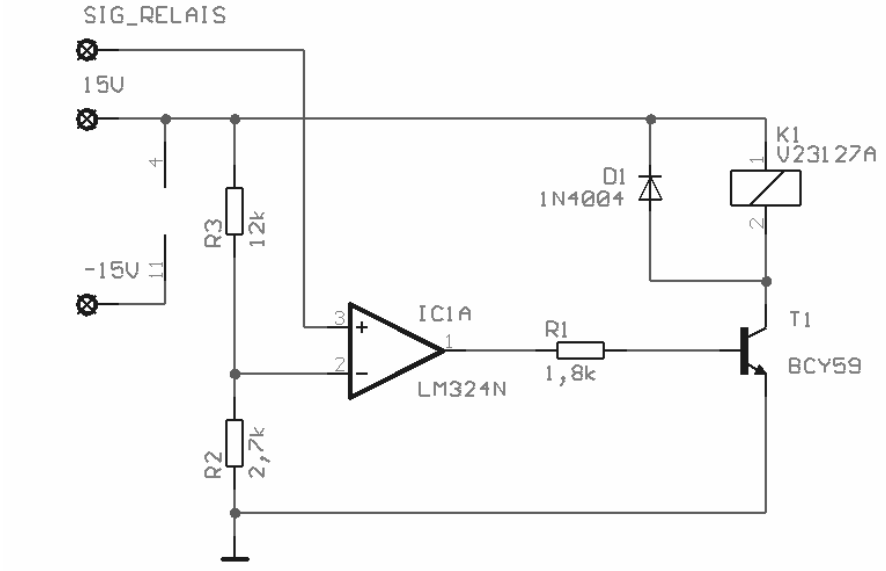


Abbildung 120

Zum Betrieb der Relais war in frühen Versionen der Schaltung ein Operationsverstärker (LM324N) vorgesehen. Allerdings stellten wir fest, das der für die Schaltvorgänge notwendige Strom immer noch zu hoch war, und der OPV sich infolge dessen stark erwärmte. Um dies zu umgehen, fügten wir Transistoren in die Schaltung ein, die die Signale des OPV noch einmal verstärken und die Relais ansteuern sollten. (***Fehler! Verweisquelle konnte nicht gefunden werden.***)

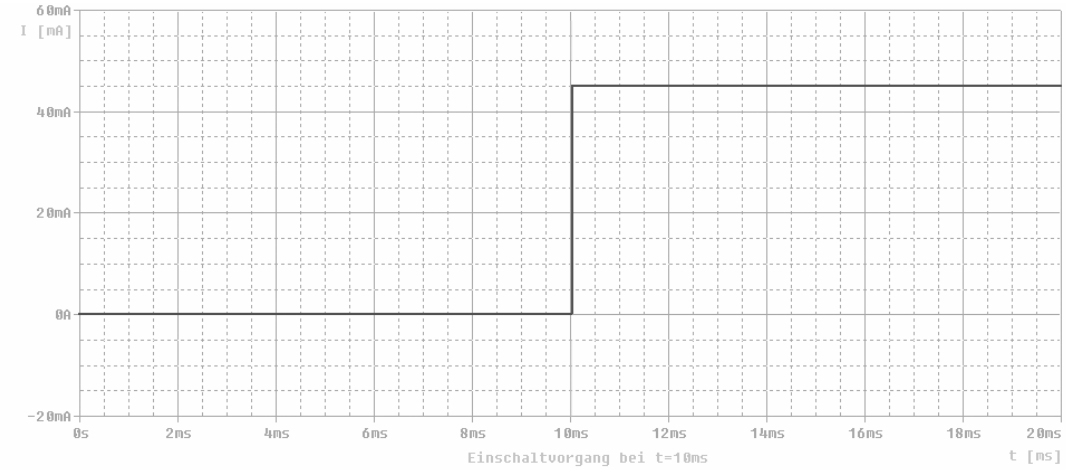


Abbildung 121 Simulation der Treiberschaltung

Schaltplan

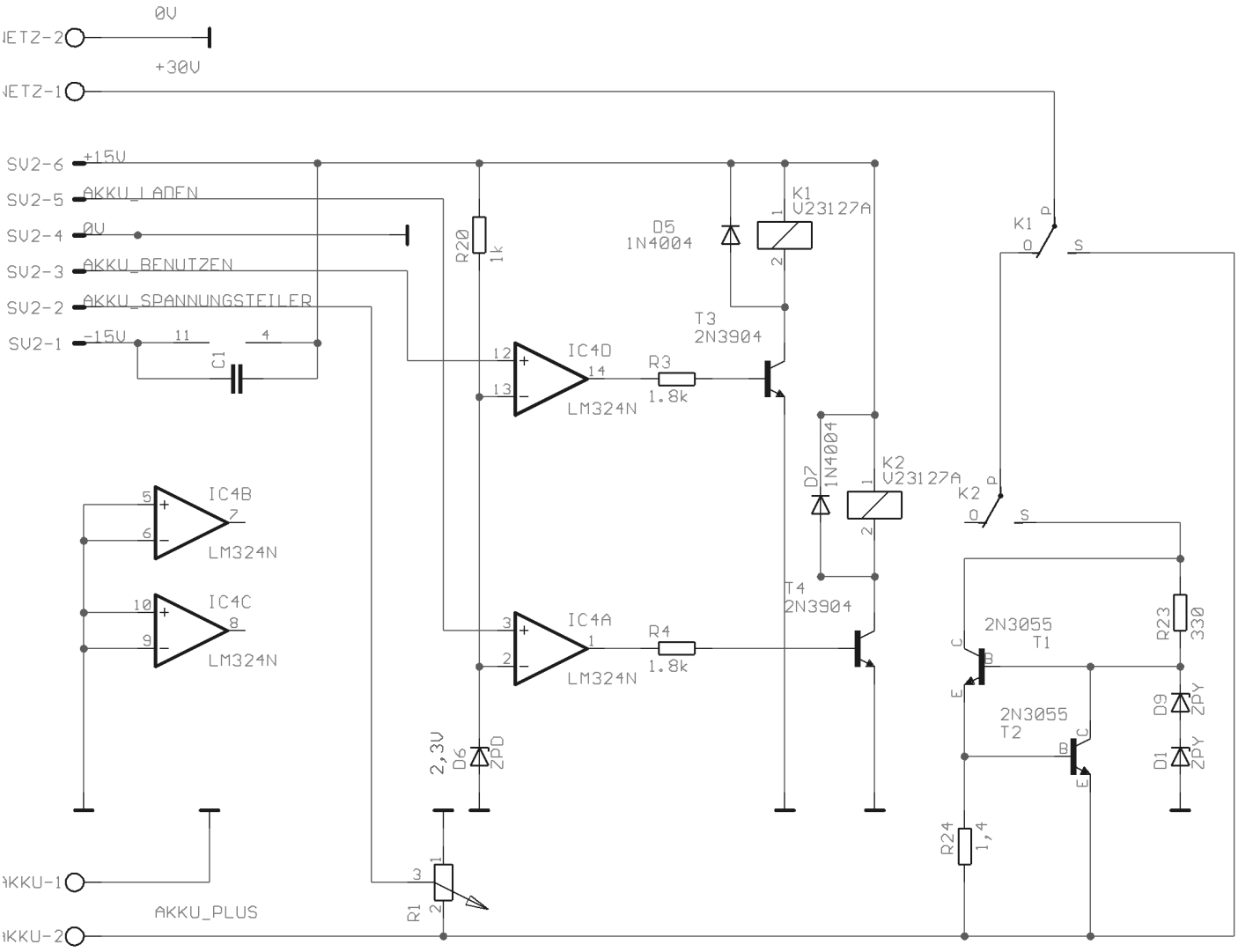
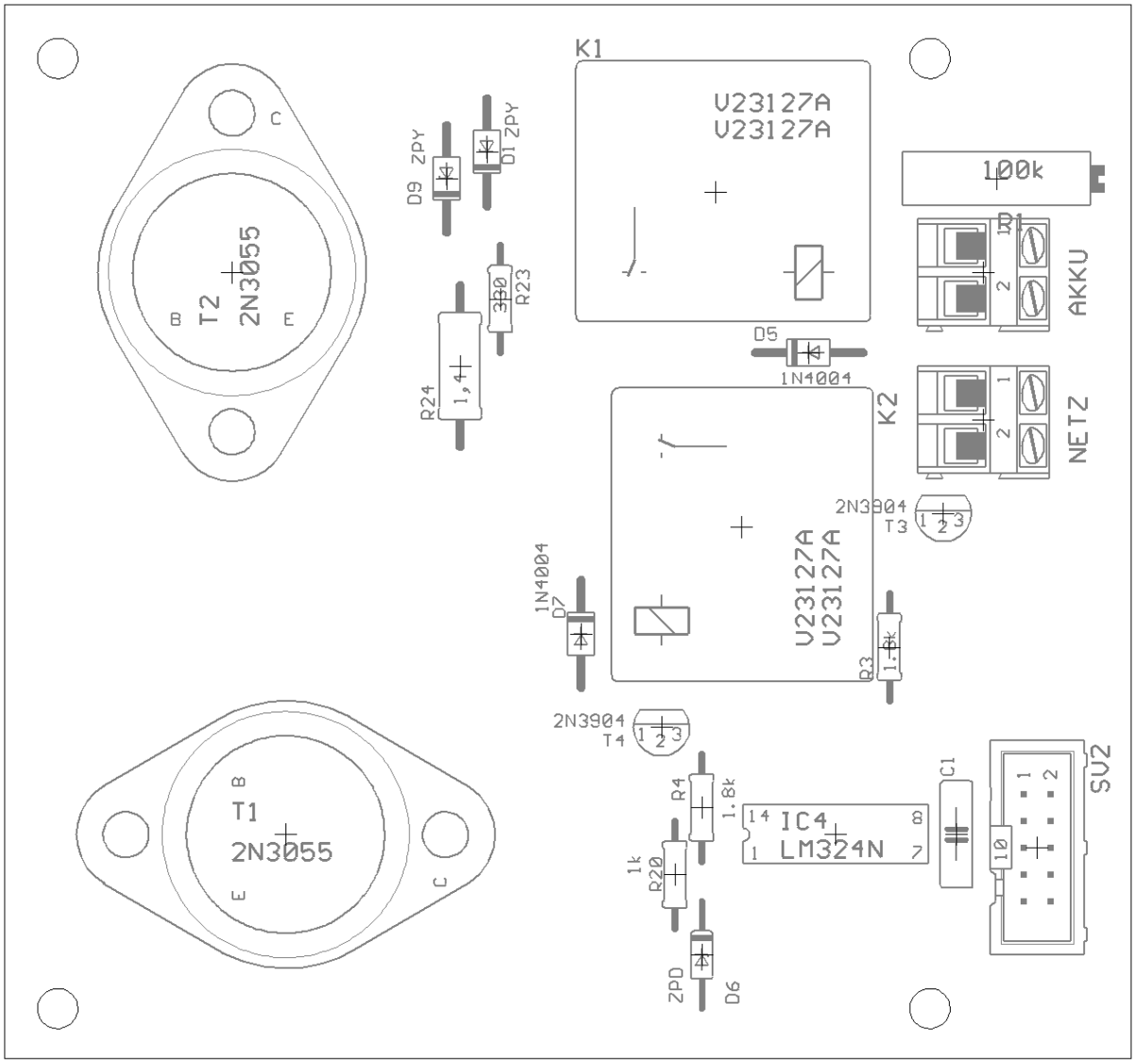


Abbildung 124: zeigt den gesamten Aufbau der Schaltung

Der Treiberschaltung ist zweimal vorhanden, da für die verschiedenen Zustände 2 Relais benötigt werden. Die beiden Relais sind im Normalbetrieb
offen, und trennen den Akku vom Netzteil.
Im Ladebetrieb bleibt K1 offen und K2 schließt. Durch die Strombegrenzung fließt dann ein Ladestrom von maximal 550 mA in den Akku.
Im Notstrombetrieb schließt K1, und der Akku versorgt.

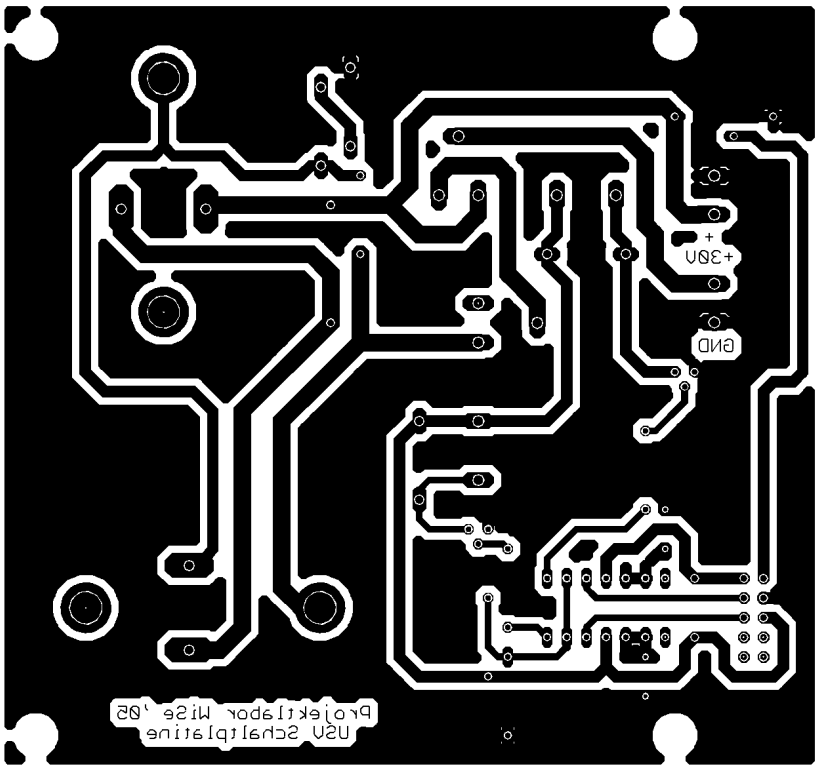
Stückliste



AKKU
1 x ZPY (20V)
2 x 1N4004
1 x ZPD (2,7V)
1 x ZPY (9,1V)
1 x LM324N
2 x V23127A
1 x 100k Ω
2 x 1.8k Ω
1 x 1k Ω
1 x 330 Ω
1 x 1,4 Ω
2 x 2N3055
2 x 2N3904
DIL-Sockel 14Pol.

Ätzlayout:

Bottom-Layer



7. Bedienungsanleitung

Zur Inbetriebnahme sollte man das Gerät auf einen glatten ebenen und stabilen Untergrund stellen, am besten den Boden.
Danach verbinde man, mit dem dafür vorgesehenen Kabel, das Gerät mit einer Steckdose.
Wenn alles korrekt durchgeführt wurde, sollte die Anzeige nun etwas darstellen.

8. Wartung

Das Gerät ist völlig wartungsfrei, bei versagen ist das Gerät als defekt anzusehen.

9. Spezifikationen

Höhe	125cm
Volumen	140 Liter
Gewicht	(schwer ca. 20 kg)
Energieverbrauch	? bei 230V 50Hz
Nominale Arbeitstemperatur	25°C ± 3
Maximale Arbeitstemperatur	-25°C ~ 40°C
Anzahl LEDs	546 + 10 (Akku)
Anzahl 7-Segmente	15

10. Anhang

10.1. Anhänge der Gruppe 1

10.2. Anhänge der Gruppe 2

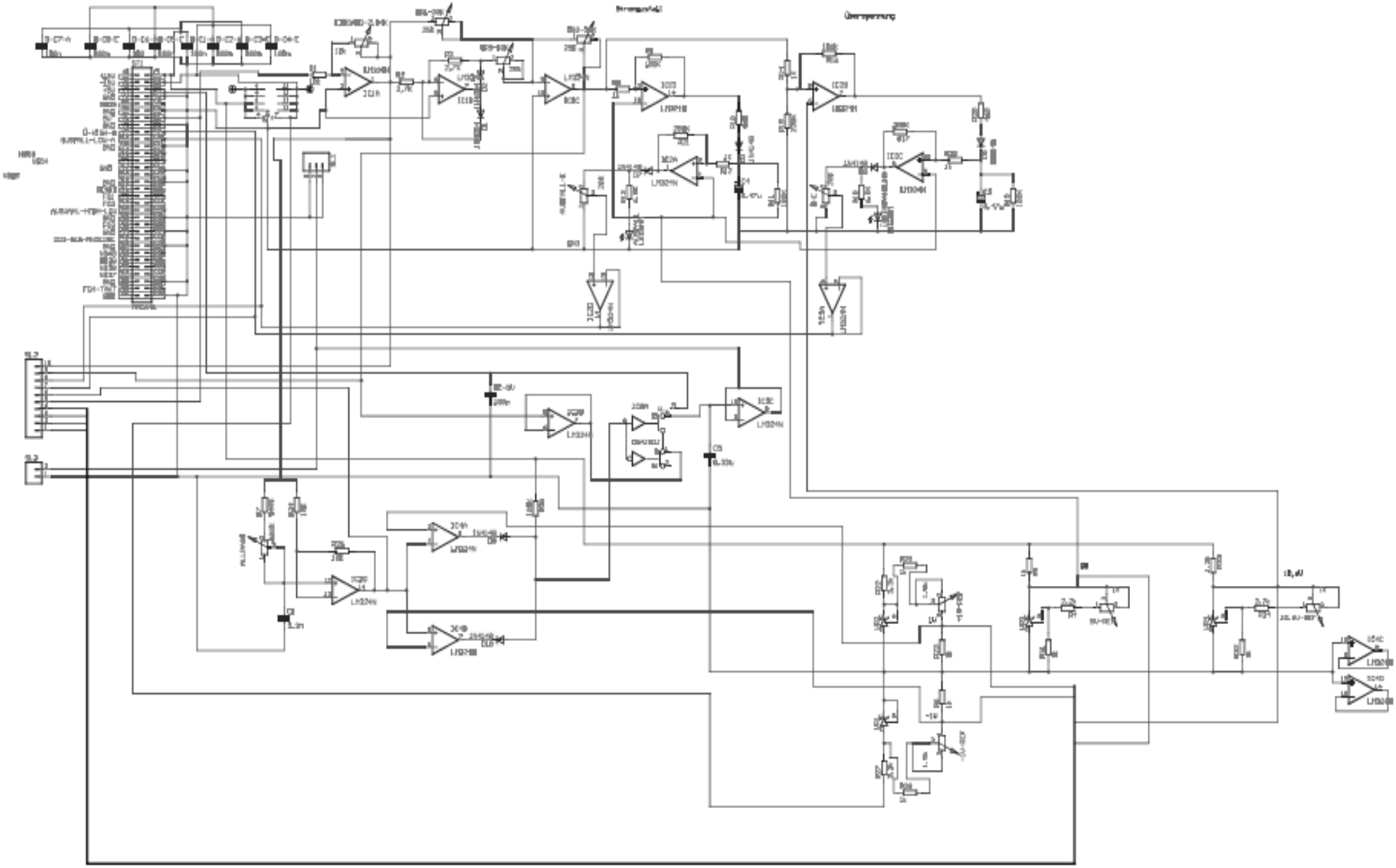


Abbildung 125: Schematic der Analogplatine

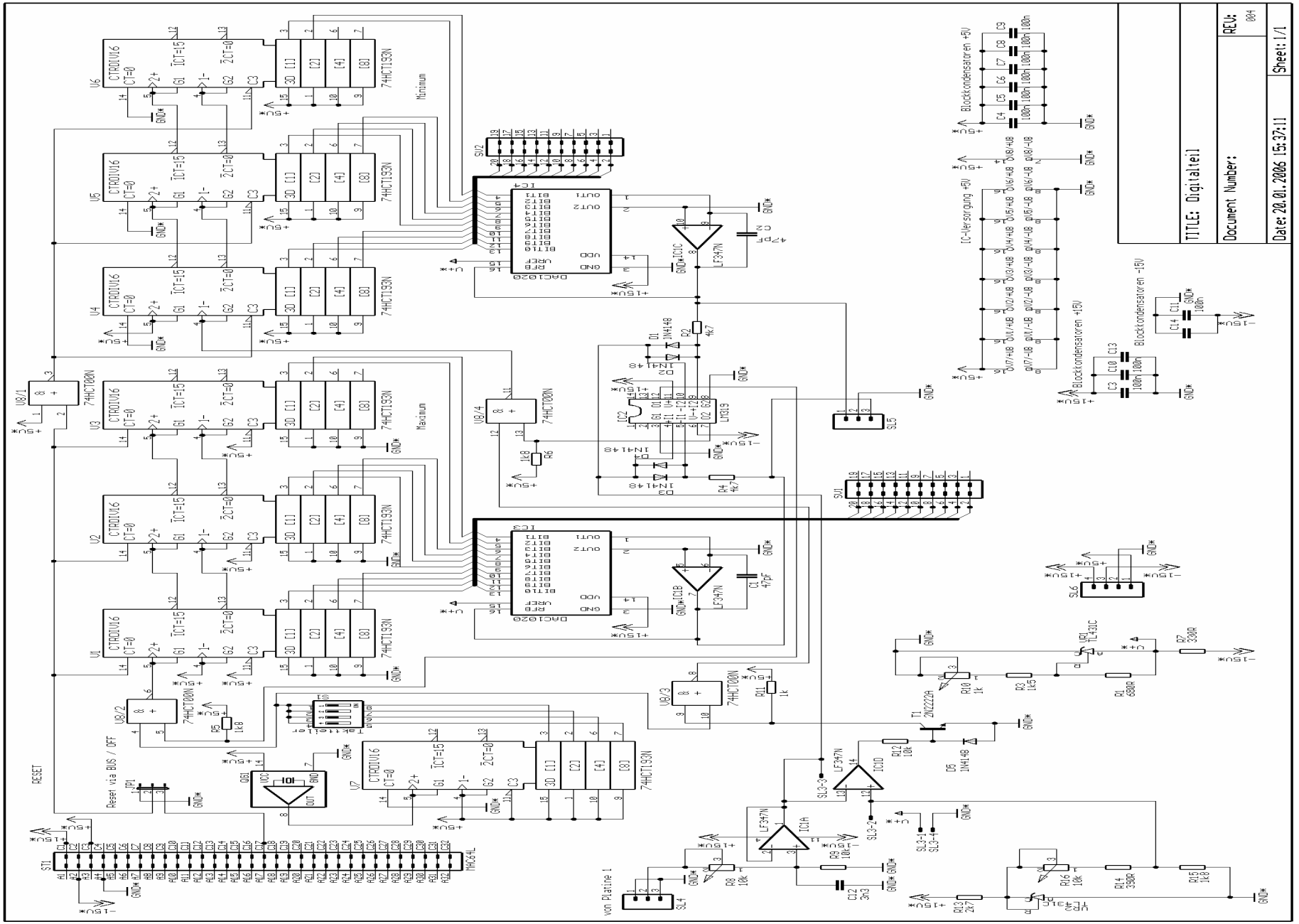


Abbildung 126: Schematic der Digitalteilplatine

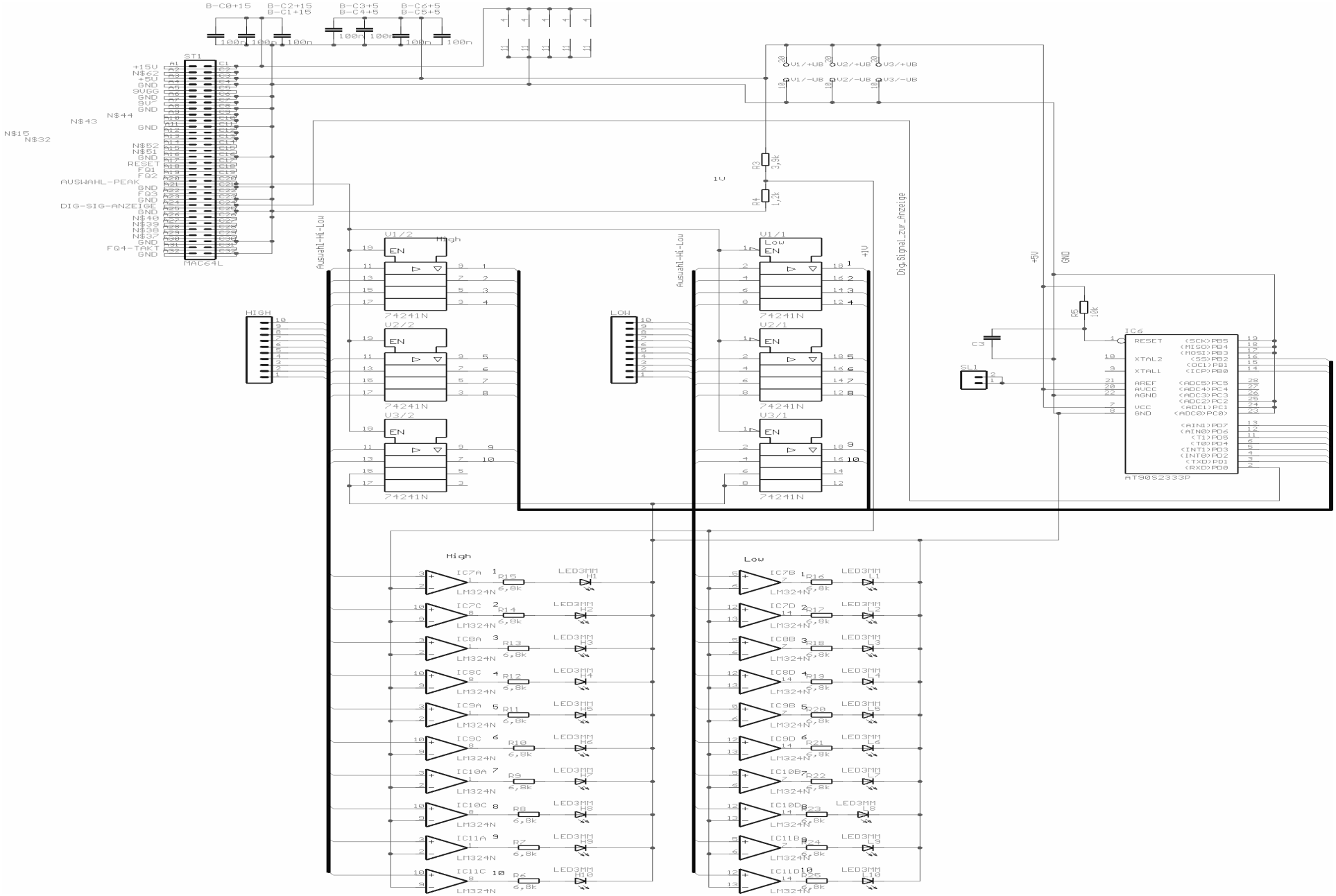


Abbildung 127: Schematic der Umschaltplatine

10.3. Anhänge der Gruppe 3

Anhang1: TestEmpfanger (code)

D:\juergen\Projekt\Projekt\prog\TestEmpf\Global.h

```
1:
/*************************************************************************
****/
2: /*Projektname: TestEmpf
*/
3: /*
*/
4: /*Modul: global.h
*/
5: /*
*/
6: /*Makefile: -
*/
7: /*
*/
8: /*Inhalt: Datenstrukturen und -typen zum Umgang mit Messwerten
*/
9:
/*************************************************************************
****/
10: /*Autor: Jürgen Funck Initialien: JHF
*/
11:
/*************************************************************************
****/
12:
13: #ifndef global
14:
15: #define global
16:
17: #define OK 0x01
18: #define ERROR 0x02
19:
20: /***eigene Datentypen***/
21: typedef unsigned char UINT8;
22: typedef signed char INT8;
23: typedef unsigned short int UINT16;
24: typedef signed short int INT16;
25: typedef unsigned int UINT32;
26: typedef signed int INT32;
27:
28: enum boolean_t {false, true};
29:
30:
31: /***Strukturen***/
32: struct IntVektor {
33: INT16 x;
34: INT16 y;
35: };
36:
37: struct LongVektor {
38: INT32 x;
39: INT32 y;
40: };
41:
42: struct UINT32_vect {
```

```
43: UINT32 word1;
44: UINT32 word2;
45: };
46:
47: struct EMPFANG1 {
48: UINT8 package_counter;
49: UINT8 error_rate;
50: INT16 ax;
51: INT16 ay;
52: INT16 dvx;
53: INT16 dvy;
54: INT16 dsx;
55: INT16 dsy;
56: INT16 dt;
57: };
58:
59: struct DATSATZ {
60: UINT8 status;
D:\juergen\Projekt\Projekt\prog\TestEmpf\Global.h
61: struct IntVektor a;
62: struct LongVektor v;
63: struct LongVektor s;
64: UINT32 t;
65: UINT32 package_counter;
66: UINT8 error_rate;
67: };
68:
69: struct command {
70: char *befehl;
71: char *param;
72: };
73:
74: struct mw{ UINT16 T1x;
75: UINT16 T1y;
76: UINT16 T2;
77: };
78:
79: struct ds{ UINT16 Counter;
80: INT16 Ax;
81: INT16 Ay; };
82:
83: struct res{ UINT16 Counter;
84: INT16 Ax;
85: INT16 Ay;
86:
87: enum boolean_t error; };
88:
89: struct pass{ struct ds dat;
90: UINT8 status; };
91:
92: struct freq
93: {
94: double F;
95: double T2;
96: double T1x;
97: double T1y;
98:
99: double Ax;
100: double Ay;
101:
102: enum boolean_t error;
103: };
104:
105: #define EMPFANGEN struct pass //Empfagnstyp festlegen
106: static const EMPFANGEN RESET_EMPFANGEN = {{0,0,0},ERROR};
107:
108: #define DATENSATZ struct res //Datensatztype festlegen
109: static const DATENSATZ RESET_DATENSATZ = {0,0,0,OK};
110:
111: #endif
D:\juergen\Projekt\Projekt\prog\TestEmpf\TestEmpf.c
```

```
1: #include <stdio.h>
2: #include <windows.h>
3: #include <process.h>
4:
5: #include "Global.h"
6: #include "Com.h"
7:
8: int main(void) {
9: enum boolean_t empf = true;
10: enum boolean_t b = false;
11: UINT32 empfangen = 0x00000000;
12: UINT16 f;
13:
14: printf("Wilkommen zum Testempfaenger der Frequnzmessung\n");
15:
16: HANDLE hCom = OpenCom("COM1",2400,8,NOPARITY, TWOSTOPBITS); //öffne
Com-Port
17:
18: while(empf==true) {
19: b = ReadFromCom(hCom,&empfangen,4);
20:
21: if(b==true){
22: f=0x0000;
23:
24: f |= (empfangen & 0x0F000000)>>24;
25: f |= (empfangen & 0x000F0000)>>12;
26: f |= (empfangen & 0x00000F00);
27: f |= (empfangen & 0x0000000F)<<12;
28:
29: //printf("%#x \n",empfangen);
30: //printf("%#x \n",f);
31: printf("%i mHz\n",f);
32: }
33: }
34: return 0;
35: }
```

D:\juergen\Projekt\Projekt\prog\TestEmpf\Com.h

```
1:
/*************************************************************************
****/
2: /*Projektname: TestEmpf
*/
3: /*
*/
4: /*Modul: Com.h
*/
5: /*
*/
6: /*Makefile: -
*/
7: /*
*/
8: /*Inhalt: Funktionen zur Ansteuerung eines COM-Ports
*/
9:
/*************************************************************************
****/
10: /*Autor: Jürgen Funck Initialien: JHF
*/
11:
/*************************************************************************
****/
12:
13: #include <windows.h>
14:
15: extern HANDLE OpenCom(LPCTSTR com, long baudrate,BYTE datenbits,BYTE
paritaet,BYTE stopbit);
16: //Öffnet und initialisiert einen COM-Port
17:
18: extern void CloseCom(HANDLE hCom);
19: //Schließt einen COM-Port
```

```
20:
21: extern enum boolean_t WriteToCom(HANDLE hCom, LPVOID pDaten, DWORD
AnzBytes);
22: //Schreibt AnzBytes in einen COM-Port
23:
24: extern enum boolean_t ReadFromCom(HANDLE hCom, LPVOID pSpeicher, DWORD
AnzBytes);
25: //Ließt AnzBytes aus einem COM-Por
```

```
D:\juergen\Projekt\Projekt\prog\TestEmpf\Com.c
1:
/*************************************************************************
****/
2: /*Projektname: TestEmpf
*/
3: /*
*/
4: /*Modul: Com.c
*/
5: /*
*/
6: /*Makefile: -
*/
7: /*
*/
8: /*Inhalt: Funktionen zur Ansteuerung eines COM-PORTs
*/
9:
/*************************************************************************
****/
10: /*Autor: Jürgen Funck Initialien: JHF
*/
11:
/*************************************************************************
****/
12:
13: #include <windows.h>
14: #include <stdio.h>
15:
16: #include "global.h"
17:
18: static COMMTIMEOUTS oldtimeouts;
19:
20:
21:
//*************************************************************GLOBAL****
*********************************************************
22:
23:
/*************************************************************************
****/
24: /*Funktinsname: OpenCom()
*/
25: /*Dateiname: Com.c
*/
26: /*Übergabe: Portname als string (LPCTSTR)
*/
27: /* Baudrate als int
*/
28: /* Anzahl der Datenbits pro Byte als BYTE
*/
29: /* Parität als BYTE
*/
30: /* Stopbits als BYTE
*/
```

```
31: /*Rückgabe: Handle auf den geöffneten Port als HANDLE
*/
32: /*
*/
33: /*Beschreibung: Öffnet und initialisiert einen COM-Port
*/
34:
/*************************************************************************
****/
35: /*Autor: Jürgen Funck Initialien: JHF
*/
36:
/*************************************************************************
****/
37:
38: HANDLE OpenCom(LPCTSTR com, long baudrate,BYTE datenbits,BYTE
paritaet,BYTE stopbit)
39: {
40: DCB dcb;
41: HANDLE hCom;
42: DWORD dwError;
43: BOOL fSuccess;
44: COMMTIMEOUTS newtimeouts;
D:\juergen\Projekt\Projekt\prog\TestEmpf\Com.c
45:
46: //öffne COM-Port
47:
48: hCom = CreateFile( com, //öffne COM-PORT
49: GENERIC_READ | GENERIC_WRITE,
50: 0, //Muss bei COM-Ports 0 sein
51: NULL, //keine Sicherheits Attribute
52: OPEN_EXISTING, //COM-Ports müssen OPEN_EXISTING
benutzen
53: 0, //keine Überlappende I/O
54: NULL //hTemplate muss bei COM-Ports NULL
sein
55: );
56:
57: //überprüfe ob erfolgreich
58:
59: if(hCom == INVALID_HANDLE_VALUE)
60: {
61: dwError = GetLastError();
62:
63: //behandle Fehler
64: printf("Mist! nicht getroffen!\n");
65: }
66:
67: fSuccess = GetCommState(hCom, &dcb);
68:
69: if(!fSuccess)
70: {
71: //behandle Fehler
72: printf("Mist! kein Status\n!");
73: }
74:
75: //initialise COM
76:
77: dcb.BaudRate = baudrate;
78: dcb.ByteSize = datenbits;
79: dcb.Parity = paritaet;
80: dcb.StopBits = stopbit;
81: fSuccess = SetCommState(hCom, &dcb);
82:
83: //assure success
84: if(!fSuccess)
85: {
86: //Handle the error
87: printf("Mist! Initialisiereung fehlgeschlagen\n!");
88: }
89: /*
```

```
90: GetCommTimeouts(hCom,&oldtimeouts); //sichere CommTimeouts
91:
92: newtimeouts.ReadIntervalTimeout = 12;
93: newtimeouts.ReadTotalTimeoutMultiplier = 6;
94: newtimeouts.ReadTotalTimeoutConstant = 4;
95: newtimeouts.WriteTotalTimeoutMultiplier = 6;
96: newtimeouts.WriteTotalTimeoutConstant = 4;
97:
98: SetCommTimeouts(hCom,&newtimeouts); //setze CommTimeouts
99: */
100: return hCom;
101: }
102:
103:
/*********************************************************************
****/
104: /*Funktinsname: CloseCom()
*/
105: /*Dateiname: Com.c
*/
106: /*Übergabe: HANDLE auf den Port als HANDLE
*/
107: /*Rückgabe: void
*/
108: /*
D:\juergen\Projekt\Projekt\prog\TestEmpf\Com.c
*/
109: /*Beschreibung: Schließt einen COM-Port
*/
110:
/*********************************************************************
****/
111: /*Autor: Jürgen Funck Initialien: JHF
*/
112:
/*********************************************************************
****/
113:
114: void CloseCom(HANDLE hCom)
115: {
116: SetCommTimeouts(hCom,&oldtimeouts); //wiederherstellen der
CommTimeouts
117: CloseHandle(hCom); //schließe COM-Port
118: }
119:
120:
/*********************************************************************
****/
121: /*Funktinsname: WriteToCom()
*/
122: /*Dateiname: Com.c
*/
123: /*Übergabe: HANDLE auf den Port als HANDLE
*/
124: /* Pointer auf die zu schreibenden Daten als LPCVOID
*/
125: /* Anzahl der zu schreibenden Bytes als DWORD
*/
126: /*Rückgabe: Erfolgsmeldung als BOOL
*/
127: /*
*/
128: /*Beschreibung: Schreibt AnzBytes in einen COM-Port
*/
129:
/*********************************************************************
****/
130: /*Autor: Jürgen Funck Initialien: JHF
*/
131:
/*********************************************************************
```

```
****/
132:
133: enum boolean_t WriteToCom(HANDLE hCom, LPVOID pDaten, DWORD AnzBytes)
134: {
135: DWORD geschrieben = 0;
136:
137: WriteFile(hCom, pDaten, AnzBytes,&geschrieben, 0); //schreibe Daten
138:
139: if(AnzBytes != geschrieben)
140: return false;
141: else
142: return true;
143: }
144:
145:
/*********************************************************************
****/
146: /*Funktinsname: ReadFromCom()
*/
147: /*Dateiname: Com.c
*/
148: /*Übergabe: HANDLE auf den Port als HANDLE
*/
149: /* Pointer auf Speicher, in den die Daten geschrieben werden
*/
150: /* sollen als LPCVOID
*/
151: /* Anzahl der zu schreibenden Bytes als DWORD
*/
152: /*Rückgabe: Erfolgsmeldung als BOOL
*/
153: /*
D:\juergen\Projekt\Projekt\prog\TestEmpf\Com.c
*/
154: /*Beschreibung: Ließt AnzBytes aus einem COM-Port
*/
155:
/*********************************************************************
****/
156: /*Autor: Jürgen Funck Initialien: JHF
*/
157:
/*********************************************************************
****/
158:
159: enum boolean_t ReadFromCom(HANDLE hCom, LPVOID pSpeicher, DWORD
AnzBytes)
160: {
161: DWORD gelesen = 0;
162: char str[20];
163:
164: ReadFile(hCom, pSpeicher, AnzBytes, &gelesen, 0); //lese Daten
165:
166: if(AnzBytes != gelesen)
167: return false;
168: else
169: return true;
170: }
```

Anhang2: Frequenzaeler

D:\juergen\Projekt\Projekt\prog\Frequenzzaehler\settings.h

```
1: /*****************************************************
2: * KONSTANTEN *
3: *****************************************************/
4: // Taktfrequenz des Mikrocontrollers
5: #define FCK 14745600 // 14,7456MHz
6: //#define FCK 10000000 // 10MHz
7: #define PERIODEN 50 // Anzahl der Perioden pro Messung
8: #define ZEITBASIS FCK/8 // Zeitbasis für die
9: #define NENNER PERIODEN*ZEITBASIS // Nenner für die Frequenzberechnung
10: #define START_PER 1 // Perioden, die gemessen werden bevor etwas
angezeigt wird
11:
12: #define UART
13: //#define LCD
```

D:\juergen\Projekt\Projekt\prog\Frequenzzaehler\Frequenzzaehler.c

```
1: /*****************************************************
2: * HEADER-DATEIEN *
3: *****************************************************/
4:
5: #include <inttypes.h>
6: #include <stdlib.h>
7: #include <avr/io.h>
8: #include <avr/interrupt.h>
9: #include <avr/signal.h>
10: #include <avr/pgmspace.h>
11:
12: #include "settings.h"
13:
14: #ifdef LCD
15: #include "lcd.h"
16: #endif
17:
18: #ifdef UART
19: #include "uart.h"
20: #endif
21:
22: /*****************************************************
23: * FUNKTIONS PROTOTYPEN *
24: *****************************************************/
25: void init_timer(); // initialisiert den Timer
26: void init_ports(); // initialisiert die Ports des Mikrocontrollers
27: void init_lcd(); // initialisiert das LCD-Display
28: void put_to_lcd(uint16_t mw); // gibt einen Messwert auf dem
29: // LCD-Display aus
30: uint16_t berechne(); // berechnet die Frequenz
31: void check_and_send(uint16_t f); // überprüft was angefordert wird und
sendet es
32:
33:
34: /*****************************************************
35: * GLAOBALE-VARIABLEN *
36: *****************************************************/
37: volatile uint32_t overflows; // Überlaufzähler
38: volatile uint32_t ov; // Zwischenspeicher zur Ausgabe des
Überlaufzählers
39: volatile uint32_t dt; // Periodendauer
40: volatile uint8_t wait; // wait-Flag, zeigt an ob messung läuft
41:
42: volatile uint16_t f_max; // maximale gemesseneFrequenz
43: volatile uint16_t f_min; // minimlae gemessene Frequenz
44:
45: /*****************************************************
46: * INTERRUPT-ROUTINEN *
47: *****************************************************/
48:
```

```
49: /*****************************************************
50: * Intterupt: input capture *
51: * Beschreibung: liest den Timerstand aus *
52: * Parameter: void *
53: * Rückgabe: void *
54: *****************************************************/
55: SIGNAL (SIG_TIMER1_CAPT) {
56: // sicherer Timerstand und resete Timer
57: uint16_t TimerStand = ICR1;
58: TCNT1 = 0x00;
59:
60: // Berechne Periodendauer
61: ov = overflows;
62: dt = ((((uint32_t)ov)<<16) + TimerStand);
63:
64: // resete Überlaufzähler
65: overflows = 0;
66: wait = 0;
67: }
68:
69: /*****************************************************
D:\juergen\Projekt\Projekt\prog\Frequenzzaehler\Frequenzzaehler.c
70: * Intterupt: ´Timer1 Overflow *
71: * Beschreibung: inkrementiert den Überlaufzähler *
72: * Parameter: void *
73: * Rückgabe: void *
74: *****************************************************/
75:
76: SIGNAL (SIG_TIMER1_OVF) {
77: overflows++;
78: }
79:
80: /*****************************************************
81: * HAUPTPROGRAMM *
82: *****************************************************/
83: int main(void) {
84: // zählt die gemessenen Perioden ab dem start
85: uint8_t i=0;
86:
87: // zuletzt gemessene Frequnez
88: uint16_t f = 0xffff;
89: f_max = 0xffff;
90: f_min = 0xffff;
91:
92: // deaktiviere globalen interrupt
93: cli();
94:
95: // initialisiere
96: #ifdef UART
97: init_uart();
98: #endif
99: #ifdef LCD
100: init_lcd();
101: #endif
102:
103: init_ports();
104: init_timer();
105:
106: // sende testbyte
107: sende(0x1234);
108:
109: // aktiviere globalen interrupt
110: sei();
111:
112: for(;;){
113: // warte auf neuen Messwert
114: wait = 1;
115: // aktueller Zustand der Signalleitungen
116: uint8_t lastStat = PIND & 0x30;
117:
118: while(wait) {
```

```
119: // Poll Signalleitungen
120:
121: if(bit_is_set(PIND,PD3)) {
122: // reset
123: //i=0;
124: }
125: if(lastStat != (PIND & 0x30)){
126: // Auswahlleitungen haben sich geändert
127: lastStat = PIND & 0x30;
128: check_and_send(f);
129: }
130: }
131:
132: // Messung beendet !!!
133: if(i > START_PER) {
134: // berechne Frequnez
135: f=berechne();
136:
137: /*
138: if(bit_is_set(PORTB,PB7))
139: PORTB &= ~(1<<PB7);
140: else
D:\juergen\Projekt\Projekt\prog\Frequenzzaehler\Frequenzzaehler.c
141: PORTB |= (1<<PB7);
142: */
143:
144: // setzte minimal und Maximalwerte
145: if(f>f_max)
146: f_max = f;
147: if(f<f_min)
148: f_min = f;
149:
150:
151: #ifdef UART
152: // sende Messwert
153: check_and_send(f);
154: #endif
155:
156: #ifdef LCD
157: // gebe Messwert auf LCD aus
158: put_to_lcd(f);
159: #endif
160: }
161: else if(i == START_PER) {
162: // berechne Frequnez
163: f=berechne();
164: f_min = f;
165: f_max = f;
166: i++;
167: }
168: else {
169: i++;
170: }
171:
172: } // Endlosschleife
173:
174: return 0;
175: }
176:
177: /*************************************************
178: * FUNKTIONEN *
179: *************************************************/
180:
181: /*************************************************
182: * Funktion: init_timer *
183: * Beschreibung: initialisiert den 16Bit-Timer *
184: * Parameter: void *
185: * Rückgabe: void *
186: *************************************************/
187: void init_timer(void) {
188: // globale variablen initialisieren
```

```
189: overflows = 0;
190: dt = 0;
191:
192: // lösche Timerstand
193: TCNT1H = 0x00;
194: TCNT1L = 0x00;
195:
196: // NOISE-Canceller deaktiviert
197: // Capture an abfallender Flanke
198: // takte Timer mit Systemtakt/8
199: // starte Timer
200: TCCR1B = 0x00 | 0x02;
201:
202: // aktiviere Input/Capture Interrupt
203: TIMSK = 0x00 | (1<<TOIE1) | (1<<ICIE1);
204:
205: }
206:
207: /*************************************************
208: * Funktion: init_ports *
209: * Beschreibung: initialisiert die Ports *
210: * Parameter: void *
211: * Rückgabe: void *
D:\juergen\Projekt\Projekt\prog\Frequenzzaehler\Frequenzzaehler.c
212: *************************************************/
213: void init_ports(void) {
214: DDRD = 0xff & ~((1<<PD3)|(1<<PD4)|(1<<PD5)|(1<<PD6));
215: PORTD = ((1<<PD3)|(1<<PD4)|(1<<PD5)|(1<<PD6));
216:
217: DDRB = 0xff & ~((1<<PB0)|(1<<PB1));
218: PORTB = ((1<<PB0)|(1<<PB1));
219: }
220:
221: /*************************************************
222: * Funktion: berechne *
223: * Beschreibung: berechnet die Frequenz *
224: * Parameter: void *
225: * Rückgabe: Frequenz *
226: *************************************************/
227: uint16_t berechne() {
228: // Speichere Nenner in Variable, dass spart aus irgendeinem
229: // Grund sehr viel Prorgammcode
230: uint32_t nenner = NENNER;
231:
232: // Berechne Quotienten
233: uint32_t f_q = nenner/dt;
234:
235: // berechne Divisionsrest
236: uint32_t f_r = nenner%dt;
237:
238: // multipliziere Ergebnis mit 1000 unter Berücksichtigung
239: // des Rest. Das ist nun die Frequnez
240: uint32_t f = 1000*f_q+(1000*f_r)/dt;
241:
242: // reduziere auf 16Bit (alle höherwertigen Bits sollten jetzt
243: // sowie so null sein) und gebe zurück
244: return (uint16_t)f;
245:
246: //return 0x1234;
247: }
248:
249: /*************************************************
250: * Funktion: check_and_send *
251: * Beschreibung: überprüft was angefordert ist und *
252: * sendet *
253: * Parameter: f - Ergebnis der letzten Messung *
254: * Rückgabe: void *
255: *************************************************/
256:
257: void check_and_send(uint16_t f){
258: uint16_t temp = f; // aktueller Wert ist Defaultwert
```

```
259:
260: if(bit_is_set(PINB,PB1) || !bit_is_set(PINB,PB0))
261: ;//temp = 0xffff; // Netzspannung zu niedrig oder zu hoch
262:
263: if(!bit_is_set(PIND,PD4) && bit_is_set(PIND,PD5))
264: temp = f_min; // Minimalwert gefordert
265: else if(bit_is_set(PIND,PD4) && !bit_is_set(PIND,PD5))
266: temp = f_max; // Maximalwert gefordert
267:
268: // sende Wert
269: sende(temp);
270: }
271:
272: /*****************************************************
273: * Funktion: init_lcd *
274: * Beschreibung: initialisiert das LCD display *
275: * Parameter: void *
276: * Rückgabe: void *
277: *****************************************************/
278: #ifdef LCD
279:
280: void init_lcd(void) {
281: // initialisiere LCD
282: // Display initialisieren, Cursor OFF
D:\juergen\Projekt\Projekt\prog\Frequenzzaehler\Frequenzzaehler.c
283: lcd_init(LCD_DISP_ON);
284:
285: lcd_clrscr(); // LCD-Inhalt löschen, Cursor an den Anfang setzen
286: }
287:
288: /*****************************************************
289: * Funktion: put_to_lcd *
290: * Beschreibung: gibt einen Messwert auf dem LCD *
291: * Display aus *
292: * Parameter: mw - der auszugebene Messwert *
293: * Rückgabe: void *
294: *****************************************************/
295:
296: void put_to_lcd(uint16_t mw) {
297: char str[10];
298: char str2[10];
299: char* pstr = str;
300: pstr = utoa(mw,pstr,10);
301: pstr = str2;
302: pstr = utoa(ov,pstr,10);
303: lcd_home();
304: lcd_puts(str);
305: lcd_gotoxy(0,1);
306: lcd_puts(str2);
307: }
308:
309: #endif
D:\juergen\Projekt\Projekt\prog\Frequenzzaehler\uart.h
1: #include "settings.h"
2:
3: // UBRR = (f_ocs/(16*Baud))-1
4: #define BAUD 2400
5: #define BAUDVALUE (((FCK/16)/BAUD)-1)
6:
7: /*****************************************************
8: * Funktion: sende *
9: * Beschreibung: sendet eine 16-Bit Wert über die *
10: * serielle Schnittstelle *
11: * Parameter: daten *
12: * Rückgabe: void *
13: *****************************************************/
14: void sende(uint16_t data);
15:
16: /*****************************************************
17: * Funktion: init_uart *
18: * Beschreibung: initialisiert den UART *
```

```
19: * Parameter: void *
20: * Rückgabe: void *
21: ************************************************/
22: void init_uart(void);
D:\juergen\Projekt\Projekt\prog\Frequenzzaehler\uart.c
1: #include <inttypes.h>
2: #include <avr/io.h>
3:
4: #include "uart.h"
5:
6: /************************************************
7: * Funktion: init_uart *
8: * Beschreibung: initialisiert den UART *
9: * Parameter: void *
10: * Rückgabe: void *
11: ************************************************/
12: void init_uart(void) {
13: // setzte Baudrate
14: UBRRH = (unsigned char)(BAUDVALUE>>8);
15: UBRRL = (unsigned char)BAUDVALUE;
16:
17: // aktiviere Transmitter
18: UCSRB = (1<<TXEN);
19: // Frame Format: 8data, 2stop bits
20: UCSRC = (1<<USBS)|(3<<UCSZ0);
21: }
22:
23: /************************************************
24: * Funktion: sende *
25: * Beschreibung: sendet eine 16-Bit Wert über die *
26: * serielle Schnittstelle *
27: * Parameter: data - die zu sendenden Daten *
28: * Rückgabe: void *
29: ************************************************/
30: void sende(uint16_t data) {
31: uint8_t counter;
32: uint8_t temp;
33:
34: // sende Daten in 4 Halbbytes
35: for(counter=1; counter<=4; counter++) {
36: // warte bis uart frei
37: while ( !( UCSRA & (1<<UDRE)) );
38: // Sende erstes Halbbyte
39: temp = data>>(4*(4-counter));
40: temp &= 0x0F;
41: temp |= (counter<<4);
42: UDR = temp;
43: }
44: }
45:
D:\juergen\Projekt\Projekt\prog\Frequenzzaehler\lcd.h
1: #ifndef LCD_H
2: #define LCD_H
3:
/************************************************************************
*
4: Title : C include file for the HD44780U LCD library (lcd.c)
5: Author: Peter Fleury <pfleury@gmx.ch> http://jump.to/fleury
6: File: $Id: lcd.h,v 1.12.2.4 2005/02/28 22:54:41 Peter Exp $
7: Software: AVR-GCC 3.3
8: Hardware: any AVR device, memory mapped mode only for
AT90S4414/8515/Mega
9:
************************************************************************
**/
10:
11: /**
12: @defgroup pfleury_lcd LCD library
13: @code #include <lcd.h> @endcode
14:
15: @brief Basic routines for interfacing a HD44780U-based text LCD display
```

```
16:
17: Originally based on Volker Oth's LCD library,
18: changed lcd_init(), added additional constants for lcd_command(),
19: added 4-bit I/O mode, improved and optimized code.
20:
21: Library can be operated in memory mapped mode (LCD_IO_MODE=0) or in
22: 4-bit IO port mode (LCD_IO_MODE=1). 8-bit IO port mode not supported.
23:
24: Memory mapped mode compatible with Kanda STK200, but supports also
25: generation of R/W signal through A8 address line.
26:
27: @author Peter Fleury pfleury@gmx.ch http://jump.to/fleury
28:
29: @see The chapter <a
href="http://homepage.sunrise.ch/mysunrise/peterfleury/avr-lcd44780.html"
target="_blank">Interfacing a HD44780 Based LCD to an AVR</a>
30: on my home page.
31:
32: */
33:
34: /*@{*/
35:
36: #if (__GNUC__ * 100 + __GNUC_MINOR__) < 303
37: #error "This library requires AVR-GCC 3.3 or later, update to newer
AVR-GCC compiler !"
38: #endif
39:
40: #include <inttypes.h>
41: #include <avr/pgmspace.h>
42:
43: /**
44: * @name Definitions for MCU Clock Frequency
45: * Adapt the MCU clock frequency in Hz to your target.
46: */
47: #define XTAL 4000000 /**< clock frequency in Hz, used to
calculate delay timer */
48:
49: /**
50: * @name Definitions for Display Size
51: * Change these definitions to adapt setting to your display
52: */
53: #define LCD_LINES 2 /**< number of visible lines of the
display */
54: #define LCD_DISP_LENGTH 16 /**< visibles characters per line of
the display */
55: #define LCD_LINE_LENGTH 0x40 /**< internal line length of the
display */
56: #define LCD_START_LINE1 0x00 /**< DDRAM address of first char of
line 1 */
57: #define LCD_START_LINE2 0x40 /**< DDRAM address of first char of
line 2 */
58: #define LCD_START_LINE3 0x14 /**< DDRAM address of first char of
line 3 */
D:\juergen\Projekt\Projekt\prog\Frequenzzaehler\lcd.h
59: #define LCD_START_LINE4 0x54 /**< DDRAM address of first char of
line 4 */
60: #define LCD_WRAP_LINES 0 /**< 0: no wrap, 1: wrap at end of
visibile line */
61:
62:
63: #define LCD_IO_MODE 1 /**< 0: memory mapped mode, 1: IO port
mode */
64: #if LCD_IO_MODE
65: /**
66: * @name Definitions for 4-bit IO mode
67: * Change LCD_PORT if you want to use a different port for the LCD pins.
68: *
69: * The four LCD data lines and the three control lines RS, RW, E can be
on the
70: * same port or on different ports.
71: * Change LCD_RS_PORT, LCD_RW_PORT, LCD_E_PORT if you want the control
```

```
lines on
72: * different ports.
73: *
74: * Normally the four data lines should be mapped to bit 0..3 on one
port, but it
75: * is possible to connect these data lines in different order or even on
different
76: * ports by adapting the LCD_DATAx_PORT and LCD_DATAx_PIN definitions.
77: *
78: */
79: #define LCD_PORT PORTB /**< port for the LCD lines */
80: #define LCD_DATA0_PORT LCD_PORT /**< port for 4bit data bit 0 */
81: #define LCD_DATA1_PORT LCD_PORT /**< port for 4bit data bit 1 */
82: #define LCD_DATA2_PORT LCD_PORT /**< port for 4bit data bit 2 */
83: #define LCD_DATA3_PORT LCD_PORT /**< port for 4bit data bit 3 */
84: #define LCD_DATA0_PIN 0 /**< pin for 4bit data bit 0 */
85: #define LCD_DATA1_PIN 1 /**< pin for 4bit data bit 1 */
86: #define LCD_DATA2_PIN 2 /**< pin for 4bit data bit 2 */
87: #define LCD_DATA3_PIN 3 /**< pin for 4bit data bit 3 */
88: #define LCD_RS_PORT LCD_PORT /**< port for RS line */
89: #define LCD_RS_PIN 4 /**< pin for RS line */
90: #define LCD_RW_PORT LCD_PORT /**< port for RW line */
91: #define LCD_RW_PIN 5 /**< pin for RW line */
92: #define LCD_E_PORT LCD_PORT /**< port for Enable line */
93: #define LCD_E_PIN 6 /**< pin for Enable line */
94:
95: #elif defined(__AVR_AT90S4414__) || defined(__AVR_AT90S8515__) ||
defined(__AVR_ATmega64__) || \
96: defined(__AVR_ATmega8515__)|| defined(__AVR_ATmega103__) ||
defined(__AVR_ATmega128__) || \
97: defined(__AVR_ATmega161__) || defined(__AVR_ATmega162__)
98: /*
99: * memory mapped mode is only supported when the device has an external
data memory interface
100: */
101: #define LCD_IO_DATA 0xC000 /* A15=E=1, A14=RS=1
*/
102: #define LCD_IO_FUNCTION 0x8000 /* A15=E=1, A14=RS=0
*/
103: #define LCD_IO_READ 0x0100 /* A8 =R/W=1 (R/W: 1=Read, 0=Write
*/
104: #else
105: #error "external data memory interface not available for this device,
use
4-bit IO port mode"
106:
107: #endif
108:
109:
110: /**
111: * @name Definitions for LCD command instructions
112: * The constants define the various LCD controller instructions which
can be passed to the
113: * function lcd_command(), see HD44780 data sheet for a complete
description.
D:\juergen\Projekt\Projekt\prog\Frequenzzaehler\lcd.h
114: */
115:
116: /* instruction register bit positions, see HD44780U data sheet */
117: #define LCD_CLR 0 /* DB0: clear display
*/
118: #define LCD_HOME 1 /* DB1: return to home position
*/
119: #define LCD_ENTRY_MODE 2 /* DB2: set entry mode
*/
120: #define LCD_ENTRY_INC 1 /* DB1: 1=increment, 0=decrement
*/
121: #define LCD_ENTRY_SHIFT 0 /* DB2: 1=display shift on
*/
122: #define LCD_ON 3 /* DB3: turn lcd/cursor on
*/
```

```
123: #define LCD_ON_DISPLAY 2 /* DB2: turn display on
*/
124: #define LCD_ON_CURSOR 1 /* DB1: turn cursor on
*/
125: #define LCD_ON_BLINK 0 /* DB0: blinking cursor ?
*/
126: #define LCD_MOVE 4 /* DB4: move cursor/display
*/
127: #define LCD_MOVE_DISP 3 /* DB3: move display (0-> cursor)
? */
128: #define LCD_MOVE_RIGHT 2 /* DB2: move right (0-> left) ?
*/
129: #define LCD_FUNCTION 5 /* DB5: function set
*/
130: #define LCD_FUNCTION_8BIT 4 /* DB4: set 8BIT mode (0->4BIT
mode) */
131: #define LCD_FUNCTION_2LINES 3 /* DB3: two lines (0->one line)
*/
132: #define LCD_FUNCTION_10DOTS 2 /* DB2: 5x10 font (0->5x7 font)
*/
133: #define LCD_CGRAM 6 /* DB6: set CG RAM address
*/
134: #define LCD_DDRAM 7 /* DB7: set DD RAM address
*/
135: #define LCD_BUSY 7 /* DB7: LCD is busy
*/
136:
137: /* set entry mode: display shift on/off, dec/inc cursor move direction
*/
138: #define LCD_ENTRY_DEC 0x04 /* display shift off, dec cursor
move dir */
139: #define LCD_ENTRY_DEC_SHIFT 0x05 /* display shift on, dec cursor
move dir */
140: #define LCD_ENTRY_INC_ 0x06 /* display shift off, inc cursor
move dir */
141: #define LCD_ENTRY_INC_SHIFT 0x07 /* display shift on, inc cursor
move dir */
142:
143: /* display on/off, cursor on/off, blinking char at cursor position */
144: #define LCD_DISP_OFF 0x08 /* display off
*/
145: #define LCD_DISP_ON 0x0C /* display on, cursor off
*/
146: #define LCD_DISP_ON_BLINK 0x0D /* display on, cursor off, blink
char */
147: #define LCD_DISP_ON_CURSOR 0x0E /* display on, cursor on
*/
148: #define LCD_DISP_ON_CURSOR_BLINK 0x0F /* display on, cursor on, blink
char */
149:
150: /* move cursor/shift display */
151: #define LCD_MOVE_CURSOR_LEFT 0x10 /* move cursor left (decrement)
*/
152: #define LCD_MOVE_CURSOR_RIGHT 0x14 /* move cursor right (increment)
*/
153: #define LCD_MOVE_DISP_LEFT 0x18 /* shift display left
*/
D:\juergen\Projekt\Projekt\prog\Frequenzzaehler\lcd.h
154: #define LCD_MOVE_DISP_RIGHT 0x1C /* shift display right
*/
155:
156: /* function set: set interface data length and number of display lines
*/
157: #define LCD_FUNCTION_4BIT_1LINE 0x20 /* 4-bit interface, single line,
5x7 dots */
158: #define LCD_FUNCTION_4BIT_2LINES 0x28 /* 4-bit interface, dual line,
5x7 dots */
159: #define LCD_FUNCTION_8BIT_1LINE 0x30 /* 8-bit interface, single line,
5x7 dots */
160: #define LCD_FUNCTION_8BIT_2LINES 0x38 /* 8-bit interface, dual line,
5x7 dots */
```

```
161:
162:
163: #define LCD_MODE_DEFAULT ((1<<LCD_ENTRY_MODE) | (1<<LCD_ENTRY_INC) )
164:
165:
166:
167: /**
168: * @name Functions
169: */
170:
171:
172: /**
173: @brief Initialize display and select type of cursor
174: @param dispAttr \b LCD_DISP_OFF display off\n
175: \b LCD_DISP_ON display on, cursor off\n
176: \b LCD_DISP_ON_CURSOR display on, cursor on\n
177: \b LCD_DISP_ON_CURSOR_BLINK display on, cursor on
flashing
178: @return none
179: */
180: extern void lcd_init(uint8_t dispAttr);
181:
182:
183: /**
184: @brief Clear display and set cursor to home position
185: @param void
186: @return none
187: */
188: extern void lcd_clrscr(void);
189:
190:
191: /**
192: @brief Set cursor to home position
193: @param void
194: @return none
195: */
196: extern void lcd_home(void);
197:
198:
199: /**
200: @brief Set cursor to specified position
201:
202: @param x horizontal position\n (0: left most position)
203: @param y vertical position\n (0: first line)
204: @return none
205: */
206: extern void lcd_gotoxy(uint8_t x, uint8_t y);
207:
208:
209: /**
210: @brief Display character at current cursor position
211: @param c character to be displayed
212: @return none
213: */
214: extern void lcd_putc(char c);
215:
216:
217: /**
218: @brief Display string without auto linefeed
D:\juergen\Projekt\Projekt\prog\Frequenzzaehler\lcd.h
219: @param s string to be displayed
220: @return none
221: */
222: extern void lcd_puts(const char *s);
223:
224:
225: /**
226: @brief Display string from program memory without auto linefeed
227: @param s string from program memory be be displayed
228: @return none
229: @see lcd_puts_P
```

```
230: */
231: extern void lcd_puts_p(const char *progmem_s);
232:
233:
234: /**
235: @brief Send LCD controller instruction command
236: @param cmd instruction to send to LCD controller, see HD44780 data
sheet
237: @return none
238: */
239: extern void lcd_command(uint8_t cmd);
240:
241:
242: /**
243: @brief Send data byte to LCD controller
244:
245: Similar to lcd_putc(), but without interpreting LF
246: @param data byte to send to LCD controller, see HD44780 data sheet
247: @return none
248: */
249: extern void lcd_data(uint8_t data);
250:
251:
252: /**
253: @brief macros for automatically storing string constant in program
memory
254: */
255: #define lcd_puts_P(__s) lcd_puts_p(PSTR(__s))
256:
257: /*@}*/
258: #endif //LCD_H
D:\juergen\Projekt\Projekt\prog\Frequenzzaehler\lcd.c
1:
/************************************************************************
****
2: Title : HD44780U LCD library
3: Author: Peter Fleury <pfleury@gmx.ch> http://jump.to/fleury
4: File: $Id: lcd.c,v 1.13.2.6 2005/12/21 22:06:40 peter Exp $
5: Software: AVR-GCC 3.3
6: Target: any AVR device, memory mapped mode only for
AT90S4414/8515/Mega
7:
8: DESCRIPTION
9: Basic routines for interfacing a HD44780U-based text lcd display
10:
11: Originally based on Volker Oth's lcd library,
12: changed lcd_init(), added additional constants for lcd_command(),
13: added 4-bit I/O mode, improved and optimized code.
14:
15: Library can be operated in memory mapped mode (LCD_IO_MODE=0) or
in
16: 4-bit IO port mode (LCD_IO_MODE=1). 8-bit IO port mode not
supported.
17:
18: Memory mapped mode compatible with Kanda STK200, but supports also
19: generation of R/W signal through A8 address line.
20:
21: USAGE
22: See the C include lcd.h file for a description of each function
23:
24:
************************************************************************
****/
25: #include <inttypes.h>
26: #include <avr/io.h>
27: #include <avr/pgmspace.h>
28: #include "lcd.h"
29:
30:
31:
32: /*
```

```
33: ** constants/macros
34: */
35: #define DDR(x) (*(&x - 1)) /* address of data direction register of
port x */
36: #if defined(__AVR_ATmega64__) || defined(__AVR_ATmega128__)
37: /* on ATmega64/128 PINF is on port 0x00 and not 0x60 */
38: #define PIN(x) ( &PORTF==&(x) ? _SFR_IO8(0x00) : (*(&x - 2)) )
39: #else
40: #define PIN(x) (*(&x - 2)) /* address of input register of port x
*/
41: #endif
42:
43:
44: #if LCD_IO_MODE
45: #define lcd_e_delay() __asm__ __volatile__( "rjmp 1f\n 1:" );
46: #define lcd_e_high() LCD_E_PORT |= _BV(LCD_E_PIN);
47: #define lcd_e_low() LCD_E_PORT &= ~_BV(LCD_E_PIN);
48: #define lcd_e_toggle() toggle_e()
49: #define lcd_rw_high() LCD_RW_PORT |= _BV(LCD_RW_PIN)
50: #define lcd_rw_low() LCD_RW_PORT &= ~_BV(LCD_RW_PIN)
51: #define lcd_rs_high() LCD_RS_PORT |= _BV(LCD_RS_PIN)
52: #define lcd_rs_low() LCD_RS_PORT &= ~_BV(LCD_RS_PIN)
53: #endif
54:
55: #if LCD_IO_MODE
56: #if LCD_LINES==1
57: #define LCD_FUNCTION_DEFAULT LCD_FUNCTION_4BIT_1LINE
58: #else
59: #define LCD_FUNCTION_DEFAULT LCD_FUNCTION_4BIT_2LINES
60: #endif
61: #else
62: #if LCD_LINES==1
63: #define LCD_FUNCTION_DEFAULT LCD_FUNCTION_8BIT_1LINE
64: #else
D:\juergen\Projekt\Projekt\prog\Frequenzzaehler\lcd.c
65: #define LCD_FUNCTION_DEFAULT LCD_FUNCTION_8BIT_2LINES
66: #endif
67: #endif
68:
69:
70: /*
71: ** function prototypes
72: */
73: #if LCD_IO_MODE
74: static void toggle_e(void);
75: #endif
76:
77: /*
78: ** local functions
79: */
80:
81:
82:
83:
/************************************************************************
*
84: delay loop for small accurate delays: 16-bit counter, 4 cycles/loop
85:
************************************************************************
/
86: static inline void _delayFourCycles(unsigned int __count)
87: {
88: if ( __count == 0 )
89: __asm__ __volatile__( "rjmp 1f\n 1:" ); // 2 cycles
90: else
91: __asm__ __volatile__ (
92: "1: sbiw %0,1" "\n\t"
93: "brne 1b" // 4 cycles/loop
94: : "=w" (__count)
95: : "0" (__count)
96: );
```

```
97: }
98:
99:
100:
/*************************************************************************
*
101: delay for a minimum of <us> microseconds
102: the number of loops is calculated at compile-time from MCU clock
frequency
103:
*************************************************************************
/
104: #define delay(us) _delayFourCycles( ( ( 1*(XTAL/4000) )*us)/1000 )
105:
106:
107: #if LCD_IO_MODE
108: /* toggle Enable Pin to initiate write */
109: static void toggle_e(void)
110: {
111: lcd_e_high();
112: lcd_e_delay();
113: lcd_e_low();
114: }
115: #endif
116:
117:
118:
/*************************************************************************
*
119: Low-level function to write byte to LCD controller
120: Input: data byte to write to LCD
121: rs 1: write data
122: 0: write instruction
123: Returns: none
124:
*************************************************************************
/
125: #if LCD_IO_MODE
126: static void lcd_write(uint8_t data,uint8_t rs)
127: {
128: unsigned char dataBits ;
D:\juergen\Projekt\Projekt\prog\Frequenzzaehler\lcd.c
129:
130:
131: if (rs) { /* write data (RS=1, RW=0) */
132: lcd_rs_high();
133: } else { /* write instruction (RS=0, RW=0) */
134: lcd_rs_low();
135: }
136: lcd_rw_low();
137:
138: if ( ( &LCD_DATA0_PORT == &LCD_DATA1_PORT) && ( &LCD_DATA1_PORT ==
&LCD_DATA2_PORT ) && ( &LCD_DATA2_PORT == &LCD_DATA3_PORT )
139: && (LCD_DATA0_PIN == 0) && (LCD_DATA1_PIN == 1) && (LCD_DATA2_PIN
== 2) && (LCD_DATA3_PIN == 3) )
140: {
141: /* configure data pins as output */
142: DDR(LCD_DATA0_PORT) |= 0x0F;
143:
144: /* output high nibble first */
145: dataBits = LCD_DATA0_PORT & 0xF0;
146: LCD_DATA0_PORT = dataBits |((data>>4)&0x0F);
147: lcd_e_toggle();
148:
149: /* output low nibble */
150: LCD_DATA0_PORT = dataBits | (data&0x0F);
151: lcd_e_toggle();
152:
153: /* all data pins high (inactive) */
154: LCD_DATA0_PORT = dataBits | 0x0F;
155: }
```

```
156: else
157: {
158: /* configure data pins as output */
159: DDR(LCD_DATA0_PORT) |= _BV(LCD_DATA0_PIN);
160: DDR(LCD_DATA1_PORT) |= _BV(LCD_DATA1_PIN);
161: DDR(LCD_DATA2_PORT) |= _BV(LCD_DATA2_PIN);
162: DDR(LCD_DATA3_PORT) |= _BV(LCD_DATA3_PIN);
163:
164: /* output high nibble first */
165: LCD_DATA3_PORT &= ~_BV(LCD_DATA3_PIN);
166: LCD_DATA2_PORT &= ~_BV(LCD_DATA2_PIN);
167: LCD_DATA1_PORT &= ~_BV(LCD_DATA1_PIN);
168: LCD_DATA0_PORT &= ~_BV(LCD_DATA0_PIN);
169: if(data & 0x80) LCD_DATA3_PORT |= _BV(LCD_DATA3_PIN);
170: if(data & 0x40) LCD_DATA2_PORT |= _BV(LCD_DATA2_PIN);
171: if(data & 0x20) LCD_DATA1_PORT |= _BV(LCD_DATA1_PIN);
172: if(data & 0x10) LCD_DATA0_PORT |= _BV(LCD_DATA0_PIN);
173: lcd_e_toggle();
174:
175: /* output low nibble */
176: LCD_DATA3_PORT &= ~_BV(LCD_DATA3_PIN);
177: LCD_DATA2_PORT &= ~_BV(LCD_DATA2_PIN);
178: LCD_DATA1_PORT &= ~_BV(LCD_DATA1_PIN);
179: LCD_DATA0_PORT &= ~_BV(LCD_DATA0_PIN);
180: if(data & 0x08) LCD_DATA3_PORT |= _BV(LCD_DATA3_PIN);
181: if(data & 0x04) LCD_DATA2_PORT |= _BV(LCD_DATA2_PIN);
182: if(data & 0x02) LCD_DATA1_PORT |= _BV(LCD_DATA1_PIN);
183: if(data & 0x01) LCD_DATA0_PORT |= _BV(LCD_DATA0_PIN);
184: lcd_e_toggle();
185:
186: /* all data pins high (inactive) */
187: LCD_DATA0_PORT |= _BV(LCD_DATA0_PIN);
188: LCD_DATA1_PORT |= _BV(LCD_DATA1_PIN);
189: LCD_DATA2_PORT |= _BV(LCD_DATA2_PIN);
190: LCD_DATA3_PORT |= _BV(LCD_DATA3_PIN);
191: }
192: }
193: #else
194: #define lcd_write(d,rs) if (rs) *(volatile uint8_t*)(LCD_IO_DATA) = d;
else *(volatile uint8_t*)(LCD_IO_FUNCTION) = d;
195: /* rs==0 -> write instruction to LCD_IO_FUNCTION */
196: /* rs==1 -> write data to LCD_IO_DATA */
D:\juergen\Projekt\Projekt\prog\Frequenzzaehler\lcd.c
197: #endif
198:
199:
200:
/*************************************************************************
*
201: Low-level function to read byte from LCD controller
202: Input: rs 1: read data
203: 0: read busy flag / address counter
204: Returns: byte read from LCD controller
205:
*************************************************************************
/
206: #if LCD_IO_MODE
207: static uint8_t lcd_read(uint8_t rs)
208: {
209: uint8_t data;
210:
211:
212: if (rs)
213: lcd_rs_high(); /* RS=1: read data */
214: else
215: lcd_rs_low(); /* RS=0: read busy flag */
216: lcd_rw_high(); /* RW=1 read mode */
217:
218: if ( ( &LCD_DATA0_PORT == &LCD_DATA1_PORT) && ( &LCD_DATA1_PORT ==
&LCD_DATA2_PORT ) && ( &LCD_DATA2_PORT == &LCD_DATA3_PORT )
219: && ( LCD_DATA0_PIN == 0 )&& (LCD_DATA1_PIN == 1) && (LCD_DATA2_PIN
```

```
== 2) && (LCD_DATA3_PIN == 3) )
220: {
221: DDR(LCD_DATA0_PORT) &= 0xF0; /* configure data pins as
input */
222:
223: lcd_e_high();
224: lcd_e_delay();
225: data = PIN(LCD_DATA0_PORT) << 4; /* read high nibble first */
226: lcd_e_low();
227:
228: lcd_e_delay(); /* Enable 500ns low */
229:
230: lcd_e_high();
231: lcd_e_delay();
232: data |= PIN(LCD_DATA0_PORT)&0x0F; /* read low nibble */
233: lcd_e_low();
234: }
235: else
236: {
237: /* configure data pins as input */
238: DDR(LCD_DATA0_PORT) &= ~_BV(LCD_DATA0_PIN);
239: DDR(LCD_DATA1_PORT) &= ~_BV(LCD_DATA1_PIN);
240: DDR(LCD_DATA2_PORT) &= ~_BV(LCD_DATA2_PIN);
241: DDR(LCD_DATA3_PORT) &= ~_BV(LCD_DATA3_PIN);
242:
243: /* read high nibble first */
244: lcd_e_high();
245: lcd_e_delay();
246: data = 0;
247: if ( PIN(LCD_DATA0_PORT) & _BV(LCD_DATA0_PIN) ) data |= 0x10;
248: if ( PIN(LCD_DATA1_PORT) & _BV(LCD_DATA1_PIN) ) data |= 0x20;
249: if ( PIN(LCD_DATA2_PORT) & _BV(LCD_DATA2_PIN) ) data |= 0x40;
250: if ( PIN(LCD_DATA3_PORT) & _BV(LCD_DATA3_PIN) ) data |= 0x80;
251: lcd_e_low();
252:
253: lcd_e_delay(); /* Enable 500ns low */
254:
255: /* read low nibble */
256: lcd_e_high();
257: lcd_e_delay();
258: if ( PIN(LCD_DATA0_PORT) & _BV(LCD_DATA0_PIN) ) data |= 0x01;
259: if ( PIN(LCD_DATA1_PORT) & _BV(LCD_DATA1_PIN) ) data |= 0x02;
260: if ( PIN(LCD_DATA2_PORT) & _BV(LCD_DATA2_PIN) ) data |= 0x04;
261: if ( PIN(LCD_DATA3_PORT) & _BV(LCD_DATA3_PIN) ) data |= 0x08;
262: lcd_e_low();
D:\juergen\Projekt\Projekt\prog\Frequenzzaehler\lcd.c
263: }
264: return data;
265: }
266: #else
267: #define lcd_read(rs) (rs) ? *(volatile
uint8_t*)(LCD_IO_DATA+LCD_IO_READ)
: *(volatile uint8_t*)(LCD_IO_FUNCTION+LCD_IO_READ)
268: /* rs==0 -> read instruction from LCD_IO_FUNCTION */
269: /* rs==1 -> read data from LCD_IO_DATA */
270: #endif
271:
272:
273:
/*************************************************************************
*
274: loops while lcd is busy, returns address counter
275:
*************************************************************************
/
276: static uint8_t lcd_waitbusy(void)
277:
278: {
279: register uint8_t c;
280:
281: /* wait until busy flag is cleared */
```

```
282: while ( (c=lcd_read(0)) & (1<<LCD_BUSY)) {}
283:
284: /* the address counter is updated 4us after the busy flag is cleared
*/
285: delay(2);
286:
287: /* now read the address counter */
288: return (lcd_read(0)); // return address counter
289:
290: }/* lcd_waitbusy */
291:
292:
293:
/*************************************************************************
*
294: Move cursor to the start of next line or to the first line if the
cursor
295: is already on the last line.
296:
*************************************************************************
/
297: static inline void lcd_newline(uint8_t pos)
298: {
299: register uint8_t addressCounter;
300:
301:
302: #if LCD_LINES==1
303: addressCounter = 0;
304: #endif
305: #if LCD_LINES==2
306: if ( pos < (LCD_START_LINE2) )
307: addressCounter = LCD_START_LINE2;
308: else
309: addressCounter = LCD_START_LINE1;
310: #endif
311: #if LCD_LINES==4
312: if ( pos < LCD_START_LINE3 )
313: addressCounter = LCD_START_LINE2;
314: else if ( (pos >= LCD_START_LINE2) && (pos < LCD_START_LINE4) )
315: addressCounter = LCD_START_LINE3;
316: else if ( (pos >= LCD_START_LINE3) && (pos < LCD_START_LINE2) )
317: addressCounter = LCD_START_LINE4;
318: else
319: addressCounter = LCD_START_LINE1;
320: #endif
321: lcd_command((1<<LCD_DDRAM)+addressCounter);
322:
323: }/* lcd_newline */
324:
325:
326: /*
327: ** PUBLIC FUNCTIONS
D:\juergen\Projekt\Projekt\prog\Frequenzzaehler\lcd.c
328: */
329:
330:
/*************************************************************************
*
331: Send LCD controller instruction command
332: Input: instruction to send to LCD controller, see HD44780 data sheet
333: Returns: none
334:
*************************************************************************
/
335: void lcd_command(uint8_t cmd)
336: {
337: lcd_waitbusy();
338: lcd_write(cmd,0);
339: }
340:
341:
```

```
342:
/*************************************************************************
*
343: Send data byte to LCD controller
344: Input: data to send to LCD controller, see HD44780 data sheet
345: Returns: none
346:
*************************************************************************
/
347: void lcd_data(uint8_t data)
348: {
349: lcd_waitbusy();
350: lcd_write(data,1);
351: }
352:
353:
354:
355:
/*************************************************************************
*
356: Set cursor to specified position
357: Input: x horizontal position (0: left most position)
358: y vertical position (0: first line)
359: Returns: none
360:
*************************************************************************
/
361: void lcd_gotoxy(uint8_t x, uint8_t y)
362: {
363: #if LCD_LINES==1
364: lcd_command((1<<LCD_DDRAM)+LCD_START_LINE1+x);
365: #endif
366: #if LCD_LINES==2
367: if ( y==0 )
368: lcd_command((1<<LCD_DDRAM)+LCD_START_LINE1+x);
369: else
370: lcd_command((1<<LCD_DDRAM)+LCD_START_LINE2+x);
371: #endif
372: #if LCD_LINES==4
373: if ( y==0 )
374: lcd_command((1<<LCD_DDRAM)+LCD_START_LINE1+x);
375: else if ( y==1)
376: lcd_command((1<<LCD_DDRAM)+LCD_START_LINE2+x);
377: else if ( y==2)
378: lcd_command((1<<LCD_DDRAM)+LCD_START_LINE3+x);
379: else /* y==3 */
380: lcd_command((1<<LCD_DDRAM)+LCD_START_LINE4+x);
381: #endif
382:
383: }/* lcd_gotoxy */
384:
385:
386:
/*************************************************************************
*
387:
*************************************************************************
/
388: int lcd_getxy(void)
389: {
390: return lcd_waitbusy();
D:\juergen\Projekt\Projekt\prog\Frequenzzaehler\lcd.c
391: }
392:
393:
394:
/*************************************************************************
*
395: Clear display and set cursor to home position
396:
*************************************************************************
```

```
/
397: void lcd_clrscr(void)
398: {
399: lcd_command(1<<LCD_CLR);
400: }
401:
402:
403:
/*************************************************************************
*
404: Set cursor to home position
405:
*************************************************************************
/
406: void lcd_home(void)
407: {
408: lcd_command(1<<LCD_HOME);
409: }
410:
411:
412:
/*************************************************************************
*
413: Display character at current cursor position
414: Input: character to be displayed
415: Returns: none
416:
*************************************************************************
/
417: void lcd_putc(char c)
418: {
419: uint8_t pos;
420:
421:
422: pos = lcd_waitbusy(); // read busy-flag and address counter
423: if (c=='\n')
424: {
425: lcd_newline(pos);
426: }
427: else
428: {
429: #if LCD_WRAP_LINES==1
430: #if LCD_LINES==1
431: if ( pos == LCD_START_LINE1+LCD_DISP_LENGTH ) {
432: lcd_write((1<<LCD_DDRAM)+LCD_START_LINE1,0);
433: }
434: #elif LCD_LINES==2
435: if ( pos == LCD_START_LINE1+LCD_DISP_LENGTH ) {
436: lcd_write((1<<LCD_DDRAM)+LCD_START_LINE2,0);
437: }else if ( pos == LCD_START_LINE2+LCD_DISP_LENGTH ){
438: lcd_write((1<<LCD_DDRAM)+LCD_START_LINE1,0);
439: }
440: #elif LCD_LINES==4
441: if ( pos == LCD_START_LINE1+LCD_DISP_LENGTH ) {
442: lcd_write((1<<LCD_DDRAM)+LCD_START_LINE2,0);
443: }else if ( pos == LCD_START_LINE2+LCD_DISP_LENGTH ) {
444: lcd_write((1<<LCD_DDRAM)+LCD_START_LINE3,0);
445: }else if ( pos == LCD_START_LINE3+LCD_DISP_LENGTH ) {
446: lcd_write((1<<LCD_DDRAM)+LCD_START_LINE4,0);
447: }else if ( pos == LCD_START_LINE4+LCD_DISP_LENGTH ) {
448: lcd_write((1<<LCD_DDRAM)+LCD_START_LINE1,0);
449: }
450: #endif
451: lcd_waitbusy();
452: #endif
453: lcd_write(c, 1);
454: }
455:
D:\juergen\Projekt\Projekt\prog\Frequenzzaehler\lcd.c
456: }/* lcd_putc */
457:
```

```
458:
459:
/*************************************************************************
*
460: Display string without auto linefeed
461: Input: string to be displayed
462: Returns: none
463:
*************************************************************************
/
464: void lcd_puts(const char *s)
465: /* print string on lcd (no auto linefeed) */
466: {
467: register char c;
468:
469: while ( (c = *s++) ) {
470: lcd_putc(c);
471: }
472:
473: }/* lcd_puts */
474:
475:
476:
/*************************************************************************
*
477: Display string from program memory without auto linefeed
478: Input: string from program memory be be displayed
479: Returns: none
480:
*************************************************************************
/
481: void lcd_puts_p(const char *progmem_s)
482: /* print string from program memory on lcd (no auto linefeed) */
483: {
484: register char c;
485:
486: while ( (c = pgm_read_byte(progmem_s++)) ) {
487: lcd_putc(c);
488: }
489:
490: }/* lcd_puts_p */
491:
492:
493:
/*************************************************************************
*
494: Initialize display and select type of cursor
495: Input: dispAttr LCD_DISP_OFF display off
496: LCD_DISP_ON display on, cursor off
497: LCD_DISP_ON_CURSOR display on, cursor on
498: LCD_DISP_CURSOR_BLINK display on, cursor on flashing
499: Returns: none
500:
*************************************************************************
/
501: void lcd_init(uint8_t dispAttr)
502: {
503: #if LCD_IO_MODE
504: /*
505: * Initialize LCD to 4 bit I/O mode
506: */
507:
508: if ( ( &LCD_DATA0_PORT == &LCD_DATA1_PORT) && ( &LCD_DATA1_PORT ==
&LCD_DATA2_PORT ) && ( &LCD_DATA2_PORT == &LCD_DATA3_PORT )
509: && ( &LCD_RS_PORT == &LCD_DATA0_PORT) && ( &LCD_RW_PORT ==
&LCD_DATA0_PORT) && (&LCD_E_PORT == &LCD_DATA0_PORT)
510: && (LCD_DATA0_PIN == 0 ) && (LCD_DATA1_PIN == 1) && (LCD_DATA2_PIN
== 2) && (LCD_DATA3_PIN == 3)
511: && (LCD_RS_PIN == 4 ) && (LCD_RW_PIN == 5) && (LCD_E_PIN == 6 ) )
512: {
513: /* configure all port bits as output (all LCD lines on same port)
```

```
*/
514: DDR(LCD_DATA0_PORT) |= 0x7F;
515: }
516: else if ( ( &LCD_DATA0_PORT == &LCD_DATA1_PORT) && ( &LCD_DATA1_PORT
D:\juergen\Projekt\Projekt\prog\Frequenzzaehler\lcd.c
== &LCD_DATA2_PORT ) && ( &LCD_DATA2_PORT == &LCD_DATA3_PORT )
517: && (LCD_DATA0_PIN == 0 ) && (LCD_DATA1_PIN == 1) &&
(LCD_DATA2_PIN == 2) && (LCD_DATA3_PIN == 3) )
518: {
519: /* configure all port bits as output (all LCD data lines on same
port, but control lines on different ports) */
520: DDR(LCD_DATA0_PORT) |= 0x0F;
521: DDR(LCD_RS_PORT) |= _BV(LCD_RS_PIN);
522: DDR(LCD_RW_PORT) |= _BV(LCD_RW_PIN);
523: DDR(LCD_E_PORT) |= _BV(LCD_E_PIN);
524: }
525: else
526: {
527: /* configure all port bits as output (LCD data and control lines
on different ports */
528: DDR(LCD_RS_PORT) |= _BV(LCD_RS_PIN);
529: DDR(LCD_RW_PORT) |= _BV(LCD_RW_PIN);
530: DDR(LCD_E_PORT) |= _BV(LCD_E_PIN);
531: DDR(LCD_DATA0_PORT) |= _BV(LCD_DATA0_PIN);
532: DDR(LCD_DATA1_PORT) |= _BV(LCD_DATA1_PIN);
533: DDR(LCD_DATA2_PORT) |= _BV(LCD_DATA2_PIN);
534: DDR(LCD_DATA3_PORT) |= _BV(LCD_DATA3_PIN);
535: }
536: delay(16000); /* wait 16ms or more after power-on */
537:
538: /* initial write to lcd is 8bit */
539: LCD_DATA1_PORT |= _BV(LCD_DATA1_PIN); // _BV(LCD_FUNCTION)>>4;
540: LCD_DATA0_PORT |= _BV(LCD_DATA0_PIN); // _BV(LCD_FUNCTION_8BIT)>>4;
541: lcd_e_toggle();
542: delay(4992); /* delay, busy flag can't be checked here */
543:
544: /* repeat last command */
545: lcd_e_toggle();
546: delay(64); /* delay, busy flag can't be checked here */
547:
548: /* repeat last command a third time */
549: lcd_e_toggle();
550: delay(64); /* delay, busy flag can't be checked here */
551:
552: /* now configure for 4bit mode */
553: LCD_DATA0_PORT &= ~_BV(LCD_DATA0_PIN); //
LCD_FUNCTION_4BIT_1LINE>>4
554: lcd_e_toggle();
555: delay(64); /* some displays need this additional delay */
556:
557: /* from now the LCD only accepts 4 bit I/O, we can use lcd_command()
*/
558: #else
559: /*
560: * Initialize LCD to 8 bit memory mapped mode
561: */
562:
563: /* enable external SRAM (memory mapped lcd) and one wait state */
564: MCUCR = _BV(SRE) | _BV(SRW);
565:
566: /* reset LCD */
567: delay(16000); /* wait 16ms after power-on
*/
568: lcd_write(LCD_FUNCTION_8BIT_1LINE,0); /* function set: 8bit
interface */
569: delay(4992); /* wait 5ms
*/
570: lcd_write(LCD_FUNCTION_8BIT_1LINE,0); /* function set: 8bit
interface */
571: delay(64); /* wait 64us
*/
```

```
572: lcd_write(LCD_FUNCTION_8BIT_1LINE,0); /* function set: 8bit
interface */
573: delay(64); /* wait 64us
*/
574: #endif
D:\juergen\Projekt\Projekt\prog\Frequenzzaehler\lcd.c
575: lcd_command(LCD_FUNCTION_DEFAULT); /* function set: display
lines */
576: lcd_command(LCD_DISP_OFF); /* display off
*/
577: lcd_clrscr(); /* display clear
*/
578: lcd_command(LCD_MODE_DEFAULT); /* set entry mode
*/
579: lcd_command(dispAttr); /* display/cursor control
*/
580:
581: }/* lcd_init */
```

10.4. Anhänge der Gruppe 4

Programmcode

```
;#########################################################################
######################
;#                                                                       #
;#                    Prolab Frequency Display Control                   #
;#                            251105 V.0.0.3                             #
;#                 Philipp Fabian Benedikt Maier & Daniel Hahn           #
;#                                                                       #
;#########################################################################
######################


;#                                  CIRCUIT                              DESCRIPTION
#########################################################################
;
;
;                    _________
;       TO            -|RST V PC5|-
; FREQUNECY <== UART ------|PD0   PC4|-
;  ANALYSER    CTRLA ------|PD1   PC3|------D
;     UNIT    CTRLB ------|PD2   PC2|------C ==> To Demux
;                    -|PD3   PC1|------B
;                    -|PD4   PC0|------A
;                    -|VCC   GND|-
;                    -|GND   REF|-
;                    -|CLK   VCC|-
;                    -|CLK   PB5|-
;                    -|PD5   PB4|-
;                    -|PD6   PB3|------D
;                    -|PD7   PB2|------C ==> To BCD-Decoder
;               +------|PB0   PB1|------B
;               |      |_________|   +--A
;               +--------------------+
;
; Communication Protocol:
; Recive:
;   Date will be reived via UART in enumerated packet form. The first 4 bits are
;   reserved for enumeration (sequence number) the last 4 bits can be used to
;   transfer data.
;   for exemple: 00011011 First Packet, contains the first lower halfbyte 1011
;                00101011 Sacond Packet, contains the first higher halfbyte 0011
;                ==> Data will be stored in regLowerValueByte as 10110011
;                00111011 Third Packet, contains the sacond lower halfbyte 1111
;                01001011 Fourth Packet, contains the sacond higher halfbye 1010
```

```
;               ==> Data will be stored in regHigerValueByte as 11111010
;               Now the sequence number will reset and the process starts again
;
; Control:
;    From time to time this IC will request you for three different values (max, min, courrent)
;     This is easiely done by two wires (PD1, PD2) which transfer the ID of the transfered
;   value as it is shown in the following table:
;   01 Minimum Value
;   10 Maximum Value
;   11 Courrent Value
;
;#########################################################################
########################


;#                                ASSEMBLER                                INITALISATION
#########################################################################
;//Params
.include "m8def.inc"                        ;//We use Atmel Mega8 for our Application
.org 0x00                                   ;//Lets write the programm to 0x00

;//Register Declaration
.def regTemp = r16                          ;//This Register will be used for temorary values (eg. counters) all over the program
.def regLowerValueByte = r17                ;//This Register contains the first 8 Bit of the transfered value
.def regHigerValueByte = r18                ;//This Register contains the last 8 Bit of the transfered value
.def regParam = r19                         ;//This Register will be used to bring parameters to the sub routines

;//Baud-Rate
.equ CLOCK = 4000000
.equ BAUD = 9600
.equ UBRRVAL = CLOCK/(BAUD*16)-1

;//Start Programm
rjmp boot
;#########################################################################
########################


;#                                                                        BOOT
#########################################################################
###############
;//Interrupt Vector
.org URXCaddr
```

```
        rjmp intReciveByteFromUart

;//Boot Process
boot:

;//Initalize Ports
ldi regTemp, 0b00001111                         ;//Port                    B:
PB0,PB1,PB2,PB3:=OUTPUT
out DDRB, regTemp
ldi regTemp, 0b00001111                         ;//Port                    C:
PC0,PC1,PC2,PC3:=OUTPUT
out DDRC, regTemp
ldi regTemp, 0b00000110                         ;//Port   D:   PD0:=UARTINPUT
PD1,PD2:=OUTPUT                         ;
out DDRD, regTemp

;//Clear Data
ldi r16,0x00                                    ;//Clear Register 16
ldi r17,0x00                                    ;//Clear Register 17
ldi r18,0x00                                    ;//Clear Register 18
ldi r19,0x00                                    ;//Clear Register 19
ldi r20,0x00                                    ;//Clear Register 20
ldi r21,0x00                                    ;//Clear Register 21
ldi r22,0x00                                    ;//Clear Register 22
ldi r23,0x00                                    ;//Clear Register 23
ldi r24,0x00                                    ;//Clear Register 24
ldi r25,0x00                                    ;//Clear Register 25
ldi r26,0x00                                    ;//Clear Register 26
ldi r27,0x00                                    ;//Clear Register 27
ldi r28,0x00                                    ;//Clear Register 28
ldi r29,0x00                                    ;//Clear Register 29
ldi r30,0x00                                    ;//Clear Register 30
ldi r31,0x00                                    ;//Clear Register 31

out PORTB, r16                                  ;//Clear PortB
out PORTC, r16                                  ;//Clear PortC
out PORTD, r16                                  ;//Clear PortD

;//Initalize Stack
ldi regTemp, LOW(RAMEND)            ;//LOW-Byte of upper ram Adress
out SPL, regTemp
ldi regTemp, HIGH(RAMEND)           ;//HIGH-Byte of downer ram adress
out SPH, regTemp

;//Initalize Uart
ldi regTemp, LOW(UBRRVAL)                       ;//Setup Baudrate
out UBRRL, regTemp
ldi regTemp, HIGH(UBRRVAL)
out UBRRH, regTemp
ldi regTemp, (1<<URSEL)|(3<<UCSZ0)              ;//Setup 8-Bit Frame-Format
out UCSRC, regTemp
```

```
sbi UCSRB, RXCIE                    ;//Activate Interrupt controlled recivement
sbi UCSRB, RXEN                     ;//Activate Reciver

sei                                 ;//Activate Interrupts Global
;#########################################################################
#######################

;#                                                                    CORE
#########################################################################
###############
core:                                                   ;//Start Core Application

;//Test and Debug Routines
;rcall subDisplayTest                                   ;//Perform a full featured
Display Test
;rcall subLampTest                                      ;//Perform a simple Lamp Test
;rcall subSpecificLampTest                              ;//Perform a specific lamp test

rcall intReciveByteFromUart

rjmp core                                               ;//Restart Core
Application
;#########################################################################
#######################

;#                                  FETCH                                VALUES
#########################################################################
#######
;//Fetch the Maximum Frequency Value from Analyser Unit
subFetchMaximumFrequency:
push regTemp                                            ;//Save temp to stack

;10 Maximum Value
ldi regTemp, 0b00000100
out PORTD, regTemp                                      ;//Poll request

pop regTemp                 ;//Restore temp
ret
```

```
;//Fetch the Minimum Frequency Value from Analyser Unit
subFetchMinimumFrequency:
push regTemp                                    ;//Save temp to stack

;01 Minimum Value
ldi regTemp, 0b00000010
out PORTD, regTemp                              ;//Poll request

pop regTemp                 ;//Restore temp
ret



;//Fetch the Minimum Frequency Value from Analyser Unit
subFetchCourrentFrequency:
push regTemp                                    ;//Save temp to stack

;11 Courrent Value
ldi regTemp, 0b00000110
out PORTD, regTemp                              ;//Poll request

pop regTemp                 ;//Restore temp
ret
;#########################################################################
######################

;#                                                                  DISPLAY
#########################################################################
############
subDisplayValue:

ret
;#########################################################################
######################

;#                    RECIVE                    UART                    BYTE
#########################################################################
###
;//Recive Byte via Uart
intReciveByteFromUart:
push regTemp                                    ;//Save temp to stack

in regTemp, UDR                                 ;//Recive Data from serial port
;ldi regTemp, 0b01001011            ;//Inject a fix value for debug (uncommit this if
you whish to debug)
```

```
;//Look for the first Packet (SEQ=0001)
push regTemp                                    ;//Save temp to stack (we need to destroy it because we want to filter sequence number)
andi regTemp, 0b11110000                        ;//Filter sequence number
cpi regTemp,0b00010000                          ;//Check if sequence number is 0001 (first packet)
pop regTemp                     ;//Restore temp (we had to destroy it because we wanted to filter out the sequence number)
breq intReciveByteFromUart_storebyte_1 ;//Store byte
rjmp intReciveByteFromUart_cancel_1 ;//Cancel if the sequence number does not fit.
intReciveByteFromUart_storebyte_1:  ;//The packet was for us, we will store it
push regTemp                                    ;//Save temp to stack (we need to destroy it because we want to filter data)
andi regTemp, 0b00001111                        ;//Ok, the packet fits, lets forget the sequence number
andi regLowerValueByte, 0b11110000  ;//Forget the old halfbyte value in data register
add regLowerValueByte, regTemp      ;//Update halfbyte
pop regTemp                     ;//Restore temp (we had to destroy it because we wanted to filter data)
intReciveByteFromUart_cancel_1:                 ;//The packet was not for us, do nothing and continue.

;//Look for the sacond Packet (SEQ=0010)
push regTemp                                    ;//Save temp to stack (we need to destroy it because we want to filter sequence number)
andi regTemp, 0b11110000                        ;//Filter sequence number
cpi regTemp,0b00100000                          ;//Check if sequence number is 0010 (sacond packet)
pop regTemp                     ;//Restore temp (we had to destroy it because we wanted to filter out the sequence number)
breq intReciveByteFromUart_storebyte_2 ;//Store byte
rjmp intReciveByteFromUart_cancel_2 ;//Cancel if the sequence number does not fit.
intReciveByteFromUart_storebyte_2:  ;//The packet was for us, we will store it
push regTemp                                    ;//Save temp to stack (we need to destroy it because we want to filter data)
andi regTemp, 0b00001111                        ;//Ok, the packet fits, lets forget the sequence number
andi regLowerValueByte, 0b00001111  ;//Forget the old halfbyte value in data register
rol regTemp                                     ;//Rotate value until it fits...
rol regTemp                                     ;//Rotate value until it fits...
rol regTemp                                     ;//Rotate value until it fits...
rol regTemp                                     ;//Rotate value until it fits...
add regLowerValueByte, regTemp      ;//Update halfbyte
pop regTemp                     ;//Restore temp (we had to destroy it because we wanted to filter data)
intReciveByteFromUart_cancel_2:                 ;//The packet was not for us, do nothing and continue.

;//Look for the third Packet (SEQ=0011)
```

```
push regTemp                                    ;//Save temp to stack (we need to destroy it because we want to filter sequence number)
andi regTemp, 0b11110000                        ;//Filter sequence number
cpi regTemp,0b00110000                          ;//Check if sequence number is 0011 (third packet)
pop regTemp                     ;//Restore temp (we had to destroy it because we wanted to filter out the sequence number)
breq intReciveByteFromUart_storebyte_3 ;//Store byte
rjmp intReciveByteFromUart_cancel_3 ;//Cancel if the sequence number does not fit.
intReciveByteFromUart_storebyte_3:  ;//The packet was for us, we will store it
push regTemp                                    ;//Save temp to stack (we need to destroy it because we want to filter data)
andi regTemp, 0b00001111                        ;//Ok, the packet fits, lets forget the sequence number
andi regLowerValueByte, 0b11110000  ;//Forget the old halfbyte value in data register
add regHigerValueByte, regTemp      ;//Update halfbyte
pop regTemp                     ;//Restore temp (we had to destroy it because we wanted to filter data)
intReciveByteFromUart_cancel_3:                 ;//The packet was not for us, do nothing and continue.

;//Look for the fourth Packet (SEQ=0100)
push regTemp                                    ;//Save temp to stack (we need to destroy it because we want to filter sequence number)
andi regTemp, 0b11110000                        ;//Filter sequence number
cpi regTemp,0b01000000                          ;//Check if sequence number is 0100 (fourth packet)
pop regTemp                     ;//Restore temp (we had to destroy it because we wanted to filter out the sequence number)
breq intReciveByteFromUart_storebyte_4 ;//Store byte
rjmp intReciveByteFromUart_cancel_4 ;//Cancel if the sequence number does not fit.
intReciveByteFromUart_storebyte_4:  ;//The packet was for us, we will store it
push regTemp                                    ;//Save temp to stack (we need to destroy it because we want to filter data)
andi regTemp, 0b00001111                        ;//Ok, the packet fits, lets forget the sequence number
andi regLowerValueByte, 0b00001111  ;//Forget the old halfbyte value in data register
rol regTemp                                     ;//Rotate value until it fits...
rol regTemp                                     ;//Rotate value until it fits...
rol regTemp                                     ;//Rotate value until it fits...
rol regTemp                                     ;//Rotate value until it fits...
add regHigerValueByte, regTemp      ;//Update halfbyte
pop regTemp                     ;//Restore temp (we had to destroy it because we wanted to filter data)
intReciveByteFromUart_cancel_4:                 ;//The packet was not for us, do nothing and continue.

;//Check if injected packet was stored correctly (uncommit this if you whish to debug)
```

```
;cpi regHigerValueByte,0b10110000        ;//Check if regHigerValueByte has a
                                         specific value

;breq stopthis                           ;//... and stop here
;rjmp dontstop                           ;//... or continue
;stopthis:                               ;//stop here!
;rjmp stopthis
;dontstop:                               ;//go on and do nothing...

pop regTemp                 ;//Restore temp
reti
;#####################################################################
######################

;#                                DISPLAY                                TEST
#####################################################################
#######

;//Perform a full featured Display Test
subDisplayTest:
push regTemp                            ;//Save temp to stack

;// Toggle everz number once
ldi regTemp, 0x00                       ;//Reset temp
goon2_subDisplayTest:
out PORTB, regTemp                      ;//Send Data
ldi regParam, 0x20          ;//Setup how long to delay
rcall delay                             ;//Wait a short time
cpi regTemp, 0b00001010                 ;//IF regTemp = 00001111 then...
breq escape2_subDisplayTest             ;//...goto escape2_subDisplayTest
inc regTemp                             ;//regTemp = regTemp + 1

;// Toggle every digit once
push regTemp                            ;//Save temp to stack
ldi regTemp, 0x00                       ;//Reset temp
goon_subDisplayTest:
out PORTC, regTemp                      ;//Send Data

ldi regParam, 0x10          ;//Setup how long to delay
rcall delay                             ;//Wait a short time

cpi regTemp, 0b00001000                 ;//IF regTemp = 00001111 then...
breq escape_subDisplayTest              ;//...goto escape_subDisplayTest
inc regTemp                             ;//regTemp = regTemp + 1
rjmp goon_subDisplayTest                ;//goto goon_subDisplayTest
escape_subDisplayTest:                  ;//Leave This Part
pop regTemp                             ;//Restore temp
;// Done toggle every digit once

rjmp goon2_subDisplayTest               ;//goto goon2_subDisplayTest
```

```
escape2_subDisplayTest:                 ;//Leave This Routine
;// Done toggle every number once

pop regTemp                             ;//Restore temp
ret



;//Perform a simple Lamp Test
subLampTest:
push regTemp                            ;//Save temp to stack

;// Toggle all lamps
ldi regTemp, 0b00001000                 ;// Write a "8"
out PORTB, regTemp

;// Toggle every digit once
push regTemp                            ;//Save temp to stack
ldi regTemp, 0x00                       ;//Reset temp
goon_subLampTest:
out PORTC, regTemp                      ;//Send Data

ldi regParam, 0x20          ;//Setup how long to delay
rcall delay                             ;//Wait a short time

cpi regTemp, 0b00001000                 ;//IF regTemp = 00001111 then...
breq escape_subLampTest                 ;//...goto escape_subLampTest
inc regTemp                             ;//regTemp = regTemp + 1
rjmp goon_subLampTest                   ;//goto goon_subLampTest
escape_subLampTest:                     ;//Leave This Part
pop regTemp                             ;//Restore temp
;// Done toggle every digit once

pop regTemp                             ;//Restore temp
ret



;//Perform a specific lamp test
subSpecificLampTest:
push regTemp                            ;//Save temp to stack

;// Configure Output Values
ldi regTemp, 0b00000101                 ;//Configure Number
push regTemp                            ;//Save temp to stack
ldi regTemp, 0b00000000                 ;//Configure Digit

;// Poll Data
out PORTC, regTemp                      ;//Send Digit Data
pop regTemp                             ;//Restore temp
out PORTB, regTemp          ;//Send Number Data
```

```
pop regTemp                                                    ;//Restore temp
ret
;#########################################################################
######################

;#                                SYSTEM                                CALLS
#########################################################################
#######
;// Valid System Calls are:
;// boot: Boot the System (Already done when switched on)
;// reboot: Forget everything and restart
;// halt: Halt the whole programm
;// core: Load Core Programm (Already done when switched on)
;// delay: Wait a short time (preconfigure how long to delay in regParam)



;//Delay the programm
delay:
push regTemp                                          ;//Save temp to stack

;// Primary delay counter
ldi regTemp, 0x00                              ;//Reset temp
goon_delay:
cp regTemp, regParam                  ;//IF regTemp = regParam then...
breq leave_delay                      ;//...goto leave_delay
inc regTemp                           ;//regTemp = regTemp + 1

;// Secondary delay counter
push regTemp                          ;//Save temp to stack
ldi regTemp, 0x00                     ;//Reset temp
goon2_delay:
cp regTemp, regParam                  ;//IF regTemp = regParam then...
breq leave2_delay                     ;//...goto leave_delay
inc regTemp                           ;//regTemp = regTemp + 1
rjmp goon2_delay                      ;//goto goon2_delay
leave2_delay:                         ;//leave delay
pop regTemp                           ;//Restore temp

rjmp goon_delay                       ;//goto goon_delay
leave_delay:                          ;//leave delay
pop regTemp                           ;//Restore temp
ret

;//Halt Programm
halt:
stay_halted:
rjmp stay_halted
ret
```

```
;//Reboot Programm
reboot:
rjmp boot                                                    ;//Reboot
ret
;#########################################################################
#######################
```

10.5. Anhänge der Gruppe 5

10.6. Datenblätter

Die Datenblätter sind in alphabetischer Reihenfolge sortiert.

2N2222A
2n3055n_datasheet
2N3055NPN
7-Segment_38mm
74HC_HCT00_3
74HCT193
7404
7408
74154
Akku
Analog Schalter
AtmelMega8AVR
BC_559_B_BC_560_C
BCY58
C549C
DAC1020
DG417
KTY_81_220
LF347
LM118
LM124X_4
LM318
LM319
LM324N
LM393
LM3914_NSC
PN2222A
SN_74154