hochschule mannheim

Hochschule Mannheim

Institut für Regelungs- und Prozessleittechnik

Paul-Wittsack-Str. 10,D- 68163 Mannheim

Bachelorarbeit

Thema: Entwicklung eines Zündsteuergerätes auf Basis eines PIC 18F2550 incl. Einer PC-Bediensoftware

Verfasser: Adam Horn

Matrikel Nr.: 1020925

Studiengang: Technische Informatik

Erstkorrektur: Prof. Dr. –Ing. Faulhaber

Zweitkorrektur: Dipl. Ing. K. Keller

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmitteln benutzt habe.

Ort, Datum Unterschrift

Ich möchte mich bei Herrn Prof. Dr. –Ing. Faulhaber bedanken, dass er mir die Möglichkeit gegeben hat meine Bachelorarbeit im Institut für Regelungs- und Prozessleittechnik zu bearbeiten. Besonderer Dank gilt Herrn Dipl. Ing. K. Keller und Herrn Manuel Joos für die tatkräftige Unterstützung.

Inhaltverzeichnis

1. Einleitung

Im Institut für Regelungs- und Prozessleittechnik der Hochschule Mannheim wurde im Jahr 2006 von Frau Kremena Donczewa eine Studienarbeit zur Verbesserung einer bestehenden Zündsteuereinheit eines Kraftfahrzeugmotors auf Basis eines PIC 16f84A geschrieben, diese ist aber veraltet und arbeitet ungenau.

Über die Jahre hat sich die Brennmethode für Mikrocontroller drastisch verändert. Statt dem früher üblichen Standard einer seriellen Schnittstelle (COM-Port) dient heute die USB-Verbindung als Schnittstelle zur Datenübertragung zwischen PC und µController.

Auch die Ausgangs- sowie die Eingangsschaltung der Platine muss überarbeitet werden, denn die vorhandenen Bauteile genügen nicht den elektrischen Leistungen. Es fließen höhere Ströme und es liegen viel höhere Spannungen an der Schaltung an.

Zusätzlich wurde die Computerbediensoftware und die Mikrocontrollersoftware von der Französischen Firma Alpine Renault – Les Trucs & Astuces, die auch Frau Donczewa in Ihrer Studienarbeit verwendet hat, nur in Französisch dokumentiert und ließ sich nicht auf Deutsch oder Englisch übersetzen.

Dies führt zu einer ungenügenden Bedienbarkeit und dazu, dass die Hard- und Software nochmals vollständig neu entwickelt werden muss.

2. Aufgabenstellung

Die mechanische Zündzeitpunktverstellung der Zündanlage eines Verbrennungsmotors soll durch ein elektronisches Zündsteuergerät realisiert werden.

Das Zündsteuergerät soll zwischen den Unterbrecherkontakt und die Zündspule geschaltet werden, um den konventionellen mechanischen Zündvorgang elektronisch zu realisieren.

Das zu entwickelnde Gerät soll mit Hilfe eines modernen Mikrokontrollers umgesetzt werden. Als Hardware Grundlage dient die von Frau Donczewa entwickelte Zündsteuereinheit auf Basis des PIC 16f84A.Die Mikrocontrollersoftware soll auf der von der Firma Alpine Renault – Les Trucs & Astuces entwickelten Software, die auch von der Frau Donczewa in Ihrer Studienarbeit verwendet wurde, basieren.

Es soll zusätzlich möglich sein mit dem Mikrokontroller über eine USB-Schnittstelle mit dem Desktop PC zu kommunizieren.

Der Benutzer soll die Möglichkeit haben über eine PC-Software das Zündungsverhalten zwischen 500 und 8000 Umdrehungen pro Minute des Motors selbst zu bestimmen. Als Grundlage soll die Computerbediensoftware von der Firma Alpine Renault – Les Trucs & Astuces dienen.

Anschließend soll das neue definierte Zündverhalten durch die oben erwähnte USB-Schnittstelle übertragen und auf dem Mikrocontroller gespeichert werden.

3. Funktionsprinzip der Zündanlage

Die Rotationsbewegung der Nockenwellen[1] eines Ottomotors spielt für die Zündanlage eine entscheidende Rolle. Durch sie werden das Einspritzen, das Zünden und die Abgasabfuhr im Zylinder gesteuert. Der von der Nockenwelle mechanisch betriebene Zündverteiler steuert den Unterbrecherkontakt, welcher wiederum das Zündverhalten der Zündspule beeinflusst. Zusätzlich gibt der Zündverteiler den von der Zündspule kommenden Hochspannungs-Zündimpuls an die jeweilige Zündkerze weiter. Diese zündet dann das Kraftstoff-Luft-Gemisch im Zylinder, wodurch die Kurbelwelle in Rotation versetzt wird und gleichzeitig auch die Nockenwelle in Bewegung versetzt.
Die nächste Abbildung zeigt schematisch den beschriebenen Ablauf der Zündanlage.

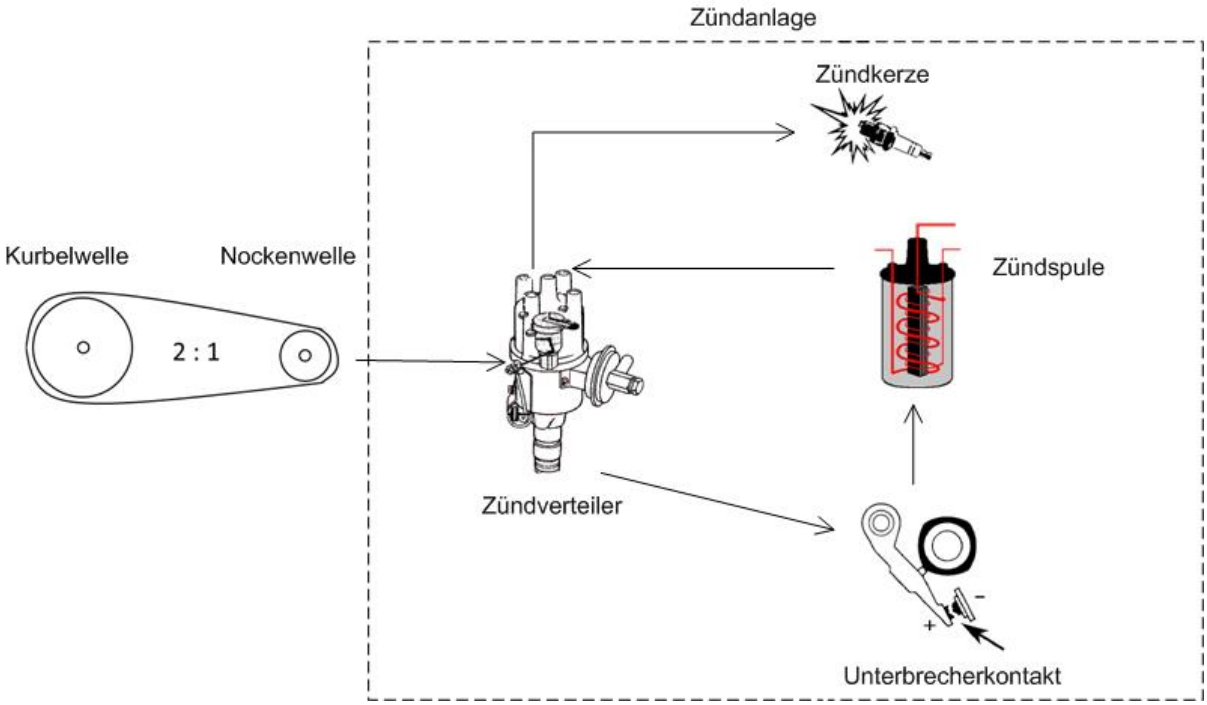


Abbildung 1 Schematischer Verlauf des Zündvorgangs

[1] Eine Nockenwelle ist ein Maschinenelement in Form eines Stabes auf dem mindestens ein gerundeter Vorsprung angebracht ist. Die Welle dreht sich um die eigene Achse, durch den oder die auf ihr angebrachten Nocken wird diese Drehbewegung wiederholt in eine kurze Längsbewegung umgewandelt. Die Nockenwelle wird in Nockenschaltern und hauptsächlich jedoch in Verbrennungsmotoren eingesetzt.

Bei einer konventionellen Zündanlage (siehe Abbildung 2 Konventionelle Zündanlage) wird durch das Schließen des Unterbrecherkontakts (siehe Kapitel 3.2) ein Magnetfeld in der Zündspule (siehe Kapitel 3.3) erzeugt. Der Zündkontakt unterbricht den Stromkreis, wodurch das Magnetfeld in der Zündspule zusammenbricht und eine große Spannung induziert. Diese Spannung wird über den Zündverteiler in den entsprechenden Zylinder auf die anzusteuernde Zündkerze übertragen, welche dann an ihren Elektroden einen Zündfunken erzeugt.

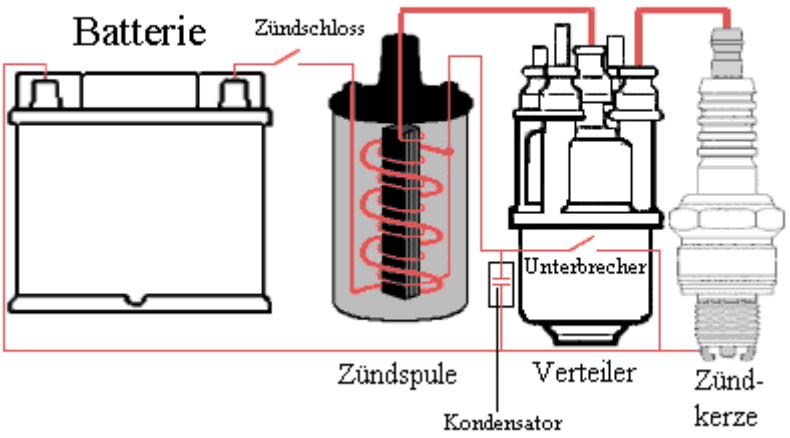


Abbildung 2 Konventionelle Zündanlage

3.1.Zündverteiler

Der Zündverteiler ist eine Baugruppe der Zündanlage eines Ottomotors, der zwei Funktionen erfüllt:

- Die Verteilung des Zündimpulses an die entsprechende Zündkerze
- Das optimierte zeitliche Schließen bzw. Öffnen des Stromkreises (Unterbrecherkontakt) zur Ansteuerung der Zündspule

Auf der Innenseite der Zündverteilerkappe befinden sich Kontaktflächen, welche mit den einzelnen Zündkerzen verbunden sind. Unter der Zündverteilerkappe rotiert ein Verteilerläufer, welcher über eine Schleifkohle den Zündimpuls an die Kontaktflächen bzw. an die Zündkerze weiter gibt. Dadurch werden die in der Zündspule erzeugten Hochspannungen in die entsprechende Zündkerze geleitet.

Die mechanische Zündzeitpunkt-Vorverstellung ist für den Ottomotor enorm wichtig. Sie sorgt dafür, dass der Motor immer zum richtigen Zeitpunkt das Kraftstoff-Luft-Gemisch entzündet. Der optimale Zündzeitpunkt hängt von der Drehzahl und der Last ab. Zum einen muss ein zündfähiges Gemisch an der Zündkerze vorhanden sein und zum anderen ist die Flammausbreitung zu beachten. Beide Aspekte müssen exakt getimte werden.

Da die Flammausbreitungsgeschwindigkeit von der Drehzahl unabhängig ist, muss der Zündzeitpunkt bei steigender Drehzahl nach Früh verlegt werden um gewährleisten zu können, dass die anschließende Druckausbreitung nach der eigentlichen Zündung in der Abwärtsbewegung des Kolbens erfolgt. Erfolgt das Abrennen und die Druckausbreitung in der Aufwärtsbewegung des Kolbens, kann es zu gravierenden Schäden des Motors kommen.

Die konventionelle Zündzeitpunkt-Vorverstellung wird durch zwei Verfahren realisiert. Zum einen wird durch einen Unterdruckversteller in einer Unterdruckdose eine Membran angesteuert (siehe Abbildung 3 Unterdruckversteller), welche über eine Zugstange die Kontaktplatte im Zündverteiler verstellt. Herrscht im Vergaser ein Unterdruck, so wird die Membran zurückgezogen, was wiederum die Zugstange bewegt und die Kontaktplatte verdreht. Dies hat zur Folge, dass eine Früh-Zündung erfolgt.

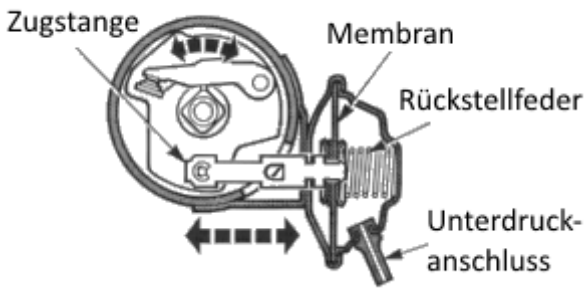


Abbildung 3 Unterdruckversteller

Zum anderen gibt es eine mechanische Zündzeitpunkt-Vorverstellung durch Fliehgewichte (siehe Abbildung 4 Fliehkraftversteller). Je höher die Drehzahl des Motors ist, umso weiter nach außen werden die Fliehgewichte im Zündverteiler gedrängt. Über einen Hebel-Mechanismus verstellt dies den Zündzeitpunkt. Es wird früher gezündet.

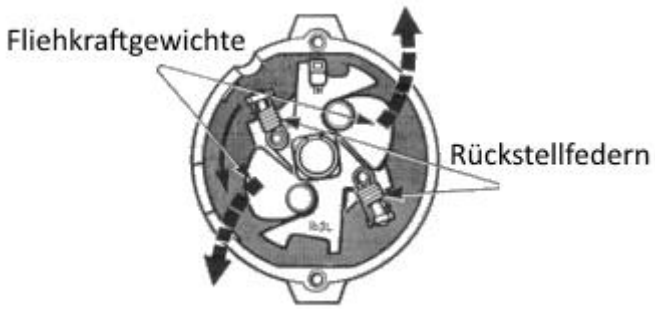


Abbildung 4 Fliehkraftversteller

3.2.Unterbrecherkontakt

Ein Unterbrecherkontakt, auch Unterbrecher, ist ein elektrischer Kontakt, der in regelmäßiger Abfolge einen Stromkreis öffnet bzw. schließt. Er stellt eine spezielle Bauform eines elektrischen Schalters dar. Es wird zwischen Selbstunterbrecherkontakten, welche sich im Aufbau wie bei einem Oszillator laufend selbst unterbrechen, und Unterbrecherkontakten, welche durch äußere Einflüsse gesteuert werden, unterschieden.

In vielen Anwendungsbereichen sind Unterbrecherkontakte in ihrer Funktion durch Halbleiterschalter wie Schalttransistoren ersetzt, welche ohne mechanische Bewegungen Stromkreise schließen und unterbrechen können. Mit diesen Halbleiterschaltern sind höhere Schaltfrequenzen zu erreichen und im Vergleich zu Unterbrecherkontakten weisen sie deutlich geringere Verschleißerscheinungen auf.

Bei Unterbrecherkontakten als elektromechanisches Bauelement kommt es durch das laufende Ein- und Ausschalten zu einem Kontaktabbrand, welcher durch den Schaltlichtbogen verursacht wird.

Für die zu entwickelnde Zündsteuereinheit besitzt er eine wichtige Aufgabe. Er liefert dem µC (siehe Kapitel 4.1.2) eine drehzahlabhängige Impulsfolge, welche für die Berechnung der Zündzeitpunkt-Vorverstellung erforderlich ist.

3.3.Zündspule

Die Zündspule ist ein Bauteil der Zündanlage eines Ottomotors oder einer Gasfeuerungsanlage. Der Aufbau entspricht einem Transformator, im Speziellen einem Spartransformator.

Die Zündspule arbeitet wie ein Funkeninduktor. Bei eingeschalteter Zündung wird die Primärwicklung der Zündspule von Strom durchflossen, wodurch sich ein Magnetfeld um die Spule bildet. Dieses Magnetfeld wird durch den gemeinsamen Eisenkern beider Wicklungen auch auf die Sekundärwicklung übertragen.

Das Öffnen des Unterbrechers im Primärkreis der Zündspule induziert im Sekundärkreis einen Hochspannungsimpuls, da das Magnetfeld rasch zusammenbricht. Die Hochspannung gelangt durch das Zündkabel zur Funkenstrecke einer Zündkerze, um zum Beispiel das Kraftstoff-Luft-Gemisch im Zylinder eines Ottomotors zum richtigen Zeitpunkt zu entzünden.

Sie dient beim Ottomotor dazu, aus der bordeigenen 12V-Spannung eine Hochspannung von ca. 15.000 bis 30.000 V zu erzeugen.

Bei der entwickelten Schaltung liefert der µC im Zusammenspiel mit der Ausgangsbeschaltung anstelle des Zündverteilers bzw. des Unterbrecherkontaktes den nötigen Zündimpuls an die Zündspule (siehe Kapitel 4.1.3).

3.4.Überarbeitung der Zündanlage

In der überarbeiteten Version der Zündanlage wird die mechanische drehzahlabhängige Zündzeitpunkt-Vorverstellung blockiert. Dies betrifft die Vorverstellung über die Fliehgewichte, wie auch die unterdruckabhängige Verstellung über die Membran.
Statt der mechanischen Zündspulenansteuerung erfolgt die Ansteuerung elektronisch mit Hilfe eines Mikrocontrollers und der dazu passenden Beschaltung.

In Abbildung 5 wird die Zündanlage mit dem Zündsteuergerät gezeigt. Dieses wird zwischen den Unterbrecherkontakt und die Zündspule geschaltet, um den konventionellen mechanischen Zündvorgang elektronisch zu realisieren. Dabei werden die vom Unterbrecherkontakt ausgesendeten Impulse im Mikrocontroller verarbeitet und eine Zündzeitpunkts-Vorverstellung je nach Drehzahl neu berechnet. Der Zündimpuls wird dann zu dem neu berechneten Zeitpunkt auf die Zündspule gegeben.

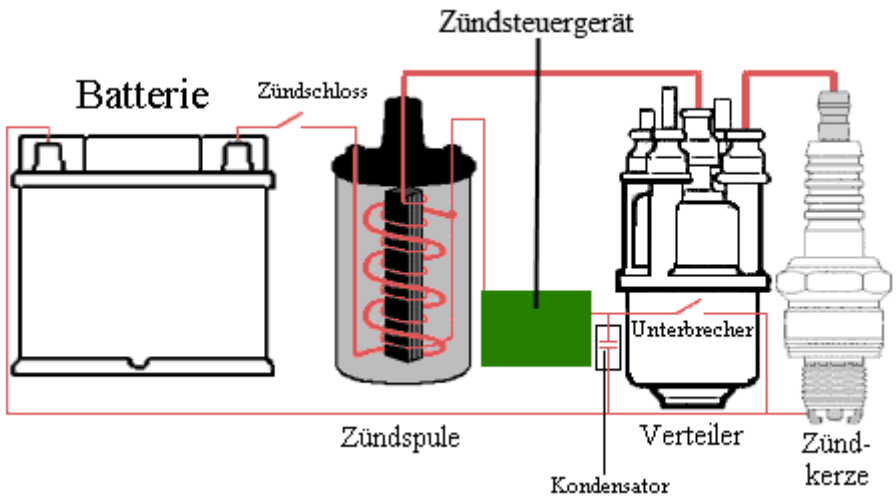


Abbildung 5 Überarbeitete Zündanlage

4. Umsetzung

Die Umsetzung der gestellten Aufgabe erfolgte in 4 Schritten.

1. Entwicklung eines Platinen-Layouts für die gestellte Aufgabe
2. Entwicklung eines Mikrocontrollerprogramms in der Programmiersprache C
3. Entwicklung einer PC-Bediensoftware
4. Erstellung der USB-Schnittstelle und Erweiterung des µController Programmcodes für eine Kommunikation zwischen PC und µC

Die von der Firma Alpine Renault – Les Trucs & Astuces entwickelte Mikrocontrollersoftware und von Frau Donczewa entwickelte Hardware ist nicht vollständig kompatibel mit dem neu gewählten Mikrocontroller (PIC18F2550), deswegen muss eine vollständig neue Software, sowie eine vollständig neue auf den µC zugeschnittene Hardware (Beschaltung, Versorgung) entwickelt werden. Die genaue Problematik der veralteten Software im Vergleich zur neu entwickelnden Software wird in Kapitel 4.1.5 näher erläutert.

Die von der oben erwähnten Firma entstandene PC-Bediensoftware ist ein geschlossenes, gekapseltes und nicht bearbeitbares Softwareinstallationspaket. Die dazu in französischer Sprache geschriebenen Dokumentationen und das geschlossene Softwarepaket konnten nicht als Grundlagen genutzt werden, deshalb wurde die PC-Bediensoftware ebenfalls vollständig neu entwickelt.

4.1.Hardware

Die im Anhang 2 (Neu entwickelter Schaltplan) abgebildete Schaltung basiert auf der von Frau Donczewa (siehe Anhang 1 Schaltplan von Frau Donczewa) entwickelten Schaltung. Es wurden Änderungen im Eingangs-, Ausgangs- sowie im Mikrocontroller-Modul vorgenommen. Lediglich das Versorgungs-Modul des Mikrocontrollers ist unter Berücksichtigung geringfügiger Änderungen gleich geblieben.

Bei der Entwicklung der neuen Platine müssen die hohen Spannungen und Ströme beachtet werden, die bei der Zündung der Zündkerze erzeugt werden. Sie können bei unzureichender Schutzbeschaltung zu einem Ausfall der gesamten Zündsteuereinheit führen. Deswegen werden die Module (Spannungsversorgung, Mikrocontroller-Modul, Eingangsschaltung, Ausgangsschaltung) um die notwendigen Schutzmaßnahmen ergänzt, welche nun im Folgenden erklärt werden.

Der Massenbezug wird in zwei getrennte Bereiche (Digital und Analog) aufgeteilt. Die Zündspule arbeitet im Hochspannungsbereich, wohingegen der Mikrokontroller mit 5 V und die Ausgangbeschaltung mit 12 V versorgt werden. Um den µC galvanisch vom Rest der Platine zu trennen (Trennung des Digitalteils vom Analogteil), werden je ein Optokoppler in die Eingangs- bzw. Ausgangsschaltung eingebaut. Somit kann auch verhindert werden, dass Hochspannungsanteile der Zündspule den µC negativ beeinflussen (siehe Anhang 2 Neu entwickelter Schaltplan). Das Prinzip und die Vorgehensweise werden in Kapitel 4.1.3 und 4.1.4 näher erläutert.

Durch den Einbau von Z-Dioden werden die Spannungsspitzen begrenzt. Der Massenbezug des Digitalteils wird als Massenfläche ausgelegt. Weiterhin werden Stützkondensatoren zur störungsunterdrückung eingesetzt.

Die Schaltung ist in 4 Bereichen untergliedert:

- Der Versorgungsspannungsmodul
- Der Eingangsmodul
- Der Ausgangsmodul
- Der µC Modul

Der Versorgungsspannungsmodul hat die Aufgabe zur Wandel von den bekommen 12 V Spannung auf 5V,die der µC für seine Versorgung nutzt.

Der Eingangsmodul bekommt von den Unterbrecherkontakt bei einer Nockenwellenumdrehung des Motors je 4 Zündimpulse. Die werden dann zur weiteren Verarbeitung an den µC gesendet.

Die vom µC umgesetzten Zündimpulse werden mit einen optimalen für den Zündfall vorgesehenen Timing an die Ausgangschaltung gesendet, um die damit verbundene Zündspule zu Aktivieren.

Der µC Modul hat die Aufgabe alle nötigen Informationen dem µC zu liefern. Sowie die Möglichkeit zu geben dem Software Entwickler einen neuen Code zu brennen

4.1.1. Versorgungsspannungsschaltung

Die Schaltung wandelt mit Hilfe eines Spannungsreglers [2]7805T die eingehende 12 V Spannung auf 5 V. Die 5 V werden als Versorgungsspannung für den µC benötigt.

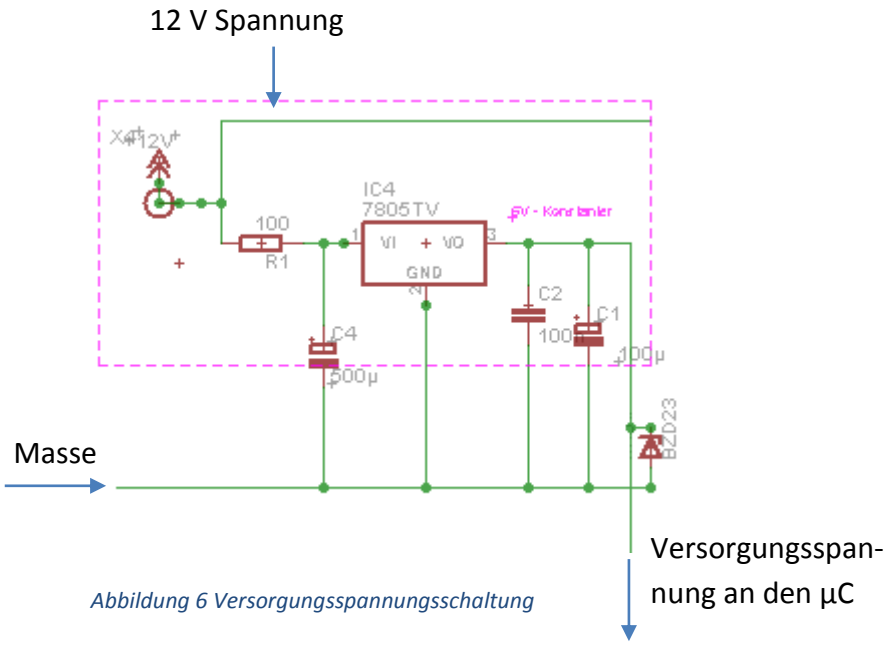


Abbildung 6 Versorgungsspannungsschaltung

Damit der µC nicht von möglichen Spannungsspitzen zerstört wird, bewacht die 5 V Z-Diode (BZD23) die µC Versorgungsspannung.

Der C2, C1 und C 4 Kondensatoren werden für die Filterung der möglichen Spannungsschwankungen eingebaut

[2] Spannungsregler sind elektronische Schaltungen, die elektrische Spannungen stabilisieren und die Schwankungen von Batterie- oder Netzspannung in weiten Grenzen ausgleichen können. Es können sowohl Gleich- als auch Wechselspannungen stabilisiert werden. Bei kleinen Gleichspannungen wird die gesamte Schaltung oft in einem Bauteil platzsparend vereinigt.

4.1.2. Eingangsschaltung

Die Eingangsschaltung bekommt bei einer Nockenwellenumdrehung des Motors je 4 Rechteckimpulse vom Unterbrecherkontakt. Die Mechanik öffnet und schließt den Unterbrecherkontakt, wodurch High-[3] und Low-Pegel generiert werden, die dann wie Rechtecke aussehen.

Störungen werden mit Hilfe eines RC-Tiefpasses geglättet. Der RC-Eingangs-Tiefpass wird durch den Widerstand R16 und den Kondensator C7 (siehe Abbildung 7 Eingangsbeschaltung) realisiert. Die Zeitkonstante beträgt bei 1 kΩ und 100 nF 100µs bei einer Grenzfrequenz von 1,6 kHz. Wiederum bilden der Widerstand R17 und der Kondensator C12 einen Ausgangstiefpass. Da die beiden Tiefpässe baugleich sind, besitzen sie auch die gleichen Zeitkonstanten und Grenzfrequenzen.

Anschließend wird das geglättete Signal an den Optokoppler (OK 1)gesendet.

Der Impuls vor dem OK1 hat als High 12 V und als Low 0V,danach liegt High bei 5 Volt und Low bei 0 V.

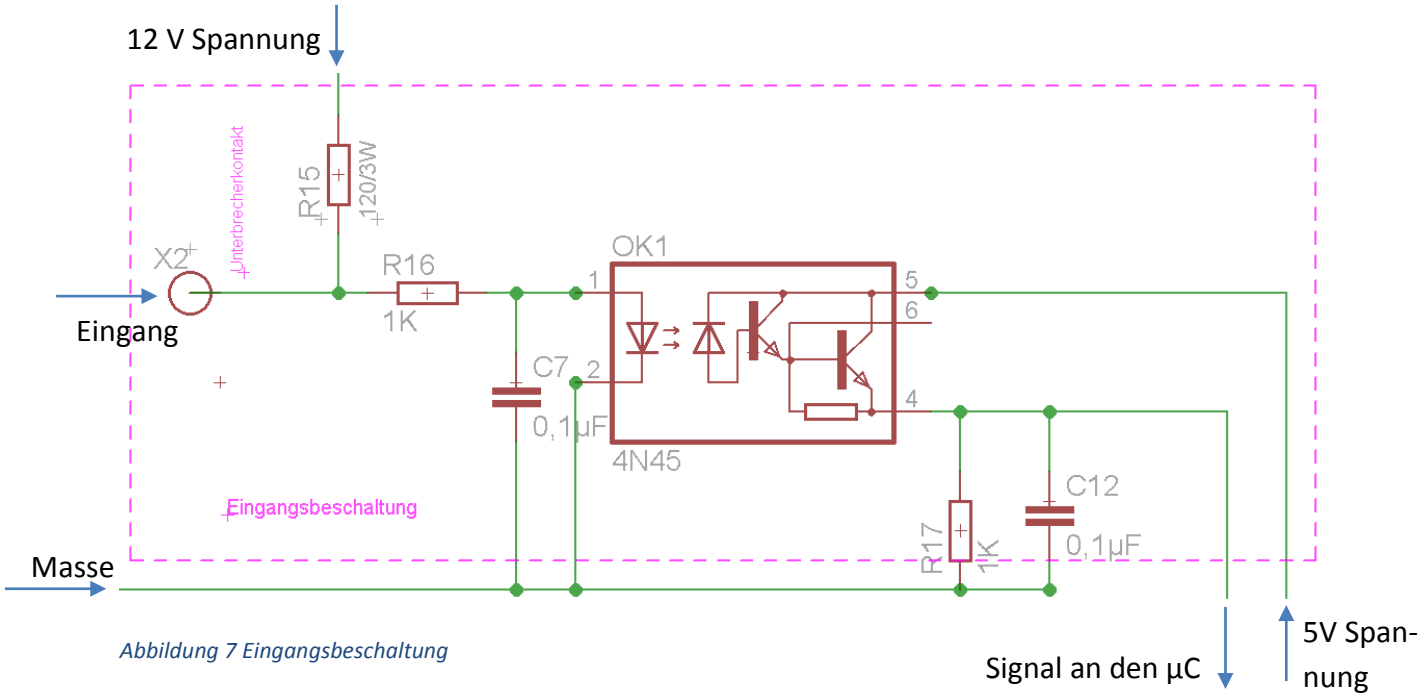


Abbildung 7 Eingangsbeschaltung

Nach der optischen Übertragung wird der empfangene Rechteckimpuls weiter an den µC-Eingang gesendet (PortC Pin 3), um dort verarbeitet zu werden.

[3]Ein Logikpegel bezeichnen in der Digitaltechnik die meist zur Repräsentation der Logikwerte verwendeten elektrischen Spannungspegel. Bei digitalen, üblicherweise binär codierten Signalen sind zwei Spannungsbereiche erlaubt, die High-Pegel (auch H-Pegel, High, H) bzw. Low-Pegel (L-Pegel, Low, L) genannt werden.

4.1.2.1. Optokoppler

Ein Optokoppler ist ein Bauelement der Optoelektronik und dient zur Übertragung eines Signals zwischen zwei galvanisch getrennten Stromkreisen.

Er besteht aus einem optischen Sender, typischerweise ist dies eine Leuchtdiode (LED) und einem optischen Empfänger wie einem Fototransistor, welche beide in einem lichtundurchlässigen Gehäuse untergebracht sind.

Mit Optokopplern können sowohl digitale als auch analoge Signale übertragen werden.

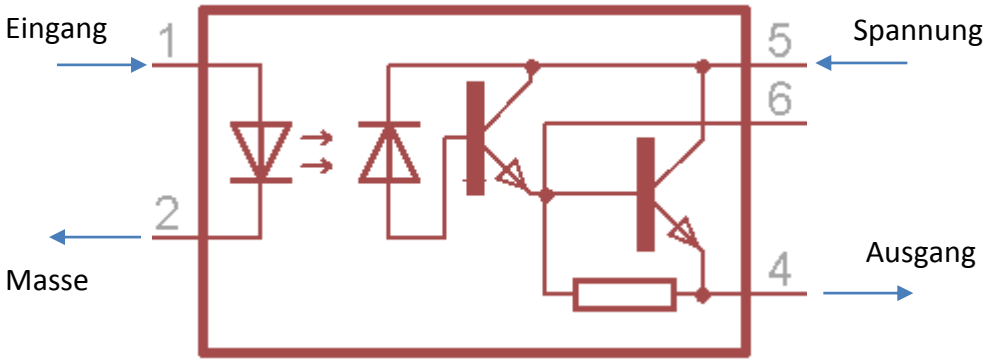


Abbildung 8 Optokoppler

Auf der Abbildung 8 Optokoppler wird das eingehende Signal beim Eingang durch die Leuchtdiode optisch übertragen damit zwei galvanisch getrennte Stromkreise keinen gemeinsamen Bezugspunkt bekommen.

4.1.3. Ausgangsschaltung

Der vom µC gesendete Zündimpuls wird zunächst durch den Optokoppler(OK2) galvanisch vom Rest der Schaltung abgekoppelt. Der Impuls, der als High 5V und als Low 0V beträgt, öffnet die Transistoren der Ausgangsschaltung und die Power MOSFETs heben den Spannungsbereich auf 12V an.

Das wiederum bereitet die Zündspule vor um das Kraftstoff/Luft-Gemisch im Zylinder des Ottomotors zum richtigen Zeitpunkt zu entzünden.

Die Kondensatoren C8 und C6 arbeiten als Stützkondensator. Auf den Ausgangs-Signalleitungen vermindern sie sowohl „Überschwinger“ als auch „Unterschwinger“ der Sig-

nalpegel und damit die Möglichkeit von Störungen in der Signalverarbeitung. Umgekehrt können Kondensatoren störende, zeitlich begrenzte Überspannungen in elektronischen Schaltungen aufnehmen und so ihre Ausbreitung und schädliche Wirkung vermindern.

Die Z-Diode D15V hat die Aufgabe die Schaltung vor Spannungsspitzen über 15 V zu schützen. Die Z-Diode D1 sperrt für Spannungen über 440 V die möglichen Rückspannungen, die von der Zündspule kommen. Die Kondensatoren, Dioden und Optokoppler sind nötig, weil es ohne diese zu Rückspannungen von der Zündspule bei jedem Zünden kommt und sie schädlich für die Mikrocontrollerbeschaltung sind.

Ein gravierender Effekt liegt darin, dass der Massenbezug von 0V auf 3V verrutscht und somit der Spannungsbezug für den µController nicht mehr gewährleistet ist und er falsch oder gar nicht mehr sauber arbeitet. Somit sind diese Bauteile essenziell wichtig, damit die Zündsteuereinheit jeder Zeit einsatzfähig bleibt.

Vor dem Transistor T4 befindet sich ein Tiefpass, der die Aufgabe besitzt mögliche Störungen zu filtern. Der T4 Transistor Aktiviert selbst den T5 Transistor der die Power MOSFET haben mit der Bezeichnung IRFPC40 auf 12 V hebt.

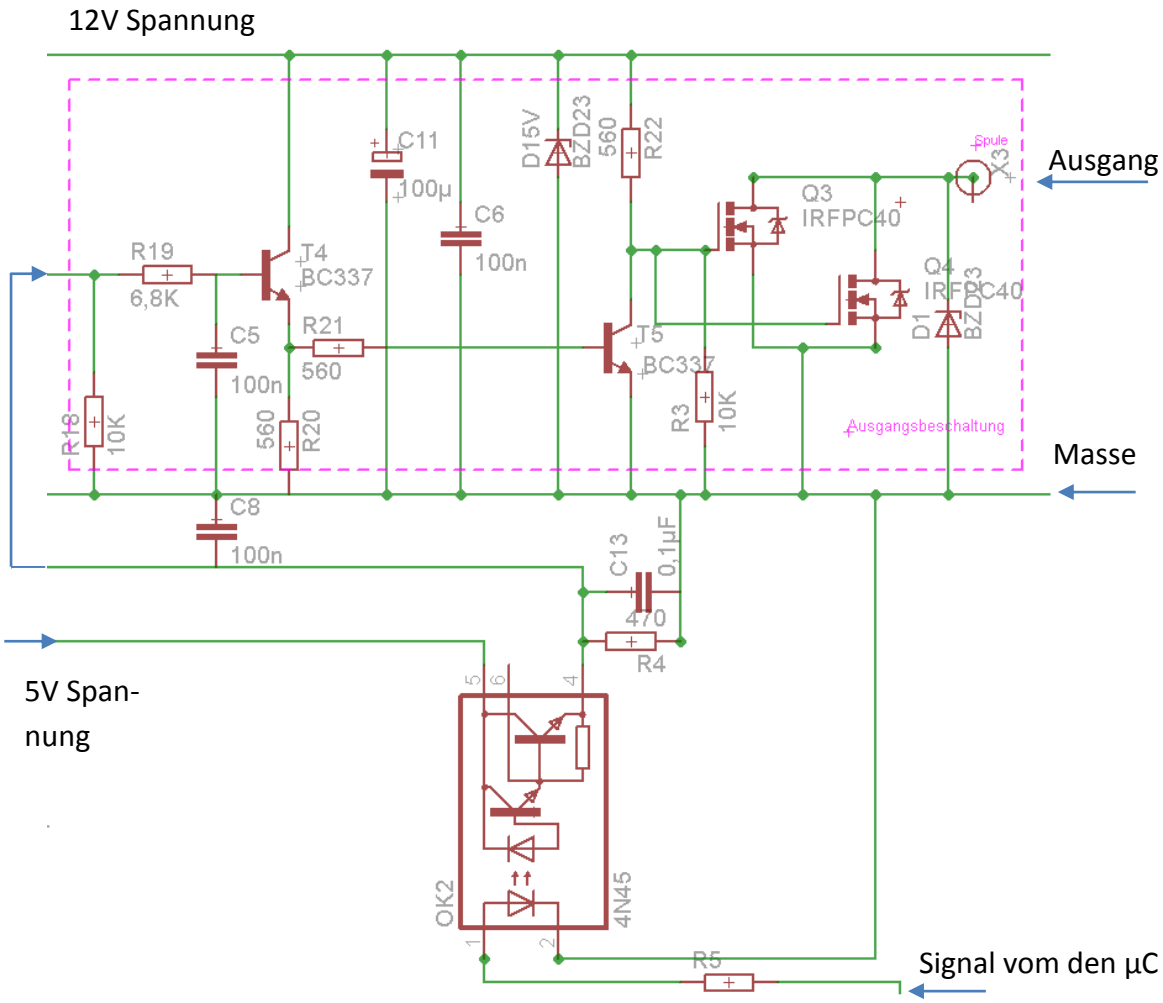


Abbildung 9 Ausgangsbeschaltung

4.1.3.1. Power MOSFET

Ein Leistungs-MOSFET ist eine spezielle Version eines Metall-Oxid-Halbleiter-Feldeffekttransistors (MOSFET), der für das Leiten und Sperren von großen elektrischen Strömen und Spannungen optimiert ist.

Leistungs-MOSFETs unterscheiden sich von bipolaren Leistungstransistoren sowohl in der Funktionsweise als auch in der Effizienz. Einige Vorteile von Leistungs-MOSFETs sind die schnelle Schaltzeit, kein zweiter Durchbruch und stabile Verstärkungs- und Antwortzeiten. Ab einer Strombelastung von etwa 1A wird ein MOSFET den Leistungs-MOSFETs zugeordnet.

4.1.4. Boardlayout

Jede Leitung (Leiterbahn) einer Leiterplatte weist eine gewisse Impedanz auf. Fließt nun über eine dieser Leitungen Strom, verursacht dieser zwangsläufig einen Spannungsabfall. In den meisten Fällen wird ein Stromkreis über eine Masseleitung geschlossen, wodurch auch an dieser Verbindung ein Spannungsabfall entsteht.

Die Masseverbindung hat somit an jedem Punkt ein anderes Potenzial. Da integrierte Schaltkreise auf der Leiterplatte, wie beispielsweise Messverstärker oder Analog-Digital-Umsetzer, die Spannung ihrer Ein- oder Ausgänge auf das Massepotenzial beziehen, tritt somit eine Verfälschung dieser Signale auf, da das Bezugspotenzial – die Masseleitung – nicht für jeden Schaltkreis dasselbe ist. Um dem entgegenzuwirken kann die Masseleitung als Fläche ausgeführt werden.

In manchen Fällen kann es sinnvoll sein, nicht eine gemeinsame Massefläche, sondern mehrere separate Massefläche zu verwenden und diese an einer bestimmten Stelle zusammenzuführen. Werden auf einer Leiterplatte unterschiedliche Komponenten (analoge-, digitale- und Leistungsbauteile) eingesetzt, so kann die Masse auf zusammengehörende Komponenten aufgeteilt werden.

Werden nun beispielsweise drei unterschiedliche Masseflächen (Analogmasse, Digitalmasse und Versorgungsmasse) eingesetzt, treten durch höheren Strom zwar nach wie vor Potenzialdifferenzen entlang der Masse auf, jedoch beeinflussen Komponenten, die hohe Ströme fordern, nicht mehr die gesamte Masse, sondern nur mehr die eigene. Wenn ein Schaltregler auf der Leiterplatte durch hohe hochfrequente Ströme nun Störungen auf der Versorgungsmasse verursacht, sind diese auf der getrennten Analogmasse nicht merkbar.

Der Analogteil der Leiterplatte hat somit mit der getrennten Analogmasse ein störungsfreies Bezugspotenzial. Dasselbe gilt für den Digitalteil, der durch eine eigene Digitalmasse nicht vom Schaltregler gestört werden kann und selbst auch nicht den Analogteil stören kann.

Das Boardlayout wurde so konzipiert, dass der 5V-Bereich und 12V-Bereich getrennte Massen besitzen. Nur in einem kleinen Punkt laufen die beiden Grundpotentiale zusammen, damit ein gemeinsames Bezugspotential besteht(siehe Anhang 3 Board Layout).

Andersfalls würde die von der Zündspule generierte Hochspannung den Mikrocontroller außer Gefecht setzen. Dies würde dann zu einem instabilen Zündungsverlauf führen. Wie schon im Kapitel 4.1.3 erwähnt wurde.

4.1.5. Microcontroller

4.1.5.1. µC-Schaltung

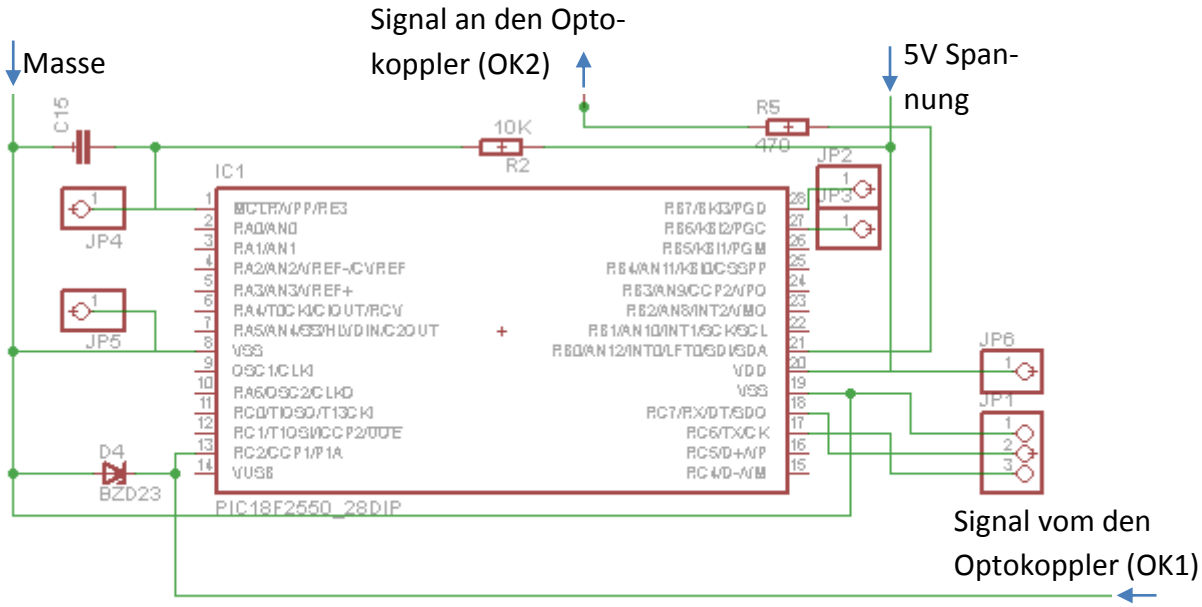


Abbildung 10 µC-Schaltung

Die Beschaltung ermöglicht dem PIC18F2550 alle nötigen, für sein Betrieb, Informationen zu bekommen. Die µC-Software wird, durch die eingebaute Zugriffstelle JP1 bis JP6, mit Hilfe eines Programmiergeräts (PICKit 3) gebrannt. Zum Schutz vor den eingehenden Signalen wird die D4 Z-Diode eingefügt.

4.1.5.2. PIC 18F2550

Mit der 18F-Microcontrollerfamilie versucht Microchip Technology Controller anzubieten, die deutlich leistungsstärker sind als die 16F- Microcontrollerfamilie.

Aus diesem Grund wird der µC PIC18F2550 für die Umsetzung der Aufgabenstellung gewählt. Weitere Vorteile sind unter anderen eine ausreichende Anzahl an I/O Ports besitzt und ein für die Aufgabenstellung genügen verfügbaren Speicher. Der Speicher wird für die dort abgelegten Daten und den dafür vorgesehenen Entwickelten Code verwendet. Ein weiterer Vorteil ist das vorhandene Capture und Compare Modul, welches für diese Umsetzung der Aufgabe wichtig ist.

Die interne Architektur der µC PIC18F2550 wurde so optimiert, dass der Entwickler mit einer verbesserten Performance in der Programmiersprache C entwickeln kann und deshalb ist es garnicht möglich oder in manchen Fällen nur bestimmte Teile des Assembler Codes vom der älteren PIC Microcontrollerfamilien zu benutzen.

Der Kontroller hat 28 PINs, die sich u.a. auf 3 Digitale I/O-Ports mit je 8 Pins gliedern. Jeder I/O-Pin kann bis zu 25 mA treiben. Die USB 2.0 Unterstützung ist vorhanden und kann in 2 verschiedenen Geschwindigkeitseinstellungen betrieben werden. Der erste nennt sich Low-Speed (1,5 Mb/s) und der Zweite Full-Speed (12 Mb/s). Dazu gibt es 1 Kbyte Dual Access RAM.

Die CPU wird mit 96 MHz getaktet und unterstützt externe Oszillatoren bis zu 48 MHz. Ein interner Oszillator kann bis zu 8 MHz schnell getaktet werden. Dazu wurden für den Entwickler 4 verschiedene Timer- und die UART- Verbindungen, die man asynchron konfigurieren und benutzen kann, vorbereitet. Die CCP/Compare/PWM-Module sind für die Timer 1 und 3 vorhanden.Für die Speicherung stehen 24Kbyte Flash Speicher, 2048 Bytes SRAM und 256 Bytes EEPROM zur Verfügung.

In der Abbildung 9 Pin Belegung des PIC18F2550 ist eine vereinfachte Darstellung der Pin-Belegung des PIC18F2550 zu sehen.

28-Pin PDIP, SOIC

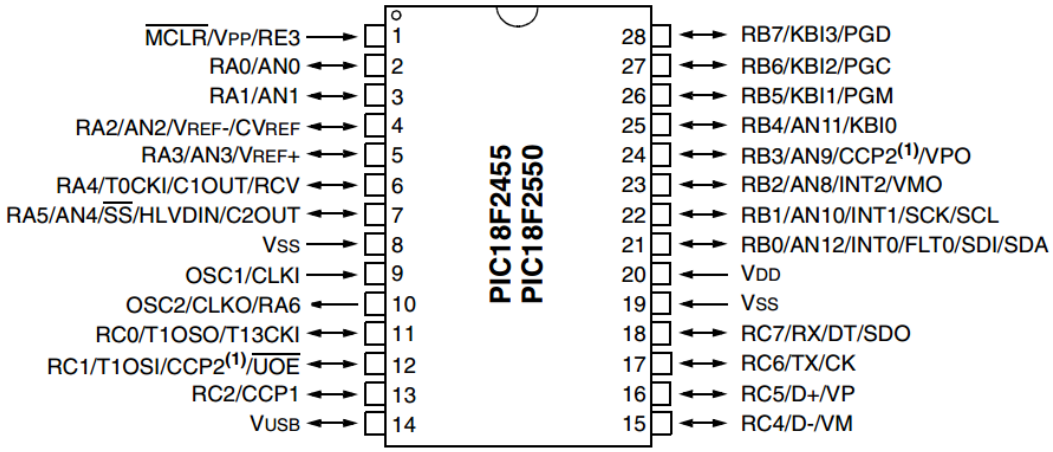


Abbildung 9 Pin Belegung des PIC18F2550

4.2.Mikrokontroller Software

4.2.1. Prinzipieller Ablauf der Mikrokontroller Software

Die Mikrocontroller Software funktioniert prinzipiell, wie im Anhang 4 Mess-Zünd Ablauf dargestellt ist, in 5 Schritten:

- Die beim µC eingehende Periode wird gemessen
- Der durch den Messvorgang resultierender Time1 Wert, wird in der dafür vorgesehener Speicher Struktur gespeichert
- Die gewichtete Periodenmittelwert-Berechnung wird aus 14 Takten ermittelt, damit bei dem aktuellen Zündfall, die Zugriffstelle für die Liste mit Verfrühten Zündzeitpunkten auslesen werden kann
- Aus der Lookup-Tabelle[4], eine Liste mit verfrühten Zündzeitpunkten, wird der gesuchte Verzögerungswert ausgelesen
- Und anschließend in dem Compare Modul, der für Zündungsvorgang zuständig ist, gespeichert, damit die verfrühte Zündung erfolgen kann

Eine Nockenwellenumdrehung des Motors entspricht 2 Umdrehungen der Kurbelwelle und erzeugt durch den Unterbrecherkontakt 4 Messperioden für den µC.

Das führt dazu, dass eine vom µC gemessene Periode in Wirklichkeit nur ein Viertel der Nockenwellenperiode beträgt.

Zum Beispiel beträgt die Nockenwellenumdrehungszahl 350 U/Min, so misst der µC 1400 Pulsen/Min. Daraus folgt, dass die Periodendauer einer Nockenwellenumdrehung ca. 0.17142 s beträgt, wohingegen am Mikrocontroller 4 Signalperioden von 0.042857s gemessen werden können. Nach jeder Periode wird ein Zylinder bzw. eine Kerze gezündet.

Durch die ständige Messung der eingegangenen Zündperiode (1/4 Nockenwellenperiode) kann der µC 4-mal schneller berechnen, in welchem Fall (Drehzahl) er sich gerade befindet.

Die Umrechnung von den eingehenden Periodenlängen auf die dafür passenden Zündzeitpunkte erfolgt nicht im µC selbst, sondern wird in der PC-Bedienungssoftware erstellt und mit Hilfe eines Übertagungskabels an dem µC gesendet und in einer Lookuptable gespeichert. Die errechneten Zahlen entsprechen den Timer 1 Werten(siehe dazu das Zahlenbeispiel unten und den Anhang 4 Mess-Zünd Ablauf).

[4] Lookup-Tabellen (LUT) bzw. Umsetzungstabellen werden in der Informatik und in der Digitaltechnik verwendet, um Informationen statisch zu definieren und diese zur Laufzeit des Programms - zur Vermeidung aufwändiger Berechnungen oder hohen Speicherverbrauchs - zu benutzen

Zahlenbeispiel:
Die Nockenwellenumdrehungszahl beträgt 350 U/Min. Dies entspricht 1400 Pulsen/min am µC und die Periodendauer beträgt dann 0.042857s. Dieser Periodendauer entspricht einem Timerwert von 65015, welcher im Timer-Messregister(CCPR1H, siehe Kapitel 4.2.7) abgelegt wird. Der µC ruft für diesen Timerwert den entsprechenden Wert aus der Lookup-Tabelle auf. Dieser neue Wert definiert die Zündperiode neu und verursacht damit ein verfrühtes Zünden der Zündkerze.

Ein Mess- und Zündbeispiel ist in Abbildung 11 Prinzipieller Mess- und Ausgabeverlauf abgebildet.

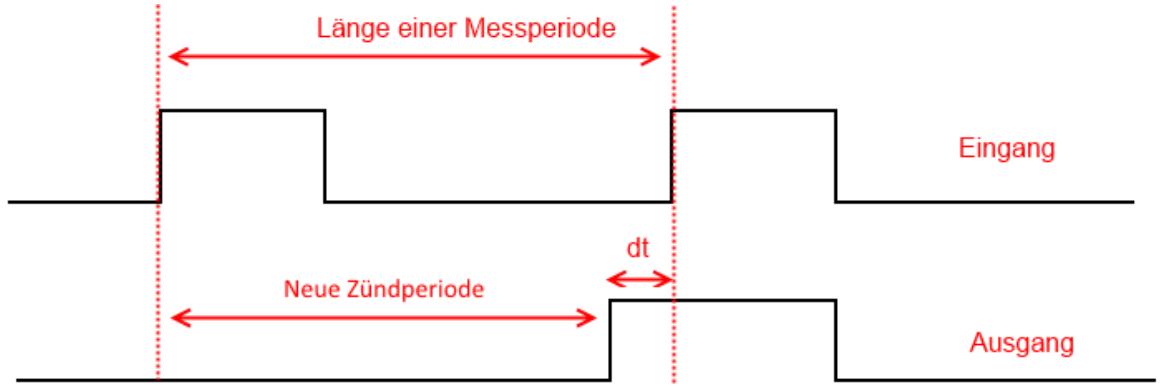


Abbildung 11 Prinzipieller Mess- und Ausgabeverlauf

Die Differenzzeit dt ist die Zeit, die für die frühere Zündung benötigt wird, um bei einer gewünschten Drehzahl, die in der Burn Software angegebenen wird, den gewünschten Verzögerungsgrad einzustellen.

4.2.2. Microchip MPLAB X IDE Entwicklungs-Software

Die Entwicklungs-Software basiert auf der älteren Version MPLAB IDE. Im Gegensatz zur alten Version wurde die neue Entwicklungsumgebung in der Programmiersprache JAVA entwickelt. Das MPLAB X beinhaltet viele Neuerungen im Gegensatz zur älteren Version, die direkt von Microchip Technology als Entwicklungs-Software angeboten und empfohlen wird.

Abbildung 12 MPLABX Struktur

In der neuen Softwareentwicklungsumgebung wird der XC8 C Compiler genutzt. Dieser Compiler bildet mit dem integrierten Texteditor eine schlüssige Entwicklungsbasis.

Die geschriebenen und kompilierten Programme können in der Umgebung simuliert oder mit Hilfe des PIC-Kits3 direkt auf einem PIC-Controller getestet werden.

4.2.3. PIC Kit 3

Dieses Werkzeug wurde von Microchip Technology entwickelt und ist für PIC Mikrocontroller geeignet. Die MPLAB X IDE unterstützt das Brenner und Debuggen direkt in der Software und aktualisiert entsprechend die Daten des Chips durch eine USB-Schnittstelle.

Abbildung 13 PICkit3

4.2.4. USB-UART Umsetzer TTL-232R-RPi

Das TTL[5]-232R-RPi Kabel bietet eine asynchrone serielle Datenübertragung und unterstützt Datenraten von 300 Bit / s bis 3 Mbit / s bei einem TTL-Pegel von 3,3 V. Das UART-Protokoll besteht aus 7 oder 8 Datenbits, 1 oder 2 Stoppbits und einer möglichen Parität.

Der interne Chip im Kabel übernimmt die bidirektionale Umsetzung eines USB-Signals in ein UART-Signal bzw. umgekehrt und beachtet dabei die entsprechenden Anforderungen der beiden Protokolle. Dies ermöglicht ein verbessertes Debuggen und zusätzlich eine weitere Möglichkeit den µController zu programmieren.

Das TTL-232R-RPi Debug-Kabel bietet eine USB to TTL serielle Schnittstelle für den Raspberry Pi und andere UART fähige Controller wie PIC18F2500.

Der Umsetzer verfügt über den On-Board FT232RQ Chip und ermöglicht somit die USB-UART Übertragung.

[5] Die Transistor-Transistor-Logik (TTL) ist eine Schaltungstechnik für logische Schaltungen (Gatter), bei der als aktives Bauelement der Schaltung planare npn-Bipolartransistoren verwendet werden. Meist wird am Eingang ein Multiemitter-Transistor eingesetzt, so dass für mehrere Eingänge nur ein Transistor erforderlich ist.

Abbildung 14 USB - UART Verbindung Kabel

Der Umsetzer besitzt einen internen EEPROM mit beschreibbarem Bereich. Die TTL-Eingänge sind für Spannungen über 5 V abgesichert.

Alles ist USB 2.0 Full Speed kompatibel und in einem Betriebstemperaturbereich von -40 ° C bis +85 ° C einsatzfähig. Die Kabellänge beträgt 1,0 m.

4.2.5. UML Mappe

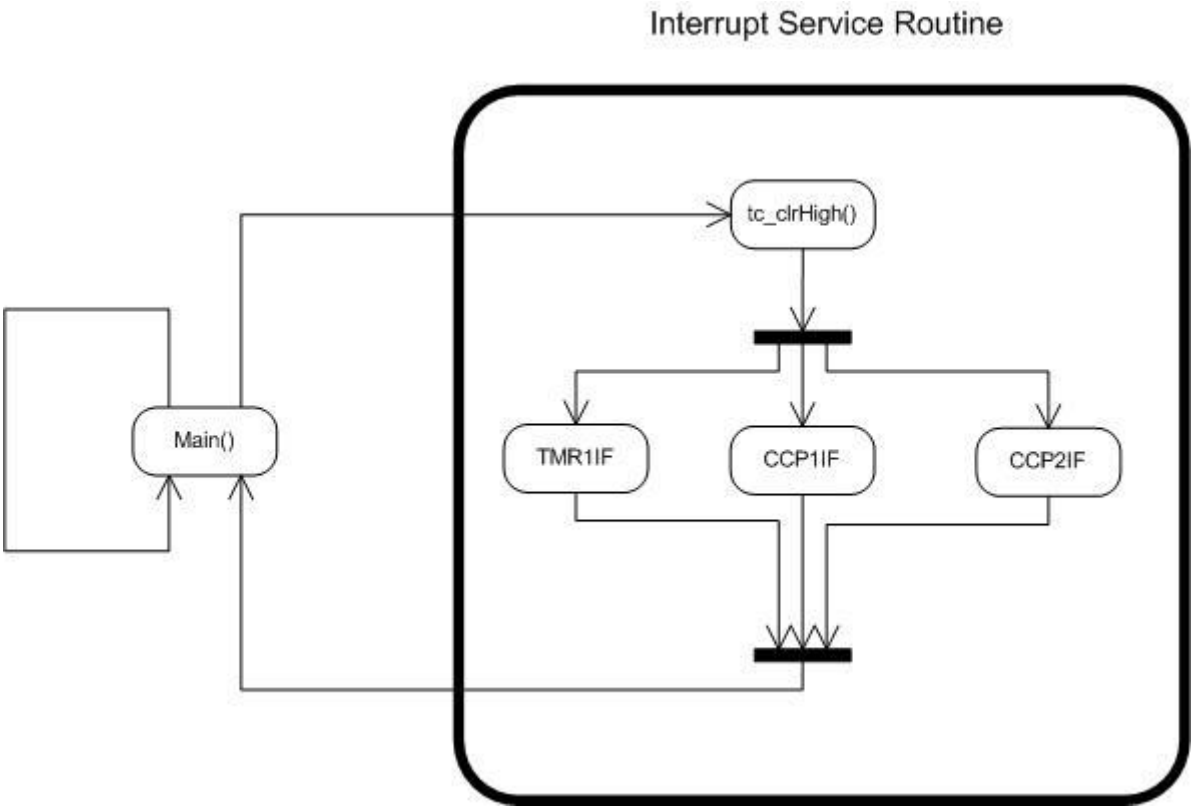


Abbildung 15 Die Interrupt Service Routine im µC

Die Umsetzung der zwei wichtigsten Aufgaben: das Messen der eingehenden Periodenlänge und die synchrone Ausgabe von Zündimpulsen, wird wie oben angezeigt größtenteils in der Interrupt Service Routine umgesetzt.

4.2.6. µC Konfiguration

Die I/O-Ports des PIC18F2550 werden wie folgt konfiguriert:

Außer Pin 4 werden alle Pins des PORTA auf Output gestellt.

TRISA = 0xBF;

Bei PORTB sind alle Ports auf Output gestellt, außer dem 0 Pin.

TRISB = 0xFE;

Bei PORTC sind alle Ports auf Output gestellt.

TRISC = 0xFF;

Die interne Taktung des Mikrocontrollers beträgt 8MHz, nachdem die folgende Einstellung getätigt wird:

```
OSCCONbits.IRCF2 = 1;

OSCCONbits.IRCF1 = 1;

OSCCONbits.IRCF0 = 1;
```

Die Konfiguration für Timer1 sieht folgendermaßen aus:

T1CON = 0b00111101;

Bit7 = 0;

Das Lesen und Schreiben wird im 8-Bit-Modus ausgeführt.

Bit6 = 0;

Die Quelle für den Timer wird von außen geliefert.

Bit5-4 = 11;

Timer1 Prescaler beträgt 1:8

Bit3 = 1;

Timer1 Oszillator wird aktiviert

Bit2 = 1;

Timer1 soll sich nicht mit sich selbst synchronisieren

Bit1 = 0;

Der Interne Clock wird gewählt

Bit0 = 1;

Startet den Zählvorgang.

Die Konfiguration des Timer3 sieht wie folgt aus:

T3CON = 0b00110101;

Bit7 = 0;

Das Lesen und Schreiben wird im 8-Bit-Modus durchgeführt.

Bit6-3 = 0;

Die Quelle für beide CCP-Module ist Timer1.

Bit5-4 = 11;

Timer3 Prescaler beträgt 1:8

Bit2 = 1;

Timer3 soll sich nicht mit sich selbst synchronisieren

Bit1 = 0;

Der Interne Clock wird gewählt

Bit0 = 1;

Startet den Zählvorgang.

4.2.7. Capture Modul

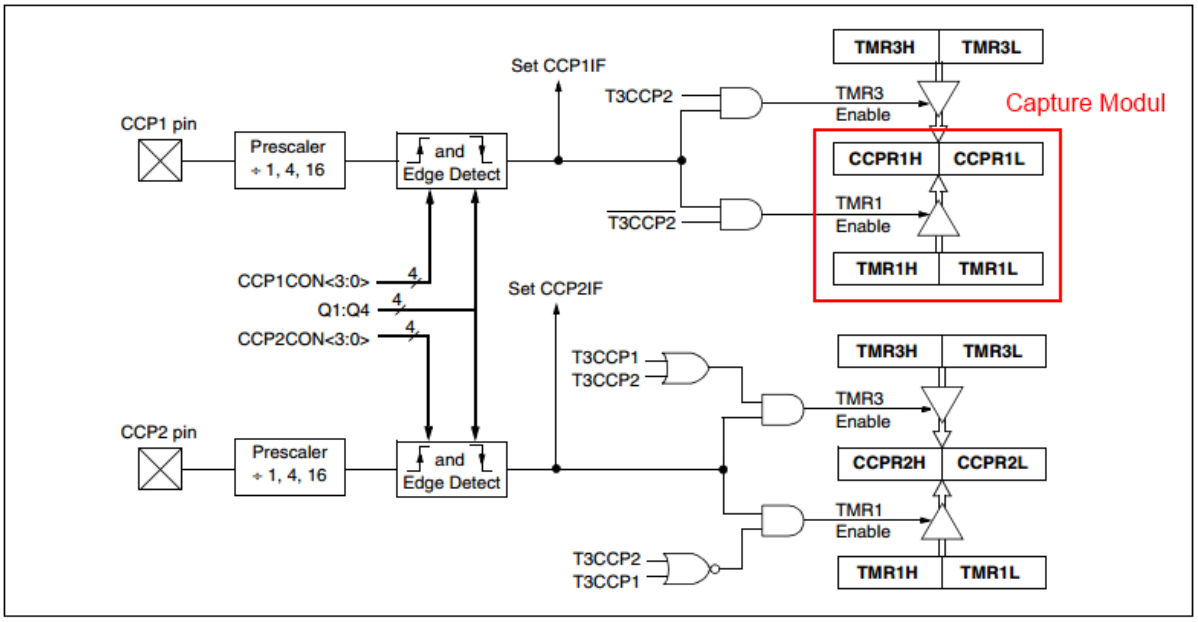


Abbildung 16 Aufbau des Capture Modul

Die Aufgabe des Capture Moduls ist die Erfassung der Zeit zwischen zwei steigenden Flanken. Anschließend wird diese Zeit in den Registern CCPR1H und CCPR1L gespeichert und in Register CCP1IF ein Interrupt ausgelöst. Die Konfiguration wird über das Register CCP1CON eingestellt. In diesem Fall wird folgende Bitkombination gewählt: 0b00000101. Dadurch wird das Erfassung Modul auf die Steigende Flanke eingestellt. Es wäre noch möglich den Erfassungsmodul auf die fallende Flanke einzustellen oder jeder 4 steigende Flanke, jedoch für diese Aufgabenstellung wird die gewählte Konfiguration seine Aufgabe erfüllen.

Die gemessenen Zeiten werden in der ISR gespeichert und dienen als Grundlage für die Berechnung des Zündzeitpunktes.

4.2.8. Compare Modul

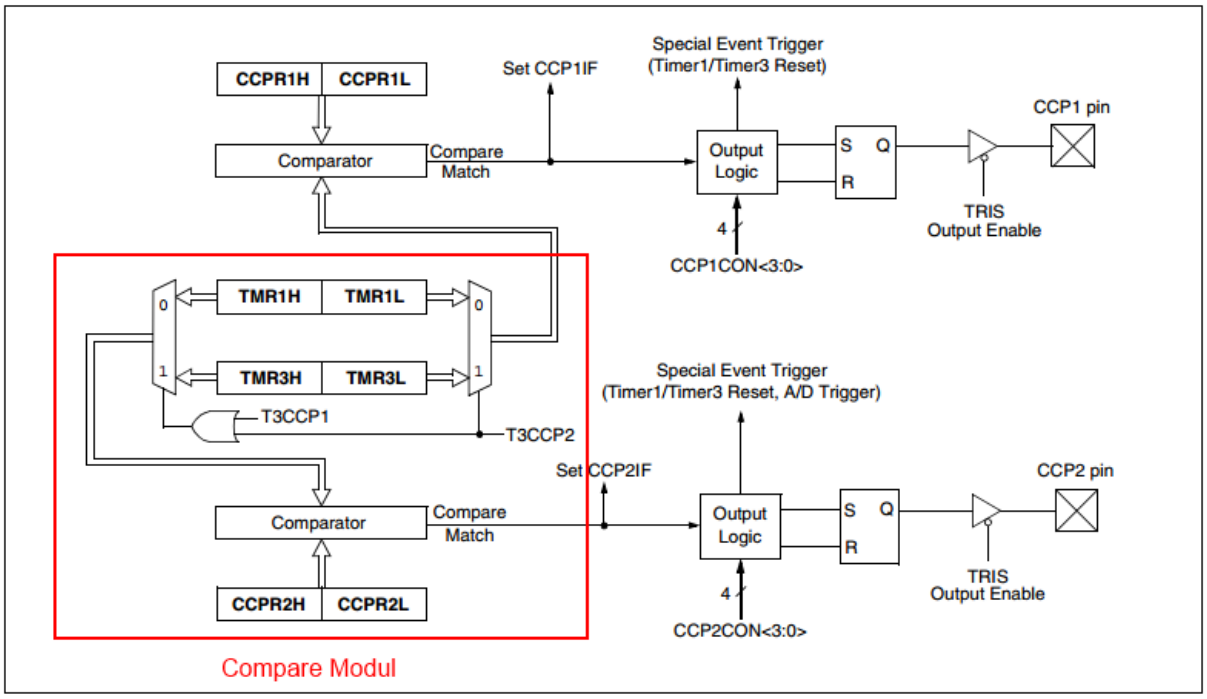


Abbildung 17Aufbau des Compare Moduls

Das Compare Modul vergleicht zwei Werte auf Gleichheit. Bei Gleichheit löst das Modul im Register CCP2IF ein Interrupt aus. Der Komparator vergleicht dabei den in der Lookup-Tabelle abgelegten Zündzeitpunkt (liegt auch in Register CCPR2H, CCPR2L) mit dem aktuellen Timerwert (Zeitwert) des Timers 1. Ist der aktuelle Zeitwert gleich dem abgelegten Zeitwert, so wird ein Interrupt ausgelöst.

Die Konfiguration erfolgt durch das Setzen des Registers CCP2CON auf die Bitkombination 0b00001010. Dadurch wird das Compare Modul aktiviert.

4.2.9. Interrupt Service Routine

Für das einwandfreie Messen, Rechnen und Zünden mit Hilfe des µController ist es von Vorteil die einzelnen Aufgaben in kleinen Unterroutinen aufzugliedern. Diese Unterroutinen werden Interrupt Service Routinen (ISR) genannt. Folgende Unterbrechungen werden konfiguriert und aktiviert.

RCONbits.IPEN = 1;

Schaltet die Interrupt Prioritätsunterscheidung ein.

INTCON = 0x40;

Der Peripherie Interrupt wird aktiviert.

PIE1 = 0b00000101;

Der Timer1 Überlauf und das Capture Modul werden aktiviert.

IPR1 = 0b00000101; IPR2 = 0x01;

Die Interrupt Stufen werden festgelegt

PIE2 = 0x01;

Das Compare Modul wird aktiviert.

INTCON2 = 0xf0;

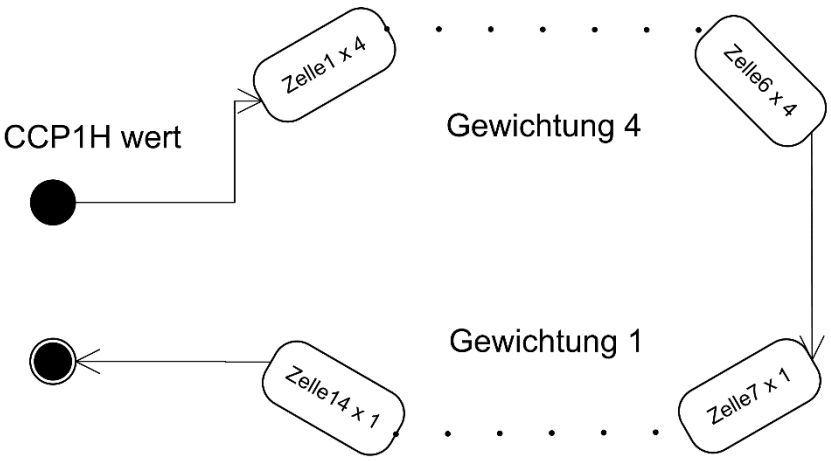
Die Interrupts werden bei steigende Flanke ausgeführt

INTCON3 = 0x00;

Ausschalten von nicht gebrauchten Interrupts.

4.2.10. Gewichtete Mittelwert-Berechnung von zukünftigen Verzögerungszeit

4.2.10.1. Berechnungsablauf



$$\frac{\left(\sum_{1}^{6} Zelle\ [i] * 4\right) + \left(\sum_{7}^{14} Zelle\ [j] * 1\right)}{\left(\sum_{1}^{6} i * 4\right) + \left(\sum_{7}^{14} j * 1\right)}$$

Abbildung 18 Berechnungsformel und Speicher Struktur von gesuchten verzögerungszeiten

Die Berechnung der Verzögerungszeiten für die Zündung basiert auf einer Formel, die die neuste Werte (Zeitwerte) mehr gewichtet als die „älteren" Zeiten. Dadurch wird die zeitliche Unregelmäßigkeit des Eingangssignals stabilisiert (Behebung des Jitters im Eingangssignal) und es wird ein regelmäßiges und zeitlich stabiles Ausgangssignal generiert. Ohne diese Pufferung wäre die Ausgabe fehlerhaft, da diese auf falschen Messwerten beruht. Es würde zu sprunghaften Zündungen kommen, welche den Motor beschädigen könnten. Der Ringpuffer-Algorithmus kann leider nicht schnell auf Veränderungen reagieren(siehe Kapitel Ausblick). Bei jedem Messzyklus werden die 8 höherwertigen Bits des Capture Registers in die erste

Zelle des Ringpuffers gespeichert und die alten Werte der Zellen um eine Position weiterverschoben.

```
timer_low[7] = timer_low[6];

timer_low[6] = timer_low[5];

timer_low[5] = timer_low[4];

timer_low[4] = timer_low[3];

timer_low[3] = timer_low[2];

timer_low[2] = timer_low[1];

timer_low[1] = timer_low[0];


temp=timer_high[5];

timer_high[5] = timer_high[4];

timer_high[4] = timer_high[3];

timer_high[3] = timer_high[2];

timer_high[2] = timer_high[1];

timer_high[1] = timer_high[0];

timer_low[0]=temp;

timer_high[0]=CCPR1H;
```

Danach werden die gespeicherten Messwerte im Ringpuffer mit verschiedenen Gewichtungen versehen und aufaddiert(siehe Abbildung 18 Berechnungsformel und Speicher Struktur von gesuchten verzögerungszeiten). Die schnellste Art der Multiplikation in einem µController ist durch eine Verschiebung von Bits zu realisieren. Es ergibt sich bei einer Gewichtung von vier ein zweimaliges bitweises Verschieben nach links.

```
for(i=0; i<6; ++i)

{

temp2=timer_high[i];

timer3+=temp2<<2;

}
```

```
for(i=0; i<8; ++i)

{

temp2=timer_low[i];

timer3+=temp2;

}
```

Am Ende wird durch 32 geteilt (Mittelung über alle 14 abgelegten Messperioden), was einer Verschiebung von fünf Bits nach rechts entspricht.

```
timer3 = timer3 >> 5;
```

Aus der LookupTable, eine Liste mit Verzögerungswerten, wird der gesuchte Verzögerungswert ausgelesen und in der Variable sub_wert gespeichert.

```
sub_wert = lookup[timer3];
```

Am Ende wird der Compare Modus mit dem neusten Wert geladen.

```
CCPR2L=sub_wert;

CCPR2H=sub_wert>>8;
```

4.2.11. PC-µC Verbindung

Die Konfiguration der UART-Verbindung des µC sieht wie folgt aus:

```
TRISCbits.RC6 = 0; //TX pin set as output

TRISCbits.RC7 = 1; //RX pin set as input
```

Die Pin-Belegung des UART mit seinen Ein- und Ausgängen wird festgelegt.

```
SPBRG = 12;

TXSTA = 0b00100010;

RCSTA = 0b10010000;
```

Die nächste Konfiguration legt die Datenrate auf 9600 Baud fest. Für die Ein- und Ausgänge wird die asynchrone Verbindung im 8-Bit-sende-Modus eingestellt.

```
TXREG = 0x00;
```

Löscht den Empfangspuffer.

RCIF = 0; //reset RX pin flag

RCIP = 1; //Not high priority

RCIE = 1; //Enable RX interrupt

Die Interruptroutine für die UART-Verbindung wird aktiviert und auf LOW-Priorität gesetzt.

Die Interruptroutine sieht wie folgt aus:

```
if(mssglen<6 && send_frei == 1) //buffer size
        {
           dummy = ReadUSART();
           if(dummy == 0x0D) //check for return key
           {
              lookup[lookZahl]=atoi(MessageBuffer);
              MessageBuffer[0] = 0x00;
              MessageBuffer[1] = 0x00;
              MessageBuffer[2] = 0x00;
              MessageBuffer[3] = 0x00;
              MessageBuffer[4] = 0x00;
              MessageBuffer[5] = 0x00;
              mssglen = 0;
              send_frei = 0;
              if(lookZahl == 255)
              {
                 lookZahl =0;
              }
              else
              {
                 lookZahl++;
              }
              putsUSART("f");
              return;
           }
           else
           {
              MessageBuffer[mssglen] = dummy;
           }
            mssglen++;
        }
        if(send_frei == 0) //our buffer size
        {
           dummy = ReadUSART();
           if(dummy == 'g')
           {
              send_frei = 1;
           }
        }
```

```
        PIR1bits.RCIF = 0;
    }
```

Am Anfang wird überprüft, ob der µC das korrekte Zeichenformat empfangen hat. Dies wird durch das erste Zeichen („Dummy“) der Zeichenfolge überprüft. Entspricht das erste Zeichen dem „Dummy“-Buchstaben „g“, so wird die Variable send_frei auf 1 gesetzt.
Dadurch weiß das Programm, dass die Zeichen aus der Burn Software kommen. Anschließend wird die empfangene Zeichenkette mit Hilfe der atoi(MessageBuffer)-Methode in eine Zahl umgewandelt. Listeneintrag für Listeneintrag wird die LookUp-Table des µC überschrieben. Durch das Senden des Buchstabens „f“ wird der Burn-Software vom µC mitgeteilt, dass das Übertragen des aktuellen Listeneintrags erfolgreich beendet wurde und der nächste Eintrag gesendet werden kann. Das wird solange durchgeführt, bis alle Zahlen empfangen wurden.

4.2.12. Sicherheitsüberwachung

Wenn der Motor sich nicht mehr dreht, werden auch keine Rechteckimpulse durch den Unterbrecherkontakt generiert, weswegen der Eingang des µC ständig überwacht werden muss, um zu verhindern das Zündimpuls erzeugt werden, welche den Motor beschädigen könnten.

Die Überwachung wird mit Hilfe des Timer 3 umgesetzt.

```
if(PIR2bits.TMR3IF == 1)
    {
      T3CONbits.TMR3ON = 0;
      TMR3H = 0x00;
      TMR3L = 0x00;
      tmr3over = 1;
      PIR2bits.TMR3IF = 0;
    }
```

Das Ausschalten erfolgt sobald der Timers 3 überläuft. Jeder eingegangener Rechteckimpuls setzt den Timer zurück. Die Ausgangsschaltung bleibt solange aktiv, wie weitere Impulse empfangen werden.

4.3.Bedienung PC-Software

Die Software für die Erstellung von Zündzeitpunkt-Vorverstellungswerte, die für den Mikrocontroller gedacht sind, wurde ,,Burn‘‘ genannt.

Die Zündzeitpunkt-Vorverstellungswerte teilen dem Mikrokontroller mit, bei welcher Umdrehungszahl welche Frühzündungen gestartet werden. Die µC-Umsetzung wird in Kapitel 4.2.1 näher erläutert.

Die Bedienung erfolgt durch 5 unterschiedlich gefärbte Buttons (siehe Abbildung 19 Graphische Darstellung des Burns Software). Diese befinden sich auf der rechten Seite. Die genaue Erläuterung der Buttons und die dazugehörigen Funktionen werden später beschrieben. In der Mitte befindet sich die vom Anwender angegebene Frühzündkurve, die durch 3 verschiedene Module auch jederzeit änderbar ist.

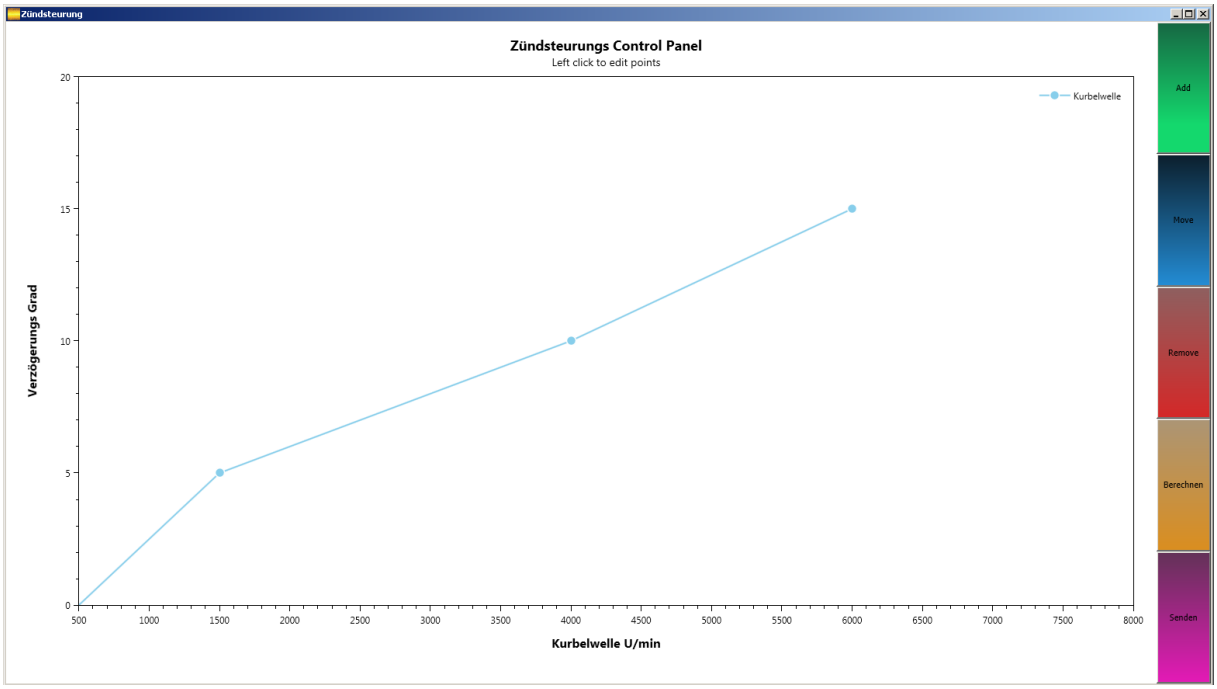


Abbildung 19 Graphische Darstellung des Burns Software

Es besteht die Möglichkeit neue Punkte in den existierenden Linienverlauf hinzuzufügen. Dazu wird mit der linken Maustaste auf den Add-Knopf geklickt. Anschließend wird mit Hilfe der linken Maustaste auf ein bestimmtes Drehzahlmoment geklickt und der gewünschte Verzögerungsgrad als neuer Punkt eingestellt.

Für die Verschiebung des bestehenden Punktes wird das Move-Modul verwendet. Durch Anklicken mit der linken Maustaste wird ein vorhandener Punkt verschoben

Mit Hilfe des Remove-Buttons kann der Benutzer einen falsch platzierten Punkt löschen. Das Entfernen von einem bestehenden Punkt kann durch Anklicken des gewünschten Punkts mit der linken Maustaste erfolgen.

Die Erstellung der Wertetabelle, die auch für die Mikrocontroller-Software verständlich umsetzbar ist, erfolgt über den Berechnen-Modus. Dies geschieht nachdem der Anwender den „Berechnen“-Knopf getätigt hat.

Nachdem der Anwender ein-Button gewählt hat, werden die anderen Möglichkeiten wie Verschiebung, Löschen etc. auf inaktiv geschaltet.

Für das Versenden der gewünschten Zündverzögerungspunkte zum µC ist der Senden-Modus zuständig.

Der Anwender muss zunächst das Verbindungskabel an den PC anschließen um sich mit dem Mikrocontroller zu verbinden. Anschließend wählt er den richtigen COM[6]-Port aus, mit dem der PC mit dem µController kommuniziert. Wird der falsche COM-Port ausgewählt oder das Kabel nicht richtig eingesteckt, so wird auf dem Bildschirm folgende Information angezeigt: „Der COM-Port wurde nicht ausgewählt!“.

Wenn der Senden-Vorgang erfolgreich abgeschlossen wurde, wird die Information ,,Versenden erfolgreich abgeschlossen‘‘ auf dem Bildschirm angezeigt.

Zur Umsetzung des Codes wird die Version 4.5 von Microsoft.Net genutzt. Als Entwicklungsumgebung wird Microsoft Visual Studio 2012 benutzt.

Die Software wurde in der Programmiersprache C# geschrieben und mit Hilfe der Tools WPF[7] und Oxyplot[8] unter der Berücksichtigung der Plotting-Bibliothek (Zeichnen) dargestellt.

[6] COM bekannt als auch RS-232 ist eine Spannungsschnittstelle. Die binären Zustände werden durch verschiedene elektrische Spannungspegel realisiert. Für die Datenleitungen wird eine negative Logik verwendet, wobei eine Spannung zwischen −3 V und −15 V eine logische Eins und eine Spannung zwischen +3 V und +15 V eine logische Null darstellt. Signalpegel zwischen −3 V und +3 V gelten als undefiniert.

[7] Windows Presentation Foundation (WPF) ist ein Grafik-Framework und Teil des .NET Framework von Microsoft. Die dient zur Gestaltung von Oberflächen und zur Integration von Multimedia-Komponenten.

[8] Oxyplot ist eine Open Source kostenlosse .NET zeichnen und plotting Bibliotheken Sammlung für die .NET Framework

4.3.1. UML Mappe

Die Architektur der PC-Bedienungssoftware wird in einem MVVM[9] Konzept entwickelt und erzeugt. Das Konzept besagt, dass der Anwender seine Vorgaben direkt in der Graphischen Darstellung auswählt und umsetzen kann.

Diese Vorgaben werden dann an das Model (siehe Kapitel Model Klasse) weitergegeben und dort verarbeitet. Die Änderungen werden dann an die Graphische Oberfläche versendet und angezeigt. Die Kommunikation zwischen der Oberfläche und der internen Struktur der Burn Software erfolgt zwischen MainWindow Klasse aus dem Burn UML-Baum und der Model Klasse die sich in Burn.Model UML-Baum befindet(siehe Abbildung 20 UML Diagramm des Burn Programms).

Für die gewünschte Funktionsfähigkeit werden externe Funktionsbibliotheken verwendet (siehe UML Diagramm Abschnitt Extern).

Es sind folgende Bibliotheken enthalten:

- Systemabhängige Bibliotheken
- Darstellungs-Bibliotheken
- Prozessor Bibliotheken
- Animations-Bibliotheken
- Konvertierungs-Bibliotheken

Die Windows abhängige Konfiguration wird in der Burn.Properties gespeichert und beim Öffnen des Programms in den Hauptspeicher geladen.

[9] Model View - View Model (MVVM) ist eine Variante des Model View Controller(MVC) zur Trennung von Darstellung und Logik der Graphischen Benutzer Schnittstelle.

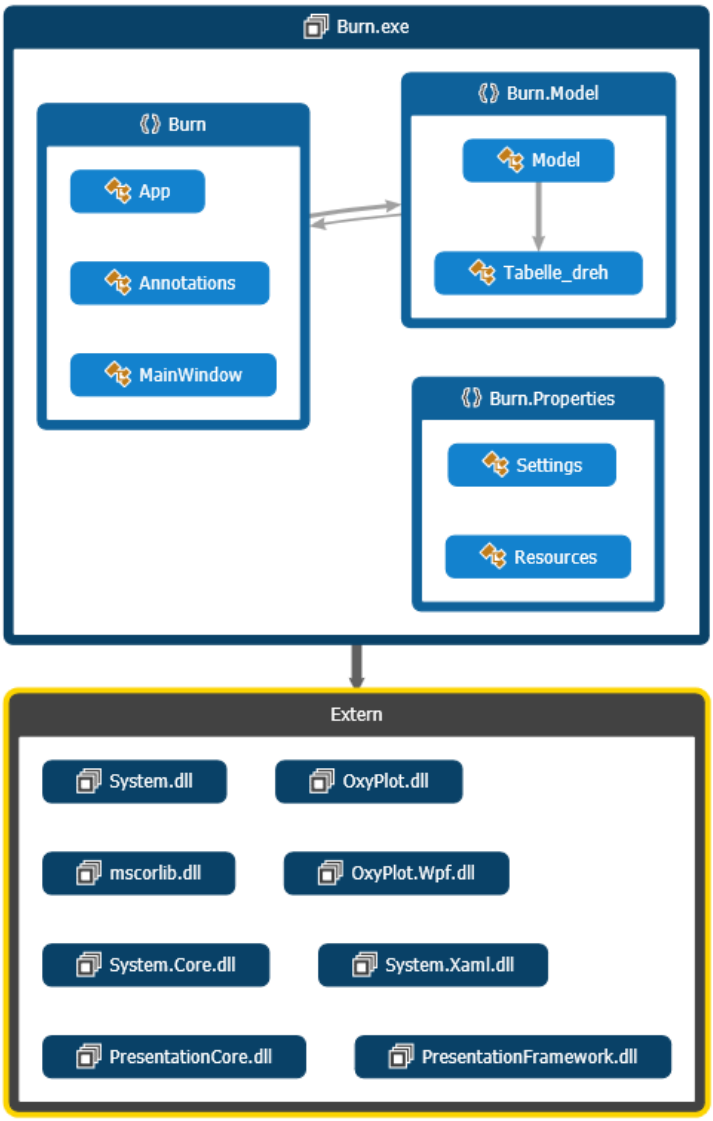


Abbildung 20 UML Diagramm des Burn Programms

4.3.2. Model Klasse

4.3.2.1. Klassen Model Diagramm

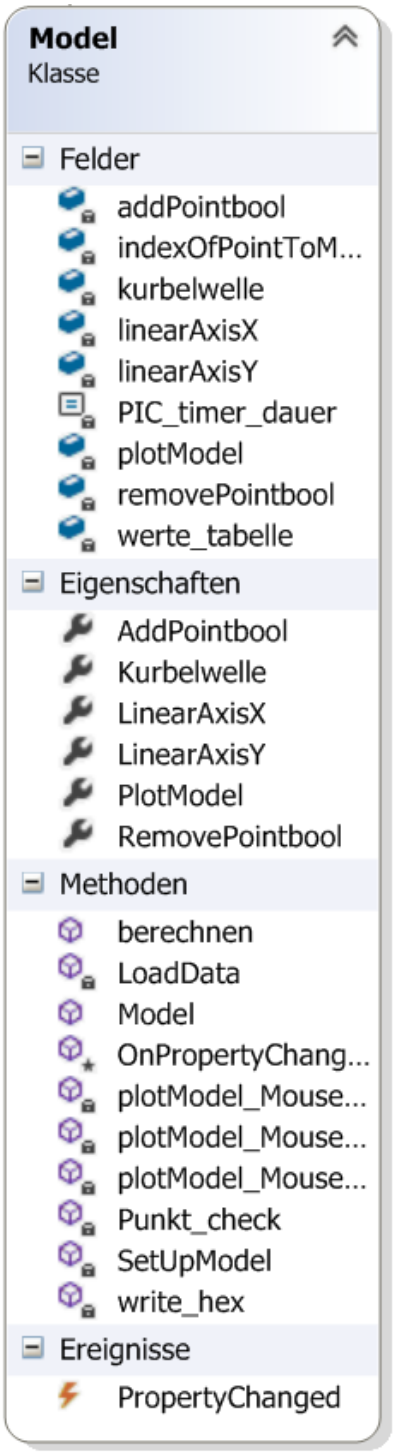


Abbildung 21 Klassen Model Diagramm

Ein Klassendiagramm ist ein Strukturdiagramm der Unified Modeling Language (UML) zur grafischen Darstellung (Modellierung) von Klassen, Schnittstellen sowie deren Beziehungen.

Eine Klasse ist in der Objektorientierung ein abstrakter Oberbegriff für die Beschreibung der gemeinsamen Struktur und des gemeinsamen Verhaltens von Objekten (Klassifizierung).

Sie dient dazu Objekte zu abstrahieren. Im Zusammenspiel mit anderen Klassen ermöglichen sie die Modellierung eines abgegrenzten Systems in der objektorientierten Analyse und Entwurf.

Die wichtigste und zugleich auch die umfangreichste Klasse erfüllt in der MVVM Architektur die Model Aufgabe.

Die Felder, Eigenschaften und Methoden sind gekapselt und als private deklariert, damit kein Missbrauch aus einer externen Software stattfindet kann.

Mit Hilfe dem PropertyChanged Event, das sich selbst aufrufen kann, kann man Aktualisierungsereignisse automatisieren.

4.3.2.2. Klassen-Konstruktor

Der Klassen-Konstruktor hat die Aufgabe alle nötigen Werte und Initialisierungen vorzubereiten und dies innerhalb der Methode durchzuführen.

```
public Model()
        {
            PlotModel = new PlotModel();
            linearAxisX = new LinearAxis(AxisPosition.Bottom, 500, 8000,500,100);
            linearAxisX.Title = "\nKurbelwelle U/min";
            linearAxisX.TitleFontSize = 16;
            linearAxisX.TitleFontWeight = 500;
            linearAxisX.IsZoomEnabled = false;
            linearAxisX.IsPanEnabled = false;
            linearAxisY = new LinearAxis(AxisPosition.Left, 0, 45, 5, 1);
            linearAxisY.Title = "Verzögerungs Grad\n\n";
            linearAxisY.TitleFontWeight = 500;
            linearAxisY.TitleFontSize = 16;
            linearAxisY.IsZoomEnabled = false;
            linearAxisY.IsPanEnabled = false;
            PlotModel.MouseDown += new
            EventHandler<OxyPlot.OxyMouseEventArgs>(plotModel_MouseDown);
            PlotModel.MouseMove += new
            EventHandler<OxyPlot.OxyMouseEventArgs>(plotModel_MouseMove);
            PlotModel.MouseUp += new
            EventHandler<OxyPlot.OxyMouseEventArgs>(plotModel_Mouseup);
            kurbelwelle = new LineSeries();
            werte_tabelle = new List<Tabelle_dreh>();
            SetUpModel();
            LoadData();
        }
```

Am Anfang wird das Zeichnen-Model erzeugt und mit dem Namen „PlotModel“ versehen. Danach werden die X-Achsen und Y-Achsen begrenzt, sodass der Anwender nur zwischen 1000 bis 8000 Kurbellwellenumdrehungen pro Minute den Zündlinienverlauf wählen kann. Die Skala des Verzögerungsgrads (Y-Achse)fängt bei 0 Grad an und endet bei 20 Grad.

Anschließend wird der Eventhandler für die Maus-Bewegungen aktiviert. Zuletzt wird ein Listen Array mit je einem Klassenprofil für jeden neuen Punkt vorbereitet. Dies wird in Kapitel 4.3.3 näher erläutert.

4.3.3. Klasse Tabelle_dreh

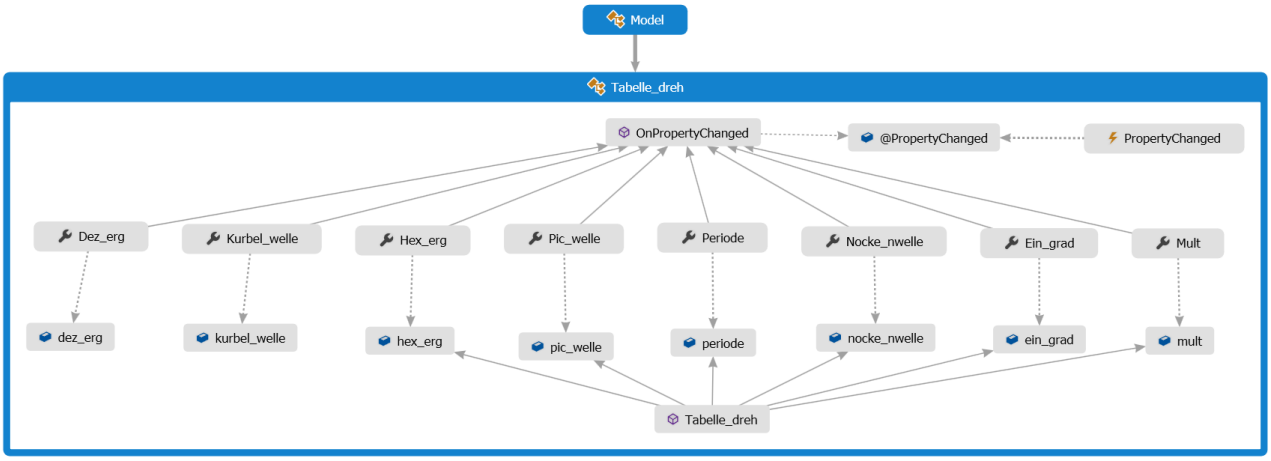


Abbildung 6 Attributen Struktur der Model Klasse

Die Klasse wurde entwickelt um eine saubere Strukturierung der Software zu erreichen. Sie beinhaltet folgende Attribute:

- `double nocke_nwelle`
- `double pic_welle`
- `double kurbel_welle`
- `double ein_grad`
- `double mult`
- `double Periode`
- `int dez_erg`
- `String hex_erg`

Jeder Punkt, der in der Visualisierung vom Anwender angegeben wird, besitzt die oben aufgelisteten Attribute und die dazu passenden Werte.

Um einen kontrollierten Zugriff auf die Variablen zu gewährleisten, werden diese gekapselt und können deswegen nur mit Getter und Setter Methoden von außen angesprochen werden. (siehe unteres Beispiel).

```
public double Kurbel_welle
 {
     get { return kurbel_welle; }
     set { if(value > 0)
             {
                     kurbel_welle = value; OnPropertyChanged("Kurbel_welle");
             }
         }
 }
```

Ohne eine Kapselung wäre jederzeit ein öffentlicher Zugang zur unkontrollierten Änderung dieser Variablen vorhanden. Dies könnte unter Umständen zu undefinierten Änderungen

oder zum Programmabsturz führen. Es ist daher immer ratsam alle Variablen durch Getter und Setter Methoden zu schützen.

Im abgebildeten Codeausschnitt wird die private Variable „kurbel_welle“ erst aktualisiert, wenn der gewünschte Wert größer als 0 ist. Dies schütz vor möglichen negativen Zahlen, welche das Programm für die Variable nicht vorgesehen hat. Ohne die Kapselung wäre dies nicht möglich.

4.3.4. Hinzufügen

Es besteht die Möglichkeit neue Punkte in den existierenden Linienverlauf hinzuzufügen. Dazu wird mit der linken Maustaste auf den Add-Knopf geklickt. Anschließend wird mit Hilfe der linken Maustaste auf ein bestimmtes Drehzahlmoment geklickt und der gewünschte Verzögerungsgrad als neuer Punkt eingestellt.

Nachdem der Anwender den Add-Button gewählt hat, werden die anderen Möglichkeiten wie Verschiebung, Löschen etc. auf inaktiv geschaltet.

Nachdem ein Mausklick auf die gewünschte Stelle erfolgte, wird der EventHandler für ein MouseDow-Event aufgerufen.

```
if (e.ChangedButton == OxyMouseButton.Left)
{
  kurbelwelle.Points.Add(Axis.InverseTransform(e.Position,linearAxisX,linearAxisY));
   if (kurbelwelle.Points.Count != 1)
       {
       if(kurbelwelle.Points[kurbelwelle.Points.Count-1].X < kurbelwel-
       le.Points[kurbelwelle.Points.Count-2].X)
       {
       kurbelwelle.Points[kurbelwelle.Points.Count - 1].X = kurbelwel-
       le.Points[kurbelwelle.Points.Count - 2].X + 1;
       }
       }
 }
   PlotModel.RefreshPlot(true);
```

In diesem Handler wird zunächst geprüft ob die Angabe durch die linke Maustaste erfolgte. Ist dies der Fall, wird ein neuer Punkt in den schon bestehenden Linienverlauf hinzugefügt. Anschließend wird geprüft, ob der neue Punkt vielleicht nicht vor dem letzten Punkt angegeben wurde.

Falls es irrtümlicherweise doch so wäre, wird der neue Punkt um eine Einheit hinter den letzten existierenden Punkt in X-Richtung verschoben. Es muss hierbei vom Anwender auf eine eindeutige Zuordnung (Funktions-Prinzip) geachtet werden. Ohne diese Überprüfung ist die spätere Berechnung nicht mehr möglich.

Die Berechnung ist auf einen kontinuierlichen Verlauf von hintereinander platzierten Punkten ausgelegt. Die letzte Überprüfung gewährleistet einen sinnreichen Verlauf für die Berechnung von einer Tabelle die der Mikrocontroller auch umsetzen kann.

Ein nicht kontinuierlicher Verlauf würde zu Zündungssprüngen während des Zündungsverlaufs führen. Was bei einem Ottomotor zur einen schlechten Zündzeitpunkt führen würde. Das birgt die Gefahr, dass es zu unerwünschten Explosionen im Motor kommen kann und dadurch der Motor beschädigt wird.

4.3.5. Verschiebung

Für die Verschiebung des bestehenden Punktes wird das Move-Modul verwendet. Durch Anklicken mit der linken Maustaste wird ein vorhandener Punkt verschoben (Siehe Aufruf unten).

```
OxyPlot.TrackerHitResult result = series.GetNearestPoint(e.Position, true);
```

Die Position wird mit Hilfe der Funktion GetNearestPoint() ermittelt und durch die Mausbewegung neu definiert. Erst beim Loslassen der linken Maustaste bleibt der verschobene Punkt auf der neuen Position.

```
kurbelwelle.Points[indexOfPointToMove] = kurbelwelle.InverseTransform(e.Position);
PlotModel.RefreshPlot(true);
```

Am Ende wird die Darstellung aktualisiert.

4.3.6. Remove Button

Mit Hilfe des Remove-Buttons kann der Benutzer einen falsch platzierten Punkt löschen. Das Entfernen von einem bestehenden Punkt kann durch Anklicken des gewünschten Punkts mit der linken Maustaste erfolgen.

```
if (e.ChangedButton == OxyMouseButton.Left)
{
Series series = PlotModel.GetSeriesFromPoint(e.Position, 10);
     if (series != null)
     {
     OxyPlot.TrackerHitResult result = series.GetNearestPoint(e.Position, true);
```

```
                        if (result != null && result.DataPoint != null)
                        {
                            indexOfPointToMove = (int)Math.Round(result.Index);
                            kurbelwelle.Points.RemoveAt(indexOfPointToMove);
                            PlotModel.RefreshPlot(true);
                        }
        }
}
```

In diesem Handler wird zunächst geprüft ob die Angabe durch die linke Maustaste erfolgte. Falls es die linke Taste war, wird unter der Bedingung, dass der Mauszeiger in der Nähe des gewünschten Punktes ist, der Punkt aus der Linie mit Hilfe der Methode RemoveAt(), entfernt.

4.3.7. Berechnen Button

Die Erstellung der Wertetabelle, die auch für die Mikrocontroller-Software verständlich umsetzbar ist, erfolgt über den Berechnen-Modus. Dies geschieht in 2 Schritten nachdem der Anwender den „Berechnen“-Knopf getätigt hat.

Die Wertetabelle besteht aus 256 Einträgen, welche jeweils aus 16-Bit unsigned Integer aufgebaut sind. Dies entspricht einem Zahlenbereich von 0 bis 65536. Die Zahlen entsprechen der Timer Internen Zählung, die dem µC mitteilt wann dieser für den jeweiligen Fall zünden soll (siehe Kapitel 4.2.1).

4.3.7.1. Punkt_Check() Methode Schritt 1

Die Punkt_Check Methode prüft in einer Schleife, ob alle Punkte hintereinander platziert wurden. Falls der neu platzierte Punkt auf der X-Achse vor seinem Vorgänger steht, so wird der neue Punkt auf die gleiche X-Achsen-Position, wie der Vorgänger verschoben, dabei bleibt die Y-Achsen-Position wie eingestellt erhalten. Anschließend wird der X-Wert (Kurbelwellenumdrehungszahl) des neuen Punktes um eine Einheit erhöht. Es dürfen weder mehrdeutige Zuweisungen entstehen noch darf ein neuer Punkt vor seinem Vorgänger in X-Achse-Richtung platziert werden.

```
for (int i = 0; i < kurbelwelle.Points.Count; i++)
{
    if( i != kurbelwelle.Points.Count -1)
                    if (kurbelwelle.Points[i].X > kurbelwelle.Points[i + 1].X)
                    {
                        kurbelwelle.Points[i + 1].X = kurbelwelle.Points[i].X + 1;
                    }
}
```

4.3.7.2. Berechnen() Methode Schritt 2

Zunächst wird überprüft, ob sich der gewünschte Kurvenverlauf umsetzen lässt. Ist dies nicht der Fall, kann die spätere Berechnung nicht mehr durchgeführt werden. Die Berechnung ist auf einen kontinuierlichen hintereinander platzierten Punkteverlauf ausgelegt und jegliche Unstimmigkeiten im Linienverlauf könnten zu Problemen bei der Berechnung führen und nachher wie im Kapitel 4.3.4 schon erwähnt zu einer Beschädigung des Motors führen. Mit folgendem Ausdruck wird dies durchgeführt:

```
if (Punkt_check() == true)
```

Die Punkt_check() Methode führt diese Prüfung durch (siehe Kapitel 4.3.7.1). Ist die Überprüfung erfolgreich verlaufen, so kann die Berechnungsschleife starten. In der Schleife werden die Anwender-Angaben in einer Liste zusammengefasst.

Diese Liste nennt sich

```
temp_liste = kurbelwelle.Points.ToList();
```

und ist in verschiedene Fälle untergliedert. Jeder Fall verläuft zwischen 2 vom Anwender definierten Punkten. Pro Fall wird die Steigung zwischen den Punkten berechnet. Die Kurbelwellenumdrehungszahl und der Verzögerungsdifferenzgrad werden in den Variablen delta_grad und delta_dreh gespeichert.

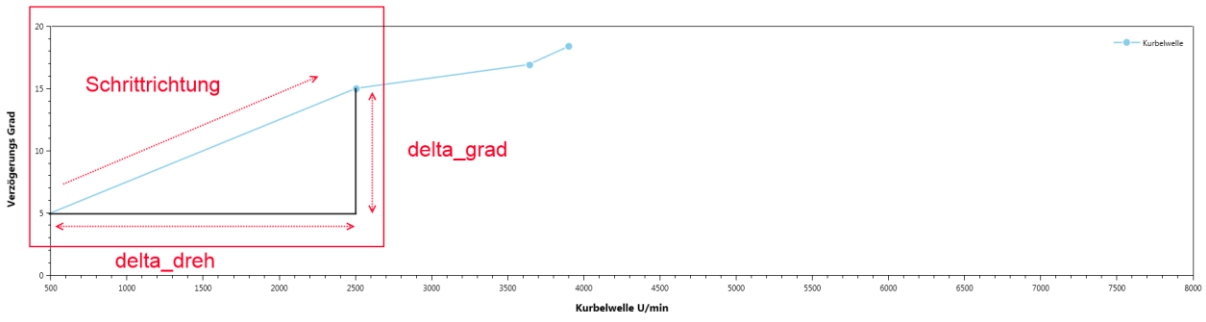


Abbildung 23 Graphische Darstellung des Berechnungsprozesses in Burn

Nachdem die Grenzen und Differenzen festgelegt wurden kann die Berechnungsschleife starten.

Am Anfang fängt die Berechnungsschleife bei 500 Kurbelwellenumdrehungen an. In der Schleife wird die Kurbelwellenumdrehungszahl inkrementiert und anschließend überprüft ob

sie das festgelegte Maximum für den aktuellen Fall erreicht hat (siehe Abbildung 23 Graphische Darstellung des Berechnungsprozesses in Burn).

Wenn das Maximum noch nicht erreicht ist, wird in der Schleife die Kurbelwellenumdrehungszahl um ein Grad pro Minute erhöht. Dies wird solange durchgeführt, bis die inkrementierte Kurbelwellenumdrehungszahl dem Maximum des aktuellen Falls entspricht. Jeder Fall hat seine eigene Steigung, deshalb werden die vorher ausgerechnete Steigung, der Verzögerungsgrad und die Kurbelwellenumdrehungszahl in der Schleife miterhöht.

```
while (dreh_aktuel < dreh_max)
{

if (werte_tabelle.ElementAt(j).Kurbel_welle <= dreh_aktuel)
    {
         werte_tabelle.ElementAt(j).Mult = (int)Math.Round(grad_aktuel);
                                    j++;
    }
else
    {
    dreh_aktuel++;
    grad_aktuel = grad_aktuel + schritweise;
    }
}
```

Die vorher ausgerechneten Punkte für die Mikrocontrollerliste werden in der Variable „kurbel_welle“ gespeichert. Sobald die Schleife die Inkrementierung soweit durchgeführt hat, dass der Wert der Variablen „kurbel_welle“ gleich der Variablen „dreh_aktuel“ ist, wird der aktuelle Verzögerungswert für die Geschwindigkeit in der Variable „Mult“ gespeichert. Wenn die Ausrechnung des Verzögerungsgrads für alle gesuchten Kurbellwellen Punkte abgeschlossen ist, wird die tatsächliche zeitliche Verzögerung in der nächsten Schleife mit folgender Formel für jeden einzelnen Fall ausgerechnet.

```
(Periode - (Ein_grad * Mult)) / PIC_timer_dauer
```

4.3.8. Senden Button

Für das Versenden der gewünschten Zündverzögerungspunkte zum µC ist der Senden-Modus zuständig.

In Abbildung 24 Senden Modus sind auf der rechten Seite die Bedienknöpfe zu sehen. Auf der linken Seite befindet sich eine Tabelle von möglichen COM-Ports, die dem Bedienung–Pc zur Verfügung stehen. Mit den Schaltflächen „OK Senden“ (in blau) und „Abrechen“ wird der Sende-Vorgang gestartet oder abgebrochen. Im unteren Teil des Fensters ist in roter Schrift das Informationsfeld platziert.

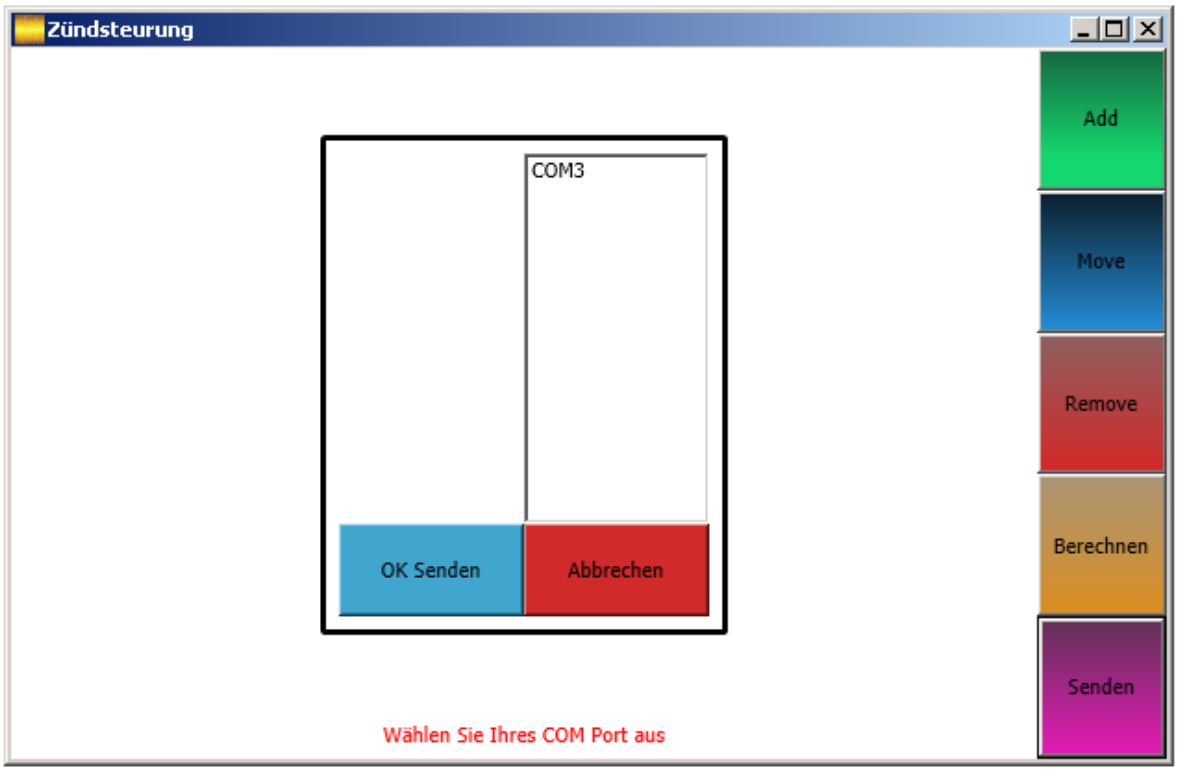


Abbildung 24 Senden Modus

Der Anwender muss zunächst das Verbindungskabel an den PC anschließen um sich mit dem Mikrocontroller zu verbinden. Anschließend wählt er den richtigen COM-Port aus, mit dem der PC mit dem µController kommuniziert. Wird der falsche COM-Port ausgewählt oder das Kabel nicht richtig eingesteckt, so wird auf dem Bildschirm folgende Information angezeigt: „Der COM-Port wurde nicht ausgewählt!“.

Wenn der Senden-Vorgang erfolgreich abgeschlossen wurde, wird die Information ,,Versenden erfolgreich abgeschlossen‘‘ auf dem Bildschirm angezeigt. Konnte die Nachricht hingegen nicht erfolgreich versendet werden, so erscheint folgende Information ,,Beim Senden ist ein Fehler aufgetreten‘‘.

Die Umsetzung erfolgt in der `senden_an_picmikrokontroller()` Methode.

```
public int senden_an_picmikrokontroller()
     {
         SerialPort port = new SerialPort(Comname, 9600, Parity.None, 8,StopBits.One);
         port.Open();
        for (int k = 255; k >= 0; k--)
        {
            port.Write("g"+werte_tabelle.ElementAt(k).Dez_erg.ToString() + "\r");
           port.ReadExisting();
            .
            .
            .
```

Der oben angezeigte Ausschnitt aus der versenden Methode zeigt, dass das Versenden in einer Schleife erfolgt, die auch die vorher ausgerechneten Werte mit einen „g“ + zahl +Enter Zeichen Format an den µC sendet und wartet auf seine Antwort durch die `port.ReadExisting();` Methode.

Das Format stellt sicher, dass nur die Daten aus der Burn Software vom µC angenommenen werden. Die µC Umsetzung finden Sie im Kapitel 4.2.11.

Die Kommunikation mit dem µC erfolgt mit einer Baudrate von 9600. Es wird keine Parität verwendet, jedoch wird ein Stopp-Bit am Ende jeder Zeile eingefügt.

4.3.9. Installation-CD

Damit die PC-Bedienungssoftware in einem Desktop PC ausführen werden kann, wird ein Installationspaket erstellt.

Das Installationspaket beinhaltet alle nötige für die Bedienungssoftware Software Pakete und die dazugehörige Nutz Bibliotheken.

Abbildung 25 Setup Icon

Um die Installation zur starten muss ein mindestens Windows XP basiertes Computers vorhanden sein und durch das öffnen entweder die Burn oder die Setup Datei, wird die Installation des Programms gestartet (siehe Abbildung 25 Setup Icon).

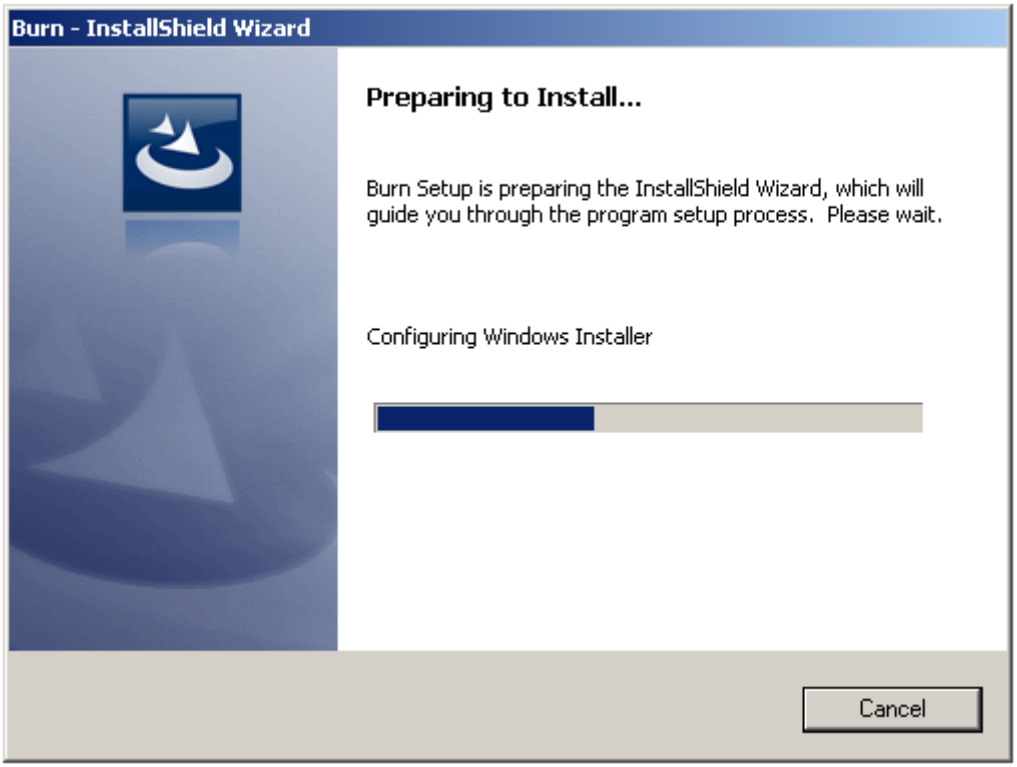


Abbildung 26 Installationsprogramm

Nachdem das Programm gestartet wird, prüft die Software welche Komponente nachinstalliert werden müssen(siehe Abbildung 26 Installationsprogramm). Anschließend installiert die Software das PC-Bedienungsprogramm auf dem Rechner und erstellt eine Desktopverknüpfung zu der Burn Software(siehe Abbildung 27 Desktopverknüpfung).

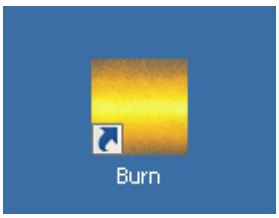


Abbildung 27 Desktopverknüpfung

5. Zusammenfassung

Die Arbeit befasst sich mit der Entwicklung und Verwirklichung einer µC basierter elektronischer Zündsteuergerät. Dafür wurden neue Platinen Layouts erstellt, eine neue PC-Bediensoftware zur benutzerdefinierten zündverhalten, mittels Erstellung einer Zündkurve, entwickelt. Sowie die Verbindung zwischen den Zündsteuergerät und der PC-Bedienungssoftware auf einen Desktop PC.

Bei der Entwicklung der Platine wurde eine Optimale Realisierung einer Eingangsschaltung und Ausgangschaltung und µC Schaltung geschaffen. Durch die bei der Zündung der Zündkerze erzeugten hohen Spannungen wurden starke Rückströme über die Platine geleitet, wodurch sich eine unzureichende Steuerung der Zündung einstellte. Das Spannungsversorgungsmodul des Mikrocontrollers wurde dabei so stark gestört, dass dieser rückgesetzt wurde und damit seine Funktion nicht mehr ausüben konnte.

Die Erfolgreiche Behebung des Problems konnte durch die folgende Neubaumaßnahmen vorgenommen werden:

- der Massenbezug wurde auf zwei getrennte Bereiche (Digital und Analog)aufgeteilt
- mittels zweier Optokoppler wurde der Digitalteil vom Analogteil getrennt
- durch den Einbau von Z-Dioden wurden die Spannungsspitzen begrenzt
- der Massenbezug des Digitalteils wurde als Massenfläche ausgelegt
- weiterhin wurden Stützkondensatoren zur Störungsunterdrückung eingesetzt

Es wurde auch versucht nur mit RLC-Tiefpässen diese Problematik zu beheben, jedoch ohne größere Erfolge.

Die Zündsteuereinheit wurde mit Hilfe des Mikrocontrollers PIC18F2550 in der Programmiersprache C entwickelt und umgesetzt. Der Zündungsverlauf wurde durch das Nutzen des Capture- und Compare-Moduls realisiert und in einer hoch priorisierten Interrupt Service Routine umgesetzt.

Die PC-Bediensoftware wurde in der Programmiersprache C# mit Hilfe der Tools WPF und Oxyplot Zeichnen-Bibliotheken verwirklicht. Die Software ,,Burn‘‘ kann einfach, durch die integrierten Modulen wie Neuer Zündpunkt hinzufügen, Drehzahl abhängiges Zündverlauf ändern, bedient werden.

Anschließend wurde mit Hilfe eines USB-UART Umsetzers die Verbindung zwischen dem PC und dem µC erstellt. Die Daten werden mit Hilfe der ,,Burn‘‘ Software und einem dafür vorgesehenen Verbindungskabel gesendet. Der Anwender bestimmt dabei selbst den Zündungsverlauf und somit auch indirekt welche Daten der µC erhält und umsetzt.

Die Aufgabenstellung konnte erfolgreich umgesetzt werden.

6. Ausblick

Die unvorhergesehenen Schwankungen am Unterbrecherkontakt verursachen eine gravierende Schwankung der Messperiode am µC-Eingang. Diese Schwankung beträgt ca. 400 µs pro Periode, wodurch der angesteuerte Zylinder zu früh oder zu spät gezündet wird. Der µC benötigt für die Zündimpuls-Generierung eine möglichst stabile Periodenlage.

Um den Schwankungen entgegenzuwirken werden 14 Perioden gemessen, in einem Ringpuffer gespeichert und unterschiedlich gewichtet. Anschließend werden die gewichteten Perioden gemittelt, um die statistisch möglichst exakte Periodendauer zu bestimmen. Der Ringpuffer fängt im Prinzip die Schwankungen auf, jedoch verlängert er die Reaktionszeit. Das System braucht länger um die gewünschten Zündzeitpunkte an die aktuelle Drehzahl anzupassen.

Ein möglicher Lösungsansatz wäre, statt der mechanischen Umsetzung der Rotationsbewegung der Nockenwelle in einen elektrischen Impuls mittels Unterbrecherkontakt, eine optische Abtastung der Drehzahl vorzunehmen. Auch ein Induktiver-Näherungsgeber wäre denkbar.

Bei beiden Ansätzen wäre eine stabilere Impulsfrequenzlage zu erwarten, wodurch sich die Anwendung des Ringpuffers erübrigt. Diese Lösungen umzusetzen würde aber den Rahmen der Arbeit sprengen.

7. Abbildungsverzeichnis

8. Quellenverzeichniss

Joachim Specovius: Grundkurs Leistungselektronik, Bauelemente, Schaltung und Systeme 6. Auflage, Springer-Verlag Wiesbaden 2013, ISBN 978-3-8348-2447-9

Andreas Gräßer :Analyse linearer und nichtlinearer elektrischer Schaltungen, Springer-Verlag Wiesbaden 2012, ISBN 978-3-8348-2369-4

PeterF.Orlowski: Praktische Elektronik, Analogtechnik und Digitaltechnik für die industrielle Praxis, Springer-Verlag Berlin Heidelberg 2013, ISBN 978-3-642-39004-3

Joachim Franz: EMV Störungssicherer Aufbau elektronischer Schaltungen, Springer Fachmedien Wiesbaden 2013, ISBN 978-3-8348-1781-5

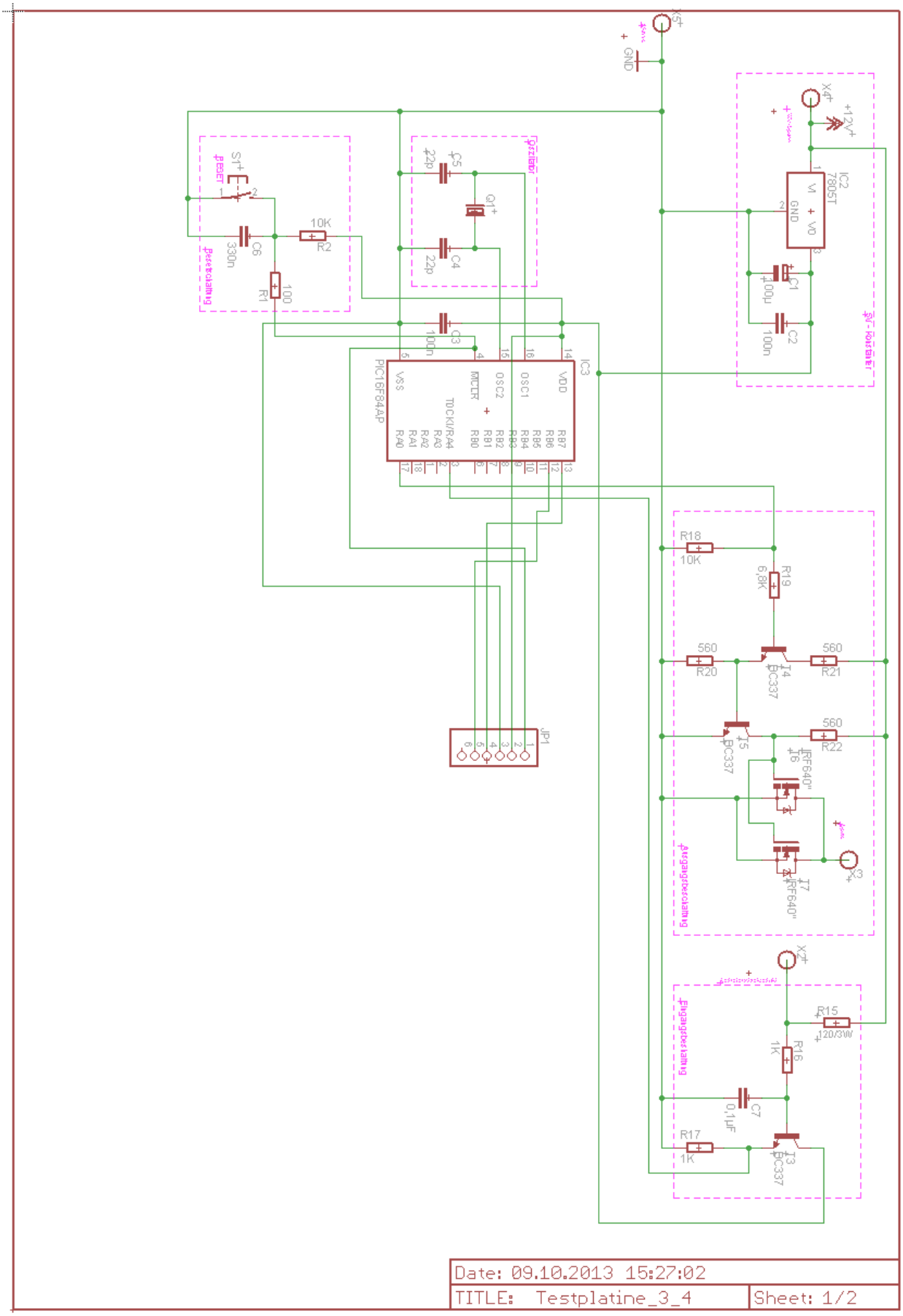
http://de.wikipedia.org

http://www.hk-auto.de

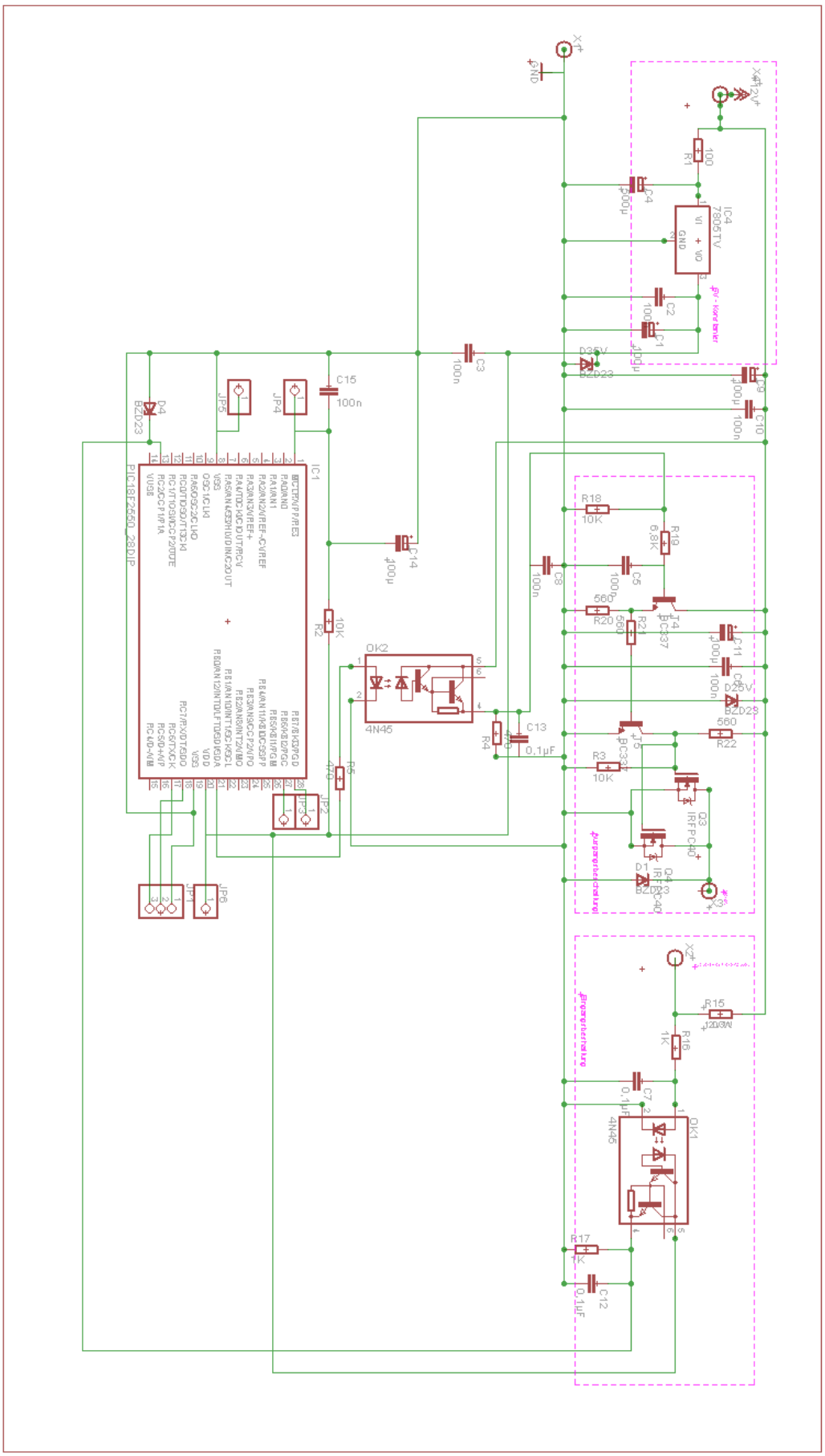
http://www.leifiphysik.de

http://www.osnanet.de

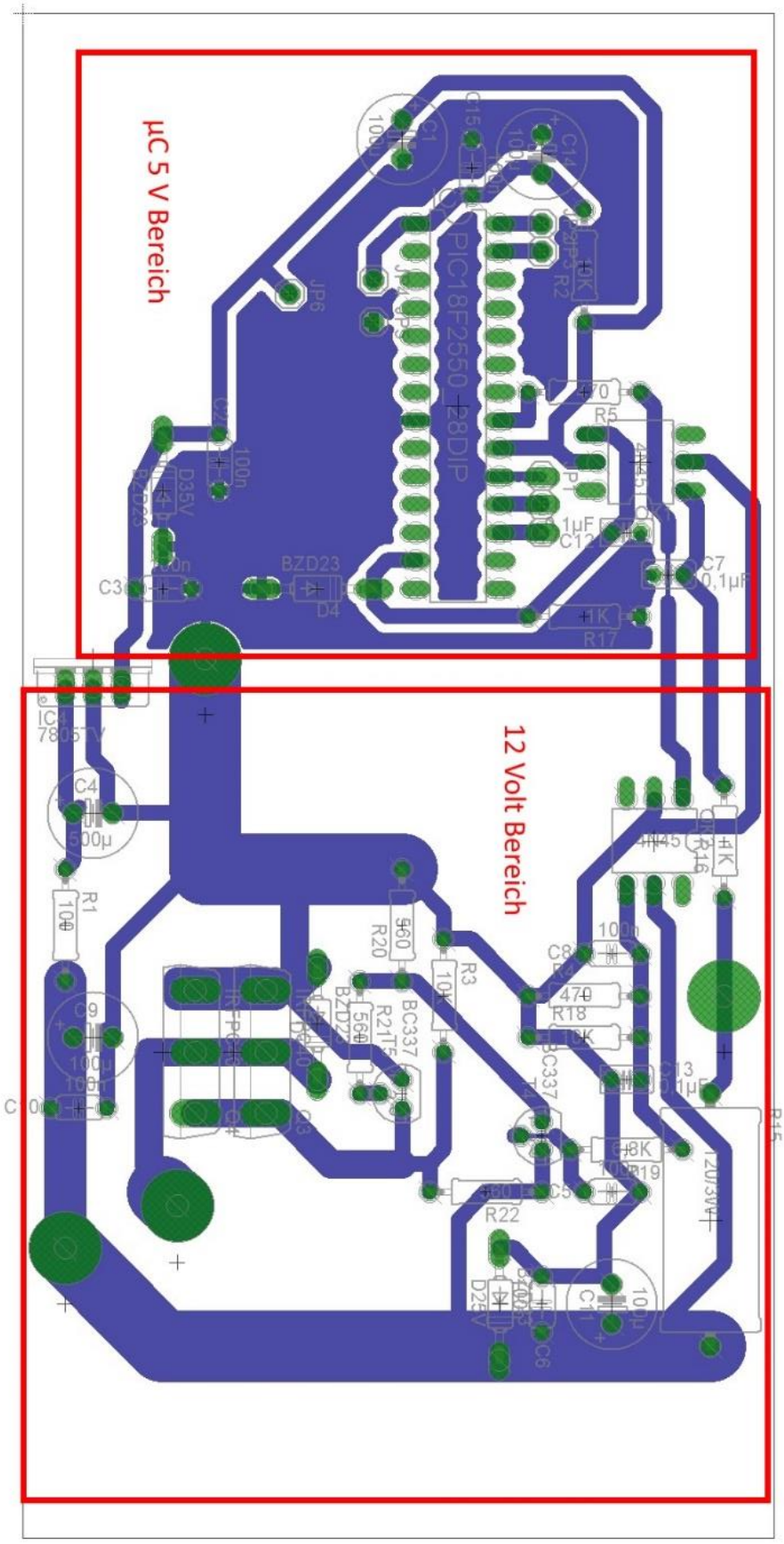
Anhang



Anhang 1 Schaltplan von Frau Donczewa



Anhang 2 Neu entwickelter Schaltplan



Anhang 3 Board Layout

Anhang 4 Mess-Zünd Ablauf