Christoph Aigner

Entwicklung eines Pulsoxymeters zur Bestimmung der Sauerstoffsättigung und der Herzrate von sedierten Labormäusen

Masterarbeit

Institut für Medizintechnik
Technische Universität Graz
Kronesgasse 5, A - 8010 Graz
Vorstand: Univ.-Prof.Dipl.-Ing.Dr.techn. Rudolf Stollberger

Betreuer: Dipl.-Ing. Clemens Diwoky

Begutachter: Univ.-Prof.Dipl.-Ing.Dr.techn. Rudolf Stollberger

Graz, (Oktober, 2012)

Deutsche Fassung:

EIDESSTATTLICHE ERKLÄRUNG

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

Graz, am ....
(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

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

....
(date) (signature)

Zusammenfassung

Ziel: Die vorliegende Masterarbeit beschäftigt sich mit der Entwicklung eines Pulsoximeters zur Anwendung an sedierten Labormäusen, um die Vitalparameter Herzfrequenz und Sauerstoffsättigung bestimmen und überwachen zu können. Dazu wurde eine Messhardware mit geeigneter Auswertesoftware in einem Mikrokontroller entworfen, die diese Parameter berechnet, auf einem LCD ausgibt und die digitalen Daten über eine realisierte USB-Verbindung einem MATLAB-GUI bereitstellt.

Methoden: Durch den Einsatz eines modifizierten Pulsoximetriesensors kann die unterschiedliche Absorption des Oxihämoglobins und Desoxyhämoglobins bei zwei verschiedenen Wellenlängen im arteriellen Blut gemessen und zur Berechnung der funktionellen Sauerstoffsättigung verwendet werden. Dazu regelt ein Mikrokontroller die LED-Ströme der Sendedioden, steuert die Signaltrennung sowie die analoge Signalverstärkung und digitalisiert die als Spannungssignal vorliegenden Pulswellensignale bei zwei unterschiedlichen Wellenlängen.

Ergebnisse: Nach einer digitalen Vorverarbeitung werden die Pulswellenamplituden extrahiert, daraus die Vitalparameter berechnet und auf einem LCD ausgegeben. Die digital im Mikroprozessorspeicher vorhandenen Signaldaten können über eine USB-Verbindung an einen PC mit einem MATLAB-GUI übertragen, bearbeitet und grafisch ausgewertet werden.

Schlussfolgerung: Mit der erstellten Hardware ist es möglich, die Herzfrequenz und die Sauerstoffsättigung einer sedierten Maus zu messen und dem Benutzer bequem über ein LCD darzustellen. Um die Korrektheit dieser Werte zu überprüfen, bedarf es jedoch einer Kalibration mit anschließender Validierung, die in dieser Arbeit nicht durchgeführt wurde.

Abstract

Objective: This master's thesis deals with the development of a pulse oximeter for use in sedated laboratory mice to determine the vital signs oxygen saturation and heart rate in order to allow continuous monitoring. To achieve that, a measuring hardware with the appropriate analysis software in a micro-controller has to be designed, which calculates these parameters, displays them on an LCD and supplies the digital data via a USB connection to a MATLAB GUI.

Methods: Through the use of a modified pulse oximeter sensor, the different absorption of oxihemoglobin and desoxihemoglobin at two different wavelengths in the arterial blood can be measured and used to calculate the functional oxygen saturation. For this purpose a microcontroller controls the LED currents of the transmitting diodes and the signal isolation, as well as the analog signal amplification and digitalizes the pulse wave signals at two different wavelengths.

Results: After a pre-processing, the amplitudes of the digital pulse wave are extracted, the heart rate and the oxygen saturation are calculated and displayed on an LCD. The digital signals that are present in the microprocessor memory data can be transferred via a USB connection to a PC with MATLAB GUI in order to process and analyze them graphically.

Conclusion: With the developed hardware it is possible to measure the heart rate and the oxygen saturation of a sedated mouse and display them clearly on an LCD. In order to verify the accuracy of these values a calibration with subsequent validation is needed. However, this has not been done in this study.

Inhaltsverzeichnis

Symbole und Abkürzungen

Symbole

I	$\mathrm{W/m^2}$	Strahlungsintensität
c	$\frac{\mathrm{mol}}{\mathrm{L}}$	Konzentration
ϵ	$\frac{\mathrm{L}}{\mathrm{mol\,m}}$	Extinktionsfaktor
d	m	Weglänge
R	1	Ratio der Pulswellenamplituden
SpO_2	%	funktionelle Sauerstoffsättigung
U	V	elektrische Spannung
I	A	elektrischer Strom
R	Ω	elektrischer Widerstand
C	F	Kapazität
GB	1	Gain–bandwidth product

Abkürzungen

LCD	Liquid Crystal Display
LED	lichtemittierende Diode
μP	Mikroprozessor
AC	Alternating Current (Wechselstrom)
DC	Direct Current (Gleichstrom)
FFT	Fast-Fourier-Transformation
PWM	Pulsweitenmodulation
OPV	Operationsverstärker
IR	Infrarot
VR	visible red (sichtbares Rot)
D-SUB	D-Subminiature
DIP	Dual in-line package
I/O	Input-Output
IDE	Integrierte Entwicklungsumgebung
SDK	Software Development Kit
RxD	Receive Data
TxD	Transmit Data
GUI	Graphical User Interface

1 Einleitung

Um den Vitalzustand eines Patienten bei Erstellung einer Anamnese während einer Untersuchung oder eines Eingriffs zu überprüfen, werden im Klinikalltag verschiedene biologische Parameter gemessen, um dadurch auf den Kreislaufzustand schließen zu können. Dazu gehören neben der Körpertemperatur, dem EKG und dem Blutdruck auch die mittels Pulsoximetrie durchgeführte indirekte Bestimmung der Sauerstoffsättigung [1]. Diese Größe ermöglicht es, Rückschlüsse über den Sauerstofftransport und damit indirekt Aussagen zur Effektivität der Atmung zu treffen [1], [2]. Da es sich zudem um ein nicht invasives, kontinuierliches und „nahezu zeitidentisches“ [1] Verfahren ohne Patientenrisiko handelt, gehört die Pulsoximetrie, erstmals 1972 von Takuo Aoyagi und Michio Kishi vorgestellt [3], vor allem in der Anästhesie zu einer Standardanwendung, um den Beatmungszustand und die Lungenfunktion effektiv überwachen zu können [1].

Dazu werden an einem leicht zugänglichen, gut durchbluteten und nach Möglichkeit unbehaartem Körperteil (wie etwa Finger, Zehen und Ohren) unter Einsatz von zumindest zwei LEDs mit verschiedenen Wellenlängen (meist im infraroten und sichtbaren roten Spektrum) die unterschiedliche Absorption des Gewebes gemessen. Die vorhandene Perfusion und in weiterer Folge die Herzaktivität steuert den Absorptionssignalen das sich zeitlich verändernde Pulssignal bei, dessen Frequenz und Amplitudenunterschiede verwendet werden, um die Herzfrequenz bzw. die Sauerstoffsättigung zu bestimmen.

1.1 Aufgabenstellung

In der vorliegenden Masterarbeit soll dieses für den Menschen bereits seit den 1980er Jahren eingesetzte Konzept [3] verwendet werden, um von Grund auf eine Hardware zu entwerfen, die eine Messung der Herzfrequenz und der Sauerstoffsättigung bei Mäusen durchführt. Dies soll ermöglichen, den Vitalzustand von sedierten Mäusen im Rahmen von Untersuchungen zu überwachen. Um die Bedienung der Messapparatur auf ein Minimum zu beschränken, ist die Auswertung nach erfolgter Sensorapplikation automatisiert durchzuführen und deren Ergebnisse sind auf einem LCD darzustellen. Dies bedeutet, dass die Signalgewinnung, Signalverarbeitung als auch die Parameterberechnung über einen μP realisiert werden soll. Des Weiteren sind über eine geeignete PC-Schnittstelle die gemessenen Daten zur mathematischen und analytischen Überprüfung einem PC bereitzustellen.

2 Methoden

In diesem Kapitel werden alle während der Entwicklung und dem Testen der Hardware verwendeten Grundlagen, Methoden, Materialien und Vorgehensweisen näher beschrieben. Um einen gewissen Überblick zu behalten, ist dieser Abschnitt in die sechs Bereiche „Grundlagen der Pulsoxymetrie“, „Designvoraussetzungen zur Anwendung der Pulsoxymetrie an Mäusen“, „Hardware“, „Mikroprozessor-Software“, „MATLAB-GUI“ und „Versuchsaufbau“ geteilt worden.

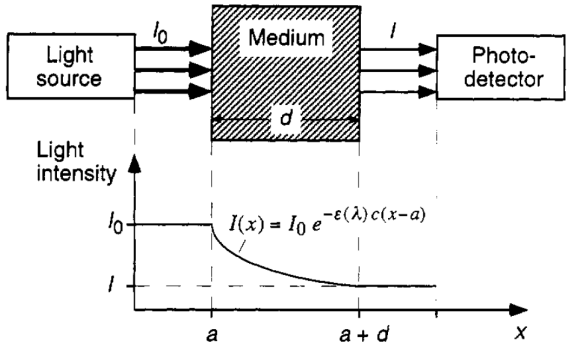
Der erste Teil befasst sich mit den theoretischen Hintergründen zu den Grundlagen der Pulsoxymetrie (siehe 2.1, S. 3). Dies beinhaltet eine kurze Einführung in die dazu notwendigen physikalischen Grundlagen (siehe 2.1.2, S. 7), deren Anwendung zur Berechnung der Sauerstoffsättigung (siehe 2.1.3, S. 8) und die dabei herrschenden limitierenden Faktoren (siehe 2.1.1, S. 4).

Um die physiologischen Unterschiede der Maus zum Menschen darzulegen und darauf hinzuweisen, worauf bei der Entwicklung eines Pulsoxymeters für Mäuse zu achten ist, werden diese Designvoraussetzungen im Unterpunkt 2.2 (S. 13) näher erläutert.

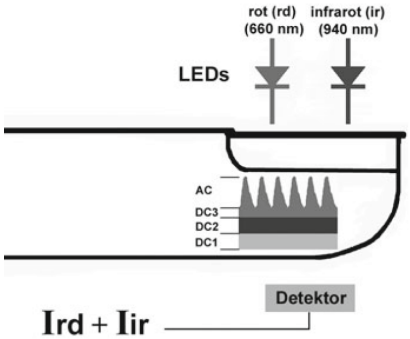
Im dritten Unterkapitel „Hardware“ wird der Fokus auf die Beschreibung der Hardwareentwicklung und deren Elemente gelenkt (siehe 2.3, S. 15). Ausgehend von einer Erläuterung des eingesetzten Blockschaltplans (siehe 2.3.2, S. 15), werden die Hardwareblöcke in den zugehörigen Unterkapiteln schrittweise näher ausgeführt. Es umfasst eine Beschreibung des Sensors als auch des durchgeführten Sensordesigns (siehe 2.3.3, S. 17), die Auswahl, Eigenschaften und Möglichkeiten des verwendeten μP-Entwicklungsboards (siehe 2.3.4, S. 19), das Design der LED-Treiberstufe (siehe 2.3.6) und der Verstärkerstufen (siehe 2.3.7). In einem weiteren Unterkapitel erfolgt die Erläuterung von weiteren unterstützenden Schaltungsteilen (siehe 2.3.8).

Im vierten Unterkapitel „Mikroprozessor-Software“ (siehe 2.4, S. 29) wird zunächst die zur Erstellung der Mikrokontrollersoftware notwendige Entwicklungsumgebung vorgestellt (siehe 2.4.1) und überblicksartig in einem Blockschaltplan die Interaktion der einzelnen Programmteile beschrieben (siehe 2.4.3, S. 32). Auf diese Programmteile wird in den darauf folgenden Unterpunkten 2.4.5 (Interrupts und Timer, S. 35), 2.4.6 (Regelung der LED-Ströme, S. 36), 2.4.7 (Digitalisierung und Aufnahme der Messdaten, S. 36), 2.4.8 (Signalverarbeitung, S. 37), 2.4.9 (Anzeigen der berechneten Daten auf dem LCD, S. 39) und 2.4.10 (Serielles Interface, S. 40) näher eingegangen.

Das fünfte Unterkapitel „’MATLAB-GUI“ (siehe 2.5, S. 41) befasst sich mit einer Erklärung des erstellten MATLAB-Programms, deren Elementen und grafischer Oberfläche



(a) Exponentielle Verringerung in einem absorbierenden Medium [5]



(b) Schematische Anordnung der Lichtquellen und des Detektors zur Absorptionsmessung [4]

Abbildung 1: Grundlagen der Absorption am Beispiel des Lambert-Beer'schen-Absorptionsgesetzes

sowie den verfügbaren Bedienungselementen. Im Speziellen wird dabei auf die verschiedenen übertragbaren Signaltypen (siehe 2.5.1, S. 45), das Kommunikationsinterface (siehe 2.5.2) mit den seriellen Einstellungen sowie Sende- und Empfangsroutine, dem Aufbau des Logfiles (siehe 2.5.6, S. 48), die digitale Filterung (siehe 2.5.7, S. 49) und FFT (siehe 2.5.8, S. 49), als auch die Berechnung der Herzfrequenz und der Sauerstoffsättigung (siehe 2.5.9, S. 50) eingegangen.

Im abschließenden Unterkapitel „Versuchsaufbau“ (siehe 2.6) werden die zur Anästhesie verwendeten Geräte und der Messaufbau näher beschrieben und dokumentiert.

2.1 Grundlagen der Pulsoxymetrie

Als Pulsoxymetrie bezeichnet man laut Definition ein „nichtinvasives Verfahren zur kontinuierlichen Messung der Sauerstoffsättigung im arteriellen Blut“ [4], das basierend auf der unterschiedlich starken Absorption von oxygeniertem und nicht-oxygeniertem Hämoglobin (O_2Hb bzw. Hb) bestimmt wird. Dazu wird ein Sensor, bestehend aus zwei LEDs verschiedener Wellenlänge (infrarot und rot) und einem Photodetektor (Photodiode oder Phototransistor), an einem leicht zugänglichen Körperteil appliziert (siehe Abbildung 1, S. 3) und die durchtretenden Lichtintensitäten abwechselnd gemessen. Der Photodetektor wandelt die auftreffende Lichtintensität in ein proportionales Stromsignal um, das in weiterer Folge verwendet wird, um ein proportionales Spannungssignal zu erzeugen. Die vom Gewebe und dem durchtretenden Blut durch Absorption verringerten Lichtintensitäten bzw. das durch Umwandlung erzeugte Absorptionssignal setzen sich dabei

unter der Annahme, dass sich das durchstrahlte Gewebe (Haut, Bindegewebe, Knochen, venöses Blut) in ihrer Position nicht verändert und somit immer dieselbe Absorption herrscht, aus zwei Gruppen zusammen. Einem statischen, zeitlich invarianten Teil, der sich als DC-Anteil ausprägt und die sich zeitlich verändernden Komponenten des pulsierenden arteriellen Blutes, die als AC-Anteil ins Absorptionssignal eingehen [4]. Dieses Pulssignal ermöglicht es, durch Division der beiden mit unterschiedlicher Wellenlänge durchgeführten Messungen auf das Absorptionsverhältnis des arteriellen Blutes und in weiterer Folge auf die Sauerstoffsättigung zu schließen. Die Grundlage dazu liefern die unterschiedlichen Absorptionsspektren der Hämoglobinformen O_2Hb und Hb (siehe Abbildung 2, S. 5) sowie die Berechnungsmethode um die funktionelle Sauerstoffsättigung [1] (siehe Gleichung 1, S. 4) zu bestimmen. Dieser Wert entspricht dem theoretisch vorhandenen Prozentsatz des Oxyhämoglobins im Vergleich zum gesamten funktionellen Hämoglobin [4], [5].

$$SpO_2 = \frac{c_{O2Hb}}{c_{O2Hb} + c_{Hb}} * 100\% \tag{1}$$

In der Praxis erfolgt die Berechnung jedoch nicht, wie in 2.1.3 (S. 8) beschrieben, auf einer analytischen Anwendung des Lambert-Beerschen-Absorptionsgesetzes, sondern auf einem empirisch ermittelten Zusammenhang zwischen dem Absorptionsverhältnis R und der funktionellen Sauerstoffsättigung SpO_2. Diese als Fittingfunktion oder Lookup-Table ausgeführte Berechnungsmethode (siehe 2.1.5, S. 12), bildet die durch die Messmethodik bedingten nicht-linearen Einflüsse ab und verhindert eine falsche Ermittlung der Sauerstoffsättigung.

2.1.1 Limitierende Faktoren und Probleme bei der Durchführung einer Pulsoxymetrie

Bei Verwendung der klassischen Prinzipien der Pulsoxymetrie (siehe 2.1, S. 3) treten limitierende Faktoren und Probleme auf, die die Messung und in weiterer Folge die Berechnung der Sauerstoffsättigung maßgeblich beeinflussen. Diese für das Hardwaredesign relevanten Einschränkungen sind für die Entwicklung und spätere Signaldeutung von großer Wichtigkeit und werden deshalb in den folgenden Abschnitten näher erläutert (im Originaltext [7] sind diese und weitere Einflussfaktoren bei der Anwendung am Menschen ausführlich beschrieben).

[1] Es wird zwischen fraktioneller Sauerstoffsättigung (theoretisch vorhandener Prozentsatz des Oxyhämoglobins in Relation zum gesamten gemessenen Hämoglobin) und der funktionellen Sauerstoffsättigung unterschieden [4].

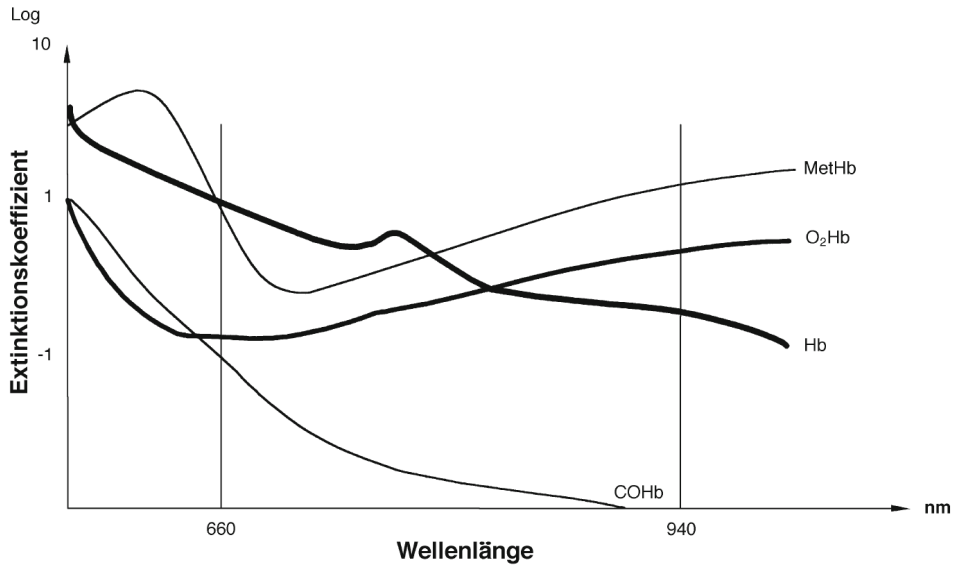


Abbildung 2: Diagramm der Extinktionskoeffizienten von Hämoglobinverbindungen im Messbereich (sichtabres rotes bis infrarotes Licht) [4]

geringes Signal-Rausch-Verhältnis Ein großes Problem bei der Berechnung der Sauerstoffsättigung kann ein zu geringes Signal-Rausch-Verhältnis bedeuten. Ausgelöst von zu geringer Perfusion oder schlechter Sensorplatzierung (geringe Signalamplitude), können eine Sensorbewegung, einfallendes Umgebungslicht, elektromagnetische Einstrahlung und die Messhardware selbst die Rauschamplitude des Messsignals so weit erhöhen, dass eine Berechnung zu falschen Ergebnissen führt. Um dies zu verhindern, sind das Umgebungslicht als auch eine Bewegung auf ein Minimum zu verringern und ungewünschte Frequenzanteile zu entfernen. Darüber hinaus ist auch die Messhardware, bestehend aus OP-Verstärkerstufen, optimal in Bezug auf deren Rauscheigenschaften zu wählen.

Sensorplatzierung Die Sensorplatzierung erweist sich als kritisches Auswahlwahlfaktor für eine möglichst gute Signalqualität. Sie hat an einer Körperstelle zu erfolgen, die möglichst wenig der applizierten Lichtintensität durch unnötige Absorption oder Streuung entfernt und vor allem gut durchblutet ist. Beim Menschen sind dies die Finger, Zehen oder Ohren und bei der Maus der proximale Anteil des Schwanzes oder die Hinterpfoten.

Vasokonstriktion und geringe Perfusion Eine Einschränkung bei Verwendung von körperfernen Körperstellen ergibt sich durch eine einsetzende Gefäßverengung, die eine verringerte Perfusion und damit eine Signalverringerung nach sich zieht. Als Auslöser dafür wird in der Literatur [7] ein geringer arterieller Druck oder im einfachsten Fall eine geringe Temperatur genannt. Speziell bei der Messung an einer sedierten Maus ist darauf zu achten, dass sich die Körpertemperatur nicht zu stark absenkt und dadurch die Perfusion zu stark verringert.

Bewegungsartefakte Durch Bewegung des Sensors können dem Nutzsignal, das sich bei der Maus in einem Frequenzbereich von 3.3 bis 7.5Hz (siehe 2.2, S. 13) befindet, unerwünschte Störungen überlagert werden, die nachträglich nicht mehr zu identifizieren sind und möglicherweise in die Berechnung der Vitalparameter einfließen. Um Bewegungseinflüsse zu vermeiden, sind diese auf ein Minimum zu reduzieren, was durch den Anwendungsbereich des entwickelten Pulsoxymeters - die sedierte Maus - automatisch gegeben ist.

Pulswellenform Die Form der Pulswelle bestimmt durch ihre Amplitude und Frequenzanteile maßgeblich die Berechnung der Sauerstoffsättigung. Eine starke Überlagerung der reflektierten Pulswelle kann dazu führen, dass [8] nicht nur die Grundwelle, sondern auch die reflektierende Welle als Pulssignal gedeutet und zur Berechnung herangezogen wird. Dieser Umstand verschärft sich bei Verwendung einer frequenzbasierten Berechnungsmethodik, wie sie oft in der Praxis eingesetzt wird. Die Zerlegung des Zeitsignals in ihre Frequenzanteile liefert bei einer starken Reflexion als Maximum nicht die Grundwelle, sondern die erste Oberwelle - die Berechnungen zur Herzfrequenz ergeben dadurch die doppelte Rate und die Sauerstoffsättigung wird falsch bestimmt. Dieser Umstand ist nicht zu vermeiden und muss bei der Beurteilung der Messergebnisse berücksichtigt werden.

Umgebungslicht Die Messung der Absorptionsrate wird durch Messung bei zwei Wellenlängen durchgeführt, die auch im natürlichen oder künstlichen Umgebungslicht auftreten. Fällt dieses zu stark aus, überlagert sich das durch den Photodetektor umgesetzte Nutzsignal mit diesem Störsignal und kann ihn im schlimmsten Fall in die Sättigung treiben - eine Signalanalyse wird erschwert oder sogar verhindert. Aus diesem Grund wird durch lichtundurchlässige Abschirmung des Sensors und vor allem des Photodetektors eine Einstrahlung von Störquellen auf ein Minimum verringert. Trotzdem soll zur Stei-

gerung des Signal-Rausch-Verhältnisses das Umgebungslicht auf ein Minimum reduziert werden.

Carboxy- und Methämoglobin Die Verwendung von zwei Wellenlängen ermöglicht es grundsätzlich, zwei verschiedene Hämoglobinformen zu berücksichtigen. Weitere Formen wie das Carboxyhämoglobin (COHb) oder das Methämoglobin (MetHb) sind dabei nicht inkludiert und werden bei der Berechnung der funktionellen Sauerstoffsättigung (siehe Gleichung 1, S. 4) als nicht vorhanden angenommen. Dies führt bei ausgeprägten COHb- oder MetHb-Konzentrationen zu falsch hohen $O_2 -$ Sättigungswerten und kann z.B. bei einer Rauchgasvergiftung oder starken Rauchern zu gefährlichen Schlussfolgerungen führen. Auf den Einsatz an der Maus hat dies nur geringe Auswirkungen, da davon ausgegangen werden kann, dass sich im Mausblut nur geringe Konzentrationen von COHb− oder MetHb befinden.

2.1.2 Physikalische Grundlagen: Das Lambert Beersche Absorptionsgesetz

Die Bestimmung der Sauerstoffsättigung basiert, wie in 2.1 (S. 3) bereits kurz angeführt, auf der Verwendung von Absorptionsunterschieden im pulsierenden arteriellen Blut und im Speziellen deren für die Absorption bestimmenden Anteile: des Oxyhämoglobins und des nicht oxygenierten Hämoglobins. Die physikalische Grundlage liefert dazu eine Anwendung des Lambert-Beerschen Gesetzes (siehe Gleichung 2, S. 7), das die exponentielle Abnahme der Intensität einer monochromatischen Strahlung beim Durchtritt eines homogenen und absorbierenden Volumens mit unterschiedlicher Konzentration beschreibt.

$$I_1 = I_0 e^{-\epsilon(\lambda)cd} \tag{2}$$

Die exponentielle Abnahme vergrößert sich mit der zurückgelegten Distanz d und hängt in weiterer Folge vom Extinktionskoeffizienten $\epsilon(\lambda)$ bei einer festgelegten Wellenlänge λ und der Konzentration c der vom Licht durchtretenden Substanz ab. Unter der Annahme, dass in der Substanz durch Reflexions- oder Streuprozesse keine Strahlung verloren geht - was natürlich nicht erfüllt ist (siehe dazu 2.1.1, S. 4) - bildet die Differenz von einfallender Intensität I_1 und absorbierter Intensität die transmittierte Intensität I_0. Das Lambert-Beersche Gesetz erlaubt durch Anwendung der Superposition die Beschreibung von mehr als einer Substanz im Durchtrittsmedium (siehe Gleichung 3, S. 8) und kann somit verwendet werden, um die Absorption im statischen Gewebe (gekennzeichnet

durch die Tiefstellung DC) als auch von oxygeniertem (HbO2) und nicht oxygeniertem Hämoglobin (Hb) zu beschreiben (siehe Gleichung 4, S. 8).

$$I_1 = I_0 e^{-\epsilon_1(\lambda)c_1d_1} + I_0 e^{-\epsilon_2(\lambda)c_2d_2} + ... + I_0 e^{-\epsilon_n(\lambda)c_nd_n} = I_0 \sum_{i=1}^{n} e^{-\epsilon_i(\lambda)c_id_i} \tag{3}$$

$$I_1 = I_0 \sum_{i=1}^{n} e^{-\epsilon_i(\lambda)c_id_i} = I_0\, e^{-\epsilon_{DC}(\lambda)\, c_{DC}\, d_{DC}}\; e^{-(\epsilon_{Hb}(\lambda)\, c_{Hb}\, d_{Hb} + \epsilon_{HbO2}(\lambda)\, c_{HbO2}\, d_{HbO2})} \tag{4}$$

Unter der Annahme, dass der optische Wegpfad d für oxygeniertes (d_{Hb}) und nicht oxygeniertes Hämoglin (d_{HbO2}) gleich groß ist, kann Gleichung 4 (S. 8) zu Gleichung 5 (S. 8) vereinfacht werden.

$$I_1 = I_0\, e^{-\epsilon_{DC}(\lambda)\, c_{DC}\, d_{DC}}\; e^{-(\epsilon_{Hb}(\lambda)\, c_{Hb} + \epsilon_{HbO2}(\lambda)\, c_{HbO2})\, d} \tag{5}$$

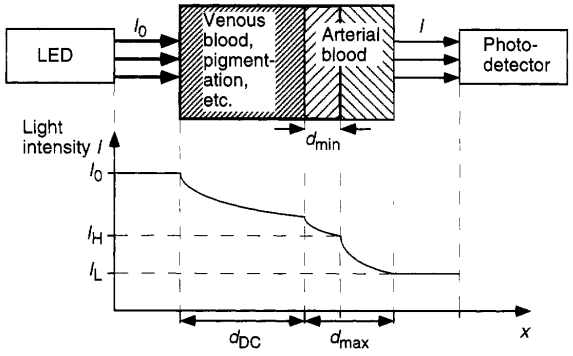
Diese Gleichung wird als Ausgangspunkt verwendet, um durch geschickte Substitution [5] und weitere Annahmen auf die Absorptionsunterschiede des pulsierenden arteriellen Blutes und in weiterer Folge auf die Sauerstoffsättigung zu schließen (siehe dazu 2.1.3, S. 8).

2.1.3 Anwendung der Absorptionsunterschiede zur theoretischen Berechnung der Sauerstoffsättigung

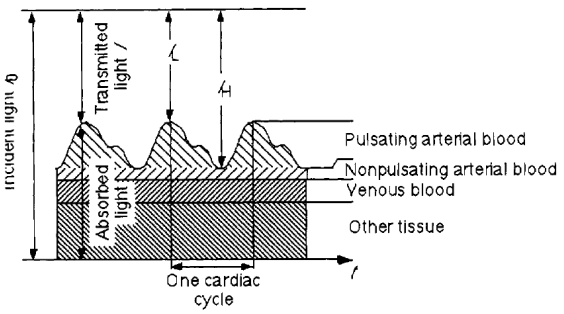
Ausgehend von Gleichung 5 (S. 8) werden für die zwei Extremwerte der Pulswelle Gleichungen aufgestellt und spiegeln die Absorptionswerte zum Zeitpunkt der Systole I_S und der Diastole I_D wieder. Wie in Abbildung 3 (S. 9) dargestellt ist, bildet sich die Pulswelle aufgrund von Änderungen des Gefäßdurchmessers aus. Diese Änderung hat Absorptionsunterschiede zur Folge und kann durch den zeitlich variablen Anteil an transmittierter Strahlung aufgezeichnet werden. Während sich der statische Anteil zu den beiden Zeitpunkten nicht ändert, geht der unterschiedliche Durchmesser der Blutgefäße in den Exponenten des variablen Anteils ein (siehe Gleichungen 6 (S. 8) und 7 (S. 8)).

$$I_S = I_0\, e^{-\epsilon_{DC}(\lambda)\, c_{DC}\, d_{DC}}\; e^{-(\epsilon_{Hb}(\lambda)\, c_{Hb} + \epsilon_{HbO2}(\lambda)\, c_{HbO2})\, d_{max}} \tag{6}$$

$$I_D = I_0\, e^{-\epsilon_{DC}(\lambda)\, c_{DC}\, d_{DC}}\; e^{-(\epsilon_{Hb}(\lambda)\, c_{Hb} + \epsilon_{HbO2}(\lambda)\, c_{HbO2})\, d_{min}} \tag{7}$$



(a) Exponentielle Verringerung in einem absorbierenden, gut durchbluteten Gewebe [5]



(b) Zeitliche Veränderung der transmittierten Lichtintensität in einem durchbluteten biologischen Gewebe [5]

Abbildung 3: Darstellung der Absorptionsunterschiede zur Berechnung der theoretischen Sauerstoffsättigung

Durch Umformen und Gleichsetzen kürzt sich der statische Anteil (Subscript DC) und die Substitution von $\Delta d = d_{max} - d_{min}$ ergibt folgenden Ausdruck:

$$I_S = I_D \, e^{-(\epsilon_{Hb}(\lambda)\, c_{Hb} + \epsilon_{HbO2}(\lambda)\, c_{HbO2})\, \Delta d} \tag{8}$$

Diese Gleichung (8, S. 9) wird durch die Intensität der Diastole (I_D) normalisiert und ermöglicht es bei der Messung mit zwei Wellenlängen, das in der Literatur als Ratio bezeichnete Verhältnis R zwischen diesen Wellenlängen aufzustellen.

$$R = \frac{ln(I_{S,R}/I_{D,R})}{ln(I_{S,IR}/I_{D,IR})} = \frac{(\epsilon_{Hb}(\lambda_R)\, c_{Hb} \; + \; \epsilon_{HbO2}(\lambda_R)\, c_{HbO2})\, \Delta d_R}{(\epsilon_{Hb}(\lambda_{IR})\, c_{Hb} \; + \; \epsilon_{HbO2}(\lambda_{IR})\, c_{HbO2})\, \Delta d_{IR}} \tag{9}$$

Durch Einsetzen von Gleichung 1 (S. 4) für die Hämoglobinkonzentrationen und der Annahme, dass sich die optischen Weglängen bei der Messung der beiden Wellenlängen nicht unterscheiden, ist es durch weiteres Umformen möglich, die Gleichung 9 (S. 9) von den Konzentrationen c_{HbO2} und c_{Hb} zu befreien.

$$R = \frac{\epsilon_{Hb}(\lambda_R) \; + \; [\epsilon_{HbO2}(\lambda_R) \; - \; \epsilon_{Hb}(\lambda_R)] \; SpO_2}{\epsilon_{Hb}(\lambda_{IR}) \; + \; [\epsilon_{HbO2}(\lambda_{IR}) \; - \; \epsilon_{Hb}(\lambda_{IR})] \; SpO_2} \tag{10}$$

Ein Auflösen dieser Gleichung 10 (S. 9) nach dem Term SpO_2 ermöglicht die analytische Berechnung der Sauerstoffsättigung durch Berechnung des Verhältnisses R aus den Messdaten der beiden Wellenlängen. Dazu sind aus den Messdaten die durch den Gleichanteil normalisierten Absorptionsmaxima zu bestimmen und durch Einsetzen der

Extinktionskoeffizienten kann die Sauerstoffsättigung berechnet werden.

$$SpO_2 = \frac{\epsilon_{Hb}(\lambda_R) \; - \; \epsilon_{Hb}(\lambda_{IR}) \; R}{\epsilon_{Hb}(\lambda_R) \; - \; \epsilon_{HbO_2}(\lambda_R) \; + \; [\epsilon_{HbO2}(\lambda_{IR}) \; - \; \epsilon_{Hb}(\lambda_{IR})] \; R} \tag{11}$$

2.1.4 Annahmen zum Lambert Beerschen Absorptionsgesetz

Der Gültigkeitsbereich und damit die korrekte Verwendung des Lambert-Beerschen Gesetzes ist an bestimmte Voraussetzungen gebunden, die beim Einsatz der Pulsoxymetrie nur ansatzweise erfüllt sind. Dies hat zur Folge, dass sich die Sauerstoffsättigung nicht analytisch mit der im Unterkapitel 2.1.3 (S. 8) hergeleiteten Gleichung 11 (S. 10) berechnen lässt, sondern eine empirisch ermittelte Angleichung durch eine Fitting-Kurve erfolgen muss (siehe Abbildung 4, S. 11 und Unterkapitel 2.1.5, S. 12). In den folgenden Unterpunkten wird näher auf diese eingegangen und deren Auswirkungen beschrieben.

Homogene Verteilung der absorbierenden Substanz Eine Voraussetzung des Lambert-Beerschen Gesetzes ist die Anforderung der homogenen Verteilung der absorbierenden Substanz. Das durchstrahlte Medium besteht aber aus vielen unterschiedlichen stark inhomogen verteilten Gewebstypen, wie Haut, Muskel, Knochen und im Besonderen das Blut selbst. Um diesen Umstand zu kompensieren, wird zur Auswertung nur der dynamische Absorptionsanteil, der durch das pulsierende Blut verursacht wird, verwendet und ausgewertet. Da sich die absorbierenden Substanzen (Oxyhämoglobin und Deoxyhämoglobin) im Blut aber nicht homogen verteilen, sondern konzentriert in den Erythrozyten auftreten, sind die gemessenen Absorptionssignale erst ab einer ausreichenden Anzahl an durchstrahlten Blutkörperchen auswertbar. Besonders im Fall der Mäuse muss beachtet werden, dass diese über ein sehr kleines Blutvolumen verfügt (siehe 2.2, S. 13) - und die dabei messbaren Absorptionsunterschiede um ein vielfaches kleiner sind als z. B. beim Menschen.

Monochromatisches Licht Durch die Verwendung von zwei LEDs zur Erzeugung der Messstrahlung wird eine weitere Voraussetzung des Lambert-Beerschen Gesetzes verletzt. Bei der durch Elektronen ausgelösten Lichtquantenemission in der LED wird ein Spektrum von ±15nm erzeugt. Aufgrund von Fertigungsunterschieden kann sich zudem der Maximalwert um ±15nm verschieben [5]. Dies hat bei der analytischen Berechnung der Sauerstoffsättigung zur Folge, dass die Emissionskoeffizienten an das Maximum angepasst werden müssen (wird über eine Widerstandscodierung realisiert, die Informatio-

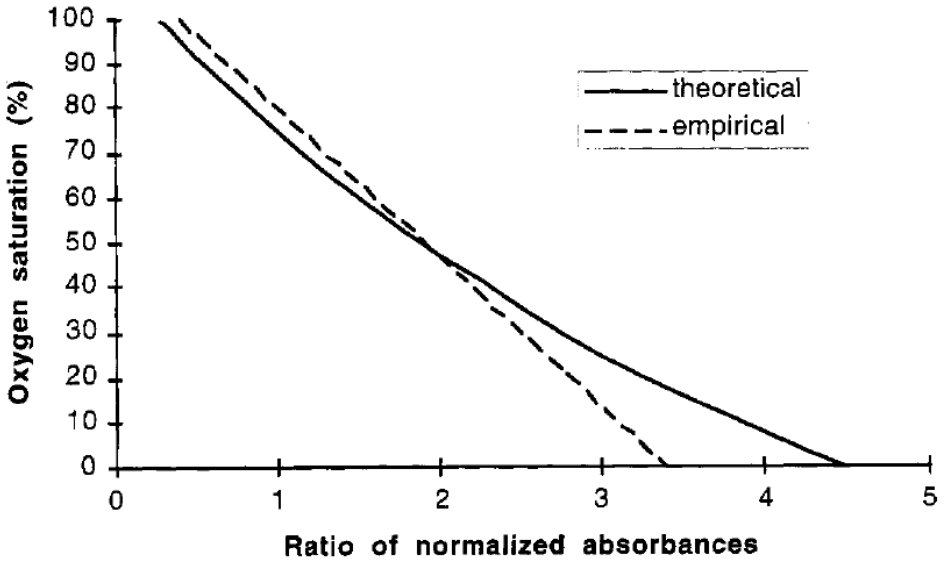


Abbildung 4: Zusammenhang zwischen theoretischer und praktischer Berechnungsmethodik zur Ermittlung der Sauerstoffsättigung [5]

nen über das Emissionsmaximum der im Sensor eingebauten LEDs bereitstellt). Trotz dieser Maßnahme verursacht das nicht monochromatische Licht eine Verschleifung der Extinktionskoeffizienten und erzeugt dadurch eine Ungenauigkeit der Berechnung.

Lichtstreuung Die durchstrahlten Gewebstypen verursachen zusätzlich zur Absorption eine Primär- und Mehrfachstreuung, die sich in einer Erhöhung der Absorptionsrate äußert [5]. Während dies für das statische Gewebe vernachlässigbar ist, werden durch Form und Bewegung der Erythrozyten - je nach ihrer räumlichen Lage - die Transmissionseigenschaften maßgeblich verändert. Vor allem die unterschiedliche Ausrichtung der Erythrozyten während Systole (parallel zum Blutfluss) und Diastole (normal zum Blutfluss) in Kombination mit den geometrischen Abmessungen führt zu unterschiedlich starkem Streuverhalten. Dies führt bei analytischer Berechnung nach der Theorie des Lambert-Beerschen Gesetzes zu teilweise falschen Resultaten (siehe Abbildung 4, S. 11). Um die durch eine analytische Berechnung der Sauerstoffsättigung entstehenden Fehler zu minimieren, wird diese nicht analytisch, sondern über eine Fitting-Kurve bestimmt.

2.1.5 Anwendung der Absorptionsunterschiede zur praktischen Berechnung der Sauerstoffsättigung

Aufgrund der unter 2.1.4 (S. 10) beschriebenen Einschränkungen zur Verwendung des Lambert-Beerschen Absorptionsgesetzes, kann die Berechnung nicht direkt über die analytisch hergeleitete Gleichung 11 (S. 10) erfolgen. Diese Problematik wird in vorhandenen Systemen durch empirisch erstellte Kalibrationskurven oder Look-Up-Tables gelöst und ermöglicht es, trotz der nicht-linearer Einflüsse der Messapparatur und des Messverfahrens die Sauerstoffsättigung basierend auf dem Verhältnis R ausreichend genau zu bestimmen [4], [5]. Dieses Ratio (siehe Gleichung 12, S. 12, [9]) wird durch Division der beiden mittels ihrem Gleichanteil normalisierten Wechselanteile bestimmt und spiegelt die Absorptionsunterschiede des pulsierenden arteriellen Bluts bei den gemessenen Wellenlängen wieder.

$$R \approx \frac{Ivr_{AC}/Ivr_{DC}}{Iir_{AC}/Iir_{DC}} = \frac{Ivr_{AC}/Iir_{AC}}{Ivr_{DC}/Iir_{DC}} \overset{Ivr_{DC} \equiv Iir_{DC}}{\cong} \frac{Ivr_{AC}}{Iir_{AC}} \tag{12}$$

Die dazu notwendigen Intensitätswerte $\mathrm{Ivr_{DC}}$, $\mathrm{Ivr_{AC}}$, $\mathrm{Iir_{DC}}$ und $\mathrm{Iir_{AC}}$ können entweder direkt aus den zugehörigen Absorptionssignalen oder durch FFT-Berechnung aus dem Amplitudenspektrum entnommen werden. Weisen die beiden Absorptionssignale durch schaltungstechnische Normierung denselben DC-Anteil auf, beinhaltet die Berechnung des Ratios lediglich die Amplitudenwerte des pulsierenden AC-Signals. Über das berechnete Verhältnis ist es durch Einsatz einer empirisch ermittelten Gleichung, die Nichtlinearitäten der Messungsapparatur berücksichtigt, möglich, die tatsächliche funktionelle Sauerstoffsättigung ausreichend genau zu berechnen [5] [6]. Dabei werden in der Literatur verschiedene Fitting-Funktionen beschrieben, die in den Gleichungen 13, 14 und 15 (S. 12) angeführt sind [5]. Die in diesen Fittingfunktionen vorkommenden Konstanten k sind für jeden Gerätetyp unterschiedlich und werden in einem aufwendigen Validierungsverfahren experimentell ermittelt. Dazu misst man von Blutproben mit unterschiedlicher aber bekannter Sauerstoffsättigung das Absorptionsverhältnis R und erstellt einen Kurvenfit oder eine Look-Up-Table.

$$SpO_2 = \frac{k_1 - k_2 R}{k_3 - k_4 R} \tag{13}$$

$$SpO_2 = k_1 + k_2 R + k_3 R^2 \tag{14}$$

$$SpO_2 = 110 - 25R \tag{15}$$

Die Qualität des erstellten Fits trägt maßgeblich zur Genauigkeit des Pulsoxymeters bei und ist von den meisten Herstellern mit einer maximalen Abweichung (±SD) von ±3% angegeben [10].

2.2 Designvoraussetzungen zur Anwendung der Pulsoxymetrie an Mäusen

Um die Prinzipien der Pulsoxymetrie auf die Maus anzuwenden, sind neben den unter 2.1.1 (S. 4) beschriebenen limitierenden Faktoren auch die physiologischen und anatomischen Unterschiede zum Menschen zu beachten. Aufgrund des sehr geringen Körpergewichts von durchschnittlich ca. $20 * 10^{-3}$kg bis $40 * 10^{-3}$kg (je nach Rasse und im ausgewachsenen Zustand [11]) ergibt sich, im Vergleich zum Menschen, ein stark verringertes Blutvolumen [11], [12] (ungefähr um den Faktor $2 * 10^{-3}$). Durch die geringe Körpergröße bzw. das kleine Körpergewicht der Maus und bedingt durch das proportional kleinere Herz ([15]), verfügt die Maus jedoch über eine weitaus höhere Herzfrequenz als der Mensch (siehe Tabelle 1, S. 13).

Tabelle 1: Gegenüberstellung der für die Anwendung einer Pulsoxymetrie relevanten physiologischen Paramter der Maus und des Menschen [11], [12], [13] und [14]

Parameter	Einheit	Maus	Mensch
Atemrate	min^{-1}	106-230 [4]	12 [4]
Herzrate in Ruhe	min^{-1}	450-550 [2]	40 - 80 [3]
Blutvolumen	ml	2-2.75 [1]	4500 - 6000 [3]
Herzminutenvolumen in Ruhe	$\frac{ml}{min}$	15 [4]	ca. 5600 [3]

Das Blutvolumen bildet in Kombination mit dem Herzminutenvolumen (Cardiac Output) und der Herzfrequenz die Grundlage des zu messenden pulsierenden arteriellen Bluts. Deren durchschnittliche Werte sind in Tabelle 1 (S. 13) übersichtlich zusammengefasst und bestimmen maßgeblich die Bandbreite des zu erwartenden Pulssignals (ca. 7.5 bis 9Hz) sowie die Bandbreite des Störsignals Atmung (1.8 bis 3.8Hz). Diese Frequenzbänder beziehen sich auf physiologische Parameter, die den Ruhezustand beschreiben. Die unter Narkose herrschenden Parameter für die Herzfrequenz von anästhesierten Mäusen liegen je nach Literaturquelle (siehe [13], [16]) unter den in Tabelle 1 (S. 13)

[1] [11]
[2] [13]
[3] [12]
[4] [14]

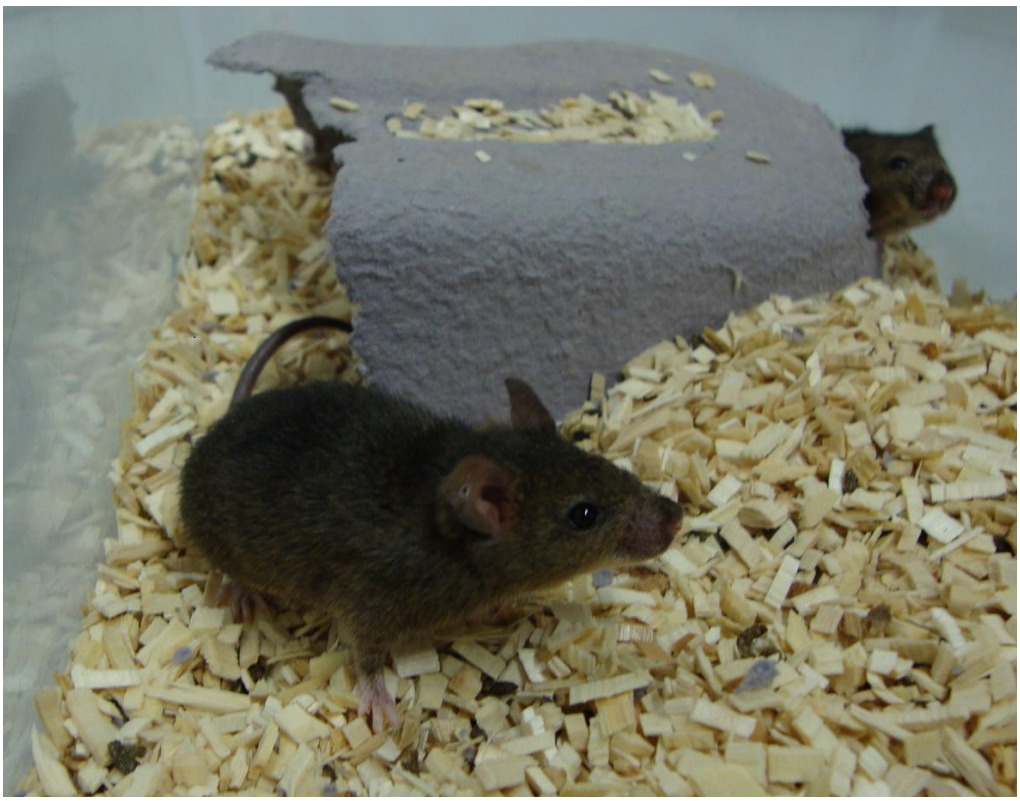


Abbildung 5: Versuchsmaus in Ihrem Käfig

angegebenen Kenngrößen und kann bis zu einem Wert von 200min^{-1} absinken. Auch die Atemrate kann sich bis auf 80min^{-1} verringern ([16]). Da das Pulsoxymeter nur an sedierten Mäusen eingesetzt werden soll, wird das Augenmerk auf die dabei auftretenden Frequenzbereiche gelenkt und die in Tabelle 1 (S. 13) vermerkten Parameter fungieren als obere Grenze der Signalbandbreiten, die in der Tabelle 2 (S. 14) vermerkt sind.

Tabelle 2: Frequenzbereiche der im Absorptionssignal vorkommenden Signaltypen

Frequenzbereich des Signaltyps	Einheit	sedierte Maus
Pulssignal	Hz	3.3-7.5
Atmung	Hz	1.3-3.8

Neben den physiologischen Parametern ist auch die Anatomie der Maus beim Hardware- und vor allem dem Sensordesign zu beachten (siehe 2.3.3, S. 17). Aufgrund der geringen Körperabmessungen einer durchschnittlichen adulten Maus ist es nicht möglich, den Sensor an Finger oder Zehen zu applizieren. Um ein möglichst gutes Signal-Rausch-Verhältnis zu erreichen, muss eine alternative Körperstelle zudem gut durchblutet und frei zugänglich sein. Starke Behaarung und Pigmentierung der Haut reduzieren die mög-

lichen Einsatzorte zur Sensorbefestigung auf die Vorder- und Hinterbeine sowie den proximalen Anteil des Schwanzes.

2.3 Hardware

Dieses Kapitel befasst sich mit einer Beschreibung der zur Hardwareentwicklung und deren Elemente verwendeten Methoden, die es durch geschicktes Schaltungsdesign ermöglichen, die Prinzipien der Pulsoxymetrie auf die sedierte Maus anzuwenden und aus den gemessenen Absorptionssignalen die Herzrate sowie die Sauerstoffsättigung zu berechnen.

2.3.1 Hardwareentwicklung

Die Hardwareentwicklung erfolgt durch den Aufbau eines ersten Schaltungsentwurfs auf einer Steckplatine, der sukzessive verbessert und weiterentwickelt wird. Über die Software „Eagle 6.2.0 Light“ (CadSoft Computer GmbH, Pleiskirchen, Deutschland) wird der am Steckbrett ausführlich getestete Schaltungsentwurf in einen Stromlaufplan übertragen, dessen Version 2 im Anhang A unter 4.6 (S. 84) zu entnehmen ist. Dieser Stromlaufplan wird durch die Software Eagle in ein Printlayout übertragen, das nach erfolgreichem Routing auf zwei Ebenen zur Platinenherstellung verwendet wird (siehe Anhang A unter Layout). Das dazu verwendete Messprinzip wird im folgenden Unterkapitel 2.3.2 (S. 15) am Beispiel des Blockschaltplans und dessen Elemente näher erläutert.

2.3.2 Schaltungsbeschreibung anhand des Blockschaltplans

Das Schaltungsprinzip ist in Abbildung 6 (S. 16) durch einen Blockschaltplan dargestellt und beschreibt in grafischer Form, wie die Messdaten gewonnen werden. Das zentrale Element stellt der Mikroprozessor des Entwicklungsboards „mbed NXP LPC11U24“ (NXP Semiconductors, Eindhoven Niederlande) (siehe 2.3.4, S. 19) dar, der das zur Signalaufzeichnung notwendige Element „LED-Regelung“ durch eine PWM ansteuert und über die Pulsweite den LED-Strom und die dadurch proportional erzeugte Lichtintensität der beiden LEDs regelt. Um den Empfang der zwei Lichtintensitäten unterschiedlicher Wellenlänge über einen Photodetektor zu ermöglichen, sind die antiparallel verschalteten Sendedioden abwechselnd zu betreiben. Dazu werden die dem LED-Strom und von einem TP geglätteten proportionalen PWM-Signale von der LED-Treiberstufe (einem modifi-

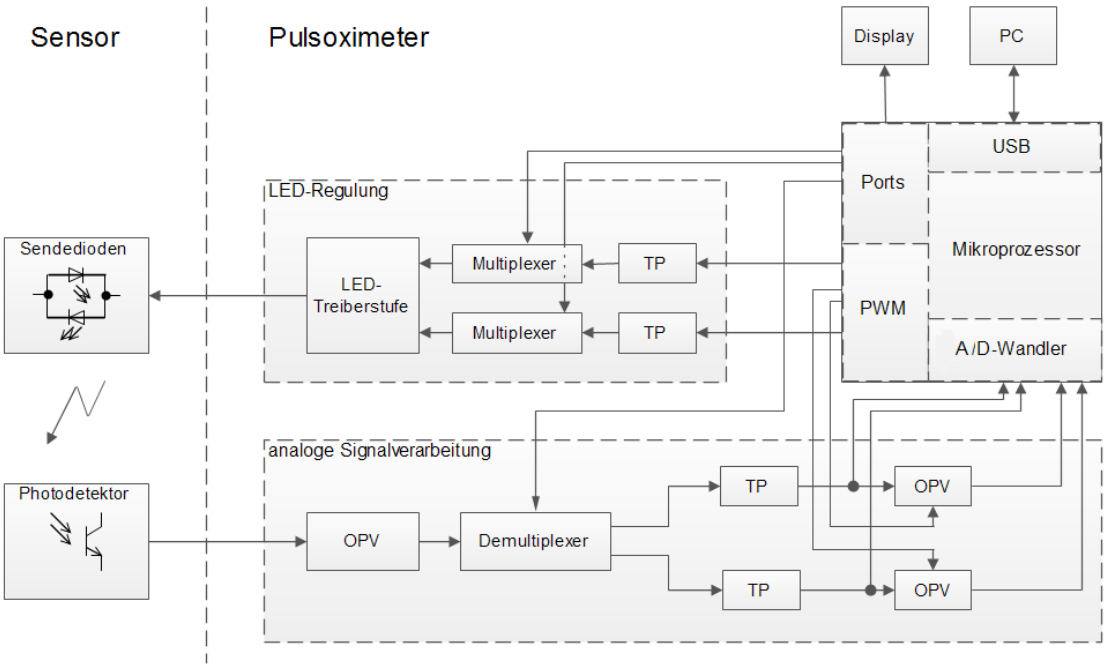


Abbildung 6: Blockschaltplan der entwickelten Messhardware

zierten Vierquadrantensteller) in ein im Vorzeichen alternierendes Stromsignal umgewandelt, das die LEDs abwechselnd leuchten lässt. Über den Photodetektor wird die einfallende Strahlung in einen Strom umgewandelt und von einem Transimpedanzwandler (OPV) in ein proportionales Spannungssignal umgesetzt. Dieses Spannungssignal ist, bedingt durch die abwechselnde Ansteuerung der Sendedioden, in gleicher Form „codiert“ und wird von einem Demultiplexer in die beiden Signalbestandteile der jeweiligen Wellenlänge aufgetrennt. Die dabei entstehenden pulsförmigen Signale werden von zwei TP demoduliert und entsprechen den gemessenen Absorptionssignalen. Um die spätere Berechnung der Sauerstoffsättigung zu vereinfachen (siehe 2.1.5, S. 12 und Gleichung 12, S. 12), werden über den im μP eingebauten A/D-Wandler diese Spannungswerte digitalisiert und für die LED-Regelung ausgewertet. Durch Erhöhung bzw. Verringerung der PWM-Pulsbreite werden die Ströme für die zwei sich im Sensor verbauten LEDs solange angepasst, bis sich die zwei vom Photodetektor erzeugten und von einem OPV in ein Spannungssignal umgewandelten Amplituden im selben Wertebereich befinden. In zwei weiteren OPV-Stufen werden nur noch die Wechselanteile verstärkt und von einem aktiven Filter bandbegrenzt. Dies wird erreicht, in dem zwei PWM-Signale von Tiefpässen in proportionale Gleichspannungssignale, die dem DC-Anteil der Absorptionssignale entsprechen, umgewandelt und als Offsetspannung an die positiven OPV-Eingänge ge-

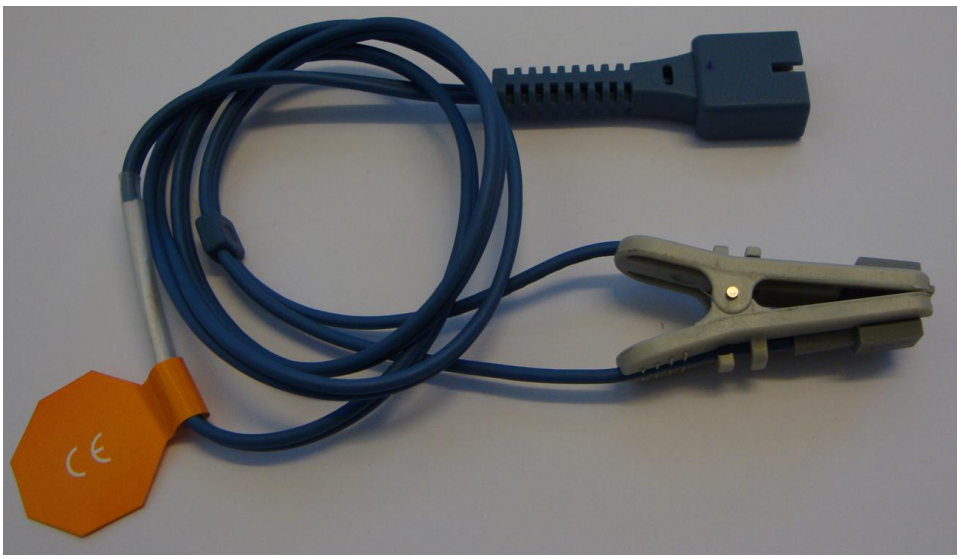


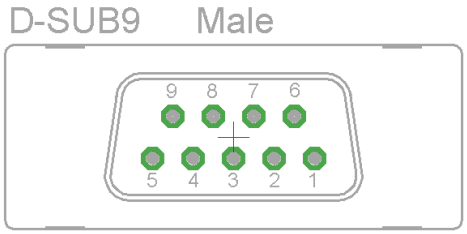
Abbildung 7: Nachbau des Sensors DS100A (China PanSW Med, Yuexiu, China)

schaltet werden. Die verstärkten und bandbegrenzten AC-Absorptionssignale werden vom μP digitalisiert und digital weiterverarbeitet (siehe dazu 2.4, S. 29). Nach erfolgter Berechnung der Herzfrequenz und Sauerstoffsättigung werden diese in periodischen Abständen an einem LCD ausgegeben und können vom Bediener bequem abgelesen werden. Ein virtuelles serielles Interface ermöglicht es zudem, die im μP vorhandenen Daten über die USB-Schnittstelle an einen PC zu übertragen.

2.3.3 Sensordesign

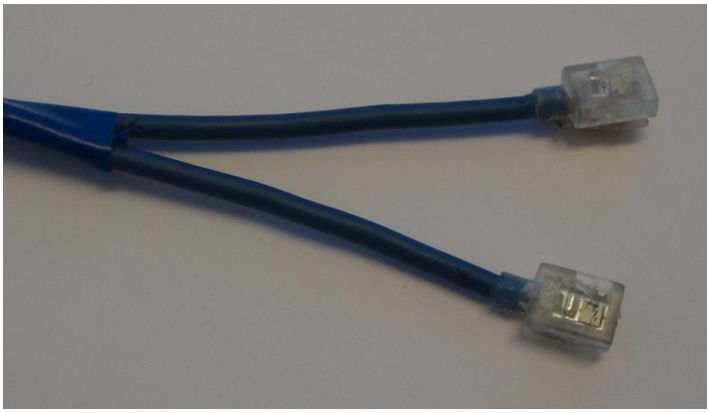
Zur Messung der Absorptionsunterschiede des pulsierenden arteriellen Mausblutes bestehen, bis auf die unter 2.2 (S. 13) angeführten physiologischen Eigenschaften, keine weiteren Unterschiede zu einer Messung am Menschen. Aus diesem Grund und um die Entwicklung der gesamten Hardware zu beschleunigen, wird als Ausgangspunkt ein Nachbau des weitverbreiteten Sensors DS100A (Nellcor, Hayward, USA) verwendet. Dieser ist in seiner Originalform in Abbildung 7 (S. 17) dargestellt. Er besteht im Wesentlichen aus zwei Schaltungselementen, den Sendedioden und der Empfangsdiode, die über einen DSUB-9 Stecker (male) angesteuert werden können. Aufgrund fehlender technischer Beschreibung wurde die Pinbelegung als auch die Spezifikationen der verwendeten Sendedioden über andere Quellen und weiterführende Internetrecherche bestimmt [17], [18], [20] und [21]. Als Wellenlänge werden für den verwendeten Sensortyp 660nm (sichtbares rotes Licht) und 905nm (infrarot) angeben (so wie alle Nellcor-Sensoren).

Tabelle 3: Pinbelegung der D-Sub-9 Schnittstelle des Sensors

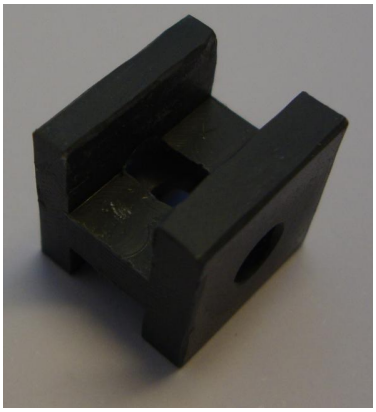


Pinnummer	Beschreibung
1	keine Verbindung (NC)
2	Anode der IR-LED, Kathode der roten LED
3	Kathode der IR-LED, Anode der roten LED
4	keine Verbindung (NC)
5	Anode des Phototransistors
6	keine Verbindung (NC)
7	Kupferschirmung, Masse
8	keine Verbindung (NC)
9	Kathode des Phototransistors

Die vorherrschende Pinbelegung des DSUB-9-Anschlusssteckers ist in Tabelle 3 (S. 18) zusammengefasst.



(a)



(b)

Abbildung 8: Darstellung des zerlegten Sensors DS100A (a) und der entwickelten Sensorhalterung (Seitenabmessung 14 mm, b)

In die vorhandenen Schlitze (einer befindet sich auf der Unterseite und ist daher nur ansatzweise zu sehen) werden die beiden Kunststoffpads mit den Sendedioden bzw. dem Phototransisor des demontierten Sensors (siehe Abbildung 8, S. 18) eingeschoben und mit lichtundurchlässigem Isolierband fixiert sowie optisch abgeschirmt. In die Bohrung,

deren Abmessung aus Erfahrungswerten resultiert und die zwischen den Sensoreinschüben verläuft, können die Maushinterbeine oder der Mausschwanz zur Messung eingeführt werden. Der durch eine neue Sensorhalterung modifizierte Pulsoxymetriesensor ist in Abbildung 9 (S. 19) dargestellt und kann für die Messung an der Maus verwendet werden.

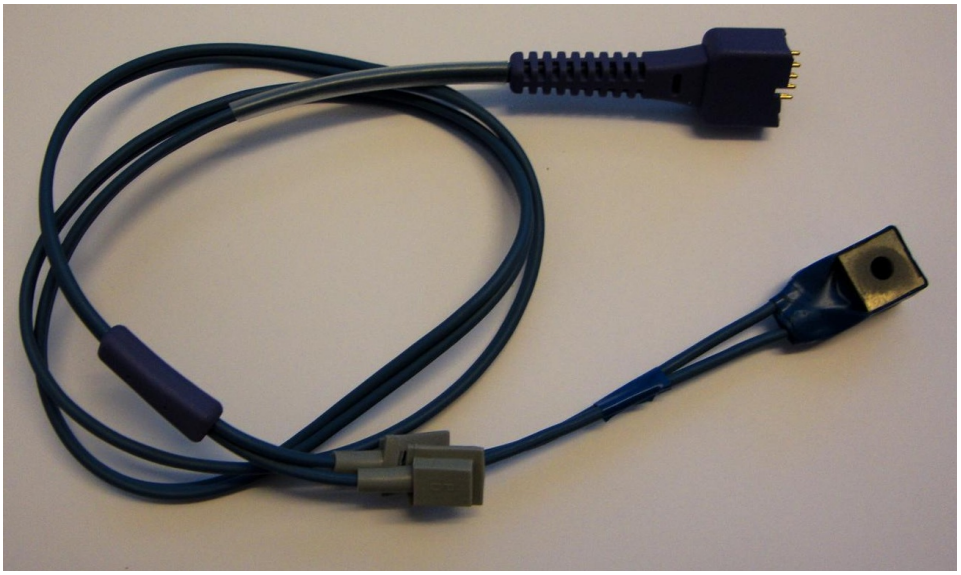


Abbildung 9: Modifizierter Sensor mit der entwickelten Sensorhalterung

2.3.4 Mikroprozessor-Entwicklungsboard

Die Prinzipien der Pulsoxymetrie (siehe 2.1, S. 3) erfordern, um erfolgreich die Sauerstoffsättigung bestimmen zu können, neben der Signalerzeugung, die von einem externen Sensor durchgeführt wird (siehe 2.3.3, S. 17), eine Signalregelung, Signalaufzeichnung und digitale Signalverarbeitung. Dazu ist es notwendig, einen modernen und leistungsstarken Mikroprozessor einzusetzen. Um den Schaltungsaufbau so kompakt als möglich gestalten zu können, soll der μP zusätzlich über mehrere eingebaute Analog-Digital-Wandler, PWM-Ausgänge und eine einfach zu implementierende Schnittstelle zu einem PC verfügen. Nach eingehender Analyse des μP-Markts und einem Vergleich der angebotenen Features zum Verkaufspreis, wird das μP-Entwicklungsboard „mbed NXP LPC1768“ (NXP Semiconductors, Eindhoven Niederlande) ausgewählt. Dieses Entwicklungsboard basiert auf dem μP „LPC1768“ und ermöglicht es über die USB-Verbindung per Drag-and-Drop-Aktion kompilierte μP-Programme zu übertragen und dadurch zu „flashen“. Dieses Merkmal vereinfacht die Hardwareentwicklung immens und erlaubt es,

Tabelle 4: Übersicht der technisch relevanten Spezifikation des μP-Entwicklungsboards „mbed NXP LPC1768“ (NXP Semiconductors, Eindhoven Niederlande)[23]

Parameter	Beschreibung
Prozessorkern	32-bit ARM Cortex-M3
Taktung	96 MHz
RAM	32 KB
Flash-Speicher	512 KB
Versorgungsspannung	über den USB-Anschluss oder extern (4.5V - 9.0V)
relevante Ausstattung	6xPWM 6xADC

das über zwei Steckleisten im 40pin DIP-Format ausgeführte Entwicklungsboard direkt in die Schaltung zu integrieren. Die Programmierung der μP-Software erfolgt dabei über den im Lieferumfang inbegriffenen Zugang zu einem Onlinecompiler in der Programmiersprache C++. Des Weiteren stellt der Hersteller NXP eine vorkompilierte Library zur Verfügung, die zu allen im μP verfügbaren Features vordefinierte Funktionen enthält, welche für die Softwareerstellung verwendet werden können.

Wie Tabelle 4 (S. 20) zu entnehmen ist, kann die Spannungsversorgung bequem über ein USB-Kabel erfolgen, das in weiterer Folge zur Verfügung steht, um über ein serielles Interface die Signaldaten an einen PC zu übertragen. In Abbildung 10 (S. 21) sind die Pinbelegung und die verfügbaren Funktionen übersichtlich dargestellt und zusammengefasst.

Zur digitalen Signalaufzeichnung sind insgesamt sechs ADC-Eingänge (p15 - p20) mit einer maximalen Auflösung von 12bit [24] vorhanden, von denen vier benötigt und verwendet werden. Der bei der Digitalisierung limitierende Faktor Konversionsrate beträgt 200kHz [24] und ermöglicht eine ausreichende zeitliche Auflösung. Die Signalregelung erfolgt über vier der sechs verfügbaren PWM-Ausgänge (p21 - p26) mit einer maximalen Zyklusfrequenz von 1MHz (10^{-6}s). Neben dem 96MHz getakteten Mikroprozessor ermöglicht der 32KB-RAM eine aufwendige Signalverarbeitung sowie den Einsatz von digitalen Filtern um Störfrequenzen zu entfernen und der Berechnung einer FFT zur späteren Sauerstoffsättigungsermittlung (siehe dazu 2.4, S. 29). Der zeitliche Messablauf wird durch freie I/O-Ports realisiert, die durch Änderung des logischen Pegels (low = 0V und high = 3.3V) einen Multiplexer und Demultiplexer ansteuern. Nach erfolgter μP interner Berechnung der Herzfrequenz und der Sauerstoffsättigung, werden diese über weitere I/O-Ports an ein LCD übertragen und dargestellt.

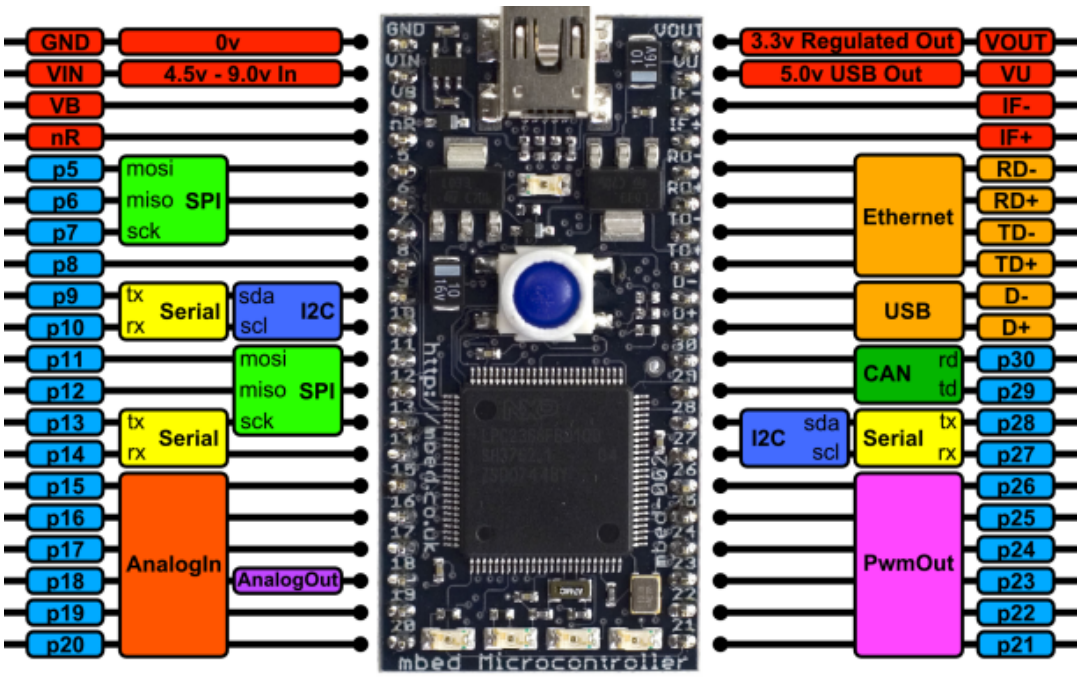


Abbildung 10: Übersicht der Pinbelegung des μP-Entwicklungsboards „mbed NXP LPC1768“ (NXP Semiconductors, Eindhoven Niederlande) [23]

2.3.5 Zeitmanagement

Die antiparallele Verschaltung der beiden Sendedioden sowie das Vorhandensein von nur einem Phototransistor zur Detektion zweier Lichtintensitäten verschiedener Wellenlängen, erfordert eine gut überlegte Ansteuerung. In der Literatur werden dazu zwei grundlegend verschiedene Ansätze beschrieben [5], [17], [18] und [19]. Zum Einen kann eine Modulation des Sendediodenstroms mit unterscheidbarer Frequenz erfolgen, die es ermöglicht, durch Demodulation aus dem vom Phototransistor umgesetzten Absorptionssignal die Intensitätssignale der zwei Wellenlängen zu bestimmen und zum Anderen kann dies über ein sequenzielles Multiplexing und Demultiplexing erfolgen. Die Tatsache, dass die Sendedioden im Sensor antiparallel verschalten vorliegen, ermöglicht mit geeigneter Treiberstufe eine kompakte, digital geregelte sequenzielle Ansteuerung. Um jedoch eine Signalerzeugung und Rückgewinnung mittels Frequenzmodulation bei diesem Sensortyp durchzuführen, ist dies ungleich aufwendiger, weshalb bereits seit dem frühen Entwicklungsstadium auf die Verwendung einer sequentiellen Ansteuerung gesetzt wird.

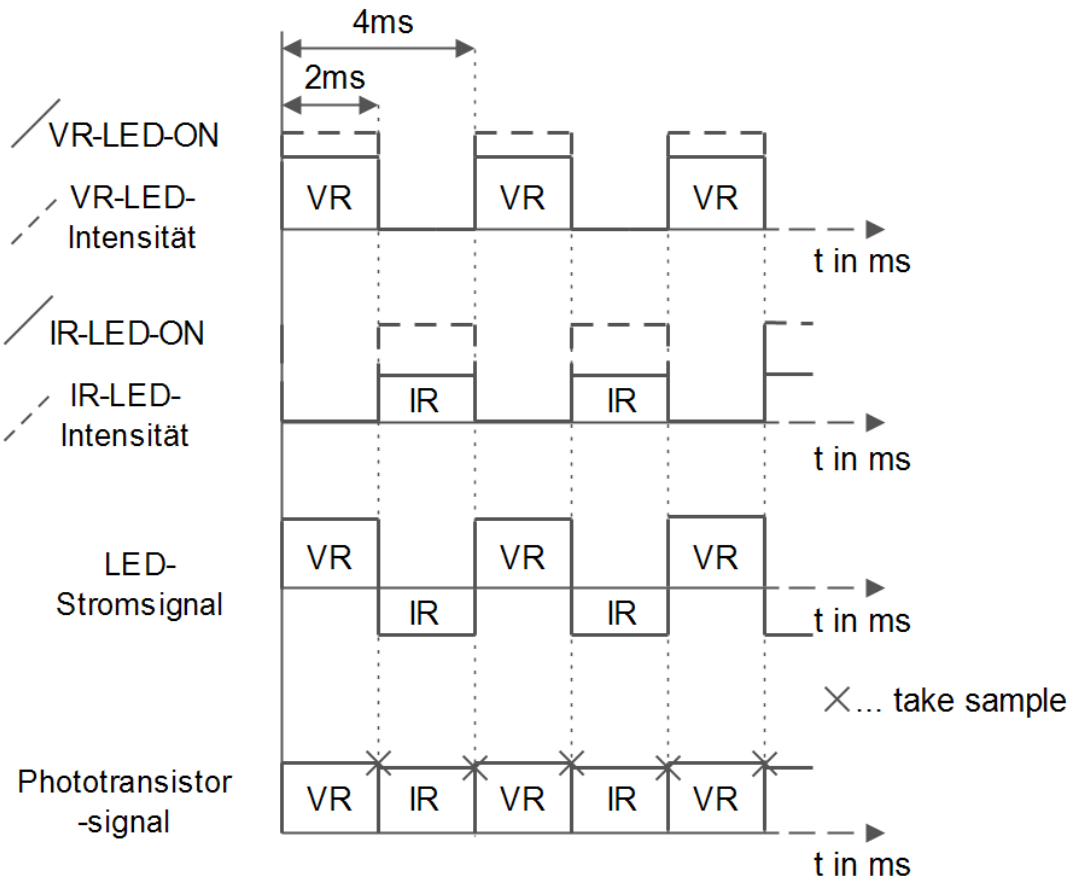
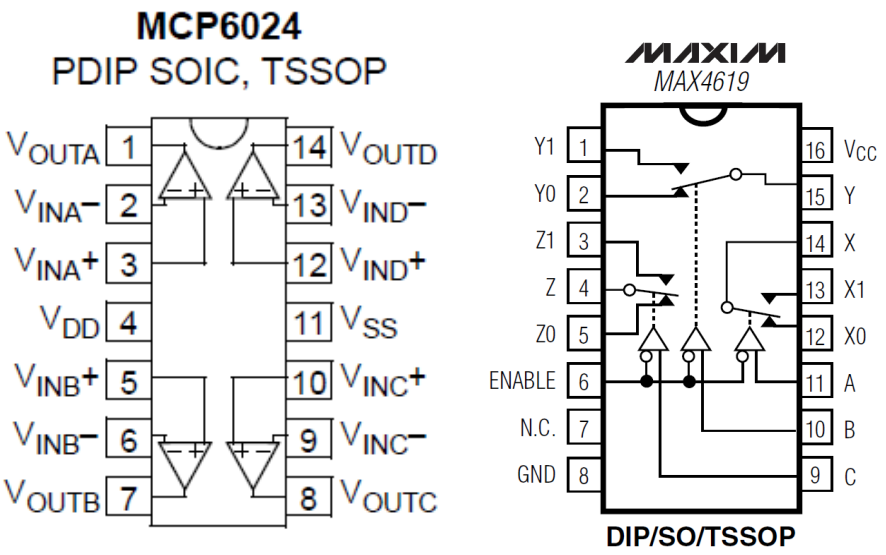


Abbildung 11: Schemenhafte Darstellung des Zeitprofils zur sequentiellen Ansteuerung der Sendedioden

Die Verwendung einer sequenziellen Ansteuerung der Sendedioden bringt den Vorteil, dass sich die dazu notwendige Hardwareschaltung (im Blockschaltplan unter Abbildung 6 (S. 16) als „LED-Regelung“ markiert und unter 2.3.6 (S. 24) näher beschrieben) auf drei Schaltungselemente beschränkt. Das Timing der Multiplexer und die LED-Intensitäten im Sendepfad sowie das Timing des Demultiplexers werden vom Mikrokontroller nach dem in Abbildung 2.3.5 (S. 21) abgebildeten Zeitplan geregelt und dienen der Signalakquise zu einer vorgegebenen Abtastrate. Deren Wahl ist ein Trade-off zwischen der Leistungsfähigkeit des μPs, der die Intensitätsregelung und Signalverarbeitung über digitale Filter zeitgleich durchführt, des verfügbaren RAMs, da die Frequenzauflösung der FFT-Berechnung proportional zur Anzahl der Signalsamples (Fenstergröße) zunimmt und der Genauigkeit der Signalberechnung, die auf der Annahme beruht, dass die beiden Wellenlängen zeitgleich aufgenommen werden. Um all diese Anforderungen bestmöglich zu erfüllen und deren negative Auswirkungen zu minimieren, wird die Abtastrate für die



(a) Pinbelegung des MCP6024 [26] (b) Pinbelegung des MAX4619 [25]

Abbildung 12: Pinbelegung der verwendeten IC-Bausteine

beiden Signale nach mehrmaligen Änderungen mit 250Hz festgesetzt (siehe auch 4.1.1, S. 64).

Diese Samplerate bestimmt die Länge eines Zyklus des in Abbildung 11 (S. 22) dargestellten Zeitprofils zur Digitalisierung von einem Sample der IR-Absorption und einem Sample der VR-Absorption. Dieser Zeitzyklus wird von der Abtastrate (250Hz) bestimmt und beträgt $4 * 10^{-3}$s. Zu diesem Zwecke werden in $2 * 10^{-3}$s-Abständen per Multiplexer die Signalintensitäten der jeweiligen LED sequenziell codiert und über einen modifizierten Vierquadrantensteller in ein proportionales, aber im Vorzeichen alternierendes Stromsignal umgewandelt (siehe 2.3.6). Dieses aktiviert abwechselnd die beiden LEDs und sorgt dafür, dass der Phototransistor ein Signal umsetzt, in dem abwechselnd die Absorptionsinformationen von IR- und VR-Licht auftreten (siehe Abbildung 11, S. 22). Dieses Signal wird von einem Demultiplexer, der ebenfalls durch den Mikrokontroller gesteuert wird, in ihre Bestandteile aufgetrennt, gefiltert, verstärkt und digitalisiert. Der Zeitaufbau ist dabei nur von der Variable Samplezeit, die sich aus der Samplerate ergibt, abhängig und kann zentral im μP eingestellt werden. Das zentrale Element zur Erstellung einer sequentiellen Signalkodierung ist der als Multiplexer im Sendepfad und Demultiplexer im Empfangspad betriebene Umschalterbaustein MAX4619 (Maxim Integrated Products, Sunnyvale, USA) [25]. Dessen Pinbelegung ist in Abbildung 12 (S. 23) dargestellt.

2.3.6 LED-Treiberstufe

Die LED-Treiberstufe soll es dem μP ermöglichen, die LED-Intensität über ein PWM-Signal zu regeln. Dazu werden die PWM-Signale, deren Tastgrad (duty cycle) in 100 Schritten durch die Software verändert werden kann und über eine Zyklusfrequenz von f = 10000Hz verfügt, von jeweils einem Tiefpass mit einer Grenzfrequenz von $f_g = 6.4$Hz geglättet und durch einen Impedanzwandler von seinem, bedingt durch die Filterung, hochohmigen Widerstand befreit und über den Umschalterbaustein MAX4619 (Maxim Integrated Products, Sunnyvale, USA) [25], der als sequenzieller Multiplexer fungiert, in ein gepulstes Signal umgewandelt. Dieser Vorgang hat für beide PWM-Signale gesondert zu erfolgen und ermöglicht eine unabhängige Regelung der beiden LED-Intensitäten. Der zeitliche Ablauf basiert auf dem in Abbildung 2.3.5 (S. 21) dargestellten Zeitprofil und wird durch I/O-Ports des μPs gesteuert.

Die Erzeugung des für die antiparallel verschalteten Sendedioden des Empfängers notwendigen Stromprofils erfolgt durch die Verwendung eines modifizierten Vierquadrantenstellers. Dessen Aufbau als auch die Funktionen sind schemenhaft in Abbildung 13 (S. 25) dargestellt und ermöglichen die Umwandlung einer Gleichspannung bzw. deren Amplitude in ein im Vorzeichen alternierendes proportionales Stromsignal. Dazu wird die von vier Transistoren (zwei PNP- und zwei NPN-Typen) symmetrisch aufgebaute Brückenschaltung von zwei I/O-Ports des μPs zur Signalauswahl und den beiden Gleichspannungssignalen zur Intensitätsregelung (siehe Abbildung 13, S. 25) angesteuert. Dabei ergeben sich zwei Betriebsarten, die von der Basis der PNP-Transistoren D1 und D3 durch Anlegen einer Spannung von 0V oder VCC = 3.3V ausgewählt werden. Über die NPN-Transistoren D2 und D4 erfolgt die Einstellung der gewünschten, vom Transistor über den Emitterwiderstand erzeugten Stromstärke. Durch abwechselnde Ansteuerung kann die Stromrichtung des Brückenzweigs in dem sich die antiparallel verschalteten LEDs des Sensors befinden, als auch dessen Intensität unabhängig und digital vom μP eingestellt werden. Dazu bilden die Transistoren D1 und D4 als auch die Transistoren D2 und D3 zwei Gruppen, die abwechselnd in Sperr- bzw. Durchlass und Verstärkungsmodus betrieben werden. Jener PNP-Transistor (D1 oder D3), dessen Basis mit einer Spannung von 0V betrieben wird, leitet, während der gegenüberliegende Transistor mit einer Basisspannung VCC sperrt. Gleichzeitig wird der zur Gruppe gehörende, sich im unteren Teil der Schaltung befindende NPN-Transistor (D4 oder D2) im Verstärkungsmodus betrieben, während der gegenüberliegende Transistor durch Anlegen von 0V sperrt. Um dieses Prinzip grafisch zu visualisieren, sind die beiden Betriebsmodi

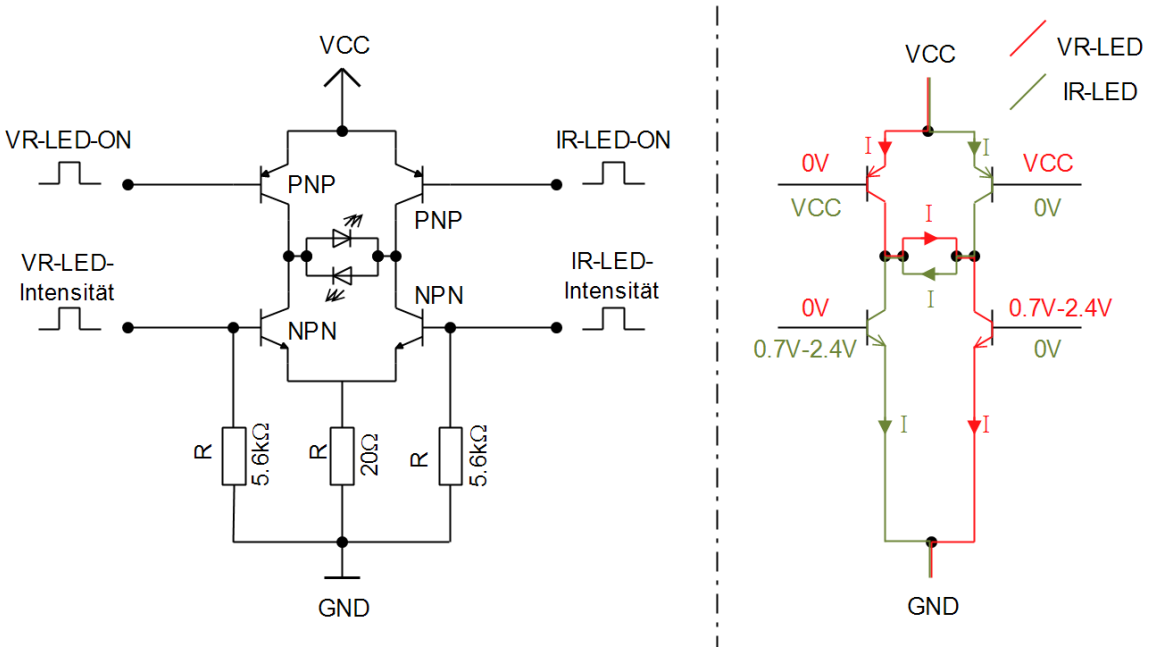


Abbildung 13: Schematische Darstellung der LED-Treiberstufe (links) und ihrer Wirkungsweise (rechts) zur Erstellung der LED-Ströme

in Abbildung 13 (S. 25) grafisch und in der Tabelle 5 (S. 25) in Textform dargestellt und ausgeführt.

Tabelle 5: Übersicht der zwei Betriebsmodi zur Regelung und Erzeugung der LED-Ströme

	Transistor	Wertebereich
Betriebsmodus 1: IR-LED-ON	D1	0V
	D2	0.66V bis 1.8V
	D3	VCC
	D4	0V
	Transistor	Wertebereich
Betriebsmodus 2: VR-LED-ON	D1	VCC
	D2	0V
	D3	0V
	D4	0.66V bis 1.8V

Die Regelung der Stromstärke erfolgt in einem Wertebereich von ca. 0.66V bis 1.8V der sich aus der Versorgungsspannung VCC = 3.3V und den Spannungsabfällen an den eingesetzten Halbleiterelementen (0.66V für den NPN- und PNP-Transistor) und einer Begrenzung auf den annähernd linearen Verstärkungsbereich des NPN-Transistors er-

gibt. Diese Spannung bzw. der zugehörige Strom wird von Transistor D2 oder D3 über den Basiswiderstand verstärkt und in ein proportionales Stromsignal umgewandelt. Die Verstärkungsdimensionierung ist auf einen zeitlichen Mittelwert der LED-Ströme von ca. 20mA ausgelegt, der sich über die zeitliche Länge der Einschalt- und Ausgangvorgänge sowie der Stromamplitude selbst ergibt. Der durch Verwendung des 20Ω Widerstands mögliche Strombereich ist über die μP-Software auf den annähernd linearen Bereich begrenzt und erstreckt sich von 2mA bei 0.66V bis zu einem Maximalwert von $43mA$ bei 1.8V. Über das gepulste Auswahlsignal an den Transistoren D1 und D3 wird abwechselnd ein Strom mit unterschiedlichem Vorzeichen (besonders gut in Abbildung 13 (S. 25) zu sehen) erzeugt, der über einen D-Sub 9 Stecker durch die sich im Sensor befindlichen Sendedioden fließt und dabei eine zur im μP geregelten Spannung proportionale Strahlungsintensität nach dem Zeitprofil aus 2.3.5 (S. 21) erzeugt.

2.3.7 Verstärkerstufen

Das von der LED-Treiberstufe erzeugte Stromsignal führt zu abwechselnd ein- und ausgeschalteten Sende-LEDs im Sensor, deren erzeugte Lichtintensitäten das Messmedium durchstrahlen und, durch Absorption verringert, auf den im Sensor verbauten Phototransistor treffen. Dieser wandelt die Lichtstrahlung in ein proportionales Stromsignal um, das für die Auswertung der im Medium herrschenden Absorptionseigenschaften herangezogen werden kann. Zur besseren Weiterverarbeitung wird dieses Stromsignal über einen Transimpedanzwandler in ein Spannungssignal umgewandelt und von einem Multiplexer in seine zwei Bestandteile aufgetrennt. In zwei verschiedenen Signalpfaden erfolgt eine Filterung und mittels jeweils einer weiteren OPV-Stufe zur besseren Signalauflösung eine erneute Verstärkung des zu messenden AC-Anteils. Das dadurch erzeugte Spannungssignal, dass das arterielle Pulssignal beinhaltet, wird vom Mikroprozessor und dessen integriertem A/D-Wandler digitalisiert und zur Berechnung der Herzfrequenz und Sauerstoffsättigung herangezogen.

In Abbildung 14 (S. 27) ist der prinzipielle Aufbau des Empfangspfads abgebildet und soll grafisch die Signalgewinnung und Aufbereitung wiedergeben. Dazu werden der Emitter und Kollektor des Phototransistors über den D-SUB-9-Stecker (Pinbelegung siehe Tabelle 3, S. 18) des Sensors in Form einer Emitterschaltung mit der Hardware des Empfangspfads verbunden und das durch Lichteinstrahlung erzeugte Stromsignal über einen Transimpedanzwandler in ein Spannungssignal umgewandelt. Das zu erwartende sehr geringe Stromsignal bedarf spezieller Anforderungen an das Schaltungsdesign sowie an die eingesetzten OPV. Zusammenfassend kann an dieser Stelle erwähnt wer-

Tabelle 6: Übersicht der technischen Leistungsmerkmale des OPV-ICs „MCP6024“ (Microchip Technology, Chandler, USA)[26]

Parameter	Beschreibung
Verstärkungsbreich	Rail-to-Rail
Spannungsversorgung	unipolar: $2.5\text{V} - 5.5\text{V}$
Bandbreite	$10 * 10^6\text{Hz}$
Rauschspannung	$8.7 * 10^{-9} \frac{\text{V}}{\sqrt{\text{Hz}}}$, bei $10 * 10^3\text{Hz}$
Offsetspannung	$\pm 500 * 10^{-6}\text{V}$
Bias-Strom	$1 * 10^{-12}\text{A}$
Offsetstrom	$\pm 1 * 10^{-12}\text{A}$
Slewrate	$7 \frac{\text{V}}{10^{-6}\text{s}}$

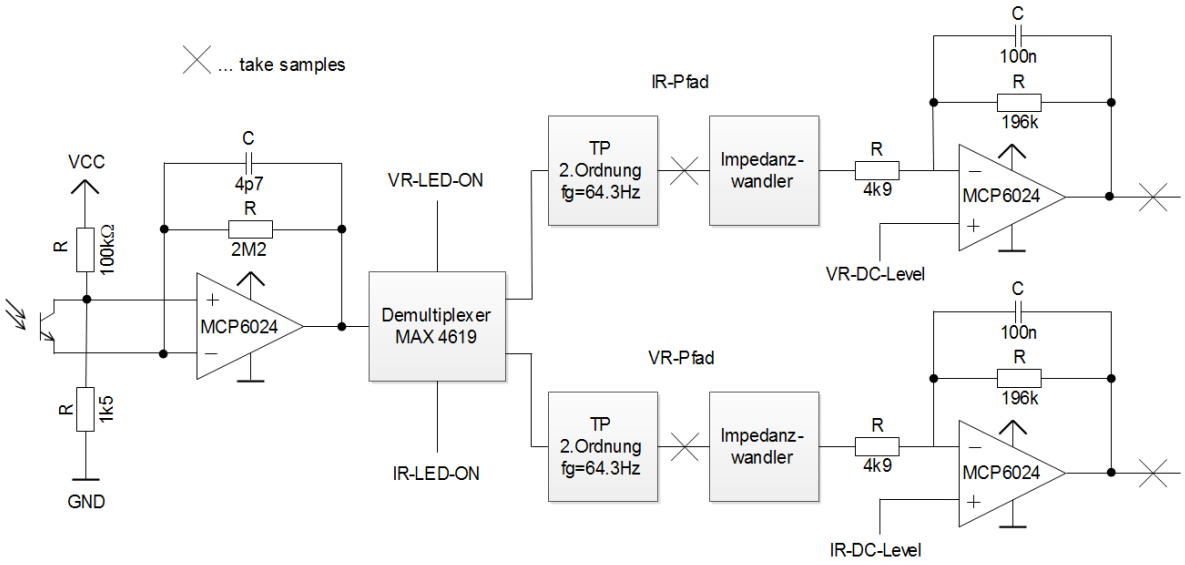


Abbildung 14: Schematischer Stromlaufplan zur Beschreibung der Wirkungsweise des Empfnagspfads und ihrer Verstärkerstufen

den, dass die Auswahl und Dimensionierung der ersten Verstärkerstufe sehr stark die Signalqualität beeinflusst (siehe dazu 4.1.3, S. 66) und entscheidend ist, ob das Signal in weiterer Form eine Berechnung der Sauerstoffsättigung zulässt. Zur Anwendung kommt der OPV-IC „MCP6024“ (Microchip Technology, Chandler, USA) der sich durch die in Tabelle 6 (S. 27) angemerkten Eigenschaften auszeichnet und unter anderem für den Einsatz in Transimpedanzwandler entwickelt wurde.

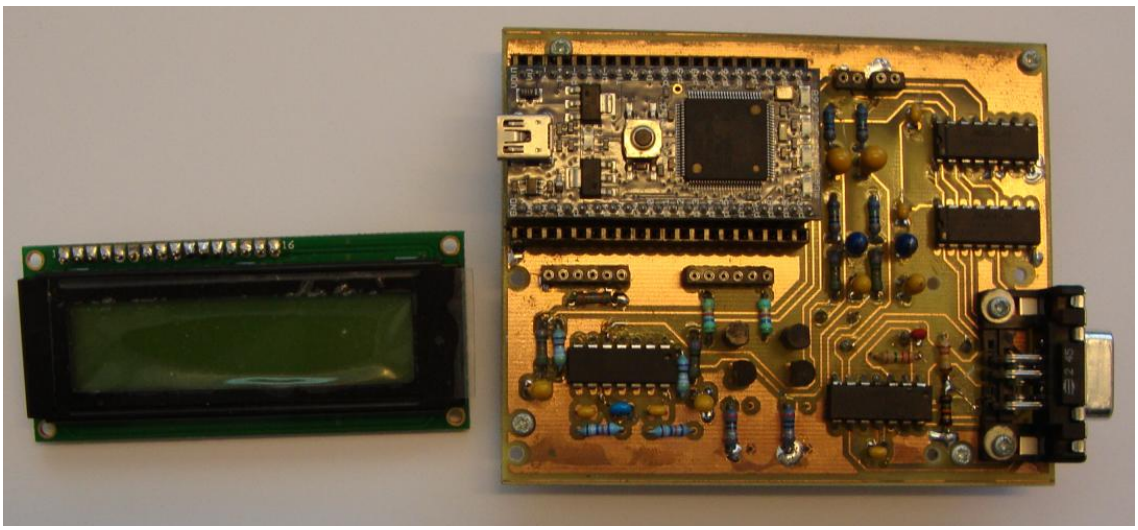


Abbildung 15: Bestückte und betriebsfertige Platine

2.3.8 Sonstige Schaltungselemente

Neben den unter 2.3.6 (S. 24) und 2.3.7 (S. 26) näher erläuterten Schaltungselementen, verfügt die entwickelte Hardware, über eine 16X2 Flüssigkristallanzeige um die berechneten Ergebnisse der Herzfrequenz und Sauerstoffsättigung dem Benutzer bequem darstellen zu können. Diese wird direkt vom Mikrokontroller über die I/O-Ports (p5 für RS, p6 für Enable sowie p12, p13, p14 und p15 für die Datenports D4 bis D8) angesteuert und über die 5V Spannungsversorgung der USB-Verbindung versorgt. Das dazu notwendige Ansteuerungsprotokoll sowie deren Erscheinungsform ist im Kapitel Software unter 2.4.9 näher ausgeführt und beschrieben.

Die geforderte Datenkommunikation (siehe 1.1, S. 1) wird über die als Versorgungsspannung bereits verwendete USB-Verbindung zu einem PC sichergestellt. Dazu wird genutzt, dass sich der μP bei der Verbindung mit einem PC als virtueller COM-Port initialisiert und über die μP-Software ein serielles Interface aufgebaut werden kann. Deren Einstellungen und Eigenschaften werden in 2.4.10 (S. 40) ausführlich behandelt und beschrieben.

2.3.9 Bestückte und betriebsfertige Hardware

Die gefertigte Platine wird mit den in Anhang A unter Stückliste angeführten Bauteilen und Elementen bestückt (siehe Abbildung 15, S. 28) und kann zur Messung verwendet werden. Um einen Blick auf alle Schaltungselemente zu ermöglichen, ist in Abbil-

dung 15 (S. 28) das 16x2 LCD abgenommen und befindet sich links neben der Platine. Gut erkennbar sind das sich im linken oberen Bereich der Platine befindliche μP-Entwicklungsboard und dessen USB-Stecker, über den die Hardware durch Anschluss an einen PC versorgt wird und der D-SUB-9 Stecker (female), der sich im rechten unteren Bereich der Platine befindet und über den der modifizierte Sensor angeschlossen wird.

2.4 Mikroprozessor-Software

Dieses Unterkapitel widmet sich der Beschreibung und Darstellung der zur Entwicklung der Mikroprozessorsoftware eingesetzten Methoden und erläutert deren Eigenschaften, die notwendig sind, um die Erzeugung eines sauberen, der Absorption des pulsierenden arteriellen Blutes proportionales Spannungssignal softwaretechnisch zu regeln, darauf basierend eine Berechnung der gewünschten Vitalparameter Herzfrequenz und Sauerstoffsättigung durchzuführen und diese dem Benutzer über ein LCD anzuzeigen oder per virtueller serieller Schnittstelle an einen PC zu übertragen.
Die Programmierung des Mikrokontrollers erfolgt dabei in der Programmiersprache C/C++, die es - gegenüber dem maschinennahen Assembler - ermöglicht, bequem und leicht nachvollziehbar die gewünschten Funktionen zu implementieren. Als Entwicklungs- und Programmierumgebung wird der für das verwendete μP-Entwicklungsboard „mbed NXP LPC11U24“ (NXP Semiconductors, Eindhoven Niederlande) (siehe 2.3.4, S. 19) optimierte Onlinecompiler verwendet, dessen Umgang und Eigenschaften in 2.4.1 (S. 30) näher ausgeführt sind. Die grundlegenden Funktionsinteraktionen sowie der prinzipielle Programmablauf sind im Unterpunkt 2.4.3 (S. 32) durch Abbildung 18 (S. 33) dargestellt und soll eine übersichtliche Programmzusammenfassung darstellen, deren Elemente in den darauf folgenden Unterpunkten (2.4.5, S. 35 bis 2.4.10, S. 40) näher beschrieben werden. Der gesamte Original-μP-Quellcode ist dem Anhang B als „Programmlisting der μP-Software“ zu entnehmen.
An diesem Punkt ist anzumerken, dass einige Funktionsteile des Quellcodes auf Überlegungen anderer Entwickler basieren und für den Einsatz in dieser Software modifiziert zum Einsatz kommen. Diese Programmstellen sind im Quellcode durch den Funktionsheader oder an den jeweiligen Stellen durch Kommentare deutlich gekennzeichnet und identifizierbar.

2.4.1 Entwicklungsumgebung

Im Lieferumfang des eingesetzten μP-Entwicklungsboards ist ein Webzugang zum firmeneigenen Onlinecompiler inbegriffen, der zum Softwareentwurf und zur -entwicklung eingesetzt wird. Es handelt sich hierbei um eine auf den Mikrokontrollertyp „LPC11U24“ (NXP Semiconductors, Eindhoven Niederlande) optimierte C/C++ IDE, die es plattformunabhängig ermöglicht, über einen Webbrowser die Software zu erstellen, kompilieren zu lassen und sich eine $*$.bin Datei herunterzuladen. Diese Datei kann per Drag-and-Drop-Aktion über das als Massenspeicher am PC angemeldete Mikrokontrollerboard an den Flash-Speicher übertragen werden und wird nach einem Restart des μP automatisch ausgeführt. Der Start dieser Entwicklungsumgebung erfolgt durch Eingabe von „https://mbed.org/compiler/“ in die Adressleiste des unterstützten Browsers (z.B. Google Chrome (Google Inc., Mountain View, USA)) und Anmeldung mit dem erstellten Benutzerkonto. Die Bedienung ist in einem Tutorial im „Handbook“ [27] gut erklärt und genügt ähnlichen Kriterien, die von „Offline“-Compilern bekannt sind. Durch Anlegen eines neuen Programms kann die dabei erstellte cpp-Datei den Anforderungen angepasst, kompiliert und die bin-Datei heruntergeladen werden. In Abbildung 16 (S. 31) ist ein Screenshot des mbed-Onlinecompilers dargestellt, der die wichtigsten Compilerlemente visualisiert. Die IDE nutzt dabei den ARM-C-Compiler zur Erstellung des ausführbaren Maschinencodes und bindet dabei automatisch die vorkompilierte Mikroprozessor-SDK des Herstellers NXP ein. Dieses Software Development Kit erlaubt es durch einfachen C/C++ Aufruf auf vordefinierte Funktionen, wie die Generierung eines digitalen Samples über den A/D-Wandler (`read_u16`), zuzugreifen und erleichtert dadurch die Entwicklungsarbeit. Diese vorkompilierten Funktionen sind übersichtlich im so genannten „Handbook“ [27] zugänglich und werden verwendet, um Mikrokontroller-Funktionen auszuführen.
Des Weiteren kann über den Onlinezugang auf die von anderen mbed-Benutzern erstellten und freiwillig veröffentlichten Libraries und Projekte zugegriffen werden. Diese für eine spezielle Anwendung entworfenen Libraries zeigen die universalen Anwendungsgebiete des Mikrokontrollerboards und werden unentgeltlich zur freien Verfügung gestellt. Im vorliegenden Fall wird die vom Benutzer „Simon Ford“ erstellte Library „TextLCD“ [28] verwendet, um das zur Ausgabe der Berechnungsergebnisse eingesetzte 16x2 LCD bequem über eine Lokalisierungs- und Ausgabefunktion anzusteuern (siehe 2.4.9, S. 39).

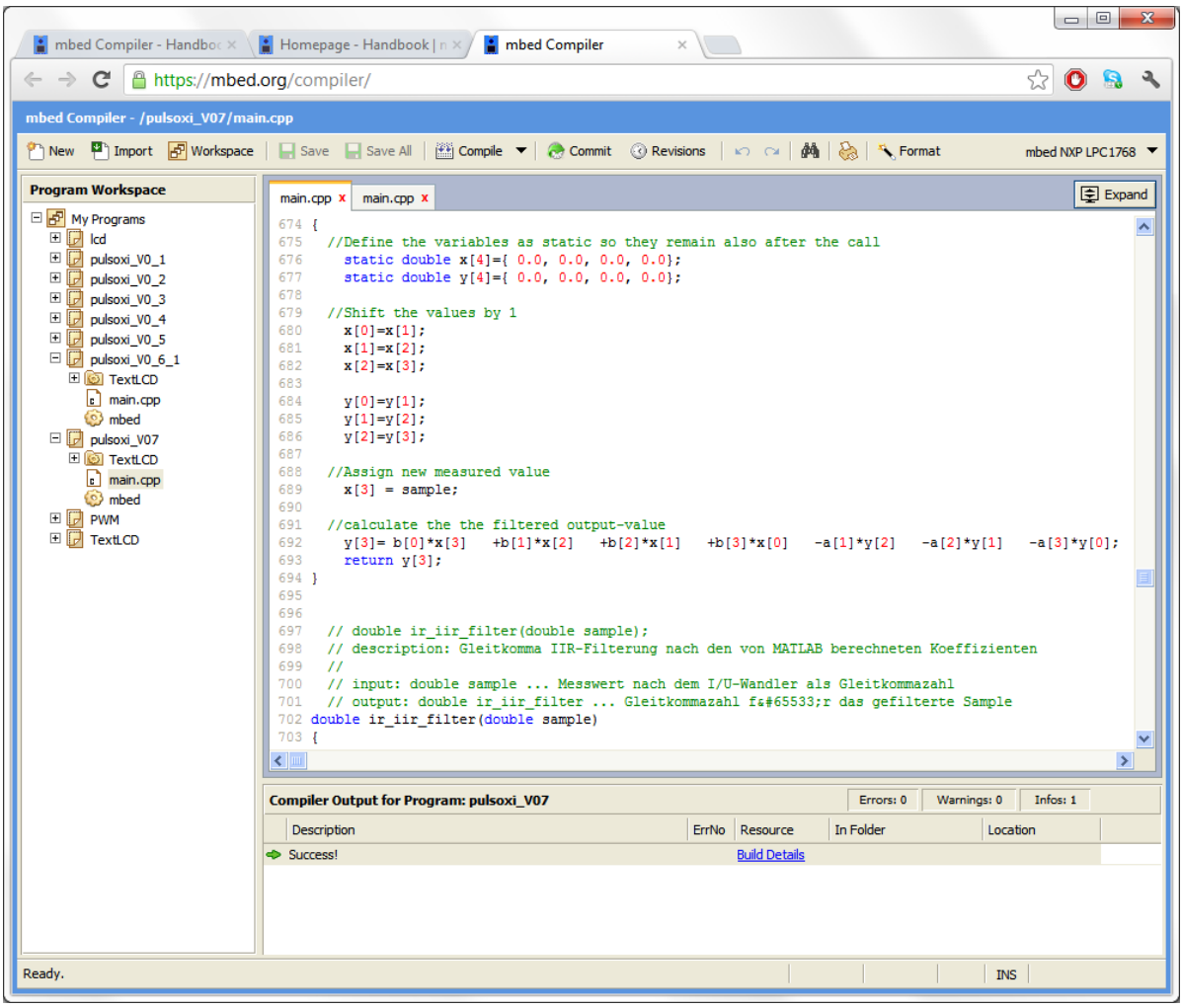


Abbildung 16: mbed Entwicklungsumgebung

2.4.2 Speicherbedarf und Management

Das über die Onlineentwicklungsumgebung erstellte Mikroprozessorprogramm umfasst in der cpp-Datei 966 Zeilen (siehe Anhang B) und führt, gemeinsam mit den inkludierten Libraries, nach der Kompilierung zu einer bin-Datei mit einer Größe von 33 kB. Diese Datei wird per Drag-and-Drop Aktion and den Flash-Speicher des Mikrokontrollers übertragen und nimmt dort etwa 6.4% des verfügbaren 512kB Speichers ein. Die zur Signalregelung und Parameterberechnung benötigten Variablen werden im RAM allokiert und belegen mit 25kB ca. 79% des verfügbaren Speichers. Diese Gesamtgröße limitiert die Anzahl der Samples zur FFT-Berechnung auf eine maximale Anzahl von 2048 float-Datenpunkte ($4\text{Byte} * 2048 * 2\text{Signaltypen} = 16\text{kB}$) und damit die im Code festgelegte Fenstergröße. In Abbildung 17 (S. 32) sind diese Build-Informationen nach

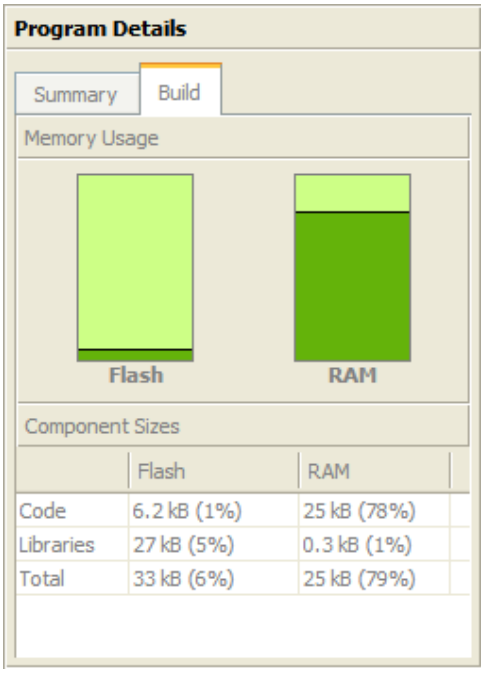


Abbildung 17: Speicherbedarf der finalen μP-Software

erfolgreicher Kompilation dargestellt. Deren Informationen beziehen sich auf die finale Version der Mikrokontrollersoftware, deren Quellcode in Anhang B angefügt ist.

2.4.3 Blockschaltplan

Die Software des Mikrokontrollers hat die Aufgabe, die zur Berechnung von Herzfrequenz und Sauerstoffsättigung notwendigen vom Sensor erzeugten Absorptionssignale über eine Digitalisierung auszuwerten und deren Gleichanteil in ein vergleichbares Niveau zu regeln. Der dazu notwendige Ablauf ist schemenhaft als Blockschaltplan in Abbildung 18 (S. 33) dargestellt und visualisiert, in welcher Form die Messdatenakquise und Parameterberechnung abläuft. Die dabei eingesetzten Elemente werden in den folgenden Unterkapiteln ausführlich beschrieben und spiegeln die vom Quellcode erzeugten (siehe Anhang B) Funktionseigenschaften wieder. Die Signalaufnahme und Regelung der Intensitätsstärke erfolgt in einer timerbasierten Interruptroutine (`take_samples`), die alle $4 * 10^{-}3$s ausgeführt wird. Dabei werden jeweils ein Sample des IR- und des VR-Signals erzeugt und in zwei zugehörige Datenarrays abgelegt. Nach jeweils 2048 Samples erfolgt aus diesen Daten eine FFT-Berechnung mit anschließender Berechnung der Herzfrequenz sowie des Absorptionsratios und in weiterer Folge der Sauerstoffsättigung. Diese Berechnungsergebnisse werden an einem 16x2 LCD ausgegeben und dem Benutzer angezeigt.

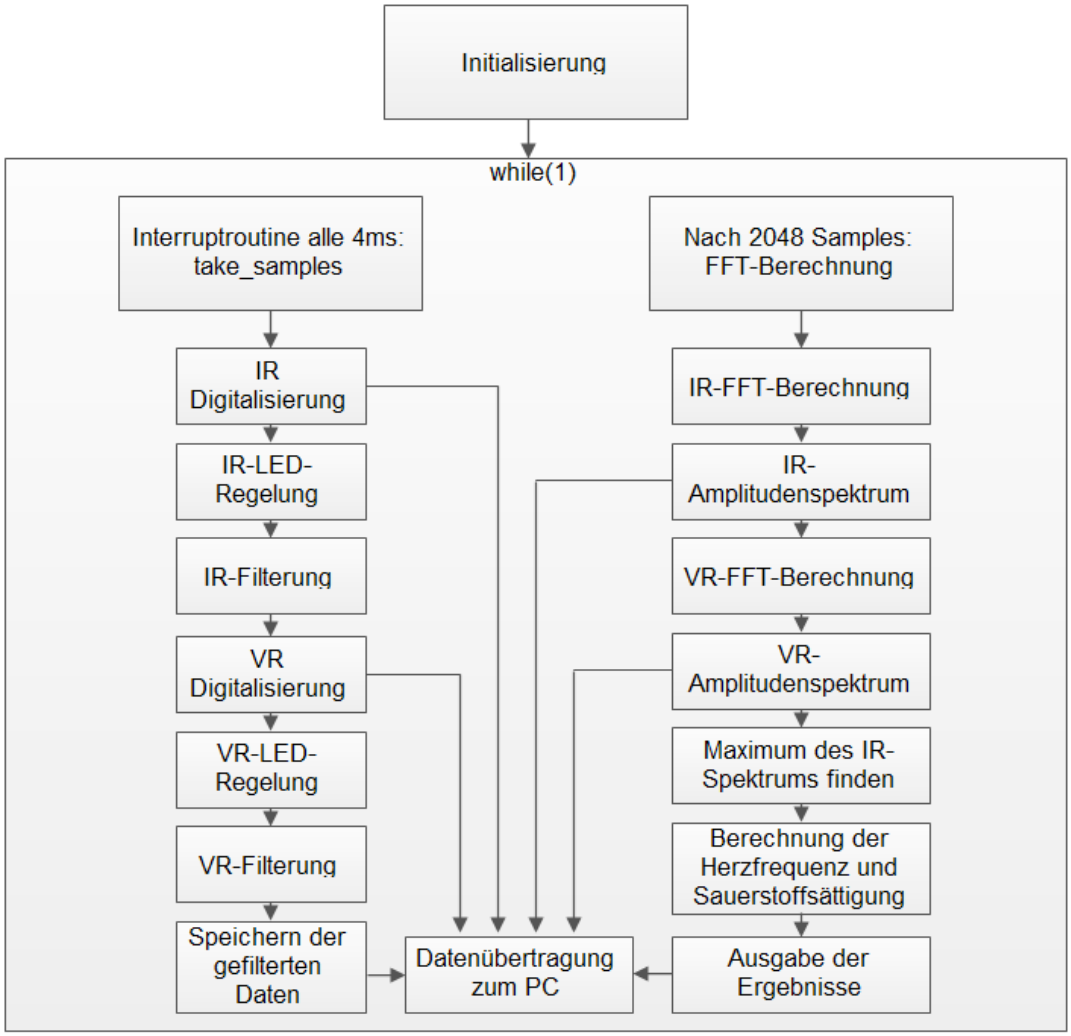


Abbildung 18: Schemenhafter Blockschaltplan der Mikroprozessorsoftware

Über ein serielles Interface in Kombination mit einem MATLAB (Mathworks Inc., Natwick, USA) Programm ist es zudem möglich, die digitalen oder gefilterten Samples als auch die Berechnungsergebnisse der FFT anzufordern, die dadurch an das MATLAB (Mathworks Inc., Natwick, USA) GUI übertragen und dargestellt wird.

2.4.4 Initialisierung

Im ersten Programmteil erfolgt die Deklarierung der zur Auswertung der Vitalparameter notwendigen Variablen sowie der Portdefinition des Mikrokontrollers und die Deklarierung der dazu notwendigen μP-spezifischen Variablen. Um einen raschen Überblick der verwendeten Mikroprozessorstruktur zu erlangen, sind diese in Listing 1 (S. 34) ange-

führt. Die dabei links durch Nummern markierten Codezeilen sind ident mit den Codezeilen des Originalquellcodes in Anhang B und sollen die Orientierung erleichtern.

Listing 1: Portdefinition des Mikrokontrollers und die Deklarierung der dazu notwendigen μP-spezifischen Variablen

```
157 //-----------------------------------------------
158 // MBED Port definition
159 //
160   //LCD-Panel
161     TextLCD lcd(p5, p6, p12, p13, p14, p15);
162
163   //Digitale I/O Ports (output)
164     DigitalOut ir_control(p9);
165     DigitalOut vl_control(p10);
166     DigitalOut ir_demultiplex_control(p28);
167     DigitalOut vl_demultiplex_control(p26);
168
169   //Digitale I/O Ports (input)
170     DigitalIn dac_switch(p7);
171
172   //ADC
173     AnalogIn tiefpass1(p19);
174     AnalogIn tiefpass2(p20);
175
176     AnalogIn second_op1(p17);
177     AnalogIn second_op2(p16);
178
179
180   //DAC
181     AnalogOut dac(p18);
182
183   //PWM Ausgang
184     PwmOut ir_ledcontrol(p21);
185     PwmOut vr_ledcontrol(p22);
186     PwmOut ir_dcoffset(p24);
187     PwmOut vr_dcoffset(p23);
188
189 //-----------------------------------------------
190 // MBED-Variablen
191 //
192   //Emulierte Serielle Schnittstelle via USB
193     Serial pc(USBTX, USBRX); // tx, rx
```

```
194
195   //LED
196     DigitalOut led(LED1);
197
198   //Timer
199     Timer t;
200
201   //Interrupt for the given Ticker
202     Ticker sample_ticker;
```

2.4.5 Interrupt, Timer und Zeitmanagement

Über die Verwendung einer timerbasierten Interruptroutine und Abfragen eines weiteren Timer-Status, bildet die genau in $4*10^3$s-Abständen ausgeführte Funktion `take_samples()` das in Abbildung 2.3.5 (S. 21) dargestellte Zeitprofil nach. Die Verwendung einer timerbasierten Interruptroutine hat hierbei den großen Vorteil, dass trotz weiterer Berechnungen über den Interrupttimer die zugehörigen Funktionen verlässlich in $4*10^3$s-Abständen aufgerufen und bevorzugt abgearbeitet werden. Dadurch wird eine feste Abtastrate sichergestellt, die für den ordnungsgemäßen Einsatz von digitalen Filtern als auch für die Berechnung einer FFT vorausgesetzt wird. Die Regelung der Abtastung erfolgt über eine direkt nach dem Funktionsaufruf gestartete Timer-Variable, deren Inhalt (in Mikrosekunden) über die auszuführenden Aktionen bestimmt. Ausgedehnte wait-Kommandos (insgesamt vier mal) stellen sicher, dass die außerhalb der Interruptroutine durchgeführten Berechnungen in diesen Zeiträumen durchgeführt werden können. Der grobe Interruptablauf ist in Listing 2 (S. 35) angeführt und zeigt die Erstellung des Zeitprofils basierend auf der Timervariable `t`.

Listing 2: Pseudocode zur Beschreibung der zyklischen Messdatenakquise

```
// Initialisierung der Timer-basierten Interruptroutine
sample_ticker.attach_us(&take_samples, sample_time);

while(1)
{
  //Berechnungen außerhalb der Interruptroutine
}

void take_sample()
{
  t.start();
```

```
  //VR Aus, IR EIN
  wait_us(sample_time/4);
  //Digitalisierung des IR-Signals
  //Regelung der IR-Intensität
  while(t.read_us() <= sample_time/2);
  //reset the timer
  t.reset();

  t.start();
  //VR Ein, IR Aus
  wait_us(sample_time/4);
  //Digitalisierung des VR-Signals
  //Regelung der VR-Intensität
  t.reset();
}
```

2.4.6 Regelung der LED-Intensitäten

Die Angleichung der vom Messobjekt absorbierten und über einen Transimpedanzverstärker in ein Spannungssignal umgesetzten Intensitätssignale erfolgt in der Interruptroutine `take_samples` durch Regelung der LED-Ströme über einen PWM-Ausgang (siehe 2.3.6, S. 24). Dazu werden die Spannungssignale, aufgetrennt durch den Umschalter und gefiltert über einen TP, digitalisiert und durch Erhöhung bzw. Verringerung der zugehörigen PWM-Duty-Cycle-Variablen (`vr_LED_level` und `ir_LED_level`) über die Funktion `float control_LED_level(sample_op1, LED_level)` und Aktualisierung der PWM-Ausgänge in denselben Spannungsbereich geregelt. Der Ausgabebereich der PWM-Variable wird auf den durch TP-Filterung entstehenden Gleichspannungsbereich limitiert und kann Werte zwischen 0.2 (entspricht 0.66V) bis 0.6 (entspricht ca. 2V) einnehmen. Dazu wird diese Funktion pro Zeitzyklus jeweils einmal für die Regelung des IR- und VR-LED-Stroms aufgerufen und besteht aus dem in Abbildung 19 (S. 37) dargestellten Ablaufplan.

2.4.7 Digitalisierung und Aufnahme der Messdaten

Zur späteren Auswertung der Messdaten werden diese vom Mikrokontroller bzw. von dessen integrierten 12 bit A/D-Wandlern digitalisiert und als 16 bit unsigned short Variable über die Funktion `read_u16`bereitgestellt. Nach anschließender Typkonversion und einer Filterung mit digitalen IIR-Filtern, werden diese Samples in zwei float-Arrays

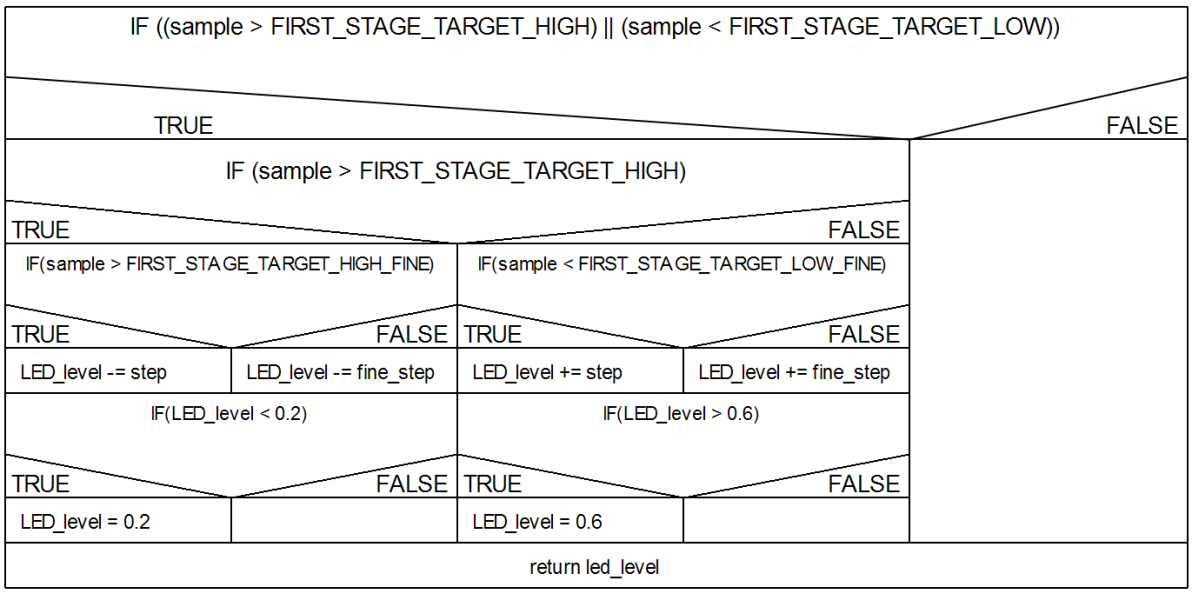


Abbildung 19: Nassi-Schneidermann-Diagramm der LED-Intensitätsregelung

(jeweils 2048-Elemente) abgelegt, aus denen in weiterer Folge die Berechnung der FFT erfolgt (siehe 2.4.8, S. 37).

2.4.8 Signalverarbeitung der Messdaten

Die Aufbereitung der gesampelten und digital vorliegenden Messdaten zur Berechnung der Sauerstoffsättigung erfolgt in zwei Schritten. Zunächst werden über zwei digitale IIR-Filter 3. Ordnung (Direktform II) aus den Zeitsignalen ungewünschte Frequenzanteile entfernt, bevor diese per FFT in den Frequenzraum transformiert werden. Aus diesen FFT-Daten werden die Maxima extrahiert und zur Berechnung der Herzfrequenz und Sauerstoffsättigung herangezogen.

Über die Filterfunktionen `ir_iir_filter`, `vr_iir_filter`, `ir_hp_iir_filter` und `ir_hp_iir_filter` erfolgt die Filterung der Signaldaten, die anschließend in zwei 2024-Elemente große float-Arrays abgelegt werden. Nach vollständiger Befüllung werden diese Datenarrays herangezogen, um eine FFT durchzuführen (Implementierung nach [30]). Dabei kommt eine reelle FT mit N/2 Abtastpunkten wie in Abbildung 21 (S. 38) zum Einsatz. Dies hat zur Folge, dass, bedingt durch die nur im reellen Raum vorliegenden Messdaten, sich der Speicheraufwand auf N/2 reduziert. Nach erfolgter Berechnung des Betragspektrums kann das Maximum des IR-Spektrums gesucht und der Frequenzindex sowie die zugehörige Amplitude zwischengespeichert werden. Über den Frequenzindex wird in einem weiteren Schritt der Amplitudenwert des VR-Spektrums

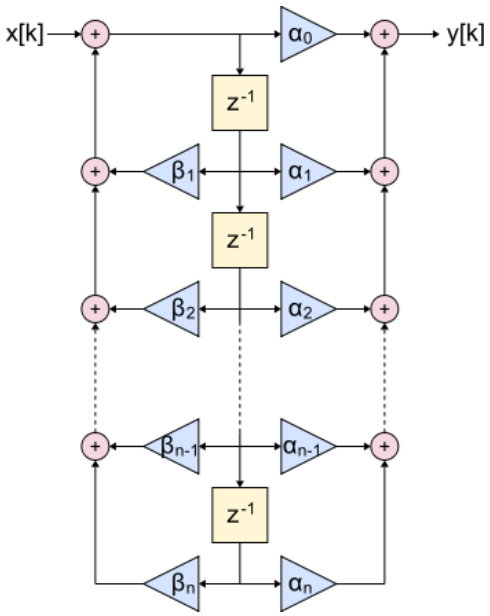


Abbildung 20: Blockschaltdarstellung der IIR-Filterimplementierung für die Direktform II [31]

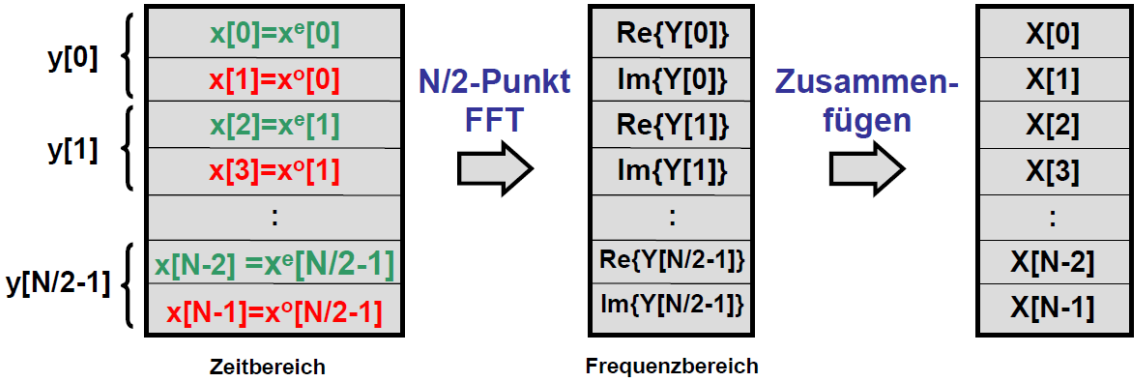


Abbildung 21: Überblick des angewandten FFT-Prinzps zur FT von reellen Daten [32]

ausgelesen, bevor durch einfache Multiplikation die Herzfrequenz (siehe Gleichung 16, S. 38) bestimmt wird.

$$\texttt{heartrate} = \texttt{int(index\_fft\_max*60*125/1024)} \tag{16}$$

Aus den ermittelten Spektralwerten, die den durch unterschiedlich starke Absorption des pulsierenden arteriellen Bluts hervorgerufenen Wechselanteilen entsprechen, wird das Ratio nach Gleichung 17 (basierend auf Gleichung 12, S. 12) berechnet.

$$\texttt{ratio} = \frac{\texttt{vr\_fft\_out[index\_fft\_max]}}{\texttt{ir\_fft\_out[index\_fft\_max]}}; \tag{17}$$

Basierend auf einem von TI (Texas Instruments, Dallas, USA) übernommenen Look-up-Table [17] wird eine Extrapolation durchgeführt und eine Fittingfunktion nach Gleichung 15 (S. 12) erstellt, mit der die Sauerstoffsättigung aus dem Ratio berechnet wird. Bedingt durch die maximale Fenstergröße der FFT-Berechnung (siehe 2.4.2, S. 31), können die dabei ermittelten Werte für das Ratio leicht variieren und werden vor der Berechnung einer gleitenden Mittelwertberechnung unterzogen. Dazu werden die jeweils vier letzten Werte gemittelt und erst dieser Wert wird für die Berechnung der Sauerstoffsättigung herangezogen (siehe Gleichung 18, S. 39) und als Prozentwert dem Benutzer am LCD angezeigt.

$$\texttt{SaO2} = \texttt{int}(110.28 - 17.51 * \texttt{ratio\_average}) \tag{18}$$

2.4.9 Anzeige der Daten am LCD

Um die Parameter Herzfrequenz und Sauerstoffsättigung dem Benutzer komfortabel anzuzeigen, besitzt die Hardware eine 16X2 Flüssigkristallanzeige. Die Mikrokontrollersoftware greift über die in der Initialisierung (siehe 2.4.4, S. 33) festgelegten I/O-Ports auf die Hardware zu und gibt unter Zuhilfenahme der vom mbed-Benutzer „Simon Ford“ erstellten Library `TextLCD` die jeweiligen Integerwerte aus. Dazu braucht, nach erfolgter LCD-Initialisierung durch die Funktion `locate(int x-position, y-position)`, lediglich die gewünschte Position ausgewählt werden, um per `printf`-Funktion den gewünschten Text auszugeben. Zur Visualisierung dieses Vorgangs sind die dazu notwendigen Befehle in Listing 3 (S. 39) dargestellt.

Listing 3: Ansteuerung der 16x2 Flüssigkristallanzeige über Verwendung der Library TEXTLCD

```
510  //Ceck if there is a signal (LED_levels will differ from 0.2)
511     //and print the results to the display
512     if(ir_LED_level != vr_LED_level)
513     {
514       //Print the heartrate
515       lcd.locate(5,0);
516       lcd.printf("frequ%3dbpm", heartrate);
517
518       //Check if the moving average is ready
```

```
519         if(start_plot == 1)
520         {
521           //Print the Oxygen-Saturation
522           lcd.locate(5,1);
523           lcd.printf("SPO2 %3d%%  ", SaO2);
524         }
525         else
526         {
527           lcd.locate(11,1);
528           lcd.printf("XXX");
529         }
530       }
531       //there is no signal
532       else
533       {
534           lcd.locate(5,0);
535           lcd.printf("      no    ");
536           lcd.locate(5,1);
537           lcd.printf("    signal ");
538       }
```

2.4.10 Serielles Interface

Die über den Mikrokontroller aufgenommenen und zur Berechnung der Sauerstoffsättigung herangezogenen Daten können durch Erstellung eines seriellen Interfaces an einen PC übertragen werden. Dazu wird in der Mikroprozessorsoftware eine RS232 imitierende Schnittstelle über die `Serial`-Variable und die Wahl von RxD- und TxD-Leitungen als USBTX und USBRX initialisiert (siehe 2.4.4, S. 33) und kann, nach Festlegung der seriellen Eigenschaften, zur Datenübertragung verwendet werden. Die dabei vorgenommenen Einstellungen für die Baudrate sowie die Anzahl der Daten-, Parity- und Stop-Bits erfolgt nach den in Tabelle 7 (S. 40) vermerkten Parametern. Diese Varia-

Tabelle 7: Übersicht der im μP getroffenen Einstellungen zum seriellen Interface

Eingenschaft	Wert
BaudRate	460800
DataBits	8
Parity	0
StopBits	1

ble ermöglicht es, im Programmcode nach erfolgter Digitalisierung oder Berechnung der

FFT, die Daten nach Anforderung des erstellten MATLAB-GUIs per `printf`-Funktion bequem an den PC zu übertragen und dort darzustellen und auszuwerten. Dazu wird in der `while(1)`-Schleife des Mikrokontrollerprogramms zyklisch abgefragt ob ein Charakterwert zum Empfang bereitsteht, um anschließend die zugehörige Variable (siehe Tabelle 8, S. 41) an den PC zu schicken.

Tabelle 8: Übersicht des Kommunikationsinterfaces

Signalauswahl `popup_choose_signal`	Sendevariable in `char`	Empfangsvariable in `char*`	Beschreibung
'Samples nach OP1'	'3'	`ir_sample_op1` `vr_sample_op1`	Übertragung von zwei `unsigned int`-Variablen
'Samples nach OP2'	'2'	`ir_sample_op2` `vr_sample_op2`	Übertragung von zwei `unsigned int`-Variablen
'gefilterte Samples'	'1'	`ir_filtered_sample` `vr_filtered_sample`	Übertragung von zwei `float`-Variablen
'μP-FFT'	'c'	`ir_fft_out[1024]` `vr_fft_out[1024]`	Übertragung von 2*1024 `float`-Variablen

2.5 MATLAB-GUI

Bevor die Implementierung der gewünschten Funktionen erfolgte, wurde die grafische Oberfläche, im speziellen die Anordnung der gewünschten Elemente, durch Zuhilfenahme des „GUI Layout Editor“ entworfen. Dieser Editor wird mit der Eingabe von `guide` in die MATLAB-Kommandozeile gestartet und ermöglicht eine einfache und rasche Erstellung einer grafischen Oberfläche. Diese wird als fig-File (`pofm_gui.fig`) im entsprechenden Ordner abgespeichert. Des Weiteren werden auch die Initialisierung und die Callback-Funktionen der GUI-Elemente in einem m-file zur Verfügung gestellt (`pofm_gui.m`). Darauf aufbauend wurden die jeweiligen Code-Abschnitte um die gewünschte Funktionalität erweitert.
Um einen kompakten Überblick zu erhalten, ist in Abbildung 22 (S. 42) die grafische Oberfläche mit den markierten Elementgruppen abgebildet. Jede dieser Elementgruppen wird in der folgenden Aufzählung mit ihrem Elementnamen, Elementtyp, dem Default-Wert und deren wichtigsten Funktionen kurz beschrieben.

1. **`popup_com_port`**, Pop-up Menu, default=' '
   Beschreibung: Ermöglicht die Auswahl des COM-Ports basierend auf den am Computer verfügbaren COM-Ports.

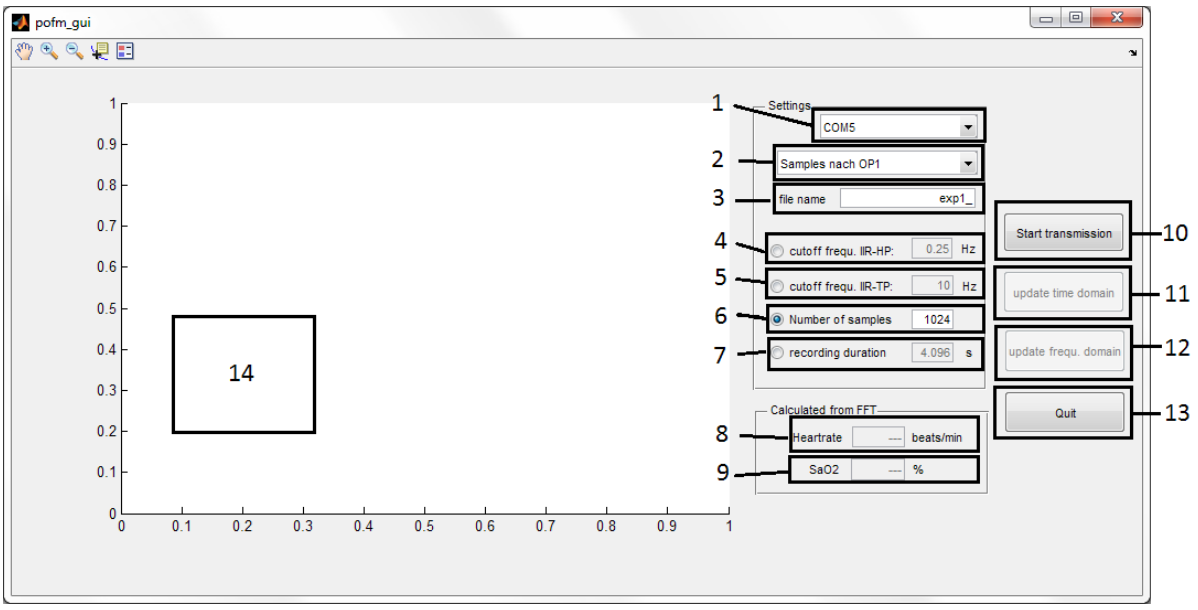


Abbildung 22: Überblick über die grafische Oberfläche. Markiert und nummeriert sind funktionell zusammenhängende Elemente, die in 2.5 (S. 41) beschrieben werden

2. **`popup_choose_signal`**, Pop-up Menu, default: 'Samples nach OP1','Samples nach OP2','gefilterte Samples','μP FFT'
   Beschreibung: Ermöglicht die Auswahl des zu übertragenden Signaltyps.

3. **`edit_filename`**, Edit Text, default: 'exp1_'
   Beschreibung: Textbox um den Dateinamen des Logfiles anzugeben.

4. **`radio_hp`**, Radio Box, default: '0' (nicht ausgewählt)
   Beschreibung: Radiobutton um abzufragen, ob eine IIR-HP-Filterung durchgeführt werden soll. Wird der Radiobutton vom Benutzer ausgewählt (Value: 1), so wird die Textbox `edit_fg_hp` aktiviert. Ist der Radiobutton jedoch nicht ausgewählt (Value = 0), wird die Textbox `edit_fg_hp` deaktiviert.

   `edit_fg_hp`, Edit Text, default: '0.25'
   Beschreibung: Textbox um die Grenzfrequenz des IIR-HP-Filters anzugeben. In der Callback-Funktion wird überprüft, ob eine gültige Zahl eingegeben wurde. Ist dies nicht erfüllt, wird ein Warnungsfenster ausgegeben und der Inhalt auf den default-Wert gesetzt.

5. **radio_tp**, Radio Box, default: '0' (nicht ausgewählt)
   Beschreibung: Radiobutton um abzufragen, ob eine IIR-TP-Filterung durchgeführt werden soll. Wird der Radiobutton vom Benutzer ausgewählt (Value: 1), so wird die Textbox `edit_fg_tp` aktiviert. Ist der Radiobutton jedoch nicht ausgewählt (Value = 0), wird die Textbox `edit_fg_tp` deaktiviert.

   edit_fg_tp, Edit Text, default: '10'
   Beschreibung: Textbox um die Grenzfrequenz des IIR-HP-Filters anzugeben. In der Callback-Funktion wird überprüft, ob eine gültige Zahl eingegeben wurde. Ist dies nicht erfüllt, wird ein Warnungsfenster ausgegeben und der Inhalt auf den default-Wert gesetzt.

6. **radio_number**, Radio Box, default: '1' (ausgewählt)
   Beschreibung: Radiobutton um abzufragen, ob die Sampleanzahl (Anzahl der zu übertragenden Samples) veränderbar sein soll. Wird der Radiobutton vom Benutzer ausgewählt (Value: 1), so wird die Textbox `edit_number_of_samples` aktiviert und die Elemente `radio_time` und `edit_scan_duration` deaktivert. Eine Deaktivierung der Radio Box `radio_number` erfolgt durch Auswahl der Radio Box `radio_time`.

   edit_number_of_samples, Edit Text, default: '1024'
   Beschreibung: Textbox um die Anzahl der gewünschten Samples pro Wellenlänge und Signaltyp anzugeben. In der Callback-Funktion wird überprüft, ob eine gültige Zahl eingegeben wurde. Ist dies nicht erfüllt, wird ein Warnungsfenster ausgegeben und der Inhalt auf den default-Wert gesetzt. Ist der Inhalt jedoch eine Zahl, wird die zugehörige Sampledauer berechnet und in der Textbox `edit_scan_duration` ausgegeben.

7. **radio_time**, Radio Box, default: '0' (nicht ausgewählt)
   Beschreibung: Radiobutton um abzufragen, ob die Sampledauer veränderbar sein soll. Wird der Radiobutton vom Benutzer ausgewählt (Value: 1), so wird die Textbox `edit_scan_duration` aktiviert und die beiden Elemente `radio_number` und `edit_number_of_samples` deaktivert. Eine Deaktivierung der Radio Box `radio_time` erfolgt durch Auswahl der Radio Box `radio_number`.

edit_fg_tp, Edit Text, default: '4.096'
Beschreibung: Textbox um die gewünschte Sampledauer der Übertragung in Sekunden anzugeben. In der Callback-Funktion wird überprüft, ob eine gültige Zahl eingegeben wurde. Ist dies nicht erfüllt, wird ein Warnungsfenster ausgegeben und der Inhalt auf den default-Wert gesetzt. Ist der Inhalt eine Zahl, wird die zugehörige Sampledauer berechnet und in der Textbox `edit_scan_duration` ausgegeben.

8. **edit_hr**, Textbox, default: '—'
   Beschreibung: Textbox um die aus den FFT-Werten berechnete Herzfrequenz auszugeben.

9. **edit_sao2**, Textbox, default: '—'
   Beschreibung: Textbox um die aus den FFT-Werten berechnete Sauerstoffsättigung auszugeben.

10. **button_start**, Push Button, default: enabled
    Beschreibung: Die Aktivierung dieses Buttons liest diverse GUI-Felder aus, startet die Kommuniktion mit dem μP, empfängt die per serieller Schnittstelle übertragenen Datensamples und speichert diese in einem Logfile. Ist zumindest einer der IIR-Radiobuttons (`radio_hp` oder `radio_tp`) aktiviert, werden die Datensamples vor der Ausgabe im Plot `axes_signal` digital gefiltert.

11. **button_timespace**, Push Button, default: disabled
    Beschreibung: Die Aktivierung dieses Buttons liest die Daten aus dem Logfile aus und fragt den Status der IIR-Radiobuttons (`radio_hp` oder `radio_tp`) ab. Je nach deren Stellung werden die Datensamples vor der Ausgabe im Plot `axes_signal` digital gefiltert oder direkt ausgegeben.

12. **button_fft**, Push Button, default: disabled
    Beschreibung: Die Aktivierung dieses Buttons liest die Daten aus dem Logfile aus und fragt den Status der IIR-Radiobuttons (`radio_hp` oder `radio_tp`) ab. Je nach deren Stellung werden die Datensamples digital gefiltert oder die FFT wird ohne Filterung durchgeführt. Aus dem Ergebnis der FFT wird das einseitige Am-

plitudenspektrum berechnet und im Plot `axes_signal` ausgegeben. Aus dem Amplitudenspektrum der infraroten Datenpunkte werden das Maximum und der zugehörige Frequenzindex bestimmt und zur Berechnung der Herzfrequenz und der Sauerstoffsättigung herangezogen und in der zugehörigen Textbox (`edit_hr` und `edit_sao2`) ausgegeben.

13. **`button_quit`**, Push Button, default: enabled
    Beschreibung: Die Aktivierung dieses Buttons schließt die grafische Oberfläche und beendet damit das MATLAB-Programm.

14. **`axes_signal`**, Axes
    Beschreibung: Dieses GUI-Element dient der Darstellung der Datensamples bzw. der einseitigen Amplitudenspektren.

2.5.1 Auswahlmöglichkeiten der verschiedenen Signaltypen

Die grafische Oberfläche ermöglicht es dem Benutzer drei verschiedene Signaltypen zu übertragen - neben den Samples im Zeitbereich (ungefiltert und gefiltert) können auch die Daten des im μP berechneten einseitigen FFT-Amplitudenspektrums übertragen werden. Da diese Signaltypen unterschiedliche Parameterveränderungen erlauben, wurden diese Möglichkeiten softwaremäßig durch die Auswahl des Pop-up Menüs `popup_choose_signal` beschränkt. Die folgende Auflistung soll die Möglichkeiten des Users übersichtlich zusammenfassen.

- **'Samples nach OP1'** und **'Samples nach OP2'**
  digitale Samples nach der ersten bzw. der zweiten OPV-Stufe (Transimpedanzwandler bzw. invertierender Verstärker)
  - Darstellung der übertragenen Daten
  - Änderung der Sampleanzahl oder Sampledauer
  - digitale Filterung per IIR-HP
  - digitale Filterung per IIR-TP
  - Update des Zeitbereichs
  - Durchführung und Update der FFT sowie der Berechnung von Herzfrequenz und Sauerstoffsättigung
- **'gefilterte Samples'**

im μP digital gefilterte Samples (IIR-HP 3.Ordnung $f_g = 0.25Hz$ und IIR-TP 3.Ordnung $f_g = 10Hz$

  - Darstellung der übertragenen Daten
  - Änderung der Sampleanzahl oder Sampledauer
  - Update des Zeitbereichs
  - Durchführung und Update der FFT sowie der Berechnung von Herzfrequenz und Sauerstoffsättigung

- **'μP-FFT'**
  im μP berechnetes einseitiges Amplitudenspektrum (1024 samples im Frequenzbereich)
  - Darstellung des übertragenen einseitigen Amplitudenspektrums

2.5.2 Kommunikationsinterface

Um eine Datenverbindung mit dem μP-Board aufzubauen, wird in dessen Code eine virtuelle serielle Schnittstelle initialisiert. Diese ermöglicht es, über das USB-Kabel, das gleichzeitig der $5V$-Spannungsversorgung dient, eine Kommunikation mit dem MATLAB-GUI aufzubauen. Dazu befindet sich im μP-Code ein serieller Interrupt, der bei Empfang spezieller Character-Werte die zugehörigen Variablen (siehe Tabelle 8, S. 41) zurückschickt. Dabei ist darauf hinzuweisen, dass bei der Auswahl des GUI Elements `popup_choose_signal` lediglich die zugehörige Pop-up-Stellung ausgewählt wird. Das Senden des Charactercodes und der Empfang der vom μP gesendeten Variablen erfolgt erst durch die Auswahl des Buttons `button_start`. Dabei wird die Sendevariable im μP-Code gespeichert und pro Sample-Zyklus (4ms) werden die zugehörigen Variablen übertragen bis die in dem Element `edit_number_of_samples` angezeigte Anzahl an Samples erreicht wird. Der μP-Code ist dabei so konzipiert, dass die Übertragung stoppt, sobald keine von den in Tabelle 8 (S. 41) aufgelisteten Sendevariablen übertragen wird. Eine Ausnahme liefert die Übertragung eines `'c'`, bei der erst gewartet wird bis eine vollständige FFT der zwei gefilterten Signale vorliegt. Diese Daten werden im Anschluss an die Berechnung komplett übertragen (2*1024 Variablen). Dabei kann es vorkommen, dass sich eine kurze Wartezeit von weniger als 8.192 Sekunden ergibt (2048*0.004 Sekunden = 8.192 Sekunden). Die Übertragung der Variablen erfolgt durch eine automatische Umwandlung im μP in einen Character-String, der wiederum in MATLAB in eine zugehörige double-Zahl umgewandelt wird.

2.5.3 serielle Einstellungen

Um eine fehlerfreie Übertragung mit dem μP sicherzustellen, müssen bei der Initialisierung des seriellen Port-Objekts in MATLAB dieselben Eigenschaften wie im μP ausgewählt werden. Diese im μP getroffenen seriellen Einstellungen sind in Tabelle 7 (S. 40) zusammengefasst und werden in der Callback-Funktion des Elements `button_start` durch folgende Zeilen realisiert:

Listing 4: Serielle Einstellungen

```
126 %Open the Serial-Connection
127 serial_handle = serial(choosen_port,'BaudRate',460800,...
128   'DataBits',8, 'Parity', 'none', 'StopBits', 1);
129 fopen(serial\_handle);
```

2.5.4 Sendevorgang

Eine weitere wichtige Anmerkung bezieht sich auf eine Eigenheit des vom μP erstellten seriellen Interfaces in Kombination mit dem durch MATLAB erstellten seriellen Interface. Trotz Übereinstimmung der in Tabelle 7 (S. 40) aufgelisteten Einstellungen ist eine direkte Kommunikation nur unter Verwendung eines try-Blocks möglich. Erfolgt der Sendevorgang per `fwrite` oder `fprintf` außerhalb eines try-Blocks, wird die Variable laut MATLAB zwar übertragen, kann aber vom μP nicht korrekt empfangen werden. Die folgenden Codezeilen zeigen die Anwendung dieses try-Blocks, um erfolgreich eine Variable von MATLAB über die im vorliegenden Fall virtuelle serielle Schnittstelle an den μP zu übertragen.

Listing 5: Sendevorgang

```
165 %%% start the transmission by sending the transmission-
166 %%% character
167 try
168   fwrite(serial\_handle,data\_type, 'uchar');
169 end
```

2.5.5 Empfangsvorgang

Um die vom μP gesendeten Variablen zu empfangen, werden diese in einer for-Schleife alternierend abgefragt (zuerst die ir-samples und dann vr-samples), bis die gewünschte Anzahl an Samples erreicht ist (siehe GUI Element `edit_number_of_samples`). Obwohl in MATLAB für den Sendevorgang ein try-Block erforderlich ist (siehe 2.5.4, S. 47), kann das Empfangen direkt durch die Verwendung der Funktion `fscanf` erfolgen. Dabei ist anzumerken, dass über die serielle Schnittstelle lediglich char-Variablen übertragen werden können. Die daraus resultierenden Empfangsvariablen (char string) werden in dem Logfile (der Name wird aus dem Eintrag des GUI-Elements `edit_filename` ausgelesen) abgespeichert und können durch einfache Typkonversion in eine Zahl (mittels MATLAB-Funktion `str2double`) umgewandelt werden. Das folgende Code-Segment zeigt diesen Vorgang:

Listing 6: Empfangsvorgang

```
171 %%% receive the samples over the COM-port, write them to
172 %%% the log-file and save them in the corresponding array
173 for counter=1:number_samples
174   sample\_ir= fscanf(serial\_handle, '\%s');
175   fprintf(fid,'\%s\textbackslash n',sample\_ir);
176   ir\_data(counter) = str2double(sample\_ir);
177   sample\_vr= fscanf(serial\_handle, '\%s');
178   fprintf(fid,'\%s\textbackslash n',sample\_vr);
179   vr\_data(counter) = str2double(sample\_vr);
180 end
```

2.5.6 Dateiformat des Logfiles

Die vom μP übertragenen und durch das MATLAB-Interface empfangenen Daten werden zur Archivierung und späteren Bereitstellung für Filteroperationen in einem Logfile abgespeichert. Der Dateiname wird aus dem GUI-Element `edit_filename` ausgelesen und eine neue Datei - bereits bestehende Dateien werden automatisch überschrieben - wird erstellt. Um eine gewisse Konsistenz zu dem Sendeprotokoll zu bewahren, werden die empfangenden Daten in der selben Reihenfolge abgespeichert, wie diese empfangen werden. Das bedeutet, dass im abwechselnden Rhythmus zunächst die ir-samples und anschließend die vr-samples als string abgespeichert werden. Als Anschauungsbeispiel zeigt Abbildung 23 (S. 49) diese Formatierung.

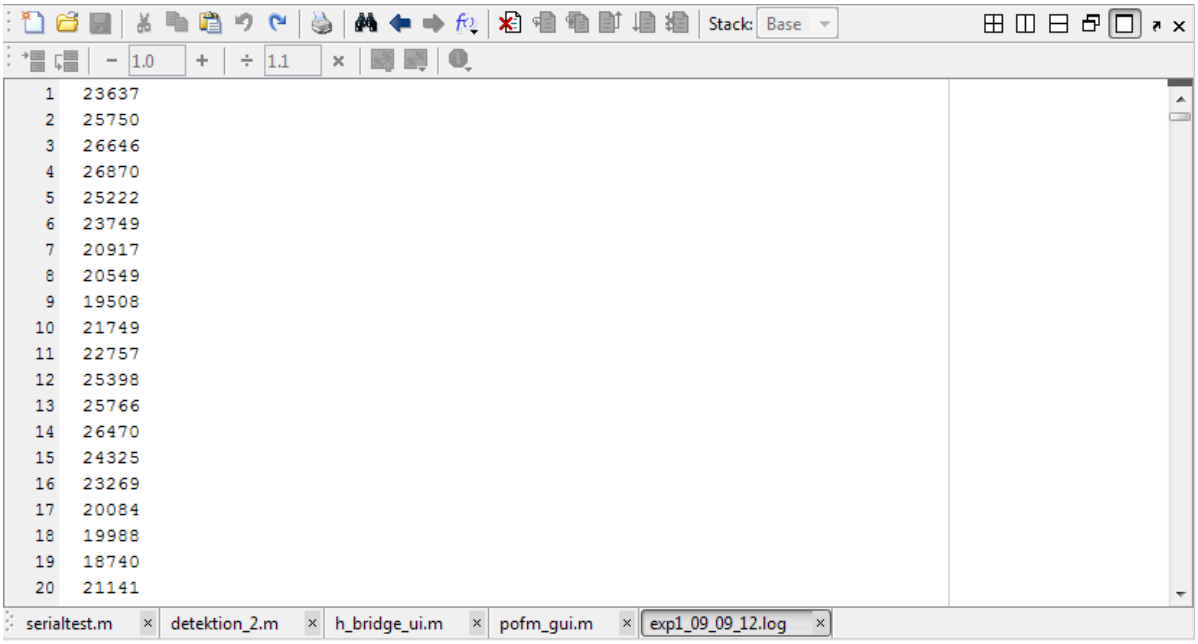


Abbildung 23: Überblick über den Inhalt eines Logfiles nach der Übertragung von OP1-Samples. Die ungeraden Zeilennummern repräsentieren ir-samples und die geraden Zeilennummern die vr-samples

2.5.7 Einsatz von digitalen Filtern per MATLAB

Um dem Benutzer die Möglichkeit zu geben die empfangenen Daten (sofern OP1- oder OP2-Samples übertragen wurden) vor der Ausgabe oder FFT-Berechnung per digitalem Filter zu bearbeiten, kann dieser durch Aktivierung der zugehörigen Radio-Buttons (`radio_hp` bzw. `radio_tp`) und Eingabe der Grenzfrequenzen eine IIR-Filterung auslösen. Dazu werden die zuvor übertragenen Daten aus dem Logfile ausgelesen, per MATLAB-Funktion `butter` die Filterkoeffizienten für ein IIR-Filter 3.Ordnung berechnet (basierend auf der vom User eingegebenen Grenzfrequenz) und über die MATLAB-Funktion `filtfilt` wird eine Filterung durchgeführt. Die resultierenden Datenarrays werden entweder dem Benutzer direkt dargestellt (beim Aufruf des Buttons `button_start` bzw. des Buttons `button_timespace`) oder zur Berechnung der FFT bzw. des einseitigen Amplitudenspektrums verwendet (beim Aufruf des Buttons `button_fft`).

2.5.8 FFT-Berechnung per MATLAB

Die Berechnung des einseitigen FFT-Amplitudenspektrums wird zu zwei Zwecken durchgeführt. Zum Einen erlangt der Benutzer dadurch einen schnellen Überblick der im Signal

enthaltenen Frequenzanteile - er kann dadurch intuitiv die Qualität des Zeitsignals beurteilen - und zum Anderen ermöglichen die Ergebnisse des einseitigen Amplitudenspektrums eine automatische Berechnung der Herzfrequenz sowie der Sauerstoffsättigung. Dazu wird das Signalarray zunächst per MATLAB-Funktion `nextpow2` auf eine Länge erweitert, die der nächstgrößeren Potenz von 2 entspricht (zb von 1453 auf $2^{11} = 2048$). Mit diesem erweiterten Datenarray wird über die MATLAB-Funktion `fft` die FFT berechnet. In einem letzten Schritt wird das FFT-Ergebnis (zweiseitig und definitionsgemäß imaginär) geshiftet und über eine Absolutwertbildung in ein einseitiges Amplitudenspektrum umgewandelt. Dieses wird dem Benutzer über das GUI-Element `axes_signal` ausgegeben und zur Berechnung der Herzfrequenz sowie der Sauerstoffsättigung herangezogen.

2.5.9 Berechnung der Herzfrequenz und der Sauerstoffsättigung

Die in diesem Unterpunkt angeführte Beschreibung ist eine kurze Zusammenfassung der Berechnungsmöglichkeiten und der dahinterstehenden Grundlagen. Um die Herzfrequenz sowie die Sauerstoffsättigung des Probanden basierend auf den übertragenen Datensamples zu bestimmen, gibt es prinzipiell zwei Möglichkeiten. Man kann eine direkte, großteils manuelle Auswertung des Zeitsignals über einen max-min Peak-Detektor realisieren, oder man berechnet sich die FFT und zieht diese zur Auswertung heran.

Der große Nachteil, einer manuellen Auswertung des Zeitsignals liegt in der starken Anfälligkeit auf Störsignale (Bewegungsartefakte, Netzbrumm), die die Ergebnisse sehr rasch unbrauchbar machen können. Dieses instabile Verhalten verhindert man durch eine ausreichend aufgelöste Fouriertransformation des Zeitsignals. Aus diesem Grund werden im vorliegenden MATLAB-GUI die Herzfrequenz als auch die Sauerstoffsättigung aus den Ergebnissen des einseitigen Amplitudenspektrums bestimmt.

Bevor dies automatisiert erfolgen kann, müssen die im Signal enthaltenen Störanteile (hoher Gleichanteil, niederfrequente Atem- und Bewegungssignale sowie hochfrequente Anteile wie der Netzbrumm und deren Oberwellen) durch geeignete IIR-Filterung entfernt werden. Als beste Filterkombination erwiesen sich dabei ein IIR-Hochpass mit einer Grenzfrequenz von 2Hz und ein IIR-Tiefpass mit einer Grenzfrequenz von 10Hz. Natürlich ist es über das GUI möglich die Filter zu deaktivieren bzw. die Grenzfrequenzen zu ändern. Dann muss jedoch bedacht werden, dass die Berechnung der Herzfrequenz und in weiterer Folge der Sauerstoffsättigung bei ungeeigneter Filterwahl auch falsche Werte aufweisen können.

Zur Berechnung der gewünschten Parameter wird zunächst das Maximum sowie der zugehörige Frequenzindex des auf den ir-samples basierenden Amplitudenspektrums gesucht (`index_max_ir`). Aus dem gefundenen Frequenzindex kann über die Gleichung 19 (S. 51) die Herzfrequenz in Schlägen pro Minute berechnet und im GUI-Element `edit_hr` ausgegeben werden.

$$\text{Herzrate} = \frac{(\text{index\_max\_ir}) * (\text{Schläge pro Minute}) * (\text{FFT\_ Bandbreite})}{(\text{halbe Länge der FFT})} \tag{19}$$

Dieser Frequenzindex wird dazu verwendet, die Sauerstoffsättigung zu berechnen. Dazu liest man den Arraywert an diesem Frequenzindex aus (für ir und vr) und berechnet das sogenannte Ratio der Maximalwerte über die Gleichung 20 (S. 51). Diese Maximalwerte repräsentieren die Amplitude der Pulswelle, die mit zwei verschiedenen Wellenlängen gemessen wurde.

$$\text{Ratio} = \frac{(\text{fft\_max\_vr})}{(\text{fft\_max\_ir})} \tag{20}$$

Aus diesem Verhältnis wird, analog zur Berechnung im Mikrokontroller, in einem weiteren Schritt über eine Fittingkurve die Sauerstoffsättigung (siehe Gleichung 18, S. 39 und 21, S. 51) berechnet und im GUI-Element `edit_sao2` ausgegeben.

$$\text{SaO2} = \texttt{round}(110.28 - 17.51 * \text{Ratio}) \tag{21}$$

2.6 Versuchsaufbau

Die Messung der Herzfrequenz und Sauerstoffsättigung an einer sedierten Maus ist in Abbildung 24 (S. 52) dargestellt und zeigt übersichtlich die dazu verwendeten Geräte bei einer Messdurchführung. Dazu gehören die entwickelte Hardware, der modifizierte Sensor, eine Induktionskammer (Nr- PS-0347 Rothacher-Medical Gmbh, Bern, Schweiz) zur Narkoseeinleitung, der Schnauzenaufsatz zur Aufrechterhaltung der Narkose (PS-0305-A Rothacher-Medical Gmbh, Bern, Schweiz) und der Isofluranzerstäuber (Isoflurane Vaporizer, Serial-Nr. 081741, Rothacher-Medical Gmbh, Bern, Schweiz) mit einer Sauerstoffzufuhr. Bevor die Messhardware per USB-Kabel an einen PC angeschlossen und der modifizierte Sensor (siehe 2.3.3, S. 17) über die D-SUB-9 Schnittstelle mit der Hardware verbunden wird, muss die Maus mit einem Sauerstoff-Isoflurangemisch ($1.6\frac{l}{7}\text{min}O_2$ bei

einer Isoflurankonzentration von 1.6%) in Tiefschlaf versetzt werden (Dauer: c.a 10min). Mit einem Umschalten des Anästhesiegasgemisches von der Induktionskammer auf den Schnauzenaufsatz und dem Anbringen dieses Schnauzenaufsatzes an der Maus sind die Vorbereitungen für eine Messung abgeschlossen. Zur Messeinleitung ist der Pulsoxymetriesensor an dem Hinterbein oder dem Schwanz durch einen vorsichtigen Schiebevorgang anzubringen und die Hardware per USB-Kabel an einem PC anzuschließen. Der Messvorgang startet, sobald die blaue LED in der Nähe des USB-Steckers durchgehend leuchtet (siehe Abbildung 24, S. 52). Sollte es während dem Messvorgang zu Fehlern in der Ausgabe bzw. Berechnung kommen und werden atypische Werte am LCD dargestellt, kann über den am μP-Entwicklungsboard angebrachten Taster die Software neu gestartet werden.

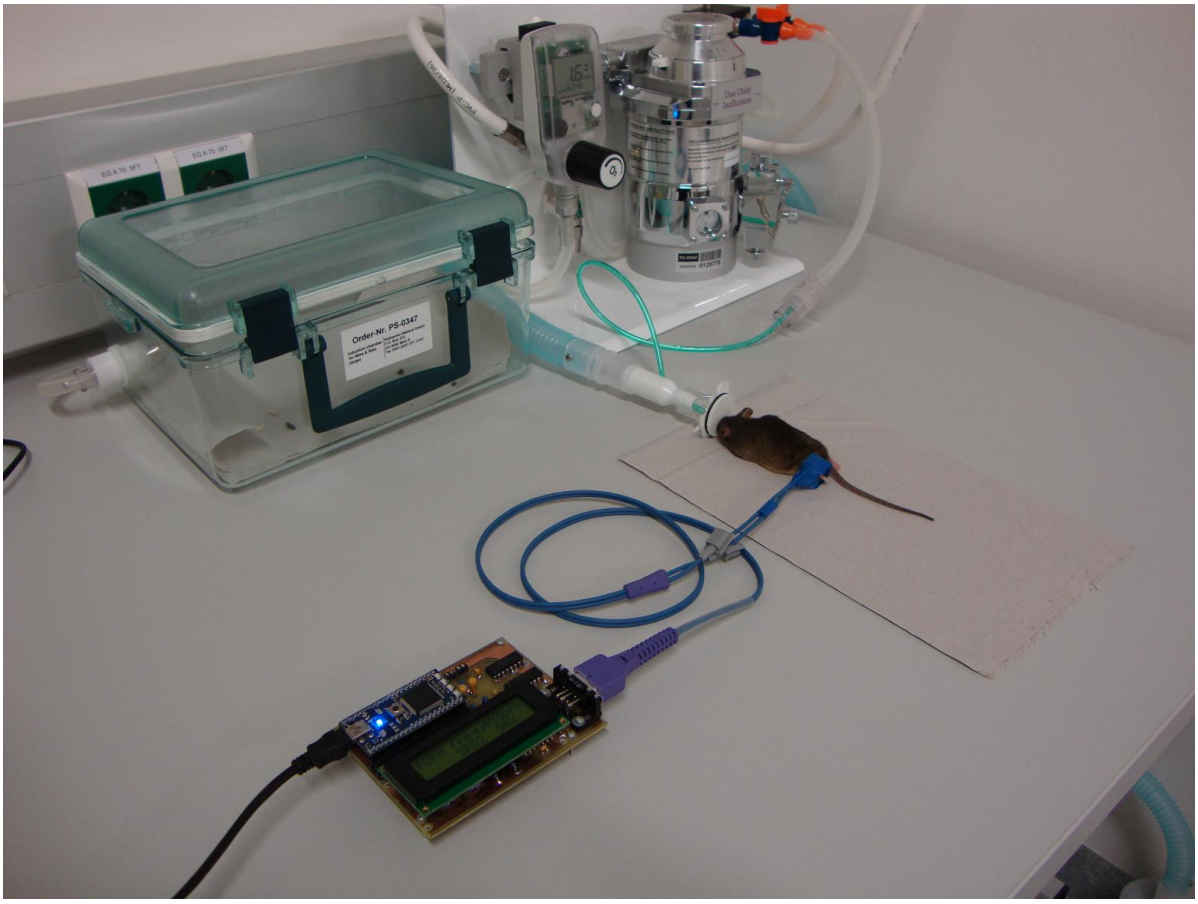


Abbildung 24: Überblick des Versuchsaufbaus zur Messung der Vitalparameter einer sedierten Maus

3 Ergebnisse

Die mit der selbst entwickelten Hardware erzielten Ergebnisse sind in drei Unterpunkte aufgeteilt und sollen visuell die Möglichkeiten und Eigenschaften der Pulsoxymetrie darstellen. Zu diesem Zweck wird zuerst die Hardware-Ausgabe an der Flüssigkristallanzeige näher erläutert, um anschließend die per virtueller serieller Schnittstelle an den PC und in weiterer Folge das MATLAB-GUI übertragenen Signaldaten darzustellen. In einem weiteren Schritt werden anhand der Pulsoxymetrie am Menschen die Hardware- und Softwaremöglichkeiten durch geeignete Datendarstellung visualisiert.

3.1 Hardwareausgabe

In diesem Abschnitt werden die verschiedenen Ausgaben der Hardware zu den Zeitpunkten „Initialisierung“, „Aufnahme der ersten Messdaten“ und „nach Berechnung der Sauerstoffsättigung“ an dem 16x2 LCD dargestellt. Dazu wurden per Digitalkamera im laufenden Messbetrieb Fotos angefertigt, die unter Abbildungen 25 (S. 53), 26 (S. 54) und 27 (S. 54) zu finden sind.

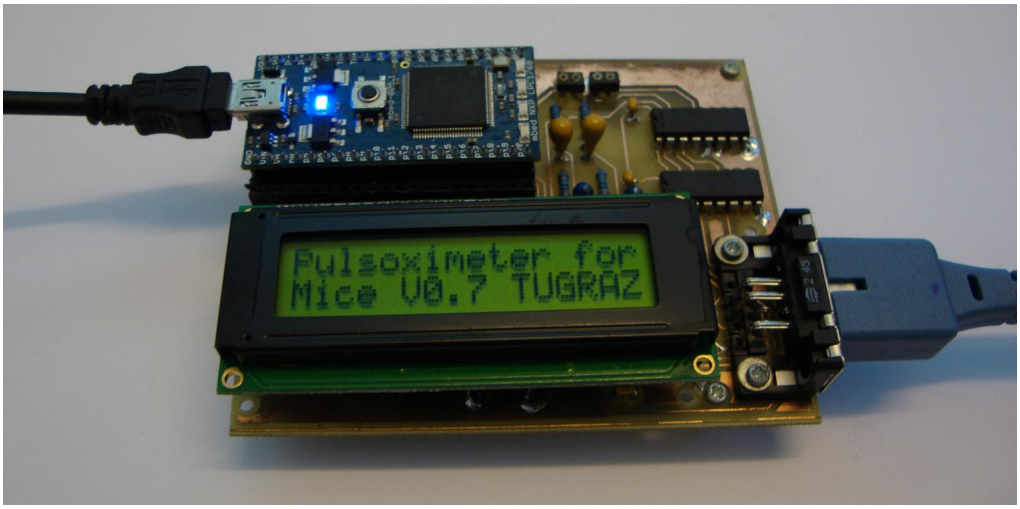


Abbildung 25: Hardwareausgabe während des Initialisierungvorgangs

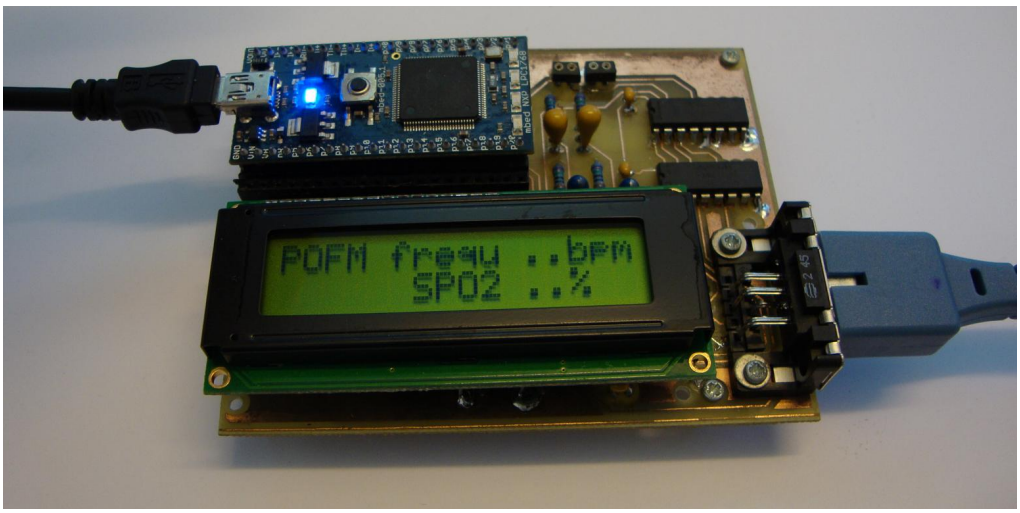


Abbildung 26: Hardwareausgabe während Aufnahme der ersten Messdaten

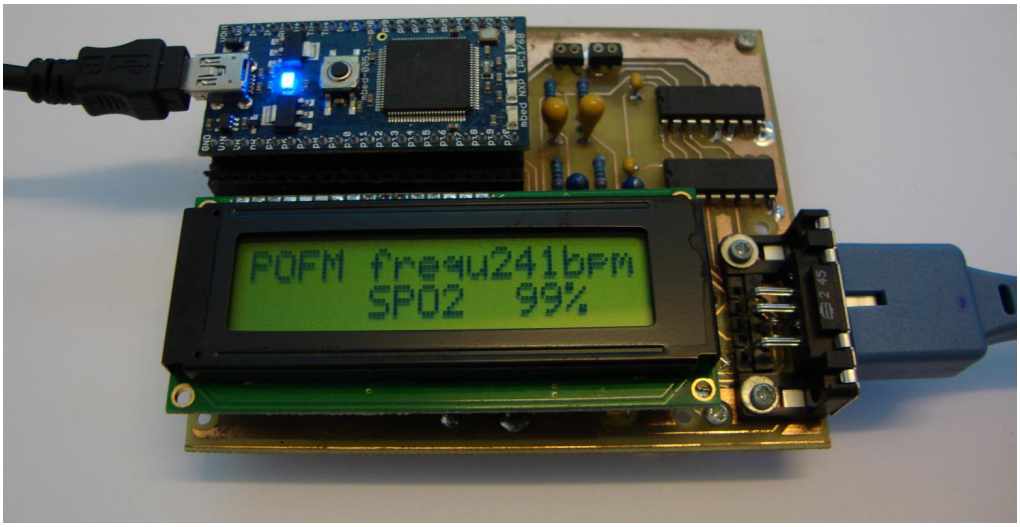


Abbildung 27: Hardwareausgabe im laufenden Messbetrieb

3.2 Messergebnisse am Beispiel eines regulär aufgenommenen Datensatzes

Neben der unter 3.1 (S. 53) dargestellten Ausgabe der berechneten Vitalparameter Herzfrequenz und Sauerstoffsättigung an einem LCD ist es möglich, die Messdaten per USB-Verbindung und ein virtuell erzeugtes serielles Kommunikationsinterface an einen PC zu übertragen. Um dies und die durch den Sensor und in weiterer Folge der Hardware und Mikroprozessorsoftware erzeugten digitalen Signaldaten der Pulswelle bei zwei verschiedenen Wellenlängen darzustellen, wurden diese Signaldaten an das MATLAB-GUI übertragen und ausgewertet. Zu sehen ist in Abbildung 28 (S. 55) ein digital gefiltertes

Zeitsignal und in Abbildung 29 (S. 55) das zugehörige einseitige Amplitudenspektrum. Eine Detailansicht des physiologisch relevanten Frequenzbereichs erfolgt in Abbildung 30 (S. 56).

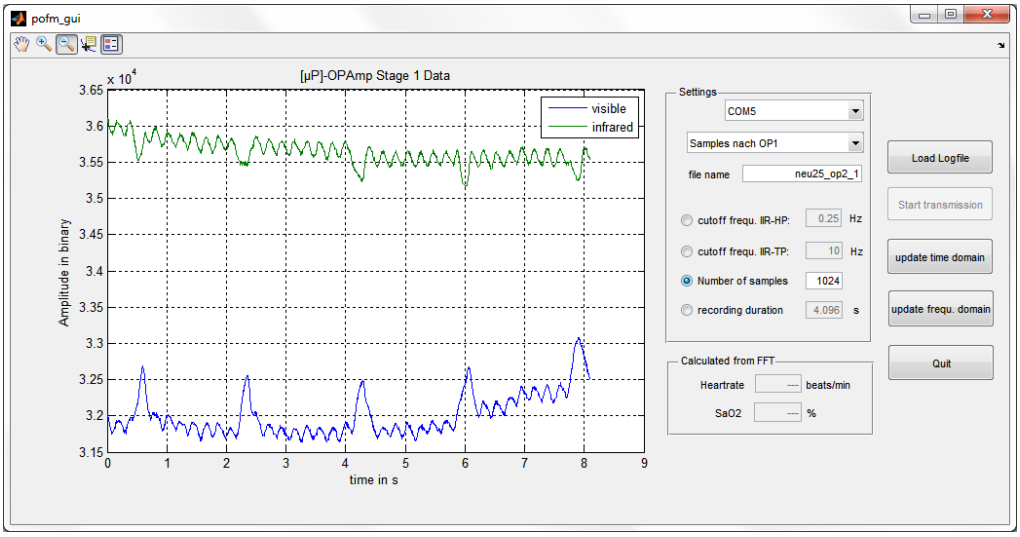


Abbildung 28: Darstellung der vom μP gesendeten und dem MATLAB-GUI empfangenen Pulswellendaten einer sedierten Maus (Samples nach der zweiten OPV-Stufe, grün: Infrarot, blau: sichtbares Rot)

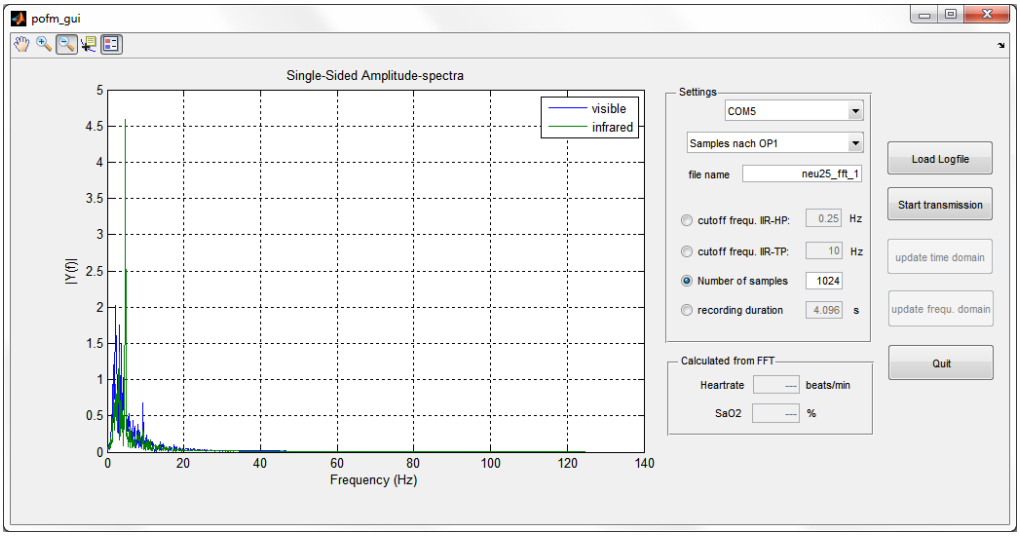


Abbildung 29: Ausgabe des vom μP berechneten únd an den PC übertragenen einseitigen Amplitudenspektrums der Pulswellendaten einer sedierten Maus (grün: Infrarot, blau: sichtbares Rot)

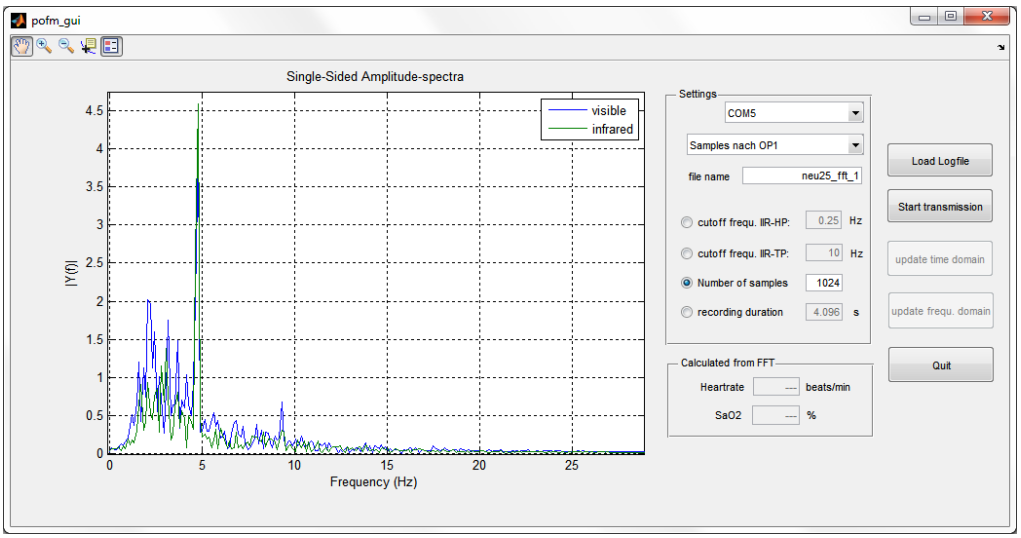


Abbildung 30: Detailausgabe des vom μP berechneten únd an den PC übertragenen einseitigen Amplitudenspektrums der Pulswellendaten einer sedierten Maus (grün: Infrarot, blau: sichtbares Rot)

3.3 Messergebnisse am Beispiel eines artefaktbehafteten Datensatzes

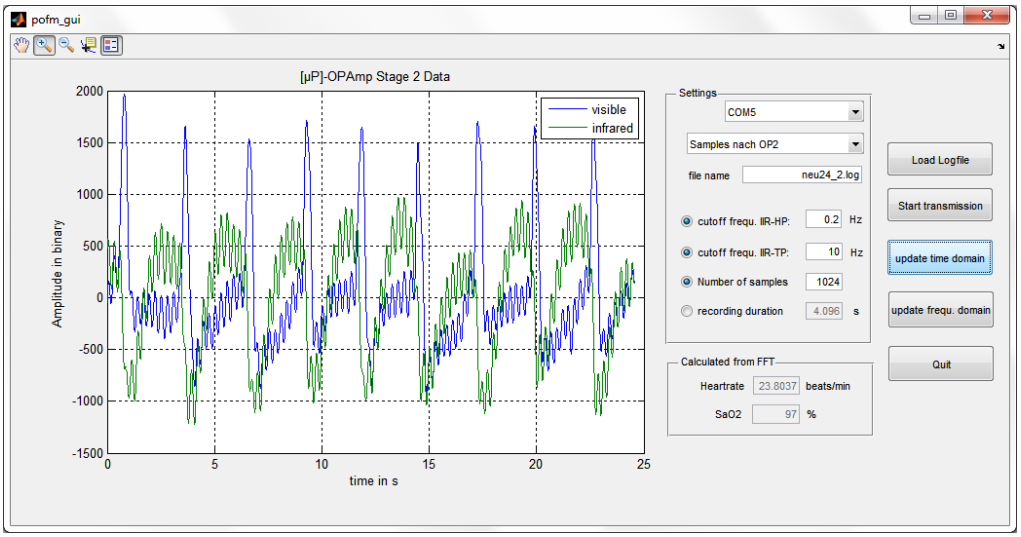


Abbildung 31: Darstellung der vom μP gesendeten Daten (Samples nach der zweiten OPV-Stufe) und per IIR-TP ($f_g = 10Hz$) und IIR-HP ($f_g = 0.2Hz$) gefilterten Daten (grün: Infrarot, blau: sichtbares Rot)) einer sedierten Maus mit ausgeprägter Schnappatmung

Aufgrund stark ausgeprägter Schnappatmung der Maus kann es vorkommen, dass die dabei gemessenen Absorptionssignale sehr stark von einem Atmungssignal überla-

gert sind und dieses die Auswertung beeinträchtigt. Um dies zu visualisieren, ist in den Abbildungen 31 (S. 56), 32 (S. 57) und 33 (S. 58) exemplarisch dargestellt, wie sich ein solches Artefakt auf die Signalqualität auswirkt. Ähnlich zu 3.2 (S. 54) werden die Absorptionssignale vom Mikroprozessor an ein MATLAB-GUI übertragen und dort gefiltert und als Zeitsignal abgebildet. Zusätzlich werden zur besseren Beurteilung auch die Berechnungsergebnisse einer FFT als einseitiges Amplitudenspektrum gesamt und im Detail dargestellt.

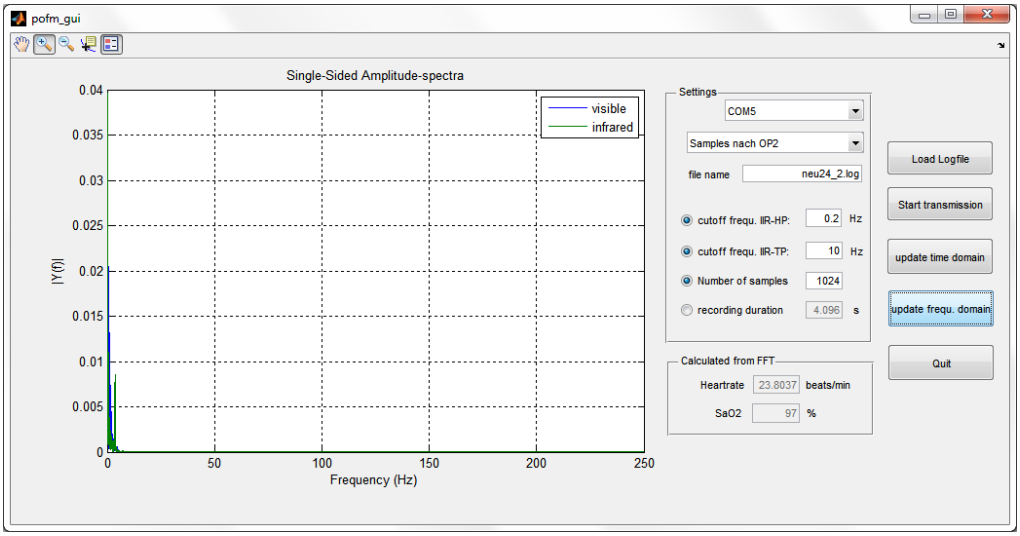


Abbildung 32: Ausgabe des einseitigen Amplitudenspektrums der in Abbildung 28 (S. 55) dargestellten gefilterten Daten (grün: Infrarot, blau: sichtbares Rot) einer sedierten Maus mit ausgeprägter Schnappatmung

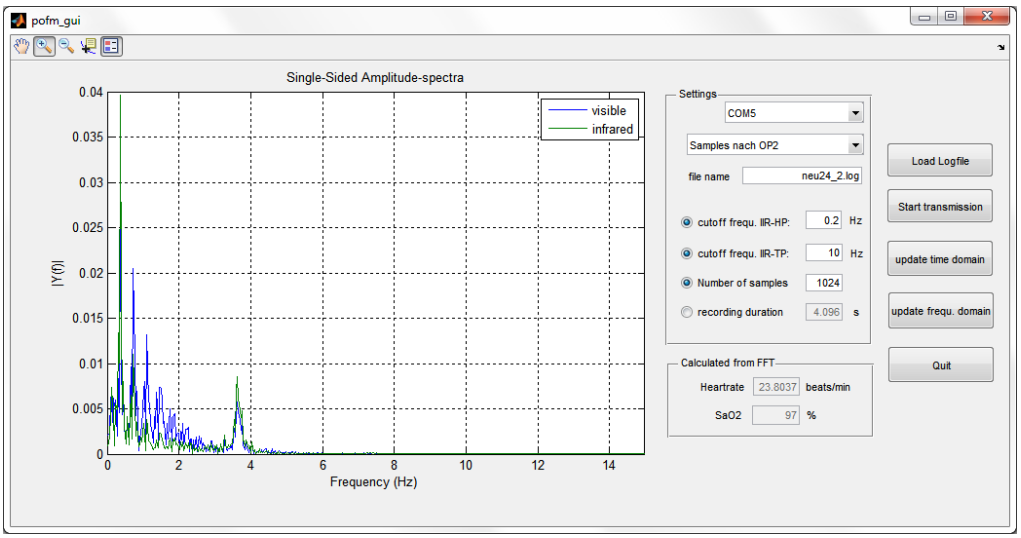


Abbildung 33: Ausgabe des einseitigen Amplitudenspektrums der in Abbildung 28 (S. 55) dargestellten gefilterten Daten (grün: Infrarot, blau: sichtbares Rot) einer sedierten Maus mit ausgeprägter Schnappatmung

3.4 Überprüfung der erzeugten Signale durch Anwendung der Hardware am Menschen

Zur Plausibilitätsüberprüfung der an Mäusen eingesetzten Hardware erfolgte eine Überprüfung durch Messung am Menschen. Dieser Vorgang wird in den Unterpunkten 3.4.1 (S. 58), 3.4.2 (S. 60) und 3.4.3 (S. 62) durch Signalabbildungen visualisiert und in der Diskussion näher erläutert und erklärt.

3.4.1 Überblick über die unterschiedlichen vom Mikroprozessor übertragenen Signaltypen

Die nun folgenden Unterpunkte sollen beispielhaft die verschiedenen vom μP übertragbaren Signaltypen, deren Manipulation und mögliche Komplikationen darstellen. Dazu wurde jeweils ein Screenshot des MATLAB-GUI angefertigt und als Abbildungen angefügt. Die dazugehörige Beschreibung ist der Abbildungsbeschreibung bzw. der Diskussion (siehe 4, S. 64) zu entnehmen. Um eine gewisse Vergleichbarkeit zwischen diesen Abbildungen zu erreichen, wurde bei allen durchgeführten Messungen eine Samplelänge (GUI-Element `edit_number_of_samples`) von 2024 eingestellt. Um einen raschen Überblick der verschiedenen Signaltypen zu erhalten, wurden alle Auswahlmöglichkeiten des GUI-Elements `popup_choose_signal` verwendet, um Signale aus dem μP an das MATLAB-GUI zu übertragen. Diese Ergebnisse sind in den Abbildungen 34 (Samples

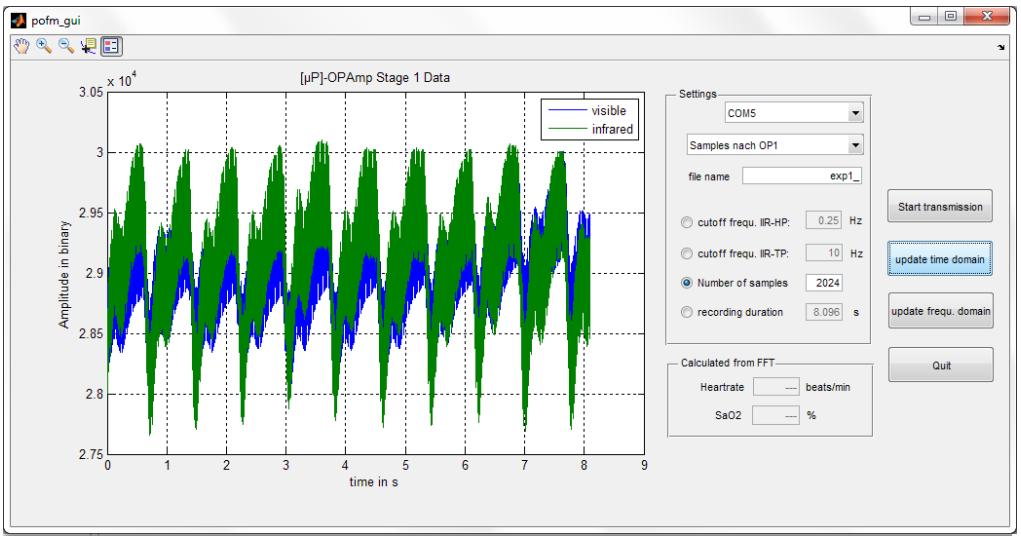


Abbildung 34: Ausgabe der vom μP gesendeten humanen Pulswellendaten - Samples nach der ersten OPV-Stufe (Transimpedanzwandler, grün: Infrarot, blau: sichtbares Rot)

nach OP1, S. 59), 35 (Samples nach OP2, S. 59), 36 (gefilterte Samples, S. 60) und 37 (μP FFT, S. 60) abgebildet und zeigen die jeweiligen Signalcharakteristika.

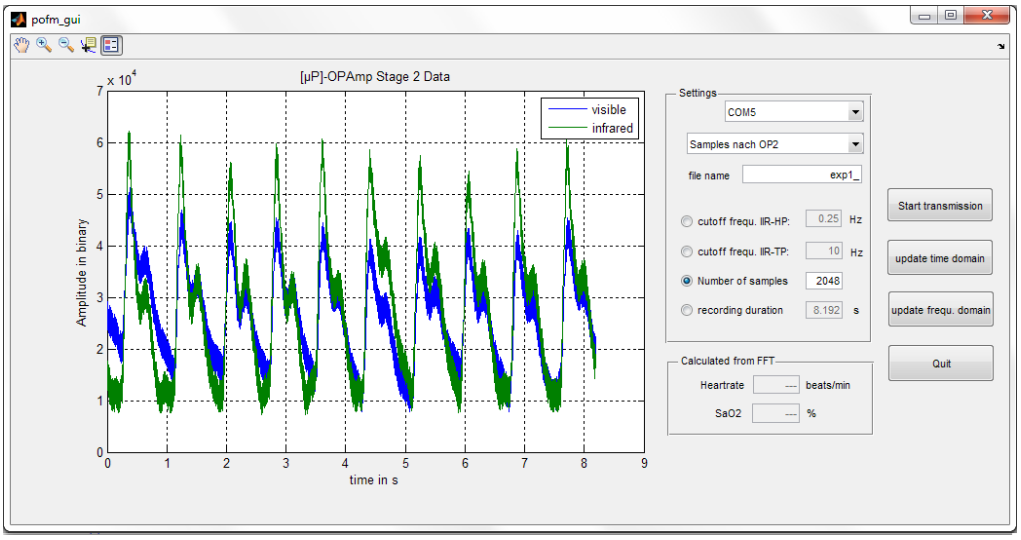


Abbildung 35: Ausgabe der vom μP gesendeten humanen Pulswellendaten - Samples nach der zweiten OPV-Stufe (nicht invertierender Verstärker, grün: Infrarot, blau: sichtbares Rot)

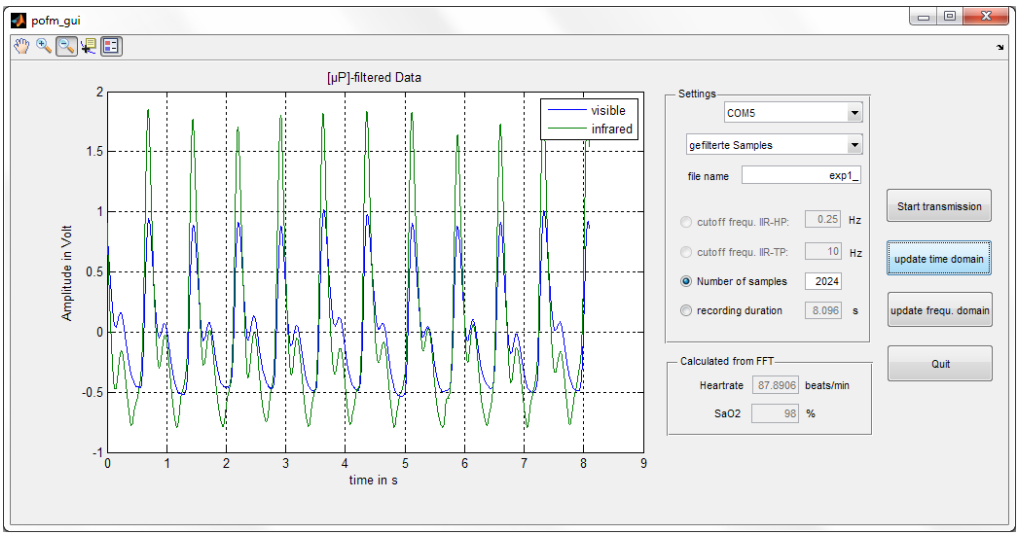


Abbildung 36: Ausgabe der vom μP gesendeten humanen Pulswellendaten - vom μP gefilterte Samples (grün: Infrarot, blau: sichtbares Rot)

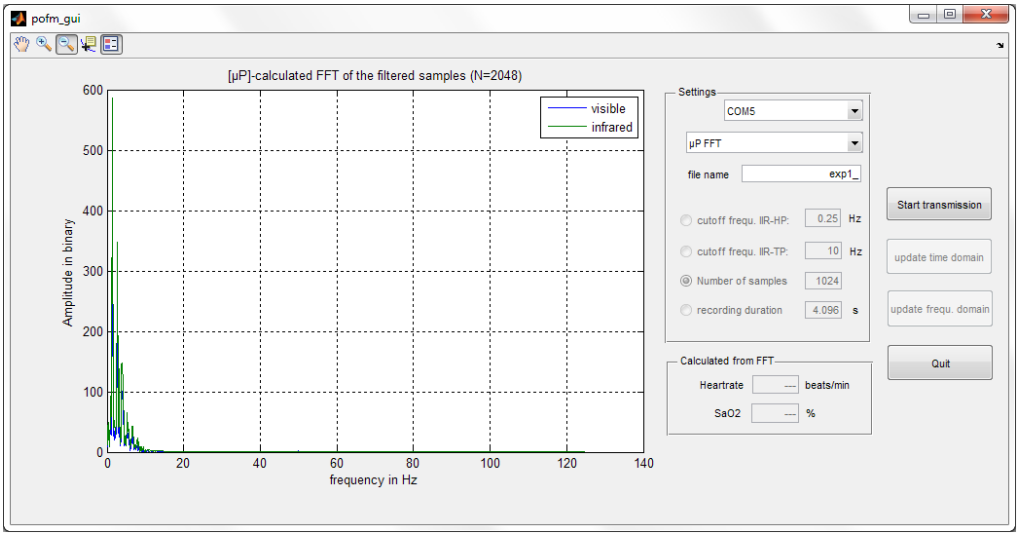


Abbildung 37: Ausgabe der vom μP gesendeten humanen Pulswellendaten - vom μP berechnete FFT (Grundlage liefern 2048 im μ gefilterte Samples, grün: Infrarot, blau: sichtbares Rot)

3.4.2 Signalmanipulation am Beispiel von OP2-Samples in Zeit- und Frequenzbereich

Nach erfolgter Übertragung der Daten bei Auswahl von „`Samples nach OP1`“ oder „`Samples nach OP2`“ muss der User für eine korrekte Berechnung der Herzfrequenz und der Sauerstoffsättigung dieses Datenarray per nachträglicher IIR-Filterung bearbeiten. Dieser Vorgang und deren Auswirkungen sind in den Abbildungen 38 (S. 61), 39

(S. 61) und 40 (S. 62) in Zeit- als auch Frequenzbereich dargestellt. Als Datengrundlage werden dazu die in Abbildung 35 (S. 59) abgebildeten „`Samples nach OP2`“ verwendet.

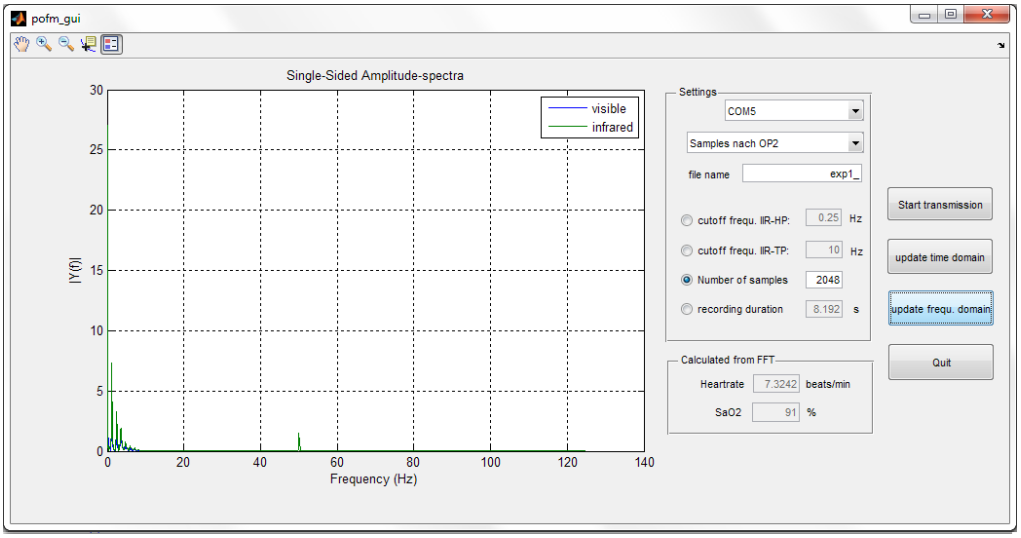


Abbildung 38: Ausgabe des einseitigen Amplitudenspektrums (basierend auf der FFT der in Abbildung 35 (S. 59) dargestellten humanen Pulswellendaten, grün: Infrarot, blau: sichtbares Rot)

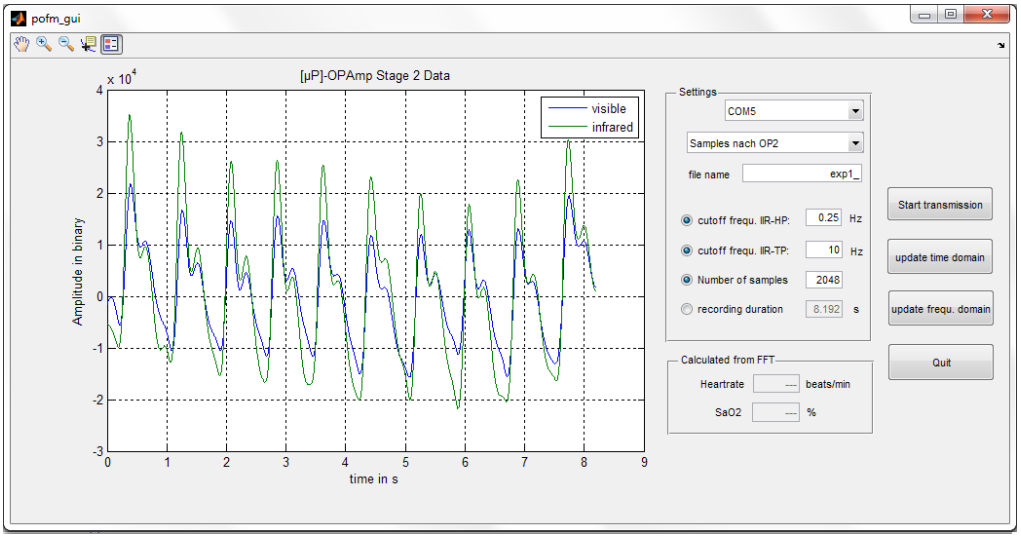


Abbildung 39: Ausgabe der mit einem IIR-TP ($f_g = 10Hz$) und IIR-HP ($f_g = 0.25Hz$) gefilterten humanen Pulswellendaten (basierend auf den in Abbildung 35 (S. 59) dargestellten Daten, grün: Infrarot, blau: sichtbares Rot)

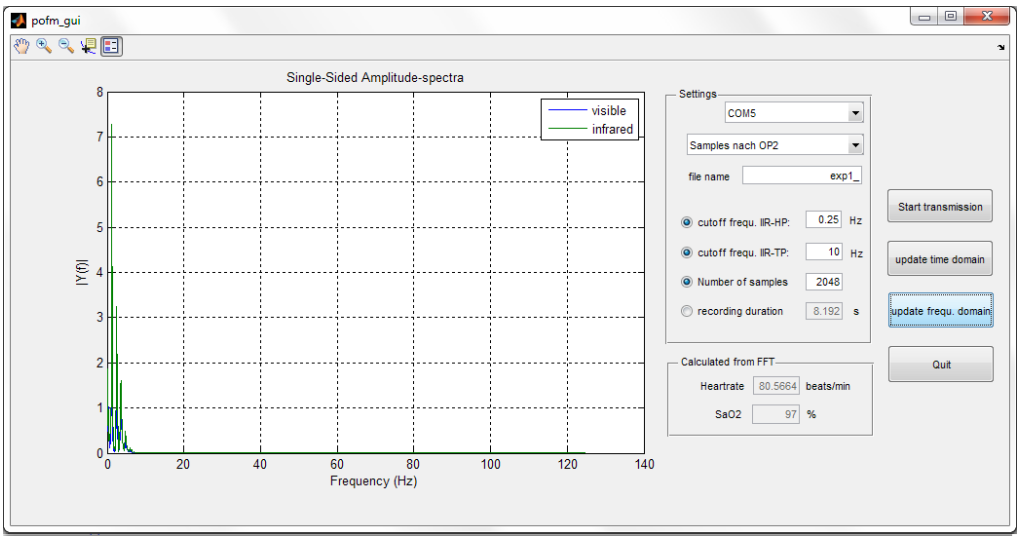


Abbildung 40: Ausgabe des einseitigen Amplitudenspektrums der in Abbildung 39 (S. 61) dargestellten, gefilterten humanen Pulswellendaten (grün: Infrarot, blau: sichtbares Rot)

3.4.3 Artefaktbehaftete Signale

Da es bei unsachgemäßer Anwendung (siehe 4.4.6, S. 75) zu sehr starken Signalartefakten kommen kann, sind die wichtigsten Vertreter in den Abbildungen 41 (zu früh durchgeführte Messung, S. 62), 42 (Sensorabnahme während der Messung, S. 63) und 43 (Bewegung des Fingerclips, S. 63) dargestellt.

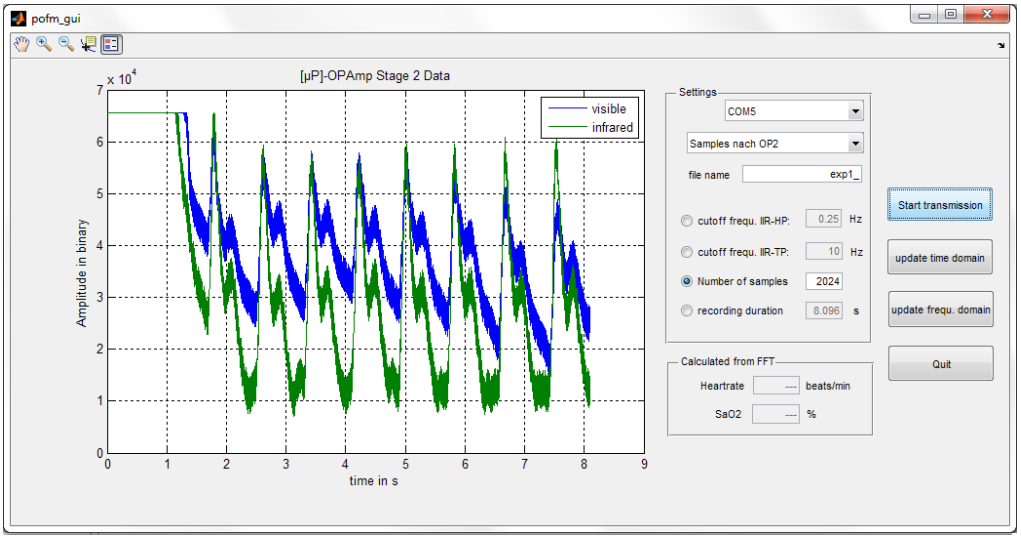


Abbildung 41: Ausgabe der vom μP gesendeten OP2-Daten - Der Signalanfang ergibt sich durch die zu frühe Messung (direkt nach der Sensorapplikation, grün: Infrarot, blau: sichtbares Rot)

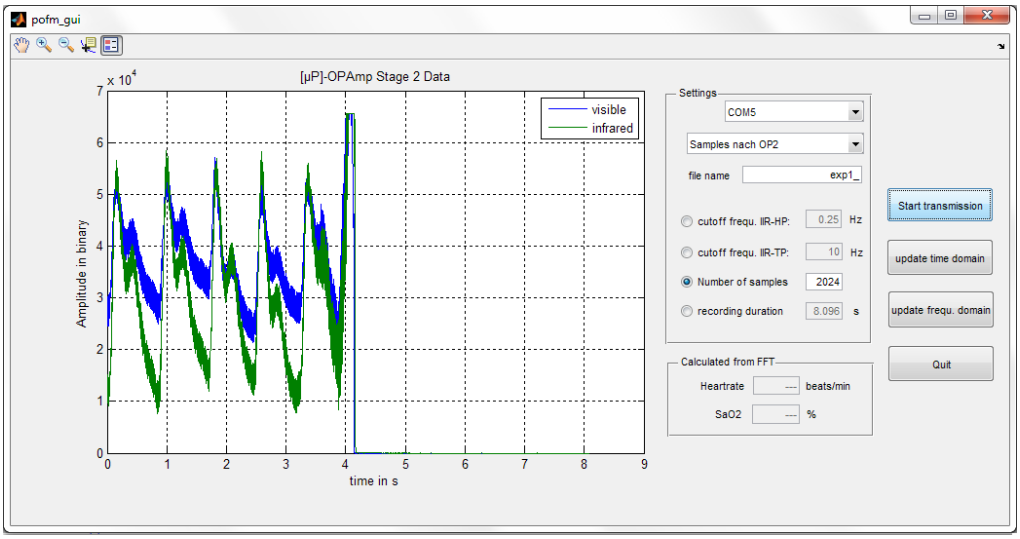


Abbildung 42: Ausgabe der vom μP gesendeten OP2-Daten - Der Signalausfall ergibt sich durch eine Sensorabnahme während der Messung (grün: Infrarot, blau: sichtbares Rot)

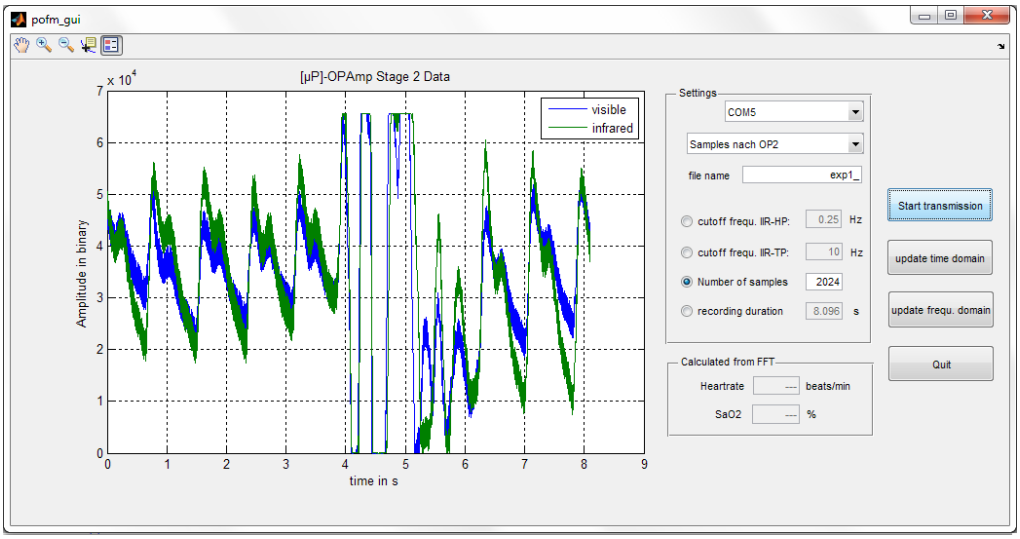


Abbildung 43: Ausgabe der vom μP gesendeten OP2-Daten - Der Artefaktabschnitt zwischen Sekunde 4 und 5 ergibt sich durch eine ausgeprägte Bewegung des Fingerclips während der Messung (grün: Infrarot, blau: sichtbares Rot)

4 Diskussion

Das abschließende Kapitel Diskussion erläutert, zur besseren Orientierung mit einer zum Methodenkapitel vergleichbaren Gliederung, die Überlegungen, Probleme und Eigenschaften der entwickelten Hardware (4.1, S. 64), Mikroprozessor- (4.2, S. 68) und MATLAB-Software (4.3, S. 70). Als weiterer Punkt wird unter 4.5 (S. 76) auf eine mögliche, aus Zeit- und Aufwandsgründen im Rahmen dieser Masterarbeit jedoch nicht durchgeführte Validierung der Messergebnisse eingegangen. Am Ende dieses Kapitels erfolgen eine kritische Betrachtung der entwickelten Messvorrichtung sowie Verbesserungsvorschläge für spätere Projekte.

4.1 Hardware

Wie in 2.2 (S. 13) bereits beschrieben, sind die zu erwartenden Absorptionssignale bei der Messung an einer Maus um ein Vielfaches geringer als die ohnehin schon geringen Amplituden beim Einsatz am Menschen (in der Literatur wird der Anteil der Wechselgröße am Gesamtsignal im einstelligen Prozentbereich angegeben [5]). Dies erfordert es, beim Hardwaredesign vor allem auf die Signalqualität sowie die durch geringe Nutzsignalamplituden stärker in Erscheinung tretenden Stör- und Rauschquellen zu beachten und diese nach Möglichkeit zu minimieren, um das Signal-Rausch-Verhältnis auf einem auswertbaren Niveau zu halten. Die hohe Herzrate der Maus (siehe Tabelle 1, S. 13) und die daraus resultierende Bandbreite des Nutzsignals (siehe Tabelle 2, S. 14) erfordert es diese bei der Signalakquise wie auch bei der analogen und digitalen Signalverarbeitung zu berücksichtigen. Ausgehend von in Literatur und Internet veröffentlichten Blockschalt- und Stromlaufplänen ([5], [17], [18] und [19]) für Messhardware zur Anwendung der Pulsoxymetrie am Menschen, wird auf diesen Überlegungen basierend eine Messschaltung entworfen, die die Absorptionssignale von sedierten Mäusen druchführen und eine Berechnung der Sauerstoffsättigung ermöglicht.

4.1.1 Zeitmanagement

Das in den Methoden vorgestellte Zeitprofil ist das Ergebnis eines langen Entwicklungsprozesses, dessen Ziele, eine möglichst gute zeitliche Auflösung als auch ein stabiles Berechnungsergebnis, erst nach einigen Monaten erfolgreich erreicht werden konnten. Die dazu getroffenen Anpassungen, wie eine Verkürzung oder Verlängerung der Periodendauer oder die Wahl der Abtastzeitpunkte, ergeben sich durch im frühen Entwicklungs-

stadium fehlende Erfahrung der zur erfolgreichen Berechnung der Sauerstoffsättigung notwendigen μP-Rechenzeit, einer fehlerhaften Einschätzung der FFT-Fenstergröße und eine zu genau geforderte zeitliche Auflösung. Als beste Wahl der Abtastzeitpunkte hat sich das Ende jedes Stomimpulses herausgestellt, da dadurch die Einflüsse der andere LED am besten unterdrückt werden. Messungen mit dem ersten funktionstüchtigen Prototypen zeigten bald, dass die ursprüngliche Abtastfrequenz von 1000Hz die Leistungsfähigkeit des μPs übersteigt und während den `wait`-Kommandos nicht genügend Rechenzeit vorhanden ist, um die FFT-Berechnung, die LCD-Ausgabe und die serielle Kommunikation unterzubringen. Die dadurch folgende Halbierung der Abtastfrequenz auf 500Hz behob dieses Problem und wurde bis knapp vor Ende des Entwicklungsprozesses eingesetzt. Durch den begrenzten RAM von 32kB ergibt sich bei der FFT-Berechnung eine maximale Anzahl an Zeitsamples von 2048. Daraus folgen bei einer Abtastfrequenz von 500Hz eine zeitliche Länge von 4.096s und eine Frequenzauflösung von 0.244Hz (bzw. 14.65bpm). Diese Auflösung wurde als zu klein eingestuft und kann nur durch eine Vergrößerung des Fensters erzielt werden. Da der Speicherplatz aber begrenzt ist, musste die Abtastfrequenz erneut auf 250Hz halbiert werden, was das Fenster bei gleicher Anzahl an Zeitsamples auf 8.19s erhöht. Neben einer verbesserten Frequenzauflösung von 0.122Hz (bzw. 7.32bpm) werden für die FFT-Berechnung mehr Pulswellen verwendet, was zudem für eine stabilere Berechnung sorgt.

4.1.2 LED-Treiberstufe

Die in 2.3.6 (S. 24) ausgeführte LED-Treiberstufe wurde, basierend auf einem Anwendungsbeispiel von TI (Texas Instruments, Dallas, USA) [17], entworfen und dient der Umwandlung eines Gleichspannungssignals in einen dazu proportionalen Strom mit alternierendem Vorzeichen um die antiparallel verschalteten LEDs ansteuern zu können. Die Regelung der Stromstärke erfolgt in einem theoretischen Wertebereich von ca. 0.66V bis 1.9V der sich aus der Versorgungsspannung VCC = 3.3V und den Spannungsabfällen an den eingesetzten Halbleiterelementen (0.66V für den NPN- und PNP-Transistor) ergibt. Diese Spannung wird von Transistor D2 oder D3 über den Basiswiderstand verstärkt und in einen proportionalen Strom umgewandelt. Eine durchgeführte PSpice Simulation brachte als Ergebnis einen Wertebereich von 1.2mA bei 0.66V bis zu 52.9mA bei 1.9V. Basierend auf diesen theoretischen Erkenntnissen erfolgte eine Validierung dieser Werte auf einem Steckboard, die ergab, dass sich der dynamisch einstellbare Strom annähernd linear zwischen 2mA bei 0.66V und 50.8mA bei 2V einstellen lässt. Über eine limitierung der Regelgröße in der Software ist es möglich diesen Wertebereich auf

den linearen Bereich einzuschränken und es ergibt sich schlussendlich ein Bereich von $0.66V$ bis 1.9V (2mA bis 43.5mA). Über das gepulste Auswahlsignal und die Brückenkonstruktion wird abwechselnd ein Strom mit unterschiedlichem Vorzeichen erzeugt, der über eine der beiden LEDs fließt. Dieses On-Off-Verhalten ermöglicht es, durch Bildung eines zeitlichen Mittelwerts den in Internetforen gefundenen Maximalstrom von 20mA auszureizen.

4.1.3 Verstärkerstufe

Die Aufgabe des Empfangspfads ist es, die vom Phototransistor in ein Stromsignal umgeformten Absorptionssignale zu verstärken und in ihre Bestandteile aufzutrennen. Dazu wird eine Kaskade von Transimpedanzverstärker, Demultiplexer, passive Filter und zwei invertierenden Verstärker verwendet. Die unipolare Versorgungsspannung des Transimpedanzwandlers ermöglicht eine Versorgung der vom μP-Entwicklungsboard bereitgestellten $3.3V$, erfordert es aber, um auch geringe Ströme messen zu können (diese würden unter die Rail-To-Rail-Aussteuerbarkeit des OPV fallen), den Kollektor des Phototransistors per Spannungsteiler auf ein von GND ausreichend unterschiedliches Spannungsniveau zu heben. Im vorliegenden Fall wird der „virtuelle“ Nullpunkt des Phototransistors über zwei Widerstände mit 100kΩ und 15kΩ realisiert und beträgt 0.495V bei einer Versorgungsspannung von VCC = 3.3V. Um diesen Nullpunkt wird das Stromsignal über den Widerstand im Rückkopplungspfad verstärkt und von dem parallel geschalteten Kondensator bandbegrenzt.

Rückkopplungswiderstand des Transimpedanzwandlers Um den Rauscheinfluss der ersten Verstärkerstufe auf die nachfolgende Schaltung zu minimieren, ist es zielführend, eine möglichst große Verstärkung zu wählen. Zwar steigt das thermische Rauschen mit dem Rückkopplungswiderstand, jedoch geschieht dies nur mit der Wurzel des Widerstandswerts. Dies bedeutet, dass das ohnehin zu verstärkende Signal bei der Wahl eines großen Widerstands geringer rauscht als bei mehrmaliger Verstärkung mit kleineren Widerständen. Um die Einflüsse des thermischen Rauschens zu minimieren und eine ausreichende Verstärkung zu erzielen, wird der Widerstandwert auf $\mathrm{R_f} = 2.2 * 10^6\Omega$ festgesetzt. Unter Andwendung der Gleichung 22 (S. 66) hat dies einen Umwandlungs- oder Verstärkungsfaktor von $2.2 * 10^6$ zur Folge.

$$U_a = -I * R_f \tag{22}$$

Feedback-Kondensator Zur Bandbegrenzung des verstärkten Signals und um die Phototransistorkapazität auszugleichen und damit eine stabile Verstärkung zu erzielen, wird im Rückkopplungspfad parallel zum Widerstand ein Kondensator eingesetzt. Dessen Wert ist vom Gain–Bandwidth Produkt des OPVs (GWB) als auch von der Eingangskapazität C_I und dem Rückkopplungswiderstand abhängig. Unter Anwendung von Gleichung 23 (S. 67) kann der minimale Kapazitätswert des Rückkopplungskondensators bestimmt werden. Der eingesetzte Kondensator liegt mit einem Wert von $\mathrm{C_f} = 4.7 * 10^{-12}\mathrm{F}$ deutlich über dem minimalen Wert von $0.674 * 10^{-12}\mathrm{F}$ und erfüllt daher - in Kombination mit dem Rückkopplungswiderstand $\mathrm{R_f} = 2.2 * 10^{6}\Omega$ - das Stabilitätskriterium.

$$C_f \geq \sqrt{\frac{C_I}{GWBR_f}} \tag{23}$$

Das über den Transimpedanzwandler nach Abbildung 14 (S. 27) erzeugte Spannungssignal das wie in Abbildung 2.3.5 (S. 21) aus sequenziell und abwechselnd angeordneten Teilsignalen besteht, wird unter Einsatz des Umschalterbausteins „MAX4619“ (Maxim Integrated Products, Sunnyvale, USA) [25] durch Ansteuerung des Mikrokontrollers über zwei I/O-Ports in die zwei Teilsignale zerlegt. Über jeweils zwei Tiefpässe zweiter Ordnung mit einer Grenzfrequenz von $f_g = 64.3Hz$ und einem anschließenden Spannungsfolger werden einem Demodulator ähnlich die beiden Teilsignale in ihren Gleichspannungsanteil demoduliert. Dieses abstrakte Konzept basiert auf der Überlegung, dass der Umschalter zwischen den beiden Signalpfaden das zugehörige Spannungssignal des Transimpedanzwandlers mit derselben Frequenz schaltet, mit der die sequenzielle Kodierung des Signals der Sendedioden erzeugt wurde. Während ein Signalpfad durch den Umschalter an den Ausgang des Transimpedanzwandlers angeschlossen ist, das Signal über den Tiefpass gefiltert wird und die Kondensatoren der Ausgangsspannung folgen, hängt der andere Signalpfad „in der Luft“ und kann sich bedingt durch die großen Kapazitäten und den Impedanzwandler nicht entladen. Die Kondensatoren halten das Spannungsniveau, bis über den Umschalter dieser Signalpfad an den Transimpedanzwandler angeschlossen wird und das Signal über den Tiefpass und Spannungsfolger übertragen wird, während der andere Pfad „in der Luft hängt“. Dadurch entstehen zwei gefilterte quasikontinuierliche Spannungssignale, die den Absorptionswerten für die durch Aktivierung der IR-LED bzw. VR-LED ins Messobjekt eingestrahlte und abgeschwächte Intensitäten entsprechen und denen als Wechselanteil das Pulssignal überlagert ist (höchste vorkommende Nutzfrequenz ist 7.5Hz, siehe 2, S. 14).
Die dabei erzeugten Spannungssignale werden durch μP-interne 12-bit A/D-Wandler

digitalisiert und zur LED-Intensitätsregelung verwendet (siehe 2.4.6, S. 36). Die Wechselanteile werden durch aktive Regelung des Mikrokontrollers in den zwei Signalpfaden symmetrisch durch invertierende Verstärker mit integriertem aktiven TP erster Ordnung ($f_g = 8.12Hz$) gefiltert und um den Faktor V = 40 verstärkt. Die Regelung durch den Mikrokontroller erfolgt über zwei von Tiefpässen erster Ordnung ($f_g = 64.3Hz$) gefilterte PWM-Ausgänge (Zyklusfrequenz von $f = 10000Hz$), die als Offsetspannung an den positiven Eingang des OPVs angeschlossen sind. Über den Duty-Cycle des PWM-Ausgangs kann der Mikrokontroller den Gleichanteil des zu verstärkenden Spannungssignals nachbilden und somit, symmetrisch um diese Offsetspannung, nur das Wechselsignal verstärken. Diese am Ausgang des jeweiligen invertierenden Verstärkers vorliegenden Spannungssignale - mit verstärktem Wechselanteil - werden über μP-interne 12-bit A/D-Wandler abwechselnd digitalisiert und können für die Berechnung der Sauerstoffsättigung verwendet werden.

4.2 Mikroprozessor-Software

Die Hauptaufgabe der μP-Software ist es, die LED-Intensitäten zu regeln, die verstärkten Absorptionssignale zu digitalisieren und diese über digitale Signalverarbeitung zur Berechnung der Vitalparameter Herzfrequenz und Sauerstoffsättigung heranzuziehen.

4.2.1 Regelung der LED-Intensitäten

Die Regelung der LED-Ströme erfolgt über einen PWM-Ausgang, dessen Duty-Cycle vergrößert bzw. verringert werden, bis die vom Transimpedanzverstärker umgesetzten Absorptionssignale sich im binären Bereich 30000 (entspricht $1,51$V) und 33000 (entspricht $1,66$V) befinden. Dadurch wird erreicht, dass der statische Anteil einen vergleichbaren Wert aufweist und in weiterer Folge nur noch der Wechselanteil verstärkt werden muss (siehe 2.1.5, S. 12).
Um bestmöglich den Wechselanteil dieser Absorptionssignale zu verstärken, werden die positiven Eingänge der invertierenden Verstärkerschaltungen an die Ausgänge von Tiefpässen geglätteter PWM-Signale geschlossen, was durch Variation des Duty-Cycles den Offset auf den Gleichanteil des jeweiligen Signals einstellt. Dies hat zur Folge, dass nur der um dieses Offsetsignal variable Wechselanteil verstärkt wird. Die dabei erzeugten Signale werden über im μP-Entwicklungsboard eingebaute A/D-Wandler digitalisiert und stehen zyklisch zur Signalverarbeitung bereit.

4.2.2 Signalverarbeitung der Messdaten

Das Design und die Berechnung der verwendeten digitalen IIR-Filterkoeffizienten erfolgt über die MATLAB (Mathworks Inc., Natwick, USA)-Funktion `butter`. Unter Angabe von Filterordnung, Grenzfrequenz und Samplerate berechnet diese Funktion die Pol- und Nullstellen des Filters und die daraus resultierenden Filterkoeffizienten. Die Bandbreite des Nutzsignals wird in Tabelle 2 (S. 14) mit $3.3 - 7.5$Hz angegeben, was dazu führt, dass die umliegenden Frequenzbereiche für die Berechnung nicht benötigt und problemlos entfernt werden können. Zu diesem Zweck wurden zwei IIR-Filter 3.Ordnung mit MATLAB (Mathworks Inc., Natwick, USA) und der Funktion `butter` entworfen und per C-Funktion im Mikrokontrollercode integriert (Direktform II). Die Implementierung der IIR-Filter basiert auf den Angaben in der Literatur [29] und es ist erforderlich durch die als static definierten Signalarrays für jeden eingesetzten Filter eine eigene Funktion zu schreiben. (siehe Funktionen `ir_iir_filter`, `vr_iir_filter`, `ir_hp_iir_filter` und `ir_hp_iir_filter`). Um Rundungsfehler bei der Anwendung und Berechnung der gefilterten Zeitsignale zu minimieren, erfolgt die Berechnung des Filters trotz erhöhtem Rechenaufwand mit Gleitkommakoeffizienten.
Die Signalbeschaffenheit (reelle Datensamples) erlaubt es, eine Eigenschaft der FT-auszunützen. Ist die FT von reellen Daten zu bestimmen, kann dies durch die reelle Frequenzhälfte der komplexen FT geschehen (siehe Funktion `vRealFFT)`. Dabei reduziert sich der Rechenaufwand als auch der Speicherbedarf bei gleicher Frequenzauflösung und Genauigkeit um 50%. Die somit bestimmte reelle FT wird in ein Betragsspektrum umgewandelt und steht zur Signalanalyse bereit.
Die Berechnung der Vitalparameter basiert auf einer Ratiobildung der gefundenen Pulswellenmaxima für die Sauerstoffstättigung und einer Umwandlung des zugehörigen Frequenzindex für die Herzfrequenz. Während die Herzfrequenz ohne große Probleme berechnet werden kann (siehe Gleichung 16, S. 38), ist es notwendig, eine empirisch ermittelte Funktionsgleichung zur korrekten Bestimmung der Sauerstoffsättigung zu verwenden (siehe 2.1.5, S. 12). Aufgrund des enormen Aufwands den die Erstellung einer eigenen Fittingkurve bedeutet (siehe 4.5, S. 76), ist an diesem Punkt anzumerken, dass hierfür auf einen Look-up-Table eines im Internet veröffentlichten Pulsoximeterprojekts zurückgegriffen wird [17].

4.3 MATLAB-Software

Dieser Abschnitt befasst sich zum Einen mit grundlegenden Anmerkungen für einen ordnungsgemäßen Gebrauch der Hardware und zum Anderen werden die in Abschnitt 3 (S. 53) dargestellten Ergebnisse näher beschrieben.

4.3.1 Grundlegende Anmerkungen zum erfolgreichen Messvorgang

Zur erfolgreichen Datenaufzeichnung der vom μP übertragenen Daten sind mehrere Faktoren ausschlaggebend. Vor dem MATLAB-Start und der eigentlichen Messung ist die Hardware über das beigelegte USB-Kabel mit dem PC zu verbinden. Da während der MATLAB-Initialisierung die verfügbaren COM-Ports abgefragt werden, führt ein Start der MATLAB-GUI `pofm_gui.m` bei nicht verbundener Hardware - auch bei einem nachträglichen Anstecken - zu einem Programmabbruch mit zugehörigem Pop-up-Fenster. Um sicherzugehen, dass der PC die Hardware korrekt erkannt hat und die virtuelle Schnittstelle eingerichtet wurde, empfiehlt es sich, in der Systemsteuerung unter „Harware und Sound“ die verfügbaren „Geräte und Drucker“ anzuzeigen. Erscheint dort ein Eintrag mit dem Namen „mbed Serial Port (COM..)“ kann MATLAB und im Anschluss die MATLAB-GUI `pofm_gui.m` gestartet werden.
Verbindet man den `Nellcor DS-100A` Fingerclipsensor über den 9 poligen D-SUB-Stecker und appliziert diesen an einem Finger (vorzugsweise der Zeigefinger der rechten oder linken Hand), beginnt die Datenauswertung über den μP und die Daten können per MATLAB-GUI an den PC übertragen werden. Dabei ist anzumerken, dass aufgrund von Regelprozessen (Anpassung der LED-Ströme) es rund eine Sekunde dauert bis ein auswertbares Signal messbar ist (siehe dazu 4.4.6, S. 75). Über die GUI-Elemente können der Signaltyp, die Anzahl der zu übertragenden Datensamples sowie etwaige Filter aktiviert und eingestellt werden um anschließend über den Button `button_start` die Datenübertragung zu starten. Die dabei empfangenen und abgespeicherten Samples stehen über das Logfile für eine weitere Manipulation bereit (siehe 2.5.1, S. 45).
Ist der Sensor am Finger appliziert, soll die Hand und im Speziellen der Sensor nach Möglichkeit nicht bewegt werden. Dazu bietet sich an, die Hand ausgestreckt auf den Tisch zu legen und mit der anderen Hand das GUI zu bedienen. Des Weiteren ist darauf hinzuweisen, dass sich Störlicht negativ auf die Signalqualität auswirkt. Es soll daher versucht werden, das Umgebungslicht auf ein Minimum zu reduzieren - vor allem direkte Sonneneinstrahlung ist durch Abdunkelung zu verhindern.
Aufgrund der hohen Baudrate von 460800 (siehe Tabelle 7, S. 40) kann es während der

Datenübertragung zu Übertragungsfehlern kommen, die dem Benutzer als MATLAB-Error angezeigt werden.

- `Open failed: StopBits could not be set to the specified value.`
- `Open failed: DataBits could not be set to the specified value.`
- `Open failed: ParityBit could not be set to the specified value.`

Nach der Error-Ausgabe kann durch erneute Aktivierung des Buttons `button_start` versucht werden, erneut eine Verbindung mit der Hardware aufzubauen.
Wird bei erneuter Ausführung der Fehler „`Open failed: Port: COM5 is not available. Available ports: COM5.`“ ausgegeben, ist es zu Buffer-Problemen auf der Seite des μPs oder auf PC-Seite gekommen. Dies hat zur Folge, dass die Software MATLAB komplett beendet und die Hardware über den USB-Stecker aus- und wieder eingesteckt werden muss. Danach initialisiert sich der COM-Port und MATLAB sowie das GUI `pofm_gui.m` können erneut ausgeführt werden.

4.4 Ergebnisse

Die durch die Hardware erzeugten Ergebnisse teilen sich grundsätzlich in zwei Bereiche. Während eine Ausgabe der Vitalparameter über das an der Messhardware angebrachte 16x2 LCD erfolgt, können die digital im Mikrokontroller vorliegenden Daten per seriellem Interface an einen PC übertragen werden. Die dabei erzeugten Ergebnisse sind als Digitalfotos bzw. Screenshots im Kapitel Ergebnisse abgebildet. Um die Plausibilität der erzeugten Maussignale zu überprüfen, wurden zudem humane Messungen durchgeführt und mit MATLAB ausgewertet. Deren Signalformen sind ebenfalls im Kapitel Ergebnisse unter 3 (S. 40) zu finden und werden in diesem Kapitel näher erläutert.

4.4.1 Hardwareausgabe

Die Hardwareausgabe über die Flüssigkristallanzeige wird über die Software des μPs über sechs I/O-Ports gesteuert und ermöglicht es dem Benutzer die zyklisch berechneten Vitalparameter Herzfrequenz und Sauerstoffsättigung bequem abzulesen. Zur Beschreibung der drei unterschiedlichen Programmabschnitte sind die am LCD angezeigten Ausgaben mit den Abbildungen 25 (S. 53), 26 (S. 54) und 27 (S. 54) gut dargestellt und zeigen den μP-internen Berechnungsstand. Direkt nach dem Programmstart erfolgt die Initialisierung der Mikroprozessorkomponenten und es wird genau eine Sekunde gewar-

tet, bis sich die Potentiale der OPVs und der TPs stabilisiert haben. Während dessen erfolgt die Ausgabe des Initialisierungstextes „Pulsoximeter for Mice V0.7 TUGRAZ“ wie in Abbildung 25 (S. 53) gut erkennbar ist.
Nach der Wartezeit beginnt der Mikrokontroller mit der Regelung der LED-Intensitäten und nimmt zyklisch in $4 * 10^{-}3$s Abständen zwei Samples (für infrarot und sichtbares rotes Licht) auf. Gleichzeitig erfolgt die digitale Filterung und die Zeitsignale werden in zwei 2048 großen `float`-Arrays abgelegt. Nach etwa 8.1s sind die beiden Arrays befüllt und eine 2048-Punkte FFT kann berechnet werden. Daraus bestimmt die μP-Software die Herzfrequenz und Sauerstoffsättigung. Während zur erstmaligen Ausgabe der Sauerstoffsättigung vier volle FFT-Perioden vorliegen müssen (Ausgabe eines gleitenden Mittelwerts über vier Berechnungsergebnisse), wird die Herzfrequenz für jede FFT-Berechnung am LCD als bpm (beats per minute) ausgegeben (siehe 26, (S. 54).
Etwa 33 Sekunden (1s Wartezeit + $4 * 8.1$s für die FFT) nach dem Messbeginn wird erstmals auch die gemittelte Sauerstoffsättigung über die LCD-Anzeige dargestellt (siehe Abbildung 27, (S. 54). Eine Aktualisierung der Sauerstoffsättigung erfolgt nach jedem Berechnungszyklus ähnlich zur Herzfrequenz in 8.1s Abständen.

4.4.2 Messergebnisse am Beispiel eines regulär aufgenommenen Datensatzes

Zur Darstellung der hinter der Berechnung der Vitalparameter stehenden Absorptionssignale wurde dafür beispielhaft eine Mausmessung durchgeführt und deren Signale an das MATLAB-GUI übertragen. Die vom GUI empfangenen digitalen Daten sind als zeitliche Signale in Abbildung 28 (S. 55) dargestellt und zeigen anschaulich die Charakteristik der Pulswelle. Zwischen dem für die Auswertung relevanten arteriellen Pulssignal ist der niederfrequente Atemschnapper der Maus zu sehen, der dem Nutzsignal überlagert ist. Für die Auswertung wird dieses Atemsignal und der in dieser Darstellung gut erkennbare Gleichanteil über einen digitalen IIR-HP mit einer Grenzfrequenz von $f_g = 2$Hz, als auch der hochfrequente Anteil durch einen IIR-TP mit einer Grenzfrequenz von $f_g = 10$Hz, entfernt. Dieses digital gefilterte Signal wird über die FFT-Berechnung in die Frequenzanteile zerlegt und zur Auswertung herangezogen. Zur Visualisierung eines dieser Amplitudenspektren ist dieses in den Abbildungen 29 (S. 55) und 30 (S. 56) im Detail dargestellt und wurde, wie auch die Daten von Abbildung 28 (S. 55), vom Mikroprozessor an das MATLAB-GUI übertragen. Im einseitigen Amplitudenspektrum der Abbildung 29 ist der gesamte Frequenzbereich abgebildet und zeigt anschaulich, dass sich das Pulssignal (Maximum) gut von den übrigen Frequenzanteilen abhebt. Eine Detailaufnahme des physiologisch relevanten Frequenzbereichs in Abbildung 30 (S.

56) zeigt den gut ausgeprägten Pulspeak bei ca. 5Hz. Gut zu erkennen sind in dieser Darstellung die gefilterten niederfrequenten Atemanteile und deren Oberwellen die für die Auswertung durchaus in einem störenden Bereich auftreten können, sofern sie durch ihre Amplitude das Pulssignal überragen (siehe 4.4.3, S. 73). Eine Analyse des Amplitudenspektrums ergibt des Weiteren, dass die Frequenzanteile relativ stark rauschen, was auf die angewandte Fensterung durch ein einfaches Rechteckfenster zurückgeführt werden kann. Für weitere Entwicklungen empfiehlt es sich dieses Rechteckfenster durch ein geeigneteres Fenster (Hann, Hanning,..) zu ersetzen, wozu in der vorliegenden Arbeit nicht genügend Zeit vorhanden war (siehe 4.5.1, S. 77).

4.4.3 Messergebnisse am Beispiel eines artefaktbehafteten Datensatzes

Die in 4.4.2 (S. 72) bereits kurz angeführte Problematik eines ausgeprägten Atemartefakts, das besonders bei Einsetzen der Schnappatmung auftritt, ist durch die vom Mikroprozessor übertragenen und im MATLAB-GUI ausgewerteten Signalverläufen in Abbildung 31 (S. 56) dargestellt. Zu erkennen sind die Atemschnapper durch die im 3 bis 4 Sekunden Abstand auftretenden Signalausreißer. Zwischen diesen Spitzen ist das Pulssignal zu erkennen, das im Vergleich zu den Atemschnappern eine weitaus geringere Amplitude aufweist. Trotz digitaler Filterung ergibt das Amplitudenspektrum ausgeprägte niederfrequente Anteile, deren Amplitude das bei ca. 4Hz befindliche Pulssignal bei weitem überragt und zu einer falschen Detektion und Auswertung führt (Abbildungen 32, S. 57 und 33, S. 58). In diesem Fall entspricht die angezeigte Herzfrequenz bzw. Sauerstoffsättigung dem Atemsignal und ist demzufolge nicht zu gebrauchen. Zwar wird in der Maximadetektion der Frequenzbereich auf den physiologisch sinnvollen Bereich eingeschränkt (siehe Tabelle 2, S. 14), jedoch kann durch das starke Atemartefakt eine korrekte Berechnung nicht garantiert werden. Erkennbar ist dies durch die Anzeige einer viel zu geringen Herzfrequenz von unter 200 bpm.

4.4.4 Überblick über die unterschiedlichen vom Mikroprozessor übertragenen Signaltypen

Die im Unterpunkt 3.4.1 (S. 58) angeführten Screenshots (Abbildungen 34 (Samples nach OP1, S. 59), 35 (Samples nach OP2, S. 59), 36 (gefilterte Samples, S. 60) und 37 (μP FFT, S. 60)) zeigen eine erfolgreiche Messung und Datenübertragung der Absorptionskurven für die beiden Wellenlängen im infraroten und sichtbaren roten Lichtbereich. Dabei ist in Abbildung 34 (S. 59), die die digitalisierten Samples nach der ersten OPV-

Stufe (Transimpedanzwandler) repräsentieren, gut erkennbar, dass dem eigentlichen Nutzsignal ein sehr starkes höherfrequentes Störsignal überlagert ist. Es handelt sich dabei um den $50Hz$ Netzbrumm und dessen Oberwellen, die durch den aktiven Hochpass am Transimpedanzwandler ($f_g = 15.4kHz$) ungehindert mitverstärkt werden. Die augenscheinlich invertierten Absorptionskurven werden durch die invertierende Verstärkungscharakteristik des Transimpedanzwandlers erklärt (vergleiche mit Abbildung 36, (S. 60)).
Wertet man die Samples nach der zweiten OPV-Stufe (einem invertierenden Verstärker mit einer Verstärkung von $V = 40$) aus (siehe Abbildung 35, S. 59), ist zu erkennen, dass sich das in Abbildung 34 sehr stark vorhandene Störsignal im Vergleich zum Nutzsignal stark verringert hat. Dies wird durch einen dem invertierenden Verstärker zugeschalteten aktiven Hochpass erster Ordnung mit einer Grenzfrequenz von $f_g = 7.96Hz$ erreicht. Des Weiteren werden die anfänglich invertierten Absorptionskurven durch den invertierenden Verstärker umgekehrt und liegen nun in der aus der Literatur bekannten Form vor. In diesem Signaltyp ist erstmals eindeutig auch mit freiem Auge die unterschiedlich starke Absorption der beiden Wellenlängen zu beobachten. Dieser Unterschied kann in weiterer Folge dazu verwendet werden die Sauerstoffsättigung zu berechnen (siehe dazu 4.4.5, S. 74)
Der dritte verfügbare Signaltyp ist in Abbildung 36 (S. 60) dargestellt. Darin abgebildet sind die im μP durch zwei IIR-Filter dritter Ordnung (einem IIR-HP mit einer Grenzfrequenz von $f_g = 0.25Hz$ und einem IIR-TP mit einer Grenzfrequenz von $f_g = 10Hz$) gefilterten Absorptionskurven. Der digitale Hochpass entfernt den sehr starken Gleichanteil und niederfrequente Erscheinungen wie die Atmung, während der Tiefpass hochfrequente Störsignale (v.a. Netzbrumm) blockiert. Dieses Signal wird in weiterer Folge vom μP verwendet um über eine FFT-Berechnung die Herzfrequenz und die Sauerstoffsättigung zu bestimmen.
Neben den drei Zeitsignalen ist es zudem möglich, sich das einseitige FFT Amplitudenspektrum basierend auf 2048 gefilterten Zeitsamples zu übertragen. In Abbildung 37 (S. 60) ist ein solches Spektrum abgebildet und zeigt anschaulich, dass sich in den Signalen lediglich das Nutzsignal (Pulswelle) und deren Oberwellen befinden.

4.4.5 Signalmanipulation am Beispiel von OP2-Samples in Zeit- und Frequenzbereich

Um beispielhaft vorzuführen wie die Berechnung von Herzfrequenz und Sauerstoffsättigung über die MATLAB GUI durchgeführt werden kann und welche Probleme dabei

entstehen können, ist dieser Prozess in den Abbildungen 38 (S. 61), 39 (S. 61) und 40 (S. 62) dargestellt.
Als ersten Schritt wird aus dem Datensatz der in Abbildung 35 (S. 59) dargestellten "`Samples nach OP2`“ mit dem Button `button_fft` die FFT berechnet und das einseitige Amplitudenspektrum im Plot dargestellt. Da es sich hierbei um nicht digital gefilterte Daten handelt, sind im Spektrum zwei wichtige Merkmale zu erkennen. Zum Einen besitzt das Amplitudenspektrum ausgeprägte niederfrequente Signalanteile mit einem starken Gleichanteil (Wert am Frequenzindex 0) und zum Anderen einen Peak bei 50Hz. Dieser Umstand führt bei der durchgeführten Berechnung der Herzfrequenz und der Sauerstoffsättigung zu einem grundlegend falschen Ergebnis. Nachdem dazu zuerst das Maximum des auf den gefilterten ir-samples basierenden Amplitudenspektrums gesucht wird, erhält man als Antwort nicht etwa den Amplitudenwert des schwächeren Pulssignals, sondern das niederfrequente Amplitudenmaximum (wie den zugehörigen Textboxen zu entnehmen ist). Um dies zu umgehen, muss zumindest der niederfrequente und unerwünschte Signalanteil entfernt werden. Als zusätzliche Maßnahme empfiehlt es sich auch die unerwünschten höherfrequenten Signalanteile (z.B. $> 10Hz$) per digitaler Filterung zu entfernen.
Um die Auswirkungen der digitalen Filterung darzulegen, wurden für den in Abbildung 39 (S. 61) dargestellten Screenshot über die Radiobuttons (`radio_hp` und `radio_tp`) eine IIR-TP- ($f_g = 10Hz$) und IIR-HP-Filterung ($f_g = 0.25Hz$) aktiviert und per Button `button_time` ein Update des Plots durchgeführt. Als grundlegende Datenbasis wurden die in Abbildung 35 (S. 59) dargestellten Datensamples verwendet.
Aus diesen nachträglich digital gefilterten Absorptionssignalen werden in einem weiteren Schritt durch Aktivierung des Buttons `button_fft` die FFT berechnet und deren Ergebnis in Abbildung 40 (S. 62) dargestellt. Zu erkennen ist, dass die im oberen Teil dieses Unterpunkts beschriebenen störenden Signalanteile durch die Filterung entfernt wurden und auch die Kalkulation der Herzfrequenz und der Sauerstoffsättigung vernünftige Werte liefert.

4.4.6 Artefaktbehaftete Signale

In diesem abschließenden Unterpunkt soll kurz auf die in den Abbildungen 41 (S. 62), 42 (S. 63) und 43 (S. 63) dargestellten Screenshots eingegangen und erläutert werden, wie diese Artefakte zu vermeiden sind. Aus Gründen der Vergleichbarkeit wurden diese Artefakte bei der Übertragung von „`Samples nach OP2`“ produziert - sie treten aber natürlich in ähnlicher Weise auch bei den anderen Signaltypen auf.

In Abbildung 41 (S. 62) ist dargestellt, wie sich die Absorptionssignale verhalten, wenn direkt nach der Sensorapplikation eine Messung und Signalübertragung gestartet wird. Da durch das Anbringen des Sensors am Finger zunächst die LED-Ströme geregelt werden um an der Photodiode eine ähnlich große Intensität zu erzeugen, dauert es einen Moment, bis sich das Signal des Pulssignals einstellt. Um dies zu vermeiden, empfiehlt es sich mit der Datenübertragung einige Sekunden zu warten.
Ein vergleichbares Artefakt ist in Abbildung 42 (S. 63) dargestellt. Hierbei wurde jedoch während dem Mess- und Übertragungsvorgang der Sensor vom Finger entfernt - das Signal reißt sprunghaft ab und pendelt um die Nulllinie. Um dieses Artefakt auszuschließen, soll mit der Sensorabnahme bis zum Ende der Datenübertragung gewartet werden.
Die Auswirkungen eines weiteren und weitaus üblicheren Artefakts sind in Abbildung 43 (S. 63) abgebildet. Dabei wurde der Finger bzw. die Hand, an der der Sensor appliziert wurde, sehr stark bewegt. Die dabei erzeugten Signalschwankungen übersteigen die Dynamik des Regelvorgangs und bilden sich als undefinierbare Signalsprünge aus. Beendet man die Bewegung und hält den Sensor ruhig, stellt sich innerhalb weniger Sekunden wieder ein schönes Pulssignal ein. Um den Einfluss der Bewegung auf ein Minimum zu reduzieren, empfiehlt es sich die Hand mit dem Sensor auf den Tisch zu legen und diesen nicht zu bewegen.

4.5 Kalibration und Validierung

Um die durch die Messhardware erzeugten Berechnungsergebnisse der Herzfrequenz und der Sauerstoffsättigung auf ihre Korrektheit zu überprüfen, muss eine Validierung mit Ergebnissen einer geeichten Messapparatur erfolgen. Da im Handel erhältliche Pulsoximeter, wie sie auch zur Überwachung am Menschen eingesetzt werden, mit einem Genauigkeitsbereich von $\pm 3\%$ arbeiten [33], scheiden diese als Referenzmessgerät aus und es muss auf komplexe Laborhardware wie einen Blutgasanalysator zurückgegriffen werden.
In der vorliegenden Arbeit wird keine eigene Kalibrationskurve erstellt, sondern jeine von einem Pulsoximeter-Projekt [17], das auch den Sensortyp DS100A verwendet, übernommen. Die dadurch erzeugten Berechnungsergebnisse sind mit äußerster Vorsicht zu interpretieren, da nur eine Plausibilitätsprüfung durchgeführt wurde. Eine Begutachtung der in mehreren Messsitzungen erstellten Signalkurven und der zugehörigen Berechnungsergebnisse lassen die Vermutung zu, dass die Hardware vernünftige Werte

liefert, die der realen Sauerstoffsättigung sehr nahe kommen müsste. Bis jedoch keine vollständige Validierung erfolgt, kann nicht darauf vertraut werden, dass es sich bei den berechneten und angezeigten Werten um die tatsächliche Sauerstoffsättigung handelt. Aufgrund der, für eine vollständige Kalibration mit anschließender Validierung notwendigen und sehr aufwendigen Verfahren, musste im Rahmen dieser Arbeit darauf verzichtet werden. Zur Vollständigkeit ist an diesem Punkt lediglich eine theoretische Beschreibung einer solchen Kalibration angeführt [5]. Damit kann der Sauerstoffgehalt der Blutprobe genau genug bestimmt und mittels in-vitro-Messungen an einem Dummy-Objekt, das den Einsatzort des Pulsoximeters nachbildet (z.B. Mauspfote), durchgeführt werden. Eine aufwendige, physiologisch möglichst korrekte Pumpvorrichtung simuliert dabei das pulsierende arterielle Blut und bildet die Grundlage zur Bestimmung des Ratioverhältnisses der Absorptionsmaxima. Über eine ausreichende Anzahl von Messpunkten kann eine Fitting-Kurve zur Kalibration erstellt und zur endgültigen Auswertung herangezogen werden. Über die Erstellung einer solchen Fittingkurve ist es zudem möglich, die Messergebnisse einer Validierung zu unterziehen und die berechneten Ergebnisse mit den bekannten Sauerstoffsättigungen der verwendeten Proben zu vergleichen.

4.5.1 Verbesserungsvorschläge

Während der Entwicklung, aber vor allem während der Auswertung der Ergebnisse, hat sich herausgestellt, dass mehrere Hard- und Softwareelemente optimiert werden könnten, um dadurch die Qualität der Messergebnisse zu verbessern. Um diese Überlegungen für zukünftige Projekte oder einem Update dieser Hardware nutzbar zu machen, sind diese in den folgenden Unterpunkten angeführt und näher beschrieben.

Kalibration und Validierung Um verlässliche Berechnungsergebnisse zu erhalten und sich auf die Korrektheit dieser zu verlassen, ist es unumgänglich, eine für die Hardware optimierte Kalibrationskurve zu erstellen. Dies kann wie in 4.5 (S. 76) beschrieben oder durch vergleichbare Messungen erreicht werden. Die unter [5] angeführte Literatur bietet dazu in Kapitel 3 und Kapitel 10 ausreichende Informationen.

Schirmen der Messhardware Über die großen Widerstände des Transimpedanzwandlers und der nachfolgenden Verstärkerstufen werden hochfrequente Störfelder (wie der Netzbrumm und dessen Oberwellen) in die Signalverstärkung eingestrahlt (siehe dazu Abbildung 34, S. 59 und 3.4.1, S. 58). Um dies zu vermeiden, empfiehlt es sich, die

Hardware in ein geschirmtes Gehäuse zu verbauen um dadurch die Einflüsse externer Störfelder zu verringern.

Änderung des Zeitprofils um den Einfluss des Umgebungslichts zu bestimmen Die vorliegende Hardware und das Design des Zeitprofils (siehe 2.3.5, S. 21) basiert auf der Annahme, dass neben den durch die Sende-LEDs erzeugten Lichtintensitäten keine weiteren Lichteinstreuungen vorliegen. Bei der Anwendung der Hardware am Menschen hat sich jedoch gezeigt, dass trotz optischer Abschirmung diese Annahme nur in gut abgedunkelten Räumen gilt. Vor allem bei direkter Sonneneinstrahlung wurden leicht unterschiedliche Signale gemessen. Um die Einflüsse dieses für die Auswertung störenden Lichts zu eliminieren, empfiehlt es sich, das Zeitprofil um eine Phase, in der keine der beiden LEDs aktiv ist, zu ergänzen und das dabei erzeugte Signal bei der Berechnung zu berücksichtigen.

höhere Verstärkung Um mit der Messhardware auch humane Pulssignale für die Berechnung der Sauerstoffsättigung aufzeichnen zu können, wurde die maximale Verstärkung so ausgelegt, dass diese Wechselsignale in den Digitalisierungsbereich von 0V bis 3.3V passen. Die humanen Pulswellensignale sind aber um etwa drei Zehnerpotenzen größer (Amplitude) als die Pulswellensignale der Maus, woraus folgt, dass für ein reines Maus-Pulsoximeter die Verstärkung durchaus höher angesetzt werden kann. Dies würde die Spannungsauflösung der Pulswelle erheblich verbessern und möglicherweise zu einer genaueren Auswertung führen.

bessere Atemunterdrückung Zur Unterdrückung der niederfrequenten Atemsignale wird in der Software des Mikrokontrollers eine digitale HP-Filterung mit einem IIR-Filter dritter Ordnung durchgeührt. Dabei kann es vorkommen, dass das Atemsignal nicht stark genug unterdrückt wird - eine falsche Pulswellendetektion und falsche Berechnungsergebnisse sind die Folge. Um dies zu verhindern, könnte man in weiteren Projekten diese Unterdrückung durch eine Erhöhung der Filterordnung oder den Einsatz von adaptiven Filtern realisiern.

FFT-Fenster Die Berechnung der FFT im Mikrokontroller basiert auf der Anwendung eines Rechteckfensters, was zu teilweise sehr starken Gibbsartefakten führt. Diese im Amplitudenspektrum der FFT auftretende Verschleifung der Maxima kann durch geeignetere Fenstertypen wie z.B. Hann- oder Kaiser-Fenster sehr stark minimiert werden.

4.6 Ausblick

Die vorliegende Messhardware kann unter der Voraussetzung einer ausgewachsenen adulten Maus und korrekter Befestigung des Sensors dazu verwendet werden, um die Herzrate und die Sauerstoffsättigung zu messen, darzustellen und die digital vorliegenden Signaldaten zur weiteren Bearbeitung oder Visualisierung an ein MATLAB-GUI zu übertragen. Trotz einiger Verbesserungsvorschläge für weiterführende Projekte (siehe 4.5.1, S. 77) sind die von der Messhardware erstellten Absorptionskurven von ausreichender Qualität um, unter der Voraussetzung, dass die Fittingkurve die das Ratio korrekt in die Sauerstoffsättigung umrechnet, über die in einer ersten Form vorliegende Hard- und Software stabile Werte für die Sauerstoffsättigung zu berechnen. Dies ermöglicht es dieses Hardware- und Softwaredesign zu verwenden um darauf aufbauend weitere Einsatzgebiete wie etwa die MR-Tauglichkeit zu erschließen, oder durch den Einsatz von verschieden großen Sensorhalterungen auch kleinere Mäuse messen zu können.

Literatur

[1] Striebel H.: *Die operative Intensivmedizin: Sicherheit in der klinischen Praxis.* Schattauer Gmbh (2007)

[2] Burchadi H. et. al.: *Die Intensivmedizin.* Springer Berlin Heidelberg (2007)

[3] Severinghaus J, Honda Y.: History of Blood Gas Analysis. VII. Pulse Oximetry. *Journal of Clinical Monitoring* 3: 135–138 (1987)

[4] Kramme R: *Medizintechnik: Verfahren - Systeme - Informationsverarbeitung.* Springer Berlin Heidelberg (2006)

[5] Webster J: *Design of Pulse Oximeters.* Taylor & Francis (1997)

[6] Yan YS, Poon CCY et al.: Reduction of motion artifact in pulse oximetry by smoothed pseudo Wigner-Ville distribution *J Neuroengineering Rehabil* 2: 3 (2005)

[7] Severinghaus JW, Kelleher JF: Recent Developments in Pulse Oximetry *Anesthesiology* 76: 1018-1038 (1992)

[8] Nürnberger J, Mitchel A et al.: Pulswellenreflexion - Bestimmung, Einflussgrößen, Analyse und Anwendungsoptionen *Dtsch Med Wochenschr* 128: 97 - 102 (2003)

[9] Baura G: *Medical Device Technologies: A Systems Based Overview Using Engineering Standards.* Academic Press (2011)

[10] Jubran A: Pulse oximetry *Crit Care* 3: R11-R17 (1999)

[11] Suckow MA, Dannemann P et al.: *The Laboratory Mouse* Crc Pr Inc (2001)

[12] Silbernagel S, Despopoulos A: *Taschenatlas Physiologie* Thieme (2007)

[13] Bernstein D: Exercise assessment of transgenic models of human cardiovascular disease *Physiol. Genomics* Vol. 13 No. 3: 217-226 (2003)

[14] Fox JG, Barthold S et al.: *The Mouse in Biomedical Research, Volume 3: Normative Biology, Husbandry, and Models* Academic Press (2006)

[15] Schmidt-Nielsen K: *Animal Physiology: Adaptation and Environment* Cambridge University Press (1997)

[16] Constantinides C, Mean R et al.: Effects of Isoflurane Anesthesia on the Cardiovascular Function of the C57BL/6 Mouse *ILAR e-Journal* Vol. 52: e21-e31 (2011)

[17] Chan V, Underwood S: *A Single-Chip Pulsoximeter Design Using the MSP430*, unter: http://www.ti.com/litv/pdf/slaa274b (abgerufen am 17.10.2011)

[18] Nisarga B: *Revised Pulsoximeter Design Using the MSP430*, unter: www.ti.com/lit/an/slaa458/slaa458.pdf (abgerufen am 10.12.2011)

[19] Matviyenko S: *Pulse Oximeter - Standard*, unter: http://www.eetasia.com/STATIC/PDF/201002/EEOL_2010FEB22_STECH_AN_01.pdf (abgerufen am 17.12.2011)

[20] Multinfor: *Reusable SpO Sensor*, unter: http://www.multinfor.pt/---PRODUTOS/---CABOS_NIBP_SPO2/SPO2_ComCabo.pdf (abgerufen am 23.01.2012)

[21] Nellcor: *N-600x Pulse Oximeter Alarm Management System*, unter: http://www.nellcor.com/_catalog/pdf/sns/prodman/10044103a00.pdf (abgerufen am 23.01.2012)

[22] Kent Scientific Corporation: *PhysioSuite*, unter: https://www.kentscientific.com/products/productView.asp?ProductId=6437&Mouse_Rat=Physiological+Monitoring&Products=PhysioSuite (abgerufen am 12.02.2012)

[23] NXP: *mbed NXP LPC1768 Handbook*, unter: http://mbed.org/handbook/mbed-NXP-LPC1768 (abgerufen am 03.11.2011)

[24] NXP: *Datenblatt des mbed NXP LPC1768*, unter: http://www.nxp.com/documents/user_manual/UM10360.pdf (abgerufen am 03.11.2011)

[25] Maxim Integrated Products: *Datenblatt des MAX4617/MAX4618/MAX4619*, unter: http://datasheets.maximintegrated.com/en/ds/MAX4617-MAX4619.pdf (abgerufen am 12.11.2011)

[26] Microchip: *Datenblatt des MCP6021/1R/2/3/4*, unter: http://ww1.microchip.com/downloads/en/DeviceDoc/21685d.pdf (abgerufen am 17.05.2012)

[27] NXP: *mbed Handbook*, unter: http://mbed.org/handbook/Homepage (abgerufen am 03.11.2011)

[28] NXP, Ford S: *Library TextLCD*, unter: http://mbed.org/users/simon/code/TextLCD/ (abgerufen am 03.11.2011)

[29] Malepati H: *Digital Media Processing: DSP Algorithms Using C* Newnes (2010)

[30] Press WH, Teukolsky SA et. al.: *Numerical Recipes 3rd Edition: The Art of Scientific Computing* Cambridge University Press (2007)

[31] Wikipedia: *Filter mit unendlicher Impulsantwort*, unter: http://de.wikipedia.org/wiki/Filter_mit_unendlicher_Impulsantwort (abgerufen am 03.01.2012)

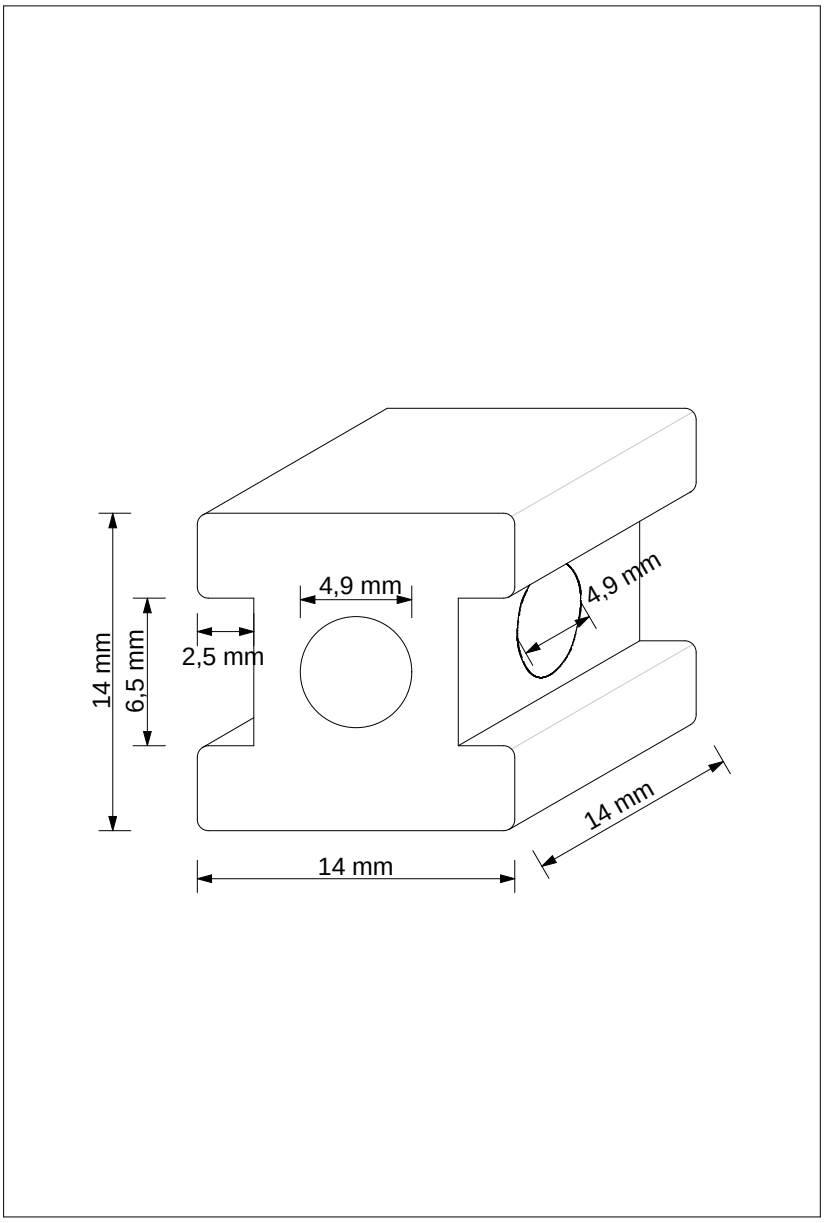
[32] zhaw (Zürcher Hochschule für Angewandte Wissenschaften): *Kapitel 3: DFT und FFT*, unter: https://home.zhaw.ch/~rumc/dsv1/unterlagen/dsv1kap3dftfft.pdf (abgerufen am 03.05.2012)

[33] NIHON KOHDEN CORPORATION: *SPO2 Monitoring: Pulse Oximeter Accuracy Study*, unter: http://www.nihonkohden.de/uploads/media/SpO2-Report_01.pdf (abgerufen am 03.06.2012)

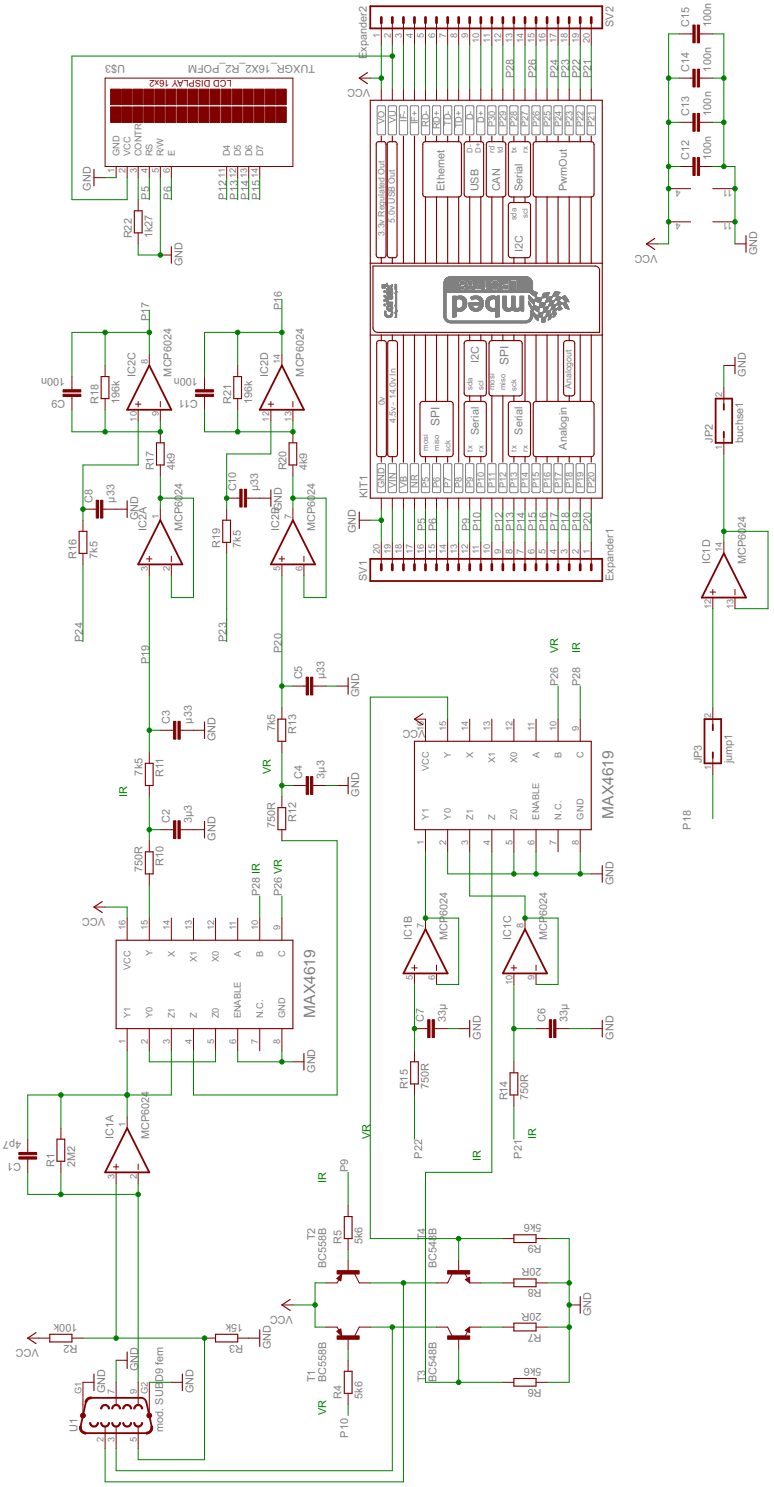
Anhang

Anhang A: Hardware

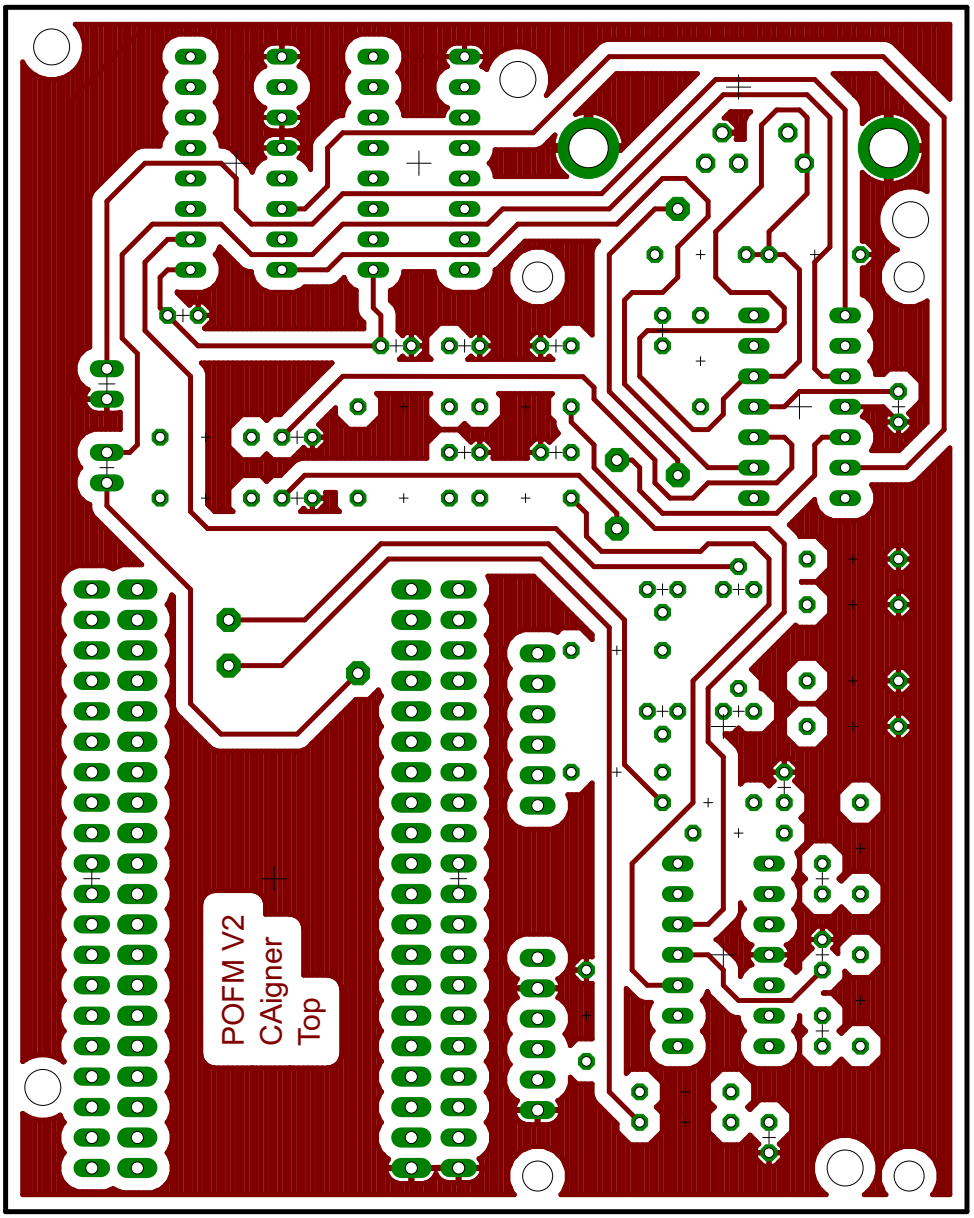
Konstruktionszeichnung für die Sensorhalterung



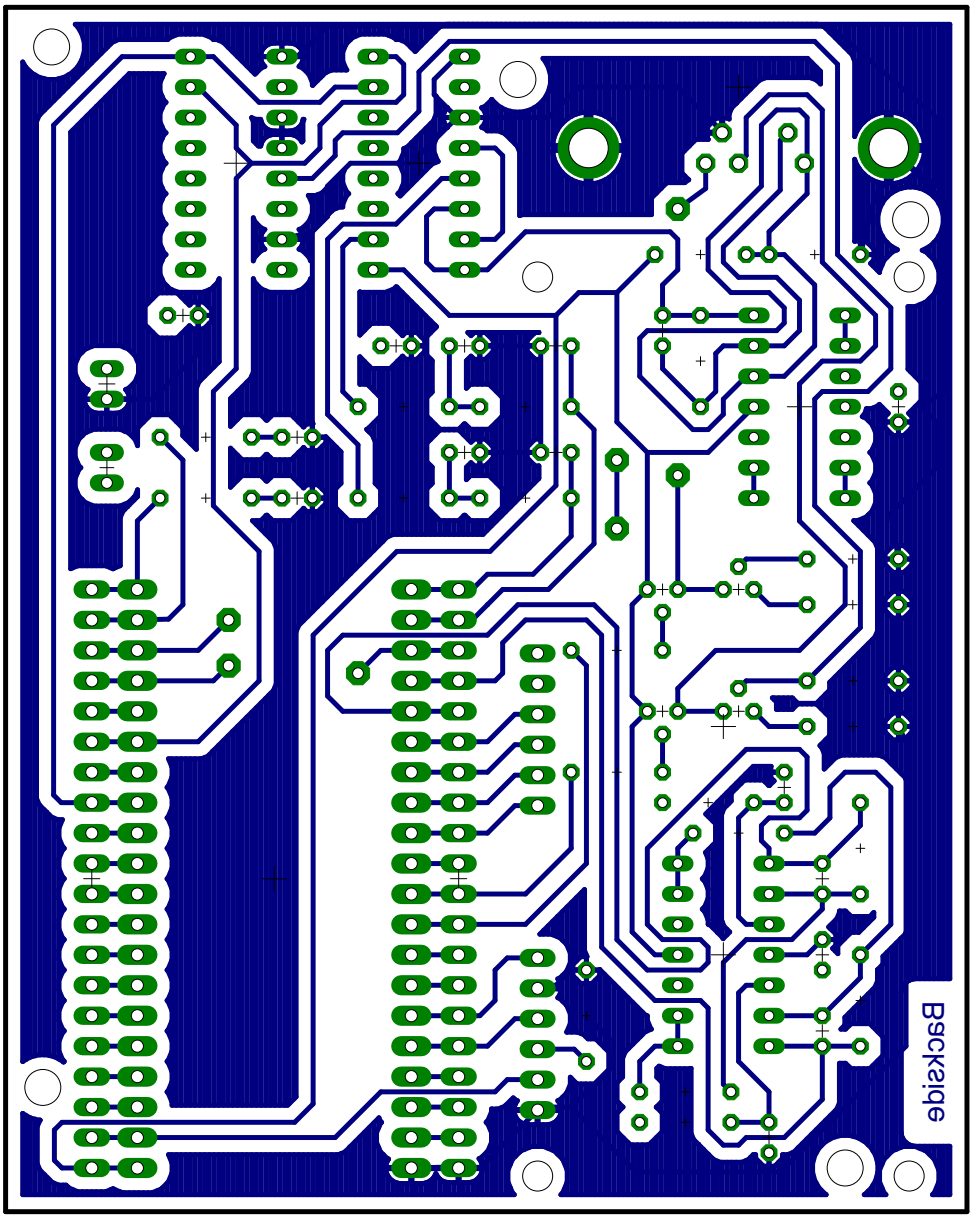
Stromlaufplan



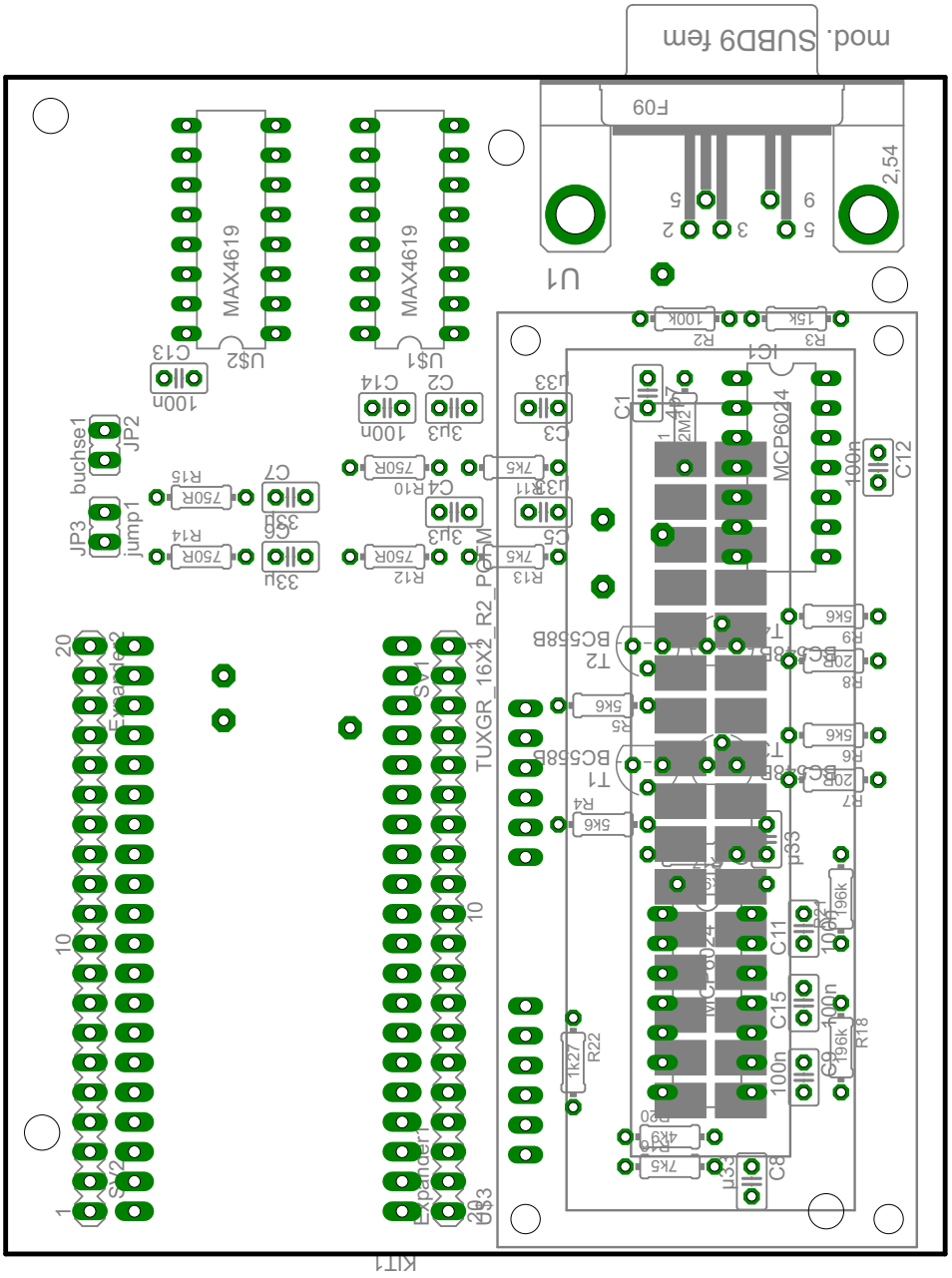
Printlayout (Oberseite)



Printlayout (Rückseite)



Bestückungsplan (Oberseite)



Stückliste

Tabelle 9: Stückliste

Schaltplan	Beschreibung	Farnell-BNr.	Anzahl	Einzelpreis	ges. Preis
C1	Keramik 3p9	1694255	1	0,03 €	0,03 €
C3, C5, C8, C10	Tantal μ33	9708502	4	0,45 €	1,78 €
C2, C4	Tantal 3μ3	1869992	2	0,09 €	0,18 €
C6, C7	Tantal 33μ	1869977	2	0,14 €	0,29 €
C9, C11, C12, C13, C14, C15	100nF	2112751	6	0,04 €	0,24 €
DIS1	16x2 LCD	1847932	1	12,37 €	12,37 €
IC1, IC2	MCP6024	1627199	2	2,53 €	5,06 €
IC3, IC4	Maxim 4619	1593372	2	2,97 €	5,94 €
KIT1	MBED	1761179	1	50,97 €	50,97 €
R7, R8	Widerstand 20R	9340181	2	0,06 €	0,11 €
R10, R12, R14, R15	Widerstand 750R	9340882	4	0,06 €	0,22 €
R22	Widerstand 1k27	1083263	1	0,25 €	0,25 €
R17, R20	Widerstand 4k9 0.1%	9502297	2	1,22 €	2,44 €
R4, R5, R6, R9	Widerstand 5k6	9340734	4	0,06 €	0,22 €
R11, R13, R16, R19	Widerstand 7k5	9340890	4	0,03 €	0,10 €
R3	Widerstand 15k	9340009	1	0,06 €	0,06 €
R2	Widerstand 100k	9339795	1	0,06 €	0,06 €
R18, R21	Widerstand 196k 0.1%	9500685	2	1,22 €	2,44 €
R1	Widerstand 2M2	1265095	1	0,15 €	0,15 €
SV1, SV2	Buchsenleiste 20x1	9728910	2	3,79 €	7,58 €
T1, T2	PNP Transistor BC 558B	1467883	2	0,128 €	0,26 €
T3, T4	NPN Transistor BC 548B	1467874	2	0,145 €	0,29 €
X1	DSUB9 Female	1848372	1	0,99 €	0,99 €
			50	77,78 €	92,02 €

Anhang B: Software

Programmlisting der μP-Software

```
1  ////////////////////////////////////////////////////////////
2  //  Project POFM (Pulsoxy for mice)
3  //  Christoph Aigner
4  //  01.10.2012
5  //  Version 1
6  //  Softwarecode for the Microcontroller to measure the pulse
7  //  waves and calculate the heartrate and oxygen saturation
8  //  A full description is available in the maser's thesis
9  //  "Entwicklung eines Pulsoxymeters zur Bestimmung der
10 //   Sauerstoffsättigung und der Herzrate von sedierten
11 //   Labormäusen" under 2.4 (Page 29)
12 //
13 //  The following code uses adapted functions that are used
14 //  originally in the TI-Application "slaa274a".
15 //  to identify them, I marked it with a seperate comment -> [TI]
16 //  source: http://www.ti.com/general/docs/
17 //          litabsmultiplefilelist.tsp?literatureNumber=slaa274a
18 //  and the functions to calculate the FFT and the amplitude are
19 //  implemented after the library "FFT" done by the mbed-user
20 //  "Ale C.
21 //  source: http://mbed.org/users/Suky/code/FFT/file/
22 //          e3af07c00c13/FFT.cpp
23 ////////////////////////////////////////////////////////////
24
25
26 //------------------------------------------------------------
27 // Libraries
28 //
29 #include "mbed.h"
30 #include "TextLCD.h"
31
32 //------------------------------------------------------------
33 // Constants
34 //
35
36 #define fft_length 2048
37
38
39 //Definition of the range for the control of the LED-current
```

```
40 #define FIRST_STAGE_TARGET_HIGH          30000      //38000
41 #define FIRST_STAGE_TARGET_LOW           28000     //33000
42 #define FIRST_STAGE_TARGET_HIGH_FINE     33000      //60000
43 #define FIRST_STAGE_TARGET_LOW_FINE      25000      //30000
44
45 //Definition of the stepsize for the control of the offset
46 //voltage
47 #define FIRST_STAGE_STEP                 0.001
48 #define FIRST_STAGE_FINE_STEP            0.0005    //0.000025
49
50
51
52
53 // IIR Koeffizienten
54 //lowpass, fs=250, fg=10Hz, order: 3, coeffs calculated by
55 //         MATLAB (butter) -> filterdesign/iir_fg10_order3
56   static double a[4] ={ 1.0,
57                        -2.2755089982457601,
58                         1.7908933685573403,
59                        -0.48078217176168281};
60   static double b[4] ={ 0.0043252748187371749,
61                         0.012975824456211524,
62                         0.012975824456211524,
63                         0.0043252748187371749};
64
65 //USE FOR MICE
66 //lowpass, fs=250, fg=2Hz, order: 3, coeffs calculated by
67 //         MATLAB (butter) -> filterdesign/iir_fg10_order3
68   static const double a_hp[4] ={ 1.0,
69                                  -2.8994795946118641,
70                                   2.8039479773830003,
71                                  -0.90434753139240909};
72   static const double b_hp[4] ={ 0.95097188792340903,
73                                  -2.8529156637702271,
74                                   2.8529156637702271,
75                                  -0.95097188792340903};
76
77 //USE FOR HUMAN
78 //lowpass, fs=250, fg=0.25Hz, order: 3, coeffs calculated by
79 //         MATLAB (butter) -> filterdesign/iir_fg10_order3
80 //  static double a_hp[4] ={ 1.0,
81 //                           -2.9874336500557224,
```

```
82 //                              2.9749461326654423,
83 //                             -0.9875122361107358};
84 //  static double b_hp[4] ={0.99373650235398769,
85 //                            -2.9812095070619629,
86 //                             2.9812095070619629,
87 //                            -0.99373650235398769};
88
89
90 //-----------------------------------------------------------------
91 // Variable declaration
92 //
93
94     int sample_time=0;
95
96   //Running variable
97     int counter_data=0;
98
99  //communication variable
100    char pc_command='0';
101
102  // Variables to calculate the heartfrequ and SPO2
103    double ln_vr=0.0;
104    double ln_ir=0.0;
105    int Ratio=0;
106    int SaO2=0;
107
108
109  //Variables for IR (infrared) sample
110    unsigned short ir_sample_op1=0;
111    unsigned short ir_sample_op2=0;
112    unsigned short ir_dc=0;
113
114    double ir_filtered_sample=0;
115    float ir_LED_level=0.2;
116    double ir_sample_float=0.0;
117    unsigned ir_filtered_ac_sample=0.0;
118
119
120    float ir_dc_offset = 0;
121
122  //Variables for VR (visible red) sample
123    unsigned short vr_sample_op1=0;
```

```
124     unsigned short vr_sample_op2=0;
125     unsigned short vr_dc=0;
126
127     double vr_filtered_sample=0;
128     float vr_LED_level=0.2;
129     double vr_sample_float=0.0;
130     double vr_filtered_ac_sample=0.0;
131
132     float vr_dc_offset = 0;
133
134
135  // Variables for fft
136     float ir_filtered_array[fft_length];
137     float vr_filtered_array[fft_length];
138
139     float ir_fft_out[fft_length/2];
140     float vr_fft_out[fft_length/2];
141
142     int start_fft=0;
143     int save_samples=1;
144     int heartrate=0;
145     int upper_border=0;
146
147  //Variables for SaO2
148     float max=0.0;
149     int index_fft_max=0;
150
151     float ratio=0.0;
152     float ratio_array[4]={0,0,0,0};
153     float ratio_average=0;
154     int counter_ring=0;
155     int start_plot=0;
156
157 //-------------------------------------------------------------------
158 // MBED Port definition
159 //
160   //LCD-Panel
161     TextLCD lcd(p5, p6, p12, p13, p14, p15); // rs, e, d4-d7
162
163   //Digitaler Ausgang
164     DigitalOut ir_control(p9);
165     DigitalOut vl_control(p10);
```

```
166     DigitalOut ir_demultiplex_control(p28);
167     DigitalOut vl_demultiplex_control(p26);
168 
169   //Digital Eingang
170     DigitalIn dac_switch(p7);
171 
172   //ADC
173     AnalogIn tiefpass1(p19);
174     AnalogIn tiefpass2(p20);
175 
176     AnalogIn second_op1(p17);
177     AnalogIn second_op2(p16);
178 
179 
180   //DAC
181     AnalogOut dac(p18);
182 
183   //PWM Ausgang
184     PwmOut ir_ledcontrol(p21);
185     PwmOut vr_ledcontrol(p22);
186 
187     PwmOut ir_dcoffset(p24);
188     PwmOut vr_dcoffset(p23);
189 
190 //------------------------------------------------------------------------
191 // MBED-Variablen
192 //
193   //Emulierte Serielle Schnittstelle via USB
194     Serial pc(USBTX, USBRX); // tx, rx
195 
196   //LED
197     DigitalOut led(LED1);
198 
199   //Timer
200     Timer t;
201 
202     Timer check_time;
203 
204   //Interrupt for the given Ticker
205     Ticker sample_ticker;
206 
207 //------------------------------------------------------------------------
```

```
208 // functions
209 //
210 // void take_samples();
211 // description: Funktion der Ticker-Interrupt-Routine.
212 //              Nimmt in periodischen Abstaenden Samples,
213 //              berechnet die IIR-Filterung, die Peaks und
214 //              sendet diese Daten gegebenenfalls per USB
215 //              an den PC
216 void take_samples();
217
218 // float control_LED_level(unsigned short sample,
219 //                         float LED_level);
220 // description: Regelung des LED-Stromes in dem der Messwert
221 //              nach dem I/U-Wandler mit den gewuenschten
222 //              Grenzen verglichen und gegebenen Falles erhoeht
223 //              bzw. verringert wird
224 //
225 // input: unsigned short sample ... Messwert nach dem
226 //                                  I/U-Wandler
227 //        float LED_level ... Gleitkommazahl der "alten"
228 //                            LED_Stromstaerke
229 // output: float control_LED_level ... Gleitkommazahl fuer die
230 //                                     "neue" LED-Stromstaerke
231 float control_LED_level(unsigned short sample, float LED_level);
232
233 // double ir_iir_filter(double sample);
234 // description: Gleitkomma IIR-Filterung nach den von MATLAB
235 //              berechneten Koeffizienten
236 //
237 // input: double sample ... Messwert nach dem I/U-Wandler
238 //                          als Gleitkommazahl
239 // output: double ir_iir_filter ... Gleitkommazahl fuer das
240 //                                  gefilterte Sample
241 double ir_iir_filter(double sample);
242
243 // double vr_iir_filter(double sample);
244 // description: Gleitkomma IIR-Filterung nach den von MATLAB
245 //              berechneten Koeffizienten
246 //
247 // input: double sample ... Messwert nach dem I/U-Wandler
248 //                          als Gleitkommazahl
249 // output: double vr_iir_filter ... Gleitkommazahl fuer das
```

```
250 //                                  gefilterte Sample
251 double vr_iir_filter(double sample);
252
253
254 // double ir_iir_filter(double sample);
255 // description: Gleitkomma IIR-Filterung nach den von MATLAB
256 //              berechneten Koeffizienten
257 //
258 // input: double sample ... Messwert nach dem I/U-Wandler
259 //                          als Gleitkommazahl
260 // output: double ir_iir_filter ... Gleitkommazahl fuer das
261 //                                  gefilterte Sample
262 double ir_hp_iir_filter(double sample);
263
264 // double vr_iir_filter(double sample);
265 // description: Gleitkomma IIR-Filterung nach den von MATLAB
266 //              berechneten Koeffizienten
267 //
268 // input: double sample ... Messwert nach dem I/U-Wandler
269 //                          als Gleitkommazahl
270 // output: double ir_iir_filter ... Gleitkommazahl fuer das
271 //                                  gefilterte Sample
272 double vr_hp_iir_filter(double sample);
273
274 // unsigned short dc_est(unsigned short p, unsigned short x);
275 // description: Regelt den Gleichsignalanteil fuer den
276 //              invertierenden Verstaerker
277 // input: unsigned short sample ... Messwert nach dem
278 //                                  I/U-Wandler als
279 //                                  Gleitkommazahl
280 //        float dc_offset ... Gleitkommazahl des "alten"
281 //                            Offsets
282 // output: float control_dc_offset ... Gleitkommazahl fuer den
283 //                                     Offset
284 float control_dc_offset(unsigned short sample, float dc_offset);
285
286
287 // Extracted from Numerical Recipes in C and the mbed user
288 // "Ale C"
289 //Replaces data[1..2*nn] by its discrete Fourier transform, if
290 //isign is input as 1; or replaces data[1..2*nn] by nn times
291 //its inverse discrete Fourier transform, if isign is input as
```

```
292 //-1. data is a complex array of length nn or, equivalently, a
293 //real array of length 2*nn. nn MUST be an integer power of 2
294 //(this is not checked for!).
295 void vFFT(float data[], unsigned int nn);
296
297 // Extracted from Numerical Recipes in C and the mbed user
298 // "Ale C"
299 //Calculates the Fourier transform of a set of n real-valued
300 //data points. Replaces this data (which is stored in array
301 //data[1..n]) by the positive frequency half of its complex
302 //Fourier transform. The real-valued first and last components
303 //of the complex transform are returned as elements data[1] and
304 //data[2], respectively. n must be a power of 2. This routine
305 //also calculates the inverse transform of a complex data array
306 //if it is the transform of real data. (Result in this case
307 //must be multiplied by 2/n.)
308 void vRealFFT(float data[], unsigned int n);
309
310
311 // Extracted from Numerical Recipes in C and the mbed user
312 // "Ale C"
313 // Calculates the single sided amplitudes spectrum
314 void vCalPowerf(float Input[],float Power[], unsigned int n);
315
316 int main()
317 {
318   //Sampledauer in ms
319     sample_time=4000;
320
321   //Initialisierung der verwendeten Elemente
322   //PWM-Ausgaenge
323     ir_ledcontrol.period_us(50);
324     vr_ledcontrol.period_us(50);
325
326     ir_dcoffset.period_us(100);
327     vr_dcoffset.period_us(100);
328
329     //default-Werte fuer die Initialisierung
330     ir_LED_level=0.25;
331     vr_LED_level=0.25;
332
333     ir_dc_offset=0.15;
```

```
334     vr_dc_offset=0.15;
335
336     //Zuweisen der default-Werte
337     ir_ledcontrol = ir_LED_level;
338     vr_ledcontrol = vr_LED_level;
339
340     ir_dcoffset = ir_dc_offset;
341     vr_dcoffset = vr_dc_offset;
342
343
344   //USB(Serial Emulator)
345     pc.baud(460800);
346
347
348   //LCD Init
349     lcd.cls();
350     lcd.locate(0,0);
351     lcd.printf("Pulsoximeter for");
352     lcd.locate(0,1);
353     lcd.printf("Mice V0.7 TUGRAZ");
354
355
356   //wait 1 s to allow suppyly/signals to settle
357     wait(1);
358
359   //LCD Measure
360     lcd.cls();
361     lcd.locate(0,0);
362     lcd.printf("POFM frequ ..bpm");
363     lcd.locate(0,1);
364     lcd.printf("      SPO2 ..%% ");
365
366
367   // Turn on the IR-LED to allow the first opamp to settle
368     ir_control = 1;
369     vl_control = 0;
370
371
372   //-------------------------------------------------------------
373   // Measurement and Calculations
374   // one loop = 4ms (controlled by ticker and timer)
375   // --> 250 samples per second per wavelength
```

```
376   //
377   // the address of the function to be attached (take_samples)
378   //and the interval (4 Milliseconds)
379   sample_ticker.attach_us(&take_samples, sample_time);
380
381
382   while(1) // infinite loop
383   {
384     //if the PC send a character catch the character
385     if(pc.readable())
386     {
387       pc_command = pc.getc();
388     }
389
390    //if 2048 samples are done the variable start_fft gets set
391    //to 1 by the ticker-function sample_ticker
392    if(start_fft == 1)
393     {
394      //set start_fft to 0; because the if while(1) will ask
395      //for its value
396      start_fft = 0;
397      //stop saving the samples because the array will be used
398      //for the fft-calculation
399      save_samples = 0;
400
401      // clear the single sided amplitude spectra
402      for ( int counter_fftdata = 0;
403            counter_fftdata < fft_length/2;
404            counter_fftdata++ )
405      {
406        ir_fft_out[counter_fftdata]=0.0;
407        vr_fft_out[counter_fftdata]=0.0;
408      }
409
410      /* //Can be used to send the sample-array
411      if(pc_command == 'a')
412      {
413        for ( int counter_fftdata = 0;
414              counter_fftdata < fft_length;
415              counter_fftdata++ )
416        {
417          pc.printf("%f\n", ir_filtered_array[counter_fftdata]);
```

```
418           pc.printf("%f\n", vr_filtered_array[counter_fftdata]);
419         }
420       }
421       */
422
423       //Calculate the Real-FFT of the underlying sample-data and
424       //the corresponding single sided amplitude spectra
425       //takes ~0.62 seconds
426       vRealFFT(vr_filtered_array, fft_length);
427       vCalPowerf(vr_filtered_array, vr_fft_out, fft_length/2);
428
429       vRealFFT(ir_filtered_array, fft_length);
430       vCalPowerf(ir_filtered_array, ir_fft_out, fft_length/2);
431
432       /* //Can be used to send the Real-FFT
433       if(pc_command == 'b')
434       {
435         for ( int counter_fftdata = 0;
436               counter_fftdata < fft_length;
437               counter_fftdata++ )
438         {
439           pc.printf("%f\n", ir_filtered_array[counter_fftdata]);
440           pc.printf("%f\n", vr_filtered_array[counter_fftdata]);
441         }
442       }
443       */
444
445       //If the PC asked for the single sided amplitude spectra
446       //send them
447       if(pc_command == 'c')
448       {
449         for ( int counter_fftdata = 0;
450               counter_fftdata < fft_length/2;
451               counter_fftdata++ )
452         {
453           pc.printf("%lf\n", ir_fft_out[counter_fftdata]);
454           pc.printf("%lf\n", vr_fft_out[counter_fftdata]);
455         }
456       }
457
458      //Find the maximum value in the single sided amplitude
459      //spectrum (only ir-data; higher values)
```

```
460     upper_border = 10*1024*sample_time/1000000;
461     for ( int counter_fftmax = 0;
462           counter_fftmax < upper_border;
463           counter_fftmax++ )
464     {
465       if (max < ir_fft_out[counter_fftmax])
466       {
467         max = ir_fft_out[counter_fftmax];
468         index_fft_max = counter_fftmax;
469       }
470     }
471
472     //Compute the ratio of the two maximum-values
473     ratio = (vr_fft_out[index_fft_max]) /
474             (ir_fft_out[index_fft_max]);
475
476     //Store the ratio in the ring-buffer ratio_array that gets a
477     //new element per round
478     ratio_array[counter_ring] = ratio;
479
480     //Compute the moving average of the last four ratios
481     //(because of the ring-buffer)
482     for(int counter_average=0;
483         counter_average < 4;
484         counter_average++)
485       ratio_average = ratio_average +
486                       ratio_array[counter_average]/4;
487
488     //Set back the ring-counter counter_ring and start the
489     //display output with the variable start_plot or increase
490     //the ring_counter
491     if(counter_ring == 3)
492     {
493       counter_ring = 0;
494       start_plot=1;
495     }
496     else
497       counter_ring++;
498
499     //compute the heartrate with the frequency index that
500     //was found before
501     heartrate= int(index_fft_max*60*125/1024);
```

```
502
503     //Ask the Fittingcurve for the corresponding SaO2-Value
504     SaO2 = int(110.28-17.51*ratio_average);
505
506     //limit the percentage
507     if(SaO2 >= 100)
508       SaO2=100;
509
510     //Ceck if there is a signal (LED_levels will differ from 0.2)
511     //and print the results to the display
512     if(ir_LED_level != vr_LED_level)
513     {
514       //Print the heartrate
515       lcd.locate(5,0);
516       lcd.printf("frequ%3dbpm", heartrate);
517
518       //Check if the moving average is ready
519       if(start_plot == 1)
520       {
521         //Print the Oxygen-Saturation
522         lcd.locate(5,1);
523         lcd.printf("SPO2 %3d%%  ", SaO2);
524       }
525       else
526       {
527         lcd.locate(11,1);
528         lcd.printf("XXX");
529       }
530     }
531     //there is no signal
532     else
533     {
534         lcd.locate(5,0);
535         lcd.printf("      no    ");
536         lcd.locate(5,1);
537         lcd.printf("    signal ");
538     }
539
540
541     // clear variables
542     max=0;
543     index_fft_max=0;
```

```
544     ratio_average = 0;
545
546     // clear the single sided fft spectra; the samples will
547     // be stored there
548     for ( int counter_fftdata = 0;
549           counter_fftdata < fft_length;
550           counter_fftdata++ )
551     {
552       ir_filtered_array[counter_fftdata]=0.0;
553       vr_filtered_array[counter_fftdata]=0.0;
554     }
555
556     // Continue saving the samples by the flag save_samples
557     save_samples = 1;
558     }
559   }
560 }
561
562 void take_samples()
563 {
564   ////////////////////////////////////////////////////////////
565   //Get one sample with the IR-LED
566   ////////////////////////////////////////////////////////////
567   t.start();
568
569   // Visible AUS
570   vl_control = 1;
571   vl_demultiplex_control = 0;
572
573   // IR EIN
574   ir_control = 0;
575   ir_demultiplex_control = 1;
576
577   //wait until all signals are stable (0.5 ms)
578   wait_us(sample_time/4);
579
580   // read signal after first op-stage
581   ir_sample_op1 = tiefpass1.read_u16();
582
583   //read signal after second op-stage
584   ir_sample_op2 = second_op1.read_u16();
585
```

```
586   ir_sample_float = ir_sample_op2;
587
588   //Filter away frequencies > 10Hz
589   ir_filtered_sample =
590       ir_iir_filter((ir_sample_float/0xFFFF)*3.3);
591
592   //Filter away frequencies < 2Hz
593   ir_filtered_sample = ir_hp_iir_filter(ir_filtered_sample);
594
595   //put actual sample in array
596   if(save_samples == 1)
597     ir_filtered_array[counter_data]=float(ir_filtered_sample);
598
599   // Bring the IR signal into range
600   ir_LED_level = control_LED_level(ir_sample_op1, ir_LED_level);
601
602   //Calculate DC
603   ir_dc_offset = control_dc_offset(ir_sample_op2, ir_dc_offset);
604
605   //Update the PWM-Outputs
606   ir_ledcontrol = ir_LED_level;
607   ir_dcoffset = ir_dc_offset;
608
609   // wait until 1ms is over (from the start)
610   while(t.read_us() <= sample_time/2);
611
612   //reset the timer
613   t.reset();
614
615
616   ////////////////////////////////////////////////////////////
617   //Get one sample with the visible Red-LED
618   ////////////////////////////////////////////////////////////
619   t.start();
620
621   //IR Aus
622   ir_control = 1;
623   ir_demultiplex_control = 0;
624
625   //Visible Ein
626   vl_control = 0;
627   vl_demultiplex_control = 1;
```

```
628
629   //wait until all signals are stable (1ms)
630   wait_us(sample_time/4);
631
632   // read signal after first op-stage and lowpass
633   vr_sample_op1=tiefpass2.read_u16();
634
635   vr_sample_op2=second_op2.read_u16();
636
637   vr_sample_float = vr_sample_op2;
638
639   //Filter away frequencies > 10Hz
640   vr_filtered_sample =
641        vr_iir_filter((vr_sample_float/0xFFFF)*3.3);
642
643   //Filter away frequencies < 2Hz
644   vr_filtered_sample = vr_hp_iir_filter(vr_filtered_sample);
645
646   //put actual sample in array
647   if(save_samples == 1)
648     vr_filtered_array[counter_data]=float(vr_filtered_sample);
649
650   // Bring the IR signal into range through the 2nd opamp
651   vr_LED_level = control_LED_level(vr_sample_op1, vr_LED_level);
652
653   //Calculate DC
654   vr_dc_offset = control_dc_offset(vr_sample_op2, vr_dc_offset);
655
656   vr_ledcontrol = vr_LED_level;
657   vr_dcoffset = vr_dc_offset;
658
659
660 /*
661   //Write Signal to DAC
662   if(dac_switch)
663   {
664     dac.write((ir_filtered_sample/(10*3.3)));
665   }
666   else
667   {
668     dac.write((vr_filtered_sample/(10*3.3)));
669   }
```

```
*/

  //send Data per USB if the following char was sent

  if(pc_command == '1')
  {
    pc.printf("%lf\n", ir_filtered_sample);
    pc.printf("%lf\n", vr_filtered_sample);
  }
  else if(pc_command == '2')
  {
    pc.printf("%u\n", ir_sample_op2);
    pc.printf("%u\n", vr_sample_op2);
  }
  else if(pc_command == '3')
  {
    pc.printf("%u\n", ir_sample_op1);
    pc.printf("%u\n", vr_sample_op1);
  }
  else if(pc_command == '4')
  {
    pc.printf("%lf\n", ir_LED_level);
    pc.printf("%lf\n", vr_LED_level);
  }
  else if(pc_command == '5')
  {
    pc.printf("%lf\n", ir_dc_offset);
    pc.printf("%lf\n", vr_dc_offset);
  }

  t.reset();

  if(save_samples == 1)
    counter_data++;


  if(counter_data == fft_length)
  {
    counter_data=0;
    start_fft=1;
  }
}
```

```
712
713 // double vr_iir_filter(double sample);
714 // description: Gleitkomma IIR-Filterung nach den von MATLAB
715 //              berechneten Koeffizienten
716 //
717 // input: double sample ... Messwert nach dem I/U-Wandler
718 //                          als Gleitkommazahl
719 // output: double ir_iir_filter ... Gleitkommazahl fuer das
720 //                                  gefilterte Sample
721 double vr_iir_filter(double sample)
722 {
723   //Define the variables as static so they remain also
724   //after the call
725     static double x[4]={ 0.0, 0.0, 0.0, 0.0};
726     static double y[4]={ 0.0, 0.0, 0.0, 0.0};
727
728   //Shift the values by 1
729     x[0]=x[1];
730     x[1]=x[2];
731     x[2]=x[3];
732
733     y[0]=y[1];
734     y[1]=y[2];
735     y[2]=y[3];
736
737   //Assign new measured value
738     x[3] = sample;
739
740   //calculate the the filtered output-value
741     y[3]= b[0]*x[3] + b[1]*x[2] + b[2]*x[1] +
742           b[3]*x[0] - a[1]*y[2] - a[2]*y[1] -
743           a[3]*y[0];
744     return y[3];
745 }
746
747 // double ir_iir_filter(double sample);
748 // description: Gleitkomma IIR-Filterung nach den von MATLAB
749 //              berechneten Koeffizienten
750 //
751 // input: double sample ... Messwert nach dem I/U-Wandler
752 //                          als Gleitkommazahl
753 // output: double ir_iir_filter ... Gleitkommazahl fuer das
```

```
754 //                                  gefilterte Sample
755 double ir_iir_filter(double sample)
756 {
757   //Define the variables as static so they remain also
758   //after the call
759     static double x[4]={ 0.0, 0.0, 0.0, 0.0};
760     static double y[4]={ 0.0, 0.0, 0.0, 0.0};
761
762   //Shift the values by 1
763     x[0]=x[1];
764     x[1]=x[2];
765     x[2]=x[3];
766
767     y[0]=y[1];
768     y[1]=y[2];
769     y[2]=y[3];
770
771   //Assign new measured value
772     x[3] = sample;
773
774   //calculate the the filtered output-value
775     y[3]= b[0]*x[3] + b[1]*x[2] + b[2]*x[1] +
776           b[3]*x[0] - a[1]*y[2] - a[2]*y[1] -
777           a[3]*y[0];
778     return y[3];
779 }
780
781 // double vr_hp_iir_filter(double sample);
782 // description: Gleitkomma IIR-Filterung nach den von MATLAB
783 //              berechneten Koeffizienten
784 //
785 // input: double sample ... Messwert nach dem I/U-Wandler
786 //                          als Gleitkommazahl
787 // output: double ir_iir_filter ... Gleitkommazahl fuer das
788 //                                  gefilterte Sample
789 double vr_hp_iir_filter(double sample)
790 {
791   //Define the variables as static so they remain also
792   //after the call
793     static double x[4]={ 0.0, 0.0, 0.0, 0.0};
794     static double y[4]={ 0.0, 0.0, 0.0, 0.0};
795
```

```
796   //Shift the values by 1
797     x[0]=x[1];
798     x[1]=x[2];
799     x[2]=x[3];
800
801     y[0]=y[1];
802     y[1]=y[2];
803     y[2]=y[3];
804
805   //Assign new measured value
806     x[3] = sample;
807
808   //calculate the the filtered output-value
809     y[3]= b_hp[0]*x[3] + b_hp[1]*x[2] + b_hp[2]*x[1] +
810           b_hp[3]*x[0] - a_hp[1]*y[2] - a_hp[2]*y[1] -
811           a_hp[3]*y[0];
812     return y[3];
813 }
814
815 // double ir_hp_iir_filter(double sample);
816 // description: Gleitkomma IIR-Filterung nach den von MATLAB
817 //              berechneten Koeffizienten
818 //
819 // input: double sample ... Messwert nach dem I/U-Wandler
820 //                          als Gleitkommazahl
821 // output: double ir_iir_filter ... Gleitkommazahl fuer das
822 //                                  gefilterte Sample
823 double ir_hp_iir_filter(double sample)
824 {
825   //Define the variables as static so they remain also
826   //after the call
827     static double x[4]={ 0.0, 0.0, 0.0, 0.0};
828     static double y[4]={ 0.0, 0.0, 0.0, 0.0};
829
830   //Shift the values by 1
831     x[0]=x[1];
832     x[1]=x[2];
833     x[2]=x[3];
834
835     y[0]=y[1];
836     y[1]=y[2];
837     y[2]=y[3];
```

```
838
839   //Assign new measured value
840     x[3] = sample;
841
842   //calculate the the filtered output-value
843     y[3]= b_hp[0]*x[3] + b_hp[1]*x[2] + b_hp[2]*x[1] +
844           b_hp[3]*x[0] - a_hp[1]*y[2] - a_hp[2]*y[1] -
845           a_hp[3]*y[0];
846     return y[3];
847 }
848
849
850
851 // modified from [TI]
852 // float control_LED_level(unsigned short sample,
853 //                         float LED_level);
854 // description: Regelung des LED-Stromes in dem der Messwert
855 //              nach dem I/U-Wandler mit den gewuenschten
856 //              Grenzen verglichen und gegebenen Falles erhoeht
857 //              bzw. verringert wird
858 //
859 // input: unsigned short sample ... Messwert nach dem
860 //                                  I/U-Wandler
861 //        float LED_level ... Gleitkommazahl der "alten"
862 //                            LED_Stromstaerke
863 // output: float control_LED_level ... Gleitkommazahl fuer die
864 //                                     "neue" LED-Stromstaerke
865 float control_LED_level(unsigned short sample, float LED_level)
866 {
867     if (sample > FIRST_STAGE_TARGET_HIGH ||
868         sample < FIRST_STAGE_TARGET_LOW)
869     {
870     //We are out of the target range: Starting kicking the LED
871     //intensity in the right direction to bring us back into
872     //range. We use fine steps when we are close to the target
873     //range, and coarser steps when we are far away.
874
875       if (sample > FIRST_STAGE_TARGET_HIGH)
876       {
877         if (sample >= FIRST_STAGE_TARGET_HIGH_FINE)
878           LED_level -= FIRST_STAGE_STEP;
879         else
```

```
880             LED_level -= FIRST_STAGE_FINE_STEP;
881
882           // Clamp to the range of the DAC
883           if (LED_level < 0.2) //circa 0.7V
884             LED_level = 0.2;
885         }
886         else
887         {
888           if (sample < FIRST_STAGE_TARGET_LOW_FINE)
889             LED_level += FIRST_STAGE_STEP;
890           else
891             LED_level += FIRST_STAGE_FINE_STEP;
892
893           // Clamp to the range of the DAC
894           if (LED_level > 0.6) // circa 2V
895             LED_level = 0.6;
896         }
897       }
898
899       return LED_level;
900 }
901
902 // unsigned short dc_est(unsigned short p, unsigned short x);
903 // description: Regelt den Gleichsignalanteil fuer den
904 //              invertierenden Verstaerker
905 // input: unsigned short sample ... Messwert nach dem
906 //                                  I/U-Wandler als
907 //                                  Gleitkommazahl
908 //        float dc_offset ... Gleitkommazahl des "alten"
909 //                            Offsets
910 // output: float control_dc_offset ... Gleitkommazahl fuer den
911 //                                     Offset
912 float control_dc_offset(unsigned short sample, float dc_offset)
913 {
914 // Bring the IR signal into range through the second opamp
915       if (sample >= 0xE000)
916       {
917         if (dc_offset > 0.025)
918           dc_offset-=FIRST_STAGE_FINE_STEP;
919       }
920       else if (sample < 0x2000)
921       {
```

```
922         if (dc_offset < 0.99)
923           dc_offset+=FIRST_STAGE_FINE_STEP;
924       }
925 
926       return dc_offset;
927 }
928 
929 
930 // Extracted from Numerical Recipes in C and the mbed user
931 // "Ale C"
932 //Replaces data[1..2*nn] by its discrete Fourier transform, if
933 //isign is input as 1; or replaces data[1..2*nn] by nn times
934 //its inverse discrete Fourier transform, if isign is input as
935 //-1. data is a complex array of length nn or, equivalently, a
936 //real array of length 2*nn. nn MUST be an integer power of 2
937 //(this is not checked for!).
938 void vFFT(float data[], unsigned int nn)
939 {
940   unsigned int n,mmax,m,j,istep,i;
941   double wtemp,wr,wpr,wpi,wi,theta;
942   float tempr,tempi;
943 
944   n=nn << 1;
945   j=1;
946   for (i=1;i<n;i+=2)
947   {
948     if(j>i)
949     {
950       tempr=data[j];
951       data[j]=data[i];
952       data[i]=tempr;
953 
954       tempr=data[j+1];
955       data[j+1]=data[i+1];
956       data[i+1]=tempr;
957     }
958     m=n >> 1;
959     while (m >= 2 &&j>m)
960     {
961       j-=m;
962       m >>= 1;
963     }
```

```
964       j+=m;
965     }
966     mmax=2;
967     while (n > mmax)
968     {
969       istep=mmax << 1;
970       theta=(6.28318530717959/mmax);
971       wtemp=sin(0.5*theta);
972       wpr = -2.0*wtemp*wtemp;
973       wpi=sin(theta);
974       wr=1.0;
975       wi=0.0;
976       for (m=1;m<mmax;m+=2)
977       {
978         for (i=m;i<=n;i+=istep)
979         {
980           j=i+mmax;
981           tempr=wr*data[j]-wi*data[j+1];
982           tempi=wr*data[j+1]+wi*data[j];
983           data[j]=data[i]-tempr;
984           data[j+1]=data[i+1]-tempi;
985           data[i] += tempr;
986           data[i+1] += tempi;
987         }
988         wr=(wtemp=wr)*wpr-wi*wpi+wr;
989         wi=wi*wpr+wtemp*wpi+wi;
990       }
991       mmax=istep;
992     }
993 }
994
995 // Extracted from Numerical Recipes in C and the mbed user
996 // "Ale C"
997 //Calculates the Fourier transform of a set of n real-valued
998 //data points. Replaces this data (which is stored in array
999 //data[1..n]) by the positive frequency half of its complex
1000 //Fourier transform. The real-valued first and last components
1001 //of the complex transform are returned as elements data[1] and
1002 //data[2], respectively. n must be a power of 2. This routine
1003 //also calculates the inverse transform of a complex data array
1004 //if it is the transform of real data. (Result in this case
1005 //must be multiplied by 2/n.)
```

```
1006 void vRealFFT(float data[], unsigned int n)
1007 {
1008   unsigned int i,i1,i2,i3,i4,np3;
1009   float c1=0.5,c2,h1r,h1i,h2r,h2i;
1010   double wr,wi,wpr,wpi,wtemp,theta;
1011   theta=3.141592653589793/(double) (n>>1);
1012
1013   c2 = -0.5;
1014   vFFT(data,n>>1);
1015   wtemp=sin(0.5*theta);
1016   wpr = -2.0*wtemp*wtemp;
1017   wpi=sin(theta);
1018   wr=1.0+wpr;
1019   wi=wpi;
1020   np3=n+3;
1021   for (i=2;i<=(n>>2);i++)
1022   {
1023     i4=1+(i3=np3-(i2=1+(i1=i+i-1)));
1024     h1r=c1*(data[i1]+data[i3]);
1025     h1i=c1*(data[i2]-data[i4]);
1026     h2r = -c2*(data[i2]+data[i4]);
1027     h2i=c2*(data[i1]-data[i3]);
1028     data[i1]=h1r+wr*h2r-wi*h2i;
1029     data[i2]=h1i+wr*h2i+wi*h2r;
1030     data[i3]=h1r-wr*h2r+wi*h2i;
1031     data[i4] = -h1i+wr*h2i+wi*h2r;
1032     wr=(wtemp=wr)*wpr-wi*wpi+wr;
1033     wi=wi*wpr+wtemp*wpi+wi;
1034   }
1035   data[1] = (h1r=data[1])+data[2];
1036   data[2] = h1r-data[2];
1037 }
1038
1039
1040 // Extracted from Numerical Recipes in C and the mbed user
1041 // "Ale C"
1042 // Calculates the single sided amplitudes spectrum
1043 void vCalPowerf(float Input[],float Power[], unsigned int n)
1044 {
1045   unsigned short k,j;
1046
1047   for(k=0,j=0;k<n;k++,j+=2)
```

```
1048     {
1049         Power[k]=sqrt(Input[j]*Input[j]+Input[j+1]*Input[j+1]);
1050     }
1051 }
```

Programmlisting des MATLAB-GUI

```
function varargout = pofm_gui(varargin)
% POFM_GUI MATLAB code for pofm_gui.fig
%      POFM_GUI, by itself, creates a new POFM_GUI or raises the existing
%      singleton*.
%
%      H = POFM_GUI returns the handle to a new POFM_GUI or the handle to
%      the existing singleton*.
%
%      POFM_GUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in POFM_GUI.M with the given input arguments.
%
%      POFM_GUI('Property','Value',...) creates a new POFM_GUI or raises the
%      existing singleton*. Starting from the left, property value pairs are
%      applied to the GUI before pofm_gui_OpeningFcn gets called. An
%      unrecognized property name or invalid value makes property
%      application stop.
%      All inputs are passed to pofm_gui_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help pofm_gui

% Last Modified by GUIDE v2.5 23-Sep-2012 15:50:10

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @pofm_gui_OpeningFcn, ...
                   'gui_OutputFcn',  @pofm_gui_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
```

```
42 else
43     gui_mainfcn(gui_State, varargin{:});
44 end
45 % End initialization code - DO NOT EDIT
46
47
48 % --- Executes just before pofm_gui is made visible.
49 function pofm_gui_OpeningFcn(hObject, eventdata, handles, varargin)
50 % This function has no output args, see OutputFcn.
51 % hObject    handle to figure
52 % eventdata  reserved - to be defined in a future version of MATLAB
53 % handles    structure with handles and user data (see GUIDATA)
54 % varargin   command line arguments to pofm_gui (see VARARGIN)
55
56 % Choose default command line output for pofm_gui
57 handles.output = hObject;
58
59 % Update handles structure
60 guidata(hObject, handles);
61
62 %%% Disable the objects that can only be used after a data-transmission
63 set(handles.button_fft, 'Enable', 'off');
64 set(handles.button_timespace, 'Enable', 'off');
65
66
67
68 % UIWAIT makes pofm_gui wait for user response (see UIRESUME)
69 % uiwait(handles.figure1);
70
71 %%%% Creates the main window
72 % --- Outputs from this function are returned to the command line.
73 function varargout = pofm_gui_OutputFcn(hObject, eventdata, handles)
74 % varargout  cell array for returning output args (see VARARGOUT);
75 % hObject    handle to figure
76 % eventdata  reserved - to be defined in a future version of MATLAB
77 % handles    structure with handles and user data (see GUIDATA)
78
79 % Get default command line output from handles structure
80 varargout{1} = handles.output;
81
82 %%% Check for the available COM-ports
83 cell_com_ports = getAvailableComPort();
```

```
84
85 %%% Close the program if no COM-port is available or write them into the
86 %%% popup-menu
87 if(isempty(cell2mat(cell_com_ports)))
88     warndlg('The POFM-device is not connected. Please connect it,restart
       MATLAB and launch the program');
89     close all;
90 else
91     set(handles.popup_com_port, 'String', cell_com_ports);
92 end
93
94 %%% initialize the gui-objects
95 set(handles.popup_choose_signal, 'String', {'Samples nach OP1', ...
96     'Samples nach OP2','gefilterte Samples','mP FFT'});
97 set(handles.edit_number_of_samples, 'string', '1024');
98 set(handles.edit_scan_duration, 'string', '4.096');
99 set(handles.edit_hr, 'string', '---');
100 set(handles.edit_sao2, 'string', '---');
101 set(handles.edit_filename, 'string', 'exp1_');
102 set(handles.edit_fg_hp, 'string', '0.25');
103 set(handles.edit_fg_tp, 'string', '10');
104
105
106 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
107 %%%% FUNCTIONS FOR THE GUI ELEMTS: BUTTONS
108 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
109
110 % --- Executes on button press in button_start.
111 function button_start_Callback(hObject, eventdata, handles)
112 % hObject    handle to button_start (see GCBO)
113 % eventdata  reserved - to be defined in a future version of MATLAB
114 % handles    structure with handles and user data (see GUIDATA)
115
116 %%%% Open a file to store the samples
117 filename = get(handles.edit_filename, 'String');
118 fid = fopen(filename, 'w');
119
120 %%%% Get the choosen COM-Port
121 % which number is choosen
122 choose_port = get(handles.popup_com_port, 'Value');
123 string_choose_port = get(handles.popup_com_port, 'String');
124 choosen_port = string_choose_port(choose_port);
```

```
125
126 %%%% Open the Serial-Connection
127 serial_handle = serial(choosen_port,'BaudRate',460800,'DataBits',8,...
128                        'Parity', 'none', 'StopBits', 1);
129 fopen(serial_handle);
130
131 %%%% Get the chosen fg_hp
132 string_fg_hp = get(handles.edit_fg_hp, 'String'); % the whole string
133 fg_hp = str2num(string_fg_hp);
134
135 %%%% Get the chosen fg_tp
136 string_fg_tp = get(handles.edit_fg_tp, 'String'); % the whole string
137 fg_tp = str2num(string_fg_tp);
138
139 %%%% Get the Number of requested samples
140 str_number_samples = get(handles.edit_number_of_samples, 'string');
141 number_samples = str2double(str_number_samples);
142
143 %%%% Initialize the arrays for the transmitted samples
144 ir_data(number_samples)=0;
145 vr_data(number_samples)=0;
146
147 %%%% Calculate the sample frequency
148 fa=1/(4*1e-3);
149
150 %%%% Get the Transmission-Code for choosen data
151 choose_signal = get(handles.popup_choose_signal, 'Value');
152
153 %%%% Get the corresponding character for the transmission
154 switch choose_signal
155     case 1
156         data_type='3'; %muP sends the samples after OP1
157     case 2
158         data_type='2'; %muP sends the samples after OP2
159     case 3
160         data_type='1'; %muP sends the filtered samples
161     case 4
162         data_type='c'; %muP sends the in the muP calculated fft (absolute)
163 end
164
165 %%%% start the transmission by sending the transmission-character
166 try
```

```
167     fwrite(serial_handle,data_type, 'uchar');
168 end
169
170
171 %%% receive the samples over the COM-port, write them to the log-file and
172 %%% save them in the corresponding array
173 for counter=1:number_samples
174     sample_ir= fscanf(serial_handle, '%s');
175     fprintf(fid,'%s\n',sample_ir);
176     ir_data(counter) = str2double(sample_ir);
177
178     sample_vr= fscanf(serial_handle, '%s');
179     fprintf(fid,'%s\n',sample_vr);
180     vr_data(counter) = str2double(sample_vr);
181 end
182
183 %%% stop the transmission by sending a '0'
184 try
185     fwrite(serial_handle,'0', 'uchar');
186 end
187
188 %%% if the muP-FFT was sent, different settings are necessary
189 if(data_type == 'c')
190     %frequency-variable in Herz
191     frequ = fa/2*linspace(0,1,number_samples);
192
193     %plot the data
194     plot(frequ,vr_data);
195     hold all
196     plot(frequ,ir_data);
197     hold off
198     grid on
199
200     %plot the descriptions
201     legend('visible','infrared');
202     title('[muP]-calculated FFT of the filtered samples (N=2048)');
203     xlabel('frequency in Hz');
204     ylabel('Amplitude in binary');
205
206     %Deactivate the buttons to perform or update the domains
207     set(handles.button_fft, 'Enable', 'off');
208     set(handles.button_timespace, 'Enable', 'off');
```

```
209
210 else
211     %if unfiltered samples got transmitted they have to be filterd with
212     %the choosen settings
213     if(data_type == '2' || data_type == '3')
214
215         %if the checkbox is on, perform a IIR-HP-Filter (3rd order)
216         %with the user's cutoff frequency
217         if (get(handles.radio_hp,'Value') == get(handles.radio_hp,'Max'))
218             str_hp_fg = get(handles.edit_fg_hp, 'string');
219             hp_fg = str2num(str_hp_fg)
220
221             [b_hp a_hp] = butter(3, hp_fg/fa, 'high');
222             ir_data=filtfilt(b_hp, a_hp, ir_data);
223             vr_data=filtfilt(b_hp, a_hp, vr_data);
224         end
225
226         %if the checkbox is on, perform a IIR-TP-Filter (3rd order) with
227         %the user's cutoff frequency
228         if (get(handles.radio_tp,'Value') == get(handles.radio_tp,'Max'))
229             str_tp_fg = get(handles.edit_fg_tp, 'string');
230             tp_fg = str2num(str_tp_fg)
231
232             [b_tp a_tp] = butter(3, tp_fg/fa, 'low');
233             ir_data=filtfilt(b_tp, a_tp, ir_data);
234             vr_data=filtfilt(b_tp, a_tp, vr_data);
235         end
236     end
237
238     %time-variable in seconds
239     time=1/fa:1/fa:length(ir_data)/fa;
240
241     %plot the data
242     plot(time,vr_data);
243     hold all
244     plot(time,ir_data);
245     hold off
246     grid on;
247
248     %plot the descriptions
249     legend('visible','infrared');
250     if(data_type == '1')
```

```
251         title('[muP]-filtered Data');
252     elseif (data_type == '2')
253         title('[muP]-OPAmp Stage 2 Data');
254     elseif (data_type == '3')
255         title('[muP]-OPAmp Stage 1 Data');
256     end
257     xlabel('time in s');
258     ylabel('Amplitude in binary');
259 
260     %Activate the buttons to perform or update the domains
261     set(handles.button_timespace, 'Enable', 'on');
262     set(handles.button_fft, 'Enable', 'on');
263 end
264 
265 %%%% Close the Serial-Handle and the File-Handle
266 fclose(serial_handle);
267 delete(serial_handle);
268 clear serial_handle;
269 fclose(fid);
270 
271 
272 % --- Executes on button press in button_quit.
273 function button_quit_Callback(hObject, eventdata, handles)
274 % hObject    handle to button_quit (see GCBO)
275 % eventdata  reserved - to be defined in a future version of MATLAB
276 % handles    structure with handles and user data (see GUIDATA)
277 close(gcbf);
278 
279 
280 
281 % --- Executes on button press in button_fft.
282 function button_fft_Callback(hObject, eventdata, handles)
283 % hObject    handle to button_fft (see GCBO)
284 % eventdata  reserved - to be defined in a future version of MATLAB
285 % handles    structure with handles and user data (see GUIDATA)
286 
287 %%%% read the filename out of the textbox
288 filename = get(handles.edit_filename, 'String');
289 data = csvread(filename);
290 
291 fa=1/0.004;
292 
```

```
293 %%%% read the ir- and vr-samples out of the data-array (alternating)
294 counter_ir = 1;
295 counter_vr = 1;
296 for counter_elem =1:size(data,1)
297      if mod(counter_elem,2)==1
298          ir_data(counter_ir)=data(counter_elem);
299          counter_ir=counter_ir+1;
300      elseif mod(counter_elem,2)==0
301          vr_data(counter_vr)=data(counter_elem);
302          counter_vr=counter_vr+1;
303      end
304 end
305
306 %%%% Get the Transmission-Code for choosen data
307 choose_signal = get(handles.popup_choose_signal, 'Value');
308
309 %%%% check for unfiltered samples
310 if(choose_signal == 1 || choose_signal == 2)
311   %if the checkbox is on, perform a IIR-HP-Filter (3rd order) with the
312   %user's cutoff frequency
313      if (get(handles.radio_hp,'Value') == get(handles.radio_hp,'Max'))
314          str_hp_fg = get(handles.edit_fg_hp, 'string');
315          hp_fg = str2num(str_hp_fg)
316
317          [b_hp a_hp] = butter(3, hp_fg/fa, 'high');
318          ir_data=filtfilt(b_hp, a_hp, ir_data);
319          vr_data=filtfilt(b_hp, a_hp, vr_data);
320      end
321   %if the checkbox is on, perform a IIR-TP-Filter (3rd order) with the
322   %user's cutoff frequency
323      if (get(handles.radio_tp,'Value') == get(handles.radio_tp,'Max'))
324          str_tp_fg = get(handles.edit_fg_tp, 'string');
325          tp_fg = str2num(str_tp_fg)
326
327          [b_tp a_tp] = butter(3, tp_fg/fa, 'low');
328          ir_data=filtfilt(b_tp, a_tp, ir_data);
329          vr_data=filtfilt(b_tp, a_tp, vr_data);
330      end
331 end
332
333 %%%% Calculate the single sided amplitude spectra
334 vr_NFFT = 2^nextpow2(length(vr_data)); % Next power of 2 from length of y
```

```
335 vr_Y = fft(vr_data,vr_NFFT)/length(vr_data);
336 vr_Y_plot=abs(vr_Y(1:vr_NFFT/2)/(vr_NFFT/2));
337
338
339 ir_NFFT = 2^nextpow2(length(ir_data)); % Next power of 2 from length of y
340 ir_Y = fft(ir_data,ir_NFFT)/length(ir_data);
341 ir_Y_plot=abs(ir_Y(1:ir_NFFT/2)/(ir_NFFT/2));
342
343 %%%% frequency in Hz
344 frequ = fa/2*linspace(0,1,vr_NFFT/2);
345
346 % Plot single-sided amplitude spectrum.
347 plot(frequ,vr_Y_plot,frequ,ir_Y_plot)
348 title('Single-Sided Amplitude-spectra')
349 xlabel('Frequency (Hz)')
350 ylabel('|Y(f)|')
351 legend('visible','infrared');
352 grid on
353
354 %%%% find the maximum and the corresponding array-index
355 [max_ir,index_max_ir] = max(ir_Y_plot);
356
357 %%%% Calculate the Heartrate
358 heartrate=index_max_ir*60*125/(ir_NFFT/2);
359
360 %%%% Calculate the Ratio of the maxima
361 fft_max_ir=ir_Y_plot(index_max_ir)
362 fft_max_vr=vr_Y_plot(index_max_ir)
363
364 R=fft_max_vr/fft_max_ir
365
366 %%%% Calculate the Oxygensaturation
367 SaO2=round(110.28-17.51*R);
368
369 %%%% Plot the values
370 set(handles.edit_hr, 'string', num2str(heartrate));
371 set(handles.edit_sao2, 'string', num2str(SaO2));
372
373 %%%% Enable the gui-buttons
374 set(handles.button_fft, 'Enable', 'on');
375 set(handles.button_timespace, 'Enable', 'on');
376
```

```
377
378 % --- Executes on button press in button_timespace.
379 function button_timespace_Callback(hObject, eventdata, handles)
380 % hObject    handle to button_timespace (see GCBO)
381 % eventdata  reserved - to be defined in a future version of MATLAB
382 % handles    structure with handles and user data (see GUIDATA)
383
384 %%%% read the filename out of the textbox
385 filename = get(handles.edit_filename, 'String');
386 data = csvread(filename);
387
388 fa=1/0.004;
389
390 %%%% read the ir- and vr-samples out of the data-array (alternating)
391 counter_ir = 1;
392 counter_vr = 1;
393 for counter_elem =1:size(data,1)
394      if mod(counter_elem,2)==1
395          ir_data(counter_ir)=data(counter_elem);
396          counter_ir=counter_ir+1;
397     elseif mod(counter_elem,2)==0
398          vr_data(counter_vr)=data(counter_elem);
399          counter_vr=counter_vr+1;
400     end
401 end
402
403 %%%% Get the Transmission-Code for choosen data
404 choose_signal = get(handles.popup_choose_signal, 'Value');
405
406
407 %%%% check for unfiltered samples
408 if(choose_signal == 1 || choose_signal == 2)
409   %if the checkbox is on, perform a IIR-HP-Filter (3rd order) with the
410   %user's cutoff frequency
411     if (get(handles.radio_hp,'Value') == get(handles.radio_hp,'Max'))
412          str_hp_fg = get(handles.edit_fg_hp, 'string');
413          hp_fg = str2num(str_hp_fg)
414
415          [b_hp a_hp] = butter(3, hp_fg/fa, 'high');
416          ir_data=filtfilt(b_hp, a_hp, ir_data);
417          vr_data=filtfilt(b_hp, a_hp, vr_data);
418     end
```

```
419   %if the checkbox is on, perform a IIR-TP-Filter (3rd order) with the
420   %user's cutoff frequency
421      if (get(handles.radio_tp,'Value') == get(handles.radio_tp,'Max'))
422          str_tp_fg = get(handles.edit_fg_tp, 'string');
423          tp_fg = str2num(str_tp_fg)
424
425          [b_tp a_tp] = butter(3, tp_fg/fa, 'low');
426          ir_data=filtfilt(b_tp, a_tp, ir_data);
427          vr_data=filtfilt(b_tp, a_tp, vr_data);
428      end
429 end
430
431 %%%% time-variable in seconds
432 time=1/fa:1/fa:length(ir_data)/fa;
433
434 %%%% Plot the data
435 plot(time,vr_data);
436 hold all
437 plot(time,ir_data);
438 hold off
439 grid on;
440
441 %%%% Plot the right description
442 legend('visible','infrared');
443 if(choose_signal == 3)
444      title('[muP]-filtered Data');
445      ylabel('Amplitude in Volt');
446 elseif (choose_signal == 2)
447      title('[muP]-OPAmp Stage 2 Data');
448      ylabel('Amplitude in binary');
449 elseif (choose_signal == 1)
450      title('[muP]-OPAmp Stage 1 Data');
451      ylabel('Amplitude in binary');
452 end
453 xlabel('time in s');
454
455 %%%% Enable the gui-buttons
456 set(handles.button_fft, 'Enable', 'on');
457 set(handles.button_timespace, 'Enable', 'on');
458
459
460 % --- Executes on button press in button_load.
```

```
461 function button_load_Callback(hObject, eventdata, handles)
462 % hObject    handle to button_load (see GCBO)
463 % eventdata  reserved - to be defined in a future version of MATLAB
464 % handles    structure with handles and user data (see GUIDATA)
465 %%%% read the filename out of the textbox
466 filename = get(handles.edit_filename, 'String');
467 data = csvread(filename);
468
469 fa=1/0.004;
470
471 %%%% read the ir- and vr-samples out of the data-array (alternating)
472 counter_ir = 1;
473 counter_vr = 1;
474 for counter_elem =1:size(data,1)
475      if mod(counter_elem,2)==1
476          ir_data(counter_ir)=data(counter_elem);
477          counter_ir=counter_ir+1;
478      elseif mod(counter_elem,2)==0
479          vr_data(counter_vr)=data(counter_elem);
480          counter_vr=counter_vr+1;
481      end
482 end
483
484 %%%% Get the Transmission-Code for choosen data
485 choose_signal = get(handles.popup_choose_signal, 'Value');
486
487
488 %%%% check for unfiltered samples
489 if(choose_signal == 1 || choose_signal == 2)
490   %if the checkbox is on, perform a IIR-HP-Filter (3rd order) with the
491   %user's cutoff frequency
492      if (get(handles.radio_hp,'Value') == get(handles.radio_hp,'Max'))
493          str_hp_fg = get(handles.edit_fg_hp, 'string');
494          hp_fg = str2num(str_hp_fg)
495
496          [b_hp a_hp] = butter(3, hp_fg/fa, 'high');
497          ir_data=filtfilt(b_hp, a_hp, ir_data);
498          vr_data=filtfilt(b_hp, a_hp, vr_data);
499      end
500   %if the checkbox is on, perform a IIR-TP-Filter (3rd order) with the
501   %user's cutoff frequency
502      if (get(handles.radio_tp,'Value') == get(handles.radio_tp,'Max'))
```

```
503         str_tp_fg = get(handles.edit_fg_tp, 'string');
504         tp_fg = str2num(str_tp_fg)
505 
506         [b_tp a_tp] = butter(3, tp_fg/fa, 'low');
507         ir_data=filtfilt(b_tp, a_tp, ir_data);
508         vr_data=filtfilt(b_tp, a_tp, vr_data);
509     end
510 end
511 
512 %%% time-variable in seconds
513 time=1/fa:1/fa:length(ir_data)/fa;
514 
515 %%% Plot the data
516 plot(time,vr_data);
517 hold all
518 plot(time,ir_data);
519 hold off
520 grid on;
521 
522 %%% Plot the right description
523 legend('visible','infrared');
524 if(choose_signal == 3)
525     title('[muP]-filtered Data');
526     ylabel('Amplitude in Volt');
527 elseif (choose_signal == 2)
528     title('[muP]-OPAmp Stage 2 Data');
529     ylabel('Amplitude in binary');
530 elseif (choose_signal == 1)
531     title('[muP]-OPAmp Stage 1 Data');
532     ylabel('Amplitude in binary');
533 end
534 xlabel('time in s');
535 
536 %%% Enable the gui-buttons
537 set(handles.button_fft, 'Enable', 'on');
538 set(handles.button_timespace, 'Enable', 'on');
539 set(handles.button_start, 'Enable', 'off');
540 
541 
542 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
543 %%% FUNCTIONS FOR THE GUI ELEMTS: POP-UP MENU
544 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
545
546 % --- Executes on selection change in popup_choose_signal.
547 function popup_choose_signal_Callback(hObject, eventdata, handles)
548 % hObject    handle to popup_choose_signal (see GCBO)
549 % eventdata  reserved - to be defined in a future version of MATLAB
550 % handles    structure with handles and user data (see GUIDATA)
551
552 % Hints: contents = cellstr(get(hObject,'String')) returns
553 %        popup_choose_signal contents as cell array
554 %        contents{get(hObject,'Value')} returns selected item
555 %        from popup_choose_signal
556
557 %%%% Get the number of the user's choice
558 choose_signal = get(handles.popup_choose_signal, 'Value');
559
560 %%%% If the user choose to transmit the muP-FFT set and deactivate the
561 %%%% gui-objects
562 if(choose_signal == 4)
563     set(handles.edit_number_of_samples, 'string', '1024');
564     set(handles.edit_scan_duration, 'string', '4.096');
565     set(handles.edit_number_of_samples, 'Enable', 'off');
566     set(handles.edit_scan_duration, 'Enable', 'off');
567     set(handles.radio_time, 'Enable', 'off');
568     set(handles.radio_number, 'Enable', 'off');
569     set(handles.edit_fg_hp, 'Enable', 'off');
570     set(handles.edit_fg_tp, 'Enable', 'off');
571     set(handles.radio_tp, 'Enable', 'off');
572     set(handles.radio_hp, 'Enable', 'off');
573     set(handles.radio_tp, 'Value', 0);
574     set(handles.radio_hp, 'Value', 0);
575
576 %%%% In case of time-samples
577 else
578     % check for unfiltered samples and set activate the IIR-radioboxes
579     if(choose_signal == 1 || choose_signal == 2)
580         set(handles.radio_tp, 'Enable', 'on');
581         set(handles.radio_hp, 'Enable', 'on');
582     % if already filtered samples should be transmitted, set the IIR-
583     % radioboxes to 0 (not choosen) and deactivate them
584     else
585         set(handles.radio_tp, 'Value', 0);
586         set(handles.radio_hp, 'Value', 0);
```

```
587         set(handles.radio_tp, 'Enable', 'off');
588         set(handles.radio_hp, 'Enable', 'off');
589     end
590
591     % Enable the radiobox to change the samplenumer
592     set(handles.radio_number, 'Enable', 'on');
593
594     if (get(handles.radio_time,'Value') == get(handles.radio_time,'Max'))
595         % Radio button is selected-take appropriate action
596         set(handles.edit_scan_duration, 'Enable', 'on');
597     else
598         % Radio button is not selected-take appropriate action
599         set(handles.edit_number_of_samples, 'Enable', 'on');
600     end
601
602 end
603
604
605 % --- Executes during object creation, after setting all properties.
606 function popup_choose_signal_CreateFcn(hObject, eventdata, handles)
607 % hObject    handle to popup_choose_signal (see GCBO)
608 % eventdata  reserved - to be defined in a future version of MATLAB
609 % handles    empty - handles not created until after all CreateFcns called
610
611 % Hint: popupmenu controls usually have a white background on Windows.
612 %       See ISPC and COMPUTER.
613 if ispc && isequal(get(hObject,'BackgroundColor'), ...
614                    get(0,'defaultUicontrolBackgroundColor'))
615     set(hObject,'BackgroundColor','white');
616 end
617
618 % --- Executes on selection change in popup_com_port.
619 function popup_com_port_Callback(hObject, eventdata, handles)
620 % hObject    handle to popup_com_port (see GCBO)
621 % eventdata  reserved - to be defined in a future version of MATLAB
622 % handles    structure with handles and user data (see GUIDATA)
623
624 % Hints: contents = cellstr(get(hObject,'String')) returns popup_com_port
625 %        contents as cell array contents{get(hObject,'Value')} returns
626 %        selected item from popup_com_port
627 cell_com_ports = getAvailableComPort();
628 set(handles.popup_com_port, 'String', cell_com_ports);
```

```
629
630 % --- Executes during object creation, after setting all properties.
631 function popup_com_port_CreateFcn(hObject, eventdata, handles)
632 % hObject    handle to popup_com_port (see GCBO)
633 % eventdata  reserved - to be defined in a future version of MATLAB
634 % handles    empty - handles not created until after all CreateFcns called
635
636 % Hint: popupmenu controls usually have a white background on Windows.
637 %       See ISPC and COMPUTER.
638 if ispc && isequal(get(hObject,'BackgroundColor'),...
639         get(0,'defaultUicontrolBackgroundColor'))
640     set(hObject,'BackgroundColor','white');
641 end
642
643
644 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
645 %%%% FUNCTIONS FOR THE GUI ELEMTS: TEXTBOX
646 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
647
648 function edit_number_of_samples_Callback(hObject, eventdata, handles)
649 % hObject    handle to edit_number_of_samples (see GCBO)
650 % eventdata  reserved - to be defined in a future version of MATLAB
651 % handles    structure with handles and user data (see GUIDATA)
652
653 % Hints: get(hObject,'String') returns contents of edit_number_of_samples
654 %        as text
655 %        str2double(get(hObject,'String')) returns contents of
656 %        edit_number_of_samples as a double
657
658 %%%% Get the number that is inside of the textbox
659 str_number_samples = get(handles.edit_number_of_samples, 'string');
660 number_samples = str2double(str_number_samples);
661
662 %%%% Check if it is a correct number and calculate the corresponding
663 %%%% scan-duration
664 if isnan(number_samples)
665     % the textboxcontent was not a number - set back to default and show a
666     % warning
667     set(handles.edit_number_of_samples,'string','1024');
668     set(handles.edit_scan_duration, 'string', '4.096');
669     warndlg('Input must be numerical');
670 else
```

```
671     % show the corresponding scan duration
672     set(handles.edit_scan_duration, 'string', ...
673         num2str(number_samples*4e-3));
674 end
675
676
677 % --- Executes during object creation, after setting all properties.
678 function edit_number_of_samples_CreateFcn(hObject, eventdata, handles)
679 % hObject    handle to edit_number_of_samples (see GCBO)
680 % eventdata  reserved - to be defined in a future version of MATLAB
681 % handles    empty - handles not created until after all CreateFcns called
682
683 % Hint: edit controls usually have a white background on Windows.
684 %       See ISPC and COMPUTER.
685 if ispc && isequal(get(hObject,'BackgroundColor'), ...
686                    get(0,'defaultUicontrolBackgroundColor'))
687     set(hObject,'BackgroundColor','white');
688 end
689
690
691 function edit_filename_Callback(hObject, eventdata, handles)
692 % hObject    handle to edit_filename (see GCBO)
693 % eventdata  reserved - to be defined in a future version of MATLAB
694 % handles    structure with handles and user data (see GUIDATA)
695
696 % Hints: get(hObject,'String') returns contents of edit_filename as text
697 %        str2double(get(hObject,'String')) returns contents of
698 %        edit_filename as a double
699
700 % --- Executes during object creation, after setting all properties.
701 function edit_filename_CreateFcn(hObject, eventdata, handles)
702 % hObject    handle to edit_filename (see GCBO)
703 % eventdata  reserved - to be defined in a future version of MATLAB
704 % handles    empty - handles not created until after all CreateFcns called
705
706 % Hint: edit controls usually have a white background on Windows.
707 %       See ISPC and COMPUTER.
708 if ispc && isequal(get(hObject,'BackgroundColor'), ...
709                    get(0,'defaultUicontrolBackgroundColor'))
710     set(hObject,'BackgroundColor','white');
711 end
712
```

```
713 function edit_fg_tp_Callback(hObject, eventdata, handles)
714 % hObject    handle to edit_fg_tp (see GCBO)
715 % eventdata  reserved - to be defined in a future version of MATLAB
716 % handles    structure with handles and user data (see GUIDATA)
717
718 % Hints: get(hObject,'String') returns contents of edit_fg_tp as text
719 %        str2double(get(hObject,'String')) returns contents of edit_fg_tp
720 %        as a double
721
722 %%%% get the cutoff-frequency out of the textbox to check it
723 str_edit_fg_tp = get(handles.edit_fg_tp, 'string');
724 edit_fg_tp = str2double(str_edit_fg_tp);
725
726 %%%% check if it is a correct number
727 if isnan(edit_fg_tp)
728     set(handles.edit_fg_tp,'string','10');
729     warndlg('Input must be numerical with a "." (Point) as a decimal mark'
        );
730 end
731
732
733 % --- Executes during object creation, after setting all properties.
734 function edit_fg_tp_CreateFcn(hObject, eventdata, handles)
735 % hObject    handle to edit_fg_tp (see GCBO)
736 % eventdata  reserved - to be defined in a future version of MATLAB
737 % handles    empty - handles not created until after all CreateFcns called
738
739 % Hint: edit controls usually have a white background on Windows.
740 %       See ISPC and COMPUTER.
741 if ispc && isequal(get(hObject,'BackgroundColor'),...
742                    get(0,'defaultUicontrolBackgroundColor'))
743     set(hObject,'BackgroundColor','white');
744 end
745
746
747
748 function edit_fg_hp_Callback(hObject, eventdata, handles)
749 % hObject    handle to edit_fg_hp (see GCBO)
750 % eventdata  reserved - to be defined in a future version of MATLAB
751 % handles    structure with handles and user data (see GUIDATA)
752
753 % Hints: get(hObject,'String') returns contents of edit_fg_hp as text
```

```
754 %        str2double(get(hObject,'String')) returns contents of edit_fg_hp
755 %        as a double
756
757 %%%% get the cutoff-frequency out of the textbox to check it
758 str_edit_fg_hp = get(handles.edit_fg_hp, 'string');
759 edit_fg_hp = str2double(str_edit_fg_hp);
760
761 %%%% check if it is a correct number
762 if isnan(edit_fg_hp)
763     set(handles.edit_fg_hp,'string','0.25');
764     warndlg('Input must be numerical with a "." (Point) as a decimal mark'
        );
765 end
766
767
768 % --- Executes during object creation, after setting all properties.
769 function edit_fg_hp_CreateFcn(hObject, eventdata, handles)
770 % hObject    handle to edit_fg_hp (see GCBO)
771 % eventdata  reserved - to be defined in a future version of MATLAB
772 % handles    empty - handles not created until after all CreateFcns called
773
774 % Hint: edit controls usually have a white background on Windows.
775 %       See ISPC and COMPUTER.
776 if ispc && isequal(get(hObject,'BackgroundColor'), ...
777                    get(0,'defaultUicontrolBackgroundColor'))
778     set(hObject,'BackgroundColor','white');
779 end
780
781
782
783 function edit_sample_duration_Callback(hObject, eventdata, handles)
784 % hObject    handle to edit_sample_duration (see GCBO)
785 % eventdata  reserved - to be defined in a future version of MATLAB
786 % handles    structure with handles and user data (see GUIDATA)
787
788 % Hints: get(hObject,'String') returns contents of edit_sample_duration
789 %        as text
790 %        str2double(get(hObject,'String')) returns contents of
791 %        edit_sample_duration as a double
792
793 %%%% get the sample-duration out of the textbox to check it
794 str_edit_sample_duration = get(handles.edit_sample_duration, 'string');
```

```
795 edit_sample_duration = str2double(str_edit_sample_duration);
796
797 %%%% check if it is a correct number
798 if isnan(edit_sample_duration)
799     set(handles.edit_sample_duration,'string','4');
800     warndlg('Input must be numerical with a "." (Point) as a decimal mark'
        );
801 end
802
803 % --- Executes during object creation, after setting all properties.
804 function edit_sample_duration_CreateFcn(hObject, eventdata, handles)
805 % hObject    handle to edit_sample_duration (see GCBO)
806 % eventdata  reserved - to be defined in a future version of MATLAB
807 % handles    empty - handles not created until after all CreateFcns called
808
809 % Hint: edit controls usually have a white background on Windows.
810 %       See ISPC and COMPUTER.
811 if ispc && isequal(get(hObject,'BackgroundColor'), ...
812                    get(0,'defaultUicontrolBackgroundColor'))
813     set(hObject,'BackgroundColor','white');
814 end
815
816 function edit_hr_Callback(hObject, eventdata, handles)
817 % hObject    handle to edit_hr (see GCBO)
818 % eventdata  reserved - to be defined in a future version of MATLAB
819 % handles    structure with handles and user data (see GUIDATA)
820
821 % Hints: get(hObject,'String') returns contents of edit_hr as text
822 %        str2double(get(hObject,'String')) returns contents of edit_hr as
823 %        a double
824
825 % --- Executes during object creation, after setting all properties.
826 function edit_hr_CreateFcn(hObject, eventdata, handles)
827 % hObject    handle to edit_hr (see GCBO)
828 % eventdata  reserved - to be defined in a future version of MATLAB
829 % handles    empty - handles not created until after all CreateFcns called
830
831 % Hint: edit controls usually have a white background on Windows.
832 %       See ISPC and COMPUTER.
833 if ispc && isequal(get(hObject,'BackgroundColor'),...
834                    get(0,'defaultUicontrolBackgroundColor'))
835     set(hObject,'BackgroundColor','white');
```

```
836 end
837
838 function edit_sao2_Callback(hObject, eventdata, handles)
839 % hObject    handle to edit_sao2 (see GCBO)
840 % eventdata  reserved - to be defined in a future version of MATLAB
841 % handles    structure with handles and user data (see GUIDATA)
842
843 % Hints: get(hObject,'String') returns contents of edit_sao2 as text
844 %        str2double(get(hObject,'String')) returns contents of edit_sao2
845 %        as a double
846
847
848 % --- Executes during object creation, after setting all properties.
849 function edit_sao2_CreateFcn(hObject, eventdata, handles)
850 % hObject    handle to edit_sao2 (see GCBO)
851 % eventdata  reserved - to be defined in a future version of MATLAB
852 % handles    empty - handles not created until after all CreateFcns called
853
854 % Hint: edit controls usually have a white background on Windows.
855 %       See ISPC and COMPUTER.
856 if ispc && isequal(get(hObject,'BackgroundColor'), ...
857                    get(0,'defaultUicontrolBackgroundColor'))
858     set(hObject,'BackgroundColor','white');
859 end
860
861
862 function edit_scan_duration_Callback(hObject, eventdata, handles)
863 % hObject    handle to edit_scan_duration (see GCBO)
864 % eventdata  reserved - to be defined in a future version of MATLAB
865 % handles    structure with handles and user data (see GUIDATA)
866
867 % Hints: get(hObject,'String') returns contents of edit_scan_duration as
868 %        text
869 %        str2double(get(hObject,'String')) returns contents of
870 %        edit_scan_duration as a double
871
872 %%%% Get the scan-duration out of the textbox
873 str_scan_duration = get(handles.edit_scan_duration, 'string');
874 scan_duration = str2double(str_scan_duration);
875
876 %%%% Check if it is a correct number
877 if isnan(scan_duration)
```

```
878     % scan_duration is not a number - set to default
879     set(handles.edit_number_of_samples,'string','1024');
880     set(handles.edit_scan_duration, 'string', '4.096');
881     warndlg('Input must be numerical with a "." (Point) as a decimal mark'
            );
882 else
883     % scan_duration is a number - check for the smallest resolution (4ms)
884     round_sample_number = round(scan_duration/(4e-3));
885     round_sample_time=round_sample_number*(4e-3);
886
887     if((scan_duration-round(scan_duration)) ~= 0)
888         warndlg('The smallest time resolution is 0.004s.');
889     end
890
891     set(handles.edit_number_of_samples, 'string', ...
892         num2str(round_sample_number));
893     set(handles.edit_scan_duration, 'string', num2str(round_sample_time));
894 end
895
896 % --- Executes during object creation, after setting all properties.
897 function edit_scan_duration_CreateFcn(hObject, eventdata, handles)
898 % hObject    handle to edit_scan_duration (see GCBO)
899 % eventdata  reserved - to be defined in a future version of MATLAB
900 % handles    empty - handles not created until after all CreateFcns called
901
902 % Hint: edit controls usually have a white background on Windows.
903 %       See ISPC and COMPUTER.
904 if ispc && isequal(get(hObject,'BackgroundColor'), ...
905                    get(0,'defaultUicontrolBackgroundColor'))
906     set(hObject,'BackgroundColor','white');
907 end
908
909
910 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
911 %%%% FUNCTIONS FOR THE GUI ELEMTS: RADIO-BUTTONS
912 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
913
914 % --- Executes on button press in radio_number.
915 function radio_number_Callback(hObject, eventdata, handles)
916 % hObject    handle to radio_number (see GCBO)
917 % eventdata  reserved - to be defined in a future version of MATLAB
918 % handles    structure with handles and user data (see GUIDATA)
```

```
919
920 % Hint: get(hObject,'Value') returns toggle state of radio_number
921
922 if (get(hObject,'Value') == get(hObject,'Max'))
923     %%%% radiobutton gets activated
924   set(handles.radio_time, 'Value', 0);
925     set(handles.edit_number_of_samples, 'Enable', 'on');
926   set(handles.edit_scan_duration, 'Enable', 'off');
927 else
928     %%%% radiobutton gets deactivated
929     set(handles.radio_number, 'Value', 1);
930 end
931
932
933 % --- Executes on button press in radio_time.
934 function radio_time_Callback(hObject, eventdata, handles)
935 % hObject    handle to radio_time (see GCBO)
936 % eventdata  reserved - to be defined in a future version of MATLAB
937 % handles    structure with handles and user data (see GUIDATA)
938
939 % Hint: get(hObject,'Value') returns toggle state of radio_time
940
941 if (get(hObject,'Value') == get(hObject,'Max'))
942     %%%% radiobutton gets activated
943   set(handles.radio_number, 'Value', 0);
944   set(handles.edit_scan_duration, 'Enable', 'on');
945     set(handles.edit_number_of_samples, 'Enable', 'off');
946 else
947     %%%% radiobutton gets deactivated
948     set(handles.radio_time, 'Value', 1);
949 end
950
951 % --- Executes on button press in radio_hp.
952 function radio_hp_Callback(hObject, eventdata, handles)
953 % hObject    handle to radio_hp (see GCBO)
954 % eventdata  reserved - to be defined in a future version of MATLAB
955 % handles    structure with handles and user data (see GUIDATA)
956
957 % Hint: get(hObject,'Value') returns toggle state of radio_hp
958
959 %%%% Check the state of the radiobox
960 if (get(hObject,'Value') == get(hObject,'Max'))
```

```
961   % Radio button is selected-take appropriate action
962      set(handles.edit_fg_hp, 'Enable', 'on');
963 else
964     % Radio button is not selected-take appropriate action
965      set(handles.edit_fg_hp, 'Enable', 'off');
966 end
967
968 % --- Executes on button press in radio_tp.
969 function radio_tp_Callback(hObject, eventdata, handles)
970 % hObject    handle to radio_tp (see GCBO)
971 % eventdata  reserved - to be defined in a future version of MATLAB
972 % handles    structure with handles and user data (see GUIDATA)
973
974 % Hint: get(hObject,'Value') returns toggle state of radio_tp
975
976 %%% Check the state of the radiobox
977 if (get(hObject,'Value') == get(hObject,'Max'))
978   % Radio button is selected-take appropriate action
979      set(handles.edit_fg_tp, 'Enable', 'on');
980 else
981     % Radio button is not selected-take appropriate action
982      set(handles.edit_fg_tp, 'Enable', 'off');
983 end
```

Anhang C: Ordnerstruktur der beigelegten CD