

Technische Universität Berlin
Fakultät IV
Institut für Energie- und Automatisierungstechnik
Fachgebiet Lichttechnik



Entwicklung und Realisierung einer Messeinrichtung mit den Sensoren AS7261 und AS72651 von ams

Bachelorarbeit

Vorgelegt von: Lennard Bödiger

Matrikelnr.: 363470

Studiengang: Technische Informatik

21. Oktober 2020

1 Einleitung

1.1 Aufbau des Messsystems

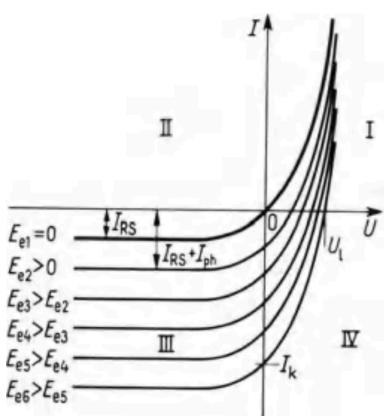
Ein Einplatiernen Computer wird mit bis zu 10 AS7261 und 10 AS72651 Sensoren verbunden. Die Messungen werden über eine I2C Bus verbindung GEsteuert und ausgelsen. Auf dem Einplatiernen Computer wird eine Datenbank mit den Messdaten gefüllt und über ein webinterface welches auch auf dem Einplatiernen Computer gehostet wird verfügbar gemacht.

2 Technische Grundlagen

2.1 Fotodioden

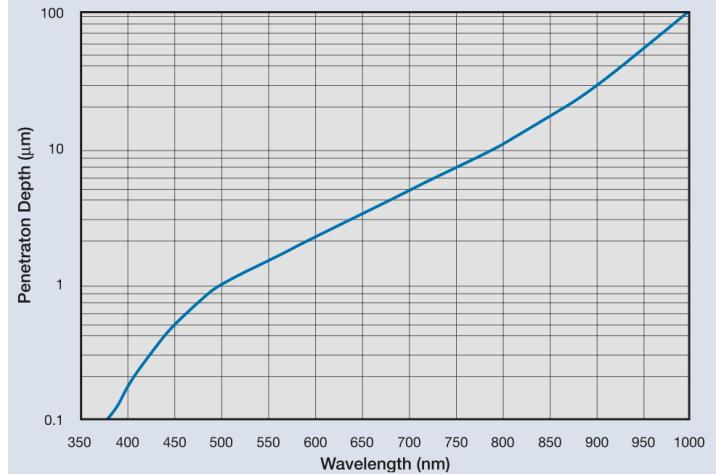
Photodioden sind beleuchtete pn-Übergänge. Im Kurzschlussbetrieb ($U = 0$) fließt ein über einen Bereich von mehr als acht Zehnerpotenzen linear von der Beleuchtungsstärke abhängiger Kurzschlussstrom I_k (Abb. 1a). Welcher sich aus dem durch ein strahlendes Licht verursachten Photostrom und dem Temperaturabhängigen Dunkel Strom zusammensetzt. Allerdings hängt der Photostrom auch von der Eindringtiefe in das Silizium Substrat ab, welche wiederum von der Wellenlänge abhängt (Abb 1b).

(a) Kennlinienfeld Photodiode



Kennlinienfeld $I = f(U)$ der Photodiode mit der Bestrahlungsstärke E_e als Parameter
Kurzschlussstrom I_k bei $U = 0$

(b) Silizium-Eindringtiefe-Licht



Quelle: osioptoelectronics.com

Damit Rückschlüsse über den Kurzschlussstrom I_K zur Lichtintensität zulässig sind, muss also die Temperatur konstant oder zumindest bekannt sein. Außerdem ist es wichtig, bei Ta-

gesichtsmessungen, dass einfallende Licht auf eine möglichst begrenzten Wellenlängenbereich zu beschränken, da so ein möglichst akkurate Temperatur und Frequenzabhängiger kompensations faktor gewählt werden kann.

2.2 I2C

I2C ist ein simples und effizientes Busprotokoll. Es wurde ursprünglich von Phillips entwickelt, wird aber seit einigen Jahren von NXP weiterentwickelt. In seiner einfachsten Form ermöglicht es einen Master mit bis zu 128 Slave Geräten zu verbinden. Dafür werden nur 2 Leitungen benötigt, die die SCL und SDA genannt werden. SCL ist die Taktleitung. Sie wird verwendet, um alle Datenübertragungen über den I2C-Bus zu synchronisieren. SDA ist die Datenleitung. Außerdem müssen alle Bus Teilnehmer mit dem gleichen GND Potential verbunden sein um Stromfluss über SDA und SCL Leitungen zu ermöglichen.

Da SCL und SDA als "open drain" betrieben werden, was bedeutet, dass die Bus Teilnehmer den Output Low aber nicht High setzen können, muss ein Pull up Widerstand zur Versorgungsspannung verwendet werden.

Die Clock Leitung SCL wird nur vom Bus Master gesteuert. Die SDA Leitung wird vom Master und Slave genutzt, allerdings antworten die Slaves im Normalbetrieb nur nachdem sie vom Master auf ihrer Adresse eine Anfrage erhalten haben. Die Spezifikation des Protokolls empfiehlt die SDA und SCL Leitung möglichst weit voneinander zu entfernen, um so die Signalqualität zu verbessern.

3 Hardware Komponenten

3.1 Sensoren

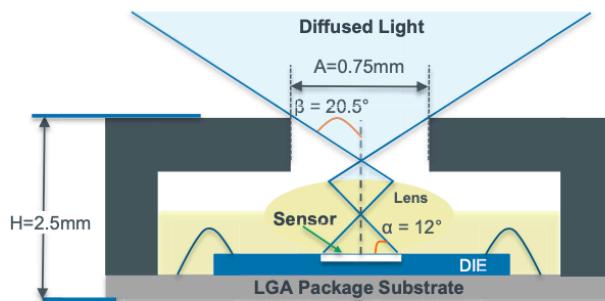
Die Sensoren aus der AS726X Reihe sind in der Lage Licht, also elektromagnetische Strahlung zu messen. In jedem Sensor sind 6 Photodioden verbaut. Vor jeder Photodiode ist ein Silizium-Interferenzfilter montiert, welcher wie ein Bandpassfilter arbeitet, er ist nur für einen bestimmten Ausschnitt des Lichtspektrums durchlässig. Jeder Baustein enthält einen Analog-digital-Wandler mit 16 Bit Auflösung, der den Photostromstrom aus den unterschiedlichen Fotodioden integriert. Nach Abschluss einer Messung wird das integrierte Ergebnis in die entsprechenden Datenregister übertragen.

So kann über das beschriebene Sensorarray die farbliche Zusammensetzung des eingestrahlten Lichts erfasst werden.

Abbildung 2: AS726X



Abbildung 3: Seitenansicht AS726X



Quelle: Datenblatt AS7261

Wie in Abb:17 dargestellt, fällt das Licht durch die Öffnung in der Mitte des Sensors ein, eine intern verbaute Linse verteilt das Licht auf die Interferenzfilter. Die Genauigkeit der Filter verändert sich je nach Einstrahlwinkel, daher ist es wichtig, vor den Sensor noch eine Streuscheibe zu montieren.

Die so gemessenen Daten können über UART oder I2C an einen Mikrocontroller übertragen werden, da über UART nur ein Gerät verbunden werden kann, eignet sich aber für diesen Anwendungsfall nur die I2C Schnittstelle.

Alle Sensoren erhalten vom Hersteller dieselbe nicht veränderbare I2C Adresse: 0x49. Daher muss ein I2C Translator genutzt werden, welcher es ermöglicht, mehrere Sensoren im gleichen Bussystem zu adressieren (siehe Abschnitt 3.3).

Die Modelle unterscheiden sich durch die verbauten Silizium-Interferenzfilter, also die unterscheidbaren Wellenlängenbereiche sowie in der benötigte Peripherie. Die grundsätzliche Messmethode ist aber immer gleich. Im Folgenden werden die verwendeten Sensoren AS7261(3.1.1) sowie der AS7265X(3.1.2) beschrieben.

3.1.1 AS7261

Das Sensorarray des AS7261 unterscheidet zwischen X,Y,Z,C,D und NIR.

Abbildung 4: AS7261-Bank Modes

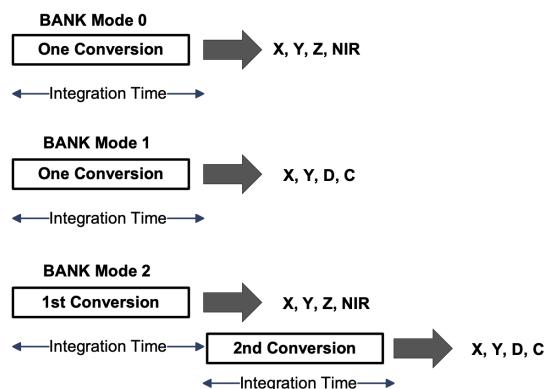
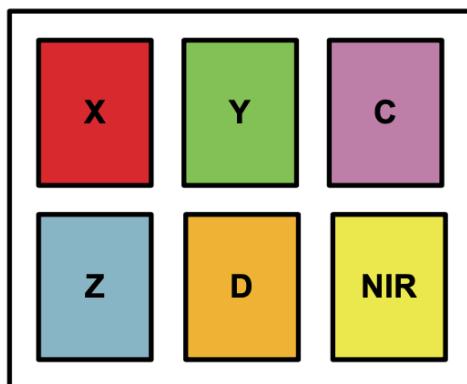


Abbildung 5: AS7261-Sensor Array



Quelle: Datenblatt AS7261

Die Ergebnisse des ADC werden direkt in die in Tabelle 1 beschriebenen Register geschrieben.

Intern im Sensor werden noch kalibrierte Werte errechnet, diese verhalten sich aber eratisch und es wird keine genaue Dokumentation bereitgestellt. Daher werden sie im Messaufbau nicht verwendet.

Tabelle 1: Your caption.

kürzel	Beschreibung	Register	Filtereigenschaften
C	Clear	0x12-0x13	kein Filter
D	Dark	0x10-0x11	"lichtundurchlässig"
NIR	nahes Infrarot	0x0E-0x0F	keine angaben
X	X (CIE 1931)	0x08-0x09	keine angaben
Y	X (CIE 1931)	0x0A-0x0B	keine angaben
Z	X (CIE 1931)	0x0C-0x0D	keine angaben

kürzel	Register
Cal-X	0x14:0x17
Cal-Y	0x18:0x1B
Cal-Z	00x1C:0x1F

Es gibt 3 sogenannte Bank Modes in denen der Sensor Arbeiten Kann.

Bank Mode 0

Die Konvertierungen erfolgen kontinuierlich und Daten sind in den I2C-Registern X, Y, Z und NIR verfügbar.

Bank Mode 1

Die Konvertierungen erfolgen kontinuierlich und Daten sind in den I2C-Registern X, Y, D und C verfügbar.

Bank Mode 2

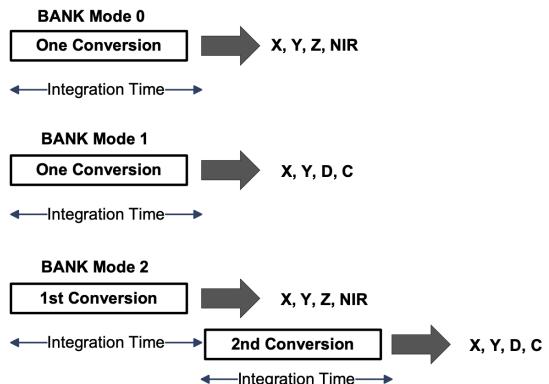
Die Konvertierungen erfolgen kontinuierlich, und Daten sind nach zwei Integrationsperioden in den Registern X, Y, Z, NIR, D und C verfügbar. In diesem Modus können dauch die kalibrierten, korrigierten Werte auch aus den entsprechenden I2C-Registern abgerufen werden.

Bank Mode 3

Die Konvertierungen erfolgen nur einmal, und Daten sind wie in Bank Mode 2 nach zwei Integrationsperioden in den Registern X, Y, Z, NIR, D und C verfügbar. Auch Die kalibrierten, korrigierten Werte auch aus den entsprechenden I2C-Registern können abgerufen werden. Das DATA RDY-Bit wird auf 1 gesetzt, sobald Daten verfügbar sind.

Für diesen Anwendungsfall wird Bankmode 3 verwendet da so an alle angeschlossenen sen-

Abbildung 6: AS7261-Bank Modes



soren möglichst gleichzeitig eine messung gestartet werden kann. Die daten können nach abgeschlossener messung an den Raspberry Pi übertragen werden.

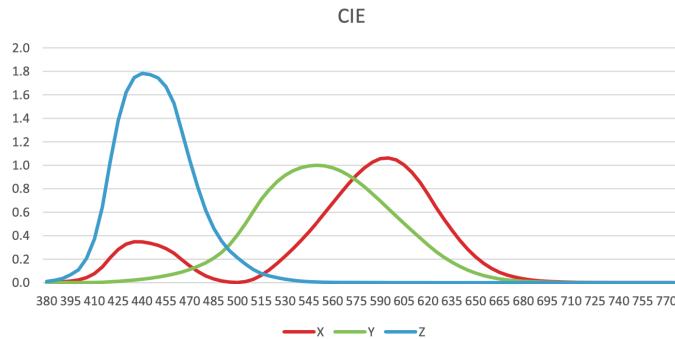


Abbildung 7: AS7261-Sensor Array

3.1.2 AS7265X

AS7265X beschreibt AS72651, AS72652 und AS72653 wobei der AS72651 als master für AS72652 und AS72653 fungiert indem er über einen weiteren separaten I2C Bus ihre Daten abfragt und ansonsten wie der AS7261 arbeitet.

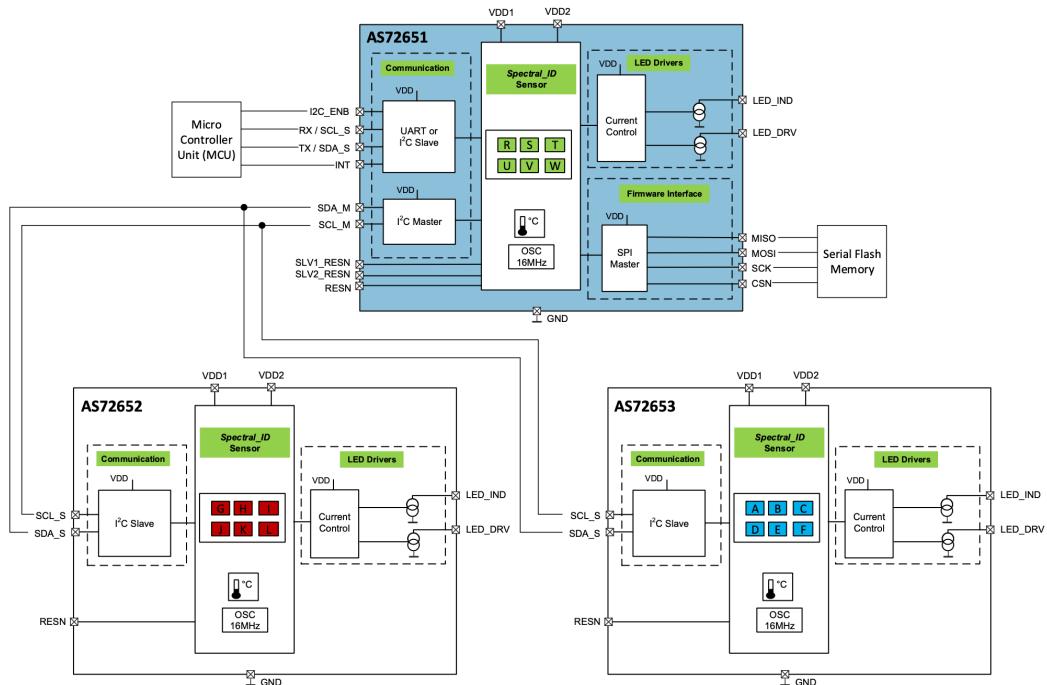


Abbildung 8: S7265X-Scematic

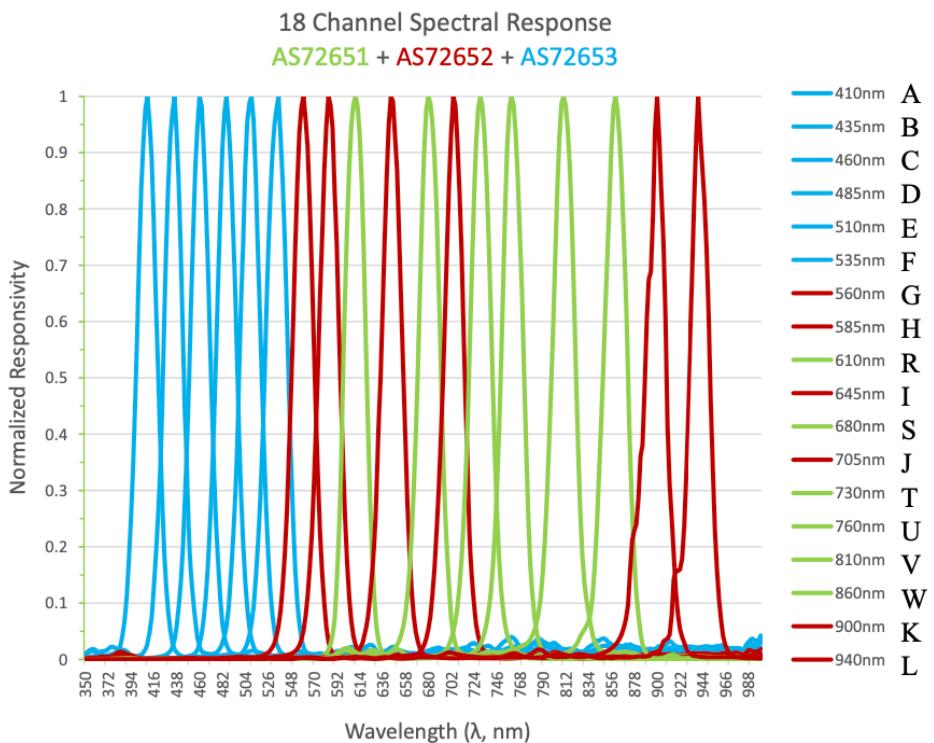


Abbildung 9: AS7261-Spectral Responsivity

Die drei Sensoren messen in Kombination mit 18 unterschiedliche Photodioden, so können sie 18 unterschiedliche Frequenz Channel im Bereich zwischen 410 nm und 940 nm mit einer Halbwertsbreite von jeweils 20 nm erfassen. Die Frequenz Channel sind wie in Abb: TODO zu sehen mit den Buchstaben A-L gekennzeichnet.

Wie der AS7261 kann der AS7265X in 3 weitgehend identischen Bankmodi betrieben werden, wobei der Bankmodus 3 verwendet wird, da alle Farbkanäle in einer manuell ausgelösten Messung aufgezeichnet werden.

3.2 Mikrocontroller

Bei der Auswahl des Mikrocontrollers ist die kleine Bauform, ausreichend langlebiger Speicher sowie eine Netzwerkschnittstelle und I2C Anschluss entscheidend.

Die abfrage der Messdaten, sowie die Messkonfiguration soll über einen Fernzugriff möglich sein. Die Daten sollen grafisch in einem Webinterface dargestellt werden ohne das eine weitere Server Instanz benötigt wird. Daher eignet sich ein Linux fähiger Einplatinencomputer besser als ein simplerer Mikrocontroller. Außerdem ermöglicht ein Einplatinencomputer einfache nachträgliche Änderungen ohne das eine komplexe Entwicklungsumgebung eingerichtet

werden muss.

Der Raspberry Pi 4 Model B erfüllt alle diese Anforderungen:

Abmessungen	85.6 mm × 56.5 mm
Speicher	eMMC Flash Module Socket
Anschlüsse	Gigabit Ethernet, USB, WLAN
GPIO	UART, I2C, IO
RAM	2 GB
CPU	Broadcom BCM2711 ARM Cortex A72
Preis	37€ (+20€ für Netzteil und 32GB SD-Karte)



Abbildung 10: Raspberry Pi 4 Model B

An den SD-Karten slot kann bis zu 256 GB Flash speicher angeschlossen werden. TODO warum reichen 32 GB.

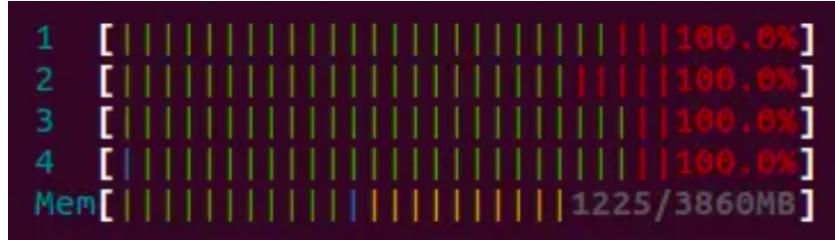
Die Internet verbindung kann über WLAN oder Ethernet hergestellt werden.

Da eine kabelgebundene Lösung mehr Zuverlässigkeit bietet, wird eine Netzwerkverbindung über Ethernet einer WLAN-Schnittstelle vorgezogen. Die GPIO des RaspebrryPi arbeitet mit 3.3V. Der 3.3V Pin des RaspebrryPi stellt nur 850mA am 3.3V output bereit. Der 5V-Pin ist direkt mit dem Netzteil verbunden und kann daher genauso viel Leistung wie das Netzteil liefern, mit Hilfe eines Abwärtswandlers ist es möglich, die Sensorplatinen mit 3,3V zu betreiben. Die CPU ist für den Anwendungsfall weitaus ausreichend dimensioniert. In Abbildung 11 ist ein Performancetest zu sehen, die reihen 1-4 beschreiben die prozentuale CPU Auslastung der jeweiligen Prozessorkerne. Mem beschreibt die Auslastung des Arbeitsspeichers.

Für den Performancetest wurde eine Messung an 10 Sensor Boards mit minimalem Messintervall gestartet. Außerdem wurden gleichzeitig Daten über Grafana exportiert.

TODO: Richtiges Bild Performance Test

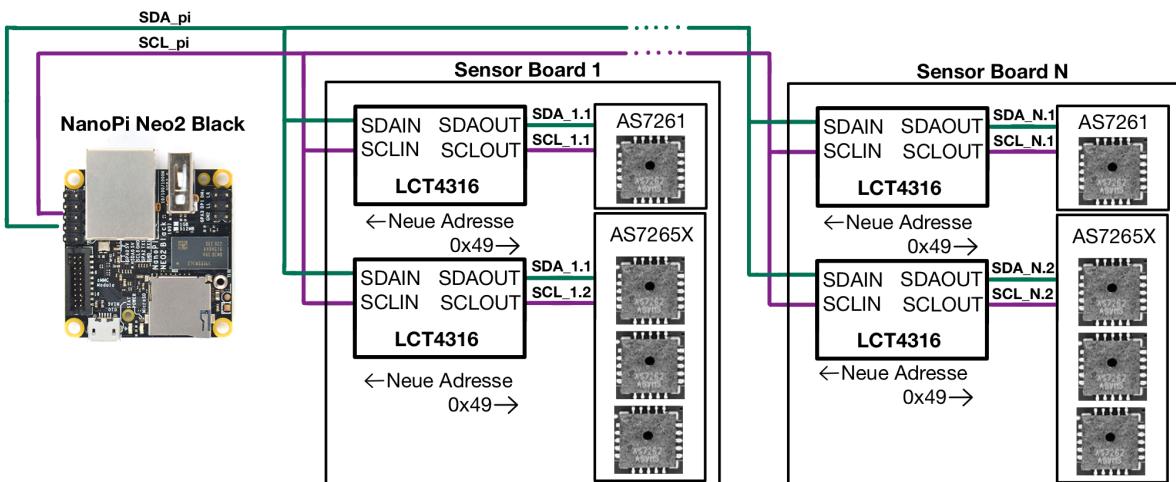
Abbildung 11: CPU-Performance



3.3 I2C Address Translator LTC4316

Da, wie in Abschnitt 3.1 genannt alle Sensoren unter derselben I2C Adresse erreichbar sind, wird ein I2C Translator genutzt, um für eine individuelle Adressierung zu sorgen.

Abbildung 12: I2C-Bus im Messaufbau



Wie in Abb 12 zu sehen wird für jeden Sensor ein LTC4316 an die Busschnittstelle des Raspberry Pi angeschlossen.(SDAIN, SCLIN).

An jeden LTC4316 wird ein AS7261 oder AS72651 angeschlossen.(SDAOUT, SCLOUT) Bei Kommunikation vom Raspberry Pi zum Sensor wird dann die I2C Adresse mit einem Faktor (Translation Byte) welcher mit diskreten Widerständen eingestellt wird mit Formel 1 verrechnet, um so die Adresse anzupassen.(XORH,XORL). Um das Translation Byte einzustellen

müssen die Widerstände R_{HT} , R_{LT} , R_{HB} und R_{LB} wie in Abb 13 am LTC4316 angeschlossen werden.

$$SensorAdresse \oplus TranslationByte = NeueAdresse \quad (1)$$

Beispiel Rechnung

$$0x49 \oplus 0x01 = 0x48$$

$$0x49 \oplus 0x02 = 0x4b$$

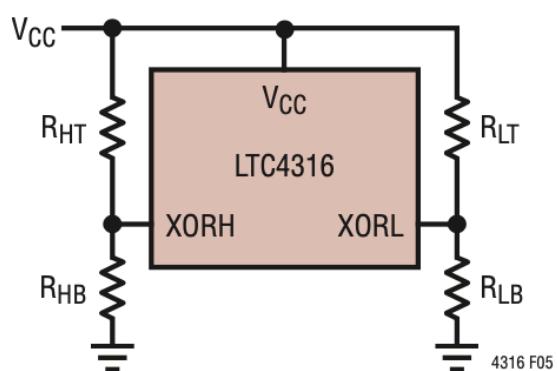
$$0x49 \oplus 0x05 = 0x4c$$

$$0x49 \oplus 0x06 = 0x4f$$

$$0x49 \oplus 0x0A = 0x43$$

$$0x49 \oplus 0x49 = 0x00$$

Abbildung 13: Translation Byte



In Tabelle 2 und 3 sind die unterschiedlichen Konfigurationen des Translation Bytes aufgelistet.

Im Handbuch ?? ist eine Liste zu finde welche Translation Bytes bereits verwendet werden. Wenn weitere Sensor Boards angefertigt werden ist die Liste dort weiter zu pflegen.

Tabelle 2: Untere 4 Bit des Translation Byte

a3	a2	a1	a0	R_{LT}	R_{LB}
0	0	0	0	Open	Short
0	0	0	1	976	102
0	0	1	0	976	182
0	0	1	1	1000	280
0	1	0	0	1000	392
0	1	0	1	1000	523
0	1	1	0	1000	681
0	1	1	1	1000	887
1	0	0	0	887	1000
1	0	0	1	681	1000
1	0	1	0	523	1000
1	0	1	1	392	1000
1	1	0	0	280	1000
1	1	0	1	182	976
1	1	1	0	102	976
1	1	1	1	Short	Open

Tabelle 3: Obere 3 Bit des Translation Byte

a6	a5	a4	R_{LT}	R_{LB}
0	0	0	Open	Short
0	0	1	976	102
0	1	0	976	182
0	1	1	1000	280
1	0	0	1000	392
1	0	1	1000	523
1	1	0	1000	681
1	1	1	1000	887

3.4 Companion Flash

Die Sensoren AS7261 und AS72651 benötigen einen flash speicher von welchem sie ihre Firmware laden können. Die jeweilige Firmware von AMS wird mithilfe von Flashcat-USB, einem USB Memory Programmer über das SPI Protokoll auf den den Flash Speicher übertragen. Der AT25SF041-SSHD-B wurde aus der von AMS bereitgestellten liste kompatiebler Flash speichern ausgewählt da er am günstigsten ist. Da die Firmware nur auf nachfrage bei AMS erhältlich. Für den Messaufbau werder folgende versionen verwendet die .bin files sind im anhang zu finden.

AS7261_complete ??

AS726X_AS7265_complete_moonlight_v1 ??

4 Platine

4.0.1 System Topologie

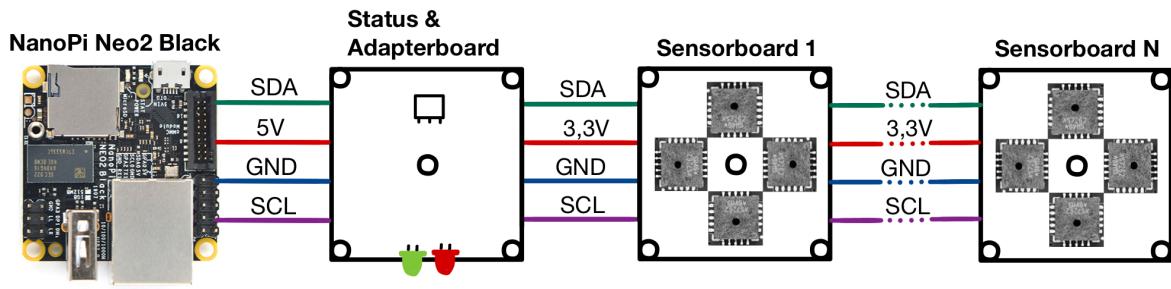


Abbildung 14: Verschaltung der der Platienen

Der Msssaufbau besteht aus einem Rapberry Pi 4 Model B welcher über eine status und Adapterplatinene mit 1-10 Sensnorboards verbunden werden kann.

4.0.2 Status & Adapterboard

Da der Rapberry Pi nicht die benötigte 3,3V Stromversorgung bereitstellt wird eine Adapter-Shield mit einem Spannungswandler(LM3940IT-3.3) verwendet. Auf dem Adapter-Shield findet der einheitliche Steckverbinder sowie die Pull Up widerstände des I2C Bus Platz, da das Adapter-Shield aufgrund seiner Bauform nicht falsch montiert werden kann so die Verpolung der Sensoren ausgeschlossen werden. Die Bedeutung der Status LED wird im Handbuch ?? erläutert.

Der überflüssige Platz auf dem Adapter-Shield wurde für einen Prototyping-Bereich genutzt.

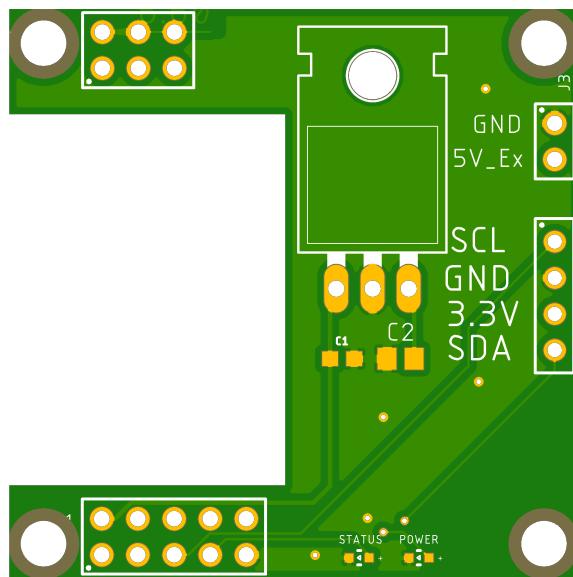


Abbildung 15: Verschaltung der der Platienen

4.0.3 Sensorboard

Die hauptaufgabe der Sensorboard Platine ist es den AS7261 und der AS7265X mit ihrem companion flash (??) und über den I2C Translator (??) dem I2C Bus verbunden. Außerdem werden verschiedene LED, Widerstände und Kondensatoren verbaut.

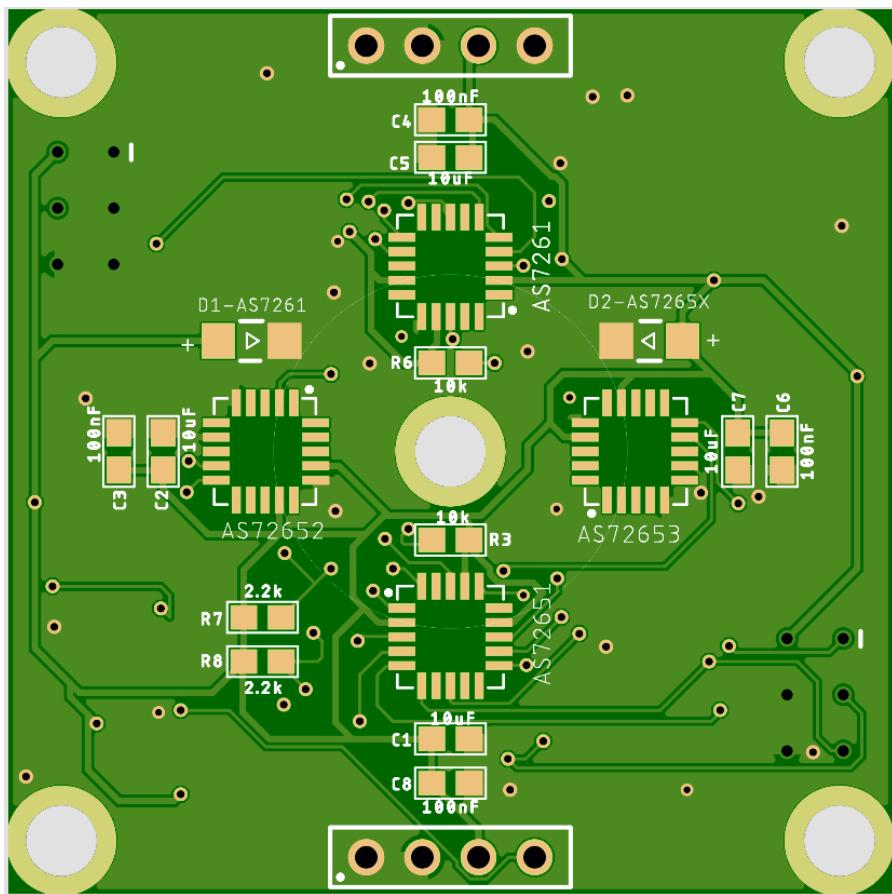


Abbildung 16: Verschaltung der der Platienen

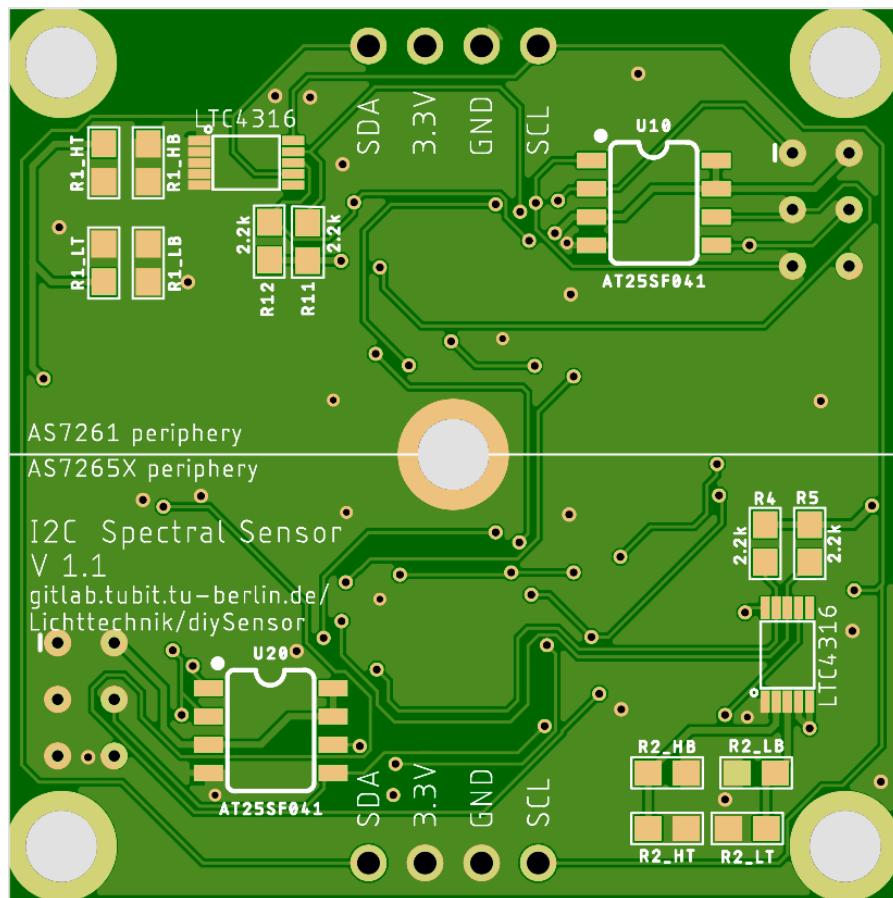


Abbildung 17: Verschaltung der der Platienen

Status LED: Am AS7261 und AS7265X Befindet sich jewals eine Rote Status LED wenn es ein problem mit dem Compairain Flash gibt fängt sie an zu blinken im normalbetrieb kann die LED Softwareseitg ein und ausgeschaltet werden. Wärend der Messubng solle sie ausgeschaltet werden da das Rote licht sonst die Messung verfälscht.

Pull-up-Widerstände: R7 und R8 sind die Pullup Widerstände des seperaten I2C Bus welcher die AS7265X Sensoren miteinander verbindet.

R12 und R11 sind die Pullup Widerstände des I2C Bus welcher den AS7261 mit seinem LTC4316 verbindet.

R4 und R5 sind die Pullup Widerstände des I2C Bus welcher den AS72651 mit seinem LTC4316 verbindet.

I2C Enable Wiederstände: Pin R6 ist mit 3.3V und Pin 8 (I2C Enable) des AS7261 verbunden. So wird der AS7261 in den I2C modus gebracht. R3 erfüllt die gleiche Aufgabe für den AS72651.

Translation Byte Wiederstände: Die 8 Wiederstände auf der Rückseite R1_XX und R2_XX sind die in ?? beschriebenen wiederstände welche das Translation Byte einstellen. Die wiederstände R1_XX bestimmen die Adresse des AS7261 und R1_XX die Adresse des AS72651.

Entstörkondensatoren: Das Datenblatt der Sensoren empfiehlt für jeden Sensor 2 Entstörkondensatoren möglichst nah am Sensor zwischen GND und VCC anzubringen.

Kürzel	Wert	Sensor
C5	$10\mu F$	AS7261
C4	$100nF$	AS7261
C1	$10\mu F$	AS72651
C8	$100nF$	AS72651
C2	$10\mu F$	AS72652
C3	$100nF$	AS72652
C7	$10\mu F$	AS72653
C6	$100nF$	AS72653

I2C Lanes: Die I2C Leitungen haben ihren Ursprung auf der Adapterplatine und werden mithilfe der seitlichen Steckverbinder über die Sensorsboards durchgeschleift. Die maximal mögliche Länge einer I2C Leitung hängt von der Länge der Leitungskapazität sowie äußeren Störeinflüssen ab. Die Data und die Clock Leitung sind möglichst weit von anderen Datenleitungen also auch von einander entfernt platziert, da so Störeinflüsse durch elektromagnetische Felder minimiert werden. Außerdem wurde darauf geachtet dass die Leitungen auf den Platinen die gleiche Länge haben da sich sonst die Differenzen der Leitungslänge mit jeder angeschlossenen Platine addiert und es zu Timing Differenzen zwischen der Daten und Clock Leitung kommt. Der Verlauf der Leitungen ist in Abb?? Rot gekennzeichnet.

Connector: An Rechts und Links an der Platine befinden sich 4 durchkontakteierte Löcher, hier können unterschiedliche Steckverbinder mit 2.54 mm Pitch montiert werden. Es empfiehlt sich, verpolungssichere Steckverbinder zu verwenden, um Hardware Schäden vorzubeugen.

Für dieses Bauteil wurde keine SMD-Technik, sonder Durchsteckmontage gewählt, da so eine bessere mechanische Festigkeit erreicht wird. Außerdem kann so alternativ zu einem Steckverbinder direkt ein Flachbandkabel angelötet werden.

5 Datenbank & Webinterface

6 C Code

6.1 wiringPi

Auf dem Raspberry Pi 4 ist die wiringPi Libarry vorinstalliert. Die Funktionalität der drei verwendeten C-Funktionen wird im folgenden beschrieben.

6.1.1 wiringPiI2CSetup

Der Funktion wird beim Aufruf die I2C Adresse übergeben mit welcher eine Verbindung aufgebaut werden soll: *wiringPiI2CSetup(address)*.

Der Rückgabewert ist der Standard Linux File Descriptor oder -1, falls ein Fehler auftritt. Die Anzahl der File Descriptoren ist begrenzt daher muss die Verbindung mit der Funktion *close()* aus der Library: unistd.h geschlossen werden wenn sie nicht mehr benötigt wird.

6.1.2 wiringPiI2CWriteReg8

Der Funktion wird beim Aufruf der File Descriptor einer aktiven I2C Verbindung sowie ein Zielregister und die zu schreibenden 8bit Daten übergeben:

wiringPiI2CWriteReg8(fd, RegAdr, 8BitData). Wenn der Schreibzugriff vom I2C Gerät bestätigt wurde wird eine 0 zurück gegeben.

6.1.3 wiringPiI2CReadReg8

Der Funktion wird beim Aufruf der File Descriptor einer aktiven I2C Verbindung sowie ein Zielregister übergeben: *wiringPiI2CReadReg8(fd, RegAdr)*. Der gelesene 8-Bit-Inhalt des Registers ist der Rückgabewert. Wenn das Lesen Fehlschlägt bleibt das Programm in einer Endlosschleife hängen.

6.2 AS726X Library

Die WiringPi_AS726X_Library enthält alle Funktionen um den AS7261 und AS7265X zu steuern und auszulesen. Da der Beispiel Code und die sonstigen Angaben im Datenblatt in vielen Detailfragen ungenau und Fehlerhaft sind wurde die Arduino OpenSource Libarrys von sparkfun SparkFun_AS726X_Arduino_Library-master und SparkFun_AS7265x_Arduino_Library als Implementierungsgrundlage verwendet. Im ersten schritt der Entwicklung wurde sie für die I2C Schnittstelle, wiringPi Libarry des Raspberry Pi 4 umgeschrieben und anschließen in ihrer Funktionalität erweitert um für das Messsystem mit mehreren Sensoren auf dem gleichen I2C-Bus nutzbar zu sein.

Im Folgenden Text werden die Register Adressen und mit den gleichen Namen wie im source code bezeichnet die Numerische Adressen sind in Tabelle ?? aufgelistet. TODO table

STATUS Register	I ² C slave interface STATUS register Read-only	Register Address = 0x00 Bit 1: TX_VALID 0 → New data may be written to WRITE register 1 → WRITE register occupied. Do NOT write. Bit 0: RX_VALID 0 → No data is ready to be read in READ register. 1 → Data byte available in READ register.
WRITE Register	I ² C slave interface WRITE register Write-only	Register Address = 0x01 8-Bits of data written by the I ² C Master intended for receipt by the I ² C slave. Used for both virtual register addresses and write data.
READ Register	I ² C slave interface READ register Read-only	Register Address = 0x02 8-Bits of data to be read by the I ² C Master.

Abbildung 18: PysicalRegister

Abbildung 19: AS726x_CONTROL_SETUP 0x04

Bit	Bit Name	Default	Access	Bit Description
7	RST	0	R/W	Soft Reset, Set to 1 for soft reset, goes to 0 automatically after the reset
6	INT	0	R/W	Enable interrupt pin output (INT), 1: Enable, 0: Disable
5:4	GAIN	10	R/W	Sensor Channel Gain Setting (all channels) 'b00=1x; 'b01=3.7x; 'b10=16x; 'b11=64x;
3:2	BANK	10	R/W	Data Conversion Type (continuous) 'b00=Mode 0: X, Y, Z and NIR 'b01=Mode 1: X, Y, D and C 'b10=Mode 2: X, Y, Z, NIR, D and C 'b11=Mode 3: One-Shot operation
1	DATA_RDY	0	R/W	1: Data Ready to Read, sets INT active if interrupt is enabled. Can be polled if not using INT.
0	RSVD	0	R	Reserved; Unused

Quelle: Datenblatt AS7261

Abbildung 20: AS726x_INT_T 0x05

Bit	Bit Name	Default	Access	Bit Description
7:0	INT_T	0xFF	R/W	Integration time = <value> * 2.8ms

Quelle: Datenblatt AS7261

Abbildung 21: AS726x_DEVICE_TEMP 0x06

Bit	Bit Name	Default	Access	Bit Description
7:0	Device_Temp	0xFF	R/W	Internal device temperature data byte (°C)

Quelle: Datenblatt AS7261

Abbildung 22: AS726x_LED_CONTROL 0x07

Bit	Bit Name	Default	Access	Bit Description
7:6	RSVD	0	R	Reserved
5:4	ICL_DRV	00	R/W	LED_DRV current limit 'b00=12.5mA; 'b01=25mA; 'b10=50mA; 'b11=100mA;
3	LED_DRV	0	R/W	Enable LED_DRV 1: Enabled; 0: Disabled
2:1	ICL_IND	00	R/W	LED_IND current limit 'b00=1mA; 'b01=2mA; 'b10=4mA; 'b11=8mA;
0	LED_IND	0	R/W	Enable LED_IND 1: Enabled; 0: Disabled

Quelle: Datenblatt AS7261

Abbildung 23: AS7265X_DEV_SELECT_CONTROL 0x4F

Bit	Bit Name	Default	Access	Bit Description
5	AS72653_id	0	R	Second slave Available
4	AS72652_id	0	R	First slave available
1:0	SELECT DATA	00	R/W	0x00: Select master data 0x01: Select first slave data 0x02: Select second slave data

Quelle: Datenblatt AS7265X

Tabelle 4: Raw Data Channel Registers

Name	Adresse
AS7261_X	0x08
AS7261_Y	0x0A
AS7261_Z	0x0C
AS7261_NIR	0x0E
AS7261_DARK	0x10
AS7261_CLEAR	0x12
AS7265X_R_G_A	0x08
AS7265X_S_H_B	0x0A
AS7265X_T_I_C	0x0C
AS7265X_U_J_D	0x0E
AS7265X_V_K_E	0x10
AS7265X_W_L_F	0x12

6.2.1 virtualWriteRegister

Wie bei Embedded-Geräten üblich werden Einstellungen auf dem Sensor verändert indem verschiedene sogenannte Special Function Register mit Daten beschrieben werden.

Jedes Special Function Register ist 8Bit groß und hat eine Adresse. Jedes Bit des Register repräsentiert eine Einstellung. Beispielsweise ist 0x07 (Abb. 22) das LED Control Register des Sensors. Bit 0 des Registers Beschreibt den Zustand der Status LED. Die Restlichen 7 Bit des Registers Beschreibt den Zustand anderer LEDs die für den Messaufbau aber irrelevant sind.

Wird Register 7 mit dem Dezimalwert 0 beschrieben sind alle LED aus, wird es mit dem Dezimalwert 1 beschrieben leuchtet nur die Status LED. Die Register Lassen sich aber nicht direkt Beschreiben, stattdessen sind sie als sogenannte Virtuelle Register implementiert. Das heißt das nur Register 0x01 (WRITE Register Abb. 18) beschrieben werden kann. Um Daten in eins der Special Funktion Register zu schreiben wird die C-Funktion virtualWriteRegister verwendet. Die Funktionsweise lässt sich in 4 schritten zusammenfassen:

- Zeile 8 wartet bis das WRITE Register leer ist, was angezeigt wird indem das Bit AS72XX_TX_VALID im Register AS72XX_STATUS_REG den Wert 1 annimmt.
- Zeile 14 Schreibt die Virtuelle Adresse in das WRITE Register und Setzt zusätzlich Bit 8 des WRITE Register auf 1 um zu zeigen das es sich um einen Schreibenden zugriff auf das Virtuelle Register handelt.
- Zeile 20 wartet erneut bis das WRITE Register leer ist.
- Zeile 26 schreibt die Daten in das WRITE Register

Der Sensor wird jetzt selber die übertragenen Daten aus dem WRITE Register in das angegebene Virtuelle Register kopieren.

```
1 // Write to a virtual register in the AS726x
2 void virtualWriteRegister(uint8_t virtualAddr, uint8_t ←
3     dataToWrite, int fd){
4     uint8_t status;
5     //Wait for WRITE register to be empty
6     while (1){
7         status = wiringPiI2CReadReg8(fd, AS72XX_SLAVE_STATUS_REG);
8         if((status & AS72XX_SLAVE_TX_VALID) == 0) {
9             break; // No inbound TX pending at slave. Okay to ←
10            write now.
11        }
12    }
13 }
```

```

11         delay(POLLING_DELAY);
12     }
13     // Send the virtual register address (setting bit 7 to ←
14     // indicate a write a register).
14     wiringPiI2CWriteReg8(fd, AS72XX_SLAVE_WRITE_REG, ←
15     (virtualAddr | 0x80));
16
16     //Wait for WRITE register to be empty
17     while (1)
18     {
19         status = wiringPiI2CReadReg8(fd, ←
20             AS72XX_SLAVE_STATUS_REG);
21         if ((status & AS72XX_SLAVE_TX_VALID) == 0){
22             break; // No inbound TX pending at slave. Okay to ←
23             write now.
24         }
25         delay(POLLING_DELAY);
26     }
25     // Send the data to complete the operation.
26     wiringPiI2CWriteReg8(fd, AS72XX_SLAVE_WRITE_REG, ←
27     dataToWrite);
27 }
```

6.2.2 virtualReadRegister

Die Unterschiedlichen Messdaten des Sensors werden in dedizierten Registern gespeichert. Es ist aber nur über den indirekten weg des AS72XX_READ_REG und der Virtuellen Register Adressen möglich Daten auszulesen. Die Funktionsweise der zum Daten auslesen benötigten Funktion `virtualReadRegister` lässt sich wieder in 4 Schritte aufteilen.

- Das AS72XX_READ_REG wird ausgelesen ohne das die Daten verarbeitet werden. Dieser schritt ist wie ein Reset des Registers zu verstehen.
- Zeile 12 schreibt die Virtuelle Adresse in das WRITE Register und setzt zusätzlich Bit 8 des WRITE Register auf 0 um zu zeigen das es sich um einen Lesenden zugriff auf das Virtuelle Register handelt.
- Sobald das AS72XX_STATUS_REG den Wert AS72XX_TX_VALID annimmt sind die Daten aus dem angebenden Virtuellen Register in das AS72XX_READ_REG kopiert worden.

- Zeile 28 liest die Daten aus dem S72XX_READ_REG
-

```

1 //Read a virtual register from the AS726x
2 uint8_t virtualReadRegister(uint8_t virtualAddr, int fd){
3     uint8_t status;
4     //Do a prelim check of the read register
5     status = wiringPiI2CReadReg8(fd, AS72XX_SLAVE_STATUS_REG);
6     if ((status & AS72XX_SLAVE_RX_VALID) != 0){ //There is data ←
7         to be read
8         /*uint8_t incoming = */wiringPiI2CReadReg8(fd, ←
9             AS72XX_SLAVE_READ_REG); //Read the uint8_t but do ←
10            nothing with it
11    }
12    //Wait for WRITE flag to clear
13    while (1) {
14        status = wiringPiI2CReadReg8(fd, ←
15            AS72XX_SLAVE_STATUS_REG);
16        if ((status & AS72XX_SLAVE_TX_VALID) == 0){
17            break; // If TX bit is clear, it is ok to write
18        }
19        delay(POLLING_DELAY);
20    }
21    // Send the virtual register address (bit 7 should be 0 to ←
22    // indicate we are reading a register).
23    wiringPiI2CWriteReg8(fd, AS72XX_SLAVE_WRITE_REG, ←
24        virtualAddr);
25
26    //Wait for READ flag to be set
27    while (1)
28    {
29        status = wiringPiI2CReadReg8(fd, ←
30            AS72XX_SLAVE_STATUS_REG);
31        if ((status & AS72XX_SLAVE_RX_VALID) != 0) break; // ←
32            Read data is ready.
33        delay(POLLING_DELAY);
34    }
35
36    uint8_t incoming = wiringPiI2CReadReg8(fd, ←
37        AS72XX_SLAVE_READ_REG);
38    return (incoming);
39 }

```

6.2.3 MeasurementFromAddress

Die Funktion baut einen I2C Verbindung zur übergebenen Bus-Adresse auf und ruft die Funktion takeMeasurements mit dem File Descriptor der aktiven I2C Verbindung auf. Nachdem die Funktion takeMeasurements durchlaufen ist wird die I2C Verbindung wieder geschlossen.

```
1 //Calls takeMeasurements on gives I2C Address
2 void MeasurementFromAddress(int address){
3     int fd = wiringPiI2CSetup(address);
4     if (fd == -1) {
5         printf("i2c failed");
6     }
7     takeMeasurements(fd); // takesMeasurements Readings can now ←
                           be accessed via getX(), getY(), etc
8     close(fd);
9 }
```

6.2.4 takeMeasurements

Die Funktion takeMeasurements ruft die Funktion setMeasurementMode mit dem Parameter 3 auf das setzt den aus ?? bekannten Bankmode der übergebenen I2C Verbindung (fd) auf Bank Mode 3. Die On-Shot Messung wird sofort gestartet, in Zeile 9 wird gewartet bis die Messung abgeschlossen ist. Um sicherzustellen das die Funktion DataAvailable richtig arbeitet muss vor der Messung das Flag DataAvailable auf 0 gesetzt werden (Zeile 3). Die Daten werden hier nicht ausgelesen daher gibt es keinen Rückgabewert.

```
1 //Tells IC to take measurements and polls for data ready flag
2 void takeMeasurements(int fd) {
3     clearDataAvailable(fd); //Clear DATA_RDY flag when using ←
                           Mode 3
4
5     //Goto mode 3 for one shot measurement of all channels
6     setMeasurementMode(3, fd);
7
8     //Wait for data to be ready
9     while (dataAvailable(fd) == 0) delay(POLLING_DELAY); ←
                           //Potential TODO: avoid this to get faster nearly ←
                           parallel measurements
```

10

```
11     //Readings can now be accessed via getViolet(), getBlue(), ←  
     etc  
12 }
```

6.2.5 setMeasurementMode

Mit der Funktion setMeasurementMode werden die Bankmode Bits 2 und 3 des AS726x_CONTROL_SETUP Registers mit dem gewünschten Wert für den Bankmode beschrieben.

Da die anderen Bits des Registers noch weitere Einstellungen repräsentieren welche nicht verändert werden sollen, muss das Register erst ausgelesen werden. Anschließend werden die Bankmode Bits auf 0 gesetzt um im nächsten schritt mit dem gewünschten neuen Bankmodewert beschrieben zu werden. Die Bedeutung der Bankmodes ist in ?? erläuteret.

```
1 //Sets the measurement mode  
2 //Mode 0: Continuous reading of VBGY (7262) / STUV (7263)  
3 //Mode 1: Continuous reading of GYOR (7262) / RTUX (7263)  
4 //Mode 2: Continuous reading of all channels (power-on default)  
5 //Mode 3: One-shot reading of all channels  
6 void setMeasurementMode(uint8_t mode, int fd) {  
7     if (mode > 0b11) mode = 0b11;  
8  
9     //Read, mask/set, write  
10    uint8_t value = virtualReadRegister(AS726x_CONTROL_SETUP, ←  
11        fd); //Read  
12    value &= 0b11110011; //Clear BANK bits  
13    value |= (mode << 2); //Set BANK bits with user's choice  
14    virtualWriteRegister(AS726x_CONTROL_SETUP, value, fd); ←  
        //Write  
15 }
```

6.2.6 dataAvailable & clearDataAvailable

Das Bit dataAvailable im Register AS726x_Control_Setup wird vom Sensor auf 1 gesetzt wenn nach einer Messung neue Daten vorhanden sind, Interrupts müssen dafür ausgeschaltet sein. dataAvailable wird auf 0 gesetzt wenn Daten gelesen werden. Wenn eine One-Shot Messung im Bankmode 3 durchgeführt wird muss das dataAvailable Bit mit der Funktion clearDataAvailable auf 0 gesetzt werden, da sonst nicht sicher gestellt ist das die Daten vor der Messung

gelesen wurden besteht die Möglichkeit das sich das Bit fälschlicherweise noch im falsch positiven zustand befindet. Die Funktion dataAvailable gibt den Wert des DataAvailable Bit zurück.

6.2.7 Rohwerte des AS7261 Auslesen

Die in ?? beschrieben 6 Channel des AS7261 werden mit den folgenden Funktionen auslesen:

- getX_CIE(fd)
- getY_CIE(fd)
- getZ_CIE(fd)
- getNIR(fd)
- getDark(fd)
- getClear(fd)

Der File Deskriptor einer I2C Verbindung mit einem AS7261 und das zu lesende Register(Abb. 4) wird als Übergabe Parameter erwartet.

Um den Messwert auszulesen, wir die Funktion getchannel aufgerufen. Der Rückgabewert ist der 16-Bit Messwert aus dem jeweiligen Register vom Datentyp integer.

```
1 //Get RAW AS7261 readings
2 int getX(int fd) { return(getChannel(AS7261_X, fd));}
3 int getY(int fd) { return(getChannel(AS7261_Y, fd));}
4 int getZ(int fd) { return(getChannel(AS7261_Z, fd));}
5 int getNIR(int fd) { return(getChannel(AS7261_NIR, fd));}
6 int getDark(int fd) { return(getChannel(AS7261_DARK, fd));}
7 int getClear(int fd) { return(getChannel(AS7261_CLEAR, fd));}
```

6.2.8 getChannel

Da die Messdaten 16-Bit groß sind, der Sensor aber nur über 8 Bit Register verfügt werden 2 aufeinander folgende Register ausgelsen und im Big-Endian Format aneinander geheftet. Die Funktion getChannel erwartet den File Descriptor einer I2C Verbindung zu einem Sensor und die Adressen des High Bytes eines Raw Data Registers. Der Rückgabewert ist der 16-Bit

Messwert aus dem jeweiligen Register vom Datentyp Integer.

```
1 //A the 16-bit value stored in a given channel registerReturns
2 int getChannel(uint8_t channelRegister, int fd){
3     int colorData = virtualReadRegister(channelRegister, fd) ←
4         << 8; //High uint8_t
5     colorData |= virtualReadRegister(channelRegister + 1, fd); ←
6         //Low uint8_t
7     return(colorData);
8 }
```

6.2.9 Rohwerte des AS7265X Auslesen

Da beim lesen des AS7265X, 3 Sensoren unter der gleichen Adresse erreichbar sind (Abb. 4), muss zusätzlich zum File Disciptor und des ziel Registers der Device Identifier angeben werden. Die folgenden Funktionen übernehmen diese Aufgaben und können genutzt werden, um Rohdaten auslesen:

AS72651	AS72652	AS72653
getR(fd)	getG(fd)	getA(fd)
getS(fd)	getX(fd)	getB(fd)
getT(fd)	getI(fd)	getC(fd)
getU(fd)	getJ(fd)	getD(fd)
getV(fd)	getK(fd)	getE(fd)
getW(fd)	getL(fd)	getF(fd)

Um den Messwert auszulesen, wir die Funktion `getChannel_AS7265X` aufgerufen.

Der Rückgabewert ist der 16-Bit Messwert aus dem jeweiligen Register vom Datentyp integer.

6.2.10 `getChannel_AS7265X`

Die Funktion verarbeitet den aus ?? bekannten Device Identifier indem die Funktion `selectDevice` aufgerufen wird. Da es aber keine Möglichkeit gibt zu überprüfen ob der Gerätewechsel erfolgreich war muss vorher überprüft werden ob das jeweiligen Slavegerät AS72652 oder AS72653 vorhanden ist. Ist ein Slave nicht vorhanden wird der Wert -1 zurück gegeben.

Ohne diese Überprüfung ist nicht sichergestellt das keine falschen Werte aus den gleich nummerierten Registern des AS72651 ausgelesen werden, obwohl Werte eines nicht vorhandenen oder falsch aufgelötzten Slavesensors (AS72652 / AS72653) erwartet werden. Das eigentliche auslesen des 16-Bit Messwerts erfolgt mithilfe der Funktion getChannel, das Ergebnis ist der Rückgabewert der Funktion getChannel_AS7265X.

```

1 // returns Color channel of AS7265X
2 // returns -1 if slave AS72651 or AS72652 is not available
3 int getChannel_AS7265X(int device, uint8_t channelRegister, int ←
4     fd){
5     selectDevice(AS72651_id, fd);    //select AS72651 to verify ←
6     presence of slave sensors
7     if (device == AS72651_id){
8         return getChannel(channelRegister, fd);
9     }
10    else if(device == AS72652_id && scan_AS7262(fd)){
11        selectDevice(device, fd);
12        return getChannel(channelRegister, fd);
13    }
14    else if (device == AS72653_id && scan_AS7263(fd)){
15        selectDevice(device, fd);
16        return getChannel(channelRegister, fd);
17    }
18    return -1;
19 }
```

6.2.11 selectDevice

Die Notwendigkeit für selectDevice wurde in ?? bereits erläutert. Laut Datenblatt sollen nur die Bits 0 und 1 des DEV_SELECT_CONTROL Registers beschrieben werden, das stimmt aber nicht.

In der Realität muss das gesamte 8-bit Register mit folgenden Werten beschrieben werden um beim anschließenden Lesevorgang Daten vom jeweiligen Sensor zu erhalten.

DEV_SELECT_CONTROL	Sensor
0x00	AS72651
0x01	AS72652
0x02	AS72653

```
1 //Select which AS7265X Device is used
2 //AS72651_id or AS72652_id or AS72653_id
3 void selectDevice(uint8_t device, int fd) {
4     //Set the bits 0:1. Just overwrite whatever is there because ←
5         masking in the correct value doesn't work.
6     virtualWriteRegister(AS7265X_DEV_SELECT_CONTROL, device, fd);
7 }
```

6.2.12 Enable/Disable Indicator

Mit der Funktion enableIndicator wird das 0 Bit des AS726x_LED_CONTROL Register aus 1 gesetzt, so wird die Rote Status LED des jeweiligen Sensors auf dem Sensorboard ange schaltet.

Mit der Funktion disableIndicator wird das gleiche Bit auf 0 gesetzt, also die Led ausgeschal tet.

Als Übergabeparameter wird bei beiden Funktionen der File Diskiptor einer aktiven I2C Verbindung erwartet.

6.2.13 softReset

Der Sensor liefert in einigen Situationen Messwerte außerhalb des Erwartungsbereiches. Dieses Problem kann in manchen fällen behoben werden indem ein softReset durchgeführt wird. Da bei der hier beschriebenen Implementierung keine Fehler auftreten, wird die Funktion nicht benötigt, kann aber in zukünftigen Versionen verwendet werden.

Die Funktion softReset setzt das 8 Bit des Registers CONTROL_SETUP auf 1 um einen softReset auszulösen. Das Datenblatt gibt an, das mindestens 1000 ms gewatet werden muss, bevor der softReset abgeschlossen ist und der Sensor wieder genutzt werden kann. Für beide Funktionen wird der FileDiskiptor einer aktiven I2C-Verbindung als Übertragungsparameter erwartet.

6.2.14 I2CScan

Die Funktion I2CScan wird zu Beginn des Programms aufgerufen, um festzustellen, welche Sensoren an den I2C DataBus angeschlossen sind. Die gefundenen Sensoradressen und der SensorTyp werden in das Struct sensor_list geschrieben.

Die Adressen werden außerdem im Terminal angezeigt, so dass der Benutzer überprüfen kann, ob alle erwarteten Sensoren erkannt werden. Als Übergabeparameter wird im call by reference style ein Pointer zu einem Struct vom Typ sensor_list erwartet. Da die Daten direkt in das extern deklarierte(?) Struct geschrieben werden gibt es keinen Rückgabewert. Um die angeschlossenen Sensoren zu detektieren wir zu jeder der 2^8 möglichen I2C Adressen eine Verbindung aufgebaut und ein Schreibversuch mithilfe der Funktion wiringPil2CWriteReg8 vorgenommen. Wenn die Funktion wiringPil2CWriteReg8 eine 0 zurück gibt war der Schreibversuch erfolgreich also muss ein Sensor unter dieser Adresse vorhanden sein. Das im Datenblatt nicht erwähnte Register mit der Adresse 0x05 wurde ausgewählt da es mit dem Wert 0x01 beschrieben werden kann ohne das der Sensor sein Verhalten verändert. Die Funktion getVersion wird genutzt um die Version des gefunden Sensors zu ermitteln. Wenn ein AS72651 erkannt wird zusätzlich abgefragt ob auch die Slavesensoren AS72652 und AS72653 vorhanden sind, diese Information wird nur im Terminal ausgegeben und nicht in das struct sensor_list geschrieben, da es für den Programmablauf nicht notwendig ist.

```
1 // Scans for sensors on all 128 possible addresses
2 // input pointer to array of sensor_list struct size has to ←
   be 128
3 // writes sensor address and type to array of sensor_list struct
4 void I2C_Scan(sensor_list *const s){
5     //printf("test struct address in function %i is value ←
6         %i\n", s[0].address, s[0].type );
7     printf("-----I2C Scan -----\\n");
8     uint8_t sensor_count = 0;
9     for (int address = 0; address < 128; ++address)
10    {
11        int fd = wiringPiI2CSetup(address);
12        if (fd != -1){
13            //try to write to some hopefully unused ←
               register(5) → return value: 0 indicates that ←
               someone was listening
14            //this possibly writes to the Integration Value ←
               Register so make shure to properly set it ←
```

```

        before starting a measurement
14     if (wiringPiI2CWriteReg8 (fd, 5, 1) == 0){
15         int version = getVersion(fd);
16         if (version == SENSORTYPE_AS7261){
17             printf("Device at: 0x%X is ←
18                 AS7261\n", address);
19             s[sensor_count].address = address;
20             s[sensor_count].type = SENSORTYPE_AS7261;
21             sensor_count++;
22         }
23         else if(version == SENSORTYPE_AS72651){
24             printf("Device at: 0x%X is AS72651",address);
25             s[sensor_count].address = address;
26             s[sensor_count].type = SENSORTYPE_AS72651;
27             sensor_count++;
28             if (scan_AS7262(fd)){
29                 printf(", AS72652");
30             }
31             if (scan_AS7263(fd)){
32                 printf(", AS72653");
33             }
34             printf("\n");
35         }
36     }
37     close(fd);
38 }
39 printf("-----\n");
40 }

```

6.2.15 getVersion

Die Funktion getVersion gibt den Inhalt der Registers AS726x_HW_VERSION zurück, indem der zurückgegebene Wert mit den beiden SENSORTYPE Werten verglichen wird(SENSORTYPE_AS7261 und SENSORTYPE_AS72651) kann festgestellt werden um welchen Sensor es sich handelt. Als Überabeparameter wird bei beiden Funktionen der File Diskiptor einer aktiven I2C-Verbindung erwartet.

6.2.16 scanAS7262

Als Übergabeparameter wird bei der Funktionen der FileDiskiptor einer aktiven I2C Verbindung mit einem AS72651 erwartet.

Die Funktion überprüft den Status des 4. Bit des Registers DEV_SELECT_CONTROL und gibt ihn zurück. Das Bit ist auf 1 gesetzt wenn der Slavesensor AS7262 vorhanden ist, im falle der Abwesenheit ist es 0.

Das AS7265X-Datenblatt gibt fälschlicherweise an, dass das 5. Bit geprüft werden muss.

6.2.17 scanAS7263

Als Übergabeparameter wird bei der Funktionen der FileDiskiptor einer aktiven I2C Verbindung mit einem AS72651 erwartet.

Die Funktion überprüft den Status des 5. Bit des Registers DEV_SELECT_CONTROL und gibt ihn zurück. Das Bit ist auf 1 gesetzt wenn der Slavesensor AS7263 vorhanden ist, im falle der Abwesenheit ist es 0.

6.2.18 setGain

Die Messergebnisse der Sensoren können intern verstärkt werden, was z.B. in dunklen Messumgebungen oder bei der Verwendung relativ lichtundurchlässiger Streuscheibe notwendig ist.

Außerdem kann das Speicherregister besser ausgelastet werden, um ein genauereres Messergebnis zu erhalten. Um den Verstärkungsfaktor einzustellen, wird die Funktion setGain benötigt. Die Funktion setGain beschreibt das 4 und 5 Bit des Registers CONTROL_SETUP mit einem der 4 möglichen Zustände welcher an die Funktion sie übergebenen wird.

Wert	Verstärkungsfaktor
0	1x (power-on default)
1	3.7x
2	16x
3	64x

Ist der übergebe Wert größer als 3 ist wird das Register auf den Wert 3 gesetzt.

6.2.19 setIntegrationTime

Um die Integrationszeit der Messung einzustellen wird in das Register AS726x_INT_T ein Wert (integrationValue) zwischen 0 und 255 geschrieben, die Integrationszeit errechnet sich indem dieser Wert mit dem Faktor 2.8ms multipliziert wird. Die Funktion setIntegrationTime erwartet als Übergabeparameter das integrationValue und den FileDiskiptor sowie eine aktive I2C-Verbindung zu einem Sensor.

6.2.20 disableInterrupt

Die Funktion disableInterrupt setzt das INT Bit im Register AS726x_CONTROL_SETUP auf 0 um Interrupts aus zu schalten. Da der Interrupt-Pin auf der Sensor-Platine nicht angeschlossen ist, können Messungen nur durchgeführt werden, wenn der Interrupt ausgeschaltet ist. Laut Datenblatt wird der Interrupt beim Systemstart ausgeschaltet.

6.3 influxDB Library writeToDatabase

Um die Messdaten der Sensoren in die aus ?? bekannte InfluxDB zu schreiben wurde die InfluxDB Library entwickelt, sie enthält nur eine einzige nach außen sichtbare Funktion: writeToDatabase.

6.3.1 writeToDatabase

Um die Datenbank zu beschreiben muss zuerst ein Socket geöffnet werden um sich mit der InfluxDB Server-Instanz zu verbinden. der InfluxDB Sever ist unter der Localhostaddress 127.0.0.1 am Port 8086 zu erreichen. Diese Werte können in der influxdb.h Datei angepasst werden falls in Zukunft die Notwendigkeit besteht den Server auf einem externen Gerät zu betreiben.

Die Kommunikation mit dem InfluxDB Server erfolgt über das Http basierte InfluxDB line protocol. In Zeile 68 wird der Datenbankname , Username, Passoword der Datenbank und die Größe des Body's der Anfrage in den Header teil der http anfrage geschrieben. In Zeile 64 wird der Body teil der anfrage im InfluxDB line protocol Format erzeugt, dieser enthält den Bezeichner des Messwerts (z.B. X oder Y), die I2C Adresse des jeweiligen Sensors von dem die Messdaten stammen, den Messwert selbst sowie den Zeitstempel der Messung in ms. In

Zeile 85 wird der Http request bestehend aus Header und Body an den Server übertragen. Die Antwort vom Server wird gespeichert und anschließen auf die erwartete Rückmeldung "HTTP/1.1 204 No Content" untersucht, wenn der Schreibversuch fehlschlägt wird eine Meldung im Terminal angezeigt(TODO hier sollte die Status LED sich verändern).

6.4 main

In der Mainfunktion kommen

6.4.1 changeSettings

TODO

6.4.2 printTime

TODO

6.4.3 delayMeasurementMin

TODO

6.4.4 currentTimestamp

TODO

6.5 measurement

TODO

6.5.1 settings

TODO

6.5.2 matchValueToMaxGain

TODO

6.5.3 cleanAS7261Data

TODO

6.5.4 cleanAS7265XData

TODO

6.5.5 getAS7261Measurement

TODO

6.5.6 getAS7265XMeasurement

TODO

6.5.7 saveAS7261Measurement

TODO

6.5.8 saveAS7265XMeasurement

TODO

6.5.9 saveAS7261Gain

TODO

6.5.10 autoGainMeasurementAS7261

TODO

6.5.11 saveAS7265XGain

TODO

6.5.12 autoGainMeasurementAS7265X

TODO

6.5.13 manualGainMeasurementAS7261

TODO

6.5.14 manualGainMeasurementAS7265X

7 Benutzerhandbuch

8 Messungen

9 Zusammenfassung

Literatur