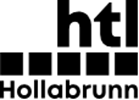
****

**HÖHERE TECHNISCHE BUNDESLEHRANSTALT**

**HOLLABRUNN**

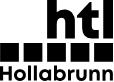
Höhere Abteilung für Elektronik – Technische Informatik

|  |  |
| --- | --- |
| Klasse/ Jahrgang:  4BHEL | Übungsbetreuer:  Dipl.-Ing. Josef Reisinger |
| Übungsnummer:  4 | Übungstitel:  Gestensensor APDS-9960 |
| Datum der Vorführung:  17.06.2020 | Gruppe:  Clemens Marx, Stefan Grubmüller |
| Datum der Abgabe:  17.06.2020 | Unterschrift: |

**Beurteilungskriterien**

|  |  |
| --- | --- |
| **Programm:** | **Punkte** |
| Programm Demonstration |  |
| Erklärung Programmfunktionalität |  |
| **Protokoll:** | **Punkte** |
| Pflichtenheft  (Beschreibung Aufgabenstellung) |  |
| Beschreibung SW Design (Flussdiagramm, Blockschaltbild,..) |  |
| Dokumentation Programmcode |  |
| Testplan (Beschreibung Testfälle) |  |
| Kommentare / Bemerkungen |  |
| **Summe Punkte** |  |

Note: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Gestensensor APDS-9960**

**Inhaltsverzeichnis**

1 Originalangabe und Unterschriften 3

2 Gestensensor APDS9960 4

2.1 Allgemeines 4

2.2 Blockdiagram 4

2.3 Pin-Konfiguration 5

2.4 Blockschaltbilder 5

2.5 Elektrische Charakteristiken 7

2.6 I2C-Bus Protokoll 7

2.6.1 Bus Zustände 7

2.6.2 Write Befehl 8

2.6.3 Read Befehl 8

2.7 Systemtakt HSE 36MHz 9

2.8 Uhr 10

2.9 Gestensensor 10

2.10 Abstandssensor 10

2.10.1 Register 11

2.10.2 Enable Register (0x80) 11

2.10.3 Proximity Interrupt Threshold Register (0x89/0x8B) 12

2.10.4 Persistence Register (0x8C) 12

2.10.5 Proximity Pulse Count Register (0x8E) 12

2.10.6 Control Register One (0x8F) 13

2.10.7 Configuration Register Two (0x90) 13

2.10.8 Status Register (0x93) 14

2.10.9 Proximity Data Register (0x9C) 14

2.10.10 Configuration Three Register (0x9F) 14

2.11 Testprogramm 15

2.11.1 Konfiguration 15

2.11.2 Externer Interrupt PA1 17

2.12 Probleme 18

2.13 Erkenntnisse 18

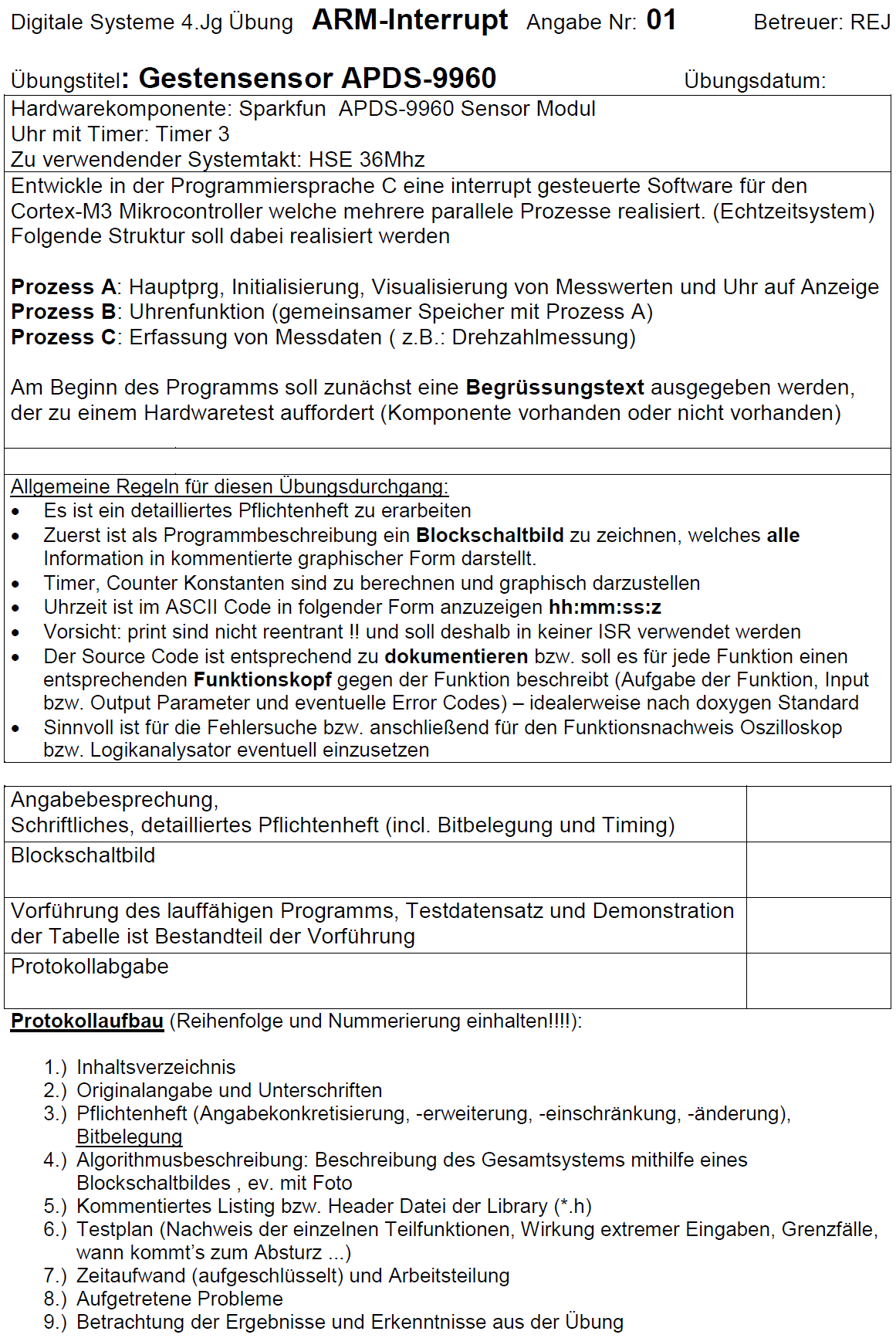
2.14 Zeitaufwand – Stefan Grubmüller 18

2.15 Zeitaufwand – Clemens Marx 18

Anhang:

* Source-Code
* Datenblatt

# Originalangabe und Unterschriften



# Gestensensor APDS9960

## Allgemeines

Das Sensor APDS9960 unterstützt Funktionen wie Gestenerkennung, Abstandserkennung, Umgebungslicht und Farben im RGBC Spektrum. Die gesamte Platine umfasst eine Fläche von rund 9,3mm2.

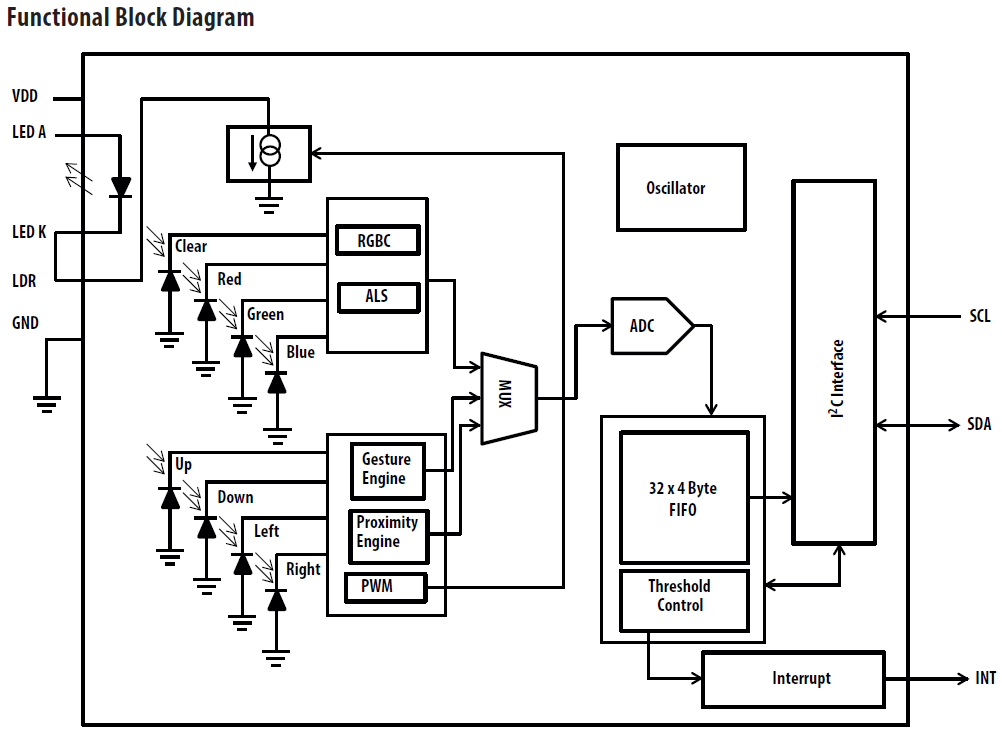
Für die Gestenerkennung gibt es vier Fotodioden, welche Infrarotlicht von einer integrierten LED, welche von Objekten, wie zum Beispiel einer Hand reflektiert werden, erkennen und in ein digitales Signal umwandeln. Erkannt werden können einfache UP-DOWN-LEFT-RIGHT Bewegungen sowie kompliziertere Bewegungen (diagonal, …). Stromverbrauch wird bei diesem Sensor durch Sleep Mode und einstellbaren Infrarot und LED Timing minimiert.

Der Abstandssensor bietet Distanzmessung (z.B.: Mobilgerät zum Ohr beim Telefonieren). Die Erkennung erfolgt mittels Interrupts, welche immer dann ausgelöst werden, wenn das Messergebnis innerhalb der unteren und oberen Grenze liegt.

Der Sensor kann auch mittels der Fotodioden für Rot, Grün, Blau und normales Licht die Intensität herausfinden. Damit kann genau das Umgebungslicht und Farben gemessen werden, was Geräten ermöglicht die Farbtemperatur zu berechnen oder das Hintergrundlicht bei Displays zu steuern.

Die Kommunikation mit dem Sensor erfolgt bei uns mittels I2C-Bus Protokoll. Als I2C-Library wurde die von Jakob Pachtrog verwendet.

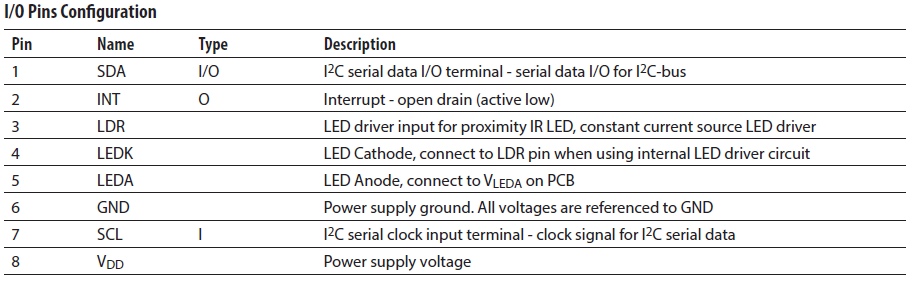
## Blockdiagram



*Blockdiagram des APDS9960*

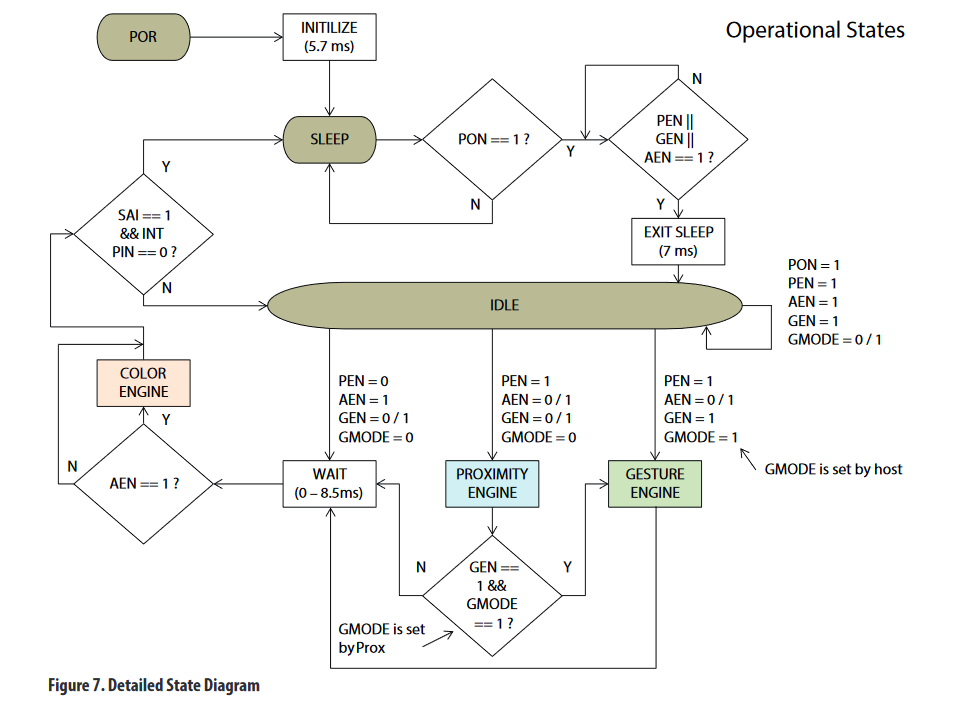
Vereinfacht ausgedrückt, werden über die Fotodioden Messwerte aufgenommen, welche über den Speicher mit FIFO Prinzip („First in, first out“ – „Erster rein, erster raus“), welcher über den I2C-Bus angesteuert werden, ausgewertet. Dementsprechend löst dieser dann über den Interrupt Pin ein Interrupt Signal aus. Dieses Interrupt Signal wird bei uns beim Cortex über einen Externen Interrupt über den Pin PA7 geregelt.

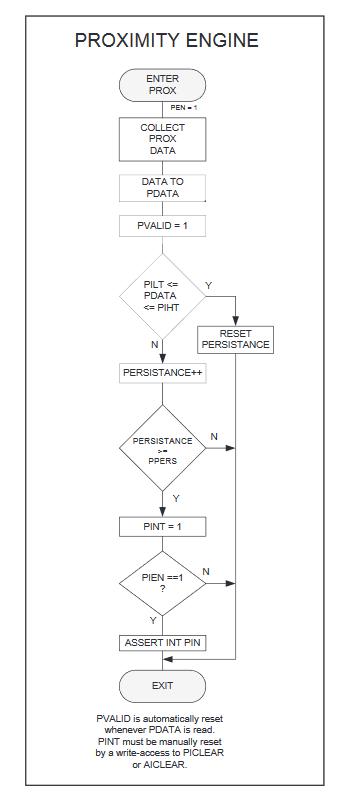
## Pin-Konfiguration



Für unserer Anwendung werden nur fünf Anschlüsse verwendet. SDA (Signal Data) und SCL (Signal Clock) sind die Ansteuerleitungen für den I2C-Bus. Über die SDA-Leitung werden die Daten gesendet und über die SCL-Leitung das Clock Signal. Die Daten auf der Datenleitung können sich bidirektional bewegen (Input und Output). Zur Versorgung verwenden wir über den VDD Pin den 3,3V Anschluss auf unserem Mikrocontroller sowie Masse über den GND Pin. Der Interrupt Pin wird verwendet, um eine Ausgabe bei Erreichen entsprechender Messwerte auszugeben und am Cortex über den Pin PA7 über einen Externen Interrupt aufzunehmen.

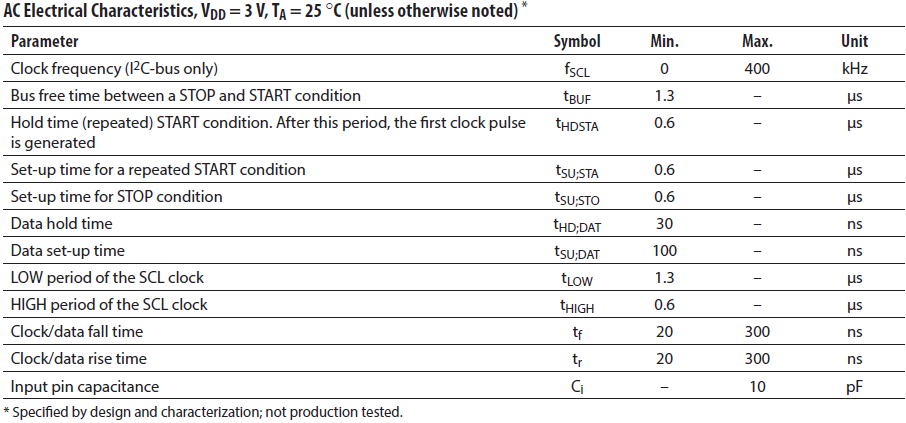
## Blockschaltbilder





Auf diesen Blockschaltbildern erkennt man die genaue Funktionsweise des Sensors, welche Bits gesetzt und geprüft werden.

## Elektrische Charakteristiken



Als Clock Frequenz wurde der Einfachheit halber 80kHz genommen. Darüber wäre der I2C-Bus nämlich nicht mehr im Standard Mode, sondern im Fast Mode (100 bis 400kHz) und es müssten mehr Konfigurationen getroffen werden (Low/High Periode von der Clock, …).

## I2C-Bus Protokoll

Der I2C Bus ist ein synchroner serieller 2-Draht Bus, der in den 80er Jahren von Philips entwickelt wurde. I2C gesprochen 'I Quadrat C' kommt von der der Abkürzung IIC und bedeutet Inter-Integrated Circuit. Er wird hauptsachlich dazu benutzt, zwischen Schaltkreisen, die sich auf einer Platine verbinden, Daten auszutauschen. Die beiden Leitungen, die den I2C Bus bilden heißen SCL und SDA. SCL steht für Signal Clock und ist die Taktleitung für den Bus. Deshalb spricht man auch von einem synchronen Bus. SDA steht für Signal Data und ist die Datenleitung. Die Datenübertragungsrate des I2C Busses beträgt 100kHz im Standard Mode, bzw. 400kHz im Fast Mode.

Der I2C Bus ist ein Multi Master/Slave Bus. Das bedeutet, es gibt mindestens einen I2C Master und ebenso mindestens einen I2C Slave. Der Master selektiert einen Slave durch seine Slave Adresse, die innerhalb eines Busses eindeutig sein muss. Eine Datenübertragung kann nur durch einen I2C Master initiiert werden. Der Slave bleibt immer passiv und lauscht nur auf die Slave Adresse und vergleicht diese mit seiner eigenen Slave Adresse. Erst wenn er seine Slave Adresse erkennt, greift der Slave auch aktiv in das Busgeschehen ein.

Aus Sicht des I2C Masters unterscheidet man zwischen Read und Write Sequenzen. Bei einer Read Sequenz liest der I2C Master Daten vom I2C Slave. Bei einer Write Sequenz sendet der I2C Master Daten zum Slave.

### Bus Zustände

Bus frei: Wenn SCL und SDA dauerhaft HIGH sind, spricht man von 'Bus free'. Ein I2C Master muss diese Bedingung immer zuerst abprüfen, bevor er den Bus belegen darf.

Start Bedingung: Die Start Bedingung kennzeichnet den Beginn einer Datenübertragung durch einen I2C Master. Der Master zieht die Datenleitung SDA von HIGH auf LOW, während die Taktleitung SCL auf HIGH bleibt.

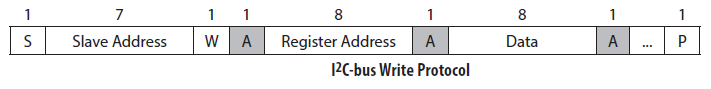
Datenbit: Ein Datenbit kann, wie in der Digitaltechnik üblich 2 Zustände einnehmen '0' oder '1'. Die Daten sind gültig während die Taktleitung SCL auf HIGH liegt. Ein LOW Pegel auf der Datenleitung SDA bedeuted '0', ein HIGH bedeutet '1'.

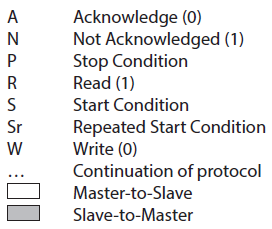
Acknowledge: Bei einer Schreib Sequenz quittiert der I2C Slave nach Erkennen seiner Slave Adresse bzw. nach jedem geschriebenen Daten Byte mit einem Acknowledge. Bei einer Lese Sequenz quittiert der I2C Slave nach Erkennen seiner Slave Adresse mit Acknowledge. Nach jedem gelesenen Daten Byte quittiert der I2C Master mit einem Acknowledge dem Slave, dass er bereit ist, weitere Daten zu empfangen. Dabei wird zu einem ebenfalls vom Slave generierten Takt Impuls die Daten Leitung auf LOW gehalten.

No Acknowledge: Bei einer Lese Sequenz sendet der I2C Master nach dem Lesen des letzten Daten Byte ein No Acknowledge. Das bedeutet, er möchte keine weiteren Daten mehr lesen. Dabei wird zu einem ebenfalls vom Slave generierten Takt Impuls die Daten Leitung auf HIGH gehalten.

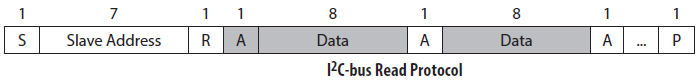
Stopp Bedingung: Die Stop Bedingung kennzeichnet das Ende einer Datenübertragung durch einen I2C Master. Der Master zieht die Datenleitung SDA von LOW auf HIGH, während die Taktleitung SCL auf HIGH bleibt.

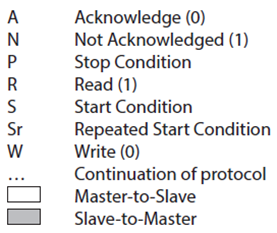
### Write Befehl



Für den Schreibe Befehl beginnt die Anfrage mit einem Start-Bit und der Adresse des entsprechenden Slaves, welcher angesprochen werden soll und einem Write Bit (0). Wenn dies funktioniert gibt es vom Slave ein Acknowledge zurück und das Register kann angesprochen werden. Nun gibt es wieder ein Acknowledge zurück und das Register ist bereit Daten zu empfangen. Wenn dies abgeschlossen ist gibt es wieder ein Acknowledge zum Master und mit dem Stopp Bit wird die Übertragung beendet.

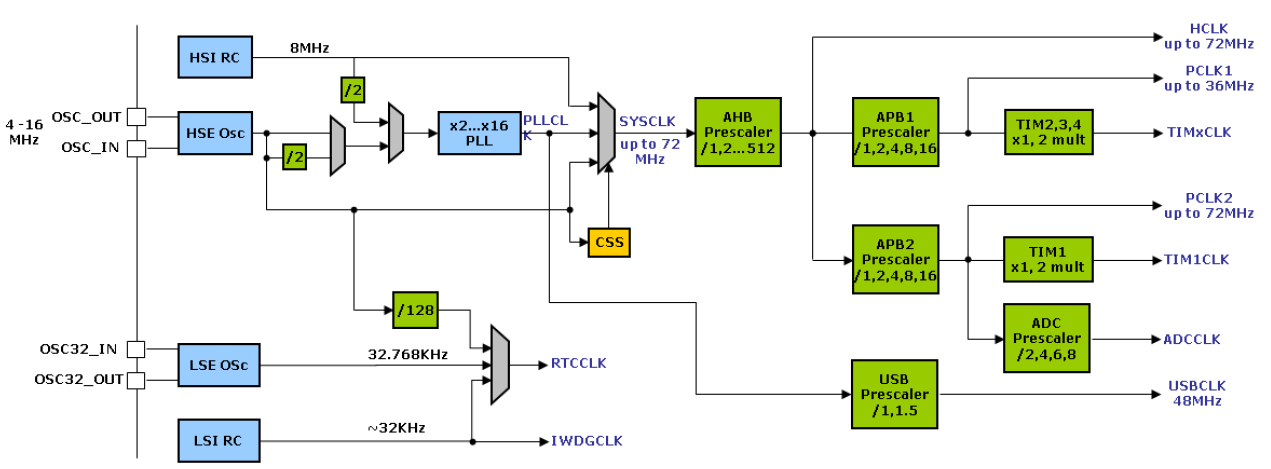
### Read Befehl





Für den Read Befehl wird ein Start Bit gesendet mit der entsprechenden Slave Adresse und dem Read Bit (1). Hat die funktioniert, wird ein Acknowledge und die Daten zurückgesendet. Der Master bestätigt immer nach einem Byte die Daten und ein Weiteres Byte wird gesendet.

## Systemtakt HSE 36MHz



*Ausschnitt aus Punkt 4.2.4 aus dem DIC 4. Klasse Skriptum von Hr. Wihsböck*

Um als Systemtakt den HSE mit 36MHz einzustellen mussten einige Vorkehrungen getroffen werden. Um nämlich den PLL und den HSE umstellen zu können muss zuerst der Systemtakt auf den HSI wechseln. Anschließend wird der PLL ausgeschalten und angepasst. Dafür wird er mittels dem PLLXTPRE Register halbiert und mit dem PLLMULL Register mit Neun multipliziert und als Systemtakt definiert. Der PLL und der HSE werden wieder eingeschalten und es wird gewartet, bis beide stabil sind.

Folgende Rechnung ergibt sich dadurch:

Bemerkung: Der genaue Code für den beschriebenen Vorgang findet man im Unterprogramm void set\_clock\_36MHz();

## Uhr

Ein Punkt der Aufgabenstellung war es, eine Uhr mit Interrupts zu programmieren, welche beginnt zu laufen, wenn das Programm gestartet wird. Die Anzeige erfolgt in dem Format hh:mm:ss:z. Daher musste ein Interrupt alle 0,1 Sekunden ausgelöst werden, um dadurch eine Millisekunden Variable zu inkrementieren. Dafür wurde der Timer 3 verwendet. Um diesen zu konfigurieren, wurde zuerst dieser aktiviert und PSC (prescaler) und ARR (auto-reload Wert) berechnet.

Berechnung für 36MHz (Tgesucht=100ms):

Berechnung für 8MHz (Tgesucht=100ms):

## Gestensensor

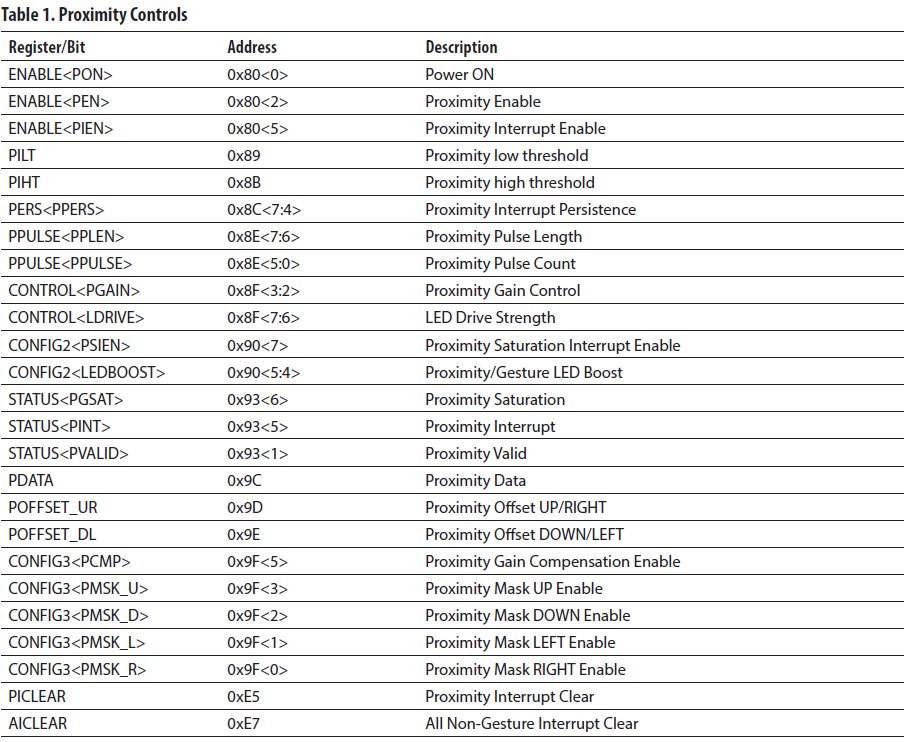
Die anfängliche Aufgabe bestand darin, von dem Sensor, der auch Gesten erkennen kann, die Daten entsprechend auszuwerten und anzugeben, welche Geste gemacht wird. Dies hat sich als zu kompliziert herausgestellt, da es noch dazu keine passende vorgefertigte Library für diesen Sensor gibt (nur für Arduino). Daher wurde anschließend die Aufgabestellung angepasst und nur der Abstandssensor verwendet.

## Abstandssensor

Die Ergebnisse von dem Sensor sind durch drei Hauptfaktoren beeinflusst. Und zwar von der Infrarot LED-Abstrahlung, von dem Infrarot Licht das empfangen wird, und Umgebungseinflüsse wie die Distanz.

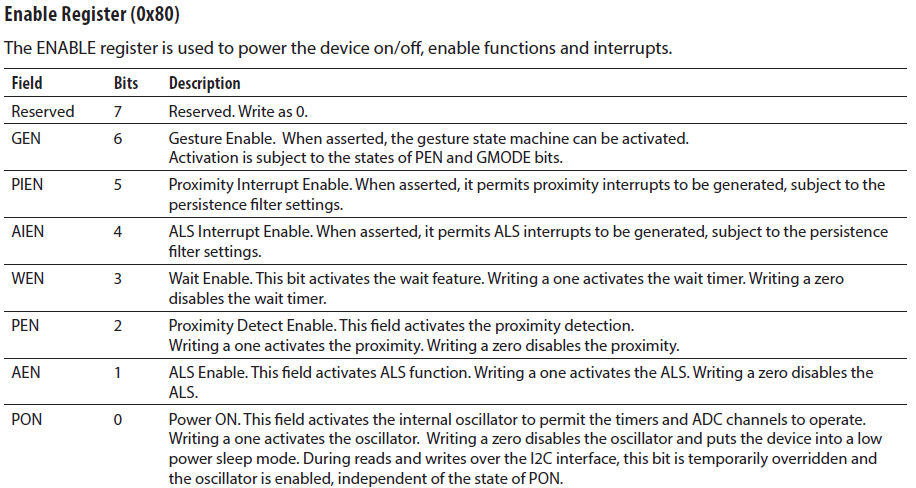
### Register

Um den Sensor zu konfigurieren wurden folgende Register entsprechend gesetzt:



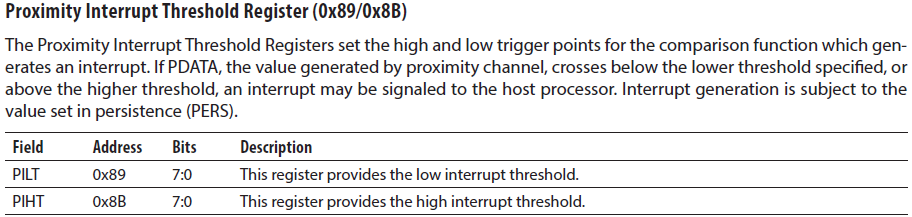
**Nachfolgend ist eine kurze Erklärung zu jedem Register, die genaue Konfiguration ist im Programmcode nachzulesen.**

### Enable Register (0x80)



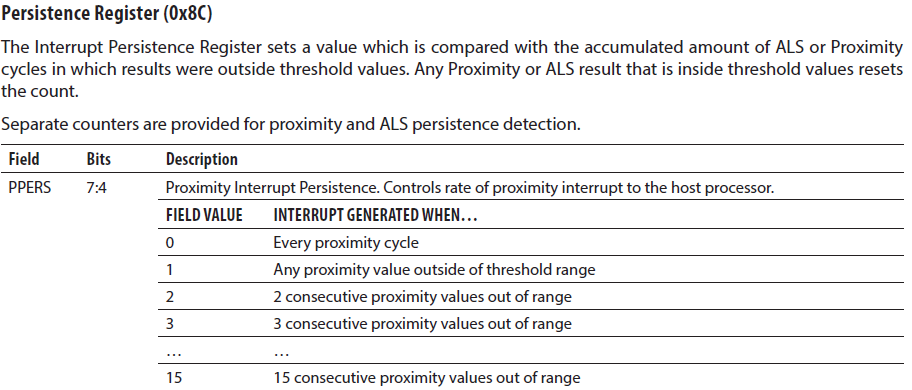
Gesetzt werden muss das Bit 5 PIEN immer dann, wenn ein Interrupt ausgelöst werden soll, also wenn ein der Sensor einen Wert innerhalb der oberen und unteren Grenze erkennt. Mit PEN wird der Sensor für den Abstand aktiviert und mit PON der ganze Sensor mit Strom versorgt.

### Proximity Interrupt Threshold Register (0x89/0x8B)



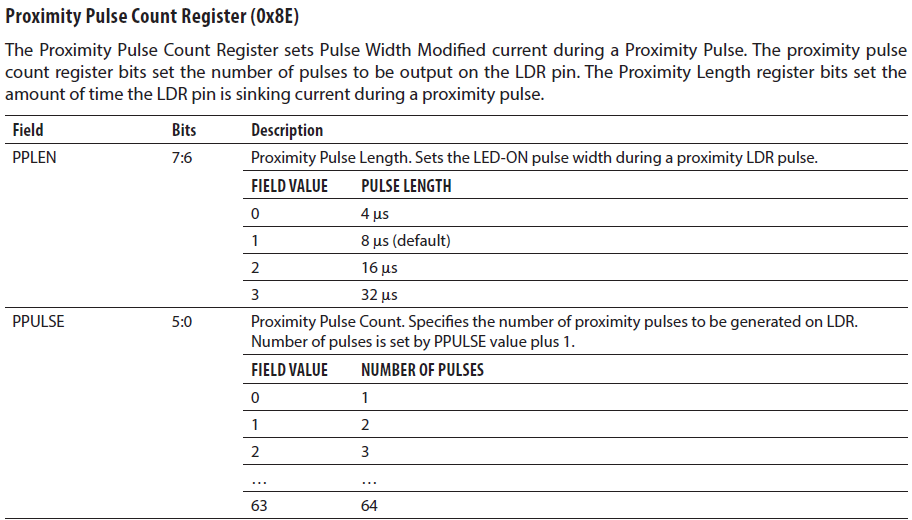
Mit diesen Registern wird der obere und untere Grenzwert zum Detektieren festgelegt.

### Persistence Register (0x8C)



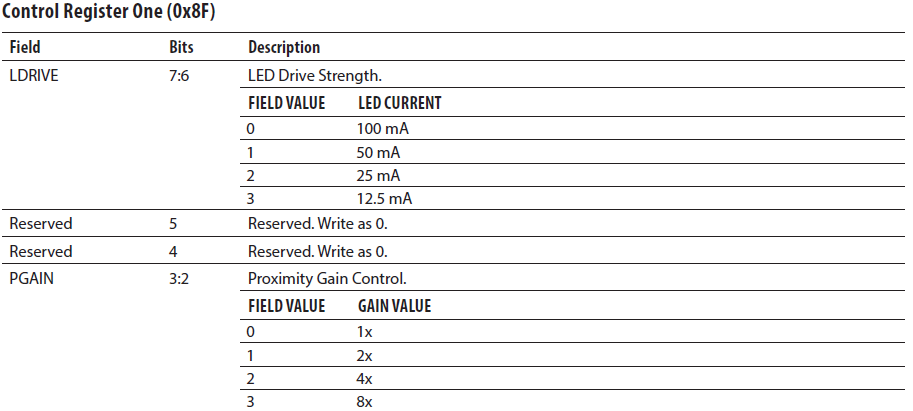
Mit diesem Register wird festgelegt, nach wie vielen Messungen der Interrupt auslösen soll.

### Proximity Pulse Count Register (0x8E)



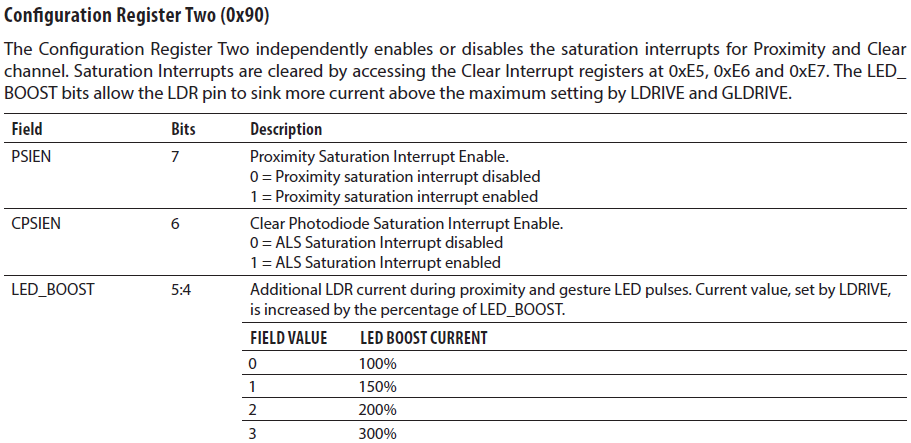
In diesem Register wird festgelegt, wie lange der Puls für den Abstandsensor sein soll und wie viele Pulse gezählt werden müssen.

### Control Register One (0x8F)



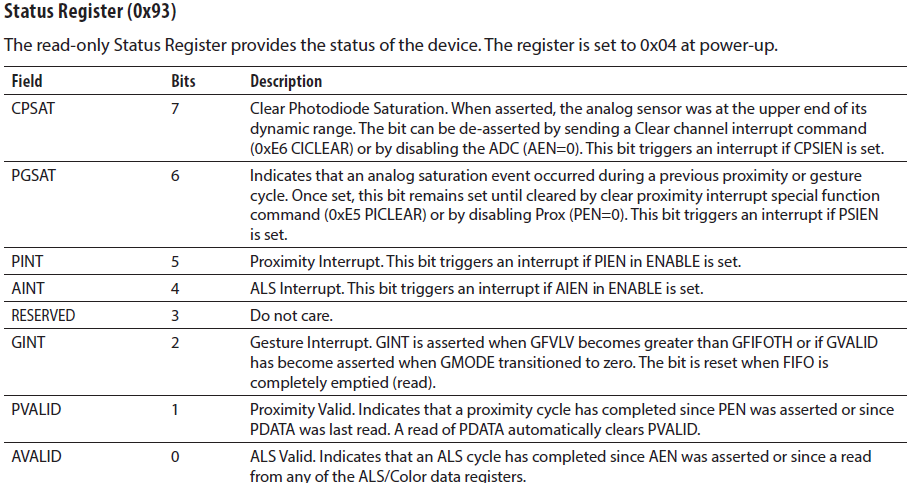
Bei diesem Register wird die stärke der LED und die Verstärkung eingestellt um die Intensität mit welcher der Sensor reagiert zu steuern.

### Configuration Register Two (0x90)



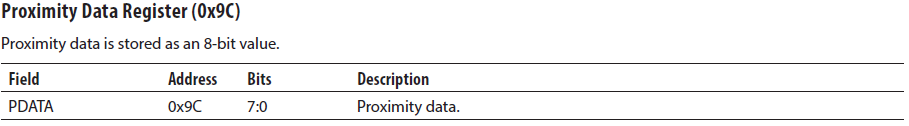
Mit diesem Register kann der Strom durch die LED nochmals verstärkt werden um Gesten die weiter weg sind zu erkennen.

### Status Register (0x93)



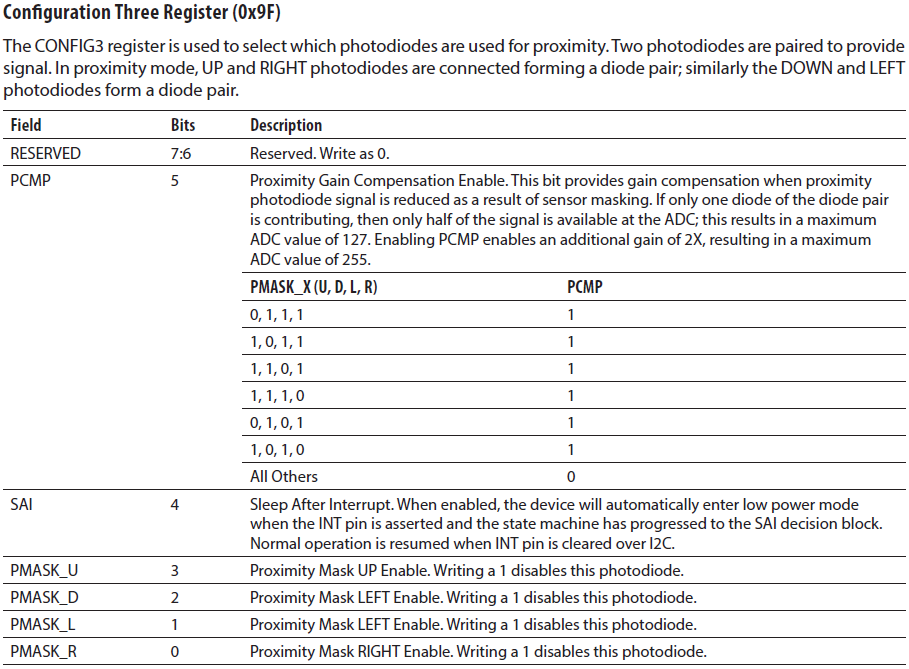
Das Status Register kann nur ausgelesen werden und stellt die Informationen zum Status des Geräts bereit.

### Proximity Data Register (0x9C)



In diesem Register werden die Daten von dem Abstandssensor gespeichert.

### Configuration Three Register (0x9F)



Bei diesem Register könnte man bestimmte Fotodioden deaktivieren.

## Testprogramm

Das Programm (main.c, proximity.c, proximity.h => Source Code siehe Anhang) liest über den Sensor die Distanz zu einem Objekt aus und gibt diese in einer nicht linearen Einheit an (siehe). Die ersten zwei Sekunden wird jedoch ein Begrüßungstext („HELLO“) ausgegeben und es wird überprüft, ob der Sensor mit dem Mikrocontroller verbunden ist oder nicht.

Die Ausgabe am LCD sieht folgenden Maßen aus:

HELLO  
DEVICE DIS/CONNECTED

Zusätzlich wird im Programm eine Range festgelegt (z.B. 16 - 172) und es wird angezeigt, ob das gemessene Objekt innerhalb dieser Range liegt. Außerdem startet nach dem Begrüßungstext ein Timer, welcher angibt, wie lange das Programm schon läuft. Dieser wird nach jedem Reset neu gestartet. Der Text am LCD sieht folgender Maßen aus:

Ein Bild, das Elektronik, Schaltkreis enthält.

Automatisch generierte Beschreibung

Hierbei liegt das gemessene Objekt (in diesem Fall einen Ordner vor dem Sensor) in einer Entfernung von 91 Punkten und liegt somit innerhalb der festgelegten Range.

### Konfiguration

Wie bereits erwähnt müssen am Anfang des Programms die verschieden Register wie folgt konfiguriert werden:

**Enable Register** (Regaddr = 0x80): 00000101b (Data = 0x05)

Bit 7 = 0: is being reserved as 0

Bit 6 = 0: gesture enable (GEN)

Bit 5 = 0: proximity interrupt enable (PIEN)

Bit 4 = 0: ambient light sense (ALS) interrupt enable (AIEN)

Bit 3 = 0: wait enable (WEN) actives wait feature

Bit 2 = 1: proximity detect enable (PEN)

Bit 1 = 0: ALS enable (AEN)

Bit 0 = 1: Power ON (PON)

**Persistance Register** (Regaddr = 0x8C): 00000000b (Data = 0x0)

Bit 7 : 4 = 0: Controls rate of proximity interrupt to host process

Bit 3 : 0 = 0: Controls rate of clear channel interrupt to host process

**Proximity Interrupt Threshold Register Low** (Regaddr = 0x89): 00010000b (Data = 0x10):

Bit 7 : 0 = adjustable

**Proximity Interrupt Threshold Register High** (Regaddr =0x8B): 10101100b (Data = 0xAC)

Bit 7 : 0 = adjustable

**Proximity Pulse Register** (Regaddr = 0x8E): 01111111b (Data = 0x7F)

Bit 7 : 6 = 01

Bit 5 : 0 = 1: 111111

**Control Register One** (Regaddr = 0x8F): 00001000b (Data = 0x08)

Bit 7 : 6 = 10

Bit 5 : 4 = reserved as 0

Bit 3 : 2 = adjustable

Bit 1 : 0 = 0

**Control Register Two** (Regaddr = 0x90): 10100000b (Data = 0x80)

Bit 7 = 1

Bit 6 = 0

Bit 5 : 4 = adjustable

Bit 3 : 0 = reserved as 0

**Status Register** (Regaddr = 0x93): 01100010b (Data = 0x62)

Bit 7 = 0 Clear Photodiode Saturation (CPSAT)

Bit 6 = 1 Indicates that an analog saturation event occurred (PGSAT)

Bit 5 = 1 Proximity Interrupt. This bit triggers an interrupt if PIEN in ENABLE is set (PINT)

Bit 4 = 0 ALS Interrupt (AINT)

Bit 3 = 0 reserved as 0

Bit 2 = 0 Gesture Interrupt (GINT)

Bit 1 = 1 Proximity Valid (PVALID)

Bit 0 = 0 ALS Valid (AVALID)

**Proximity Offset UP / RIGHT Register** (Regaddr = 0x9D): 00000000b (Data = 0x00)

Bit 7 : 0 = 0x0

**Proximity Offset DOWN / LEFT Register** (Regaddr = 0x9E): 00000000b (Data = 0x00)

Bit 7 : 0 = 0x0

**Configuration Register Three** (Regaddr = 0x9F): 00000000b (Data = 0x00)

Bit 7 : 6 = reserved as 0

Bit 5 = 0 use all diodes

Bit 4 = 0 sleep after interrupt

Bit 3 = 0 Proximity Mask UP Enable

Bit 2 = 0 Proximity Mask LEFT Enable

Bit 1 = 0 Proximity Mask LEFT Enable

Bit 0 = 0 Proximity Mask RIGHT Enable

**Proximity Interrupt Clear** (Regaddr = 0xE5): 00000000b (Data = 0x00)

Bit 7 : 0 = 0x0

**Clear All Non-Gesture Interrupts** (Regaddr = 0xE7): 00000000b (Data = 0x00)

Ein Bild, das Monitor, Screenshot, Bildschirm, drinnen enthält.

Automatisch generierte BeschreibungBit 7 : 0 = 0x0

Ein Bild, das Monitor, Elektronik, Bildschirm, drinnen enthält.

Automatisch generierte Beschreibung

Ein Bild, das Monitor, Bildschirm, Screenshot, drinnen enthält.

Automatisch generierte Beschreibung

Auf diesen Bildern sieht man die Konfiguration der verschieden Register im Logik Analysator.

### Externer Interrupt PA1

Der INT Pin des Sensors ist mit unserer Portleitung PA1 direkt verbunden, welcher als externer Interrupt Eingang konfiguriert und festgelegt ist. Jedes Mal, wenn das gemessene Objekt innerhalb der Range liegt, führt der Sensor ein Interrupt aus, indem das PIEN bit des ENABLE REGISTERs gesetzt wird. Unser Programm erkennt diesen Interrupt als steigende Flanke und gibt den oben genannten Text am LCD aus.

Ein Bild, das Elektronik, Computer, Monitor, sitzend enthält.

Automatisch generierte Beschreibung

Auf diesem Bild sieht man den Pin PA1 auf dem Logikanalysator, während der Sensor einen Impuls auslöst, da ein Objekt in die vorgegebene Range gelangt ist.

## Probleme

* Fehler beim setzten der Interrupts
* Fehler beim Konfigurieren der 36MHz, da man HSE und PLL ausschalten muss
* Fehler mit der Library von Jakob Pachtrog, beim Einstellen des Systemtakts auf 36MHz
* Probleme beim Verstehen des Datenblattes
* Probleme bei schöner Ausgabe auf dem LC Display

## Erkenntnisse

* Verwendung eines Logikanalysators
* Verwendung eines Bussystems und das bitweise senden in bestimmten zeitlichen Abständen
* Passende Informationsfindung über Datenblätter und dergleichen
* Verwenden einer vorgefertigten Library
* Funktionsweise des I2C-Bus Protokolls
* Funktionsweise und Verwendung von Interrupts
* Funktionsweise des Timers und Berechnungen von PSC und ARR

## Zeitaufwand – Stefan Grubmüller

|  |  |
| --- | --- |
| **Tätigkeit** | **Aufwand** |
| Erstellung des Pflichtenhefts | 1h |
| Programmcodierung | 26h |
| Testen der Software | 4h |
| Dokumentation (Protokoll) | 2h |
| **Gesamt:** | **33h** |

## Zeitaufwand – Clemens Marx

|  |  |
| --- | --- |
| **Tätigkeit** | **Aufwand** |
| Erstellung des Pflichtenhefts | 1h |
| Programmcodierung | 26h |
| Testen der Software | 2h |
| Dokumentation (Protokoll) | 4h |
| **Gesamt:** | **33h** |