htl Hollabrunn

# HÖHERE TECHNISCHE BUNDESLEHRANSTALT HOLLABRUNN

Höhere Abteilung für Elektronik – Technische Informatik

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

**Beurteilungskriterien**

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

Note: _____

# Gestensensor APDS-9960

htl Hollabrunn

## Inhaltsverzeichnis



Anhang:

- Source-Code
- Datenblatt

# 1 Originalangabe und Unterschriften

Digitale Systeme 4.Jg Übung **ARM-Interrupt** Angabe Nr: **01** Betreuer: REJ

Übungstitel**: Gestensensor APDS-9960** Übungsdatum:

| |
|---|
| Hardwarekomponente: Sparkfun APDS-9960 Sensor Modul<br>Uhr mit Timer: Timer 3<br>Zu verwendender Systemtakt: HSE 36Mhz |
| Entwickle in der Programmiersprache C eine interrupt gesteuerte Software für den Cortex-M3 Mikrocontroller welche mehrere parallele Prozesse realisiert. (Echtzeitsystem)<br>Folgende Struktur soll dabei realisiert werden<br><br>**Prozess A**: Hauptprg, Initialisierung, Visualisierung von Messwerten und Uhr auf Anzeige<br>**Prozess B**: Uhrenfunktion (gemeinsamer Speicher mit Prozess A)<br>**Prozess C**: Erfassung von Messdaten ( z.B.: Drehzahlmessung)<br><br>Am Beginn des Programms soll zunächst eine **Begrüssungstext** ausgegeben werden, der zu einem Hardwaretest auffordert (Komponente vorhanden oder nicht vorhanden) |
| |
| Allgemeine Regeln für diesen Übungsdurchgang:<br>• Es ist ein detailliertes Pflichtenheft zu erarbeiten<br>• Zuerst ist als Programmbeschreibung ein **Blockschaltbild** zu zeichnen, welches **alle** Information in kommentierte graphischer Form darstellt.<br>• Timer, Counter Konstanten sind zu berechnen und graphisch darzustellen<br>• Uhrzeit ist im ASCII Code in folgender Form anzuzeigen **hh:mm:ss:z**<br>• Vorsicht: print sind nicht reentrant !! und soll deshalb in keiner ISR verwendet werden<br>• Der Source Code ist entsprechend zu **dokumentieren** bzw. soll es für jede Funktion einen entsprechenden **Funktionskopf** gegen der Funktion beschreibt (Aufgabe der Funktion, Input bzw. Output Parameter und eventuelle Error Codes) – idealerweise nach doxygen Standard<br>• Sinnvoll ist für die Fehlersuche bzw. anschließend für den Funktionsnachweis Oszilloskop bzw. Logikanalysator eventuell einzusetzen |

| | |
|---|---|
| Angabebesprechung,<br>Schriftliches, detailliertes Pflichtenheft (incl. Bitbelegung und Timing) | |
| Blockschaltbild | |
| Vorführung des lauffähigen Programms, Testdatensatz und Demonstration der Tabelle ist Bestandteil der Vorführung | |
| Protokollabgabe | |

**Protokollaufbau** (Reihenfolge und Nummerierung einhalten!!!!):

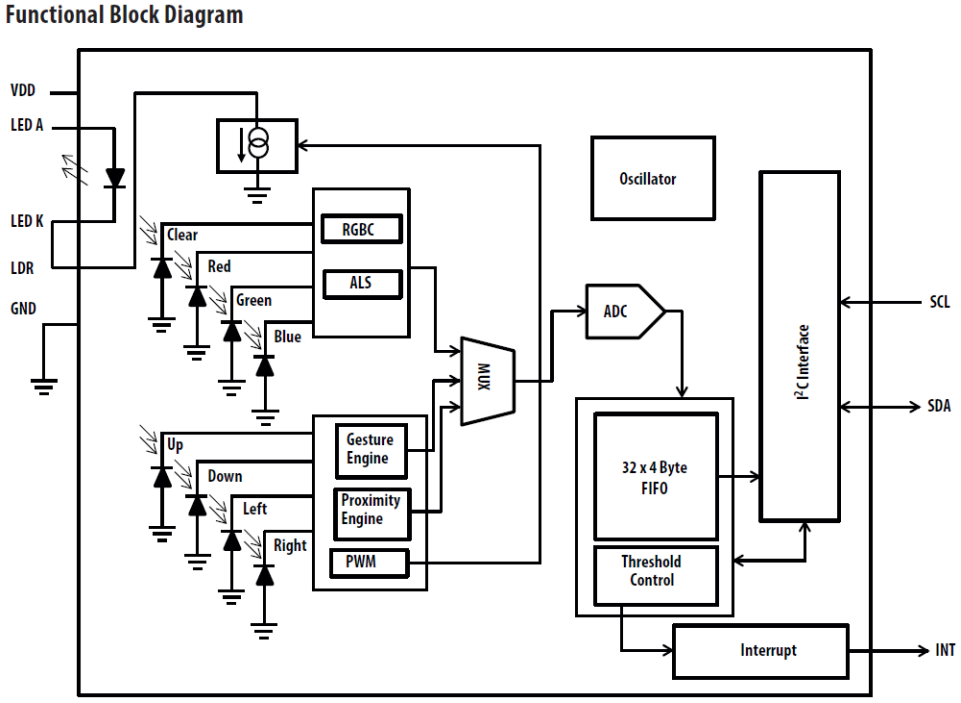
1.) Inhaltsverzeichnis
2.) Originalangabe und Unterschriften
3.) Pflichtenheft (Angabekonkretisierung, -erweiterung, -einschränkung, -änderung), Bitbelegung
4.) Algorithmusbeschreibung: Beschreibung des Gesamtsystems mithilfe eines Blockschaltbildes , ev. mit Foto
5.) Kommentiertes Listing bzw. Header Datei der Library (*.h)
6.) Testplan (Nachweis der einzelnen Teilfunktionen, Wirkung extremer Eingaben, Grenzfälle, wann kommt's zum Absturz ...)
7.) Zeitaufwand (aufgeschlüsselt) und Arbeitsteilung
8.) Aufgetretene Probleme
9.) Betrachtung der Ergebnisse und Erkenntnisse aus der Übung

# 2 Gestensensor APDS9960

## 2.1 Allgemeines


Das Sensor APDS9960 unterstützt Funktionen wie Gestenerkennung, Abstandserkennung, Umgebungslicht und Farben im RGBC Spektrum. Die gesamte Platine umfasst eine Fläche von rund 9,3mm$^2$.
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 I$^2$C-Bus Protokoll. Als I$^2$C-Library wurde die von Jakob Pachtrog verwendet.

## 2.2 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 I$^2$C-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.

## 2.3 Pin-Konfiguration

**I/O Pins Configuration**

| Pin | Name | Type | Description |
|---|---|---|---|
| 1 | SDA | I/O | I²C serial data I/O terminal - serial data I/O for I²C-bus |
| 2 | INT | O | Interrupt - open drain (active low) |
| 3 | LDR | | LED driver input for proximity IR LED, constant current source LED driver |
| 4 | LEDK | | LED Cathode, connect to LDR pin when using internal LED driver circuit |
| 5 | LEDA | | LED Anode, connect to $V_{LEDA}$ on PCB |
| 6 | GND | | Power supply ground. All voltages are referenced to GND |
| 7 | SCL | I | I²C serial clock input terminal - clock signal for I²C serial data |
| 8 | $V_{DD}$ | | Power supply voltage |

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 $V_{DD}$ 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.

## 2.4 Blockschaltbilder

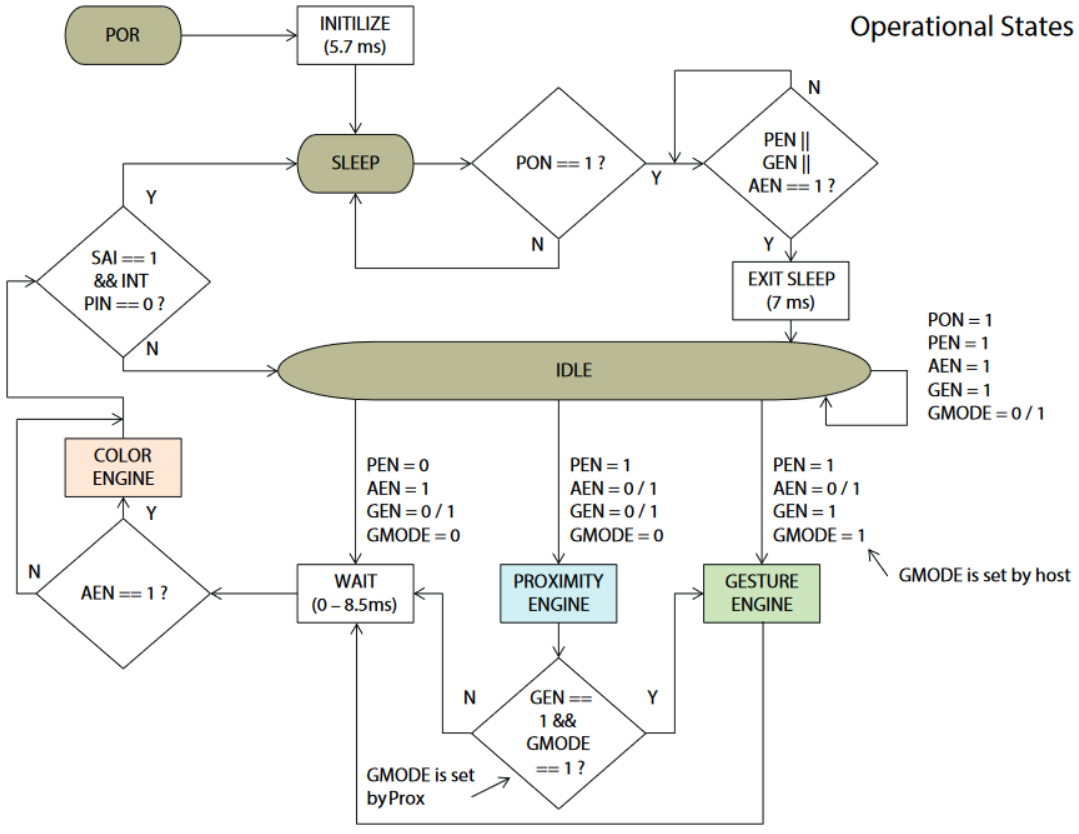
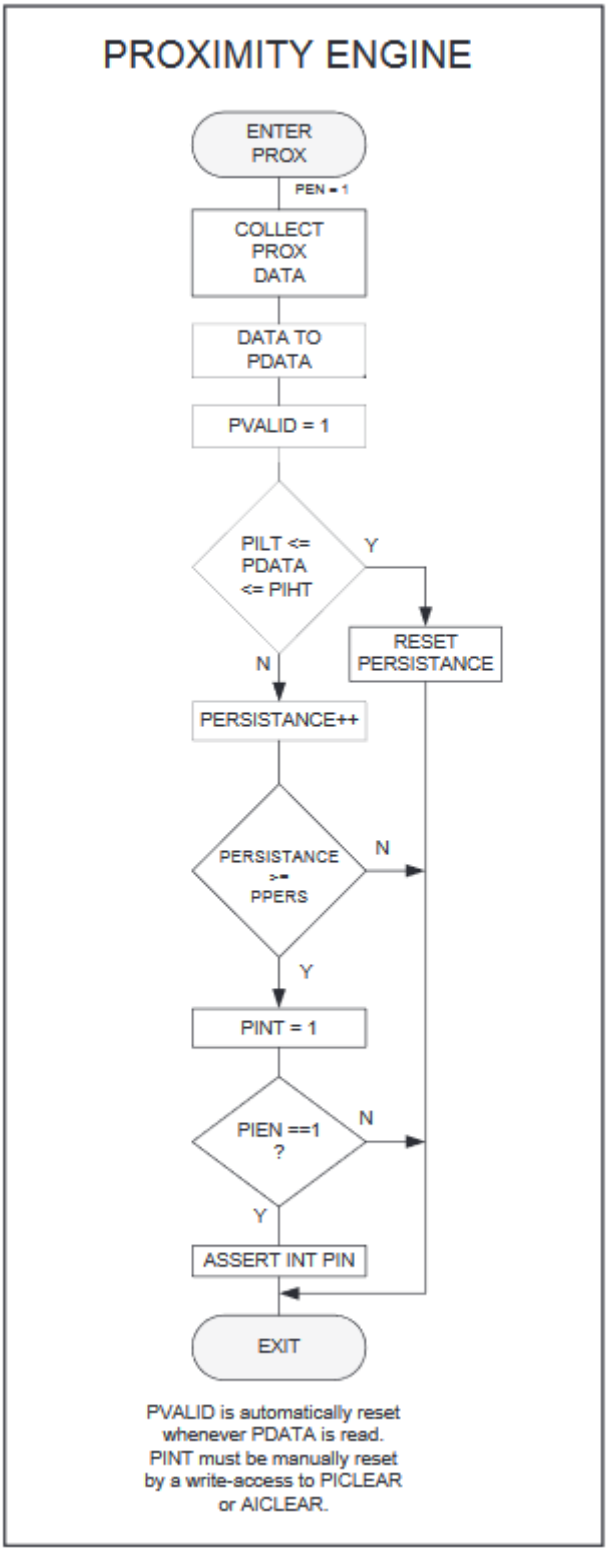



**Figure 7. Detailed State Diagram**




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

## 2.5 Elektrische Charakteristiken

**AC Electrical Characteristics, $V_{DD} = 3$ V, $T_A = 25$ °C (unless otherwise noted) ***

| Parameter | Symbol | Min. | Max. | Unit |
|---|---|---|---|---|
| Clock frequency (I²C-bus only) | $f_{SCL}$ | 0 | 400 | kHz |
| Bus free time between a STOP and START condition | $t_{BUF}$ | 1.3 | – | µs |
| Hold time (repeated) START condition. After this period, the first clock pulse is generated | $t_{HDSTA}$ | 0.6 | – | µs |
| Set-up time for a repeated START condition | $t_{SU;STA}$ | 0.6 | – | µs |
| Set-up time for STOP condition | $t_{SU;STO}$ | 0.6 | – | µs |
| Data hold time | $t_{HD;DAT}$ | 30 | – | ns |
| Data set-up time | $t_{SU;DAT}$ | 100 | – | ns |
| LOW period of the SCL clock | $t_{LOW}$ | 1.3 | – | µs |
| HIGH period of the SCL clock | $t_{HIGH}$ | 0.6 | – | µs |
| Clock/data fall time | $t_f$ | 20 | 300 | ns |
| Clock/data rise time | $t_r$ | 20 | 300 | ns |
| Input pin capacitance | $C_i$ | – | 10 | pF |

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, …).

## 2.6 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.

### 2.6.1 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'.

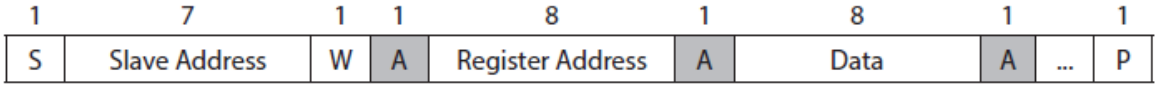
Acknowledge: Bei einer Schreib Sequenz quittiert der I²C Slave nach Erkennen seiner Slave Adresse bzw. nach jedem geschriebenen Daten Byte mit einem Acknowledge. Bei einer Lese

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.

### 2.6.2 Write Befehl

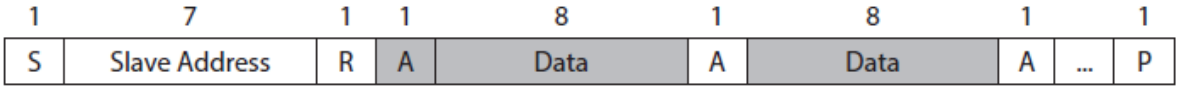



I²C-bus Write Protocol

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.

| | |
|---|---|
| A | Acknowledge (0) |
| N | Not Acknowledged (1) |
| P | Stop Condition |
| R | Read (1) |
| S | Start Condition |
| Sr | Repeated Start Condition |
| W | Write (0) |
| … | Continuation of protocol |
| ☐ | Master-to-Slave |
| ▒ | Slave-to-Master |

### 2.6.3 Read Befehl

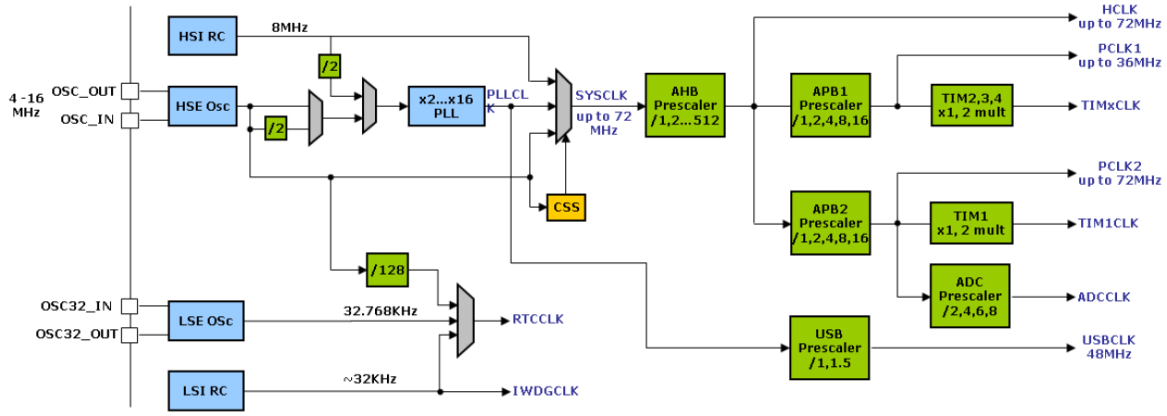



I²C-bus Read Protocol

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.

| | |
|---|---|
| A | Acknowledge (0) |
| N | Not Acknowledged (1) |
| P | Stop Condition |
| R | Read (1) |
| S | Start Condition |
| Sr | Repeated Start Condition |
| W | Write (0) |
| … | Continuation of protocol |
| ☐ | Master-to-Slave |
| ▒ | Slave-to-Master |

## 2.7 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:

$$\frac{8MHz\ (HSE)}{2\ (PLLXTPRE)} * 9\ (PLLMULL9) = 36MHz$$

Bemerkung: Der genaue Code für den beschriebenen Vorgang findet man im Unterprogramm `void set_clock_36MHz();`

## 2.8 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 ($T_{gesucht}$=100ms):

$$T_{CK_{INT}} = \frac{1}{f_{CK\_INT}} = \frac{1}{36MHz} = 27{,}78ns$$

$$ARR = \frac{T_{gesucht}}{T_{TCK\_INT}} = \frac{100ms}{27{,}78ns} = \sim 3\,600\,000 => 0xFFFF$$

$$PSC = \frac{T_{gesucht}}{T_{CK\_INT} * 65535} - 1 = \frac{100ms}{27{,}78ns * 65535} - 1 = 53{,}93 => 54$$

Berechnung für 8MHz ($T_{gesucht}$=100ms):

$$T_{CK_{INT}} = \frac{1}{f_{CK\_INT}} = \frac{1}{8MHz} = 125ns$$

$$ARR = \frac{T_{gesucht}}{T_{TCK\_INT}} = \frac{100ms}{125ns} = \sim 800\,000 => 0xFFFF$$

$$PSC = \frac{T_{gesucht}}{T_{CK\_INT} * 65535} - 1 = \frac{100ms}{125ns * 65535} - 1 = \sim 11$$

## 2.9 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.

## 2.10 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.

### 2.10.1 Register

Um den Sensor zu konfigurieren wurden folgende Register entsprechend gesetzt:

**Table 1. Proximity Controls**

| Register/Bit | Address | Description |
|---|---|---|
| ENABLE<PON> | 0x80<0> | Power ON |
| ENABLE<PEN> | 0x80<2> | Proximity Enable |
| ENABLE<PIEN> | 0x80<5> | Proximity Interrupt Enable |
| PILT | 0x89 | Proximity low threshold |
| PIHT | 0x8B | Proximity high threshold |
| PERS<PPERS> | 0x8C<7:4> | Proximity Interrupt Persistence |
| PPULSE<PPLEN> | 0x8E<7:6> | Proximity Pulse Length |
| PPULSE<PPULSE> | 0x8E<5:0> | Proximity Pulse Count |
| CONTROL<PGAIN> | 0x8F<3:2> | Proximity Gain Control |
| CONTROL<LDRIVE> | 0x8F<7:6> | LED Drive Strength |
| CONFIG2<PSIEN> | 0x90<7> | Proximity Saturation Interrupt Enable |
| CONFIG2<LEDBOOST> | 0x90<5:4> | Proximity/Gesture LED Boost |
| STATUS<PGSAT> | 0x93<6> | Proximity Saturation |
| STATUS<PINT> | 0x93<5> | Proximity Interrupt |
| STATUS<PVALID> | 0x93<1> | Proximity Valid |
| PDATA | 0x9C | Proximity Data |
| POFFSET_UR | 0x9D | Proximity Offset UP/RIGHT |
| POFFSET_DL | 0x9E | Proximity Offset DOWN/LEFT |
| CONFIG3<PCMP> | 0x9F<5> | Proximity Gain Compensation Enable |
| CONFIG3<PMSK_U> | 0x9F<3> | Proximity Mask UP Enable |
| CONFIG3<PMSK_D> | 0x9F<2> | Proximity Mask DOWN Enable |
| CONFIG3<PMSK_L> | 0x9F<1> | Proximity Mask LEFT Enable |
| CONFIG3<PMSK_R> | 0x9F<0> | Proximity Mask RIGHT Enable |
| PICLEAR | 0xE5 | Proximity Interrupt Clear |
| AICLEAR | 0xE7 | All Non-Gesture Interrupt Clear |

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

### 2.10.2 Enable Register (0x80)

**Enable Register (0x80)**

The ENABLE register is used to power the device on/off, enable functions and interrupts.

| Field | Bits | Description |
|---|---|---|
| Reserved | 7 | Reserved. Write as 0. |
| GEN | 6 | Gesture Enable. When asserted, the gesture state machine can be activated. Activation is subject to the states of PEN and GMODE bits. |
| PIEN | 5 | Proximity Interrupt Enable. When asserted, it permits proximity interrupts to be generated, subject to the persistence filter settings. |
| AIEN | 4 | ALS Interrupt Enable. When asserted, it permits ALS interrupts to be generated, subject to the persistence filter settings. |
| WEN | 3 | Wait Enable. This bit activates the wait feature. Writing a one activates the wait timer. Writing a zero disables the wait timer. |
| PEN | 2 | Proximity Detect Enable. This field activates the proximity detection. Writing a one activates the proximity. Writing a zero disables the proximity. |
| AEN | 1 | ALS Enable. This field activates ALS function. Writing a one activates the ALS. Writing a zero disables the ALS. |
| PON | 0 | Power ON. This field activates the internal oscillator to permit the timers and ADC channels to operate. Writing a one activates the oscillator. Writing a zero disables the oscillator and puts the device into a low power sleep mode. During reads and writes over the I2C interface, this bit is temporarily overridden and the oscillator is enabled, independent of the state of PON. |

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.

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

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

The Proximity Interrupt Threshold Registers set the high and low trigger points for the comparison function which generates an interrupt. If PDATA, the value generated by proximity channel, crosses below the lower threshold specified, or above the higher threshold, an interrupt may be signaled to the host processor. Interrupt generation is subject to the value set in persistence (PERS).

| Field | Address | Bits | Description |
|---|---|---|---|
| PILT | 0x89 | 7:0 | This register provides the low interrupt threshold. |
| PIHT | 0x8B | 7:0 | This register provides the high interrupt threshold. |

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

## 2.10.4 Persistence Register (0x8C)

**Persistence Register (0x8C)**

The Interrupt Persistence Register sets a value which is compared with the accumulated amount of ALS or Proximity cycles in which results were outside threshold values. Any Proximity or ALS result that is inside threshold values resets the count.

Separate counters are provided for proximity and ALS persistence detection.

| Field | Bits | Description | |
|---|---|---|---|
| PPERS | 7:4 | Proximity Interrupt Persistence. Controls rate of proximity interrupt to the host processor. | |
| | | **FIELD VALUE** | **INTERRUPT GENERATED WHEN…** |
| | | 0 | Every proximity cycle |
| | | 1 | Any proximity value outside of threshold range |
| | | 2 | 2 consecutive proximity values out of range |
| | | 3 | 3 consecutive proximity values out of range |
| | | … | … |
| | | 15 | 15 consecutive proximity values out of range |

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

## 2.10.5 Proximity Pulse Count Register (0x8E)

**Proximity Pulse Count Register (0x8E)**

The Proximity Pulse Count Register sets Pulse Width Modified current during a Proximity Pulse. The proximity pulse count register bits set the number of pulses to be output on the LDR pin. The Proximity Length register bits set the amount of time the LDR pin is sinking current during a proximity pulse.

| Field | Bits | Description | |
|---|---|---|---|
| PPLEN | 7:6 | Proximity Pulse Length. Sets the LED-ON pulse width during a proximity LDR pulse. | |
| | | **FIELD VALUE** | **PULSE LENGTH** |
| | | 0 | 4 µs |
| | | 1 | 8 µs (default) |
| | | 2 | 16 µs |
| | | 3 | 32 µs |
| PPULSE | 5:0 | Proximity Pulse Count. Specifies the number of proximity pulses to be generated on LDR. Number of pulses is set by PPULSE value plus 1. | |
| | | **FIELD VALUE** | **NUMBER OF PULSES** |
| | | 0 | 1 |
| | | 1 | 2 |
| | | 2 | 3 |
| | | … | … |
| | | 63 | 64 |

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

## 2.10.6 Control Register One (0x8F)

**Control Register One (0x8F)**

| Field | Bits | Description | |
|---|---|---|---|
| LDRIVE | 7:6 | LED Drive Strength. | |
| | | **FIELD VALUE** | **LED CURRENT** |
| | | 0 | 100 mA |
| | | 1 | 50 mA |
| | | 2 | 25 mA |
| | | 3 | 12.5 mA |
| Reserved | 5 | Reserved. Write as 0. | |
| Reserved | 4 | Reserved. Write as 0. | |
| PGAIN | 3:2 | Proximity Gain Control. | |
| | | **FIELD VALUE** | **GAIN VALUE** |
| | | 0 | 1x |
| | | 1 | 2x |
| | | 2 | 4x |
| | | 3 | 8x |

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

## 2.10.7 Configuration Register Two (0x90)

**Configuration Register Two (0x90)**

The Configuration Register Two independently enables or disables the saturation interrupts for Proximity and Clear channel. Saturation Interrupts are cleared by accessing the Clear Interrupt registers at 0xE5, 0xE6 and 0xE7. The LED_BOOST bits allow the LDR pin to sink more current above the maximum setting by LDRIVE and GLDRIVE.

| Field | Bits | Description | |
|---|---|---|---|
| PSIEN | 7 | Proximity Saturation Interrupt Enable.<br>0 = Proximity saturation interrupt disabled<br>1 = Proximity saturation interrupt enabled | |
| CPSIEN | 6 | Clear Photodiode Saturation Interrupt Enable.<br>0 = ALS Saturation Interrupt disabled<br>1 = ALS Saturation Interrupt enabled | |
| LED_BOOST | 5:4 | Additional LDR current during proximity and gesture LED pulses. Current value, set by LDRIVE, is increased by the percentage of LED_BOOST. | |
| | | **FIELD VALUE** | **LED BOOST CURRENT** |
| | | 0 | 100% |
| | | 1 | 150% |
| | | 2 | 200% |
| | | 3 | 300% |

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

### 2.10.8 Status Register (0x93)

**Status Register (0x93)**

The read-only Status Register provides the status of the device. The register is set to 0x04 at power-up.

| Field | Bits | Description |
|---|---|---|
| CPSAT | 7 | Clear Photodiode Saturation. When asserted, the analog sensor was at the upper end of its dynamic range. The bit can be de-asserted by sending a Clear channel interrupt command (0xE6 CICLEAR) or by disabling the ADC (AEN=0). This bit triggers an interrupt if CPSIEN is set. |
| PGSAT | 6 | Indicates that an analog saturation event occurred during a previous proximity or gesture cycle. Once set, this bit remains set until cleared by clear proximity interrupt special function command (0xE5 PICLEAR) or by disabling Prox (PEN=0). This bit triggers an interrupt if PSIEN is set. |
| PINT | 5 | Proximity Interrupt. This bit triggers an interrupt if PIEN in ENABLE is set. |
| AINT | 4 | ALS Interrupt. This bit triggers an interrupt if AIEN in ENABLE is set. |
| RESERVED | 3 | Do not care. |
| GINT | 2 | Gesture Interrupt. GINT is asserted when GFVLV becomes greater than GFIFOTH or if GVALID has become asserted when GMODE transitioned to zero. The bit is reset when FIFO is completely emptied (read). |
| PVALID | 1 | Proximity Valid. Indicates that a proximity cycle has completed since PEN was asserted or since PDATA was last read. A read of PDATA automatically clears PVALID. |
| AVALID | 0 | ALS Valid. Indicates that an ALS cycle has completed since AEN was asserted or since a read from any of the ALS/Color data registers. |

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

### 2.10.9 Proximity Data Register (0x9C)

**Proximity Data Register (0x9C)**

Proximity data is stored as an 8-bit value.

| Field | Address | Bits | Description |
|---|---|---|---|
| PDATA | 0x9C | 7:0 | Proximity data. |

In diesem Register werden die Daten von dem Abstandssensor gespeichert.

### 2.10.10 Configuration Three Register (0x9F)

**Configuration Three Register (0x9F)**

The CONFIG3 register is used to select which photodiodes are used for proximity. Two photodiodes are paired to provide signal. In proximity mode, UP and RIGHT photodiodes are connected forming a diode pair; similarly the DOWN and LEFT photodiodes form a diode pair.

| Field | Bits | Description | |
|---|---|---|---|
| RESERVED | 7:6 | Reserved. Write as 0. | |
| PCMP | 5 | Proximity Gain Compensation Enable. This bit provides gain compensation when proximity photodiode signal is reduced as a result of sensor masking. If only one diode of the diode pair is contributing, then only half of the signal is available at the ADC; this results in a maximum ADC value of 127. Enabling PCMP enables an additional gain of 2X, resulting in a maximum ADC value of 255. | |
| | | **PMASK_X (U, D, L, R)** | **PCMP** |
| | | 0, 1, 1, 1 | 1 |
| | | 1, 0, 1, 1 | 1 |
| | | 1, 1, 0, 1 | 1 |
| | | 1, 1, 1, 0 | 1 |
| | | 0, 1, 0, 1 | 1 |
| | | 1, 0, 1, 0 | 1 |
| | | All Others | 0 |
| SAI | 4 | Sleep After Interrupt. When enabled, the device will automatically enter low power mode when the INT pin is asserted and the state machine has progressed to the SAI decision block. Normal operation is resumed when INT pin is cleared over I2C. | |
| PMASK_U | 3 | Proximity Mask UP Enable. Writing a 1 disables this photodiode. | |
| PMASK_D | 2 | Proximity Mask LEFT Enable. Writing a 1 disables this photodiode. | |
| PMASK_L | 1 | Proximity Mask LEFT Enable. Writing a 1 disables this photodiode. | |
| PMASK_R | 0 | Proximity Mask RIGHT Enable. Writing a 1 disables this photodiode. | |

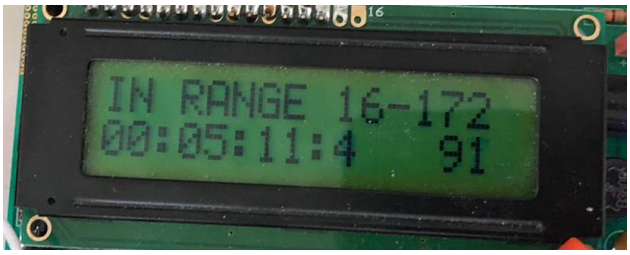
Bei diesem Register könnte man bestimmte Fotodioden deaktivieren.

## 2.11 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:




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.

### 2.11.1 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)

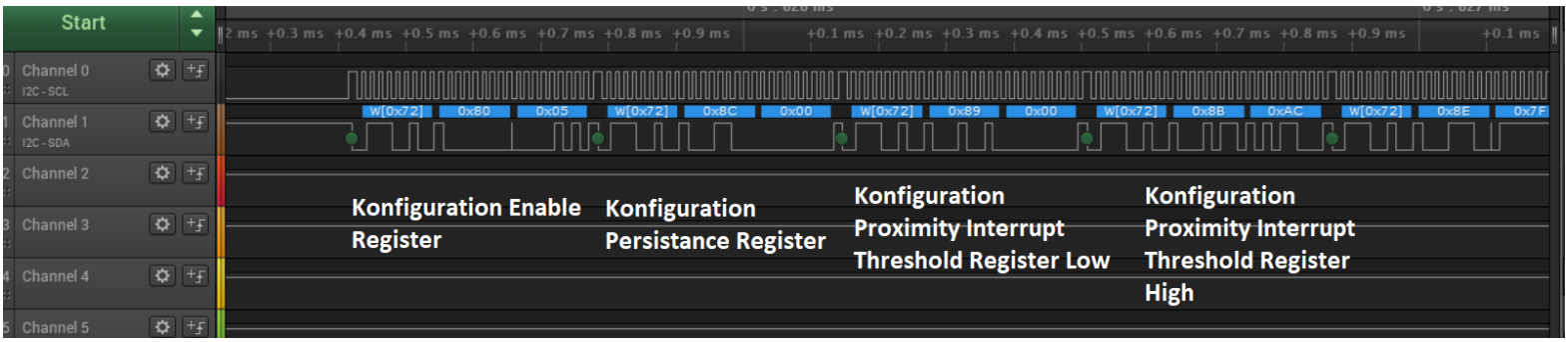
Bit 7 : 6 = reserved as 0
Bit 5 = 0 use all diodes
Bit 4 = 0 sleep after interrupt
Bit 3 = 0 Proximity Mask UP Enable
Bit 2 = 0 Proximity Mask LEFT Enable
Bit 1 = 0 Proximity Mask LEFT Enable
Bit 0 = 0 Proximity Mask RIGHT Enable

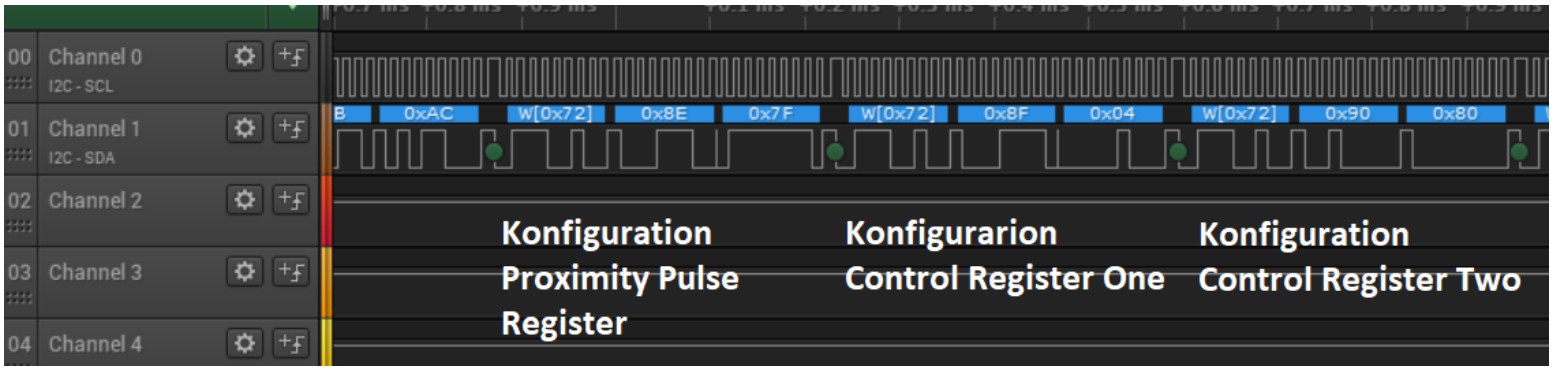
**Proximity Interrupt Clear** (Regaddr = 0xE5): 00000000b (Data = 0x00)

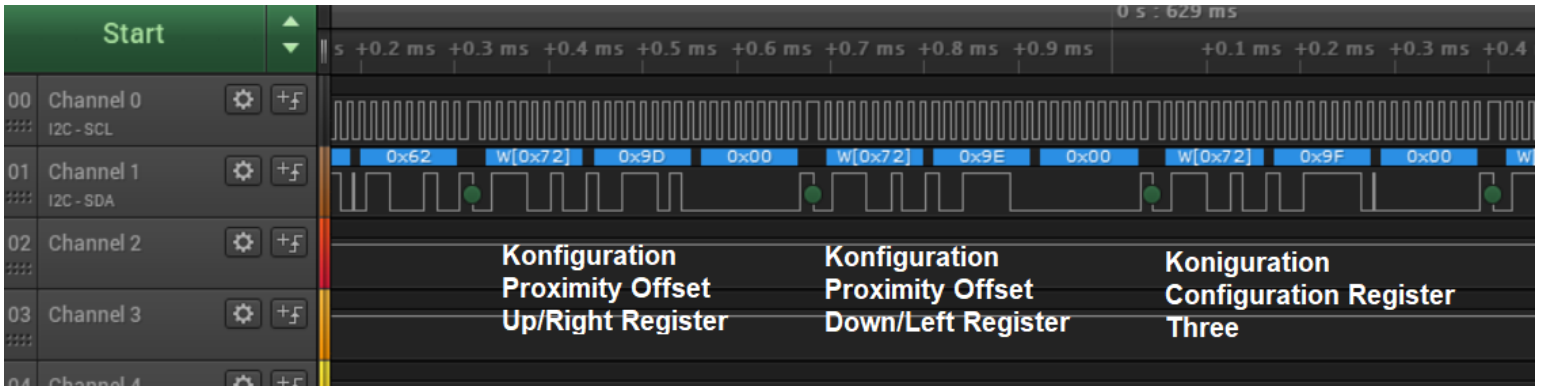
Bit 7 : 0 = 0x0

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

Bit 7 : 0 = 0x0



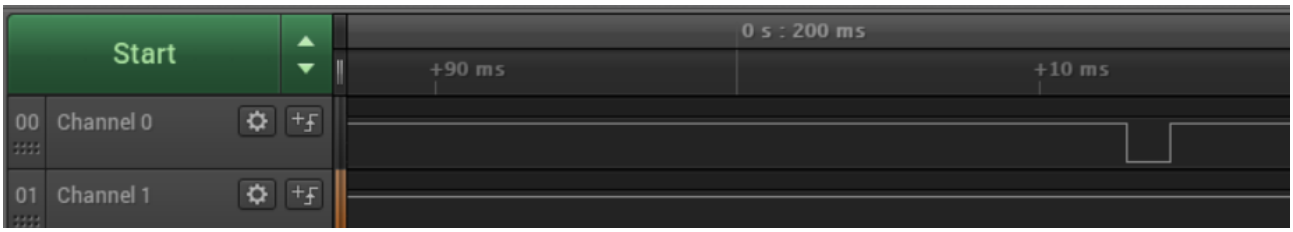




Auf diesen Bildern sieht man die Konfiguration der verschieden Register im Logik Analysator.

## 2.11.2 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.




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.

## 2.12 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

## 2.13 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 $I^2C$-Bus Protokolls
- Funktionsweise und Verwendung von Interrupts
- Funktionsweise des Timers und Berechnungen von PSC und ARR

## 2.14 Zeitaufwand – Stefan Grubmüller

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

## 2.15 Zeitaufwand – Clemens Marx

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

```c
/*****************************************************************************
* main.c                                                                     *
* Grubmüller Stefan, Marx Clemens                                            *
* May 2020                                                                   *
*                                                                            *
* Function: This programm should use the sparkfun sensor and the             *
* proximity engine. At the beginning the LCD will show you the Text:         *
*                                                                            *
* WELCOME                                                                    *
* DEVICE LINKED / DISLINKED                                                  *
*                                                                            *
* After sending data via the I2C - so data from the device/cortex can        *
* be transmitted and recieved - the LCD will show you a real time            *
* clock, the distance of a object near the threashold and if it is in        *
* threshold range. Example:                                                  *
*                                                                            *
* IN RANGE  20-140                                                           *
* 12:04:31      50                                                           *
*                                                                            *
* This process will run paralell due to interrupts.                          *
* To recieve data from the slave/device you have to configure the            *
* devive registers shown in proximity.c.                                     *

* More information in the scritum by @JosefReisinger and in the              *
* Specifications by @ClemensMarx and @StefanGrubmueller or in the            *
* datasheet by @sparkfun:                                                    *
* https://cdn.sparkfun.com/datasheets/Sensors/Proximity/apds9960.pdf         *
*****************************************************************************/

/* ------------------------------ Main ----------------------------------*/
#include "proximity.h"
#include "stdlib.h"
#include <string.h>
#include <stdio.h>
int main()
{
  // Initalisations
  // set_clock_36MHz();           // set system clock to 36MHz
  InitI2CPorts();                 // initialisation of GPIO ports (PB6 = SCL and PB7 = SDA)
  i2c_init(&device, &SCL, &SDA);  // initialisation of I2C (extra library)
  lcd_init();                     // initialisation of LCD
  lcd_clear();                    // clear screen
  uart_init(9600);                // 9600,8,n,1
  uart_clear();                   // send clear string to VT 100 terminal


  // PA1 as Input (external interrupt Pin of sparkfun sensor)
  RCC->APB2ENR |= RCC_APB2ENR_IOPAEN;  // enable clock for GPIOA (APB2 Peripheral clock enable register)
  GPIOA->CRL &= 0xFFFFFF0F;            // set Port Pins PA1 to Pull Up/Down Input mode (50MHz) = Mode 8
  GPIOA->CRL |= 0x00000080;
  GPIOA->ODR |= 0x0002;


  // starting text on LCD
  lcd_set_cursor(0, 0);           // set position on LCD
  lcd_put_string("WELCOME");      // write on LCD
  check_device_con();             // is the device connected?
  wait_ms(2000);                  // wait 2 seconds
  lcd_clear();

  // timer on lcd (real time clock)
  milsek = 0;       // initalise milliseconds
  TIM3_Config();    // start timer 3: Upcounter --> triggers every 0,1s an update interrupt



  start_proximity_engine(); // set of configuration registers for proximity detection

  EXTI_config(a, 1);        // external interrupt pin; triggers when int pin of sensor sends falling edge


  // endless loop
  while (1)
  {
```

```
75        clock_lcd();
76
77        // read data of PDATA regsiter (0x90)
78        char pdata_w[] = {0x9C};
79        i2c_write(&device, pdata_w, 1, END_WITHOUT_STOP);
80        char pdata_r;
81        i2c_read(&device, &pdata_r, 1);
82
83        // output of proximity data
84        char buffer_i [8]= {0};           // set and clear buffer
85        sprintf(buffer_i, "%d", pdata_r); // proximity data as int
86        lcd_set_cursor(1,13);
87        lcd_put_string(buffer_i);         // output of proximity data on lcd as int
88
89        //output of range (lower threshold to higher threshold)
90        char threshold[2];
91        sprintf(threshold, "%d-%d", LOWTHRES, HIGHTHRES);
92        lcd_set_cursor(0,9);
93        lcd_put_string(threshold);        // output of range on lcd as int
94
95        // check if data of proximity data register is inside range
96        if (((unsigned char)pdata_r >= LOWTHRES) && ((unsigned char)pdata_r <= HIGHTHRES))
97        {
98          char en_reg[2] = {ENABLE_REG, SET_PIEN};          // set PIEN (enable Proximity Interrupt)
99          i2c_write(&device, en_reg, 2, END_WITHOUT_STOP);  // set enable register
100       }
101       wait_ms(500);   // wait to avoid lecking image
102       lcd_clear();    // clear screen
103     }
104   }
105
```

```c
/*****************************************************************************
* proximity.h                                                                *
* Grubmüller Stefan, Marx Clemens                                            *
* May 2020                                                                   *
*                                                                            *
* Function: This header should define all the functions and variables        *
* being used in different files.                                             *
*                                                                            *
* More information in the scritum by @JosefReisinger and in the              *
* Specifications by @ClemensMarx and @StefanGrubmueller or in the            *
* datasheet by @sparkfun:                                                    *
* https://cdn.sparkfun.com/datasheets/Sensors/Proximity/apds9960.pdf         *
*****************************************************************************/

#ifndef __PROXIMITY_H__
#define __PROXIMITY_H__

/* ---------------------------Includes ---------------------------------*/

#include <stm32f10x.h>
#include "armv10_std.h"
#include "i2c.h"                    // I2C library by Jakob Pachtrog
//#include <stm32f10x_i2c.h>        // I2C default library


/* -------------------------- defines ----------------------------------*/
// bitbanding for SDA (PB7)
#define GPIOB_IDR GPIOB_BASE + 2*sizeof(uint32_t)
#define SDA_IN *((volatile unsigned long *)(BITBAND_PERI(GPIOB_IDR,7))) //PB7 - Input

#define ENABLE_REG          0x80      // enable register
#define PERS_REG            0x8C      // persistance register
#define LOWTHRES_REG        0x89      // lower threashold
#define HIGHTHRES_REG       0x8B      // higher threashold
#define PROX_PULSE_REG      0x8E      // proximity pulse register
#define CONTROL_REG1        0x8F      // control register one
#define CONTROL_REG2        0x90      // control register two
#define STAT_REG            0x93      // status register
#define UPRIGHT_OFFSET_REG  0x9D      // proximity offset UP / RIGHT register
#define DLEFT_OFFSET_REG    0x9E      // proximity offset UP / RIGHT register
#define CONF_REG3           0x9F      // configuration register three
#define PROX_INT_CLEAR      0xE5      // proximity interrupt clear
#define CLEAR_ALL_INT       0xE7      // clear all non-gesture interrupts
#define SET_PIEN            0x25      // set Proximity interrupt enable
#define DEL_PIEN            0x05      // delete PIEN bit (PIEN = 0)

// Threshold- Low to High
// 0xFF...very near
// 0x00...very far
#define LOWTHRES            0x10      // lower threashold for proximity
#define HIGHTHRES           0xAC      // higher threashold for proximity

// Light Intensity
// Very Dark to very Bright
// Various Gains
#define GAIN_x1             0x00      // very bright conditions (GAIN = 1)
#define GAIN_x2             0x04      // not that bright but also not dark ~  (GAIN = 2)
#define GAIN_x4             0x08      // need of light (normal room condtions in evening) (GAIN = 4)
#define GAIN_x8             0x0C      // dark conditions (GAIN = 8)

// activates or deactivates the LED Boost option of sensor
#define LED_BOOST_ON        0xA0      // additional current up to 200%
#define LED_BOOST_OFF       0x80      // non addtional current

extern int milsek;


/* ------------------------- Prototypes ------------------------------*/

// init ports (PA7 Open Drain)
void InitI2CPorts(void);

// interrupts
void NVIC_init(char position, char priority);
// Nestet Vector Interrupt Controller
void NVIC_init(char position, char priority);
// timer
```

```
78  void TIM3_Config(void);
79
80  // real time clock on lcd
81  void clock_lcd(void);
82  // check connection of device
83  void check_device_con(void);
84
85  // start proximity engine due to setting the register bits
86  void start_proximity_engine(void);
87  // reads data out of the Proximity data register (0x9C)
88  void read_data(void);
89
90  // transfer pins for function of EXTI_config
91  typedef enum { a, b, c, d, e, f, } pin;
92  // external interrupt
93  void EXTI_config(pin p, int n);
94
95  /* --------------------------- structures ---------------------------------*/
96
97  extern I2C_Device device;
98
99  extern I2C_PIN_CONF SCL;    // defenition of SCL (I2C1)
100
101 extern I2C_PIN_CONF SDA;    // defenition of SDA (I2C1)
102
103 #endif
104
```

```c
/*****************************************************************************
* proximity.c                                                                *
* Grubmüller Stefan, Marx Clemens                                            *
* May 2020                                                                   *
*                                                                            *
* Function: In this file you can find any functions used in the main         *
* file.                                                                      *
*                                                                            *
* More information in the scritum by @JosefReisinger and in the              *
* Specifications by @ClemensMarx and @StefanGrubmueller or in the            *
* datasheet by @sparkfun:                                                    *
* https://cdn.sparkfun.com/datasheets/Sensors/Proximity/apds9960.pdf         *
*****************************************************************************/

/* ----------------------------Includes ---------------------------------*/
#include "proximity.h"


/*----------------------- Static Variables ----------------------------*/
int h, min, sek, milsek;  // used for the timer
char buffer[30];

I2C_PIN_CONF SCL =      // defenition of SCL (I2C1)
{
  .GPIOx = GPIOB,
  .Pin = 6              // Pin PB6
};

I2C_PIN_CONF SDA =      // defenition of SDA (I2C1)
{
  .GPIOx = GPIOB,
  .Pin = 7              // Pin PB7
};

I2C_Device device =
{
  .Adress = 0x39,       // hardware adress = 0x39
  .I2C_Pereph = I2C1,   // I2C1 peripherals
  .pclr = 8,            // 36 MHz perepherial clock rate
  .sclr = 80,           // 80 kHz SCL Clock Rate (0 - 400kHZ)
  .maxRiseTime = 200    // 200ns max rise time (0 - 300ns)
};




/* ----------------------- Initalitations -----------------------------*/

// init GPIO ports PB6 and PB7 for I2C
void InitI2CPorts(void)
{
  // activate GPIO clock
  RCC->APB2ENR |= RCC_APB2ENR_IOPAEN;

  // init PB6 (SCL) as AF Open Drain
  int tmp = GPIOB->CRL;
  tmp &= 0xF0FFFFFF;      // delete configuration for PB6
  tmp |= 0x0F000000;      // define port PB6 as AF Open Drain
  GPIOB->CRL = tmp;

  // init PB7 (SDA) as AF Open Drain
  tmp = GPIOB->CRL;
  tmp &= 0x0FFFFFFF;      // delete configuration for PB7
  tmp |= 0xF0000000;      // define Port PB7 as AF Open Drain
  GPIOB->CRL = tmp;
}




/* -------------------------- Interrupts ------------------------------*/

// interrupt service routine timer3 (General Purpose Timer)
void TIM3_IRQHandler(void)  //Timer 3, very 100ms
{
  TIM3->SR &=~0x01; // clear interrupt pending bit(precent Interrupt-trigger)
  milsek = milsek + 2;
```

```
78  }
79
80
81  // External Interrupt Service Routine PA1
82  void EXTI1_IRQHandler(void)//ISR
83  {
84    EXTI->PR |= (0x01 << 1); // reset pending bit EXT0 (otherwise ISR will repeat)
85
86    lcd_set_cursor(0,0);
87    sprintf(&buffer[0],"IN RANGE");               // if the object is in range
88    lcd_put_string(&buffer[0]);
89
90    char en_reg[2] = {ENABLE_REG, DEL_PIEN};      // disable proximity interupt
91    i2c_write(&device, en_reg, 2, END_WITH_STOP); // set enable register
92  }
93
94  // initialisation Timer3 (General Purpose Timer)
95  void TIM3_Config(void)
96  {
97    /*-------------------------- configuration timer 3 -------------------------*/
98    RCC->APB1ENR |= 0x0002; // timer 3 clock enable
99    TIM3->SMCR = 0x0000;    // timer 3 clock Selection: CK_INT wird verwendet
100   TIM3->CR1  = 0x0000;    // selcet timer mode: Upcounter --> counts 0 to value of autoreload-register
101
102   // Tck_INT = 27,78ns, Presc = 54 ---> Auto Reload value = 3,6Mio (=0xFFFF) --> 0,1s Update Event
103   TIM3->PSC = 23;       //value of prescalers (Taktverminderung)
104   TIM3->ARR = 0xFFFF;   //Auto-Reload Wert = Maximaler Zaehlerstand des Upcounters
105
106   /*-------------------- configuration Interrupt Timer 3  --------------------*/
107   TIM3->DIER = 0x01;          // enable Interrupt bei einem UEV (Überlauf / Unterlauf)
108   NVIC_init(TIM3_IRQn,3);     // enable Timer 3 Update Interrupt, Priority 3
109
110   /*--------------------------- start Timer 3  -------------------------------*/
111   TIM3->CR1 |= 0x0001;        // set counter-Enable bit
112 }
113
114
115 // EXTI1_config
116 // connecct PA1 with EXTI1, Interrupt at falling edge, priority 2
117 void EXTI_config(pin p, int n)
118 {
119   NVIC_init(n + 6, 2);    //init NVIC for EXTI Line1 (Position n+6, Priority 2)
120
121   RCC->APB2ENR |= 0x0001;      //AFIOEN  - Clock enable
122   AFIO->EXTICR[0] &= ~((0xF & ~p) << (4 * n)); //Interrupt-Line EXTIn mit Portpin PAn verbinden
123   // ...
124   EXTI->FTSR |= (0x01 << n);     //Falling Edge Trigger für EXITn Aktivieren
125   EXTI->RTSR &= ~(0x01 << n);    //Rising Edge Trigger für EXTIn Deaktivieren
126
127   EXTI->PR |= (0x01 << n);  //EXTI_clear_pending: Das Auslösen auf vergangene Vorgänge nach dem
    enablen verhindern
128   EXTI->IMR |= (0x01 << n);   // Enable Interrupt EXTIn-Line. Kann durch den NVIC jedoch noch
    maskiert werden
129 }
130
131
132 // NVIC_init(char position, char priority)
133 // initialisation of an interrupts in the Nested Vectored Interrupt
134 // Controller (set priority, prevent trigger, enable
    interrupt
135 // parameters: "position" = 0-67 (number of interrupt)
136 //             "priority" = 0-15 (priority of interrupt)
137 void NVIC_init(char position, char priority)
138 {
139   NVIC->IP[position]=(priority<<4); //Interrupt priority register: Setzen der Interrupt Priorität
140   //Interrupt Clear Pendig Register: prevent trigger after enable
141   NVIC->ICPR[position >> 0x05] |= (0x01 << (position & 0x1F));
142   //Interrupt Set Enable Register: Enable interrupt
143   NVIC->ISER[position >> 0x05] |= (0x01 << (position & 0x1F));
144 }
145
146 /* --------------------------- functions --------------------------------*/
147
148 // after 0.1 seconds an interrupt is being triggered (milsek ++)
149 // function prints output real clock on lcd in hh:mm:ss:z
    (hour:minute:second:millisecond)
150 void clock_lcd(void)
```

```
151   {
152     if(milsek==10)
153     {
154       milsek=0;
155       if(++sek==60)
156       {
157         sek=0;
158         if(++min==60)
159         {
160           min=0;
161           if(++h==24)
162           {
163             h=0;
164           }
165         }
166       }
167     }
168     else if(milsek>=10)   // error detection
169     {
170       milsek=0;
171     }
172
173     // output of lcd
174     sprintf(&buffer[0], "%02d:%02d:%02d:%d", h, min, sek, milsek);
175     lcd_set_cursor(1,0);
176     lcd_put_string(&buffer[0]);
177   }
178
179   void check_device_con()
180   {
181     // check if device is there and connected
182     char ack;
183     int i;
184     char dev_con[] = {"DEVICE CONNECTED"};                // device connected
185     char dev_discon[] = {"NOT CONNECTED"};        // device disconnected
186     char buffer[1] = {0x1};
187
188     i2c_write(&device, buffer, 1, END_WITH_STOP); // write bit to test gettin ack for checking
    connection
189     ack = SDA_IN;         // read one bit of data pin
190     lcd_set_cursor(1,0);  // read acknowledge
191     if (ack == 0x0)       // if there is an acknowledge
192     {
193       // output
194       for(i=0; dev_con[i] != '\0'; i++)
195       {
196         lcd_put_char(dev_con[i]);                 // device connected
197       }
198     }
199     else                  // if no ack detected
200     {
201       // output
202       for(i=0; dev_discon[i] != '\0'; i++)
203       {
204         lcd_put_char(dev_discon[i]);              // device disconnected
205       }
206     }
207   }
208
209
210   // start proximity engine (configuration of registers)
211   void start_proximity_engine()
212   {
213
214     // Enable Register (0x80):
215     // 00000101 (0x05)
216     // Bit 7 = 0: is being reserved as 0
217     // Bit 6 = 0: gesture enable (GEN)
218     // Bit 5 = 0: proximity interrupt enable (PIEN)
219     // Bit 4 = 0: ambient light sense (ALS) interrupt enable (AIEN)
220     // Bit 3 = 0: wait enable (WEN) actives wait feature
221     // Bit 2 = 1: proximity detect enable (PEN)
222     // Bit 1 = 0: ALS enable (AEN)
223     // Bit 0 = 1: Power ON (PON)
224     char en_reg[2] = {ENABLE_REG, 0x05};
225     i2c_write(&device, en_reg, 2, END_WITH_STOP); // set enable register
226
```

```
227     // Persistance Register (0x8C):
228     // 00000000 (0x0)
229     // Bit 7 : 4 = 0: Controls rate of proximity interrupt to host process
230     // Bit 3  : 0 = 0: Controls rate of clear channel interrupt to host process
231     char pers_reg[2] = {PERS_REG, 0x00};
232     i2c_write(&device, pers_reg, 2, END_WITH_STOP);
233
234
235     // Proximity Interrupt Threshold Register - Low (0x89):
236     // set as define LOWTHRES in proximity.h
237     // Bit 7 : 0 = ?? (adjustable)
238     char low_threshold[2] = {LOWTHRES_REG, 0x00};
239     i2c_write(&device, low_threshold, 2, END_WITH_STOP);
240
241
242     // Proximity Interrupt Threshold Register - High (0x8B):
243     // set as define HIGHTHRES in
    proximity.h
244     // Bit 7 : 0 = ?? (adjustable)
245     char high_threshold[2] = {HIGHTHRES_REG, HIGHTHRES};
246     i2c_write(&device, high_threshold, 2, END_WITH_STOP);
247
248
249     // Proximity Pulse Register (0x8E):
250     // 01111111 (7F)
251     // Bit 7 : 6 = 01
252     // Bit 5 : 0 = 1: 111111
253     char prox_pulse_reg[2] = {PROX_PULSE_REG, 0x7F};
254     i2c_write(&device, prox_pulse_reg, 2, END_WITH_STOP);
255
256
257     //  Control Register One(0x8F):
258     //  0000??00
259     //  Bit 7 : 6 = 10
260     //  Bit 5 : 4 = reserved as 0
261     //  Bit 3 : 2 = LIGHT_INTENSITY set in defines
262     //  Bit 1 : 0 = 0
263     char control_reg1[2] = {CONTROL_REG1, GAIN_x2};
264     i2c_write(&device, control_reg1, 2, END_WITH_STOP);
265
266     //  Control Register Two(0x90):
267     // 10??0000
268     // Bit 7 = 1
269     // Bit 6 = 0
270     // Bit 5 : 4 = LED_BOOST_ON/LED_BOOST_OFF
271     // Bit 3 : 0 = reserved as 0
272     char control_reg2[2] = {CONTROL_REG2, LED_BOOST_OFF};
273     i2c_write(&device, control_reg2, 2, END_WITH_STOP);
274
275
276     // Status Register(0x93):
277     // 01100010 (0x62)
278     // The read-only Status Register provides the status of the device.
279     // Bit 7 = 0  Clear Photodiode Saturation
    (CPSAT)
280     // Bit 6 = 1  Indicates that an analog saturation event occurred
    (PGSAT)
281     // Bit 5 = 1  Proximity Interrupt. This bit triggers an interrupt if PIEN in ENABLE is set
    (PINT)
282     // Bit 4 = 0  ALS Interrupt (AINT)
283     // Bit 3 =    reserved as 0
284     // Bit 2 = 0  Gesture Interrupt (GINT)
285     // Bit 1 = 1  Proximity Valid (PVALID)
286     // Bit 0 = 0  ALS Valid (AVALID)
287     char stat_reg[2] = {STAT_REG, 0x62};
288     i2c_write(&device, stat_reg, 2, END_WITH_STOP);
289
290
291     // Proximity Offset UP / RIGHT Register(0x9D)
292     // 00000000 (0x00)
293     // Bit 7 : 0 = 0x0
294     char ur_offset_reg[2] = {UPRIGHT_OFFSET_REG, 0x00};
295     i2c_write(&device, ur_offset_reg, 2, END_WITH_STOP);
296
297
298     // Proximity Offset DOWN / LEFT Register(0x9E)
299     // 00000000 (0x00)
```

```
300     // Bit 7 : 0 = 0x0
301     char dl_offset_reg[2] = {DLEFT_OFFSET_REG, 0x00};
302     i2c_write(&device, dl_offset_reg, 2, END_WITH_STOP);
303
304
305     //Configuration Register Three(0x9F):
306     // 00000000 (0x00)
307     // select which photodiodes are used for proximity
308     // Bit 7 : 6 = reserved as 0
309     // Bit 5 = 0 use all diodes
310     // Bit 4 = 0 sleep after interrupt
311     // Bit 3 = 0 Proximity Mask UP Enable
312     // Bit 2 = 0 Proximity Mask LEFT Enable
313     // Bit 1 = 0 Proximity Mask LEFT Enable
314     // Bit 0 = 0 Proximity Mask RIGHT Enable
315     char conf_reg3[2] = {CONF_REG3, 0x00};
316     i2c_write(&device, conf_reg3, 2, END_WITH_STOP);
317
318
319   //  // Proximity Interrupt Clear (0xE5):
320   //  // 00000000 (0x00)
321   //  // Bit 7 : 0 = 0x0
322   //  char prox_int_clear[2] = {PROX_INT_CLEAR, 0x00};
323   //  i2c_write(&device, prox_int_clear, 2, END_WITH_STOP);
324
325
326   //  // Clear All Non-Gesture Interrupts (0xE7):
327   //  // 00000000 (0x00)
328   //  // Bit 7 : 0 = 0x0
329   //  char clear_all_int[2] = {CLEAR_ALL_INT, 0x00};
330   //  i2c_write(&device, clear_all_int, 2, END_WITH_STOP);
331   }
332
```