

# DIY Optische ToF Distanzmessung

CAS Sensorik und Sensor Signal Conditioning

**Matthias Schär, Timon Burkard**  
OST – Ostschweizer Fachhochschule

15. Januar 2025



# **CAS Sensorik und Sensor Signal Conditioning an der OST – Ostschweizer Fachhochschule**

<b>Titel</b>	<b>DIY Optische ToF Distanzmessung</b>
<b>Diplomandin/Diplomand</b>	<b>Matthias Schär, Timon Burkard</b>
<b>Studiengang</b>	<b>CAS Sensorik und Sensor Signal Conditioning</b>
<b>Semester</b>	<b>HS24</b>
<b>Dozentin/Dozent</b>	<b>Prof. Guido Keel, Michael Lehmann</b>

## **Abstract**

Die vorliegende Projektarbeit befasst sich mit der Entwicklung eines...

Ort, Datum                            Rapperswil, 15. Januar 2025  
© Matthias Schär, Timon Burkard, OST – Ostschweizer Fachhochschule

---

Alle Rechte vorbehalten. Die Arbeit oder Teile davon dürfen ohne schriftliche Genehmigung der Rechteinhaber weder in irgendeiner Form reproduziert noch elektronisch gespeichert, verarbeitet, vervielfältigt oder verbreitet werden.

Sofern die Arbeit auf der Website der Ostschweizer Fachhochschule online veröffentlicht wird, können abweichende Nutzungsbedingungen unter Creative-Commons-Lizenzen gelten. Massgebend ist in diesem Fall die auf der Website angezeigte Creative-Commons-Lizenz.

# Inhaltsverzeichnis

# Abkürzungsverzeichnis

**ADC** Analog to Digital Converter. 16

**CPU** Central Processing Unit. 35, 36, 38, 39, 44

**DIY** Do It Yourself. 10

**DSO** Digital Storage Oscilloscope. 6, 35, 37, 38, 40, 41

**GPIO** General Purpose Input/Output. 6, 7, 9, 35–42, 44

**HAL** Hardware Abstraction Layer. 6, 7, 9, 35–41, 44

**IC** Integrated Circuit. 17

**MCU** Microcontroller Unit. 16–18, 20–22, 35, 38, 44

**PCB** Printed Circuit Board. 26

**SPI** Serial Peripheral Interface. 20, 21

**TDC** Time to Digital Converter. 6, 10, 17, 18, 20–24, 36, 37, 39, 40, 43–45, 48

**TIA** Trans-Impedance Amplifier. 15, 23, 24

**ToF** Time of Flight. 10, 11, 16, 17, 19

**UART** Universal Asynchronous Receiver/Transmitter. 36, 39

**USB** Universal Serial Bus. 20

# Abbildungsverzeichnis

# Formelverzeichnis

1	Eintreffende Lichtleistung . . . . .	11
2	Strahlungsintensität . . . . .	11
3	Raumwinkel . . . . .	11
4	Photostrom . . . . .	11
5	Werte des RLD94PZJ5 . . . . .	11
6	Werte des BPV23NF . . . . .	11
7	Nummerische Resultate mit RLD94PZJ5 und BPV23NF . . . . .	12
8	Werte des RLD65NZX1 . . . . .	12
9	Werte des NJL6401R-3 . . . . .	12
10	Nummerische Resultate mit RLD65NZX1 und NJL6401R-3 . . . . .	12
11	Vergrösserung der Empfangsfläche durch Linse . . . . .	13
12	Maximale räumliche Auflösung des TDC7200 . . . . .	16
14	GPIO Toggle mit HAL . . . . .	35
15	GPIO Toggle ohne HAL . . . . .	38
16	Zurückrechnen auf Kabellänge . . . . .	41
17	Mode 1 vs. Mode 2 . . . . .	44

# Tabellenverzeichnis

# Codeverzeichnis

1	GPIO Toggle mit HAL . . . . .	34
2	GPIO Toggle mit HAL . . . . .	35
3	GPIO Toggle ohne HAL . . . . .	37
4	GPIO Toggle ohne HAL . . . . .	38
5	Mode 1 . . . . .	43
6	TDC Driver (Header) . . . . .	47
7	TDC Driver (Source) . . . . .	51
8	Python Analyse . . . . .	55
9	Python Analyse (Multi) . . . . .	56

# 1 Einleitung

Bei dieser Projektarbeit geht es darum ein Do It Yourself (DIY) optisches Time of Flight (ToF) Distanzmesssystem aufzubauen. Dazu soll ein Time to Digital Converter (TDC) verwendet werden.

...

## 2 Theorie

### 2.1 Time of Flight

Bei ToF handelt es sich um ...

## 2.2 Photostrom

Zur Berechnung des theoretisch zu erwartenden Photostrom wird von einer Distanz zur Wand von 10 m ausgegangen.

Der Laserstrahl gehe idealisiert mit 0° zur Wand und werde dort uniform Halbkugel-förmig gestreut. In der Realität wird sich die Streuung nicht uniform verteilen, sondern in der Mitte stärker konzentriert sein. Die folgende Berechnung zeigt also den worst case.

Die Berechnung der empfangenen Strahlungsleistung, der Strahlungsintensität, dem Raumwinkel und dem Photostrom sind in Formel 1, 2, 3 bzw. 4 gezeigt.

$$P_{in} = E_e \cdot A = \frac{I_e}{r^2} \cdot A \quad (1)$$

$$I_e = \frac{P_{out}}{\Omega} \quad (2)$$

$$\Omega = \frac{4 \cdot \pi \cdot 0.5}{d} \quad (3)$$

$$I_{ph} = S \cdot P_{in} \quad (4)$$

### 2.2.1 Berechnung mit RLD94PZJ5 und BPV23NF

Ersten Berechnungen wurden mit der Laserdiode RLD94PZJ5 (ROHM, 2020) und der Photodiode BPV23NF (Vishay Semic., 2024) durchgeführt.

Die relevanten Werte aus den Datenblättern sind in Formel 5 und 6 aufgelistet.

$$P_{out} = 285 \text{ mW} \quad (5)$$

$$\begin{aligned} A &= 4.4 \text{ mm}^2 \\ S &= 0.6 \frac{A}{W} \end{aligned} \quad (6)$$

Diese Werte eingesetzt in Formel 2, 1 und 4 ergibt die Resultate in Formel 7.

$$\begin{aligned}
 I_e &= \frac{P_{out}}{\Omega} = \frac{285 \text{ mW}}{\frac{4\pi \cdot 0.5}{d}} = \frac{285 \text{ mW}}{\frac{4\pi \cdot 0.5}{10 \text{ m}}} = 45 \frac{\text{mW}}{\text{sr}} \\
 P_{in} &= \frac{I_e}{r^2} \cdot A = 45 \frac{\text{mW}}{\text{sr}} \cdot 4.4 \text{ mm}^2 = 2 \text{ nW} \\
 I_{ph} &= S \cdot P_{in} = 0.6 \frac{A}{W} \cdot 2 \text{ nW} = 1.2 \text{ nA}
 \end{aligned} \tag{7}$$

## 2.2.2 Berechnung mit RLD65NZX1 and NJL6401R-3

Die Laserdiode RLD94PZJ5 hat im Bezug auf diese Projektarbeit zwei Nachteile: Sehr hohe Leistung, welche für das menschliche Auge gefährlich werden kann und ein Wellenlängenbereich, der für das menschliche Auge nicht sichtbar ist.

Aus diesen Gründen wurde eine zweite Laserdiode evaluiert: RLD65NZX1 (ROHM, 2019). Gepaart wird sie mit der Photodiode NJL6401R-3 (JRC, 2014). Die folgenden Berechnungen wurden basierend auf diesen beiden Komponenten durchgeführt.

Die relevanten Werte aus den Datenblättern sind in Formel 8 und 9 aufgelistet.

$$P_{out} = 10 \text{ mW} \tag{8}$$

$$\begin{aligned}
 A &= 0.7 \text{ mm} \cdot 0.7 \text{ mm} = 0.49 \text{ mm}^2 \\
 S &= 0.42 \frac{A}{W}
 \end{aligned} \tag{9}$$

Diese Werte eingesetzt in Formel 2, 1 und 4 ergibt die Resultate in Formel 10.

$$\begin{aligned}
 I_e &= \frac{P_{out}}{\Omega} = \frac{10 \text{ mW}}{\frac{4\pi \cdot 0.5}{d}} = \frac{10 \text{ mW}}{\frac{4\pi \cdot 0.5}{10 \text{ m}}} = 1.6 \frac{\text{mW}}{\text{sr}} \\
 P_{in} &= \frac{I_e}{r^2} \cdot A = 1.6 \frac{\text{mW}}{\text{sr}} \cdot 0.49 \text{ mm}^2 = 8 \text{ pW} \\
 I_{ph} &= S \cdot P_{in} = 0.42 \frac{A}{W} \cdot 8 \text{ pW} = 3.3 \text{ pA}
 \end{aligned} \tag{10}$$

## 2.2.3 Berechnung mit Empfangs-Linse

Auf Vorschlag der Dozenten wurden verschiedene Optiken aus einem Baukasten-System von QI-OPTIQ ausprobiert. Besonders vielversprechend erschien hierbei eine Linse mit einem Durchmesser von 17 mm bei einer Brennweite von 40 mm. Eine solche Linse vergrößert die effektive Fläche, auf welcher der Lichtstrom empfangen werden kann, was eine höhere Empfangsleistung, sprich einen höheren Lichtstrom zur Folge hat.

Im Idealfall kann diese Optik die Fläche um etwa folgenden Faktor vergrößern:

$$A' = \frac{A_L}{A_{PD}} = \frac{\left(\frac{17 \text{ mm}}{2}\right)^2 \cdot \pi}{0.49 \text{ mm}^2} = 463.2 \quad (11)$$

TODO: Wollen wir hier noch eine bessere Annäherung für den Laser machen?

## 2.3 Transimpedanzverstärker

Bei einem Trans-Impedance Amplifier (TIA) handelt es sich um ...

## 3 Konzept

Die folgenden Unterkapitel zeigen auf, wie die entwickelte Schaltung aussehen wird. Hierbei wird vor allem darauf eingegangen, wie eine Time of Flight Elektronik aufgebaut sein kann. Zudem werden weitere Herausforderungen eines solchen Systems und die Bewältigung ebendieser aufgezeigt.

### 3.1 Signalkette

Dieses Unterkapitel soll näherbringen, wie die ToF-Messung realisiert wird. Eine Skizze der Signalkette ist in der Abbildung 1 ersichtlich.

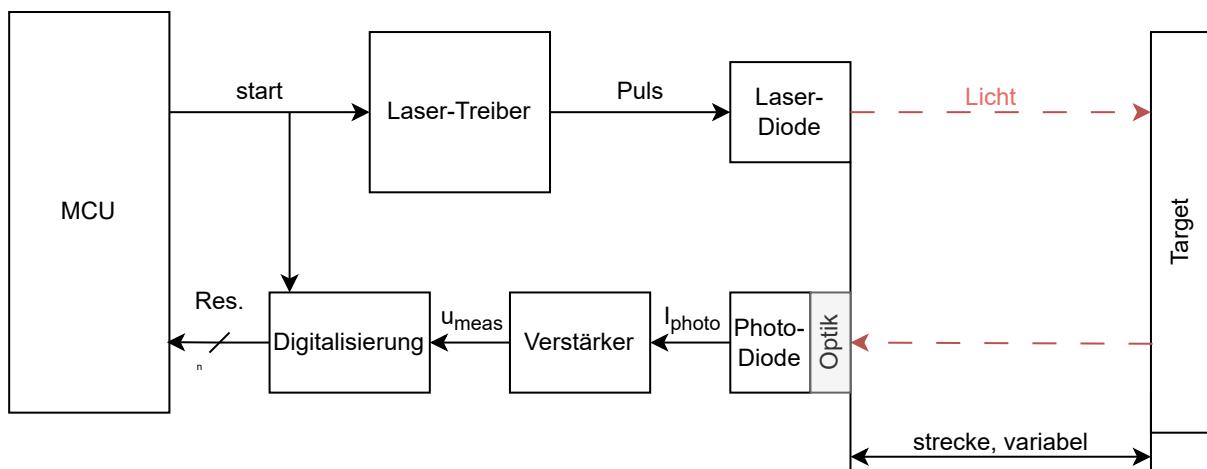


Abbildung 1: Konzeptioneller Signalfad

Eine MCU kümmert sich um das Starten und Auswerten der Messresultate. Das Generieren eines Startsignals hat zur Folge, dass über einen Laser-Treiber eine Laser-Diode mit einem Puls angesteuert wird. Dieser Puls, nach der Laser-Diode in Form eines Lichtpulses, wird dann von einem Target reflektiert, welches in einer variablen Distanz zur Elektronik steht.

Der Reflektierte Lichtpuls kann über eine Optik eingefangen werden und wird anschliessend von einer Photo-Diode in einen Lichtstrom umgewandelt. Dieser Lichtstrom wird von einem Transimpedanzverstärker in eine Spannung umgewandelt. Bevor die MCU diese Spannung auswerten kann, muss diese erst digitalisiert werden. Das passiert unter Verwendung eines Komparators. Dieser kann als 1 bit ADC verstanden werden, da bloss eine Unterscheidung der beiden Zustände "Licht" und "kein Licht" notwendig ist. Es ergibt sich also ein Empfangspuls.

Wird nun die Zeit gemessen zwischen Start- und Empfangspuls, ist es möglich einen Rückschluss zu ziehen über die variable Strecke zwischen Elektronik und Target. Die Zeitmessung unterliegt höchsten Anforderungen. Als Versinnbildlichung: Innert einer Nanosekunde kann Licht eine Distanz von 30 cm zurücklegen. Wie eine solche Zeitmessung realisiert werden kann, soll das nächste Kapitel erklären.

## 3.2 Zeitmessung

Damit ein direct ToF-Signal ausgewertet werden kann, benötigt es eine hochauflöste Zeitmessung. Die Firma Texas Instruments bietet dafür mit dem TDC7200 eine gute Lösung an. Laut Datenblatt (TI, 2016b) kann dieser bis zu 55 ps auflösen, mit einer Standardabweichung von 35 ps. Setzt man nun die Lichtgeschwindigkeit sowie diese minimale zeitliche Auflösung in eine Bewegungsgleichung ein, sollte der Baustein gemäss der Formel 12 folgende räumliche Auflösung erreichen:

$$s_{min} = c \cdot t = 299.8 \cdot 10^6 \frac{m}{s} \cdot 55 \text{ ps} = 16.5 \text{ mm} \quad (12)$$

Prinzipiell soll die ToF-Funktionalität in zwei Schritten in Betrieb genommen werden. In einem ersten Teil soll es rein darum gehen, den TDC7200 IC genauer kennenzulernen.

Bemerkung: In der Realität wird eine solche Auflösung nicht erreichbar sein. Die Herausforderungen bestehen beispielsweise darin, dass in der gesamten Signalkette diverse Verzögerungen vorhanden sind. So spielen zum Beispiel das Einschalten der Laser-Diode, die Bandbreite des Verstärkers sowie die Schaltzeit des Komparators eine grosse Rolle. Auch der Jitter der MCU beim Ein- und Ausschalten ihrer Ausgänge wird bei der minimalen Auflösung ins Gewicht fallen. Im Kapitel **TODO Verweis** wird mittels verschiedener Messungen dargestellt, wie nah an die vom TDC7200 vorgegebene Grenze rangekommen wird.

## 3.3 Vorangehensweise

Wie bereits erläutert ist davon auszugehen, dass der gesamte Signalpfad eine Ungenauigkeit in die Messung einbringt, welche weit ausserhalb derjenigen des TDC7200 liegt. Es bietet sich also an, dass diese im Vorfeld untersucht wird. Dazu wird ein zweiter TDC7200 auf der Elektronik eingesetzt, welcher zur Messung eines rein elektrischen Signals zuständig ist.

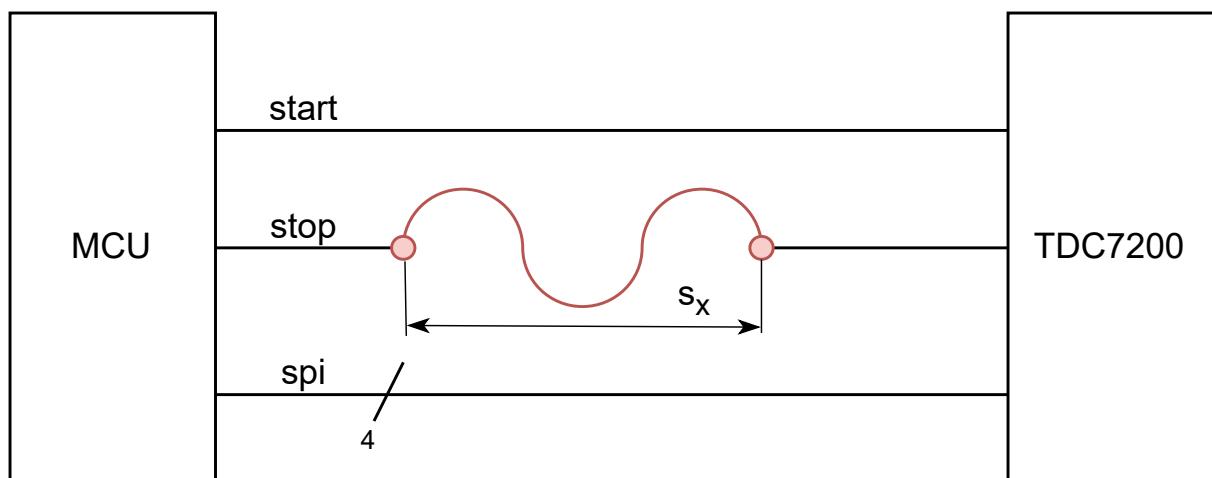


Abbildung 2: Konzept Betrieb TDC rein elektrisch

In Abbildung 2 sei skizziert, wie eine solche Schaltung aussehen könnte. Die gewellte, rot markierte Linie versinnbildlicht dabei einen Variablen Messpfad. Über einen Stecker können hier

unterschiedlich lange Kabel angeschlossen werden. Somit ist es möglich, die Messgenauigkeit des TDC7200 rein elektrisch nachzuweisen.

Folgende Punkte können mit einer solchen Erweiterung überprüft werden:

- Messung Jitter der Ausgänge der verwendeten MCU
- Konfiguration und Inbetriebnahme TDC7200 (z.B. verschiedene Messmodi)
- Auswertung der Resultate des TDC7200

## 4 Umsetzung

### 4.1 Firmware

Der selbst entwickelte Firmware-Treiber für den TDC befindet im Anhang Kapitel 8.1.

## 4.2 Schaltungen

Nachfolgend werden sämtliche Teil-Schaltungen thematisiert, welche für das entwickelten ToF-Evaluationsmodul nötig sind. Ein vollständiges Schema kann dem Anhang 8.3 entnommen werden. Die kompletten Projekt-Dateien sind im elektronischen Anhang dieses Projektes verfügbar. Designt wurde das Schema mit Hilfe des Open-Source Tools "KiCad EDA 8.0".

### 4.2.1 Selective Input Voltage

Abbildung 3 zeigt die Beschaltung zur Selektion der Speisung.

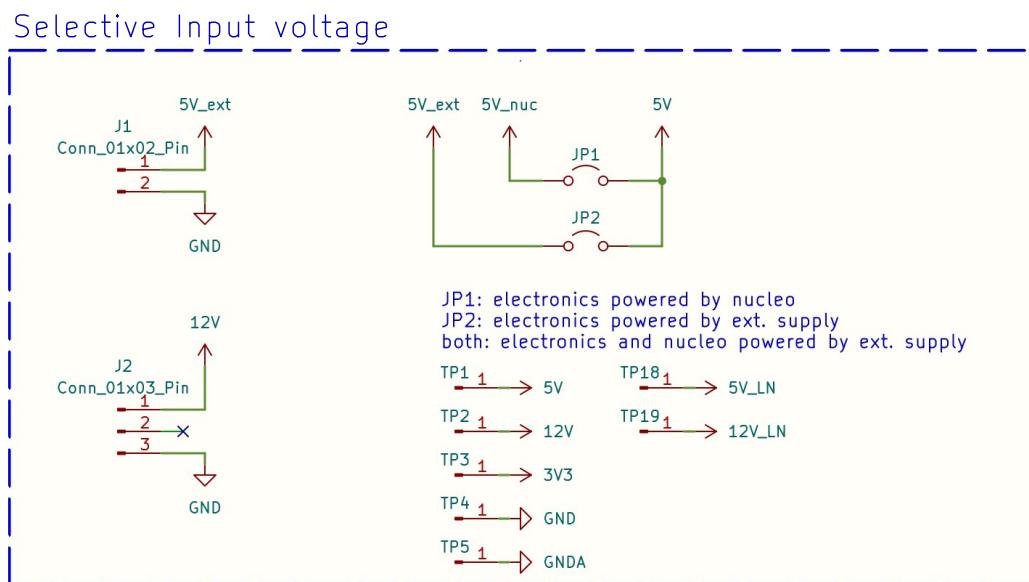


Abbildung 3: Selective Input Voltage

Für die Speisung des Nucleo-Boards bestehen die folgenden Möglichkeiten:

- 5V von USB-Buchse
- 5V von externem Power-Supply (JP1 + JP2)
- 12V von externem Power-Supply (JP3)

Siehe dazu auch Kapitel 4.2.2.

Für die Speisung der 5V-Elektronik bestehen die folgenden Möglichkeiten:

- 5V von Nucleo-Board (JP1)
- 5V von externem Power-Supply (JP2)
- 12V von externem Power-Supply via Nucleo-Board (JP1 + JP3)

Für die Speisung der Photodiode bestehen die folgenden Möglichkeiten:

- 5V von 5V-Elektronik (SW2 Position 3)
- 12V von externem Power-Supply (SW2 Position 1)

Siehe dazu auch Kapitel 4.2.8.

## 4.2.2 Nucleo Board

Die Beschaltung des NUCLEO-F042K6 Boards (ST, 2024) ist in Abbildung 4 gezeigt.

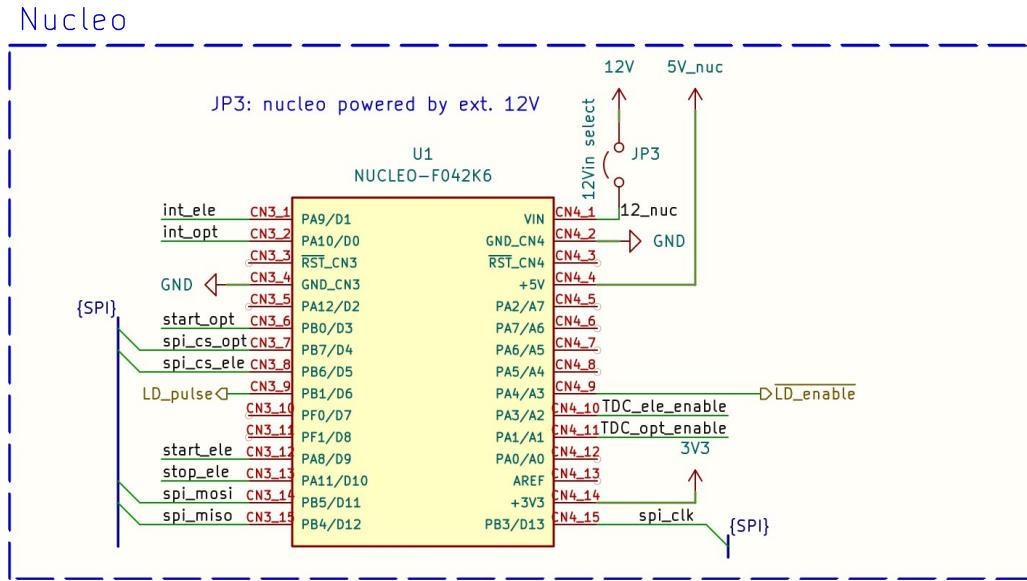


Abbildung 4: Nucleo Board

Das NUCLEO Board ist ein sogenanntes "Development-Kit", welches eine STM32F042K6 MCU beinhaltet. Am Rand des Boards werden diverse Pins der Microcontroller Unit (MCU) via Pin-Header einfach zur Verfügung gestellt. Dies erleichtert die Integration in eine eigene Elektronik enorm. Programmiert wird die MCU über eine USB-Schnittstelle.

In diesem Design wird das Entwicklerboard dazu benötigt, die TDC7200 ICs zu bedienen. Dazu wird einerseits ein Serial Peripheral Interface (SPI) benötigt, um die Mess-ICs zu konfigurieren und auch auszulesen (siehe SPI-Bus in der Abbildung). Weiter können auf beiden TDCs Messungen gestartet werden mit den Signalen `start_ele`, resp. `start_opt`. Für den TDC welcher sich um die elektrischen Signale kümmert kann zudem mit `stop_ele` ein Stopp-Puls generiert werden. Zuguterletzt ist das NUCLEO dafür zuständig, die Laser-Diode anzusteuern, was mit den Signalen `LD_pulse` sowie `LD_enable` geschieht.

## 4.2.3 TDC Electrical Signal

Die Beschaltung des TDC7200 (TI, 2016b) für den elektrischen Teil ist in Abbildung 5 gezeigt.

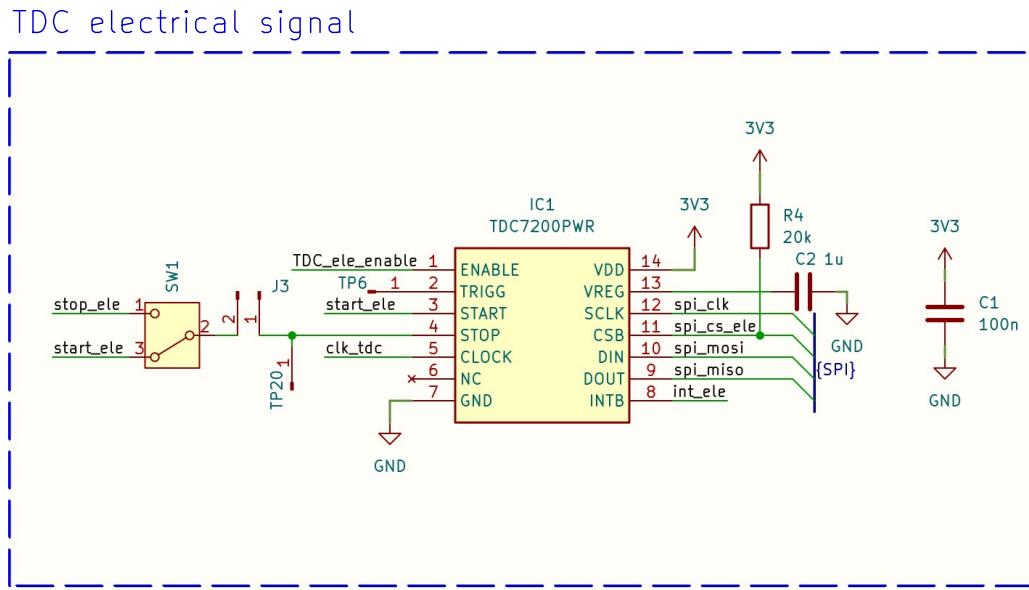


Abbildung 5: TDC Electrical Signal

Der TDC7200 ist über die Start- und Stoppleitungen, ein Enable-Signal sowie die SPI-Leitungen mit der MCU verbunden.

Wie bereits im Kapitel 3.3 angesprochen, wird der elektrische TDC dazu verwendet, den Chip erstmalig in Betrieb zu nehmen und sich damit vertraut zu machen. Am Anschluss J3 kann in einem nächsten Schritt ein beliebig langes Kabel angeschlossen werden. Dies ermöglicht es, erste Messresultate, natürlich rein elektrisch, vom TDC7200 auszulesen. Das stop-Signal kann wahlweise via `stop_ele` oder `start_ele` generiert werden.

#### 4.2.4 TDC Optical Signal

Die Beschaltung des TDC7200 (TI, 2016b) für den optischen Teil ist in Abbildung 6 gezeigt.

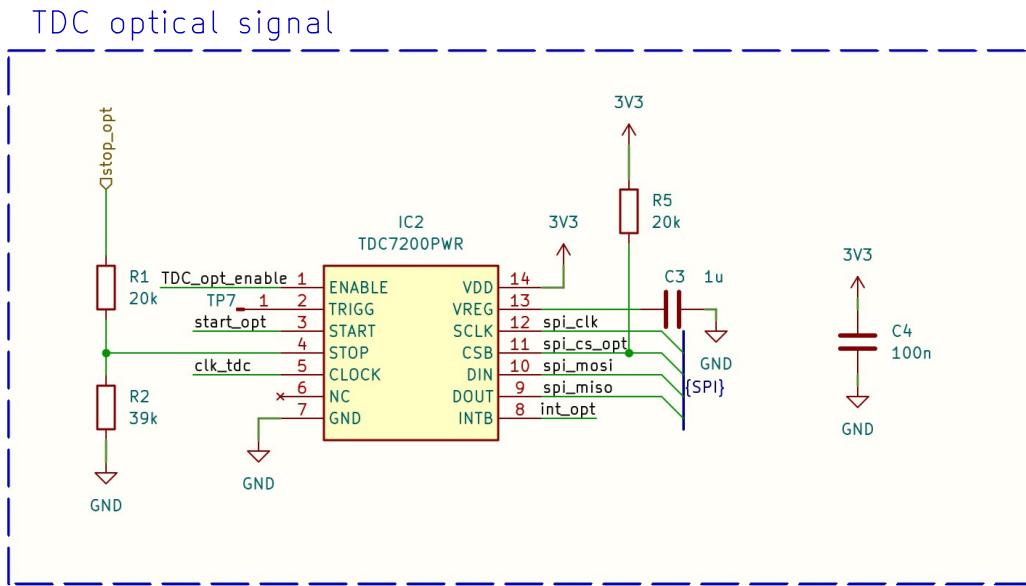


Abbildung 6: TDC Optical Signal

Die Beschaltung des TDCs für die optische Messung gestaltet sich praktisch gleich wie beim elektrischen Gegenstück. Der hauptunterschied ist, hier aber, dass dessen **STOP**-Signal nicht von der MCU selber generiert wird, sondern von einem Komparator, welcher am Ende des optischen Messpfades steht. Da der Komparator mit einem 5 V Pegel arbeitet, ist der Spannungsteiler  $R_1 / R_2$  vonnöten, welcher den 5 V Puls auf die geeigneten 3.3 V herunterteilt.

#### 4.2.5 Oscillator For TDCs

Die Beschaltung des Oszillators für die beiden TDC ist in Abbildung 7 gezeigt. Es handelt sich hierbei um einen normalen Quartz-Oszillator mit integriertem Schwingkreis. Praktischerweise ist bei diesem also keine weitere Beschaltung notwendig.

Oscillator for TDCs

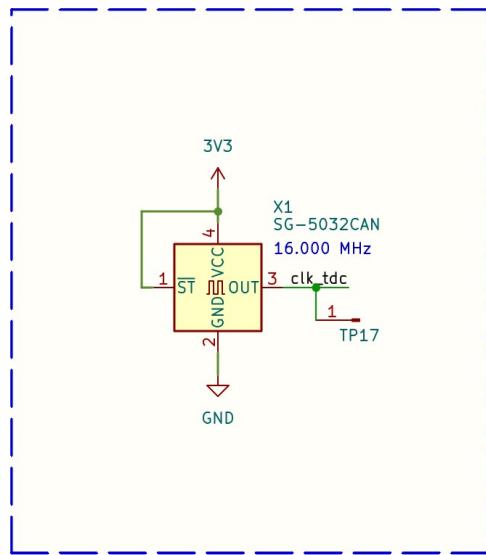


Abbildung 7: Oscillator for TDCs

#### 4.2.6 Power Supply Separation

Für die Beschaltung der Photodiode, inkl. TIA und Komparator, macht es Sinn eine Spannungsversorgung mit möglichst wenig Rauschen zu haben.

Dazu wurde die Separierung vorgenommen, welche in Abbildung 8 dargestellt ist.

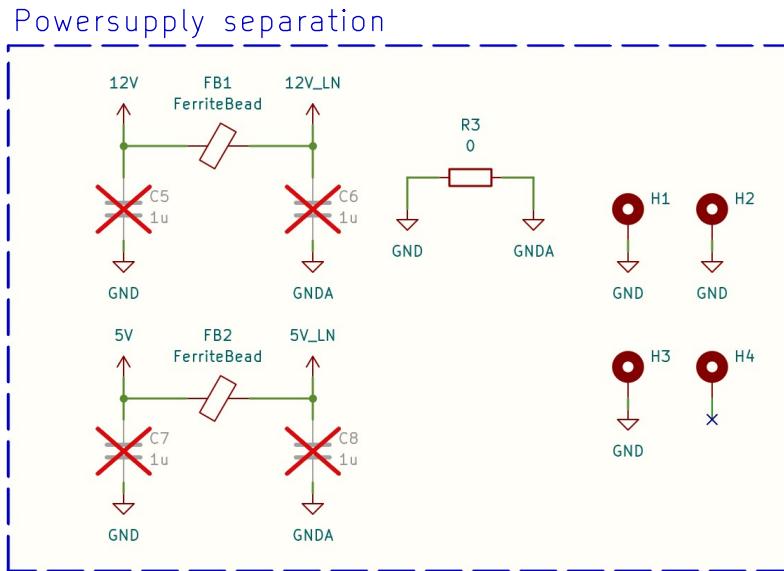


Abbildung 8: Power Supply Separation

Prinzipiell sollen die Speisungen über eine Ferrit-Perle etwas entstört werden. Massgeblich ist hier natürlich auch der physikalische Verlauf des Speisungspfades auf dem Layout. Dazu mehr im entsprechenden Kapitel 4.3. Wahlweise besteht nebst den Ferrit-Perlen die Möglichkeit, mittels

Kondensatoren die Speisung weiter zu entkoppeln. Es wird jedoch davon ausgegangen, dass dies in einem ersten Schritt nicht notwendig ist.

#### 4.2.7 Laser Driver

Die Laser Diode RLD65NZX1 (ROHM, 2019) wird mittels Lasertreiber LMG1025-Q1 (TI, 2024) und NexFET (TI, 2016a) angesteuert. Für die Generierung eines kurzen Pulses ( $0.5 \dots 100 \text{ ns}$ ) wurde mittels Hochpass und AND-Gatter (Diodes Inc., 2020) implementiert. Siehe dazu Abbildung 9.

Der Widerstand  $R_{25}$  bildet den Vorwiderstand für die Laser-Diode, welcher sich gemäss der Formel 13 berechnet.

$$R_v = \frac{V_{cc} - V_{fld}}{I_{ld}} = \frac{5 \text{ V} - 2 \text{ V}}{40 \text{ mA}} = 75 \Omega \quad (13)$$

Im Schema eingezeichnet ist aktuell ein Platzhalterwert. Während der Inbetriebnahme wird der Widerstand je nach Bedarf verändert. Wird dieser beispielsweise verkleinert, so vergrössert sich der Strom durch die Laser-Diode und somit die ausgesandte Lichtleistung.

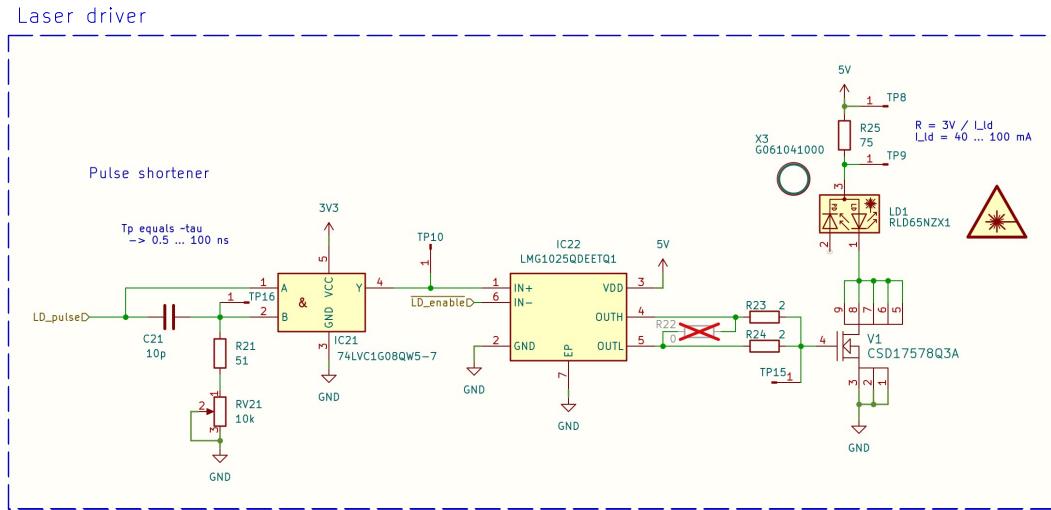


Abbildung 9: Laser Driver

#### 4.2.8 Photo Receiver

Um den Photostrom der Photodiode NJL6401R (JRC, 2014) zu verstärken und in eine Spannung umzuwandeln, wurde mit dem Operationsverstärker OPA858 (TI, 2018) ein Transimpedanzverstärker aufgebaut. Der Ausgang des Transimpedanzverstärkers geht auf den Komparator TLV3501 (TI, 2016c), um das stop-Signal für den TDC zu generieren. Siehe dazu Abbildung 10.

Der Feedback-Widerstand  $R_{26}$  kann je nach bedarf verändert werden. Wird dieser vergrössert, so steigt auch der negative Puls am Ausgang des TIAs. Mit  $C_{20}$  kann zudem bei Bedarf eine kleine Kapazität hinzugefügt werden. Diese hat zum Ziel, ein Schwingen des Verstärkers zu unterdrücken, falls dieser eine Neigung zum Schwingen haben sollte.

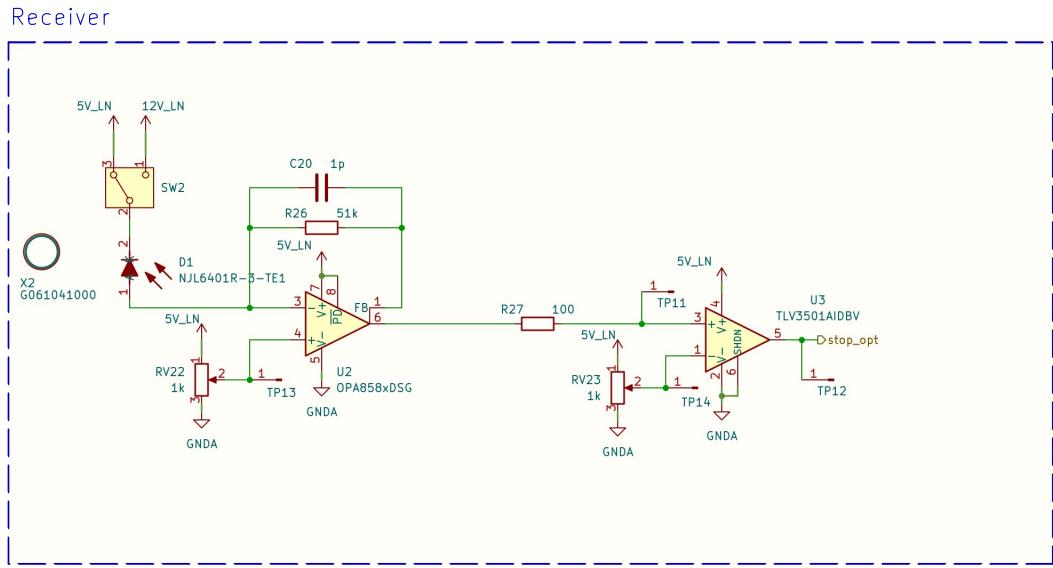


Abbildung 10: Photo Receiver

#### 4.2.9 Decoupling Capacitors

Die Beschaltung der Entkopplungs-Kondensatoren ist in Abbildung 11 dargestellt. Die Kondensatoren für den Operationsverstärker OPA858 und für den Komparator TLV3501 wurden dem Vorschlag im Datenblatt entnommen. Zusätzlich wird auch der Gate-Treiber beim Laser-Treiber mit einer zusätzlichen, grösseren Kapazität gestützt, was sich positiv auf die Stabilität der Speisung auswirken sollte.

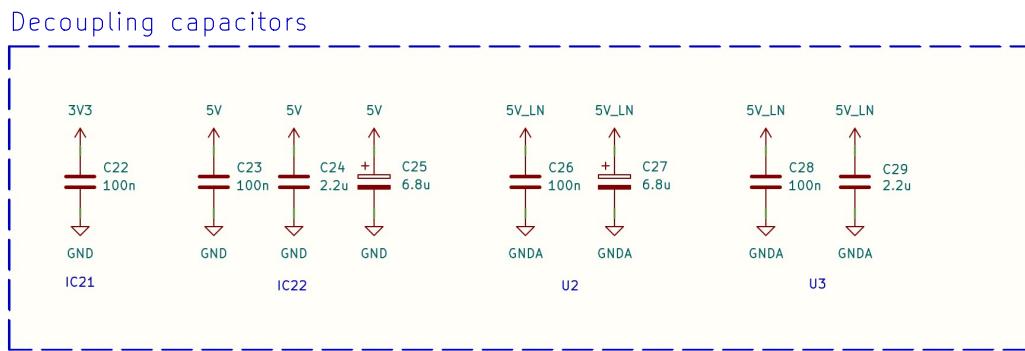


Abbildung 11: Decoupling Capacitors

## 4.3 Layout

In diesem Kapitel werden die PCB-Layouts dokumentiert.

### 4.3.1 Kupfer-Layer

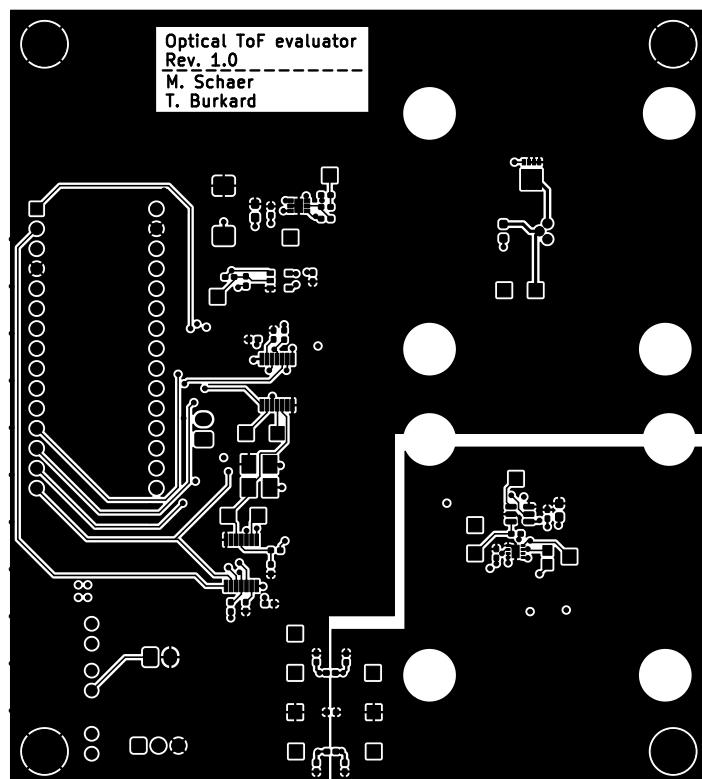


Abbildung 12: PCB Layout Top

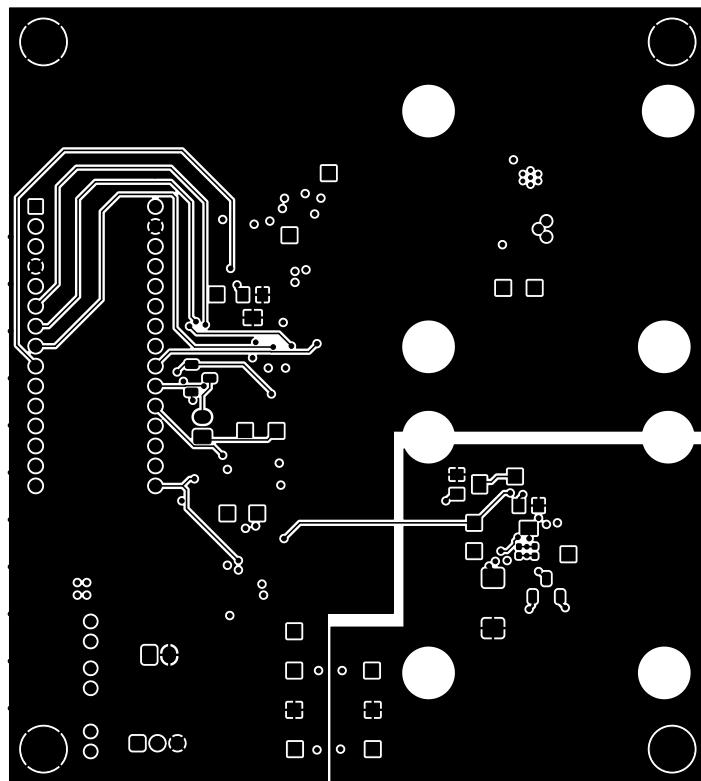


Abbildung 13: PCB Layout Bottom

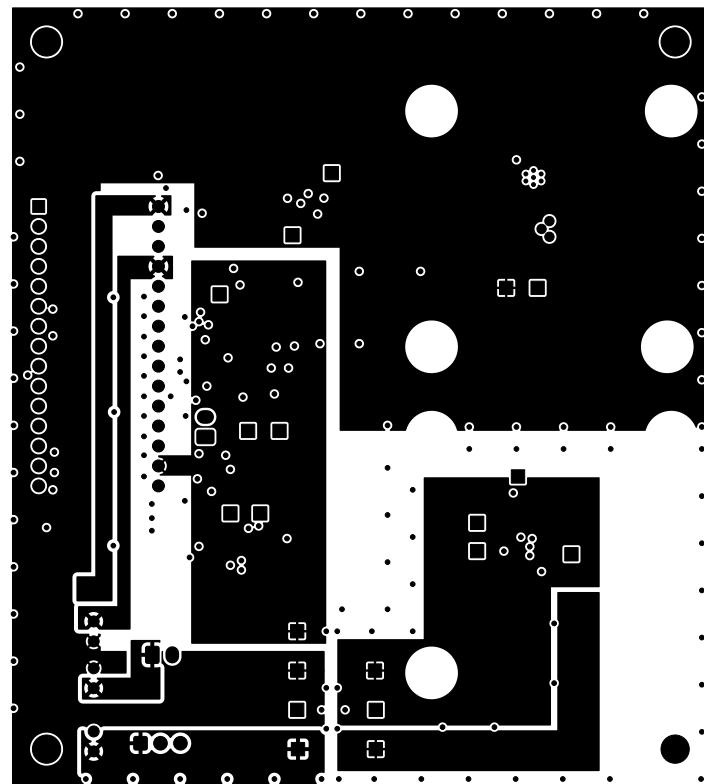


Abbildung 14: PCB Layout Innen 1

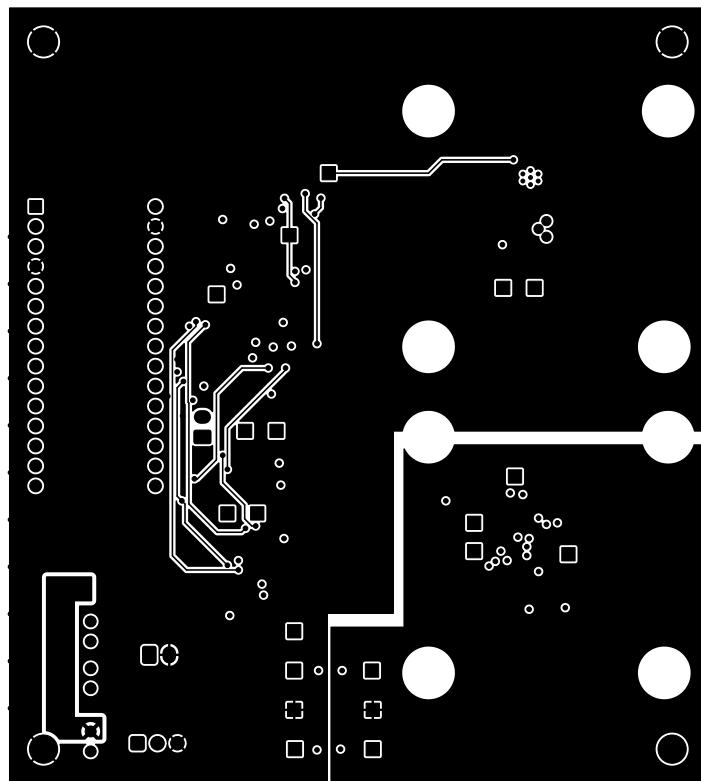


Abbildung 15: PCB Layout Innen 2

#### 4.3.2 Komponenten-Platzierung

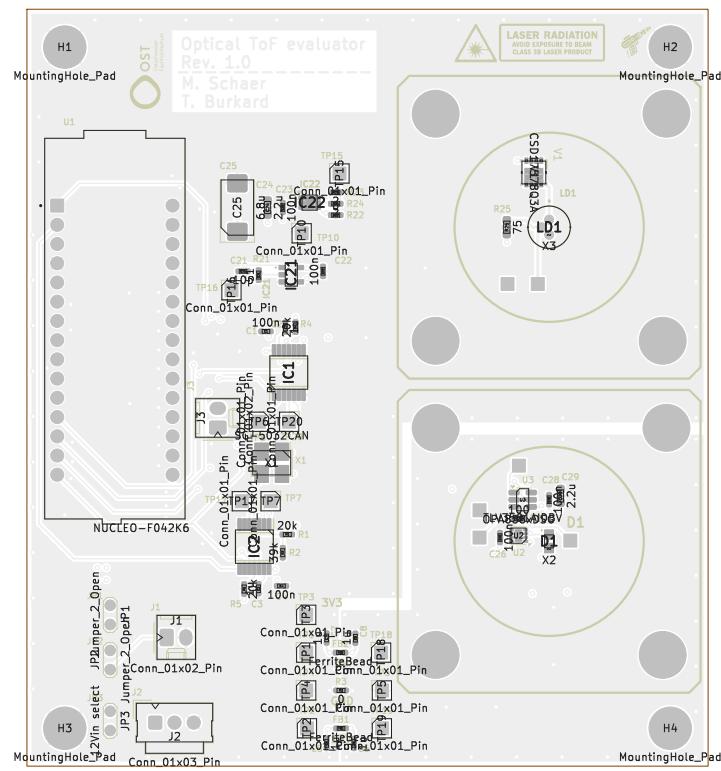


Abbildung 16: PCB Komponenten-Platzierung Top

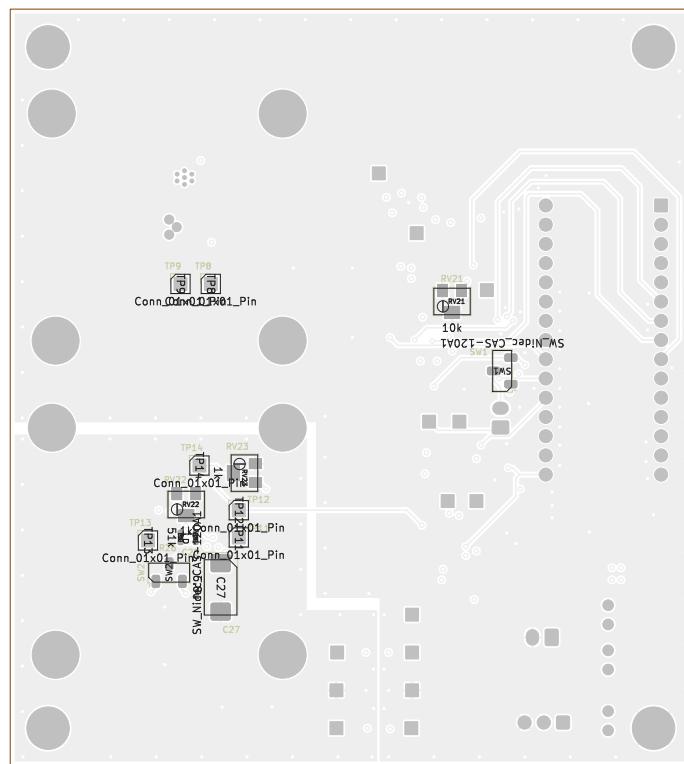


Abbildung 17: PCB Komponenten-Platzierung Bottom

## 4.4 3D View

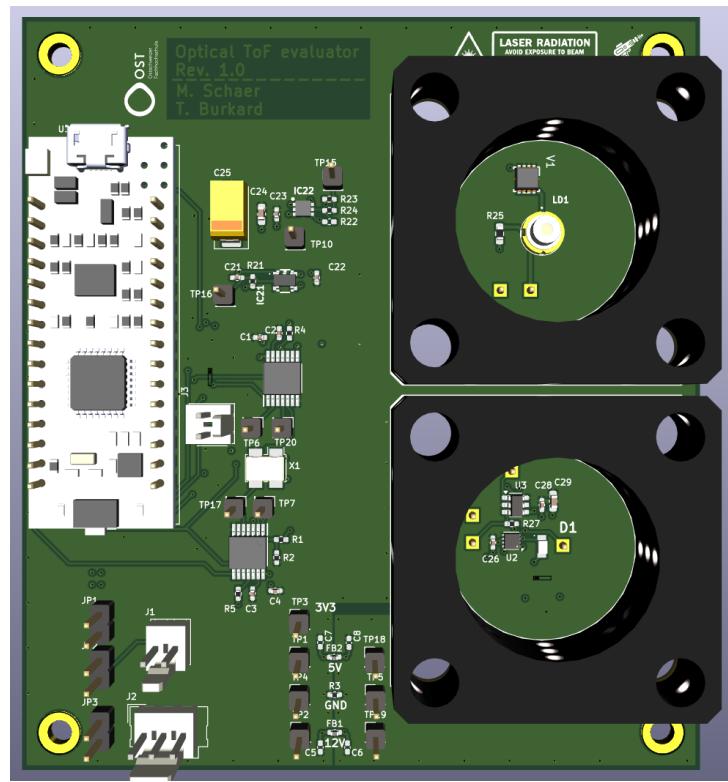


Abbildung 18: 3D View Top

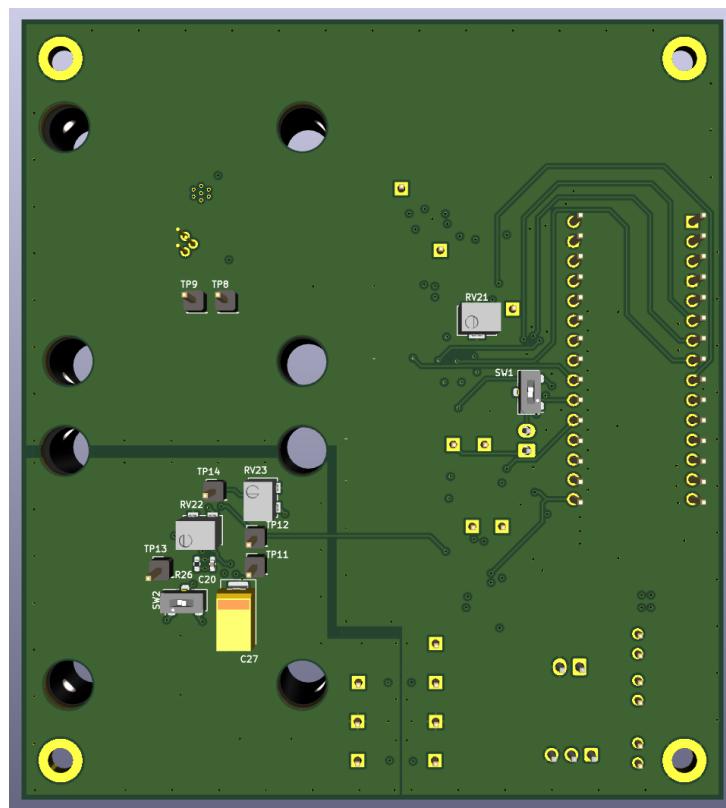


Abbildung 19: 3D View Bottom

## 5 Simulationen

### 5.1 Laser Treiber

### 5.2 Transimpedanzverstärker

## 6 Messungen

In diesem Kapitel werden die Messresultate dokumentiert.

Die verwendeten Python-Skripte zur Berechnung der statistischen Größen und zum Plotten der Diagramme befinden sich im Anhang Kapitel 8.2.

Das verwendete Digital Storage Oscilloscope (DSO) ist ein Rohde & Schwarz RTB2004 1.25 GSa/s.

### 6.1 Elektrische Messungen

In diesem Teilkapitel werden die Messresultate dokumentiert, welche rein elektrisch (also ohne optischen Teil) erfasst wurde.

Die Zeitmessungen werden von `IC1` (siehe Abbildung 5) durchgeführt und von der Firmware, welche auf dem Nucleo Board `U1` (siehe Abbildung 4) getriggert und ausgelesen.

#### 6.1.1 GPIO Toggle mit HAL

Als erstes wird gemessen, wie lange es für die CPU der MCU dauert mittels Hardware Abstraction Layer (HAL) - Library (ST, 2020) zwei GPIO-Pins zu schalten.

In Code 1 ist die Firmware-Implementation dazu gezeigt.

```
1 // Configure TDC
2
3 TDC_init(&tdc_ele, &hspi1, SPI_CS_ELE_GPIO_Port, SPI_CS_ELE_Pin,
4           TDC_ELE_ENABLE_GPIO_Port, TDC_ELE_ENABLE_Pin);
5 TDC_enable(&tdc_ele);
6
7 TDC_read(&tdc_ele, TDC_ADR_CONFIG1, data);
8 data[0] |= TDC_CONFIG1_MEAS_MODE_2;
9
10 TDC_write(&tdc_ele, TDC_ADR_CONFIG1, data);
11
12 while (1) {
13     TDC_start(&tdc_ele);
14
15     // Set Pins to High with HAL
16     HAL_GPIO_WritePin(START_ELE_GPIO_Port, START_ELE_Pin, GPIO_PIN_SET);
17     HAL_GPIO_WritePin(STOP_ELE_GPIO_Port, STOP_ELE_Pin, GPIO_PIN_SET);
18
19     TDC_read_result(&tdc_ele, &tof_fs);
20
21     sprintf(string, "ToF = %lu [ps]\n", tof_fs / 1000);
22     HAL_UART_Transmit(&huart2, (uint8_t *)string, strlen(string), 10000);
23
24     // Set Pins to Low with HAL (preparation for next iteration)
25     HAL_GPIO_WritePin(START_ELE_GPIO_Port, START_ELE_Pin, GPIO_PIN_RESET);
26     HAL_GPIO_WritePin(STOP_ELE_GPIO_Port, STOP_ELE_Pin, GPIO_PIN_RESET);
27
28     HAL_Delay(10); // 10 ms
29 }
```

Code 1: GPIO Toggle mit HAL

Der TDC misst also die Zeit zwischen Zeile 15 und 16 in Code 1. Dazu wird der STOP-Pin des TDC via `sw1` mit `stop_ele` verbunden. Der Kabelanschluss `j3` wird kurzgeschlossen. Siehe Abbildung 5.

Via UART wurden 2000 Messwerte erfasst. Ein Ausschnitt davon ist in Code 2 gezeigt. Die restlichen Daten befinden sich im elektronischen Anhang.

```

1 ToF = 6375779 [ps]
2 ToF = 6374666 [ps]
3 ToF = 6377003 [ps]
4 ToF = 6375333 [ps]
5 ToF = 6377061 [ps]
6 ToF = 6374721 [ps]
7 ToF = 6377504 [ps]
8 ToF = 6374888 [ps]
9 ToF = 6376725 [ps]
10 ToF = 6377005 [ps]
11 ...

```

Code 2: GPIO Toggle mit HAL

Arithmetischer Mittelwert und Standardabweichung sind in Formel 14 aufgeführt.

$$\overline{ToF} = 6'375'887.9 \text{ ps} \quad (14)$$

$$\sigma = 1'059.2 \text{ ps}$$

Da die CPU mit 8 MHz läuft, lässt sich daraus schliessen, dass ein Pin-Toggle mit HAL ca. 50 CPU-Cycles benötigt. Dies erscheint plausibel.

Histogramm und Boxplot sind in Abbildung 20 bzw. 21 dargestellt.

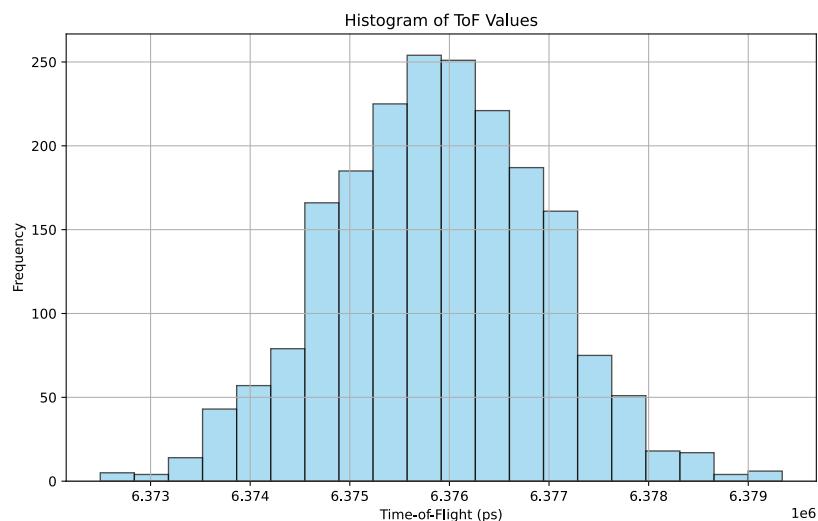


Abbildung 20: GPIO Toggle mit HAL - Histogramm

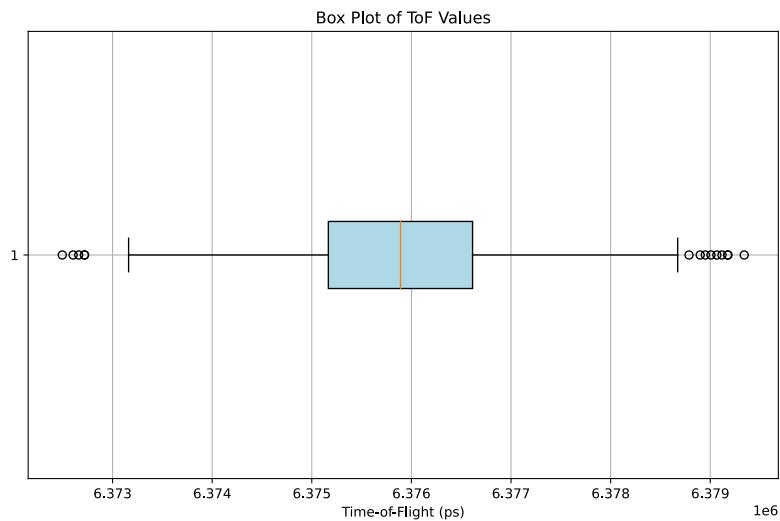


Abbildung 21: GPIO Toggle mit HAL - Boxplot

Um die Resultate des TDC zu validieren, wurde dieselbe Messung auch mittels Digital Storage Oscilloscope (DSO) durchgeführt. Die Messungen sind in Abbildung 22 und 23 dargestellt.



Abbildung 22: GPIO Toggle mit HAL - DSO



Abbildung 23: GPIO Toggle mit HAL - DSO (Zoom)

### 6.1.2 GPIO Toggle ohne HAL

Als nächstes wird gemessen wie lange es für die CPU der MCU dauert mit direktem Register-Zugriff (via Pointer; ohne HAL-Library) zwei GPIO-Pins zu schalten.

In Code 3 ist die Firmware-Implementation dazu gezeigt.

```

1 // Configure TDC
2
3 TDC_init(&tdc_ele, &hspi1, SPI_CS_ELE_GPIO_Port, SPI_CS_ELE_Pin,
4   TDC_ELE_ENABLE_GPIO_Port, TDC_ELE_ENABLE_Pin);
5 TDC_enable(&tdc_ele);
6
7 TDC_read(&tdc_ele, TDC_ADR_CONFIG1, data);
8 data[0] |= TDC_CONFIG1_MEAS_MODE_2;
9
10 TDC_write(&tdc_ele, TDC_ADR_CONFIG1, data);
11
12 while (1) {
13   TDC_start(&tdc_ele);
14
15   // Set Pins to High with direct register access
16   *((volatile uint32_t*)(GPIOA_BASE + 0x14)) |= (1 << 8); // START_ELE
17   *((volatile uint32_t*)(GPIOA_BASE + 0x14)) |= (1 << 11); // STOP_ELE
18
19   TDC_read_result(&tdc_ele, &tof_fs);
20
21   sprintf(string, "ToF = %lu [ps]\n", tof_fs / 1000);
22   HAL_UART_Transmit(&huart2, (uint8_t *)string, strlen(string), 10000);
23
24   // Set Pins to Low with direct register access (preparation for next
25   // iteration)
26   *((volatile uint32_t*)(GPIOA_BASE + 0x14)) &= ~(1 << 8); // START_ELE
27   *((volatile uint32_t*)(GPIOA_BASE + 0x14)) &= ~(1 << 11); // STOP_ELE
28
29   HAL_Delay(10); // 10 ms

```

### Code 3: GPIO Toggle ohne HAL

Der TDC misst also die Zeit zwischen Zeile 15 und 16 in Code 3. Dazu wird, wie in Kapitel 6.1.1, der STOP-Pin des TDC via `sw1` mit `stop_ele` verbunden. Der Kabelanschluss `j3` wird kurzgeschlossen. Siehe Abbildung 5.

Via UART wurden 2000 Messwerte erfasst. Ein Ausschnitt davon ist in Code 4 gezeigt. Die restlichen Daten befinden sich im elektronischen Anhang.

```

1 ToF = 1377894 [ps]
2 ToF = 1378450 [ps]
3 ToF = 1377615 [ps]
4 ToF = 1377840 [ps]
5 ToF = 1377729 [ps]
6 ToF = 1377615 [ps]
7 ToF = 1377337 [ps]
8 ToF = 1378063 [ps]
9 ToF = 1377949 [ps]
10 ToF = 1377615 [ps]
11 ...

```

### Code 4: GPIO Toggle ohne HAL

Arithmetischer Mittelwert und Standardabweichung sind in Formel 15 aufgeführt.

$$\overline{ToF} = 1'377'772.6 \text{ ps} \quad (15)$$

$$\sigma = 401.6 \text{ ps}$$

Da die CPU mit 8 MHz läuft, lässt sich daraus schliessen, dass ein Pin-Toggle ohne HAL ca. 10 CPU-Cycles benötigt. Dies erscheint plausibel.

Histogramm und Boxplot sind in Abbildung 24 bzw. 25 dargestellt.

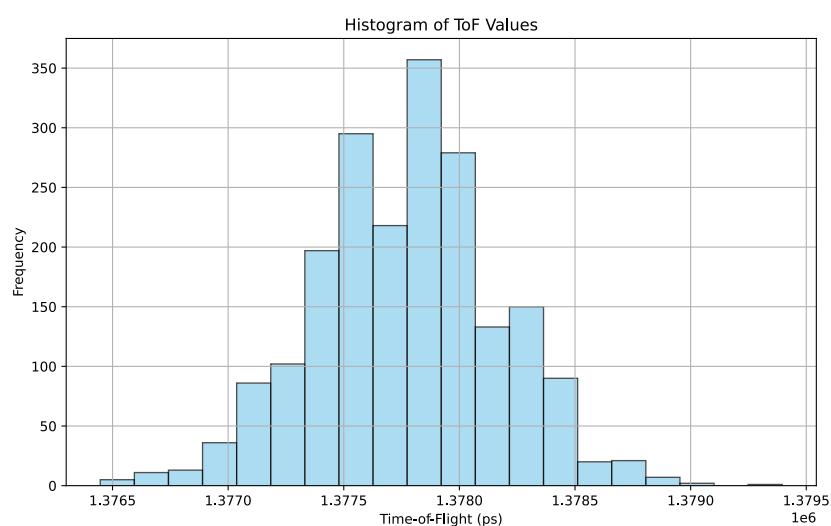


Abbildung 24: GPIO Toggle ohne HAL - Histogramm

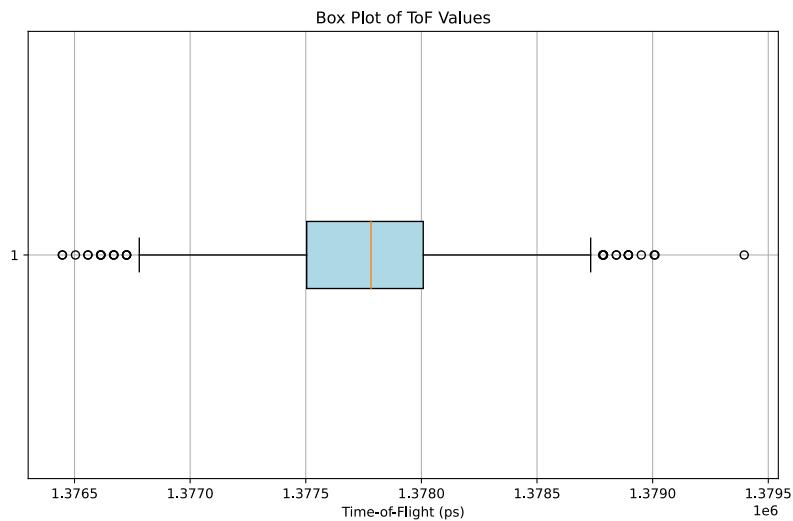


Abbildung 25: GPIO Toggle ohne HAL - Boxplot

Um die Resultate des TDC zu validieren, wurde dieselbe Messung auch mittels Digital Storage Oscilloscope (DSO) durchgeführt. Die Messungen sind in Abbildung 26 und 27 dargestellt.



Abbildung 26: GPIO Toggle ohne HAL - DSO



Abbildung 27: GPIO Toggle ohne HAL - DSO (Zoom)

### 6.1.3 Unterschiedliche Kabellängen

Für diese Messung wird dasselbe Setup wie in Kapitel 6.1.2 verwendet.

Anstelle eines Kurzschlusses von J3 (siehe Abbildung 5) werden nun verschiedene Kabellängen angeschlossen.

Es hat sich herausgestellt, dass eine kreisförmige Anordnung des Kabels wichtig ist. Denn bei einer Überlappung der beiden Kabelenden werden kurze Zeiten gemessen. Dies hat mit der kapazitiven Kopplung zwischen den Leitern zu tun.

Die Resultate sind in Abbildung 28 dargestellt. Die Liste mit den Datenpunkten befindet sich im elektronischen Anhang.

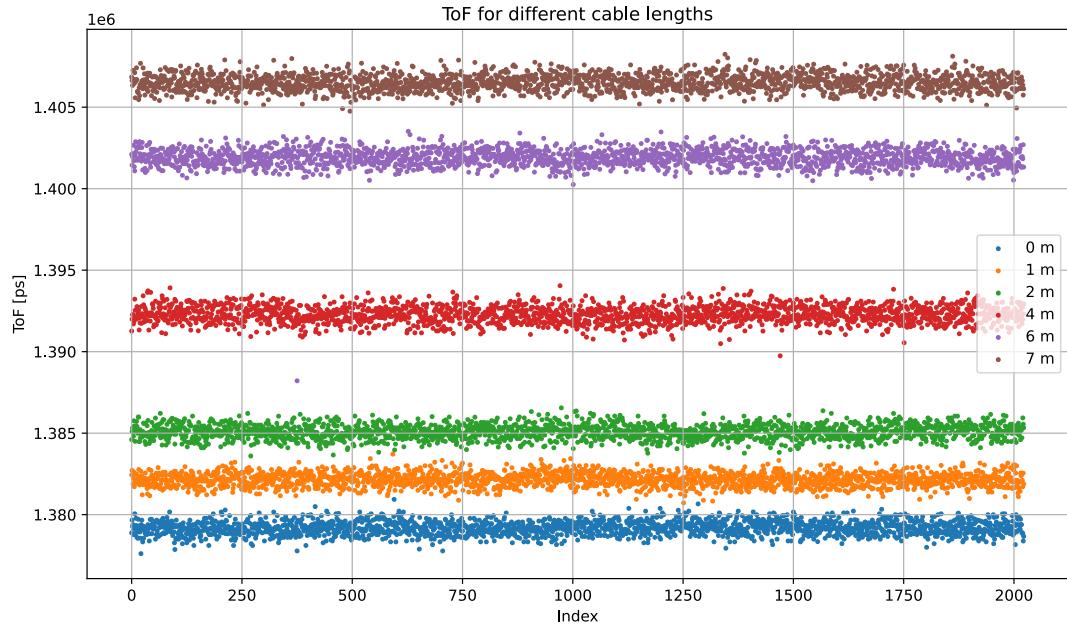


Abbildung 28: Unterschiedliche Kabellängen

Die arithmetischen Mittelwerte und Standardabweichungen sind in Tabelle 1 aufgeführt.

Länge	Mittelwert	Standardabweichung
0 m	1'379'188.1 ps	421.3 ps
1 m	1'382'152.1 ps	413.0 ps
2 m	1'385'074.0 ps	436.6 ps
4 m	1'392'274.9 ps	518.8 ps
6 m	1'401'903.2 ps	577.1 ps
7 m	1'406'512.4 ps	490.8 ps

Tabelle 1: Unterschiedliche Kabellängen

Die Signal-Ausbreitungsgeschwindigkeit in Kupfer beträgt ca. 2/3 der Lichtgeschwindigkeit (firewall.xc, 2025). Um die Resultate in Tabelle 1 zu validieren, rechnen wir wie in Formel 16 gezeigt, auf die Kabellänge zurück. Die Laufzeit bei 0 m wird dabei abgezogen, um die Verzögerung zu kompensieren, welche durch das Schalten der GPIOs entsteht.

$$c_{cu} \approx \frac{2}{3} \cdot c_0 = \frac{2}{3} \cdot 299'792'458 \frac{m}{s} \approx 200'000'000 \frac{m}{s}$$

$$ToF_n = ToF_{n_{abs}} - ToF_0$$

$$l_n = ToF_n \cdot c_{cu}$$
(16)

Die Resultate sind in Tabelle 2 dargestellt.

Tatsächliche Länge	ToF_n	Zurückgerechnete Länge
0 m	0 ps	0.6 m
1 m	2'964.0 ps	0.6 m
2 m	5'885.9 ps	1.2 m
4 m	13'086.8 ps	2.6 m
6 m	22'715.1 ps	4.5 m
7 m	27'324.3 ps	5.5 m

Tabelle 2: Kabellängen zurückgerechnet

Es fällt auf, dass die Resultate nicht genau übereinstimmen. Dies hat mehrere Ursachen: Die Ausbreitungsgeschwindigkeit ist nicht genau bekannt und die tatsächlichen Kabellängen wurden nicht genau gemessen.

Es ist jedoch eine klare Korrelation zu erkennen.

#### 6.1.4 Mode 1 vs. Mode 2

Der TDC7200 unterstützt zwei Modi mit unterschiedlichen Messbereichen (TI, 2016b):

- Mode 1: 12 ns bis 500 ns
- Mode 2: 250 ns bis 8 ms

Im Mode 1 wird nur der interne Ring-Oszillator des TDC verwendet. Siehe dazu Abbildung 29.

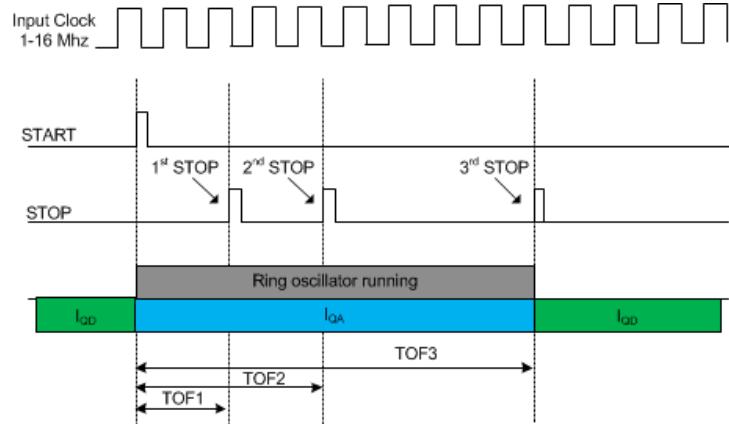


Abbildung 29: TDC Mode 1 (TI, 2016b)

Im Mode 2, um längere Zeiten messen zu können, wird zusätzlich der externe Clock des TDC verwendet. Siehe dazu Abbildung 30.

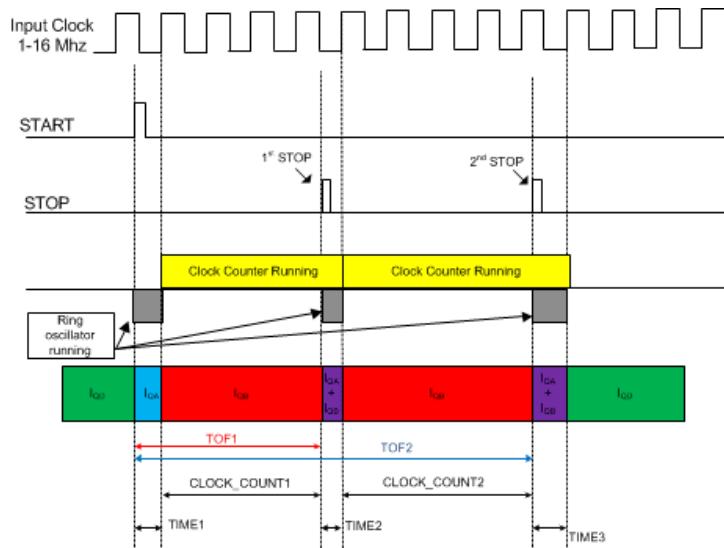


Abbildung 30: TDC Mode 2 (TI, 2016b)

Die bisherigen Messungen (Kapitel 6.1.1, 6.1.2 und 6.1.3) wurden im Mode 2 durchgeführt, da das Schalten der GPIOs mit der CPU mehr als 500 ns brauchte.

In künftigen Messungen soll das Schalten der GPIOs von einem Hardware-Timer der MCU erledigt werden. Damit werden Schaltzeiten von 125 ns bei 8 MHz bzw. 20.8 ns bei 48 MHz möglich sein. Es soll deshalb in diesem Kapitel ein Vergleich der Messresultate der beiden Modi gemacht werden.

Dazu wurden drei Messungen gemacht:

1. GPIO Toggle ohne HAL im Mode 2 mit Kabellänge = 0 m (wie in Kapitel 6.1.2 und 6.1.3)
2. GPIO Toggle ohne HAL im Mode 2 mit Kabellänge = 6 m (wie in Kapitel 6.1.3)
3. Ohne GPIO Toggle Im Mode 1 mit Kabellänge = 6 m

Für die Messung im Mode 1 soll auf die Verzögerung durch das Schalten der GPIOs verzichtet werden. Dazu wird das Start- und Stop-Signal vom selben GPIO-Pin, `START_ELE`, generiert. Dazu wird `SW1` mit `start_ele` verbunden (siehe Abbildung 5).

In Code 5 ist die Firmware-Implementation für eine Messung im Mode 1 gezeigt.

```

1 // Configure TDC
2
3 TDC_init(&tdc_ele, &hspi1, SPI_CS_ELE_GPIO_Port, SPI_CS_ELE_Pin,
           TDC_ELE_ENABLE_GPIO_Port, TDC_ELE_ENABLE_Pin);
4 TDC_enable(&tdc_ele);
5
6 TDC_read(&tdc_ele, TDC_ADR_CONFIG1, data);
7 data[0] |= TDC_CONFIG1_MEAS_MODE_1;
8
9 TDC_write(&tdc_ele, TDC_ADR_CONFIG1, data);
10
11 while (1) {
12     TDC_start(&tdc_ele);
13
14     // Set Pin to High
15     *((volatile uint32_t*)(GPIOA_BASE + 0x14)) |= (1 << 8); // START_ELE

```

```

16
17     TDC_read_result(&tdc_ele, &tof_fs);
18
19     sprintf(string, "ToF = %lu [ps]\n", tof_fs / 1000);
20     HAL_UART_Transmit(&huart2, (uint8_t *)string, strlen(string), 10000);
21
22     // Set Pin to Low (preparation for next iteration)
23     *((volatile uint32_t*)(GPIOA_BASE + 0x14)) &= ~(1 << 8); // START_ELE
24
25     HAL_Delay(10); // 10 ms
26 }
```

Code 5: Mode 1

Die Unterschiede im Vergleich zu den Messungen in Kapitel 6.1.2 und 6.1.3 sind:

- Zeile 7: TDC wird im Mode 1 konfiguriert (anstatt Mode 2)
- Zeile 15 und 23: Es wird nur der START\_ELE Pin getoggelt (anstatt START\_ELE und STOP\_ELE Pin)

Die Erwartung ist, dass die Differenz aus Messung 1 und Messungen 2 ungefähr dem Resultat aus Messung 3 entspricht.

Die Resultate dieser drei Messungen sind in Tabelle 3 aufgeführt. Die restlichen Daten befinden sich im elektronischen Anhang.

Messung	Mittelwert	Standardabweichung
1	1'378'222.5 ps	406.9 ps
2	1'403'223.8 ps	541.5 ps
3	25'145.3 ps	331.1 ps

Tabelle 3: Mode 1 vs. Mode 2

Die Berechnung in Formel 17 zeigt, dass die Messresultate nahe beieinander liegen.

$$\begin{aligned} \Delta \text{Mode 2} &= 1'403'223.8 \text{ ps} - 1'378'222.5 \text{ ps} = 25'001.3 \text{ ps} \\ \text{Mode 1} &= 25'145.3 \text{ ps} \end{aligned} \quad (17)$$

Es kann also davon ausgegangen werden, dass Mode 1 und Mode 2 im Firmware-Treiber korrekt implementiert wurden.

Bemerkung: Im Firmware-Treiber (siehe Anhang Kapitel 8.1) kann für beide Modi dieselbe Funktion `TDC_read_result()` verwendet werden. Für Mode 1 vereinfacht sich die Berechnung, weil `TIME2 = 0` und `CLOCK_COUNT1 = 0`.

### 6.1.5 Timer Output

## 6.2 Optische Messungen

## 7 Fazit

## 8 Anhang

### 8.1 TDC Treiber

In Code 6 und 7 ist der selbst entwickelte Firmware-Treiber für den TDC dargestellt.

```

1  /*
2   * TDC.h
3   *
4   * Driver for TI TDC7200 -- Header file
5   *
6   * (C) T. Burkard | M. Schaeer
7   *
8   */
9
10 #ifndef TDC_H_
11 #define TDC_H_
12
13 #include <stdint.h>
14 #include "stm32f0xx_hal.h"
15
16 #define TDC_CLOCK_PERIOD_FS 62500000 // 16 MHz in [fs]
17
18 /**
19 * Convert 24bit buffer received from SPI to integer
20 */
21 #define TDC_24BIT_BUF_TO_INT(buf) ((buf[0] << 16) + (buf[1] << 8) + buf[2])
22
23 /**
24 * @brief Register addresses of the TDC
25 *
26 */
27 typedef enum {
28     TDC_ADR_CONFIG1          = 0x00, // Configuration Register 1
29     TDC_ADR_CONFIG2          = 0x01, // Configuration Register 2
30     TDC_ADR_INT_STATUS       = 0x02, // Interrupt Status Register
31     TDC_ADR_INT_MASK         = 0x03, // Interrupt Mask Register
32     TDC_ADR_COARSE_CNTR_OVF_H = 0x04, // Coarse Counter Overflow Value High
33     TDC_ADR_COARSE_CNTR_OVF_L = 0x05, // Coarse Counter Overflow Value Low
34     TDC_ADR_CLOCK_CNTR_OVF_H = 0x06, // CLOCK Counter Overflow Value High
35     TDC_ADR_CLOCK_CNTR_OVF_L = 0x07, // CLOCK Counter Overflow Value Low
36     TDC_ADR_CLOCK_CNTR_STOP_MASK_H = 0x08, // CLOCK Counter STOP Mask High
37     TDC_ADR_CLOCK_CNTR_STOP_MASK_L = 0x09, // CLOCK Counter STOP Mask Low
38     TDC_ADR_TIME1             = 0x10, // Measured Time 1
39     TDC_ADR_CLOCK_COUNT1     = 0x11, // CLOCK Counter Value
40     TDC_ADR_TIME2             = 0x12, // Measured Time 2
41     TDC_ADR_CLOCK_COUNT2     = 0x13, // CLOCK Counter Value
42     TDC_ADR_TIME3             = 0x14, // Measured Time 3
43     TDC_ADR_CLOCK_COUNT3     = 0x15, // CLOCK Counter Value
44     TDC_ADR_TIME4             = 0x16, // Measured Time 4
45     TDC_ADR_CLOCK_COUNT4     = 0x17, // CLOCK Counter Value
46     TDC_ADR_TIME5             = 0x18, // Measured Time 5
47     TDC_ADR_CLOCK_COUNT5     = 0x19, // CLOCK Counter Value
48     TDC_ADR_TIME6             = 0x1A, // Measured Time 6
49     TDC_ADR_CALIBRATION1     = 0x1B, // Calibration 1, 1 CLOCK Period
50     TDC_ADR_CALIBRATION2     = 0x1C, // Calibration 2, 2/10/20/40 CLOCK
51     Periods
52     TDC_ADR_AMOUNT // Amount of registers
53 } TDC_adr_t;
```

```

54 /**
55 * @brief Register sizes of the TDC
56 *
57 * Index = Register address (TDC_adr_t)
58 * Value = Register size (in bytes)
59 *
60 */
61 extern const uint8_t TDC_REG_SIZE[TDC_ADR_AMOUNT];
62
63 /**
64 * @brief Bit description of the TDC CONFIG1 register
65 *
66 */
67 typedef enum {
68     TDC_CONFIG1_FORCE_CAL_OFF    = (0b0 << 7), // Calibration is not performed
69     after interrupted measurement (for example, due to counter overflow or
70     missing STOP signal)
71     TDC_CONFIG1_FORCE_CAL_ON    = (0b1 << 7), // Calibration is always
72     performed at the end (for example, after a counter overflow)
73     TDC_CONFIG1_PARITY_EN_OFF   = (0b0 << 6), // Parity bit for Measurement
74     Result Registers disabled (Parity Bit always 0)
75     TDC_CONFIG1_PARITY_EN_ON    = (0b1 << 6), // Parity bit for Measurement
76     Result Registers enabled (Even Parity)
77     TDC_CONFIG1_TRIGG_EDGE_RISE = (0b0 << 5), // TRIGG is output as a Rising
78     edge signal
79     TDC_CONFIG1_TRIGG_EDGE_FALL = (0b1 << 5), // TRIGG is output as a Falling
80     edge signal
81     TDC_CONFIG1_STOP_EDGE_RISE  = (0b0 << 4), // Measurement is stopped on
82     Rising edge of STOP signal
83     TDC_CONFIG1_STOP_EDGE_FALL  = (0b1 << 4), // Measurement is stopped on
84     Falling edge of STOP signal
85     TDC_CONFIG1_START_EDGE_RISE = (0b0 << 3), // Measurement is started on
86     Rising edge of START signal
87     TDC_CONFIG1_START_EDGE_FALL = (0b1 << 3), // Measurement is started on
88     Falling edge of START signal
89     TDC_CONFIG1_MEAS_MODE_1     = (0b00 << 1), // Measurement Mode 1 (for
90     expected time-of-flight < 500 ns).
91     TDC_CONFIG1_MEAS_MODE_2     = (0b01 << 1), // Measurement Mode 2
92     (recommended)
93     TDC_CONFIG1_START_MEAS      = (0b1 << 0), // Start New Measurement.
94     Writing a 1 will clear all bits in the Interrupt Status Register and Start
95     the measurement (by generating an TRIGG signal) and will reset the content
96     of all Measurement Results registers
97 } TDC_config1_t;
98
99 /**
100 * @brief Bit description of the TDC CONFIG2 register
101 *
102 */
103 typedef enum {
104     TDC_CONFIG2_CALIBRATION2_PERIODS_2    = (0b00 << 6), // Measuring 2 CLOCK
105     periods
106     TDC_CONFIG2_CALIBRATION2_PERIODS_10   = (0b01 << 6), // Measuring 10 CLOCK
107     periods
108     TDC_CONFIG2_CALIBRATION2_PERIODS_20   = (0b10 << 6), // Measuring 20 CLOCK
109     periods
110     TDC_CONFIG2_CALIBRATION2_PERIODS_40   = (0b11 << 6), // Measuring 40 CLOCK
111     periods
112     TDC_CONFIG2_AVG_CYCLES_1              = (0b000 << 3), // 1 Measurement Cycle
113     only (no Multi-Cycle Averaging Mode)
114     TDC_CONFIG2_AVG_CYCLES_2              = (0b001 << 3), // 2 Measurement Cycles

```

```

94     TDC_CONFIG2_AVG_CYCLES_4          = (0b010 << 3), // 4 Measurement Cycles
95     TDC_CONFIG2_AVG_CYCLES_8          = (0b011 << 3), // 8 Measurement Cycles
96     TDC_CONFIG2_AVG_CYCLES_16         = (0b100 << 3), // 16 Measurement Cycles
97     TDC_CONFIG2_AVG_CYCLES_32         = (0b101 << 3), // 32 Measurement Cycles
98     TDC_CONFIG2_AVG_CYCLES_64         = (0b110 << 3), // 64 Measurement Cycles
99     TDC_CONFIG2_AVG_CYCLES_128        = (0b111 << 3), // 128 Measurement
100    Cycles
101    TDC_CONFIG2_NUM_STOP_1           = (0b000 << 0), // Single Stop
102    TDC_CONFIG2_NUM_STOP_2           = (0b001 << 0), // Two Stops
103    TDC_CONFIG2_NUM_STOP_3           = (0b010 << 0), // Three Stops
104    TDC_CONFIG2_NUM_STOP_4           = (0b011 << 0), // Four Stops
105    TDC_CONFIG2_NUM_STOP_5           = (0b100 << 0), // Five Stops
106
107 } TDC_config2_t;
108
109 /**
110 * @brief Bit description of the TDC INT_STATUS register
111 */
112 typedef enum {
113     TDC_STATUS_MEAS_COMPLETE_FLAG   = (1 << 4), // Measurement has completed
114     TDC_STATUS_MEAS_STARTED_FLAG    = (1 << 3), // Measurement has started
115     (START signal received)
116     TDC_STATUS_CLOCK_CNTR_OVF_INT  = (1 << 2), // Clock overflow detected,
117     running measurement will be stopped immediately
118     TDC_STATUS_COARSE_CNTR_OVF_INT = (1 << 1), // Coarse overflow detected,
119     running measurement will be stopped immediately
120     TDC_STATUS_NEW_MEAS_INT        = (1 << 0), // Interrupt detected - New
121     Measurement has been completed
122 } TDC_status_t;
123
124 /**
125 * @brief Bit description of the TDC INT_MASK register
126 */
127 typedef enum {
128     TDC_MASK_CLOCK_CNTR_OVF_MASK_OFF = (0 << 2), // CLOCK Counter Overflow
129     Interrupt disabled
130     TDC_MASK_CLOCK_CNTR_OVF_MASK_ON  = (1 << 2), // CLOCK Counter Overflow
131     Interrupt enabled
132     TDC_MASK_COARSE_CNTR_OVF_MASK_OFF = (0 << 1), // Coarse Counter Overflow
133     Interrupt disabled
134     TDC_MASK_COARSE_CNTR_OVF_MASK_ON  = (1 << 1), // Coarse Counter Overflow
135     Interrupt enabled
136     TDC_MASK_NEW_MEAS_MASK_OFF      = (0 << 0), // New Measurement Interrupt
137     disabled
138     TDC_MASK_NEW_MEAS_MASK_ON       = (1 << 0), // New Measurement Interrupt
139     enabled
140 } TDC_mask_t;
141
142 /**
143 * @brief Error codes of the TDC module
144 */
145 typedef enum {
146     TDC_OK = 0,
147     TDC_ERROR,
148     TDC_WRONG_ADDRESS,
149     TDC_WRONG_SIZE,
150     TDC_COM_ERROR,
151 } TDC_error_t;
152
153

```

```

144 /**
145 * @brief Handle for a TDC instance
146 *
147 */
148 typedef struct {
149     SPI_HandleTypeDef* spi;
150     GPIO_TypeDef* cs_port;
151     uint16_t cs_pin;
152     GPIO_TypeDef* en_port;
153     uint16_t en_pin;
154 } TDC_t;
155
156 /**
157 * @brief Initialize a TDC instance
158 *
159 * Note, SPI and GPIOs need to be initialized manually before calling this
160 * function.
161 *
162 * @param [out] tdc -- Pointer to TDC handle
163 * @param [in] spi -- Pointer to SPI handle
164 * @param [in] cs_port -- GPIO port of Chip Select
165 * @param [in] cs_pin -- GPIO pin of Chip Select
166 * @param [in] en_port -- GPIO port of Enable
167 * @param [in] en_pin -- GPIO pin of Enable
168 *
169 * @return TDC_error -- Error code
170 */
171 TDC_error_t TDC_init(TDC_t* tdc, SPI_HandleTypeDef* spi, GPIO_TypeDef* cs_port,
172                      uint16_t cs_pin, GPIO_TypeDef* en_port, uint16_t en_pin);
173
174 /**
175 * @brief Enable the TDC
176 *
177 * @param [in] tdc -- Pointer to TDC handle
178 *
179 * @return TDC_error_t -- Error code
180 */
181 TDC_error_t TDC_enable(TDC_t* tdc);
182
183 /**
184 * @brief Disable the TDC
185 *
186 * @param [in] tdc -- Pointer to TDC handle
187 *
188 * @return TDC_error_t -- Error code
189 */
190 TDC_error_t TDC_disable(TDC_t* tdc);
191
192 /**
193 * @brief Read register data from TDC
194 *
195 * @param [in] tdc -- Pointer to TDC handle
196 * @param [in] address -- Address of the register(s) to read
197 * @param [out] rx_data -- Pointer to data
198 *
199 * @attention @p rx_data needs to provide space for at least TDC_REG_SIZE[ @p
200 * address ] bytes.
201 *
202 * @return TDC_error_t -- Error code
203 */
204 TDC_error_t TDC_read(TDC_t* tdc, TDC_addr_t address, uint8_t* rx_data);

```

```

202 /**
203 * @brief Write register data to TDC
204 *
205 * @param[in] tdc -- Pointer to TDC handle
206 * @param[in] address -- Address of the register(s) to write
207 * @param[in] tx_data -- Pointer to data
208 *
209 * @attention @p tx_data needs to provide space for at least TDC_REG_SIZE[ @p
210 * address ] bytes.
211 *
212 * @return TDC_error_t -- Error code
213 */
214 TDC_error_t TDC_write(TDC_t* tdc, TDC_addr_t address, uint8_t* tx_data);
215
216 /**
217 * @brief Start measurement of TDC
218 *
219 * @param[in] tdc -- Pointer to TDC handle
220 *
221 * @return TDC_error_t -- Error code
222 */
223 TDC_error_t TDC_start(TDC_t* tdc);
224
225 /**
226 * @brief Read ToF measurement result from TDC
227 *
228 * @param[in] tdc -- Pointer to TDC handle
229 * @param[out] tof_fs -- ToF Measurement result [fs]
230 *
231 * @return TDC_error_t -- Error code
232 */
233 TDC_error_t TDC_read_result(TDC_t* tdc, uint64_t* tof_fs);
234
235 #endif /* TDC_H_ */

```

Code 6: TDC Driver (Header)

```

1 /*
2 * TDC.c
3 *
4 * Driver for TI TDC7200 -- Source file
5 *
6 * (C) T. Burkard | M. Schaeer
7 *
8 */
9
10 #include "TDC/TDC.h"
11 #include <stdbool.h>
12
13 #define TDC_AUTO_INC_OFF (0 << 7)
14 #define TDC_AUTO_INC_ON (1 << 7)
15
16 #define TDC_RW_READ (0 << 6)
17 #define TDC_RW_WRITE (1 << 6)
18
19 #define TDC_CMD_READ (TDC_AUTO_INC_ON | TDC_RW_READ)
20 #define TDC_CMD_WRITE (TDC_AUTO_INC_ON | TDC_RW_WRITE)
21
22 const uint8_t TDC_REG_SIZE[TDC_ADR_AMOUNT] = {
23     // Register Address           Register Size [bytes]

```

```
24     [TDC_ADR_CONFIG1]          = 1,
25     [TDC_ADR_CONFIG2]          = 1,
26     [TDC_ADR_INT_STATUS]       = 1,
27     [TDC_ADR_INT_MASK]         = 1,
28     [TDC_ADR_COARSE_CNTR_OVF_H] = 1,
29     [TDC_ADR_COARSE_CNTR_OVF_L] = 1,
30     [TDC_ADR_CLOCK_CNTR_OVF_H] = 1,
31     [TDC_ADR_CLOCK_CNTR_OVF_L] = 1,
32     [TDC_ADR_CLOCK_CNTR_STOP_MASK_H] = 1,
33     [TDC_ADR_CLOCK_CNTR_STOP_MASK_L] = 1,
34     [TDC_ADR_TIME1]            = 3,
35     [TDC_ADR_CLOCK_COUNT1]      = 3,
36     [TDC_ADR_TIME2]            = 3,
37     [TDC_ADR_CLOCK_COUNT2]      = 3,
38     [TDC_ADR_TIME3]            = 3,
39     [TDC_ADR_CLOCK_COUNT3]      = 3,
40     [TDC_ADR_TIME4]            = 3,
41     [TDC_ADR_CLOCK_COUNT4]      = 3,
42     [TDC_ADR_TIME5]            = 3,
43     [TDC_ADR_CLOCK_COUNT5]      = 3,
44     [TDC_ADR_TIME6]            = 3,
45     [TDC_ADR_CALIBRATION1]      = 3,
46     [TDC_ADR_CALIBRATION2]      = 3,
47 };
48
49 static TDC_error_t send(TDC_t* tdc, TDC_addr_t address, uint8_t* data, bool
50                         write);
51
52 TDC_error_t TDC_init(TDC_t* tdc, SPI_HandleTypeDef* spi, GPIO_TypeDef* cs_port,
53                      uint16_t cs_pin, GPIO_TypeDef* en_port, uint16_t en_pin)
54 {
55     if ((tdc == NULL) || (spi == NULL) || (cs_port == NULL) || (en_port ==
56         NULL)) {
57         return TDC_ERROR;
58     }
59
60     tdc->spi      = spi;
61     tdc->cs_port  = cs_port;
62     tdc->cs_pin   = cs_pin;
63     tdc->en_port  = en_port;
64     tdc->en_pin   = en_pin;
65
66     return TDC_OK;
67 }
68
69 TDC_error_t TDC_enable(TDC_t* tdc)
70 {
71     if (tdc == NULL) {
72         return TDC_ERROR;
73     }
74
75     HAL_GPIO_WritePin(tdc->en_port, tdc->en_pin, GPIO_PIN_SET);
76
77     return TDC_OK;
78 }
79
80 TDC_error_t TDC_disable(TDC_t* tdc)
81 {
82     if (tdc == NULL) {
83         return TDC_ERROR;
84     }
```

```
82     HAL_GPIO_WritePin(tdc->en_port, tdc->en_pin, GPIO_PIN_RESET);
83
84     return TDC_OK;
85 }
86
87
88 TDC_error_t TDC_write(TDC_t* tdc, TDC_addr_t address, uint8_t* tx_data)
89 {
90     return send(tdc, address, tx_data, true);
91 }
92
93 TDC_error_t TDC_read(TDC_t* tdc, TDC_addr_t address, uint8_t* rx_data)
94 {
95     return send(tdc, address, rx_data, false);
96 }
97
98 TDC_error_t TDC_start(TDC_t* tdc)
99 {
100     uint8_t      data[TDC_REG_SIZE[TDC_ADR_CONFIG1]];
101     TDC_error_t error_code = TDC_OK;
102
103     if ((error_code = TDC_read(tdc, TDC_ADR_CONFIG1, data)) != TDC_OK) {
104         return error_code;
105     }
106
107     data[0] |= TDC_CONFIG1_START_MEAS;
108
109     return TDC_write(tdc, TDC_ADR_CONFIG1, data);
110 }
111
112 TDC_error_t TDC_read_result(TDC_t* tdc, uint64_t* tof_fs)
113 {
114     uint8_t time1_buf[TDC_REG_SIZE[TDC_ADR_TIME1]];
115     uint8_t time2_buf[TDC_REG_SIZE[TDC_ADR_TIME2]];
116     uint8_t clock_count1_buf[TDC_REG_SIZE[TDC_ADR_CLOCK_COUNT1]];
117     uint8_t calibration1_buf[TDC_REG_SIZE[TDC_ADR_CALIBRATION1]];
118     uint8_t calibration2_buf[TDC_REG_SIZE[TDC_ADR_CALIBRATION2]];
119     uint8_t config2_buf[TDC_REG_SIZE[TDC_ADR_CONFIG2]];
120
121     uint32_t time1;
122     uint32_t time2;
123     uint32_t clock_count1;
124     uint32_t calibration1;
125     uint32_t calibration2;
126     uint32_t calibration2_periods;
127
128     TDC_error_t error_code = TDC_OK;
129
130     // Read registers
131
132     if ((error_code = TDC_read(tdc, TDC_ADR_TIME1, time1_buf)) != TDC_OK) {
133         return error_code;
134     }
135
136     time1 = TDC_24BIT_BUF_TO_INT(time1_buf);
137
138     if ((error_code = TDC_read(tdc, TDC_ADR_TIME2, time2_buf)) != TDC_OK) {
139         return error_code;
140     }
141
142     time2 = TDC_24BIT_BUF_TO_INT(time2_buf);
```

```

143     if ((error_code = TDC_read(tdc, TDC_ADR_CLOCK_COUNT1, clock_count1_buf)) !=
144         TDC_OK) {
145         return error_code;
146     }
147
148     clock_count1 = TDC_24BIT_BUF_TO_INT(clock_count1_buf);
149
150     if ((error_code = TDC_read(tdc, TDC_ADR_CALIBRATION1, calibration1_buf)) !=
151         TDC_OK) {
152         return error_code;
153     }
154
155     calibration1 = TDC_24BIT_BUF_TO_INT(calibration1_buf);
156
157     if ((error_code = TDC_read(tdc, TDC_ADR_CALIBRATION2, calibration2_buf)) !=
158         TDC_OK) {
159         return error_code;
160     }
161
162     calibration2 = TDC_24BIT_BUF_TO_INT(calibration2_buf);
163
164     if ((error_code = TDC_read(tdc, TDC_ADR_CONFIG2, config2_buf)) != TDC_OK) {
165         return error_code;
166     }
167
168     if (config2_buf[0] & TDC_CONFIG2_CALIBRATION2_PERIODS_10) {
169         calibration2_periods = 10;
170     } else if (config2_buf[0] & TDC_CONFIG2_CALIBRATION2_PERIODS_20) {
171         calibration2_periods = 20;
172     } else if (config2_buf[0] & TDC_CONFIG2_CALIBRATION2_PERIODS_40) {
173         calibration2_periods = 40;
174     } else { // TDC_CONFIG2_CALIBRATION2_PERIODS_2
175         calibration2_periods = 2;
176     }
177
178     // Calculations
179
180     double cal_count = (double)(calibration2 - calibration1) /
181     (calibration2_periods - 1);
182     uint64_t norm_lsb_fs = (uint64_t)(TDC_CLOCK_PERIOD_FS / cal_count);
183
184     *tof_fs = (int64_t)norm_lsb_fs * ((int64_t)time1 - (int64_t)time2) +
185     ((int64_t)clock_count1 * TDC_CLOCK_PERIOD_FS);
186
187     return TDC_OK;
188 }
189
190 /**
191 * @brief Read/write register data from/to TDC
192 *
193 * Function for both reading and writing register data, depending on @p write
194 * parameter.
195 *
196 * @param[in] tdc -- Pointer to TDC handle
197 * @param[in] address -- Address of the register(s) to read/write
198 * @param[in,out] data -- Pointer to data
199 * @param[in] write -- True = write, false = read
200 *
201 * @attention @p data needs to provide space for at least TDC_REG_SIZE[ @p
202 * address ] bytes.

```

```

197 *
198 * @return TDC_error_t -- Error code
199 */
200 static TDC_error_t send(TDC_t* tdc, TDC_addr_t address, uint8_t* data, bool
201 write)
202 {
203     uint8_t size; // number of data bytes
204     uint8_t cmd = address | (write ? TDC_CMD_WRITE : TDC_CMD_READ);
205
206     if ((tdc == NULL) || (data == NULL)) {
207         return TDC_ERROR;
208     }
209
210     if (address >= TDC_ADR_AMOUNT) {
211         return TDC_WRONG_ADDRESS;
212     }
213
214     size = TDC_REG_SIZE[address];
215
216     HAL_GPIO_WritePin(tdc->cs_port, tdc->cs_pin, GPIO_PIN_RESET);
217     HAL_Delay(1); // 1 ms
218
219     if (HAL_SPI_Transmit(tdc->spi, &cmd, 1, HAL_MAX_DELAY) != HAL_OK) {
220         return TDC_COM_ERROR;
221     }
222
223     if (write) {
224         if (HAL_SPI_Transmit(tdc->spi, data, size, HAL_MAX_DELAY) != HAL_OK) {
225             return TDC_COM_ERROR;
226         }
227     } else {
228         if (HAL_SPI_Receive(tdc->spi, data, size, HAL_MAX_DELAY) != HAL_OK) {
229             return TDC_COM_ERROR;
230         }
231     }
232
233     HAL_GPIO_WritePin(tdc->cs_port, tdc->cs_pin, GPIO_PIN_SET);
234     HAL_Delay(1); // 1 ms
235
236     return TDC_OK;
237 }
```

Code 7: TDC Driver (Source)

## 8.2 Python Analyse

In Code 8 ist das Python-Skript zur Berechnung des arithmetischen Mittelwerts und der Standardabweichung sowie zum Plotten des Histogramms und des Boxplots dargestellt.

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # Read the data from the log.txt file
5 with open('log.txt', 'r') as file:
6     tof_values = []
7     for line in file:
8         # Extract the ToF value using the pattern 'ToF = <value> [ps]'
9         if 'ToF' in line:
10             tof_value = int(line.split('=')[1].split('[')[0].strip())
11             tof_values.append(tof_value)
```

```

12
13 # Convert the list to a numpy array for easier calculation
14 tof_array = np.array(tof_values)
15
16 # Calculate the arithmetic mean and standard deviation
17 mean_toft = np.mean(tof_array)
18 std_toft = np.std(tof_array)
19
20 # Print the results
21 print(f'Arithmetic Mean of ToF: {mean_toft}')
22 print(f'Standard Deviation of ToF: {std_toft}')
23
24 # Plot a histogram
25 plt.figure(figsize=(10, 6))
26 plt.hist(tof_array, bins=20, color='skyblue', edgecolor='black', alpha=0.7)
27 plt.title('Histogram of ToF Values')
28 plt.xlabel('Time-of-Flight (ps)')
29 plt.ylabel('Frequency')
30 plt.grid(True)
31 plt.show()
32
33 # Plot a box plot
34 plt.figure(figsize=(10, 6))
35 plt.boxplot(tof_array, vert=False, patch_artist=True,
             boxprops=dict(facecolor='lightblue', color='black'))
36 plt.title('Box Plot of ToF Values')
37 plt.xlabel('Time-of-Flight (ps)')
38 plt.grid(True)
39 plt.show()

```

Code 8: Python Analyse

In Code 9 ist das Python-Skript zur Berechnung des arithmetischen Mittelwerts und der Standardabweichung sowie zum Plotten der Werte für mehrere Messungen dargestellt.

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # Load data from logs.txt
5 filename = 'logs.txt'
6 data = np.loadtxt(filename, delimiter=';', dtype=float)
7
8 # Calculate arithmetic mean and standard deviation for each column
9 means = np.mean(data, axis=0)
10 std_devs = np.std(data, axis=0)
11
12 # Print results
13 for i, (mean, std_dev) in enumerate(zip(means, std_devs), start=1):
14     print(f"Column {i}: Mean = {mean:.2f}, Standard Deviation = {std_dev:.2f}")
15
16 # Plot all columns with points
17 plt.figure(figsize=(10, 6))
18 labels = ["0 m", "1 m", "2 m", "4 m", "6 m", "7 m"]
19 for i in range(data.shape[1]):
20     plt.scatter(range(len(data[:, i])), data[:, i], label=labels[i], s=5)
21
22 plt.title('ToF for different cable lengths')
23 plt.xlabel('Index')
24 plt.ylabel('ToF [ps]')
25 plt.legend()
26 plt.grid()

```

```
27 plt.tight_layout()  
28 plt.show()
```

Code 9: Python Analyse (Multi)

## 8.3 Schema

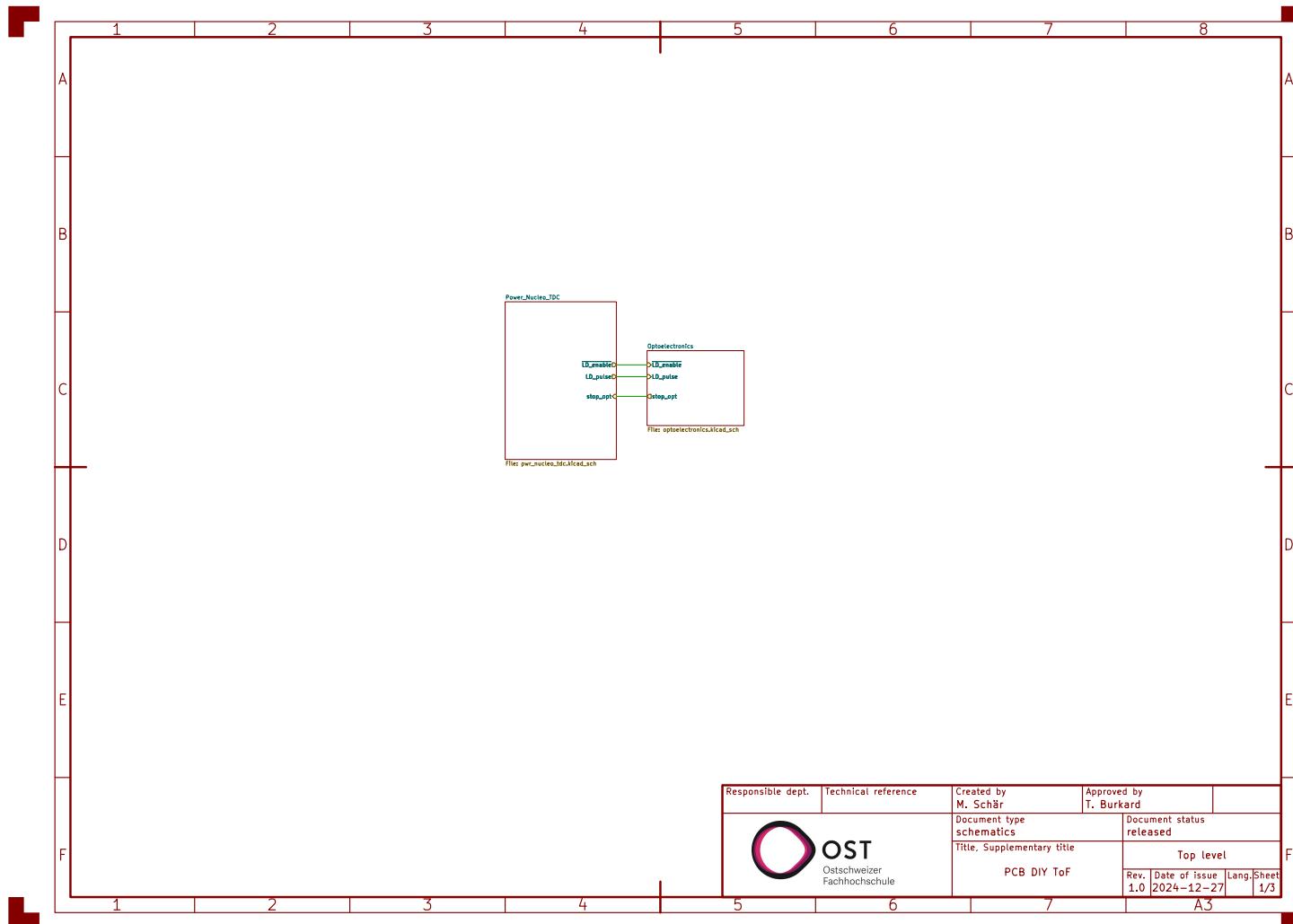


Abbildung 31: Schema S.1/3

# DIY Optische ToF Distanzmessung

## Projektarbeit

OST – Ostschweizer Fachhochschule  
CAS Sensorik und Sensor Signal Conditioning

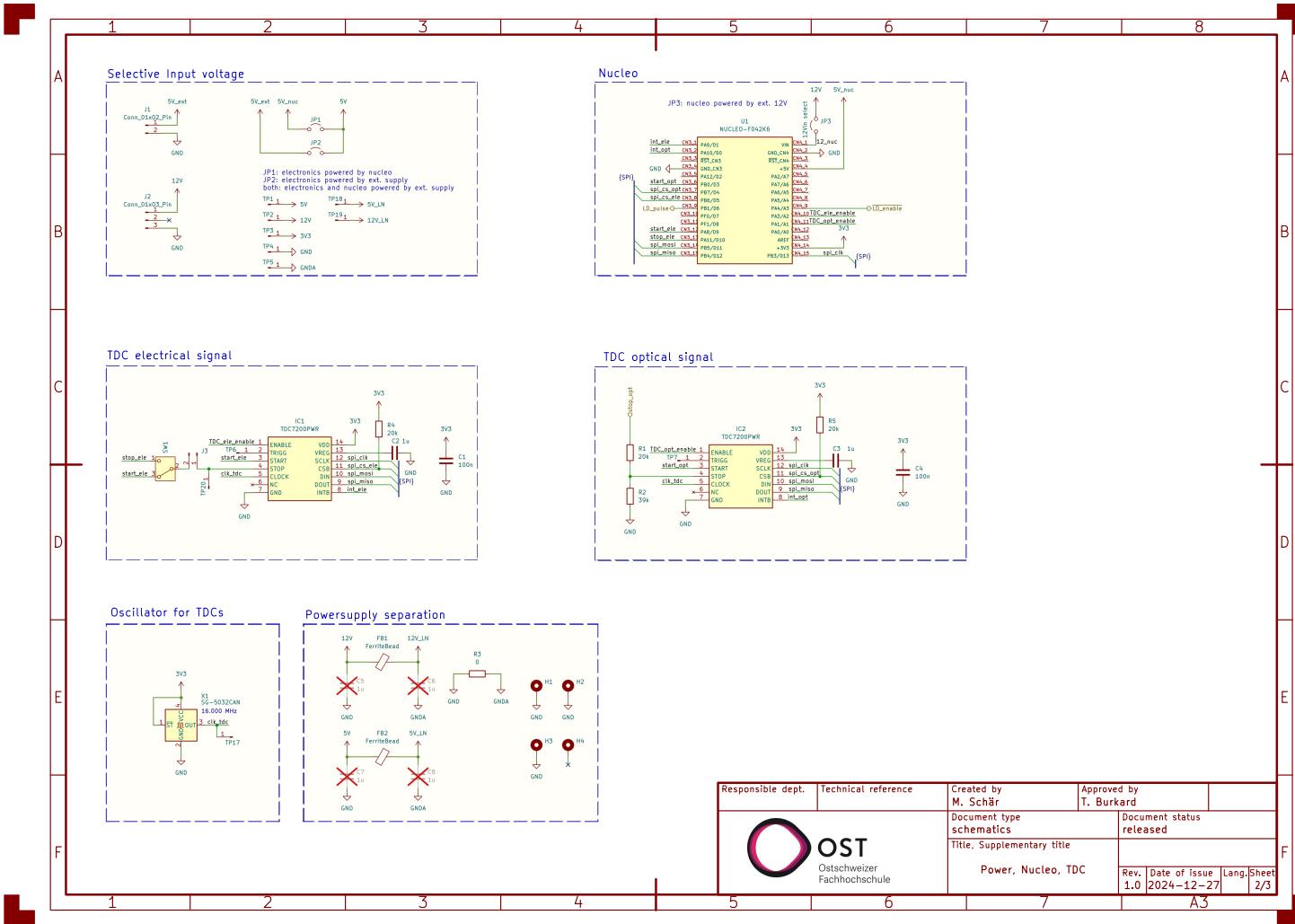


Abbildung 32: Schema S.2/3

# DIY Optische ToF Distanzmessung

Projektarbeit

**OST – Ostschweizer Fachhochschule**  
CAS Sensorik und Sensor Signal Conditioning

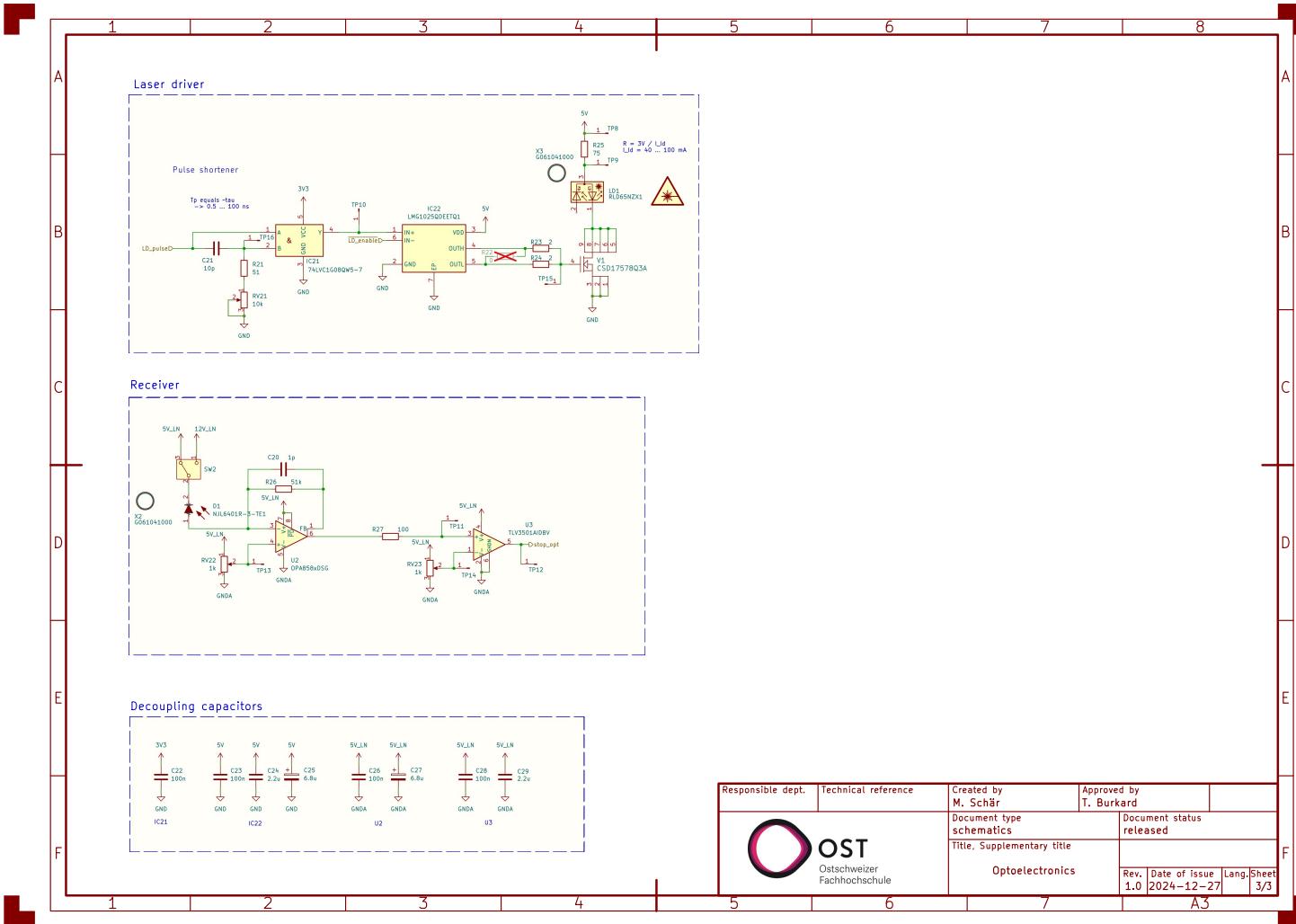


Abbildung 33: Schema S.3/3

## 8.4 Stückliste

Reference	Value	Datasheet	Footprint	Qty	DNP
C1,C4,C22,C23,C26,C28	100n		Capacitor_SMD:C_0402_1005Metric_Pad0.74x0.62mm	6	
C2,C3	1u		Capacitor_SMD:C_0402_1005Metric_Pad0.74x0.62mm	2	
C5,C6,C7,C8	1u		Capacitor_SMD:C_0402_1005Metric_Pad0.74x0.62mm	4	DNP
C20	1p		Capacitor_SMD:C_0402_1005Metric_Pad0.74x0.62mm	1	
C21	10p		Capacitor_SMD:C_0402_1005Metric_Pad0.74x0.62mm	1	
C24,C29	2.2u		Capacitor_SMD:C_0402_1005Metric_Pad0.74x0.62mm	2	
C25,C27	6.8u		Capacitor_SMD:C_0603_1608Metric_Pad1.08x0.95mm	2	
D1	NJL6401R-3-TE1	(JRC, 2014)	Capacitor_Tantal_SMD:CP_EIA-7343-43_Kemet-X_Pad2.25x2.55mm	2	
FB1,FB2	FerriteBead	(TDK, 2019)	NJL6401R3TE1	1	
IC1,IC2	TDC7200PWR	(TI, 2016b)	Inductor_SMD:L_0402_1005Metric_Pad0.77x0.64mm	2	
IC21	74LVC1G08QW5-7	(Diodes Inc., 2020)	SOP65P640X120-14N	2	
IC22	LMG1025QDEETQ1	(TI, 2024)	74LVC1G08QW57	1	
J1,J3	Conn_01x02_Pin		SON65P200X200X80-7N	1	
J2	Conn_01x03_Pin		Connector:Molex_KK-254_AE-6410-02A_1x02_P2.54mm_Vertical	2	
JP1,JP2	Jumper_2_Open		Connector:Molex_SL_171971-0003_1x03_P2.54mm_Vertical	1	
JP3	12Vin select		TestPoint:TestPoint_2Pads_Pitch2.54mm_Drill0.8mm	2	
LD1	RLD65NZX1-00A	(ROHM, 2019)	TestPoint:TestPoint_2Pads_Pitch2.54mm_Drill0.8mm	1	
R1,R4,R5	20k		RLD65NZX1-00A:RLD85PZJ400A	1	
R2	39k		Resistor_SMD:R_0402_1005Metric_Pad0.72x0.64mm	3	
R3	0		Resistor_SMD:R_0402_1005Metric_Pad0.72x0.64mm	1	
R21	51		Resistor_SMD:R_0402_1005Metric_Pad0.72x0.64mm	1	
R22	0		Resistor_SMD:R_0402_1005Metric_Pad0.72x0.64mm	1	
R23,R24	2		Resistor_SMD:R_0402_1005Metric_Pad0.72x0.64mm	2	DNP
R25	75		Resistor_SMD:R_0603_1608Metric_Pad0.98x0.95mm	1	
R26	51k		Resistor_SMD:R_0402_1005Metric_Pad0.72x0.64mm	1	
R27	100		Resistor_SMD:R_0402_1005Metric_Pad0.72x0.64mm	1	
RV21	10k	(Bourns, 2024)	Potentiometer_SMD:Potentiometer_Bourns_3224W_Vertical	1	
RV22,RV23	1k	(Bourns, 2024)	Potentiometer_SMD:Potentiometer_Bourns_3224W_Vertical	2	
SW1,SW2	SW_Nidec_CAS-120A1	(Nidec Comp., 2024)	Button_Switch_SMD:Nidec_Copal_CAS-120A	2	
TP1..TP20	Conn_01x01_Pin		Connector_2.54mm:PinHeader_1x01_P2.54mm_Vertical	20	
U1	NUCLEO-F042K6	(ST, 2019)	NUCLEO_F042K6:MODULE_NUCLEO-F042K6	1	
U2	OPA858xDSG	(TI, 2018)	Package SON:WSON-8-1EP_2x2mm_P0.5mm_EP0.9x1.6mm	1	
U3	TLV3501AIDBV	(TI, 2016c)	Package TO_SOT_SMD:SOT-23-6	1	
V1	CSD17578Q3A	(TI, 2016a)	DNH0008A	1	
X1	SG-5032CAN	(Epson Timing, 2024)	Oscillator:Oscillator_SMD_SeikoEpson_SG8002LB-4Pin_5.0x3.2mm	1	
X2,X3	G061041000	(Qioptiq, 2024)	user:Qioptiq-Mount	2	

Tabelle 4: Bill of Material

# Quellenverzeichnis