DIY Optische ToF Distanzmessung

CAS Sensorik und Sensor Signal Conditioning

Matthias Schär, Timon Burkard
OST – Ostschweizer Fachhochschule

12. Januar 2025

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

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

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

Ort, Datum Rapperswil, 12. 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

CPU Central Processing Unit. 30, 31, 33, 34, 39

DIY Do It Yourself. 9

DSO Digital Storage Oscilloscope. 5, 30, 32, 33, 35, 36

GPIO General Purpose Input/Output. 5, 6, 8, 30–37, 39

HAL Hardware Abstraction Layer. 5, 6, 8, 30–36, 39

MCU Microcontroller Unit. 30, 33, 39

PCB Printed Circuit Board. 20

TDC Time to Digital Converter. 5, 9, 14, 17, 19, 31, 32, 34, 35, 38–40, 43

TIA Trans-Impedance Amplifier. 13, 17

ToF Time of Flight. 9, 10

UART Universal Asynchronous Receiver/Transmitter. 31, 34

Abbildungsverzeichnis

Formelverzeichnis

Tabellenverzeichnis

Codeverzeichnis

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.

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

$$I_e = \frac{P_{out}}{\Omega} \tag{2}$$

$$\Omega = \frac{4 \cdot \pi \cdot 0.5}{d} \tag{3}$$

$$I_{ph} = S \cdot P_{in} \tag{4}$$

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.

$$P_{out} = 285\ mW \tag{5}$$

$$\begin{aligned} A &= 4.4\ mm^2 \\ S &= 0.6\ \frac{A}{W} \end{aligned} \tag{6}$$

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

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

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.

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

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

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

$$
\begin{aligned}
I_e &= \frac{P_{out}}{\Omega} = \frac{10\ mW}{\frac{4 \cdot \pi \cdot 0.5}{d}} = \frac{10\ mW}{\frac{4 \cdot \pi \cdot 0.5}{10\ m}} = 1.6\ \frac{mW}{sr} \\
P_{in} &= \frac{I_e}{r^2} \cdot A = 45\ \frac{mW}{sr} \cdot 0.49\ mm^2 = 8\ pW \\
I_{ph} &= S \cdot P_{in} = 0.42\ \frac{A}{W} \cdot 8\ pW = 3.3\ pA
\end{aligned} \tag{10}
$$

2.3 Transimpedanzverstärker

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

3 Umsetzung

3.1 Firmware

Der selbst entwickelte Firmware-Treiber für den TDC befindet im Anhang Kapitel 7.1.

3.2 Schaltungen

3.2.1 Selective Input Voltage

Abbildung 1 zeigt die Beschaltung zur Selektion der Speisung.

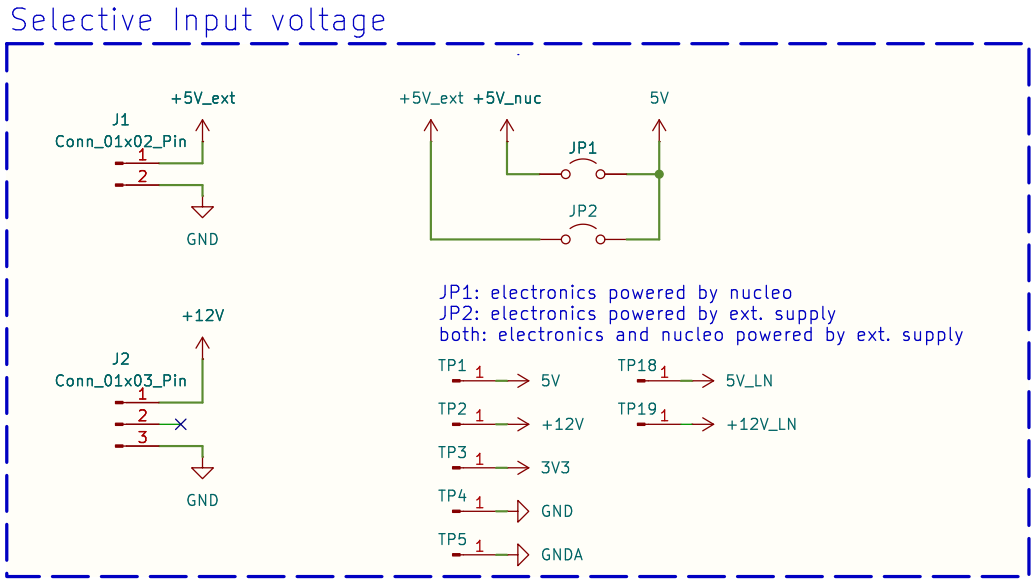

Abbildung 1: Selective Input Voltage

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 3.2.2.

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 3.2.8.

3.2.2 Nucleo Board

Die Beschaltung des NUCLEO-F042K6 Boards (ST, 2024) ist in Abbildung 2 gezeigt.

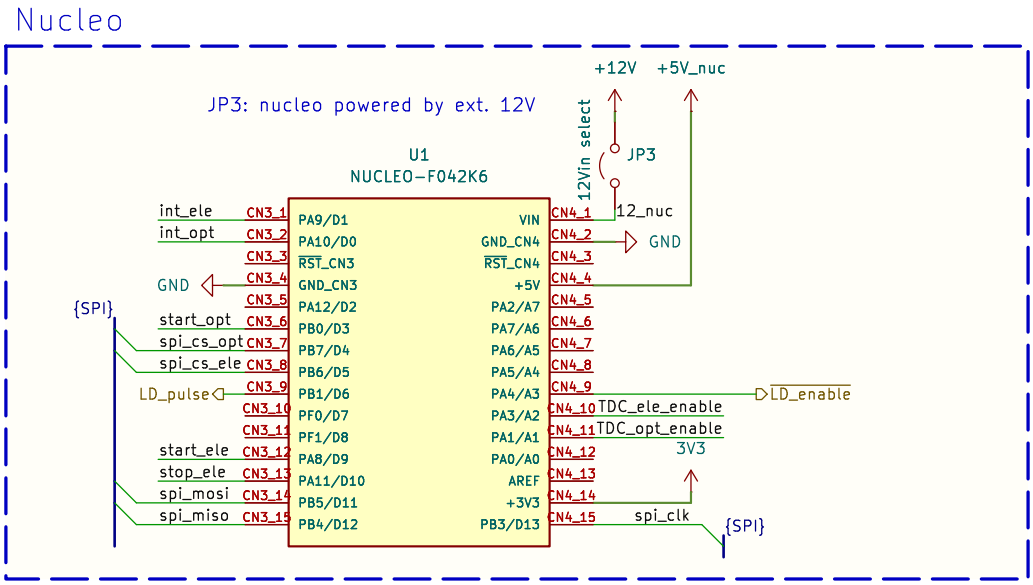

Abbildung 2: Nucleo Board

3.2.3 TDC Electrical Signal

Die Beschaltung des TDC7200 (TI, 2016b) für den elektrischen Teil ist in Abbildung 3 gezeigt.

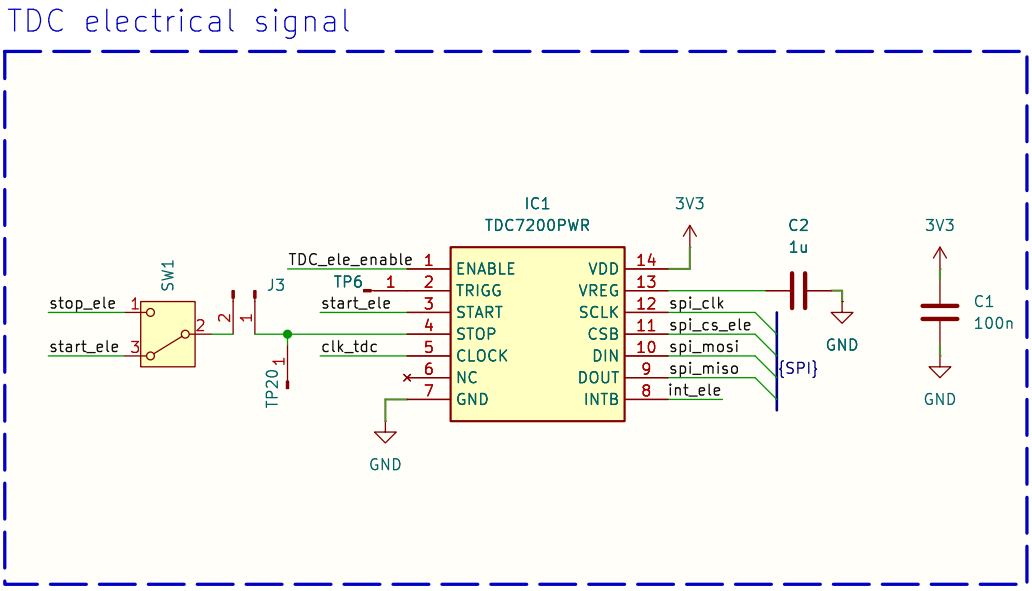

Abbildung 3: TDC Electrical Signal

3.2.4 TDC Optical Signal

Die Beschaltung des TDC7200 (TI, 2016b) für den optischen Teil ist in Abbildung 4 gezeigt.

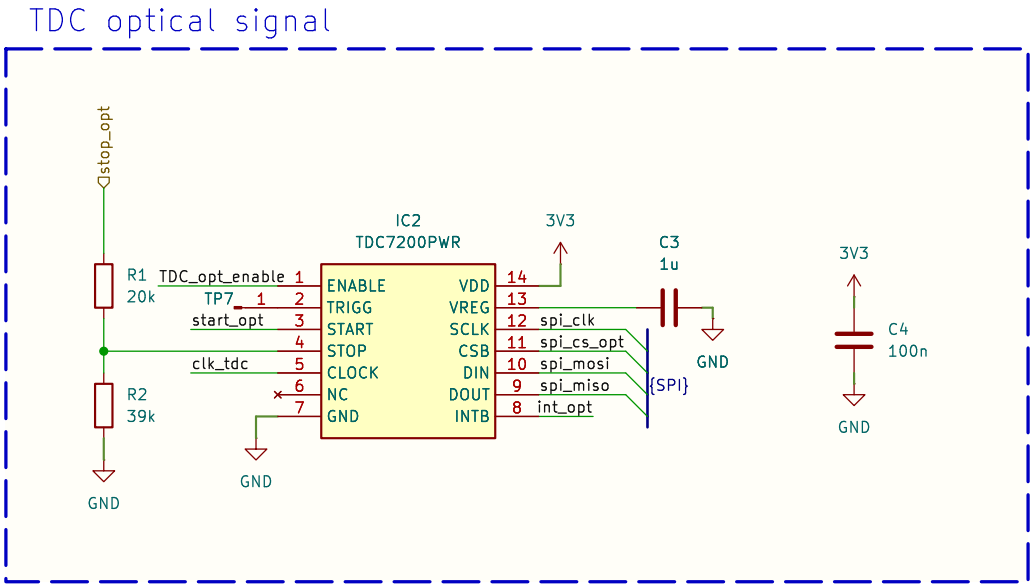

Abbildung 4: TDC Optical Signal

3.2.5 Oscillator For TDCs

Die Beschaltung des Oszillators für die beiden TDC ist in Abbildung 5 gezeigt.

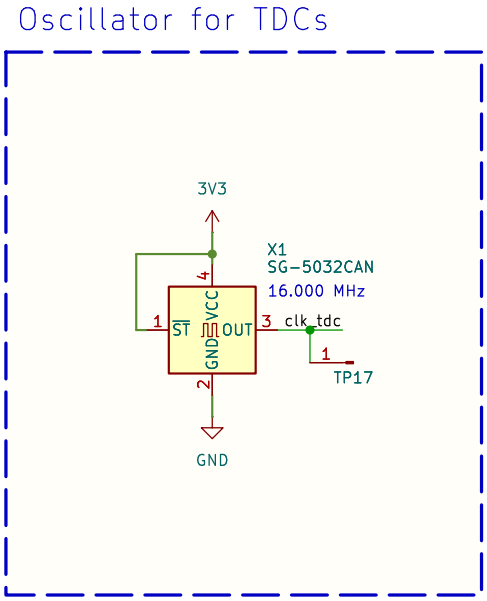

Abbildung 5: Oscillator for TDCs

3.2.6 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 6 dargestellt ist.

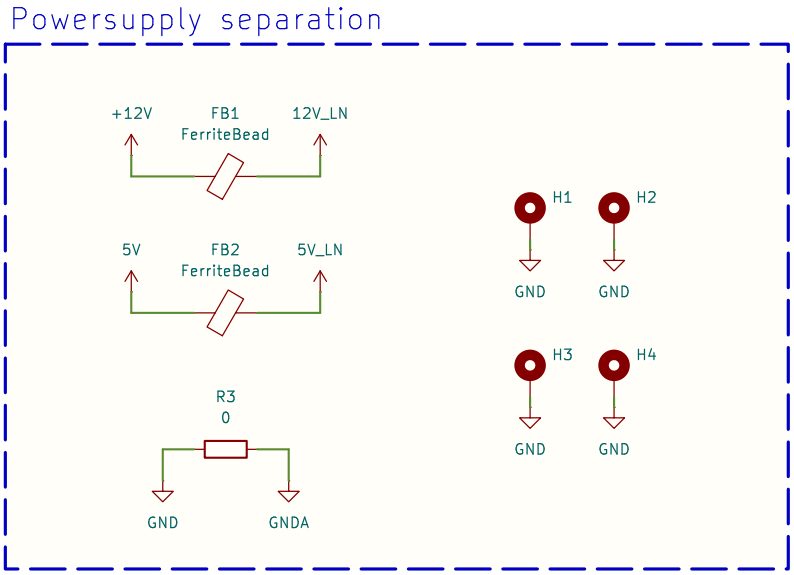

Abbildung 6: Power Supply Separation

3.2.7 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 . . . 100 ns) wurde mittels Hochpass und AND-Gatter (Diodes Inc., 2020) implementiert. Siehe dazu Abbildung 7.

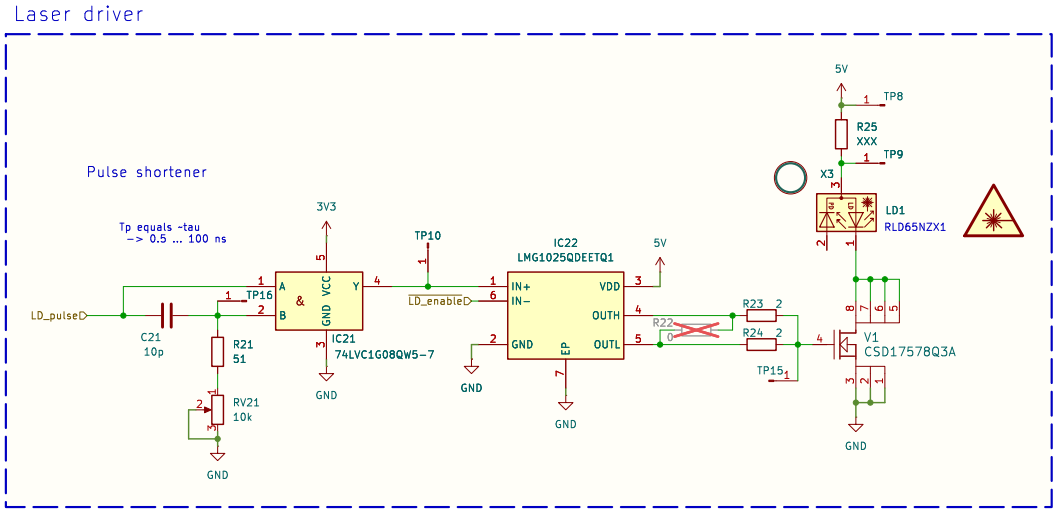

Abbildung 7: Laser Driver

3.2.8 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 Ausgangs des Transimpedanzverstärkers geht auf den Komparator

TLV3501 (TI, 2016c), um das STOP-Signal für den TDC zu generieren. Siehe dazu Abbildung 8.

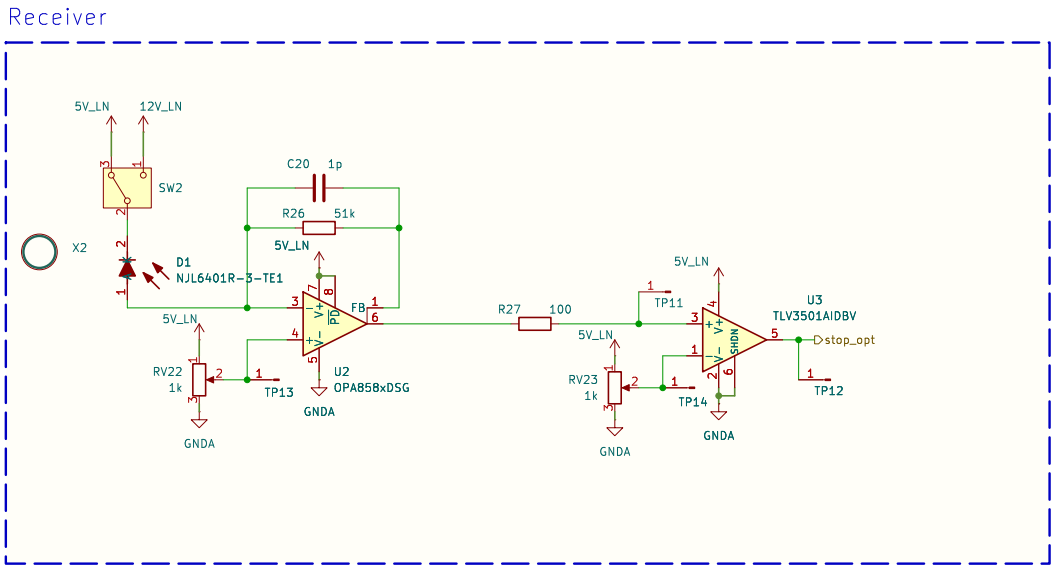

Abbildung 8: Photo Receiver

3.2.9 Decoupling Capacitors

Die Beschaltung der Entkopplungs-Kondensatoren ist in Abbildung 9 dargestellt.

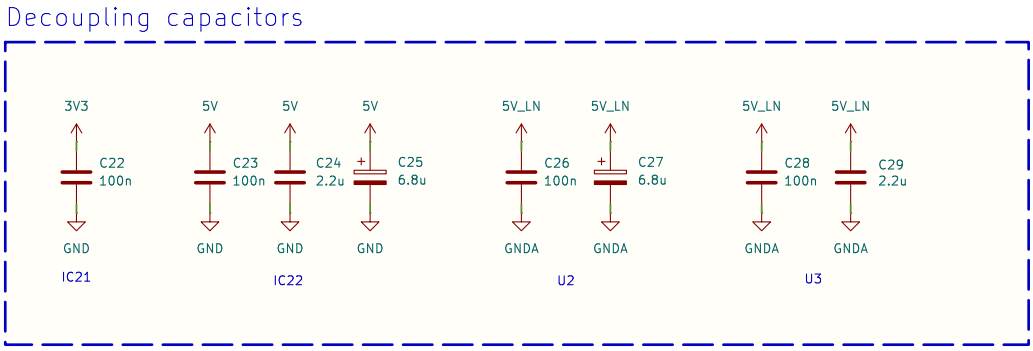

Abbildung 9: Decoupling Capacitors

3.3 Layout

In diesem Kapitel werden die PCB-Layouts dokumentiert.

3.3.1 Kupfer-Layer

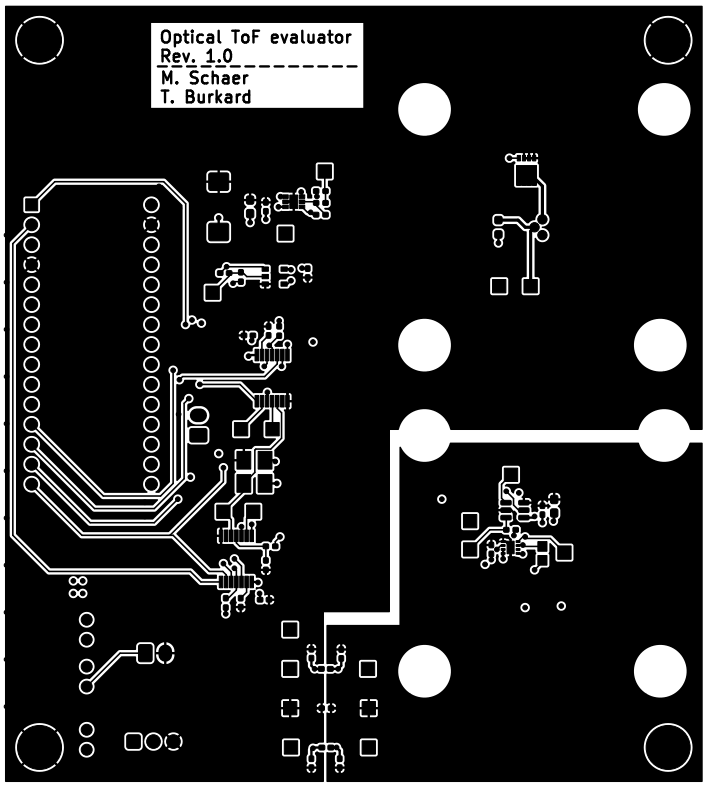

Abbildung 10: PCB Layout Top

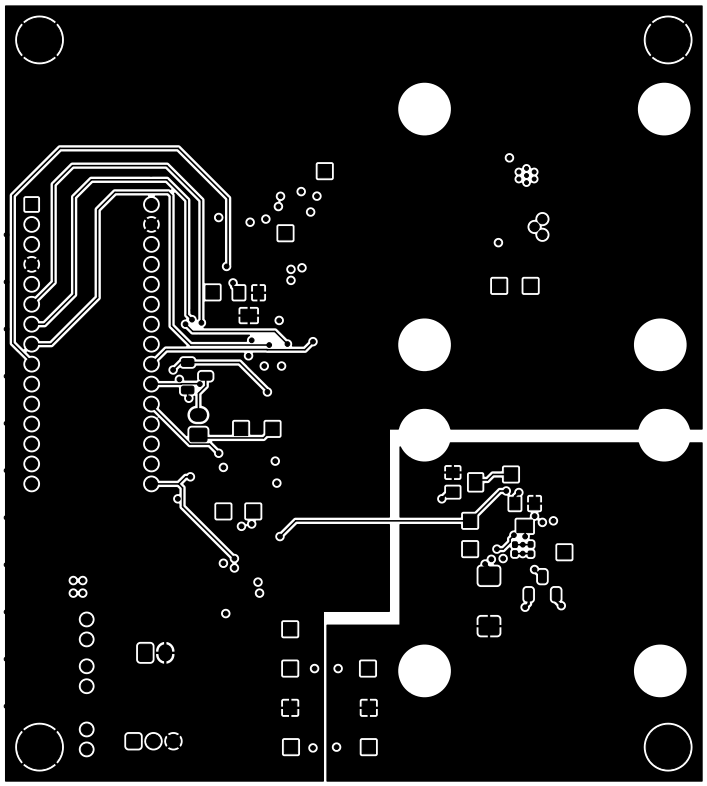

Abbildung 11: PCB Layout Bottom

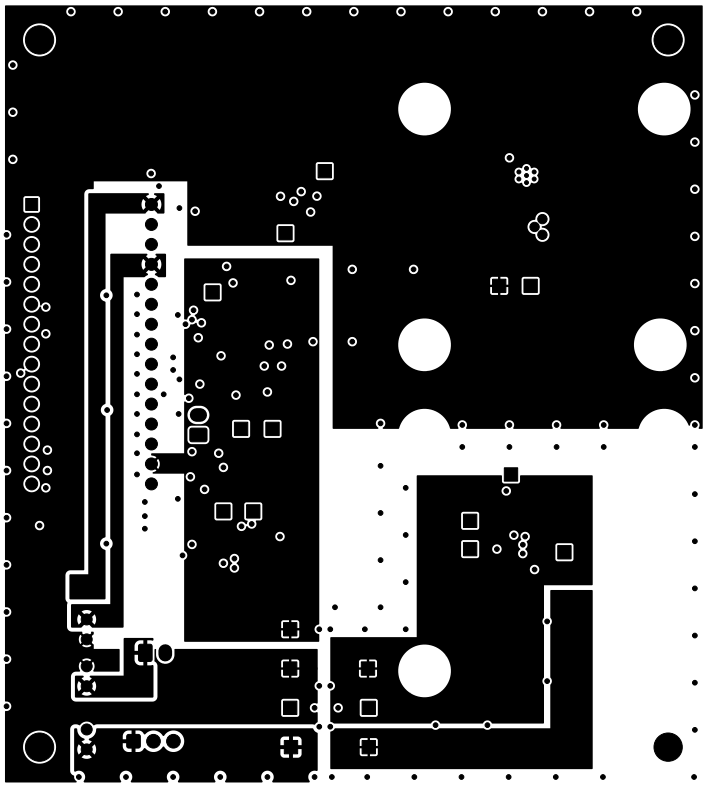

Abbildung 12: PCB Layout Innen 1

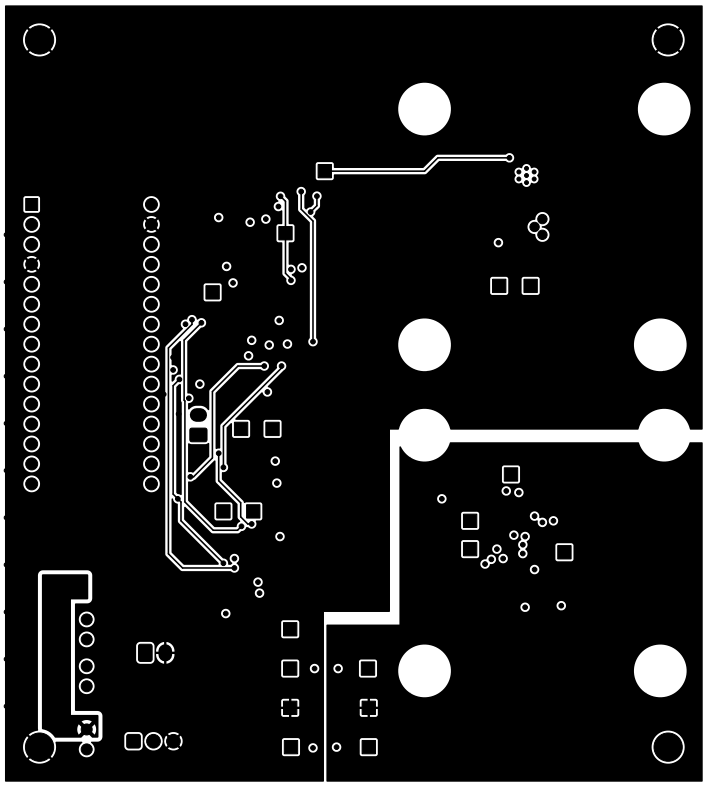

Abbildung 13: PCB Layout Innen 2

3.3.2 Komponenten-Platzierung

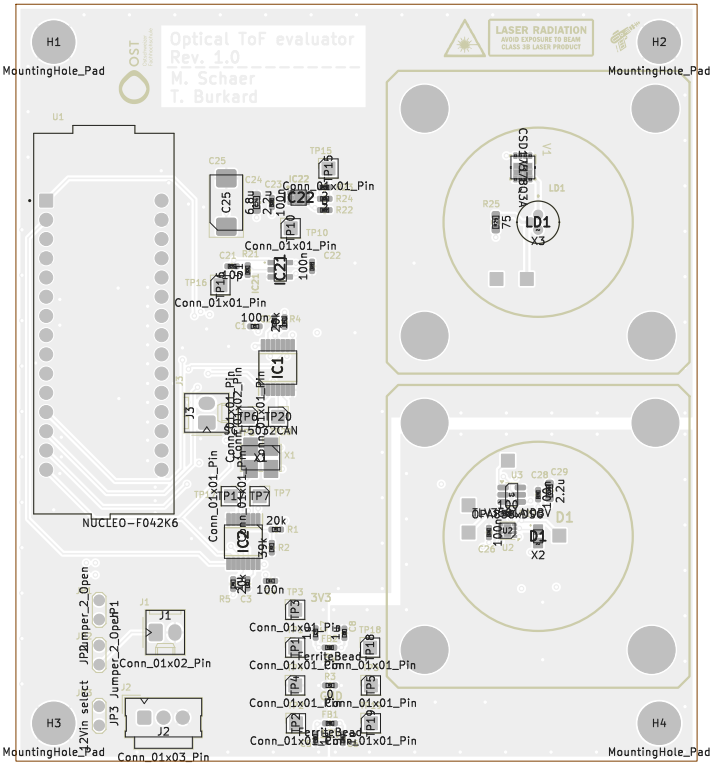

Abbildung 14: PCB Komponenten-Platzierung Top

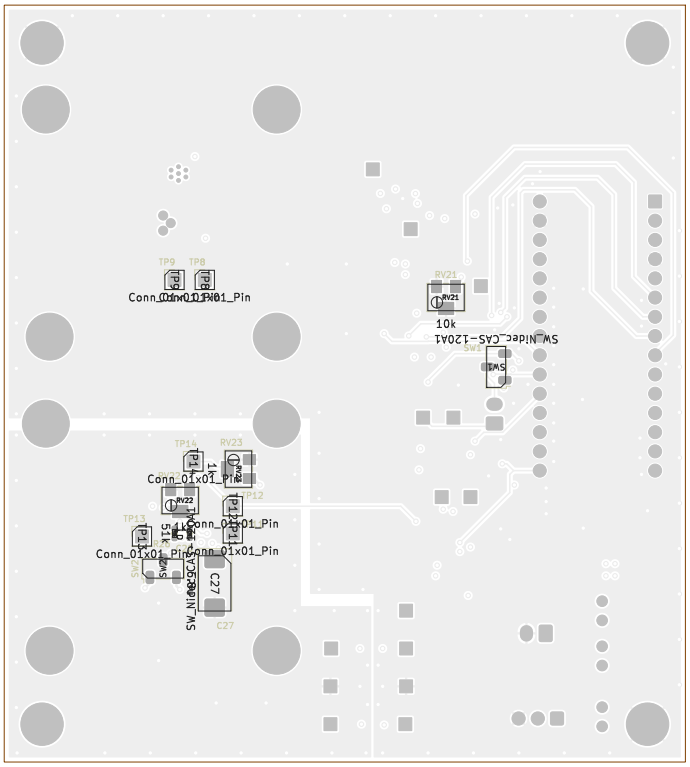

Abbildung 15: PCB Komponenten-Platzierung Bottom

3.4 3D View

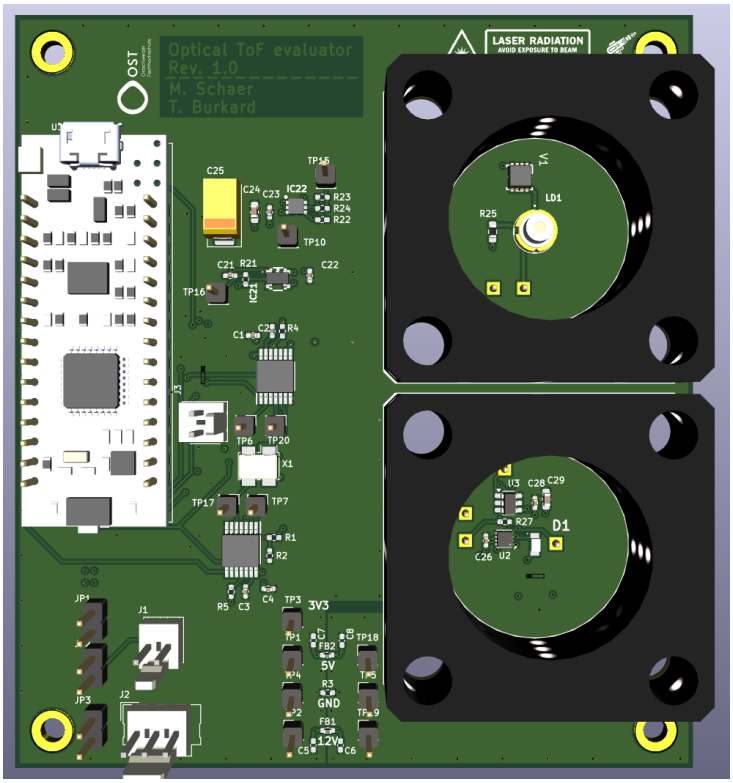

Abbildung 16: 3D View Top

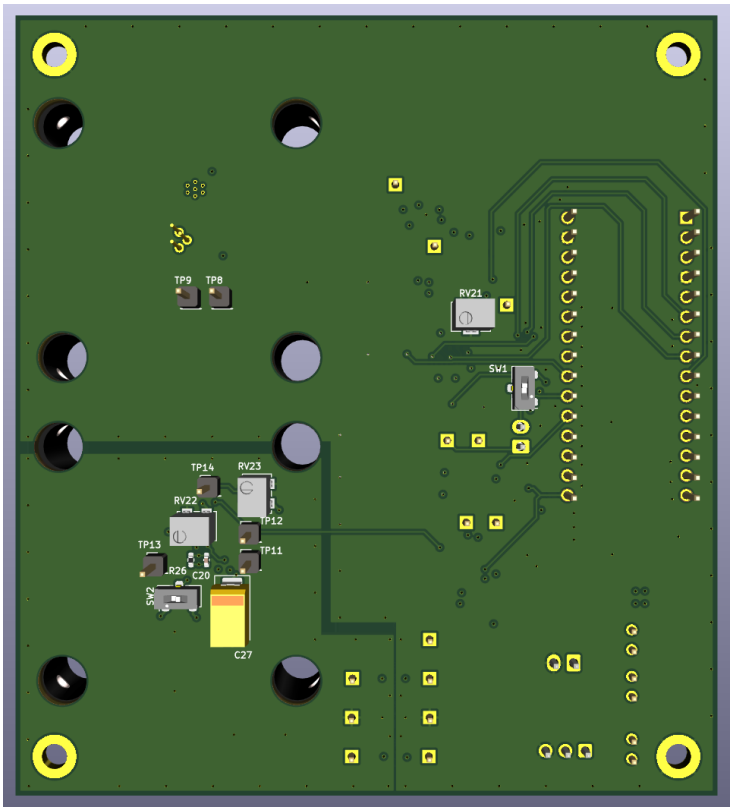

Abbildung 17: 3D View Bottom

3.5 Komponenten

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_0603_1608Metric_Pad1.08x0.95mm	2	
C25,C27	6.8u		Capacitor_Tantal_SMD:CP_EIA-7343-43_Kemet-X_Pad2.25x2.55mm	2	
D1	NJL6401R-3-TE1	(JRC, 2014)	NJL6401R3TE1	1	
FB1,FB2	FerriteBead	(TDK, 2019)	Inductor_SMD:L_0402_1005Metric_Pad0.77x0.64mm	2	
IC1,IC2	TDC7200PWR	(TI, 2016b)	SOP65P640X120-14N	2	
IC21	74LVC1G08QW5-7	(Diodes Inc., 2020)	74LVC1G08QW57	1	
IC22	LMG1025QDEETQ1	(TI, 2024)	SON65P200X200X80-7N	1	
J1,J3	Conn_01x02_Pin		Connector:Molex_KK-254_AE-6410-02A_1x02_P2.54mm_Vertical	2	
J2	Conn_01x03_Pin		Connector:Molex_SL_171971-0003_1x03_P2.54mm_Vertical	1	
JP1,JP2	Jumper_2_Open		TestPoint:TestPoint_2Pads_Pitch2.54mm_Drill0.8mm	2	
JP3	12Vin select		TestPoint:TestPoint_2Pads_Pitch2.54mm_Drill0.8mm	1	
LD1	RLD65NZX1-00A	(ROHM, 2019)	RLD65NZX1-00A:RLD85PZJ400A	1	
R1,R4,R5	20k		Resistor_SMD:R_0402_1005Metric_Pad0.72x0.64mm	3	
R2	39k		Resistor_SMD:R_0402_1005Metric_Pad0.72x0.64mm	1	
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	DNP
R23,R24	2		Resistor_SMD:R_0402_1005Metric_Pad0.72x0.64mm	2	
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 1: Bill of Material

4 Simulationen

4.1 Laser Treiber

4.2 Transimpedanzverstärker

5 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 7.2.

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

5.1 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 3) durchgeführt und von der Firmware, welche auf dem Nucleo Board U1 (siehe Abbildung 2) getriggert und ausgelesen.

5.1.1 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.

```
// Configure TDC

TDC_init(&tdc_ele, &hspi1, SPI_CS_ELE_GPIO_Port, SPI_CS_ELE_Pin,
    TDC_ELE_ENABLE_GPIO_Port, TDC_ELE_ENABLE_Pin);
TDC_enable(&tdc_ele);

TDC_read(&tdc_ele, TDC_ADR_CONFIG1, data);
data[0] |= TDC_CONFIG1_MEAS_MODE_2;

TDC_write(&tdc_ele, TDC_ADR_CONFIG1, data);

while (1) {
    TDC_start(&tdc_ele);

    // Set Pins to High with HAL
    HAL_GPIO_WritePin(START_ELE_GPIO_Port, START_ELE_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(STOP_ELE_GPIO_Port, STOP_ELE_Pin, GPIO_PIN_SET);

    TDC_read_result(&tdc_ele, &tof_fs);

    sprintf(string, "ToF = %lu [ps]\n", tof_fs / 1000);
    HAL_UART_Transmit(&huart2, (uint8_t *)string, strlen(string), 10000);

    // Set Pins to Low with HAL (preparation for next iteration)
    HAL_GPIO_WritePin(START_ELE_GPIO_Port, START_ELE_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(STOP_ELE_GPIO_Port, STOP_ELE_Pin, GPIO_PIN_RESET);

    HAL_Delay(10); // 10 ms
}
```

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 3.

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

```
ToF = 6375779 [ps]
ToF = 6374666 [ps]
ToF = 6377003 [ps]
ToF = 6375333 [ps]
ToF = 6377061 [ps]
ToF = 6374721 [ps]
ToF = 6377504 [ps]
ToF = 6374888 [ps]
ToF = 6376725 [ps]
ToF = 6377005 [ps]
...
```

Code 2: GPIO Toggle mit HAL

Arithmetischer Mittelwert und Standardabweichung sind in Formel 11 aufgeführt.

$$\begin{aligned} \overline{ToF} &= 6'375'887.9\ ps \\ \sigma &= 1'059.2\ ps \end{aligned} \tag{11}$$

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 18 bzw. 19 dargestellt.

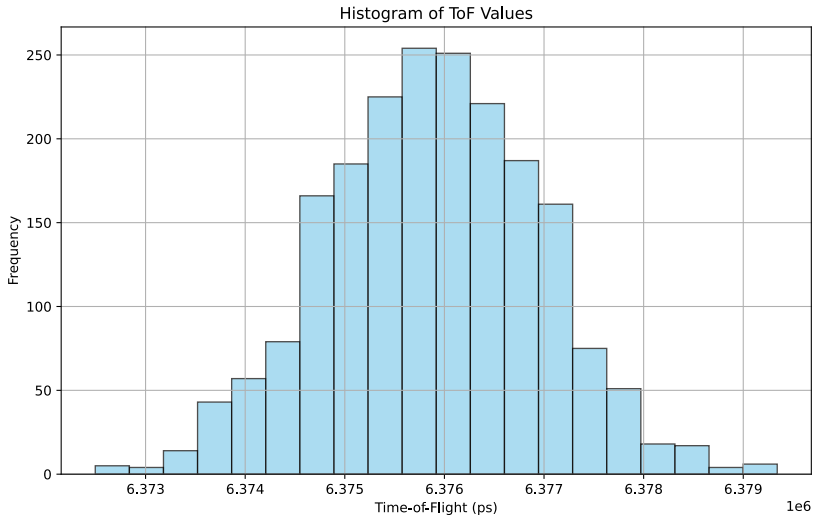

Abbildung 18: GPIO Toggle mit HAL - Histogramm

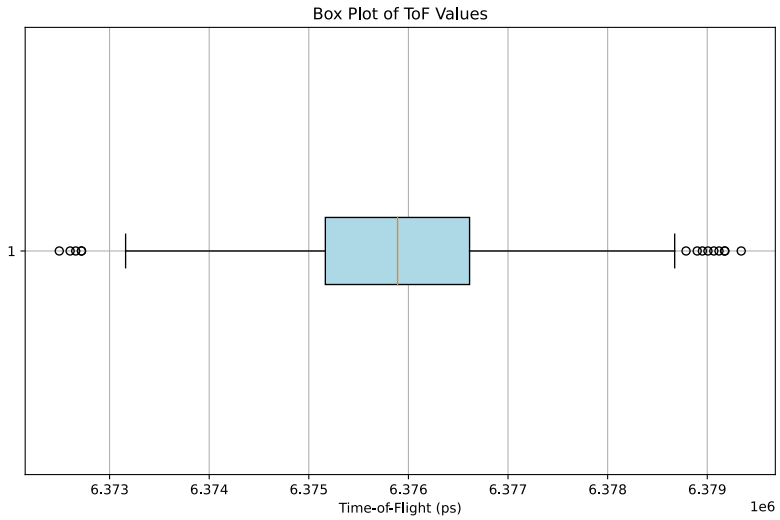

Abbildung 19: GPIO Toggle mit HAL - Boxplot

Um die Resultate des TDC zu validieren, wurde dieselbe Messung auch mittels Digital Storage Oscilloscope (DSO) durchgeführt. Die Messungen sind in Abbildung 20 und 21 dargestellt.

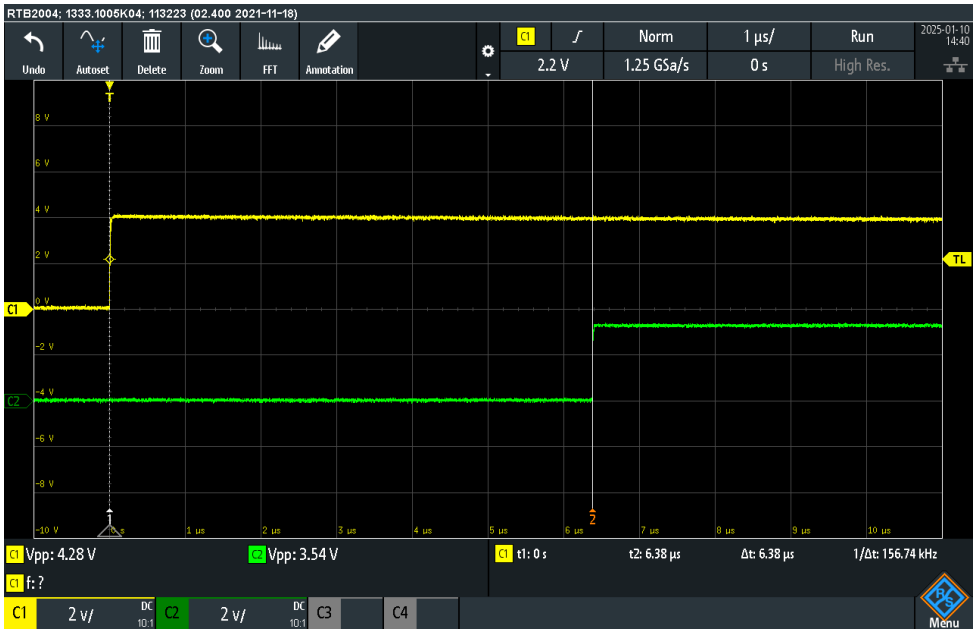

Abbildung 20: GPIO Toggle mit HAL - DSO

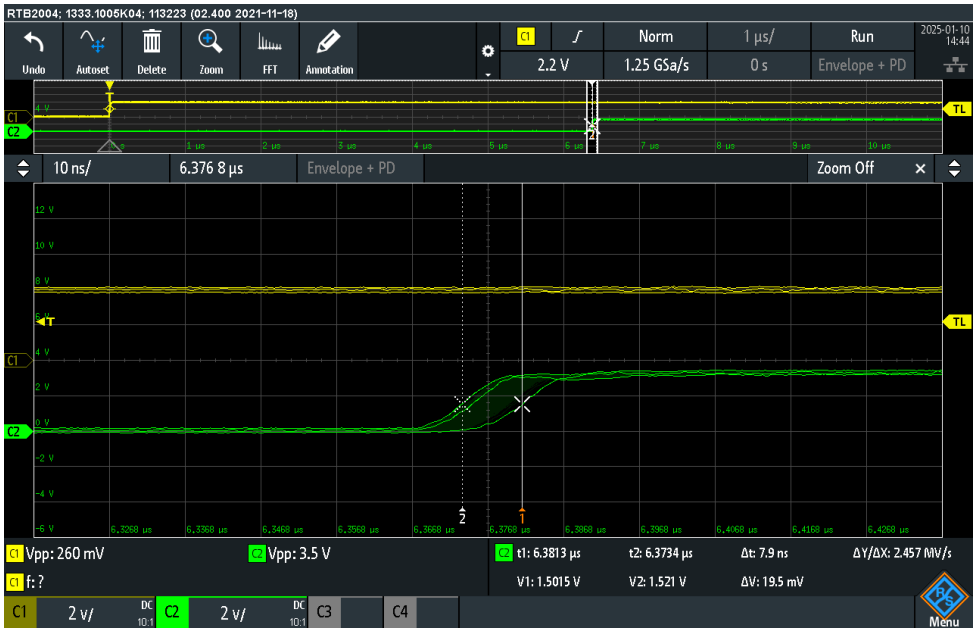

Abbildung 21: GPIO Toggle mit HAL - DSO (Zoom)

5.1.2 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.

```
// Configure TDC

TDC_init(&tdc_ele, &hspi1, SPI_CS_ELE_GPIO_Port, SPI_CS_ELE_Pin,
    TDC_ELE_ENABLE_GPIO_Port, TDC_ELE_ENABLE_Pin);
TDC_enable(&tdc_ele);

TDC_read(&tdc_ele, TDC_ADR_CONFIG1, data);
data[0] |= TDC_CONFIG1_MEAS_MODE_2;

TDC_write(&tdc_ele, TDC_ADR_CONFIG1, data);

while (1) {
    TDC_start(&tdc_ele);

    // Set Pins to High with direct register access
    *((volatile uint32_t*)(GPIOA_BASE + 0x14)) |= (1 << 8);  // START_ELE
    *((volatile uint32_t*)(GPIOA_BASE + 0x14)) |= (1 << 11); // STOP_ELE

    TDC_read_result(&tdc_ele, &tof_fs);

    sprintf(string, "ToF = %lu [ps]\n", tof_fs / 1000);
    HAL_UART_Transmit(&huart2, (uint8_t *)string, strlen(string), 10000);

    // Set Pins to Low with direct register access (preparation for next
    iteration)
    *((volatile uint32_t*)(GPIOA_BASE + 0x14)) &= ~(1 << 8);  // START_ELE
    *((volatile uint32_t*)(GPIOA_BASE + 0x14)) &= ~(1 << 11); // STOP_ELE

    HAL_Delay(10); // 10 ms
```

```
28 }
```

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 5.1.1, der STOP-Pin des TDC via `SW1` mit `stop_ele` verbunden. Der Kabelanschluss `J3` wird kurzgeschlossen. Siehe Abbildung 3.

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 12 aufgeführt.

$$\begin{aligned} \overline{ToF} &= 1'377'772.6\ ps \\ \sigma &= 401.6\ ps \end{aligned} \tag{12}$$

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 22 bzw. 23 dargestellt.

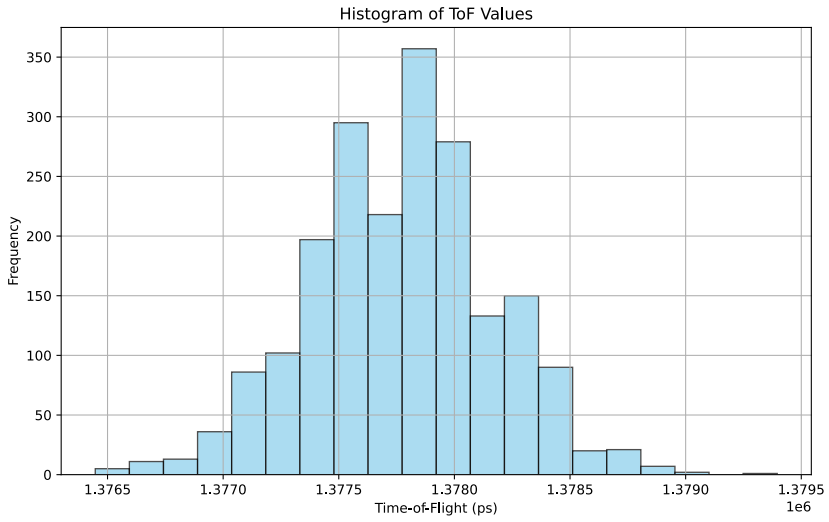

Abbildung 22: GPIO Toggle ohne HAL - Histogramm

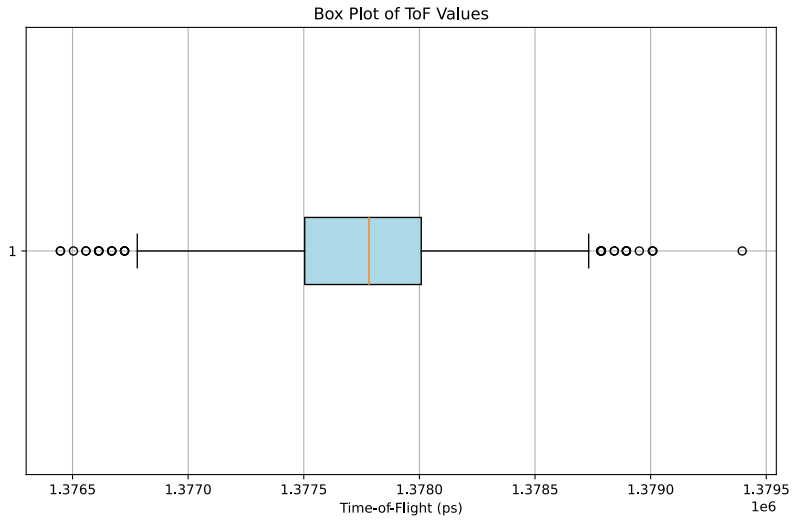

Abbildung 23: GPIO Toggle ohne HAL - Boxplot

Um die Resultate des TDC zu validieren, wurde dieselbe Messung auch mittels Digital Storage Oscilloscope (DSO) durchgeführt. Die Messungen sind in Abbildung 24 und 25 dargestellt.

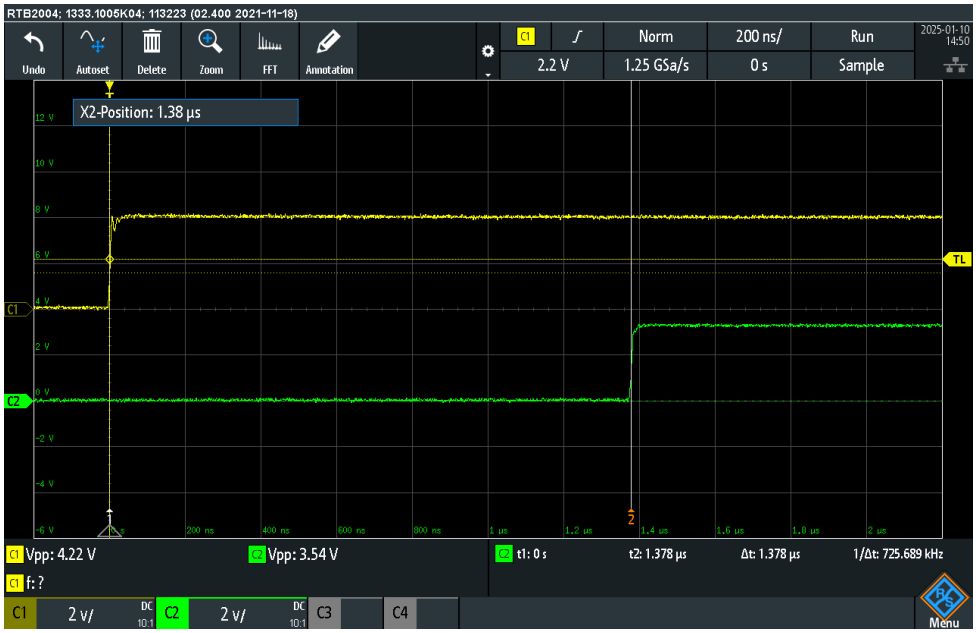

Abbildung 24: GPIO Toggle ohne HAL - DSO

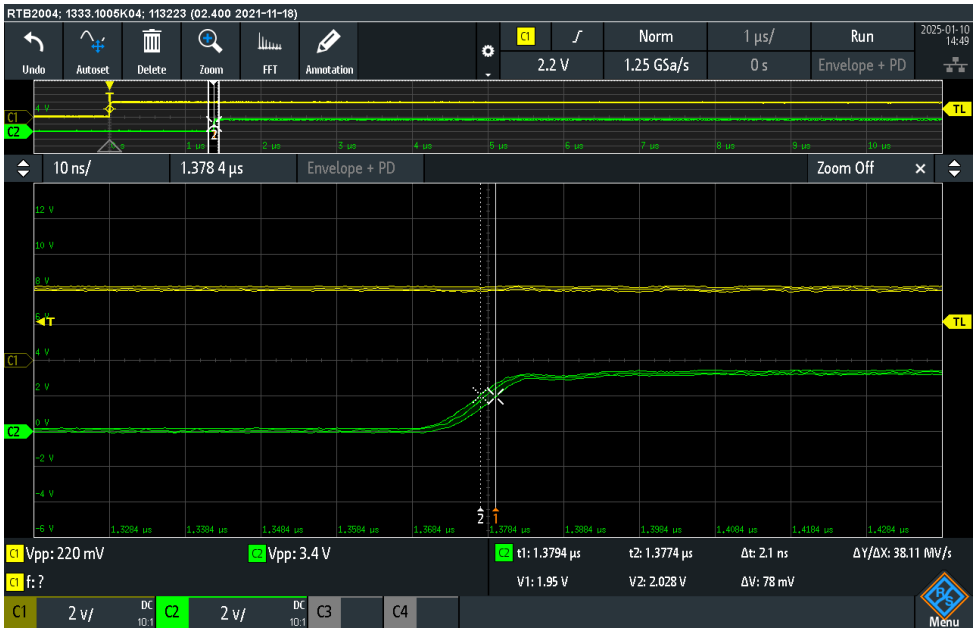

Abbildung 25: GPIO Toggle ohne HAL - DSO (Zoom)

5.1.3 Unterschiedliche Kabellängen

Für diese Messung wird dasselbe Setup wie in Kapitel 5.1.2 verwendet.

Anstelle eines Kurzschlusses von `J3` (siehe Abbildung 3) 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 kürze Zeiten gemessen. Dies hat mit der kapazitiven Kopplung zwischen den Leitern zu tun.

Die Resultate sind in Abbildung 26 dargestellt. Die Liste mit den Datenpunkten befindet sich im elektronischen Anhang.

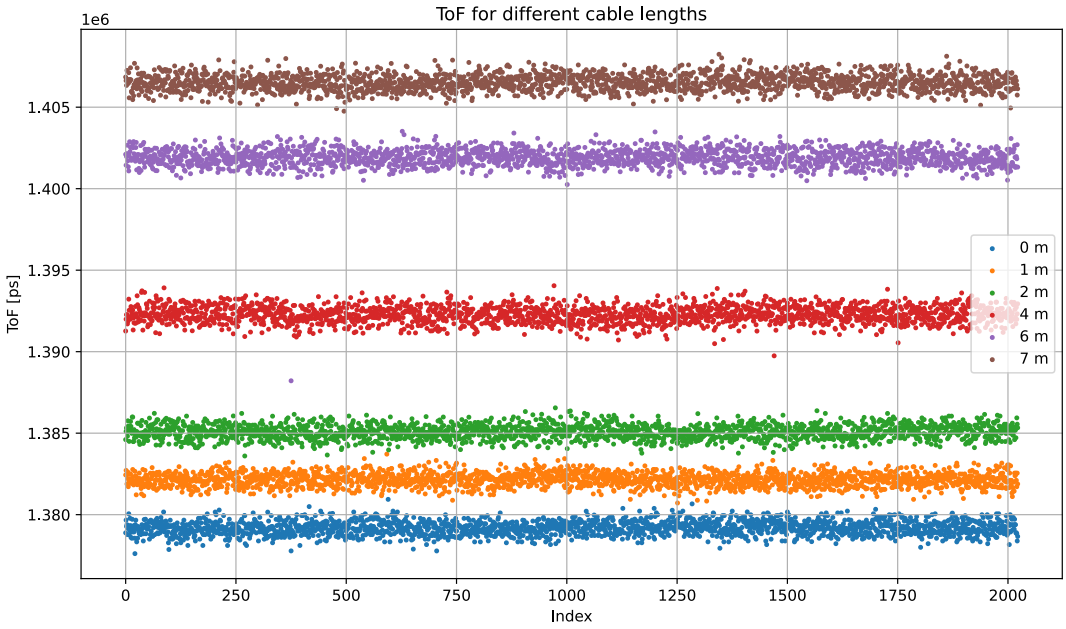

Abbildung 26: Unterschiedliche Kabellängen

Die arithmetischen Mittelwerte und Standardabweichungen sind in Tabelle 2 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 2: Unterschiedliche Kabellängen

Die Signal-Ausbreitungsgeschwindigkeit in Kupfer beträgt ca. 2/3 der Lichtgeschwindigkeit (firewall.xc, 2025). Um die Resultate in Tabelle 2 zu validieren, rechnen wir wie in Formel 13 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.

$$\begin{aligned} c_{cu} &\approx \frac{2}{3} \cdot c_0 = \frac{2}{3} \cdot 299'792'458 \ \frac{m}{s} \approx 200'000'000 \ \frac{m}{s} \\ ToF_n &= ToF_{n_{abs}} - ToF_0 \\ l_n &= ToF_n \cdot c_{cu} \end{aligned} \tag{13}$$

Die Resultate sind in Tabelle 3 dargestellt.

Tatsächliche Länge	ToF_n	Zurückgerechnete Länge
0 m	0 ps	
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 3: 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.

5.1.4 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 27.

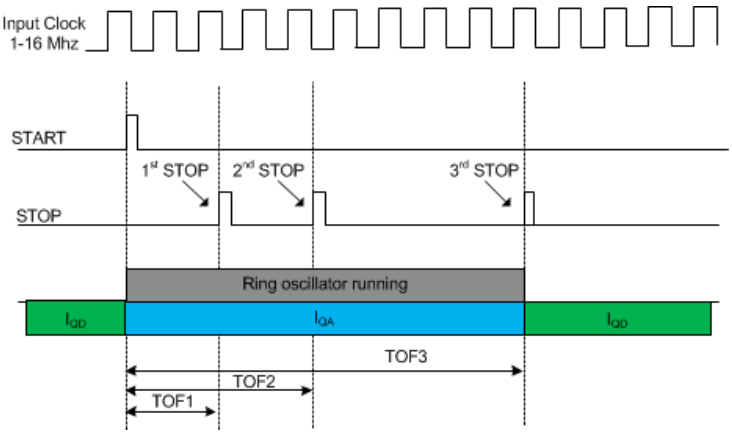

Abbildung 27: TDC Mode 1 (TI, 2016b)

Im Mode 2, um längere Zeiten messen zu können, wir zusätzlich der externe Clock des TDC verwendet. Siehe dazu Abbildung 28.

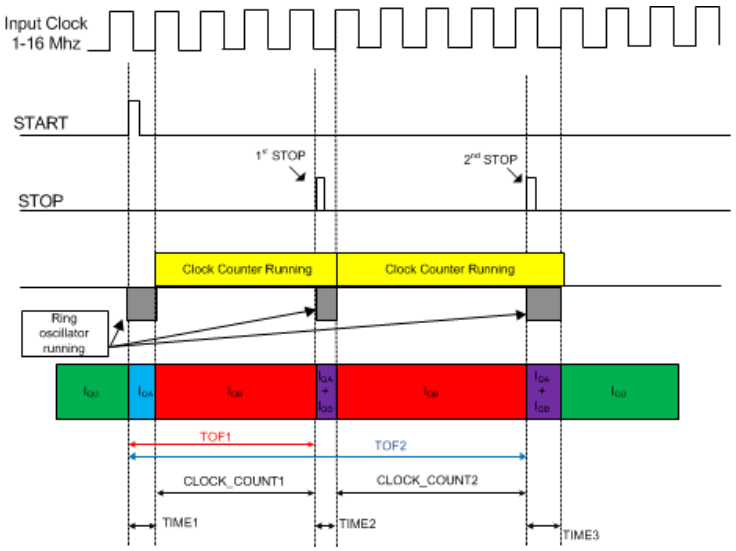

Abbildung 28: TDC Mode 2 (TI, 2016b)

Die bisherigen Messungen (Kapitel 5.1.1, 5.1.2 und 5.1.3) 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 5.1.2 und 5.1.3)
2. GPIO Toggle ohne HAL im Mode 2 mit Kabellänge = 6 m (wie in Kapitel 5.1.3)
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 3).

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

```
// Configure TDC

TDC_init(&tdc_ele, &hspi1, SPI_CS_ELE_GPIO_Port, SPI_CS_ELE_Pin,
    TDC_ELE_ENABLE_GPIO_Port, TDC_ELE_ENABLE_Pin);
TDC_enable(&tdc_ele);

TDC_read(&tdc_ele, TDC_ADR_CONFIG1, data);
data[0] |= TDC_CONFIG1_MEAS_MODE_1;

TDC_write(&tdc_ele, TDC_ADR_CONFIG1, data);

while (1) {
    TDC_start(&tdc_ele);

    // Set Pin to High
    *((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 5.1.2 und 5.1.3 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 4 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 4: Mode 1 vs. Mode 2

Die Berechnung in Formel 14 zeigt, dass die Messresultate nahe beieinander liegen.

$$\begin{aligned} \Delta Mode\ 2 &= 1'403'223.8\ ps - 1'378'222.5\ ps = 25'001.3\ ps \\ Mode\ 1 &= 25'145.3\ ps \end{aligned} \tag{14}$$

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 7.1) kann für beide Modi dieselbe Funktion `TDC_read_result()` verwendet werden. Für Mode 1 vereinfacht sich die Berechnung, weil `TIME2` $= 0$ und `CLOCK_COUNT1` $= 0$.

5.1.5 Timer Output

5.2 Optische Messungen

6 Fazit

7 Anhang

7.1 TDC Treiber

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

```
/*
 * TDC.h
 *
 * Driver for TI TDC7200 -- Header file
 *
 * (C) T. Burkard | M. Schaer
 *
 */

#ifndef TDC_H_
#define TDC_H_

#include <stdint.h>
#include "stm32f0xx_hal.h"

#define TDC_CLOCK_PERIOD_FS 62500000 // 16 MHz in [fs]

/**
 * Convert 24bit buffer received from SPI to integer
 */
#define TDC_24BIT_BUF_TO_INT(buf) ((buf[0] << 16) + (buf[1] << 8) + buf[2])

/**
 * @brief Register addresses of the TDC
 *
 */
typedef enum {
    TDC_ADR_CONFIG1                = 0x00, // Configuration Register 1
    TDC_ADR_CONFIG2                = 0x01, // Configuration Register 2
    TDC_ADR_INT_STATUS             = 0x02, // Interrupt Status Register
    TDC_ADR_INT_MASK               = 0x03, // Interrupt Mask Register
    TDC_ADR_COARSE_CNTR_OVF_H      = 0x04, // Coarse Counter Overflow Value High
    TDC_ADR_COARSE_CNTR_OVF_L      = 0x05, // Coarse Counter Overflow Value Low
    TDC_ADR_CLOCK_CNTR_OVF_H       = 0x06, // CLOCK Counter Overflow Value High
    TDC_ADR_CLOCK_CNTR_OVF_L       = 0x07, // CLOCK Counter Overflow Value Low
    TDC_ADR_CLOCK_CNTR_STOP_MASK_H = 0x08, // CLOCK Counter STOP Mask High
    TDC_ADR_CLOCK_CNTR_STOP_MASK_L = 0x09, // CLOCK Counter STOP Mask Low
    TDC_ADR_TIME1                  = 0x10, // Measured Time 1
    TDC_ADR_CLOCK_COUNT1           = 0x11, // CLOCK Counter Value
    TDC_ADR_TIME2                  = 0x12, // Measured Time 2
    TDC_ADR_CLOCK_COUNT2           = 0x13, // CLOCK Counter Value
    TDC_ADR_TIME3                  = 0x14, // Measured Time 3
    TDC_ADR_CLOCK_COUNT3           = 0x15, // CLOCK Counter Value
    TDC_ADR_TIME4                  = 0x16, // Measured Time 4
    TDC_ADR_CLOCK_COUNT4           = 0x17, // CLOCK Counter Value
    TDC_ADR_TIME5                  = 0x18, // Measured Time 5
    TDC_ADR_CLOCK_COUNT5           = 0x19, // CLOCK Counter Value
    TDC_ADR_TIME6                  = 0x1A, // Measured Time 6
    TDC_ADR_CALIBRATION1           = 0x1B, // Calibration 1, 1 CLOCK Period
    TDC_ADR_CALIBRATION2           = 0x1C, // Calibration 2, 2/10/20/40 CLOCK
    Periods
    TDC_ADR_AMOUNT // Amount of registers
} 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
    after interrupted measurement (for example, due to counter overflow or
    missing STOP signal)
69     TDC_CONFIG1_FORCE_CAL_ON    = (0b1 << 7), // Calibration is always
    performed at the end (for example, after a counter overflow)
70     TDC_CONFIG1_PARITY_EN_OFF   = (0b0 << 6), // Parity bit for Measurement
    Result Registers disabled (Parity Bit always 0)
71     TDC_CONFIG1_PARITY_EN_ON    = (0b1 << 6), // Parity bit for Measurement
    Result Registers enabled (Even Parity)
72     TDC_CONFIG1_TRIGG_EDGE_RISE = (0b0 << 5), // TRIGG is output as a Rising
    edge signal
73     TDC_CONFIG1_TRIGG_EDGE_FALL = (0b1 << 5), // TRIGG is output as a Falling
    edge signal
74     TDC_CONFIG1_STOP_EDGE_RISE  = (0b0 << 4), // Measurement is stopped on
    Rising edge of STOP signal
75     TDC_CONFIG1_STOP_EDGE_FALL  = (0b1 << 4), // Measurement is stopped on
    Falling edge of STOP signal
76     TDC_CONFIG1_START_EDGE_RISE = (0b0 << 3), // Measurement is started on
    Rising edge of START signal
77     TDC_CONFIG1_START_EDGE_FALL = (0b1 << 3), // Measurement is started on
    Falling edge of START signal
78     TDC_CONFIG1_MEAS_MODE_1     = (0b00 << 1), // Measurement Mode 1 (for
    expected time-of-flight < 500 ns).
79     TDC_CONFIG1_MEAS_MODE_2     = (0b01 << 1), // Measurement Mode 2
    (recommended)
80     TDC_CONFIG1_START_MEAS      = (0b1 << 0),  // Start New Measurement.
    Writing a 1 will clear all bits in the Interrupt Status Register and Start
    the measurement (by generating an TRIGG signal) and will reset the content
    of all Measurement Results registers
81 } TDC_config1_t;
82
83 /**
84  * @brief Bit description of the TDC CONFIG2 register
85  *
86  */
87 typedef enum {
88     TDC_CONFIG2_CALIBRATION2_PERIODS_2  = (0b00 << 6),  // Measuring 2 CLOCK
    periods
89     TDC_CONFIG2_CALIBRATION2_PERIODS_10 = (0b01 << 6),  // Measuring 10 CLOCK
    periods
90     TDC_CONFIG2_CALIBRATION2_PERIODS_20 = (0b10 << 6),  // Measuring 20 CLOCK
    periods
91     TDC_CONFIG2_CALIBRATION2_PERIODS_40 = (0b11 << 6),  // Measuring 40 CLOCK
    periods
92     TDC_CONFIG2_AVG_CYCLES_1            = (0b000 << 3), // 1 Measurement Cycle
    only (no Multi-Cycle Averaging Mode)
93     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
        Cycles
100     TDC_CONFIG2_NUM_STOP_1              = (0b000 << 0), // Single Stop
101     TDC_CONFIG2_NUM_STOP_2              = (0b001 << 0), // Two Stops
102     TDC_CONFIG2_NUM_STOP_3              = (0b010 << 0), // Three Stops
103     TDC_CONFIG2_NUM_STOP_4              = (0b011 << 0), // Four Stops
104     TDC_CONFIG2_NUM_STOP_5              = (0b100 << 0), // Five Stops
105 } TDC_config2_t;
106
107 /**
108  * @brief Bit description of the TDC INT_STATUS register
109  *
110  */
111 typedef enum {
112     TDC_STATUS_MEAS_COMPLETE_FLAG  = (1 << 4), // Measurement has completed
113     TDC_STATUS_MEAS_STARTED_FLAG   = (1 << 3), // Measurement has started
        (START signal received)
114     TDC_STATUS_CLOCK_CNTR_OVF_INT  = (1 << 2), // Clock overflow detected,
        running measurement will be stopped immediately
115     TDC_STATUS_COARSE_CNTR_OVF_INT = (1 << 1), // Coarse overflow detected,
        running measurement will be stopped immediately
116     TDC_STATUS_NEW_MEAS_INT        = (1 << 0), // Interrupt detected - New
        Measurement has been completed
117 } TDC_status_t;
118
119 /**
120  * @brief Bit description of the TDC INT_MASK register
121  *
122  */
123 typedef enum {
124     TDC_MASK_CLOCK_CNTR_OVF_MASK_OFF  = (0 << 2), // CLOCK Counter Overflow
        Interrupt disabled
125     TDC_MASK_CLOCK_CNTR_OVF_MASK_ON   = (1 << 2), // CLOCK Counter Overflow
        Interrupt enabled
126     TDC_MASK_COARSE_CNTR_OVF_MASK_OFF = (0 << 1), // Coarse Counter Overflow
        Interrupt disabled
127     TDC_MASK_COARSE_CNTR_OVF_MASK_ON  = (1 << 1), // Coarse Counter Overflow
        Interrupt enabled
128     TDC_MASK_NEW_MEAS_MASK_OFF        = (0 << 0), // New Measurement Interrupt
        disabled
129     TDC_MASK_NEW_MEAS_MASK_ON         = (1 << 0), // New Measurement Interrupt
        enabled
130 } TDC_mask_t;
131
132 /**
133  * @brief Error codes of the TDC module
134  *
135  */
136 typedef enum {
137     TDC_OK = 0,
138     TDC_ERROR,
139     TDC_WRONG_ADDRESS,
140     TDC_WRONG_SIZE,
141     TDC_COM_ERROR,
142 } TDC_error_t;
143
```

```c
/**
 * @brief Handle for a TDC instance
 *
 */
typedef struct {
    SPI_HandleTypeDef* spi;
    GPIO_TypeDef*      cs_port;
    uint16_t           cs_pin;
    GPIO_TypeDef*      en_port;
    uint16_t           en_pin;
} TDC_t;

/**
 * @brief Initialize a TDC instance
 *
 * Note, SPI and GPIOs need to be initialized manually before calling this function.
 *
 * @param[out] tdc     -- Pointer to TDC handle
 * @param[in]  spi     -- Pointer to SPI handle
 * @param[in]  cs_port -- GPIO port of Chip Select
 * @param[in]  cs_pin  -- GPIO pin of Chip Select
 * @param[in]  en_port -- GPIO port of Enable
 * @param[in]  en_pin  -- GPIO pin of Enable
 *
 * @return TDC_error -- Error code
 */
TDC_error_t TDC_init(TDC_t* tdc, SPI_HandleTypeDef* spi, GPIO_TypeDef* cs_port, uint16_t cs_pin, GPIO_TypeDef* en_port, uint16_t en_pin);

/**
 * @brief Enable the TDC
 *
 * @param[in] tdc -- Pointer to TDC handle
 *
 * @return TDC_error_t -- Error code
 */
TDC_error_t TDC_enable(TDC_t* tdc);

/**
 * @brief Disable the TDC
 *
 * @param[in] tdc -- Pointer to TDC handle
 *
 * @return TDC_error_t -- Error code
 */
TDC_error_t TDC_disable(TDC_t* tdc);

/**
 * @brief Read register data from TDC
 *
 * @param[in]  tdc     -- Pointer to TDC handle
 * @param[in]  address -- Address of the register(s) to read
 * @param[out] rx_data -- Pointer to data
 *
 * @attention @p rx_data needs to provide space for at least TDC_REG_SIZE[ @p address ] bytes.
 *
 * @return TDC_error_t -- Error code
 */
TDC_error_t TDC_read(TDC_t* tdc, TDC_adr_t address, uint8_t* rx_data);
```

```
202
203 /**
204  * @brief Write register data to TDC
205  *
206  * @param[in] tdc     -- Pointer to TDC handle
207  * @param[in] address -- Address of the register(s) to write
208  * @param[in] tx_data -- Pointer to data
209  *
210  * @attention @p tx_data needs to provide space for at least TDC_REG_SIZE[ @p
     address ] bytes.
211  *
212  * @return TDC_error_t -- Error code
213  */
214 TDC_error_t TDC_write(TDC_t* tdc, TDC_adr_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. Schaer
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]
```

```
    [TDC_ADR_CONFIG1]                = 1,
    [TDC_ADR_CONFIG2]                = 1,
    [TDC_ADR_INT_STATUS]             = 1,
    [TDC_ADR_INT_MASK]               = 1,
    [TDC_ADR_COARSE_CNTR_OVF_H]      = 1,
    [TDC_ADR_COARSE_CNTR_OVF_L]      = 1,
    [TDC_ADR_CLOCK_CNTR_OVF_H]       = 1,
    [TDC_ADR_CLOCK_CNTR_OVF_L]       = 1,
    [TDC_ADR_CLOCK_CNTR_STOP_MASK_H] = 1,
    [TDC_ADR_CLOCK_CNTR_STOP_MASK_L] = 1,
    [TDC_ADR_TIME1]                  = 3,
    [TDC_ADR_CLOCK_COUNT1]           = 3,
    [TDC_ADR_TIME2]                  = 3,
    [TDC_ADR_CLOCK_COUNT2]           = 3,
    [TDC_ADR_TIME3]                  = 3,
    [TDC_ADR_CLOCK_COUNT3]           = 3,
    [TDC_ADR_TIME4]                  = 3,
    [TDC_ADR_CLOCK_COUNT4]           = 3,
    [TDC_ADR_TIME5]                  = 3,
    [TDC_ADR_CLOCK_COUNT5]           = 3,
    [TDC_ADR_TIME6]                  = 3,
    [TDC_ADR_CALIBRATION1]           = 3,
    [TDC_ADR_CALIBRATION2]           = 3,
};

static TDC_error_t send(TDC_t* tdc, TDC_adr_t address, uint8_t* data, bool
    write);

TDC_error_t TDC_init(TDC_t* tdc, SPI_HandleTypeDef* spi, GPIO_TypeDef* cs_port,
    uint16_t cs_pin, GPIO_TypeDef* en_port, uint16_t en_pin)
{
    if ((tdc == NULL) || (spi == NULL) || (cs_port == NULL) || (en_port ==
    NULL)) {
        return TDC_ERROR;
    }

    tdc->spi     = spi;
    tdc->cs_port = cs_port;
    tdc->cs_pin  = cs_pin;
    tdc->en_port = en_port;
    tdc->en_pin  = en_pin;

    return TDC_OK;
}

TDC_error_t TDC_enable(TDC_t* tdc)
{
    if (tdc == NULL) {
        return TDC_ERROR;
    }

    HAL_GPIO_WritePin(tdc->en_port, tdc->en_pin, GPIO_PIN_SET);

    return TDC_OK;
}

TDC_error_t TDC_disable(TDC_t* tdc)
{
    if (tdc == NULL) {
        return TDC_ERROR;
    }
```

```
82
83      HAL_GPIO_WritePin(tdc->en_port, tdc->en_pin, GPIO_PIN_RESET);
84
85      return TDC_OK;
86  }
87
88  TDC_error_t TDC_write(TDC_t* tdc, TDC_adr_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_adr_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
144     if ((error_code = TDC_read(tdc, TDC_ADR_CLOCK_COUNT1, clock_count1_buf)) !=
        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)) !=
        TDC_OK) {
151         return error_code;
152     }
153
154     calibration1 = TDC_24BIT_BUF_TO_INT(calibration1_buf);
155
156     if ((error_code = TDC_read(tdc, TDC_ADR_CALIBRATION2, calibration2_buf)) !=
        TDC_OK) {
157         return error_code;
158     }
159
160     calibration2 = TDC_24BIT_BUF_TO_INT(calibration2_buf);
161
162     if ((error_code = TDC_read(tdc, TDC_ADR_CONFIG2, config2_buf)) != TDC_OK) {
163         return error_code;
164     }
165
166     if (config2_buf[0] & TDC_CONFIG2_CALIBRATION2_PERIODS_10) {
167         calibration2_periods = 10;
168     } else if (config2_buf[0] & TDC_CONFIG2_CALIBRATION2_PERIODS_20) {
169         calibration2_periods = 20;
170     } else if (config2_buf[0] & TDC_CONFIG2_CALIBRATION2_PERIODS_40) {
171         calibration2_periods = 40;
172     } else { // TDC_CONFIG2_CALIBRATION2_PERIODS_2
173         calibration2_periods = 2;
174     }
175
176     // Calculations
177
178     double cal_count = (double)(calibration2 - calibration1) /
        (calibration2_periods - 1);
179     uint64_t norm_lsb_fs = (uint64_t)(TDC_CLOCK_PERIOD_FS / cal_count);
180
181     *tof_fs = (int64_t)norm_lsb_fs * ((int64_t)time1 - (int64_t)time2) +
        ((int64_t)clock_count1 * TDC_CLOCK_PERIOD_FS);
182
183     return TDC_OK;
184 }
185
186 /**
187  * @brief Read/write register data from/to TDC
188  *
189  * Function for both reading and writing register data, depending on @p write
        parameter.
190  *
191  * @param[in]     tdc     -- Pointer to TDC handle
192  * @param[in]     address -- Address of the register(s) to read/write
193  * @param[in,out] data    -- Pointer to data
194  * @param[in]     write -- True = write, false = read
195  *
196  * @attention @p data needs to provide space for at least TDC_REG_SIZE[ @p
        address ] bytes.
```

```
 *
 * @return TDC_error_t -- Error code
 */
static TDC_error_t send(TDC_t* tdc, TDC_adr_t address, uint8_t* data, bool
    write)
{
    uint8_t size; // number of data bytes
    uint8_t cmd = address | (write ? TDC_CMD_WRITE : TDC_CMD_READ);

    if ((tdc == NULL) || (data == NULL)) {
        return TDC_ERROR;
    }

    if (address >= TDC_ADR_AMOUNT) {
        return TDC_WRONG_ADDRESS;
    }

    size = TDC_REG_SIZE[address];

    HAL_GPIO_WritePin(tdc->cs_port, tdc->cs_pin, GPIO_PIN_RESET);
    HAL_Delay(1); // 1 ms

    if (HAL_SPI_Transmit(tdc->spi, &cmd, 1, HAL_MAX_DELAY) != HAL_OK) {
        return TDC_COM_ERROR;
    }

    if (write) {
        if (HAL_SPI_Transmit(tdc->spi, data, size, HAL_MAX_DELAY) != HAL_OK) {
            return TDC_COM_ERROR;
        }
    } else {
        if (HAL_SPI_Receive(tdc->spi, data, size, HAL_MAX_DELAY) != HAL_OK) {
            return TDC_COM_ERROR;
        }
    }

    HAL_GPIO_WritePin(tdc->cs_port, tdc->cs_pin, GPIO_PIN_SET);
    HAL_Delay(1); // 1 ms

    return TDC_OK;
}
```

Code 7: TDC Driver (Source)

7.2 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.

```
import numpy as np
import matplotlib.pyplot as plt

# Read the data from the log.txt file
with open('log.txt', 'r') as file:
    tof_values = []
    for line in file:
        # Extract the ToF value using the pattern 'ToF = <value> [ps]'
        if 'ToF' in line:
            tof_value = int(line.split('=')[1].split('[')[0].strip())
            tof_values.append(tof_value)
```

```

# Convert the list to a numpy array for easier calculation
tof_array = np.array(tof_values)

# Calculate the arithmetic mean and standard deviation
mean_tof = np.mean(tof_array)
std_tof = np.std(tof_array)

# Print the results
print(f'Arithmetic Mean of ToF: {mean_tof}')
print(f'Standard Deviation of ToF: {std_tof}')

# Plot a histogram
plt.figure(figsize=(10, 6))
plt.hist(tof_array, bins=20, color='skyblue', edgecolor='black', alpha=0.7)
plt.title('Histogram of ToF Values')
plt.xlabel('Time-of-Flight (ps)')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()

# Plot a box plot
plt.figure(figsize=(10, 6))
plt.boxplot(tof_array, vert=False, patch_artist=True,
    boxprops=dict(facecolor='lightblue', color='black'))
plt.title('Box Plot of ToF Values')
plt.xlabel('Time-of-Flight (ps)')
plt.grid(True)
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.

```
import numpy as np
import matplotlib.pyplot as plt

# Load data from logs.txt
filename = 'logs.txt'
data = np.loadtxt(filename, delimiter=';', dtype=float)

# Calculate arithmetic mean and standard deviation for each column
means = np.mean(data, axis=0)
std_devs = np.std(data, axis=0)

# Print results
for i, (mean, std_dev) in enumerate(zip(means, std_devs), start=1):
    print(f"Column {i}: Mean = {mean:.2f}, Standard Deviation = {std_dev:.2f}")

# Plot all columns with points
plt.figure(figsize=(10, 6))
labels = ["0 m", "1 m", "2 m", "4 m", "6 m", "7 m"]
for i in range(data.shape[1]):
    plt.scatter(range(len(data[:, i])), data[:, i], label=labels[i], s=5)

plt.title('ToF for different cable lengths')
plt.xlabel('Index')
plt.ylabel('ToF [ps]')
plt.legend()
plt.grid()
```

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

Code 9: Python Analyse (Multi)

Quellenverzeichnis

Bourns. (2024). *3224W-2-10XE Datasheet.* Zugriff auf https://www.mouser.ch/datasheet/2/54/3224-776900.pdf (aufgerufen am 26.12.2024)

Diodes Inc. (2020). *74LVC1G08Q Datasheet.* Zugriff auf https://www.mouser.ch/datasheet/2/115/DIOD_S_A0010762531_1-2543394.pdf (aufgerufen am 26.12.2024)

Epson Timing. (2024). *SG5032CAN Datasheet.* Zugriff auf https://www.mouser.ch/datasheet/2/137/SG5032CAN_en-961596.pdf (aufgerufen am 26.12.2024)

firewall.xc. (2025). *Propagation Delay.* Zugriff auf https://www.firewall.cx/networking/ethernet/propagation-delay.html (aufgerufen am 12.01.2025)

JRC. (2014). *NJL6401R-3 Datasheet.* Zugriff auf https://www.mouser.ch/datasheet/2/294/NJL6401R_3_E-1019028.pdf (aufgerufen am 26.12.2024)

Nidec Comp. (2024). *CAS-120A1 Datasheet.* Zugriff auf https://www.mouser.ch/datasheet/2/972/cas-1628136.pdf (aufgerufen am 26.12.2024)

Qioptiq. (2024). *G061041000 Datasheet.* Zugriff auf https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/692/G061041000.pdf (aufgerufen am 26.12.2024)

ROHM. (2019). *RLD65NZX1 Datasheet.* Zugriff auf https://fscdn.rohm.com/en/products/databook/datasheet/opto/laser_diode/red/rld65nzx100a008-e.pdf (aufgerufen am 26.12.2024)

ROHM. (2020). *RLD94PZJ5 Datasheet.* Zugriff auf https://fscdn.rohm.com/en/products/databook/datasheet/opto/laser_diode/infrared/rld94pzj5-e.pdf (aufgerufen am 26.12.2024)

ST. (2019). *NUCLEO-F042K6 Datasheet.* Zugriff auf https://www.mouser.ch/datasheet/2/389/nucleo-f031k6-1484037.pdf (aufgerufen am 26.12.2024)

ST. (2020). *Description of STM32F0 HAL.* Zugriff auf https://www.st.com/resource/en/user_manual/dm00122015-description-of-stm32f0-hal-and-low-layer-drivers-stmicroelectronics.pdf (aufgerufen am 12.01.2025)

ST. (2024). *NUCLEO-F042K6 Usermanual.* Zugriff auf https://www.st.com/resource/en/user_manual/um1956-stm32-nucleo32-boards-mb1180-stmicroelectronics.pdf (aufgerufen am 26.12.2024)

TDK. (2019). *MPZ0402S100 Datasheet.* Zugriff auf https://product.tdk.com/system/files/dam/doc/product/emc/emc/beads/catalog/beads_commercial_power_mpz0402_en.pdf (aufgerufen am 26.12.2024)

TI. (2016a). *CSD17578Q3A Datasheet.* Zugriff auf https://www.ti.com/lit/ds/symlink/csd17578q3a.pdf?ts=1735200469410 (aufgerufen am 26.12.2024)

TI. (2016b). *TDC7200 Datasheet.* Zugriff auf http://www.ti.com/lit/gpn/tdc7200 (aufgerufen am 25.12.2024)

TI. (2016c). *TLV3501 Datasheet.* Zugriff auf https://www.ti.com/lit/ds/symlink/tlv3501.pdf?ts=1735168630258 (aufgerufen

am 26.12.2024)

TI. (2018). *OPA858 Datasheet.* Zugriff auf `https://www.ti.com/lit/ds/symlink/opa858.pdf?ts=1735178891996` (aufgerufen am 26.12.2024)

TI. (2024). *LMG1025-Q1 Datasheet.* Zugriff auf `https://www.ti.com/lit/ds/symlink/lmg1025-q1.pdf?ts=1735200095732` (aufgerufen am 26.12.2024)

Vishay Semic. (2024). *BPV23NF Datasheet.* Zugriff auf `https://www.vishay.com/docs/81513/bpv23nf.pdf` (aufgerufen am 26.12.2024)