

# Skript zum Praktikum Labor C Embedded Systems im Fachgebiet Messtechnik

Sommersemester 2022

Stand 10. Juni 2022

<b>Authoren</b>	M.Sc. Pascal Gollor Dr. rer. nat. Michael Diehl Dr.-Ing. Stanislav Tereschenko
<b>Kontakt</b>	pascal.gollor@uni-kassel.de
<b>Fachgebiet</b>	Messtechnik
<b>Fachgebietsleitung</b>	Prof. Dr.-Ing. habil. Peter Lehmann

©2022 Fachgebiet Messtechnik des Fachbereiches Elektrotechnik/Informatik, Universität Kassel

Dieses Dokument darf ausschließlich zu Lehr- und Unterrichtszwecken an der Universität Kassel verwendet werden. Für diese Zwecke ist sowohl seine Nutzung als auch Vervielfältigung ausdrücklich gestattet und erwünscht. Jede anderweitige Nutzung bedarf der schriftlichen Genehmigung des Fachgebietes Messtechnik des Fachbereiches Elektrotechnik/Informatik der Universität Kassel.

# Inhaltsverzeichnis

1	Einleitung	1
1.1	Ablauf . . . . .	2
2	Grundlagen	4
2.1	Mikrocontroller . . . . .	6
2.1.1	GPIO . . . . .	7
2.1.2	Timer . . . . .	8
2.2	Analog-Digital-Wandler . . . . .	8
2.3	Digital-Analog-Wandler . . . . .	10
2.4	Pulsweitenmodulation . . . . .	10
2.5	Direct Memory Access . . . . .	12
2.6	Übertragungsprotokolle . . . . .	13
2.6.1	UART . . . . .	13
2.6.2	I <sup>2</sup> C . . . . .	14
2.6.3	1-Wire BUS . . . . .	15
2.7	Peripherie . . . . .	16
2.7.1	Widerstände . . . . .	17
2.7.2	I <sup>2</sup> C-Display . . . . .	18
2.7.3	Temperatursensor . . . . .	18
2.7.4	Encoder . . . . .	19
2.7.5	Lüfter . . . . .	20
2.7.6	IR-Detektor . . . . .	20
2.7.7	RC-Tiefpassfilter . . . . .	20
2.8	Vorbereitungsaufgaben . . . . .	23
2.9	Informationsübersicht . . . . .	24
2.9.1	Video Tutorials . . . . .	24
3	Software-Tools	25
3.1	Beschreibung, Installation und Einrichtung . . . . .	26
3.1.1	STM32CubeIDE . . . . .	26

3.1.2	ST-Link Utility . . . . .	28
3.1.3	HTerm . . . . .	29
3.2	Pfade . . . . .	30
3.3	Erstes Projekt . . . . .	31
3.4	Bibliotheken . . . . .	36
<b>4</b>	<b>Aufgaben</b>	<b>37</b>
4.1	Block 1: Ein- und Ausgaben . . . . .	38
4.1.1	Blinkende LED . . . . .	38
4.1.2	UART . . . . .	40
4.1.3	Display . . . . .	41
4.2	Block 2: Sensoren . . . . .	42
4.2.1	Temperaturmessung mit DHT11 . . . . .	42
4.2.2	Encoder . . . . .	43
4.2.3	ADC und DAC . . . . .	45
4.3	Block 3: PWM-Ansteuerung / Drehzahlmessung . . . . .	46
4.3.1	Lüfter-Ansteuerung . . . . .	46
4.3.2	Drehzahlmessung mit Tacho-Signal . . . . .	47
4.4	Block 4 - Abschlussaufgabe: Signalerzeugung mittels PWM und DAC sowie Signalaufzeichnung mit einen ADC. . . . .	48
	<b>Literaturverzeichnis</b>	<b>53</b>
	<b>Abkürzungen</b>	<b>56</b>
<b>A</b>	<b>Datenblätter</b>	<b>i</b>
A.1	TCRT5000 . . . . .	i
A.2	DHT11 . . . . .	vi
A.3	STM32F446 Blockdiagramm . . . . .	xv

## 1 Einleitung

Das Praktikum **Labor C Embedded Systems im Fachgebiet Messtechnik** ist Bestandteil des Moduls **Labor C/Embedded Systems**, mit dem Ziel grundlegende Kenntnisse in der hardwarenahen Programmierung mit Hilfe der Programmiersprache C, mit messtechnischem Anwendungsbezug zu vermitteln. Die theoretischen Grundlagen zur Programmierung in C werden in diesem Skript nicht behandelt und sollen durch den Vorlesungsteil des Moduls erworben werden. Alle weiteren Grundlagen werden im Kapitel 2 erwähnt, jedoch nicht vollumfänglich erläutert. Für weitergehende Informationen wird an den entsprechenden Stellen auf geeignete Fachliteratur verwiesen. Die genannten Grundlagen sollen vor Beginn des Praktikumsteils durchgearbeitet werden. Dies ist einerseits erforderlich, um zum Praktikum zugelassen zu werden, anderseits wird damit die Grundlage geschaffen das Praktikum in der vorgegebenen Zeit erfolgreich abzuschließen. Des Weiteren wird empfohlen, die Inhalte aus der Lehrveranstaltung "Elektrotechnik für Informatiker" aufzufrischen und den Umgang mit einem Multimeter und einem Oszilloskop zu wiederholen.

Das Praktikum **Labor Embedded Systems im Fachgebiet Messtechnik** findet in diesem Sommersemester zum zweiten Mal statt. Dementsprechend fehlt uns bisher das Feedback über die Vollständigkeit und den Umfang der Vorbereitungsinformationen, den Schwierigkeitsgrad und die Verständlichkeit der Aufgaben, sowie mögliche Fehler in den Aufgabenstellungen und Grundlageninformationen. Wir würden uns daher über jede Art konstruktiver Kritik, nützlicher Hinweise und Verbesserungsvorschläge Ihrerseits freuen. Dementsprechend könnte es vorkommen, dass wir die Unterlagen im Laufe des Praktikums aktualisieren. Bei möglichen Änderungen werden wir Sie über Moodle informieren und die aktualisierte Version des Skripts bereitstellen.

## 1.1 Ablauf

Der Umfang des Praktikums beträgt 120 Stunden. Dies umfasst den kompletten Arbeitsaufwand der sich aus der Präsenzzeit des Praktikums im Fachgebiet Messtechnik und aus Heimarbeit bzw. Selbststudium zusammensetzt. Einige der Aufgaben können jedoch nicht ohne das Equipment im Praktikumslabor gelöst werden. **Zu Beginn des Praktikums sind bis zum ersten Termin die Fragen am Ende des Grundlagenkapitels 2.8 zu beantworten und ausformuliert vorzuzeigen.** Aufgrund der aktuellen Lage wird es vermutlich nicht möglich sein in einer Gruppe zu arbeiten, sondern alle Aufgaben müssen in Einzelarbeit erledigt werden. Es spricht aber nichts dagegen, wenn Sie sich gegenseitig absprechen solange kein Code kopiert wird.

Um auch außerhalb der Termine in der Universität weiter an Ihrer Software zu arbeiten wird Ihnen die benötigte Hardware zur Verfügung gestellt. Darüber hinaus enthält dieses Skript Informationen (Quellen) zum Bezug der benötigten Software. Um das Bearbeiten der Aufgaben einfacher zu gestalten, empfiehlt es sich einen eigenen Laptop zum Praktikum mitzubringen und ggf. ein Versionsverwaltungssystem wie `git` zu nutzen. Dazu wird Ihnen bei der Einführungsveranstaltung des Praktikums eine Zugangsmöglichkeit zum fachgebietseigenen `git`-Server zur Verfügung gestellt. Dieser Server ist ausschließlich für Ihre Arbeit in diesem Praktikum zu nutzen. Bei Missbrauch behält sich das FG Messtechnik das Recht vor, den Zugang ohne vorherige Ankündigung zu deaktivieren.

Die zu bearbeitenden Aufgaben sind in 4 Blöcke eingeteilt und ausführlich im Kapitel 4 beschrieben. Am Ende jedes Aufgabenblocks ist ein mündliches Testat abzulegen, mit Ausnahme des Blocks 4.

Ein solches Testat beinhaltet die Vorführung der Aufgaben, sowie eine Erläuterung des Programmcodes und der Erklärung, wie und warum die Aufgabe auf diese Art und Weise bearbeitet wurde. **Dieses Testat wird bewertet und geht in die Gesamtnote für das Praktikum anhand folgender Bewertungskriterien ein:**

- Erreichen der Aufgabenziele
- Programmierstil
- sinnvolle Variablenbenennung und Mikrocontroller-Peripherieausnutzung
- Zuverlässigkeit und Reproduzierbarkeit der Ergebnisse
- Quellcodedokumentation

Der letzte Aufgabenblock beinhaltet eine komplexere Aufgabe, in der die erworbe-

nen Kenntnisse aus den vorherigen Aufgaben vorausgesetzt werden. Dazu ist kein Testat, sondern eine schriftliche Ausarbeitung anzufertigen. In der Ausarbeitung sollte die Vorgehensweise bei der Bearbeitung der Aufgaben beschrieben werden. Es sollte auf die verwendeten Hardwarekomponenten, Software-Konstrukte und ggf. Bibliotheken eingegangen sowie erklärt werden, wie und auf welche Weise die gestellte Aufgabe umgesetzt wurde und welche Ergebnisse erzielt wurden. Auch Schwierigkeiten oder Komplikationen bei der Umsetzung können mit aufgenommen werden. Außerdem soll der relevante Programmcode (nicht der automatisch generierte) in die Ausarbeitung integriert und beschrieben werden. Weiterhin gelten bei der Bewertung die vorher beschriebenen Kriterien für die Testate.

Nach der Abgabe der Ausarbeitung wird eine Gesamtnote gebildet, die sich aus der Bewertung der Vorbereitungsfragen zu Beginn des Praktikums, der Bewertung der drei Testate und der Ausarbeitung in einem Verhältnis von 1/6, 3/6 und 2/6 zusammensetzt.

Das Fachgebiet Messtechnik wünscht Ihnen viel Erfolg und Spaß bei dem Praktikum.

## 2 Grundlagen

### Inhalt des Kapitels

2.1	Mikrocontroller . . . . .	6
2.1.1	GPIO . . . . .	7
2.1.2	Timer . . . . .	8
2.2	Analog-Digital-Wandler . . . . .	8
2.3	Digital-Analog-Wandler . . . . .	10
2.4	Pulsweitenmodulation . . . . .	10
2.5	Direct Memory Access . . . . .	12
2.6	Übertragungsprotokolle . . . . .	13
2.6.1	UART . . . . .	13
2.6.2	I <sup>2</sup> C . . . . .	14
2.6.3	1-Wire BUS . . . . .	15
2.7	Peripherie . . . . .	16
2.7.1	Widerstände . . . . .	17
2.7.2	I <sup>2</sup> C-Display . . . . .	18
2.7.3	Temperatursensor . . . . .	18
2.7.4	Encoder . . . . .	19
2.7.5	Lüfter . . . . .	20
2.7.6	IR-Detektor . . . . .	20

2.7.7	RC-Tiefpassfilter	20
2.8	Vorbereitungsaufgaben	23
2.9	Informationsübersicht	24
2.9.1	Video Tutorials	24

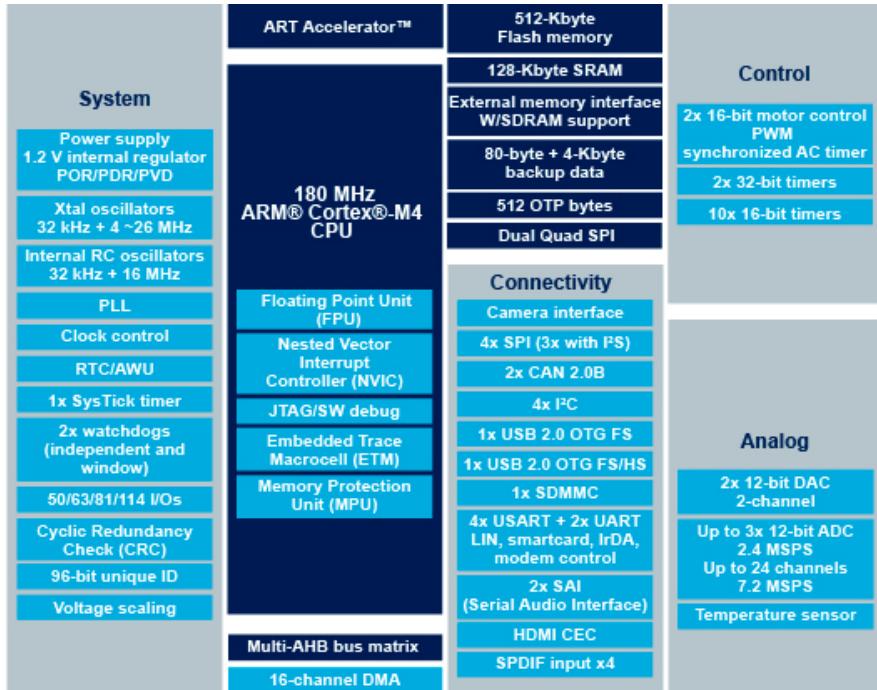
Neben den bereits in dem Vorlesungsteil dieses Moduls vermittelten programmier-technischen Inhalten werden im Folgenden vor allem elektrotechnische Grundlagen eingeführt, welche für die Bearbeitung der Aufgaben notwendig sind. Zunächst erfolgt eine Einführung in die Grundlagen des verwendeten Mikrocontrollers. Im Anschluss werden verschiedene Hardwarekomponenten, die im Controller verbaut sind, erläutert sowie die genutzten Schnittstellen beschrieben. Weiterführende Informationen können online über die Universitätsbibliothek aus folgenden Quellen bezogen werden:

- [Mikrocontroller und Mikroprozessoren \[5\]](#)
- [Schaltungstechnik \[10\]](#)
- [Anwendungsorientierte Mikroprozessoren \[6\]](#)
- [Digitaltechnik \[7\]](#)
- [Grundlagen der Hard- und Software der Mikrocontroller ATtiny2313, ATtiny26 und ATmega32 \[2\]](#)

## 2.1 Mikrocontroller

Der im Praktikum verwendete Controller ([STM32F446RET6](#)) ist ein 32-Bit Mikrocontroller der ARM Cortex-M4 Familie des Herstellers STMicroelectronics. Abb. 2.1 gibt einen Überblick über die im Chip verbauten Komponenten, auf die der Controller zurückgreifen kann. Hauptsächlich seien hier der ADC, DAC, DMA, sowie die Schnittstellen I<sup>2</sup>C und UART erwähnt, die im Laufe dieses Kapitels genauer erläutert werden. Dazu ist dringend das Reference Manual [\[20\]](#) zum Controller zu empfehlen, in dem sehr detaillierte Informationen über die Peripherie und den Betrieb des Mikrocontrollers enthalten sind. **Dieses Handbuch werden Sie während der Bearbeitung der Aufgaben benötigen, um Spezifikationen der oben genannten Komponenten nachzulesen und zu verstehen!** Die wichtigsten Informationen zum Mikrocontroller sind die Größe des vorhandenen Flash-Speichers mit 512 kB, des Arbeitsspeichers mit 128 kB, der maximal möglichen Taktrate von 180 MHz und die Nutzung einer Floating Point Unit (FPU), die eine effiziente Verarbeitung von Gleitkommazahlen ermöglicht. Der Controller wird in C über eine integrierte Entwicklungsumgebung (Integrated Development Environment (IDE)) programmiert, die in Kapitel 3 beschrieben wird.

Wie eingangs erwähnt werden verschiedene Komponenten des Controllers genutzt, die im Folgenden erläutert werden. Die genauen Spezifikationen dazu können im Datenblatt [\[12\]](#) zum Controller nachgelesen werden. Das Blockdiagramm aus dem



**Abbildung 2.1:** Überblick über die verfügbaren Komponenten des STM32F446RE. [Bildquelle](#) vom 27.03.2020

Datenblatt visualisiert die innere logische Verknüpfung der einzelnen Komponenten und ist in Anhang A.3 zu finden.

### 2.1.1 GPIO

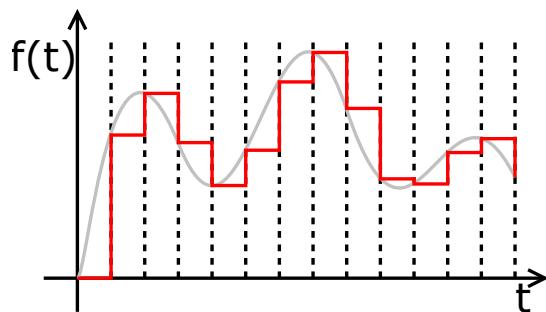
General Purpose Input/Output (GPIO) sind Ein- und Ausgänge die multifunktional genutzt werden können, jedoch einigen Beschränkungen unterliegen, die im Datenblatt aufgeführt werden [12, Kap. 6]. Die einfachste Verwendung ist eine Nutzung als Ausgang, welcher z.B. einen Transistor oder eine LED ansteuert, oder als Eingang für einen Taster. Es sind aber auch sogenannte alternative Funktionen möglich, welche die Aufschaltung eines analogen Eingangs für den ADC (Abschnitt 2.2) oder einen Timerausgang (Abschnitt 2.1.2) für eine Pulsweitenmodulation (Abschnitt 2.4) möglich machen. Weiterführende GPIO Informationen können in [16] nachgelesen werden.

### 2.1.2 Timer

Ein Timer kann verschiedene Funktionen übernehmen. Grundsätzlich zählt ein Timer von einer Zahl hoch oder runter bis zu einer anderen Zahl und kann beim Erreichen der Endzahl ein sogenanntes Event auslösen. Falls kein Ende spezifiziert wird, läuft der Timer nach Erreichen der maximal möglichen Zahl über. Bei den STM32F4 Controllern entspricht das je nach verwendetem Timer 16 oder 32 Bit. Das Zählen erfolgt immer mit dem gleichen Takt, welcher variabel eingestellt werden kann. Dazu wird der Systemtakt über verschiedene **Prescaler** geteilt, um einen langsameren Takt zu erreichen. Der vom Timer ausgelöste Event kann z.B. genutzt werden, um den Zustand eines GPIO zu ändern und somit z.B. eine Pulsweltenmodulation (PWM) zu realisieren oder einen Taktausgang für eine Schnittstelle zur Verfügung zu stellen. Nähere Informationen sind dem Datenblatt des Mikrocontrollers zu entnehmen [12, Kap. 3.21]. Grundsätzliche Beschreibungen sind in [5, Kap. 4.3] zu finden. Weiterführende Informationen können auch der Application Note AN4776 [15] von ST entnommen werden.

### 2.2 Analog-Digital-Wandler

Mit Hilfe eines Analog-Digital-Wandlers (ADCs) ist es möglich zeit- und wertkontinuierliche Signale in zeit- und wert-diskrete Signale umzuwandeln. Die daraus resultierenden Informationen können anschließend in digitaler Form verarbeitet werden. Dazu wird ein analoges Spannungssignal kontinuierlich über die Zeit abgetastet und der Spannungswert zum jeweiligen Zeitpunkt digital gespeichert. In Abb. 2.2 ist in grau ein kontinuierlicher Signalverlauf dargestellt, der an den Stellen der gestrichelten Linien abgetastet wird. Der Verlauf in rot entspricht dabei den



**Abbildung 2.2:** Beispiel einer Diskretisierung (in rot) eines kontinuierlichen Signalverlaufs (in grau). [Bildquelle](#) vom 08.04.2019

digitalen Werten über die Zeit. Die wesentlichen Parameter eines ADCs sind die Auflösung  $2^n$  (n: Bit) und die Abtastdauer bzw. die maximale Abtastrate des ADCs. Der kleinste mögliche Schritt wird als Least Significant Bit (LSB) bezeichnet. Die Zuordnung eines Spannungswertes wird in der Regel binär betrachtet und immer in Relation zu einer Referenzspannung angegeben. Das folgende Beispiel soll dies deutlich machen.

Ein ADC mit einer Auflösung von 3 Bit kann  $2^3 = 8$  verschiedene Werte annehmen, wobei der Wert 0 einer Spannung von 0 V entspricht. Die 8 verschiedenen Werte werden gleichmäßig auf den gesamten Spannungsbereich der Referenzspannung aufgeteilt [10, Kap. 9.4], in diesem Beispiel  $U_{ref} = 2 \text{ V}$ . Der Abstand zwischen zwei möglichen Spannungswerten entspricht

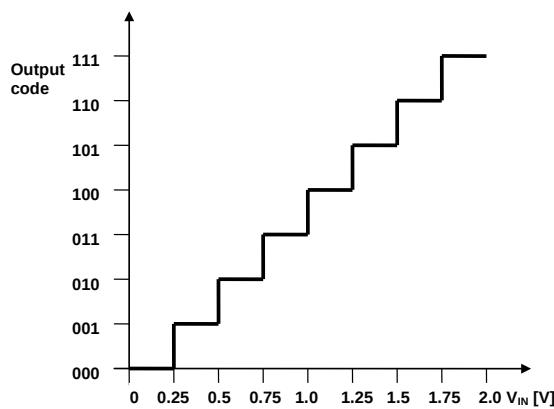
$$U_{LSB} = \frac{2 \text{ V}}{2^3} = 0,25 \text{ V}. \quad (2.1)$$

Das heißt, es können Spannungen von 0 V bis 1,75 V im Abstand von 0,25 V gemessen werden.

Durch solch eine Digitalisierung analoger Daten entstehen unweigerlich Fehler.

## Fehlerquellen

Eine nicht zu eliminierende Fehlerquelle ist der Quantisierungsfehler. Im Gegensatz zu einem idealen ADC hat der reale ADC eine treppenförmige Übertragungsfunktion wie in Abb. 2.3 zu erkennen ist. Daraus resultiert ein maximaler Quantisierungs-

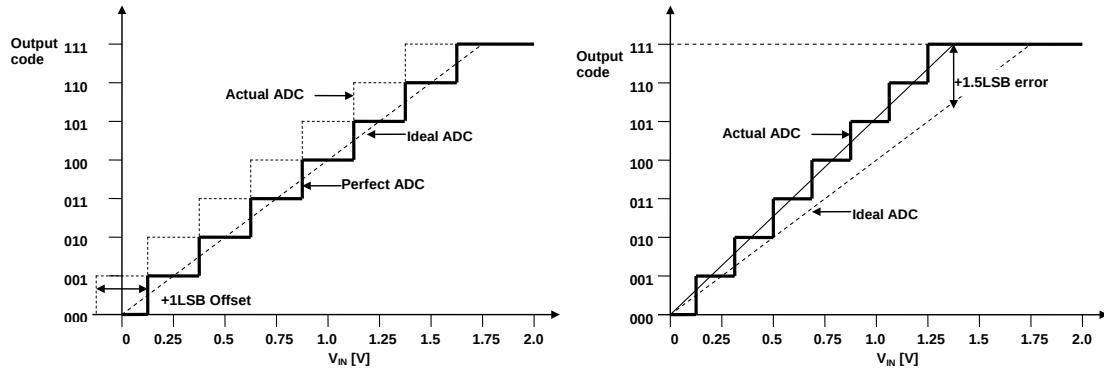


**Abbildung 2.3:** Übertragungsfunktion eines 3 Bit ADCs mit  $LSB = 0,25 \text{ V}$  [1].

fehler von  $f_{q,max} = \pm 0,5 \text{ LSB}$  (in Abb. 2.3  $f_{q,max} = \pm 0,125 \text{ V}$ ).

Darüber hinaus gibt es den sogenannten Offsetfehler (linker Verlauf in Abb. 2.4).

Bei diesem Fehler hat der Messwert im Vergleich zum wahren Wert einen konstanten Offset. Außerdem kann die Steigung der Übertragungsfunktion bei einem realen ADC leicht von der Steigung eines idealen ADCs abweichen und wird als Steigungsfehler (gain error) angegeben.



**Abbildung 2.4:** Mögliche Fehler eines ADC [1]. Links Offset-Fehler und rechts Steigungsfehler.

## 2.3 Digital-Analog-Wandler

Der zuvor beschriebene ADC wird sehr häufig in der Signalverarbeitung eingesetzt. Sein Gegenstück ist der Digital-Analog-Wandler (DAC), dieser wandelt ein digitales Signal in eine analoge Spannung um. Da das Verhalten dem des ADCs ähnelt wird hier für weitere Informationen auf [10, Kap. 9.2] verwiesen. Mit der Umsetzung der digitalen Werte in eine Spannung über die Zeit, kann z.B. ein Sinusverlauf erzeugt werden, der annähernd einem kontinuierlichen Verlauf entspricht. Nähere Informationen gibt es außerdem in einem Beispiel direkt zum Mikrocontroller [14], in dem **sehr gut erklärt** wird, wie und wozu der DAC benutzt werden kann.

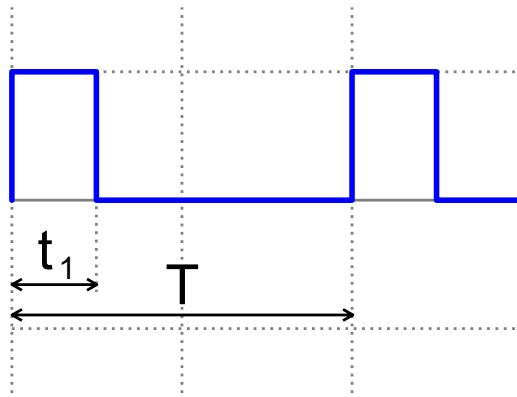
## 2.4 Pulsweitenmodulation

Die Pulsweitenmodulation (PWM) ist neben dem DAC eine weitere Möglichkeit aus einer digitalen Schaltung ein analoges Signal zu generieren. Im Allgemeinen wechselt eine PWM zwischen zwei Werten. Bei einem digitalen Ausgang eines Mikrocontrollers entspricht dies in der Regel der Spannung 0 V und der Betriebsspannung

(logisch 0 und logisch 1). Die PWM wird dabei mit einer konstanten Frequenz  $f_{PWM}$  bzw. der entsprechenden Periodendauer

$$T = \frac{1}{f_{PWM}} \quad (2.2)$$

getaktet. Innerhalb der Periodendauer wird der Ausgang zu einem bestimmten Zeitpunkt geändert. Wird über die eine Periode integriert, resultiert je nach Länge des HIGH-Pegels eine bestimmte Spannung. Oder anders ausgedrückt wird die Spannung über eine Periode des PWM-Taktes gemittelt. In Abb. 2.5 ist eine PWM über eine Periode  $T$  aufgetragen, wobei die Länge der logischen 1 der Zeit  $t_1$  entspricht. Aus den beiden genannten Größen kann der Tastgrad, auch **duty cycle**



**Abbildung 2.5:** Beispiel einer PWM Periode mit der Periodendauer  $T$  und dem Tastgrad  $T_g = \frac{t_1}{T} = 25\%$ . [Bildquelle](#) vom 08.04.2019

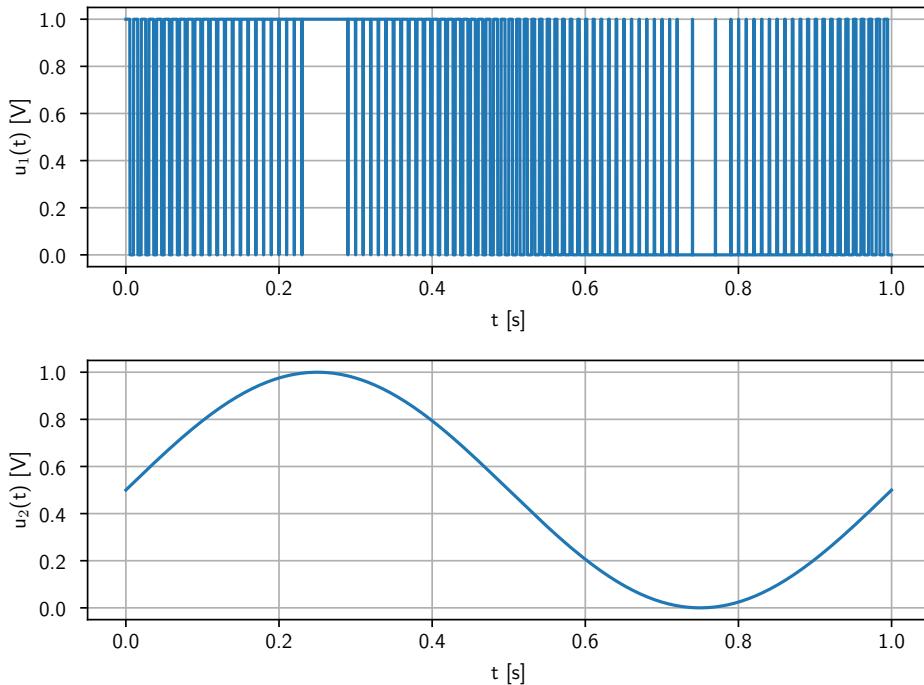
genannt

$$T_g = \frac{t_1}{T} \quad (2.3)$$

in % bestimmt werden. Daraus ist recht einfach ersichtlich, dass die Zeit  $t_1$  den Tastgrad maßgeblich beeinflusst, da die Periodendauer  $T$  konstant ist. Neben dem **duty cycle** ist die PWM-Auflösung ein wichtiger Begriff und beschreibt, wie genau man den Tastgrad einstellen kann. Bei einer Auflösung von z.B. 3 Bit können 8 verschiedene Werte für den **duty cycle** gewählt werden.

Wird ein Verlauf wie in Abb. 2.5 einem geeigneten tragen System, z.B. einem Tiefpassfilter, als Eingang vorgegeben so folgt am Ausgang ein analoges Spannungssignal. Für das oben genannte Beispiel mit einem Tastgrad von 25 % und einer Betriebsspannung von 3 V beträgt der Mittelwert einer Periode 0,75 V. Wenn die Zeit  $t_1$  für jede weitere PWM-Periode einen anderen Wert annimmt, kann somit ein kontinuierlicher Signalverlauf nachgebildet werden. Als Beispiel ist in Abb. 2.6 oben ein PWM-Signal mit einer Frequenz von 10 kHz über mehrere Pe-

rioden dargestellt. Vergleicht man dies direkt mit dem unteren Verlauf in der Abbildung, so wird der Zusammenhang recht gut deutlich. Theoretisch sind nicht



**Abbildung 2.6:** PWM Beispiel: Oberes Diagramm zeigt eine PWM mit einer Frequenz von 10 kHz und das untere Diagramm entspricht einem gefilterten Verlauf mit einem idealen Tiefpassfilter.

nur Sinus- oder Kosinusverläufe umsetzbar, sondern z.B. auch ein Sägezahn oder eine Dreiecksspannung. In der Praxis ist dafür aber eher ein DAC geeignet, da der DAC im Gegensatz zur PWM keinen Tiefpassfilter oder ähnliches benötigt, um den analogen Signalverlauf zu erzeugen. Eine geeignete analoge Schaltung zum Erzeugen eines kontinuierlichen Verlaufs aus einem PWM-Signal wird in Kapitel 4.4 vorgestellt.

## 2.5 Direct Memory Access

Der Datenaustausch eines Mikrocontroller wird oft vom Hauptprozessor durchgeführt. Wenn dieser aber z.B. durch einen Interrupt oder ähnliches unterbrochen wird, ist ein kontinuierlicher Datenaustausch zwischen den einzelnen Komponenten nicht gewährleistet. Aber gerade DAC oder ADC sind oft auf kontinuierlichen Datenaustausch angewiesen. Um dies zu gewährleisten, haben viele Mikrocontroller einen Direct Memory Access (DMA) Controller der nach Initialisierung direkt

Daten ohne Zutun des Prozessorkerns übermitteln kann [5, Kap. 4.6]. Dazu wird z.B. bei einem DAC der DMA so konfiguriert, dass er aus einem Speicherbereich, in welchem eine diskrete Tabelle für eine Sinusschwingung liegt, kontinuierlich Daten ausliest und an den DAC weiterleitet, damit dieser die Werte in eine analoge Spannung umsetzen kann. In dem genannten Szenario ist es nach Initialisierung der Komponenten nicht mehr wichtig, mit welchen Aufgaben der Prozessor beschäftigt ist, denn der Datenaustausch erfolgt unabhängig. Zusätzliche Informationen zum DAC werden von ST bereitgestellt [11].

## 2.6 Übertragungsprotokolle

Neben den bereits beschriebenen Komponenten werden drei verschiedene Kommunikationsschnittstellen im Praktikum verwendet, um Daten mit externen Bausteinen (Komponenten) oder einem Computer auszutauschen. Zwei davon (UART und I<sup>2</sup>C) sind in der Controllerhardware umgesetzt, der 1-Wire BUS muss per Software implementiert werden.

### 2.6.1 UART

Der Universal Asynchronous Receiver Transmitter (UART) ist eine asynchrone serielle Datenübertragung über zwei Leitungen RX und TX, um somit unabhängig voneinander Daten senden und empfangen zu können. RX steht für Receiver exchange (empfangen) und TX für Tranceiver exchange (senden), immer aus der Sicht des jeweiligen Bausteins. Die TX-Leitung eines Mikrocontrollers muss also an die RX-Leitung des Computers angeschlossen werden und umgekehrt. Neben den beiden erwähnten Datenleitungen ist keine weitere Taktleitung vorhanden. Daher wird diese Übertragung asynchron [7, Kap. 14.7.3.1] genannt. Da die Daten mit einer definierten Frequenz übertragen werden, muss diese sogenannte Baudrate [25] Empfänger und Sender bekannt sein. Theoretisch ist die Baudrate frei wählbar, wenn beide Seiten diese kennen und unterstützen. Folgende Baudraten sind in der Praxis gängigen und sollten im Praktikum genutzt werden:

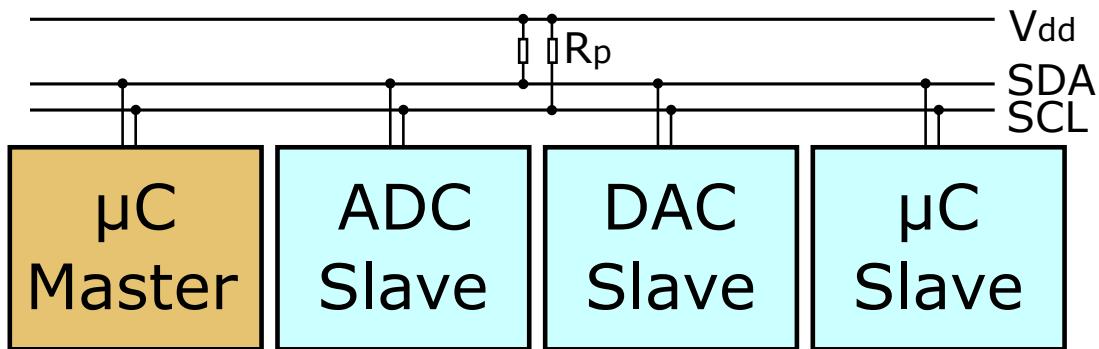
- 9.600 Bit/s
- 57.600 Bit/s
- 115.200 Bit/s
- 500.000 Bit/s

Dabei gilt, je höher die Baudrate umso schneller ist auch die Datenübermittlung. Da wie bereits erwähnt die Daten ohne extra Takteleitung übertragen werden, muss die Datenleitung auf der Empfangsseite mit einer vielfach höheren Frequenz abgetastet werden. Dabei wird oft eine 8- oder 16-fach höhere Abtastrate als die Baudfrequenz genutzt.

Die Übertragung der Informationen kann in verschiedenen Modi erfolgen. Der gebräuchlichste Modus beinhaltet das Senden von einem Startbit, 8 Datenbits, einem Stopabit, ohne Paritätsbit und wird häufig mit **8N1** abgekürzt. Ein komplettes Sendepaket im **8N1** Modus beinhaltet demnach 10 Bits. Eine Übersicht der UART-Interfaces des **STM32F446**-Mikrocontrollers ist in [12], Abschnitt 3.23 Universal synchronous/asynchronous receiver transmitters (USART), aufgeführt.

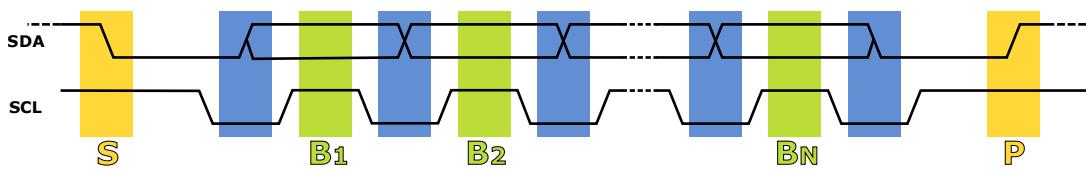
## 2.6.2 I<sup>2</sup>C

Die Inter-Integrated Circuit (I<sup>2</sup>C) Schnittstelle ist ein Datenbus, der aus zwei Datenleitungen besteht und nach dem Master-Slave-Prinzip arbeitet. Eine der beiden Leitungen ist die Takteleitung und wird meist mit **SCL** beschrieben, die andere Leitung ist die bidirektionale Datenleitung und wird oft mit **SDA** bezeichnet. Bei solch einem Master-Slave-Prinzip gibt es einen Master auf dem Bus, der die Kommunikation steuert und den Takt vorgibt. Typische Taktfrequenzen liegen zwischen 100 kHz und 400 kHz. In Abb. 2.7 sind exemplarisch ein Mikrocontroller als Master und drei Slaves dargestellt. Die beiden Widerstände  $R_p$  müssen einmal pro I<sup>2</sup>C-Bus vorhanden sein, da sowohl der Master als auch der Slave die Leitungen zur Kommunikation auf Masse ziehen. Die Slaves antworten nur, wenn sie vom



**Abbildung 2.7:** Exemplarische Darstellung des I<sup>2</sup>C-Bus mit einem Mikrocontroller als Master, drei Slaves und  $R_p$  als Pull-Up-Widerstände. **SDA** entspricht der Datenleitung und **SCL** ist der Taktekanal. [Bildquelle](#) vom 05.04.2019

Master über eine eindeutige Adresse angesprochen wurden. Dabei besteht die Adresse in der Regel aus 7 Bits und ein ganzes Datenpaket aus 8 Bit, also einem Byte. Vor und nach dem Datenpaket folgen sogenannte Start bzw. Stopppbits. Der Start wird durch die erste fallende Flanke auf der Datenleitung und High-Pegel auf der Takteleitung signalisiert. Analog dazu erfolgt das Stopppbit, jedoch bei steigender Flanke auf der Datenleitung. In Abb. 2.8 ist eine Datenübertragung mit N Bits dargestellt. Ein Flankenwechsel auf der Datenleitung soll immer bei Low-Pegel auf der Takteleitung erfolgen, damit nach der steigenden Flanke von SCL, High-Pegel von SDA für eine logische 1 und Low-Pegel von SDA für eine logische 0 steht. Zur



**Abbildung 2.8:** Datenübertragung von N-Bits in grün auf dem I<sup>2</sup>C-Bus mit Start- und Stopppbit in gelb und den Flankenwechseln in blau. [Bildquelle](#) vom 05.04.2019

Adressierung stehen, wie bereits erwähnt 7 Bit zur Verfügung. Dies sind die ersten 7 Bits und das höchste Bit ist das sogenannte R/W-Bit das dem Slave signalisiert, ob die Daten geschrieben oder gelesen werden sollen. Neben den Daten-, Start- und Stopppbits gibt es noch ein ACK Bit, das entweder als Antwort vom Slave gesendet wird, wenn Daten geschrieben werden, oder vom Master, wenn Daten vom Slave gelesen werden. Das ganze Protokoll muss jedoch nicht händisch implementiert werden, da die meisten Mikrocontroller bereits eine Hardwareumsetzung dieser Schnittstelle besitzen, die direkt über Register auf den Controllern angesprochen werden kann. Die in diesem Praktikum verwendete HAL-Bibliothek von ST stellt dafür bereits Funktionen zur Verfügung.

### 2.6.3 1-Wire BUS

Wie der Name bereits vermuten lässt setzt der 1-Wire BUS auf nur eine einzige Leitung zur Datenübertragung. In der Regel wird diese Leitung durch einen Pull-Up-Widerstand auf Betriebsspannung gezogen. Die Verbindung arbeitet wie bei den Schnittstellen zuvor seriell und bidirektional (wie die I<sup>2</sup>C-Schnittstelle) sowie asynchron (wie die UART-Schnittstelle). Da eine Vielzahl von verschiedenen Sensoren existieren, die oft nach nicht standardisierten Protokollen Daten über einen 1-Wire BUS versenden, wird darauf nicht weiter eingegangen. Im Unterkapitel 2.7.3

wird der spezielle 1-Wire BUS für den dort verwendeten Temperatursensor genauer beschrieben.

## 2.7 Peripherie

Bisher wurden einige allgemeine Grundlagen zu verschiedenen Komponenten eines Mikrocontrollers sowie verschiedene Übertragungsmöglichkeiten vorgestellt. Dieses Unterkapitel beschreibt im Detail die verwendete Peripherie bzw. Komponenten, welche bei der Aufgabenlösung zu verwenden sind. Der eingangs genannte Controller ist auf einem [Nucleo-64 Board](#) (siehe Abb. 2.9) verbaut, das zusätzlich noch eine Programmierschnittstelle, einen UART zu USB Wandler, eine ansteuerbare LED, einen frei nutzbaren Taster und einen Reset-Taster enthält. Außerdem sind fast alle

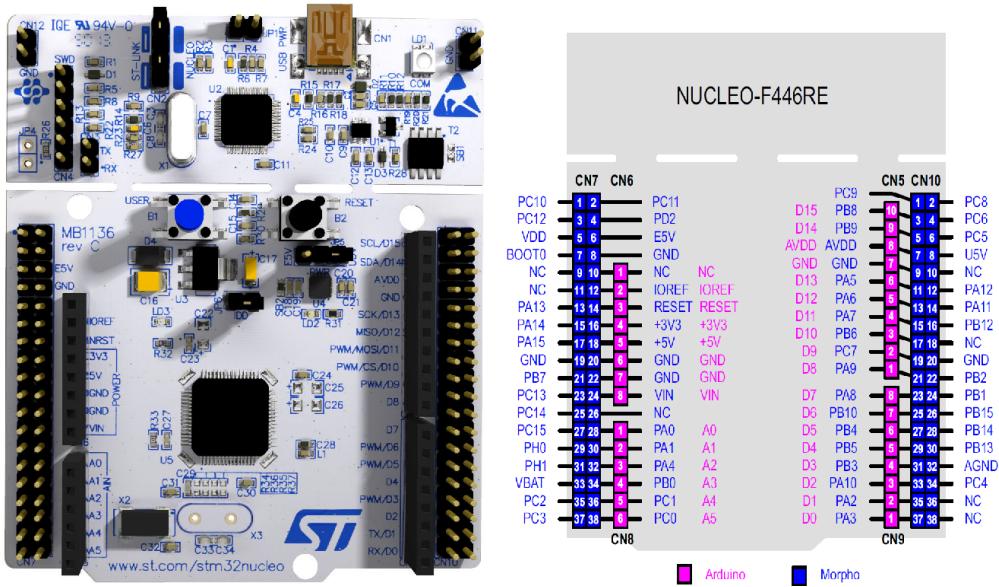
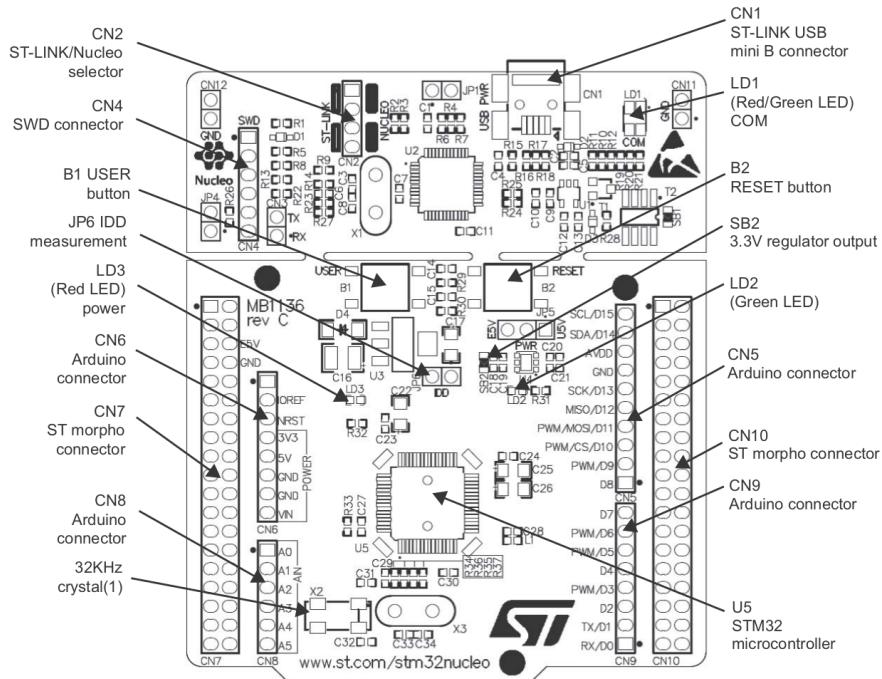


Abbildung 2.9: Foto des Nucleo-64 Boards und Zuordnung der Pinheader [17, Sn. 1,35].

Anschlüsse (GPIO) des Mikrocontrollers über sogenannte Pinheader nach außen geführt (Abb. 2.9, rechts), um diese verwenden zu können. Nähere Informationen sind [13, 17] und dem Anhang A.3 zu entnehmen. Alle vorhandenen Elemente sind zusätzlich in Abb. 2.10 dargestellt. Die einzelnen Komponenten für die Bearbeitung der Aufgaben können entweder direkt auf die Pinheader aufgesteckt werden oder müssen anhand eines Schaltplans auf dem Steckbrett verschaltet werden. Dabei ist grundsätzlich immer darauf zu achten, dass keine Kurzschlüsse entstehen.

**Vor jeder Veränderung der Schaltung gilt es die Betriebsspannung und die USB Verbindung vom Controller zum PC zu trennen!**



für einen GPIO in der Software aktiviert werden oder als diskrete Bauteile auf dem Steckbrett über Verbindungskabel angebracht werden.

### 2.7.2 I<sup>2</sup>C-Display

Ein häufig eingesetztes Element für die Interaktion des Benutzers mit dem Mikrocontrollern ist ein Display. Auf dem Display werden üblicherweise Informationen bereitgestellt. Darüber hinaus kann eine Anzeige auch verwendet werden, um Debug-Informationen beim Testen der Software bereitzustellen.

Das im Praktikum verwendete Display mit 128x64 Pixeln besitzt einen eigenen Controller, der Befehle über die I<sup>2</sup>C-Schnittstelle erwartet, um die Pixel mit einer logischen Eins (Licht an) oder Null (Licht aus) zu versehen. Jedoch muss in diesem Fall nicht jedes einzelne Pixel angesteuert werden, da eine Bibliothek verwendet wird, welche Interface-Funktionen zur Nutzung des Displays bereit stellt. Weiterführende Informationen sind im [Datenblatt](#) zum Display SSD1306 zu finden. Angeschlossen wird das Display mit vier Leitungen nach der Tabelle 2.1. Im Kapitel 3.4 wird

**Tabelle 2.1:** Anschlussplan des SSD1306 an das Nucleo Board. Die Leitungen SDA und SCL sind entsprechend in CubeMX zu konfigurieren.

SSD1306	Nucleo
Vcc	3,3 V
GND	GND
SDA	SDA
SCK	SCL

auf die Dokumentation der erwähnten Bibliothek hingewiesen, in der die einzelnen Funktionen beschrieben sind. Zur Benutzung wird anfangs das Display initialisiert und geleert. Zur Darstellung von Strings sind Bibliotheksfunktionen vorhanden, mit denen übliche ASCII Zeichen in verschiedenen Schriftgrößen dargestellt werden können. Darüber hinaus können einzelne Pixel manuell gesetzt werden, um z.B. ein Logo oder benutzerdefinierte Zeichen darzustellen. Wichtig dabei ist, dass jeder Änderung erst mit dem Aufruf der Update-Funktion an das Display gesendet und dort dargestellt wird.

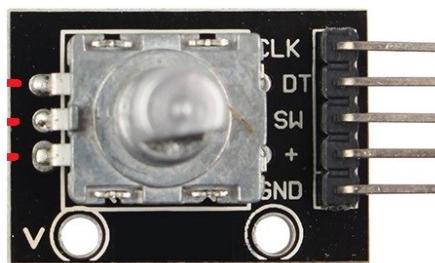
### 2.7.3 Temperatursensor

Eine der Aufgaben beinhaltet das Messen der Umgebungstemperatur und der relativen Luftfeuchte. Dazu wird der DHT11-Sensor verwendet, dessen Datenblatt

im Anhang A.2 zu finden ist. Der Sensor wird mit dem 1-Wire BUS angesprochen, dessen Timings und Abläufe im oben genannten Datenblatt beschrieben sind. Als Betriebsspannung für den DHT11 sind 3,3 V und ein Pull-Up-Widerstand von 4,7 kΩ zu verwenden.

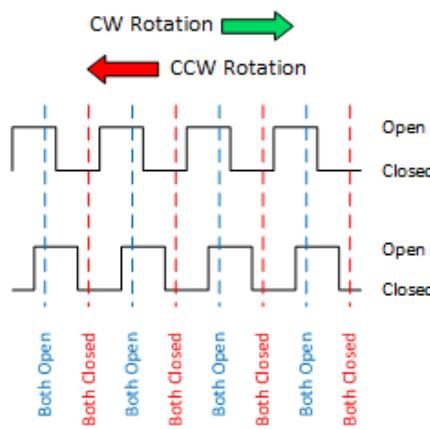
### 2.7.4 Encoder

Der Encoder KY-040 aus Abb. 2.11 sieht im ersten Moment wie ein Drehpotenzimeter aus. Mit Hilfe dieses Encoders kann sowohl die Drehrichtung als auch die



**Abbildung 2.11:** Encoder KY-040. [Bildquelle](#) vom 11.04.2019

Drehweite ermittelt werden. Zusätzlich ist noch ein einfacher Taster vorhanden. Der Encoder hat fünf Anschlüsse CLK, DT, SW, + und GND. An GND wird die Masse und an + die Betriebsspannung von 3,3 V angeschlossen. Der SW ist der Anschluss zum Taster und gibt bei Betätigung eine logische 0 aus. Abb. 2.12 zeigt den Signalverlauf für die beiden Drehrichtungen einmal im (grüner Pfeil) und einmal gegen den Uhrzeigersinn (roter Pfeil). Der obere Signalverlauf zeigt das Signal am CLK-Ausgang und der untere Verlauf das Signal am DT-Ausgang. Durch die



**Abbildung 2.12:** Encoder Ausgangssignale. Oben sind die Signale von CLK zu GND und unten die Signale von DT zu GND dargestellt. [Bildquelle](#) vom 11.04.2019

Auswertung beider Signale und das Zählen der Takte können Drehrichtung und Drehweite gemessen werden. Viele Encoder neigen dazu, bei schnellen Drehänderungen zu **prellen**. Abhilfe dazu können kleine Kondensatoren (10-100 nF) schaffen, die zwischen Masse und dem jeweiligen Anschluss (**CLK**, **DT**, und **SW**) angeschlossen werden.

### 2.7.5 Lüfter

Der Versuchsaufbau beinhaltet einen sogenannten PWM-Lüfter, der über vier Anschlüsse (Tabelle 2.2) verfügt. Die Frequenz des vom Lüfter generierten Tacho-

**Tabelle 2.2:** Lüfter-Anschlüsse

	Farbe	Funktion
Pin 1	schwarz	GND
Pin 2	gelb	12 V
Pin 3	grün	Tacho-Signal
Pin 4	blau	PWM

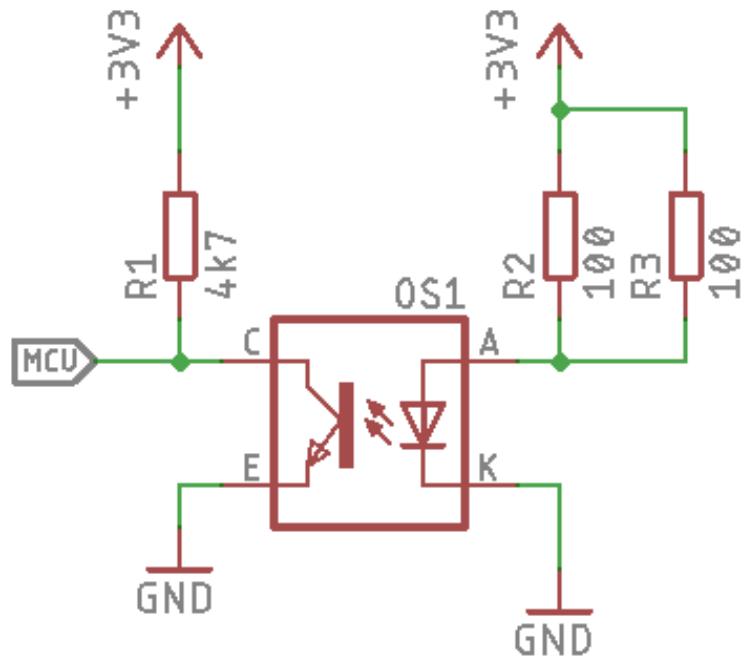
Signals entspricht der doppelten Drehfrequenz des Lüfters. Am PWM-Eingang wird ein PWM-Signal mit einer Frequenz von ca. 20 kHz erwartet, mit welchem die Drehzahl des Lüfters eingestellt werden kann.

### 2.7.6 IR-Detektor

Um die Drehfrequenz des Lüfters optisch zu messen, wird ein Infrarotlicht(IR)-Detektor TCRT5000 verwendet, dessen Datenblatt in Anhang A.1 zu findet ist. Dieses Bauteil beinhaltet eine Infrarotdiode sowie einen Phototransistor, der empfindlich auf Infrarotstrahlung reagiert. Der IR-Detektor ist nach dem Schaltplan in Abb. 2.13 anzuschließen. Der Widerstand R1 hat einen Wert von  $47\text{ k}\Omega$ , die Widerstände R2 sowie R3 sind  $100\text{ }\Omega$  groß.

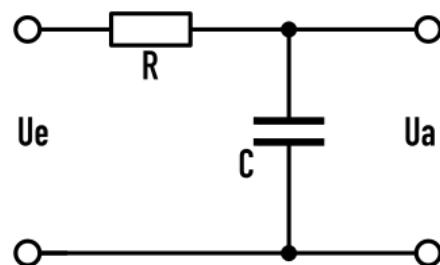
### 2.7.7 RC-Tiefpassfilter

Ein Tiefpass ist ein Filter, das die Signale unterhalb einer Grenzfrequenz  $f_g$  annähernd ungeschwächt passieren lässt, während die Frequenzanteile oberhalb der Grenzfrequenz gedämpft werden. Grundsätzlich können Tiefpassfilter mit analogen



**Abbildung 2.13:** Anschluss des TCRT5000. Links befindet sich der Phototransistor an dessen Kollektor (C) der Controller angeschlossen wird und rechts die Photodiode, die über einen Vorwiderstand an die Versorgungsspannung angeschlossen wird.

elektronischen Bauteilen oder rein digital in der Software implementiert werden. Zu Aufbau, Funktionsweise, Dimensionierung und Implementierung von Filtern sei an dieser Stelle auf folgende Literatur [9, 10] verwiesen. Im Praktikum wird ein analoges RC-Tiefpassfilter, bestehend aus einem Widerstand und einem Kondensator verwendet. In Abb. 2.14 ist der Schaltplan eines RC-Tiefpassfilters dargestellt. Die Grenzfrequenz ist als die Frequenz definiert, bei welcher die Amplitude des



**Abbildung 2.14:** Die Eingangsspannung  $U_e$  wird von dem Tiefpass in ihrer Amplitude und Phase beeinflusst. Oberhalb der Grenzfrequenz weist die Ausgangsspannung  $U_a$  eine reduzierte Amplitude und eine nacheilende Phasenverschiebung auf.

Ausgangssignals um 3 dB (ca. 29,3 %) gegenüber der Amplitude des Eingangssignals

reduziert wird. Es gilt:

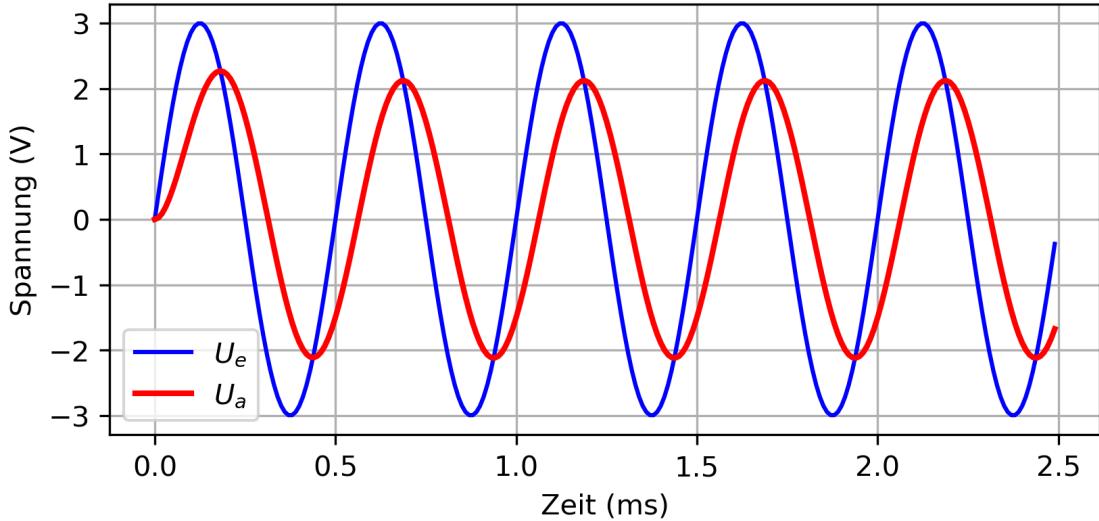
$$\hat{u}_a = \frac{\hat{u}_e}{\sqrt{2}}. \quad (2.4)$$

Weiterhin lässt sich die Grenzfrequenz des RC-Tiefpassfilters mit folgender Gleichung berechnen:

$$f_g = \frac{1}{2\pi RC}, \quad (2.5)$$

wobei der Wert des Widerstands  $R$  in Ohm und die Kapazität des Kondensators  $C$  in Farad vorgegeben werden.

Oberhalb der Grenzfrequenz werden die Signalanteile nicht nur gedämpft, sondern erfahren auch eine Phasenverschiebung. Das Ausgangssignal ist dem Eingangssignal nacheilend. In Abb. 2.15 ist ein Eingangssinussignal (blau) mit der Frequenz von 2 kHz und 3 V Amplitude dargestellt, welches mit einem RC-Tiefpassfilter mit der Grenzfrequenz  $f_g = 2$  kHz gefiltert wird.



**Abbildung 2.15:** Das Signal  $U_e$  (blau) mit 3 V Amplitude und der Frequenz von 2 kHz gefiltert mit einem Tiefpass mit der Grenzfrequenz  $f_g = 2$  kHz. Das Ausgangssignal  $U_a$  (rot) hat nach dem Einschwingen des Filters eine um ca. 29,3 % reduzierte Amplitude und ist um  $45^\circ$  phasenverschoben.

Das Ausgangssignal (rot) zeigt im Vergleich zum Eingangssignal eine gedämpfte Amplitude und eine Phasenverschiebung. Der Amplitudengang  $H(\omega)$ , also das Verhältnis der Ausgangsspannung  $U_a$  zur Eingangsspannung  $U_e$  in Abhängigkeit von der (Kreis)frequenz  $\omega$ , wird mit folgender Gleichung angegeben:

$$H(\omega) = \frac{\hat{u}_a}{\hat{u}_e} = \frac{1}{\sqrt{1 + (\omega RC)^2}}, \quad (2.6)$$

während der Phasengang  $\varphi(\omega)$ , also die Verzögerung des Ausgangssignals zum

Eingangssignal, mit

$$\varphi(\omega) = -\arctan(\omega RC) \quad (2.7)$$

berechnet wird. Die Kreisfrequenz  $\omega$  berechnet sich aus der Frequenz mit:

$$\omega = 2\pi f. \quad (2.8)$$

## 2.8 Vorbereitungsaufgaben

Zum ersten Praktikumstermin sind folgende Aufgaben schriftlich zu beantworten:

1. Erklären Sie kurz mit eigenen Worten die Funktion eines Pull-Down-Widerstands.
2. Gegeben ist ein ADC mit einer Auflösung von 10 Bit und einer Referenzspannung von 3 V. Wie groß ist die Spannungsauflösung des ADC?
3. Erklären Sie den Unterschied zwischen einem ADC und einem DAC?
4. Gegeben ist eine PWM bei 5 V und einem Tastgrad von 20 %. Wie groß ist die Spannung, wenn sie über eine PWM-Periode gemittelt wird?
5. Wofür wird ein DMA-Controller benötigt?
6. Wie groß muss die minimale Taktfrequenz eines Mikrocontrollers sein, wenn eine UART-Schnittstelle mit einer Baudrate von 115.200 Bd<sup>1</sup> implementiert werden soll? Wie schnell (Angabe in Datenbytes pro Sekunde) können Daten im Modus 8N1 übertragen werden?
7. Wie viel Strom kann ein GPIO-Pin des STM32F446 maximal abgeben? (Hinweis: Die Information kann dem Datenblatt entnommen werden.)
8. Mit welcher Frequenz kann der Timer 2 maximal takten? (Hinweis: Die Information kann dem Datenblatt entnommen werden.)

---

<sup>1</sup>Baud:  $1Bd = \frac{1}{s}$

## 2.9 Informationsübersicht

Einige der bereits verlinkten Informationen können Sie hier noch einmal gebündelt nachlesen.

- Nucleo Board User manual [17]
- großes Datenblatt [12]
- Timer cookbook [15]
- Signalerzeugung mit DAC [14]

Es wird an dieser Stelle nochmals dringend empfohlen, das eine oder andere Beispiel von ST auszuprobieren, um sich mit der Peripherie des Mikrocontrollers vertraut zu machen.

### 2.9.1 Video Tutorials

Als zusätzliche Hilfe sind im Weiteren einige Youtube Videos verlinkt, für deren Inhalt wir keinerlei Verantwortung übernehmen. Dem einen oder anderen hilft es ggf., Informationen zu bestimmten Themenfeldern noch einmal anders erklärt zu bekommen.

- offizielle STM32 Cube HAL basics Playlist
- ADC Modi
- Timer PWM
- Timer Input Capture (IC) Mode
- DAC und DMA

## 3 Software-Tools

### Inhalt des Kapitels

3.1	Beschreibung, Installation und Einrichtung . . . . .	26
3.1.1	STM32CubeIDE . . . . .	26
3.1.2	ST-Link Utility . . . . .	28
3.1.3	HTerm . . . . .	29
3.2	Pfade . . . . .	30
3.3	Erstes Projekt . . . . .	31
3.4	Bibliotheken . . . . .	36

Im Kapitel zuvor wurde die Hardware ausführlich beschrieben. Nun folgt eine Einführung in die IDE und weitere hilfreiche Software-Tools für Windows. Die Installation der IDE ist auch für Linux und MAC möglich, wird jedoch nicht Bestandteil des Praktikums sein. Vorweg sei erwähnt, dass die hier beschriebenen Anleitungen nicht sehr weit ins Detail gehen werden, da die jeweiligen Hersteller sehr umfassende Informationen in Form von PDFs und Videos bereitstellen. Verwendet werden die folgenden Softwarekomponenten:

- IDE: [STM32CubeIDE](#)
- [ST-Link Utility](#)
- UART Terminal: STM32CubeIDE-integriert oder [HTerm](#)

Einige Informationen gibt es direkt unter den oben aufgeführten Links auf der jeweiligen Softwareseite. Auf den Computern des Fachgebiets sind alle nötigen Tools und Treiber bereits installiert. Die nachfolgenden Anleitungen sind daher als Installationshilfe für Ihren privaten PC gedacht.

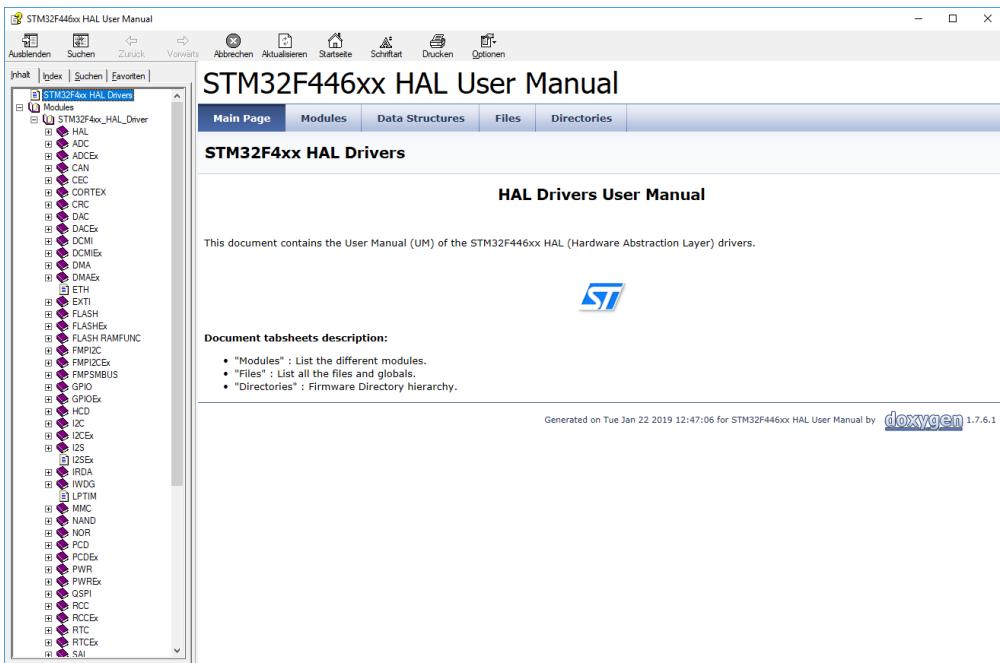
## 3.1 Beschreibung, Installation und Einrichtung

Die Programme STM32CubeIDE und ST-Link Utility werden von STMicroelectronics zur Verfügung gestellt. Zusätzlich wird HTerm im Praktikum verwendet, um die UART Schnittstelle am Computer zu nutzen. Es ist jedoch möglich, jede andere serielle Konsole zu verwenden.

### 3.1.1 STM32CubeIDE

STM32CubeIDE basiert auf der freien IDE **Eclipse**. Sie unterstützt den Anwender, verschiedene Komponenten aller Mikrocontroller der STM32-Serie des Herstellers zu initialisieren und konfigurieren. Zudem stellt sie ein Grundgerüst zur Entwicklung von eigenem Programmcode bereit. Die erzeugte Struktur nutzt die Hardware Abstraktion Layer (HAL)-Bibliothek (Dokumentation [18]), welche es dem Programmierer einfacher machen soll, mit komplexen Strukturen des Controllers zu interagieren. Ohne diese Bibliothek müsste die Initialisierung und Verwendung der Peripherie ausschließlich über die jeweiligen Register erfolgen. Die Konfiguration der Peripherie erfolgt über eine graphische Benutzeroberfläche. Die Verwendung der HAL-Bibliothek verfolgt das Ziel, den eigenen Programmcode weitgehend von der jeweiligen Hardware zu entkoppeln und auf diese Weise die

Portierbarkeit der Programme auf andere Controller-Modelle zu vereinfachen. Die automatische Code-Generierung zur Konfiguration der jeweiligen Peripherie ist zudem weniger fehleranfällig. Die HAL-Bibliothek führt jedoch zu einem zusätzlichen Programm-Overhead, wodurch es bei der Abarbeitung von Programmen mit zeitkritischen Abläufen auf dem Mikrocontroller zu Engpässen kommen kann. In solchen Fällen kann man die Register des Mikrocontrollers (ohne den Umweg über die HAL-Bibliothek) direkt ansprechen. Die Dokumentation der Bibliothek (siehe Abb. 3.1) ist als Windows-Hilfe-Datei im STM32CubeIDE Repository (siehe auch `...\\Repository\\STM32Cube_FW_F4_Vxxxxx\\Drivers\\STM32F4xx_HAL_Driver\\`) unter dem Namen `STM32F446xx_User_Manual` zu finden und bereits auf den Desktops der Praktikum-PC's verlinkt. Die integrierte Suchfunktion der Dokumentation erleichtert die Arbeit mit Bibliothek und Mikrocontroller.



**Abbildung 3.1:** Dokumentation der HAL-Bibliothek als Windowshilfe-Datei zum verwendeten Mikrocontroller

Das Tool STM32CubeIDE enthält unter anderem auch Informationen über das Nucleo-Board und kann dessen Hardwarekomponenten (Taster, LED, Takteingang und UART Schnittstelle) direkt einbinden. Die Installation des Programms wird, falls benötigt, in [22] ausführlich erklärt. Weiterführende Informationen zur IDE können der [Herstellerhomepage](#) entnommen werden; u.a. sind dort auch viele Video-Anleitungen zur Konfiguration und Verwendung des Programms verlinkt. Zusätzlich sollte die Kurzeinführung [21] zu STM32CubeIDE unbedingt gelesen werden.

Sehr **WICHTIG** ist bei der Verwendung von STM32CubeIDE, dass der eigene Benutzercode nur in den dazugehörigen, durch Kommentarblöcke gekennzeichneten Sektionen eingetragen werden darf:

```
1 /* USER CODE BEGIN 1 */
2
3 /* USER CODE END 1 */
```

**Listing 3.1:** Kommentarblock als Rahmen für Benutzercode

Alle Modifikationen außerhalb solcher Bereiche werden nach einer erneuten Generierung durch STM32CubeMX entfernt. Daher empfiehlt sich dringend die Verwendung von [git](#) als Versionsverwaltung. So kann falsch positionierter Programmcode wiederherzustellen, falls die Änderungen in git `commitet` wurden.

## Wichtigste Funktionen

In den eingangs erwähnten Tutorials werden die Funktionen der STM32CubeIDE erklärt. Abb. 3.2 zeigt die Werkzeugeleiste der IDE zum Erstellen und Debuggen



**Abbildung 3.2:** STM32CubeIDE: Werkzeugeleiste zum Erstellen und Debuggen eines Projekts.

eines Projekts. Dabei erstellt das Projekt, wechselt in den Debugmodus und flasht den Controller ohne zu debuggen. Wichtig bei diesen Funktionen ist, dass im **Project Explorer** das korrekte Projekt gewählt ist, da es sonst beim Klick auf die Schaltflächen zu einer Fehlermeldung kommen kann.

### 3.1.2 ST-Link Utility

Das Tool ST-Link Utility ist als Option im Installer der STM32CubeIDE integriert und sollte zur Installation ausgewählt werden. Es bietet die Möglichkeit, über den ST-Link-Programmer die Mikrocontroller der STM8- und STM32-Serie von STMicroelectronics zu konfigurieren und zu programmieren. Des weiteren kann mit diesem Tool die Firmware des ST-Link-Programmers aktualisiert werden. Das Tool ist insbesondere bei Kommunikationsfehlern mit dem Mikrocontroller hilfreich, da eine korrekte Installation und ein Erkennen des Programmers vom

PC-Betriebssystem sichergestellt werden kann. Zudem lässt sich auch die Verbindung zwischen Programmer und Mikrocontroller überprüfen. Wird die Verbindung erfolgreich hergestellt, so werden die Bezeichnung des Mikrocontrollers und der Inhalt des Flash-Speichers im Tool angezeigt. Kann das Tool keine Verbindung zum Mikrocontroller aufbauen, liegt möglicherweise ein Hardwareproblem bzw. Defekt vor. In solchen Fällen wird empfohlen, sich an die Praktikumsbetreuer zu wenden.

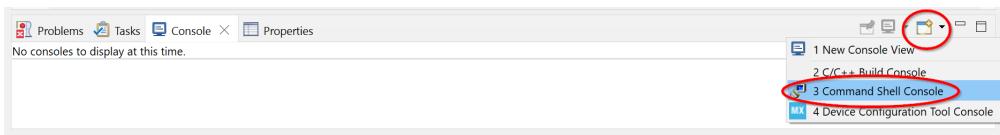
### 3.1.3 Terminal

Eine der Aufgaben wird sein, eine UART-Kommunikation zwischen Computer und Controller herzustellen, um diese in beide Richtungen zu nutzen.

Die Computer-internen serielle Schnittstellen (COM-Ports) werden von Windows mit Zahlen ab 1 benannt (COM1, COM2, ...). Sie werden in diesem Praktikum nicht verwendet.

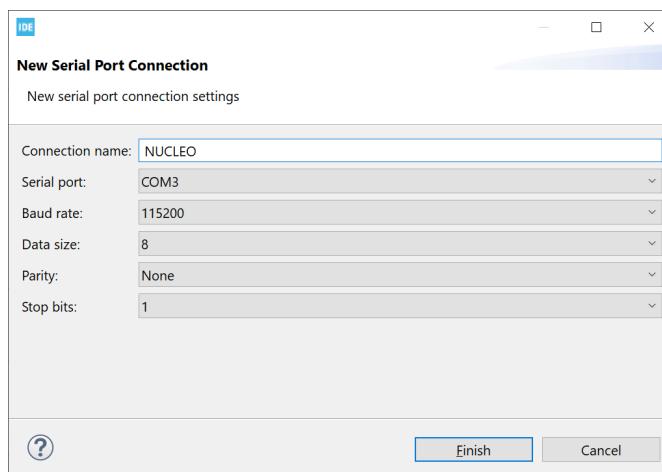
Die von den Windows-Treibern des Nucleo-Boards emulierte UART Schnittstelle hat eine entsprechend größere Zahl. Die genaue Bezeichnung des gesuchten Ports lässt sich im Windows **Geräte-Manager** unter **Anschlüsse** anhand des Eintrags **STMicroelectronics STLink Virtual COM Port (COMxx)** ermitteln. Neben der richtigen Wahl des COM-Ports im Terminal ist es für eine funktionierende Kommunikation ebenso wichtig, dass die Übertragungsparameter (Baudrate, Datenlänge, Stopbit, Parität und Handshake) von dem genutzten Terminal und dem Mikrocontroller übereinstimmen.

Zum Aufbau einer Verbindung kann als Terminal entweder das Tool **HTerm** oder eine entsprechend konfigurierte **STM32CubeIDE Console** verwendet werden. Leider wird HTerm nicht mehr supportet. Es bietet jedoch einen großen Funktionsumfang, ist übersichtlich und eignet sich daher nach wie vor für solche Anwendungen. Sind alle Parameter korrekt eingestellt, verbindet man **HTerm** mit einem Klick des **Connect**-Buttons mit dem Mikrocontroller. Anschließend können Daten empfangen und gesendet werden.



**Abbildung 3.3:** Aktivieren einer STM32CubeIDE Command Shell Console

Alternativ kann man in STM32CubeIDE eine Konsole für serielle Kommunikation konfigurieren. Dazu aktiviert man den **Console**-Reiter im unteren Bereich der IDE oder, falls noch nicht sichtbar, startet die Konsole unter **Window -> Show View -> Console**. Klickt man im Konsolen-Fenster auf das Dreieck ganz rechts, findet man im Menü die Option **3 Command Shell Console** (Abb. 3.3). Im folgenden Dialog wählt man als **Connection Type** den **Serial Port** und klickt auf **New**.



**Abbildung 3.4:** Einstellungen zur STM32CubeIDE Command Shell Console

In dem weiteren Dialog (Abb. 3.4) vergibt man einen Namen für die zu erstellende Verbindung und kann den COM-Port des Nucleo-Boards sowie die weiteren Übertragungsparameter auswählen.

## 3.2 Pfade

Die wichtigsten Dateipfade auf den Computern des Fachgebiets im Überblick:

**STMCube Repository:** D:\STM32Cube\Repository  
**Für alle Teilnehmer:** S:\  
**Benutzerverzeichnis:** Z:\

**Bitte speichern Sie keinerlei Daten lokal (C:, D:) auf den Fachgebiets-PCs!**

### 3.3 Erstes Projekt

Um den Umgang mit der IDE zu erlernen, wird ein erstes typisches Hello-World Projekt erstellt. Dazu soll die auf dem Nucleo-Board fest verbaute LED zum Blinken gebracht werden. Dieses Startprojekt wird Teil der Aufgabe 4.1.1 sein und sollte daher zur Übung unbedingt erstellt werden.

1. Starten Sie STM32CubeIDE.

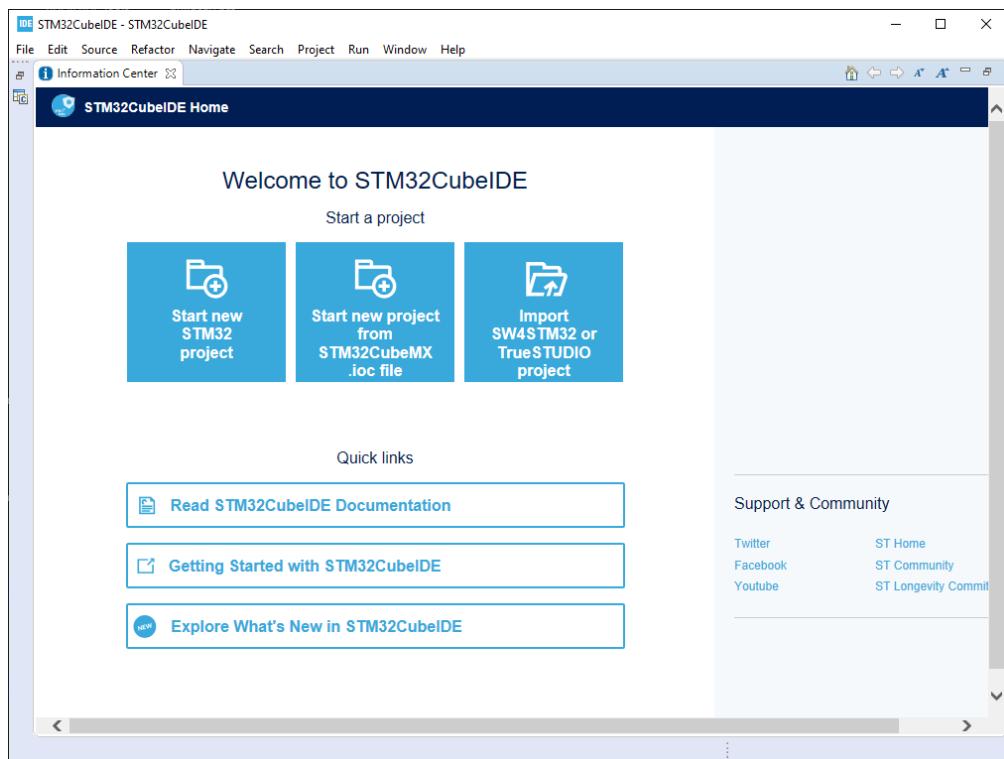


Abbildung 3.5: Startansicht von STM32CubeIDE (Version 1.2.0)

2. Generieren Sie eine Projekt-Grundstruktur. Klicken Sie dazu entweder in der Startansicht (Abb. 3.5) auf `Start new STM32 project` oder wählen im Menü `File - New - STM32 Project`.
3. Im nun erscheinenden Dialog **Target Selection** (Abb. 3.6.) wählen Sie den Reiter **Board Selector** und geben den Namen NUCLEO-F446RE des im Praktikum verwendeten Boards in die Suche ein.
4. Selektieren Sie das Board in der rechts erscheinenden **Boards List** und klicken Sie `Next >`.

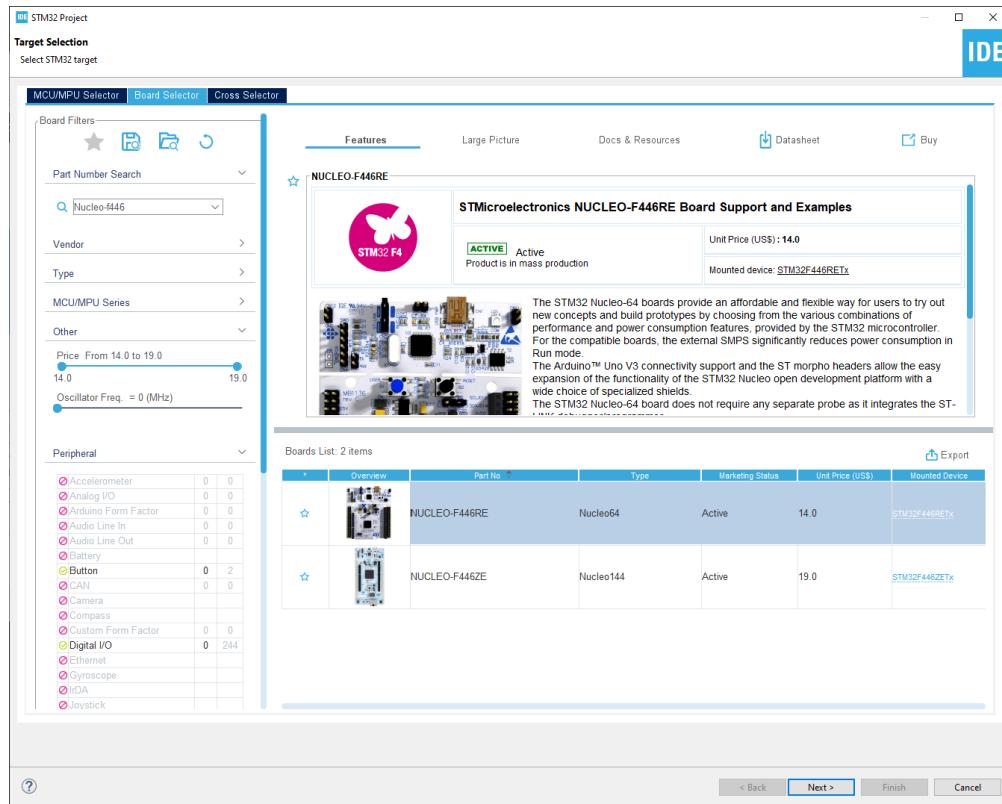


Abbildung 3.6: STM32CubeIDE Board Selector

5. Im Dialog **STM32 Project** (Abb. 3.7) vergeben Sie einen Projektnamen und klicken **Finish**.

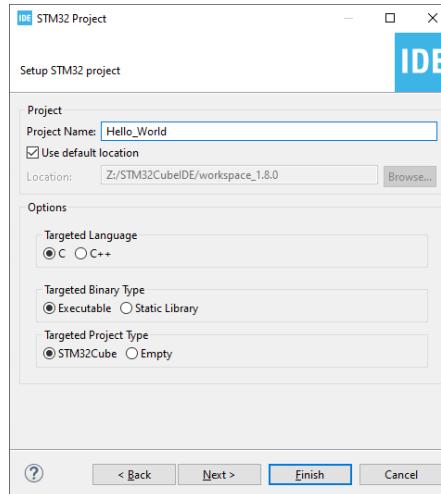


Abbildung 3.7: STM32 Project

6. Den Dialog **Board Project Options** (Abb. 3.8) beantworten Sie schließlich mit **yes**, so dass die Peripherie (LED, Takteingang, Taster und UART) der Hardware automatisch eingebunden wird.

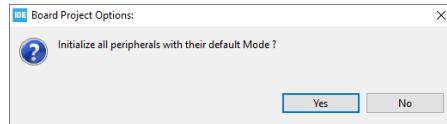


Abbildung 3.8: Board Project Options

Der letzte Schritt nimmt nach einer Erstinstallation einige Zeit in Anspruch, da Daten von der Hersteller-Website nachgeladen werden müssen. Das generierte Projekt sollte etwa die Ansicht aus Abb. 3.9 zeigen. Hier werden alle Initialisierungseinstellungen für den Controller vorgenommen. Dazu stehen verschiedene Reiter zur Verfügung.

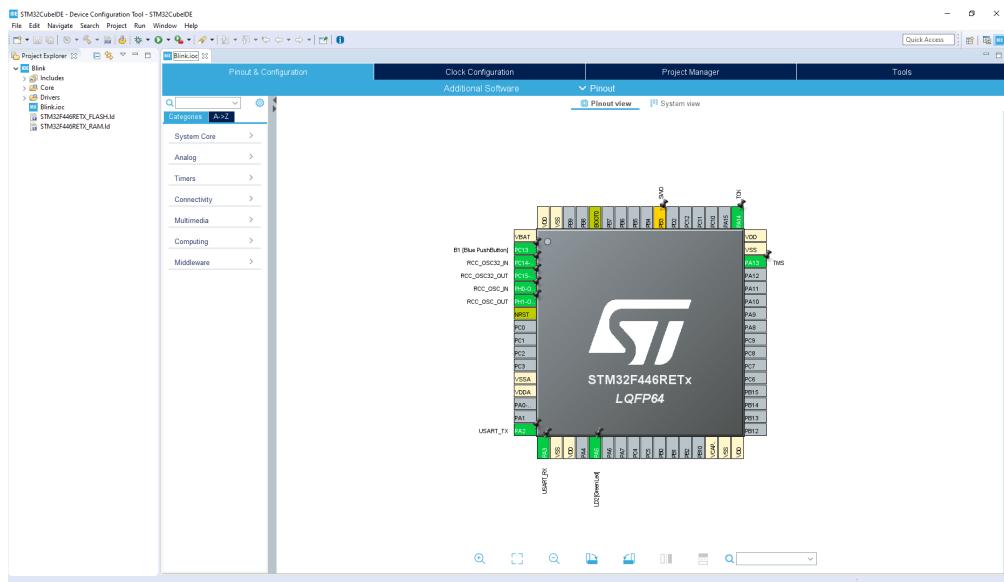
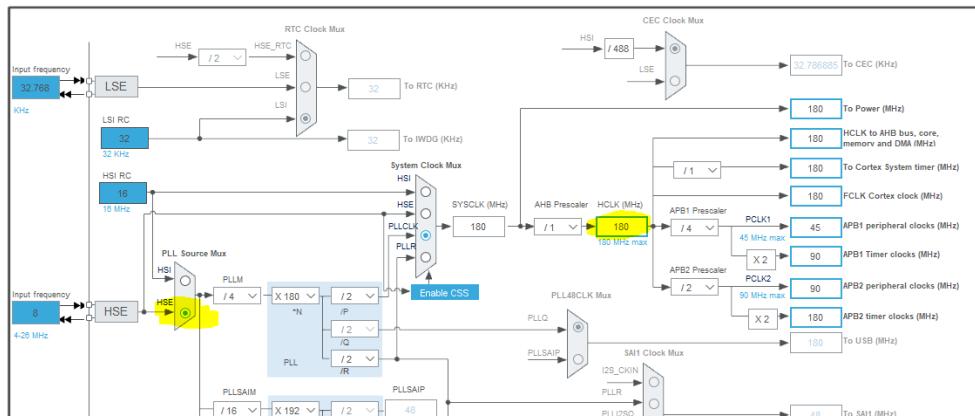


Abbildung 3.9: Konfigurationsübersicht des Mikrocontrollers mit vier Reitern

Im Reiter **Pinout and Configuration** kann man die benötigte Peripherie (linke Seite über ausklappbare Menüs) aktivieren und die Ein- und Ausgänge konfigurieren. Für das aktuelle Projekt sind alle benötigten Anschlüsse bereits vorkonfiguriert. In diesem Projekt wird der Ausgang PA5 mit dem Namen LD2 für die grüne LED verwendet.

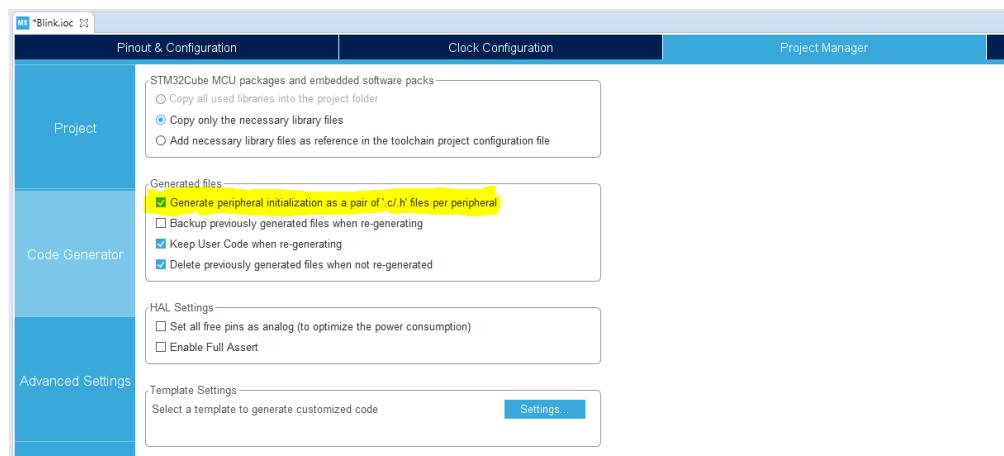
Der **Clock Configuration** Reiter beinhaltet die Takteinstellungen des gesamten Controllers. Für alle Projekte dieses Praktikums wird das HSE Register als Eingang mit einer Frequenz von 8 MHz genutzt. Einstellungen können durch manuelles Verstellen der einzelnen Teiler und Multiplikatoren oder über die Eingabe von Frequenzen vorgenommen werden. Konfigurationen, die aufgrund von Hardwarebeschränkungen nicht möglich sind, werden dabei rot markiert.

7. Stellen Sie auf dem **Clock Configuration** Reiter die PLL Source auf HSE und ändern Sie die weiteren Einstellungen so, dass an der Stelle HCLK eine Frequenz von 180 MHz eingestellt ist (Abb. 3.10).



**Abbildung 3.10:** Takteinstellungen mit HSE als Eingang und max. Frequenz

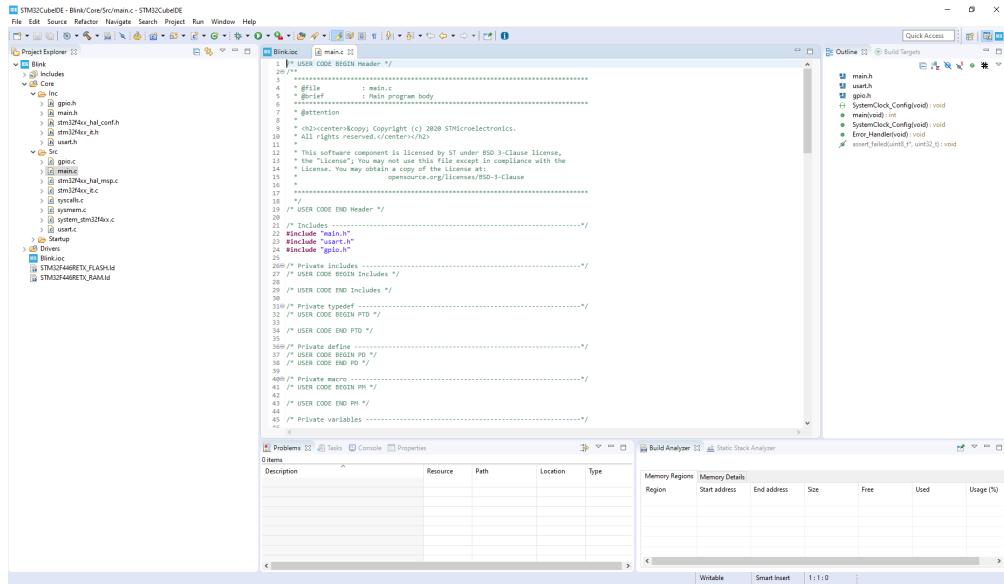
8. Wählen Sie den Reiter **Project Manager** und aktivieren im Feld **Code Generator** die Option **Generate peripheral initialization as a pair of '.c/.h' files ...** (siehe Abb. 3.11). Der vorgenerierte Quellcode wird damit übersichtlicher.



**Abbildung 3.11:** Project Manager Tab, Code Generator Einstellungen

9. Speichern Sie das Projekt (Disketten-Symbol oder **Strg+S**). Falls anschließend abgefragt wird, ob der Code generiert werden soll, beantworten Sie dies mit **Yes**. Oder stoßen Sie diese Operation an im Menü **Project - Generate Code**.

Alle anderen Einstellungen sollten unverändert bleiben. Im **Project Explorer** (in Abb. 3.12 links) wird das Projekt angezeigt, wobei sich im **Core/Src**-Ordner die Quellcode-Dateien und im **Core/Inc**-Ordner die Header-Dateien befinden. Falls die Quellcode-Datei **main.c** noch nicht automatisch im Editor angezeigt wird, sollte dieses jetzt durch Doppelklick auf die Datei erreicht werden. In der Datei **main.c**



**Abbildung 3.12:** Programmieroberfläche mit **Project Explorer** links

sind einige Initialisierungsfunktionen, sowie die **main()**-Funktion enthalten. Eigener Programmiercode sollte ausschließlich zwischen den mit **USER CODE BEGIN** und **USER CODE END** markierten Zeilen eingefügt werden. Programmzeilen an anderen Stellen werden bei einer Neugenerierung des Codes, z.B. bei Änderungen zu den Ein- und Ausgängen, gelöscht.

- Zur Realisierung der gestellten Aufgabe, das Blinken der LED, fügen Sie in der while-Schleife der main-Funktion die Funktionen **HAL\_GPIO\_WritePin** zum Schalten des Zustands der LED und **HAL\_Delay** als Pause zwischen den Zuständen ein (siehe Listing 3.2).

```

1 /* Infinite loop */
2 /* USER CODE BEGIN WHILE */
3 while (1)
4 {
5     HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
6     HAL_Delay(100);
7     HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);
8     HAL_Delay(100);
9 /* USER CODE END WHILE */

```

```

10
11  /* USER CODE BEGIN 3 */
12 }
13 /* USER CODE END 3 */

```

**Listing 3.2:** LED-Blinken als blockierende Implementierung in der Hauptschleife. Die Funktion `HAL_Delay` wartet die angegebene Zahl in ms und `HAL_GPIO_WritePin` setzt den Pin der LED auf logisch 1 bzw. auf logisch 0.

11. Verbinden Sie nun das NUCLEO-F446RE Entwickler-Board mit einem USB Anschluss des PC's. Das neue Gerät wird erkannt.
12. Zum Erstellen des für den Controller ausführbaren Codes klicken Sie auf  in der Toolbar der Programmieroberfläche (Abb. 3.12). In der `Console` (Aufruf `Window - Show View - Console`) sollte `Build Finished. 0 errors, 0 warnings.` erscheinen.
13. Mit einem Klick auf  wird der Controller geflashed und das Programm startet. Die LED blinkt mit einer Frequenz von ca. 5 Hz.

## 3.4 Bibliotheken

Für einige Aufgaben werden externe Bibliotheken benötigt. Diese Bibliotheken mit zugehörigen Dokumentationen stehen auf dem Fachgebiets-eigenen [git-Server](#) (auch über Internet erreichbar) zum Download bereit. Alle erforderlichen Dateien sind in den jeweiligen Repositories enthalten oder werden dort verlinkt. Aktuell werden zwei Bibliotheken für die weiteren Aufgabenstellungen benötigt:

[DHT11 Bibliothek](#) und [Dokumentation](#)

[Display Bibliothek](#) und [Dokumentation](#)

Die benötigten Dateien werden bei Bedarf am einfachsten in die beiden Projekt-Ordner `Core/Inc` und `Core/Src` kopiert.<sup>1</sup>

---

<sup>1</sup>Es ist auch möglich die oben angegebenen Bibliotheken als `git submodule` einzubinden, dies benötigt aber etwas Vorwissen beim Umgang mit `git` und `Eclipse` und wird daher nur für erfahrene Anwender empfohlen.

## 4 Aufgaben

### Inhalt des Kapitels

4.1	Block 1: Ein- und Ausgaben . . . . .	38
4.1.1	Blinkende LED . . . . .	38
4.1.2	UART . . . . .	40
4.1.3	Display . . . . .	41
4.2	Block 2: Sensoren . . . . .	42
4.2.1	Temperaturmessung mit DHT11 . . . . .	42
4.2.2	Encoder . . . . .	43
4.2.3	ADC und DAC . . . . .	45
4.3	Block 3: PWM-Ansteuerung / Drehzahlmessung . . . . .	46
4.3.1	Lüfter-Ansteuerung . . . . .	46
4.3.2	Drehzahlmessung mit Tacho-Signal . . . . .	47
4.4	Block 4 - Abschlussaufgabe: Signalerzeugung mittels PWM und DAC sowie Signalaufzeichnung mit einem ADC. . . . .	48

In den Kapiteln zuvor wurden die Grundlagen geschaffen, welche Sie für die folgenden Aufgaben benötigen. Die Aufgaben werden zu Anfang eine relativ starre Struktur vorgeben, damit Sie einen einfachen Einstieg in das Thema erhalten. Zum Ende hin werden sie etwas allgemeiner gehalten, um Ihnen auch eigene Kreativität zu ermöglichen. Der Programmcode sollte dabei stets gut lesbar, strukturiert und dokumentiert sein. Der Code soll so geschrieben werden, dass er von jemandem nachvollzogen werden kann, der sich zwar mit der Programmierung im Allgemeinen auskennt, jedoch den Programmcode nicht selbst geschrieben hat. Achten Sie darauf, die wichtigen Schritte sowohl durch Einrückungen als auch durch Kommentare zu trennen und zu beschreiben. Es wird empfohlen, die Kommentare auf Englisch zu verfassen, da sich dies bei den ohnehin englischen Code-Ausdrücken gängiger ließt. Zudem empfehlen wir Ihnen jede Aufgabe in einem einzelnen Projekt umzusetzen, um eine Fehleranalyse zu vereinfachen. Es sei nochmals darauf hingewiesen, dass die Verwendung von `git` als Versionsverwaltung sehr empfehlenswert ist.

**Bitte nutzen Sie zur Datenspeicherung ausschließlich das Netzwerk!**

**Home-Verzeichnis als Netzlaufwerk:** Z:\

**Project Location für SMT32CubeIDE:** Z:\CubeIDE

So finden Sie unabhängig vom genutzten Arbeitsplatz immer die gleiche Projektumgebung vor. Der Ihnen zugeteilte User-Account steht über das gesamte Praktikum nur Ihnen bzw. Ihrer Gruppe zur Verfügung.

## 4.1 Block 1: Ein- und Ausgaben

Der erste Aufgabenblock beinhaltet die grundlegenden Funktionen der Kommunikation zwischen Controller und Computer/Benutzer.

### 4.1.1 Blinkende LED

Zum Einstieg soll in dieser Aufgabe das Hello-World Projekt aus Kapitel 3.3 implementiert und modifiziert werden. Ihre Aufgabe ist es, die LED mit einer Frequenz von 10 Hz blinken zu lassen. Dazu können Sie das Beispiel aus dem zuvor erwähnten Kapitel nutzen und müssen nur die Wartezeit anpassen.

Mit den darin enthalten wenigen Quellcode-Zeilen ist der Controller bereits aufgrund des endlosen Wartens in seiner main-Funktion voll ausgelastet und jeder Interrupt würde das Blinken unterbrechen oder stören. Daher sollen Sie als nächstes das Blinken über einen Timer realisieren, dessen Interrupt für einen Funktionsaufruf (außerhalb der while-Schleife der main-Funktion) genutzt werden kann, um die LED blinken zu lassen.

In der Konfigurationsübersicht (Abb. 3.9) können Sie einen geeigneten Timer aktivieren. Wählen Sie dazu **Internal Clock** als **Clock Source** des ausgewählten Timers. Je nachdem ob ein Timer mit APB1 oder APB2 verbunden ist, taktet dieser mit 90 MHz oder 180 MHz (siehe [12], 3.21 Timers and watchdogs). Der gewünschten Takt kann unter **Configuration - Parameter Settings** des Timers auf verschiedene Arten eingestellt werden. Als erstes kann der **Prescaler** (Teiler) bis zu einem Wert von  $2^{16} - 1$  gesetzt werden, um den Takt des Zählers zu reduzieren. Dabei entspricht der angegebene Wert dem Teiler minus Eins, so dass z.B. für einen Teiler von 180 ein Wert von 179 als **Prescaler** anzugeben ist. Mit der so eingestellten Taktfrequenz startet der Zähler bei 0 und zählt bis zum Wert des **AutoReload**-Registers aufwärts. Wird der Wert erreicht, löst dies den sogenannter **Timer-Overflow** Hardware-Event aus, zu welchem ein Interrupt unter **NVIC Settings** aktiviert werden kann. Aus der Kombination von **Prescaler** und **AutoReload**-Register kann die Interrupt-Periode so eingestellt werden, dass die geforderten 10 Hz als Blinkfrequenz der LED resultieren.

Die Konfiguration eines Timers bedingt eine Neu-Generierung des Quellcodes. Dabei wird in der Datei `stm32f4xx_it.c` eine neue Funktion zu dem aktivierten Timer angelegt, die vom Interrupt des Timers aufgerufen wird, z.B. für Timer2 `TIM2_IRQHandler(void)`. In dieser Funktion kann nun z.B. die LED gesteuert werden ohne dabei den Controller für so eine einfache Aufgabe in seiner while-Schleife der main-Funktion voll auszulasten. Wird übrigens die Funktion `HAL_GPIO_TogglePin` anstatt `HAL_GPIO_WritePin` genutzt, ist keine Fallunterscheidung des Portzustands nötig.

Schließlich muss der Timer in Verbindung mit dem Auslösen des Interrupts noch gestartet werden. Dazu positioniert man als Beispiel für Timer2 den Aufruf `HAL_TIM_Base_Start_IT(&htim2)` vor die while-Schleife der main-Funktion. (Der Timer-Handle `htim2` wurde bei der Code-Generierung in der Datei `tim.c` angelegt. Zum Starten des Timers in Verbindung mit DMA wird ein anderer Funktionsaufruf benötigt, siehe dazu auch die Dokumentation der HAL-Bibliothek.)

### 4.1.2 UART

Nachdem Sie die LED erfolgreich zum Blinken gebracht haben, ist der nächste Schritt Werte über die UART-Schnittstelle auszugeben und auch einzulesen. Ihre Aufgabe ist es, den bereits aktivierten UART2 so zu benutzen, dass Sie beim Starten des Programms einen String mit dem Projektnamen ausgeben und im Anschluss auf eine Eingabe warten. Diese Eingabe soll dazu genutzt werden, die Frequenz der blinkenden LED zu variieren.

Zur Ausgabe von Text über den UART kann die Funktion `HAL_UART_Transmit` verwendet werden. Die Übergabeparameter entnehmen Sie bitte der Hilfe zur HAL-Bibliothek. Das Timeout kann z.B. mit dem Makro `HAL_MAX_DELAY` auf einen maximalen Wert gesetzt werden. Als Textabschluss zur Zeilenende-Signatur sind im Windows-Betriebssystem die beiden Zeichen `\r\n` zu empfehlen.

**Hinweis:** Hilfe zu den HAL-Funktionsaufrufen erhält man auch, wenn sich der Cursor in der Entwicklungsumgebung auf einer UART Funktion befindet und man die Taste F3 drückt. Man gelangt dann in die Datei `stm32f4xx_hal_uart.c` zur Implementierung der jeweiligen Funktion. In dieser Datei sind alle nutzbaren UART-Funktionen enthalten und kommentiert.

Zum Empfangen von Daten, die per **HTerm** oder einem anderen Terminal-Programm gesendet werden, gibt es verschiedene Möglichkeiten. Sie können sowohl die Funktion `HAL_UART_Receive` als auch die Funktion `HAL_UART_Receive_IT` verwenden. Die erste Funktion blockiert den gesamten Ablauf des Controllers bis die erwartete Anzahl von Zeichen empfangen wurde oder die angegebene Zeit abgelaufen ist. Die zweite Funktion füllt den übergebenen Buffer bis zu der angegebenen Länge und löst dann den UART Interrupt zur Übergabe der Daten aus, ohne zu blockieren. Ähnlich wie bei der Nutzung des Timer-Interrupts in der vorherigen Aufgabe kann so auch beim Empfangen von Daten eine Blockierung der Main-Loop verhindert werden. Mit dem Auslösen des UART Interrupts wird die Callback-Funktion `HAL_UART_RxCpltCallback` aufgerufen.

```
1 /* USER CODE BEGIN 4 */
2 void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
3 {
4     // insert user code here to use UART input
5 }
```

```
6 /* USER CODE END 4 */
```

**Listing 4.1:** UART-receiving-complete-callback Function (`HAL_UART_RxCpltCallback`)  
Implementierung in `main.c`; wird bei aktiviertem UART2-Interrupt ausgeführt.

Diese Funktion ist in die `main.c` einzufügen (siehe Listing 4.1) und kann zur Verarbeitung der empfangenen Daten genutzt werden. **Denken Sie daran, im Device Configuration Tool den zugehörigen Interrupt für UART2 zu aktivieren!**

Allgemein gilt: Je weniger Programmcode in der Main-Loop enthalten ist, desto effizienter werden die Aufgaben vom Mikrocontroller umgesetzt.

**Hinweis:** Damit der für den Empfang genutzte Buffer auch in der Callback-Funktion verfügbar ist, muss er global definiert werden.

**Hinweis:** Senden Sie eine fest definierte Anzahl von Ziffern und füllen bei kleinen Zahlen mit Nullen am Anfang auf.

Die vom Computer gesendeten Zahlen werden üblicherweise als ASCII-Zeichen übertragen. Überlegen Sie sich daher eine praktikable Lösung wie Sie die ASCII Zeichen in eine Zahl umwandeln können.

#### 4.1.3 Display

Nachdem eine Ein- und Ausgabemöglichkeit per UART implementiert wurde, soll eine zusätzliche Ausgabe hinzukommen. Ein Display wird in den weiteren Aufgaben genutzt, um berechnete Werte anzuzeigen oder Debug-Ausgaben darzustellen.

Zunächst soll die zuvor übertragene Signalfrequenz für die blinkende LED auf dem Display dargestellt werden. Dazu schließen Sie das Display an (siehe auch Kap. 2.7.2) und binden die Bibliothek SSD1306 (Kap. 3.4) ein. Grundlegende Funktionen der Bibliothek sind in der [Dokumentation](#) erklärt. Auf dem Display soll idealerweise eine kurze Bezeichnung und Einheit des angezeigten Wertes dargestellt werden. Darüber hinaus ist darauf zu achten, dass die Daten auf dem Display aktualisiert werden und sich nur der Wert ändert, nicht aber die Position der Bezeichnung oder Einheit.

## Aufgabenblock

Am Ende dieser Aufgabe soll ein Programm vorhanden sein, das die folgenden Anforderungen erfüllt:

- Eine LED blinkt in einer definierten Frequenz.
- Die Frequenz kann über die UART-Schnittstelle vorgegeben, verändert und auch dargestellt werden.
- Die Frequenz wird im Display ausgegeben.

Achten Sie bitte auf eine saubere Programmstruktur und fügen Sie ausreichend viele Kommentare ein, damit für den Betreuer beim Testat der Programmcode gut lesbar und nachvollziehbar ist. Außerdem sollen Sie selbstständig die wichtigsten Programm-punkte ausführlich erklären und begründen können.

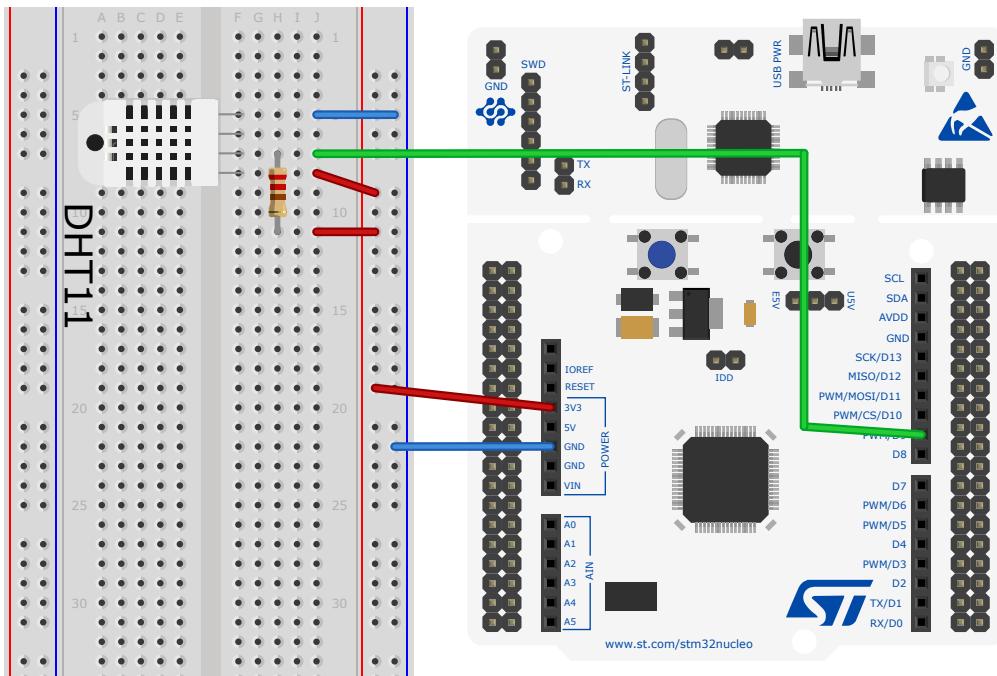
## 4.2 Block 2: Sensoren

Wenn Sie den Aufgabenblock 1 erfolgreich abgeschlossen und das dazugehörige Testat abgelegt haben, können Sie mit den drei folgenden Aufgaben fortfahren.

### 4.2.1 Temperaturmessung mit DHT11

In dieser Aufgabe soll zunächst das Datenblatt in Anhang A.2 des DHT11-Sensors aufmerksam gelesen und nachvollzogen werden. Der Sensor ist kostengünstig und daher im Hobbybereich ziemlich beliebt zur Messung von Temperatur und relativer Luftfeuchtigkeit. Den relativ großen Messfehler des Sensors können Sie für die folgende Anwendung ignorieren, da es in der Aufgabe nicht auf eine exakte Messung ankommt. Sie sollen vielmehr lernen, wie plausible Daten aus dem Sensor ausgelesen werden können.

Der DHT11 wird über einen 1-Wire Bus (vgl. Kap. 2.6.3) angesteuert. Zur elektrischen Verbindung nutzen Sie bitte den Anschlussplan aus Abbildung 4.1. Den Datenpin (grün in Abb. 4.1) können Sie auf dem Nucleo-Board frei wählen. Im oben genannten Datenblatt sind alle notwendigen Informationen enthalten, wie die Daten aus dem Sensor ausgelesen werden können. Es ist genau beschrieben, welche Pegel in welcher Reihenfolge an dem 1-Wire Bus anliegen müssen und was diese bedeuten.



**Abbildung 4.1:** Anschlussübersicht des DHT11 an das Nucleo-Board. Für den richtigen Widerstandswert schauen Sie bitte in Kapitel 3 des Datenblatts A.2 nach. Die Pinbelegung des DHT11 ist in der obigen Ansicht von oben nach unten absteigend von 4 bis 1.

Auch hier existiert bereits eine [Bibliothek](#) mit den grundlegenden Funktionen, die mit einem Beispiel in der dazugehörigen [Dokumentation](#) beschrieben sind. Es sollen folgende Aufgaben erledigt werden:

- Lesen und **Verstehen** des Datenblattes in A.2.
- Anschluss des DTH11 Sensors.
- Nutzung der dazugehörigen [Bibliothek](#).
- Zyklisches Auslesen der Temperatur und der rel. Luftfeuchtigkeit in definierten Zeitabständen. (**Hinweis:** Alle 100 ms sind nicht sinnvoll.)
- Ausgabe der Messdaten per UART.
- Ausgabe der Messdaten auf dem Display.

### 4.2.2 Encoder

Die grundlegende Funktionsweise eines Encoders wurde im Kapitel 2.7.4 beschrieben. In dieser Aufgabe soll nun der vorliegende Encoder in Betrieb genommen

werden. Die Encoder werden hauptsächlich für die Benutzereingaben verwendet. Beispielsweise werden damit Zahlenparameter verändert oder sie werden zum Navigieren durch die Benutzer-Menüs eingesetzt. In den meisten Fällen erfordert die Interaktion des Benutzers mit der Peripherie keinen Echtzeit-Feedback, der im einstelligen Millisekunden oder sogar Mikrosekundenbereich liegt. Feedback-Zeiten von bis zu 200 ms werden vom Menschen als "sofort" empfunden. Die Änderung des Encoder-Zustandes sollte daher von der Hardware zwar möglichst schnell detektiert werden, die Reaktion auf den veränderten Zustand kann jedoch mit etwas "Verzögerung" erfolgen. Bei dieser Aufgabe soll der Encoder dazu genutzt werden, die Puls-Frequenz der LED aus der Aufgabe 4.1.1 zu verändern und die aktuelle Frequenz über UART oder auf dem Display anzuzeigen. Dabei soll der vorgegebene Wert erst dann übernommen werden, wenn der im Encoder eingebaute Taster betätigt wird. Überlegen Sie vorher den sinnvollen Bereich für die LED-Blinkfrequenz, damit diese Frequenz vom menschlichen Auge noch wahrgenommen werden kann.

Nachfolgend sind die Anforderungen aufgelistet, die in der Mikrocontrollersoftware umgesetzt werden sollen.

- Encoder inkl. Taster an den Mikrocontroller anschließen. (**Hinweis:** Achten Sie dabei auf die richtige Anschlussbelegung, denken Sie dabei an die Entprell-Kondensatoren.)
- Encoder Modus im Mikrocontroller an den dafür zur Verfügung stehenden Pins aktivieren und entsprechend parametrieren. (**Hinweis:** Encoder-Funktionalität wird von Timern bereitgestellt.)
- Taster-Pin als digitalen Eingang konfigurieren. (**Hinweis:** die Betätigung des Tasters soll sofort detektiert werden, dies lässt sich am einfachsten mit einem externen-Interrupt realisieren)
- In regelmäßigen Abständen den Encoder-Wert auslesen und ggf. eine globale Variable aktualisieren. (**Hinweis:** Mit einem weiteren Timer kann eine regelmäßige Abfrage realisiert werden.)
- Encoder-Zustand (bei Änderung) über die Schnittstelle Ihrer Wahl ausgeben/anzeigen.
- Bei Betätigung des Tasters den aktuellen Wert übernehmen und die Pulsfrequenz der LED verändern. (**Hinweis:** Beachten Sie dabei den festgelegten Blick-Frequenzbereich und limitieren Sie ggf. die vorgegebenen Werte.)

### 4.2.3 ADC und DAC

In dieser Aufgabe lernen Sie die grundlegende Funktionsweise der in den Mikrocontroller integrierten ADC- und DAC-Bausteine kennen. Es soll mit Hilfe eines Potentiometers die Spannung zwischen 0 V und VCC verstellt werden, diese soll mit dem Mikrocontroller über den ADC ausgelesen, ggf. korrigiert und als Spannungswert über die Schnittstelle Ihrer Wahl ausgegeben/angezeigt werden. Weiterhin soll die gemessene (digitalisierte) Spannung über den DAC-Ausgang des Mikrocontrollers ausgegeben werden. Dabei ist darauf zu achten, dass die Abweichung zwischen der gemessenen (ADC) und der ausgegebenen Spannung (DAC) möglichst klein bleibt. Bei dieser Aufgabe wird keinen großen Wert auf die Schnelligkeit (Echtzeitfähigkeit) gelegt, daher kann die Peripherie (ADC und DAC) über langsamere Funktionsaufrufe (typischerweise HAL-Funktionen ohne Interrupt- oder DMA-Funktionalität) erfolgen.

Im Einzelnen sollen folgende Anforderungen/Aufgaben umgesetzt werden:

- Potentiometer an den Mikrocontroller anschließen. (Achten Sie dabei auf die richtige Anschlussbelegung und **vermeiden** Sie auf jeden Fall **Kurzschlüsse**.)
- ADC-Peripherie und einen dazu passenden Pin aktivieren.
- Die Spannung am ADC-Pin bei verschiedenen Potentiometer-Stellungen auslesen (mehrere Werte über den gesamten Messbereich von 0 V bis VCC) und in den Spannungswert umrechnen. (siehe Kapitel 2.2)
- Bei zu großen Abweichungen von dem tatsächlichen Spannungswert (mit Multimeter kontrollieren) eine Korrekturstrategie überlegen und ggf. umsetzen. (siehe Application note [AN2834](#) des Herstellers)
- In regelmäßigen Zeitabständen den ADC-Pin auslesen, den gemessen Wert ggf. korrigieren und über die Schnittstelle ihrer Wahl ausgeben.
- DAC-Peripherie und den dazugehörigen Pin aktivieren.
- Grenzen des DAC-Spannungsbereichs durch die Vorgabe verschiedener Werte bestimmen und ggf. einen Hinweis (Warnung) über die Kommunikationschnittstelle ausgeben, wenn ein Wert oberhalb oder unterhalb dieses Bereichs vorgegeben wird. (**Hinweis:** Es kann vorkommen, dass an den Rändern des

DAC-Wertebereichs (z.B.  $\leq 50$  oder  $\geq 4000$ ) trotz der Vorgabe eines anderen Wertes keine Änderung der Spannung erfolgt, weil die Hardware in die Sättigung kommt.)

- Bei zu großen Abweichungen von den vorgegebenen Werten eine Korrekturstrategie überlegen und ggf. umsetzen. (siehe Hinweise für ADC)
- Die mit dem ADC gemessene Spannung über den DAC ausgeben/stellen.  
**(Hinweis:** Achten Sie darauf, dass die Abweichung zwischen der gemessenen und ausgegebenen Spannung möglichst klein ist.)

Im Anschluss an diese Aufgabe erfolgt das nächste Testat mit Vorführung des Programmcodes bevor Sie mit dem nächsten Aufgabenblock beginnen können.

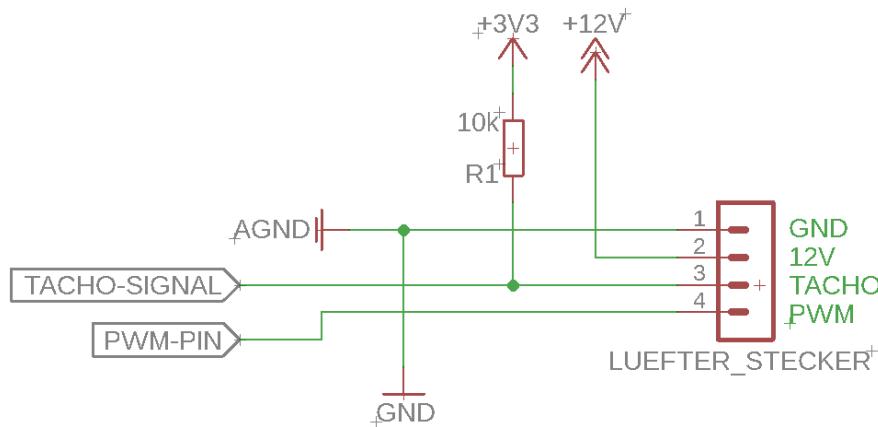
## 4.3 Block 3: PWM-Ansteuerung / Drehzahlmessung

Die Aufgaben zuvor dienten dazu die einzelnen Strukturen sowie die Hardware des Mikrocontrollers genauer kennenzulernen, die Sie für den weiteren Verlauf benötigen. In diesem Block sammeln Sie praktische Erfahrungen anhand eines Lüfters, wie mit verschiedenen Messmethoden die selbe Messgröße (Drehzahl) ermittelt werden kann.

### 4.3.1 Lüfter-Ansteuerung

In nahezu jedem Computer aber auch in vielen anderen Geräten, die größere elektrische Leistungen umsetzen, kommen Lüfter zum Abführen der Wärme zum Einsatz. Insbesondere im Computer-Bereich haben sich die sogenannten PWM-Lüfter etabliert (siehe Kapitel 2.7.5). In dieser Aufgabe wird ein solcher Lüfter über die PWM-Peripherie des Mikrocontrollers angesteuert. Konkret sollen folgende Aufgaben umgesetzt werden:

- Lüfter an den Mikrocontroller und die externe Spannungsversorgung gemäß des nachfolgenden Schaltplans in Abb. 4.2 anschließen. (**ACHTUNG:** Achten Sie besonders darauf, dass die externe Versorgungsspannung (12 V) keinen der Mikrocontroller-Pins berührt. Eine Spannung über 5 V führt nahezu unausweichlich zum Zerstören des Mikrocontrollers. (Das Tacho-Signal muss für diese Aufgabe nicht angeschlossen werden.)



**Abbildung 4.2:** Anschluss des Lüfters an die Spannungsquelle und den Mikrocontroller. Die Massen von dem Mikrocontroller (GND) und dem Netzteil (AGND) müssen für die korrekte Signalerfassung miteinander verbunden werden.

- Timer inklusive PWM an einem passenden Mikrocontroller-Pin aktivieren und PWM Frequenz gemäß der Vorgaben aus Kapitel 2.7.5 einstellen.
- PWM-Sollwert als duty cycle (in %) über den Encoder oder die UART-Schnittstelle vorgeben. (Achten Sie auf den zulässigen Wertebereich und limitieren Sie ggf. die übergebenen Werte)
- Aktuellen duty cycle in % auf dem Display ausgeben.

### 4.3.2 Drehzahlmessung mit Tacho-Signal

Der im Praktikum zum Einsatz kommende Lüfter bietet einen "Rückkanal" an, über den die aktuelle Drehzahl des Lüfters ermittelt/ausgelesen werden kann. An der gelben (Tacho)-Anschlussleitung des Lüfters werden pro Umdrehung zwei Pulse ausgegeben (siehe Abschnitt 2.7.5). In dieser Aufgabe soll mit Hilfe des Mikrocontrollers die Drehzahl ermittelt und auf dem Display angezeigt werden. Insgesamt sollen folgende Aufgaben umgesetzt werden:

- Schließen Sie das Tacho-Signal an einen passenden Mikrocontroller-Pin gemäß des Schaltplans aus Abb. 4.2 an. (**ACHTUNG:** Verwechseln Sie nicht die Lüfter(12 V)- und die Mikrocontroller(3,3 V)-Versorgungsspannung.)
- Aktivieren und konfigurieren Sie die Peripherie des Mikrocontrollers zum Messen von externen Ereignissen. (**Hinweis:** Pulse (Pegelwechsel) können entweder mit einem externen Interrupt oder mit Hilfe eines Timers (*Input capture*

*direct mode* und der Callback-Funktion *HAL\_TIM\_IC\_CaptureCallback()* detektiert werden.)

- Messen Sie die Dauer zwischen den zwei aufeinander folgenden Pulsen und bestimmen Sie daraus die Drehfrequenz in Hz und in revolutions per minute (rpm) (**Hinweis:** pro Umdrehung werden zwei Pulse Ausgegeben. CNT-Register eines Timers kann für die Zeitmessung verwendet werden.)
- Aktuelle Drehzahl (in Hz oder rpm) auf dem Display ausgeben.
- Zusatzaufgabe: Eine Kennlinie **PWM duty cycle über Drehzahl** aufnehmen und graphisch (z.B. mit Excel) darstellen.

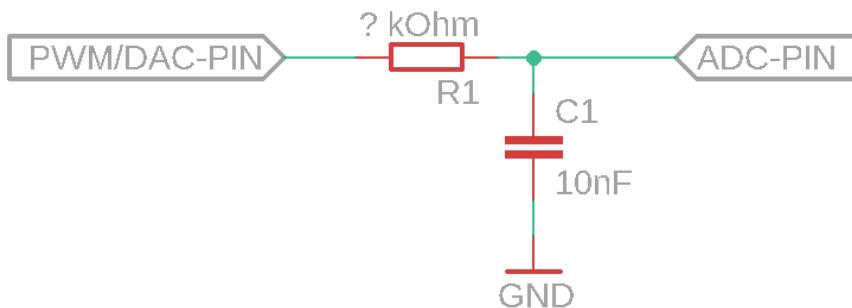
#### 4.4 Block 4 - Abschlussaufgabe: Signalerzeugung mittels PWM und DAC sowie Signalaufzeichnung mit einem ADC.

In der modernen Leistungs-(Energie-)technik ist die Pulsweitenmodulation nicht mehr wegzudenken. Überall, wo die Spannung (und folglich auch die Leistung) eines Systems über einen größeren Bereich, schnell und möglichst effizient eingestellt werden muss, kommt die PWM zum Einsatz. Eine praktische Anwendung haben Sie bereits bei der Lüftersteuerung kennengelernt. Aufgrund der Trägheit des Lüfters konnte die über die PWM vorgegebene Spannung abschnittsweise konstant bzw. als Gleichspannung betrachtet werden. Sollen mit der PWM Spannungsverläufe im zwei- oder dreistelligen Hertz-Bereich oder sogar im Kilohertz-Bereich erzeugt werden, so muss sowohl die PWM-Frequenz relativ hoch gewählt, als auch die Timer-Frequenz sehr hoch eingestellt (zweistelliger MHz-Bereich) werden, um damit verschiedene Signalabstufungen (Spannungslevel siehe Abschnitte 2.2 und 2.3) realisieren zu können. Des weiteren ist es erforderlich, bei Anwendungen, die ein kontinuierliches Spannungssignal (ohne Stufen) voraussetzen, die mit Hilfe der PWM generierte Rechteck-Spannung (nur zwei Spannungspegel) zu glätten bzw. filtern. Damit die hochfrequente PWM-Frequenz nicht im generierten Signalverlauf auftritt, muss die Grenzfrequenz des Filters deutlich unterhalb der PWM-Frequenz liegen. Als grober Richtwert gilt: die Grenzfrequenz sollte mindestens eine Dekade (eine Zehnerpotenz) unterhalb der PWM-Frequenz liegen. Hier ist darauf zu achten, dass die gewünschte Signalform bzw. Frequenzen, die im Nutzsignal auftreten werden, nicht von dem Tiefpassfilter beeinflusst werden.

Eine häufige Aufgabe in der Messtechnik besteht darin, einen Signalverlauf aufzuzeichnen und diesen zum Computer für weitere Untersuchungen zu übertragen. Eine weitere Anwendung ist, ein kontinuierliches (Mess-)Signal aufzuzeichnen, auszuwerten und zeitnah (im besten Fall in Echtzeit) ein Ergebnis auszugeben bzw. auf das Ergebnis zu reagieren und beispielsweise einen neuen Sollwert vorzugeben oder einen Trigger-Puls auszulösen.

In der Abschlussaufgabe werden Sie sich einerseits mit der Signalgenerierung mittels PWM, Signalabtastung und der Übertragung zum Computer beschäftigen. Andererseits sollen mit Hilfe des DAC sinusförmige Signalverläufe generiert und mit einem variablen Tiefpassfilter gedämpft (gefiltert) werden. Dabei sollen die Spannungsverläufe vor und nach dem Filter mit dem ADC aufgezeichnet und der Phasenversatz sowie die Amplitudenänderung auf dem Mikrocontroller bestimmt werden. Nachfolgend sind die Arbeitsschritte/Anforderungen aufgelistet, die umgesetzt werden sollen.

- Schließen Sie gemäß des nachfolgenden Schaltplans (Abb. 4.3) das RC-Tiefpassfilter an einem der PWM-Ausgänge an. Der Wert des Widerstands  $R_1$  kann frei zwischen  $1\text{ k}\Omega$  und  $100\text{ k}\Omega$  ausgewählt werden, sollte jedoch in der Dokumentation angeben und begründet werden.



**Abbildung 4.3:** RC-Tiefpassfilter zum Anschluss an den Mikrocontroller. Durch die Variation des Widerstandes  $R_1$  wird die Grenzfrequenz des Filters beeinflusst (vgl. Kap. 2.7.7).

- Aktivieren und konfigurieren Sie die PWM-Peripherie an einem dazu passenden Pin. Es sollte möglich sein mit Hilfe der PWM periodische Signale mit mindestens 32 Punkten pro Periode bis zu einer Frequenz von  $5\text{ kHz}$  und mindestens 9 Bit-Auflösung zu generieren. Geben Sie die maximal mögliche Signalfrequenz für Ihre PWM-Konfiguration an. (**Hinweis:** Überlegen Sie, im welchen Zusammenhang die Parameter Timer-Frequenz, PWM-Auflösung

und Anzahl der Punkte pro Signalperiode miteinander stehen. Damit kann die maximal erreichbare Signalfrequenz errechnet werden.)

- Für die kontinuierliche Erzeugung eines periodischen Signals mittels PWM oder DAC empfiehlt es sich den DMA zu verwenden. Aktivieren Sie einen passenden DMA-Kanal, mit dem die Daten (der Signalverlauf) aus einem Speicherbereich in den **PWM-Counter-Compare-Register** (CCR) zyklisch übertragen werden können.
- Implementieren Sie eine Methode zur Erzeugung einer PWM/DAC-Tabelle, die eine Periode eines Kosinussignals durch die Vorgabe der Anzahl der Punkte pro Periode und der Amplitude (vollständiger Aussteuerbereich des PWM/DAC) im dynamischen Speicher generiert. Achten Sie hier auf die Speicherverwaltung. Beachten Sie auch, dass das generierte Signal bei periodischer Fortsetzung keine Unstetigkeiten aufweisen darf (**Hinweis:**  $\cos(0) = \cos(2\pi)$ ).
- Aktivieren und konfigurieren Sie einen ADC-Kanal zum Abtasten und Aufzeichnen eines analogen Pins. Der ADC soll über einen DMA-Stream die aufgenommenen Daten in einen vorher allozierten Speicherbereich übertragen.
- Implementieren Sie die Funktionalität zum Steuern der PWM-Signalgenerierung über UART. Es sollte möglich sein über die UART-Schnittstelle die Frequenz und die Amplitude des Sinussignals vorzugeben, Signalgenerierung zu aktivieren und deaktivieren und die Aufzeichnung einer vorgegebenen Periodenanzahl zu starten sowie anschließend zum Computer zu übertragen. Die Signalspeicherung/-aufzeichnung sollte über den DMA erfolgen. Achten Sie darauf, dass bei der dynamischen Allozierung des Speicherbereichs entsprechend der Vorgaben (Periodenanzahl, Signalfrequenz und Anzahl der Punkte pro Periode) nicht mehr als 10 kB Speicher reserviert werden. Geben Sie ggf. eine Warnung aus, falls die vorgegeben Werte nicht eingestellt werden können und setzen Sie stattdessen die minimal/maximal möglichen Werte um. Folgende Randbedingungen sollen miteinbezogen werden:
  - Signalfrequenzbereich: 50 Hz bis 5 kHz
  - PWM-Auflösung: 9-Bit
  - Minimale Punkteanzahl pro Periode: 32 Punkte
- Es empfiehlt sich die Signalaufzeichnung zum Beginn einer Periode zu starten. Dazu kann i.d.R. die **XferCpltCallback**-Funktion des DMA-Streams (für die

kontinuierliche Aktualisierung der PWM/DAC-Tabelle) verwendet werden. Jedoch wird diese bei der Verwendung von HAL\_TIM\_PWM\_Start\_DMA überschrieben. Daher ist die Funktion HAL\_TIM\_PWM\_PulseFinishedCallback als Alternative zu nutzen. Diese wird automatisch am Ende jedes PWM-Pulses aufgerufen sobald sie implementiert wurde. Die korrekte Implementierung kann Listing. 4.2 entnommen werden.

```
1 void HAL_TIM_PWM_PulseFinishedCallback(TIM_HandleTypeDef *  
2     htim) {  
3  
4 // insert your code here  
5 }
```

**Listing 4.2:** PWM Pulse finished callback in non-blocking mode.

- Die aufgezeichneten Daten sollen über die UART-Schnittstelle zum Computer übertragen werden. Es empfiehlt sich dazu das Programm HTerm zu verwenden. Die im HTerm empfangenen Daten sollen in Excel oder in einem anderem Programm (MATLAB, Python, etc.) zum Erstellen von Diagrammen importiert werden. Das aufgezeichnete Signal soll über die Zeit in einem Diagramm aufgetragen und in der Ausarbeitung dargestellt sowie beschrieben werden. (Achten Sie auf die korrekte Achsenbeschriftung. Die Y-Achse kann entweder in Volt oder als Digits dargestellt werden.)
- Nun wird der DAC für die Signalgenerierung verwendet. Schließen Sie dazu den RC-Tiefpassfilter aus Abb. 4.3 an den freien DAC-Ausgang des Mikrocontrollers an.
- Der DAC soll ähnlich wie die PWM über den DMA zyklisch mit Daten versorgt werden. Konfigurieren Sie daher die Peripherie (DAC und DMA) so, dass nach dem Start des DAC das vorher allozierte Kosinussignal (Signaltablette) zum DAC-Ausgang mit dem DMA übertragen werden kann.
- Anders als bei der PWM, kann der DAC nicht direkt mit einem definierten Takt konfiguriert und anschließend unabhängig von anderer Peripherie zyklisch betrieben werden. Wird also ein Wert in den DAC-Ausgangsregister geschrieben, behält der DAC-Ausgang solange den damit vorgegebenen Signalpegel, bis ein neuer Wert in das Register geschrieben wird. Für eine kontinuierliche Aktualisierung der Werte muss der DAC daher periodisch angesteuert werden. Dies lässt sich mit einem Timer realisieren, in dem

ein Timer-Event, welches beispielsweise nach jedem Timer-Überlauf von der Hardware generiert wird, als "Takt" - bzw. "Trigger" -Eingang des DAC konfiguriert wird. Konfigurieren Sie daher auch einen passenden Timer, mit dem der DAC getriggert werden kann. (Vergessen Sie nicht das jeweilige **Trigger Out event** des Timers als Trigger in der DAC Konfiguration auszuwählen.)

- Ähnlich wie bei PWM soll hier ebenfalls eine Funktionalität zum Steuern des DAC (Start/Stop, Frequenz- und Amplitudenvorgabe, Signalaufzeichnung und Übertragung, Berechnung der Grenzfrequenz, Amplitude und Phasenverschiebung) implementiert werden.
- Für die Dokumentation Ihrer Arbeit sollen die Signale zu Darstellungszwecken von beiden ADC-Kanälen ebenfalls zum Computer übertragen werden. Erweitern Sie daher die bereits für die PWM-Signalübertragung erstellte Funktionalität auf das Übertragen von zwei Datenkanälen. (Überlegen Sie dafür eine geeignete Methode zum eindeutigen Differenzieren (Unterscheiden) von Daten verschiedener Kanäle.)
- Stellen Sie in einem Diagramm (ähnlich wie in Abb. 2.15) das aufgenommene Ein- und Ausgangssignal für die Signalfrequenz von 2,5 kHz und den Widerstand des Filters von xx k $\Omega$  dar. (Achten Sie auf die Achsenbeschriftung und die Kennzeichnung der beiden Signale.)

Fassen Sie Ihr Vorgehen bei den einzelnen Punkten zusammen und fügen Sie alle nötigen Berechnungen sowie wichtige Programmteile in einer Dokumentation zusammen. Dem Leser sollte klar werden, wieso Sie den gewählten Weg gegangen sind. So sollen Sie z.B. erklären, aus welchen Gründen der Timer X und nicht der Timer Y verwendet wurde. Darüber hinaus sollen Messergebnisse tabellarisch und/oder grafisch aufgearbeitet werden. **Außerdem ist es enorm wichtig alle Quellen anzugeben, die Sie verwendet haben.**

## Literaturverzeichnis

- [1] *Understanding ADC Parameters*. Atmel Corporation, 2013.
- [2] Herbert Bernstein. *Mikrocontroller : Grundlagen der Hard- und Software der Mikrocontroller ATtiny2313, ATtiny26 und ATmega32*. Springer Vieweg, Wiesbaden, 2015. ISBN 9783658028121. URL <http://dx.doi.org/10.1007/978-3-658-02813-8>.
- [3] Jürgen Beyerer, Fernando Puente León, and Christian Frese. *Automatische Sichtprüfung : Grundlagen, Methoden und Praxis der Bildgewinnung und Bildauswertung*. Springer Vieweg, Berlin, Heidelberg, 2. aufl. 2016 edition, 2016. ISBN 9783662477854. URL <http://dx.doi.org/10.1007/978-3-662-47786-1>.
- [4] E. Oran Brigham. *The fast Fourier transform and its applications*. Prentice-Hall International, 1988. ISBN 0133075478.
- [5] Uwe Brinkschulte and Theo Ungerer. *Mikrocontroller und Mikroprozessoren*. eXamen.press. Springer, Berlin, Heidelberg, 2010. ISBN 9783642053979. URL <http://dx.doi.org/10.1007/978-3-642-05398-6>.
- [6] Helmut Bähring. *Anwendungsorientierte Mikroprozessoren : Mikrocontroller und Digitale Signalprozessoren*. eXamen.press. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 9783642122910. URL <http://dx.doi.org/10.1007/978-3-642-12292-7>.
- [7] Winfried Gehrke, Marco Winzker, Klaus Urbanski, and Roland Woitowitz. *Digitaltechnik : Grundlagen, VHDL, FPGAs, Mikrocontroller*. Springer-Lehrbuch. Springer Vieweg, Berlin, Heidelberg, 7. aufl. 2016 edition, 2016. ISBN 9783662497302. URL <http://dx.doi.org/10.1007/978-3-662-49731-9>.
- [8] Peter Lehmann, Holger Knell, Markus Schake, and Markus Schulz. *analoge und digitale Messverfahren - Analoge Messtechnik*. Vorlesungsskript, Fachgebiet Messtechnik, FB Elektrotechnik/Informatik, Universitat Kassel, 2018.

- [9] Martin Meyer. *Signalverarbeitung - Analoge und digitale Signale, Systeme und Filter*. Springer Vieweg, Wiesbaden, 7. aufl. 2014 edition, 2014. ISBN 9783658026127. URL <https://doi.org/10.1007/978-3-658-02612-7>.
- [10] Johann Siegl. *Schaltungstechnik – Analog und gemischt analog/digital : Entwicklungsmethodik, Funktionsschaltungen, Funktionsprimitive von Schaltkreisen*. Springer-Lehrbuch. Springer, Berlin, Heidelberg, 3 edition, 2009. ISBN 9783540683698. URL <http://dx.doi.org/10.1007/978-3-540-68370-4>.
- [11] STMicroelectronics. *Using the STM32F2, STM32F4 and STM32F7 Series DMA controller*, 2016. URL [https://www.st.com/content/ccc/resource/technical/document/application\\_note/27/46/7c/ea/2d/91/40/a9/DM00046011.pdf/files/DM00046011.pdf/jcr:content/translations/en.DM00046011.pdf](https://www.st.com/content/ccc/resource/technical/document/application_note/27/46/7c/ea/2d/91/40/a9/DM00046011.pdf/files/DM00046011.pdf/jcr:content/translations/en.DM00046011.pdf).
- [12] STMicroelectronics. *STM32F446 Datasheet*, 2016. URL <https://www.st.com/resource/en/datasheet/stm32f446re.pdf>.
- [13] STMicroelectronics. *Getting started with STM32 Nucleo board software development tools*, 2016. URL [https://www.st.com/resource/en/user\\_manual/dm00105928.pdf](https://www.st.com/resource/en/user_manual/dm00105928.pdf).
- [14] STMicroelectronics. *Audio and waveform generation using the DAC in STM32 microcontrollers*, 2017. URL [https://www.st.com/resource/en/application\\_note/cd00259245.pdf](https://www.st.com/resource/en/application_note/cd00259245.pdf).
- [15] STMicroelectronics. *General-purpose timer cookbook*, 2017. URL [https://www.st.com/resource/en/application\\_note/dm00236305.pdf](https://www.st.com/resource/en/application_note/dm00236305.pdf).
- [16] STMicroelectronics. *STM32 GPIO configuration for hardware settings and low-power consumption*, 2017. URL [https://www.st.com/resource/en/application\\_note/dm00315319.pdf](https://www.st.com/resource/en/application_note/dm00315319.pdf).
- [17] STMicroelectronics. *STM32 Nucleo-64 boards*, 2017. URL [https://www.st.com/resource/en/user\\_manual/dm00105823.pdf](https://www.st.com/resource/en/user_manual/dm00105823.pdf).
- [18] STMicroelectronics. *Description of STM32F4 HAL and LL drivers*, 2017. URL [https://www.st.com/resource/en/user\\_manual/dm00105879.pdf](https://www.st.com/resource/en/user_manual/dm00105879.pdf).
- [19] STMicroelectronics. *Getting started with STM32F4xxxx MCU hardware development*, 2018. URL [https://www.st.com/resource/en/application\\_note/dm00115714.pdf](https://www.st.com/resource/en/application_note/dm00115714.pdf).

- [20] STMicroelectronics. *Reference manual*, 2018. URL [https://www.st.com/resource/en/reference\\_manual/dm00135183.pdf](https://www.st.com/resource/en/reference_manual/dm00135183.pdf).
- [21] STMicroelectronics. *STM32CubeIDE quick start guide*, 2019. URL [https://www.st.com/resource/en/user\\_manual/dm00598966-stm32cubeide-quick-start-guide-stmicroelectronics.pdf](https://www.st.com/resource/en/user_manual/dm00598966-stm32cubeide-quick-start-guide-stmicroelectronics.pdf).
- [22] STMicroelectronics. *STM32CubeIDE installation guid*, 2019. URL [https://www.st.com/resource/en/user\\_manual/dm00603964-stm32cubeide-installation-guide-stmicroelectronics.pdf](https://www.st.com/resource/en/user_manual/dm00603964-stm32cubeide-installation-guide-stmicroelectronics.pdf).
- [23] Thundertronics. *TrueSTUDIO 9.0.1 Flash and run without debugging*, 2019. URL <http://thundertronics.com/2018/07/17/truestudio-9-0-1-flash-and-run-without-debugging/>.
- [24] Helmut Ulrich and Hubert Weber. *Laplace-, Fourier- und z-Transformation : Grundlagen und Anwendungen*. Springer Fachmedien Wiesbaden, Wiesbaden, 10., korrig. Aufl. 2017 edition, 2017. ISBN 9783658034504. URL <http://dx.doi.org/10.1007/978-3-658-03450-4>.
- [25] Wikipedia. Symbolrate, Baudrate. <https://de.wikipedia.org/wiki/Symbolrate>, 2019.

## Abkürzungen

<b>PWM</b>	Pulsweitenmodulation
<b>ADC</b>	Analog-Digital-Wandler
<b>DAC</b>	Digital-Analog-Wandler
<b>DMA</b>	Direct Memory Access
<b>USB</b>	Universal Serial Bus
<b>UART</b>	Universal Asynchronous Receiver Transmitter
<b>I<sup>2</sup>C</b>	Inter-Integrated Circuit
<b>LSB</b>	Least Significant Bit
<b>FPU</b>	Floating Point Unit
<b>GPIO</b>	General Purpose Input/Output
<b>IDE</b>	Integrated Development Environment
<b>HAL</b>	Hardware Abstraktion Layer
<b>rpm</b>	revolutions per minute

## **A Datenblätter**

### **A.1 TCRT5000**



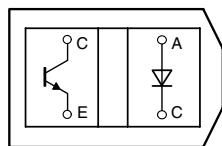
## TCRT5000, TCRT5000L

Vishay Semiconductors

### Reflective Optical Sensor with Transistor Output



19156\_2



19156\_1

Top view

#### FEATURES

- Package type: leaded
- Detector type: phototransistor
- Dimensions (L x W x H in mm): 10.2 x 5.8 x 7
- Peak operating distance: 2.5 mm
- Operating range within > 20 % relative collector current: 0.2 mm to 15 mm
- Typical output current under test:  $I_C = 1 \text{ mA}$
- Daylight blocking filter
- Emitter wavelength: 950 nm
- Lead (Pb)-free soldering released
- Compliant to RoHS directive 2002/95/EC and in accordance to WEEE 2002/96/EC



#### DESCRIPTION

The TCRT5000 and TCRT5000L are reflective sensors which include an infrared emitter and phototransistor in a leaded package which blocks visible light. The package includes two mounting clips. TCRT5000L is the long lead version.

#### APPLICATIONS

- Position sensor for shaft encoder
- Detection of reflective material such as paper, IBM cards, magnetic tapes etc.
- Limit switch for mechanical motions in VCR
- General purpose - wherever the space is limited

PRODUCT SUMMARY				
PART NUMBER	DISTANCE FOR MAXIMUM CTR <sub>rel</sub> <sup>(1)</sup> (mm)	DISTANCE RANGE FOR RELATIVE I <sub>out</sub> > 20 % (mm)	TYPICAL OUTPUT CURRENT UNDER TEST <sup>(2)</sup> (mA)	DAYLIGHT BLOCKING FILTER INTEGRATED
TCRT5000	2.5	0.2 to 15	1	Yes
TCRT5000L	2.5	0.2 to 15	1	Yes

#### Notes

(1) CTR: current transfere ratio,  $I_{out}/I_{in}$

(2) Conditions like in table basic characteristics/sensors

ORDERING INFORMATION			
ORDERING CODE	PACKAGING	VOLUME <sup>(1)</sup>	REMARKS
TCRT5000	Tube	MOQ: 4500 pcs, 50 pcs/tube	3.5 mm lead length
TCRT5000L	Tube	MOQ: 2400 pcs, 48 pcs/tube	15 mm lead length

#### Note

(1) MOQ: minimum order quantity

ABSOLUTE MAXIMUM RATINGS <sup>(1)</sup>				
PARAMETER	TEST CONDITION	SYMBOL	VALUE	UNIT
<b>INPUT (EMITTER)</b>				
Reverse voltage		V <sub>R</sub>	5	V
Forward current		I <sub>F</sub>	60	mA
Forward surge current	$t_p \leq 10 \mu\text{s}$	I <sub>FSM</sub>	3	A
Power dissipation	$T_{amb} \leq 25^\circ\text{C}$	P <sub>V</sub>	100	mW
Junction temperature		T <sub>j</sub>	100	°C

**TCRT5000, TCRT5000L**

Vishay Semiconductors

Reflective Optical Sensor with  
Transistor Output**ABSOLUTE MAXIMUM RATINGS (1)**

PARAMETER	TEST CONDITION	SYMBOL	VALUE	UNIT
<b>OUTPUT (DETECTOR)</b>				
Collector emitter voltage		$V_{CEO}$	70	V
Emitter collector voltage		$V_{ECO}$	5	V
Collector current		$I_C$	100	mA
Power dissipation	$T_{amb} \leq 55^\circ\text{C}$	$P_V$	100	mW
Junction temperature		$T_j$	100	°C
<b>SENSOR</b>				
Total power dissipation	$T_{amb} \leq 25^\circ\text{C}$	$P_{tot}$	200	mW
Ambient temperature range		$T_{amb}$	- 25 to + 85	°C
Storage temperature range		$T_{stg}$	- 25 to + 100	°C
Soldering temperature	2 mm from case, $t \leq 10$ s	$T_{sd}$	260	°C

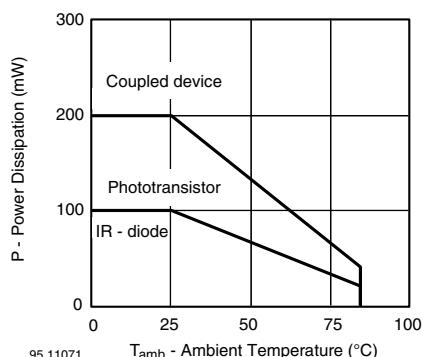
**Note**(1)  $T_{amb} = 25^\circ\text{C}$ , unless otherwise specified**ABSOLUTE MAXIMUM RATINGS**

Fig. 1 - Power Dissipation Limit vs. Ambient Temperature

**BASIC CHARACTERISTICS (1)**

PARAMETER	TEST CONDITION	SYMBOL	MIN.	TYP.	MAX.	UNIT
<b>INPUT (EMITTER)</b>						
Forward voltage	$I_F = 60$ mA	$V_F$		1.25	1.5	V
Junction capacitance	$V_R = 0$ V, $f = 1$ MHz	$C_J$		17		pF
Radiant intensity	$I_F = 60$ mA, $t_p = 20$ ms	$I_e$			21	mW/sr
Peak wavelength	$I_F = 100$ mA	$\lambda_P$	940			nm
Virtual source diameter	Method: 63 % encircled energy	$d$		2.1		mm
<b>OUTPUT (DETECTOR)</b>						
Collector emitter voltage	$I_C = 1$ mA	$V_{CEO}$	70			V
Emitter collector voltage	$I_e = 100$ µA	$V_{ECO}$	7			V
Collector dark current	$V_{CE} = 20$ V, $I_F = 0$ A, $E = 0$ lx	$I_{CEO}$		10	200	nA
<b>SENSOR</b>						
Collector current	$V_{CE} = 5$ V, $I_F = 10$ mA, $D = 12$ mm	$I_C$ (2) (3)	0.5	1	2.1	mA
Collector emitter saturation voltage	$I_F = 10$ mA, $I_C = 0.1$ mA, $D = 12$ mm	$V_{CEsat}$ (2) (3)			0.4	V

**Note**(1)  $T_{amb} = 25^\circ\text{C}$ , unless otherwise specified

(2) See figure 3

(3) Test surface: mirror (Mfr. Spindler a. Hoyer, Part No. 340005)



## TCRT5000, TCRT5000L

Reflective Optical Sensor with  
Transistor Output

Vishay Semiconductors

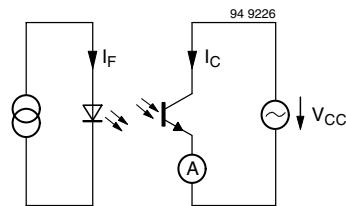


Fig. 2 - Test Circuit

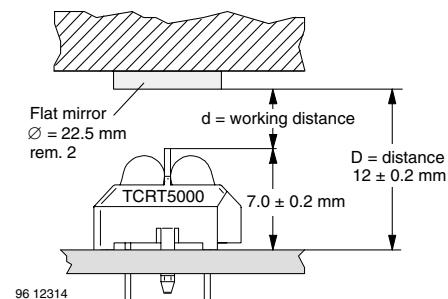
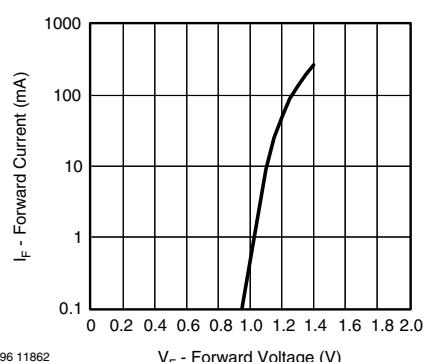


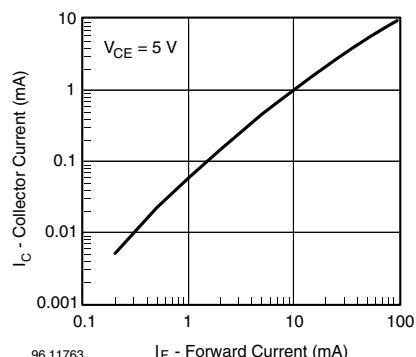
Fig. 3 - Test Circuit

### BASIC CHARACTERISTICS

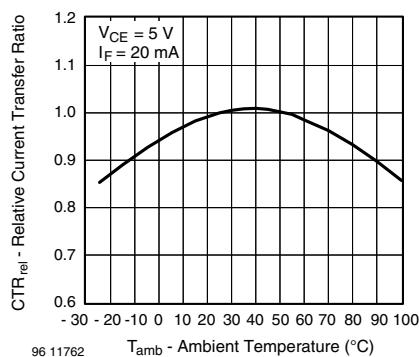
$T_{amb} = 25^\circ\text{C}$ , unless otherwise specified



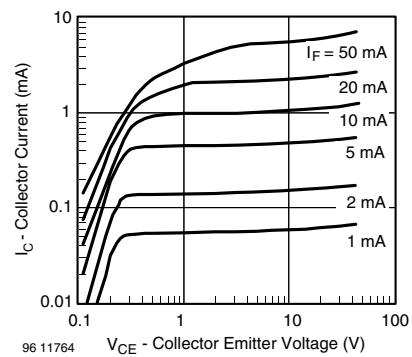
96 11862      Fig. 4 - Forward Current vs. Forward Voltage



96 11763      Fig. 6 - Collector Current vs. Forward Current



96 11762      Fig. 5 - Relative Current Transfer Ratio vs. Ambient Temperature



96 11764      Fig. 7 - Collector Emitter Saturation Voltage vs. Collector Current

**TCRT5000, TCRT5000L**

Vishay Semiconductors

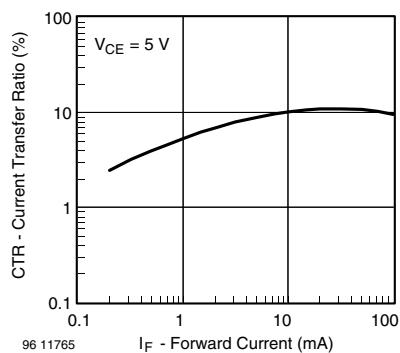
Reflective Optical Sensor with  
Transistor Output

Fig. 8 - Current Transfer Ratio vs. Forward Current

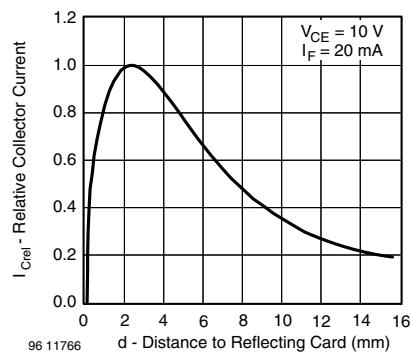
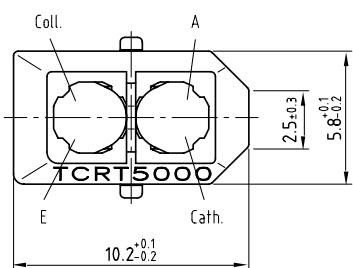
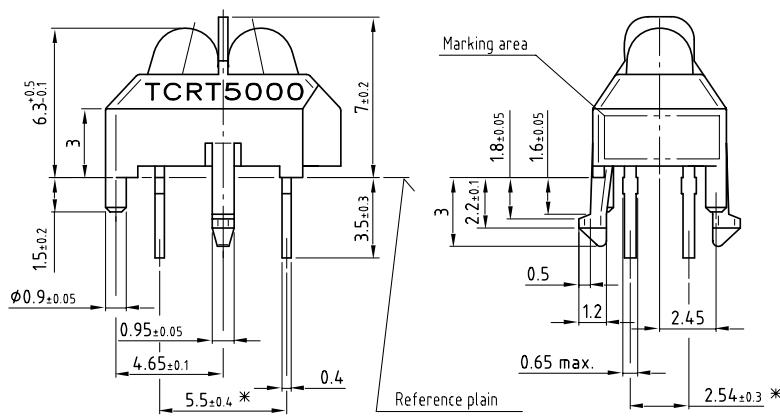
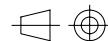


Fig. 9 - Relative Collector Current vs. Distance

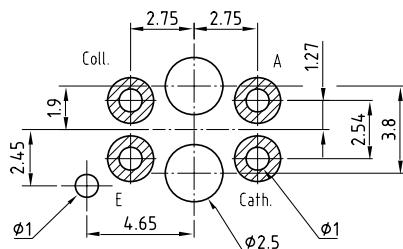
**PACKAGE DIMENSIONS** in millimeters, **TCRT5000**

\* Tolerances related to reference plain

weight: ca. 0.23g

Technical drawings  
according to DIN  
specifications

## Footprint Top View



Drawing-No.: 6.550-5096.01-4  
Issue: 4; 11.04.02  
96 12073

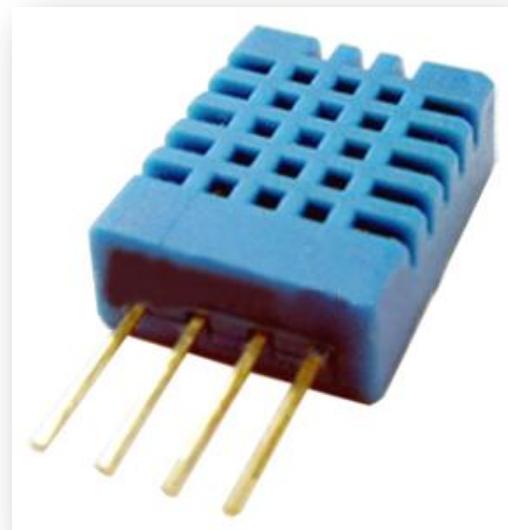
## **A.2 DHT11**

# DHT 11 Humidity & Temperature Sensor

---

## 1. Introduction

DHT11 Temperature & Humidity Sensor features a temperature & humidity sensor complex with a calibrated digital signal output. By using the exclusive digital-signal-acquisition technique and temperature & humidity sensing technology, it ensures high reliability and excellent long-term stability. This sensor includes a resistive-type humidity measurement component and an NTC temperature measurement component, and connects to a high-performance 8-bit microcontroller, offering excellent quality, fast response, anti-interference ability and cost-effectiveness.



Each DHT11 element is strictly calibrated in the laboratory that is extremely accurate on humidity calibration. The calibration coefficients are stored as programmes in the OTP memory, which are used by the sensor's internal signal detecting process. The single-wire serial interface makes system integration quick and easy. Its small size, low power consumption and up-to-20 meter signal transmission making it the best choice for various applications, including those most demanding ones. The component is 4-pin single row pin package. It is convenient to connect and special packages can be provided according to users' request.

## 2. Technical Specifications:

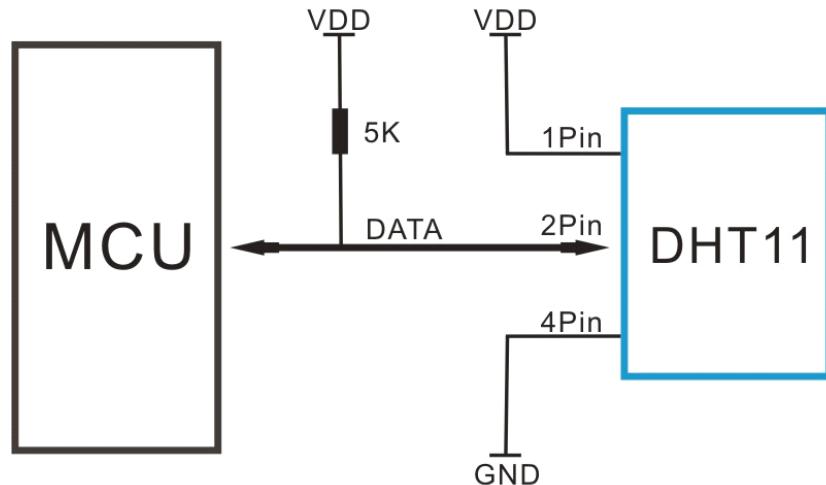
### Overview:

Item	Measurement Range	Humidity Accuracy	Temperature Accuracy	Resolution	Package
DHT11	20-90%RH 0-50 °C	±5%RH	±2°C	1	4 Pin Single Row

### Detailed Specifications:

Parameters	Conditions	Minimum	Typical	Maximum
<b>Humidity</b>				
<b>Resolution</b>		1%RH	1%RH	1%RH
			8 Bit	
<b>Repeatability</b>			±1%RH	
<b>Accuracy</b>	25 °C		±4%RH	
	0-50 °C			±5%RH
<b>Interchangeability</b>	Fully Interchangeable			
<b>Measurement Range</b>	0 °C	30%RH		90%RH
	25 °C	20%RH		90%RH
	50 °C	20%RH		80%RH
<b>Response Time (Seconds)</b>	1/e(63%) 25 °C , 1m/s Air	6 S	10 S	15 S
<b>Hysteresis</b>			±1%RH	
<b>Long-Term Stability</b>	Typical		±1%RH/year	
<b>Temperature</b>				
<b>Resolution</b>		1 °C	1 °C	1 °C
		8 Bit	8 Bit	8 Bit
<b>Repeatability</b>			±1 °C	
<b>Accuracy</b>		±1 °C		±2 °C
<b>Measurement Range</b>		0 °C		50 °C
<b>Response Time (Seconds)</b>	1/e(63%)	6 S		30 S

### 3. Typical Application (Figure 1)



**Figure 1 Typical Application**

Note: 3Pin – Null; MCU = Micro-computer Unite or single chip Computer

When the connecting cable is shorter than 20 metres, a 5K pull-up resistor is recommended; when the connecting cable is longer than 20 metres, choose a appropriate pull-up resistor as needed.

### 4. Power and Pin

DHT11's power supply is 3-5.5V DC. When power is supplied to the sensor, do not send any instruction to the sensor in within one second in order to pass the unstable status. One capacitor valued 100nF can be added between VDD and GND for power filtering.

### 5. Communication Process: Serial Interface (Single-Wire Two-Way)

Single-bus data format is used for communication and synchronization between MCU and DHT11 sensor. One communication process is about 4ms.

Data consists of decimal and integral parts. A complete data transmission is **40bit**, and the sensor sends **higher data bit** first.

**Data format:** 8bit integral RH data + 8bit decimal RH data + 8bit integral T data + 8bit decimal T data + 8bit check sum. If the data transmission is right, the check-sum should be the last 8bit of "8bit integral RH data + 8bit decimal RH data + 8bit integral T data + 8bit decimal T data".

## 5.1 Overall Communication Process (Figure 2, below)

When MCU sends a start signal, DHT11 changes from the low-power-consumption mode to the running-mode, waiting for MCU completing the start signal. Once it is completed, DHT11 sends a response signal of 40-bit data that include the relative humidity and temperature information to MCU. Users can choose to collect (read) some data. Without the start signal from MCU, DHT11 will not give the response signal to MCU. Once data is collected, DHT11 will change to the low-power-consumption mode until it receives a start signal from MCU again.

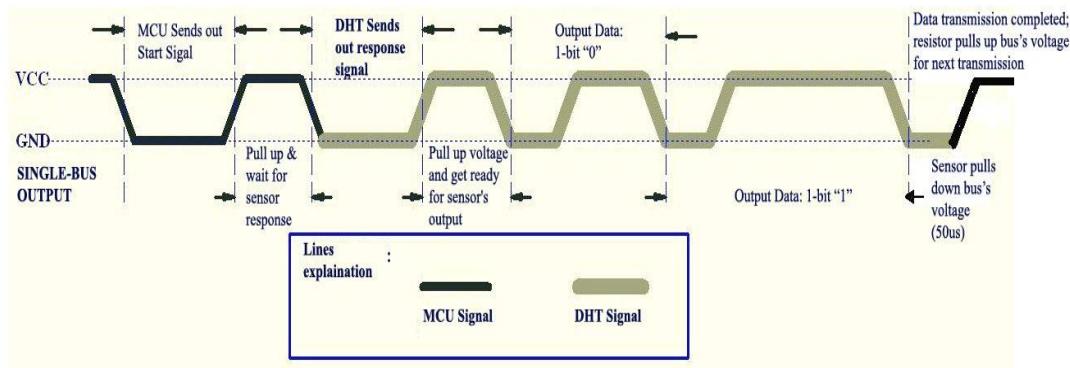


Figure 2 Overall Communication Process

## 5.2 MCU Sends out Start Signal to DHT (Figure 3, below)

Data Single-bus free status is at high voltage level. When the communication between MCU and DHT11 begins, the programme of MCU will set Data Single-bus voltage level from high to low and this process must take at least 18ms to ensure DHT's detection of MCU's signal, then MCU will pull up voltage and wait 20-40us for DHT's response.

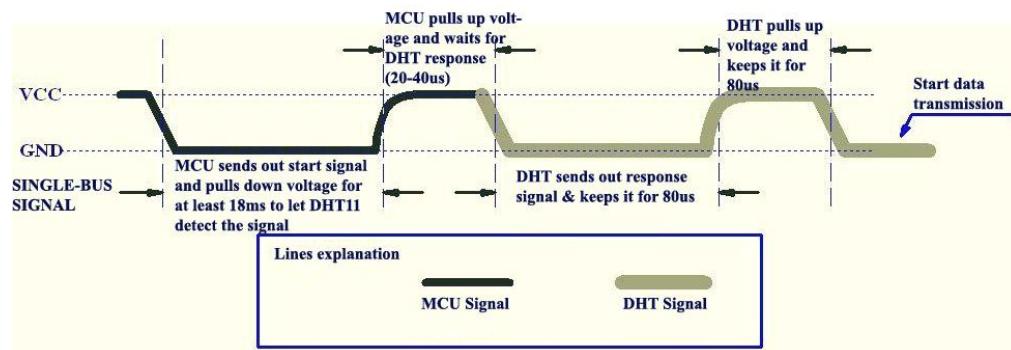


Figure 3 MCU Sends out Start Signal & DHT Responses

### 5.3 DHT Responses to MCU (Figure 3, above)

Once DHT detects the start signal, it will send out a low-voltage-level response signal, which lasts 80us. Then the programme of DHT sets Data Single-bus voltage level from low to high and keeps it for 80us for DHT's preparation for sending data.

When DATA Single-Bus is at the low voltage level, this means that DHT is sending the response signal. Once DHT sent out the response signal, it pulls up voltage and keeps it for 80us and prepares for data transmission.

When DHT is sending data to MCU, every bit of data begins with the 50us low-voltage-level and the length of the following high-voltage-level signal determines whether data bit is "0" or "1" (see Figures 4 and 5 below).

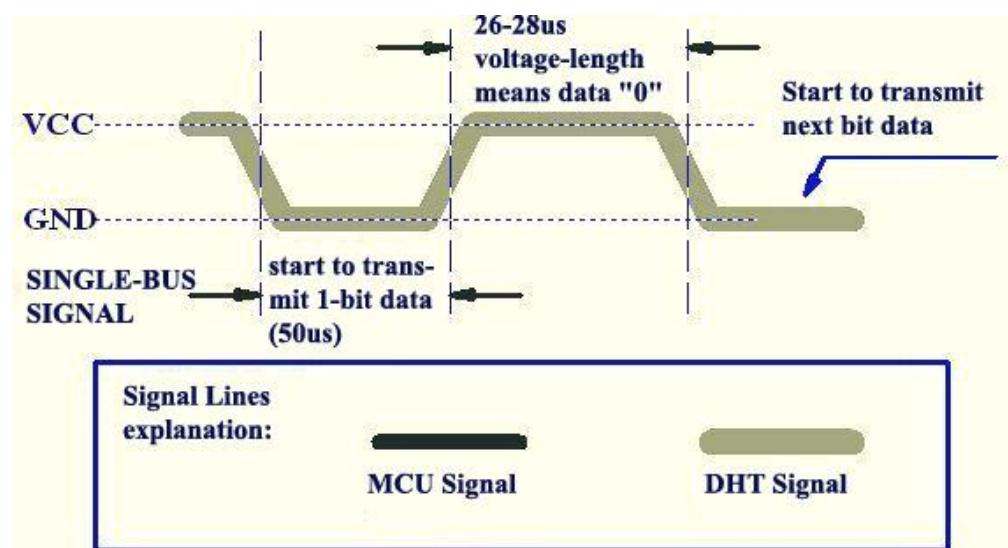


Figure 4 Data "0" Indication

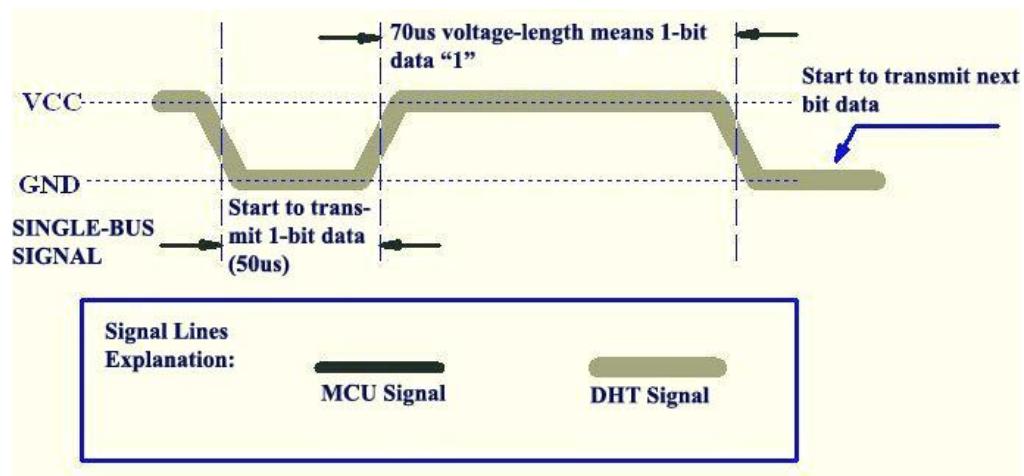


Figure 5 Data "1" Indication

If the response signal from DHT is always at high-voltage-level, it suggests that DHT is not responding properly and please check the connection. When the last bit data is transmitted, DHT11 pulls down the voltage level and keeps it for 50us. Then the Single-Bus voltage will be pulled up by the resistor to set it back to the free status.

## 6. Electrical Characteristics

VDD=5V, T = 25°C (unless otherwise stated)

	Conditions	Minimum	Typical	Maximum
Power Supply	DC	3V	5V	5.5V
Current Supply	Measuring	0.5mA		2.5mA
	Average	0.2mA		1mA
	Standby	100uA		150uA
Sampling period	Second	1		

Note: Sampling period at intervals should be no less than 1 second.

## 7. Attentions of application

### (1) Operating conditions

Applying the DHT11 sensor beyond its working range stated in this datasheet can result in 3%RH signal shift/discrepancy. The DHT11 sensor can recover to the calibrated status gradually when it gets back to the normal operating condition and works within its range. Please refer to (3) of

this section to accelerate its recovery. Please be aware that operating the DHT11 sensor in the non-normal working conditions will accelerate sensor's aging process.

#### (2) Attention to chemical materials

Vapor from chemical materials may interfere with DHT's sensitive-elements and debase its sensitivity. A high degree of chemical contamination can permanently damage the sensor.

#### (3) Restoration process when (1) & (2) happen

Step one: Keep the DHT sensor at the condition of Temperature 50~60Celsius, humidity <10%RH for 2 hours;

Step two: Keep the DHT sensor at the condition of Temperature 20~30Celsius, humidity >70%RH for 5 hours.

#### (4) Temperature Affect

Relative humidity largely depends on temperature. Although temperature compensation technology is used to ensure accurate measurement of RH, it is still strongly advised to keep the humidity and temperature sensors working under the same temperature. DHT11 should be mounted at the place as far as possible from parts that may generate heat.

#### (5) Light Affect

Long time exposure to strong sunlight and ultraviolet may debase DHT's performance.

#### (6) Connection wires

The quality of connection wires will affect the quality and distance of communication and high quality shielding-wire is recommended.

#### (7) Other attentions

\* Welding temperature should be below 260Celsius and contact should take less than 10 seconds.

\* Avoid using the sensor under dew condition.

\* Do not use this product in safety or emergency stop devices or any other occasion that failure of DHT11 may cause personal injury.

\* Storage: Keep the sensor at temperature 10-40°C, humidity <60%RH.

#### Disclaimer

This is a translated version of the manufacturer's data sheet. OSEPP is not responsible for the accuracy of the translated information.

### A.3 STM32F446 Blockdiagramm [12, S. 16]

## Description

STM32F446xC/E

Figure 3. STM32F446xC/E block diagram

