



Workshop-Unterlagen

Marcus Drobisch, Konglomerat e.V., Dresden, Germany, 2016

Licence under Creative Commons CC-by-nc-sa



Inhaltsverzeichnis

1 Übersicht	3
2 Einleitung	3
3 Grundlagen	4
3.1 Arduino	4
3.2 Hardware	5
3.3 Software	6
3.4 Programmierung	9
3.5 Simulation	9
4 Workshop Abend 1	10
4.1 Unser erstes Programm, Blink!	10
4.2 Überwachen von Tastern	13
4.3 Mit dem Timer zum Lauflicht	14
5 Workshop Abend 2	18
5.1 Hausaufgabe: Lauflicht	18
5.2 Ereignisse Zählen per Polling	18
5.3 Transistoren	19
5.4 Ereignisse Zählen per Interrupt	20
5.5 Ereignisse entprellen	21
5.6 Kommunikation mit dem PC, Hallo Welt!	22
5.7 Kontrollstrukturen in C	23
6 Workshop Abend 3	25
6.1 Servos ansteuern	25
6.2 Messen mit dem ADC. Der Joystick!	27
6.3 ADC und Servo	29
6.4 Bonus: Jede Menge Extras	31
6.5 Bonus: Steuerung und Visualisierung mit Python	31
6.6 Hausaufgabe: LED Matrix	31
7 Ausblick	34
7.1 Grenzen des Arduino / STM32Duino	34
7.2 Alternative Entwicklungsumgebungen	34
7.3 Alternative Mikrocontroller	34
7.4 Einplatinenrechner	34
7.5 Projekte	34

1 Übersicht

2 Einleitung

Arduino ist eine der beliebtesten Mikrocontroller-Plattformen. Durch die einfache Handhabung und den zahlreichen Möglichkeiten ist es besonders unter Künstlern und Bastlern weit verbreitet. Viele Projekte die sich früher eher umständlich und aufwändig realisieren ließen sind mit einem Arduino deutlich einfacher umzusetzen. Dies gilt besonders für den Nachbau. So lassen sich täglich neue Projekte zum Nachbau oder als Anregung in Foren, Blogs und Portalen finden. Um auch euch diese spannende Welt der Mikrocontroller näher zu bringen soll euch in diesem Kurs die Grundlagen an praktischen Beispielen vermitteln werden. Diese Unterlagen werden Kurs begleiten. Alle wichtigen Informationen sind hier zusammengefasst, sodass der Kurs ohne Abschreibübungen auskommt. Es empfehlen sich dennoch die ein oder andere Notiz bzw. Ergänzung zu den Folien und Quellcodes. Die Folien zu den Kursen werden auf Anfrage gern auch als Datei herausgegeben. Später können euch die Unterlagen als Nachschlagewerk in euren Projekten zur Seite stehen.

Ziel des Workshops ist ein möglichst praktischer Einstieg. Der Kurs ist daher an konkreten Beispielen aufgebaut. Diese behandeln die wichtigsten Elemente, die für eigene Projekte und das Verständnis größerer Nachbauprojekte wichtig sind. Zusätzlich wird an den Beispielen die Programmiersprache und konkretes Hardwarewissen zum Mikrocontroller erklärt. Die Präsentationen und die vorherige Simulation sollen dies zusätzlich unterstützen und werden von den Aufbau der Schaltungen begleitet. Später können die Beispiele als Sammlung funktionierender Bausteine verwendet werden um den Anfang eigener Lösungen zu vereinfachen.

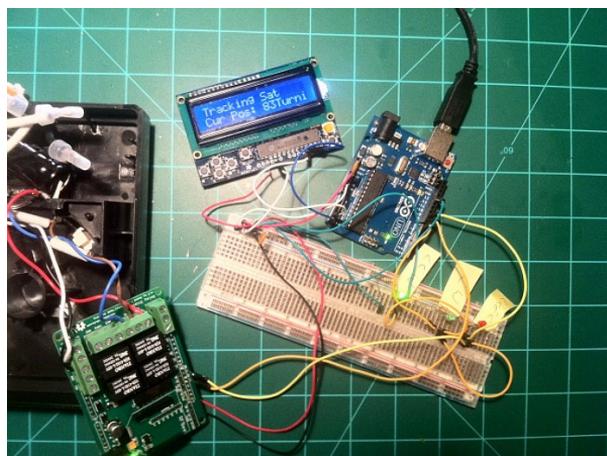


Abbildung 1: Spannende Arduino-Projekte gibt es überall im Netz. Quelle: <https://creative-chaos.com>

Der Kurs sollte daher auch als Einstieg verstanden werden. In den 3 Abenden kann allerdings nur ein grober Umriss erfolgen. Das aktiv Gelernte sowie die Unterlagen werden später als solide Grundlage für ein Aknügen oder einen Wiedereinstieg legen. Besonders der Anhang ist als Lektüre und Kompendium leserwert. Dieser wird auch in der Zukunft ausgebaut und weiterhin an alle Teilnehmer verteilt. Sinnvoll ist das gelernte später an eigenen Projekten zu festigen. Gute Programmierung und Stil kommen aus Erfahrung. Dazu kann besonders auch das Studium von Programmen anderer helfen. Im Generellen heißt es hier am Ball zu bleiben und den Spaß am Programmieren nicht zu verlieren. Um seinen Kenntnisstand zu testen wurden

die den Workshop Übungen eingebaut. Sie sind zur weiteren Festigung gedacht und umfassen theoretische wie auch praktische Aufgaben. Das Lösen dieser ist freiwillig und kann zum Beispiel im Anschluss an die Abende erfolgen. Generell ist eine weitere Beschäftigung zwischen den Abenden mit der Thematik sinnvoll. Die Mikrocontroller-Sets können in der Woche mit nach Hause genommen werden. Für ein einheitliches Niveau im Kurs und gegen Langeweile empfehle ich jedoch explizit, nicht vorzuarbeiten. Das bezieht sich auf die Aufgaben, die im Kurs durchgearbeitet werden. Sie sind so konzipiert, dass sie auch in der Abendveranstaltung schaffbar sind.

Abschließend ein paar letzte Informationen zum Kursleiter. Ich bin '85 geboren. Habe an der TU Dresden Elektrotechnik studiert. Diesen habe ich 2012 mit der Vertiefung Feinwerk und Mikrotechnik abgeschlossen und arbeite nun als leitender Entwickler bei einer Messtechnik-Firma aus Dresden. Hier entwickle ich sowohl die Elektronik als auch die Mikrocontroller-Programmierung (Firmware). Im Kurs werden die gleichen Vorgehensweise, Strukturen und Muster verwendet, wie ich sie in der täglichen Arbeit einsetze. Die aufkommende Wichtigkeit von Mikrocontrollern und den Spaß mit diesen darf ich daher täglich aufs neue erleben.

3 Grundlagen

3.1 Arduino

Mikrocontroller wie der von Arduino verwendete ATmega328 verstecken sich in vielen Produkten des alltäglichen Lebens von der Kaffeemaschine bis hin zum Satelitten steuern und automatisieren sie überall Geräte und Maschinen und machen diese zu den „intelligenten“ Systemen wie wir Sie kennen. Dabei ist vielen diese Allgegenwärtigkeit nicht bewusst, geschweige denn das die Funktionsweise und die Arbeit mit ihnen verstanden werden kann. Einer der Gründe hierfür ist das Mikrocontroller vor den 90er Jahren noch einer kleinen Zahl Entwicklern und Hobbyelektronikern vorbehalten. Die benötigten Programmiergeräte, die Software und Boards waren schlichtweg zu teuer oder mussten umständlich selbst gebaut werden. Zusätzlich war auch die Programmierung z.B. in Assembler und C in komplexen Entwicklungsumgebungen zu unverständlich. Ende der 90er Jahre änderte sich dies mit einhergehenden Verfall von Elektronik und Computerpreisen. Zusätzlich änderte sich bei vielen Firmen die Philosophie hin zu einem niederschwülligen Einstieg. Microchip und Atmel mit ihren Mikrocontroller waren hier 2 der ersten Vorreiter. So bot Atmel zusätzlich zu ihren günstigen Mikrocontroller eine freie Entwicklungsung. Auch die Hardware konnte für unter 20€ gebaut oder gekauft werden. Nicht viel später interessierte sich auch die Open Source Bewegung für die Boards und entwickelte freie Open Source Varianten der Entwicklungsumgebungen und Programmiergeräte. Die Popularität der AVR-Mikrocontroller von Atmel führte bald zu einer Dominanz in der Bastlerszene. Doch die Entwicklung mit Mikrocontroller war immernoch einer relativ kleinen Gruppe vorbehalten. Der Einstieg vornehmlich nun meist in C stellte weiterhin eine große Hürde da. Die Entwicklungsumgebungen (von Entwicklern für Entwickler konzipiert) waren riesig und erschlugen viele Einsteiger mit der schieren Anzahl von Funktionen und Möglichkeiten. Dies änderte sich mit dem erscheinen der Arduino-Umgebung. Mit der aufkommenden, teils eng mit Arduino verbundenen Makerbewegung wächst die Anzahl von Projekten und die dazugehörige Community noch immer. Da die Projekte meist gut dokumentiert und mit ihrem vollständigen Quellcode auf Blogs, auf Github gestellt werden, stellen sie für Anfänger eine gute Referenz dar. Hier können Ideen gesammelt, Projekte kombiniert und Zusammenarbeit koordiniert werden. Für nahezu jeden Bereich gibt es solche Beispiele zum Ansteuern von Schaltkreisen, Displays, Motoren und ganzen Geräten. Hinzu kommt die auf Arduino ausgerichtete Beschaffbarkeit von Modulen die direkt für die Arduino-Boards ausgelegt wurden

und im Internet als auch in Fachgeschäften erworben werden können.

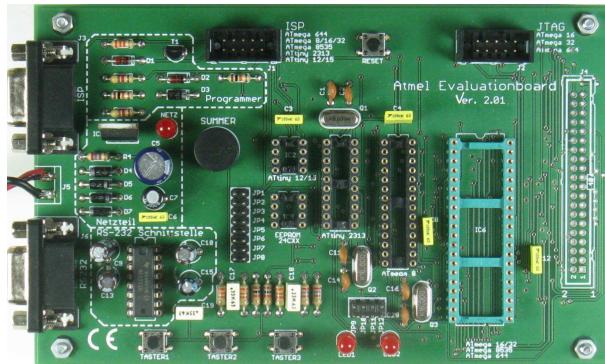


Abbildung 2: Altes Atmel-Evaluation-Board Quelle: <http://docplayer.org/>

Dies sind neben der einfachen Beschaffung, und des relativ moderaten Preises einige der Hauptgründe warum auch für diesen Kurs das Arduino-Framework ausgewählt wurde. Dies bezieht sich hierbei jedoch nur auf die Software und die Tools von Arduino. Die Hardwareseite, also die konkrete Elektronik und der Mikrocontroller selbst muss hier abgegrenzt betrachtet werden. Sie ist beim Arduino-Framework (Tools+Software) nicht zwangsläufig auf einen Mikrocontroller beschränkt. Im Arduino Uno kommt so z.B. ein 8-bit Mikrocontroller von AVR zum Einsatz. Im Arduino Due hingegen ein ARM basierter SAM90. Im Kurs soll wiederum ein Arduino-Kompatibler STM32F103 zum Einsatz kommen. Die Gründe für die Wahl eines Mikrocontrollers ohne Arduino-Logo werden im Kapitel Hardware näher beleuchtet. Auf die Arbeit mit anderen Mikrocontroller-Familien und Entwicklungsumgebungen wird darüber hinaus im Ausblick zu professionellen Arbeiten eingegangen. Spätestens hier wird die Wahl des STM32F103 ersichtlich werden.

3.2 Hardware

Arduino wurde ursprünglich für den Einsatz mit AVR-Mikrocontrollern von Atmel entwickelt. Diese hatten bereits vor Arduino eine sehr umfangreiche kostenlose Entwicklungsumgebung für die Programmierung mit Assembler und C. Atmel war einer der Vorreiter als er dies den Entwicklern anbot, in einer Zeit in der für Workstations und Programmiergeräte tausende Euro ausgegeben werden musste.

In den ersten Arduinos 2005 waren Atmega8 AVR-Mikrocontroller verbaut. Die nächsten Boards bauten auf den Weiterentwicklungen dieses Mikrocontrollers auf. Hier kamen mit Atmeg168, Atmega328 bishin zum Atmega2560 weitere Mikrocontroller der gleichen 8bit-Familie zum Einsatz. Diese sind bis heute die weit verbreitetsten Board der Arduino Familie. Doch was ist ein Mikrocontroller überhaupt? Ein Mikrocontroller ist ein Rechner, der durch eine Programmierung beliebige Berechnungen ausführen kann. Hier unterscheidet er sich nicht von einem herkömmlichen PC. Ebenso wie ein PC hat er sogenannte Pins (Schnittstellen) über die er mit der Außenwelt in Verbindung treten kann. Dabei zeichnen sich Mikrocontroller dadurch aus das besonders viele Schnittstellen zusammen mit der Recheneinheit untergebracht wird. Diese Schnittstellen und die Möglichkeit durch die Programmierung deren Eingaben mit Ausgaben machen die Mikrocontroller zu autonomen Geräten. Dadurch lassen sich komplexe Abläufe, Algorithmen und Regelungen mit geringen Aufwand und Kosten automatisieren. Das Verständnis der Schnittstellen, ihren Einsatz, der Konfiguration und Verwendung stellt dabei den Kern der Arbeit mit Mikrocontrollern dar. Im Kurs werden daher genau diese

Kenntnisse vermittel. Neben dem Erlernen der eigentlichen Programmierung werden wir die Schnittstellen durchgehen und lernen, wofür wir welche Einsetzen, wie wir sie Kombinieren und worauf wir achten müssen. Zunächst hier eine kleine Zusammenfassung der Schnittstellen und Teile des Mikrocontrollers.

Spannungsversorgung:**Mikroprozessor:****GPIOs:****USB:****UART:****Timer:****ADC:****SPI und I2C:****DAC:****Hardware im Kurs:**

Über die Jahre haben sich neben den 8bit-Mikrocontrollern auch mächtigere Mikrocontrollern mit neuen Schnittstellen wie USB oder Bluetooth zur Arduino-Hardware-Familie hinzugesellt. Sie können Alle mit der gleichen Arduino-Umgebung programmiert werden unterscheiden sich aber stark in ihrer Leistungsfähigkeit. So können die älteren Mikrocontroller in einem Befehl z.B. jediglich 8bit-Berechnungen durchführen (Zahlen von 0 bis 255). Größere Zahlen oder Multiplikationen müssen bei ihnen Aufwendig zerlegt werden und brauchen deshalb deutlich Länger. Neuere Mikrocontroller haben 32bit-Architekturen und können so Zahlen von 0 bis ca. 4 Milliarden in einem Befehl miteinander verrechnen. Auch Multiplikationen und Divisionen sind so deutlich schneller ausführbar. Zusätzlich können neuere Mikrocontroller mit deutlichen mehr Befehle pro Sekunde berechnen. Im Vergleich: Der Atmega Uno kann 16 Millionen Berechnungen pro Sekunde ausführen, der Arduino Due hingegen satte 72 Millionen.

Für unseren Kurs wird daher ebenfalls ein Mikrocontroller der neuen Generation eingesetzt. Wegen des Preises und der Möglichkeit komplexere Aufgaben bewältigen zu können wird allerdings keine Arduino-Hardware eingesetzt. Wir setzen im Kurs natürlich die Arduino-Umgebung mit allen Werkzeugen und Möglichkeiten ein.

Konkret werden wir den 32bit-Mikrocontroller STM32F103 der europäischen Firma STMicroelectronics verwenden. Diese bieten mit ihrem Nucleo-Board Kompatible Arduino-Geräte für das Einsteigersegment ein. Dieser ist Vergleichbar mit dem dem Arduino Due aber deutlich einfacher beschaffbar. Zum Vergleich hier eine kurze Tabellarische Zusammenfassung der Mikrocontroller.

3.3 Software

Kommen wir nun zur Arduino Umgebung. Die sogenannten Arduino IDE (Integrated development environment / Integrierte Entwicklungsumgebung) unterstützt uns bei der Programmierung. Es übernimmt im

Tabelle 1: Vergleich zwischen Arduino Uno, STM32F103 und Arduino Due

Vergleich	Arduino Uno	STM32F103	Arduino Due
Abgearbeitete Befehle pro Sekunde	16 Mio.	72 Mio.	72/84 Mio.
GPIO-Pins	14	Nucelo: 51, Blue-Pill: 20	54
Architektur	8bit (Zahlen von 0-255)	32bit (Zahlen von 0-4 Mrd)	32bit (Zahlen von 0-4Mrd)
Spannungslevel	5V	3,3V	3,3V
USB	Kein echtes USB. nur über UART	Ja	Ja
Timer	3	7	9
ADCs	6 Kanäle, 1024 Stufen, 10 Tausend Messwerte pro Sekunde	max. 16 Kanäle, 4096 Stufen, 500 Tausend Messwerte pro Sekunde	12 Kanäle, 1024/4096 Stufen, 25 Tausend Messwerte pro Sekunde
PWM	3 Kanäle	Max. 12 Kanäle	Max. 12 Kanäle
DAC	Nein	Nein	Ja
Preis	Reichelt: 20€, Ebay: 3,5€	Digikey (Nucleo): 12€, Ebay (Blue Pill): 2,5€	Reichelt 35€, Ebay 17€

Hintergrund die nötige Verwaltung und Kommandos zur Übersetzung der Programmiersprache in die Maschinensprache. Dieser Vorgang („Kompilierung“) besteht aus vielen Kommandozeilen-Befehlen, Programmen und Skripten die im Hintergrund ablaufen. Der übersetzte Maschiencode kann anschließend mit der IDE in den entsprechenden Arduino geladen werden („Flashen“). Zusätzlich bietet die Software zahlreiche Werkzeuge zum vereinfachen des Programmierprozesses auf die wir später bei ihrer Benutzung eingehen werden.

Im Anschluss öffnet bitte die Arduino IDE und geht auf "Werkzeuge -> Board -> Boardverwalter" (bzw. in Englisch "Tools -> Board ... -> Boardmanager"). Hier installiert ihr das Paket "Arduino SAMD Boards (32-bits ARM Cortex-M0+) by Arduino" installieren. Nach anklicken erscheint hier die Schaltfläche zur Installation. Voreingestellt sollte die neueste Version 1.6.6 installiert werden. Hier werden im Hintergrund die richtigen Übersetzer für die Mikrocontroller nachgeladen.

Da wir im Kurs das Nucleo-Board von STM anstelle der klassischen Arduino-Hardware einsetzen werden, müssen wir nun noch das Addon STM32Duino für die Arduino-IDE installieren. Dieses Plugin erweitert die Umgebung, sodass diese Mikrocontroller/Boards ebenfalls angesprochen und programmiert werden können. Hierzu muss folgende gepackte Datei von <https://github.com/rogerclarkmelbourne/Arduino-STM32/archive/master.zip> heruntergeladen werden. Anschließend ist der Inhalt je nach Betriebssystem in einen der folgenden Ordner zu entpacken:

- Unter Windows: „Dokumente/Arduino/hardware“ (englische Systeme „My Documents/Arduino/hardware“)
- Unter MacOS: ~/Documents/Arduino/hardware/

- Unter Linux: In den „hardware“-Ordner deiner Arduino-Installation („Sketches“-Ordner). Führe anschließend das Skript „tools/linux/install.sh“ aus

Für MacOS und Linux sind zusätzlich noch weitere Schritte notwendig die ihr unter <https://github.com/rogerclarkm> nachlesen könnt.

Nach erfolgreicher Installation der IDE und der Erweiterung sollte die Arduino IDE nach dem Start wie folgt (3) aussehen. Im Menüpunkt „Werkzeuge->Board ... ->“ muss das „STM Nucleo F103RB“ aufgelistet werden. Anderfalls sollten bitte die Installation der STM32Duino-Erweiterung überprüft oder kontaktiert ggf. den Kursleiter.

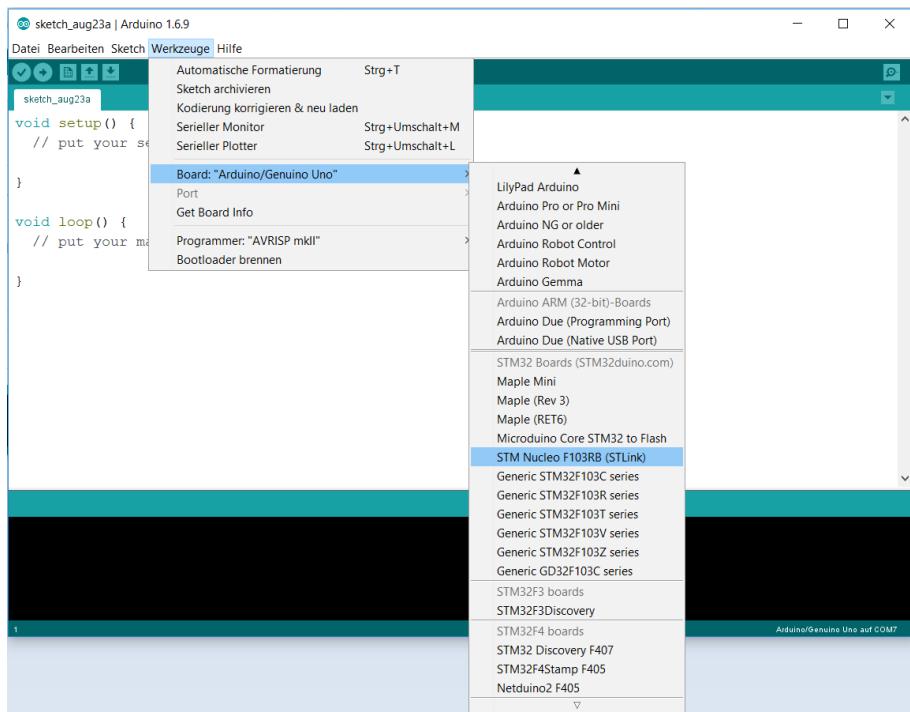


Abbildung 3: Arduino Uno

Die Arduino IDE ist in 4 Bereiche eingeteilt. Von oben nach unten:

- Das Menü mit allen generellen Funktionen zum speichern, verwalten und programmieren der Projekte (sogenannte „Sketche“). Dazu Werkzeuge mit für die leichtere Arbeit und Einstellungsmöglichkeiten zu Boards.
- Die Toolbar, welche die wichtigsten Funktionen zum Sketch enthält (Überprüfen, Hochladen, Speichern etc.)
- Der Quellcodeeditor (Programmm) augeteilt in Tabs für die Arbeit in mehreren Dateien.
- Die Status-Konsole als Anzeige der aktuellen Ereignisse und Ergebnisse.

Ein neues Programm („Sketch“) wird über Datei -> Neu erstellt oder ein bestehendes über Datei -> Öffnen geladen. Anschließend muss unter Werkzeuge -> Board ... das richtige Board ausgewählt werden (hier im Kurs „STM Nucleo F103RB“). Für den Einstieg bringt die Arduino einige Beispiele mit. Für die Arbeit mit dem STM Nucleo findet man nach Installation der STM32Duino-Erweiterung und Auswahl des richtigen Board ein eigener Bereich („Datei -> Beispiele -> A_STM32_Examples“). Für einen ersten Test mit Hardware lässt sich folgende Arbeitsabfolge zusammenfassen:

- Öffnen der Arduino IDE
- Einstellung des Board „Werkzeuge -> Board ...“ auf „STM Nucleo F103RB“
- Öffnen eines Beispielprogramms „Datei -> Beispiele -> A_STM32_Examples ->“

3.4 Programmierung

Absatz Programmierung für Mikrocontroller

Absatz Möglichkeiten Programmierung

- Blockly
- Vereinfachte Form der C (keine Header)
- C Programmierung

Programme heißen sketches

- Absatz C Programmierung
- C Programmierung im Kurs
 - Verweis auf Anhang

3.5 Simulation

Für den Einstieg in die Arbeit mit Mikrocontroller bietet das Arduino-Universum die bislang benutzerfreundlichsten Werkzeuge. Hierunter zählen auch ein sehr umfangreiches Simulationswerkzeug CircuitIO. Dies wird von Autodesk herausgegeben und ordnet sich in das Autodek-Entwickler-Universum ein. Die Nutzung ist kostenfrei, erfordert jedoch eine Registrierung beim Anbieter. Der Simulator kann nur mit Internetverbindung verwendet werden. Für den Kurs wird besonders für die Vorführung und Erklärung der Funktionsweise von Programmen und Schaltungen CircuitIO eingesetzt. Auch während der eigenen Arbeit lohnt sich die Verwendung. So können Fehler bereits vorher erkannt, Hardwareaufbauten korrigiert und komplexe Programme auseinandergenommen werden. Besonders die bereitgestellten Beispiele machen den Simulator zu einem einsteigerfreundlich und mächtigen Helfer. Einen guten Ausgangspunkt zum Erlernen der Arbeitsweise und Komponenten von CircuitIO bilden die Beispiele und Videos. Diese lassen sich meist auf eigenen Projekten anpassen und erklären die von vornherein intuitive Software. Ein guter Einstieg mit Einsteiger-Videos lässt sich unter <https://circuits.io/home/learn> finden.



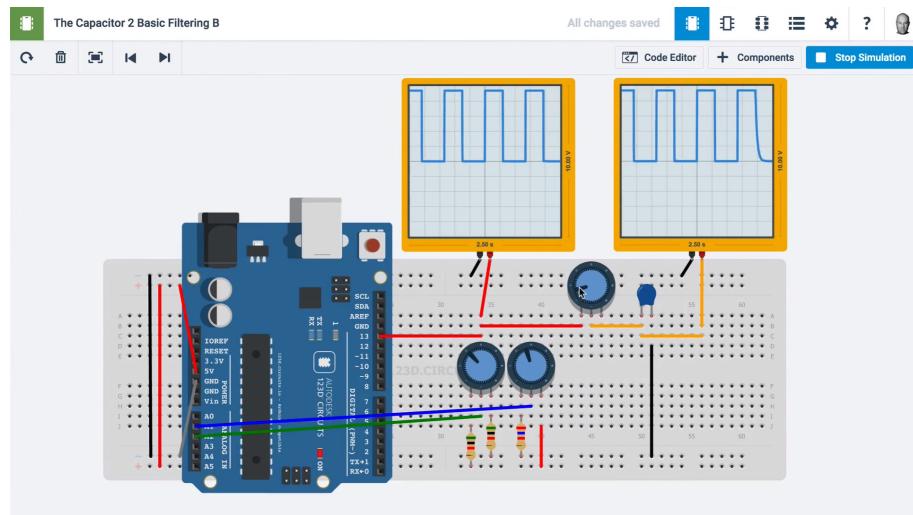


Abbildung 4: CircuitIO mit Arduino Uno - Beispiel

4 Workshop Abend 1

4.1 Unser erstes Programm, Blink!

Das Blink Programm ist das berühmte Einsteiger-Programm. Wir lassen unsere erste LED blinken.

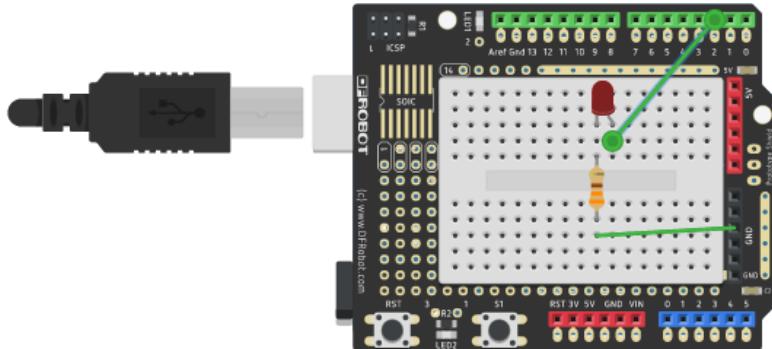


Abbildung 5: Schaltung 1 für den Aufbau von Blink!

Die Schaltung lässt sich, wie die anderen Schaltungen unter <https://circuits.io/circuits/2734640-arduino-im-konglomerat-tag-1-blink> finden und testen. Unser erster Quellcode sieht dabei wie folgt aus und ist unter https://github.com/mdrobisch/arduino_kurs zu finden.

```
1 /*
```

```

2  * Unser erstes Arduino Programm, die blinkende LED.
3  */
4
5 // Die setup-Funktion wird einmalig nach "reset" oder power-up" ausgeführt
6 void setup()
7 {
8     // Initialisiert den Pin 2 als Ausgang, der Ausgang steuert die LED
9     pinMode(2, OUTPUT);
10}
11
12 // Die loop-Funktion wird ständig wiederholt durchlaufen
13 void loop()
14 {
15     digitalWrite(2, HIGH); // Schaltet die LED ein (HIGH ist der Schaltzustand/
16     // Spannungslevel)
17     delay(1000);           // Warte (delay) 1000ms = 1 Sekunde
18     digitalWrite(2, LOW); // Schaltet die LED aus (LOW Schaltzustand/Spannungslevel)
19     delay(1000);           // Warte 1 Sekunde
}

```

Listing 1: Blink (Quellcode unter www.github.de/arduino)

Funktionsweise: Das Programm startet mit dem Ausführen der `setup` Funktion. Hier initialisieren wir den GPIO 2 als Ausgang. Anschließend führt der Mikrocontroller die Funktion „`loop`“ als Schleife wiederholend aus. Hier wird

Wissen Arduino: Pins und digitale Ausgänge:

- Die Steuerung des Mikrocontrollers erfolgt über Funktionen, welche die Register des Mikrocontrollers entsprechend steuern (Register setzen)
- Hauptbestandteil eines Programmes ist die Ansteuerung von Pins des Mikrocontrollers, diese können eine der folgende Funktion einnehmen
 - Eingänge (Input Pins) zur Detektion von Spannungsleveln (0V-0,8 LOW, 1,6-3,3V HIGH)
 - Ausgänge (Output Pins) zur Ansteuerung von externen Bauelementen (0V LOW, 3,3V HIGH)
 - Spezialfunktionen wie Spannungswandler, Pulssweitenmodulation, Kommunikation (wird in den nächsten Abschnitten detailliert beschrieben)
- Die Funktion oder Modus eines Pins muss vor der Funktion umgeschalten werden. Für die Verwendung als Ausgangs-Pin dient die Funktion `pinMode(Pinname [PB1], Modus [INPUT])`
- Der Spannungspegel/level Ausgänge wird mit der `digitalWrite`-Funktion gesetzt. `digitalWrite(Pinname [0], Level [LOW / HIGH])` .

Hinweis: Die Namenbezeichnungen/nummern der jeweiligen Pins sind im Angehang in einer Übersicht aufgeführt. Pin 0 und Pin 1 sind auf dem Nucleo-Board nicht verfügbar, da sie für Kommunikationszwecke vorbelegt sind.

Wissen C-Programmierung: Aufbau, Funktionen und Kommentare:

- Funktionen haben in C folgende Struktur

```

1 // Funktion ohne Übergabewerte (Parameter) und ohne Rückgabewert (void)
2 void funktionsname()
3 {
4
5 }
6
7 // Funktion mit Übergabewerte (Parameter) und ohne Rückgabewert (void)
8 void funktionsname_mit_parametern(int parameter)
9 {
10    // Parameter int ist eine Variable die ganzzahlige Werte speichert
11 }
12
13 // Funktion ohne Übergabewerte (Parameter)
14 int funktionsname_mit_parametern()
15 {
16    // als int Rückgabewert muss ein ganzzahliger Wert zurückgegeben werden
17    return 42;
18 }
```

Listing 2: Blink (Quellcode unter www.github.de/arduino)

- Funktionsnamen dürfen nicht mit Zahlen anfangen und keine Leerzeichen enthalten (weitere Namensregeln im Anhang: C Programmierung)
- Funktionen werden mit einem { und einem } vom restlichen Programm abgegrenzt
- Funktionen können nach ihrer Definition wie folgt aufgerufen werden

```

1 // Aufruf einer Funktion ohne Übergabewert, z.B. innerhalb der main-Funktion
2 void main()
3 {
4     // Funktionsaufruf einer Funktion ohne Parameter
5     funktionsname();           // Anweisungen werden in C mit einem Semikolon
6     // abgeschlossen
7
8     // Funktionsaufruf einer Funktion mit konstanten Parameter
9     funktioname(1,2,3);
10
11    // Funktionsaufruf einer Funktion mit konstanten und variablen Parameter
12    funktioname(1,variabelnamem,3);          // die Verwendung von Variablen
13    // werden in den nächsten Abschnitten erläutert
14 }
```

Listing 3: Blink (Quellcode unter www.github.de/arduino)

- Wie jede Anweisung in C wird auch ein Funktionsaufruf mit einem Semikolon abgeschlossen.
- Einzelne Kommentare werden mit // eingeleitet, ab dieser Stelle wird die komplette Zeile zum Kommentar (für den Programmablauf ignoriert)

- Mehrzeilige Kommentare werden mit einem /* eingeleitet. Das Kommentar gilt dann bis zum abschließenden */
- Die hier eingeführte Verwendung von Variablen (und Parametern in Funktionen) wird in den nächsten Abschnitt erläutert

4.2 Überwachen von Tastern

Nachdem wir nun Ausgänge beliebig schalten können werden wir uns an den Eingängen versuchen. Mit diesen können Ereignisse wargenommen werden. Diese Ereignisse treten als Spannungspegeln an den jeweiligen Pins auf. Es können die Pegeln LOW und HIGH unterschieden und erkannt werden. Mit folgendem Programm werden wir z.B den Zustand des Tasters anhand des Pegels an Pin 12 überwachen. Entsprechend der Stellung wird die LED blinken oder erlöschen.

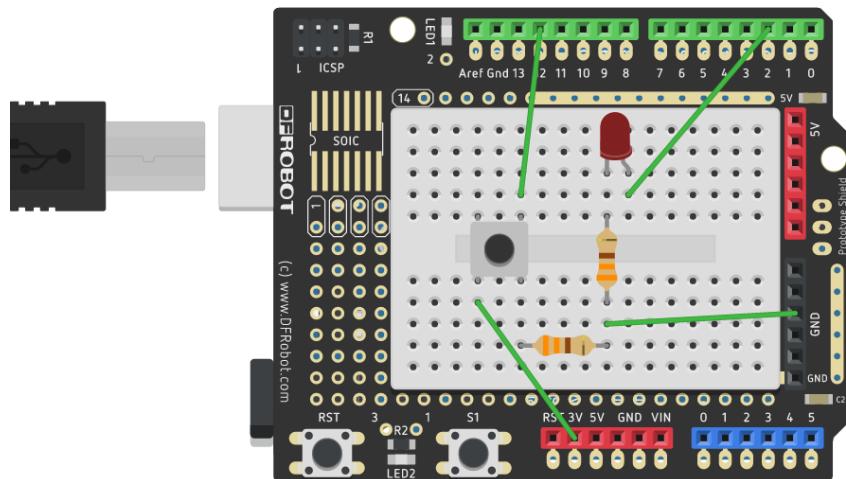


Abbildung 6: Schaltung 2 für das Überwachen von Tastern

```

1  /*
2   * Die LED blinkt, wenn der Taster gedrückt ist
3   */
4
5 // Die setup -Funktion wird einmalig nach "reset" oder power -up" ausgeführt
6 void setup()
7 {
8     // Initialisiert den Pin2 als Ausgang, der Asugang steuert die LED
9     pinMode(2, OUTPUT);
10    pinMode(12, INPUT);
11
12 }
13
14 // Die loop -Funktion wird ständig wiederholt durchlaufen

```

```

15 void loop()
16 {
17   if (digitalRead(12) == HIGH) {
18     digitalWrite(2, HIGH); // Schaltet die LED ein (HIGH ist der Schaltzustand/
19                           // Spannungslevel)
20     delay(500);          // Warte (delay) 1000ms = 1 Sekunde
21     digitalWrite(2, LOW); // Schaltet die LED aus (LOW Schaltzustand/Spannungslevel)
22     delay(500);          // Warte 1 Sekunde
23   } else {
24     digitalWrite(2, LOW); // Schaltet die LED aus (LOW Schaltzustand/Spannungslevel)
25   }
}

```

Listing 4: Taster (Quellcode unter www.github.de/arduino)

Wissen Arduino: Digitale Eingänge:

- Zur Verwendung eines Pins als Eingang muss dieser mit **pinMode(Pinname [PB1], Modus [INPUT])** gesetzt werden
- Eingänge erkennen von 0 - 0.8 V ein logisches LOW. Ab 1.6-3.3V ein logisches HIGH (der Spannungsbereich dazwischen ist nicht definiert, die Erkennung ist hier nicht vorhersehbar)
- Der erkannte Zustand eines Eingangs kann mit einem **digitalRead(Pinname [PB2])** ausgelesen werden. Diese Funktion gibt entsprechend den Wert HIGH oder LOW zurück.

Wissen C-Programmierung Variablen und bedingte Ausführung (if-Anweisungen):

- Funktionen haben in C folgende Struktur

Wissen Arduino: Zeitfunktionen

- Zum erfassen der Zeit und zum gezielten Warten stellt das Arduino-Framework verschiedenen Funktionen zur Verfügung
- Mit der häufigste Verwendung in vielen Programmen findet man die **delay(millisekunden)** Funktion. Das Programm pausiert für die übergebenen Anzahl von Millisekunden.
- Die Gegenstücke zu **delay** sind die Funktionen **millis()** und **micros()**. Sie dienen der Erfassung der vergangenen Zeit. Die Funktion **millis()** gibt die vergangenen Millisekunden seit Programmstart wieder. Die **micros**-Funktion die vergangenen Mikrosekunden, Hierbei ist zu beachten, dass es jeweil einen Überlauf der Zahl nach 50 Tagen bzw. 70 Minuten gibt. Der zurückgegebene Wert beginnt dann von 0.

4.3 Mit dem Timer zum Lauflicht

Zum Abschluss des Tages werden wir unsere erste Bibliothek kennenlernen. Bibliotheken sind Programmteile, die bestimmte wiederkehrende Aufgaben vereinfachen sollen. Meist geschieht dies durch bereitgestellten Funktionen oder globale Variablen. Wir werden nun die Timer Bibliothek kennenlernen. Diesen werden wir verwenden um ein Lauflicht zu steuern. Ein Timer oder auch Counter ist ein Modul des Mikrocontrollers, welches die Takte des Taktgebers (Quarz oder oscillator) zählen kann. Dieser lässt sich so einstellen, dass nach

erreichen eines vorgegebenen Schwellwertes eine Funktion ausgeführt werden. Damit bietet sich die Möglichkeit Zeit unabhängig vom jeweiligen Programm zu messen oder Funktionen nach einer festen vorgegebener auszuführen. Mikrocontroller haben meist mehrere dieser Timermodule. Sie sind ein elementarer Bestandteil der Arbeit mit Mikrocontrollern. Timer werden mit Spezialregistern gesteuert, die in den Datenblättern der Mikrocontroller beschrieben sind. Für einen Arduino mit Atmega32 (Uno) sieht die Einrichtung des Timers z.B. wie folgt aus.

```

1  /* Konfigurieren des Timer1 mit Spezialregistern */
2  /* Timer/Counter1 zurücksetzen */
3  TCCR1A = 0;
4  TCCR1B = 0;
5  TIMSK1 = 0;
6
7  /* Einrichten des Timer/Counter1 */
8  TCCR1B |= (1 << CS11) | (1 << CS10); // Prescalar/Vorteiler = 64
9  TCNT1 = 10000; // Zählermarke setzen (Angabe in Takt/Zeit)
10 TIMSK1 = (1 << TOIE1); // aktivieren des Timers

```

Listing 5: Blink (Quellcode unter www.github.de/arduino)

Das Arduino-Framework bietet hierfür keine mitgelieferten Funktionen. Auch unterscheiden sich die Spezialregister und die Funktionsweise sehr stark zwischen den eingesetzten Boards (z.B. Arduino Uno, Mikro Due). So würde selbe Code beim Einsatz des STM32F103, der im Kurs eingesetzt wird, nicht kompiliert werden können. Wegen der Mikrocontrollerabhängigkeit und des unübersichtlichen Arbeiten mit Spezialregistern, haben sich deshalb für Hardwaremodule (Timer, Schnittstellen, Analog-Digital-Wandler) Bibliotheken durchgesetzt. Sie stellen verständlichere Funktionen zur Verfügung. Für den Arduino Uno ist z.B. die TimerOne-Bibliothek sehr verbreitet. Für diesen Fall sieht der Quelltext zum initialisieren desselben Timers wie folgt aus.

```

1  /* Konfigurieren des Timer1 mit Spezialregistern */
2  Timer1.initialize(150000); // Zählermarke (Angabe in Mikrosekunden)
3  Timer1.attachInterrupt(blinkLED);

```

Listing 6: Blink (Quellcode unter www.github.de/arduino)

Zunächst unsere Schaltung. Wir können die Schaltung aus dem vorherigen Projekt übernehmen.

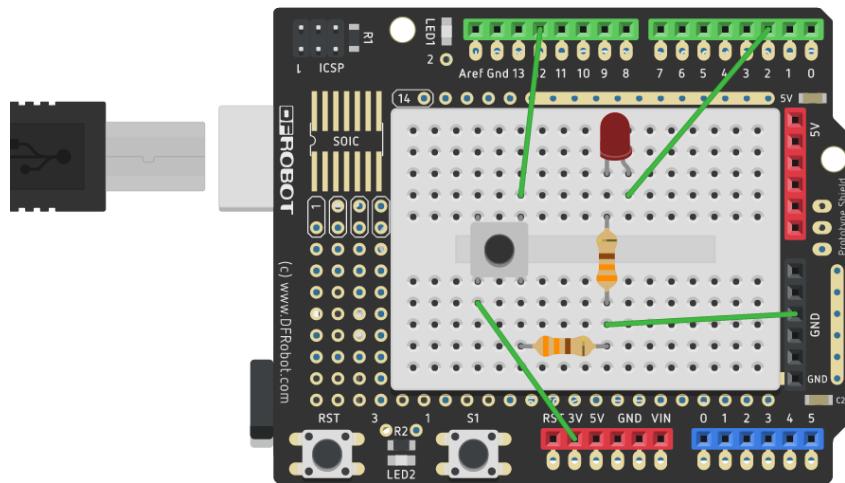


Abbildung 7: Schaltung 2 für das Arbeiten mit Timern

Hier unterscheidet sich der Quellcode für den Einsatz eines Arduino-Boards noch einmal von dem des eingesetzten STM32Duino-Bibliotheken. Eine sehr umfangreiche Dokumentation der bereitgestellten STM32Duino-Bibliotheken findet sich unter <http://docs.leaflabs.com/docs.leaflabs.com/index.html>. Besonders der Bereich der **Hardware Peripherals** ist ein guter Anlaufpunkt. Wir werden uns im letzten Beispiel-Programm des Abends auf die Timer-Bibliothek konzentrieren. Dabei beschränkt sich das Kennenlernen auf die wesentlichen Funktionen und Aufrufe ohne tiefgründig die Funktionsweise zu hinterfragen. In den nächsten Teilen des Workshops werden wir dann gezielter auf die Hintergründe eingehen.

```
1  /*
2   * LED mit Timer
3   */
4
5 void timer_funktion(void); // die Timer-Funktion wird vom Timer-Modul aufgerufen, wenn
6   // der Schwellwert des Zählers erreicht ist, hier zunächst nur die Definition
7
8 int led_zustand = 0; // der Zustand der LED, 0 = aus, 1 = an
9
10
11 void init_timer1(unsigned long period_in_us)
12 {
13   Timer1.setChannel1Mode(TIMER_OUTPUTCOMPARE);
14   Timer1.setPeriod(period_in_us); // Periodendauer in Mikrosekunden
15   Timer1.setCompare1(1);
16   Timer1.attachCompare1Interrupt(timer_funktion);
17   Timer1.resume();
18 }
19
20 // Die setup-Funktion wird einmalig nach "reset" oder power-up" ausgeführt
21 void setup()
22 {
23   // Initialisiert den Pin 2 als Ausgang, der Ausgang steuert die LED
24   pinMode(2, OUTPUT);
```

```

22 // Konfiguration des Timers
23 // der Quellcode sieht in der Simulation anders aus, da wir das STM-Board verwenden
24 init_timer1(1000000);
25 }
26
27
28 // Die loop-Funktion wird ständig wiederholt durchlaufen
29 void loop()
30 {
31 // In diesem Beispiel kann die Schleifenfunktion leer bleiben, da alles in der Timer-
32 // Funktion abgearbeitet wird
33 }
34
35 void timer_funktion(void)
36 {
37 if(led_zustand == 0)
38 {
39     digitalWrite(2, HIGH); // Schaltet die LED ein (HIGH ist der Schaltzustand/
40     // Spannungsevel)
41     led_zustand = 1;
42 }
43 else
44 {
45     digitalWrite(2, LOW); // Schaltet die LED aus (LOW Schaltzustand/Spannungslevel)
46     led_zustand = 0;
47 }
}

```

Listing 7: LED-Timer (Quellcode unter www.github.de/arduino)

HINWEIS: Der abgebildetet Quellcode liegt im Quellcode-Verezeichnis in zwei Varianten vor. Variante A ist der Quellcode für den Einsatz mit STM32Duino-Boards. In Variante B wird zwischen dem Einsatz der STM32Duino und der Arduino-Uno Boards unterschieden. Dies wird benötigt um Die Funktionsweise im Simulator testen zu können, welcher nicht für den Einsatz mit STM32-Boards konzipiert ist.

Übung:

Für Einsteiger: Erweitere das Programm so, dass die LED nur blinkt, wenn der Taster geschlossen ist.
 Für Profis: Erweitere das Programm so, dass auch hier SOS nach Knopfdruck ausgeben wird

Hausaufgabe (freiwillig)

Für Einsteiger:

5 Workshop Abend 2

5.1 Hausaufgabe: Lauflicht

5.2 Ereignisse Zählen per Polling

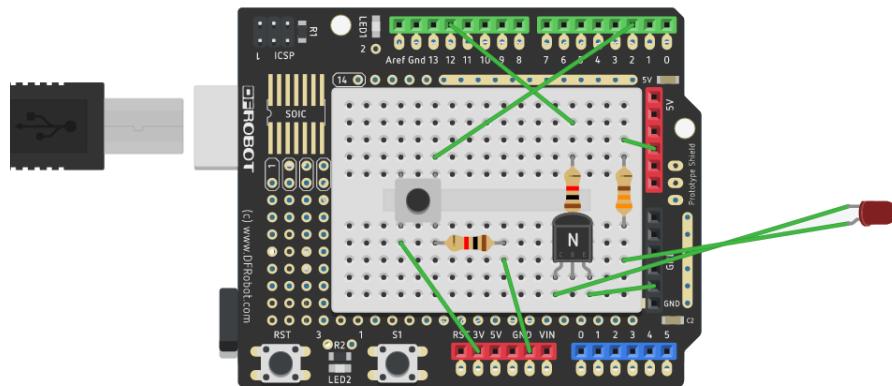


Abbildung 8: Schaltung 3 für das Zählen mit Polling

```

1  /*
2   * Die LED blinkt mit der eingestellten Helligkeit
3   */
4
5 // Variable zur Speicherung des Pin Zustandes
6 int buttonState = 0;           // Variable zum auslesen des Taster-Zustands
7 int buttonOldState = 0;
8 int helligkeit = 1;
9
10 // Die setup-Funktion wird einmalig nach "reset" oder power-up" ausgeführt
11 void setup()
12 {
13     // Initialisiert den Pin 12 als Ausgang , der Ausgang steuert die LED
14     pinMode(12, OUTPUT);
15     // Initialisiere den Pin 2 als Eingang , der den Zustand des Tasters überwacht
16     pinMode(2, INPUT);
17
18 }
19
20 // Die loop-Funktion wird ständig wiederholt durchlaufen
21 void loop()
22 {
23     // Auslesen des Zustandes des Tasters in jedem Zyklus wird Polling genannt
24     buttonState = digitalRead(2);
25
26     if(helligkeit > 0)          // Schaltet die LED nur ein, wenn die helligkeit > 0 ist

```

```

27     digitalWrite(12, HIGH); // Schaltet die LED ein (HIGH ist der Schaltzustand/
28     // Spannungslevel)
29     delay(helligkeit*1);      //
30     digitalWrite(12, LOW);    // Schaltet die LED aus (LOW Schaltzustand/Spannungslevel)
31     delay(20);              // Warte 1 Millisekunde
32
33     if(buttonOldState == LOW) // ein verändern der Helligkeit ist nur gewünscht, wenn
34     // der Zustand des Tasters vorher LOW war
35     {
36         if (buttonState == HIGH) { // wenn der Taster gedrückt wird erhöhen wir anschließend die Helligkeit
37             helligkeit++;
38
39             if(helligkeit > 3) // ist die Helligkeit bereits auf Maximum (5) stezen wir
34             // die Helligkeit auf 0
40             helligkeit = 0;
41     }
42 }
43 buttonOldState = buttonState; // der "alte" Zustand ist nun der neue Zustand
44 }
```

Listing 8: Polling (Quellcode unter www.github.de/arduino)

Die Simulationen sind unter <https://circuits.io/users/763753> zu finden. Die Quellcodes unter https://github.com/mdrobisch/arduino_kurs.

5.3 Transistoren

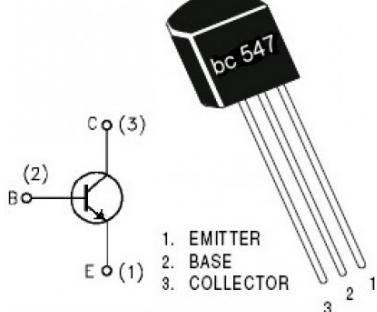


Abbildung 9: Schaltsymbol mit Pinbelegung des BC547-NPN-Transistor (Quelle: [www.http://potentiallabs.com/](http://potentiallabs.com/))

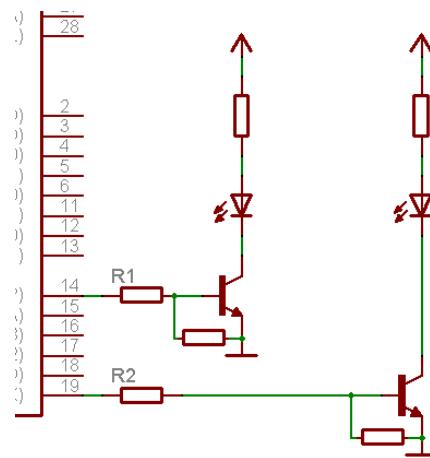


Abbildung 10: Schaltung mit NPN-Transistor (Quelle: www.mikrocontroller.net)

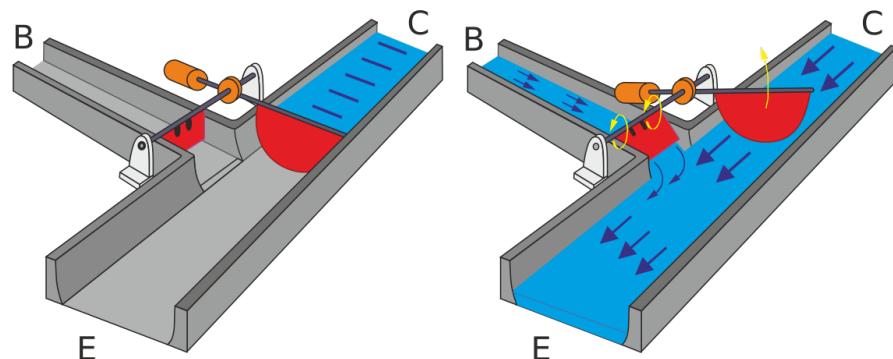


Abbildung 11: Funktion des Transistors, gesperrt und offen (Quelle: <http://www.leiphysik.de/>)

Andere Transistortypen und elektronische Schalter:

Die oben beschriebenen Transistoren sind sogenannte Bipolartransistoren. Darüberhinaus sind weitere Transistortypen und elektronische Schalter sehr verbreitet.

- Mosfet Transistoren: höhere Leistungen
- Thyristoren: elektronische Schalter für Netzspannungen (nicht für Gleichspannung geeignet)
- Relais: Universale Schalter für Netz- oder Gleichspannung und sehr hohe Leistungen (meist nur geringe Schaltfrequenzen mit Prelliegenschaft)

5.4 Ereignisse Zählen per Interrupt

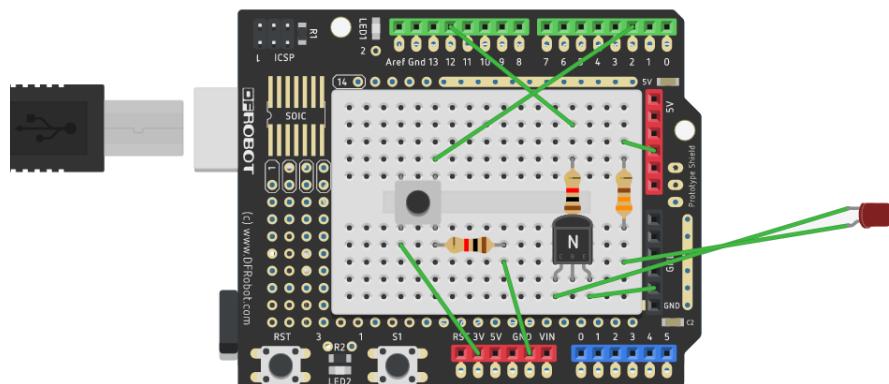


Abbildung 12: Schaltung 3 für das Zählen mit Interrupt

```

1  /*
2   * Extra Schaltung: Interrupt mit Pulldown
3   */
4  int helligkeit = 1;
5
6  void interruptFunction()
7 {
8     helligkeit++;
9
10    if(helligkeit > 3) // ist die Helligkeit bereits auf Maximum (5) stezen wir die
11        Helligkeit auf 0
12    helligkeit = 0;
13}
14
15 // Die setup-Funktion wird einmalig nach "reset" oder power-up" ausgeführt
16 void setup()
17 {
18     // Initialisiert den Pin 12 als Ausgang, der Asugang steuert die LED
19     pinMode(12, OUTPUT);
20     // Initialisiert den Pin 2 als Ausgang, der den Status des Tasters überwacht
21     pinMode(2, INPUT);
22     // Zum Verknüpfen der Interrupts wird die attachInterrupt Funktion verwendet
23     // Dabei ist der erste Parameter die Interrupt-Nummer
24     // Für STM32Duino gilt: Die Interrupt-Nummer ist gleich der Pinnummer
25     // Der Parameter "RISING" bezieht sich auf die Art des Interrupts.
26     // Mit RISING oder FALLING wird die Änderung des Signals von LOW<->HIGH Pegel
27     angegeben
28     // Mit CHANGE wird auf beide Ereignisse ein Interrupt ausgelöst.
29     attachInterrupt(2, interruptFunction, RISING);
30 }
31
32 // Die loop-Funktion wird ständig wiederholt durchlaufen
33 void loop()
34 {
35     if(helligkeit > 0) // Schaltet die LED nur ein, wenn die helligkeit > 0 ist
36     {
37         digitalWrite(12, HIGH); // Schaltet die LED ein (HIGH ist der Schaltzustand/
38             Spannungsevel)
39     }
40     delay(helligkeit*1); // Warte (delay) 1000ms = 1 Sekunde
41     digitalWrite(12, LOW); // Schaltet die LED aus (LOW Schaltzustand/Spannungslevel)
42     delay(20); // Warte 20 Milli-Sekunde
43 }
```

Listing 9: Interrupt (Quellcode unter www.github.de/arduino)

5.5 Ereignisse entprellen

Besonders bei Ereignissen, die durch mechanische Taster oder Schalter ausgelösst werden kann es zu einem Schwingverhalten kommen. Da der Mikrocontroller sehr schnell auf Ereignisse reagieren kann (Millionstel einer Sekunden) reagiert er auch auf dieses Schwingverhalten. Ein Tastendruck wird so z.B. oft mehrmals

erkannt und lsst hintereinander mehrere Ereignisse aus. Dieser Effekt wird Prellen genannt. Um dies zu verhindern werden zustzlich Mechanismen eingebaut, um dies zu verhindern. Das sogenannte „Entprellen“. In dem Arduino-Beispiel „EreignisInterruotErweitert“ wird dies fr den angeschlossenen Taster (gleicher Schaltungsaufbau) realisiert.

Dies wird hier (in einer einfachen Realisierung) ermöglicht, indem das Ereignis nach dem erstmaligen auftreten zunächst eine definierte Zeit nicht erneut ausgelöst werden kann.

5.6 Kommunikation mit dem PC, Hallo Welt!

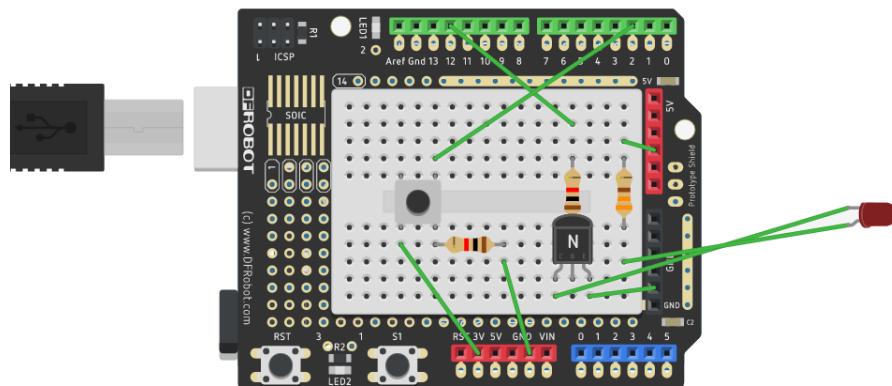


Abbildung 13: Schaltung 4 für unser „Hallo Welt“

```
1  /*
2   * Extra Schaltung: Interrupt mit Pulldown
3   */
4  int tasterEreigniss = 0;
5
6  void interruptFunction()
7  {
8      tasterEreigniss = 1;
9  }
10
11 // Die setup-Funktion wird einmalig nach "reset" oder power-up" ausgeführt
12 void setup()
13 {
14     // Initialisiert den Pin 2 als Ausgang, der den Status des Tasters überwacht
15     pinMode(2, INPUT_PULLDOWN);
16     // Zum Verknüpfen der Interrupts wird die attachInterrupt Funktion verwendet
17     // Dabei ist der erste Parameter die Interrupt-Nummer
18     // Für STM32Duino gilt: Die Interrupt-Nummer ist gleich der Pinnummer
19     // Der Parameter "RISING" bezieht sich auf die Art des Interrupts.
20     // Mit RISING oder FALLING wird die Änderung des Signals von LOW<->HIGH Pegel
21     angegeben
```

```
21 // Mit CHANGE wird auf beide Ereignisse ein Interrupt ausgelösst.  
22 attachInterrupt(2, interruptFunction, RISING);  
23  
24 // Init .....  
25 Serial1.begin(115200);  
26 delay(500); // Kurz warten damit die IDE die Verbindung aufbauen kann  
27 Serial1.println("Hallo Welt!");  
28 }  
29  
30 // Die loop-Funktion wird ständig wiederholt durchlaufen  
31 void loop()  
32 {  
33     if(tasterEreigniss == 1)  
34     {  
35         Serial1.println("Taste gedrueckt");  
36         delay(200); // warten zum entprellen  
37         tasterEreigniss = 0;  
38     }  
39 }
```

Listing 10: Hallo Welt (Quellcode unter [www.github.de/arduino](http://www.github.com/arduino))

5.7 Kontrollstrukturen in C

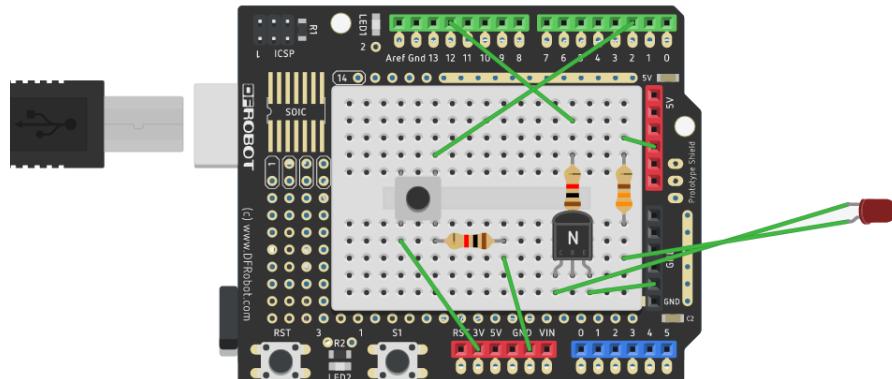


Abbildung 14: Schaltung 3 Kontrollstrukturen in C

Im Kurs wird nur wenig auf die Programmiersprache C eingegangen. Die meisten Anfänger ziehen (meines Kenntnisstandes) das Erlernen und die Eingewöhnung an Beispielen, den theoretischen Diskurs vor. In diesen Abschnitt sollen dennoch die Grundlagen der Programmiersprache vermittelt werden. Da dieser Teil nicht Bestandteil des Abendkurses ist empfiehlt sich für Interessierte eine Lektüre als Hausaufgabe oder nach Abschluss des Kurses. Die Erklärungen finden auch in diesem Fall an konkreten Beispielen statt. Die Simulationen sind unter <https://circuits.io/users/763753> zu finden. Die Quellcodes unter

https://github.com/mdrobisch/arduino_kurs. Im Internet kursieren darüber hinaus weitere sehr zu empfehlenswerte Tutorials. Zum Teil speziell für Mikrocontroller. Folgende sollen hier kurz genannt werden:

- www.mikrocontroller.net

Mathematische Ausdrücke:

- Zuweisungen: =
- Basisoperationen: +, -, *, /
- Erweiterte Zuweisung: +=, -=, *=, /=
- Dekrement / Inkrementieren
- Weitere Operationen: %

Boolsche (logische Ausdrücke):

- AND - &&
- OR - ||

Schleifen:

- while
- for

Fallunterscheidungen:

- switch
- if

Besondere Ausdrücke:

- Löschen und Setzen von Bits mit
- Bitshift
- Abfragen von Bits
- Bedingte Zuweisung

Hausaufgabe (freiwillig)

Ansteuern eines Schrittmotors



Schrittstellung	IN1	IN2	IN3	IN4
1	HIGH	LOW	LOW	LOW
2	HIGH	HIGH	LOW	LOW
3	LOW	HIGH	LOW	LOW
4	LOW	HIGH	HIGH	LOW
5	LOW	LOW	HIGH	LOW
6	LOW	LOW	HIGH	HIGH
7	LOW	LOW	LOW	HIGH
8	HIGH	LOW	LOW	HIGH

Tabelle 2: Schrittmotoransteuerung

6 Workshop Abend 3

6.1 Servos ansteuern

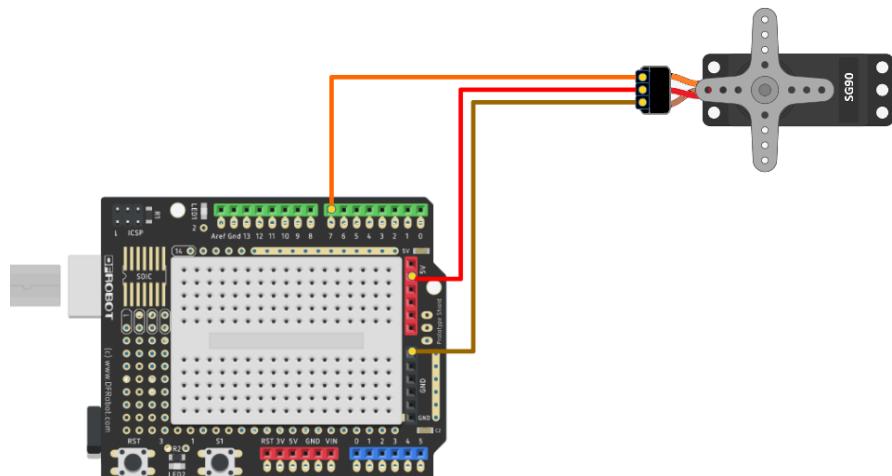


Abbildung 15: Schaltung 5 für das Ansteuern des Servos

```

1  /*
2   * Mit der Pulsweitenmodulation die Motoren steuern
3   */
4
5 #include <Servo.h>
6
7 Servo servo;
8 int servoPin = 7;
9
10 // Variablen, welche die ADC-Werte speichern

```

```

11 | int sensorWertX = 0;           // ausgelesener Wert des Potentiometer X
12 | int ausgabeWertX = 0;          // berechneter Ausgabewert X
13 |
14 // Die setup-Funktion wird einmalig nach "reset" oder power-up" ausgeführt
15 void setup()
16 {
17     // Initialisierung der seriellen Schnittstelle (UART) mit einer BAUD-Rate von 9600 (Bits pro Sekunde)
18     Serial1.begin(9600);
19     // Ausgabe eines Begrüßungstextes
20     Serial1.println("Hallo Welt");
21
22     servo.attach(servoPin);
23
24 }
25 // Die loop-Funktion wird ständig wiederholt durchlaufen
26 void loop()
27 {
28     // Ausgangsposition anfahren (Wert von 0 bis 180 Grad)
29     servo.write(0);
30     delay(500);
31
32     for (int stellung = 0 ; stellung <= 180; stellung += 10) {
33         // Schreibt die Stellung in den PWM-Pin (Wert von 0 bis 180 Grad):
34         servo.write(stellung);
35         // Warten bis zur nächsten Änderung der Stellung
36         delay(500);
37         Serial1.println("Tick");
38     }
39
40     for (int stellung = 180 ; stellung >= 0; stellung -= 10) {
41         // Schreibt die Stellung in den PWM-Pin (Wert von 0 bis 180 Grad):
42         servo.write(stellung);
43         // Warten bis zur nächsten Änderung der Stellung
44         delay(500);
45         Serial1.println("Tick");
46     }
47
48
49 /*
50 * Das auslesen des Joysticks wird zunächst auskommentiert
51
52 // Auslesen der Spannung am A0-Pin für X
53 sensorWertX = analogRead(A0);
54 // Zuweisung/Aufspreizung des analogen Wertes von 0 .. 1023 auf einen Wert von 0 .. 65535
55 ausgabeWertX = map(sensorWertX, 0, 1023, 0, 65535);
56 // Ausgabe des Rohwertes und des berechneten/transformierten Wertes über die serielle Schnittstelle
57 Serial.print("Rohwert= ");
58 Serial.print(sensorWertX);
59 Serial.print(" Ausgabe= ");
60 Serial.println(ausgabeWertX);

```

```

61 */
62     */
63     delay(100); // warten bis zum nächsten Aufruf
64
65
66 }

```

Listing 11: Servo (Quellcode unter www.github.de/arduino)

6.2 Messen mit dem ADC. Der Joystick!

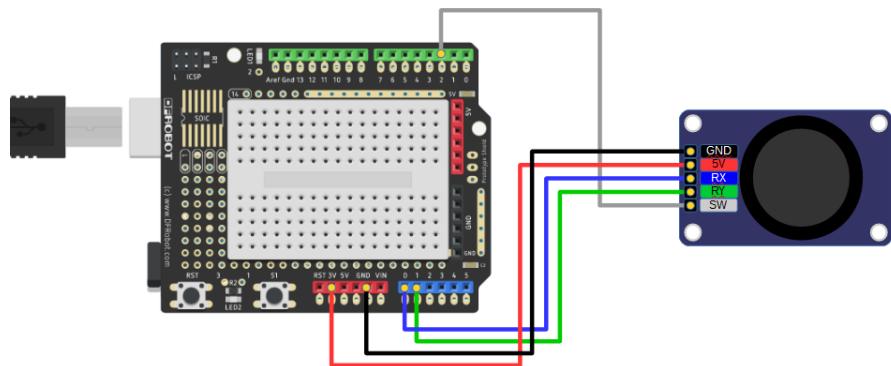


Abbildung 16: Schaltung 4 für unser das Messen mit dem ADC

```

1 /*
2  * Mit dem ADC Spannungen messen
3 */
4
5 // buttonEvent zeigt ein zu bearbeitendes Taster-Ereignis (zu bearbeitenden Interrupt)
6 // an
7 int buttonEvent = 0;
8
9 // Variablen, welche die ADC-Werte speichern
10 int sensorWertX = 0;           // ausgelesener Wert des Potentiometer X
11 int ausgabeWertX = 0;          // berechneter Ausgabewert X
12
13 void tastenDruck() {
14     // Setzen des Taster-Ereignisses
15     buttonEvent = 1;
16     // in Interrupt-Funktionen sollten keine längeren oder rechenintensiven Befehle
17     // ausgeführt werden
18     // deshalb wird die Serial.println hier nicht direkt aufgerufen.
19 }
20
21 // Die setup-Funktion wird einmalig nach "reset" oder power-up" ausgeführt

```

```

20 void setup()
21 {
22     // Initialisiere den Pin 2 als Eingang, der den Taster überwacht
23     // Mithilfe der INPUT_PULLUP wird der interne Pullup-Widerstand (ca. 10kΩ) aktiviert, damit wird ein externer Widerstand eingespart
24     pinMode(2, INPUT_PULLUP);
25     // Zum Verknüpfen der Interrupts wird die attachInterrupt Funktion verwendet
26     // Dabei ist der erste Parameter die Interrupt-Nummer
27     // Für Arduino UNO gilt: Pin 2 => Interrupt 0, Pin 3 => Interrupt 1
28     // Für andere Boards sind andere Pin-Interrupt-Zuweisungen vorhanden
29     // Siehe hierfür: https://www.arduino.cc/en/Reference/AttachInterrupt
30     // Der Parameter "FALLING" bezieht sich auf die fallende Flanke. Durch den Pullup ist diese invertiert
31     attachInterrupt(0, tastenDruck, FALLING);
32
33     // Initialisierung des ADC Pins
34     // diese kann für die Pins A0 ... A5 weggelassen werden
35     // die STM32Duino Board verfügen aber meist über deutlich mehr ADC-Eingänge
36     // für andere ADC-Pins wird der pinMode auf INPUT_ANALOG gesetzt
37     // pinMode(A0, INPUT_ANALOG); // set up pin for analog input
38
39
40
41     // Initialisierung der seriellen Schnittstelle (UART) mit einer BAUD-Rate von 9600 (Bits pro Sekunde)
42     Serial1.begin(115200);
43     // Ausgabe eines Begrüßungstextes
44     delay(500);
45     Serial1.println("Hallo Welt");
46
47     // Initialisierung des A0-Analog-Eingang
48     // Muss nur für den STM32 ausgeführt werden, deshalb hier auskommentiert
49     // pinMode(A0, INPUT_ANALOG);
50 }
51 // Die loop-Funktion wird ständig wiederholt durchlaufen
52 void loop()
53 {
54     // Auf das Taster-Event reagieren, falls dieses gesetzt wurde
55     if(buttonEvent == 1)
56     {
57         Serial1.println("Push the button!");
58         delay(100); // warte 100 millisekunden bis das Ereignis erneut ausgelöst werden kann
59         buttonEvent = 0;
60     }
61
62     // Auslesen der Spannung am A0-Pin für X
63     sensorWertX = analogRead(A0);
64     // Zuweisung/Aufspreizung des analogen Wertes von 0 .. 1023 auf einem Wert von 0 .. 65535
65     ausgabeWertX = map(sensorWertX, 0, 4095, 0, 1000);
66     // Ausgabe des Rohwertes und des berechneten/transformierten Wertes über die serielle Schnittstelle

```

```

67  Serial1.print("Rohwert= ");
68  Serial1.print(sensorWertX);
69  Serial1.print(" Ausgabe= ");
70  Serial1.println(ausgabeWertX);
71
72  delay(100); // warten bis zum nächsten Aufruf
73
74
75 }

```

Listing 12: ADC (Quellcode unter www.github.de/arduino)

6.3 ADC und Servo

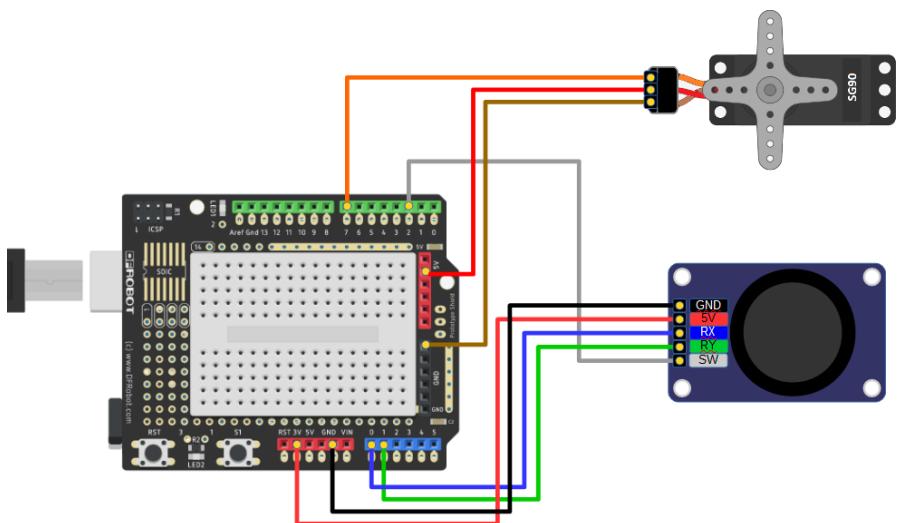


Abbildung 17: Schaltung 4 für unser das Messen mit dem ADC

```

1  /*
2   * Mit dem ADC Spannungen messen
3   */
4  #include <Servo.h>
5
6  Servo servo;
7  int servoPin = 7;
8
9
10 // buttonEvent zeigt ein zu bearbeitendes Taster-Ereignis (zu bearbeitenden Interrupt)
11 // an
12 int buttonEvent = 0;

```

```

13 // Variablen, welche die ADC-Werte speichern
14 int sensorWertX = 0;           // ausgelesener Wert des Potentiometer X
15 int ausgabeWertX = 0;          // berechneter Ausgabewert X
16
17 void tastenDruck() {
18     // Setzen des Taster-Ereignisses
19     buttonEvent = 1;
20     // in Interrupt-Funktionen sollten keine längeren oder rechenintensiven Befehle
21     // ausgeführt werden
22     // deshalb wird die Serial.println hier nicht direkt aufgerufen.
23 }
24
25 // Die setup-Funktion wird einmalig nach "reset" oder power-up" ausgeführt
26 void setup()
27 {
28     // Initialisiere den Pin 2 als Eingang, der den Taster überwacht
29     // Mithilfe der INPUT_PULLUP wird der interne Pullup-Widerstand (ca. 10kΩ) aktiviert, damit wird ein externer Widerstand eingespart
30     pinMode(2, INPUT_PULLUP);
31     // Zum Verknüpfen der Interrupts wird die attachInterrupt Funktion verwendet
32     // Dabei ist der erste Parameter die Interrupt-Nummer
33     // Für Arduino UNO gilt: Pin 2 => Interrupt 0, Pin 3 => Interrupt 1
34     // Für andere Boards sind andere Pin-Interrupt-Zuweisungen vorhanden
35     // Siehe hierfür: https://www.arduino.cc/en/Reference/AttachInterrupt
36     // Der Parameter "FALLING" bezieht sich auf die fallende Flanke. Durch den Pullup ist diese invertiert
37     attachInterrupt(0, tastenDruck, FALLING);
38
39     // Initialisierung des ADC Pins
40     // diese kann für die Pins A0 ... A5 weggelassen werden
41     // die STM32Duino Board verfügen aber meist über deutlich mehr ADC-Eingänge
42     // pinMode(A0, INPUT_ANALOG); // set up pin for analog input
43
44
45
46     // Initialisierung der seriellen Schnittstelle (UART) mit einer BAUD-Rate von 9600 (Bits pro Sekunde)
47     Serial1.begin(115200);
48     // Ausgabe eines Begrüßungstextes
49     delay(500);
50     Serial1.println("Hallo Welt");
51
52     servo.attach(servoPin);
53
54 }
55
56 // Die loop-Funktion wird ständig wiederholt durchlaufen
57 void loop()
58 {
59     // Auf das Taster-Event reagieren, falls dieses gesetzt wurde
60     if(buttonEvent == 1)
61     {
62         Serial1.println("Push the button!");

```

```

62     delay(100); // warte 100 millisekunden bis das Ereignis erneut ausgelöst werden
63     kann
64     buttonEvent = 0;
65
66 // Auslesen der Spannung am A0-Pin für X
67 sensorWertX = analogRead(A0);
68 // Zuweisung/Aufspreizung des analogen Wertes von 0 .. 1023 auf einen Wert von 0 ..
69   65535
70 ausgabeWertX = map(sensorWertX, 0, 4095, 0, 180);
71 // Ausgabe des Rohwertes und des berechneten/transformierten Wertes über die
72   serielle Schnittstelle
73 Serial1.print("Rohwert= ");
74 Serial1.print(sensorWertX);
75 Serial1.print(" Ausgabe= ");
76 Serial1.println(ausgabeWertX);
77
78 servo.write(ausgabeWertX);
79 delay(100); // warten bis zum nächsten Aufruf
80

```

Listing 13: ADC & Servo (Quellcode unter www.github.de/arduino)

6.4 Bonus: Jede Menge Extras

eSchrittmotoren, Die LED Anzeige, Ansteuern von Relays, Pulsweitenmodulation (PWM), WLAN

6.5 Bonus: Steuerung und Visualisierung mit Python

6.6 Hausaufgabe: LED Matrix

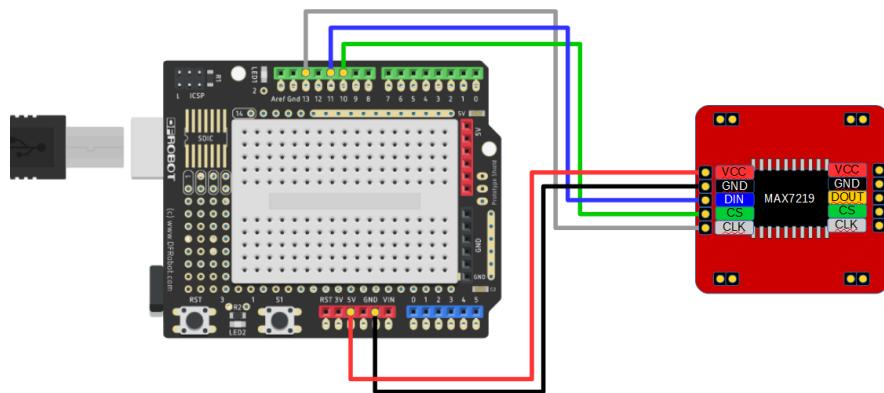


Abbildung 18: Schaltung 5 für die Ansteuerung der LED Matrix

```

1 #include <SPI.h>
2
3 // Definition des Chip-Select-Pins zur Kommunikation über die SPI-Schnittstelle des
4 // MAX7219/MAX7221
5 // Wir verwenden Pin 10
6 #define CS_PIN    10
7
8 // Variable zum setzen der Helligkeit
9 int helligkeit = 0;
10
11 void maxTransfer(uint8_t address, uint8_t value) {
12
13     // Mit dem MAX7219 wird über die SPI-Schnittstelle kommuniziert
14     // Den Verlauf der Kommunikation ist im Datenblatt (siehe Ordner) Abbildung 1
15     // dargestellt
16     // Die Einstellungen werden im LED-Treiber in Registern gespeichert
17     // Es müssen jeweils 2 Bytes übertragen werden um diese zu ändern
18     // Ein Adressbyte und ein Datenbyte (siehe Tabelle 1 im Datenblatt)
19
20     // Der CS-Pin von HIGH zu LOW signalisiert den Start der Übertragung
21     digitalWrite(CS_PIN, LOW);
22
23     // Senden des 1. Bytes. Die Adresse des Registers
24     SPI.transfer(address);
25
26     // Senden des 2. Bytes. Der Wert, der in das Register geschrieben werden soll
27     SPI.transfer(value);
28
29 }
30
31 void setup() {
32
33     // den CS-PIN (Chip-Select benötigen wir zum Start/Stop der Kommunikation)
34     pinMode(CS_PIN, OUTPUT);
35
36     // Initialisieren der SPI-Schnittstelle (Standardeinstellungen)
37     SPI.begin();
38
39     // Setzen der Dekodierung auf Matrix-Ansteuerung
40     maxTransfer(0x09, 0x00);
41     // Setzen der Leuchtintensität
42     maxTransfer(0x0A, 0x00);
43     // Setzen der Zeilen (8 Zeilen)
44     maxTransfer(0x0B, 0x07);
45     // Aktivierung der Anzeige
46     maxTransfer(0x0C, 0x01);
47
48 }
49
50 void loop() {
51

```

```
52 // Wir verwenden die binäre Zahlendarstellung in C
53 // ein Byte entspricht einer Zeile
54 // jedes Bit stellt eine LED einer Zeile dar (0 = AUS, 1 = AN)
55 // wir müssen für jede Zeile (1-8) die LED einschalten
56 // dafür benutzen wir die maxTransfer-Funktion mit den Parametern Adresse des
57 // Registers und Wert
58 // ein Register kann als beschreibbare Variable gesehen werden
59 // die Details zu den Registern finden sich im Datenblatt im Ordner "Datenblaetter"
60 maxTransfer(0x01, 0b00000000);
61 maxTransfer(0x02, 0b01101100);
62 maxTransfer(0x03, 0b10010010);
63 maxTransfer(0x04, 0b10000010);
64 maxTransfer(0x05, 0b01000100);
65 maxTransfer(0x06, 0b00101000);
66 maxTransfer(0x07, 0b00010000);
67 maxTransfer(0x08, 0b00000000);
68 // Zusätzlich können wir die Helligkeit einstellen
69 maxTransfer(0x0A, helligkeit);
70
71 // warten
72 delay(1500);
73
74 }
```

Listing 14: LED-Matric (Quellcode unter www.github.de/arduino)

7 Ausblick

7.1 Grenzen des Arduino / STM32Duino

7.2 Alternative Entwicklungsumgebungen

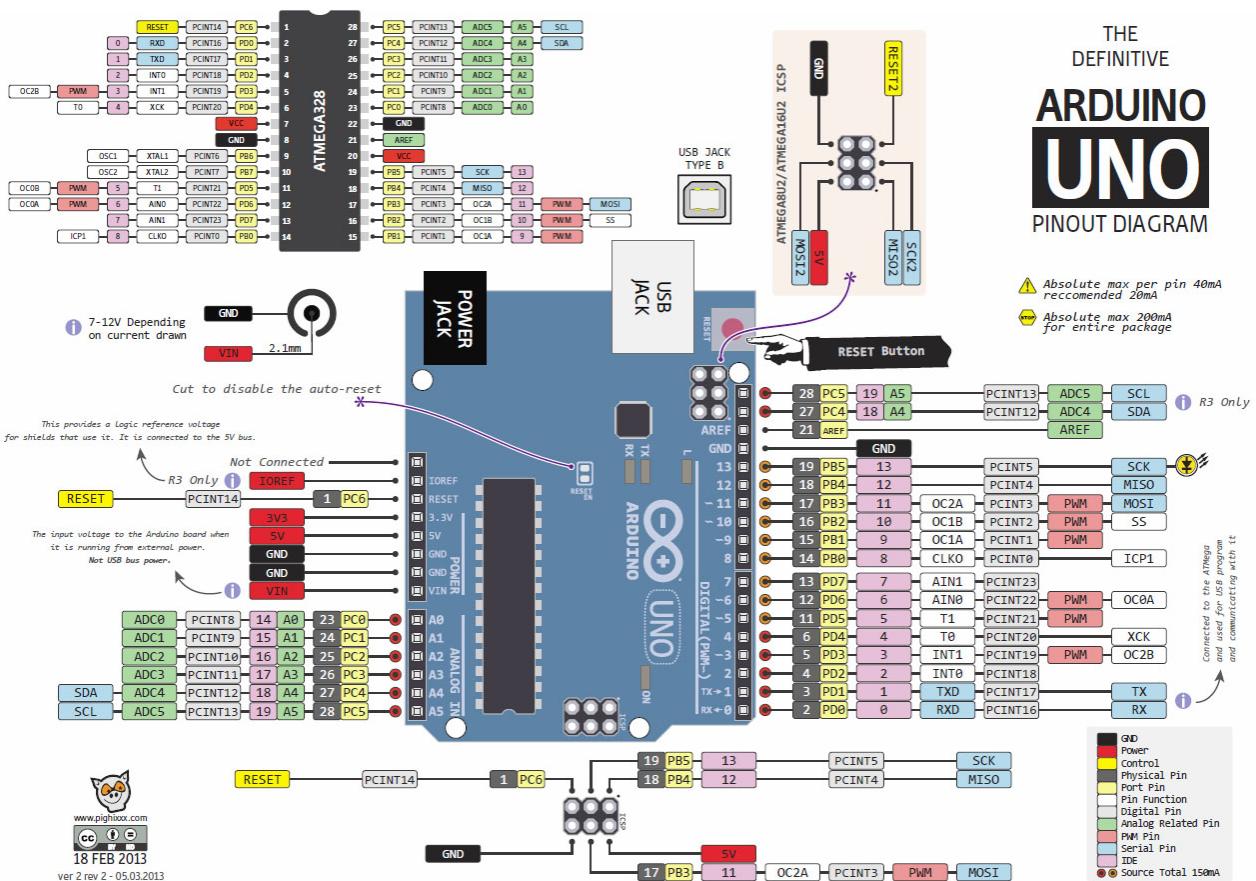
7.3 Alternative Mikrocontroller

7.4 Einplatinenrechner

7.5 Projekte

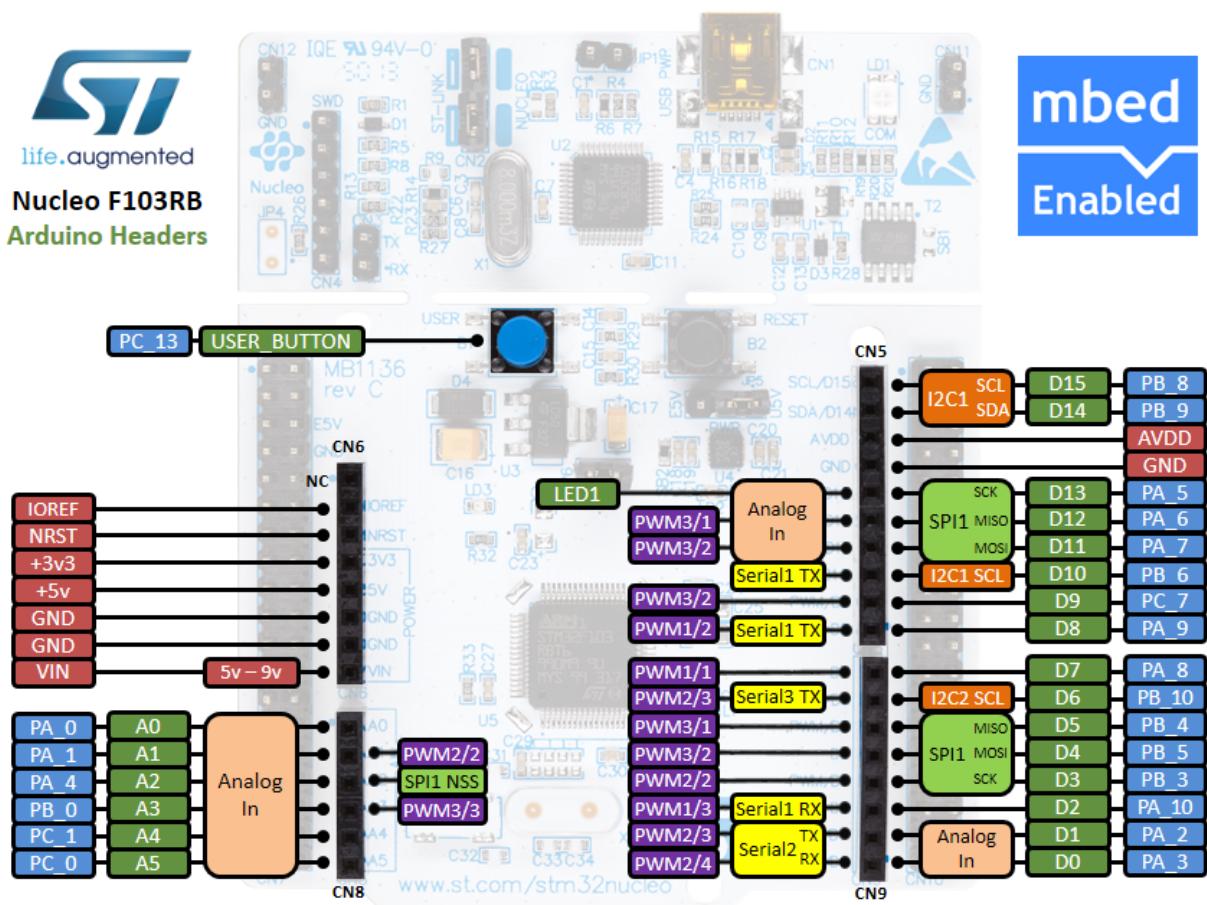


Anhang: Pinbelegungen



Arduino Uno



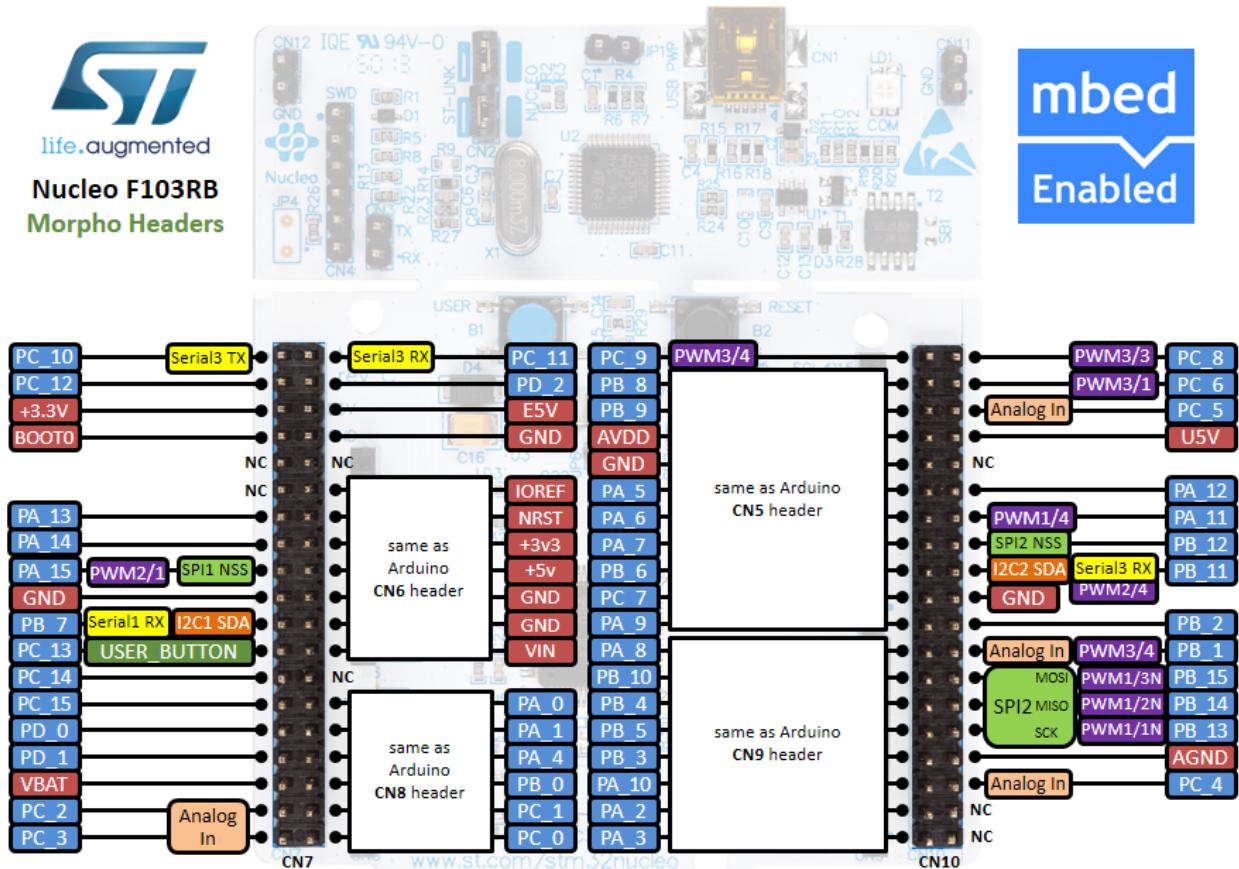


STM32F103 Nucleo Arduino Pinbelegung (Quelle: www.heise.de)

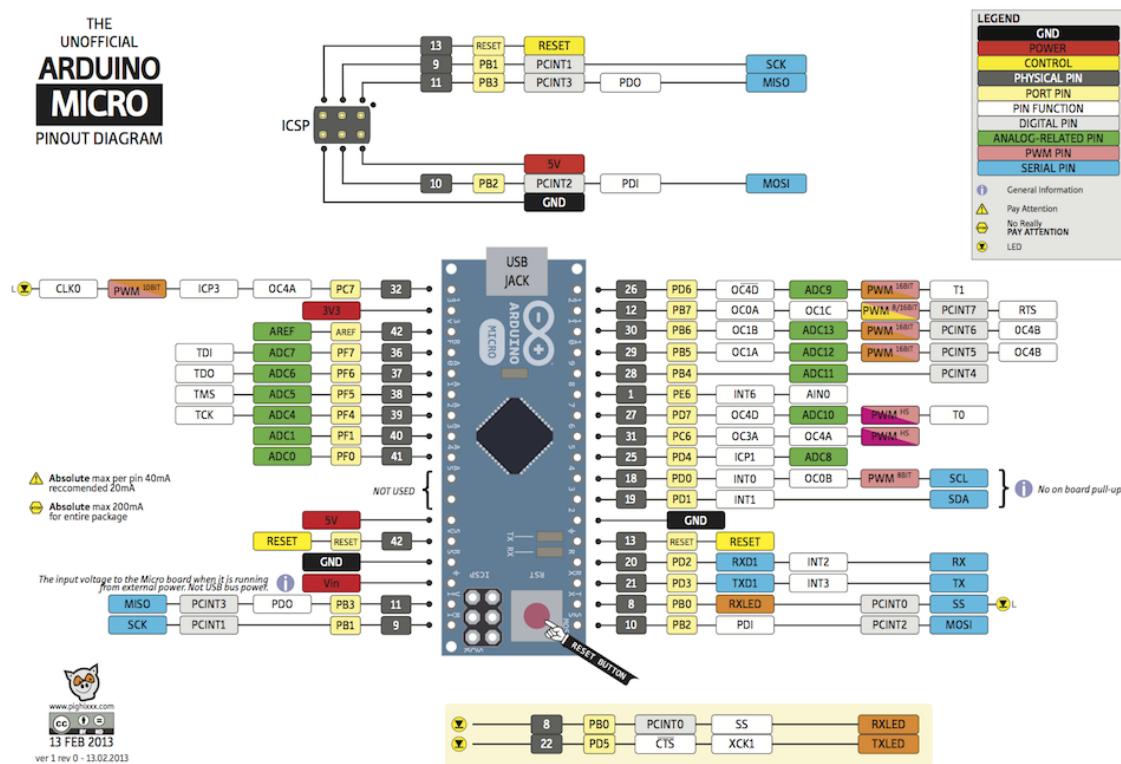
Nucleo-F103RB Pinout Sheet for Arduino IDE

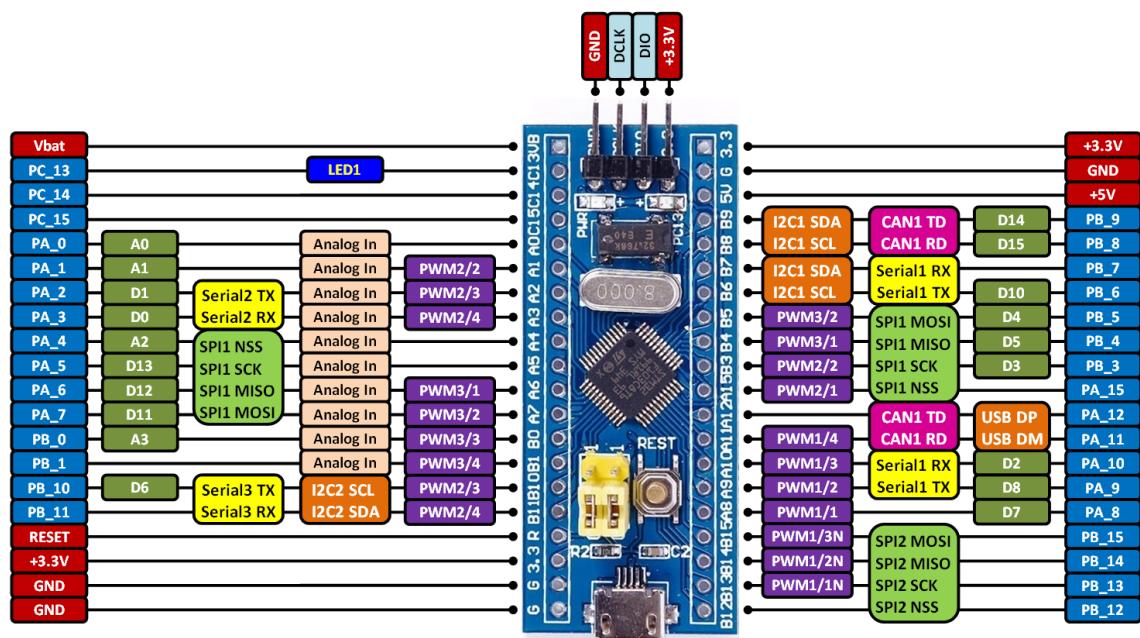
left					right					
Arduino	Port	Pin	Pin	Port	Arduino	Port	Pin	Pin	Port	Arduino
TX	D22	PC10	1	2	PC11	D30		RX		
	D23	PC12	3	4	PD2	D31				
	VDD	5	6	E5V						
	Boot0	7	8	GND						
	NC	9	10	NC	NC					
	NC	11	12	IoRef	IoRef					
	NC	PA13	13	14	RST	RST				
	NC	PA14	15	16	3,3V	3,3V				
	NC	PA15	17	18	5V	5V				
	GND	19	20	GND						
I2C1_SDA	T4	D24	PB7	21	22	GND				
Button	D25	PC13	23	24	Vin	Vin				
32khz!!	D26	PC14	25	26	NC					
32khz!!	D27	PC15	27	28	PA0	A0/D16	0			
OSC**		PD0	29	30	PA1	A1/D17	1			
OSC**		PD1	31	32	PA4	A2/D18	4	SPI1_SS		
		Vbat	33	34	PB0	A3/D19	8	T3		
	12	D28	PC2	35	36	PC1	A4/D20	11		
	13	D29	PC3	37	38	PC0	A5/D21	10		
Pin	Pin Nr.		Dxx	Arduino Pin		12	ADC Pin (Channel)			
Port	Port + Number		Dxx	Morpho Pin		T1	PWM Pin (T=Timer)			
Arduino	Arduino IDE Pin ID		1	5V Tolerant		TX1	Serial			
			SPI1	SPI Pin		I2C1	I2C Pin			

STM32F103 Nucleo Konfiguration (Quelle: <http://library.skema.edu/>)



STM32F103 Nucleo vollständige Pinbelegung (Quelle: www.heise.de)



STM32F103C8T6 (Quelle: <http://thdarduino.blogspot.de/>)

Anhang: Übungen

Workshop Abend 1:

- Fragen
 - Wie wird ein Ausgang für den Arduino gesetzt?
 - Wie muss der Quellcode für den schnellstmöglichen Wechseln eines GPIO-Ausgangspins in einer while-Schleife aussehen? (mit den gelernten Funktionen)
 - Wie heißt der Funktionskopf (Prototyp) für eine Funktion foo, die als Parameter bar (ganzahlige, positiv, 8-bit) verwendet und einen Wert des gleichen Typs zurückgibt?
 - Welche Parameter müssen für den Einsatz eines Timers zwingend gesetzt werden (Initialisierung und Konfiguration)?
- Quellcode korrigieren



Anhang: C Programmierung für Mikrocontroller

```
1 x |= (1 << Bitnummer); // So wird ein Bit in x gesetzt  
2 x &= ~(1 << Bitnummer); // So wird ein Bit in x geloescht  
3 x ^= (1 << Bitnummer); // So wird ein Bit in x getoggelt/umgeschalten
```

Listing 15: Bitmanipulation



Anhang: Bibliotheken und Dokumentation

Marcus Drobisch, Konglomerat e.V., Dresden, Germany, 2016

Licence under Creative Commons CC-by-nc-sa



Anhang: Alternative Arduino-Boards

Der Vorteil der Arduino-Umgebung ist die vielseitige Einsatzmöglichkeit unterschiedlicher Boards. Während im Kurs

Empfohlene Boards

Unterschiede bei der Programmierung



Anhang: Alternative Entwicklungsumgebung OpenSTM32

OpenSTM32 bietet eine Alternative Entwicklungsumgebung für das im Kurs verwendeten Nucleo Board. Es richtet sich an fortgeschrittene und professionelle Programmier bzw. Entwickler. Besonders die Möglichkeit den Programmcode Schritt für Schritt durchlaufen zu können und an bestimmten Stellen (Breakpoints), wie im Simulator anzuhalten, wird von vielen Entwicklern vermisst. Auch kann deutlich tiefer in die einzelnen Schnittstellen eingegriffen werden, als das bei Arduino der Fall ist. Dies ermöglicht die Performance des Mikrocontroller vollständig auszuschöpfen und unterscheidet sich stark von der Verwendung der Arduino-Umgebung. Zusätzlich bietet STM große gut gepflegte Bibliotheken an, die die Konfiguration einzelner Bausteine vereinfachen. Aus diesem Grund unterscheidet sich die Arbeit mit OpenSTM32 hier auch vollkommen von der mit der Arduino-Umgebung. Die Programmierung ist weniger Intuitiv und die Arbeit weniger für Anfänger geeignet.

An dieser Stelle soll die Installation sowie das Erstellen und Compilieren eines Blink_Projektes gezeigt werden. OpenSTM32 wird kostenlos von AC6 im Auftrag von STM zur Verfügung gestellt und basiert auf der weit verbreiteten Eclipse Umgebung.

Schritt 1: Download & Installation

OpenSTM32 kann kostenlos unter <http://www.openstm32.org/> heruntergeladen werden. Um in den Downloadbereich zu gelangen ist eine Registrierung und Anmeldung nötig. Anschließend kann OpenSTM32 auf bekannte Weise installiert werden

Schritt 2: Öffnen von OpenSTM32

Nach dem öffnen von OpenSTM32 (Eclipse.exe) wird zunächst der Workspace abgefragt. Der Workspace ist der Speicherort (Verzeichnis) für ein oder mehrere Projekte und kann beliebig gewählt werden. Nach Auswahl des Workspaces kommt man anschließend auf die Willkommensseite.

Schritt 3: Projektname und Projekttyp

Nun werden wir nach einem Projektname (z.B. „blink“) und dem Projekttyp bzw. der Toolchain (Übersetzungswerkzeuge) gefragt. Als Projekttyp wählen wir „Ac6 STM32 MCU Project“. Als Toolchain die Auswahl „Ac6 STM32 MCU GCC“. Anschließend wird die Eingabe mit „Next“ bestätigt.

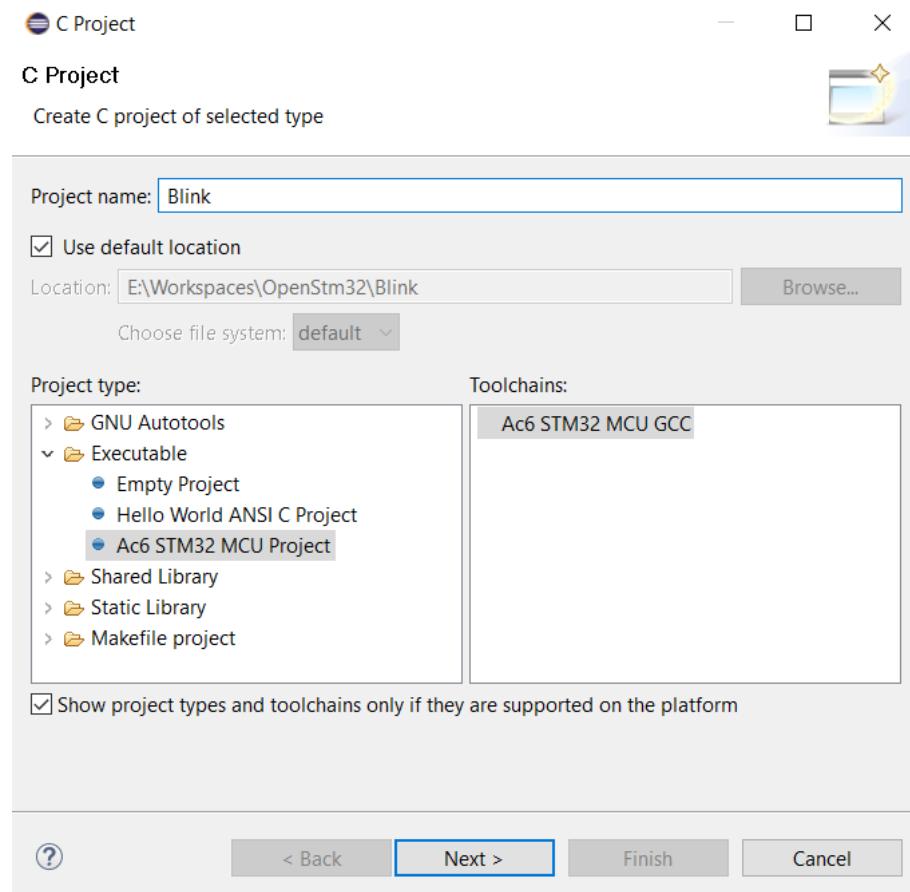


Abbildung 19: Schaltung 5 für die Ansteuerung der LED Matrix

Schritt 4: Projektname und Projekttyp

Nun werden wir nach einem Projektname (z.B. „blink“) und dem Projekttyp bzw. der Toolchain (Übersetzungswerkzeuge) gefragt. Als Projekttyp wählen wir „Ac6 STM32 MCU Project“. Als Toolchain die Auswahl „Ac6 STM32 MCU GCC“. Anschließend wird die Eingabe mit „Next“ bestätigt.

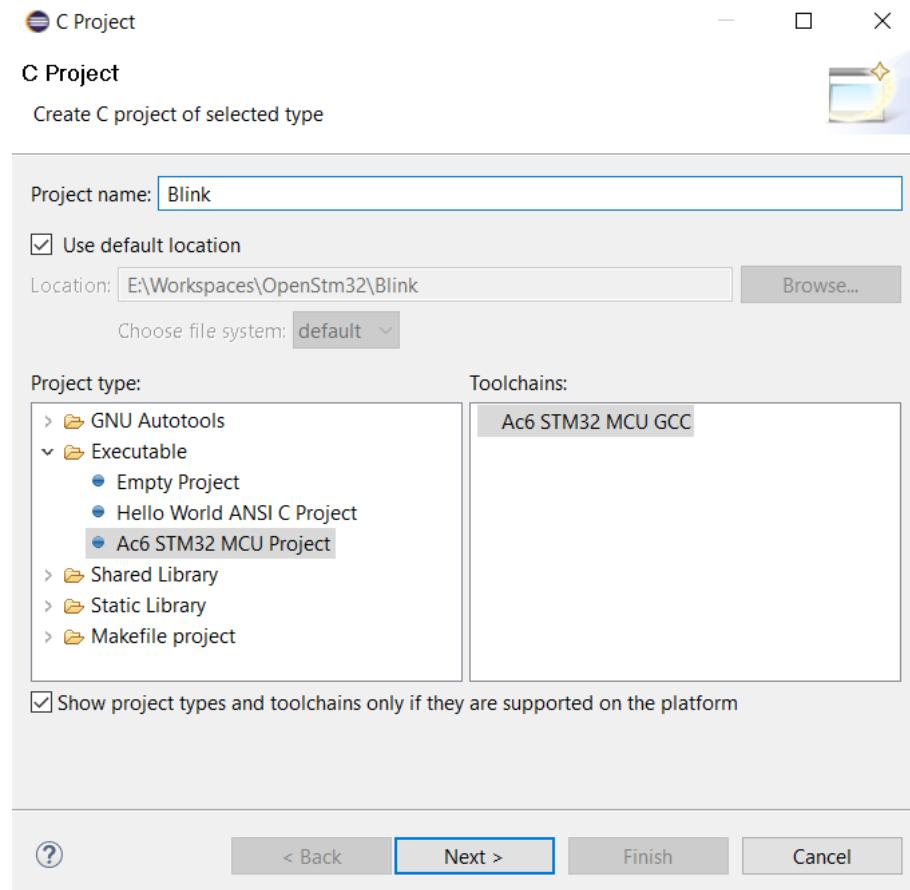


Abbildung 20: Schaltung 5 für die Ansteuerung der LED Matrix

Schritt 5: Projektkonfigurationen

Unter Projektkonfigurationen verstehen sich unterschiedlich Übersetzungmechanismen für den Microkontroller. In den meisten Fällen reicht eine einzelne „Debug“-Konfiguration. Die „Release“-Konfiguration wird abgewählt und mit „Next“ bestätigt.

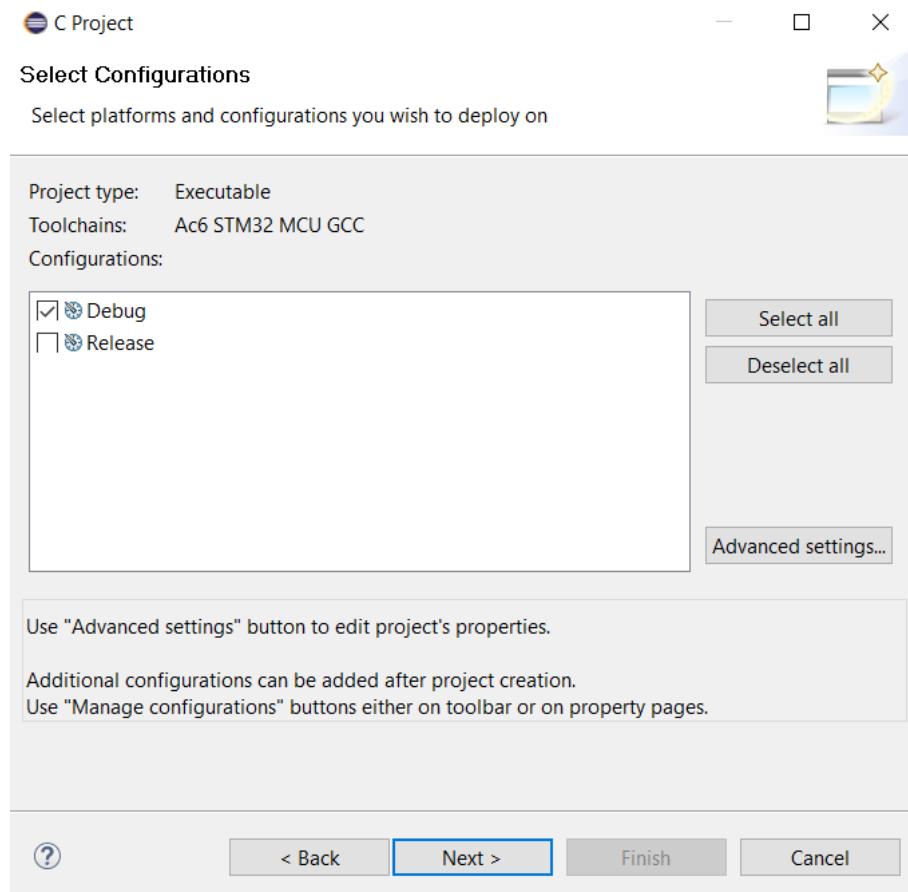


Abbildung 21: Schaltung 5 für die Ansteuerung der LED Matrix

Schritt 6: Board-Auswahl

Im nächsten Schritt geben wir das Mikrocontroller-Board an. Mikrocontroller-Familie (Series) wählen wir so z.B. die „STM32F1“-Familie aus und haben anschließend das Nucleo-Board zur Auswahl. Für andere Nucleo-Board ist dies Analog möglich. Zusätzlich können an dieser Stelle auch eigene Boards nach Vorlagen erstellt werden. Anschließend bestätigen wir unsere Auswahl mit „Next“.

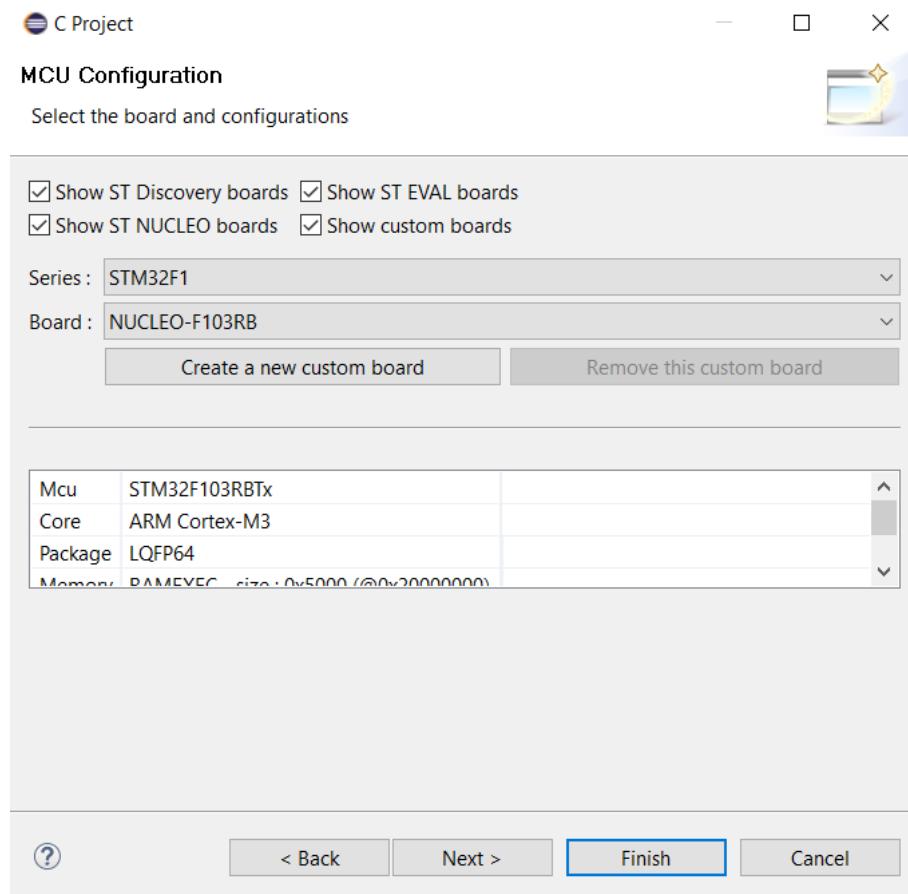


Abbildung 22: Schaltung 5 für die Ansteuerung der LED Matrix

Schritt 7: Mikrocontroller-Bibliotheken

Wie bereits erwähnt stellt STM bzw. ARM umfangreiche Bibliotheken zur Ansteuerungen der Schnittstellen und Module zuer Verfüzung. Den sogenannten „Cube HAL“. Diese sollen auch hier eingesetzt und ausgewählt werden. Nach der Auswahl der „Cube HAL“ können diese per Download in das Projekt hinzugefügt werden (einmaliger Vorgang). Die Voreinstellungen („Add low level drivers ...“ und „As static external libraries“) sollten beibehalten werden. Zusätzlich können Mikrocontrollerahngige Bibliotheken für z.B. Touch oder USB eingebunden werden. Für unser Beispielprojekt verzichten wir darauf. Auch Bibliotheken von 3-Anbieter (wie die Dateisystem-Unterstützgn FatFS oder das Echtzeit-Betriebssystem FreeRTOS) werden wir hier nicht einsetzen. Wir schließen unsere Projektkonfiguration mit „Finish“ ab. OpenSTM32 wird das Projekt nun für uns erstellen.

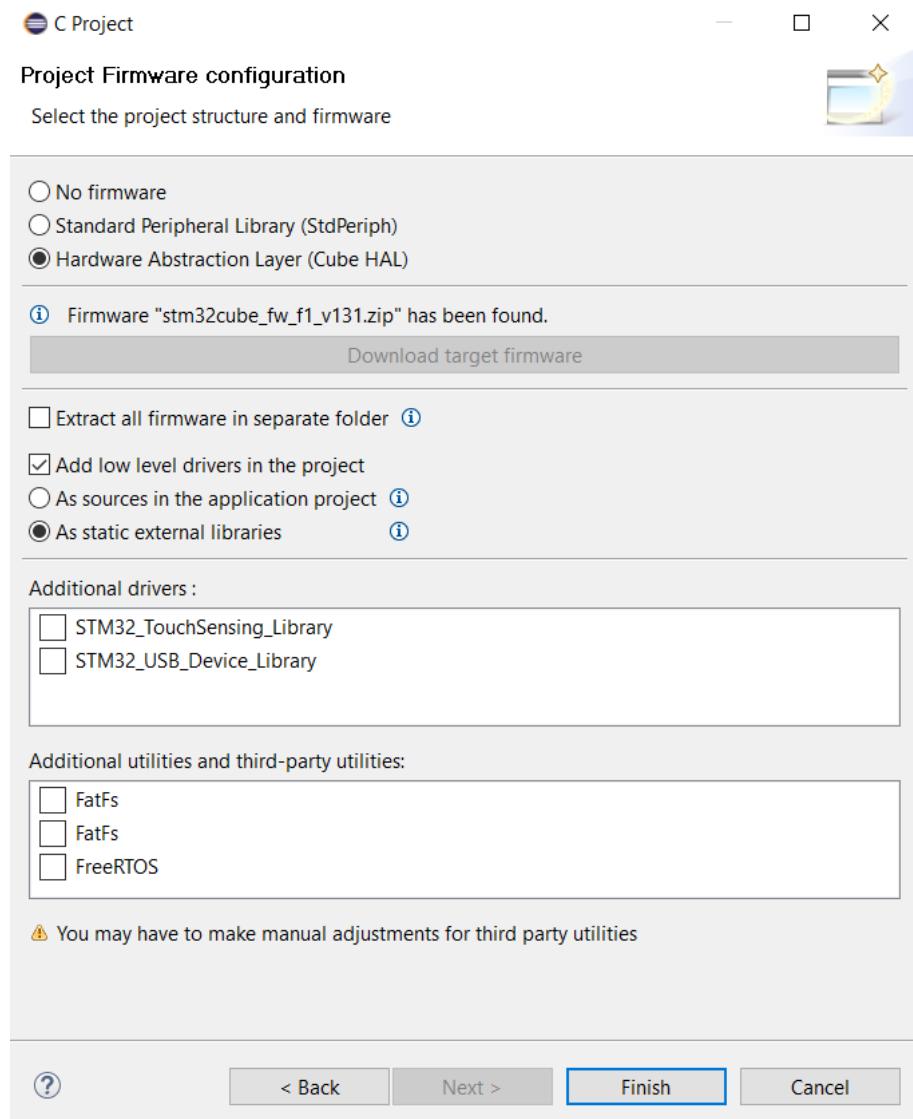


Abbildung 23: Schaltung 5 für die Ansteuerung der LED Matrix

Schritt 8: Programmierung

Nachdem das Projekt eingerichtet wurde sehen wir uns zunächst den nötigen Quellcode an. Zu beachten ist die grundlegend andere Struktur und Vorgehensweise im Vergleich zu den Programmen in der Arduino-Umgebung. Der Startpunkt eines C-Projektes ist die main()-Funktion, die meist in einer main.c-Quellcode-Datei eingebettet ist. Der Quellcode (main.c) für das „Blink“-Testprogramm sieht wie folgt aus.

```

1 #include "stm32f1xx.h"
2 #include "stm32f1xx_nucleo.h"
3
4 int main(void)
5 {

```

```

6 // initialisation of variables, interfaces and modules
7 int counter = 0;
8 HAL_Init();
9 GPIO_InitTypeDef GPIO_InitStruct;
10 /* GPIO Ports Clock Enable */
11 __GPIOA_CLK_ENABLE();
12 /*Configure GPIO pin : P11 */
13 GPIO_InitStruct.Pin = GPIO_PIN_5;
14 GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
15 GPIO_InitStruct.Pull = GPIO_NOPULL;
16 GPIO_InitStruct.Speed = GPIO_SPEED_LOW;
17 HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
18
19 // the main loop
20 while(1)
21 {
22     HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
23     HAL_Delay(100);
24     counter += 1;
25 }
26 }
```

Listing 16: Blink (Quellcode unter www.github.de/arduino)

Schritt 9: Kompilieren (Build)

Nach der Eingabe des Quellcodes können wir den Kompilierprozess starten. Dies wird mit „Project->Build All“ angestoßen. Durch einen Fehler in OpenSTM können fälschlicherweise Fehlermeldungen in der Form „Programm arm-.... not found in PATH“ eingeblendet werden. Diese können per Rechtsklick gelöscht werden und tauchen anschließend nicht wieder auf. Der Fehler in die OpenSTM-Umgebung mit Quellcode sieht entsprechend der folgenden Abbildung aus. Der „Build“-Vorgang sollte dabei ohne Fehler („Build Finished“) durchlaufen.



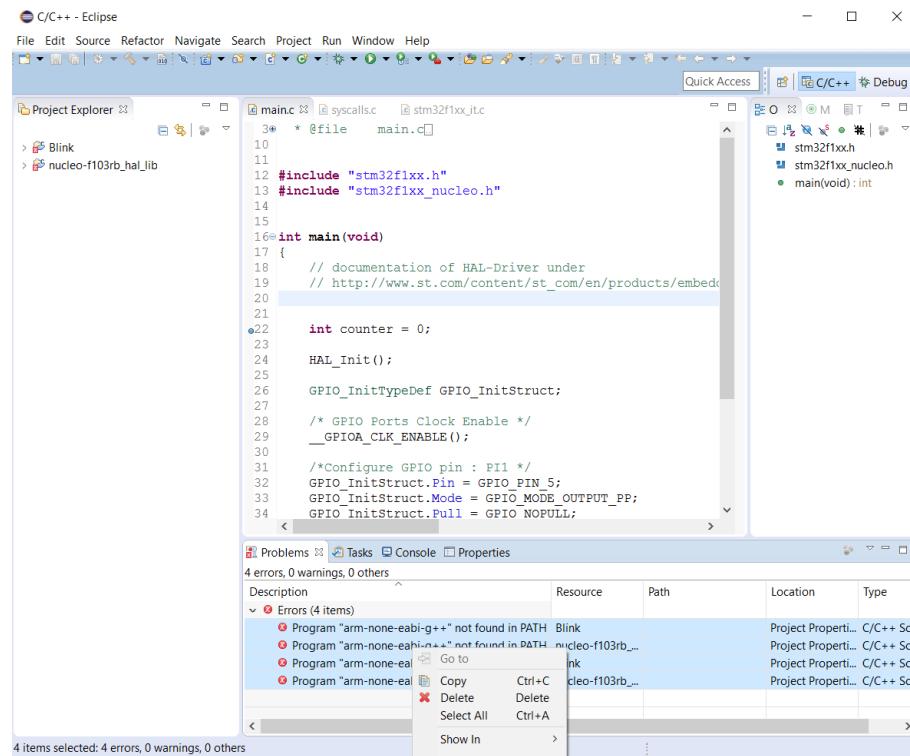


Abbildung 24: Schaltung 5 für die Ansteuerung der LED Matrix

Schritt 10: Programme auf das Nucelo-Board laden und ausführen (Flash)

Das kompilierte Programm kann im nächsten Schritt mit „Run->Run“ auf das Board und den Mikrocontroller geladen werden. Das letzte Programm ist anschließend permanent auf dem Mikrocontroller gespeichert und startet auch ohne Programmierschnittstelle zum PC direkt nach einem „Power-Up“.

Schritt 11: Fehleranalyse (Debug)

Meist ist es notwendig Fehler im Programmablauf nachzuvollziehen. Eine Methode ist die im Kurs kennengelernte „printf“-Funktion, die Status und Variableninhalte ausgeben kann. In komplizierten Fällen oder bei der Verwendung in Interrupts wird diese Methode jedoch schnell Unübersichtlich oder Unzuverlässig. Mit OpenSTM32 kann ein Debug Modus gestartet werden. Dieser ermöglicht das Schritt-Für-Schrittdurchlaufen des Programmes. Zusätzlich kann an beliebigen Stellen Haltepunkte eingefügt werden. Erreicht das Programm diese Stelle, wird er Prozessor anhalten und auf Eingabe des Programmierers warten. Dies ermöglicht Variablen auszulesen und auch komplexe Verzweigungen einfach nachzuvollziehen.. Durch Doppelklick auf die „Blaue“-Leiste im Quellcode-Fenster können diese Haltepunkte (Breakpoints) eingefügt werden. Anschließend kann der Debug-Modus mit „Run>Debug“ gestartet werden.

Anhang: Blue Pill STM32F103C8T6

Es gibt 2 Möglichkeiten mit dem sogenannten Blue Pill zu arbeiten. Zum einen das Flashen eines STM32Duino Bootloaders. In diesem Fall ist der Blue Pill über die gezeigte Arduino Umgebung mit STM32Duino-Erweiterung Programmierbar. Zum anderen ist die Verwendung des Nucleo-Boards als Programmierschnittstelle möglich. Bisher ist hier aber nur die Verwendung mit der OpenSTM32-Umgebung getestet. Die direkte Verwendung mit der Arduino-Umgebung ist daher bisher nicht vorgesehen.

Verwendung mit Arduino und STM32Duino Boatloader

- Für die Verwendung mit der gezeigten Arduino-Umgebung muss der Blue Pill mit einem Bootloader gespielt worden sein. Diese werden meist ohne diesen Angeboten. Eine Anleitung hierzu findet ihr im nächsten Abschnitt.
- Die Arduino-Umgebung muss wie im Kurs gezeigt Konfiguriert und Lauffähig sein
- Anschluss des Boards erfolgt per USB
- Folgende Einstellungen müssen in der Arduino-Umgebung eingestellt werden:
 - Board: „Generic STM32F103C Series“
 - Variant: „STM32F103C8 (20K RAM 64K Flash)“
 - Upload method: „STM32Duino Bootloader“

ACHTUNG: Manche Board in Verbindung mit einigen PCs/Laptops sollen Probleme haben, die USB-Verbindung richtig zu erkennen. Dies liegt an einer Fehlerhaften Beschaltung mancher Boards von einigen Händlern. Als Lösung bietet sich an zwischen den „PA12“ Pin und 3.3V

Beschaltung mit einem Nucleo Board

Die beiden Jumper CN2 müssen offen sein bzw. entfernt werden.

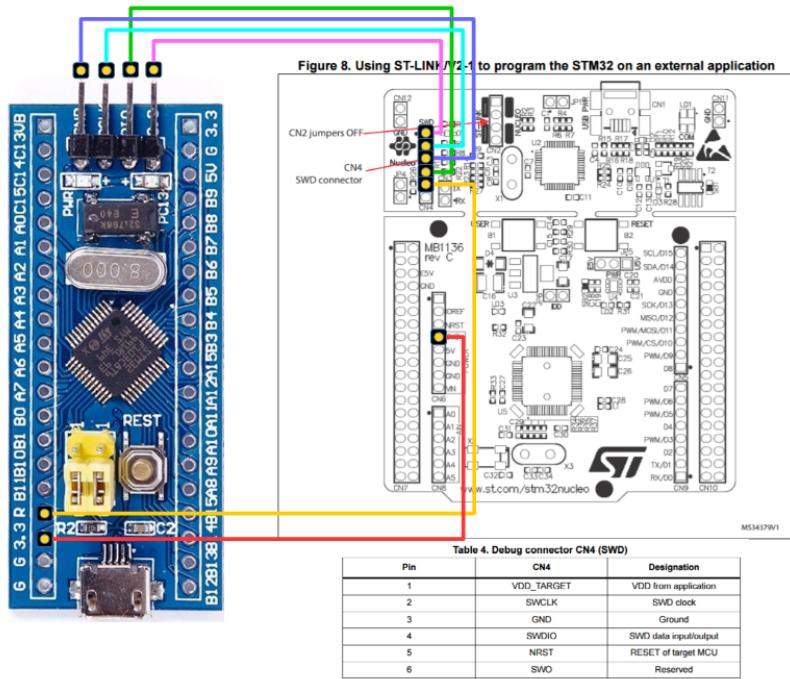


Abbildung 25: Schaltung 5 für die Ansteuerung der LED Matrix

ACHTUNG: Bei Verwendung in Zusammenspiel mit der OpenSTM32-Umgebung muss in Schritt 7 neues Board erstellt werden.

Flash Bootloader

Für den Flash-Vorgang wird das ein Nucleo-Board mit der Beschaltung aus dem vorherigen Abschnitt benötigt. Zusätzlich ist das STM32 LINK Utility von STM zu downloaden und zu installieren. Die benötigte Firmware in kompakter Form wird von STM32Duino zur Verfügung gestellt. Ein Download der passenden Version (generic_boot20_pc13.bin) ist unter <https://github.com/rogerclarkmelbourne/STM32duino-bootloader/tree/master/STM32F1/binaries> möglich.

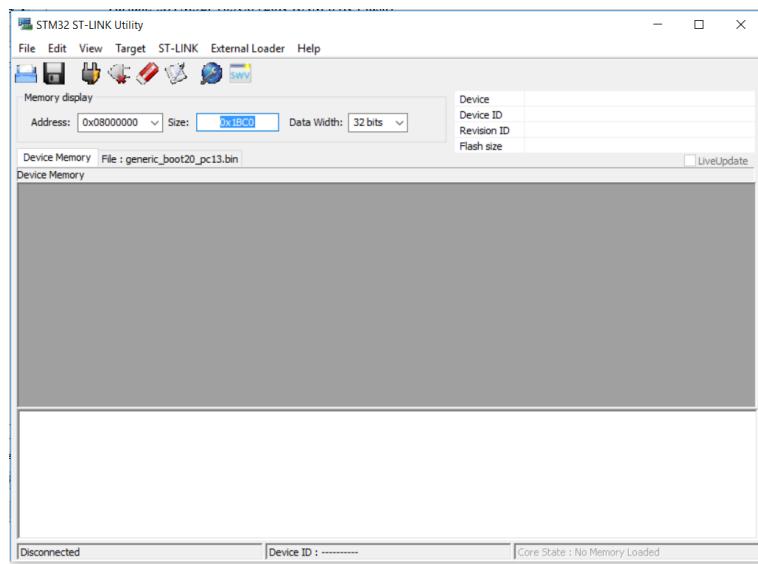


Abbildung 26: Schaltung 5 für die Ansteuerung der LED Matrix

Mit dessen Hilfe de Link Utilities wird die Bootloader-Firmware (generic_boot20_pc13.bin) auf den Mikrocontroller „geflasht“.

Anhang: Links & Projekte

In diesem Kapitel werden Projekte und Seiten vorgestellt, die beim Erlernen und als Ideengeber sinnvoll sind.

Links zum Arduino

Links zur Programmierung von Arduinos

Links zu besonderen Projekten mit Arduinos

