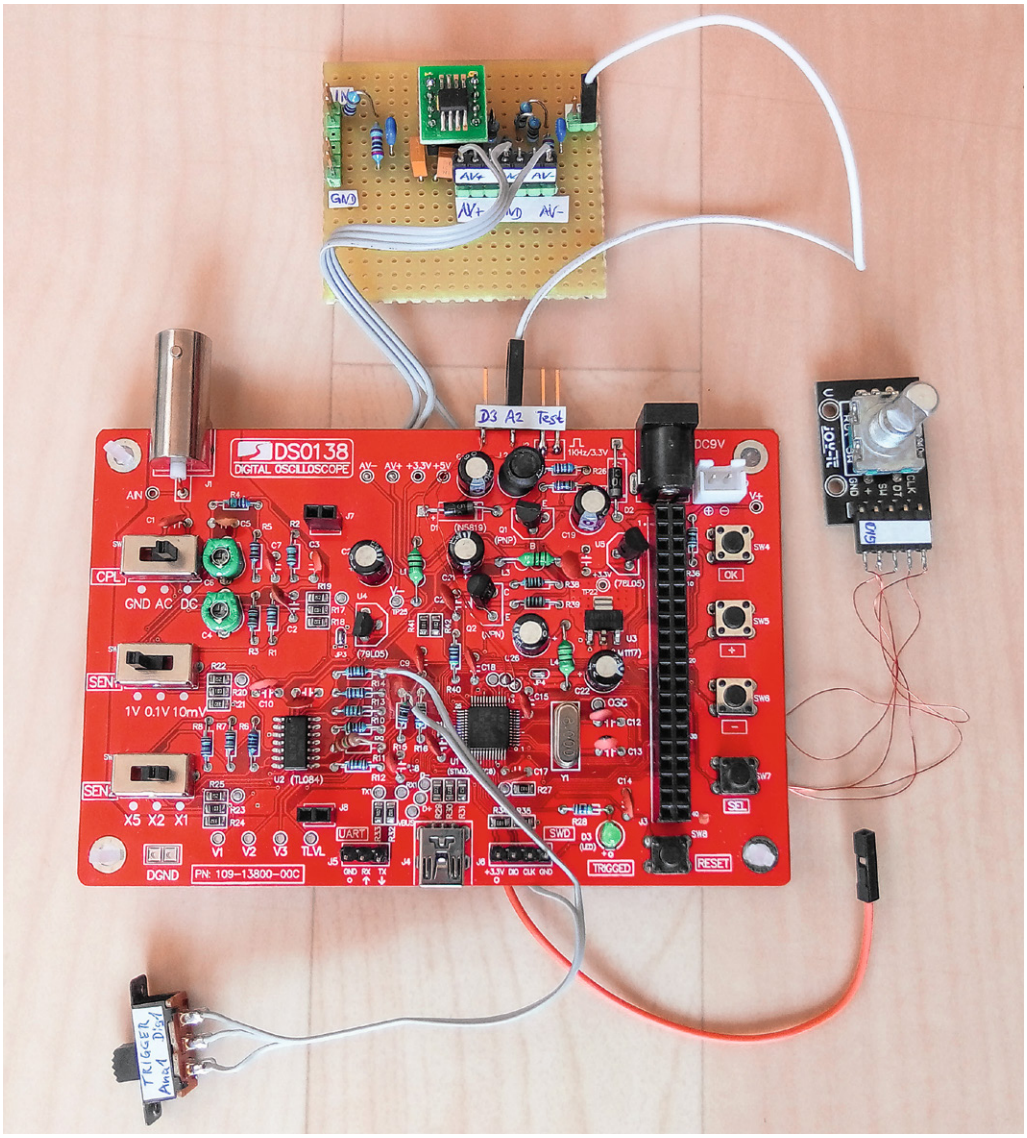
Besser machen

STM32 mit Arduino IDE nutzen und Mini-Oszilloskop DSO138 optimieren, Teil 2

Mit der Arduino IDE steht eine Programmier-Plattform für Arduino-Entwicklungsboards, die hauptsächlich auf Microchip-Prozessoren (ehemals Atmel-Prozessoren) basieren, zur Verfügung. Mit der Zeit sind auch andere Mikrocontroller wie ESP8266/ESP32 von Espressif in diese Entwicklungsumgebung integriert worden. Wir zeigen in unserem praktischen Beispiel anhand der Programmierung des kostengünstig erhältlichen Einsteiger-Oszilloskops DSO138, wie man auch mit STM32/STM8-Controllern von STMicroelectronics diese Plattform nutzen kann. In diesem Teil erfahren wir mehr über mögliche Hardware-Umbauten, über sogenannte STM32-BluePill-Boards und wie man auch STM8-Controller aus der Arduino IDE heraus programmieren kann.



Hinweis: *Alle hier vorgestellten Modifikationen geschehen auf eigenes Risiko und eigene Gefahr. Außerdem erlöschen sämtliche Garantieansprüche.*

DSO138-Hardware umbauen

Während wir uns im ersten Teil mit der Programmierung der verbesserten Firmware für das DSO138 beschäftigt haben, wollen wir im zweiten Teil herausfinden, welche Möglichkeiten es für einen Umbau der Hardware gibt. Alle Downloads (Codes, Schaltpläne, Logos) finden Sie wie immer im Download-Bereich des Beitrags [1].

Zuerst sollen noch einmal kurz die zusätzlichen Features aufgelistet werden, die mit der DLO-138-Firmware bereits ohne Hardwareumbau funktionieren:

- Ausgabe der aufgezeichneten Daten über die serielle USB-CDC-Schnittstelle
- 2048 Samples Speichertiefe (statt 1024)
- Zwei zusätzliche Digitalkanäle. Diese sind bereits an J6 (SWD-Port) herausgeführt

Durch einen Hardware-Umbau bietet die DLO-138-Oszilloskop-Firmware folgende zusätzliche Features:

- Trigger-Quelle umschaltbar zwischen Analogkanal 1 und Digitalkanal 1. Hierzu muss ein mechanischer Umschalter eingebaut werden (Bild 1)
- Nutzung eines Dreh-Encoders anstatt der Plus-, Minus- und Select-Taste für komfortablere Bedienung. Hierzu muss ein Encoder (z. B. KY-040 [2]) an den Tastenpins angeschlossen werden und in der Firmware in der datei global.h der Define USE_ENCODER aktiviert werden
- Zweiter Analogkanal. Zur Nutzung muss die entsprechende Mess-Hardware nachgerüstet werden

Für die umschaltbare Trigger-Quelle zwischen Analogkanal 1 und Digitalkanal 1 muss die Verbindung zwischen R15 und U2D (Pin 14) aufgetrennt (z. B. R15 an der passenden Seite auslöten) und ein Umschalter eingeschleift werden. Hierfür wird das Display vorsichtig aus den Buchsen herausgehebelt, um besser an R15 zu gelangen. Bei Schalterstellung 1–2 ist der Analogkanal 1 die Trigger-Quelle und bei Stellung 2–3 der Digitalkanal 1 am Pin SWDIO (J6 Pin 2).

Das KY-040-Modul wird mit den passenden Pins der drei Tasten verbunden. Der Pin CLK entspricht dem Define ENCODER_A in der DLO-138-Firmware, DT ist ENCODER_B und SW ist ENCODER_SW. Auf dem Modul sind drei Pull-up-Widerstände vorhanden, die mit dem Plus-Pin von J1 verbunden sind (Pin 4) und an 3,3 V angeschlossen werden müssen (Bild 2). Ein erster Test, ob die Verkabelung richtig ist, kann mit angeschlossenem Dreh-Encoder durchgeführt werden, indem man den Taster des Encoders drückt. Dadurch müsste sich auch ohne angepasste Firmware das Select-Feld verändern. Will man den Dreh-Encoder nutzen, muss dazu in der Firmware in der Datei global.h der #Define USE_ENCODER aktiviert (Kommentarzeichen entfernen) werden. Die veränderte Firmware wird dann wie im ersten Teil des Beitrags beschrieben mithilfe der Arduino IDE auf das DSO138 hochgeladen.

Mit dem Dreh-Encoder lassen sich die Einstellungen nun wesentlich komfortabler als mit den Tastern verändern. Die Funktion der OK-/HOLD-Taste bleibt unverändert und ist weiterhin über das DSO138 bedienbar.

Bei dieser abgespeckten Schaltung (Bild 3) wurden auf die Umschaltung der Kopplung sowie die Verstärkungsumschaltung verzichtet. Über den Analogkanal 2 kann dann nur mit DC-Kopplung und 1 V/div gemessen werden. Alternativ kann auch die aus dem DSO138-Schaltplan [3] ersichtliche komplette Messverstärkerschaltung nachgerüstet werden.

Im vorderen Seitenteil des Gehäuses ist die Aussparung für die Mini-USB-Buchse leider nicht groß genug, um ein USB-Kabel einstecken zu können. Entweder vergrößert man diese Aussparung vorsichtig mit einer Laubsäge oder schneidet mit einem Messer bei einem USB-Kabel ein Stück der Kunststoffummantelung an der Ober- und Unterseite des Steckers ab.

Bevor das Gehäuse wieder zusammengebaut wird, lötet man das orange Kabel am BOOT0-Pin auf der Platinenunterseite wieder ab und entfernt noch die Schutzfolie auf dem Display.

Eigene Firmware entwickeln

Die DSO138-Hardware lässt sich auch als universelle Experimentier-Plattform mit STM32-Controller und TFT-Display verwenden. Auf der Platine sind sieben freie GPIO-Pins vorhanden (Tabelle 1), die auf Stiftleisten oder Lötpads herausgeführt sind.

Pin am STM32F103	Portpin	Signal	Verfügbar an
14	PA4	–	TP11
15	PA5	–	TP12
34	PA13	SWDIO	J6 Pin2, 10 kΩ Pull-up vorhanden
37	PA14	SWCLK	J6 Pin3, 10 kΩ Pull-down vorhanden
30	PA9	TXD	J5 Pin3 über 1 kΩ Serienwiderstand
31	PA10	RXD	J5 Pin2 über 1 kΩ Serienwiderstand
17	PA7	–	J2 1 kΩ Serienwiderstand

Tabelle 1: Freie GPIO-Pins auf der DSO138-Hardware

Der DLO-138-Sketch kann als Vorlage verwendet und entsprechend angepasst werden. In der Loop-Funktion wird der Aufruf von controlLoop() entfernt und stattdessen ein eigener Code eingefügt. Direkt in der Arduino IDE sind unter Datei → Beispiele → Beispiele für Generic STM32F103C series auch einige Beispielcodes zu finden.

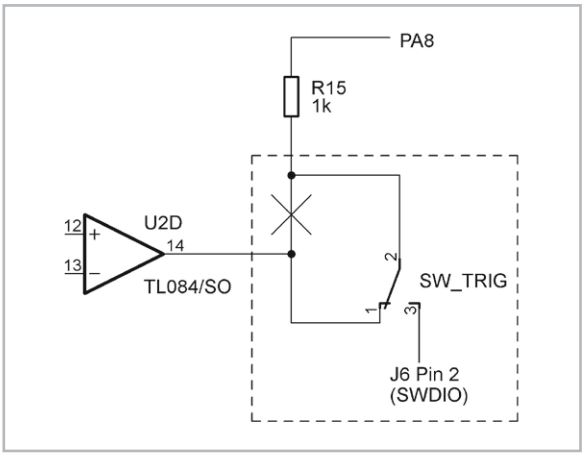


Bild 1: Schaltplan Umschalter Trigger-Quelle

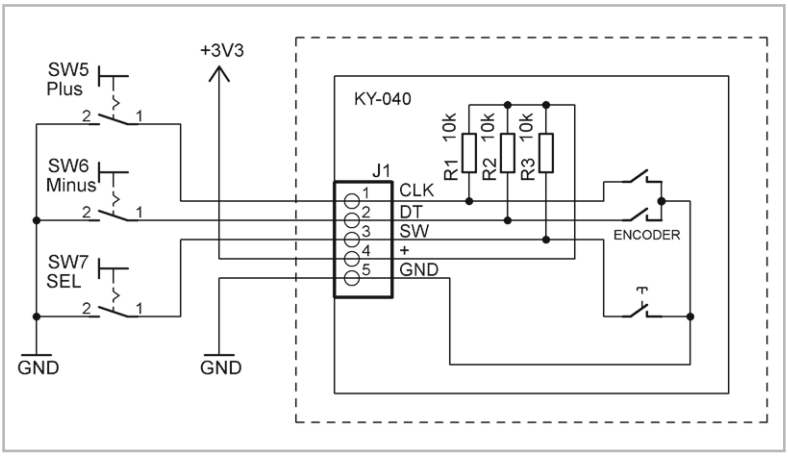


Bild 2: Schaltplan Dreh-Encoder

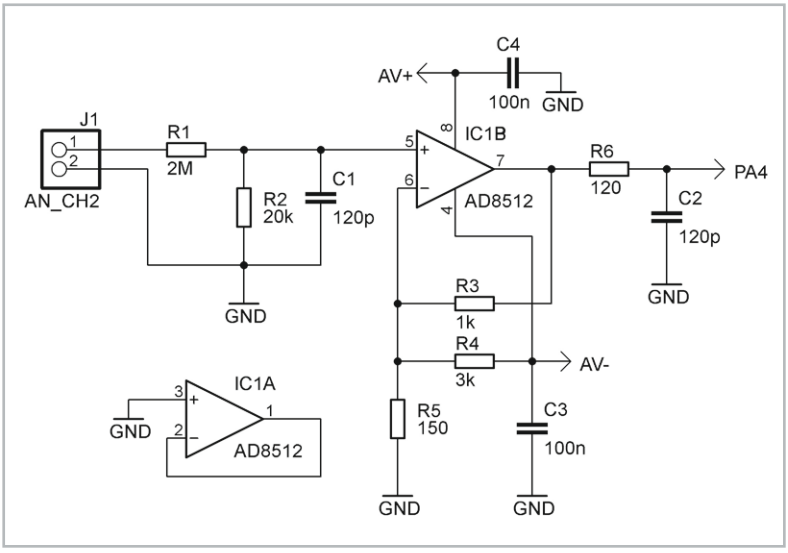


Bild 3: Schaltplan einfacher Messverstärker für den zusätzlichen Analogkanal 2

Die verwendeten GPIOs werden in der Funktion initIO in der Datei io.ino durch Aufruf von pinMode initialisiert. Hier sollten alle Initialisierungen der DLO-138-Vorlage, die nicht gewünscht sind, entfernt werden.

Um einen GPIO zu steuern, kann wie gewohnt digitalWrite() verwendet werden. Die Pin-Bezeichnungen orientieren sich an der Namensgebung des STM32, so wie sie auch im Schaltplan bezeichnet sind, beispielsweise PA15 für die LED. Hierfür gibt es auch die Makros LED_ON und LED_OFF, die direkt verwendet werden können.

Zum Lesen eines Pins kann digitalRead() verwendet werden. Um einen AD-Kanal mit analogRead() auslesen zu können, muss der Aufruf von setADC() in der Datei io.ino entfernt werden. Der entsprechende AD-Pin muss mit

```
pinMode(PAx, INPUT_ANALOG);
```

konfiguriert werden.

Die Tastenabfrage erfolgt per Interrupt, wodurch je nach Taste die Funktionen btn4ISR (OK), readESwitchISR (SEL), readASwitchISR (Minus) oder readBSwitchISR (Plus) aufgerufen werden. Die beiden letztgenannten befinden sich in encoder.ino, die anderen in interface.ino. Wenn ein Encoder verwendet wird, erfolgt bei einer Drehbewegung der Aufruf der Funktion encoderChanged(), diese liefert die Schritte.

Über die Klasse Serial können Daten über den virtuellen COM-Port per USB-CDC gesendet und empfangen werden, eine Initialisierung per Serial.begin() ist nicht erforderlich. Der USART1 (Portpins PA9 und PA10) des STM32F103C8 ist über die Klasse Serial1 ansprechbar und muss mit der Methode begin() mit der gewünschten Baudrate konfiguriert werden. Falls die USART-Pins stattdessen als GPIOs verwendet werden sollen, konfiguriert man die Pins wie gewünscht mit pinMode().

Die Displayansteuerung erfolgt über display.ino. Dort kann das Objekt tft verwendet werden. Beispielhaft werden einige Funktionsaufrufe gezeigt, um Text auszugeben, ein einzelnes Pixel zu setzen, eine Linie zu zeichnen oder ein Adressfenster aufzuziehen:

```
tft.setTextColor(ILI9341_WHITE, ILI9341_BLACK);
tft.setTextSize(3);
tft.setCursor(x, y);
tft.println(„TFT“);
tft.drawPixel(x, y, ILI9341_GREEN);
tft.drawLine(x1, y1, x2, y2, ILI9341_GREEN);
tft.setAddrWindow(x1, y1, x2, y2);
```

Der STM32F103C8 hat zwar offiziell nur 64 KB Flash, in der Praxis stehen bei den DSO138-Boards jedoch meistens 128 KB zur Verfügung, die eigentlich nur die CB-Variante haben sollte. In der Arduino IDE kann man dann als Variante STM32F103CB mit 128 KB Flash auswählen und hat somit mehr Platz im Flash, um beispielsweise kleine Grafiken ablegen zu können.

Die nachfolgend beschriebenen Sketches benötigen diesen zusätzlichen Speicherplatz. Es muss jedoch beachtet werden, dass in den letzten 2 KB des 128-KB-Flashbereichs die EEPROM-Daten abgelegt werden, falls die EEPROM-Library benutzt wird, z. B. mit EEPROM.write().

Die DLO-138-Firmware nutzt diesen EEPROM-Bereich ebenfalls. Wenn auf der Platine ein eigener Code vorhanden war und nach Aufspielen der DLO-138-Firmware dort in den Debugmeldungen auf dem Terminal „Unable to save param in EEPROM“ erscheint, dann muss der EEPROM-Bereich neu formatiert werden. Dies kann durch den Beispielsketch EEPROM_example.ino erreicht werden.

Der Sketch EEPROM_example.ino gibt ein Auswahlmenü auf dem seriellen Monitor aus und erwartet dann hierüber entsprechende Benutzereingaben. Über den Serial Monitor kann dann mit der Tasteneingabe 1 die Konfiguration mit 0x400 (1 KB große Flash-Sektoren) gesetzt werden und anschließend mit Tasteneingabe 7 neu formatiert werden.

Um Grafiken im Flash zu speichern, kann der Onlinekonverter unter [4] verwendet werden. Bei „Output Format“ wird „C array“ selektiert. Nach dem Auswählen einer Grafikdatei wird mit „Convert“ eine C-Datei mit mehreren Arrays mit verschiedenen Pixelformaten erzeugt. Hier wird RGB565 benötigt.

Dieses beginnt nach der Zeile

```
#if LV_COLOR_DEPTH == 16 && LV_COLOR_16_SWAP == 0
```

und endet vor dem nächsten #endif. Dieser Bereich wird herauskopiert und in display.ino eingefügt.

Im Beispiel wird die 130x100 Pixel große Datei elv_logo.png [1] konvertiert. Davor werden noch folgende Zeilen geschrieben, um ein Array anzulegen:

```
#define BMP_ELV_WIDTH 300
#define BMP_ELV_HEIGHT 130
const uint8_t bmp_Elv[BMP_ELV_WIDTH * BMP_ELV_HEIGHT * 2] =
{
```

Am Ende des Arrays wird mit }; abgeschlossen. Die Ausgabe der Grafik auf dem Display erfolgt durch unsere Beispielfunktion

```
drawBitmap(const uint8_t *pBmp, uint16_t x, uint16_t y, uint16_t width, uint16_t height)
```

die hier folgendermaßen aufgerufen wird:

```
drawBitmap(bmp_Elv, 9, 1, BMP_ELV_WIDTH, BMP_ELV_HEIGHT);
```

Hinweis: Alle Code-Beispiele aus diesem Beitrag finden Sie im Download-Bereich des Beitrags unter [1].

Es ist auch möglich, mehrere Grafiken im Flash abzulegen und anzuzeigen.

Der Beispielsketch elv_display_testapp.ino [1] zeigt das ELV Logo auf dem Display an und gibt zusätzlich einen Text aus. Nach 2 s wird das Display animiert gelöscht, und 1 s später wiederholt sich der Ablauf. Mit der SEL-Taste kann die Schriftfarbe verändert werden.

I²C-Bus-Zugriffe sind per Bitbanging mit zwei GPIOs möglich. Hierzu wird die beim Arduino_STM32 vorhandene SoftWire-Bibliothek verwendet. Im Sketch wird SoftWire.h inkludiert und ein Objekt der Klasse SoftWire angelegt. Mit begin() erfolgt die Initialisierung. Ein Schreibzugriff erfolgt mit write() und wird mit beginTransmission() und endTransmission() umrahmt. Ein Lesezugriff erfolgt mit requestFrom(), gefolgt von read()-Zugriffen.

Die Pins PA13 und PA14 bieten sich geradezu an, da sie zusammen mit 3V3 und GND auf einer 4-poligen Stiftleiste herausgeführt sind. Es gibt allerdings einen kleinen Schönheitsfehler: An PA14 (SWCLK) ist ein 10 kΩ-Pull-down vorhanden, der für I²C nicht geeignet ist. Dieser muss entfernt werden. Da es sich um eine SMD-Ausführung handelt, lötet man zuerst beide Pads mit frischem Lötzinn nach, um dann anschließend durch abwechselndes Erwärmen den Widerstand vorsichtig mit der Lötspitze vom Lötpad herunterzuschieben. Meist sind an den I²C-Sensormodulen Pull-up-Widerstände vorhanden, falls nicht, muss an PA14 ein externer Pull-up angeschlossen werden.

Es können auch die freien Pins PA4 und PA5 benutzt werden, die als Testpunkte unterhalb des Displays auf der Platinenoberseite vorhanden sind. Da PA4 in der DLO-138-Vorlage als Analogkanal 2 verwendet wurde, muss dann jedoch der Define AN_CH2 in global.h umbelegt oder einfach auf AN_CH1 gemappt werden.

Der Beispielsketch elv_i2c_testapp.ino verwendet PA14 als SCL und PA13 als SDA. Die Temperatur und relative Luftfeuchte wird mit einem HTU21D-Sensormodul gemessen und jede Sekunde auf dem Display angezeigt. Es gibt eine Vielzahl von I²C-Sensoren auf dem Markt, sodass per I²C-Bus viele physikalische Messgrößen erfasst und angezeigt werden können.

STM32-BluePill-Boards

Eine sehr bekannte günstige Platine mit STM32-Mikrocontroller ist das BluePill-Board (Bild 4). Dort ist meistens ebenfalls ein STM32F103 verbaut, wobei es welche mit 64 KB Flash (Controller-Variante STM32F103C8) und 128 KB Flash (Variante CB) gibt. Auch eine Variante C6 mit lediglich 32 KB Flash wird angeboten.

Neben der Power-LED ist noch eine User-LED am Pin PC13 vorhanden, die gegen Vcc geschaltet ist, also eine Low-aktive Ansteuerung benötigt. Manche Board-Varianten verwenden auch einen anderen Pin für die LED. Auf dem Board ist auch eine USB-Buchse vorhanden, die zum einen der Spannungsversorgung dient, zum anderen aber auch softwaremäßig genutzt werden kann. Im Auslieferzustand ist jedoch kein USB-Bootloader installiert, sodass das Flashen zunächst nur über den eingebauten ROM-Bootloader per UART möglich ist oder aber über das SWD-Interface mithilfe eines speziellen Programmers wie z. B. des ST-Link V2.

Praktischerweise sind bereits zwei Stiftleisten mit Jumpern sowie ein Reset-Taster auf dem BluePill-Board vorhanden. Um den Bootloader-Modus zu aktivieren, muss der Jumper BOOT0 auf die Position 1 umgesteckt werden. Der andere Jumper BOOT1 kann dauerhaft in Stellung 0 verbleiben.

Einen passenden Bootloader findet man unter [5], wobei wie beim DSO138-Bootloader zusätzlich eine Applikation enthalten ist, die Begrüßungsmeldungen auf dem USB-CDC ausgibt. Im Dateinamen steht am Ende der Port-Pin für die User-LED, d. h., es muss je nach Board die passende Datei gewählt werden. Für das Board mit der LED an PC13 ist die passende Datei also [6].

Mit dem STM32CubeProgrammer kann dieser Bootloader über ein USB-UART-Modul auf das BluePill-Board geladen werden, analog wie im ELVjournal 5/2020 beim DSO138 beschrieben (Abschnitt: DLO-138-Firmware auf DSO138 laden, die Pins für RX1/TX1 finden sich im DSO138-Schaltplan unter [3]) .

Anschließend wird die Platine vom USB-Port getrennt und der BOOT0-Jumper wieder auf Stellung 0 gesteckt.

Nach dem Wiederanstecken des Boards sollte im Gerätemanager ein weiterer COM-Port vorhanden sein, vorausgesetzt der Maple-Treiber wurde wie im vorhergehenden Beitrag beschrieben installiert.

Falls kein Gerät am USB-Port erkannt wird, sollte man prüfen, ob der Pull-up-Widerstand am Signal D+ des USB-Interfaces auf der Platine den korrekten Wert von 1,5 kΩ hat, und ansonsten diesen tauschen, da manche BluePill-Boards mit einem falschen Typ (z. B. 4,7 kΩ oder 10 kΩ) bestückt sind.

Einen Schaltplan findet man z. B. auf [7], dort ist der Pull-up mit R10 bezeichnet und fälschlicherweise mit 4,7 kΩ statt den korrekten 1,5 kΩ angegeben.

In der Arduino IDE arbeiten wir dieses Mal mit dem offiziellen STM32duino-Core. Unter Werkzeuge werden folgende Einstellungen vorgenommen (Bild 5):

- Board: „STM32 Boards" → „Generic STM32F1 series"
- Board part number: „Blue Pill F103C8" bzw. das passende Board je nach Variante

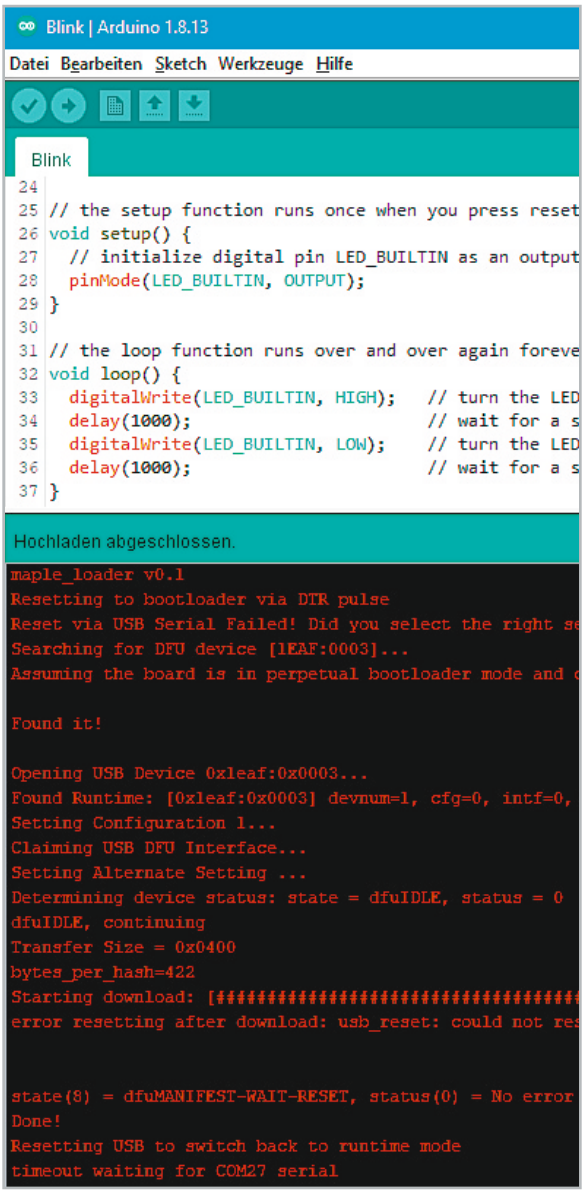


Bild 6: Hochladen des Blink-Sketches

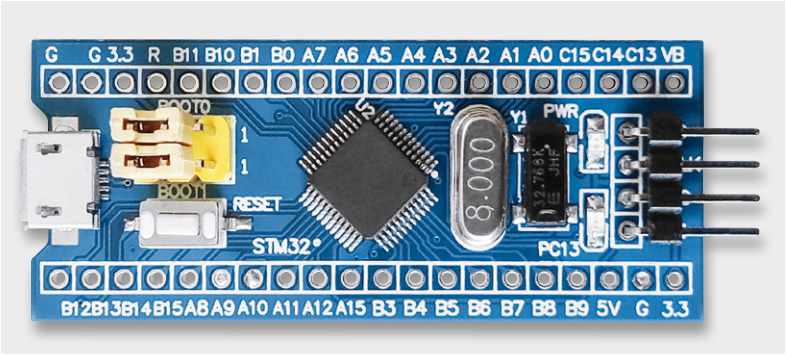


Bild 4: BluePill-Board

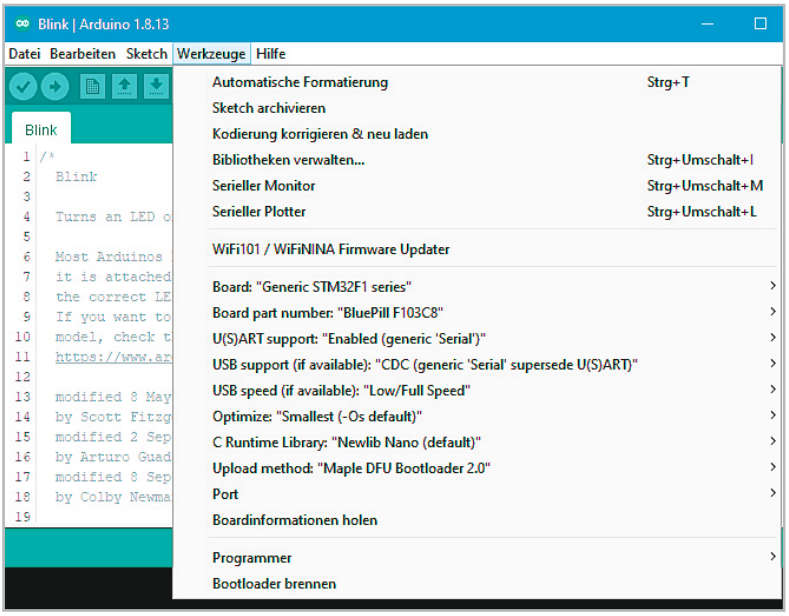


Bild 5: Einstellungen zur Programmierung des BluePill-Boards mit dem STM32duino-Core

- USB support: CDC (generic Serial supersede USART)
- Upload method: Maple DFU Bootloader 2.0
- Port: passender COM-Port

Für einen ersten Test kann dann unter Datei → Beispiele → 01 Basics der Blink-Sketch ausgewählt und kompiliert werden. In der Arduino IDE kann dann wie gewohnt über Sketch → Hochladen das Blink-Programm übertragen werden (Bild 6), sodass die LED auf dem Board ihren Dienst tun sollte.

Nach dem Hochladen des ersten Sketches kann es sein, dass Windows eine neue COM-Portnummer vergibt und deshalb im Ausgabefenster der Arduino IDE eine Timeout-Meldung angezeigt wird. In diesem Fall muss unter Werkzeuge → Port die Einstellung angepasst werden.

Im nächsten Schritt soll die LED jede Sekunde kurz aufblitzen und zusätzlich eine UART-Ausgabe erfolgen. Dies ist über die USB-CDC-Schnittstelle (angesprochen über „Serial") oder aber auch über einen der USARTs des STM32 (Serial1, Serial2 usw.) möglich.

Im Beispiel wird sowohl die USB-CDC als auch der USART1 benutzt. Im Sketch wird in der Setup-Funktion folgende Zeile eingefügt:

```
Serial1.begin(115200);
```

Dadurch wird der USART1 mit einer Baudrate von 115.200 Baud initialisiert. Für die USB-CDC ist dies nicht notwendig.

In der Loop-Funktion werden die Delay-Zeiten angepasst und folgende Zeilen eingefügt:

```
Serial.println("Hello STM32: USB");
Serial1.println("Hello STM32: UART");
```

Nach dem Hochladen des Sketches sollte im Arduino-Serial-Monitor jede Sekunde die Meldung „Hello STM32: USB“ ausgegeben werden (hier im Beispiel auf COM28, Bild 7) und zusätzlich die LED kurz aufblitzen. Um die Ausgabe „Hello STM32: UART“ des USART1 zu sehen, muss ein USB-UART-Modul (z. B. UM2102N) an die Pins PA9 (TxD-Ausgang) und PA10 (RxD-Eingang) der Blue Pill angeschlossen werden und ein zweiter Arduino-Serial-Monitor mit dem entsprechenden COM-Port mit einer Baudrate von 115200 Baud verbunden werden. Im Screenshot ist dies COM14.

Hinweis: Alle Code-Beispiele aus diesem Beitrag finden Sie im Download-Bereich des Beitrags unter [1].

Arduino mit STM8: Sduino

Die 8-Bit-Mikrocontrollerfamilie STM8 wird in einigen ELV Bausätzen eingesetzt. Für die Standardbaureihe STM8S gibt es eine Arduino-Erweiterung namens Sduino unter [8].

Der verwendete freie Compiler ist der bekannte SDCC. Dieser kann jedoch nur C-Code kompilieren und keinen C++-Code. Aus diesem Grund ist die Syntax von Bibliotheksfunktionen in vielen Fällen anders als beim Arduino gewohnt. So erfolgt z. B. die Ausgabe eines Textes auf der seriellen Schnittstelle mit dem Befehl Serial_println_s anstatt Serial.println().

Um einmal in diese Materie hineinzuschnuppern, bietet sich beispielsweise ein günstiges Board mit dem STM8S103F3 an.

Der Controller wird mit dem internen 16-MHz-Oszillator getaktet und verfügt über 8 KB Flash, 1 KB RAM sowie 640 Bytes EEPROM onchip. Die Micro-USB-Buchse dient nur zur Spannungsversorgung. Neben der Power-LED ist noch eine User-LED am Pin PB5 vorhanden, die gegen Vcc geschaltet ist, also eine Low-aktive Ansteuerung benötigt.

Leider hat dieser Controller keinen eingebauten Bootloader im ROM. Um den Flash-Speicher programmieren zu können, wird ein Programmieradapter mit dem SWIM-Interface benötigt. Hierzu kann ein günstiger ST-Link V2 (Bild 8) verwendet werden, der auch als Clone im USB-Stick Format erhältlich ist.

Das Breakout-Board hat bereits passend vorgesehene Lötaugen für eine 4-polige Stiftleiste mit den SWIM-Signalen. Achtung: Im Gegensatz zum Original ST-Link V2 gibt der Clone am 3V3-Pin aktiv eine Spannung von 3,3 V aus. Wenn das STM8-Board per USB-Buchse versorgt wird, darf beim Clone VCC am SWIM-Interface nicht verbunden werden, da ansonsten die Spannungsregler beider Boards gegeneinandertreiben.

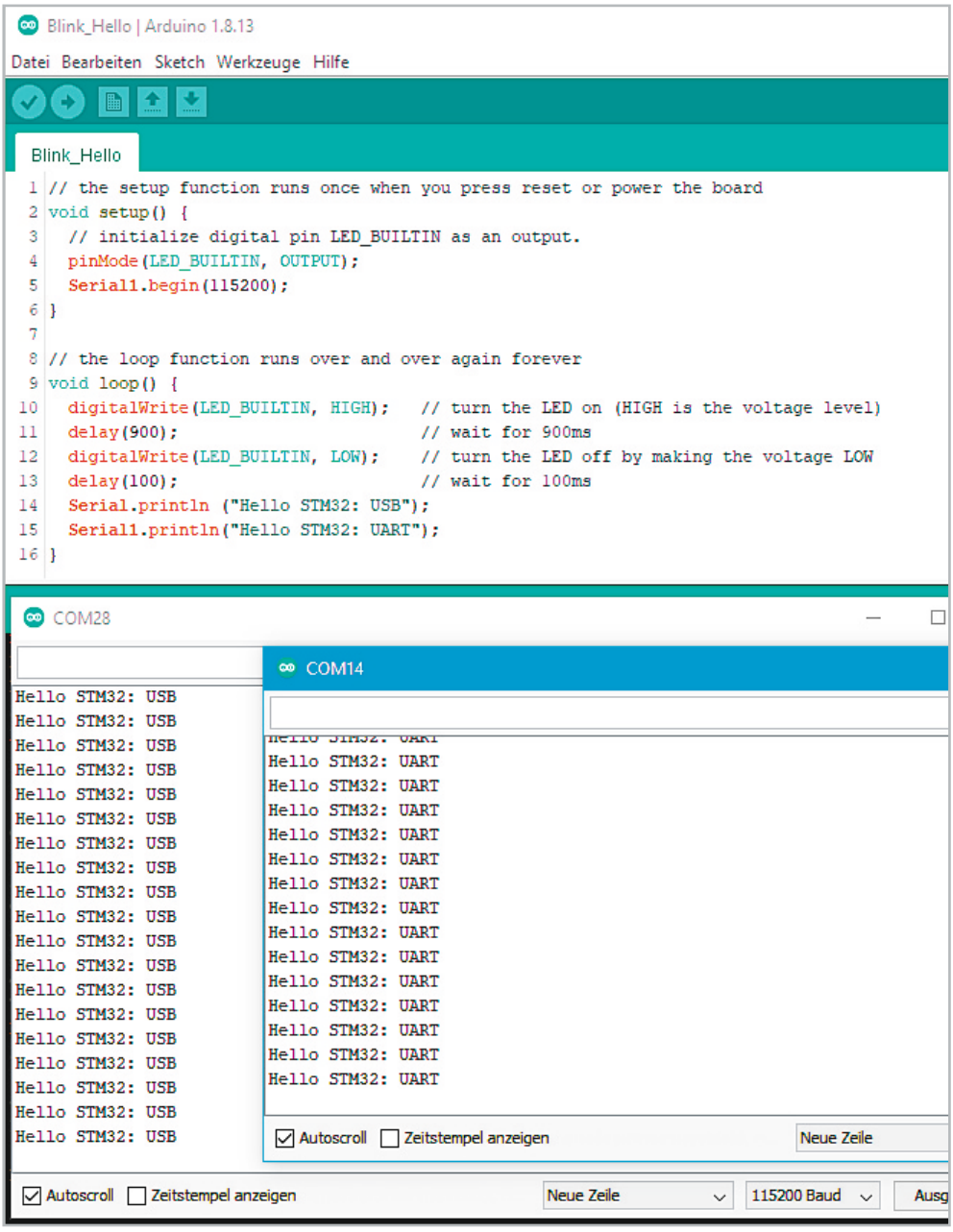


Bild 7: Ausgabe im seriellen Monitor

Um die Sduino-Erweiterung zu installieren, trägt man in der Arduino IDE unter Datei → Voreinstellungen → „Zusätzliche Boardverwalter-URLs“ folgende Adresse ein (mehrere Einträge können per Komma getrennt werden):

https://github.com/tenbaht/sduino/raw/master/package_sduino_stm8_index.json

Anschließend wird der Dialog mit OK beendet. Unter Werkzeuge → Board → Boardverwalter wird oben in das Suchfeld „Sduino“ eingegeben. Auf diese Weise sollte das Board „Sduino“ gefunden werden (Bild 9).

Nach der Installation wählt man unter Werkzeuge → Board bei „STM8S Boards“ den Eintrag „STM8SF103F3 breakout board“ aus und unter Board → Upload method den Eintrag ST-Link V2 (Bild 10).

Bei vielen dieser Boards ist im Auslieferzustand bereits ein LED-Blinkprogramm vorhanden und zusätzlich erstaunlicherweise der Ausleseschutz des STM8-Controllers gesetzt. Dieser muss zuerst entfernt werden, bevor eigene Programme aufgespielt werden können (Bild 11).

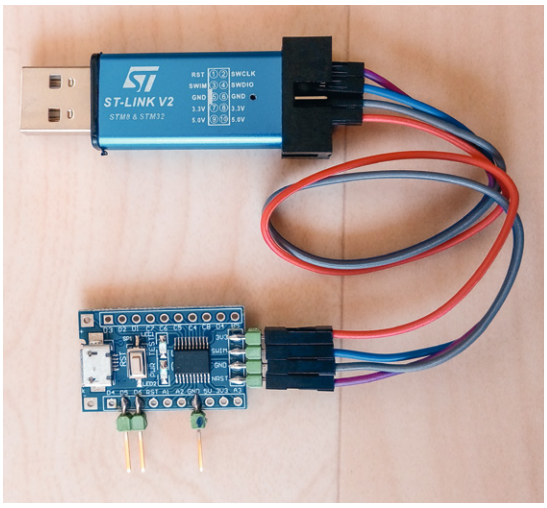


Bild 8: Programmieren des STM8 mittels Programmieradapter

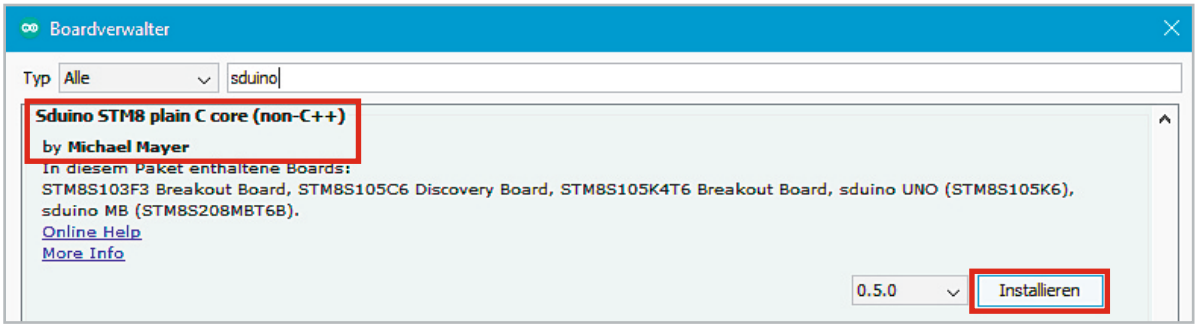


Bild 9: Sduino-Paket in der Arduino IDE

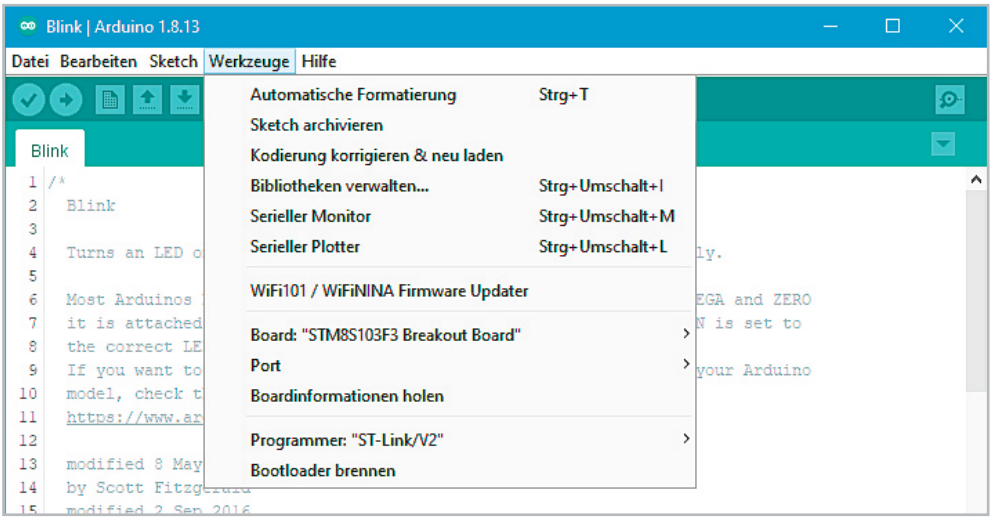


Bild 10: Voreinstellungen für Sduino

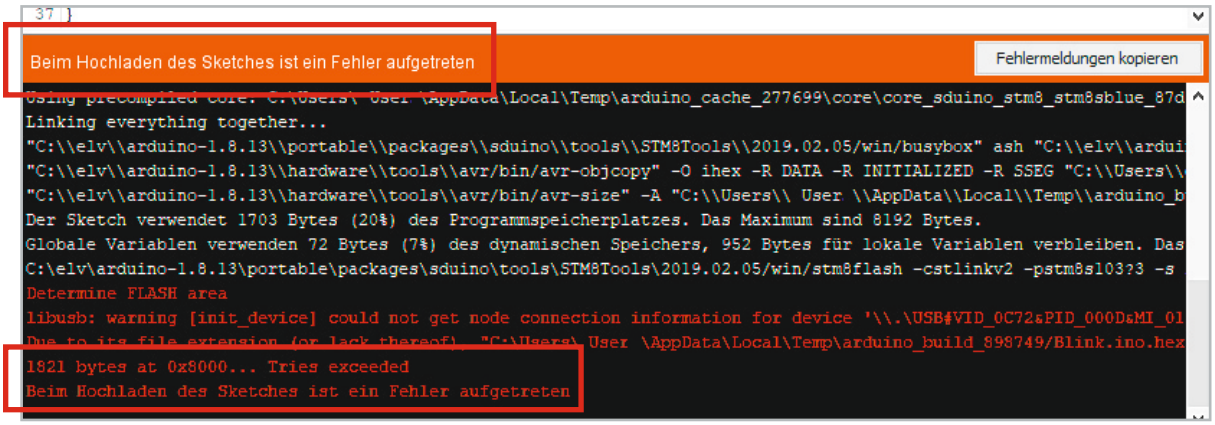


Bild 11: Mit gesetztem Ausleseschutz ist kein Hochladen möglich

Hierzu muss man lediglich in der Arduino IDE unter Werkzeuge auf „Bootloader brennen“ klicken (Bild 12). Die Bezeichnung passt zwar nicht ganz, aber der Ausleseschutz wird dadurch entfernt. Etwaige Warnmeldungen kann man ignorieren.

Für einen ersten Test kann dann unter Datei → Beispiele → 01 Basics der Blink-Sketch ausgewählt und kompiliert werden. Falls noch kein Treiber für den ST-Link auf dem PC vorhanden ist, kann dieser unter [9] heruntergeladen werden.

In der Arduino IDE kann dann wie gewohnt über Sketch → Hochladen das Blink-Programm übertragen werden, sodass die LED auf dem Board ihren Dienst tun sollte (Bild 13).

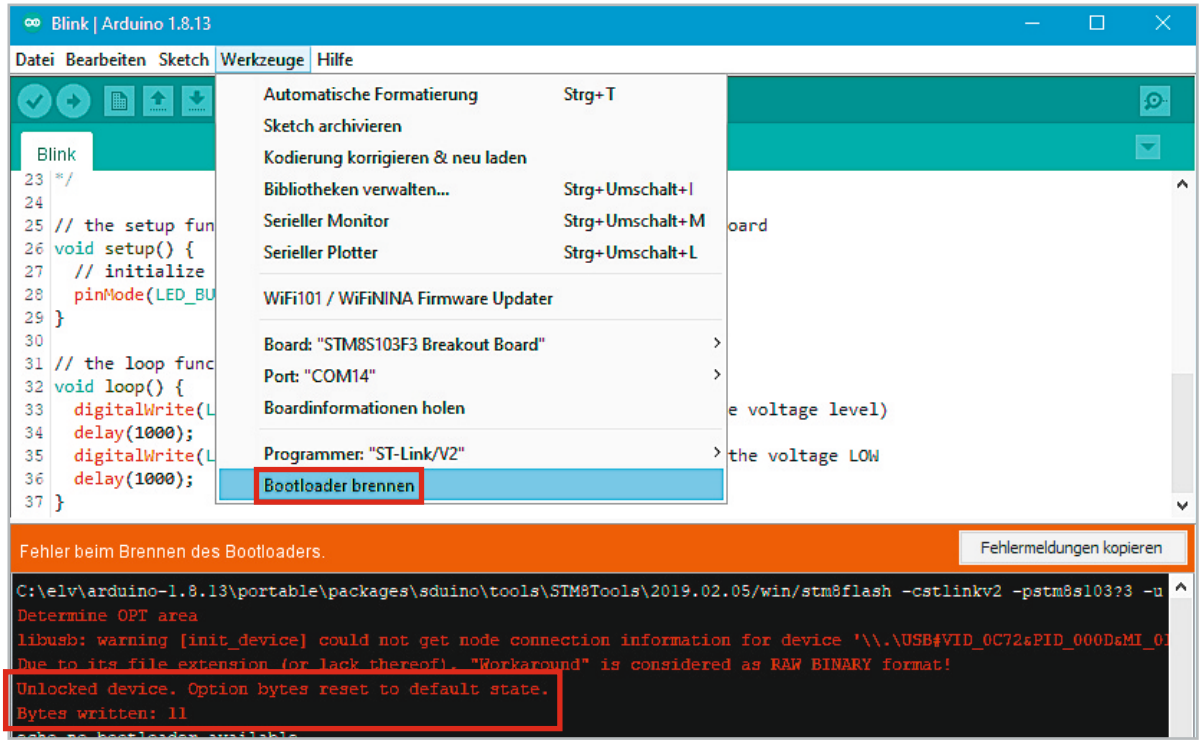


Bild 12: Durch Brennen des Bootloaders wird der Ausleseschutz deaktiviert.

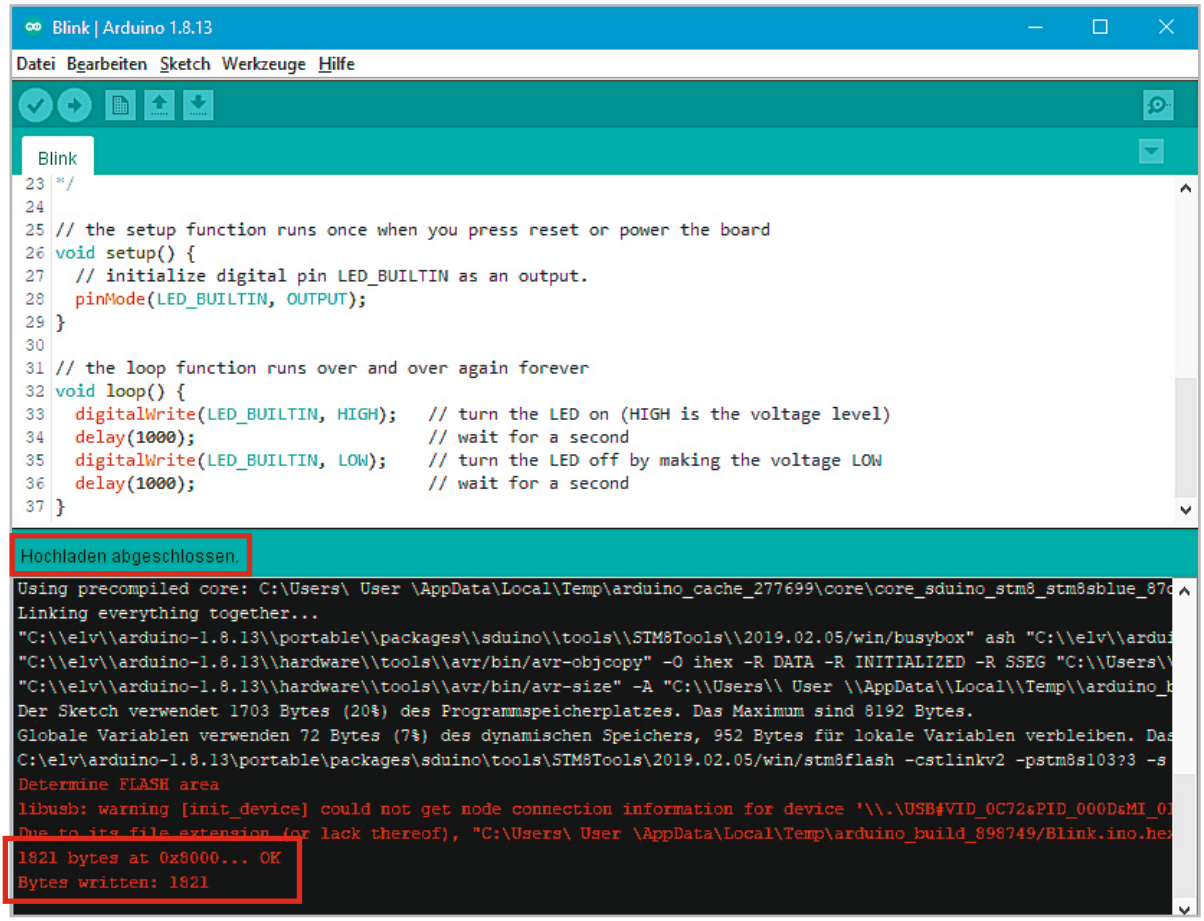


Bild 13: Hochladen des Blink-Sketches

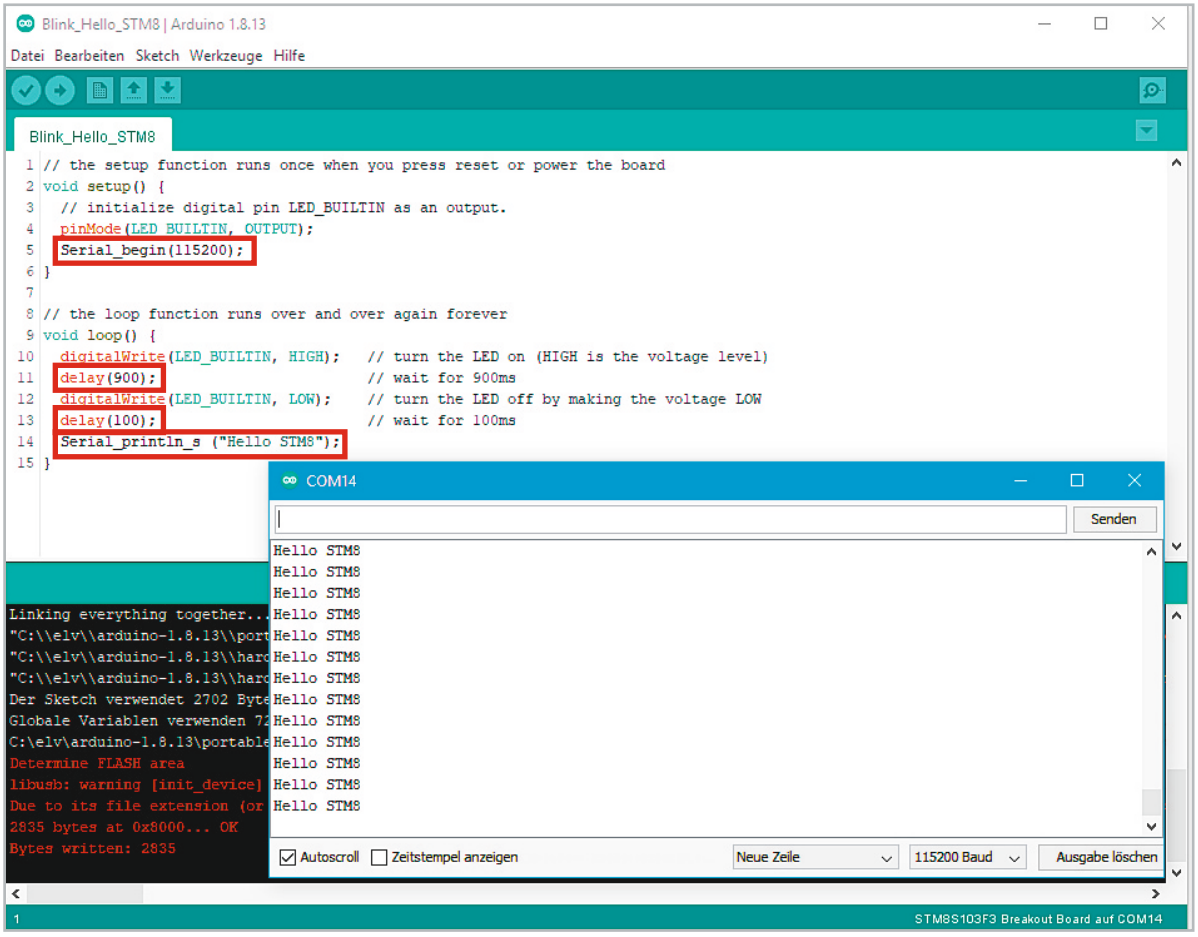


Bild 14: Ausgabe von Text per UART-Modul im seriellen Monitor

Im nächsten Schritt soll die LED jede Sekunde kurz aufblitzen und zusätzlich eine UART-Ausgabe erfolgen. Die entsprechenden Pins sind auf [10] unter der Überschrift „Pin number mappings" zu finden. PD5 ist der TXD-Ausgang und PD6 der RXD-Eingang des STM8S-Boards. Diese können passend mit RXD und TXD des UM2102N-USB-UART-Moduls verbunden werden. Im Sketch wird in der Setup-Funktion folgende Zeile eingefügt:

```
Serial_begin(115200);
```

In der loop-Funktion werden die Delay-Zeiten angepasst und folgende Zeile eingefügt:

```
Serial_println_s("Hello STM8S");
```

Unter Werkzeuge → Port muss der passende COM-Port des USB-UART-Moduls eingetragen werden. Nach dem Hochladen des Sketches sollte im Arduino-Terminal jede Sekunde die Meldung „Hello STM8S" ausgegeben werden und zudem weiterhin die LED blinken (Bild 14).

Fazit

Mit unserer Einführung in die Programmierung von STM32/STM8-Controllern von STMicroelectronics per Arduino IDE haben wir einen einfachen Einstieg vor allem für Nutzer geschaffen, die mit dieser Entwicklungsumgebung bereits vertraut sind. So ist recht einfach die Möglichkeit gegeben, Geräte mit Controllern aus dieser Familie – wie wir das beispielhaft mit dem Einsteiger-Oszilloskop DSO138 und den STM32/STM8-Entwicklerboards demonstriert haben – mit eigenem, angepasstem oder verbessertem Code auszustatten. Auch Hardware-Modifikationen sind durch Firmware-Anpassungen möglich. Im Internet finden sich zudem weitere Open-Source-Beispiele für STM32/STM8-Mikrocontroller. ELV

Weitere Infos:

[1] Fachbeitrag im ELVshop (Download-Bereich) unter Artikel-Nr. 251816
[2] Dreh-Encoder KY-040: Artikel-Nr. 250447
[3] Schaltplan DSO138: Download-Bereich unter Artikel-Nr. 127893
[4] Onlinekonverter: https://lvgl.io/tools/imageconverter
[5] Bootloader BluePill-Board: https://github.com/rogerclarkmelbourne/STM32duino-bootloader/tree/master/binaries
[6] Bootloader für Board mit der LED an PC13: https://github.com/rogerclarkmelbourne/STM32duinobootloader/blob/master/binaries/generic_boot20_pc13.bin
[7] Schaltplan BluePill-Board: https://cdn.shopify.com/s/files/1/1509/1638/files/STM32F103C8T6_Mikrocontroller_Schematics.pdf?17591767635849911504,
[8] Arduino-Erweiterung Sduino: https://github.com/tenbaht/sduino
[9] Treiber ST-Link: https://www.st.com/en/development-tools/stsw-link009.html
[10] Pin number mappings: https://tenbaht.github.io/sduino/hardware/stm8blue/

Alle Links finden Sie auch online unter: de.elv.com/elvjournal-links