Bausatz-Artikel-Nr.: 160856
Version: 1.0
Stand: November 2024

ELV Basismodul Mikrocontroller

ELV-BM-MCU

Bitte lesen Sie die Bau- und Bedienungsanleitung vor der Inbetriebnahme komplett und bewahren Sie sie für späteres Nachlesen auf. Wenn Sie das Gerät anderen Personen zur Nutzung überlassen, übergeben Sie auch diese Bau- und Bedienungsanleitung.

Kontakt:
Sie haben Fragen zum Produkt oder zur Bedienung, die über die Bau- und Bedienungsanleitung nicht geklärt werden konnten?
Sie haben eine Reklamation zu Ihrem Gerät?

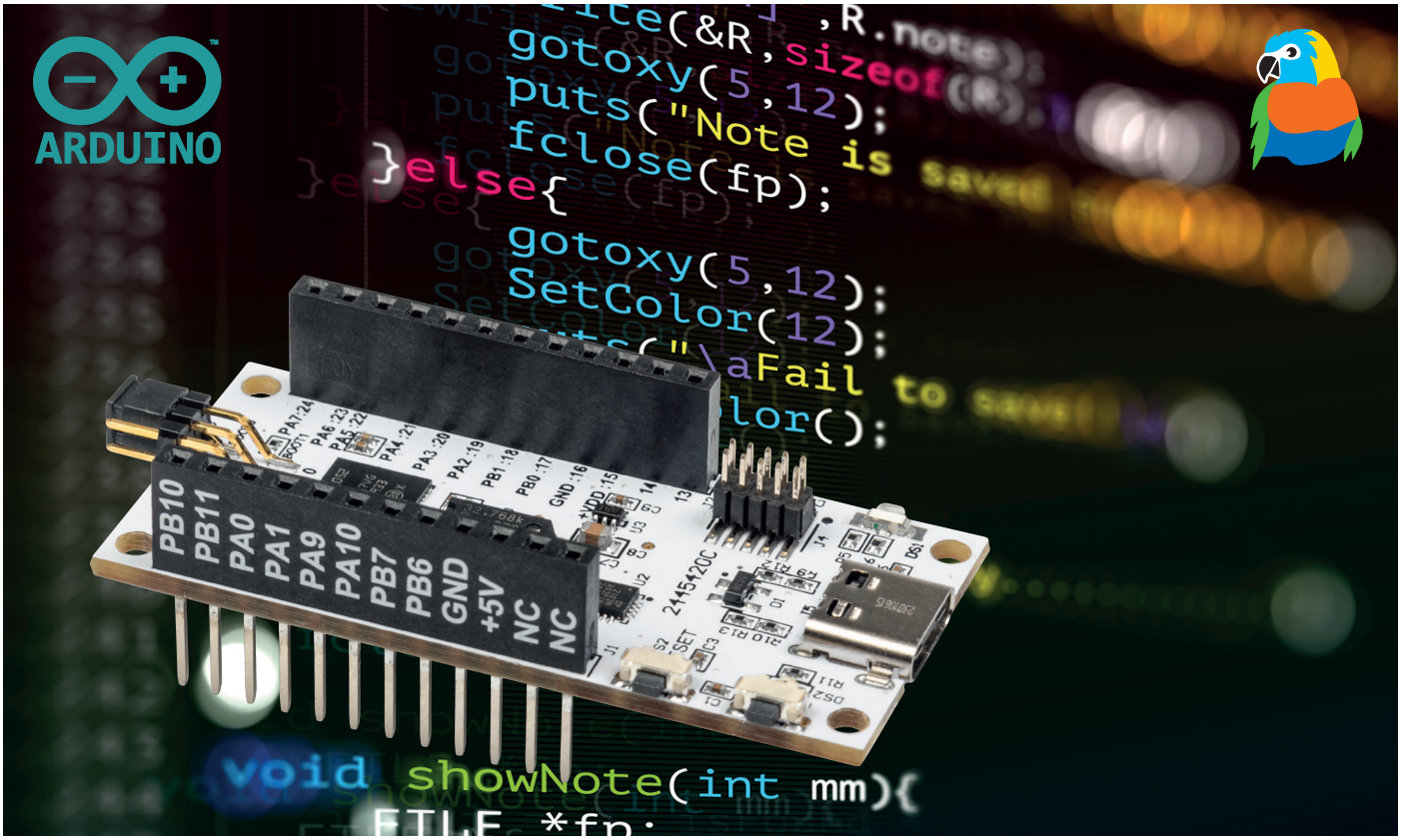
Kontaktieren Sie unser Team gerne über unsere Homepage www.elv.com im Bereich Service und Kontakt.

Häufig gestellte Fragen und aktuelle Hinweise zum Betrieb des Produkts finden Sie zudem bei der Artikelbeschreibung im ELVshop.

Reparaturservice
Für Geräte, die aus ELV Bausätzen hergestellt wurden, bieten wir unseren Kunden einen Reparaturservice an. Selbstverständlich wird Ihr Gerät so kostengünstig wie möglich instand gesetzt. Im Sinne einer schnellen Abwicklung führen wir die Reparatur sofort durch, wenn die Reparaturkosten den halben Komplettbausatzpreis nicht überschreiten. Sollte der Defekt größer sein, erhalten Sie zunächst einen unverbindlichen Kostenvoranschlag.

Bitte senden Sie Ihr Gerät an: ELV · Reparaturservice · Maiburger Straße 29–36 · 26787 Leer · Germany

ELV Elektronik AG · Maiburger Straße 29–36 · 26789 Leer · Germany
Telefon (+49)491/6008-88 · www.elv.com



Aus Ideen mehr machen!

Einfacher Einstieg in die Welt der Mikrocontroller-Programmierung mit dem ELV-BM-MCU

Das ELV Basismodul Mikrocontroller ELV-BM-MCU bietet volle Kompatibilität zum beliebten ELV-Modulsystem, flexible Anschlussmöglichkeiten und eine hohe Energieeffizienz – perfekt für innovative DIY-Projekte und smarte Anwendungen!

In Zeiten, in denen Technologie und Kreativität Hand in Hand gehen, wird es immer wichtiger, Werkzeuge zu finden, die sowohl flexibel als auch vielseitig genug sind, um innovative Ideen Wirklichkeit werden zu lassen. Ob man als Tüftler, Entwickler oder Technikenthusiast an neuen Projekten und Ideen arbeitet – es braucht Lösungen, die einfach zu bedienen sind, aber gleichzeitig genügend Spielraum für komplexe Anwendungen bieten. Doch wie schafft man es, die eigenen Visionen auf das nächste Level zu heben? Diese Frage beantwortet das ELV Basismodul Mikrocontroller. Der verbaute STM32L052-Mikrocontroller bietet eine beeindruckende Kombination aus Leistungsfähigkeit und Energieeffizienz, die ihn besonders für batteriebetriebene und mobile Projekte attraktiv machen. Im Vergleich zu vielen anderen Mikroprozessoren zeichnet er sich durch einen extrem niedrigen Stromverbrauch aus, ohne dabei an Funktionalität einzubüßen.

Infos zum Bausatz ELV-BM-MCU

- **Schwierigkeitsgrad:** leicht
- **Bau-/Inbetriebnahmezeit:** ca. 0,5 h
- **Besondere Werkzeuge:** keine
- **Löterfahrung:** nein
- **Programmierkenntnisse:** ja
- **Elektrofachkraft:** nein

Vielfältige Schnittstellen

Der STM32L052 bietet zahlreiche Schnittstellen, die ihn zum ELV-Modulsystem kompatibel machen. Dazu gehören u. a. UART, I²C und SPI, was ihn ideal für Projekte macht, die die Kommunikation mit Modulen des Systems, aber auch mit anderen Geräten und Sensoren erfordern. Trotz seiner kompakten Bauweise liefert der Mikroprozessor genug Rechenleistung, um auch anspruchsvollere Aufgaben zu bewältigen.

Die Programmierung des ELV Basismoduls Mikrocontroller erfolgt über die Arduino-Entwicklungsumgebung, die eine intuitive Oberfläche und zahlreiche Ressourcen bietet, um schnell in die Entwicklung einzusteigen. Außerdem steht sowohl hinter der STM32duino-Board-Bibliothek als auch hinter der Arduino-Entwicklungsumgebung eine große Community an Entwicklern und Enthusiasten, die ständig daran arbeiten, dass beide Komponenten immer besser werden. Insgesamt bietet das ELV-BM-MCU eine leistungsstarke und vielseitige Plattform, die besonders durch Energieeffizienz und Kompatibilität hervorsticht. Auf die einzelnen Bestandteile wird im Folgenden ausführlich eingegangen.

Schaltungsbeschreibung

Spannungsversorgung

Die Spannungsversorgung der ELV-MCU-Base erfolgt über die USB-Buchse J5 (USB 3.1, Typ C, 5 V). Der USB-UART-Wandler CP2102 (U2) dient an dieser Stelle (neben seiner Hauptfunktion, siehe Konnektivität) als Spannungsregler, der die vom USB-Bus kommenden 5 V auf 3,3 V (typ.) herunterregelt. Mit dieser Spannung kann der Mikrocontroller U1 betrieben werden.

Um zu verhindern, dass eine fehlerhafte Spannung den UART-Wandler beschädigt, ist zwischen diesem und dem Mikrocontroller eine ideale Diode verbaut (U3). Dieses Bauteil sperrt, wenn an Pin 6 (VOUT) eine höhere Spannung anliegt als an Pin 1 (VIN). Vorteil gegenüber einer „echten" Diode ist, dass über dieses Bauteil nur eine Spannung von etwa 0,02 V abfällt und nicht wie bei einer Silizium-Diode von 0,7 V.

Die Kondensatoren C7 bis C17 fangen Spannungsspitzen auf und puffern die Versorgungsspannung.

Mikrocontroller

Der namensgebende Mikrocontroller des ELV Basismoduls ist ein STM32L052 (U1). Dieser Mikrocontroller des Herstellers STMicroelectronics passt sich aufgrund seiner hervorragenden Low-Power-Eigenschaften und der einfachen Benutzbarkeit sehr gut in die Reihe der restlichen Basismodule des ELV-Modulsystems ein. Er besitzt eine Flash-Speichergröße von 64 KB und einen 8 KB großen RAM und wird von einem 32,768-kHz-Oszillator (Y1) mit einem externen Clock-Signal versorgt. Hiermit ist die Verwendung einer Real-Time-Clock (RTC) möglich. Die Kapazitäten C4 und C5 sorgen dafür, dass dieser Oszillator-Quarz anschwingen kann.

Details können dem Schaltbild des ELV-BM-MCU in Bild 1 und den Platinenfotos mit den zugehörigen Bestückungsdrucken in Bild 2 entnommen werden.

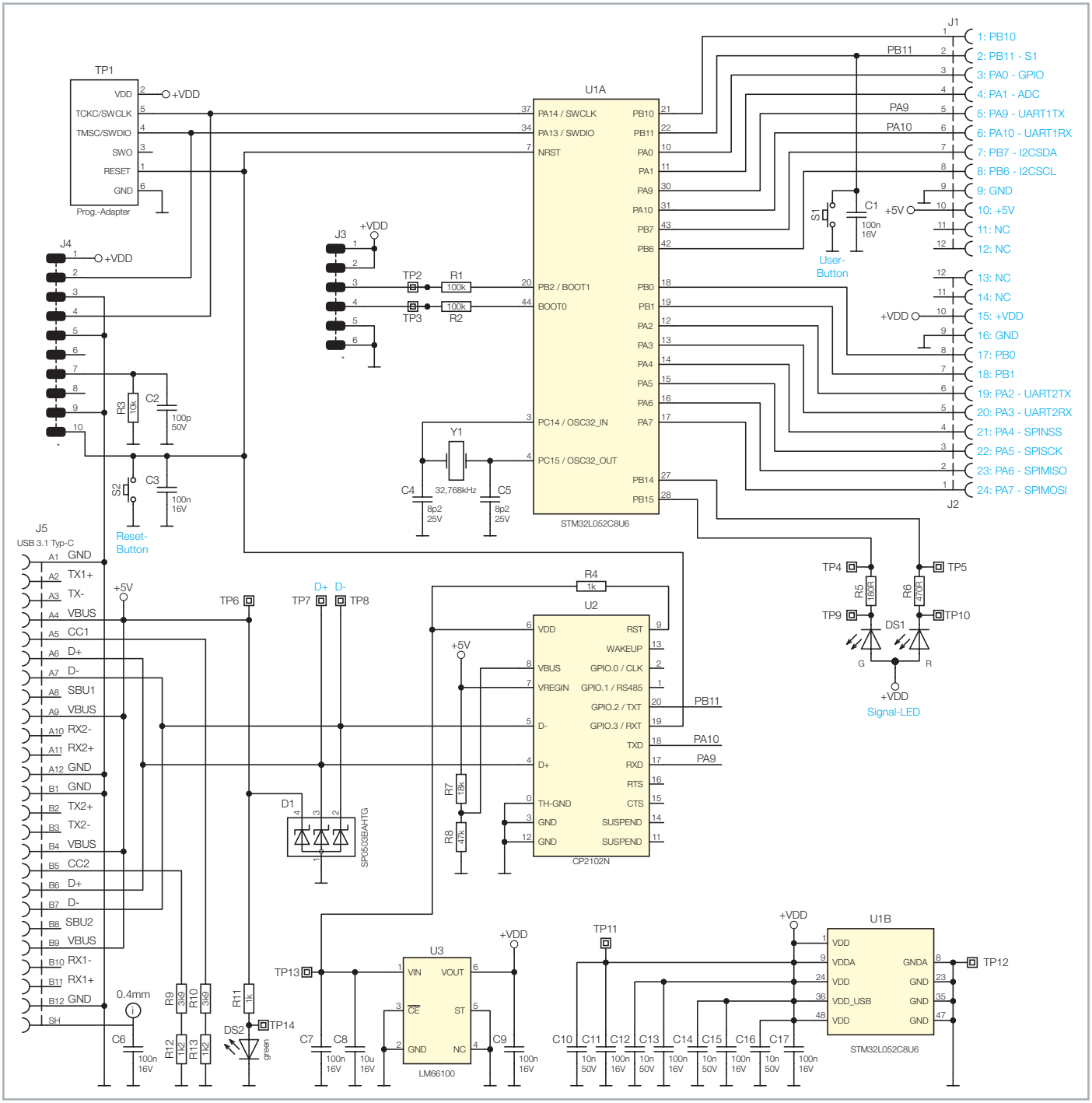


Bild 1: Schaltbild des ELV Basismoduls Mikrocontroller Unit ELV-BM-MCU

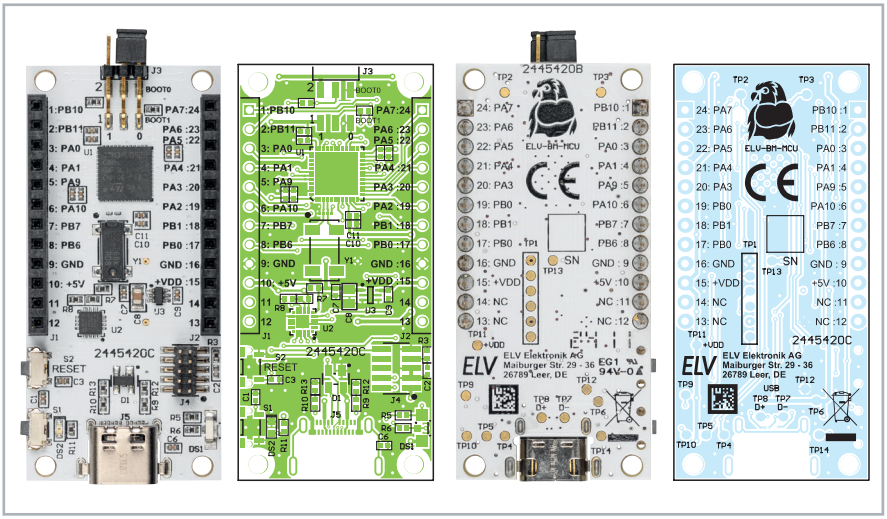


Bild 2: Die Platinen mit den zugehörigen Bestückungsdrucken

Konnektivität

Eine ganze Reihe an Pins des Mikrocontrollers ist mit den Stift-Buchsenleisten J1 und J2 verbunden. Hierbei sind einige Pin-Gruppen besonders hervorzuheben. Pin 5 und Pin 6 sind mit Pin PA9 und Pin PA10 des Mikrocontrollers verbunden und dienen standardmäßig als UART-Schnittstelle.

VORSICHT: Wenn Sie Pin PA9 und PA10 umkonfigurieren, ist keine Kommunikation mehr über die USB-Buchse mit dem Mikroprozessor möglich. Das bedeutet, dass auch über die serielle Schnittstelle keine Firmware mehr übertragen werden kann. Ein Neubespielen des Mikrocontrollers ist dann nur noch über die Programmierschnittstelle J4 möglich. Hierzu ist ein ST-Link-Programmieradapter notwendig.

Diese Pins sind ebenfalls mit dem UART-USB-Wandler verbunden und ermöglichen somit die Kommunikation mit dem Gerät über ein am Computer angeschlossenes USB-Kabel über einen virtuellen COM-Port.

Die Applikationsmodule des ELV-Modulsystems sind immer so konfiguriert, dass die I²C-Schnittstelle auf Pin 7 (PB7-I2CSDA) und Pin 8 (PB8-I2CSCL) der Stiftleisten liegen. Die Pins des Mikrocontrollers, die mit diesen Pins auf der Stiftleiste verbunden sind, sind als Standard-Pins des I²C-Bausteins des STM32L052 konfiguriert. Zur Kommunikation mit anderer Hardware über eine serielle Schnittstelle können Pin 19 (PA2-UART2TX) und Pin 20 (PA3-UART2RX) verwendet werden.

Zu guter Letzt gibt es noch ein weiteres gängiges Bussystem mit dem Namen SPI. Für dieses System sind die Pins 21–24 vorgesehenen. Die bidirektionalen Busleitungen MISO und MOSI liegen auf den Pins 23 und 24, die Taktleitung auf Pin 22 und der dazugehörige Chip-Select auf Pin 21. Pin 2 ist direkt hardwareseitig mit dem User-Button S1 verbunden. Direkt neben dem User-Button befindet sich der Reset-Button S2. Dieser ist nicht mit der Stiftleiste verbunden, sondern direkt mit dem nReset-Pin (NRST) des Mikrocontrollers. Das Präfix *n* bedeutet, dass dieser Pin in der Logik umgekehrt ist, also nur ein Reset durchgeführt wird, wenn dieser Pin auf das Logik-Level LOW gebracht wird.

Alle Pins auf den Stiftleisten können als Standard-GPIOs konfiguriert werden und somit entweder als Inputs für Buttons und Schiebeschalter verwendet werden oder als Outputs für LEDs oder Schaltrelais. Beachten Sie bitte den Warnhinweis zu Pin PA9 und PA10.

Direkt neben dem Mikrocontroller U1 befindet sich die Stiftleiste J3. Diese ist ab Werk mit zwei Jumpern bestückt. Der Jumper in der Reihe, die sich näher an der Platine befindet, verbindet den wichtigen Steuer-Pin BOOT0 des Mikrocontrollers entweder mit der Masse oder der Versorgungsspannung (VDD). Die zweite Reihe der Stiftleiste J3 kann mit dem zweiten Jumper einen weiteren GPIO-Pin (PB2) nach Masse oder VDD überbrücken.

Lieferumfang

Im Lieferumfang des Bausatzes (Bild 3) sind neben dem Basismodul Mikrocontroller zusätzlich zwei Jumper für die Stiftleiste J3 enthalten. Die Jumper sind ab Werk aufgesteckt. Das ELV Basismodul Mikrocontroller wird vollständig montiert geliefert und kann sofort in Betrieb genommen werden.

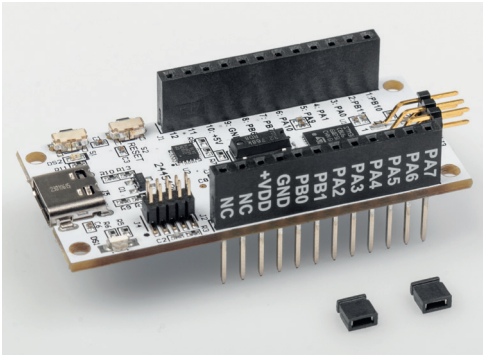


Bild 3: Lieferumfang des Bausatzes ELV-BM-MCU

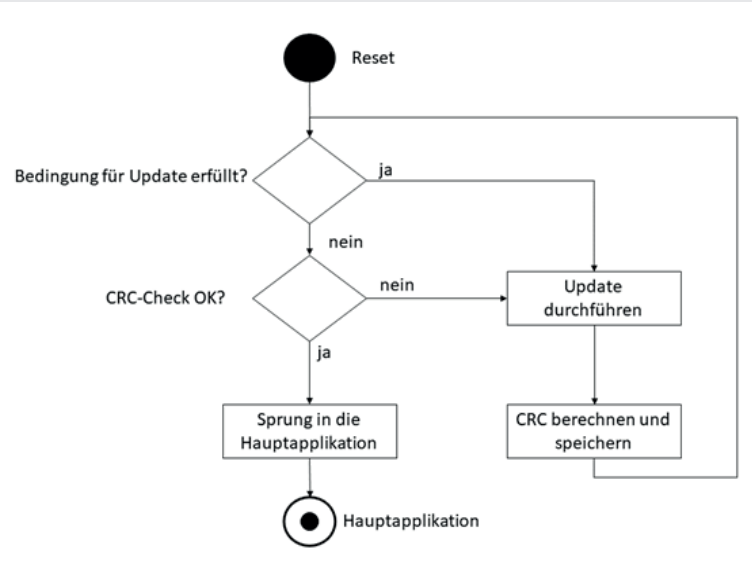
Stückliste

Widerstände:	
180 Ω/SMD/0402	R5
470 Ω/SMD/0402	R6
1 kΩ/SMD/0402	R4, R11
1,2 kΩ/SMD/0402	R12, R13
3,9 kΩ/SMD/0402	R9, R10
10 kΩ/SMD/0402	R3
18 kΩ/SMD/0402	R7
47 kΩ/SMD/0402	R8
100 kΩ/SMD/0402	R1, R2
Kondensatoren:	
8,2 pF/25 V/SMD/0201	C4, C5
100 pF/50 V/SMD/0402	C2
10 nF/50 V/SMD/0402	C10, C12, C14, C16
100 nF/16 V/SMD/0402	C1, C3, C6, C7, C9, C11, C13, C15, C17,
10 µF/16 V/SMD/0805	C8
Halbleiter:	
STM32L052C8U6/SMD	U1
CP2102N/SMD	U2
LM66100/SMD	U3
SP0503BAHTG/SMD	D1
Duo-LED/rot-grün/SMD	DS1
LED/grün/SMD/0603	DS2
Sonstiges:	
Quarz, 32,768 kHz, SMD	Y1
Taster mit 1,2-mm-Tastknopf, 1x ein, SMD, 1,8 mm	S1, S2
Buchsenleisten, 1x 12-polig, 10 mm Pinlänge, gerade	J1, J2
Stiftleiste, 2x 3-polig, abgewinkelt, SMD	J3
Stiftleiste, 2x 5-polig, 5,97 mm, gerade, RM = 1,27 mm, SMD	J4
USB-Buchse, Typ C, SMD	J5
Jumper, RM = 2,0 mm, schwarz, ohne Fahne	

Technikwissen

Bootloader

Der Bootloader ist ein Programm, das ganz zu Beginn der Initialisierung eines Mikrocontrollers ausgeführt wird. Bei Ausführung werden grundlegende Peripherien des Controllers gestartet. Anhand mehrerer Kriterien wird überprüft, ob und an welche Speicheradresse nach Ausführung des Bootloaders gesprungen wird, um den Code der Applikationsfirmware auszuführen. Sollte diese Prüfung zum Ergebnis kommen, dass nicht in die Applikation gesprungen werden soll, kann nun der Speicherbereich der Applikationsfirmware gefahrlos überschrieben werden. So kann ein Firmware-Update durchgeführt werden. Zum Schluss kann noch durch eine CRC-Prüfung sichergestellt werden, dass die komplette Firmware überspielt wurde (Ablauf siehe Bild rechts).



Inbetriebnahme

Bevor Sie mit der Entwicklung auf der ELV-MCU-Base beginnen können, müssen Sie die neuste Version des VCP-Treibers installieren. Laden Sie den Treiber von der Produktseite des ELV-Basismoduls Mikrocontroller herunter und installieren Sie diesen.

Verbinden Sie nach Installation der Treiber ein USB-C-Kabel mit dem Mikrocontroller und Ihrem PC, um die Kommunikation über die serielle Schnittstelle mit dem Gerät zu ermöglichen. Sobald die ELV-MCU-Base mit einem PC verbunden ist, leuchtet die LED DS2 grün.

Bei dem in der ELV-MCU-Base verwendeten Mikrocontroller wird im Bootloader geprüft, ob der Flash-Speicher leer ist oder ob der BOOT0-Pin des Mikrocontrollers den Pin-Status HIGH hat. Dieser Pin ist auf der Stiftleiste J3 zu finden und kann mit einem der beiden beiliegenden Jumper entweder nach Masse (LOW) oder nach VDD (HIGH) überbrückt werden (siehe Bild 4).

Um eine neue Firmware einzuspielen, überbrücken Sie BOOT0-Pin und VDD, um den Update-Modus des Mikrocontrollers zu initialisieren.

Zur Programmierung der ELV-MCU-Base sind einige Software-Pakete notwendig. Installieren Sie Arduino IDE 2.3.3 sowie die Bibliothek STM32duino Board Library (Mindestversion 2.8.1).

Wie diese Bibliothek installiert wird, finden Sie in diesem Fachbeitrag. Installieren Sie zudem den STM32CubeProgrammer.

Ein guter Startpunkt ist die Beispiel-Firmware aus dem ELVshop. In dieser wird eine der LEDs regelmäßig ein- und ausgeschaltet und der Zustand dieser LED über die serielle Schnittstelle übertragen. Um diese in die Entwicklungsumgebung einzubinden, laden Sie die Zip-Datei herunter und entpacken Sie diese. Öffnen Sie die im Ordner enthaltene ELV-BM-MCU-Beispiel-Firmware.ino-Datei:

Codeblock 1

```
#include "ELV-BM-MCU.h"

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  Serial1.begin(115200);
}

void loop() {
  digitalWrite(LED_BUILTIN, LOW);
  Serial1.println("An");
  delay(1000);
  digitalWrite(LED_BUILTIN, HIGH);
  Serial1.println("Aus");
  delay(1000);
}
```

Bevor die Datei übertragen werden kann, wählen Sie zunächst den korrekten Mikrocontroller aus.
Klicken Sie im Menüband auf
„Tools“ → „Board“ → „STM32 MCU based boards“ → „Generic STM32L0 series“.
Wählen Sie anschließend unter
„Tools“ → „Board part number“ die Option „Generic L052C8Ux“ aus.
Zum Schluss wählen Sie unter
„Tools“ → „Upload method“ noch „STM32CubeProgrammer (Serial)“ aus.
Diese Einstellungen müssen zu Beginn der Entwicklung einmal ausgeführt werden, danach bleiben diese erhalten, bis sie geändert werden.

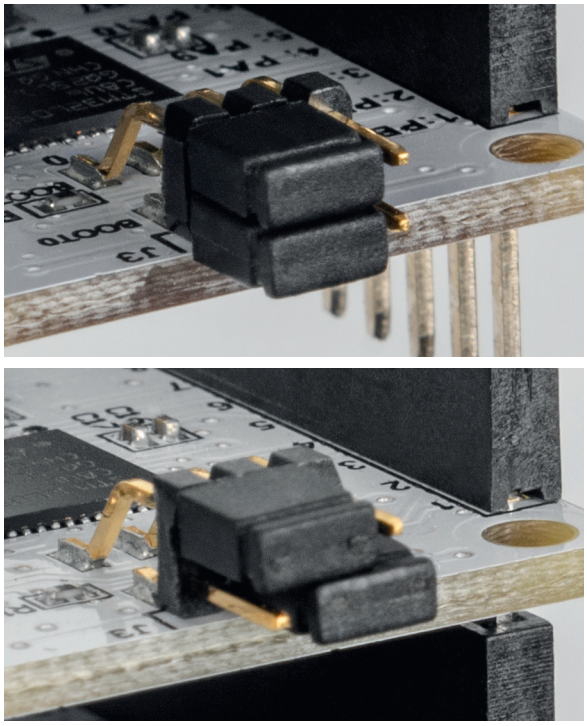


Bild 4: Oben: Pin BOOT0 ist nach GND überbrückt, bei Neustart des Geräts wird direkt in die Applikation gesprungen.
Unten: Pin BOOT0 ist nach VDD überbrückt, bei Neustart des Geräts wird im Bootloader verblieben, und die Firmware kann neu aufgespielt werden.

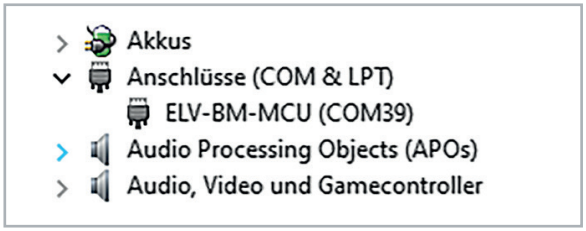


Bild 5: Das Gerät wird als COM-Anschluss im Gerätemanager erkannt.

Bild 6: Buttons zum Kompilieren und Übertragen der Firmware in der Arduino IDE

Bild 7: Um die Ausgabe über die serielle Schnittstelle auslesen zu können, wird der serielle Monitor der Arduino-Entwicklungsumgebung verwendet.

Ist die ELV-MCU-Base über ein USB-Kabel mit dem PC verbunden, wird aber nicht korrekt vom Gerätemanager erkannt (Bild 5), installieren Sie die VCP-Treiber wie zu Beginn des Kapitels beschrieben.

Die letzte Einstellung vor dem Programmieren ist die Angabe des von Windows zugewiesenen COM-Ports in der Arduino IDE. Wählen Sie unter „Tools" → „Port" den entsprechenden COM-Port aus.

Übertragen von Firmware

Um den Code aus dem Code-Editor in Maschinencode zu übersetzen und diesen auf den Mikroprozessor zu übertragen, stellen Sie sicher, dass sich das Gerät im Bootloader befindet (BOOT0-Jumper in der korrekten Position, siehe Bild 4 unten). Drücken Sie anschließend auf den Reset-Button S2. Nun kann eine neue Firmware aufgespielt werden.

Klicken Sie auf den Upload-Button (siehe Bild 6 rechts), um die zuvor geschriebene Firmware auf das Gerät zu übertragen. Nach erfolgreicher Übertragung erteilt die Entwicklungsumgebung den expliziten Befehl, zum Beginn des Programms zu springen. Das Programm wird also direkt ausgeführt und die LED in DS1 blinkt grün.

Die Funktionalität lässt sich auch am seriellen Monitor (Bild 7) überprüfen. Um nach einem Neustart das Programm erneut zu starten, muss die Position des Jumpers des BOOT0-Pins entsprechend angepasst werden (siehe Bild 4 unten).

Klicken Sie in der rechten oberen Ecke des Programmfensters auf die Schaltfläche „Serial Monitor".

Nach der Übertragung des Programms und der Angabe der korrekten Baud-Rate (115200) werden die Ausgaben des Programms nun im unteren Bildschirmteil angezeigt. Diese zeigt nun für die Beispiel-Firmware den Status der LED an.

Programmieren mit Arduino

Die Grundstruktur eines Arduino-Programms besteht aus zwei voneinander getrennten Abschnitten. Im `setup()`-Teil wird ein Code eingefügt, der einmal zu Beginn der Applikation ausgeführt wird. Initialisierungen und Kalibrierungen von Sensoren oder Peripherien des Mikrocontrollers werden in der Regel im `setup()`-Teil ausgeführt. Der zweite Teil ist die `loop()`-Funktion. Diese wird immer wieder durchgeführt. Hier sollten Sensorabfragen oder Ausgänge geschaltet werden.

Beispiel Schaltrelais

Um mithilfe des ELV Basismoduls Mikrocontroller einen anderen elektrischen Verbraucher zu schalten, kann das ELV Relais-Schaltmodul RSM1 verwendet werden. Dieses kann an einen Open-Collector-Ausgang angeschlossen werden. Da das RSM1 eine Schaltspannung von 12 V voraussetzt, ist zusätzlich ein Step-up-Wandler nötig. An die Eingänge des Spannungswandlers wird die 5-V-USB-Versorgungsspannung angeschlossen, und die Ausgänge werden mit den äußeren beiden Eingangsklemmen (+ und GND) des RSM1 verbunden. Des Weiteren wird an einem beliebigen GPIO-Pin hinter einem 10-kΩ-Widerstand die Basis eines NPN-Transistors angeschlossen. Der Collector dieses Transistors wird mit der Eingangsklemme OC (Open Collector) des Relais-Schaltmoduls und der Emitter mit der verbleibenden Masseklemme des RSM1 verbunden. Zuletzt wird als Auslöser noch ein Vibrations- bzw. Kugelsensor aus dem Prototypenadapter-Set PAD8 parallel zu einem 100-nF-Kondensator mit einem anderen Pin verbunden. Die gesamte Schaltung finden Sie in Bild 8.

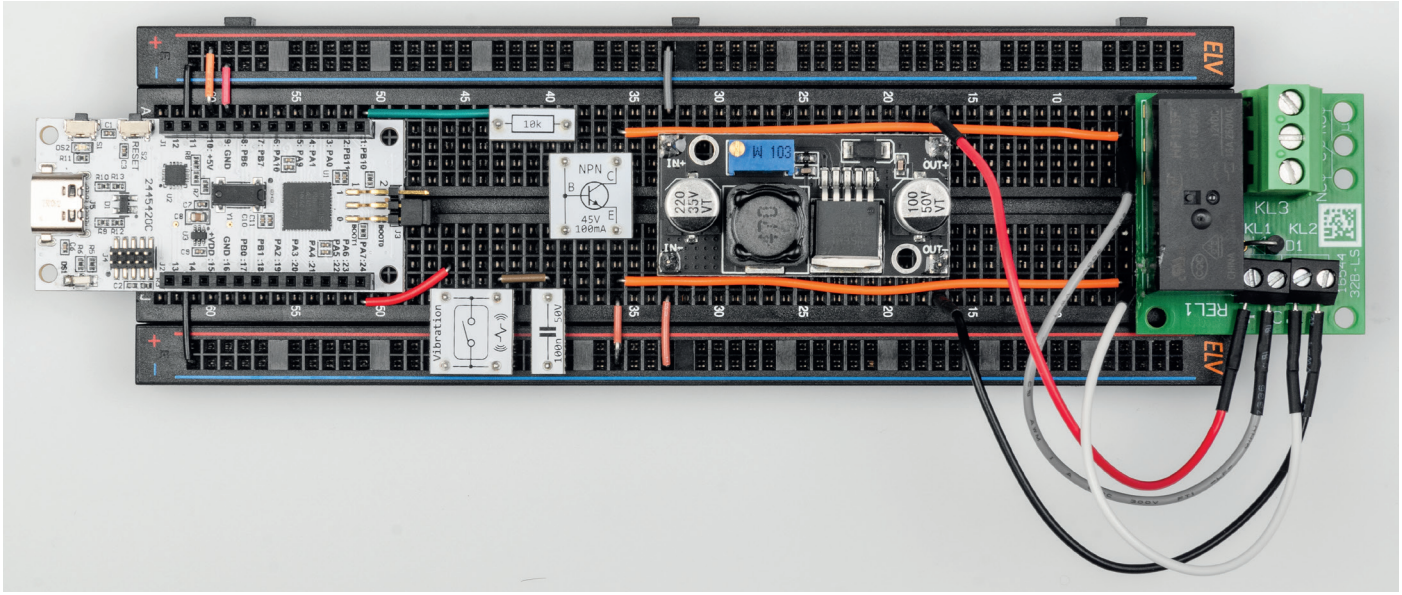


Bild 8: Das Relais-Schaltmodul kann mithilfe eines Spannungswandlers, der die USB-Spannung von 5 V auf 12 V erhöht, und einer Open-Collector-Schaltung verwendet werden.

Die Besonderheit dieser Schaltung ist der Vibrationssensor als Auslöser. Dieser legt ein elektronisches Verhalten an den Tag, das sich „prellen“ nennt. Beim „Prellen“ kommt es zu einem kurzzeitigen „Schwingen“ der Kontakte z. B. eines Tasters oder Schalters. Eine genauere Erklärung zum Prellverhalten des Kugelsensors finden Sie im Fachbeitrag Viele kleine Helferlein - Prototypenadapter-Set PAD8. Auf dieses Verhalten kann sowohl mit Filterschaltungen als auch in der Firmware reagiert werden. Hierfür kann das von Arduino bereitgestellte Codebeispiel „Debounce“ als Vorlage dienen. Um dieses zu öffnen, wählen Sie im Menüband der Entwicklungsumgebung „File“ → „Examples“ → „02. Digital“ → „Debounce“ aus. Um die Schaltung in Bild 8 korrekt zu verarbeiten, sind Anpassungen im Beispiel nötig:

Codeblock 2: Debounce (modifiziert)

```
/*
  Debounce

  Each time the input pin goes from LOW to HIGH (e.g. because of a push-button
  press), the output pin is toggled from LOW to HIGH or HIGH to LOW. There's a
  minimum delay between toggles to debounce the circuit (i.e. to ignore
noise).

  The circuit:
  - LED attached from pin 13 to ground through 220 ohm resistor
  - pushbutton attached from pin 2 to +5V
  - 10 kilohm resistor attached from pin 2 to ground

  - Note: On most Arduino boards, there is already an LED on the board
connected
    to pin 13, so you don't need any extra components for this example.

  created 21 Nov 2006
  by David A. Mellis
  modified 30 Aug 2011
  by Limor Fried
  modified 28 Dec 2012
  by Mike Walters
  modified 30 Aug 2016
  by Arturo Guadalupi

  This example code is in the public domain.

  https://www.arduino.cc/en/Tutorial/BuiltInExamples/Debounce
*/
#include "ELV-BM-MCU.h"
// constants won't change. They're used here to set pin numbers:
const int buttonPin = PIN24;  // the number of the pushbutton pin
const int ledPin = PIN01;    // the number of the LED pin

// Variables will change:
int ledState = HIGH;         // the current state of the output pin
int buttonState;             // the current reading from the input pin
int lastButtonState = HIGH;  // the previous reading from the input pin

// the following variables are unsigned longs because the time, measured in
// milliseconds, will quickly become a bigger number than can be stored in an
int.
unsigned long lastDebounceTime = 0;  // the last time the output pin was
toggled
unsigned long debounceDelay = 50;    // the debounce time; increase if the
output flickers

void setup() {
  pinMode(buttonPin, INPUT_PULLUP);
  pinMode(ledPin, OUTPUT);

  // set initial LED state
  digitalWrite(ledPin, ledState);
}

void loop() {
  // read the state of the switch into a local variable:
  int reading = digitalRead(buttonPin);

  // check to see if you just pressed the button
  // (i.e. the input went from LOW to HIGH), and you've waited long enough
  // since the last press to ignore any noise:

  // If the switch changed, due to noise or pressing:
  if (reading != lastButtonState) {
    // reset the debouncing timer
    lastDebounceTime = millis();
  }
  if ((millis() - lastDebounceTime) > debounceDelay) {
    // whatever the reading is at, it's been there for longer
    than the debounce
    // delay, so take it as the actual current state:

    // if the button state has changed:
    if (reading != buttonState) {
      buttonState = reading;

      // only toggle the LED if the new button state is LOW
      if (buttonState == LOW) {
        ledState = !ledState;
      }
    }
  }

  // set the LED:
  digitalWrite(ledPin, ledState);

  // save the reading. Next time through the loop, it'll be the
lastButtonState:
  lastButtonState = reading;
}
      // only toggle the LED if the new button state is LOW
      if (buttonState == LOW) {
        ledState = !ledState;
      }
    }
  }

  // set the LED:
  digitalWrite(ledPin, ledState);

  // save the reading. Next time through the loop, it'll be the
lastButtonState:
  lastButtonState = reading;
}
```

Fügen Sie zunächst die ELV-BM-MCU.h-Header-Datei (enthält die Zuordnungen der Pins vom Mikrocontroller zu den Stiftleisten) wie folgt hinzu:

Klicken Sie in der oberen rechten Ecke des Editors auf die Schaltfläche mit den drei Punkten, und wählen Sie die Option „New Tab" aus. Geben Sie im Dialogfenster bei „Dateinamen" ELV-BM-MCU.h ein. In diese Datei wird dann der Inhalt der gleichnamigen Datei aus dem Zip-Verzeichnis im Downloadbereich eingefügt.

Codeblock 3

```
#define PIN01 PB10
#define PIN02 PB11
#define PIN03 PA0
#define PIN04 PA1
#define PIN05 PA9
#define PIN06 PA10
#define PIN07 PB7
#define PIN08 PB8
#define PIN09 NC
#define PIN10 NC
#define PIN11 NC
#define PIN12 NC
#define PIN13 NC
#define PIN14 NC
#define PIN15 NC
#define PIN16 NC
#define PIN17 PB0
#define PIN18 PB1
#define PIN19 PA2
#define PIN20 PA3
#define PIN21 PA4
#define PIN22 PA5
#define PIN23 PA6
#define PIN24 PA7

#define LED_GREEN PB15
#define LED_RED PB14
#define LED_BUILTIN LED_GREEN

HardwareSerial Serial1(PA10, PA9);
```

Nach dem Hinzufügen dieser Datei können Sie den Konstanten `buttonPin` und `ledPin` die der Schaltung entsprechenden Werte zuweisen.

Konfigurieren Sie unbedingt den buttonPin in Zeile 45 (Codeblock 2) mit einem Pull-up-Widerstand, da ansonsten der Initialzustand des Sensors nicht definiert ist. Ändern Sie in Zeile 75 (Codeblock 2) die Abfrage auf das Logiklevel `LOW` statt `HIGH`, da der Kugelsensor den verwendeten Pin mit der Masse verbindet.

In der `loop()`-Methode wird regelmäßig abgefragt, ob sich der Zustand des Buttons geändert hat. Ist dies der Fall, wird der aktuelle Zeitpunkt in einer Variable `lastDebounceTime` gespeichert (Zeile 61-63). Wenn die Zeitspanne seit dem letzten Mal, an dem sich der Zustand geändert hat, kürzer ist als das Intervall, das in Zeile 42 (Codeblock 2) definiert wurde, wird der Zustand des Sensors als aktueller Zustand ignoriert. Das führt dazu, dass ein stetiges Hin- und Herwechseln des Logiklevels ignoriert wird, bis sich ein neuer Zustand konstant eingestellt hat.

Beispielapplikation UV-Ampel

Als Beispiel für die Verwendung digitaler Sensoren wird aus dem ELV Applikationsmodul Optische Strahlungssensoren (ELV-AM-ORS) eine UV-Ampel gebaut. Hierfür wird das Applikationsmodul einfach auf das Basismodul aufgesteckt. Zuletzt verbinden Sie jeweils noch Masse und Versorgungsspannung mit den entsprechenden Rails (siehe Beispielbild). Die gesamte Verschaltung ist in Bild 9 abgebildet.

Für den AS7331-UV-Sensor gibt es bereits eine fertige Arduino-Bibliothek. Diese kann im Library Manager unter

„SparkFun_AS7331_Arduino_Library"

installiert und im Code verwendet werden.

In der `Setup()`-Methode wird zuerst die serielle Schnittstelle initialisiert und eine initiale Ausgabe gemacht (Zeile 34-36, Codeblock 4).

Danach wird über den Befehl `Wire.begin();` die I²C-Schnittstelle initialisiert. Diese wird benötigt, um mit dem AS7331-UV-Sensor zu kommunizieren. Dann wird in den Zeilen 41-45 (Codeblock 4) die Initialisierung des Sensors vorgenommen. War diese erfolgreich, wird der Sensor auf das Messen vorbereitet (Codeblock 4, Zeilen 50-54).

Zuletzt wird noch die Duo-LED in den Zeilen 58 und 59 (Codeblock 4) initialisiert.

Während der `loop()`-Methode wird die Messung des Sensors mit dem Befehl `setStartState(true);` gestartet. Nun wird so lange gewartet, bis die Messung fertig ist. Die Messwerte des Sensors werden übermittelt, ausgegeben und umgerechnet (Zeile 72-80, Codeblock 4). Anhand dieser Werte wird anschließend die Farbe der LED gesetzt. Die LED zeigt rotes bzw. grünes Licht, wenn die beiden LEDs jeweils einzeln angesteuert werden. Wenn sie in Kombination angeschaltet werden, vermischt sich das Licht und es erscheint Orange-Gelb.

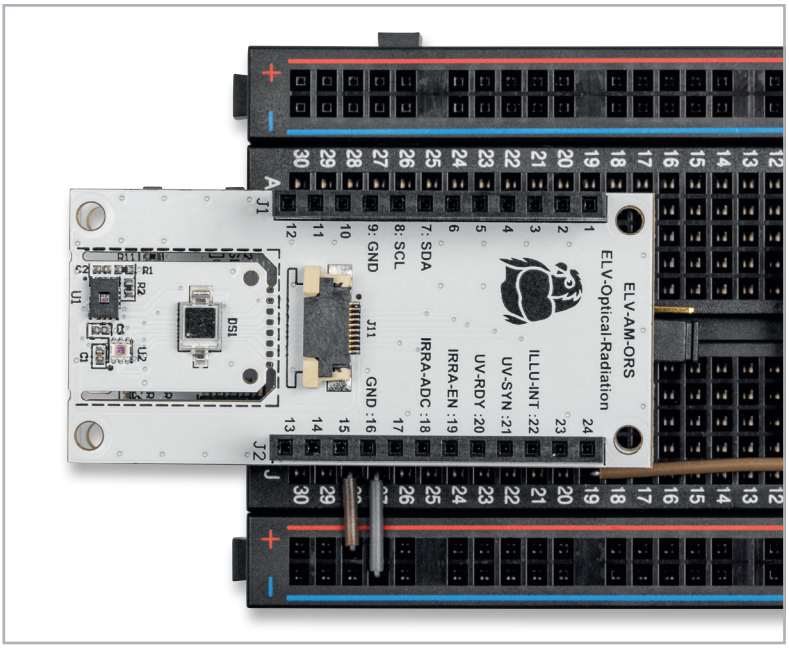


Bild 9: Unterhalb des ELV-AM-ORS befindet sich das ELV Basismodul Mikrocontroller.

Codeblock 4

```
/*
  Using the AMS AS7331 Spectral UV Sensor in Command/One Shot (CMD) Mode.

  This example shows how operate the AS7331 in the default CMD mode. The start
  command is sent, then delays until the conversion time has passed before
  reading out the UV values.

  By: Alex Brudner
  SparkFun Electronics
  Date: 2023/11/17
  SparkFun code, firmware, and software is released under the MIT License.
  Please see LICENSE.md for further details.

  Hardware Connections:
  IoT RedBoard --> AS7331
  QWIIC --> QWIIC

  Serial1.print it out at 115200 baud to serial monitor.

  Feel like supporting our work? Buy a board from SparkFun!
  https://www.sparkfun.com/products/23517 - Qwiic 1x1
  https://www.sparkfun.com/products/23518 - Qwiic Mini
*/

#include <Arduino.h>
#include <Wire.h>
#include <SparkFun_AS7331.h>

#include "ELV-BM-MCU.h"

SfeAS7331ArdI2C myUVSensor;

void setup() {
  Serial1.begin(115200);
  while(!Serial1){delay(100);};
  Serial1.println("AS7331 UV A/B/C Command (One-shot) mode Example.");

  Wire.begin();

  // Initialize sensor and run default setup.
  if(myUVSensor.begin(kQuaternaryAS7331Addr) == false) {
    Serial1.println("Sensor failed to begin. Please check your wiring!");
    Serial1.println("Halting...");
    while(1);
  }

  Serial1.println("Sensor began.");

  // Set measurement mode and change device operating mode to measure.
  if(myUVSensor.prepareMeasurement(MEAS_MODE_CMD) == false) {
    Serial1.println("Sensor did not get set properly.");
    Serial1.println("Halting...");
    while(1);
  }

  Serial1.println("Set mode to command.");

  pinMode(LED_GREEN, OUTPUT);
  pinMode(LED_RED, OUTPUT);
}

void loop() {

  // Send a start measurement command.
  if(kSTkErrOk != myUVSensor.setStartState(true))
    Serial1.println("Error starting reading!");

  // Wait for a bit longer than the conversion time.
  delay(2+myUVSensor.getConversionTimeMillis());

  // Read UV values.
  if(kSTkErrOk != myUVSensor.readAllUV())
    Serial1.println("Error reading UV.");
  int uv_gesamt = (myUVSensor.getUVA() + myUVSensor.getUVB() +
myUVSensor.getUVC())*0.025;
  Serial1.print("UVA:");
  Serial1.print(myUVSensor.getUVA());
  Serial1.print(" UVB:");
  Serial1.print(myUVSensor.getUVB());
  Serial1.print(" UVC:");
  Serial1.println(myUVSensor.getUVC());
  if(uv_gesamt < 3)
  {
    digitalWrite(LED_GREEN, LOW);
    digitalWrite(LED_RED, HIGH);
  }
  else if(uv_gesamt < 7)
  {
    digitalWrite(LED_GREEN, LOW);
    digitalWrite(LED_RED, LOW);
  }
  else if(uv_gesamt < 11)
  {
    digitalWrite(LED_GREEN, HIGH);
    digitalWrite(LED_RED, LOW);
  }
  else
  {
    digitalWrite(LED_GREEN, HIGH);
    digitalWrite(LED_RED, HIGH);
  }

  delay(2000);

}
```

Beispielapplikation Drehzahlmesser für die Windgeschwindigkeit

Als drittes Beispiel wird aus einem Windrad, an dem ein Magnet befestigt wird, dem ELV Applikationsmodul Hallsensor (ELV-AM-Hall) und einer 7-Segment-Anzeige ein Windgeschwindigkeitsdrehzahlmesser aufgebaut. Der Hallsensor wird hierzu einfach auf das Basismodul aufgesteckt und mit dem Windrad verbunden (Bild 10 und Bild 11).

Anschließend stecken Sie die 7-Segment-Anzeige auf und verbinden diese wie in der Tabelle aufgezeigt.

Pins MCU-Base	Pins 7-Segment-Anzeige
21 (PA4)	A (links)
22 (PA5)	B (links)
24 (PA6)	C (links)
24 (PA7)	D (links)
7 (PB7)	A (rechts)
3 (PA0)	B (rechts)
2 (PB11)	C (rechts)
1 (PB10)	D (rechts)

In den vorherigen Beispielen wurde die normale Programmstruktur mit `setup()` und `loop()` verwendet. Nun gibt es in der Welt der Mikrocontroller Ereignisse, die ohne Wartezeit abgearbeitet werden müssen. Hierzu werden sogenannte Interrupts verwendet. Ein Interrupt kann, wenn bestimmte Bedingungen eintreten, das regulär laufende Programm anhalten (Interrupt Request IRQ) und die sogenannte Interrupt Service Routine (ISR) durchlaufen. Wenn diese fertig abgearbeitet wurde, wird das Hauptprogramm an der Stelle weitergeführt, an der es unterbrochen wurde.

Dieses Verhalten wird sich bei diesem Beispielaufbau zunutze gemacht. Der Ausgangspin des ELV Applikationsmoduls Hallsensor wird als externer Interrupt in Zeile 41 von Codeblock 5 definiert .

Die Funktion `ISR1_anemometer_callback()` wird als ISR definiert, der Interrupt soll nur bei einer steigenden Spannungsflanke ausgelöst werden. In der definierten ISR wird eine Variable im Wert inkrementiert, also gezählt, wie oft der Interrupt ausgelöst wurde. In der Auswertung wird dieser Wert genommen und durch das Messintervall geteilt, um daraus die Drehzahl pro Minute über diesen Zeitraum im Mittelwert zu berechnen (Zeilen 18–22, Codeblock 5).

Der reguläre Programmablauf ist, dass zu jedem Messintervall diese Drehzahl abgefragt wird, und dann die Ausgänge, die mit den Eingängen der 7-Segment-Anzeige aus dem PAD6-Set verbunden sind (siehe Bild 11) entsprechend den Messwerten geschaltet werden. Hierfür wird mithilfe des Modulo-Operators (`%`) die letzte Ziffer des Messwerts extrahiert und über die `bitRead()`-Funktion auf einzelne Binärkomponenten aufgeteilt (Zeile 54–73, Codeblock 5).

Die 7-Segment-Anzeigen funktionieren so, dass sie ein 4-Bit-Signal an 4 Eingabepins entgegennehmen und diese dann in eine Anzeige umrechnen. Nachdem diese Prozedur für die „Einer"-Stelle der Drehzahl durchlaufen wurde, wird die Drehzahl ohne Rest durch 10 geteilt und wiederholt.

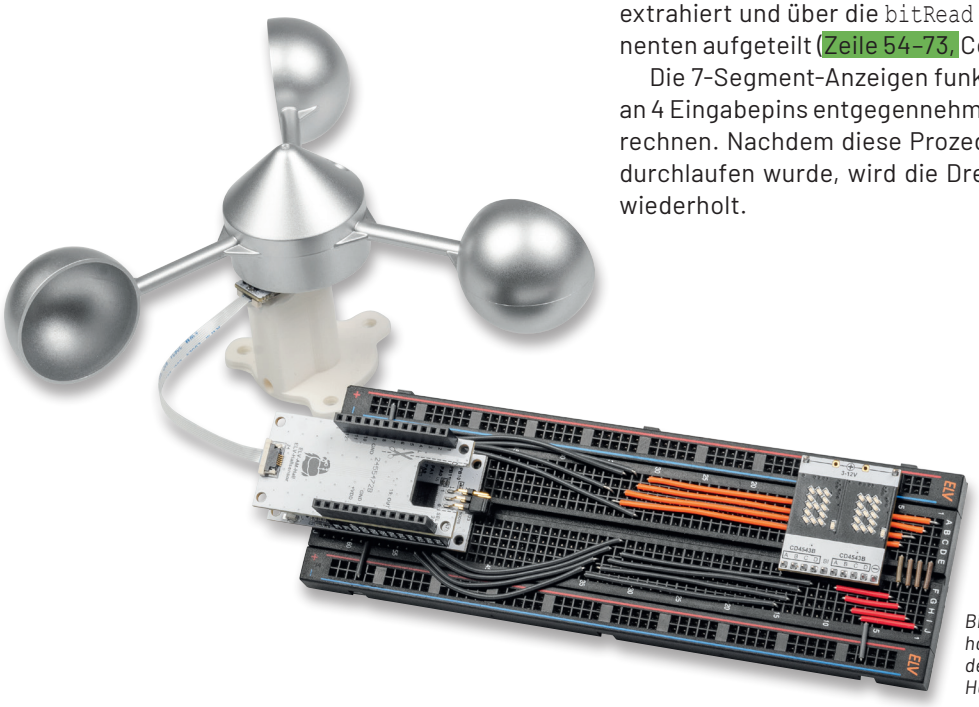


Bild 10: Gesamtaufbau des Drehzahlmessers. Unterhalb des Anemometers befindet sich ein Magnet, der den Magnetsensor des ELV Applikationsmoduls Hallsensor auslöst.

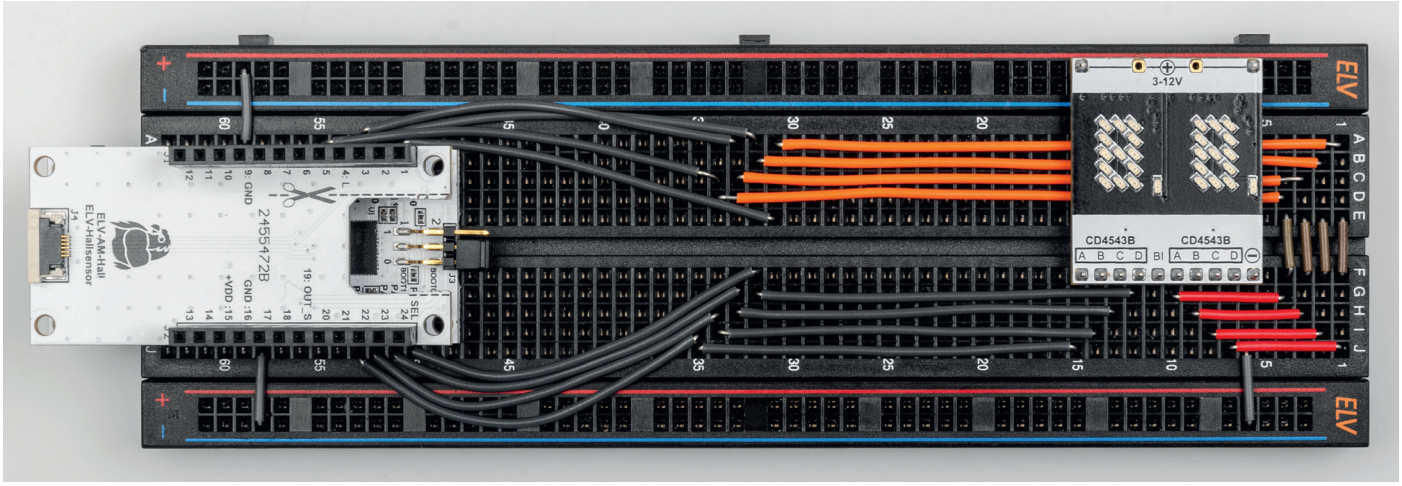


Bild 11: Detailaufnahme der Anschlüsse des Drehzahlmessers

Codeblock 5

```
#include "ELV-BM-MCU.h"

unsigned int rotation_counter = 0;
const unsigned int measure_period_seconds = 10;

#define SEVEN_SEG_PIN_A1 PIN07
#define SEVEN_SEG_PIN_B1 PIN03
#define SEVEN_SEG_PIN_C1 PIN02
#define SEVEN_SEG_PIN_D1 PIN01

#define SEVEN_SEG_PIN_A2 PIN21
#define SEVEN_SEG_PIN_B2 PIN22
#define SEVEN_SEG_PIN_C2 PIN23
#define SEVEN_SEG_PIN_D2 PIN24

#define PIN_ELV_AM_Hall_LATCH PIN04

unsigned int calculateRotationsPerMinute()
{
  unsigned int rpm = (unsigned int)((float)rotation_counter /
(float)measure_period_seconds * 60.0f);
  rotation_counter = 0;
  return rpm;
}

void ISR1_anemometer_callback()
{
  rotation_counter++;
}

void setup() {
  pinMode(SEVEN_SEG_PIN_A1, OUTPUT);
  pinMode(SEVEN_SEG_PIN_B1, OUTPUT);
  pinMode(SEVEN_SEG_PIN_C1, OUTPUT);
  pinMode(SEVEN_SEG_PIN_D1, OUTPUT);

  pinMode(SEVEN_SEG_PIN_A2, OUTPUT);
  pinMode(SEVEN_SEG_PIN_B2, OUTPUT);
  pinMode(SEVEN_SEG_PIN_C2, OUTPUT);
  pinMode(SEVEN_SEG_PIN_D2, OUTPUT);

  pinMode(PIN_ELV_AM_Hall_LATCH, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(PIN_ELV_AM_Hall_LATCH),
ISR1_anemometer_callback, RISING);
  digitalWrite(SEVEN_SEG_PIN_A1, LOW);
  digitalWrite(SEVEN_SEG_PIN_B1, LOW);
  digitalWrite(SEVEN_SEG_PIN_C1, LOW);
  digitalWrite(SEVEN_SEG_PIN_D1, LOW);
  digitalWrite(SEVEN_SEG_PIN_A2, LOW);
  digitalWrite(SEVEN_SEG_PIN_B2, LOW);
  digitalWrite(SEVEN_SEG_PIN_C2, LOW);
  digitalWrite(SEVEN_SEG_PIN_D2, LOW);
}

void setOutputToDigit(unsigned int decimal, unsigned int output_number)
{
  if((0 <= decimal) && (decimal < 10))
  {
    if(output_number == 1)
    {
      digitalWrite(SEVEN_SEG_PIN_A1, bitRead(decimal, 0));
      digitalWrite(SEVEN_SEG_PIN_B1, bitRead(decimal, 1));
      digitalWrite(SEVEN_SEG_PIN_C1, bitRead(decimal, 2));
      digitalWrite(SEVEN_SEG_PIN_D1, bitRead(decimal, 3));
    }
    else if(output_number == 2)
    {
      digitalWrite(SEVEN_SEG_PIN_A2, bitRead(decimal, 0));
      digitalWrite(SEVEN_SEG_PIN_B2, bitRead(decimal, 1));
      digitalWrite(SEVEN_SEG_PIN_C2, bitRead(decimal, 2));
      digitalWrite(SEVEN_SEG_PIN_D2, bitRead(decimal, 3));
    }
  }
}

void loop() {
  delay(measure_period_seconds * 1000);
  noInterrupts();
  unsigned int rotations_per_seconds = calculateRotationsPerMinute();
  interrupts();
  if (rotations_per_seconds < 99)
  {
    setOutputToDigit(rotations_per_seconds%10, 1);
    setOutputToDigit((rotations_per_seconds/10)%10, 2);
  }

}
```

Fazit

Abschließend lässt sich festhalten, dass das ELV Basismodul Mikrocontroller Entwicklern eine leistungsfähige und vielseitige Plattform bietet. Die gezeigten Code-Beispiele verdeutlichen die Flexibilität des Boards und die einfache Programmierung des Controllers, was besonders bei der Entwicklung und dem Testen neuer Anwendungen von Vorteil ist.

Dank der breiten Unterstützung verschiedener Applikationsmodule und seiner umfassenden Peripherie-Ausstattung eignet sich das Board für eine Vielzahl von Projekten: von IoT-Lösungen bis hin zu industriellen Anwendungen. Es ermöglicht Entwicklern, ihre Ideen schnell und effizient in die Realität umzusetzen und dabei von einer stabilen, gut dokumentierten und erweiterbaren Hardwarebasis zu profitieren. ELV

Alle Programmcodes
finden Sie auch als Source-Datei
im ELVshop bei diesem Fachbeitrag
unter Downloads.

Technische Daten

Geräte-Kurzbezeichnung:	ELV-BM-MCU
Stromaufnahme:	Stop Mode: typ. 3 µA Normal Mode: 2,9 mA
Versorgungsspannung:	3,0–3,3 VDC (Pin 15) 5 VDC (Pin 10) 5 VDC (USB-Powered)
Leitungslängen an J1 und J2:	max. 10 cm
Abmessungen (B x H x T):	26 x 59 x 19 mm
Gewicht:	9 g

Hinweis zu den vorbestückten Bausatz-Leiterplatten

Sehr geehrter Kunde,

das Gesetz über das Inverkehrbringen, die Rücknahme und die umweltverträgliche Entsorgung von Elektro- und Elektronikgeräten (ElektroG) verbietet (abgesehen von wenigen Ausnahmen) seit dem 1. Juli 2006 u. a. die Verwendung von Blei und bleihaltigen Stoffen mit mehr als 0,1 Gewichtsprozent Blei in der Elektro- und Elektronikproduktion.

Die ELV Produktion wurde daher auf bleifreie Lötzinn-Legierungen umgestellt, und sämtliche vorbestückte Leiterplatten sind bleifrei verlötet.

Bleihaltige Lote dürfen im Privatbereich zwar weiterhin verwendet werden, jedoch kann das Mischen von bleifreien und bleihaltigen Loten auf einer Leiterplatte zu Problemen führen, wenn diese im direkten Kontakt zueinander stehen. Der Schmelzpunkt an der Übergangsstelle kann sich verringern, wenn niedrig schmelzende Metalle wie Blei oder Wismut mit bleifreiem Lot vermischt werden. Das unterschiedliche Erstarren kann zum Abheben von Leiterbahnen (Lift-off-Effekt) führen. Des Weiteren kann der Schmelzpunkt dann an der Übergangsstelle unterhalb des Schmelzpunkts von verbleitem Lötzinn liegen. Insbesondere beim Verlöten von Leistungsbauelementen mit hoher Temperatur ist dies zu beachten.

Wir empfehlen daher beim Aufbau von Bausätzen den Einsatz von bleifreien Loten. ELV

WichtigerHinweise zum ESD-Schutz

Das Produkt enthält elektrostatisch gefährdete Bauelemente, die durch unsachgemäße Behandlung beschädigt werden können. Sie müssen beim Umgang mit den Komponenten elektrostatisch entladen sein!

Wichtige Hinweise!

- Für einen ausreichenden Schutz vor elektrostatischen Entladungen ist der Einbau in ein geeignetes Gehäuse erforderlich, damit die Schaltung nicht durch eine Berührung mit den Fingern oder Gegenständen gefährdet werden kann.
- Zur Gewährleistung der elektrischen Sicherheit muss es sich bei der speisenden Quelle um eine Sicherheits-Schutzkleinspannung handeln. Außerdem muss es sich um eine Quelle begrenzter Leistung gemäß EN62368-1 handeln (PS1), die nicht mehr als 15 W, gemessen nach 3 s, liefern kann.
- Es darf jeweils nur eine Spannungsquelle genutzt werden, entweder nur 3,0-3,3 V (+VDD), nur +5 V oder nur USB, da keine ausreichende Trennung zwischen den Spannungen besteht, wenn die Spannungen aus unterschiedlichen SELV-Quelle stammen.
- Leitungen an J1 und J2 dürfen eine Länge von 10 cm nicht überschreiten.

Entsorgungshinweis

Dieses Zeichen bedeutet, dass das Gerät nicht mit dem Hausmüll, der Restmülltonne oder der gelben Tonne bzw. dem gelben Sack entsorgt werden darf.

Sie sind verpflichtet, zum Schutz der Gesundheit und der Umwelt das Produkt und alle im Lieferumfang enthaltenen Elektronikteile zur ordnungsgemäßen Entsorgung bei einer kommunalen Sammelstelle für Elektro- und Elektronik-Altgeräte abzugeben. Auch Vertreiber von Elektro- und Elektronikgeräten sind zur unentgeltlichen Rücknahme von Altgeräten verpflichtet.
Durch die getrennte Erfassung leisten Sie einen wertvollen Beitrag zur Wiederverwendung, zum Recycling und zu anderen Formen der Verwertung von Altgeräten.

Wir machen ausdrücklich darauf aufmerksam, dass Sie als Endnutzer eigenverantwortlich für die Löschung personenbezogener Daten auf dem zu entsorgenden Elektro- und Elektronik-Altgerät sind.

Bevollmächtigter des Herstellers:
ELV Elektronik AG · Maiburger Straße 29–36 · 26789 Leer · Germany