www.elektormagazine.com

elektor MAG

design > share > earn

WK 01 * SINCE 1961 *
DEC 2022/JAN 2023
ELEKTORMAGAZINE.COM

Special edition guest-edited by ARDUINO

Declassified Bonus Edition!

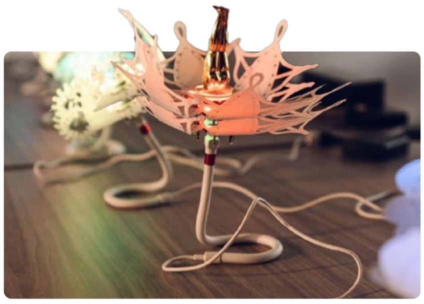
p. 4 Retro Gaming with Arduino

p. 14 A Controller for Spotify

Articles for Pros, Makers, and Students!

Elektor's Arduino Training Board p. 10

Bringing the Human Element to Manufacturing p. 24



Create Kinetic Sculptures p. 32

From Arduino and Elektor with Engineering Love

OUT NOW

A unique Elektor Magazine edition curated by guest editor Arduino!

DIY electronics projects, engineering insights, and more from Arduino and Elektor engineers

Packed with projects and tutorials

Dive into hot topics such as MicroPython, TinyML, and home automation with Arduino

£9.95 | €10.95 | $12.95
elektor MAG
design > share > earn
518
DEC. 2022 & JAN. 2023
ELEKTORMAGAZINE.COM
SINCE 1961
Special edition guest-edited by
ARDUINO
140 Pages
Arduino Projects
Home Automation
Connectivity Simplified
Go Professional with Arduino Pro
Arduino Portenta Machine Control and Arduino Portenta H7
Prototyping to Production
Art in Arduino
In this issue
Writing Arduino Sketches Just Got Better
Mozzi Arduino Library for Sound Synthesis
Introduction to TinyML
Connected Projects Simplified
MicroPython Enters the World of Arduino
and much more!
Taking over Elektor!
Building Future Skills with Arduino
Save the Planet with Home Automation
Getting to know Arduino with Fabio, Massimo, and David
p. 90
p. 62
p. 32
ISSN 1757-0875

Get Started with the Portenta x8

Get it now! From your favorite newsstand or buy in the Elektor web stores!

Direct links in articles give you easy access to Arduino products and solutions

Get to know Arduino: Insights from Fabio, Massimo and David

More information

www.elektor.com/arduino-magazine

Elektor Magazine, English edition

Volume 48, BONUS EDITION
December 2022 & January 2023
ISSN 1757-0875 (UK / US / ROW distribution)
www.elektor.com / www.elektormagazine.com

Head Office:
Elektor International Media b.v.
PO Box 11, 6114 JG Susteren (NL)
Phone: (+31) 46 4389444

Memberships: service@elektor.com
www.elektor.com/memberships

Advertising & Sponsoring:
Raoul Morreau: raoul.morreau@elektor.com
Phone: +31 (0)6 4403 9907

The Team:
International Editor-in-Chief: Jens Nickel
Content Director: C. J. Abate
International Editorial Staff:
Eric Bogers, Jan Buiting, Stuart Cording,
Rolf Gerstendorf, Alina Neacsu,
Dr Thomas Scherer, Brian Tristam Williams
Laboratory Staff: Mathias Claussen,
Ton Giesberts, Luc Lemmens, Clemens Valens
Graphic Design & Prepress: Harmen Heida,
Sylvia Sopamena, Patrick Wielders
Publisher: Erik Jansen

Declassified Bonus Edition!

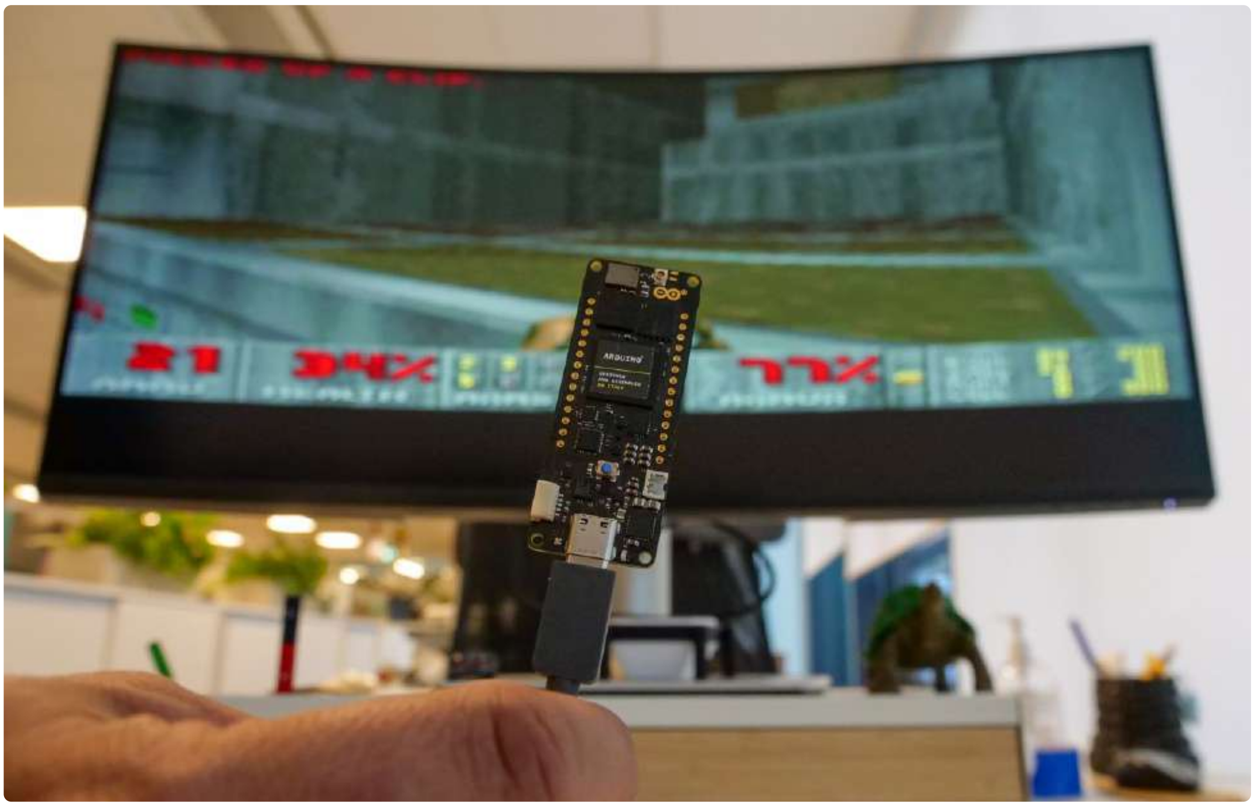
The creative collaboration between Elektor and Arduino did not end with the guest-edited edition of Elektor Mag that we published in early December 2022. We have more projects, technical insights, and informative articles for you to keep inspired for months to come. Over the course of four weeks, we are declassifying content in this edition until you have the complete bonus magazine in early January 2023. What a great way to kick off the new year!

Whether you are a pro engineer working on a new industrial product or a DIYer looking for a fun weekend Arduino-based project, you will find this extra edition of Elektor Mag informative and stimulating. We provide you with articles on a wide range of Arduino-related topics and projects, including retro gaming with Arduino, an Elektor Arduino training board, and a portable Arduino-based controller for Spotify.

As you read the projects and articles in this magazine, feel free to share your thoughts with us at elektormagazine.com, arduino.cc, and on social media. We look forward to your feedback. Enjoy!

C. J. Abate (Content Director, Elektor)

THIS EDITION



You can run Doom on a Portenta!

Running **Doom** on a **Portenta**

Retro Gaming with Arduino

By David Cuartielles (Arduino)

You can run Doom on an Arduino Portenta H7. Curious how it's done? Want to know why Arduino engineers ran the game on it in the first place? Martino Facchin, head of Arduino's firmware team, has the answers.

Doom is probably the most sold game in history, with sales of over 3.5 million copies. At a retail price of $50 (USD), the Doom developers, forming a small company called id Software, became millionaires overnight. When Doom was released in 1993, id Software had already been out with another well-known game for about a year, Wolfenstein 3D. Doom is a first-person shooter game, where the player is meant to navigate a 3D space and confront different enemies using different weapons and ammunition that can also be found all over the game's battlefield. Doom has been ported to all operating systems but also runs on bare metal on multiple systems. The source code is open, currently licensed under GPL. While it is not trivial to compile it, we have seen versions of Doom run on very small computers and also inside other programs. At some point, Microsoft Office's Excel 95 made a homage to Doom in the form of a playable easter egg to be found in the software's credits page.

The game has become a way to both check the performance of small computers and to display hacking wizardry. A few months ago, at DEF CON 22 in Las Vegas, the hacker known as @sickcodes on Twitter and GitHub demoed Doom on a John Deere tractor modded to include farm-related graphics. Arduino is no exception. And when the first prototype of the Portenta H7 — our most powerful board to that point in 2018 — came out, we used Doom to test the technical capabilities of the board. I recently invited

Martino Facchin, head of Arduino's firmware team, to tell us more about this story and how it was done.

David Cuartielles: Let's talk about the Arduino Portenta H7 running Doom. I have prepared a little summary of the history of Doom, how the guys at the id Software company made the game, how it sold like crazy, and how the makers became millionaires.

Martino Facchin: And then they published the source code — the most important thing.

Cuartielles: Exactly! They published the source code, but under which license did they publish it?

Facchin: I've got to check, but I think it is a license which is compatible with GPL, the Doom Source Code Licence. The important thing is that only the engine is open source. The assets are closed, and, in fact, you can only play the shareware version of Doom, so you cannot really play the full game unless you bought it.

A Note from David
I went to check this because I wanted to be sure about it, and in 1997, id Software published Doom under the above-mentioned licence, which was opening up the source for educational purposes. After an accident that happened to the maintainers of glDoom, which left the world without a copy of the openGL port Doom because of the non-distribution clause in the Doom licence, id Software agreed to modify the licence to GPL.

Cuartielles: The cool thing with this game is that people were making their own mods. I remember a version of Wolfenstein 3D with Star Wars characters.

Facchin: I don't remember that.

Cuartielles: I am that old. I remember for the both of us.

Facchin: Well, I recall playing Wolfenstein 3D when I was a child. But we didn't have an Internet connection, so we couldn't get all of these goodies from the modders' community.

Cuartielles: **Before we continue, let me ask you first. Could you introduce yourself?**

Facchin: [laughs] Martino Facchin, firmware engineer at Arduino.

Cuartielles: **What is your role in Arduino?**

Facchin: I am head of firmware — the main guy when you need some firmware help. Now, I have a wonderful team of colleagues that help me with this, because I started this team just by myself. The team keeps on growing. We try to also make the community grow with us by making everyone more aware of what we are doing.

Cuartielles: Besides making Doom run on a Portenta (something we will talk about more later), what is the thing you made at Arduino that makes you the most proud?

Facchin: I'd say that it's the PluggableUSB framework. When I just came to Arduino, having been there for six months, we had this big issue of people willing to add multiple functionalities to the USB port of the Arduino Due and the — by then — forthcoming Arduino Zero. Every time people were plugging an Arduino Leonardo into a computer, it was bringing up the keyboard drivers, the mouse drivers, etc., even if you were not using them. We had to build this thing on the fly, a USB descriptor that would help the users see just the things they wanted to use at the time. At once, we also allowed for other things to happen, things that people wanted to use such as USB MIDI and the like. For me, this was a huge development. I was pretty young at the time and had to interact with the community with the help of Matthijs Kooijman (read more about his work at www.stderr.nl) and Paul Stoffregen (creator of Teensy) to sort out the best possible strategy. And it worked. Every now and then, people come and say, "I used the MIDI library for this or that," or there is even a developer now making all of the layouts for all international keyboards built on top of the foundations of that code. I am very proud of that.

Cuartielles: **You said that you were "pretty young at the time." For how long have you been working for Arduino?**

Facchin: Six and a half years, now. Based at the Torino office.

Cuartielles: **That's quite some time. And how many people are involved with the Firmware team?**

Facchin: Six people. It might seem like a lot, but at the Firmware team, we maintain all of the products, while also performing other activities such as certification. On the other hand, we have

The obvious choice was to port Doom and try to instrument all the features that were needed for it to run.

separated Firmware from Tooling, which is a different team, dedicated to the Arduino CLI and other parts of the higher end software.

Cuartielles: **And all developers work against GitHub, correct?**

Facchin: Yes, everything is open sourced by the end of the development process.

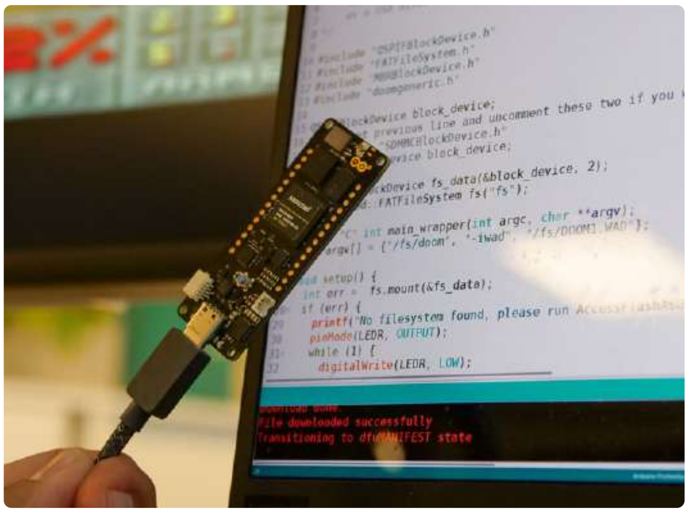
Cuartielles: **Great. Let's go back to talk about Doom. We know that it was a very successful game that sold over 3.5 million units, making their developers filthy rich. It was the first best-seller first-person shooter (FPS) game. The code was open sourced and was ported to all kinds of devices.**

Facchin: Phones, calculators, every operating system.

Cuartielles: **And Arduino's Portenta H7, a dual core processor board aimed at industrial environments. Who had the idea of running Doom on the Portenta H7?**

Facchin: It was more that we had just got the first prototype of the Portenta H7, a nice board with a lot of chips, and everything was still to be done. We had this chip from Analogix that you can typically find in other devices that converts MIPI signals into DisplayPort, which would allow it to address external monitors. We had no experience in the subsystems in the chip that should be handling this, and the existing examples were not helping either. At first, we managed to render some yellow boxes on the screen, then the Arduino logo with a few artefacts, but it was far from perfect. We were not really understanding why things were not working as expected, so I went for something with moving images that I could easily recognise.

The obvious choice was to port Doom and try to instrument all the features that were needed for it to run. It took very low effort to make it work on something without an operating system. It is not really the fork of Doom you would play, but it is very easy to port. You just need to change six or seven functions to adapt it to your hardware and you are ready to go. To begin with, we had no code to run the internal RAM or the external memory. I had to get those to work first and then fire up the simulator. From the beginning, we



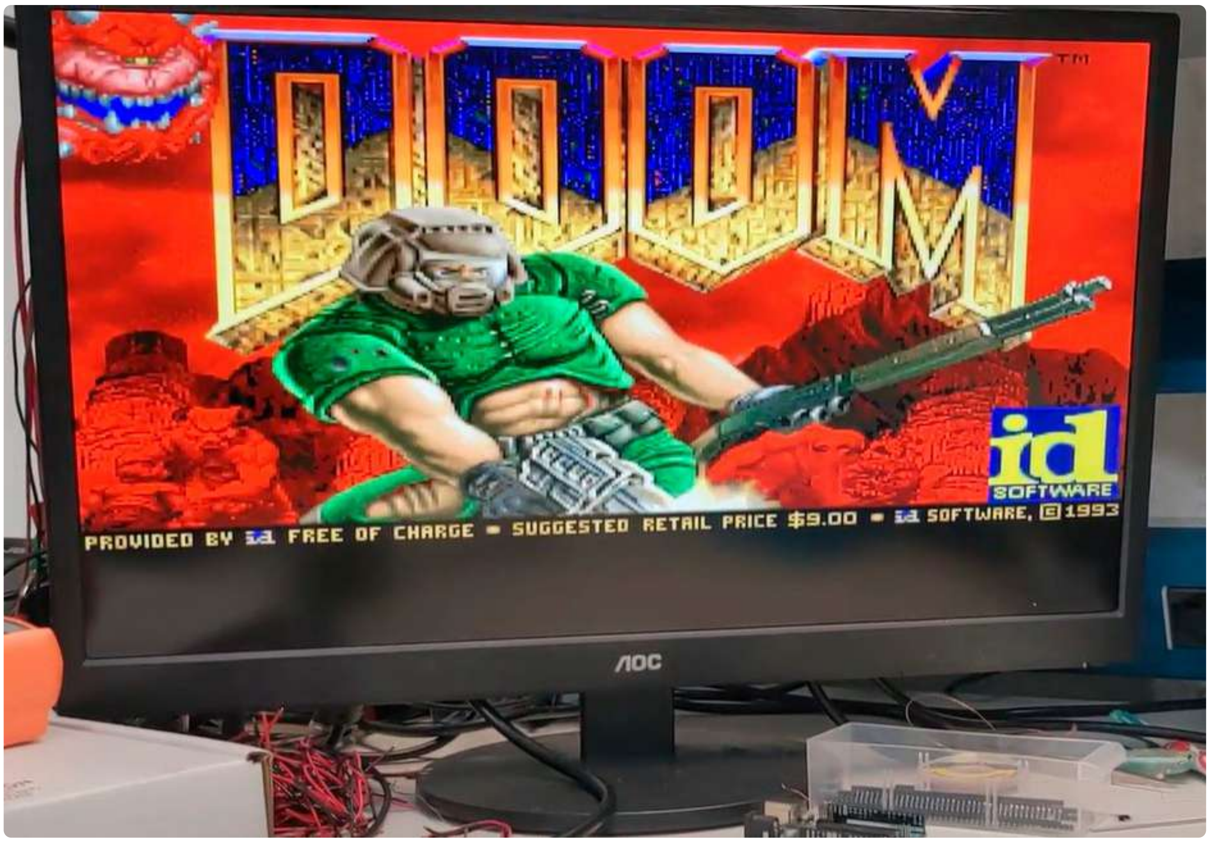
```
#include "QSPIFBlockDevice.h"
#include "FATFileSystem.h"
#include "MBRBlockDevice.h"
#include "doomgeneric.h"

BlockDevice block_device;
 previous line and uncomment these two if you
 "SDMMCBlockDevice.h"
 evice block_device;

kDevice fs_data(&block_device, 2);
d::FATFileSystem fs("fs");

"C" int main_wrapper(int argc, char **argv);
argv[] = {"/fs/doom", "-iwad", "/fs/DOOM1.WAD"};

setup() {
 int err =  fs.mount(&fs_data);
 if (err) {
   printf("No filesystem found, please run AccessFlashAs
   pinMode(LEDR, OUTPUT);
   while (1) {
     digitalWrite(LEDR, LOW);
```

```
File downloaded successfully
Transitioning to dfuMANIFEST state
```

It is up to the reader to take over the project and add controls to make the game playable.

How to run Doom in your Portenta H7

1. Download the latest version of the Arduino IDE. We recommend Arduino 2.0 or more recent.
2. Download the Portenta core from the board manager.
3. Select the M7 core for your Portenta H7 board. All of the following steps have to be executed on that core.
4. Make sure the IDE has identified the port here the board is connected to.
5. There is an example at "Examples / Doom" where you can see the basic instructions. Before you install this, you need to run a couple of examples on your board.
6. [Optional] Upgrade your Portenta H7 bootloader to the latest version with "Examples / STM32H747_System / STM32H747_manageBootloader".
7. Format the external Flash using the example "Examples / STM32H747_System / QSPIFormat." After the installation, you will have to open the serial terminal and follow the instructions given to you in it.
8. Transform your board into a mass storage device as if it was a USB drive with "Examples / USB As Mass Storage / AccessFlashAsUSBDisk."
9. Open the Serial monitor and choose how the formatting of the board should proceed. Once it is done, your computer should register two new external drives connected to it.
10. Download DOOM1.WAD from the DoomWiki site at: https://doomwiki.org/wiki/DOOM1.WAD and copy it in the largest partition of the virtual Portenta drive.
11. Go back to the Doom.ino example we saw on step 5 and flash it on your board, remember: always on the M7 core. If you had problems seeing the programming port, just double click the reset button on your Portenta prior to uploading and make sure the serial port is properly recognized.
12. Disconnect the Portenta H7 from your computer and plug it to a USB-C hub as if it was a laptop. The hub will need to have the external power connected and an HDMI cable to send the video signal to a computer monitor.


DOOM
TM
id SOFTWARE
PROVIDED BY id FREE OF CHARGE - SUGGESTED RETAIL PRICE $9.00 - id SOFTWARE, © 1993
AOC
Doom is a classic!

didn't get video to work, I had to prepare a framebuffer, do some magic, and get some proper output over USB-C.

Cuartielles: **The Portenta H7 is a dual-core processor. It has an Arm Cortex M4 and an M7 inside the processor. Which one was the one running Doom?**

Facchin: Today we use the M7, but back then we ran it on the M4, because it was much simpler from an embedded programmer point of view. It is more like a typical microcontroller, without any special features. On the other hand, the M7 has this cache that you have to look into and invalidate at the right time when generating video. The first time I tried to run things on the M7, it was real fast, but also completely broken, and I could not see anything on the screen. The M4 was fast enough (25 frames per second) and pixel perfect.

Cuartielles: **25 fps is a lot faster than my first computer. But let's summarise this for the reader: You had Doom running in the slowest of the two processors, on a board which has Bluetooth, Wi-Fi as connectivity. Video is sent out via USB-C. There you can have a hub, mouse, keyboard, whatever. Which were the controls you implemented there?**

Facchin: Unfortunately, the project died there for us, because once we saw we could make video run, we started working with LVGL, which is a much more useful library for other developers to build applications on top of Portenta. LVGL is completely integrated with the USB hub, keyboard and mouse, so that you can build all needed interfaces for the professional context that many of Arduino's end users need.

Cuartielles: **Wouldn't it be cool to have a professional PLC based on Portenta H7 where you could be playing Doom on the Cortex M4 core while doing the serious work on the M7?**

Facchin: Absolutely!

Cuartielles: **Thanks, Martino. It was great to hear the story of Doom running on Arduino Portenta H7. We will share with our community the basic instructions on how to get it up and running. It is up to the reader to take over the project and add controls in order to make the game playable.**

220542-01

About the Author

David Cuartielles co-founded Arduino. He holds a PhD in Interaction Design and an MSc in Telecommunications Engineering, and he teaches at Malmö University.

Questions or Comments?

Do you have any questions or comments relating to this article? Contact the team at Elektor at editor@elektor.com.

Related Products

> **Arduino Portenta H7**
www.elektormagazine.com/arduino-portenta-h7

20% discount on the first year of your membership

Join the Elektor Community

Take out a GOLD membership or a GREEN membership!

Gold	Green	
✓	✓	The Elektor web archive from 1974!
✓	☐	8x Elektor Magazine (print)
✓	✓	8x Elektor Magazine (digital)
✓	✓	10% discount in our web shop and exclusive offers
✓	✓	Access to more than 5,000 Gerber files
✓	✓	Free shipping within US, UK & Ireland

www.elektormagazine.com/arduino-member

Use the coupon code:

ARDUINO22

Unboxing the Elektor **LCR METER** with **David Cuartielles**

Save the date: January 26, 2023

Would you like to unbox the Elektor LCR Meter Kit with Arduino co-founder David Cuartielles? Watch the January 26, 2023 (18:00 CET) episode of *Elektor Lab Talk*, where he will join Elektor engineer Mathias Claussen and Editor Jens Nickel to discuss the LCR Meter Kit, as well as take your questions about the Arduino technology and this guest-edited edition of *Elektor*. Don't miss the livestream. Bring your questions!

220555-01

Elektor LabTalk

Watch David live on Elektor Lab Talk on January 26, 2023!

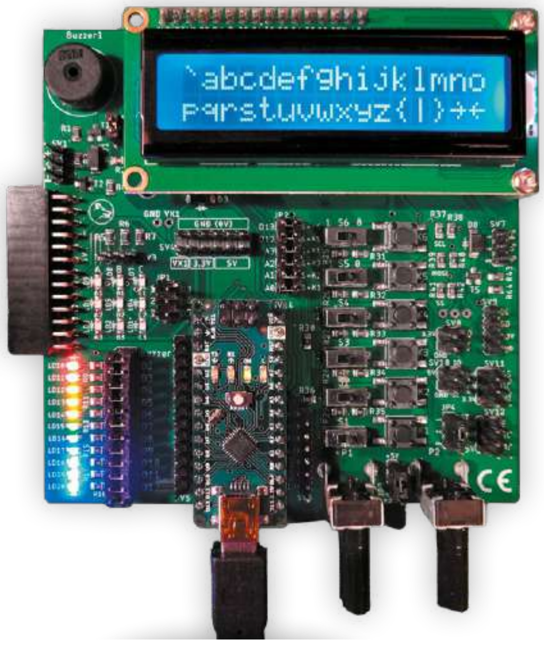
www.elektormagazine.com/labtalk-david

Crash Course Into the Arduino World

Development Board for the Arduino Nano

By Wolfgang Trampert (Germany)

Elektor remains true to its educational mission: here we present a brand new training board with an Arduino Nano at its heart. Together with a well-structured, hands-on training course, it provides an ideal platform to upgrade your skills and to explore the world of microcontrollers.



Editor's Note
At the time of publication, the book associated with this kit is available in German only. Translations are planned for the near future. Once finished, the translated version will be available in the Elektor Store.

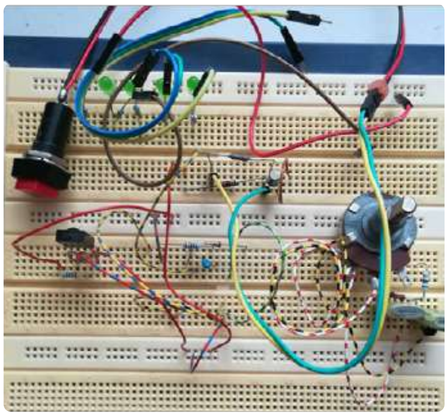


Figure 1: Typical plugboard circuit layout for an Arduino sketch.

You might say the "Arduino philosophy" is associated with a hardware-near design approach: in most cases, the Arduino software or sketch accesses components such as switches, pushbuttons, potentiometers, LEDs, LC displays, piezo buzzers, driver transistors etc via the microcontroller GPIOs. Other types of peripheral devices or electronic modules such as sensors, display or driver boards use various serial interfaces such as SPI, I²C or 1-wire bus to communicate with the controller. In order to familiarize yourself with the world of microcontrollers and Arduino boards, you need to build new practice circuits and control them using an Arduino board.

The essence, however, of any project based on Arduino hardware is development of the software (sketch). The hardware is only a means to an end. You can of course connect up all the peripherals you need for a particular project using a prototyping plug board (**Figure 1**). This approach gives you maximum freedom to install peripherals and connect signals wherever you want. That's not always a good idea, especially when you are just starting out. The layout can quickly grow in size so you end up with a rat's nest of Dupont jumper leads. Often you'll spend more time debugging the hardware and sorting out wiring errors than actually writing code. This only increases the frustration level, and correcting dumb errors won't necessarily teach you anything useful.

The MCCAB Training Board

To get around these hurdles, we have developed the Elektor Arduino training board,

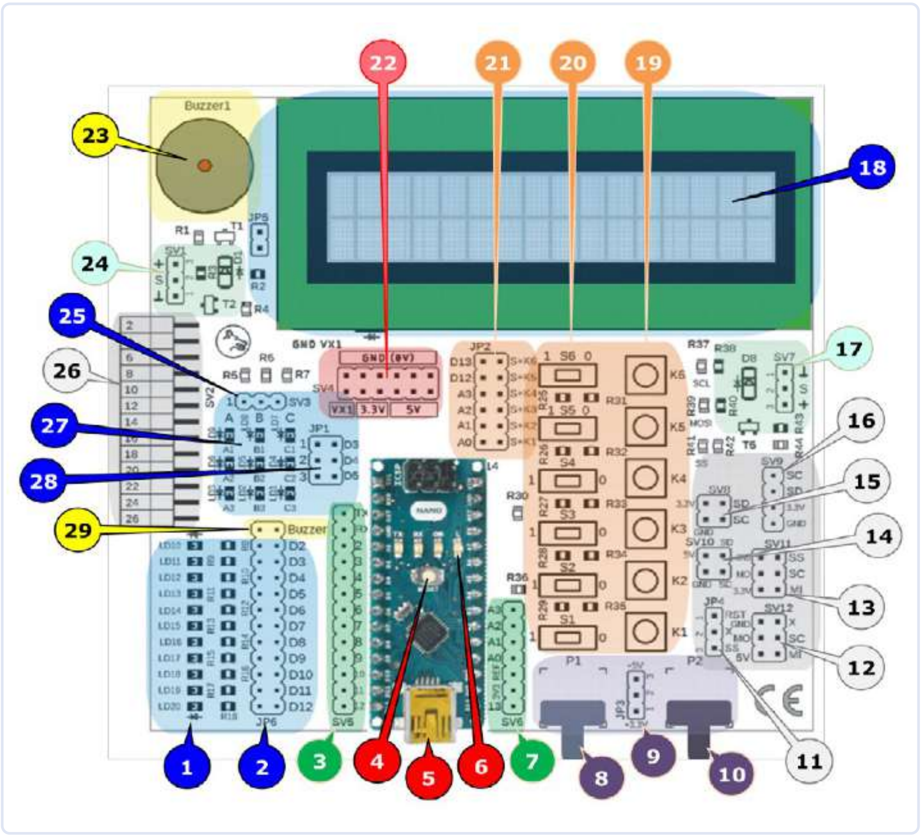


Figure 2: The MCCAB training board, Rev. 3.3.

also known as the *MCCAB Training Board* (**Figure 2**). At its heart is an Arduino Nano board which plugs onto the MCCAB training board. Alongside this are many of the basic peripheral devices you would generally need to build a new prototype for many applications such as a lab setup, test and experimental circuits, projects and exercises to support your studies and training and also hobby projects. The microcontroller GPIOs are all available on two pin header strips on the *MCCAB Training Board* which gives the board maximum deployment flexibility. Additional peripherals or external signals can be hooked up using Dupont jumper leads as required. You won't need to worry about incorrectly wiring the on-board peripherals and will spend less time rummaging through boxes of spare parts to find that elusive component you need to complete a circuit.

Additional circuits built on breadboards can also be easily connected using Dupont cables, since all GPIOs of the microcontroller on the Arduino Nano are connected to the two header strips SV5 and SV6 on the MCCAB training board (pointers 3 and 7 in **Figure 2**). Running down the left hand edge of the board you can also see the double row 26-pin right-angled pin header connector SV2 (pointer 26 in **Figure 2**) where an external expansion PCB can be plugged in.

This connector provides all the important GPIO signals from the microcontroller. External boards to implement functions such as an electronic component curve tracer, a lab power supply or a traffic light controller can be docked to the MCCAB training board here and be controlled by it. Information and results from a running sketch can be written to the on-board 2×16 character LCD (pointer 18), which interfaces via the board's I²C bus. Also on board is a 3×3 LED matrix (pointer 27).

The MCCAB training board is powered by a supply of Vcc = +5 V. This will usually be provided by the USB cable plugged into your PC which you need for creating and uploading the exercise sketch to the MCCAB. The MCCAB can also be powered from an external power pack.

In the Training Board schematic (**Figure 2**), all components associated with a specific board function are identified using a common background colour.

The *MCCAB_Lib* Library for use with the Training Boards

Software development involves using the Arduino IDE to write the program (or 'Sketch' in Arduino speak) which tells the microcontroller how to behave. The sketch is then compiled

The MCCAB training board controls and indicators

1	11 × LED (status indication of input/outputs D2 to D12)
2	Connector linking LEDs LD10 to LD20 with GPIOs D2 to D12
3	Microcontroller inputs and outputs
4	*RESET* button
5	Arduino NANO with mini USB socket
6	LED *L*, linked to GPIO D13
7	Microcontroller GPIOs
8	Potentiometer P1
9	Supply voltage to P1 and P2
10	Potentiometer P2
11	Signal at pin X of SV12
12	SPI-Interface 5 V (The signal at pin X is selected by JP4)
13	SPI-Interface 3.3 V
14	I²C-Interface 5 V
15	I²C-Interface 3.3 V
16	I²C-Interface 3.3 V
17	Switch output for external equipment
18	2×16 character LCD
19	6 × Pushbuttons K1 to K6
20	6 × slider switches S1 to S6
21	Pin header to link switches to microcontroller GPIOs
22	Supply voltage distributor
23	*Buzzer1*
24	Switch output for equipment
25	3×3 LED matrix columns
26	2×13 pin header strip to connect an external module
27	3×3 LED matrix (red)
28	Connections of 3x3 matrix rows to D3, D4 and D5
29	Jumper position links *Buzzer1* to GPIO D9

Table 1: Classes available in the MCCAB_Lib library.

Class	Usage
KeySwitch	Debounced status of switches S1 to S6 and pushbuttons K1 to K6
Matrix	Control of the 3×3 LED matrix.
LED	On / off / blink control of the 12 LEDs LD10 to LD20 and LED
LedBlock	Output a bit pattern on all 11 LEDs (LD10 to LD20)
Sound	Control of Buzzer1 and square wave signal generator.

and uploaded to the Arduino Nano's microcontroller on the training board via a mini USB cable.

The microcontroller GPIOs can be configured as usual using the Arduino function `pinMode()` and the value of signals to and from components on the training board can be read or controlled using `digitalRead()`, `digitalWrite()`, `analogRead()` etc.

However, a library called *MCCAB_Lib* [1] is available which supports the developer by providing additional commands to control the extensive hardware peripherals on the MCCAB training board. The library can be downloaded free of charge and integrated into your own sketch. This library makes handling the on board peripherals much easier.

The library *MCCAB_Lib* contains five classes for controlling the switches, LEDs and buzzer on the training board and can be easily included into the user's sketch as required. **Table 1** shows a list of the classes available.

Using this library means the user does not have to worry about defining time periods for switch debouncing, generating multiplex control signals for the 3×3 LED matrix and flashing the LEDs *LD10* to *LD20* or *L*, or even generating the buzzer tone frequencies. The library functions do this automatically in the background of the program flow, unnoticed by the user.

Listing 1 is a small example sketch to demonstrate use of the *MCCAB_Lib* library.

In line 15 of the sketch, the object variable `Led` is declared of the Class `LED` from the library *MCCAB_Lib*. The parameter `LED_PIN` passed in the declaration of the object variable `Led` is defined as a constant in line 13 indicating the pin to which the LED is connected. This pin is automatically configured as an output during instantiation.

The object variable `Key` of the class `KeySwitch` from the *MCCAB_Lib* library (declared in line 22) during execution monitors (in the background) the state of the switch input on

Listing 1.

```
/*
* Sketch which uses pushbutton K4 to toggle LED LD10 on and off using object variables in the
  “KeySwitch” und “LED” classes in the MCCAB_Lib library.
* To read the status of pushbutton K4 its necessary to insert a jumper to link position S+K4 (the switch
  connection) with A3 (GPIO A3 of the microcontroller)on double header strip JP2 of the MCCAB.
* Insert another jumper (in position D2 of the double header strip J6) to link LED LD10 with the
  microcontroller GPIO of the MCCAB.
*/

11 #include <MCCAB_Lib.h> // bind the MCCAB_Lib Library to the Sketch
12
13 #define LED_PIN  2     // the LED is connected to pin D2
14
15 LED Led(LED_PIN);      // Object-Variable
16
17 //function called by the object-variable “Key” when the switch is closed.
18 void switchTurnedOn() {
19   Led.toggle();        // toggle or flip the state of the LED
20 }
21
22 KeySwitch Key(SK4, ACTIVE_HIGH, switchTurnedOn, nullptr);  // Object-Variable
23
24 void setup() { }   // nothing to do here...
26 void loop() { }    // or here
```

pin SK4, which is passed to it as a parameter according to its declaration. It performs switch debouncing when the pushbutton K4 is pressed or released and calls the function `switchTurnedOn()` when the button is pressed. The `toggle()` method of the class `LED` from the *MCCAB_Lib* library is activated in the `switchTurnedOn()` function in line 19 to invert the current state of the LD10 light-emitting diode.

Since the connection pins for the switch and LED are automatically configured as input and output when the objects are declared nothing else needs to be done in the `setup()` function in line 24 in this sketch.

The `loop()` function in line 26 also does not contain any instructions because the only action to be performed in this sketch is to switch the LED state when the K4 button is pressed. This action is event-driven by the `KeySwitch` class by calling the `switchTurnedOn()` function.

Using these classes in the library *MCCAB_Lib* in more extensive sketches, the two functions `setup()` and `loop()` of the standard Arduino software model would not need be required to continually poll the state of peripheral components, thereby freeing them up for more important tasks.

12 Project Sketches and 46 Exercises

A detailed instruction manual for the MCCAB training board is available and can be downloaded from the website [1]. The MCCAB training board and the *MCCAB_Lib* library will also be described in detail in an upcoming book which will be available shortly (see editor's note).

The book explains in detail the hardware and software basics of a microcontroller system and introduces the programming language C, which is used to write Arduino sketches.

The book's principle focus is on practical exercises, so that "learning by doing" is the key concept used here to acquire the skills you will need when you go on to build your own projects. In a comprehensive practical section, there are 12 project sketches and 46 exercises, so that your knowledge builds as you work through the many examples. The exercises are structured in such a way that the reader is given a task that needs to be solved with the MCCAB training board using the knowledge gained from the theory section of the book. For each exercise there is then a detailed explanation and well-commented example solution that help solve problems.

220450-01

Questions or comments?

If you have technical questions, feel free to email the Elektor editorial team at editor@elektor.com.

About the Author

Wolfgang Trampert has been developing and programming microcontroller systems since he finished his studies in electronics. His engineering business developed microcontroller based solutions to meet customer requirements. He has authored a number of specialist books and articles and conducts training courses on the subject of microcontrollers.

Related Products

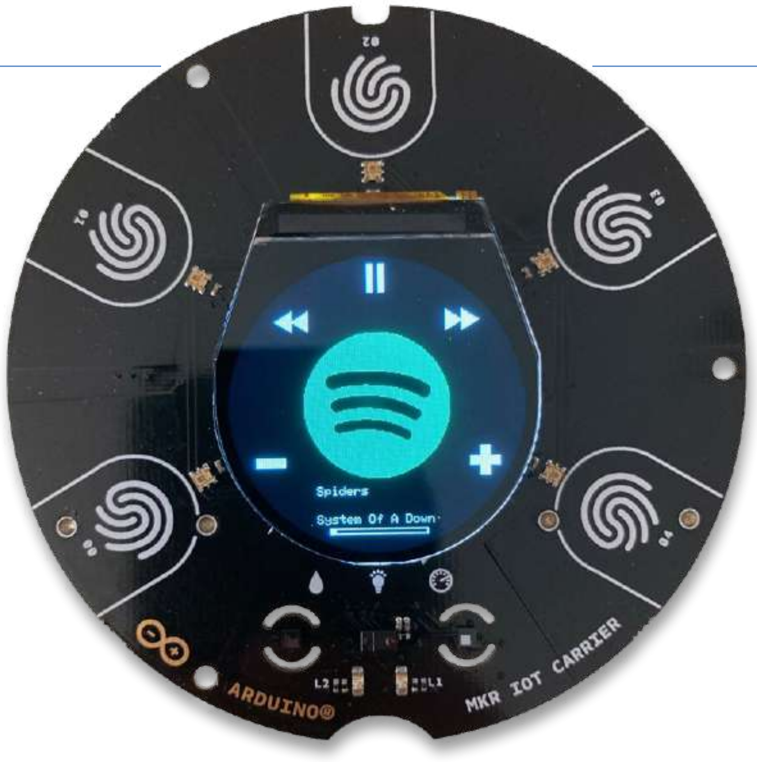
- **MCCAB Training Board (SKU 20295)**
  www.elektor.com/20295
- ***Mikrocontroller-Praxiskurs für Arduino-Einsteiger* (Book in German, SKU 20293)**
  www.elektor.de/20293

WEB LINK

[1] The MCCAB_Lib Library: http://www.elektor.de/20295

A Controller For Spotify

The Oplà IoT Kit Is (Almost) All You Need



By Altuğ Bakan (Turkey)

The Arduino Oplà IoT kit contains the MKR WiFi 1010 Maker Board and a carrier board that integrates relays, a round-shaped OLED display, capacitive touch buttons, and some sensors. Here we describe how to build a portable controller for the popular Spotify music player. Of course, some security is needed.

The Arduino MKR WiFi 1010 Maker Board is — thanks to its Wi-Fi capabilities — a perfect brain for your next IoT project. You are even better equipped with the Arduino Oplà IoT kit, which contains this Maker Board and a carrier board (**Figure 1**). The latter integrates relays, a round shaped OLED display and capacitive touch buttons.

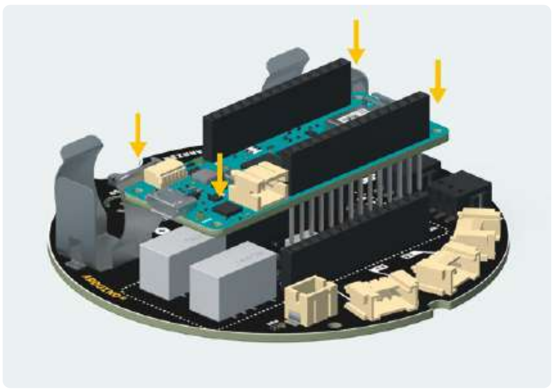


Figure 1: The MKR WiFi 1010 Maker Board is put on the carrier board, which integrates relays and other useful peripherals.

Also in the kit is a moisture and a PIR sensor (**Figure 2**). Projects such as home security alarms and automatic plant watering are therefore easy to implement.

The Wi-Fi feature also allows you to control programs running on your PC, if they have a network interface. The touch buttons, the display, the battery socket, and a housing make it easy to design a portable controller for different kinds of PC software — as an addition to a mouse and keyboard (**Figure 3**).

I am a fan of the music player Spotify, and so I used the Oplà kit to build my own wireless Spotify controller. You can press buttons to skip to the next and previous song, play/pause a song, and increase and decrease the volume. To do so, it goes without saying that the Spotify player must be started on your PC or smartphone.

Secure Communication

Spotify comes with an easy-to-handle programming interface to control your Spotify player via the network; however, you will need the Spotify Plus license for it. Of course, some security is needed. To use the Spotify web API, which is based on REST, you have to authenticate first at the Spotify Accounts Server with your Spotify login username and password. Once authenticated, your software has to send a Client ID and a Client Secret. The Spotify server will return an access token, which you have to send with each call of the Web API to control your Spotify player. This two-step authentication flow is based on the popular *OAuth2* process (see **Figure 4**).

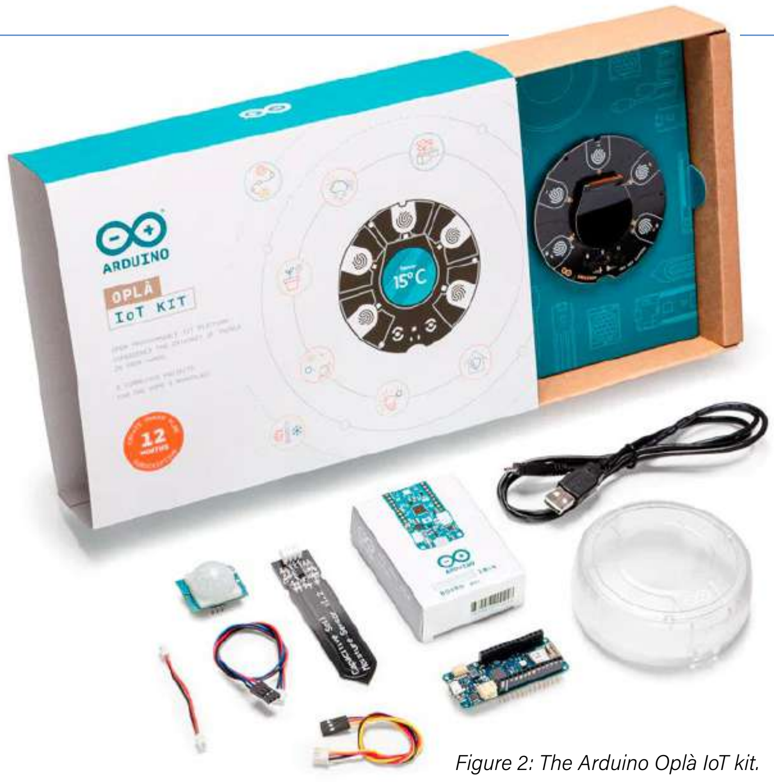


Figure 2: The Arduino Oplà IoT kit.

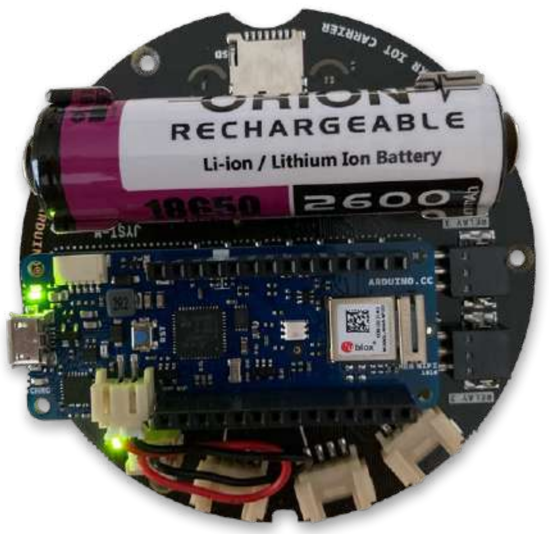

▲

Figure 3: The battery socket makes the Oplà IoT Kit portable.

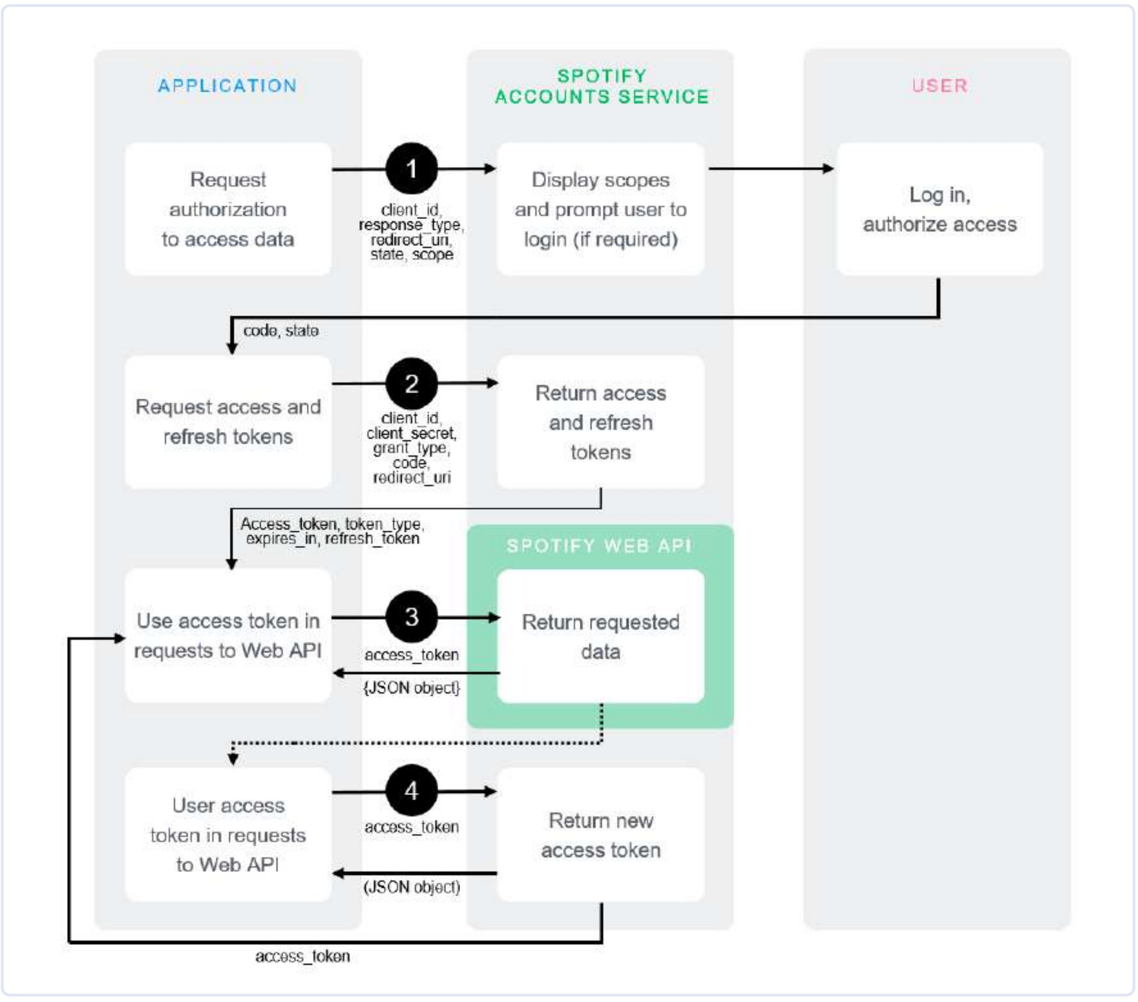
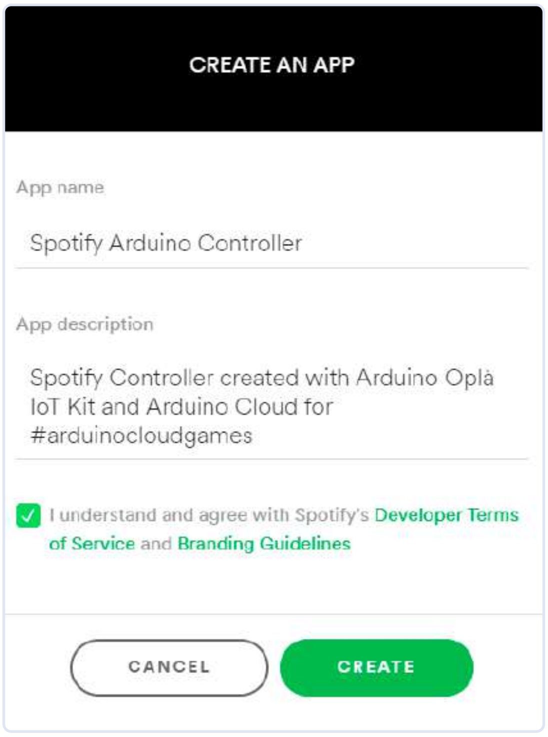

APPLICATION
SPOTIFY ACCOUNTS SERVICE
USER
Request authorization to access data
1
client_id, response_type, redirect_uri, state, scope
Display scopes and prompt user to login (if required)
Log in, authorize access
code, state
Request access and refresh tokens
2
client_id, client_secret, grant_type, code, redirect_uri
Return access and refresh tokens
Access_token, token_type, expires_in, refresh_token
SPOTIFY WEB API
Use access token in requests to Web API
3
access_token
{JSON object}
Return requested data
User access token in requests to Web API
4
access_token
{JSON object}
Return new access token
access_token

Figure 4: The multi-step-authentication flow is based on the popular OAuth 2.0 process.

◀



CREATE AN APP

App name

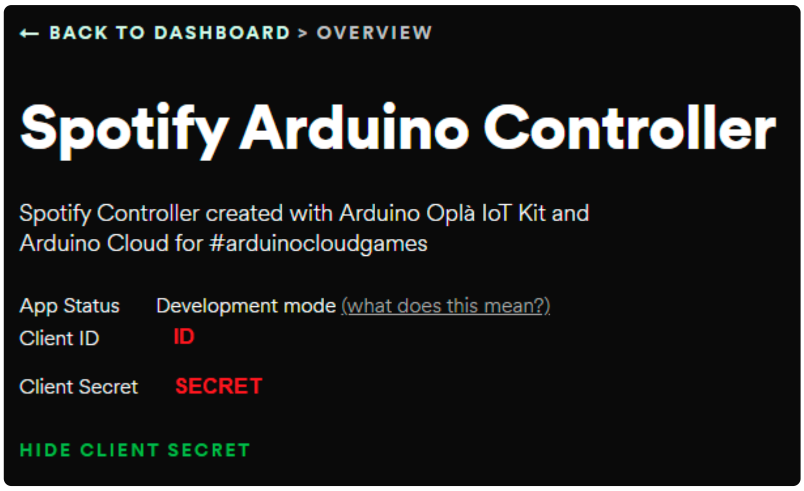
Spotify Arduino Controller

App description

Spotify Controller created with Arduino Oplà IoT Kit and Arduino Cloud for #arduinocloudgames

✓ I understand and agree with Spotify's Developer Terms of Service and Branding Guidelines

Figure 5: You have to create an "App" to get ...



Spotify Arduino Controller

Spotify Controller created with Arduino Oplà IoT Kit and Arduino Cloud for #arduinocloudgames

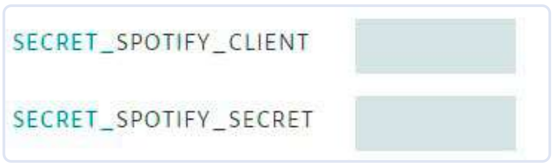
App Status Development mode (what does this mean?)
Client ID ID
Client Secret SECRET

Figure 6: ...your Client ID and Client Secret.

How do you get your Client ID and Secret? Just use the Spotify App Builder [1], which you can use to design your own PC software or mobile App to control Spotify (**Figure 5**). However, we don't do this here; we just want the credentials (see **Figure 6**). Client ID and Secret must be stored on our Arduino MKR Board. Of course, you could do this hardcoded in the Arduino sketch, but there is a more comfortable and more secure way to do so. The Arduino Web Editor [2] provides a *Secrets* tab, where you can set environmental variables to be later used in your code (**Figure 7**). Just enter the Spotify Client ID and the Secret as well as your Wi-Fi network name and password in the fields of the tab. If you compile and upload the software to the controller, your individual secret values will be also uploaded to be used by the project's code. In your sketch you have to replace the strings containing sensitive data by writing a `SECRET_xxx` expression — so, for example: `SECRET_SPOTIFY_CLIENT`.

Authentication

To start the OAuth2 flow, you have to authenticate at Spotify. When starting the Spotify controller described here, it will log in the specified (home) Wi-Fi network and show the IP address it got by the router on the OLED display. I wanted to give the user a chance to easily authenticate at Spotify, so I created the following approach. The Arduino Controller generates a



SECRET_SPOTIFY_CLIENT
SECRET_SPOTIFY_SECRET

Figure 7: Enter all private values in the **Secrets** Tab of the Arduino Web Editor, before compiling and uploading the code.

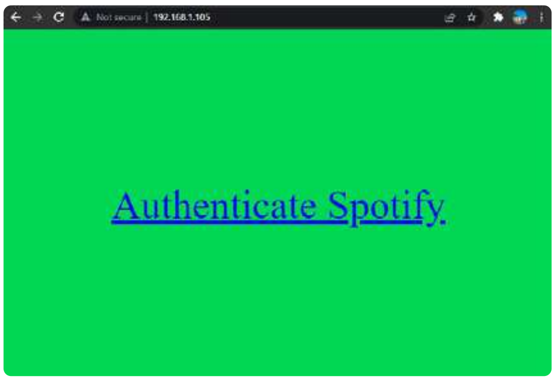


Figure 8: Webpage offered by the controller to log in at Spotify.

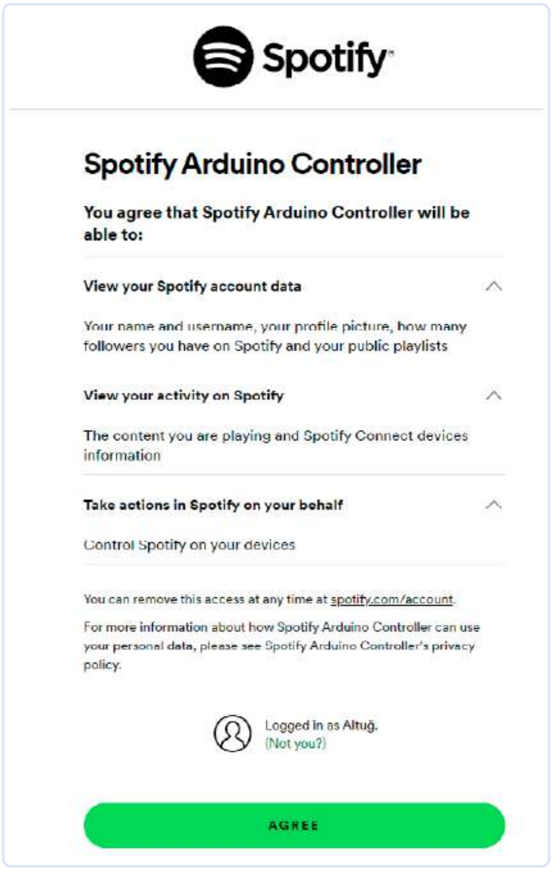
small webpage which will be shown in a web browser, when you enter the IP address of your controller there (**Figure 8**). This small webpage contains a weblink. (Refer to **Listing 1** to see how the webpage is generated in the Arduino code.) If pressed, the browser goes to the authentication page at Spotify, where you easily can log in. You will then be asked if you give the controller permission to control Spotify (**Figure 9**).

Please note: To get all this working, you also have to enter the IP address of the controller as a "*Redirect URI*" in the Spotify App editor (**Figure 10**).

From now on the Arduino Controller can get the API access token with sending the Client ID and Secret to Spotify (**Listing 2**). The access token must be regularly refreshed during operation. This is also done by a function in the sketch (**Listing 3**), which is called every 3000 seconds.

Listing 1: Webpage to authenticate at Spotify, offered by the Spotify controller.

```
String webpage = "<!DOCTYPE html>\n";
webpage += "<html><body>";
webpage += getStyle();
webpage += "<a href=\"https://accounts.spotify.com/authorize?client_id=";
webpage += SPOTIFY_CLIENT;
webpage += "&response_type=code&redirect_uri=http://";
webpage += ip_address;
webpage += "/redirect/&scope=user-read-playback-state user-modify-playback-state\">Authenticate Spotify</a>\n";
webpage += "</body></html>";
wifiClient.print(webpage);
```



Spotify

Spotify Arduino Controller

You agree that Spotify Arduino Controller will be able to:

View your Spotify account data

Your name and username, your profile picture, how many followers you have on Spotify and your public playlists

View your activity on Spotify

The content you are playing and Spotify Connect devices information

Take actions in Spotify on your behalf

Control Spotify on your devices

You can remove this access at any time at spotify.com/account.

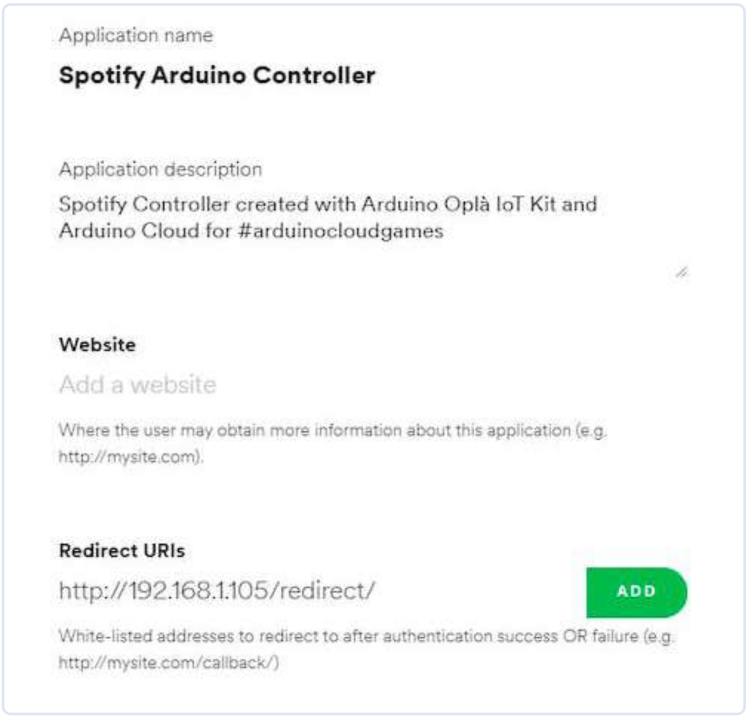
For more information about how Spotify Arduino Controller can use your personal data, please see Spotify Arduino Controller's privacy policy.

Logged in as Altuğ.
(Not you?)

AGREE

Figure 9: If you are logged in at Spotify, you have to give the controller permission to act on your behalf.

Figure 10: Redirect URL: The address of the controller in your home network.



Application name

Spotify Arduino Controller

Application description

Spotify Controller created with Arduino Oplà IoT Kit and Arduino Cloud for #arduinocloudgames

Website

Add a website

Where the user may obtain more information about this application (e.g. http://mysite.com).

Redirect URIs

http://192.168.1.105/redirect/

ADD

White-listed addresses to redirect to after authentication success OR failure (e.g. http://mysite.com/callback/)

Listing 2: Function to get the token from Spotify for further use of the API.

```
// Get the user authorization token
 bool getAccessToken(String userCode) {
   String postData = "grant_type=authorization_code&code=" + userCode + "&redirect_uri="
                     "http://" + ip_address + "/redirect/";
   authClient.beginRequest();
   authClient.post("/api/token");
   authClient.sendHeader("Content-Type", "application/x-www-form-urlencoded");
   authClient.sendHeader("Content-Length", postData.length());
   authClient.sendBasicAuth(SPOTIFY_CLIENT, SPOTIFY_SECRET);
       // send the client id and secret for authentication
   authClient.beginBody();
   authClient.print(postData);
   authClient.endRequest();

  // If successful
   if (authClient.responseStatusCode() == 200) {
     lastTokenTime = millis();
     DynamicJsonDocument json(512);
     deserializeJson(json, authClient.responseBody());
     accessToken = json["access_token"].as<String>();
     refreshToken = json["refresh_token"].as<String>();
     return true;
   }
   return false;
 }
```

Listing 3: Function to refresh the token.

```
// Refresh the user authentication token
 void refreshAccessToken() {
   String postData = "grant_type=refresh_token&refresh_token=" + refreshToken;
   authClient.beginRequest();
   authClient.post("/api/token");
   authClient.sendHeader("Content-Type", "application/x-www-form-urlencoded");
   authClient.sendHeader("Content-Length", postData.length());
   authClient.sendBasicAuth(SPOTIFY_CLIENT, SPOTIFY_SECRET);
       // send the client id and secret for authentication
   authClient.beginBody();
   authClient.print(postData);
   authClient.endRequest();

  // If successful
   if (authClient.responseStatusCode() == 200) {
     lastTokenTime = millis();
     DynamicJsonDocument json(256);
     deserializeJson(json, authClient.responseBody());
     accessToken = json["access_token"].as<String>();
   }
 }
```

Listing 4: Example for using the API (next and previous song).

```
// Skip a song towards a given direction
 void skipSong(String direction) {
   apiClient.beginRequest();
   apiClient.post("/v1/me/player/" + direction);
   apiClient.sendHeader("Content-Length", 0);
   apiClient.sendHeader("Authorization", "Bearer " + accessToken);
   apiClient.endRequest();
 }
```

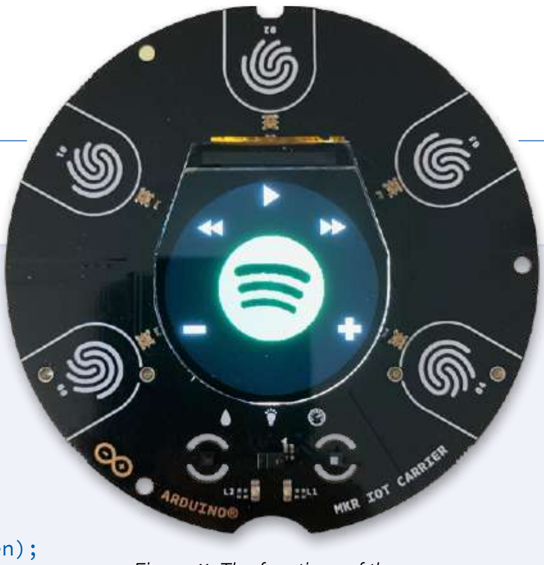


Figure 11: The functions of the buttons are shown on the display.

Operation

The rest of the code is less complex. The device will show the Spotify logo and the function of the buttons on the OLED display (**Figure 11**). If the user touches a button, the corresponding API function is called. Refer to **Listing 4** to see how this is done for skipping a song to the previous or next one.

There is also a function in the code which requests the status of the player from the Spotify API. The answer is a JSON string. I am using the *ArduinoJson.h* library and some of my own functions to process JSON strings more easily.

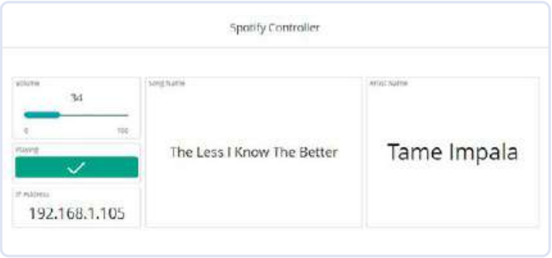
To get the status of the buttons, to control the LEDs, and to show graphics on the OLED, I am using the *Arduino_MKRIoTCarrier.h* library. You can dive into my code to get inspiration for your own projects you can do with the Oplà Kit. My software can be downloaded at [3].

Cloud Connection

I also set up a connection to the Arduino Cloud and created a dashboard that shows the current song and artist name next to the volume of the device (**Figure 12**). Of course you can create your own personal dashboard, with the data you want.

My project won the 3rd place award in the Arduino Cloud Games 2022!

220407-01



Spotify Controller

34

The Less I Know The Better

Tame Impala

192.168.1.105

Figure 12: Current song and artist name are sent to the Arduino Cloud, where they are visible on your personal dashboard.

◄

Questions or Comments?

If you have technical questions feel free to e-mail the author at mail@alt.ug or the Elektor editorial team at editor@elektor.com.

About the Author

Altuğ Bakan has been working as an electronics engineer, mostly with embedded systems. He loves to use Arduino in his work for rapid prototyping and ease-of-use. His favorite electronics subjects are bare-metal embedded programming and Internet of Things (IoT).

Related Products

> **Arduino Oplà IoT Kit**
  www.elektormagazine.com/arduino-opla-iot-kit

WEB LINKS

[1] Spotify App Builder: https://developer.spotify.com/dashboard/
[2] Arduino Web Editor: https://create.arduino.cc/editor
[3] This Project on create.arduino.cc: https://create.arduino.cc/projecthub/Altug/opla-spotify-controller-6e7bc4

Build, Deploy, and Maintain Scalable, Secure Applications

With Arduino Portenta X8 Featuring NXP's i.MX 8M Mini Applications Processor and EdgeLock® SE050 Secure Element

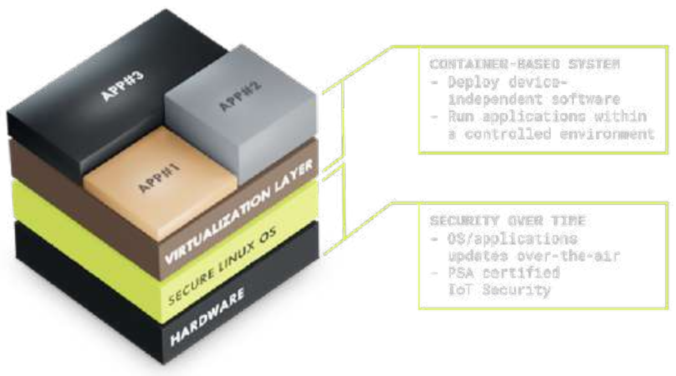
Contributed by NXP Semiconductors

Bringing an IoT device to the market involves significant design and development effort – with scalability issues, security challenges, and device limitations around every corner. Adding intelligence makes it even more complicated. This makes the selection of the right development hardware and software critical to getting secure edge products to market faster. This article introduces the Arduino Portenta X8 platform, an industrial-grade, secure SOM based on NXP's i.MX 8M Mini applications processor and an onboard EdgeLock® SE050 hardware secure element. This PSA-certified platform is also Arm® SystemReady IR for assured security.

Arduino Portenta X8 is a powerful, industrial-grade system on a module with Linux® OS preloaded onboard, capable of running device-independent software because of its modular container architecture. It offers two approaches: flexibility of usage of Linux combined with real-time applications through the Arduino environment. Onboard Wi-Fi/Bluetooth® Low Energy connectivity allows remote OS/application updates, always keeping the Linux kernel environment at top performance levels.

State-of-the-Art Security

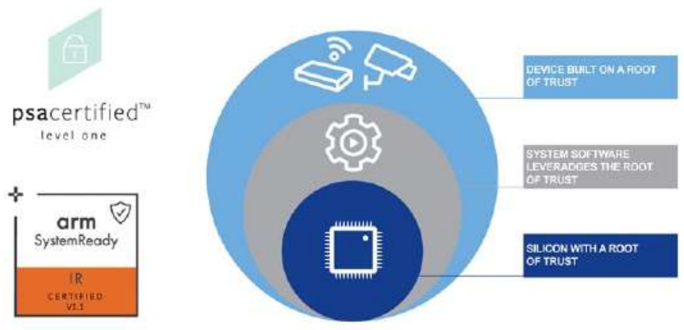
The container-based system integrates different layers of security starting from the hardware layer which includes NXP's Secure Element. It utilizes the cloud-based DevOps platform from Foundries.io [1] to reinvent the way embedded Linux solutions are built, tested, deployed and maintained. The Portenta X8 includes the customizable open-source Linux microPlatform OS, built using best industry practices for end-to-end security, incremental OTA updates and fleet management.


APP#3
APP#2
APP#1
VIRTUALIZATION LAYER
SECURE LINUX OS
HARDWARE
CONTAINER-BASED SYSTEM
- Deploy device-independent software
- Run applications within a controlled environment
SECURITY OVER TIME
- OS/applications updates over-the-air
- PSA certified IoT Security

Portenta X8 Container and Security.

The virtualization layer allows users to deploy device-independent software running within a controlled environment. They can create their own containers using Docker and download premade images from Docker Hub or other public registries available to build a tailored application. If the developer wants to enter the embedded world, they can do so easily by building their application, running it on a container, putting it on the board and testing it out of the box. This provides a wide range of opportunities by mixing the Linux capabilities and the Arduino standard experience.

Portenta X8 achieved PSA Certification and the NXP EdgeLock SE050 hardware secure element provides key generation, accelerated crypto operations and secure storage. X8 also achieved Arm® SystemReady [2] certification and integrated Parsec services, making it one of the first Cassini Products or Cloud Native Edge devices available to developers in the market. It seamlessly runs Fedora IoT, Fedora Server, Debian and Linux microPlatform. Enabling the migration of cloud-native workloads from the Cloud to the edge, the Portenta X8 contributes to a cloud-native developer experience across Arm's diverse and secure IoT ecosystem.


psacertified™
level one
arm
SystemReady
IR
CERTIFIED
V1.1
DEVICE BUILT ON A ROOT OF TRUST
SYSTEM SOFTWARE LEVERAGES THE ROOT OF TRUST
SILICON WITH A ROOT OF TRUST

Platform Security Architecture.

EdgeLock SE050 – A Trust Anchor for IoT

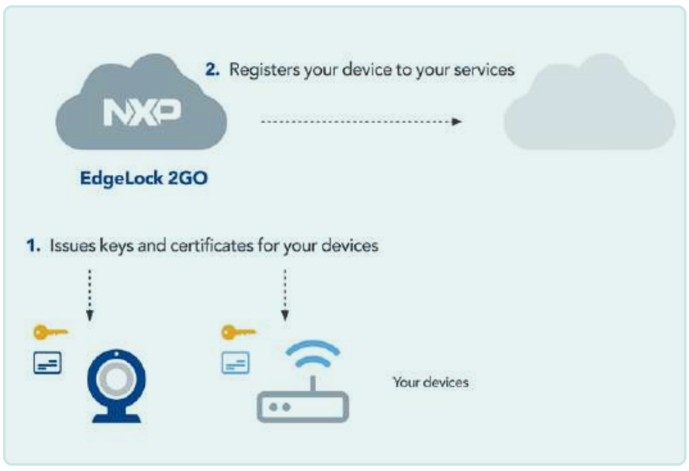
NXP's EdgeLock SE050 [3] is a discrete and tamper-resistant security hardware for protecting the identity of a device, including cryptographic keys and certificates. It's a standalone embedded secure element that is attached to the main processor over the I2C interface. The EdgeLock SE050 is certified Common Criteria EAL 6+ for the hardware and operating system. This ready-to-use secure element for IoT devices provides a root of trust at the IC level and delivers real end-to-end security – from edge to cloud – without the need to implement security code nor handle critical keys and credentials.

Silicon-based Root of Trust: EdgeLock® SE050 Secure Element.

Delivered as a ready-to-use solution, EdgeLock SE050 comes with multiple pre-implemented cryptographic algorithms and protocols and a complete product support package that simplifies design-in and reduces time to market. In addition to libraries for different MCUs and MPUs, the support package also offers integration with the many common OSs including Linux, RTOS and Android.

IoT device designers are facing two major challenges when implementing device onboarding to the cloud: provisioning of the device identity and managing device identities once released to the field. The provisioning of the device refers to the installation of keys and certificates. Managing device identities refers to the update, addition or revocation of keys and certificates throughout the device lifecycle.

To help designers solve these challenges, NXP provides the EdgeLock 2GO [4] managed service. The platform is a purpose-built hardware and service combination that establishes a silicon-based root of trust. EdgeLock 2GO issues the identities required for IoT devices and installs the credentials securely into the EdgeLock SE050 hardware. It also automatically registers the IoT device directly to the cloud service.


NXP
2. Registers your device to your services
EdgeLock 2GO
1. Issues keys and certificates for your devices
Your devices

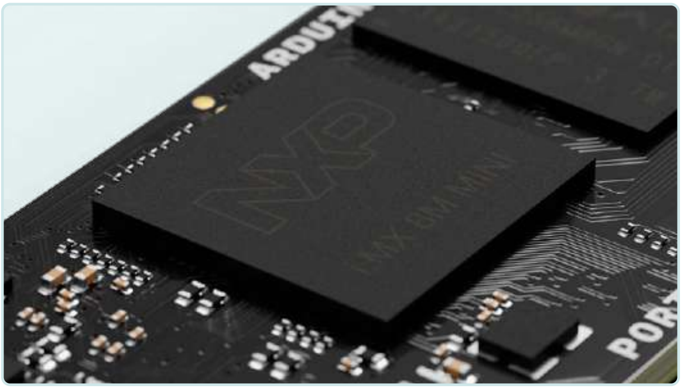
NXP Manages Device Credentials.

This flexible service supports multiple types of credentials and applies different configurations depending on the project. Credentials can be renewed or added to devices released in the field. With the commissioning of EdgeLock SE050 and EdgeLock 2GO, users get an end-to-end solution that is simple, secure and flexible.

As IoT continues to expand, so do the risks. NXP's EdgeLock combination, with its hardware-based security and service for credential management, gives device manufacturers a safer way to do business. With NXP EdgeLock supporting the deployment of a device, it reduces time-to-market and lowers the day-to-day costs of operating an IoT deployment while having the confidence of knowing devices are protected by high-level security.

Unleash the Power: Providing More Speed and Improved Efficiency

The i.MX 8M Mini [5] SoC is NXP's first embedded multicore applications processor built using advanced 14LPC FinFET process technology, providing more speed and improved power efficiency. The i.MX 8M Mini family of applications processors brings together high-performance computing, power efficiency, and embedded security needed to drive the fast-growing edge node computing, streaming multimedia, and machine learning applications.



The i.MX 8M Mini SoC is offered in single, dual and quadcore variants using Arm® Cortex®-A53 operating at up to 1.8 gigahertz per core. Delivered in advanced low-power process, the core complex is optimized for fanless operation, low thermal system cost and long battery life. The Cortex-A cores can be powered off while the Cortex-M4 subsystem performs low-power, real-time system monitoring. The DRAM controller supports 32-bit/16-bit LPDDR4, DDR4, and DDR3L memory, providing great system design flexibility.

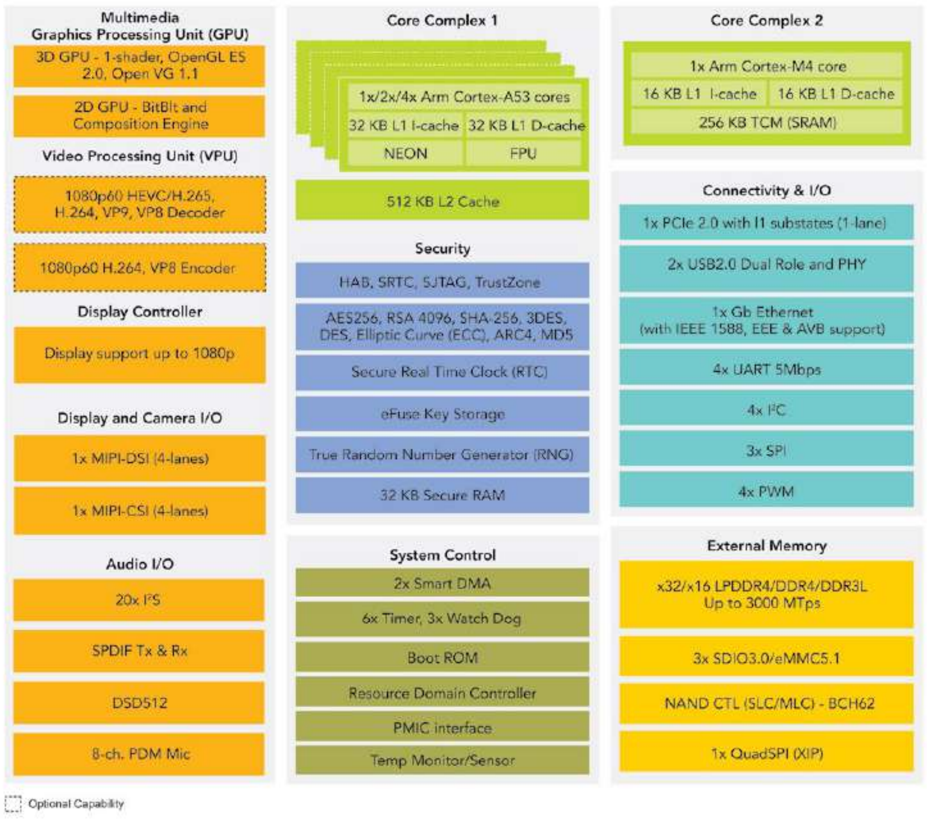
i.MX 8M Mini core options are optimized for ultra-low-power, even sub-Watt in specific applications, but offer the breadth of processing power necessary for consumer, audio, industrial, machine learning training and inferencing across a range of cloud providers. The i.MX 8M Mini SoC also packs-in hardware 1080p video acceleration to enable two-way video applications, 2D and 3D graphics to provide a rich visual HMI experience, and advanced audio capabilities to enable audio-rich applications. An extensive selection of high-speed interfaces enables broader system connectivity and targets industrial-level qualification.

Application Examples Include:

- **Industrial Automation**
  - The Portenta X8 can then act as a multi-protocol gateway, sending data to the Cloud or ERP system via Wi-Fi, LoRa, NB/IoT, LTE Cat.M1.
  - The availability of Linux containers like ROS within the Arduino environment makes the Portenta X8 a great fit for autonomous guided vehicles.
- **Building Automation**
  - Interacting with environmentally smart sensors, Portenta X8 allows the implementation of real-time ML and image processing on the edge.
  - Smart kiosks usually leverage several components (e.g. card readers, cameras, microphones), requiring a diverse selection of I/Os. When combined with a Max Carrier, the Portenta X8 ensures Wi-Fi connectivity and allows administrators to remotely monitor machine usage.
  - The Portenta X8 can simultaneously control HVAC systems, switch on/off smart appliances, autonomously adjust lighting and control accesses on the edge.

Start developing today with the industrial-grade, secure Portenta X8 SOM [6] with outstanding computational density.

220576-01


Multimedia
Graphics Processing Unit (GPU)
3D GPU - 1-shader, OpenGL ES 2.0, Open VG 1.1
2D GPU - BitBlt and Composition Engine
Video Processing Unit (VPU)
1080p60 HEVC/H.265, H.264, VP9, VP8 Decoder
1080p60 H.264, VP8 Encoder
Display Controller
Display support up to 1080p
Display and Camera I/O
1x MIPI-DSI (4-lanes)
1x MIPI-CSI (4-lanes)
Audio I/O
20x I²S
SPDIF Tx & Rx
DSD512
8-ch. PDM Mic
Core Complex 1
1x/2x/4x Arm Cortex-A53 cores
32 KB L1 I-cache
32 KB L1 D-cache
NEON
FPU
512 KB L2 Cache
Security
HAB, SRTC, SJTAG, TrustZone
AES256, RSA 4096, SHA-256, 3DES, DES, Elliptic Curve (ECC), ARC4, MD5
Secure Real Time Clock (RTC)
eFuse Key Storage
True Random Number Generator (RNG)
32 KB Secure RAM
System Control
2x Smart DMA
6x Timer, 3x Watch Dog
Boot ROM
Resource Domain Controller
PMIC interface
Temp Monitor/Sensor
Core Complex 2
1x Arm Cortex-M4 core
16 KB L1 I-cache
16 KB L1 D-cache
256 KB TCM (SRAM)
Connectivity & I/O
1x PCIe 2.0 with l1 substates (1-lane)
2x USB2.0 Dual Role and PHY
1x Gb Ethernet (with IEEE 1588, EEE & AVB support)
4x UART 5Mbps
4x I²C
3x SPI
4x PWM
External Memory
x32/x16 LPDDR4/DDR4/DDR3L Up to 3000 MTps
3x SDIO3.0/eMMC5.1
NAND CTL (SLC/MLC) - BCH62
1x QuadSPI (XIP)
Optional Capability

i.MX 8M Mini Applications Processor Block Diagram

WEB LINKS

[1] Foundries.io: https://foundries.io/
[2] Arm SystemReady: https://www.arm.com/architecture/system-architectures/systemready-certification-program
[3] EdgeLock SE050: https://bit.ly/EdgeLockSE050
[4] EdgeLock 2GO: https://bit.ly/EdgeLock2GO
[5] i.MX 8M Mini: https://bit.ly/iMX8MMini
[6] Portenta X8 SOM: https://www.arduino.cc/pro/hardware/product/portenta-x8

New *Arduino* or *Electronics* Project?

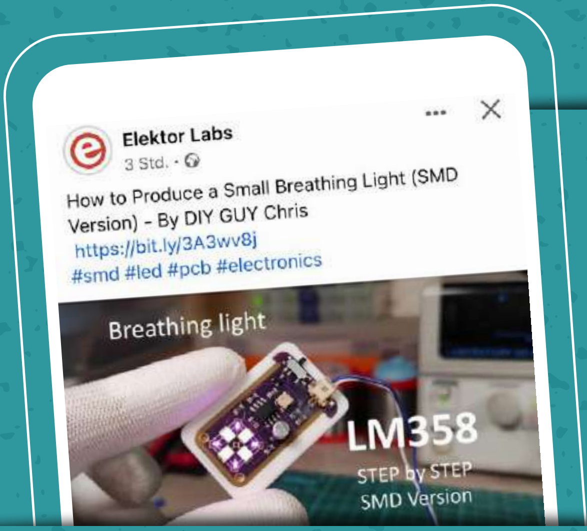
Share it with our community!

Elektor @Elektor · 4T
NEW 🤩🔊 Assembling the Elektor Fortissimo-100 Audio Power Amplifier
@Clemens_Elektor
bit.ly/3tORi8y
#amplifier #audio #power #electronics

Follow us on:

elektorlabs

www.elektor.com/TW



Elektor Labs
3 Std. ·
How to Produce a Small Breathing Light (SMD Version) - By DIY GUY Chris
https://bit.ly/3A3wv8j
#smd #led #pcb #electronics

www.elektor.com/Intsta

www.elektor.com/FB

Bringing the Human Element to Manufacturing

▲

Daria Baradel (front) and her Arduino colleagues.

Meet Daria Baradel, the Person Responsible for Production at Arduino

By Keith Jackson (Arduino)

How has one of the world's most popular electronics manufacturers successfully navigated the supply chain challenges of the past three years? With an unwavering passion for innovation, close collaboration with local partners, and a human touch.

Keith Jackson: Hi, Daria. With everything we see in the news about the global shortages of electronic components ceasing production in factories around the world, it must have been a challenging year leading manufacturing for Arduino?

Daria Baradel: Yes. Whilst everyone has heard about shortages in the electronics industry affecting everything from new cars to games consoles, it's not just about the lack of stock that is causing the biggest difficulties, it is the unpredictable delays and constant cost increases that make keeping control virtually impossible.

The whole team has had to adapt our approach and become more flexible. Keeping control requires predictability and risk management, but when predictability is no longer there, maintaining production is a daily challenge. On any given day, an expected delivery may be canceled, causing production stops at the factory. It's not always cancellations that cause last-minute changes in the production scheduling though. There are the exact opposites where there is suddenly the chance to take a random delivery of components that wasn't expected until 2023. In cases where this was the one outstanding line in a BOM, it suddenly gives us the opportunity to get a model into production again, so we are constantly adapting plans to what is available and what our customers are demanding. We had to review all standard procedures and adapt them in this unstable and unpredictable situation, making them less standard and more flexible in order to be ready to manage every single issue that would have arisen.

Keith: Can you share some insights as to how you and the team have adapted your approach to deal with these challenges?

Daria: At the beginning, I was trying to predict what was going to happen, but this proved to be impossible. Thus, I had to change my mindset and adopt more proactive procedures. As a team we agreed it was necessary to gain greater control on the deliveries; therefore, every Friday, we check with all of our suppliers what is due for delivery next week. For the main components, we then follow this up on a daily basis.

It isn't just a case though of how we work with our suppliers; it also became absolutely necessary to work much closer with the Arduino Hardware team responsible for designing and testing the products. Before finalizing a BOM for a new product or even confirming which components to mount on a board during prototyping, both teams now meet to assess availability and reliability of the vendor, as part of the design process.

Availability of a component, as well as its capability, is now a consideration in the final design. It's not all negative though, as the team now have a far greater understanding of the boards and the components as they have been involved much earlier in the design process. So we are all living and breathing every new product from the initial concept.

In 2022, the supply chain team together with hardware team have managed the validation of more than 100 alternative components and reworked the design of 16 products to adjust the BOM and being able to proceed in manufacturing.

Keith: What's it been like with the suppliers, given that Arduino is just one of many customers crying out for components?

Daria: Ironically, over the past few years, I had invested a lot of time automating as many purchasing procedures as possible for ordering components. This year, though, human interaction has come back to the fore. It's essential to communicate and work with other humans. Only through constant phone calls are we able to keep pace with the latest changes and the knock-on consequences. I'd probably be considered a stalker with the constant daily messages I've been sending them!

Many of Arduino's suppliers take a keen interest in the whole Arduino concept, with quite a few of them being Arduino enthusiasts. Therefore, it is clearly apparent that we got great support from some suppliers and poor from others; but on the whole, those we got great support from are now really tight Arduino suppliers, working closely with us as part of one big community.

Keith: How is manufacturing going and what's your outlook for the future?

Daria: Despite everything that's happened this year, manufacturing volumes are still over 20% up versus the same stage in 2021. This has been achieved by a complete cross team effort. The Hardware team have contributed immensely by checking and approving literally hundreds of alternative components, in many cases mounting and testing them for approval the same day to ensure we could source the available stock. So, although there is no light at the end of the tunnel in terms of components shortages with many lead times extending into the backend of 2023, I'm still positive that we can keep growing as, after all, companies are made by humans. The changes to mindset and procedures we've had to put in place this year will only make us stronger and more effective in the future.

Keith: You mentioned testing and approving alternative components on the same day. How is this doable?

Daria: Having our manufacturers close to the Arduino R&D offices has been a huge benefit through the COVID pandemic and into these times of shortages. Arduino has always prided itself on manufacturing in Italy, with all the boards manufactured across two plants in the Piedmont region of Italy. In practice, this has meant when any problems have occurred the engineers have jumped in a car and immediately gone to the factory nearby to Turin to check and change components if necessary. When we are informed of an alternative component's availability, it is necessary to confirm the order the same day; otherwise, the stock gets taken by someone else. You have to be very quick off the mark. So, the engineers will go to the factory, test the alternative product, and give us the greenlight, enabling the order to be placed that same day. This clearly would not have been possible if our manufacturing was carried out offshore.

Keith: Arduino must be one of the few electronics companies still using local manufacturers to produce its boards?

Daria: Yes. Manufacturing locally was a deliberate choice, as Arduino has always taken a positive approach to sustainability, environmental impact, and the local community. Our sustainability goals are more than a piece of paper: we have always tried to keep as low a carbon footprint as possible and limit waste. For example, we give away old stock to local schools, and the manufacturing and packaging of the products is carried out within a 50-km radius of our Turin office. The community spirit is applied when selecting our partners, as well as manufacturing the products locally. The kitting and packing is carried out by a local company that actively employs people with impairments and provides them with an opportunity to be independent.

Keith: It sounds like you're very proud to work at Arduino. But other than dealing with daily component shortages, what else inspires Daria?

Daria: I'm an engineer by training, with two master's degrees in production engineering and engineering management. Back in 2014 when I joined the company as a project manager, we were manufacturing between 3,000 to 5,000 boards per month. This has grown to over 50 times that level — meaning, I now have a full team of eight people to manage this.

Apart from work, though, I love to travel and actively participate in lots of sports, especially climbing, running, wind surfing, and beach volleyball. I find sporting activities are the best release for all the stresses of the day. But if I had a dream for the future, it would be to one day own a farm, as I love animals, coming from a family that kept cows, sheep, and goats. I'm sure even then I will have Arduinos all around the place to make it as efficient as possible.

220426-01

About the Author

Keith Jackson works in marketing for Arduino, and is passionate about all things Arduino as it's more than a company or a brand, it's a whole diverse community.

Questions or Comments?

Do you have any questions or comments relating to this article? Feel free to contact the author at k.jackson@arduino.cc or contact the team at Elektor at editor@elektor.com.

Elektor
Webinars

FEB
2023
16:00 CET

MicroPython Enters the World of Arduino

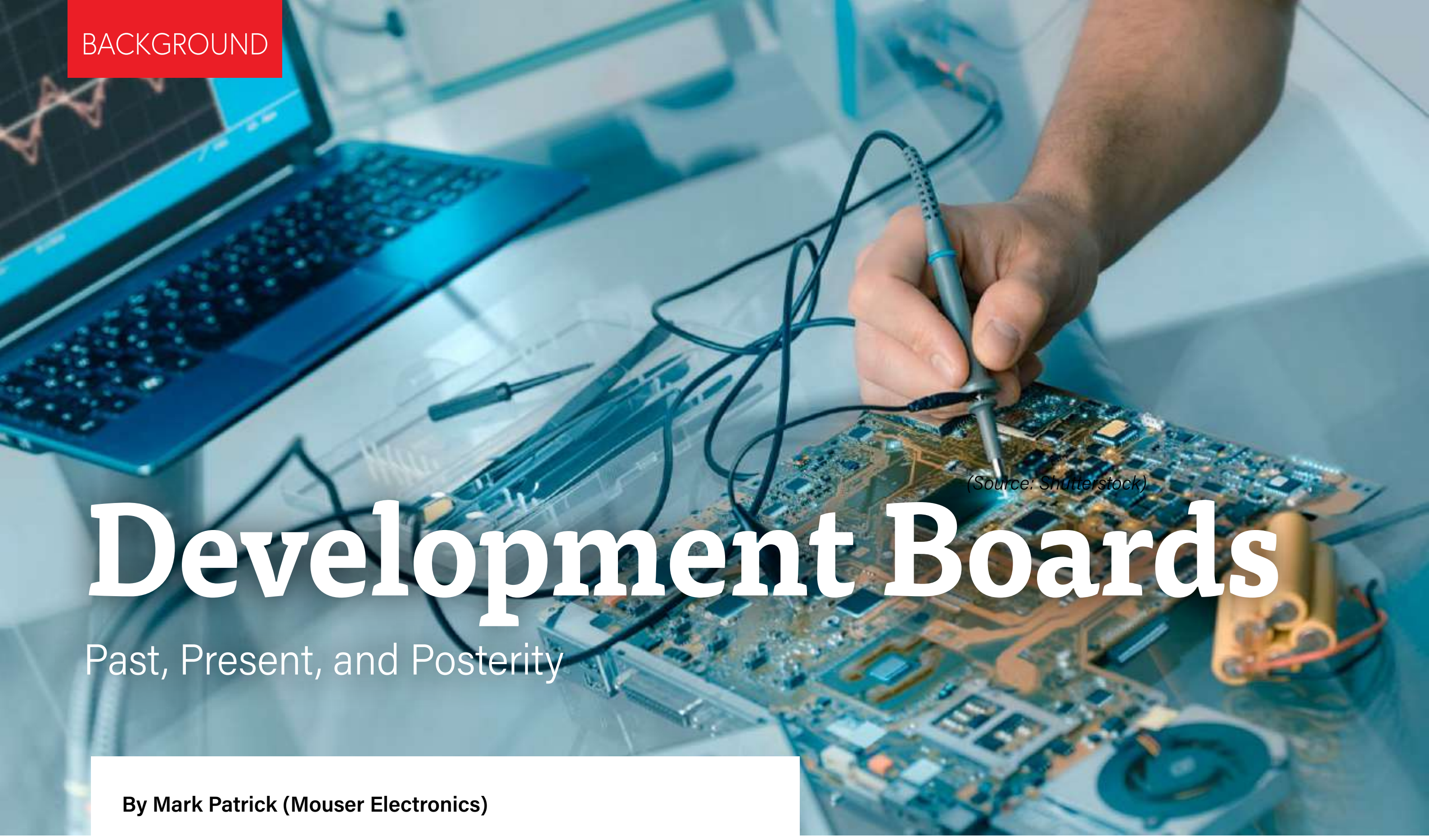
with Stuart Cording & Sebastian Romero

MicroPython has made it to the world of Arduino, providing the first significant alternative to programming in C and C++. So, what's all the fuss, how easy is it to use, and who can benefit from programming in this, for microcontrollers, relatively new language? Stuart Cording will speak with Sebastian Romero (Head of Content, Arduino) during our live webinar to find out more.

Join for free

www.elektor.com/webinar-MicroPython

Arduino × elektor


(Source: Shutterstock)

Development Boards

Past, Present, and Posterity

By Mark Patrick (Mouser Electronics)

In recent years, the meaning of the term 'development board' has become lost in the myriad of terms being used to describe hardware boards used for development purposes, including demonstration (demo) boards, evaluation kits and reference designs. In this article, we explain the meanings of these various terms and show how they differ from their close relations — the single board computers (SBC). We chart their evolution from past to present and investigate some trends for how they may evolve in the future.

What Is a Development Board?

At the outset, a clear definition of what is meant by a development board and how they differ from a single-board computer (SBC) is required. A development board is typically created by the manufacturer of a microcontroller to highlight its features (although the term is now also often applied to other types of components, as well). A microcontroller is an integrated circuit that contains a processor, some RAM, flash storage and has I/O functionality that allows it to interface with the real world. It effectively functions like a miniature computer housed in a single package, its purpose being to provide developers with a convenient way to interface with it and control external components like lights, small motors, etc. An SBC also provides this functionality with the main difference being that the CPU, RAM, and storage are each contained within separate ICs on the board, and interfaces allow it to be connected to a keyboard and/or display.

The microprocessor on an SBC requires an operating system, whereas a microcontroller is managed using an integrated development environment (IDE) provided by the manufacturer. In many cases, manufacturers now create development boards that include a microcontroller but whose main purpose is not to demonstrate the features of the microcontroller itself but those of sensors or other integrated circuits to which it interfaces. These are referred to as 'demo boards,' 'evaluation kits,' or — if they have been assembled to enable the collection of parts to perform a tangible purpose — 'reference designs.'

The purpose of some boards is not primarily for hardware development, but to provide access to the real-world data that software developers require to create and refine algorithms needed for artificial intelligence and machine learning applications. While these may not conform to the original definition and purpose of a 'development board,' these are now collectively understood to refer to any piece of hardware that can be used in the software and hardware development of new electronic products.

Past

The first microcontroller development board to capture the attention of the engineering community was released in 2006. This prototyping platform, which later became known as Arduino [1] (**Figure 1**) was quickly adopted by a new category of electronic designers that included enthusiasts, hobbyists, and DIY engineers. Arduino laid the foundations for the commercial success of later SBCs and microcontroller-based platforms and was soon followed in 2008 by BeagleBoard [2], which provided engineers with a low-cost, open-source community-supported development platform. 2012 saw the release of the Raspberry Pi [3] — the first modern single-board computer. Like BeagleBoard, it was conceived as an educational platform intended to provide a low-cost way in which students could use to learn how to write program code. The appeal of Raspberry Pi was much wider than students alone, and it was quickly adopted by amateur hobbyists and professional engineers alike.

Present

Today, there are two main categories of SBCs — proprietary and open-source. Proprietary SBCs are typically designed for use in end applications and have been subjected to the same type of testing and quality assurance as other end products. They are either integrated into electronic equipment or installed in a rack-mount configuration. Open-source SBCs provide users with access to their hardware design and layout and to any source code that they use. This allows users to quickly and easily learn how the software and hardware operates and then adopt the design to match their requirements.

Today, development boards and SBCs come with a wide variety of processor types. These range from x86-based processors within the traditional PC space (AMD and Intel) to ARM processors used in industrial and mobile applications. Linux and its derivatives (Ubuntu, Fedora, Debian, etc.), Android, and Windows CE are the operating systems most used on SBCs. Microcontroller development boards do not require an operating system and are programmed via an IDE provided by the manufacturer. Microcontroller development boards and SBCs have both evolved to include wireless connectivity (Wi-Fi, Bluetooth), and the most recent audio and video interfaces, meaning that some SBCs now have features equivalent to those found in many PCs and tablets.

Posterity: Development Boards Become the Final Product

Traditionally, manufacturers created development tools with the intention of using them as a marketing aid that could improve the likelihood of selling their microcontrollers to prospective customers (often referred to as 'Design-in' within the industry). They hoped that by minimizing the amount of work required by design engineers to get a part up and running in the lab and by making it easy for them to access and investigate its features, it would make them more likely to choose their microcontroller and ancillary parts for use in initial product prototyping and would eventually lead to higher volume orders if the part was chosen for use in a mass production. For products where the difference in the technical specifications between parts from different suppliers is negligible, this is a prudent approach. However, for manufacturers, this approach has been, in some respects, a victim of its own success. They realized that they must continue to reduce the amount of work required for an engineer to engage with their products, so the development board has become the key differentiator — especially for products that are broadly similar to those of their competitors.

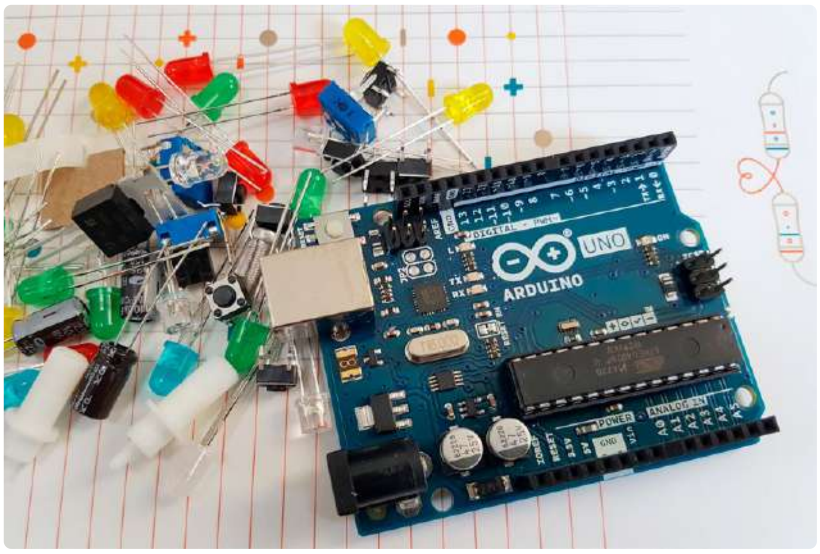
The expectations of design engineers has risen such that, even for parts that have a clearly identifiable competitive advantage (for example, in power or speed) they still expect the associated development boards to have plug-and-play levels of accessibility.

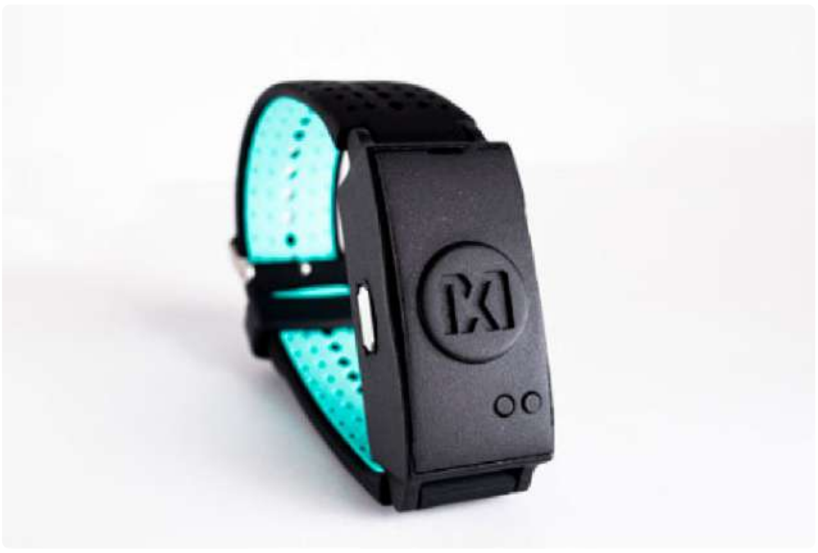
Manufacturers have further improved their value proposition by offering reference designs consisting of a microcontroller and other integrated circuits (typically sensors). Initially, these were intended to provide a guide for how devices could be interconnected to emulate the electrical functionality of a final product, with little focus on their form factor, size of the design, or ease of manufacture. However, some manufacturers have taken reference designs to the next level by creating fully-fledged product prototypes and even completely viable products.

The health sensor platform (HSP) reference designs [4] by Maxim Integrated (now part of Analog Devices) can be used as an example in charting this evolution. The initial version of these reference designs was a small development board that features an assortment of sensors

Figure 1: Arduino microcontroller development board. (Source: Shutterstock)

▼





▲

Figure 2: HSP3.0 by Maxim Integrated.

(temperature, pressure, accelerometer, biopotential, etc.) suitable for use in health and fitness applications and which could be configured using a microcontroller. Its successors, HSP2.0 and HSP3.0, had form factors that allowed them to be wrist-worn, and they look much like other wearables available on the market (**Figure 2**).

This allowed developers to evaluate the functionality of their sensors in real-world scenarios. Importantly, these designs also provided software developers with free access to sensor readings (information not easily accessible from other health and fitness wearables). The purpose of this approach was to enable the development of machine learning and artificial intelligence algorithms that would add value to the application.

Figure 3: Programmable Logic Controller. (Source: Shutterstock)

▼

By demonstrating how their hardware facilitates easy access to data, Maxim hoped that product developers would choose some (or all) of the ICs in Maxim's sensor solution for use in products. Maxim extended this approach as far as to develop the MAX HEALTH BAND [5] (wrist) and MAX ECG MONITOR [6] (chest strap), both fully designed and constructed to be fully viable wearable health and fitness devices. While they were not intended for sale directly to consumers, businesses could enter into an agreement with Maxim to have these products branded under their own label in return for a royalty payment.

Offering a fully functioning product in this way, where all the development work had already been done, has the potential to appeal to a new and broader non-technical business customer base. Nordic Semiconductor's Thingy:91 [7] is another example of a development platform where hardware has become almost incidental to the task of providing developers with access to the data they need to develop the software and algorithms that allow the intrinsic value of the hardware to be realized (but, in so doing, conveniently make them the obvious choice for use in new product designs that leverage these algorithms). It is likely that this approach will be adopted by even more manufacturers in the future.

Increased Use of Development Boards in Industrial Products

Adapting development boards and SBCs for use in commercial products has become increasingly common, but another emerging trend is the use of boards in lower-volume but higher-value applications, such as in industrial end-products such as programmable logic controllers (**Figure 3**), which are subject to more exacting standards than their commercial equivalents.

Testing Boards for Industrial Applications

Many of today's SBCs have inherently become fully verified designs because the parts they contain were originally developed for use in end-products and have therefore been tested and quality-assured. This is also because open-source designs are constantly reviewed by an army of competent designers and programmers who update and appraise the boards and the software that they use.

Testing of SBC boards is now performed through high-quality design and manufacturing firms, and they are subjected to the same rigorous degree of quality control as any other end products, thus allowing them even to achieve CE or FCC certifications. This test flow can easily be extended to meet the requirements of industrial products.

On the other hand, microcontroller development boards supplied by manufacturers or third parties, while usually suitable for use in commerical products, are not typically subjected to the same stringent levels of testing required for industrial products. This means that manufacturers do not currently recommend them for immediate use (in their current guise) in these applications.

While some boards include industrial grade components, they are more often only of commercial grade, with the boards being designed to operate at room temperature. Prototypes of development boards typically undergo testing at room temperature for several days or weeks, but this varies depending on the manufacturer, as there are no set standards. The primary quality requirement for manufacturers is that their boards operate reliably at room temperature, and therefore purchasers should be aware that it is unlikely that these boards have been tested at extremes of temperature or humidity. They also aren't normally tested to survive the stresses associated with intense vibration or shock.

As a result, the main objective when determining which development board to use in an industrial application is to reduce risk: The board components must be of the proper temperature grade. It also makes sense to stress test several boards simultaneously at high temperature for a period of days. Similarly, if planning to use a development board in a product that will experience high humidity, the boards must be evaluated under comparable conditions. Where a board is intended for use in a high vibration application, then it should be mounted in a test frame and vibration tested.

Conclusion

SBCs and microcontroller development boards provide small companies with a convenient way to bring their designs to market quickly without the expense of new hardware development. The boards allow them to focus on software innovation and increasingly on the development of machine learning and artificial intelligence algorithms. SBCs and development boards have broadened their remit well beyond that which was originally envisioned for them, and they have made a real impact on the recent history of the electronics industry. They continue to become more powerful, intelligent, and responsive, while remaining easily accessible to both professional engineers and electronics enthusiasts.

220597-01

About the Author

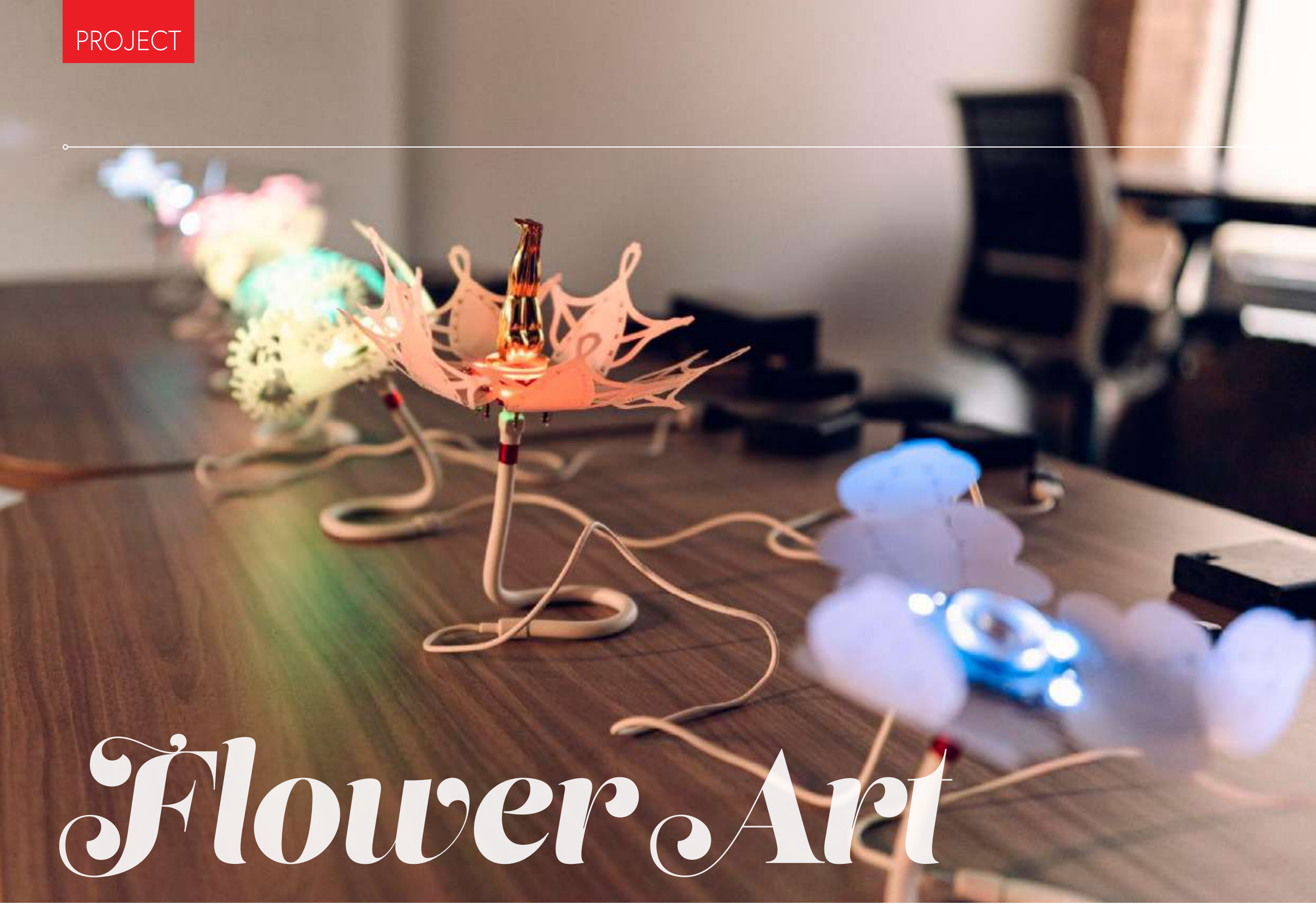
As Mouser Electronics' Technical Marketing Manager for EMEA, Mark Patrick is responsible for the creation and circulation of technical content within the region — content that is key to Mouser's strategy to support, inform and inspire its engineering audience.

Prior to leading the Technical Marketing team, Patrick was part of the EMEA Supplier Marketing team and played a vital role in establishing and developing relationships with key manufacturing partners. In addition to a variety of technical and marketing positions, Patrick's previous roles include eight years at Texas Instruments in Applications Support and Technical Sales.

A "hands-on" engineer at heart, with a passion for vintage synthesizers and motorcycles, he thinks nothing of carrying out repairs on either. Patrick holds a first-class Honours Degree in Electronics Engineering from Coventry University.

WEB LINKS

[1] Arduino Boards Distributor: https://elektor.link/MouserArduino
[2] BeagleBoard Distributor: https://elektor.link/MouserBeagleBoard
[3] Raspberry Pi Distributor: https://elektor.link/MouserRaspberryPi
[4] Maxim Integrated Distributor: https://elektor.link/MouserMaxim
[5] MAX HEALTH BAND: https://elektor.link/MouserMaxHealthBand
[6] MAX-ECG-MONITOR : https://elektor.link/MouserMaxECGMonitor
[7] Nordic Semiconductor Thingy:91™ Multisensor Prototyping Kit : https://elektor.link/MouserThingy91



Flower Art from Muscle Wires

Figure 1: The results of a 'Creating Kinetic Flora' workshop.

Kinetic Sculptures That Communicate With Sound

By Dave Vondle (IDEO)

Since their strange properties were discovered over 60 years ago, Nitinol wires have been a solution looking for an application. Here, the inspired application is art — flowers that respond to sound with light, sound, and movement.

This year, a few of us at the design company IDEO were invited to do a workshop at Eyeo Festival [1]. We wanted to build something physical with people, but something that could be controlled with code and experimented with. My colleague and collaborator, Jenna Fizel, has done a lot of inspiring stuff writing software that supports making 3D objects from paper and other thin materials. Recently, I've been experimenting with Nitinol wire, a material that can change shape when you pass current through it. With the additional support of IDEO software designers Derek Olson and YC Sun, we set out to see how we could make a kit of parts that allows others to make moving paper sculptures (**Figure 1**).

In the workshop, we wanted people to leave with something that was uniquely their own, but we also wanted the workshop's final product to be collaborative. This was the first in-person workshop we did coming out of the pandemic, and we wanted there to be some sort of unique value for all of us to build something together, in person. Could the sculptures we make perform with one another? How would they communicate? We figured if they could communicate sonically, we could hear them perform together. The workshop could create a generative symphony!

The participants in the workshop had a wide variety of backgrounds and electronics knowledge. By building these flowers, we were able to familiarize folks with concepts across a large spectrum. Some of the fundamentals were 'Getting Started with Arduino' and Ohm's law for understanding the voltage required to drive the Nitinol. Some of the more complex concepts we could cover were what pulse-width modulation is, fast Fourier transforms (FFT) for identifying the frequencies of neighboring flowers, and the basics of audio synthesis.

Each participant built a hand-painted flower, where they designed the shape of the petals (**Figure 2**). The electronics in the flowers control the LED lights on the top, the sounds the flower makes, as well as independent control of the petal position.

You can see a video of what the final performance at a subsequent IDEO workshop looked like at [2].

Figure 2: Threading the Nitinol wire through the flowers.

As we get into the details below, I'll walk through the design of the hardware, then I'll talk about the software. Lastly, I'll get into how you can make your own.

The Hardware

The hardware is loosely based on a combination of the Adafruit Trinket M0 [3] and the Sparkfun Electret Microphone Breakout Board [4]. In addition to the functionality of these boards, we added another amplifier for a speaker, a 5 V buck-boost circuit so it could run from a battery source, some WS2812 addressable LEDs, and 8 MOSFETs to control the current in the Nitinol wire.

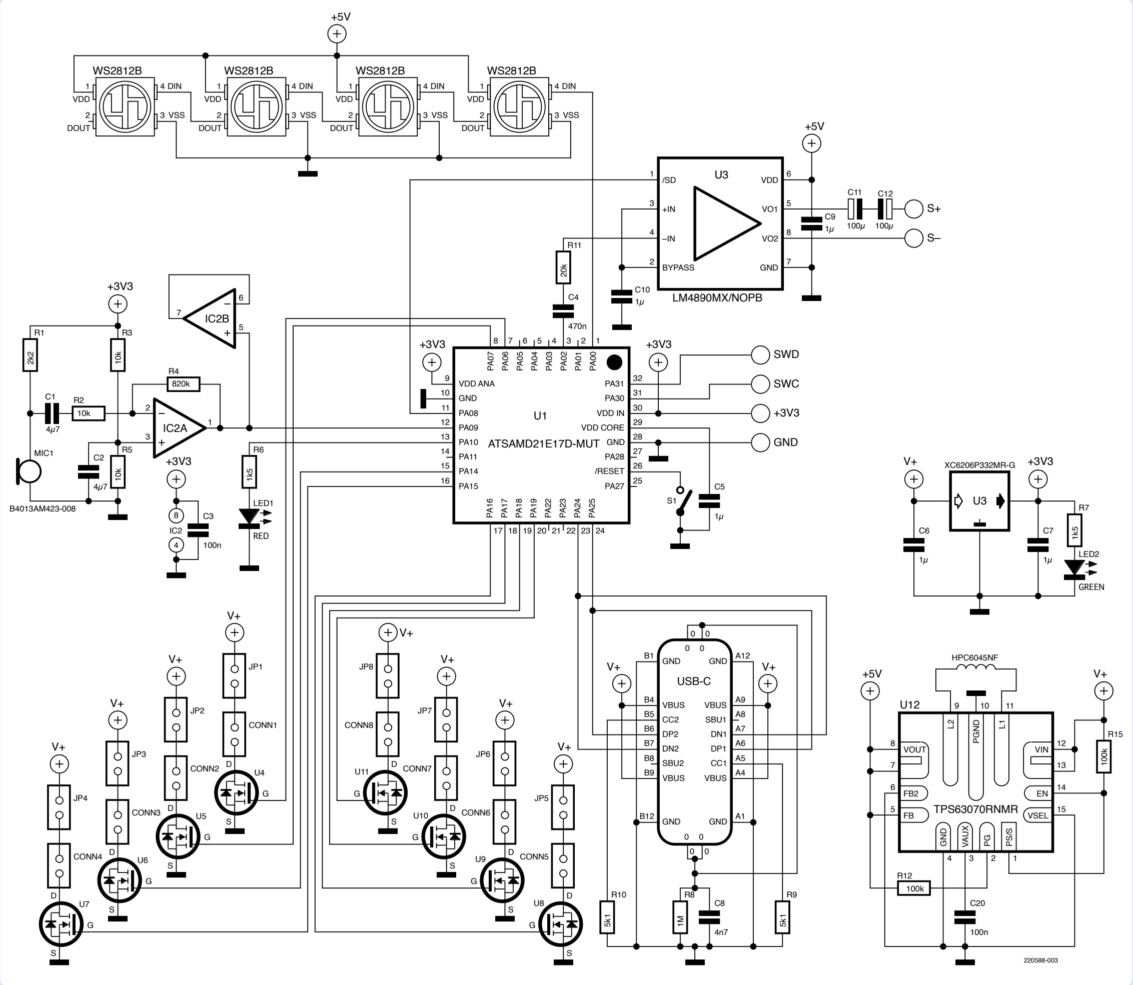
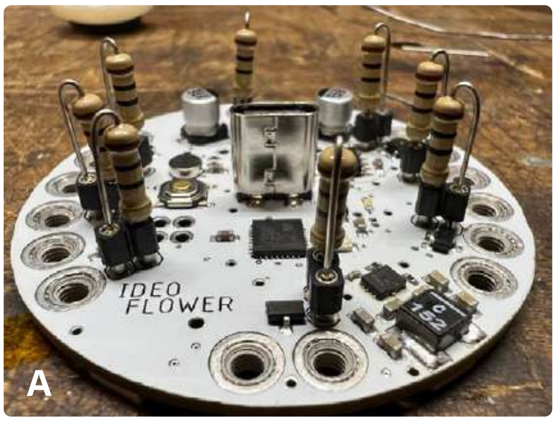


Figure 3: Schematic for the Eyeo Flower.



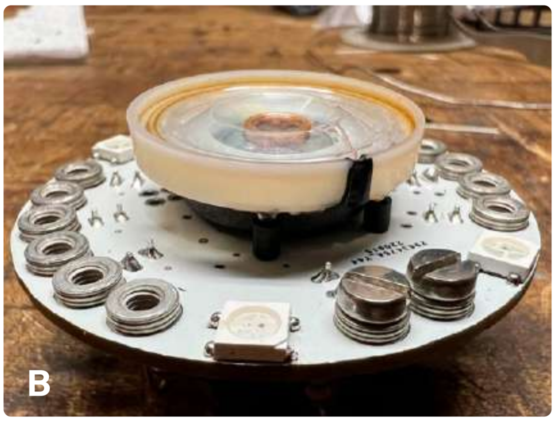


Figure 4a: Finished board, stem side.

Figure 4b: Finished board, petal side.

The full schematic is shown in **Figure 3**. The finished board looks as shown in **Figure 4a** and **4b**.

For our workshop, we wanted to avoid soldering, as Nitinol can be extremely challenging to solder to without specialized materials and techniques, so we use a system of threaded inserts and screws to mechanically connect the wire. This also allows for fine control of the petals' wire pre-tension.

The socketed resistors are selected based on the length of the Nitinol wire to limit the maximum current. We are using 100 µm Nitinol wire [5] with 126 Ω/m resistance and rated current maximum of 200 mA. From this, a resistor can be selected to prevent damage being done to the wire. In prior tests, we left the resistors out, opting to limit current by setting a maximum PWM ratio in software. But we found that, in the programming process, sometimes a pin would go high, so we resolved this with the physical resistors.

A speaker is prepared with test-point pins soldered onto the back, allowing it to be plugged into the top of the board. On the bottom of the board, there is a central vertical USB connector, so the USB cable can be used as the "stem" of the flower.

The Microcontroller

We chose the ATSAMD21 microcontroller family due to its ability to be Arduino-compatible in a single chip (most other Arduino boards utilize another component for the serial UART-to-USB capability). However, the ATSAMD21E18 variant that is used in the Adafruit Trinket M0 was not available due to the chip shortage. We were able to find ATSAMD21E17D chips, which were available. The primary difference between these chips is a reduction in the available flash memory from 256 KB to 128 KB. While this difference is small, we still had to modify the Arduino Bootloader [6] (forked from *uf2-samdx1*), and make an associated new board in the Arduino IDE to speak with this new chip/bootloader.

Example Code

The example Arduino sketch [7] provides a baseline for the flowers to "sing" together by outputting audible sounds from the speaker, listening to the notes using the microphone, and reacting with movement in the petals and light from the LEDs. I'll do a little breakdown of how this all works.

Playing Sounds

The ATSAMD21 has a pin that can be configured as a DAC (digital-to-analog converter). This feature is great for being able to get a wide variety of musical timbres coming out of the chip. It allows the flower to be much more expressive than if we were using the Arduino Tone library [8], whose `tone()` function allows you to vary the pitch of a square wave, but the DAC we use allows us to control the shape of the waveform itself. We are using the fantastic Mozzi [9] library, which provides a framework for defining and playing sounds from this DAC.

In the code, we set up a couple of cosine oscillators — one for the main carrier frequency and one for vibrato. We also set up an envelope (otherwise known as an ADSR — Attack, Decay, Sustain, Release), which gives us control of the amplitude of the sound over time.

As we play the audio waveform, the microcontroller goes through a lookup and calculation for what the next voltage level is that needs to be sent out of the DAC. This happens at a frequency of 16.384 kHz, so, if we have blocking code or lots of processing that happens in that time period, we can throw off the creation of a smooth wave. Because of this, the code first listens, reacts with light and movement, then plays the sound on its own when other processes aren't happening.

Listening to Sound

As discussed in the intro, the flower listens for a specific tone, then reacts with its own tone. The microphone signal is fed to a pin on the microcontroller (one that can be configured as an "analog in") via an amplifier.

In order for the flower to understand the tone or frequency of the sounds it hears, we use the aforementioned fast Fourier transform principle. This is a mathematical transform that takes a set of samples in the time domain (time vs. amplitude) and changes it to a set of samples in the frequency domain (frequency vs. amplitude). This mathematical process can be quite resource-intensive and trades frequency granularity for speed/memory. To get this running as fast as possible, we use Adafruit's Zero DMA (Direct Memory Addressing) library [10] to pull samples from the microphone into an array, and the Adafruit Zero FFT library [11] to perform the FFT. Once the data has been processed through the FFT, we can look at the primary frequency of the sample to see if it matches the note we are looking for.

Moving the Petals

You can see an example of the paper prototype in **Figure 5**, or see an animated GIF of it in action at [12]. The wire is threaded through the petal asymmetrically. You can see in **Figure 6** that there is much more wire on the top surface of the petal versus the bottom surface. When the wire contracts, it shrinks the top surface of the petal, resulting in a shear strain on the surface, which curls the petal.

Before we go into the code of the petal movement, let's take a moment to dive into how Nitinol works.

Nitinol uses electric current to heat the wire up past a threshold temperature (for our wire, it's 70 °C) where it undergoes a phase change and contracts. In order for the wire to go back to its original length, it has to be physically stretched again.

The phase change follows this cycle (see **Figure 7**): When the wire is in its elongated state, it is in the deformed martensitic state. The crystal structure of martensite is called "body-centered cubic." When heated, the Nitinol changes to an austenitic crystal structure. This is called "face-centered cubic," which is "close-packed," meaning that the structure allows for more atoms to be packed closer together. So, as this crystal structure changes when the wire heats up, the atoms rearrange into a close-packed arrangement, and this causes the wire to contract.

As we mentioned, the wire has to be physically stretched back out. We needed a way to have this happen without clunky or complicated springs. Instead of using paper for the petals, we chose a matte finish polypropylene "water-color paper" called YUPO [13]. By using this material, we can use the elasticity inherent in the material to act as a spring to return the flower petals to an open state after bending.

Each petal can be controlled separately and is connected through a MOSFET to a pulse-width modulation (PWM)-capable line. In the code, we close the petals, and then, because they're "off" when we open them, this frees our processor up to play sound (even while the petals are still opening).

To figure out the Nitinol protection resistors, we need to do a little bit of calculation. With our petal shape, we end up with about 13 cm of wire going from post to post. Our wire has 126 Ω/m resistance and a rated current maximum of 200 mA. So, our wire resistance is 126 Ω/m × 0.13 m = 16.38 Ω. To calculate the resistance that we want at 5 V maximum voltage, we use Ohm's law to get V / I = R, so 5 V / 0.2 A = 25 Ω. We selected a 10 Ω resistor in series with our 16.38 Ω wire (10 Ω + 16.38 Ω is more than 25 Ω) to make sure that we never deliver too much current to the wire.

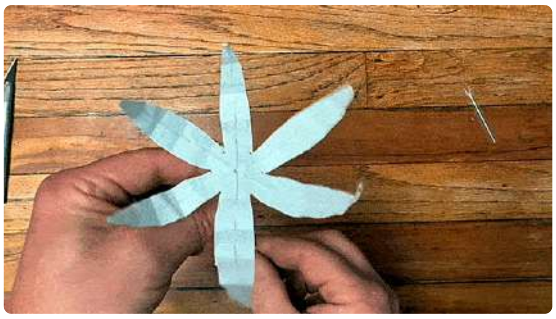


Figure 5: The first paper prototype using a straw, string, and paper to test the idea before designing hardware.

◄

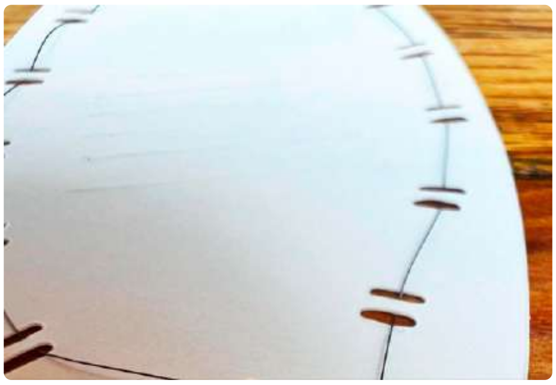


Figure 6: Close-up of a threaded petal.

◄

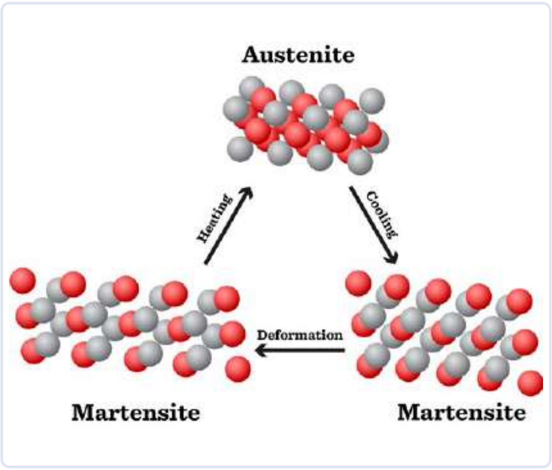


Figure 7: A model of Nitinol's shape-memory crystal structure cycle.

◄

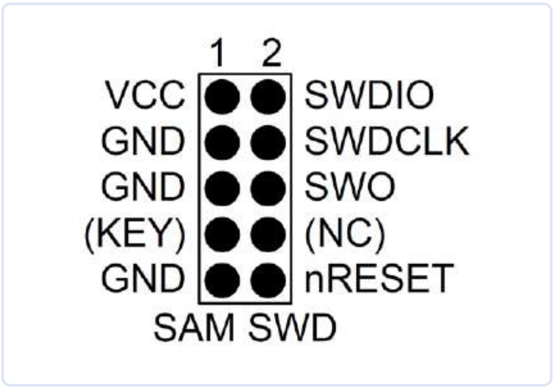


Figure 8: Recommended ARM SWD/JTAG header pinout. Source: Atmel datasheet [19]

LED Light

The *uf2* bootloader we modified had support for NeoPixels, which are Adafruit-branded WS2812 LEDs. This allows the LEDs to be diagnostic and give us the status of the bootloader/USB connection. To control the LEDs when the code is running, we use the Adafruit NeoPixel library [14]. WS2812 LEDs are designed to be daisy-chained. When sending data to the LEDs, each LED receives 8 bits per color (24 bits total). As we send more data, this data gets shifted to the next LED in line, allowing independent control of numerous LEDs from a single pin.

How to Make Your Own Flowers

We made an interactive website [15] that allows you to quickly sketch a flower petal shape to be cut with a desktop cutting machine such as a Cricut.

You can find the source Eagle files, Gerber, CPL, and BOM files to make the PCBs on the project's GitHub page [16]. With these files, you can use a board fab house to get assembled boards. We used JLCPCB [17], so these files are already formatted for JLCPCB's PCB and assembly services. You may attempt to assemble them yourself, but there are a number of fine-pitch ICs as well as 0402-sized passives that may pose a challenge.

In the GitHub repo, there are also links for where you can find the YUPO paper, the gooseneck USB cables, the Nitinol wire, the screws, and the threaded insert broaching nuts.

To get the Arduino bootloader onto the chip, we used a SEGGER J-Link programmer and followed the excellent instructions on how to program SAMD bootloaders from Adafruit [18].

To hook the board to the programmer, we look at the SAM SWD pinout from Atmel's ICE User Guide [19] (see **Figure 8**).

What matters to us is:

Pin 1: VCC / Vref
Pin 2: SWDIO (SWD)
Pin 3: GND
Pin 4: SWDCLK (SWC)

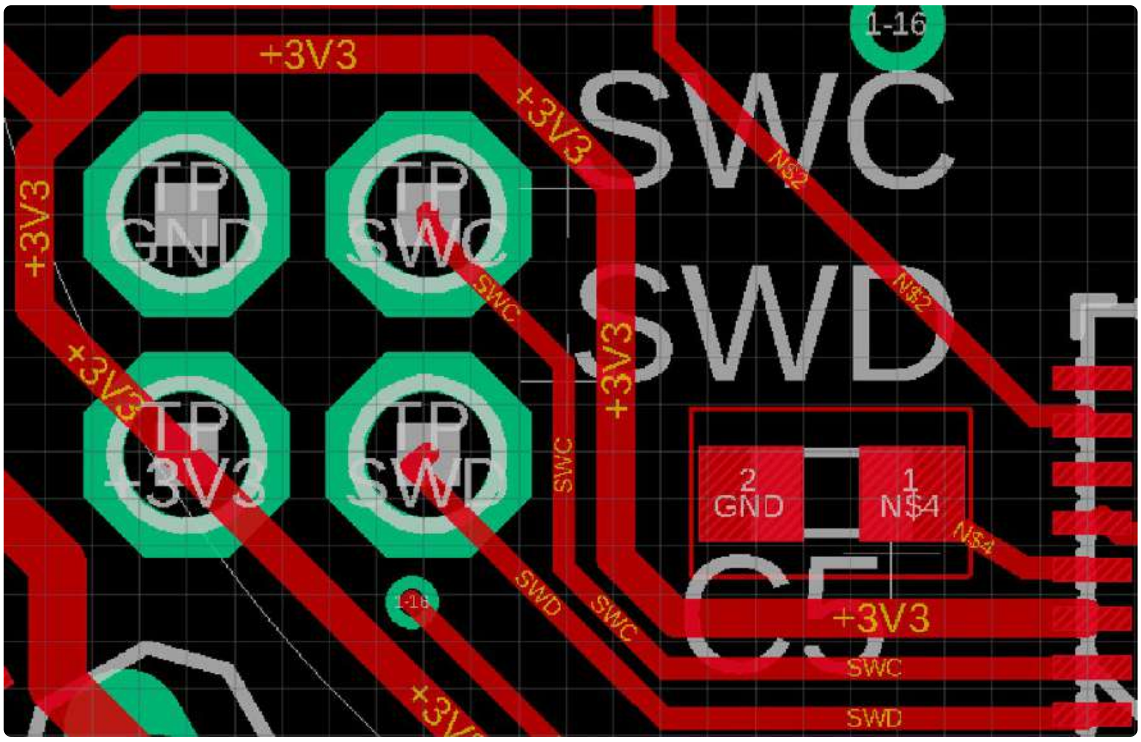


Figure 9: Location of the SWD pins on our board.

These need to go to the corresponding pins on the PCB — indicated in **Figure 9** as GND, SWC, +3V3, and SWD.

Because we were programming many boards, we built a programmer using pogo pins, to easily swap boards out (**Figure 10**).

We also need to press-fit the "broaching nuts" (threaded inserts) into our board so that we can screw down the Nitinol wire to the board. We used an old X-ACTO handle to make a custom press-fit jig to press the broaching nuts into the board (**Figure 11**).

Depending on the tools you have, there are likely better ways to make programming jigs and broaching nut jigs. Let us know if you find a better way!

Conclusion

We loved making these and sharing them with the world. We think it's a fun platform for experimenting with a number of different principles in a small package. If you end up building on this, or want to discuss this with us, please let us know!

220588-01

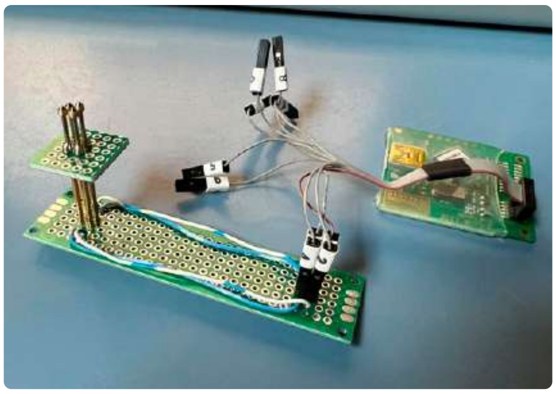


Figure 10: Custom programming jig.

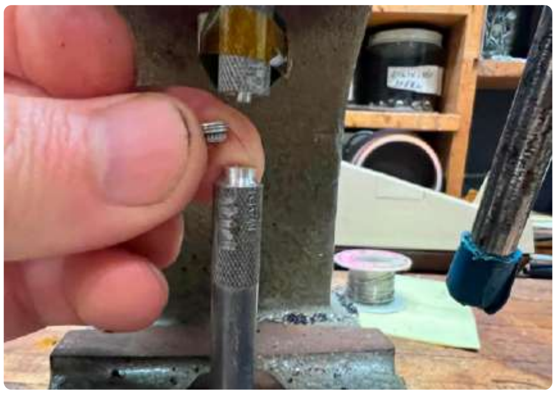


Figure 11: Custom tools for a small arbor press.

Questions or Comments?

If you end up building on this, or want to discuss this with us, send a message to @ideo on Instagram, or email me at dvondle@ideo.com, or contact Elektor editorial at editor@elektor.com.

About the Author

Dave Vondle is a Director of Experimentation and Publishing at IDEO. He works to create well-crafted products and experiences by designing and facilitating their development. Preferring not to stay put in one role, Dave moves between guiding projects, coding, designing interfaces, building prototypes, and designing circuits. Prior to IDEO, he received a BS from Brown University in Electrical Engineering, while taking classes at the Rhode Island School of Design to fill out his creative interests. On the side, he is working on building a set of Eurorack-format CRT analog X-Y oscilloscopes. You can find him on Instagram at @ideo and @vondle_ synths.

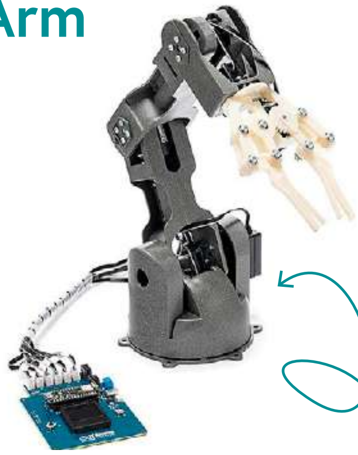
WEB LINKS

[1] Eyeo Festival: https://eyeofestival.com
[2] Nitonol Flowers Demonstration: https://youtu.be/MBdbXO-WHJ4
[3] Adafruit Trinket M0: https://adafruit.com/product/3500
[4] Sparkfun Electret Microphone Breakout: https://sparkfun.com/products/12758
[5] Muscle Wires Actuator Wire 100 µm: https://elektor.link/musclewires
[6] Modified Arduino bootloader: https://github.com/ideo/eyeo-flower/tree/main/Bootloader
[7] Eyeo Flower Example Sketch: https://github.com/ideo/ArduinoCore-samd/tree/master/libraries/Eyeo_Flower
[8] Arduino Tone library: https://arduino.cc/reference/en/libraries/tone
[9] Mozzi: https://sensorium.github.io/Mozzi
[10] Adafruit Zero DMA library: https://arduino.cc/reference/en/libraries/adafruit-zero-dma-library
[11] Adafruit Zero FFT library: https://www.arduino.cc/reference/en/libraries/adafruit-zero-fft-library
[12] Nitinol Flower Paper Prototype (animated GIF): https://elektor.link/gif/nitinol-flower-paper-prototype.gif
[13] YUPO synthetic paper: https://yupousa.com/what-is-yupo
[14] Adafruit NeoPixel library: https://arduino.cc/reference/en/libraries/adafruit-neopixel
[15] EYEO Flora interactive site: https://observablehq.com/@jftesser/eyeo-flower
[16] EYEO Flower GitHub repository: https://github.com/ideo/eyeo-flower
[17] JLCPCB circuit board manufacturer: https://jlcpcb.com
[18] How to Program SAMD Bootloaders: https://learn.adafruit.com/how-to-program-samd-bootloaders
[19] Atmel-ICE Debugger User Guide: https://elektor.link/AtmelICEUserGuide

Get your hands on new Arduino Hardware!

There is nothing that excites us more than getting our hands on new hardware, and so this collaboration with Arduino has been a treat! Want to experience the real deal yourself? Elektor has stocked up the stores to accommodate all products that are featured in this edition!

Arduino Braccio++ RP2040 powered Robot Arm



The next evolution of the Tinkerkit Braccio robot is called Arduino Braccio ++ , a brand new robotic arm designed for advanced users. Arduino Braccio ++ can be assembled in several ways for multiple tasks, such as moving objects, mounting a camera and tracking your movements, or attaching a solar panel and tracking the movement of the sun. Arduino Braccio ++ offers a multitude of expansive possibilities from the very outset, including a new Braccio Carrier with LCD screen, new RS-485 servo motors, and a totally enhanced experience.

www.elektor.com/20174

PID-based Practical Digital Control with Raspberry Pi and Arduino Uno

www.elektor.com/20274

Arduino Pro Nicla Vision

Nicla Vision combines a powerful STM32H747AII6 Dual ARM Cortex-M7/-M4 IC processor with a 2 MP color camera that supports TinyML, as well as a smart 6-axis motion sensor, integrated microphone and distance sensor.

www.elektor.com/20152

Arduino Pro Portenta H7

Portenta H7 allows you to build your next smart project. Ever wanted an automated house? Or a smart garden? Well, now it's easy with the Arduino IoT Cloud compatible boards. It means: you can connect devices, visualize data, control and share your projects from anywhere in the world.

www.elektor.com/19351

Arduino Pro Portenta X8

Portenta X8 is a powerful, industrial-grade SOM with Linux OS preloaded onboard, capable of running device-independent software thanks to its modular container architecture. It's basically two industrial products in one combining Arduino's availability of libraries/skills with container-based Linux distribution.

www.elektor.com/20270

Arduino Pro Nicla Sense ME

A new standard for intelligent sensing solutions.

www.elektor.com/20327

Portenta Vision Shield (Ethernet)

The Portenta Vision Shield brings industry-rated features to your Portenta. Professional computer vision, directional audio detection, Ethernet, and JTAG for Arduino Portenta.
www.elektor.com/19511

Portenta Vision Shield LoRa®

This Portenta hardware add-on will let you run embedded computer vision applications, connect wirelessly via LoRa® to the Arduino Cloud or your own infrastructure.
www.elektor.com/20332

Portenta Breakout

Portenta Breakout board is designed to help hardware engineers and makers to prototype and help test devices connections and capacity within the Portenta family boards.
www.elektor.com/20341

Arduino Pro Portenta Max Carrier

Easily prototype your Portenta applications. Deploy in zero time. Max Carrier transforms Portenta modules into single-board computers or reference designs that enable edge AI for high-performance industrial, building automation and robotics applications.

www.elektor.com/20271

Arduino Uno Rev3

The classic high-performance, low-power AVR® microcontroller. The Uno is the best board to get started with electronics and coding. The Uno is simply the most robust board to enable you to start tinkering with the Arduino platform.

www.elektor.com/15877

The Classic

Arduino Make-Your-Uno Kit

A new kit including a **DIY through-hole UNO** with all components to build up your UNO and make your own UNO-powered synth!

www.elektor.com/20330

Arduino Ethernet Shield 2

www.elektor.com/19941

Arduino Sensor Kit Base

www.elektor.com/19944

Arduino Nano

The Arduino Nano is a small, complete, and breadboard-friendly board based on the ATmega328 packed in the smallest available form factor of 18 × 45 mm!
www.elektor.com/17002

Arduino Nano RP2040 Connect

The Arduino Nano RP2040 Connect is an RP2040-based Arduino board equipped with Wi-Fi, Bluetooth, a microphone and a six-axis smart motion sensor with AI capabilities.
www.elektor.com/19754

Arduino Nano 33 BLE Sense

Bring the power of AI to your pocket with the more powerful nRF52840 processor and a series of embedded sensors and the possibility of running Edge Computing applications (AI).
www.elektor.com/19936

Supporting **Arduino Resellers**

This guest-edited Arduino Edition of Elektor Magazine was made possible with the support of these members of the Arduino reseller community.

Check them out for your Arduino-related needs.

GOTRON AALST GENT HASSELT

www.gotron.be

HELLAS digital

www.hellasdigital.gr

TINYTRONICS

www.tinytronics.nl

Paradisetronic.com

www.paradisetronic.com

Techni Science.

www.techniscience.com

WHADDA MADE BY VELLEMAN

www.whadda.com

KUBii

www.kubii.fr

GO TRONIC ROBOTIQUE ET COMPOSANTS ÉLECTRONIQUES

www.gotronic.fr

Check out any one of these resellers for your Arduino-related needs.