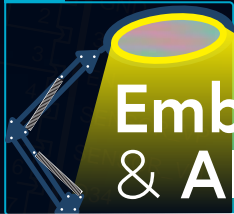


## AI Specialist

### Machine Learning with the Jetson Nano

lamp



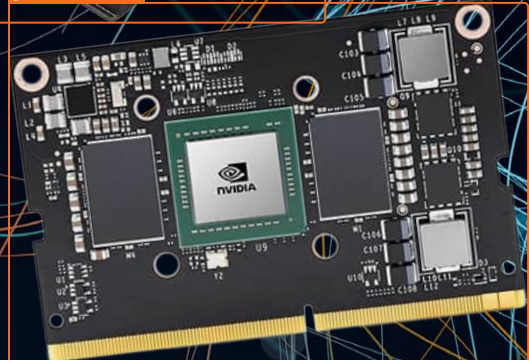
FOCUS ON

## Embedded & AI

jetson nano



jetson tx2

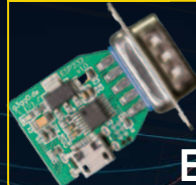


buttons

### Image Recognition Made Easy With Edge Impulse Framework



ESP32-RS-232



### ESP32-to-RS-232 Adapter A Wi-Fi Link for Classic COM Ports

laptop



### CaptureCount An Object Detector and Counter on the Raspberry Pi 5

lamp



box

crate

crate

crate

crate

crate

crate

crate

crate

crate

crate

crate

crate

crate

crate

crate

crate

crate

crate

crate

crate

crate

crate

workbench

workbench

workbench

workbench

workbench

workbench

workbench

workbench

workbench

workbench

workbench

workbench

workbench

workbench

workbench

workbench

workbench

workbench

workbench

workbench

workbench

workbench

workbench

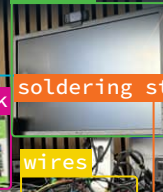
workbench

workbench

workbench

workbench

screen



book

book

book

book

book

book

book

book

book

book

book

book

book

book

book

book

book

book

book

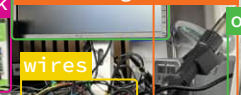
book

book

book

book

soldering station



wires

wires

wires

wires

wires

wires

wires

wires

wires

wires

wires

wires

wires

wires

wires

wires

wires

wires

wires

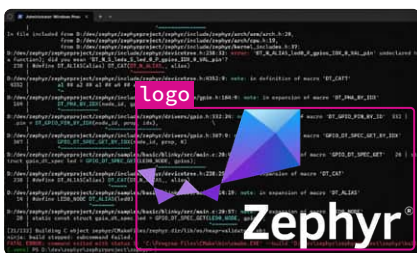
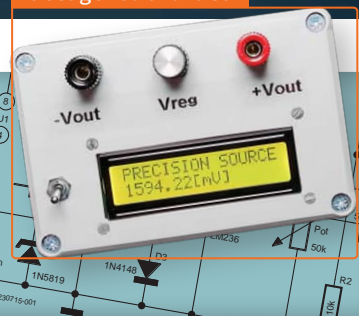
wires

wires

wires

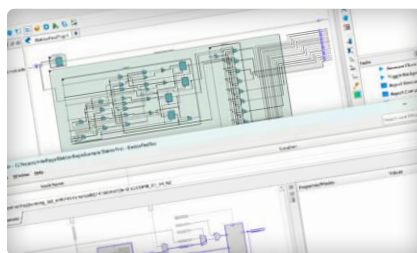
wires

voltage calibrator



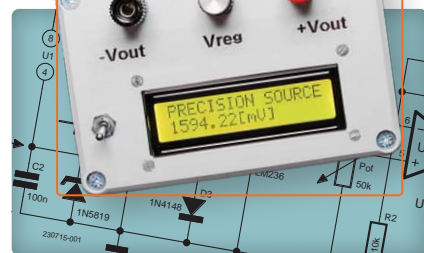
Getting Started With  
the Zephyr RTOS  
As Powerful as It Is Hard  
to Master

p. 98



FPGAs for Beginners  
The Path From MCU to FPGA  
Programming

p. 22



Voltage Reference With  
Arduino Pro Mini  
Linearize and Calibrate Your  
Analog Inputs

p. 14



# Join the Elektor Community



Take out a  
membership!



- ✓ 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 5000 Gerber files



## Also available

The Digital  
membership!



- ✓ The Elektor web archive from 1974!
- ✓ 8x Elektor Magazine (digital)
- ✓ 10% discount in our web shop and exclusive offers
- ✓ Access to more than 5000 Gerber files



[www.elektormagazine.com/Member](http://www.elektormagazine.com/Member)



Volume 50, No. 528  
March & April 2024  
ISSN 1757-0875

Elektor Magazine is published 8 times a year by  
**Elektor International Media b.v.**  
PO Box 11, 6114 ZG Susteren, The Netherlands  
Phone: +31 46 4389444  
[www.elektor.com](http://www.elektor.com) | [www.elektormagazine.com](http://www.elektormagazine.com)

**For all your questions**  
[service@elektor.com](mailto:service@elektor.com)

**Become a Member**  
[www.elektormagazine.com/membership](http://www.elektormagazine.com/membership)

**Advertising & Sponsoring**  
Büsa Kas  
Tel. +49 (0)241 95509178  
[büsa.kas@elektor.com](mailto:büsa.kas@elektor.com)  
[www.elektormagazine.com/advertising](http://www.elektormagazine.com/advertising)

**Copyright Notice**  
© Elektor International Media b.v. 2024

The circuits described in this magazine are for domestic and educational use only. All drawings, photographs, printed circuit board layouts, programmed integrated circuits, digital data carriers, and article texts published in our books and magazines (other than third-party advertisements) are copyright Elektor International Media b.v. and may not be reproduced or transmitted in any form or by any means, including photocopying, scanning and recording, in whole or in part without prior written permission from the Publisher. Such written permission must also be obtained before any part of this publication is stored in a retrieval system of any nature. Patent protection may exist in respect of circuits, devices, components etc. described in this magazine. The Publisher does not accept responsibility for failing to identify such patent(s) or other protection. The Publisher disclaims any responsibility for the safe and proper function of reader-assembled projects based upon or from schematics, descriptions or information published in or in relation with Elektor magazine.

**Print**  
Senefelder Misset, Mercuriusstraat 35,  
7006 RK Doetinchem, The Netherlands

**Distribution**  
IPS Group, Carl-Zeiss-Straße 5  
53340 Meckenheim, Germany  
Phone: +49 2225 88010



## Jens Nickel

*International Editor-in-Chief, Elektor Magazine*



## Set Yourself Goals!

In an editorial long ago, I wrote that it was one specific goal that led me to become an ambitious programmer. Back then, for our internal article management, I had to familiarize myself with VBA for Excel, MS Access, C#, HTML + JavaScript, and much more. I learned an incredible amount, which was also useful for my work as an editor and various other projects. I would never have come this far by merely studying tutorials and undertaking a few small practice projects. A few professionals who evaluated my system were likewise amazed. They themselves had often contemplated creating their own editing system but had never completed it.

And the principle applies to hardware as well. As many of you know, I came to electronics in a roundabout way. I enjoyed tinkering and programming, but I always left the board design to my colleagues. Recently, with a friend, I embarked on a new, larger electronics project — remote control (and remote measurement) of modified audio amplifiers. Suddenly, I was “forced” to familiarize myself with KiCad, various connectors, touch-display programming, and several options from GitHub. And, that’s certainly not the end of the story — I will report on this at [elektor-labs.com](http://elektor-labs.com) soon.

What do I mean by this? Set yourself goals and dare to tackle things that seem a bit too daunting at first glance. You will be driven by the ambition to turn your idea into reality, and this will provide you with enough enthusiasm to venture into new territories. Occasionally round off your project, no matter how rough it looks and how uneven the progress. The sense of achievement will motivate you to want to improve everything.

Right in this issue, you can start working on AI-supported image processing. Have you been intending to do that for a long time? Great! It’s not that difficult to develop your own presentable application (pages 6 and 86). Enjoy our gateway to a new world full of ideas!



### Submit to Elektor!

Your electronics expertise is welcome! Want to submit an article proposal, an electronics tutorial on video, or an idea for a book? Check out Elektor’s Author’s Guide and Submissions page:

[www.elektormagazine.com/submissions](http://www.elektormagazine.com/submissions)



### ElektorLabs Ideas & Projects

The Elektor Labs platform is open to everyone. Post electronics ideas and projects, discuss technical challenges and collaborate with others.

[www.elektormagazine.com/labs](http://www.elektormagazine.com/labs)

### The Team

**International Editor-in-Chief:** Jens Nickel | **Content Director:** C. J. Abate | **International Editorial Staff:** Asma Adhimi, Roberto Armani, Eric Bogers, Jan Buiting, Stuart Cording, Rolf Gerstendorf (RG), Ton Giesberts, Ouafae Hassani, Hedwig Hennekens, Saad Imtiaz, Alina Neacsu, Dr. Thomas Scherer, Jean-Francois Simon, Clemens Valens, Brian Tristram Williams | **Regular Contributors:** David Ashton, Tam Hanna, Ilse Joostens, Prof. Dr. Martin Ossmann, Alfred Rosenkränzer | **Graphic Design & Prepress:** Harmen Heida, Sylvia Sopamena, Patrick Wielders | **Publisher:** Erik Jansen | **Technical questions:** [editor@elektor.com](mailto:editor@elektor.com)



## Regulars

- 3 Colophon**
- 29 STM32 Wireless Innovation Design Contest 2024**  
An Update
- 52 2024: An AI Odyssey**  
First Forays Into TensorFlow
- 56 Project in Brief**  
262,144 Ways to Play The Game of Life
- 62 From Life's Experience**  
The Chinese Dragon
- 77 Starting Out in Electronics...**  
...More About Opamps
- 84 Peculiar Parts**  
Piezoelectric Devices
- 112 Err-electronics**  
Corrections, Updates, and Readers' Letters

## Features

- 22 FPGAs for Beginners**  
The Path From MCU to FPGA Programming
- 30 Bluetooth LE with MAUI**  
Control Apps for Android & Co.
- 64 Get Your (Brushed DC) Motor Running!**  
Sample Projects from the Elektor Motor Control Development Bundle
- 80 ESP Library Recommendations**
- 95 ESP32 Terminal**  
A Handheld Device with a Touch-Capable Display
- 98 Getting Started With the Zephyr RTOS**  
As Powerful as It Is Hard to Master

## Industry

- 92 Resolve Your Trickiest Embedded Development Challenges**
- 107 Ethics in Electronics**  
A Dialog with CTO Alexander Gerfer of Würth Elektronik eiSos



**CaptureCount**  
An Object Detector  
and Counter on  
the Raspberry Pi 5

6





## Projects

- 6 CaptureCount**  
An Object Detector and Counter on the Raspberry Pi 5
- 14 Voltage Reference With the Arduino Pro Mini**  
Linearize and Calibrate Your Analog Inputs
- 38 Port-Expanding Breakout Board**  
Increase the Number of I/Os on Your Dev Board
- 44 AI Specialist**  
Machine Learning with the Jetson Nano
- 70 ESP32-to-RS-232 Adapter**  
A Wireless Link for Classic Test Equipment
- 86 A Smart Object Counter**  
Image Recognition Made Easy with Edge Impulse



**Voltage Reference With the Arduino Pro Mini**  
Linearize and Calibrate Your Analog Inputs

14

## Next Edition

### Elektor Magazine May & June 2024

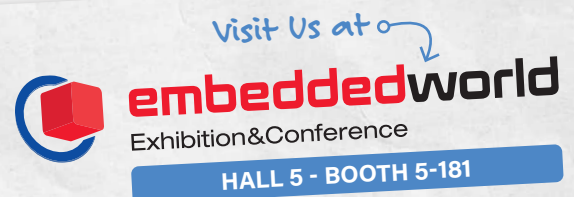
We'll again have an exciting mix of projects, fundamentals, and tips and tricks for engineers and makers. We'll focus on Test & Measurement.

#### From the contents:

- > Human Presence Detector
- > ECG Graph Monitoring
- > Gain Phase Analyzer with Sound Interface
- > Repairing Electronic Equipment
- > DDS Signal Generator
- > In-Circuit LC Meter
- > Power Meter Interface (5 A / 60 V)
- > Active Stroboscopic Disc for Turntables

and much more!

Elektor Magazine May & June 2024 edition will be published around **May 15th**. Gold Members' print edition arrival is subject to transport.



FOCUS ON

**Embedded & AI**

# CaptureCount

## An Object Detector and Counter on the Raspberry Pi 5

By Saad Imtiaz (Elektor)

Machine vision is a fascinating technology that enables the recognition and classification of a wide array of objects in our surroundings. This article presents an intriguing Raspberry Pi project utilizing the YOLO object detection model. The project is adept at identifying numerous objects, ranging from common items such as cars and bicycles to various animals like cats, dogs, and birds.

The project began with a clear goal: to create an all-round object detection and counting system that recognizes the type of object and then counts the object of each category it detects. The Raspberry Pi was an obvious choice due to its capabilities and support in AI projects. As this was one of my first computer vision projects, it took me some time to learn Python and to use the right tools and libraries to develop this project. As there are so many projects, resources [1][2] and our beloved ChatGPT, it was not difficult to learn and develop such a project.

### Object Detection

In the quest of making this project, the journey began with finding the right object detection model, preferably a model that can detect and recognize a wide variety of objects with minimal training time. But before looking for such an object detection model, let's take a step back and talk about how object detection works.

Object detection is a technology that allows computers to identify and locate objects within an image or video. Unlike image recognition, which tells you what is in an image, object detection goes further by pinpointing exactly where those objects are and outlining them. This is typically achieved through two key processes:

- **Object Localization:** Determines the location of a single object in an image. The model outputs the coordinates of the bounding box surrounding the object.



Figure 1: CaptureCount running on a Raspberry Pi 5 with the Raspberry Pi Camera Module 3 (Wide).

- **Object Classification:** Identifies what the object in the bounding box is from a set of known categories.

Advanced object detection models integrate these two steps, efficiently processing an image to provide both the location and the classification of multiple objects within it.

### Exploring Object Detection Models

The search for the ideal object detection model involved evaluating various options, each with its strengths and limitations. Models like R-CNN and its derivatives offered high accuracy but at a slower processing speed, unsuitable for real-time applications. Single Shot Multibox Detector (SSD) presented a faster alternative, though with a trade-off in accuracy. Mask R-CNN provided precise detection, but was too complex for our needs. Ultimately, YOLO (You Only Look Once) was selected for its optimal balance of speed, efficiency, and reasonable accuracy. Its ability to process images in real-time and its simplicity, coupled with the strong support from the AI community, made YOLO the ideal choice for our project's goals.





The selection of the YOLO model was a pivotal point in this project. YOLO by Joseph Redmon [3], is renowned for its ability to perform real-time object detection — a critical feature for any detection system. What drew me to YOLO was its unique approach to object detection.

Unlike traditional methods that process an image in multiple stages, YOLO does it in a single pass. This accelerates the process and enhances the model's ability to generalize from its training, making it more effective in detecting objects in various and unpredictable real-world settings. This capability was crucial for this project, which aimed to recognize a diverse range of objects.

## Hardware Software Integration and Setup

The project required the Raspberry Pi 5, known for its enhanced processing power, and a compatible camera module. The integration process involved setting up the Raspberry Pi OS, connecting the camera, and installing YOLO. To get started with the Raspberry Pi, first it needs an OS to be installed onto its microSD card. That can simply be done by downloading the Raspberry Pi Imager from the Raspberry Pi's official website [4]. Then, run the imager, select the correct device, select the latest Raspberry Pi OS (64 bit), select the microSD card, and then finally just click Next. In a few minutes, the Raspberry Pi OS will be installed onto the microSD card. After all that, insert the SD Card into the Raspberry Pi's microSD card slot, and power it. You will also need a Monitor, Mini HDMI to HDMI Cable, a keyboard, and a mouse to interact with the Raspberry Pi for the first time, but after setting up VNC Server or SSH on to it, the Raspberry Pi can be controlled remotely via your computer over Wi-Fi or Ethernet connection.

Now it's time to install the required libraries for this project. Installing the libraries is basic; it is done through the Terminal. Before installing any new libraries, it's recommended that we update the system's packages, that is done by typing:

```
sudo apt-get update && sudo apt-get upgrade
```

This might take some time to complete and after that, the libraries we need for the project can be installed, which can be done with the following commands:

### ➤ Installing Image I/O Packages

```
sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev
```

### ➤ Setting up Video I/O Packages and GTK Development Library

```
sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
sudo apt-get install libxvidcore-dev libx264-dev
sudo apt-get install libgtk2.0-dev
```

### ➤ Additional Dependencies for OpenCV

```
sudo apt-get install libatlas-base-dev gfortran
```

### ➤ Installing pip and pipx (Package Management Tool)

```
sudo apt-get install python3-pip
sudo apt install pipx -y
```

### ➤ Installing Numpy, Pandas and Open CV

```
pip install numpy
pip install pandas
sudo apt install python3-opencv
```

After installing all the libraries, the important part is to ensure seamless communication between the hardware and the software, enabling the Raspberry Pi to process live camera feeds effectively. To keep things compact, the Raspberry Pi Camera was used hooked up to the MIPI camera port on the Raspberry Pi 5. Now everything is set up, so let's discuss the project's code.

## Diving Into the Code

The core of this project lies in its Python code, which is available on GitHub [4]. The script starts with importing essential libraries like OpenCV for image processing, Pandas for data handling, and Numpy for numerical operations.

```
import cv2
import pandas as pd
import numpy as np
import subprocess
import os
from datetime import datetime
```

At the outset, the script imports essential libraries. *cv2* (OpenCV) is crucial for image processing tasks. *pandas* and *numpy* handle data manipulation and numerical calculations, respectively. *subprocess* and *os* are standard Python libraries for interacting with the system, like running external commands and handling file paths. *datetime* is used for timestamping detections.

```
model = './yolo/yolov3.weights'
config = './yolo/yolov3.cfg'
net = cv2.dnn.readNetFromDarknet(config, model)
```

The script specifies the paths to YOLO's pre-trained weights and configuration file, then loads them using OpenCV's deep neural network (DNN) module. This step initializes the YOLO model for object detection.

```
classes = []
with open("./yolo/coco.names", "r") as f:
    classes = [line.strip() for line in f.readlines()]
```

The *coco.names* file, containing names of objects that YOLO can detect, is read line by line to create a list of class names. Several functions

are defined to streamline the detection process. Let us take a look in **Listing 1** for the initial set parameters and functions for the main loop.

The function `get_output_layers(net)` extracts the names of the output layers of the YOLO network. YOLO's architecture has multiple output layers, and this function helps identify them for processing the detections.

`draw_bounding_box(...)` draws rectangles (bounding boxes) around detected objects and labels them with the object's name and confidence score.

`calculate_centroid(x, y, w, h)` calculates the center point of the detected object's bounding box, a critical step for tracking objects across frames.

```
def capture_image(image_path):
    subprocess.run(["libcamera-still", "-o", image_path,
"--width", "1920", "--height", "1080", "-n", "-t", "1"],
stdout=subprocess.DEVNULL, stderr=subprocess.STDOUT)
```

This function uses the `subprocess` module to run `libcamera-still`, a command-line tool on Raspberry Pi OS, which captures an image from the camera and saves it to the specified path. The resolution is set to 1,920×1,080. A lower resolution can also be used to minimize the CPU and GPU usage as well. To do so, the above code snippet can be adjusted as follows:

```
def capture_image(image_path):
    subprocess.run(["libcamera-still", "-o", image_path,
"--width", "1280", "--height", "720", "-n", "-t", "1"],
stdout=subprocess.DEVNULL, stderr=subprocess.STDOUT)
```

Before entering the main detection loop, the script initializes a Pandas `DataFrame` to log detections with timestamps and a dictionary to keep track of the count of each object type. `centroid_tracking` is used to track the movement of objects.

```
data_frame = pd.DataFrame(columns=['Timestamp', 'Type',
'Count'])
object_counts = {cls: 0 for cls in classes}
centroid_tracking = {}
```

The following portion of the code (see **Listing 2**) checks if the detected object is new or has been previously identified. It uses `centroid_tracking` to determine this. If an object is identified as new, it increments the count for that object type, logs the detection time, and updates a Pandas `DataFrame`. This `DataFrame` can later be exported as a .csv file for further analysis.

## Setting Up the Main Detection Loop

The main detection loop of the Raspberry Pi 5 and YOLO system is a crucial part of its functionality. It starts with capturing an image from the Raspberry Pi camera, converting it into a format suitable for

YOLO processing. The system then processes this image through YOLO, extracting vital information like class IDs, confidence scores, and bounding box coordinates.

To refine the detections, Non-Maximum Suppression (NMS) is applied, filtering out less confident and overlapping detections. The system then draws bounding boxes around the detected objects and, if new objects are identified, saves these images. Throughout this process, the system updates tracking and logging data structures, maintaining a continuous record of detections. This loop is central to the real-time object detection capabilities of the Raspberry Pi 5 and YOLO system. See **Listing 3** for the main loop of the complete code.

After processing the detections, the script compiles the data into a `DataFrame` and saves it as a .csv file. This provides a detailed record of each detection event.

## CaptureCount in Action

When put into action, the CaptureCount demonstrated good object detection, albeit with some intriguing quirks. The Raspberry Pi was mounted on the camera tripod and the view of the camera was outside the window as seen in **Figure 2**. As you can see, I live in an urban side of the city, and the only things the CaptureCount detects are cars, trucks,



Figure 2: The view for the camera.

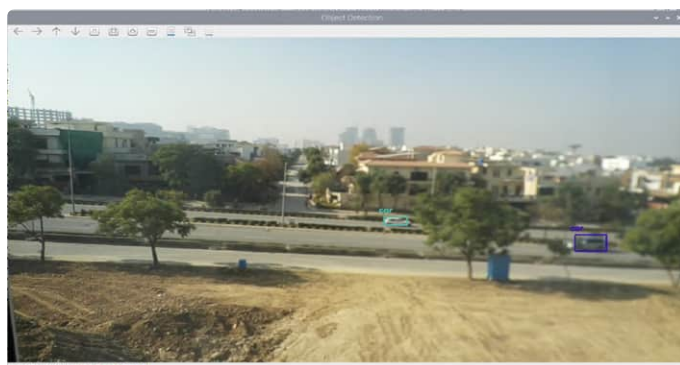


Figure 3: Two cars detected by CaptureCount.



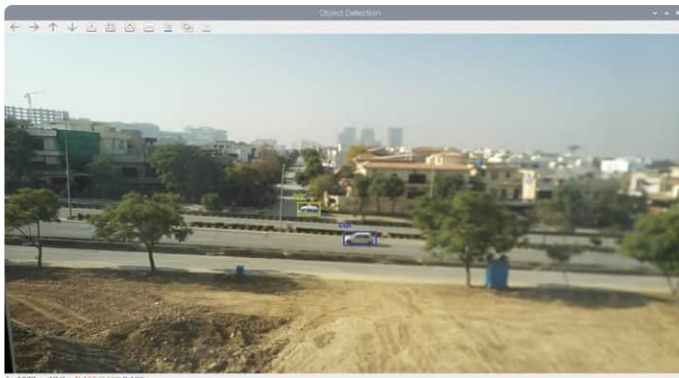


Figure 4: Picture of the cars detected by the system.

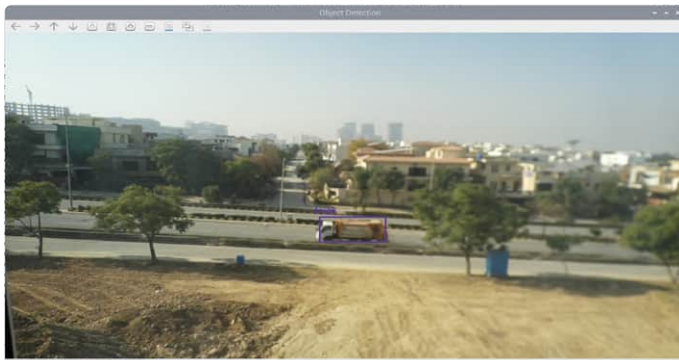


Figure 5: Truck being detected by the system.

people, and motorcycles. It would have been so much better if it was a countryside setting as I would have captured some wildlife with it.

The majority of its performance was characterized by an admirable 80% accuracy rate in identifying and categorizing various objects. This level of precision was particularly noteworthy considering the camera's placement, which was relatively far from the objects, coupled with the use of a wide-angle lens. In **Figure 3**, **Figure 4**, and **Figure 5**, the images of the detected objects are shown. In **Figure 6** and **Figure 7** the entire log is shown in a .csv of when which object was detected and how many objects on the same category were detected during various time intervals.

## Encountered Challenges and Unique Observations

Despite its successes, the system faced some interesting challenges:

- **Selective Detection in Dynamic Settings:** Instances were noted where the system identified a person on a motorcycle but missed detecting the motorcycle itself. This selective detection highlighted a challenge in distinguishing closely associated objects, particularly when in motion.
- **Occasional Misclassifications:** There were rare occurrences where pedestrians were misclassified as bicycles. This misclassification points to the complexities involved in object classification, especially when objects share similar spatial characteristics.

## A Path Forward for CaptureCount

In summary, this project, harnessing the Raspberry Pi 5 and YOLO, demonstrated notable performance, opening doors for diverse enhancements and applications. The system's occasional detection anomalies and misclassifications not only underscore areas for future improvements but also open opportunities for further research.

	A	B	C	D
1	Timestamp	Type	Count	
2	2023-12-09 12:32:13.076846	car	1	
3	2023-12-09 12:32:15.389343	car	1	
4	2023-12-09 12:32:22.641547	car	1	
5	2023-12-09 12:32:51.806253	car	1	
6	2023-12-09 12:33:15.493817	car	1	
7	2023-12-09 12:33:37.409609	car	1	
8	2023-12-09 12:33:49.290162	car	1	
9	2023-12-09 12:33:51.289117	car	1	
10	2023-12-09 12:33:53.289954	car	1	
11	2023-12-09 12:33:55.269961	car	1	
12	2023-12-09 12:33:57.246952	car	1	
13	2023-12-09 12:34:01.136601	car	1	
14	2023-12-09 12:34:18.922022	person	1	
15	2023-12-09 12:34:22.908511	person	1	
16	2023-12-09 12:34:32.785654	person	1	
17	2023-12-09 12:34:54.565159	truck	1	

Figure 6: Screenshot of the .csv file that shows the detected objects with their respective timestamps.

	A	B	C	D
1	Type	Total Count		
2	person	3		
3	bicycle	0		
4	car	12		
5	motorbike	0		
6	aeroplane	0		
7	bus	0		
8	train	0		
9	truck	1		
10	boat	0		
11	traffic light	0		
12	fire hydrant	0		
13	stop sign	0		
14	parking meter	0		

Figure 7: Screenshot of the .csv file that shows the total sum of objects detected in a single category.



Future enhancements could include the integration of servo motors for targeted object tracking and IoT connectivity for automated responses. Just one example: Blinking an RGB LED for a specific type of object. To give you hints how to expand your project with this or any other hardware control, see **Listing 4**. This function turns on the red LED for a car, green for a person, and both red and green (creating yellow) for a truck. After a delay, it turns off all LEDs.

Furthermore, modifications can be made for broadened applications in advanced surveillance, traffic and crowd management, wildlife monitoring, retail analytics, and healthcare. This system's proficiency in real-time, precise object detection lays the foundation for innovative solutions across various industries, showcasing the expansive possibilities in the field of AI and computer vision. ◀

230749-01

### Questions or Comments?

If you have questions about this article, feel free to email the author at [saad.imtiaz@elektor.com](mailto:saad.imtiaz@elektor.com) or the Elektor editorial team at [editor@elektor.com](mailto:editor@elektor.com).

### About the Author

Saad Imtiaz (Senior Engineer, Elektor) is a mechatronics engineer with experience in embedded systems, mechatronic systems, and product development. He has collaborated with numerous companies, ranging from startups to enterprises globally, on prototyping and development. Saad has also spent time in the aviation industry and has led a technology startup company. At Elektor, he drives project development in both software and hardware.



### Related Products

- > **Raspberry Pi 5 Ultimate Starter Kit (8 GB)**  
[www.elektor.com/20721](http://www.elektor.com/20721)
- > **Raspberry Pi High Quality Camera Module**  
[www.elektor.com/19279](http://www.elektor.com/19279)
- > **Raspberry Pi Camera Module 3 Wide**  
[www.elektor.com/20364](http://www.elektor.com/20364)

### WEB LINKS

- [1] Implementation of YOLOv3: Simplified: <https://analyticsvidhya.com/blog/2021/06/implementation-of-yolov3-simplified>
- [2] YOLOv3 code explained: <https://pylessons.com/YOLOv3-code-explanation>
- [3] YOLO: Real-Time Object Detection: <https://pjreddie.com/darknet/yolo>
- [4] CaptureCount Github Repository: <https://github.com/ElektorLabs/CaptureCount>



### Listing 1: Detection functions

```
import cv2
import pandas as pd
import numpy as np
import subprocess
import os
from datetime import datetime

# Load pre-trained model and configuration
model = './yolo/yolov3.weights' # Update this path to your model's path
config = './yolo/yolov3.cfg'    # Update this path to your config file's path
net = cv2.dnn.readNetFromDarknet(config, model)

# Load class names
classes = []
with open("./yolo/coco.names", "r") as f: # Update to the path of your coco.names file
    classes = [line.strip() for line in f.readlines()]

# Function to get output layers
def get_output_layers(net):
    layer_names = net.getLayerNames()
    return [layer_names[i - 1] for i in net.getUnconnectedOutLayers().flatten()]
```



```
# Function to draw bounding box
def draw_bounding_box(img, class_id, confidence, x, y, x_plus_w, y_plus_h):
    label = str(classes[class_id])
    color = np.random.uniform(0, 255, size=(3,))
    cv2.rectangle(img, (x, y), (x_plus_w, y_plus_h), color, 2)
    cv2.putText(img, label, (x-10, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

# Function to calculate centroid of a bounding box
def calculate_centroid(x, y, w, h):
    return (int(x + w/2), int(y + h/2))

# Function to capture image using libcamera-still
def capture_image(image_path):
    subprocess.run(["libcamera-still", "-o", image_path, "--width", "1920", "--height", "1080", "-n", "-t",
                    "1"], stdout=subprocess.DEVNULL, stderr=subprocess.STDOUT)

# Initialize a DataFrame to store counts
data_frame = pd.DataFrame(columns=['Timestamp', 'Type', 'Count'])
object_counts = # Object count per category
centroid_tracking = {} # Tracks centroids of detected objects

# Ensure output directory exists
output_dir = "output"
if not os.path.exists(output_dir):
    os.makedirs(output_dir)
```



## Listing 2: Centroid tracking — detect if object is new or previously-detected

```
# Check if this object was already detected
if class_id not in centroid_tracking or not any(np.linalg.norm(np.array(centroid) - np.array(old_centroid))
                                                < 50 for old_centroid in centroid_tracking[class_id]):

    object_counts[classes[class_id]] += 1
    centroid_tracking.setdefault(class_id, []).append(centroid)
    new_row = pd.DataFrame([{'Timestamp': datetime.now(), 'Type': classes[class_id], 'Count': 1}])
    data_frame = pd.concat([data_frame, new_row], ignore_index=True)
```



## Listing 3: Main loop

```
# Main loop
image_path = "temp.jpg"
object_id = 0 # Unique identifier for each object

try:
    while True:
        capture_image(image_path)
        frame = cv2.imread(image_path)
        if frame is None:
            continue

        Width = frame.shape[1]
        Height = frame.shape[0]
        scale = 0.00392
```

(Continues on the next page)

```

# Create a blob and pass it through the model
blob = cv2.dnn.blobFromImage(frame, scale, (416, 416), (0, 0, 0), True, crop=False)
net.setInput(blob)
outs = net.forward(get_output_layers(net))

# Process the outputs
class_ids = []
confidences = []
boxes = []
centroids = []
conf_threshold = 0.5
nms_threshold = 0.4

for out in outs:
    for detection in out:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > conf_threshold:
            center_x = int(detection[0] * Width)
            center_y = int(detection[1] * Height)
            w = int(detection[2] * Width)
            h = int(detection[3] * Height)
            x = center_x - w / 2
            y = center_y - h / 2
            class_ids.append(class_id)
            confidences.append(float(confidence))
            boxes.append([x, y, w, h])
            centroids.append(calculate_centroid(x, y, w, h))

indices = cv2.dnn.NMSBoxes(boxes, confidences, conf_threshold, nms_threshold)

for i in indices:
    box = boxes[i]
    x, y, w, h = box
    x, y, w, h = int(x), int(y), int(w), int(h) # Ensure the values are integers

    centroid = centroids[i]
    class_id = class_ids[i]

    # Check if this object was already detected
    if class_id not in centroid_tracking or not any(np.linalg.norm(np.array(centroid) -
        np.array(old_centroid)) < 50 for old_centroid in centroid_tracking[class_id]):
        object_counts[classes[class_id]] += 1
        centroid_tracking.setdefault(class_id, []).append(centroid)

    # Update data_frame using pandas.concat
    new_row = pd.DataFrame([{'Timestamp': datetime.now(), 'Type': classes[class_id], 'Count': 1}])
    data_frame = pd.concat([data_frame, new_row], ignore_index=True)

    # Save the entire frame when an object is detected
    frame_filename = os.path.join(output_dir, f"frame_{object_id}.jpg")
    cv2.imwrite(frame_filename, frame)
    object_id += 1

    draw_bounding_box(frame, class_id, confidences[i], round(x), round(y), round(x+w), round(y+h))

# Display the frame
cv2.imshow("Object Detection", frame)

# Break loop with 'q' key

```



```

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
finally:
    cv2.destroyAllWindows()
    if os.path.exists(image_path):
        os.remove(image_path)

# Print and save the total count of objects detected per category
total_counts = pd.DataFrame(object_counts.items(), columns=['Type', 'Total Count'])
print(total_counts)
total_counts.to_csv("total_object_counts.csv", index=False)

# Save detailed counts to CSV
data_frame.to_csv("object_counts.csv", index=False)

# Write the total count to a text log
with open("total_counts_log.txt", "w") as log_file:
    log_file.write(str(total_counts))

```



#### Listing 4: RGB LED blinking for detected objects.

```

#include these libraries in the shared code
import RPi.GPIO as GPIO
import time

# GPIO pin setup
RED_PIN, GREEN_PIN, BLUE_PIN = 17, 27, 22

# Update with your GPIO pin numbers

# add this part in the initial part of the code
GPIO.setmode(GPIO.BCM)
GPIO.setup([RED_PIN, GREEN_PIN, BLUE_PIN], GPIO.OUT)

def blink_rgb_led(object_type):
    GPIO.output(RED_PIN, object_type == 'car' or object_type == 'truck')
    GPIO.output(GREEN_PIN, object_type == 'person' or object_type == 'truck')
    GPIO.output(BLUE_PIN, False)
    time.sleep(1)
    GPIO.output([RED_PIN, GREEN_PIN, BLUE_PIN], False)

# Usage of this function:

# ... [Previous code] ...
for out in outs:
    for detection in out:
        # ... [add the following lines to the code to this 'for' loop]
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > conf_threshold:
            detected_object = classes[class_id]
            blink_rgb_led(detected_object) # Blink the LED based on the detected object

# ... [Rest of your detection and saving logic] ...
# ... [Remaining code] ...

```

# Voltage Reference With Arduino Pro Mini

## Linearize and Calibrate Your Analog Inputs

By Giovanni Carrera (Italy)

If you use the analog inputs of your favorite microcontrollers, sooner or later you have needed to check their linearity and accuracy over the entire range of input values. This precise and compact Arduino-based voltage calibrator, capable of generating stable and accurate reference voltages in the range from 0 to 5 V, will solve this problem.



Many will wonder what a voltage calibrator is and what it is used for. A voltage calibrator is an instrument that outputs known voltages with high accuracy and is used to calibrate or verify the class of voltmeters, ADC and DAC converters, acquisition systems, etc.

In voltage measurements that are made with microcontrollers, it is often necessary to know the conversion constant or verify the linearity of the supplied analog-to-digital converter. If we settle for an accuracy around 5% it is sufficient to divide the  $V_{ref}$  by the maximum number you have at the output, for example with Arduino UNO it is  $2^{10} - 1 = 1023$ . If we use the supply voltage (5 V) as the  $V_{ref}$ , this can vary depending on whether we power the board from USB or other power supply and is also noisy. If we use the internal  $V_{ref}$  (definitely preferable case), for example 1.1 V, we do not know exactly its value. Other MCUs such as the ESP32 do not even have analog inputs starting from 0 V, but from 80 to 150 mV, and the full scale is between 950 and 1100 mV.

To overcome these drawbacks, we need to do a calibration, so we need to use a stable and low-noise voltage source together with a digital voltmeter with higher accuracy than the MCU's converter. Then make a dozen or so measurements of various values within the input range of the ADC, enter the values on an Excel-type spreadsheet and do a linear regression to derive the slope and intercept. We will need these values to get an output in V or mV. This project combines two: a very stable, low-noise voltage generator and an extremely accurate

digital voltmeter with a measuring range of 0 to 5 V and a resolution of about 0.2 mV.

### The Voltage Source

To have a stable, i.e., low thermal drift, reference voltage, it is not sufficient to use an ordinary Zener diode or voltage regulator, but a device designed to provide an output voltage that remains as constant as possible as the supply voltage, temperature change, and over time. For this project, an LM236H-2.5 was used, which has a temperature drift of only 3.5 mV between -25 and +85°C when supplied with a reverse current of 1 mA, as was done. If its behavior were linear, there would be a drift of about 32  $\mu\text{V} / ^\circ\text{C}$  (about 12.8 ppm /  $^\circ\text{C}$ ). Holding temperature and current constant, the change in voltage is about 20 ppm over a 1,000-hour time period.

If slightly higher characteristics are desired, with minor circuit modifications an AD680 can be used instead. If, on the contrary, chips of lower characteristics were also sufficient, one could replace this IC with a more common TL431 without making any circuit modifications. **Figure 1** shows the circuit diagram. The two diodes D2 and D3 are used to further reduce thermal drift, and trimmer Rp1 should be calibrated to have an output voltage of 2.49 V, the value where it achieves the best characteristics. The operational amplifier (more on its function below) used is an MCP6002, a dual rail-to-rail type op-amp, which is capable of operating at voltage levels very close to those of rails,

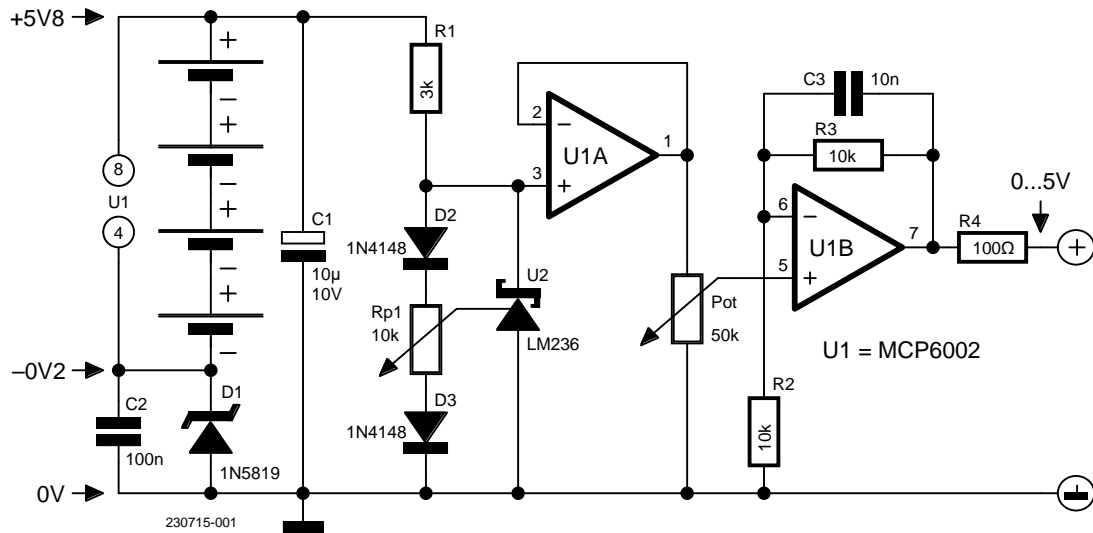


Figure 1: Schematic diagram of the variable voltage source.

unlike normal op-amps that saturate at a few volts difference versus a few tens of mV of rail-to-rail.

The thermal drift of this chip is very small:  $\pm 2 \mu\text{V}/\text{C}^\circ$  and the offset is a few mV. To get even closer to zero, the D1 schottky diode was used. With this stratagem, the negative voltage of the op-amps is below zero by about 100...200 mV (direct voltage of these diodes), and the minimum output voltage is affected only by the offset of the op-amps. As can be seen in the figure, the zero potential or GND is not the negative pole of the batteries but the anode of diode D1, which is precisely why a differential input of the DVM is needed.

With a deterioration in the features of this design, an ordinary LM358 can be used instead of the MCP6002. The U1A op amp acts as a high input impedance separator, and to its output is connected the potentiometer that varies the potential between zero and 2.5 V. The potentiometer used is a 10-turn; the value is not very important; it can vary between 5 k $\Omega$  and 100 k $\Omega$ . The second operational U1B operates as a non-inverting amplifier with gain equal to

$$G = (1 + R3/R2) = 2$$

So the output will be between about 0 V and 5 V. Capacitor C3 serves as a low-pass filter to reduce thermal noise from semiconductors. Resistor R4 serves to limit the output current in case of an accidental short circuit. An output resistance of 100  $\Omega$  creates no problems because it is negligible compared with the input resistances of the systems to be calibrated. The power supply is battery-powered to avoid mains noise, and a switching power supply that has an output noise of a few hundred kHz with peaks of up to 100 mV should be avoided at all costs.

## The High-Precision Digital Voltmeter

To make a calibrator, you need a digital voltmeter with a precision that is a number of digits greater than that of the system we need to calibrate, and with adequate accuracy. The proposed DVM (Digital Volt Meter) has a 16-bit A/D converter, for positive only values there are  $2^{15} = 32,768$  levels from zero, with a maximum full-scale number of 32,767. In this system, the ADC was programmed with a full scale of 6144 mV, so the

resolution is  $6,144 / 32,767 = 0.1875 \text{ mV}$ . The converter does not admit inputs with voltages 0.3 V higher than the supply voltage  $V_{CC}$ , i.e., 5.3 V, and the voltage source is designed for a maximum value of about 5 V. So the maximum number corresponding to 5 V is  $5,000 / 0.1875 = 26,666$ .

This is a very high number compared to that of a normal 3 1/2 digit panel DVM which is 1,999. With the addition of a switch, one can disconnect the voltmeter to use it separately. What does not work is to put a high impedance input divider to measure higher voltages, something that had been tried for other applications and created noise problems that made even two digits vary with stable voltage input.



## Voltage Source Component List

### Resistors

- R1 = 3 k $\Omega$ , 1%, 1/4 W
- R2, R3 = 10 k $\Omega$ , 1%, 1/4 W
- R4 = 100  $\Omega$ , 5% 3 W, wire wound
- Rp1 = 10 k $\Omega$ , multi-turn trimmer
- Pot = 50 k $\Omega$ , potentiometer, linear, 10 turns

### Capacitors

- C1 = 10  $\mu\text{F}$ , 25V, tantalum
- C2 = 100 nF, ceramic
- C3 = 10 nF, ceramic

### Semiconductors

- U1 = MCP6002, op-amp
- U2 = LM236H-2.5, 2.49 V voltage reference
- D1 = 1N5819, schottky diode
- D2, D3 = 1N4148, diode

### Miscellaneous

- SW1 = switch, SP
- B1...B4 = 4 x AA, 1.5 V alkaline cells with battery holder



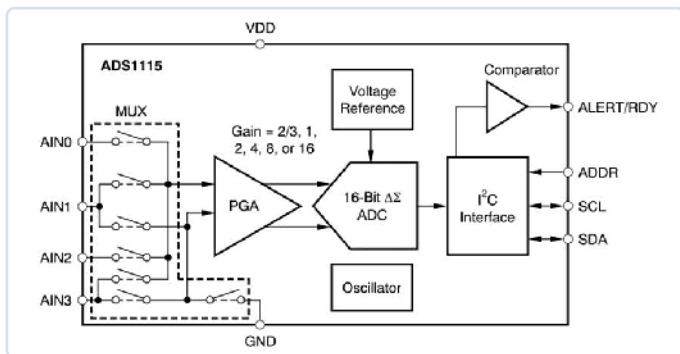


Figure 2: Main components of the ADS1115 converter.  
(Source: <https://ti.com/lit/ds/symlink/ads1115.pdf>)

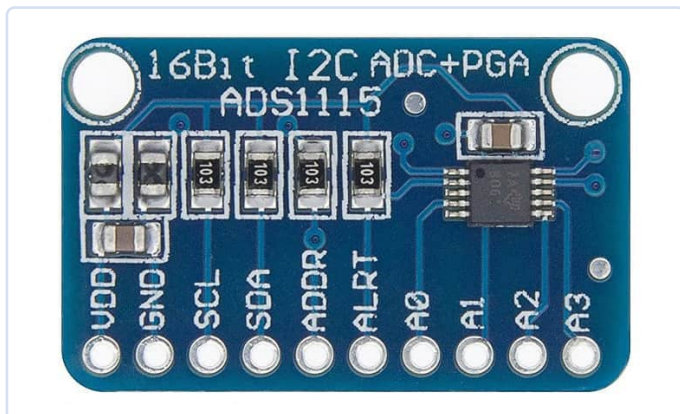


Figure 3: The ADS1115 module.

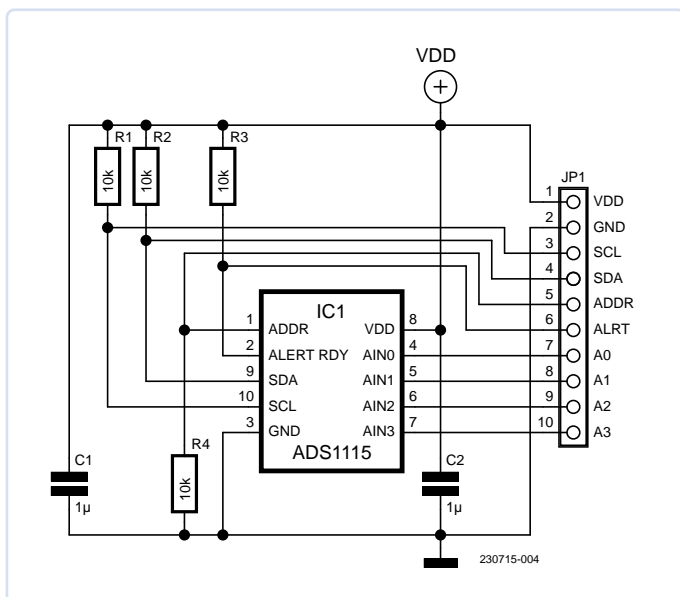


Figure 4: Schematic diagram of the ADS1115 module.

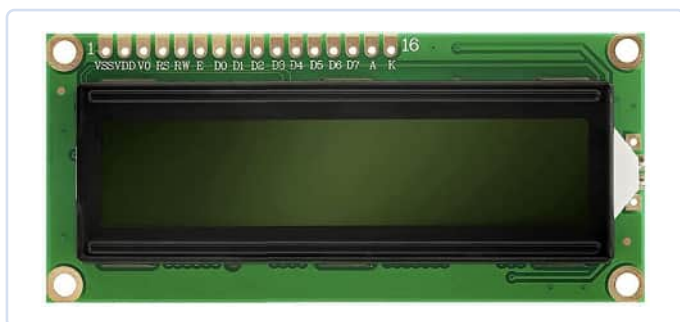


Figure 5: The 16x2 LCD.

## The ADS1115 Converter

The heart of the DVM is the ADS1115 chip, a 16-bit Delta-Sigma type analog to digital converter with high precision and accuracy specifications. It can have up to four inputs. Of the 16 bits, the most significant is reserved for the sign; in fact, the chip can measure negative values, especially when configured in differential mode, but with some limitations related to the supply voltage. This interesting chip, manufactured by Texas Instruments, has remarkable features:

- A resolution of 16 bits, compared to the Arduino's 10 bits.
- Four single ended (referenced to ground) or two differential channels.
- Sampling rate of 8 to 860 samples/second.
- Uses a programmable amplifier (PGA) that allows even small voltages to be measured.
- Has an internal reference source with low thermal drift.
- Can be supplied with voltages from 2 to 5.5 V with power consumption of only 150  $\mu$ A, further reduced in single-shot mode.
- I²C interface with several selectable addresses.
- Has an alert function (ALERT/RDY) to efficiently monitor voltages. It can reduce power consumption, waking-up and make the micro-controller work when certain events occur, or generate interrupts.

Figure 2 shows the functional diagram of the ADS1115 chip, which has two conversion modes:

- **Continuous conversion:** the ADS1115 performs conversions continuously. Once a conversion is completed, the ADS1115 enters the result into the conversion register and immediately starts another conversion.
- **Single-shot conversion:** the ADS1115 waits until the OS bit is set High. Once affirmed, the bit is set to 0, indicating that a conversion is currently in progress. Once the conversion data is ready, the OS bit is reaffirmed and the device shuts down. Writing a 1 to the OS bit during a conversion has no effect.

In this project, single-shot mode with a conversion every half second will be used.

## ADS1115 Module

Modules similar to the one in Figure 3, used in the project, can be found on the market. It has few other components, as seen in the schematic in Figure 4. To use this chip with the Arduino IDE, Wolfgang Ewald's *ADS1115\_WE* library was used, which can be downloaded from the IDE itself (Library Manager) or from the web [1]. It has numerous functions that allow all the features of the chip to be exploited.

## LCD

An LCD compatible with the Hitachi HD44780 controller was chosen. In particular, to have larger characters, the classic two-line 16-character display similar to the one in Figure 5 was used. The reason for this choice is that these displays are very readable, have low-power consumption (without backlighting), and have 5 V power supply and logic levels, which are perfectly compatible with Arduino Pro Mini. The needed Arduino library can be found here [2].

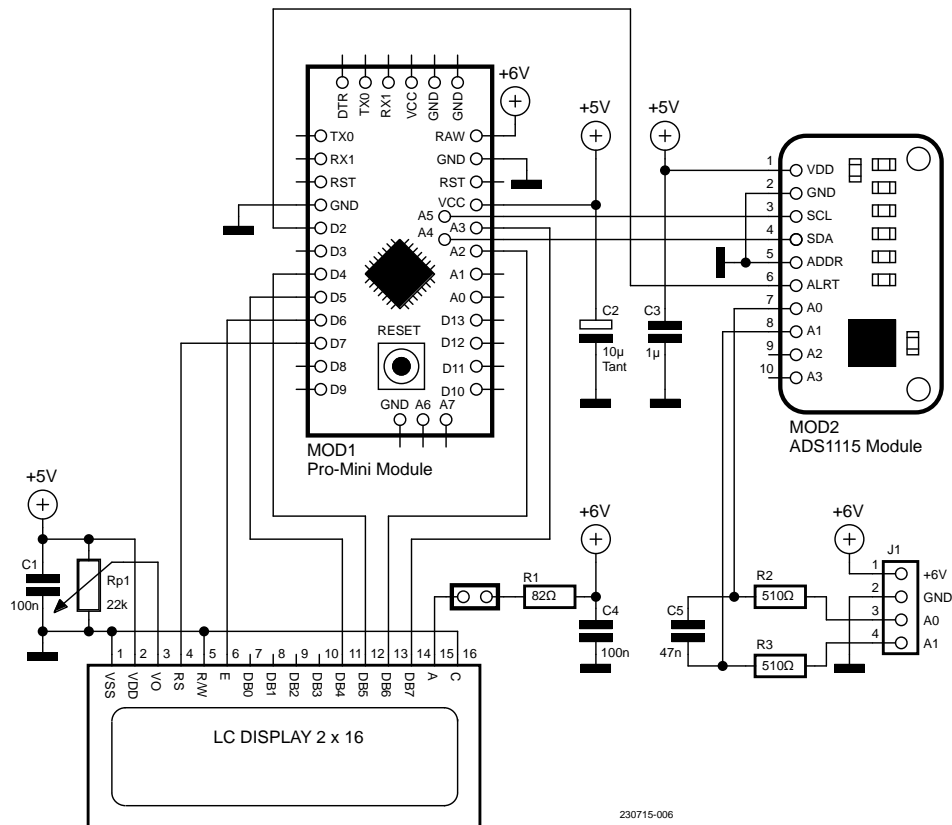


Figure 6: Schematic diagram of the DVM.

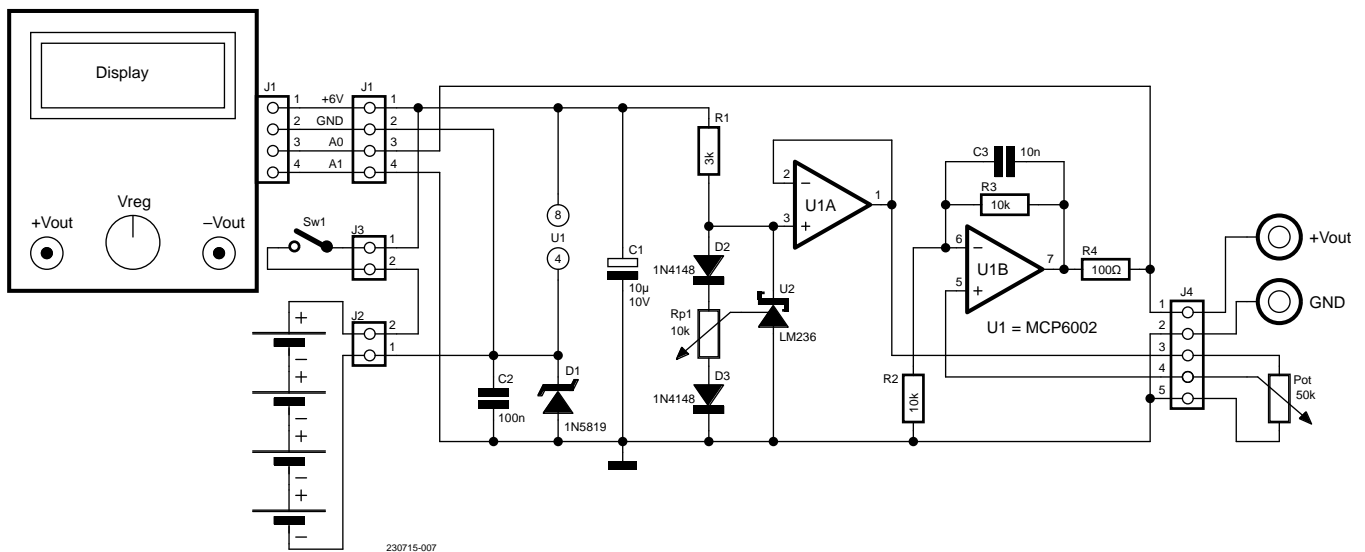


Figure 7: Wiring between the DVM and the voltage source.

## Wiring Diagram of the DVM

**Figure 6** shows the wiring diagram of the DVM. An Arduino Pro Mini, a very compact board, was used as the microcontroller, which also provides the 5 V to power the LCD and converter. In this project, inputs A0 and A1 of the ADC module configured as differential input are used. The consumption of the whole system ranges from about 20 to 25 mA if the LCD backlight is not used. The ideal voltage to be applied would be 6 V; the system also accepts higher voltages (12 V max) but the

tiny built-in regulator may overheat. The backlight current should not be taken from the internal regulator, but supplied by the external one with resistor R1 in series, as in **Figure 6**, which should be calculated according to the required current.

An 82  $\Omega$  resistor was used in the prototype, but a 100  $\Omega$  value can also be used to save a few mA. Older displays required a current even up to 100 mA, as they had old technology LEDs. Recent ones require

maximum currents around 20 mA because they have high-brightness LEDs. A 4-pin connector connects the DVM with the variable voltage source board. Capacitor C5 and 510  $\Omega$  resistors R2 and R3 were used to reduce noise. They constitute a first-order low-pass filter. The time constant is  $\tau = 47.94 \mu\text{s}$ , to which corresponds a cutoff frequency of

$$f = 1/(\tau * 2 * \pi) = 3.320 \text{ kHz}.$$

## Wiring Diagram

Two boards were made in the prototype, one for the DVM and one for the voltage source (**Figure 7**), but there is nothing to prevent mounting all components on one board. The power supply for the digital circuits produces noise due to switching currents, so bypass capacitors were used to reduce noise peaks, such as tantalum capacitor C2.

## The Prototype

**Figure 8** shows what the prototype looks like. The two boards are of the single-sided multi-hole type, the display and ADC converter are connected with 16 and 10 pin, 0.1" pitch strip connectors, and the Arduino Pro module is connected with two 12-pin strips.

**Figure 9** shows the board of the DVM. Unfortunately, pins A4 and A5 of the Arduino Pro Mini's I<sup>2</sup>C bus are not available on the 12+12 pins and are also not on a 2.54 mm grid, so two wires (white and brown, pictured) and a 2 pin connector were soldered to carry the signals to the ADS1115 module. The 16-pin strip connector seen in the upper left is for the LCD, which is connected at 90 degrees. The multi-hole board is very compact and is also attached to the display with a small brass bracket, visible at the top left of the picture.

## Programming

For programming, we need a USB TTL serial adapter similar to the one in **Figure 10**. Of course, we also need to load the adapter driver which, in addition to the Rx and Tx signals, must have the DTR that is used to reset the system. It also serves as a USB serial interface, but in this case, its predominant use is to transfer the sketch, compiled from the Arduino IDE, to our system. In the Arduino IDE, you have to set the board as *Arduino Pro* at 16 MHz and 5 V. Once programmed, if all goes well, you can remove the programmer and power the board from the batteries.

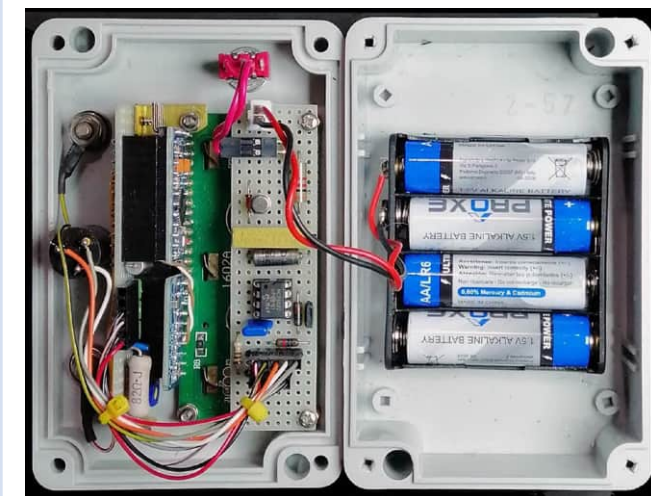


Figure 8: Interior view of the completed prototype.

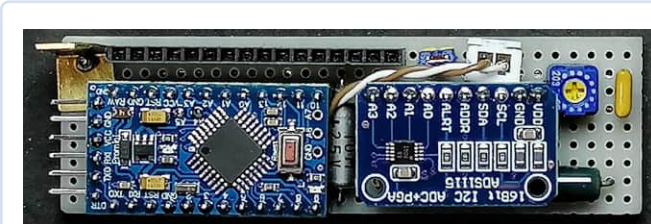


Figure 9: The fully populated DVM board.

The connector is a 6-pin female header with 0.1" pitch, soldered directly to the adapter board connector, which is male, but female-to-female breadboard leads can also be used. Not all serial adapters have the pins arranged as shown or have the same chip. With some serial adapter chips, such as the Prolific PL2303, driver problems have been encountered with Windows 10.

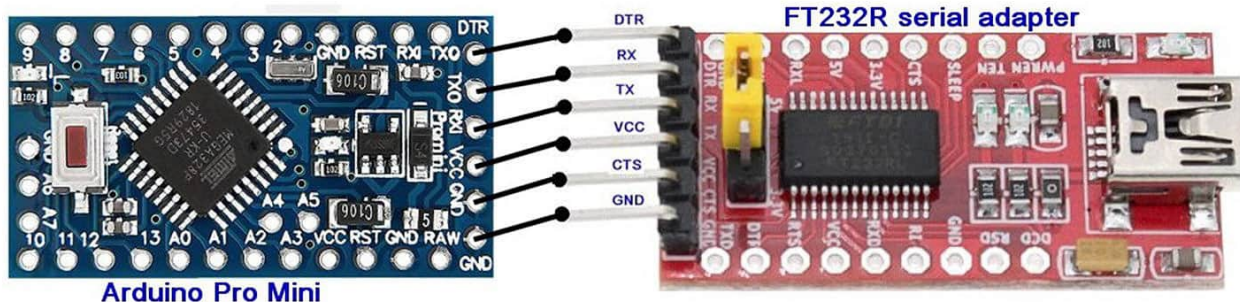


Figure 10: Connections between serial adapter and Arduino Pro Mini.



## Calibration of the DVM

Although the ADS1115 measurements were already very accurate, calibration was done using an HP 3478A bench-top multimeter with 5 1/2 digits with 0.1  $\mu\text{V}$  resolution. To achieve this, we need to replace the line, in the `printData()` function:

```
// calibrated and result in millivolts
voltage = Ch0*1.000515+0.022;
```

with this one, that is, remove the correction:

```
// not calibrated and result in millivolts
voltage = Ch0;
```

About ten values were generated, the reference multimeter readings and the calibrator readings were put on Excel, the *scatter plot* with the *trend line* was entered with *linear* options and checking *Display equation on graph* and *Display R square value on graph*. The result was excellent, as seen in **Figure 11**. The value of R2 was 1 with as many as seven decimal places equal to zero: this corresponds to an extremely high value of linearity. To correct the data, we need to do the inverse formula, that is, invert the axes. So we get:

- > slope = 1.000515286
- > intercept = 0.022110099
- > error = 0.051941236



## DVM Component List

### Resistors

R1 = 82  $\Omega$ ,  $\pm 5\%$ , 1 W, see text  
R2, R3 = 510  $\Omega$ ,  $\pm 1\%$ , 1/4 W  
Rp1 = 22 k $\Omega$ , trimmer

### Capacitors

C1, C4 = 100 nF, 50 V, ceramic  
C2 = 10  $\mu\text{F}$ , 25 V, tantalum  
C3 = 1  $\mu\text{F}$ , 63 V, ceramic  
C5 = 47 nF, 63 V, polyester

### Modules

1 x Arduino Pro Mini (or compatible)  
1 x 16x2 LCD display, with backlighting  
1 x ADS1115 A/D converter

### Miscellaneous

12+12+10+16 Pin Strip, female

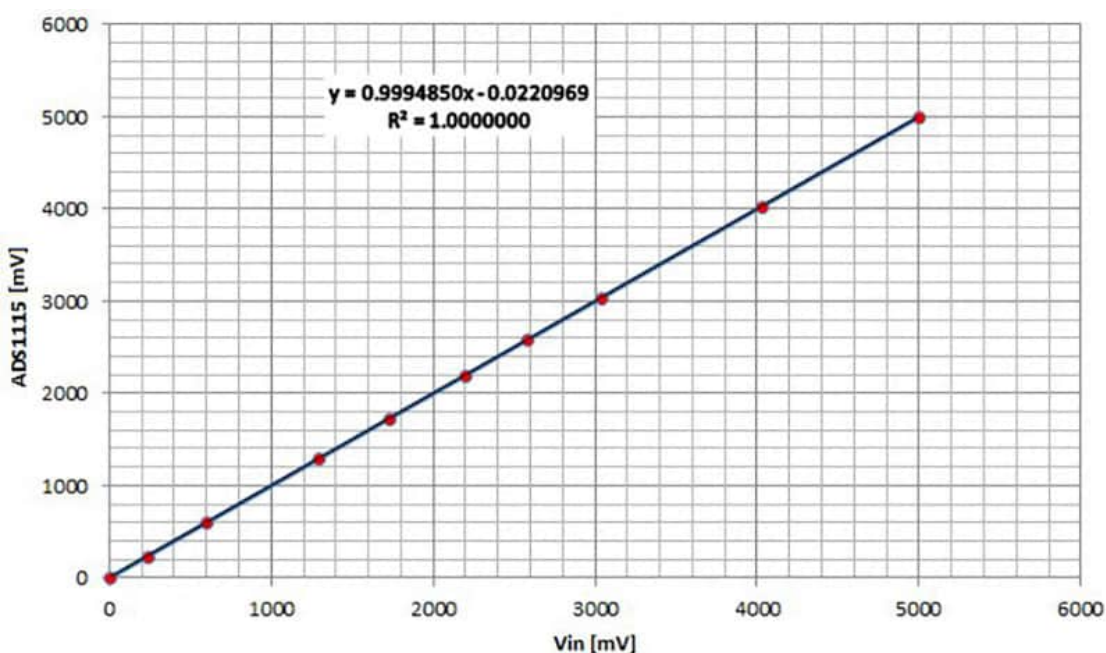


Figure 11:  
DVM calibration  
(linearization) chart.

As can be seen, the error is minimal. If we do not have a very precise instrument available we may not make the correction, settling for a standard error of 0.051941236; otherwise we correct the line with the values we got from our calibration. In the prototype the minimum voltage generated was 1.7 mV, the maximum 4,997.03. The minimum is not equal to zero because of the offset of the op-amps.

## Program Description

The program is relatively simple; after initializing the I/O, display, and ADC converter, it makes a measurement every 500 ms. The `readChannel(ADS1115_MUX channel)` function is the one suggested

for single-shot mode and makes the measurement in millivolts. The `LCDprintln(String text, byte line)` function prints a string on line 0 or 1. The `printData()` function corrects the measurement with the results of the initial calibration and prints it. ◀

230715-01

## Questions or Comments

Do you have technical questions or comments about this article? Please contact the author or Elektor editorial staff at [editor@elektor.com](mailto:editor@elektor.com).



### Listing 1

```
/* program ArduCalibrator.ino for the ADS1115, 16 bit 4 ch ADC
a single differential input (A0,A1), Ch0 6144 mV (5000 generated)
read the precision voltage source
uses an Arduino Pro Mini, LCD 16x2 display
Giovanni Carrera 23/11/2022 with calibration
*/
#include <Wire.h>
#include<ADS1115_WE.h>
#include <LiquidCrystal.h>
ADS1115_WE adc = ADS1115_WE();// uses Wire / ADC Address = 0x48
// LCD pins
#define rs 7
#define en 6
#define d4 5
#define d5 4
#define d6 A2
#define d7 A3
// initialize the library by associating any needed LCD interface pin
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
#define Alert 2 // AD Alert pin
const unsigned long deltat_ms = 500;
unsigned long cms, pms;
float Ch0,Ch1;
void setup(){
  Wire.begin();
  //Serial.begin(115200);
  pinMode(Alert,INPUT);
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  lcd.print(F("ArduCalibrator"));
  lcd.setCursor(0, 1);// print on the second row
  lcd.print(F("GCar V231122"));
  delay(2000);
  if(!adc.init()){
```

```

    lcd.setCursor(0, 1);
    lcd.print(F("No ADS1115 found"));
    while( true );// ends here
}
adc.setVoltageRange_mV(ADS1115_RANGE_6144); // 6144 mV input range
LCDprintln("PRECISION SOURCE", 0);
}
void loop() {
    cms = millis();
    // checks if passed deltat milliseconds
    if(cms - pms > deltat_ms) { // period timebase
        pms = cms;// update pms
        Ch0 = readChannel(ADS1115_COMP_0_1);// Ch0 differential A0(+) with A1(-)
        Ch0 = readChannel(ADS1115_COMP_0_1);// repeating helps
        printData();
    }
}
}

```



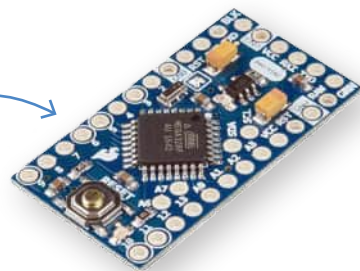
### About the Author

Giovanni Carrera holds a degree in Electronic Engineering. As a university professor in the Faculty of Naval Engineering in Genoa, Italy, he taught numerous courses, such as naval automation and the simulation of ship propulsion systems. Carrera started working in the late 1970s with the 6502 CPU and then moved on to other processors. Today, he enjoys the designing and developing analog and digital electronic circuits, many of which he has written about on his blogs (ArduPicLab and GnssRtkLab) and in various magazines.



### Related Products

- > **SparkFun Arduino Pro Mini 328 (5 V, 16 MHz)**  
[www.elektor.com/20091](http://www.elektor.com/20091)
- > **Standard 2x16 Character back-lit LCD**  
[www.elektor.com/16414](http://www.elektor.com/16414)



### WEB LINKS

- [1] Wolfgang Ewald's ADS1115 library: [https://github.com/wollewald/ADS1115\\_WE](https://github.com/wollewald/ADS1115_WE)
- [2] LiquidCrystal library : <https://github.com/arduino-libraries/LiquidCrystal>
- [3] Download Link for This Project: <https://elektormagazine.com/230715-01>
- [4] G. Carrera, "Low Noise ADC Calibrator for Modern Microcontrollers," Elektor Circuit Special 2022: <https://elektormagazine.com/magazine/elektor-261/60679>
- [5] Ultra-Small, Low-Power, 16-Bit Analog-to-Digital Converter with Internal Reference: <https://ti.com/lit/ds/symlink/ads1114.pdf>



# FPGAs for Beginners

## The Path From MCU to FPGA Programming

By Theo Mulder (The Netherlands)

There are many low-cost microcontroller development boards available, but FPGAs remain expensive. For applications that need higher computational performance (e.g., AI) or higher speed I/O (e.g., high-speed camera, high-speed display, fast ADC or DAC), it is desirable to find small and affordable FPGA boards. Let's delve into the thrilling world of FPGAs for beginners!

When looking for low-cost FPGAs to select, it's helpful to understand the current business trends. The FPGA market was valued at USD 8.0 billion in 2022, expected to grow to USD 15.5 billion by 2027. The sector is competitive and has seen significant vendor consolidation. Today, the remaining brands are AMD/Xilinx, Intel/Altera, Lattice, Microchip, QuickLogic, Efinix, Flex Logix, GOWIN, Achronix, S2C, and Renesas.

Through acquisitions, the landscape never stops changing. As far back as 1999, Lattice acquired Vantis, then Acree in 2001, and Microsemi acquired Actel in 2010. In 2013, the major players were Xilinx, Altera, Actel, Vantis, Lattice, Lucent, QuickLogic, and Cypress. However, the ballet of mergers and acquisitions has never ceased; even the "Big Two," Xilinx and Altera, were themselves acquired by AMD and Intel, respectively.

By 2019, Xilinx led the market with a 52% share, followed by Altera at 35%, and smaller shares held by Microchip, Lattice, and others. Despite the dominance of

these larger companies, smaller FPGA vendors remain active, which is excellent for supporting different types of businesses and promoting innovation.

The automotive industry's growing use of FPGAs presents opportunities for vendors such as Lattice, as industry designers look for cost-effective alternatives to the pricier offerings from the biggest companies. Renesas is emerging as a significant competitor due to its established presence in the automotive sector.

The dynamics of the FPGA market, with its competitive shifts, acquisitions, and the entry of new players could benefit those seeking low-cost FPGA options for their projects.

### Affordable FPGA Boards for Beginners

As the different FPGA manufacturer toolchains are quite different from one another, and learning how to use a new software tool takes time, it makes sense to differentiate boards by FPGA manufacturers.

That way, the learning time will have been well invested if in the future you decide to upgrade to a higher-end model made by the same FPGA manufacturer.

The three main manufacturers to consider are Altera/Intel, Xilinx/AMD, and Lattice. The latter tends to offer less powerful, more affordable models than the former two.

Starting with boards powered by FPGAs from Lattice Semiconductor, the Olimex iCE40HX1K-EVB evaluation board, priced at €15, offers an extremely affordable entry point, though it requires an external programmer. This board features an iCE40HX1K with 1280 Logic Elements (LE). Interestingly, this FPGA has an internal, non-volatile configuration memory which can be programmed through the SPI interface of an Arduino board you may have, thanks to the sketch provided by Olimex. The official programmer from Lattice uses an FT232H from FTDI, and it is also possible to use a FT232H breakout board for this. One of the exciting features of this FPGA family is that it's compatible with the open source graphical tool Icestudio, which allows users to experiment with block-based graphical programming.

For those seeking other very compact, breadboardable and budget-friendly options, there are interesting products in the TinyFPGA family. The TinyFPGA BX comes in at €40, featuring a Lattice iCE40LP8K, also compatible with Icestudio. The AX1 and AX2 models are available at €13 and €17, respectively, providing smaller, cheaper alternatives albeit with fewer features. Sadly, these three tend to be often

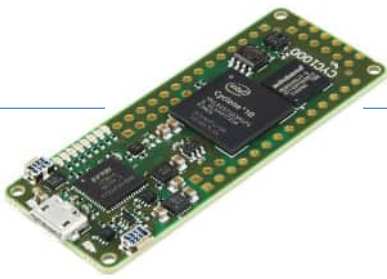


Figure 1: The CYC1000 board.

out of stock. From another manufacturer, the Upduino v3.1, priced at €30, offers an iCE40UP5K FPGA (5.3 kLE).

The Lattice iCEstick is a compact option with a USB thumb drive form factor, housing an iCE40HX1K FPGA and an onboard programmer at €48. It's the official tool from Lattice, so it works well with Lattice's software, and there are many tutorials using it.

Rounding off the Lattice options, let's also mention an open-source platform in an Arduino UNO form factor, the Alhambra II (iCE40HX4K, 3.52 kLE, €60) and also the Nandland.com Go Board (iCE40HX1K, €70) which helps to support the creator of the excellent tutorials at Nandland.com.

Shifting to the Altera (now owned by Intel), the MAX1000 board from Trenz Electronic provides a MAX10 FPGA, available in 8 or 16 kLE (€34 to €49), offering a breadboard-friendly design. The CYC1000 [1] (**Figure 1**), also from Trenz, priced at €40, ups the ante with a Cyclone 10 FPGA (25 kLE).

For more traditional, full-featured boards, there are options from Terasic. Some of their products are expensive, but others are very interesting for beginners, such as the DE0-nano which offers a Cyclone IV FPGA with 22 kLE and a slew of onboard features for €110. The DE10-Lite (€140) stands as a feature-rich board with a MAX10 FPGA (50 kLE), various I/O options, seven-segment displays, LEDs and switches, plus support for Arduino shields.

Last but not least, under the AMD (formerly Xilinx) umbrella, the Digilent Cmod S7 and Cmod A7-35T both offer breadboardable designs with Spartan 7 and Artix 7 FPGAs, 23.4 and 33.3 kLE, respectively, each priced at €90.

For a feature-rich learning experience with a bigger board, the Digilent Basys 3 brings to the table an Artix 7 FPGA with 33.3 kLE, extensive I/O options including VGA out and USB host, at a price of €155.

In this article, I will focus on Intel/Altera products. I recommend the CYC1000. In the next sections, we are going to experiment with Intel's Quartus Prime Lite, which is the free version of the official development tool for the Altera/Intel FPGAs.

## Installation

On the Intel Quartus Prime Design Software page [2], you will find download links for the latest version, for Windows or Linux. A good Internet connection is required, as the file is about 5.5 GB. The installer needs 15 GB of empty space. Some parts of Quartus Prime Lite use code from early versions of Quartus. As a consequence, spaces in filenames should be avoided, even if that sounds surprising by today's standards.

Only the Low Level Design tool is needed. There are additional packages that are outside the scope of this introduction, such as DSP Builder (for signal processing applications), High Level Synthesis Compiler (used with C++ as an input, for advanced applications) as well as the Nios II Embedded Design Suite (used to implement a Nios II soft processor within an FPGA).

## Creating a New Project

First, create a directory — for example, C:\Projects\IntelFpga\Elektor\BeginExample. Then open Quartus Prime Lite. You can see the main screen in **Figure 2**. Go to *File*, then select *New Project Wizard*. After you read the welcome screen, click *Next*. When prompted for the working directory, choose

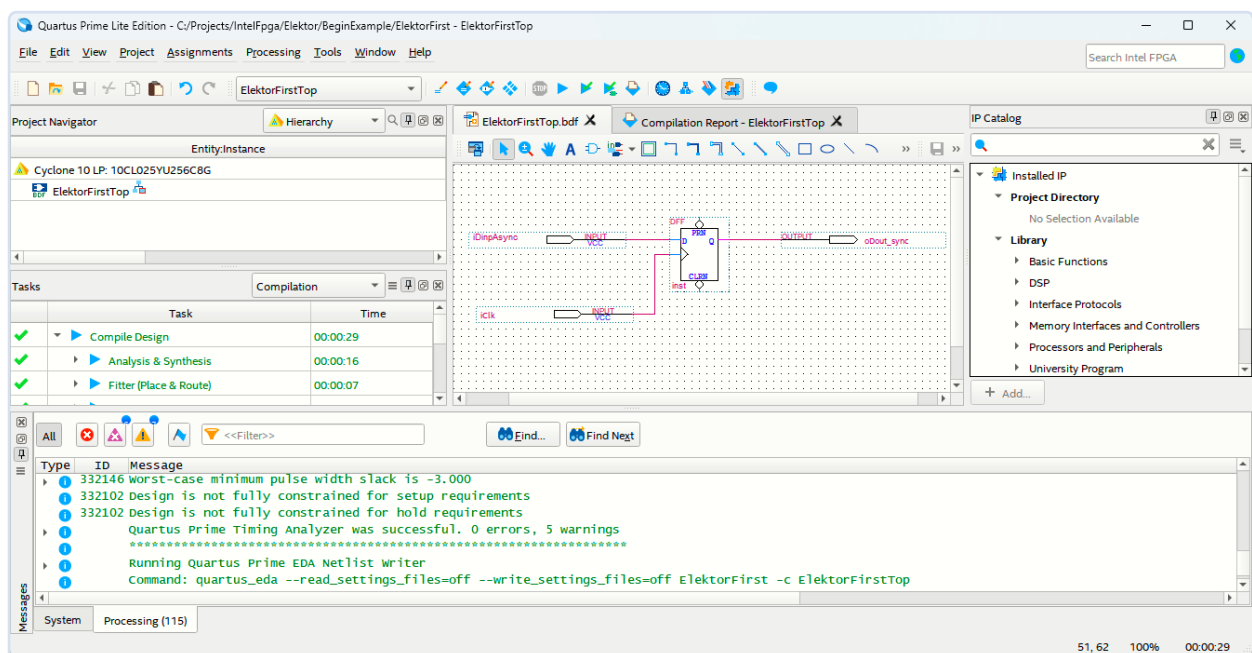


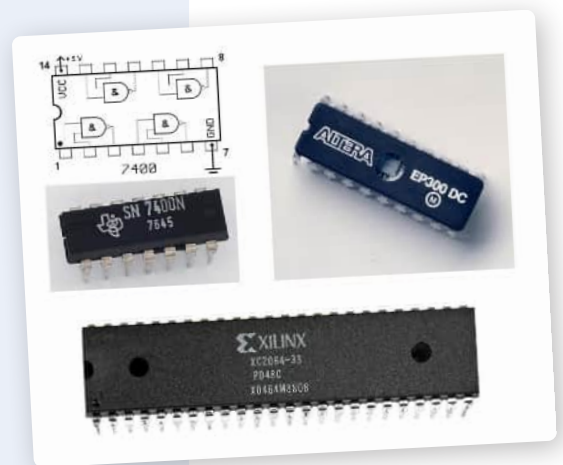
Figure 2: Quartus Lite Edition, with ElektorFirstTop.bdf.

## History of Logic: From TTL to FPGA

Before the advent of FPGAs, digital circuits primarily relied on TTL chips and programmable devices like FPLDs, PLAs, and PALs. The TTL SN7400 series operated up to 20 MHz. Altera introduced the EP300 40 years ago, and was followed by Xilinx's XC2064. The EP300 was built on a 5000 nm process with speeds up to about 10 MHz, whereas modern mid-range FPGAs use a 20 nm process and achieve clock speeds of at least 250 MHz.

The EP300 was a 20-pin reprogrammable device using software A+PLUS on IBM-PCs running DOS. Xilinx's XC2064 featured logic cells with 1200 gates each and used a low-power CMOS process, with design software that included Future-Net, VIEW-logic, and XACT-Design-Editor. The XC2064 was advertised with an internal toggle rate of 100 MHz, but typically achieved around 70 MHz.

Initially, FPGA software tools employed schematic capture, but this was later supplemented by text-based Hardware Description Languages (HDL) such as AHDL for Altera, and VHDL and Verilog for Xilinx. Today, both companies support VHDL and Verilog. Older versions of Altera's Quartus tool used AHDL with built-in simulation features, later requiring the use of ModelSim for simulation due to the lack of AHDL support. Today, all the manufacturers offer a free version of their tools.



the directory you created. A project name must be chosen, as well as a name for the top-level design entity, which the compiler will use as the root of the design. In this example, *ElektorFirst* and *ElektorFirstTop* were used. In the next step, choose *Empty Project*. Then you'll be taken to the *Add files* step. Since we don't have any files, simply click *Next* again.

You'll then arrive at a page where you can select a Device, this can be done via either one of the two tabs: *Device* and *Board*. In the latter, several development boards are listed in the MAX10 and Cyclone V families and additional boards can be installed later. In the *Device* tab you can choose the exact family and part number of the FPGA you are using, such as the 10LC025YU256C8G in the Cyclone 10 LP family if you have the CYC1000 board. The Cyclone 10 family consists of low-cost and low-power devices that were derived from the Cyclone V.

If you don't have any development board, you can still select that part to continue with the tutorial and experiment with the compilation process. Click *Next*, *Next* again, and *Finish*.

## Project Directory Structure

Take a moment to check out your working directory. You'll notice there's a db directory which Quartus will use later. The file named *ElektorFirst.qpf* is the Quartus Project File (QPF). When you want to return to your work later, you can simply double-click this file to reopen the Quartus software. There's also a file called *ElektorFirst.qsf*, which is the Quartus Settings File (QSF). Both of these are text files, so you can take a look at them if you're curious — just right-click and choose *Open with* to find a text editor.

## Opening the Project

In the upper-left corner, you'll see the *Project Navigator* with *Hierarchy* selected. Look for *ElektorFirstTop*, which we named earlier. When you click on it, a pop-up will appear stating "Can't find design entity *ElektorFirstTop*" because we haven't created a file yet. Click OK, then go to the *File* menu. Here, you can see various file types that you can create. Choose *Block Diagram/Schematic File* and click OK.

Now you're presented with a white canvas in the center pane, labeled *Block1.bdf* at the top. Go to *File*, select *Save As*, and it will suggest saving the file as *ElektorFirstTop*.

*bdf*. That's perfect — click *Save*. Now, when you click on *ElektorFirstTop* under *Project Navigator* and *Hierarchy*, you'll land on this empty schematic sheet.

## Adding Parts to Our First Schematic

In order that all our readers, even those unfamiliar with the Verilog and VHDL languages, can follow this tutorial, we are going to use schematic programming. Click on the *ElektorFirstTop.bdf* tab to bring up the sheet, right-click anywhere on the sheet, and choose *Insert Symbol*. This will bring up the *Libraries* menu. Click on the small triangle next to *Libraries* to expand the list, where you'll see categories like *Megafunctions*, *Other*, *Primitives*, and more. There's a lot to explore here, so feel free to browse around.

Scroll to the bottom of the *Libraries* window and in the *Name* field, type in "Input." An input symbol will appear; click OK. Place this symbol somewhere on the page, ideally toward the upper left. Then add a second input by right-clicking, choosing *Insert Symbol*, and since "Input" should still be in the *Name* field, just click OK again and place this new input beneath the first one on your sheet.



Next, we'll add a D-FlipFlop. Right-click, select *Insert Symbol*, and in the *Name* field, type "dff". Choose the *dff* from the options provided and click OK, then place it on the sheet. To complete this step, right-click, select *Insert Symbol*, enter "Output" in the *Name* field, and click OK. Place this output on the sheet as well.

## Renaming Pins

We'll need to give meaningful names to pins. It's possible to add prefixes or suffixes to give details as needed, such as *i* for input, *o* for output, *sync* or *async* for synchronous or asynchronous, respectively, etc. Hence, we'll rename the input *pin\_name1* to *iDinpAsync*. To do this, double-click on *pin\_name1* and enter the new name. Also rename *pin\_name2* to *iClk* and the output *pin\_name3* to *oDout\_sync*. Then click *File* and *Save*.

## Adding Wires

Click the *iDinpAsync* pin. This action will automatically activate the *Orthogonal Node Tool* used to draw wires. Begin drawing from this pin and connect it to the D input of the D Flip-Flop (DFF). Then, proceed to connect *iClk* to the clock input of the DFF, which is indicated by a triangle within the symbol. Finally, create a connection from the Q output of the DFF to the *oDout\_sync* output symbol. Then save again.

## Compiling

To compile the design, simply click on the blue triangle icon in the icon bar. Keep an eye on the *Task* pane on the left to follow the progress and watch for messages scrolling at the bottom of the screen. It's quite an interesting process.

After the compilation, the center pane will display the *Flow Summary* pane. This summary indicates that only one out of the 24,624 total logic elements was used. It's rare to use 100% of a device's logic elements; in bigger, well-optimized designs, there may be around 80% of the LEs used. Beyond that point, you may encounter routing difficulties or the device might not meet the timing requirements due to congestion.

The summary also shows that one register was used, and there are three out of 151

possible pins in use, equating to 2% of the total pins. Keep in mind, however, that some pins may be reserved and not available for general use.

Check out your project directory again. You'll see some new subfolders there. Take a peek inside the *db* folder to get a sense of the extensive background work that's been done. Inside the *output\_files* folder is the *ElektorFirstTop.sof* file that you'll use to program your FPGA.

Lastly, examine the *ElektorFirstTop.pin* file by opening it with a text editor. This file shows the pin connections that the compiler has

set up. Since we didn't specify any particular constraints, Quartus selected pins at random. Typically, as a designer, you would define your pin assignments. Look for the *iDinpAsync*, *iClk*, and *oDout\_sync* pins. In my situation, *iDinpAsync* was connected to pin T4, operates at a 2.5 V level, and is found in bank 3 of the FPGA.

## Programming the FPGA

Of course, we are curious to see if this works in the real FPGA, so let's give it a try! First, connect your board to an available USB port on your computer and follow the steps to upload the configuration file to the FPGA SRAM. Alternatively, you can upload it to

### Selecting the Ideal FPGA for Your Project

When selecting FPGAs and boards, there are several parameters to consider. One is the number of Logic Elements, which will give a rough idea of the size of the FPGA. Also, one must consider the need for low or high-speed I/Os. High-end FPGAs with fast memory and transceivers are expensive, as are the associated software tools. Of course, the classifications of high, mid, and low-end are subjective and can vary from one vendor to another. For the leading two FPGA vendors, what's considered mid-range performance might be labeled as high-end by Lattice.

When it comes to I/Os, transceivers are often mentioned. They are serial connections that offer high transfer rates (e.g. 4 to 32 Gbps), but greatly affect the cost. They are not necessary for all applications. Instead of high-speed PCIe or COM Express connections, a more cost-effective and convenient option for many is to use USB, possibly with an FTDI device for interfacing.

For high-speed ADC or DAC connections to an FPGA, it is possible to use the JESD204B interface, but that needs transceivers on both the FPGA and the ADC/DAC side, which can get expensive. Attention to signal integrity is needed when routing the board, but the layout is somewhat simplified by the smaller number of tracks. Another option when large throughputs are needed is to use several slower Low Voltage Differential Signaling (LVDS) pairs in parallel. In any case, it's useful to make a fairly precise calculation of your throughput requirements to help in choosing a suitable component.

I tend to consider the interfaces as low-speed when they operate under 1000 Mbps, and high-speed above that. The cost of FPGAs increases with the number of high-speed interfaces. Yet, there are also many times when even moderate-speed LVDS isn't required: general purpose I/O pins may be enough.

On FPGAs, the general purpose I/O pins are the slowest of the available interfaces, but they still have faster toggle rates than general-purpose I/O pins on microcontrollers. They are usually compatible with either 3.3 V, 2.5 V or 1.8 V logic. If you were thinking of using a microcontroller for your next project, but with faster I/Os than what typical microcontrollers can produce, FPGAs can be a compact solution, as it is possible to implement a processor (RISC-V or other) in the FPGA fabric.

## VHDL or Verilog?

The decision on whether a beginner should learn Verilog or VHDL in their FPGA learning journey is influenced by several factors. For those aiming to work in the defense or avionics sectors, VHDL is the predominant language, especially in European countries like Germany and France. On the other hand, in the United States and in many sectors outside of defense, Verilog is more commonly used.

However, anyone serious about a career in digital design will likely need to be able to use both languages. It's important to adapt to your learning context; if it's for academic purposes, align with the language taught in your courses, whereas for professional settings, it's best to use what your colleagues are using.

From a technical perspective, Verilog tends to be more compact and, resembling C in some aspects, which might not always be advantageous as it could lead to a software-oriented mindset rather than the required focus on hardware. VHDL's verbose and strongly typed nature can help in avoiding certain errors that Verilog's weak typing might miss. This distinction might make VHDL a better starting point for those who prefer a language that enforces a hardware-centric thought process, crucial for FPGA design.

flash, which allows the FPGA to retain the design even after power cycles.

To do this, go to *Tools* and then *Programmer*, or simply double-click on *Program Device* in the task list. Click on *Hardware Setup*, and select *USB-Blaster [USB-o]*. Then, click *Add File*, and choose the SRAM object file *ElektorFirstTop.sof* that was generated in the *output\_files* directory after compilation. Hit *Start*. Remember to save the configuration of the programmer, for example as *example1.cdf*, so it opens with these settings the next time you use it. After a few seconds, the transfer is complete! We've successfully uploaded the design to the FPGA, where it will remain until the power is disconnected. Congratulations, you now have a very fancy D flip-flop, you can try it out by hooking wires and switches to the appropriate pins of the CYC1000 board!

## Hierarchical Design

It can be very useful to design some part of a system once, and then be able to use it again in various places of the system. This can be done by wrapping a portion of the design in what's called a symbol. As an example, let's create a symbol for our circuit.

First, create a new block diagram/schematic file as in the above "Opening the Project"

section: *File, New*, etc. Name it *Synchronizer.bdf*. Then, on the *ElektorFirstTop.bdf* window, click and hold the mouse to select all the components in a rectangle. Then copy and paste the contents in the empty *Synchronizer.bdf* and save. Then with the *Synchronizer.bdf* window still open, go to *File*, select *Create/Update*, select *Create Symbol Files for Current File* and click *Save* (with the suggested name *Synchronizer.bsf*). On the popup, click *OK*.

Now, a Symbol for a DFF-Synchronizer has been created with the name *Synchronizer.bsf*. You can close the *Synchronizer* window. In *ElektorFirstTop.bdf*, you can experiment and use this newly created symbol. Remove the D flip-flop, move the inputs and outputs to the sides to make room, and right click in the empty space to insert the symbol as we did in the "Adding parts to our first schematic" section. The newly created *Synchronizer* symbol will be available in the *Insert Symbol* menu. Now you can connect the symbol to its appropriate inputs and outputs and *Save* again. This hierarchical design can now be compiled once more.

## Using Hardware Description Languages

Most people prefer using an HDL, such as Verilog or VHDL, over block schematic entry. There are many resources available

as support for these languages. Moreover, schematic designs become notoriously challenging to maintain and update as they grow in size.

Let's convert the *Synchronizer.bdf* into a Verilog file. Click on *Synchronizer.bdf*, navigate to *File*, then to *Create/Update*, and choose *Create HDL Design File from Current File*. Select *Verilog HDL* and hit *OK*. Now, a *Synchronizer.v* file should appear in your design directory. You can open this file with any text editor or with Quartus itself. Go to *File*, select *Open*, find *Synchronizer.v*, and open it. Voilà, you now have the Verilog description of your synchronizer circuit right in front of you. That can be helpful for learning purposes! Of course, it is also possible to create a VHDL file instead.

You'll have to create a new symbol for this new *Synchronizer.v* file. Follow the same steps outlined in the "Hierarchical Design" section and use the *Create Symbol for Current File* command to accomplish this.

## Experiment on Your Own

Feel free to explore and experiment on your own now. In *ElektorFirstTop.bdf*, right-click on the schematic area, choose *Insert Symbol*, and click the small triangle next to the bottom item. Dive into the primitives, then into the logic, and you'll discover an array of basic logic components like AND, OR, XOR, etc. Go ahead and craft your own circuit!

Moreover, to locate the 7400-series logic circuits in the library, right-click on the schematic area in *ElektorFirstTop.bdf*, select *Insert Symbol*, and type "7400" in the *Name* field. A wide selection of 7400 series devices can be found. You can browse through them and preview the symbols in the right pane. There are plenty of intriguing logic circuits that utilize 7400 series logic available online, which you can emulate on an FPGA. It may not be the most efficient method, but it can certainly be an entertaining and educational experience! You can see an example of a design using 74 series logic in **Figure 3**.

You can also take a look at the overall design inside the FPGA. In the top bar, choose

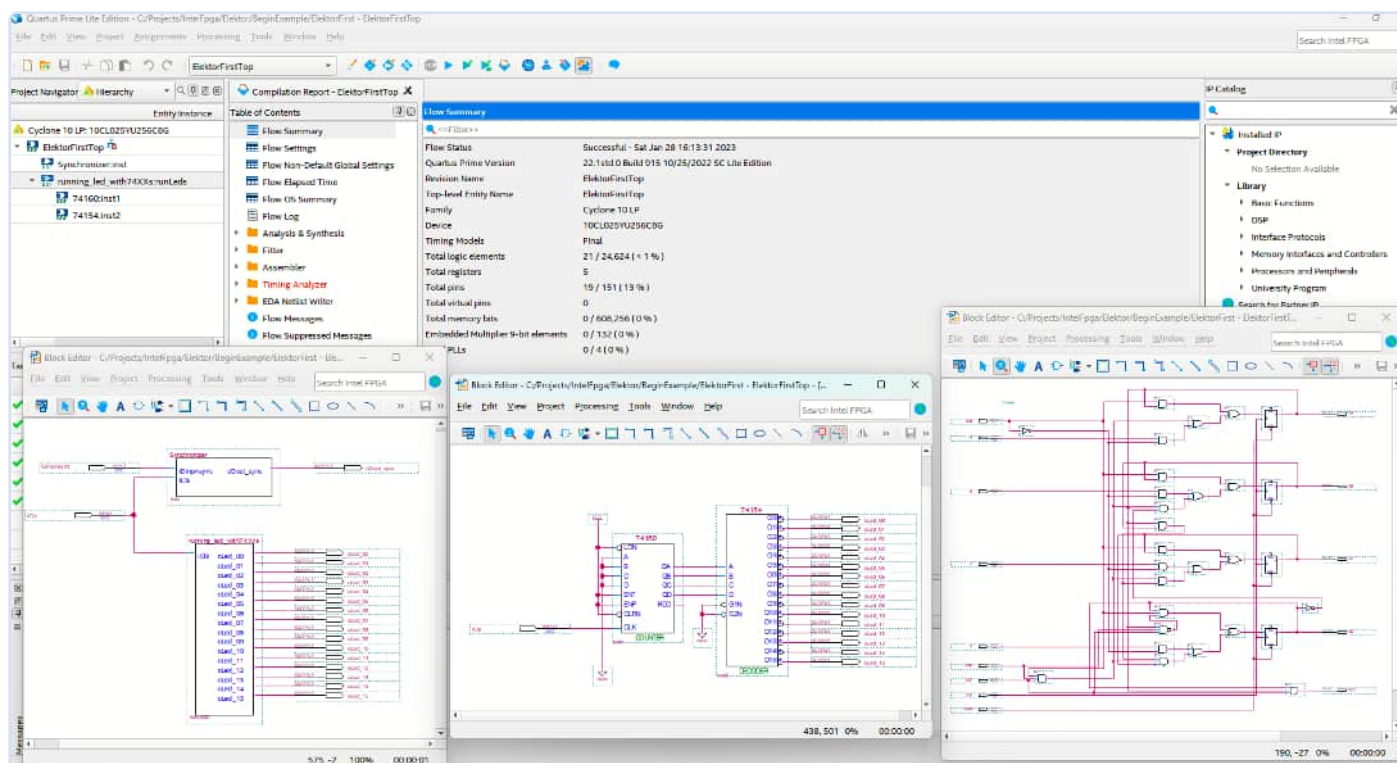


Figure 3: Quartus Lite Edition, with the running\_led\_with74XXs example.

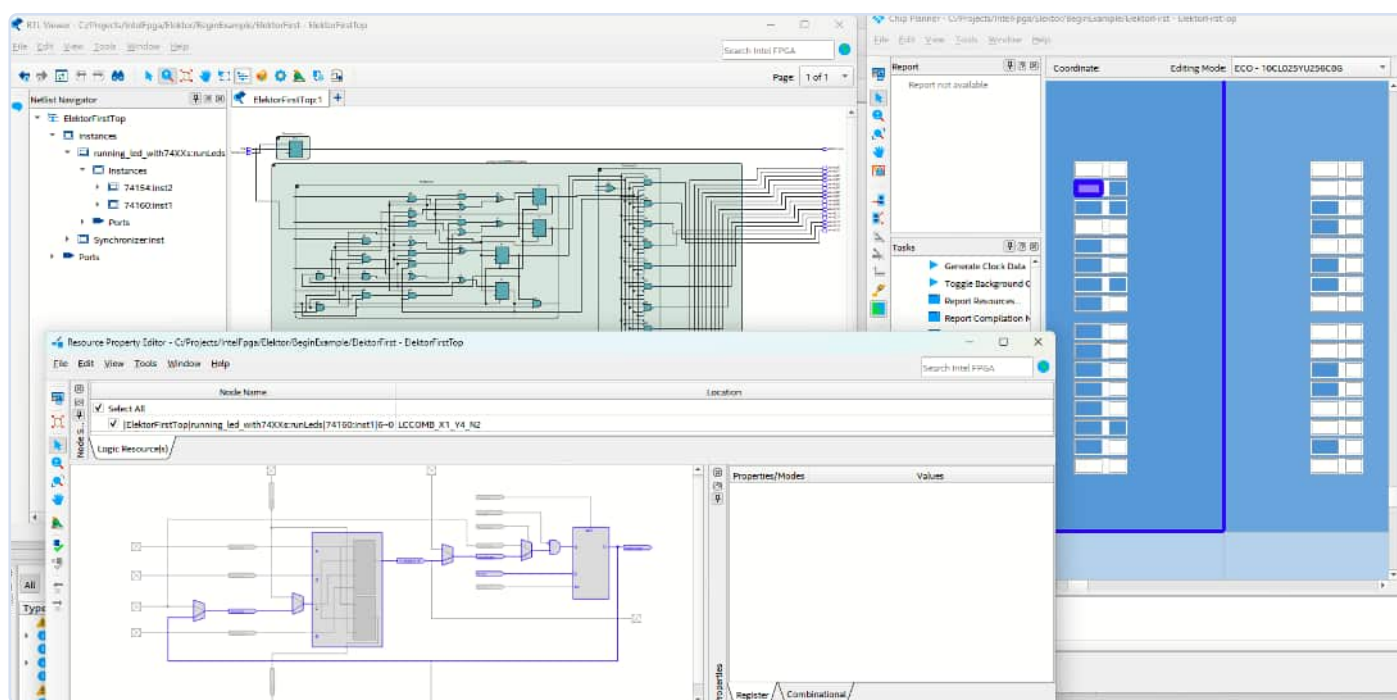


Figure 4: Quartus Lite Edition, showing some of the lower level implementations of running\_led\_with74XXs example.



Tools, Netlist viewers, RTL viewer. You can see the result in **Figure 4**. The schematics are visible in the top left. To examine the placed elements within the FPGA, select Tools, Chip planner. Here, you can zoom on the area containing the logic cells of the design. Additionally, in the picture, on the top right, a small purple cell is highlighted. By selecting a logic cell and clicking on it, you can view its contents, as shown in the bottom left.


## Going Further

We've just scratched the surface of FPGA design by showcasing the free Quartus Prime Lite. It offers a complex array of features, many of which we couldn't cover in this introductory piece, such as simulation, the Signal Tap logic analyzer, Platform Designer, and more. There's no shortage of accessible FPGA boards for beginners to start experimenting with. For enthusiasts leaning towards open-source, the Icestorm project and Icestudio graphical IDE are noteworthy gateways into this field.

Moreover, there is a plethora of online resources for future learning. Websites

such as fpga4fun [3], NandLand [4], and VHDLwhiz [5] are excellent starting points. Joel's list of budget-friendly FPGA boards [6] can also help in selecting the right hardware.

And, when it comes to books, the digital version of *Free Range VHDL* by Bryan Mealy and Fabrizio Tappero is offered completely free of charge by the authors, and *Getting Started with FPGA*, from Russel Merrick (the author of the NandLand tutorials) is also a comprehensive addition to your bookshelf.

The FPGA world is at your fingertips, with more learning tools and communities available than ever before. Now is an exciting time to start your journey in digital design! 

230067-01

## Questions or Comments?

If you have technical questions or comments about this article, feel free to contact the author by email at [tf.mulder@outlook.com](mailto:tf.mulder@outlook.com) or the Elektor editorial team at [editor@elektor.com](mailto:editor@elektor.com).

## About the Author

Theo Mulder is an electronics engineer. He is very familiar with signal processing and has worked with analog and digital electronics, especially in medical electronics for ultrasound equipment. He is used to C++, DSPs, and FPGAs. He transforms research concepts into working electronics. Some of his own concepts were patented. He is a systems thinker, at electronics board level, to find a good balance between analog/digital hardware and software, to realize well-functioning systems with good price/performance ratios.



## Related Products

- > **M. Dalrymple, *Microprocessor Design Using Verilog HDL* (E-book, Elektor)**  
[www.elektor.com/18518](http://www.elektor.com/18518)
- > **Alchitry Au FPGA Development Board (Xilinx Artix 7)**  
[www.elektor.com/19641](http://www.elektor.com/19641)



## WEB LINKS

- [1] Trenz Electronic - CYC1000 board: <https://tinyurl.com/trenzcy1000>
- [2] Intel Quartus Software Downloads: <https://tinyurl.com/downloadquartus>
- [3] fpga4fun: <http://fpga4fun.com>
- [4] NandLand: <https://nandland.com>
- [5] VHDLwhiz: <https://vhdlwhiz.com>
- [6] Joel's list of FPGA boards: <https://joelw.id.au/FPGA/CheapFPGADevelopmentBoards>



UPDATE:



# STM32 Wireless Innovation Design Contest 2024

By The Elektor Content Team

With a share of €5,000 in cash prizes up for grabs in the STM32 Wireless Innovation Design Contest 2024, innovators from around the globe have been working hard over the past few months with STM32 solutions to develop a variety of creative wireless applications. Public nominees for winners will be announced in March 2024. The final announcement will take place at embedded world 2024.

The STM32 Wireless Innovation Design Contest offers engineers and makers a unique opportunity to showcase their design skills by creating wireless applications using the powerful development and evaluation boards from STMicroelectronics. Whether you're passionate about IoT, robotics, gaming, home automation, or AI, the possibilities are endless. €5,000 in cash prizes are up for grabs!

## Winners: Stay Tuned!

The submission deadline for the contest was March 1st, 2024. At the time of this magazine's publication, the jury is assessing the project submissions. The winners will be announced live at the embedded world 2024 conference in Nuremberg, Germany at 17:00 CEST on Wednesday, April 10, as well as online at [elektormagazine.com/st-contest](https://elektormagazine.com/st-contest). If you plan to attend embedded world 2024, be sure to stop by both the Elektor and STMicroelectronics booths ([embedded-world.de](https://embedded-world.de)).

## Judging

A panel of independent judges will select the top three winners based on the following criteria:

- > **Creativity and Innovation:** The uniqueness and originality of the wireless application design.
- > **Technical Excellence:** The technical proficiency and skill demonstrated in utilizing the chosen development board.
- > **Functionality and Practicality:** The effectiveness and practicality of the wireless application in solving a real-world problem or enhancing user experience.
- > **Aesthetics and User Experience:** The visual appeal, user interface design, and overall user experience of the application.
- > **Documentation and Presentation:** The clarity, completeness, and quality of the project documentation and presentation.

We wish all the participants good luck!

## STM32 Technology

To enter the contest, participants must create a project using one of the featured boards: NUCLEO-WBA52CG, STM32WB5MM-DK, or NUCLEO-WL55JC. The idea is to use a board's capabilities to design and develop wireless applications in any field. Participants can explore standardized protocols such as LoRaWAN, Sigfox, and Bluetooth Low Energy (BLE), or they can create their own proprietary protocols.

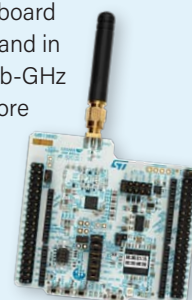
- > The **NUCLEO-WBA52CG** is a Bluetooth Low Energy wireless and ultra-low-power board embedding a powerful and ultra-low-power radio compliant with the Bluetooth Low Energy SIG specification v5.3. The ARDUINO Uno V3 connectivity support and the ST morpho headers allow the easy expansion of the functionality of the STM32 Nucleo open development platform with a wide choice of specialized shields.



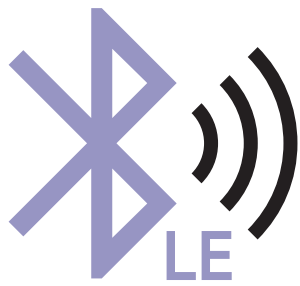
- > The **STM32WB5MM-DK Discovery Kit** is a demonstration and development platform for the STMicroelectronics STM32W5MMG module. This dual-core 32-bit Arm Cortex-M4/M0+ device integrates an ultra-low-power radio compliant with Bluetooth Low Energy (BLE) 5.2, 802.15.4 with Zigbee, Thread, and proprietary protocols.



- > The **Nucleo-WL55JC** board is an evaluation board for the STM32WL-series of microcontrollers, and in particular the STM32WL55. This so-called sub-GHz wireless microcontroller is based on a dual-core 32-bit Arm Cortex-M4/M0+ with a clock frequency of 48 MHz. It features ultra-low power consumption, an RF transceiver with a 150 MHz to 960 MHz frequency range, 256 KB of Flash memory and 64 KB of SRAM.



Looking for more information? Visit the STM32 Wireless Innovation Design Contest website — [elektormagazine.com/st-contest](https://elektormagazine.com/st-contest) — for details about the winning projects and much more. [▶](#)



# Bluetooth LE with MAUI

Control Apps for Android & Co.

By Tam Hanna (Hungary)

A smartphone app is ideal for controlling your own electronics. Communication can take place via Bluetooth LE to save energy.

However, developing and maintaining software for Android and iOS requires some effort. This is where the MAUI framework already presented in Elektor comes into play, which can be used to program apps independently of the platform.

In this article, we will show you how to access the BLE interface of a smartphone.

Especially those with extensive knowledge of .NET development will find using MAUI a convenient way to develop software for Android and iOS — a general introduction can be found in another article [1]. In this article, we want to “deepen” the whole thing by programming a Bluetooth interface on Android. It should be noted that the library used here also works on iOS; however, the necessary adaptations would exceed the scope of this article, which is why they are not covered here.

## Setting Up the Working Environment

This article is based on a practical consulting project by the author, who works as a freelance developer. An ESP32 serves as the “other station,” which executes a control program based on the code example [2]. Visual Studio 2022 is used to develop the smartphone application; the actual sample project is based on the .NET MAUI app template.

Microsoft decided a long time ago not to “directly” offer a Bluetooth API in MAUI (or Xamarin, on which MAUI is based). This was a wise



*The BopSync ash tray, developed by Icy Beats LLC, uses a Bluetooth LE interface for control.*

decision, as Microsoft must have had first-hand experience of the development process for Bluetooth APIs on Android, which was anything but “smooth.” However, since an interface exists that allows Xamarin to access native elements of the host operating system, various NuGet packages can be used with libraries that attempt to make up for this shortcoming. In the following steps, we want to rely on the *Plugin.Ble* library provided under [3]. The first action is to open the NuGet console, where you download the missing library.

## Adaptation of the Manifest File

As a matter of principle, cross-platform environments can only partially isolate the developer from the underlying platform. In the case of the library used here, this is reflected in the fact that adaptations to the respective manifest files are required for both Android and iOS (which is not discussed further here). These tell the operating systems which capabilities the application intends to use.

In the case of the Visual Studio version 17.6.0 Preview used here, clicking on *AndroidManifest.xml* normally opens a graphical editor, which is not yet fully developed though, which is why right-clicking on the file and opening the manifest file in a text editor is the more convenient approach.

In the next step, the author replaced or added some structures to the existing declarations (**Listing 1**).

Due to the "dynamic permission system" introduced in Android 6.0, it is no longer sufficient to modify the manifest file at this point. Instead, the application must ask the user for permission to access the sensitive features at runtime.

In order to avoid a "cat and mouse fight" between the MAUI runtime and Google's maintenance of new permissions, Microsoft implements a "general" interface for the procurement of permissions. In principle, this is a class structure that holds a list of permission constants.

To use a new permission attribute, the developer only has to wrap the constant in an instance of `Permissions.BasePlatformPermission`. The rest of the interaction with the user interface, which follows a standardized procedure, is handled by the runtime contained in MAUI.

Our next task is therefore to create a new class of the type `Permissions.BasePlatformPermission`, which holds the various required permissions (**Listing 2**).



### Listing 1: Declarations.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
    <application android:allowBackup="true" android:icon="@mipmap/appicon" android:roundIcon
        = "@mipmap/appicon_round" android:supportsRtl="true"></application>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses-permission android:name="android.permission.BLUETOOTH_SCAN" />
    <uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADVERTISE" />
    <uses-feature android:name="android.hardware.bluetooth_le" android:required="true"/>
</manifest>
```



### Listing 2: Creating a New Class of the Type `Permissions.BasePlatformPermission`.

```
public class BluetoothLEPermissions : Permissions.BasePlatformPermission
{
    public override (string androidPermission, bool isRuntime)[] RequiredPermissions
    {
        get
        {
            return new List<(string androidPermission, bool isRuntime)>
            {
                (Manifest.Permission.Bluetooth, true),
                (Manifest.Permission.BluetoothAdmin, true),
                (Manifest.Permission.BluetoothScan, true),
                (Manifest.Permission.BluetoothConnect, true),
                (Manifest.Permission.AccessFineLocation, true),
                (Manifest.Permission.AccessCoarseLocation, true),
                //(Manifest.Permission.AccessBackgroundLocation, true),
            }.ToArray();
        }
    }
}
```

An important note: This permission selection works with Android 13. Older versions and the Kindle Fire require a different permission complement.

## Scanning for BLE Devices

We can then move on to the actual search for accessible Bluetooth devices. In the following steps, the author assumes that the reader is generally familiar with the use of Bluetooth LE. A brief introduction can be found at [4].

To list all elements in the transmitter's environment, the author will use a `ListView` in the following steps — open the *MainPage.xaml* file and insert the following *ListView* at a convenient location in the layout:

```
<ListView x:Name="deviceList">
  <ListView.ItemTemplate>
    <DataTemplate>
      <ViewCell>
        <StackLayout Margin="20,0,0,0"
          Orientation="Horizontal"
          HorizontalOptions="FillAndExpand">
          <Label Text="{Binding}"
            VerticalOptions="Start"
            TextColor="White"/>
        </StackLayout>
      </DataTemplate>
    </ListView.ItemTemplate>
  </ListView>
```

Developers who have grown up with Visual Basic 6 & Co. have to rethink when working with MAUI list boxes. Such a MAUI list box does not only consist of the collection of elements to be displayed — additional information known as *ItemTemplate* is required for the actual display. This is a small piece of XAML markup that is used by the GUI stack for each row of data to be displayed in the `ListView`. Our instance is simple in that it actually just puts a label on the screen. The string `Text=""` establishes a binding relationship to allow the XAML parser to directly obtain the elements to be displayed in the item.

The XAML markup file also requires an ordinary button that the user will use to initiate the scanning process.

In the next step, we can return to the Code Behind, where we create some support classes according to the following scheme:

```
public partial class MainPage : ContentPage
{
    int count = 0;
    IBluetoothLE myBLE = CrossBluetoothLE.Current;
    IAdapter myAdapter =
        CrossBluetoothLE.Current.Adapter;
    public ObservableCollection<String> BTLEDevices;
```

In addition to the variables required by the library, an *ObservableCollection* is also necessary here. It serves the data binding engine as a "data source" for the population of the `ListView`. It

should be noted that the decision to use an *ObservableCollection* was not made randomly; it is a *Collection* class that is capable of sending notifications when changes are made to the database it manages.

To complete the data binding relationship, some housekeeping is then required in the constructor of the main page:

```
public MainPage() {
    InitializeComponent();
    BTLEDevices = new ObservableCollection<String>();
    deviceList.ItemsSource = BTLEDevices;
}
```

The next step is the Code Behind method, which takes care of clicking the scan button. Android implements a permission cache, which is why in the first step, we check whether our application already has the necessary authorizations for interacting with the Bluetooth transmitter. If this is the case, we call the *goScan* method, which takes care of the actual scan run.

```
private async void OnScanClicked
(object sender, EventArgs e) {
    PermissionStatus status = await
        Permissions.CheckStatusAsync
        <BopSyncNetPOC.Platforms.
            Android.BluetoothLEPermissions>();
    if (status == PermissionStatus.Granted){
        goScan();
    }
```

If the return value of *CheckStatusAsync* is not affirmative, we ask the user for permission:

```
    else {
        await DisplayAlert("Permission required",
            "Google requires the declaration
            of this permission to perform a BTLE scan.
            Please grant it!", "OK");
        status = await
            Permissions.RequestAsync
            <BopSyncNetPOC.Platforms.
                Android.BluetoothLEPermissions>();
        if (status == PermissionStatus.Granted) {
            goScan();
        }
        else {
            await DisplayAlert("Alert",
                "Permission denied.
                Please reinstall application!", "OK");
        }
    }
}
```

The extensive permission description texts used here are necessary to persuade the user to agree. Practical experience has shown that



“technically challenged” users in particular usually react extremely negatively to unexpected pop-up permission requests due to paranoia fueled by the media. In addition, modern Android versions sometimes save refusals “permanently” — if the user clicks *No* once, in the worst case, they will have to uninstall and reinstall the application before they can grant the permission again.

Be that as it may, the `goScan` method comes up next:

```
private void goScan() {
    if (myBLE.State == BluetoothState.On) {
        myAdapter.ScanMode = ScanMode.LowLatency;
        myAdapter.DeviceDiscovered += FoundDevice;
        myAdapter.ScanTimeout = 12000;
        //MUST be called last or ignores settings
        myAdapter.StartScanningForDevicesAsync();
    }
}
```

If the Bluetooth transmitter is already active, we parameterize the adapter object and command a scan run by calling `StartScanningForDevicesAsync`. It is important to note that all settings required for the scan run must be written to the class before the method is called — subsequent changes are ignored.

If the Bluetooth transmitter is not active, we show the user a window instead that prompts them to switch on the transmitter:

```
else {
    DisplayAlert("Alert",
        "Please switch Bluetooth on!", "OK");
}
```

The delegate takes care of the “localization” of devices. At the moment, we want to focus on the population of the list:

```
private void FoundDevice(object sender,
    DeviceEventArgs e){
    try
    {
        BTLEDevices.Add(e.Device.Name.ToString());
    }
    catch(Exception ex)
    {
        var aD = e.Device.NativeDevice;
        //Dont muck around with the GUID in ID
        PropertyInfo aProp =
            aD.GetType().GetProperty("Address");
        BTLEDevices.Add("N/A " + aProp.GetValue(aD, null));
    }
}
```

The majority of Bluetooth LE devices do not have a name. The routine shown here uses a try-catch block to write a “more primitive” default string to the ListView in this case.

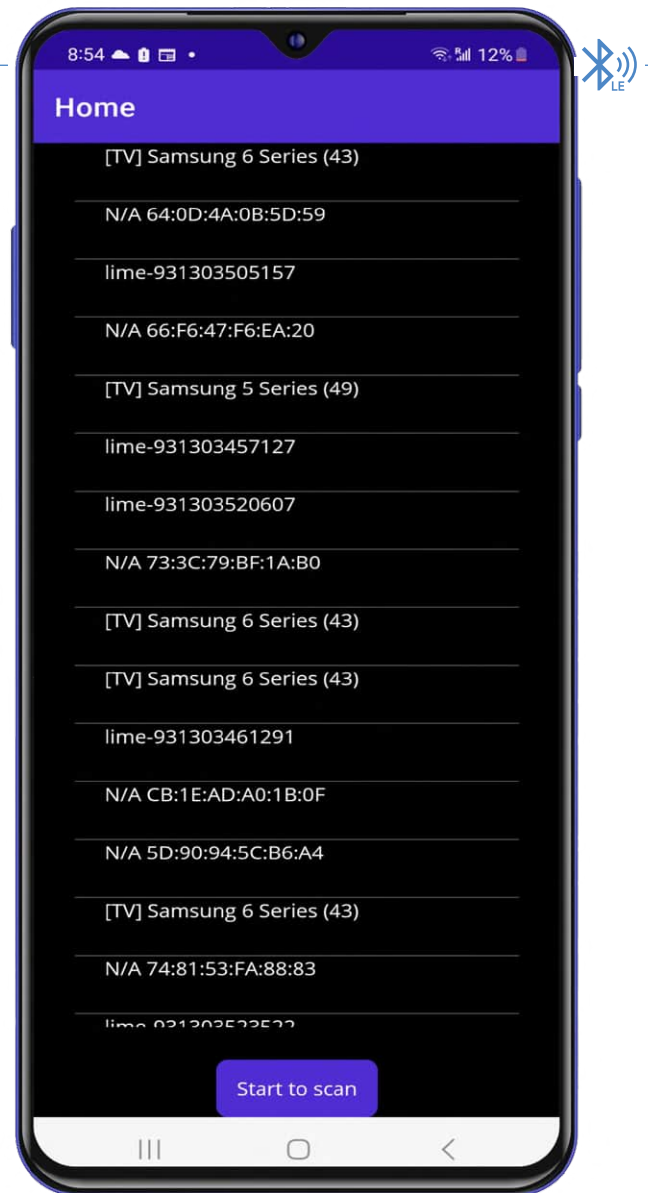


Figure 1: Bluetooth LE detection works.

At this point, the program is ready for a test run. **Figure 1** shows what a scan run looks like.

It should be noted that the direct placement of native elements in the general part of the Solution leads to some funny things — specifically, errors occur that point to the non-existence of the `Platforms.Android` namespace. This is because some operations in Visual Studio attempt to compile not only the Android variant, but also the various other target platform projects created in the MAUI project skeleton. To work around this problem, it is sufficient to place a preprocessor protection around the platform-specific code:

```
private async void OnScanClicked
    (object sender, EventArgs e) {
    #if ANDROID
        PermissionStatus status . . .
    #endif
}
```

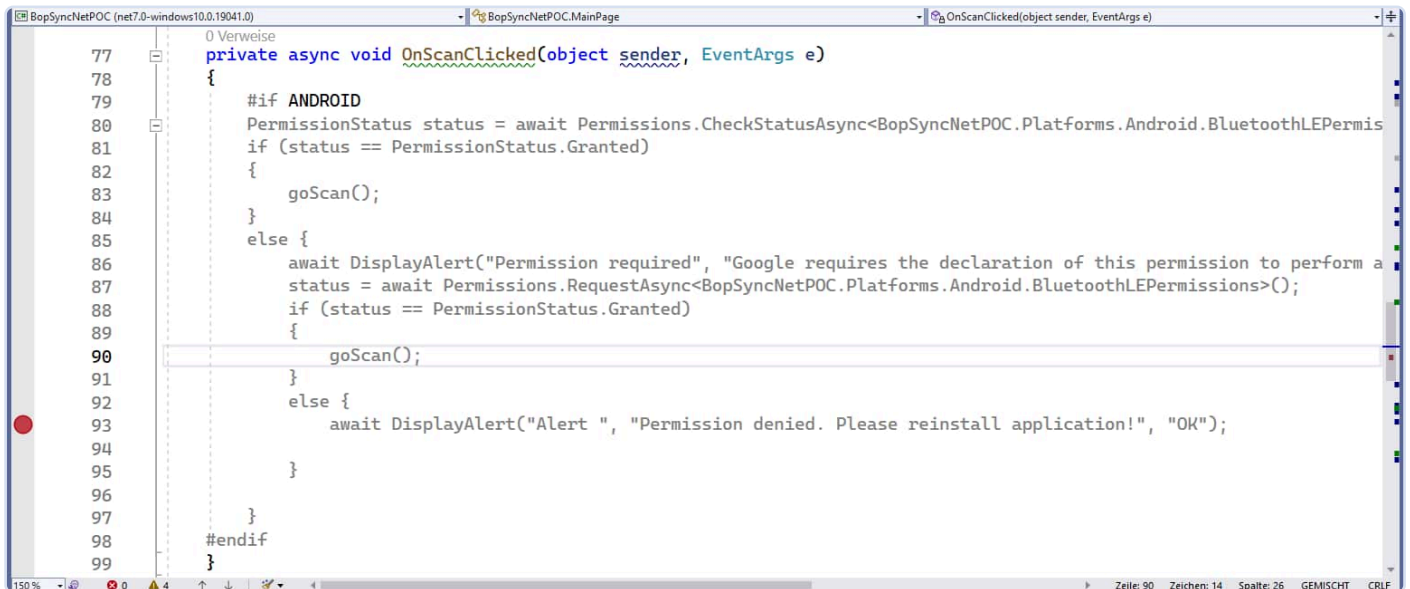


Figure 2: The daily battle: Man against Visual Studio.

It is annoying that Visual Studio has a hard time with such elements and disables syntax completion as shown in **Figure 2**.

## Identification and Connection Establishment

Now we are ready to take control of the peripheral device. The author extended the main page with an additional list in which he accommodates the individual Bluetooth device instances — they represent the remote stations found by the Bluetooth LE scan.

In the case of the author's application, only one device needs to be addressed at a time. For this reason, it makes sense to persist the instance in the `Application` class. Open the `App.xaml.cs` file and insert a global member:

```
public partial class App : Application
{
    public Plugin.BLE.Abstractions.
        Contracts.IService myBTLEService;
```

The complete qualification is reasonable because there is often more than one class with the name `IService` in the .NET world. If you declare "completely," you are on the safe side.

The actual connection setup then begins by extracting the `IDevice` class from the device list mentioned above:

```
private async void deviceList_ItemSelected
(object sender, SelectedItemChangedEventArgs e)
{
    IDevice myDevice =
        BTLEDeviceClasses[e.SelectedItemIndex];
```

The next step is the procurement of a device instance and the listing of all services:

```
try
{
```

```
    await myAdapter.ConnectToDeviceAsync(myDevice);
    var services = await myDevice.GetServicesAsync();
    services = services;
    foreach (IService serv in services) {
        String aString = serv.Id.ToString();
        if (aString.CompareTo
            ("000000ff-0000-1000-8000-00805f9b34fb") == 0)
        { //Swap view
            App curApp = (App) Application.Current;
            curApp.myBTLEService = serv;
            await Navigation.
                PushModalAsync(new MainWorkPage()) ;
        }
    }
}
```

Here, the author decided to identify the remote station by checking the availability of a specific service. If a service with the required GUI is found, the currently displayed activity is changed.

It is important to use the `Navigation.PushModalAsync` method. This is because the author uses a page derived from [5] in his commercial application — the use of normal `PushAsync`, which is recommended in the Microsoft documentation, by the way, leads to strange crashes in connection with a window built according to this scheme.

The try-catch block is necessary because it protects against communication problems:

```
catch (DeviceConnectionException ex)
{
    // ... could not connect to device
}
```

The next task of the application is to communicate with the characteristics. In the interest of reducing the logic contained in the views,

the author relies on the structure shown in the flowchart in **Figure 3**.

If you place the communication function in an (ideally static) class, you do not have to create it in the individual pages. The author once again decided to use the `App` class in his application; the next step is the declaration:

```
public async void setLightMode(int _what)
{
    var x = await myBTLEService.
        GetCharacteristicAsync(Guid.Parse
            ("0000ff01-0000-1000-8000-00805f9b34fb"));
```

The first step is to use the `GetCharacteristicAsync` method, which determines a specific characteristic based on its respective GUID. As we performed an "identification" above, the author refrains from safeguarding the value of `x` in this piece of code — in practice, of course, it would be reasonable to do so.

It should be noted that the library would also be capable of a "global" search. The code required for this would look like this:

```
public async void setLightMode(int _what) {
    var ibx = await myBTLEService.GetCharacteristicsAsync();
    foreach (var chara in ibx) {
        var str = chara.Id.ToString();
        str = str;
    }
    ...
}
```

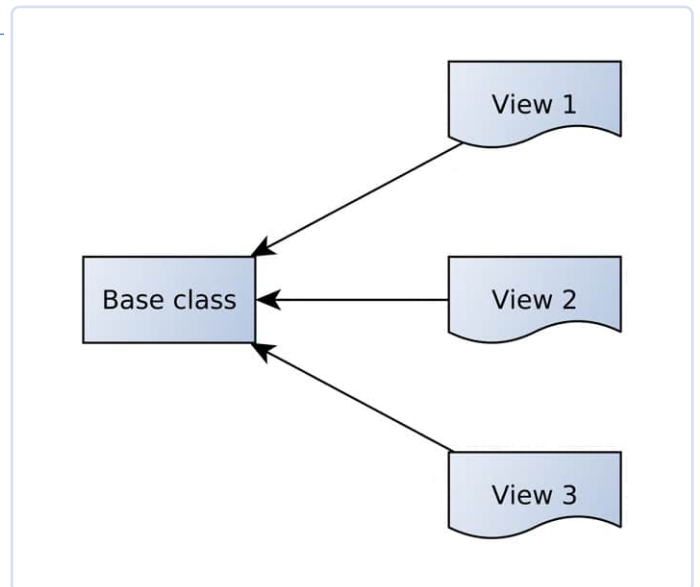


Figure 3: Isolating the communication logic reduces redundancy in the system.

The tautological instruction placed in this snippet is useful because it allows the "harvesting" of the GUIs of the individual characteristics. If you cannot extract the GUI characteristic from the ESP32 program in the form of a string, a convenient procedure is to set a breakpoint on the tautological instruction and execute the program behind it.

For each run, you can then extract the respective characteristics in the debugger — as shown in **Figure 4** — and (if required) compare them to the information displayed in the Bluetooth scanner.

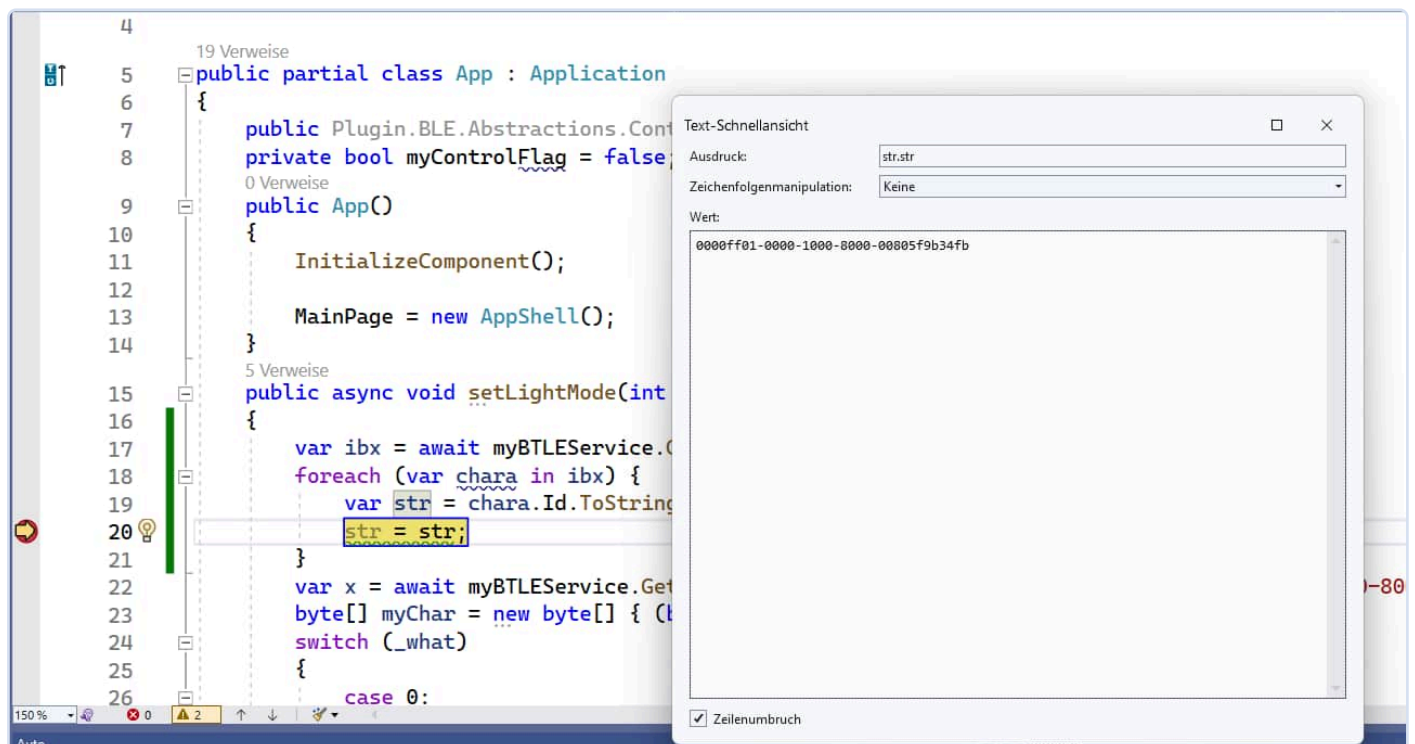


Figure 4: Tautological instructions save typing work!

## Bluetooth Scanners Are Helpful

Anyone implementing a Bluetooth LE application is well advised to install a Bluetooth LE scanner on their Android smartphone. This is an application that connects to any remote stations and lists the services and characteristics located there. The scanner application should be run on Android or an old iPhone with iOS 15. iOS 16 introduced changes to the Bluetooth stack, greatly reducing device visibility and limiting the functionality of the scanner!

The author likes to use the nRF Connect application in his company, which can be downloaded free of charge from the Play Store [7].

The next step is to assemble the payload. In the author's case, the proprietary communication protocol is based on data streams that transmit ASCII values. The payload is then assembled depending on the integer value supplied using array constants:

```
byte[] myChar = new byte[] { (byte)'0' };
switch (_what)
{
    case 0:
        myChar = new byte[] { (byte)'0' };
        break;
    case 1:
        myChar = new byte[] { (byte)'1' };
        break;
    case 2:
        myChar = new byte[] { (byte)'2' };
        break;
    case 3:
        myChar = new byte[] { (byte)'3' };
        break;
    case 4:
        myChar = new byte[] { (byte)'4' };
        break;
    ...
}
```

Last but not least, a connection establishment command is required, which has the following structure:

```
var y = await x.WriteAsync(myChar);
}
```

## Sending Data From the ESP32 to the Android Phone

As a final task, we want to realize the delivery of data from the ESP32 via the wireless interface. At this point, the author would like to go into a little more detail about the ESP-IDF code. The examples based on the GATT table system are, in the author's opinion, very poorly documented; there are some confused and only partially answered questions in the forum.

The most important question in this area is how the characteristics are initialized. If the constant `ESP_GATT_AUTO_RSP` is passed, as specified

by default in the example, the stack handles the values contained in the characteristic.

The application also receives a read event according to the following scheme; however, the Bluetooth stack takes the returned values from an internal memory and normally even sends them on their way before firing the event:

```
case ESP_GATTS_READ_EVT:
    ESP_LOGI(GATTS_TABLE_TAG, "ESP_GATTS_READ_EVT");
    break;
```

In theory, the `esp_ble_gatts_set_attr_value` method provides a function that allows the memory values stored in the Bluetooth stack to be adjusted. A naive implementation would then look like this:

```
void updateBTLECache() {
    esp_err_t ret;
    ret = esp_ble_gatts_set_attr_value
        (heart_rate_handle_table[IDX_CHAR_VAL_A],
         1, (const uint8_t *)"1");
}
```

The problem with this method is that it returns a null value even if the handle passed is invalid or has not yet been initialized by the Bluetooth stack.

This already completes the "problem description" — the setup or population of the handle array takes place relatively late. One way that the author's company has found to work well is to update the values in the BTLE cache whenever a new device connects to the ESP32. The following code is ideal for this purpose:

```
case ESP_GATTS_CONNECT_EVT:
    ESP_LOGI(GATTS_TABLE_TAG, "ESP_GATTS_CONNECT_EVT",
             conn_id = %d", param->connect.conn_id);
    updateBTLECache();
    esp_log_buffer_hex(GATTS_TABLE_TAG,
                      param->connect.remote_bda, 6);
```

Last but not least, here is a code snippet that illustrates the reading of the values on the MAUI side:

```
public async void setLightMode(int _what)
{
    var x = await myBTLEService.
        GetCharacteristicAsync(Guid.Parse
            ("0000ff01-0000-1000-8000-00805f9b34fb"));
    byte[] z = await x.ReadAsync();
    z = z;
    ...
}
```

The values supplied in the form of a byte array can then be processed more or less as you wish.



## Accelerating the Development

Our experiments show that MAUI's Bluetooth LE API can interact with other target systems without any problems. If you need a companion application for your embedded system and build it with MAUI, you can program in C# or Visual Basic and spare yourself a good part of the struggle with the native platforms. Especially if your company already has expertise with the .NET framework, this can lead to a significant acceleration of the development process. Of course, Bluetooth LE control on a smartphone is only "half the battle." In a previously published article [6], the author shows how to program an STM32 microcontroller with a BLE interface. [🔗](#)

*Translated by Jörg Starkmuth — 230381-01*

## Questions or Comments?

Do you have questions or comments about this article? Email the author at [tamhan@tamoggemon.com](mailto:tamhan@tamoggemon.com) or contact Elektor at [editor@elektor.com](mailto:editor@elektor.com).

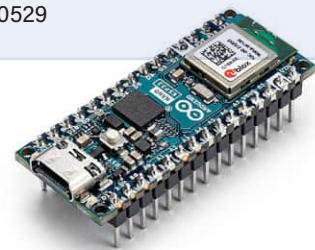
## About the Author

As an engineer, Tam Hanna has been working with electronics, computers, and software for more than 20 years. He is a self-employed designer, book author, and journalist (@tam.hanna on Instagram). In his spare time, Tam designs and produces 3D-printed solutions and, among other things, has a passion to trade and enjoy high-end cigars.



## Related Products

> **Arduino Nano ESP32 with Headers**  
[www.elektor.com/20529](http://www.elektor.com/20529)



## WEB LINKS

- [1] V. Krypczyk, "MAUI: Programming for PC, Tablet, and Smartphone," Elektor 1-2/2024: <https://elektormagazine.com/220442-01>
- [2] GATT Server Service Table: <https://bit.ly/45KupIU>
- [3] Plugin.Ble library: <https://github.com/dotnet-bluetooth-le/dotnet-bluetooth-le>
- [4] Brief introduction to Bluetooth Low Energy (BLE): <https://developer.android.com/develop/connectivity/bluetooth/ble/ble-overview>
- [5] Navigation.PushModalAsync method: <https://github.com/dotnet/maui-samples/tree/main/8.0/Navigation/FlyoutPageSample>
- [6] T. Hanna, "Bluetooth LE on the STM32," Elektor 1-2/2024: <https://elektormagazine.com/230698-01>
- [7] nRF Connect application: <https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp>

## Meet Elektor at **embeddedworld 2024** Exhibition&Conference

Chat with our editors, client-, and marketing-experts. Students, come for the chat, stay for the goodies.

Show us this news item when you visit us and get a great surprise!

Hall 5 - Booth 5-181  
 See you there!

JOIN OUR  
 SCAVENGER  
 HUNT WITH  
 ESPRESSIF!

 **elektor**  
 design > share > earn

# Port-Expanding Breakout Board

Increase the Number of I/Os on Your Dev Board

By Alessandro Sottocornola (Italy)

**Elettronica In**  
WWW.ELETRONICA.IN.IT

Are you tired of trade-offs due to the lack of I/O ports whilst developing your designs with a preferred platform? Do you want more to handle a multitude of signals and actuators? This breakout board, managed via I<sup>2</sup>C bus, enables you to add up to 16 I/Os per unit until you have a maximum of 128 I/Os on your existing system.

There are various practical situations where it is necessary to manage several signals and actuators with a micro-controller that's not featuring enough I/O pins. In such cases, it could be useful to leverage ICs known as "I/O expanders," whose function is to add a certain number of input and output lines and to control them via a serial I<sup>2</sup>C or SPI communication line.

To meet the need to multiply the I/O lines, we made a breakout board based on the Microchip MCP23017 [1], a 16-bit I/O expander. Additionally, we put on the board the minimal hardware to make it work, which in this case is a dip-switch and some pull-up resistors. Finally, to give you an idea of what can be accomplished with an I/O expander, we will propose an application example where the breakout board drives a series of relays (eight in all) mounted on a board that can be driven via TTL levels or button presses. But let's go in order and see first how this chip works.

## The MCP23X17

This versatile integrated chip provides 16-bit generic serial/parallel I/O expansion and is available in two versions: the one used here, the MCP23017, equipped with an I<sup>2</sup>C Bus interface, and the MCP23S17, an SPI variant. The chip is a 16-bit I/O-expander, divided into two ports of 8 bits each, interfaced via I<sup>2</sup>C Bus. This means that with only two wires, referenced to ground, it allows

the acquisition of the state of as many as 16 lines (input mode) or setting the logical state of each of them (output mode). The I/O lines are, by default, operating as inputs.

The MCP23017 consists of multiple registers in 8-bit configuration for input, output, and polarity selection. The system master can enable the I/Os as inputs or outputs by writing the corresponding I/O configuration bits (IODIRA/B). Data for each input or output is stored in the corresponding input or output register. Input Port register polarity can be reversed using the Polarity Inversion Register. All registers can be read from the system master. The 16-bit I/O port is structurally composed of two 8-bit ports — namely, port A and port B, which are headed by pins, 21...28 and 1...8, respectively. The MCP23X17 can be configured to operate in either 8-bit or 16-bit modes. Furthermore, it has the two interrupt pins, INTA and INTB, which can be mapped in two ways:

- Interrupt pins operate independently. INTA reflects interrupt conditions on Port A and INTB reflects interrupt conditions on Port B.
- Both interrupt pins go active when an interrupt occurs on either port.

## Circuit Diagram

As you can see in the schematic diagram in **Figure 1**, the breakout board is something extremely basic, since

on board there is the MCP23017 integrated circuit in DIP version, with all the pins connected to pin strips (which on the PCB are 2.54 mm pitch for insertion in other boards or breadboards). The three pins for setting the least significant bits of the peripheral I<sup>2</sup>C Bus address are connected to a three-way dip-switch (labeled SW1 in the wiring diagram) to set the communication address on the I<sup>2</sup>C Bus. The pins A0, A1, and A2 are pulled up by resistors R1 to R3 that keep them at logic level High when the switches are open.

The I/O-related pins of registers A and B have been arranged on both sides so that we can connect our board with ease. At these digital I/O pins, you can connect anything you want, within the limits of the current and voltage supported by the MCP23017 IC. In addition to the 2.54-mm pitch side pin strips, another four pins have been provided on a strip (also 2.54-mm pitch) called *I2C*, so that we can connect to the bus directly at the top of the module for a greater flexibility. These pins, that include the 5 V positive supply and ground, are connected in parallel to the corresponding ones on the side connectors, that can also be used in alternative. Both SDA and SCL have pull-ups.

Note that on the circuit board, for convenience, the A register pins have been placed on one side and the B pins are on the opposite side, again with the purpose of simplifying connections. As said, the pins belonging to the I<sup>2</sup>C bus have been returned to the pin-strip marked *I2C*, along with the 5 V positive and ground; both have pull-ups. The I/O-expander reset in our breakout board is not used; therefore, to disable it, we placed the relevant pin (18, RST) at logic level High via resistor R4.

Between the side pins of the board, we also find repeated GND and +5 V lines. The entire circuit is powered by the 5 V contact (we actually have two contacts: 1 and 15, located on the long sides of the breakout board) referenced to ground (GND contacts, i.e., 2 and 23 of the side rows).

## The I/O Expander

The main element of the circuit is, of course, the MCP23017, which we can consider to be an I<sup>2</sup>C Bus/parallel converter, manufactured by Microchip (marked U1). The integrated circuit, of which you see the internal block diagram in **Figure 2**, functions as a peripheral (Slave) of the I<sup>2</sup>C bus and supports two input and output modes. In the former, it allows the I/O states of the A and B registers to be transferred to the bus in serial format, one byte for each register, at the request of the I<sup>2</sup>C Bus Master device; in the latter, it goes to set the I/O lines by converting the incoming data on the I<sup>2</sup>C bus to the corresponding state of the A and B register lines.

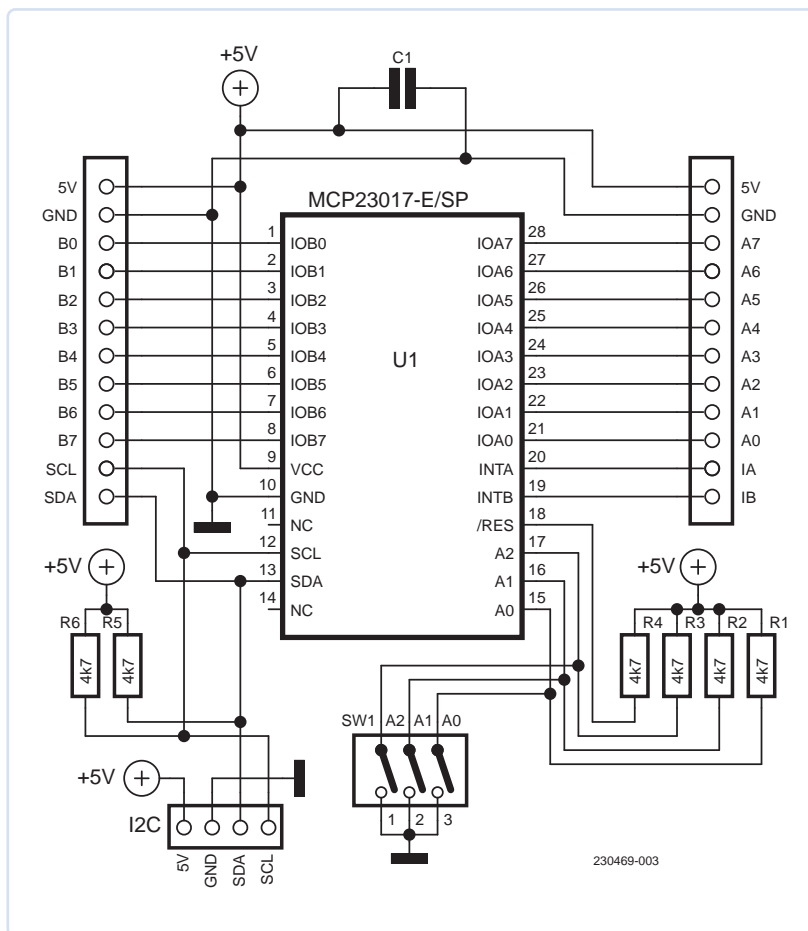


Figure 1: The schematic diagram of the breakout board.

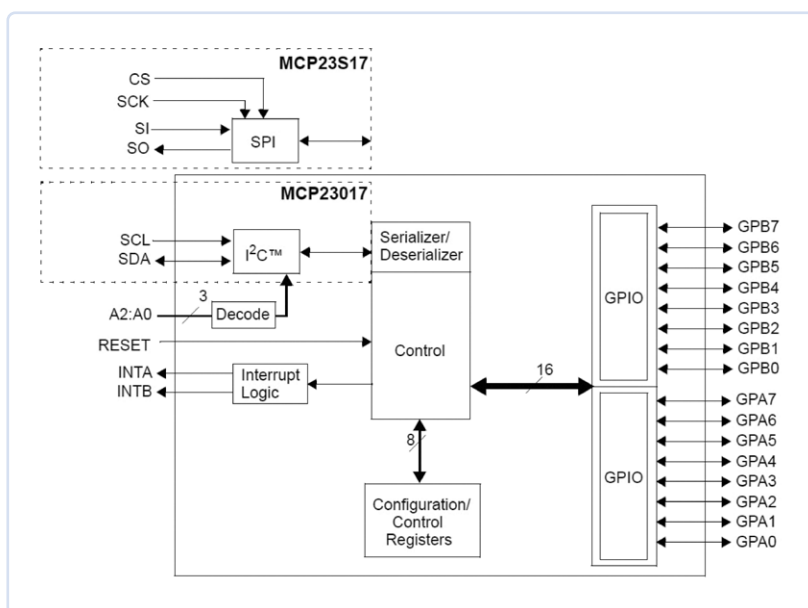


Figure 2: Block diagram of the MCP23017. The SPI variant, which is labeled MCP23S17, is also shown. (Source: Microchip [3])

### The MCP23017 IC in Brief

The chip at the core of the breakout board described here is an I/O expander with 16 bits, divided into two ports of 8 bits each, interfaced via I<sup>2</sup>C-Bus. This means that with only two wires, referenced to ground, it allows you to acquire the status, or set it as the output, of as many as 16 lines. Its technical characteristics are as follows:

- High-speed I<sup>2</sup>C data interface, operating at 100 kHz, 400 kHz or 1.7 MHz
- I<sup>2</sup>C bus address settable in 8 combinations
- Configurable interrupt pins, by level and logic function
- Configurable interrupt source
- Reversed polarity register

For the inputs:

- External reset input
- Standby current of 1 µA max.
- Supply voltage from 1.8 V to 5.5 V

The interrupt output can be configured to trigger under two (mutually exclusive) conditions:

- When an input state differs from its corresponding input port register state. This condition is used to indicate to the system master that an input state has changed.
- When the state of an input is different from the preconfigured value of the register (DEFVAL register).

The INTA and INTB interrupt lines can be configured as active-high, active-low or open-drain. The Interrupt Capture register captures the values of the ports at the time the interrupt is triggered, thus storing the condition that caused the interrupt. The Power-On-Reset (POR) sets the registers to their default values and initializes the device state machine. The need for bidirectional operation is due to the fact that each I<sup>2</sup>C Bus peripheral must both be able to read (e.g., commands) and be able to send acquired, 8+8 bit data along the bus.

Like all units for I<sup>2</sup>C Bus, the MCP23017 allows its address to be set in a range of eight addresses, and for this purpose, it has pins A0, A1, A2, which allow the addresses of the Slave unit to be set, if addressed directly from the I<sup>2</sup>C Bus. Each of these lines is set through a dip-switch switch SW1: each closed dip sets the logical zero on the respective address, while, vice versa, an open dip determines the logical state 1. The ability to define eight addresses allows up to eight I/O-expanders to be placed on the same bus and thus control a maximum of 128 I/Os with only three lines. For each application to which you will want to allocate the breakout board, in **Table 1**, we give you the correspondence between the addresses and the dip-switch setting.

**Table 1: Setup of MCP23017 peripheral address.**

ADDR	A2	A1	A0
0x20	ON	ON	ON
0x21	ON	ON	OFF
0x22	ON	OFF	ON
0x23	ON	OFF	OFF
0x24	OFF	ON	ON
0x25	OFF	ON	OFF
0x26	OFF	OFF	ON
0x27	OFF	OFF	OFF

With this hardware, the logic of operation is as follows: whenever it receives a string along the SDA line of the I<sup>2</sup>C Bus (paced by the clock signal on the SCL line), the MCP23017 integrated circuit executes the command contained therein (in this case the one indicating to load the data byte) and arranges the eight output lines IOA0...IOA7 and IOB0...IOB7 as the relevant bits. For example, IOA0 will assume the state of the first bit of byte 1, IOA1 that of the second bit, and so on. The same will occur on IOB0...IOB7, which will exactly replicate the bits of the second data byte.

Of course, conversion and output occur only assuming the received string contains the I<sup>2</sup>C Bus address corresponding to the one set, via the dip-switches of SW1, for U1. When each string is received, the IC updates the status of its outputs, and the respective logic levels determine whether the downstream circuitry (i.e., LEDs or display segments) are turned on, or remain in the off state; if no string is subsequently sent, the output status retains the last byte pattern because the outputs of the MCP23017 IC are latched. The above applies to output mode, i.e., writing the status of the two I<sup>2</sup>C-Bus bytes to output registers A and B. If, on the other hand, the command coming from the bus is read, the MCP23017 acquires the I/O status of each register and generates two bytes, the first containing the status of IOA0...IOA7 and the second the logical condition of IOB0...IOB7; it then sends them along the I<sup>2</sup>C bus as a response.

### Practical Realization

Well, now that we have described the wiring diagram, we can move on to the construction notes. Then we will propose an application example based on interfacing with an Arduino board accompanied by the corresponding sketch. As usual, we have drawn a printed circuit board of which we make available to you for download the two layouts (it's a double-sided board) at the Elektor Labs Webpage of this project [2]. From them, you can proceed to prepare the PCB by photoengraving. After you'll have etched and drilled it, you can assemble the



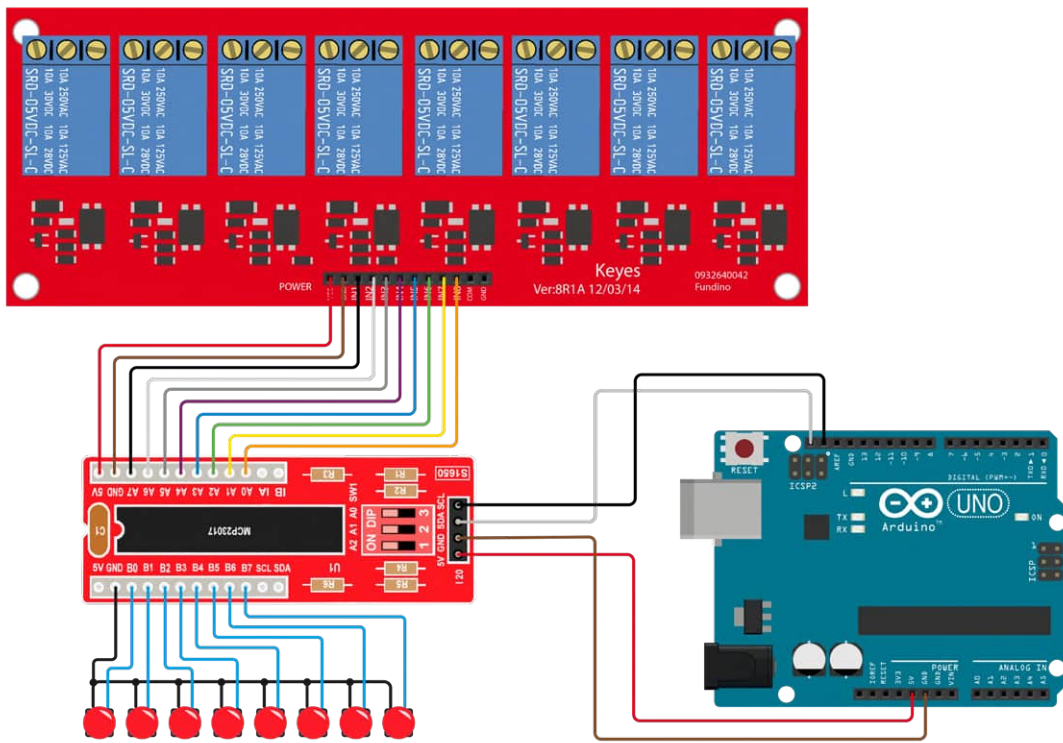


Figure 3: Wiring diagram of the pushbutton-operated relays control application.



## Component List

### Resistors

R1...R6: 4.7 k

### Capacitor

C1 = 100 nF, ceramic

### Semiconductor

U1 = MCP23017-E/SP

### Various

SW1 = 3-Way Dip-Switch

1 x 14+14 Pin DIL socket

2 x 12 Pin Header, male

1 x 4 Pin Header, male

1 x PCB (see text)

very few components needed, which, in this project, are all traditional, TH-type, for those who do not like the SMT technique too much. First, insert and solder the resistors, then proceed with the socket for the integrated circuit (to be placed with the notch facing as shown in the assembly plan you see in this article) and the 3-way dip-switch, to be mounted with switch 1 facing left, looking at the board with the socket for U1 on top.

Finally, insert and tin into the respective holes the four-pin strip marked I2C, and then, on the opposite side of the PCB, insert and tin two rows of 12 pins headers that will allow mounting on breadboards or insertion on other boards, i.e., to eventually connect with Arduino using classic male/female wire jumpers. Once you're done soldering the components, insert the MCP23017 into the socket, holding it with the notch facing as shown

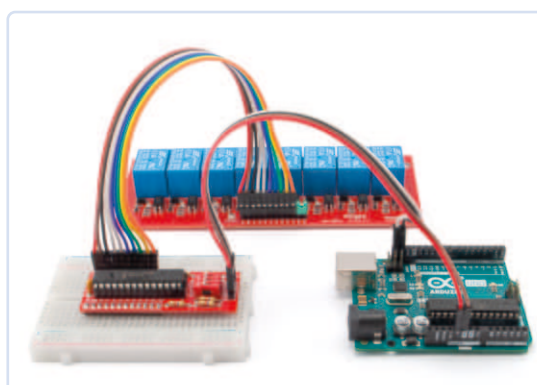
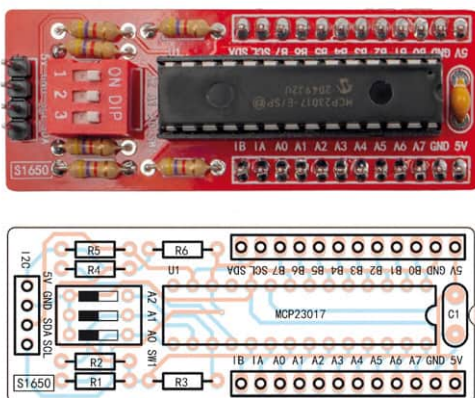


Figure 4: The hardware for testing the example sketch.



### Listing 1: Demo project.

```
#include <Adafruit_MCP23X17.h>.
#include <Adafruit_MCP23X17.h>
Adafruit_MCP23X17 mcp;
int i = 0;
int OUT[] = ;           //Represents MCP23017 PIN (A7...A0)
int IN[] = ;            //Represents MCP23017 PIN (B0...B7)
int STATO[] = ;         //For each output, every toggle the status is being saved
void setup()
{
  Serial.begin(9600);
  Serial.println("MCP23017 INPUT/OUTPUT");
  if (!mcp.begin_I2C(0x20))           //0x20 is MCP23017's address with A0=A1=A2 > ON(GND)
  {
    Serial.println("MCP Error!"); //If MCP is not found, the error is visualized
    while (1);
  }
  //bank A Pin set as outputs and B bank as inputs
  //The STATO variable to 0 to indicate idling outputs
  for (i=0; i<8; i=i+1)
  {
    mcp.pinMode(OUT[i], OUTPUT);
    mcp.pinMode(IN[i], INPUT_PULLUP);
    STATO[i] = 0;
  }
}

//***** L O O P *****
void loop()
{
  String Testo_Debug = "";
  for (i=0; i<8; i=i+1)
  {
    //If button pressed or output not activated, I activate it
    if ((mcp.digitalRead(IN[i])==0) && (STATO[i]==0))
    {
      STATO[i] = 1;
      Testo_Debug = "Pulsante " + String(i+1) + " premuto";
      Serial.println(Testo_Debug);
      mcp.digitalWrite(OUT[i], HIGH);
    }
    //If button released and output is active,I de-activate it
    if ((mcp.digitalRead(IN[i])==1) && (STATO[i]==1))
    {
      STATO[i] = 0;
      Testo_Debug = "Pulsante " + String(i+1) + " rilasciato";
      Serial.println(Testo_Debug);
      mcp.digitalWrite(OUT[i], LOW);
    }
  }
  delay(10);
}
```

in the mounting plan on these pages. With that done, your breakout board is ready for experimentation or prototyping.

## Let's Use It With Arduino

The breakout board was created to be interfaced to a microcontroller, since typically I/O-expanders are used by devices equipped with a serial interface to I<sup>2</sup>C Bus. Since Arduino supports this bus, we have made a sample code to read and manage the MCP23017's I/O via Arduino, it is downloadable at [2]. This sketch basically allows you to write the state of the Port A register as a function of a byte sent by Arduino along the bus, whose bits correspond to the state read on Port B, which this time acts as the input. To give the example a concrete application, we decided to use the logical states of the I/Os of Port A, which will operate here as digital outputs, to drive a relay board. Specifically, we need to connect the eight relay output control lines of an 8-Channel-Relay board to the I/O bank of Port A, while eight normally open pushbuttons are to be connected to Port B, having in common the pole connected to GND. To realize this application, it is necessary to connect Arduino UNO, the breakout board, the relay board and the buttons, as shown in the wiring diagram proposed in **Figure 3**, whilst **Figure 4** shows the real prototype of this application. Since these are quite common pushbuttons (normally open) and have no external electronics, the internal pull-ups of the MCP were enabled (via library) to handle them and recognize the change of state, so we activated the respective output when the button is brought to ground (GND).

To accomplish this small demo project, simple Arduino UNO-based code was written, taking advantage of the Adafruit library, which, as you see in **Listing 1**, is included in the first line of the sketch.

Before loading the code into the program memory of our Arduino board, it is essential to download the library from [www.adafruit.com](http://www.adafruit.com) and install it using the Library Manager included in the IDE, or simply extract the contents of the ZIP file and then copy the entire *Adafruit\_MCP23017\_Arduino\_Library* folder within the *libraries* directory

normally found within the operating system at the path *Documents\Arduino\libraries*. Having loaded the library, it will be sufficient to load our example code and load it inside the board after choosing the correct COM port from the *Tools* menu of the IDE.

In the code, a byte is requested to be sent to the MCP23017, containing the status of the buttons, and the related data is processed and then written to a byte directed to the integrated circuit, which will set the status of the Port A lines in a stable manner, until the refresh. For the interfacing to work, dip-switches A0, A1 and A2 must be set correctly because if the sensor was not assigned the address 0x20, an error message would be shown on the serial; the breakout board address is assigned with the 000 combination of the three bits A0, A1, A2 (i.e., closing all three dip-switches to ground). If you want to change the address, refer to Table 1, with the understanding that you must also change the address reported in the sketch. ◀

230469-01



### Related Product

> **Arduino UNO Rev3**  
[www.elektor.com/15877](http://www.elektor.com/15877)



## WEB LINKS

- [1] Microchip's MCP23017 webpage : <https://microchip.com/en-us/product/mcp23017>
- [2] Elektor Labs webpage for this project: <https://elektormagazine.com/labs/port-expanding-breakout-board>
- [3] MCP23017/MCP23S17 datasheet (PDF): <https://tinyurl.com/mcp23017datasheet>



# AI Specialist

## Machine Learning with the Jetson Nano

By Tam Hanna (Hungary)

Artificial intelligence on GPUs does not necessarily mean expensive graphics cards and massive power consumption.

NVIDIA has been moving towards smaller systems for some time now, certainly also due to the competition from microcontroller manufacturers. In this article, we look at the Jetson Nano and a small demo application.

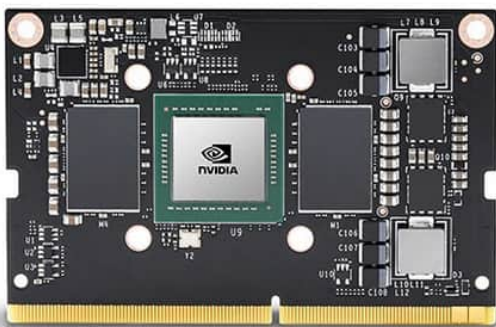
The market for artificial intelligence (AI) acceleration in embedded systems is in flux. Companies such as Canaan and Maxim Integrated are fighting tooth and nail for supremacy. ARM has also driven a gladiator into this arena with the announcement of the Cortex M52. The aim is to offer small AI systems that perform basic AI tasks on the much talked-about *edge* without a connection to the Internet. This leads to a much more stable system architecture because the ML or AI tasks can then continue to be processed even if the connection to the parent server is lost. NVIDIA has been trying to play in this area for some time with the Jetson series.

In the interest of didactic honesty, it should be noted that custom board designs with these systems are not really manageable for smaller companies, if only because of the extreme bandwidth required to access the DDR RAM. NVIDIA is aware of this problem. The portfolio overview available at [1], which provides electronics engineers with an overview of the entire Jetson ecosystem, therefore also includes various “compute cards,” such as the Jetson TX2 shown in **Figure 1**.

It should also be noted that due to NVIDIA's dominance in the GPU sector, its well-developed third-party ecosystem has now also discovered the Jetson product range. A list of various third-party products can be found at [2], the integration of which into an in-house solution can save a great deal of development time (think, for example, of housing design and the selection of cameras and the like).

### Getting the System Up and Running

For “lateral entry,” NVIDIA offers the NVIDIA Jetson Nano Developer Kit shown in **Figure 2** (next to a Raspberry Pi and an Orange Pi 5+). At the time this article went to press, the best OEMSecrets price (see [3]) was €155. While this is slightly more expensive than a Raspberry Pi, the NVIDIA technology is better integrated into the general AI ecosystem. When purchasing the NVIDIA Jetson Nano, it is recommended that you also order a power supply unit with a barrel connector with the common dimensions of 5.5/2.1 mm. In the following steps, the author uses a MeanWell GST25E05-P1J.



### Jetson TX2 Series

The extended Jetson TX2 family of embedded modules gives you up to 2.5X the performance of Jetson Nano in as little as 7.5W. Jetson TX2 NX offers pin- and form factor compatibility with Jetson Nano, while Jetson TX2, TX2 4GB, and TX2i all share the original Jetson TX2 form factor. The rugged Jetson TX2i is ideal for settings including industrial robots and medical equipment.

[Learn More >](#)

Figure 1: Using this prefabricated module makes it much easier to integrate the Jetson into in-house circuits. (Source: NVIDIA)



A careful look at the system (see also **Figure 3**) reveals that it is a two-part product. In addition to the carrier board, which exposes the various interfaces, there is the actual computing module with the heat sink. It is relevant that a microSD card with the operating system must be inserted into the computing module.

### Micro-USB Is Also Possible in Theory

If you have a very powerful Micro-USB power supply at hand and you don't use the Micro-USB port to connect external hardware, you can also choose this method of power supply. However, due to "quarrels" with the Raspberry Pi, the author, as a burnt child, dreads the fire and relies on a classic DC power supply unit.

The built-in SoC is a multicore system that also has four fully-fledged Arm Cortex A57 cores in addition to the actual AI accelerator. This results in a setup reminiscent of classic process computers that usually run Embedded Linux. NVIDIA recommends using a MicroSD card with at least 32 GB of capacity and a minimum speed of UHS1. The image file can be found at [4], which you can extract to your memory card using a card reader as usual.

A USB mouse and a USB keyboard are also required for the initial start-up; Jetson supports both HDMI and DisplayPort for screen output. We will assume that an Ethernet cable is also connected in the following steps, as this is reasonable.

### Not for Clumsy Hands!

The microSD holder of the Jetson module is based on the form-fit principle. It is inserted and removed by "pushing it in deeply" – if you use electronic tweezers and a fine screwdriver, you can complete the operation without dismantling the evaluation board.

The lack of a camera proves problematic during setup. Instead of a CCD sensor, NVIDIA uses two of the connectors familiar from Raspberry Pi 4 and the like. Some Raspberry Pi cameras can be connected directly to the Jetson. In the following steps, the author uses a Raspberry Pi 2 camera that connects to the CAM0 port via an FPC cable designed for the Raspberry Pi. It should only be noted that the version of the cable intended for the Raspberry Pi 5 is not compatible with the Jetson. Various 3D models for positioning the camera are available in Thingiverse, and printing them will make life easier for the developer.

If you want to put the NVIDIA Jetson into operation using the MeanWell power supply unit mentioned above, you must be aware of a little beginner's pitfall. The jumper shown plugged in **Figure 4** is not plugged in when delivered: In this case, the DC connector is not connected to the power supply. When the power supply unit is plugged in, the status LED does not light up, which can be confusing.



Figure 2: The Jetson, next to a Raspberry Pi 5 and an OrangePi 5 Plus.



Figure 3: If you loosen the two screws, you can take your developer kit apart.

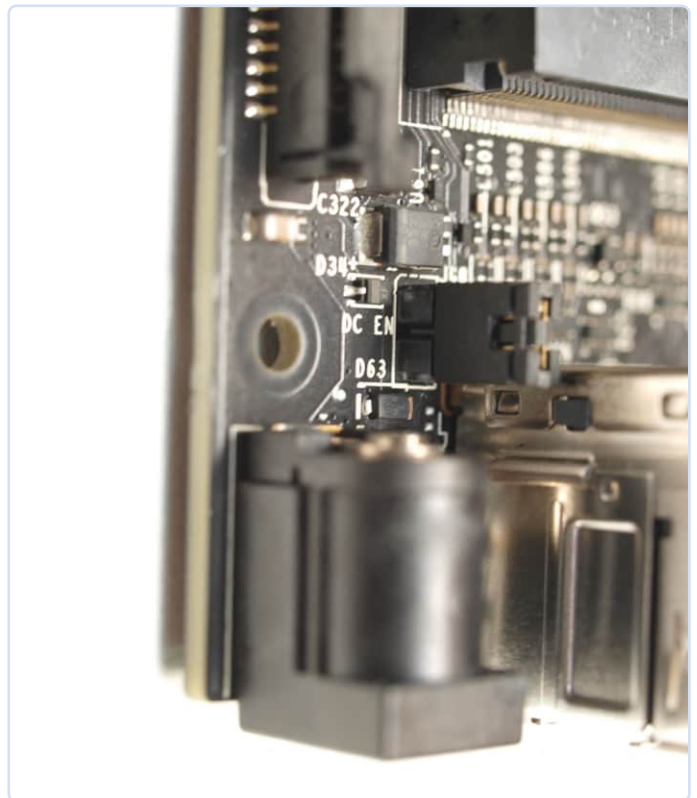


Figure 4: This jumper is of crucial importance.

During the first boot, the process computer immediately switches on the HDMI screen — the reconfiguration of the microSD card and some other “initiation rituals” take some time. Once the work is done, the system presents an Ubuntu desktop and then prompts you to complete the welcome wizard.

In general, this is the usual Ubuntu setup wizard, but NVIDIA has added a step to approve the in-house software licenses, among other things. The wizard for selecting the power model is important: It is recommended to keep the selected option. It ensures that the Jetson receives the maximum amount of power.

Once the wizard has been successfully completed, there is still some time to go before the Jetson board is ready for action — some prompts to reboot the system may also appear on the desktop.

## First Experiments

NVIDIA relies on a more or less completely standard-compliant Ubuntu 18.04 in the Jetson. The author likes to configure such process computers for remote access to save himself the trouble of switching between the PC and process computer keyboards.

However, the settings application included in Ubuntu is unsuitable for this task, which is why we run `sudo apt-get update` and `sudo apt-get upgrade` to update the package inventory in the first step instead. Any queries must be acknowledged by pressing *Enter*; the restart of the Docker system must be allowed. The Jetson must then be restarted. Entering `sudo apt install vino` ensures that the VNC server is ready for use.

Finally, the following commands are required to bring the configuration into a usable state. The string `thepassword` must, of course, be replaced by a password suitable for your installation.

```
tamhan@tamhan-desktop:~$ mkdir -p ~/.config/autostart
tamhan@tamhan-desktop:~$ cp /usr/share/applications/
vino-server.desktop ~/.config/autostart
tamhan@tamhan-desktop:~$ gsettings set org.gnome.Vino
prompt-enabled false
tamhan@tamhan-desktop:~$ gsettings set org.gnome.Vino
require-encryption false
tamhan@tamhan-desktop:~$ gsettings set org.gnome.Vino
authentication-methods "['vnc']"
tamhan@tamhan-desktop:~$ gsettings set org.gnome.Vino
vnc-password $(echo -n 'thepassword'|base64)
```

After the next reboot, the Jetson can be accessed with the VNC client Remmina if a user is logged in. Please note, however, that the applet for remote access in the Settings application still does not work.

As a next step, you can perform an initial test of the camera connection by entering the following command:

```
tamhan@tamhan-desktop:~$ gst-launch-1.0 nvarguscamerasrc
! nvoverlaysink
```

The camera preview that can be activated with this basic command only appears on a monitor physically connected to the Jetson — the system communicating with the computer via VNC sees the terminal output instead. To end the whole process, it is sufficient to send an interrupt event using *Control + C*.

The actual camera access is then — in general — carried out using methods known from the PC or workstation. Of particular interest is the file [5], which demonstrates the setup of an Open CV-based pipeline.

The following commands are required to make it executable:

```
tamhan@tamhan-desktop:~$ git clone https://github.com/
JetsonHacksNano/CSI-Camera
tamhan@tamhan-desktop:~$ cd CSI-Camera/
tamhan@tamhan-desktop:~/CSI-Camera$ python3 simple_
camera.py
```

The launched camera window is then also visible via VNC because it does not send the information directly to the frame buffer of the Tegra GPU.

If the camera image is upside-down, adjusting the parameter `flip_method` in the file will help:

```
def show_camera():
    window_title = "CSI Camera"
    print(gstreamer_pipeline(flip_method=2))
    video_capture = cv2.VideoCapture(gstreamer_
        pipeline(flip_method=2), cv2.CAP_GSTREAMER)
    if video_capture.isOpened():
        . . .
```

## Interacting with GPIO Pins (via Python)

Artificial intelligence systems require a completely different skillset from the developer, which generally has little overlap with classic embedded development. Practical experience shows that people from outside the field with a background in traditional mainframe computer technology often get to grips with AI better than embedded developers. The author's partner, who programs very efficiently in Java, solves AI tasks faster — but she has little knowledge of assembly language.

The purpose of this little detour into software architecture is to point out that we are intentionally illustrating GPIO access on the Jetson in Python in the following steps. The reason for this is that the majority of user-oriented creation of AI systems takes place in Python: If you rely on C here, you will end up with an unnecessary native interface.

The standard distribution of Python does not include package management on the Jetson, which is why the commands `sudo apt install python-pip` and `sudo apt install python3-pip` must be entered. The next step is to check that the GPIO modules

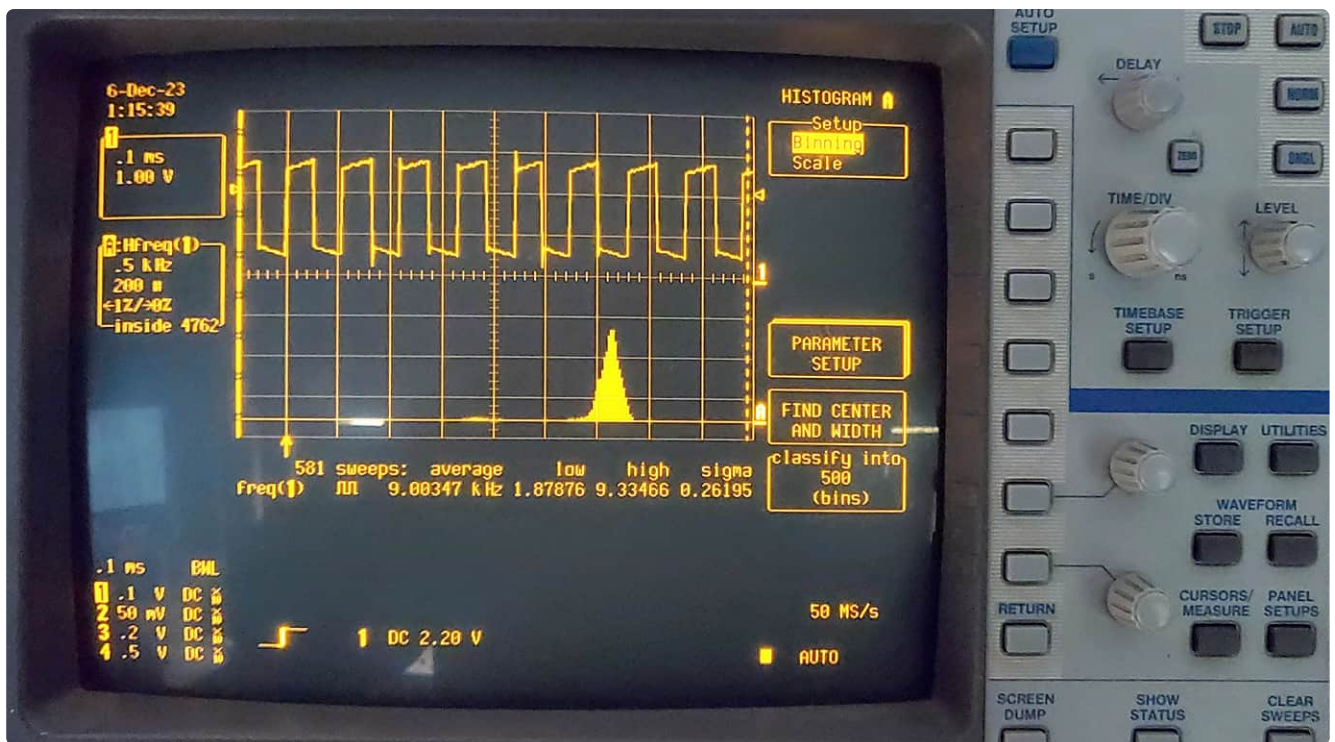


Figure 5: The NVIDIA Jetson produces square wave signals using Python.

have been installed correctly. Two commands are also required here because the Python environments are (naturally) not really capable of sharing their library repository:

```
tamhan@tamhan-desktop:~$ sudo pip install Jetson.GPIO
tamhan@tamhan-desktop:~$ sudo pip3 install Jetson.GPIO
```

We cannot present the Jetson GPIO API in full here due to lack of space. Under [6] you will find a group of ready-made examples that explain the API in full.

The following program is sufficient for our little test:

```
import RPi.GPIO as GPIO
import time

output_pin = 18 # BCM pin 18, BOARD pin 12

def main():
    GPIO.setmode(GPIO.BCM)
    # BCM pin-numbering scheme from Raspberry Pi
    GPIO.setup(output_pin, GPIO.OUT, initial=GPIO.HIGH)

    print("Starting demo now! Press CTRL+C to exit")
    try:
        while True:
            GPIO.output(output_pin, GPIO.HIGH)
            GPIO.output(output_pin, GPIO.LOW)
            GPIO.output(output_pin, GPIO.HIGH)
            GPIO.output(output_pin, GPIO.LOW)

    finally:
        GPIO.cleanup()
```

```
if __name__ == '__main__':
    main()
```

What is particularly interesting here is that NVIDIA offers the budding Jetson developer several ways to address the pins — we use the `GPIO.BCM` option here, which is based on the Raspberry Pi pinout. The reward for our efforts is the screen image shown in **Figure 5** — the frequency stability may not be particularly high, but it is more than sufficient for triggering IoT events.

It should also be noted that the execution of GPIO program examples without superuser rights can sometimes cause problems. See [7] for a detailed discussion of this topic.

## Experiments with the ML Function

In theory, the availability of a fully-fledged Linux operating system and a (very fast) USB3 interface encourages the execution of experiments. One possible example would be Stable Diffusion: In practice, it is possible to use the Jetson as an image generator using an (adapted) runner.

In practice, however, this approach is not recommended: Both the comparatively small graphics memory of only 4 GB VRAM and the small number of “only” 128 cores ensure that image generation requires patience. On Reddit, there are reports from ML experimenters who estimate up to 5 hours of computing time per image with a cluster size of  $512 \times 512$  pixels.

### Jump-Start for Popular Frameworks

NVIDIA is trying to make it as easy as possible for developers to implement various widely used ML frameworks. A list of supported products including — optimized — installation instructions can be found at [8].



```

tamhan@tamhan-desktop: ~/jetson-inference/build/aarch64/bin
[OpenGL] glDisplay -- X screen 0 resolution: 1920x1200
[OpenGL] glDisplay -- X window resolution: 1920x1200
[OpenGL] glDisplay -- display device initialized (1920x1200)
[video] created glDisplay from display://0
-----
glDisplay video options:
-----
-- URI: display://0
- protocol: display
- location: 0
-- deviceType: display
-- ioType: output
-- width: 1920
-- height: 1200
-- frameRate: 0
-- numBuffers: 4
-- zeroCopy: true
-----
[image] loaded 'images/orange_0.jpg' (1024x683, 3 channels)
imagenet: 96.68% class #950 (orange)
[OpenGL] glDisplay -- set the window size to 1024x683
[OpenGL] creating 1024x683 texture (GL_RGBA format, 2098176 bytes)
[cuda] registered openGL texture for interop access (1024x683, GL_RGBA, 2098176 bytes)
[image] saved 'images/test/output_0.jpg' (1024x683, 3 channels)

[TRT] -----
[TRT] Timing Report networks/Googlenet/bvlc_googlenet.caffemodel
[TRT] -----
[TRT] Pre-Process CPU 0.14219ms CUDA 0.64583ms
[TRT] Network CPU 45.00319ms CUDA 44.40969ms
[TRT] Post-Process CPU 0.32704ms CUDA 0.31964ms
[TRT] Total CPU 45.47242ms CUDA 45.37515ms
[TRT] -----

[TRT] note -- when processing a single image, run 'sudo jetson_clocks' before
to disable DVFS for more accurate profiling/timing measurements

tamhan@tamhan-desktop:~/jetson-inference/build/aarch64/bin$

```

Figure 6: The demo program supplied with the Jetson is not very communicative at the command line level.

The same applies to the “end-to-end” solutions that are often demonstrated — training a model requires so much computing power and resources that NVIDIA recommends outsourcing to a desktop or mainframe in most tutorials — the Jetson is then parameterized with the ready-to-use model weights.

At [9] you will find more or less ready-to-use examples that illustrate the performance of the Jetson in a straightforward manner.

To use this model powershow, it is necessary to download and deploy an NVIDIA software package. In theory, the use of a Docker container is also possible here — for people unfamiliar with container technology, local compilation is an alternative, which can be achieved by entering the following commands:

```

sudo apt-get update
sudo apt-get install git cmake libpython3-dev
python3-numpy
git clone --recursive --depth=1 https://github.com/
dusty-nv/jetson-inference
cd jetson-inference
mkdir build
cd build
cmake ../
make -j$(nproc)

```

```

sudo make install
sudo ldconfig

```

Feel free to answer the question about installing the training components as you wish — the author has answered No in the following steps in order to save some memory on his microSD card, which is only 32 GB in size.

After successfully completing the compilation process, a relatively complex project structure can be found in the directory `~/jetson-inference/build/aarch64/bin`, which provides Python files in addition to various binary files. It is interesting to note that NVIDIA even places some ready-made test examples here.

The first thing we want to do is use the classifier — it analyzes image files and determines what can be seen in the supplied image:

```

tamhan@tamhan-desktop:~/jetson-inference/build/aarch64/
bin$ ./imagenet.py images/orange_0.jpg images/test/
output_0.jpg

```

The first execution of this command takes a little more time because the modules provided are optimized for the requirements of the Jetson system. The timing information shown in **Figure 6** is then displayed.



To actually visualize the results delivered by the ML process, you need to view the generated image file instead — entering the following command opens the output folder directly in the Nautilus file manager:

```
tamhan@tamhan-desktop:~/jetson-inference/build/aarch64/bin$ nautilus images/test/output_0.jpg
```

The reward for your efforts is the appearance of the screen image shown in **Figure 7**.

### Analysis of the Python File

Next, we will take a quick look at the example program we have just used. Its source code can be opened by entering the following command line:

```
tamhan@tamhan-desktop:~/jetson-inference/build/aarch64/bin$ gedit imagenet.py
```

Even at first glance, it is obvious that the library used here is related to the classic *ImageNet* — NVIDIA packages various widely used artificial intelligence systems in the Jetson starter kit.

The core of the program example is an endless loop that takes an image from the input stream, feeds it to the ML model and finally outputs the “calculated” information (**Listing 1**).

The initialization of the data streams and the model to be used is carried out further above according to the following scheme:

```
net = imageNet(args.network, sys.argv)
input = videoSource(args.input, argv=sys.argv)
output = videoOutput(args.output, argv=sys.argv)
font = cudaFont()
```

Another interesting aspect is the procurement or population of the `network` parameter, which supplies the name of the model to be processed. Here, NVIDIA relies on the `ArgParser` class, which is —

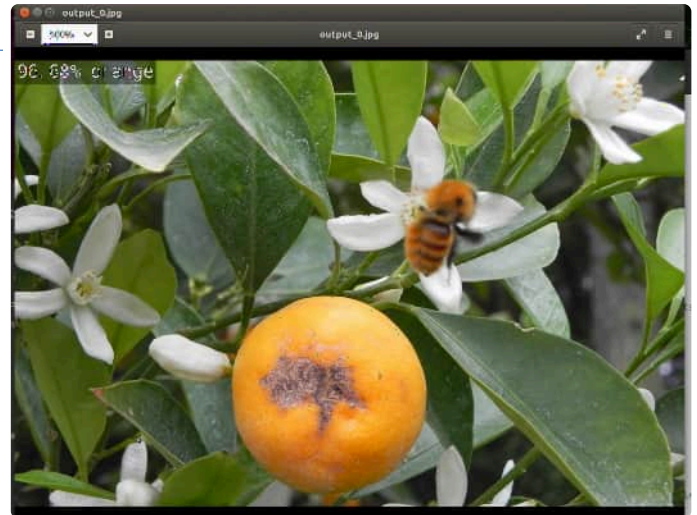


Figure 7: The NVIDIA sample program proved to be an exemplary fruit detector.

normally — specialized in the processing of parameters supplied via the command line.

In the case of this declaration, a default value is entered that normally activates and loads the GoogLeNet:

```
parser.add_argument("--network", type=str,
                    default="googlenet", help="pre-trained model to load
                    (see below for options)")
```

### Side Note: “Guided” Online Training Including Certification

The Soviet Union’s successes in various Arab and African countries can be attributed in part to the close training partnership — what the cadet learns is what he later uses in his job. NVIDIA is obviously aware of this situation, which is why Jetson AI Certification — as shown in **Figure 8** — is a completely free two-part ML course. It should also be emphasized that successful completion of the course is rewarded by NVIDIA with a certificate.

If you are interested, we recommend that you visit [10]. There you will find further information on how to organize your participation in the NVIDIA training course most efficiently.

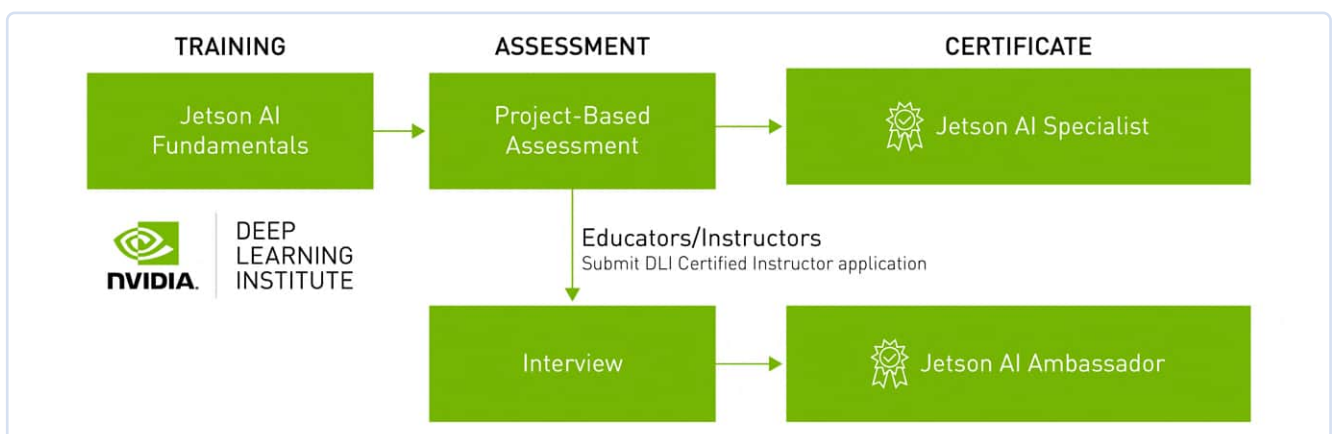


Figure 8: Two-track education, NVIDIA style! (Image source: NVIDIA [10])



## Listing 1: Classification.

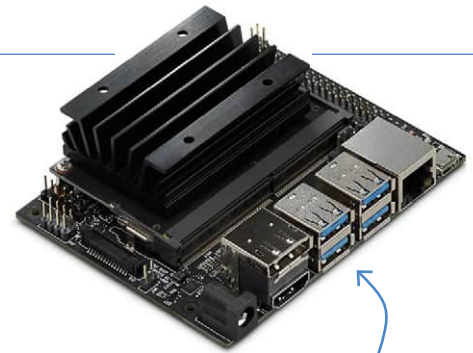
```
while True:
    img = input.Capture()

    if img is None: # timeout
        continue
    predictions = net.Classify(img, topK=args.topK)
    for n, (classID, confidence) in enumerate(predictions):
        classLabel = net.GetClassLabel(classID)
        confidence *= 100.0

    print(f"imagenet: % class # ()")

    font.OverlayText(img, text=f"% ",
                     x=5, y=5 + n * (font.GetSize() + 5),
                     color=font.White, background=font.Gray40)

output.Render(img)
```



### Related Products

➤ **NVIDIA Jetson Nano Developer Kit (B01)**  
[www.elektor.com/19001](http://www.elektor.com/19001)

## Advantages due to Linux

With the Jetson, NVIDIA is sending a hybrid into the race that falls between all the stools in the field of artificial intelligence. On the one hand, dedicated low-power microcontrollers such as the Maxim MAX78000 offer significantly lower energy consumption. On the other hand, such controllers suffer from the fact that they do not offer support for CUDA: A model that can run on a PC or mainframe computer therefore requires adaptation before it can be used in the IoT using these chips.

On the other hand, the NVIDIA Jetson is not a fully-fledged GPU: Both in terms of energy consumption and the supported CUDA variant (CUDA 11 does not work), the module is not a fully-fledged replacement for an RTX 4000.

The bottom line is that a deployment pays off if a model that works smoothly on a computer or mainframe has to be mobilized with little effort and gets by with the resources offered by Jetson.

The availability of a Linux operating system means that comparatively little conversion work is required for a data scientist — familiarization with the embedded APIs used by Maxim & Co. requires considerably more man-hours. The higher costs of the hardware can thus be amortized quickly and efficiently, especially in smaller series. ◀

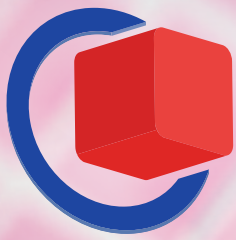
*Translated by Jörg Starkmuth — 230740-01*

### Questions or Comments?

Do you have questions or comments about this article? Email the author at [tamhan@tamoggemon.com](mailto:tamhan@tamoggemon.com), or contact Elektor at [editor@elektor.com](mailto:editor@elektor.com).

## WEB LINKS

- [1] Portfolio overview: <https://www.nvidia.com/en-eu/autonomous-machines/embedded-systems/>
- [2] List of various third-party products: <https://developer.nvidia.com/embedded/ecosystem>
- [3] OEMSecrets best price: <https://www.oemsecrets.com/compare/%20945-13450-0000-100>
- [4] Download of the image file: <https://developer.nvidia.com/jetson-nano-sd-card-image>
- [5] Open CV-based pipeline file: [https://github.com/JetsonHacksNano/CSI-Camera/blob/master/simple\\_camera.py](https://github.com/JetsonHacksNano/CSI-Camera/blob/master/simple_camera.py)
- [6] Ready-made examples: <https://github.com/NVIDIA/jetson-gpio/tree/master/samples>
- [7] Execution of GPIO program examples without superuser rights: <https://github.com/NVIDIA/jetson-gpio/issues/20>
- [8] List of supported products: [https://elinux.org/Jetson\\_Zoo](https://elinux.org/Jetson_Zoo)
- [9] Ready-to-use examples: <https://github.com/dusty-nv/jetson-inference>
- [10] Jetson AI courses and certificates: <https://developer.nvidia.com/embedded/learn/jetson-ai-certification-programs>



# embeddedworld

Exhibition & Conference



CONNECTING THE  
EMBEDDED COMMUNITY

9 – 11.4.2024



Get your  
free ticket now!

[embedded-world.de/codes](https://embedded-world.de/codes)

Use the voucher code **ew24ELE**

Media partners

**Markt & Technik**  
die unverzichtbare Wochenzeitschrift für Elektronik

**Elektronik**

**computer &  
automation**

**Elektronik**  
automotive

**Elektronik**  
•medical

**elektronik**net.de

NÜRNBERG / MESSE



# 2024 An AI Odyssey

## First Forays Into TensorFlow Lite

By Brian Tristam Williams (Elektor)

Exploring TensorFlow Lite's potential on the Raspberry Pi, this installment takes a look at the practicalities of AI in compact form, testing the limits of what's possible on lower-power devices.

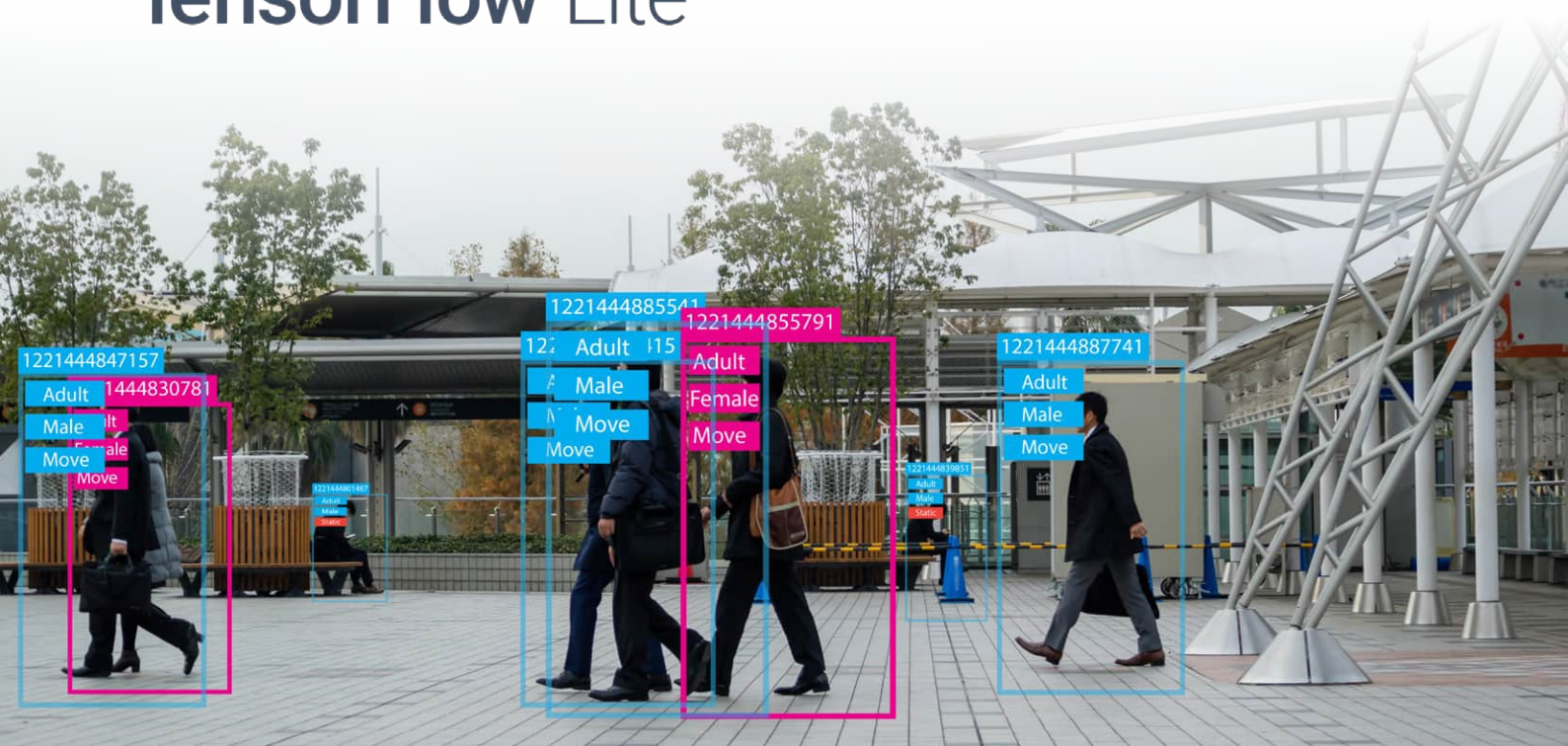


## TensorFlow Lite

### What Is TensorFlow?

Developed by the Google Brain team, TensorFlow is an open-source library for numerical computation and machine learning, playing a critical role in the broader AI ecosystem, which includes generative AI technologies such as ChatGPT. It's a cornerstone in deep learning, enabling the creation of models that can process and learn from "big data."

TensorFlow and generative AI models often complement each other, with TensorFlow providing the foundational platform for building and training a variety of AI models, including those that could power generative applications. Its versatility allows it to cater to a wide range of tasks from simple classifications to complex decision-making processes, making it a popular choice in both academic and industrial fields. Its ability to process large sets of data, recognize patterns, and learn from them positions it as a powerful tool in the broader context of generative AI, which focuses on creating new content and data models.





## TensorFlow Lite: Less Is More

In this next piece of our AI journey, I'm taking a somewhat unconventional route. While I have access to a reasonably capable PC equipped with a decent RTX 4070 GPU and enough RAM, I find myself drawn to the world of TensorFlow Lite [1] on the Raspberry Pi. There's a certain charm in the challenge of optimizing AI processes for such a compact, less powerful device. But it's not just about doing more with less; it's about creating smart, self-contained solutions that are portable and efficient.

Sometimes, the project at hand doesn't need, or can't accommodate, the bulk and power of a large desktop tower. In these scenarios, the Raspberry Pi offers a perfect alternative. Running TensorFlow Lite on this tiny yet capable machine helps make AI more accessible and adaptable to various environments, whether it be for educational purposes, hobbyist projects, or real-world applications where space and power consumption are at a premium.

With the rise of edge computing, the need for powerful yet efficient AI tools has become more pronounced. This is where TensorFlow Lite, a lighter and more efficient version of TensorFlow, comes into play, especially for devices such as the Raspberry Pi. The Raspberry Pi being a small, affordable, yet powerful computer provides the perfect platform for running TensorFlow Lite, bringing machine learning capabilities to the fingertips of hobbyists, educators, and professionals alike.

Putting TensorFlow Lite onto a small single-board computer offers significant advantages for AI applications on bigger machines. TensorFlow Lite is optimized for performance on lightweight hardware, making it suitable for a wide range of other practical applications beyond object detection. These include real-time data processing, local decision-making in IoT devices, and running AI models for tasks such as gesture recognition, environmental sensing, and health monitoring. This empowers Raspberry Pi makers to deploy complex AI models in a cost-effective, accessible manner, opening up a world of possibilities for innovation and exploration in AI, even on smaller-scale projects.

In the following sections, we'll look at how to install TensorFlow Lite on a Raspberry Pi and prepare ourselves for future innovative applications it unlocks.

## Starting Blocks

I have a drawer full of Raspberry Pis, from the O.G. to the Zero to the 5, but I'm going to use a Raspberry Pi 4 for this one. It's a good middle ground in terms of accessibility to our readers, sitting neatly between the Threes, which I still see on sale sometimes, and the new kid on the block, the Five. (Besides, I only have the one Raspberry Pi 5, so there's that.)

So, what I have as a starting point is:

- **Raspberry Pi 4 Model B:** We'll see how it gets by with its 4 GB of RAM. (I used `free -h` at the command line to remind me of how much memory is on board.)

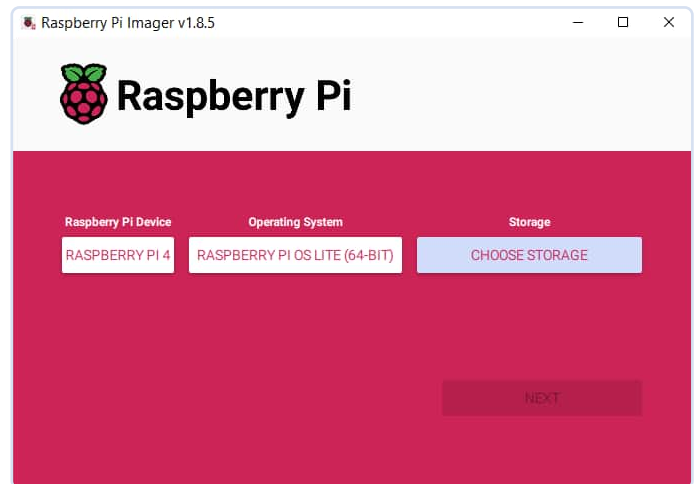


Figure 1: Raspberry Pi Imager has a neat, easy-to-use interface.

- **32 GB microSD card:** I don't know if I necessarily need that much, but it's the first one I found when I was scrounging for one.
- **Raspberry Pi OS Lite (64-bit):** I'm not really a fan of the overhead introduced by GUIs, and I prefer the yes/no on/off reassurance that I get from the text interface, so, this is the port of Debian Bookworm with no desktop environment. Having worked with web servers for a long time, I've never really found a use for a GUI in Linux. Let me know if I should get with the times!

I installed fresh from Raspberry Pi Imager v1.8.5, the version available at the time of writing from [2]. Simply download the version for your OS and run it. It's really great at simplifying the installation process. Just choose your Raspberry Pi board version, the operating system you want, and the card you have plugged into your computer's card reader. The user interface is really clean and simple (Figure 1). Even better, it works.

I verified my version number once installed, using `cat /etc/os-release`. Yours may be different, but this was my result:

```
briantw@raspberrypi:~$ cat /etc/os-release
PRETTY_NAME="Debian GNU/Linux 12 (bookworm)"
NAME="Debian GNU/Linux"
VERSION_ID="12"
VERSION="12 (bookworm)"
VERSION_CODENAME=bookworm
ID=debian
HOME_URL="https://www.debian.org/"
SUPPORT_URL="https://www.debian.org/support"
BUG_REPORT_URL="https://bugs.debian.org/"
```

As far as accessories go, I simply have a keyboard (no mouse needed — yay!), a Raspberry Pi 4 power supply, and both Micro HDMI outputs connected (Figure 2). Currently, one goes to a great little portable monitor and the other to a capture device on my PC, which, for now, I'm using for some screenshots. The eventual idea is to have my code running on one screen and the video output — for example from an object detection script — visible on the other.

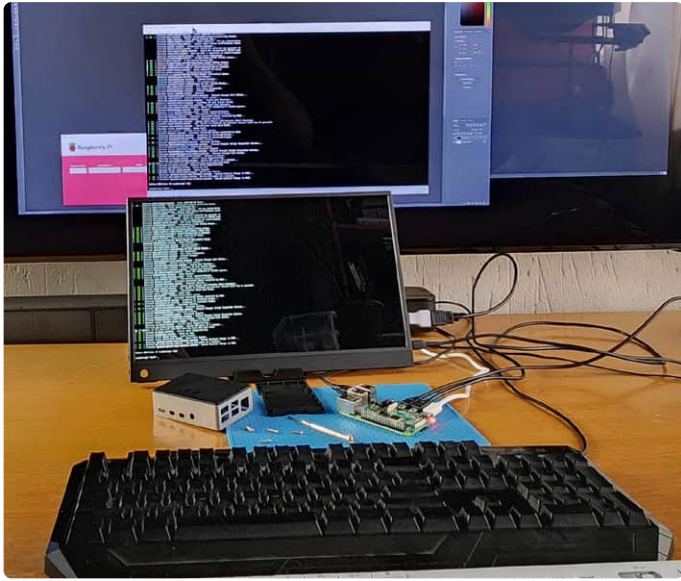


Figure 2: The basic Raspberry Pi setup I'm using for this inquiry.

## TensorFlow Lite Installation

Now that we have our starting point, let's get TensorFlow Lite running on the Raspberry Pi.

Firstly, some housekeeping. Let's run an update/upgrade cycle on the Raspberry Pi, just to make sure we have all the latest iterations of everything that's installed. Start with:

```
sudo apt-get update
```

How much you have to wait for depends on how far behind you are from the latest updates. I only had 12 items update. Next:

```
sudo apt-get upgrade
```

For me, this resulted in an entire screen full of upgrades, which always surprises me a little when I've just installed the latest official OS offering. Have patience with the process — your mileage may vary.

As for the installation of TensorFlow Lite, I did a dive into various Google search results, and the most approachable guide I found was by EdgeElectronics over on GitHub [3]. Naturally, some time had passed since Edge's initial outline of the process and new OS releases and upgrades, so I hit a couple of minor snags, which I'll document here for you.

The first step was to clone the GitHub repository using:

```
git clone https://github.com/EdgeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi.git
```

My Pi's encouraging response:

```
-bash: git: command not found
```

Oh yeah, it's a fresh installation. Oops. Install *git* using:

```
sudo apt install git
```

That worked for me, so now that we have *git* installed, back to the *git clone...* command from above. If successful, you'll end up with a subdirectory within your home directory called *TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi*. That's a little unwieldy, so rename it to *tflite1*, using the *mv* (move) command:

```
mv TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi tflite1
```

Then, enter that directory using:

```
cd tflite1
```

EdgeElectronics suggests creating a virtual environment, so we'll follow along with:

```
sudo pip3 install virtualenv
```

Response from the Pi, almost predictably:

```
-bash: pip3: command not found
```

Installing *pip3* is solved almost as easily as it was for *git*. I first did another *sudo apt-get update*, where nine things were updated, and then this:

```
sudo apt-get install python3-pip
```

A lot of installation happened there, resulting in another screenful of text.

Next was to retry the *sudo pip3...* command. Debian didn't like it. After a bit of digging, I found that I had to create my own virtual environment. Assuming that you're still in the *tflite1* directory, use:

```
python3 -m venv tflite1-env
```

Then, activate this virtual environment:

```
source tflite1-env/bin/activate
```

Now to install the TensorFlow Lite dependencies and OpenCV. As discussed earlier, TensorFlow Lite has many areas of application, but this installation will prepare it for a very popular area: machine vision and object recognition. So, OpenCV, an open-source library designed for computer vision and machine learning tasks, will be handy.

Fortunately, you won't need to *curl*, *wget*, or *git clone* anything here, because the repo creator made a nice 40-line shell script for us called *get\_pi\_requirements.sh*. If you're curious to see what the script

is going to do before running it, you can view its contents by executing a `cat get_pi_requirements.sh`, but we'll run it by entering:

```
bash get_pi_requirements.sh
```

I did make the mistake of trying this after having rebooted the Raspberry Pi, and it threw out some error messages and warnings. Reason? That source command that we executed earlier while in the `tfliet` directory has to be run every time you reboot or start a new Terminal session. Also, this took a few minutes to run, so have patience, again!

Now, we get the option to use a Google sample TensorFlow Lite model, or to use one we trained ourselves. As a first-timer, I went with getting Google's:

```
wget https://storage.googleapis.com/download.tensorflow.org/models/tflite/coco_ssd_mobilenet_v1_1.0_quant_2018_06_29.zip
```

That's a long line, which won't fit here without a line break, but just note that the only space you need to type is the one after `wget` — there are none from `https` onward.

Next, unzip what was just downloaded:

```
unzip coco_ssd_mobilenet_v1_1.0_quant_2018_06_29.zip -d Sample_TFLite_model
```

And that's it, we've installed everything we need to start playing with object detection and image classification using TensorFlow Lite on the Raspberry Pi.

Now we can start experimenting.

In our next installment: As a data hoarder, particularly of historical images and video that is often not available anywhere else, I want to share what I have available with the world. But, I will never have the time to go through each still image and each frame of video to add metadata and tell the search engines what I've got. So, my intention is to use these tools to help me analyze and classify my mountain of media. That phase starts now, and I will update you on my successes and failures next time! 📺

230181-E-01

## Questions or Comments?

Do you have technical questions or comments about this article? Email the author at [brian.williams@elektor.com](mailto:brian.williams@elektor.com).



## About the Author

Brian Tristam Williams has been fascinated with computers and electronics since he got his first "microcomputer" at age 10. His journey with Elektor Magazine began when he bought his first issue at 16, and since then, he's been following the world of electronics and computers, constantly exploring and learning. He started working at Elektor in 2010, and nowadays, he's keen on keeping up with the newest trends in tech, particularly focusing on artificial intelligence and single-board computers such as Raspberry Pi.



## Related Products

➤ **Raspberry Pi 4B (2 GB RAM)**  
[www.elektor.com/18965](http://www.elektor.com/18965)



## WEB LINKS

- [1] TensorFlow Lite official website: <https://tensorflow.org/lite>
- [2] Raspberry Pi Imager Download: <https://raspberrypi.com/software>
- [3] TensorFlow Lite Object Detection on Android and Raspberry Pi [GitHub]: <https://tinyurl.com/edjetflite>



# 262,144 Ways to Play The Game of Life

A Reader's Project in Brief

By Brian White (United States)

I've had a long-standing fascination with Conway's Game of Life. Follow me in my journey of turning this 1970s programming puzzle into an enthralling visual spectacle, using an RGB LED matrix coupled with unique coding methods that allow us to simulate every conceivable rule set for the game!

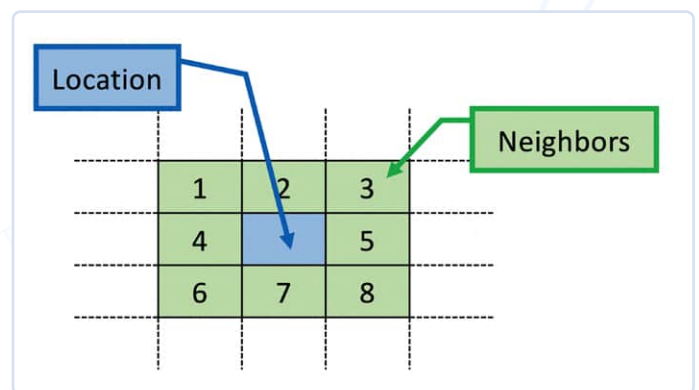


Figure 1: A location in the "world" and its eight neighboring locations.

Conway's rules are:

- A cell shall persist into the next generation if it has two or three neighbors. Cells with fewer neighbors "die of loneliness;" cells with more neighbors "die of overcrowding."
- A cell can be born into an empty location in the next generation if that cell has exactly three neighbors.

Conway selected these rules after a period of experimentation to meet the following criteria (as described in [2]):

- There should be no initial pattern for which there is a simple proof that the population can grow without limit.
- There should be initial patterns that apparently do grow without limit.
- There should be simple initial patterns that grow and change for a considerable period of time before coming to end in three possible ways: fading away completely (from overcrowding or becoming too sparse), settling into a stable configuration that remains unchanged thereafter, or entering an oscillating phase in which they repeat an endless cycle of two or more periods.

In the original 1970 version, the game was played with physical checkers on a checkerboard. Soon after, the process was computerized and innovations continue to this day [3].

This project is a story of many pieces, some of which have been circulating in my brain for 50 years, finally coming together in a very satisfying conclusion. It begins with Conway's Game of Life, which I first heard of in the late 1970s. At the time, I was taking my first (and only) programming course — BASIC programming using a DECwriter terminal connected to a PDP-11 at my high school — and writing code for "The Game of Life" was one of the exercises. It's a fun example of nested loops (`FOR...NEXT` in BASIC), a great beginning programming exercise, and it produced fascinating changing patterns. In the years that followed, it continued to fascinate me as computing — and thus Conway's game — moved from printing terminals to CRT monitors to laptops and iPads.

## The Game of Life

In brief, Conway's Game of Life [1] (GoL) is a simple rule-based cellular automata simulation. It is played on a rectangular matrix of arbitrary size. The playing pieces are cells that occupy one location each in the matrix. In the game, cells can persist, die, or be born from one generation to the next. The presence or absence of a given cell in the next generation depends only on how many neighbors that location has in the current generation. In a rectangular matrix, each location has between zero and eight neighbors (**Figure 1**).

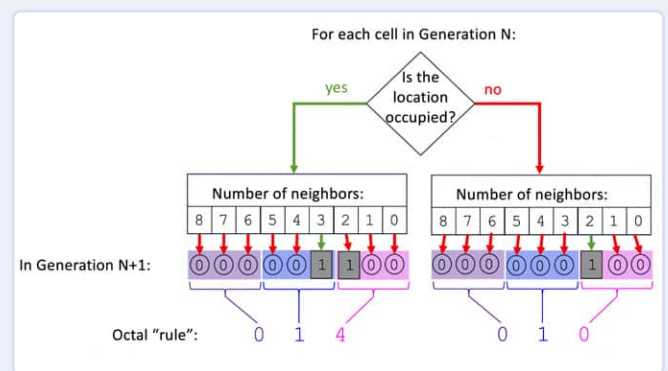
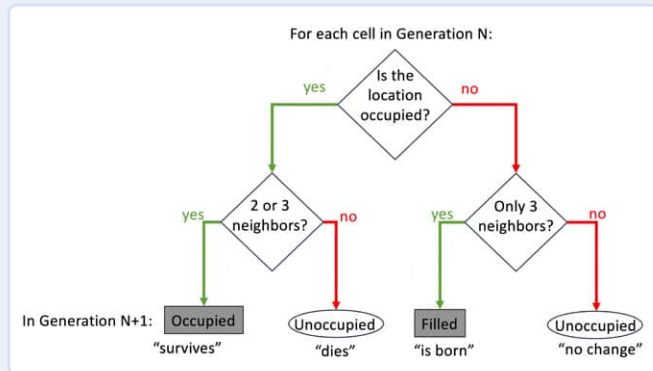


## Encoding the Rules in Six Octal Digits

The original rules of the Game of Life can be described in brief:

A cell survives if it has 2 or 3 neighbors, otherwise, it dies.  
A cell will be born into an empty location with exactly three neighbors.

These same rules can also be shown like this, where a “1” in the rule indicates an occupied location in the next generation (survival or birth) for that number of neighbors and a “0” in the rule indicates an unoccupied location in the next generation (death or remaining empty).



The bit pattern specifies the state of the cell in the next generation; the index into the bit pattern is the number of neighbors combined with whether the cell is currently occupied or not. Shown below is the decoding of the rule 256020 as an example:

Table 1: Interpretation of the octal rule set “256020”

Original location	Occupied									Unoccupied								
Neighbors	8	7	6	5	4	3	2	1	0	8	7	6	5	4	3	2	1	0
Next Gen	0	1	0	1	0	1	1	1	0	0	0	0	0	1	0	0	0	0
Octal	2			5			6			0			2			0		

Translated into text, this would become:

Currently, living cells survive if they have 1, 2, 3, 5, or 7 neighbors; otherwise, they die.  
A cell will be born into each currently unoccupied location that has exactly 4 neighbors.

Using this encoding format, any of the 262,244 possible rule sets for the Game of Life can be specified.

## Interesting Algorithms

The next piece of the puzzle was the book “A New Kind of Science” by Stephen Wolfram, where he discussed various cellular automata algorithms that produced very interesting visual patterns. One of the key parts of Wolfram’s analysis was to encode the rules for the generation-to-generation transitions as binary numbers. By encoding the rules this way, it is possible to explore the consequences of all possible rule sets in a well-defined and repeatable way [4].

This got me thinking about a way to encode the rules of GoL so that it would be possible to explore all possible rule sets. I realized that you could encode the rules in an 18-bit binary number (see the **Encoding the Rules in Six Octal Digits** text frame for details). Each of the low-order nine bits (0 to 8) gives the state of

the location in the next generation (1 = occupied; 0 = unoccupied) for an unoccupied location with zero to eight neighbors — bit 0 gives the state in the next generation if the location currently has zero neighbors; bit 1 for one neighbor, etc. Similarly, the high order nine bits (9 to 17) give the state of the next generation for an occupied location with zero to eight neighbors — bit 9 for zero neighbors; bit 10 for one neighbor, etc. Using this encoding, it is possible to specify any set of rules that are based on the number of neighbors a location has.

Interestingly, there is no need for these rules to “make biological sense.” For example, rule set 256020, where cells persist if they have 1, 2, 3, 5, or 7 neighbors. This rule set gives interesting behavior even though there’s no biological reason why 4, 6, and 8 neighbors would be overcrowded but 3, 5, and 7 would not.

Since all possible rule sets can be encoded in 18 binary bits and  $2^{18} = 262,144$ , this means there are 262,144 ways to play the GoL. This makes it possible to build a device that would allow a user to explore all of these possibilities and observe how a population of each different “species” would develop over time.

The last piece of the software design came from many conversations with the electronic artist, Kelly Heaton [5], whom I met through an article in *Elektor Magazine* [6]. She encouraged me to think beyond the canonical and make works that are both true to the algorithm and visually compelling. This led me to modify the standard coloring scheme for GoL simulations, where occupied locations are colored and unoccupied locations are black. I wanted to give the patterns a stronger sense of change, so I colored newborn cells blue; cells that persist for more than one generation are green; and, cells that die leave a deep purple “ghost” for one generation (the “ghosts” are not counted as neighbors). This is detailed in **Figure 2**.

### Hardware Assembly

The last piece, the hardware, arrived as a Christmas present from my sons: an Adafruit Matrix Portal [7] and a 32×64 RGB LED matrix [8]. I then assembled all the pieces in an acrylic box to make all the internals visible. The final product is shown in **Figure 3**. On the left are three sets of three thumb wheel switches that set the rules and the parameters for the run. The top and middle sets of three switches set the rules as six octal digits — each octal digit

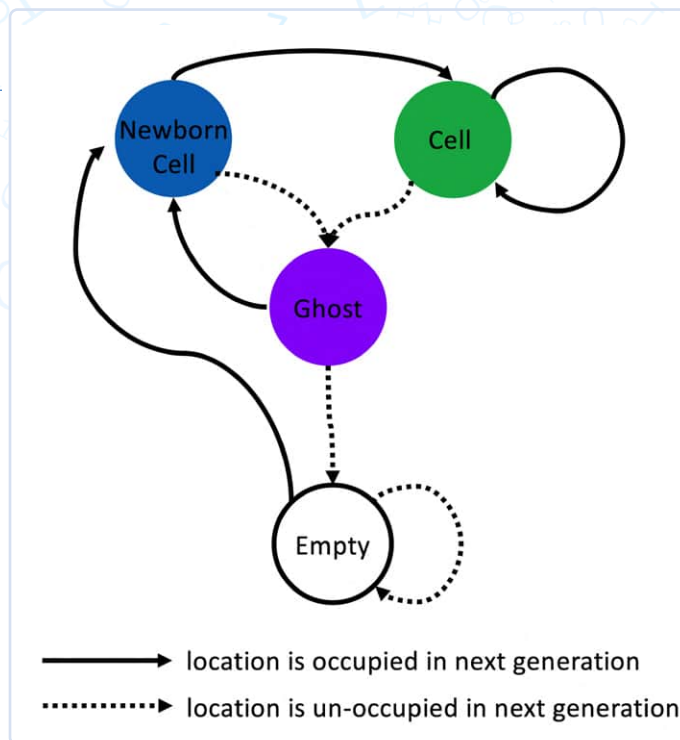


Figure 2: State diagram for multicolored cell pattern.

is 3 binary bits; multiplied by 6 digits gives the required 18 binary bits. The top three digits specify the rules for occupied locations; thus, the rules for which cells survive to the next generation. The middle three digits specify the rules for unoccupied locations; thus, the rules for birth of new cells. As an example, Conway’s canonical rules would be expressed as 014010. The **Encoding the Rules in Six Octal Digits** text frame describes this in more detail.

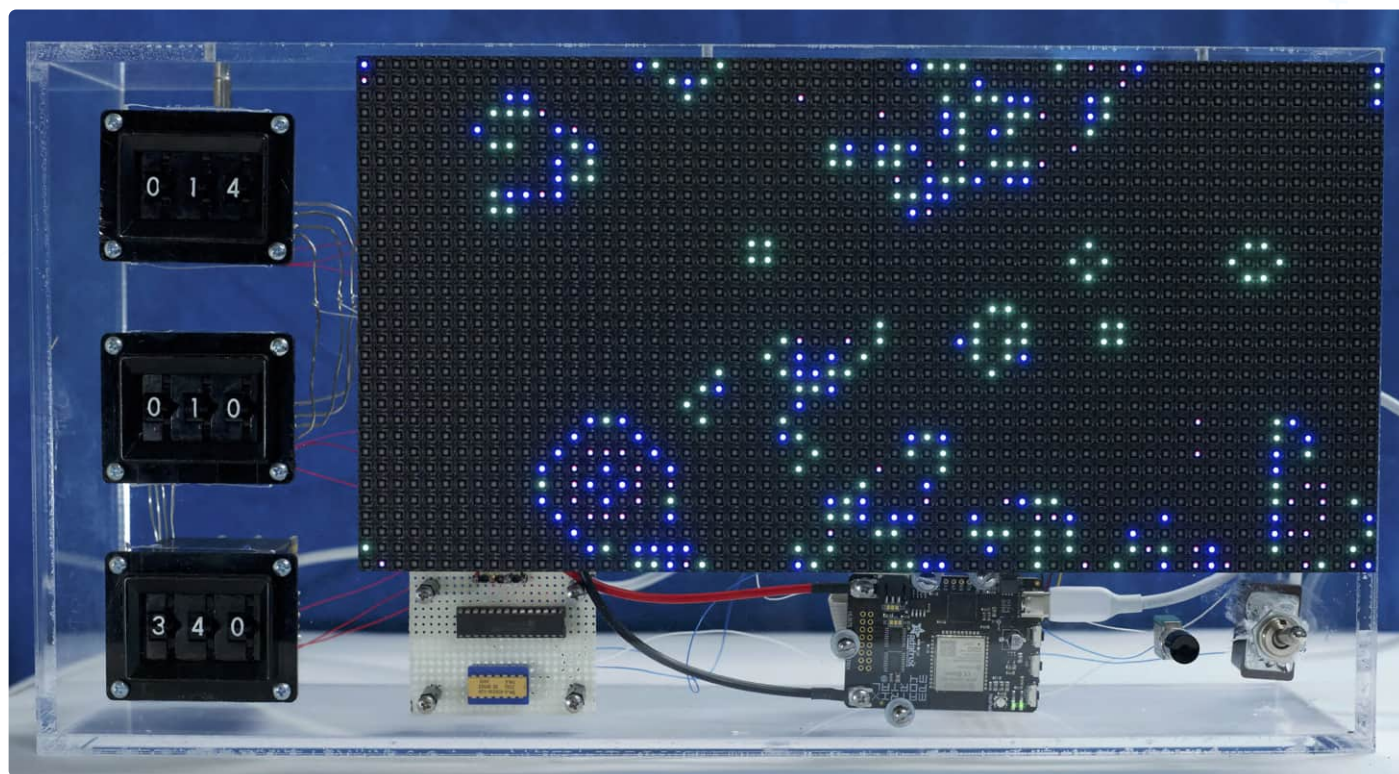


Figure 3: Demo of the assembled project.

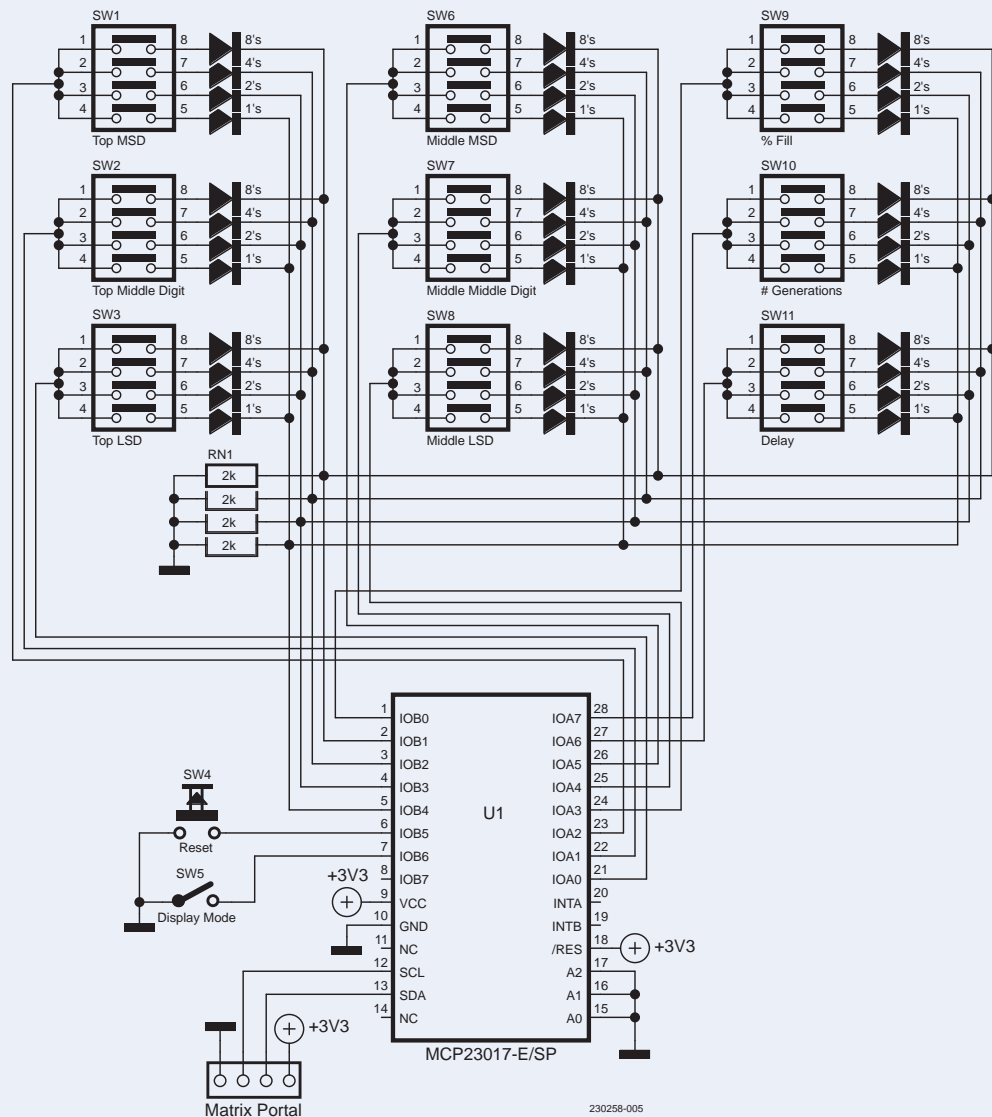


Figure 4: Schematic excluding Matrix Portal, 64x32 display, and USB power.

Below the display, moving from left to right, are the MCP23017 port expander board, the Matrix Portal MCU board, the reset pushbutton, and the display mode toggle switch.

Each run starts with a random arrangement of occupied and unoccupied locations and then proceeds for a fixed number of generations before restarting with a new random field of cells. The bottom three digits specify the details of this process. The left digit specifies the density of organisms in the initial population; specifically, the chance of any location being occupied at the start is 10% times the setting of this thumb wheel. Thus, a setting of “3” means the world starts roughly 30% filled with cells. Since some rule sets give more interesting results with more or fewer occupied locations, this allows exploration of the effects of varying initial cell density. The center digit specifies the number of generations to run before restarting. For this, I chose an exponential scale to allow a wide range of run lengths. Specifically, it runs  $2 \times 10^{(N/2)}$  generations; thus, a setting of 4 would run for 200 generations. Some patterns resolve quickly, while others never resolve; this control helps to keep the display in an interesting state. Finally, the rightmost digit specifies the delay in seconds between generations;

thus, a setting of 0 runs at maximum speed. Slowing down the generations makes it possible to follow the rules in detail if desired. The last two controls are a pushbutton that immediately re-starts the run and re-randomizes the population and a toggle switch to select standard coloration (blue = occupied; blank = unoccupied) or my multicolor coloration scheme for the locations in the matrix.

### Building It Yourself

The schematic in **Figure 4** is very simple. The Matrix Portal and display are connected as described in Adafruit’s instructions for the MCU and the LED matrix. The additions shown get their +3.3 V power and I<sup>2</sup>C communication from the Matrix Portal via its Stemma QT connector. All the heavy lifting is accomplished by an MCP23017 I<sup>2</sup>C Port Expander. Individual BCD switches (SW1...SW3, SW6...SW11) are individually enabled by outputs of the MCP23017, and the BCD outputs of the enabled switch are then read in. Because the MCP23017 does not have a weak pull-down option, the BCD lines are pulled down to ground by a resistor array. The last two inputs of the MCP23017 are used with weak pull-ups to read the reset and display mode switches. Power is provided by a 5 V, 4 A “wall wart” that feeds the Matrix Portal via a USB-C cable.



## Component List

Adafruit Matrix Portal – CircuitPython Powered Internet Display  
(Adafruit 4745)  
64x32 RGB LED Matrix - 5mm pitch (Adafruit 2277)  
3 x 3-digit BCD-encoded thumbwheel switch (example:  
Littlefuse 3P-3-23-3-1-0-2)  
MCP23017 port expander  
4 x 2K pull-up resistors  
Pushbutton switch  
SPST toggle switch  
USB power adapter  
Acrylic box

The code is based on GoL code provided by Adafruit for the Matrix Portal and Display [9] which uses some very clever tricks to make the loops-within-loops code that GoL requires run very quickly. I modified the code to implement the rules based on the numbers entered on the thumb wheel switches instead of the “standard” rules. Briefly, I read in the binary values of the thumb wheels and use that to create an 18-element list of 1s and 0s representing the state of the location in the next generation, depending on the location's current occupancy and number of non-empty and non-ghost neighbors. I then count the neighbors and use that count, with 9 added if the location is currently occupied, and use that as an index into the rule list. I also implement the multicolor pattern described above. The code is available from the Elektor Labs project page [10].

## Going Further

The results have been fascinating. I've created a YouTube playlist [11] with a collection of short videos showing the patterns that emerge with different settings of the switches. It is remarkable how some changes to the rules have a large effect while others make only subtle changes in how the patterns grow, change, move, and die out. I've only begun to survey the possibilities (262,144 is a rather large number!) but here are some things that I've noticed:

- Setting the lowest order bits (bits 0 and 9) — thus allowing births and/or survival with zero neighbors — leads to dramatic positive-negative reversals in each generation.
- Setting higher-order bits — controlling births and survival with higher numbers of neighbors tends to have less of an effect, perhaps because locations with many neighbors are rarer.
- The original rule set, 014010, usually evolves into some relatively stable structures or oscillating figures. Other rules, such as 114110, are favorites of my wife since they never seem to resolve.
- You can create rules where structures are created and gradually fill in interesting ways if you allow survival for many neighbor counts and births in only a few; for example, 256020.

Right now, it lives by our kitchen table and I often find that someone in the house has found a new rule set that gives a fascinating new pattern.

As we have lived with this piece and shared it with others, we've had several ideas for those who might want to build one of their own. First, it would be interesting to add more colors to the multicolor scheme. For example, the colors could change gradually through more hues as the cells “mature.” Kelly Heaton further suggested having the colors fade one into the other rather than changing abruptly to give a more flowing effect. Finally, the non-octal-speaking members of my household suggest having a way to show the correspondence between the bits in the rules and the numbers of neighbors more explicitly. One could add a row of 18 LEDs to display the binary equivalent of the octal settings or, even simpler, use 18 individual switches, one for each bit in the rule settings.

I hope that readers of this article will be inspired to build a version of this project, either with a dedicated display and MCU, or entirely on computer, and share the rule sets that they find interesting. Who knew that sharing 6-digit octal numbers could be so interesting? ◀

230258-01



## About the Author

Brian White is a professor in the Biology Department at the University of Massachusetts, Boston. He has a background in Biology from MIT and Stanford. As a teacher, he also conducts research in Biology and Biology Education and develops related software. Brian is also an electronics enthusiast, a musician, and a radio amateur (KA1TBQ). Further details on his electronics projects are available at [12].

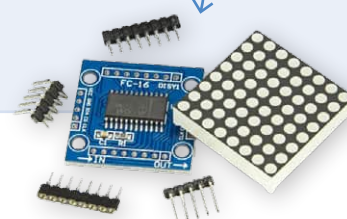
## Questions or Comments?

If you have technical questions or comments about this article, feel free to contact the author by email at [brian.white@umb.edu](mailto:brian.white@umb.edu) or the Elektor editorial team at [editor@elektor.com](mailto:editor@elektor.com).



## Related Products

- **MAX7219 Dot Matrix Module (Set of 8)**  
[www.elektor.com/18422](http://www.elektor.com/18422)
- **Adafruit Feather RP2040**  
[www.elektor.com/19689](http://www.elektor.com/19689)
- **ESP32-C3-DevKitM-1**  
[www.elektor.com/20324](http://www.elektor.com/20324)





## WEB LINKS

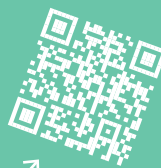
- [1] Wikipedia: Conway's Game of Life: [https://en.wikipedia.org/wiki/Conway%27s\\_Game\\_of\\_Life](https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life)
- [2] Martin Gardner, "MATHEMATICAL GAMES: The fantastic combinations of John Conway's new solitaire game 'life,'" Scientific American 223 (October 1970): 120-123: <https://web.stanford.edu/class/sts145/Library/life.pdf>
- [3] LifeWiki: <https://conwaylife.com/wiki>
- [4] Stephen Wolfram, A New Kind of Science, see especially this section: <https://wolframscience.com/nks/p53--more-cellular-automata>
- [5] Kelly Heaton Studio: <https://kellyheatonstudio.com>
- [6] C.J. Abate, "Making Art with Electricity: A Q&A With Kelly Heaton," Elektor Circuit Special 2022: <https://elektormagazine.com/articles/making-art-with-electricity-a-q-a-with-kelly-heaton>
- [7] Matrix Portal — CircuitPython-Powered Internet Display: <https://adafruit.com/product/4745>
- [8] 64x32 RGB LED Matrix — 5 mm pitch: <https://adafruit.com/product/2277>
- [9] Adafruit Example: Conway's "Game of Life": <https://learn.adafruit.com/rgb-led-matrices-matrix-panels-with-circuitpython/example-conways-game-of-life>
- [10] Project page on Elektor Labs: <https://elektormagazine.com/labs/262144-ways-to-play-the-game-of-life>
- [11] Game of Life videos: [https://youtube.com/playlist?list=PL2wCLlpn1MkRpBHeZ2svMs-CdLv2B\\_9N](https://youtube.com/playlist?list=PL2wCLlpn1MkRpBHeZ2svMs-CdLv2B_9N)
- [12] The author's website: <https://brianwhite94.wixsite.com/electronics>

## Ignite Your Electronics Innovations with

# ElektorLabs

- Free Project Sharing
- Expert Support
- Collaboration Opportunities
- Access to Exclusive Resources
- Get published in Elektor Magazine

**Share Your Projects Now!**  
[www.elektormagazine.com/e-labs](http://www.elektormagazine.com/e-labs)



**elektor**  
design > share > learn



# From Life's Experience

## The Chinese Dragon

By Ilse Joostens (Belgium)

Fine Chinese Kraak porcelain with blue and white motifs was popular in the Netherlands and had been imported since the beginning of the 17th century by the Dutch East India Company, which also had a location in Delft. This porcelain was mainly used as a decorative item in wealthy households, and when the import ceased after the fall of the Ming Dynasty in 1644, the Delft pottery factories took over the production with cheaper copies in earthenware, initially featuring typical Ming-like motifs on white tin glaze. Well, 380 years later, the roles seem reversed and now it is China that copies Western products.

On the morning of October 12, 1654, Cornelis Soetens inadvertently destroyed a large part of the city of Delft when he went to collect a sample of gunpowder from the gunpowder depot known as the "Secret van Holland." The Delft thunderclap and the city's reconstruction kick-started rapid development in the already flourishing earthenware industry, and Delftware remains globally recognized to this day. The depot was rebuilt, but well outside the city walls.

### Need a Copy?

Of course, Delftware [1] was more than just a copy of Chinese porcelain, and the Delft

ceramic painters not only took the quality of their products to great heights but also consistently surprised with new techniques. On the other hand, China copies the West very literally — the research campus opened in 2019 by Chinese technology company Huawei in Dongguan looks like a fake Europe [2]. The train that stops at stations such as Heidelberg, Bologna, and Granada clearly resembles that of the Swiss Jungfrau Railway. Earlier, parts of Paris, such as the Champs-Élysées, and the picturesque Austrian village of Hallstatt [3] were recreated as tourist attractions. Unfortunately, the copying goes much further, from all kinds of products, including artworks. Chinese

stores replicate the look, feel, and service of successful Western retail concepts, such as a well-known Swedish home furnishing store or stores of a computer brand with a half-eaten fruit as a logo.

These stores were shut down after some legal wrangling, but the copying of products and ideas continues. Not long ago, I was almost knocked off my chair when I saw a two-meter-high replica of Kindred Spirits sculpture by Alex Pentek [4] at Uncle Ali's. The electronics sector is not spared either. Many are familiar with counterfeit ICs and recycled components sold as new. In addition to dubious parts, you can order all kinds of electronic modules, kits, gadgets, tools, and machines of all sizes at very low prices. Even niche products are targeted, and Dalibor Farný, the Czech maker of the R|Z568M Nixie tubes, now certainly fears formidable competition.

In 2016, I wrote an article for Elektor about 7-segment displays [5], based on LED filaments (then still relatively new). Apparently, you can now order kits for clocks with LED filaments from Uncle Ali at prices for which I can't even gather the components. Shipping costs are also minimal because China, with the Universal Postal Union status of a developing country, can ship much cheaper [6]. It seems that the organization missed the part where China successfully landed a spacecraft on the far side of the moon. I don't mind an idea being used, but using your own idea to compete you out of the market is painful.

### Tofu Dregs

As in many societies, corruption, bribery, lying, and cheating exist at all layers in China, even

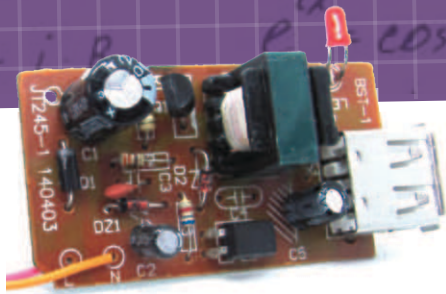


Figure 1: Cheap power adapter, component side...

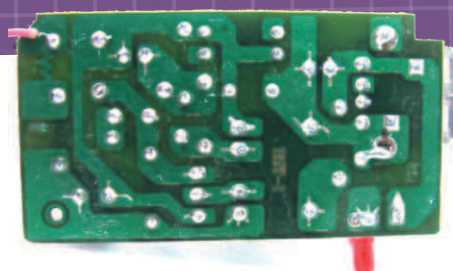


Figure 2: ...and copper side (figures 1 and 2: photo's by Leo Potjewijd).



Figure 3: The light inside from the Chinese-made salt lamp.

though the majority of its citizens are friendly and upstanding. The problems these underhanded machinations bring are not minor, and include buildings constructed from inferior materials with frequent collapses. Named after the remains left over after making tofu, "tofu dregs" is also slang for poorly executed projects and shoddy work [7]. In addition to dangerous infrastructure and even counterfeit food, the electronics sector also struggles with these problems, and many Chinese consumer goods imported and sold here are not only of poor quality but sometimes even life-threatening.

A constant are cheap power adapters and USB chargers built with a minimal number of components. Any insulation distance between

their primary and secondary sections is almost nonexistent, and a thin PCB trace serves as a fuse. It's not surprising that someone occasionally gets electrocuted in the bath by a smartphone. It seems a trivial matter in the media, and apart from a general warning not to use your smartphone connected to the charger, no attention is paid to the reasons why. In a forum recently, I saw another photo of the insides of such a power adapter (**Figure 1** and **Figure 2**). Besides the aforementioned defects, the optocoupler on the board serves no function if you look closely at the printed circuit. So, an unnecessary component, guys and girls...

I also wasn't happy with the salt lamp I received as a Christmas gift. It was dimma-

ble but flickered annoyingly when the dimmer control wasn't at maximum, and I ended up taking everything apart and revising it. The internal LED light (**Figure 3**) consisted of nothing more than a bridge rectifier, an IC that became scorching hot, and an LED matrix — not even a smoothing capacitor. The housing of the dimmer (**Figure 4**) could be disassembled without tools, and the power cord was nothing more than copper-plated aluminum with a cross-sectional area of less than a tenth of a square millimeter per conductor (**Figure 5**). Combined with a fake Chinese circuit breaker [8], great pyrotechnic effects are almost guaranteed (**Figure 6**). It's not always easy, but buying more locally produced goods and giving our own people a chance doesn't seem like the worst idea ever. ◀

230609-01



Figure 4: The salt lamp's dimmer.

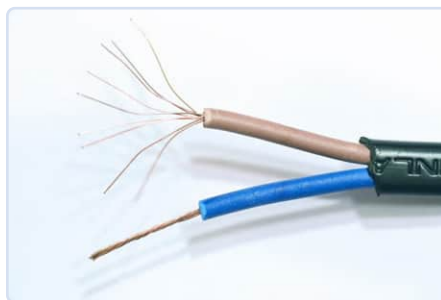


Figure 5: The salt lamp's frighteningly thin power cord...

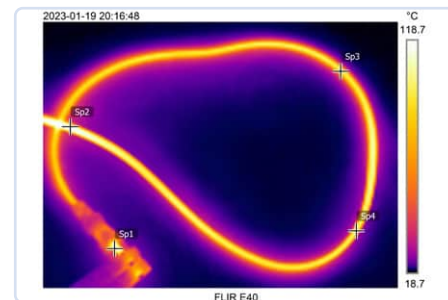


Figure 6: ...got very frighteningly hot at 6 A (almost 118°C at Sp2).

## WEB LINKS

- [1] Wikipedia: Delftware: <https://en.wikipedia.org/wiki/Delftware>
- [2] The Atlantic, "Photos of Huawei's European-Themed Campus in China": <https://tinyurl.com/atlanticoxhorn>
- [3] Wikipedia: Hallstatt (China): [https://en.wikipedia.org/wiki/Hallstatt\\_\(China\)](https://en.wikipedia.org/wiki/Hallstatt_(China))
- [4] Wikipedia: Kindred Spirits (sculpture): [https://en.wikipedia.org/wiki/Kindred\\_Spirits\\_\(sculpture\)](https://en.wikipedia.org/wiki/Kindred_Spirits_(sculpture))
- [5] Peter S'heeren, "LEDitron," Elektor 5/2016: <https://elektormagazine.com/magazine/elektor-201609/39800>
- [6] RTL News: "How AliExpress Can Ship Packages So Cheaply" [Dutch]: <https://tinyurl.com/rtlaliexpress>
- [7] China Insights, "Fragile steel bars/Tofu-dreg project in China/Shaky building/Collapsing buildings/Poor quality" [YouTube]: <https://youtu.be/s-2DtL-Wjkc>
- [8] bigclivedotcom, "Inside a fake un-trippable circuit breaker" [YouTube]: <https://youtu.be/2TJEzdtXlQ>



# Get Your (Brushed DC) Motor Running!

## Sample Projects from the Elektor Motor Control Development Bundle

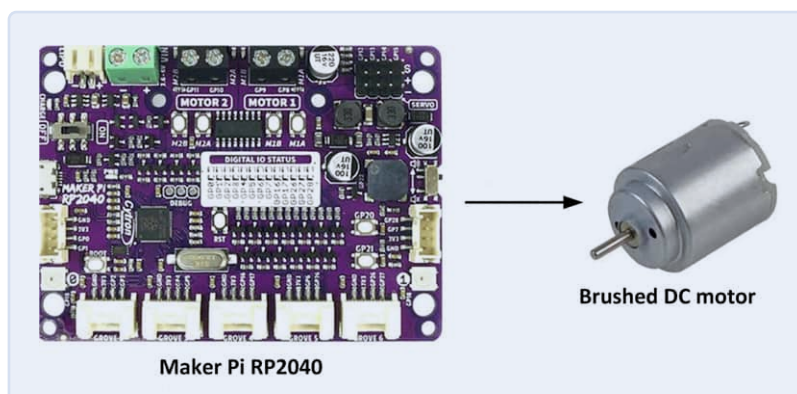
By Dogan Ibrahim (United Kingdom)

There's no better introduction to programming and practical electronics than to study how things can be made to move, buzz, or flash. Movement often involves some kind of motor, and a small, brushed DC motor is just great to learn a thing or two before embarking on serious robotics stuff and mechatronics. Here, we take the first steps toward intelligent DC motor control, aided by a fantastic development kit from Elektor that combines hardware, software, and a solid guidebook.

Figure 1: The two main elements of the project.



**Editor's Note.** *This article is an excerpt from the 192-page Elektor Guide: Motor Control Development Kit. The Guide is part of Elektor's Motor Control Development Bundle. This excerpt from the Guide was formatted and lightly edited to match Elektor Magazine's conventions and page layout. The author and editor are happy to help with queries. Contact details are in the Questions or Comments? box.*



In this article, three simple DC motor projects are described based on a small, brushed, permanent-magnet, DC motor driven by the Cytron/Maker Pi RP2040 development board. Both items are contained in the Motor Control Development Bundle available from the Elektor Store [1]. The projects are exploratory and educational rather than fully-engineered examples.

### DC Motor On/Off Control

This is a basic project showing how a small, brushed, DC motor operating at 1...6 V DC is connected to one of the Maker Pi RP2040's DC MOTOR terminal blocks. The motor is turned ON in forward direction for 5 seconds and then stopped for 5 seconds. Next, it is run in reverse direction for 5 seconds and stopped again for 5 seconds. This process is repeated forever until halted manually.

The aim of the project is to show the "bare metal" of brushed DC motor connection, operation, and control from the Maker Pi RP2040 dev board. **Figure 1** shows the "block diagram" of the project, and **Figure 2** the actual motor used. It's a small, brushed, DC motor designed to operate at +1 V to +6 V with a recommended operating voltage of +3 V. Let's see how to make it spin.

A dual-channel H-bridge motor driver, controlling up to two brushed DC motors, M1 and M2, or one stepper motor, is provided on the dev board. The I/O pin allocation is

- > M1A: GP8
- > M1B: GP9
- > M2A: GP10
- > M2B: GP11



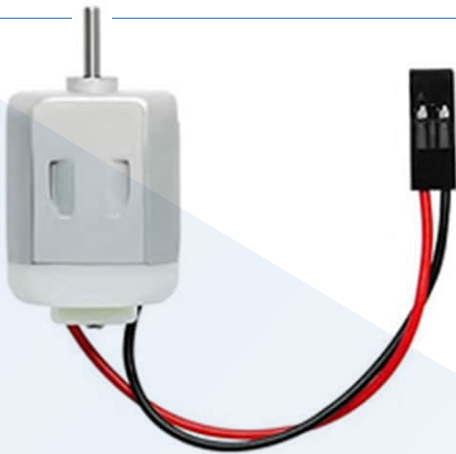


Figure 2: The little DC motor used.

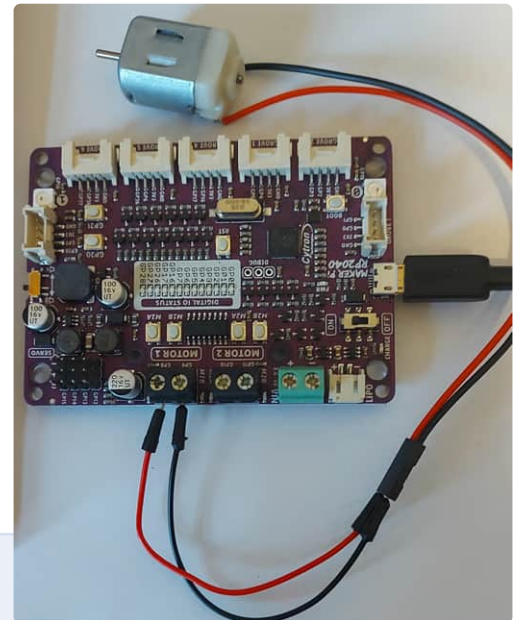


Figure 3: Connecting the motor to the RP2040 Maker Pi dev board.

The motor driver truth table is shown in **Table 1**. In this project, motor inputs PWM1A (port GP8) and PWM1B (port GP9) are used with the motor outputs on M1A and M1B. These outputs (*Output A*) are available on the MOTOR 1 screw terminals located at the middle top part of the development board. As shown in Table 1, the motor is controlled from ports GP8 and GP9. For example, setting GP8 HIGH and GP9 LOW rotates the motor forward. Similarly, setting GP8 LOW and GP9 HIGH rotates the motor backward.

**Figure 3** shows the project with the motor connected to MOTOR 1 outputs.

The program listing for the get-you-going demo is given in **Listing 1**. The program called *DCMotor1.py* is contained in a large software archive file available for free on the Elektor website [1]. On that page, look under *Downloads Software\_Motor Control Development Kit*.

The motor is controlled using PWM waveforms as described elsewhere in the *Guide* referred to. The built-in DC motor library called *adafruit\_motor* is used in this project. At the beginning of the program, library modules *board*, *time*, *pwmio* and *motor* are imported to the program. Motor 1 ports GP8 and GP9 are then configured to operate with PWM waveform at a frequency of 50 Hz. The remainder of the program runs in an endless loop.

The *throttle* class property in the can take a value between  $-1$  and  $+1$  and controls the motor as follows:

*throttle* = 0 motor idle  
*throttle* = 1 motor rotates forward at full speed  
*throttle* = 0.5 motor rotates forward at 50% of its full speed  
*throttle* = -1 motor rotates backward at full speed  
*throttle* = -0.5m motor rotates backward at 50% of its full speed

In this program, the motor is rotated at 25% of its full speed by setting the throttle to  $+0.25$  in one direction and  $-0.25$  in the opposite direction.



### Listing 1: DCMotor1.py

```
#-----
#                               BRUSHED DC MOTOR CONTROL
#
# In this program a brushed DC moor is controlled as follows:
# The motor is turned ON in forward direction for 5 seconds
# and then stopped for 5 seconds, then in reverse direction
# for 5 seconds and then stopped again for 5 seconds. This
# process is repeated forever
#
# Author: Dogan Ibrahim
# File : DCMotor1.py
# Date : February, 2023
#-----

import board
import time
import pwmio
from adafruit_motor import motor

#
# Initialize DC Motor 1
#
m1a = pwmio.PWMOut(board.GP8, frequency=50)
m1b = pwmio.PWMOut(board.GP9, frequency=50)
motor1 = motor.DCMotor(m1a, m1b)

while True:
    motor1.throttle = 0.25           # 25% of full speed
    time.sleep(5)
    motor1.throttle = 0             # Idle
    time.sleep(5)
    motor1.throttle = -0.25        # 25% of full speed
    time.sleep(5)
    motor1.throttle = 0            # Idle
    time.sleep(5)
```

Table 1: Motor Driver Truth Table.

Input A	Input B	Output A	Output B	Motor Action
Low	Low	Low	Low	Brake
High	Low	High	High	Forward
Low	High	Low	High	Backward
High	High	Hi-Z (Open)	Hi-Z (Open)	Coast



## Listing 2: DCMotor2.py.

```
#-----
#      TWO SPEED BRUSHED DC MOTOR CONTROL
#
# In this program a brushed DC motor is controlled as follows:
# The motor normally rotates at 25% of its full speed. Pressing
# button at port GP20 increases the speed to 50% of its full
# speed. Releasing the button returns the speed to 25%
#
# Author: Dogan Ibrahim
# File  : DCMotor2.py
# Date  : February, 2023
#-----

import board
import time
import pwmio
import digitalio
from adafruit_motor import motor

#
# Initialize DC Motor 1
#
m1a = pwmio.PWMOut(board.GP8, frequency=50)
m1b = pwmio.PWMOut(board.GP9, frequency=50)
motor1 = motor.DCMotor(m1a, m1b)
#
# Configure button at port GP20. The button state is
# normally at logic 1
#
btn = digitalio.DigitalInOut(board.GP20)
btn.direction = digitalio.Direction.INPUT
btn.pull = digitalio.Pull.UP
while True:
    motor1.throttle = 0.25          # 25% of full speed
    while btn.value == 0:          # If button pressed
        motor1.throttle = 0.5      # Increase speed
```

## Two-Speed DC Motor Rev Control

This is again a very simple project where a small, brushed DC motor is connected to the Maker Pi RP2040 dev board. The on-board push button on port GP20 is also used in the project. Normally, the motor spins at 25% of its full speed. Pressing the button will rev it up to 50% of its full speed.

The “ingredients” and the motor/dev board connections for this project are the same as in the previous project.

Listing 2 shows the *DCMotor2.py* program. The main program runs in a loop, where the state of the button on GP20 is checked. While the button is pressed, the motor speed is increased to 50% of its full speed.

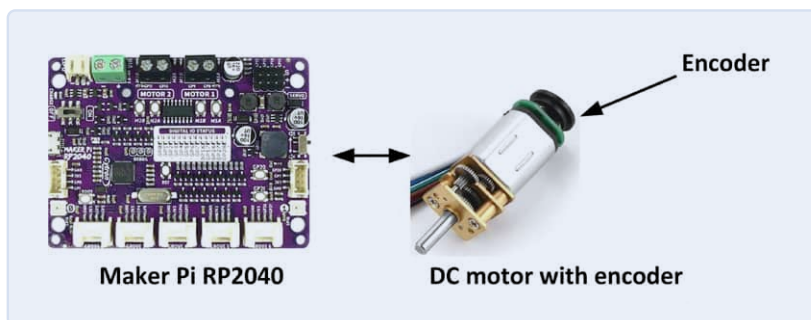
## Displaying DC Motor Speed Using a Rotary Encoder

This project shows how a rotary encoder can be used to sense the speed of a DC motor. The speed is then displayed on the screen. In “diagram representation,” this looks like **Figure 4**. Note that the motor, with its built-in rotary encoder, is not included in the Motor Control Development Kit and must be purchased separately. Sources include eBay, AliExpress, and others.

A small, geared, brushed DC motor with a built-in rotary encoder is used. **Figure 5** shows a picture of the motor where the rotary encoder is attached to the back shaft of the motor. In this project, the following motor is used: DC 6 V Gear Motor, 2 W, type GBMQ-GM12BY20 with encoder, 70 RPM. The no-load speed of the motor is specified as 70 RPM. In this project, you’ll be using an external +5 V power supply for the motor and, therefore, you’d expect the no-load speed to be under 70 RPM. Note that the current consumption of the motor can go up to 200 mA.

A rotary encoder (or a shaft encoder) is a device that converts the angular position of a rotating device such as a motor into an electrical signal. Rotary encoders are used in motor control applications to sense the speed of the motor or the position of the motor shaft. There are basically two types of rotary encoders: Optical and Hall Effect. Optical rotary encoders work with optical principles, where light shines onto a photodiode through slits (or holes) in a metal or any other form of disc. By counting the number of holes passing in front of the photodiode in a given time, you (or your micro!) can calculate the speed of the motor.

In this project, a Hall Effect-based rotary encoder is used. Two static magnetic sensors (called Phase A and Phase B) are used on the encoder board together with



▲  
Figure 4: The project's block diagram.

Figure 5: The motor/encoder assembly, type GBMQ-GM12BY20.



a rotating magnetic disc. The sensors supply pulses as the motor rotates. By counting the pulses over a given time, you can calculate the speed of the motor. **Figure 6** shows output waveforms of the motor used in this project. The top waveform represents Phase A and the bottom Phase B. These types of encoders are also known as *quadrature encoders* since the magnetic sensors are placed at 90 degrees to each other and there are four possible output states. The direction of rotation is easily determined by finding the order in which the output signals change from 0 to 1. For example, if the Phase A signal rises from its LOW condition, then the motor rotates in one direction. If, on the other hand, the Phase B signal rises while the Phase A signal is LOW, then the motor rotates in the opposite direction — see **Figure 7**.

The specifications of the motor used in this project are as follows (dependent on the voltage applied):

- > 6 V DC operation
- > Shaft speed after gears: 70 RPM (at 6 V)
- > Power: 2 W
- > Current 170...200 mA
- > Two built-in Hall Effect sensors
- > Weight: 14 grams

A 6-wire connector is attached to the motor back panel. The pin configuration of the motor is as follows:

Black wire	motor power GND
Red wire	motor power (+5 V in this project)
Yellow	encoder power (+3.3 V in this project)
Green	encoder power GND
Blue	encoder output Phase A
White	encoder output Phase B

**Figure 8** shows the wiring diagram for the project. Notice that the motor is operated with an external +5 V power supply capable of providing up to 500 mA. The encoder is operated off +3.3 V line from dev board so that the voltage is compatible with the input voltage levels of the RP2040 processor. Also notice that only the Phase A output of the motor is used in this project.

Sadly, electrical motor manufacturer and distributors do not give the gear ratio and the number of encoder pulses output from the motor per second. These are important specifications, though. In this section, a program is written to display these important parameters for the motor used. You may have to carry out these calculations in order to find the important specifications of the motor you are actually using.

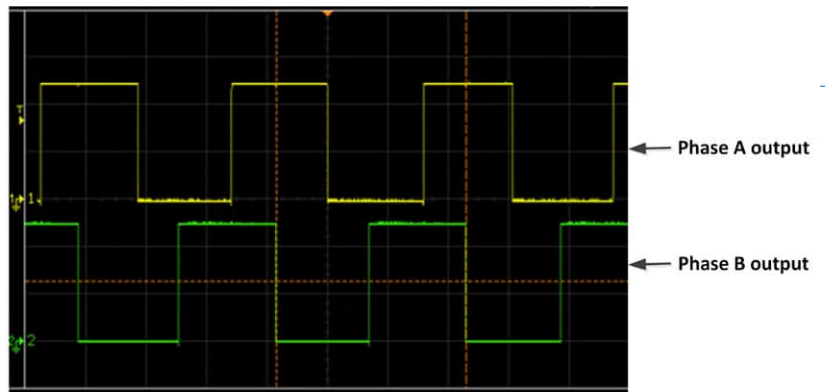


Figure 6: Rotary encoder's output waveforms.

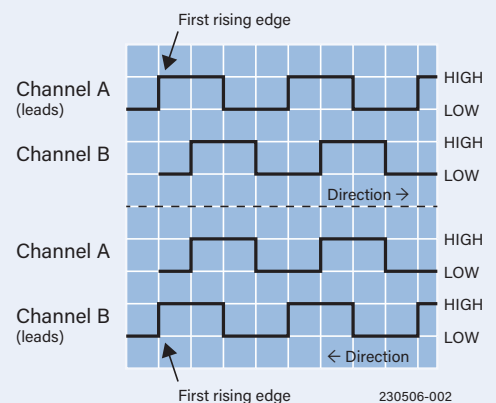
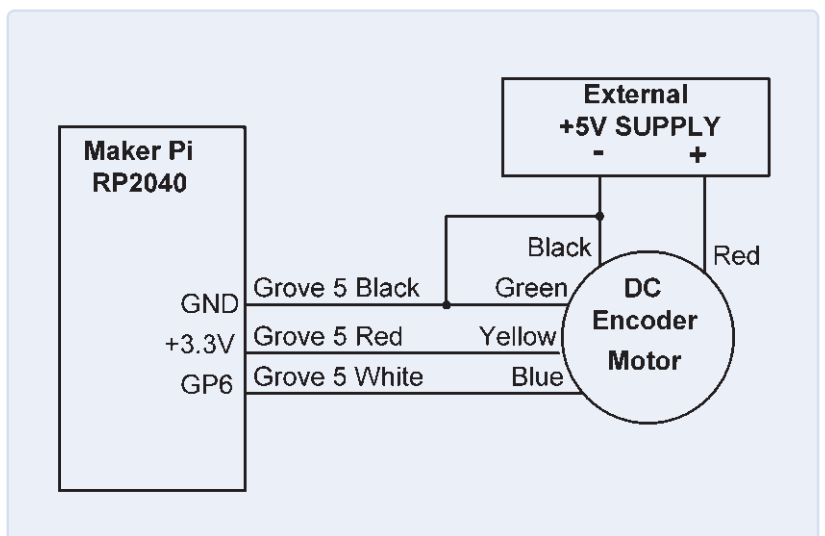


Figure 7: Determining the direction of rotation (Source: robotoid.com).



**Listing 3** shows the *MotorPulses.py* program created to display the number of motor pulses output from the Phase A encoder every second. In this program, port GP6 is assigned to `Encoder` object and is configured as an input, and the number of pulses received from the encoder is calculated and displayed at the end of each second. Notice that this program uses the `time.monotonic()` function to calculate the timing, and it may not be very accurate. What is ideally required is to receive the pulses as external interrupts and use one-second timer interrupts to calculate and display the number of pulses received every second. Unfortunately, CircuitPython does not support external or timer interrupts.

Figure 8: The project's wiring diagram.



### Listing 3: MotorPulses.py

```
#-----
#      DISPLAYING THE NUMBER OF ENCODER PULSES
#
# In this program a geared DC motor with Hall Effect sensors
# is connected to the Maker Pi. This program calculates
# and displays the number of pulses received from the encoder
# every second. Only Phase A encoder output is used here
#
# Author: Dogan Ibrahim
# File  : MotorPulses.py
# Date  : March, 2023
#-----

import board
import digitalio
import time

Encoder = digitalio.DigitalInOut(board.GP6)
Encoder.direction = digitalio.Direction.INPUT

while Encoder.value == 0:      # Wait while 0
    pass

strtime = time.monotonic()    # Start time, rising edge
                                # detected
count = 0                     # Initialize count
#
# Main program loop
#
while True:
    while Encoder.value == 1:  # Wait while 1
        pass
    while Encoder.value == 0:  # Wait while 0
        pass
    endtime = time.monotonic() # End time
    if endtime-strtime > 1.0:  # Just over a second
        print(count)          # Display count
        strtime = time.monotonic() # Start time
        count = 0
    else:
        count = count + 1     # Increment count
    while(Encoder.value) == 1: # Wait while 1
        pass
```



### Listing 4: MotorSpeed.py

```
#-----
#      DISPLAYING THE MOTOR SPEED
#
# This program displays the motor speed in RPM using the
# number of encoder pulses received every second and the
# formula given in the text
#
# Author: Dogan Ibrahim
# File  : MotorSpeed.py
# Date  : March, 2023
#-----

import board
import digitalio
import time

Encoder = digitalio.DigitalInOut(board.GP6)
Encoder.direction = digitalio.Direction.INPUT

while Encoder.value == 0:      # Wait while 0
    pass

strtime=time.monotonic()      # Start time
count=0                       # Initialize count
#
# Main program loop
#
while True:
    while Encoder.value == 1:  # Wait while 1
        pass
    while Encoder.value == 0:  # Wait while 0
        pass
    endtime=time.monotonic()   # End time
    if endtime-strtime > 1.0:  # Just over a second
        RPM = count * 60 / 880 # Motor speed
        print("Speed=%6.2f RPM" %RPM) # Display speed
        strtime=time.monotonic() # Start time
        count=0
    else:
        count=count+1         # Increment count
    while(Encoder.value) == 1: # Wait while 1
        pass
```

CircuitPython REPL

```
Speed= 63.95 RPM
Speed= 64.02 RPM
Speed= 64.02 RPM
Speed= 63.95 RPM
Speed= 63.89 RPM
Speed= 63.95 RPM
```

Figure 9: Output from the program.

Figure 10: The DT-2234C digital photo tachometer.



#### Related Products

> **Motor Control Development Bundle**  
[www.elektor.com/20534](http://www.elektor.com/20534)





When the motor is run, the *MotorPulses.py* program displays 938 pulses/second. A round shaped object (a small tyre) was attached to the motor shaft and a mark was made on it so that the rotations could be counted, and the number of revolutions of the motor shaft under no-load was observed to be 64 RPM.

Now, the number of pulses received every minute is  $938 \times 60 = 56,280$  pulses/minute, and this corresponds to a no-load speed of 64 RPM. Therefore,  $56,280/64 = 879.38$ , or 880 as the nearest integer, and then the motor speed can be calculated using the following formula:

*Motor speed (RPM) = (measured number of pulses per second  $\times$  60) / 880*

For example, if the number of pulses received is counted to be 450 every second, then the motor speed is:

*Motor speed =  $450 \times 60 / 880 = 30.68$  RPM*

The above formula will be used to calculate the motor speed.

The program *MotorSpeed.py* listed for you in **Listing 4** is used to calculate and then display the motor speed using the above formula. This program is essentially the same as in Listing 3, but here the motor speed is calculated and displayed on the screen. **Figure 9** shows some sample output from the program.

As discussed earlier, more accurate results could have been obtained if external interrupts were used to count the encoder pulses, with timer interrupts used to measure the time.

### R U Sure, RP2040?

The motor speed can easily be measured and verified using a digital photo tachometer device. There are many such devices at different prices. The one used by the author was a model DT-2234C (**Figure 10**) that set him back around £8 plus the cost of a sone PP3-style 9 V battery. Before using this device, a piece of reflective paper must be fixed to the motor shaft, as shown in **Figure 11**.

The motor speed is measured with the DT-2234C tachometer as follows:

- Start the motor rotating.
- Press the *TEST* button on the tachometer and aim the device light at the reflective paper.
- The motor speed will be displayed on the LCD continuously.
- You can save the readings by pressing the *MEM* button.

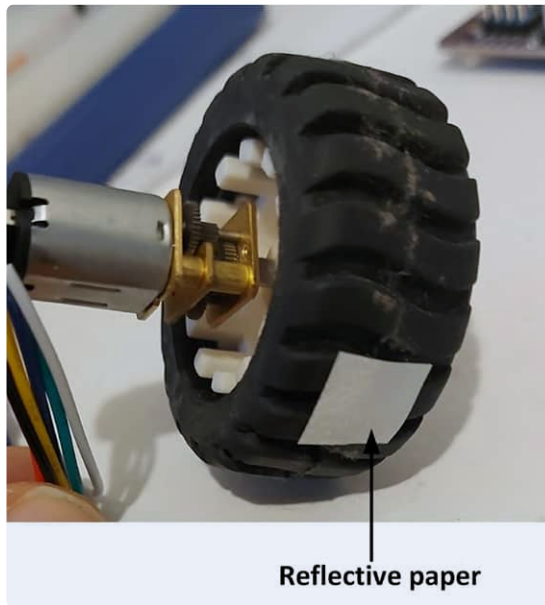


Figure 11: Affixing a piece of reflective paper to the motor shaft.

**Note:** In this project, only one encoder output phase (Phase A) is used. More accurate motor speed results can be obtained if both encoder phases (i.e., Phase A and Phase B) are used. Also, it is possible to drive the motor from the MOTOR 1 or MOTOR 2 screw terminals of the Maker Pi RP2040 development board. This will limit the maximum motor voltage to +3.3 V and will also require the motor speed to be controlled by sending PWM voltage waveforms to the motor through the ports GP8/GP9 (MOTOR 1) or ports GP10/GP11 (MOTOR 2). ◀

230506-01

### Questions or Comments?

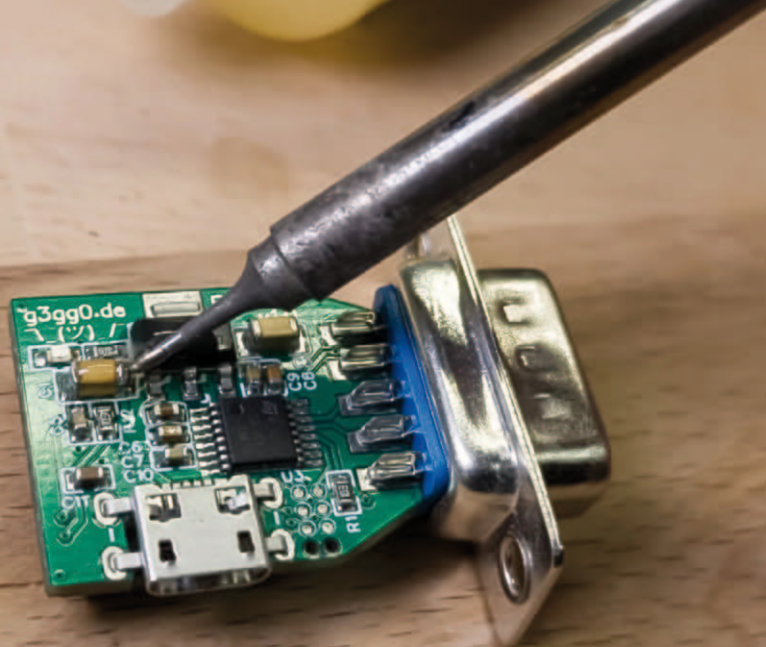
Do you have any questions or comments related to this article? Email the author at [d.ibrahim@btinternet.com](mailto:d.ibrahim@btinternet.com) or Elektor at [editor@elektor.com](mailto:editor@elektor.com).

### About the Author

Dogan Ibrahim holds a BSc (Hons) degree in Electronic Engineering, an MSc degree in Automatic Control Engineering, and a PhD in Digital Signal Processing and Microprocessors. Dogan has worked in many organizations and is a Fellow of the Institution of Engineering and Technology (IET) in the UK and is a Chartered electrical engineer. Dogan has authored over 100 technical books and over 200 technical articles on electronics, microprocessors, microcontrollers, and related fields. Dogan is a certified Arduino professional and has many years of experience with almost all types of microprocessors and microcontrollers.

### WEB LINKS

- [1] Motor Control Development Bundle:  
<https://elektor.com/motor-control-development-bundle>



# ESP32-RS-232 Adapter

A Wireless Link for Classic Test Equipment

By Georg Hofstetter (Germany)

RS-232 goes wireless! A modern and low-cost ESP32 module will give wings to a serial interface. In addition to its serial link communication capability, this neat adapter will allow SCPI-equipped test equipment to receive commands and send data to the home network via MQTT.

As you work on different electronics projects over the years, the stack of test equipment on your workbench inevitably grows. In my case, I use a Keithley SourceMeter 2400 to identify faults in electronic assemblies, bring them into operation, or even measure the properties of various components. It's also great for testing the leakage current of electrolytic capacitors or measuring their nominal capacitance. Another piece of equipment in my setup is an AOR AR5000 receiver, which I use to keep in touch with amateur radio contacts worldwide.

Both of these units, like many others in the lab, can be controlled by a PC thanks to their built-in RS-232 serial interface ports. The availability of USB-RS-232 adapter cables means that connection to a home PC should be fairly straightforward. In my case, the adapter worked fine with the AR5000, but commu-

nication with the Keithley SourceMeter was noticeably less stable. Characters occasionally get lost when sending SCPI commands, and connection dropouts are frequent.

## The RS-232 Link

The RS-232 standard, defined back in the 1960s, is used to transmit serial data bits of a character using changes in voltage levels. The electrical properties were defined in V.28, with the overarching functionality regulated in V.24. The V.28 standard stipulates that, depending on the logical bit value, the sender signal voltage transition required to send a logic "0" should rise to a value from 5 to 12 V and fall to a value between -5 and -12 V to send a logic "1". The receiver has a slightly wider voltage window and will interpret a signal in the range from 3 to 12 V as a logic "0" and -3 to -12 V as a logic "1". This takes into account

signal degradation in the cable from noise, voltage drop, and signal edge rounding due to the cable impedance properties.

In older, low-cost, USB-to-Serial adapters and some laptops, the signal level swing from the sender side barely gets greater than  $\pm 5$  V or even  $\pm 3$  V due to shortcuts made to the design of the RS-232 signal driver stage to minimize development and component costs. This has led to the popular belief that laptops and RS-232 devices often don't work well together. The illustration in **Figure 1** from a TI document illustrates the voltage levels that must be present to comply with the standard.

Whether the aforementioned communication problem with the Keithley device was due to the use of a USB extension cable, a ground loop, or the electrical characteristics of the built-in transceiver is irrelevant at this point. After all, a USB cable stretched across the room is not only untidy but also a trip hazard. What's needed is a neat, compact wireless solution!

I searched high and low for a ready-made solution to my problem, but all I could find were fairly large modules that mounted onto DIN rails or more limited DIY designs built to solve one specific problem. It was no good;

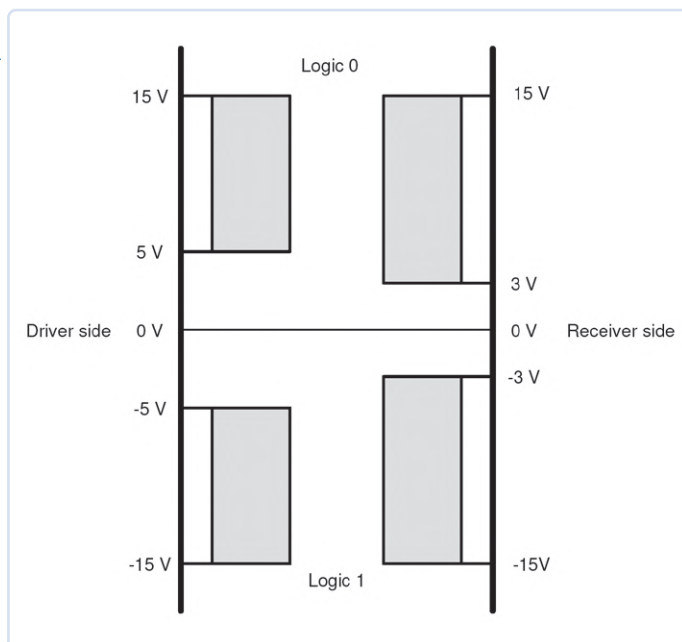


Figure 1: Specified RS-232 signal levels from a sending device and at a receiver input. (Source: Texas Instruments [9])

Monolithic Systems [3] AMS1117 linear voltage regulator from the +5 V USB port supply. Although the AMS1117 is rated for up to 1 A, it gets quite hot on this relatively small board. A more efficient alternative would be the Texas Instruments [4] TPS62291 step-down switch-mode converter, which comes in a  $2 \times 2$  mm WSON-6 package. It also operates up to 1 A output and would save some space. Unless you have a microscope and all the necessary tools and experience for working down at this level, this chip is a bit more challenging to mount due to the tiny package outline. Those uncertain about soldering can find useful guidance and tips in the video [5].

if I wanted a neat, general-purpose RS-232 radio communication unit, I was going to have to sit down and design it myself.

## The Design Criteria

This design is consciously aimed at the maker-lab environment and will not be a beautifully encased product you might find for sale in an electronics store. It is nevertheless good practice that the unit's design is carefully considered so that it meets all the requirements for the job it's tasked to perform.

### An ESP32 does all the work

The RS-232 gateway is primarily intended to transmit data to and from the network. This is typically done using TCP-Port 23 (Telnet). However, application-specific software adaptations should also be possible, which the user may decide on according to the application requirements. For me, an SCPI-to-MQTT service was important, so that the SourceMeter test equipment can transmit its readings overnight to Grafana (see below) for evaluation.

The ESP32 provides sufficient computing power beyond sending simple TCP packets so that the connected device can be easily integrated into Home Assistant or a comparable service via MQTT or to share it via a dedicated front end over the ESP32 web server.

An alternative would be the ESP8266, which takes up slightly less space and is a bit cheaper. However, the space saved comes at the cost of a significantly over-worked CPU

and less RAM. On top of that, the ESP8266 lacks a Bluetooth module, which would be an ace up the sleeve in this application, alongside the desired Wi-Fi function.

The drawback of the ESP32 is, of course, its significantly higher power consumption, which peaks at around 500 mA from the 3.3 V rail. Of course, the CPU will not be continuously occupied calculating, but depending on how well-optimized the software is in the end, its peak current is likely to reach this magnitude, as indicated in Table 4.2 of the datasheet [1]. A comparison with the ESP8266 datasheet [2] shows that the ESP32's power consumption is about 35% higher while transmitting and approximately 80% higher when receiving.

### A neat, compact design

Most people would probably agree there's already enough cable clutter snaking around lab benches and under desks. This circuit board can be soldered directly to a D-sub DE-9 connector and, in terms of size, works out to be not much larger than a standard IEC C13 (kettle lead) power connector including the bend radius of the power cable. This means the board, when plugged in, doesn't increase the bench footprint of the RS-232 device. If a case is needed, it should be easy to 3D print; ideally, the circuit could even fit into a standard off-the-shelf enclosure.

### Power from a Micro-USB connector

Power to the board is provided via a micro-USB socket as is already standard for many small devices. The +3.3 V required by the ESP32 is produced by the Advanced

It should also be noted that the supply of this particular component over the last couple of years has also been a little erratic. This has led to its price soaring to well over €20 from some distributors instead of the more usual €1.70. Comparing prices on the internet, you may be lucky enough to find units at prices reminiscent of the good old days. To reduce hassle, I just used the AMS1117 in this version, which makes hand soldering much simpler.

### Power supply using the 9-way D-Sub

If you are not averse to occasionally going "off-piste," the board could also be powered via the 9-pin D-Sub connector. With a slight adjustment to the terminal device or devices, the required 5 V can be provided via the RS-232 connector (and thereby avoid an additional power supply). The standard, however, does not support such a power supply option via the connector, so we have to be aware this will be a non-standard modification.

According to V.28, all signal lines can carry voltages from +15 V to -15 V, to supply power it makes sense to replace one of the pin functions which is seldom used. Pin 9 is designated as the Ring Indicator (RI) and is only used by a modem to signal an incoming call. RS-232 modems are now mostly redundant, which makes the RI pin a suitable candidate to use as a power supply pin.

First, however, we need to check what happens if we unexpectedly apply full voltage, for example, by connecting a terminal device that drives our supply line according to V.28.



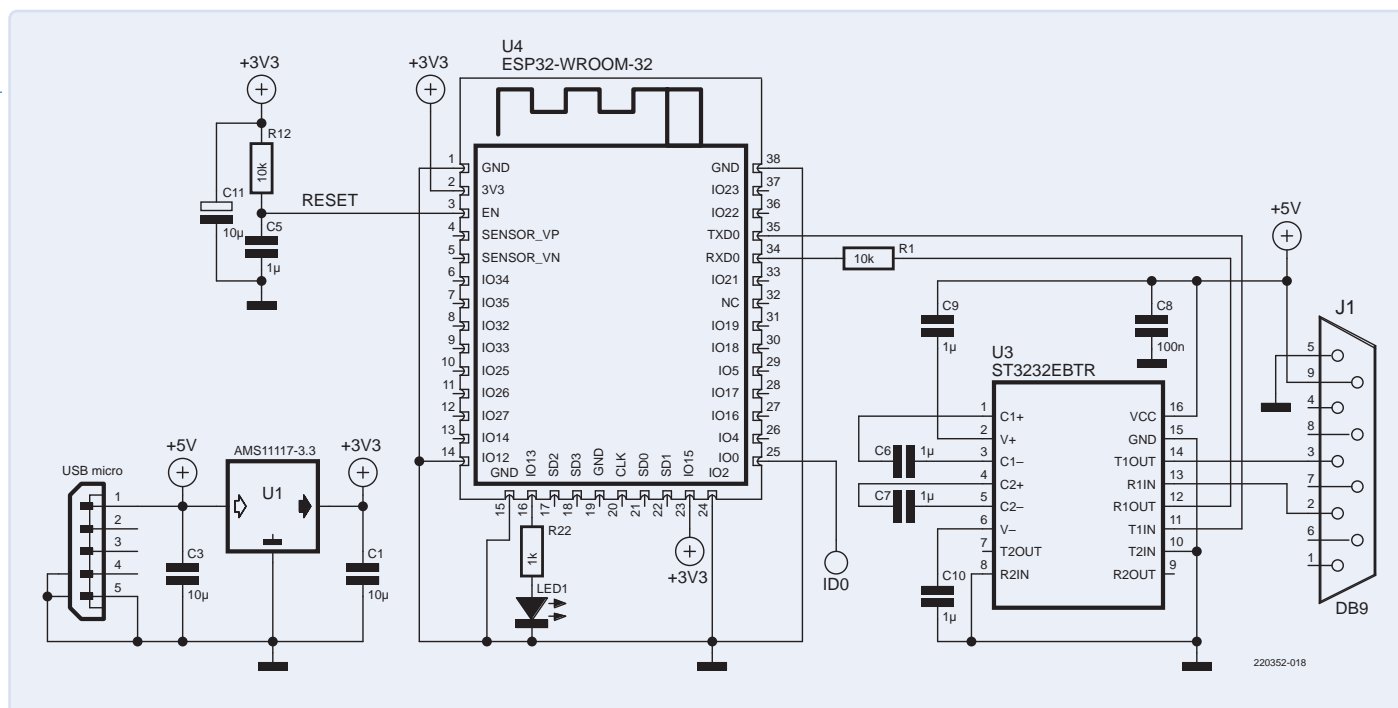


Figure 2: The simple circuit diagram of the ESP32-RS-232-Dongle.

The +15 V would not be a problem for the AMS1117 itself; this voltage is within its specified limits. However, if -15 V were present, the internal protection diodes of the AMS regulator would conduct the usually 10 to 20 mA of RS-232 signal to ground. Empirical observations show that these diodes have no problem handling this, and according to their specification, the RS-232 drivers are short-circuit-proof, so no damage occurs here either.

The VDD supply voltage of our RS-232 transceiver is only specified up to 5.5 V and is also connected to the +5 V. Fortunately, in this case, the voltage drops to around 3.5 V due to current limiting by the RS-232 drivers. Admittedly, we are flying by the seat of our pants on this one, but we have only considered all these factors for the rare case that RI on Pin 9 will actually be driven by the terminal device. It is usually unconnected.

### True RS-232

As mentioned earlier, our aim here is to adhere to the V.28 standard as closely as possible, which is why a data signal transceiver will be absolutely necessary. The ST232EBTR from ST [6] will do the job and is available in a TSSOP-16 outline. This device is also quite affordable at around €1 and features an internal voltage doubler and inverter circuits, it generates supply voltages of plus and minus 10 V, which are used to drive the RS-232 signals, typically measuring around 9 V under normal operating conditions.

The circuit diagram of the ESP32-RS-232 dongle with the ESP32-WROOM module (or the WROVER with PSRAM) and the two ICs: ST232EBTR and AMS1117, along with all peripheral components, can be seen in **Figure 2**.

### Component Placement

For such a small board with so few components, it makes little sense to have the components mounted professionally or to have the added expense of a layout stencil and subsequent assembly using a home hotplate. For this, PCB shown in **Figure 3**, we can get away with using old-fashioned hand soldering methods because the component count is low, and the pads are accessible.

The first component to be assembled is the ST232 (U3). It is relatively flat, and the pins may require some rework to clean the joints. Next, the capacitors, resistors, and LED can be soldered in place followed by the AMS1117 (U1) and the USB socket. Now the ESP32 and, finally, the 9-pin D-Sub connector can be soldered.

### Firmware

The firmware's most basic function is to route characters received over the network to the RS232 port and vice versa, but thanks to the versatility of the ESP32, much more is possible here. The firmware, as found in the repository [7], currently offers two operating modes — Telnet and MQTT/SCPI — which are explained below.

### Telnet / Raw TCP-Socket

As we already outlined in the design criteria, the primary function is to transfer data from the network to the RS-232 port. For such adapters, it's common to transmit characters raw using the TCP-Port 23. This port was previously used for the Telnet connection to a server but is rarely used today (except in embedded systems). For our purpose of transmitting commands or measured values to and from serial devices, this is absolutely sufficient in a home network.

Telnet is a network communication protocol developed to enable the remote control of a computer over a network. It provides a text-based communication channel allowing a user to access a remote computer and execute commands there. This connection is bidirectional, so that both inputs and responses are relayed over the network.

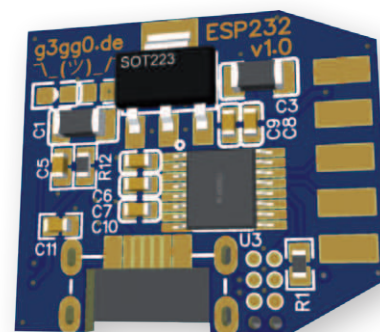


Figure 3: This small PCB with easily accessible pads is ideal for hand soldering SMD chips.





Figure 4: A example Grafana dashboard, showing waveforms made up from measured values over time. (Not related to this project).

Originally developed in the more naive era of early internet standards, Telnet is often used in applications where simple, text-based communication is sufficient. It is, however, considered insecure as it lacks encryption, potentially allowing interception of all transferred data including passwords and other sensitive information.

For initial communication tests, you can use the *telnet.exe* tool, which came as standard in earlier versions of Windows. If this is not available, you can use the free and more universal alternative, PuTTY [8].

### MQTT/SCPI

For applications which need to regularly capture measured values over periods of several hours, you can develop appropriate Python scripts or even start ready-made tools on your PC and read them via the Telnet port. A practical alternative to this, for those who already have established a suitable infrastructure in their domestic network, is to bypass the power-hungry PC altogether and provide a more simplified measurement setup.

Many households will already be running services such as MQTT, InfluxDB, and Grafana for capturing measurement values from sensors on a Raspberry Pi or via Docker Containers using their home NAS. Without going into too much detail and in simple terms:

➤ **MQTT** is a lightweight messaging

protocol designed for efficient communications between devices in IoT and low bandwidth, high-latency networks. It uses an information broker with a publish/subscribe model to enable seamless data exchange with minimal overhead.

- **InfluxDB** is a time-series database, designed to efficiently handle high write/query loads for time-stamped data. Tags are key-value pairs for indexing while fields contain the actual data.
- **Grafana** takes data values from a database and displays the information in a user-friendly way, with user-defined graph axes.

For those with such a setup at home, this

firmware provides the option to publish measurement values generated by the SCPI-speaking device directly to an MQTT broker for further dissemination.

The set of commands used by each SCPI device varies significantly. The only device currently supported by the code is the Keithley SourceMeter 2400. This, however, serves as a good starting point for anyone to add their own firmware variants. Skilled embedded developers are invited to expand the software functionality and share it with the community.

To see how measurement representations can look in different applications, refer to **Figure 4** (Grafana), **Figure 5** (MQTT Explorer,

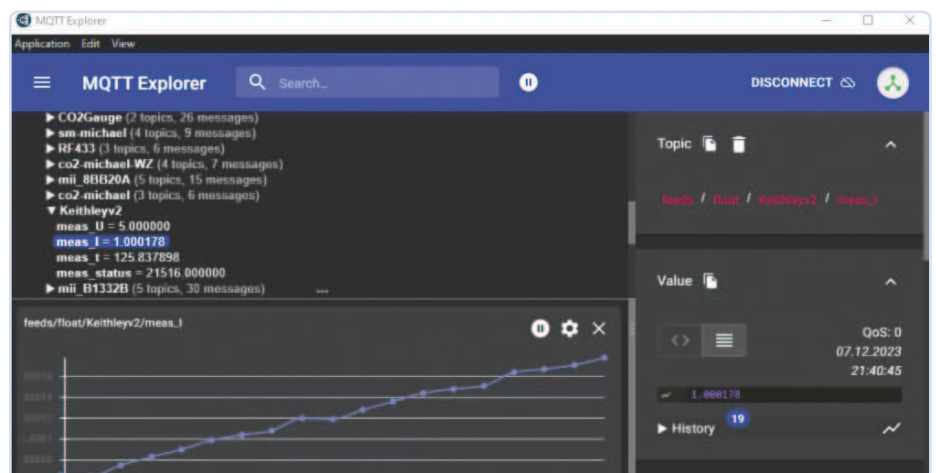


Figure 5: MQTT Explorer shows measurements published by the ESP32-RS-232 adapter via MQTT.

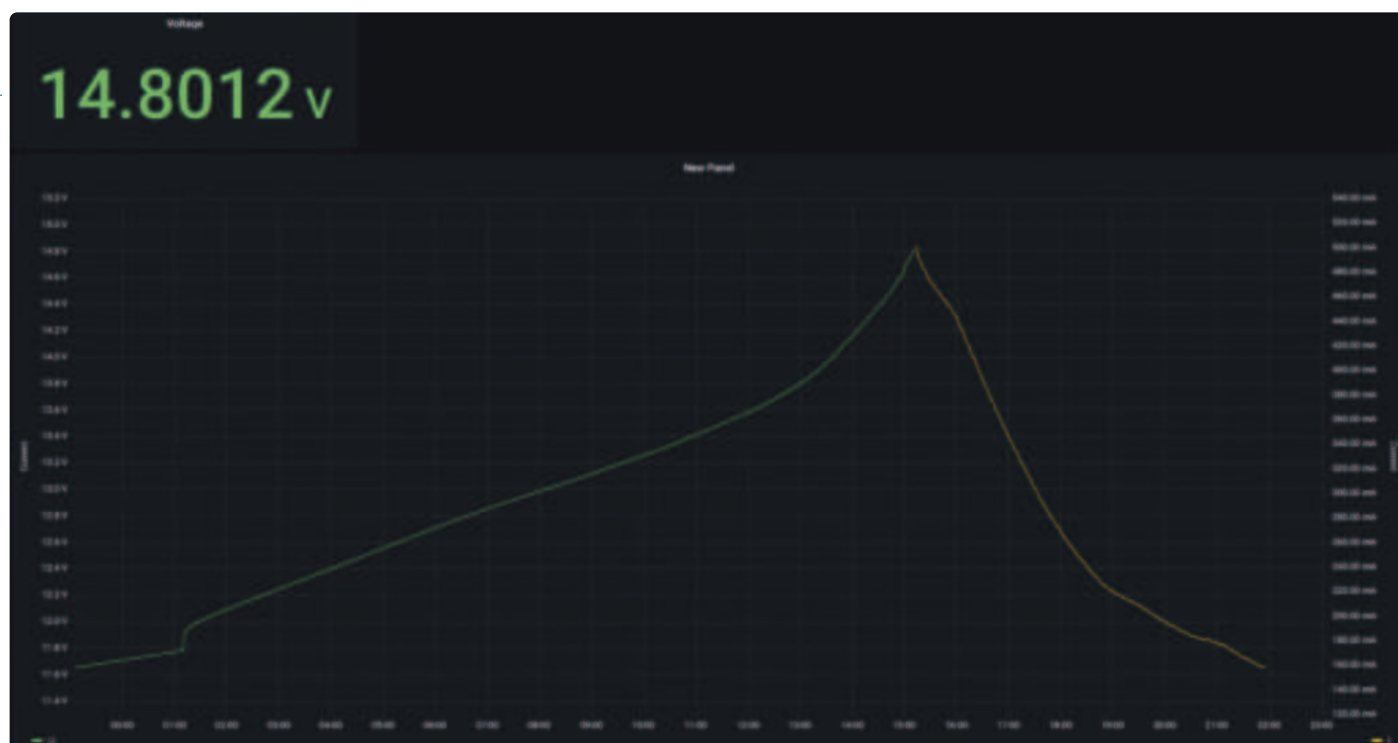


Figure 6: Battery voltage (green) and current flow (yellow) characteristics when charging a 12 V lead acid battery, read by an ESP32-RS232 adapter.

a comprehensive MQTT client for various platforms [10]), and **Figure 6** (also Grafana).

To make this possible, the ESP32 needs to take care of a few things, and for this it requires information that we need to provide via a web interface.

### Wi-Fi Setup

If a configured access point (AP) is not found or none is configured, the ESP32 becomes active and offers itself as an AP with the name *esp232-config*. It's best to connect to this AP using a Smartphone. After logging in, you can configure the most important parameters by accessing <http://192.168.4.1/> via a browser. The web page hosted on the ESP32 can be seen in **Figure 7**. **Table 1** explains what needs to be entered in each row.

### OTA: Updates Over-The-Air

In the Arduino IDE or in PlatformIO, which I prefer, the ESP32 can be updated "remotely." The underlying protocol is implemented by the ArduinoOTA component. While this feature facilitates frequent updates of the microcontroller, in practice significant drawbacks were revealed. It seems that constantly allocating and releasing buffers for each received UDP packet fragments the ESP32's memory led to unpredictable resets. To make matters worse, this happens regardless of whether OTA packets are in transit. You can imagine that a crash message output via the ESP32's

## ESP232 - ESP232

v1.13 - b376f34

[\[Enable OTA\]](#)

Hostname:	ESP232
WiFi SSID:	g3gg0.de
WiFi Password:	
WiFi networks:	<a href="#">Scan WiFi</a>
MQTT Server:	
MQTT Port:	
MQTT Username:	
MQTT Password:	
MQTT Client Identification:	ESP232
Baudrate:	57600
Data bits (5-8):	8
Parity (0=none, 1=even, 2=odd):	0
Stop bits (0=1, 1=1.5, 2=2):	0
Connect Message:	
Disconnect Message:	
Verbosity:	Serial <b>UDP</b> <input type="checkbox"/> <input type="checkbox"/>
Publish rate [ms]:	500
Publish data via MQTT:	<input type="button" value="Publish string"/> <input type="button" value="Publish parsed"/> <input type="button" value="Exit REM"/> <input type="button" value="Publish MEAS"/>
Update URL ( <a href="#">Release</a> ):	
<input type="button" value="Save"/> <input type="button" value="Save &amp; Reboot"/>	

Figure 7: The ESP32 Wi-Fi setup.

Table 1: Entries in the ESP32 Wi-Fi setup.

Hostname	The name the device uses to identify itself via MDNS in the network.
WiFi SSID/Password	The Wi-Fi network the device should connect to at startup.
WiFi Networks	Displays a list of found Wi-Fi networks; clicking on one takes its name.
MQTT [...]	Configuration settings for the MQTT client, in conjunction with SCPI below.
Baudrate / Databits / Stop bits	Corresponding parameters for RS232 communication.
Dis-/Connect Message	[Special Cases] Message to be sent via RS232 for TCP connection.
Verbosity	[Experts] Debugging communication via UDP.
Publish data via MQTT	[Experts] For the Keithley Source Meter 2400: Parsing display data or initiating a measurement and publishing the results via MQTT.
Update URL	For firmware updates, simply insert the HTTP URLs, and the expected .bin file will be downloaded and flashed.

Table 2: Pin assignment of the programming interface.

Pin Name	Purpose	Level
IO0	Sets boot mode, takes effect only after a RESET has been executed.	Flash: GND (Normal: floating, 3.3 V)
RESET	The ESP32 reset pin (CHIP_PU)	active: GND, inactive: floating (3.3 V)
TXD0	Transmit signal from the ESP32	0V / 3.3 V
RXD0 (U3_34)	The ESP32 Receive signal	0V / 3.3 V
+5V	Supply voltage	5 V
GND	Ground	GND

UART is rarely well-received by the connected device. Fortunately, there have been improvements [11] that significantly reduce the crash rate. Due to this level of uncertainty, the OTA feature is only activated briefly and on user request, which is why there's a field (*Enable OTA*) for it in the web interface.

Flashing

If you don't already have a Pogo-Adapter for the ESP32 modules, the microcontroller needs to be provided with the appropriate firmware after assembly using the transmit and receive data pins of the ESP32. To do this,

download the source code from [7], import it into PlatformIO, compile, and finally, flash it to the board.

Provision has been made for a programming port at the bottom right-hand corner of the board. For occasional use, you can just link the IO0 signal to GND and solder the TXD0 and the RXD0 (Pin U4\_34 on the microcontroller, see **Figure 8**) to a USB-UART adapter.

As **Table 2** shows, the ESP32 has a few "strapping pins" that need to be set when loading the ROM code using this method. In this setup,

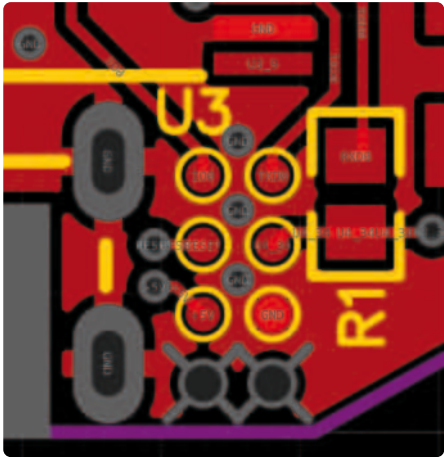


Figure 8: Connections on the PCB for the programmer.

we only need to worry about one (IO0). Once the firmware is uploaded, an LED should start blinking after about 30 seconds.

The Ugly

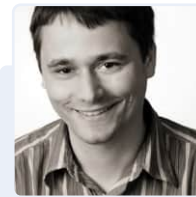
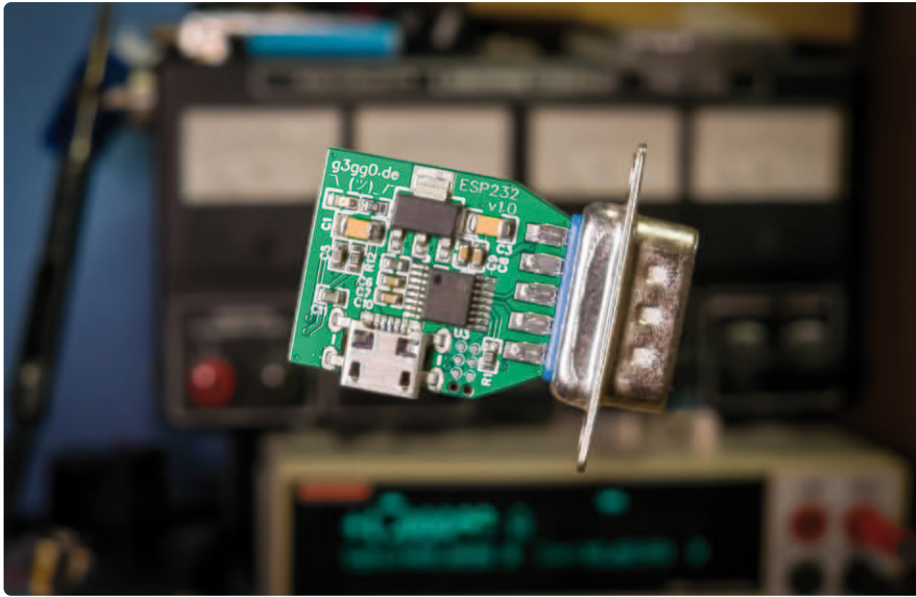
The ESP32 has proven its capabilities in my projects. There were however, initially various problems in the Arduino library regarding use of the UART buffer, leading to the code directly accessing the libraries in the IDF framework. Bugs in the ArduinoOTA library also led to me spending some nights burning the midnight oil in an effort to try and isolate the problems. As a consequence, there are certain workarounds in the source code. But since this firmware is open source, it will continue to evolve and benefit from improvements contributed by the community.

The Next Generation

What bothers me about v1.0 of the project until now is the somewhat cumbersome flashing process during the initial setup (or flashing to correct code errors). Thanks to a flash adapter with Pogo pins that I use for all my projects, the effort is reduced considerably, but not everyone has a suitable adapter on hand or likes working with jumper leads.

For this reason, the ESP32-RS-232-Adapter is moving on to its next incarnation and is currently being developed as version 2.0 using an ESP32-S3-PICO. There are already some initial prototypes produced, and they look promising. The PICO is a complete module with SPI flash and RAM, all integrated into an





### About the Author

Georg Hofstetter's interest in electronics was kindled in the era of the Commodore 64 and perfboard. After training as an electronics technician, he went on to complete a degree in computer science. Since then, he has been developing or reverse-engineering software and embedded systems, publishing selected projects on his website [www.g3gg0.de](http://www.g3gg0.de). Some of the more well-known projects include firmware modifications for Canon DSLRs called Magic Lantern ([www.magiclantern.fm](http://www.magiclantern.fm)) and for the Toniebox (<https://gt-blog.de/toniebox-hacking-how-to-get-started>).

LGA-56 package. Although the PICO looks like a QFN outline at first glance, which is usually easy to hand solder, GND must be connected to an exposed pad, requiring the pad to be soldered through a hole during hand assembly. This package also has no lead frame, and the pins can only be soldered from the edge with difficulty. Ultimately, a (mini) hotplate would seem to be the best method to use here [12].

The significant advantage of the S3 is that it has a full USB peripheral, allowing it to be flashed or debugged. If necessary, ESP232 v2.0 can also be used as a USB-to-RS232 adapter.

Whether we're just going around in circles and ending up where we started is a rhetorical question. When you think that the ESP8266 was from the start ideally capable for building a Wi-Fi-to-Serial bridge, it leaves you wondering why there still isn't such a device available on the market today. ◀

220352-01

### Questions or Comments?

Do you have any technical questions or comments about this article? The please contact the author at [elektor@g3gg0.de](mailto:elektor@g3gg0.de) or the Elektor editorial team at [editor@elektor.com](mailto:editor@elektor.com).



### Related Products

> **ESP32-WROOM-32**  
[www.elektor.com/18421](http://www.elektor.com/18421)

> **ESP32-S2-WROVER** ◯  
[www.elektor.com/19693](http://www.elektor.com/19693)



### WEB LINKS

- [1] ESP32 Data sheet: [https://www.espressif.com/sites/default/files/documentation/esp32\\_datasheet\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf)
- [2] ESP8266 Data sheet : [https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex\\_datasheet\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf)
- [3] AMS1117: <http://www.advanced-monolithic.com/pdf/ds1117.pdf>
- [4] TPS62291: <https://www.ti.com/product/de-de/TPS62291>
- [5] SMD soldering by hand, Elektor video : <https://www.elektormagazine.de/news/uberzeugendes-video-loten-von-smd-bauteilen>
- [6] ST232: <https://www.st.com/resource/en/datasheet/st202eb.pdf>
- [7] Project Repository: <https://github.com/g3gg0/ESP232>
- [8] PuTTY: <https://www.putty.org/>
- [9] MQTT Explorer: <https://mqtt-explorer.com/>
- [10] WiFiUDP Crash issue: <https://github.com/espressif/arduino-esp32/issues/4104>
- [11] Texas Instruments Application Report: RS-232 Frequently Asked Questions: <https://www.ti.com/lit/an/slla544/slla544.pdf>
- [12] M. Divito, "Mini Reflow Plate," Elektor 11-12/2023: <https://www.elektormagazine.de/230456-01>



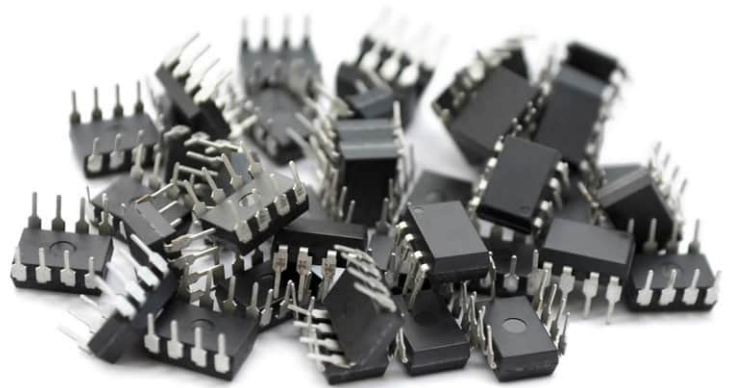


# Starting Out in Electronics...

## ...More About Opamps

By Eric Bogers (Elektor)

In the previous installation, we took a look at discrete FETs and got started with a discussion about what is probably the most important component in analog electronics: the opamp. There's a whole lot to be said about opamps, as you will see, and there are a lot of formulas that we can (and will!) use in connection with opamps.



The main thing you need to pay attention to is the open loop gain, which is usually in the range of 10,000 and 100,000. Using this parameter, you can calculate the output voltage with the following formula, where  $U_{ni}$  is the voltage on the non-inverting input and  $U_i$  is the voltage on the inverting input:

$$U_{out} = V_0 \cdot (U_{ni} - U_i)$$

### The Operational Amplifier as a Black Box

When you want to use an opamp in a practical application, as a neophyte electronics enthusiast you are hardly interested in what it looks like on the inside. In fact, you only need a few important parameters to start using an opamp. This means you can regard the opamp as a sort of black box with inputs and outputs (**Figure 1**).

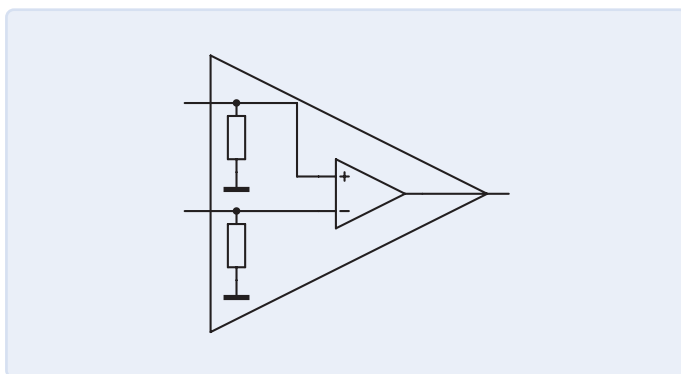


Figure 1: The operational amplifier as a black box.

From this, you might get the impression that the exact value of the open loop gain (which means the gain without negative feedback) is critically important. However, this is *not* true, as you will see later on.

In the case of opamps made from bipolar transistors, the input impedance is on the order of 1 MΩ. If field effect transistors (FETs) are used for the input stage of the opamp, the input impedance will be several orders of magnitude higher.

The maximum supply voltage is usually between ±16 V and ±22 V, but it can be as high as ±150 V with high-voltage opamps. The maximum output current is typically on the order of 20 mA, although power opamps can handle several amperes if they are sufficiently cooled. The cut-off frequency will be discussed in detail later on.

### Amplifier Circuits

The two commonly used forms of amplifier circuit with opamps are the inverting amplifier and the non-inverting amplifier. As indicated by its name, the inverting amplifier reverses the polarity of the input signal, while the non-inverting amplifier does not.

## The Inverting Amplifier

**Figure 2** shows the basic schematic diagram of the inverting amplifier. As you can see, it requires only two external components in the form of two resistors. For audio applications, a pair of capacitors is usually also added.

How much gain does this circuit provide? The non-inverting input is tied to ground, so the open loop gain is the initial gain-determining parameter:

$$V_0 = -\frac{U_{out}}{U_V} \Rightarrow U_V = -\frac{U_{out}}{V_0}$$

The following applies to the input voltage:

$$U_{in} = U_{R1} + U_V = I_{R1} \cdot R_1 - \frac{U_{out}}{V_0}$$

The gain is defined as the output voltage divided by the input voltage. If you enter the result obtained above into this formula, you have:

$$V = \frac{U_{out}}{U_{in}} = \frac{U_{out}}{I_{R1} \cdot R_1 - \frac{U_{out}}{V_0}}$$

The following formula applies to the currents flowing into and out of the node ahead of the inverting input:

$$I_{R1} = I_{R2} + I_{Ri} = I_{R2} + \frac{U_V}{R_i} = I_{R2} - \frac{U_{out}}{R_i \cdot V_0}$$

Entering this in the formula for the gain gives you:

$$V = \frac{U_{out}}{U_{in}} = \frac{U_{out}}{I_{R2} \cdot R_1 - \frac{U_{out} \cdot R_1}{R_i \cdot V_0} - \frac{U_{out}}{V_0}}$$

The network with resistor  $R_2$  is governed by the following formula:

$$U_{out} = U_V - U_{R2} = -\frac{U_{out}}{V_0} - I_{R2} \cdot R_2$$

This can be solved for the current  $I_{R2}$ :

$$I_{R2} = \frac{U_{out}}{V_0 \cdot R_2} - \frac{U_{out}}{R_2}$$

Ultimately, if you enter this into the formula for the gain and then cancel out  $U_{out}$  wherever it appears, you arrive at the exact formula for the gain of the inverting amplifier:

$$V = \frac{1}{-\frac{R_1}{V_0 \cdot R_2} - \frac{R_1}{R_2} - \frac{R_1}{R_i \cdot V_0} - \frac{1}{V_0}}$$

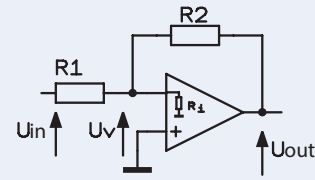


Figure 2: The inverting amplifier.

Now things start to get interesting. Here, you can reason as follows: The open-loop gain is usually much larger than the ratio  $R_2/R_1$ , while the internal impedance of the opamp is also much larger than the values of the external resistors. With this in mind, you can obtain a good approximation for the gain, in simplified form, as:

$$V \approx -\frac{R_2}{R_1}$$

This is a very nice, handy formula for calculating the gain of the inverting operational amplifier. Let's use a numerical example to see how much the difference actually is in practice. Assuming an amplifier circuit with  $R_2 = 100 \text{ k}\Omega$  and  $R_1 = 10 \text{ k}\Omega$  and using the simplified formula, we arrive at a gain of  $-10$ . According to the previously derived exact formula (assuming an open loop gain of  $100,000$  and an input impedance of  $1 \text{ M}\Omega$ ), we arrive at a gain of approximately  $-9.99889$  (if you're interested, you can do the calculation yourself). This means the difference is several orders of magnitude smaller than the tolerance of precision resistors, so it can easily be ignored. (By the way, we have totally ignored a number of other factors, such as the output impedance, temperature drift, and so on.)

## The Non-Inverting Amplifier

**Figure 3** shows the basic schematic diagram of the non-inverting amplifier. In this minimalistic version, here you also need only two external resistors.

The following formula applies to the output voltage of an opamp:

$$U_{out} = (U_{in} - U_V) \cdot V_0 \Rightarrow (U_{in} - U_V) = \frac{U_{out}}{V_0}$$

The gain of the overall circuit can be expressed as:

$$V = \frac{U_{out}}{U_{in}} = \frac{U_{out}}{(U_{in} - U_V) + U_V}$$

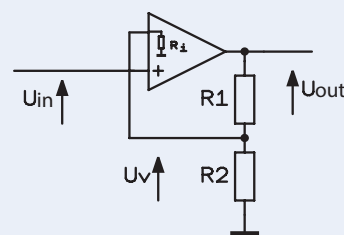


Figure 3: The non-inverting amplifier.



In the course of the derivation, it will become clear why we have included the apparently meaningless term  $U_V$  in the above formula. If you reverse this formula and enter the formula for the output voltage of the opamp, you obtain the following formula (note that here we have omitted a few steps to avoid making this explanation too technical):

$$\frac{1}{V} = \frac{U_{out}}{V_0 \cdot U_{out}} + \frac{U_V}{U_{out}} = \frac{1}{V_0} + \frac{U_V}{U_{out}}$$

Here,  $R_1$  and  $R_2$  form a voltage divider that is loaded by the resistance,  $R_i$ . For  $U_V$ , this means:

$$U_V = \frac{U_{out} \cdot (R_2 \parallel R_1)}{R_i + (R_2 \parallel R_1)} = \frac{U_{out}}{R_i \cdot \left( \frac{1}{R_2} + \frac{1}{R_i} \right) + 1}$$

If you enter this into the (reversed) formula for the gain, you get:

$$\frac{1}{V} = \frac{1}{V_0} + \frac{1}{R_i \cdot \left( \frac{1}{R_2} + \frac{1}{R_i} \right) + 1}$$

Reversing this result gives you the exact formula for the gain:

$$V = \frac{1}{\frac{1}{V_0} + \frac{1}{R_i \cdot \left( \frac{1}{R_2} + \frac{1}{R_i} \right) + 1}}$$

In this case as well, you should bear in mind that the open-loop gain is much larger than the ratio  $R_2/R_1$ , and in addition,  $R_i$  is very

much larger than  $R_2$ . If you apply these considerations to the above formula, you ultimately arrive at the simplified formula for the gain of the non-inverting opamp amplifier:

$$V \approx \frac{R_1}{R_2} + 1$$

This is also a very nice, compact formula, and often you will not even need a calculator to use it.

In the next installment, we will dig a bit deeper into opamp theory, and after that, we will finally turn our attention to real opamp circuits.

*Translated by Kenneth Cox — 230756-01*

Editor's Note: This series of articles, *Starting Out in Electronics*, is based on the book, *Basiskurs Elektronik*, by Michael Ebner, which was published in German and Dutch by Elektor.

#### Questions or Comments?

If you have any technical questions or comments about this article, feel free to contact the Elektor editorial team at [editor@elektor.com](mailto:editor@elektor.com).



#### Related Products

- > **B. Kainka, Basic Electronics for Beginners (Elektor, 2020)**  
Book: [www.elektor.com/19212](http://www.elektor.com/19212)  
Ebook: [www.elektor.com/19213](http://www.elektor.com/19213)

# Join our Community

[www.elektormagazine.com/community](http://www.elektormagazine.com/community)

## ESP Library Recommendations

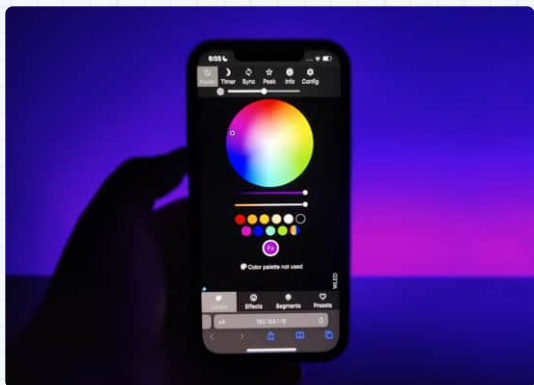
By Saad Imtiaz and Jean-François Simon (Elektor Lab)

Looking for ESP-related libraries? Check out these recommendations from Elektor's engineering team.

### WLED (github.com/aircoookie)

WLED, by Christian Schwinne, is a fast and feature-rich implementation of an ESP8266/ESP32 web server to control NeoPixels (WS2812B, WS2811, SK6812) LEDs. It also supports SPI-based chipsets like the WS2801, APA102 and many more. The library features over 100 special effects for NeoPixels, 50 palettes, FastLED noise effects, and more! It has a modern UI with advanced controls. Configuration is done via the network. Among notable advantages, it supports up to 10 LED outputs per instance and can work with RGBW strips. Up to 250 user presets can be used to save and load colors/effects easily, and support cycling through them. Presets can also be used to automatically execute API calls. There is also a Nightlight function, which gradually dims down the LEDs. Regarding updates, Over the Air (HTTP + ArduinoOTA) is supported and can be password protected. If you are looking to control your RGB lights or looking to give your room a new theme, you'll go ahead for this one! Also see Adafruit\_NeoPixel below.

<https://github.com/Aircoookie/WLED>



Welcome to ExpressLRS!  
The best RC link that you can build yourself

### ExpressLRS (github.com/ExpressLRS)

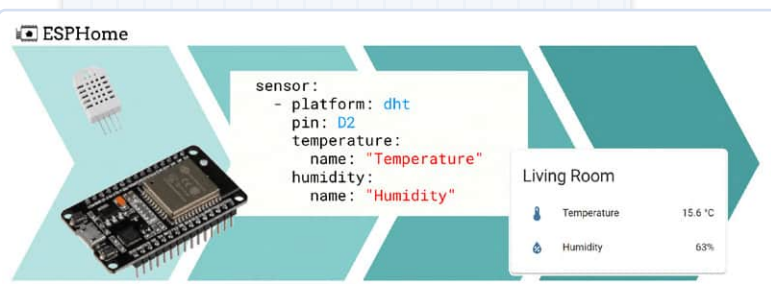
ExpressLRS is an open-source radio link for RC (Radio Control) applications. It is designed to give excellent performance using the SX127x/SX1280 LoRa chip combined with an Espressif or STM32 microcontroller. ExpressLRS allows users to benefit from a good range and very low latency, thanks to LoRa modulation as well as reduced packet size. ExpressLRS supports hardware from many manufacturers: AxisFlying, BETAFPV, Flywoo, FrSky, HappyModel, HiYounger, HGLRC, ImmersionRC, iFlight, JHEMCU, Jumper, Matek, NamimnoRC, QuadKopters and SIYI. It features 1 kHz packet rate, telemetry, Wi-Fi updates, two frequencies for the RC link (2.4 GHz or 900 MHz) and more. It is undoubtedly very interesting for many RC projects, with different hardware suited to different requirements.

<https://github.com/ExpressLRS/ExpressLRS>

### ESPHome (github.com/esphome)

ESPHome is an open-source framework that simplifies the process of configuring and managing devices based on ESP8266 and ESP32. It enables users to create custom firmware for these devices without the need for extensive programming. With ESPHome, you can define device functionality and settings using a user-friendly YAML configuration file, making it accessible to both beginners and experienced developers. Key features of ESPHome include support for a wide range of sensors and components, automatic discovery and integration with popular home automation platforms like Home Assistant, and over-the-air (OTA) updates for seamless firmware upgrades. It fosters the creation of DIY smart home solutions, allowing users to tailor their devices to specific needs, such as temperature monitoring, lighting control, or home security. ESPHome has gained popularity in the smart home community. Be sure to check it out, if you don't know it already!

<https://github.com/esphome/esphome>



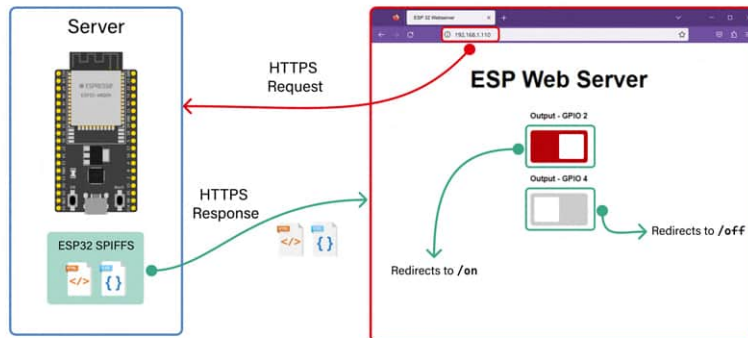


## ESPAsyncWebServer (github.com/me-no-dev)

ESPAsyncWebServer is an Asynchronous HTTP and WebSocket Server for ESP8266 used in the Arduino environment.

It can be used with PlatformIO and Arduino IDE. Using an asynchronous network means that you can handle more than one connection at the same time. You are called once the request is ready and parsed. When you send the response, you are immediately ready to handle other connections while the server is taking care of sending the response in the background. The processing speed is very high! ESPAsyncWebserver provides an easy-to-use API and is compatible with HTTP Basic and Digest MD5 Authentication (by default), as well as Chunked Response. There are various plugins to offer various benefits, such as: different locations, sending events to the browser, advanced URL rewrite and more.

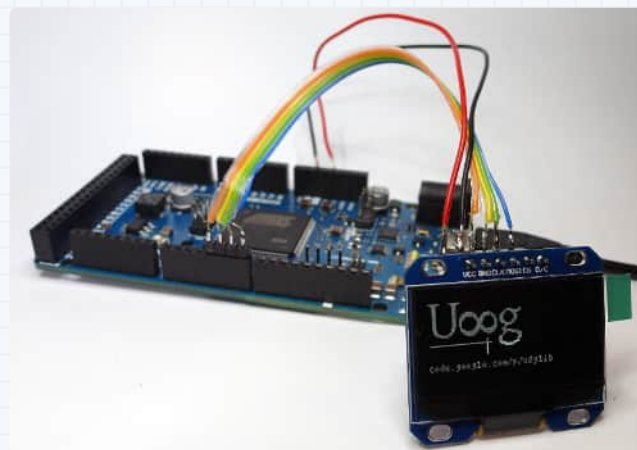
<https://github.com/me-no-dev/ESPAsyncWebServer>



## U8glib (github.com/olikraus)

U8glib is a graphics library with support for many monochrome displays. The latest version of U8glib for Arduino is available in the Library Manager and can also be downloaded from GitHub. It is compatible with Arduino (ATmega and ARM), AVR, ARM (with example for LPC1114) and many other devices such as SSD1325, ST7565, ST7920, UC1608, UC1610, UC1701, PCD8544, PCF8812, KS0108 and more. This library supports many fonts (both monospaced and proportional), mouse cursor mode, landscape and portrait mode... All in all, a pretty comprehensive library that will come in handy for many projects.

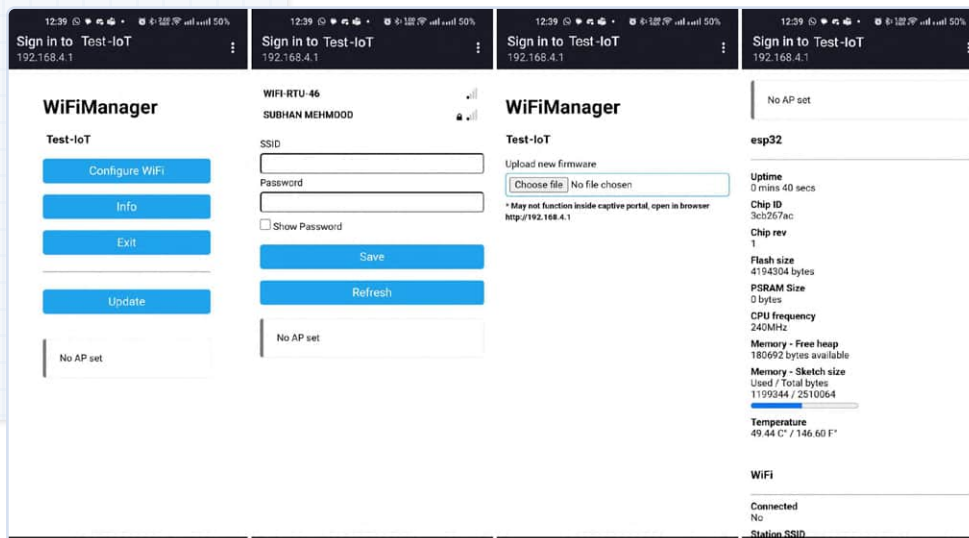
<https://github.com/olikraus/u8glib>

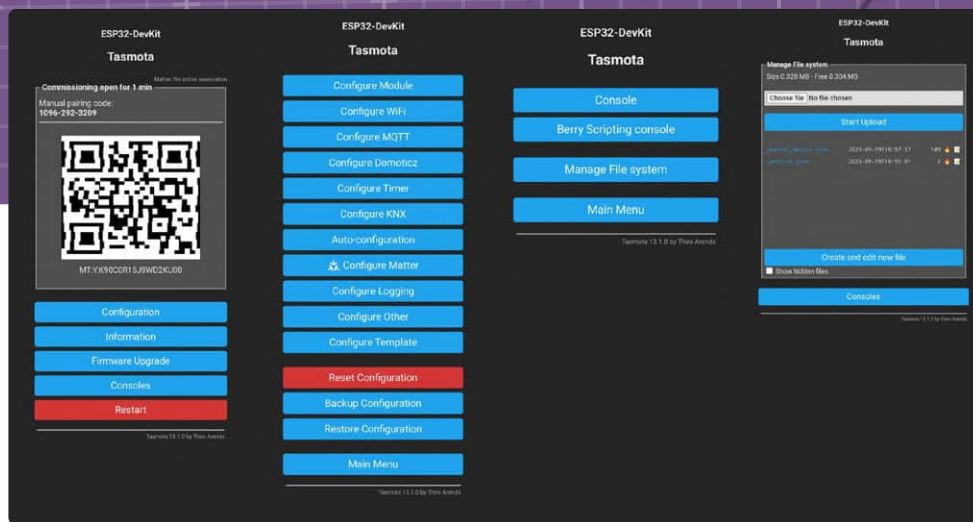


## WiFiManager (github.com/tzapu)

WiFiManager is a connection manager for ESP-based projects. It gives the user the possibility to configure Wi-Fi credentials and custom parameters at runtime, using a captive portal. How does it work? When your ESP starts up, it sets it up in Station mode and tries to connect to a previously saved Access Point. If this is unsuccessful, the firmware then moves the ESP into Access Point mode and spins up a DNS and WebServer. Then, using any Wi-Fi enabled device with a browser (computer, phone, tablet), you can connect to the newly created Access Point and set the credentials. The ESP will then connect to the network of your choice! There are also options to change this behavior or manually start the configuration portal and the web portal independently. It is also possible to run them in non-blocking mode.

<https://github.com/tzapu/WiFiManager>





## Tasmota ([github.com/arendst](https://github.com/arendst))

Tasmota is an open-source firmware designed for ESP devices, specifically tailored for home automation and smart homes. It provides support for ESP8266, ESP32, ESP32-S, or ESP32-C3 microcontrollers. Theo Arends initiated this project in 2016 under the name Sonoff-MQTT-OTA. His primary objective

was to equip ESP8266-based devices produced by ITEAD (Sonoff) with MQTT and over-the-air (OTA) firmware capabilities. What began as a straightforward solution to modify a cloud-bound Sonoff Basic into a locally manageable device, has evolved into a full ecosystem suitable for almost any ESP-based device. Tasmota is developed for PlatformIO, facilitating easy configuration through a web-based user interface (webUI). Updates can be performed wirelessly over the air. Users can automate devices using timers or custom rules. Full local control and expandability are achievable through MQTT, HTTP, Serial, or KNX protocols.

<https://github.com/arendst/Tasmota>

## Esp32FOTA ([github.com/chrisjoyce911](https://github.com/chrisjoyce911))

esp32FOTA is a simple library to add support for over-the-air (OTA) updates to your ESP32 project. This library will try to access a JSON file hosted on a web server, analyze its contents to determine if a newer firmware version is available, and if so, download and install it. To make this update process work, you need a web server with a valid JSON file on it (which can be optionally compressed with zlib or gzip). The complete list of detailed requirements (including details about HTTPS) is available on GitHub. This library is quite comprehensive and includes support for compressed firmware, SPIFFS/LittleFS partition updates, compatibility with various file systems for certificate and signature storage. Besides, esp32FOTA also allows web-based updates (with a web server), batch firmware synchronization and the ability to force firmware updates. Finally, the library also has some advanced features such as signature checks for downloaded firmware images, signature verification, and support for semantic versioning. In a nutshell, it's a library well worth discovering if you want to use OTA updates for your next project.

<https://github.com/chrisjoyce911/esp32FOTA>

## chrisjoyce911/ esp32FOTA

Experiments in firmware OTA updates for ESP32 dev boards

21 Contributors 7 Issues 301 Stars 77 Forks



## Willow ([github.com/toverainc](https://github.com/toverainc))

Willow is an Espressif IDF (IoT Development Framework) project used to turn an ESP32-S3-BOX into a versatile voice assistant with various features. It can be triggered by custom wake words, such as "Hi ESP", and listens for voice commands, automatically detecting when to start and stop listening. Willow integrates well with popular home automation platforms like Home Assistant, openHAB, and generic REST APIs. This voice assistant performs admirably in challenging environments, recognizing wake words and speech from distances of approximately 25 feet (7.62 m). It ensures high-quality audio with features like automatic gain control, echo cancellation, noise reduction, and source separation. In Wi-Fi congested environments, Willow can employ audio compression to optimize airtime usage. Additionally, Willow offers on-device speech command recognition, allowing you to configure up to 400 commands locally. Alternatively, you can use the open-source Willow Inference Server for more extensive speech transcription capabilities. In conclusion, it's a very impressive project!

<https://github.com/toverainc/willow>

## Adafruit NeoPixel ([github.com/adafruit](https://github.com/adafruit))

This is an Arduino library for controlling, well, NeoPixels. They are, in Adafruit's jargon, individually addressable RGB color pixels and strips based on the WS2812, WS2811 and SK6812 LEDs. These LEDs each have an integrated driver and use a single-wire control protocol. These tend to be a bit tricky to use due to tight timing requirements for the control signals and specific protocol. With this library, things get much easier! Adafruit develops and maintains it for free. In return, users may decide to purchase some of their products. The main goals of Adafruit\_NeoPixel are being easy to use and being flexible in terms of supported chipsets. The library supports AVR (ATmega and ATtiny), Teensy, Arduino Due, Arduino 101, ATSAM21/51, Adafruit STM32 Feather, ESP8266, ESP32, Nordic nRF51/52 as well as some ICs in the XMC series from Infineon. Among other things, the many well-tested functions such as `setBrightness()`, `setPixelColor()` and 12 more make this library very useful for quick prototyping.

[https://github.com/adafruit/Adafruit\\_NeoPixel](https://github.com/adafruit/Adafruit_NeoPixel)



## LVGL ([github.com/lvgl](https://github.com/lvgl))

To stay on the subject of dashboards and graphics, LVGL (**Figure 12**) is an embedded graphics library, enabling the creation of graphical user interfaces for a wide range of microcontrollers, including, of course, ESP32 and Arduino (full list on GitHub). This library is written in C and is portable, i.e., it doesn't rely on any external dependencies. It can be used with or without a real-time operating system (RTOS) and has extensive display support, monochrome, ePaper, OLED, etc. It needs 32 kB of RAM and 128 kB of Flash memory. LVGL boasts a wide array of built-in widgets, styles, layouts, and more, making it highly customizable. Animations, anti-aliasing, opacity control, smooth scrolling, shadows, image transformation... The list goes on. Various input methods like mouse, touchpad, keypad are supported. Make and CMake are both supported, allowing you to develop on a PC and use the same UI code on embedded hardware, which can be an interesting feature for some users.

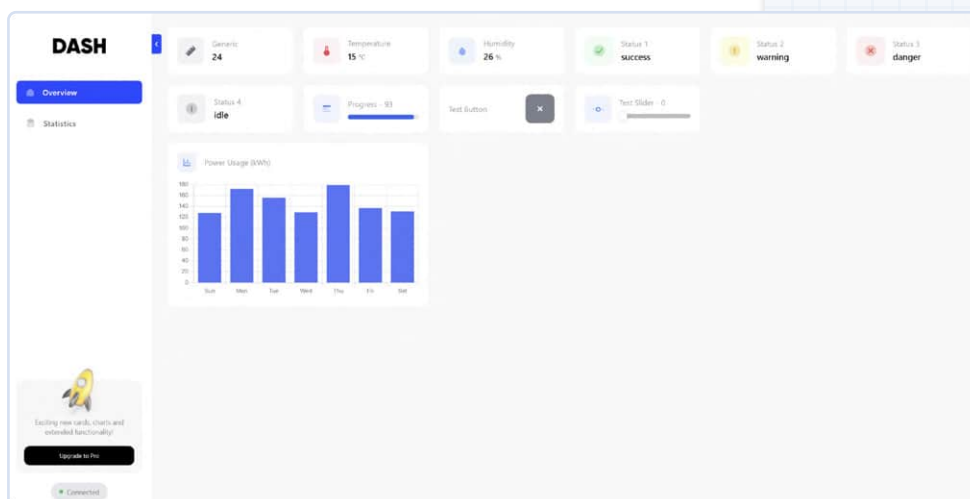
<https://github.com/lvgl/lvgl>

## ESP-DASH ([github.com/ayushsharma82](https://github.com/ayushsharma82))

ESP-DASH is a high-speed library designed to develop a functional and real-time dashboard tailored for ESP8266 and ESP32 microcontrollers. It is developed by GitHub user ayushsharma82 and other contributors, and it's currently at Version 4. This library comes with many features, including charts, display cards, interactive buttons, and numerous other components, enabling the creation of beautiful dashboards. These will be accessible locally and will not necessitate an internet connection, as all data is stored on the chip. ESP-DASH boasts automatic webpage generation and real-time updates for all connected clients.

Users are not required to delve into HTML, CSS, or JavaScript, as it offers a user-friendly C++ interface. It provides pre-configured components for managing your data and offers flexibility—you can effortlessly add or remove components directly from the webpage. Furthermore, it comes with built-in support for charts, enhancing its functionality.

<https://github.com/ayushsharma82/ESP-DASH>



230588-01



# Piezoelectric Devices

## Peculiar Parts, the Series

By David Ashton (Australia)

The piezoelectric effect has been known since the 1880s, but how it got from a scientific curiosity to a staple in modern electronics is a story of innovation and discovery. This unique property of certain crystals has led to its widespread use in everything from audio transducers to precision instruments in astronomy and digital imaging.

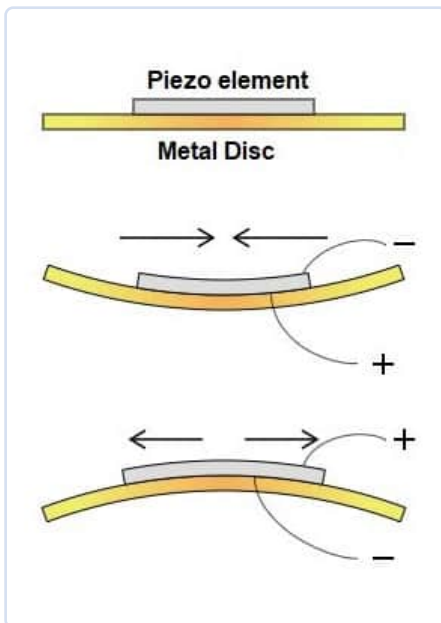


Figure 1: Piezo effect. (Source: Sonitron Support - CC BY-SA 3.0, [commons.wikimedia.org/w/index.php?curid=115322872](https://commons.wikimedia.org/w/index.php?curid=115322872))

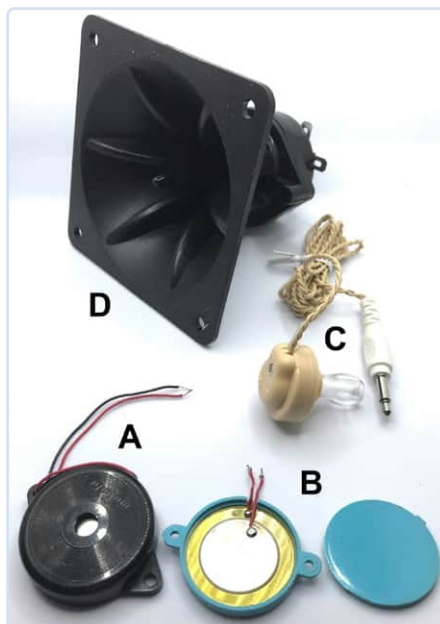


Figure 2: A selection of piezo sounders. These are just piezo elements and have to be driven with an AC signal. A: a typical Piezo sounder. B: A similar one opened to show the construction. C: A crystal earpiece. D: a 100 W Piezo horn tweeter for hi-fi use.

If certain crystalline ceramic substances have electrodes plated on two opposite sides of the crystal, and a voltage is applied to it, the crystal will deform slightly. The effect was first demonstrated in 1880 by brothers Pierre and Jacques Curie, though it had been mathematically predicted in 1861 by Gabrielle Lippmann. For the next few decades, however, it was regarded as nothing more than a laboratory curiosity.

This effect has many uses in electronics. Probably the one most familiar to electronics enthusiasts is audio transducers. I remember over 50 years ago making my first Crystal Radio, using a crystal earpiece — which is just a piezo transducer mounted on a small metal disc. **Figure 1** shows the principle of operation. They have the advantage of being fairly high impedance, important for a crystal set, as it does not load it much. They are also very thin, which leads to their use in those greeting cards which play a melody when you open them! You can get larger devices which have a half-decent





Figure 3: A selection of piezo buzzers (beepers). These have electronics inside to drive the element. Back: A 12 V Piezo siren. Middle: Three different piezo buzzers. Front: a multi-voltage, AC/DC piezo buzzer for industrial switchboard panel use.



Figure 4: A piezo igniter out of a gas lighter. Press the button at the end, and it will “click” and generate a high voltage pulse to produce a spark across two electrodes.

frequency response, and some Hi-Fi tweeters use piezo technology (**Figure 2**). Most of these devices have a resonant frequency at which they are most efficient, and use of this is made in sounders, which have an oscillator built into them which drives the disc at or near the resonant frequency. They are used as “beepers” and alarm sounders (**Figure 3**) to give warning signals and have the advantage of being very efficient and consuming little power for the sound they put out.

Other applications of piezo transducers include inkjet printers (a small movement is used to eject a droplet of ink) and astronomical telescopes — piezo actuators are used to apply very small corrections to the shape of the heavy mirror used to gather the light, to minimise distortion. They can apply relatively large forces over very small distances. One other application is image stabilization in digital cameras, where a small accurate movement with a fast response time is needed.

The effect is reversible — that is, if a piezo element is bent or deformed, it will generate a voltage across its electrodes. This is used in gas igniters, where an element is deformed slowly and then rapidly allowed to go back to its usual position by means of a ratchet mechanism (**Figure 4**). As it does so, a pulse of very high voltage is generated — enough to produce a spark across two electrodes to ignite the gas. They are also used in vibration sensors, where their small size and low cost are of great advantage. ◀

230684-01

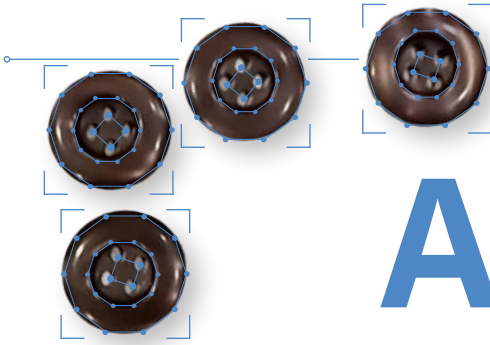
### Questions or Comments?

If you have technical questions or comments about this article, feel free to contact the Elektor editorial team by email at [editor@elektor.com](mailto:editor@elektor.com).



### About the Author

David Ashton was born in London, grew up in Rhodesia (now Zimbabwe), lived and worked in Zimbabwe, and now lives in Australia. He has been interested in electronics since he was “knee-high to a grasshopper.” Rhodesia was not the center of the electronics universe, so adapting, substituting, and scrounging components were skills he acquired early (and still prides himself on). He has run an electronics lab, but has worked mainly in telecommunications.



# A Smart Object Counter

Image Recognition Made Easy with Edge Impulse

By Somnath Bera (India)

Discover how you can transform a Raspberry Pi and a camera into a smart object-counting tool using the Edge Impulse platform! This fun and accessible project demonstrates the ease of starting with Edge Impulse on a Raspberry Pi, perfect for both beginners and more seasoned enthusiasts.

Edge Impulse specializes in providing tools and platforms for developing machine learning models for edge computing, running particularly on embedded devices. Edge computing involves processing data near the source of data, instead of relying on a remote server. That's perfect to implement on a Raspberry Pi! In this example, we'll count small objects, in this case ordinary buttons found on fabric.

Machine learning platforms such as Edge Impulse use so-called models, which are specific types of algorithms used for data analysis and pattern recognition. These models are trained to recognize patterns, make predictions, or perform tasks based on input data.

Classifying of objects is easily possible by Edge Impulse models. You can identify between a man and animals, between bicycles and cars, and so on. In addition, you can easily count one type of object among other types of objects. All you need to have is a good quality camera, sufficient light, proper focus, and, lastly, a moderately built computer (a Raspberry Pi 3 or Raspberry Pi 4 is good enough), and you are ready to go for counting.

From the start, this project was aimed to be installed on the micro-controller (MCU) level — an Espressif ESP32, an Arduino Nicla vision, etc. And that's why it was made for a very small area of

counting (120 pixel × 120 pixel) with a relatively small button as an object of interest. Ultimately, it transpired that even for the smallest area, the MCUs were no match at all. The machine learning models are pre-trained on Edge Impulse servers and a so-called model file is generated to be stored on the embedded device. Here, the model file itself is about 8 MB! Therefore, the project was finally installed in a Raspberry Pi computer, where it works easily.

## Knowledge and Wisdom

If you know Edge Impulse [1], then believe me, half of your job is already done. For the rest, you just need to tweak your model to fine-tune it for an acceptable level of performance. A computer AI model works like a child. Imagine how you learned things like, "A is for apple" and "B is for ball." You were shown an apple from various angles, and then you were taught to name it "apple." The same went for "ball." Now, from all possible angles, a child will identify an apple and a ball fairly easily! And so it is with AI, which can identify them easily.

Now consider there is a basket where seemingly apple-sized balls and ball-sized apples are mixed and all look the same from the point of view. Being a child, what would you do? With your only knowledge of apple and ball, you would simply miss! AI would miss too. But consider the fruit basket is displayed by a fruit and vegetable vendor. Then, in all probability, neither of them is a ball! And some or all of them could be apples. This "trick" of connecting an apple with a fruit and vegetable seller is called wisdom, which you cannot expect either from a child or from AI unless it is specifically taught otherwise. However, human beings have learned, over the years, many more associated things that eventually have given us enough wisdom to connect apple with a fruit and vegetable seller.

However, AI is improving so fast that someday it will have wisdom to work on. For now, you must teach the ML model for apple and ball from all possible angles to perform them without any confusion (e.g., the texture profile of an apple, its stem, its creases on its body, look from top and bottom and much more). In any case,

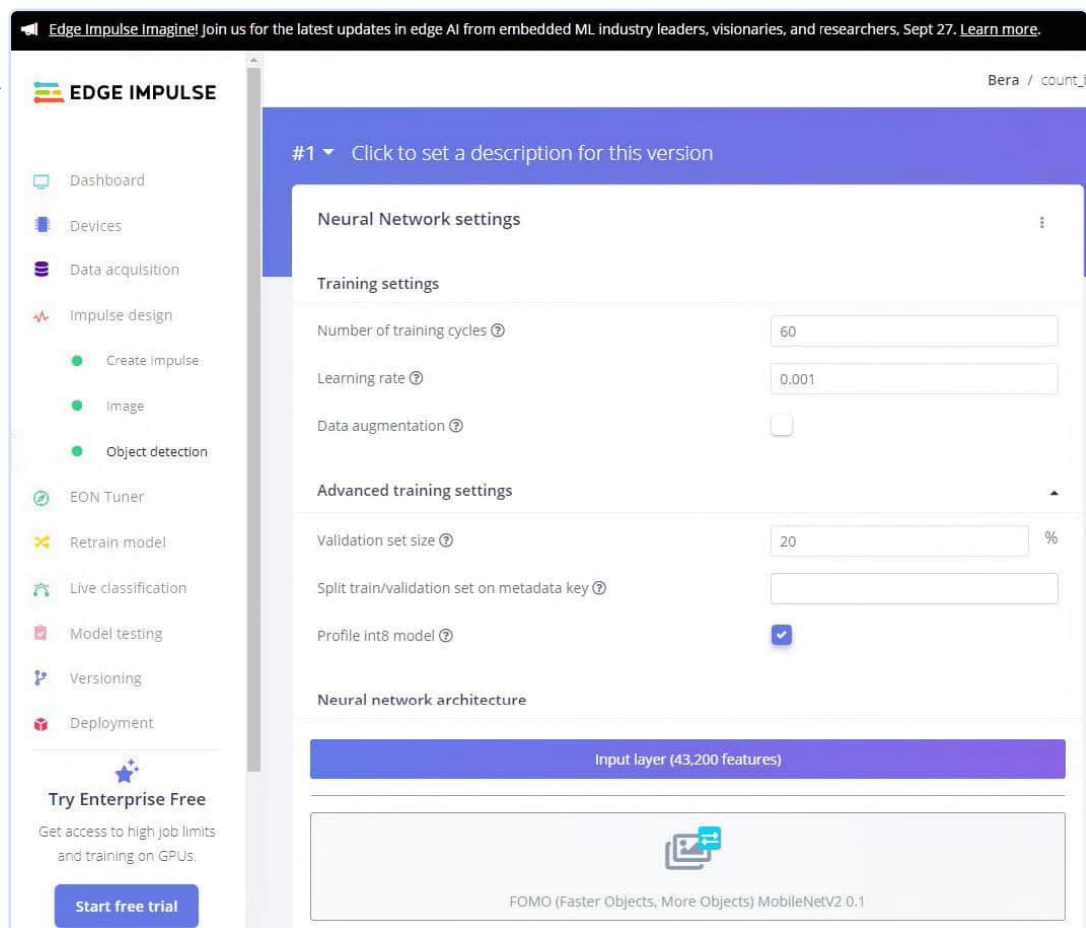


Figure 1: Neural network settings.

many different models with different capabilities are available in Edge Impulse for testing and experimenting.

## Getting Started with Edge Impulse

First, open an account in Edge Impulse [1], which requires an email ID. Collect similar types of buttons, a handful number. If you open the site from a Raspberry Pi computer, using the camera of the Raspberry Pi computer (either USB connected or cam port connected), you can collect images of buttons from several angles (which is required while the model is deployed in a real working field). Edge Impulse has also provisions to connect your cell phone or laptop as an input device for collecting data, which is also more convenient for data acquisition in the Edge Impulse project.

## The Project

The Edge Impulse project is broadly divided into the following steps, all of which must be followed on the Edge Impulse website.

1. Data acquisition: This could be images, sound, temperatures, distances, etc. Part of the data is separated as test data, while all other data is used as training data.
2. Impulse design: The main part of it being labeled *Create Impulse*. In this context, an “impulse” refers to a pipeline or workflow for creating a machine learning model. This impulse encompasses several stages, including tweaking of input parameters associated with the data that were just collected, signal processing, feature extraction, and the machine learning model itself. “Features” are individual measurable properties or characteristics of a phenomenon being observed. Essentially, features are the data

attributes used by models to detect patterns and make decisions. The Impulse pipeline is subdivided:

- Input parameters: image (width, height), sound (sound parameters)
- Processing block: how to process the input data
- Learning block: object data of this model

You have to select and configure these three steps.

3. Image processing: Generate Features of the collected images.
4. Object detection: Select your Neural Network model and train the model.

For the final part — the object detection part — your expertise is needed, or I would rather call it trial and error effort, so that the accuracy of the model becomes 85% or above. At times, you have to remove some bad images (aka outliers) from the model to improve its efficiency.

There are a handful of models in which you can try and see the accuracy level of the model. Anything above 90% is great; but certainly, it should not be 100% accurate! If it is so, then you have something wrong with your data. It could be very little data or insufficient features are there. Recheck and retry again for that case! For this project, the accuracy was 98.6%. Certainly, our number of data (about 40) was small. However, for a starter project, this is pretty good (see **Figure 1**). The files for this project are available on the Elektor Labs page of this project [4].

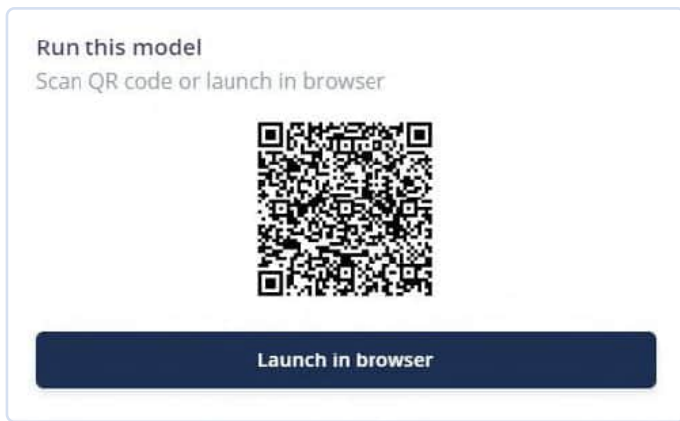


Figure 2: Scan QR code to run this model.

## Model Testing

You can test your model on the test data first. Start there, and then point your device to the real-life data and see if it works!

In the dashboard of the Edge Impulse opening page, the test feature is available. You can straight away run the model in the browser, or you can use your smartphone to test it. To do so, Edge Impulse offers a QR code to scan with your smartphone (**Figure 2**). Point the camera to the buttons (**Figure 3**, **Figure 4**, and **Figure 5**) and see whether it can count them or not!

## Raspberry Pi Deployment

To run the model on a Raspberry Pi computer, you have to download the \*.eim file. But unlike for other hardware (Arduino, Nicla Vision, or ESP32 where you can download directly), in the case of Raspberry Pi, you have to install Edge Impulse on the Raspberry Pi computer first. From inside that *edge-impulse-daemon* software, you have to download this file. But don't worry, Edge Impulse has devoted a full page to installing Edge Impulse on Raspberry Pi. There are a few dependencies to install first. Look at [2]. It's pretty easy. The process is well described.

OK, so after you install Edge Impulse on the Raspberry Pi computer, the fun begins. Remember to keep the Raspberry Pi connected to the Internet.

Run the command `edge-impulse-linux-runner` from the Raspberry Pi terminal. This will start a wizard which will ask you to log in, and choose an Edge Impulse project. If you want to switch between projects later, run that command again with the `--clean` option. This command will automatically compile and download the AI model of your project and then start running on your Raspberry Pi. Show the buttons to the camera connected to your Raspberry Pi and it should count them. That's good! In the following, we will modify the system using Python and a voice synthesizer which, after counting, speaks up the number of buttons it got to count.

## Deploy Model in Python

In the above deployment, it would work as it is intended in the Edge Impulse model. To make it work for your special purpose — for example to sound an audio alarm or light an LED when the count reaches “2 or more” — you have to find some other means! Here comes Python 3 to help you out. *Linux-sdk-python* needs to be installed on your Raspberry Pi computer.

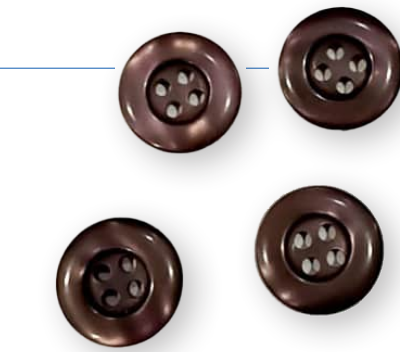


Figure 3: Data Acquisition: Sample-1.



Figure 4: Data Acquisition: Sample-2.



Figure 5: Data Acquisition: Sample-3.

The Edge Impulse SDK Software Development Kit (SDK) is available for many models, including Python, Node.js, C++, etc. Check the SDK Python page [3].

Once the *linux-sdk-python* is installed, go to the *linux-sdk-python/examples/image* directory and run the Python file for image identification. Don't get confused. In the example directory, there are three subdirectories — one each for audio data, image data and custom data. In the image directory, the video classification file is also available for video input data. The custom directory is for customizing other kinds of data (for experts only!).

Now run the command:

```
python3 classify-image.py /home/bera/downloads/model.eim
```

The model file \*.eim is to be loaded from the respective directory of its location. If you prefer, you can copy it into the SDK directory as well!



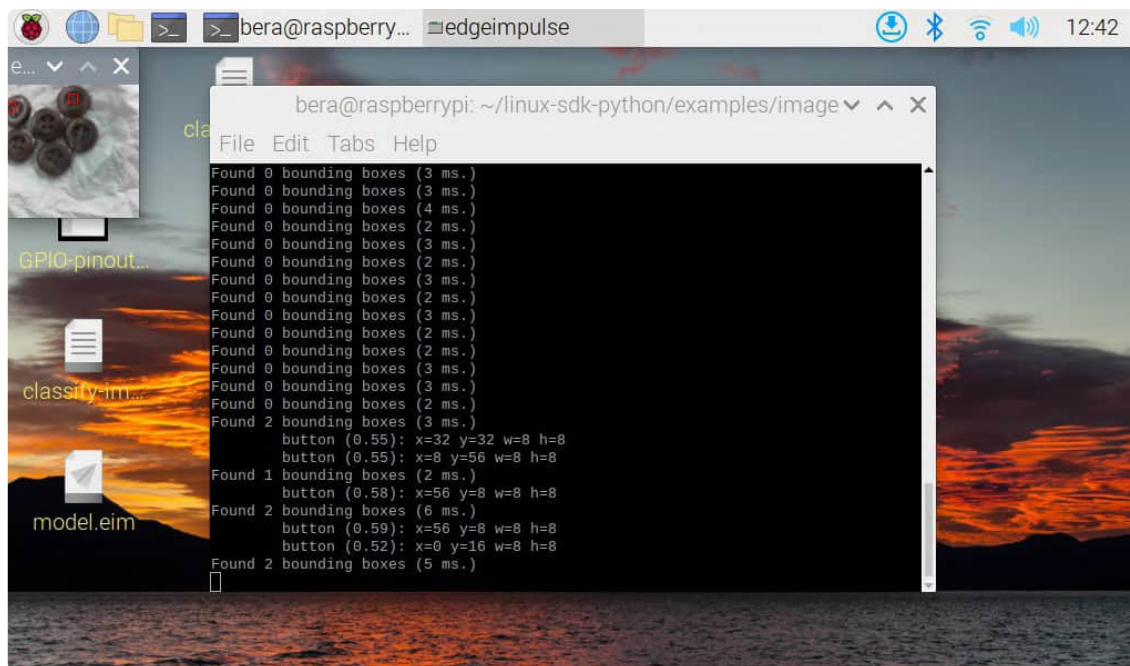


Figure 6: Four buttons are missed because of inadequate focus and light issue.

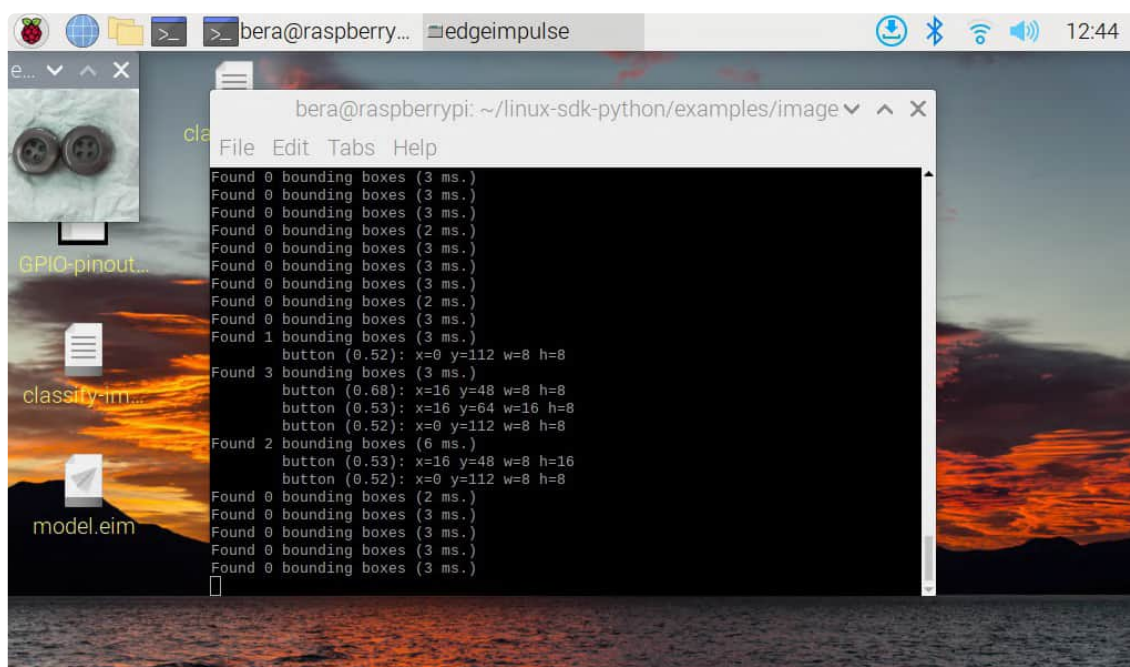


Figure 7: After removing one button, the model counted it right at first.

That's how you have to load the Python file with the downloaded `model.eim` file. The program will automatically find the camera module (USB-connected or Cam-Port connected) and will start running! In the top-left corner, a small  $120 \times 120$  camera window will open, and the identified buttons will be marked with a small red marker. The numbers identified will be shown on the terminal. Please ensure sufficient light is available, and the camera is properly focused for the buttons. This is particularly important for cheap cameras. That's why if you run the model on your smartphone it produces far superior images and counts far more quickly. Nonetheless, ensure proper light and focus, and it will produce better results.

In the following screenshot pictures of the Raspberry Pi computer, on the top left, the small  $120 \times 120$  exposure window is visible wherein all three buttons are identified and counted by the model.

See the distinct visible red marker on all three buttons!

In **Figure 6**, four buttons are missed because of inadequate focus and light issue. The camera also does not have a stand to be fixed for this work! That's why I recommend getting the camera fixed on a stand, like in a "microscope setup." Be sure not to look at an angle from the top.

In **Figure 7**, I removed one button and the model counted it right first. It read two buttons, but at the moment I pressed the *pr-screen* button, it got misaligned and missed. Also, ensure the camera has a long cable (ribbon cable — see my prototype in **Figure 8**) to move around. This cable is available on Amazon. However, once the camera is fixed on a stand with a good amount of light/daylight, it will work infallibly.



Figure 8: My prototype.

## Customize Your Model

Please have a look at the *classify-image.py* file. It's a simple Python file that can be tailor-made with little difficulty. In this Python file, I've added an *espeak* module such that the moment it finds a button(s), it speaks out the number of button(s) it finds. To install *espeak* on your Raspberry Pi, run the command:

```
sudo apt-get install espeak
```

Refer to **Listing 1** with the Python file including my modifications.

*Espeak* is a stand-alone text-to-speech module for Python. It does not require an Internet connection to work.

## Modified Run

Now you have modified the Python program. If you run the Python file now, it will locate the button (on the top left, a small 120 × 120 camera port will open), and the numbers will be shown on the terminal window and the associated speaker will speak out the number: "Found five buttons / Found two buttons," etc. If you want to run some relay, light an LED, etc., import the GPIO library of the Python and then fire the associated GPIO to run the relay, etc. However, for running a relay, you have to use a switching transistor to increase the amount of current required for running the relay.

## Aftermath

Edge Impulse started in 2019 with an objective to enable developers to create the next generation of intelligent devices. Since then, AI-powered programs and devices have appeared on ESP32, Jetson Nano, Raspberry Pi, Orange Pi, Maixduino, OpenMV, Nicla Vision, and many more. This trend will further improve in the coming days! Gone are the days of supercomputers or big-brand computers. Small, low-power modular devices are covering that space fast. And who knows? Maybe soon we will have the built-in wisdom installed and ready right out of the package! ◀

230575-01

## Questions or Comments

If you have technical questions or comments about this article, feel free to contact the author by email at [berasomnath@gmail.com](mailto:berasomnath@gmail.com) or the Elektor editorial team at [editor@elektor.com](mailto:editor@elektor.com).



## Listing 1: Python program to speak out the number of buttons with espeak tool.

```
#!/usr/bin/env python
import device_patches    # Device Specific patches - taken care by the software
import cv2 #import Computer Vision
import os
import sys, getopt
import signal
import time
from edge_impulse_linux.image import ImageImpulseRunner
import subprocess    #this one have been added by Bera
...
    elif "bounding_boxes" in res["result"].keys():
        print('Found %d bounding boxes (%d ms.)' % (len(res["result"]["bounding_boxes"]),
            res['timing']['dsp'] + res['timing']['classification']))
        if (len(res["result"]["bounding_boxes"])>0):
            exitCode = subprocess.call(["espeak", "-ven+f3", "-a200", " Found %d Buttons" %
                len(res["result"]["bounding_boxes"]) ])    #This one have been added by Bera
...
```



### About the Author

Somnath Bera, a mechanical engineer from Jalpaiguri Govt. Engg. College, India, works as a General Manager at NTPC, the largest power producer in the country. He has a profound passion for electronics, evidenced by his 60+ innovative projects on Elektor Labs, over 10 of which have been featured in Elektor Mag. His projects are often focused on problem-solving in areas like waste and natural resource management. Somnath likes to use innovative approaches and platforms like Arduino, Raspberry Pi, and ESP32 coupled with various kinds of sensors and wireless systems to create efficient and cost-effective solutions.



### Related Products

- > **Raspberry Pi 4 B (1 GB RAM)**  
[www.elektor.com/18966](http://www.elektor.com/18966)
- > **G. Spanner, *Machine Learning with Python for PC, Raspberry Pi, and Maixduino* (E-book, Elektor)**  
[www.elektor.com/20150](http://www.elektor.com/20150)
- > **D. Situnayake, Jenny Plunkett, *AI at the Edge* (O'Reilly)**  
[www.elektor.com/20465](http://www.elektor.com/20465)

### WEB LINKS

- [1] Edge Impulse: <https://edgeimpulse.com/>
- [2] Installing Edge Impulse on Raspberry Pi 4: <http://tinyurl.com/ysc6mtuz>
- [3] Linux Python SDK: <http://tinyurl.com/2bat4w6z>
- [4] Project Page on Elektor Labs: <https://www.elektormagazine.fr/labs/count-speak-number-of-buttons>

# JOIN OUR COMMUNITY



Subscribe today at  
[elektormagazine.com/ezone-24](http://elektormagazine.com/ezone-24)

GET FREE  
DOWNLOAD



**elektor**  
design > share > learn



# Resolve Your Trickiest Embedded Development Challenges

By Stuart Cording,  
for Mouser Electronics

Embedded system development offers engineers many opportunities and at least twice as many challenges! And the challenges are as diverse and varied as it gets. Thanks to the unique combination of electronics, software, and systems, you can find yourself delving into the analog domain, resolving a digital timing issue, examining complex control loop issues, or assessing the quality of sensor readings. Test equipment manufacturers have long since recognized this, responding with clever features, new functionality, connectivity, and smartphone apps to ease the developer's life.

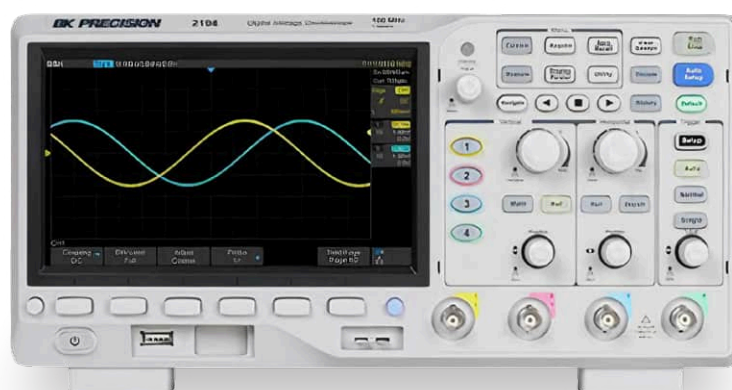


Figure 1: The B&K Precision Model 2194 offers an impressive 14 Mpts memory depth. A search function makes it easy to find signal anomalies.

One example: Oscilloscopes no longer simply display voltage against a time axis. They can perform complex mathematics on incoming signals to support analysis, decode serial data communications, and trigger on complex signals. Then there are “soft” oscilloscopes, based upon FPGA technology, that are more like a miniature lab. Out of the box, they offer the standard range of measurement capability, but they can be tuned to undertake a lot more thanks to their programmability. If you’re happy with using your laptop as its screen, and you’re short of real estate at your lab bench, they’re a great alternative.

## Searching For Runt Bits with Oscilloscopes

The advent of the digitizing oscilloscope made searching for the causes of issues much more straightforward than the old method, namely taking a photo of the CRT screen. Today’s equipment is easy to connect to networks and computers,

enabling captured signals to be reviewed and shared, simplifying configuration and automated testing. In these respects, the B&K Precision Model 2194 [1] seems much like many other four-channel oscilloscopes on the market (**Figure 1**). However, like with all significant investments, it’s worthwhile reviewing the user manual to discover its full capabilities.

With a respectable 100 MHz input bandwidth, 1 GSa/s maximum sampling rate, and a 7-inch, 800 × 480 pixel TFT display, the 2194 is as well-equipped as similar devices for more than twice the price. And, at 2.6 kg (5.7 lbs) and around 31 × 15 × 13 cm, it is easy to move and takes up little of your valuable bench space. BNC connectors and USB for storage media are easily accessible at the front. At the rear, there are only USB for control, Ethernet, and the BNC trigger output (doubling as a pass/fail output for test automation). It also includes an attachment point for the cable



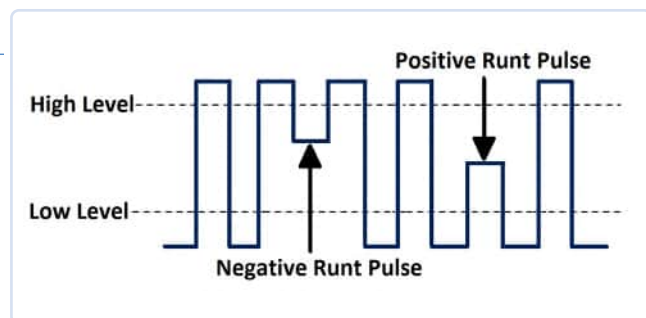


Figure 2: Runt pulses, caused by issues with drive circuitry, can be used as a trigger or can be found in captured signals using the 2194's search capability.



Figure 3: The Moku:Go is more like a miniature laboratory, offering the capability of eight different pieces of test equipment in a unit slightly larger than your hand.

locks used with laptops. For signal analysis, the unit features 38 types of measurement parameters and statistics, including count, standard deviation, rise and fall times, and peak-to-peak. It also decodes I<sup>2</sup>C, SPI, UART, CAN, and LIN.

A 100,000 wfms/s update rate helps with finding glitches and other infrequent events. Also impressive is the memory depth. With 14 Mpts available, long periods of signals can be captured and, using the zoom function, analyzed to find anomalies. However, searching for anomalies in such massive datasets can be a challenge. Thanks to the 2194's search function, a range of events can be found, including edges, slopes, pulses, and intervals.

Another developer challenge is finding digital signals that aren't being driven correctly. Known as runt pulses (**Figure 2**), they occur when the driver has insufficient slew rate to reach the desired logic level in the time required. Runt pulses can be found on the 2194 by searching, but they can also be used as a trigger. This is achieved by defining both an upper and lower trigger level. Should a signal cross one threshold but not the other, the trigger captures the waveform for review, simplifying those hard-to-find intermittent failures.

### Liquid Instruments Moku:Go

For those continuously on the move or working in the field, a traditional oscilloscope can be a heavy and bulky tool. However, there are alternatives if you are willing to use your laptop as the visual interface. One such tool is the Liquid Instruments Moku:Go [2], a compact 24 × 3.8 × 13 cm unit weighing just 750 g (**Figure 3**). Thanks to its FPGA-based design, the Moku:Go is actually eight instruments in one, operating as a 30-MHz

oscilloscope and real-time spectrum analyzer, 20 MHz waveform generator, PID controller, and an arbitrary waveform generator, are but a few capabilities.

The Moku:Go Mo unit offers two analog inputs, two analog outputs, and 16 digital I/Os. It also connects easily to both Windows and Mac machines via USB-C. The M1 adds a two-channel programmable power supply (PSU) for -5 V to +5 V and 0 V to 16 V at 150 mA along with the required cables. The M2 adds two further programmable PSUs of 0.6 V to 5 V at up to 1 A per channel, along with Ethernet connectivity.

While programmable PSUs may not seem like a killer feature, they make more sense once you realize that the current drawn can be used as a parameter in test setups. This is demonstrated in the application note "Converter Evaluation Using TI TPS63802EVM" [3] where users are shown how to measure a buck-boost converter's efficiency. The PSUs coupled with the two analog channels are more than enough to measure efficiency, and the Math channel, when provided with the load resistance, conveniently provides the power in Watts. Measurement automation is also available

using Python, MATLAB, and LabView programming APIs.

It is difficult to provide enough background on the Moku:Go in the available space. However, besides the impressive application notes [4], praise must also be given for the tool's app [5]. Firstly, the GUI is clean and tidy and, thanks to the demo mode, you can try all the different features. This lets you get a feel for how the X and Y axis zooms in and out, and how channels and math features are configured. If you're in any doubt about the competency of such a tool, give the app a try to help you decide.

### Testing and Binning Components

Power supply design requires an excellent understanding of the inductors and capacitors used. In some cases, accuracy demands require that every component is tested and binned. Regardless of whether your challenge is in the design lab or on the factory floor, the Teledyne LeCroy T3LCR Series [6] of LCR meters can help (**Figure 4**). Featuring a large 3.5" TFT display, this unit is easily configurable for four-wire measurements using the front panel or via its USB or RS-232C ports. The three T3LCR



Figure 4: In the design lab, the Teledyne LeCroy T3LCR can be used to characterize components while, in a production environment, it can be used for component binning.



Figure 5: A range of environmental measurements are possible with the Extech 250W range of meters, including sound level, RPM, luminous flux, relative humidity, and airspeed.

models support test frequencies from 10 Hz to 2 kHz, 100 kHz, or 300 kHz, and each offers a basic accuracy of 0.05%.

Features such as automatic level control (ALC) suit the constant test voltage needs of MLCC devices, while the adjustable test current suits inductor measurements. An adjustable  $\pm 2.5$  V internal DC bias can be used to simulate AC and DC simultaneously for assessing capacitance variation. In List Measurement mode, up to 10 automated measurement parameters can be gathered, allowing the characterization of components along with any variation trends. At its rear, the T3LCR provides a DB-25 connector for interfacing with a handler for component binning purposes, supporting up to ten bins. Both series and parallel equivalent model measurements are available for resistance, inductance, and capacitance. A further dozen or so parameters can also be acquired, including dissipation factor (D), quality factor (Q), phase angle, and direct current resistance.

## Keeping Track of Your Environment

When developing embedded systems, it is often necessary to compare sensor data collected by the microcontroller with professional test equipment measurements.

Other times, long-term measurements are needed when making environmental tests. The 250W series of Bluetooth Connected Environmental Meters from Extech [7] are ideal in such situations (Figure 5), offering everything from RPM measurements, light intensity, and relative humidity to sound level and air velocity. At  $54 \times 28 \times 120$  – 176 mm, the units fit easily in the hand and are easy to read with their large and brightly lit LCDs.

The RPM250W is a laser tachometer, offering RPM measurements between 10 and 99,999 RPM to an accuracy of 0.04%, while the LT250W measures light intensity up to 100,000 Lux. The compact airflow meter AN250W delivers its results in ft/min, m/s, and knots along with ambient air temperature. For sound pressure level, the SL250W offers “A” weighted “human hearing” frequency measurements with recording of the max/min values. Finally, relative humidity and temperature can be measured and logged with the RH250W.

Each device also offers Bluetooth connectivity, allowing the units to be paired with an iOS or Android device and used with the Extech ExView app. The app can simultaneously collect data from up to eight 250W Series meters and can be configured to

generate audible alarms on high/low levels. Data is logged locally and can be shared in .csv format. Furthermore, photos of test locations may be embedded into PDF reports that include measurement data.

## A Measurement Tool for Every Embedded Developer

While the development of embedded systems is diverse and fun, some of the challenges faced demand clever test equipment to find the cause of sporadic failures, determine the actual efficiency of power converters, or compare environmental measurements with embedded sensor results. Other times it is critical to understand the precise value of the components being used, or binning may be necessary during production. Hopefully, whatever your embedded development challenge, one of the test solutions covered here will help you on your journey. ◀

230750-01



### About the Author

Stuart Cording is a freelance journalist who writes for, amongst others, Mouser Electronics. He specializes in video content and is focused on technical deep-dives and insight. This makes him particularly interested in the technology itself, how it fits into end applications, and predictions on future advancements.

Mouser Electronics is an authorized semiconductor and electronic component distributor focused on new product introductions from its leading manufacturer partners.

## WEB LINKS

- [1] B&K Precision Model 2194: <https://eu.mouser.com/new/bk-precision/bk-2194-oscilloscope/>
- [2] Liquid Instruments Moku:Go: <http://tinyurl.com/Moku-Go>
- [3] Converter Evaluation Using TI TPS63802EVM: <http://tinyurl.com/AppNoteConverter>
- [4] Application Notes by Liquid Instruments: <https://www.liquidinstruments.com/blog/category/application-notes/>
- [5] Windows & macOS apps by Liquid Instruments: <https://www.liquidinstruments.com/products/desktop-apps/>
- [6] Teledyne LeCroy T3LCR: <http://tinyurl.com/TeledyneT3LCR>
- [7] Extech 250W: <https://eu.mouser.com/new/extech/extech-250w-meters/>

# ESP32 Terminal

## A Handheld Device with a Touch-Capable Display

By Johan van den Brande (Belgium)

The Elecrow ESP32 Terminal is an ESP32-S3-driven handheld device with a 3.5" 480 × 320 TFT capacitive touch display and a multitude of possibilities. The device can be a nice addition to your projects if you need a touch-capable display with interfacing capabilities.

The display based on an ILI9488 display driver has a 16-bit color depth; its touch capabilities are handled by an FT6236. Both chips are well-supported by the Arduino community. The ESP32 Terminal is housed in a black acrylic shell, which makes it feel robust; it certainly is not fiddly plastic housing. A battery connector is present, with LiPo charging circuitry, but it would be more portable if there was a battery present in the device. If you want to make your project portable, you'll need to add a battery pack to the back. There are two M3 mounting holes at the back that you can use to attach a backpack or mount it to a wall.

### ESP32-S3 MCU

Looking at what powers the device, I'm always amazed by the incredible amount of computing power packed in modern MCUs. The ESP Terminal is powered by an Espressif ESP32-S3 [2], which houses a 32-bit dual-core XTensa LX7 MCU running at 240 MHz. Compare that to the original Arduino UNO with its 8-bit Microchip ATmega328 running at 16 MHz!

The MCU has 512 KB of SRAM on board, together with 8 MB PSRAM. PSRAM stands for pseudo-static RAM, a kind of external static RAM, which in the case of the ESP32-S3 is connected via an SPI interface to the MCU.

As wireless connectivity goes, we have 2.4 GHz Wi-Fi (2.4 GHz 802.11 b/g/n) and Bluetooth 5 LE. I wasn't aware of this, but Bluetooth 5 has long-range capabilities, and this module claims to support it. Long-range mode extends the range from 10 to 30 m to more than 1 km.

### Connecting to the Physical World

The ESP32 Terminal has a battery connector and an on-board LiPo charger. You can use the USB-C connector to power the device, but also to charge the LiPo battery. There's also a micro-SD card slot, which is handy if you want to store a few images or other (graphical) assets like webpages for your application.

The module's sides have a total of four "Crowtail" ports [3]. These are all 4-wire interfaces compatible with Grove from Seeed Studio. There's one digital, one analog, one serial (UART) and one I<sup>2</sup>C connector, enough for simple projects. If you need more, you can use the I<sup>2</sup>C port as a breakout.



Figure 1: The ESP32 Terminal in its black enclosure showing weather data.



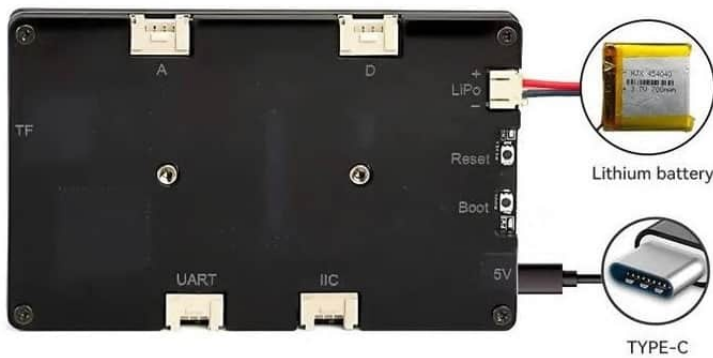


Figure 2: The four "Crowtail" ports are visible on the rear side.

## Programming the ESP32 Terminal

Uploading firmware happens through the USB-C cable, over a serial connection exposed by a CH340 USB-to-serial converter [4]. I had no luck trying to get this to work with my MacBook, for which I tried to install the official driver. Luckily, I still have my old Linux laptop, which runs Ubuntu and that works perfectly!

The first thing I wanted to try out was running some flavor of Python on the device. That turned out to be difficult. Although the documentation says it can be programmed with Python and MicroPython, I could not find a ready-made downloadable firmware.

Figure 3: Add a DHT-11 sensor and some code, and you have a simple weather station.

Reading through their website, there are numerous examples and a tutorial for Arduino [5], so that's what I started to use for my demo applications.



## UI Design

According to the ESP32 Terminal's webpage, the device is LVGL certified. LVGL [6][7] stands for Light and Versatile Graphics Library, which is an open-source embedded graphics library for creating UIs for various MCUs and display types. Although the library itself is open source, a closed source drag-and-drop layout editor, Squareline Studio [8], is available as well.

I installed the trial version, and although it is a bit overwhelming at first sight, I quickly managed to draw up a simple UI for the servo project below. If creating UIs for embedded devices is your goal, then it is worthwhile to invest some time to learn this product.

## Two Small Projects

I decided to create two example projects. The first project is a small weather station, where we draw our own user interface using line and circle primitives. The second project is an example where we control a servo from a user interface designed with Squareline Studio.

## Weather Station

For the weather station, we use a DHT-11 sensor, which can sense temperature and humidity. It has a digital one-wire interface and is well-supported by Arduino. For this project, I chose the *Adafruit DHT* library [9]. It is part of a set of sensor libraries sharing common code, and therefore we also need to install the *Adafruit Unified Sensor Driver* library [10].

In this project, I decided to create my own UI from scratch, using primitives to print text and create some line graphics. For this, I used the *LovyanGFX LCD* and *e-Ink graphics* library [11][12], another open-source graphics library.

After initializing the screen, for which I just copied one of the examples from the ESP32 Terminal website, we read the DHT-11 sensor. This gives us three values: temperature, humidity, and the heat index. The heat index is a temperature adjusted by the actual humidity. These three values are displayed in a simple dial, drawn using three circles and a line as the pointer. We update the screen every 2 s with a new reading.

## Servo Controller

For the servo controller, I wanted to try the Squareline Studio tool to create the UI. A slider in the form of an arc is the only widget used. The range of the slider is set from 0 to 180, corresponding to the angle of the servo.

When you design a UI for the ESP32 using this tool, you'll need to start from the *Arduino with TFT\_eSPI* template and set the color depth to 16 bits and the



resolution to  $480 \times 320$ . After exporting the generated code, you'll need to unzip it into a directory named *ui* in your Arduino *libraries* folder. Normally, you find this *libraries* directory in the same directory as where your Arduino sketches live. It took me some time to figure this out.

The servo is driven by the *ESP32Servo* library [13]. The standard Arduino servo library does not work with the ESP32.

The code is elementary as in the `loop` function we read the angle of the arc slider, which returns a number between 0 and 180, and then send that value to the `servo.write` function.

You can find the source code of both projects on the Elektor website [14]. I did not include the *libraries* folder in the download, as it is (far) too big. Installing the dependencies yourself is not complicated.

### A Lot of Potential

The ESP32 Terminal from Elecrow is not that easy to tame as I had hoped, but it has a lot of potential. If you want to invest the time to learn the LVGL library or another UI library, and if you are comfortable with Arduino, then with the ESP32 Terminal you can add a user interface with touch control to your next project.

It would have been easier if MicroPython [15] firmware had been available for download on the product site.

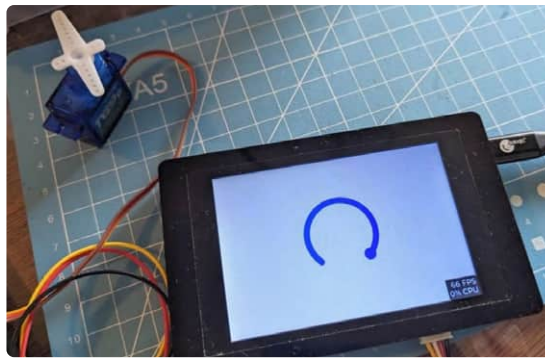


Figure 4: The circular slider controls the position of the servo.

I searched around a bit on the Internet, but could not find anything. It is certainly doable to compile your own, and maybe that's a nice idea for your next project? ◀

230755-01

### Questions or Comments?

If you have any technical questions, you can contact the Elektor editorial team at [editor@elektor.com](mailto:editor@elektor.com).



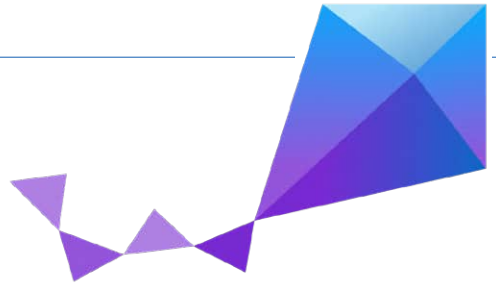
### Related Products

- **ESP Terminal**  
(ESP32-S3 based Development Board with 3.5" Capacitive TFT Touch Display)  
[www.elektor.com/20526](http://www.elektor.com/20526)



### WEB LINKS

- [1] Elecrow ESP32 Terminal: <https://www.elektor.com/esp-terminal-esp32-s3-based-development-board-with-3-5-capacitive-tft-touch-display>
- [2] ESP32-S3 datasheet: [https://www.espressif.com/sites/default/files/documentation/esp32-s3\\_datasheet\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/esp32-s3_datasheet_en.pdf)
- [3] "Crowtail" ports: [https://www.elecrow.com/wiki/index.php?title=Main\\_Page#Crowtail](https://www.elecrow.com/wiki/index.php?title=Main_Page#Crowtail)
- [4] CH340 USB-to-serial converter: <https://cdn.sparkfun.com/datasheets/Dev/Arduino/Other/CH340DS1.PDF>
- [5] Tutorial for Arduino: [https://www.elecrow.com/wiki/index.php?title=Lesson01\\_Introducing\\_the\\_ESP32\\_Display\\_series\\_and\\_environment\\_configuration](https://www.elecrow.com/wiki/index.php?title=Lesson01_Introducing_the_ESP32_Display_series_and_environment_configuration)
- [6] LVGL: <https://lvgl.io/>
- [7] LVGL Documentation: <https://docs.lvgl.io/latest/en/html/index.html>
- [8] Squareline Studio: <https://squareline.io/>
- [9] DHT-11 library: <https://github.com/adafruit/DHT-sensor-library>
- [10] Unified Sensor Library: [https://github.com/adafruit/Adafruit\\_Sensor](https://github.com/adafruit/Adafruit_Sensor)
- [11] LovyanGFX LCD and e-Ink graphics library: <https://github.com/ropg/LovyanGFX>
- [12] LovyanGFX Documentation: <https://lovyangfx.readthedocs.io/en/latest/index.html>
- [13] ESP32 servo library: <https://www.arduino.cc/reference/en/libraries/esp32servo/>
- [14] Software Download: <https://www.elektormagazine.com/news/elecrow-esp32-terminal-review>
- [15] Günter Spanner, "MicroPython for the ESP32 and Friends (Part 1): Our First Programs," 2021: <https://www.elektormagazine.com/articles/micropython-esp32-microcontroller>



# Getting Started

# With the Zephyr RTOS

As Powerful as It Is Hard to Master

By Clemens Valens (Elektor)

Zephyr is a small, scalable, real-time operating system (RTOS) optimized for resource-constrained devices, across multiple architectures. Hosted by the Linux Foundation (see [1]), the project is an open-source collaborative effort with the goal of building a best-in-class RTOS. The Zephyr OS has been gaining traction over the past years in embedded computing and today, new microcontrollers and development boards proudly sport Zephyr support. Time to take a closer look.

Zephyr is scalable, making it suitable for a wide range of devices with different resource constraints. Scalability is achieved through a modular architecture that allows developers to include only the components they need, minimizing the system footprint. The Zephyr website claims that it runs on systems from as small as 8 KB of memory up to systems with gigabytes of memory.

## Wide Hardware Support

Zephyr supports a broad range of architectures, including ARM, x86, RISC-V, Xtensa, MIPS and more. FPGAs too are supported with Nios2 and MicroBlaze soft cores. At the time of writing, Zephyr lists over 600 usable boards, including the Arduino UNO R4 Minima, GIGA R1 WIFI and Portenta H7, multiple flavors of the ESP32, both versions of the BBC micro:bit, the Raspberry Pi Pico (and even the Raspberry Pi 4B+), nRF51 and nRF52 boards, the NXP MIMXRT1010-EVK and family, and the STM32 Nucleo and Discovery families. While I only listed boards commonly encountered in Elektor, there are many more.

Besides processor boards, Zephyr also has support for many add-on boards (known as “shields”) and it comes with drivers for all sorts of interfaces and more than 150 sensors.

## Multitasking, Networking, and Power Management

Being a real-time operating system (RTOS), Zephyr provides features such as preemptive multitasking, inter-thread communication, and real-time clock support. The OS also comes with networking technologies and protocols like TCP/IP, Bluetooth, and IEEE 802.15.4 (used in e.g., Zigbee), MQTT, NFS, and LoRaWAN. Together with its networking capabilities, the power management features built into Zephyr make it suited for energy-efficient IoT applications and battery-operated devices.

A set of libraries and middleware simplify common tasks, such as communication protocols, file systems, and device drivers.

Know that Zephyr is also designed to be compatible with safety certifications such as ISO 26262, making it suitable for safety-critical applications.

## Inspired by Linux

Zephyr is not Linux, but it makes use of concepts, techniques and tools used by Linux. As an example, Kconfig is used for configuring the OS, and hardware properties and configurations are described using the Device Tree Specification (DTS) [2]. Therefore, Linux developers will feel quickly at home when coding for Zephyr.

## Open Source

Last, but not least, Zephyr is released under the permissive Apache 2.0 license, which allows for both commercial and non-commercial use. Its user community provides support and documentation. You can join too.

## Trying Out Zephyr

Trying out Zephyr has been on my to-do list for several years, but my first experiences with it were not very encouraging, so I never dug much deeper. At the time, one of its main issues was (besides getting it to compile without errors) that it required a programming



Figure 1: The tiny BBC micro:bit board is an excellent target for trying out the Zephyr RTOS. Who would have thought that this little board intended for teaching programming to 10-year-olds using MakeCode, a Scratch-like graphical language, would also be a super tool for seasoned embedded software developers interested in learning an industrial-grade, real-time operating system?

pod to program the target controller, making it less suitable for makers. Thanks to Arduino and its bootloader, we have become used to not needing special programming tools, and so requiring one felt like a step back.

## Choosing a Board

Things have evolved since. As mentioned above, today, Zephyr supports over 600 microcontroller boards. Chances are that you already own one or more compatible boards. Looking through the list, I discovered that I have more than a dozen different supported boards at my disposal.

## Long Live the BBC micro:bit!

I tried most of them to finally settle on the BBC micro:bit for my experiments (**Figure 1**, known by Zephyr as `bbc_microbit` or `bbc_microbit_v2`, depending on the board's version). Compared to my other options, besides being readily available, the BBC micro:bit has probably the best Zephyr support, meaning that all of its peripherals are accessible and supported by a few examples. Best of all, it can be programmed *and* debugged without needing extra tools.

The popular ESP-WROOM-32 (known by Zephyr as `esp32_devkitc_wroom`) is also a suitable candidate, but debugging requires an external tool.

The Arduino GIGA R1 WIFI is another good option, but its bootloader is destroyed when using Zephyr. You can restore it, of course, but it is an unwanted collateral.

Officially, the Arduino UNO R4 Minima requires an SWD-capable programming pod (as do many other boards, including the Raspberry Pi Pico), but I did find a way around this using `dfu-util` (see below). Like the GIGA R1, its Arduino bootloader gets trampled by Zephyr, though.

## Use an Emulator Instead

In case you don't have a suitable board, but you really want to try Zephyr, know that it has built-in emulator support for QEMU (on Linux/macOS only). This lets you run and test applications virtually. Renode from Antmicro is capable of similar feats, although I didn't try it.

## Installing Zephyr

I installed Zephyr on a computer running Windows 11 — I did not try Linux or macOS. Detailed and working installation instructions are available online ([3]). The steps to take are clearly pointed out and don't need further explanation. I used a virtual Python environment, as suggested. This does mean that you must write down the command to activate the virtual environment somewhere, as you will need it every time you start working. If you use Windows PowerShell, you should launch the script `activate.ps1`; in Command Prompt it is the `activate.bat` batch file. Windows PowerShell is better at processing compiler and linker output (**Figure 2**).

Zephyr consists of two parts — the OS itself and an SDK containing a collection of MCU tool chains (21 at the time of writing...).

```
Administrator: Windows PowerShell
In file included from D:/dev/zephyr/zephyrproject/zephyr/include/zephyr/arch/arm/arch.h:20,
from D:/dev/zephyr/zephyrproject/zephyr/include/zephyr/arch/cpu.h:19,
from D:/dev/zephyr/zephyrproject/zephyr/include/zephyr/kernel/includes.h:37:
D:/dev/zephyr/zephyrproject/zephyr/include/zephyr/devicetree.h:238:32: error: 'DT_N_ALIAS_led0_P_gpios_IDX_0_VAL_pin' undeclared here (not in
a function); did you mean 'DT_N_S_leds_S_led_0_P_gpios_IDX_0_VAL_pin'?
238 | #define DT_ALIAS(alias) DT_CAT(DT_N_ALIAS_, alias)
|
D:/dev/zephyr/zephyrproject/zephyr/include/zephyr/devicetree.h:4352:9: note: in definition of macro 'DT_CAT7'
4352 | a1 ## a2 ## a3 ## a4 ## a5 ## a6 ## a7
|
D:/dev/zephyr/zephyrproject/zephyr/include/zephyr/devicetree/gpio.h:164:9: note: in expansion of macro 'DT_PHA_BY_IDX'
164 | DT_PHA_BY_IDX(node_id, gpio_pha, idx, pin)
|
D:/dev/zephyr/zephyrproject/zephyr/include/zephyr/drivers/gpio.h:332:24: note: in expansion of macro 'DT_GPIO_PIN_BY_IDX' 332 |
.pin = DT_GPIO_PIN_BY_IDX(node_id, prop, idx),
|
D:/dev/zephyr/zephyrproject/zephyr/include/zephyr/drivers/gpio.h:367:9: note: in expansion of macro 'GPIO_DT_SPEC_GET_BY_IDX'
367 | GPIO_DT_SPEC_GET_BY_IDX(node_id, prop, 0)
|
D:/dev/zephyr/zephyrproject/zephyr/samples/basic/blinky/src/main.c:20:40: note: in expansion of macro 'GPIO_DT_SPEC_GET' 20 | static const
struct gpio_dt_spec led = GPIO_DT_SPEC_GET(LED0_NODE, gpio);
|
D:/dev/zephyr/zephyrproject/zephyr/include/zephyr/devicetree.h:238:25: note: in expansion of macro 'DT_CAT'
238 | #define DT_ALIAS(alias) DT_CAT(DT_N_ALIAS_, alias)
|
D:/dev/zephyr/zephyrproject/zephyr/samples/basic/blinky/src/main.c:14:19: note: in expansion of macro 'DT_ALIAS'
14 | #define LED0_NODE DT_ALIAS(led0)
|
D:/dev/zephyr/zephyrproject/zephyr/samples/basic/blinky/src/main.c:20:57: note: in expansion of macro 'LED0_NODE'
20 | static const struct gpio_dt_spec led = GPIO_DT_SPEC_GET(LED0_NODE, gpio);
|
[21/132] Building C object zephyr/CMakeFiles/zephyr.dir/lib/os/heap-validate.c.obj
ninja: build stopped: subcommand failed.
FATAL ERROR: command exited with status 1: 'C:\Program Files\CMake\bin\cmake.EXE' --build 'D:/dev/zephyr/zephyrproject/zephyr/build'
(.venv) PS D:/dev/zephyr/zephyrproject/zephyr>
```

Figure 2: Zephyr's west build tool is intended to run inside a terminal. Windows' `cmd.exe` can be used, but it is not a terminal. Windows PowerShell, a.k.a. Terminal, is thus more suitable.





Figure 3: In many (but not all) situations, a JTAG or SWD programmer/debug pod is required for your Zephyr experiments.

```

Administrator: Windows PowerShell
Merged configuration 'D:\dev\zephyr\zephyrproject\zephyr\samples\hello_world\prj.conf'
Configuration saved to 'D:\dev\zephyr\zephyrproject\zephyr\samples\hello_world\prj.conf'
Mconfig header saved to 'D:\dev\zephyr\zephyrproject\zephyr\samples\hello_world\prj.conf'
-- Found GnuId: c:/users/ener/zephyr-g
zephyr-eabi/bin/ld.bfd.exe (found version 2.38)
-- The C compiler identification is GNU
-- The CXX compiler identification is GNU
-- The ASM compiler identification is GNU
-- Found assembler: C:/Users/ener/zephyr-g
-- Configuring done (29.5s)
-- Generating done (0.2s)
-- Build files have been written to: D:\dev\zephyr\zephyrproject\zephyr\build
[92m-- west build: building application
[132] Generating include/generated/ve
-- Zephyr version: 3.5.99 (D:\dev\zephyr\zephyrproject\zephyr)
[132/132] Linking C executable zephyr\zephyr.elf
Memory region      Used Size  Region Size  %age Used
FLASH:             23260 B    256 KB      8.87%
RAM:               5448 B     16 KB      33.25%
IDT_LIST:          0 GB      2 KB      0.00%
(.venv) PS D:\dev\zephyr\zephyrproject\zephyr> west flash
-- west flash: rebuilding
ninja: no work to do.
-- west flash: using runner pyocd
-- runners.pyocd: Flashing file: D:\dev\zephyr\zephyrproject\zephyr\build\zephyr\zephyr.hex
0001522 I Loading D:\dev\zephyr\zephyrproject\zephyr\build\zephyr\zephyr.hex [load_cmd]
[=====] 100%
0002020 I Erased 23552 bytes (23 sectors), programmed 23552 bytes (23 pages), skipped 0 bytes (0 pages) at 10.04 kB/s [loader]
(.venv) PS D:\dev\zephyr\zephyrproject\zephyr>

```

Figure 4: The Hello World sample is not very expensive. If you are too slow opening a serial terminal, you will not see the welcome message.

The OS and the SDK don't have to be installed in the same place. In total, for me, it consumed about 12 GB of precious hard disk space. To reclaim some space, you can delete tool chains that you don't need.

After installation, see if it is working by building a sample and uploading it to the board with the command below. Replace `<my_board>` by the name of your board, e.g., `arduino_uno_r4_minima`:

```
west build -p always -b <my_board> samples/basic/blinky
```

If you don't want to change the path to the example, you must run this command inside the folder (where `.venv` indicates that you are in a virtual environment):

```
(.venv) <my_path_to_zephyr>\zephyrproject\zephyr
```

If the example builds without errors, you can upload it to the board using

```
west flash
```

and the board's "default" LED starts flashing at a rate of 0.5 Hz.

As mentioned before, flashing may require an external programming tool, such as a J-Link adapter or another (JTAG or SWD-capable) programmer (**Figure 3**) and the driver software for it must be accessible (i.e. in the search path, `%PATH%` on Windows). If it isn't, you will be informed about it by a message (often long and cryptic).

On the BBC micro:bit V2, the first time I had to copy the HEX file manually to the board using the standard micro:bit programming procedure. After that, the flash command worked fine. The executable `zephyr.hex` is located in `zephyrproject\zephyr\build\zephyr\`.

The default flash command for the Arduino boards, UNO R4 Minima and GIGA R1 WIFI, requires that the `dfu-util` programming utility is in the OS's search path (before you activate the virtual environment, if you use one). This utility is included in the Arduino IDE. However,

where exactly it lives on your computer is up to you to find out (by default in `%HOMEPATH%\AppData\Local\Arduino15\packages\arduino\tools\dfu-util\<your most recent Arduino version>`). The board must also be put in DFU mode. Do that by pressing the reset button twice. When the LED starts "pulsing" or "breathing," you can flash the program.

## Blinky Compatibility

You may have noticed that I suggested the Arduino UNO R4 Minima as board to use for the Blinky example and not the BBC micro:bit that I was so enthusiastic about earlier. The reason is that despite it having 25 LEDs (not counting the power-on indicator), it does not have an LED that is compatible with the Blinky example. The ESP Wroom 32 doesn't have one either, but the R4 Minima does.

The GIGA R1 is also Blinky-compatible. The MCU on this board has two cores (Cortex-M7 and -M4) and Zephyr lets you choose which one to use by selecting either `arduino_giga_r1_m4` or `arduino_giga_r1_m7` for the build command. You can show that the cores are indeed independent by flashing the Blinky example twice, once for the -M4 and once for the -M7. The GIGA has an RGB LED and Blinky will use different colors for each core: blue for the M4 and red for the M7. To make the two Blinkies more distinct, you can change the blink rate for one of them (in `samples\basic\blinky\src\main.c`, change the value of `SLEEP_TIME_MS`).

## Hello World

For boards without a Blinky LED, there is the `hello_world` example that outputs a string on the serial port.

```
west build -p always -b <my_board> samples/hello_world
west flash
```

This example works with both the BBC micro:bit and the ESP-WROOM-32 module. To see the output string, open a serial terminal program on your computer. The data rate is usually 115,200 baud (115200,n,8,1). You may have to reset the board first because the message is printed only once, and you may have missed it (**Figure 4**).



On the R4 Minima and the GIGA R1, the serial output is on pin 1 and not, as you naively may expect, on the USB-C port as it would be in the Arduino IDE. This is because the USB port is a peripheral of the MCU and not a separate chip. Since Zephyr is a modular and scalable OS, USB support — like every other module or peripheral — must be explicitly enabled for the project before it can be used. You do this in the project's configuration files. More on those later.

For boards without a built-in serial-to-USB converter, you must find the serial port (usually port 0 in case the MCU has more than one) and connect it to your computer through an external serial-to-USB converter.

## Pushing It a Bit Further

If you managed to get both the Blinky and *hello\_world* examples to work on your board, you are in a pretty good position to create a working application running on Zephyr. If only one works, and you would like the other to work too, things are a bit more complicated.

I selected the BBC micro:bit as my preferred board for Zephyr experiments, even though it isn't compatible with the Blinky example. That's not really a problem, however, as the board comes with a few examples (in the *bbc\_microbit* subfolder of the *samples\boards\* folder), one of which (*display*) is much nicer than a single-LED Blinky. There are also examples for other boards, but for very few compared to the 600+ supported boards (not even 5%). Furthermore, most of these examples concern advanced or obscure use cases.

When you try to build Blinky for the BBC micro:bit (or the ESP-WROOM-32 or any other incompatible board), you will end up with an incomprehensible error message. What it tries to tell you is that `led0` is an unknown entity. This is the default Blinky LED (a bit like `LED_BUILTIN` on Arduino). As the micro:bit has an extension port to which you can connect LEDs and stuff, let's attempt to define one of these port pins as `led0`.

Before we do so, make a back-up copy of the *samples/basic/blinky* folder or create a copy with a new name and use that instead. In what follows, *samples/basic/blinky* is used.

## The Device Tree

Defining an `led0` takes us to the device tree, already briefly mentioned. The device tree, contained in one or more text files, lists the peripherals and memory available on a board or in a controller. In Zephyr, these files have the *.dts* or *.dtsi* ("I" for "include") and files may include other files. Processor *.dtsi* files are in the *dts* folder, board *.dts* and *.dtsi* files are in the *boards* folder. Both folders are organized by processor architecture.

To view DTS(I) files in a somewhat more comfortable way, you can use the DeviceTree plugin for Visual Studio Code [4]. This plugin provides syntax highlighting and code folding, making the files easier to read (DTS files use C language-style syntax). **Figure 5** shows an extract of the *.dtsi* file for the nRF51822 that is at the heart of the BBC micro:bit V1 board. This file is included by the board's DTS file. As an example, note how the status of `uart0` is

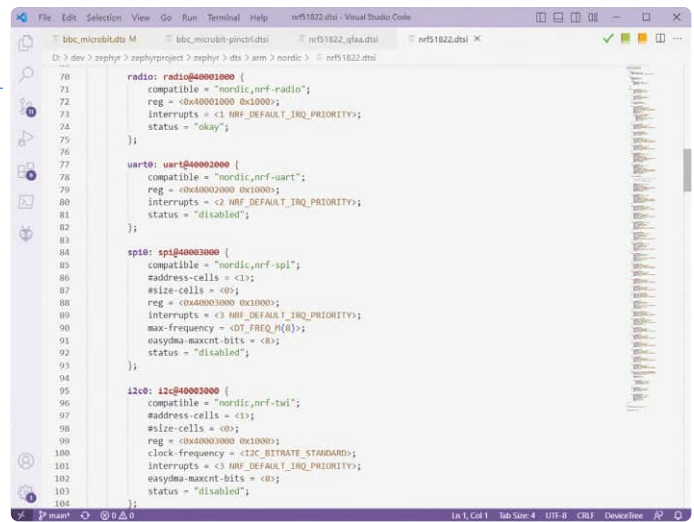


Figure 5: An extract from the *nrf51822.dtsi* file. The zoomed-out view on the right shows how long this file is.

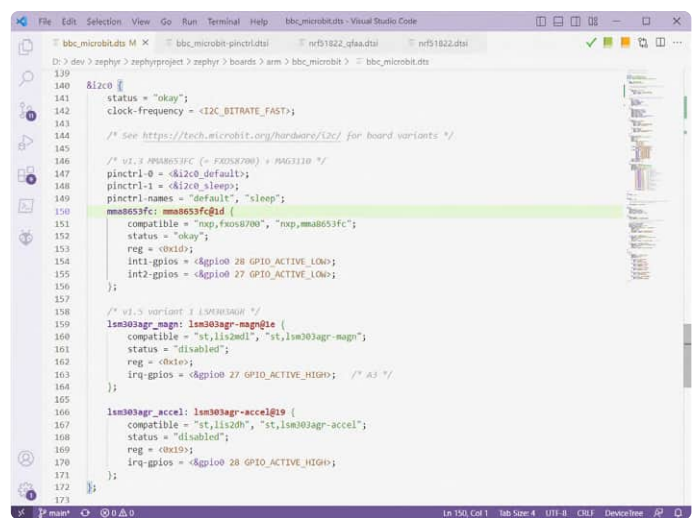


Figure 6: This section of the device tree represents the I<sup>2</sup>C bus of the BBC micro:bit board, not of its processor. Taken from the *bbc\_microbit.dts* file.

set to "disabled". This status is overridden in the board's DTS file, where it is set to "okay", meaning that it may be used. The same is true for `gpio0` and `i2c0`.

## I<sup>2</sup>C in the Device Tree

Another snippet of the *.dts* file for the BBC micro:bit is visible in **Figure 6**. It shows the device tree for the I<sup>2</sup>C bus. The micro:bit has one or two sensors attached to the bus (depending on the micro:bit V1 board variant) and they are represented in the tree by `mma8653fc` and `lsm303agr` (the latter comprises two sensors, which is why it appears twice in the tree). The first one has status "okay", while the other two are "disabled". This is correct for my board variant, which is a sample from the very first micro:bit V1 generation.

As the snippet shows, this sensor is compatible with the FXOS8700 and MMA8653FC, its address on the I<sup>2</sup>C bus is 0x1d, and two `int` (interrupt) signals are declared that are connected to two GPIO pins: 27 and 28. If you want to try it, a demo program is available:

```
west build -p always -b bbc_microbit samples/sensor/
west flash
```

```

[00:01:56.302.398] <dbg> temp_nrf5: temp_nrf5_channel_get: Temperature:21,0
[00:02:00.303.588] <dbg> temp_nrf5: temp_nrf5_sample_fetch: sample: 84
[00:02:00.303.741] <dbg> temp_nrf5: temp_nrf5_channel_get: Temperature:21,0
[00:02:04.321.350] <dbg> temp_nrf5: temp_nrf5_sample_fetch: sample: 84
[00:02:04.321.502] <dbg> temp_nrf5: temp_nrf5_channel_get: Temperature:21,0
[00:02:08.322.692] <dbg> temp_nrf5: temp_nrf5_sample_fetch: sample: 84
[00:02:08.322.845] <dbg> temp_nrf5: temp_nrf5_channel_get: Temperature:21,0
[00:02:12.340.454] <dbg> temp_nrf5: temp_nrf5_sample_fetch: sample: 84
[00:02:12.340.606] <dbg> temp_nrf5: temp_nrf5_channel_get: Temperature:21,0
[00:02:16.341.766] <dbg> temp_nrf5: temp_nrf5_sample_fetch: sample: 84
[00:02:16.341.918] <dbg> temp_nrf5: temp_nrf5_channel_get: Temperature:21,0
[00:02:20.359.527] <dbg> temp_nrf5: temp_nrf5_sample_fetch: sample: 84
[00:02:20.359.680] <dbg> temp_nrf5: temp_nrf5_channel_get: Temperature:21,0
[00:02:24.360.870] <dbg> temp_nrf5: temp_nrf5_sample_fetch: sample: 84
[00:02:24.361.022] <dbg> temp_nrf5: temp_nrf5_channel_get: Temperature:21,0
[00:02:28.378.631] <dbg> temp_nrf5: temp_nrf5_sample_fetch: sample: 84
[00:02:28.378.784] <dbg> temp_nrf5: temp_nrf5_channel_get: Temperature:21,0
[00:02:32.379.974] <dbg> temp_nrf5: temp_nrf5_sample_fetch: sample: 84
[00:02:32.380.126] <dbg> temp_nrf5: temp_nrf5_channel_get: Temperature:21,0
[00:02:36.397.735] <dbg> temp_nrf5: temp_nrf5_sample_fetch: sample: 84
[00:02:36.397.888] <dbg> temp_nrf5: temp_nrf5_channel_get: Temperature:21,0
[00:02:40.399.047] <dbg> temp_nrf5: temp_nrf5_sample_fetch: sample: 84
[00:02:40.399.200] <dbg> temp_nrf5: temp_nrf5_channel_get: Temperature:21,0

```

Figure 7: The output of the samples/sensor/foxos8700 demo running on the BBC micro:bit.

Note that this won't work for the BBC micro:bit V2, as it has a different sensor in its device tree.

The output of the demo is shown in **Figure 7**, but we are digressing.

## Overlaying the Device Tree

Back to our LED, `led0`. The board's device tree does, as expected, not mention `led0`, so we must add it. We could insert it directly into the board's device tree file, but that would be incorrect as the board does not have an `led0`. The proper way to extend a device tree is by adding an overlay for it. The contents of an overlay file are added to the device tree. Sections that exist in the tree are extended (in case of a new item) or overwritten (in case the item already exists in the tree); new sections are added.

Overlays must be placed inside the project folder, in a subfolder named `boards`. When this subfolder is present, the build process will search it for a file with the name `<my_board>.overlay`. In my case, the filename is `bbc_microbit.overlay` (`bbc_microbit_v2.overlay` for V2 users) **Figure 8** shows the contents of the file.

## Add a Blinky LED

Zephyr has a special device tree keyword for LEDs, which is `leds`, so we create a node (branch) for it (you can call it whatever you want, but stick to `leds` if it is supposed to be overlayed on an existing `leds` node). This node is to be added to the root of the device tree; therefore a forward slash '/' is placed in front of it as that means root in DT language. The next line states that this branch is compatible with the `gpio-leds` driver built into Zephyr (you can find the interface for this driver in `zephyr/include/zephyr/drivers/led.h`).

## Child Nodes

Next comes a list of LED child nodes. Since I have only one LED, there is only one child node, `led_0`, which I labeled `led0`. Labeling a node is optional, but it allows referencing the node elsewhere in the tree, which we will do a few lines further down. Also, they can be used by the application (developer) to get access to nodes and their properties.

A child node must specify the properties of the device. In the case of an LED, only the GPIO pin is a required property, but the optional property named `label` may be added. Such labels can be used for

providing documentation or human-readable information to the application. They have no other use.

As GPIO pin, I chose `1`, which corresponds to the large hole/pad 2 on the micro:bit extension connector. If you have a BBC micro:bit V2, use `4` as GPIO pin (instead of `1`).

## Create an Alias

The next step is needed because the Blinky example expects it. It consists of creating the `led0` alias for our LED. You might think that labeling the child node would have been enough, but it isn't, because Blinky uses the `DT_ALIAS` macro to get access to the LED child node. Therefore, we must provide something that this macro can digest, which happens to be an alias. It goes inside the `aliases` block. If Blinky had used the `DT_NODELABEL` macro instead, then an alias would have been superfluous, as `DT_NODELABEL` grabs the `led0` child node label directly. I know, having labels and aliases with the same name is a bit confusing, but it is required for my explanation.

```

1  SPDX-License-Identifier: Apache-2.0
2
3  * Copyright (c) 2023 Elektor
4
5  /*
6
7
8  leds {
9      compatible = "gpio-leds";
10     led0: led_0 {
11         gpios = <&gpio0 1 GPIO_ACTIVE_HIGH>;
12     };
13 };
14
15 aliases {
16     led0 = &led0;
17 };
18
19

```

Figure 8: This device tree overlay file makes the BBC micro:bit V1 compatible with the Blinky example.



## Zephyr Macros

Even though macros are frowned upon in C/C++ programming, Zephyr uses them a lot. Macros such as `DT_ALIAS` and `DT_NODELABEL` allow the application and project configuration tools to extract information from the device tree, and they are plentiful. You can find their descriptions in the Zephyr manual, in the chapter, “Devicetree API” [5].

Fun fact: Many (all?) Zephyr macros expect their arguments to be lowercase, with the characters that are not letters (‘a’ to ‘z’) or numbers (‘0’ to ‘9’) replaced by underscores (‘\_’). This is called lowercase-and-underscores compatible. As an example, imagine I had labeled the LED child node from before `LED-0` instead of `led0`. Then, the argument to use with `DT_NODELABEL` would have been `led_0`, i.e. `DT_NODELABEL(led_0)` because ‘-’ is not a letter or a number, and letters must be lowercase. In other words, to the application developer using device tree macros, the underscore character is a wildcard. Thus, `led_0` in the application can refer to either `led_0`, `led-0`, `Led_0`, `LED-0` and `ledé0` (and every other variation you can come up with) in the device tree. With this in mind, it is highly recommended to carefully read the documentation of Zephyr macros.

## Thou Shalt Not Make Mistakes

Note that making mistakes in the device tree is punished by the compiler shouting “FATAL ERROR” at you without providing any further information.

## Pristine Builds

When playing around with the device tree or your application, you will likely rebuild your project a lot. To speed things up a bit, remove the `-p always` (“p” from “pristine”) from the build command. This stops it from rebuilding everything from scratch. If, on the other hand, you are trying out many different examples one after the other, then keep it, as it will save you an annoying error about the build folder not being intended for your project.

Note that the flash command also triggers the last build command, so it is enough to just run the flash command every time you change something.

## Use a Device Driver

The Blinky example calls the `gpio_pin_toggle_dt()` to toggle the state of the LED. This is a function of the GPIO driver. Of course, this is perfectly fine, but Zephyr also includes a collection of LED drivers. Using an LED driver not only makes the program more readable, but it also makes the program more flexible and portable, as the LED driver can be replaced by another one without changing the application itself. This is where the scalability and modularity of Zephyr come in.

## Kconfig Has a GUI

Integrating an LED device driver in our program requires a few steps. First, the project must be reconfigured so that it will include the driver. The project configuration is handled by Kconfig, the kernel build-time configuration system also used by Linux.

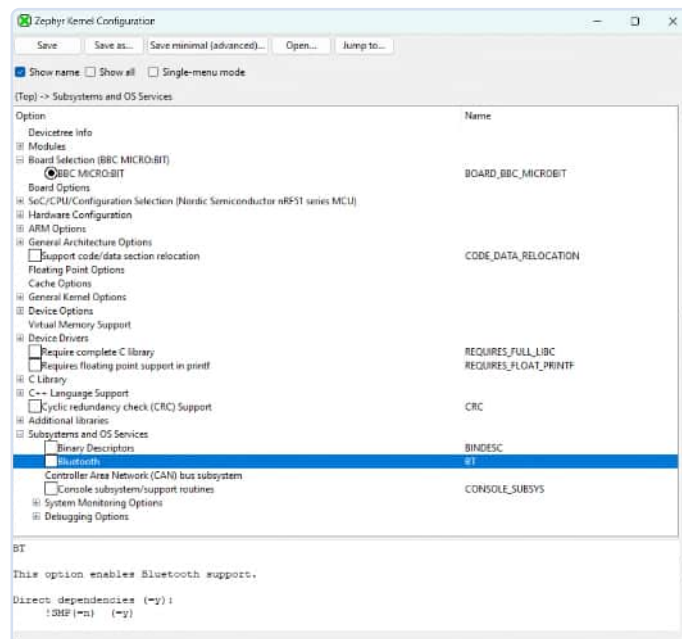


Figure 9: The GUI of the Kconfig project configuration tool. There are many options.

There are several ways to interact with it, and one of them is a graphical user interface (GUI). In Zephyr, you open it like this:

```
west build -t guiconfig
```

The GUI takes a while to open, but when it does, it looks like **Figure 9**. It shows a lot of information about the project under development. Note the project-under-development part. To ensure that Kconfig is indeed working on your project, do a pristine build (with the `-p always` flag) of your project before launching the Kconfig GUI.

## So Many Configuration Options...

Take some time to explore the configuration tree. Unfold branches by clicking on the + symbols. Highlight options by clicking on them. This will display some information in the bottom pane. Note how floating-point support for `printf()` is a configuration option, as is C++ language support. Similarly, under Build and Link Features, you can find compiler optimization options.

There are tons of configuration options. The one that is of interest to us is in the *Device Drivers* branch. Unfold it and scroll down while looking at all that is available. The LED driver is about halfway down: *Light-Emitting Diode (LED) Drivers*. Check the checkbox. Leave the options in its subbranch on their default values (**Figure 10**). Click the Save button and note the configuration file’s path printed at the bottom of the window. It is instructive to inspect this file, just to see what it contains (a lot). Close the GUI.

From this point on, do not specify the `-p always` flag anymore in build commands, as it will undo the changes you made above. I will show later how to make the configuration change permanent.



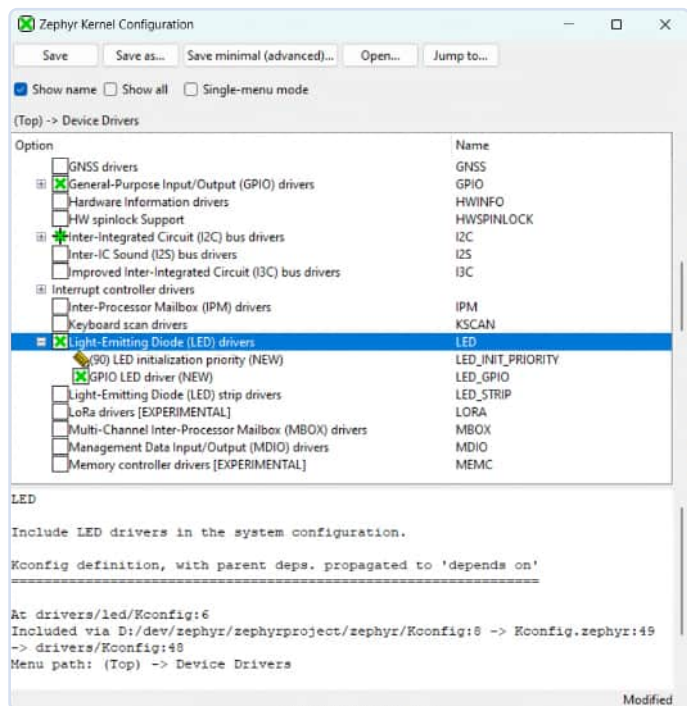


Figure 10: Select Light-Emitting Diode (LED) Drivers and leave the child values as they are.

```
#include <zephyr/kernel.h>
#include <zephyr/device.h>
#include <zephyr/drivers/led.h>

#define SLEEP_TIME_MS (1000) /* 1000 msec = 1 sec */

int main(void)
{
    printf("\nBlinky with LED device driver\n");

    const struct device *const led_device = DEVICE_DT_GET_ANY(gpio_leds);
    if (!led_device)
    {
        printf("No device with compatible 'gpio-leds' found\n");
        return 0;
    }
    else if (!device_is_ready(led_device))
    {
        printf("LED device %s not ready\n", led_device->name);
        return 0;
    }

    while (1)
    {
        int result = led_on(led_device, 0);
        if (result < 0)
        {
            printf("led_on failed\n");
            return 0;
        }
        k_msleep(SLEEP_TIME_MS);

        result = led_off(led_device, 0);
        if (result < 0)
        {
            printf("led_off failed\n");
            return 0;
        }
        k_msleep(SLEEP_TIME_MS);
    }

    return 0;
}
```

Figure 11: The Blinky program rewritten for use with an LED device driver. Note how there is no board-specific code. This program should run on any board that has a `gpio_leds` (or `gpio-leds`) driver listed in its device tree.

## Blinky With LED Device Driver

Now we can write the new Blinky program — see **Figure 11**. It starts by including the device and LED driver header files. Then, in the main, we employ the `DEVICE_DT_GET_ANY` macro to obtain a device reference for the LED from the device tree. Note how the macro's `gpio_leds` argument is lowercase- and underscore-compatible, so it will match with the `gpio-leds` value of the `compatible` property of the `leds` node in the device tree (as explained above). If it can't find one because you made a typo or something, the new Blinky will print the message, "No device with compatible gpio-leds found." This message will also be triggered when the `status` property of a device is set to `"disabled"`.

The use in Zephyr of the word "compatible" as a noun takes a bit of getting used to. Therefore, the error message above does not mean that there is no compatible device, but that there is no device having a property named `compatible` with the value `gpio-leds` (nor, for that matter, with `gpio_leds`, remember that the underscore '\_' replaces any character except for 'a' to 'z' and 'o' to '9').

A second check verifies that the device initialized properly. Assuming that it did, we continue.

In the infinite `while()` loop, the LED is toggled on and off using the `led_on` and `led_off` commands provided by the driver [6]. The argument `0` signifies the first (and only) device found by the macro `DEVICE_DT_GET_ANY`, which is `led0`.

## Check Return Values

As we are using a device driver instead of directly toggling a GPIO pin on the hardware register level, it is good practice to check the return values of all the driver function calls, as they may fail for some reason. A driver must provide certain functions and callbacks, but it can also have optional features. An LED driver, for instance, must implement `led_on` and `led_off`, but `led_blink` is optional. If you call `led_blink` in our project, nothing will happen because it is not implemented. It exists, but it is empty. The return value will show you this. In general, it is good programming practice to check the return value of every function call.

Build and upload the program like this (note the absence of the `-p always` flag).

```
west build -b bbc_microbit samples/basic/blinky
west flash
```

## Configure the Project

If the LED started blinking at 0.5 Hz, we have a working program. To keep it that way, we must make the current configuration permanent. To accomplish this, open the `prj.conf` file in the folder of our Blinky and add the line (unlike device tree files that use C-language syntax, Kconfig configuration files use Python-language syntax):

```
CONFIG_LED=y
```





```
Administrator: Windows PowerShell
0001118 I DP IDR = 0x0bb1477 (v1 MINDP rev0) [dap]
0001137 I AHB-AP#0 IDR = 0x04778021 (AHB-AP var2 rev0) [discovery]
0001142 I AHB-AP#0 Class 0x1 ROM table #0 @ 0xf0000000 (designer=244 part=001) [rom_table]
0001142 I [0]<e00ff000:ROM class=1 designer=43b:Arm part=471> [rom_table]
0001142 I AHB-AP#0 Class 0x1 ROM table #1 @ 0xe00ff000 (designer=43b:Arm part=471) [rom_table]
0001158 I [0]<e000e000:SCS v6-M class=14 designer=43b:Arm part=008> [rom_table]
0001158 I [1]<e0001000:DWT v6-M class=14 designer=43b:Arm part=00a> [rom_table]
0001158 I [2]<e0002000:BPV v6-M class=14 designer=43b:Arm part=00b> [rom_table]
0001174 I [1]<f0002000:MTB M0 class=9 designer=43b:Arm part=9a3 devtype=13 archid=0000 devid=0:0:0> [rom_table]
0001174 I CPU core #0: Cortex-M0 r0p0, v6.0-M architecture [cortex_m]
0001174 I 2 hardware watchpoints [dwt]
0001189 I 4 hardware breakpoints, 0 literal comparators [fpb]
0001205 I Semihost server started on port 4444 (core 0) [server]
0001632 I GDB server started on port 3333 (core 0) [gdbserver]
Remote deb002128 I Client connected to port 3333! [gdbserver]
ugging using :3333
arch_cpu_idle () at 0:/dev/zephyr/zephyrproject/zephyr/arch/arm/core/cortex_m/cpu_idle.S:139
139      cpsie i
0002237 I Attempting to load RTOS plugins [gdbserver]
Successfully halted device
Resetting target
Loading section rom_start, size 0xa8 lma 0x0
Loading section text, size 0x566c lma 0xa8
Loading section initlevel, size 0x50 lma 0x5714
Loading section device_area, size 0x8c lma 0x576c
Loading section sw_isr_table, size 0xd0 lma 0x57f8
Loading section rodata, size 0x2f8 lma 0x58d0
Loading section datas, size 0xc0 lma 0x5bc8
Loading section device_states, size 0xe lma 0x5c8c
Loading section k_timer_area, size 0x38 lma 0x5ca0
Loading section .last_section, size 0x4 lma 0x5cd8
[=====] 100%
0003427 I Erased 0 bytes (0 sectors), programmed 0 bytes (0 pages), skipped 24576 bytes (24 pages) at 23.45 kB/s [loader]
Start address 0x000007c4, load size 23758
Transfer rate: 22 KB/sec, 1131 bytes/write.
(gdb) |
```

Figure 12: The BBC micro:bit supports debugging with gdb out of the box, so no need for extra tools or anything.

To check that it works, execute a pristine build of your project, and upload the executable to the board.

## Debugging

If your board allows it (as the BBC micro:bit does) or you have a suitable debugging tool, you can debug the application with

`west debug`

This will start a gdb server and open a terminal (**Figure 12**). Consult the internet for learning how to work with gdb as that is outside the scope of this article.

## Zephyr Versus Arduino

Now that you have seen how to get started with the Zephyr OS, you may wonder why you would use it. Isn't it much easier to use Arduino? Like Zephyr, Arduino supports multiple processor architectures and hundreds of boards. Thousands of drivers and libraries have been written for Arduino. If an application or library uses the Arduino Core API, it can be ported easily to any other supported platform with similar peripherals, just like a Zephyr application. Both are open source. So?

Well, Zephyr is intended as an industry-grade, robust RTOS with features such as task scheduling, memory management, and device drivers. Furthermore, Zephyr is designed to accommodate a wide range of project complexities, from small IoT devices to complex embedded systems. It provides greater flexibility, but requires a more profound understanding of embedded development.

Arduino, while offering some real-time capabilities, is not an RTOS but a framework for single-threaded applications with a focus on simplicity and ease of use. Arduino abstracts many low-level details,

making it accessible to beginners. However, it can be used on top of an RTOS such as Mbed OS for more complex applications. Work to make the Arduino Core API run on Zephyr is in progress ([7]).

It is up to you to decide whether you need Zephyr for your next project or not. You could at least give it a try, as it looks great on any embedded developer's resume.

## Further Reading

That's it for now. As you will have seen, the Zephyr OS is complicated and has a somewhat steep learning curve. In this article, I have tried to make the ascension a bit easier. However, there is much, much more to say and learn about Zephyr, so don't think now that you know it all. References [8] and [9] provide links to two topics in which you might be interested. ◀

230713-01

## Questions or Comments?

Do you have technical questions or comments about his article? Email the author at [clemens.valens@elektor.com](mailto:clemens.valens@elektor.com) or contact Elektor at [editor@elektor.com](mailto:editor@elektor.com).



## About the Author

After a career in marine and industrial electronics, Clemens Valens started working for Elektor in 2008 as Editor-in-Chief of Elektor France. He has held different positions since and recently moved to the product development department. His main interests include signal processing and sound generation.



## Related Products

- > **BBC micro:bit V2**  
[www.elektor.com/19488](http://www.elektor.com/19488)
- > **Get Started with the NXP i.MX RT1010 Development Bundle**  
[www.elektor.com/20699](http://www.elektor.com/20699)
- > **Warren Gay, *FreeRTOS for ESP32-Arduino*, Elektor 2020**  
[www.elektor.com/19341](http://www.elektor.com/19341)



## Elektor Webinars

**MARCH  
28  
2024  
16:00 CET**

**Catch our Zephyr Webinar**  
for a hands-on IoT project introduction

Details at  
[elektormagazine.com/webinars](http://elektormagazine.com/webinars)

## WEB LINKS

- [1] About the Zephyr Project — A Linux Foundation project: <https://zephyrproject.org/learn-about>
- [2] Device Tree Specifications: <https://devicetree.org/specifications>
- [3] Getting started with Zephyr: [https://docs.zephyrproject.org/latest/develop/getting\\_started/index.html](https://docs.zephyrproject.org/latest/develop/getting_started/index.html)
- [4] Devicetree syntax high-lighting plugin for Visual Code Studio: <https://github.com/plorefice/vscode-devicetree>
- [5] Devicetree API: <https://docs.zephyrproject.org/latest/build/dts/api/api.html>
- [6] The LED peripheral: <https://docs.zephyrproject.org/latest/hardware/peripherals/led.html>
- [7] Running Arduino on Zephyr: <https://dhruvag2000.github.io/Blog-GSoC22>
- [8] See the Whitepaper by Eli Hughes: "From Hardware Concept to Zephyr Bring Up": <https://zephyrproject.org/white-papers>
- [9] USB Serial Console Output in a Zephyr RTOS Application: <https://www.gnd.io/zephyr-console-output>

# They trust us, do you?

We love electronics and projects, and we do our utmost to fulfil the needs of our customers.

The Elektor Store: 'Never Expensive, Always Surprising'

**UPDATED  
STORE**

Check out more reviews on our Trustpilot page: [www.elektor.com/TP](http://www.elektor.com/TP)  
Or make up your own mind by visiting our Elektor Store, [www.elektor.com](http://www.elektor.com)



**elektor**  
design > share > learn

# Award-Winning Ethics

A Dialog with CTO Alexander Gerfer of Würth Elektronik eiSos  
on Enabling Innovation and Mindful Behavior

Questions by Shenja Panik  
(Ethics in Electronics / Elektor)

Würth Elektronik eiSos CTO, Alexander Gerfer, shares his commitment to ethics and innovation in an exclusive interview with Shenja Panik. The interview discusses the Hightech Innovation Center's initiatives for environmental responsibility, fairness, and predictability, and highlights how ethics are integrated with business success at Würth Elektronik.

**Shenja Panik: Congratulations, Mr. Gerfer, on winning the 2023 Ethics in Electronics Award for your tireless commitment to sustainability, green tech, and vertical farming. Thank you for inviting us to your Hightech Innovation Center in Munich today and for taking the time to answer some questions. Can you please describe your role in the field of ethics in electronics?**

**Alexander Gerfer:** To explain, it's important to look at my background: growing up on a farm with cows and chickens, you have to take on responsibility very early on. Another crucial aspect is the German saying,

“Was du nicht willst, das man dir tut, das füg auch keinem andern zu” (meaning: Do as you would be done by). These are the basics of my upbringing. Our focus at Würth Elektronik is always on our customers. We do not discriminate against customers depending on the size of their company or their importance. We maintain a very cooperative relationship with each and every one. We let them benefit from our expertise through advice, sample designs, online presentations, development kits, and software tools that help with dimensioning. We don't charge any money for this or for laboratory samples of our components. Everyone is equally important to us. We support anyone who needs knowledge about selection and application. We supply and support everyone, from start-ups to established companies, from those ordering small quantities to buyers of large batches. We have no minimum order quantities. It's based on our company culture. As a start-up, you can fail during development because of a cent component. I know that from my own experience. That's why we always aim to solve problems quickly — and don't give preference to the one that promises maximum turnover.

**Shenja Panik: Not everyone gets the same chances as a start-up, so it's awesome that Würth Elektronik is actively working to level the playing field. What other initiatives or strategies being implemented in Munich to foster sustainable and ethical business?**

**Alexander Gerfer:** In our new Hightech Innovation Center in Munich, we test prototypes of electronic devices and assemblies for electromagnetic

compatibility and make targeted suggestions for improvement. After all, EMC certification is often a major hurdle for developers.

Fairness, predictability, equality, mutual support — these are the principles for management within the company and for how employees treat each other.

As part of the Würth Group, we also proactively contribute to the preservation of our environment, both through our products and environmentally friendly business practices. Nowadays, it is more important than ever to take care of the environment.

We always think a step ahead. Take Vertical Farming, for example: Cultivating food in multi-level greenhouses is already making a contribution to locally producing high-quality food. However, this contribution needs to be significantly expanded. According to estimates from the UN Food and Agriculture Organization (FAO), by 2050, 10 billion people will inhabit this planet. This necessitates a 50% increase in global agricultural production by the year 2050. Yet, agricultural land has decreased in recent decades, from nearly 40% of the world's land area in 1991 to only 37% in 2018. Plants will literally need to grow towards the sky if we want to ensure everyone is well-fed. Climate change and demographics pose enormous challenges, and we need to work on solutions now

— even if they are initially perceived as inefficient and “too expensive.”

There is no shortage of good ideas on how to make our world better. However, every significant idea today requires electronics for its implementation. Here, we contribute as a reliable supplier, but more importantly, as a helping hand for small and medium-sized businesses and ambitious startups. While we must generate profits as a company, at Würth Elektronik, the focus is not on short-term profit but on the meaningfulness of the project.

### **Shenja Panik: What does this award mean for you and your network?**

**Alexander Gerfer:** For me personally, the award is a very special honor. We also shared it with the team because I am not doing it alone. It is an honor for the whole team working on these projects — but above all, it is an incentive. It would probably be an overestimation to claim that the world has only improved in recent times. Unfortunately, signs of confrontation prevail in many areas, and regrettably, even the most fundamental ethical principles are being disregarded. It is even more important that we remain true to our principles, selflessly assist, act together, treat each other with respect, preserve our living space, and proactively drive innovations for the benefit of the public. We all face significant challenges. We have enormous problems to solve, and it is best to start addressing them together, right from today.

### **Shenja Panik: You told us about your background and the way it brought you to this point. What are now your primary sources of inspiration when making ethical decisions?**

**Alexander Gerfer:** Certainly, first and foremost, there is my very personal ethical and value compass that I inherited from home. I was instilled with a sense of responsibility towards nature and fellow human beings at a very young age. This is why, for example, the topic of Vertical Farming with LEDs is very close to my heart.

Important to us in the Würth Group are our corporate values, which determine our attitude and business conduct in all key areas. Mutual trust, predictability, honesty, and straightforwardness, both internally and externally, are fundamental principles that Prof. Dr. h. c. mult. Reinhold Würth formulated back in the 1970s.

The Code of Compliance of the Würth Group spans 32 pages, outlining general behavioral principles,

Figure 1: Handover of the Ethics in Electronics Award 2023 to winner Alexander Gerfer.





guidelines for dealing with business partners, information on avoiding conflicts of interest, and implementing the Code of Compliance.

Before making any decision, any of my colleagues or I ask: Is my decision or action in accordance with applicable laws? Does the decision align with the known values and rules of Würth? Assuming my colleagues and family were aware of my decision, would I have a clear conscience? If everyone were to decide this way, could I live with the consequences? If my decision would be in the newspaper tomorrow, could I justify it?

As I said, being fair in your decisions and being respectful and grateful for every contribution, but also very important is being predictable and reliable.

### **Shenja Panik: How much resilience is required to implement ethical and sustainable initiatives in a large company?**

**Alexander Gerfer:** In our daily business? Little to none, as we select our employees very carefully. Compliance, for us, means not only adhering to all applicable regulations, laws, and company guidelines but also a corresponding internal attitude of employees, which is a crucial component for the sustainable business success of Würth Elektronik.

Therefore, we expect our managers, employees, and business partners to uphold nationally and internationally valid laws and regulations. In addition, as part of our value catalog, we are committed to adhering to the standards, guidelines, and norms clearly defined in the Code of Compliance. This shared value compass passes to all areas of the company and is internalized by all employees.

During the selection process, we assess applicants not only based on their professional qualifications but especially on their personal attitudes.

Do they align with our principles? Do prospective employees identify with our corporate values, or have they merely memorized them? Are they more focused on sustainable success, or just quarterly figures? Do they view colleagues as partners or arrogantly? Do they think holistically and environmentally conscious, or egocentric and opportunistic?

These and many more questions must be answered before hiring. Knowledge can be acquired through on-the-job training, and skills can be learned from experienced colleagues. However, the indispensable foundation is ethical guidelines that everyone

must bring within the group. So, there is not much resilience — people working here believe in what we do and how we do it. And we experience daily that our employees handle responsibility very adeptly. Here, corporate leadership, executives, and employees pull together. Ethics is not enforced but lived within our organization.

### **Shenja Panik: It is not about going from project to project but maintaining a company philosophy. All of you are involved in the question of how to make the world a better place.**

**Alexander Gerfer:** It is not only our own people whom we empower, but we also enable individuals outside our company. Drawing from my own start-up experience and its challenges, I realized the importance of reaching out to developers in the early stages, particularly in the realm of hardware, which is one of the most challenging steps at the beginning. A wrong decision here can undermine even the best idea. Thus, we provide help and support. And we learn ourselves in the process... So, our overarching vision is a fully predictable understanding of customer needs. Ethics and business success are not contradictory for us; instead, they are inseparably linked. ◀

240013-01

### **Questions or Comments?**

Do you have questions or comments about this article? Email Shenja at [shenja.panik@elektor.com](mailto:shenja.panik@elektor.com).

### **About Shenja Panik**

Shenja Panik is a passionate sociologist writing for the content team of EiE, specializing in ethics within the electronics industry. Her work revolves around exploring and promoting ethical considerations in the ever-evolving landscape of technology.



### **About Alexander Gerfer**

Alexander Gerfer is the CTO of Würth Elektronik eiSos, a large European manufacturer of electronic components. He has been with the company for 27 years, holding various positions and promoting technology partnerships in areas such as LED light recipes, energy harvesting and e-mobility. With his expert knowledge, he is passionate about supporting and advising innovative start-ups. Gerfer is also a technical book author, lecturer and keynote speaker at global conferences.

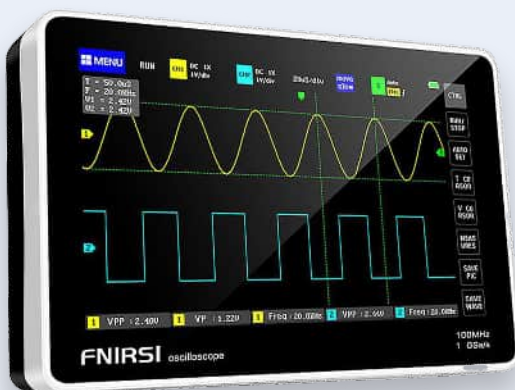
# The Elektor Store

## Never expensive, always surprising

The Elektor Store developed from the community store for Elektor's own products, such as books, magazines, kits and modules, into a mature web store that offers great value for surprising

electronics. We offer the products that we ourselves are enthusiastic about or that we simply want to try out. If you have a nice suggestion, we are here: [sale@elektor.com](mailto:sale@elektor.com).

## FNIRSI 1013D 2-ch Tablet Oscilloscope (100 MHz)

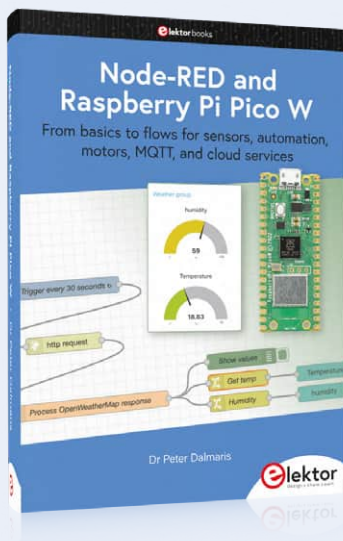


The FNIRSI 1013D is a fully featured 2-channel tablet oscilloscope with a high-resolution 7-inch LCD screen (800 x 480 pixels). The oscilloscope has a real-time sampling rate of 1 GSa/s and an analog bandwidth of 100 MHz.

Price: €159.95

**Member Price: €143.96**

[www.elektor.com/20644](http://www.elektor.com/20644)



## Node-RED and Raspberry Pi Pico W

This book is a learning guide and a reference. Use it to learn Node-RED, Raspberry Pi Pico W, and MicroPython, and add these state-of-the-art tools to your technology toolkit. It will introduce you to virtual machines, Docker, and MySQL in support of IoT projects based on Node-RED and the Raspberry Pi Pico W.

Price: ~~€44.95~~

**Member Price: €40.46**

[www.elektor.com/20745](http://www.elektor.com/20745)



## Oxocart Connect Innovator Kit



Price: €89.95

**Member Price: €80.96**

[www.elektor.com/20718](http://www.elektor.com/20718)

## SunFounder Kepler Kit (Ultimate Starter Kit for Raspberry Pi Pico W)



Price: €69.95

**Member Price: €62.96**

[www.elektor.com/20730](http://www.elektor.com/20730)

## Temperature-controlled Soldering Station ZD-931



Price: €49.95

**Member Price: €44.96**

[www.elektor.com/20623](http://www.elektor.com/20623)

## FNIRSI DSO152 Mini Oscilloscope (200 kHz)



Price: €39.95

**Member Price: €35.96**

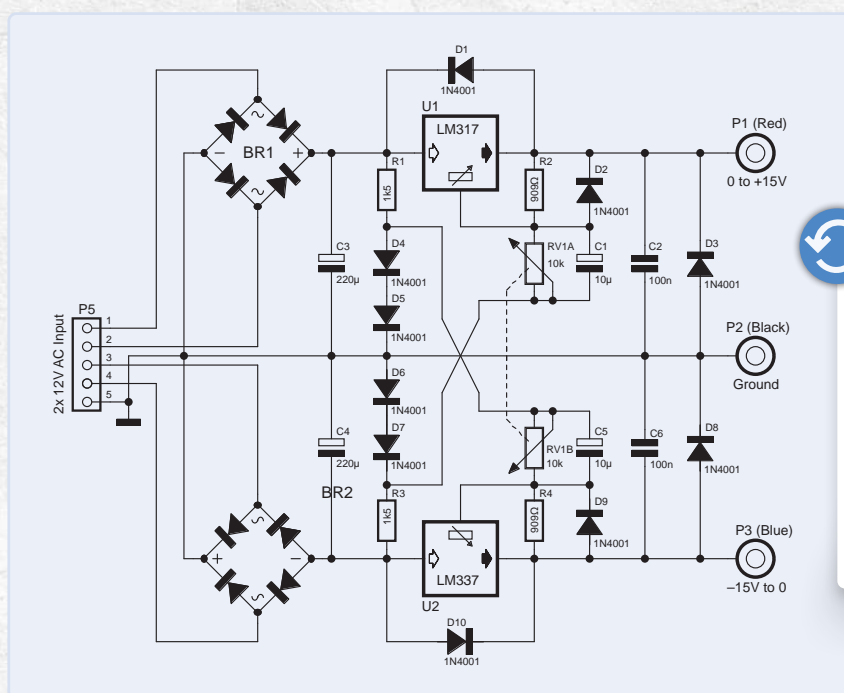
[www.elektor.com/20642](http://www.elektor.com/20642)



# Err-electronics

## Corrections, Updates, and Readers' Letters

Compiled by Jean-François Simon (Elektor)



### Variable Linear Power Supply Ensemble

Elektor 1-2/2024, p. 26 (220457)

There is an error in the schematic of the dual variable PSU (p. 29). The negative voltage regulator at the bottom of the drawing is an LM337, not an LM317. The text in the article correctly indicates LM337.



### USB Killer Detector

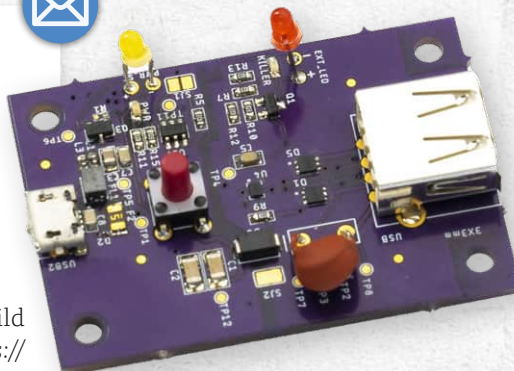
Elektor 11-12/2023, p. 32 (220549)

Is the USB Killer Detector available to buy as a finished device?

N. D. (Germany)

You can find all the files needed to build the USB Killer Detector on GitHub (<https://github.com/instantdevices/USB-Killer-Detector>). While I may be able to supply parts, I'm also seeking community assistance due to a persisting issue where a specific model of "killer" defeats the detector. So I'd say that the device is not fully market-ready yet. You can reach me at [instant.devices@yahoo.com](mailto:instant.devices@yahoo.com).

Carlos Guzman (Author of the article)



Got a bright idea or valuable feedback for Elektor? Reach out to us at [editor@elektor.com](mailto:editor@elektor.com) - we're eager to hear from you!



## Setting Up an SMT Line

**Elektor 11-12/2023, p. 108 (230593)**

In that article, you presented the Whizoo Controleo3 device. Can you give us a source of supply, possibly in Germany?

*Wolf-Peter Kleinau (Germany)*

That article was written by our partners at Opulo, a US-based company. Whizoo is also a US-based company and, unfortunately, as far as I know, there are no resellers in the EU. Also, Opulo's article implies that the "Controleo3" is a ready-made, off-the-shelf reflow solution. To be more precise, the Controleo3 was first known as a kit to build your own reflow oven, and it happens that Whizoo also sells a fully assembled oven for around €1200. Unfortunately, this product is only available in the US and Canada and comes with a 115-V oven. If you choose Controleo3 products, you can still buy them as a kit in the US, but you have to pay VAT and import duties separately, which can be a hassle, and you have to provide and assemble your own oven, so this is not "reflow ready."

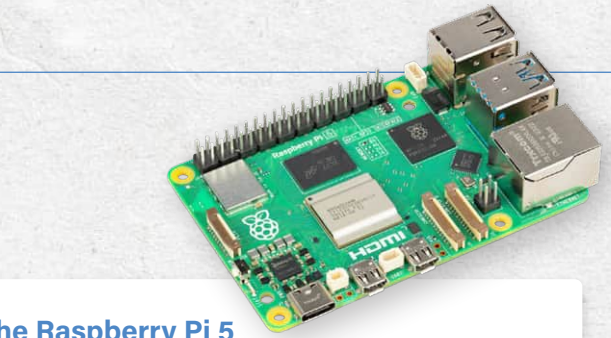
There is no simple answer to the question "Which reflow oven should I buy?" This is a broad area, and I would encourage you to do extensive research and decide what best suits your needs. Personally, I can't give any recommendations, but I have heard about the company Beta Layout. From other manufacturers, two popular options are the T962 oven (in many variations) and the ZB3530HL. Some people on the internet also found some ways to improve the T962 for a better heat distribution or better user interface.

*Jean-François Simon (Elektor)*

## MEMS Microphone

**Elektor 11-12/2023, p. 70 (230188)**

We made an error while redrawing the schematic (Figure 3, p. 72). Our apologies for that! The non-inverting inputs of op-amps U1 to U6 are connected to Vref through 47 kΩ resistors, not to ground (see the circuit diagram fragment on the right). The original schematic available in the Hackaday.io project linked to in the article is of course correct.



## The Raspberry Pi 5

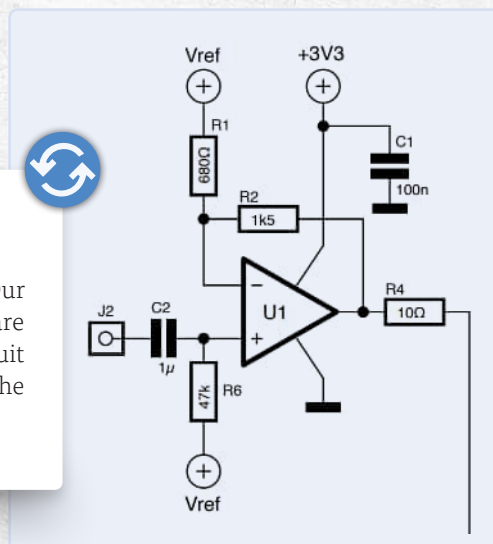
**Elektor 11-12/2023, p. 6 (230635)**

Many thanks for the informative article. On the various Raspberry Pi 4s that I have, I am used to copying SD cards to port everything that was laboriously installed (such as VS code, Qt 5, CAN bus, etc.) to different units easily. Can you tell me and the other readers how to port these to the Pi 5? I have inserted the copied SD card in the Pi 5, but it does not work. Thank you very much!

*Hanspeter Schären (Germany)*

Thank you for your email. This has to do with the version of the Raspberry Pi OS that is present on your SD cards that you used with the Pi 4. The Pi 5 requires the OS version *Bookworm* and cannot work with earlier versions. So, unfortunately, in your case you will have to create a new SD card from scratch, for example with the Raspberry Pi Imager, which is easy to use. And you will have to manually reinstall the software you need. Once you've done that, this SD card with *Bookworm* on it should in principle work with both the Pi 5 and the Pi 4 if you ever need to switch from one Raspberry to another, but you should do your own tests to see if everything works. Good luck!

*Jean-François Simon (Elektor)*



## Cold-Cathode Devices

**Elektor 1-2/2024, p. 74 (230373)**

I have heard that some cold cathode tubes are weakly radioactive. Is this true? For example, tubes from Elesta and others like the ER21A, GT21, 5823, or the GR16?

*Adolf Parth (Italy)*

You are right. Some of these tubes contain radioactive material to help ionize gas in them! You will find several websites on the Internet, made by radioactivity and Geiger counter enthusiasts, that give you more details about which specific tubes contain which radioactive source.

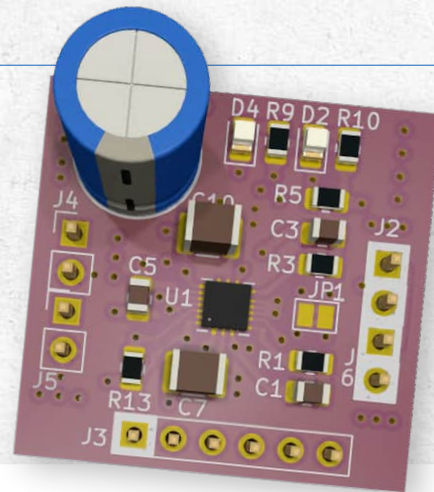
*Jean-François Simon (Elektor)*

The function of a cathode (in heated or cold-cathode tubes) is usually to emit electrons. In the case of heated cathodes, they emit electrons, which are usually modulated by a grid and collected by an anode. In cold-cathode tubes, they usually emit electrons to ionize a gas, either for surge protection, for display (neons and nixies) or for switching, which is the case with the cold-cathode thyatron tubes cited by Mr Parth.

For this reason, cathodes are very often coated with substances to aid the emission of electrons. In the case of standard heated cathode tubes, Thorium is commonly used. In cold-cathode tubes, a number of rare earth metals and compounds are used. Some of these exhibit slight radioactivity. In the periodic table, rare earth metals are in the Lanthanide series, and Thorium is in the related Actinide series. A few of the Lanthanides are radioactive, and most of the Actinides are. Thorium is also used in the mantles of gas lamps, and these will be found to be slightly radioactive. Many common elements have a proportion of isotopes which are radioactive, Potassium being a well-known one.

*David Ashton (Author of the article)*

Source: [https://en.wikipedia.org/wiki/Nixie\\_tube#/media/File:ZM1210-operating\\_edit2.jpg](https://en.wikipedia.org/wiki/Nixie_tube#/media/File:ZM1210-operating_edit2.jpg)



## Motor Driver Breakout Board

**Elektor 9-10/2023, p. 56 (210657)**

Hello everyone! Is it possible to create a temperature-dependent speed control for a fan with the motor driver MP6619, which outputs then from 3 V DC at 20°C to 12 V DC at 40°C for example, using an NTC 10-kΩ sensor? Thank you very much for any information or circuit examples. Or perhaps there is even another possibility to build such a speed control?

*Matthias Seesemann (Germany)*

It is possible to do what you describe with an MP6619, but you also need other components. The MP6619 alone is just a H-bridge, so it turns its output transistors On or Off depending on the state of the IN1, IN2 and ENABLE input pins. To have an output voltage of 3 V to 12 V, you would need to use PWM to control those input pins of the MP6619. The EN pin should be pulled high with a pull-up resistor, then you can pull IN2 high or low depending on the direction of rotation that you need, and at the same time, you send a PWM signal to IN1 to vary the speed of the motor. To generate the PWM signal, you can use any microcontroller, Arduino or other. I recommend you use an Arduino Uno R3. That way, you will find plenty of code examples online. That will show how to read a temperature with an NTC, and also how to use PWM functions on the Arduino. Also, you may be interested in looking at some online projects using the "Fan speed controller" TC648: I think it does just what you need. You may need to change one or two resistor values to make it behave exactly as you want. You should post your project to our Elektor Labs platform ([www.elektormagazine.com/labs](http://www.elektormagazine.com/labs)), so other members of the Elektor community can comment and help you. Have fun!

*Jean-François Simon (Elektor)*



# PROTEUS DESIGN SUITE



**Driving forward with Manual Routing**

Push and Shove Routing  
for dense layouts

Dedicated Differential  
Pairs Routing mode

Length Matching and  
Net Tuning Support

Visual DRC shows legal  
paths for route placement



FASTER  
ROUTING  
AVAILABLE



**Labcenter**  
Electronics

[www.labcenter.com](http://www.labcenter.com)

[info@labcenter.com](mailto:info@labcenter.com)







# Build on your proficiency

Quick tips, tools and articles  
for purchasing professionals

[mou.sr/purchasing-resources](https://mouser.com/purchasing-resources)

